Wiener Gábor

KÖZELÍTŐ ALGORITMUSOK

BME SZIT

Készült a

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Számítástudományi és Információelméleti Tanszék

gondozásában.

Lektorálta: Csima Judit

Tartalomjegyzék

Előszó

Ez a jegyzet a BME mérnök-informatikus MSc képzésének Rendszeroptimalizálás és a BME alkalmazott matematikus MSc képzésének Kombinatorikus optimalizálás című tárgyaihoz készült. Az említett tárgyak anyagának ugyanakkor természetesen csak a közelítő algoritmusokkal (és az ehhez szorosan kapcsolódó ütemezéselméleti alapokkal) kapcsolatos részét fedi le, a tárgyak többi anyagrészéhez Jordán-Recski-Szeszlér: Rendszeroptimalizálás című tankönyvét (Typotex, Budapest, 2004.) ajánljuk. Ez a tankönyv is tartalmaz „Közelítő algoritmusok" és „Ütemezési algoritmusok" című fejezeteket, ezek anyagától azonban a Rendszeroptimalizálás/Kombinatorikus optimalizálás előadások jelentősen eltérnek, éppen ez tette szükségessé a jegyzet megírását.

A jegyzet felépítése a következő. Az első fejezetben a szükséges algoritmuselméleti alapokat tekintjük át (ennek döntő többsége persze ismétlés). A második fejezetben megvizsgáljuk, milyen lehetőségeink vannak, ha NP-nehéz problémával kerülünk szembe. Az additív hibával közelítő algoritmusokról szól a harmadik, a multiplikatív hibával közelítő algoritmusokról pedig a negyedik fejezet, az ötödik fejezetben foglalkozunk a polinomiális approximációs sémákkal. Végül a hatodik fejezet az ütemezéselmélet alapjaiba ad (nagyon) rövid bevezetést. A részletesebben tárgyalt problémák leírását a jobb érthetőség kedvéért az alábbiak szerinr tagoljuk. A *Probléma* részben ismertetjük a feladatot és esetlegesen annak hátterét, alkalmazási lehetőségeit. A *Bonyolultság* részben foglalkozunk a probléma NP-nehézségével, közelíthetőségével. Az *Algoritmus* részben adunk meg egy vagy több közelítő algoritmust, ezek polinomialitásával, illetve a közelítés pontosságával az *Elemzés* részben foglalkozunk. Esetenként szerepel egy *Éles példa* rész is, itt azt mutatjuk meg, hogy az adott algoritmus közelítése a megadottnál nem jobb. Az algoritmusok leírásakor nem adunk meg pszeudokódokat és nem foglalkozunk adatstruktúrákkal sem, az elemzésekben pedig általában nem szerepelnek pontos lépésszámok, az algoritmusok polinomialitását azonban természetesen tárgyaljuk.

Az ütemezéselméleti fejezet kivételével minden fejezet végén szerepel egy, az adott fejezet anyagához kapcsolódó feladatokból összeállított rövidebb-hosszabb gyűjtemény (ütemezéselméleti feladatok a zárthelyiben sem szerepelnek). Ezekben helyet kapnak egyszerű gyakorló példák (mint például egy algoritmus futta-

tása adott bemeneten vagy egy definíció megértését ellenőrző kérdések), egy algoritmus vagy bizonyítás mélyebb megértését segítő feladatok és időnként olyan feladatok is, melyek ennél lazábban kötődnek az anyaghoz. A feladatok egy része a korábbi félévek során szerepelt a zárthelyik valamelyikében. Minden feladatnál zárójelben közöljük a nehézségét (a szerző szerint), ez lehet könnyű (1), közepes (2) vagy nehéz (3). A feladatokhoz (ahol van értelme) szerepel a jegyzetben egy egy mondatos segítség. Ezt akkor érdemes igénybe venni, ha pár perc gondolkodás után sincs ötletünk, hogy hogyan induljunk el. A feladatok döntő többségéhez megoldást is közlünk, az anyag megértéséhez azonban célszerű a feladatokat önállóan, esetleg csoportmunkában megoldani vagy legalább megkísérelni megoldani. Ez olyasmit jelent, hogy amíg nem töltöttünk el 10-15 percet a feladaton való érdemi gondolkodással, addig nem érdemes megnézni a megoldást. Ez az idő akkor sem vész kárba, ha úgy tűnik, hogy semmire sem jutottunk, a megoldás megértését a gondolkodással töltött idő nagyban segíti.

A jegyzet végén szerepel egy függelék, melybe azon korábban tanult (szinte kizárólag gráfelméleti) fogalmak és tételek leírását gyűjtöttük össze, melyek az anyag megértéséhez szükségesek, de a tapasztalat szerint a hallgatók egy része nem mindegyikkel van tisztában. Ez a függelék a jegyzet folyamatos feldolgozását van hivatva segíteni (vagyis hogy ne kelljen korábbi jegyzetekben vagy az interneten keresgélni egy-egy tétel vagy definíció után); a korábbi tananyag esetleg szükséges átismétlésére nem alkalmas, e célra Katona-Recski-Szabó: A számítástudomány alapjai című tankönyvét, illetve Friedl-Recski-Simonyi: Gráfelméleti feladatok című feladatgyűjteményét javasoljuk.

Végül szeretném megköszönni a jegyzet megírásához nyújtott segítséget Csima Judit és Szeszlér Dávid tanszéki kollégáimnak.

1. fejezet

Algoritmuselméleti alapok

Ebben a fejezetben a későbbiek megértéséhez szükséges algoritmuselméleti alapokat tekintjük át röviden. Ezek többségével a korábbi tanulmányok során már foglalkoztunk, többnyire az itt leírtnál jóval részletesebben.

1.1. Problémák és algoritmusok

A számítógéppel megoldandó problémák legáltalánosabb osztályát alkotják a *kiszámítási problémák*: egy I bemenet valamely $f(I)$ függvényét szeretnénk kimenetként megadni. Lehet például a bemenet az (a,b) számpár, a kimenet pedig $a+b$, de akár a^b vagy az $(a-b,ab,a)$ számhármas, és gyakorlatilag bármi más is.
A kiszámítási problémáknak két fontos speciális esetével fogunk foglalkozni: az *eldöntési problémákkal* és az *optimalizálási problémákkal*. Eldöntési problémáról akkor beszélünk, ha $f(I)$ értéke csak IGEN vagy NEM lehet. Ilyen például egy szám paritásának meghatározása (a bemenet egy egész szám, a kimenet IGEN, ha a szám páros, NEM, ha páratlan) vagy annak eldöntése, hogy egy gráfban van-e teljes párosítás. Az optimalizálási problémák definiálása már valamivel macerásabb. Ilyenkor az I bemenethez egy X_I halmaz tartozik, ami az összes lehetséges kimenetet tartalmazza. Az X_I halmaz x elemeihez egy $c(x)$ valós függvényértéket rendelünk, amit optimalizálni (tipikusan maximalizálni vagy minimalizálni) szeretnénk. A kimenet egy olyan $x^* \in X_I$, melyre a $c(x^*)$ érték (az összes $c(x), x \in X_I$ értékek között) optimális. Elképzelhető például, hogy I egy gráf, X_I a gráf köreiből áll, az $x \in X_I$ körhöz pedig a hosszát rendeljük $c(x)$ függvényértékként. A kimenet (mondjuk) egy olyan $x^* \in X_I$ kell, hogy legyen, melyre $c(x^*)$ maximális. Ezzel egy gráf leghosszabb körének megkeresését definiáltuk, mint optimalizálási problémát. Fontos, hogy nem csak az optimális $c(x^*)$ értéket (jelen esetben a leghosszabb kör hosszát) kell megkapnunk kimenetként, hanem magát x^*-ot (vagy-

is jelen esetben egy leghosszabb kört). Az is lehetséges, hogy I egy élsúlyozott, összefüggő gráf, az X_I halmaz I feszítőfáiból áll, $c(x)$ pedig az $x \in X_I$ fa éleinek összsúlya. Ha a kimenet olyan $x^* \in X_I$, melyre $c(x^*)$ minimális, akkor a jól ismert minimális feszítőfa problémát kapjuk.

Problémák megoldhatósága

Már alapvető halmazelméleti ismeretek birtokában be lehet látni, hogy vannak olyan kiszámítási, sőt eldöntési problémák, melyeket nem lehet számítógéppel megoldani (némiképp leegyszerűsítve: a valamilyen programnyelven írt kódok halmaza megszámlálhatóan végtelen, az eldöntési problémák halmaza viszont kontinuum számosságú). Ilyen problémát meg is lehet adni: például a megállási probléma (annak eldöntése, hogy egy adott program egy adott bemeneten megáll-e) számítógéppel nem eldönthető. Azok a problémák, amikkel ebben a jegyzetben foglalkozni fogunk, nem ilyenek, mindegyikre létezik olyan algoritmus, ami a bemenetből véges sok lépésben kiszámítja a kívánt kimenetet. Ez azonban önmagában még nem jelenti azt, hogy ezek az algoritmusok kellően hatékonyak is; előfordulhat, hogy az említett véges sok lépés annyi időt igényel, hogy még az elképzelhető leggyorsabb számítógép is csak évmilliárdok alatt készülne el vele. Fontos ezért, hogy meg tudjuk becsülni egy-egy algoritmus várható futási idejét.

Algoritmusok lépésszáma

Az algoritmusok hatékonyságának vizsgálatánál több szempontot is figyelembe kell vennünk. A futásidő még egy konkrét implementáció esetén is függeni fog az algoritmust futtató számítógéptől és persze a bemenettől is (egy nagy bemenetet például már végigolvasni is sokáig tart). Ráadásul különböző szituációkban különböző futásidőket tartunk gyorsnak, illetve lassúnak. Az interneten böngészve egy kattintás után néhány másodperc várakozást is soknak érzünk, miközben elképzelhető, hogy egy komolyabb tervezési feladat esetében szívesen várunk több hetet is az optimális eredményre (a Galaxis Útikalauz Stopposoknak című könyvben Bölcs Elme híres válaszára hét és fél millió évet várnak).

A hardvertől való függés kérdését kezelhetjük azzal, hogy nem futásidőről, hanem lépésszámról beszélünk, ekkor persze meg kell állapodnunk abban, hogy mit értünk egy lépés alatt. A lépéseket célszerű a feladat jellegétől függően definiálni (más lépések lesznek egy számokkal kapcsolatos problémánál, mint egy gráfelméletinél), ügyelve ugyanakkor arra, hogy a különféle lépések tényleges időigénye közt ne legyen túl nagy különbség. Ezt azzal érjük el, hogy a lépéseket ún. elemi lépésekből építjük fel, mégpedig úgy, hogy túl sok elemi lépést nem engedünk meg egy lépésen belül (hogy mi számít túl soknak, arra később visszatérünk). Elemi lépésnek tekinthetjük például egy bit valamely memóriarekeszbe való beírását

vagy kiolvasását, de olyan műveleteket is, melyek konstans sok bitenkénti művelet segítségével elvégezhetők, mint például két egyjegyű szám összeadása, összeszorzása, összehasonlítása, adott konstansnál nem nagyobb értékek memóriába írása vagy kiolvasása. Az ezekből felépülő lépések például: számokkal kapcsolatos problémáknál két szám összeadása, szorzása, összehasonlítása, gráfelméleti problémáknál egy szomszédsági mátrix vagy éllista egy elemének kiolvasása, átírása, halmazokkal kapcsolatos problémáknál unió, metszet, különbség képzése.

A bemenettől való függést úgy kezeljük, hogy az algoritmusok lépészámát nem egyszerűen számként, hanem a bemenet méretének függvényében vizsgáljuk. Az, hogy egy algoritmus 500 lépést tett, semmit nem árul el róla és az is nagyon keveset, hogy ha annyit tudunk, hogy egy 10 méretű bemeneten tett 500 lépést, hiszen lehet, hogy egy 11 méretű bemeneten 10000 lépésre lesz szüksége, de az is, hogy csak 501-re, más méretekről nem is beszélve. Ha azonban tudjuk, hogy a lépésszám n méretű bemeneten legfeljebb $5n^2$, akkor képet kaptunk az algoritmus gyorsaságáról. Itt érdemes megjegyezni, hogy lépésszám alatt mindig legrosszabb eseti lépésszámot értünk, vagyis az algoritmusnak a lehetséges n hosszú bemenetekhez tartozó lépésszámai közül a legnagyobbat (szokás még időnként átlagos lépésszámmal is foglalkozni, de erről most nem lesz szó).

A bemenet mérete a leírásához szükséges bitek száma. Itt természetesen ésszerű kódolásokat akarunk figyelembe venni, vagyis egy a pozitív egész számot például nem a darab 1-es bit egymás mögé írásával, hanem egy $\lfloor \log a \rfloor + 1$ hosszú kettes számrendszerbeli számként kódolunk (itt, és a továbbiakban is a nem jelzett alapú logaritmus kettes alapú logaritmust jelöl). Gráfokat kódolhatunk (mondjuk) a szomszédsági mátrixukkal, halmazokat a karakterisztikus vektorukkal (egy $H \subseteq \{0,1,\ldots,n\}$ halmaz karakterisztikus vektora egy olyan n elemű 0-1 sorozat, melynek i. eleme pontosan akkor 1, ha $i \in H$). Fontos, hogy a „bemenet mérete” elnevezés a lehetséges bemenetekre vonatkozik, vagyis értelmes az, hogy a bemenet egy n jegyű szám, de értelmetlen azt mondani, hogy most épp a bemenet az 1, és annak mérete 1, hiszen 1 biten tudom ábrázolni (valójában bármely konkrét szám ábrázolható 1 biten, sőt tulajdonképpen 0 biten is).

A bemenet méretével és az algoritmusok lépésszámával kapcsolatos meggondolások sokkal precízebbé tehetők, ha az algoritmusokat a mostaninál formálisabban definiáljuk (szigorúan véve magukat az algoritmusokat nem is definiáltuk), például Turing-gépek vagy közvetlen elérésű gépek (RAM-ok) segítségével, ez azonban a jegyzet kereteit meghaladja.

1.2. Polinomialitás és NP-nehézség

A lépésszámuk alapján bizonyos algoritmusokat megfelelelőnek (kellően gyorsnak), másokat lassúnak tartunk. Célszerű a kettő között éles határvonalat húzni, hogy minden egyes algoritmusról eldönthessük, hogy melyik csoportba tartozik, ez azonban nem is olyan egyszerű. Minden igényt kielégítő definíciót nem fogunk tudni adni, a következő megközelítés azonban elméleti és gyakorlati szempontból is elég jól működik.

1.1. Definíció *Egy $f : \mathbb{N} \to \mathbb{N}$ függvényt polinomiálisnak nevezünk, ha léteznek olyan $c_1, c_2 \in \mathbb{R}^+$ konstansok, melyekre bármely $n \in \mathbb{N}$ esetén teljesül, hogy $f(n) \leq c_1 n^{c_2}$.*

Fontos, hogy c_1 és c_2 nem függhetnek n-től. Egy függvény tehát akkor polinomiális, ha létezik olyan polinom (valójában monom, vagyis egy tagú polinom), ami minden n-re felülről becsüli.

1.2. Definíció *Egy algoritmus akkor polinomiális, ha minden n méretű bemenetre legfeljebb $f(n)$ lépésben kiszámítja a kimenetet, ahol f polinomiális függvény. Egy problémát pedig akkor nevezünk polinomiálisnak (vagy precízebben polinom időben megoldhatónak), ha létezik a megoldására polinomiális algoritmus.*

Említettük korábban, hogy a lépések elemi lépésekből épülnek fel, az 1.2. Definíció alapján pedig az is világos, hogy ha polinom sok elemi lépést engedünk meg egy lépésen belül, akkor az algoritmusok polinomialitását nem befolyásolja, hogy elemi vagy összetettebb lépésekkel dolgozunk.
A polinomiális algoritmusokat tartjuk gyorsnak, a polinomiális problémákat pedig jól kezelhetőknek. Bár egy probléma polinomialitásának igazolása nem feltétlenül egyszerű feladat, a definíció alapján világos, hogy hogyan érdemes nekiállni: adunk rá egy polinomiális algoritmust, ami megoldja. Annak bizonyítása azonban, hogy egy probléma *nem* polinomiális, sokkal kellemetlenebb lehet, hiszen ehhez azt kell belátnunk, hogy a problémát megoldó algoritmusok *egyike sem* polinomiális. Könnyű például belátni, hogy két egész szám összeadása vagy összeszorzása megoldható polinom időben, mint ahogy láttunk polinomiális algoritmusokat számos gráfelméleti problémára is korábban. Ilyen például egy gráf összefüggőségének eldöntése, maximális párosítás megadása, maximális folyam keresése. Olyan problémával azonban, amiről beláttuk, hogy csak nem polinomiális algoritmusok vannak rá, ritkán találkoztunk. Ilyen volt az egész számok hatványozása, vagyis az (a,b) bemenetből a^b kiszámítása. a^b számjegyeinek száma (a kettes számrendszerben) $\lfloor \log a^b \rfloor + 1 = \lfloor b \log a \rfloor + 1$, ez pedig a bemenet méretében (ami $\lfloor \log a \rfloor + 1 + \lfloor \log b \rfloor + 1$) exponenciális (hacsak nem tudjuk valahonnan, hogy b sokkal kisebb, mint a). a^b kiszámítása tehát csakugyan nem valósítható

meg polinom időben, hiszen már a számjegyeinek kiírása is exponenciálisan sok lépést igényel. Efféle bizonyítékot általában nem könnyű találni; annak igazolására, hogy egy probléma nehéz, tipikusan más módszereket fogunk használni.

Bonyolultsági osztályok

1.3. Definíció *A polinom időben megoldható eldöntési problémák osztályát P-vel jelöljük.*

Számos olyan eldöntési probléma van, melyről nem világos, hogy P-be tartozik-e, de ha a probléma egy inputjára a válasz IGEN, akkor lehetséges olyan polinomiális méretű bizonyítékot mutatni, aminek helyességét polinom időben le lehet ellenőrizni. Nem ismert például, hogy létezik-e polinomiális algoritmus annak eldöntésére, hogy egy gráfnak van-e Hamilton-köre, ha azonban egy adott gráfról valaki azt állítja, hogy van Hamilton-köre és bizonyíték gyanánt meg is mutat egyet, akkor könnyen eldönthetjük, hogy a bizonyíték helyes-e, azaz hogy valóban Hamilton-körről van-e szó.

1.4. Definíció *Azon eldöntési problémák osztályát, melyekre létezik polinom méretű, polinom időben ellenőrizhető tanú az* IGEN *válaszra, NP-vel jelöljük.*

Az NP jelölés a nem determinisztikus polinomiális (angol nyelvű) elnevezés rövidítése, az NP osztály egy ekvivalens (és az itt megadottnál precízebb) definíciója ugyanis a nem determinisztikus Turing-gépekhez kötődik. Azon eldöntési problémák osztályát, melyeknél ugyanilyen tanú a NEM válaszhoz adható, co-NP-vel jelöljük. Világos, hogy a P-beli problémák NP-ben és co-NP-ben is benne vannak, hiszen akár az IGEN, akár a NEM válaszhoz alkalmas az üres sorozat tanúnak, a válasz helyességéről a problémára adható polinomiális algoritmus lefuttatásával győződhetünk meg. Ez alapján $\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$, nem ismert ugyanakkor, hogy a tartalmazás valódi-e. Számos problémára igaz, hogy az NP-belisége könnyen látható, a co-NP-belisége viszont valamilyen fontos, de egyáltalán nem triviális tétel következménye (pl. teljes párosítás létezése páros gráfban – Kőnig-tétel, Hall-tétel, síkbarajzolhatóság – Kuratowski-tétel, teljes párosítás létezése tetszőleges gráfban – Tutte-tétel).

NP-nehéz problémák

Számos olyan probléma van NP-ben, melyre nem ismerünk polinomiális algoritmust, ezek jelentős részéről azt gondoljuk, hogy nincsenek is P-ben. Ezt bizonyítani jelenleg nem tudjuk, de azt be tudjuk látni, hogy valamilyen értelemben nehezek.

1.5. Definíció *Egy A probléma polinom időben visszavezethető egy B problémára, ha A-t meg tudjuk oldani polinom időben úgy, hogy egy, a B problémát megoldó algoritmust szubrutinként meghívhatunk (és a szubrutin meghívása egy lépésnek számít).*

1.6. Definíció *Egy NP-beli problémát NP-teljesnek nevezünk, ha minden NP-beli probléma polinom időben visszavezethető rá.*

A definíció alapján tehát egy *A* NP-teljes probléma legalább olyan nehéz, mint bármely NP-beli probléma, abban az értelemben, hogy ha *A* megoldható polinom időben, akkor minden NP-beli probléma is megoldható polinom időben. NP-teljes problémák létezése a legkevésbé sem magától értetődő, az első ilyet Cook és Levin találta egymástól függetlenül 1971-ben (Levin formálisan csak valamivel később írta le az eredményeit), ez a Boole-formulák kielégíthetőségének problémája (SAT) volt. Azt, hogy az NP-teljes problémák nem csak ilyen elvont (-nak tűnő) környezetben, hanem a számítástudomány legkülönbözőbb, gyakorlati szempontból fontos területein is felbukkannak, Karp mutatta meg 1972-ben, amikor is 21 problémáról bizonyította be, hogy szintén NP-teljesek. A bizonyítás során persze felhasználta Cook eredményét, a problémákra visszavezette a SAT-ot.
Azóta rengeteg problémáról derült ki, hogy NP-teljes; ezen problémák bármelyikének polinomiális idejű megoldása azt jelentené, hogy P=NP. Ez azonban nem valószínű: NP-teljes problémák százait próbálták és próbálják hatékonyan megoldani kutatók ezrei, polinomiális algoritmust azonban senki nem talált egyikre sem, holott ha csak egyre is lenne ilyen, akkor ez természetesen az összes többire is teljesülne. Ez azt sejteti, hogy P$\neq$NP, azonban ezen sejtés bizonyítása felé sem sikerült egyelőre igazán komoly lépést tenni.
Az optimalizálási problémák nem lehetnek NP-teljesek, hiszen nem lehetnek NP-beliek sem (mivel nem eldöntési problémák), ezt a fajta nehézség-fogalmat azonban rájuk is be tudjuk vezetni.

1.7. Definíció *Egy problémát NP-nehéznek nevezünk, ha minden NP-beli probléma polinom időben visszavezethető rá.*

Az NP-nehézséghez tehát nem szükséges, hogy egy probléma eldöntési legyen, bármilyen kiszámítási probléma szóba jöhet, így persze optimalizálási problémák is. A definíciók alapján egyértelmű, hogy az NP-teljes problémák halmaza az NP-nehéz problémák és az NP-beli problémák halmazainak metszete. Világos, hogy az NP-nehéz problémákra is igaz, hogy legalább olyan nehezek, mint bármely NP-beli probléma (köztük az NP-teljes problémák), így ezekre sem várható polinomiális algoritmus. Problémák NP-nehézségét tipikusan a Karp által 1972-ben bevezetett módon szokás belátni, vagyis úgy, hogy polinom időben visszavezetünk rájuk egy már tudottan NP-nehéz problémát. Ez a módszer a polinomiális

idejű visszavezetés tranzitivitása miatt működik (azaz ha A visszavezethető B-re és B C-re, akkor A visszavezethető C-re is).

Érdemes megjegyezni, hogy az általunk használt visszavezetés-fogalom nem pontosan ugyanaz, mint amit Karp használt és amivel a korábbi tanulmányok során már találkoztunk, tehát ebben a jegyzetben nem a Karp-redukciót nevezzük visszavezetésnek. Az 1.5. Definícióban tárgyalt visszavezetés neve Cook-redukció. Minden Karp-redukció egy speciális Cook-redukció, a Cook-redukcióval definiált NP-nehéz osztály tehát tartalmazza a Karp-redukció szerint NP-nehéz problémákat, nem ismert viszont, hogy a tartalmazás valódi-e. A Cook-redukciók használatának a Karp-redukciók helyett kényelmi okai vannak: a visszavezetések könnyebbek (vagy könnyebben leírhatók) lesznek, miközben az NP-nehéznek bizonyuló problémák nehézsége (vagyis hogy legalább olyan nehezek, mint bármely NP-beli probléma) így is fennáll.

