

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 feladatokautomatikus generálása

Konzulens

Dr. Katona Gyula

2022. december 9.

Tartalomjegyzék

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. 09.

...…………………………………………….

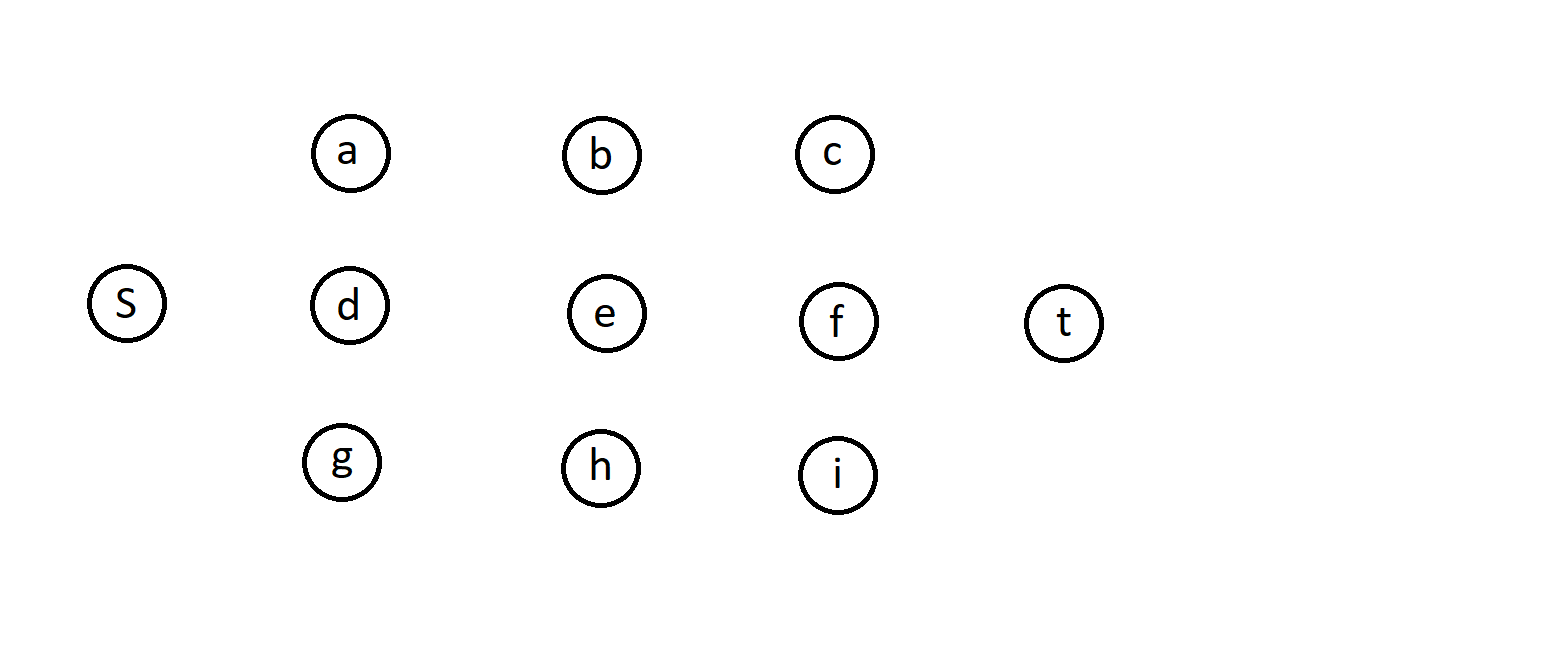
Tokovics Dávid

**Összefoglaló**

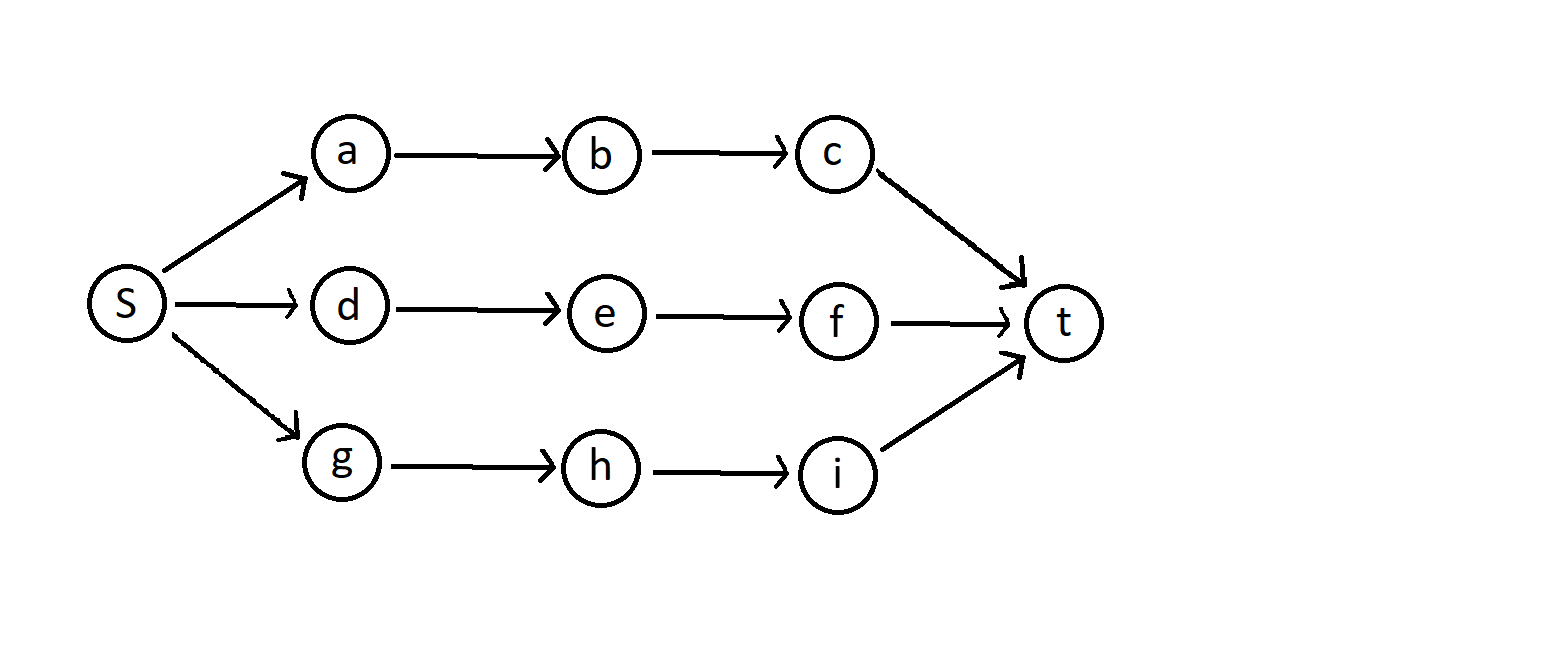
**Abstract**

Feladatom egy program írása volt, amely a megfelelően bevitt paraméterek alapján hálózati folyamos feladatokat generál. A bevitt paraméterek kétfélék lehetnek, egyrészt futtathatjuk a programot egy általunk készített gráfra, a megadott vágással együtt, illetve le is generáltathatjuk a gráfot, valamint a vágást. Mindkét esetben onnantól, hogy megvan a gráf, és a hozzá kapcsolódó vágás, a hálózati folyamos feladatot ugyanúgy generáljuk le. De előbb ejtsünk szót a gráf és a vágás generálásáról.

