



SZAKDOLGOZAT FELADAT

Tokovics Dávid Tamás

Mérnök-informatikus hallgató részére

Hálózati folyam feladatok automatikus generálása

A Számítástudomány alapjai és a Bevezetés a számítástudományba tárgyak anyaga a Hálózati folyamatok elmélete. Minden félévben szükségünk van több konkrét példára, amit a hallgatóknak kell megoldani. Nem könnyű feladat rendszeresen előállítani ilyen példákat, hiszen nem lehet se túl könnyű, se túl nehéz, se túl kicsi, se túl nagy.

A hallgató feladata egy olyan alkalmazás létrehozása, ami ilyen példákat automatikusan előállít. A feladat első nehézsége annak definiálása, hogy mikor „jó” egy ilyen példa.

A hallgató feladatának egy olyan szoftver létrehozása, ami tudja a következőket:

- Adott típusú gráfokból egy véletlen hálózat generálása.
- Egy véletlenül kiválasztott majdani minimális vágás generálása.
- Az élekhez véletlen kapacitások rendelése úgy, hogy a minimális vágás éppen a kiválasztott legyen.
- A folyam feladat megoldása.
- A feladat és megoldás grafikus megjelenítése.
- A feladat nehézségének megfelelően egy mérőszámot is rendeljen a feladathoz egy adott skálán.
- A feladat exportálható legyen LaTeX TikZ formátumba.

Tanszéki konzulens: Dr. Katona Gyula egyetemi docens

Budapest, 2022. október 08.

Dr. Katona Gyula
egyetemi docens
tanszékvezető



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Számítástudományi és Információelméleti Tanszék

Tokovics Dávid

HÁLÓZATI FOLYAM FELADATOK AUTOMATIKUS GENERÁLÁSA

KONZULENS

Dr. Katona Gyula

2022. december 9.

Tartalomjegyzék

Összefoglaló	1
Abstract.....	2
Bevezetés	3
Hálózati folyam feladatok	5
2.1 Hálózatok	5
2.2 Folyamok	6
2.3 Maximális folyam	7
2.4 Minimális vágás	8
Hálózatok generálása.....	9
3.1 Csúcsok elrendezése	9
3.2 Keret éleinek behúzása	11
3.3 További élek hozzáadása	13
3.3.1 s-ből vagy t-be mutató élek.....	13
3.3.2 Azonos oszlopba mutató élek	14
3.3.3 Eggyel előre mutató élek	14
3.3.4 Kettővel előre mutató élek.....	15
3.3.5 Eggyel vissza mutató élek.....	15
3.4 Vágás sorsolása.....	19
3.5 Folyam létrehozása	19
3.5.1 Utak keresése s-ből t-be	20
3.5.2 Utak keresése visszaéllal	21
3.5.3 Keletkezett folyamok.....	23
3.6 Folyam átalakítása hálózattá	23
3.6.1 Élek növelése a vágás éleinek kivételével	24
3.6.2 Keletkezett hálózatok.....	24
Feladat nehézsége.....	26
4.1 Élekkel kapcsolatos nehézségek	26
4.1.1 Élek száma	26
4.1.2 Nullélek száma.....	27
4.1.3 Hosszú élek száma	29
4.1.4 Visszafele mutató élek száma	30

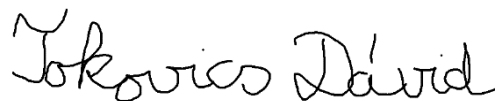
4.2 Csúcsok száma	31
4.3 Vágáserősség	31
4.4 Szórás	32
4.5 Teljes képlet	33
4.6 Példák könnyű és nehéz feladatokra	35
A program bemenetei	37
5.1 Grafikus felület kezdőoldala	37
5.2 Hálózat generálása paraméterek megadásával	38
5.3 Hálózat generálása fájlból	40
5.3.1 Hálózat generálása random gráffal	40
5.3.2 Hálózat generálása előre megadott gráffal és vágással	41
A program kimenetei	44
6.1 Grafikus megjelenítés	44
6.1.1 Feladat	44
6.1.2 Megoldás	46
6.1.3 Újragenerálás	49
6.2 Txt fájl	49
6.3 LaTeX fájl	50
6.3.1 Program által generált LaTeX fájl	50
6.3.2 Javított LaTeX fájl	52
Összegzés	54
Irodalomjegyzék	55

HALLGATÓI NYILATKOZAT

Alulírott **Tokovics Dávid**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022. 12. 09.



.....
Tokovics Dávid

Összefoglaló

A Bevezetés a számításelméletbe és a Számítástudomány alapjai tárgyakban a Hálózati folyamatok témakörében szükség van ellenőrizni a hallgatók tudását, ehhez pedig feladatokat kell készíteni. Ez azonban nem olyan egyszerű feladat, ezért nagy segítség egy olyan program, ami ezt a munkát elvégzi az oktatók helyett. Erről szól a dolgozat témája, amely egy program, ami hálózati folyamat feladatokat generál automatikusan, bizonyos paraméterek megadásával.

A paraméterek egyrészt a gráf tulajdonságai, mégpedig, hogy hány sorból és hány csúcsból áll. Emellett meg kell adni egy feladatnehezséget is, és ezen paraméterekből alkotja meg a megfelelő hálózatot a program, amelyeket bevihetünk a grafikus felületen keresztül, vagy fájlból. Utóbbi esetben akár egy konkrét gráfra is futtathatjuk.

A feladat nehézségekor figyelembe vesszük az élek számát, a hosszú élek számát, a visszafele mutató élek számát, azt, hogy a hálózatból keletkező maximális folyamban hány él kapacitása 0, illetve a minimális vágás erősségét, és az élek kapacitásának szórását, és a csúcsok számát is.

A program kimenete 3 féle lehet, egy futtatáskor mindhármát megkapjuk. Van egy JFrame alapú grafikus megjelenítés, ahol a megoldás megjelenítésére is van lehetőség, valamint újra is lehet generálni a feladatot, ha számunkra nem megfelelő az eredmény. Emellett egy txt fájlt is kapunk eredményként, ahol szöveges formában van leírva a feladat megjelenítése. A harmadik, amit kimenetként kapunk az egy LaTeX TikZ formátumú forráskód.

Abstract

It is necessary to check the knowledge of the students in the subjects of Introduction to the Theory of Computing and Foundation of Computer Science in the topic of Network Flows, and tasks must be prepared for this. However, this is not such an easy task, so a program that does this work instead of instructors is a great help. This is the topic of the thesis, which is a program that automatically generates network flow tasks by specifying certain parameters.

On the one hand, the parameters are properties of the graph, namely how many rows and vertices it consists of. In addition, a task difficulty must be entered, and the program creates the appropriate network from these parameters, which can be entered via the graphical interface or from a file. In the latter case, we can even run it on a specific graph.

When we count the difficulty of the task is, we take into account the number of edges, the number of long edges, the number of edges pointing back, how many edges have a capacity of 0 in the maximum flow from the network, as well as the strength of the minimum cut, and the standard deviation of the edge capacity, as well as the number of vertices.

The output of the program can be 3 types, we get all three in one run. There is a graphic display based on JFrame, where it is possible to display the solution, and it is also possible to generate the task again if the result is not good enough for us. In addition, we also receive a txt file as a result, where the display of the task is described in text form. The third output is a source code in LaTeX TikZ format.

1. fejezet

Bevezetés

Több szakon a BME-n és más egyetemeken is vannak tárgyak, amelyek a gráfok témakörére épülnek. Általában ennek a területnek része a hálózati folyamatok elmélete is, és az ehhez kapcsolódó maximális folyam és minimális vágás feladatok. A dolgozat célja egy olyan program létrehozása volt, amely képes hálózati folyam feladatok automatikus generálására. A fő motiváció erre az volt, hogy ezt a programot aztán lehessen hasznosítani a Számítástudomány alapjai és a Bevezetés a számításelméletbe tárgyakban, ugyanis ezen tárgyak témakörei közé tartoznak a hálózati folyamatok. Szükség van ebben a témakörben a hallgatók tudásának ellenőrzésére, és ennek érdekében feladatok készítésére.

A feladatok létrehozása azonban nem olyan egyszerű dolog, mivel nem egyértelműen megállapítható egy létrehozott feladatnak a nehézsége. Nyilván nem szeretnénk például egy zárthelyi dolgozatban, ha a hallgató nagyon hamar megoldaná a feladatot, és azt sem, ha egy megoldhatatlannak tűnő feladattal találkozna zárthelyi írás közben. Viszont az sem ideális, ha a megfelelő feladat kidolgozására az oktátónak a kelleténél több ideje menne rá, hogy az szerinte pont jó nehézségű legyen, egyébként is a feladat létrehozásakor nehéz ellenőrizni, hogy valóban olyan nehéz-e, mint ahogy mi azt gondoljuk. Tehát az igazi kihívás az, hogy egy számunkra megfelelő nehézségű feladatot kreáljunk. Ezért jól jönne egy program, ami ezt megteszi helyettünk, vagyis egy általunk megadott nehézséggel hoz létre egy hálózati folyam feladatot, amelyet aztán a gyakorlatokon és a zárthelyi dolgozatokban fel lehet használni, ezzel pedig az oktatóknak rengeteg időt spórolva.

A dolgozat további részében szükség lesz először is tisztázni, hogy mik is azok a hálózatok, folyamatok, hogyan lehet belőlük feladatot csinálni. Aztán szükség van annak az elemzésére, hogy mitől is lesz nehéz egy ilyen feladat, és hogyan tudunk a programunkkal egy kívánt nehézségű feladatot generálni. Ezután pedig arról is szó kell essen, hogy milyen módon tudjuk azt a programnak megmondani, hogy hogyan is nézzen ki a feladatunk, ugyanis nem csak egy féleképpen tudunk egy hálózatot generálni, lehet

például több sora is az egyiknek, mint egy másiknak. Illetve ezen kívül a kívánt nehézség értékét is meg kell adnunk valamilyen módon a programunknak, amiből az aztán feladatot tud generálni. Végül pedig azt is tisztáznunk kell, hogy az eredményt mi milyen módon kapjuk meg, illetve azt, hogy tudjuk a későbbiekben, például egy zárthelyi dolgozatban felhasználni.

2. fejezet

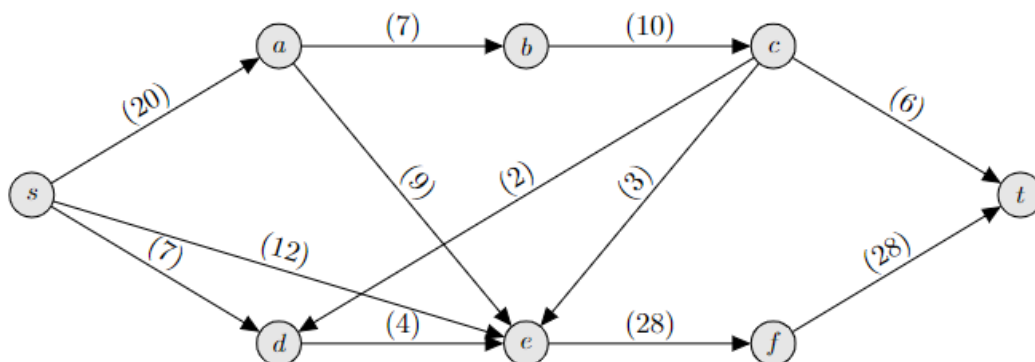
Hálózati folyam feladatok

Hálózati folyam feladatok alatt gyakorlatilag T. E. Harris és F. S. Ross 1954-ben megfogalmazott maximális áramlási problémával [1] kapcsolatos feladatokat értjük. Ők ezt először a szovjet vasúti forgalom egyszerűsített modelljeként fogalmazták meg, és rendkívül jól hasznosítható a probléma a közlekedési hálózatokban, de más területeken is célszerű lehet a használata, például Jon Kleinberg és Éva Tardos Algorithm Design [2] című könyvében kép szegmentálására használt algoritmusban alkalmazzák a maximális áramlás elméletét.

A probléma lényege, hogy van egy hálózatunk, aminek kezdőpontja s , végpontja pedig t , és ezen kezdő- és végpontok között kell megtalálnunk a maximális folyamot, és esetlegesen a minimális vágást is (ezek értéke egyenlő). Hogy mik is a hálózatok, folyamok és a vágás, azt alább részletezzük. Ez a gyakorlatban például úgy képzelhető el, hogy van egy termék szállítására alkalmas hálózatunk, és azt kell kiderítenünk, hogy mekkora a legnagyobb termékmennyiség a mi a kezdőpontból a végpontba elszállítható.

2.1 Hálózatok

Hálózatoknak azokat az irányított gráfokat nevezzük, amiben s és t a gráf egy-egy csúcsai, és a gráf minden éléhez tartozik egy nemnegatív kapacitásérték. Ilyen hálózatot tartalmaz a 2.1. ábra.

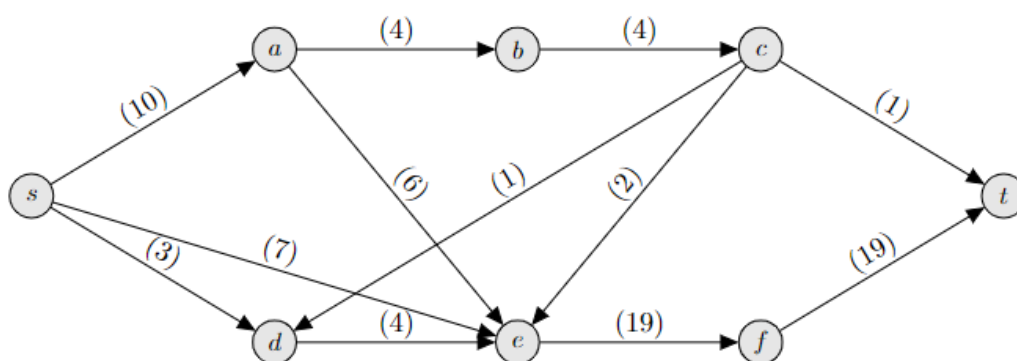


2.1. ábra: Hálózat

Ez a hálózat reprezentálhat például egy vonathálózatot, ahol az élek kapacitása megmutatja, hogy két megálló között az adott irányba óránként hány vonat képes menni. Alapvetően viszont nekünk lényegtelen lesz, hogy két random megálló között, mennyi vonat haladhat át, mi arra vagyunk kíváncsiak, hogy s -ből t -be mennyi tud eljutni egy óra alatt. Éppen erre fog szolgálni a maximális folyam kiszámolása, amivel megkaphatjuk ezt az értéket. Viszont mielőtt kitérnénk arra, hogy mi is az a maximális folyam, és hogy tudjuk azt kiszámolni, előtte azt nézzük meg, hogy maga a folyam mit jelent.

2.2 Folyamok

Folyamoknak nevezzük a hálózatokhoz tartozó azon irányított gráfokat, amelyekben a kapacitás mellé minden e élhez egy másik $f(e)$ értéket is rendelünk, ahol az $f(e)$ érték minden esetben kisebb vagy egyenlő, mint az él kapacitása, viszont nagyobb vagy egyenlő, mint 0, emellett pedig minden csúcsba tartó élen található érték összege egyenlő a minden csúcsból induló élen található él összegével, kivéve az s és t éleket. Vagyis amilyen értékek bemennek egy csúcsba, azok ki is jönnek. Az s élből induló élek értékeinek összege pedig a folyam értéke lesz, levonva belőle az s -be bemenő élek értékeit, ez pedig egyenlő lesz a t -be tartó élek értékeinek összegével, levonva belőle a t -ből induló élek értékeit. Mi most a dolgozatban csak olyan folyamokkal hálózatokkal foglalkozunk, ahol s -be és t -ből nem mennek élek, így ezeket a lehetőségeket a továbbiakban elhagyjuk. Egy ilyen a 2.1. ábra hálózatához tartozó folyamot mutat be a következő ábra.



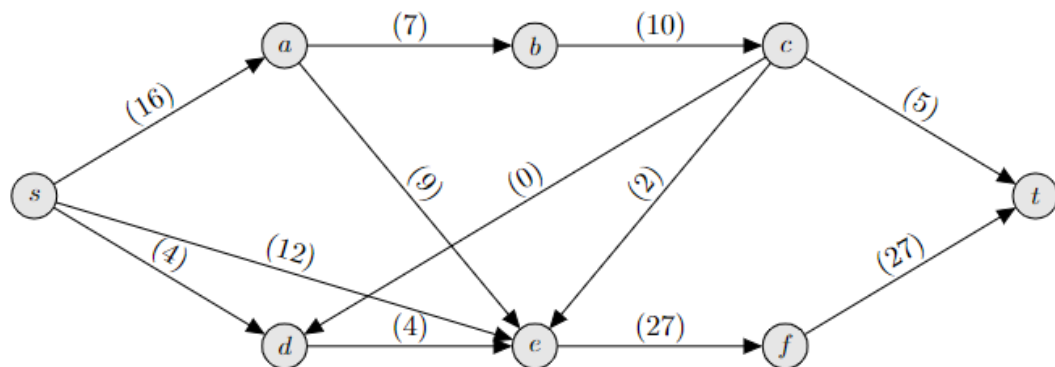
2.2. ábra: Folyam

Itt az éleken található értékek most nem a kapacitást, hanem az $f(e)$ értéket jelentik, jól látható, hogy minden csúcsba tartó élek értékeinek összege megegyezik a csúcsból induló élek értékeinek összegével, például az e csúcsba tartó élek értékeinek összege $4+7+6+2=19$, ami egyenlő az e élből induló egyetlen él értékével. A vonatos

példánál maradva ez tehát megmutat egy olyan lehetőséget, hogy s-ből t-be óránként mennyi vonat juthat át, és ebből két megálló között mennyi megy át. Ahhoz viszont, hogy azt megvizsgáljuk, hogy mennyit bír el a rendszer, ahhoz ebből a maximálisra vagyunk kíváncsiak.

2.3 Maximális folyam

A maximális folyam tehát egy hálózathoz tartozó azon folyam lesz, ahol az s-ből induló élek értékének összege maximális, tehát nem rendelhető olyan folyam a hálózathoz, ahol ez az összeg nagyobb lenne. Egy hálózathoz több maximális folyam is tartozhat, azonban ezekben az s-ből induló élek értékeinek összege azonos lesz. Fontos megjegyezni, hogy ez az összeg minden esetben egyenlő a t-ben végződő élek értékeinek összegével. Egy ilyen a 2.1. ábra által tartalmazott hálózathoz tartozó maximális folyam látható alább.



2.3. ábra: Maximális folyam

Így tehát ismételten a vonatos témára visszatérve s-ből t-be maximum $16+12+4=32$ vonat mehet, ami egyenlő a $27+5$ -tel természetesen.

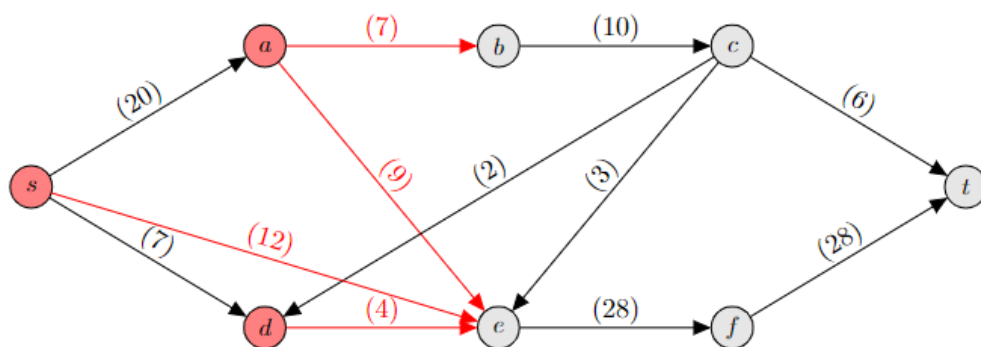
A maximális folyam megtalálására vannak különböző módszerek, például Dinic's algoritmus [3], a bináris blokkoló áramlási algoritmus [4], a push-relabel algoritmus [5] különböző variációi, King, Rao és Tarjan algoritmus [6] vagy az MPM algoritmus [7]. A Számítástudomány alapjai és a Bevezetés a számításelméletbe tárgyakban a Ford és Fulkerson által megalkotott javítóutas algoritmusról [8] esik szó. Alapvetően a megfigyelés az, hogy a hallgatók a feladat megoldásakor elsősorban azonban nem feltétlenül használják az algoritmust, legalábbis egy minimális kezdőpont kell hozzá, hogy a használata ne legyen túl időigényes, így a feladat létrehozásakor nem feltétlenül azt kell figyelembe venni, hogy ezzel az algoritmussal milyen nehezen tudja a hallgató megoldani a maximális folyam feladatot, hanem inkább azt, hogy ránézésre, és

próbálkozással mennyire nehezen tudja megoldani a feladatot, vagy eljutni egy olyan pontig, ahol könnyen használhatóvá válik az algoritmus.

2.4 Minimális vágás

Azt, hogy a maximális folyamban mennyi az s -ből induló élek értékeinek összege nem csak egy maximális folyam megtalálásával tudhatjuk meg, létezik erre más módszer is. Ez pedig nem más, mint a minimális vágás keresése. De először tisztázzuk, hogy mi is az a vágás. A gráf egy csúcshalmazát nevezzük vágásnak, amelyben benne van s , de nincs benne t . A vágás értéke pedig nem más, mint a vágás csúcaiból induló olyan élek kapacitásainak az összege, amelyeknek végpontja nem a vágás csúcsai között található. A minimális vágás pedig nem más, mint az a vágás, aminek az értéke az adott hálózaton minimális. Ebből több is lehet, viszont a programunk által generált feladatokban olyan hálózatok fognak szerepelni, ahol a minimális vágás csak egy féle lehet. Mivel a feladat ennek megtalálásával megoldható, illetve egy maximális folyam helyessége ellenőrizhető, így a minimális vágás bonyolultságával is vizsgálhatjuk a feladat nehézségét.

