Algoritmusok II. gyakorló feladatok

Ásványi Tibor <asvanyi@inf.elte.hu>

2024. október 25.

Mj.: Ebben a feladatsorban MW az algoritmusok maximális munkatárigényére vonatkozik. Ez nem tartalmazza az input és az output adatszerkezeteket sem, csak az algoritmus futása során felhasznált temporális adatszerkezetek és a programfutás (pl. rekurzív hívások) adminisztrálához szükséges tárterület összegének maximumát.

A gráfalgoritmusok szemléltetésénél a csúcsokat gyakran az ábécé első néhány betűjével jelöljük. Ilyenkor a=1, b=2, c=3, d=4, e=5, f=6, g=7, h=8, i=9 stb.

1. AVL fák

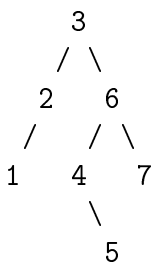
Jelölés: Az alábbi feladatokban a konkrét AVL fákat a bináris fák szokásos szöveges, azaz zárójeles reprezentációban adjuk meg. Tetszőleges nemüres bináris fa zárójeles, azaz szöveges alakja (textual form):

(balRészFa Gyökér jobbRészFa)

Az üres (rész)fát egy üres zárójelpár reprezentálja, pl. ().

Elkerülendő a formulák felesleges bonyolítását, a levélcsúcsok üres részfáihoz tartozó üres zárójelpárokat elhagyjuk. Pl. az "5" kulcsú, egy csúcsú bináris fa szöveges reprezentációja (() (5) ()) helyett egyszerűen (5).

A bináris fák szöveges ábrázolásánál a könnyebb olvashatóság kedvéért többféle zárójelpárt is használhatunk. A példákban, a részfák különböző szintjei szerint, három- vagy négyféle zárójelpárt alkalmazunk: { }, [], ( ) és esetleg ⟨ ⟩. Pl.
{ [(1) 2 ()] 3 [(⟨⟩ 4 ⟨5⟩) 6 (7)] }
az alábbi fa zárójeles, azaz szöveges formája.



AVL.1. Adott a { [(2) 4 (⟨6⟩ 8 ⟨10⟩)] 12 [() 14 (16)] } AVL fa. Rajzoljuk le a fát a belső csúcsok egyensúlyaival együtt! Szemléltessük a remMin művelet és az 5 beszúrását, **mindkét esetben az eredeti fára**! Jelöljük, ha ki kell egyensúlyozni, a kiegyensúlyozás helyét, és a kiegyensúlyozás után is rajzoljuk újra fát! A rajzokon jelöljük a belső csúcsok egyensúlyait is, a szokásos módon! Rajzoljuk le a hat általános kiegyensúlyozási séma közül azokat, amiket alkalmaztunk!

AVL.2. Adott az { [(1) 2 (⟨⟩ 3 ⟨4⟩)] 6 [(7) 8 ()] } AVL fa. Rajzoljuk le a fát a belső csúcsok egyensúlyaival együtt! Szemléltessük a 8 törlését és az 5 beszúrását, **mindkét esetben az eredeti fára**! Jelöljük, ha ki kell egyensúlyozni, a kiegyensúlyozás helyét, és a kiegyensúlyozás után is rajzoljuk újra fát! A rajzokon jelöljük a belső csúcsok egyensúlyait is, a

szokásos módon! Rajzoljuk le a hat általános kiegyensúlyozási séma közül azokat, amiket alkalmaztunk!

AVL.3. Rajzoljuk le a következő AVL fát a belső csúcsok egyensúlyaival együtt! – { [(⟨1⟩ 2 ⟨3⟩) 5 (6)] 7 [() 8 (9)] } – Szemléltessük a 4 beszúrását és a 7 törlését, **mindkét esetben az eredeti fára**! Jelöljük, ha ki kell egyensúlyozni, a kiegyensúlyozás helyét, és a kiegyensúlyozás után is rajzoljuk újra fát! A rajzokon jelöljük a belső csúcsok egyensúlyait is, a szokásos módon! Rajzoljuk le a hat általános kiegyensúlyozási séma közül azokat, amiket alkalmaztunk!

AVL.4. Rajzoljuk le a következő AVL fát a belső csúcsok egyensúlyaival együtt! – { [() 1 (2)] 4 [(5) 6 (⟨7⟩ 8 ⟨⟩)] } – Szemléltessük a 3 beszúrását és a 4 törlését, **mindkét esetben az eredeti fára**! Jelöljük, ha ki kell egyensúlyozni, a kiegyensúlyozás helyét, és a kiegyensúlyozás után is rajzoljuk újra fát! A rajzokon jelöljük a belső csúcsok egyensúlyait is, a szokásos módon! Rajzoljuk le a hat általános kiegyensúlyozási séma közül azokat, amiket alkalmaztunk!

AVL.5. Szemléltessük az 1 2 7 3 5 6 8 9 4 számok egymás utáni beszúrását egy, kezdetben üres AVL fába! Az így adódó fából indulva, ezután szemléltessük az 1 3 4 8 9 2 számok egymás utáni törlését! Minden egyes beszúrás illetve törlés után rajzoljuk újra fát! Jelöljük, ha ki kell egyensúlyozni, a kiegyensúlyozás helyét, és a kiegyensúlyozás után is rajzoljuk újra fát! A rajzokon jelöljük a belső csúcsok egyensúlyait is, a szokásos módon!

AVL.6. Rajzoljunk 0, 1, 2, 3, 4 és 5 magasságú Fibonacci fákat, amelyek egyben AVL fák is!

AVL.7. Mutassunk olyan, öt magasságú AVL fát, amelynek adott levelét törölve, minden, a fában a törölt csúcs feletti szinten ki kell egyensúlyozni!

AVL.8. Az előadásról ismert leftSubTreeShrunk(t, d) eljárás mintájára dolgozzuk ki az ott csak felhasznált rightSubTreeShrunk(t, d) eljárást, ennek segédeljárásait, és az ehhez szükséges kiegyensúlyozási sémát! Mutassuk be ennek működését néhány példán! Mutassunk néhány olyan, legalább négy magasságú fát és kulcsot, amelyekre az AVLdel eljárás minden, a fizikailag törölt csúcs feletti szinten kiegyensúlyozást végez!

