Algoritmusok és adatszerkezetek II. előadásjegyzet (részletek)

Ásványi Tibor – asvanyi@inf.elte.hu

2019. október 6.

Tartalomjegyzék

1. Bevezetés, Ajánlott irodalom

Kedves Hallgatók!

A vizsgára való készülésben elsősorban az előadásokon és a gyakorlatokon készített jegyzeteikre támaszkodhatnak. További ajánlott források:

Hivatkozások

[1] Ásványi Tibor, Algoritmusok és adatszerkezetek II.
Útmutatások a tanuláshoz, Tematika, AVL fák (2019)
http://aszt.inf.elte.hu/~asvanyi/ad/ad2jegyzet.pdf

[2] Ásványi Tibor, Algoritmusok II. gyakorló feladatok (2016)
http://aszt.inf.elte.hu/~asvanyi/ad/ad2feladatok.pdf

[3] 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.

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

[5] Koruhely Gábor, Szalay Richárd,
Algoritmusok és adatszerkezetek 2, 2015/16 tavaszi félév
(hallgatói jegyzet, lektorált és javított)
http://aszt.inf.elte.hu/~asvanyi/ad/

[6] 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

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

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

[9] Carl Burch, Ásványi Tibor, B+ fák
http://aszt.inf.elte.hu/~asvanyi/ad/B+ fa.pdf

[10] Ásványi Tibor, Algoritmusok és adatszerkezetek I. előadásjegyzet (2019)
http://aszt.inf.elte.hu/~asvanyi/ad/ad1jegyzet.pdf

[11] Wirth, N., Algorithms and Data Structures,
Prentice-Hall Inc., 1976, 1985, 2004.
magyarul: Algoritmusok + Adatstruktúrák = Programok, *Műszaki Könyvkiadó*, Budapest, 1982. ISBN 963 10 3858 0

Az itt következő előadásjegyzetekben bizonyos fejezetek még nem teljesek; egyelőre csak az AVL fákkal kapcsolatos részeket dolgoztam ki részletesen, az általános fákat részben, illetve a B+ fák témakörben angol nyelvű egyetemi tananyag fordítását bocsátom rendelkezésükre. Nagy segítség lehet még a felkészülésben a fent hivatkozott hallgatói jegyzet is [5]. Az előadásokon tárgyalt programok struktogramjait igyekeztem minden esetben megadni, a másolási hibák kiküszöbölése érdekében. E tananyagon kívül a megértést segítő ábrák találhatóak bőségesen az ajánlott segédanyagokban.

Ezúton szeretnék köszönetet mondani *Umann Kristófnak* az AVL fákról szóló fejezetben található szép, színvonalas szemléltető ábrák elkészítéséért, az ezekre szánt időért és szellemi ráfordításért!

A vizsgákon az elméleti kérdések egy-egy tétel bizonyos részleteire vonatkoznak. Lesznek még megoldandó feladatok, amik részben a tanult algoritmusok működésének szemléltetését, bemutatását, részben a szerzett ismeretek kreatív felhasználását kérik számon. Egy algoritmus, program, művelet bemutatásának mindig része a műveletigény elemzése.

Az előadások elsősorban a CLRS könyv [3] (ld. alább) angol eredetijének harmadik kiadását követik, de pl. a piros-fekete fák helyett az AVL fákat tárgyaljuk. (A CLRS könyv [3] érintett fejezeteiben a magyar és az angol változat között leginkább csak néhány jelölésbeli különbséget találtunk.)

Az egyes struktogramokat általában nem dolgozzuk ki az értékadó utasítások szintjéig. Az olyan implementációs részleteket, mint a listák és egyéb adatszerkezetek, adattípusok műveleteinek pontos kódja, a dinamikusan allokált objektumok deallokálása stb. általában az Olvasóra hagyjuk, hiszen ezekkel az előző félévben foglalkoztunk. Használni fogunk olyan absztrakt fogalmakat, mint a véges halmazok, sorozatok, gráfok. Ezeket, ha valamely eljárás paraméterlistáján szerepelnek – mint strukturált adatokat – minden esetben cím szerint vesszük át. (A skalárok változatlanul érték vagy cím szerint adódnak át, ahol az érték szerinti paraméterátvétel az alapértelmezett.)

A struktogramokban a „for” ciklusok (illetve a „for each” ciklusok) mintájára alkalmazni fogunk a ciklusfeltételek helyén pl. „$\forall x : P(x)$” alakú

kifejezéseket, ami azt jelenti, hogy a ciklusmagot a $P(x)$ állítás igazsáhalmazának minden x elemére végre kell hajtani, valamint „$\forall v \in V$” alakúakat, ami a „$\forall v : v \in V$” rövidítése. Ehhez $P(x)$ igazsághalmazának, illetve V-nek végesnek, és hatékonyan felsorolhatónak kell lennie. Ez ideális esetben azt jelenti, hogy a felsorolás műveletigénye az igazsághalmaz, illetve V méretétől lineárisan függ.

Ha a ciklus fejében „$i := u$ to v” alakú kifejezést látunk, akkor a ciklusmag az $u..v$ egész intervallum i elemeire szigorúan monoton növekvő sorrendben hajtódik végre.

Ha pedig a ciklus fejében „$i := u$ downto v” alakú kifejezést látunk, akkor a ciklusmag a $v..u$ egész intervallum i elemeire hajtódik végre, de szigorúan monoton csökkenő sorrendben.

A fentiek szerint az egyszerűbb programrészletek helyén gyakran szerepelnek majd magyar nyelvű utasítások, amiknek részletes átgondolását, esetleges kidolgozását, a korábban tanultak alapján, szintén az Olvasóra bízzuk. A struktogramokban az ilyen, formailag általában felszólító mondatok végéről a felkiáltójelet elhagyjuk (mivel az adott szövegkörnyezetben ez gyakran faktoriális függvényként lenne [félre]érthető).

2. Tematika

Minden tételhez: Hivatkozások: például a „[3] 8.2, 8.3, 8.4” jelentése: a [3] sorszámú szakirodalom adott (al)fejezetei.

1. Veszteségmentes adattömörítés. Naiv módszer. Huffman kód, kódfa, optimalitás. LZW (Lempel-Ziv-Welch) tömörítés. [5]

2. Általános fák, bináris láncolt reprezentáció, bejárások ([1]; [11] 4.7 bevezetése). Egyéb reprezentációk? (HF)

3. AVL fák és műveleteik: kiegyensúlyozási sémák, programok. Az AVL fa magassága ([1], [4] 12; [11] 4.5).

4. B+ fák és műveleteik [9].

5. Elemi gráf algoritmusok ([3] 22). Gráf ábrázolások (representations of graphs).

A szélességi gráfkeresés (breadth-first search: BFS). A szélességi gráfkeresés futási ideje (the run-time analysis of BFS). A legrövidebb utak (shortest paths). A szélességi feszítőfa (breadth-first tree). HF: A szélességi gráfkeresés megvalósítása a klasszikus gráf ábrázolások esetén; hatékonyság.

A mélységi gráfkeresés (depth-first search: DFS). Mélységi feszítő erdő (depth-first forest). A gráf csúcsainak szín és időpont címkéi (colors and timestamps of vertexes). Az élek osztályozása (classification of edges, its connections with the colors and timestamps of the vertexes). A mélységi gráfkeresés futási ideje (the run-time analysis of DFS). Topologikus rendezés (topological sort). HF: A mélységi gráfkeresés és a topologikus rendezés megvalósítása a klasszikus gráf ábrázolások esetén; hatékonyság.

6. Minimális feszítőfák (Minimum Spanning Trees: MSTs). Egy általános algoritmus (A general algorithm). Egy tétel a biztonságos élekről és a minimális feszítőfákról (A theorem on safe edges and MSTs). Prim és Kruskal algoritmusai (The algorithms of Kruskal and Prim). A futási idők elemzése (Their run-time analysis). HF: A Prim algoritmus implementációja a két fő gráfábrázolás és a szükséges prioritásos sor különböző megvalósításai esetén (The implementations of the algorithm of Prim with respect to the main graph representations and representations of the priority queue).

7. Legrövidebb utak egy forrásból (Single-Source Shortest Paths). A legrövidebb utak fája (Shortest-paths tree). Negatív körök (Negative cycles). Közelítés (Relaxation).

A sor-alapú (Queue-based) Bellman-Ford algoritmus. A menet (pass) fogalma. Futási idő elemzése. Helyessége. A legrövidebb út kinyomtatása.

Legrövidebb utak egy forrásból, körmentes irányított gráfokra. (DAG shortest paths.) Futási idő elemzése. Helyessége.

Dikstra algoritmusa. Helyessége. Fontosabb implementációi a két fő gráfábrázolás és a szükséges prioritásos sor különböző megvalósításai esetén. A futási idők elemzése.

8. Legrövidebb utak minden csúcspárra (All-Pairs Shortest Paths). A megoldás ábrázolása a (D, Π) mátrix-párral. HF: Adott csúcspárra a legrövidebb út kinyomtatása.

A Floyd-Warshall algoritmus és a $(D^{(k)}, \Pi^{(k)})$ mátrix párok. A futási idő elemzése. Összehasonlítás a Dijkstra algoritmus, illetve (HF:) a sor-alapú Bellman-Ford algoritmus $|G.V|$-szeri végrehajtásával.

Irányított gráf tranzitív lezártja (Transitive closure of a directed graph) és a $T^{(k)}$ mátrixok. Az algoritmus és futási ideje. HF: összehasonlítás a szélességi keresés $|G.V|$-szeri végrehajtásával.

9. Mintaillesztés (String Matching). Egy egyszerű mintaillesztő algoritmus (The naive string-matching algorithm). A futási idő elemzése.

A Quick Search algoritmus. Inicializálása. A futási idő elemzése.

A Knuth-Morris-Pratt algoritmus. Inicializálása. A futási idő elemzése.

3. AVL fák

Az AVL fák kiegyensúlyozásának szabályai megtalálhatók az 1-6. ábrákon.

Az *AVL fák* magasság szerint kiegyensúlyozott bináris keresőfák. Egy bináris fa *magasság szerint kiegyensúlyozott*, ha minden csúcsa kiegyensúlyozott. (Mostantól *kiegyensúlyozott* alatt *magasság szerint kiegyensúlyozottat* értünk.) Egy bináris fa egy $(*p)$ csúcsa *kiegyensúlyozott*, ha a csúcs $(p \to b)$ egyensúlyára (balance) $|p \to b| \leq 1$. A $(*p)$ csúcs egyensúlya definíció szerint:

$$p \to b = h(p \to right) - h(p \to left)$$

Az AVL fákat láncoltan reprezentáljuk. A csúcsokban a b egyensúly attribútumot expliciten tároljuk. (Egy másik lehetőség a két részfa magasságainak tárolása lenne; egy harmadik, hogy csak az aktuális részfa magasságát tároljuk.) A Node osztályt tehát a következőképpen módosítjuk:

Node
+ $key : \mathcal{T}$ // $\mathcal{T}$ is some known type + $b : -1..1$ // the balance of the node + $left, right$: Node*
+ Node() { $left := right := \oslash$; $b := 0$ } // create a tree of a single node + Node(x:$\mathcal{T}$) { $left := right := \oslash$; $b := 0$; $key := x$ }

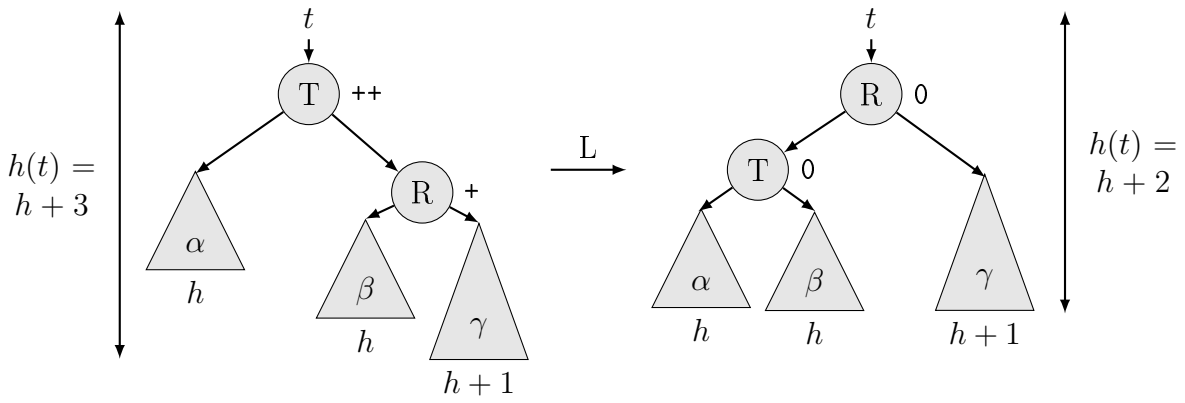
Egyelőre részletes bizonyítás nélkül közöljük az alábbi eredményt:

Tétel: Tetszőleges n csúcsú nemüres AVL fa h magasságára:

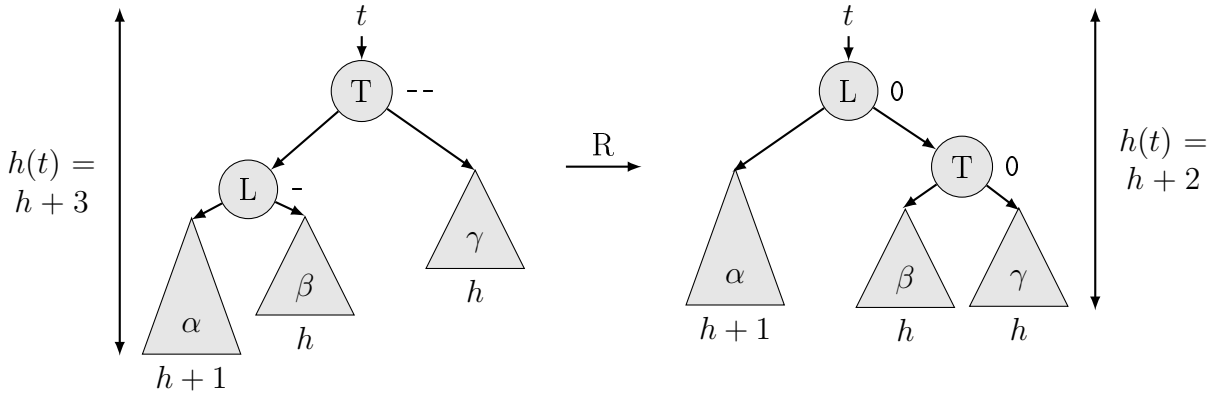
$$\lfloor \lg n \rfloor \leq h \leq 1,45 \lg n, \quad \text{azaz} \quad h \in \Theta(\lg n)$$

A bizonyítás vázlata: Először a h magasságú, nemüres KBF-ek (kiegyensúlyozott, bináris fák) n méretére adunk alsó és felső becslést. Az $n < 2^{h+1}$ becslésből azonnal adódik $\lfloor \lg n \rfloor \leq h$. Másrészt meghatározzuk a h mélységű, legkisebb méretű KBF-ek csúcsainak f_h számát. Erre kapjuk, hogy $f_0 = 1$, $f_1 = 2$, $f_h = 1 + f_{h-1} + f_{h-2} \quad (h \geq 2)$. Ezért az ilyen fákat *Fibonacci fáknak hívjuk.* Mivel tetszőleges h magasságú KBF n méretére $n \geq f_h$, némi matematikai ügyességgel kaphatjuk a $h \leq 1,45 \lg n$ egyenlőtlenséget.