Ez a minimális vágás pedig úgy kapcsolódik a maximális folyamhoz, hogy a Ford-Fulkerson tétel [9] szerint a maximális folyam értéke egyenlő lesz a minimális vágás értékével. Tehát ha az érték a kérdés, akkor az könnyen megtalálható egy minimális vágás megtalálásával is, ha pedig egy maximális folyam a kérdés, akkor pedig annak helyessége, vagyis, hogy valóban maximális-e ellenőrizhető azzal, ha találunk egy minimális vágást. A 2.1. ábra hálózatához például az alábbi ábrán láthatunk egy minimális vágást, ahol a vágásból induló élek összege $7+9+12+4=32$, ami valóban egyenlő a maximális folyam értékével.



2.4. ábra: Minimális vágás

3. fejezet

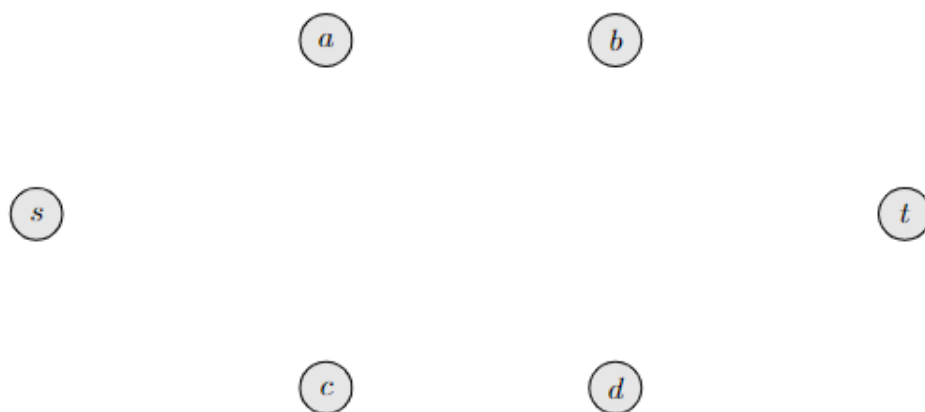
Hálózatok generálása

A feladatokat generáló program csak a megadott paraméterek alapján gyárt hálózatokat, amelyeket aztán kiértékel nehézség szerint, és ez alapján eldönti, hogy elveti vagy megtartja azt. Később a megtartott hálózatot is ugyanúgy elvetheti, ha talál egy jobbat, mivel rengeteget generál, és mindegyik nehézségét összehasonlítja a kívánt értékkel, így, ha egy újonnan létrehozott hálózathoz tartozó nehézség közelebb áll a kívánt feladat nehézségéhez, akkor azt fogjuk megtartani, és az eddig megtartott hálózatot elvetni. Ebben a fejezetben megnézzük, hogy hogyan is állítjuk elő ezeket a hálózatokat. Először legyártjuk a gráfot részekre lebontva, majd utána adunk kapacitásokat az éleknek, így hálózatot létrehozva belőle, amely, ha megfelelő nehézségűre sikerült, akkor később maga a feladat, mint a program eredménye is lehet belőle.

3.1 Csúcsok elrendezése

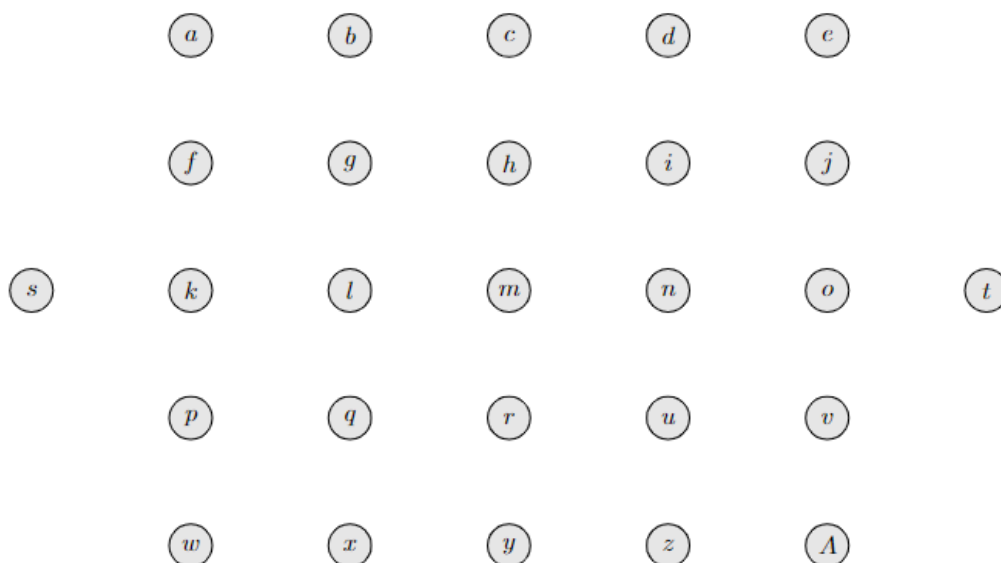
A jelen program az egyszerűség kedvéért, csak bizonyos rácsszerű gráfokat tud generálni. Első lépés az, hogy a megadott sor- és csúcsszámok alapján megalkotjuk a csúcsok elrendezését. A megadott paraméterek alapján mindig csak egyféle elrendezés lehetséges, ami pedig a következő: Ha páros számú sorral kell generálnunk, akkor valójában annak a gráfnak is páratlan számú sora lesz, viszont a középső sorban csak az s és t szerepel a bal és jobb oldalakon a sor szélén. A többi sorban pedig a megadott számú mínusz 2 – ami az s és a t – csúcs található elosztva úgy, hogy minden, a megadott számú sorban azonos mennyiségű legyen belőlük. Ha páratlan a megadott sorok száma, akkor a középső sorban a bal és a jobb szélén található s és t , valamint ebben a sorban 2-vel több a csúcsok száma, mint a többiben, egyébként pedig ebben az esetben is ugyanúgy a megadott számú sorban elosztjuk a megadott számú csúcsot úgy, hogy a sorokban azonos mennyiség legyen.

A legkisebb gráf, amit így alkothatunk az 2 sorból és 6 csúcsból áll, (ezt 2×2 -es gráfként írjuk le) ezt tartalmazza a 3.1. ábra.



3.1. ábra: 2×2-es csúcselrendezés

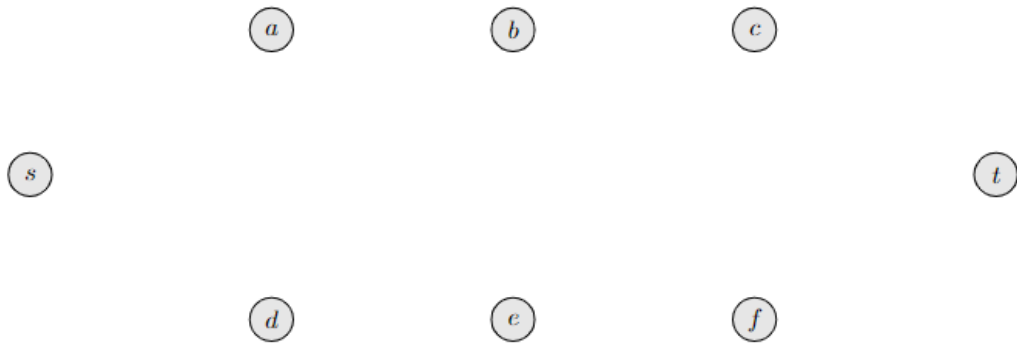
A legnagyobb gráf pedig, amit így létrehozhatunk az 5 soros, és 27 csúcsa van (5×5-ös), ezt a 3.2. ábra tartalmazza.



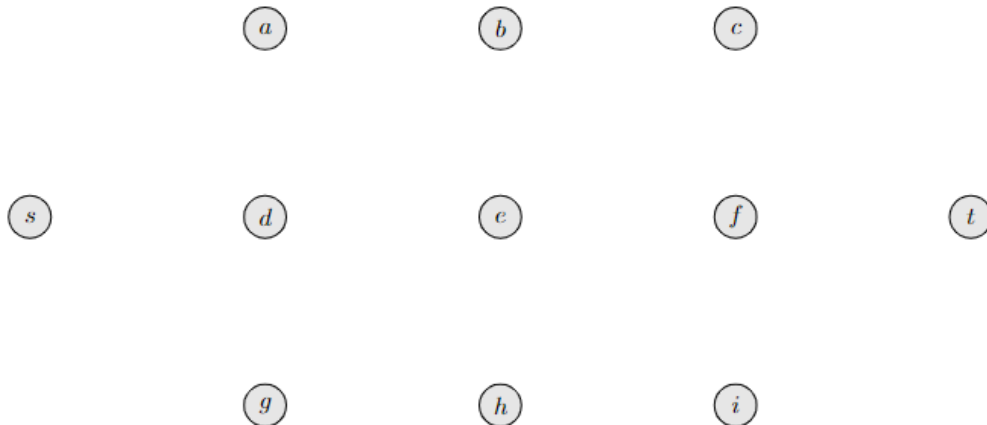
3.2. ábra: 5×5-ös csúcselrendezés

A két érték között sor- és oszlopszámban is bármilyen értékkel hozhatunk létre gráfot, vagyis a 2 és 5 mellett lehet 3 vagy 4 soros is, ugyanígy a 2 és 5 mellett állhat 3 vagy 4 oszlopból is, ahol az oszlopszámításban nem számoljuk bele az s és t oszlopait, csak a többi.

A fenti két csúcselrendezést ritkán fogjuk használni, a két leggyakoribb gráf típus a 2 csúcsból és 8 csúcsból álló 2×3 -as, valamint a 3 sorból és 11 csúcsból álló 3×3 -as lesz, amelyeket a 3.3. ábra és 3.4. ábra tartalmazza.



3.3. ábra: 2×3 -as csúcselrendezés



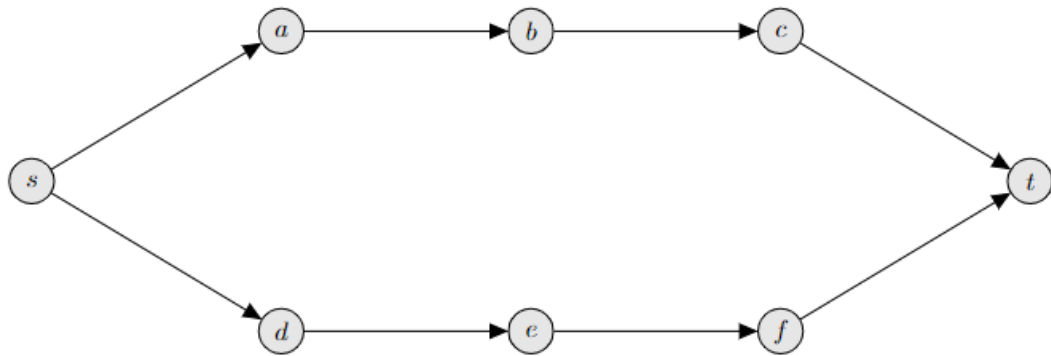
3.4. ábra: 3×3 -as csúcselrendezés

3.2 Keret éleinek behúzása

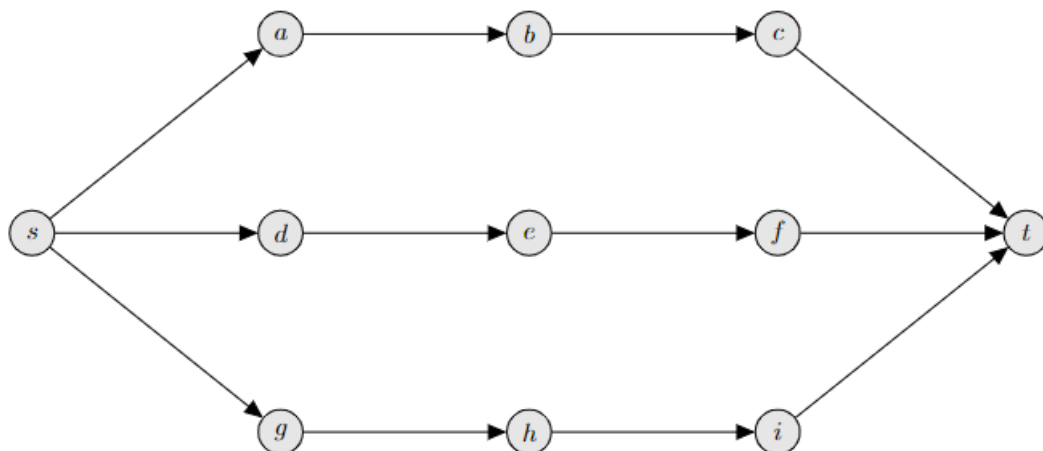
Ha megalkottuk a keretet, akkor a következő feladat, hogy éleket húzzunk be a gráfba. Ezt nem teljesen randomizált módon tesszük, ugyanis lesznek élek, amelyeket biztosan bele kell tegyünk minden egyes gráfba, és lesznek olyanok is, amiket viszont semmiképp nem húzzunk be. A maradékot pedig az él típusától függően bizonyos valószínűségekkel húzzuk majd be.

Először nézzük meg azokat az éleket, amelyek biztosan részei lesznek a gráfnak. Az egyik ilyen él az s -ből minden sor első csúcsába mutató él. Szintén mindenképp behúzzuk minden csúcs utolsó sorából a t -be mutató éleket. Ezen kívül a harmadik éltípus, amiből az összeset biztosan hozzáadjuk a gráfhoz, azok a sorban a csúcsból a tőle jobbra található csúcsba húzott élek, tehát, hogy minden sorban balról jobbra a sor elejétől a végéig vezessen egy út. Így egy olyan gráfot kapunk, ahol s -ből t -be minden soron keresztül vezet út.

Ezek az élek tehát minden hálózati folyam feladatban szerepelni fognak. Így lesz a 2×3 -as és 3×3 -as csúcstelrendezésből az alábbi két gráf, amit a 3.5. ábra és 3.6. ábra tartalmaz, ezekhez fogunk még behúzni további éleket.



3.5. ábra: Keret a 2×3 -as csúcstelrendezésben



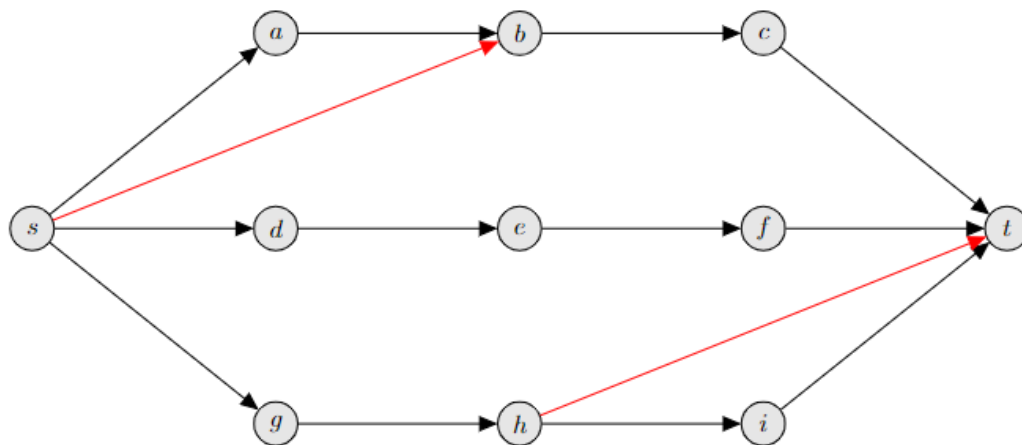
3.6. ábra: Keret a 3×3 -as csúcstelrendezésben

3.3 További élek hozzáadása

Ha létrehoztuk a gráfot a megfelelő csúcselrendezéssel, és behúztuk a keret éleit, akkor jöhet a többi él behúzása is, amely most már bizonyos mértékben randomizált módon fog működni, az alapján, hogy milyen fajta élt szeretnénk behúzni. Fontos, hogy két csúcs közé csak egy élt húzunk be, vagyis, ha már hozzá van adva a gráfhoz egy él, akkor ugyanazon két csúcs közé nem húzható be egy a másik irányba mutató él, vagyis a két csúcs közé behúzható két élnak a valószínűsége, hogy része lesz a gráfnak, az egymástól nem független. Fontos még azt is megjegyezni, hogy azokban a gráfokban, ahol páros számú sor van, (de valójában ugyanúgy páratlan ott is) azoknál az s és t sora feletti, illetve alatti sorok szomszédosnak számítanak a továbbiakban.

3.3.1 s -ből vagy t -be mutató élek

Ugyan az s -ből, illetve a t -be húztunk már be éleket, viszont van még pár, amit még hozzáadhatunk a gráfhoz. Az ilyenek a szomszédos, vagyis a tőlük eggyel feljebb vagy lejjebb található sorokban az s -ből a sor második csúcsába húzott, vagy a t -be a sor utolsó előtti csúcsából húzott élek.

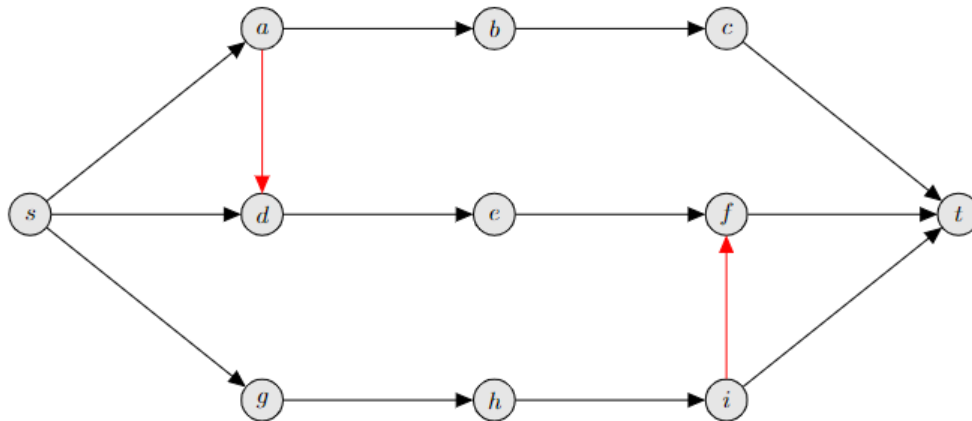


3.7. ábra: s -ből vagy t -be mutató élek

Mindegyik ilyen fajta élre $1/3$ az esély, hogy bekerül a gráfba. Mivel minden gráfban pontosan 4 ilyen fajta behúzható él van, ezért leggyakrabban 1 ilyen él lesz behúzva.

3.3.2 Azonos oszlopba mutató élek

A következő kategória azok az élek, amelyek valamelyik sorban egy csúcsból az egygel alatta vagy felette lévő sorban, vele egy oszlopban található csúcsba húzott élek.

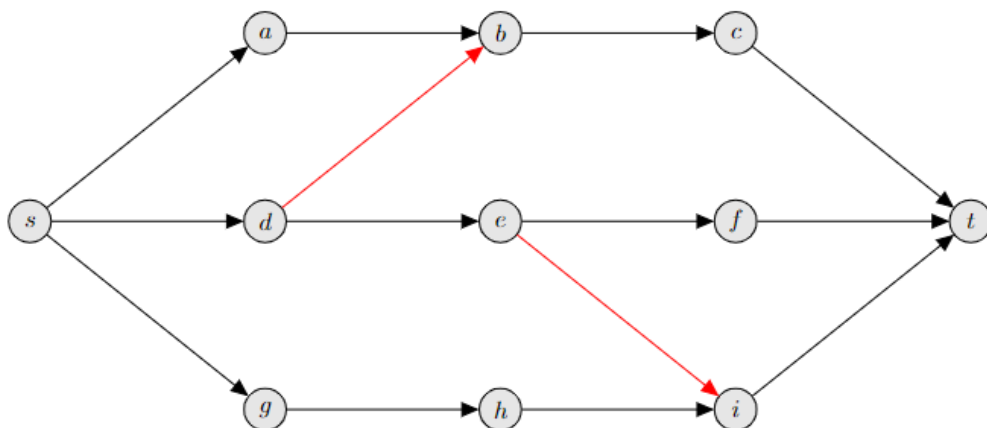


3.8. ábra: Azonos oszlopba mutató élek

Minden ilyen élt $1/4$ eséllyel húzunk be, viszont 2 csúcs között $1/2$ eséllyel lesz behúzva ilyen fajta él, csak azon belül $1/2$ eséllyel az egyik, $1/2$ eséllyel pedig a másik irányba.

3.3.3 Egygel előre mutató élek

A következő fajta élek a valamelyik sorban egy csúcsból az egygel alatta vagy felette lévő sorban, vele egygel jobbra lévő oszlopban található csúcsba húzott élek.

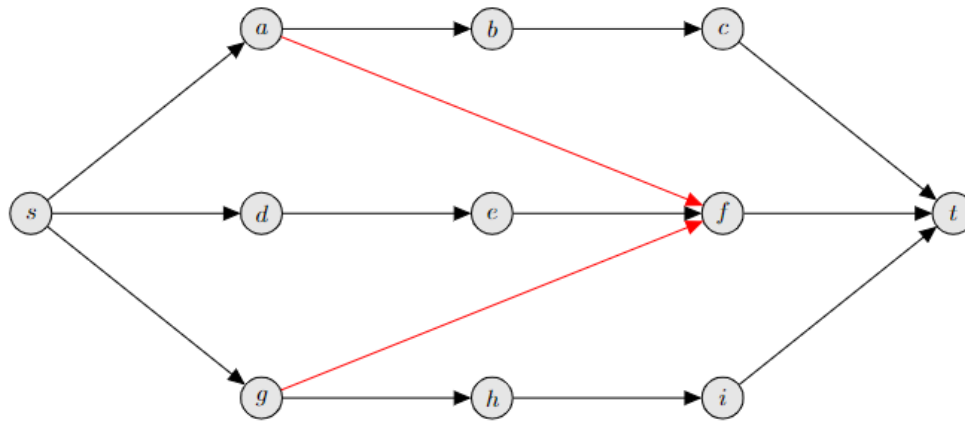


3.9. ábra: Egygel előre mutató élek

Az ilyen fajta éleket $1/3$ eséllyel adjuk hozzá a gráfhoz, ebből egy kicsivel kevesebb lehetőség van, mint az előző kategóriából, de nagyobb esély van rá, hogy behúzzuk, így nagyjából ugyanannyi lesz a gráfban a kétfajtából.

