Algoritmusok és adatszerkezetek II. előadásjegyzet:

Elemi gráfalgoritmusok

Ásványi Tibor – asvanyi@inf.elte.hu

2025. október 2.

Tartalomjegyzék

Hivatkozások

[1] Ásványi Tibor, Algoritmusok és adatszerkezetek II.
Útmutatások a tanuláshoz, jelölések, tematika,
fák, gráfok,
mintaillesztés, tömörítés
http://aszt.inf.elte.hu/~asvanyi/ad/ad2jegyzet/

[2] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.,
magyarul: Új Algoritmusok, *Scolar Kiadó*, Budapest, 2003.
ISBN 963 9193 90 9
angolul: Introduction to Algorithms (Third Edititon),
The MIT Press, 2009.

[3] Fekete István, Algoritmusok jegyzet
http://ifekete.web.elte.hu/

[4] Rónyai Lajos – Ivanyos Gábor – Szabó Réka, Algoritmusok,
TypoTEX Kiadó, 1999. ISBN 963 9132 16 0
https://www.tankonyvtar.hu/hu/tartalom/tamop425/2011-0001-526_ronyai_algoritmusok/adatok.html

[5] Tarjan, Robert Endre, Data Structures and Network Algorithms,
CBMS-NSF Regional Conference Series in Applied Mathematics, 1987.

[6] Weiss, Mark Allen, Data Structures and Algorithm Analysis,
Addison-Wesley, 1995, 1997, 2007, 2012, 2013.

[7] Ásványi Tibor, Algoritmusok és adatszerkezetek I. előadásjegyzet (2021)
http://aszt.inf.elte.hu/~asvanyi/ad/ad1jegyzet/ad1jegyzet.pdf

1. Egyszerű gráfok és ábrázolásaik ([2] 22)

Gráfok segítségével pl. hálózatokat, folyamatokat modellezhetünk. A modelleket természetesen ábrázolnunk kell tudni a számítógépben, illetve matematikailag; vagy éppen szemléletesen, a jobb megértés céljából.

1.1. Gráfelméleti alapfogalmak

1.1. Definíció. Gráf *alatt egy* $G = (V, E)$ *rendezett párost értünk, ahol* V *a csúcsok (vertices) tetszőleges, véges halmaza,* $E \subseteq V \times V \setminus \{(u, u) : u \in V\}$ *pedig az élek (edges) halmaza. Ha* $V = \{\}$*, akkor üres gráfról, ha* $V \neq \{\}$*, akkor nemüres gráfról beszélünk.*

Tehát már a definíció szintjén kizárjuk a gráfokból párhuzamos éleket és a hurokéleket. Nincs ugyanis semmilyen eszközünk arra, hogy két (u, v) élet megkülönböztessünk (párhuzamos élek), az (u, u) alakú, ún. hurokéleket pedig expliciten kizártuk.

Így a továbbiakban gráf alatt tulajdonképpen *egyszerű gráfot* értünk.

1.2. Definíció. *A* $G = (V, E)$ *gráf* irányítatlan*, ha tetszőleges* $(u, v) \in E$ *élre* $(u, v) = (v, u)$*.*

1.3. Definíció. *A* $G = (V, E)$ *gráf* irányított*, ha tetszőleges* $(u, v), (v, u) \in E$ *élpárra* $(u, v) \neq (v, u)$*. Ilyenkor azt mondjuk, hogy az* (u, v) *él* fordítottja *a* (v, u) *él, és viszont.*

Mint látható, az irányítatlan gráfoknál tetszőleges (u, v) éllel együtt (v, u) is a gráf éle, hiszen ez a két él egyenlő.

Irányított gráfoknál általában – de nem szükségszerűen – lesz a gráfnak olyan (u, v) éle, hogy ennek fordítottja, (v, u) nem éle a gráfnak.

1.4. Definíció. *A* $G = (V, E)$ *gráf csúcsainak* (V) *egy* $\langle u_0, u_1, \ldots u_n \rangle$ *(*$n \in \mathbb{N}$*) sorozata* a gráf egy útja*, ha tetszőleges* $i \in 1..n$*-re* $(u_{i-1}, u_i) \in E$*. Ezek az* (u_{i-1}, u_i) *élek az út élei. Az* út hossza *ilyenkor* n*, azaz az utat alkotó élek számával egyenlő.*

1.5. Definíció. *Tetszőleges* $\langle u_0, u_1, \cdots, u_n \rangle$ *út* rész-útja $0 \leq i \leq j \leq n$ *esetén az* $\langle u_i, u_{i+1}, \cdots, u_j \rangle$ *út.*

A kör *olyan út, aminek kezdő és végpontja (csúcsa) azonos, a hossza* > 0*, és az élei páronként különbözőek.*

Az egyszerű kör *olyan kör, aminek csak a kezdő és a végpontja azonos.*

Tetszőleges út *akkor* tartalmaz kört*, ha van olyan rész-útja, ami kör.*

Körmentes út *alatt olyan utat értünk, ami nem tartalmaz kört.*

Körmentes gráf *alatt olyan gráfot értünk, amiben csak körmentes utak vannak.*

Vegyük észre, hogy (az „Algoritmusok" témakörben elterjedt szokásnak megfelelően) az utak köröket is tartalmazhatnak! A fentiek szerint tetszőleges kör hossza ≥ 2.

1.6. Definíció. DAG *alatt irányított, körmentes gráfot értünk (directed acyclic graph).*

A DAG-ok modelezhetnek például összetett folyamatokat, ahol a gráf csúcsai elemi műveletek, az élei pedig az ezek közötti rákövetkezési kényszerek.

1.7. Definíció. *Tetszőleges* $G = (V, E)$ irányított gráf irányítatlan megfelelője *az a* $G' = (V, E')$ *irányítatlan gráf, amire*
$E' = \{(u, v) : (u, v) \in E \vee (v, u) \in E\}$.

1.8. Definíció. *A* G irányítatlan gráf összefüggő, *ha* G *tetszőleges csúcsából bármelyik csúcsába vezet út.*

A G irányított gráf összefüggő, *ha az irányítatlan megfelelője összefüggő.*

1.9. Definíció. *Az irányítatlan, körmentes, összefüggő gráfokat* szabad fáknak, *más néven* irányítatlan fáknak *nevezzük.*

1.10. Definíció. *Az* u *csúcs a* G *irányított gráf* generátor csúcsa, *ha* u*-ból a* G *tetszőleges* v *csúcsa elérhető, azaz létezik* $u \rightsquigarrow v$ *út.*

1.11. Tulajdonság. *Ha a* G *irányított gráfnak van generátor csúcsa, akkor összefüggő, de fordítva nem igaz az állítás.*

1.12. Definíció. T gyökeres fa, *más néven* irányított fa, *ha* T *olyan irányított gráf, aminek van generátor csúcsa, nincs olyan éle, aminek a fordítottja is éle a gráfnak, és a* T *irányítatlan megfelelője körmentes. Ilyenkor a generátor csúcsot* a fa gyökér csúcsának *is nevezzük.*

1.13. Tulajdonság. *Tetszőleges (gyökeres vagy szabad) nemüres fának pontosan eggyel kevesebb éle van, mint ahány csúcsa.*

1.14. Definíció. *A* $G = (V, E)$ *gráfnak* részgráfja *a* $G' = (V', E')$ *gráf, ha* $V' \subseteq V \wedge E' \subseteq E$*, és mindkét gráf irányított, vagy mindkettő irányítatlan.*

A G *gráfnak* valódi részgráfja *a* G' *gráf, ha* G*-nek részgráfja* G'*, de* $G \neq G'$ *és* G' *nemüres.*

1.15. Definíció. *Két (rész)gráf* diszjunkt, *ha nincs közös csúcsuk (és ebből következően közös élük sem).*

1.16. Definíció. *A G gráf* összefüggő komponense *a G' gráf, ha G-nek részgráfja G' és G' összefüggő, de G-nek nincs olyan összefüggő részgráfja, aminek G' valódi részgráfja.*

1.17. Tulajdonság. *Tetszőleges gráf vagy összefüggő, vagy felbontható (egymástól diszjunkt) összefüggő komponensekre (amelyek együtt kiadják a teljes gráfot).*

1.18. Definíció. *A G gráf erdő, ha összefüggő komponensei fák (vagy egyetlen fából áll).*

1.19. Tulajdonság. *A G irányítatlan gráf erdő $\iff$ G körmentes.*

A G irányított gráf erdő $\iff$ G irányítatlan megfelelője körmentes, és G mindegyik összefüggő komponensének van generátor csúcsa.