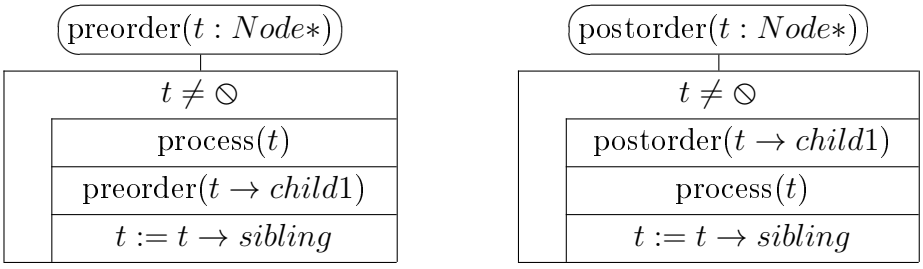
AVL.9. Írjuk meg a $t \neq \oslash$ AVL fára a remMax$(t, maxp)$ eljárást! Mekkora lesz a futási idő? Miért?

2. Általános fák

Á.1. Próbáljunk megadni az általános fákra a bináris láncolt ábrázolás mellett másfajta láncolt reprezentációkat is! Hasonlítsuk össze az általunk adott ábtázolásokat és a bináris láncolt reprezentációt, memóriaigény és rugalmasság szempontjából![1]

Á.2. Rajzoljuk le az {1 [2 (5)] [3] [4 (6) (7)]} általános fát! Írjunk programot, ami kiír egy binárisan láncolt fát a fenti zárójeles alakban! Írjunk olyat is, ami egy szövegfájlból visszaolvassa! Készítsük el a kiíró és a visszaolvasó programokat az általunk kidolgozott alternatív láncolt ábrázolásokra is! (A visszaolvasó programokat általában nehezebb megírni.)

Á.3. A binárisan láncolt általános fáknak az előadásról ismert preorder és postorder bejárásai:



Ahol a Node típus a következő.

Node
+ $child1, sibling$: Node* // $child1$: első gyerek; $sibling$: következő testvér + $key : \mathcal{T}$ // $\mathcal{T}$ ismert típus
+ Node() { $child1 = sibling = \oslash$ } // egycsúcsú fát képez belőle + Node($x : \mathcal{T}$) { $child1 = sibling = \oslash$; $key = x$ }

Lássuk be a fenti bejárások helyességét! (Ötlet: A testvérek listájának mérete szerinti teljes indukció pl. célravezető.) Lássuk be egy ellenpélda segítségével, hogy az általános fa inorder bejárása[2] nem szimulálható a bináris reprezentáció egyik nevezetes bejárásával[3] sem! Írjuk meg a bináris láncolt

[1] Ötlet: Próbáljuk meg a részfákra mutató pointereket (1) statikus tömbben, (2) dinamikus tömbben, (3) egyszerű láncolt listában, ún. *csatolóelemek* segítségével tárolni!

[2] A (G $t_1 \ldots t_n$) fára $n > 0$ esetén előbb t_1-et járja be, majd feldolgozza G-t, aztán bejárja sorban a $t_2 \ldots t_n$ részfákat; $n = 0$ esetén csak feldolgozza G-t.

[3] preorder, inorder, postorder, szintfolytonos

reprezentációra az általános fa inorder bejárását és szintfolytonos bejárását is!

Á.4. Írjuk meg a fenti bejárásokat az általunk adott alternatív láncolt reprezentációkra is!

Á.5. Írjunk olyan programokat, amelyek képesek tetszőleges általános fa egy adott reprezentációjából egy adott másik ábrázolású másolatot készíteni!

3. B+ fák

Az itt következő feladatokban a beszúrásokat és törléseket a
http://aszt.inf.elte.hu/~asvanyi/ad/B+ fa.pdf
címen és az előadásokon megadott algoritmusok szerint kell elvégezni. A B+ fák mindegyik, az algoritmusok működésének bemutatására vonatkozó feladatban negyedfokúak ($d = 4$).

Megjegyzés:
- Egy negyedfokú B+ fa tetszőleges részfájának szöveges formája, ha a részfa nem egyetlen levélcsúcsból áll, a következő sémák valamelyikére illeszkedik:
(T_1 k_1 T_2), (T_1 k_1 T_2 k_2 T_3) és (T_1 k_1 T_2 k_2 T_3 k_3 T_4),
ahol T_i az i-edik részfa, k_i pedig a hasító kulcs az i-edik és az $(i + 1)$-edik részfa között.
- A negyedfokú B+ fák tetszőleges levelének szöveges formája a következő sémák valamelyikére illeszkedik:
(k_1 k_2) és (k_1 k_2 k_3),
ahol k_i a B+ fa által reprezentált halmaz kulcsa. Ha azonban a levélcsúcs egyben a B+ fa gyökércsúcsa, alakja (k_1) is lehet.
- A példákban, a részfák különböző szintjei szerint, háromféle zárójelpárt használunk: { }, [ ] és ( ).

+.1. Vegyük a legelső B+ fa illusztrációt a fent hivatkozott „B+ fa.pdf" fájlból! Szövegesen:

{ [(1 4) 6 (9 10) 11 (11 12)] 13 [(13 15) 16 (16 20 25)] }.

Szúrjuk be az 5-öt a fenti fába, majd a 6, 17, 8 és 7 számokat sorban az eredményül adódó B+ fákba! Rajzoljuk le az eredeti fát, majd a sorban az egyes beszúrások eredményeit!

+.2. Adott az alábbi B+ fa:

{ [2 4] 5 [6 8 10] 12 [12 14] 15 [16 18] }.

Szúrjuk be az 5 kulcsot a fenti fába! Rajzoljuk le az eredeti fát és a beszúrás eredményét is!