3.3.4 Kettővel előre mutató élek

A következő kategória azok az élek, amelyek egy csúcsból az eggyel alatta vagy felette lévő sorban, vele kettővel jobbra lévő oszlopban található csúcsba húzott élek.

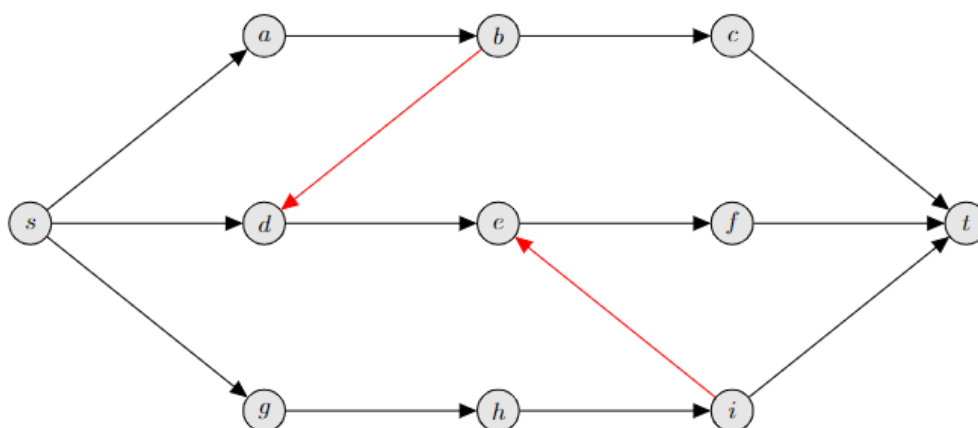


3.10. ábra: Kettővel előre mutató élek

Ebből a fajtából már nem szeretnénk olyan sokat belerakni, amúgy is kevesebb lehetőség van ilyen élek behúzására, és ezt 1/4 eséllyel tesszük meg.

3.3.5 Eggyel vissza mutató élek

A következő fajta élek a valamelyik sorban egy csúcsból az eggyel alatta vagy felette lévő sorban, vele eggyel balra lévő oszlopban található csúcsba húzott élek.



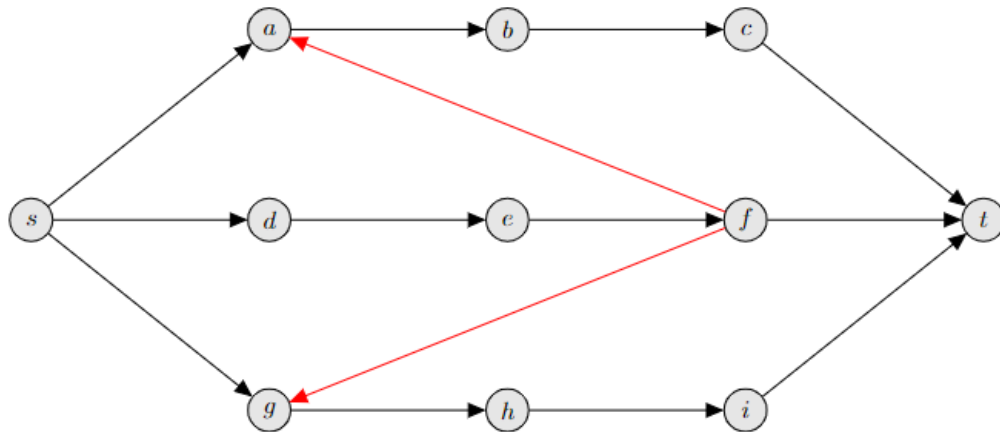
3.11. ábra: Eggyel vissza mutató élek

Visszafele mutató élekből kevesebb fog kelleni nekünk, mivel alapvetően s-ből t felé nem akarjuk a megfelelő utakat úgy bonyolítani, hogy túl sok él visszafele mutasson,

így egy ilyen él behúzására az esély $1/6$. Ezzel nagyjából fele annyi kellene legyen a számuk, mint az eggyel előre mutató éleknek.

3.3.6 Kettővel vissza mutató élek

A következő kategória azok az élek, amelyek egy csúcsból az eggyel alatta vagy felette lévő sorban, vele kettővel balra lévő oszlopban található csúcsba húzott élek.



3.12. ábra: Kettővel vissza mutató élek

Ebből lesz a legkevesebbre szükségünk, mivel se a hosszú élek, se a visszafele mutató élek nem olyan fontosak, hogy annyira sok legyen belőlük. $1/8$ eséllyel húzunk be egy ilyet, vagyis ebből is nagyjából fele annyi kellene legyen, mint az előre felé mutató párjából.

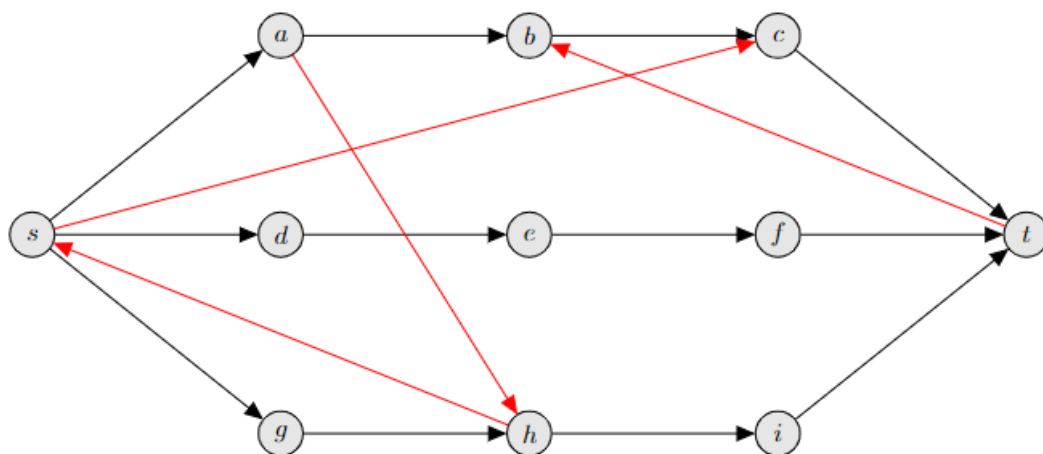
3.3.7 Egyéb, nem használt éltípusok

A fentiekben az összes éltípust felsoroltuk, amiket használunk a gráfunkban, és behúzunk a folyam elkészítésekor, viszont kimaradt pár élfajta, amikről nem esett szó, azokat pedig nem fogjuk használni a hálózatok generálásakor. Ilyen él például, ha s-ből egy szomszédos sor harmadik, vagy attól jobbra lévő csúcsába húzunk egy élt. Ugyanígy, ha egy szomszédos sorban egy csúcs nem az utolsó két csúcs egyike, akkor semmiképp nem húzunk be onnan a t-be egy élt. Emellett nem húzunk be az s-be vagy a t-ből visszamutató éleket, vagyis nem lehet a gráfunknak olyan éle, amelynek végpontja s vagy kezdőpontja t.

De nem csak s-sel vagy t-vel kapcsolatos élek vannak tiltólistán, ugyanis például olyan élt sem húzunk be, amely bármelyik oszlopból húzott csúcsból egy tőle legalább 3-

mal balra vagy jobbra lévő csúcsba mutat. Emellett semmilyen élt nem húzunk be olyan csúcsok között sem, amelyek nem szomszédos sorokban találhatóak. Illetve azt már említettük, hogy két csúcs közé csak egy él húzható, így például egy sorban lévő két szomszédos csúcs közé sem húzhatunk be újabb élt, de igazából nem húzhatunk be egy sorban lévő két nem szomszédos él közé sem, mivel ezek is fednék egymást más élekkel.

Természetesen ezek a szabályok nem a hálózat definíciójából erednek, nem emiatt nem húzzuk be a felsorolt éleket, hanem kizárólag esztétikai szempontból az átláthatóság érdekében. Ahogy az sem, hogy kizárólag egyenes élekkel dolgozunk. Így is fogunk kapni meglehetősen bonyolult hálózatokat, amelyeket elég nehéz átlátni, de ezekkel a szabályokkal azért ez valamennyire kordában tartható. A lenti ábra tartalmaz pár élt ezekből a „tiltott” élekből.



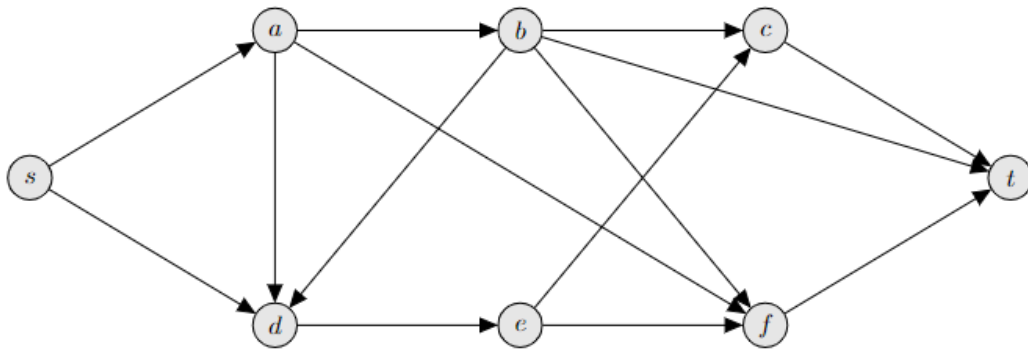
3.13. ábra: Nem használt éltípusok

3.3.8 Az élek hozzáadásával generált gráfok

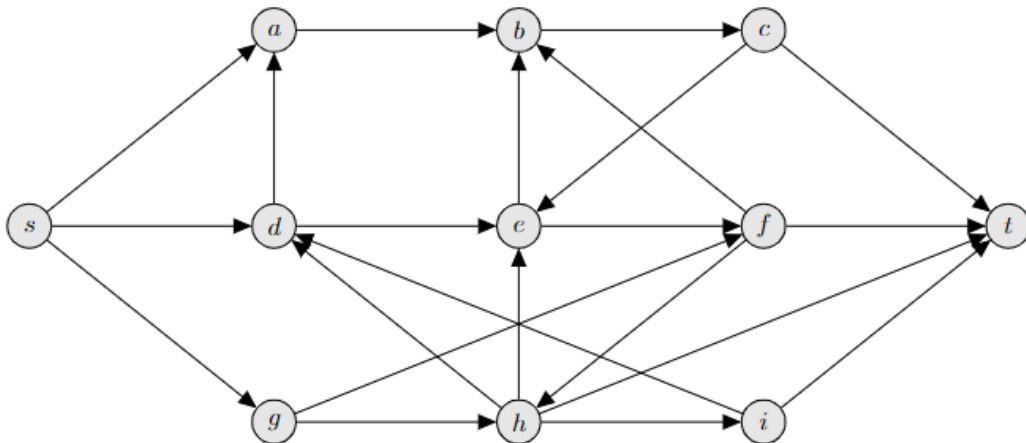
Így tehát a keret éleivel, valamint a többi hozzáadott éllel kaptunk egy irányított gráfot, amely az s-ből t-be mutató utakat tartalmaz. A feljebb leírtak szerint randomizálva történt az élek hozzáadása, így az is lehet, hogy nagyon sok éle lesz a gráfnak, de előfordulhat, hogy csak pár él lesz behúzva a kereten kívül. Természetesen, ha több éle van a gráfnak, akkor az abból létrehozott hálózatból nehezebb feladat készíthető, úgyhogy a nehézség kiszámolásánál ez is fontos tényező lesz, ahogy az is, hogy melyik fajta élből mennyit húztunk be, ugyanis például minél több visszafele mutató él van a hálózatban, annál bonyolultabb lesz a feladat. Így tehát ha a randomizálás által legyártott gráfból nem

lehet olyan nehéz feladatot létrehozni, mint amit megadtunk, akkor sincs probléma, mivel a program generál új gráfot, amiből nehezebb vagy könnyebb feladat készíthető. Ez alól kivételt képez az, amikor mi adjuk meg a gráfot, mivel erre is van lehetőség. Ekkor nem túl nagy a nehézség tartomány, amiben tud generálni feladatot a program, de akkor azzal dolgozik.

Az alábbiakban látható egy-egy a 2×3 -as és a 3×3 -as csúcstelrendezéshez generált gráf, amiből aztán később a feladatot gyárthatjuk.



3.14. ábra: Egy generált 2×3 -as gráf



3.15. ábra: Egy generált 3×3 -as gráf

Jól látható, hogy például az utóbbi ábránál nagy mértékben előjött a random faktor, ugyanis az egyik leggyakoribb éltípusból, az eggyel előre felé mutató élből egy sem került bele, az eggyel visszafelé mutatóból viszont 4 is.

3.4 Vágás sorsolása

A gráf megalkotása után alapvetően a folyam létrehozása kellene következzen, de előbb szükséges kisorsolnunk egy vágást, ami később a minimális vágás lesz, mivel ettől függően kell majd megalkotnunk a folyamat, hogy ne rontsa el azt, hogy melyik a mi minimális vágásunk, mivel ebben az esetben bonyolítaná a feladatot, hogy meg kell találnunk az új minimális vágást, ami ráadásul nem is biztos, hogy csak egy darab van. Ezért törekednünk kell arra, hogy megtartsuk az általunk sorsolt minimális vágást, és ne a folyam alapján kelljen megtalálnunk.

A vágást meglehetősen egyszerűen generáljuk. Vesszük a csúcsok halmazát, és abból s -t beletesszük a vágásba, ezen kívül pedig a t -n kívül minden más csúcsot $2/5$ valószínűséggel beveszünk a vágás csúcsai közé. Szeretnénk, ha egy vágás legalább 3 csúcsból állna, így ha az s -en kívül maximum egy csúcs került be véletlenszerűen a vágás csúcsai közé, akkor újra sorsoljuk a vágást, majd ezt addig folytatjuk, ameddig legalább 3 csúcsból álló vágást nem kapunk. Ha ez teljesül, akkor kész van a vágásunk, ami később az egyetlen minimális vágása lesz a feladatnak.

3.5 Folyam létrehozása

Ha megalkottuk az irányított gráfunkat, valamint sorsoltunk egy vágást, akkor jöhet a következő feladat, még hozzá egy folyam, mégpedig egy maximális folyam létrehozása, amiből aztán a hálózatot generáljuk, amihez ez a maximális folyam tartozik. Erre azért van szükség, mert alapvetően azt szeretnénk, ha az élek kapacitása nem teljesen random lenne, hanem nagyjából a létrehozott hálózat éleinek kapacitása, és az ahhoz tartozó maximális folyam éleinek értéke nem térne el annyira egymástól. Emellett azt is szeretnénk elérni, hogy az s -ből induló élek kapacitásának összege nagyjából egyenlő legyen a t -be tartó élek kapacitásának összegével. Ezekről szebb lesz a feladatunk, viszont praktikus is abból a szempontból, hogy így sokkal könnyebb követni a hálózat egy maximális folyamát, és a minimális vágást is, így mivel azok szerepet játszanak a nehézség kiszámításában, így kevesebb időt veszítünk vele a feladat generálásakor.

Kezdetben minden élnek adjuk 0-t értéknek, ekkor ez értelemszerűen egy folyam, mivel teljesül rá, hogy minden csúcsban a benne végződő élen található értékek összege egyenlő az abból induló élek értékeinek összegével, mivel ez mind 0. Ezután növeljük az élek értékeit, ezt kétféleképpen tehetjük meg.

3.5.1 Utak keresése s-ből t-be

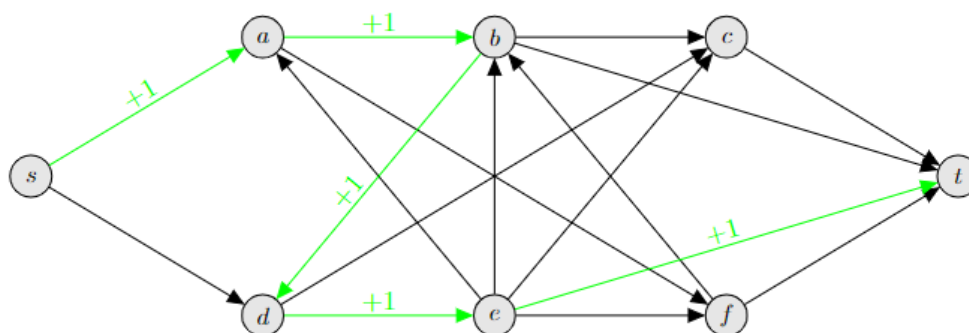
Az első lehetőség az élek értékeinek növelésére, hogy s-ből t-be tartó utakat keresünk, majd az út éleinek értékét egy előre kisorsolt, 1 és 4 közötti számmal növeljük (minden élt ugyanannyival). Ekkor ezek nem rontják el, hogy folyam maradjon a gráf, mivel minden az útban s és t kivételével, ha egy csúcsba bemegy egy él, akkor ugyanabban az útban a csúcsból kifelé is fog tartani egy él, és mindkettőnek az értékét ugyanannyival növeltük, így a csúcsba és a csúcsból húzott élek értékeinek összege ugyanannyi marad.

Két dologra azonban figyelniünk kell az élek növelésekor. Egyrészt az egy fontos kitétel, hogy s-ből t-be utakat keresünk, vagyis egy csúcs nem szerepelhet többször. Ez azért fontos, hogy ne kerüljünk végtelen ciklusba. Ezt ugyan azzal is elérhetnénk, ha utak helyett sétát keresnénk, de a program működése szempontjából optimálisabb az utak keresése. Ez nem csak ennél, hanem majd a következő növelésfajtánál is igaz lesz, hogy minden csúcs csak egyszer szerepelhet.

Másik fontos dolog amire figyelniünk kell az pedig az, hogy ha kiléptünk a kisorsolt vágás csúcsaiból egy éllel, akkor nem léphetünk vissza, vagyis utána már csak olyan éleket használhatunk fel az út további részében, aminek végpontja nem a vágás egyik csúcsa. Erre azért van szükség, mert ha kilépünk majd visszalépünk a vágás csúcsai közé, akkor utána valamikor megint ki fogunk lépni mindenképp, mivel t nem lehet benne a vágásban és oda kell eljutni. Ez pedig azért probléma, mivel akkor ezt a vágást kétszer is növeltük, mivel két vágásból kilépő élet is növeltük, viszont előfordulhat egy olyan vágás, aminek éleit csak egyszer növeltük, ebből következően pedig az általunk sorsolt vágás már nem lehet minimális, mivel van egy nála kisebb vágás. Így, ha figyelünk arra, hogy csak egyszer léphetünk ki a vágás élei közül, akkor biztosan megmarad minimális vágás, mivel minden vágást legalább egyszer növeltünk a kisorsolt értékkel, mert nem létezhet olyan út s és t között, valamint olyan vágás, amelyek között ne lenne legalább egy azonos él. Ez azért van, mert minden útban átlépünk egy olyan csúcsból, ami egy bármilyen vágás része egy olyanba, ami pedig nem, vagyis ezzel növelni fogjuk a vágás értékét.

A következő ábrán egy ilyen s-ből t-be vezető út látható, és itt most 1-gyel növeltük az élek értékét, de természetesen lehet 2-vel, 3-mal vagy 4-gyel is, csak az a lényeg, hogy minden élt ugyanannyival. Ugyan most itt nem lett megjelölve, hogy mely

csúcsok a vágás csúcsai, így például, ha az s , b , e vágás tartozna a gráfhoz, akkor ez nem lenne egy megfelelő növelés, mivel a b - d éllel kilépünk, a d - e éllel pedig visszalépünk a vágás csúcsai közé. Viszont például, ha a vágás az a , b , d csúcsokból áll, akkor megfelelő ez a növelés.



3.16. ábra: út élkapacitásainak növelése

3.5.2 Utak keresése visszaéllel

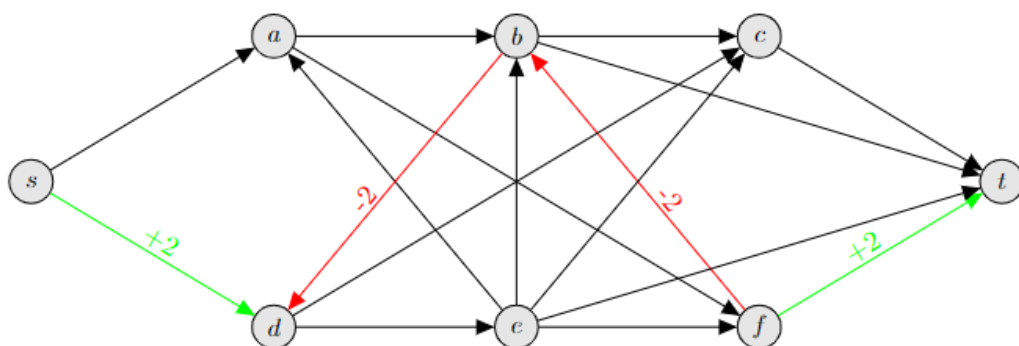
A normál utak mellett kereshetünk a javítóutas algoritmushoz hasonlóan is utakat s -ből t -be. Ezalatt azt kell érteni, hogy használhatunk egy élt visszafelé is, viszont ekkor nem növelni, hanem csökkenteni kell az értékét a kiválasztott 1 és 4 közötti számmal. Mivel egy hálózatban nem szeretnénk olyan éleket, amelyeknek kapacitása negatív, ezért azt ebben az esetben fontos kikötnünk, hogy ha visszafelé használunk egy élt, akkor a csökkentés után is egy nemnegatív érték legyen az élen, vagyis eredetileg az él értéke minimum annyi kell legyen, mint az az 1 és 4 közötti érték, amellyel csökkenteni szeretnénk.