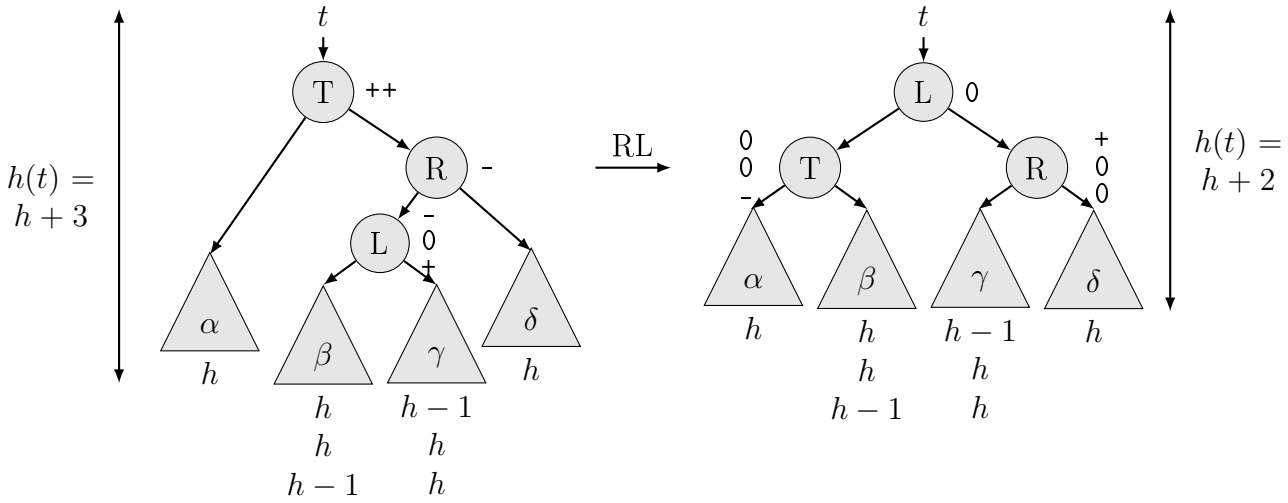
Mivel az AVL fák magassága $\Theta(\lg n)$, ezért a bináris keresőfák search(t, k), min(t) és max(t) függvényeire t AVL fa esetén automatikusan
$MT(n) \in \Theta(\lg n)$, ahol $n = |t|$.



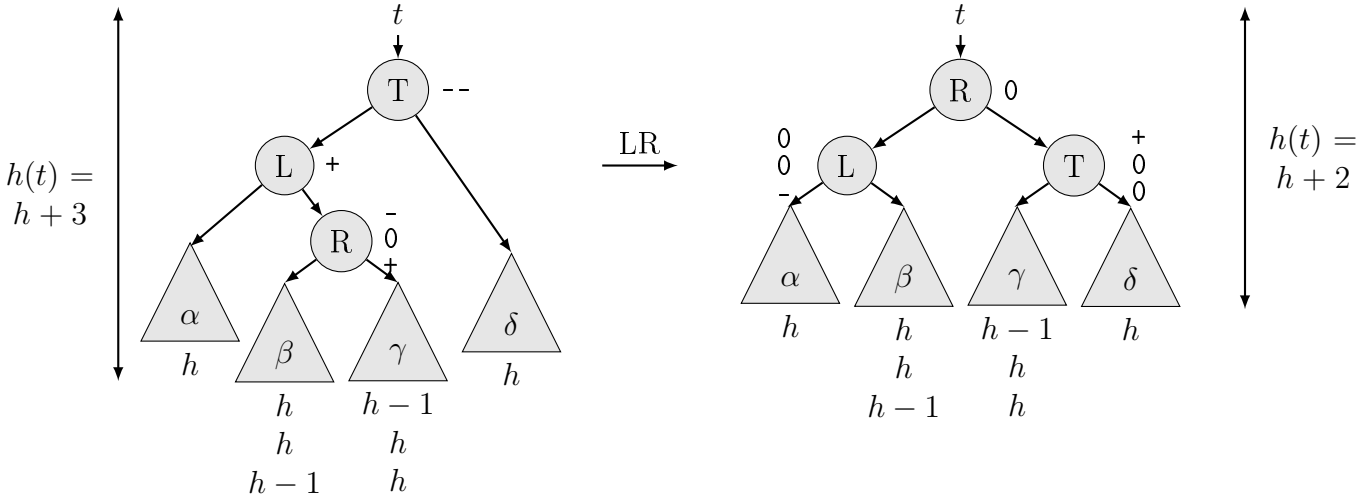
1. ábra. $(++, +)$ forgatás.



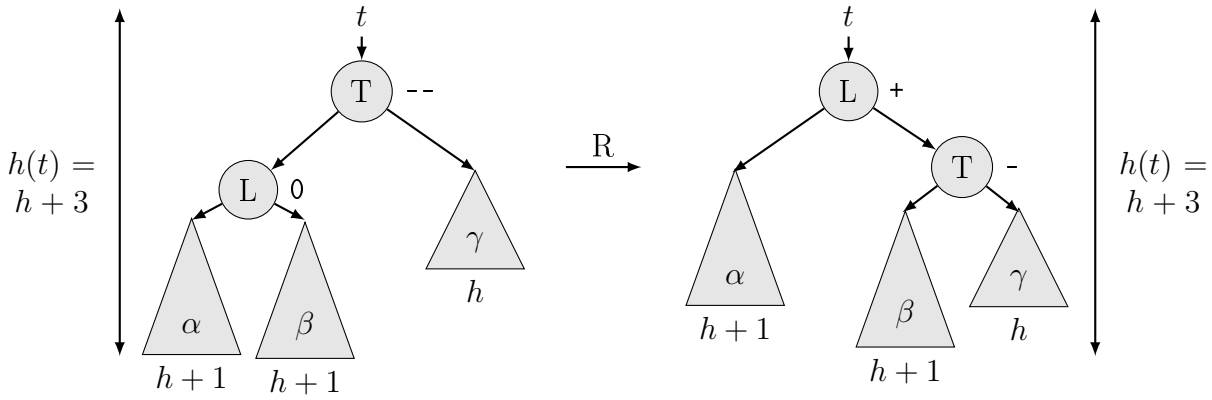
2. ábra. $(--, -)$ forgatás.



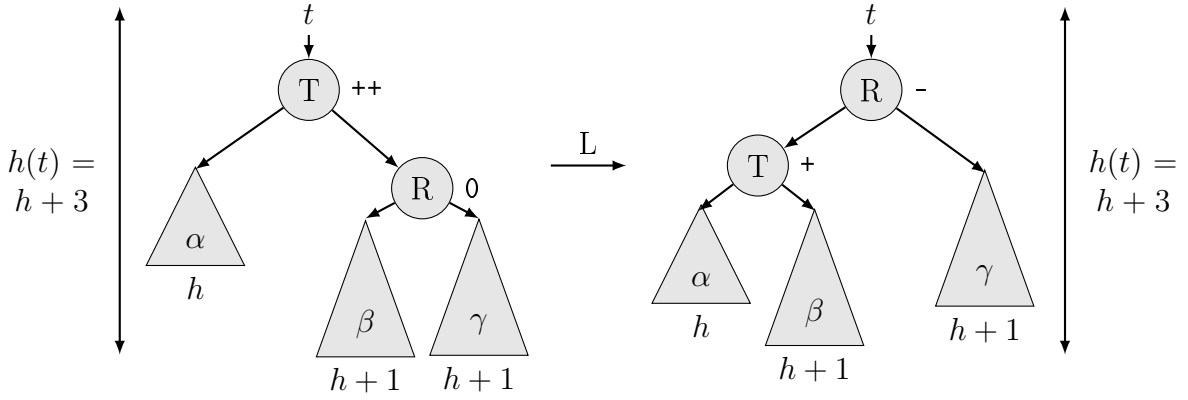
3. ábra. $(++, -)$ forgatás.



4. ábra. $(--, +)$ forgatás.



5. ábra. $(--, 0)$ forgatás.



6. ábra. $(++, 0)$ forgatás.

Az insert(t, k), a del(t, k), a remMin$(t, minp)$ és a remMax$(t, maxp)$ eljárások azonban változtatják a fa alakját. Így elromolhat a kiegyensúlyozottság, és már nem garantált a fenti műveletigény. Ennek elkerülésére ezeket az eljárásokat úgy módosítjuk, hogy minden egyes rekurzív eljáráshívás után ellenőrizni fogjuk, hogyan változott a megfelelő részfa magassága, és ez hogyan befolyásolta a felette levő csúcs kiegyensúlyozottságát, szükség esetén helyreállítva azt. Ez minden szinten legfeljebb konstans mennyiségű extra műveletet fog jelenteni, és így a kiegészített eljárások futási ideje megtartja az $MT(n) \in \Theta(\lg n)$ (ahol $n = |t|$) nagyságrendet.

A példákhoz a *(bal_részfa gyökér jobb_részfa)* jelölést használjuk, ahol az üres részfákat elhagyjuk, és a könnyebb olvashatóság kedvéért [] és {} zárójeleket is alkalmazunk.

Például a {[2]4[(6)8(10)]} bináris keresőfa gyökere a 4; bal részfája a [2], ami egyetlen levélcsúcsból (és annak üres bal és jobb részfáiból) áll; jobb részfája a [(6)8(10)], aminek gyökere a 8, bal részfája a (6), jobb részfája a (10). *Az így ábrázolt fák inorder bejárása a zárójelek elhagyásával adódik.* A belső csúcsok egyensúlyait *kb* formában jelöljük, ahol k a csúcsot azonosító kulcs, b pedig a csúcs egyensúlya, a 0:○, 1:+, 2:++, -1:−, -2:−− megfeleltetéssel. Mivel a levélcsúcsok egyensúlya mindig nulla, a leveleknél nem jelöljük az egyensúlyt. A fenti AVL fa például a megfelelő egyensúlyokkal a következő: {[2]4+[(6)8○(10)]}

Az AVL fák műveletei során a kiegyensúlyozatlan részfák kiegyensúlyozásához *forgatásokat* fogunk használni. Az alábbi forgatási sémákban görög kisbetűk jelölik a részfákat (amelyek minden esetben AVL fák lesznek), latin nagybetűk pedig a csúcsok kulcsait (1. és 2. ábrák, az egyensúlyok nélkül):

Balra forgatás (Left rotation): $[\alpha\ \mathrm{T}\ (\beta\ \mathrm{R}\ \gamma)] \to [(\alpha\ \mathrm{T}\ \beta)\ \mathrm{R}\ \gamma]$
Jobbra forgatás (Right rotation): $[(\alpha\ \mathrm{L}\ \beta)\ \mathrm{T}\ \gamma] \to [\alpha\ \mathrm{L}\ (\beta\ \mathrm{T}\ \gamma)]$

Vegyük észre, hogy a fa inorder bejárását egyik forgatás sem változtatja, így a bináris keresőfa tulajdonságot is megtartják. Mint látni fogjuk, a kiegyensúlyozatlan részfák kiegyensúlyozása minden esetben egy vagy két forgatásból áll. *Ezért a bináris keresőfa tulajdonságot a kiegyensúlyozások is megtartják.*

3.1. AVL fák: beszúrás

Nézzük most először a bináris keresőfák insert(t, k) eljárását!

A {[2]4+[(6)8○(10)]} AVL fából
az 1 beszúrásával az {[(1)2−]4○[(6)8○(10)]} AVL fa adódik,
a 3 beszúrásával pedig az {[2+(3)]4○[(6)8○(10)]} AVL fa.

A fenti {[2]4+[(6)8∘(10)]} AVL fából
a 7 beszúrásával azonban a {[2]4++[(6+{7})8−(10)]} fát kapjuk,
a 9 beszúrásával pedig a {[2]4++[(6)8+({9}10−)]} fát.
Mindkettő bináris keresőfa, de már nem AVL fa, mivel a 4++ csúcs nem kiegyensúlyozott.

A legutolsó esetben a bináris keresőfa könnyen kiegyensúlyozható, ha meggondoljuk, hogy az a következő sémára illeszkedik, amiben görög kisbetűk jelölik a kiegyensúlyozott részfákat, latin nagybetűk pedig a csúcsok kulcsait:
[α T++ (β R+ γ)], ahol T=4, α=[2], R=8, β=(6) és γ=({9}10−).

A kiegyensúlyozáshoz szükséges transzformáció a fa *balra forgatása* (1. ábra):

$$[\ \alpha\ \mathrm{T}{+}{+}\ (\beta\ \mathrm{R}{+}\ \gamma)\ ] \to [\ (\alpha\ \mathrm{T}{\circ}\ \beta)\ \mathrm{R}{\circ}\ \gamma\ ]$$

A fenti példán ez a következőt jelenti:
{[2]4++[(6)8+({9}10−)]} → { [(2)4∘(6)] 8∘ [(9)10−] }

A transzformáció helyességének belátásához bevezetjük a $h = h(\alpha)$ jelölést. Ebből a kiinduló fára a T++ és R+ egyensúlyok miatt
$h((\beta\ \mathrm{R}\ \gamma)) = h{+}2$, $h(\gamma) = h{+}1$ és $h(\beta) = h$ adódik, innét pedig az eredmény fára $h(\alpha) = h = h(\beta)$ miatt T∘; továbbá $h((\alpha\ \mathrm{T}{\circ}\ \beta)) = h + 1 = h(\gamma)$ miatt R∘ adódik.

Az eredeti {[2]4+[(6)8∘(10)]} AVL fába a 7 beszúrásával adódó
{ [2]4++[(6+{7})8−(10)] } kiegyensúlyozatlan bináris keresőfa kiegyensúlyozásával pedig {[(2)4−] 6∘ [(7)8∘(10)]} adódik, amire az

$$\{\ \alpha\ \mathrm{T}{+}{+}\ [(\beta\ \mathrm{L}{-}{\circ}{+}\ \gamma)\ \mathrm{R}{-}\ \delta]\ \} \to \{\ [\alpha\ \mathrm{T}{\circ}{\circ}{-}\ \beta]\ \mathrm{L}{\circ}\ [\gamma\ \mathrm{R}{+}{\circ}{\circ}\ \delta]\ \}$$

séma (3. ábra) szolgál, ahol α, β, γ és δ AVL fák, és a három jelből álló egyensúlyok sorban három esetet jelentenek úgy, hogy a bal oldalon az i-edik alternatívának a jobb oldalon is az i-edik eset felel meg ($i \in \{1; 2; 3\}$).

A fenti séma a példában T=4, α = [2], R=8, δ = (10), L=6, + egyensúllyal (harmadik eset), $\beta = \oslash$ és $\gamma = \{7\}$ helyettesítéssel alkalmazható.

A transzformációt *kettős forgatásnak* is tekinthetjük: Az
{ α T [(β L γ) R δ] } fára először az R csúcsnál alkalmaztunk egy jobbra forgatást, aminek eredményeképpen az { α T [β L (γ R δ)] } bináris keresőfát kaptuk, majd az eredmény fát balra forgattuk, ami az { [α T β] L [γ R δ] } AVL fát eredményezte.

