Írta:
**BERTÓK BOTOND**
**KOVÁCS ZOLTÁN**

# GYÁRTÓRENDSZEREK MODELLEZÉSE

**Egyetemi tananyag**

TYPOTEX

**2011**

LEKTORÁLTA: Nyakasné Dr Tátrai Judit, Széchenyi István Egyetem Logisztikai és Szállítmányozási Tanszék



TÁMOGATÁS:
Készült a TÁMOP-4.1.2-08/1/A-2009-0008 számú, „Tananyagfejlesztés mérnök informatikus, programtervező informatikus és gazdaságinformatikus képzésekhez” című projekt keretében.



ISBN 978-963-279-491-4

KÉSZÜLT: a Typotex Kiadó gondozásában
FELELŐS VEZETŐ: Votisky Zsuzsa
AZ ELEKTRONIKUS KIADÁST ELŐKÉSZÍTETTE: Juhász Lehel

**KULCSSZAVAK:**
gráf, hálózat, ellátási lánc, matematikai programozás, folyamatszintézis, ütemezés.

**ÖSSZEFOGLALÁS:**
A tárgy és a jegyzet célja bevezetést adni olyan modellezési eljárásokba, melyek lehetővé teszik ipari gyártórendszerek és kapcsolódó hálózatok analízisét és optimalizálását. A gyártórendszerekhez kapcsolódó hálózat lehet az energiaellátó rendszer, a gyártáshoz kapcsolódó logisztika, az informatikai rendszer, de akár a teljes ellátási lánc is. Mindezen rendszereket és hálózatokat a következőkben elsősorban gráfokkal és hálózati folyamatokkal vagy ezek valamilyen formális matematikai leírásával modellezzük, melyek szoftverekkel jól kezelhetőek. A jegyzet második fejezete átismétli a halmazelmélet alapjait, a relációk és függvények, gráfok és hálózatok fogalmát. A későbbi fejezetekben ilyen modellek készítésével foglalkozunk, aszerint csoportosítva őket, hogy milyen algoritmusokkal vagy szoftverekkel vizsgálhatóak. A modelleket gráf algoritmusokkal, matematikai programozással és folyamathálózat szintézis eljárásokkal elemezzük és optimalizáljuk. Ezen elemző és optimalizáló módszerek mindegyikét egy-egy fejezetben tárgyaljuk.

# Tartalomjegyzék


**1. Bevezetés** **5**

**2. Modellezési alapismeretek** **7**
2.1. Eligazodás modellekben . . . . . 7
2.1.1. Térkép . . . . . 7
2.1.2. Lépték . . . . . 8
2.1.3. Iránytű . . . . . 9
2.2. Formális modellezés alapjai . . . . . 10
2.2.1. Halmazok . . . . . 10
2.2.2. Relációk és függvények . . . . . 13
2.2.3. Gráfok . . . . . 15
2.3. Feladatok . . . . . 18

**3. Gráf algoritmusok** **19**
3.1. Minimális feszítőfa meghatározása . . . . . 19
3.2. Legrövidebb út meghatározása . . . . . 22
3.3. Gráf bejárások . . . . . 27
3.3.1. Szélességi bejárás . . . . . 27
3.3.2. Mélységi bejárás . . . . . 30
3.4. Maximális folyam . . . . . 34
3.5. Feladatok . . . . . 41

**4. Lineáris programozás** **44**
4.1. Modellezés és formalizálás . . . . . 44
4.2. Feladatmegoldás glpk-val . . . . . 47
4.3. Egészértékű változók . . . . . 54
4.4. Nevezetes feladatok megoldása MILP-el . . . . . 56
4.4.1. Legrövidebb út . . . . . 57
4.4.2. Minimális feszítőfa . . . . . 60
4.4.3. Maximális folyam . . . . . 63
4.5. Feladatok . . . . . 65

**5. Folyamathálózat-szintézis és optimalizálás** **68**
5.1. A P-gráf módszertan alapjai . . . . . 71
5.1.1. Alapfogalmak . . . . . 72

5.1.2. P-gráf . . . 73
5.1.3. Kombinatorikusan lehetséges megoldásstruktúrák . . . 75
5.1.4. Axiómák . . . 76
5.1.5. Maximális struktúra . . . 78
5.1.6. Az MSG algoritmus . . . 79
5.1.7. Az SSG algoritmus . . . 81
5.1.8. A vegyes-egész matematikai programozási modell . . . 83
5.1.9. Az ABB algoritmus . . . 86
5.2. Nevezetes feladatok megoldása folyamatszintézissel . . . 86
5.2.1. Minimális feszítőfa szintézise . . . 86
5.2.2. Legrövidebb út szintézise . . . 88
5.2.3. Maximális folyam szintézise . . . 89
5.3. Demonstrációs szoftverek . . . 90
5.3.1. PNS Draw . . . 90
5.3.2. PNS Studio . . . 92
5.4. Feladatok . . . 94

**6. Ütemezés** **96**
6.1. Bevezetés . . . 96
6.2. Flow shop, job shop, open shop . . . 97
6.3. Ütemezési feladat általános megfogalmazása . . . 100
6.4. S-gráf leírás . . . 103
6.4.1. Recept-gráf . . . 105
6.4.2. Ütemezési-gráf . . . 106
6.5. Alapalgoritmus ütemezésre S-gráffal . . . 111
6.5.1. Szétválasztási lépés . . . 112
6.5.2. Az algoritmus korlátozási lépése . . . 112
6.5.3. Az algoritmus működésének szemléltetése . . . 113
6.6. S-gráf módszertan kiterjesztései . . . 115
6.6.1. Taszk alapú döntési stratégia . . . 115
6.6.2. Profit maximalizálása . . . 116
6.7. S-Graph Studio . . . 119
6.7.1. Feladat megjelenítése és készítése . . . 120
6.7.2. Feladat megoldás . . . 122
6.7.3. Az eredmény megjelenítése . . . 122
6.8. Feladatok . . . 123

**7. Tévedések és kockázatok** **126**
7.1. Strukturális leírás . . . 126
7.2. Megoldások strukturális tulajdonságai . . . 127
7.3. Ismeretlen információ szükséges a megoldáshoz . . . 127
7.4. Nem megvalósítható megoldás . . . 127

**Irodalomjegyzék** **129**

# 1. fejezet

# Bevezetés

A tárgy és a jegyzet célja bevezetést adni olyan modellezési eljárásokba, melyek lehetővé teszik ipari gyártórendszerek és kapcsolódó hálózatok analízisét és optimalizálását. Az elemzések célja a rendszerekre vonatkozó kérdések megválaszolása mint például „Adott erőforrás kapacitás mellett, mekkora lehet egy termékből a maximális kihozatal?" vagy „Egy adott számítógép hálózati topológia mekkora maximális sávszélességet képes biztosítani két csatlakozási pont között?" Az optimalizálás célja pedig a rendszer valamely szempont szerinti legjobb állapotának meghatározása, ami lehet például a profit maximalizálása, vagy a környezeti terhelés csökkentése. A gyártórendszerekhez kapcsolódó hálózat lehet az energia ellátó rendszer, a gyártáshoz kapcsolódó logisztika, az informatikai rendszer, de akár a teljes ellátási lánc is.

Az elmúlt tíz év során ipari megbízások és kutatás-fejlesztési projektek kapcsán végzett munkánk számos szép példát szolgáltat a jegyzet számára. Az optimalizálási projektek eredményeként elért jelentős megtakarítások pedig motivációt adhatnak az olvasónak, hogy a jegyzet megismerésével olyan ismeretek elsajátításába kezdjen, melyek hasonló sikerek elérésére teszik őt képessé későbbi pályája során.

A modellezés előfeltétele annak, hogy a rendszerek formális eszközökkel vagy azok szoftver implementációival hatékonyan vizsgálhatóak legyenek. Mindezen rendszereket és hálózatokat ezért a következőkben elsősorban gráfokkal és hálózati folyamatokkal vagy ezek valamilyen formális matematikai leírásával modellezzük, melyek szoftverekkel jól kezelhetőek. A matematikai leírások halmazokat, változókat, valamint ezeken értelmezett összefüggéseket, feltételeket tartalmaznak. A jegyzet következő fejezete átismétli a halmazelmélet alapjait, a relációk és függvények, gráfok és hálózatok fogalmát. A későbbi fejezetekben ilyen modellek készítésével foglalkozunk, aszerint csoportosítva őket, hogy milyen algoritmusokkal vagy szoftverekkel vizsgálhatóak. A modellek elkészítéséhez egyrészt sémákat mutatunk, másrészt modellgeneráló szoftvereket ismertetünk.

A modelleket gráf algoritmusokkal, matematikai programozással és folyamathálózat szintézis eljárásokkal elemezzük és optimalizáljuk. Ezen elemző és optimalizáló módszerek mindegyikét egy-egy fejezetben tárgyaljuk. Megismerjük ezen módszerek fontosabb lépéseit és a velük kezelhető feladatok körét. A fejezetek tartalmaznak olyan egyszerű példákat, melyek számítógépes támogatás nélkül is könnyen megoldhatóak, de emellett hivatkozásokat Internetről letölthető szoftverekre is, melyek nagyméretű, valós ipari feladatok kezelését is lehetővé teszik. A különböző módszerek gyakran alkalmasak nagyon hasonló vagy akár azonos

feladatok megoldására, erre mindig felhívjuk a figyelmet. Tesszük ezt annak érdekében, hogy az olvasó kellő tapasztalatot szerezzen ahhoz, hogy a későbbiekben a gyakorlatban felmerülő kérdések nehézségét meg tudja becsülni, és a megoldáshoz megfelelő eszközt ki tudja választani.

Bízunk benne, hogy a jegyzet hasznos olvasmány lesz mindazok számára, akik pályájuk bizonyos szakaszában ipari gyártórendszerek tervezésével vagy működtetésével foglalkoznak, legyenek ők mérnökök, közgazdászok vagy informatikusok. Az egyes témakörök tárgyalását igyekszünk mindvégig gyakorlati példákkal színesíteni és közérthető szinten tartani bárki számára, aki az alapvető matematikai fogalmakat és a szövegértés képességét bírja.

# 2. fejezet

# Modellezési alapismeretek

Modellezés esetén két kérdés megkerülhetetlen: mit modellezünk, és mivel modellezünk. Gyártórendszerek számos szegmense modellezhető többféle szempont szerint. A modellezés eszköze pedig esetünkben matematikai modelleket jelent: például halmazokat, gráfokat.

## 2.1. Eligazodás modellekben

Ebben a fejezetben áttekintést adunk a gyártórendszerekről három szempont szerint. Először ismertetjük a tipikus rendszerek főbb komponenseit és azok kapcsolatait (térkép). Ezután bemutatjuk a kapcsolódó modellek csoportosítását részletességük szerint. Végül bemutatjuk azokat az informatikai rendszereket, melyek a gyártórendszerek tervezésében és üzemeltetésében, segítségünkre lehetnek.

### 2.1.1. Térkép

A globalizáció eredményeként a gazdasági szereplők általában nem lokálisan, nem szűk környezetükben versenyeznek egymással, versenyképességüket az határozza meg, hogy milyen ellátási láncban szerepelnek. Ami jellegéből adódóan nem vagy csak erős kompromisszumok árán szállítható, az helyi versenyt eredményezhet. Ilyenek a gyorsan romló élelmiszerek vagy olyan szolgáltatások, melyek fizikai kapcsolat igényelnek. Jellemző példái az éttermek, cukrászdák, színházak, turisztikai látványosságok, vagy az egészségügyi szolgáltatások. Minden olyan termék, mely reálisan szállítható nagy távolságokra értékvesztés nélkül, az bárhol elkészíthető. Ilyenek például a műszaki és háztartási cikkek, ruházati termékek és a tartós élelmiszerek.

Egy ellátási lánc tartalmazza a gyártáshoz szükséges erőforrások beszerzését és a termékek életpályáját a vásárlókig, sőt néha a karbantartásig vagy hulladékhasznosításig is, ha ezen lépésekben a gyártónak szerepe van. Mindezen lépések költsége hatással van a termék árára, így bármelyiken múlhat a versenyképesség. A gyártást éppen ezért szinte értelmetlen önmagában vizsgálni. Sokszor a logisztikai rendszer optimalizálásával több megtakarítást lehet elérni, mint a gyártási technológia finomhangolásával.

Az erőforrások egyik része az energiaellátás: például elektromos energia, gáz, olaj, távhő, ipari víz. Másik része a nyersanyagok, melyek lehetnek feldolgozatlan vagy kevéssé feldol-

gozott ásványkincsek és mezőgazdasági termékek. Harmadik a munkaerő, mely biztosítása nagy létszám esetén szintén komoly logisztikai feladatot jelent.

Mind az energiahordozókat, mind a nyersanyagokat, mind a munkaerőt, mind a termékeket tipikusan szállítani kell. A szállítás módja az elektromos távvezetéktől, a csőhálózaton át, a különböző vízi, légi és szárazföldi szállítóeszközökig terjed. A szállítás bizonytalansága, időigénye vagy a nagyobb méretben fajlagosan kedvezőbb költsége miatt gyakran érdemes tárolni, tartalékot képezni. Tartalék lehet egy raktár, egy szünetmentes tápegység, de a készenléti állományban tartott személyzet is.

Ami a nagy távolságokat átölelő ellátási láncok létjogosultságát megalapozza és ezért a logisztikára a korábbinál nagyobb hangsúlyt helyez, az a gyártási költségek jelentős eltérése földrajzi helyenként. Ez a költség adódhat a nyersanyagok vagy a termékeket felvevő piac földrajzi eloszlásából, a helyi munkaerő árából, de a gazdasági környezetből is. Gazdasági környezet alatt értjük például a helyi adórendszert, támogatási rendszert és a bevonható helyi alvállalkozók körét is.

Mindezekből kitűnik, hogy gyártórendszert nem lehet modellezni a környezetének ismerete nélkül. A fenti tényezők nem csak azért érdekesek, mert támpontot adnak arra nézve, hogy milyen paramétereket érdemes összegyűjteni, hanem leginkább azért, hogy elgondolkodjunk rajta, hogy valóban az Achilles-sarkánál fogtuk-e meg a feladatot.

### 2.1.2. Lépték

Egy gyártórendszer modelljének felépítéséhez nem csak azzal kell tisztában lennünk, hogy egy összetett rendszer elemi közül mely elemeket kívánjuk figyelembe venni, hanem azzal is, hogy milyen részletességgel. Részletesség alapján háromféle kategóriába sorolhatjuk a modelleket. A három kategóriába a makroszkopi-kus, a mezoszkopikus és a mikroszkopikus léptékű modellek tartoznak.

A makroszkopikus a legmagasabb absztrakciós szint, itt a legnagyobb a modell által átfogott terület, ugyanakkor ez a legkevésbé részletes. Ilyenkor „madártávlatból” tekintjük a gyártórendszert. Az ellátó lánc makroszkopikus modelljének elemei a beszerzési, gyártó, értékesítő és közbülső logisztikai rendszerek, nem részletezve azok belső folyamatait. A gyár makroszkopikus modelljének elemei általában az egyes üzemek vagy technológiai lépések, nem törődve azok belső felépítésével. A felsorolt modellelemeket fekete doboznak tekintjük, csak a be- és kimenetei, külső kapcsolatai alapján írjuk le őket mintha nem is tudnánk belelátni a belsejükbe. A makroszkopikus modellezés a koncepcionális tervezés eszköze. A koncepcionális tervezés csak alapvető kérdésekre keres választ, mint hogy mit hol érdemes gyártani, mely technológiába érdemes beruházni, és mekkora a várható megtérülési idő.

A mikroszkopikus modell a legkisebb absztrakciós szint. Egy mikroszkopikus modell a gyártórendszernek általában csak kis részletét emeli ki, de azt nagyon részletesen írja le. Ilyen modellek adják meg például egy berendezés működését, a benne lejátszódó fizikai, kémiai vagy biológiai átalakulásokat. Általában új technológiák vagy berendezések tervezéséhez használjuk.

A mezoszkopikus szint léptéke a makró és a mikró közé esik. Itt található a technológia lépések kapcsolatainak részletezése, vagy a koncepcionális modell bizonyos szempontú kifejtése. Tipikusan mezo modellek írják le például vegyipari rendszerekben a technológiai fo-

lyamatok során keletkező termékek tisztítását végző szétválasztási hálózatokat, vagy a hűtési és fűtési igényeket biztosító hőcserélő hálózatokat. Ezek a rendszerek nem a termék közvetlen előállítását végzik, ugyanakkor költségük olyan mértékű lehet, mely eldöntheti egy gyár életképességét. Éppen ezért a mezoszkopikus modellezést a részletes költségszámításokhoz használjuk.

### 2.1.3. Iránytű

A modellek elemzése mellett gyakorlatban legalább olyan fontosak azok az eszközök, melyek a valós folyamatokról adnak visszajelzéseket. A reális átfogó profit mellett egy ellátási lánc életképes működéséhez arra is szükség van, hogy a változásokra gyorsan és hatékonyan tudjon válaszokat adni. Ez pedig számos informatikai és kommunikációs megoldást igényel. A változás, amire reagálni kell lehet külső körülmények változása, de belső folyamatok nem megfelelősége is. Hogyan tudjuk ellenőrizni, hogy a folyamatok tervszerűen zajlanak-e?

Az ellenőrzéshez követőrendszerek szükségesek. Folyamat követésre alkalmas információt szolgáltathat a gyártási folyamatban egy mérőrendszer, az ellátási láncban egy átfogó vállalatirányítási rendszer, a logisztikában pedig a műholdas járműkövetés és a logisztikai azonosítók alkalmazása. Egy vegyipari üzemben az áramlásmérők jelzik a be- és kimenő anyagok mennyiségét. A vállalatirányítási rendszerben rögzített átadás-átvételi dokumentumok igazolják az üzleti folyamat előrehaladását. A műholdas követőrendszerek segítségével, folyamatosan ellenőrizhető a járművek helye és haladási iránya. A logisztikai azonosítók, mint a vonalkód vagy a rádiófrekvenciás azonosító (RFID) akár a termékek és szállítóeszközök minden egyes példányának automatikus detektálását is lehetővé teszik a folyamat kritikus pontjain. A pillanatnyi történések ismerete azonban csak akkor hasznos, ha tudjuk, hogy minek kellene történnie, és a terv és a tény automatikusan is összehasonlítható. Az összehasonlíthatóságnak feltétele, hogy a tervek is olyan részletességgel álljanak elő, mint a mért adatok. Ehhez pedig számítógéppel célszerű tervezni, a tervek időbeli lezajlását pedig szimulációval leszámlálni.

Ha a folyamatok tervezését módszeresen végezzük, akkor lépten-nyomon nagyszámú megvalósítási alternatíva tárul elénk, melyek közül ki kell választanunk azt az egyet, amit a gyakorlatban realizálunk. Ha eközben módszeresen törekszünk arra, hogy az alternatívák közül a lehető legjobbat válasszuk, akkor optimalizálásról beszélünk. Ebben a jegyzetben számos példát látunk majd erre. Ráadásul olyan eszközöket mutatunk be, melyek helyes feladatmegfogalmazás mellett garantáltan meg is találják a legjobb megoldást. A szimuláció jó esetben már nem igényel további döntéseket, azon felül, melyek a tervezéskor megszülettek. Ugyanakkor, éppen a bizonytalanságok feloldása miatt, több információval dolgozva, sokkal pontosabb számításokat tud végrehajtani. Ezen számítások egy része a tervezői döntésekhez felesleges is lenne, ugyanakkor a tervek megvalósításához már jól hasznosítható. Például egy logisztikai terv összeállításakor csak az érdekel minket, hogy egy adott tevékenység elvégzéséhez van-e elég kapacitás. A megvalósításhoz ugyanakkor a feladatokat konkrét szereplőkhöz kell rendelni. Szerencsére ezen hozzárendelések is nagyrészt automatikusan elvégezhetőek, még ha egyéni szempontok szerint a felelős döntéshozó módosít is rajtuk. Ha a módosítás szintén bekerül az informatikai rendszerbe, akkor folyamat automatikusan követhető és felügyelhető lesz. Felügyeleti rendszernek hívjuk azt a komponenst, mely a szimuláció és a követés eredményeit összeveti és a kritikus eltéréseket kiválogatja.

Egy informatikával megfelelően támogatott ellátási láncban a döntéshozóknak csak a tervek jóváhagyásával, egyedi igények szerinti módosításával és a kritikus kivételek kezelésével kell foglalkozniuk. Egy rosszul támogatott rendszerben pedig azzal, hogy a megfelelő munkatársakat alkalmazza, akik képesek a folyamatos kommunikációra, a nagyszámú esemény fejben követésre és ellenőrzésre, valamint a bekövetkező hibákat gyorsan jelzik ahelyett hogy elfednék.

A jegyzetben bemutatására kerülő modellek nem csak a tervezésben és optimalizálásban, de a szimuláció során is megjelenhetnek. Így nem csak elméleti vizsgálatokra alkalmasak, hanem a mindennapi feladatok hasznos segítői is lehetnek.

## 2.2. Formális modellezés alapjai

Gyakorlati feladatok számítógépes elemzéshez formális leírások szükségesek. A módszerek helyességének bizonyításához pedig matematikai alapok. A következőkben a gyártórendszerek formális modelljeinek legegyszerűbb építőköveit, a halmazokat, függvényeket és a gráfokat tekintjük át.

### 2.2.1. Halmazok

Egy **halmaz** alatt valamilyen objektumok összességét, sokaságát értjük. Pl. az $a$, $b$, $c$, $d$ betűk összessége egy halmaz, amelyet $L$ betűvel jelölve a következőképpen adhatunk meg: $L = \{a, b, c, d\}$. A halmazt alkotó objektumokat **elemeknek** vagy **tagoknak** nevezzük. A $b$ betű például az $L$ halmaz egy eleme, amelynek jelölése $b \in L$. Néha egyszerűen csak annyit mondunk, hogy $b$ $L$-ben van, vagy azt, hogy $L$ tartalmazza $b$-t. A $z$ betű ellenben nem eleme az $L$ halmaznak, amelynek jelölése $z \notin L$.

Egy halmazban minden elem csak egyszer van felsorolva, így a {*piros*, *fekete*, *piros*} halmaz megegyezik a {*piros*, *fekete*} halmazzal. Az elemek sorrendje sem lényeges, így $\{3,1,9\}$, $\{9,3,1\}$ és$\{1,3,9\}$ ugyanazt a halmazt jelenti. Összegezve: két halmaz egyenlő akkor és csak akkor, ha azonosak az elemeik.

Egy halmaz elemei között semmilyen összefüggésnek nem kell fennállnia (azon kívül, hogy ugyanannak a halmaznak az elemei), például a {3, *piros*, {*fekete*, *d*}} egy háromelemű halmaz, amelynek egy eleme önmagában is egy halmaz. Előfordulhat, hogy egy halmaznak csak egyetlen eleme van; ebben az esetben **szingleton**ról beszélünk. Például {1} egy halmaz, amelynek az egyetlen eleme 1, ezért {1} nem azonos 1-el. Azt a halmazt, amelynek nincs eleme, **üres halmaz**nak hívjuk. Jelölése: $\emptyset$. Természetesen egyetlen üres halmaz van, minden más halmazra azt mondjuk, hogy nemüres.

Egy $A$ halmaz elemeinek számát a számosságának nevezzük és $|A|$-val jelöljük. Például: $|\emptyset| = 0$, $|\{3,1,9\}| = 3$ és $|\{piros, fekete\}| = 2$.

Mindezidáig a halmazokat úgy adtuk meg, hogy elemeiket egymástól vesszővel elválasztva, kapcsos zárójelek között soroltuk fel. Ez a módszer végtelen halmazok megadására nem alkalmas. A természetes számok halmaza, $\mathbf{N}$, például egy végtelen halmaz , amelynek elemeit megpróbálhatjuk megadni $\mathbf{N} = \{1,2,3, \ldots\}$ módon, a három pont helyére intuitívan végtelen sok elemet odaképzelve. Amely halmaz nem végtelen, az véges.

Lehetőség van halmazok megadására más halmazok, illetve elemeik tulajdonságainak segítségével is. Így például ha $I = \{1,3,9\}$ és $G = \{3,9\}$, akkor $G$ megadható úgy, mint $I$ kettőnél nagyobb elemeinek halmaza. Ezt az alábbi módon írhatjuk:

$$G = \{x : x \in I \text{ és } x \text{ nagyobb mint } 2\}. \tag{2.1}$$

Általánosságban, ha az $A$ halmazt definiáltuk és $P$ egy tulajdonság, amely $A$ elemeire teljesül vagy nem teljesül, akkor egy új halmazt a következőképpen adhatunk meg:

$$B = \{x : x \in A \text{ és } x\text{-re teljesül a } P \text{ tulajdonság}\}. \tag{2.2}$$

Például a páratlan természetes számok halmaza a következőképpen adható meg:

$$O = \{x : x \in \mathbf{N} \text{ és } x \text{ nem osztható 2-vel}\}. \tag{2.3}$$

Azt mondjuk, hogy az $A$ halmaz **részhalmaza** a $B$ halmaznak, ha $A$ minden eleme $B$-nek is eleme. Jelölése: $A \subseteq B$. Így $O \subseteq \mathbf{N}$, mivel minden páratlan természetes szám egyben természetes szám is. Minden halmaz egyben önmaga részhalmaza is. Ha $A$ részhalmaza $B$-nek, de $A$ nem egyenlő $B$-vel, akkor azt mondjuk, hogy $A$ **valódi részhalmaz**a $B$-nek, és ezt a következőképpen jelöljük: $A \subset B$. Az üres halmaz minden halmaznak részhalmaza, azaz tetszőleges $B$ halmazra $\emptyset \subseteq B$, mivel $\emptyset$ minden eleme $B$-nek is eleme. Annak bizonyítására, hogy két halmaz $A$ és $B$ egyenlő, egy lehetséges módszer, ha belátjuk, hogy $A \subseteq B$ és $B \subseteq A$. Mivel $A$ minden elemét tartalmaznia kell $B$-nek és fordítva is, nem nehéz belátni, hogy $A = B$.

A már meglévő halmazainkból különböző *halmazműveletek* segítségével újabb halmazokat hozhatunk létre. Ilyen halmazművelet az **unió** is. Két halmaz uniójaként kapott halmaz elemei azok az elemek lesznek, amelyek legalább az egyik, vagy mindkét halmaznak elemei voltak. Az uniót az $\cup$ szimbólummal jelöljük.

$$A \cup B = \{x : x \in A \text{ vagy } x \in B\}. \tag{2.4}$$

Például:

$$\{1,3,9\} \cup \{3,5,7\} = \{1,3,5,7,9\}. \tag{2.5}$$

Két halmaz **metszet**e alatt a két halmaz közös elemeinek összességét értjük, azaz

$$A \cap B = \{x : x \in A \text{ és } x \in B\}. \tag{2.6}$$

Például:

$$\{1,3,9\} \cap \{3,5,7\} = \{3\}, \tag{2.7}$$

és

$$\{1,3,9\} \cap \{a, b, c, d\} = \emptyset. \tag{2.8}$$

Végül két halmaz, $A$ és $B$ különbségén azoknak az elemeknek a halmazát értjük, amelyek $A$-nak elemei, de $B$-nek nem. Jelölése: $A \setminus B$.

$$A \setminus B = \{x : x \in A \text{ és } x \notin B\}. \tag{2.9}$$

Például:

$$\{1,3,9\} \setminus \{3,5,7\} = \{1,9\}. \tag{2.10}$$

A halmazműveletek bizonyos tulajdonságai egyértelműen következnek a definícióikból. Például ha $A$, $B$ és $C$ halmazok, akkor a következő állítások teljesülnek:
Idempotencia:

$$A \cup A = A \tag{2.11}$$

Kommutativitás:

$$A \cup B = B \cup A \tag{2.12}$$

$$A \cap B = B \cap A \tag{2.13}$$

Asszociativitás:

$$(A \cup B) \cup C = A \cup (B \cup C) \tag{2.14}$$

$$(A \cap B) \cap C = A \cap (B \cap C) \tag{2.15}$$

Disztributivitás:

$$(A \cup B) \cap C = (A \cap C) \cup (B \cap C) \tag{2.16}$$

$$(A \cap B) \cup C = (A \cup C) \cap (B \cup C) \tag{2.17}$$

Abszorpció:

$$(A \cup B) \cap A = A \tag{2.18}$$

$$(A \cap B) \cup A = A \tag{2.19}$$

de Morgan azonosságok:

$$A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C) \tag{2.20}$$

$$A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C) \tag{2.21}$$

Két halmaz **diszjunkt**, ha nincs közös elemük, azaz a metszetük üres.

A metszet és unió kettőnél több halmazra is definiálható. Jelölje $S$ tetszőleges halmazok összességét. Ekkor $\bigcup S$ jelöli azt a halmazt, amelynek elemeit úgy kapjuk, ha az összes $S$-beli halmaz elemeit uniózzuk. Például ha $S = \{\{a, b\}, \{b, c\}, \{c, d\}\}$ akkor $\bigcup S = \{a, b, c, d\}$, illetve ha $S = \{\{n\} : n \in \mathbf{N}\}$, azaz $S$ az olyan egyelemű halmazok halmaza, amely halmazok elemei a természetes számok, akkor $\bigcup S = \mathbf{N}$. Általánosságban

$$\bigcup S = \{x : x \in P \text{ tetszőleges } P \in S \text{ halmazra }\}. \tag{2.22}$$

Hasonlóképpen

$$\bigcap S = \{x : x \in P \text{ minden } P \in S \text{ halmazra }\}. \tag{2.23}$$

Egy $A$ halmaz összes részhalmazainak halmazát az $A$ halmaz **hatványhalmaz**ának nevezzük. Jelölése: $\wp(A)$ vagy $2^A$. Például a $\{c, d\}$ halmaz részhalmazai $\{c, d\}$, $\{c\}$, $\{d\}$ illetve $\emptyset$, így

$$\wp(\{c, d\}) = 2^{\{c,d\}} = \{\emptyset, \{c\}, \{d\}, \{c, d\}\}. \tag{2.24}$$

Egy nemüres $A$ halmaz **particionálás**a alatt $2^A$ egy olyan $\Pi$ részhalmazát értjük, amelynek $\emptyset$ nem eleme és $A$ minden eleme $\Pi$-nek pontosan egy halmazában van benne. Tehát $\Pi$ $A$-nak egy particionálása, ha $\Pi$ $A$ részhalmazainak olyan halmaza, amelyre a következők teljesülnek:

1. $\Pi$ összes eleme nemüres
2. $\Pi$ elemei páronként diszjunktak
3. $\bigcup \Pi = A$

Például az $\{a, b, c, d\}$ halmaznak az $\{\{a, b\}, \{c\}, \{d\}\}$ egy partíciója, de a $\{\{b, c\}, \{c, d\}\}$ nem. A páros és páratlan számok halmaza egy megfelelő partícionálása a természetes számok halmazának.

## 2.2.2. Relációk és függvények

A matematika az objektumokkal kapcsolatos állítások mellett az objektumok közötti relációkkal foglalkozik. Például a „kisebb mint" egy reláció bizonyos típusú objektumok, nevezetesen a számok között, amely 4 és 7 esetén teljesül, de 4 és 2 esetén már nem. Hogyan tudjuk kifejezni az objektumok közötti relációkat a jelenleg rendelkezésünkre álló matematikai eszközökkel, azaz halmazokkal? Egyszerűen magát a relációt is egy halmaznak tekintjük. Például a „kisebb mint" reláció olyan párok halmaza, ahol a párok első száma kisebb, mint a második.

Relációkhoz tartozó párok esetén elengedhetetlenül szükséges, hogy a pár két tagját megfelelően meg tudjuk különböztetni. Ezeket a párokat halmazként nem írhatjuk fel, hiszen a $\{4,7\}$ halmaz például megegyezik a $\{7,4\}$ halmazzal. A probléma kezelésére bevezetjük a **rendezett pár** fogalmát.

Legyen $a$ és $b$ két objektum. Az $a$ és $b$ objektumokból képzett rendezett párt $(a, b)$-vel jelöljük; ekkor $a$-t és $b$-t az $(a, b)$ rendezett pár **komponens**einek hívjuk. Az $(a, b)$ rendezett pár nem egyenlő az $\{a, b\}$ halmazzal. Rendezett pár esetén fontos a sorrend, $(a, b)$ nem ugyanaz, mint $(b, a)$, ezzel szemben $\{a, b\}=\{b, a\}$. Egy rendezett pár két komponensének nem kell eltérőnek lennie; a $(7,7)$ egy teljesen megfelelő rendezett pár. Két rendezett pár $(a, b)$ és $(c, d)$ csak akkor egyenlő, ha $a = c$ és $b = d$.

Az $A$ és $B$ halmazok **Descartes-szorzat**a az összes olyan $(a, b)$ rendezett pár halmaza, ahol $a \in A$ és $b \in B$. Jelölése: $A \times B$. Például:

$$\{1,3,9\} \times \{b, c, d\} = \{(1, b), (1, c), (1, d), (3, b), (3, c), (3, d), (9, b), (9, c), (9, d)\} \quad (2.25)$$

Egy $A$ és $B$ halmazokon értelmezett bináris reláció az $A \times B$ halmaz egy részhalmaza. Például$\{(1, b), (1, c), (3, d), (9, d)\}$ egy bináris reláció az $\{1,3,9\}$ és $\{b, c, d\}$ halmazok között. A „kisebb mint" reláció nem más, mint az $\mathbf{N} \times \mathbf{N}$ halmaz egy részhalmaza, $\{(i, j) : i, j \in \mathbf{N}$ és $i < j\}$. A két halmaz, mely között bináris reláció áll fenn, gyakran identikus.

Vegyük az általános esetet, legyen $n$ egy tetszőleges természetes szám. Ha $a_1, a_2, \ldots, a_n$ $n$ darab egymástól nem szükségszerűen különböző objektum, akkor $(a_1, a_2, \ldots, a_n)$ egy **rendezett n-es** és bármely $i = 1, \ldots, n$-re $a_i$ az $(a_1, a_2, \ldots, a_n)$ $i$. komponense. A rendezett kettesek megegyeznek a fent bevezetett rendezett párokkal. Ha $A_1, A_2, \ldots, A_n$ tetszőleges halmazok, akkor az **n-szeres Descartes-szorzat** $A_1 \times \ldots \times A_n$ az összes $(a_1, a_2, \ldots, a_n)$ rendezett $n$-es halmaza, ahol minden $i = 1, \ldots, n$-re $a_i \in A_i$. Ha minden $A_i$ halmaz ugyanazt az $A$ halmazt jelöli, akkor az $A \times \ldots \times A$ szorzat helyett használhatjuk az $A^n$ alakot is. Például $\mathbf{N}^2$ a természetes számokból képzett rendezett párok halmaza. Egy $n$-szeres reláció az $A_1, A_2, \ldots, A_n$ halmazok között az $A_1 \times A_2 \times \ldots \times A_n$ halmaz egy részhalmaza.

Alapvető matematikai fogalom a függvény is. Ha intuitívan szeretnénk meghatározni, akkor a függvény nem más, mint bizonyos típusú objektumok mindegyikének egy valamilyen más típusú egyedi objektummal való társítása, például személyeket az életkorukkal, kutyákat a tulajdonosukkal, stb. A bináris reláció fogalmát felhasználva az intuitív meghatározás mellett egy konkrét definíciót is megadhatunk. Egy $A$ halmazból $B$ halmazba képező **függvény** egy olyan $R$ bináris reláció $A$ és $B$ között, amely teljesíti a következő speciális tulajdonságot: minden $a \in A$-hoz *pontosan egy* olyan rendezett pár tartozik $R$-ben, amelynek első komponense $a$. Példaként legyen $C$ a Magyarország városainak a halmaza $S$ pedig a megyék halmaza és legyen

$$R_1 = \{(x, y) : x \in C, y \in S \text{ és } x \text{ egy város } y \text{ megyében}\}, \tag{2.26}$$

$$R_2 = \{(x, y) : x \in S, y \in C \text{ és } y \text{ egy város } x \text{ megyében}\}. \tag{2.27}$$

Ekkor $R_1$ függvény, hiszen minden város csak egy megyéhez tartozik, $R_2$ azonban nem, mivel egy megyéhez több város is tartozik.

A függvényeket általában betűkkel, azok közül is gyakran $f$,$g$ vagy $h$-val jelöljük. $f: A \to B$ azt jelöli, hogy $f$ egy függvény $A$ és $B$ között. Az $A$ halmazt az $f$ függvény **értelmezési tartományának (domain)** nevezzük. Ha $a$ egy tetszőleges eleme $A$-nak, akkor $f(a)$-val jelöljük azt a $B$-beli $b$ elemet, amelyre $(a, b) \in f$; mivel $f$ egy függvény, pontosan egy olyan $b \in B$ lesz, amelyre ez teljesül, így $f(a)$ egy egyedi objektumot jelöl. Az $f(a)$ objektumot $a$ $f$ alatti **képének (image)** hívjuk. Egy $f: A \to B$ függvény megadásához elegendő minden $a \in A$-hoz a hozzátartozó $f(a)$ értéket megadni; így a példában szereplő $R_1$ függvény megadásához elegendő megadni minden városhoz, hogy melyik megyében található. Az $f$ függvény értelmezési tartományának képét $f$ **értékkészlet**ének **(range)** nevezzük.

Ha a függvény értelmezési tartománya egy Descartes-szorzat, akkor nem szoktunk minden zárójelet kitenni. Például ha az $f : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ függvényt úgy definiáljuk, hogy az $(m, n)$ rendezett pár képe$f$ alatt $m$ és $n$ összege, akkor a jelölés egyszerűsítése érdekében egyszerűen $f(m, n) = m+n$-t írunk $f((m, n)) = m+n$ helyett.

Ha $f : A_1 \times A_2 \times \ldots \times A_n \to B$ egy függvény és $f(a_1, \ldots, a_n) = b$ ahol $a_i \in A_i$ minden $i = 1, \ldots, n$-re és $b \in B$, akkor $a_1, \ldots, a_n$ -et a függvény **argumentum**ainak, $b$-t pedig az argumentumokhoz tartozó **érték**nek nevezzük. Így $f$ megadható az argumentumaihoz tartozó értékek meghatározásával is.

Az $f : A \to B$ függvényt **injekció**nak, vagy **kölcsönösen egyértelmű leképezés**nek nevezzük, ha bármely két különböző $a, a' \in A$ elemre $f(a)$ $f(a')$. Például legyen $C$ Magyarország városainak halmaza, $S$ a megyék halmaza, $g: S \to C$ pedig legyen a következőképpen megadva:

$$g(s) = s \text{ megye székhelye minden } s \in S\text{-re.} \tag{2.28}$$

Ekkor $g$ injekció, mivel két különböző megyének nem lehet ugyanaz a székhelye.

Az $f: A \to B$ függvény egy **szürjekció**, vagy **ráképezés** $B$-re, ha $B$ minden eleme valamely $A$-beli elem $f$ alatti képe. A $g$ függvény nem ráképezés, ellenben az $R_1$ függvény igen, mivel minden megyében van legalább egy város.

Az $f : A \to B$ leképezést **bijekció**nak, vagy **kölcsönösen egyértelmű ráképezés**nek nevezzük, ha egyidejűleg szürjekció és injekció is. Például ha $C_0$ a megyeszékhelyek halmaza, a $g : S \to C_0$ függvényt pedig a (2.28) egyenlet szerint definiáljuk, akkor $g$ egy bijekció $S$ és $C_0$ között.

Az $R \subseteq A \times B$ bináris reláció **inverz**e a $\{(b, a) : (a, b) \in R\}$ bináris reláció, amelynek jelölése: $R^{-1} \subseteq A \times B$. Például a fent definiált $R_2$ reláció az $R_1$ reláció inverze. Ebből következik, hogy egy függvény inverze nem feltétlenül függvény. $R_1$ inverze sem függvény, mivel vannak megyék, amikben több, mint egy város található, azaz van két olyan egymástól különböző $c_1, c_2$ város, amelyre $R_1(c_1) = R_1(c_2)$. Bijektív függvénynek mindig van inverze, ami szintén bijekció lesz. Továbbá ha $f : A \to B$ egy bijekció, akkor $f^{-}1(f(a)) = a$ minden $a \in A$-ra, illetve $f^{-}1(f(b)) = b$ minden $b \in B$-re.

### 2.2.3. Gráfok

Ebben a fejezetben, az optimalizálás területén alkalmazott alapvető matematikai eszközöket, a gráfokat mutatjuk be. A gráfok akkor nyújtanak segítséget, mikor egy megoldandó problémát több, egymással valamilyen kapcsolatban álló objektumként szeretnénk modellezni és a feladat megoldásához ezeket a kapcsolatokat kell figyelembe venni. Ilyen probléma lehet például az, hogy egy úthálózaton mely útvonalon haladva lehet elérni a célhoz a legrövidebb idő alatt, vagy egy gyárban hogy tudunk minél kevesebb kábel felhasználásával árammal ellátni minden berendezést. De gráfokkal leírható kapcsolat lehet az is, hogy egy munkafolyamatban melyik tevékenység melyiket követi.

Legyen $A$ egy halmaz, $R \subseteq A \times A$ pedig egy bináris reláció. Az $R$ reláció egy úgynevezett **irányított gráf**fal szemléltethető. $A$ elemeit ilyenkor apró körökkel ábrázoljuk, amiket az irányított gráf **csúcs**ainak hívunk. Az irányított gráf egy $a$ csúcsából nyíl mutat egy $b$ csúcsába akkor és csak akkor, ha $(a, b) \in R$. Ezeket a nyilakat az irányított gráf **él**einek nevezzük.

Az $R \subseteq A \times A$ relációt **szimmetrikus**nak mondjuk, ha minden $(a, b) \in R$ esetén $(b, a) \in R$. Ez azt jelenti, hogy a relációhoz tartozó irányított gráfban ha van egy nyíl két csúcs között, akkor a két csúcs között van az ellenkező irányba mutató él is. Az ilyen relációkat ábrázolhatjuk **irányítatlan gráf**fal is. Ilyenkor a csúcsokat nem oda-vissza mutató nyilak, hanem egyszerű vonalak kötik össze.

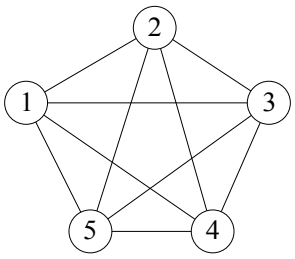
A relációk ábrázolásán kívül a gráfok sok gyakorlati feladat modellezésében is alapvető szerepet játszanak. A hálózati feladatok (úthálózat, kommunikációs hálózat, stb.) modellezéséhez a gráf, mint eszköz, természetes választásnak tűnik, de rengeteg egyéb gyakorlati feladat (szintézis, ütemezés, stb.) is hatékonyan kezelhető vele.

A gráfelmélet hatalmas tématerület, különböző ágairól könyveket írnnak, ezért átfogó tárgyalását a jegyzet meg sem kíséreli. A téma iránt mélyebben érdeklődő Olvasónak az alábbi könyvet ajánljuk: [11]. A továbbiakban azokat a fontos jelöléseket és alapfogalmakat vezetjük be, amelyek a jegyzet további részéhez, és az ott található gyakorlati alkalmazások megértéséhez szükségesek.

A jegyzetben megkülönböztetjük az irányított és irányítatlan gráfokat. Egy **irányított gráf**ot, vagy más néven digráfokat egy $\vec{G} = (V, A)$ párral adjuk meg, ahol $V$ a csúcsok halmaza (vertices, nodes), $A \subseteq V \times V$ pedig az élek halmaza (arcs). Ebben az esetben tehát az éleket rendezett párok reprezentálják, hiszen egy élen belül megkülönböztetjük a kezdő és a cél csúcsot.

Egy **irányítatlan gráf**okat pedig a $G=(V, E)$ párral adjuk meg, ahol $E \subseteq \{e_i \subseteq V$ és $1 \leq |e_i| \leq$ $\leq 2)$, azaz az éleket (edges) legfeljebb két elemű halmazok definiálják, melyek a kezdő és cél csúcsot tartalmazzák, mivel egy él végpontjait egyenrangúnak tekintjük. Abban az esetben,

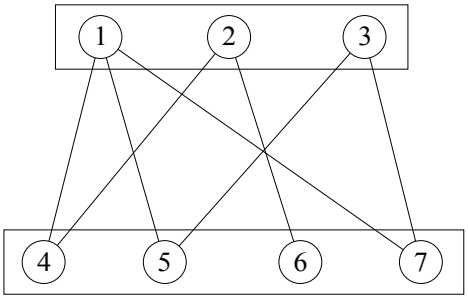
ha nem fontos, hogy egy gráf irányított, vagy irányítatlan, a jegyzet hátralevő részében a $G = (V, E)$ jelölést használjuk.




*2.1. ábra. Öt csúcsot tartalmazó teljes irányítatlan ($K_5$) gráf*

Olyan éleket, amelyek kezdő és vég csúcsa ugyanazon csúcs, **hurokél**nek hívjuk. A hurokél formális jelölése irányított gráfban egy olyan pár, melynek mindkét komponense azonos (például $(v_i, v_i)$), irányítatlan gráfban pedig a csúcsot tartalmazó egyelemű halamaz (például $\{v_i\}$). Azt mondjuk, hogy egy $v_i$ csúcs szomszédja egy $v_j$ csúcs akkor és csak akkor, ha irányítatlan esetben létezik $\{v_i, v_j\} \in E$ vagy irányított esetben $(v_i, v_j) \in A$ él.

Egy $G = (V, E)$ gráf részgráfja az a $H = (W, F)$ gráf, ahol $W \subseteq V, F \subseteq E$, úgy, hogy ha $\{i, j\} \in F$, akkor $i, j \in W$.

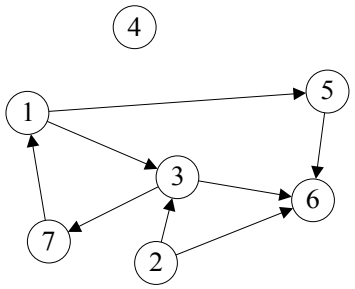



*2.2. ábra. Páros gráf*

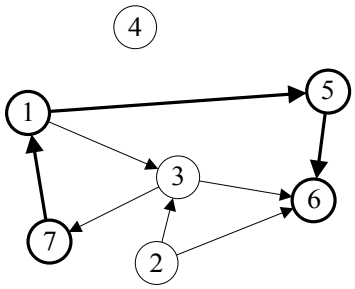
A nevezetes gráfok közé tartozik a **teljes gráf**, ahol $E = V^2$, azaz minden csúcs szomszédja minden másik csúcsnak. Az $n$ csúcsú teljes gráfot $K_n$-nel jelöljük. A 2.1 ábrán egy $K_5$ gráf látható.

A nevezetes gráfok másik fontos csoportját alkotják a **páros gráf**ok. Ezek olyan $(V, E)$ gráfok, ahol a csúcsok halmazát két partícióra oszthatjuk ($V = A \cup B$, $A \cap B = \emptyset$) úgy, hogy minden él egyik végpontja az egyik, a másik végpontja a másik partícióban van (bármely $\{e_i, e_j\} \in E$ esetén $e_i \in A$ és $e_j \in B$). Például, a 2.2 ábra egy olyan páros gráfot tartalmaz, melyben a csúcsok az 1, 2, 3 és 4, 5, 6, 7 partíciókra bontható.

Sok gyakorlati feladat esetén nem elégséges azt modelleznünk, hogy a valóság milyen objektumai állnak kapcsolatban egymással, hanem e kapcsolatnak a minőségét is tudnunk kell formálisan kezelni. Például azon kívül hogy, tudjuk, hogy egyik városból el lehet jutni a másik városba, fontos lehet, hogy az utazás mennyi időt vesz igénybe. Ipari gyártórendszerben, egy csőhálózatban, alapvetően fontos a cső áteresztő képessége, hossza, vagy kapacitása. E modellezési folyamat eszköze lehet, ha az objektumokat reprezentáló gráf csúcsokat összekötő élekhez súlyokat rendelünk. Ekkor **súlyozott gráf**ról beszélünk. A súlyt egy $w_E : E \to \mathbb{R}$,




*2.3. ábra. Izolált csúcsot (4-es csúcs) tartalmazó irányított gráf.*



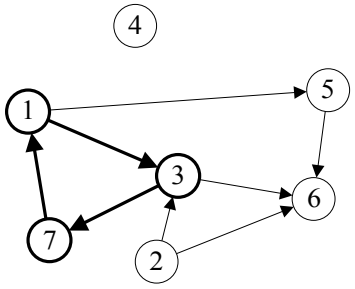

*2.4. ábra. Irányított út gráfban (7 – 1 – 5 – 6)*

illetve $w_A : A \to \mathbb{R}$ súlyfüggvénnyel adjuk meg. A gráf ábrázolásakor a súlyokat az élekre írva tüntetjük fel.