Az előzőhöz hasonlóan itt is fontos az, hogy utakat keresünk, tehát egy csúcs csak egyszer szerepelhet, viszont a másik, miszerint csak olyan éleket használhatunk, ami nem lép vissza a vágás csúcsai közé, az csak félig lesz igaz. Ha a sima útkereséshez hasonlóan előre felé megyünk az élen, akkor az ugyanúgy nem lehet egy olyan él, aminek kezdőcsúcsa nem a vágás része, a végpontjában lévő csúcs viszont igen. Ennek miéértje az előzőhöz hasonlóan ugyanúgy kifejezhető, pontosan ugyanazon okok miatt rontaná el azt, hogy a kiválasztott vágás biztosan minimális vágás maradjon. Viszont, ha visszafelé megyünk egy élen, akkor az bármilyen két csúcs közötti él lehet. Egyrészt először nézzük azt az esetet, amikor egy olyan élen megyünk vissza, ami a vágásból kifelé mutat, szóval így ezzel fogunk visszalépni a vágás csúcsai közé. Ez azonban nem probléma, mivel

ugyan itt is többször fogunk kilépni a vágás csúcsaiból emiatt, viszont mindig eggyel kevesebbszer egy élen visszafele be is kell lépni hozzá, vagyis mindig eggyel kevesebbszer csökkentjük ugyanazzal a számmal a vágás valamelyik élet, mint amennyivel növeljük, emiatt ugyanúgy mindig egyszer növeljük igazából a kisorsolt számmal a vágás értékét. Olyan lehetőség elméletben lehetne, hogy a kiválasztott vágás ne nőjön, ugyanis megtörténhetne, hogy csak egy élen visszafele lépünk ki a vágásból, így a vágás éleit nem növeljük, ez azonban nem lehetséges, mivel az olyan éleken, amin visszafele léphetnénk ki csak 0 érték lehet, mivel azokat az éleket a fentebb leírtak miatt nem növelhetjük.

Emellett felmerülhet az a kérdés is, hogy biztosan az általunk választott él marad-e a minimális vágás ezáltal, hogy visszafele is mehetünk az éleken, mivel előfordulhat az, hogy a mi általunk választott vágás értékét ugyan csak egyszer növeltük, viszont egy másik random vágást egyszer sem, mivel abból csak egy élen visszafele léptünk ki. Ez ugyan lehetséges, de csak akkor, ha az addig nem volt minimális vágás, és esetleg ezzel azzá válhat. Ez azért triviális, mivel a minimális vágás értéke nem lehet kisebb a maximális folyam értékénél, és a maximális folyamot is ugyanúgy egyszer növeltük, mint a minimális vágást, ami jelen esetben például a kiválasztott vágásunk, de most még nem probléma, ha más vágás is minimális, így egy másik random is lehet az. Így tehát, ha visszafele megyünk egy élen, csökkentve annak értékét, akkor nem ronthatjuk el azt, hogy az általunk kiválasztott vágás minimális vágás maradjon.

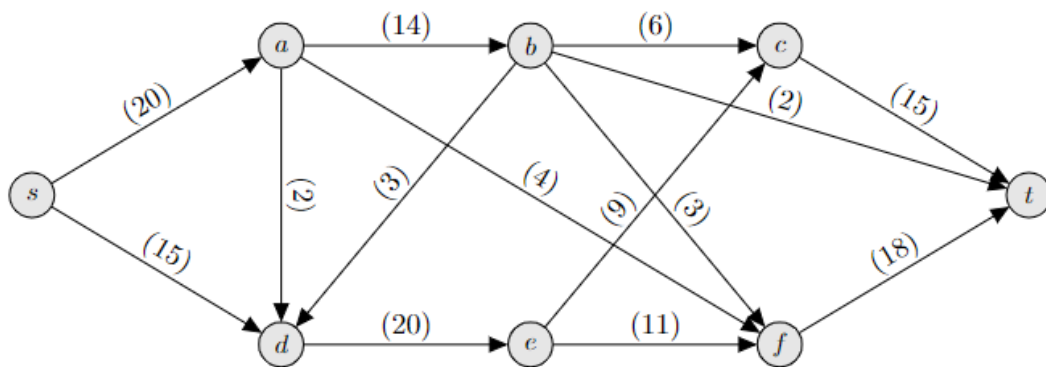
Egy ilyen útkeresés található a következő ábrán, ahol használtunk visszafele is éleket, azok értékét csökkentve. Most nem 1-gyel hanem 2-vel növeltük, illetve csökkentettük az élek értékét.



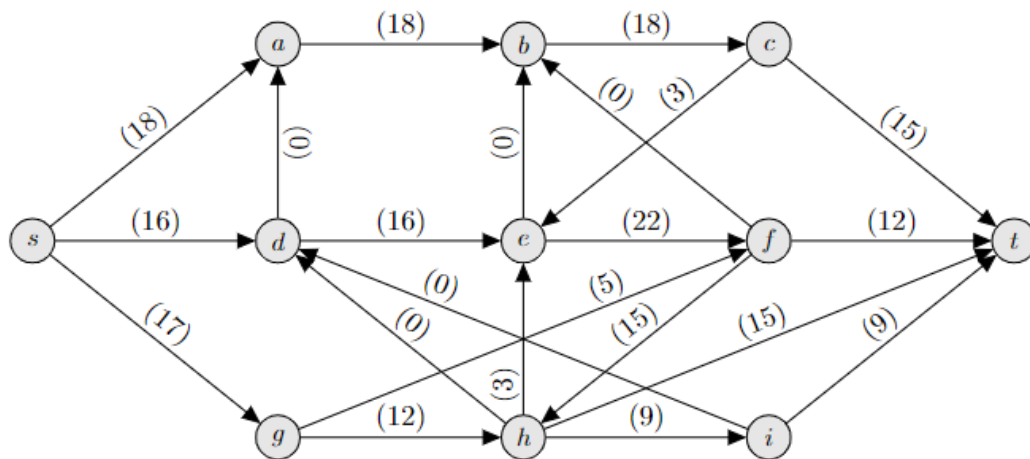
3.17. ábra: visszaélek élkapacitásának csökkentése az útban

3.5.3 Keletkezett folyamok

Az útkeresést néhányszor lefuttatva, mégpedig számszerűsítve az élek számának 1.3-szorosának egészrészével megkapjuk a folyamunkat. A két leírt módon az élek értékeit 0-ról növelve kaptuk meg az alább látható két folyamot, amit a 3.14-es és a 3.15-ös ábrán látható gráfokból hoztunk létre. Látható, hogy a 3.19-es ábrán látható folyamban az élek értéke több esetben is 0. Ez egyrészt, amikor kapacitás lesz belőle, akkor változni fog általában, viszont alapvetően, ha feladat lesz belőle a későbbiekben, akkor annak megoldása ettől könnyebb lesz, így ezzel később még kell számolnunk.



3.18. ábra: Egy generált 2×3-as folyam



3.19. ábra: Egy generált 3×3-as folyam

3.6 Folyam átalakítása hálózattá

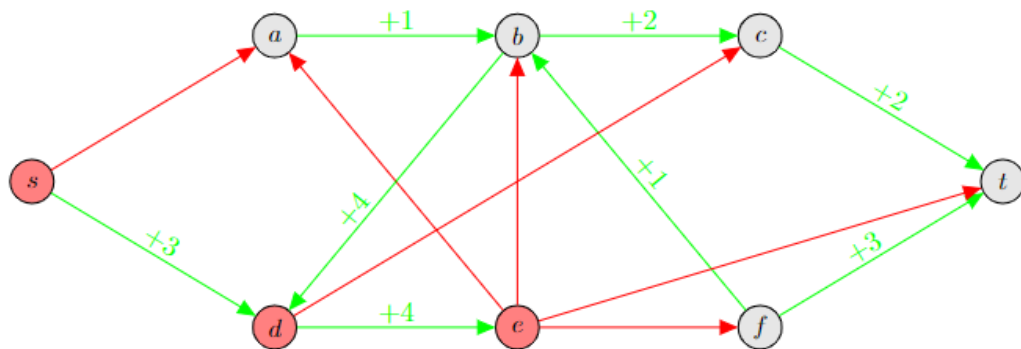
Megvan tehát egy folyamunk, amely később egy maximális folyama lesz a hálózatunknak, ami a feladat lehet ezután. Ebből kell most hálózatot képeznünk úgy, hogy

valóban megmaradjon a generált folyamunk maximális folyamnak, valamint, hogy a kiválasztott vágásunk minimális vágás maradjon, sőt azt szeretnénk, hogy az maradjon az egyetlen minimális vágása a hálózatnak.

3.6.1 Élek növelése a vágás éleinek kivételével

Ahhoz, hogy teljesüljön az is, hogy a generált folyam legyen egy maximális vágás, valamint az, hogy minimális vágás maradjon a kiválasztott vágásunk az kell, hogy a vágás egyetlen élén se növeljük már az értéket. Viszont azt is szeretnénk elérni, hogy a kiválasztott vágás legyen az egyetlen minimális vágás. Mivel még lehet más minimális vágás is, így ezt úgy tudjuk elérni a legegyszerűbben, hogy minden másik vágás értékét, akár minimális a vágás, akár nem, növelünk valamennyivel. Ezt pedig úgy tudjuk a legkönnyebben elérni, hogy a vágás élein kívül minden más él értékét növeljük.

A növelés értéke ismét egy 1 és 4 közötti szám, azonban most nem fontos az, hogy minden élt ugyanannyival növeljünk. Így például a 3.20. ábrán látható, ahogy egy folyamat hálózattá alakítunk, amely hálózat minimális vágása az s, d, e csúcsokból fog állni.



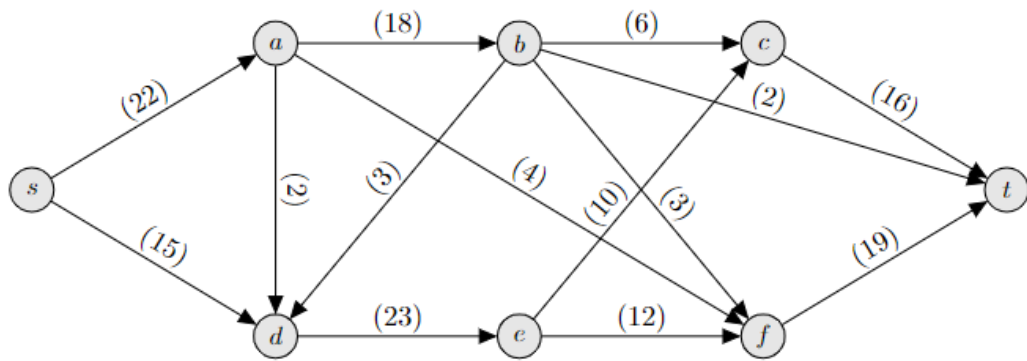
3.20. ábra: Nem vágásélek kapacitásának növelése

3.6.2 Keletkezett hálózatok

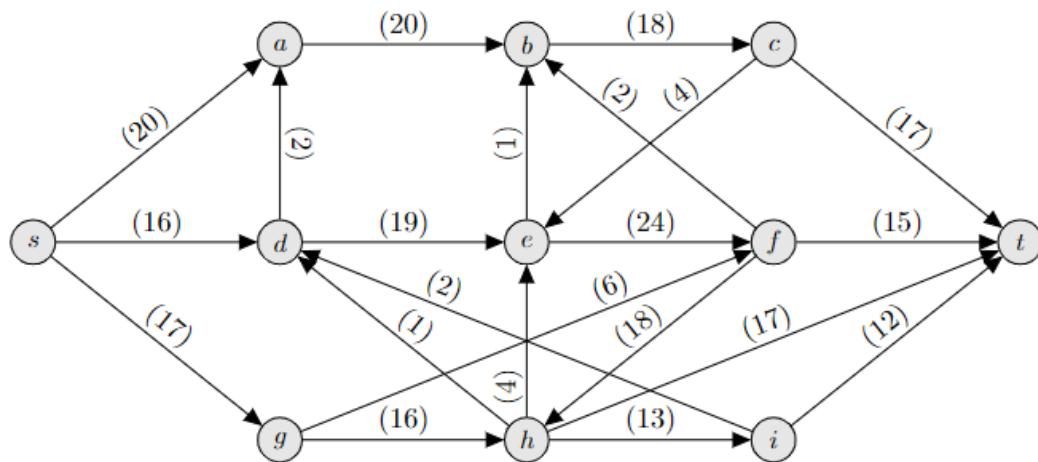
Így tehát a folyamokból hálózatokat gyártottunk, ami gyakorlatilag már egy kész feladatnak is megfelel, és tudjuk már a minimális vágását is, valamint egy hozzá tartozó maximális folyamot is. Így tehát, ha ez lesz a generált feladatunk, akkor a megoldás is megvan hozzá. Azonban az kérdéses lesz, hogy megfelelő feladat lesz-e ez nekünk, mivel

attól még, hogy a hálózat paraméterei megfelelnek nekünk, nem biztos, hogy a feladat nehézsége is számunkra megfelelő lesz.

A 3.18. és 3.19. ábrákon található folyamatokból hoztuk létre az alább látható hálózatokat, amelyeket feladatoknak is feladhatunk a későbbiekben.



3.21. ábra: Elkészült 2×3-as hálózat



3.22. ábra: Elkészült 3×3-as hálózat

4. fejezet

Feladat nehézsége

A hálózati folyam feladatunkhoz szükséges hálózat előállítás a megadott paraméterek alapján tehát már megvan, ez már akár egy megfelelő output is lehetne, azonban nem elég nekünk ennyi, mivel még közel sem biztos, hogy a feladat a kívánt nehézségű lesz. Ezért szükségünk van valahogy megmérni ezeknek a feladatoknak a nehézségét, ami alapján egy megfelelőt tudunk generálni. Ez nem olyan egyszerű feladat, mivel nem minden hallgató ugyanúgy gondolkodik, így nem lehet például az alapján megmondani egy feladat nehézségét, hogy egy módszerrel vagy egy algoritmussal milyen bonyolult megoldani a feladatot, mert nem feltétlenül azt fogja minden hallgató használni. Viszont vannak a hálózatnak olyan tulajdonságai, ami alapján minden hallgató számára nehezebb vagy könnyebb lehet a feladat, és mi ezen tulajdonságok alapján próbálunk egy 0 és 10 közötti mérőszámot alkotni a feladat nehézségére.

4.1 Élekkel kapcsolatos nehézségek

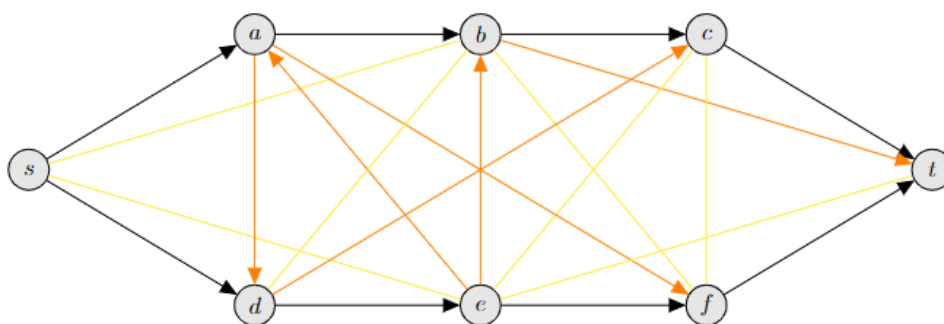
Sok múlik azon, hogy a gráf hogyan néz ki, és ezen belül is legfőképpen az, hogy a gráf élei milyen módon vannak behúzva. Így elég sokat számít a nehézségbe bizonyos éltípusoknak a számossága, ezekről lesz most szó. Fontos, hogy az olyan éleket, amelyeknek a kapacitása 0, azokat nem vesszük figyelembe, gyakorlatilag úgy tekinthetők, mintha azok az élek nem léteznének, így sehova nem kell beleszámolnunk őket.

4.1.1 Élek száma

Az első szimplán az élek száma a gráfban. Nyilván minél több éle van a gráfnak, annál nehezebb lesz az abból készített feladat, így összeszámoljuk őket. Ahhoz, hogy ez az érték ne nőhessen a csúcsok száma alapján a végtelenségig, így igazából nem arra vagyunk kíváncsiak valójában, hogy hány éle van a gráfnak, hanem hogy a behúzható élek hányad része került bele a gráfba. Természetesen itt a keret éleit nem számoljuk bele, valamint csak azokat az éleket vizsgáljuk, amelyek a szabályok szerint behúzhatóak lennének. Emellett itt most nem vesszük figyelembe az élek irányát, vagyis a két csúcs

közé behúzható éleket egynek vesszük, hiába lehetne adott esetben mindkét irányba behúzni.

A 4.1-es ábrán láthatjuk narancssárgával a kereten kívül behúzott éleket, sárgával pedig a maradék nem behúzott, de a szabályok szerint behúzható élt. A kettő számának az összege lesz az összes behúzható él, a narancssárga élek száma pedig a behúzott élek, ezeknek a számoknak az arányára vagyunk kíváncsiak a nehézség ezen részének kiszámításához.



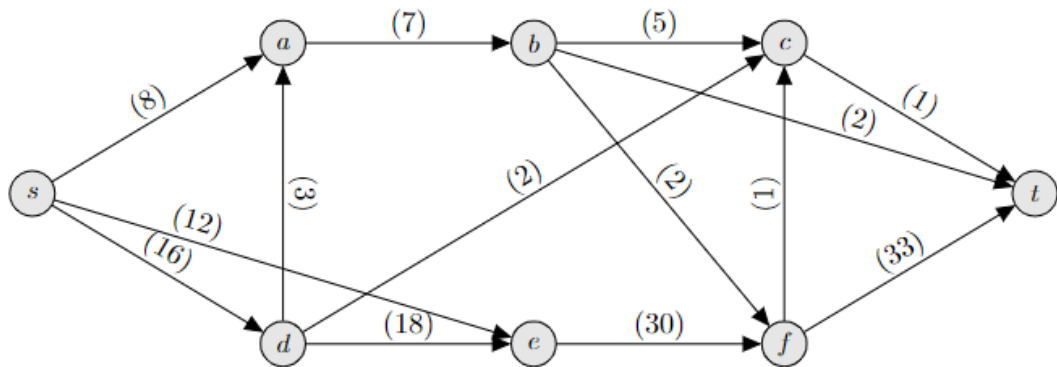
4.1. ábra: Élek aránya a maximumhoz képest

Látható tehát, hogy 6 él lett behúzva a 13-ból, vagyis az arány, amire kíváncsiak vagyunk $6/13$. Alapból ez egy 0 és 1 közötti szám lehet csak, mivel nyilvánvalóan se negatív számú, sem a maximálisnál több él nem húzható be. Viszont ezt az eredményt megszorozzuk 2-vel, így $12/13$ -ot kapunk. Szeretnénk azonban, ha továbbra is 0 és 1 közötti lenne a szám, így, ha a szorzás után 1-nél nagyobb számot kapnánk, akkor arra megkapja a maximális 1-et, de nem többet, vagyis 1-re csökkentjük az értéket. Erre azért van szükség, mert szeretnénk, ha nem csak akkor kapná meg a maximális nehézséget, ha minden egyes lehetséges él be van húzva. Ez a feladat nehézségéhez pluszos előjellel, 3-szoros súlyozással lesz hozzáadva. Így tehát az élek számára kapott nehézség képlete $\min(1; 2 \times \text{behúzott élek száma} / \text{összes behúzható él száma}) \times 3$.

4.1.2 Nullélék száma

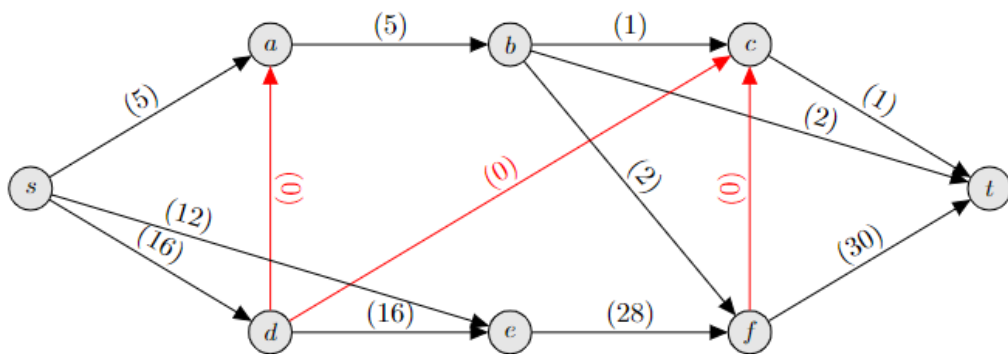
Lehetnek olyan élek a hálózatban, amiknek ugyan nem 0 a kapacitása, így ott lesz az élek között, de az általunk eltárolt maximális folyamban, ami a feladat megoldása, ott pár élre az érték 0 lesz. Ez ugyan nem feltétlenül az egyetlen megoldása a feladatnak, de

általában könnyebb egy ilyen megoldást megtalálni, ahol sok élnél az élre írt érték 0, és ha már biztosan van egy ilyen megoldása a feladatnak, még hozzá a mi általunk eltárolt, akkor tudjuk, hogy az könnyíti a feladat megoldását. Az ilyen éleket, aminek a hálózatban nem 0 a kapacitása, de a maximális folyamunkban 0 az élre írt érték, nevezzük el nulléleknek.



