Eötvös Loránd Tudományegyetem
Természettudományi Kar

Sápi András
Matematika BSc
Alkalmazott matematikus szakirány

Gráfok élfelbontása

Szakdolgozat

Témavezető: Barát János tudományos főmunkatárs
MTA-ELTE Geometriai és Algebrai Kombinatorika kutatócsoport

Belső konzulens: Szőnyi Tamás egyetemi tanár
Számítógéptudományi Tanszék

Budapest, 2015

Köszönetnyilvánítás

Először szeretnék köszönetet mondani a témavezetőmnek, Barát Jánosnak. Hálás vagyok hasznos ötleteiért, magyarázataiért és tanácsaiért, melyekkel rengeteget segített, hogy elkészüljön ez a szakdolgozat.

Nagyon köszönöm szüleimnek és testvéreimnek a végtelen türelmüket és támogatásukat, amire mindig számíthattam.

Köszönettel tartozom a barátaimnak, akik mindig segítségemre voltak mindenben és mellettem álltak.

Tartalomjegyzék

1. Élfelbontás

1.1. Bevezetés

1.1.1. Definíció. *Legyen Γ gráfok egy halmaza. Azt mondjuk, hogy G gráfnak van Γ-felbontása, ha G éleit olyan éldiszjunkt részgráfokra tudjuk osztani, hogy mindegyik izomorf egy Γ-beli gráffal.*

Sokszor $\Gamma = \{H\}$, vagyis egy gráfot használunk csak, ekkor a H-felbontása G-nek kifejezést használjuk. Ekkor feltesszük, hogy H élszáma osztja G élszámát, különben triviálisan nem létezhet felbontás.

Vizsgálni fogjuk, hogy milyen feltételeket kell megkövetelnünk gráfokra, hogy létezzen Γ-felbontásuk adott Γ-ra. A legegyszerűbb kérdés, hogy minden páros élszámú összefüggő gráf felbontható-e kettő élű utakra.

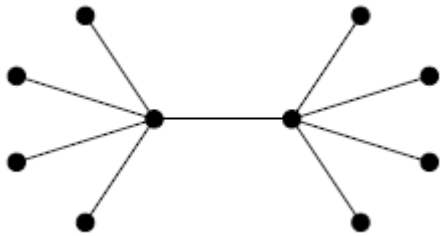
1.1.2. Definíció. *Útnak nevezzük egy $G = (V, E)$ gráf éleinek olyan egymáshoz csatlakozó sorozatát, melyben sem él, sem csúcs nem fordulhat elő egynél többször. Jelölése: $P = (v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n)$, ahol $v_i \in V$, $e_i \in E$, illetve $e_i = v_{i-1}v_i$. Az n csúcsú utat P_n-nel jelöljük, ennek $(n-1)$ éle van.*

1.1.3. Lemma (Kötzig)**.** *[8] Minden összefüggő, páros élszámú gráf felbontható 2-élű utakra.*

Bizonyítás. Mohó módon helyezzünk el 2-élű utakat a gráfon, amíg ez lehetséges, vagyis nincs már két olyan él, melyek szomszédosak, és nem fedettek. Ekkor vagy lefedtük az összes élet, vagy páros sok fedetlen él maradt. Mivel a gráf összefüggő bármely két fedetlen él között létezik út. Válasszunk ezek közül kettőt, e_1-et és e_2-t úgy, hogy a köztük lévő P út a legrövidebb ilyen út legyen. Mivel e_1 és e_2 nem voltak szomszédosak, így a P út tartalmaz egy e' élet, mely szomszédos e_1-gyel, és része egy 2-élű útnak, legyen ez T. Legyen T másik éle e''. Ekkor három eset lehetséges: ha $e'' \in P$, akkor a T útban e''-t e_1-re cseréljük. Ha e'' nincs a P útban, de szomszédos e_1-gyel, akkor T-ben e'-t cseréljük e_1-re. Ha e'' nincs a P útban, és nem szomszédos e_1-gyel, akkor T-ben cseréljük e''-t e_1-re. Mindhárom esetben a két kiválasztott fedetlen él (az egyik változik, de a cserével fennmaradót vizsgáljuk) közötti út hossza csökken. Elegendő alkalommal ismételve ezt a lépést a két él szomszédos lesz, így egy újabb utat helyezhetünk el, csökkentve a fedetlen

élek számát. Ezt iterálva a végén 0 fedetlen élünk marad, mert páros sok éle van a gráfnak. □

Következő lépésként vizsgálhatnánk a 3-élű utakat, viszont itt nem elegendő feltennünk az összefüggőséget, mert az 1. ábra ellenpélda erre.



1. ábra. Összefüggő, háromélű utakra nem bontható példa

Vizsgálva ezt a gráfot könnyen észrevehetjük, hogy a felbonthatatlanságot az elvágó él létezése okozza. Ezt kiküszöbölendő feltesszük, hogy nincs ilyen, vagyis a gráf legalább kétszeresen élösszefüggő. Definiáljuk a k-élösszefüggőséget:

1.1.4. Definíció. *Egy G gráf k-szorosan élösszefüggő, ha tetszőleges k-nál kisebb elemszámú élhalmazát elhagyva a kapott gráf összefüggő marad.*

Vezessük be az s-partíció fogalmát:

1.1.5. Definíció. *Legyen s tetszőleges természetes szám. Egy olyan felbontást amelyben minden s-élű összefüggő gráf megengedett, s-partíciónak nevezzük.*

Ebben a megfogalmazásban az előbbi lemma szerint minden páros élszámú összefüggő gráfnak létezik 2-partíciója.

1.2. Partíciók

Jünger, Reinelt és Pulleyblank [6] vizsgálták, hogy adott s-partíció létezéséhez milyen élösszefüggőségi számot kell megkövetelnünk. Feltesszük továbbá, hogy hurokélmentes a gráf, mivel a hurokélet egy megfelelő hosszúságú körre cserélve ugyanakkor fedhető a két gráf. Az imént láttuk, hogy 2-partíció létezéséhez elegendő 1-élösszefüggő gráfot vennünk. A továbbiakban látni fogjuk, hogy 3-partíció létezéséhez 2-élösszefüggőségre van szükségünk, 4-partícióhoz 3-élösszefüggőségre, és ha 4-élösszefüggő a gráfunk, akkor tetszőleges s-re létezik s-partíciója. Először egy hasznos lemma, ami segítségünkre lesz ezek belátásához:

1.2.1. Definíció. *Vonalnak nevezzük egy $G = (V, E)$ gráf éleinek olyan egymáshoz csatlakozó sorozatát, melyben él nem fordulhat elő egynél többször (csúcsok ismétlődése megengedett). Jelölése: $L = (v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n)$, ahol $v_i \in V$, $e_i \in E$, illetve $e_i = v_{i-1}v_i$.*

1.2.2. Lemma. *Legyen G egy olyan gráf, amely tartalmaz egy $L = (v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n)$ vonalat, mellyel minden G-beli él szomszédos. Ekkor G-nak létezik s-partíciója minden $s \geq 2$ egész számra úgy, hogyha s nem osztja az élszámot, akkor a maradék élek is összefüggőek, és ez a "kis rész" illeszkedik v_n-re.*

Bizonyítás. Minden $i \in 0,1,\ldots,n$-re legyen E_i azon nem L-beli élek halmaza, melyek illeszkednek v_i-re, de v_j-re nem $j < i$ esetén (mivel a csúcsok ismétlődhetnek, így lehetséges $v_i = v_j$, ekkor a nagyobb indexű csúcshoz egy él sem tartozik). Minden i-re legyen $\tilde{E}_i$ az E_i-beli élek egy tetszőleges sorrendje, és tekintsük G éleinek következő felsorolását: $\tilde{E}_0, e_1, \tilde{E}_1, e_2, \ldots, e_{n-1}, \tilde{E}_{n-1}, e_n, \tilde{E}_n$. Ebben a felsorolásban két egymást követő él mindig szomszédos G-ben, így véve az s darab soron következőt kapunk egy s partíciót, illetve a kimaradó élek is szomszédosak lesznek, és legalább egy illeszkedik v_n-re. □

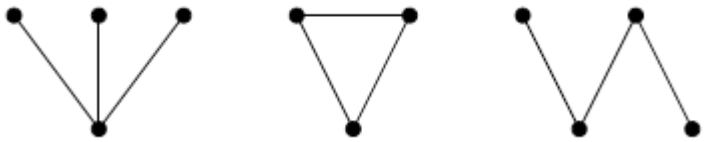
Azt szeretnénk belátni, hogy bármely kétszeresen élösszefüggő gráfnak létezik 3-partíciója, amennyiben az élszám többszöröse háromnak. Ennél egy kicsit erősebb állítást fogunk bizonyítani:

1.2.3. Tétel (Jünger, Reinelt, Pulleyblank)**.** *[6] Ha $G = (V, E)$ egy 2-élösszefüggő gráf, akkor bármely $v \in V$ csúcs esetén létezik egy 3-partíciója, mely*

1. *ha létezik egy "kis rész", akkor ennek minden éle illeszkedik v-re,*
2. *ha bármely 3-élű résznek elvágó pontja v, akkor az a rész csak karom lehet, vagy pszeudokarom, és v ennek a három fokú csúcsa.*

A tételben nem tettük fel, hogy háromnak többszöröse az élszám, ezzel kapcsolatos a tétel első pontja. Mivel 3-élű részgráfokra akarjuk felbontani a gráfot, így egyszerű gráf esetén csak 3 különböző lehetőségünk van: a 3 élű út, a háromszög és a karom ($K_{1,3}$). Mivel nem feltétlen egyszerű a gráfunk, így még lehetséges két elem. Ezek a karom általánosításai, olyan szempontból, hogy a háromfokú pontnak háromnál kevesebb szomszédja van. 3-kötésnek nevezzük, mikor csak egy szomszédja van, vagyis két csúcs van összekötve 3 éllel. A másik eshetőség, ha két szomszédja

van, az egyikhez két éllel van kötve, így egy 2 hosszú kört kapunk, és hozzákötve még egy élet. Ezt nevezzük pszeudokaromnak. A tétel második állítása azt mondja, hogy találhatunk olyan partícionálást, melyben v nem egy 3 élű út belső pontja (ez az induktív bizonyítás miatt fontos).