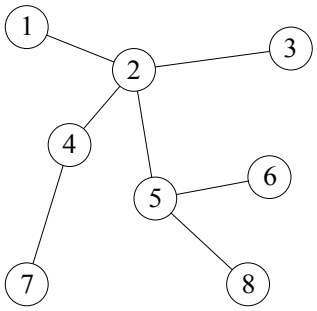
Egy csúcs fokszáma az a szám, amennyi élhez csatlakozik az adott csúcs. A 0 fokszámú csúcsokat izolált csúcsnak nevezzük. A 2.3 ábrán az 1-es csúcs fokszáma 3, a 4-es csúcs pedig izolált csúcs.

Két csúcs szomszédos, ha él köti őket össze. A szomszédos csúcsok és az őket összekötő élek váltakozó sorozatát **sétá**nak nevezzük. Azt a sétát, amin belül a csomópontok nem ismétlődnek **út**nak nevezzük. A 2.4 ábrán láthatunk példát olyan útra, ahol a 7-es, 1-es, 5-ös és 6-os csúcsokat látogatjuk meg. Azt a sétát, amely első és utolsó csomópontját kivéve minden csomópont különböző, körnek nevezzük, ilyen a 2.5 ábrán az 1-es, 3-as, 7-es csomópontok és köztül levő élek sorozata.

Egy gráfra azt mondjuk, hogy **összefüggő**, ha bármely csomópontjából eljuthatunk egy út mentén bármely másik csomópontjába. Egy gráfot **körmentes**nek hívjuk, ha nincs olyan




*2.5. ábra. Irányított kör a gráfban: (7 – 1 – 3)*




*2.6. ábra. Fa gráf*

részgráfja, mely kör. Az összefüggő körmentes gráfokat **fá**nak nevezzük. Ilyen például a 2.6 ábrán látható gráf.

## 2.3. Feladatok

**2.1. Feladat** Igazak-e az alábbi állítások?

(a) $\emptyset \subseteq \emptyset$

(b) $\emptyset \in \emptyset$

(c) $\emptyset \in \{\emptyset\}$

(d) $\emptyset \subseteq \{\emptyset\}$

(e) $\{a, b\} \in \{a, b, c, \{a, b\}\}$

(f) $\{a, b\} \subseteq \{a, b, \{a, b\}\}$

(g) $\{a, b\} \subseteq 2^{\{a,b,\{a,b\}\}}$

(h) $\{\{a, b\}\} \in 2^{\{a,b,\{a,b\}\}}$

(i) $\{a, b, \{a, b\}\} \setminus \{a, b\} = \{a, b\}$

**2.2. Feladat** Mi lesz az alábbi műveletek eredménye?

(a) $(\{1,3,5\} \cup \{3,1\}) \cap \{3,5,7\} = ?$

(b) $\bigcup\{\{3\}, \{3,5\}, \bigcap\{\{5,7\}, \{7,9\}\}\} = ?$

(c) $(\{1,2,5\} \setminus \{5,7,9\}) \cup (\{5,7,9\} \setminus \{1,2,5\}) = ?$

(d) $2^{\{7,8,9\}} \setminus 2^{\{7,9\}} = ?$

(e) $2^{\emptyset} = ?$

# 3. fejezet

# Gráf algoritmusok

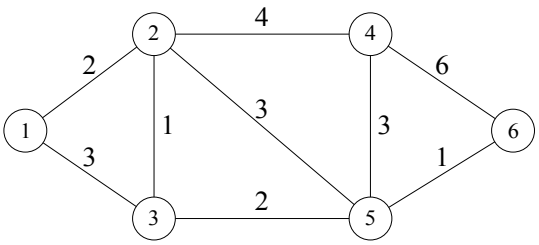
A gráfok sok, gyakorlatban felmerülő feladat esetén kézenfekvő modellező eszközök. Az alábbiakban néhány nevezetes feladatot és a feladatok algoritmust ismertetünk, mely sok gyártórendszerekhez kapcsolódó gyakorlati feladat megoldására alkalmas.

## 3.1. Minimális feszítőfa meghatározása

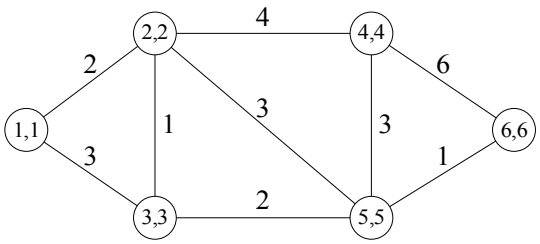
A $G = (V, E)$ gráf minimális feszítőfája olyan részgráfja, amely fa,tartalmaz minden $v \in V$ csomópontot, továbbá a fa éleinek összsúlya minimális. Minimális feszítőfa meghatározása optimális megoldást adhat olyan tervezési feladatra, ahol költségoptimális hálózatot kell terveznünk, melyben bármely két objektum között kell útnak lennie és a hálózat költsége az összeköttetések fix költségének összegéből adódik. Ekkor a kiindulási gráf az összes lehetséges összeköttetést tartalmazza, az élek súlya pedig az összeköttetés megvalósításának költsége.

A minimális feszítőfa meghatározására több algoritmus is létezik, az alábbiakban a Kruskal algoritmust ismertetjük, és működését egy példán mutatjuk be. Tekintsük a 3.1 ábrán szereplő gráfot.

Az algoritmus a 3.3 ábrán látható. Lényege, hogy a meghatározandó feszítőfába sorra megpróbáljuk beszúrni a még nem vizsgált legrövidebb élt. Amennyiben az él beszúrásával az épülő gráfban kör alakulna ki, az élt nem szúrjuk be. Előfordulhat, hogy az épülő fa egy ideig több különböző komponensből áll, amelyek később kapcsolódnak össze a megfelelő élek hozzáadása után. Kör akkor alakulhat ki, ha egy él egy meglevő komponenshez tartozó két




*3.1. ábra. A lehetséges kapcsolatokat és azok költségét leíró gráf*




*3.2. ábra. Minden csúcs különböző partícióban*

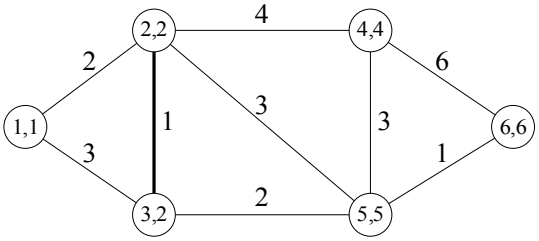
csúcsot köt össze. Ezért élt csak különböző komponensbe tartozó csúcsok közé szúrhatunk be. Ennek egyszerű vizsgálatához „színezzük” meg az összes csomópontot különböző színekkel. A színezést jelölhetjük különböző számozással is, ahogy azt a 3.2 ábra mutatja.

**bemenet:** $G(V, E, w)$ súlyozott irányítatlan gráf
**kimenet:** $G^*(V, E^*, w)$ a bemeneti gráf minimális súlyú feszítőfája

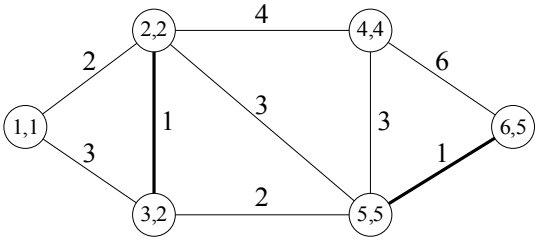
$i := 1$
**ciklus** minden $v \in V$-re
    Színek[$v$] := $i$;
    $i := i+1$;
**ciklus vége**
Élek := az $E$ elemeit súly szerint növekvő sorrendben tartalmazó vektor;
$E^* := \emptyset$;
$i := 1$;
**ciklus** amíg $i \leq |E|$ és $|E^*| \leq |V|-1$
    $\{v_i, v_j\}$ :=Élek[$i$];
    **ha** Színek[$v_i$] Színek[$v_j$] **akkor**
        $E^* := E^* \cup \{\{v_i, v_j\}\}$;
        **ciklus** minden $v \in V$-re
            **ha** Színek[$v$] =Színek[$v_j$] **akkor**
                Színek[$v$] :=Színek[$v_i$];
            **elágazás vége**
        **ciklus vége**
    **elágazás vége**
    $i := i+1$;
**ciklus vége**
**ha** $|E^*| = |V|-1$ **akkor**
    kiír $(V, E^*, w)$;
**különben**
    kiír „A megadott gráfnak nincs feszítőfája.”;
**elágazás vége**

*3.3. ábra. Kruskal algoritmusa*

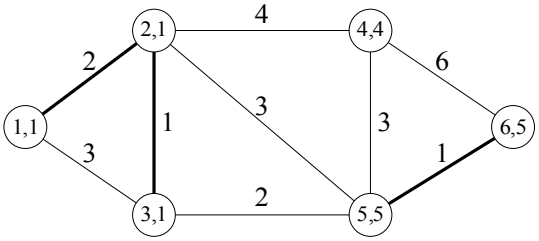



*3.4. ábra. Az első iteráció után*




*3.5. ábra. A második iteráció után*

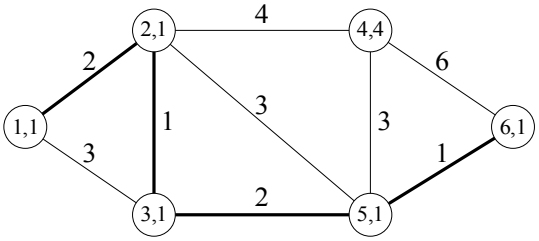
A csúcsoknál két számot látunk. Az első a csúcs sorszáma, a második pedig, hogy hányas partícióba tartozik. Kezdetben minden csúcs külön partícióban van. Az algoritmus előrehaladásával a partíciók száma csökken. Egy él beszúrása után az él egyik végpontjába tartozó csomópontokat áttesszük a másik csomópont partíciójába, azaz a két partíciót „összeolvasztjuk". Az algoritmus sikeres futásának eredményeként minden csomópont azonos partícióban lesz. Ennek egyetlen akadálya az lehet, ha az induló gráf nem összefüggő. Ekkor nincs benne feszítőfa.




*3.6. ábra. A harmadik iteráció után. Látható hogy a feszítőfa több komponensből fog összeállni*

Fontos kérdés, hogy meddig fusson az algoritmus? Tudjuk, hogy minden fában $|E| = |V| - 1$, tehát az algoritmust addig kell futtatni, amíg a beszúrt élek száma el nem éri a $|V| - 1$-et. Megjegyezzük, hogy önmagában az a tény, hogy egy gráfban $|E| = |V| - 1$, még nem jelenti azt, hogy a gráf fa. Ehhez szükséges az is, hogy a gráf összefüggő legyen, vagy ne legyen benne kör. Az él beszúrásának ismertetett szabálya miatt biztos, hogy nem alakul ki kör a meghatározott részgráfban, ezért a kapott gráf biztosan fa lesz. Továbbá, a fa élsúlyainak összege minimális, ennek bizonyításától most eltekintünk, de az érdeklődők megtalálhatják például ([21])-ben.

Lássuk az algoritmus működését egy példán. Határozzuk meg a 3.2 ábrán látható gráfban a minimális feszítőfát. A fába választott éleket a további ábrákon vastag vonallal jelöljük. Az



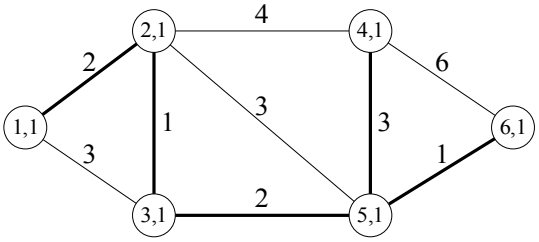

*3.7. ábra. A negyedik iteráció után*

éleket súly szerint rendezzük növekvő sorrendbe, majd a legkisebb súlyú élt beválasztjuk a feszítőfába. Válasszuk kezdetben a {2, 3} élt, majd a 3-as csúcs színét írjuk át arra, ami a 2-es csúcs színe, ezzel jelezzük, hogy a 2-es és 3-as csúcsok egy partícióba tartoznak (3.4 ábra).

A következő legrövidebb él az {5, 6}. Mivel az 5-ös és 6-os csúcsok színei különbözőek, biztos, hogy ha beszúrjuk ezt az élt, akkor nem keletkezik kör, tehát az élt beválasztjuk, majd a két csúcsot tartalmazó partíciókat összeolvasztjuk (3.5 ábra).

A következő legrövidebb élsúly a 2, válasszuk most az {1,2} élt. Ekkor az először létrehozott partíció új éllel bővül, majd a partíció minden elemét az 1-es csúcs színére színezzük (3.6 ábra).

A következő 2 súlyú élt is beszúrhatjuk, hiszen a 3-as és 5-ös csúcs más komponenshez tartozik. Ekkor a két meglevő partíció egybeolvad, minden ide tartozó csúcs színe 1 lesz (3.7 ábra).




*3.8. ábra. Az ötödik iteráció után megkapjuk a minimális feszítőfát*

A következő élsúly a 3, vizsgáljuk meg először az {1, 3} élt. Itt az 1-es és 3-as csúcs színe 1, tehát azonos partícióhoz tartoznak, ezért van már út az 1-es csúcsból a 3-asba, a két csúcs közé újabb élt behúzva kört hozunk létre, tehát ezt az élt elvetjük. Ugyanez igaz a {2, 5} élre is, viszont a {4, 5} élt behúzhatjuk (3.8 ábra). Az élek száma ekkor 5, ami egyel kisebb, mint a csomópontok száma, tehát az algoritmus sikeresen felépítette a minimális feszítőfát.

Megjegyezzük, hogy a körök kialakulásának vizsgálatára több módszer is létezik, az érdeklődő olvasóknak ajánljuk a diszjunkt halmazok modellezésére való adatszerkezeteket és algoritmusokat ([22]).

## 3.2. Legrövidebb út meghatározása

Ipari és logisztikai folyamtok tervezésekor és üzemeltetésekor gyakran felmerülnek olyan kérdések, hogy ismert kapcsolatrendszerben egy objektum hogyan érhető el legkönnyebben

egy másikból térben vagy időben. Például hogyan juthatunk leghamarabb egyik helyről a másikra, vagy melyik terméket tudjuk legelőször csomagolni. Ha az objektumok kapcsolatait gráfon modellezzük, és gráfban az élek súlya annál kisebb, minél könnyebben elérhető egyik a másikból, akkor ilyen kérdésekre gyakran megadja a választ a két objektum közti legrövidebb út meghatározása.

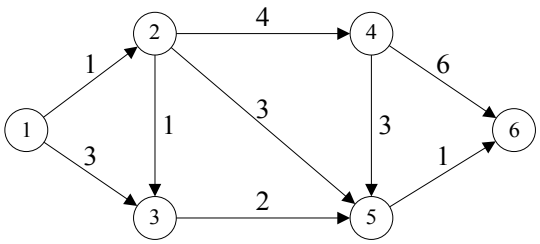
**bemenet:** $G(V, A, w)$ súlyozott irányított gráf, $s, t \in V$ csúcsok
**kimenet:** $(V^*, A^*)$ a legrövidebb út $s$-ből $t$-be

$F := \emptyset$;
**Ciklus** minden $v \in V$-re
    $d(v) := \infty$;
    $p(v) := v$;
**Ciklus vége**
$d(s) := 0$;
**Ciklus** amíg van $v \in V \setminus F$, melyre $d(v) < \infty$
    Legyen $v^* \in V \setminus F$, melyre $d(v^*)$ minimális;
    $F := F \cup \{v^*\}$;
    **Ciklus** minden $(v^*, u) \in A$-ra
        **Ha** $d(v^*) + w(v^*, u) < d(u)$ **akkor**
            $d(u) := d(v^*) + w(v^*, u)$;
            $p(u) := v^*$;
        **Elágazás vége**
    **Ciklus vége**
**Ciklus vége**
**Ha** $d(t) < \infty$ **akkor**
    $v := t$
    **Ciklus** amíg $v$ $s$
        $V^* := V^* \cup \{v\}$;
        $A^* := A^* \cup \{(p(v), v)\}$;
        $v := p(v)$;
    **Ciklus vége**
    $V^* := V^* \cup \{s\}$;
    Kiír $(V^*, A^*)$;
**Különben**
    Kiír „Nincs út a gráfban $s$-ből $t$-be.”;
**Elágazás vége**

*3.9. ábra. Dijkstra algoritmusa*

Egy adott $G=(V, A)$ gráfban egy út hossza az utat alkotó élek súlyainak összege. Súlyozatlan gráf esetén az élek súlyait 1-nek tekintjük. Gyakran szükségünk van két csomópont között ($s$-ből $t$-be) a legrövidebb út meghatározására. Az egyik legismertebb algoritmust Dijkstra

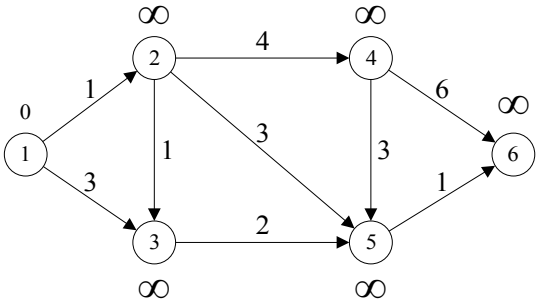
adta meg, amelyet Dijkstra algoritmusnak nevezünk. Az algoritmus a 3.9 ábrán látható. Futásának eredményeként megadja egy adott $s$ csúcsból a legrövidebb utat minden más csúcsba.




*3.10. ábra. Ebben az irányított gráfban keressük a legrövidebb utakat az 1. csomópontból*

Kezdetben minden csúcsra megbecsüljük, hogy milyen távolságban van az s csúcstól. Jelöljük ezt egy$d : V \to \mathbb{R}$ függvénnyel, ahol kezdetben $d(s) := 0$, és $d(v) := \infty$, $\forall v \in V \setminus \{s\}$, azaz az s csúcsból önmagába 0 hosszúságú úton juthatunk el, a többi csúcs távolságát pedig végtelennek tekintjük mindaddig, míg meg nem győződtünk róla, hogy elérhető egyáltalán a kiindulási csúcsból. Az algoritmus futása során a távolságok becsléseit folyamatosan pontosítjuk. Az F halmazba gyűjtsük azokat a csomópontokat, amelyekre tudjuk, hogy a rájuk becsült távolság pontos. Kezdetben ez a halmaz legyen üres. Végül, hogy a legrövidebb utat tárolni tudjuk, szükségünk lesz egy függvényre, amely minden csúcshoz hozzárendeli a kiindulás csúcsból hozzá vezető legrövidebb úton az őt megelőző csúcsot: $p : V \to V$. Kezdetben minden csúcs megelőzőjeként önmagát tüntetjük fel.

| | d | F | p |
|---|---|---|---|
| 1 | 0 | - | 1 |
| 2 | ∞ | - | 2 |
| 3 | ∞ | - | 3 |
| 4 | ∞ | - | 4 |
| 5 | ∞ | - | 5 |
| 6 | ∞ | - | 6 |




*3.11. ábra. Az iterációk előtti állapot*

Egy iteráció a következő lépésekből áll. Kiválasztjuk a $v \in V \setminus F$ csúcsok közül (tehát ahol a távolság becslések még nem pontosak) azt, amelyre a távolság becslés minimális. Ekkor a $v$ csúcsot betesszük az F halmazba, ezzel jelezzük, hogy az $s$-ből $v$-be menő legrövidebb út hossza pontosan ismert, mégpedig $d(v)$. Megvizsgáljuk a $v$ csúcs minden $u$ szomszédját, hogy $u$-ba $v$-n keresztül rövidebb úton juthatunk-e el, mint $d(u)$. Azaz, ha teljesül, hogy $d(v)+$ $+w(v, u) < d(u)$, akkor találtunk egy rövidebb utat $u$-ba. Ekkor $d(u) := d(v)+w(v, u)$ lesz, majd feljegyezzük, hogy u-ba a v csúcsból jutottunk el: $p(u) := v$.

Az iterációkat addig ismételjük, amíg van olyan $v \in V \setminus F$ csúcs, ahol $d(v) < \infty$. Amennyiben az algoritmus lefutása után $V \setminus F$ $\emptyset$, úgy az $s$ csúcsból nem érhetőek el irányított út mentén a $V \setminus F$-beli csomópontok.

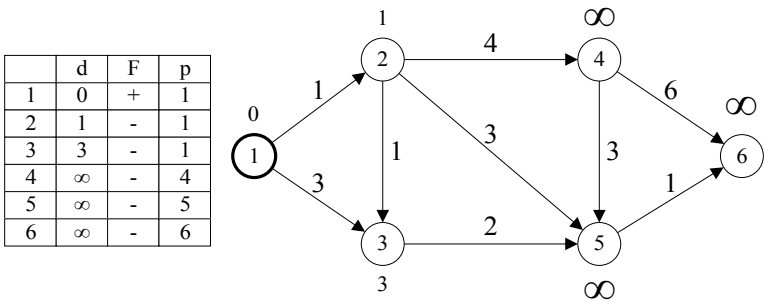
Az iterációk után $d(t)$ megadja a legrövidebb út hosszát $s$-ből kiindulva bármely $t \in F$-re. Tudjuk, hogy $v = p(d)$ az a csomópont, amelyről a legrövidebb úton tovább kell lépni a $t$-re. A $p$ függvényt felhasználva $t$-ből $s$ felé visszafelé haladva sorra kiolvashatjuk a legrövidebb útban résztvevő csúcsokat.

Egy példán bemutatjuk az algoritmus működését. A 3.10 ábrán látható gráfban keressük a legrövidebb utat az 1-es csomópontból a 6-osba.

A gráfon a továbbiakban a $d$ értékeket is feltüntetjük (3.11 ábra).

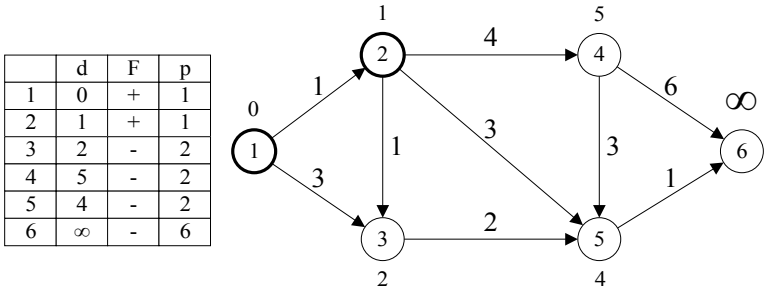
Kezdetben csak az 1-es csomópontot választhatjuk, hiszen $d(1) = 0$, ami ekkor a legkisebb érték, és az 1-es csomópont még nincs benne az $F$ halmazban. Az 1-es csomópont szomszédai a 2-es és 3-as, az 1-esből mindkettőbe rövidebb úton lehet eljutni mint $\infty$, ezért ezen csúcsok $d$ értékeit frissítjük, valamint jelöljük, hogy mindkettőbe az 1-esen keresztül lehet eljutni (3.12 ábra). Az 1-es csomóponttal végeztünk, ezért berakjuk az $F$ halmazba.

A következő lépésben az 2-es és 3-as csomópontok közül választhatunk. Mivel a 2-es csomóponthoz tartozó $d$ érték a kisebb, ezért ezzel a csomóponttal megyünk tovább. Az 2-es szomszédai a 3, 4 és 5. Amennyiben lenne egy 1-be menő él is a 2-esből, akkor az 1-es csomóponton nem változtatunk, hiszen egyrészt itt a $d$ értéke pontos, másrészt ha a 2-esről visszalépnénk az 1-re, akkor csak feleslegesen növelnénk az út hosszát. Az ismertetett algoritmusban nem vizsgáltuk külön, hogy csak a $V \setminus F$ halmazbeli szomszédokat frissítsük, hiszen az említett jelenség miatt ezeket nem fogja már frissíteni az algoritmus. A 3-as csomóponthoz tartozó $d$ érték is kisebb lesz, mivel ha a 2-esről megyünk a 3-asra, akkor rövidebb úton jutunk el oda, mintha az 1-esről indulnánk. A 4 és 5-ös csomópontok $d$ értékei is frissülnek, valamint jelezzük, hogy a 3-as, 4-es és 5-ös csomópontba a 2-esből juthatunk el (3.13 ábra).




| | d | F | p |
|---|---|---|---|
| 1 | 0 | + | 1 |
| 2 | 1 | - | 1 |
| 3 | 3 | - | 1 |
| 4 | ∞ | - | 4 |
| 5 | ∞ | - | 5 |
| 6 | ∞ | - | 6 |

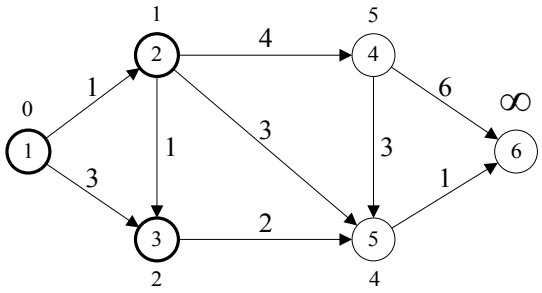
*3.12. ábra. Az első iteráció után*




| | d | F | p |
|---|---|---|---|
| 1 | 0 | + | 1 |
| 2 | 1 | + | 1 |
| 3 | 2 | - | 2 |
| 4 | 5 | - | 2 |
| 5 | 4 | - | 2 |
| 6 | ∞ | - | 6 |

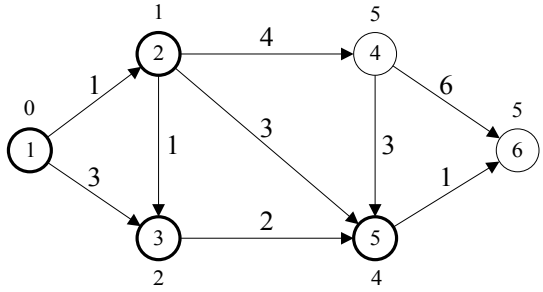
*3.13. ábra. A második iteráció után*

| | d | F | p |
|---|---|---|---|
| 1 | 0 | + | 1 |
| 2 | 1 | + | 1 |
| 3 | 2 | + | 2 |
| 4 | 5 | - | 2 |
| 5 | 4 | - | 2 |
| 6 | ∞ | - | 6 |



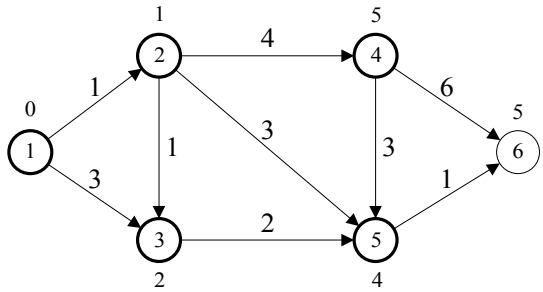

*3.14. ábra. A harmadik iteráció után*

| | d | F | p |
|---|---|---|---|
| 1 | 0 | + | 1 |
| 2 | 1 | + | 1 |
| 3 | 2 | + | 2 |
| 4 | 5 | - | 2 |
| 5 | 4 | + | 2 |
| 6 | 5 | - | 5 |




*3.15. ábra. A negyedik iteráció után*

| | d | F | p |
|---|---|---|---|
| 1 | 0 | + | 1 |
| 2 | 1 | + | 1 |
| 3 | 2 | + | 2 |
| 4 | 5 | + | 2 |
| 5 | 4 | + | 2 |
| 6 | 5 | - | 5 |




*3.16. ábra. Az ötödik iteráció után*

A következő választható csomópont a 3-as, mert a becsült $d$ itt a legkisebb és ez az érték egyben pontos is, tehát ez is bekerül az $F$ halmazba. Most nem tudunk változtatni a többi csomópont $d$ értékén, mivel az 5-ös csomópontba ugyanakkora úton lehet eljutni 2-ből mint 3-ból (3.14 ábra).

A $d$ értékei szerint az 5-ös csomópont következik, ennek a szomszédja a 6-os, aminek a $d$ és $p$ értékét frissíthetjük (3.15 ábra). A 4-es csomópont elérése (3.16 ábra) nem javít egyetlen becsült úthosszon sem. Végül marad a 6-os csomópont véglegesítése (3.17 ábra).

Mivel már minden csomópont benne van az $F$ halmazban, ezért nem hajtunk végre újabb iterációt. Mivel $d(6)\ \infty$, ezért találtunk útvonalat. Az utolsó lépésben pedig határozzuk meg, hogy mely csomópontokon át lehet eljutni 1-ből 6-ba. Mivel $p(6) = 5$, tehát az út utolsó előtti állomása az 5-ös. $p(5) = 2$, tehát ide a 2-esről lépünk. Végül $p(2) = 1$, tehát a legrövidebb utat alkotó csomópontok az 1-es, 2-es, 5-ös és a 6 -os (3.18).

## 3.3. Gráf bejárások

Mérnöki tervezési feladatok során olyan kérdések is felmerülnek, ahol nem egy legjobb megoldás megtalálása a cél, hanem csak eldöntendő kérdésekre kell választ adnunk. Ilyen kérdés lehet, hogy bizonyos objektumok között van-e közvetlen vagy közvetett kapcsolat. Például, hogy a gyár két üzeme között van-e vezetékes összeköttetés, vagy hogy egy ütemezésben két feladatnak meg kell-e előznie egymást. Ha az objektumok vagy események kapcsolatait gráfon reprezentáljuk, akkor ilyen kérdésekre adhatnak választ a gráf bejárások.

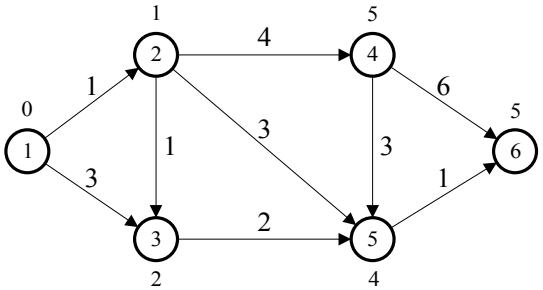
Egy gráfon vizsgálhatjuk azt, hogy egy adott csomópontból létezik-e út egy másikba, a gráf összefüggő-e. Ennek eldöntésére alkalmas a Dijkstra-algoritmus is, de ha nem szükséges, hogy a legrövidebb utat találjuk meg, akkor a szélességi vagy mélységi bejárás is alkalmazható. Bejárás helyett gyakran használják a keresés szót is. A két algoritmus egy adott csomópontból indul ki. Nyilvántartjuk azokat a csomópontokat, amelyeknek a szomszédait még nem vizsgáltuk meg, kiválasztunk közülük egyet és megjelöljük ennek a szomszédait. Ezt addig ismételjük, amíg van olyan csomópont amit megjelöltünk, de a szomszédait még nem vizsgáltuk. Ügyelni kell arra, hogy ha léteznek körök a gráfban, akkor így végtelen ciklusba kerülhet az algoritmus, hiszen a kör csomópontjait újra és újra végigjárjuk. Ezt úgy lehet kivédeni, hogy csak azokat a szomszédokat jelöljük meg, amelyeket még nem vizsgáltunk. A következő vizsgálandó csomópont kiválasztására két fő módszer létezik, ez alapján beszélünk szélességi, illetve mélységi bejárásról.

A szélességi bejárást úgy képzelhetjük el, mint mikor egy városban elindulunk megkeresni egy házszámot úgy, hogy a kiindulási csomópontból egyre nagyobb sugarú körzetben keresünk, a várost „széltében” járjuk be. A mélységi keresés során pedig elindulunk az egyik irányba, majd elmegyünk addig, ameddig lehetséges, ha nem értük el a célt, akkor folytatjuk a keresést egy másik irányba, tehát ekkor kevesebb lépésből jóval „mélyebbre” jutunk a városban. Először a szélességi keresést, majd a mélységi keresést ismertetjük.

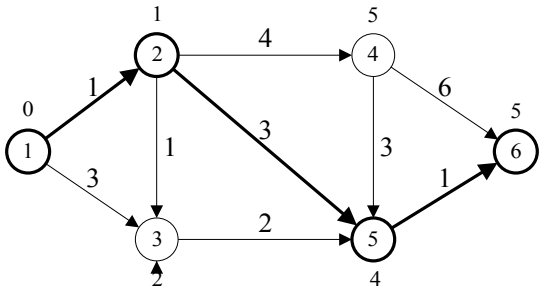
### 3.3.1. Szélességi bejárás

A kezdő csomópontot belerakjuk egy sor adatszerkezetbe. A sor a következő módon működik. Minden új elemet a végére fűzünk és csak a sor elején lévő elemet vehetjük ki belőle. Szokás FIFO adatszerkezetnek is mondani, ami az „elsőnek be, elsőnek ki” (first in, first out) elv rövidítése. Miután az első csúcspontot beillesztettük a sorba, következnek az iterációk.

| | d | F | p |
|---|---|---|---|
| 1 | 0 | + | 1 |
| 2 | 1 | + | 1 |
| 3 | 2 | + | 2 |
| 4 | 5 | + | 2 |
| 5 | 4 | + | 2 |
| 6 | 5 | + | 5 |




*3.17. ábra. A hatodik iteráció után*




*3.18. ábra. A legrövidebb út 1-ből a 6-ba: (1 – 2 – 5 – 6)*

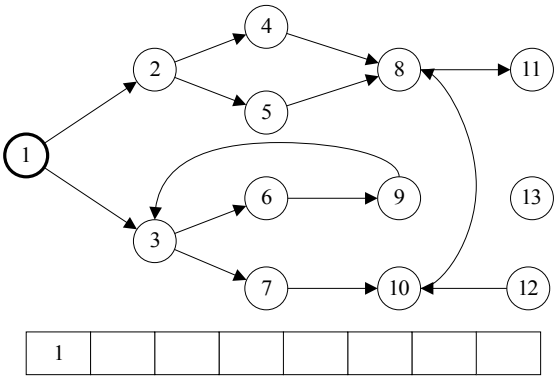
Minden iterációban megnézzük, hogy van-e elem a sorban, ha nincs, akkor az algoritmus megáll. Ellenkező esetben kivesszük az első elemet és ennek a szomszédait egyenként a sor végére fűzzük. Minden szomszédról megjegyezzük, hogy ők már egyszer bekerültek a sorba. Ha egy kör miatt később az algoritmus visszatérne ezekhez a csomópontokhoz, akkor ezeket már nem illesszük be újra, így elkerüljük a végtelen ciklust. Másrészt, felesleges ezeket újra megvizsgálni, hiszen egyszer már bebizonyítottuk, hogy ezek a csomópontok elérhetőek irá-

**bemenet:** $G(V, A)$ súlyozott irányított gráf, $s, t \in V$ csúcsok
**kimenet:** Elérhető-e a $t$ csúcs $s$-ből irányított úton

```
Elérhető:=hamis;
Ciklus minden v ∈ V\{s}-re
    Vizsgált[v] :=hamis;
Ciklus vége
Sorba(s);
Vizsgált[s] :=igaz;
Ciklus amíg a sor nem üres és nem Elérhető
    v :=Sorból;
    Ciklus minden (v, u) ∈ A-ra, amíg nem Elérhető
        Ha u = t akkor
            Elérhető:= igaz;
        Különben
            Ha nem Vizsgált[u] akkor
                Sorba(u);
                Vizsgált[u] :=igaz;
            Elágazás vége
        Elágazás vége
    Ciklus vége
Ciklus vége
Kiír Elérhető;
```

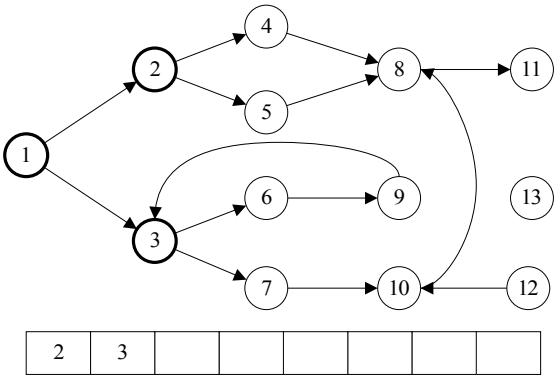
*3.19. ábra. Szélességi bejárás*




*3.20. ábra. Ezt a gráfot járjuk be az 1-es csomópontból kiindulva*

nyított úton a kezdő csomópontból. Az algoritmus nem csak akkor érhet véget, ha elfogytak az elemek a sorból. Előfordulhat, hogy az a feladat, hogy el kell dönteni, hogy egy $v \in V$ csomópontból elérhető-e irányított úton egy $u \in V$ csomópont, és haigen, akkor milyen útvonalon. Ez esetben az algoritmus akkor is megállhat, ha a sorból kiválasztottunk egy $w \in V$ csomópontot, aminek az egyik szomszédja $u$. Útvonalat hasonló módon lehet meghatározni, mint ahogy azt a Dijkstra-algoritmusnál láthattuk. Az alábbiakban olyan algoritmust mutatjuk be, ahol el kell dönteni, hogy létezik-e irányított út $s$-ből $t$ csomópontba. Az algoritmus pszeudokódja a 3.19 ábrán láthatjuk.




*3.21. ábra. Az első iteráció után*

Az algoritmus működését a 3.20 ábrán látható gráfon mutatjuk be. A megválaszolandó kérdés, hogy létezik-e irányított út az 1-es csomópontból a 12-esbe. Az algoritmus végigkövetése során vastag körvonallal jelöljük azokat a csomópontokat, amelyeket már vizsgáltunk, a gráf alatt pedig feltüntetjük a sor aktuális tartalmát.

Kezdetben beszúrjuk a sorba a kiinduló csúcspontot. Az első iterációban kivesszük a sorból az 1-es csúcsot, majd annak szomszédait beillesszük a sorba, valamint ezeket vizsgáltnak jelöljük (3.21 ábra).

Ezután a sor elején a 2-es csomópont áll. Ezt kivesszük, és beillesztjük szomszédait a 4-est és 5-öst (3.22 ábra).

A 3-as csomópont következik a sorban. Szomszédai a 6-os és 7-es (3.23 ábra). Jól látható, hogy miért szélességi bejárás az algoritmus neve. Először a kiinduló csomóponthoz legköze-

lebb eső csomópontokat vizsgáljuk, majd fokozatosan távolodunk tőle. Csak akkor lépünk az $n+1$ távolságra álló csomópontokra, ha megvizsgáltuk az összes $n$ távolságra elhelyezkedőt.

A 4-es kivétele után beszúrjuk a 8-ast (3.24 ábra). Az 5-ös következik. Az 5-ös szomszédja szintén a 8-as, de a 8-ast már megjelöltük, ezért mégegyszer nem szúrjuk be a sorba, így a sor rövidebb lesz (3.25 ábra). A 6-os csomópontot vesszük ki a sorból, majd beillesztjük a 9-est (3.26 ábra). A 7-es kivétele után szúrjuk be a 10-est (3.27 ábra).

A 8-as szomszédja a 11-es, ezzel felderítettünk minden 1-es csomópontból elérhető elemet (3.28 ábra), ám az algoritmus még nem ér véget, még meg kell mutatni, hogy semmilyen úton nem érhetünk el a 12-es csomópontba.

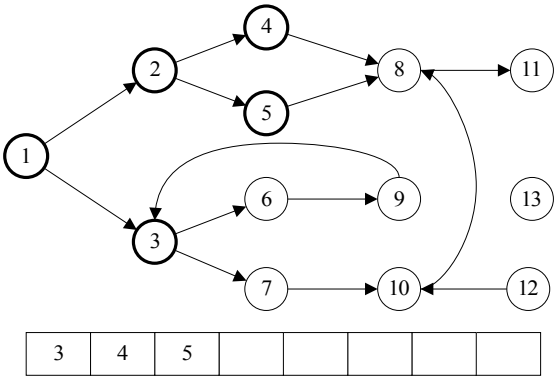
A 9-es szomszédja a 3-as, ám ha a 3-ast újra beszúrjuk, akkor végtelen ciklusba kerülünk, hiszen a sorban el fogunk érni a 3-as csomóponthoz, majd onnan újra elérünk a 9-eshez. Mivel tudjuk, hogy a 3-ast már vizsgáltuk, ezért ezt elkerülhetjük úgy, hogy a 3-ast már nem szúrjuk be újra (3.29 ábra).

Mivel sem a 10-esnek, sem a 11-es csomópontnak nincsenek már jelöletlen szomszédai, az utolsó iterációkban csak annyi történik, hogy ezt a két elemet eltávolítjuk a sorból, ami így kiürül. Ekkor a Vizsgált tömb értékei alapján láthatjuk, hogy mely csomópontok érhetőek el az 1-esből. A keresett 12-es nincs köztük, így az algoritmus kimenete hamis.

### 3.3.2. Mélységi bejárás

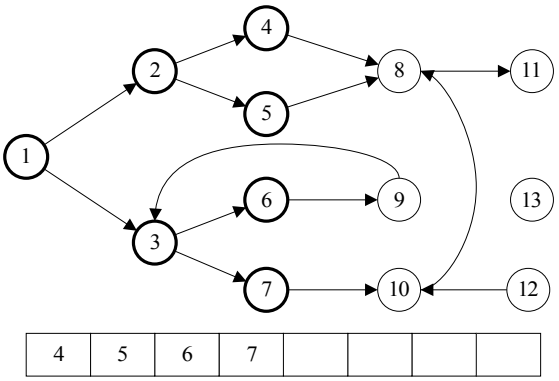
A mélységi bejárás nagyban hasonlít a szélességihez, az eltérés az, hogy nem sort, hanem vermet használunk hozzá. A verem olyan adatszerkezet, amely végére szúrhatjuk az új elemeket, viszont a sorral ellentétben nem az elejéről szedjük ki a következő elemet, hanem a végéről. Ezt LIFO elvnek is szokták mondani, azaz „utolsónak be, elsőnek ki” (last in, first out). Ez azt eredményezi, hogy ha beszúrjuk egy csomópont szomszédait a verembe, akkor a legutoljára beszúrt szomszéddal megyünk tovább. Ennek eredményeként lesznek a kezdő csomóponthoz közel és tőle távol levő csomópontok is, amelyeket megvizsgálunk. Az algoritmus pszeudokódja a 3.30 ábrán látható.

Lássuk, hogy a 3.20 ábrán látható gráfot hogyan járja be az algoritmus. Kezdetben itt is az 1-es csomópontot szúrjuk be (3.31). Az első iterációban 1-es csúcsot kivesszük és beszúrjuk

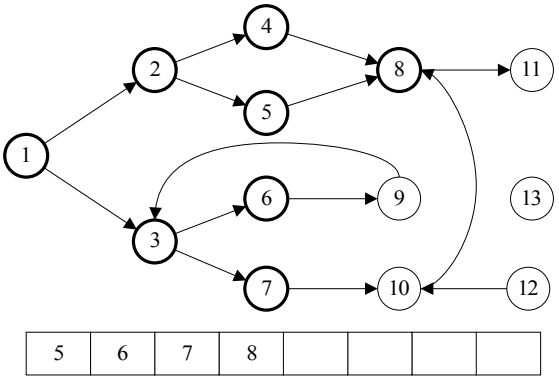



*3.22. ábra. A második iteráció után*



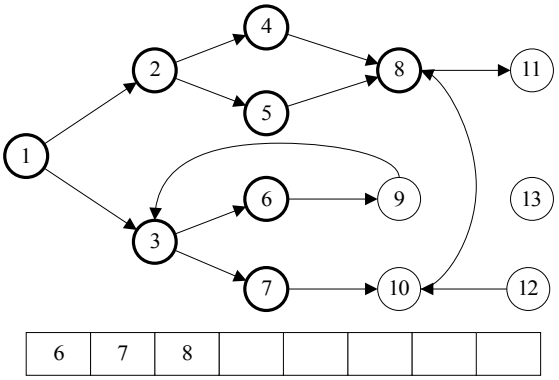

*3.23. ábra. A harmadik iteráció után*



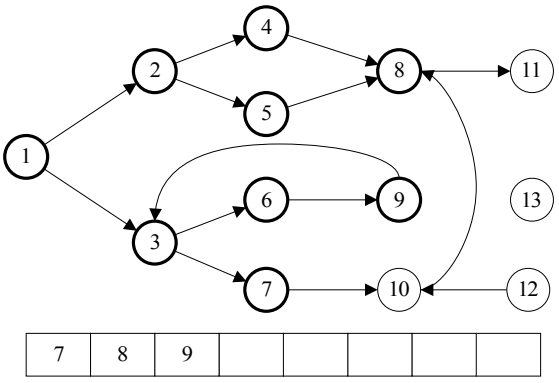

*3.24. ábra. Az negyedik iteráció után*

a 2-est és 3-ast (3.32 ábra). Ezután a 3-as van a verem tetején, ezért a 3-ast lecseréljük a szomszédaira (3.33). Ezután a 7-es csomópont következik (3.34 ábra), majd a 10-es (3.35 ábra).

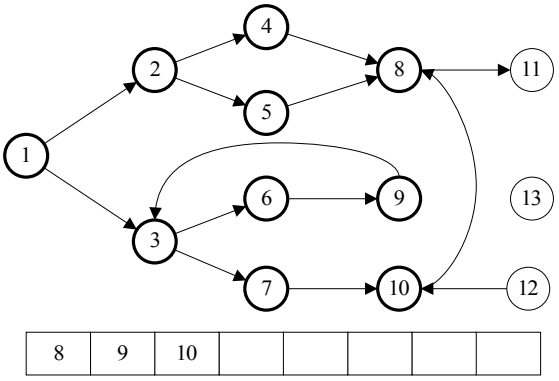
Láthatjuk, hogy a mélységi bejárás mindig csak a gráf egy bizonyos tartományára koncentrál, egy adott irányba halad tovább, ameddig csak lehet. A 10-es után következik a 8-as csomópont (3.36 ábra). Majd elérkezünk a 11-eshez (3.37 ábra).




*3.25. ábra. Az ötödik iteráció után*



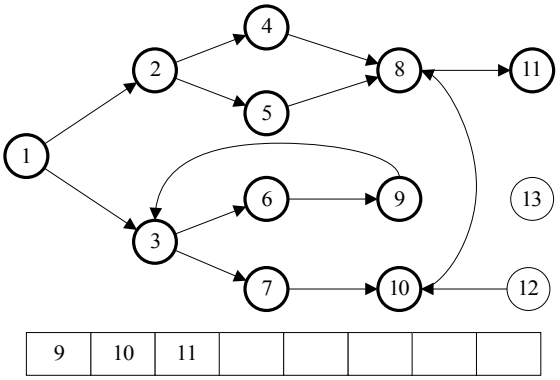

*3.26. ábra. A hatodik iteráció után*




*3.27. ábra. A hetedik iteráció után*

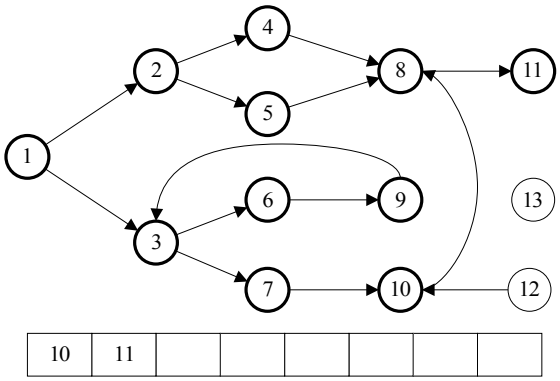
A 11-esnek nincs több szomszédja, tehát innen már nem tudunk tovább haladni, a bejárt út mentén visszalépünk a legközelebbi olyan csomópontra, ahonnan megpróbálhatunk tovább haladni, ez lesz a 6-os csomópont (3.38 ábra).

A 9-es szomszédja a 3-as, ám hasonlóan az előző algoritmushoz, a 3-ast megint mellőzzük. Ezek után csak a 2-es csomópont lesz a veremben. Ennek a szomszédait is beillesztjük (3.38 ábra).




*3.28. ábra. A nyolcadik iteráció után*




*3.29. ábra. A kilencedik iteráció után*

**bemenet:** $G(V, A)$ súlyozott irányított gráf, $s, t \in V$ csúcsok
**kimenet:** Elérhető-e a $t$ csúcs $s$-ből irányított úton

```
Elérhető:=hamis;
Ciklus minden v ∈ V \ {s}-re
    Vizsgált[v] :=hamis;
Ciklus vége
Verembe(s);
Vizsgált[s] :=igaz;
Ciklus amíg a verem nem üres és nem Elérhető
    v :=Veremből;
    Ciklus minden (v, u) ∈ A-ra, amíg nem Elérhető
        Ha u = t akkor
            Elérhető:= igaz;
        Különben
            Ha nem Vizsgált[u] akkor
                Verembe(u);
                Vizsgált[u] :=igaz;
            Elágazás vége
        Elágazás vége
    Ciklus vége
Ciklus vége
Kiír Elérhető;
```

*3.30. ábra. Mélységi bejárás*

Az 5-ösnek és a 4-esnek sincs már olyan szomszédja, amit még nem jelöltünk meg, így ez a két elem anélkül kerül veremből, hogy újakat szúrnánk be. A verem kiürül, így az algoritmus megáll, az eredmény ugyanaz, mint a szélességi bejárásnál.