1.2. Bevezetés a gráfábrázolásokhoz

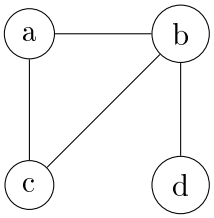
A gráfábrázolásoknál a $G = (V, E)$ gráfról általában föltesszük, hogy $V = \{v_1, \cdots, v_n\}$, ahol $n \in \mathbb{N}$, azaz hogy a gráf csúcsait egyértelműen azonosítják az $1..n$ sorszámok.

Grafikus és *szöveges* reprezentációban (ld. alább) a csúcsok sorszámait a szemléletesség kedvéért gyakran az angol ábécé kisbetűivel jelöljük. Ilyenkor például $a = 1, b = 2, \cdots, z = 26$ lehet. (Szemléltető ábráinkhoz ez bőven elég lesz.)

1.3. Grafikus ábrázolás

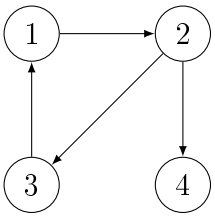
A gráfoknál megszokott módon a csúcsokat kis körök jelölik, az éleket irányított gráfoknál a körök közti nyilak, irányítatlan esetben a köröket összekötő vonalak reprezentálják. A csúcsok sorszámát (illetve az azt reprezentáló betűt) általában a körökbe írjuk.
Az 1. ábrán egy egyszerű, irányítatlan gráfot láthatunk, grafikus és szöveges reprezentációban, a csúcsokat kisbetűkkel azonosítva.

A 2. ábrán pedig egy hasonló, irányított gráfot találunk, újra grafikus és szöveges reprezentációban is, a csúcsokat 1-től 4-ig sorszámozva. (Irányítatlan gráfoknál is sorszámozhatjuk a csúcsokat és irányított gráfoknál ugyanúgy használhatunk betű azonosítókat is.)



a − b ; c.
b − c ; d.

1. ábra. Ugyanaz az irányítatlan gráf grafikus (balra) és szöveges (jobbra) ábrázolással.



$1 \to 2.$
$2 \to 3$; 4.
$3 \to 1.$

2. ábra. Ugyanaz az irányított gráf grafikus (balra) és szöveges (jobbra) ábrázolással.

1.4. Szöveges ábrázolás

Az irányítatalan gráfoknál „$u - v_{u_1}; \cdots ; v_{u_k}$." azt jelenti, hogy a gráfban az u csúcsnak szomszédai a $v_{u_1}, \cdots , v_{u_k}$ csúcsok, azaz $(u, v_{u_1}), \cdots , (u, v_{u_k})$ élei a gráfnak. (Ld. az 1. ábrát!)

Az irányított gráfoknál pedig „$u \to v_{u_1}; \cdots ; v_{u_k}$." azt jelenti, hogy a gráfban az u csúcsból az $(u, v_{u_1}), \cdots , (u, v_{u_k})$ irányított élek indulnak ki, azaz az u csúcs közvetlen rákövetkezői, vagy más néven gyerekei a $v_{u_1}, \cdots , v_{u_k}$ csúcsok. (Ld. a 2. ábrát!)

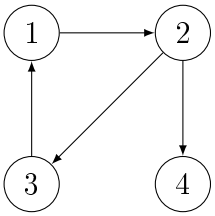
1.5. Szomszédossági mátrixos (adjacency matrix), más néven csúcsmátrixos reprezentáció

A szomszédossági mátrixos, vagy más néven csúcsmátrixos ábrázolásnál a $G = (V, E)$ gráfot ($V = \{v_1, \cdots , v_n\}$) egy $A/1 : bit[n, n]$ mátrix reprezentálja, ahol $n = |V|$ a csúcsok száma, $1..n$ a csúcsok sorszámai, azaz azonosító indexei,
type *bit* **is** $\{0, 1\}$; és tetszőleges $i, j \in 1..n$ csúcssorszámokra

$$A[i, j] = 1 \iff (v_i, v_j) \in E$$

$$A[i, j] = 0 \iff (v_i, v_j) \notin E.$$

A 3. ábrán látható irányított gráfot például a mellette lévő bitmátrix reprezentálja.



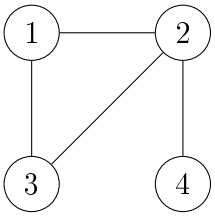
A	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	1	0	0	0
4	0	0	0	0

3. ábra. Ugyanaz az irányított gráf grafikus (balra) és szomszédossági mátrixos (jobbra) ábrázolással.

A főátlóban mindig nullák vannak, mert csak egyszerű gráfokkal foglalkozunk (amelyekben nincsenek hurokélek).

Vegyük észre, hogy irányítatlan esetben a szomszédossági mátrixos reprezentáció mindig szimmetrikus, hiszen $(v_i, v_j) \in E \iff (v_j, v_i) \in E$.

Az 1. ábráról már ismerős irányítatlan gráf csúcsmátrixos ábrázolása a szokásos $a = 1, b = 2, c = 3, d = 4$ megfeleltetéssel a 4. ábrán látható.



A	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	0
4	0	1	0	0

4. ábra. Ugyanaz az irányítatlan gráf grafikus (balra) és szomszédossági mátrixos (jobbra) ábrázolással.

Az irányítatlan gráfoknál tehát tetszőleges v_i és v_j csúcsokra $A[i, j] = A[j, i]$, valamint $A[i, i] = 0$. Elég azért az alsóháromszög (vagy a felsőháromszög) mátrixot ábrázolnunk, a főátló nélkül, pl. sorfolytonosan. A ténylegesen ábrázolt alsóháromszög mátrix így egy elemet tartalmaz a 2. sorban, kettőt a 3. sorban, ... $(n-1)$ elemet az utolsó sorban, ami

$$n^2 \text{ bit helyett csak } 1 + 2 + \cdots + (n-1) = n * (n-1)/2 \text{ bit.}$$

Az A mátrix helyett tehát felvehetünk pl. egy $B : bit[n * (n-1)/2]$ tömböt, ami az $a_{ij} = A[i, j]$ jelöléssel az

$$\langle a_{21}, a_{31}, a_{32}, a_{41}, a_{42}, a_{43}, \ldots, a_{n1}, \ldots, a_{n(n-1)} \rangle$$

absztrakt sorozatot reprezentálja sorfolytonosan. Innét

$A[i,j] = B[(i-1)*(i-2)/2+(j-1)]$ ha $i > j$ (az alsóháromszög mátrixban)
$A[i,j] = A[j,i]$ ha $i < j$ ($A[i,j]$ a felsőháromszög mátrixban)
$A[i,i] = 0$. ($A[i,i]$ a főátlón van)

Ha ui. meg szeretnénk határozni tetszőleges $a_{ij} = A[i,j]$, az alsóháromszög mátrixban található elem helyét a ténylegesen is létező B tömbben, akkor azt kell megszámolnunk, hogy hány elem előzi meg sorfolytonosan az a_{ij} elemet a B tömbben. Mivel a B tömböt nullától indexeljük, a_{ij} indexe a B-ben egyenlő lesz az a_{ij}-t megelőző elemek számával. Az alsóháromszög mátrixban az a_{ij} elemet az alábbi elemek előzik meg sorfolytonosan:

a_{21}
a_{31}, a_{32}
a_{41}, a_{42}, a_{43}
$\vdots$
$a_{(i-1)1}, a_{(i-1)2}, \cdots, a_{(i-1)(i-2)}$
$a_{i1}, a_{i2}, \cdots, a_{i(j-1)}$

Ez pedig összesen $(1+2+3+\cdots+(i-2))+(j-1) = (i-1)*(i-2)/2+(j-1)$ elem.

A csúcsmátrixos (más néven szomszédossági mátrixos) ábrázolásnál $\Theta(1)$ idő alatt eldönthető a $(v_i, v_j) \in E$ kérdés, így olyan algoritmusoknál előnyös, ahol gyakori ez a művelet.

Adott csúcs (irányított gráfoknál) gyerekeinek, vagy (irányítatlan gráfoknál) szomszédainak felsorolásához viszont n lépésre van szükségünk, ami általában lényegesen több, mint ahány gyerek vagy szomszéd ténylegesen van.