+.3. Adott az alábbi B+ fa:
{ [(1 3 5) 8 (9 10)] 11 [(11 12) 13 (13 15) 20 (20 25 30)] }.
Töröljük a 20-at a fenti fából, majd a 10, 5 és 25 számokat sorban az eredményül adódó B+ fákból! Rajzoljuk le az eredeti fát, majd a sorban az egyes törlések eredményeit!

+.4. A t pointer egy d fokszámú, h magasságú B+ fa gyökércsúcsára mutat. Írjuk meg a printBplusTree0(t, d, h) eljárást, ami kiíratja a fát szöveges formában, csak kerek zárójeleket használva! A levelekben lévő kulcsokhoz tartozó, az adatrekordokra mutató hivatkozásokat ebben az eljárásban nem vesszük figyelembe. $MT(n, d) \in O(n * d)$, ahol n a t fa csúcsainak száma (ami az eljárás számára nem adott).

+.5. A t pointer egy d fokszámú, h magasságú B+ fa gyökércsúcsára mutat. Írjuk meg a printBplusTree(t, d, h) eljárást, ami kiíratja a fát szöveges formában úgy, hogy az egész fa „{ }" zárójelpárok között van, a gyökér gyerekeihez, az első szinten levő csúcsokhoz tartozó részfák „[ ]" zárójelpárok között, a második szinten levő csúcsokhoz tartozó részfák „( )" zárójelpárok között, a harmadik szinten levő csúcsokhoz tartozó részfák újra „{ }" zárójelpárok között vannak stb.! A levelekben lévő kulcsokhoz tartozó, az adatrekordokra mutató hivatkozásokat ebben az eljárásban nem vesszük figyelembe. $MT(n, d) \in O(n * d)$, ahol n a t fa csúcsainak száma (ami az eljárás számára nem adott).

4. Gráfok ábrázolásai

GR.1. A `b` pointer egy bináris láncolt ábrázolású általános fa gyökércsúcsára mutat. Csúcsait az `1..n` számok egyértelműen címkézik. Írjuk meg a `b2g(b,G)` eljárást, ami a fát átmásolja a `G:Edge*[n]` pointertömb segítségével reprezentált, szomszédossági listás ábrázolású gráfba! (A híváskor `G[1..n]` nem definiált.) $MT(n) \in O(n)$, $MW(n) \in O(n)$, ahol n a fa csúcsainak száma.

GR.2. A `G:Edge*[n]` pointertömb egy élsúlyozatlan gráf szomszédossági listás ábrázolása, amiről tudjuk, hogy gyökeres fa. Írjuk meg a `g2b(G,b)` eljárást, ami a gráfként ábrázolt fát átmásolja a `b` gyökérpointerű, bináris láncolt ábrázolású általános fába! (A híváskor `b` nem definiált hivatkozás.)

$MT(n) \in O(n)$, $MW(n) \in O(n)$, ahol n a fa csúcsainak száma. **Ötlet:** használjunk egy `T[1..n]` pointer segédtömböt, ahol `T[i]` az eljárás által létrehozott fa `i` kulcsú csúcsára mutat! (`i`$\in$`1..n`)

GR.3. Egy gráf csúcsait az `1..n` egész számok címkézik. Egy gráfkeresés eredménye a `P[1..n]` vektorban ábrázolt fa: Minden `i` csúcsra `P[i]` az `i` szülője a fában, kivétel a gyökércsúcs, amire `P[i]==0`, és a Gráfkeresés során el nem ért csúcsok, ahol `P[i]==-1`.

Írjuk meg az `fb(P,t)` eljárást, ami a fáról binárisan láncolt általános fa másolatot készít! (Az eredmény gyökérpointere `t`.) A láncolt reprezentációban a testvérek listái index szerint növekvően rendezettek legyenek! $MT(n) \in O(n)$, $MW(n) \in O(n)$

(**Ötlet:** a megoldáshoz használjuk a `T[1..n]` pointer-segédtömböt: `T[i]` mutasson a láncolt fa `i` cimkéjű csúcsára, ha a feszítőfa tartalmazza a csúcsot! A `P[1..n]` vektort fordítva (n-től 1-ig visszafelé) érdemes feldolgozni.)

GR.4. Egy gráf csúcsai az `1..n` egész számok. Egy gráfkeresés eredménye a `P[1..n]` vektorban ábrázolt feszítőfa: Minden `i` csúcsra `P[i]` az `i` szülője a fában, kivétel a gyökércsúcs, amire `P[i]==0`, és a gráfkeresés során el nem ért csúcsok, ahol `P[i]==-1`.

Erről a `t` binárisan láncolt általános fa másolatot készítettük. A `P[1..n]` vektor már nem érhető el.

Írjuk meg a `bf(t,P)` eljárást, ami a `t` általános fából rekonstruálja az eredeti `P[1..n]` fát! $MT(n) \in O(n)$, $MW(n) \in O(n)$

(**Ötlet:** Töltsük fel a `P[1..n]` vektort `-1` értékekkel, majd járjuk be a `t` általános fát preorder módon úgy, hogy mindig ismerjük az aktuális csúcs szülőjét és ezt be is írjuk a `P[1..n]` vektorba a megfelelő helyre.)

GR.5. Egy gráfkeresés eredménye a `P[1..n]` fa.
`i,j` eleme `1..n` esetén:
`P[i]=j` akkor, ha az `i.` csúcs szülője a feszítőfában a `j.` csúcs.
`P[i]=0` akkor, ha az `i.` csúcs a feszítőfa gyökere.
`P[i]<0` akkor, ha az `i.` csúcs nincs benne a fában.

Írjuk meg a `melysegek(P,d)` eljárást, ami kiszámítja a `d[1..n]` vektort, ami sorban az egyes csúcsok fa-beli mélységeit tartalmazza! A gyökér mélysége `0`. `d[i]` végtelen akkor, ha az `i.` csúcs nincs benne a fában. $MT(n) \in O(n)$, $MW(n) \in O(n)$. A megoldáshoz *ne* használjunk láncolt adatszerkezetet!