A kettős forgatás helyességének ellenőrzéséhez kiszámítjuk az eredmény fa egyensúlyait. Most is bevezetjük a $h = h(\alpha)$ jelölést. Mivel a kettős forgatás

előtti fában T++, azért $h([(\beta\ \mathrm{L}\ \gamma)\ \mathrm{R}\ \delta]) = h+2$. Innét R− miatt $h((\beta\ \mathrm{L}\ \gamma)) = h+1$ és $h(\delta) = h$. Most L lehetséges egyensúlyai szerint három eset van.
(1) L− esetén $h(\beta) = h$ és $h(\gamma) = h-1 \Rightarrow$ az eredmény fában T○ és R+.
(2) L○ esetén $h(\beta) = h$ és $h(\gamma) = h \Rightarrow$ az eredmény fában T○ és R○.
(3) L+ esetén $h(\beta) = h-1$ és $h(\gamma) = h \Rightarrow$ az eredmény fában T− és R○.
Mindhárom esetben $h([\alpha\ \mathrm{T}\ \beta]) = h+1 = h([\gamma\ \mathrm{R}\ \delta])$, így L○.
Ezzel beláttuk a kettős forgatási séma helyességét.

Vegyük észre, hogy a fentiek alapján, az L csúcsnak a kettős forgatás előtti egyensúlyát s-sel, a T és a R csúcsoknak pedig a kettős forgatás utáni egyensúlyát rendre s_t-vel, illetve s_r-vel jelölve a következő összefüggéseket írhatjuk fel:

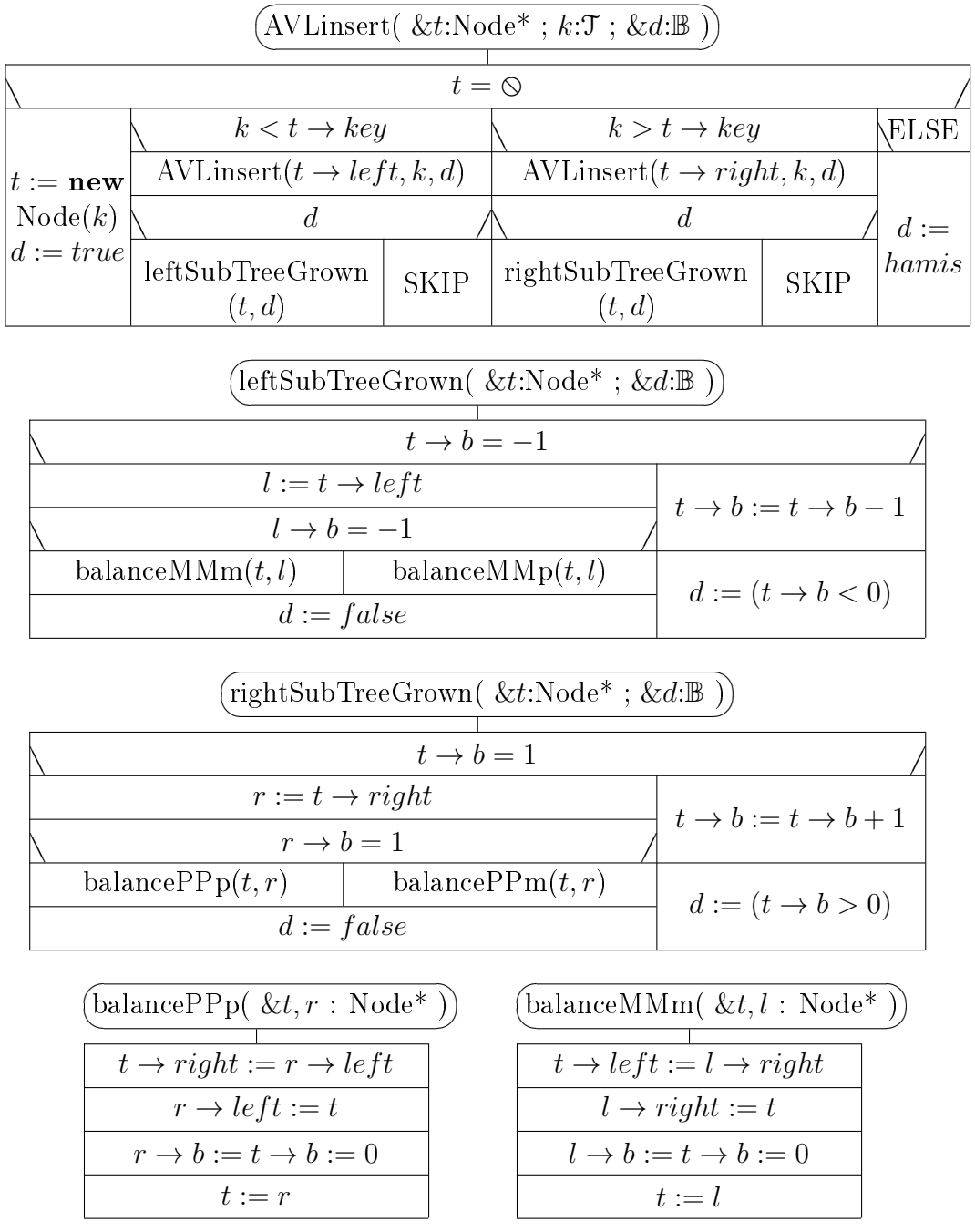
$$s_t = -\lfloor (s+1)/2 \rfloor \quad \text{és} \quad s_r = \lfloor (1-s)/2 \rfloor$$

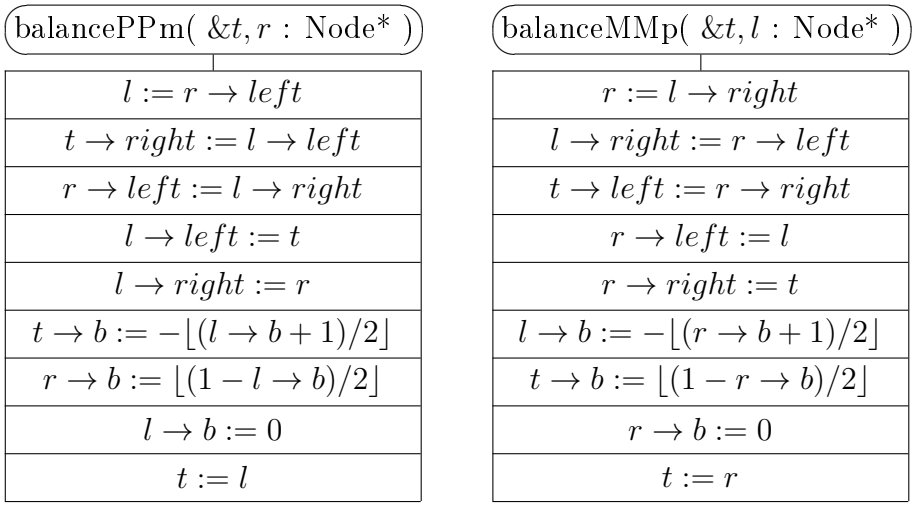
Az eddigi két kiegyensúlyozási séma mellett természetes módon adódnak ezek tükörképei, a forgatások előtt a T−− csúccsal a gyökérben. Ezek tehát a következők (2. és 4. ábra):

$$[\ (\alpha\ \mathrm{L}-\ \beta)\ \mathrm{T}--\ \gamma\ ] \to [\ \alpha\ \mathrm{L}\circ\ (\beta\ \mathrm{T}\circ\ \gamma)\ ]$$
$$\{\ [\alpha\ \mathrm{L}+\ (\beta\ \mathrm{R}-\circ+\ \gamma)]\ \mathrm{T}--\ \delta\ \} \to [\ (\alpha\ \mathrm{L}\circ\circ-\ \beta)\ \mathrm{R}\circ\ (\gamma\ \mathrm{T}+\circ\circ\ \delta)\ ]$$
$$s_l = -\lfloor (s+1)/2 \rfloor \quad \text{és} \quad s_t = \lfloor (1-s)/2 \rfloor$$

ahol s az R csúcs kettős forgatás előtti egyensúlya; s_l, illetve s_t pedig rendre az L és T csúcsoknak a kettős forgatás utáni egyensúlya

Eddig nem beszéltünk arról, hogy az AVL fában az új csúcs beszúrása után mely csúcsoknak és milyen sorrendben számoljuk újra az egyensúlyát, sem hogy mikor ütemezzük be a kiegyensúlyozást. Csak a beszúrás nyomvonalán visszafelé haladva kell újraszámolnunk az egyensúlyokat, és csak itt kell kiegyensúlyoznunk, ha kiegyensúlyozatlan csúcsot találunk. Így a futási idő a fa magasságával arányos marad, ami AVL fákra $O(\lg n)$. Most tehát részletezzük a beszúró és kiegyensúlyozó algoritmus működését. Ennek során a fa magassága vagy eggyel növekszik ($d = true$), vagy ugyanannyi marad ($d = false$).


AVLinsert(&t:Node* ; k:ℑ ; &d:𝔹)
t = ⊘
t := new Node(k)
d := true
k < t → key
k > t → key
ELSE
AVLinsert(t → left, k, d)
AVLinsert(t → right, k, d)
d
d
d := hamis
leftSubTreeGrown (t, d)
SKIP
rightSubTreeGrown (t, d)
SKIP
leftSubTreeGrown(&t:Node* ; &d:𝔹)
t → b = −1
l := t → left
l → b = −1
t → b := t → b − 1
balanceMMm(t, l)
balanceMMp(t, l)
d := false
d := (t → b < 0)
rightSubTreeGrown(&t:Node* ; &d:𝔹)
t → b = 1
r := t → right
r → b = 1
t → b := t → b + 1
balancePPp(t, r)
balancePPm(t, r)
d := false
d := (t → b > 0)
balancePPp(&t, r : Node*)
t → right := r → left
r → left := t
r → b := t → b := 0
t := r
balanceMMm(&t, l : Node*)
t → left := l → right
l → right := t
l → b := t → b := 0
t := l



Az AVL fába való beszúrást röviden összefoglalva:
1. Megkeressük a kulcs helyét a fában.
2. Ha a kulcs benne van a fában, STOP.
3. Ha a kulcs helyén egy üres részfa található, beszúrunk az üres fa helyére egy új, a kulcsot tartalmazó levélcsúcsot. Ez a részfa eggyel magasabb lett.
4. Ha a gyökércsúcsnál vagyunk, STOP. Különben egyet fölfelé lépünk a keresőfában. Mivel az a részfa, amiből fölfele léptünk, eggyel magasabb lett, az aktuális csúcs egyensúlyát megfelelőképp módosítjuk. (Ha a jobb részfa lett magasabb, hozzáadunk az egyensúlyhoz egyet, ha a bal, levonunk belőle egyet.)
5. Ha az aktuális csúcs egyensúlya 0 lett, akkor az aktuális csúcshoz tartozó részfa alacsonyabb ága hozzánőtt a magasabbikhoz, tehát az aktuális részfa most ugyanolyan magas, mint a beszúrás előtt volt, és így egyetlen más csúcs egyensúlyát sem kell módosítani: STOP.
6. Ha az aktuális csúcs új egyensúlya 1 vagy -1, akkor előtte 0 volt, ezért az aktuális részfa magasabb lett eggyel. Ekkor a 4. ponttól folytatjuk.
7. Ha az aktuális csúcs új egyensúlya 2 vagy -2[1], akkor a hozzá tartozó részfát ki kell egyensúlyozni. A *kiegyensúlyozás után az aktuális részfa visszanyeri a beszúrás előtti magasságát*, ezért már egyetlen más csúcs egyensúlyát sem kell módosítani: STOP.

Az az állítás, hogy ebben az algoritmusban a *kiegyensúlyozás után az aktuális részfa visszanyeri a beszúrás előtti magasságát*, még igazolásra vár. Azt az esetet nézzük meg, amikor a kiegyensúlyozandó részfa gyökere T++. A T−−

[1] A 2 és -2 eseteket a struktogramban nem számoltuk ki expliciten, hogy az egyensúly tárolására elég legyen két bit.

eset hasonlóan fontolható meg. A T++ esethez tartozó kiegyensúlyozási sémák (1. és 3. ábra):

$$[\ \alpha\ \mathrm{T}{+}{+}\ (\beta\ \mathrm{R}{+}\ \gamma)\ ] \to [(\alpha\ \mathrm{T}{\circ}\ \beta)\ \mathrm{R}{\circ}\ \gamma]$$
$$\{\ \alpha\ \mathrm{T}{+}{+}\ [(\beta\ \mathrm{L}{-}{\circ}{+}\ \gamma)\ \mathrm{R}{-}\ \delta]\ \} \to \{\ [\alpha\ \mathrm{T}{\circ}{\circ}{-}\ \beta]\ \mathrm{L}{\circ}\ [\gamma\ \mathrm{R}{+}{\circ}{\circ}\ \delta]\ \}$$

Először belátjuk, hogy ha a T csúcs jobboldali R gyereke + vagy − súlyú, akkor a fenti sémák közül a megfelelő alkalmazható: Mivel a beszúró algoritmus a fában a beszúrás helyétől egyesével fölfele lépked, és az első kiegyensúlyozatlan csúcsnál azonnal kiegyensúlyoz, ez alatt nincs kiegyensúlyozatlan csúcs, azaz az α, β és γ, illetve a δ részfák is kiegyensúlyozottak, ez pedig éppen a fenti forgatások feltétele, amellett, hogy bináris keresőfát akarunk kiegyensúlyozni, amit viszont a kiegyensúlyozás nélküli beszúró algoritmus garantál.

Most még be kell látni, hogy a fenti sémák minden esetet lefednek, azaz R+ vagy R−: egyrészt, R nem lehet a beszúrás által létrehozott új csúcs, mert különben T-nek a beszúrás előtti jobboldali részfája üres lett volna, tehát most nem lehetne T++. Másrészt, ha a fölfele lépkedés során nulla egyensúlyú csúcs áll elő, akkor a fölötte levő csúcsok egyensúlya már nem módosul, így kiegyensúlyozatlan csúcs sem állhat elő. Márpedig most T++. Így tehát az új csúcstól T-ig fölfelé vezető úton minden csúcs, azaz R egyensúlya is + vagy −.

Most belátjuk, hogy a kiegyensúlyozások visszaállítják a részfa beszúrás előtti magasságát. A T++ kiegyensúlyozatlan csúcs a beszúrás előtt kiegyensúlyozott volt. Mivel a beszúrás, a beszúrás helyétől kezdve T-ig fölfelé vezető úton mindegyik részfa magasságát pontosan eggyel növelte, a beszúrás előtt T+ volt. Ezért a beszúrás előtt, $h = h(\alpha)$ jelöléssel, a T gyökerű részfa $h+2$ magas volt. A beszúrás után tehát T++ lett, és így a T gyökerű részfa $h+3$ magas lett. A beszúrás utáni, de még a kiegyensúlyozás előtti állapotot tekintve a továbbiakban megkülönböztetjük a T jobboldali R gyerekére a R+ és a R− eseteket.

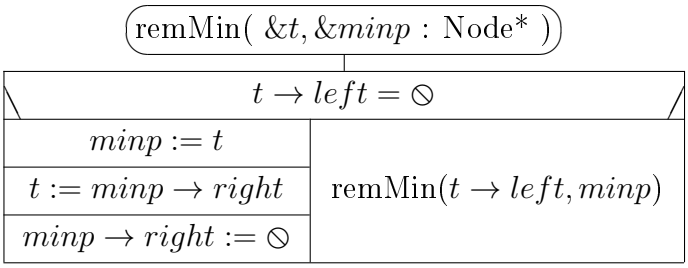
R+ esetén már láttuk, hogy $h(\alpha) = h = h(\beta)$ és $h(\gamma) = h+1$. Ezért a kiegyensúlyozás után $h([(\alpha\ \mathrm{T}\ \beta)\ \mathrm{R}\ \gamma]) = h+2$.