4.2. ábra: Hálózati folyam feladat

A fenti ábrán látható egy hálózati folyam feladat, lent pedig ehhez egy megoldás, vagyis a program által eltárolt maximális folyam. Látható, hogy a piros élekre írt érték 0, vagyis ezek nullélek.



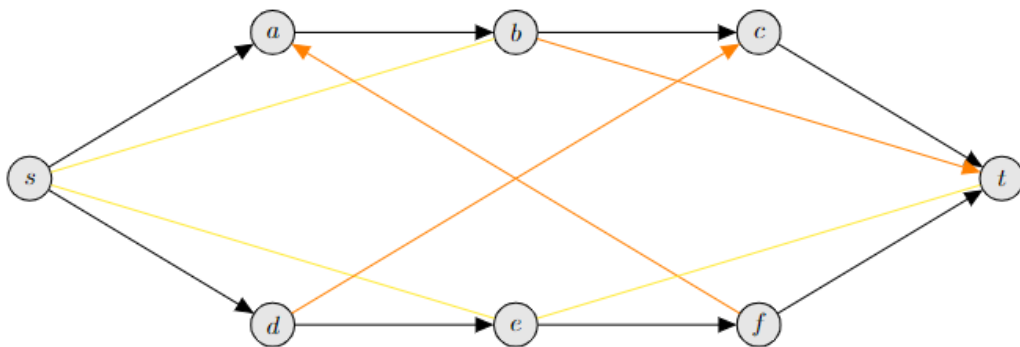
4.3. ábra: Nullélek a maximális folyamban

3 darab nulléle van tehát a feladatnak, amit a nehézségbe beleszámítunk. Igazából ezt is a képletünkben az élek számába számítjuk bele, még hozzá úgy, hogy a behúzott élek számából levonjuk a nullélek számának a felét. Mivel azt utána megkértszerezzük,

így gyakorlatilag a behúzott élek számának kétszereséből levonjuk a nullélek számát. Fontos viszont hozzátenni, hogy elméletben lehet több nullél, mint a behúzott élek kétszerese, mivel nullélek lehetnek a keret élei is, így ha ez előfordulna, vagyis negatív lenne a kivonás eredménye, akkor 0-ra átírjuk az eredményt. Tehát a fent leírt képletet a következőre egészítjük ki: $\min(1; (\max(0; 2 \times \text{behúzott élek száma} - \text{nullélek száma})) / \text{összes behúzható él száma}) \times 3$.

4.1.3 Hosszú élek száma

Minimális mértékben nehezíti a feladatunkat az, ha több hosszú él található meg benne és nem csak az ugyanabban vagy a szomszédos oszlopban lévő csúcsba húzott élek. Tehát hosszú éleknek a kettővel balra vagy kettővel jobbra mutató éleket nevezzük. Ugyanúgy itt is arra leszünk kíváncsiak, hogy a szabályok szerint behúzható hosszú élek közül hányad részét húztuk be a gráfban. Ugyanúgy itt sem számít az él iránya, a két csúcs közé behúzható éleket egynek számítjuk. Az alábbi ábrán narancssárgával vannak jelölve a behúzott, és sárgával pedig a behúzható, de nem behúzott élek.

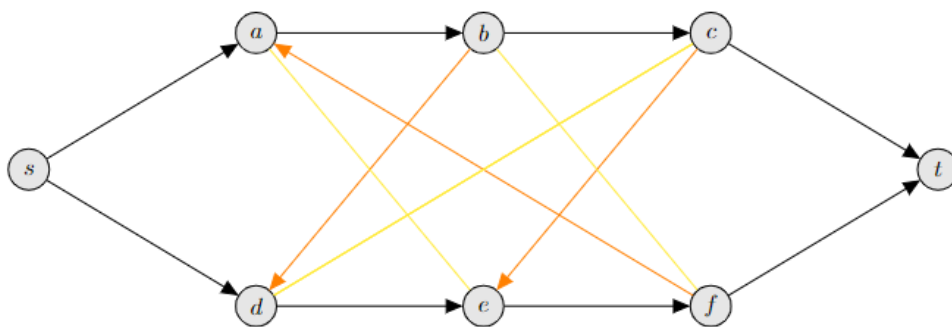


4.4. ábra: Hosszú élek aránya a maximumhoz képest

Láthatjuk tehát, hogy a behúzható hosszúélek fele lett hozzáadva a gráfhoz. Viszont itt sem szükséges az, hogy minden hosszú él be legyen húzva ahhoz, hogy a feladatunk megkapja a maximális nehézséget. A maximálisan behúzható hosszú élek számát megszorozzuk 0.7-tel, és ezzel fogjuk elosztani a behúzott hosszú élek számát, valamint itt is maximum 1-et szeretnénk kapni eredménynek, így a képlet a következő lesz: $\min(1; \text{hosszú élek száma} / (\text{összes behúzható hosszú él száma} \times 0.7))$, ezt egyszeres súlyozással, pozitív előjellel adjuk a nehézséghez.

4.1.4 Visszafele mutató élek száma

Az utolsó éltípus, amelynek számosságára kíváncsiak vagyunk az visszafele mutató élek. Ezek is nehezítik a feladatot, jobban, mint a hosszú élek. Ugyanúgy itt sem arra vagyunk kíváncsiak, hogy pontosan hány darab visszafele mutató élt húztunk be, hanem arra, hogy a maximálisan behúzhatók közül milyen arányban kerültek bele ezek az élek a gráfba. A lenti ábrán láthatunk egy gráfot, amelyben a narancssárgák a behúzott visszafele mutató élek, a sárgák pedig a nem behúzott, de behúzhatóak.



4.5. ábra: Visszafele mutató élek aránya a maximumhoz képest

Itt a behúzhatók fele van ténylegesen behúzva. Itt azonban ennél nem feltétlenül akkor lesz a legnehezebb, ha minél több visszafele mutató él van, mivel, ha csak visszaéleket húzunk be a gráfba, akkor az könnyebb lenne, mintha csak a behúzottak fele lenne visszaél. Azt szeretnénk, hogy akkor legyen a maximális a visszafele mutató élek nehézsége, ha a behúzható éleknek az $1/3$ -a lenne benne a gráfban. Viszont itt is kell, hogy ne csak ekkor kapja meg a maximumot a nehézség, így egy intervallumot használunk. Így ezen nehézség kiszámítására az alábbi bonyolultabb képletet használjuk: $\min(1; 1.8 \times (1 - (\max(0; 1 - (\text{összes behúzható visszafele mutató él száma} / 3 - \text{visszafele mutató élek száma}) / (\text{összes behúzható visszafele mutató él száma} / 3))))$). Ez tehát azt csinálja, hogy ha 0 vagy több, mint a behúzható visszafele mutató élek kétharmada lesz a gráfban, akkor 0-t ad eredményül, az $5/27$ és $14/27$ arányban behúzott visszafele mutató élek között 1-et ad eredményül, egyébként pedig 0 és $5/27$ között egyenletesen nő, $14/27$ és $2/3$ között pedig egyenletesen csökken a nehézség. Az $1/2$ beleesik abba az intervallumba, amiért a maximális nehézséget kapja, így a fenti gráf 1-et kap rá. Ez az egészet 2-es súlyozással, pozitív előjellel adjuk hozzá a nehézséghez.

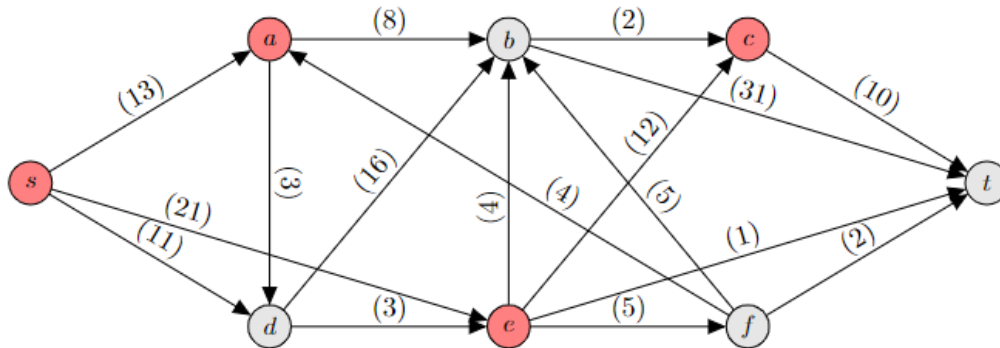
4.2 Csúcsok száma

A csúcsok száma egy olyan paraméter, amit mi adunk meg, mint a gráf egyik tulajdonsága, ez tehát nem a generáláskor keletkezik, így nem is lenne muszáj beleraknunk a nehézségbe, mivel amikor egy feladatot generálunk, akkor mindig ugyanakkora csúcsszámú gráfokat hasonlít össze, így mindig ugyanakkora nehézséget kapunk ebben a pontban. Ennek ellenére nyilván egy olyan feladat, amiben egy jóval több csúcsú gráf van, mint a másikban, az nehezebb lesz, ha a többi része a nehézségnek ugyanaz az érték lesz, így egy minimális mértékben a képletünkben is szerepel a csúcsok száma. Méghozzá úgy, hogy a 20 csúcsú gráf és a felette lévők megkapják a maximumot, vagyis az 1-et, alatta pedig nézzük, hogy hányad része a 20-nak a csúcsok száma. Vagyis a képlet a következő: $\min(1; \text{csúcsok száma}/20)$. Ez kétszeres súlyozással, pozitív előjellel kerül hozzáadásra a nehézséghez.

4.3 Vágáserősség

Fontos része a nehézség kiszámításának a minimális vágás vizsgálata. Ugyanis vagy a minimális vágás megtalálása is a feladat része szokott lenni, vagy csak egyszerűen sokat segít a maximális folyam megtalálásához, hogy megtaláljuk a minimális vágást. Így minél nehezebb megtalálni, értelemszerűen annyival nehezebb a feladat is. Azt, hogy mitől is lesz nehéz ennek a vágásnak a megtalálása, azt a vágás, valamint a többi csúcs elhelyezkedése közötti kapcsolat jellemzi. Mi kétféle nehezítő tényezőt veszünk figyelembe. Az egyik az, hogy egy csúcs benne van a vágásban, az azonos sorban lévő, tőle egyvel balra található csúcs viszont nincs. Ez 1 pontot ad hozzá a vágáserősségbe. A másik pedig ennek a továbbfejlesztett verziója, mégpedig, hogy egy csúcs benne van a vágásban, a tőle egyvel balra levő, azonos sorban lévő csúcs nincs benne, viszont az attól egyvel balra levő azonos sorban lévő csúcs ismét benne van. Ez az előző miatt már alapból kap egy pontot az erősségbe, viszont kap még egy pontot. Vagyis, ha olyat találunk a hálózatban, hogy egy sorban két vágásban benne lévő csúcs közrefog egy harmadik vágásban nem lévő csúcsot (úgy, hogy ez 3 egymás melletti csúcs), akkor ez ad 2 pontot az erősségbe. Ismételten nem arra vagyunk kíváncsiak, hogy a maximum erősségből mennyit fog elérni a hálózatunk, vagyis itt is egy arányszám lesz a lényeg, és nem az, hogy valójában hány pontot kap az erősségre.

Alább látható egy hálózat, amelynek az s, a, c, e csúcsokból áll a vágása, ezen megvizsgálhatjuk a vágáserősség kiszámítását.



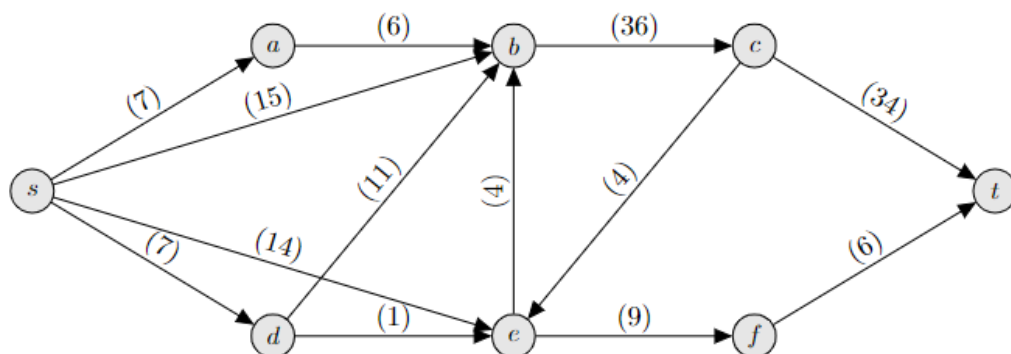
4.6. ábra: Vágás csúcsai

Láthatjuk, hogy a gráf felső sorában két vágáscsúcs összefog egy nem vágáscsúcsot, így az 2 pontot ad az erősségbe, az alsó sorban pedig csak található egy nem vágáscsúcs egy vágáscsúctól eggyel balra, így ez 1 pontot ad az erősségbe. Így tehát 3 pontot kapott összesen, és kiszámítható, hogy 4 pont lenne a max, ha e helyett d és f lenne benne a vágásban. Így tehát 3/4-e a maximum vágáserősségnek a kapott erősség. Ugyanígy itt sem szükséges, hogy a maximális nehézséghez a maximális erősség legyen a vágásban, így itt a maximális erősséget 0.35-tel felszorozzuk, mivel nagyon kicsi az esélye annak, hogy a maximumhoz közel legyen a vágás erőssége. Ugyanígy itt is maximum 1-et szeretnénk kapni erre a nehézségre, így a képletünk, amit használunk erre a nehézségre: $\min(1; \text{vágáserősség}/(\text{maximális vágáserősség} \times 0.35))$. Ezt kétszeres szorzóval, pozitív előjellel adjuk hozzá a nehézségünkhöz.

4.4 Szórás

Könnyítheti a feladatunkat az, ha az élekre írt kapacitások nagy mértékben eltérnek egymástól, vagyis van sok nagy és sok kicsi kapacitású él. Ha minden élnek nagyjából ugyanannyi lenne a kapacitása, akkor jóval nehezebb lenne megoldani a feladatot. Így tehát ki kell számolni az élek kapacitásainak szórását. Ez általában egy elég nagy szám lesz, nekünk nincs szükségünk ekkora értékre, így a szórást elosztjuk 8-cal. Nem szeretnénk azonban, ha ez az érték kisebb lenne mint 1, ezért ebben az esetben 1-et adunk neki. Viszont annyira azt sem szeretnénk, ha túl nagy lenne ez az érték, mivel annyira azért nem nagy tényező a nehézségben, ezért 1.25-ben maximalizáljuk.

Tekintsük például az alábbi hálózatot, számoljuk ki rá, hogy mekkora nehézséget rendelünk hozzá a szórás által.

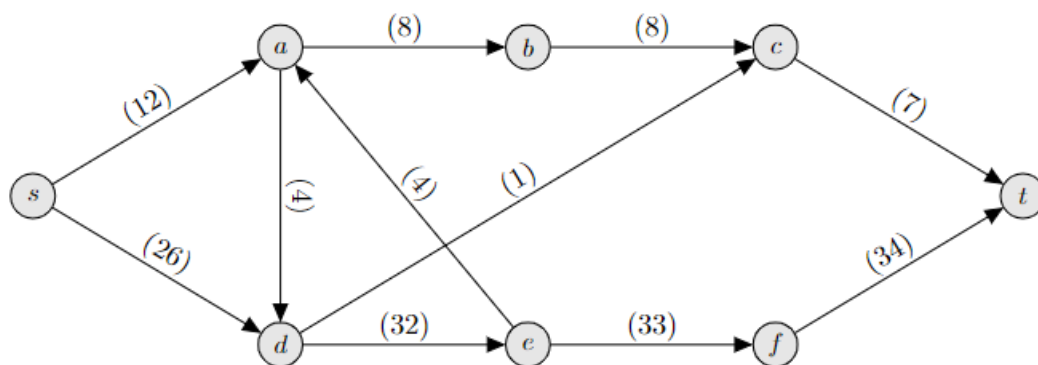


4.7. ábra: Egyszerű hálózat szórás vizsgálatához

Az élek kapacitásainak szórása egy egyszerű szórásképlettel kiszámítva nagyjából 10.575. Ezt 8-cal elosztva nagyjából 1.322-t kapunk eredményül, ami nagyobb, mint 1.25, ezért a 1.25 lesz a kapott eredményünk. Vagyis a képletünk a szórás nehézségének kiszámolására: $\max(1; \min(1.25; \text{élek kapacitásának szórása}))$. Ezzel a számmal az eddig kapott eredményünket elosztjuk, és így kapjuk meg a nehézséget.

4.5 Teljes képlet

A teljes képletünk tehát a következő: $(\min(1; (\max(0; 2 \times \text{behúzott élek száma} - \text{nullélek száma}) / \text{összes behúzható él száma}) \times 3 + \min(1; \text{hosszú élek száma} / (\text{összes behúzható hosszú él száma} \times 0.7)) + \min(1; 1.8 \times (1 - (\max(0; 1 - |\text{összes behúzható visszafele mutató él száma} / 3 - \text{visszafele mutató élek száma}| / (\text{összes behúzható visszafele mutató él száma} / 3)))) \times 2 + \min(1; \text{csúcsok száma} / 20) \times 2 + \min(1; \text{vágáserősség} / (\text{maximális vágáserősség} \times 0.35))) \times 2 / \max(1; \min(1.25; \text{élek kapacitásának szórása}))$. Ezzel kapjuk meg a feladat nehézségét, ami alapján pedig eldöntjük, hogy megfelelő-e nekünk a kapott hálózat. Számoljunk nehézséget az alábbi generált hálózatra:



4.8. ábra: Egyszerű hálózat a képlet bemutatásához

A hálózathoz nem egyértelműen látható, de hozzá tartozó minimális vágás az s, a, b és c csúcsokból áll, az általunk eltárolt maximális folyamatra pedig 2 darab élre írt érték 0.

Azt látjuk, hogy 3 darab él lett behúzva a 13-ból, és ez a 2 darab nullél miatt a nehézséghez a $\min(1; (\max(0; 2 \times \text{behúzott él szám} - \text{nullél szám}))/\text{összes behúzható él szám}) \times 3$ képlet alapján nagyjából 0.923-et ad.

Azt is látjuk, hogy 1 db hosszú él van behúzva a 6-ból. Így ez a nehézséghez a $\min(1; \text{hosszú él szám}/(\text{összes behúzható hosszú él szám} \times 0.7))$ képlet alapján nagyjából 0.238-et ad.

Látjuk továbbá az 1 db visszaélt is, a maximális 6-ból, ez a nehézséghez a $\min(1; 1.8 \times (1 - (\max(0; 1 - |\text{összes behúzható visszafele mutató él szám}|/3 - \text{visszafele mutató él szám}|/(\text{összes behúzható visszafele mutató él szám}/3)))) \times 2$ képlet alapján 1.8-at ad.

Ezen felül az s, a, b, c vágásban nem kap semmire erősséget, vagyis a $\min(1; \text{vágáserősség}/(\text{maximális vágáserősség} \times 0.35)) \times 2$ képlet által 0-t kapunk eredménynek.

A gráfnak, amiből a feladat készült 8 csúcsa van, így arra a $\min(1; \text{csúcsok szám}/20) \times 2$ képlet alapján 0.8 nehézséget kap.

Az él kapacitásának szórása pedig 8-cal leosztva pedig nagyjából 1.555, ami nagyobb, mint 1.25, így 1.25 lesz a szóráshoz tartozó érték.

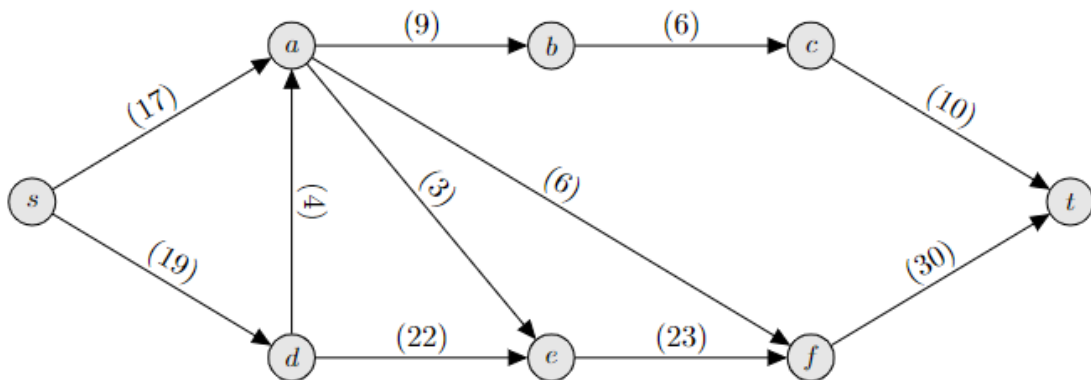
A teljes nehézség tehát $(0.923 + 0.238 + 1.8 + 0 + 0.4)/1.25 = 3.001$, így tehát ez egy tökéletes 3-as nehézségű feladatnak tekinthető.

Így tudunk tehát nehézséget rendelni egy hálózathoz. Egy hálózathoz pedig akkor lesz feladat, ha annak nehézsége minél közelebb áll a kívánt nehézséghez. Ehhez $1000 \times$ lefuttatjuk a hálózatgenerálást, és mindegyiknek megvizsgáljuk a nehézségét. Természetesen, ha mi adjuk meg a gráfot bemenetként, akkor arra generáljuk újra, egyébként pedig a gráfot is újra generáljuk. Amelyiknek pedig a legközelebb lesz a nehézsége a kívánthoz, az „nyert”, vagyis az lesz a kapott feladatunk. Azt említettük, hogy az olyan éleket, amiknek 0 a kapacitása azokat gyakorlatilag nem tekintjük a hálózat részének. Ez akkor okoz problémát, ha ez az él a keret egyik éle, mivel azt nem szeretnénk, hogy olyan feladatot kapjunk, ahol hiányzik a keretnek egy éle. Így ebben az esetben automatikusan újra generáljuk a hálózatot, és nem is fog beleszámítani az 1000 próbálkozásba. Ez alól kivételt képez az, ha mi adjuk meg a gráfot, amelyből a folyamat

létrehozzuk, mivel ekkor megeshet, hogy arra a gráfra nem generálható olyan folyam, amiben nem 0 egy keretél kapacitása, így emiatt végtelen ciklusba kerülnénk.

