

SZAKDOLGOZAT FELADAT

**Tokovics Dávid Tamás**

Mérnökinformatikus hallgató részére

Hálózati folyam feladatok automatikus generálása

A Számítástudomány alapjai és a Bevezetés a számítástudományba tárgyak anyaga a Hálózati folyamok 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

2022. december 9.

Tartalomjegyzék

[Összefoglaló 1](#_Toc121170494)

[Abstract 2](#_Toc121170495)

[Bevezetés 3](#_Toc121170496)

[Hálózati folyam feladatok 5](#_Toc121170497)

[2.1 Hálózatok 5](#_Toc121170498)

[2.2 Folyamok 6](#_Toc121170499)

[2.3 Maximális folyam 7](#_Toc121170500)

[2.4 Minimális vágás 8](#_Toc121170501)

[Hálózatok generálása 9](#_Toc121170502)

[3.1 Csúcsok elrendezése 9](#_Toc121170503)

[3.2 Keret éleinek behúzása 11](#_Toc121170504)

[3.3 További élek hozzáadása 13](#_Toc121170505)

[3.3.1 s-ből vagy t-be mutató élek 13](#_Toc121170506)

[3.3.2 Azonos oszlopba mutató élek 14](#_Toc121170507)

[3.3.3 Eggyel előre mutató élek 14](#_Toc121170508)

[3.3.4 Kettővel előre mutató élek 15](#_Toc121170509)

[3.3.5 Eggyel vissza mutató élek 15](#_Toc121170510)

[3.4 Vágás sorsolása 19](#_Toc121170511)

[3.5 Folyam létrehozása 19](#_Toc121170512)

[3.5.1 Utak keresése s-ből t-be 20](#_Toc121170513)

[3.5.2 Utak keresése visszaéllel 21](#_Toc121170514)

[3.5.3 Keletkezett folyamok 23](#_Toc121170515)

[3.6 Folyam átalakítása hálózattá 23](#_Toc121170516)

[3.6.1 Élek növelése a vágás éleinek kivételével 23](#_Toc121170517)

[3.6.2 Keletkezett hálózatok 24](#_Toc121170518)

[Feladat nehézsége 26](#_Toc121170519)

[4.1 Élekkel kapcsolatos nehézségek 26](#_Toc121170520)

[4.1.1 Élek száma 26](#_Toc121170521)

[4.1.2 Nullélek száma 26](#_Toc121170522)

[4.1.3 Hosszú élek száma 27](#_Toc121170523)

[4.1.4 Visszafele mutató élek száma 28](#_Toc121170524)

[4.2 Csúcsok száma 28](#_Toc121170525)

[4.3 Vágáserősség 28](#_Toc121170526)

[4.4 Szórás 29](#_Toc121170527)

[4.5 Teljes képlet 29](#_Toc121170528)

[A program bemenetei 31](#_Toc121170529)

[5.1 Grafikus felület kezdőoldala 31](#_Toc121170530)

[5.2 Hálózat generálása paraméterek megadásával 31](#_Toc121170531)

[5.3 Hálózat generálása fájlból 32](#_Toc121170532)

[5.3.1 Hálózat generálása random gráffal 33](#_Toc121170533)

[5.3.2 Hálózat generálása előre megadott gráffal és vágással 34](#_Toc121170534)

[A program kimenetei 36](#_Toc121170535)

[6.1 Grafikus megjelenítés 36](#_Toc121170536)

[6.1.1 Feladat 36](#_Toc121170537)

[6.1.2 Megoldás 37](#_Toc121170538)

[6.1.3 Újragenerálás 40](#_Toc121170539)

[6.2 Txt fájl 40](#_Toc121170540)

[6.3 LaTeX fájl 41](#_Toc121170541)

[6.3.1 Program által generált LaTeX fájl 41](#_Toc121170542)

[6.3.2 Javított LaTeX fájl 43](#_Toc121170543)

[Összegzés 46](#_Toc121170544)

[Irodalomjegyzék 47](#_Toc121170545)

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.

A képen szöveg látható

Automatikusan generált leírásKelt: Budapest, 2022. 12. 05.

...…………………………………………….

Tokovics Dávid

# Összefoglaló

A Bevezetés a számításelméletbe és a Számítástudomány alapjai tárgyakban a Hálózati folyamok 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 folyam 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 feladatnehézséget 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 mindhármat 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 folyamok 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 folyamok. 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 oktatónak 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, folyamok, 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**

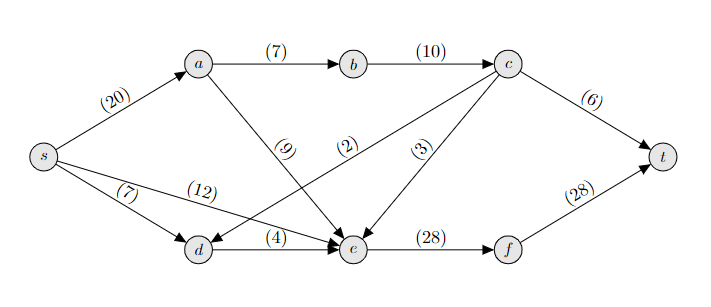
# Hálózati folyam feladatok

Hálózati folyam feladatok alatt gyakorlatilag T. E. Harris és F. S. Ross 1954-ben megfogalmazott maximális áramlási problémával 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 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.