2. ábra. Háromélű egyszerű gráfok: karom, háromszög, háromélű út

3. ábra. Háromélű nem egyszerű gráfok: 3-kötés, pszeudokarom

Bizonyítás. Induktívan bizonyítjuk a tételt. Először egy segédállítás:

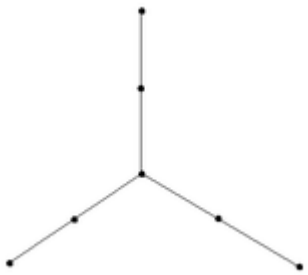
1.2.4. Állítás. *Ha G tartalmaz egy olyan v-t tartalmazó C kört, melyben minden csúcs legfeljebb egyszer szerepel és G minden éle illeszkedik ennek a körnek valamely csúcsára, akkor létezik a tétel feltételeinek megfelelő 3-partíció.*

Ezt az 1.2.2. lemma alapján könnyen láthatjuk. Legyen J a v csúccsal szomszédos élek halmaza. Legyen $P = (v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n)$ az az út, melyet akkor kapunk, ha töröljük v-t C-ből. Legyen $j = v_0v \in J$. Ha $|E\backslash J| \equiv 0 \mod 3$, vagy $|E\backslash J| \equiv \equiv 2 \mod 3$, akkor alkalmazzuk a lemmát $G\backslash J$-re. Ha $|E\backslash J| \equiv 1 \mod 3$, akkor $G\backslash(J\backslash\{j\})$-re (ekkor v foka 1). A lemma szerint kapunk egy 3-partíciót, és ha létezik "kis rész", akkor két éle van, és v_n-re illeszkedik. Ebben az esetben ehhez hozzáadjuk a v_n-re és v-re illeszkedő élet. Ezután csak J maradék éleit kell partícionálnunk. Ezt megtehetjük karmokká, pszeudokarmokká vagy 3-kötésekké, illetve ha van kimaradó él, akkor a "kis rész" minden éle illeszkedik v-re.

Innen csúcsszám szerinti indukcióval beláthatjuk a tételt: legyen C bármely olyan kör G-ben, amely tartalmazza v-t (ilyen létezik, mert G kétszeresen összefüggő). Ha minden G-beli él szomszédos ezzel a körrel, akkor a segédállítás szerint kész vagyunk. Ha nem, akkor húzzuk össze C csúcsait egy ponttá, így létrehozva egy v' csúcsot egy

kisebb csúcsszámú G' gráfban. Erre alkalmazzuk az indukciós feltevésünk (G' is 2-élösszefüggő, így megtehetjük), vagyis igaz rá az állítás (v' a kitüntetett csúcs). Ekkor a kapott partícionálásból eltávolítjuk a "kis részt", ha van, illetve a v'-re illeszkedő karmokat és pszeudokarmokat. Így az eredeti G-ben kaptunk egy részleges fedést, illetve a kimaradt élek mind a C körre illeszkednek, így alkalmazhatjuk a segédállításunk (v-t tartalmazza C). Így a teljes gráfot partícionáltuk. □

Az egyszeres élösszefüggőség nem elegendő, könnyen tudunk mutatni ellenpéldát:



4. ábra. Összefüggő, nem 3-partícionálható gráf

Következőként azt szeretnénk belátni, hogy bármely 3-élösszefüggő gráfnak létezik 4-partíciója. Az előzőhöz hasonlóan itt is egy kicsit erősebb állítást bizonyítunk, illetve itt is feltesszük, hogy nem tartalmaz a gráf hurokélet(már láttuk, hogy ezt megtehetjük).

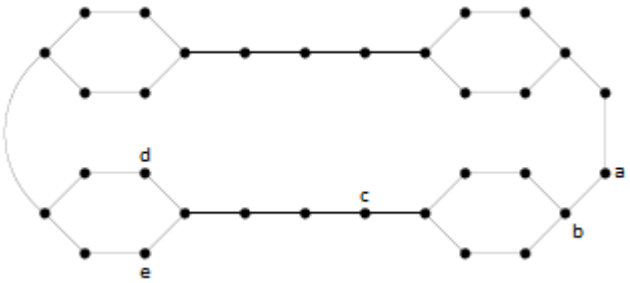
1.2.5. Tétel (Jünger, Reinelt, Pulleyblank)**.** *[6] Ha G egy háromszorosan élösszefüggő gráf és $v \in V$ tetszőleges csúcsa, akkor létezik olyan 4-partíciója, mely:*

1. *ha létezik "kis rész", akkor minden éle illeszkedik v-re,*
2. *ha bármely 4-élű résznek elvágó pontja v, akkor mind a 4 él illeszkedik v-re.*

Bizonyítás. A bizonyítás hasonló az előző tétel bizonyításához. Még felhasználjuk Menger tételét, hogy 3-élösszefüggő gráfban bármely pontpár között létezik 3 éldiszjunkt út (megtalálható a [7] könyvben). Ez azt jelenti, ha veszünk egy C kört, ami tartalmazza v-t, akkor létezik a kör éleit nem tartalmazó út, amely v és egy másik körbeli pont között fut (mely nem szomszédja a körben). Találhatunk olyan

kört is, amikor ez nem út hanem csak egy él. Elegendő venni egy pontot v szomszédjai közül. Az összekötő élen kívül létezik még két éldiszjunkt út, ezek egy kört alkotnak. Nevezzük az élet ami összeköti a másik körbeli ponttal "húrnak". Bizonyításunk alapja ismét indukció lesz. Legyen C olyan kör, mely tartalmazza v-t és v-hez tartozik egy h húr. Tegyük fel, hogy minden G-beli él illeszkedik a C körre. Legyen $P = (v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n)$ az az út, melyet akkor kapunk, ha töröljük v-t C-ből. Legyen $j = vv_0$ és $l = vv_n$. Legyen J az összes v-re illeszkedő él halmaza. Ha $|E\backslash J| \equiv 0 \bmod 4$ vagy $3 \bmod 4$, akkor használjuk 1.2.2. lemmát $G\backslash J$-re. Ha $|E\backslash J| \equiv 2 \bmod 4$, akkor a lemmát $G\backslash(J\backslash\{j\})$-re alkalmazzuk. Ha $|E\backslash J| \equiv 1 \bmod 4$, akkor a lemmát $G\backslash(J\backslash\{j,k\})$-ra alkalmazzuk úgy, hogy v ne legyen elvágó pontja a keletkező "kis résznek". Így kapunk egy részleges 4-partíciót, ahol ha van "kis rész", akkor annak mérete három. Az esetleges "kis részhez" hozzávehetjük l-t, így ez is 4 elemű lesz. A maradék J-beli éleket már csak csoportosítanunk kell négyesével, ezzel kész a partícionálás, és teljesül a két tételbeli állítás is. Ha C nem tartalmaz minden G-beli élet, akkor az előző tételhez hasonlóan összehúzható egy ponttá, és alkalmazható az indukció. Az esetleges "kis rész" minden éle illeszkedik az összehúzással kapott v'-re, vagyis illeszkedik valamely C-beli csúcsra. Így az eredeti gráfban van egy részleges fedésünk, és egy C kör amire illeszkednek a fedetlen élek, így alkalmazható az 1.2.2. lemma. □

A feltétel nem gyengíthető, létezik 2-élösszefüggő gráf, melynek nincs 4-partíciója (5. ábra).



5. ábra. 2-élösszefüggő ellenpélda

A gráfnak 36 éle van, így 9 darab részgráfra kell bontanunk. A gráfban található 4 darab 6-élű kör. Az könnyen láthatjuk, hogy nem lehetséges, hogy valamely részgráf egy ilyen körnek pontosan 1 élét tartalmazza, mert ekkor a maradék 5-öt egyetlen részgráffal kellene lefednünk (különben maradna egy szeparált él). Emiatt az ábrán a gráf jobb oldalán található 3 összekötő él nem kerülhet egy részgráfba. Legyen ab

él ami külön részgráfba kerül, ebben még 3 él szerepel a szomszédos körből. Így az a és c közötti rész pontosan 2 részgráf lesz (8 darab él). A következő rész c-től indul, és d-ig vagy e-ig egy út. Így pontosan egy éle van egy adott körben, ezért nincs felbontása.

A következő tételhez szükségünk lesz Tutte és Nash-Williams tételére:

1.2.6. Tétel (Tutte, Nash-Williams)**.** *[13, 18] Egy gráf akkor és csak akkor tartalmaz k darab éldiszjunkt feszítőfát, ha csúcsainak minden P partícionálása esetén létezik legalább $k(|P|-1)$ darab kereszt-él, vagyis olyan él, amelynek két csúcsa különböző partícióban fekszik.*

Lássuk ennek a számunkra fontos következményét:

1.2.7. Következmény. *Minden $2k$-szorosan élösszefüggő gráf tartalmaz k darab éldiszjunkt feszítőfát.*

Bizonyítás. Minden P-beli partícióból indul legalább $2k$ él a többi partícióba. Így minden r darab partícióra bontás esetén G-nek legalább $\frac{1}{2}\sum_{i=1}^{r} 2k = kr$ kereszt-éle van. Ez kielégíti a tétel feltételét, vagyis létezik k darab éldiszjunkt feszítőfa. □

1.2.8. Tétel (Jünger, Reinelt, Pulleyblank)**.** *[6] Ha G egy négyszeres élösszefüggő gráf, és $|E(G)| = m_1 + m_2 + \ldots + m_k$, akkor $\exists G_1 \dot{\bigcup} G_2 \dot{\bigcup} \ldots \dot{\bigcup} G_k$ összefüggő részgráfokra való felbontás úgy, hogy $\forall i : |E(G_i)| = m_i$.*

Az előbbi tétel szerint G-ben található 2 éldiszjunkt feszítőfa. Egy segédtétel amit felhasználunk:

1.2.9. Tétel (Jaeger)**.** *[4] Minden 4-élösszefüggő gráfban létezik egy összefüggő feszítő Euler-részgráf.*

Bizonyítás. Legyen a két éldiszjunkt feszítőfa T_1 és T_2. Jelöljük meg T_1 páratlan fokszámú csúcsait. Ezekből páros sok van, mivel a fokszámok összege páros. Vizsgáljuk ezeket a csúcsokat T_2-ben. Ekkor létezik $P_1, P_2, \ldots, P_l$ éldiszjunkt útrendszer a megjelölt csúcsokból képzett párok között. Ezen utak unióját jelöljük $P_1 \cup P_2 \cup \cup \ldots \cup P_l = N$-nel. Ezután tekintsük $T_1 \cup N$ éleit. Mivel T_1 feszítőfa, így ezek is feszítik G-t, illetve minden csúcs fokszáma páros, mivel ha T_1-ben páratlan volt a fokszáma, akkor N-ben egy út végpontja, így összesen páros a foka, ha T_1-ben páros volt a foka, akkor N-ben útnak csak belső pontja lehet, így páros sok él illeszkedik rá. Így $T_1 \cup N$-ben létezik Euler-részgráf. □

Innen az 1.2.8. tétel bizonyítása már könnyen következik. Az előbbi tétel biztosította Euler-részgráfra alkalmazzuk az 1.2.2. lemmát, így megkapjuk a megfelelő részgráfokat.

1.2.10. Következmény. *Minden négyszeresen élösszefüggő gráfnak létezik s-partíciója, ha a gráf élszámát osztja s.*

Ehhez elegendő minden m_i-t s-nek választani.

A továbbiakban foglalkozunk a 3-élű gráfokra bontással. A 2. ábrán láttuk, hogy három különböző 3-élű gráf van, és az 1.2.3. tétel szerint a 2-élösszefüggő gráfok lefedhetőek a három gráf segítségével. Bizonyos esetekben nem használhatjuk mindhármat, például a páros gráfok nem tartalmaznak háromszöget. A következő két alfejezetben vizsgálni fogjuk, hogy mely gráfok fedhetőek le csak $K_{1,3}$, vagy csak P_4 felhasználásával.