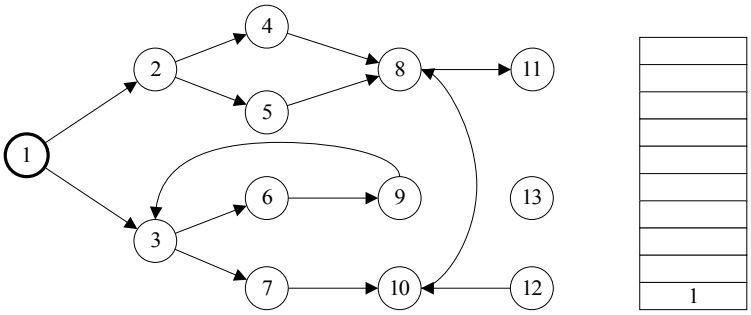
Mikor melyik bejárást érdemes alkalmazni? Abban az esetben, ha a gráf nem túl „mély”, azaz a kezdő csomóponttól a legtávolabbi elem is elég közel van, továbbá a gráf sok elágazást

tartalmaz, úgy a mélységi keresés memóriaigénye gyakran nagyságrendekkel kisebb, mint a szélességi keresésé, hiszen a veremben csak kevés elemet kell tárolni. Ezzel szemben a sok elágazást tartalmazó gráfnál a szélességi bejárásnál a sorban tárolt csomópontok száma akár exponenciálisan is növekedhet, ami könnyen memória problémákhoz vezethet. Ennek főleg nagy kiterjedésű fák bejárásakor van szerepe. A mélységi keresés alkalmazhatatlan lehet akkor, ha a gráf mélységét gyakorlatilag végtelen nagyságúnak tekinthetjük. Továbbá, előfordulhat, hogy a mélységi keresés feleslegesen tölt el sok időt a gráf egy olyan tartományában, ahol nem található meg a keresett csomópont.

## 3.4. Maximális folyam

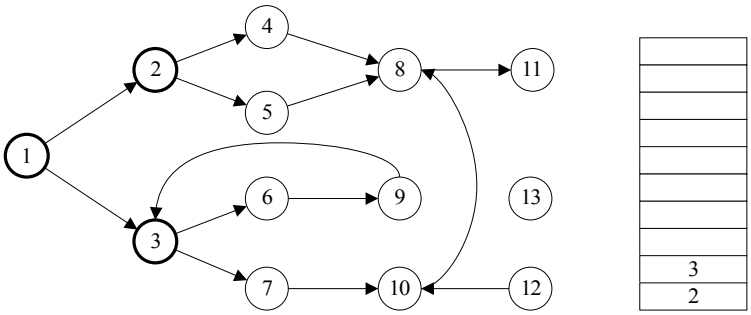
Rendszerek elemzésekor találkozhatunk olyan kérdéssel, amely megválaszolásához a rendszer egészének képességei együtt és egyszerre számítanak. Egyik legegyszerűbb ilyen kérdés egy úthálózat vagy csőhálózat áteresztő képessége vagy egy gyár maximális kihozatala bizonyos termékekből. Ilyenkor az indulástól a célig egyszerre több úton is eljuthatunk és mindezek kapacitása egyszerre számít a megoldásban. Ha rendszert gráfon modellezzük, akkor a gráfelmélet fogalmaival megfogalmazva egy maximális folyamot keresünk.

Tekintsünk egy irányított $G=(V, A)$ gráfot, amely egy szállító hálózatot reprezentál. A gráf egyik csomópontja termelőként, egy másik pedig fogyasztóként működik. A gráf élein keresztül lehet szállítani anyagokat. Minden élnek ismerjük a kapacitását, vagyis azt, hogy egy időegység alatt mennyi egység anyagot lehet szállítani. A kapacitást egy $c : A \to \mathbb{R}^+$ függvénnyel adjuk meg. Azon élekre, amelyek nem részei a gráfnak, a kapacitást 0-nak tekintjük. A nem termelő és nem fogyasztó csomópontokra érvényes az anyagmegmaradási törvény. Vagyis, ezekben a csomópontokban nem termelődik és nem fogy anyag, valamint itt nem is tárolunk semmit. Egy folyam $f : A \to \mathbb{R}$ függvény minden élre megadja, hogy az adott élen mennyi anyagot szállítunk éppen. Ennek az értéke nem haladhatja meg az élre előírt kapacitáskorlátot, tehát $f(a) \leq c(a)$ minden $a \in A$-ra. Fontos, hogy érvényesülnie kell a ferde szimmetria szabálynak, miszerint $f(v, u) = -f(u, v)$, azaz ha $u$-ból $v$-be adott a folyam értéke, akkor visszafelé a folyam értéke ennek ellentettje. Ha $n$ egységnyi anyagot szállítunk $u$-ból $v$-be és ugyanezt az anyagmennyiséget a másik irányba is szállítjuk, akkor a folyam értéke $u$ és $v$ között lenullázódik, hiszen végeredményeben ugyanazt kapjuk ha nem szállítunk a kettő között semmit. Maximális folyamról beszélünk akkor, ha a termelőből a lehető legnagyobb anyag-

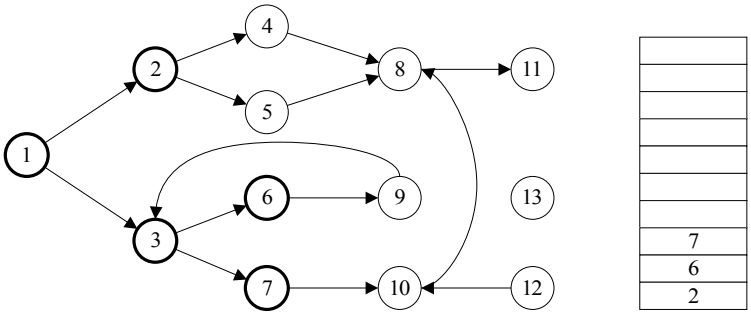



*3.31. ábra. Kezdetben az 1-es csúcsot jelöljük meg*

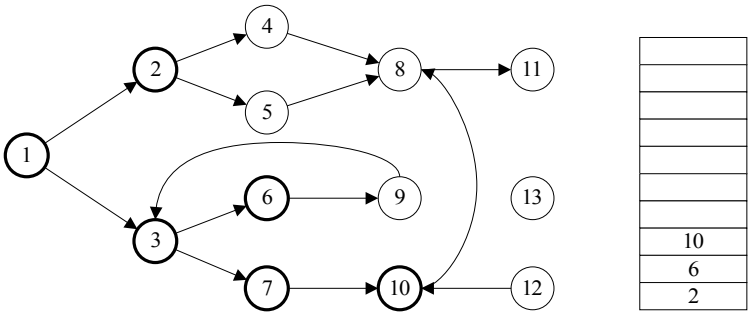
 



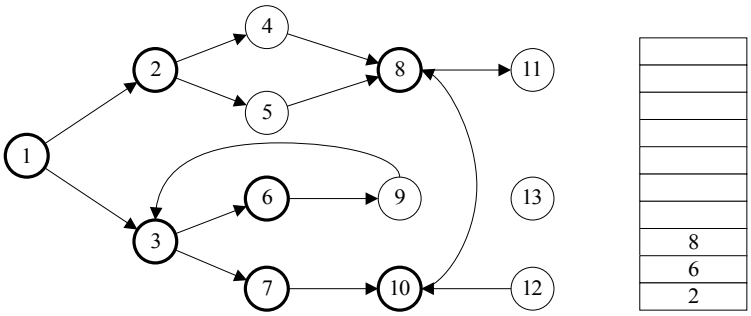

*3.32. ábra. Az első iteráció után*




*3.33. ábra. A második iteráció után*

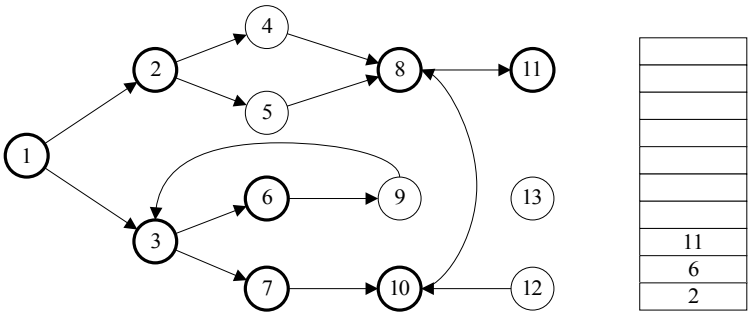



*3.34. ábra. A harmadik iteráció után*

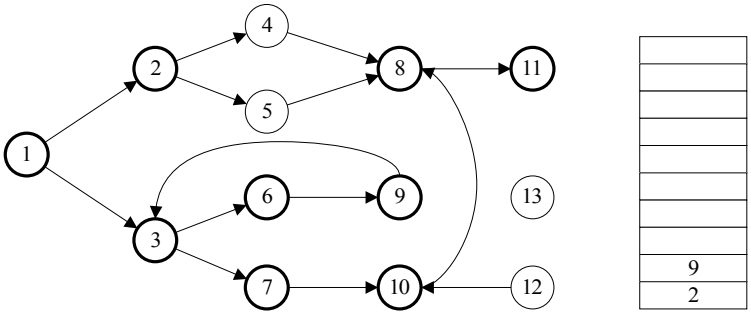



*3.35. ábra. A negyedik iteráció után*

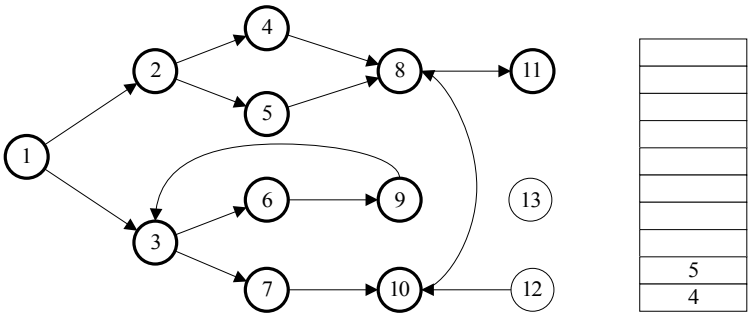
mennyiséget szállítjuk a fogyasztóba a hálózaton úgy, hogy teljesül az anyagmegmaradás, és a kapacitási korlátozásokat sem sértjük meg.



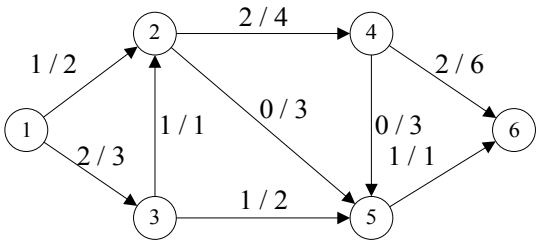

*3.36. ábra. Az ötödik iteráció után*




*3.37. ábra. A hatodik iteráció után*




*3.38. ábra. A hetedik iteráció után*

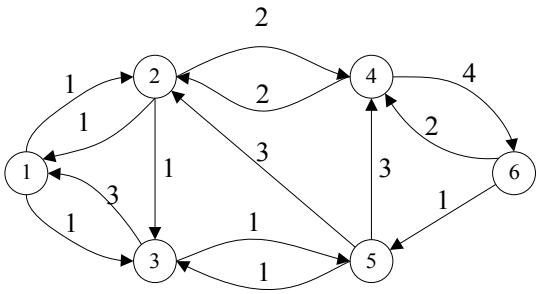



*3.39. ábra. Hálózat kapacitásokkal és folyam értékekkel*

Mielőtt rátérnénk a probléma megoldására, szükségünk lesz a reziduális hálózat fogalmára. Tegyük fel, hogy már beállítottunk néhány folyam értéket 0-nál nagyobbra. Egy adott $(u, v)$ élre a reziduális kapacitás a következő: $c_f = c(u, v) - f(u, v)$, azaz egy olyan érték, amely

megadja, hogy adott folyam esetén legfeljebb mennyivel növelhetjük a folyamot úgy, hogy a kapacitást ne haladja meg. A reziduális hálózat olyan gráf, amely tartalmazza az eredeti összes csúcspontját, valamint olyan irányított éleket, amelyekre a reziduális kapacitás nagyobb mint 0. A reziduális hálózat tartalmazhat olyan éleket is, amelyeket az eredeti gráf nem. Például, ha az eredeti gráf csak az $(u, v)$ élt tartalmazza, amelyre adott egy $f(u, v)$ pozitív folyam érték, akkor $f(vu)$ ennek ellentettje. Ekkor a reziduális kapacitás a $(v, u)$ irányban megegyezik az él kapacitásával, valamint a folyam értékének abszolút értékével. Például, ha az $u$ csomópontból a $v$-be szállítunk 5 egységnyi anyagot, a kapacitás pedig 20, akkor a reziduális kapacitás $u$-ból $v$-be 205, azaz 15. $v$-ből $u$-ba viszont a folyam értékét $-5$-nek tekintjük, ugyanebben az irányban a reziduális kapacitás $20-(-5)$, azaz 25. Ez azt jelenti, hogy ha az ellenkező irányba szállítunk először 5 egységnyi anyagot, akkor megszüntetjük az anyagáramlást, majd a mennyiséget növelve megindul a szállítás $v$-ből $u$-ba. A 3.39 ábrán láthatunk példát egy hálózatra, ahol az éleken az első szám a folyam, a második pedig a kapacitás értékét adja meg. A 3.40 ábrán látható a reziduális hálózat.

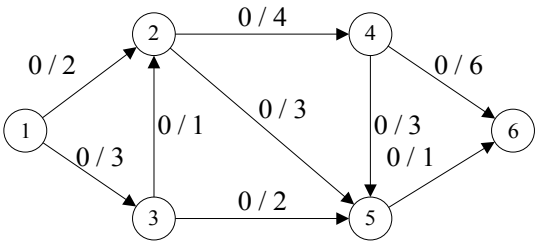



*3.40. ábra. Az előző hálózat reziduális hálózata*

**Bemenet:** G = (V, A, c) súlyozott irányított gráf, $c : A \to \mathbb{R}^+$, $s, t \in V$
**Kimenet:** $f : A \to \mathbb{R}$ az $s$ és $t$ közti maximális folyamban
az éleken áthaladó mennyiség

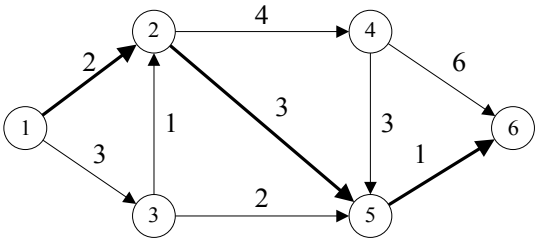
**Ciklus** minden $u, v \in V$-re
f(u, v) := 0;
**Ciklus vége**
**Ciklus** amíg van $(V^*, A^*)$ út $s$-ből $t$-be, ahol $c(u, v) - f(u, v) > 0$
minden $(u, v) \in A^*$-ra
$C_{fmin} := min\{c(u, v) - f(u, v)$ minden $(u, v) \in A^*$-re$\}$;
**Ciklus** minden $(u, v) \in A^*$-ra
$f(u, v) := f(u, v) + C_{fmin}$;
$f(v, u) := f(v, u) - C_{fmin}$;
**Ciklus vége**
**Ciklus vége**

*3.41. ábra. A maximális folyamot meghatározó algoritmus pszeoudokódja*



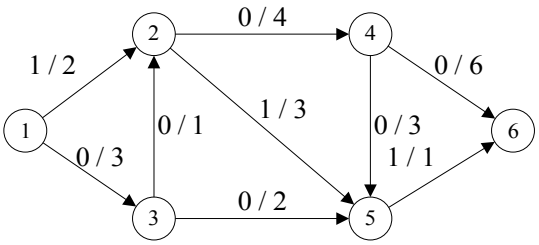

*3.42. ábra. A hálózat az első iteráció előtt*

Látható, hogy ahol a folyam értéke nullától különböző, de kisebb mint a kapacitás, ott a két csomópont között két reziduális él található. Az egyik reziduális él súlya a kapacitás és a folyam különbsége, a másik pedig a folyam értékével egyenlő, mivel az egyik irányban a kapacitást 0-nak tekintjük.



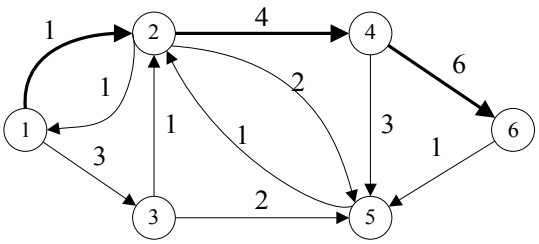

*3.43. ábra. Az induló hálózat reziduális hálózata*

A maximális folyam meghatározására a 3.41 ábrán látható Ford-Fulkerson algoritmust használjuk. A következő módon működik. Kezdetben minden folyam értékét 0-ra állítjuk. A reziduális hálózaton keresünk egy utat a termelőből a gyártóba. Erre használhatjuk valamelyik korábbi gráf bejáró algoritmust. Amennyiben nem találunk utat, az algoritmus véget ér. A kapott úton megvizsgáljuk, hogy melyik utat alkotó élen a legkisebb a reziduális kapacitás, majd az út minden élén ennyivel növeljük a folyamok értékét. Tehát meg kell vizsgálni, hogy lehet-e még a gyártóból valamelyik irányba plusz anyagmennyiséget továbbítani úgy, hogy az anyagmegmaradás ne sérüljön, vagyis ezt a plusz mennyiséget is el lehessen juttatni a fogyasztóba. A meghatározott úton azért a legkisebb reziduális kapacitású élt kell megkeresnünk, mert ha ennél nagyobb mértékben növelnénk a szállítandó anyagmennyiséget az út mentén, akkor legalább egy élnél megsértenénk a kapacitáskorlátot. A keresett utat javító útnak nevezzük. A folyamot addig tudjuk növelni, amíg van javító út.




*3.44. ábra. A hálózat az első iteráció után*

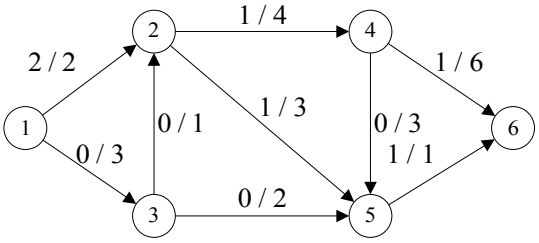
 




*3.45. ábra. A reziduális hálózat az első iteráció után*

A 3.39 ábrán szereplő gráfon bemutatjuk az algoritmus működését, ahol az 1-es csomópont a termelő, a 6-os pedig a fogyasztó. Kezdetben minden folyam értéket 0-ra állítunk (3.42 ábra).

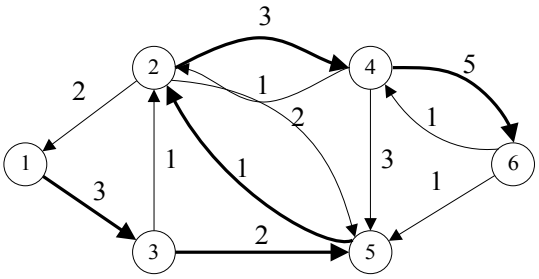
Keressünk javító utat a reziduális hálózaton. A talált utat vastag élekkel jelöljük (3.43 ábra). Itt az (5, 6) él kapacitása a szűk keresztmetszet, ezért az (1,2), (2,5) és (5,6) éleken egyaránt 1-re állítjuk a folyamot.



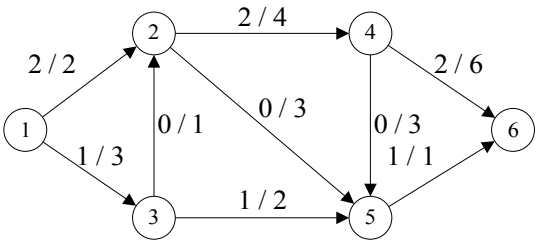

*3.46. ábra. A hálózat a második iteráció után*

Az első iteráció után kialakuló hálózaton (3.44 ábra) az előbbi javító utat nem kaphatjuk meg újra, hiszen az a hozzá reziduális hálózatban (3.45 ábra) az (5, 6) él már nem szerepel. A 3.45 ábrán látható második javító úton szintén 1 a szűk keresztmetszet. Miután növeltük a megfelelő folyam értékeket, látható, hogy az (1,2) élen telítődött a folyam (3.46 ábra), a következő javító út már nem haladhat ezen az élen (3.47 ábra).

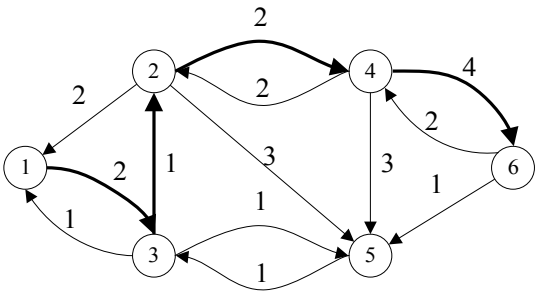
A 3.47 ábrán levő javító út áthalad az (5, 2) reziduális élen is, így annak a folyam értékét növelni kell. Ez azt eredményezi, hogy a (2,5) él folyam értéke 0 lesz (3.48 ábra), azaz itt nem áramlik anyag. Figyeljük meg, hogy ennek hatására a 2-es csomópontban az anyag nem válik szét, hanem a teljes mennyiség halad tovább a 4-es felé.




*3.47. ábra. A reziduális hálózat a második iteráció után*

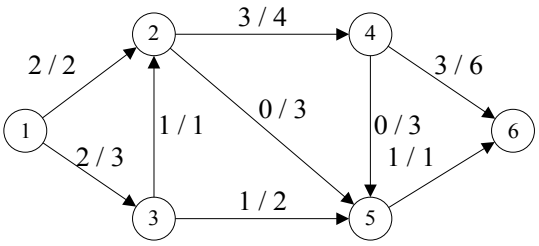



*3.48. ábra. A hálózat a harmadik iteráció után*

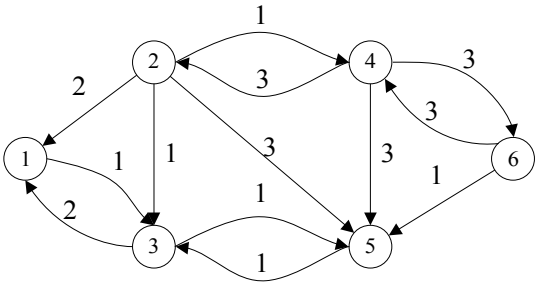



*3.49. ábra. A reziduális hálózat a harmadik iteráció után*

A következő javító út (3.49 ábra) szintén 1-el növeli néhány élen a folyamot (3.50). A reziduális hálózat a következő a 3.51 ábrán látható.



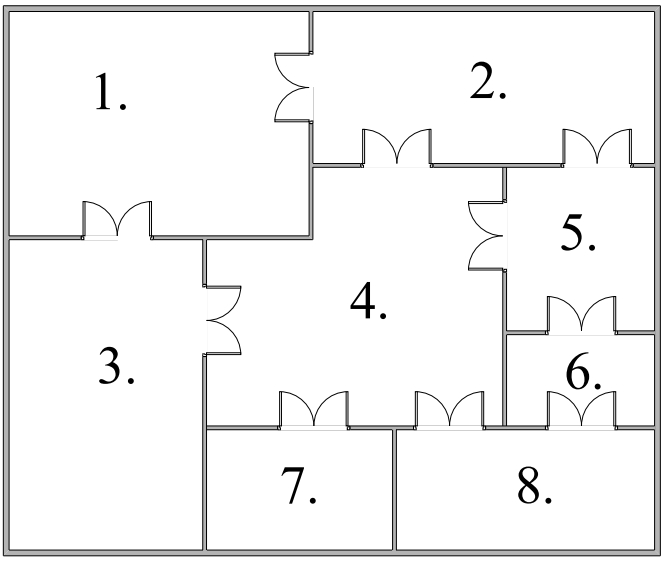

*3.50. ábra. A hálózat a negyedik iteráció után*




*3.51. ábra. A reziduális hálózat a negyedik iteráció után*

Ezen a reziduális hálózaton már nem létezik javító út 1-ből 6-ba, ezért az algoritmus megáll. A hálózaton legfeljebb 4 egységnyi anyagot lehet szállítani egy időegység alatt: A forrásból ennyi indul ki, valamint a fogyasztóba is ennyi érkezik összesen.

## 3.5. Feladatok




*3.52. ábra. Az 1. feladatban szereplő gyár alaprajza*

**1. feladat** A 3.52 ábra egy gyár alaprajzát tartalmazza. A gyár üzem épülete 8 teremből áll, ezek között átjárók találhatóak. Az egyes termek között targoncával szállítják az anyagokat, termékeket, szerszámokat. Minden helyiségre meghatároztak egy sebesség korlátozást, amelyet a targoncák vezetőinek be kell tartani (3.53 ábra).

*3.53. ábra. Az 1. feladathoz tartozó sebességkorlátok*

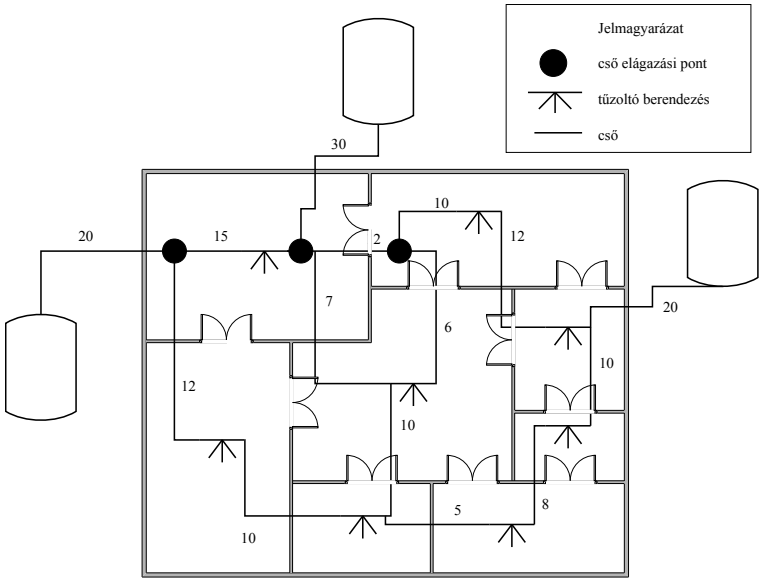
| Sorszám | Helyiség | Maximális megengedett sebesség |
|---|---|---|
| 1. | anyag raktár | 10 km/h |
| 2. | gépház | 6 km/h |
| 3. | tisztító csarnok | 12 km/h |
| 4. | fém megmunkáló csarnok | 3 km/h |
| 5. | minőség ellenőrző csarnok | 4 km/h |
| 6. | étkező | 2 km/h |
| 7. | szerszám raktár | 5 km/h |
| 8. | szerelő csarnok | 5 km/h |

Az egyes termekben a targoncák számára olyan áthaladási útvonalakat jelöltek ki, hogy minden teremben ugyanakkora távolságot kell megtenni, ha egyik helyiségből kell átmenni a másikba. Ezeken az útvonalakon mindkét irányba lehet haladni.

(a) Rajzoljon fel egy gráfot, amellyel modellezhető a targoncák által bejárható úthálózat!

(b) Határozza meg, hogy mely útvonalon érdemes haladni a targoncának, ha a raktárból kell szállítani a szerelő üzembe úgy, hogy ezt az utat a legrövidebb idő alatt tegye meg! Milyen algoritmussal oldható meg a feladat? Írja le az algoritmus lépéseit az a) feladatban megrajzolt gráfra, majd a feladatot oldja meg számítógép segítségével is!

(c) Az új üzemvezető úgy döntött, hogy ezentúl az anyag raktárba csak a tisztító üzemen keresztül lehet belépni, és a raktárt pedig csak a gépházon keresztül lehet elhagyni. Módosule az előző feladatban megtervezett legrövidebb út, ha igen, akkor mennyi idővel nő az út megtétele?




*3.54. ábra. A 2. feladatban szereplő üzemek*

**2. feladat** Az üzemben minden helyiséget el kell látni tűzoltó rendszerrel, amely úgy működik, hogy tűz esetén egy szelep kinyílik a plafonon, és vizet szór a terembe. Minden terem mennyezete fölött található egy ilyen tűzoltó berendezés, ám ezekhez cső vezetékeken keresztül el kell juttatni a vizet. A cső hálózat az alábbi 3.54 ábrán látható, ahol feltüntettük az épületen kívül elhelyezkedő három víz tartályt is.

Minden cső szakasz más és más átmérőjű, így az egyes szakaszokon egységnyi idő alatt különböző mennyiségű víz képes egyszerre áramlani, az ábrán a cső szakaszok mellé ezeket az értékeket jelenítettük meg.

(a) Határozza meg, hogy az ábrán látható rendszerben melyik terembe hány liter vizet lehet szállítani egy időegység alatt. Melyik terem ellátottsága a legjobb és melyiké a legrosszabb?

(b) Úgy ítélték meg, hogy a tűzvész előfordulásának valószínűsége csekély, ha előfordul, akkor is csak kis tűz keletkezhet. Emiatt elhatározták, hogy bizonyos csőszakaszokat megszüntetnek. Olyan csöveket szeretnének meghagyni, hogy minden helyiségbe lehessen

vizet szállítani, továbbá minél nagyobb kapacitású csöveket szeretnének kiszerelni, mert azokat beolvasztva hasznosítani szeretnék. Határozza meg, hogy mely csöveket érdemes leszerelni!

# 4. fejezet

# Lineáris programozás

Az alábbi szakaszban ismertetjük a lineáris programozás módszerét, majd kitérünk a legfontosabb feladatokra és azok megoldására. A lineáris programozás az optimalizálási technikák egyik alapvető eszköze. Történetének kezdete a második világháborúra nyúlik vissza, mikor felmerült az igény egy olyan matematikai módszer kidolgozására, amely segítségével komplex döntési problémákat tudtak megoldani. A háború után felgyorsultak a kutatások, mivel az iparban is nagy érdeklődést mutattak az elmélet iránt. 1947-ben George B. Dantzig kidolgozta az úgynevezett szimplex módszert, amely segítségével megoldhatóak a lineáris programozási feladatok. Az elmélethez ugyanabban az évben Neumann János is nagyban hozzájárult.

Ez a fejezet csupán betekintést nyújt a lineáris programozásba, a téma hazai és külföldi irodalma rendkívül gazdag. A téma iránt mélyebben érdeklődő Olvasóknak az alábbi könyveket ajánljuk: [15], [2] és [13].

## 4.1. Modellezés és formalizálás

Bevezetésként tekintsük az alábbi feladatot. Egy fiktív pizzériát üzemeltetünk, ahol kétfajta pizzát készíthetünk el, margarétát és hawaiit. Tudjuk, hogy akármennyi pizzát elkészítünk a nap elején, az estére mind elfogy, tehát nem marad felesleg. A margarétát 600, a hawaii-t 800 forintért áruljuk. A nap végére a lehető legnagyobb bevételre szeretnénk szert tenni. Döntési szabadságunk pedig abban áll, hogy melyik pizzából mennyit készítünk. Ha csak a bevétel érdekel minket, nem pedig a profit, akkor a nyersanyagok árától, a munkadíjaktól és rezsi költségektől eltekinthetünk. Az egyszerűség kedvéért az is megengedett, hogy ne csak egész pizzát gyártsunk.

Az eddigi feltételek alapján az a helyes döntés, hogy annyi hawaiit állítunk elő, amennyit csak lehetséges. Ám figyelembe kell vennünk bizonyos megszorításokat. A feltételekhez felhasználható nyersanyagok mennyisége korlátozott és vannak olyan hozzávalók, amelyek több pizzára is jók. A 4.1 táblázatban összefoglaltuk, hogy melyik pizzához melyik alapanyagból mennyi szükséges.

Egy nap sajtból 550, sonkából 150, ananászból pedig 120 adag áll rendelkezésre. Ezen feltételek mellett pedig a megoldás már nem triviális, megfelelő megoldó módszert kell választanunk. Az említett lineáris programozás alkalmas ilyen jellegű feladatok megoldására. A módszer két fő lépésből áll. Először felírunk egy modellt, másodszor megoldjuk azt. Ebben

*4.1. táblázat. A feladat paraméterei*

| | margaréta | hawaii |
|---|---|---|
| sajt | 10 | 5 |
| sonka | 2 | 4 |
| ananász | 0 | 3 |

a szakaszban a modell felírására koncentrálunk. Mivel a kurzus során a megoldó módszerek belső működésének mély ismeretére nincs szükség, ez utóbbiaknak csupán a lényegét vázoljuk fel. Lássuk tehát, hogy hogyan kell felírni a modellt.

Azt szeretnénk tudni, hogy melyik pizzából mennyit kell előállítanunk, vagyis a kérdésünkre két mennyiség lesz a válasz. Vezessünk be tehát két változót, amelyek a megoldást fogják tárolni. Jelölje $x_1$ az előállítandó margaréta mennyiségét, $x_2$ pedig az előállítandó hawaii mennyiségét.

Minden változóhoz meg kell adni, hogy milyen értékeket vehetnek fel. Negatív mennyiségű pizzát nem gyárthatunk, így a változók legalább 0-t kell hogy felvegyenek.

$$x_i \geq 0, \quad i = 1,2 \tag{4.1}$$

A modell a lineáris kifejezéseket, egyenlőtlenségeket tartalmaz. Egy lineáris kifejezéssel le kell írnunk, hogy a bevételt szeretnénk maximalizálni.

$$\max \quad 600x_1 + 800x_2 \tag{4.2}$$

Amennyiben minimalizálni szeretnénk, úgy max helyett min-t írunk. Ezt a kifejezést célfüggvénynek nevezzük. Célunk az, hogy a változók értékeit úgy határozzuk meg, hogy a célfüggvény értéke maximális legyen. Az alapanyagokra vonatkozó megszorításokat egyenlőtlenségekkel tudjuk megadni.

$$10x_1 + 5x_2 \leq 550 \tag{4.3}$$
$$2x_1 + 4x_2 \leq 150 \tag{4.4}$$
$$3x_2 \leq 120 \tag{4.5}$$

Ezeket az egyenlőtlenségeket korlátozásoknak hívjuk.

A fenti modell lineáris, mert a benne szereplő egyenlőtlenségekben és a célfüggvényben a változók súlyozott összegei szerepelnek. Egy korlátozás jobb és bal oldala között szerepelhet még = és $\geq$ is. Fontos, hogy a $<$ vagy $>$ nem megengedett. Ennek oka, hogy nem tudunk nyitott halmazban szélsőértéket keresni. Nem tudjuk megválaszolni például, hogy melyik az a legkisebb szám, mely határozottan nagyobb mint egy.

Miután felírtuk a modellt, azt meg kell oldani. A könnyebb tárgyalhatóság kedvéért az $x_i$ változókat gyűjtsük egy $\mathbf{x} = (x_1, x_2)$ vektorba. A modell egy megoldása olyan $\mathbf{x}$ vektor, amely kielégíti a korlátozásokat. Megengedett megoldásnak nevezzük azt a megoldást, amely eleget tesz a változókra vonatkozó megkötéseknek is. Maximalizálás esetén optimális megoldásnak

nevezzük azt a megengedett megoldást, ahol nincs olyan **x'** vektor, amely szintén megengedett megoldást reprezentál, és amelyhez tartozó célfüggvény értéke nagyobb lenne. Minimalizálás esetén természetesen akkor találtunk optimális megoldást, ha nincs olyan megengedett **x'** vektor, amelyre a célfüggvény értéke kisebb lenne. Felhívjuk a figyelmet arra, hogy az optimális már önmagában egy lehető legjobb megoldást jelöl, tehát a hétköznapi életben gyakran elhangzó „optimálisabb" és „legoptimálisabb" kifejezés értelmetlen, ezért óvakodjunk a használatuktól.

A lineáris korlátozásokkal és célfüggvénnyel adott optimalizálási feladat felírását és megoldását **lineáris programozás**nak vagy **LP**-nek nevezzük, magát a feladat ezen formáját pedig **lineáris programozási modellnek** vagy **LP modell**nek hívjuk.

Vegyük észre, hogy az optimális megoldás definíciója megengedi több optimális megoldás létezését is, ekkor azt mondjuk, hogy a modellnek alternatív megoldása is van. Ez utóbbiak nagy jelentősége van akkor, mikor az optimalizáló szakember a döntéshozóknak bemutatja az általa meghatározott optimális megoldást.

Gyakran előfordul, hogy a kapott megoldás gyakorlati szempontból nem megfelelő, mert a megbízó nem közölt minden szempontot, mely szerint egy megoldás megengedett, tehát hiányzik korlátozás a matematikai modellből. Ekkor be lehet mutatni az alternatív megoldásokat. Amennyiben ezek sem megfelelőek, akkor az eddig nem ismert feltételeket is figyelembe kell venni a modellben, majd azt újra meg kell oldani.

Lineáris programozási modelleket több módszerrel is megoldhatunk. A korábban említett szimplex módszert publikálták először, de létezik ezen kívül más elven működő, például az úgynevezett belső pontos módszerek is. Ezen módszerek gyakorlati alkalmazhatósága különböző lehet. Például a szimplex típusú módszerekkel könnyebben találhatunk alternatív megoldásokat. Ugyanakkor, a belsőpontos módszerek számítógépes implementációi gyakran nagyobb méretű feladatok megoldására alkalmasak, mert kevésbé érzékenyek a számábrázolási pontosságra.

A két módszercsoport bemutatásához ábrázoljuk a három egyenlőtlenséget egy koordináta rendszerben, ahol a vízszintes tengelyen az $x_1$, a függőlegesen pedig az $x_2$ változót helyezzük el. Szürkével jelöljük azt a tartományt, amely mindhárom egyenlőtlenséget kielégíti. Az optimális megoldás biztosan nem az alakzatunk belsejében található, hanem a kijelölt tartomány valamely sarkán, vagy oldalán kell keresnünk. Az egyenlőtlenségekkel meghatározott konvex tartományt szimplexnek nevezzük. A korábban említett szimplex módszer a tartomány sarkait vizsgálva keresi az optimális megoldást, ha talál ilyent, akkor nem megy tovább a többi sarokra.

A belső pontos módszerek nem a szimplex szélén, hanem a belsejében haladva közelítik meg az egyik élen vagy sarkon található optimális megoldást. Ha több optimális megoldás is van, akkor gyakran, az azonosan jó megoldásokat tartalmazó felület egy belső pontját eredményezik.

A fenti algoritmusok részletes tárgyalása más kurzusok témakörébe tartozik. Az algoritmusokat számos szoftverben implementálták, így működésül mély ismerete nélkül is megoldhatunk lineáris programozás feladatokat. Mivel gyakori, hogy egy LP feladat csupán egy bonyolultabb probléma részfeladata, ezért léteznek erre a célra írt függvénykönyvtárak, valamint ezek köré írt megoldó szoftverek is. Ezek között létezik előfizetéses és ingyenes is. Jelen jegyzet írásakor az egyik legelismertebb és legjobb teljesítményű megoldó az IBM ILOG

CPLEX szoftver, mely használatáért piaci környezetben fizetni kell. Az ingyenes változatok között a két legelterjedtebb a Coin-OR és a glpk. Ezek teljesítménye szerényebb mint a CPLEX-é, ám gyakran elegendő a vizsgált feladatok megoldására. A következő szakaszban bemutatjuk, hogy kell megadni és megoldani egy LP feladatot a glpk szoftver segítségével.

## 4.2. Feladatmegoldás glpk-val

A glpk (GNU Linear Programming Kit) nyilt forráskódú, ingyenes szoftver. Több operációs rendszeren is elérhető, telepíthető Linux és Windows opereációs rendszerek alatt is.

A modellünket a glpk számára értelmezhető formában, egy erre a célra megalkotott modellezési nyelven kell megfogalmaznunk. A modellezési nyelvek segítségével könnyen generálhatunk nagy feladatokhoz helyes modellt, ahelyett, hogy a változókat és egyenlőtlenségeket külön-külön írnánk fel. Az első és legelterjedtebb nyelv a GAMS (General Algebraic Modeling System), amely a 70-es évek végén jelent meg. Sajnos a nyelv megalkotásakor a billentyűzetek nem rendelkeztek több, ma gyakran használt karakterrel, mint például a relációs jelek, ezért ma már nagyon barátságtalannak tűnik, ráadásul fizetni kell érte még akadémiai környezetben is. A kurzus során a GNU MathProg modellezési nyelvet fogjuk használni, amelyet a glpk szoftverhez adott egyik program is képes értelmezni.

A modellt egy szövegszerkesztő segítségével kell megírnunk, majd fájlba mentjük. A következő lépésben parancssorból el kell indítani a glpsol nevű programot, amelynek paraméterben meg kell adni az előbb elkészített fájl nevét, majd a program megoldja a feladatot.

A modellt felírhatjuk a korábban bemutatott módon is, azaz külön megfogalmazzuk a változókra vonatkozó feltételeket, egyenként leírjuk a korlátozásokat. A korábbi feladat modellje a GNU MathProg nyelvén megfogalmazva a 4.1 ábrán látható.

```
var x1 >= 0;
var x2 >= 0;

s.t. sajt: 10*x1 + 5*x2 <= 550;
s.t. sonka: 2*x1 + 4 * x2 <= 150;
s.t. ananasz: 3*x2 <= 120;

maximize koltseg: 600 * x1 + 800 * x2;
end;
```

*4.1. ábra. A feladat modellje GNU MathProg nyelven*

Egy GNU MathProg nyelvű leírás kifejezésekből áll, minden kifejezést pontosvesszővel zárunk. Több kifejezést írhatunk egy sorba is. Fontos, hogy a nyelv megkülönbözteti a kis és nagy betűket. A # karaktertől kezdve a sor végéig minden megjegyzésnek számít, így a bonyolultabb modelleket dokumentálhatjuk is.

Az első két sorban a **var** kulcsszóval létrehozzuk az $x_1$ és $x_2$ változót. Opcionálisan megadhatunk egy korlátot is, ebben az esetben a >= 0 kódrészlet azt jelenti, hogy a változók alsó korlátja 0 legyen. Megadhatunk még egyenlőséget és <= típusú korlátot is. Mivel másképp nem rendelkezünk, a változók folytonosak lesznek.

A következő lépés a korlátozások felsorolása. Egy korlátozást az **s.t.** Kulcsszóval kezdünk, majd ezt követi a korlátozás neve, ez után egy kettőspontot követően megadjuk a korlátozást. A korlátozásban használhatunk zárójeleket is, az egyenlőtlenséget rendezhetjük bármelyik konstansra, vagy változóra.

A célfüggvényt a **maximize** vagy **minimize** kulcsszóval adjuk meg, attól függően, hogy maximalizálni, vagy minimalizálni szeretnénk. Ahogy a korlátozásoknak, a célfüggvénynek is meg kell adni egy nevet, kettőspont után magát a célfüggvényt. Végül a leírást az **end** kulcsszóval jelezzük, hogy a leírásnak vége van, bármi amit az end után írunk, azt a program figyelmen kívül hagyja. Amennyiben az end-et nem íjuk le, az nem számít hibának.

A GNU MathProg nyelvű leírásokban különféle objektumokat kell megadnunk. Ilyen objektumok a már említett változók, korlátozások és a célfüggvény is, továbbá a későbbiekben találkozunk majd további objektumokkal is. Minden objektum rendelkezik egy egyedi névvel.

A fenti modellt mentsük el a pizza.txt nevű fájlba, majd parancssorból adjuk ki a következő parancsot:

```
glpsol -m pizza.txt -o megoldas.txt
```

Amennyiben elgépeltük a modellt, a beolvasás a hiba helyén megáll és megjelenik a képernyőn egy tájékoztató szöveg a hiba jellegéről. Miután a hibákat javítottuk, és sikeresen futtattuk a megoldót, a képernyőn a 4.2 látható információk jelennek meg:

```
GLPSOL: GLPK LP/MIP Solver 4.38
Reading model section from pizza.txt…
9 lines were read
Generating sajt…
Generating sonka…
Generating ananasz…
Generating koltseg…
Model has been successfully generated
glp_simplex: original LP has 4 rows, 2 columns, 7 non-zeros
glp_simplex: presolved LP has 2 rows, 2 columns, 4 non-zeros
Scaling…
 A: min|aij| =  2.000e+00  max|aij| =  1.000e+01  ratio =  5.000e+00
Problem data seem to be well scaled
Crashing…
Size of triangular part = 2
*     0: obj =   0.000000000e+00  infeas =  0.000e+00 (0)
*     2: obj =   3.966666667e+04  infeas =  0.000e+00 (0)
OPTIMAL SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.1 Mb (114490 bytes)
Writing basic solution to `kimenet.txt'…
```

*4.2. ábra. A képernyőn megjelenő információk*

Ebből a legfontosabb az a sor, amely az OPTIMAL SOLUTION FOUND szöveget tartalmazza, a program ezzel jelzi, hogy sikerült optimális megoldást találnia. Amennyiben a modellnek nincs megoldása, azt a program a PROBLEM HAS NO FEASIBLE SOLUTION üzenettel jelzi. Ez utóbbi akkor fordul elő, ha a korlátozások közül vannak olyanok, amelyek egymásnak ellent mondanak.

A glpsol a -o kapcsoló hatására az eredményt a megoldas.txt-be mentette. Ennek a fájlnak az eleje 4.3 ábrán látható:

```
Problem:    pizza
Rows:       4
Columns:    2
Non-zeros:  7
Status:     OPTIMAL
Objective:  koltseg = 39666.66667 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------   ------------   --   -------------   -------------   -------------
-------------
     1 sajt         NU          550                       550       26.6667
     2 sonka        NU          150                       150       166.667
     3 ananasz       B             40                             120
     4 koltseg       B        39666.7

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------   ------------   --   -------------   -------------   -------------
-------------
     1 x1            B        48.3333             0
     2 x2            B        13.3333             0
```

*4.3. ábra. A kimeneti fájl eleje*

Az Objective szó után találjuk a célfüggvény értékét az optimális megoldásban. Alul kiolvashatjuk az x1 és x2 változó értékét, azaz, hogy melyik pizzából mennyit kell előállítani. Az ismertetett feladat esetében akkor tehetünk szert maximális profitra, ha egy nap 48 és egy harmad margaréta, valamint 13 és egy harmad hawaii pizzát gyártunk, a bevétel ekkor 39 666767 Forint. A fejezet egy későbbi szakaszában kitérünk arra is, hogyan kell megoldani a feladatot akkor, ha csak egész pizzákat szeretnénk előállítani.

Az előzőekben láthattunk egy egyszerű GNU MathProg nyelvű leírást, ahol minden változóról és korlátozásról külön nyilatkoztunk. Ez hasonlóan kis méretű problémáknál nem okoz gondot, ám ha nagyságrendekkel több pizzát és hozzávalót kell figyelembe venni, akkor a modell ilyen módon való leírása fáradságos és időrabló folyamat, valamint megnő a hibák elkövetésének a valószínűsége. Továbbá vegyük észre, hogy ha a pizzás feladat méretét akármennyire növeljük, a modell logikai felépítése változatlan marad, csupán a változók és korlátozások száma, valamint az együtthatók változnak. Érdemes tehát a modell szerkezetének leírását és a konkrét adatokat külön választani. Ennek a megközelítésnek az előnye akkor mu-

tatkozik meg, ha az adatok változnak, például új pizza kerül az étlapra, megváltozik az egyik ára. Ekkor a modellt nem változtatjuk meg, csak a hozzá tartozó adatokat.

Használjunk most két külön fájlt, az egyikbe a modell szerkezetének leírását, a másikba az adatokat helyezzük el. Ehhez szükségünk lesz néhány új GNU MathProg objektumra.

A megoldandó problémában azonosítani kell a részt vevő objektumokat, ezek most a pizzák és a hozzávalók. Szükségünk lesz tehát két halmazra, amelyek ezeket fogják tárolni. A halmazok megadása a 4.4 ábrán látható. A halmazok tényleges tartalmát a 4.5 ábrán látható módon adhatjuk meg.

```
set Pizzak;
set Hozzavalok;
```

*4.4. ábra. Halmazok megadása*

```
set Pizzak:= margareta hawaii;
set Hozzavalok:= sajt sonka ananasz;
```

*4.5. ábra. A halmazok tartalmának megadása*

Az adat fájlban csak olyan halmazok, és paraméterek szerepelhetnek, amelyek a modell fájlban is megtalálhatóak. A halmaz elemeinek felsorolásánál nem szabad kapcsos zárójelet használni, valamint a vesszők elhagyhatóak. A halmaz elemei ebben az esetben szimbólumok lesznek, de számokat is írhatunk az elemek közé.