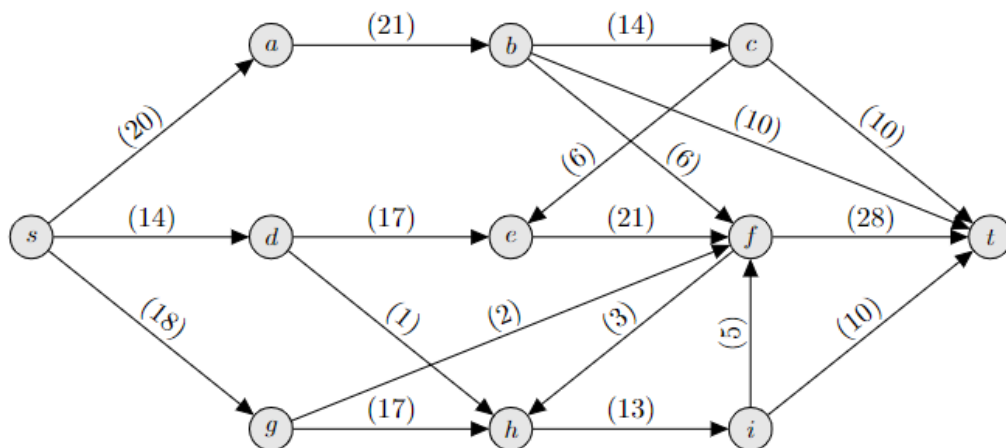
4.6 Példák könnyű és nehéz feladatokra

Most nézzünk néhány példát könnyebb és nehezebb feladatokra is.



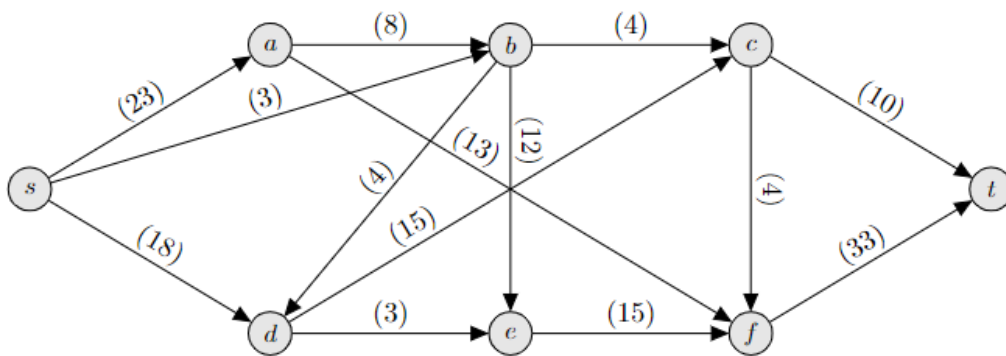
4.9. ábra: 2-es nehézségű feladat

A fenti feladat 2-es nehézséget kapott, mivel elég kevés éle van, nincs visszafele mutató éle, s, a és b csúcsokból áll minimális vágás, amit nem túl nehéz megtalálni, valamint hosszú éle is csak egy van.



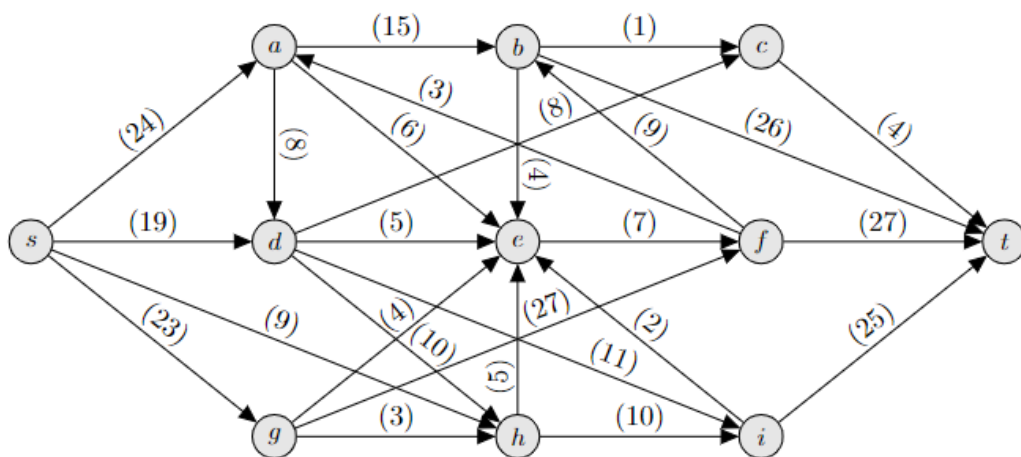
4.10. ábra: 4-es nehézségű feladat

A fent látható feladat 4-es nehézségű, mivel ugyan van 2 visszafele mutató, és 2 hosszú éle is, de még mindig viszonylag kevés éle van, és az s, g, h csúcsokból minimális vágása sem ad sokat a nehézséghez.



4.11. ábra: 6-os nehézségű feladat

A fenti feladat 6-os nehézséget kapott, mivel már egészen sok éle be van húzva, van 3 hosszú éle, és 1 visszafele mutató éle is, valamint az s, a, c, d minimális vágása is ad hozzá sokat a nehézséghez.



4.12. ábra: 8-as nehézségű feladat

Az utolsó feladat pedig 8-as nehézségű, látható, hogy rendkívül sok éle van, van 3 visszafele mutató él, hosszú élből is van néhány, és az s, a, c, d, e és h csúcsokból álló minimális vágása is sokat ad a nehézséghez.

5. fejezet

A program bemenetei

A korábban írtak szerint néhány dolgot meg kell adnunk a programnak, amiből az a feladatot le tudja generálni. Ezek a hálózat paraméterei, vagy a gráf sorainak és csúcsainak száma, illetve emellett még a kívánt feladatnehézséget is meg kell adnunk. Ebből már képesek leszünk az eddig leírtak segítségével megfelelő feladatot generálni, és azt később felhasználni. Azonban kérdés az, hogy hogyan is tudjuk a program számára átadni a kívánt paramétereket.

5.1 Grafikus felület kezdőoldala

A paraméterek átadására két mód van, az, hogy a grafikus felület segítségével adjuk át, vagy fájlból generáljuk a feladatot. A program elindításakor választhatunk a két lehetőség közül a grafikus felület kezdőoldalán az 5.1. ábra alapján.

Hálózat generálása

☒ Paraméterek megadásával ☐ Fájlból

Gráf sorainak száma

☒ 2 ☐ 3 ☐ 4 ☐ 5

Gráf csúcsainak száma

☒ 6 ☐ 8 ☐ 10 ☐ 12

Feladat nehézsége

0 1 2 3 4 5 6 7 8 9 10

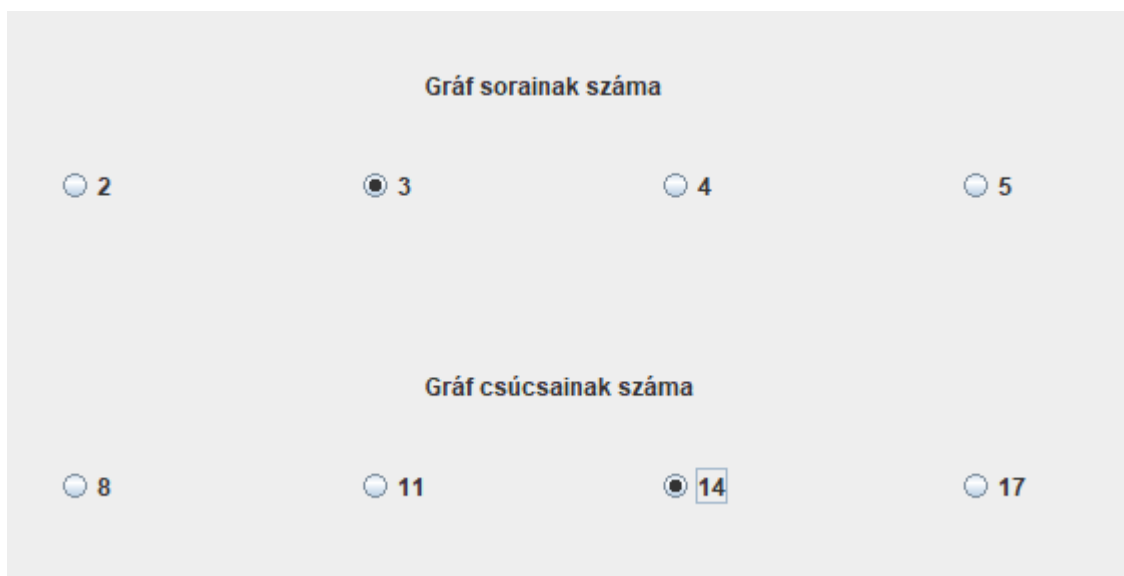
Feladat generálása

5.1. ábra: Kezdőképernyő

A grafikus felhasználói felületünk Java Swing alapú, és leginkább a program kimenetekor fog nagyobb szerepet játszani, de a paraméterek megadásakor is hasznát vehetjük. Amikor azt szeretnénk megadni, hogy fájlból vagy paraméterek megadásával szeretnénk a feladatot generálni, akkor egy JRadioButton segítségével választhatunk a lehetőségek közül, és ez alapján változik a felületen az, hogy miket választhatunk még ki. Ezután a Feladat generálása gombra kattintva indul el a hálózati folyamat feladat elkészítése. Az első kérdésben, miszerint a grafikus felületen vagy fájlból szeretnénk megadni a paramétereket, az előbbi választás az alapértelmezett, ez van bejelölve a program indításakor.

5.2 Hálózat generálása paraméterek megadásával

Amennyiben a Hálózat generálása pontnál a Paraméterek megadásával lehetőség van bejelölve, akkor megjelenik a paraméterek kiválasztásának felülete. Ez egyrészt a gráf tulajdonságai, amiből a hálózatot létrehozzuk majd.



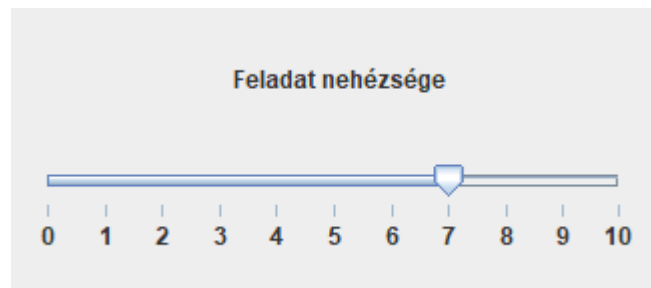
The image shows a graphical user interface for selecting graph parameters. It consists of two sections. The first section, titled "Gráf sorainak száma", contains four radio buttons labeled 2, 3, 4, and 5. The radio button for 3 is selected. The second section, titled "Gráf csúcsainak száma", contains four radio buttons labeled 8, 11, 14, and 17. The radio button for 14 is selected and is highlighted with a blue rectangular box.

5.2. ábra: Gráf sorai és csúcsai számának megadása

Először a gráf soraira vagyunk kíváncsiak, itt az alapértelmezett érték a 2. Aztán a gráf csúcsainak száma a kérdés. Mindkettőt szintén JRadioButton-k segítségével adhatjuk meg. Az, hogy a gráf csúcsszámainak megadásakor milyen értékek közül választhatunk, az a gráf sorainak száma alapján jelenik meg, úgy, hogy a 3.1 Csúcsok elrendezése fejezetben leírt csúcselrendezés szabályai teljesülhessenek. Így például látható, hogy a 2 sornál kiválasztható csúcsszámok 6, 8, 10 és 12 voltak, viszont 3 sornál

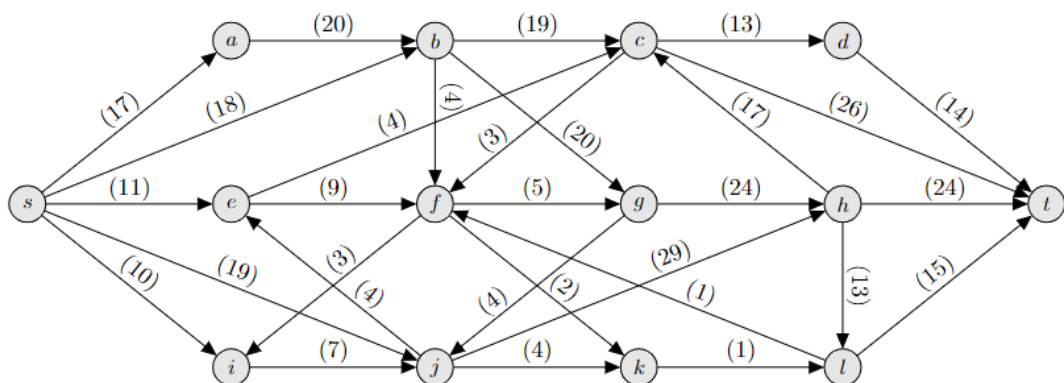
már 8, 11, 14 és 17, szóval gyakorlatilag azt választhatjuk ki, hogy a gráfnak 2, 3, 4 vagy 5 oszlopa legyen.

A gráf tulajdonságai után a feladat nehézségét kell megadnunk egy 0 és 10 közötti egész számmal. Ezt egy JSlider segítségével tehetjük meg, ahol 0-tól 10-ig egyesével be vannak jelölve az értékek, és a kívánt nehézségre kell húznunk a csúszkát. Alapértelmezetten 5-ös nehézségre van állítva. Az alábbi képen látható, hogy 7-es nehézségűre van állítva a feladat.



5.3. ábra: Feladatnehézség megadása

Az 5.2-es és 5.3-as ábrákból összerakva tehát egy 3 soros, 14 csúcsból álló gráfból generálunk egy 7-es nehézségű feladatot. Ezzel a bemenettel hoztuk létre az alábbi hálózati folyam feladatot a Feladat generálása gombra kattintva.

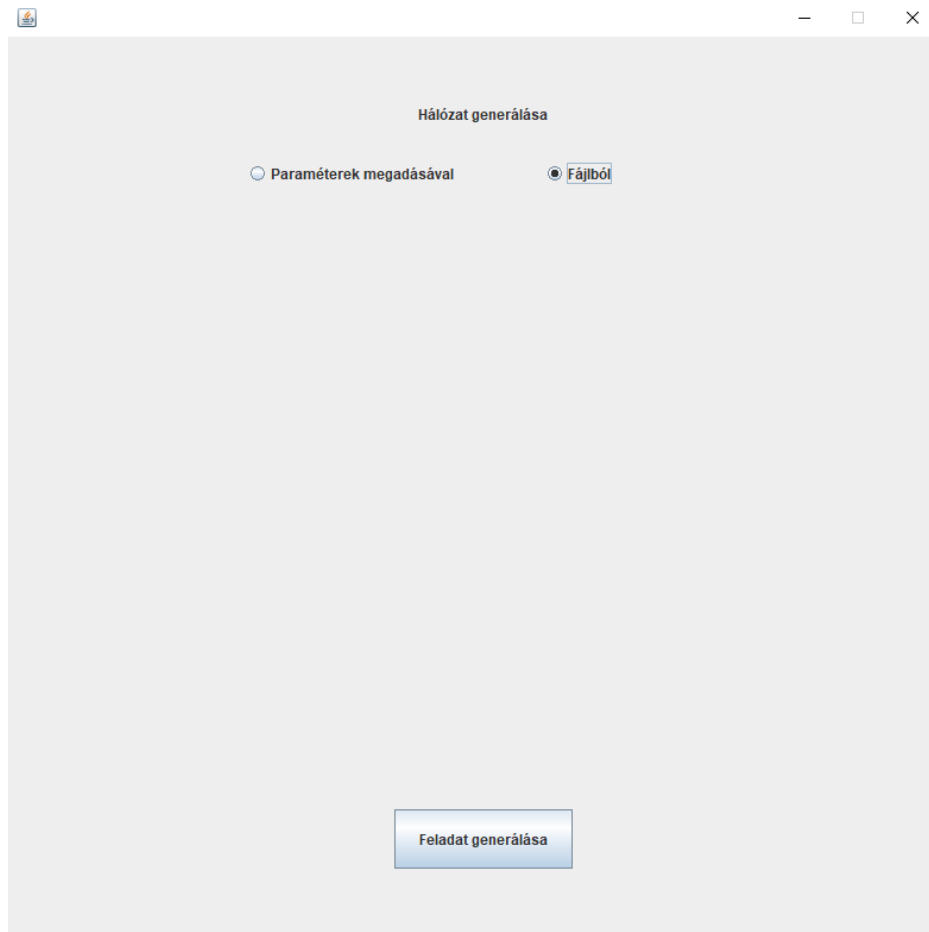


5.4. ábra: Generált 3×4-es, 7-es nehézségű feladat

Így adhatjuk meg tehát a grafikus felületen a paramétereket a feladat generálásához, azonban nem csak ez az egy lehetőségünk van.

5.3 Hálózat generálása fájlból

Fájlból is generálhatjuk a feladatot, méghozzá egy in.txt elnevezésű szöveges fájlt használhatunk e célra bemenetként. Ha ezt a lehetőséget választjuk, akkor a kezdőoldalon a Hálózat generálása résznél a Fájlból pontot kell kiválasztanunk, majd ezután rányomi a feladat generálása gombra.



5.5. ábra: Hálózat generálása fájlból

Ennél a lehetőségnél ugyan lesz kétféle verzió is, hogy egy fájlból hogyan visszük be az adatokat, azonban a két lehetőség közül nem a grafikus felületen választunk, hanem a fájl első sora fogja jelezni ezt, hogy melyiket szeretnénk.

5.3.1 Hálózat generálása random gráffal

Az első gyakorlatilag ugyanaz, mint ha a grafikus felületen vinnénk be a paramétereket, szóval pontosan ugyanazokat kell megadni, és ugyanúgy fogja a feladatot generálni belőle. Ez például akkor lehet hasznos, ha ugyanazokkal a konkrét paraméterekkel szeretnénk többször egymás után lefuttatni a generálást, és akkor nem

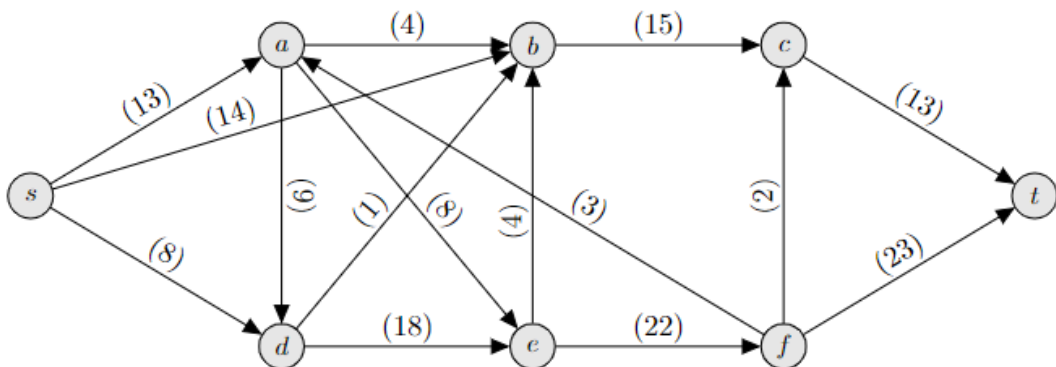
kell mindig újra a grafikus felületen kiválasztani a számunkra megfelelő tulajdonságokat, hanem elég, ha csak azt választjuk ki, hogy fájlból szeretnénk indítani a hálózati folyam feladat generálását.

Ebben az esetben a bemeneti fájl a következőképp néz ki. Első sora egy 1-es, ez jelzi azt, hogy ezzel a verzióval, vagyis egy random gráffal szeretnénk generálni a feladatot. Ellenkező esetben egy 0 szerepelne az 1 helyén. A második, harmadik és negyedik sorok pedig a grafikus felülethez hasonlóan sorban a sorok száma, csúcsok száma, feladat nehézsége. Ez látható az 5.6. ábrán.

1
2
8
6

5.6. ábra: Txt fájl tartalma random gráffal

Ezzel tehát egy 2 soros, 8 csúcsból álló random gráfból hozunk létre egy 6-os nehézségű feladatot. Ezzel a bemenettel generáltuk az alább látható feladatot.



5.7. ábra: Generált feladat random gráffal

Így ezzel és a grafikus felülettel tudunk az eddig leírt módon feladatot generálni, viszont van egy olyan lehetőség, hogy a 3. fejezet 3.1-től 3.4-ig található pontjait kihagyjuk, és azt elvégezzük a program helyett, így azzal generáljuk le a feladatot.

5.3.2 Hálózat generálása előre megadott gráffal és vágással

Tehát lehetőségünk van saját gráf és vágás segítségével feladatot generálni a megadott nehézséggel. Ez általában elég macerás, mert nem olyan könnyű a gráfot szövegfájlba a megadott formátumba kiírni, viszont vannak olyan esetek, amikor ez

hasznos lehet, például, ha bizonyos eseteket a hálózati folyamatok témakörében jobban szeretnénk szemléltetni, amihez egy bizonyos speciális gráf vagy vágás könnyebben lehetővé tesz számunkra.