1.3. Feladatok

1. **(1)** Polinomiálisak-e az alábbi függvények?

(a) $\frac{n^3}{3}+\frac{n^2}{2}$ (b) $n\log n+3\sqrt{n}$ (c) $3^{\log n}$ (d) $\sqrt{n}^{\log n}$ (e) $\log n^{n^2}$ (f) $\log n^{n^n}$

2. **(1)** Tekintsük a következő algoritmust az a és b pozitív egészek összeszorzására: az a számhoz hozzáadjuk a-t, összesen $(b-1)$-szer. Polinomiális-e ez az algoritmus?

3. **(1)** Egy probléma bemenete az x,y pozitív egész számokból áll. Polinomiálisak-e az alábbi lépésszámú algoritmusok a problémára?

(a) $\log x^{\log xy}$ (b) $\log\log x^y$ (c) $(\log x)^{\log\log|x-y|}$

4. **(1)** Egy probléma bemenete az $a_1\le a_2\le\ldots\le a_n$ pozitív egész számokból áll. Polinomiálisak-e az alábbi lépésszámú algoritmusok a problémára?

(a) n (b) a_1 (c) $\log a_1a_2\ldots a_n$ (d) $(\log a_1)^{\log a_1}$ (e) $(\log a_1)^n$

2. fejezet

NP-nehéz problémák kezelése

NP-nehéz optimalizálási problémák pontos megoldására az eddig látottak szerint nem érdemes polinom idejű algoritmust keresni. Ez azonban nem jelentheti azt, hogy ezen problémák megoldásáról teljesen lemondunk, hiszen ilyenek számos gyakorlati alkalmazás során előkerülnek. Az egyik lehetőség, hogy olyan exponenciális lépésszámú algoritmust próbálunk adni, amely nem túl nagy bemeneteken még elfogadható idő alatt lefut. A lineáris programozás feladatát megoldó, exponenciális futásidejű szimplex-módszer például ma is használatos, annak ellenére, hogy jó ideje ismertek polinomiális algoritmusok is a feladatra. Előfordulhat az is, hogy egy, a gyakorlatban felmerülő NP-nehéz problémáról kiderül, hogy valójában csak egy speciális esetét kell kezelnünk, ami már elképzelhető, hogy megoldható polinom időben. Erre a 2.1 alfejezetben látunk majd példákat. Bevett módszer az is, hogy a problémák pontos (de lassú) megoldása helyett közelítő megoldást próbálunk adni, ami polinomiális időben fut. Ha ilyenkor sikerül bebizonyítanunk, hogy a kapott megoldás valóban jól közelíti az optimumot és hogy a futásidő polinomiális, akkor beszélünk *közelítő algoritmusról* – ezekkel foglalkozunk a hátralévő fejezetekben. *Heurisztikának* nevezzük azokat az eljárásokat, amelyekről nem tudjuk belátni, hogy jó közelítést adnak vagy hogy polinom idejűek, de a tapasztalat szerint időben lefutnak és megfelelő eredményt adnak (szokás heurisztikának nevezni a bizonyítottan polinomiális, de nem garantált közelítésű algoritmusokat is és az is előfordul, hogy közelítő algoritmusokat heurisztika gyanánt használnak – ilyenkor a közelítésre van bizonyított becslés, de a tapasztalati eredmények ennél általában jobbak). Heurisztikákkal ebben a jegyzetben nem foglalkozunk, mint ahogy az Algoritmusok elmélete tárgyban korábban már szereplő pszeudopolinomiális algoritmusokkal, illetve branch and bound módszerrel sem.

2.1. NP-nehéz problémák polinomiális speciális esetei

Ebben az alfejezetben néhány olyan NP-nehéz feladatot mutatunk be, melyeknek bizonyos (nem triviális) speciális esetei már megoldhatók polinom időben. Ez a gyakorlati alkalmazások szempontjából azért lehet érdekes, mert előfordulhat, hogy egy problémát sikerül (például) gráfelméleti nyelven megfogalmazni, de a kapott probléma szükségtelenül általános és ezért azt gondoljuk, hogy az eredeti feladat NP-nehéz volt, miközben ennek esetleg az ellenkezője igaz. Korábbi tanulmányaink során például már esett szó a leghosszabb út problémáról, s az is ismert, hogy ez NP-nehéz (hiszen speciális esetként tartalmazza a Hamilton-út problémát), ugyanakkor irányított aciklikus gráfban (DAG-ban) a leghosszabb irányított út keresésére már láttunk polinomiális algoritmust.

Erőforrások ütemezése

Adott munkáknak egy $\{M_1, M_2, \ldots, M_n\}$ halmaza, ezeket szeretnénk elvégezni a lehető legkevesebb erőforrás bevonásával (más optimalizálási szempontunk nincs). Bármelyik erőforrás el tudja végezni bármelyik munkát, munkák bizonyos párjai azonban nem végezhetők el ugyanazon erőforrás segítségével; ezeket a párokat egy gráffal adjuk meg, melynek csúcsai a munkák, élei pedig a kérdéses párok. Azon kívül, hogy a gráfban összekötött csúcsokhoz különböző erőforrásokat kell rendelnünk, más megkötés nincs. Könnyen látható, hogy a feladat nem más, mint a gráf csúcsainak szokásos színezése a lehető legkevesebb színnel (az azonos színű csúcsokhoz tudjuk ugyanazt az erőforrást rendelni). Ennek az eldöntési változata (ahol a bemenet része egy k szám is, és eldöntendő, hogy $\chi(G) \leq k$ teljesül-e) egyike Karp 21 NP-teljes problémájának, tehát a feladat NP-nehéz (sőt, már azt is NP-teljes eldönteni, hogy egy gráf színezhető-e 3 színnel). Ha azonban tudjuk a munkák közti konfliktusok okát (vagyis hogy mi okozza azt, hogy két munka nem végezhető ugyanazon erőforrással), már elképzelhető, hogy találunk polinomiális idejű algoritmust. Ha például minden munkát két adott időpillanat között kell folyamatosan elvégezni és a konfliktusokat mindig az okozza, hogy a két munka időben részben fedi egymást (ez lehet a helyzet például, ha az erőforrások processzorok, amiknek egy nagyobb számítási feladat részfeladatait kell elvégezniük), akkor a gráf intervallumgráf lesz, aminek optimális színezésére könnyű polinomiális algoritmust adni. (Az intervallumok kezdőpontjai szerinti sorrendben vett mohó színezés (ld. a Függelékben) optimális lesz, hiszen csak a klikkszámmal megegyező számú színt használ.)

Maximális független ponthalmaz keresése

Ismert, hogy egy gráf maximális független ponthalmazának megtalálása (a továbbiakban MAX_FTL) NP-nehéz, hiszen a maximális klikk mérete azonos a gráf komplementerében vett maximális független halmaz méretével, a klikk-probléma eldöntési változata pedig szerepel Karp 21 NP-teljes problémája között. Megmutatjuk viszont, hogy páros gráfok esetében a javítóutas algoritmus segítségével polinom időben találunk egy maximális független ponthalmazt. Bármely páros gráf osztályai persze független halmazt alkotnak és csábító az a gondolat, hogy a kettő közül a nagyobbik maximális független halmaz is lesz, ez azonban nem feltétlenül teljesül (ld. 5. feladat).)

Tegyük fel, hogy az (A,B,E) páros gráfra lefuttattuk a javítóutas algoritmust (vagyis már nem létezik több javítóút). Jelöljük az A, illetve a B osztályban a kapott M párosítás által nem lefedett pontok halmazát A_1-gyel, illetve B_1-gyel, az A_1-ből alternáló úton elérhető B-beli csúcsok halmazát B_2-vel. Legyen továbbá A_2 a B_2-beli csúcsok M-beli párjainak halmaza, végül $A_3 := A \setminus A_1 \setminus A_2$, $B_3 := B \setminus B_1 \setminus B_2$. Ekkor az $F := A_1 \cup A_2 \cup B_1 \cup B_3$ halmaz a gráf egy maximális független ponthalmaza lesz. A halmaz függetlensége könnyen látható: A_1 és A_2, illetve B_1 és B_3 között a gráf páros volta miatt nem megy él, $A_1 \cup A_2$ és B_1 között pedig azért nem, mert ellenkező esetben még létezne javítóút (mivel B_2 pontjai elérhetők alternáló úton A_1-ből, ezért ugyanez igaz A_2 pontjaira is). Végül ha $A_1 \cup A_2$ egy csúcsa és valamely $b \in B_3$ között lenne él, akkor b elérhető lenne alternáló úton A_1-ből, ami B_3 definíciója szerint lehetetlen. Azt kell még belátnunk, hogy F-nél nagyobb független halmaz nincs a gráfban. Ha az F' független halmaz nagyobb méretű lenne, mint F, akkor az $A \cup B \setminus F'$ halmaz (ami lefogó ponthalmaz lenne), mérete kisebb volna, mint $|A \cup B \setminus F| = |A_3 \cup B_2|$, ami éppen az M párosítás éleinek száma, ez azonban lehetetlen, hiszen semelyik lefogó ponthalmaz sem lehet kisebb, mint egy tetszőleges párosítás mérete (mivel a párosítás élei a gráfban is szerepelnek és semelyik kettőt sem tudjuk ugyanazzal a csúccsal lefogni). A javítóutas algoritmussal tehát páros gráfban csakugyan meg tudjuk oldani polinom időben a MAX_FTL-t.

Élszínezés

Bár az élszínezés valamivel könnyebb, mint a csúcsszínezés (hiszen a G gráf élszínezése nem más, mint G élgráfjának csúcsszínezése, vagyis minden élszínezés egy speciális csúcsszínezéssel egyenértékű) és jóval többet is lehet tudni róla (pl. Vizing tétele szerint minden egyszerű gráf élszínezhető $\Delta(G)+1$ színnel, ahol $\Delta(G)$ a G-beli maximális fokszám), a gráfok optimális élszínezésének feladata is NP-nehéz (sőt, még azt is NP-teljes eldönteni, hogy egy G egyszerű gráf élkromatikus száma $\Delta(G)$ vagy $\Delta(G)+1$). Páros gráfok esetében azonban Kőnig egy

tétele szerint $\chi_e(G) = \Delta(G)$. Ebből önmagában még nem következik, hogy páros gráfokra polinom időben meg is találunk egy jó élszínezést, Kőnig bizonyítása azonban algoritmikus és az algoritmus polinom időben fut, ezt írjuk le az alábbiakban. Kőnig a tételt 1916-ban bizonyította, azóta számos gyorsabb (és tipikusan jóval bonyolultabb) algoritmus is született a problémára.

Legyenek a G páros gráf osztályai A és B, élei pedig $e_1, e_2, \ldots, e_m$. Az algoritmus egymás után foglalkozik G éleivel és a sorra kerülő élt mindig megszínezi a rendelkezésre álló $\Delta(G)$ szín valamelyikével (mégpedig helyesen, vagyis csatlakozó, már színezett élek különböző színt kapnak). Eközben előfordulhat, hogy a korábban már megszínezett élek egy részét át kell színezni más színűre, de semelyik él nem változik vissza színtelenné. Tegyük fel tehát, hogy az $e_1, e_2, \ldots, e_{i-1}$ élek már kaptak valamilyen színt és legyen $e_i = \{u, v\}$, ahol $u \in A$ és $v \in B$. Mivel u-ra és v-re is legfeljebb $\Delta(G)$ él illeszkedik, de ezek között még van színtelen (nevezetesen e_i), ezért mindkét csúcsnak van szabad színe – vagyis olyan szín, amilyen színű él még nem illeszkedik a csúcsra. Ha u-nak és v-nek van közös szabad színe, akkor persze e_i megkaphatja ezt a színt és készen is vagyunk.
Tegyük fel most, hogy nem ez a helyzet. Legyen u egy szabad színe a piros, v egy szabad színe a kék. Legyen C a G-nek az a részgráfja, amely (G összes csúcsából és) az aktuális színezés szerint pirosra vagy kékre színezett élekből áll. Ekkor C-ben minden pont foka legfeljebb 2, hiszen minden csúcsra legfeljebb egy piros és egy kék él illeszkedhet. Így C diszjunkt utakból és körökből áll. Ráadásul u és v foka C-ben csak 1 lehet, hiszen u-nak a piros, v-nek a kék szabad színe. Vagyis u és v is egy-egy C-beli út végpontja.
Legyen P_u az a C-beli út, amelynek egyik végpontja u. Állítjuk, hogy v nincs rajta P_u-n. Ha ugyanis rajta volna, akkor nyilván v volna a P_u másik végpontja. Ekkor P_u páratlan sok élből állna, hiszen a páros gráf egyik osztályából (A-ból) indul és a másikban (B-ben) ér véget. Mivel P_u-n nyilván váltakozva következnek kék és piros élek és az első (u-ra illeszkedő) éle kék (hiszen u-nak a piros szabad színe), ezért az utolsó éle is kék kell legyen (mert P_u páratlan sok élű). Így P_u másik végpontja nem lehet v, mert arra nem illeszkedik kék él.
Színezzük most át a P_u kék éleit pirosra, a piros éleit pedig kékre. Ezzel a színezést nyilván nem rontottuk el, viszont mostantól az u-nak a kék szabad színe lesz. Mivel a P_u élein zajló színcsere (a fentiek szerint) a v-re illeszkedő éleket nem érintette, ezért a kék v-nek is szabad színe maradt. Így az e_i él kékre színezésével továbbra is helyes színezést kapunk.

2.2. Feladatok

5. **(2)** Adjunk meg olyan összefüggő páros gráfot, melyben egyik osztály sem alkot maximális független halmazt.

6. (2) Minden $k \geq 2$-re adjunk meg olyan G gráfot, melyre $\chi(G) = k$ és G egyetlen k-színezésében sincs olyan színosztály, mely a gráf maximális független csúcshalmaza lenne.

7. (1) Egy gráfnak pontosan nyolc jó 2-színezése van. Határozzuk meg a komponenseinek számát.

8. (3) Egy n csúcsú gráfnak pontosan hat jó 3-színezése van. Mutassuk meg, hogy az éleinek száma legalább $2n-3$.

3. fejezet

Közelítés additív hibával

Ebben a fejezetben olyan közelítő algoritmusokkal foglalkozunk, melyeknél az optimális megoldástól való eltérés nem nagyobb egy előre rögzített fix konstansnál. Ilyen algoritmusokat NP-nehéz problémákra nem könnyű találni; jóval gyakoribb, hogy az eltérés függ (mondjuk) az optimum értékétől. Ennek megfelelően a fejezet nem hosszú: két speciális színezési problémára adunk ilyen algoritmust, majd mutatunk egy olyan problémát, melyről be tudjuk látni, hogy ez nem lehetséges.

3.1. Definíció *Egy optimalizálási problémát egy algoritmus C additív hibától eltekintve helyesen old meg, ha minden I bemenetre ad egy olyan $x^* \in X_I$ kimenetet, melyre a $c(x^*)$ érték legfeljebb C-vel tér el a $c(x), x \in X_I$ értékek optimumától (ahol $c(x)$ a problémában szereplő célfüggvény, X_I pedig az I bemenethez tartozó, szóba jövő kimenetek halmaza). Maximalizálási probléma esetén tehát*

$$c(x^*) \geq \max_{x \in X_I} c(x) - C,$$

minimalizálási probléma esetén pedig

$$c(x^*) \leq \min_{x \in X_I} c(x) + C$$

az elvárás.

Ugyanúgy, mint az optimalizálási problémák pontos megoldásánál, itt is nagyon fontos, hogy nem csak az optimális $c(x^*)$ értéket, hanem magát x^*-ot is meg kell kapnunk.

3.2. Definíció *Egy algoritmust C additív hibával közelítő algoritmusnak nevezünk, ha az adott optimalizálási problémát polinom időben, C additív hibától eltekintve helyesen oldja meg.*

3.1. Speciális színezési feladatok

Egyszerű gráfok élszínezése

Már esett róla szó, hogy egy egyszerű gráf optimális élszínezésének megtalálása NP-nehéz. Mivel bármely G esetén $\chi_e(G) \geq \Delta(G)$ és Vizing tétele szerint egyszerű gráfokra $\chi_e(G) \leq \Delta(G) + 1$, egyszerű gráfok élkromatikus számát legfeljebb 1 additív hibával könnyen meg tudjuk határozni. Ez azonban önmagában még nem jelenti azt, hogy a problémára lenne 1 additív hibával közelítő algoritmusunk, hiszen ahhoz polinom időben meg is kell adnunk egy olyan élszínezést, amely (amellett, hogy helyes) legfeljebb 1-gyel több színt használ, mint az optimális. Szerencsére a Vizing-tételnek van konstruktív bizonyítása, vagyis olyan eljárás, ami egy G egyszerű gráf éleit legfeljebb $\Delta(G) + 1$ színnel megszínezi. Mivel az említett eljárás ráadásul polinom idejű, az egyszerű gráfok élszínezésére csakugyan létezik 1 additív hibával közelítő algoritmus. (Magát az algoritmust itt nem részletezzük, leírása megtalálható például Katona-Recski-Szabó: A számítástudomány alapjai című tankönyvében.)

Síkgráfok színezése

A gráfok optimális színezésének még az a speciális esete is NP-nehéz, amikor a bemenetnek síkba rajzolható gráfnak kell lennie. Ismert, hogy a síkba rajzolható gráfok 4 színnel színezhetők (négyszín-tétel), ami azt is jelenti, hogy egy ilyen gráf kromatikus számát kis additív hibával meg tudjuk mondani. Ez azonban, csakúgy, mint az előző példában, önmagában nem elég az additív hibával közelítő algoritmushoz. Bár a négyszín-tételnek még az új (a korábbinál lényegesen rövidebb és egyszerűbb, Robertson, Sanders, Seymour és Thomas nevéhez fűződő) bizonyítása is igen hosszú és bonyolult, tartozik hozzá egy algoritmus, ami tetszőleges síkgráfot polinom időben megszínez 4 színnel, így a síkgráfok színezésére létezik 2 additív hibával közelítő algoritmus (a kromatikus szám nyilván legalább 1, sőt ha a gráfnak van éle, akkor legalább 2). A hiba 2-ről 1-re csökkenthető, ha a 4-színező algoritmus bevetése előtt (mondjuk) szélességi kereséssel eldöntjük, hogy a gráf páros-e. Ha a gráf páros, akkor a szélességi keresés segítségével meg is tudunk adni egy 2-színezést (ekkor ez lesz a kimenet), ha nem páros, akkor jöhet a 4-színező algoritmus, ami ekkor már legfeljebb 1-et téved (hiszen a kromatikus szám ilyenkor legalább 3). Mivel a szélességi keresés polinom időben megy, a teljes algoritmus is polinomiális lesz. A négyszín-tételhez tartozó algoritmus helyett használhatjuk az ötszín-tétel (polinomiális) konstruktív bizonyításainak valamelyikét is, vagy akár egy ennél is egyszerűbb, hat színnel színező algoritmust (ld. 11. feladat), ekkor az additív konstans persze rosszabb lesz.

3.2. A leghosszabb kör probléma

Egy gráf (egyik) leghosszabb körének megtalálása (a továbbiakban MAX_KÖR) nyilván NP-nehéz probléma, hiszen a leghosszabb kör ismeretében el tudjuk dönteni, hogy a gráfnak van-e Hamilton-köre (a Hamilton-kör probléma egyike Karp 21 NP-teljes feladatának). Most megmutatjuk, hogy ennél több is igaz: a problémára még additív hibával közelítő algoritmus sem adható (feltéve, hogy P≠NP). Tegyük fel ugyanis, hogy létezik valamely C konstans mellett C additív hibával közelítő algoritmus a MAX_KÖR-re és legyen G egy tetszőleges gráf, melyről szeretnénk eldönteni, hogy van-e Hamilton-köre. Készítsük el először G-ből a G' gráfot a következőképp: osszuk fel G minden élét C darab új ponttal, vagyis helyettesítsük G éleit $C+1$ élű utakkal, ahogy az az alábbi ábrán is látható (itt G a $K_{2,3}$ gráf, $C=2$).

Világos, hogy G egy tetszőleges, k élből álló körének megfelel G' egy $k(C+1)$ élű köre. Azt sem nehéz végiggondolni, hogy ez a megfeleltetés kölcsönösen egyértelmű, azaz hogy különböző G-beli körökhöz különböző G'-beliek tartoznak és hogy minden G'-beli kört megfeleltettünk egy G-belinek (mivel G' új csúcsai mind másodfokúak, a régi csúcsok pedig csak az újakkal vannak összekötve). Ha a feltételezett C additív hibával közelítő algoritmusunkat G'-re alkalmazzuk, akkor a kimenet G' egy olyan köre kell legyen, mely a leghosszabb kör hosszánál legfeljebb C éllel tartalmaz kevesebbet. Mivel a fentiek szerint G' minden körének hossza osztható $(C+1)$-gyel, az algoritmus kimeneteként kapott kör G' egy leghosszabb köre kell, hogy legyen (ellenkező esetben ezen kör hosszának és az optimális kör hosszának különbsége legalább $C+1$ lenne). Az ennek megfelelő G-beli kör így G egy leghosszabb köre lenne, ennek ismeretében pedig el tudnánk dönteni, hogy van-e G-ben Hamilton-kör. Mivel G'-t polinom időben elő tudjuk állítani G-ből és G' mérete konstansszorosa G méretének, a feltételezett algoritmus is polinom időben futna G méretében, amiből P=NP következne.

3.3. Feladatok

9. **(1)** Határozzuk meg azon G gráfokat, melyek minden élét $C \geq 1$ ponttal felosztva olyan gráfot kapunk, melynek van Hamilton-köre.

10. (2) Mutassunk olyan (nem egyszerű) gráfot, melynek élkromatikus száma nagyobb, mint $\Delta(G)+1$. (Vagyis igazoljuk, hogy a Vizing-tétel feltételei közül a gráf egyszerűsége nem elhagyható.)

11. (3) Adjuk meg egy tetszőleges G egyszerű síkgráf csúcsainak egy olyan sorrendjét, melyben a csúcsokat mohón színezve legfeljebb 6 színt kell használni.

4. fejezet

Közelítés multiplikatív hibával

Említettük, hogy NP-nehéz optimalizálási problémákra additív hibával közelítő algoritmusokat ritkán lehet találni. Annál gyakoribbak a multiplikatív hibával közelítő algoritmusok, ahol az optimum és a talált megoldás értékének hányadosát korlátozzuk. Kis hányados esetén ezek a közelítések gyakorlati problémákban is jól használhatók lehetnek (annak ellenére, hogy az optimumtól való eltérést ilyenkor tipikusan semmilyen konstanssal sem lehet felülről becsülni).

4.1. Definíció *Egy maximalizálási problémát egy algoritmus $k > 1$ multiplikatív hibától eltekintve helyesen old meg, ha minden I bemenetre ad egy olyan $x^* \in X_I$ kimenetet, melyre*

$$c(x^*) \geq \frac{1}{k} \max_{x \in X_I} c(x).$$

Minimalizálási probléma esetén az elvárás olyan $x^* \in X_I$ kimenet, melyre

$$c(x^*) \leq k \min_{x \in X_I} c(x).$$

$c(x)$ itt is a problémában szereplő célfüggvény, X_I pedig az I bemenethez tartozó, szóba jövő kimenetek halmaza.

Természetesen itt is rendkívül fontos, hogy nem csak az optimális $c(x^*)$ értéket, hanem magát x^*-ot is meg kell kapnunk.