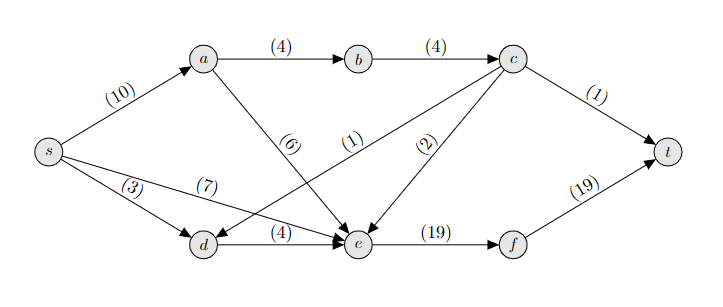


**2.1. ábra: Hálózat**

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 rendelük, 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. Vagyis amilyen értékek bemennek egy csúcsba, azok ki is jönnek. Egy ilyen a 2.1. ábraán látható hálózathoz tartozó folyamot mutat be a következő ábra.

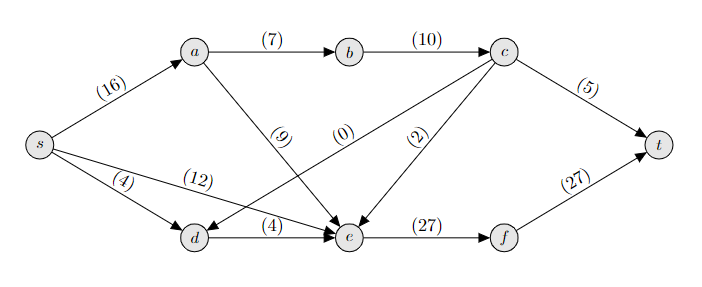


**2.2. ábra: Folyam**

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.

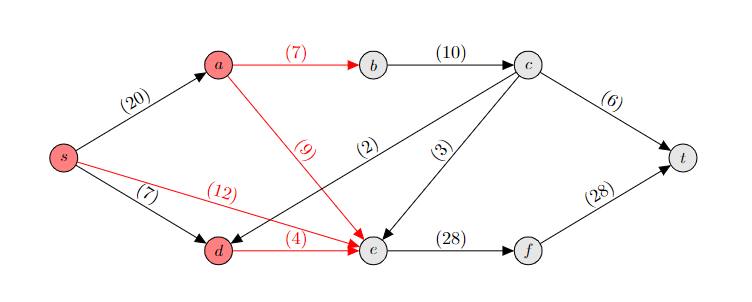
A maximális folyam megtalálására vannak különböző módszerek, például Dinic’s algoritmusa, a bináris blokkoló áramlási algoritmus, a push-relabel algoritmus különböző variációi, King, Rao és Tarjan algoritmusa vagy az MPM algoritmus. A Számítástudomány alapjai és a Bevezetés a számításelméletbe tárgyakban a Ford és Fulkerson által megalkotott javítóutas algoritmusról 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 csúcsaiból induló olyan élek értékeinek 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 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

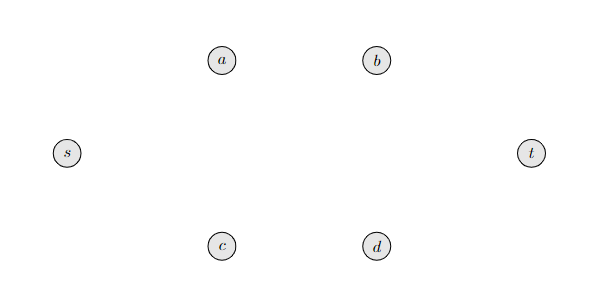
# Hálózatok generálása

A feladatokat generáló program csak a megadott paraméterek alapján gyárt hálózatokat, amelyeket aztán kiértékel nehézség szerint, és ezalapján 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 feladatnehézséghez, 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

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 szélen 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) ez látható a 3.1. ábraán.



**3.1. ábra: 2×2-es csúcselrendezés**

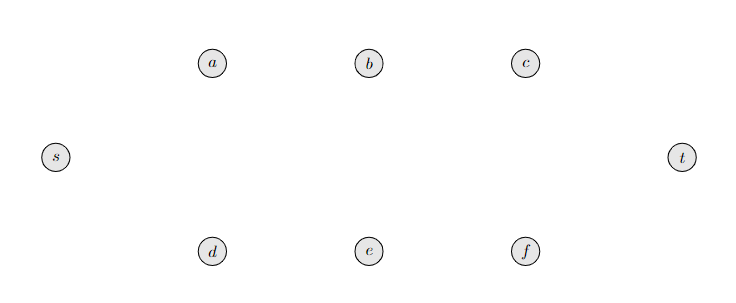
A legnagyobb gráf pedig, amit így létrehozhatunk az 5 soros, és 27 csúcsa van (5×5-ös), ez a 3.2. ábraán látható.



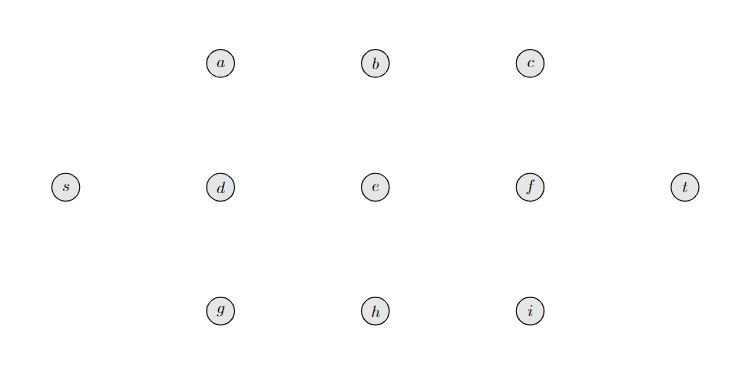
**3.2. ábra: 5×5-ös csúcselrendezés**

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 többit.

A fenti két csúcselrendezést ritkán fogjuk használni, a két leggyakoribb gráftípus 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 húzunk 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 csúcselrendezésből az alábbi két gráf, amit a 3.5. ábra és 3.6. ábra tartalmaz, ezekhez fogunk még behúzni további éleket.