1.3. Háromélű utak

1.3.1. Tétel. *[1] Legyen G egy kétszeresen élösszefüggő, 3-reguláris gráf. Ekkor G-nek van P_4-felbontása.*

Bizonyítás. A gráf éleinek számát osztja három, mivel minden csúcs foka 3, így a gráfnak $\frac{3n}{2}$ éle van (ebből az is következik, hogy csakis páros sok csúcsa lehet). Petersen tétele szerint G-ben van teljes párosítás (a [7] könyvben megtalálható a tétel és bizonyítása). Legyen a teljes párosítás M. Elhagyva ezt G-ből minden él fokszáma eggyel csökken, így egy 2-reguláris gráfot kapunk. Ekkor $G - M$ vagy egy kör, vagy körök uniója. Irányítsuk meg az éleket úgy, hogy irányított köröket kapjunk. Így minden pont kifoka pontosan 1. Ekkor megkapjuk a 3-élű utakat, ha M minden éléhez hozzávesszük az előbbi irányításban a végpontokból induló éleket. □

Részben ezt felhasználva bizonyíthatjuk a következő maximális síkgráfokra vonatkozó állítást.

1.3.2. Definíció. *Egy síkgráfot maximálisnak nevezünk, ha egy további élet hozzávéve,a keletkező gráf már nem síkgráf.*

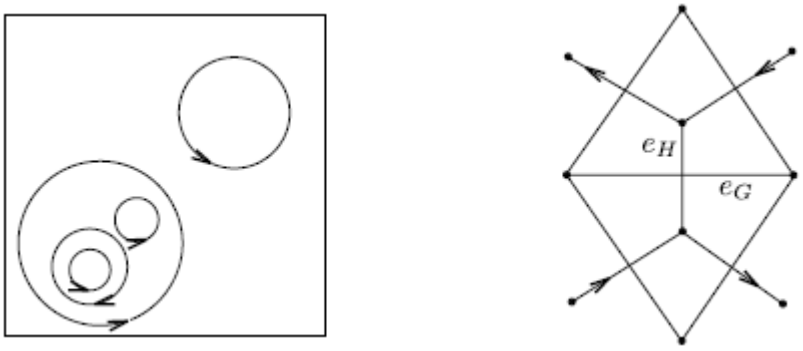
Egy ilyen gráf minden tartománya háromszög, és Euler-tétele miatt $3n - 6$ éle van, vagyis élszáma osztható hárommal.

1.3.3. Definíció. *Egy síkba rajzolható G gráf duális gráfja alatt azt a G' gráfot értjük, amelynek csúcsai az eredeti gráf tartományai, és azok a csúcsok vannak összekötve, amik megfelelői szomszédosak voltak (ha több él mentén voltak szomszédosak, akkor többszörösen is).*

1.3.4. Megjegyzés. *Síkbarajzolható gráf duálisa is síkbarajzolható gráf.*

1.3.5. Tétel (Häggkvist, Johansson)**.** *[3] Minden legalább 4 csúcsú maximális síkgráfnak van P_4 felbontása.*

Bizonyítás. Legyen G egy gráf a fenti tulajdonságokkal. Ekkor G duálisa 3-reguláris, 2-élösszefüggő gráf, legyen ez H. Petersen tétele alapján ebben található egy teljes párosítás, melyet elhagyva körök unióját kapjuk. Csináljunk ezen körökből irányított köröket. Azon körök, melyek a végtelen tartománnyal határosak kapják a sík irányításának megfelelő irányt. Azok a körök, amelyek az első lépésben irányított körök belsejében a határon vannak, azok kapják az ellentétes irányítást. Folytassuk ezt az eljárást, amíg minden kört meg nem irányítunk (6. ábra).



6. ábra. Körök irányítása és az eljárás eredménye

Ekkor a H duálisában azok az élek, amelyek áthaladnak egy adott élre illeszkedő két háromszög-tartományon, ellentétesen vannak irányítva. Minden $e \in M$ esetén ragasszuk hozzá e-hez azt a két H-beli élt, amely kifelé van irányítva e végpontjaiból. Így kapjuk a H gráf egy P_4 felbontását. Ezeknek az éleknek megfelelő élek a G gráfban szintén egy P_4-felbontást adnak. Csak azt kell ellenőriznünk, hogy sem $K_{1,3}$, sem háromszög nem keletkezhet, az irányítások megfelelő választása miatt. □

Ellenpéldát tudunk mutatni 2-élösszefüggő gráfra, melynek nincs P_4 felbontása. Egy ellenpélda lehet a kocka élgráfja, vagy a következő konstrukciójú gráf. Vegyünk 3 darab háromszöget, és ezeket kössük össze $1-1$ éllel úgy, hogy két összekötő élnek

ne legyen közös csúcsa. Így a gráfnak 9 csúcsa és 12 éle van. Valóban 2-élösszefüggő, mert tetszőleges élet elhagyva összefüggő marad. Négy úttal viszont nem tudjuk lefedni, mert csak 3 darab összekötő élünk van, így az egyik út csak háromszögbeli élekből állna, ami nem lehetséges.

Egy érdekes eredmény, melyet Thomassen bizonyított:

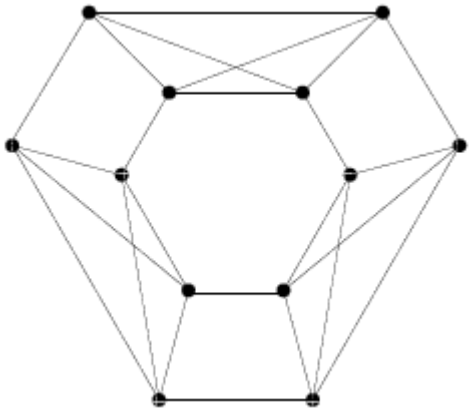
1.3.6. Tétel (Thomassen)**.** *[15] Ha G egy 171-élösszefüggő gráf, melynek élszáma osztható hárommal, akkor létezik P_4 felbontása.*

1.4. Karomfelbontás és irányítások

Láttuk, hogy 3-partíció létezéséhez szükséges legalább 2-élösszefüggőség. Most azt vizsgáljuk, hogy ha csak a karmot használhatjuk, akkor milyen élösszefüggőségre van szükségünk.

1.4.1. Sejtés. *[2] Létezik egy legkisebb k_c természetes szám, hogy minden egyszerű k_c-élösszefüggő, három többszörös élszámú gráfnak létezik karomfelbontása.*

4-élösszefüggő, karmokra nem bontható gráfot tudunk mutatni. Ehhez veszünk 3 darab K_4-et, és ezeket $2-2$ éllel összekötjük. Így minden csúcs foka 4 lesz, összesen 24 élünk van, ami 8 karom elhelyezését jelenti. Vagyis 4 csúcs lesz, amibe nem kerül 3-fokú karom csúcs (minden karom egy ilyen csúccsal rendelkezik, vagyis 8 lehet ilyen). A skatulya-elv szerint lesz olyan K_4, amibe két ilyen is kerül, vagyis a köztük futó élet nem tudjuk semmiképp sem lefedni.



7. ábra. 4-élösszefüggő ellenpélda

Vagyis már tudjuk, hogy $k_c \geq 5$. Célunk ezután, hogy minél kisebb felső korlátot adjunk. Ehhez vizsgáljuk meg a felbontások és irányítások kapcsolatát. A karomfelbontások megfogalmazhatóak irányításként is. Ha egy G gráfnak létezik karomfelbontása, akkor G irányítható a következő módon: a felbontás minden karmában az élek mutassanak a 3-fokú csúcsból az 1-fokúakba. Így minden v csúcs kifoka d^+ $+(v) \equiv 0 \bmod 3$. Ha G-nek van egy ilyen irányítása, akkor létezik karomfelbontása is (az adott pontból kimenő éleket csoportosítjuk hármasával, ezek egy-egy karmot adnak). Ha egy irányított gráf v csúcsára $d^+(v) \equiv d^-(v) \bmod 3$, akkor azt mondjuk, hogy v kiegyensúlyozott (mod 3). A G gráf egy irányítása Tutte-irányítás, ha minden csúcs kiegyensúlyozott (mod 3).

1.4.2. Sejtés. *[5] Létezik egy legkisebb k_t természetes szám, hogy minden k_t-élösszefüggő gráfnak van Tutte-irányítása.*

Látszólag a két irányítás lényegesen különböző. A következő tételek már mutatnak hasonlóságot:

1.4.3. Tétel. *[1] Minden legalább 5 csúcsú G maximális síkgráfnak létezik karomfelbontása.*

Bizonyítás. Felhasználjuk Nash-Williams eredményét, hogy minden síkgráf felbomlik legfeljebb három erdő uniójára [14]. Ezután elérjük, hogy minden erdő pontosan két fából álljon. Mivel G maximális síkgráf, így $3n - 6$ éle van. Tudjuk, hogy egy feszítőfa élszáma $n - 1$, illetve egy k darab fából álló feszítőerdő élszáma $n - k$. Így a három erdő élszámára csak 3 lehetőség van: $(n-1, n-1, n-4)$, $(n-1, n-2, n-$ $-3)$, $(n-2, n-2, n-2)$. Ezek az esetek megfelelnek annak, hogy az egyes erdők (1,1,4), (1,2,3), (2,2,2) fából állnak. Ebből mi az utolsót szeretnénk elérni, vagyis azt, hogy minden erdőben pontosan két fa legyen. Ha valamelyik erdő legalább 3 fából áll, akkor van egy 1 fából álló erdő, vagyis feszítőfa, ennek van olyan éle, amely az előbbi erdőben két komponens között fut. Ezt az élet áthelyezve a feszítőfa két fára esik szét, míg a másik erdő komponenseinek száma eggyel csökken. Ezt legfeljebb még egyszer alkalmazva minden erdő két fából áll. Keressünk két csúcsot, u-t és v-t, melyek lefedik a 6 darab fát, mindegyikük pontosan hármat. Könnyű látni, hogy mindenképpen van 2 olyan erdő, amelyre találunk két csúcsot, ami $2 - 2$ különböző fában van. Legyen $u \in T_1 \cap T_3$, $v \in T_2 \cap T_4$. A harmadik erdő két fája lehet olyan helyzetű, hogy az egyik csúcshalmaza megegyezik $(T_1 \cap T_3) \cup (T_2 \cap T_4)$-gyel. Ekkor

éláthelyezésekkel megoldható, hogy legyen két pontunk, amelyek lefedik a 6 fát. Ekkor irányítsuk G éleit a megfelelő fákban u illetve v felé. Így minden fában minden csúcs kifoka egy lesz, kivéve u-t és v-t. Ezen két csúcs kifoka nulla lesz, az egész irányított G-ben is. Viszont minden más csúcs 3 erdőben lesz benne, így a kifoka pontosan 3. Az előbb láttuk, hogy ez meghatároz egy karomfelbontást. □

