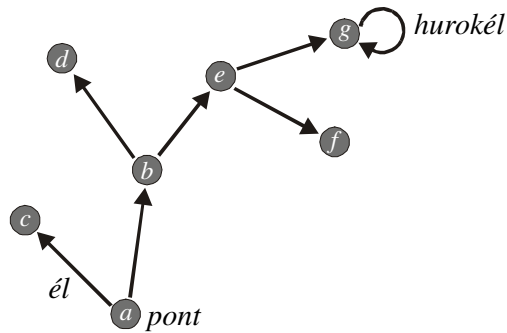


#### 4/2. tétel:

#### Graphok fogalma és ábrázolása. Alapvető graphalgoritmusok. Szélességi és mélységi keresés.



$G = \langle V, E \rangle$  rendezett pár, ahol:

$V$  (vertex): pontok, csomópontok, csúcsok, szögpontok nemüres halmaza

$E$  (edge): élek halmaza, mely egy homogen relatio  $V-n: E \subseteq V \times V$ .

Hurokél: kezdőpontja azonos a végpontjával.

Többszörös él: két pontot több él köt össze.

Egyszerű graph: nincs benne hurok és többszörös él.

Az ábrán lévő graph adatai:

$V = \{a, b, c, d, e, f, g\}$

$E = \{(a,b), (a,c), (b,d), (b,e), (e,f), (e,g), (g,g)\}$

Írányított graph: az élek rendezett párok, pl.  $(u,v)$  él az  $u$  pontból vezet a  $v$  pontba. Az  $u$  pont a kezdőpont, a  $v$  pont a végpont. Viszont lehet az is, hogy két pont között symmetricus a kapcsolat:  $(u,v)$  és  $(v,u)$  pár is létezik. Ettől még irányított!

Írányítatlan graph: az élek rendezetlen párok. Az adott élen egyik irány sincs kitüntetve a másikkal szemben.

Szomszédos pontok: adott kiindulópont esetén a belőle 1 lépésben elérhető pontok. Nem irányított graph esetén mindazon pontok szomszédosak, melyek között 1 lépésnyi távolság van. Két pont szomszédos, ha él köti őket össze; két él szomszédos, ha van közös pontjuk.

Pont fokszáma (foka): irányítatlan graph esetén az illető csúcsból kiinduló élek száma. Irányított graph esetén a bemenő fokszám (vagy befok) és kimenő fokszám (kifok) összege – a befok az adott pontba befutó, a kifok pedig az onnan kiinduló élek száma. Isolált (független) pont: fokszáma 0. Egyedülálló (singli-singularis) pont: fokszáma maximum 1 és ez is csak egy hurokél.

Üres graph: csak izolált pontjai vannak. Teljes graph: bármely két pontot él köt össze.

Séta: élsorozat. Vonal: olyan séta, melyben pont ismétlődhet, de él nem.

Út: olyan séta, melyben nincsenek ismétlődő pontok. Kör: zárt út, kezdőpontja és végpontja azonos.

A séta vagy út nyitott, ha  $a_0 \neq a_n$  (vagyis a kiindulópont és a végpont nem azonos). Zárt séta (út): a kiindulópont és a végpont azonos. Mindkét esetben  $n$  jelöli a séta (út) hosszát, vagyis a benne lévő élek számát. Zárt út másként irányított kör is lehet.

Teljes graph: az élek száma azonos a pontok számának négyzetével.  $|E| = |V|^2$ . Minden pont össze van kötve az összes többivel, valamint hurokéllal is rendelkezik. Ritka graph esetén az  $|E| \ll |V|^2$  összefüggés teljesül.

Pont elődje (őse): olyan pont, ahonnan az adott pontunkba vezet él. Pont utódja (leszármazottja): olyan pont, ahová az adott pontunkból vezet él. Adiacentia: környék, vagyis egy adott csomópont szomszédjainak halmaza.

Nyelő: olyan pont, melyből nem vezet ki él. Forrás: olyan pont, ahová nem vezet él.

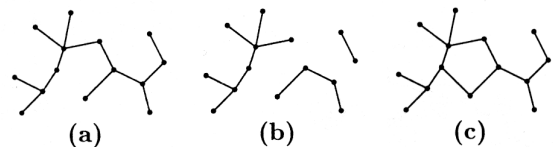
Összefüggő (egybefüggő) graph (irányítatlan): bármely két pontját összeköti élsorozat. Szétteső (disjoint) graph: vannak csomópontok, melyek között nincs összekötő út.

Erdő: körmentes, irányítás nélküli graph (b). Ha összefüggő, akkor (nyílt) fáról beszélünk (a). (A (c) sem fa, sem erdő.)