A pizzák rendelkeznek eladási árral, a hozzávalókhoz tartozik a naponta elhasználható maximális mennyiség, valamint a pizza-hozzávaló párokhoz hozzárendelhetünk egy mennyiséget, ami leírja, hogy az adott pizzából az adott hozzávalóból mennyit kell felhasználni. Ezeket az értékeket paraméter objektumokkal írhatjuk le. Egy paraméter egy adott halmaz adott eleméhez tartozik, tehát egy új paraméter objektum létrehozásakor meg kell adni azt a halmazt, amely elemeihez tartozó paramétereket szeretnénk megadni a 4.6 ábrán látható módon.

```
param PizzaAr{p in Pizzak};
param Mennyiseg{p in Pizzak, h in Hozzavalok};
param HozzavaloMax{h in Hozzavalok};
```

*4.6. ábra. A paraméterek létrehozása*

A PizzaAr paraméter objektum a Pizzak halmaz minden eleméhez hozzárendel egy tulajdonságot, vagy egy paramétert. A Mennyiseg pedig minden (p, h) párhoz rendel egy paramétert, ahol $p \in Pizzak$ és $h \in Hozzavalok$. A paraméterek tényleges értékeit az adat fájlban kell megadni, mégpedig a 4.7 ábrán látható módon.

A PizzaAr és HozzavaloMax esetében a paramétereket a következő módon adjuk meg: a param kulcsszó után következik a paraméter objektum neve, majd := után felsoroljuk a paramétereket. Mivel az előbbi két paraméter objektum egy-egy halmaz elemeihez rendel hozzá paramétereket, ezért a felsoroláskor leírjuk a halmaz egy elemét, majd az ahhoz tartozó paramétert, végül a felsorolást pontosvesszővel zárjuk. A Mennyiseg paraméter objektum esetében

```
param PizzaAr:= margareta 600
                hawaii    800;

param HozzavaloMax:= sajt    550
                     sonka   150
                     ananasz 120;

param Mennyiseg: sajt sonka ananasz:=
          margareta   10   2   0
          hawaii      5    4   3;
```

*4.7. ábra. A paraméterek megadása*

(sajt, margareta) → 10
(sajt, hawaii) → 5
(sonka, margareta) → 2
(sonka, hawaii) → 4
(ananasz, margareta) → 0
(ananasz, hawaii) → 3

*4.8. ábra. A példa hozzárendelései*

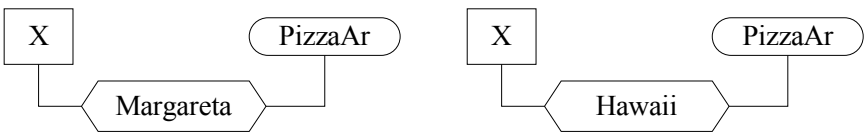
nem egy halmaz egy adott elméhez rendelünk hozzá paramétert, hanem két halmaz elemeiből alkotott rendezett kettesekhez. A példa hozzárendeléseit szemlélteti a 4.8 ábra.

A paraméter objektum neve után egy kettőspontot követően felsoroljuk a Hozzavalok halmaz elemeit. Ezzel jelezzük, hogy a paraméter értékek ezen halmaz megfelelő elemeihez tartoznak. Kettőspont és egyenlőségjel után megadunk egy elemet a Pizzak halmazból, majd következnek azok az értékek, amelyek ehhez a pizzához, és a megfelelő hozzávalóhoz tartoznak. Érdemes úgy tagolni a felsorolást, hogy az egy táblázathoz hasonlítson, így könnyen átláthatóvá válik.

A következő lépés a változók megadása. A modellünkben minden pizzához tartozik egy változó, ezt a következő módon fejezhetjük ki:

```
var x{Pizzak} >= 0;
```

Azaz, a Pizzak halmaz minden eleméhez tartozni fog egy x változó. Ezeket a változókat úgy különböztetjük meg egymástól, hogy az x után szögletes zárójelben feltüntetjük a Pizzak halmaz egyik elemét, például: x[margareta]. Ezt szemlélteti a 4.9 ábra.




*4.9. ábra. A Pizzak halmaz minden eleméhez hozzárendeltünk egy változót és egy paramétert*

Miután specifikáltuk az adatokat, meg kell adnunk a célfüggvényt és a korlátozásokat. A célfüggvény ebben a feladatban a gyártott pizzák mennyisége és eladási árai szorzatának összege:

```
maximize koltseg: sum{i in Pizzak} x[i] * PizzaAr[i];
```

A sum művelet a következő módon működik: A zárójelben megadott halmaz minden elemét behelyettesíti az i ideiglenes változóba, majd a sum utáni kifejezés kerül kiértékelésre. Ebben az esetben az i értéke először margareta lesz, majd hawaii. A fenti kifejezés a következővel ekvivalens:

```
maximize koltseg: x[margareta] * PizzaAr[margareta] +
x[hawaii] * PizzaAr[hawaii];
```

A korlátozásokat hasonló módon adhatjuk meg:

```
s.t. hozzavalo{h in Hozzavalok}: sum{p in Pizzak} x[p]
Mennyiseg[p, h] <= HozzavaloMax[h];
```

Az s.t. kulcsszó jelöli azt, hogy korlátozásokat adunk meg. A hozzavalo{h in Hozzavalok} kifejezés azt jelenti, hogy a Hozzavalok halmaz minden eleméhez generálunk egy-egy korlátozást. Az aktuális korlátozáshoz tartozó hozzávalót a h ideiglenes változó jelöli, amely a sum-hoz hasonlóan végighalad a Hozzavalok halmazon. A kifejezésben a h helyére mindig egy Hozzavalok halmaz-beli elemet helyettesít a megoldó program.

Végül, a modell és adat fájlt az end; kulcsszóval zárjuk le. A teljes modell fájl tartalma a 4.10 ábrán, az adat fájl tartalma pedig a 4.11 ábrán látható.

```
set Pizzak;
set Hozzavalok;
param PizzaAr{p in Pizzak};
param Mennyiseg{p in Pizzak, h in Hozzavalok};
param HozzavaloMax{h in Hozzavalok};

var x{Pizzak} >= 0;

s.t. hozzavalo{h in Hozzavalok}: sum{p in Pizzak} x[p] * Mennyiseg[p, h]
<= HozzavaloMax[h];

maximize koltseg: sum{i in Pizzak} x[i] * PizzaAr[i];

end;
```

*4.10. ábra. A teljes modell file tartalma*

Később, mikor az adatok módosulnak (megváltozik az egyik pizza ára, új pizza kerül az étlapra), elég csupán az adat fájlt módosítani. A glpsol-t most a következő módon kell indítani:

```
glpsol -m pizza_model.txt -d pizza_data.txt -o megoldas.txt
```

```
data;

set Pizzak:= margareta hawaii;
set Hozzavalok:= sajt sonka ananasz;

param PizzaAr:= margareta 600
                hawaii    800;

param Mennyiseg: sajt sonka ananasz:=
        margareta   10   2   0
        hawaii      5    4   3;

param HozzavaloMax:=  sajt    550
                      sonka   150
                      ananasz 120

end;
```

*4.11. ábra. Az adat fájl tartalma*

A -m kapcsoló után a modell fájl, a -d után az adat fájlt kell megadni. Egy sikeres megoldás utáni megoldas.txt tartalma látható a 4.12 ábrán. Láthatjuk, hogy az eredmény ugyanaz, mint

```
Problem:    pizza_model
Rows:       4
Columns:    2
Non-zeros:  7
Status:     OPTIMAL
Objective:  koltseg = 39666.66667 (MAXimum)

  No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------  ------------  --  -------------  -------------  -------------
-------------
     1 hozzavalo[sajt]
                   NU          550                       550       26.6667
     2 hozzavalo[sonka]
                   NU          150                       150       166.667
     3 hozzavalo[ananasz]
                    B           40                       120
     4 koltseg      B        39666.7

  No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------  ------------  --  -------------  -------------  -------------
-------------
     1 x[margareta] B        48.3333             0
     2 x[hawaii]    B        13.3333             0
```

*4.12. ábra. A megoldás fájl tartalma*

az egyszerűbb modellünk esetén, csupán a változók és a korlátozások nevei különböznek.

A GNU Mathprog ezen kívül még számos eszközt ad a kezünkbe ahhoz, hogy minél nagyobb szabadsággal rendelkezzünk a modellek felírásakor: Használhatunk halmazműveleteket (unio, metszet, stb.), létrehozhatunk új halmazokat, képernyőre írathatunk bizonyos információkat. Az érdeklődő Olvasónak ajánljuk, hogy tanulmányozza a GNU Mathprog dokumentációját.

## 4.3. Egészértékű változók

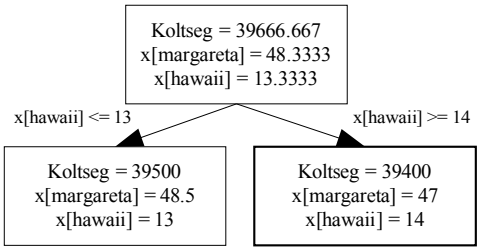
A bevezető feladatban megengedtük, hogy ne csak egész pizzát gyártsunk. Gyakoriak azonban az olyan feladatok, mikor szükség van arra, hogy bizonyos változók egész értéket vehessenek fel, például a pizzéria tulajdonosa úgy dönt, hogy csak egész pizzákat lehet sütni és eladni. Ez a megkötés nagyban bonyolítja a feladatot, a megoldó programok jóval nehezebben oldják meg az ilyen problémákat, szemben a csak folytonos változókat tartalmazókkal. Az olyan feladatokat, amelyek csak egész változókat tartalmaznak, egészértékű lineáris problémáknak (integer programming, IP) nevezzük. Előfordulhat, hogy a változók egy része egész, míg a többi folytonos, ekkor vegyes egészértékű lineáris problémáról (mixed integer linear programming, MILP) beszélünk. Az egészértékű változókon belül megkülönböztetjük a bináris változókat. Ezen változók két értéket vehetnek fel, 0-t vagy 1-et.

Általában, minél több egész változót tartalmaz egy modell, annak megoldása annál bonyolultabb, több időt vesz igénybe. Míg az LP modelleknél minél több korlátozást adunk meg, a megoldás megkeresése annál tovább tart, az egészértékű feladatoknál a korlátozások hozzáadása többnyire gyorsít a megoldás menetén: Minél szűkebbre szabjuk a keresési teret, annál hamarabb találunk megfelelő megoldást. Itt akkor nő a megoldási idő, ha több egészértékű változót adunk a modellhez. Amennyiben egy LP modellt egy korlátozással bővítjük, a megoldóprogram futási ideje csak kis mértékben (akár észrevehetetlen mértékben) nő, ám ha egy IP, vagy MILP modell bővül egyetlen egész változóval, a megoldási idő akár a többszörösére is nőhet. Éppen ezért bizonyos problémákra a hatékonyan megoldható MILP modellek felírásának, vagy azok megoldásának kutatásával számos kutató foglalkozik.

Az alábbiakban egy elterjedt megoldó algoritmust ismertetünk. A megoldáshoz felhasználjuk a már korábban ismertetett szimplex módszert. Az első lépésben azt mondjuk, hogy eltekintünk attól, hogy bizonyos változók csak egész értéket vehetnek fel, és így oldjuk meg a modellt. Ezt a módosított modellt hívjuk az eredeti egészértékű feladat relaxációjának. Oldjuk meg a pizzás feladatot úgy, hogy a pizza mennyiségeket reprezentáló változók csak egészek lehetnek! Először tehát oldjuk meg a modell relaxációját, ekkor azt kapjuk, hogy a célfüggvény értéke 39666.66667, az x[margareta] 48.3333, az x[hawaii] pedig 13.3333. Mivel a kapott megoldásban az egész változók értékei nem egészek, ezért nem ér véget az algoritmus. Válasszunk ki egy változót, amely bár egész típusú, de a relaxált feladat megoldásában nem egészre jött ki. Legyen ez az x[hawaii]. Oldjuk meg úgy a feladatot, hogy ez a változó ne vehesse fel a 13.3333 értéket: 13-at, vagy annál kevesebbet, illetve 14-et vagy annál többet viszont felvehet. Származtassunk tehát az eredeti modellből két újabb modellt, azzal a módosítással, hogy az egyik esetében egy új korlátozással előírjuk, hogy az x[hawaii] kisebb vagy egyenlő legyen mint 13, a másik változatban pedig nagyobb, vagy egyenlő legyen, mint 14. Azt mondjuk, hogy az x[hawaii] változó mentén szétválasztjuk a feladatot. Oldjuk meg a két

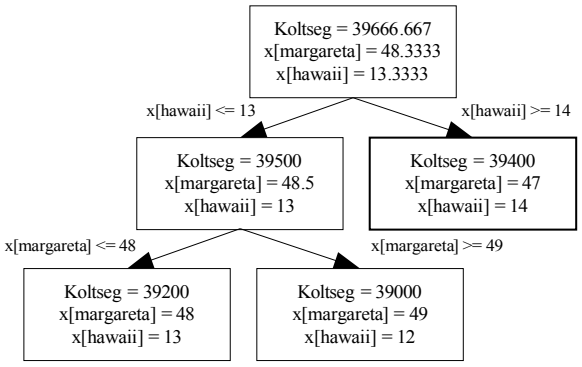
 

módosított modell relaxációját is, majd újra vizsgáljuk meg a kapott változókat. Az eredményt a 4.13 ábrán láthatjuk.




*4.13. ábra. Elvégezzük az első szétválasztást*

Láthatjuk, hogy az x[hawaii] >= 14 esetben a változók értékei egészek, tehát ez egy megvalósítható megoldás. Vegyük észre, hogy az újabb korlátozásnak köszönhetően, mindkét gyerek probléma célfüggvényének értéke romlott az eredeti problémához képest. Ezt a tulajdonságot kihasználjuk az optimális megoldás keresése során. Az x[hawaii] $\leq$= 13 esetben a célfüggvény értéke jobb, mint a másikban, ezért nem zárható ki, hogy ezt az ágat tovább bontva a 39400-nál jobb megoldást kapunk. Vezessünk be egy *max* változót, amely az aktuális legjobb megoldás értékét tárolja. Maximalizálási feladat esetén ennek kezdeti értéke mínusz végtelen. Amennyiben találunk egy újabb megvalósítható megoldást, és a *max* értéke rosszabb, mint ezen megoldás értéke, úgy a *max* eltárolja az új megoldás értékét. Később, ha olyan részproblémát kapunk, amely célfüggvényének értéke nem jobb, mint a *max* aktuális értéke, ezt a részproblémát eldobhatjuk, mivel ebből biztosan nem származik a max-nál jobb megoldás. Ezt az eldobási lépést szokás vágásnak is hívni. A példánkban az első szétválasztás után a max értéke tehát 39400. Válasszuk szét a bal oldali részproblémát az x[margareta] változó szerint, az eredmény a 4.14 ábrán látható.




*4.14. ábra. Szétválasztjuk az első gyerek részproblémát*

Sajnos mindkét esetben rosszabb megvalósítható megoldást kaptunk, mint a max értéke. Mivel már nincs olyan részproblémánk, ahol legalább egy változó értéke nem egész, ezért az algoritmus futása véget ér, és megállapíthatjuk, hogy a feladat optimális megoldása 47 margaréta és 14 hawaii pizza gyártása naponta, így pedig 39400 forint bevételre tehetünk szert.

Az ismertetett algoritmus a korlátozás és szétválasztás elvét alkalmazta (angolul branch and bound vagy B&B). Láthatjuk, hogy több egész változó esetén miért tart lényegesen tovább

a probléma megoldása. Minél több egész változó van, annál több elágazás jön létre a fában, a kiértékelendő részproblémák száma gyakran a feladat méretéhez képest exponenciálisan növekszik.

Módosítsuk a GNU MathProg-ban megfogalmazott modellünket úgy, hogy a változók csak egészek lehessenek. Ehhez mindössze az integer kulcsszót kell beírnunk a változók létrehozásának kifejezésébe.

```
var x{Pizzak} >= 0, integer;
```

A módosított modellt megoldva valóban azt kapjuk, hogy x[margareta] = 47, x[hawaii] = = 14, és a célfüggvény értéke 39400. Vizsgáljuk meg a megoldó kimenetét, amely a 4.15 ábrán látható.

```
Solving LP relaxation…
*     0: obj =   0.000000000e+00  infeas =  0.000e+00 (0)
*     3: obj =   3.966666667e+04  infeas =  0.000e+00 (0)
OPTIMAL SOLUTION FOUND
Integer optimization begins…
+     3: mip =     not found yet <=              +inf        (1; 0)
+     4: >>>>>   3.940000000e+04 <=   3.940000000e+04   0.0% (3; 0)
+     4: mip =   3.940000000e+04 <=     tree is empty   0.0% (0; 5)
INTEGER OPTIMAL SOLUTION FOUND
```

*4.15. ábra. A megoldó kimenete*

Láthatjuk, hogy a megoldó először az eredeti modell relaxációját oldja meg, majd utána bontja ki a keresőfát. Az Integer optimization begins… felirat utáni sorokban követhetjük nyomon, hogy a megoldónak a keresési tér hány %-át kell még átnéznie, valamint hogy eddig mennyi nyitott részprobléma található a kereső fában, és mennyi részprobléma került szétválasztásra. Valós, ipari feladatok esetén ez a lista ennél jóval hosszabb lehet, ilyenkor a program akár órákig, vagy napokig is futhat. Ekkor a megoldó időnként azt is kiírja, hogy mennyi ideje fut, és mennyi memóriát használ jelen pillanatban.

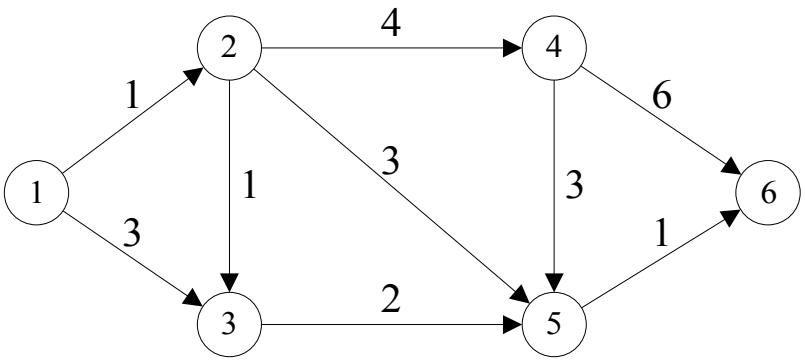
## 4.4. Nevezetes feladatok megoldása matematikai programozással

A 3. fejezetben ismertettünk néhány gráf algoritmust, ebben a fejezetben pedig bemutatunk egy-egy MILP modellt, amelyekkel az ott felvetett feladatokat szintén megoldhatjuk. Az ott bemutatott problémákat ebben a fejezetben ismertnek tekintjük, tehát ha az Olvasó kihagyta a gráf algoritmusokról szóló részt, akkor ajánlott elolvasni azt.

### 4.4.1. Legrövidebb út

A legrövidebb út kereséséhez azt kell kihasználni, hogy az út egy olyan részgráfja a fának, amelynek az első és utolsó csomópontját kivéve minden csomópont foka kettő, az elsőé és utolsóé pedig egy. Azaz, ha az út mentén átlépünk az egyik csomópontról a másikra, akkor legfeljebb egy irányba haladhatunk tovább. Továbbá a problémát úgy is megfogalmazhatjuk, hogy mely éleket kell kiválasztani ahhoz, hogy a kiválasztott élek egy minimális hosszúságú utat határozzanak meg. Valamint tudjuk azt is, hogy a kezdő csomópontról tovább kell lépni valamelyik szomszédos csomópontra. A feladatot egy $G = (V, A, w)$ irányított gráfon oldjuk meg, ahol w egy $A \to \mathbb{R}^+$ súlyfüggvény. Vezessük be még a következő jelöléseket: $d^+(v)$ jelölje azon csomópontok halmazát, amelyekre van irányított él a v csomópontból, a $d^-(v)$ pedig a v-t megelőző csomópontok halmazát. Például a 4.16 ábrán látható gráfon $d^+(5) = \{1\}$, $d^-(5) = \{2, 3, 4\}$. Végül, jelöljük $s$-el a kiinduló, $t$-vel pedig a cél csomópontot.




*4.16. ábra. Keressük meg a legrövidebb utat az 1. csomópontból a 6-osba*

A fenti észrevételeket kihasználva a következő módon írhatunk fel egy MILP modellt. Először meg kell állapítani, hogy mik lesznek a döntési változók. Mivel arra vagyunk kíváncsiak, hogy mely éleket kell beválasztani, így célszerű minden élhez hozzárendelni egy-egy bináris változót, amiket úgy értelmezünk, hogy ha az adott változó értéke 1, akkor a hozzá tartozó élt beválasztjuk, egyébként nem. Az $a \in A$ élhez tartozó változót jelölje $x(a)$. Ezek után könnyen felírhatjuk a célfüggvényt. Az élek súlyainak megfelelően felírjuk a változók súlyozott összegét és ezt az értéket kell minimalizálni.

$$\min \sum_{a \in A} x(a) * w(a) \tag{4.6}$$

Ez még nem elegendő a megoldáshoz, elő kell írnunk, hogy valamilyen szempontok szerint mindenképpen ki kell választani bizonyos éleket. Tudjuk, hogy a kezdő csomópontból kiinduló élek egyikét biztosan ki kell választani, vagyis ha összegezzük az induló csomópontból kiinduló élekhez tartozó változókat, akkor pontosan 1-et kell kapnunk:

$$\sum_{(s,i) \in d^+(s)} x((s, i)) = 1 \tag{4.7}$$

Ha kiválasztottunk egy s-ből induló élt és az ahhoz tartozó változó értéke 1 lesz, akkor biztosítani kell, hogy az él másik csomópontjából újra kiválasszunk pontosan egy élt. Viszont, ha az egyik élt nem választottuk ki, akkor az ahhoz tartozó másik csomópontból nem szabad kiindulnia egyetlen élnek sem. Például, ha az $x((1,2))$ változó értéke 1, akkor kiválasztottuk az $(1,2)$ élt. Ez esetben választani kell a $(2,3)$, $(2,4)$ és $(3,5)$ élek közül. Tehát, az $s$ és $t$ csomópontokat kivéve minden más csomópontra igaz az, hogy ha egy irányított él mentén ráléptünk az adott csomópontra, akkor és csak akkor kell pontosan egy kimenő él mentén tovább lépni egy másik csomópontra:

$$\forall v \in V \setminus \{s,t\} : \sum_{(i,v)\in d^-(s)} x((i,v)) = \sum_{(v,i)\in d^+(s)} x((v,i)) \tag{4.8}$$

Lássuk, hogy a 4.16 ábrán látható gráf esetén hogy néz ki a modellünk a GNU Mathprog nyelvén, ha az 1-es csomópontból szeretnénk eljutni a 6-osba. A modell a 4.17 ábrán látható.

```
var x12 binary;
var x13 binary;
var x23 binary;
var x24 binary;
var x25 binary;
var x35 binary;
var x45 binary;
var x46 binary;
var x56 binary;

s.t. indulas: x12 + x13 = 1;

s.t. n2: x12 = x23 + x24 + x25;
s.t. n3: x13 + x23 = x35;
s.t. n4: x24 = x45 + x46;
s.t. n5: x25 + x35 + x45 = x56;

minimize ut: x12 + 3*x13 + x23 + 4*x24 + 3*x25 + 2*x35 + 3*x45 +
6*x46 + x56;
end;
```

*4.17. ábra. A feladathoz tartozó modell*

A modellt megoldva azt kapjuk, hogy az x12, x25 és x56 változók értéke 1, a többi pedig 0, tehát a legrövidebb út az 1, 2, 5, 6 csomópontokon halad keresztül, a hossza pedig 5.

Írjunk egy általános GNU Mathprog modellt, ahol csak a gráf adatait kell módosítani! Először a modell fájlt ismertetjük, utána megmutatjuk a példához tartozó adat fájlt.

Első lépésként vegyük fel a csúcsok és élek halmazát. Az élek halmaza rendezett ketteseket tartalmaz, ezért a halmaz neve után a dimen kulcsszóval meg kell adni, hogy 2 dimenziós értékek halmazáról van szó:

```
set Csucsok;
set Elek dimen 2;
```

Minden élhez tartozik egy súly paraméter:

```
param Suly{ (i, j) in Elek};
```

Vegyünk fel még két paramétert, amelyekkel megadhatjuk, hogy melyik csomópontból melyik csomópontba szeretnénk megkeresni a legrövidebb utat:

```
param start;
param cel;
```

Végül adjunk egy-egy bináris változót minden élhez:

```
var x{(i, j) in Elek} binary;
```

Adjuk meg azt a korlátozást, ami biztosítja, hogy a start csomópontból mindenképpen elinduljunk az egyik irányba:

```
s.t. kezdoUt: sum{ (start, i) in Elek} x[start, i] = 1;
```

A start és cel csomópontokat kivéve minden más csomópontra fel kell írni a továbbhaladást biztosító feltételt:

```
s.t. tovabbHaladas{v in Csucsok diff {start, cel} }: sum{ (i, v) in
Elek } x[i, v] = sum{ (v, j) in Elek } x[v, j];
```

A diff kulcsszó segítségével a Csucsok diff {start, cel} kifejezés eredménye egy olyan halmaz lesz, amelyben nem szerepel a start és cel csomópont. Végül adjuk meg a célfüggvényt:

```
minimize ut: sum{ (i,j) in Elek } x[i, j] * Suly[i, j];
end;
```

A példához tartozó modell fájl tartalma a 4.18 ábrán látható.

```
set Csucsok:= 1 2 3 4 5 6;
set Elek:= (1, 2) (1, 3) (2, 3) (2, 4) (2, 5) (3, 5) (4, 5) (4, 6)
(5, 6);

param Suly:= [1, 2] 1 [1, 3] 3 [2, 3] 1 [2, 4] 4 [2, 5] 3 [3, 5]
2 [4, 5] 3 [4, 6] 6 [5, 6] 1;

param start:= 1;
param cel  := 6;

end;
```

*4.18. ábra. A modell fájl tartalma*

### 4.4.2. Minimális feszítőfa

Ahhoz, hogy felírjunk egy modellt, amellyel egy gráf minimális feszítőfáját szeretnénk meghatározni, a következő követelményeket kell teljesíteni. Az előző feladathoz hasonlóan bináris változók segítségével ki kell választani a megfelelő éleket, amelyek a feszítőfát alkotják, mégpedig úgy, hogy ezen élek összsúlya minimális legyen. Továbbá az éleknek le kell fedniük minden csomópontot, végül pedig az éleknek fát kell alkotniuk. Az első feltételt legrövidebb út keresésekor alkalmazott módszerhez hasonlóan biztosíthatjuk, azaz az élekhez tartozó bináris változókat az élek súlyával szorozva össze kell adni, majd ezt az értéket kell minimalizálni.

A fákról tudjuk, hogy bennük az élek száma egyel kisebb, mint a csomópontok száma, összefüggőek és körmentesek. Ha a fenti 3 állítás közül bármelyik 2 teljesül egy gráfra, akkor az fa. Az élek mennyiségét könnyű biztosítani, elő kell írni, hogy az éleket kiválasztó bináris változók összege legyen a csomópontok számánál 1-el kisebb érték. A körmentesség általános esetben úgy teljesíthető, hogy a gráfban előforduló összes körre felírunk egy-egy korlátozást, amelyekben tiltjuk azt, hogy a kört alkotó élek egyszerre bekerüljenek a fába. Ennek a módszernek viszont nagy hátránya, hogy már közepes méretű gráfokban is rengeteg kör fordulhat elő, ezek felkutatása alkalmanként roppant időigényes feladat. Az irodalomban egyébként gyakran hivatkoznak erre a módszerre. Mi most az összefüggősség fenntartására fektetjük a hangsúlyt, látni fogjuk, hogy a korlátozások száma jóval barátságosabb lesz, mint a körök elkerülésekor.

Mikor összefüggő egy gráf? Gondoljunk a gráfra egy anyag szállító hálózatként, ahol az egyik csomópont termel, a többi fogyaszt. A beválasztott élek mentén szállíthatjuk a csomópontokból az anyagot úgy, hogy ha egy csomópontban jelen van $n$ egységnyi anyag, akkor csak $n-1$ egységnyit ad tovább, mivel 1 egységet elfogyaszt. Ha valamelyik csomópontnak nem jut anyag, akkor a gráf nem összefüggő. A modell a következő módon írható fel. A $G = (V, E)$ irányítatlan gráfra tekintsünk úgy, mint egy $G = (V, A)$ irányított gráfra úgy, hogy ha egy $u$ és $v$ csomópont össze van kötve, akkor az $\{u, v\}$ irányítatlan élből alkossunk egy $(u, v)$ és egy $(v, u)$ élt, ahol a két él súlya megegyezik az eredeti irányítatlan él súlyával. Minden újonnan kapott irányított élhez tartozik egy bináris és egy folytonos változó. Jelölje $y(u, v)$ az $(u, v)$ élhez tartozó bináris, $x(u, v)$ pedig az ehhez az élhez rendelt folytonos változót. A bináris változó szabja meg, hogy az élt beválasztjuk-e a feszítőfába, vagy nem, a folytonos pedig megadja, hogy az élen mennyi anyagot szállítunk az egyik csomópontból a másikba. Egy $(u, v)$ élen akkor lehet anyagot szállítani, ha az élt kiválasztottuk, tehát ha $y(u, v)$ értéke 1. Az éleken szállítható maximális anyagmennyiség pedig egyel kevesebb, mint a csomópontok száma, ugyanis ennyi fogyasztó van és minden fogyasztó pontosan 1 egységnyi anyagot fogyaszt. A továbbiakban legyen $M = |V| - 1$. Ezek alapján az $x(u, v)$ változókra igaz, hogy

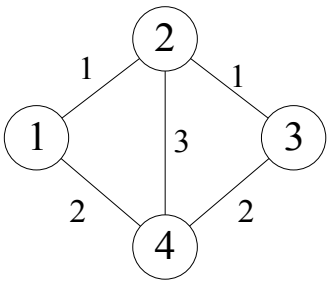
$$0 \leq x(u, v) \leq M * y(u, v) \tag{4.9}$$

minden $(u, v) \in A$-ra.

Az anyagáramlás biztosításához pedig minden fogyasztó csomópontra elő kell írni egy olyan korlátozás, hogy ezen csomópontokból egyel kevesebb anyag egység áramoljon ki, mint ami be áramlott. Jelöljük $s$-el a forrásnak választott csomópontot (ez bármelyik lehet).

$$\sum_{(i,v)\in A} x(i, v) - \sum_{(v,i)\in A} x(v, i) = 1 \tag{4.10}$$




*4.19. ábra. A minimális feszítőfa példához tartozó gráf*

minden $v \in E \setminus \{s\}$-re.

Ez akkor teljesülhet, ha a fa 1 fokú csomópontjaiba pontosan 1 egység anyag érkezik, a többi csomópontnak pedig tovább kell adnia valamennyi anyagot a szomszédjának, ennek következtében biztos, hogy minden csomópontot lefedünk, mégpedig egy összefüggő részgráffal. A gráfban nem keletkezik kör, hiszen ha kör keletkezne, az azt jelentené, hogy van olyan csomópont, amelybe több élen keresztül is áramlik anyag, ám ez feleslegesen növelné a beválasztott élek és így azok összsúlyainak összegét, ám azzal, hogy ez utóbbi értéket minimalizáljuk, kizárjuk az ilyen eseteket.

```
var xAB >= 0;   var xBA >= 0;
var xBC >= 0;   var xCB >= 0;
var xCD >= 0;   var xDC >= 0;
var xAD >= 0;   var xDA >= 0;
var xDB >= 0;   var xBD >= 0;

var yAB binary;   var yBA binary;
var yBC binary;   var yCB binary;
var yCD binary;   var yDC binary;
var yAD binary;   var yDA binary;
var yDB binary;   var yBD binary;

s.t. ab: xAB <= 3*yAB;   s.t. ba: xBA <= 3*yBA;
s.t. bc: xBC <= 3*yBC;   s.t. cb: xCB <= 3*yCB;
s.t. cd: xCD <= 3*yCD;   s.t. dc: xDC <= 3*yDC;
s.t. ad: xAD <= 3*yAD;   s.t. da: xDA <= 3*yDA;
s.t. db: xDB <= 3*yDB;   s.t. bd: xBD <= 3*yBD;

s.t. b: xAB + xDB + xCB - (xBA + xBD + xBC) = 1;
s.t. c: xBC + xDC - (xCB + xCD) = 1;
s.t. d: xAD + xBD + xCD - (xDA + xDB + xDC) = 1;

minimize feszitofa: yAB + yBA + yBC + yCB + 2*(yAD + yDA) + 2*(yDC + yCD)
+ 3*(yBD + yDB);
end;
```

*4.20. ábra. A gráfhoz tartozó modell*

```
set Csucsok;
set Elek dimen 2;
set Elek2:= Elek union setof { (i, j) in Elek } (j, i);

set forras;

param Suly{ (i, j) in Elek };

var x{(i, j) in Elek2} >= 0;
var y{(i, j) in Elek2} binary;

s.t. kapacitas{ (i, j) in Elek2 }: x[i, j] <= y[i, j] * (card(Csucsok) -
1);

s.t. aramlas{ v in Csucsok diff forras }: sum{ (i, v) in Elek2 } x[i, v]
- sum{ (v, i) in Elek2 } x[v, i] = 1;

minimize feszitofa: sum{ (i, j) in Elek } (y[i, j] + y[j, i]) * Suly[i,
j];

end;
```

*4.21. ábra. A feszítőfa feladat általános modellje*

Lássuk, hogyan néz ki egy modell egy konkrét gráfra. Most kivételesen nem arra a gráfra írjuk fel a modellt, mint amit a megfelelő gráf algoritmus során használtunk, hanem egy kisebbet választunk, hogy kevesebb változót használjunk. Ez a gráf a 4.19 ábrán, a gráfhoz tartozó modell pedig a 4.20 ábrán látható. A feszítőfa feladathoz tartozó általános modell a 4.21 ábrán látható.

Az Elek2 halmaz objektumra azért van szükségünk, mert az adat fájlban az éleket úgy tüntetjük fel, mint irányítatlan éleket, ahogy az az eredeti feladatban is van. A setof kulcsszóval képzünk egy olyan halmazt, amely olyan csúcspárokat tartalmaz, amelyek az Elek halmazban megadott élek fordítottjai, majd ezt a halmazt unio művelettel összefűzzük az eredeti élek halmazával. A forrást elegánsabb lenne paraméterként megadni, ám csak numerikus érték lehet paraméter, ezért most kivételesen ezt egy egyelemű halmazzal adjuk meg. Magyarázatra szorul még a card kulcsszó: Visszaadja a paraméterében megadott halmaz számosságát. A példa adatfájlja a 4.22 ábrán látható. A modellnek köszönhetően az adatfájl megadásakor nem kell azzal foglalkozni, hogy az éleket irányítottá kell tenni és minden élt megduplázunk, ezt a glpsol automatikusan elvégzi a modell alapján.

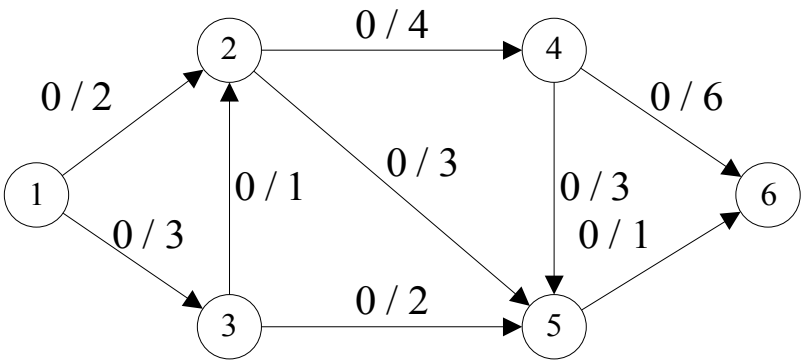
```
set Csucsok:= A B C D;
set Elek:= (A, B) (B, C) (D, C) (B, D) (A, D);

param Suly:= [A, B] 1 [B, C] 1 [D, C] 2 [B, D] 3 [A, D] 2;

set forras:= A;
end;
```

*4.22. ábra. A feszítőfa feladathoz tartozó adatfáj tartalma*

### 4.4.3. Maximális folyam

Ha olyan modellt szeretnénk felírni, amely egy hálózat maximális folyamát adja meg, akkor elég, ha megadjuk mindazokat a feltételeket, amelyeket a Ford-Fulkerson algoritmus tárgyalásakor is láttunk, nevezetesen: Egyik élhez tartozó folyam nagysága sem haladhatja meg az él kapacitását, valamint a kezdő és cél csomópontot kivéve minden más csomópontra igaz, hogy amennyi anyag oda befolyik, annyinak kell tovább haladnia. A modellt a 4.23 ábrán látható hálózatra írjuk fel úgy, hogy az 1. csomópontból juttatjuk el az anyagot a 6-osba.




*4.23. ábra. Ezen a hálózaton határozzuk meg a maximális folyamot az 1. és 6. csomópontok között*

```
var x12 >= 0;
var x13 >= 0;
var x24 >= 0;
var x25 >= 0;
var x32 >= 0;
var x35 >= 0;
var x45 >= 0;
var x46 >= 0;
var x56 >= 0;

s.t. c12: x12 <= 2;
s.t. c13: x13 <= 3;
s.t. c24: x24 <= 4;
s.t. c25: x25 <= 3;
s.t. c32: x32 <= 1;
s.t. c35: x35 <= 2;
s.t. c45: x45 <= 3;
s.t. c46: x46 <= 6;
s.t. c56: x56 <= 1;
```

*4.24. ábra. Korlátozások a maximális folyam feladathoz*

Mivel a folyam értékeket szeretnénk meghatározni, a változókat érdemes úgy megválasztani, hogy minden folyamhoz tartozzon egy-egy folytonos változó. Minden változóra meg

kell adni egy-egy korlátot, amely a kapacitás alapján megszabja a változó, azaz a folyam felső korlátját a 4.24 ábrán látható módon, illetve az anyagmegmaradást biztosító korlátozásokat is fel kell írni a 2, 3, 4 és 5-ös csomópontokra a 4.25 ábrán látható módon.

```
s.t. kibe2: x12 + x32 = x24 + x25;
s.t. kibe3: x13 = x32 + x35;
s.t. kibe4: x24 = x45 + x46;
s.t. kibe5: x25 + x35 + x45 = x56;
```

*4.25. ábra. Korlátozások a maximális folyam feladathoz*

Végül adjuk meg a célfüggvényt. A maximális folyam értéke megegyezik a kezdő csomópontból kiinduló összes anyag, vagy a cél csomópontba befolyó összes anyag mennyiségével, hogy melyiket választjuk, az lényegtelen:

```
maximize folyam: x12 + x13;
end;
```

A modellt megoldva a változók a 4.2 táblázatban látható értékeket veszik fel.

*4.2. táblázat. A változók értékei*

| Változó | Érték |
|---|---|
| x12 | 2 |
| x13 | 2 |
| x24 | 3 |
| x25 | 0 |
| x32 | 1 |
| x35 | 1 |
| x45 | 0 |
| x46 | 3 |
| x56 | 1 |

Láthatjuk, hogy a maximális folyam értéke négy. Vessük össze ezeket az értékeket a 3.4. fejezetben megoldott maximális folyam feladat megoldásával és láthatjuk, hogy ugyanazokat a folyam értékeket kaptuk.

Az általános GNU Mathprog modell a 4.26 ábrán, az adatfájl tartalma pedig a 4.27 ábrán látható.

```
set Csucsok;
set Elek dimen 2;

param Kapacitas{ (i, j) in Elek};

param start;
param cel;

var x{(i, j) in Elek} >= 0;

s.t. KapacitasKorlat{ (i, j) in Elek } : x[i, j] <=Kapacitas[i, j];

s.t. anyagMegmaradas{v in Csucsok diff {start, cel} } : sum{ (i, v) in Elek } x[i, v] = sum{
(v, j) in Elek } x[v, j];

maximize maxFolyam: sum{ (start, i) in Elek } x[start, i];

end;
```

*4.26. ábra. A maximális folyam feladathoz tartozó általános modell*

```
set Csucsok := 1 2 3 4 5 6;
set Elek := (1, 2) (1, 3) (3, 2) (2, 4) (2, 5) (3, 5) (4, 5) (4, 6) (5, 6);

param Kapacitas := [1, 2] 2 [1, 3] 3 [3, 2] 1 [2, 4] 4 [2, 5] 3 [3, 5] 2 [4, 5] 3 [4, 6] 6 [5, 6] 1;

param start := 1;
param cel  := 6;

end;
```

*4.27. ábra. Az adatfájl tartalma*

## 4.5. Feladatok

**4.1. Feladat** Egy bútorgyárban a következő termékeket gyártják az alábbi eladási árak mellett:

- íróasztal: 20 000 Forint
- szék: 10 000 Forint
- ruhásszekrény: 25 000 Forint
- étkezőasztal: 35 000 Forint

A fenti bútorokhoz a következő nyersanyagokat használják: tölgyfa, fenyőfa, farostlemez, enyv, szegek. Az alábbiakban megadjuk, hogy melyik bútorhoz, hány egységnyit kell felhasználni a felsorolt alapanyagokból (a szegeknél 10 db felel meg 1 egységnek):

- íróasztal: 2 fenyőfa, 1 farostlemez, 3 enyv, 2 szeg
- szék: 1 tölgyfa, 4 enyv

- ruhásszekrény: 3 farostlemez, 1 fenyőfa, 1 szeg
- étkezőasztal: 3 tölgyfa, 3 enyv, 4 szeg

Tudjuk, hogy amit egy héten előállítanak, azt mind eladják a következő hétre. A hét elején a nyersanyagokból az alábbi mennyiségeket szerzik be:

- tölgyfa: 20 egység
- fenyőfa: 30 egység
- farostlemez: 50 egység
- enyv: 40 egység
- szeg: 60 egység

(a) Lineáris programozás segítségével határozza meg, hogy az adott árak és rendelkezésre álló nyersanyag mennyiségek mellett a héten mely bútorból hány darabot kell készíteni ahhoz, hogy a lehető legnagyobb bevételre tegyünk szert! Írja fel a használt modellt formálisan, majd azt oldja meg GLPK segítségével. Fontos, hogy a vásárlók kizárólag teljes bútorokat hajlandóak megvásárolni.

(b) A héten fel nem használt anyag mennyiséget hétvégén raktárban kell tartani, a raktár használata viszont pénzbe kerül. A különféle nyersanyagok tárolási költsége 1 egységre a következő:

- tölgyfa: 5000 Forint
- fenyőfa: 4000 Forint
- farostlemez: 3000 Forint
- enyv: 500 Forint
- szeg: 700 Forint

Módosítsa az előző feladatban felírt modellt, hogy a raktározási költséget figyelembe véve maximalizálja a profitot!

*4.3. táblázat. A feladat paraméterei*

| | szerelő csarnok | raktár | irodák | minőség ellenőrző |
|---|---|---|---|---|
| szerelő csarnok | hallható | hallható | | hallható |
| raktár | hallható | hallható | hallható | |
| irodák | | hallható | hallható | |
| minőség ellenőrző | hallható | | | hallható |

**4.2. Feladat** Egy gyárba hangos bemondó rendszert telepítenek. A fejlesztés célja az, hogy a bemondón elhangzott információkat mindenhol hallani lehessen, továbbá a feladatot minél kevesebb hangszóróval kell megoldani. A 4.3 táblázatban feltüntettük, hogy ha hangszórót telepítünk egy adott helyiségbe, akkor annak hangját mely helyiségekből lehet hallani.

(a) Írj fel egy MILP modellt, amely segítségével meghatározható, hogy hogyan lehet a legkevesebb számú hangszóró telepítésével teljesen lefedni a gyárat.

(b) Módosítsd az előző modellt úgy, hogy a szerelő csarnokban kétszer annyi hangszórót lehessen hallani, mint a raktárban!

# 5. fejezet

# Folyamathálózat-szintézis és optimalizálás

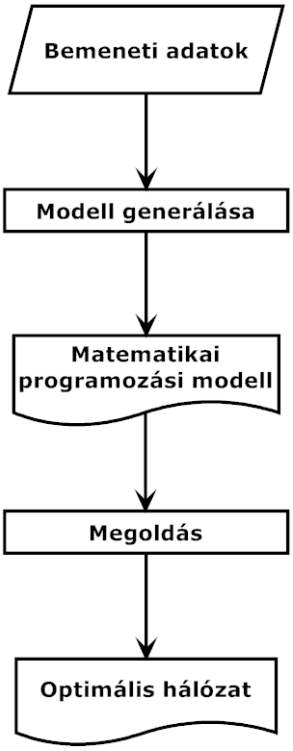
A folyamatszintézis feladata egy folyamatrendszer optimális struktúrájának, valamint a rendszert alkotó, különféle műveleteket végrehajtó funkcionális egységek optimális konfigurációinak, kapacitásainak meghatározása [16]. Szerepe kritikus az anyag- és energiafogyasztás, illetve a környezetre gyakorolt negatív hatások csökkentésében, ezáltal pedig a nyereségesség növelésében. Szakirodalmi példa támasztja alá, hogy a hatékony folyamatszintézis az energiafogyasztást akár 50, a költséget pedig 35%-al is csökkentheti [20].

Ideális esetben egy folyamat struktúráját, és a folyamatot alkotó műveletek konfigurációit egyidejűleg tudnánk megtervezni és szintetizálni, mivel a teljesítményük kihatással van egymásra. Ez azonban a gyakorlatban rendkívül nehéz, ha nem lehetetlen, köszönhetően a probléma duális, egyidejűleg folytonos és diszkrét természetének. Az utóbbi a feladat kombinatorikus komplexitásához vezet, ami a probléma optimális megoldásának megtalálását jelentősen megnehezíti. Emiatt, a 2.1.2 fejezetben már említett módon, a folyamatszintézisnek 3 fázisát különböztetjük meg : a makroszkopikus, mezoszkopikus és a mikroszkopikus fázist. A különböző részfeladatokat már ezek között a fázisok között osztjuk el, a végcélt, azaz az optimális folyamat megtervezését és szintézisét szem előtt tartva. Ahogyan az nevükből is következik, a 3 fázist az alapján különböztetjük meg, hogy milyen részletességgel foglalkoznak a folyamat megtervezésével. A részrendszerek, a funkcionális egységek összekapcsolása, azaz a rendszer szintézise a makroszkopikus fázisban megy végbe, ezért a folyamatszintézis szempontjából ez a fázis a legjelentősebb.

A folyamatszintézis feladatok megoldására kidolgozott módszerek két nagy csoportba osztjuk, mégpedig a heurisztikus és az algoritmikus azaz matematikai programozáson alapuló módszerekre. Léteznek úgynevezett hibrid módszerek is, amelyek heurisztikus szabályok mellett egyidejűleg támaszkodnak a matematikai programozásra is.

A heurisztikus módszerek megvalósítása általában egyszerű, még nagy feladatok esetében is, azonban természetüknél fogva csak lokálisan hatékonyak. Ennek oka, hogy az emberi tapasztalatok, melyeken a heurisztikák szabályai alapulnak, véges, és gyakran korlátozott, limitált számú megfigyelésből erednek. Következésképpen a heurisztikus módszerek önmagukban gyakran alkalmatlanok a globális, vagy közel globális optimális megoldások megtalálására [3].

A heurisztikus módszerekkel szemben, a meglévő, hagyományos algoritmikus módszerek, melyek főbb lépései az 5.1 ábrán láthatóak, csak viszonylag kis, illetve mérsékelt méretű feladatok kezelésére alkalmasak. Ezek az algoritmikus módszerek csak akkor lesznek megfelelően precízek, ha az általuk használt matematikai programozási modelleket explicit módon meg tudjuk konstruálni.



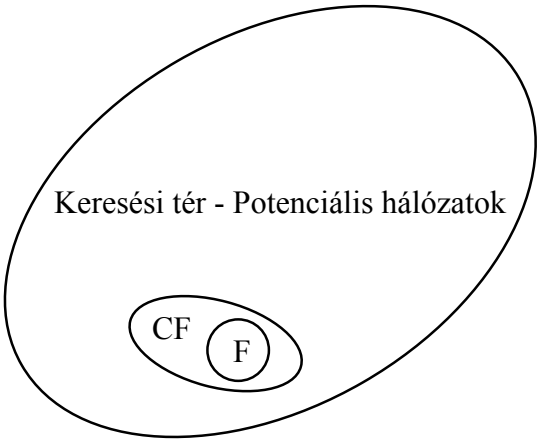

*5.1. ábra. A hagyományos algoritmikus módszerek főbb lépései*

Ezen módszerek esetében a feladat definíciója mindig megadja a rendelkezésre álló nyersanyagokat, a gyártani kívánt termékeket valamint a felhasználható műveleti egységeket. A különböző kapcsolódó paraméterekre – mint költségekre, árakra illetve anyagegyensúly feltételekre – írjuk fel a matematikai programozási modelleket. Ezeket a modelleket általában félig-meddig korábbi tapasztalatokra alapozva adják meg, folytonos függvények segítségével. Mindemellett ezek a hagyományos módszerek nem tudják közvetlenül és szisztematikusan, azaz algoritmikusan képezni a költségfüggvényt és a kapcsolódó feltételrendszert, amely megfelelően kifejezné a folyamatok hálózat-struktúráját. Nem adnak módszert arra sem, hogy algoritmikusan készítsük el egy olyan struktúráját a folyamatnak, amely minden lehetséges hálózatot redundancia nélkül tartalmaz. Ez azért kritikus, mert a matematikai programozás egyrészt csak olyan eredményt adhat, ami része a matematikai modellnek, másrészt használhatatlanul lelassulhat, ha a modell indokolatlanul nagy. A konvencionális módszerek esetében nincs garanciánk a folyamatszintézishez használt modellek és az abból származó megoldások minőségére.