A képen kard, lámpa látható

Automatikusan generált leírás

**3.5. ábra: Keret a 2×3-as csúcselrendezésben**

A képen kard, különböző, lámpa látható

Automatikusan generált leírás

**3.6. ábra: Keret a 3×3-as csúcselrendezésben**

## 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 mértében randomizált módon fog működni, azalapján, 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 élnek 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.

A képen kard, modellt állás, különböző látható

Automatikusan generált leírás

**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 eggyel alatta vagy felette lévő sorban, vele egy oszlopban található csúcsba húzott élek.

A képen kard látható

Automatikusan generált leírás

**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 Eggyel előre mutató élek

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

A képen kard, különböző látható

Automatikusan generált leírás

**3.9. ábra: Eggyel 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.

A képen kard látható

Automatikusan generált leírás

**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.

A képen kard, különböző látható

Automatikusan generált leírás

**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.

A képen kard látható

Automatikusan generált leírás

**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 előrefelé 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.

A képen kard látható

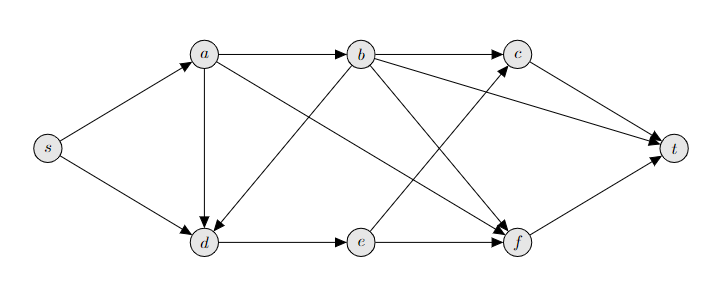
Automatikusan generált leírás

**3.13. ábra: Nem használt éltípusok**

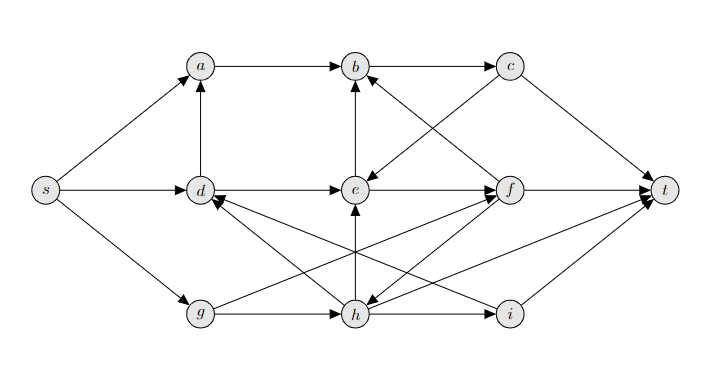
**3.3.8 Az élek hozzáadásával generált gráfok**

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 csúcselrendezéshez 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 előrefelé mutató élből egy sem került bele, az eggyel visszafele 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 folyamot, 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, méghozzá 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. Ezektől 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.

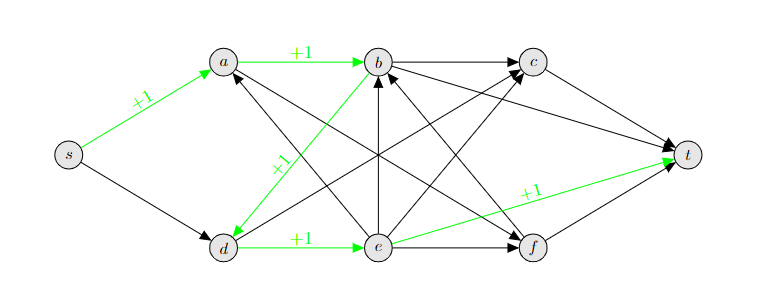
### 3.5.1 Utak keresése s-ből t-be

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 élét 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 kifele 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 figyelnünk 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 figyelnünk 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**

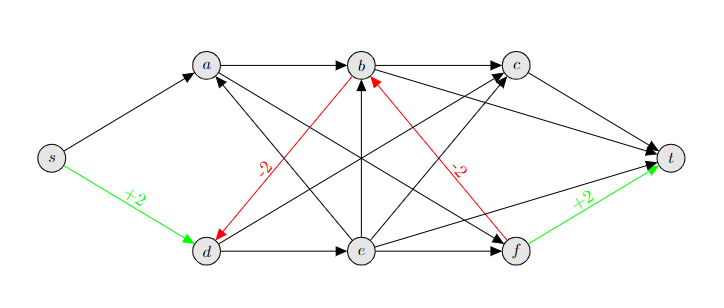
### 3.5.2 Utak keresése visszaéllel

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 előrefelé megyünk az élen, akkor az ugyanúgy nem lehet egy olyan él, aminek kezdőcsúcsa nem a vágás része, a végpontjában lévő csúcs viszont igen. Ennek miértje az előzőhöz hasonlóan ugyanúgy kifejthető, pontosan ugyanazon okok miatt rontaná el azt, hogy a kiválasztott vágás biztosan minimális vágás maradjon. Viszont, ha visszafele 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 kifele 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 élét, 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.

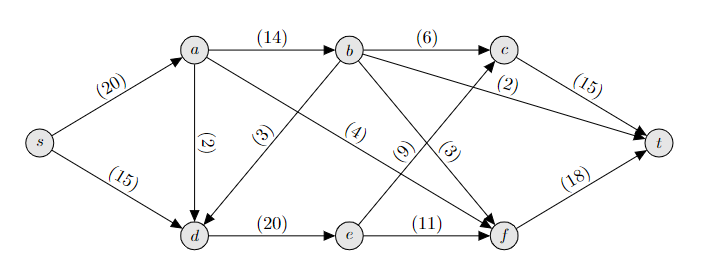
Egy ilyen útkeresés található a következő ábrán, ahol használtunk visszafele is éleket, azok értékét csökkentve. Most nem 1-gyel hanem 2-vel növeltük, illetve csökkentettük az élek értékét.