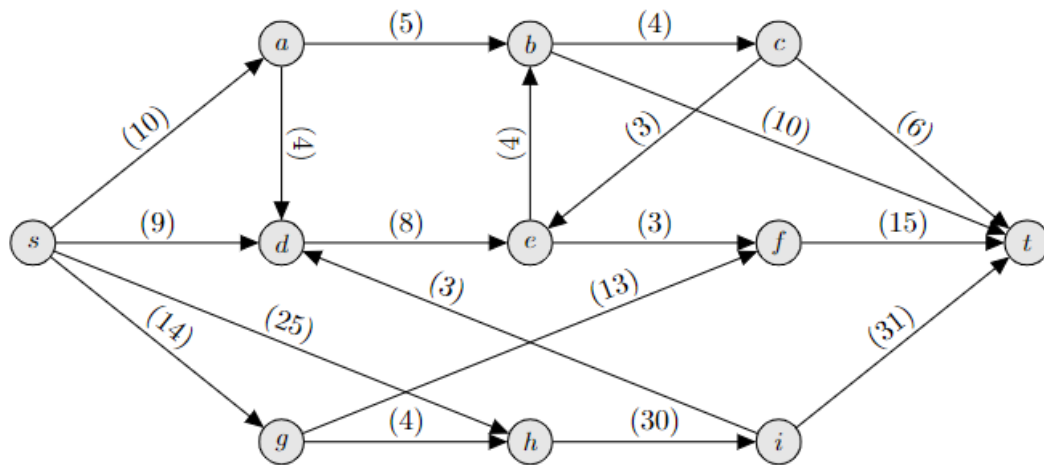
Ha saját gráffal és vágással szeretnénk dolgozni, azt a következőképpen vihetjük be a szövegfájlba keresztül. Először is az első sorába egy 0-t írunk, ami jelzi azt, hogy nem random gráffal és vágással szeretnénk dolgozni. Ezután jönnek a szokásos paraméterek, mint a sorok száma, csúcsok száma, valamint a feladat nehézsége. Ezután az élek száma következik, majd pedig az, hogy hány csúcsból áll a vágás. Ha ezek megvannak, akkor utána kezdhetjük el felsorolni az éleket. Egy él két csúcsát szóközzel elválasztva kell beleírni, előbb azt, amelyikből indul az él, majd azt amelyikbe tart. Ezt minden éllel egy-egy külön sorba írva megteesszük. Végül pedig az utolsó sorban szóközzel elválasztva felsoroljuk a vágás csúcsait.

```
0
3
11
4
19
4
s a
s d
s h
s g
a b
a d
b c
b t
c t
c e
d e
e f
e b
f t
g h
g f
h i
i t
i d
s a d e
```

5.8. ábra: Txt fájl tartalma egy megadott gráffal és vágással

A fent ábrán látható egy ilyen in.txt bemenet, ahol mi adtuk meg a gráfot és a vágást. Ez egy 3 sorból álló, 11 csúcsú gráfból a megadott élekkel rendelkező gráf lesz,

amihez az s , a , d , e csúcsokból álló vágást rendeljük, az ebből készített feladat pedig 4-es nehézségű lesz. Így ezzel a bemenettel kapjuk meg az alábbi hálózati folyam feladatot.



5.9. ábra: Generált feladat megadott gráffal és vágással

Ezek voltak tehát a program bemenetei, ami alapján a feladatot generáljuk, viszont még a továbbiakban arról is szót kell ejtsünk, hogy hogyan is kapjuk meg a feladatot eredményként, amit aztán hasznosítani tudunk.

6. fejezet

A program kimenetei

Az eddigiekben szó volt már nagyjából minden fontos lépésről, ami szükséges a feladat generálásához, így ennek elvégzésére már képes a programunk, viszont még hátra van annak a tárgyalása, hogy mi ezt az eredményt hogyan is kapjuk meg. Egyrészt a feladatot is valahogy jó lenne látnunk, illetve az ahhoz tartozó megoldást is. 3 módon kapjuk meg az eredményt, viszont ezek közül most nem kell választanunk, mindhárom automatikusan előállítja a program, az első kettőnél megoldást is kapunk hozzá, a harmadiknál viszont nem. Ha olyan hálózatunk van a feladatban, amelynek valamelyik élének kapacitása 0, akkor azt nem is vesszük figyelembe élként, és bele sem tesszük egyik kimenetbe sem.

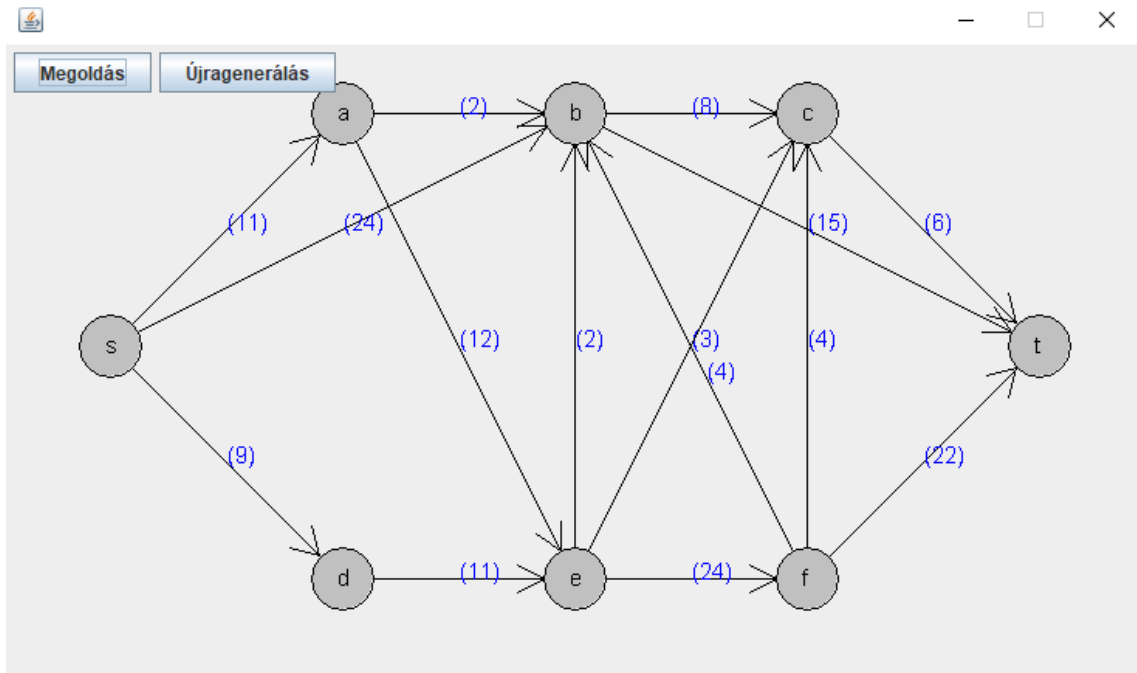
6.1 Grafikus megjelenítés

Ha a programunk legenerálta a feladatot, akkor rögtön megkapjuk az eredményt grafikusán, az eddigiekben használt Java Swing segítségével. Ezek alapvetően csak arra valóak, hogy ellenőrizzük, hogy számunkra megfelelő feladatot kaptunk-e és ezt az eredményt nem valószínű, hogy utána mondjuk egy zárthelyi készítésben felhasználnánk.

6.1.1 Feladat

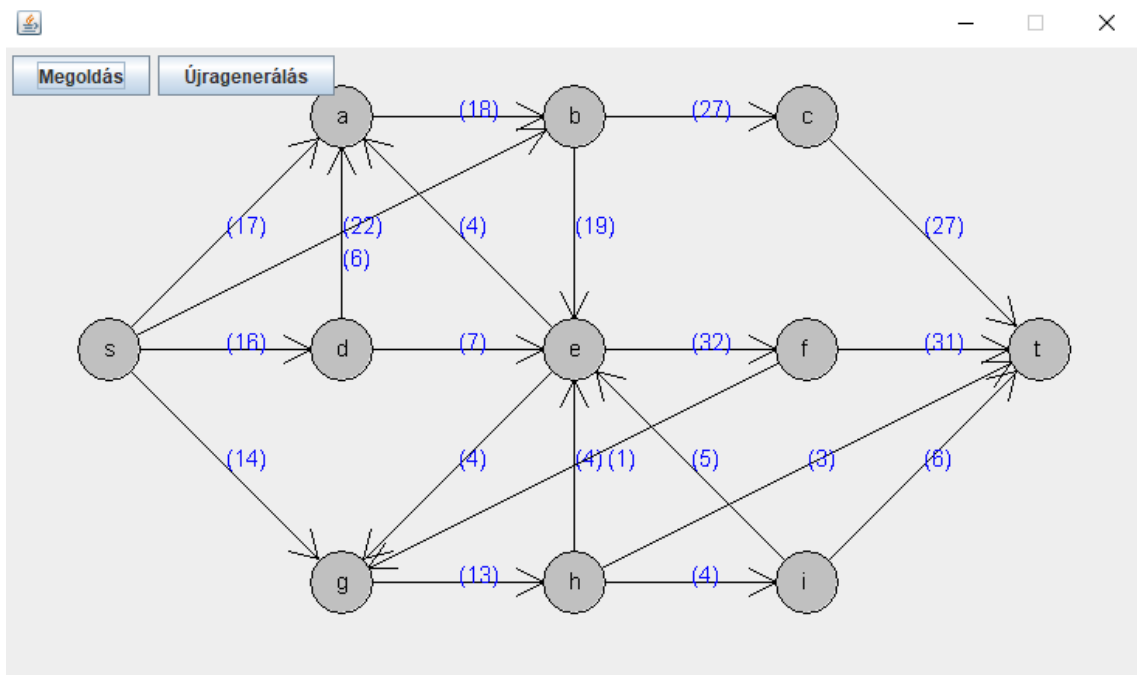
Ha végez a generálással a program, akkor kirajzolja a Graphics osztály függvényei segítségével a kapott hálózatot. A megjelenítés egyszerű, a csúcsokat egy szürke kör reprezentálja, benne a csúcs nevével, Az élek pedig egyszerű vonalakból álló fekete nyilak, amelyeken az él közepén kék színnél, zárójelbe téve szerepelnek az élekhez tartozó kapacitások. A program figyel arra, hogy két kapacitást ne próbáljon ugyanarra a helyre írni, és ezzel ne legyen átláthatatlan az egész, így, ha valahova írtunk már egy kapacitást, akkor egy másikat eltolunk valamelyik irányba, hogy azt is látható módon tudja felírni. Ez mindig sikerül, azonban néha nem teljesen látható az eltolás után, hogy a kapacitás melyik élhez is tartozik, de ehhez segítségünkre lesz a másik kétfajta kimenet, illetve említettük, hogy a zárthelyibe például nem ez lesz beletéve, szóval nem olyan nagy probléma, ha ez a megjelenítés nem olyan szép.

Alább látható 3 generált feladat, ezeken jól látszik, hogy milyen is a grafikus megjelenítésünk.

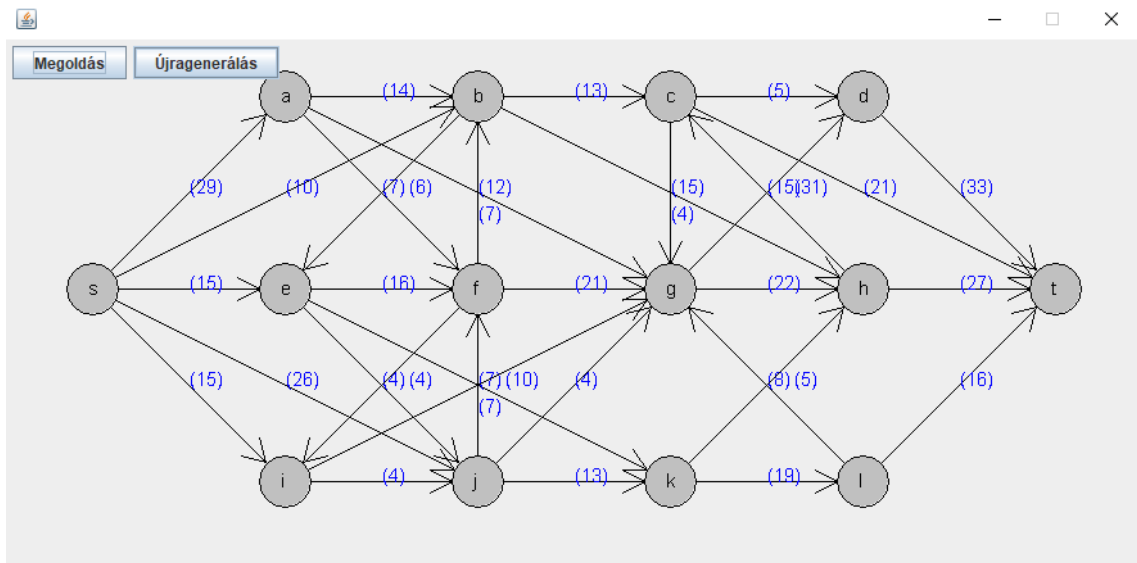


6.1. ábra: 2×3-as feladat grafikus megjelenítése

A fenti 2×3-as gráfból alkotott feladat még teljesen átlátható, mivel hosszabbak az élek, így jobban látjuk, hogy melyik élhez melyik kapacitás tartozik. Jól látszik például, hogy az fb élen a kapacitás f felé el lett tolva, mert egyébként egy pontra tette volna az ec él kapacitásával.



6.2. ábra: 3×3-as feladat grafikus megjelenítése

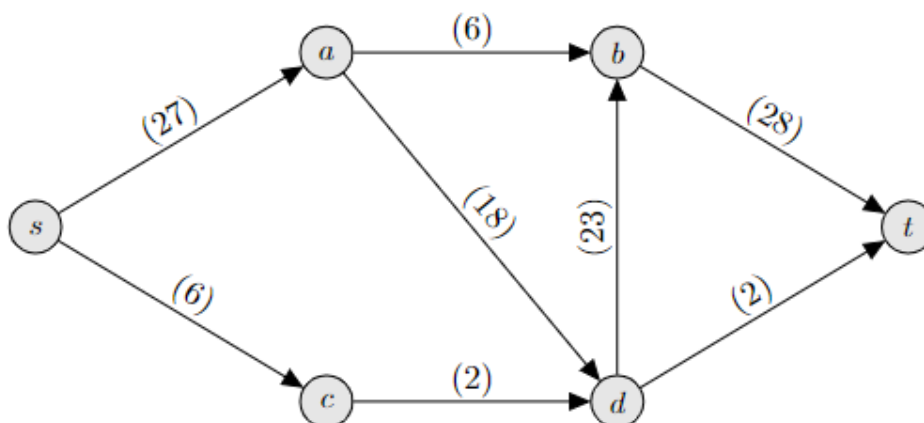


6.3. ábra: 3×4-es feladat grafikus megjelenítése

A fenti ábrán már látható, hogy ha sok csúcsból áll a gráf, akkor rövidebbek lesznek az élek, így a rájuk írt kapacitások sokka átláthatatlanabbak, néhol nehéz megállapítani, hogy melyik élhez melyik kapacitás tartozik.

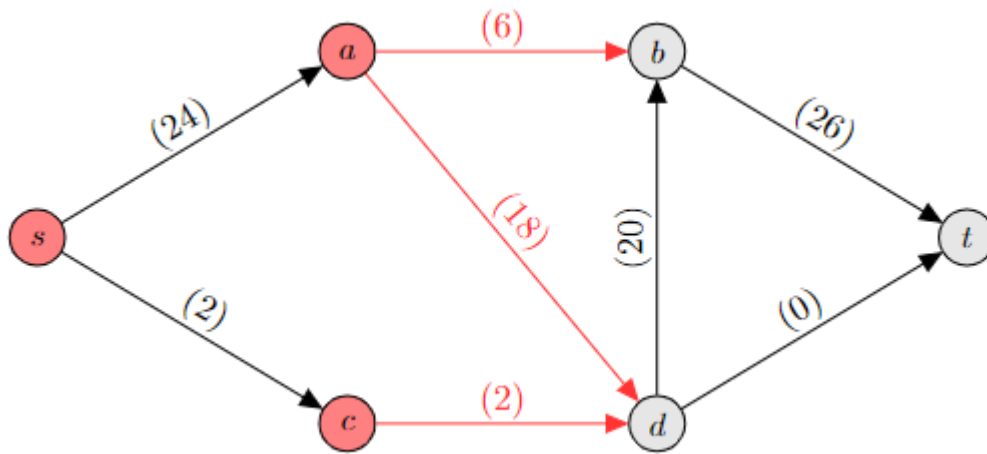
6.1.2 Megoldás

Pont ugyanezzel a módszerrel grafikusan meg tudjuk jeleníteni a megoldást is, vagyis a minimális vágást, és a maximális folyamot. Korábban már írtunk róla, de itt is fontos megemlíteni, hogy a maximális folyam nem csak egyféle lehet. Például vegyük az alábbi hálózatot:

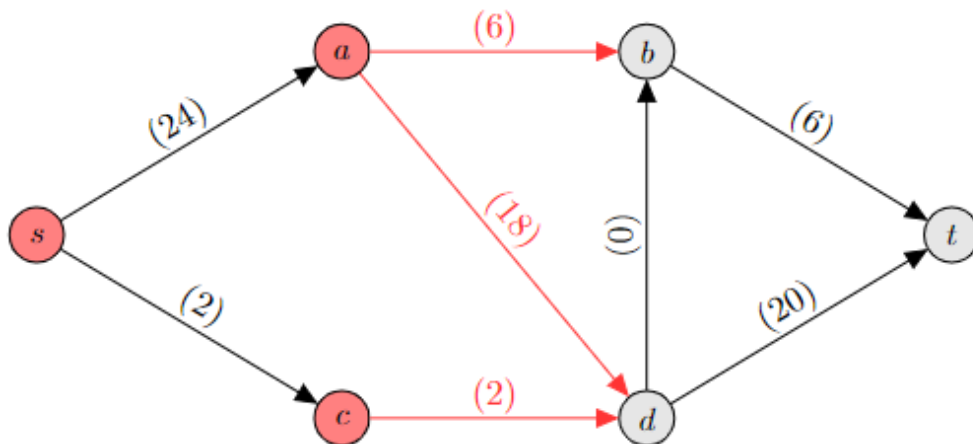


6.4. ábra: Egyszerű hálózati folyam feladat

Ehhez a következő két maximális folyam is tartozhat megoldásként:



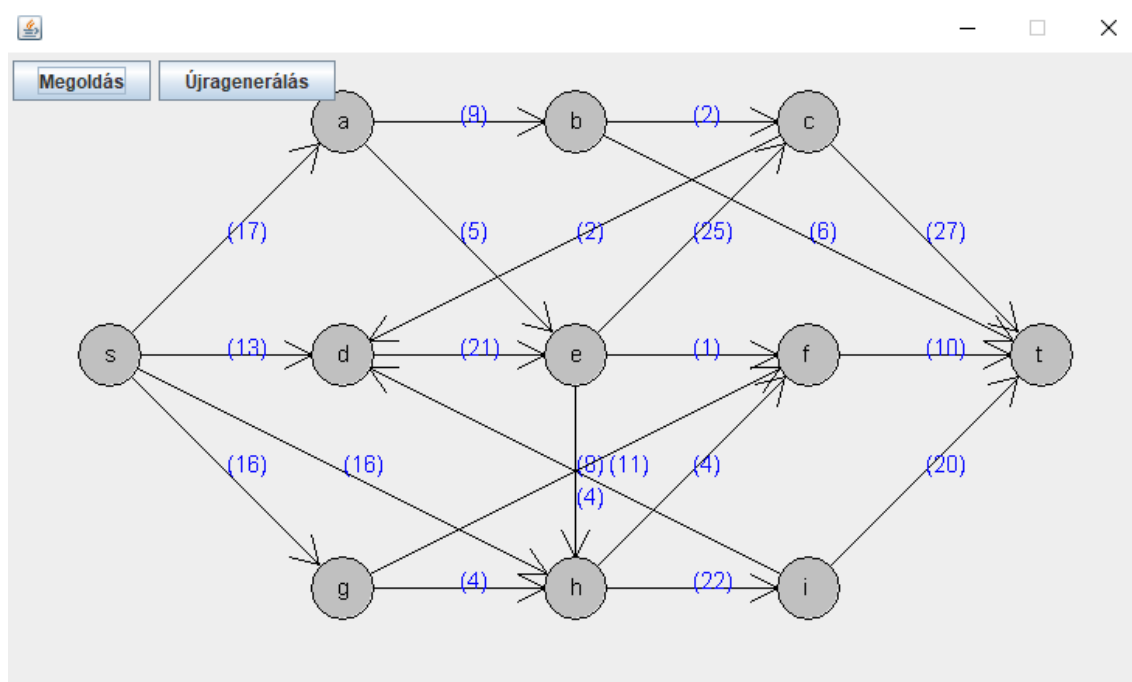
6.5. ábra: Maximális folyam megoldás 1



6.6. ábra: Maximális folyam megoldás 2

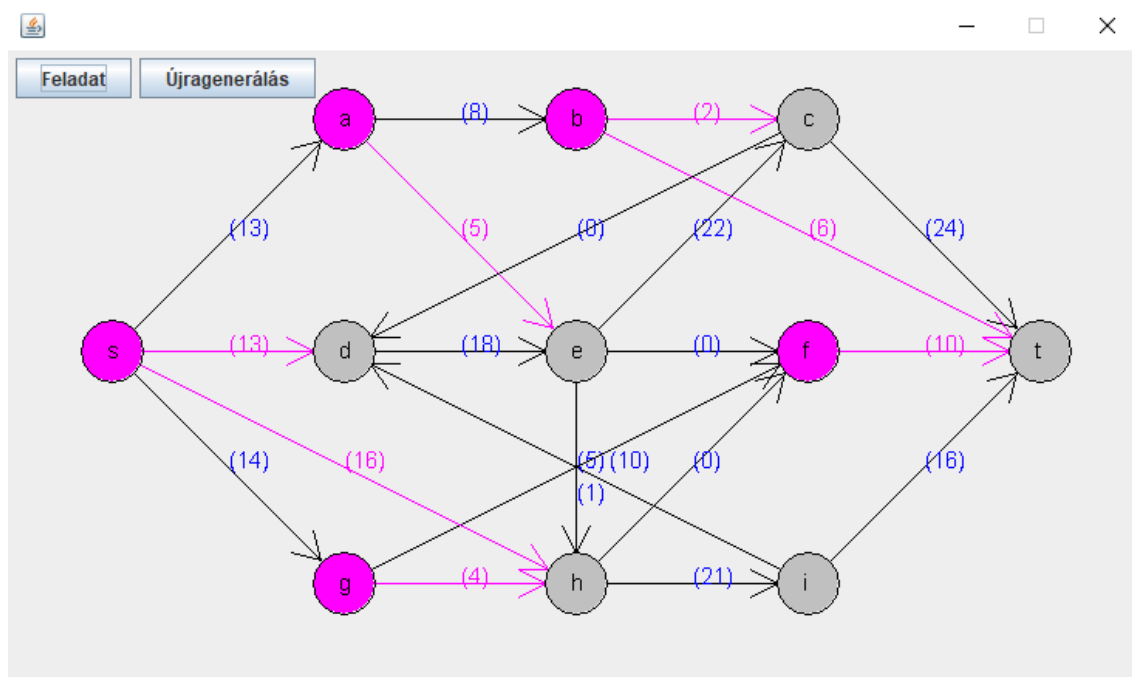
Látható, hogy a vágás ugyanaz lesz, mivel a hálózatot úgy generáljuk, hogy az csak egy féle lehet, azonban azt nem tudjuk kizárni, hogy a maximális folyamból legyen több is, így például erre találhat a hallgató egy olyat is, ami nem a mi általunk talált megoldás, ezt bele kell kalkulálnunk. Azonban ahhoz, hogy például a megoldókulcsba bele tudjunk rakni egy maximális folyamot, ahhoz ez tökéletes. Ez a maximális folyam nem más lesz, mint ami a hálózat generálása közben a nem vágáséleknek a növelése előtti folyam. Ezt meg tudjuk jeleníteni grafikusán is.