Bármely hagyományos algoritmikus módszer esetében, egy vizsgált folyamat lehetséges hálózat struktúrája „manuálisan”, folyamatos fejlesztés, javítások és módosítások eredményeként áll elő. A javítások és módosítások folyamata során egyrészt gyakori a heurisztikus és

intuitív megoldások alkalmazása, másrészt pedig a kézzel felírt, minden lehetséges és nem lehetséges hálózatot tartalmazó teljes hálózat struktúrába beválasztott műveleti egységek közötti redundáns kapcsolatok megszüntetése. Mindkét megközelítésnek megvannak a maga nehézségei. A heurisztikus módszereknek a fent már említett buktatói mellett – ahogyan azt hamarosan majd látni fogjuk – a lehetséges műveleti egységek potenciális összekapcsolási lehetőségeinek nagy száma. Látható tehát, hogy heurisztikák alkalmazása még az algoritmikus módszerek esetében is gyakori.

Ahogyan azt korábban már említettük, a folyamatszintézis komplexitását a probléma kettős, egyidejűleg folytonos és diszkrét természete okozza. Az utóbbinak köszönhetően a probléma komplexitása exponenciálisan növekszik a rendelkezésre álló műveleti egységek számával, amit jelöljön $n$, mivel az optimális hálózatot $2^{n-1}$ lehetséges alternatíva közül kellene meghatározni mindaddig, amíg nem rendelkezünk olyan tudással a hálózat struktúrájáról, ami alapján helytelen alternatívák eltávolításával ezt a számot csökkenteni lehetne. A $2^{n-1}$ már viszonylag kis $n$ esetén is hatalmas szám is lehet: $n=35$ esetén $34.36\times10^9$, míg $n=36$ esetén már $68.72\times10^9$. Ez a probléma exponenciális jellegét is jól illusztrálja, az $n$-t eggyel növelve az alternatívák száma lényegében megduplázódott. Olyan robusztus döntéstámogatási rendszerekre van tehát szükség a folyamatszintézis feladatok kombinatorikus komplexitásának megfelelő kezelésére, amelyek matematikailag szigorúak, lehetőleg axiomatikusak, és számítógépekre hatékonyan implementálhatóak.




*5.2. ábra. A keresési tér csökkentése*

Mindez a kombinatorika egy jól kidolgozott ágának, a gráfelmélet eredményeinek felhasználásával valósítható meg. Eredményül egy gráfelméletben gyökerező, algoritmikus módszert kapunk, amelyet ebben a fejezetben részletesebben is bemutatunk. A módszer a gráfok egy speciális osztályán alapszik, amely segítségével a folyamatok struktúrája egyértelműen reprezentálható, így lehetőség nyílik a gyakorlatban is megvalósítható folyamatokra jellemző kombinatorikus tulajdonságok kihasználására. Matematikailag az ilyen kombinatorikus tulajdonságok axiómaként fogalmazhatóak meg, amelyek három algoritmus alapjául szolgálnak [4, 7, 6]. Az 5.2 ábrán látható, hogyan csökkenti a módszer a keresési teret az optimális megoldás keresése során. A keresési tér, amely az összes lehetséges hálózatot tartalmazza, leszűkül azokra a hálózatokra, amelyek kielégítik az axiómákat, azaz az úgynevezett kombinatorikusan lehetséges hálózatokra, amit az ábrán CF jelöl. A két halmaz mérete között több

nagyságrendbeli eltérés is lehet, amely a megoldáshoz igényelt számítási kapacitás drasztikus csökkenéséhez vezethet. A módszerrel megoldott gyakorlati példák során előfordult olyan eset is, amikor ez az arány egy az egymilliárdhoz volt. Itt érdemes talán megjegyezni, hogy a folyamathálózatok merőben eltérőek és lényegesen komplexebbek sok egyéb típusú (például telefon vagy autópálya) hálózatnál az érintett műveleti egységek nagy számának, illetve a rajtuk keresztüláramló anyagok változatosságának tekintetében (például egy telefonhálózaton csak fotonok és elektronok haladnak át). Emellett a probléma duális, folytonos és kombinatorikus (egész) természete miatt egy folyamatszintézis feladat matematikai programozási modelljéül vagy egy MILP, vagy egy MINLP szolgál. Ezen dualitás mértéke eltérő a különböző folyamatszintézis feladatok, mint például az RNS (reaction-network synthesis azaz reakcióhálózat-szintézis), SNS (separation-network synthesis azaz szétválasztási hálózatok szintézise), HENS (heat-exchanger-network synthesis azaz hőcserélő hálózatok szintézise) vagy PNS (process-network synthesis azaz folyamathálózat-szintézis) esetében. Természeténél fogva a legutóbbi a leginkább kombinatorikus.

Ennek a gráfelméleti alapokon nyugvó, algoritmikus módszernek egy korai [4, 7], sikeres alkalmazása egy régebbi, már létező Folpet folyamat újraszintetizálása, amely N-(trichloromethylthio) phthalimde előállítására szolgál. Ez egy gyomirtó, amelyet egészen az 1980-as évek végéig gyártottak, jelenleg azonban már nem alkalmazható a toxicitása miatt. Az újraszintetizálás a folyamat strukturális optimalitását vizsgálta. A vizsgálat eredményeként kiderült, hogy a folyamatban résztvevő műveleti egységek közül 5 redundáns. A fennmaradó 35 műveleti egységből a módszer PC-re (PC/AT) implementált verziója kevesebb, mint egy perc alatt előállította a szigorú szuperstruktúrát, 3465 kombinatorikusan lehetséges folyamathálózatot eredményezve.

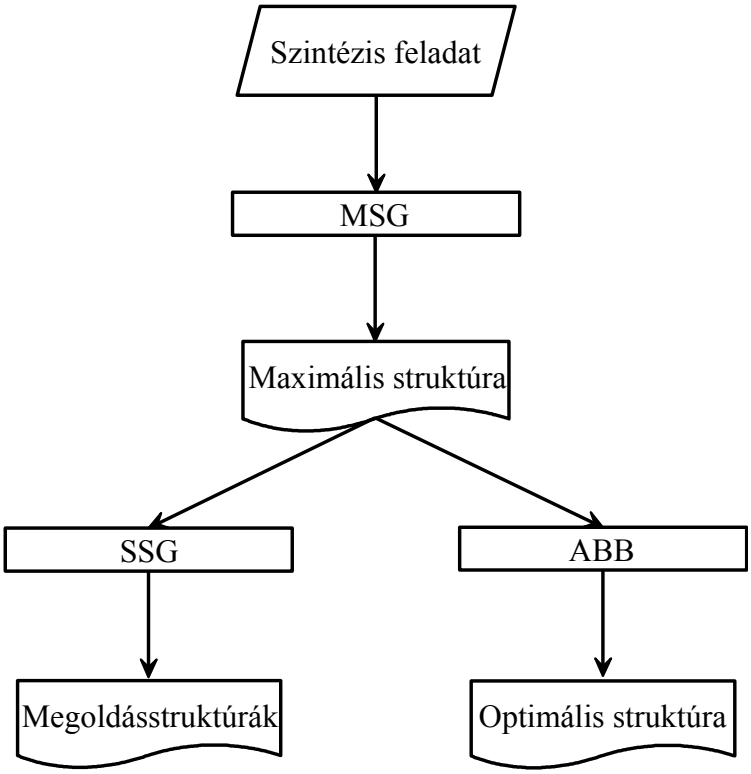
Egy másik esetben azt vizsgálták a módszer felhasználásával, hogyan lehet egy meglévő mezőgazdasági komplexumot optimálisan átalakítani (Hertwig et al., 2001). A komplexum 10 nagyobb üzemből és a hozzájuk kapcsolodó melléképületekből állt, amit a modellben 20 műveleti egység reprezentált. A feladathoz tartozó optimális hálózat meghatározása kevesebb, mint 5 másodpercet vett igénybe egy alacsony teljesítményű (Pentium III, 500 MHz) asztali számítógépen.

Az első esetben a 35 műveleti egységnek köszönhetően lehetséges $(2^{35} - 1)$ alternatíva közül kell megtalálni az optimálisat, ami 34 539 738 647, a második esetben pedig a 20 műveleti egység miatt $(2^{20} - 1) = 1\ 048\ 575$ közül. A lehetséges hálózatok nagy száma miatt a hagyományos algoritmikus módszerekkel az ilyen nagyságrendű feladatok már nem kezelhetőek megfelelően. Az alábbiakban bemutatjuk a már említett, gráfelméleti alapokon nyugvó, algoritmikus módszert, amivel még az ilyen komplexitású gyártási folyamatok is hatékonyan kezelhetőek és modellezhetőek folyamatszintézis feladatként.

## 5.1. A P-gráf módszertan alapjai

A P-gráf módszertant az 1990-es évek elején dolgozták ki komplex vegyipari termelőrendszerek modellezésére és optimalizálására. A nevét egy irányított páros gráfról, a P-gráfról kapta, amely segítségével lehetőség nyílik a lehetséges megoldásstrukturák kombinatorikus tulajdonságainak kihasználására, ezáltal pedig a nagyméretű feladatok optimumának megha-

tározására is. A módszertan főbb lépései az 5.3 ábrán láthatóak. A módszertanról jó áttekintést ad a következő könyv: [12].




*5.3. ábra. A P-gráf módszertan főbb lépései*

### 5.1.1. Alapfogalmak

Jelölje $M$ azoknak az anyagoknak a halmazát, amelyek a modellezendő gyártási folyamatban definiálva vannak. $M$ egy véges, nemüres halmaz. Az, hogy $M$ elemeit milyen részletességgel adjuk meg, mindig az adott feladattól, illetve a modellezőtől függ. Például

$$M_1 := \{A, B, C, D, E, F\} \tag{5.1}$$

és

$$M_2 := \{(x_1, \emptyset, \emptyset), (\emptyset, x_2, \emptyset), (\emptyset, \emptyset, x_3), (x_1, x_2, \emptyset), (\emptyset, x_2, x_3), (x_1, x_2, x_3)\} \tag{5.2}$$

egyaránt megfelelő lehet, attól függően, hogy csak maguk az anyagok, vagy azok összetétele is lényeges.

Az $M$ anyagaihoz kapcsolódó szintézis feladatot egy $(P, R, O)$ hármassal adjuk meg, ahol $P$ jelöli a termékek, $R$ a nyersanyagok, $O$ pedig a rendelkezésre álló, gyártás során felhasználható műveleti egységek halmazát. A $P$ halmaz elemei azok az anyagok, amelyeket a folyamat során valamilyen szempont szerint optimálisan szeretnénk előállítani, az $R$ halmaz elemei pedig azok az anyagok, amelyek kiinduláskor, a gyártás megkezdésekor rendelkezésünkre

állnak. Matematikailag az $M$, $P$ és $R$ halmazok közötti reláció a következőképpen fejezhető ki:

$$P \subset M, R \subset M \text{ és } P \cap M = \emptyset. \tag{5.3}$$

Az $O$ halmaz elemei, azaz a műveleti egységek végzik az egyes anyagok közötti átalakításokat. Ennek megfelelően az $O$ halmaz és az $M$ halmaz kapcsolata:

$$O \subseteq \wp(M) \times \wp(M). \tag{5.4}$$

Egy műveleti egységet matematikailag egy rendezett párral modellezünk, azaz $(\alpha, \beta) \in O$, ahol $\alpha$ jelöli az egység bemenetei anyagainak, $\beta$ pedig a kimenetei anyagainak a halmazát. A fentiek illusztrálására, azaz egy gyártási folyamat megadására tekintsük az alábbi hipotetikus példát:

$$M := \{A, B, C, D, E, F, G, H, I, J, K\} \tag{5.5}$$

$$P := \{A\} \tag{5.6}$$

$$R := \{D, F, H, I\} \tag{5.7}$$

illetve

$$O := \{(\{B\}, \{A, E\}), (\{C\}, \{A, J\}), (\{D, E\}, \{B\}), (\{E, F\}, \{B\}), (\{F, G\}, \{C, K\}), \\ (\{H\}, \{E\}), (\{I, J\}, \{G\})\} = \{O_1, O_2, O_3, O_4, O_5, O_6, O_7\} \tag{5.8}$$

A fenti, 10 anyagot és 7 műveleti egységet tartamalmazó példában az $A$ anyagot szeretnénk gyártani a kiinduláskor rendelkezésre álló $D$, $F$, $H$, $I$ anyagokból. Azokat az anyagokat, amelyek se nem nyersanyagok, se nem végtermékek, mint a fenti példában $B$, $C$, $E$, $G$, $J$ nem szoktuk külön halmazként feltüntetni a feladat megadásakor.

Azokat az anyagokat, amelyek bizonyos műveleti egységek kimeneti, illetve bizonyos műveleti egységek bemenetei anyaghalmazának is elemei, köztes anyagoknak nevezzük. Tehát köztes anyagok azok az anyagok, amelyek kiinduláskor nem állnak rendelkezésünkre, azokat a gyártás során műveleti egységek állítják elő, illetve a termék előállításhoz fel is használják őket. A fenti példában $B$, $C$, $E$, $G$, $J$ is ilyen, a $G$ anyagot például az $O_7$ műveleti egység gyártja és $O_5$ használja fel.

Azokat az anyagokat, amelyeket a gyártás során műveleti egységek állítanak elő, de más műveleti egységek nem használják fel és nem elemei a $P$ halmaznak sem, melléktermékeknek nevezzük. A fenti példában ilyen melléktermék a $K$ anyag, amelyet $O_5$ gyárt, de nem használja fel egyetlen más műveleti egység sem, továbbá nem eleme a termékek halmazának sem.

## 5.1.2. P-gráf

Annak érdekében, hogy a lehetséges, gyakorlatban is megvalósítható megoldások kombinatorikus tulajdonságait ki tudjuk használni az optimális megoldás keresése során, a folyamat egyértelmű strukturális reprezentációja szükséges. A gyakorlatban általában használt hagyományos gráfok erre alkalmatlanok, az egyértelmű reprezentációhoz egy speciális, irányított páros gráf, az úgynevezett P-gráf (Process-graph, P-graph) szükséges. Egy gráf akkor páros, ha a csúcshalmaza partícionálható két diszjunkt halmazra úgy, hogy az azonos halmazban

lévő csúcsok közül egymással semelyik kettő sem szomszédos. A folyamat strukturájának reprezentálása során is ezt a tulajdonságot használjuk ki, mivel alapvetően két, anyag illetve műveleti egység típusú csúcsot különböztetünk meg és két azonos típusú csúcs nem kapcsolódhat egymáshoz közvetlenül.

A P-gráf matematikai definíciója a következő. Legyen $m$ és $o$ két véges halmaz, amelyekre

$$o \subseteq \wp(m) \times \wp(m). \tag{5.9}$$

Ekkor a P-gráf egy olyan $(m, o)$ pár, ahol a gráf csúcsai a

$$V = m \cup o \tag{5.10}$$

halmaz elemei. Az $m$ halmaz elemei az anyag típusú csúcsok, az $o$ halmaz elemei pedig a műveleti egység típusú csúcsok. A gráf élei az

$$A = A_1 \cup A_2 \tag{5.11}$$

halmaz elemei, ahol

$$A_1 := \{(x, y) | y = (\alpha, \beta) \in o \text{ és } x \in \alpha\} \tag{5.12}$$

és

$$A_2 := \{(y, x) | y = (\alpha, \beta) \in o s x \in \beta\}. \tag{5.13}$$

A fenti formális definícióban $x$ jelöli az anyag típusú csúcsokat, $y$ a műveleti egység típusú csúcsokat, $\alpha$ jelöli azon anyag típusú csúcsok halmazát, amelyekből mutat irányított él a műveleti egység típusú csúcsokba, $\beta$ pedig azon anyag típusú csúcsok halmazát, amelyekbe mutat irányított él a műveleti egység típusú csúcsokból. Más szavakkal mondva, az $A_1$ élhalmaz minden eleme anyag típusú csúcsokból műveleti egység típusú csúcsokba mutat, míg az $A_2$ élhalmaz minden eleme műveleti egység típusú csúcsból mutat anyag típusú csúcsba.

Két P-gráf, $(m_1, o_1)$ és $(m_2, o_2)$ unióját és metszetét, melyek szintén P-gráfok, a következőképpen definiáljuk:

$$(m_1, o_1) \cup (m_2, o_2) = (m_1 \cup m_2, o_1 \cup o_2) \tag{5.14}$$

$$(m_1, o_1) \cap (m_2, o_2) = (m_1 \cap m_2, o_1 \cap o_2) \tag{5.15}$$

Továbbá az $(m_1, o_1)$ gráf az $(m_2, o_2)$ gráf részgráfja, azaz

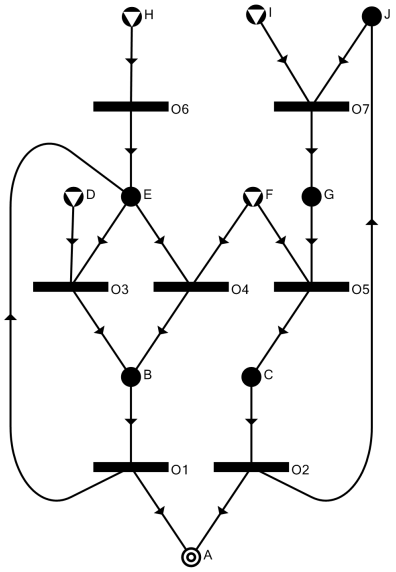
$$(m_1, o_1) \subseteq (m_2, o_2) \tag{5.16}$$

ha

$$m_1 \subseteq m_2 \text{ és } o_1 \subseteq o_2. \tag{5.17}$$

Egy folyamat struktúrája definiálható P-gráfként. Legyen $(P, R, O)$ egy szintézis feladat, legyen $m$ az $M$ anyaghalmaz részhalmaza, $o$ pedig az $O$ műveleti egység részhalmaza, továbbá tegyük fel, hogy $o \subseteq \wp(m) \times \wp(m)$ teljesül. Ekkor a rendszer struktúrája az $m$ anyaghalmaz, és az $o$ műveleti egység halmaz felhasználásával az $(m, o)$ P-gráffal formálisan definiálható. A definícióból közvetlenül következnek az alábbi tulajdonságok:

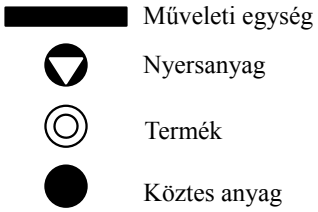
 




*5.4. ábra. A hálózatszintézis-feladathoz tartozó P-gráf*

1. Az anyagokat a gráf anyag típusú, a műveleti egységeket a műveleti egység típusú csúcsai reprezentálják.
2. Egy anyag és egy műveleti egység típusú csúcs között akkor és csak akkor megy él, ha a megfelelő anyag és a műveleti egység kapcsolata része a reprezentálandó folyamatnak.
3. Az élek irányai megegyeznek a folyamat előrehaladásának irányával.

Az anyag típusú csúcsokat körökkel, a műveleti egységek típusúakat vízsszintes téglalapokkal ábrázoljuk. A fenti példához tartozó P-gráf az 5.4 ábrán látható. A különböző anyag típusú csúcsok reprezentáció, mint nyersanyag, termék, vagy melléktermék az 5.5 ábrán láthatóak.




*5.5. ábra. Szimbólumok*

### 5.1.3. Kombinatorikusan lehetséges megoldásstruktúrák

A P-gráf segítségével nem csak a rendszer szintaktikája, hanem a szemantikája is kifejezhető. A gyakorlatban egy valós folyamat strukturája nem írható le egy tetszőleges P-gráffal: egy valós folyamat megfelelő reprezentáláshoz minden P-gráfnak teljesítenie kell bizonyos kombinatorikus tulajdonságokat. Ezeken a tulajdonságokon alapulva különböző axiómákat fogalmazhatunk meg.

Vannak bizonyos kombinatorikus tulajdonságok, amelyeket egy megoldásban szereplő műveleti egységeknek és nyersanyagoknak mindig teljesíteniük kell. Például ha egy struktúrában nincs kapcsolat a termékek és a nyersanyagok között, akkor a struktúra alkalmatlan gyakorlati folyamatok reprezentálására. Kulcsfontosságú tehát azoknak az általános kombinatorikus tulajdonságoknak a megtalálása, amelyeket egy folyamat struktúrájának rendelkeznie kell. Az így összegyűjtött tulajdonságokkal minden, a szintézis feladat lehetséges megoldásaihoz tartozó struktúrának rendelkeznie kell. Azok, és csakis azok a struktúrák lehetnek lehetséges megoldásstruktúrák amelyek ezekkel a tulajdonságokkal rendelkeznek; semmilyen egyéb struktúrát vagy feltételt nem kell figyelembe venni az optimális megoldás keresése során.

### 5.1.4. Axiómák

Az alábbi axiómák [5] fogalmazzák meg azokat a tulajdonságokat, amelyekkel egy valós, gyakorlatban megvalósítható folyamatot reprezentáló struktúrának rendelkeznie kell:

(S1) Minden legyártandó termék, azaz $P$ minden eleme szerepel a struktúrában.

(S2) Egy a struktúrában szereplő anyag akkor és csak akkor nyersanyag, ha egyetlen a struktúrában szereplő műveleti egység sem állítja elő.

(S3) Minden a struktúrában szereplő műveleti egység a szintézis feladatban definiált.

(S4) Minden a struktúrában szereplő műveleti egységből vezet út legalább egy legyártandó termékhez.

(S5) Ha egy $x$ anyag része a struktúrának, akkor létezik a struktúrában olyan műveleti egység, amelynek $x$ anyagot felhasználja vagy előállítja.

Önmagában mindegyik axióma triviálisnak tűnik, azonban azokat egyszerre alkalmazva kiszűrhetőek a kombinatorikusan nem megfelelő, gyakorlati alkalmazások modellezésére használhatatlan hálózatok. Ha egy szintézis feladathoz tartozó P-gráf kielégíti ezeket az axiómákat, akkor a P-gráfot a probléma egy megoldásstruktúrájának mondjuk. Tekintsük az alábbi szintézis feladatot a megoldásstruktúrák koncepciójának az illusztrálására:

$$M := \{A, B, C, D, E, F, G, H, I\} \tag{5.18}$$

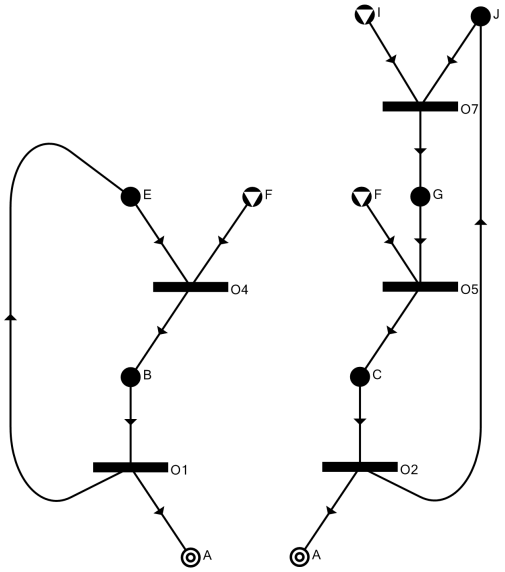
$$P := \{A\} \tag{5.19}$$

$$R := \{D, F, H\} \tag{5.20}$$

$$\begin{aligned} O := \{&(\{C\}, \{A, I\}), (\{B\}, \{A, E\}), (\{D, E\}, \{B\}), \\ &(\{E, F\}, \{B\}), (\{F, G\}, \{C\}), (\{H, I\}, \{G\}), \} \end{aligned} \tag{5.21}$$

A feladathoz tartozó két különböző megoldásstruktúra látható az 5.6 ábrán. Érdemes megjegyezni, hogy ha egy műveleti egység típusú csúcs része egy megoldásstruktúrát reprezentáló P-gráfnak, akkor a műveleti egység bemeneti és kimenetei anyaghalmazainak összes eleme is




*5.6. ábra. Két kombinatorikusan lehetséges megoldásstruktúra*

a gráfhoz tartozik. Az is említést érdemel, hogy egy megoldásstruktúra nem tartalmazza feltétlenül az $M$ anyaghalmaz összes elemét, illetve az $R$ nyersanyaghalmazból sem kell feltétlenül mindent felhasználnia a gyártás során.

Mivel a gyártandó termék, $A$, mindkét struktúrában szerepel, az első axióma mindkét esetben teljesül. A második axióma az első struktúrában az $F$, a második struktúrában az $F$ és $G$ csúcsok miatt teljesül, mivel nyersanyagként egyedül ezekbe a csúcsokba nem vezet él műveleti egységekből, azaz ezeket az anyagokat nem gyártja semmi. Az első struktúra két, a második pedig három műveleti egységet tartalmaz, amelyek mindegyike definiálva van a szintézisfeladatban, így a hármas axióma is teljesül. A négyes axiómának megfelelően, mindkét struktúrában minden műveleti egység típusú csúcsból vezet út a legyártandó termékhez. Az ötös axióma is teljesül, mivel mindkét struktúrában minden egyes anyag típusú csúcs legalább egy műveleti egységnek a kimenete, vagy a bemenete. Ez a két struktúra tehát kielégíti az összes axiómát, szemben az 5.7 ábrán látható struktúrával, amelyre az egyes, kettes, négyes és ötös axióma sem teljesül.

Legyen $S(P, R, O)$ az a halmaz, amely a $(P, R, O)$ szintézisfeladat összes megoldás-struktúráját tartalmazza. Ekkor a megoldás-struktúrákra vonatkozóan az alábbi tételt fogalmazhatjuk meg:

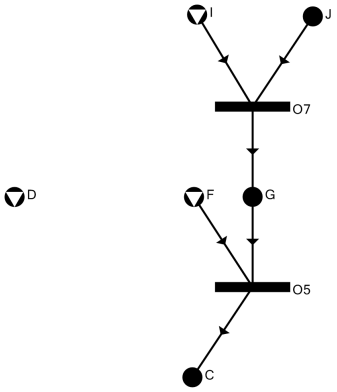
**1. Tétel.** *A megoldásstruktúrák zártak az unióra, azaz két megoldásstruktúra uniója is megoldásstruktúra. Formálisan: ha*

$$\sigma_1 \in S(P, R, O) \text{ és } \sigma_2 \in S(P, R, O) \tag{5.22}$$

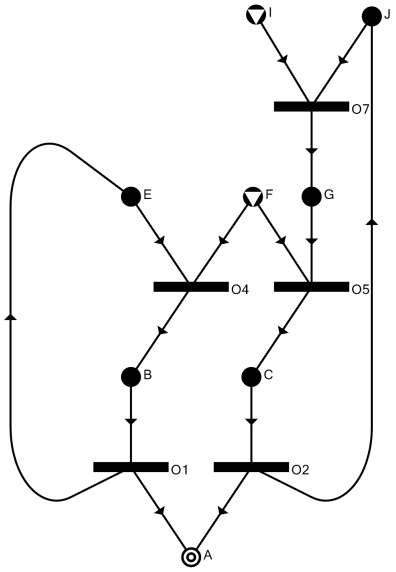
*akkor*

$$(\sigma_1 \cup \sigma_2) \in S(P, R, O). \tag{5.23}$$

Az előző példa két megoldásstuktúrájának unióját ábrázolja az 5.8 ábra. Mint az látható, a két megoldásstruktúra uniója maga is egy megoldásstruktúra.




*5.7. ábra. Ez a struktúra nem elégíti ki az axiómákat*




*5.8. ábra. Az 5.6 ábrán látható struktúrák uniója*

### 5.1.5. Maximális struktúra

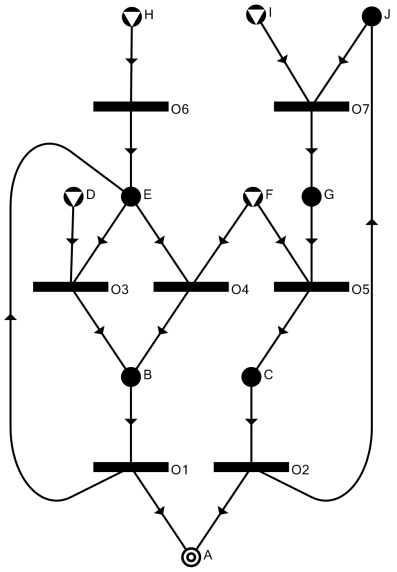
Mivel a megoldásstruktúrák halmaza véges és zárt az unióra, a halmaznak lesz egy eleme, $\mu(P, R, O)$, amely az összes megoldásstruktúra uniója, azaz

$$\mu(P, R, O) = \bigcup_{\sigma \in S(P,R,O)} \sigma \tag{5.24}$$

feltéve ha a megoldásstruktúrák halmaza nem üres, azaz $S(P, R, O)$ $\emptyset$. Ekkor a halmaz $\mu(P, R, O)$ elemét a $(P, R, O)$ szintézisfeladat maximális struktúrájának nevezzük. A szakirodalomban a maximális struktúrát gyakran nevezik szuperstruktúrának is. A maximális struktúra minden csúcsa és éle része legalább egy megoldás-struktúrának, illetve minden megoldásstruktúra P-gráfja részgráfja a maximális struktúrát reprezentáló P-gráfnak. A fent bevezetett példához tartozó maximális struktúra az 5.9 ábrán látható.

**2. Tétel.** *A maximális struktúra maga is megoldásstruktúra, azaz* $\mu(P, R, O) \in S(P, R, O)$.




*5.9. ábra. A szintézisfeladathoz tartozó maximális struktúra*

A maximális struktúra a feladat összes lehetséges megoldását tartalmazza, így az optimálisat is. Tulajdonképpen a maximális struktúra meghatározásával lecsökkentjük a keresési teret, mivel az optimális megoldás meghatározása során elegendő csupán a kombinatorikusan lehetséges megoldásstruktúrákat megvizsgálnunk. A korábban már említett, 35 műveleti egységes ipari példa esetében ez a csökkenés több mint 99,99%-os, mivel 34,539,738,647 helyett elegendő csupán 3465 lehetséges alternatívát megvizsgálni.

## 5.1.6. Az MSG algoritmus

A szintézis feladathoz tartozó maximális struktúrát a P-gráf módszertan az MSG (Maximal Structure Generation) algoritmus segítségével határozza meg. Az algoritmus a fent már ismertetett axiómákon alapszik, és számítógépes implementációja az alábbi négy főbb lépésből áll.

Az első lépésben definiáljuk magát a ($P$, $R$, $O$) szintézis-feladatot az $M$ anyaghalmaz, a $P$ termékhalmaz, az $R$ nyersanyaghalmaz illetve az $O$, a lehetséges műveleti egységek halmazának megadásával. Az $M$ halmaznak nem csak a köztes anyagokat, de a termékeket és a nyersanyagokat, azaz $P$ és $R$ elemeit is tartalmaznia kell.

A második lépésben meghatározunk egy kezdeti struktúrát, vagy kiindulási hálózatot úgy, hogy összekapcsoljuk a műveleti egységeket a hozzájuk tartozó közös anyag típusú csúcsok mentén.

A harmadik lépésben eltávolítjuk a hálózatból azokat az anyagokat és műveleti egységeket, amelyek nem lehetnek részei a maximális struktúrának, mert megsértik az axiómák valamelyikét. Ez az algoritmus redukciós szakasza. Kikerülnek azok a műveleti egységek, amelyek nyersanyagokat gyártanak, illetve azok az anyagok, amelyek nem számítanak nyersanyagnak, de mégsem állítja elő őket semmi. Ez iteratívan történik, mivel előfordulhat, hogy bizonyos anyagok és műveleti egységek eltávolítása újabb anyagok és műveleti egységek eltávolítását vonja maga után.

A végső fázis az algoritmus építő szakasza. Ebben a fázisban lépésről lépésre építjük fel a hálózatot a termékektől kiindulva először azokat a műveleti egységeket hozzáadva, amelyek a termékeket gyártják, majd azokat a műveleti egységeket, amelyek ezek bemeneteit gyártják, egészen addig, amíg el nem érünk a nyersanyagokig. Ha szemléletesen szeretnénk fogalmazni, akkor a lebontás felülről lefelé, a nyersanyagoktól a termékek felé haladva, az építés pedig alulról fölfelé, azaz a termékektől a nyersanyagok felé haladva történik. A szuperstruktúrát persze nem elég önmagában meghatározni, hanem lehetőleg minél hatékonyabban szeretnénk ezt megtenni. Az MSG ennek is eleget tesz, mivel a futási ideje polinomiális. Az MSG algoritmus működését az alábbi hipotetikus példán szemléltetjük:

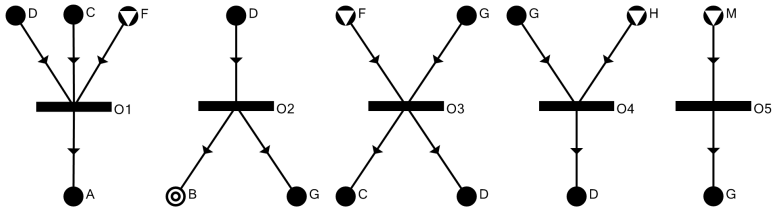
$$M = \{A, B, C, D, F, G, H, L, M\} \tag{5.25}$$

$$P = \{B\} \tag{5.26}$$

$$R = \{F, H, L, M\} \tag{5.27}$$

$$O = \{(\{C, D, F\}, \{A\}), (\{D\}, \{B, G\}), (\{F, G\}, \{C, D\}), (\{G, H\}, \{D\}), (\{M\}, \{G\})\} = \{O_1, O_2, O_3, O_4, O_5\} \tag{5.28}$$

A gyártás során a $B$ terméket szeretnénk előállítani a rendelkezésre álló $F$, $H$, $L$, $M$ nyersanyagokból. Definíciónak megfelelően, $P$ és $R$ elemei szerepelnek az $M$ halmazban is. A példában szereplő 5 műveleti egység látható az 5.10 ábrán.



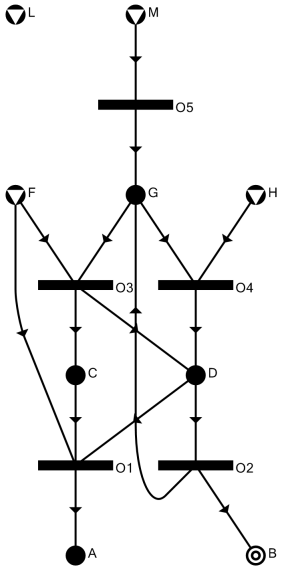

*5.10. ábra. A példához tartozó műveleti egységek*

A kiindulási hálózatot a műveleti egységek közös kimeneteik és bemeneteik összekapcsolásával kapjuk. Ebben a példában $O_3$-at és $O_1$-et $C$-n keresztül, $O_3$-at és $O_4$-et $O_1$-hez és $O_2$-höz $D$-n keresztül végül $O_2$-t és $O_5$-öt $O_3$-hoz és $O_4$-hez $G$-n keresztül kapcsoljuk össze. Az eredményül kapott kiindulási hálózat az 5.11 ábrán látható.

A redukciós szakaszban az $L$ nyersanyagot kivesszük a struktúrából, mert sérti az ötös axiómát, illetve az $O_1$ műveleti egységet is, mert sérti a négyes axiómát. Az $O_1$ műveleti egység eltávolításából következik, hogy az $A$ anyag sem marad része a struktúrának, mivel nem lesz, ami előállítja. A többi anyag és műveleti egység nem sért egyetlen axiómát sem, így részei maradnak a struktúrának.

Az eredményül kapott struktúra az 5.12 ábrán látható, amely megfelel a maximális struktúrának is, amit az algoritmus építő szakaszának végrehajtása után kapunk.

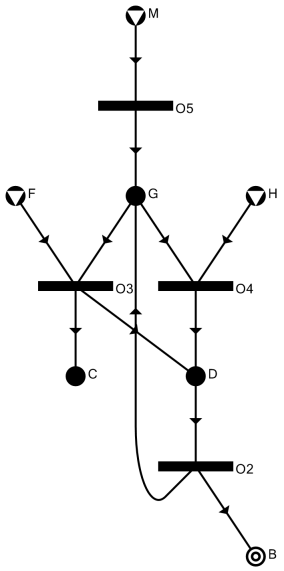
 




*5.11. ábra. A szintézisfeladathoz tartozó kiindulási struktúra*

### 5.1.7. Az SSG algoritmus

Az MSG eredményeként kapott maximális struktúra tartalmazza az összes kombinatorikusan lehetséges megoldásstruktúrát, amelyek alkalmasak annak a gyártási folyamatnak a modellezésére, amely során a megadott termékeket állítjuk elő a rendelkezésre álló nyersanyagokból. A maximális struktúra tartalmazza többek között a megadott célfüggvény, általában a költség szerinti optimális megoldást is. Nincs azonban általánosan elfogadott szabály azzal kapcsolatban, melyik megoldás az optimális, például nem biztos, hogy az a megoldásstruktúra állítja




*5.12. ábra. A szintézisfeladathoz tartozó maximális struktúra*

elő a terméket minimális költséggel, amely a legkevesebb műveleti egységet tartalmazza, így ebben a fázisban még mindenképpen a megoldásstruktúrák további vizsgálata szükséges.

A további vizsgálathoz ad segítséget az SSG (Solution Structure Generation) algoritmus, amely minden kombinatorikusan lehetséges struktúrát pontosan egyszer generál. Az algoritmusnak többféle számítógépes megvalósítása is létezik.

Az egyik megvalósítás a döntési leképezéseken alapszik, az alábbiakban ennek a vázlatát ismertetjük. A döntési leképezések során arról döntünk, hogy mely anyagot mely műveleti egységgel vagy egységekkel gyártunk, azaz mely műveleti egységeket vonjuk be egy adott megoldás-struktúrába. Ebből következőleg a döntési leképezések során arról is döntünk, mely műveleti egységeket zárjuk ki az adott struktúrából. A döntések során ügyelnünk kell a konzisztenciára is, ha egy műveleti egységről már döntöttünk egy anyagra vonatkozóan, hogy nem kerül be a struktúrába, akkor egy másik anyagra vonatkozó döntés során már nem választhatjuk be. Egy műveleti egységnek, ha szerepel a struktúrában, minden kimeneti anyagát elő kell állítania; egy inkonzisztens döntés azt eredményezné, hogy bizonyos anyagokat előállít, bizonyos anyagokat pedig nem. Ez a gyakorlatban nem megengedett, például sok vegyipari anyag előállításakor keletkeznek káros melléktermékek, amelyek a gyártás velejárói. Az SSG algoritmus döntési leképezésen alapuló megvalósítása rekurzívan hívja magát. Az algoritmus működését az alábbi hipotetikus példán illusztráljuk:

$$M = \{A, B, C, D, E, F, G\} \tag{5.29}$$

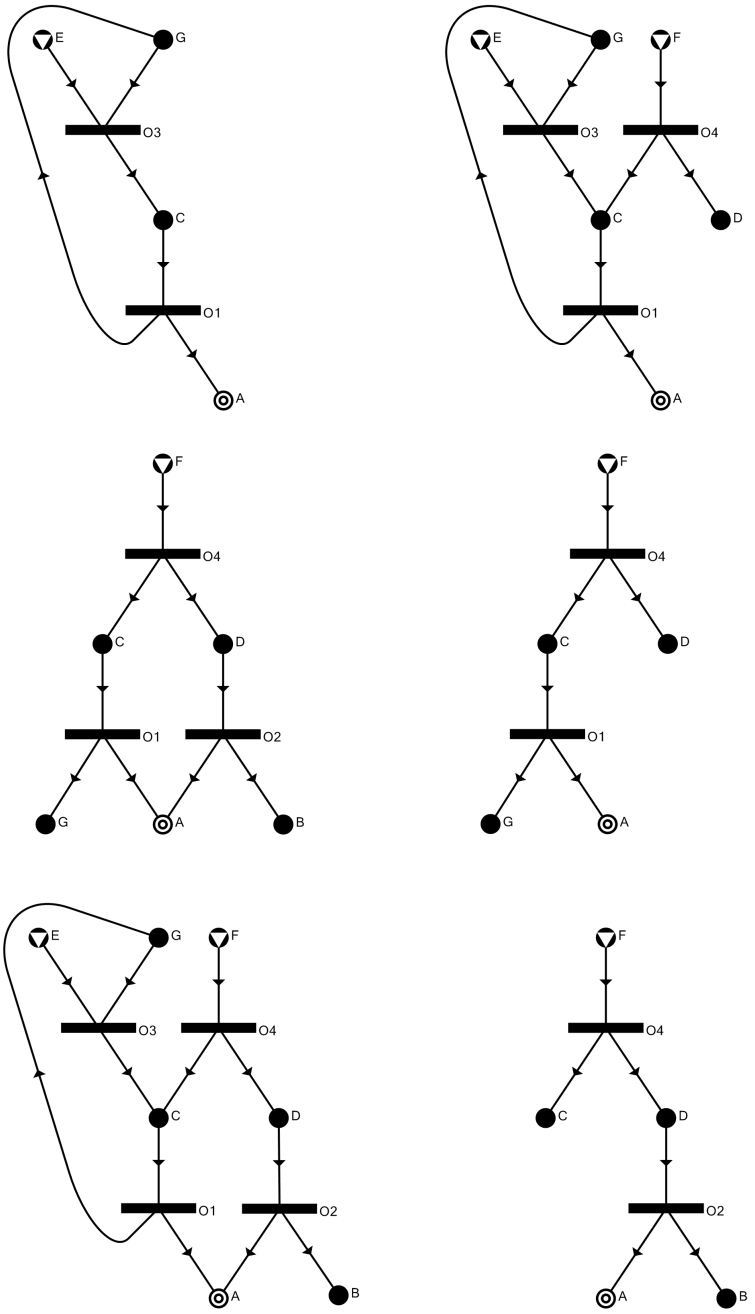
$$P = \{A\} \tag{5.30}$$

$$R = \{E, F\} \tag{5.31}$$

$$\begin{aligned} O &= \{(\{C\}, \{A, G\}), (\{D\}, \{A, B\}), (\{E, G\}, \{C\}), (\{F, C\}, \{D\})\} = \\ &= \{O_1, O_2, O_3, O_4\} \end{aligned} \tag{5.32}$$

Az SSG a megadott szintézisfeladat összes kombinatorikusan lehetséges megoldásstruktúráját, közte a maximális struktúrát, amely maga is megoldásstruktúra, pontosan egyszer generálja. A generált megoldás-struktúrák az 5.13 ábrán láthatóak.




*5.13. ábra. Az SSG által generált megoldás-struktúrák*

### 5.1.8. A vegyes-egész matematikai programozási modell

Eddig csak olyan eszközeit ismertettük a P-gráf módszertannak, amely strukturális szempontból vizsgálja a folyamatszintézis feladatokat. Az alábbiakban ismertetjük a módszertan által használt vegyes-egész matematikai programozási modellt is.

A modellben szereplő folytonos változókat $x$-el, a bináris változókat pedig $y$-al jelöljük. Ez a jelölés más, általános MILP modellek esetén is elfogadott a szakirodalomban, bár ott a bináris változók jelölésére gyakran használják a görög $\delta$ betűt is. Ezeket a változókat a műveleti egységekhez rendeljük hozzá. Az $x_i$ folytonos változó jelöli az $O_i$ műveleti egység méretét vagy kapacitását, az $y_i$ bináris változó pedig azt, hogy az adott műveleti egység szerepel-e a struktúrában, vagy sem: ha az $y_i \in \{0,1\}$ bináris változó értéke 0, akkor az $O_i$ műveleti egység nem szerepel a struktúrában, ha pedig 1, akkor igen.

Ha az $O_i$ műveleti egység része a struktúrának, azaz $y_i = 1$, akkor a műveleti egység kapacitását reprezentáló $x_i$ folytonos változó bármilyen értéket felvehet 0, és a műveleti egység felső kapacitáskorlátja, $U_i$ között. Formálisan:

$$x_i \leq y_i U_i \tag{5.33}$$

ahol $U_i$ az $O_i$ műveleti egység kapacitására vonatkozó felső korlát. Ha a feladatban nincs ilyen korlát definiálva, akkor $U_i$ helyett egy tetszőleges, megfelelően nagy $M$ szám használható.

A célfüggvényt a költség minimalizálására írjuk fel. A költség a beruházási költségek összegéből, a műveleti egységek működtetésének a költségéből, illetve a nyersanyagok árából tevődik össze. Ezek az összetevők lefedik a hálózat, azaz a szintetizálandó folyamat teljes költségét. A modellben a műveleti egységek költségét egyszerűen az $a + bx$ összefüggéssel adjuk meg, ahol $x$ jelöli az adott műveleti egység méretét vagy kapacitását, $a$ a fix költséget, $b$ pedig a proporcionális költséget.

*5.1. táblázat. A műveleti egységek kimeneti és bemeneti anyagai*

| Műveleti egységek | Bemenetek | Kimenetek |
|---|---|---|
| $O_1$ | C(5) | A(4),G(1) |
| $O_2$ | D(9) | A(8),B(1) |
| $O_3$ | E(4),G(1) | C(5) |
| $O_4$ | F(10) | C(1),D(9) |

*5.2. táblázat. A műveleti egységek költségparaméterei*

| Műveleti egységek | Fixköltség | Arányos költség |
|---|---|---|
| $O_1$ | 4 | 1 |
| $O_2$ | 3 | 1 |
| $O_3$ | 2 | 1 |
| $O_4$ | 2 | 0,5 |

A fent már említett, $x_i \leq y_i U_i$ típusú korlátozási feltételek mellett további feltételeket szabunk meg az anyagegyensúlyokra, a termékekre, illetve a nyersanyagokra vonatkozóan. A termékekre általában alsó korlátokat szoktunk megadni, amellyekkel azt szabjuk meg, mennyit

kell legalább az adott termékekből előállítanunk, míg a nyersanyagokra felső korlátokat adhatunk, amennyiben az adott típusú nyersanyagból nem áll rendelkezésre korlátlanul felhasználható mennyiség. Az anyagegyensúly feltételeket a köztes anyagokra kell definiálni. Ezek a feltételek azt fejezik ki, hogy minden köztes anyagból legalább annyit kell előállítani, amennyi az őket felhasználó műveleti egységek működéséhez szükséges, mivel enélkül a gyártás folyamata megakadna.

*5.3. táblázat. A nyersanyagok árai*

| Nyersanyag | Ár |
|---|---|
| E | 0,8 |
| F | 1,6 |

Írjuk fel a vegyes-egész matematikai programozási modellt az előző, SSG szemléltetését bemutató példára. A modell felírásához önmagában nem elegendő az az információ, hogy melyik műveleti egység milyen anyagokat használ fel illetve állít elő, hanem az is szükséges, hogy miből mennyit. Ezt az információt tartalmazza az 5.1 táblázat, az anyagok neve mellett zárójelben feltüntetve. A műveleti egységek költségparaméterei az 5.2 táblázatban találhatóak, a nyersanyagok árait az 5.3 táblázat tartalmazza, míg a termékekre és nyersanyagokra vonatkozó megkötések az 5.4 táblázatban láthatóak. Az adatok alapján az alábbi matematikai programozási modell írható fel:

*5.4. táblázat. A termékekre és nyersanyagokra vonatkozó feltételek*

| Termék | Feltétel |
|---|---|
| A | $\geq 4$ |
| Nyersanyag | Feltétel |
| E | $\leq 10$ |

A feladathoz tartozó költségfüggvény:

$$\min \quad x_1 + 4y_1 + x_2 + 3y_2 + 4.2x_3 + 2y_3 + 16.5x_4 + 2y_4 \tag{5.34}$$

Az anyagegyensúly feltételek a köztes anyagokra:

$$C \text{ anyag:} \quad -5x_1 + 5x_3 + x_4 \geq 0 \tag{5.35}$$

$$D \text{ anyag:} \quad -9x_2 + 9x_4 \geq 0 \tag{5.36}$$

$$G \text{ anyag:} \quad x_1 - x_3 \geq 0 \tag{5.37}$$

A termékre vonatkozó feltétel:

$$A \text{ termék:} \quad 4x_1 + 8x_2 \geq 4 \tag{5.38}$$

A nyersanyagokra vonatkozó megkötések:

$$E \text{ nyersanyag:} \qquad 4x_3 \qquad \leq 10 \tag{5.39}$$

További, a műveleti egységekre vonatkozó feltételek:

$$O_1: \quad x_1 \leq y_1 M \tag{5.40}$$
$$O_2: \quad x_2 \leq y_2 M \tag{5.41}$$
$$O_3: \quad x_3 \leq y_3 M \tag{5.42}$$
$$O_4: \quad x_4 \leq y_4 M \tag{5.43}$$

Az $F$ nyersanyagra vonatkozóan nem adtunk meg korlátozó feltételt.