4.2. Definíció *Egy algoritmust k-approximációs algoritmusnak nevezünk, ha az adott optimalizálási problémát polinom időben, k multiplikatív hibától eltekintve helyesen oldja meg.*

A k-approximációs algoritmusok tehát mindig polinomiálisak kell legyenek. Szokás k-approximációs algoritmus helyett egyszerűen k-approximációról beszélni, a k számot pedig approximációs faktornak nevezni. Az algoritmusok approximációs

faktorának elemzésekor egyszerűsíti a tárgyalást, ha az egyes problémákhoz tartozó optimum értékét egységesen OPT-tal jelöljük (függetlenül attól, hogy éppen milyen típusú problémáról van szó), ezért a továbbiakban használni fogjuk ezt a jelölést. Közelítő algoritmusok esetében *éles példának* nevezünk egy I bemenetet (vagy esetleg bemenetek egy sorozatát), ha az algoritmust I-n futtatva a közelítés nem jobb, mint amit (az összes lehetséges bemenetre) bizonyítani tudtunk. Az éles példák tehát azt mutatják, hogy ha az adott algoritmus közelítésénél jobbat szeretnénk, akkor nem az algoritmus elemzését kell finomítani, hanem másik algoritmusra van szükség.
A multiplikatív közelítéseket először pár egyszerű példán mutatjuk be, majd megvizsgálunk néhány bonyolultabb problémát, köztük olyanokat is, melyekre nem lehet k-approximációs algoritmust adni (feltéve, hogy P$\neq$NP).

4.1. Néhány egyszerű 2-approximáció

4.1.1. A maximális páros részgráf probléma

Probléma. A maximális páros részgráf probléma (a továbbiakban MAX_PÁROS) esetében egy adott $G = (V,E)$ gráf maximális élszámú páros részgráfját keressük (a csúcsok számára nyilván nincs értelme maximalizálni), vagyis a Vcsúcshalmaz egy olyan kettéosztását A és B halmazokra ($A \cup B = V, A \cap B = \emptyset$), melyre az A és B közt menő élek száma maximális.

Bonyolultság. A MAX_PÁROS NP-nehéz (ennek okait nem részletezzük), 2-approximációs algoritmust azonban könnyű rá mutatni.

2-approximációs algoritmus, első megoldás Osszuk ketté tetszőlegesen a csúcshalmazt és jelöljük minden v csúcsra $d_s(v)$-vel a v-ből a saját csoportjába menő élek számát, $d_m(v)$-vel pedig a v-ből a másik csoportba menő élek számát. Nyilván teljesül, hogy $d_s(v) + d_m(v) = d(v)$, ahol $d(v)$ a v csúcs foka G-ben. Egy olyan v csúcsot, melyre $d_m(v) < d_s(v)$, érdemes áthelyezni a másik csoportba, hiszen ezzel növeljük a két csoport közt menő élek számát (mivel $d_s(v)$ éllel növeljük és csak $d_m(v)$ éllel csökkentjük). Az ilyen v csúcsokat ennek megfelelően nevezzük áthelyezendőnek. Az algoritmus megvizsgálja, hogy van-e áthelyezendő csúcs és ha igen, akkor egy ilyet át is helyez a másik csoportba, majd ezt ismétli, amíg talál áthelyezendő csúcsot.

Elemzés. A fenti algoritmus nyilván megvalósítható polinom időben: bár egy csúcsot többször is áthelyezhetünk, az összes áthelyezések száma felülről becsülhető a gráf éleinek számával, hiszen a keresztbe menő élek száma minden áthelyezésnél nő.

Ha már nincs áthelyezendő csúcs, akkor minden v csúcsra teljesül, hogy $d_m(v) \geq d_s(v)$. Az adott kettéosztás mellett a két csoport közt keresztbe menő élek száma $\frac{1}{2}\sum_{v\in V} d_m(v)$, míg a csoportokon belül menő élek száma $\frac{1}{2}\sum_{v\in V} d_s(v)$, vagyis ha nincs áthelyezendő csúcs, akkor a keresztbe menő élek száma legalább akkora, mint a csoportokon belül menőké. Mivel csak ez a két éltípus van, a keresztbe menő élek száma legalább akkora lesz, mint a gráf összes élei számának fele, ami pedig nyilván legalább akkora, mint a maximális páros részgráf élszámának fele, így csakugyan 2-approximációs algoritmust kaptunk.

2-approximációs algoritmus, második megoldás Gyorsabb algoritmust kapunk, ha ahelyett, hogy V-t tetszőlegesen kettéosztanánk, a pontokat egyesével helyezzük el a kezdetben üres A és B halmazokban úgy, hogy a soron következő pontot mindig abba a halmazba rakjuk, amelyben kevesebb (nem több) szomszédja van a már elhelyezettek közül.

Elemzés. A már bekerült pontok által feszített részgráfba így minden pont bevételével legalább annyi keresztbe menő él kerül, mint amennyi csoporton belül megy, így végül itt is teljesülni fog, hogy az éleknek legalább a fele A és B között megy. Mivel ez az eljárás is nyilván polinomiális, most is 2-approximációt kaptunk. Érdemes megfigyelni, hogy a második algoritmus kimenetében a csúcsokra nem feltétlen teljesül $d_m(v) \geq d_s(v)$, így a kimenet esetleg javítható, ha még az első algoritmust is lefuttatjuk az aktuális kettéosztással kezdve. Ez ugyanakkor persze nem garantál jobb approximációs faktort (ld. 13., 14., 15., 16. feladatok). A legjobb ismert approximációs faktor a MAX_PÁROS-ra egyébként kb. 1,139.

4.1.2. A minimális lefogó ponthalmaz probléma

Probléma. Tetszőleges bemeneti gráf minimális méretű lefogó ponthalmazának keresése (MIN_LEFOGÓ).

Bonyolultság. A 2. fejezetben már esett szó róla, hogy a maximális független ponthalmaz keresése (MAX_FTL) NP-nehéz feladat. Ebből következik, hogy a MIN_LEFOGÓ is NP-nehéz, hiszen egy $G = (V,E)$ gráfban egy X ponthalmaz akkor és csak akkor minimális lefogó halmaz, ha $V \setminus X$ maximális független halmaz (egyébként a MIN_LEFOGÓ eldöntési változata is egyike Karp 21 problémájának). Érdekes tény, hogy miközben a MAX_FTL-re semmilyen k-ra nem létezik k-approximáció, feltéve, hogy P≠NP (ennek bizonyítása meghaladja a jegyzet kereteit), addig a MIN_LEFOGÓ-ra igen könnyű 2-approximációt adni.

2-approximációs algoritmus, első megoldás Keressük meg először a bemenetként kapott G gráf egy M maximális párosítását (erre csak páros gráfok esetén tanultunk algoritmust – a javítóutak módszerét –, de a feladat általános gráfokra is megoldható polinom időben), majd adjuk meg kimenetként a párosításban

szereplő élek által fedett csúcsokat.

Elemzés. A kapott csúcshalmaz tényleg lefogó lesz, hiszen ha lenne olyan él, melynek egyik végpontját sem tartalmazza, akkor ezt az élet az M párosításhoz véve ismét párosítást kapnánk, ami M maximalitása miatt lehetetlen. Azt kell még belátnunk, hogy a kapott csúcshalmaz mérete legfeljebb $2OPT$. Az M-ben szereplő élek lefogásához legalább $|M|$ csúcsra van szükség, hiszen semelyik két M-beli élnek nincs közös pontja, így $OPT \geq |M|$. Az algoritmus által adott csúcshalmaz elemszáma pedig $2|M| \leq 2OPT$.

2-approximációs algoritmus, második megoldás A fenti algoritmus valójában egyáltalán nem egyszerű, hiszen felhasznál egy általános gráfban maximális párosítást kereső algoritmust. Vegyük azonban észre, hogy az M párosításról csak azt használtuk ki, hogy G tetszőleges élét hozzávéve az új élhalmaz nem párosítás, vagyis M a G egyetlen élével sem bővíthető. Ilyen párosítást pedig nem nehéz találni: válasszuk ki G egy tetszőleges élét, majd töröljük az él két végpontját G-ből. A kapott gráfban ismét válasszunk egy élet és töröljük a végpontokat. Addig folytatjuk ezt az eljárást, amíg már nem marad csúcs hátra vagy a kapott gráfnak már nincsenek élei.

Elemzés. Világos, hogy a kapott élhalmaz párosítás lesz és az is, hogy ez a párosítás tovább nem bővíthető (ellenkező esetben nem állt volna le az eljárás, hiszen lenne még él a maradék gráfban). Az eddig látottak szerint tehát az így talált párosítás által fedett csúcsok is olyan lefogó ponthalmazt határoznak meg, melynek mérete legfeljebb $2OPT$. Az algoritmus lépésszáma nyilván polinomiális, sőt jóval kisebb, mint az előző algoritmus lépésszáma. Ez azért is figyelemre méltó, mert a második algoritmus általában valamivel jobb eredményt ad az elsőnél, hiszen a nem bővíthető párosítások nem feltétlen maximális méretűek; ez ugyanakkor nem jelenti azt, hogy az approximációs faktora jobb lenne, mint 2 (éles példát nem nehéz adni, ld. 19. feladat). Ez az algoritmus (melyet egymástól függetlenül talált meg Gavril és Yannakakis) már csakugyan nagyon egyszerű, aminek fényében igencsak meglepő, hogy ennél jobb konstans faktorú approximáció nem ismert a feladatra.

4.2. A súlyozott halmazfedési probléma

Probléma. A súlyozott halmazfedési probléma (a továbbiakban S_HALMAZ_FED) esetében adott egy n elemű U alaphalmaz, U részhalmazainak egy $\mathscr{R}$ rendszere, melyre $\cup_{S\in\mathscr{R}} S = U$, továbbá egy $c : \mathscr{R} \to \mathbb{R}^+$ költségfüggvény. A feladat U egy minimális összsúlyú fedésének megadása. Más szóval olyan $\mathscr{R}' \subseteq \mathscr{R}$ rendszert keresünk, melyre $\cup_{S\in\mathscr{R}'} S = U$ és $\sum_{S\in\mathscr{R}'} c(S)$ minimális.

Bonyolultság. A probléma NP-nehéz, ezt azzal igazoljuk, hogy visszavezetjük rá a MIN_LEFOGÓ-t. Maga a visszavezetés rendkívül egyszerű, valójában arról van szó, hogy a MIN_LEFOGÓ az S_HALMAZ_FED speciális esete. Legyen $G = (V,E)$ egy tetszőleges gráf, ennek keressük egy minimális lefogó ponthalmazát. Most megadjuk az S_HALMAZ_FED egy olyan esetét, melyet megoldva megkapjuk G egy minimális lefogó ponthalmazát. Legyen $U := E$, $\mathscr{R} := \{E_v : v \in V\}$, ahol E_v a G-ben a v csúcshoz csatlakozó élek halmaza. Legyen végül minden $\mathscr{R}$-beli halmaz költsége 1. Az így kapott súlyozott halmazfedési feladat nyilván ekvivalens lesz a G gráf egy minimális lefogó ponthalmazának megtalálásával. Ebből az is következik persze, hogy az S_HALMAZ_FED-nek még az a speciális esete is NP-nehéz, amikor minden halmaz súlya 1. Ennek a speciális esetnek az eldöntési változata is szerepel Karp 21 NP-teljes problémája között.

Az eddigiek azt sejtetik, hogy az S_HALMAZ_FED-re még approximációs algoritmust sem lesz egyszerű találni. A helyzet valóban ez, nem ismert konstans faktorú approximáció az S_HALMAZ_FED-re (sőt, azt sejtik, hogy ilyen nem is létezik, feltéve, hogy P≠NP). Ha azonban nem követeljük meg, hogy az approximációs faktor konstans legyen (vagyis hogy egyáltalán ne függjön a bemenettől), akkor már adhatunk elfogadható közelítést. Chvátal alábbi mohó algoritmusának approximációs faktora $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}$ lesz, ahol k az $\mathscr{R}$-beli legnagyobb elemszámú halmaz mérete. A k szám akár meg is közelítheti az n-et (sőt, egyenlő is lehet vele), az approximációs faktor tehát akár H_n is lehet. Ismert az analízisből, hogy $\ln n \leq H_n \leq \ln n + 1$, tehát az approximációs faktor $\ln n$ körül lesz. Ennél nagyságrendileg jobb approximáció nem is ismert a feladatra.

Algoritmus. Az algoritmus során egyesével fogunk halmazokat választani $\mathscr{R}$-ből, és egy C halmazban tároljuk az U alaphalmaz azon elemeit, melyeket a már kiválasztott halmazok lefednek. C kezdetben természetesen üres. A soron következő $S \in \mathscr{R}$ halmazt úgy választjuk ki, hogy a $\frac{c(S)}{|S \setminus C|}$ hányados minimális legyen, más szóval mindig azt a halmazt választjuk, melynek egy újonnan lefedett elemre eső költsége minimális. Ezt addig folytatjuk, míg a teljes U halmazt le nem fedtük.

Elemzés. Az algoritmus természetesen polinomiális lesz, azt kell igazolnunk, hogy a kimenet összsúlya nem haladja meg az optimum H_k-szorosát. Ehhez definiáljuk egy elem árát az algoritmus által őt elsőként fedő halmaz egy elemre eső költségeként, vagyis ha az x elemet elsőként az S halmaz fedi, akkor x ára $\frac{c(S)}{|S \setminus C|}$, ahol C az x fedése előtti helyzetben már fedett elemek halmaza. Világos, hogy bármely, a fedésbe bevett halmaz súlya egyenlő az általa elsőként fedett elemek árainak összegével és így az is, hogy a fedés összsúlya az összes elem árainak összege lesz.
Legyen most $\mathscr{R}^* \subseteq \mathscr{R}$ egy minimális összsúlyú fedés és legyen $S \in \mathscr{R}^*$ tetszőleges. Legyenek S elemei $x_1, x_2, \ldots, x_r$, ahol $r \leq k$. Tegyük fel (anélkül, hogy az

az általánosság rovására menne), hogy az algoritmusunk S elemeit $x_r, x_{r-1}, \ldots, x_1$ sorrendben fedi le és vizsgáljuk azt a lépést, amikor x_i-t lefedjük. E lépést megelőzően x_i-n kívül az $x_{i-1}, x_{i-2}, \ldots, x_1$ elemek is fedetlenek, ekkor tehát az S halmazhoz tartozó $\frac{c(S)}{|S \setminus C|}$ hányados legfeljebb $\frac{c(S)}{i}$. Mivel az algoritmus a legkisebb hányadoshoz tartozó halmazt választja, a kiválasztott, x_i-t fedő halmazhoz tartozó hányados is legfeljebb ennyi, így definíció szerint x_i ára is legfeljebb $\frac{c(S)}{i}$. Mivel $i \leq r$ tetszőleges volt, ez azt jelenti, hogy az S elemeinek fedéséhez szükséges összes ár legfeljebb $\sum_{i=1}^{r} \frac{c(S)}{i} = c(S)H_r \leq c(S)H_k$. Az összes, $\mathscr{R}^*$ valamelyik halmazában szereplő elem árainak összege így legfeljebb $\sum_{S \in \mathscr{R}^*} H_k c(S) = H_k OPT$, (ahol OPT a minimális összsúlyú fedés súlya). Figyelembe véve, hogy $\mathscr{R}^*$ halmazai U minden elemét fedik, az összes elem árainak összege pedig épp az algoritmus által adott fedés összsúlya, ezzel épp a bizonyítandó állítást kaptuk.

Éles példa. Megmutatjuk, hogy a H_k, sőt a H_n approximációs faktornál jobbat nem lehet az algoritmusra bizonyítani. Legyen ehhez $U = \{1, 2, \ldots, n\}$, az $\mathscr{R}$ rendszer pedig álljon az $\{1\}, \{2\}, \ldots, \{n\}$ halmazokból és magából U-ból. Legyen ezen kívül $c(\{i\}) = \frac{1}{i}$, $c(U) = 1 + \varepsilon$, ahol $\varepsilon > 0$ később specifikálandó szám. Világos, hogy az optimális fedés csak az U halmazt fogja használni, költsége így $1 + \varepsilon$. Az algoritmus ezzel szemben az első lépésben az $\{n\}$ halmazt választja, hiszen erre lesz a legkisebb az egy új elemre eső költség ($\frac{1}{n}$, az U esetében ez a költség $\frac{1+\varepsilon}{n}$). A második lépésben az algoritmus az $\{n-1\}$ halmazt választja, hiszen most erre lesz a legkisebb az egy új elemre eső költség ($\frac{1}{n-1}$, az U esetében ez most $\frac{1+\varepsilon}{n-1}$, hiszen egy elem már le van fedve). A harmadik lépésben az algoritmus az $\{n-2\}$ halmazt választja, és így tovább, végül utolsóként az $\{1\}$ halmaz kerül sorra, aminek az egy új elemre eső költsége ekkor már kevesebb lesz, mint U egy új elemre eső költsége (1, szemben az $(1+\varepsilon)$-nal). Az algoritmus által adott fedés teljes súlya tehát épp H_n lesz, ami $\frac{H_n}{1+\varepsilon}$-szorosa az optimálisnak. Mivel az $\varepsilon \geq 0$ szám tetszőlegesen kicsinek választható, semmilyen H_n-nél kisebb számra sem teljesülhet, hogy kisebb, mint $\frac{H_n}{1+\varepsilon}$ (hiszen ez utóbbi $\varepsilon \to 0$ esetén H_n-hez tart), így semmilyen H_n-nél kisebb szám sem lehet alkalmas az algoritmus approximációs faktorának.

4.3. Steiner-fák

Egy élsúlyozott összefüggő gráf minimális összsúlyú feszítőfájának polinom idejű megtalálására számos algoritmus ismert (pl. Borůvka, Kruskal, Jarník-Prim), és ezek gyakorlati alkalmazásokban is fel szoktak bukkanni (Borůvka például 1926-ban Morvaország villamosítása kapcsán fedezte fel a róla elnevezett algoritmust, melynek fontosságát az is mutatja, hogy 1938-ban, 1951-ben és 1965-ben is újra felfedezték). Lényegesen bonyolultabb a helyzet, ha a keresett fa nem kell, hogy

a gráf minden csúcsát tartalmazza, hanem csak bizonyos előre adottakat.

Probléma. A Steiner-fa probléma esetében adott egy $G = (V,E)$ egyszerű, összefüggő gráf, a csúcsok egy kettéosztása az S és T halmazokra ($S \cup T = V$, $S \cap T = \emptyset$, S elemeit Steiner-pontoknak, T elemeit termináloknak hívjuk), továbbá egy $c : E \to \mathbb{R}^+$ élsúlyozás. F a G Steiner-fája, ha F olyan részgráfja G-nek, amely T minden csúcsát tartalmazza és persze fa. Összefüggő gráfnak mindig van Steiner-fája, hiszen bármelyik feszítőfa ilyen lesz. Egy Steiner-fa költsége (vagy súlya) a fában szereplő élek súlyainak összege. A feladat G egy minimális költségű Steiner-fájának megadása (a bemenet természetesen G-n kívül az S,T szétosztást és a c élsúlyozást is tartalmazza).

Bonyolultság. A feladat eldöntési változata egyike Karp 21 NP-teljes problémájának, így persze az optimalizálási verzió NP-nehéz, de számos speciális esete megoldható polinom időben (ld. 27., 28., 29. feladatok). Az alábbiakban egy 2-approximációt fogunk megadni (a legjobb ismert approximáció faktora kb. 1,39, ugyanakkor $\frac{96}{95}$-nél jobb approximációs faktorú algoritmust csak P=NP esetén lehet adni).

Algoritmus. Először a probléma egy jóval egyszerűbbnek tűnő speciális esetére adunk 2-approximációt, majd megmutatjuk, hogy hogyan adhatunk ugyanilyen approximációt tetszőleges bemenetre. A speciális eset az úgynevezett metrikus Steiner-fa probléma, itt a G gráf teljes gráf, az élsúlyozás pedig teljesíti a háromszög-egyenlőtlenséget, vagyis tetszőleges x,y,z csúcsokra és a köztük futó $(x,y),(y,z),(z,x)$ élekre teljesül, hogy $c((x,z)) \leq c((x,y)) + c((y,z))$. Jelöljük egy $G = (V,E)$ gráf tetszőleges $X \subseteq V$ csúcshalmaz által feszített részgráfját $G[X]$-szel. Az algoritmus rendkívül egyszerű lesz: állítsunk elő a $G[T]$ gráfban egy (a c élsúlyozás szerint) minimális súlyú feszítőfát (pl. Kruskal algoritmusával) és adjuk meg ezt a fát kimenetként. Az általános eset kezeléséhez legyen most (G,S,T,c) egy tetszőleges, nem feltétlen metrikus bemenet, ebből előállítjuk a metrikus speciális eset egy (G',S',T',c') bemenetét; ezt az eljárást hívjuk metrizálásnak. G' legyen teljes gráf G csúcshalmazán, $S' = S, T' = T$, az (x,y) él $c'(x,y)$ súlya pedig legyen az x és y csúcsok közti, G-beli legrövidebb út hossza, a c súlyozás szerint. Ilyen út mindig létezik, mivel G összefüggő. Világos, hogy a kapott bemenet metrikus, hiszen G' teljes gráf és bármely x,y,z csúcsokra $c'((x,z)) \leq c'((x,y)) + c'((y,z))$, ellenkező esetben egy x és y közti legrövidebb út éleihez egy y és z közti legrövidebb út éleit adva olyan x és z közti élsorozatot kapnánk, melynek hossza kisebb, mint $c'((x,z))$, ami lehetetlen. Használjuk most a metrikus esetre adott algoritmust a (G',S',T',c') bemenetre (vagyis adjunk meg egy minimális súlyú feszítőfát $G'[T]$-ben). Az így kapott F' fa persze nem lesz jó kimenet az eredeti problémára, hiszen elképzelhető, hogy bizonyos élei nem is szerepelnek G-ben. Helyettesítsük F' minden (x,y) élét az x és y közti G-beli legrövidebb út éleivel. Az így kapott K élhalmazban (precízebben a K és a K-beli élek

által fedett csúcsok által meghatározott G_K gráfban) lehetnek körök, sőt K-ban G élei többszörösen is szerepelhetnek. Vegyük ezért G_K egy F'' minimális feszítőfáját (ismét Kruskal algoritmusát használva), ez már G Steiner-fája lesz; adjuk meg ezt kimenetként. Az algoritmus futására példát a Feladatok alfejezetben találunk (26. feladat).

Elemzés. Lássuk be először, hogy a metrikus esetre adott algoritmusunk 2-approximáció; legyen a bemenet (G,S,T,c). Az, hogy Steiner-fát kapunk, éspedig polinom időben, nyilvánvaló, az approximációs faktort kell igazolnunk. Legyen D optimális Steiner-fa, duplázzuk meg ennek az éleit (vagyis helyettesítsünk minden élet két párhuzamos éllel, melyek az eredeti él végpontjai közt mennek és súlyuk is azonos az eredeti él súlyával), nevezzük a kapott gráfot D'-nek. D' összefüggő, hiszen D fa volt és minden csúcs D'-beli foka páros, így létezik D'-nek egy $U = (u_0, f_1, u_1, f_2, u_2, \ldots, f_m, u_m = u_0)$ Euler-körsétája (itt $u_0, u_1, \ldots, u_m$ D'-beli csúcsok, $f_1, f_2, \ldots, f_m$ pedig D' élei és az f_i él az u_{i-1} és u_i csúcsok közt megy). U-ból előállítjuk a $G[T]$ gráf egy H Hamilton-körét a következőképp. Tekintsük D' csúcsainak az $(u_0, u_1, u_2, \ldots, u_m)$ sorozatát, vagyis vegyük a csúcsokat az U végigjárása szerinti sorrendben, egy csúcsot akár többször is (egy D'-ben $2d$ fokú csúcs d-szer fog előfordulni, de ez nem lényeges). Tartsuk meg minden T-beli csúcsnak az első előfordulását ebben a sorozatban (vagyis kidobjuk a sorozatból az S-beli csúcsokat és a T-beli csúcsok második és minden további példányát); vegyük észre, hogy ekkor T minden csúcsa szerepelni fog, hiszen D Steiner-fa, így D és D' is tartalmaz minden T-beli csúcsot. Legyen az így kapott sorrend $(u_{i_1}, u_{i_2}, \ldots, u_{i_{|T|}})$. Álljon most H az $(u_{i_1}, u_{i_2}), (u_{i_2}, u_{i_3}), \ldots, (u_{i_{|T|-1}}, u_{i_{|T|}}), (u_{i_{|T|}}, u_{i_1})$ élekből. H nyilván Hamilton-kör $G[T]$-ben. Megmutatjuk, hogy H éleinek összköltsége, $c(H)$ legfeljebb $2OPT$. Ehhez szükségünk lesz az alábbi egyszerű lemmára.

