Algoritmusok I. gyakorló feladatok (2016)

A listaelemek **class** Elem{ $+kulcs$: int; $+mut$: Elem* }, illetve az előadásról ismert Elem2 típusúak. A láncoltan ábrázolt bináris fák csúcsai a jegyzetből (ad1jegyzet.pdf) ismert Csúcs, illetve Csúcs3 típusúak.

Az eljárásokat és függvényeket megfelelően elnevezett és paraméterezett struktogramok segítségével adjuk meg! A változókat alapértelmezésben a struktogramra vonatkozóan lokálisnak tekintjük.

Függvények aszimptotikus növekedése

F.1. Az A programra $T_A(n) = 108n^2$, míg a B programra $T_B(n) = 4n^3$, ugyanazon a gépen. Mikor lesz gyorsabb az A, és mikor a B program? Mikor lesznek egyforma gyorsak?

F.2. Tegyük fel, hogy összehasonlítjuk a beszúró rendezés (insertion sort) és az összefésülő rendezés (merge sort) implementációját ugyanazon a gépen: n méretű inputokra a beszúró rendezés $8n^2$ lépést igényel, míg az összefésülő rendezés $64n \lg n$ lépést. Milyen n értékekre gyorsabb az összefésülő rendezés, mint a beszúró rendezés?

F.3. Mi az a legkisebb n érték, amire az az algoritmus, amelyre $T(n) = 100n^2$, gyorsabb, mint az, amelyre $T(n) = 2^n$, ugyanazon a gépen?

F.4. Fejezze ki az alábbi függvényeket az előadásról ismert Θ-jelöléssel! (Pl. $2n - 3 \in \Theta(n)$.)
$9n^{9/2} + n^5 - 1234n^4$
$n \ln n + n^{1,1}$
$n^3/1000 - 100n^2 - 100n + 3$
$2n^4 + 678n^3 \sin n - 8n^3 \lg n + 999n^2 \cos n - 3256n \ln n - 985n^{3/2}$

F.5. Igaz-e,hogy $2^{n+1} \in \Theta(2^n)$. Miért?

F.6. Igaz-e,hogy $2^{2n} \in O(2^n)$. Miért?

F.7. f és g a nemnegatív egészekről a valós számok halmazára leképező függvények, elég nagy helyettesítési értékekre pozitívak. Bizonyítsuk be a következő állítást! $max(f,g) \in \Theta(f+g)$, ahol tetszőleges n nemnegatív egészre $max(f,g)(n) = max(f(n), g(n))$ és $(f+g)(n) = f(n) + g(n)$.

F.8. f és g a nemnegatív egészekről a valós számok halmazára leképező függvények, elég nagy helyettesítési értékekre pozitívak. Bizonyítsuk be a következő állítást! Ha az $f(n)/g(n)$ végtelenben vett határértéke egy pozitív szám, akkor $f \in \Theta(g)$.

F.9. Bizonyítsuk be a következő állítást! $0,5 * n^2 - 9 * n + 4 \in \Theta(n^2)$. A bizonyításban a pozitív együtthatós polinomok nagyságrendjére vonatkozó általános tételt *ne* használjuk fel!

F.10. f, g és h a nemnegatív egészekről a valós számok halmazára leképező függvények, elég nagy helyettesítési értékekre pozitívak. Bizonyítsuk be a következő állítást! Ha $f \in \Theta(g)$ és $g \in \Theta(h)$ akkor $f \in \Theta(h)$.