1.6. Szomszédossági listás (adjacency list) reprezentáció

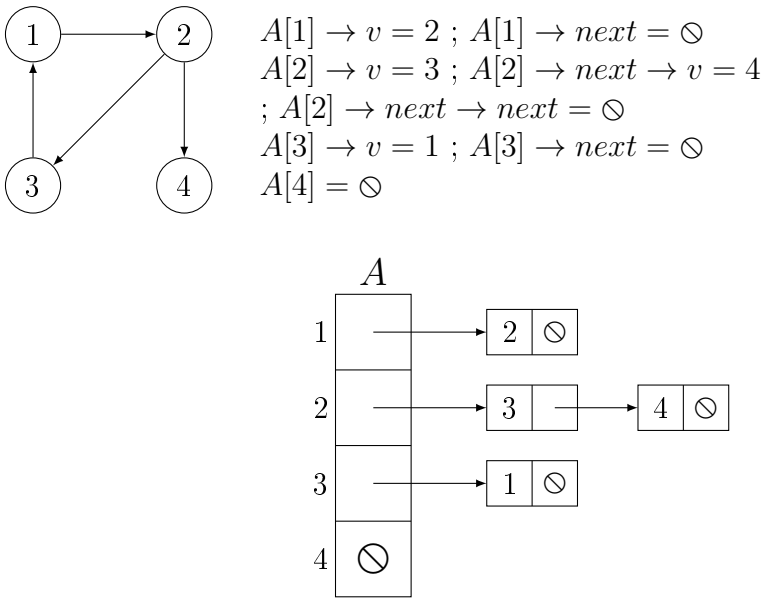
A szomszédossági listás ábrázolás hasonlít a szöveges reprezentációhoz. A $G = (V, E)$ gráfot ($V = \{v_1, \cdots, v_n\}$) az $A/1 : Edge^*[n]$ pointertömb

Edge
$+v : \mathbb{N}$
$+next : Edge*$

segítségével ábrázoljuk, ahol irányítatlan gráf esetében a v_i csúcs szomszédainak sorszámait az $A[i]$ S1L tartalmazza ($i \in 1..n$). A v_i csúcs szomszédainak indexeit tehát az $A[i]$ lista elemeinek v adattagjai tartalmazzák. Így az $A[i]$

lista elemei a v_i csúcshoz kapcsolódó éleknek felelnek meg. Irányítatlan gráfok esetén azért minden élet kétszer ábrázolunk, hiszen ha pl. a v_i csúcsnak szomszédja v_j, akkor a v_j csúcsnak is szomszédja v_i.

Irányított gráfok esetén hasonló a reprezentáció, de az $A[i]$ S1L csak az v_i csúcs gyerekeinek (más néven közvetlen rákövetkezőinek) sorszámait tartalmazza ($i \in 1..n$). Ilyen módon mindegyik élet csak egyszer kell ábrázolnunk. Egy példát láthatunk az 5. ábrán.



5. ábra. Ugyanaz az irányított gráf grafikus (balra) és szomszédossági listás (jobbra) ábrázolással.

A szomszédossági listás ábrázolásnál S1L-ek helyett természetesen másfajta listákat is alkalmazhatunk.

A szomszédossági listás ábrázolásnál a $(v_i, v_j) \in E$ kérdés eldöntéséhez meg kell keresnünk a j indexet az $A[i]$ listán, így olyan algoritmusoknál, ahol gyakori ez a művelet, lehet, hogy érdemes inkább a csúcsmátrixos reprezentációt választani.

Adott csúcs (irányított gráfoknál) gyerekeinek, vagy (irányítatlan gráfoknál) szomszédainak felsorolásához viszont pontosan annyi lépésre van szükségünk, mint ahány gyerek vagy szomszéd ténylegesen van. Mivel ebben a jegyzetben a legtöbb gráfalgoritmusnak ez a leggyakoribb művelete, ezt a reprezentációt gyakran előnyben részesítjük a csúcsmátrixos ábrázolással szemben.

1.7. Számítógépes gráfábrázolások tárigénye

A továbbiakban a $G = (V, E)$ gráf csúcsainak számát $n = |V|$, éleinek számát pedig $m = |E|$ fogja jelölni. Világos, hogy $0 \leq m \leq n * (n-1) \leq n^2$, így $m \in O(n^2)$.

A *ritka gráf*okat az $m \in O(n)$, míg a *sűrű gráf*okat az $m \in \Theta(n^2)$ összefüggéssel jellemezzük.

1.7.1. Szomszédossági mátrixok

A szomszédossági mátrixos (más néven csúcsmátrixos) ábrázolás tárigénye alapesetben n^2 bit. Irányítatlan gráfoknál, csak az alsóháromszög mátrixot tárolva, $n * (n-1)/2$ bit. Mivel $n * (n-1)/2 \in \Theta(n^2)$, az aszimptotikus tárigény mindkét esetben $\Theta(n^2)$.

1.7.2. Szomszédossági listák

Szomszédossági listás reprezentációnál a pointertömb n db mutatóból áll, a szomszédossági listáknak pedig összesen m vagy $2 * m$ elemük van, aszerint, hogy a gráf irányított vagy irányítatlan. Ezért az aszimptotikus tárigény mindkét esetben $\Theta(n + m)$.

Ritka gráfoknál (amik a gyakorlati alkalmazások többségénél sűrűn fordulnak elő) $m \in O(n)$ miatt $\Theta(n + m) = \Theta(n)$, ami azt jelenti, hogy ritka gráfokra a szomszédossági listás reprezentáció tárigénye aszimptotikusan kisebb, mint a csúcsmátrixosé.

Sűrű gráfoknál viszont $m \in \Theta(n^2)$ miatt $\Theta(n + m) = \Theta(n^2)$, azaz szomszédossági listás és a csúcsmátrixos reprezentáció tárigénye aszimptotikusan ekvivalens.

Teljes gráfoknál a szomszédossági listáknak összesen $n*(n-1)$ elemük van, és egy-egy listaelem sok bitből áll. Teljes vagy közel teljes gráfoknál tehát a szomszédossági listás ábrázolás tényleges tárigénye jelentősen nagyobb lehet, mint a csúcsmátrixosé, ahol a mátrix egy-egy eleme akár egyetlen biten is elfér.

2. Az absztrakt *halmaz*, az absztrakt *sorozat* és az absztrakt *gráf* típus

A halmazok és sorozatok leírásához bevezetünk két típuskonstruktort: Ha $\mathcal{T}$ tetszőleges típus, akkor
- $\mathcal{T}\{\}$ jelöli $\mathcal{T}$ típusú elemek tetszőleges, véges halmazát, és
- $\mathcal{T}\langle\rangle$ jelöli $\mathcal{T}$ típusú elemek tetszőleges, véges sorozatát.

A halmazokra a matematikában szokásos halmazműveleteken kívül még értelmezzük az u **from** S műveletet, ahol S tetszőleges nemüres halmaz. Ennek hatására kiválasztjuk az S halmaz egy tetszőleges elemét, u-nak értékül adjuk, majd eltávolítjuk S-ből. Az üres halmazt önmagában álló $\{\}$ jelöli.

A sorozatokat a matematikában szokásos módon, egytől indexeljük, és az indexet alsó indexként jelöljük, továbbá,
- ha $u, v : \mathcal{T}\langle\rangle$, akkor az $u + v$ kifejezés a konkatenáljukat jelöli;
- $\langle\rangle$ önmagában az üres sorozat.

Mint általában a strukturált típusú változókat, a halmaz és a sorozat típusú változókat is deklarálni kell. Feltesszük, hogy az $s : \mathcal{T}\langle\rangle$ deklaráció hatására s üres sorozattal inicializálódik, illetve a $h : \mathcal{T}\{\}$ deklaráció hatására h üres halmazzal inicializálódik. Ha a zárójelek között megadunk – pontosvesszőkkel elválasztva – néhány $\mathcal{T}$ típusú elemet, akkor a halmaz illetve sorozat a deklaráció kiértékelődése után ezeket fogja tartalmazni.

A gráfok absztrakt algoritmusainak leírásához bevezetjük a $\mathcal{V}$ (vertex, azaz csúcs) absztrakt típust. A $\mathcal{V}$ lesz a gráfok csúcsainak absztrakt típusa. Ez egy olyan elemi típus, amelyben mindegyik csúcshoz tetszőlegesen sok, névvel jelölt címke társítható, és mindegyik címkéhez tartozik valamilyen érték.

Ezek az értékkel bíró címkék tulajdonképpen a csúcsokon értelmezett parciális függvények, amelyek az absztrakt algoritmusokban közönséges értékadó utasításokkal hozhatók létre, és ezekkel is módosíthatók. Ha már léteznek, akkor hatáskörük és láthatóságuk is a teljes absztrakt program, és az absztrakt algoritmus futásának végéig élnek.

A v csúcshoz tartozó *name* címkeértéket $name(v)$ jelöli. Ha tehát végrehajtjuk a $name(v) := x$ értékadást, akkor a v csúcs *name* címkéjének értéke x lesz. Másképp fogalmazva, a *name* címke (azaz a $\mathcal{V}$ halmazon értelmezett parciális függvény) a $name(v) := x$ értékadással a v csúcsnál az x értéket veszi fel.