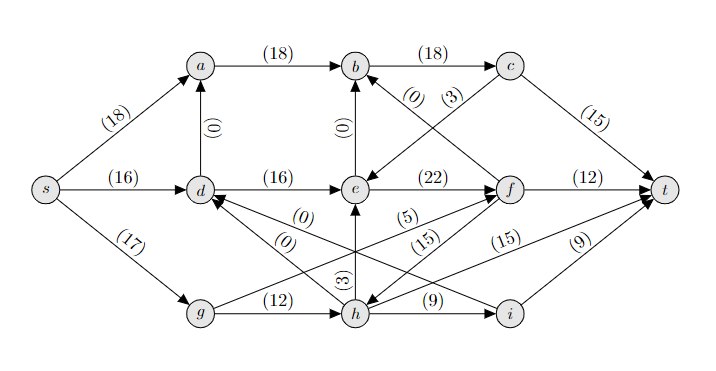
**3.17. ábra: visszaélek élkapacitásának csökkentése az útban**

### 3.5.3 Keletkezett folyamok

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 legkönnyeben 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 folyamot hálózattá alakítunk, amely hálózat minimális vágása az s, d, e csúcsokból fog állni.

A képen síelés, színes, zászló, lejtő látható

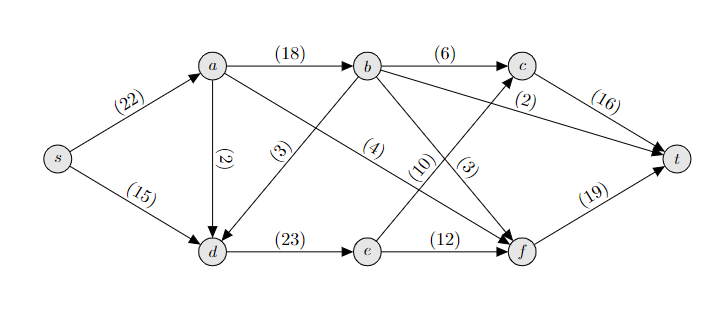
Automatikusan generált leírás

**3.20. ábra: Nem vágásélek kapacitásának növelése**

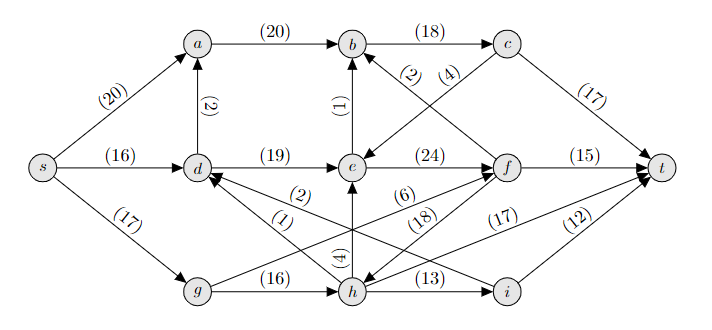
### 3.6.2 Keletkezett hálózatok

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 folyamokból 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**

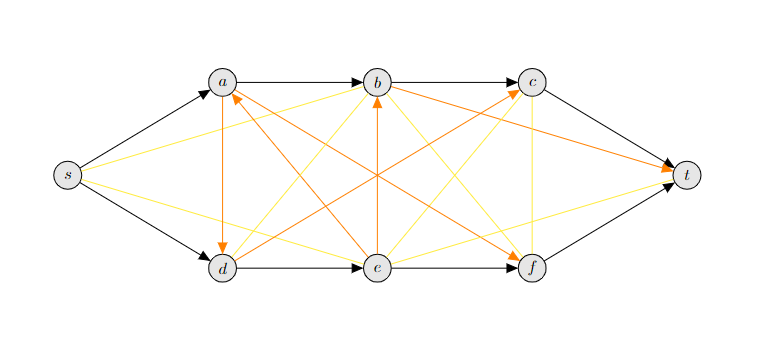
aca

## 4.1 Élekkel kapcsolatos nehézségek

adcaw

### 4.1.1 Élek száma

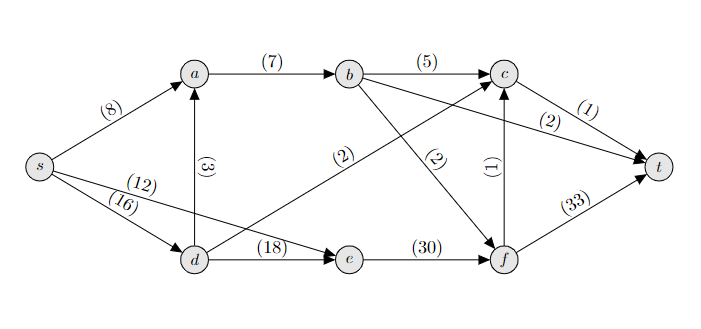
adasd



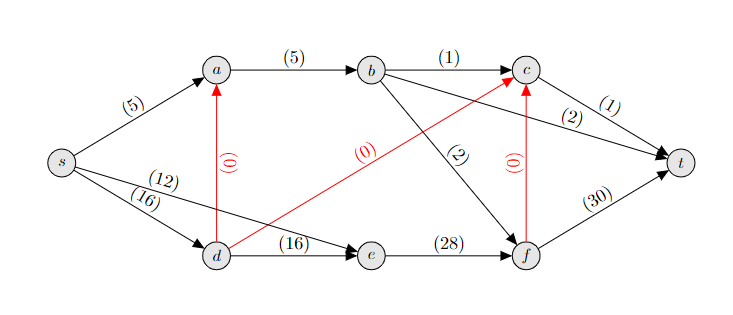
**4.1. ábra:** **Élek aránya a maximumhoz képest**