### 5.1.9. Az ABB algoritmus

Mint vegyes-egész programozási feladatnak, ennek a modellnek a megoldására is használhatóak az általános branch-and-bound típusú módszerek. Bár a feladat optimális megoldása ezekkel a módszerekkel is meghatározható, hatékonyságuk még tovább javítható, mivel a megoldás keresése során nem veszik figyelembe a szintézis feladatok speciális tulajdonságait. A P-gráf módszertan az optimális megoldás meghatározására ennek megfelelően egy speciális korlátozás és szétválasztás típusú algoritmust, az ABB-t (Accelerated Branch-and-Bound) használja.

Az ABB algoritmusnak az SSG algoritmushoz hasonlóan többféle implementációja létezik. Az egyik megvalósítás a már SSG-nél is említett döntési leképezéseken alapszik. Ennek a megközelítésnek az az előnye, hogy az ABB csak a kombinatorikusan lehetséges struktúrákat vizsgálja az optimális megoldás keresése során. Az algoritmus alkalmaz egyéb kombinatorukis gyorsításokat is, mint például az úgynevezett neutrális kiterjesztést.

## 5.2. Nevezetes feladatok megoldása folyamatszintézissel

A folyamatszintézis eszközeivel megoldhatóak azok a feladatok is, melyek elemi gráf algoritmusokkal. A gráf leképezhető folyamatgráfként, a keresés célja pedig optimalizálási célként.

A gráf minden esetben hasonló módon írható át a folyamatszintézis nyelvére. A csúcsokhoz anyagokat, az irányított élekhez pedig műveleti egységeket rendelünk. A csúcsok egy {1, 2, 3, 4, 5, 6}halmazát például az $M = \{m1, m2, m3, m4, m5, m6\}$ anyaghalmaz jelöli, míg az irányított élek egy {(1, 2), (1,3), (2,3)} halmazát az $O = \{o12 = (\{m1\}, \{m2\}), o13 = = (\{m1\}, \{m3\}), o23 = (\{m2\}, \{m3\})\}$ műveleti egység halmaz. Irányítatlan gráf esetén az élek mindkét lehetséges irányát külön-külön rögzíteni kell. Tehát irányítatlan élek egy {{1, 2}, {1,3}} halmazát az $O$={($\{m1\}$, $\{m2\}$), ($\{m2\}$, $\{m1\}$), ($\{m1\}$, $\{m3\}$), ($\{m3\}$, $\{m1\}$)} műveleti egység halmaz adja meg. Az algoritmusok célját az anyagokra és műveleti egységekre vonatkozó korlátokkal és költség paraméterekkel tudjuk kijelölni.

### 5.2.1. Minimális feszítőfa szintézise

Minimális feszítőfa egy olyan összefüggő gráf, mely minden csúcsot tartalmaz, és az élek súlyának összege minimális. A gráf összefüggőségét biztosítja, ha olyan folyamot szintetizálunk, melyben egyik csúcsot forrásként azaz nyersanyagként, minden más csúcsot pedig

*5.5. táblázat. Anyagok*

| Név | Típus | Alsó korlát | Felső korlát |
|---|---|---|---|
| m1 | nyersanyag | | 5 |
| m2 | termék | 1 | |
| m3 | termék | 1 | |
| m4 | termék | 1 | |
| m5 | termék | 1 | |
| m6 | termék | 1 | |

nyelőként azaz kötelező termékként definiálunk. A termékekre szükséges pozitív mennyiségi alsó korlát beállítása, hogy a hozzá vezető műveletek legalább egyikének mérete ne lehessen nulla. Ezután az élek súlyát, mint a leíró műveleti egységek fix költségét adjuk meg. Kruskal algoritmus bemutatásánál szereplő feladat folyamatszintézis átírása az 5.5 és 5.6 táblázatokban látható.

*5.6. táblázat. Műveleti egységek*

| Név | Bemenő anyag | Kimenő anyag | Fix költség |
|---|---|---|---|
| o12 | m1 | m2 | 2 |
| o21 | m2 | m1 | 2 |
| o13 | m1 | m3 | 3 |
| o31 | m3 | m1 | 3 |
| o23 | m2 | m3 | 1 |
| o32 | m3 | m2 | 1 |
| o24 | m2 | m4 | 4 |
| o42 | m4 | m2 | 4 |
| o25 | m2 | m5 | 3 |
| o52 | m5 | m2 | 3 |
| o35 | m3 | m5 | 2 |
| o53 | m5 | m3 | 2 |
| o45 | m4 | m5 | 3 |
| o54 | m5 | m4 | 3 |
| o46 | m4 | m6 | 6 |
| o64 | m6 | m4 | 6 |
| o56 | m5 | m5 | 1 |
| o65 | m6 | m5 | 1 |

A feszítőfát azok az élek adják, melyekhez tartozó műveleti egységek szerepelnek a szintézis feladat optimális megoldásában. Ilyen megoldást ad például az ABB algoritmus szoftver megvalósítása.

*5.7. táblázat. Anyagok*

| Név | Típus | Alsó korlát | Felső korlát |
|---|---|---|---|
| m1 | nyersanyag | | 5 |
| m2 | köztes anyag | | |
| m3 | köztes anyag | | |
| m4 | köztes anyag | | |
| m5 | köztes anyag | | |
| m6 | termék | 1 | |

## 5.2.2. Legrövidebb út szintézise

A legrövidebb út hasonlóan határozható meg mint a feszítőfa, azzal a különbséggel, hogy a legrövidebb úthoz csak a kiindulási és a cél csúcsot összekötő gráfot kell meghatározni, melyben az élek súlyának összege minimális. Ennek megfelelően, a folyamatszintézis feladatban a kiindulási csúcsot adjuk meg nyersanyagként, a cél csúcsot pedig kötelező termékként valamely pozitív előállítandó mennyiséggel. Az élek súlyát ismét a műveleti egységek fix költ-

*5.8. táblázat. Műveleti egységek*

| Név | Bemenő anyag | Kimenő anyag | Fix költség |
|---|---|---|---|
| o12 | m1 | m2 | 1 |
| o21 | m2 | m1 | 1 |
| o13 | m1 | m3 | 3 |
| o31 | m3 | m1 | 3 |
| o23 | m2 | m3 | 1 |
| o32 | m3 | m2 | 1 |
| o24 | m2 | m4 | 4 |
| o42 | m4 | m2 | 4 |
| o25 | m2 | m5 | 3 |
| o52 | m5 | m2 | 3 |
| o35 | m3 | m5 | 2 |
| o53 | m5 | m3 | 2 |
| o45 | m4 | m5 | 3 |
| o54 | m5 | m4 | 3 |
| o46 | m4 | m6 | 6 |
| o64 | m6 | m4 | 6 |
| o56 | m5 | m5 | 1 |
| o65 | m6 | m5 | 1 |

ségeként definiáljuk. A Dijkstra algoritmusnál bemutatott példára a folyamatszintézis feladat az 5.7 és 5.8 táblázatokban látható.

A legrövidebb utat azok az élek adják, melyekhez tartozó műveleti egységek szerepelnek a szintézis feladat optimális megoldásában. Ilyen megoldást ad például az ABB algoritmus szoftver megvalósítása.

### 5.2.3. Maximális folyam szintézise

A maximális folyamoknál a gráf éleihez rendelt tulajdonság nem a költség, hanem a kapacitás. Ezen kapacitás mellett kell a lehető legnagyobb mennyiséget egy forrásból a nyelőbe juttatni. A gráf leírása hasonló az előző két alfejezetben tárgyaltakhoz, de az éleket reprezentáló műveleti egységeknek itt nem költsége lesz, hanem maximális kapacitása, mely megegyezik az megfelelő él kapacitásával. A maximális folyamot az motiválja, hogy a nyelőn megjelenő anyagmennyiséget egy olyan termékkel írjuk le, aminek van egy pozitív ára, tehát a minél nagyobb termelése, egyre nagyobb profitot hoz. Annak érdekében, hogy az éleken ne folyjon olyan anyagmennyiség, melynek nincs szerepe a maximális folyamban, mert csak közbülső csúcshoz vezet, korlátozzuk nullára a közbülső anyagpontokon megmaradó megmaradó anyagmennyiséget az anyag felső korlátjával. A Ford-Fulkerson algoritmusnál megismert példát megadó szintézis feladat learása az 5.9 és 5.10 táblázatokban látható.

*5.9. táblázat. Anyagok*

| Név | Típus | Alsó korlát | Felső korlát | Ár |
|---|---|---|---|---|
| m1 | nyersanyag | | $\infty$ | 0 |
| m2 | köztes anyag | | 0 | |
| m3 | köztes anyag | | 0 | |
| m4 | köztes anyag | | 0 | |
| m5 | köztes anyag | | 0 | |
| m6 | termék | | $\infty$ | 1 |

A maximális folyamot a szintézis feladat optimális megoldásában szereplő műveleti egységek által reprezentált élek adják, felhasznált kapacitásuk pedig megegyezik a megfelelő műveleti egységek optimális kapacitásával. Ilyen megoldást ad például az ABB algoritmus szoftver megvalósítása.

*5.10. táblázat. Műveleti egységek*

| Név | Bemenő anyag | Kimenő anyag | Kapacitás felső korlát |
|---|---|---|---|
| o12 | m1 | m2 | 1 |
| o13 | m1 | m3 | 3 |
| o32 | m3 | m2 | 1 |
| o24 | m2 | m4 | 4 |
| o25 | m2 | m5 | 3 |
| o35 | m3 | m5 | 2 |
| o45 | m4 | m5 | 3 |
| o46 | m4 | m6 | 6 |
| o56 | m5 | m5 | 1 |

## 5.3. Demonstrációs szoftverek

Ebben a szekcióban két olyan szoftvert ismertetünk, amelyekkel folyamathálózat-szintézis feladatokat fogalmazhatunk meg, modellezhetünk le, illetve meghatározhatjuk a létrehozott feladatok optimális megoldását. A szoftverek a www.p-graph.com oldalról bárki számára ingyenesen hozzáférhetőek, továbbá a jegyzet mellékletén megtekinthető egy videó a szoftverek működéséről.

### 5.3.1. PNS Draw

A PNS Draw egy rajzolóprogram, amely kimondottan P-gráfok rajzolására szolgál. Bár P-gráfok rajzolására használhatóak egyéb, tetszőleges rajzprogramok is, a gyakorlati tapasztalatok azt mutatják, hogy ezek használata a legtöbbször nehézkes, kezelésük pedig nem feltétlenül felhasználóbarát, aminek köszönhetően az egyszerűbb P-gráfok rajzolása is időigényes tevékenységgé válhat.

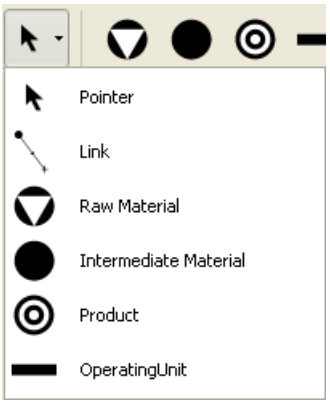
A PNS Draw futtatásához legalább egy 800 MHz-es Pentium III-as (vagy azzal egyenértékű) processzor és 256 Mbyte RAM szükséges az alábbi operációs rendszerek valamelyikével: Windows 2000 SP3, Windows XP SP2, Windows Vista vagy Windows 7. Az első két operációs rendszer esetében ügyelnünk kell arra is, hogy legyen a számítógépen .NET 2.0 is.

Indítás után azt látjuk, hogy a program felületének nagy részét elfoglalja a rajzolófelület (bár ez állítható), ahol a szerkesztés és rajzolás nagy része történik, felette található egy eszköztár, az eszköztár felett pedig egy menüsor. A rajzolófelület mellett két ablak található, az egyik az objektumok tulajdonságainak, a másik pedig a gyorsnézet megjelenítésére szolgál.

A rajzolás a lehető legegyszerűbb módon, drag'n'drop történik: kiválasztjuk az eszköztárból a kívánt típusú P-gráf csúcsot, ami lehet nyersanyag, köztes anyag, termék vagy műveleti egység, rákattintunk, majd a gomb nyomva tartása mellett lehúzzuk a rajzolófelületre, ahol szabadon pozícionálhatjuk. A rácsozás és a rácsra illesztés alapból aktív, ezeket a view menüben tudjuk kikapcsolni. A csúcsok összekapcsolásához át kell állítanunk az eszköztáron a drawing mode-ot link-re. Ezután rá kell kattintanunk a kiindulási csúcsra, majd arra a csúcsra,

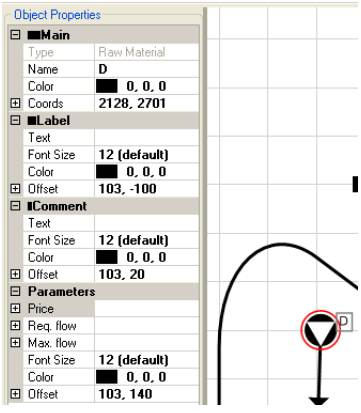
amelyikkel össze szeretnénk kötni. Amint ez megtörtént, létrejön egy irányított él, amely az első csúcsból mutat a másodikba. Az élen lévő nyíl mindig afelé a csúcs felé fog mutatni, amelyiket másodiknak választottuk. A szoftver nem enged azonos típusú csúcsokat összeköt-ni: anyag típusú csúcsot csak műveleti egységgel, műveleti egységet pedig csak anyag típusú csúccsal fog összekötni.




*5.14. ábra. A drawing mode kiválasztása*

Ahhoz, hogy az objektumok tulajdonságait szerkeszteni tudjuk, a drawing mode-ot pointer-re kell állítanunk, csak így tudjuk kijelölni az objektumokat. Ezután a különböző objektumokra kattintva, az object properties ablakban tudjuk szerkeszteni a tulajdonságaikat. A különböző objektumok különböző tulajdonságokkal rendelkeznek.

Anyagokra kattintva, tudjuk szerkeszteni a megjelenítéssel kapcsolatos tulajdonságokat, mint például név, szín, betűméret, stb., illetve meg tudunk adni a folyamatszintézis feladattal kapcsolatos tulajdonságokat is, mint például ár, alsó és felső korlát az előállítandó mennyiségre, stb. Utóbbi tulajdonságok alapból nem láthatóak az ábrán, itt található az az opció is, ami ezeket láthatóvá teszi.



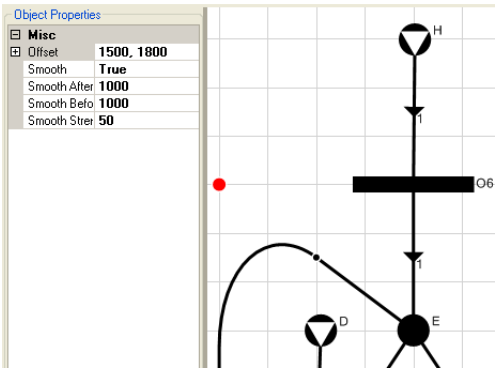

*5.15. ábra. Egy anyag tulajdonságainak szerkesztése*

A műveleti egységek megjelenítéssel kapcsolatos tulajdonságai megegyeznek az anyagokéval, a folyamatszintézis feladat szempontjából lényeges paramétereik azonban nem: itt a

feladatnak megfelelően alsó és felső korlátot, fix és változó költséget, stb. tudunk szerkeszteni. Alapbeállítások mellett ezek sem láthatóak, azonban láthatóvá tehetőek.

A PNS Draw-ban az élek is objektumoknak számítanak, amelyeknek szerkeszthető tulajdonságaik vannak. Egy élre kattintva szerkeszthetjük annak színét, a rajta elhelyezkedő nyíl helyzetét, illetve a rajta megjelenő szöveget is, amely alapból a rajta átmenő flow értéke. Ebből következik, hogy azt, hogy egy műveleti egység mennyit használ fel egy adott bemeneti anyagból, vagy mennyit állít elő egy kimeneti anyagból, az őket összekötő élen kell megadni, a rate paraméternél.

Az éleken lehetőség van töréspontok elhelyezésére, ezáltal pedig görbék létrehozására is. Kétféle töréspontot különböztetünk meg, mégpedig ideiglenes és állandó töréspontokat. Az ideiglenes töréspontokat apró körök, az állandó töréspontokat nagyobb körök jelzik. Nagyobb töréspontokat a kisebbekre kattintva, majd azokat mozgatva tudunk létrehozni. A töréspontoknak is vannak szerkeszthető tulajdonságai, ezekkel elsősorban az ív „simaságát” tudjuk befolyásolni.




*5.16. ábra. Töréspontok elhelyezésével lehetőség van görbék létrehozására is*

Az elkészített ábrákat png, svg és pns formátumokba tudjuk exportálni. Ezek a lehetőségek a file menü alatt találhatóak. Az exportálandó kép pontos testreszabását az Edit menü Settings beállításai alatt tudjuk elvégezni, a Print View-hez kapcsolódó checkmarkok segítségével. Exportálás előtt ne felejtsük el ellenőrizni, hogy az eszköztár jobb oldalán az Original Problem, vagy a Print View van-e kiválasztva. A pns formátumba exportált feladatot a PNS Studio tudja betölteni.

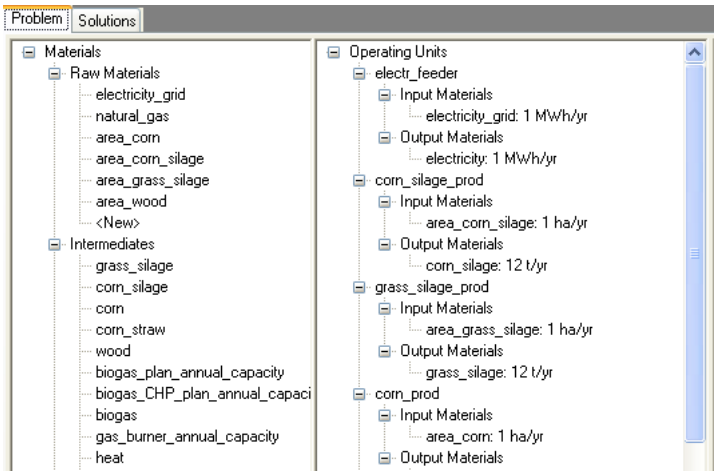
### 5.3.2. PNS Studio

A PNS Studio egy olyan szoftver, amelyben folyamathálózat-szintézis feladatokat tudunk definiálni, majd a definiált feladatokat a szoftverben implementált P-gráf módszertan algoritmusokkal megoldani. Rendszerkövetelménye megegyezik a PNS Draw rendszerkövetelményével.

A grafikus felület nagy részét itt is kitölti a szerkesztőfelület, ahol a feladatokat definiálni és szerkeszteni tudjuk. A szerkesztés itt is próbál a lehető legegyszerűbb, drag'n'drop típusú lenni. A felületet alapból négy részre oszlik, balról jobbra haladva ezek a következők: az anyagok listája, a műveleti egységek listája, az anyagok tulajdonságai, a műveleti egységek

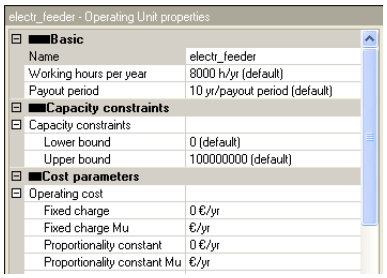
tulajdonságai. Az utóbbi kettő új feladat definiálásakor üres. A felület felépítésének köszönhetően jól áttekinthetjük a feladatot, ami jelentősen megkönnyíti a szerkesztés folyamatát.




*5.17. ábra. Az anyagok és műveleti egységek hierarchikus megjelenítése*

Új feladatot kétféleképpen tudunk létrehozni: vagy beimportálunk egy, a PNS Draw-ban készült rajzot, vagy kézzel kezdünk el szerkeszteni, anyagokat és műveleti egységeket a PNS Studio szerkesztőfelületén hozzáadva. A beimportált feladatot módosíthatjuk, illetve rögtön futtathatjuk is rá a kiválasztott folyamathálózat-szintézis algoritmusokat.

Kézi szerkesztés esetén új anyagokat és műveleti egységeket az anyagok és műveleti egységek felsorolásánál tudunk hozzáadni. Az anyagok listáján belül a szoftver megkülönbözteti a termékeket, köztes anyagokat illetve nyersanyagokat, amit tükröz a lista hierarchikus felépítése is. Új anyagot a megfelelő helyen a <New>-ra kattintva tudunk hozzáadni. Ekkor a felület eddigi üres részén megjelenik egy táblázat, amelyben az új anyag tulajdonságait szerkeszthetjük, mint például ár, előállítandó mennyiség, rendelkezésre álló mennyiség, stb. Új objektum hozzáadásánál a szoftver segít abban, hogy az objektum neve egyedi legyen, mivel nem engedi meg azonos nevű objektumok definiálását.

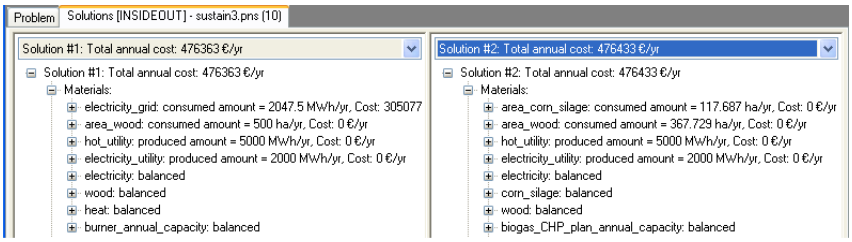



*5.18. ábra. Tulajdonságok szerkesztése*

A műveleti egységek hozzáadása és tulajdonságaiknak szerkesztése hasonló módon történik. Az anyagokat drag'n'drop tudjuk a műveleti egységekhez rendelni: rákattintunk egy anyagra, majd azt egy műveleti egység kimeneti vagy bemeneti anyagaihoz húzzuk.

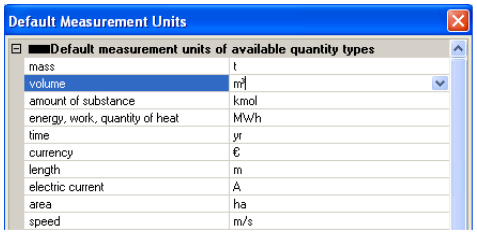
Ha elkészültünk a szerkesztéssel, akkor a Synthesize menüből kiválaszthatjuk a futtatni kívánt folyamathálózat-szintézis algoritmust. A szoftverben az MSG, SSG és ABB algorit-

musok mellett van egy SSG+LP opció is. Itt minden kombinatorikusan lehetséges megoldásstruktúra költségét kiszámolja a szoftver, irányított keresés nélkül. A szoftverben implementált ABB képes az optimális hálózat mellett az N-legjobb hálózat meghatározására is. Ennek gyakorlati szempontból lehet jelentősége, előfordulhat, hogy valamilyen egyéb előnyös tulajdonsága miatt egy viszonylag jó megoldás vonzóbb a költség szerinti optimális megoldásnál. Ha több megoldást keresünk és azok léteznek, akkor azokat a Solutions fülre kattintva tudjuk egymással összehasonlítani.




*5.19. ábra. Megoldások összehasonlítása*

Számoláskor ügyeljünk arra, milyen mértékegységeket adunk meg, ez lényeges szempont a megoldás kiértékelés során. Az alapértelmezett mértékegységeket az Options menüben tudjuk beállítani. Az options menüben lehetőség van egyéb alapértelmezett értékek beállítására is, a deafult values menüpont alatt.




*5.20. ábra. Alapértelmezett értékek megadása*

## 5.4. Feladatok

**5.1. Feladat** Rajzolja fel az alábbi folyamatszintézis feladathoz tartozó P-gráfot!

$$M := \{A, B, C, D, E, F, G, H, I\} \tag{5.44}$$

$$O := \{(\{A, B\}, \{C\}), (\{C\}, \{D, E\}), (\{E, F\}, \{G\}), (\{G\}, \{H, I\}), (\{I\}, \{F, B\})\} \tag{5.45}$$

Mely anyagok lehetnek potenciálisan termékek illetve nyersanyagok?

**5.2. Feladat** Rajzolja fel az alábbi folyamatszintézis feladathoz tartozó P-gráfot!

$$M := \{A, B, C, D, E, F, G, H, I\} \tag{5.46}$$

$$O := \{(\{A\}, \{B\}) , (\{C, D\}, \{E\}) , (\{D, F, B\}, \{E, G\}) , \\ (\{E\}, \{H, B\}) , (\{B, I\}, \{E\})\} \tag{5.47}$$

PNS Studio használatával határozza meg a maximális struktúrát és a kombinatorikusan lehetséges megoldásstruktúrákat.

**5.3. Feladat** Az axiómák segítségével határozza meg az adott folyamatszintézis feladathoz tartozó maximális struktúrát!

$$M := \{A, B, C, D, E, F, G, H, I, J, K\} \tag{5.48}$$

$$R := \{A, J\} \tag{5.49}$$

$$P := \{G\} \tag{5.50}$$

$$O := \{(\{A\}, \{B, C\}) , (\{D, C\}, \{E\}) , (\{B\}, \{F\}) , (\{E\}, \{G, D\}) , \\ (\{F\}, \{H, I\}) , (\{J\}, \{K\})\} \tag{5.51}$$

# 6. fejezet

# Ütemezés

Ütemezési feladatok széles körben fordulnak elő az informatikai rendszerekben, a termelő iparban (pl. a vegyiparban, az olajiparban, a gépiparban), a mezőgazdaságban és az építőiparban. Ezért gyakorlati szempontból fontosak az optimális vagy közel optimális megoldások meghatározására alkalmas módszerek.

A termelő rendszerek mellett a szolgáltató és az informatikai rendszerekhez is kapcsolódnak bonyolult ütemezési feladatok. Ha a rendszer tartalmaz olyan konkurens folyamatokat, melyek ugyanazokat az erőforrásokat igénylik és nem áll rendelkezésre megfelelő számú erőforrás, akkor az erőforrások ütemezésével lehet a folyamatokat kiszolgálni.

A fejezet olyan gráf leíráson (S-gráf, ahol az S az ütemezés angol nevének a schedulenak az első karakterét jelöli) alapul, amely alkalmas a gyártó rendszerekben előforduló ütemezési feladatok speciális tulajdonságait megfelelően kifejező leírására. A módszer egy, az S-gráfhoz illesztett szétválasztás és korlátozás algoritmust használ, amely képes a feladatok hatékony megoldására. A következő részben vizsgáljuk az S-gráf használata során felmerülő kérdéseket. Bemutatjuk a feladat korrekt felírásához szükséges speciális S-gráf, az úgynevezett recept-gráf felépítésének menetét. Matematikai módszerekkel megadjuk, hogy egy S-gráf hogyan és mikor reprezentál egy megoldást valamint hogyan definiálja az ehhez szükséges fogalmakat.

## 6.1. Bevezetés

Mivel egy ütemezési feladat megoldásai a teljes rendszer működési idejére vonatkozóan nagymértékben eltérhetnek egymástól, ezért fontos a megoldások közül a számunkra legjobbat megtalálni, valamint az is elengedhetetlen, hogy az ütemezési feladatokat a felhasználó számára elfogadható időn belül meg lehessen oldani.

Egy ütemezési feladat többféleképpen definiálható. A fejezet első felében ütemezési feladatnak nevezzük azt a feladatot, ahol a lehető legrövidebb idő alatt kell adott mennyiségű termékeket előállítani a rendelkezésre álló szakaszos működésű berendezések felhasználásával, ahol egy terméket taszkok adott sorrendű végrehajtásával lehet előállítani. A gyakorlatban egy taszk elvégzésére több berendezés is alkalmas lehet, a közülük való választás is az ütemezési feladathoz tartozik. A fejezet második részében egy másik típusú ütemezési feladattal is megismerkedünk, ahol adott időn belülll kell majd a lehető legnagyobb profitot elérni.

Egy szakaszos működésű rendszerben a taszkokra a következő tulajdonságok teljesülnek:

- Egy taszk adott bemenetből adott kimenetet állít elő.
- A teljes bemenetnek a taszk indulásakor rendelkezésre kell állnia.
- A kimenet csak a taszk befejezése után áll rendelkezésre.
- Egy taszk adott ideig tart, ahol az idő függhet a használt berendezéstől.
- A taszk nem megszakítható.

A szakirodalomban eddig ismert megoldási módszerek egy része gyakorlati tapasztalatoknak megfelelően megfogalmazott heurisztikus szabályokon alapul, melyek fő hátránya, hogy nem tudják biztosítani az optimumot. Egy másik napjainkban használatos módszer, hogy minden ütemezési feladatosztályhoz felírnak egy speciális matematikai modellt, és azt általános célú megoldó szoftverekkel oldják meg. Ez a megközelítés a feladat nagyfokú kombinatorikus bonyolultság miatt erősen korlátozza a megoldható feladatok méretét. A gyakorlatban egy ütemezési feladat matematikai modelljének megfelelő leírásához rengeteg változót kell bevezetni (például a berendezések taszkokhoz való rendeléséhez, a termelés és a taszkok sorrendjének leírásához, a taszkok időzítéséhez), amely modell nehezen vagy gyakran egyáltalán nem oldható meg általános célú megoldó szoftverekkel.

Az előbbiekből látszik, hogy minden ütemezési feladatosztályhoz egy specializált algoritmusra és megoldó szoftverre van szükség, amelyhez jó kiindulási pontot nyújt az S-gráfhoz ([18], [12]) illesztett korlátozás és szétválasztás módszeren alapuló algoritmus ([19]). Ehhez az alapalgoritmushoz készíthetőek feladatspecifikus valamint általánosan használható eljárások illetve gyorsítások, amelyeket könnyen illeszthetünk a meglévő algoritmushoz, és amelyek adott feladatosztály speciális tulajdonságait figyelembe veszik és kihasználják. A feladat előzetes vizsgálata során eldönthető, hogy ezek közül az eljárások közül melyeket kell, illetve melyeket érdemes használni a gyorsabb megoldás érdekében. Ezenkívül az algoritmus rugalmassága lehetővé teszi, hogy a felhasználó által kért követelményeket figyelembe lehessen venni, például az első három legjobb megoldás megadása vagy speciális heurisztikák beépítése.

A szakirodalomban eddig közölt vizsgálatokban nem fordítottak kellő hangsúlyt arra, hogy az ütemezési feladatoknak létezik-e általános, jól használható leírása, amely segítségével az ütemezés szinte minden feladatosztályát meg lehet oldani. Szakaszos működésű rendszerek ütemezéséhez a jól használható, hatékony gráf leírást, az S-gráfot, és a hozzá elkészített algoritmust használtunk. Tekintettel arra, hogy ezt a módszertant későbbi felhasználásra szánjuk, a további kiterjesztések és gyorsítások hatékony fejlesztése érdekében szigorú formalizmust vezettünk be.

Amennyiben az Olvasó komolyabban érdeklődik az S-gráf módszertan iránt, kérjük látogassa meg a www.s-graph.com honlapot.

## 6.2. Flow shop, job shop, open shop

Ütemezési feladatokat széles körű előfordulásuk és összetett jellegük miatt sokféle szempont alapján osztályozhatjuk. Az ütemezési feladatokat osztályozhatjuk a termékek elkészítéséhez

szükséges műveletek, feladatok végrehajtásának módja alapján. Így megkülönböztethetünk „flow shop”, „job shop” és „open shop” ütemezési feladatokat.

A taszkokat ebben a fejezetben műveleteknek (operation) hívjuk, ezek csoportját pedig munkának (job), amely egy termék előállításához szükséges műveleteket tartalmazza a gyártás sorrendjébne. Feltételezzük, hogy a különböző munkákhoz tartozó műveletek között nincs sorrendi feltétel, kivéve azokat a műveletek, amelyeket ugyanazon gépekkel (machine) lehet elvégezni. A fejezetben a következő feltételezésekkel élünk:

1. A műveletek nem megszakíthatóak, és minden művelethez pontosan egy gép tartozik, amivel végre lehet hajtani.

2. Nem lehet egy géppel egyszerre több műveletet végezni.

Legyen $J=\{J_1, \ldots, J_n\}$ a munkák halmaza és $M=\{M_1, \ldots, M_m\}$ a gépek halmaza. Minden $J_j$ munka $c_j$ darab műveletből áll $(O_{1j}, O_{2j}, \ldots, O_{cj},)$ ahol $c_j$ munkánként más és más lehet. Lehetnek olyan műveletei egy munkának, amelyeket ugyanaz a gép hajt végre így $c_j$ lehet nagyobb, mint $m$. Az $O_{ij}$ művelet működési idejét $p_{ij}$-vel, a $J_j$ munka működési idejének vektorát $p_j$-vel jelöljük. A gépek hozzárendelését műveletekhez $m_{ij}$ $(i=1, 2, \ldots, c_j, j=1, 2, \ldots, n)$ számokkal jelöljük, ahol $O_{ij}$ műveletet az $m_{ij}$ gépen kell elvégezni. A cél a műveletek optimális sorrendjének meghatározása úgy, hogy a rendszer teljes működési ideje minimális legyen.

Ezekkel a jelölésekkel tudjuk definiálni a flow shop, job shop és open shop feladatokat a következő módon.

- Egy **flow shop** feladatban minden munka minden gépen végigmegy pontosan ugyanolyan sorrendben. Az általánosság elvesztése nélkül feltételezhetjük, hogy a továbbiakban a sorrend $M_1, M_2, \ldots, M_m$.
  Tipikus példa flow shop feladatra a futószalag, ahol a munkások és a munkaállomások jelentik a gépeket. Egy flow shop feladat jellemezhető a következőképen $c_j=m$ és $m_{ij}=i$ $(i = 1, 2, \ldots, m, j = 1, 2, \ldots, n)$.

- Egy **job shop** feladatban nincsenek megkötések a műveletek számára és a hozzájuk rendelt gépekre. A gépek sorrendje minden munkára különböző lehet, valamint a felhasznált gépek is munkánként különbözhetnek.

- Az **open shop** ütemezési feladat hasonló a job shop feladathoz, eltérés abban van, hogy míg job shop feladatoknál a munkák műveleteinek a sorrendje rögzített, addig open shop feladatokban a munkák műveleteit bármilyen sorrendben végre lehet hajtani a megfelelő gépeken. Azaz a munkákhoz tartozó műveletek sorrendje nincs megkötve, de mindet el kell végezni.

Egy flow shop, job shop vagy open shop ütemezési feladat megoldása ábrázolható az úgynevezett Gantt diagrammal (Gantt chart), ahol a gépek a függőleges tengelyen vannak ábrázolva, az idő pedig a vízszintes tengelyen helyezkedik el.

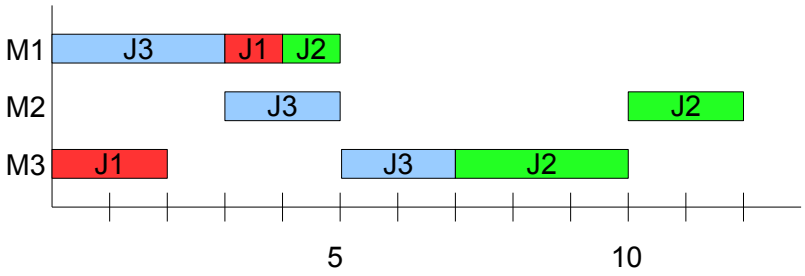
**6.1. példa.** Egy job shop feladatban három munka van, amelyek sorrendben kettő, három és három műveletből állnak. Az első munka két műveletből áll, ahol az első munkát a 2-es géppel lehet elvégezni 2 időegység alatt, a másodikat pedig az 1-es géppel 1 időegység

alatt. A második munka három műveletből áll, ahol a gépek sorrendje 1, 3, 2, a hozzájuk tartozó működési idő pedig sorrendben 1, 3, 2 időegység. A harmadik munka szintén három műveletből áll, itt a gépek sorrendje 1, 2 és 3, a működési idők pedig sorrendben 3, 2, 2. A korábbiakban bevezetett jelölésekkel ez az alábbi formában írható le:

$$J_1 : (m_{11}, m_{21}) = (2, 1) \qquad p_1 = (2, 1)$$
$$J_2 : (m_{12}, m_{22}, m_{32}) = (1, 3, 2) \qquad p_2 = (1, 3, 2)$$
$$J_3 : (m_{13}, m_{23}, m_{33}) = (1, 2, 3) \qquad p_3 = (3, 2, 2)$$

A példa egy megoldásának Gantt diagrammja a 6.1 ábrán látható, ahol például az 1-es gép ($M_1$) 0 időpillanatban elkezd dolgozni a $J_3$ munkán, ami tart 3 időegységig, majd 1 időegységet tölt a $J_1$ munkával, végül $J_2$ munkának a műveletén dolgozik 4 időpillanattól kezdve az 5 időpillanatig.




*6.1. ábra. A 6.1. példa egy megoldása Gantt diagrammal*

Gannt diagrammok rajzolásnál figyelni kell arra, hogy egy géphez tartozó műveletek ne fedjék egymást, ahogy ezt a fejezet elején tett második feltételezésből következik (nem lehet egy géppel egyszerre több műveletet végezni). Fontos továbbá, hogy az egy munkához tartozó műveleteket nem lehet egyszerre elvégezni, valamint flow shop és job shop esetén a munkához tartozó műveletek sorrendje a feladatban rögzített.

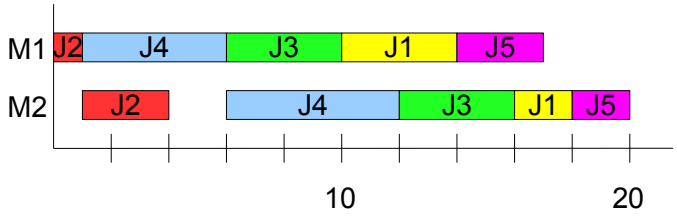
Flow shop feladatoknál egyszerű megmutatni, hogy ha minden működési idő pozitív, akkor van olyan optimális megoldás, ahol a munkák működési sorrendje megegyezik az első két műveleten, valamint megegyezik a működési sorrend az utolsó két műveleten is. Ebből következik, hogy háromgépes flow shop feladatoknál ugyanaz a gépek használati sorrendje a munkákhoz, mivel az elsőn ugyanaz a sorrend mint a másodikon valamint a másodikon ugyanaz a sorrend, mint a harmadikon. Ha legalább négy gép van, akkor lehet olyan optimális megoldás, hogy a gépek működési sorrendje különbözik a munkákon.

2-gépes flow shop feladatok hatékonyan megoldhatóak Johnson algoritmusát használva:

**begin**
$N_1 \leftarrow \{j : p_{1j} < p_{2j}\}$
$N_2 \leftarrow \{j : p_{1j} \geq p_{2j}\}$
rendezzük $N_1$-et nem csökkenő sorrendbe $p_{1j}$ szerint
rendezzük $N_2$-t nem növekvő sorrendbe $p_{2j}$ szerint
optimális megoldásban a rendezett $N_1$ halmaz elemeit a rendezett $N_2$ követi
**end**

**6.2. példa.** Adott egy öt munkát tartalmazó két gépes flow shop feladat a munkák működési idők vektorával: $p_1$=(4, 2), $p_2$=(1, 3), $p_3$=(4, 4), $p_4$=(5, 6), $p_5$=(3, 2). Ezek alapján felírhatjuk

a halmazokat: $N_1 = \{2, 4\}$, $N_2 = \{1, 3, 5\}$. Majd ezeket rendezzük: $N_1 = \{2, 4\}$, $N_2 = \{3, 1, 5\}$ és a rendzett halmazokat összefűzve végül megkapjuk az optimális megoldást: 2, 4, 3, 1, 5, melynek Gannt diagramja a 6.2 ábrán látható.




*6.2. ábra. A 6.2. példa optimális megoldása*

Több speciális flow shop feladat (amelyben több, mint két gép van) hatékonyan megoldható specializált algoritmusokkal. Egy nagy része a feladatoknak kezelhető a Johnson algoritmussal, vagy annak kis változtatásával. Például Johnson bebizonyította, hogy három gépes esetben ha a második (középső) gép legnagyobb működési ideje jóval kisebb, mint az első vagy az utolsó gép legkisebb ideje (az uralkodik rajta), akkor egy optimális megoldást kaphatunk a Johnson algoritmus segítségével, ha a második gépet „összevonjuk” azzal, amelyik uralkodik rajta.

## 6.3. Ütemezési feladat általános megfogalmazása

A továbbiakban a követező alapfogalmakat használjuk:

- **Termék**nek (product) nevezzük a gyártó rendszerben elő állítani kívánt anyagot.
- A **berendezés** (equipment unit) a termékek előállításához felhasználható eszköz, erőforrás. A berendezés megújuló erőforrás, azaz használat után újra rendelkezésre áll.
- A **taszk** (task) egy olyan tevékenység, amely nem bontható résztevékenységekre és adott idő alatt adott bemenetekből adott kimeneteket generál. Egy taszk végrehajtásához egy vagy több berendezés áll rendelkezésre, ahol a végrehajtási idő függ a felhasznált berendezéstől. Egy berendezés egy időben csak egy taszkhoz rendelhető, azaz nem lehet egy berendezéssel egyszerre több taszkot végrehajtani. Továbbá egy taszkhoz csak egy berendezés rendelhető, azaz a taszkok nem megoszthatóak.
- A **recept** (recipe) tartalmazza azokat a minimális információkat, melyek a termékek előállításához szükségesek. Ezek a termékek előállításához szükséges taszkok hálózata, a taszkokhoz rendelhető berendezések (a működési idővel együtt) és az előállítandó termékmennyiségek. A receptben, az eddigiektől eltérően lehetnek több bemenettel illetve kimenettel rendelkező taszkok is, azaz a termékek előállítása nem feltétlenül egy vonal mentén történik.
- Szakaszos termelő rendszerekben a termelés **adag**okban (batch) történik. Ez azt jelenti, hogy ha az előállítandó termék mennyisége több, mint amennyi a recept egyszeri végrehajtása során keletkezik, akkor a recept többszöri megismétlésével állítják elő a kívánt

termékmennyiséget. A recept egyszeri végrehajtását, ütemezését jelenti egy adagnyi termék előállítása.

**6.3. példa.** Tekintsük a következő ütemezési feladatot, ahol három terméket kell előállítani három lépésben. A termékek előállítási módjai (receptjei) és a felhasználható berendezések a 6.1 táblázatban láthatóak (például az A termék első taszkjához az E1 berendezés használható, melynek működési ideje 6 perc). A szükséges termékmennyiségek adagok számával adottak a 6.2 táblázatban.

*6.1. táblázat. Recept a 6.3. példához*

| | A termék | | B termék | | C termék | |
|---|---|---|---|---|---|---|
| Taszk | Berendezés | Idő [perc] | Berendezés | Idő [perc] | Berendezés | Idő [perc] |
| 1 | E1 | 6 | E3 | 9 | E2 | 7 |
| 2 | E2 | 9 | E3 | 15 | E1 | 17 |
| 3 | E1 | 14 | E2 | 16 | E3 | 8 |

*6.2. táblázat. Adagok száma a 6.3. példához.*

| Termék | A | B | C |
|---|---|---|---|
| Batch-ek száma | 1 | 1 | 1 |

A feladat egyértelmű leírásához meg kell még határozni, hogy van-e lehetőség a keletkező köztes termékek tárolására a taszkok között. Ilyen lehetőség lehet egy tároló berendezés vagy maga az anyagot előállító berendezés. Ha van ilyen lehetőség, akkor fontos, hogy milyen mennyiségben lehet tárolni az anyagot és a tárolókat hol (mely taszkok vagy berendezések között) lehet használni; ha nincs, akkor a gyártó berendezésben várakozhat-e az anyag. Ezen tulajdonságok alapján a következő tárolási stratégiákat különböztethetjük meg:

- Nincs lehetőség köztes tárolásra:
  - **NIS** tárolási stratégia (non intermediate storage policy): A köztes termékek tárolásához nem áll rendelkezésre tároló berendezés. A taszk elvégzése után a köztes terméket az azt előállító berendezésben tárolhatjuk, ilyenkor a berendezést csak akkor lehet használni egy másik taszk végrehajtásához, ha a keletkezett anyagot már áttöltöttük egy másik, értelemszerűen a következő taszkot végrehajtó berendezésbe.
    Gyakorlatban az egyik leggyakoribb eset.
  - **ZW** tárolási stratégia (zero wait policy): A köztes termék tárolásához nem áll rendelkezésre tároló berendezés valamint az őt előállító berendezésben sem tárolhatjuk. Ilyenkor az anyagot azonnal át kell tölteni a következő berendezésbe.

Ez gyakran előfordulhat olyan esetekben, amikor nem stabil anyagot gyártunk, vagy az anyag tulajdonságai megkövetelik, például acélgyártás esetében nem tárolhatjuk a forró fémet, mivel az megszilárdul és a továbbiakben nem lehet felhasználni.

- Van lehetőség köztes tárolásra:
  - **UIS** tárolási stratégia (unlimited intermediate storage policy): A köztes termékeket „végtelen” mennyiségben lehet tárolni. A taszk elvégzése után a berendezésből a köztes terméket tároljuk egy, gyakorlati szempontból „végtelen” kapacitású tárolóban, azaz a berendezést tisztítás után rögtön tudjuk használni.
    Ilyen tároló lehet egy megfelelően nagy raktárhelyiség, ahol a keletkezett félkész termékek elhelyezhetők.
  - **FIS** tárolási stratégia (finite intermediate storage policy): A rendszer véges számú és méretű tárolót tartalmaz. Minden tároló csak két előre meghatározott berendezés között használható.
    Ilyen lehet egy folyékony anyagokat gyártó rendszer, ahol a berendezések a méretük miatt nem mozdíthatóak és csak a kiépített csöveken keresztül lehet anyagot szállítani.
    Szintén FIS-nek nevezik azt az esetet, amikor a tárolók nem berendezésekhez rendeltek, hanem anyagokhoz. Azaz egy tároló berendezés csak bizonos típusú anyagot tárolhat.
  - **CIS** tárolási stratégia (common intermediate storage policy): Ebben az esetben is véges számú és méretű tároló áll rendelkeszésre. Ez a tárolási stratégia abban különbözik a FIS stratégiától, hogy itt a tárolók helye nem rögzített, valamint arra is figyelni kell, hogy ha több anyagot tárolunk benne, akkor a tárolások között ki kell tisztínai a berendezést.
    A FIS esetnél említett csövek hálózatot alkotnak, ahol nagyobb az összeköttetési lehetőség.
  - **MIS** tárolási stratégia (mixed intermediate storage policy): Az előző négy tárolási stratégia keveréke, a rendszer különböző pontjain különböző stratégiák lehetnek.

A szakirodalomban túlnyomórészt az UIS stratégiával foglalkoznak, amely stratégia főleg a gépiparban használatos, ahol például egy raktárhelyiséggel megoldható nagy mennyiségű köztes termék tárolása is. Gyakorlatban viszont figyelembe kell vennünk azokat az eseteket is, amikor nincs lehetőség köztes tárolásra (NIS stratégia).

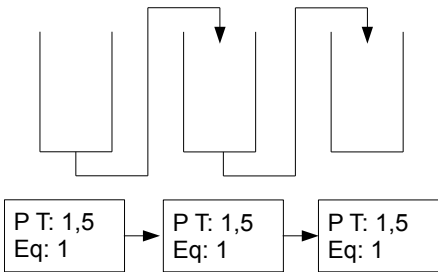
Egy egyszerű recept leírásához, amikor a termékek előállítása lineáris, azaz a taszkok sorrendje minden termékre egyértelműen rögzített, elegendőek a fentiekben bemutatott információk. Összetett recept esetén, amikor a receptben van több kimenettel, illetve több bemenettel rendelkező taszk, szükségünk lehet további adatokra a feladat korrekt leírásához. Amikor egy taszknak több bemenete is van, akkor a bemenetek időzítése is fontos. Lehet, hogy a bemeneteknek egyszerre jelen kell lenniük a taszk kezdeténél, a bemenetek sorrendje rögzített, a bemenetek tetszőleges sorrendben rendelkezésre állhatnak, illetve lehetséges ezen esetek kombinációja is. Ez az időzítés a megvalósíthatóságra is hatással lehet, például az első

esetben (egyidejű bemenetek esetén) NIS stratégiát használva egy taszk két bemenetét nem állíthatjuk elő ugyanazzal a berendezéssel, a többi esetben viszont igen.