R− esetén pedig már láttuk, hogy $h(\alpha) = h = h(\delta)$ és $h([\beta\ \mathrm{L}\ \gamma]) = h+1$. Ezért $h(\beta), h(\gamma) \leq h$. Így a kiegyensúlyozás után $h(\{[\alpha\ \mathrm{T}\ \beta]\ \mathrm{L}\ [\gamma\ \mathrm{R}\ \delta]\}) = h+2$.

Ezzel beláttuk, hogy a kiegyensúlyozások mindkét esetben visszaállítják a részfa beszúrás előtti magasságát, mégpedig úgy, hogy a beszúrás által eggyel megnövelt magasságot eggyel csökkentik. Szimmetria okokból ez hasonlóan látható be T−− esetén is, figyelembe véve a L− és a L+ eseteket.

3.2. AVL fák: a remMin($t, minp$) eljárás

Bináris keresőfákra a remMin eljárás:

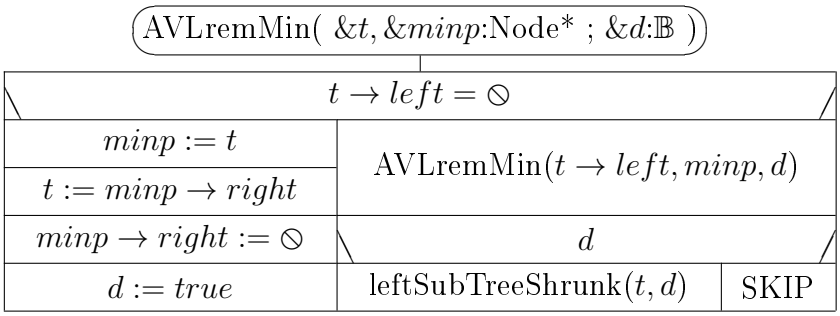


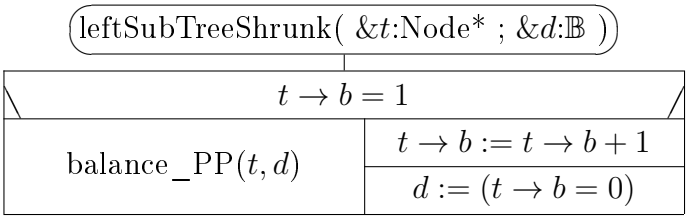
Ezt például a { [2]4+[(6)8∘(10)] } AVL fára alkalmazva, a $minp \to key = 2$ eredmény mellett, a { 4++[(6)8∘(10)] } kiegyensúlyozatlan bináris keresőfa adódna. Látjuk, hogy a kiegyensúlyozatlan 4++ csúcs jobboldali gyereke a 8∘. Ennek az esetnek a kezelésére új kiegyensúlyozási sémát kell alkalmaznunk. Szerencsére egy balra forgatás, a megfelelő egyensúly beállítások mellett most is megoldja a problémát (6. ábra):

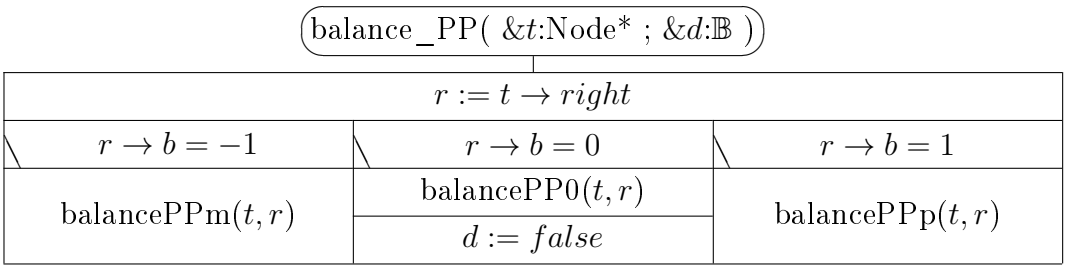
$$\{\ \alpha\ \mathrm{T{+}{+}}[\beta\ \mathrm{R}\circ\ \gamma]\ \} \to \{\ [\alpha\ \mathrm{T{+}}\ \beta]\ \mathrm{R}{-}\ \gamma\ \}.$$

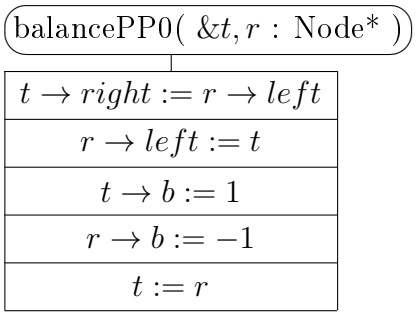
T++ miatt $h = h(\alpha)$ jelöléssel nyilván $h(\beta) = h + 1 = h(\gamma)$, így a forgatás után az egyensúlyokat a fenti séma szerint kell beállítani. A fa magassága a forgatás előtt és után is $h + 3$ lesz, ez a fajta kiegyensúlyozás tehát az eddigiekkel szemben *nem* csökkenti az aktuális részfa magasságát.

Ezután az AVL fákra alkalmazott, a megfelelő kiegyensúlyozásokkal kiegészített AVLremMin eljárás pl. a következő lehet (d most azt jelöli, hogy csökkent-e az aktuális részfa magassága):









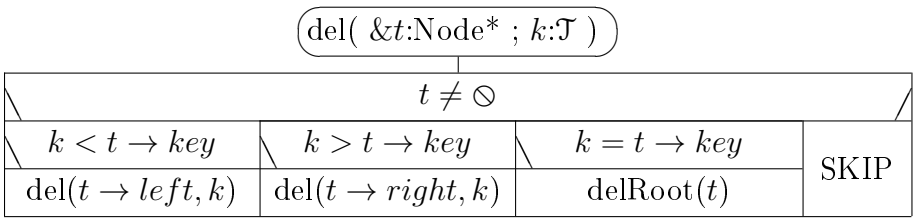
Az algoritmus helyessége hasonlóan gondolható meg, mint a beszúrás esetén. Lényeges különbség azonban, hogy ott minden beszúrást csak egyetlen kiegyensúlyozás követ, míg itt előfordulhat, hogy a minimális csúcs eltávolítása után, minden felette lévő szinten ki kell egyensúlyozni.

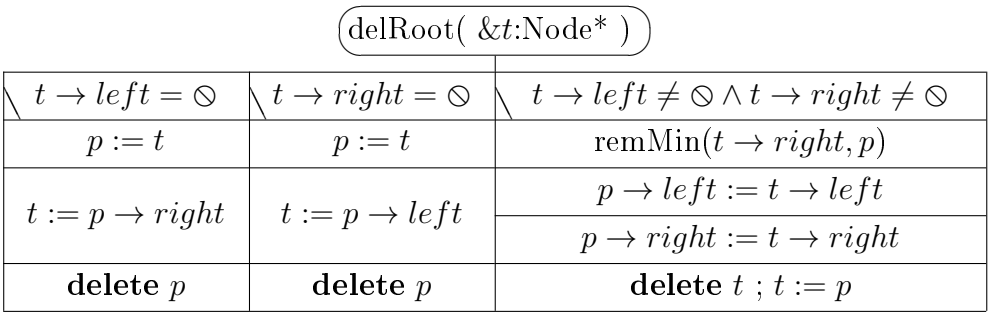
3.1. Feladat. *Mutassunk ilyen, legalább négy magasságú fát!*

3.2. Feladat. *Írjuk meg az AVLremMin(t, minp, d) eljárás mintájára az AVLremMax(t, maxp, d) eljárást és segédeljárásait!*

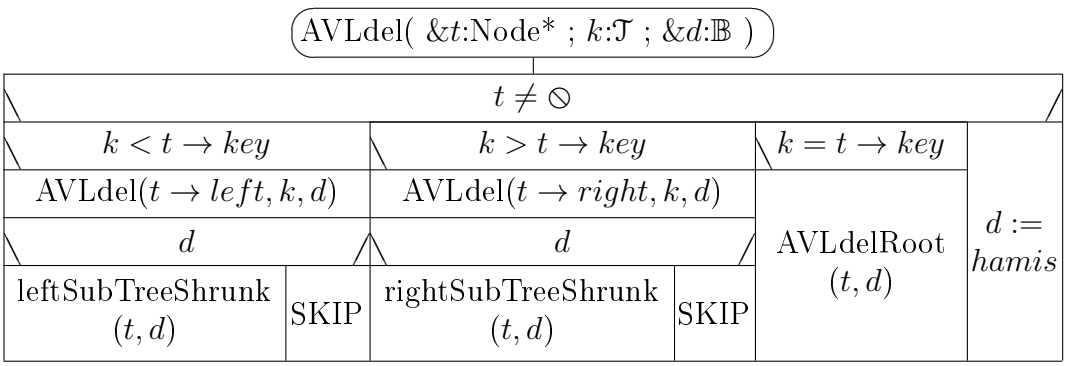
3.3. AVL fák: törlés

Bináris keresőfákra a del(t, k) és a delRoot(t) eljárások:





Ezeket fogjuk most az AVLremMin$(t, minp, d)$ eljárás mintájára kiegészíteni a d paraméterrel; majd a rekurzív töröl hívásokból, valamint az AVLremMin eljárásból való visszatérés után megkérdezzük, csökkent-e az aktuális részfa magassága. Ha igen, az AVLremMin$(t, minp, d)$ eljárás mintájára módosítjuk a $t \to b$ értéket, ha kell, kiegyensúlyozunk, és ha kell, d-t beállítjuk.



AVLdelRoot(&t:Node* ; &d:$\mathbb{B}$)

$t \to left = \oslash$	$t \to right = \oslash$	$t \to left \neq \oslash \wedge t \to right \neq \oslash$	
$p := t$	$p := t$	rightSubTreeMinToRoot(t, d)	
$t := p \to right$	$t := p \to left$		
delete p	**delete** p	d	
$d := true$	$d := true$	rightSubTreeShrunk(t, d)	SKIP

rightSubTreeMinToRoot(&t:Node* ; &d:$\mathbb{B}$)

AVLremMin$(t \to right, p, d)$
$p \to left := t \to left$; $p \to right := t \to right$; $p \to b := t \to b$
delete t ; $t := p$

Itt is lehetséges, hogy több szinten, a legrosszabb esetben akár minden szinten is ki kell egyensúlyozni. Mivel azonban egyetlen kiegyensúlyozás sem tartalmaz se rekurziót, se ciklust, és ezért konstans számú eljáráshívásból áll, ez sem itt, sem az AVLremMin eljárásnál nem befolyásolja a futási időnek az AVLinsert eljárásra is érvényes $MT(n) \in \Theta(\lg n)$ (ahol $n = |t|$) nagyságrendjét.

3.3. Feladat. *A leftSubTreeShrunk(t, d) eljárás mintájára dolgozzuk ki a rightSubTreeShrunk(t, d) eljárást, ennek segédeljárásait, és az ehhez szükséges kiegyensúlyozási sémát (megoldás: 5. ábra)! Mutassuk be az AVLdel(t, k) eljárás működését néhány példán! Mutassunk olyan, legalább négy magasságú fát és kulcsot, amelyre az AVLdel(t, k) eljárás minden, a fizikailag törölt csúcs feletti szinten kiegyensúlyozást végez!*

3.4. Feladat. *Írjuk meg az AVLdel(t, k) eljárás egy olyan változatát, amely abban az esetben, ha a k kulcsot olyan belső csúcs tartalmazza, aminek két gyereke van, ezt a törlendő csúcsot a bal részfája legynagyobb kulcsú csúcsával helyettesíti!*

3.4. Az AVL fák magassága*

Tétel: Tetszőleges n csúcsú nemüres AVL fa h magasságára:

$$\lfloor \lg n \rfloor \leq h \leq 1,45 \lg n$$

Bizonyítás:

Az $\lfloor \lg n \rfloor \leq h$ egyenlőtlenség bizonyításához elég azt belátni, hogy ez tetszőleges nemüres bináris fára igaz. Egy tetszőleges bináris fa nulladik (gyökér) szintjén legfeljebb $2^0 = 1$ csúcs van, az első szintjén maximum $2^1 = 2$ csúcs, a második szintjén nem több, mint $2^2 = 4$ csúcs. Általában, ha az i-edik szinten 2^i csúcs van, akkor az $i+1$-edik szinten legfeljebb 2^{i+1} csúcs, hiszen minden csúcsnak maximum két gyereke van. Innét egy h mélységű bináris fa csúcsainak n számára $n \leq 2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1 < 2^{h+1}$. Innét
$\lfloor \lg n \rfloor \leq \lg n < \lg 2^{h+1} = h + 1$, amiből $\lfloor \lg n \rfloor \leq h$.

A $h \leq 1,45 \lg n$ egyenlőtlenség bizonyításához elég azt belátni, hogy ez tetszőleges nemüres, kiegyensúlyozott bináris fára (KBF-re) igaz. Ehhez először meghatározzuk egy $h \geq 0$ magasságú (azaz nemüres) KBF csúcsainak minimális f_h számát. Nyilván $f_0 = 1$ és $f_1 = 2$, hiszen egy nulla magasságú KBF csak a gyökércsúcsból áll, az egy magasságú KBF-ek pedig ((B)G(J)), ((B)G), vagy (G(J)) alakúak.

Az is világos, hogy az $< f_0, f_1, f_2, \cdots >$ sorozat szigorúan monoton növekvő. (Ennek igazolásához vegyünk egy $i+1$ magas, f_{i+1} csúcsú t KBF-et! Ekkor a t bal és jobb részfái közül az egyik magassága i. Legyen ez az f részfa! Jelölje most $s(b)$ egy tetszőleges b bináris fa csúcsainak számát! Akkor $f_{i+1} = s(t) > s(f) \geq f_i$.)

Ezután $h \geq 2$ esetén $f_h = 1 + f_{h-1} + f_{h-2}$. (Ha ugyanis t egy h magasságú, minimális, azaz f_h méretű KBF, akkor ennek bal és jobb részfáiban is, a részfák magasságához mérten a lehető legkevesebb csúcs van. Az egyik részfája kötelezően $h-1$ magas, ebben tehát f_{h-1} csúcs van. Mivel t KBF, a másik részfája $h-1$ vagy $h-2$ magas. A másik részfában tehát f_{h-1} vagy f_{h-2} csúcs van, és $f_{h-2} < f_{h-1}$, tehát a másik részfában f_{h-2} csúcs van, és így $f_h = 1 + f_{h-1} + f_{h-2}$.)

A képlet emlékeztet a Fibonacci sorozatra:
$F_0 = 0$, $F_1 = 1$, $F_h = F_{h-1} + F_{h-2}$, ha $h \geq 2$.

Megjegyzés: A h magasságú, és f_h méretű KBF-eket ezért *Fibonacci fáknak* hívjuk. Az előbbiek szerint egy $h \geq 2$ magasságú Fibonacci fa mindig $(\varphi_{h-1}$ G $\varphi_{h-2})$ vagy $(\varphi_{h-2}$ G $\varphi_{h-1})$ alakú, ahol φ_{h-1} és φ_{h-2} $h-1$, illetve $h-2$ magasságú Fibonacci fák.

3.5. Feladat. *Rajzoljunk $h \in 0..4$ magasságú Fibonacci fákat!*

Megvizsgáljuk, van-e valami összefüggés az
$< f_h : h \in \mathbb{N} >$ és az $< F_h : h \in \mathbb{N} >$ sorozatok között.

h	0	1	2	3	4	5	6	7	8	9
F_h	0	1	1	2	3	5	8	13	21	34
f_h	1	2	4	7	12	20	33			

A fenti táblázat alapján az első néhány értékre $f_h = F_{h+3} - 1$. Teljes indukcióval könnyen látható, hogy ez tetszőleges $h \geq 0$ egész számra igaz: Feltéve, hogy $0 \leq h \leq k \geq 1$ esetén igaz, $h = k + 1$-re:
$f_h = f_{k+1} = 1 + f_k + f_{k-1} = 1 + F_{k+3} - 1 + F_{k+2} - 1 = F_{k+4} - 1 = F_{h+3} - 1$.