### 4.1.2 Nullélek száma

asd



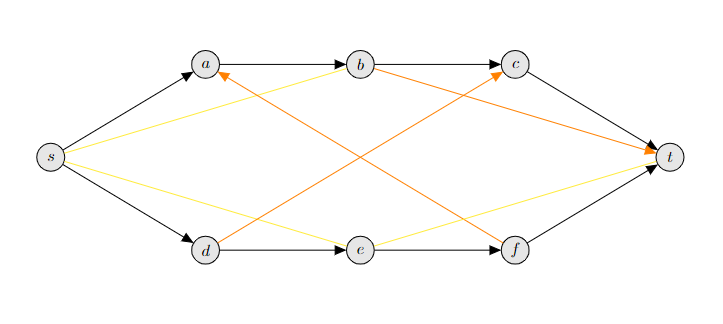
**4.2. ábra: Maximális folyam feladat**



**4.3. ábra: Nullélek a maximális folyamban**

### 4.1.3 Hosszú élek száma

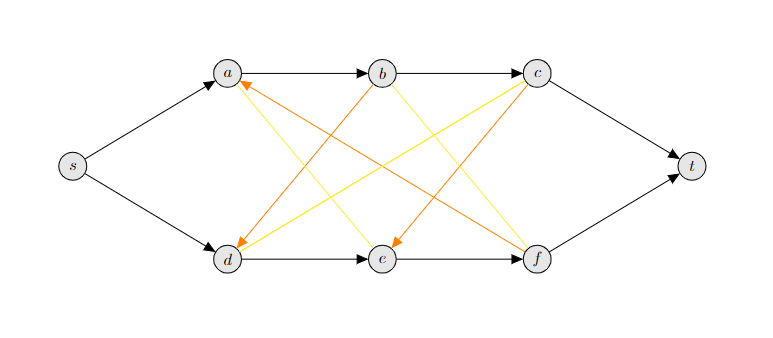
asd



**4.4. ábra: Hosszú élek aránya a maximumhoz képest**

### 4.1.4 Visszafele mutató élek száma

asd



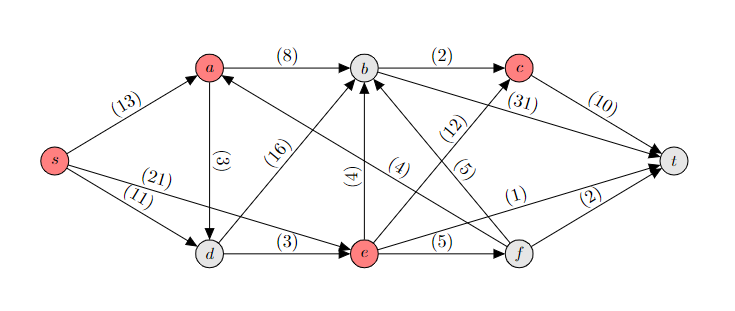
**4.5. ábra: Visszafele mutató élek aránya a maximumhoz képest**

## 4.2 Csúcsok száma

qsd

## 4.3 Vágáserősség

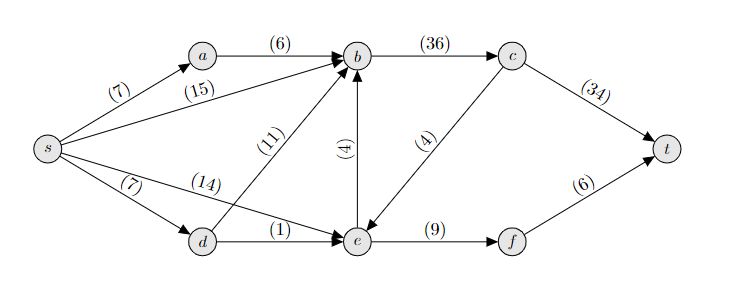
asd



**4.6. ábra: Vágás csúcsai**

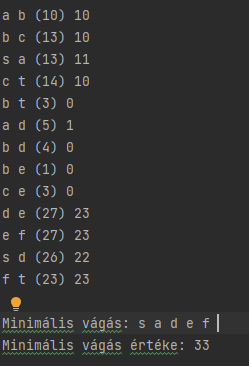
## 4.4 Szórás

ads



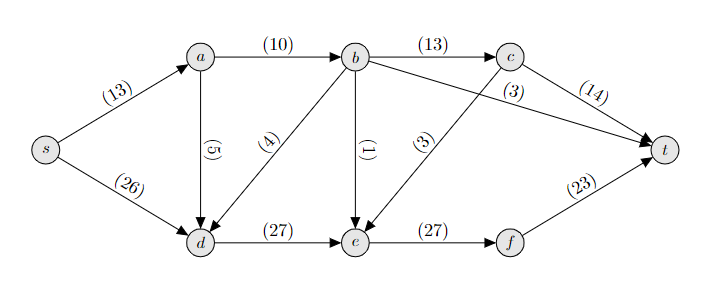
**4.7. ábra: Egyszerű hálózat szórás vizsgálatához**