A $\mathcal{V}$ halmazt az algoritmusok implementációiban legtöbbször az $\mathbb{N}$ halmaz reprezentálja, egy n csúcsú gráf csúcsait pedig egyszerűen az $1..n$ vagy a $0..(n-1)$ halmaz, attól függően, hogy a tömböket egytől vagy nullától kezdve indexeljük. A címkéket ui. gyakran tömbök reprezentálják. Emiatt viszont a láthatóságuk, a hatáskörük és az élettertamuk is korlátozott. Az ebből fakadó problémákat az absztrakt algoritmus implementálásakor kell megoldani.

Az értékkel bíró címkék mellett használni fogunk egyszerű címkéket is, amikor a gráfok csúcsaihoz és/vagy éleihez egyszerű számértékeket vagy neveket társítunk.

Most már minden készen áll az élek ($\mathcal{E}$) és élsúlyozatlan absztrakt gráfok ($\mathcal{G}$) leírásához. Gyakorlati megfontolásból elegendő az egyszerű gráfokra szorítkoznunk, amelyek nem tartalmaznak sem párhuzamos, sem hurokéleket.

$\mathcal{E}$
$+\ u, v : \mathcal{V}$

$\mathcal{G}$
$+\ V : \mathcal{V}\{\}$
$+\ E : \mathcal{E}\{\}$ // $E \subseteq V \times V \setminus \{(u,u) : u \in V\}$ // edges
$+\ A : V \to 2^V$ // $A(u) = \{\, v \in V \mid (u,v) \in E \,\}$
// $A(u) =$ the adjacent vertices of vertex u.

Az egyes gráfalgoritmusoknál a gráf-objektumoknak tipikusan vagy csak az E, vagy csak az A attributumára fogunk hivatkozni. Az A hivatkozások esetén a szomszédossági listás gráfreprezentáció az alapértelmezett (ami azért adott esetben megváltoztatható), míg az E hivatkozások esetében ez további megfontolás tárgya.

3. Elemi gráfalgoritmusok ([2] 22)

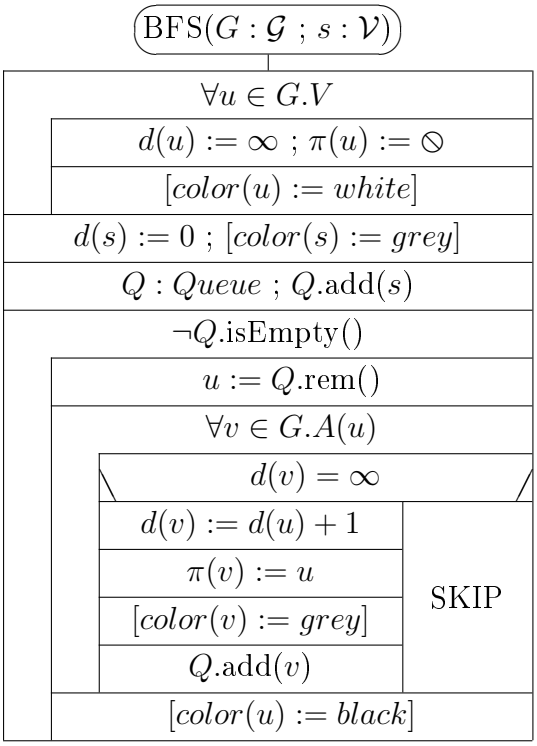
Elemi gráfalgoritmusok alatt élsúlyozatlan gráfokon értelmezett algoritmusokat értünk. Élsúlyozatlan gráfokban tetszőleges út hossza egyszerűen az út mentén érintett élek száma. Az *algoritmusok* témakörben szokásos fogalmak szerint az út tartalmazhat kört. Ha a gráfban tetszőleges u és v csúcsokra az (u, v) élt megkülönböztetjük a (v, u) éltől, *irányított gráfról* beszélünk. Az *irányítatlan gráfokban* viszont ezeket definíció szerint egyenlőknek tekintjük.

Ebben a fejezetben a két alapalgoritmust és ezek néhány alkalmazását tárgyaljuk.

3.1. A szélességi keresés (BFS: Breadth-first Search)

Ezt az algoritmust irányított és irányítatlan gráfokra is értelmezzük. Meghatározzuk a start csúcsból (s) a gráf minden, s-ből elérhető csúcsába a legkevesebb élet tartalmazó utat (ha több ilyen van, akkor az egyiket). Élsúlyozatlan gráfokban ezeket tekintjük az s-ből induló *legrövidebb*, vagy más néven *optimális* utaknak. A csúcsok fontosabb címkéi:
d – a megtalált úttal hány élen keresztül jutunk a csúcsba, és
π – melyik csúcsból jutunk közvetlenül a csúcsba (ki a szülője).
Ha az u csúcs s-ből nem érhető el, akkor $d(u) = \infty$ és $\pi(u) = \oslash$ lesz. Ezenkívül $\pi(s) = \oslash$ is igaz lesz, ami azt jelöli, hogy a legrövidebb $s \rightsquigarrow s$ út csak az s csúcsot tartalmazza, nincs benne él, és ezért s-nek szülője sincs.

Használni fogunk még egy *color* nevű címkét is, aminek az értéke nem befolyásolja a program futását, ezért a rá vonatkozó utasítások az algoritmusból elhagyhatók; csak szemléletes tartalmuk van:
- a fehér csúcsokat a gráfbejárás/keresés még nem találta meg;
- a szürke csúcsokat már megtalálta, de még nem dolgozta fel;
- a fekete csúcsokkal viszont már nincs további tennivalója.



Az első ciklust végrahajtva a gráf minden csúcsára az $d(u) = \infty$ (ami azt jelenti, hogy még egyik csúcsot sem értük el), és ennek megfelelően $\pi(u) = \oslash$ lesz[1].

Az s start (más néven source, azaz forrás) csúcsra azonban tudjuk, hogy $d(s) = 0$ az optimális $s \rightsquigarrow s$ út hossza[2]. Azok a csúcsok, amiket már elértünk, de még a gyerekeiket nem néztük meg, a Q sorba kerülnek, így most az elején s is.

A második, azaz a fő ciklus tehát addig fut, amíg van már elért, de még fel nem dolgozott csúcs. Ez először az $u = s$ csúcsot veszi ki, és mindegyik gyerekére beállítja, hogy s-ből egy lépésben (azaz egyetlen élen keresztül) elérhető, a szülője s,[3] és bekerülnek a sorba.[4]

Most a sorban azok az u csúcsok vannak, akik s-ből egy lépésben elérhetők. A fő ciklus ezeket egyesével kiveszi, megtalálja azokat a v csúcsokat, akik s-ből minimum két lépésben (azaz két élen keresztül) érhetők el (hiszen ezek azoknak a gyerekei, akik egy lépésben elérhetők), beállítja a $d(v) = 2$ és

[1] [valamint $color(u) = white$]

[2] [és ezzel el is kezdtük az s feldolgozását, így $color(s) = grey$ lett]

[3] [szürkére színezi őket]

[4] [Végül s-et feketére színezi, mert már befejezte ezt a csúcsot.]

$\pi(v) = u$ értékeket a szülőjüknek megfelelően,[5] és bekerülnek a sor végére[6]. Ha valamelyik v gyerekcsúcsra az (u, v) él feldolgozásakor már $d(v) \neq \infty$, akkor ez a csúcs már korábbról ismert[7], így $d(v) \in \{0, 1, 2\}$, ezért az újonnan v-be talált út ennél biztosan nem rövidebb, és a BFS algoritmus ennek megfelelően figyelmen kívül is hagyja (ld. az elágazást).

Mire a BFS az összes, s-ből egy lépésben elérhető, azaz $d = 1$ távolságra lévő csúcsot kiveszi a sorból és *kiterjeszti*, azaz a belőle kimenő éleket is feldolgozza, a sorban az s-ből minimum két lépésben elérhető, azaz $d = 2$ távolságra lévő csúcsok maradnak. Miközben ezeket dolgozza fel, azaz kiveszi a sorból és kiterjeszti őket, az s-től $d = 3$ távolságra lévő csúcsok kerülnek a sor végére. Ennélfogva, mire a BFS a feldolgozza az s-től $d = 2$ távolságra levő csúcsokat, a sorban éppen a $d = 3$ távolságra lévők maradnak, és így tovább.

Általában, mikor a sort már az s-től k távolságra levő csúcsok alkotják, megkezdődik azoknak a feldolgozása. Közben az s-től $d = k + 1$ távolságra lévő csúcsokat találjuk meg, beállítjuk a címkéiket és a sor végére tesszük őket. Mire az s-től k távolságra levő csúcsokat feldolgozzuk, a sort már az s-től $k + 1$ távolságra levő csúcsok alkotják stb.