Tudjuk, hogy

$$F_h = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^h - \left(\frac{1-\sqrt{5}}{2} \right)^h \right]$$

Az F_h-ra vonatkozó explicit képlet segítségével összefüggést adunk tetszőleges KBF n mérete és h magassága között.

$$n \geq f_h = F_{h+3} - 1 = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{h+3} - \left(\frac{1-\sqrt{5}}{2} \right)^{h+3} \right] - 1 \geq$$

$$\geq \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{h+3} - \left| \left(\frac{1-\sqrt{5}}{2} \right)^{h+3} \right| \right] - 1$$

Mivel $2 < \sqrt{5} < 3$, azért $|1 - \sqrt{5}|/2 < 1$ és $\left| \left(\frac{1-\sqrt{5}}{2} \right)^{h+3} \right| < 1$. Eszerint

$$n > \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{h+3} - 1 \right] - 1 = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^3 \left(\frac{1+\sqrt{5}}{2} \right)^h - \frac{1+\sqrt{5}}{\sqrt{5}}$$

Mivel

$$\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^3 = \frac{1 + 3\sqrt{5} + 3(\sqrt{5})^2 + (\sqrt{5})^3}{8\sqrt{5}} = \frac{16 + 8\sqrt{5}}{8\sqrt{5}} = \frac{2+\sqrt{5}}{\sqrt{5}}$$

Ezt behelyettesítve az előbbi, n -re vonatkozó egyenlőtlenségbe:

$$n > \frac{2+\sqrt{5}}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^h - \frac{1+\sqrt{5}}{\sqrt{5}}$$

Az első együtthatót tagokra bontva, a disztributív szabállyal:

$$n > \left(\frac{1+\sqrt{5}}{2}\right)^h + \frac{2}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^h - \frac{1+\sqrt{5}}{\sqrt{5}}$$

Most

$$\frac{2}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^h - \frac{1+\sqrt{5}}{\sqrt{5}} \geq 0 \iff \left(\frac{1+\sqrt{5}}{2}\right)^h \geq \frac{1+\sqrt{5}}{2} \iff h \geq 1$$

Eszerint $h \geq 1$ esetén

$$n > \left(\frac{1+\sqrt{5}}{2}\right)^h$$

$h = 0$ -ra pedig $n = 1$, és így $n = \left(\frac{1+\sqrt{5}}{2}\right)^h$

A fentiekből tetszőleges, nemüres KBF n méretére és h magasságára

$$n \geq \left(\frac{1+\sqrt{5}}{2}\right)^h$$

Innét, ha vesszük mindkét oldal kettes alapú logaritmusát, majd $\lg \frac{1+\sqrt{5}}{2}$ -tel osztunk:

$$h \leq \frac{1}{\lg \frac{1+\sqrt{5}}{2}} \lg n$$

Mivel $1,44 < 1,44042 < \frac{1}{\lg \frac{1+\sqrt{5}}{2}} < 1,4404201 < 1,45$, azért tetszőleges, nemüres KBF n méretére és h magasságára

$$h \leq 1,45 \lg n$$

4. Általános fák

Az általános fák esetében, a bináris fákkal összehasonlítva, egy csúcsnak tetszőlegesen sok gyereke lehet. Itt azonban, tetszőleges csúcshoz tartozó részfák száma pontosan egyenlő a gyerekek számával, azaz nem tartoznak hozzá üres részfák. Ha a gyerekek sorrendje lényeges, akkor *rendezett fákról* beszélünk.

Általános fákkal modellezhetjük például a számítógépünkben a mappák hierarchiáját, a programok blokkstruktúráját, a függvénykifejezéseket, a családfákat és bármelyik hierarchikus struktúrát.

Vegyük észre, hogy ugyan minden konkrét általános fában van az egy csúcshoz tartozó gyerekek számára valamilyen r felső korlát, de ettől a fa még nem tekinthető r-árisnak, mert ez a korlát nem abszolút: a fa tetszőleges csúcsa gyerekeinek száma tetszőlegesen növelhető. Másrészt, mivel itt nem értelmezzük az *üres részfa* fogalmát, ez mégsem általánosítása az r-áris fa fogalmának. Értelmezzük azonban a gyökércsúcs (nincs szülője) és a levélcsúcs (nincs gyereke) fogalmát, azaz továbbra is gyökeres fákról beszélünk.[2]

Az általános fák természetes ábrázolási módja a *bináris láncolt* reprezentáció. Itt egy csúcs szerkezete a következő (ahol most *child*1 az első gyerekre, *sibling* pedig a következő testvérre mutat). A $*p$ csúcs akkor levél, ha $p \to child1 = \oslash$; és a $*p$ csúcs akkor utolsó testvér, ha $p \to sibling = \oslash$.

Node
+ *child*1, *sibling* : Node* // *child*1: első gyerek; *sibling*: következő testvér + *key* : T // T ismert típus
+ Node() { *child*1 := *sibling* := $\oslash$ } // egycsúcsú fát képez belőle + Node(x:T) { *child*1 := *sibling* := $\oslash$; *key* := x }

Természetesen itt is lehet *szülő* pointer, ami a gyerekek közös szülőjére mutat vissza.

4.1. Feladat. *Próbáljunk megadni az általános fákra másfajta láncolt reprezentációkat is! Hasonlítsuk össze az általunk adott ábrázolásokat és a bináris láncolt reprezentációt, memóriaigény és rugalmasság szempontjából!*

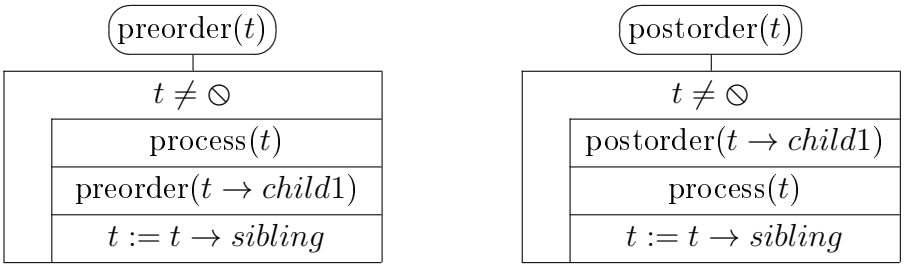
A szöveges (zárójeles) reprezentációban az általános fáknál a gyökeret előre szokás venni. Így egy nemüres fa általános alakja (G $t_1 \dots t_n$), ahol G a gyökércsúcs tartalma, $t_1 \dots t_n$ pedig a részfák.

Így pl. az { 1 [2 (5)] (3) [4 (6) (7)] } általános fában az 1 van a gyökérben, a gyerekei a 2, a 3 és a 4, a hozzájuk tartozó részfák pedig sorban a [2 (5)], a (3) és a [4 (6) (7)]. A fa levelei az 5, a 3, a 6 és a 7.

4.2. Feladat. *Rajzoljuk le a fenti 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.)*

[2]Ellentétben az ún. *szabad fákkal*, amik irányítatlan, körmentes gráfok.

Az általános fa preorder bejárása[3] a $child1 \sim left$ és $sibling \sim right$ megfeleltetéssel a bináris reprezentáció preorder bejárását igényli. Az általános fa postorder bejárásához[4] azonban (az előbbi megfeleltetéssel) a bináris reprezentáció inorder bejárása szükséges.[5]



4.3. Feladat. *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*[6] *nem szimulálható a bináris reprezentáció egyik nevezetes bejárásával*[7] *sem! Írjuk meg a bináris láncolt reprezentációra az általános fa inorder bejárását és szintfolytonos bejárását is!*

4.4. Feladat. *Írjuk meg a fenti bejárásokat az általunk adott aletrnatív láncolt reprezentációkra is! Írjunk olyan programokat, amelyek képesek tetszőleges általános fa egy adott reprezentációjából egy adott másik ábrázolású másolatot készíteni!*

5. B+ fák és műveleteik

http://aszt.inf.elte.hu/~asvanyi/ad/B+ fa.pdf

[3]először a gyökér, majd sorban a gyerekeihez tartozó részfák

[4]előbb sorban a gyökér gyerekeihez tartozó részfák, végül a gyökér

[5]A fájlrendszerekben általában preorder bejárás szerint keresünk, míg a függvénykifejezéseket (eltekintve most a lusta kiértékeléstől, aminek a tárgyalása túl messzire vezetne) postorder bejárással értékeljük ki.

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

[7]preorder, inorder, postorder, szintfolytonos

6. Egyszerű gráfok és ábrázolásaik ([3] 22)

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

6.1. Gráfelméleti alapfogalmak

6.1. 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.

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

6.3. 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.

6.4. 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.*

6.5. 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.

6.6. 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.

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

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

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

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

6.10. 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.*

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

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

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

6.14. 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 $G \neq G'$.*

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

6.16. 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.*

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

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

6.19. 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.

6.2. Bevezetés a gráf ábrázolásokhoz

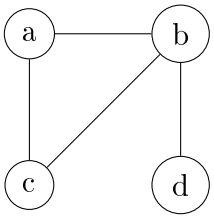
A gráf ábrázolásoknál a $G = (V, E)$ gráfról általában föltesszük, hogy $V = \{v_1, \cdots, v_n\}$, ahol $n \in \mathbb{N}$, azaz hogy a gráf csúcsait egyértelműen címkézik az $1..n$ sorszámok.

Grafikus és *szöveges* reprezentációban (ld. alább) a címkéket 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.)

6.3. Grafikus ábrázolás

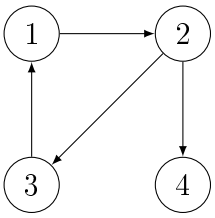
A gráfoknál megszokott módon a csúcsokat kis körök jelölik, az éleket irányított gráfoknál a körök közti nyilak, irányítatlan esetben a köröket összekötő vonalak reprezentálják. A csúcsok címkéit általában a körökbe írjuk.
A 7. ábrán egy egyszerű, irányítatlan gráfot láthatunk, grafikus és szöveges reprezentációban, a csúcsokat kisbetűkkel címkézve.

A 8. 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 címkéket is.)



a – b ; c.
b – c ; d.

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



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

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

6.4. 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. a 7. 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 8. ábrát!)

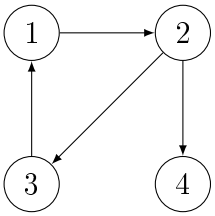
6.5. Szomszédossági mátrixos (adjacency matrix), más néven csúcsmátrixos reprezentáció

A szomszédossági mátrixos, vagy más néven csúcsmátrixos ábrázolásnál a $G = (V, E)$ gráfot ($V = \{v_1, \cdots, v_n\}$) egy $A : bit[n, n]$ mátrix reprezentálja, ahol $n = |V|$ a csúcsok száma, $1..n$ a csúcsok sorszámai, azaz azonosító címkéi,
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 9. ábrán látható irányított gráfot például a mellette lévő bitmátrix reprezentálja.



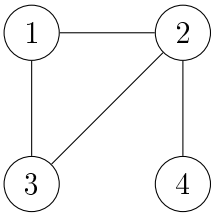
A	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	1	0	0	0
4	0	0	0	0

9. 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 $(v_i, v_j) \in E \iff (v_j, v_i) \in E$.

A 7. ábráról már ismerős irányítatlan gráf csúcsmátrixos ábrázolása a szokásos $a = 1, b = 2, c = 3, d = 4$ megfeleltetéssel a 10. 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

10. 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 $Z : bit[n * (n-1)/2]$ 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

$A[i,j] = Z[(i-1)*(i-2)/2+(j-1)]$ 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 Z tömbben, akkor azt kell megszámolnunk, hogy hány elem előzi meg sorfolytonosan az a_{ij} elemet a Z tömbben. Mivel a Z tömböt nullától indexeljük, a_{ij} indexe a Z-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.

6.6. Szomszédossági listás (adjacency list) reprezentáció

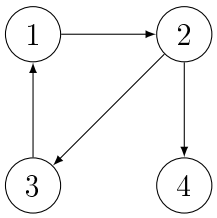
A szomszédossági listás ábrázolás hasonlít a szöveges reprezentációhoz. A $G = (V, E)$ gráfot ($V = \{v_1, \cdots, v_n\}$) az $A : Edge^*[n]$ pointertömb

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

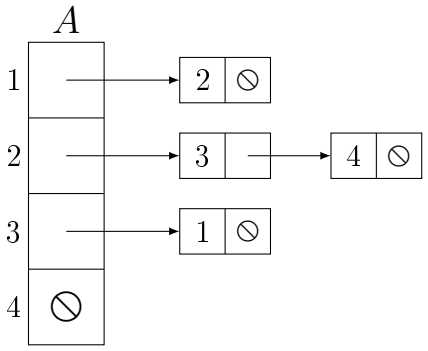
segítségével ábrázoljuk, ahol irányítatlan gráf esetében a v_i csúcs szomszédainak sorszámait az $A[i]$ S1L tartalmazza ($i \in 1..n$). A v_i csúcs szomszédainak indexeit tehát az $A[i]$ lista elemeinek v attritbutumai tartalmazzák. Így az

$A[i]$ 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) címkéit tartalmazza ($i \in 1..n$). Ilyen módon mindegyik élet csak egyszer kell ábrázolnunk. Egy példát láthatunk a 11. ábrán.



$A[1] \to v = 2$; $A[1] \to next = \oslash$
$A[2] \to v = 3$; $A[2] \to next \to v = 4$
; $A[2] \to next \to next = \oslash$
$A[3] \to v = 1$; $A[3] \to next = \oslash$
$A[4] = \oslash$



11. 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 gráf algoritmusnak ez a leggyakoribb művelete, ezt a reprezentációt gyakran előnyben részesítjük a csúcsmátrixos és az éllistás ábrázolással szemben.

6.7. Éllistás reprezentáció

Az éllistás ábrázolásnál egyszerűen listába fűzzük a gráf (v_i, v_j) éleinek megfelelő (i, j) indexpárokat. A 11. ábrán látható gráf absztrakt éllistája pl. a következő.

$$\langle (1,2), (2,3), (2,4), (3,1) \rangle$$

Számítógépen a fenti sorozatot a szokásos lineáris adatszerkezetek, pl. tömbök vagy láncolt listák segítségével reprezentálhatjuk.

Vegyük észre, hogy a gráf irányítatlan megfelelőjét éllistás esetben akár ugyanígy is ábrázolhatjuk! A kétféle gráf közötti különbség most a megfelelő adatszerkezetet kezelő programokban jelentkezhet.

Az éllistás ábrázolásnál pl. irányítatlan gráfok esetén a $(v_i, v_j) \in E$ kérdés eldöntéséhez kereséssel kell eldöntenünk, hogy az (i, j) vagy a (j, i) indexpár szerepel-e az éllistán, a v_i csúcs szomszédainak felsorolásához pedig az összes $(i, \cdot)$ vagy $(\cdot, i)$ alakú indexpárt meg kell találnunk. Így viszonylag kevés algoritmushoz célszerű ezt a reprezentációt választani.

6.8. 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.

6.8.1. 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 $\Theta(n^2)$.

6.8.2. 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.

6.8.3. Éllisták

Mivel az éllistának pontosan m eleme van, a tárigény $\Theta(m)$.

Ha a gráf nem nagyon ritka, azaz $m \in \Omega(n)$, akkor $\Theta(n+m) = \Theta(m)$, azaz az éllistás és a szomszédossági listás ábrázolás tárigénye aszimptotikusan ekvivalens.