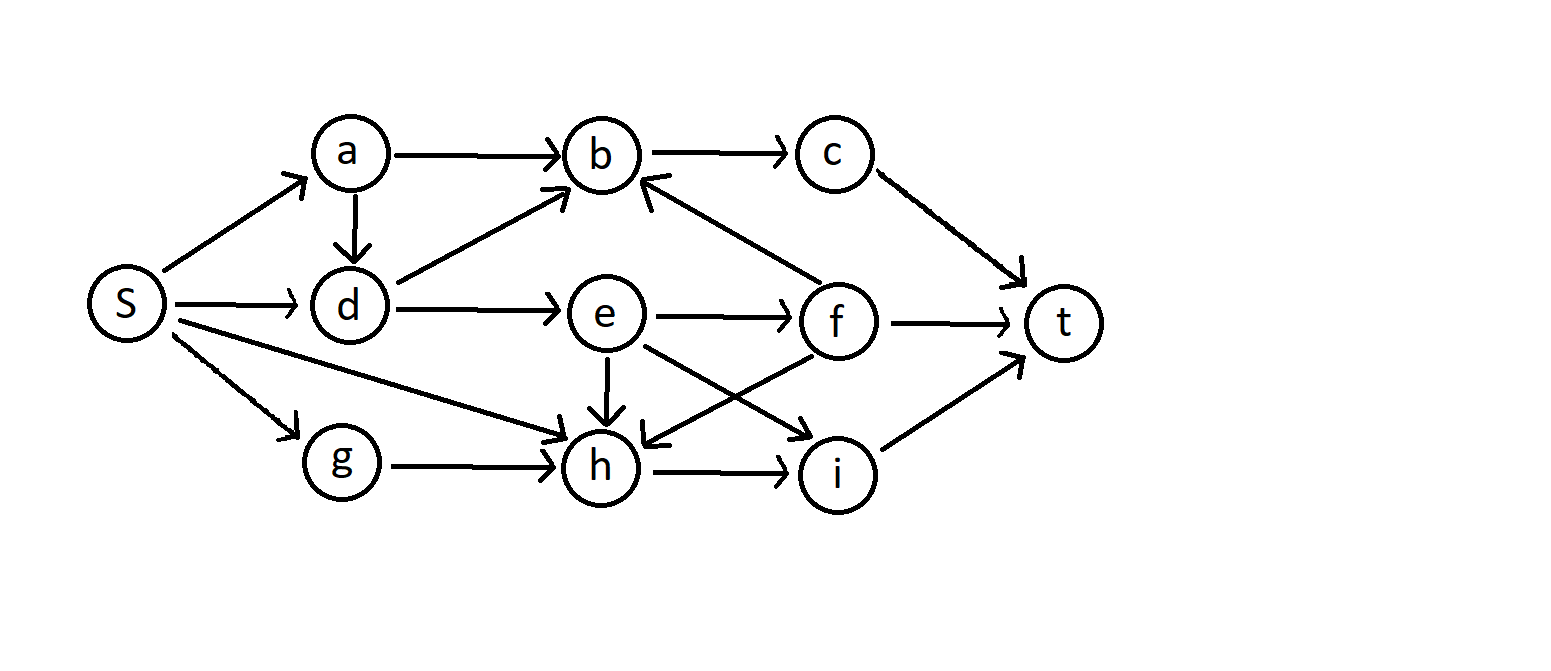
A programunk olyan gráfok generálására képes, amely az s-t és t-t leszámítva a bemeneten megadott számú sorból állnak, és minden sorban ugyanannyi csúcsunk van. A csúcsok számát is meg kell adnunk hozzá (ez az összes csúcs darabszáma, és nem az egy sorban lévőké), majd ebből, ha úgy jönne ki, hogy nem minden sorban ugyanannyi csúcs van, akkor a plusz csúcsokat elhagyja. Így tehát, ha 3 soros gráfot szeretnénk, és 16-ot adtunk meg csúcsszámnak, akkor az s-t, meg t-t levonva 14 csúcsot kapunk, és így soronként 4 csúcs lesz, a maradék 2-t pedig elhagyjuk. Tehát mondjuk a 3 11 bemenetre az alábbi csúcselrendezésű gráfot kapjuk:



Ebbe kell tehát az éleket behúznunk. Először egy keretet húzunk be neki élekből, vagyis először s-ből minden sor első csúcsába, emellett minden sor utolsó csúcsából t-be, valamint minden sorban az összes csúcsból (kivéve az utolsóból) húzunk egy élt a következő csúcsba a sorban. Így a fenti csúcselrendezésből megkapjuk a következő gráfot:

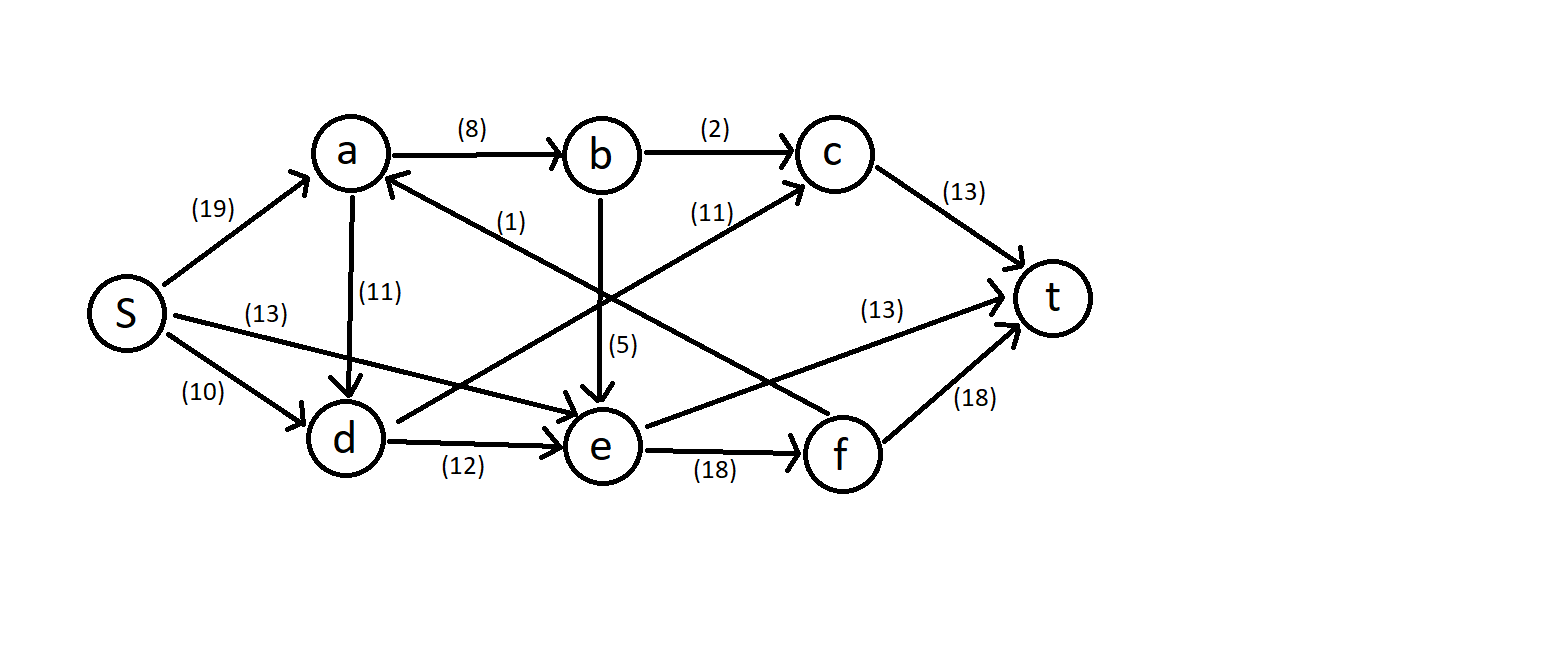


Ezután pedig húzunk még be csúcsokat az alábbiak szerint: s-ből a vele nem azonos sorokban a második csúcsba, t-be a vele nem azonos sorokban a hátulról a második csúcsból, valamint a sorokban lévő csúcsokból a szomszédos sorok csúcsaiba, ha azok a sorban maximum 2-vel balra vagy maximum 2-vel jobbra helyezkednek el. Ezeket azonban nem mindet húzzuk be, mindegyiket bizonyos valószínűséggel. A visszafele, tehát balra tartó éleket egy picit kisebb valószínűséggel, valamint a 2-vel jobbra vagy balra lévőket is kisebb valószínűséggel húzzuk be. Két csúcs közé így nagyjából 40-50% valószínűséggel húzunk be valamelyik irányba élt, ha megfelel a feltételeknek. Így kaphatjuk meg például a következő gráfot:

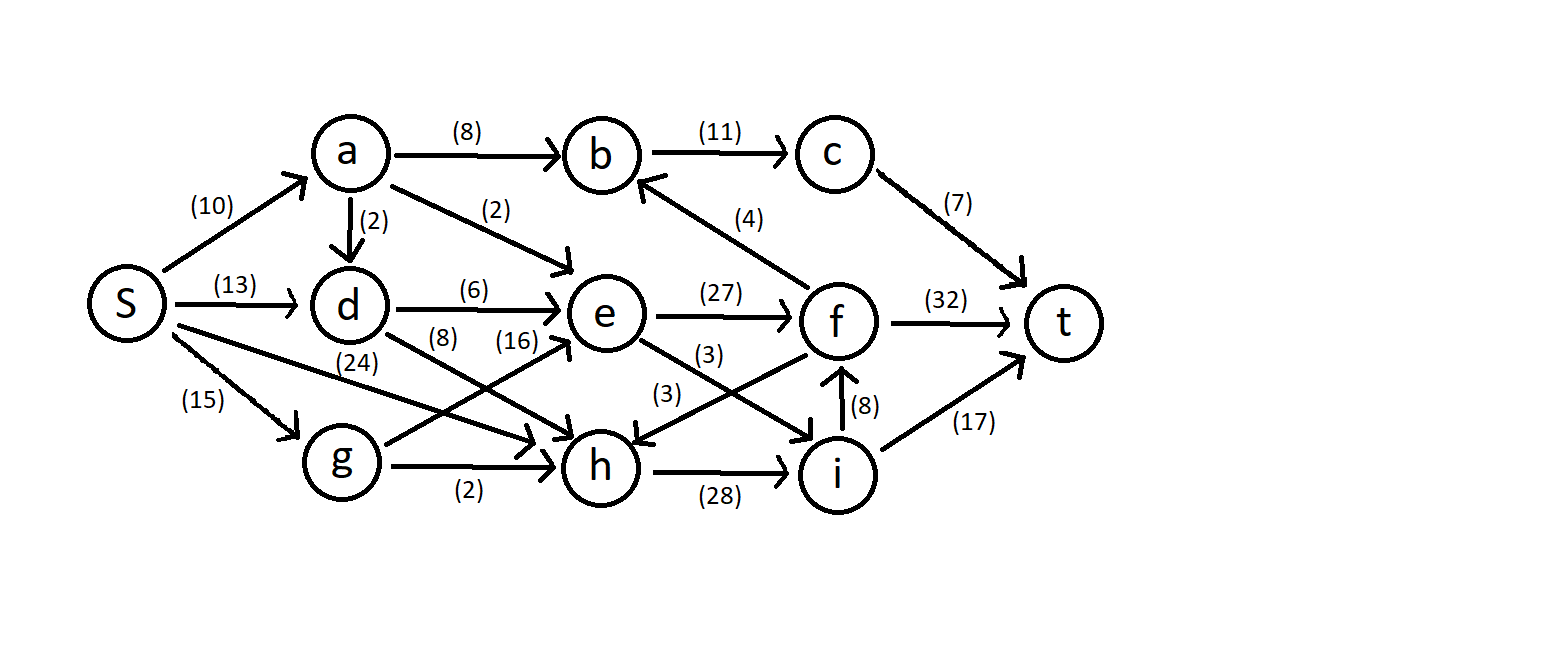


Ezzel tehát végeztünk a gráf legenerálásával, most a vágás generálása következik. Ezt elég egyszerűen tesszük meg. Beletesszük a vágás csúcsai közé s-t, majd végigmegyünk minden csúcson a t kivételével, és 2/5 eséllyel bevesszük a vágásba az adott csúcsot. Ha nincs legalább 3 csúcs a vágásban, akkor újrageneráljuk, és ezt addig folytatjuk, ameddig nem kapunk egy legalább 3 csúcsból álló vágást. Ha megvan a gráfunk és a vágásunk is, akkor innentől ugyanaz a feladat, mint amikor mi vittük be a gráfot és a vágást.

A gráfból és a vágásból kellene most egy folyamos feladatot létrehoznunk. Először a kérdés az, hogy milyen is egy jó folyamos feladat, ugyanis, ha csak random számokat írunk az élekre, az nem lesz szép. Ugyanis akkor lesz megfelelő egy ilyen feladat, ha az minél jobban hasonlít egy folyamra, nem szeretnénk például, hogy a t-be tartó élek kapacitásának összege kétszer annyi legyen, mint az s-ből kimenő éleké. Így tehát első lépésként szeretnénk egy folyamot csinálni a gráfunkból. Eredetileg minden élre 0-t írjunk folyamértéknek. Ezután keressünk s-ből t-be vezető utakat, és növeljük egy random kiválasztott 1 és 4 közötti egész számmal az úthoz tartozó élek folyamértékeit. Hasonlóan a javítóutas algoritmushoz, amennyiben az éleken 0-nál legalább a kiválasztott értékkel nagyobb szám áll, használhatjuk őket visszaélként, ekkor csökkentjük az élen található folyamértéket a kiválasztott értékkel. Annak érdekében, hogy ne kerülhessünk végtelen ciklusba, minden csúcs maximum egyszer szerepelhet egy útban. Ezen kívül fontos az a kikötés is, hogy a kiválasztott vágásunk minimális vágás legyen, ezt úgy tudjuk biztosítani, hogy ha egyszer áttérünk az úttal a vágás másik, t felőli oldalára, akkor vissza az s oldalára csak egy visszaéllel mehessünk, ellenkező esetben lehetne olyan vágás, amihez képest növeljük a kiválasztott vágásunkat, így az nem lesz minimális. Ilyen utakat keresve és a folyamértékeket növelve tehát kapunk egy érvényes folyamot, aminek minimális vágása a kiválasztott vágás (viszont feletehetőleg nem az egyetlen). Annak érdekében, hogy biztosan a vágásunk legyen az egyetlen minimális vágás, muszáj lesz elrontanunk a folyamot, vagyis már nem folyamértékként, hanem kapacitásként meg kell növelnünk minden másik vágásból legalább egy élt. Ennek legegyszerűbb módja, ha simán a vágás élein kívül minden másik élen a kapacitást növeljük mondjuk egy 1 és 4 közötti egész számmal. Ezzel tehát kapunk egy folyamos feladatot. Így kaphatjuk meg például az alábbi 2 8 bemenetű gráfot:



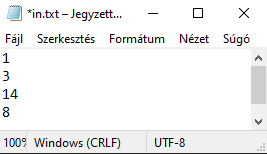
Ezek a folyamos feladatok azonban nem mindig azonos nehézségűek lesznek, mivel a randomitás által nagyon eltérő hálózatokat is kaphatunk. Éppen emiatt szeretnénk, ha lenne egy mérőszámunk a nehézségre, és aszerint generálhatnánk feladatot, hogy minél közelebb legyen egy kiválasztott nehézséghez. Ehhez kell találnunk olyan tényezőket, amelyek befolyásolják a feladat nehézségét. Például minél több az (nem 0 kapacitású) él a gráfban annál nehezebb lesz a feladat, illetve minél több benne a visszaél, vagyis jobbról balra mutató él, az is nehezíti a feladatot. Könnyítheti azonban az, ha sok az olyan él, ami egy maximális folyamban 0 folyamértékkel rendelkezik, mivel az ilyen maximális folyamokat sokkal könnyebb megtalálni. Nevezzük ezeket nulléleknek Nyilvánvalóan mindháromból annál több van, minél több csúcsa van a gráfnak, így, hogy ne legyen teljesen nagyon eltérő a sok csúcsú gráfokra a nehézség, mint a kevesebb csúcsúakra, így a csúcsszám arányában kell nézni ezeket. Nehezíti még az a feladatot, ha nehéz megtalálni a vágást. Nehezíti például az, ha az egyik csúcs benne van a vágásban, a sorban tőle balra lévő (ha van) pedig nincs, illetve az még jobban, ha emellett az attól balra lévő (ha van) az megint benne van, szóval, ha van egy olyan csúcshármas valamelyik sorban, ahol a középső csúcs nincs benne, a másik kettő viszont benne van a vágásban. Ezek számát vegyük együtt a vágáserősségnek. Emellett még az élekre írt kapacitások szórása is befolyásolja a nehézséget, minél egyenletesebben vannak elosztva az élek kapacitásai, annál nehezebb lesz a feladat. Így ezek figyelembevételével jött létre az alábbi képlet:   
***nehézség = (élek száma – nullélek \* 2 + visszaélek \* 4) / csúcsok száma \* (vágáserősség + 1) / szórás \* 20***  
Ez nagyjából egy 0 és 10 közötti szám szokott lenni általában, de tud negatív, és 10-nél nagyobb is lenni. Így nagyjából megfelel nekünk arra a célra, hogy 0-tól 10-ig egy egész számmal jelezzük, hogy milyen nehézségű feladatot szeretnénk.



A fenti hálózat nehézsége 7,544, tehát nagyjából 7 vagy 8.

Így tehát a bemeneti paraméterek között kapunk egy nehézséget is, ehhez kéne közelítenie az általunk generált gráfnak. Ezért az eddigi lépéseket nem csak egyszer futtatjuk le, hanem 50-szer. Ha mi adjuk meg a programnak a gráfot és a vágást, akkor természetesen mindig ugyanarra a gráfra fog lefutni a hálózat generálása, azonban, ha a gráfot generáltatjuk, akkor az mind az 50 lefutáskor újragenerálódik, és egy másik gráfból csinál új hálózatot a program. Minden generált hálózatra lefuttatja a nehézség vizsgálatát, és a várt nehézséget legjobban megközelítő feladatot fogjuk eredménynek megkapni.

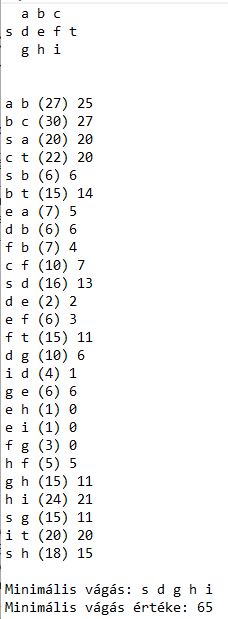
A program bemenete és a kimenete is egy-egy txt fájl, az in.txt és az out.txt. Az in.txt fájlban egymás után vár egész számokat, az első egy 0-s vagy egy 1-es, aszerint, hogy mi adjuk meg a gráfot, vagy szeretnénk, hogy a program generálja le nekünk, az előbbi a 0, az utóbbi az 1. Utána a következő szám, amit vár, hogy hány sorból álljon a gráfunk majd az utána következő pedig, hogy hány csúcsból. Az ezt követő szám pedig az, hogy mi legyen a nehézsége a feladatnak. Ha a programmal szeretnénk generáltatni a gráfot, akkor ennyit elég megadnunk neki. Így tehát ebben az esetben a következőképpen kell kinéznie a bemeneti fájlnak:



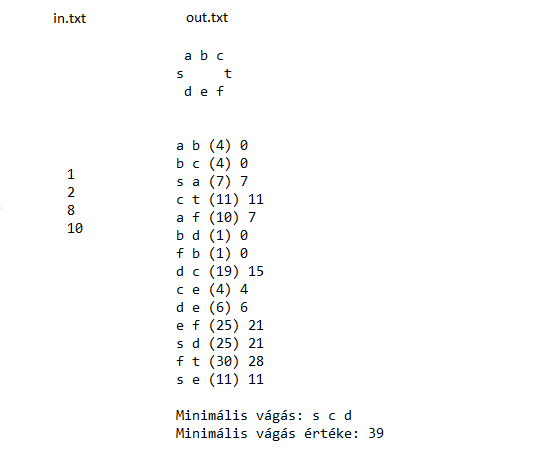
Amennyiben mi szeretnénk megadni a gráfot a vágással együtt, akkor még további beviteli adatokra van szükség. A nehézség után még két számnak kell szerepelnie. Az egyik, hogy hány éle van a gráfnak, a másik pedig, hogy hány csúcsból áll a vágás. Ezután a program azon feltételezése mellett, hogy a csúcsok s, t, valamint az abc betűi a-tól kezdve sorban, és a bemenet s, valamint a kimenet t, megadjuk a gráf éleit sorban egymás alá, a kezdő-, és végcsúccsal megadva. majd a legvégén az utolsó sorban megadjuk a vágás csúcsait. A fentebb megadott gráfot tehát így tudjuk megadni a programnak:



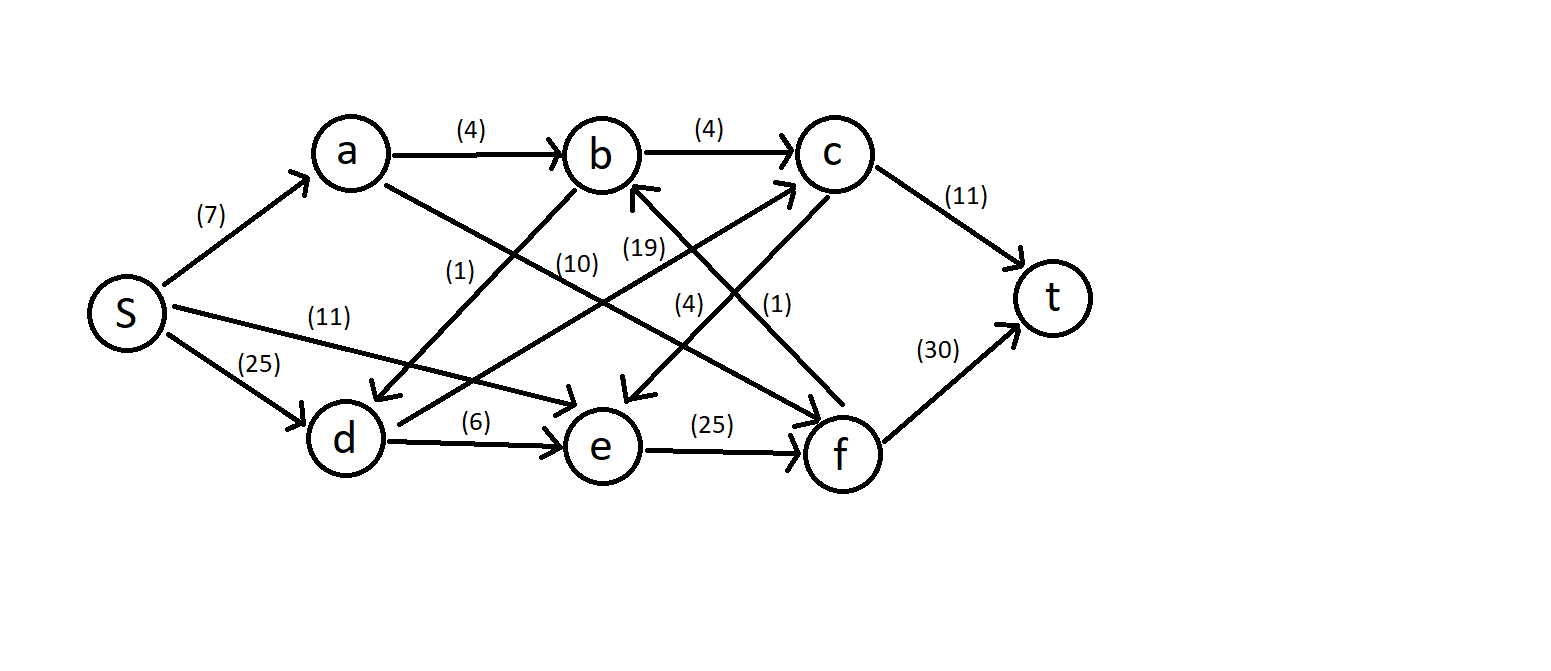
Ha elindítjuk a programunkat, akkor az kezdésként beolvassa a bemeneti txt fájlt, és létrehozza vagy a megadott vagy egy random gráfot, majd minden élnek 0 értéket ad. Ezután a programunk elvégzi a fentebb említett műveleteket. A gráf generálása simán a konstruktorban történik a megadott bemenetekkel, külön konstruktor van a generált, valamint a bemenetben megadott gráfok létrehozására. A vágás sorsolására van egy külön függvényünk a konstruktoron kívül. Ezután a folyam elrontására, vagyis arra, hogy az általunk preferált vágás legyen az egyetlen minimális vágás, valójában egy olyan függvényünk van, ami még akkor növeli meg a vágáson kívüli élek kapacitását, amikor még minden élnek 0 a kapacitása. És csak ezután kezdünk el utakat növelni benne. Erre van egy függvényünk, ami utakat keres és megnöveli az élekre írt értéket egy 1 és 4 közötti random sorsolt számmal. (Egy út során mindig ugyanazzal a számmal.) Az utakat s-ből indulva keresi, és mindig a lehetséges élek közül sorsol, amelyeken tovább haladhat. Lehetséges, hogy zsákutcába ér, ilyenkor megkeresi az előző csúcsot, ahol volt, és onnan egy másik irányba indul, el, és megjegyzi, hogy abból a csúcsból nem tud errefelé továbbmenni. Így egy idő után véges időn belül el fog jutni t-be, ha létezik s és t között út. Eközben a függvény figyel arra, hogy ha átment a vágás túloldalára, akkor ne jöjjön vissza, legfeljebb egy visszaéllel, valamint a visszaélek esetében figyeli, hogy ne lehessen negatív az élre írt érték, valamint természetesen figyeli azt is, hogy ne kerüljünk vissza egy olyan csúcsba, ahol már jártunk. Ezt a függvényt lefuttatjuk annyiszor, amennyi a gráf éleinek számának 1,3-szorosának alsó egészrésze. Az így kapott hálózatra lefuttatjuk a nehézség kiszámolásáért felelős függvényünket. Majd, ha ez nem az első futás, akkor összehasonlítjuk az eddigi legközelebb a kívánt nehézséghez lévő hálózat nehézségével. Ha közelebb van, akkor eltároljuk, és elvetjük az eddigi legközelebb lévő gráfot. Ezt 50-szer lefuttatjuk. Fontos, hogy csak azokat a hálózatokat vesszük figyelembe, a generált gráfok esetén, ha a keretben, tehát a legelején nem randomizáltan behúzott élek közül egyiknek sem 0 a kapacitása. Ha ilyen jönne ki, akkor arra le sem fut a nehézség vizsgálata, és nem számítja bele az 50 futásba. Ezután az 50 futás után megkapunk egy hálózatot, ami a program által generált feladat lesz. A kapott feladatot eredményként kiírja a program egy out.txt nevű fájlba a következőképpen: először „kirajzolja” a gráf csúcsait, hogy milyen sorrendben, hogyan szerepelnek, majd kiírja az éleket egyesével a kapott kapacitásokkal zárójelben, valamint egy folyamértéket, ami hálózat egy maximális folyamához tartozik. Az élek után még szerepel a minimális vágás, és ennek értéke is. Így tehát az 1 3 11 8 bemenetre lefuttatva az alábbi eredményt kaphatjuk például az out.txt-ben:



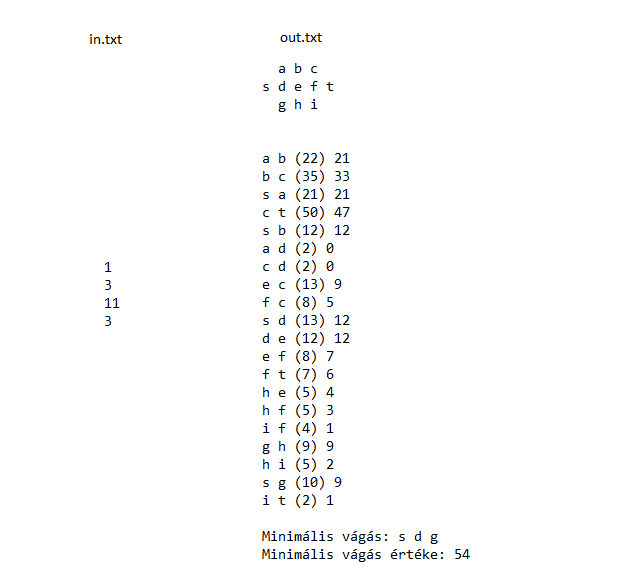
Nézzünk meg pár generált feladatot még példának:

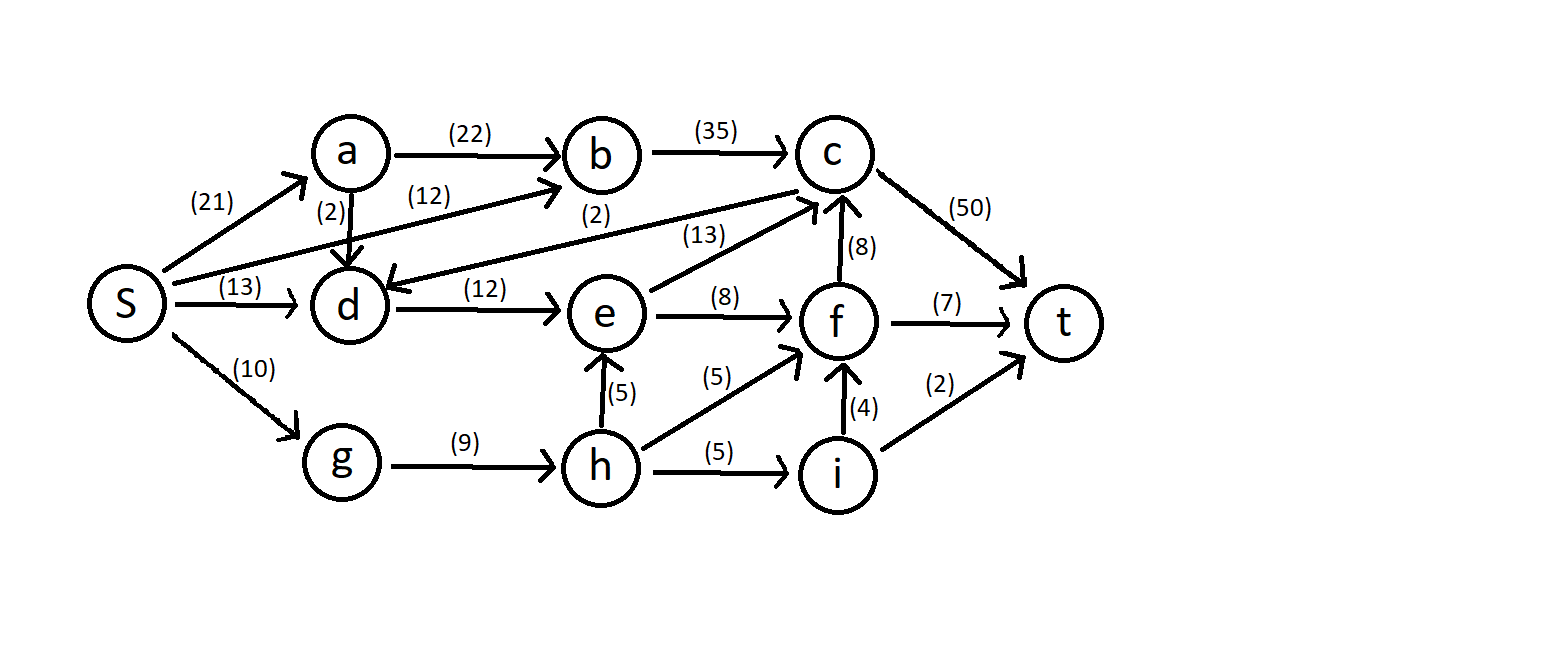


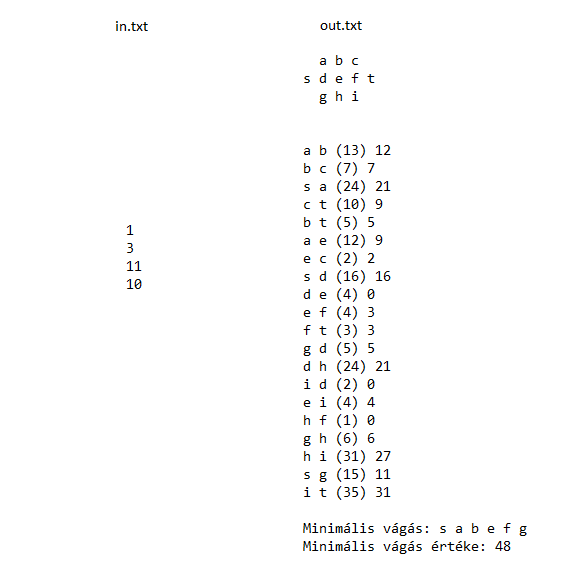
Ezekhez a be- és kimenetekhez a következő folyamos feladat tartozik:

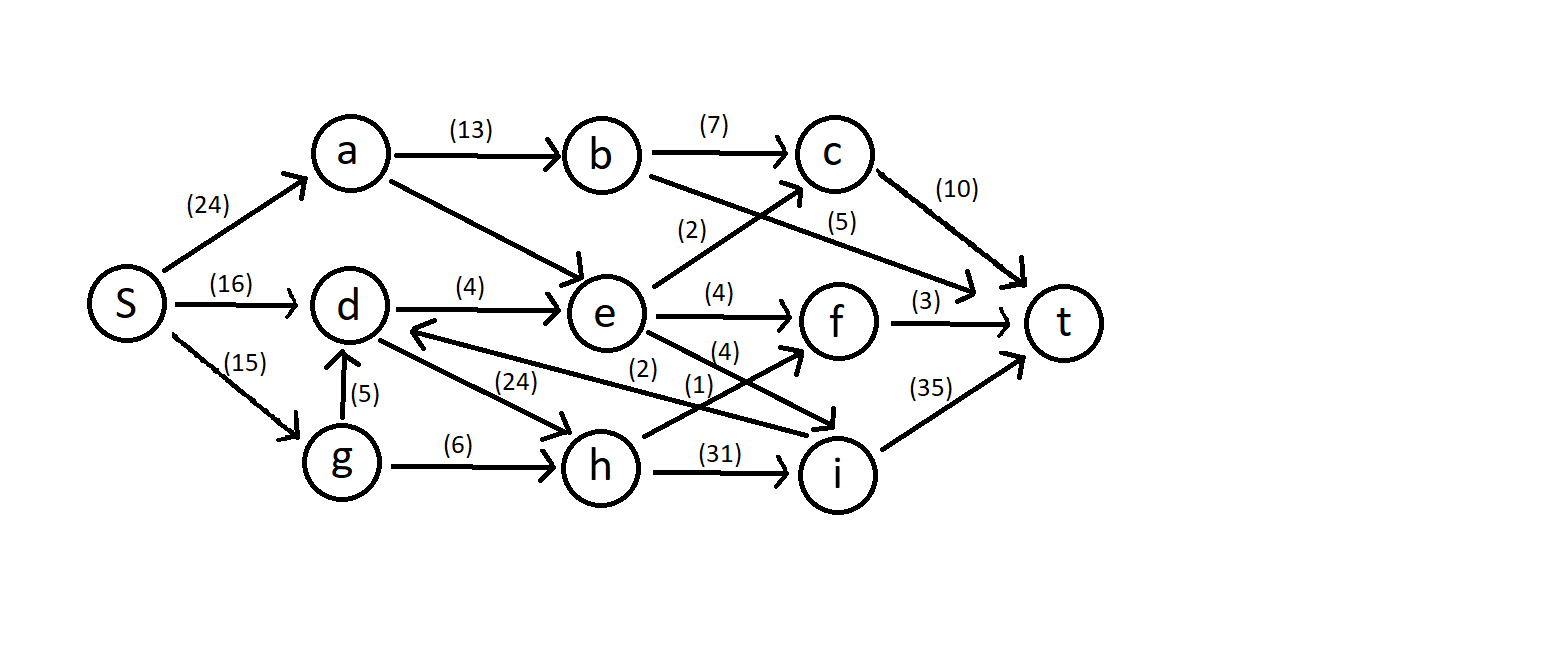


Nézzünk meg még 3 soros 11 csúcsúból kettőt, először egy könnyűt, majd egy nehezet:









Így működik tehát a hálózati folyamos feladatok automatizált előállítására szánt programom, a jövőben fejleszthető lenne egy kezelői felülettel, hogy ne txt fájlban kelljen a paramétereket bevinni, valamint a kimenet is lehetne a txt fájl helyett egy kép amin rajta van már a lerajzolt gráf.