## 4.5 Teljes képlet



2.9964165529656492 0.16666666666666666 0.23809523809523814 1.0 0.0 1.180775494846827

asdasd



**4.8. ábra: Egyszerű hálózat a képlet bemutatásához**

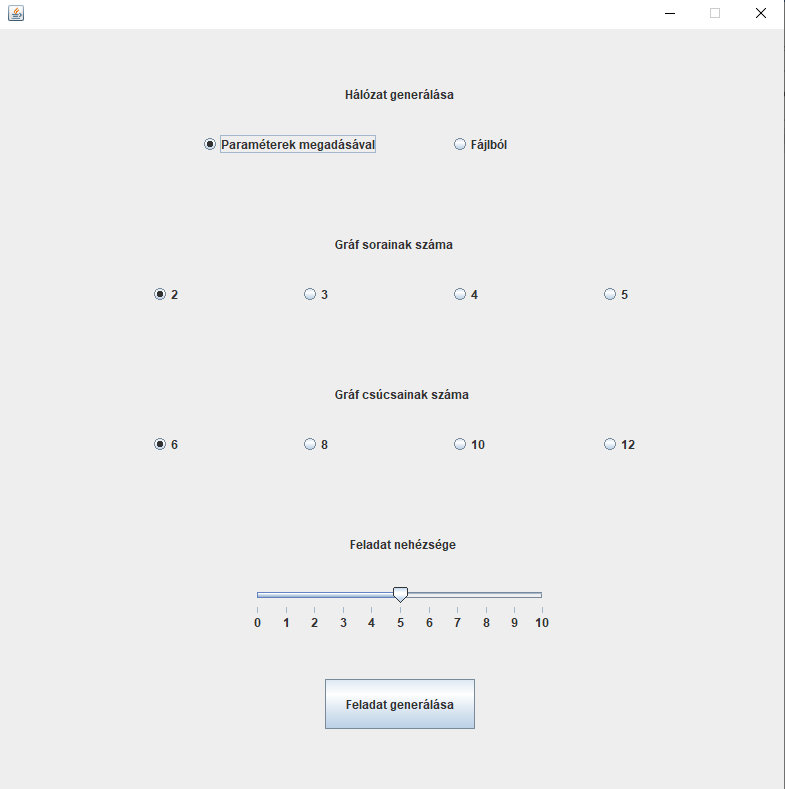
5. fejezet

# A program bemenetei

asd

## 5.1 Grafikus felület kezdőoldala

asd



**5.1. ábra: Kezdőképernyő**

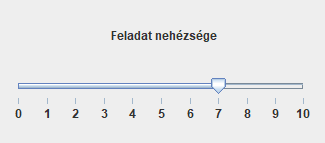
## 5.2 Hálózat generálása paraméterek megadásával

asd

A képen szöveg látható

Automatikusan generált leírás

**5.2. ábra: Gráf sorai és csúcsai számának megadása**



**5.3. ábra: Feladatnehézség megadása**

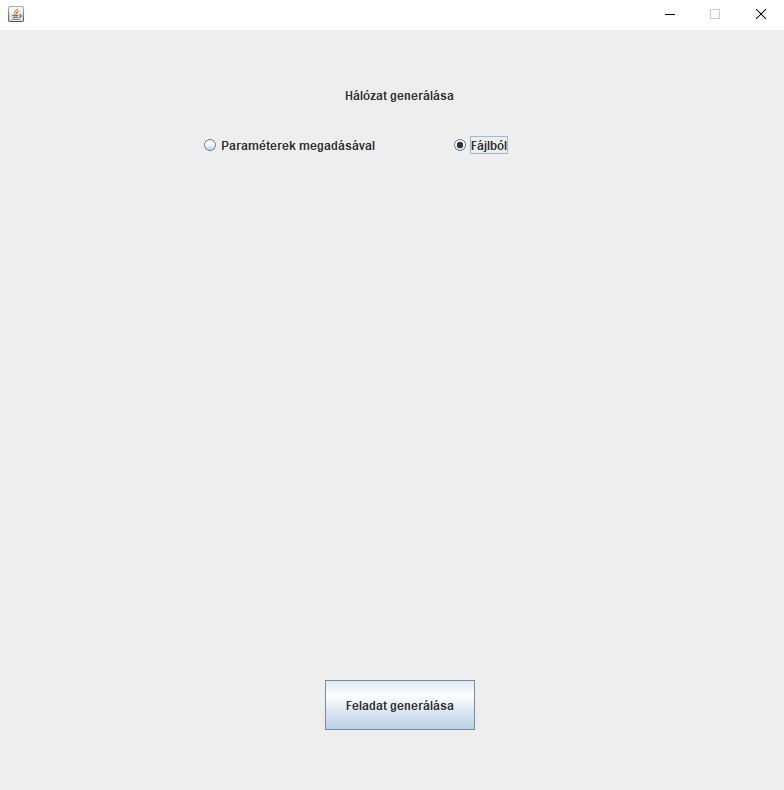
A képen különböző, különféle, változat látható

Automatikusan generált leírás

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

## 5.3 Hálózat generálása fájlból

asd



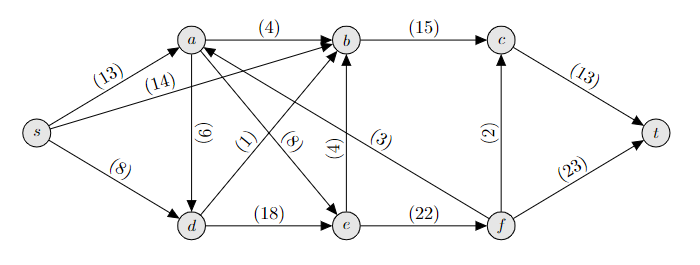
**5.5. ábra: Hálózat generálása fájlból**