(**Ötlet:** Készítsük el a `melyseg(P,i,d)` rekurzív segédeljárást, ami kiszámítja az `i.` csúcs – és ha kell, az ősei – mélységét!

GR.6. Egy körmentes irányított gráfot a `G[1..n]` pointertömb segítségével szomszédossági listákkal ábrázoltunk.

Írjuk meg a `gyf(G)` függvényt, ami visszaadja a gyökércsúcs indexét, ha a gráf egy gyökeres fa, különben nullát ad vissza! $MT(n,m) \in O(n+m)$, ahol m az élek száma. $MW(n) \in O(n)$.

(Ötlet: alkalmazhatunk egy segédtömböt, amibe beírjuk a csúcsok szülőjének indexét.)

GR.7. A G irányított gráf csúcsait az $1..n$ számokkal címkéztük. A gráfot a $C[1..n, 1..n]$ szomszédossági mátrixszal ábrázoltuk. Írjuk meg a Transzponál(C) eljárás struktogramját, ami helyben transzponálja a gráfot! $MT(n) \in \Theta(n^2)$; $MW(n) \in \Theta(1)$.

GR.8. A G irányított gráf csúcsait az $1..n$ számokkal címkéztük. A gráfot szomszédossági listákkal ábrázoltuk a $G[1..n]$ pointertömb segítségével. Írjuk meg a Transzponál(G, G^T) eljárás struktogramját, ami $G^T[1..n]$-ben előállítja a G gráf transzponáltjának szomszédossági listás ábrázolását! Az eredeti gráfot ne változtassuk meg!

$MT(n,m) \in \Theta(n+m)$, ahol m az élek száma.

GR.9. A G irányított gráf csúcsait az $1..n$ számokkal címkéztük. A gráfot a $C[1..n, 1..n]$ szomszédossági mátrixszal ábrázoltuk. Írjuk meg a befokok(C, BE) eljárás struktogramját, ami a $BE[1..n]$ tömbben meghatározza a gráf csúcsainak bemeneti fokszámait! A gráfot ne változtassuk meg! $MT(n) \in \Theta(n^2)$; $MW(n) \in \Theta(1)$.

GR.10. A G irányított gráf csúcsait az $1..n$ számokkal címkéztük. A gráfot szomszédossági listákkal ábrázoltuk a $G[1..n]$ pointertömb segítségével. Írjuk meg a befokok(G, BE) eljárás struktogramját, ami a $BE[1..n]$ tömbben meghatározza a gráf csúcsainak bemeneti fokszámait! A gráfot ne változtassuk meg!

$MT(n,m) \in \Theta(n+m)$, ahol m az élek száma; $MW \in \Theta(1)$

5. Szélességi keresés gráfokon

BFS.1. Szemléltessük a szélességi keresés működését az alábbi gráfokon az s indexű csúcsból indítva! Mutassuk be a sor, a $d()$ és a $\pi()$ értékek alakulását a gyakorlatról ismert módon! Nemdeterminisztikus esetekben mindig a kisebb indexű csúcsot dolgozzuk fel először! Végül rajzoljuk le a szélességi feszítőfát, amit az algoritmus kiszámolt!

BFS.1.a $s = $ e

	a	b	c	d	e	f
a	0	0	0	1	0	0
b	1	0	1	0	1	0
c	0	0	0	0	0	0
d	0	1	0	0	1	0
e	0	0	1	1	0	0
f	0	0	1	0	1	0

BFS.1.b[4] $s = $ c

a $\to$ b; d. b $\to$ e. c $\to$ e; f.
d $\to$ b. e $\to$ d. f.

BFS.1.c $s = $ e

a $\to$ d. b $\to$ a; c; e. c.
d $\to$ b; e. e $\to$ c; d. f $\to$ c; e.

BFS.1.d[5], $s = $ d

a $-$ b; e. b $-$ f. c $-$ d; f; g. d $-$ g.
e $-$ f. f $-$ g. g. h.

BFS.2. Az `Adj[1..n]` pointertömb egy élsúlyozatlan gráf szomszédossági listás ábrázolása. Írjuk meg a `BFS(Adj,s,d,P)` eljárást, ami az `s` kezdőcsúcsból indítva szélességi keresést hajt végre a fán! A csúcsok mélységeit a `d[1..n]`, a szélességi fát pedig a `P[1..n]` tömbben számoljuk ki, amik a híváskor definiálatlan értékeket tartalmaznak! $MT(n,m) \in O(n+m)$, $MW(n) \in O(n)$, ahol n a gráf csúcsainak, m pedig az éleinek száma.

BFS.3. Egy erdőben van két mókus, mindkettő ugyanakkorát tud ugrani. Két különböző fán vannak. Mindkettő elkezd fáról fára ugrálni. Ugyanannyit ugranak, de nem okvetlenül egyszerre. Kérdés, közben találkozhatnak-e?

Modellezzük a problémát, és írjunk rá struktogramot!

$MT(n,m) \in O(n+m)$, ahol n a fák, m pedig az erdőben a mókusok számára lehetséges, fáról fára való, egymástól különböző ugrások száma. (Vigyázat, nem biztos, hogy egy ilyen ugráshoz mindig lehetséges a visszaugrás is! Másrészt az m szám nem a tényleges ugrások száma, hanem a lehetséges, egymástól páronként különböző, fáról fára való ugrások száma.)

[4] $u \to v_1; \ldots v_n$. azt jelenti, hogy a gráfnak a következő irányított élei vannak: $(u,v_1), \ldots (u,v_n)$.

[5] $u - v_1; \ldots v_n$. azt jelenti, hogy a gráfnak a következő irányítatlan élei vannak: $(u,v_1), \ldots (u,v_n)$.

Ötlet: Indítsunk két h mélységű szélességi keresést a két adott csúcsból (a h mélységű csúcsok gyerekeit már nem tesszük a sorba). Akkor találkozhatnak, ha a második bejárás elér legalább egy, az első által is érintett csúcsot.