Azt mondjuk, hogy az s-től k távolságra levő csúcsok vannak a gráf k-adik szintjén. Így a BFS a gráfot szintenként járja be, először a nulladik szintet, aztán az elsőt, majd a másodikat stb. Minden szintet teljesen feldolgoz, mielőtt a következőre lépne, közben pedig éppen a következő szinten levő csúcsokat találja meg. Innét jönnek a *szélességi bejárás* és a *szélességi keresés* elnevezések.

Mivel a gráf véges, végül nem lesz már $k + 1$ távolságra levő csúcs, Q kiürül és a BFS megáll. Azokat a csúcsok, amelyek s-ből elérhetők, az algoritmus valamelyik szinten meg is találja, és megfelelően beállítja a d és a π címkéiket is. A többi csúcs tehát s-ből nem érhető el. Ezekre $d = \infty$ és $\pi = \oslash$ marad.[8]

3.1. Feladat. *A BFS algoritmusában milyen, az elágazás „$d(v) = \infty$" feltételével ekvivalens feltételt tudnánk adni? Miért?*

3.1.1. A szélességi fa (Breadth-first Tree)

A BFS minden s-ből elérhető, de tőle különböző csúcsra, annak π címkéjével hivatkozik annak szülőjére, az s-ből a csúcsba vezető (a BFS által megha-

[5] [szürkére színezi őket]

[6] [majd a szülőjüket feketére színezi, mert azt már befejezte]

[7] [már nem fehér, hanem szürke vagy fekete]

[8] [Az s-ből elérhető csúcsok tehát végül feketék lesznek, míg a többiek fehérek maradnak.]

tározott) legrövidebb úton. Természetesen több csúcsnak is lehet ugyanaz a szülője, a szülőcsúcs viszont a BFS által meghatározott legrövidebb utakon egyértelmű.

Az s-ből elérhető csúcsok π hivatkozásai tehát egy általános fát definiálnak, aminek a gyökere s, és ennek megfelelően $\pi(s) = \oslash$. Ezt a fát *szélességi fának* és *legrövidebb utak fájának* is nevezzük, mivel – fordított irányban – mindegyik, az s-ből elérhető csúcsra a BFS által meghatározott legrövidebb utakat tartalmazza.

Világos, hogy ez a fordított ábrázolás azért célszerű, mert minden csúcsnak legfeljebb egy szülője, viszont több gyereke is lehet, így tehát tömörebb ábrázolás érhető el.

3.2. Feladat. *Tegyük fel, hogy a G gráfra már lefutott a szélességi BFS(G, s) algoritmus. Írja meg a printShortestPathTo(v)* rekurzív *eljárást, ami kiírja az s-ből v-be vezető (a BFS által kiszámolt) legrövidebb utat, feltéve, hogy s-ből v elérhető!*

Vegyük észre, hogy az előbbi előfeltétellel nincs szükségünk az s paraméterre! Az algoritmus ne használjon segéd adatszerkezetet!

$MT(d) \in \Theta(d)$, *ahol* $d = d(v)$.

3.3. Feladat. *Tegyük fel, hogy a G gráfra már lefutott a szélességi BFS(G, s) algoritmus. Írja meg a printShortestPathTo(v)* nemrekurzív *eljárást, ami kiírja az s-ből v-be vezető (a BFS által kiszámolt) legrövidebb utat, feltéve, hogy s-ből v elérhető!*

Vegyük észre, hogy az előbbi előfeltétellel nincs szükségünk az s paraméterre! Milyen adatszerkezettel váltotta ki a rekurziót?

$MT(d) \in \Theta(d)$, *ahol* $d = d(v)$.

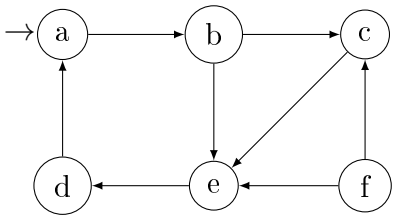
3.4. Feladat. *Tegyük fel, hogy a G gráfra már lefutott a szélességi BFS(G, s) algoritmus. Írja meg a printShortestPath(s, v) eljárást, ami kiírja az s-ből v-be vezető (a BFS által kiszámolt) legrövidebb utat, ha s-ből v elérhető! Különben a kiírás azt adja meg, hogy melyik csúcsból melyik nem érhető el!*

$MT(d) \in \Theta(d)$, *ahol* $d = d(v)$, *ha s-ből v elérhető; különben* $d = 1$.

3.1.2. A szélességi keresés szemléltetése

A 6. ábrán látható gráfra szemléltetjük a BFS működését, az alábbi táblázat segítségével, ahol most „a” a start csúcs. Az új csúcsok természetesen mindig a sor végére kerülnek.

Most is és a későbbiekben is, *indeterminisztikus esetekben* a kisebb indexű csúcsot részesítjük előnyben. (Ez a leggyengébb szabály, de a zárthelyiken,

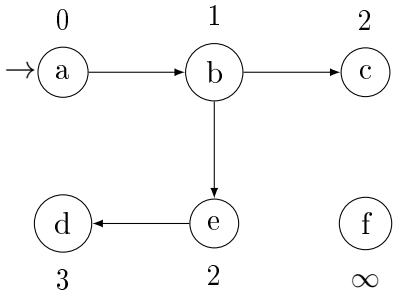


a → b.
b → c ; e.
c → e.
d → a.
e → d.
f → c ; e.

6. ábra. Ugyanaz az irányított gráf grafikus (balra) és szöveges (jobbra) ábrázolással ($s = \mathrm{a}$).

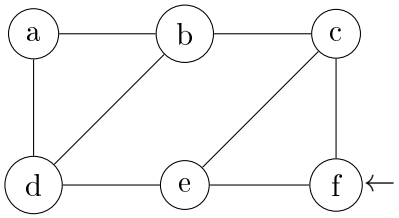
vizsgákon is elvárjuk, hogy tartsák be.) Alább pl. ez akkor érvényesül, amikor a „b” csúcs gyerekeit „c,e” sorrendben tesszük be a sorba.

ex-	changes of d and π						Q :
pand	a	b	c	d	e	f	Queue
$d\,\mathcal{V}$	$0\,\oslash$	$\infty\,\oslash$	$\infty\,\oslash$	$\infty\,\oslash$	$\infty\,\oslash$	$\infty\,\oslash$	$\langle$ a $\rangle$
0 a		1 a					$\langle$ b $\rangle$
1 b			2 b		2 b		$\langle$ c, e $\rangle$
2 c							$\langle$ e $\rangle$
2 e				3 e			$\langle$ d $\rangle$
3 d							$\langle\ \rangle$



7. ábra. A 6. ábráról ismerős gráf szélességi fája, ha $s = \mathrm{a}$.

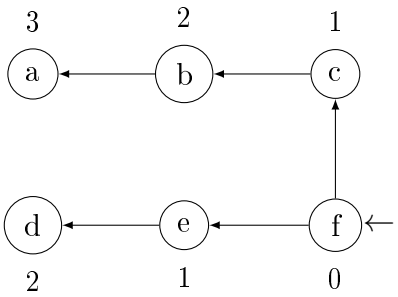
Most pedig a 8. ábrán látható gráfra szemléltetjük a BFS működését, az alábbi táblázat segítségével, ahol most „f” a start csúcs.



a – b ; d.
b – c ; d.
c – e ; f.
d – e.
e – f.

8. ábra. Ugyanaz az irányítatlan gráf grafikus (balra) és szöveges (jobbra) ábrázolással (s = f).

ex-	changes of d and π						Q :
pand	a	b	c	d	e	f	Queue
$d\,\mathcal{V}$	∞ ⊘	∞ ⊘	∞ ⊘	∞ ⊘	∞ ⊘	0 ⊘	⟨ f ⟩
0 f			1 f		1 f		⟨ c, e ⟩
1 c		2 c					⟨ e, b ⟩
1 e				2 e			⟨ b, d ⟩
2 b	3 b						⟨ d, a ⟩
2 d							⟨ a ⟩
3 a							⟨ ⟩



9. ábra. A 8. ábráról ismerős gráf szélességi fája, ha s = f.

3.1.3. A szélességi keresés hatékonysága

A gráfalgoritmusokban a továbbiakban is a szokásos $n = |G.V|$ és $m = |G.E|$ jelöléseket használjuk.

Az első, az inicializáló ciklus n-szer iterál. A második, a fő ciklus annyiszor iterál, ahány csúcs elérhető s-ből (önmagát is számítva), ami legfeljebb szintén n, minimum 1.