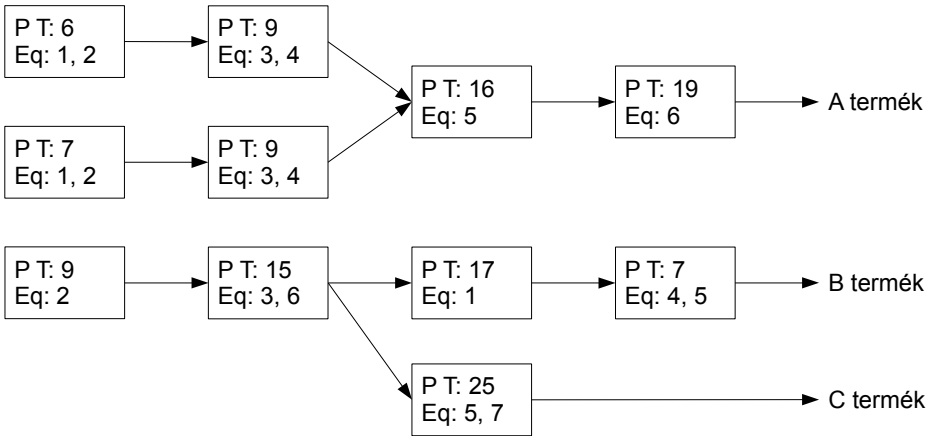
A következő fejezetben bemutatjuk az S-gráf gráf leírást, amely a feladat összes eddig bemutatott strukturális tulajdonságát jól ábrázolja. A reprezentáció NIS tárolási stratégiához készült, de bármely fent említett tárolási stratégiához alkalmazható.

## 6.4. S-gráf leírás

A termékek előállítását leíró receptet hagyományosan lehet ábrázolni egy irányított gráffal, ahol a gráf csomópontjai jelentik a taszkokat, a közöttük lévő élek pedig ezek sorrendjét. A működési idők és a taszkokhoz rendelkezésre álló berendezések a megfelelő csomópontokban adottak. A 6.3 ábra mutatja egy három lépésben (három taszkkal) előállítható termék hagyományosan megadott előállítási módját és receptjének reprezentációját. Természetesen összetett receptek is leírhatóak ezen a módon, például a 6.4 ábrán látható receptben az A terméket két köztes anyag összekeverésével, majd a keverék további feldolgozásával lehet előállítani.



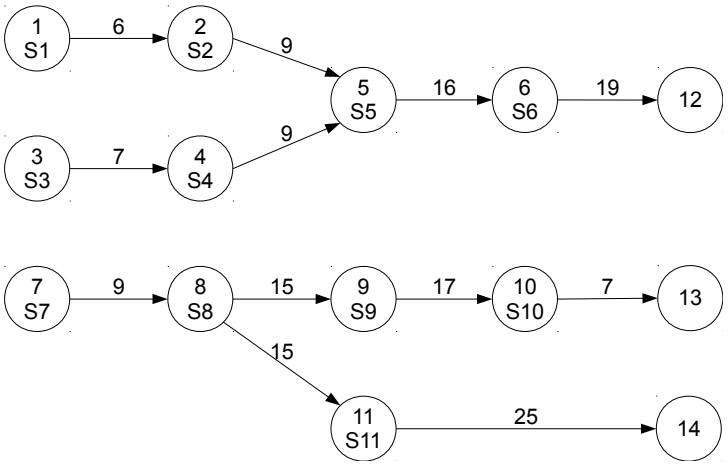

*6.3. ábra. Három lépésben előállítható termék előállítási módja és receptjének hagyományos leírása.*




*6.4. ábra. Három termék előállításának hagyományos leírása.*

Ebben a hagyományos leírásban az élek a taszkok sorrendjét adják meg, minden egyéb információ a csomópontokhoz van rendelve. A 6.4 ábrán látható receptnek a gráf leírása a 6.5 ábrán látható, ahol Si azoknak a berendezéseknek a halmaza, amelyekkel az i csomóponttal

reprezentált taszkot végre lehet hajtani (például S1 = {E1, E2}), valamint egy-egy további csomópont jelöl minden terméket. Ebben a leírásban az élek súlya alsó korlátot jelent a két kapcsolódó taszk kezdési idejének különbségére. Egy taszk működési ideje a hozzá rendelhető berendezésektől függően különböző lehet, ebben az esetben az él súlya a felhasználható berendezések működési idői közül a legkisebb lesz. Ez az érték a feladat megoldása során természetesen változhat a berendezések taszkhoz való hozzárendelése során.




*6.5. ábra. Gráf leírás a 6.4 ábrán látható recepthez.*

Ha egy szakaszos működésű gyártási folyamat struktúrája kört tartalmaz, az nem eredményez kört a receptet leíró gráfban, mivel a látszólagos recirkuláció egy időben későbbi adag (batch) valamely taszkjának a betáplálását jelenti. Ezért feltételezhetjük, hogy bármely recept egy körmentes irányított gráffal ábrázolható.

Tegyük fel, hogy minden termékre adott a legyártandó mennyiség, valamint adott, hogy melyik taszk melyik berendezéssel vagy berendezésekkel hajtható végre. A taszkok sorrendje ábrázolható egy irányított gráffal, ahol a taszkok sorrendje élekkel adott úgy, hogy az élek kötik össze az egy berendezéshez tartozó taszkokat jelölő csomópontokat. Az élek a taszkok működési idejével súlyozottak és megadják ezek működési sorrendjét. Az élek súlya azért van súlyozva a működési idővel, mert ezekre az élekre is igaz, hogy alsó korlátot jelentenek a kapcsolódó taszkok kezdési idejének különbségére és a berendezésnek előbb végre kell hatjania az aktuális taszkot és csak utána kezdheti el a következőt.

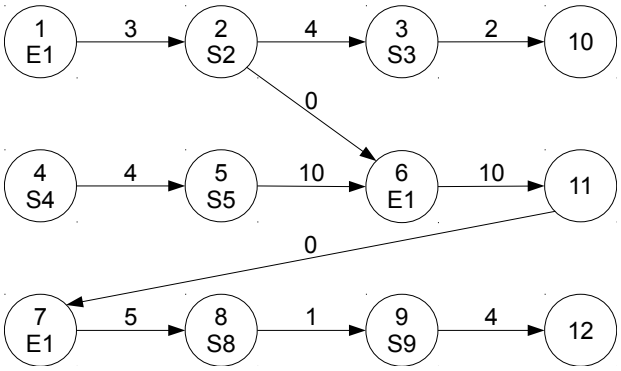
Ez a fajta leírás jól használható UIS típusú feladatok megoldására, de nem megfelelő NIS tárolási stratégia esetén. Az UIS stratégia feltételezi, hogy a berendezések a taszkok végeztével azonnal rendelkezésre állnak, azaz a köztes termékek, amelyek a taszk végrehajtása során keletkeznek, kikerülnek a berendezésből és eltárolásra kerülnek, amíg a következő taszk el nem kezdődik. A legtöbb szakaszos működésű folyamatban ezt nem lehet biztosítani, azaz NIS esetet kell figyelembe vennünk.

NIS stratégiánál a berendezések nem állnak rendelkezésre egy taszk befejezése után egészen addig, amíg a tárolt anyag át nem töltődik a következő taszkot végrehajtó berendezésbe. A gráfban ezt a további feltételt is reprezentálni kell illetve lehet egy vagy több él segítségével a következő módon. Jelölje $\tau$ azokhoz a taszkokhoz tartozó csomópontokat, amelyek a recept szerint a j csomóponthoz tartozó taszkot közvetlenül követik. Ha az Ei berendezés a k taszkot

végzi el közvetlenül a j után, akkor egy nulla súlyú (vagy váltási idővel súlyozott) élet húzunk $\tau$ minden eleméből k-ba. Ezt a fajta leírást nevezzük S-gráfnak.

A 6.6 ábrán látható az E1 berendezés 1-6-7 működési sorrendjének megadása S-gráf segítségével. Az ábráról leolvasható, hogy az E1 berendezés először végrehajtja az 1 csomópon által reprezentált taszkot, majd megvárja a 2-es taszkot végrehajtó valamelyik berendezést (S2 halmaz) és áttölti bele az anyagot, utána végrehajtja a 6-os taszkot, amivel előállít egy terméket, végezetül a 7-es taszkot hajtja végre és áttölti az nyagot az S8 halmazból a 8-as taszkohoz rendelt berendezésbe. Mivel a példában a váltási élek súlya mindenhol nulla, ez azt jelenti, hogy nincs szükség az E1 berendezés tisztítására a következő taszk előtt.




*6.6. ábra. Az E1 berendezés 1-6-7 működési sorrendjének megadása NIS esetben*

Matematikai jelölésekre áttérve, egy irányított $G$ gráfot egy $(N, A)$ párral adhatunk meg, ahol $N$ véges halmaz, a csomópontok halmaza és $A$ az élek halmaza ($A \subseteq N \times N$). Egy S-gráfnak két típusú éle van (léteznek recept-élek és ütemezési-élek). Így egy S-gráfot $G(N, A_1, A_2)$ formában adhatunk meg, ahol $N$ a csomópontok, $A_1$ a recept-élek, $A_2$ pedig az ütemezési-élek halmaza feltéve, hogy $A_1 \subseteq N \times N$, $A_2 \subseteq N \times N$ és $A_1 \cap A_2 = \emptyset$; továbbá minden $(i, j) \in A_1 \cup A_2$ élhez tartozik egy nem negatív érték $c(i, j)$, az él súlya. Ha egy él $i$-ből $j$-be mutat ($(i, j) \in A_1 \cup \cup A_2$), akkor az a gyakorlatban azt jelenti, hogy az $j$ csomóponthoz tartozó taszk legalább $c(i, j)$ idővel később kezdi a működését, mint az $i$ csomóponthoz tartozó taszk.

Speciális S-gráfot vezetünk be a recepthez (recept-gráf) és a megoldáshoz (ütemezési-gráf).

### 6.4.1. Recept-gráf

Egy ütemezési feladat receptje megadja minden termékhez a termékekhez tartozó taszkok sorrendjét, a köztük lévő anyagáramokat, valamint az egyes taszkokhoz felhasználható berendezések halmazát a hozzájuk tartozó működési idővel. Ezeknek az információknak a recept-gráfban szerepelniük kell ahhoz, hogy a receptet megfelelően le tudjuk vele írni.

A termékek gyártását leíró recept alapján a recept-gráf alapját jelentő taszk-hálózat felépítéséhez a következő lépéseket hajtjuk végre:

1. Rendeljünk egy-egy csomópontot minden taszkhoz (taszk-csomópont) és egy-egy csomópontot minden termékhez (termék-csomópont).

2. Egy él (recept-él) mutasson minden taszkot reprezentáló taszk-csomópontból a receptben utána következő taszkot reprezentáló taszk-csomópontba, valamint a terméket gyártó taszkokhoz tartozó csomópontokból a megfelelő termék-csomópontba.

3. Egy recept-él súlya legyen egyenlő az él kezdő csomópontjához tartozó taszk működési idejével abban az esetben, ha a taszkhoz csak egy berendezés áll rendelkezésre. Ha több berendezés is rendelkezésre áll, akkor a működési idők közül a legkisebbel egyezzen meg az él súlya.

4. Ha egy termékből nagyobb mennyiségre van szükség, mint amennyit a recept alapján egyszerre elő lehet állítani (több adagra van szükség), akkor a termék előállításában résztvevő taszkok taszk-csomópontjai, a termék-csomópont, valamint a közöttük lévő élek annyiszor kerülnek be a gráfba, ahány adag már minimálisan elegendő a szükséges anyagmennyiséghez.

Az így keletkezett gráfot hívjuk taszk-hálózatnak, ahol $N_t$ jelöli a taszk-csomópontok, $N_p$ pedig a termék-csomópontok halmazát ($N_t \cap N_p = \emptyset$).

Fontos megjegyezni, hogy több adag esetén minden taszkhoz több taszk-csomópont is tartozik. Ha a feladat szempontjából fontos, hogy egy taszkhoz több csomópont is tartozik, akkor ezen csomópontok által reprezentált termelési folyamatot (és továbbiakban magát a csomópontot is) tevékenységnek nevezzük, egyébként a taszk kifejezést használjuk.

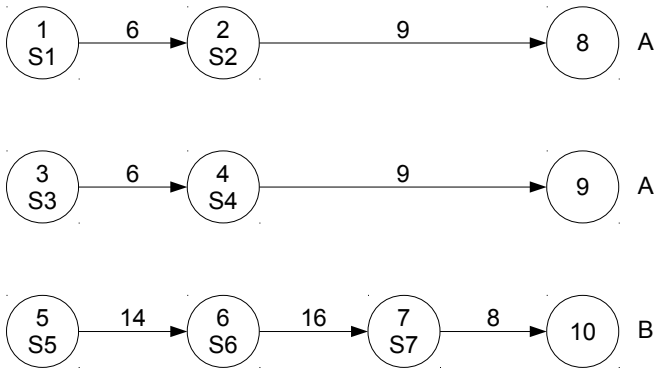
A továbbiakban nem vesszük figyelembe a több bemenettel rendelkező taszkok bemeneteinek időzítését. Ebbn az esetben a taszk-hálózat alkotja a recept-gráfot. A recept-gráfban minden a receptben található strukturális információ adott. Ha $G(N, A_1, A_2)$ egy recept-gráf, akkor körmentes és $A_2 = \emptyset$. Legyen $N_i (\subset N_t)$ azon csomópontok halmaza, amelyek az $i$ berendezéssel végrehajthatók. Feltehetjük, hogy minden taszkhoz létezik legalább egy berendezés, amivel végre lehet hajtani, azaz a taszk-csomópontok halmaza megadható a következő módon $N_t = N_1 \cup N_2 \cup \cdots \cup N_n$, ahol $N_i$ és $N_j$ $(i, j = 1, 2, \ldots, n)$ tartalmazhat azonos elemeket, azaz metszetük nem szükségszerűen üres.

**6.4. példa.** Tegyük fel, hogy két terméket kell előállítanunk, A-t és B-t, A-ból két adagot, B-ből egyet. A-t két egymás utáni lépésben lehet előállítani, ahol az első lépés az S1, a második pedig az S2 halmazban lévő berendezések bármelyikével végrehajtható 6 illetve 9 időegység alatt. B-t három egymást követő lépésben lehet előállítani az S3, az S4 illetve az S5 halmazokban lévő berendezésekkel 14, 16 illetve 8 időegység alatt. A feladat recept-gráfja a 6.7 ábrán látható.

## 6.4.2. Ütemezési-gráf

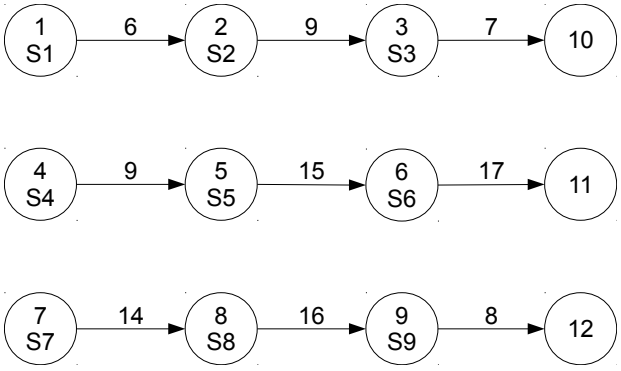
Az ütemezési-gráf olyan speciális S-gráf, amely egy megoldást reprezentál; az ütemezési feladat minden megoldásához létezik egy ütemezési-gráf és ez a gráf minden megoldáshoz különböző. A $G'(N, A_1, A_2)$ S-gráfot a $G(N, A_1, \emptyset)$ recept-gráfhoz tartozó ütemezési-gráfnak nevezünk, ha a recept-gráf minden csomópontja (taszkja) ütemezve van, figyelembe véve a berendezés-taszk hozzárendelést. Egy megfelelő keresési módszerrel az optimális megoldáshoz tartozó ütemezési-gráf és berendezés-taszk hozzárendelés megtalálható. Megfelelő keresési módszerrel nem csak az optimális, hanem az összes megoldás generálható.




*6.7. ábra. A 6.4. példa recept-gráfja: két adag az A termékből és egy a B-ből.*

**6.5. példa.** Három termék előállítása a 6.8 ábrán látható recept-gráf szerint történik. Minden taszkhoz egy berendezés áll rendelkezésre, ahol S1 = {E1}, S2 = {E3}, S3 = {E2}, S4 = {E2}, S5 = {E3}, S6 = {E1}, S7 = {E1}, S8 = {E2}, S9 = {E3}. A feladatnak nyolc különböző megoldása létezik, amelyek ütemezési-gráfjai a 6.9 ábrán láthatóak.

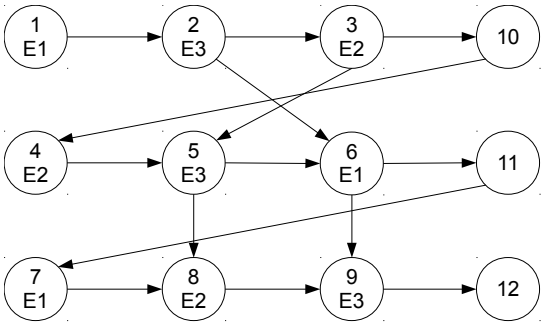



*6.8. ábra. Recept-gráf a 6.5. példához.*

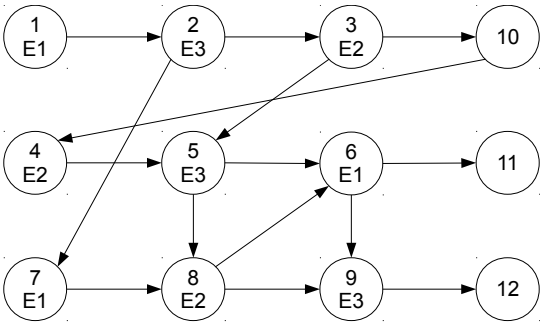
Az ütemezési-gráf formális leíráshoz tételezzük fel, hogy a $G(N, A_1, \emptyset)$ recept-gráfhoz adott egy $G'(N, A_1, A_2)$ ütemezési-gráf, ahol $A_2 \subseteq N \times N$. Jelölje $M_i$ $(i = 1, 2, \ldots, n)$ azoknak a csomópontoknak a halmazát, amelyekhez tartozó taszkokat az $i$ berendezés hajtja végre, azaz azokat a csomópontokat, amelyekhez az $i$ berendezést hozzárendeltük. Feltételezzük, hogy a megoldásban egy taszkot pontosan egy berendezéssel lehet végrehajtani, azaz $M_i \cap M_j = \emptyset$ $(i\,j, i, j = 1, 2, \ldots, n)$. Ebből következik, hogy $M_1, M_2, \ldots, M_n$ egy partícionálása a taszk csomópontok halmazának $(N_t = N_1 \cup N_2 \cup \cdots \cup N_n)$ úgy, hogy $M_i \subseteq N_i$ $(i = 1, 2, \ldots, n)$.

Ahhoz, hogy az ütemezési-gráf korrekt matematikai leírását meg tudjunk adni, szükségünk van egy speciális S-gráf bevezetésére, amelyet komponens-gráfnak nevezünk. A komponens-gráfok mutatják meg az egyes berendezéseknek a megoldáshoz tartozó működési sorrendjét. Ebből következik, hogy a komponens-gráf része az aktuális ütemezési-gráfnak. Formálisan az $i$ berendezéshez tartozó komponens-gráf $G'_i(N'_i, A_{1i}, A_{2i})$ $(\subseteq G'(N, A_1, A_2))$ egy olyan S-gráf $(i = 1, 2, \ldots, n)$, ahol

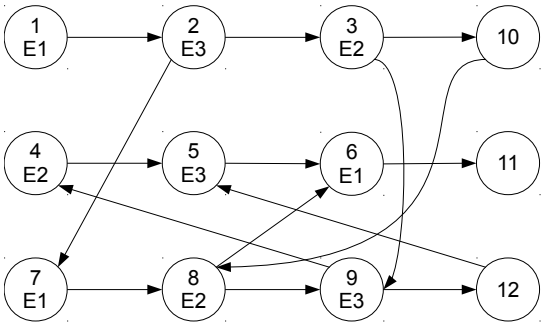
- $N'_i$ tartalmazza $G'$-ből az $M_i$ összes csomópontját és az összes olyan csomópontot, amelybe $M_i$-ből induló recept-él mutat. Azaz tartalmazza azokat a csomópontokat, amelyhez



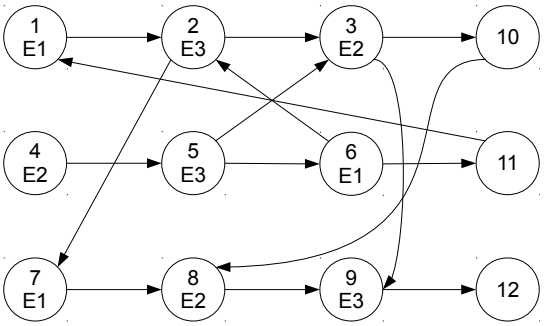

1. ütemezési-gráf



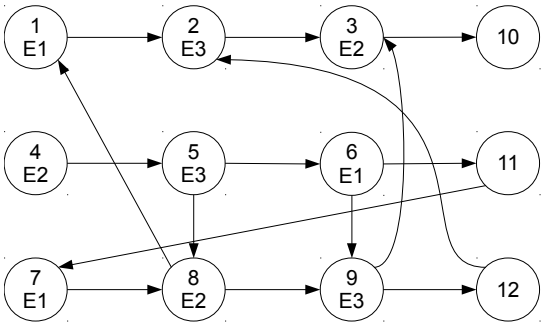

2. ütemezési-gráf



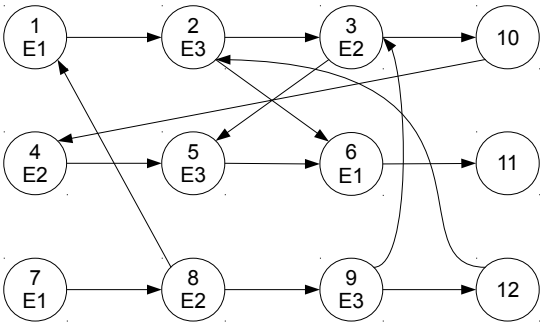

3. ütemezési-gráf



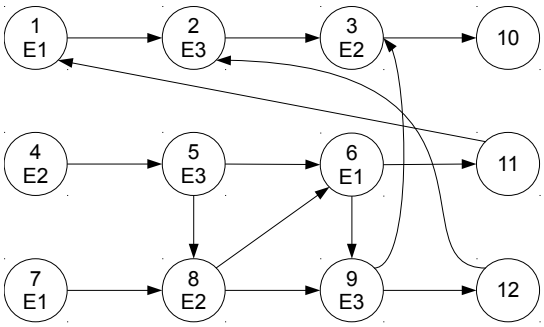

4. ütemezési-gráf



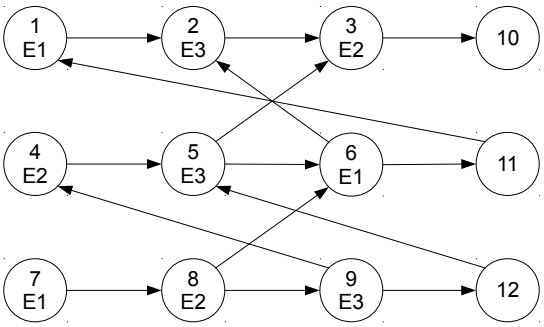

5. ütemezési-gráf




6. ütemezési-gráf




7. ütemezési-gráf




8. ütemezési-gráf

*6.9. ábra. A 6.5. példa ütemezési-gráfjai*

az $i$ berendezés hozzá lett rendelve, valamint a receptben ezeket követőket, ahova az elkészült anyagot át kell tölteni.

$$N'_i = M_i \cup \{k : k \in N \text{ és } \exists j \in M_i, \text{ hogy } (j,k) \in A_1\} \tag{6.1}$$

- $A_{1i}$ tartalmazza az összes olyan recept-élet $G'$-ből, amely $M_i$-ből indul. Azaz az összes olyan recept-élet, amely összeköti a $i$ berendezéshez rendelt csomópontokat a receptben utána következőkkel és amelyen a berendezés működési ideje szerepel.

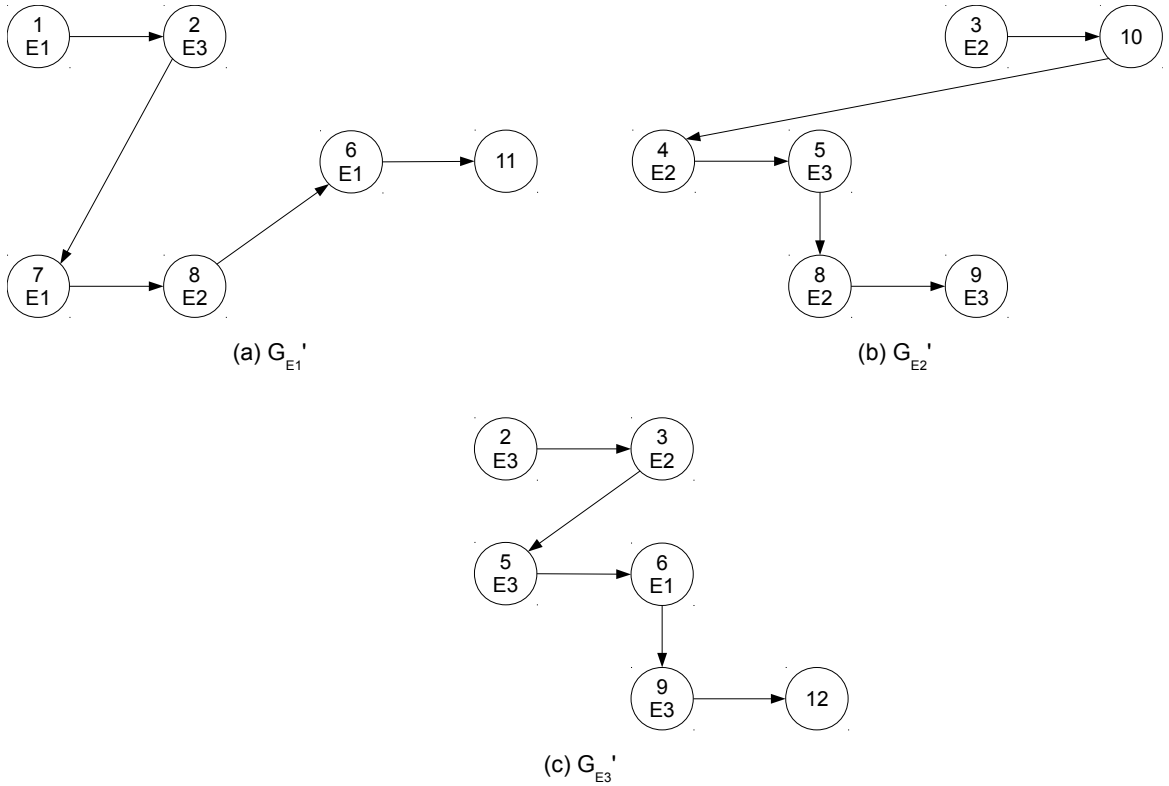
$$A_{1i} = \{(j,k) : (j,k) \in A_1 \text{ és } j \in M_i\} \tag{6.2}$$

- $A_{2i}$ tartalmazza az összes olyan ütemezési-élet $G'$-ből, amely $A_{1i}$ valamely élének végpontjából $M_i$ valamelyik elemébe mutat. Azaz az összes olyan ütemezési élet, amely az $i$ berendezés működési sorrendjére vonatkozó döntéseket jelöli.

$$A_{2i} = \{(j,k) : (j,k) \in A_2, k \in M_i \text{ és } \exists l \in M_i, \text{ hogy } (l,j) \in A_{1i}\} \tag{6.3}$$

**6.5. példa (további vizsgálat).** A 2. ütemezési-gráf (6.9 ábra) komponens-gráfjai $G'_{Ei}$ ($i = 1, 2, 3$) a 6.10 ábrán láthatóak a következő berendezés-taszk hozzárendeléssel: $M_{E1} = \{1, 6, 7\}$, $M_{E2} = \{3, 4, 8\}$, $M_{E3} = \{2, 5, 9\}$.

A komponens-gráf segítségével a következőképpen definiáljuk az ütemezési-gráfot. A $G(N, A_1, \emptyset)$ recept-gráfhoz és az $M_i$ ($i = 1, 2, \ldots, n$) berendezés-taszk hozzárendeléshez tartozó $G'(N, A_1, A_2)$ S-gráf ütemezési-gráf, ha teljesíti a következő négy feltételt.




*6.10. ábra. A 6.5. példa 2. ütemezési-gráfjának (6.9 ábra) komponens-gráfjai.*

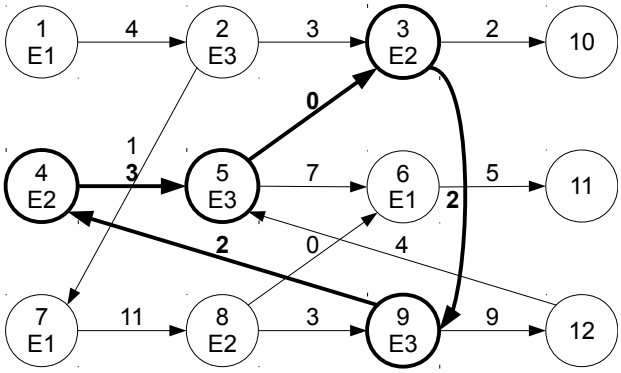
(SG1) $G'$ nem tartalmaz irányított kört.

(SG2) $G'_i$ komponens-gráf által meghatározott rendezés teljes minden $M_i \cup \{j\}$ halmazon, ahol $j \in N'_i$ és $i = 1, 2, \ldots, n$.

(SG3) Az $N'_i$ $(i = 1, 2, \ldots, n)$ halmazok minden eleméből legfeljebb egy $A_{2i}$-beli él indul.

(SG4) A $G'$ ütemezési-gráf megegyezik komponens-gráfjainak uniójával, azaz $G' = \bigcup_{i=1}^{n} G'_i$.

Az ütemezési-gráfok és a megvalósítható ütemezések közötti kapcsolat azon alapul, hogy az $(i, j)$ él az S-gráfban azt mutatja, hogy a $j$ csomópont által reprezentált taszk leghamarabb $c(i, j)$ idővel később kezdődhet el, mint az $i$ által reprezentált. Ebből következik, hogy ha az (SG1) feltétel nem teljesül, akkor a megoldás nem megvalósítható, mivel vagy az időbeliség vagy a NIS tárolási stratégia feltételei nem teljesülnek. A 6.11 ábrán látható S-gráf irányított kört tartalmaz, mely vastag vonallal van jelölve. Ha a kör mentén végignézzük az élek jelentését, akkor azt tapasztaljuk, hogy a 4-es taszknál 3 időegységgel később kezdeheti el a munkáját az 5-ös, annál nem kezdődhet hamarabb a 3-as, azután leghamarabb 2-vel kezdődhet a 9-es, majd 2-vel a 4-es. Ez azt jelenti, hogy a 4-es tasz 7 időegységgel később kezdheti el a munkát a 4-es taszknál, azaz ellentmondáshoz jutottunk.

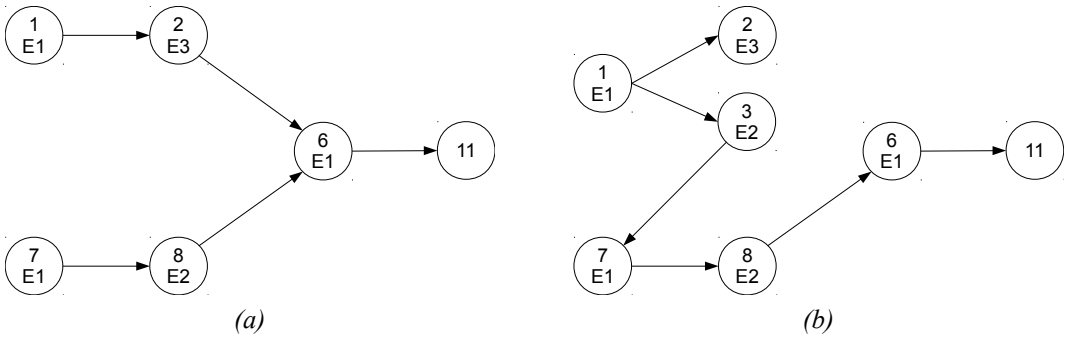



*6.11. ábra. Irányított kör az S-gráfban.*

Tegyük fel, hogy (SG1) feltétel teljesül, de (SG2) nem. Ekkor két esetet különböztethetünk meg: nincs teljes rendezés az $M_i$ halmazon, vagy teljes rendezés van az $M_i$ halmazon, de van olyan $j(\in N'_i)$, hogy ezzel kiegészítve $M_i$-t nincs teljes rendezés. Az első esetben, ha nincs teljes rendezés az $M_i$ halmazon, akkor létezik két olyan taszk, amelyek működési sorrendje nem egyértelműen meghatározott. Például a 6.12a ábra S-gráfja teljesíti a komponens-gráf definícióját, de az E1 berendezéshez tartozó 1. és 7. csomópontok között nincs út, ezért ezek működési sorrendjéről nem tudunk semmit. A második esetben a NIS stratégia sérül. Például a 6.12b ábra komponens-gráfján látható, hogy bár az E1 berendezéshez tartozó csomópontok (1, 6, 7) között teljes a rendezés, de hiányzik egy ütemezési-él a 2-es és a 7-es csomópont között, így az E1 berendezésnek nem kell megvárnia a 2. taszkon dolgozó E3 berendezést az anyag áttöltéséhez.

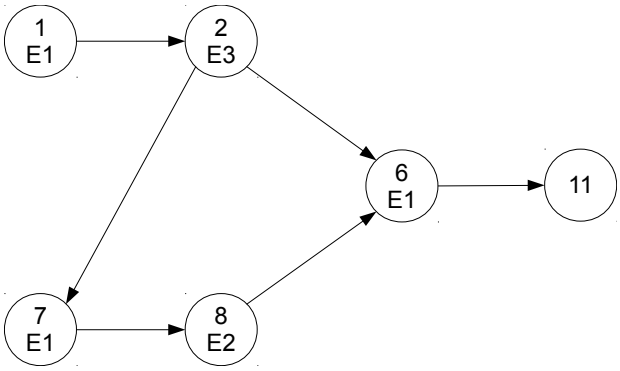
Az (SG3) és (SG4) feltételek teljesülése nem szükséges ahhoz, hogy az ütemezési-gráf által reprezentált megoldás megvalósítható és teljes legyen, de az optimális megoldás mindig megtalálható az (SG3) és (SG4) által leszűkített keresési térben. (SG3) biztosítja, hogy ne




*6.12. ábra. Az SG2 szabály megsértése*

legyenek redundáns ütemezési-élek a gráfban, azaz a minimális számú éllel valósuljon meg az ütemezés. A 6.13 ábrán látható komponens-gráfban a 2. és 6. csomópontok közötti ütemezési-él redundáns, nem hordoz információt, mivel az E1 berendezés mindenképpen csak az 1 taszk végrehajtása után kezdheti meg a 6. taszkot, ha már a 7 taszkot is végrehajtotta. Az (SG4) feltétel pedig kiszűri azokat az éleket, amelyek nem tartoznak egyik berendezés ütemezéséhez sem, azaz minden él vagy a recept-gráf recept-éle vagy valamely berendezés ütemezéséhez tartozó ütemezési-él. Az előbbeikből következik, hogy az (SG3) és (SG4) feltételek teljesülése esetén az S-gráf minimális abban az értelemben, hogy él elhagyásával nem ír le megoldást.




*6.13. ábra. Felesleges ütemezési-élek az S-gráfban.*

## 6.5. Alapalgoritmus ütemezésre S-gráffal

A fejezetben Sanmartí és társai [19] által bevezetett alapalgoritmust mutatjuk be, amely általánosan használható bármilyen típusú ütemezési feladatra. Az algoritmus bemenetként megkapja a feladat recept-gráfját, kimenete pedig egy optimális ütemezés ütemezési-gráfja. Az algoritmusban a cél az optimális megoldás (ütemezési-gráf) megtalálása a receptből (recept-gráf) kiindulva részfeladatok (általános S-gráf) sorozatán keresztül. Definíció alapján a recept-gráf mindig részgráfja a hozzá tartozó összes ütemezési-gráfnak úgy, hogy a gráfok csomópontjai nem változnak, és a recept-éleknek is legfeljebb a súlya növekedhet a berendezés-taszk hozzárendelés alapján. Az előző fejezetben leírtak szerint egy ütemezési-gráf minden éle,

amely nem tartozik a kiinduló recept-gráfhoz, ütemezési-él. Ha figyelembe vesszük az (SG1)-(SG4) feltételeket és a lehetséges berendezés-taszk hozzárendeléseket, akkor a recept-gráf összes, a feltételeknek megfelelő kiterjesztése ütemezési-élekkel elvezet bennünket az összes ütemezési-gráfhoz. Mivel a lehetséges kiterjesztések száma véges, az összes ütemezési-gráf a hozzá tartozó berendezés-taszk hozzárendelésekkel véges lépésben előállítható. Az S-gráf leírás jó alap egy szétválasztás és korlátozás (branch-and-bound, B&B) típusú algoritmushoz.

### 6.5.1. Szétválasztási lépés

Az alábbiakban bemutatandó szétválasztás és korlátozás algoritmus képes előállítani bármely recept-gráfhoz a hozzá tartozó összes ütemezési-gráfot, ahol minden részfeladathoz egy S-gráf és egy részleges berendezés-taszk hozzárendelés tartozik, valamint természetesen egy csomópont a keresőfában. A keresőfa gyökeréhez a recept-gráf tartozik, a leveleihez pedig ütemezési-gráfok.

A szétválasztás során minden részfeladatnál kiválasztunk egy berendezést. A részfeladat minden gyerek részfeladatában a berendezést hozzárendeljük egy megfelelő, még be nem ütemezett taszkhoz (természetesen mindegyik részfeladatban másikhoz), valamint a kiválasztott taszkot beütemezzük a berendezés aktuálisan utolsó lépésének. Mivel a taszkok működési ideje függhet a felhasznált berendezéstől, ezért egy berendezés hozzárendelése egy taszkhoz módosíthatja a taszkot jelölő csomópontból kiinduló recept-él (vagy elágazás esetén recept-élek) súlyát. Az egyszerűség kedvéért feltételeztük, hogy a megoldásban egy taszkot pontosan egy berendezés hajthat végre.

Az algoritmus megvalósításánál nagy szabadságot ad a keresési stratégia kiválasztása. Például, hogy melyik berendezést választjuk ki ütemezésre nagyban befolyásolhatja az algoritmus hatékonyságát. Gyakorlatban a berendezések „leterheltségének” sorrendjében való ütemezés általában jó stratégiának bizonyult. Ezen kívül a részfeladat kiválasztása is jelentős hatással van az algoritmus futási idejére. A számítógépes megvalósításnál érdemes mélységben először keresést alkalmazni, az egy szinten lévő részfeladatok közül a legjobb alsó korlátút választva.

### 6.5.2. Az algoritmus korlátozási lépése

A korlátozás során először egy megvalósíthatósági vizsgálatot (feasibility test) végzünk el. Ha az aktuális részfeladat megvalósíthatónak (feasible) bizonyul, akkor meghatározunk egy alsó korlátot a részfeladatból elérhető megoldások működési idejére. Ha a köztes termékek várakozási ideje két taszk között nem korlátozott, akkor az S-gráf által reprezentált megoldás akkor és csak akkor megvalósítható, ha a gráf nem tartalmaz irányított kört, azaz az S-gáf egy ütemezési-gráf. Ezek alapján a megvalósíthatósági vizsgálat elvégezhető egy körkereső algoritmus segítségével, melyre léteznek hatékony polinomiális algoritmusok. Ha a köztes termékek várakozási ideje korlátozva van (ZW tárolási stratégia), akkor további vizsgálat szükséges, például egy lineáris programozási (LP) feladat felírása és megoldása.

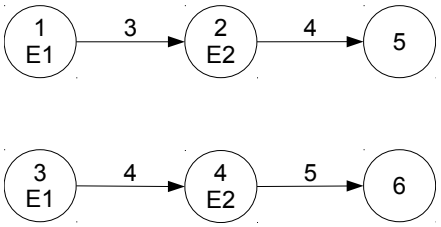
Egy részfeladat S-gráfja mindig részgráfja bármely leszármazott részfeladat S-gráfjának, mivel a gráfot a szétválasztás során legfeljebb éllel bővítjük, de soha nem törlünk belőle. Mivel egy új (nemnegatív súlyú) él hozzáadása egy gráfhoz nem csökkentheti a leghosszabb

utat, ezért egy ütemezési-gráf bármely részgráfjához tartozó leghosszabb út alsó korlát az ütemezési-gráf leghosszabb útjára. Továbbá egy ütemezési-gráfhoz tartozó leghosszabb út értéke alsó korlátja a hozzá tartozó megoldás értékének, mivel minden $(i, j)$ él egy S-gráfban azt jelenti, hogy a $j$ csomópont által reprezentált taszk elvégzése leghamarabb $c(i, j)$ idővel később kezdődhet el az $i$ csomópont által reprezentált taszk elkezdéséhez képest. Ebből következik, hogy egy ütemezési-gráf bármely részgráfjához tartozó leghosszabb út alsó korlátot ad az ütemezési-gráfhoz tartozó megoldás értékére.

### 6.5.3. Az algoritmus működésének szemléltetése

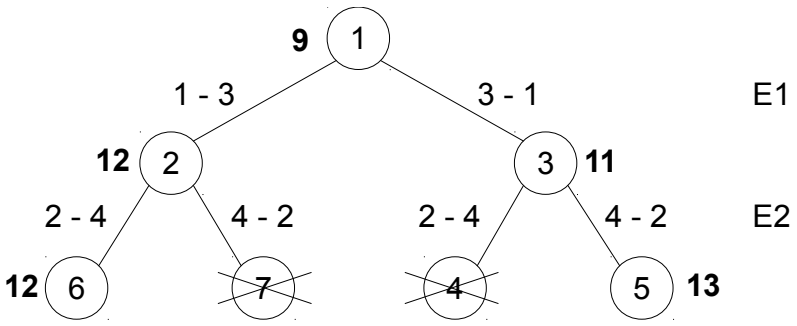
A Sanmartí és társai [19] által publikált alapalgoritmus a leghosszabb út kereső algoritmust használja, ezért az algoritmus működésének bemutatásához mi is ezt használjuk. A szemléltető példa recept-gráfja a 6.14 ábrán látható, ahol két terméket kell előállítani két lépésben két berendezéssel.



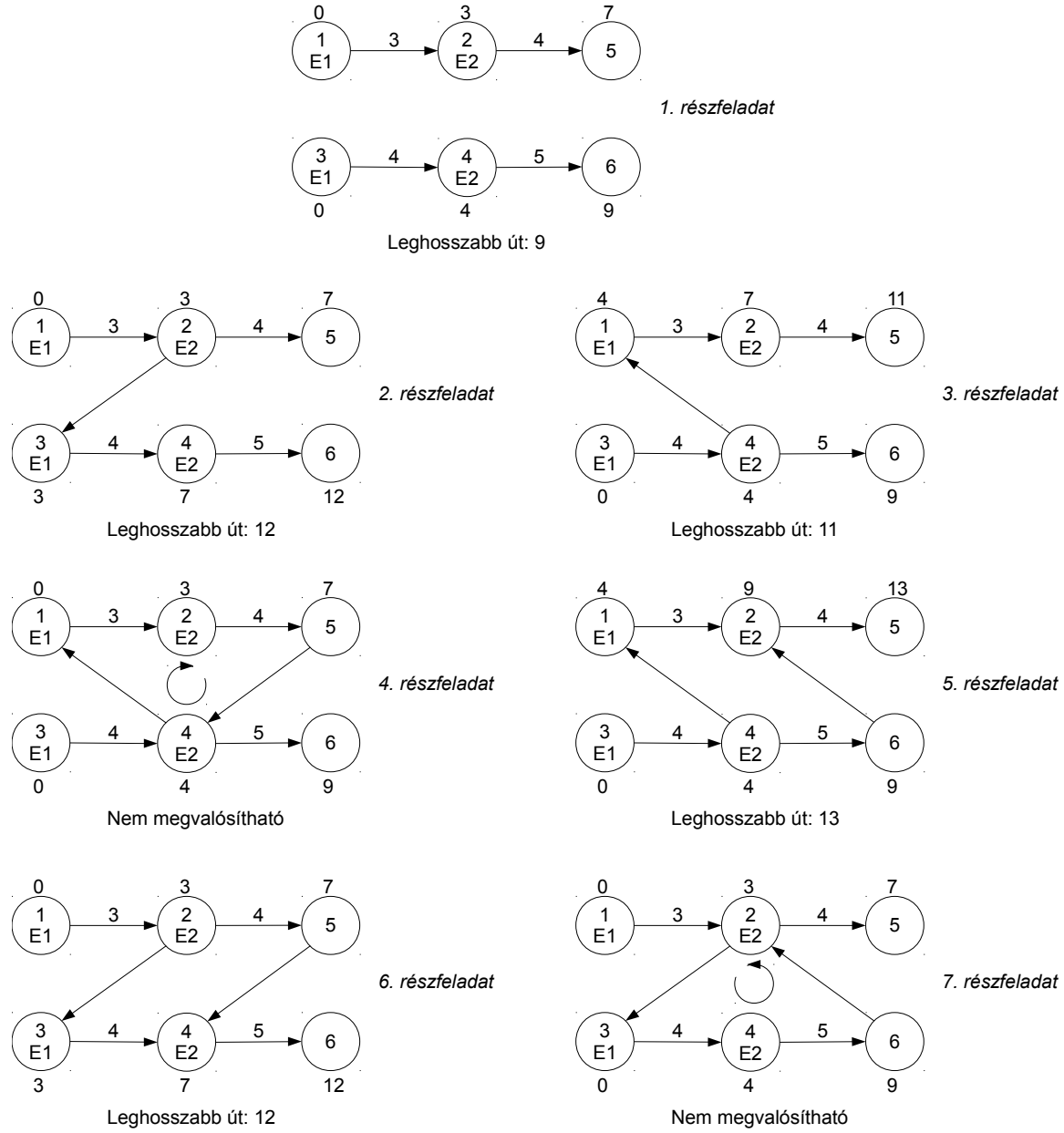

*6.14. ábra. Recept-gráf a szemléltető példához.*

A példához tartozó kereső fa a 6.15 ábrán látható. A csomópontok a részfeladatokat jelölik és a generálási sorrend szerint számoztuk őket. A szétválasztásokat (döntéseket) a taszkok sorrendjével jelöljük, amelyeket meghatároznak. A megoldást leegyszerűsítettük annyiban, hogy a két taszk-berendezés hozzárendelés és a sorrendjük meghatározása egy lépésben történik meg. Az alsó korlát minden részfeladat csomópontja mellett félkövér karakterekkel szerepel. A nem megvalósítható részfeladatok csomópontjai át vannak húzva (pl. 4. részfeladat). A részfeladatokhoz tartozó S-gráfok a 6.16 ábrán láthatóak.

A keresőfa gyökeréhez (1. részfeladat) tartozik a recept-gráf, itt a leghoszszabb út hossza, azaz az alsó korlát értéke 9. Ebből a részfeladatból két új részfeladatot származtatunk (2. és 3. részfeladat) az E1 berendezés ütemezésével: 1–3 illetve 3–1 sorrendekkel, 12 illetve 11 alsó




*6.15. ábra. A szemléltető példához tartozó keresőfa.*




*6.16. ábra. A szemléltető példa részfeladataihoz tartozó S-gráfok*

korláttal. A legjobb alsó korláttal a 3. részfeladat rendelkezik, tehát ennek szétválasztásával folytatjuk az algoritmust. Ebből a részfeladatból a szétválasztás után két újabb részfeladat keletkezik (4. és 5. részfeladat) az E2 berendezés 2–4 illetve 4–2 sorrend választásával. Mind a két részfeladat egy teljesen beütemezett megoldást jelöl, azaz egy új felső korlátot kaphatunk a feladatunkhoz. A 4. részfeladat S-gráfja kört tartalmaz, tehát nem megoldható ezért eldobjuk. Az 5. részfeladat S-gráfja ütemezési-gráf, ahol a leghosszabb út 13. Ez jelenti az új felső korlátunkat az optimumhoz. Az algoritmus a 2. részfeladat szétválasztásával folytatódik, ahol az E2 berendezésről dönthetünk 2–4 illetve 4–2 ütemezési sorrendekkel. Így két részfeladat generálódik (6. és 7. részfeladat), amelyek teljesen beütemezett megoldást jelölnek. A

6. részfeladat egy megoldást jelöl, amelynek értéke 12, ami jobb, mint az aktuális felső korlátunk, ezért ez lesz az új felső korlátunk. A 7. részfeladat kiértékelése nem szükséges, mert a szülőjének (2. részfeladat) alsó korlátja nem jobb, mint az aktuális felső korlát. Ha mégis kiértékeljük, akkor kiderül, hogy a hozzá tartozó S-gráf irányított kört tartalmaz, tehát nem megvalósítható megoldást jelöl.