Vágóél: elhagyásával a graph összefüggő componense két részre esik szét.

Részháló:  $G' = \langle V', E' \rangle$ , ahol  $V' \subseteq V$ ,  $E' \subseteq E$ , de  $E' \subseteq V' \times V'$

Súlyozott graph: az egyes élekhez súlyfüggvényt rendelünk. Szomszédsági lista esetén könnyű tárolni.

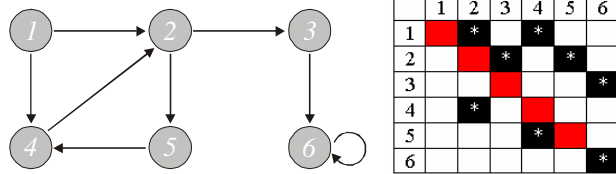


A graphokon értelmezett relatiók – egy adott graph:

- reflexiv, ha minden csomópontja rendelkezik hurokéllal.
- symmetricus, ha egy meghatározott él létezésekor annak megfordítottja is létezik. Értelmszerűen irányítatlan graph mindig symmetricus.
- antisymmetricus, ha bármely két tetszőleges pont között csak egy irányban található él.
- transitív, ha két olyan pontja között, amelyek között létezik 2 lépésből álló irányított séta; létezik közvetlen összeköttetés is, azonos irányban.
- dichotom, ha bármely 2 pontja között legalább 1 irányban van él.

## Graphok ábrázolása

### 1. Csúcsmatrix



Inkább sűrű graphokra, de az algoritmusok egyszerűbbek. Az irányítatlan graph esetén a matrix az átlóra symmetricus. Az átlóban (piros) az esetleges hurokélek találhatóak.

### 2. Szomszédsági listák

1–2,4; 2–3,5...etc. Ez helytakarékos, különösen ritka graphoknál. Az algoritmusok esetenként bonyolultabbak.

### 3. Éllisták

1–2, 1–4, 2–3....etc. (az összekötött pontpárokat tároljuk). Csak akkor jó, ha nincs izolált pont.

## Alapvető graphműveletek

- előd / utód meghatározása
- graph transponált:  $G^T$ , ez az élek megfordítását jelenti – csúcsmatrix esetén tükrözést jelent a tengelyre
- graph complemter: ahol él volt, ott megszüntetjük; ahol nem volt él, ott létrehozunk – csúcsmatrix esetén a biteket ellentétesre változtatjuk – eredetivel együtt eredménye egy teljes graph
- graph symmetricus lezárása:  $G + G^T$
- graph transitiv lezártja: minden létező  $u \rightarrow v$  és  $v \rightarrow w$  út esetén  $u \rightarrow w$  út felvétele

## Szélességi keresés

Ehhez hasonló elvet használ a Dijkstra és a Prim–algoritmus – lásd 5. tétel.

A módszer egy kitüntetett ( $s$ , kezdő) pontból (csúcsból) kiindulva végignézi az összes élt, így minden olyan pontot megtalál, amely onnan elérhető. Eközben kiszámítja az elérhető pontok távolságát (legkevesebb él)  $s$ -ből. Minden pont esetén az  $s$ -ből való elérés legrövidebb útját tárolja.

A módszer a már elért és a még nem bejárt pontok közötti határvonalat egyenletesen terjeszti ki a határ teljes szélében (innen az elnevezés). Azelőtt elérjük az összes,  $k$  távolságra lévő pontot, mielőtt akár csak egy,  $k+1$  távolságra lévő pontot is elérnénk.

A csúcsokon 3 színt különböztetünk meg.

- fehér: kezdetben minden pont ilyen és addig ilyen is marad, amíg a keresés során el nem érjük.
- szürke: ha egy pontot egyszer már elértünk, színe szürke lesz
- fekete: akkor színezzük ilyenre, ha az összes szomszédja már elért (tehát szürke) csúcs

A szürke csúcsnak lehetnek fehér szomszédai, a feketének csak szürke vagy fekete színű szomszédai vannak. A szürke csúcsok jelentik a határt a felfedezett–felfedezetlen rész között.

A keresés során szélességi fa készül. Ez kezdetben csak a gyökeret ( $s$ ) tartalmazza, ez a kezdőpont. Ha egy fehér  $v$  pontot elérünk  $u$  pont vizsgálata során, akkor a fa bővül a  $v$  csúccsal és az  $(u, v)$  éllel. Az  $u$  csúcs ezesetben a  $v$  csúcs szülője. Mivel minden csúcsot csak egyszer érünk el (vagyis egyszer színezzhető fehérből szürkévé), minden csúcsnak csak egy szülője van – függetlenül attól, hogy más úton is elérhető. Beszélhetünk viszont megelőző és rákövetkező csúcsokról, melyek a fában  $s$ -hez közelebb, illetve távolabb helyezkednek el.