Ennek megfelelően, irányított gráfoknál a belső ciklus legfeljebb m-szer iterál összesen (ha s-ből mindegyik csúcs elérhető, akkor minden él sorra kerül), irányítatlan gráfoknál pedig belső ciklus legfeljebb $2m$-szer iterál összesen (ha s-ből mindegyik csúcs elérhető, akkor minden él sorra kerül mindkét irányból). Az is lehetséges viszont, hogy a belső ciklus egyszer sem iterál (ha s-ből nem megy ki egyetlen él sem).

Összefoglalva: $MT(n,m) \in \Theta(n+m)$ és $mT(n,m) \in \Theta(n)$.

3.1.4. A szélességi keresés implementációja szomszédossági listás és szomszédossági mátrixos gráfábrázolás esetén

A $G = (V,E)$ gráfról föltesszük, hogy $V = \{v_1, \cdots, v_n\}$, ahol $n \in \mathbb{N}$, azaz a gráf csúcsait egyértelműen azonosítják az $1..n$ sorszámok. Az absztrakt v_i csúcsokat úgy ábrázoljuk, hogy d és π címkéinek megfeleltetjük a $d/1, \pi/1 : \mathbb{N}[n]$ tömböket, ahol $d(v_i)$ reprezentációja $d[i]$ és $\pi(v_i)$ reprezentációja $\pi[i]$. A *color* címkék reprezentálása felesleges. A $\oslash$ reprezentációja lehet például a 0 számérték: pl. $\pi[s] = 0$ absztrakt jelentése $\pi(v_s) = \oslash$. Mivel a szélességi keresésnél n csúcsú gráfon tetszőleges csúcshoz vezető út hossza legfeljebb $n-1$, ami egyben a véges d-értékek maximuma, a ∞ helyett használhatjuk pl. az n számot.

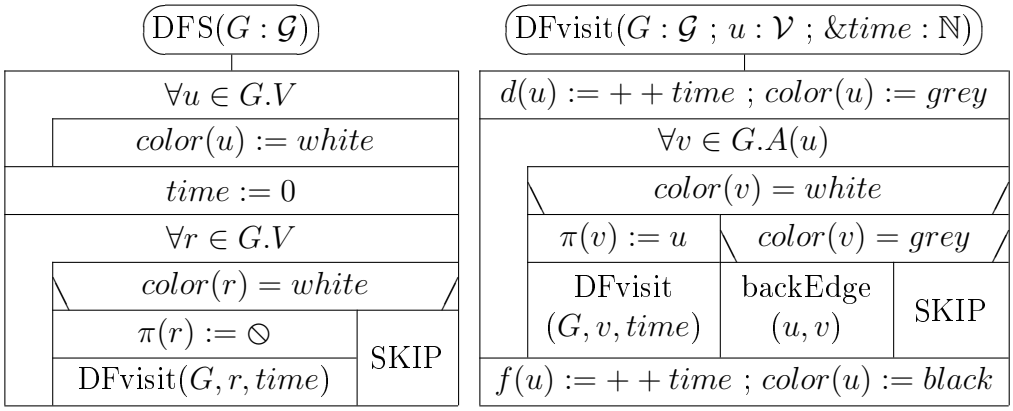
3.5. Feladat. *Írja meg a BFS($A/1 : Edge*$*[n] ; s : 1..n ; d/1, \pi/1 : \mathbb{N}[n]$) eljárást, ami a szélességi keresés implementációja szomszédossági listás (1.6) gráfábrázolás esetére. Ügyeljen arra, hogy az absztrakt algoritmus hatékonyságának aszimptotikus nagyságrendje megmaradjon!*

3.6. Feladat. *Írja meg a BFS($A/1 : bit[n,n]$; $s : 1..n$; $d/1, \pi/1 : \mathbb{N}[n]$) eljárást, ami a szélességi keresés implementációja szomszédossági mátrixos (1.5) gráfábrázolás esetére. Tartható-e az absztrakt algoritmus hatékonyságának aszimptotikus nagyságrendje? Ha nem, hogyan változik?*

3.2. A mélységi keresés (DFS: Depth-first Search)

Mélységi bejárásnak (Depth-first Traversal) vagy mélységi gráfbejárásnak is nevezik, mivel a gráf összes csúcsát és élét érinti.

Ebben a jegyzetben csak egyszerű irányított gráfokra értelmezzük. Fehér csúcs: érintetlen, szürke csúcs: belőle elérhető csúcsokat járunk be éppen, fekete csúcs: befejeztük.



$d(u)$: elérési idő (discovery time) / $f(u)$: befejezési idő (finishing time)

3.2.1. Mélységi feszítő erdő (Depth-first forest)

Mindegyik, a DFS eljárásból indított DFvisit egy-egy *mélységi fát* számol ki. Ezek együtt adják a *mélységi feszítő erdőt.*

$r \in G.V$ egy mélységi fa gyökere $\iff \pi(r) = \oslash$
$(u, v) \in G.E$ egy mélységi fa éle $\iff u = \pi(v)$

3.2.2. Az élek osztályozása (Classification of edges)

3.7. Definíció. *A gráf éleinek osztályozása:*

(u, v) fa-él (tree edge) $\iff$ (u, v) *valamelyik mélységi fa egyik éle. (A fa-élek mentén járjuk be a gráfot.)*

(u, v) vissza-él (back edge) $\iff$ v *az* u *őse egy mélységi fában.*

(u, v) előre-él (forward edge) $\iff$ (u, v) *nem fa-él, de* v *az* u *leszármazottja egy mélységi fában.*

(u, v) kereszt-él (cross edge) $\iff$ u *és* v *két olyan csúcs, amelyek ugyanannak a mélységi fának két különböző ágán vannak, vagy két különböző mélységi fában találhatók.*

3.8. Tétel. *A gráf éleinek felismerése:*
A mélységi bejárás során tetszőleges (u, v) *él feldolgozásakor az él a következő kritériumok alapján osztályozható.*

(u, v) fa-él (tree edge) $\iff$ *a* v *csúcs még fehér.*

(u, v) vissza-él (back edge) $\iff$ *a* v *csúcs éppen szürke.*

(u, v) előre-él (forward edge) $\iff$ *a v csúcs már fekete* $\wedge\ d(u) < d(v)$.

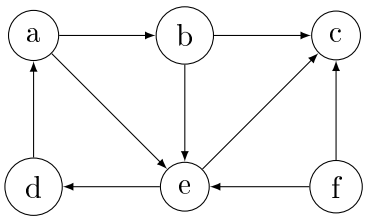
(u, v) kereszt-él (cross edge) $\iff$ *a v csúcs már fekete* $\wedge\ d(u) > d(v)$.

3.9. Feladat. *A mélységi bejárás során miért nem fordulhat elő, hogy az (u, v) él feldolgozásakor $d(u) = d(v)$? Milyen feltételekkel fordulhatna ez elő? Hogyan kellene ekkor kiegészíteni a 3.7. definíciót úgy, hogy a 3.8. tétel megfogalmazásán ne kelljen módosítani?*

3.2.3. A mélységi bejárás szemléltetése

A 10. ábrán látható gráf mélységi bejárását szemléltetjük. A bejárás indeterminisztikus abban, hogy az egyes ciklusokban a csúcsokat milyen sorrendben veszi sorra. Ezt a szemléltetésben úgy fogjuk feloldani, hogy a csúcsokat ábécé rendben, illetve indexek szerint monoton növekvően dolgozzuk fel. (Ennek a konvenciónak a betartása az összes gráfalgoritmus esetében, a zh-kon és a vizsgákon is elvárás.)

A csúcsok felett illetve alatt olvasható d/f alakú címkék a csúcs *elérési/befejezési* idejét (*discovery/finishing* time) mutatják. A fa-éleket dupla szárú nyillal, a nemfa-éleket pedig a megfelelő címkékkel jeleztük: V:visszaél, E:előre-él, K:kereszt-él.

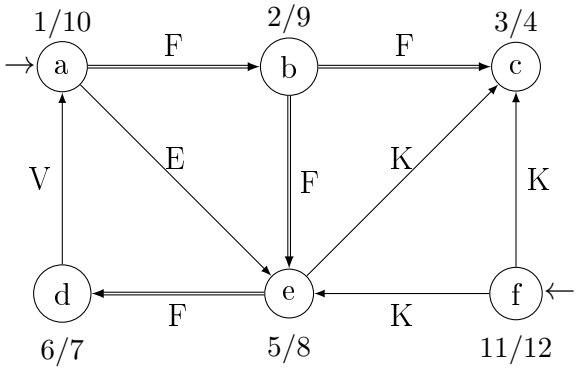


a → b ; e.
b → c ; e.
d → a.
e → c ; d.
f → c ; e.