Ha minden egyes csúccsal kapcsolatosan $\Theta(1)$ tárhely igényű információt kell az éllistán kívül tárolnunk, szintén $\Theta(n+m)$ memóriára lesz itt is szükségünk.

Ha csak az éllistát kellene tárolnunk, és $m \prec n$, akkor persze ez a reprezentáció lenne a leggazdaságosabb, $o(n)$ tárigénnyel. Ezek a feltételek viszont az ebben a jegyzetben következő algoritmusok egyikénél sem teljesülnek.

7. Elemi gráf algoritmusok ([3] 22)

A gráfok absztrakt algoritmusainak leírásához bevezetjük a $\mathcal{V}$ (vertex, azaz csúcs) absztrakt objektumtípust. Ez egy olyan nem-skalár (azaz strukturált) elemi típus, amelyben mindegyik objektumhoz tetszőlegesen sok, névvel jelölt attributum társítható, és mindegyik attributumhoz tartozik valamilyen érték: a $\mathcal{V}$ lesz a gráfok csúcsainak absztrakt típusa. Feltesszük továbbá, hogy a $\mathcal{V}$ típusú objektumok kizárólag absztrakt hivatkozásokon keresztül érhetők el.

Ha pl. $u : \mathcal{V}$, akkor u valójában egy absztrakt hivatkozás egy $\mathcal{V}$ típusú objektumra. Ha végrehajtjuk a $v := u$ utasítást, akkor v is ugyanarra a csúcsra hivatkozik, amelyikre u. Mivel a hivatkozás egyértelműen azonosítja a csúcsot, az u által hivatkozott csúcsot egyszerűen u csúcsnak fogjuk nevezni.

Az előbbi $v := u$ értékadás után tehát az u és a v csúcs azonos. Ez több, mintha csak egyenlők lennének, mert pl. az $u.name := x$ utasítás hatására az u csúcsnak lesz egy *name* nevű attributuma, amelynek értéke x, és ugyanez mondható el a v csúcsról is, hiszen azonosak. Ha ezután végrehajtódik az $u.name := y$ értékadás, akkor az u csúcs *name* nevű attributumának értéke y-ra változik (és persze $v.name = y$ is igaz lesz).

Vegyük észre, hogy az u csúcs tetszőleges *name* attributumát az „$u \rightarrow name$" kifejezés helyett az „$u.name$" kifejezéssel jelöltük. Ezt azért tehettük meg, mert a $\mathcal{V}$ típusú absztrakt objektumok csak hivatkozásokon keresztül érhetők el, ezért felesleges mind a konkrét pointer típusoknál szokásos „$*$" jelölés, mind pedig a hivatkozástalanításra utaló „$\rightarrow$" jelölés.

Bevezetünk még 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 tömbökhöz hasonlóan indexeljük (csak az indexet a matematikában szokásos módon alsó indexként jelöljük), továbbá,
- ha $u, v : \mathcal{T}\langle\rangle$, akkor $u + v$ jelöli a két sorozat konkatenáltját;
- $\langle\rangle$ önmagában az üres sorozat;
- ha $u : \mathcal{T}\langle\rangle$ és $u \neq \langle\rangle$, akkor $u.rem$ az első elemének eltávolítása után az u sorozat maradéka.

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 – vesszőkkel elválasztva – néhány $\mathcal{T}$ típusú elemet, akkor a halmaz illetve sorozat a deklaráció kiértékelődése után ezeket fogja tartalmazni.

Most már minden készen áll az élsúlyozatlan absztrakt gráfok leírásához. Gyakorlati megfontolásból elegendő az egyszerű gráfokra szorítkoznunk, amelyek nem tartalmaznak sem párhuzamos, sem hurokéleket.

$\mathcal{G}$
$+ V : \mathcal{V}\{\}$ $+ E \subseteq V \times V \setminus \{(u,u) : u \in V\}$ // edges

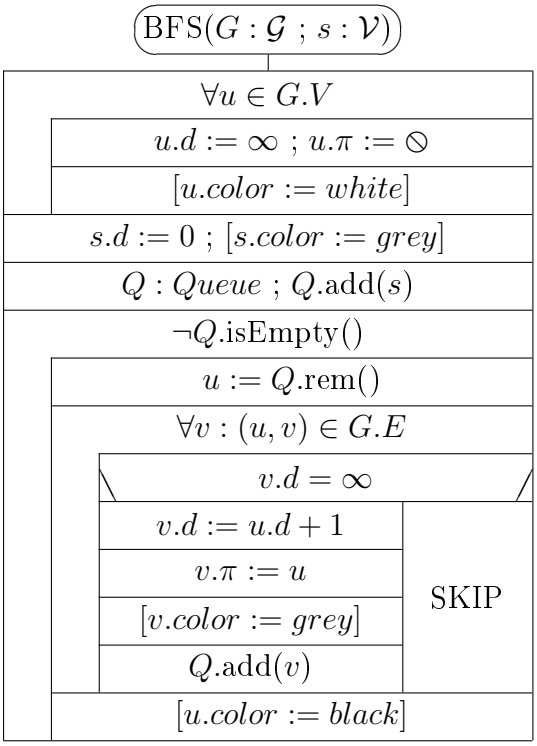
Elemi gráfalgoritmusok alatt élsúlyozatlan gráfokon értelmezett algoritmusokat értünk. Élsúlyozatlan gráfokban tetszőleges út hossza egyszerűen az út mentén érintett élek száma. Az *algoritmusok* témakörben szokásos fogalmak szerint az út tartalmazhat kört. Ha a gráfban tetszőleges u és v csúcsokra az (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.

7.1. A szélességi gráfkeresés (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 attributumai:
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 $u.d = \infty$ és $u.\pi = \oslash$ lesz. Ezenkívül $s.\pi = \oslash$ 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 attributumot 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 $u.d = \infty$ (ami azt jelenti, hogy még egyik csúcsot sem értük el), és ennek megfelelően $u.\pi = \oslash$ lesz[8].

Az s start (más néven source, azaz forrás) csúcsra azonban tudjuk, hogy $s.d = 0$ az optimális $s \rightsquigarrow s$ út hossza[9]. 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,[10] és bekerülnek a sorba.[11]

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 $v.d = 2$ és

[8] [valamint $u.color = white$]

[9] [és ezzel el is kezdtük az s feldolgozását, így $s.color = grey$ lett]

[10] [szürkére színezi őket]

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

$v.\pi = u$ értékeket a szülőjüknek megfelelően,[12] és bekerülnek a sor végére[13]. Ha valamelyik v gyerekcsúcsra az (u, v) él feldolgozásakor már $v.d \neq \infty$, akkor ez a csúcs már korábbról ismert[14], így $v.d \in \{0, 1, 2\}$, 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 az attributumaikat 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 attributumaikat is. A többi csúcs tehát s-ből nem érhető el. Ezekre $d = \infty$ és $\pi = \oslash$ marad.[15]

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

[12] [szürkére színezi őket]

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

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

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

7.1.1. A szélességi fa (Breadth-first Tree)

A BFS minden s-ből elérhető, de tőle különböző csúcsra, annak π attributumával hivatkozik annak szülőjére, az s-ből a csúcsba vezető (a BFS által meghatározott) 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 $s.\pi = \oslash$. 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.

7.2. 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* $d = v.d$.

7.3. 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?

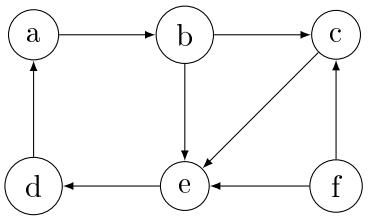
$MT(d) \in \Theta(d)$, *ahol* $d = v.d$.

7.4. 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* $d = v.d$, *ha s-ből v elérhető; különben* $d = 1$.

7.1.2. A szélességi gráfkeresés szemléltetése

A 12. 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.

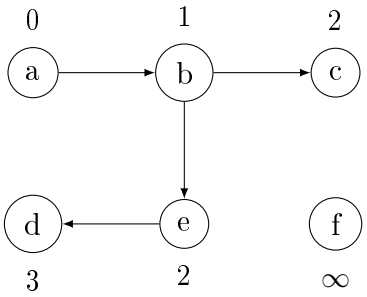


a → b.
b → c ; e.
c → e.
d → a.
e → d.
f → c ; e.

12. ábra. Ugyanaz az irányított gráf grafikus (balra) és szöveges (jobbra) ábrázolással (s = a).

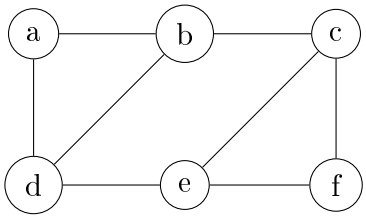
Most is és a későbbiekben is, *indeterminisztikus esetekben* a kisebb indexű csúcsot részesítjük előnyben. (Ez a leggyengébb szabály, de a zárthelyiken, vizsgákon is elvárjuk, hogy tartsák be.) Alább pl. ez akkor érvényesül, amikor a „b" csúcs gyerekeit „c,e" sorrendben tesszük be a sorba.

	d							π					
	a	b	c	d	e	f	Q	a	b	c	d	e	f
init	0	∞	∞	∞	∞	∞	⟨ a ⟩	⊘	⊘	⊘	⊘	⊘	⊘
a:0		1					⟨ b ⟩		a				
b:1			2		2		⟨ c, e ⟩			b		b	
c:2							⟨ e ⟩						
e:2				3			⟨ d ⟩				e		
d:3	0	1	2	3	2	∞	⟨ ⟩	⊘	a	b	e	b	⊘



13. ábra. A 12. ábráról ismerős gráf szélességi fája, ha s = a.

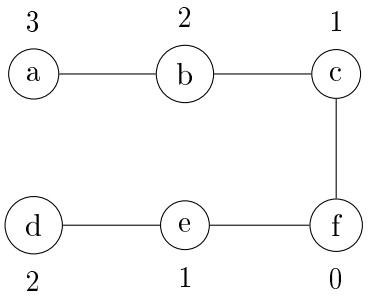
Most pedig a 14. ábrán látható gráfra szemléltetjük a BFS működését, az alábbi táblázat segítségével, ahol most „f" a start csúcs.



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

14. ábra. Ugyanaz az irányítatlan gráf grafikus (balra) és szöveges (jobbra) ábrázolással ($s = \mathrm{f}$).

	d							π					
	a	b	c	d	e	f	Q	a	b	c	d	e	f
init	∞	∞	∞	∞	∞	0	$\langle$ f $\rangle$	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$
f:0			1		1		$\langle$ c, e $\rangle$			f		f	
c:1		2					$\langle$ e, b $\rangle$		c				
e:1				2			$\langle$ b, d $\rangle$				e		
b:2	3						$\langle$ d, a $\rangle$	b					
d:2							$\langle$ a $\rangle$						
a:3	3	2	1	2	1	0	$\langle$ $\rangle$	b	c	f	e	f	$\oslash$



15. ábra. A 14. ábráról ismerős gráf szélességi fája, ha $s = \mathrm{f}$.

7.1.3. A szélességi gráfkeresés hatékonysága

A gráf algoritmusokban 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 a belső ciklus legfeljebb m-szer iterál összesen (ha s-ből mindegyik csúcs elérhető, akkor minden él sorra kerül), minimum pedig egyszer sem (ha s-ből nem megy ki egyetlen él sem).

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

7.1.4. A szélességi gráfkeresés implementációja szomszédossági listás és szomszédossági mátrixos gráfábrázolás esetén

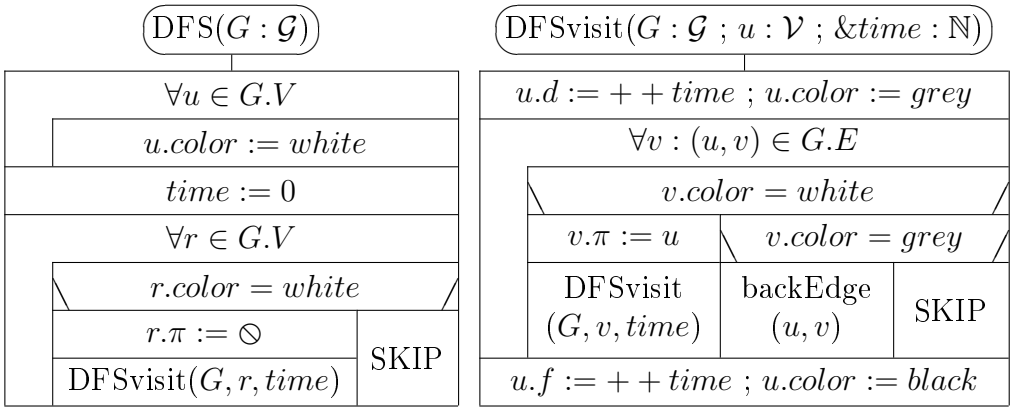
A $G = (V, E)$ gráfról föltesszük, hogy $V = \{v_1, \cdots, v_n\}$, ahol $n \in \mathbb{N}$, azaz a gráf csúcsait egyértelműen címkézik az $1..n$ sorszámok. A csúcsokat a sorszámaikkal azonosítjuk. Az absztrakt v_i csúcsokat úgy ábrázoljuk, hogy d és π attributumaiknak megfeleltetjük a $d, \pi : \mathbb{N}[n]$ tömböket, ahol $v_i.d$ reprezentációja $d[i]$ és $v_i.\pi$ reprezentációja $\pi[i]$. A *color* attributumok reprezentálása felesleges. A $\oslash$ reprezentációja lehet például a 0 számérték: pl. $\pi[s] = 0$ absztrakt jelentése $v_s.\pi = \oslash$.

7.5. Feladat. *Írja meg a BFS($A : Edge*[n]* ; $s : 1..n$; $d, \pi : \mathbb{N}[n]$) eljárást, ami a szélességi gráfkeresés implementációja szomszédossági listás (8.4) gráfábrázolás esetére. Ügyeljen arra, hogy az absztrakt algoritmus hatékonyságának aszimptotikus nagyságrendje megmaradjon!*

7.6. Feladat. *Írja meg a BFS($A : bit[n][n]$; $s : 1..n$; $d, \pi : \mathbb{N}[n]$) eljárást, ami a szélességi gráfkeresés implementációja szomszédossági mátrixos (8.3) gráfábrázolás esetére. Tartható-e az absztrakt algoritmus hatékonyságának aszimptotikus nagyságrendje? Ha nem, hogyan változik?*

7.2. A mélységi gráfkeresés (DFS: Depth-first Search)

Csak egyszerű irányított gráfokra értelmezzük. Fehér csúcs: érintetlen, szürke csúcs: belőle elérhető csúcsokat járunk be éppen, fekete csúcs: befejeztük.



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

7.2.1. Mélységi feszítő erdő (Depth-first forest)

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

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

7.2.2. Az élek osztályozása (Classification of edges)

7.7. Definíció. *A gráf éleinek osztályozása:*

(u, v) fa-él (tree edge) $\iff$ (u, v) *valamelyi 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 faél, 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.*

7.8. 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* $\wedge\ u.d < v.d$.

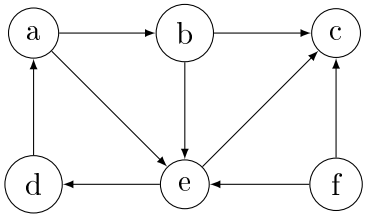
(u, v) kereszt-él (cross edge) $\iff$ *a* v *csúcs már fekete* $\wedge\ u.d > v.d$.