### 5.3.1 Hálózat generálása random gráffal

asd



**5.6. ábra: Txt fájl tartalma random gráffal**



**5.7. ábra: Generált feladat random gráffal**

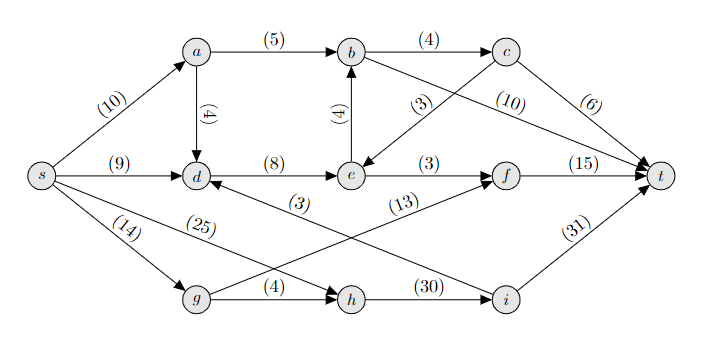
### 5.3.2 Hálózat generálása előre megadott gráffal és vágással

asd

A képen asztal látható

Automatikusan generált leírás

**5.8. ábra: Txt fájl tartalma egy megadoot gráffal és vágással**



**5.9. ábra: Generált feladat megadott gráffal és vágással**

6. fejezet

# A program kimenetei

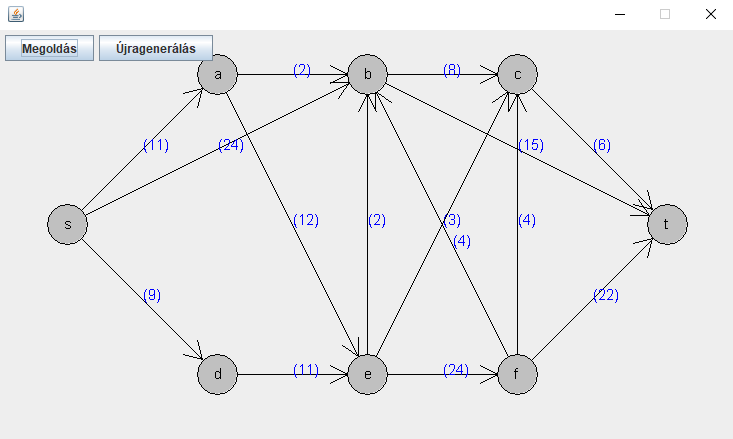
wd

## 6.1 Grafikus megjelenítés

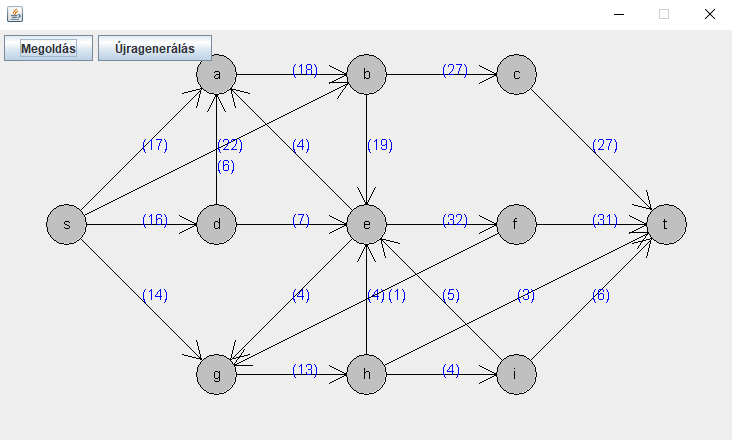
asd

### 6.1.1 Feladat

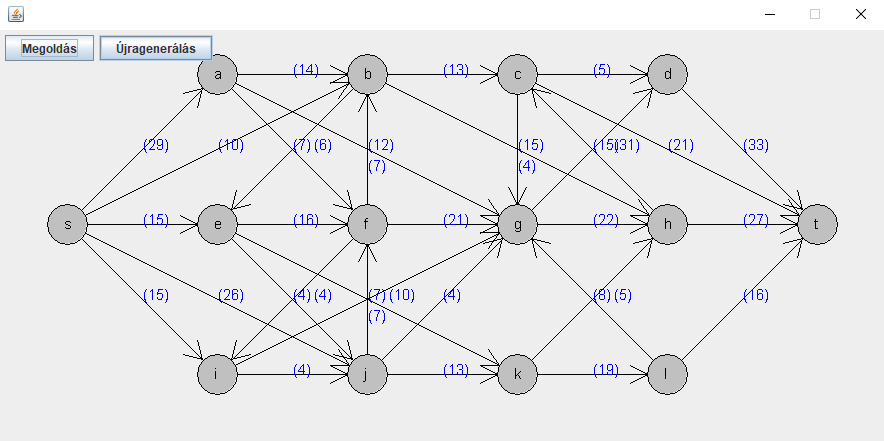
asd



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



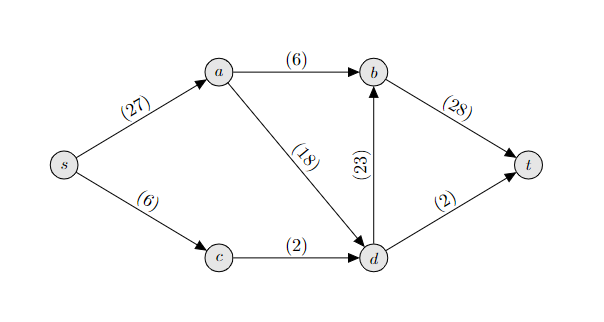
**6.2. ábra: 3×3-as feladat grafikus megjelenítése**