4.3. Lemma *Legyen c a G teljes gráf metrikus élsúlyozása és legyen R tetszőleges élsorozat az x és y csúcsok közt G-ben. Ekkor $c((x,y)) \leq c(R)$.*

Bizonyítás. Az R-ben szereplő élek r száma szerinti teljes indukcióval bizonyítunk. Ha $r = 1$, akkor az állítás magától értetődő. Az indukciós feltevés az lesz, hogy $r = i$ esetén az állítás igaz; azt kell belátnunk, hogy ekkor $r = i+1$ esetén is igaz lesz. Legyen (x,z) az R élsorozat első éle és jelöljük R'-vel az R-nek a z és y közti, $r-1 = i$ élből álló részét. Ekkor

$$c(R) = c((x,z)) + c(R') \geq c((x,z)) + c((z,y)) \geq c((x,y)),$$

ahol az első egyenlőtlenség az indukciós feltétel, a második pedig c metrikussága miatt teljesül. □

Figyeljük most meg, hogy H-t úgy kapjuk az U Euler-körsétából, hogy a körséta u_{i_1} és u_{i_2} közti szakaszát az (u_{i_1}, u_{i_2}) éllel helyettesítjük, az u_{i_2} és u_{i_3} közti szakaszt

az (u_{i_2}, u_{i_3}) éllel helyettesítjük, és így tovább, végül az $u_{i_{|T|}}$ és u_{i_1} közti szakaszt az $(u_{i_{|T|}}, u_{i_1})$ éllel helyettesítjük. Ezt az eljárást hívjuk a körséta levágásának ($G[T]$ egy Hamilton-körévé). A 4.3. Lemma szerint a helyettesítések nem növelik az élhalmaz összsúlyát (hiszen minden levágott élsorozat helyére épp az élsorozat két végpontja közti él kerül), így teljesül, hogy

$$c(H) \leq c(U) = c(D') = 2c(D) = 2OPT.$$

Hagyjunk el most egy tetszőleges élet H-ból. A kapott H' gráfra nyilván $c(H') \leq c(H) \leq 2OPT$, hiszen az élsúlyozás nem negatív. H' a $G[T]$ egy Hamilton-útja, vagyis egyben feszítőfája is, így a kimenetként adott minimális feszítőfa súlya ennél nem lehet nagyobb, tehát szintén legfeljebb $2OPT$, így a metrikus esetben csakugyan 2-approximációt kaptunk.

Lássuk be most, hogy az általános esetre adott algoritmusunk is 2-approximáció; legyen a bemenet (G,S,T,c). Az eljárás nyilván polinomiális, az F'' kimenet pedig csakugyan Steiner-fa, hiszen F'' fa, csak G-beli élei vannak és minden olyan csúcsot tartalmaz, amit F', amely pedig tartalmazza T minden csúcsát. Az approximációs faktor igazolásához legyen OPT, illetve OPT' a (G,S,T,c)-hez, illetve a (G',S',T',c')-hez tartozó optimális Steiner-fa összsúlya, jelöljük továbbá $c(H)$-val, illetve $c'(H)$-val egy H gráf éleinek összsúlyát a c, illetve c' súlyozás szerint. Ekkor

$$c(F'') \leq c(K) = c'(F') \leq 2OPT' \leq 2OPT.$$

F'' élei a K éleinek részhalmazát alkotják és a c súlyozás nem negatív, ez igazolja az első egyenlőtlenséget. Az ezt követő egyenlőség c' és K definíciójából következik, $c'(F') \leq 2OPT'$ pedig abból, hogy a metrikus esetre vonatkozó algoritmus 2-approximáció. Végül az utolsó egyenlőtlenség igazolásához legyen F^* egy (G,S,T,c)-hez tartozó optimális Steiner-fa. Ekkor F^* minden T-beli csúcsot tartalmaz, minden $e = (x,y)$ éle szerepel G'-ben is, sőt $c'(e) \leq c(e)$, hiszen e egy egy élű út x és y között G-ben, $c'(e)$ pedig az x és y közti legrövidebb út hossza (a c súlyozás szerint). Így $OPT' \leq c'(F^*) \leq c(F^*) = OPT$.
A fentiek szerint az F'' Steiner-fa összsúlya tehát valóban legfeljebb az optimum kétszerese. Érdemes megfigyelni, hogy az általános eset vizsgálatakor annak, hogy épp a 2-es approximációs faktort akartuk igazolni, nem volt jelentősége: ha bármilyen k-ra van egy k-approximációnk a metrikus esetre, akkor a leírt módon k-approximációt kapunk az általános esetre is.

Éles példa. Legyen G teljes gráf az $\{1,2,\ldots n\}$ csúcsokon, $S = \{1\}$, $T = \{2,3,\ldots,n\}$, $c((1,i)) = 1$ minden $i = 2,3,\ldots,n$ esetén és $c((i,j)) = 2$ minden $i \neq j$, $i,j \neq 1$ esetén. Ez a probléma egy metrikus esete, az algoritmus futtatása során tehát a metrizálási lépésre nincs szükség, a kimenet a $2,3,\ldots,n$ csúcsok által feszített részgráf egy feszítőfája lesz, melynek súlya $2(n-2)$. Az optimális

Steiner-fa ugyanakkor egy 1 középpontú csillag, ennek súlya $n-1$. A kapott eredmény tehát az optimum $\frac{2(n-2)}{n-1}$-szerese lesz, amiből következik, hogy semmilyen 2-nél kisebb szám sem alkalmas az algoritmus approximációs faktorának, hiszen bármely $k < 2$ esetén létezik olyan n, melyre $\frac{2(n-2)}{n-1} > k$ (mivel $n \to \infty$ esetén $\frac{2(n-2)}{n-1} \to 2$).

4.4. Az utazóügynök probléma

Képzeljük el, hogy egy automatákat üzemeltető cég alkalmazottjaként a cég telephelyéről kiindulva és ugyanoda visszaérkezve végig kell látogatnunk az automatákat a lehető legkisebb összköltséggel (költségnek tekinthetjük például az eltelt időt, a megtett út hosszát, a felhasznált üzemanyag árát, vagy akár egy ezekből számított függvényt). Ha minden automatát csak egyszer látogathatunk meg és a telephelyre sem térhetünk vissza mielőtt végeztünk volna, akkor a feladat a jól ismert utazóügynök problémával (minimális költségű Hamilton-kör keresése) ekvivalens. Érdemes azonban megemlíteni, hogy ha ilyen kikötés nincs, más feladatot kapunk: ekkor olyan minimális költségű zárt élsorozatot kell találnunk, mely minden csúcsot tartalmaz. A két feladat közti különbség egyáltalán nem elhanyagolható: mindkettő NP-nehéz, de az utazóügynök probléma nem is approximálható, míg a másik igen (ld. 35. feladat). A némiképp meglepő jelenség oka az, hogy az élsúlyozás nem feltétlenül metrikus.

Probléma. Az utazóügynök probléma bemenete egy nem negatívan élsúlyozott teljes gráf (tehát (G,c), ahol G teljes gráf, $c : E(G) \to \mathbb{R}^+$), a feladat G egy minimális összsúlyú Hamilton-körének megadása.

Bonyolultság. Korábbi tanulmányaink során már láttuk, hogy ez a probléma NP-nehéz, most ennél jóval többet mutatunk meg: a problémára semmilyen konstans k-ra sem létezik k-approximációs algoritmus, feltéve, hogy P$\neq$NP. Tegyük fel ugyanis ezzel ellentétben, hogy ilyen algoritmus valamely k-ra létezik. Megmutatjuk, hogy ekkor polinom időben el tudnánk dönteni, hogy egy tetszőlegesen adott G gráfnak van-e Hamilton-köre, amiből az állítás a Hamilton-kör probléma NP-teljessége miatt következik. Hozzunk létre G csúcshalmazán egy G' teljes gráfot a következő élsúlyozással. Azon élek, amelyek G-ben is benne voltak, kapjanak 1 súlyt, a többi él pedig legyen kn súlyú, ahol n a G csúcsainak száma. Figyeljük meg, hogy ekkor G' Hamilton-köreinek súlya vagy n (ha csupa 1 súlyú élből áll a kör), vagy legalább $kn+n-1$ (ha van a körben 1-től különböző súlyú él). Futtassuk most az így kapott bemenetre a feltételezett k-approximációs algoritmust. Ez polinom időben találna egy olyan Hamilton-kört, melynek súlya az optimumnak legfeljebb k-szorosa. Ha a talált kör súlya n, akkor az eredeti G gráfban van Hamilton-kör, hiszen az n súlyú kör csupa 1-es élből áll, ezek pedig

G-nek is élei. Ha viszont a talált kör súlya nagyobb, mint n, akkor az előzőek szerint a súly legalább $kn+n-1$. Ekkor G-ben nem lehetett Hamilton-kör, ellenkező esetben a G'-beli optimális Hamilton-kör súlya n lenne, az algoritmus (mivel k-approximációs) így legfeljebb kn súlyú kört kellene, hogy adjon kimenetként, miközben a talált kör súlya legalább $kn+n-1$. Mivel G' előállítása G-ből polinom időben megvalósítható és G' mérete csak kevéssel nagyobb, mint G mérete (szomszédsági mátrix használata esetén pl. legfeljebb $\log kn$-szerese, míg G mérete n^2), így az utazóügynök problémára adott k-approximációs algoritmus segítségével csakugyan polinom időben tudnánk eldönteni egy tetszőleges gráfról, hogy van-e Hamilton-köre.
A 2. fejezetben esett arról szó, hogy ha gyakorlati feladatok megoldásakor találkozunk NP-nehéz problémával, akkor érdemes megvizsgálni, hogy nem elég-e a probléma egy speciális esetét kezelni. Az utazóügynök probléma esetében pontosan erről van szó: a gyakorlati alkalmazások során szinte biztosan teljesül a háromszög-egyenlőtlenség az élsúlyozásra, ilyenkor beszélünk metrikus utazóügynök problémáról. Sajnos még a metrikus utazóügynök probléma is NP-nehéz (ld. 32. feladat), erre azonban már létezik kostans faktorú approximációs algoritmus. Az alábbiakban először egy 2-approximációt adunk meg, majd ezt finomítva egy $\frac{3}{2}$-approximációt, mely Christofides nevéhez fűződik. Bár Christofides algoritmusa 1976-ból származik, ennél jobb approximáció jelenleg sem ismert a feladatra. A legjobb k approximációs faktorról ennek ellenére csak annyit lehet tudni, hogy $k \geq \frac{220}{219}$. Egy további speciális esetre, az euklideszi utazóügynök problémára (ahol a csúcsok a síkon helyezkednek el, a köztük lévő élek súlya pedig a pontok közti szokásos távolság) ugyanakkor tetszőleges $\varepsilon > 0$ mellett létezik $(1+\varepsilon)$-approximáció (ilyen algoritmusokkal bővebben az 5. fejezetben foglalkozunk).

2-approximációs algoritmus. Az algoritmus három fő lépésből áll:

(1) G egy minimális összsúlyú F feszítőfájának megkeresése.

(2) F éleinek megduplázása és az így kapott F' gráfban egy S Euler-körséta megkeresése.

(3) Az S Euler-körséta átalakítása Hamilton-körré, a Steiner-fa problémánál (az approximációs faktor igazolásakor) látott levágásos technikával.

Az első lépés megvalósítása polinom időben nem okoz nehézséget: használhatjuk például Kruskal algoritmusát.

A második lépésben először F minden e élét két olyan éllel helyettesítjük, mely e végpontjai között megy. Mivel F összefüggő, ezért az így kapott F' is az lesz. Nyilvánvaló továbbá, hogy F' minden csúcsa páros fokú, így Euler tétele szerint F'-nek van Euler-körsétája. Az Euler-körsétát többféleképp is megkaphatjuk, az alábbi algoritmus polinomialitása könnyen látható (sőt, az algoritmus lineáris idejű lesz). Kiválasztjuk F' egy tetszőleges v_0 csúcsát és keresünk egy v_0-ból induló

S körsétát a következőképp: legyen $e_1 = (v_0, v_1)$ egy tetszőleges, v_0-hoz csatlakozó él. Válasszunk most egy v_1-hez csatlakozó $e_2 = (v_1, v_2)$ élet úgy, hogy az ne legyen azonos e_1-gyel, majd egy $e_2 = (v_2, v_3)$ élet úgy, hogy az ne legyen azonos sem e_1-gyel, sem e_2-vel. Folytassuk az eljárást amíg lehetséges, vagyis amíg az e_i élen elért v_i csúcsból indul ki olyan F'-beli él, mely a korábbiak egyikével sem azonos. Az így kapott

$$S = (v_0, e_1, v_1, e_2, v_2, \ldots, e_r, v_r)$$

élsorozat élei mind különbözők, ugyanez azonban nem feltétlen kell, hogy teljesüljön a csúcsokra, sőt könnyen látható, hogy $v_0 = v_r$: ha v_r különbözne v_0-tól, akkor az S élsorozatban páratlan sok él illeszkedne rá és mivel F'-ben minden fokszám páros, illeszkedne rá olyan él, amely F'-ben benne van, de S-ben nincs, vagyis az eljárás nem állt volna le v_r-ben. S tehát zárt élsorozat, sőt az is igaz (az előbbiek alapján), hogy minden $v_0 = v_r$-re illeszkedő él szerepel benne. Ha most S az F' minden élét tartalmazza, akkor az algoritmus leáll, ha nem, akkor megkeressük azt a legkisebb i számot, melyre F'-nek van olyan S-en kívüli éle, amely illeszkedik v_i-re. Keressünk most az előzőkhöz hasonlóan v_i-ből induló zárt élsorozatot az $F' \setminus S$ gráfban ($F' \setminus S$ minden fokszáma is páros, hiszen S zárt élsorozat), legyen ez

$$S' = (v_i, e_{r+1}, v_{r+1}, e_{r+2}, v_{r+2}, \ldots, e_{r+l}, v_{r+l} = v_i).$$

Illesszük be az S' élsorozatot S-be a v_i csúcsnál, vagyis a továbbiakban legyen

$$S := (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i, e_{r+1}, v_{r+1}, e_{r+2}, v_{r+2}, \ldots$$

$$\ldots, e_{r+l}, v_{r+l} = v_i, e_{i+1}, v_{i+1}, e_{i+2}, v_{i+2}, \ldots, e_r, v_r = v_0).$$

Ha most már S az F' minden élét tartalmazza, akkor az algoritmus leáll, ellenkező esetben addig ismételjük az eljárást (vagyis egy újabb S' zárt élsorozat beillesztését az aktuális S-be), amíg ez be nem következik. Ha már minden S által fedett csúcsra teljesül, hogy minden rá illeszkedő F'-beli él szerepel S-ben is, akkor F' összefüggősége miatt S az F' Euler-körsétája lesz. Euler-körséta megtalálására persze más polinomiális algoritmust is adhatunk (ld. 36. és 37. feladat).

A harmadik lépéshez legyen $S = (u_0, f_1, u_1, f_2, u_2, \ldots, f_m, u_m = u_0)$ az F' Euler-körsétája, ebből a már ismert levágásos technika segítségével készítünk Hamilton-kört. Tekintsük G csúcsainak az $(u_0, u_1, u_2, \ldots, u_m)$ sorozatát, vagyis vegyük a csúcsokat az S végigjárása szerinti sorrendben. Tartsuk meg minden csúcsnak csak az első előfordulását ebben a sorozatban, legyen az így kapott sorrend $(u_{i_1}, u_{i_2}, \ldots, u_{i_n})$. Az algoritmus kimenete ekkor az $(u_{i_1}, u_{i_2}), (u_{i_2}, u_{i_3}), \ldots, (u_{i_{n-1}}, u_{i_n}), (u_{i_n}, u_{i_1})$ élekből álló H Hamilton-kör lesz.

Elemzés. A kapott H kör nyilván Hamilton-kör, hiszen minden csúcs legfeljebb egyszer szerepel benne és mivel F a G feszítőfája, S a G minden csúcsát tartalmazza, így ugyanez igaz H-ra is. Mindhárom lépés megvalósítható polinom időben;

a 2-es approximációs faktort kell még igazolnunk. Legyen H^* a G egy minimális összsúlyú Hamilton-köre, ennek összsúlya OPT. H^* egy élét elhagyva Hamilton-utat kapunk, melynek súlya legfeljebb OPT, hiszen az élsúlyozás nem negatív. A Hamilton-utak feszítőfák is, így G-nek létezik legfeljebb OPT költségű feszítőfája, vagyis $c(F) \leq OPT$, hiszen F minimális költségű feszítőfa. F'-t F éleinek duplázásával kaptuk, így $c(F') = 2c(F) \leq 2OPT$. Mivel S az F' Euler-körsétája, annak minden élét pontosan egyszer tartalmazza, tehát $c(S) = c(F') \leq 2OPT$. A H Hamilton-kört úgy kapjuk az S Euler-körsétából, hogy a körséta u_{i_j} és $u_{i_{j+1}}$ közti szakaszát (itt $u_{i_{n+1}} = u_{i_1}$) egyetlen $(u_{i_j}, u_{i_{j+1}})$ éllel helyettesítjük. A háromszög-egyenlőtlenség miatt a két csúcs közti él súlya nem lehet nagyobb, mint egy, a két csúcs közt menő élsorozat összsúlya (4.3. Lemma), így a helyettesítés során az összsúly nem nőhet, vagyis $c(H) \leq c(S) \leq 2OPT$, amivel az approximációs faktor igazolását be is fejeztük.

$\frac{3}{2}$-approximációs algoritmus. Az imént látott 2-approximáció (2) lépésében megdupláztuk F éleit, hogy a kapott gráfnak legyen Euler-körsétája, ez a cél azonban kevesebb él hozzávételével is elérhető. Legyen V_p az F páratlan fokú csúcsainak halmaza. Ha bizonyos élek hozzáadásával a V_p-veli csúcsoknak eggyel megnövelnénk a fokát, miközben a többi csúcs foka nem változik, akkor csak páros fokú csúcsokkal rendelkező gráfot kapnánk. A páratlan fokú csúcsok száma minden gráfban páros (ellenkező esetben a gráf fokszámösszege páratlan lenne, ami lehetetlen), így F-ben is, tehát $|V_p|$ páros. Az élduplázás helyett adjuk F-hez $G[V_p]$ (vagyis G V_p által feszített részgráfja) egy teljes párosítását. Ilyen párosításból persze sok van, mi egy minimális összsúlyút szeretnénk választani, ami – ha nem is egyszerűen, de Edmonds egy (itt nem tárgyalt) algoritmusa segítségével – polinom időben megoldható. A 2-approximációs algoritmus többi részét változatlan formában átvesszük, (2) helyett pedig – a fentieknek megfelelően – az alábbi (2') lépést használjuk:

(2') F éleihez hozzávesszük $G[V_p]$ egy M minimális összköltségű teljes párosítását, ahol V_p az F páratlan fokú csúcsainak halmaza.

Az algoritmus futására példákat a Feladatok alfejezetben találunk.

Elemzés. Az M párosításban a V_p-beli csúcsok foka 1, a V_p-n kívülieké 0, így a (2') lépés csakugyan eléri a célját. Az előző algoritmus approximációs faktorának igazolásánál látottak szerint elég belátnunk, hogy M éleinek összsúlya legfeljebb $\frac{OPT}{2}$. Legyen H^* ismét egy optimális Hamilton-kör G-ben és legyen $a_1, a_2, \ldots, a_{2k}$ a V_p-beli csúcsok egy, a H^*-on való elhelyezkedés szerint vett sorrendje (vagyis H^* a_i és a_{i+1} közötti szakaszán nincs másik V_p-beli csúcs semelyik $i \leq 2k$ esetén). Legyen $\widehat{H^*}$ a V_p csúcsaiból és az $(a_1, a_2), (a_2, a_3), \ldots, (a_{2k-1}, a_{2k}), (a_{2k}, a_1)$ élekből álló Hamilton-kör. $\widehat{H^*}$-ot úgy kapjuk H^*-ból, hogy H^* a_i és a_{i+1} közti szakaszát (itt $a_{2k+1} = a_1$) az (a_i, a_{i+1})

éllel helyettesítjük. A 4.3. Lemma szerint bármely két csúcs közti él súlya nem lehet nagyobb, mint egy, a két csúcs közt menő élsorozat összsúlya, így $c(\widehat{H^*}) \leq c(H^*) = OPT$. Legyen most P_1 az $(a_1,a_2),(a_3,a_4),\ldots,(a_{2k-1},a_{2k})$ élek halmaza, P_2 pedig az $(a_2,a_3),(a_4,a_5),\ldots,(a_{2k-2},a_{2k-1}),(a_{2k},a_1)$ élek halmaza. Ekkor P_1 és P_2 teljes párosításai $G[V_p]$-nek, melyek (diszjunkt) uniója a $\widehat{H^*}$ Hamilton-kör, így $c(P_1)+c(P_2) = c(\widehat{H^*}) \leq OPT$. Így $c(P_1) \leq \frac{OPT}{2}$ és $c(P_2) \leq \frac{OPT}{2}$ közül legalább az egyik teljesül. Ez azt jelenti, hogy $G[V_p]$-nek létezik olyan párosítása, melynek összsúlya legfeljebb $\frac{OPT}{2}$, vagyis ugyanez az M minimális összköltségű teljes párosításra is igaz, és épp ezt akartuk belátni. Figyeljük meg, hogy P_1 és P_2 megkeresése nem része az algoritmusnak (hiszen ehhez egy H^* optimális Hamilton-körből indultunk ki), e párosításoknak csak a létezése fontos: ez igazolja azt, hogy az általunk használt M párosítás súlya is legfeljebb $\frac{OPT}{2}$.

4.5. Feladatok

12. (2) Adjunk 2-approximációs algoritmust tetszőleges gráfok élszínezésére.

13. (2) Mutassuk meg, hogy ha G éles példa a MAX_PÁROS-ra adott valamelyik approximációs algoritmushoz, akkor G páros élszámú páros gráf.

14. (2) Legyen k tetszőleges pozitív egész. Adjunk $2k$ élű éles példát a MAX_PÁROS-ra adott második approximációs algoritmushoz.

15. (2) Legyen k tetszőleges pozitív egész. Adjunk meg olyan $4k$ élű gráfot, mely éles példa lesz a MAX_PÁROS-ra adott mindkét approximációs algoritmushoz.

16. (3) Mutassuk meg, hogy ha G éles példa a MAX_PÁROS-ra adott első approximációs algoritmushoz, akkor az élszáma osztható 4-gyel.

17. (2) Adjunk olyan 2-approximációs algoritmust egy összefüggő gráf maximális élszámú páros részgráfjának megkeresésére, amely páros gráf bemenetekre optimális megoldást ad.

18. (2) Mutassuk meg, hogy tetszőleges G gráfra a MIN_LEFOGÓ-ra adott első algoritmus kimenete legfeljebb kétszer annyi csúcsot tartalmaz, mint a második algoritmus kimenete.

19. (1) Legyen k tetszőleges pozitív egész. Adjunk olyan G éles példát a MIN_LEFOGÓ-ra adott approximációs algoritmusokhoz, melyre $\tau(G) = k$.

20. (2) Adjunk meg olyan 10 csúcsú, 10 élű egyszerű, összefüggő gráfot, melyre a minimális lefogó ponthalmaz problémára látott 2-approximációs algoritmusok soha nem adnak optimális eredményt.

21. (3) Legyen G 10 csúcsú egyszerű gráf, melyre $\tau(G) = 8$. Igaz-e, hogy minden ilyen G gráfra létezik a MIN_LEFOGÓ-ra látott approximációk valamelyikének olyan futása, melyre a talált lefogó ponthalmaz minimális?

22. (2) Adjunk 2-approximációs algoritmust a maximális független ponthalmaz keresésének problémájára olyan bemenetek esetére, melyekre a független pontok maximális száma legalább $\frac{2n}{3}$, ahol n a bemeneti gráf csúcsszáma. Az algoritmus működését szemléltessük is a $K_{2,5}$ teljes páros gráfon.