10. ábra. Ugyanaz az irányított gráf grafikus (balra) és szöveges (jobbra) ábrázolással.

A 11. ábrán követhető a gráf mélységi bejárása. A bejárás eredménye, a mélységi feszítő erdő két mélységi fából áll: az egyiknek az **a** csúcs a gyökere és elérési sorrendben **b, c, e** és **d** a további csúcsai, a másik csak az **f** gyökércsúcsból áll.

A mélységi bejárás (DFS) az első mélységi vizittel (DFvisit) és ennek megfelelően az első mélységi fa építésével kezdődik. [Nemüres gráf mélységi bejárása mindig egy vagy több mélységi vizitből áll, és ennek megfelelő számú mélységi fát épít fel: ezekből áll majd a mélységi feszítő erdő. Ennek fái lefedik az összes csúcsot, és egymástól diszjunktak.]



11. ábra. A 10. ábrán látható gráf mélységi bejárása.

Alfabetikus konvenciónk alapján az első mélységi vizitet és egyben az első mélységi fa építését az **a** csúcsnál, mint gyökércsúcsnál kezdjük $time = 1$ idővel. Az idő számláló mindig akkor lép egyet, amikor elérünk, vagy befejezünk egy csúcsot. Azok a csúcsok fehérek, amelyeket még nem értünk el. Azok szürkék, amelyeket már elértünk, de még nem fejeztünk be. Azok feketék, amelyeket már befejeztünk. Először tehát az összes csúcs fehér volt, de most az **a** csúcsnak beállítottuk az elérési idejét és szürkére színeztük. Az **a** csúcs gyerekei a **b** és az **e** csúcsok. Alfabetikus konvenciónkat követve a **b** felé megyünk tovább. A **b** csúcs még fehér, így az **(a,b)** él fa-él lesz. (Ld. a 3.8. tételt!) Ha kijelölünk egy fa-élt, mindig tovább is lépünk abba a csúcsba, amibe mutat.

Most $time = 2$ idővel elértük a **b** csúcsot, beállítjuk az elérési idejét és szürkére színezzük. Gyerekei a **c** és **e** csúcsok. A **c** csúcs felé folytatjuk a bejárást. Mivel **c** még fehér, **(b,c)** fa-él lesz.

Most $time = 3$ idővel elértük a **c** csúcsot, beállítjuk az elérési idejét és szürkére színezzük. Mivel nincs gyereke, $time = 4$ idővel befejezzük, beállítjuk a befejezési idejét és feketére színezzük, majd visszalépünk a bele mutató fa-élen az éppen épülő mélységi fában a szülőjébe, ami a **b** csúcs.

A **b** csúcsnak van még egy címkézetlen kimenő éle, a **(b,e)**. Ez fehér csúcsba mutat, így **(b,e)** fa-él lesz.

Most $time = 5$ idővel elértük az **e** csúcsot, beállítjuk az elérési idejét és szürkére színezzük. Két kimenő élünk van, az **(e,c)** és az **(e,d)**. Először az **(e,c)** élet dolgozzuk fel. Ez fekete (azaz befejezett) csúcsba mutat, aminek az elérési ideje kisebb, mint az **e** csúcsé, így **(e,c)** élet kereszt-élnek címkézzük. (Ld. a 3.8. tételt!) Ha nemfa-élt találunk, sosem lépünk tovább az általa hi-

vatkozott csúcsba. [A kereszt-élnek és az előre-élnek megfelelő címkézést nem találhatjuk meg az algoritmus struktogramjában: ez csak a szemléltetéshez tartozik.] Másodszor az **(e,d)** élet dolgozzuk fel. Ez fehér csúcsba mutat, így az **(e,d)** fa-él lesz.

Most $time = 6$ idővel elértük a **d** csúcsot, beállítjuk az elérési idejét és szürkére színezzük. A **d** csúcsnak egy kimenő éle van, a **(d,a)**. Ez szürke, azaz elért, de még befejezetlen csúcsba mutat, így **(d,a)** vissza-él lesz. (Ld. a 3.8. tételt!) [Vegyük észre, hogy ha vissza-élt találunk, egyúttal irányított kört is találtunk a gráfban! (Ld. a 3.7. definíciót!)]

A **d** csúcsnak nincs több kimenő éle, így $time = 7$ idővel befejezzük, beállítjuk a befejezési idejét és feketére színezzük, majd visszalépünk a bele mutató fa-élen az éppen épülő mélységi fában a szülőjébe, ami az **e** csúcs.

Az **e** csúcsnak sincs már címkézetlen, azaz feldolgozatlan kimenő éle, így $time = 8$ idővel befejezzük, beállítjuk a befejezési idejét és feketére színezzük, majd visszalépünk a bele mutató fa-élen az éppen épülő mélységi fában a szülőjébe, ami a **b** csúcs.

A **b** csúcsnak sincs már címkézetlen kimenő éle, így $time = 9$ idővel befejezzük, beállítjuk a befejezési idejét és feketére színezzük, majd visszalépünk a bele mutató fa-élen az éppen épülő mélységi fában a szülőjébe, ami az **a** csúcs.

Az **a** csúcsnak még címkézetlen az **(a,e)** kimenő éle, ami az **e** fekete csúcsba mutat. Ennek az elérési ideje nagyobb, mint az **a** csúcsnak, tehát az **(a,e)** élt előre-élnek címkézzük. (Ld. a 3.8. tételt!)

Az **a** csúcsnak nincs már címkézetlen, azaz feldolgozatlan kimenő éle, így $time = 10$ idővel befejezzük, beállítjuk a befejezési idejét, feketére színezzük, és ezzel befejezzük az **a** gyökércsúcsú mélységi fa építését.

Ezzel visszalépünk a DFvisit eljárásból a DFS eljárásba. Újabb fehér csúcsot keresünk. [Vegyük észre, hogy ezen a ponton csak fehér és fekete csúcsokat találhatunk, szürkéket nem!] Sorra vesszük a **b, c, d, e** csúcsokat, amelyek mindegyike fekete már, majd megtaláljuk az **f** csúcsot, ami még fehér. Ezt beállítjuk a második mélységi fa gyökércsúcsának, majd innét indítjuk a következő mélységi vizitet.

Most $time = 11$ idővel elértük az **f** csúcsot, beállítjuk az elérési idejét és szürkére színezzük. Az **f** csúcsnak két kimenő éle van, az **(f,c)** és az **(f,e)**. Először az **(f,c)** élet dolgozzuk fel. Ez fekete (azaz befejezett) csúcsba mutat, aminek az elérési ideje kisebb, mint az **f** csúcsé, így az **(f,c)** élet kereszt-élnek címkézzük. Hasonlóan járunk el az **(f,e)** éllel.

Ezután az **f** csúcsnak sincs már címkézetlen, azaz feldolgozatlan kimenő éle, így $time = 12$ idővel befejezzük, beállítjuk a befejezési idejét, feketére színezzük, és ezzel befejezzük az **f** gyökércsúcsú mélységi fa építését.

Visszalépünk a DFvisit eljárásból a DFS eljárásba. Újabb fehér csúcsot

keresünk, de nincs már több csúcs, amit megvizsgálhatnánk. Befejeződik tehát a mélységi bejárás, és vele együtt a mélységi feszítő erdő létrehozása.

3.2.4. A mélységi keresés futási ideje

A szokásos $n = |G.V|$ és $m = |G.E|$ jelölésekkel a DFS mindkét ciklusa n-szer fut le. Mindegyik csúcsra, amikor először érjük el, azaz, amikor még fehér, pontosan egyszer, összesen n-szer hívódik meg a DFvisit rekurzív eljárás. A DFvisit ciklusa mindegyik csúcsra annyit iterál, amennyi a kimeneti fokszáma. Ez a ciklus tehát a gráf éleit dolgozza fel, mindegyiket egyszer. Ennélfogva összesen m iterációt végez. A DFS csak egyszer hívódik meg. Feltesszük most, hogy a backEdge eljárás mindegyik végrehajtása csak megjelöl egy vissza-élet, így $\Theta(1)$ műveletigényű, és műveletigénye hozzávehető az őt meghívó ciklusiterációéhoz.

Pontosan $3n+m+1$ lépést számolhatunk össze. Ebből a mélységi keresésre $MT(n), mT(n) \in \Theta(n+m)$.

3.2.5. A DAG tulajdonság eldöntése

3.10. Definíció. *A G irányított gráf akkor DAG (Directed Acyclic Graph = körmentes irányított gráf), ha nem tartalmaz irányított kört.*

A DAG-ok a gráfok fontos osztályát képezik: sok hasznos algorimus DAG-ot vár a bemenetén, így az input ellenőrzése is szükséges lehet. (Ld. pl. a *topologikus rendezést* (3.2.6) és a *DAG legrövidebb utak egy forrásból* algoritmust!)