7.9. Feladat. *A mélységi bejárás során miért nem fordulhat elő, hogy az* (u, v) *él feldolgozásakor* $u.d = v.d$ *? Milyen feltételekkel fordulhatna ez elő? Hogyan kellene ekkor kiegészíteni a 7.7. definíciót úgy, hogy a 7.8. tétel megfogalmazásán ne kelljen módosítani?*

7.2.3. A mélységi bejárás szemléltetése

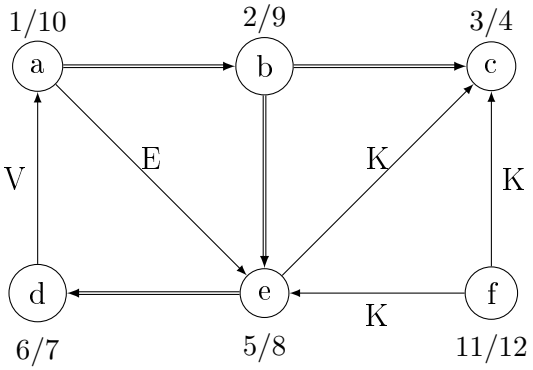
A 16. ábrán látható gráf mélységi bejárását szemléltetjük. A bejárás indeterminisztikus abban, hogy az egyes ciklusokban a csúcsokat milyen sorrendben veszi sorra. Ezt a szemléltetésben úgy fogjuk feloldani, hogy a csúcsokat ábécé rendben, illetve indexek szerint monoton növekvően dolgozzuk fel. (Ennek a konvenciónak a betartása az összes gráf algoritmus 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.

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



17. ábra. A 16. ábrán látható gráf mélységi bejárása.

A 17. ábrán követhető a gráf mélységi bejárása. A bejárás eredménye, a mélységi feszítő erdő két mélységi fából áll: az egyiknek az **a** csúcs a gyö-

kere 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 (DFSvisit) 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.]

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 7.8. 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 keresztélnek címkézzük. (Ld. a 7.8. tételt!) Ha nemfa-élt találunk, sosem lépünk tovább az általa hivatkozott csúcsba. [A keresztélnek és az előreélnek való 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 7.8. tételt!) [Vegyük észre, hogy ha vissza-élt találunk, egyúttal irányított

kört is találtunk a gráfban! (Ld. a 7.7. 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 7.8. 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 DFSvisit 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 keresztélnek 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 DFSvisit 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.

7.2.4. A mélységi gráfkeresés 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 DFSvisit rekurzív eljá-

rás. A DFSvisit 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 visszaélet, így $\Theta(1)$ műveletigényű, és műveletigénye hozzávehető az őt meghívó ciklusiterációéhoz.

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

7.2.5. A DAG tulajdonság eldöntése

7.10. 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 (ld. pl. a 7.2.6. és a 10.2. alfejezeteket), így az input ellenőrzése is szükséges lehet.

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 BFS fog visszaélet, és ezzel irányított kört találni [3]. [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.]

7.11. 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.

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

Ha a DFS talál egy (u, v) vissza-élet, akkor az $\langle u, u.\pi, u.\pi.\pi, \ldots, v, u\rangle$ 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.)

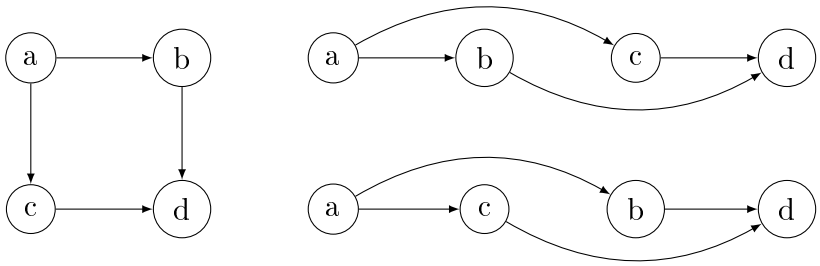
7.13. Feladat. *Írja meg a backEdge(u, v) eljárás struktogramját abban az esetben, ha ennek feladata a megtalált irányított kör explicit, jól olvasható megjelenítése is! Ügyeljen a $\Theta(n)$ maximális műveletigényre!*

7.14. 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 már a DFS minimális műveletigénye is $\Omega(n^2)$ lesz! Indokolja is az állítását!*

7.2.6. Topologikus rendezés

7.15. 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 18. ábra mutatja.



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

7.16. 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).

$\Leftarrow$ 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.

□

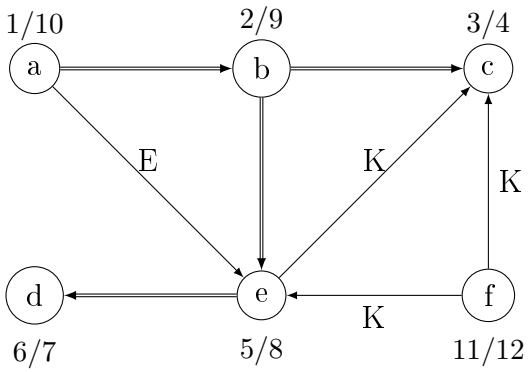
A topologikus rendezést végrehajthatjuk például az irányított gráf mélységi bejárása segítségével.

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 19. ábrán láthatunk egy példát.



19. ábra. A DAG topologikus rendezésében 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!

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.

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

7.18. Feladat. *Vegyük észre, hogy a 7.16. 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?*

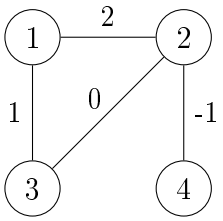
8. Élsúlyozott gráfok és ábrázolásaik ([3] 22)

8.1. Definíció. Élsúlyozott gráf *alatt egy* $G = (V, E, w)$ *rendezett hármast értünk, ahol* V *a csúcsok (vertices) tetszőleges, véges halmaza,* $E \subseteq V \times V \setminus \{(u, u) : u \in V\}$ *az élek (edges) halmaza,* $w : E \to \mathbb{R}$ *pedig a súlyfüggvény. Ez utóbbi minden egyes élhez hozzárendeli annak* súlyát*, más néven* hosszát*, illetve* költségét*. (Az* élsúly*,* élhossz *és* élköltség *elnevezések szinonímák.)*

8.2. Definíció. *Élsúlyozott gráfban tetszőleges út* hossza*, más néven* költsége*, illetve* súlya *az út mentén található élek összsúlya. Hasonlóképpen tetszőleges gráf/fa súlya az élei súlyainak összege.*

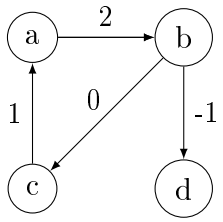
8.1. Grafikus ábrázolás

Az éleket súlyukkal címkézzük.



1 – 2, 2 ; 3, 1.
2 – 3, 0 ; 4, -1.

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



a → b, 2.
b → c, 0 ; d, -1.
c → a, 1.

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

8.2. Szöveges ábrázolás

Az irányítatalan gráfoknál „$u - v_{u_1}, w_{u_1}; \cdots ; v_{u_k}, w_{u_k}$." azt jelenti, hogy $(u, v_{u_1}), \cdots , (u, v_{u_k})$ élei a gráfnak, sorban $w(u, v_{u_1}) = w_{u_1}, \cdots , w(u, v_{u_k}) = w_{u_k}$ súlyokkal. (Ld. a 20. ábrát!)

Az irányított gráfoknál pedig „$u \to v_{u_1}, w_{u_1}; \cdots ; v_{u_k}, w_{u_k}$." azt jelenti, hogy a gráfban az u csúcsból az $(u, v_{u_1}), \cdots , (u, v_{u_k})$ irányított élek indulnak ki, most is sorban $w(u, v_{u_1}) = w_{u_1}, \cdots , w(u, v_{u_k}) = w_{u_k}$ súlyokkal. (Ld. a 21. ábrát!)

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

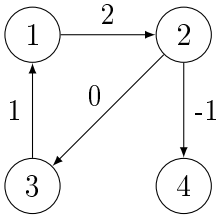
A szomszédossági mátrixos, vagy más néven csúcsmátrixos ábrázolásnál a $G = (V, E, w)$ gráfot ($V = \{v_1, \cdots , v_n\}$) egy $A : \mathbb{R}[n, n]$ mátrix reprezentálja, ahol $n = |V|$ a csúcsok száma, $1..n$ a csúcsok sorszámai, azaz azonosító címkéi, és tetszőleges $i, j \in 1..n$ csúcssorszámokra

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

$$A[i, i] = 0$$

$$A[i, j] = \infty \iff (v_i, v_j) \notin E \wedge i \neq j$$

A 22. ábrán látható irányított gráfot például a mellette lévő szomszédossági mátrix reprezentálja.



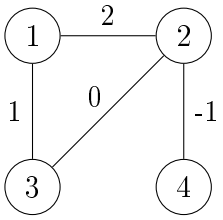
A	1	2	3	4
1	0	2	∞	∞
2	∞	0	0	-1
3	1	∞	0	∞
4	∞	∞	∞	0

22. ábra. Ugyanaz az élsúlyozott, irányított gráf grafikus (balra) és szomszédossági mátrixos (jobbra) ábrázolással.

A főátlóban mindig nullák vannak, mert csak egyszerű gráfokkal foglalkozunk (amelyekben nincsenek hurokélek), és tetszőleges csúcsból önmaga közvetlenül, nulla költségű úton érhető el.

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

A 20. ábráról már ismerős irányítatlan gráf csúcsmátrixos ábrázolása a 23. ábrán látható.



A	1	2	3	4
1	0	2	1	∞
2	2	0	0	-1
3	1	0	0	∞
4	∞	-1	∞	0

23. ábra. Ugyanaz az élsúlyozott, irányítatlan gráf grafikus (balra) és szomszédossági mátrixos (jobbra) ábrázolással.

8.4. Szomszédossági listás (adjacency list) reprezentáció

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

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

segítségével ábrázoljuk, ahol a v attributumok szerepe ugyanaz, mint az élsúlyozatlan gráfoknál, w pedig a megfelelő él súlya. Az élsúlyozatlan gráfokhoz hasonlóan az élsúlyozott gráfoknál is: irányítatlan gráfok esetén minden élet kétszer ábrázolunk, irányított gráfok esetén csak egyszer. Élsúlyozott, irányított gráfra láthatunk példát a 24. ábrán.

8.5. Éllistás reprezentáció

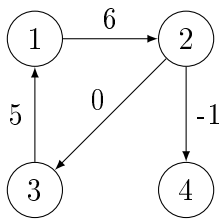
Hasonló, mint élsúlyozatlan esetben, csak az élekben most az élsúlyokat is tároljuk. A 24. ábrán látható gráf absztrakt éllistája pl. a következő.

$$\langle (1, 2, 6), (2, 3, 0), (2, 4, -1), (3, 1, 5) \rangle$$

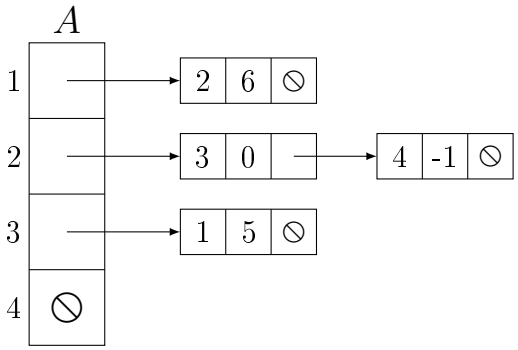
8.6. Élsúlyozott gráfábrázolások tárigénye

8.6.1. Szomszédossági mátrixok

Tárigényük hasonlóan számolható, mint élsúlyozatlan esetben. Feltéve, hogy egy valós számot egy gépi szóban tárolunk, a szomszédossági mátrixos (más néven csúcsmátrixos) ábrázolás tárigénye alapesetben n^2 szó. Irányítatlan gráfoknál, csak az alsóháromszög mátrixot tárolva, $n * (n-1)/2$ szó. Mivel $n * (n-1)/2 \in \Theta(n^2)$, az aszimptotikus tárigény mindkét esetben $\Theta(n^2)$.



$A[1] \to v = 2$; $A[1] \to w = 6$; $A[1] \to next = \oslash$
$A[2] \to v = 3$; $A[2] \to w = 0$; $A[2] \to next \to v = 4$
; $A[2] \to next \to w = -1$; $A[2] \to next \to next = \oslash$
$A[3] \to v = 1$; $A[3] \to w = 5$; $A[3] \to next = \oslash$
$A[4] = \oslash$



24. ábra. Ugyanaz az élsúlyozott, irányított gráf grafikus (balra) és szomszédossági listás (jobbra) ábrázolással.

8.6.2. Szomszédossági listák és éllisták

Mindegyik élben eggyel több mező van, mint az élsúlyozatlan esetben. Ez az aszimptotikus tárigényt nyilván nem befolyásolja.

8.7. Élsúlyozott gráfok absztrakt osztálya

$\mathcal{G}'$
$+ V : \mathcal{V}\{\}$
$+ E \subseteq V \times V \setminus \{(u,u) : u \in V\}$ // edges
$+ w : E \to \mathbb{R}$ // weights of edges

9. Minimális feszítőfák ([3] 23)

A minimális feszítőfa (MST) fogalma.

9.1. Egy általános algoritmus

Vágás, vágást keresztező él, élhalmazt elkerülő vágás, könnyű él. Egy tétel a biztonságos élekről és a minimális feszítőfákról.

9.2. Kruskal algoritmusa

Kruskal algoritmusa, mint az általános algoritmus megvalósítása. A futási idő elemzése.

9.3. Prim algoritmusa

Prim algoritmusa, mint az általános algoritmus megvalósítása. A futási idő elemzése.

HF: A Prim algoritmus implementációja a két fő gráfábrázolás és a szükséges prioritásos sor különböző megvalósításai esetén.

10. Legrövidebb utak egy forrásból([3] 24)

10.1. Dijkstra algoritmus

10.2. DAG legrövidebb utak egy forrásból

10.3. Sor alapú Bellman-Ford algoritmus

11. Legrövidebb utak minden csúcspárra ([3] 25)

11.1. Floyd-Warshall algoritmus

11.2. Gráf tranzitív lezártja

12. Mintaillesztés ([3] 32)

12.1. Egyszerű mintaillesztő (brute-force) algoritmus

12.2. Quicksearch

12.3. Mintaillesztés lineáris időben (KMP algoritmus)

13. Információtömörítés ([6] 5)

13.1. Naiv módszer

A tömörítendő szöveget karakterenként, fix hosszúságú bitsorozatokkal kódoljuk.
$\Sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_d \rangle$ az ábécé.
Egy-egy karakter $\lceil \lg d \rceil$ bittel kódolható.
$In : \Sigma\langle\rangle$ a tömörítendő szöveg. $n = |In|$ jelöléssel $n * \lceil \lg d \rceil$ bittel kódolható.

Pl. az ABRAKADABRA szövegre $d = 5$ és $n = 11$, ahonnét a tömörített kód hossza $11 * \lceil \lg 5 \rceil = 11 * 3 = 33$ bit. (A 3-bites kódok közül tetszőleges 5 kioszthatóaz 5 betűnek.) A tömörített fájl a kódtáblázatot is tartalmazza.

13.2. Huffman-kód

A tömörítendő szöveget karakterenként, változó hosszúságú bitsorozatokkal kódoljuk. A gyakrabban előforduló karakterek kódja rövidebb, a ritkábban előfordulóké hosszabb.

Prefix-mentes kód: Egyetlen karakter kódja sem prefixe semelyik másik karakter kódjának sem.