23. (1) Tekintsük az {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,r,s,t,w,y} betűhalmazt, és az elemeiből képzett alábbi szavakat, mint részhalmazokat (a szavak utáni zárójelben lévő szám jelenti az adott halmaz költségét):

```
ike (3),  chef (3),  jimbo (3), stan (4),  mackey (4),
kyle (5), wendy (5), liane (6), randy (6), gerald (6)
```

Hajtsuk végre ezen adatokkal az S_HALMAZ_FED-re adott közelítő algoritmust.

24. (1) Tekintsük az {a,b,c,e,é,g,i,k,m,r,s,v} betűhalmazt, és az elemeiből képzett alábbi szavakat, mint részhalmazokat (a szavak utáni zárójelben lévő szám jelenti az adott halmaz költségét):

```
béka (2), maki (3),  egér (4),  réce (4),   maci (4),
bika (4), csiga (5), veréb (5), gébics (6), vércse (6)
```

Hajtsuk végre ezen adatokkal az S_HALMAZ_FED-re adott közelítő algoritmust.

25. (2) Adjunk $\frac{11}{6}$-approximációs algoritmust a MIN_LEFOGÓ 3-reguláris gráfokra vonatkozó speciális esetére.

26. (1) Egy hét csúcsú teljes gráf csúcsai legyenek a síkon egy egységoldalú szabályos hatszög csúcsai (sorrendben A,B,C,D,E,F) és középpontja, G. A G-ből induló élek súlya azonos az él végpontjainak síkbeli távolságával, a többi él súlya a végpontok síkbeli távolságánál $\frac{1}{2}$-del nagyobb. Hajtsuk végre és dokumentáljuk a Steiner-fa problémára tanult közelítő algoritmust a gráfra, ha a terminálok halmaza $\{B,C,E,F\}$.

27. (1) Mutassuk meg, hogy a Steiner-fa probléma polinom időben megoldható, ha a Steiner-pontok halmaza üres, illetve ha a terminálok halmaza két elemű.

28. (2) Igaz-e, hogy a Steiner-fa probléma polinom időben megoldható, ha a Steiner-pontok halmaza pontosan két elemű?

29. (2) Igaz-e, hogy a Steiner-fa probléma polinom időben megoldható, ha a Steiner-pontok halmaza legfeljebb $2\log n$ elemű, ahol n a gráf csúcsainak száma?

30. (1) Egy négyzet egyik átlóját osszuk három pont segítségével négy egyenlő részre. Legyenek a G teljes gráf csúcsai a négyzet csúcsai és az átlón lévő három pont, minden él súlya legyen azonos végpontjainak távolságával. Hajtsuk végre és dokumentáljuk a G gráfra a Christofides-algoritmust.

31. (1) Tekintsük az utazóügynök probléma azon speciális eseteit, amikor a gráf csúcsai egy szabályos n-szög csúcsai, az élek súlya pedig a végpontok síkbeli távolsága. Igaz-e, hogy ezen esetekre a Christofides-algoritmus optimális megoldást ad?

32. (2) Mutassuk meg, hogy a metrikus utazóügynök probléma NP-nehéz.

33. (2) Tekintsük a metrikus utazóügynök probléma azon speciális eseteit, amikor a gráf bármely két csúcsa közt létezik olyan út, amely csak 1 súlyú éleket használ. Igaz-e, hogy ezen esetekre polinom időben találhatunk olyan Hamilton-kört, melynek összsúlya legfeljebb $2n-2$ (ahol n a gráf csúcsainak száma)?

34. (2) Mutassuk meg, hogy a 33. feladat feltételei mellett lehetséges, hogy a gráfnak nincs $(2n-2)$-nél kisebb súlyú Hamilton-köre.

35. (3) Adjunk $\frac{3}{2}$-approximációs algoritmust egy összefüggő, nemnegatívan élsúlyozott gráf olyan minimális összsúlyú zárt élsorozatának megtalálására, mely minden csúcsot érint.

36. (3) Legyen G olyan összefüggő gráf, melyben minden fok páros, S pedig G egy tetszőleges élsorozata, melyben egyetlen él sem szerepel egynél többször. Adjunk polinomiális algoritmust annak eldöntésére, hogy S kiterjeszthető-e G egy Euler-körsétájává (vagyis létezik-e olyan Euler-körséta, melyben S élei az S-beli sorrendben szerepelnek), majd ezt felhasználva adjunk polinomiális algoritmust G egy Euler-körsétájának megadására.

37. (2) Legyen G olyan gráf, melyet egy fa éleinek duplázásával kaptunk. Adjunk a mélységi keresésen alapuló algoritmust G egy Euler-körsétájának megadására.

5. fejezet

Polinomiális approximációs sémák

Ebben a fejezetben olyan algoritmusokkal foglalkozunk, melyek segítségével NP-nehéz optimalizálási problémákat tetszőlegesen kicsi multiplikatív hibával tudunk közelíteni. Ennek persze tipikusan ára van: minél pontosabb a közelítés, annál nagyobb lesz a lépésszám. Valójában egy adott problémához nem is egy algoritmust, hanem algoritmusok egy családját kell megadnunk: minden adott, egynél nagyobb számhoz tartoznia kell egy polinomiális algoritmusnak, aminek az approximációs faktora az adott szám.

5.1. Definíció *Egy adott optimalizálási problémára approximációs sémának nevezünk egy olyan algoritmust, melynek két bemenete van: az adott probléma egy I esete és egy $\varepsilon > 0$ szám, a kimenet pedig egy $(1+\varepsilon)$-közelítő megoldás, vagyis maximalizálási probléma esetén olyan $x^* \in X_I$, melyre*

$$c(x^*) \geq \frac{1}{1+\varepsilon} \max_{x \in X_I} c(x),$$

minimalizálási probléma esetén pedig olyan $x^* \in X_I$, melyre

$$c(x^*) \leq (1+\varepsilon) \min_{x \in X_I} c(x),$$

ahol $c(x)$ szokás szerint az adott optimalizálási problémában szereplő célfüggvény, X_I pedig az I bemenethez tartozó, szóba jövő kimenetek halmaza.

5.2. Definíció *Egy approximációs séma polinomiális, ha a lépésszáma minden rögzített ε esetén az I bemenet méretében polinomiális.*

Ha egy problémára van polinomiális approximációs sémánk, akkor a definíció szerint gyakorlatilag bármennyire meg tudjuk közelíteni a megoldást úgy, hogy közben az algoritmus lépésszáma polinomiális marad, ami már csaknem olyan

hatékony, mintha polinomiális algoritmusunk lenne a problémára. Ne feledkezzünk meg azonban arról, hogy a lépésszám függhet (és tipikusan függ is) ε-tól, vagyis ahogy ε csökken, úgy a lépésszám kellemetlenül megnőhet. Ez az approximációs séma polinomialitását nem fogja befolyásolni, hiszen rögzített ε érték mellett bárhogy is függ a lépésszám ε-tól, ez csak a konstansokat változtatja meg. Például egy $\frac{1}{\varepsilon^2}n^3$ és egy $2^{\frac{1}{\varepsilon^2}}n$ lépésszámú approximációs séma is polinomiális lesz, de míg az első jól használható kis ε esetén is (mondjuk $\varepsilon = 0,001$), addig a második már $\varepsilon = 0,1$ mellett sem fog emberi időben lefutni. Érdemes tehát külön is megvizsgálni azokat az approximációs sémákat, melyeknél a lépésszámot az ε-tól függően is korlátozzuk.

5.3. Definíció *Egy approximációs séma teljesen polinomiális, ha a lépésszáma az I bemenet méretében és $\frac{1}{\varepsilon}$-ban is polinomiális.*

A lépésszám polinomialitását azért épp $\frac{1}{\varepsilon}$-ban szokás vizsgálni, mert ez végtelenhez tart, ha ε nullához. Persze $\frac{1}{\varepsilon}$ helyett lehetne $\frac{1}{\varepsilon^2}$-et vagy $\sqrt{\frac{1}{\varepsilon}}$-t is használni, de ez a lépésszám polinomialitásán nem változtatna. Pontosabb lenne ugyanakkor $\frac{1}{\varepsilon}$ helyett $\lfloor\frac{1}{\varepsilon}\rfloor$-ről beszélni (hiszen a lépésszám polinomialitását egész számokra definiáltuk), de mivel nyilvánvalóan nem okoz problémát, a definícióban és a továbbiakban is hanyagoljuk az egészrész jel használatát. Azt, hogy egy kétváltozós $f(x,y)$ függvény (jelen esetben a lépésszám, ahol a változók az I mérete és $\frac{1}{\varepsilon}$) az x változóban polinomiális, úgy kell érteni, hogy ha az y változót konstansnak tekintjük, akkor $f(x,y)$ polinomiális x-ben (és persze y-ban polinomiális, ha x-et konstansnak tekintve $f(x,y)$ polinomiális y-ban). Például az $f(x,y) = xy^3$ függvény x-ben és y-ban is polinomiális, míg az $f(x,y) = 3^x y$ függvény csak y-ban. Érdemes megfigyelni, hogy polinomiális, illetve teljesen polinomiális approximációs sémákat elég megadni valamilyen rögzített pozitív számnál kisebb ε-ok mellett, hiszen ha van egy $(1+\varepsilon)$-approximációnk, akkor az ennél rosszabb approximációk nem érdekesek.

5.1. A részösszeg probléma

Tegyük fel, hogy bizonyos csomagokat szeretnénk egy kamionnal elszállítani. A csomagok súlyai $a_1, a_2, \ldots, a_n$ kilogramm, a kamion teherbírása t kilogramm. Melyik csomagokat válasszuk ki, ha az a cél, hogy minél nagyobb összsúlyt szállítsunk el? Kiválaszthatók-e csomagok úgy, hogy a kamionnak ne maradjon fölösleges kapacitása? Az első kérdés esetén a részösszeg probléma optimalizálási, a második esetén a döntési verzióját kell megoldanunk (ez utóbbi egyike Karp 21 NP-teljes problémájának). A nyilvánvaló alkalmazásokon túl a probléma a kriptográfiában is komoly jelentőséggel bír. Az 1970-es évek végén például nyilvános

kulcsú titkosírást készítettek a segítségével, melyet azonban 1982-ben sikerült feltörni (ettől függetlenül a probléma persze továbbra is használható kriptográfiai célokra, csak nem az adott módon).

Probléma. A részösszeg probléma eldöntési változatában adottak az $a_1, a_2, \ldots, a_n$ pozitív egész számok és egy t célérték, ami szintén pozitív egész. A kérdés az, hogy ki lehet-e választani néhányat az $a_1, a_2, \ldots, a_n$ számok közül úgy, hogy az összegük éppen t legyen. Az optimalizálási változatnál ugyanezen bemenet mellett olyan részösszeget keresünk, mely nem nagyobb t-nél, de a lehető legjobban megközelíti azt (alulról). Formálisabban: olyan $I \subseteq \{1, 2, \ldots, n\}$ indexhalmazt keresünk, melyre $\sum_{i \in I} a_i \leq t$ mellett $\sum_{i \in I} a_i$ maximális.

Bonyolultság. Említettük, hogy az eldöntési változat NP-teljes , így természetesen az optimalizálási verzió is NP-nehéz. Érdemes megemlíteni, hogy az eldöntési feladatnak még az a speciális esete is NP-teljes, amikor $t = \frac{\sum_{i=1}^{n} a_i}{2}$, vagyis amikor azt kell eldöntenünk, hogy az a_i-k halmaza két azonos összegű részre bontható-e. E speciális eset neve PARTÍCIÓ probléma. Az alábbiakban megadunk egy teljesen polinomiális approximációs sémát a részösszeg optimalizálási feladatra.

Algoritmus. Először egy pontos algoritmust adunk, aminek lépésszáma persze nem lesz polinomiális, majd ezt picit „elrontva" kapunk egy nem teljesen pontos (de azért jól közelítő) és már polinomiális algoritmust. A pontos algoritmus olyan L_i növő listákat fog létrehozni, melyek az összes olyan részösszeget tartalmazzák, melyeket az $a_1, a_2, \ldots, a_i$ számokból lehet létrehozni. Érdemes (bár nem feltétlen szükséges) az $a_1, a_2, \ldots, a_n$ számokat növekvő sorrendbe tenni az algoritmus futtatása előtt. Egy L lista és a szám esetén használjuk az $L + a$ jelölést arra a listára, melynek elemei azok a számok, melyeket L elemeiből az a hozzáadásával kapunk, vagyis $L + a = \{l + a : l \in L\}$.
Az L_0 lista egyedül a 0-t tartalmazza, az L_i listát pedig az L_{i-1} és $L'_{i-1} := L_{i-1} + a_i$ listák összefésülésével kapjuk. Könnyen látható (ha nagyon precízek akarunk lenni, akkor pl. teljes indukcióval), hogy az L_i listában éppen azok a részösszegek lesznek benne (egyesek esetleg többször is), melyeket az $a_1, a_2, \ldots, a_i$ számokból lehet létrehozni, így az L_n lista az összes lehetséges részösszeget tartalmazza. Könnyen látható az is, hogy az L_i listák növők (nem csökkenők): az L_0 (egy elemű) lista persze növő és ha L_i növő, akkor L'_i is növő. Így az összefésüléses lépésnél két növekvő lista unióját kell növekvő sorrendben megkapni, ami (a korábbi tanulmányainkból ismert összefésüléses rendezés egy lépésének segítségével) a listák összméretében lineáris időben megvalósítható. A t-t alulról legjobban közelítő részösszeget így könnyű leolvasni: ez az L_n lista legnagyobb olyan eleme lesz, amely nem nagyobb t-nél. Ahhoz, hogy tudjuk, hogy ez a szám melyik elemek összegeként állt elő, persze minden részösszegről fel kell jegyeznünk, hogy hogyan kaptuk. Figyeljük meg, hogy a t-nél nagyobb elemekre igazából nincs

szükségünk, érdemes tehát az algoritmust úgy módosítani, hogy az L_i' listákba az ilyen elemeket már be se vegyük, ekkor ezek persze L_i-be és így L_n-be sem kerülhetnek be. A kimenethez tartozó részösszeg így egyszerűen az L_n lista legnagyobb eleme lesz. További gyorsítási lehetőség, hogy ha egy részösszeget többször is megkapnánk, akkor is csak egyszer szerepeltetjük a listá(k)ban. Az algoritmus persze tipikusan így is exponenciális futásidejű lesz, hiszen a listák hossza lépésenként akár meg is duplázódhat.

Az algoritmust a listák ritkítása segítségével gyorsítjuk fel; azokat a részösszegeket, melyekhez kellően közel van egy náluk kisebb részösszeg, ki fogjuk dobni a listákból. Legyen $0 < \delta$ adott konstans, $x \leq y$ pedig pozitív számok. Azt mondjuk, hogy *x képviseli y*-t, ha $(1+\delta)x \geq y$. Az L növekvő lista *δ-val történő ritkítása* azt jelenti, hogy a lista elemeiről alulról kezdve egyesével megvizsgáljuk, hogy az őket közvetlenül megelőző elem képviseli-e őket (vagyis ha a kérdéses elem y, a listában közvetlenül előtte lévő elem x, akkor fennáll-e, hogy $(1+\delta)x \geq y$). Ha a válasz nem, akkor továbblépünk a következő elemre, ha azonban igen, akkor továbblépés előtt az y elemet töröljük a listából. A törölt elemeket a további vizsgálatok során természetesen nem vesszük figyelembe, tehát minden y elemet az őt az aktuális listában közvetlenül megelőző x elemmel kapcsolatban vizsgálunk.

Most megadunk egy teljesen polinomiális approximációs sémát a részösszeg problémára $\varepsilon \leq 1$ mellett. A pontos algoritmus azon változatából indulunk ki, amikor a t-nél nagyobb elemeket töröltük (és a listákat persze növő sorrendben tartjuk nyilván). Ezt az algoritmust csak annyiban módosítjuk, hogy $1 \leq i \leq n-1$ esetén az L_i listát a létrehozás után $\delta := \frac{\varepsilon}{2n}$-nel ritkítjuk (vagyis az L_n listát már nem ritkítjuk). Az algoritmus futására példákat a Feladatok alfejezetben találunk.

Elemzés. Be kell látnunk, hogy a kimenetként kapott részösszeg értéke legalább $\frac{OPT}{1+\varepsilon}$ és hogy az algoritmus lépésszáma polinomiális a bemenet méretében és $\frac{1}{\varepsilon}$-ban is. A kimenet értékének becsléséhez figyeljük meg, hogy bármely olyan s részösszeghez, mely az $a_1, a_2, \ldots, a_n$ számokból előállítható, létezik egy olyan s' részösszeg, mely az L_n listában benne van és $s' \geq \frac{s}{(1+\frac{\varepsilon}{2n})^n}$. Ennek bizonyítása nem nehéz, a leírás azonban némiképp körülményes, így a Feladatok alfejezetbe száműztük (42. feladat). Ha s-et az optimális részösszegnek választjuk, akkor azt kapjuk, hogy létezik olyan részösszeg az L_n listában, melynek értéke legalább $\frac{OPT}{(1+\frac{\varepsilon}{2n})^n}$, így a kimenetként kapott részösszeg értéke is legalább ennyi.

5.4. Lemma $(1+\frac{\varepsilon}{2n})^n \leq 1+\varepsilon$.

Bizonyítás. Felbontjuk a zárójeleket az $(1+\frac{\varepsilon}{2n})^n$ kifejezésben a binomiális tétel segítségével, majd az eredmény egyszerű felső becsléseit adjuk (az utolsó előtti egyenlőtlenségnél felhasználjuk, hogy $\varepsilon \leq 1$):

$$\left(1+\frac{\varepsilon}{2n}\right)^n = \sum_{i=0}^{n} \left(\frac{\varepsilon}{2n}\right)^i \binom{n}{i} = \sum_{i=0}^{n} \frac{\varepsilon^i}{2^i n^i} \frac{n(n-1)\ldots(n-i+1)}{i!} \leq$$

$$\leq \sum_{i=0}^{n} \frac{\varepsilon^i}{2^i n^i} n^i = \sum_{i=0}^{n} \frac{\varepsilon^i}{2^i} = 1 + \sum_{i=1}^{n} \frac{\varepsilon^i}{2^i} \leq 1 + \sum_{i=1}^{n} \frac{\varepsilon}{2^i} = 1 + \varepsilon \sum_{i=1}^{n} \frac{1}{2^i} \leq 1 + \varepsilon.$$

□

Az algoritmus kimeneteként kapott részössszeg értéke tehát legalább $\frac{OPT}{(1+\frac{\varepsilon}{2n})^n} \geq \frac{OPT}{1+\varepsilon}$.

A futásidő vizsgálatához először vegyük észre, hogy elegendő azt belátni, hogy a listák hossza polinomiális a bemenet méretében. Valóban, az L_i' legyártása L_i-ből, a két lista összefésülése, illetve a ritkítás is megvalósítható a lista méretében polinomiális időben, s mivel ezeket legfeljebb n-szer végezzük el, a teljes futásidő is polinomiális lesz, ha a listák mérete polinomiális (n kisebb, mint a bemenet mérete, hiszen az $n+1$ számból áll). Figyeljük meg, hogy a listák legkisebb eleme a 0, legnagyobb eleme pedig legfeljebb t, továbbá hogy a listákban ritkítás után két szomszédos elem aránya (nagyobbik osztva a kisebbikkel) legalább $1+\frac{\varepsilon}{2n}$, ellenkező esetben ugyanis a nagyobbik elemet a ritkítás során töröltük volna. Bármely lista i. eleme így legalább $(1+\frac{\varepsilon}{2n})^{i-2}$. Legyen az adott lista legnagyobb eleme m és legyen ez az i. elem a listában. Ekkor $(1+\frac{\varepsilon}{2n})^{i-2} \leq m$ és $m \leq t$ (hiszen egyetlen elem sem lehet nagyobb t-nél), így

$$(1+\frac{\varepsilon}{2n})^{i-2} \leq m \leq t,$$

ahonnan mindkét oldal $1+\frac{\varepsilon}{2n}$ alapú logaritmusát véve

$$i-2 \leq \log_{1+\frac{\varepsilon}{2n}} t.$$

A logaritmus-függvények jól ismert $\log_a b = \frac{\log_c b}{\log_c a}$ azonosságát felhasználva (az $a = 1+\frac{\varepsilon}{2n}$, $b = t$, $c = e$ szereposztásban) innen

$$i-2 \leq \frac{\ln t}{\ln(1+\frac{\varepsilon}{2n})}.$$

A listaméretek felső becsléséhez most az alábbi lemmára lesz szükségünk.

5.5. Lemma *$x \geq 0$ esetén* $\ln(1+x) \geq \frac{x}{1+x}$.

Bizonyítás. Ez a lemma analízisből már valószínűleg ismerős, de a teljesség kedvéért bizonyítsuk be. Vizsgáljuk az $\ln(1+x) - \frac{x}{1+x}$ függvényt, erről kéne belátnunk, hogy $x \geq 0$ esetén az értéke nem negatív. A függvény a 0-ban 0-t vesz fel, elég tehát belátnunk, hogy a a $[0,\infty)$ intervallumban monoton növő. Ehhez pedig az elegendő, ha megmutatjuk, hogy a deriváltja minden $x \geq 0$ esetén pozitív. A derivált (néhány jól ismert deriválási szabályt alkalmazva) $\frac{1}{1+x} - \frac{1}{(1+x)^2}$, ami nyilván pozitív, hiszen $\frac{1}{1+x}$ nem nagyobb, mint 1. □

Alkalmazzuk most az 5.5. Lemmát $x = \frac{\varepsilon}{2n} > 0$-ra: $\ln(1+\frac{\varepsilon}{2n}) \geq \frac{\frac{\varepsilon}{2n}}{1+\frac{\varepsilon}{2n}}$. Ezt, és $\frac{\varepsilon}{2n}$ pozitivitását felhasználva a korábbi, i-re vonatkozó becslés a következőképp alakul:

$$i-2 \leq \frac{\ln t}{\ln(1+\frac{\varepsilon}{2n})} \leq \frac{\ln t}{\frac{\frac{\varepsilon}{2n}}{1+\frac{\varepsilon}{2n}}} = \frac{2n(1+\frac{\varepsilon}{2n})\ln t}{\varepsilon}.$$

Így egyetlen listában sem lehet olyan elem, melynek sorszáma nagyobb, mint $\frac{2n(1+\frac{\varepsilon}{2n})\ln t}{\varepsilon} + 2$, vagyis minden lista legfeljebb ennyi elemű lehet. Ez a függvény pedig polinomiális lesz a bemenet méretében és $\frac{1}{\varepsilon}$-ban is (az utóbbi nyilvánvaló, az előbbinél használjuk, hogy $2n(1+\frac{\varepsilon}{2n}) \leq 2n+1$ és $\ln t < \log t$, ahol n és $\log t$ is nyilván kisebb, mint a bemenet mérete).

5.2. Feladatok

38. **(1)** Döntsük el, hogy az alábbi kétváltozós függvények közül melyek polinomiálisak mindkét változóban.
(a) $f(x,y) = x^{\log y}$ (b) $f(x,y) = (xy)^{\log xy}$ (c) $f(x,y) = 2^{\log xy + \sqrt{\log y^x}}$

39. **(1)** Egy approximációs séma lépésszáma $f(n,\varepsilon)$, ahol n a bemenet mérete. Döntsük el, hogy az alábbiak közül melyik esetben lesz az approximációs séma polinomiális, illetve teljesen polinomiális.
(a) $f(n,\varepsilon) = \varepsilon n^{\varepsilon}$ (b) $f(n,\varepsilon) = \frac{n^2 \log(\frac{n}{\varepsilon^2} + \frac{n}{\varepsilon})}{\varepsilon^3}$ (c) $f(n,\varepsilon) = \sqrt[\varepsilon]{n}$

40. **(1)** Hajtsuk végre és dokumentáljuk a részösszeg problémára tanult teljesen polinomiális approximációs sémát az alábbi bemenetre.

$$2,6,7,14,28,44; \quad t = 49; \quad \varepsilon = \frac{1}{2}$$

41. **(2)** Mutassuk meg, hogy a részösszeg probléma polinom időben megoldható, ha az $a_1, a_2, \ldots, a_n, t$ bemenetre $t \leq \log a_1 a_2 \ldots a_n$.