1.4.4. Tétel. *[1] Minden legalább* 5 *csúcsú G maximális síkgráfnak van Tutte-irányítása.*

Bizonyítás. Brooks tétele szerint egy gráf kromatikus száma kisebb vagy egyenlő, mint a maximális fokszám (bizonyítás található a [7] könyvben). A feltételek között szerepel, hogy összefüggő legyen és nem lehet teljes, vagy páratlan hosszú kör. A 3 és 4 csúcsú maximális síkgráf a teljes gráf, ezekre nem vonatkozik a tétel emiatt. Könnyen látható, hogy K_4-nek nincs Tutte-irányítása (minden csúcs ki- illetve befoka csak 0 vagy 3 lehetne, ami nem lehetséges). Használjuk Brooks tételét G duálisára. Mivel G minden tartománya háromszög, ezért G^* 3-reguláris, vagyis 3-színezhető. Ez azt jelenti, hogy G tartományai is színezhetők 3 színnel. A színezés segítségével definiálhatunk egy irányítást. Minden élre két háromszög-tartomány illeszkedik. Egy adott e él esetén forgassuk a gráfot úgy, hogy e vízszintes legyen. Ekkor a két tartományt nevezzük alsónak és felsőnek. Ha az alsó színe i és felső színe $i+1 \pmod 3$, akkor irányítsuk az e élet jobbról balra, ellenkező esetben balról jobbra (ez az irányítás egyértelmű, ha a gráfot elforgatjuk 180°-kal, akkor is ugyanazt az irányítást kapjuk). Ekkor nézzük egy v csúcsban az élek irányítását. Tegyük fel, hogy T tartomány szomszédos v-vel, és színe 1. Ekkor járjuk körbe a v-vel szomszédos tartományokat az óramutató járásával ellentétes irányban. Az élek irányítását meghatározza a bejárt tartományok sorrendje. A sorozatban következő tartomány színe mindig vagy eggyel nagyobb, vagy eggyel kisebb (mod 3), mint a sorozatban előtte levőé. De tudjuk, hogy 1-esből indulunk, és 1-esbe érkezünk, vagyis a a növekedések és csökkenések számának különbsége osztható 3-mal, vagyis Tutte-irányítást kaptunk. □

A következő tétel már szorosabb kapcsolatot mutat a két konstans között:

1.4.5. Tétel. *[2] Ha minden* 8*-élösszefüggő egyszerű gráfnak, melynek élszámát osztja a három, létezik karomfelbontása, akkor minden* 8*-élösszefüggő gráfnak létezik Tutte-irányítása. Másként, ha $k_c \leq 8$, akkor $k_t \leq 8$.*

Bizonyítás. Tegyük fel, hogy minden 8-élösszefüggő egyszerű gráfnak van karomfelbontása. Ebből csúcsszám szerinti indukcióval fogjuk belátni, hogy minden 8-élösszefüggő gráfnak van Tutte-irányítása.

A kétcsúcsú, nyolcélű gráfnak van Tutte-irányítása ($4-4$ élet irányítunk az egyes csúcsok felé). Továbbá feltehetjük, hogy G 2-összefüggő, vagyis nincs elvágó pontja. Ha lenne, akkor minden blokkra használhatnánk a tételt. Könnyen beláthatjuk, hogy elegendő többszörös élek nélküli gráfokat vizsgálnunk. Legyen két párhuzamos él e_1 és e_2, illetve a két pont, ami között futnak, u és v. Ekkor húzzuk össze u-t és v-t egy ponttá. Így a keletkező G' gráf is 8-élösszefüggő, így az indukció szerint létezik Tutte-irányítása. Ekkor az x és y között futó élek kivételével az összes G-beli élet irányítsuk a kapott G'-beli irányítás szerint. Az x és y közötti, e_1-től és e_2-től különböző éleket irányítsuk véletlenszerűen. Ezután irányítsuk úgy e_1-et és e_2-t, hogy x kiegyensúlyozott legyen mod 3. Ezt megtehetjük, mert a két él miatt x kifoka 0-val, 1-gyel, vagy 2-vel változhat. Ekkor y is kiegyensúlyozott lesz mod 3, mivel nem lehet pontosan egy kiegyensúlyozatlan csúcs (a kifokok és befokok összege egyenlő a teljes gráfban). Vagyis feltehetjük, hogy G nem tartalmaz többszörös éleket.

Tegyük fel, hogy $v \in V$ fokszáma páros. Ekkor használjuk Lovász és Mader tételét [9,11], ami szerint létezik két él, vx és vy, melyeket lecserélhetünk egy xy élre úgy, hogy $V \setminus \{v\}$ bármely két csúcsa közötti élösszefüggőségi szám ne változzon. Mivel v fokszáma páros, így minden rá illeszkedő élet lecserélhetünk. Így kapunk egy kisebb csúcsszámú gráfot, amire használhatjuk az indukciós feltevést. Amikor visszatérünk az eredeti gráfra, akkor csak egyszerűen az xy irányított él helyett az xv és vy irányított éleket vesszük. Így $V \setminus \{v\}$ csúcsai kiegyensúlyozottak maradnak, illetve v is, mivel kifoka egyenlő a befokával.

Így már csak azokat a gráfokat kell néznünk, ahol minden fokszám páratlan. Legyen v egy páratlan fokú csúcs, fokszáma $2k+9$ (a 8-élösszefüggőség miatt minden csúcs foka legalább 8). Ha $k > 0$, akkor két élet lecserélünk, így csökken az élek száma, és a gráf 8-élösszefüggő marad, ha ennek van Tutte-irányítása, akkor az eredetinek is. Ezt ismételve egy 9-reguláris gráfhoz juthatunk, melynek nincsenek többszörös élei. Erre alkalmazhatjuk a bizonyítás eleji feltevésünk, vagyis létezik karomfelbontása ennek a gráfnak. A karomfelbontás megfelel egy irányításnak, ahol minden kifok 3-nak többszöröse. Mivel a gráf 9-reguláris, ezért a befok is 3-többszörös, vagyis minden csúcs kiegyensúlyozott mod 3, létezik Tutte-irányítás. □

1.4.6. Megjegyzés. *A bizonyításból látjuk, hogy* 8 *helyett* $8k+6$*-ra is igaz a tétel,*

vagyis k_c létezéséből következik k_t létezése is.

Az irányítások terén általánosíthatjuk a kérdést. Ehhez definiáljunk egy w függvényt G csúcsain $w : V(G) \to \{0,1,2\}$, melyre $\sum_{v \in V(G)} w(v) \equiv |E(G)| \pmod 3$. Ha létezik olyan irányítása G éleinek, hogy $d^+(v) \equiv w(v) \pmod 3$ minden $v \in V$-re, akkor azt mondjuk, hogy G meghatároz egy w által előírt általánosított Tutte-irányítást. Ha minden, a feltételnek megfelelő w függvényre létezik ilyen irányítás, akkor azt mondjuk, hogy G meghatározza az összes általános Tutte-irányítást.

1.4.7. Sejtés. *[2] Létezik egy legkisebb természetes szám, k_g, hogy minden k_g-élösszefüggő gráf meghatározza az összes általánosított Tutte-irányítást.*

Látjuk, hogy $k_c \leq k_g$, elegendő a konstans nulla w függvényt vennünk. Továbbá $k_t \leq k_g$, ehhez olyan w-t választunk, ahol minden csúcsra w fele a fokszámnak mod 3 (ha $d(v) \equiv 0 \bmod 3 \to w(v) = 0$, $1 \to 2$, $2 \to 1$). A következő tétel szerint a három konstans lényegében ekvivalens:

1.4.8. Tétel. *[2] Ha k_c, k_t, k_g közül legalább egy létezik, akkor mindhárom létezik. Ekkor $k_g \leq 2k_t + 2$, $k_c \leq k_g$, $k_t \leq k_c + 5$.*

Bizonyítás. Már láttuk, hogy k_g létezéséből következik k_t és k_c létezése. Az 1.4.6. megjegyzés szerint k_c létezéséből következik k_t létezése, továbbá $k_t \leq k_c + 5$, mivel ha $8 + 6k \leq k_c < 8 + 6(k+1)$, akkor $k_t = 8 + 6k$. Már csak azt kell belátnunk, hogy k_t létezéséből következik k_g létezése. A tétel szerint $k_g \leq 2k_t + 2$. Tegyük fel, hogy létezik k_t és tekintsünk egy legalább $(2k_t + 2)$-élösszefüggő G gráfot, és legyen $w : V(G) \to \{0,1,2\}$ tetszőleges függvény G csúcsain. Felhasználva ismét az 1.2.7. következményt kapjuk, hogy G-nek létezik $(k_t + 1)$ darab éldiszjunkt feszítőfája, legyenek ezek $T_1, T_2, \ldots, T_{k_t+1}$.

Legyen $w^*(v) = -d_G(v) - w(v)$, minden $v \in V$-re. Ezután irányítsuk T_{k_t+1} néhány élét úgy, hogy $d_F^+(v) - d_F^-(v) \equiv w^*(v) \bmod 3$, minden $v \in V$-re, ahol F jelöli a megirányított élek erdőjét. Könnyen látható, hogy ilyen részleges irányítás létezik. Vizsgáljuk az irányítatlanul maradt éleket, ezek alkotta gráf legyen H. Mivel van k_t darab feszítőfája, így H legalább k_t-élösszefüggő. Feltevésünk szerint ennek létezik Tutte-irányítása, vagyis $d_H^+(v) \equiv d_H^-(v) \pmod 3$, minden $v \in V$-re. Ez egy jó irányítás lesz a következőek miatt:

$$d_H(v) = d_H^+(v) + d_H^-(v) \equiv 2d_H^+(v) \equiv -d_H^-(v) \; mod \; 3$$

$$d_F(v) = d_F^+(v) + d_F^-(v) = 2d_F^+(v) - (d_F^+(v) - d_F^-(v)) \equiv -d_F^+(v) - w^*(v) \; mod \; 3$$

$$d_G^+(v) = d_H^+(v) + d_F^+(v) \equiv -d_H^-(v) - d_F(v) - w^*(v) = -d_G(v) - w*(v) = w(v) \; mod \; 3$$

Vagyis létezik k_g és $k_g \leq 2k_t + 2$. □

Vezessük be az egész folyamok fogalmát:

1.4.9. Definíció. *Legyen G egy gráf és k egy természetes szám. Egy (D, f) pár egy egész k-folyama G-nek, ha D egy irányítása G-nek, és $f : E(G) \mapsto \{0, \pm 1, \ldots, \pm(k-1)\}$, hogy*

$$\sum_{e \in E^+(v)} f(e) = \sum_{e \in E^-(v)} f(e)$$

minden $v \in V$ csúcsra. Továbbá azt mondjuk, hogy a (D, f) k-folyam sehol sem nulla, ha $f(e) \neq 0$ minden $e \in E$-re.