A szélességi keresésnél szomszédsági listás tárolást feltételezünk. Adott csúcs színét a  $szin[u]$  adja meg, elődjét pedig a  $\pi[u]$ . Az  $s$ -től való távolságot  $d[u]$  tárolja.  $Q$  sor pedig a szürke csúcsok tárolására szolgál (melyeket még meg kell vizsgálni). Az  $Adj[u]$  segítségével a vizsgált csúcs szomszédait kapjuk meg.

SZK( $G, s$ )

```

1  for minden  $u \in V[G] - \{s\}$  csúcsra
2      do  $szin[u] \leftarrow FEHÉR$ 
3           $d[u] \leftarrow \infty$ 
4           $\pi[u] \leftarrow NIL$ 
5   $szin[s] \leftarrow SZÜRKE$ 
6   $d[s] \leftarrow 0$ 
7   $\pi[s] \leftarrow NIL$ 
8   $Q \leftarrow \{s\}$ 
9  while  $Q \neq \emptyset$ 
10     do  $u \leftarrow fej[Q]$ 
11         for minden  $v \in Adj[u]$ -re
12             do if  $szin[v] = FEHÉR$ 
13                 then  $szin[v] \leftarrow SZÜRKE$ 
14                      $d[v] \leftarrow d[u] + 1$ 
15                      $\pi[v] \leftarrow u$ 
16                     SORBA( $Q, v$ )
17     SORBÓL( $Q$ )
18      $szin[u] \leftarrow FEKETE$ 

```

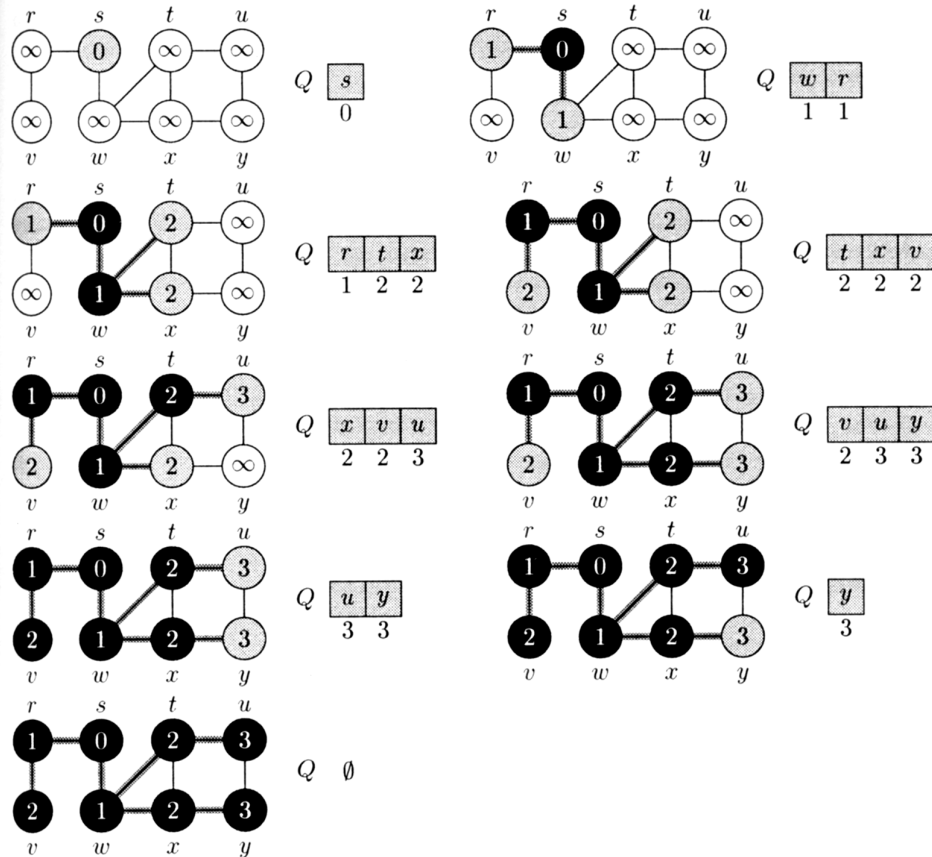
Az 1–4. sorok beállítják az összes csúcsot fehér színűre. Kezdeti távolságuk az  $s$ -től végtelen, szülőjük (egyelőre) nincs. A beállításból az  $s$  csúcs kimarad.