42. **(2)** Mutassuk meg, hogy a részösszeg feladatra látott approximációs séma futtatásakor minden olyan s részösszeghez, mely az $a_1, a_2, \ldots, a_n$ számokból előállítható, létezik egy olyan s' részösszeg, mely az L_n listában benne van és $s' \geq \frac{s}{(1+\frac{\varepsilon}{2n})^n}$.

6. fejezet

Az ütemezéselmélet alapjai

6.1. Ütemezési problémák

Az ütemezési feladatokban bizonyos paraméterekkel rendelkező $J_1, J_2, \ldots, J_n$ *munkákat* (*job*[1]) kell adott *gépek* (*machine*) segítségével úgy elvégeznünk, hogy a megoldás valamilyen (természetesen szintén adott) célfüggvény szerint optimális legyen. Minden gépre igaz, hogy egyszerre egy munkát tud végezni és hogy ha egy adott munkán elkezd dolgozni, akkor azt nem szakíthatja meg, vagyis addig kell folytatnia a munkát, míg az el nem készül. A munkákat teljes egészükben egy gépen kell elvégezni (ami persze az egyes munkák esetében lehet különböző, de egy munka csak egy gépen készülhet).

Minden munkának adott a hossza, ütemezéselméleti szakszóval a *megmunkálási ideje* (*processing time*). A J_i munka megmunkálási idejét p_i-vel jelöljük. Adottak lehetnek ezen kívül még *rendelkezésre állási idők* (*release time*), vagyis azok az időpontok, amikor az egyes munkák elkezdhetők; *határidők* (*due date*); *súlyok* (*weight*), amik az egyes munkák fontosságát mutatják meg; és persze még sok egyéb paraméter is, de jelen jegyzetben ezek egyikével sem fogunk foglalkozni. Fontosak lesznek viszont az ún. *megelőzési feltételek* (*precedence constraints*), melyek a munkák bizonyos párjaira azt szabályozzák, hogy a pár melyik tagját kell előbb elvégezni. Valamivel precízebben: ha J_i és J_k két munka, akkor a (J_i, J_k)-val jelölt megelőzési feltétel azt jelenti, hogy a J_k munkát csak akkor tudjuk megkezdeni, ha a J_i-vel már végeztünk. A megelőzési feltételeket a legegyszerűbb egy irányított gráfban tárolni: a gráf csúcsai a munkák, a (J_i, J_k) megelőzési feltételnek pedig a J_i-ből J_k-ba menő irányított él felel meg. Az így kapott gráfot nevezzük a megelőzési feltételek gráfjának, vagy rövidebben *precedenciagráfnak*. A precedenciagráf nem tartalmazhat irányított kört, ellenkező esetben az

[1] Az ütemezéselméleti szakirodalom döntő részben angol nyelvű, ezért a fogalmak angol neveit is megadjuk.

ütemezési feladatnak semmilyen megoldása nem lenne.

A gépek különböző képességekkel rendelkezhetnek, jelen jegyzetben azonban csak olyan ütemezési problémákkal foglalkozunk, melyekben egyforma (avagy *párhuzamos*) gépek (*parallel machines*) szerepelnek.

Egy ütemezés minden munkához megadja, hogy azt melyik gépen, mettől meddig végezzük el (az időt a 0 időpillanattól számítjuk). Az ütemezések tehát tulajdonképpen függvények, melyek a munkákhoz hozzárendelik a munkát végző gép sorszámát és a munka kezdési idejét (a *befejezési idő* (*completion time*) automatikusan a kezdési idő és a megmunkálási idő összege lesz). Egy adott ütemezés során a J_i munka befejezési idejét C_i-vel jelöljük. Ne feledkezzünk el róla a későbbiekben, hogy a C_i számok nem tartoznak a probléma bemenetei közé, azok egy bizonyos ütemezéshez tartoznak (mivel mindig világos lesz, hogy melyik ütemezésről van szó, ezt külön nem is jelöljük).

Az optimalizálni kívánt célfüggvények közül csak a két leggyakoribbal foglalkozunk. Az egyik ezek közül az utolsónak elkészülő munka befejezési ideje, tehát $\max C_i$, ezt C_{max} jelöli, a neve pedig *teljes átfutási idő* (*makespan*). A másik az *átlagos átfutási idő*, vagyis $\frac{\sum C_i}{n}$, ami persze ekvivalens azzal, mintha $\sum C_i$ lenne a célfüggvény, hiszen n (a gépek száma) a bemenetben megadott konstans. Természetesen mindkét célfüggvényt minimalizálni szeretnénk.

Az ütemezési problémák tömör leírásához érdemes bevezetni az alábbi jelölést. Minden problémánál három mezőben adjuk meg az adatokat, a mezőket függőleges vonalakkal választjuk el. Az első mezőben szerepel, hogy hány és milyen gépet használunk: ha csak egy gépünk van, akkor ide egyszerűen 1-et írunk, ha több, akkor P jelöli az ismeretlen számú párhuzamos gépet (a gépek száma ekkor a bemenet része), Pm pedig azt, ha m párhuzamos gépünk van. A második mezőben szerepel, hogy az átfutási időkön kívül milyen egyéb paraméterek vannak megadva (pl. határidők, súlyok), de az egyes paraméterekre vonatkozó többletinformációk is itt szerepelnek (pl. $p_i = 1$ jelentése: minden átfutási idő egységnyi). Ha precedenciagráf is adott a problémához, azt (ebben a mezőben) $prec.$ jelöli, ekkor a gráfot természetesen külön meg kell adni. Végül a harmadik mezőben adjuk meg a célfüggvényt, esetünkben ez vagy C_{max} vagy $\sum C_i$ lesz. Így például $P2|prec.|C_{max}$ jelöli azt a problémát, ahol két egyforma gépen kell ütemeznünk úgy, hogy a $p_1, p_2, \ldots, p_n$ megmunkálási időkön kívül precedenciagráf is adott, a minimalizálandó célfüggvény pedig $C_{max} = \max C_i$. Az $1||\sum C_i$ jelentése: egy gépen ütemezünk, az átfutási időkön kívül semmilyen további adat nincs megadva, a célfüggvény pedig $\sum C_i$.

6.2. Egygépes ütemezések

Bemelegítés gyanánt megvizsgálunk néhány egyszerű, egygépes ütemezési feladatot. A legkönnyebb az $1||C_{max}$ probléma, itt optimális megoldást kapunk, ha a munkákat tetszőleges sorrendben, szünetek nélkül tesszük fel az egy szem gépre (ezt meg is tudjuk tenni, hiszen semmilyen paraméter nincs megadva, ami megakadályozná), a teljes átfutási idő egyszerűen $\sum p_i$. Ez a megoldás nyilván optimális, hiszen csak egy gépünk van és minden munkát el kell végezni, tehát bármely ütemezés legalább $\sum p_i$ időt igényel.

Valamivel nehezebb az $1|prec.|C_{max}$ probléma, itt a precedenciák miatt nem tudjuk tetszőleges sorrendben ütemezni a munkákat. Szerencsére azért van jó sorrend: mivel a precedenciagráfban nincs irányított kör, létezik a csúcsainak topologikus rendezése, vagyis olyan sorrendje, melyben minden él a kisebb sorszámú csúcs felől mutat a nagyobb sorszámú felé. A munkákat ebben a sorrendben ütemezve (szünet nélkül) itt is $\sum p_i$ teljes átfutási idejű, vagyis optimális megoldást kapunk. Ahhoz, hogy az ütemezést polinom időben megadjuk, szükségünk van a topologikus sorrend polinom idejű előállítására (és ez nyilván elég is). Ehhez a legegyszerűbb az irányított gráf egy 0 kifokú csúcsát venni elsőnek, ezt a csúcsot elhagyni, majd a maradék gráfra megismételni az eljárást, hogy megkapjuk a sorrendben második csúcsot, s.í.t. (irányított kör nélküli (véges) irányított gráfban mindig van 0 kifokú csúcs, mert bármely leghosszabb irányított út kezdőpontja ilyen). Ez az eljárás nyilván polinomiális lesz, bár ismert gyorsabb módszer is a topologikus sorrend előállítására: a gráfon végrehajtott tetszőleges mélységi bejárás során a befejezési számok szerinti fordított sorrend ilyen lesz. A mélységi keresésről és a topologikus sorrendről bővebben is esett szó a Bevezetés a Számításelméletbe 2 tárgy keretei közt, erről kitűnő összefoglalót találunk a www.cs.bme.hu/bsz2/dfs.pdf címen.

Vizsgáljuk most meg az $1||\sum C_i$ problémát. Nyilván itt is szünet nélkül érdemes ütemezni, de a sorrend megállapítása már nem teljesen magától értetődő, érezhető azonban, hogy minél rövidebb egy munka, annál hamarabb érdemes elvégezni. A munkák egy $J_1, J_2, \ldots, J_n$ sorrendjét SPT-sorrendnek (SPT: Shortest Processing Time) nevezzük, ha $p_1 \leq p_2 \leq \ldots \leq p_n$. Az SPT-sorrend megállapítása nyilván megy polinom időben, most lássuk be, hogy ebben a sorrendben ütemezve a feladat optimális megoldását kapjuk. Tegyük fel indirekten, hogy van olyan S optimális ütemezési sorrend, ami nem SPT-sorrend, legyen ez $J_1, J_2, \ldots, J_n$. Ekkor létezik olyan $k \leq n-1$ pozitív egész, melyre $p_k > p_{k+1}$, ellenkező esetben $p_1 \leq p_2 \leq \ldots \leq p_n$ teljesülne, vagyis az S ütemezés — feltevésünkkel ellentétben — SPT-sorrendű lenne. Tekintsük most azt az S' ütemezést, mely S-től csak annyiban tér el, hogy a J_k és J_{k+1} munkák sorrendjét megcseréljük. Jelöljük a J_i munka befejezési idejét az S ütemezés esetén C_i-vel, az S' ütemezés esetén C'_i-vel. Könnyen látható, hogy

$C_i = C'_i$ minden $i = 1,2,\ldots,k-1$ és $i = k+1,k+2,\ldots,n$ esetén is. Teljesül továbbá, hogy $C_k = C_{k-1} + p_k$, $C_{k+1} = C_{k-1} + p_k + p_{k+1}$ és $C'_{k+1} = C_{k-1} + p_{k+1}$, $C'_k = C_{k-1} + p_{k+1} + p_k$, ahonnan $\sum_{i=1}^n C'_i = p_{k+1} - p_k + \sum_{i=1}^n C_i < \sum_{i=1}^n C_i$, ami ellentmondás, hiszen ekkor az S' ütemezés jobb lenne, mint az S, amit pedig optimálisnak választottunk. Az SPT-sorrendű ütemezés tehát valóban az $1||\sum C_i$ probléma optimális megoldását adja.

6.3. Többgépes ütemezések

A megvizsgált egygépes ütemezések mindegyikénél sikerült polinom időben optimális ütemezést találnunk. A többgépes ütemezések esetében korántsem ilyen jó a helyzet, itt már közelítő algoritmusokra lesz szükség. Kezdjük a többgépes feladatok vizsgálatát a viszonylag egyszerűnek tűnő $P2||C_{max}$ feladattal. Esett már szó az NP-teljes PARTÍCIÓ problémáról, ahol egész számok egy halmazáról kellett eldöntenünk, hogy kettéosztható-e úgy, hogy a két részben a számok összege azonos legyen. Könnyen látható, hogy ha a $P2||C_{max}$ feladatot meg tudnánk oldani polinom időben, akkor a PARTÍCIÓ problémára is igaz lenne ugyanez. Legyen ugyanis $a_1,a_2,\ldots,a_n$ a PARTÍCIÓ probléma egy bemenete és oldjuk meg a $P2||C_{max}$ feladatot az $a_1,a_2,\ldots,a_n$ megmunkálási idők mellett. Az $a_1,a_2,\ldots,a_n$ halmaz pontosan akkor lesz két azonos összegű részre szétoszthető, ha a kapott megoldás teljes átfutási ideje $\frac{\sum_{i=1}^n a_i}{2}$. A $P2||C_{max}$ probléma tehát NP-nehéz, és így persze NP-nehéz az ennél általánosabb $P||C_{max}$ probléma is.

Probléma. $P||C_{max}$.

Bonyolultság. A gépek száma a bemenet része, elképzelhető tehát, hogy két gépünk van, amikor is az NP-nehéz $P2||C_{max}$ problémát kell megoldanunk, tehát a $P||C_{max}$ is NP-nehéz. Az alábbiakban leírjuk a listás ütemezés nevet viselő $(2-\frac{1}{m})$-approximációs algoritmust (m a gépek száma), mely Graham nevéhez fűződik, majd ennek egy finomított változatát, melynek approximációs faktora $\frac{4}{3} - \frac{1}{3m}$

$(2-\frac{1}{m})$-approximációs algoritmus. Maga az algoritmus rendkívül egyszerű: a munkákból egy listát készítünk (innen a név), majd amint van szabad gép, arra azonnal feltesszük a listában soron következő munkát (ha egyidejűleg több szabad gép is van, akkor használjuk – mondjuk – a legkisebb sorszámút).

Elemzés. Az eljárás polinomialitása nyilvánvaló, feladatunk az approximációs faktor vizsgálata. Ehhez először is belátunk két egyszerű állítást a teljes átfutási idő optimumával kapcsolatban (amit szokás szerint OPT-tal jelölünk).

6.1. Állítás *a)* $OPT \geq \frac{\sum_{i=1}^n p_i}{m}$, *b)* $OPT \geq \max_i p_i$.

Bizonyítás. Figyeljük meg, hogy az *OPT* időpillanatban az összes munkának készen kell lennie. Ebből azonnal következik a b) állítás, figyelembe véve, hogy minden megkezdett munkát ugyanazon a gépen kell elvégezni, ahol elkezdtük. Sőt, a megfigyelésből az a) állítás is következik, hiszen ha – az a) állítással ellentétben – $OPT < \frac{\sum_{i=1}^{n} p_i}{m}$ teljesülne, akkor az *OPT* időpillanatban elvégzett munkák összhossza (mivel m gépünk van) kisebb lenne, mint $\sum_{i=1}^{n} p_i$. □

Legyen a listás ütemezés során az utolsónak befejeződő munka J_k és legyen J_k elkezdésének pillanata t, ekkor a teljes átfutási idő $C_{max} = t + p_k$. A listás ütemezés során amíg van munka, addig minden gépnek folyamatosan dolgoznia kell, így a t időpillanatig mind az m gép folyamatosan dolgozott (hiszen a t pillanatig biztosan volt elvégezhető munka, nevezetesen J_k). A t pillanatig az m darab gépen végzett munka összhossza tm, így $tm \leq \sum_{i \neq k} p_i$, vagyis $t \leq \frac{\sum_{i \neq k} p_i}{m}$. Innen

$$C_{max} = t + p_k \leq \frac{\sum_{i \neq k} p_i}{m} + p_k = \frac{\sum_{i=1}^{n} p_i}{m} + \left(1 - \frac{1}{m}\right) p_k$$

$$\leq OPT + \left(1 - \frac{1}{m}\right) OPT = \left(2 - \frac{1}{m}\right) OPT,$$

ahol az utolsó egyenlőtlenség a 6.1. Állítás a), illetve b) része miatt teljesül. Ezzel az approximációs faktort igazoltuk.

Éles példa. Legyen m gépünk, $n = m(m-1)+1$ munkánk, a munkák hosszai pedig legyenek $p_1 = p_2 = \ldots = p_{m(m-1)} = 1$, $p_{m(m-1)+1} = m$. Ha a lista utolsó eleme $J_{m(m-1)+1}$ (amit semmi sem zár ki), akkor az első $m(m-1)$ munkát az m gép $m-1$ időegység alatt végzi el (mindegyik gép $m-1$ darab 1 hosszú munkát végez el), végül az első gép további m időegység alatt elvégzi az utolsó munkát is, ekkor a listás ütemezés teljes átfutási ideje $C_{max} = 2m-1$. Könnyen látható ugyanakkor, hogy $OPT = m$: az egyik gép m időegység alatt elvégzi a $J_{m(m-1)+1}$ munkát, a többi $m-1$ gép pedig ugyanennyi idő alatt az $m(m-1)$ darab 1 hosszú munkát. A listás ütemezés teljes átfutási ideje tehát az optimum $\frac{2m-1}{m} = \left(2 - \frac{1}{m}\right)$-szerese.

A bizonyításból és az éles példából is látható, hogy a kellemetlenséget az okozza, ha (a többihez képest) hosszú munkát későn kezdünk el. Ha a hosszú munkákat előrevesszük, akkor lényegesen jobb approximációs faktort is elérhetünk.

$(\frac{4}{3} - \frac{1}{3m})$-approximációs algoritmus. A munkák egy $J_1, J_2, \ldots, J_n$ sorrendjét LPT-sorrendnek (LPT: Longest Processing Time) nevezzük, ha $p_1 \geq p_2 \geq \ldots \geq p_n$. Tegyük a munkákat egy listába az (egyik) LPT-sorrend szerint, majd ütemezzünk listásan e lista szerint.

Az eljárás nyilván polinomiális, az approximációs faktor bizonyítását nem részletezzük, megmutatjuk azonban, hogy a faktor nem jobb, mint $\frac{4}{3} - \frac{1}{3m}$.

Éles példa. Legyen m gépünk ($G_1, G_2, \ldots, G_m$), $n = 2m+1$ munkánk, a munkák hosszai pedig legyenek

$$p_1 = p_2 = 2m-1,\ p_3 = p_4 = 2m-2,\ p_5 = p_6 = 2m-3,\ \ldots\ ,\ p_{2m-1} = p_{2m} = m$$

és $p_{2m+1} = m$. Ekkor $J_1, J_2, \ldots, J_{2m+1}$ LPT-sorrend. Ebben a sorrendben listásan ütemezve, a teljes átfutási idő $4m-1$ lesz. Ennek belátásához azt az esetet vizsgáljuk meg részletesen, amikor m páros, páratlan m esetén a bizonyítás hasonlóan gondolható végig. A 0 időpillanatban a G_i gépre J_i kerül $i = 1, 2, \ldots, m$ esetén. A gépek közül elsőként G_{m-1} és G_m végez, hiszen ezeken vannak a legrövidebb munkák: $p_{m-1} = p_m = \frac{3}{2}m$. A következő két munka (J_{m+1} és J_{m+2}), melyek hossza $\frac{3}{2}m-1$, tehát erre a két gépre kerül. Ezt követően a G_{m-3} és G_{m-2} gépek készülnek el a munkájukkal (a $\frac{3}{2}m+1$ időpillanatban), ezekre kerül a két $\frac{3}{2}m-2$ hosszú munka (J_{m+3} és J_{m+4}). A $G_{m-3}, G_{m-2}, G_{m-1}$ és G_m gépek tehát a második munkával a $3m-1$ időpillanatban készülnek el. Könnyen ellenőrizhető, hogy ez az összes gépre igaz lesz: amennyivel hosszabb munkát kap egy gép elsőként G_m-hez képest, épp annyival rövidebb a második munkája. Tehát az m gép az első $2m$ munkával egyszerre végez a $3m-1$ időpillanatban, így az utolsó munkát az első gép kapja és a $4m-1$ időpillanatban fejezi be, valóban ez lesz tehát a teljes átfutási idő. Könnyű belátni ugyanakkor, hogy $OPT = 3m$: tegyük $i = 1, 2, \ldots, m-1$ esetén a G_i gépre a J_i és a J_{2m-1-i} munkákat (szünet nélkül), G_m-re pedig az utolsó három munkát, vagyis J_{2m-1}-et, J_{2m}-et és J_{2m+1}-et. Ekkor minden munkát elvégeztünk és minden gép a $3m$ időpillanatban végez a munkáival, a teljes átfutási idő tehát ekkor csakugyan $3m$ és persze ennél kevesebb nem is lehet (pl. a 6.1. állítás a) része miatt). Az approximációs faktor tehát csakugyan nem jobb, mint $\frac{4m-1}{3m} = \frac{4}{3} - \frac{1}{3m}$.

Probléma. $P|prec.|C_{max}$.

Bonyolultság. Ha a precedenciagráf üres (vagyis csak izolált pontokból áll), a $P||C_{max}$ problémát kapjuk, ez tehát speciális esete a $P|prec.|C_{max}$-nak, így természetesen az utóbbi is NP-nehéz. Szerencsére a listás ütemezés alkalmas módosítása $\left(2 - \frac{1}{m}\right)$-approximációs algoritmus lesz.

$(2-\frac{1}{m})$-approximációs algoritmus. A listás ütemezést annyiban módosítjuk csak, hogy az adott pillanatban elvégezhető munkák közül tesszük fel a legkisebb sorszámú elérhető gépre a listában legelöl álló munkát (ezt az algoritmust is listás ütemezésnek hívjuk). Ilyenkor már elképzelhető, hogy egy gép áll, pedig van még elvégzendő munka, de ennek oka csak az lehet, hogy az összes hátralévő munka olyan, hogy csak valamelyik még el nem végzett munka befejezése után kezdhető el.

Elemzés. Az eljárás polinomialitása nyilvánvaló, az approximációs faktor vizsgálata viszont meghaladja a jegyzet kereteit. Az, hogy az approximációs faktor nem

jobb, mint $2-\frac{1}{m}$, következik abból, hogy ez már a $P||C_{max}$ speciális esetre is igaz volt, most azonban azt is megmutatjuk, hogy az approximációs faktor most akkor sem jobb, mint $2-\frac{1}{m}$, ha a lista LPT-sorrendű.

Éles példa. A $P||C_{max}$ problémához adott listás ütemezés éles példáját módosítjuk minimálisan. Legyen m gépünk, $n = m(m-1)+2$ munkánk, a munkák hosszai pedig legyenek

$$p_1 = m - \frac{1}{2},\ \ p_2 = p_3 = \ldots = p_{m(m-1)+1} = 1,\ \ p_{m(m-1)+2} = \frac{1}{2}.$$

A precedenciagráf tartalmazzon egyetlen irányított élet, mely mutasson $J_{m(m-1)+2}$-ből J_1-be. Ekkor a $J_1, J_2, \ldots, J_{m(m-1)+2}$ lista LPT-sorrendű és evvel a listával használva az algoritmust, a teljes átfutási idő $2m-1$ lesz. Ennek az az oka, hogy a J_1 munkát csak a $J_{m(m-1)+2}$ után végezhetjük el, így az 1 időigényű munkákat fogják a gépek először elvégezni: az $m(m-1)$ munkát az m gép $m-1$ időegység alatt. Ezt követi a $J_{m(m-1)+2}$ munka, majd ennek elvégzése után a J_1, a teljes átfutási idő tehát csakugyan $C_{max} = 2m-1$. Most is könnyen látható ugyanakkor, hogy $OPT = m$: az egyik gép m időegység alatt elvégzi a $J_{m(m-1)+2}$, majd a J_1 munkát, a többi $m-1$ gép pedig ugyanennyi idő alatt az $m(m-1)$ darab 1 hosszú munkát. A listás ütemezés teljes átfutási ideje tehát itt is az optimum $\frac{2m-1}{m} = \left(2-\frac{1}{m}\right)$-szerese.

Az iménti éles példa jól mutatja, hogy a $P|prec.|C_{max}$ probléma esetében nem akkor érdemes előrevenni egy munkát, ha a megmunkálási ideje nagy, hanem sokkal inkább akkor, ha a precedenciagráfban hosszú irányított út vezet ki belőle (a hosszút úgy értve, hogy az úton lévő csúcsokhoz tartozó munkák összhossza nagy).