Egy sehol sem nulla 3-folyam meghatároz egy Tutte-irányítást, ha azokat az éleket, amelyen a folyam 2, megfordítjuk. Többen is vizsgálták, hogy mely gráfokhoz létezik sehol sem nulla 3-folyam. Több sejtés is létezik ezzel kapcsolatban:

1.4.10. Sejtés (Tutte)**.** *[17] Minden 4-élösszefüggő gráfhoz létezik sehol sem nulla 3-folyam.*

Ennek gyengített változata származik Jaegertől:

1.4.11. Sejtés (Jaeger)**.** *[4] Létezik egy h természetes szám, hogy minden h-élösszefüggő gráfnak létezik sehol sem nulla 3-folyama.*

Thomassen bizonyította a következőt:

1.4.12. Tétel (Thomassen)**.** *[16] Ha G egy 8-élösszefüggő gráf, akkor G meghatározza az összes általánosított Tutte-irányítást.*

Ebből következnek az 1.4.8. tétel szerint:

1.4.13. Következmény. *Ha G egy 8-élösszefüggő gráf, akkor meghatároz egy Tutte-irányítást.*

1.4.14. Következmény. *Ha G egy 8-élösszefüggő egyszerű gráf, akkor létezik karomfelbontása.*

A eddigi legerősebb eredmények Lovász, Thomassen, Wu és Zhang cikkéből:

1.4.15. Tétel. *[10] Minden 6-élösszefüggő gráfnak létezik sehol sem nulla 3-folyama.*

1.4.16. Következmény. *Minden 6-élösszefüggő gráf meghatároz egy Tutte-irányítást.*

1.5. További általánosítások

Az 1.4.1. sejtést általánosíthatjuk minden fára:

1.5.1. Sejtés. *[2] Minden T fára létezik egy természetes szám k_T, hogy minden k_T-élösszefüggő gráfra, melyre $|E(T)|$ osztja $|E(G)|$-t, G-nek létezik T-felbontása.*

Vizsgáltuk a 3-élű utakat, és találtunk 2-élösszefüggő gráfot, melynek nincs P_4 felbontása. Ezt általánosíthatjuk a következő módon. Keressünk ellenpéldát P_n-hez, ahol n páratlan. Vegyünk egy $(2n-2)$ csúcsú $(n-2)$-reguláris gráfot. Ennek $\frac{(2n-2)(n-2)}{2}$ éle van, vagyis $(n-2)$ darab P_n-t kellene elhelyeznünk. Mivel minden csúcs foka páratlan (reguláris $(n-2)$, ahol $(n-2)$ páratlan), ezért minden csúcsban legalább egy út végződik (egy útban csak a végpontok foka páratlan). Mivel $(n-2)$ útnak $(2n-4)$ végpontja van, és a gráfnak $(2n-2)$ csúcsa, így nem lefedhető a gráf. Ha n páros, akkor ehhez hasonlóan vegyünk egy $K_{n-1,n-1}$ teljes páros gráfot. Ebben létezik egy Hamilton-kör, ennek éleit hagyjuk el a gráfból. Így megmarad egy $(2n-2)$ csúcsú, $(n-3)$-reguláris (minden csúcs foka 2-vel csökkent) és $(n-3)$-élösszefüggő gráf. Ennek élszáma $\frac{(2n-2)(n-3)}{2}$, így $(n-3)$ darab utat kell elhelyeznünk benne. Minden csúcs foka páratlan, mivel n páros és $(n-3)$-reguláris a gráf, így minden csúcsra illeszkedni kell egy út végpontjának. Mivel $(n-3)$ útnak csak $(2n-6)$ végpontja van, így nem fedhetjük le a gráfot.

A karom mintájára vizsgálhatjuk a különböző méretű "csillagokat", vagyis $K_{1,m}$-et minden természetes m-re. Thomassen eredménye az ehhez kapcsolódó felbontásokra:

1.5.2. Tétel. *[16] Legyen m tetszőleges természetes szám. Ha G egy $(2m^2+2)$-élösszefüggő gráf és élszáma többszöröse m-nek, akkor van $K_{1,m}$ felbontása.*

A 7. ábra ellenpéldája alapján gyárthatunk hasonló példákat $K_{1,m}$-hez is. Vegyünk ehhez m darab K_{m+1}-et, a 7. ábra mintájára helyezzük el őket egy kör mentén, és a szomszédosak között húzzunk be $\left\lceil \frac{m+1}{2} \right\rceil$ darab élet úgy, hogy minden csúcsra legalább egy összekötő él is illeszkedjen. Így a gráf $(m+1)$-élösszefüggő lesz, mert ha

az összekötő élek mentén szeretnénk szétválasztani a gráfot, akkor legalább $2\left\lceil\frac{m+1}{2}\right\rceil$ élet kell elvágnunk, ez több mint m. Ha egy teljes gráfon szeretnénk megszüntetni az összefüggőséget, akkor is legalább $(m+1)$ élet kell eltávolítanunk, mert minden csúcs fokszáma legalább ennyi. A gráfnak összesen $m\binom{m+1}{2}+m\left\lceil\frac{m+1}{2}\right\rceil$ darab éle van, így $\frac{m(m+1)}{2}+\left\lceil\frac{m+1}{2}\right\rceil$ darab $K_{1,m}$-et kell elhelyeznünk. Mivel összesen $m(m+1)$ csúcsa van a gráfnak, így a skatulyaelv szerint lesz olyan K_{m+1}, melyben van két pont, amely nem középpontja valamely csillagnak. A köztük futó él így fedetlen marad.

A Tutte-irányítást is általánosíthatjuk ennek mintájára:

1.5.3. Definíció. *Egy G gráfot mod $(2p+1)$-irányíthatónak mondunk, ha létezik olyan irányítása, amiben minden csúcs kiegyensúlyozott mod $(2p+1)$, vagyis minden $v \in V$-re $d^+(v) \equiv d^-(v)$ mod $(2p+1)$.*

Jaeger sejtése ezzel kapcsolatban az 1.4.2. sejtéshez hasonlóan:

1.5.4. Sejtés. *[5] Minden $p \geq 1$-re létezik egy legkisebb $k_j(p)$ természetes szám, hogy minden $k_j(p)$-élösszefüggő gráfnak létezik mod $(2p+1)$-irányítása.*

2. Élösszefüggőség

Eddigiekben láttuk, hogy különböző felbontások létezéséhez elégséges feltételt adhatunk a gráf élösszefüggőségi számával. Még nem ejtettünk szót arról, hogyan határozhatjuk meg egy gráf élösszefüggőségi számát, illetve miket állíthatunk adott élösszefüggőség esetén gráfokról. Már korábban is használtunk egy triviális állítást, hogy egy k-élösszefüggő gráf minden csúcsának foka legalább k. Ha létezne csúcs kisebb fokkal (legfeljebb $k-1$), akkor a vele szomszédos éleket elhagyva a gráf több komponensre esne szét. Ebből már következtetni tudunk a minimális élszámra is, ugyanis ha n csúcsa van a gráfnak és k-élösszefüggő, akkor legalább $\frac{n \cdot k}{2}$ éle van.

Az élösszefüggőségi szám megállapítására használhatjuk a Ford-Fulkerson algoritmust.

2.1. Ford-Fulkerson algoritmus

Az algoritmust csak vázlatosan ismertetem, a pontos leírás megtalálható [7] könyvben. Az algoritmus maximális folyamot keres. Ehhez bemenetként szükséges egy gráf, és az élein egy kapacitásfüggvény, illetve két kitüntetett pont egy forrás és egy nyelő (ezeket s-sel és t-vel fogjuk jelölni a későbbiekben). A kapacitásfüggvény minden élen nem-negatív. A folyam egy függvény az éleken, amely rendelkezik a következő tulajdonságokkal:

- minden élen kisebb vagy egyenlő, mint a kapacitásfüggvény értéke
- a forrás és nyelő kivételével minden csúcsban 0 a folyam összege
- az s-ből induló, és a t-be érkező folyamok összege egyenlő, ezt nevezzük a folyam értékének

Utóbbit maximalizálja az algoritmus.

Az algoritmus lépései:

1. Legyen a folyam értéke minden élen 0.
2. Javítóutakat keresünk s és t között, vagyis olyan utakat, melyek mentén növelhetjük a folyam értékét.

3. Ha már nem létezik javítóút, akkor maximális folyamot kaptunk.

A javítóút kereséséhez szélességi keresést használunk.

Ahhoz, hogy az élösszefüggőséget meg tudjuk határozni szükségünk van két további tételre.

2.1.1. Definíció. *Vágásnak nevezünk egy $X \subset V$ ponthalmazt, melyre $s \in X$, de $t \notin X$. Az X vágás értéke az X-ből kilépő élek kapacitásainak összege. Az X vágás értékét $c(X)$-szel jelöljük.*

2.1.2. Tétel (Ford-Fulkerson)**.** *A maximális értéke egy $s-t$ folyamnak egyenlő a minimális $s-t$ vágás értékével.*

2.1.3. Tétel. *Ha minden élkapacitás egészértékű, akkor a maximális folyam értéke is egész, és ez megvalósítható olyan f függvénnyel is, mely minden élen egész értéket vesz fel.*

Ezek alapján már könnyen meghatározható az élösszefüggőségi szám. Ha a gráf minden élén a kapacitást 1-nek választjuk, nem-élen pedig 0-nak, akkor a maximális folyam egyenlő a minimális vágással, vagyis megkapjuk az elvágó élek minimális számát(ami s-t és t-t elvágja). Ha ezt megvizsgáljuk egy pont és minden más pont között, akkor ezek közül a legkisebb megadja a gráf elvágó élhalmazának minimális számát. Ez megegyezik az élösszefüggőségi számmal, hiszen ennél kevesebb élet elhagyva még összefüggő marad a gráf.

3. Dokumentáció

Az eddig látott eredményeket megpróbáljuk komputeres számításokkal javítani. A következő kérdéseket vizsgáljuk:

1. 5-élösszefüggő gráfok karomfelbontása
2. 3-élösszefüggő gráfok 5-partícionálása
3. 4-élösszefüggő gráfok 5-élű utakra bontása

Az első kérdéssel kapcsolatban már láttuk, hogy a 7. ábrán látható ellenpélda miatt $k_c \geq 5$. Később az 1.4.14. következmény miatt pedig tudjuk, hogy $k_c \leq 8$. Ezt az eredményt szeretnénk pontosítani, olyan gráfot keresünk, mely 5-élösszefüggő, és nincs karomfelbontása.

A második probléma az 1.2.8. tétel eredményét hivatott pontosítani. Láttuk, hogy a 4-élösszefüggőség elégséges az 5-partíció létezéséhez, de nem tudjuk, hogy szükséges-e. Amennyiben tudnánk mutatni 3-élösszefüggő, 5-partícióval nem rendelkező gráfot, akkor már szükséges és elégséges feltétele lenne az 5-partíció létezésének a 4-élösszefüggőség.

Az 5-élű útról tudjuk, hogy létezik 3-élösszefüggő gráf, amelynek nincs P_6 felbontása. Erre mutattunk konstrukciót. A 4-élösszefüggőségről még nem tudjuk, hogy elégséges-e a felbontás létezéséhez.

A teszteléshez szükségünk van nagyszámú gráfra, melyeknek ismerjük az élösszefüggőségi számát, illetve szubrutinra, mely ellenőrzi, hogy felbontható-e a bemenetként kapott gráf.