Világos, hogy amennyiben a mélységi bejárás egy (u, v) vissza-élet talál, azzal irányított kört is talált a gráfban, mert ekkor a vissza-él a definíciója szerint egy mélységi fában az u csúcs egyik őse a v csúcs, és a v-ből u-ba vezető, fa-élekből álló út az (u, v) éllel együtt irányított kört alkot.

Bebizonyítható, hogy ha a G irányított gráf nem DAG, azaz irányított kört tartalmaz, akkor a DFS fog vissza-élet, és ezzel irányított kört találni [2]. [Az viszont nem garantált, hogy az összes irányított kört megtalálja. Könnyű olyan gráfot találni, ahol nem találja meg az összes irányított kört, mert ugyanaz a vissza-él több irányított körnek is a része lehet.]

3.11. Feladat. *Rajzoljon olyan három csúcsú, négy élű egyszerű gráfot, amelyben két egyszerű irányított kör van, és a DFS lehet, hogy megtalálja mindkettőt, de lehet, hogy csak az egyiket! Indokolja is az állítását!*

A fentiekből adódik a következő tétel.

3.12. Tétel. *A G irányított gráf DAG $\iff$ a mélységi bejárás nem talál G-ben vissza-élet.*

Ha a DFS talál egy (u, v) vissza-élet, akkor az $\langle u, \pi(u), \pi(\pi(u)), \ldots, v, u\rangle$ csúcssorozat visszafelé olvasva egyszerű irányított kört ad.

A fenti tétel alapján a backEdge eljárás akár ki is nyomtathatja a megtalált irányított kört. Ebben az esetben a maximális futási ideje nyilván $\Theta(n)$ lesz. (Szélsőséges esetben a megtalált irányított körök kinyomtatása akár meg is növelheti a mélységi bejárás műveletigényének aszimptotikus nagyságrendjét.)

3.13. Feladat. *Írja meg a backEdge(u, v) eljárás struktogramját abban az esetben, ha ennek feladata a megtalált irányított kör explicit, jól olvasható megjelenítése is! Ügyeljen a $\Theta(n)$ maximális műveletigényre!*

3.14. Feladat. *Adja meg irányított gráfok egy olyan sorozatát, amelyekre $m \in O(n)$, és így alapesetben $MT_{DFS}(n, m) \in \Theta(n)$, de ha a backEdge eljárásba az irányított körök kinyomtatását is beleértjük, akkor ezen a gráfsorozaton a DFS maximális műveletigénye $\Omega(n^2)$ lesz! Indokolja is az állítását!*

3.15. Feladat. *Módosítsa a mélységi bejárás algoritmusát úgy, hogy irányítatlan gráfokon tudjunk vele kört keresni! Hogyan ismerjük fel a vissza-éleket? [Irányítatlan gráfban, ha (u, v) fa-él, világos, hogy (v, u) nem lehet vissza-él, mert $(v, u) = (u, v)$.] Mi a helyzet az előre- és a kereszt-élekkel?*

3.2.6. Topologikus rendezés

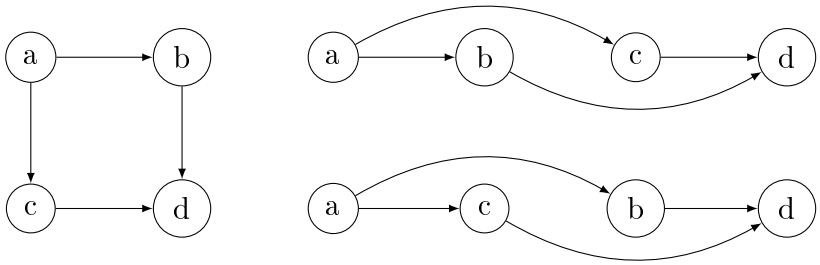
3.16. Definíció. *Irányított gráf topologikus rendezése alatt a gráf csúcsainak olyan sorba rendezését értjük, amelyben minden él egy-egy később jövő csúcsba (szemléletesen: balról jobbra) mutat.*

Ugyanannak a gráfnak lehet egynél több topologikus rendezése, mint a 12. ábra mutatja.

3.17. Tétel. *Tetszőleges irányított gráfnak pontosan akkor van topologikus rendezése, ha nincs irányított kör a gráfban, azaz a gráf DAG.*

Bizonyítás.

$\Rightarrow$ Ha van irányított kör a gráfban, jelölje $\langle u_1, u_2, \ldots, u_k, u_1\rangle$! Ekkor egy tetszőleges topologikus rendezésben u_1 után jön valahol u_2, az után valahol u_3, és így tovább, végül is u_1 után jön u_k, és u_k után u_1, ami ellentmondás. Tehát ekkor nincs topologikus rendezés (a gráf csúcsain).



12. ábra. Ugyanaz a DAG háromféleképpen lerajzolva: balra a szokásos módon ábrázolva, jobbra pedig a csúcsokat kétféleképpen, topologikus rendezésüknek megfelelően sorba rakva.

⇐ Ha nincs irányított kör a gráfban, akkor nyilván van olyan csúcs, aminek nincs megelőzője. Ha veszünk egy megelőzővel nem rendelkező csúcsot, és töröljük a gráfból, akkor a maradék gráfban nem keletkezik irányított kör, lesz megint legalább egy olyan, amelyiknek nincs megelőzője. Sorban a törölt csúcsok adják a topologikus rendezést.

□

A topologikus rendezést végrehajthatjuk például az irányított gráf mélységi bejárása segítségével.

Tetszőleges DAG-ra a topologikus rendezés algoritmusa:

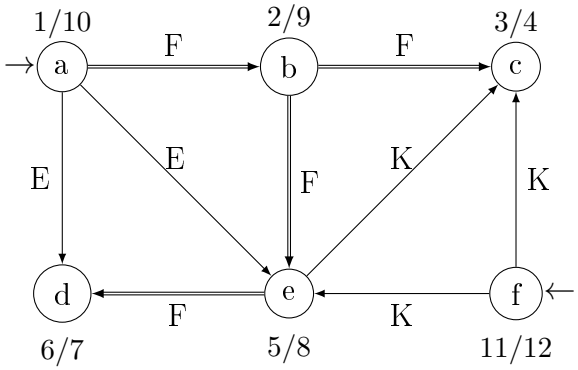
Először létrehozunk egy üres vermet, majd végrehajtuk gráf mélységi bejárását úgy, hogy valahányszor befejezünk egy csúcsot, a verem tetejére tesszük. Végül a verem tartalmát kiolvasva megkapjuk a gráf csúcsainak topologikus rendezését.

Vegyük észre, hogy ez az algoritmus képes elenőrizni a saját előfeltételét! Ha ugyanis a DFS vissza-élt talál, akkor a gráf irányított kört tartalmaz, és az algoritmus eredménye az, hogy nincs topologikus rendezés. (Ilyenkor a verem tartalma nem használható fel.)

Az algoritmus végrehajtására a 13. ábrán láthatunk egy példát.

A topologikus rendezés egy kézenfekvő alkalmazása az ún. *egygépes ütemezési probléma* megoldása: A csúcsok (munka)folyamatok, az élek a köztük lévő rákövetkezési kényszerek, és a folyamatokat ezeknek megfelelően kell sorba rakni.

3.18. Feladat. *Írja meg (1) a mélységi keresés és (2) a topologikus rendezést struktogramját (A) szomszédossági listás és (B) szomszédossági mátrixos gráfábrázolások esetén! Mit tud mondani az algoritmusok hatékonyságáról?*



13. ábra. A DAG topologikus rendezésésében a csúcsok a befejezési idők szerint szigorúan monoton csökkenően jelennek meg. Így a fenti gráfra a DFS a szokásos alfabetikus konvencióval a következő topologikus rendezést adja: $\langle f, a, b, e, d, c\rangle$. Vegyük észre, hogy ha az algoritmus indeterminizmusát más konvenció mentén oldanánk fel, gyakran az előbbitől különböző topologikus rendezést kapnánk!

3.19. Feladat. *Vegyük észre, hogy a 3.17. tétel bizonyításának második fele algoritmusként is értelmezhető! Írja meg ennek alapján a topologikus rendezés egy, a DFS-től független struktogramját! Hogyan lehetne az input gráf lebontását $O(n)$ munkatár segítségével elkerülni? Mit tud mondani az algoritmus hatékonyságáról? Ez az algoritmus hogyan fog viselkedni, ha a gráf irányított kört tartalmaz?*