A karakterenként kódoló tömörítések között a Huffman-kód hossza minimális. Ugyanahhoz a szöveghez többféle kódfa és hozzátartozó kódtáblázat építhető, de mindegyik segítségével az input szövegnek ugyanolyan hosszú tömörített kódját kapjuk. Betömörítés a kódtáblával, kitömörítés a kódfával. Ezért a tömörített fájl a kódfát is tartalmazza.

A **tömörítendő fájlt, illetve szöveget kétszer olvassa végig.**
- Először meghatározza a szövegben előforduló karakterek halmazát és az egyes karakterek gyakoriságát, majd ennek alapján kódfát, abból pedig kódtáblázatot épít.
- Másodszorra a kódtábla alapján kiírja az output fájlba sorban a karakterek bináris kódját.

A **kódfa** szigorúan bináris fa. Mindegyik karakterhez tartozik egy-egy levele, amit a karakteren kívül annak gyakorisága, azaz előfordulásainak száma is címkéz. A belső csúcsokat a csúcshoz tartozó részfa leveleit címkéző karakterek gyakoriságainak összegével címkézzük. (Így a kódfa gyökerét a tömörítendő szöveg hossza címkézi.)

A **kódfát** úgy**építjük** fel, hogy először egycsúcsú fák egy minimum-prioritásos sorát határozzuk meg, amelyben mindegyik karakter pontosan egy csúcsot címkéz. A csúcsot a karakteren kívül annak gyakorisága is címkézi.

A minimum-prioritásos sort a benne tárolt fák gyökerét címkéző gyakoriságértékek szerint építjük fel. Ezután a következőt csináljuk ciklusban, amíg a kupac még legalább kettő fából áll.

Kiveszünk a kupacból egy olyan fát, amelyeknek gyökerét a legkisebb gyakoriság címkézi. Ezután a maradék kupacra ezt még egyszer megismételjük. Összeadjuk a két gyakoriságot. Az összeggel címkézünk egy új csúcsot, amelynek bal és jobb részfája az előbb kiválasztott két fa lesz. A bal ágat a 0, a jobb ágat az 1 címkézi. Az így képzett új fát visszatesszük a minimum-prioritásos sorba.

A fenti ciklus után a minimum-prioritásos sorban maradó egyetlen bináris fa a Huffman-féle kódfa.

A kódfából ezután kódtáblát készítünk. Mindegyik karakterekhez tartozó kódot úgy kapjuk meg, hogy a kódfa gyökerétől elindulva és a karakterhez tartozó levélig lefelé haladva a kódfa éleit címkéző biteket összeolvassuk.

Befejezésül újra végigolvassuk a tömörítendő szöveget, és a kódtábla segítségével sorban mindegyik karakter bináris kódját a (kezdetben üres) tömörített bitsorozat végéhez fűzzük.

A **kitömörítést** is karakterenként végezzük. Mindegyik karakter kinyeréséhez a kódfa gyökerétől indulunk, majd a tömörített kód sorban olvasott bitjeinek hatására 0 esetén balra, 1 esetén jobbra lépünk lefelé a fában, mígnem levélcsúcshoz érünk. Ekkor kiírjuk a levelet címkéző karaktert, majd a Huffman-kódban a következő bittől és újra a kódfa gyökerétől folytatjuk, amíg a tömörített kódon végig nem érünk.

13.2.1. Huffman-kódolás szemléltetése

Pl. az *ABRAKADABRA* szöveget egyszer végigolvasva meghatározhatjuk milyen karakterek fordulnak elő a szövegben, és milyen gyakorisággal.

$$\langle (D/\mathbf{1}), (K/\mathbf{1}), [B/\mathbf{2}], \{R/\mathbf{2}\}, (A/\mathbf{5}) \rangle$$

A fenti öt kifejezést öt egycsúcsú bináris fának tekinthetjük. (A jobb olvashatóság kedvéért többféle zárójelpárt alkalmaztunk.) Mindegyik csúcs egyben levél és gyökér. A levelekhez tartozó két címkét *karakter/gyakoriság* alakban írtuk le. A tömörítés algoritmusa szerint ezeket egy minimum-prioritásos sorba tesszük. A könnyebb érthetőség kedvéért ezt a minimum-prioritásos sort a szokásos minimum-kupacos reprezentáció helyett most a fák gyökerében lévő gyakoriság-értékek (röviden *fa-gyakoriság-értékek*) szerint rendezett fa-sorozattal szemléltetjük. (Azonos gyakoriságok esetén a betűk alfabetikus sorrendje szerint rendezünk. Ez ugyan önkényes, de az algoritmus bemutatása szempontjából hasznos egyértelműsítés. A fák ágait is hasonlóképpen rendezzük sorba.)

Ezután kivesszük a két legkisebb gyakoriság-értékű fát, egy új gyökércsúcs alá tesszük őket bal- és jobboldali részfának, a új gyökércsúcsot pedig a két fa-gyakoriság-érték összegével címkézzük. Végül visszatesszük az új fát a minimum-prioritásos sorba.

$$\langle [B/\mathbf{2}], [(D/1)\mathbf{2}(K/1)], \{R/\mathbf{2}\}, (A/\mathbf{5})\rangle$$

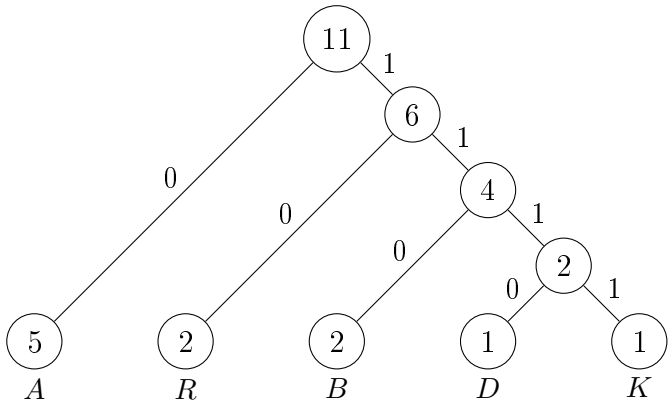
A fenti eljárást addig ismételjük, amíg már csak egy fánk marad. Ezt végül kivesszük a minimum-prioritásos sorból: ez a Huffman-féle kódfa.

$$\langle \{R/\mathbf{2}\}, \{[B/2]\mathbf{4}[(D/1)2(K/1)]\}, (A/\mathbf{5})\rangle$$

$$\langle (A/\mathbf{5}), (\{R/2\}\mathbf{6}\{[B/2]4[(D/1)2(K/1)]\})\rangle$$

$$[(A/5)\mathbf{11}(\{R/2\}6\{[B/2]4[(D/1)2(K/1)]\})]$$

A fent kapott kódfát a 25. ábrán is láthatjuk.



25. ábra. Az *ABRAKADABRA* szövegnek az alfabetikus konvencióval adódó Huffman-féle kódfája

Tekintsünk a 25. ábrán látható kódfában egy tetszőleges egyszerű, azaz körmentes utat, amely a fa gyökerétől lefelé valamelyik leveléig halad! Az út éleit címkéző biteket összeolvasva adódik a levelet címkéző karakter Huffman-kódja. Így a karaktarekre a következő kódtáblázatot kapjuk.

karakter	kód
A	0
B	110
D	1110
K	1111
R	10

A fentiek alapján az ABRAKADABRA szöveg Huffman kódja 23 bit, ami lényegesen rövidebb, mint a fenti naiv tömörítés esetén. A kódtáblázat bináris kódjait az ABRAKADABRA szöveg karakterei szerint sorban egymás után fűzve kapjuk a szöveg Huffman-kódját.

01101001111011100110100

A **kitömörítéshez** az előbbi Huffman-kód és a kódfa alapján a kezdő nulla rögtön az „A" címkéjű levélhez visz. Ezután sorban olvasva a maradékból a biteket, a 110 a B-hez visz, majd a 10 az R-hez, a 0 az A-hoz, a 1111 a K-hoz, a 0 az A-hoz, az 1110 a D-hez, a 0 az A-hoz, a 110 a B-hez, a 10 az R-hez, és végül a 0 az A-hoz. Így visszakaptuk az eredeti, tömörítetelen szöveget.

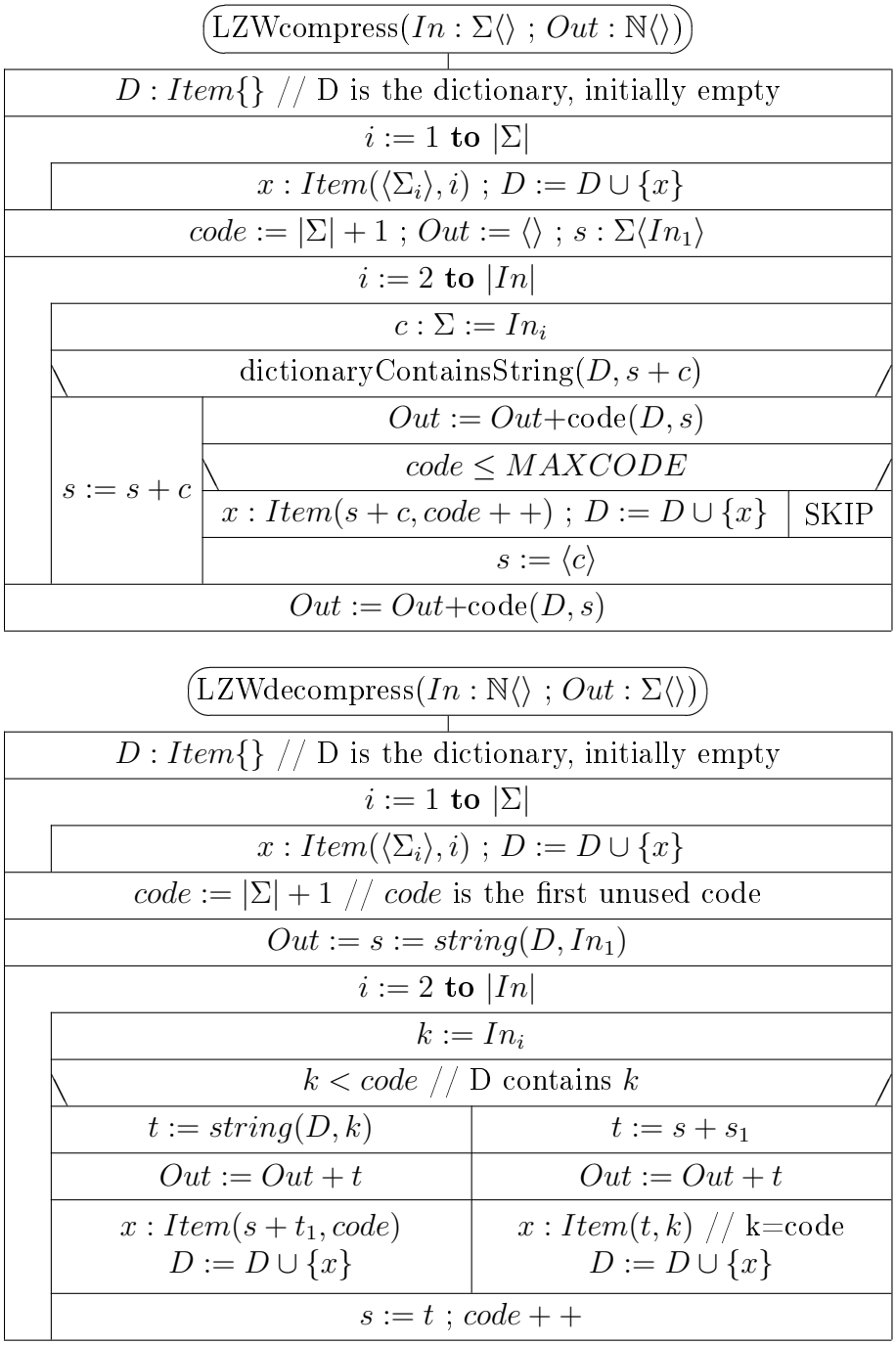
13.1. Feladat. *Próbáljuk ki, hogy ha a Huffman-kódolásban lévő indeterminizmusokat a fenti alfabetikus sorrendtől eltérően oldjuk fel, ugyanarra a tömörítendő szövegre mégis mindig ugyanolyan hosszú Huffman-kódot kapunk! (Ha például a minimum-prioritásos sorból azonos fa-gyakoriság-értekek esetén az alacsonyabb fát vesszük ki előbb – ezt az ad-hoc szabályt az alfabetikus konvenciónál erősebbnek véve –, akkor a fenti példában a kódfát felépítő ciklus második iterációjában a* $[B/\mathbf{2}]$ *és az* $\{R/\mathbf{2}\}$ *fát fogjuk összevonni.)*

13.3. Lempel–Ziv–Welch (LZW) módszer

Az input szöveget ismétlődő mintákra (sztringekre) bontja. Mindegyik mintát ugyanolyan hosszú bináris kóddal helyettesíti. Ezek a minták kódjai. A tömörített fájl a kódtáblázatot nem tartalmazza. Részletes magyarázat olvasható Ivanyos Gábor, Rónyai Lajos és Szabó Réka: *Algoritmusok* c. könyvében [6]. (Online elérhetősége az irodalomjegyzékünkben.)

Jelölések az absztrakt struktogramokhoz:
- Ha a kódok b bitesek, akkor $MAXCODE = 2^b - 1$ globális konstans a kódként használható legnagyobb számérték. Ha pl. $b = 12$, akkor $MAXCODE = 2^{12} - 1 = 4095$.
- A $\Sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_d \rangle$ sorozat tartalmazza az ábécé karaktereit.
- „*In*" a tömörítendő szöveg. „*Out*" a tömörítés eredménye: kódok sorozata.
- D a szótár, ami $(string, code)$ rendezett párok, azaz *Item*-ek halmaza.

Item
$+string : \Sigma\langle\rangle$ $+code : \mathbb{N}$
$+Item(s : \Sigma\langle\rangle \ ; \ k : \mathbb{N})\{string := s \ ; \ code := k\}$


LZWcompress($In : \Sigma\langle\rangle$; $Out : \mathbb{N}\langle\rangle$)
$D : Item\{\}$ // D is the dictionary, initially empty
$i := 1$ **to** $|\Sigma|$
$x : Item(\langle\Sigma_i\rangle, i)$; $D := D \cup \{x\}$
$code := |\Sigma| + 1$; $Out := \langle\rangle$; $s : \Sigma\langle In_1\rangle$
$i := 2$ **to** $|In|$
$c : \Sigma := In_i$
dictionaryContainsString($D, s + c$)
$s := s + c$
$Out := Out$+code(D, s)
$code \leq MAXCODE$
$x : Item(s + c, code + +)$; $D := D \cup \{x\}$
SKIP
$s := \langle c\rangle$
$Out := Out$+code(D, s)
LZWdecompress($In : \mathbb{N}\langle\rangle$; $Out : \Sigma\langle\rangle$)
$D : Item\{\}$ // D is the dictionary, initially empty
$i := 1$ **to** $|\Sigma|$
$x : Item(\langle\Sigma_i\rangle, i)$; $D := D \cup \{x\}$
$code := |\Sigma| + 1$ // $code$ is the first unused code
$Out := s := string(D, In_1)$
$i := 2$ **to** $|In|$
$k := In_i$
$k < code$ // D contains k
$t := string(D, k)$
$t := s + s_1$
$Out := Out + t$
$Out := Out + t$
$x : Item(s + t_1, code)$
$D := D \cup \{x\}$
$x : Item(t, k)$ // k=code
$D := D \cup \{x\}$
$s := t$; $code + +$