Például vegyük az alábbi grafikusan megjelenített feladatot:



6.7. ábra: Feladat megjelenítése grafikusan

Ekkor a Megoldás gombra kattinthatunk, és szintén grafikusan megjelenítjük ezzel a következő megoldást:



6.8. ábra: Megoldás megjelenítése grafikusan

Ekkor magenta színnel jelennel meg a vágás csúcsai és élei, valamint az éleken most zárójelben nem a kapacitás, hanem az érték szerepel a maximális folyamban. Ekkor a Feladat gombra kattintva ismét megjeleníthető maga a feladat.

6.1.3 Újragenerálás

A grafikus megjelenítés közben, ha nem egy számunkra megfelelő feladatot kapunk, akkor újragenerálhatjuk azt, természetesen ugyanazokkal a bevitt paraméterekkel, mint előtte. Ekkor csak rá kell nyomni az Újragenerálás gombra, és a programunk ismételtén előállít egy feladatot. Ezt bármennyiszer megtehetjük, ameddig egy számunkra megfelelő feladatot nem kapunk.

6.2 Txt fájl

A kapott feladatot eredményként kiírja a program egy out.txt nevű szöveges fájlba a következőképpen: először „kirajzolja” a gráf csúcsait, hogy milyen sorrendben, hogyan szerepelnek, majd kiírja az éleket egyesével a kapott kapacitásokkal zárójelben, valamint minden élhez egy értéket, ami hálózat általunk talált maximális folyamához tartozik. Az élék után még szerepel a minimális vágás, és ennek értéke is. Az alább látható ábrán megtekinthetjük, hogy hogyan is fog kinézni ez a kimenet. Erre alapvetően nincs túl nagy szükségünk, viszont arra mindenképp jó lehet, hogy a feladat megoldását elmentsük vele, mivel a harmadik, általunk általában használt kimenethez nem tartozik megoldás.

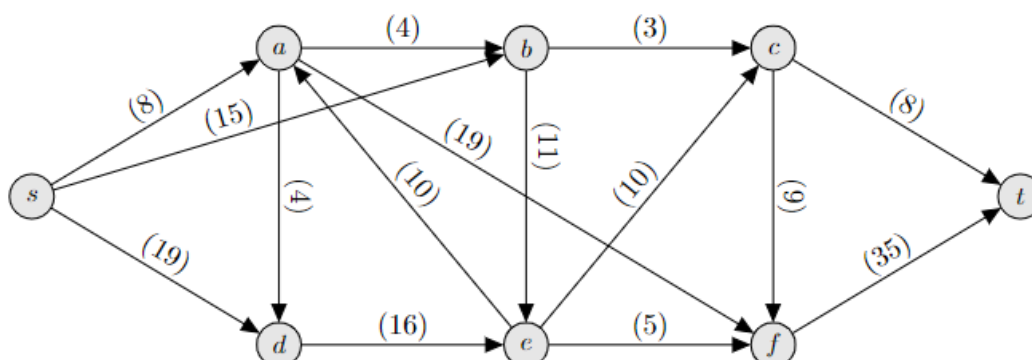
```
a b c
s    t
d e f
```

```
a b (4) 0
b c (3) 3
s a (8) 8
c t (8) 5
s b (15) 13
a d (4) 0
e a (10) 10
a f (19) 18
b e (11) 10
e c (10) 10
c f (9) 8
d e (16) 15
e f (5) 5
s d (19) 15
f t (35) 31
```

```
Minimális vágás: s b d e
Minimális vágás értéke: 36
```

6.9. ábra: Txt fájl kimenet

A fenti out.txt fájlban találhatóak a következő feladatot reprezentálják:



6.10. ábra: Txt fájl által leírt feladat

6.3 LaTeX fájl

A Bevezetés a számításelméletbe és a Számítástudomány alapjai tárgyakban a gyakorlati és zárthelyi feladatsorok LaTeX formátumban készülnek el, így ahhoz, hogy hasznosítani tudjuk a generált feladatot, jó lenne, ha kimenetként kapnánk egy LaTeX fájlt, amit simán csak bemásolhatnánk a feladatsor készítésekor. Erre szolgál a program harmadik kimeneteként kapott latex.tex fájl.

6.3.1 Program által generált LaTeX fájl

A generált fájlban szürke körökben helyezkednek el a csúcsoknak a nevei, a csúcsokat pedig fekete nyilak kötik össze. Az élekre írt kapacitások az élek közepén helyezkednek el zárójelben, de picivel az él fölött, hogy jól látható legyen, illetve az éllel együtt a kapacitás szövege is meg van döntve. A csúcsok elrendezése a következő: Az x koordináta az pontosan annyi, ahányadik oszlopban helyezkedik el a csúcs a gráfban, ha az s csúcs oszlopát 0-nak vesszük. Az y koordináta pedig ahányadik sorban található szorozva 0.8-cal páratlan számú sor esetén és 0.6-tal páros számú sor esetén, azonban az s és t sorát is külön sornak számítjuk. A 0.8-cal és 0.6-tal való szorzásoknál néha valamiért belekerül egy kis kerekítési hiba, de ezzel nem kell foglalkoznunk, mivel jelentéktelen mértékű.

Az alábbiakban látható egy kapott forráskód, amit a latex.tex fájl tartalmaz egy 3×3-as gráfból létrehozott feladatra.

```

\documentclass{article}

\usepackage{tikz}
\usetikzlibrary{arrows}
\tikzset{>=triangle 45}

\begin{document}

\begin{center}

\tikz[->, csucs/.style={draw, fill=black!10, circle, minimum size={0.6cm}, inner
sep=0cm, align=center, scale=0.9},
scale=3]
{
\node [csucs] (a) at (1,2.4000000000000004) {$a$};
\node [csucs] (b) at (2,2.4000000000000004) {$b$};
\node [csucs] (c) at (3,2.4000000000000004) {$c$};
\node [csucs] (s) at (0,1.6) {$s$};
\node [csucs] (d) at (1,1.6) {$d$};
\node [csucs] (e) at (2,1.6) {$e$};
\node [csucs] (f) at (3,1.6) {$f$};
\node [csucs] (t) at (4,1.6) {$t$};
\node [csucs] (g) at (1,0.8) {$g$};
\node [csucs] (h) at (2,0.8) {$h$};
\node [csucs] (i) at (3,0.8) {$i$};

\path

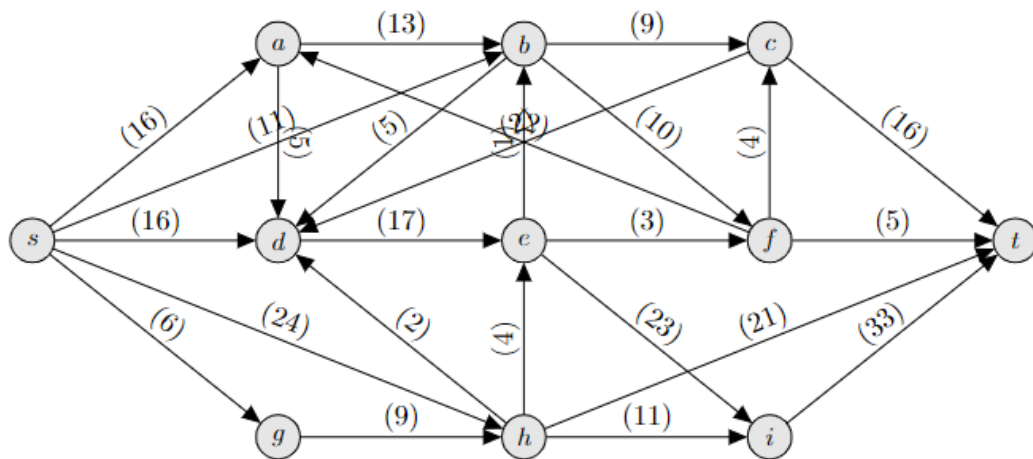
(a) edge node [above=-2pt, pos=0.5, sloped] {(13)} (b)
(b) edge node [above=-2pt, pos=0.5, sloped] {(9)} (c)
(s) edge node [above=-2pt, pos=0.5, sloped] {(16)} (a)
(c) edge node [above=-2pt, pos=0.5, sloped] {(16)} (t)
(s) edge node [above=-2pt, pos=0.5, sloped] {(11)} (b)
(a) edge node [above=-2pt, pos=0.5, sloped] {(5)} (d)
(f) edge node [above=-2pt, pos=0.5, sloped] {(2)} (a)
(b) edge node [above=-2pt, pos=0.5, sloped] {(5)} (d)
(e) edge node [above=-2pt, pos=0.5, sloped] {(1)} (b)
(b) edge node [above=-2pt, pos=0.5, sloped] {(10)} (f)
(c) edge node [above=-2pt, pos=0.5, sloped] {(2)} (d)
(f) edge node [above=-2pt, pos=0.5, sloped] {(4)} (c)
(s) edge node [above=-2pt, pos=0.5, sloped] {(16)} (d)
(d) edge node [above=-2pt, pos=0.5, sloped] {(17)} (e)
(e) edge node [above=-2pt, pos=0.5, sloped] {(3)} (f)
(f) edge node [above=-2pt, pos=0.5, sloped] {(5)} (t)
(h) edge node [above=-2pt, pos=0.5, sloped] {(2)} (d)
(h) edge node [above=-2pt, pos=0.5, sloped] {(4)} (e)
(e) edge node [above=-2pt, pos=0.5, sloped] {(23)} (i)
(g) edge node [above=-2pt, pos=0.5, sloped] {(9)} (h)
(h) edge node [above=-2pt, pos=0.5, sloped] {(11)} (i)
(s) edge node [above=-2pt, pos=0.5, sloped] {(6)} (g)
(i) edge node [above=-2pt, pos=0.5, sloped] {(33)} (t)
(s) edge node [above=-2pt, pos=0.5, sloped] {(24)} (h)
(h) edge node [above=-2pt, pos=0.5, sloped] {(21)} (t)

;
}
\end{center}

\end{document}

```

Látjuk, hogy szerepel benne az említett kerekítési hiba, de nem kell foglalkoznunk vele. Ezt legenerálva megkapjuk az alább látható feladatot.



6.11. ábra: LaTeX fájlal létrehozott hálózat képe

Látjuk, hogy ez azonban nem tökéletes, mivel néhány kapacitást egy pontba rakott, így kitakarják egymást, emiatt jelenlegi formájában még alkalmatlan arra, hogy a feladatsorba belerakjuk.

6.3.2 Javított LaTeX fájl

Szükség van tehát a forráskód kijavítására, azonban ezt egy bonyolultabb feladat lenne a programkódban megtenni, így ennek az elvégzését a feladatsor elkészítőjére bizzuk. Az alábbi forrásfájlban már a kézzel javított verzió látható. Itt az sb él esetében a kapacitás helyét 0.5-ről 0.35-re állítottuk, ad él esetében 0.7-re, fa élnél 0.35-re, eb élen 0.25-re és cd élen pedig 0.7-re.

```
\documentclass{article}

\usepackage{tikz}
\usetikzlibrary{arrows}
\tikzset{>=triangle 45}

\begin{document}

\begin{center}

\tikz[->, csucs/.style={draw, fill=black!10, circle, minimum size={0.6cm}, inner
sep=0cm, align=center, scale=0.9},
scale=3]
{
  \node [csucs] (a) at (1,2.4000000000000004) {$a$};
  \node [csucs] (b) at (2,2.4000000000000004) {$b$};
  \node [csucs] (c) at (3,2.4000000000000004) {$c$};
  \node [csucs] (s) at (0,1.6) {$s$};
  \node [csucs] (d) at (1,1.6) {$d$};
  \node [csucs] (e) at (2,1.6) {$e$};
  \node [csucs] (f) at (3,1.6) {$f$};
  \node [csucs] (t) at (4,1.6) {$t$};
}
```

```

\node [csucs] (g) at (1,0.8) {$g$};
\node [csucs] (h) at (2,0.8) {$h$};
\node [csucs] (i) at (3,0.8) {$i$};

\path

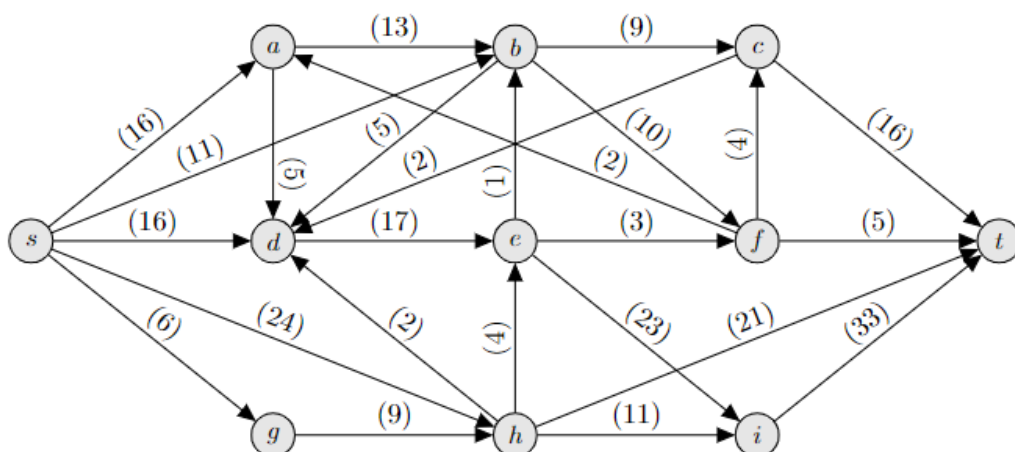
(a) edge node [above=-2pt, pos=0.5, sloped] {(13)} (b)
(b) edge node [above=-2pt, pos=0.5, sloped] {(9)} (c)
(s) edge node [above=-2pt, pos=0.5, sloped] {(16)} (a)
(c) edge node [above=-2pt, pos=0.5, sloped] {(16)} (t)
(s) edge node [above=-2pt, pos=0.35, sloped] {(11)} (b)
(a) edge node [above=-2pt, pos=0.7, sloped] {(5)} (d)
(f) edge node [above=-2pt, pos=0.3, sloped] {(2)} (a)
(b) edge node [above=-2pt, pos=0.5, sloped] {(5)} (d)
(e) edge node [above=-2pt, pos=0.25, sloped] {(1)} (b)
(b) edge node [above=-2pt, pos=0.5, sloped] {(10)} (f)
(c) edge node [above=-2pt, pos=0.7, sloped] {(2)} (d)
(f) edge node [above=-2pt, pos=0.5, sloped] {(4)} (c)
(s) edge node [above=-2pt, pos=0.5, sloped] {(16)} (d)
(d) edge node [above=-2pt, pos=0.5, sloped] {(17)} (e)
(e) edge node [above=-2pt, pos=0.5, sloped] {(3)} (f)
(f) edge node [above=-2pt, pos=0.5, sloped] {(5)} (t)
(h) edge node [above=-2pt, pos=0.5, sloped] {(2)} (d)
(h) edge node [above=-2pt, pos=0.5, sloped] {(4)} (e)
(e) edge node [above=-2pt, pos=0.5, sloped] {(23)} (i)
(g) edge node [above=-2pt, pos=0.5, sloped] {(9)} (h)
(h) edge node [above=-2pt, pos=0.5, sloped] {(11)} (i)
(s) edge node [above=-2pt, pos=0.5, sloped] {(6)} (g)
(i) edge node [above=-2pt, pos=0.5, sloped] {(33)} (t)
(s) edge node [above=-2pt, pos=0.5, sloped] {(24)} (h)
(h) edge node [above=-2pt, pos=0.5, sloped] {(21)} (t)

;
\end{center}

\end{document}

```

Ezzel pedig megkaptuk az alább látható feladatot, amit késznek tekinthetünk, és a feladatsorba rakhatunk.



6.12. ábra: Javított LaTeX fájlal létrehozott hálózat képe

7. fejezet

Összegzés

Sikerült tehát egy olyan programot fejlesztenünk, amely képes hálózati folyam feladatok generálására a kiválasztott gráf paraméterekkel, és a megadott feladatnehézséggel. Ezt kisebb méretű gráfokból képzett feladatok esetén meglehetősen gyorsan teszi, de a nagyobbaknál kell azért várnunk egy kevés időt rá.

A kapott LaTeX fájlal nagyjából már rögtön fel is használható egy zárthelyi feladatsorban, csak egy keveset kell előtte változtatni rajta. Megoldást viszont csak grafikus vagy txt fájlban kapunk hozzá, így figyelni kell ezeknek a lementésére is a feladat generálásakor.

A kapott feladat általában számunkra megfelelő lesz, de azért van, hogy nem tökéletesen passzol a kívánt nehézséghez, vagy esetleg nem a legszebb, ekkor azonban könnyedén, egy gombnyomással újragenerálhatjuk a feladatot.

Munkám során maga a Java kód megírása volt a legnagyobb feladat, azon belül is az utak keresése, és a megfelelő gráf generálása. Jelentős feladat volt azonban még annak a kidolgozása is, hogy hogyan is állapítsuk meg a feladat nehézségét, miután a szempontokat megtaláltam, ami alapján érdemes számítani, azután is nagyon sokat kellett finomhangolni, hogy nagyjából megfeleljen a valóságnak a kapott nehézség.

A grafikus felület, és főleg a gráf kirajzolása lehetne szebb és jobban átlátható, ezeken a pontokon még lehet fejleszteni, illetve a programunk csak az ilyen téglalapszerű rácsos elrendezésű gráfokból képes feladatot gyártani, úgyhogy ebben az irányban is van még fejlesztési potenciál. Ezen kívül azt sem lenne rossz megoldani, hogy a kapott LaTeX fájlban ne kézzel kelljen kijavítani azt, hogy hova tegye a kapacitást, ha kettő ütközik, és így az eredményt egyből lehessen használni. Ez azonban nem olyan egyszerű feladat.

Összességében azért a program működőképes, alkalmas arra, hogy a Bevezetés a számításelméletbe és a Számítástudomány alapjai tárgyakhoz feladatokat generáljunk vele.

Irodalomjegyzék

- [1] „Maximális áramlási probléma” Wikipédia,
https://hu.wikipedia.org/wiki/Maxim%C3%A1lis_%C3%A1raml%C3%A1si_probl%C3%A9ma
- [2] Jon Kleinberg és Éva Tardos, Algrith Design, 2005.
- [3] „Dinic’s algorithm” Wikipedia, https://en.wikipedia.org/wiki/Dinic%27s_algorithm.
- [4] „The Binary Blocking Flow Algorithm” Dinamics Rutgers,
<http://dimacs.rutgers.edu/archive/Workshops/Tarjan/materials/talk-slides/goldberg.pdf>.
- [5] „Push–relabel maximum flow algorithm” Wikipedia,
https://en.wikipedia.org/wiki/Push%E2%80%93relabel_maximum_flow_algorithm.
- [6] „A Faster Deterministic Maximum Flow Algorithm”,
<http://www.cs.umd.edu/~gasarch/BLOGPAPERS/king-rao-tarjan.pdf>.
- [7] „Maximum flow - MPM algorithm” CP algorithms, <https://cp-algorithms.com/graph/mpm.html>.
- [8] „Ford–Fulkerson algorithm” Wikipedia,
https://en.wikipedia.org/wiki/Ford%E2%80%93Fulkerson_algorithm.
- [9] „Maximális folyam – minimális vágás” Wikipédia,
https://hu.wikipedia.org/wiki/Maxim%C3%A1lis_folyam_%E2%80%93_minim%C3%A1lis_v%C3%A1g%C3%A1s.