Ehhez a következő alprogramokat használjuk:

geng

A gráfok létrehozásához felhasználjuk Brendan McKay "geng" nevű programját [12]. A program adott tulajdonságú gráfok generálására alkalmas, ebből amire szükségünk van, hogy adott csúcsszámú, adott élszámú, adott minimális fokszámú gráfokat tud generálni. A k-élösszefüggőségnek szükséges feltétele, hogy minden csúcs foka legalább k legyen, így csak ilyen gráfokat generálunk, praktikusan kis csúcsszámon, és

megfelelő élszámmal. A program által generált gráfok nem izomorfak, így elkerüljük az izomorf gráfokra való többszöri futtatás miatti kapacitásveszteséget.

A program graph6 formátumban szolgáltatja a gráfokat, ezt az ugyancsak Mckay által írt "showg" programmal tudjuk szomszédsági mátrix alakra hozni. A mátrix sorfolytonosan szerepel az outputban.

fordfulk

Az így kapott gráfokra még ellenőriznünk kell, hogy pontosan mennyi az élösszefüggőségi számuk. Ehhez használhatjuk a Ford-Fulkerson (maximális folyam - minimális vágás) tételét, amely szerint ha minden él kapacitása 1, akkor a minimális elvágó élhalmaz elemszáma megegyezik a maximális folyam értékével. Ebben az esetben a maximális folyam értéke pontosan a keresett összefüggőségi szám (3 vagy 4 vagy 5). Ennek megállapítására szolgál a Ford-Fulkerson algoritmus, melyet ha futattunk egy pontból az összes többibe, akkor a kapott legkisebb maximális folyamérték megadja az élösszefüggőségi számot. Így ki tudjuk szűrni a nem megfelelő élösszefüggőségi számmal rendelkező gráfokat. A kimenetben minden gráfnak adunk egy sorszámot, majd tabulátorral elválasztva az szomszédsági mátrixot sorfolytonosan, annak elemeit nem elválasztva (mivel minden karakter 0 vagy 1, így nem szükséges közéjük más elválasztást tennünk). A program kódja megtalálható a 4.1. alfejezetben.

claws

Az első kérdéshez használt program, a bemenetként kapott gráf karomfelbontását végzi el. A program generálja a lehetséges karmokat, ehhez megvizsgálja csúcsonként, annak szomszédait. Ha legalább 3 van, akkor generál egy karmot, melyben eltárolja ezt a 4 pontot, megkülönböztetve a 3 fokú csúcsot (ha 3-nál több szomszédja van, akkor az összes hármashoz generál $1 - 1$ karmot). Ezután megvizsgálja, hogy ezeknek valamely kombinációja fedi-e a gráfot. Ehhez verem adatszerkezetet használ, a verembe helyezve azon karmokat, amik megtalálhatóak a gráfban(adott részfedés mellett). Ha így egy nem teljes felbontáshoz jut, akkor kivesz egy elemet a veremből, annak felhasználása nélkül keres tovább. A program futása végén vagy talál egy felbontást, vagy végignézve az összes karomelhelyezést ellenpéldát talál. A program inputjának formátuma megegyezik a "fordfulk" program outputjával, vagyis egy sorszám után a szomszédsági mátrix sorfolytonosan. Az outputban az inputból

származó sorszám, és a futás eredménye látható: ”felbonthato” vagy ”ellenpelda” (ellenőrzési célzattal a program megvizsgálja, hogy az élszám osztható-e hárommal, amennyiben nem akkor felbontás keresése nélkül leáll és ”nem oszthato” kerül az outputba). A program kódja megtalálható a 4.2. alfejezetben.

details

A ”claws” program működésének helyességét ellenőrizhetjük vele. Inputja megegyezik a ”claws” program inputjával, outputja bővebb. A sorszámával adott gráfot megjeleníti szomszédsági mátrix formájában, majd megadja annak karomfelbontását, ha az létezik. Ezáltal ellenőrizhető a program helyessége, illetve tényleges bizonyítékot kapunk a felbontás létezésére. Látjuk a gráf sorszámát az adott fájlban, a gráf éleinek,

```
1
Number of edges: 42
Number of claws: 14
0       0       0       1       1       0       0       1       1       1
0       0       0       0       0       1       1       1       1       1       1
0       0       0       0       0       1       1       1       1       1       1
0       0       0       0       0       0       1       1       1       1       1
1       0       0       0       0       0       1       1       1       1       1
1       1       1       0       0       0       1       1       1       1       1
0       1       1       1       1       1       0       1       1       1       1
0       1       1       1       1       1       1       0       1       1       1
1       1       1       1       1       1       1       1       0       1       1
1       1       1       1       1       1       1       1       1       0       1
1       1       1       1       1       1       1       1       1       1       0
felbonthato
14
(10;5,6,8)
(10;0,3,4)
(9;5,8,10)
(9;0,3,4)
(7;8,9,10)
(6;7,8,9)
(5;6,7,8)
(4;6,7,8)
(3;6,7,8)
(2;8,9,10)
(2;5,6,7)
(1;8,9,10)
(1;5,6,7)
(0;4,5,8)
```

8. ábra. A details egy futásának eredménye

és az elhelyezendő karmoknak a számát, illetve a szomszédsági mátrixot. Ezután magát a program eredményét, vagyis hogy felbontható a gráf, 14 karmot helyeztünk el benne, majd a karmok. A karmok megjelenítésének formája: $(w; x, y, z)$, ahol w a 3-fokú csúcs, x,y,z az egyfokúak (a csúcsok számozása 0-val kezdődik). A program kódja megtalálható a 4.3. alfejezetben.

5conn

Az 5-élű összefüggő részgráfokra való felbontáshoz készített program. Itt a 3-élösszefüggő gráfok között keresünk. A ”claws”-hoz hasonlóan generálja a lehetséges részgráfokat, majd keres felbontást. A lehetséges részgráfok generálása során az összes lehetséges él ötösre megvizsgálja, hogy azok összefüggő részgráfot alkotnak-e. Ehhez ellenőrzi, hogy hány pontot fed az 5 él. Ha több, mint 6-ot, akkor nem lehet összefüggő a részgráf. Ha legfeljebb 5-öt, akkor biztosan összefüggő. Mikor pontosan 6 csúcsot fednek akkor pedig alkalmazható mohó algoritmus, egy élből véve azokat, amelyekkel van közös végpontjuk, és ezt a lépés ismételve vagy minden élet elérünk, így összefüggő a részgráf, vagy pedig nem. A felbontáshoz itt is vermes megoldást használunk. Mivel 3-élösszefüggő gráfokon végezzük a vizsgálatot, ezért először a 3-reguláris gráfokat nézzük, melyeknek 5 többszörös az élszáma. Ezeknek a csúcsszáma osztható 10-zel, mivel egy n csúcsú 3-reguláris gráfnak $\frac{3 \cdot n}{2}$ éle van, ami pontosan akkor osztható 5-tel, ha n osztható 2-vel és 5-tel is. Így először a 10, illetve 20 csúcsú gráfokat nézzük. A program kódja megtalálható a 4.4. alfejezetben.

P6

Az 5 hosszú utakra írt program lényegében az előzőnek a módosítása. Itt is a lehetséges él ötösökre ellenőrizzük, hogy azok összefüggőek-e, és ha azok megvizsgáljuk, hogy utat alkotnak-e. Ehhez egy kicsit bonyolultabb algoritmust kell használnunk. Az ”5conn”-hoz hasonlóan itt is először megvizsgáljuk, hogy összefüggő-e az 5 él, ha nem akkor nem kell tovább foglalkoznunk vele. Amennyiben összefüggőek elegendő megnéznünk a fokszámokat az 5 él által meghatározott részgráfban. Ha 2 darab 1-fokú csúcs és 4 darab 2-fokú csúcs van, akkor csak út lehet a részgráf (ha előbb nem ellenőriznénk az összefüggőséget, akkor lehetne egy C_3 és egy P_3 is a gráf a fokszámok alapján. Ezután a program futása az eddig is használt vermes módszerrel keresi a lehetséges részgráfokból a felbontást. A program kódja megtalálható a 4.5. alfejezetben.

3.1. Lefuttatott tesztek

Karomfelbontások

6 csúcsú egyszerű gráfok (minimális fokszám 5):

- 1 generált gráf
- 1 5-élösszefüggő
- 0 ellenpélda

7 csúcsú egyszerű gráfok (minimális fokszám 5):

- 4 generált gráf
- 3 5-élösszefüggő
- 0 ellenpélda

8 csúcsú egyszerű gráfok (minimális fokszám 5):

- 46 generált gráf
- 24 5-élösszefüggő
- 0 ellenpélda

9 csúcsú egyszerű gráfok (minimális fokszám 5):

- 1165 generált gráf
- 361 5-élösszefüggő
- 0 ellenpélda

10 csúcsú egyszerű gráfok (minimális fokszám 5):

- 108373 generált gráf
- 37183 5-élösszefüggő
- 0 ellenpélda

11 csúcsú egyszerű gráfok (minimális fokszám 5):

- 30 él: 325320 generált gráf, 238226 5-élösszefüggő, 0 ellenpélda
- 33 él: 3345048 generált gráf, 1393175 5-élösszefüggő, 0 ellenpélda

- 36 él: 3155470 generált gráf, 613818 5-élösszefüggő, 0 ellenpélda
- 39 él: 722368 generált gráf, 53175 5-élösszefüggő, 0 ellenpélda
- 42 él: 64862 generált gráf, 29209 5-élösszefüggő, 0 ellenpélda
- 45 él: 3436 generált gráf, 703 5-élösszefüggő, 0 ellenpélda
- 48 él: 168 generált gráf, 12 5-élösszefüggő, 0 ellenpélda
- 51 él: 11 generált gráf, 0 5-élösszefüggő, 0 ellenpélda
- 54 él: 1 generált gráf, 0 5-élösszefüggő, 0 ellenpélda

12 csúcsú egyszerű gráfok (minimális fokszám 5):

- 33 él: 32834010 darab generált gráf, 0 ellenpélda

5-partíciók

10 csúcsú 3-reguláris gráfok:

- 19 generált gráf
- 14 3-élösszefüggő
- 0 ellenpélda

20 csúcsú 3-reguláris gráfok:

- 510489 generált gráf
- 396150 3-élösszefüggő
- 0 ellenpélda

5 hosszú utak

10 csúcsú egyszerű gráfok (minimális fokszám 4):

- 20 él: 59 generált gráf, 58 4-élösszefüggő, 0 ellenpélda
- 25 él: 152370 generált gráf, 152310 4-élösszefüggő, 0 ellenpélda