6. Mélységi gráfbejárás

DFS.1.a, Szemléltessük a mélységi bejárás működését az alábbi gráfon! Írjuk az elérési számokat és a befejezési számokat a gráf grafikus ábrázolásán a csúcsok mellé! Jelezzük a különböző éltípusokat a szokásos módon! Nemdeterminisztikus esetekben mindig a kisebb indexű csúcsot dolgozzuk fel először! Adjuk meg a gráf csúcsainak a mélységi bejárásból adódó topologikus rendezését!

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

DFS.1.b, Tegyük fel, hogy behúzzuk a fenti gráfba a (c,v) élt, ahol $v \in$ {a,b,d,e,f}. Az így adódó öt új gráf közül melyeknek van topologikus rendezése? Hogyan változik a mélységi bejárás és az élek címkéi az egyes esetekben? Amikor nincs az új gráfnak topologikus rendezése, ez melyik időpontban derül ki és miért?

DFS.2. Egy irányított gráf csúcsait az `1..n` számokkal címkéztük. A gráfot a `C[1..n,1..n]` csúcsmátrix segítségével ábrázoljuk. Feltesszük, hogy a gráf nem tartalmaz irányított kört. Csúcsai egy topologikus rendezését, azaz ütemezését állítjuk elő a `TR[1..n]` vektorban. Adott a `be[1..n]` tömb, ami kezdetben a csúcsok bemeneti fokszámait tartalmazza. Egy `i` csúcs akkor ütemezhető be, ha `be[i]==0`. Ha egy csúcsot beütemezünk, akkor a közvetlen rákövetkezői `be[j]` értékeit eggyel csökkentjük.
(Struktogram: `TopRend(C,be,TR)`)
$MT(n) \in O(n^2)$, $MW(n) \in O(n)$

DFS.3. Írjunk struktogramot gráfok a topologikus rendezésének alábbi változatára!
Egy irányított gráf csúcsait az $1..n$ számokkal címkéztük. A gráfot szomszédossági listásan ábrázoltuk a $G[1..n]$ pointertömb segítségével. Ha a gráf nem tartalmaz irányított kört, csúcsai egy topologikus rendezését, azaz ütemezését állítjuk elő a $TR[1..n]$ vektorban, és *true* értékkel térünk vissza. (Ha egy csúcsból egy másikba irányított út vezet, akkor az első a topologikus rendezésben kisebb sorszámmal szerepel. Ha két csúcs között nincs irányított

út, a topologikus rendezésben a sorrendjük közömbös.) Ha a gráf tartalmaz irányított kört, *false* értékkel térünk vissza. A struktogramot az alábbi algoritmus szerint kell megírni.
(Struktogram: TopRend(G, be, TR) logikai függvény)
$MT(n, m) \in O(n + m)$, ahol m az élek száma; $MW(n) \in O(n)$

Algoritmus:

Adott a $be[1..n]$ tömb, ami kezdetben a csúcsok bemeneti fokszámait tartalmazza. (Ez tehát a programunk számára adott, nem kell megírni.) Egy u csúcs ($u \in 1..n$) akkor ütemezhető be, ha még nem ütemeztük be, és $be[u] = 0$. Az algoritmus minden lépésében egy beütemezhető csúcsot ütemezünk be, azaz betesszük a $TR[1..n]$ vektorba, „az első szabad helyre". A beütemezhető csúcsok egy Q sorban vannak. Ha egy csúcsot beütemezünk, akkor a közvetlen rákövetkezői $be[v]$ értékeit eggyel csökkentjük. Ha a sor elfogy, aszerint térünk vissza *true* értékkel, hogy minden csúcsot beütemeztünk-e.

DFS.4. Egy irányított gráfot a $G[1..n]$ pointertömb segítségével szomszédossági listákkal ábrázoltunk.

Írjuk meg a gye(G, Q) eljárást, ami visszaadja a Q sorban a gyökércsúcsok indexeit, ha a gráf egy erdő, azaz gyökeres fák halmaza, különben a Q sort üresen adja vissza! $MT(n, m) \in O(n+m)$, ahol m az élek száma. $MW(n) \in O(n)$.

Ötlet: Írjunk mélységi bejárást, a következő módosítással. Ha fehér csúcsot találunk, a szokásos módon folytatjuk. Ha szürke csúcsot találunk, akkor irányított kört is találtunk, és a gráf nem erdő. Ha fekete csúcsot találunk, két eset van. (1) Ha ez (a mondjuk j csúcs) egy korábban felépített részleges feszítőfa gyökere ($P[j] = 0$), akkor becsatoljuk abba a fába, amit éppen most építünk a $P[1..n]$ tömbben. (2) Ha nem gyökércsúcsot találtunk, akkor ennek a csúcsnak a bemeneti fokszáma nagyobb mint 1, tehát a gráf nem erdő. Ha végigmegy a bejárás, és nem találunk sem szürke, sem második típusú fekete csúcsot, akkor a gráf egy erdő, csak a fái gyökércsúcsait a $P[1..n]$ tömbből a Q sorba kell másolni. Különben Q üresen marad.

DFS.5. Az alábbi gráfon szemléltessük az előadásról ismert módon
(a) a mélységi bejárásos
(b) a csúcsok bemeneti fokszámait felhasználó
topologikus rendezés algoritmusát! (Ez két külön feladat.)

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

DFS.5.c, Tegyük fel, hogy behúzzuk a fenti gráfba a (d,v) élt, ahol $v \in$ {a,b,c,f}. Az így adódó négy új gráf közül melyeknek van topologikus ren-

dezése? Hogyan változik a mélységi bejárás, az élek címkéi és a topologikus rendezés az egyes esetekben?

DFS.6. Legyen az **Élek** = **Él*** mutató típus, ahol

Él
$+u, v : \mathbb{N}$ // $u, v \in 1..n$
$+mut$: **Élek**

.

DFS.6.(a) Írjuk meg a TopRend(G, TR, π) függvény struktogramját DAG-ok topologikus rendezésére mélységi bejárás segítségével! Az irányított gráf csúcsait az $1..n$ számokkal címkéztük. A gráfot szomszédossági listákkal ábrázoltuk a $G[1..n]$ pointertömb segítségével.