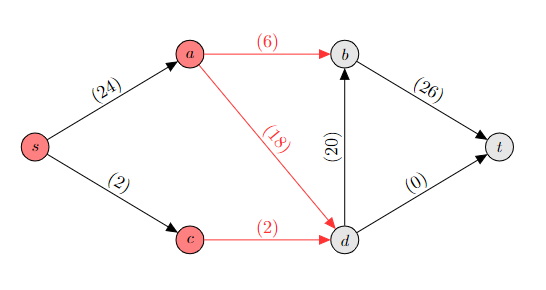
**6.3. ábra: 3×4-es feladat grafikus megjelenítése**

### 6.1.2 Megoldás

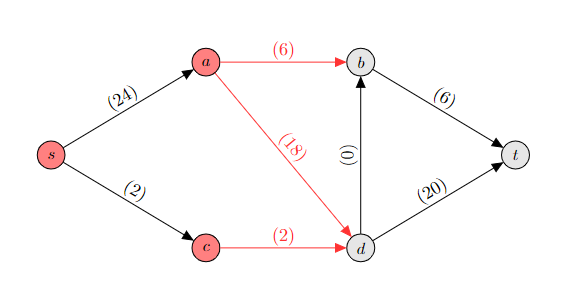
asd



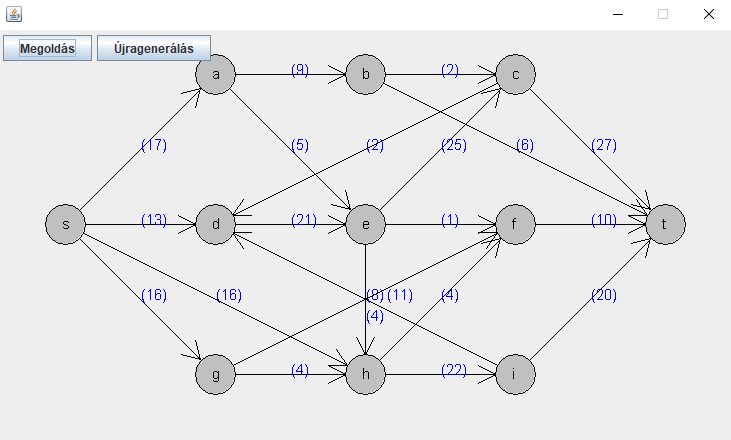
**6.4. ábra: Egyszerű hálózati folyam feladat**



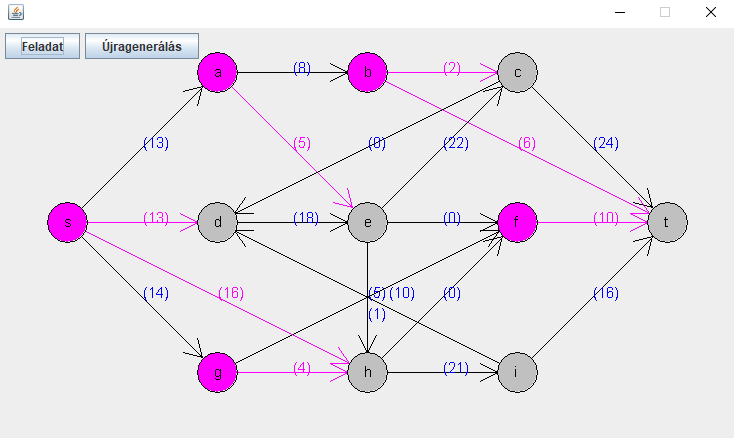
**6.5. ábra: Maximális folyam megoldás 1**



**6.6. ábra: Maximális folyam megoldás 2**



**6.7. ábra: Feladat megjelenítése grafikusan**



**6.8. ábra: Megoldás megjelenítése grafikusan**

### 6.1.3 Újragenerálás

asd

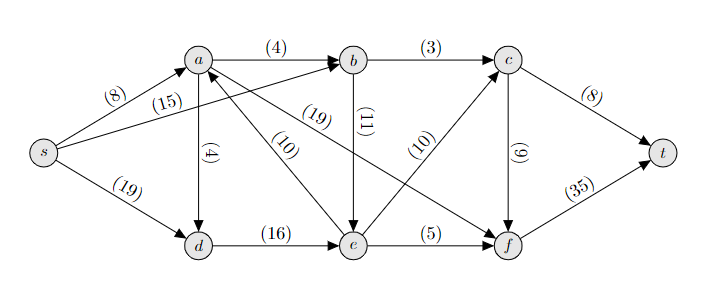
## 6.2 Txt fájl

asd

A képen asztal látható

Automatikusan generált leírás

**6.9. ábra: Txt fájl kimenet**



**6.10. ábra: Txt fájl által leírt feladat**

## 6.3 LaTeX fájl

asd

### 6.3.1 Program által generált LaTeX fájl

asd

\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}

A képen szállítás, nap látható

Automatikusan generált leírás

**6.11. ábra: LaTeX fájllal létrehozott hálózat képe**

### 6.3.2 Javított LaTeX fájl

asd

\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}

A képen szállítás látható

Automatikusan generált leírás

**6.12. ábra: Javított LaTeX fájllal létrehozott hálózat képe**

7. fejezet

# Összegzés

afasd

# Irodalomjegyzék

1. Szeszlér Dávid, Wiener Gábor: Bevezetés a számításelméletbe 2 jegyzet, <http://cs.bme.hu/bsz2/bsz2_jegyzet.pdf>
2. Wikipédia: Maximális áramlási probléma,
3. <https://hu.wikipedia.org/wiki/Maxim%C3%A1lis_%C3%A1raml%C3%A1si_probl%C3%A9ma>
4. Wikipedia: Flow network, <https://en.wikipedia.org/wiki/Flow_network>
5. <http://www.cs.columbia.edu/~bert/courses/3137/hw3_files/GraphDraw.java>