Az 5–8. sorok az  $s$  csúcsot szürkére állítják. Távolsága  $s$ -től 0, szülője nincs. Betesszük a  $Q$  sorba.

Az ezután kezdődő cyclus addig tart, amíg a  $Q$  sor ki nem ürül (9). Mindig vesszük a sor legelső elemét (10) és keressük az összes szomszédját (11). Ha a szomszéd színe fehér volt (vagyis még ismeretlen csúcsról van szó, 12), akkor a következő művelet sor zajlik: átszínezzük szürkére, a szülő  $d$ -értékénél eggyel nagyobbat tárolunk, illetve a szülő értékét beállítjuk (13–15). Ezután a már elért (szürke) csúcsot is betesszük a sorba (16).

Miután egy éppen vizsgált pont összes szomszédját feltérképeztük, kivesszük őt a sorból (17) és színét feketére állítjuk (18).

Az ábrásor egy graph fokozatos bejárását mutatja be,  $s$ -ből kiindulva:



## Mélységi keresés

A graphban a lehető legmélyebben keresünk: amíg lehet, az utoljára elért, új kivezető élekkel rendelkező csúcsból kiinduló, még nem vizsgált éleket derítjük fel. Ha a legmélyebben lévő pont összes kivezető élét megvizsgáltuk, eggyel visszalépünk...etc. Ezt addig végezzük, amíg el nem érjük az összes csúcsot, melyek elérhetők a kiindulópontból. Ha maradt még nem elért csúcs, új kezdő csúcsot választunk...etc. (Az előző algoritmus leáll, ha a kezdő csúcsból elérhető csúcsokat már feltérképeztük!)

A csúcsok állapotát hasonlóan, színekkel jelöljük (fehér: még nem elért csúcs, szürke: elért csúcs, fekete: már elhagyott csúcs).

Mélységi erdőt hozunk létre, a csúcsokhoz ( $v$ ) eközben két időpontot rendelünk: mikor értük el ( $d[v]$ ), illetve mikor hagytuk el ( $f[v]$ ). Nyilvánvalóan  $d[v]$  időpont előtt a csúcs fehér,  $f[v]$  időpont után fekete.

MK( $G$ )

```

1 for minden  $u \in V[G]$  csúcsra
2   do  $szín[u] \leftarrow$  FEHÉR
3    $\pi[u] \leftarrow$  NIL
4  $idő \leftarrow 0$ 
5 for minden  $u \in V[G]$  csúcsra
6   do if  $szín[u] =$  FEHÉR
7     then MK-BEJÁR( $u$ )

```

MK-BEJÁR( $u$ )

```

1  $szín[u] \leftarrow$  SZÜRKE           ▷ Most értük el a fehér  $u$  csúcsot.
2  $d[u] \leftarrow idő \leftarrow idő + 1$ 
3 for minden  $v \in Adj[u]$  csúcsra   ▷ ( $u, v$ ) él vizsgálata.
4   do if  $szín[v] =$  FEHÉR
5     then  $\pi[v] \leftarrow u$ 
6           MK-BEJÁR( $v$ )
7  $szín[u] \leftarrow$  FEKETE           ▷  $u$  fekete, mert elhagytuk.
8  $f[u] \leftarrow idő \leftarrow idő + 1$ 

```

Az MK( $G$ ) algoritmus először a graph minden csúcsát fehérre színezi, a szülőjét pedig NIL-re állítja (1–3. sorok). Az idő globalis változó, kezdetben 0. Ezután minden csúcsra – amennyiben még fehér – meghívjuk a másik algoritmust (5–7. sorok).

Az MK-BEJÁR( $u$ ) algoritmus a parameterként kapott csúcsot szürkére színezi (1), hiszen éppen elértük. Az elérés idejét beállítjuk (2), majd az összes szomszédos csúcsot vizsgálni kezdjük (sorban, egymás után). Ha fehér a csúcs, beállítjuk a szülőjét, majd ismét meghívjuk (recursio!) az MK-BEJÁR algoritmust. Itt a fő különbség a szélességi kereséshez képest: ugyan az  $u$  csúcsból elvileg  $x$ -et is egy lépésben elérnénk, de a recursiv hívások miatt előbb kerül sor az  $u-v-y-x$  sorozat végigjárására (és emiatt az  $x$  feketévé alakítására), mint hogy  $u$ -ból egy lépésben elérnénk  $x$ -et.

A 7–8. sorban a vizsgált csúcsot feketére színezzük (közben elhagyjuk), és beállítjuk az elhagyás időpontját is.

Az ábra itt is egy graph bejárására mutat be példát.