Algoritmus (listás ütemezés a leghosszabb út szerint). Állapítsuk meg minden munkáról, hogy a precedenciagráfban belőle kiinduló irányított utakon mennyi az úton szereplő munkák összhosszának maximuma, majd az így kapott érték (leghosszabb út hossza) szerint csökkenő lista alapján ütemezzünk listásan. A csúcsokból induló leghosszabb út hosszát nem teljesen magától értetődő polinom időben kiszámítani, hiszen egy csúcsból nagyon sok (nem polinomiális mennyiségű) irányított út is kiindulhat. Ismert azonban, hogy irányított élsúlyozott aciklikus gráfban polinom időben meg lehet találni bármely csúcsból a belőle kivezető legrövidebb és leghosszabb utakat is a topologikus sorrend segítségével, erről is ld. a www.cs.bme.hu/bsz2/dfs.pdf anyagot. A mi esetünkben nem az élek, hanem a csúcsok vannak súlyozva (a precedenciagráf csúcsai a munkák és ezeknek van hossza), de ettől a helyzet csak egyszerűbb lesz. Először is keressük meg a csúcsok egy topologikus sorrendjét, legyen ez $J_1, J_2, \ldots, J_n$. A J_n-ből induló utak maximális hossza persze p_n, hiszen J_n-ből nem indul él. Tegyük fel most, hogy a $J_{k+1}, J_{k+2}, \ldots, J_n$ csúcsokból induló leghosszabb utak $s_{k+1}, s_{k+2}, \ldots, s_n$ hosszait

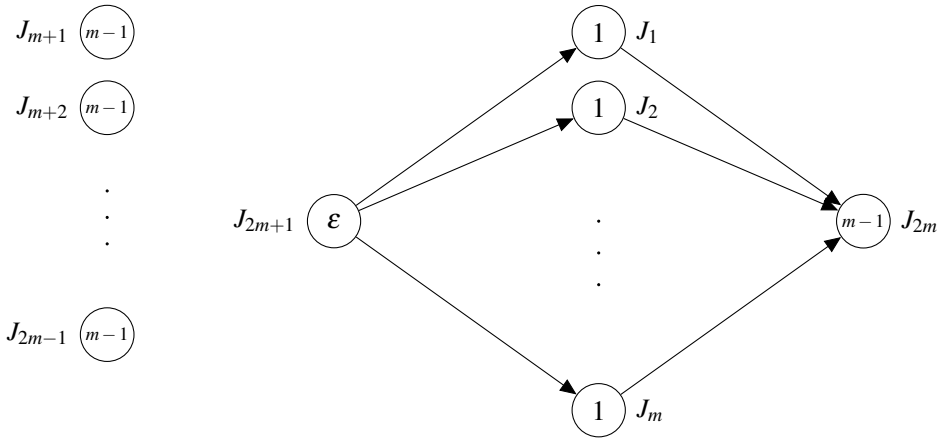
már ismerjük. Jelöljük a J_k-ból induló élek végpontjainak halmazát X-szel, ekkor persze $X \subseteq \{J_{k+1}, J_{k+2}, \ldots, J_n\}$, hiszen $J_1, J_2, \ldots, J_n$ topologikus sorrend. Ekkor a J_k-ból induló leghosszabb út hossza nyilván $p_k + \max_{i \in X} s_i$. Ezzel a módszerrel sorban a $J_{n-1}, J_{n-2}, \ldots, J_1$ csúcsokból vezető leghosszabb utak hosszait ki tudjuk számítani.

Elemzés. Foglalkozzunk először az algoritmus polinomialitásával. A topologikus sorrendet (mint már láttuk) meg tudjuk találni polinom időben, a leghosszabb utak hosszának számítása során az egyes csúcsoknál legfeljebb n szám közül kell maximumot választanunk, a teljes számítás tehát legfeljebb n^2-tel arányos lépésszámot igényel (valójában a precedenciagráf élszámával arányos lépést végzünk csak, de ennek most nincs jelentősége). A leghosszabb utak hosszainak ismeretében a munkákat ezek szerinti csökkenő sorrendben kell listásan ütemezni, ez nyilván megy polinom időben.
Az approximációs faktorról tudjuk, hogy nem rosszabb, mint $2 - \frac{1}{m}$, hiszen *bármely* listás ütemezés ilyen faktorú approximációt ad. A leghosszabb út szerinti listás ütemezés ennél gyakran sokkal jobb, nem ritkán optimális eredményt is adhat. Erről R. L. Graham: Combinatorial scheduling theory c. 1978-as összefoglalójában (melyet megtalálunk pl. a Rendszeroptimalizálás könyv 4.7. fejezetében) szerepel egy igen szórakoztató példa. Az approximációs faktor mindezek ellenére itt sem jobb, mint $2 - \frac{1}{m}$, mint azt az alábbi példa mutatja.

Éles példa. Legyen m gépünk és $2m+1$ munkánk, melyek hosszai legyenek

$$p_1 = p_2 = \ldots = p_m = 1, \;\; p_{m+1} = p_{m+2} = \ldots = p_{2m} = m-1, \;\; p_{2m+1} = \varepsilon > 0.$$

(ε pontos értékéről majd később lesz szó). A precedenciagráf legyen a következő (ld. az alábbi ábrát is, ahol a csúcsokba írt számok a munkák hosszai). A csúcsok persze a munkák, él vezet J_{2m+1}-ből a $J_1, J_2, \ldots, J_m$ csúcsokba és a $J_1, J_2, \ldots, J_m$ csúcsokból a J_{2m} csúcsba.



Az algoritmus a 0 időpillanatban az első gépre a J_{2m+1} munkát teszi, a többi gépre pedig a $J_{m+1}, J_{m+2}, \ldots, J_{2m-1}$ munkákat. ε-t kellően kicsinek választva (erről még lesz szó, egyelőre legyen kisebb, mint 1) persze az első gép végez először és elkezdi az 1 hosszú munkákat, melyek közül $m-1$ darab kerül fel rá, mielőtt a többi gép elkészülne az első munkájával az $m-1$ időpillanatban. Az utolsó 1 hosszú munka így már a második gépre kerül, mely az m időpillanatban végez vele. Csak ekkor lehet elkezdeni a J_{2m} munkát (mely az első gépre fog kerülni, de ennek nincs jelentősége, ekkor már minden gép szabad lesz), a teljes átfutási idő így $2m-1$. Nem nehéz ugyanakkor megadni egy $m+\varepsilon$ átfutási idejű ütemezést: először végezzük el az ε költségű munkát az egyik gépen, addig a többi gép ne csináljon semmit, majd tegyük fel az m gépre a most már elvégezhető m darab 1 hosszú munkát, végül ha ezek kész vannak, akkor feltehetjük az m gépre az m darab $m-1$ hosszú munkát is, melyek ekkor már mind elvégezhetők. Ez azt mutatja, hogy a leghosszabb út szerinti (sőt, igazából bármilyen sorrendű) listás ütemezés approximációs faktora nem jobb, mint $\frac{2m-1}{m+\varepsilon}$. Mivel ε-t tetszőlegesen kicsi pozitív számnak választhatjuk, semmilyen $\left(2-\frac{1}{m}\right)$-nél kisebb szám sem lehet jó approximációs faktornak.

Van ugyanakkor olyan speciális esete a $P|prec.|C_{max}$ problémának, amikor a leghosszabb út szerinti listás ütemezés optimális megoldást ad: ilyen lesz a $P|prec. = in\text{-}tree, p_i = 1|C_{max}$ probléma. Ebben minden megmunkálási idő egységnyi, a precedenciagráf pedig egy ún. be-fenyő (in-tree), vagyis olyan irányított gráf, mely irányítatlan értelemben fa, és létezik egy olyan r csúcsa, melybe bármely csúcsból el lehet jutni irányított úton. Ebben a speciális esetben a leghosszabb út szerinti listás ütemezést Hu algoritmusának nevezik. Első ránézésre a $P|prec. = in\text{-}tree, p_i = 1|C_{max}$ túlságosan is speciálisnak tűnik, de jó tudni, hogy pl. a $P|prec., p_i = 1|C_{max}$ probléma még NP-nehéz.

Segítségek a feladatokhoz

2. A bemenet mérete $\lfloor \log a \rfloor + \lfloor \log b \rfloor + 2$.

3. A bemenet mérete $\lfloor \log x \rfloor + \lfloor \log y \rfloor + 2$.

4. A bemenet mérete $\lfloor \log a_1 \rfloor + \lfloor \log a_2 \rfloor + \ldots + \lfloor \log a_n \rfloor + n$.

5. Pl. $|A| = |B| = 3$ mellett próbáljunk olyan $G = (A, B, E)$ összefüggő páros gráfot megadni, melyre $\alpha(G) = 4$.

6. $k = 2$-re jó az 5. feladat gráfja, $k > 2$-re ehhez vegyünk további pontokat és éleket.

7. Ha G összefüggő páros gráf, akkor két jó 2-színezése lesz.

8. Vizsgáljuk meg azon részgráfokat, melyeket valamelyik két színosztályhoz tartozó csúcsok alkotnak.

9. A felosztás során kapott új csúcsok foka 2, így a eredeti gráfnak nem lehet legalább 3 fokú csúcsa (miért?).

10. Három csúcs elég.

11. Olyan csúcs, aminek a foka legfeljebb 5, jó lesz utolsónak (miért? és miért létezik ilyen?).

12. A mohó algoritmus jó lesz.

13. Éles példánál az élek fele kell, hogy keresztbe menjen.

14. Minden éles példa páros gráf kell legyen, találjuk meg először a 2 élűt ($k = 1$).

15. Minden éles példa páros gráf kell legyen, találjuk meg először a 4 élűt ($k = 1$).

16. Az algoritmus kimeneteként kapott kettéosztásra éles példa esetén bármely v csúcsra $d_m(v) = d_s(v)$ (miért?); vizsgáljuk meg a $\sum_{v \in A} d_s(v)$, $\sum_{v \in A} d_m(v)$, $\sum_{v \in B} d_m(v)$ és $\sum_{v \in B} d_s(v)$ összegeket, ahol A és B a kimenetként kapott páros gráf osztályai.

17. Használjuk a MAX_PÁROS-ra adott második algoritmust úgy, hogy a már elhelyezett csúcsok minden lépés után összefüggő részgráfot alkossanak.

18. A második algoritmus kimenete is lefogó ponthalmaz, ezért lefogja minden maximális párosítás éleit is.

19. Egy megfelelő méretű független élhalmaz jó lesz.

20. Mindkét algoritmus páros csúcsú kimenetet ad.

21. Nem; próbáljunk olyan G gráfot keresni, melyre $\tau(G) = 8$ és minden 4 élű párosítása bővíthető.

22. X akkor és csak akkor független ponthalmaz $G = (V, E)$-ben, ha $V \setminus X$ lefogó ponthalmaz.

25. A MIN_LEFOGÓ az S_HALMAZ_FED speciális esete.

26. Először ellenőrizzük, hogy metrikus-e a súlyozás.

27. Az első rész magától értetődő, a másodikhoz lássuk be, hogy az optimális Steiner-fa út kell legyen.

28. Az optimális Steiner-fa négyféle lehet aszerint, hogy a Steiner-pontok mely részhalmazát tartalmazza.

29. Az optimális Steiner-fa a Steiner-pontok egy S' részhalmazát fogja tartalmazni; hány ilyen S' létezik?

32. Használjuk a 34. oldalon látott visszavezetést, de kn helyett egy alkalmasan megválasztott számmal.

33. Az 1 súlyú élekből álló részgráf összefüggő.

34. Az 1 súlyú élek alkossanak csillagot.

35. Metrizáljunk úgy, mint a Steiner-fára vonatkozó approximációnál.

36. Ha S kiterjeszthető Euler-körsétává, akkor S éleit G-ből elhagyva összefüggő gráfot kell kapnunk; ez elég is lesz.

37. Hajtsunk végre egy mélységi keresést a fán és próbáljuk az élek mélységi sorrend szerint vett végigjárását és az éleken való visszalépéseket megfelelően összerakni.

41. A látott pontos algoritmus listái nem lehetnek túl hosszúak.

42. Próbálkozzunk teljes indukcióval.

A feladatok megoldásai

1. (a) $\frac{n^3}{3}+\frac{n^2}{2} \le n^3+n^2 \le 2n^3$, a polinomialitás definíciójában szereplő konstansokra tehát $c_1=2$, $c_2=3$ jó választás lesz, így a függvény polinomiális.
(b) $n\log n+3\sqrt{n} \le n^2+3n \le 4n^2$, itt tehát $c_1=4$, $c_2=2$ igazolja a polinomialitást (valójában persze $c_2=1{,}5$ is bőven elég és c_1-en is lehet finomítani, de ebben a feladatban erre nincs szükség).
(c) $3^{\log n}=(2^{\log 3})^{\log n}=2^{\log 3\log n}=(2^{\log n})^{\log 3}=n^{\log 3}$, itt tehát $c_1=1$, $c_2=\log 3$ igazolja a polinomialitást.
(d) $\sqrt{n}^{\log n}=n^{\frac{1}{2}\log n}$, amit semmilyen c_1 és c_2 konstansok mellett sem lehet felülről becsülni $c_1 n^{c_2}$-vel, hiszen

$$\frac{n^{\frac{1}{2}\log n}}{c_1 n^{c_2}}=\frac{n^{\frac{1}{2}\log n-c_2}}{c_1},$$

ami $n\to\infty$ esetén végtelenhez tart (holott 1-nél nem lehetne nagyobb), ez a függvény tehát nem polinomiális.
(e) $\log n^{n^2}=n^2\log n \le n^3$, tehát $c_1=1$, $c_2=3$ igazolja a polinomialitást.
(f) $\log n^{n^n}=n^n\log n$, ami nem polinomiális, hiszen tetszőleges c_1 és c_2 konstansok mellett $\frac{n^n\log n}{c_1 n^{c_2}}$ végtelenhez tart $n\to\infty$ esetén.

2. A bemenet mérete $\lfloor\log a\rfloor+\lfloor\log b\rfloor+2$. Az algoritmus $b-1$ összeadást végez, $b-1$ pedig tipikusan exponenciális a bemenet méretében, hiszen a bemenet mérete például $a\le b$ esetén nem nagyobb $(2\log b+2)$-nél. Az algoritmus így természetesen nem polinomiális.

3. A bemenet mérete $n=\lfloor\log x\rfloor+\lfloor\log y\rfloor+2$, így az (a) feladat lépésszáma polinomiális, hiszen

$$\log x^{\log xy}=\log xy\log x=(\log x+\log y)\log x\le n^2.$$

A (b) feladat lépésszáma is polinomiális, mivel

$$\log\log x^y=\log(y\log x)\le\log(yx)=\log x+\log y\le n.$$

A (c) feladat lépésszáma már nem polinomiális: például az $x = 2^k$, $y = 2^{k+1}$ választással a bemenet mérete $n = 2k+3$, a lépésszám pedig $(\log x)^{\log\log|x-y|} = k^{\log k}$, amit semmilyen c_1 és c_2 konstansok mellett sem lehet felülről becsülni $c_1 n^{c_2}$-vel, hiszen

$$\frac{k^{\log k}}{c_1 n^{c_2}} = \frac{k^{\log k}}{c_1(2k+3)^{c_2}} \geq \frac{k^{\log k}}{c_1(5k)^{c_2}} = \frac{k^{\log k - c_2}}{c_1 5^{c_2}},$$

ami végtelenhez tart, ha k végtelenhez tart.

4. A bemenet mérete $\lfloor\log a_1\rfloor + \lfloor\log a_2\rfloor + \ldots + \lfloor\log a_n\rfloor + n$, így az (a) feladat lépésszáma lineáris (sőt, legfeljebb akkora, mint a bemenet mérete), így persze polinomiális. Mivel

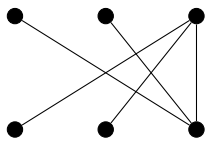
$$\log a_1 a_2 \ldots a_n = \log a_1 + \log a_2 + \ldots + \log a_n \leq \lfloor\log a_1\rfloor + \lfloor\log a_2\rfloor \ldots \lfloor\log a_n\rfloor + n,$$

ugyanez igaz a (c) feladat lépésszámára is. A (b) feladat lépésszáma azonban nem polinomiális, hiszen a_1 bármilyen nagy lehet n-hez képest, ugyanakkor a_n nem kell, hogy sokkal nagyobb legyen a_1-nél. Például az $a_1 = a_2 = \ldots = a_{n-1} = 2^n$, $a_n = 2^{n+1}$ választással a bemenet méretére

$$\lfloor\log a_1\rfloor + \lfloor\log a_2\rfloor + \ldots + \lfloor\log a_n\rfloor + n \leq n + (n-1)(n+1) + n < (n+1)^2$$

adódik, vagyis a bemenet méretében a_1 exponenciális. Ugyanez a példa mutatja, hogy a (d) és (e) feladatok lépésszámai is lehetnek exponenciálisak.

5. $|A| = |B| = 3$ mellett adunk olyan $G = (A, B, E)$ összefüggő páros gráfot, melyre $\alpha(G) = 4$. Mivel $\alpha(G) + \tau(G)$ a gráf csúcsszámával egyenlő (Gallai 1. tétele), jelen esetben $\tau(G) = 2$ kéne hogy legyen, vagyis a gráf élei lefoghatók két csúccsal. Az alábbi gráf ezt teljesíti és a többi feltételnek is megfelel.



6. Legyen H az 5. feladat megoldásában megadott gráf, G pedig az a gráf, melyet H-ból úgy kapunk, hogy hozzáveszünk $k-2$ új csúcsot, melyeket összekötünk H minden csúcsával és egymással is. G nyilván színezhető k színnel (2 szín elég H-hoz, $k-2$ a többi csúcshoz), másrészt az új csúcsoknak minden színezésben különböző színűeknek kell lenniük, és a színeiknek különbözniük kell a H-ban használt színektől is, így legalább k színre lesz szükség G színezéséhez. Sőt, k színnel is csak akkor színezhető G, ha a H részgráfot 2 színnel színezzük. Így $\chi(G) = k$ és egyetlen k-színezésben sincs olyan színosztály, mely a gráf maximális

független csúcshalmaza lenne, hiszen H 2-színezésében (ilyenből csak egy van, mivel H összefüggő) mindkét osztály 3 elemű, míg $\alpha(G) = 4$.

10. Legyen G az a gáf, amit egy 3 csúcsú teljes gráf (vagyis egy háromszög) éleinek megduplázásával kapunk (vagyis minden élet két párhuzamos éllel helyettesítünk). Ennek maximális foka 4, az élkromatikus száma viszont 6, hiszen a hat él közül bármelyik kettőnek van közös csúcsa. Érdemes megfigyelni, hogy ha a háromszög minden éle helyett k párhuzamos élet veszünk be, akkor a maximális fok $2k$, az élkromatikus szám $3k$ lesz (Shannon egy tétele szerint a $\frac{\chi_e}{\Delta}$ arány ennél nagyobb nem is lehet).

11. Ismert, hogy egy n csúcsú egyszerű síkgráfnak legfeljebb $3n-6$ éle van, így kell hogy legyen olyan csúcsa, melynek foka legfeljebb 5 (ellenkező esetben a gráf fokszámösszege, vagyis az élek számának kétszerese legalább $6n$ lenne). Válasszunk egy ilyen v csúcsot a sorrendünkben utolsónak, majd töröljük v-t és ismételjük meg az eljárást a maradék gráfra, így kapva az utolsó előtti csúcsot, és így tovább. Ha az így kapott sorrendben színezünk mohón, akkor minden csúcs színezésekor teljesül, hogy annak legfeljebb 5 szomszédja kapott már színt (hiszen legfeljebb 5 olyan szomszédja lehet, aki nála kisebb sorszámú), így 6 szín csakugyan elég lesz.

12. Színezzük az éleket mohón, tetszőleges sorrendben. Mivel minden élhez legfeljebb $2(\Delta(G)-1)$ másik él csatlakozik (az él mindkét végpontjában legfeljebb $\Delta(G)-1$), a színezés során soha nem lesz szükségünk $(2\Delta(G)-1)$-nél nagyobb sorszámú színre. Mivel az élkromatikus szám legalább $\Delta(G)$ és a mohó színezés polinom időben megvalósítható, valóban 2-approximációt kaptunk.

13. Mindkét algoritmusra igaz, hogy az éleknek legalább a fele keresztbe kell, hogy menjen. Ha ennél több menne keresztbe, akkor nem lenne éles a példánk, tehát az éleknek épp a fele megy keresztbe, így az élszám páros. Másrészt ha a bemenet nem páros gráf, akkor a maximális páros részgráfnak $|E|$-nél kevesebb éle van. Mivel az algoritmusok kimenetei legalább $\frac{|E|}{2}$ élet tartalmaznak, a kimeneti páros gráf élszáma ilyenkor nagyobb lenne, mint az optimum fele, vagyis a példa nem lenne éles.

14. Páros gráfot keresünk (a 13. feladat szerint), mégpedig $k = 1$-re 2 élűt; egy 2 élű (3 csúcsú) út jó lesz. Ha először a két szélső csúcsot helyezzük el, mégpedig különböző csoportokba (amit megtehetünk), akkor bárhova is kerül a középső csúcs, 1 élű megoldást kapunk. $k \geq 2$ esetén mindössze annyit kell tennünk, hogy a 3 csúcsú utat k példányban vesszük, a bizonyítás a fentihez hasonló.

15. Most is páros gráfot keresünk (a 13. feladat szerint), most $k = 1$-re 4 élűt: a 4 hosszú kör mindkét algoritmushoz jó lesz. Legyenek a kör csúcsai a,b,c,d, élei $(a,b),(b,c),(c,d),(d,a)$. Az első algoritmus esetében ha a kezdeti kettéosztás $A = \{a,b\}$, $B = \{c,d\}$, akkor az algoritmus áthelyezések nélkül leáll és az

optimális 4 élű páros részgráf (vagyis maga G) helyett 2 élűt talál. A második algoritmusnál legyen az elsőként elhelyezett csúcs a, a másodikként elhelyezett csúcs c, ezt bármelyik csoportba tehetjük, tegyük abba, amelyik nem tartalmazza a-t. Most b-t és d-t is bármelyik csoportba tehetjük, de bárhogy is helyezzük el őket, 2 élű megoldást kapunk. $k \geq 2$ esetén most is mindössze annyit kell tennünk, hogy a 4 hosszú kört k példányban vesszük, a bizonyítás a fentihez hasonló.

16. Az algoritmus kimeneteként kapott A, B halmazokra történő kettéosztásnál éles példa esetén bármely v csúcsra $d_m(v) = d_s(v)$. Ez azért igaz, mert ha lenne olyan v, amire $d_m(v) < d_s(v)$, akkor még nem állna le az algoritmus, míg $d_m(v) > d_s(v)$ esetén (mivel minden más x csúcsra $d_m(x) \geq d_s(x)$), a gráf éleinek több, mint a fele menne A és B között, s ekkor nem lenne éles a példánk. Jelöljük a gráf A-n belül menő éleinek számát e_A-val, B-n belül menő éleinek számát e_B-vel, végül az A és B közt menő élek számát e_{AB}-vel. Ekkor

$$2e_A = \sum_{v \in A} d_s(v) = \sum_{v \in A} d_m(v) = e_{AB} = \sum_{v \in B} d_m(v) = \sum_{v \in B} d_s(v) = 2e_B.$$

A második és az ötödik egyenlőség $d_m(v) = d_s(v)$ következménye, a többi magától értetődő. Innen a gráf éleinek száma $e_A + e_B + e_{AB} = e_A + 2e_A + e_A = 4e_A$, ahonnan a kívánt állítás adódik.

17. A gráf csúcsait sorban, egyesével két osztályba soroljuk a második algoritmus szerint, azaz egy újonnan érkező csúcsot abba az osztályba helyezünk, ahol kevesebb (nem több) szomszédja van. Láttuk, hogy ez az algoritmus 2-approximációs. Ha a csúcsokat olyan sorrendben vesszük, hogy az aktuálisan már elhelyezett csúcsok mindig összefüggő részgráfot alkossanak, akkor az egy osztályba kerülő csúcsok közt lesz (páros hosszú) út a gráfban, így semelyik kettő nem lehet szomszédos, ekkor ugyanis lenne a gráfban páratlan kör, ami páros gráfban lehetetlen. Ebben a sorrendben véve tehát a csúcsokat, minden él bekerül a páros részgráfba, így a kapott megoldás optimális lesz. Azt kell még belátnunk, hogy a kérdéses típusú sorrend létezik és előállítható polinom időben: például szélességi vagy mélységi kereséssel is ilyen sorrendhez jutunk.