Ha a gráf nem tartalmaz irányított kört, csúcsai egy topologikus rendezését, azaz ütemezését állítjuk elő a $TR[1..n]$ vektorban, és $\oslash$ pointerrel térünk vissza.

Ha a gráf tartalmaz irányított kört, akkor a mélységi bejárás által talált vissza-élek egyszerű láncolt listájával térünk vissza. A $\pi[1..n]$ tömbben a mélységi feszítő erdőt kapjuk meg.

$MT(n, m) \in O(n + m)$, ahol m az élek száma; $MW(n) \in O(n)$

DFS.6.(b) Írjuk meg a TopRend(C, TR, π) függvény struktogramját DAG-ok topologikus rendezésére mélységi bejárás segítségével! Az irányított gráf csúcsait az $1..n$ számokkal címkéztük. A gráfot a $C[1..n, 1..n]$ szomszédossági mátrixszal ábrázoltuk.

Ha a gráf nem tartalmaz irányított kört, csúcsai egy topologikus rendezését, azaz ütemezését állítjuk elő a $TR[1..n]$ vektorban, és $\oslash$ pointerrel térünk vissza.

Ha a gráf tartalmaz irányított kört, akkor a mélységi bejárás által talált vissza-élek egyszerű láncolt listájával térünk vissza. A $\pi[1..n]$ tömbben a mélységi feszítő erdőt kapjuk meg.

$MT(n) \in O(n^2)$; $MW(n) \in O(n)$.

DFS.7. Írjuk meg a KörKiíró(B, π) eljárás struktogramját, ahol a B a DFS.6. feladatban megírt topologikus rendezések által visszaadott **Élek** típusú lista, $\pi[1..n]$ pedig az ott meghatározott mélységi feszítő erdő! $MT(n, k) \in O(n * k)$, ahol k az E lista hossza; $MW(n) \in O(n)$.

7. Minimális feszítőfák

MST.1. Az alábbi egyszerű gráfokon[6] szemléltessük a gyakorlatról ismert módon (a) a Kruskal algoritmust, (b) az a, illetve a b csúcsból indított Prim algoritmust! Mindegyik esetben rajzoluk le a minimális feszítőfát, amit kaptunk!

a − b, 4; d, 1.	b − c, 1; d, 2.	c − e, 1; f, 1.
d − e, 3.	e − f, 1.	f.

a − b, 2; e, 1.	b − f, 0.	c − d, 3; f, 1; g, 1.	d − g, 4; h, 2.
e − f, 1.	f − g, 2.	g − h, 1.	h.

MST.2.* Általánosítsuk a Prim algoritmust arra az esetre, ha a gráf nem összefüggő, és így nem tudunk feszítőfát, csak feszítő erdőt megadni! A gráfot a szimmetrikus $A[1..n, 1..n]$ szomszédossági mátrix segítségével ábrázoljuk. Az eredmény a $c[1..n]$ és az $E[1..n]$ tömbökben keletkezik ($E[i] = j$ azt jelenti, hogy az i csúcsot az (i, j) élen keresztül vettük hozzá a fához, $c[i] = x$ pedig azt, hogy az $w(i, j) = x$). (Struktogram: Prim(A, c, E))
$MT(n) \in O(n^2)$, $MW(n) \in O(n)$

MST.3.* Írjunk struktogramot a Prim algoritmus alábbi változatára!
Egy nem okvetlenül összefüggő, súlyozott irányítatlan gráf csúcsait az `1..n` számokkal címkéztük. A gráfot szomszédossági listákkal ábrázoltuk a `G[1..n]` pointertömb segítségével. A gráf egy minimális feszítő erdőjét állítjuk elő a Prim algoritmus segítségével, a `P[1..n]` vektorban. A minimális feszítőerdő fáit a gráf összefüggő komponenseinek minimális feszítőfái adják. A feszítőfákat az `E[1..n]` vektorban irányított fákként ábrázoljuk, és minden él az őt tartalmazó feszítőfa gyökere felé irányítva jelenik meg. A `c[1..n]` vektor az `E[1..n]` vektorban visszadott élek súlyait fogja tartalmazni: `j>0` esetén `E[i]=j` jelentése, hogy az `(i,j)` él eleme az egyik feszítőfának. Ilyenkor `c[i]` tartalmazza az `(i,j)` él súlyát (azaz költségét). `E[i]=0` jelentése, hogy `i` az egyik feszítőfa gyökere. Ilyenkor `c[i]=0` lesz. Ha a Prim algorimus során egy csúcs még nincs feszítőfában, de szomszédja az éppen épített feszítőfa egy vagy több csúcsának, akkor azt mondjuk, hogy szomszédja ennek a feszítőfának, és a feszítőfától való távolsága egy minimális súlyú él költsége, ami őt a feszítőfához kapcsolja. A struktogramot a fenti fogalmak és

[6] $u - v_1, w_1; \ldots v_n, w_n$. azt jelenti, hogy a gráf tartalmazza az $(u, v_1), \ldots (u, v_n)$ irányítatlan éleket, sorban $w_1, \ldots w_n$ súlyokkal. Minden irányítatlan élet csak egyszer írunk fel, mivel most tetszőleges (u, v) élre $(u, v) = (v, u)$, és így $w(u, v) = w(v, u)$.

ábrázolás, valamint az alábbi algoritmus szerint kell megírni. (Struktogram: `Prim(G,E,c)`) $MT(n) \in O(n^2)$, $MW(n) \in O(n)$

Algoritmus:
I. Kezdetben egyik csúcs sincs feszítőfában.
`E[1..n]:=-1; c[1..n]:=INFINITE;`
II. Válasszunk ki egy tetszőleges `s` csúcsot, ami még nincs feszítőfában! Legyen ez egy új feszítőfa gyökere! `E[s]:=0; c[s]:=0`
III. A most kiválasztott csúcs minden olyan `j` szomszédjára, ami még nincs a feszítőfában, állítsuk be az `E[j]` és `c[j]` értékeket, a `j` csúcsnak a feszítőfától való távolsága szerint!
IV. Ha a feszítőfának van szomszédja, válasszunk ki egyet, ami minimális távolságra van tőle (ezt a `c[1..n]` tömbön egy feltételes minimumkiválasztással oldjuk meg), és a kiválasztott csúcsot vegyük hozzá a feszítőfához, majd folytassuk a III. ponttól!
V. Ha a feszítőfának nincs szomszédja, akkor az aktuális feszítőfa kész van.
VI. Ha van még olyan csúcs ami egyetlen feszítőfában sincs benne, folytassuk a II. ponttól!
VII. Különben a feszítőerdő is kész van.