4. Függelék

4.1. Fordfulk

```
#include <iostream>
#include <stdio.h>
#include <fstream>
using namespace std;
#define WHITE 0
#define GRAY 1
#define BLACK 2
#define MAX_NODES 11
#define oo 1000000000
int min(int x, int y) {
    return x<y ? x : y;
}
int head,tail;
int n;
    int e;
int capacity[MAX_NODES][MAX_NODES];
int flow[MAX_NODES][MAX_NODES];
int color[MAX_NODES];
int pred[MAX_NODES];
int q[MAX_NODES+2];
void enqueue (int x) {
    q[tail] = x;
    tail++;
    color[x] = GRAY;
}
int dequeue () {
    int x = q[head];
    head++;
    color[x] = BLACK;
    return x;
}
int bfs (int start, int target) {
    int u,v;
    for (u=0; u<n; u++) {
      color[u] = WHITE;
    }
    head = tail = 0;
    enqueue(start);
    pred[start] = -1;
    while (head!=tail) {
        u = dequeue();
      for (v=0; v<n; v++) {
            if (color[v]==WHITE && capacity[u][v]-flow[u][v]>0) {
                enqueue(v);
                pred[v] = u;
            }
        }
    }
    return color[target]==BLACK;
}
int max_flow (int source, int sink) {
    int i,j,u;
    int max_flow = 0;
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
      flow[i][j] = 0;
        }
    }
```

```
59      int c=0;
60      while (bfs(source,sink)) {
61          c++;
62          int increment = oo;
63          for (u=sink; pred[u]>=0; u=pred[u]) {
64              increment = min(increment,capacity[pred[u]][u]-flow[pred[u]][u]);
65          }
66          for (u=sink; pred[u]>=0; u=pred[u]) {
67              flow[pred[u]][u] += increment;
68              flow[u][pred[u]] -= increment;
69          }
70          max_flow += increment;
71      }
72      return max_flow;
73  }
74  int main()
75  {
76      n=MAX_NODES;
77      char c;
78      ifstream is("inputfile.txt");
79      ofstream of("outputfile.txt");
80      int counter=1;
81      for(int k=1;k<=number_of_graphs;k++)
82      {
83          cout<<k<<'\n';
84        c='a';
85        while (c!='.')
86        {
87            is>>c;
88        }
89        for(int i=0;i<MAX_NODES;i++)
90        {
91            for(int j=0;j<MAX_NODES;j++)
92            {
93                    char y;
94                    is>>y;
95                    capacity[i][j]=y-48;
96            }
97        }
98        int K[MAX_NODES];
99        for(int i=1;i<MAX_NODES;i++)
100       {
101           K[i]=max_flow(0,i);
102       }
103       int Min=K[1];
104       for(int i=1;i<MAX_NODES;i++)
105       {
106           if(K[i]<Min)
107           {
108                   Min=K[i];
109           }
110       }
111       if(Min==5)
112       {
113           of<<counter<<'\t';
114           for(int ii=0;ii<MAX_NODES;ii++)
115           {
116                   for(int jj=0;jj<MAX_NODES;jj++)
117                   {
118                       of<<capacity[ii][jj];
119                   }
120           }
121         of<<'\n';
122         counter++;
```

```
123         }
124     }
125     return 0;
126 }
```

4.2. Claws

```
1  #include <iostream>
2  #include <string.h>
3  #include <vector>
4  #include <stack>
5  #include <fstream>
6  using namespace std;
7  #define N 11 //number of vertices
8  class Claw{
9      public:
10     int head;
11     int t1;
12     int t2;
13     int t3;
14     Claw(int,int,int,int);
15     void toString() {cout<<"("<<head<<";"<<t1<<","<<t2<<","<<t3<<")";}
16 };
17 Claw::Claw (int a,int b, int c, int d)
18 {
19     head=a;
20     t1=b;
21     t2=c;
22     t3=d;
23 }
24 bool applicable(Claw c,int* a)
25 {
26     return (a[c.head*N+c.t1]>0 && a[c.head*N+c.t2]>0 && a[c.head*N+c.t3]>0);
27 }
28 bool equals(Claw c1,Claw c2)
29 {
30     return (c1.head==c2.head && c1.t1==c2.t1 && c1.t2==c2.t2 && c1.t3==c2.t3);
31 }
32 void extract(int* a,Claw c)
33 {
34     a[c.head*N+c.t1]--;
35     a[c.head*N+c.t2]--;
36     a[c.head*N+c.t3]--;
37     a[c.t1*N+c.head]--;
38     a[c.t2*N+c.head]--;
39     a[c.t3*N+c.head]--;
40 }
41 void add(int* a,Claw c)
42 {
43     a[c.head*N+c.t1]++;
44     a[c.head*N+c.t2]++;
45     a[c.head*N+c.t3]++;
46     a[c.t1*N+c.head]++;
47     a[c.t2*N+c.head]++;
48     a[c.t3*N+c.head]++;
49 }
50 int main()
51 {
52     ifstream is("inputfile.txt");
53     ofstream of("oupufile.txt");
54     int A[N][N];
55     int a[N];
```

```
56      int sorszam;
57      is>>sorszam;
58      while(!is.eof())
59      {
60      of<<sorszam<<'\t';
61      cout<<sorszam<<'\n';
62      for(int i=0;i<N;i++)
63      {
64          for(int j=0;j<N;j++)
65          {
66              char y;
67              is>>y;
68              A[i][j]=y-48;
69          }
70      }
71      int edges=0;
72      for(int i=0;i<N;i++)
73      {
74          for(int j=0;j<i;j++)
75          {
76              edges=edges+A[j][i];
77          }
78      }
79      int clawnumber = edges/3;
80      if (edges%3!=0)
81      {
82          of<<"nem oszthato";
83      }
84      else{
85      for(int i=0;i<N;i++)
86      {
87          a[i]=0;
88          for(int j=0;j<N;j++)
89          {
90              if(A[j][i]>0)
91              {
92                  a[i]++;
93              }
94          }
95      }
96      vector<Claw> C;
97      int c=0;
98      for(int i=0;i<N;i++)
99      {
100         if(a[i]>=3)
101         {
102             int k=0;
103             int B[a[i]];
104             for(int j=0;j<N;j++)
105             {
106                 if(A[j][i]>0)
107                 {
108                     B[k]=j;
109                     k++;
110                 }
111             }
112             for(int ii=0;ii<a[i];ii++)
113             {
114                 for(int ij=ii+1;ij<a[i];ij++)
115                 {
116                     for(int ik=ij+1;ik<a[i];ik++)
117                     {
118                         C.push_back(Claw(i,B[ii],B[ij],B[ik]));
119                         c++;
```

```
120                     }
121                 }
122             }
123         }
124     }
125     stack<Claw> claws;
126     int counter=0;
127     while(counter<c+1 && claws.size()!=clawnumber)
128     {
129     for(int i=counter;i<c;i++)
130     {
131         if(applicable(C[i],(int*)A))
132         {
133             claws.push(C[i]);
134             extract((int*) A,C[i]);
135             i--;
136         }
137     }
138     if(claws.size()!=clawnumber)
139     {
140         Claw c1=claws.top();
141         add((int*)A,c1);
142         int ind=0;
143         while (!equals(c1,C[ind])&&ind<c+2)
144         {
145             ind++;
146         }
147         counter=ind+1;
148         claws.pop();
149     }
150     }
151     if(claws.size()==clawnumber)
152     {
153         of<<"felbonthato"<<'\r';
154         of<<std::endl;
155     }
156     else{of<<"ellenpelda";cout<<"  !!!!!!!!!!!!!  ";}
157     }
158     is>>sorszam;
159     of<<'\r';
160     }
161     return 0;
162 }
```

4.3. Details

A program nagyrészt megegyezik a ”claws” programmal. Az eltérő részek a következők.

A ”claws” 54-61. sora helyett (megkeresi az n változóban tárolt sorszámú gráfot):

```
1 int n=150;
2 int A[N][N];
3 int a[N];
4 int sorszam;
5 is>>sorszam;
6 while(sorszam==n)
7 {
8     is>>sorszam;
9 }
```

```
of<<sorszam<<'\r';
of<<std::endl;
```

Illetve a "claws" 138-157. sora helyett (részletek kiírása):

```
of<<"Number of edges: "<<edges<<'\r';
of<<"Number of claws: "<<clawnumber<<'\r';
for(int i=0;i<N;i++)
{
    for(int j=0;j<N;j++)
    {
        of<<A[i][j]<<'\t';
    }
    of<<'\r';
    of<<std::endl;
}
if(claws.size()==clawnumber)
{
    of<<"felbonthato"<<'\r';
    of<<std::endl;
}
else{of<<"ellenpelda";cout<<"  !!!!!!!!!!!!!  ";}
is>>sorszam;
of<<claws.size()<<'\r';
of<<std::endl;
while(!claws.empty())
{
    of<<"("<<claws.top().head<<";"<<claws.top().t1<<","<<claws.top().t2<<","<<claws.top().t3<<")"<<'\r';
    of<<std::endl;
    claws.pop();
}
```