Második megoldás. Egyszerűbb megoldás, ha először ellenőrizzük, hogy a kapott gráf páros-e (szélességi kereséssel, polinom időben). Ha igen, akkor a kimenet maga a gráf lesz, ellenkező esetben a két látott algoritmus valamelyikét hajtjuk végre. Páros gráfokra optimális, más gráfokra 2-közelítő algoritmust kapunk, ami nyilván polinom időben fut.

18. A második algoritmus kimenete is lefogó ponthalmaz, ezért lefogja az első algoritmusban szereplő M párosítás éleit is, így a mérete nem lehet kisebb $|M|$-nél, amiből az állítás következik, hiszen az első algoritmus kimenete $2|M|$ csúcsból áll.

19. Ha $\tau(G) = k$ mellett szeretnénk éles példát adni, akkor vegyük egyszerűen azt a gráfot, ami k független élből áll. Erre futtatva a két algoritmus bármelyikét,

a teljes csúcshalmaz lesz a kimenet, miközben nyilván feleekkora halmaz is elég lenne (minden élnek csak az egyik végpontja).

20. Mivel mindkét algoritmus egy (nem bővíthető) párosítás végpontjait adja kimenetként, a kimenet mindig páros számú csúcsból áll. Ha az optimum értéke páratlan, akkor tehát egyik algoritmus sem adhatja ezt. Olyan 10 csúcsú, 10 élű egyszerű, összefüggő G gráfot, melyre $\tau(G) = 3$ nem nehéz adni: legyen például G egy háromszög, melynek A, B, C csúcsaihoz rendre 3,2,2 további csúcs csatlakozik egy-egy éllel. Könnyen látható, hogy ekkor A, B, C lefogó ponthalmaz és hogy 2 elemű lefogó ponthalmaz nincs G-ben.

21. Az állítás nem igaz. Legyen G egy 8 csúcsú és egy 2 csúcsú teljes gráf diszjunkt uniója, erre nyilván teljesül $\tau(G) = 8$. A tanult algoritmusok akkor (és csak akkor) találnak 8 csúcsú lefogó ponthalmazt, ha az általuk talált nem bővíthető párosítás 4 élű. Könnyen látható azonban, hogy G-nek nincs ilyen párosítása: a különálló él minden nem bővíthető párosításban szerepel (hiszen a végpontjaira más él nem illeszkedik), így mivel a gráf többi csúcsa teljes részgráfot feszít, a nem bővíthető párosítások mind 5 élűek lesznek.

22. Tudjuk, hogy egy X csúcshalmaz akkor és csak akkor független a $G = (V, E)$ gráfban, ha $V \setminus X$ lefogó ponthalmaz G-ben. Keressünk a gráfban (például a látott mohó eljárással) egy nem bővíthető párosítást. A párosítás végpontjai által alkotott halmaz legyen L. Tudjuk, hogy L lefogó, $V \setminus L$ tehát független, legyen ez az algoritmusunk kimenete. Megmutatjuk, hogy $V \setminus L$ legalább feleakkora, mint a maximális független ponthalmaz (ezzel igazolva, hogy algoritmusunk approximációs faktora csakugyan 2, az eljárás polinomialitása nyilvánvaló). Mivel $n - |V \setminus L| = |L| \leq 2\nu(G) \leq 2\tau(G)$, tehát

$$|V \setminus L| \geq n - 2\tau(G) = \alpha(G) - \tau(G) \geq \frac{\alpha(G)}{2} + \frac{\alpha(G)}{2} - \tau(G) \geq \frac{\alpha(G)}{2},$$

felhasználva az $\alpha(G) + \tau(G) = n$ egyenlőséget (Gallai 1. tétele), a feladat szövegében megadott $\frac{\alpha(G)}{2} \geq \frac{n}{3}$ egyenlőtlenséget és az ezekből adódó $\tau(G) \leq \frac{n}{3}$ egyenlőtlenséget.
A $K_{2,5}$ teljes páros gráfon való alkalmazáshoz legyenek a csúcsok $K_{2,5}$ egyik osztályában 1 és 2, a másik osztályában A, B, C, D, E. Egy nem bővíthető párosítás 2 élből fog állni, az egyik tartalmazza az 1-et (pl. $(1, A)$), a másik a 2-t (pl. $(2, B)$). Az algoritmus által kimenetként adott ponthalmaz tehát ez esetben a $\{C, D, E\}$ halmaz lesz.

23. Az algoritmus mindig azt a részhalmazt választja ki, amelyre a lehető legkisebb a halmaz költségének és az újonnan lefedett elemek számának hányadosa.
Az első lépésben a `jimbo` részhalmazt kell választanunk, mert ennek a legkisebb az egy új elemre eső költsége ($\frac{3}{5}$). Ezt követően a `chef` részhalmaz jön, $\frac{3}{4}$

költséggel. A következő halmaz a `stan` kell legyen, ez újabb 4 elemet fed le, 4 költséggel, majd a `gerald` halmaz következik, itt a hányados $\frac{3}{2}$. Az ezt követő lépésben a `mackey` halmazra lesz minimális a hányados (2), az utolsó beválasztott szó pedig a `wendy`, 5-ös értékkel.
A kapott fedés tehát: {`jimbo, chef, stan, gerald, mackey, wendy`}, költsége 25.

24. Az algoritmus mindig azt a részhalmazt választja ki, amelyre a lehető legkisebb a halmaz költségének és az újonnan lefedett elemek számának hányadosa.
Az első lépésben a `béka` részhalmazt kell választanunk, mert itt a legkisebb az egy új elemre eső költség ($\frac{1}{2}$). A következő halmaz a `vércse` lesz, ez a hátralévő 8 betűből 5-öt fed le, 6 költséggel. Következik a `maki` halmaz, $\frac{3}{2}$ -es egy új elemre eső költséggel, végül a hátralévő `g` betűt a legolcsóbban az `egér` halmaz fedi.
A kapott fedés tehát: {`béka, vércse, maki, egér`}, költsége 15.

25. A MIN_LEFOGÓ az S_HALMAZ_FED speciális esete, így használhatjuk az S_HALMAZ_FED-re adott közelítő algoritmust. Mivel a bemenetként kapott gráf 3-reguláris, az S_HALMAZ_FED-ben szereplő részhalmazok mind 3 eleműek lesznek. Az S_HALMAZ_FED-re adott algoritmus approximációs faktorának bizonyításakor látottak szerint így a kapott megoldás az optimálisnak legfeljebb $H_3 = 1 + \frac{1}{2} + \frac{1}{3} = \frac{11}{6}$-szorosa lesz.

26. A csúcsok közt háromféle síkbeli távolság lehetséges: a hatszögön szomszédos csúcsok távolsága 1, ugyancsak 1 G és bármely más csúcs távolsága. A hatszögön másodszomszédos csúcsok távolsága $\sqrt{3}$, a szemköztieké 2. Ez alapján a megadott súlyozás nem lesz metrikus, a feladat megoldását tehát metrizálással kell kezdenünk. Ehhez meg kell állapítanunk az összes csúcspárra a köztük lévő legrövidebb út hosszát. Ez G és bármely más csúcs esetén 1, a hatszögön szomszédos csúcsok esetén $\frac{3}{2}$, bármely más csúcspár esetén 2 (mivel $\sqrt{3} + \frac{1}{2} > 2$). Ezek lesznek tehát a metrizálás után kapott gráf élsúlyai. Következő lépésként minimális összsúlyú feszítőfát kell keresnünk a B, C, E, F csúcsok által feszített részgráfban. Ebben benne lesz a BC és az EF él, melyek súlya $\frac{3}{2}$, valamint a maradék négy szóba jövő él közül egy tetszőleges (hiszen ezeknek egyaránt 2 a súlya), mondjuk CE. Ezzel a metrizált gráf egy Steiner-fáját kapjuk, amit az utolsó lépésben átalakítunk az eredeti gráf Steiner-fájává. Ehhez a szereplő éleknek megfelelő pontpárok közti legrövidebb utakat kell vennünk az eredeti gráfban. Ezek a BC és EF esetében maguk az élek, CE esetében a CGE út, az algoritmus tehát a BC, CG, GE, EF élekből álló fát adja kimenetként.

28. Legyenek a és b a Steiner-pontok. Ekkor az optimális Steiner-fa négyféle lehet aszerint, hogy melyiket tartalmazza az a és b pontok közül: tartalmazhatja csak a-t, csak b-t, mindkettőt vagy egyiket sem. A gráf többi pontját a fának természetesen tartalmaznia kell, hiszen ezek terminálok lesznek. Az optimális Steiner-fa

tehát a T, $T \cup \{a\}$, $T \cup \{b\}$, $T \cup \{a,b\}$ ponthalmazok valamelyike által feszített részgráf egy feszítőfája lesz. Így az említett feszített részgráfokon minimális feszítőfát keresve (ha van) a mohó algoritmussal, majd a kapott fák közül a minimális összsúlyút véve polinomiális algoritmust kapunk a legkisebb költségű Steiner-fa megtalálására.

29. Legyen a Steiner-pontok halmaza S, a terminálok halmaza T, az optimális Steiner-fa F. F tartalmazza T minden csúcsát és S csúcsainak egy R részhalmazát (ami lehet üres is). A $T \cup R$ halmazok által feszített részgráfokon minimális feszítőfát keresve tehát megtaláljuk az F fát vagy egy másik, vele azonos súlyú Steiner-fát, ha minden lehetséges $R \subseteq S$ halmazt kipróbálunk. A szóbajövő R halmazok száma $2^{|S|} = n^2$, ami a bemenet méretében polinomiális, az egyes esetek lépésszáma pedig (pl. a Kruskal-algoritmust használva) szintén polinomális. A kapott fák közül a minimális összsúlyút véve tehát polinomiális algoritmust kapunk a legkisebb költségű Steiner-fa meghatározására.

30. Legyenek a négyzet csúcsai A,B,C,D, az AC átlón lévő pontok (A-tól vett távolság szerint növő sorrendben) E,F,G. Az algoritmus először minimális összköltségű feszítőfát keres (például) Kruskal algoritmusával, azaz kiválasztja először az AE,EF,FG,GC éleket (valamilyen sorrendben), majd a BF,DF éleket (szintén tetszőleges sorrendben), mivel ezek a legrövidebbek azok közül, amik a már meglévőkkel együtt nem alkotnak kört. Következő lépésben az így kapott feszítőfa páratlan fokú csúcsai által feszített részgráfban kell minimális összsúlyú párosítást találnunk. A páratlan fokú csúcsok A,B,C,D, a minimális összsúlyú teljes párosítás tehát az AB,CD vagy pedig az AD,BC élekből áll. (Álljon mondjuk az AD,BC élekből). A feszítőfa és a párosítás éleinek uniójaként kapott részgráf egy Euler-körsétájának megkeresése, majd a tanult módszer szerint Hamilton-körré való levágása van hátra. Az Euler-körséta lehet pl. A,D,F,B,C,G,F,E,A (csak a csúcsokat felsorolva), ekkor az első és egyetlen levágás a $G-F-E$ út GE éllel való helyettesítése lesz. A kapott Hamilton-kör ekkor tehát $A-D-F-B-C-G-E-A$.

31. Az algoritmus először minimális súlyú feszítőfát keres a gráfban, ez nyilván a sokszög oldalai közül tartalmaz $(n-1)$-et. Ebben két páratlan fokú csúcs van, tehát a páratlan fokú csúcsokon vett minimális összsúlyú teljes párosítás egy élből áll, mely szintén a sokszög egyik (az eddig be nem vett) oldala. Ezt a fához véve az eredeti gráf Hamilton-körét kapjuk, így a levágások hatására az élhalmaz már nem változik. A kapott megoldás nyilván optimális, hiszen épp az n legkisebb súlyú élből áll.

32. Ha a 34. oldalon látott visszavezetésben a G-ben nem szereplő élek kn helyett 2 súlyúak lesznek, akkor metrikus élsúlyozást kapunk. Az így kapott problémára adott pontos válasz (vagyis egy optimális Hamilton-kör) ismeretében ugyanakkor

továbbra is el tudjuk dönteni, hogy az eredeti gráfban van-e Hamilton-kör: ha a kimenet súlya n, akkor igen, ha ennél nagyobb, akkor nem.

33. A feltétel szerint a gráf csúcsaiból és 1 súlyú éleiből álló részgráf összefüggő, így létezik a gráfnak csak 1 súlyú élekből álló feszítőfája (amit meg is tudunk találni polinom időben). Ennek az éleit megduplázva, majd a kapott gráf Euler-körét megkeresve és az approximációs algoritmusoknál látottak szerint levágva legfeljebb $2n-2$ súlyú Hamilton-kört kapunk (a levágások a metrikusság miatt nem növelik a költséget). Érvelhetünk azzal is, hogy mivel a gráfnak van $n-1$ súlyú feszítőfája, ezért a (mondjuk) Kruskal-algoritmussal talált feszítőfa súlya is legfeljebb ennyi, így a 35. oldalon látott 2-approximációs algoritmus legfeljebb $2n-2$ súlyú Hamilton-kört ad. Nem igaz ugyanakkor, hogy a Christofides-algoritmust futtatva legfeljebb $\frac{3}{2}n$ súlyú Hamilton-kört kapnánk (ld. a 34. feladatot).

34. Legyenek 1 súlyúak a gráf egy v csúcsából kiinduló élek, minden más él súlya legyen 2. Ez a súlyozás nyilván metrikus és bármely két csúcs közt létezik olyan út, amely csak 1 súlyú éleket használ, ugyanakkor a minimális összsúlyú Hamilton-kör (sőt, minden Hamilton-kör) súlya $2n-2$ lesz (hiszen az 1 súlyú élek közül csak kettőt használhat).

40. Elkészítjük a részösszegek listáit, minden fázisban $\delta = \frac{\varepsilon}{2n} = \frac{1}{24}$-del ritkítva. Ez azt jelenti, hogy 24-nél kisebb számok senkit nem tudnak képviselni, a 24 és 47 közti számok a náluk eggyel nagyobbat tudják képviselni, az ennél nagyobb számok tudnának két számot is képviselni, erre azonban nem fog sor kerülni, hiszen 49-nél nagyobb számokat nem veszünk fel a listákba.

$L_0 = \{0\},\ L'_0 = \{2\},\ L_1 = \{0,2\}$, itt ritkítani nem lehet.

$L'_1 = \{6,8\},\ L_2 = \{0,2,6,8\}$, ritkítani nem lehet.

$L'_2 = \{7,9,13,15\},\ L_3 = \{0,2,6,7,8,9,13,15\}$, ritkítani nem lehet.

$L'_3 = \{14,16,20,21,22,23,27,29\}$,

$L_4 = \{0,2,6,7,8,9,13,14,15,16,20,21,22,23,27,29\}$, ritkítani továbbra sem lehet.

$L'_4 = \{28,30,34,35,36,37,41,42,43,44,48,49\}$, hiszen a 49-nél nagyobb elemeket nem vesszük be a listába.

$L_5 = \{0,2,6,7,8,9,13,14,15,16,20,21,22,23,27,28,29,30,34,35,36,37,$
$41,42,43,44,48,49\}$, itt a ritkítás során töröljük a 28,30,35,37,42,44,49 elemeket, ezt követően tehát

$L_5 = \{0,2,6,7,8,9,13,14,15,16,20,21,22,23,27,29,34,36,41,43,48\}$.

$L'_5 = \{44,46\}$, hiszen a 49-nél nagyobb elemeket nem vesszük be a listába.

$L_6 = \{0,2,6,7,8,9,13,14,15,16,20,21,22,23,27,29,34,36,41,43,44,46,48\}$. Itt már nem ritkítunk, az algoritmus kimenete az utolsó lista legnagyobb eleme, vagyis 48. Figyeljük meg, hogy a 49-es célérték megjelent az L_5 listában,

de áldozatául esett a ritkításnak. Természetesen megtehetnénk, hogy az algoritmus során figyeljük, hogy megkapjuk-e t-t részösszegként és ha igen, akkor az adott részösszeget kimenetként megadva leállítjuk az algoritmus futását. Számos más trükkel is lehetne minimálisan javítani az algoritmust, például nem csak t megtalálásakor állhatnánk le, hanem már akkor is, ha kellően megközelítettük, de megtehetnénk azt is, hogy valamely c egész konstansra az utolsó c körben nem ritkítunk. Mindezek azonban magában az algoritmusban nem szerepelnek, a 48-tól eltérő kimeneti érték tehát nem helyes megoldása a feladatnak.

41. A pontos megoldás azon változatánál, amikor a t-nél nagyobb elemeket töröljük és minden részösszeget csak egyszer szerepeltetünk, minden lista legfeljebb $t+1$ elemből áll. A feladat feltétele szerint t most polinomiális a bemenet méretében, hiszen $t \leq \log a_1 a_2 \ldots a_n = \log a_1 + \log a_2 + \ldots + \log a_n$. Mivel láttuk, hogy ha minden lista hossza polinomiális, akkor maga az algoritmus is polinomiális, kész is vagyunk.

Függelék

$\alpha(G)$ $\rightarrow$ Független halmaz.

Binomiális tétel $(a+b)^n = \sum_{i=0}^{n} a^i b^{n-i} \binom{n}{i}$.

Csúcsszínezés $\rightarrow$ Színezés.

$\Delta(G)$ A G gráf maximális fokszáma.

Euler-körséta Olyan zárt élsorozat egy gráfban, mely minden élet pontosan egyszer tartalmaz.

Euler-séta Olyan élsorozat egy gráfban, mely minden élet pontosan egyszer tartalmaz.

Élgráf A G gráf élgráfja az a gráf, melynek csúcsai G éleinek felelnek meg, az e és f éleknek megfelelő csúcsok pedig pontosan akkor szomszédosak, ha e és f csatlakozó élek G-ben.

Élkromatikus szám $\rightarrow$ Élszínezés.

Élsorozat Egy $G = (V,E)$ gráfban $(v_0, e_1, v_1, e_2, v_2, \ldots, e_r, v_r)$ élsorozat, ha $v_0, v_1, \ldots, v_r \in V$, $e_1, e_2, \ldots, e_r \in E$ és az e_i él a v_{i-1} és v_i csúcsok közt megy. Az út olyan élsorozat, melyben minden csúcs különböző. Egy élsorozat zárt, ha $v_0 = v_r$. Körnek nevezünk egy olyan zárt élsorozatot, melyben a $v_1, v_2, \ldots, v_{r-1}$ csúcsok egymástól és $v_0 = v_r$-től is különböznek.

Élszínezés Egy G gráf élszínezése r színnel egy olyan $c : E(G) \rightarrow \{1, 2, \ldots, r\}$ függvény, melyre teljesül, hogy ha az e és f élek csatlakozók (vagyis van közös végpontjuk), akkor $c(e) \neq c(f)$. A G gráf $\chi_e(G)$-vel jelölt élkromatikus száma k, ha G-nek létezik élszínezése k színnel, de nem létezik $k-1$ színnel.

Feszített részgráf Egy $G = (V,E)$ gráfnak $H = (X,F)$ feszített részgráfja, ha $X \subseteq V$ és F-ben pontosan azon E-beli élek szerepelnek, melyek mindkét végpontja X-beli. H-t ilyenkor G-nek az X által feszített részgráfjának nevezzük és $G[X]$-szel jelöljük.

Feszítőfa Egy G gráfnak F feszítőfája, ha F fa, részgráfja G-nek és G minden csúcsát tartalmazza.

Független élhalmaz $\rightarrow$ Párosítás.

Független ponthalmaz Egy G gráf $X \subseteq V(G)$ ponthalmaza független, ha X semelyik két pontja közt sincs él. $\alpha(G)$ jelöli egy gráf maximális elemszámú független ponthalmazának méretét. Ld. még: Gallai tételei, Kőnig tételei.

Gallai tételei 1. Ha G n csúcsú (hurokélmentes) gráf, akkor $\alpha(G) + \tau(G) = n$. 2. Ha G n csúcsú, izolált pontot nem tartalmazó gráf, akkor $\nu(G) + \rho(G) = n$.

Hamilton-kör A G gráf egy köre Hamilton-kör, ha G minden csúcsát tartalmazza.

Hamilton-út A G gráf egy útja Hamilton-út, ha G minden csúcsát tartalmazza.

Intervallumgráf Tegyük fel, hogy adott valós, zárt intervallumok egy rendszere. Az intervallumrendszerhez tartozó gráf csúcsai az intervallumok, két csúcsot összekötünk, ha a hozzájuk tartozó intervallumoknak van közös eleme. Egy G gráf intervallumgráf, ha létezik olyan intervallumrendszer, melyhez épp G tartozik.

Klikk, klikkszám A G gráf teljes részgráfjait klikknek nevezzük, a maximális klikk méretét (vagyis csúcsainak számát) $\omega(G)$-vel jelöljük és G klikkszámának hívjuk.

Komplementer Egy $G = (V, E)$ egyszerű gráf komplementere a $\overline{G} = (V, E')$ gráf, ahol $E' = \{(x, y) : x \neq y, (x, y) \notin E\}$.

Kör Egy élsorozat zárt, ha $v_0 = v_r$. Körnek nevezünk egy olyan zárt élsorozatot, melyben a $v_1, v_2, \ldots, v_{r-1}$ csúcsok egymástól és $v_0 = v_r$-től is különböznek.

Kőnig tételei 1. Ha G páros gráf, akkor $\nu(G) = \tau(G)$ (és így Gallai tételei miatt $\alpha(G) = \rho(G)$). 2. Ha G páros gráf, akkor $\chi_e(G) = \Delta(G)$.

Kromatikus szám $\rightarrow$ Színezés.

Kruskal algoritmusa Élsúlyozott gráf minimális összsúlyú feszítőfájának megkeresésére szolgál. Az éleket súly szerint növően rendezzük, majd e sorrend szerint haladva egy élet pontosan akkor választunk be a kimeneti élhalmazba, ha a már beválasztottakkal nem alkot kört.

Lefogó élhalmaz Egy G gráf $C \subseteq E(G)$ élhalmaza lefogó, ha a C-beli élek végpontjai közt G minden csúcsa előfordul. Ha G-ben van izolált pont, akkor természetesen nincs lefogó élhalmaza. $\rho(G)$ jelöli egy gráf minimális elemszámú lefogó élhalmazának méretét. Ld. még: Gallai tételei, Kőnig tételei.

Lefogó ponthalmaz Egy G gráf $X \subseteq V(G)$ csúcshalmaza lefogó, ha minden élnek tartalmazza legalább az egyik végpontját. $\tau(G)$ jelöli egy gráf minimális

elemszámú lefogó csúcshalmazának méretét. Ld. még: Gallai tételei, Kőnig tételei.

Mohó színezés Egy gráf csúcsait szeretnénk az $1,2,\ldots,r$ színekkel színezni. Rögzítjük a csúcsok egy sorrendjét, majd ebben a sorrendben haladva színezünk úgy, hogy az aktuális csúcs a legkisebb sorszámú olyan színt kapja, amilyen színű csúccsal nincs közös szomszédja. Értelemszerű változtatásokkal csúcsok helyett természetesen éleket is színezhetünk így.

Négyszín-tétel Minden síkbarajzolható gráf színezhető négy színnel.

$\nu(G)$ $\rightarrow$ Párosítás.

Ötszín-tétel Minden síkbarajzolható gráf színezhető öt színnel.

Reguláris gráf Egy gráf r-reguláris, ha minden csúcsának foka r.

$\rho(G)$ $\rightarrow$ Lefogó élhalmaz.

Színezés Egy G gráf (csúcs)színezése r színnel egy olyan $c : V(G) \rightarrow \{1,2,\ldots,r\}$ függvény, melyre teljesül, hogy $(x,y) \in E(G)$ esetén $c(x) \neq c(y)$, vagyis szomszédos csúcsokhoz különböző színeket kell rendelni. A G gráf $\chi(G)$-vel jelölt kromatikus száma k, ha létezik színezése k színnel, de nem létezik $k-1$ színnel.

$\tau(G)$ $\rightarrow$ Lefogó ponthalmaz.

Út Olyan élsorozat, melyben minden csúcs különböző.