MST.4. Adjuk meg Kruskal algoritmusában az Unió-HolVan adatszerkezetet inicializáló eljárás, továbbá a HolVan (FindSet) függvény és az Unió (union) eljárás struktogramját implementációs szinten!

MST.5. Módosítsa úgy Kruskal algoritmusát, hogy $O(n + m \lg n)$ műveletigénnyel meghatározza egy tetszőleges (irányított v. irányítatlan) egyszerű gráf összefüggő komponenseit! Feltesszük, hogy a gráf szomszédossági listával adott. Eredményként az összefüggő komponensek csúcshalmazait kell kiírni: a szokásos jelöléssel, mindegyik komponens csúcshalmaza elemeinek sorszámait kapcsos zárójelek között, vesszővel elválasztva jelenítjük meg, minden egyes csúcshalmazt külön sorba írva.
Pl. az $1-2$; $3.\quad 2-7$; $8.\quad 4-5.\quad 6.$ gráf esetén az eredmény:
{1, 2, 3, 7, 8}
{4, 5}
{6}

8. Legrövidebb utak egy forrásból

8.1. Sor-alapú Bellman-Ford algoritmus

8.1.1. Írjuk meg Sor-alapú Bellman-Ford algoritmust – negatív kör figyelés nélkül – arra az esetre, ha a gráfot szomszédossági listákkal ábrázoljuk

a `G[1..n]` pointertömb segítségével! (Struktogram: QBF(G, d, P) eljárás.) $MT(n, m) \in O(n * m)$, ahol m az élek száma. $MW(n) \in O(1)$.

8.1.2. Próbáljuk meg a fenti Sor-alapú Bellman-Ford algoritmust kiegészíteni a negatív kör figyelésével! Alakítsuk át az eljárást függvénnyé, ami 0 értéket ad vissza, ha nem talált negatív kört, különben pedig a negatív kör egy csúcsának indexét. Ez utóbbi esetben `d[1..n]` és részben `P[1..n]` definiálatlan marad.

8.1.3. Szemléltessük a Sor-alapú Bellman-Ford algoritmus működését az alábbi gráfon[7] a d csúcsból indítva! Mutassuk be a `d[1..6]` (elérési költségek), az `e[1..6]` (élek száma) és a `P[1..6]` (π-értékek) tömbök, valamint a Q sor alakulását! A táblázat egy sora az egy csúcsból kimenő élek feldolgozása legyen! Nemdeterminisztikus esetekben mindig a kisebb indexű csúcsot dolgozzuk fel először! Rajzoljuk le a legrövidebb utak fáját, amit kaptunk!

a $\rightarrow$ b, 2; e, 5.	b $\rightarrow$ a, 2; e, 1.	c $\rightarrow$ b, 1.
d $\rightarrow$ a, 1; b, 4; c, 1.	e $\rightarrow$ c, -2.	f.

Hogyan változik az algoritmus futása, ha az (e,c) él súlyát eggyel csökkentjük?

8.2. Legrövidebb utak egy forrásból körmentes irányított gráfokra

8.2.1. Szemléltessük az előadásról ismert módon az alábbi gráfon a DAG legrövidebb utak egy forrásból algoritmus működését, ha a b csúcs a startcsúcs! Rajzoljuk le a legrövidebb utak fáját, amit kaptunk!

a $\rightarrow$ b, 2; e, 5.	b $\rightarrow$ c, 2; e, 3.	c $\rightarrow$ d, -1; e, 1.
d $\rightarrow$ e, 1; f, 4.	e. $\rightarrow$ f, 2	f.

8.3. Legrövidebb utak egy forrásból: Dijkstra algoritmus

8.3.1. Szemléltessük a Dijkstra algoritmus működését az alábbi gráfon, a `c` csúcsból indítva! Mutassuk be a d[1..6] (elérési költségek) és a π[1..6] (az adott pillanatig talált legrövidebb utak) tömbök alakulását! Minden sor mellett jelezzük, hogy az melyik csúcs kiterjesztéséből adódott! Nemdeterminisztikus esetekben mindig a kisebb indexű csúcsot dolgozzuk fel először! Rajzoljuk le a legrövidebb utak fáját, amit kaptunk!

a $\rightarrow$ b, 1; d, 1.	b $\rightarrow$ d, 5; e, 2.	c $\rightarrow$ b, 1; f, 1.
d.	e $\rightarrow$ b, 2; d, 1.	f $\rightarrow$ e, 1.

[7] $u \rightarrow v_1, w_1; \ldots v_n, w_n$. azt jelenti, hogy a gráf tartalmazza az $(u, v_1), \ldots (u, v_n)$ irányított éleket, sorban $w_1, \ldots w_n$ súlyokkal.

8.3.2. Az alábbi egyszerű gráfon szemléltessük a gyakorlatról ismert módon az a csúcsból indított Dijkstra algoritmust! Rajzoljuk le a legrövidebb utak fáját, amit kaptunk!

	a	b	c	d	e	f
a	0	4	∞	1	∞	∞
b	4	0	1	2	0	∞
c	∞	1	0	∞	1	1
d	1	2	∞	0	0	∞
e	∞	0	1	0	0	1
f	∞	∞	1	∞	1	0

9. (Legrövidebb) utak mindencsúcspárra

9.1. Floyd-Warshall algoritmus

9.1.1. Szemléltessük a Floyd-Warshall algoritmus működését az alábbi gráfon a $(D^{(0)}, \Pi^{(0)}), \ldots, (D^{(3)}, \Pi^{(3)})$ mátrix párok megadásával! Végül rajzoljuk le a legrövidebb utak fáit, amiket kaptunk! A gráf az alábbi csúcsmátrixszal adott.