## 6.6. S-gráf módszertan kiterjesztései

Az S-gráf módszertant többféle szempont szerint lehet kiterjeszteni. Ezek közül kettőt fogunk a továbbiakban megvizsgálni. Az első esetben a szétválasztás és korlátozás algoritmus keresési tér bejárási módszerét módosítjuk, a másikban pedig a célfüggvényt változtatjuk meg, amely természetesen a keresési tér és a hozzá kapcsolódó megoldási módszer megváltoztatását is magával vonja.

### 6.6.1. Taszk alapú döntési stratégia

Az alapalgoritmusban „berendezés alapú döntési stratégiát” (EQ-SG algoritmus) használtunk. Ebben az esetben döntésnél a részfeladat egy ütemezendő berendezésének a kiválasztása után hozzáfűzünk a berendezés végrehajtási sorának végére egy még be nem ütemezett taszkot. A gyermek részfeladatok az összes lehetséges taszk hozzáfűzést figyelembe véve keletkeznek. Az ebben a fejezetben bemutatandó „taszk alapú döntési stratégiát” használó algoritmus (TA-SG algoritmus) ugyanazt a keresési teret járja be, mint a berendezés alapú és bizonyos feladatokra hatékonyabb ütemezőt biztosít ([1]). A taszk alapú döntéseket használó algoritmus szétválasztás lépésében a döntések új alapötlete az, hogy rendeljünk hozzá egy még be nem ütemezett taszkhoz egy lehetséges berendezést és szúrjuk be a taszkot a berendezés végrehajtási sorrendjébe figyelembe véve az összes lehetőséget.

Az algoritmus, mint minden szétválasztás és korlátozás elvén működő algoritmus, egy kereső fát valamilyen módon bejárva határozza meg a feladat optimális megoldását. A kereső fa csúcspontjaihoz tartoznak a részfeladatok. Minden részfeladatnak tárolnia kell a kereső fa gyökeréből indulva a részfeladathoz vezető éleken meghozott döntéseket. A kereső fa gyökeréhez a recept-gráf tartozik, leveleihez pedig az ütemezési-gráfok ugyanúgy, mint a berendezés alapú ütemezésnél.

A szétválasztási eljárás a döntési lépésében két műveletet hajt végre: kiválaszt egy olyan taszkot, amely még nincsen beütemezve, majd a kiválasztott taszkot beilleszti egy megfelelő berendezés aktuális ütemezésébe. A taszkot végrehajtani tudó összes lehetséges berendezést hozzá kell rendelni a taszkhoz, azaz befűzzük az összes rendelkezésre álló berendezés aktuális ütemezésébe az új taszkot, ahol minden hozzárendelés egy új részfeladatot jelent.

Tegyük fel, hogy egy $i$ taszkhoz egy berendezés áll rendelkezésre valamint ehhez a berendezéshez már hozzá van rendelve $k$ darab taszk $(j_1, j_2, \ldots, j_k)$ sorrendben. Ebben az esetben legfeljebb $k+1$ darab gyerek részfeladat lesz, ahol a berendezés működési sorrendje rendre $(i, j_1, j_2, \ldots, j_k)$, $(j_1, i, j_2, \ldots, j_k)$, $(j_1, j_2, i, \ldots, j_k)$, …, $(j_1, j_2, \ldots, i, j_k)$, $(j_1, j_2, \ldots, j_k, i)$, pontosabban ezek közül azok, amelyek a megvalósíthatósági teszten megfelelnek.

Abban az esetben, ha az ütemezésre kiválasztott taszkot két különböző berendezéssel lehet végrehajtani, mely berendezések jelenlegi ütemezése már $k_1$, illetve $k_2$ darab sorba fűzött

taszkot tartalmaz, akkor az új taszk valamely berendezéshez való hozzárendelése és befűzése a berendezések taszk végrehajtási sorába $(k_1 + 1) + (k_2 + 1)$ darab új gyermek részfeladatot eredményez, mivel egymástól függetlenül mindkét berendezéssel végre lehet hajtani a taszkot. Mindkét berendezés esetén az új ütemezendő taszkot be lehet fűzni valamelyik jelenleg is szereplő taszk elé, vagy pedig a taszk végrehajtási sor legvégére, azaz összesen láncolt lista elemszáma plusz egy új lehetőség áll fent.

Az EQ-SG algoritmus esetén a szülő részfeladat S-gráfjának leghosszabb útja része a gyermek részfeladat S-gráfjának, hiszen a gyermek részfeladat S-gráfjában ugyanazok az élek szerepelnek, legfeljebb a recept-élek súlya növekedhet. Ezért teljesül, hogy a gyermek részfeladat alsó korlátja nem kisebb, mint a szülő részfeladaté. A TA-SG algoritmus során a gyermek részfeladatban a berendezések ütemezésének változásakor ütemezési-él törlődik a szülő részfeladat S-gráfjához képest, ami miatt előfordulhat, hogy a szülő részfeladat S-gráfjának leghosszabb útja nem szerepel a gyermek részfeladat S-gráfjában. A TA-SG algoritmus akkor oldja meg helyesen a feladatot, ha a következő feltétel teljesül.

Ha az ütemezési feladat receptjének váltási idejeire teljesül a háromszög egyenlőtlenség, azaz bármely berendezés esetén teljesül, hogy $t_{ij} \leq t_{ik} + t_{kj}$, ahol $t_{ij}$, $t_{ik}$, $t_{kj}$ a berendezés váltási ideje az $i, j$, az $i, k$ és a $k, j$ taszkok között. Ilyenkor a szülő részfeladatból a taszk alapú döntésen alapuló szétválasztás eljárással előállított bármely gyermek részfeladatnak az alsó korlátjai nem kisebbek, mint a szülő részfeladat alsó korlátja.

A feltétel a gyakorlatban azt jelenti, hogy két taszk közötti tisztítást nem kiváltani két összeségében rövidebb idejű tisztítással. Formálisan ennél élesebb korlátot is lehet adni, amelyre még mindig helyesen működne az algoritmus, de a gyakorlatban ez a feltétel mindig teljesül.

A taszk alapú döntéseket alkalmazó szétválasztási eljárás nem befolyásolja az EQ-SG algoritmus korlátozás eljárását, ezért változtatás nélkül használhatóak az EQ-SG algoritmusban bevezetett körkereső és leghosszabb út kereső algoritmust használó korlátozás eljárás. A korlátozás lépésben a részfeladat megvalósíthatóságát az S-gráf körmentességének vizsgálatával döntjük el. Amenynyiben a részfeladat megvalósítható ütemezést jelöl, akkor a leghosszabb út kereső algoritmussal számolhatjuk ki a részfeladatból elérhető ütemezések végrehajtási idejének alsó korlátját.

### 6.6.2. Profit maximalizálása

Ebben a fejezetben a célunk nem a legrövidebb működési idő meghatározása lesz, hanem adott idő alatt a lehető legnagyobb profitot (hasznosságot) kell elérni és ehhez meghatározni, hogy melyik termékből mennyi adagot kell/lehet legyártani. Ipari feladatok esetén a profit maximalizálás a leggyakoribb célfüggvény. Ebben az esetben az előállítandó termékmennyiségek nem adottak, hanem a termékek hasznossági mutatóját és egy időhorizontot kell bevezetni, ahol a hasznossági mutató megadja, hogy egy adagnyi termék mekkora hasznossággal (profittal) jár, az időhorizont pedig felső korlátot jelent a rendszer működési idejére.

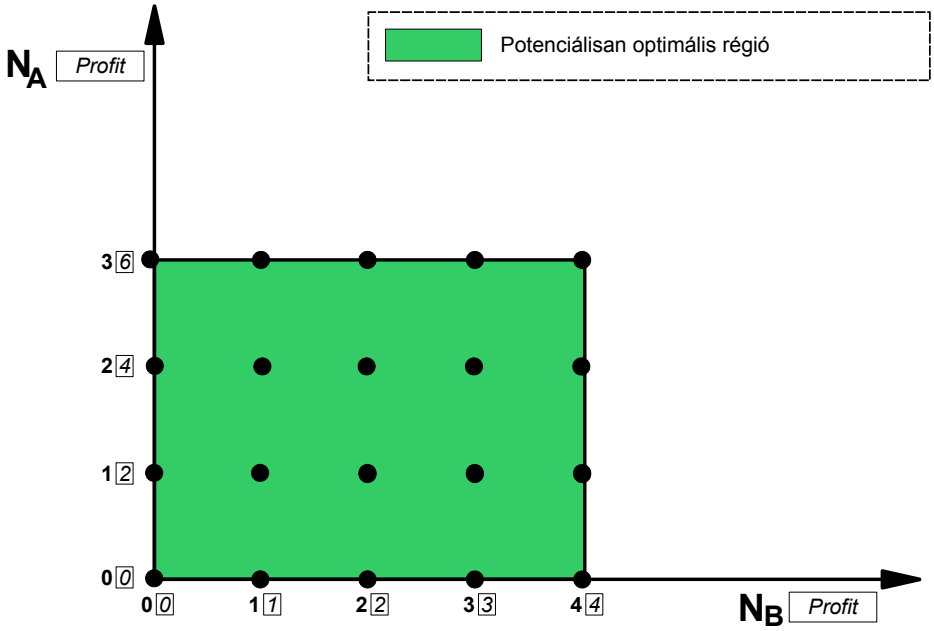
A megoldást egy irányított keresési módszerrel határozzuk meg ([14]). A keresési módszer hatékonysága két dolgon alapul. Először, a termékek adagjainak (batch-jeinek) száma rögzített a keresési tér minden pontjában és ez az érték minden pontban más és más, így kizárjuk a redundanciát a keresés során. Másodszor a keresési térből egy előfeldolgozás során

kizárjuk azon csomópontokat, melyek olyan adagszámokhoz tartoznak, amelyek nem lehetnek optimálisak, így a keresésünk sokkal gyorsabb, mintha teljes leszámlálást alkalmaznánk.

Mivel a keresési tér minden pontjában az adagok száma rögzített, ezért használhatjuk a korábban megismert S-gráf algoritmust. Ezekben a részfeladatokban nem kell az optimális működési idejű megoldást megkeresni, elég, ha az algoritmus talál egy, az időhorizonton belül megvalósítható megoldást az adott adagokkal. Azaz ha az ütemezés során tálalunk egy az időkorláton belüli ütemezést, akkor a többi részfeladatot már nem is kell megvizsgálni.

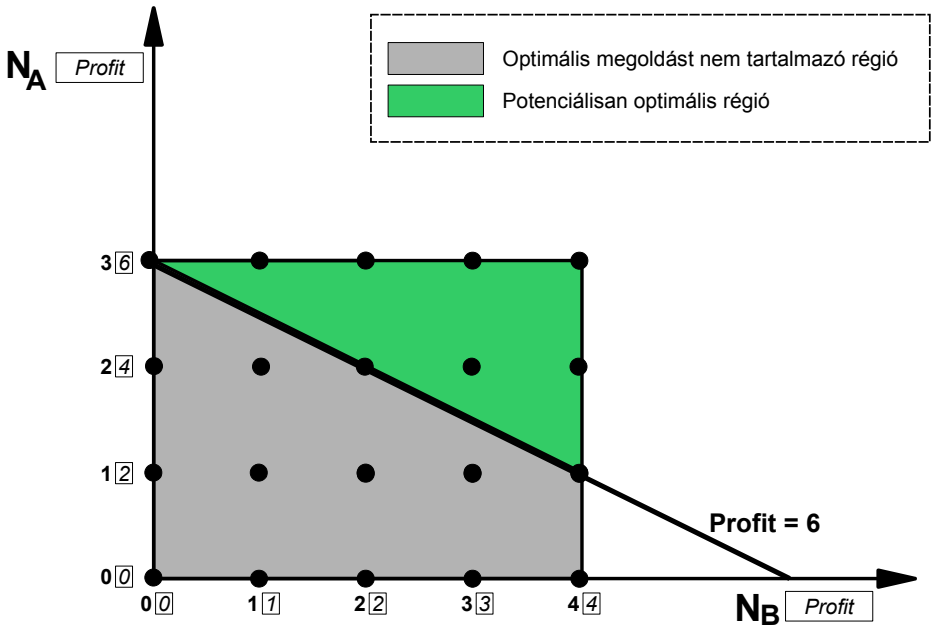
A keresési stratégia könnyebb érthetőségének kedvéért tekintsünk egy példát két termékkel (A és B). Az A termék egy adagjának a profitja (hasznossága) legyen 2 egység ($R_A$), B terméknél ez legyen 1 ($R_B$). Továbbá tételezzük fel, hogy ha csak egy terméket gyártunk, akkor A termékből az időhorizonton belül 3 adagot tudunk előállítani, B-ből pedig 4-et. Ekkor a keresési teret a 6.17 ábrán látható módon ábrázolhatjuk, ahol a függőleges tengelyen az A termék adagjainak számát ($N_A$), a vízszintesen a B termék adagjainak számát ($N_B$) jelöljük. Az adagszámok után bekeretezve az aktuális profit értéke található.




*6.17. ábra. Keresési tér két termék esetén*

A függőleges és a vízszintes tengelyek metszéspontja tartozna egy szokásos szétválasztás és korlátozás algoritmus keresőfájának gyökeréhez. Ez a metszéspont nulla adagot jelent minden termékből. A kereséséi tér vízszintes és a függőleges határát termékek maximális adagszáma adja meg, amit az időhorizont alatt elő lehet állítani belőlük. Természetesen a keresési tér csak a csúcsokat tartalmazza nem a teljes bejelölt területet.

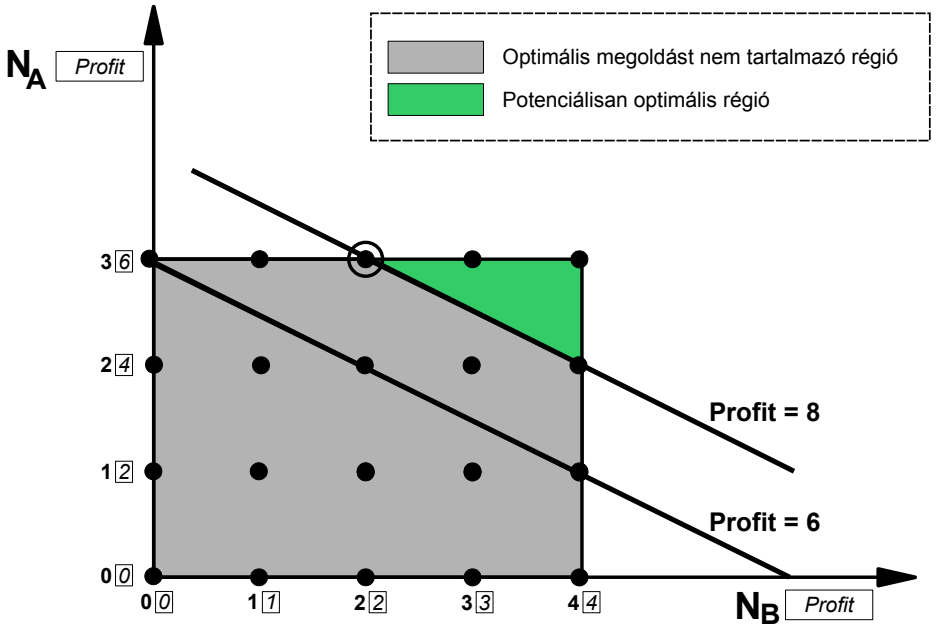
Mivel minden terméknél az egy adaghoz tartozó profit fix, ezért minden csomópontban a profitot az $N_A R_A + N_B R_B$ lineáris egyenlettel számolhatjuk. A 6.17 ábrán látható, hogy az A termékből 3 adag előállítása esetén 6 egységnyi profitot tudunk elérni, ezért, mivel ez egy megoldást jelöl, az ennél kisebb értékkel rendelkező csomópontokat nem kell figyelembe vennünk. A keresési térben a 6 egységi profitnál kisebb értékű csomópontok az $N_A R_A + N_B R_B = 6$



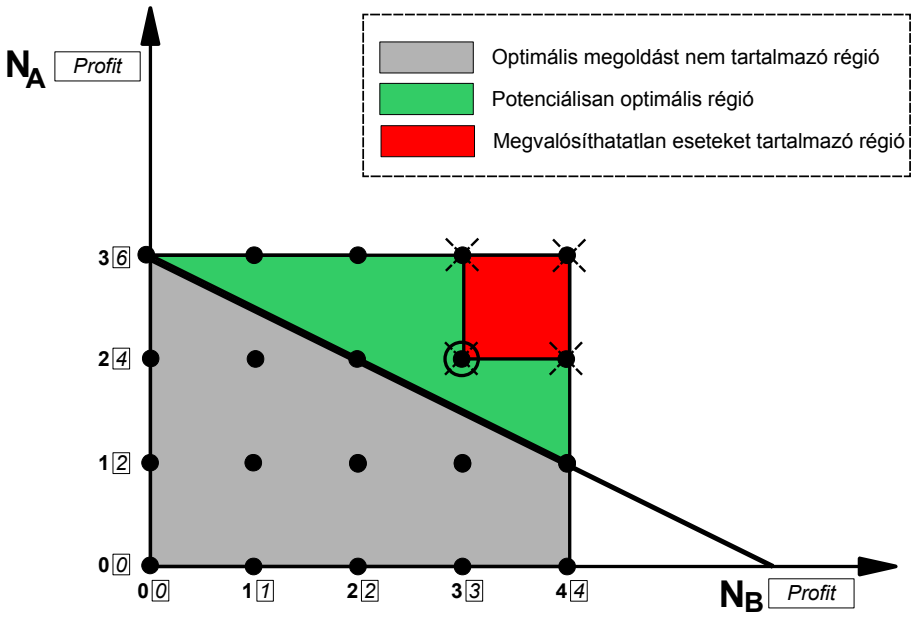

*6.18. ábra. Profit-egyenes*

egyenes (profit-egyenes) alatt találhatóak, ezeket kizárhatjuk a keresésből (lásd 6.18 ábra). Ha egy csomópont a vonalon fekszik, azt sem kell megvizsgálni, mivel az nem adhat jobb megoldást a már megtalált 4 adagnyi A-nál.

A keresés során a profit-egyenes helye változhat, amikor találunk egy az aktuálisnál jobb megoldást. Az új profit-egyenes az eredetivel párhuzamos lesz, mivel ha például az új profit értéke 8, akkor az egyenes egyenlete $N_A R_A + N_B R_B = 8$ lesz (6.19 ábra).




*6.19. ábra. A profit-egyenes mozgatása*




*6.20. ábra. Keresési tér csökkentése*

Ha a keresés során nem megvalósítható csomópontra (részfeladatra) találunk, akkor a keresési teret tovább csökkenthetjük. Tegyük fel, hogy az A-ból 2 adag és B-ből 3 adagot jelentő csomópont nem megvalósíthatónak bizonyult. Ez azt jelenti, hogy ekkora mennyiségű terméket nem lehet az időhorizont alatt előállítani. Ebből viszont az is következik, hogy többet sem lehet, azaz minden olyan csomópontot, amely legalább 2 adag A-t és 3 adag B-t tartalmaz, kizárható a keresési térből. Ez a keresési terünkben egy téglalap alakú területet jelent az aktuális csomópontunk felett jobbra (6.20 ábra).

## 6.7. S-Graph Studio: Program az S-gráf keretrendszerhez

Az S-Graph Studio egy szoftver csomag, amely segít megtervezni szakaszos rendszereket és optimalizálja őket különböző módszerekkel. A program grafikus felhasználói felülettel rendelkezik a feladatok megadásához. Az S-Graph Studio az iparban a szakaszos működésű rendszerek leírásához használatos BacthML szabványú állományokat használja. Ezen kívül a program képes kezelni megfelelő formátumú Excel állományokat is. A programban használt megoldó a Pannon Egyetem Műszaki Informatika Karának Rendszer- és Számítástudományi tanszékén kidolgozott S-gráf keretrendszer alapján készült. A jegyzet mellékletén megtekinthető egy videó, amely szoftvert működés közben mutatja be.

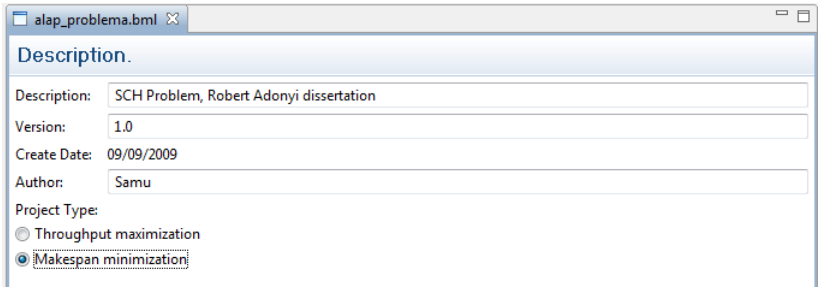
Az S-Graph Studio használata három fő lépésből áll, melyeket a következőkben egy példán keresztül mutatunk be.

1. Ütemezési feladat készítése, amely lehet egy új ütemezési feladat vagy egy korábbinak a betöltése

2. Az aktuális feladat megoldása különböző beállításokkal

3. Az eredmény megjelenítése és értelmezése

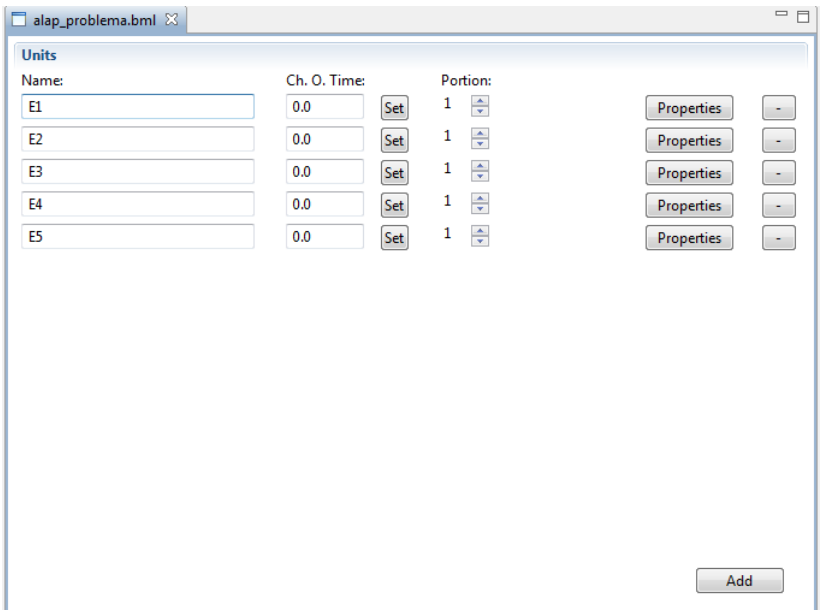
### 6.7.1. Feladat megjelenítése és készítése



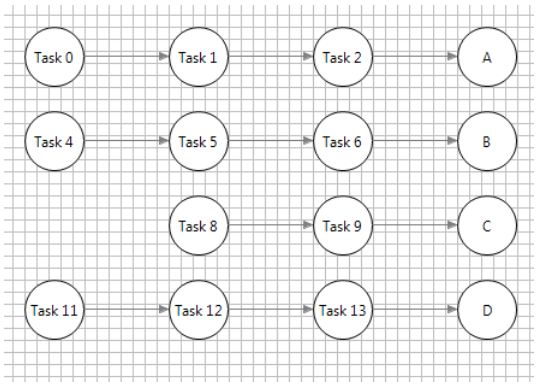

*6.21. ábra. Egyéb adatok az ütemezési feladathoz*

Az S-Graph Studio az S-gráf keretrendszeren alapul, így az ütemezési feladatok megadása egy recept-gráf megadását jelenti. Először a recept-gráf elemeit kell definiálni: a taszkokat, a taszkok sorrendjét, a rendelkezésre álló berendezéseket, a berendezések működési és tisztítási idejét az aktuális taszkhoz, valamint az előállítandó termékmennyiségeket. A bemeneti adatok három csoportra vannak osztva: berendezés adatok (unit data), recept adatok (recipe data) és egyéb adatok (additional information). Az utolsó csoportban, amely a 6.21 ábrán látható, lehet megadni a bemenet verziószámát (version), a készítőjét (author), leírását (description) és a feladat típusát (problem type). A feladat típusa lehet profit maximalizálás (throughput maximization) vagy működési idő minimalizálás (makespan minimization).

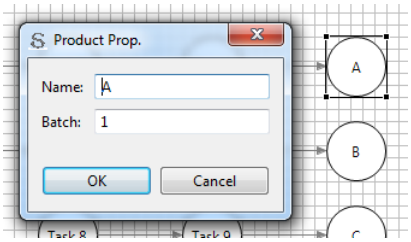
A következőkben a berendezések adatainak megadása következik (6.22 ábra). Minden berendezésre definiálni kell a berendezés nevét (Name), a tisztítási időket (Ch. O. Time) és azt,




*6.22. ábra. Berendezések adatainak beállítása*



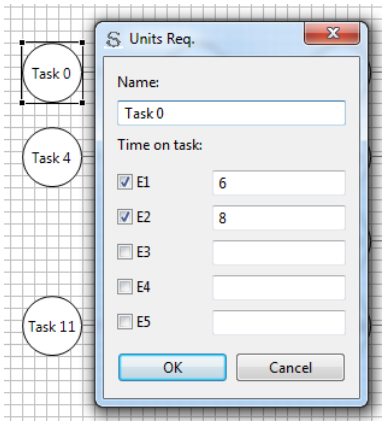

*6.23. ábra. Szemléltető példa recept-gráfja S-Graph Studio-ban*




*6.24. ábra. A termékek tulajdonságainak megadása*

hogy az adott berendezésből hány darab áll rendelkezésre (Portion). A programban a tisztítási idő nem függ attól, hogy a berendezés milyen taszkot végzett el korábban és milyen taszkot fog elvégezni. Léteznek olyan feladatok, ahol a tisztítási idő függ a kapcsolódó taszkoktól. Például festékgyártásnál egy világosabb festék gyártása után nem kell olyan alaposan kitisztítani a berendezést, mint egy sötét után.

Következő lépés a recept-gráf grafikus megadása (6.23 ábra), ahol a recept-gráf tartalmazza a taszkok halmazát és a közöttük lévő sorrendi feltételeket. S-gráf keretrendszernél a termékek szintén a recept-gráfban adottak. Minden termékhez meg kell adni a nevét (Name) és a szükséges adagok számát (Batch), lásd 6.24 ábra. Minden taszkhoz be lehet állítani a taszk

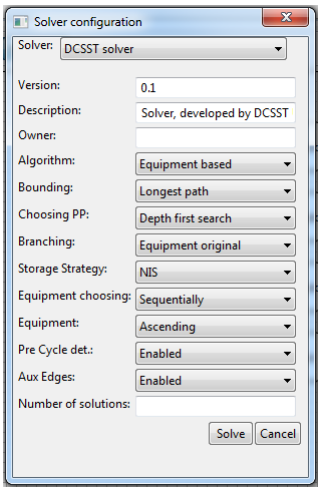



*6.25. ábra. A taszkok tulajdonságainak megadása*

nevét (Name), a taszk végrehajtásához használható berendezéseket és a hozzájuk kapcsolódó működési időket (Time on task), lásd 6.25 ábra. Ha mindezeket megtettük, akkor definiáltuk az ütemezési feladatot.

### 6.7.2. Feladat megoldás

Az S-Graph Studio-ban a felhasználói felület és a megoldó teljesen külön van választva, így különböző megoldók használhatók és ezek cseréje is egyszerű. A megoldót ki kell választani futtatás előtt és természetesen a megoldóhoz tartozó paramétereket beállítani (6.26 ábra). Itt lehet még megadni a tárolási stratégiát (Storage Strategy), amely szintén fontos paramétere a feladatnak, mivel ez nagyban befolyásolja a megoldások struktúráját. További paraméterekkel lehet befolyásolni a megoldó sebességét, mint például a részfeladat kiválasztásának módszere (Chosing PP) a korlátozási eljárás (Bounding), az előzetes körfelismerés (Pre Cycle det.) vagy a segéd-élek használata (Aux Edges, [10], [9]).

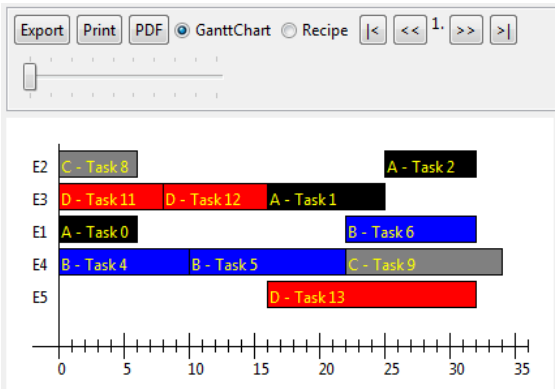



*6.26. ábra. A megoldó paramétereinek beállítása*

### 6.7.3. Az eredmény megjelenítése

A program az ütemezési feladat megoldása ábrázolni tudja Gantt diagrammal (6.27 ábra) vagy ütemezési-gráffal (6.28 ábra). Ezek különböző formátumokba exportálhatók, például Excel állományba, pdf formátumba. A megoldó képes meghatározni a második, harmadik, n-edik legjobb megoldást és mindegyiket meg is tudja jeleníteni. Ezek között a megoldások között a jobb felső sarokban lévő gombokkal navigálhatunk.

Ütemezési-gráf nézetben (6.28 ábra) az S-gráfról leolvasható a recept-gráf és az ütemezési-gráf is. A taszkok receptben definiált sorrendjét a fekete színű recept-élek mutatják, a berendezésekhez rendelt taszkok működési sorrendje pedig a piros színű ütemezési-élek segítségével olvashatóak le. Az élekhez tartozó súly kék színnel az élek közepén szerepel.




*6.27. ábra. A megoldás Gantt diagramja*

## 6.8. Feladatok

**6.1. feladat.** Adja meg az optimális ütemezését a következő két gépes flow shop feladatnak Johnson algoritmus segítségével! Rajzolja fel a megoldás Gantt diagrammját is!

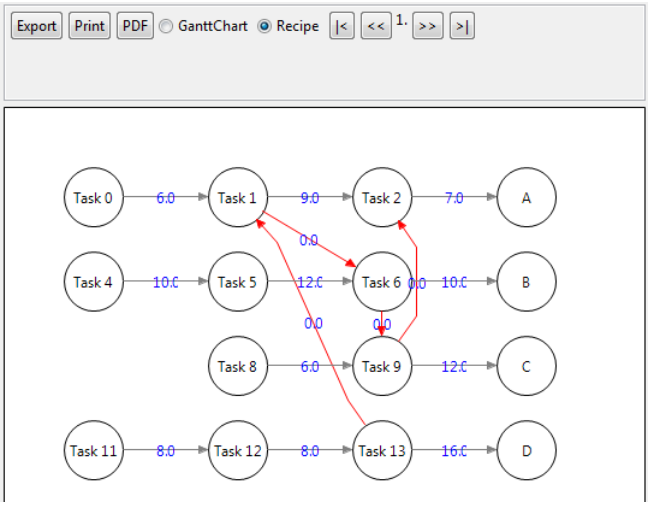
$p_1 = (3, 3), p_2 = (2, 1), p_3 = (2, 4), p_4 = (1, 2), p_5 = (2, 3), p_6 = (3, 1)$

**6.2. feladat.** Adja meg az optimális ütemezését a következő két gépes flow shop feladatnak Johnson algoritmus segítségével! Rajzolja fel a megoldás Gantt diagrammját is!

$p_1 = (2, 4), p_2 = (1, 1), p_3 = (3, 2), p_4 = (4, 2), p_5 = (1, 3), p_6 = (3, 1)$

**6.3. feladat.** Adja meg az optimális ütemezését a következő két gépes job shop feladatnak Jackson algoritmus segítségével! Rajzolja fel a megoldás Gantt diagrammját is!

$J_1 : c_1 = 1, m_{11} = 2, p_1 = (2)$
$J_2 : c_2 = 2, m_{12} = 1, m_{22} = 2, p_2 = (4, 2)$
$J_3 : c_3 = 2, m_{13} = 2, m_{23} = 1, p_3 = (2, 3)$
$J_4 : c_4 = 1, m_{14} = 1, p_4 = (1)$




*6.28. ábra. A megoldás ütemezési-gráfja*

$J_5 : c_5 = 2, m_{15} = 2, m_{25} = 1, p_5 = (1, 5)$
$J_6 : c_6 = 1, m_{16} = 1, p_6 = (5)$
$J_7 : c_7 = 1, m_{17} = 2, p_7 = (3)$
$J_8 : c_8 = 2, m_{18} = 1, m_{28} = 2, p_8 = (2, 4)$

*6.3. táblázat. A 6.4. feladat receptje*

| | A | | B | | C | |
|---|---|---|---|---|---|---|
| Task | Ber. | Idő | Ber. | Idő | Ber. | Idő |
| 1 | E1 | 1 | E2 | 4 | E3 | 2 |
| 2 | E3 | 5 | E1 | 9 | E4 | 8 |
| 3 | E2 | 3 | - | - | E2 | 3 |
| 4 | E4 | 7 | - | - | | |

**6.4. feladat.** Rajzolja fel a 6.3 és 6.4 táblázatban megadott ütemezési feladat recept-gráfját! Három termék (A, B, C) előállítása 4 berendezéssel (E1, E2, E3, E4).

*6.4. táblázat. A 6.4. feladat adagjainak száma.*

| Termék | A | B | C |
|---|---|---|---|
| Adagok | 1 | 2 | 1 |

**6.5. feladat.** Oldja meg az előző feladatot S-graph Studio segítségével.

*6.5. táblázat. A 6.6. feladat receptje*

| | A | | B | | C | |
|---|---|---|---|---|---|---|
| Taszk | Ber. | Idő | Ber. | Idő | Ber. | Idő |
| 1 | E1<br>E2 | 3<br>5 | E3 | 4 | E4 | 5 |
| 2 | E4 | 4 | E2<br>E1 | 10<br>3 | E1 | 1 |
| 3 | E3<br>E2 | 2<br>2 | E4 | 5 | E3<br>E2 | 2<br>4 |

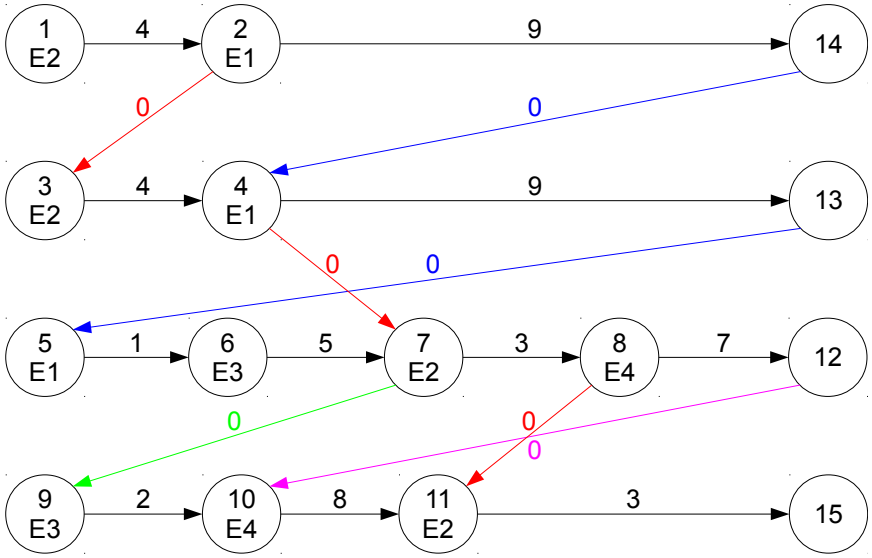
**6.6. feladat.** Rajzolja fel a 6.5 és 6.6 táblázatban megadott ütemezési feladat recept-gráfját! Három termék (A, B, C) előállítása 4 berendezéssel (E1, E2, E3, E4).

**6.7. feladat.** Oldja meg az előző feladatot S-graph Studio segítségével.

**6.8. feladat.** Határozza meg a 6.29 ábrán látható ütemezési-gráf leghosszabb útját és rajzolja fel a Gantt diagrammot.

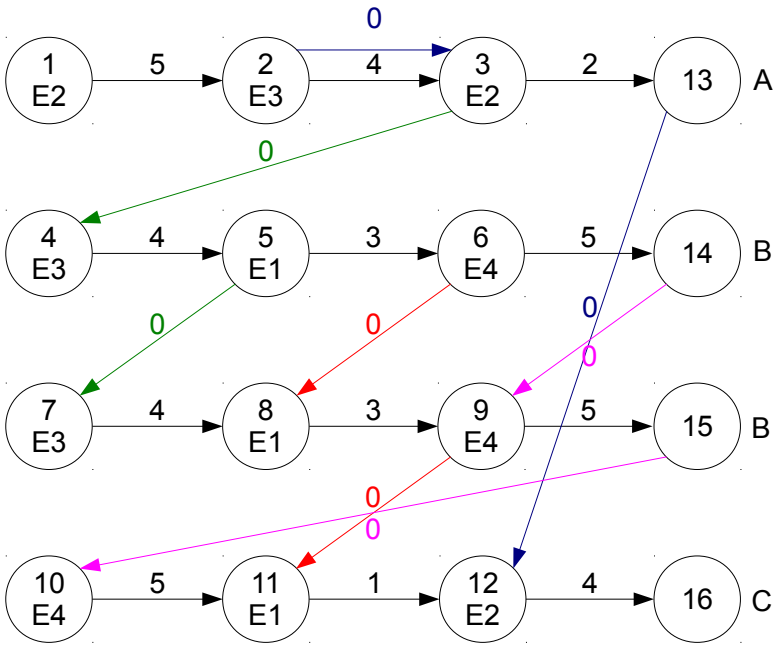
*6.6. táblázat. A 6.6. feladat adagjainak száma*

| Termék | A | B | C |
|---|---|---|---|
| Adagok | 1 | 2 | 1 |




*6.29. ábra. A 6.8. feladat ütemezési-gráfja*

**6.9. feladat.** Határozza meg a 6.30 ábrán látható ütemezési-gráf leghosszabb útját és rajzolja fel a Gantt diagrammot.




*6.30. ábra. A 6.9. feladat ütemezési-gráfja*

# 7. fejezet

# Tévedések és kockázatok

Mint a korábbi fejezetekben is láthattuk, számos módszer áll rendelkezésre folyamatok optimalizálására. Ezek módszerek azonban bemenetükön nem gyártórendszerek és ellátási láncok adatait várják, hanem azok modelljeit. A modell megoldása pedig nem egy vezetői döntésre adott válasz, hanem változók értéke, esetleg, gráfok, struktúrák. Mindebből az következik, hogy a bemutatott eszközök gyakorlati alkalmazása közel sem triviális. A gyakorlati kérdéseket modelleznünk kell, a megoldó módszert kiválasztanunk, majd a megoldást interpretálnunk. Ez a tervező felelőssége. Bármelyik lépésben nagy a tervezői hiba kockázata. A következőkben erre mutatunk néhány példát.

## 7.1. Strukturális leírás

A gyártórendszerek hatékonyságára legnagyobb hatással a struktúrájuk van. Ezért a tervezéskor, optimalizáláskor a strukturális alternatívák feltárása alapvető fontosságú. Hagyomány gráf leírások gyakran elfedik az olyan kérdéseket, hogy egy hálózat építőelemei alternatívái lehetnek egymásnak, vagy szükségszerűen egészítik ki egymást. Ebből a szempontból a P-gráfok alkalmazása célszerű választás. A P-gráfok ezen logikai kapcsolatokat tisztán és expliciten írják le. P-gráfban egy műveleti egység működéséhez minden bemenetének rendelkezésre kell állnia (és kapcsolat), ugyanakkor egy anyagot a hozzá vezető műveleti egységek bármely kombinációja előállíthatja (nem kizáró vagy kapcsolat).

A jó döntéshez alternatívák kellenek. Egy gyakorlati feladat felírásakor az aktuális folyamat leírása mellett minden lépésénél fel kell tennünk azt a kérdést, hogy lehetne-e ezt máshogyan, van-e alternatívája. Az a lehetőség, ami a modellbe nem kerül bele, az a modell megoldásban sem kerülhet elő. Ipari projektekben sokszor már e kérdések felvetései jelentős előrelépést hoz, mert kimozdítja a megrendelőt, egy megszokott állapotból, esetlegesen berögzült rossz megoldásból.

A strukturális leírás másik fontos kérdése, hogy formálisan könnyen kezelhető legyen. Számítógépes tervezésnél, ha gráfokról beszélünk, akkor nem egy papírra vetett ábrára gondolunk, hanem objektumok halmazáról és köztük levő relációkról, melyek algoritmikusan is kezelhetőek. A szakirodalomban számos grafikus módszert ismerünk, melyek tévesen gráfelméletinek titulálnak, de a rajz és a matematikai modell közötti kapcsolat nem formális, algoritmikusan nem értelmezhető.

## 7.2. Megoldások strukturális tulajdonságai

A modellek és megoldó algoritmusok ismerete azért nélkülözhetetlen, mert mindegyiknek vannak korlátai. Ilyen korlát például, hogy milyen struktúrájú megoldást képes generálni. Szétválasztási hálózatok esetén csak a nyersanyagok és a termékek ismertek, valamint többféle művelet, amelyek a nyersanyagból a termékhez vezető hálózatban többször is felhasználhatóak. Ilyen feladatoknál a modellgenerálás része az is, hogy meghatározzuk, hogy a tervezés során bizonyos berendezésből hányat és milyen kapcsolatokkal kívánunk figyelembe venni. Ha ezt nem kellő megalapozottsággal tesszük, akkor nagy a tévedés kockázata.

Rangos folyóiratban megjelent publikáció szerint például konkáv költségfüggvényű, de egyszerű és éles szétválasztókat tartalmazó hálózat optimális megoldásában nem lehet recirkuláció (Floudas, C.A., Separation Synthesis of Multicomponent Feed Streams into Multicomponent Product Streams, AICHE J., 33, 540 (1987).). Egyszerű szétválasztó, aminek csak egy bemenet és két kimenete van. Éles egy szétválasztó, ha a bemeneti áramának komponensei közül mindegyik csak egy kimenetben szerepel. Az állításra bizonyítást is közöltek. Később kiderült, hogy nem csak a bizonyítás volt hibás, de az állítás is hamis, van rá ellenpélda [24].

Lineáris költségű szétválasztási hálózatoknál is ismert olyan publikáció elismert szerzőktől, ahol a berendezések megengedett kapcsolatai nem tartalmazták a gyakorlati feladat optimális struktúráját. A tévedés eredményeként az elérhető legjobbnál közel 30%-al rosszabb megoldást adtak a gyakorlati feladatra. Később erre a feladatosztályra átfogó, bizonyítottan helyes megoldó módszer született [23].

## 7.3. Ismeretlen információ szükséges a megoldáshoz

Ütemezési feladatok matematikai programozással történő megoldása gyakran véges számú időpontot feltételez, amikor a berendezések feladatot válthatnak. Ez a szám szükséges a modell felírásához, de a gyakorlati feladatból nem kézenfekvően következik. Ráadásul ezen paraméter növelésével a megoldandó feladat nehézsége exponenciálisan nő. Ha alul becsüljük a váltási pontok számát, akkor elveszítjük az optimális megoldást, ha pedig túl nagy számot választunk, akkor egy ipari méretű feladatra a számítási idő gyakorlatban kivárhatatlan lesz. Korábban azt feltételezték, hogy ha ezt a számot egyesével növelgetik, és egy újabb növelés nem hoz jobb megoldást, akkor elérték az optimumot, de ez nem igaz [17].

Szerencsére a jegyzetben ismertetett S-gráf módszertan és megoldó alkalmazásához ilyen paraméterekre nincs szükség. Ráadásul az S-gráf megoldó futási ideje is versenyképes az említett matematikai programozási modellre épülő optimalizálással.

## 7.4. Nem megvalósítható megoldás

Az előzőekben láthattuk, hogy ha indokolatlanul szűre szabjuk az optimalizálás lehetőségeit, akkor elvéthetjük a legjobb megoldást akár 30%-al. Ha azonban fontos feltételeket hagyunk ki a modellből, akkor előfordulhat, hogy a modell optimális megoldása nem megvalósítható. Ütemezés irodalmában erre is több példát találunk. Ilyen például a köztes tárolás nélküli ütemezésben, ha több berendezés egymásra vár, hogy a másikba áttölthesse anyagát.

Az S-gráfos megoldóval ez is elkerülhető [8]. Ez annak köszönhető, hogy az S-gráf megoldó a matematikai programozási modellel szimbiózisban gráfos leírást is használ, amin bizonyos típusú megvalósíthatósági feltételek könnyebben ellenőrizhetőek.

# Irodalomjegyzék


[1] R. Adonyi. *Szakaszos folyamatok ütemezése az S-gráf módszertan kiterjesztésével*. PhD thesis, University of Pannonia, 2008.

[2] E. Bajalinov B. Imreh. *Operációkutatás*. Szegedi Tudományegyetem Bolyai Intézet, 2005.

[3] G. Feng and LT Fan. On stream splitting in separation system sequencing. *Industrial & engineering chemistry research*, 35(6):1951–1958, 1996.

[4] F. Friedler, K. Tarjan, YW Huang, and LT Fan. Combinatorial algorithms for process synthesis. *Computers & chemical engineering*, 16:S313–S320, 1992.

[5] F. Friedler, K. Tarjan, YW Huang, and LT Fan. Graph-theoretic approach to process synthesis: polynomial algorithm for maximal structure generation. *Computers & chemical engineering*, 17(9):929–942, 1993.

[6] F. Friedler, JB Varga, Feher E., and LT Fan. Combinatorially accelerated branch-and-bound method for solving the mip model of process network synthesis. In Floudas CA and Pardalos PM, editors, *State of the Art in Global Optimization*, pages 609–626. Kluwer Academic Publishers, 1996.

[7] F. Friedler, JB Varga, and LT Fan. Decision-mapping: a tool for consistent and complete decisions in process synthesis. *Chemical Engineering Science*, 50(11):1755–1768, 1995.

[8] M. Hegyhati and F. Friedler. Overview of industrial batch process scheduling. *Chemical Engineering Transactions*, 21:895–900, 2010.

[9] T. Holczinger. *Módszer köztes tárolókat nem tartalmazó szakaszos működésű rendszerek ütemezésére*. PhD thesis, University of Pannonia, 2004.

[10] T. Holczinger, J. Romero, F. Friedler, and L. Piuigjaner. Scheduling of multipurpose batch processes with multiple batches of the products. *Hungarian Journal of Industrial Chemistry*, 30:305–312, 2002.

[11] D. Jungnickel. *Graphs, networks and algorithms*. Algorithms and computation in mathematics. Springer, 2004.

[12] J. Klemes, F. Friedler, I. Bulatov, and P. Varbanov. *Sustainability in the Process Industry : Integration and Optimization*. Green Manufacturing & Systems Engineering. McGraw-Hill, 2010.

[13] D.G. Luenberger and Y. Ye. *Linear and nonlinear programming*. International series in operations research & management science. Springer, 3rd edition, 2008.

[14] T. Majozi and F. Friedler. Maximization of throughput in a multipurpose batch plant under a fixed time horizon : S-graph approach. *Industrial Engineering Chemisty*, 45 :6713–6720, 2006.

[15] I. Maros. *Computational techniques of the simplex method*. International series in operations research & management science. Kluwer Academic Publishers, 2003.

[16] N. Nishida, G. Stephanopoulos, and A.W. Westerberg. A review of process synthesis. *AIChE Journal*, 27(3) :321–351, 1981.

[17] A. P. F. D. Barbosa-Póvoa P. Castro and H. Matos. An improved rtn continuous-time formulation for the short-term scheduling of multipurpose batch plants. *Industrial & Engineering Chemistry Research*, 40 :2059–2068, 2001.

[18] E. Sanmartí, F. Friedler, and L. Puigjaner. Combinatorial technique for short term scheduling of multipurpose batch plants based on schedule-graph representation. *Computers & Chemical Engineering*, 22 :847–850, 1998.

[19] E. Sanmartí, L. Puigjaner, T. Holczinger, and F. Friedler. Combinatorial framework for effective scheduling of multipurpose batch plants. *AIChE Journal*, 48(11) :2557–2570, 2002.

[20] J.J. Siirola. Industrial applications of chemical process synthesis. *Advances in Chemical Engineering*, 23 :1–62, 1996.

[21] I. Szalkai. *Diszkrét matematika és algoritmuselmélet alapjai*. Veszprémi Egyetemi Kiadó, 2001.

[22] Charles E. Leiserson Clifford Stein Thomas H. Cormen, Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 2009.

[23] F. Friedler Z. Ercsey and L. T. Fan. Separation-network synthesis : Global optimum through rigorous super-structure. *Computers and Chemical Engineering*, 24 :1881–1900, 2000.

[24] F. Friedler Z. Kovacs and L. T. Fan. Recycling in a separation process structure. *AICHE Journal*, 39 :1087–1089, 1993.