4.4. 5conn

```
#include <iostream>
#include <string.h>
#include <vector>
#include <stack>
#include <set>
#include <fstream>
#include <iterator>
using namespace std;
#define N 10 //number of vertices
class Edge{
public:
    int u,v;
    Edge(int a ,int b);
    Edge();
    Edge operator=(Edge &e) {
    u = e.u;
    v = e.v;
    }
};
bool operator<(const Edge &e1,const Edge &e2){return 100*e1.u+e1.v<100*e2.u+e2.v;}
Edge::Edge(int a,int b)
{
    u=a;
    v=b;
}
Edge::Edge()
{
```

```
28      u=0;
29      v=0;
30  }
31  class Subgraph{
32  public:
33      Edge e1,e2,e3,e4,e5;
34      Subgraph(Edge f1,Edge f2,Edge f3,Edge f4,Edge f5);
35      void toString(){cout<< "((" << e1.u << "," << e1.v << ")(" << e2.u << "," << e2.v
            << ")(" << e3.u << "," << e3.v << ")(" << e4.u << "," << e4.v << ")(" << e5.u
            << "," << e5.v << "))";}
36  };
37  Subgraph::Subgraph(Edge f1,Edge f2, Edge f3, Edge f4, Edge f5)
38  {
39      e1=f1;
40      e2=f2;
41      e3=f3;
42      e4=f4;
43      e5=f5;
44  }
45  bool contain (Edge e, int* a)
46  {
47      return (a[e.u*N+e.v]>0);
48  }
49  bool contains(Subgraph s, int* a)
50  {
51      return
            (contain(s.e1,a)&&contain(s.e2,a)&&contain(s.e3,a)&&contain(s.e4,a)&&contain(s.e5,a));
52  }
53  bool equals(Edge e1,Edge e2)
54  {
55      return ((e1.u==e2.u && e1.v==e2.v) || (e1.v==e2.u && e1.u==e2.v));
56  }
57  bool equals(Subgraph s1,Subgraph s2)
58  {
59      return (equals(s1.e1,s2.e1) && equals(s1.e2,s2.e2) && equals(s1.e3,s2.e3) &&
            equals(s1.e4,s2.e4) && equals(s1.e5,s2.e5));
60  }
61  void extract(int* a,Subgraph s)
62  {
63      a[s.e1.u*N+s.e1.v]--;
64      a[s.e1.v*N+s.e1.u]--;
65      a[s.e2.u*N+s.e2.v]--;
66      a[s.e2.v*N+s.e2.u]--;
67      a[s.e3.u*N+s.e3.v]--;
68      a[s.e3.v*N+s.e3.u]--;
69      a[s.e4.u*N+s.e4.v]--;
70      a[s.e4.v*N+s.e4.u]--;
71      a[s.e5.u*N+s.e5.v]--;
72      a[s.e5.v*N+s.e5.u]--;
73  }
74  void add(int* a,Subgraph s)
75  {
76      a[s.e1.u*N+s.e1.v]++;
77      a[s.e1.v*N+s.e1.u]++;
78      a[s.e2.u*N+s.e2.v]++;
79      a[s.e2.v*N+s.e2.u]++;
80      a[s.e3.u*N+s.e3.v]++;
81      a[s.e3.v*N+s.e3.u]++;
82      a[s.e4.u*N+s.e4.v]++;
83      a[s.e4.v*N+s.e4.u]++;
84      a[s.e5.u*N+s.e5.v]++;
85      a[s.e5.v*N+s.e5.u]++;
86  }
87  bool conne(Edge e1, Edge e2)
```

```
88  {
89      if(e1.u==e2.u || e1.u==e2.v || e1.v==e2.u || e1.v==e2.v)
90      {
91          return 1;
92      }
93      else{return 0;}
94  }
95  bool conn(Edge e1,Edge e2,Edge e3, Edge e4, Edge e5)
96  {
97      set<int> vert;
98      vert.insert(e1.u);
99      vert.insert(e1.v);
100     vert.insert(e2.u);
101     vert.insert(e2.v);
102     vert.insert(e3.u);
103     vert.insert(e3.v);
104     vert.insert(e4.u);
105     vert.insert(e4.v);
106     vert.insert(e5.u);
107     vert.insert(e5.v);
108     if(vert.size()==6)
109     {
110         set<Edge> edges;
111         edges.insert(e1);
112         for(int i=1;i<=5;i++)
113         {
114             set<Edge>::iterator it;
115             for(it=edges.begin(); it!=edges.end(); it++)
116         {
117             if(conne(*it,e2))
118         {
119             edges.insert(e2);
120         }
121         if(conne(*it,e3))
122         {
123             edges.insert(e3);
124         }
125         if(conne(*it,e4))
126         {
127             edges.insert(e4);
128         }
129         if(conne(*it,e5))
130         {
131             edges.insert(e5);
132         }
133         }
134         }
135         if(edges.size()==5)
136         {
137             return 1;
138         }
139         else{return 0;}
140     }
141     else{
142         if(vert.size()<6)
143         {
144             return 1;
145         }
146         else{return 0;}
147     }
148 }
149 int main()
150 {
151     ifstream is("inputfile.txt");
```

```
152     ofstream of("outputfile.txt");
153     int A[N][N];
154     int a[N];
155     int sorszam;
156     is>>sorszam;
157     while(!is.eof())
158     {
159     of<<sorszam<<'\t';
160     cout<<sorszam<<'\n';
161     for(int i=0;i<N;i++)
162     {
163         for(int j=0;j<N;j++)
164         {
165             char y;
166             is>>y;
167             A[i][j]=y-48;
168         }
169     }
170     vector<Edge> E;
171     int edges=0;
172     for(int i=0;i<N;i++)
173     {
174         for(int j=0;j<i;j++)
175         {
176             edges=edges+A[j][i];
177             E.push_back(Edge(i,j));
178         }
179     }
180     int subnumber = edges/5;
181     if (edges%5!=0)
182     {
183         of<<"nem oszthato";
184     }
185     else{
186     vector<Subgraph> S;
187     for(int i=0;i<E.size()-4;i++)
188     {
189         for(int ii=i+1;ii<E.size()-3;ii++)
190         {
191             for(int iii=ii+1;iii<E.size()-2;iii++)
192             {
193                 for(int iiii=iii+1;iiii<E.size()-1;iiii++)
194                 {
195                     for(int iiiii=iiii+1;iiiii<E.size();iiiii++)
196                     {
197                         if(conn(E[i],E[ii],E[iii],E[iiii],E[iiiii]))
198                         {
199                             S.push_back(Subgraph(E[i],E[ii],E[iii],E[iiii],E[iiiii]));
200                         }
201                     }
202                 }
203             }
204         }
205     }
206     stack<Subgraph> sg;
207     int counter=0;
208     while(counter<S.size()+1 && sg.size()!=subnumber)
209     {
210         for(int i=counter;i<S.size();i++)
211         {
212             if(contains(S[i],(int*)A))
213                {
214                     sg.push(S[i]);
215                     extract((int*) A,S[i]);
```

```
216                 }
217             }
218             if(sg.size()!=subnumber)
219             {
220                 Subgraph s1=sg.top();
221                 add((int*)A,s1);
222                 int ind=0;
223             while (!equals(s1,S[ind])&&ind<S.size()+1)
224             {
225                 ind++;
226             }
227             counter=ind+1;
228             sg.pop();
229             }
230         }
231         if(sg.size()==subnumber)
232         {
233                 cout<<'\r';
234             of<<"felbonthato"<<'\r';
235         }
236         else{of<<"ellenpelda";cout<<"  !!!!!!!!!!!!!!  ";}
237         }
238         is>>sorszam;
239         of<<'\r';
240         }
241         return 0;
242 }
```

4.5. P6

A kód majdnem teljesen megegyezik a "5conn" program kódjával, eltérés csak a "conn" függvényben van (a "5conn" esetén ez csak azt vizsgálja, hogy az 5 él összefüggő-e, itt azt is, hogy utat alkotnak vagy sem).

```
1  bool conn(Edge e1,Edge e2,Edge e3, Edge e4, Edge e5)
2  {
3      set<int> vert;
4      vert.insert(e1.u);
5      vert.insert(e1.v);
6      vert.insert(e2.u);
7      vert.insert(e2.v);
8      vert.insert(e3.u);
9      vert.insert(e3.v);
10     vert.insert(e4.u);
11     vert.insert(e4.v);
12     vert.insert(e5.u);
13     vert.insert(e5.v);
14     if(vert.size()==6)
15     {
16         set<Edge> edges;
17         set<int> vert2;
18         edges.insert(e1);
19         vert2.insert(e1.u);
20         vert2.insert(e1.v);
21         for(int i=1;i<=5;i++)
22         {
23             set<Edge>::iterator it;
24             for(it=edges.begin(); it!=edges.end(); it++)
25         {
```

```
26          if (conne(*it,e2))
27          {
28              edges.insert(e2);
29              vert2.insert(e2.u);
30              vert2.insert(e2.v);
31          }
32          if (vert2.size()!=edges.size()+1)
33          {
34              return 0;
35          }
36          if (conne(*it,e3))
37          {
38              edges.insert(e3);
39              vert2.insert(e3.u);
40              vert2.insert(e3.v);
41          }
42          if (vert2.size()!=edges.size()+1)
43          {
44              return 0;
45          }
46          if (conne(*it,e4))
47          {
48              edges.insert(e4);
49              vert2.insert(e4.u);
50              vert2.insert(e4.v);
51          }
52          if (vert2.size()!=edges.size()+1)
53          {
54              return 0;
55          }
56          if (conne(*it,e5))
57          {
58              edges.insert(e5);
59              vert2.insert(e5.u);
60              vert2.insert(e5.v);
61          }
62          if (vert2.size()!=edges.size()+1)
63          {
64              return 0;
65          }
66          }
67          }
68          if (edges.size()!=5)
69          {
70              return 0;
71          }
72          int a[N];
73          for(int i=0;i<=N;i++)
74          {
75              a[i]=0;
76          }
77          a[e1.u]++;
78          a[e1.v]++;
79          a[e2.u]++;
80          a[e2.v]++;
81          a[e3.u]++;
82          a[e3.v]++;
83          a[e4.u]++;
84          a[e4.v]++;
85          a[e5.u]++;
86          a[e5.v]++;
87          int countnull=0;
88          for(int i=0;i<=N;i++)
89          {
```

```
90                  if(a[i]==0)
91                  {
92                      countnull++;
93                  }
94              }
95              int countone=0;
96              for(int i=0;i<=N;i++)
97              {
98                  if(a[i]==1)
99                  {
100                     countone++;
101                 }
102             }
103             int counttwo=0;
104             for(int i=0;i<=N;i++)
105             {
106                 if(a[i]==2)
107                 {
108                     counttwo++;
109                 }
110             }
111             if(countone==2 && counttwo==4)
112             {
113                 return 1;
114             }
115             else{return 0;}
116     }
117     else{
118             return 0;
119     }
120 }
```

Hivatkozások

[1] J. Barát. Karmok és út-felbontások. *Polygon*, XVI.(52):17–26, 2007.

[2] J. Barát and C. Thomassen. Claw-decompositions and tutte-orientations. *J. Graph Theory*, (52):135–146, 2006.

[3] R. Häggkvist and R. Johansson. A note on edge-decompositions of planar graphs. *Discrete Math.*, (283):263–266, 2004.

[4] F. Jaeger. Flows and generalized coloring theorems in graphs. *J. Combin. Theory Ser. B*, (26):205–216, 1979.

[5] F. Jaeger. Nowhere-zero flow problems. in: Selected topics in graph theory 3 edited by l. w. beineke and r. j. wilson. *Academic Press*, pages 71–95, 1988.

[6] M. Jünger, G. Reinelt, and W.R. Pulleyblank. On partitioning the edges of graphs into connected subgraphs. *J. of Graph Theory*, (9):539–549, 1985.

[7] Gy. Katona, A. Recski, and Cs. Szabó. *A számítástudomány alapjai*. Typotex, 2006.

[8] A. Kötzig. From the theory of finite regular graphs of degree three and four. *Casopis Pest. Mat.*, (82):76–92, 1957.

[9] L. Lovász. *Combinatorial Problems and Exercises*. North-Holland, 1979.

[10] L. M. Lovász, C. Thomassen, Y. Wu, and c. Q. Zhang. Nowhere-zero 3-flows and modulo k-orientations. *J. Combin Theory Ser B.*, (103):587–598, 2013.

[11] W. Mader. A reduction method for edge-connectivity in graphs. *Ann. Discrete Math.*, (3):145–164, 1978.

[12] B. D. McKay and A. Piperno. Practical graph isomorphism, ii. *J. Symbolic Computation*, (60):94–112, 2013.

[13] C. St. J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *J. London Math. Soc.*, (36):445–450, 1961.

[14] C. St. J. A. Nash-Williams. Decomposition of finite graphs into forests. *J. London Math. Soc.*, (39):12, 1964.

[15] C. Thomassen. Decompositions of highly connected graphs into paths of length 3. *J. Graph Theory*, (58):286–292, 2008.

[16] C. Thomassen. The weak 3-flow conjecture and the weak circular flow conjecture. *J. Combin Theory Ser B.*, (102):521–529, 2012.

[17] W. T. Tutte. A contribution to the theory of chromatic polynomials. *Canad. J. Math.*, (6):80–91, 1954.

[18] W. T. Tutte. on the problem of decomposing a graph into n connected factors. *J. London Math. Soc.*, (36):221–230, 1961.