	1	2	3
1	0	5	3
2	1	0	∞
3	3	1	0

9.1.2. Szemléltessük a Floyd-Warshall algoritmus működését az alábbi gráfon a $(D^{(0)}, \Pi^{(0)}), \ldots, (D^{(4)}, \Pi^{(4)})$ mátrix párok megadásával! Végül rajzoljuk le a legrövidebb utak fáit, amiket kaptunk! A gráf az alábbi csúcsmátrixszal adott.

	1	2	3	4
1	0	5	3	1
2	5	0	1	∞
3	3	1	0	1
4	1	∞	1	0

9.2. Gráfok tranzitív lezártja

9.2.1. Szemléltessük Warshall, gráfok tranzitív lezártját meghatározó algoritmusának működését az alábbi (csúcsmátrixszal adott) gráfon a $T^{(0)}, \ldots, T^{(3)}$ mátrixok megadásával!

	1	2	3
1	0	0	1
2	1	0	0
3	0	1	0

9.2.2. Szemléltessük Warshall, gráfok tranzitív lezártját meghatározó algoritmusának működését az alábbi (csúcsmátrixszal adott) gráfon a $T^{(0)}, \ldots, T^{(4)}$ mátrixok megadásával!

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

32 String Matching

32.1 The naive string-matching algorithm

32.1-1 Show the comparisons the naive string matcher makes for the pattern $P = 0001$ in the text $T = 000010001010001$.

32.1-2 Suppose that all characters in the pattern P are different. Show how to accelerate NAIVE-STRING-MATCHER to run in time $O(n)$ on an n-character text T.

32.2 The Rabin-Karp algorithm

32.2-1 Working modulo $q = 11$, how many spurious (i.e. false) hits does the Rabin-Karp matcher encounter in the text $T = 3141592653589793$ when looking for the pattern $P = 26$?

32.2-2 How would you extend the Rabin-Karp method to the problem of searching a text string for an occurrence of any of a given set of k patterns? Start by assuming that all k patterns have the same length. Then, generalise your solution to allow the patterns to have different lengths.

32.2-3 Show how to extend the Rabin-Karp method to handle the problem of looking for a given $m \times m$ pattern in an $n \times n$ array of characters ($1 \leq m \leq n$). (The pattern may be shifted vertically and horizontally, but it may not be rotated.)

32.4 The Knuth-Morris-Pratt algorithm

32.4-1 Compute $\pi[1..19]$ (also called $prefix$ or $next$ function) for the pattern $ababbabbabbababbabb$.

32.4.2a Compute $\pi[1..4]$ for the pattern $P = 0001$. Show the comparisons the Knuth-Morris-Pratt string matcher makes for this pattern in the text $T = 000010001010001$.

32.4.2b Compute $\pi[1..5]$ for the pattern $P = abaab$. Show the comparisons the Knuth-Morris-Pratt string matcher makes for this pattern in the text $T = aaababaabaababaab$.

32.4.2c Compute $\pi[1..6]$ for the pattern $P = aabaab$. Show the comparisons the Knuth-Morris-Pratt string matcher makes for this pattern in the text $T = aaabaabaabaababaab$.

32.4.2d Compute $\pi[1..6]$ for the pattern $P = babbab$. Show the comparisons the Knuth-Morris-Pratt string matcher makes for this pattern in the text $T = ababbabbababbababbabb$.

32.4.2e Compute $\pi[1..7]$ for the pattern $P = ABABAKI$.
Show the comparisons the Knuth-Morris-Pratt string matcher makes for this pattern in the text
$T = BABALABABATIBABABAKI$.

32.4-3 Explain how to determine the occurrences of pattern P in the text T by examining $\pi[1..|P|]$.

32.4-7 Give a linear-time algorithm to determine whether a text T is a cyclic rotation of another string T'. For example, *arc*, *rca*, and *car* are cyclic rotations of each other.

32.x The Quick Search algorithm

`<http://aszt.inf.elte.hu/~asvanyi/ds/Quick Searching.ppt>`

32.x.1 Compute $shift[0..1]$ for the pattern $P = 000$. Show the comparisons the Quick Search string matcher makes for this pattern in the text $T = 000010001010001$.

32.x.2 Compute $shift['A'..'F']$ for the pattern
$P = ABABACD$. Show the comparisons the Quick Search string matcher makes for this pattern in the text
$T = BABAEABABAFDBABABACD$.

32.x.3 Compute $shift['A','C','G','T']$ for the pattern
$P = GCAGAGAG$. Show the comparisons the Quick Search string matcher makes for this pattern in the text
$T = GCATCGCAGAGAGTATACAGTACG$.

DC Data Compression

DC.1 The Huffman coding

<http://en.wikipedia.org/wiki/Huffman_coding>

DC.1.1 We would like to compress the text

MATEKFELELETEMKETTESLETT

with Huffman coding. Draw the Huffman tree, give the Huffman code of each letter, the Huffman code of the text, and the length of the latest in bits. Show the letters of the original text in the Huffman code of it.

DC.1.2 Solve Exercise DC.1.1 with the following text.

EMESE MAI SMSE NEM NAIV MESE

DC.2 The Lempel–Ziv–Welch (LZW) algorithm

<http://en.wikipedia.org/wiki/Lempel-Ziv-Welch>

DC.2.1 We have compressed a text with the Lempel-Ziv-Welch algorithm. The text contains the letters 'A', 'B', and 'C'. Their codes are 1, 2, and 3 in turn. While building the dictionary, the first unused positive integer was selected for the code of each new word. In this way, we have received the following code.

1 2 4 3 5 6 9 7 1

Give the original text and the complete dictionary.

DC.2.2 Solve Exercise DC.2.1 with the following LZW code.

1 2 4 3 5 8 1 10 11 1