Rendezések

R.1. Tekintsük az $A[1..n]$ tömb rendezését a következő algoritmussal! Először megkeressük a tömb minimális elemét, majd megcseréljük $A[1]$-gyel. Ezután megkeressük a második legkisebb elemét és megcseréljük $A[2]$-vel. Folytassuk ezen a módon az $A[1..n]$ első $(n-1)$ elemére! (Itt tehát a vektort egy rendezett és egy rendezetlen szakaszra bontjuk. A rendezett szakasz a tömb elején kezdetben üres. A minimumkeresés mindig a rendezetlen részen történik, és a csere után a rendezett szakasz mindig eggyel hosszabb lesz.) Pl.:

$\langle 3;9;7;1;6;2\rangle \rightarrow \langle 1 \mid 9;7;3;6;2\rangle$
$\rightarrow \langle 1;2 \mid 7;3;6;9\rangle \rightarrow \langle 1;2;3 \mid 7;6;9\rangle$

$\rightarrow \langle 1; 2; 3; 6 \mid 7; 9\rangle \rightarrow \langle 1; 2; 3; 6; 7 \mid 9\rangle$
$\rightarrow \langle 1; 2; 3; 6; 7; 9\rangle$

Írjunk struktogramot erre a – minimumkiválasztásos rendezés néven közismert – algoritmusra MinKivRend($A[1..n]$) néven! Mi lesz a fő ciklus invariánsa? Miért elég csak az első $(n-1)$ elemre lefuttatni? Adjuk meg az $MT(n)$ és $mT(n)$ függvényeket a minimumkiválasztásos rendezésre a gyakorlatról ismert Θ-jelöléssel!

R.2. Tekintsük az $A[1..n]$ tömb rendezését a következő algoritmussal! Először megkeressük a tömb maximális elemét, majd megcseréljük $A[n]$-nel. Ezután megkeressük a második legnagyobb elemét és megcseréljük $A[n-1]$-gyel. Folytassuk ezen a módon az $A[1..n]$ utolsó $(n-1)$ elemére! Pl.:

$\langle 3; 1; 9; 2; 7; 6\rangle \rightarrow \langle 3; 1; 6; 2; 7 \mid 9\rangle$
$\rightarrow \langle 3; 1; 6; 2 \mid 7; 9\rangle \rightarrow \langle 3; 1; 2 \mid 6; 7; 9\rangle$
$\rightarrow \langle 2; 1 \mid 3; 6; 7; 9\rangle \rightarrow \langle 1 \mid 2; 3; 6; 7; 9\rangle$
$\rightarrow \langle 1; 2; 3; 6; 7; 9\rangle$

Írjunk struktogramot erre a – maximumkiválasztásos rendezés néven közismert – algoritmusra MaxKivRend($A[1..n]$) néven! Mi lesz a fő ciklus invariánsa? Miért elég csak az utolsó $(n-1)$ elemre lefuttatni? Adjuk meg az $MT(n)$ és $mT(n)$ függvényeket a maximumkiválasztásos rendezésre a gyakorlatról ismert Θ-jelöléssel!

R.3. Az előadásról ismert módon illusztrálja a beszúró rendezés (insertion sort) működését az alábbi tömbre!
$A = \langle 31; 41; 59; 26; 58; 7; 22; 91; 41\rangle$.

R.4. Az előadásról ismert módon illusztrálja a gyorsrendezés (quick sort) működését az alábbi tömbre!
$A = \langle 31; 15; 52; 23; 68; 83; 71; 24; 92; 47\rangle$.

R.5. Az előadásról ismert módon illusztrálja az összefuttató (összefésülő) rendezés (merge sort) működését az alábbi tömbre!
$A = \langle 33; 11; 25; 27; 6; 84; 73; 2; 91; 24\rangle$.

R.6. Tekintsük az F fejpointerű FKCL rendezését a következő algoritmussal! Először megkeressük a lista maximális elemét, majd átfűzzük a fejelem elé. Ezután megkeressük a második legnagyobb elemét és átfűzzük az első maximum elé. Folytassuk ezen a módon,összesen $(n-1)$ maximumkeresést végezve! Itt tehát a listát egy rendezetlen és egy rendezett szakaszra bontjuk, ahol a rendezett szakasz a lista végén kezdetben üres. Mindig a rendezetlen szakaszon keresünk maximumot, és a maximális kulcsú elemet átfűzzük a rendezett szakasz elejére. Pl.:

$\langle 3;1;9;2;7;6\rangle \to \langle 3;1;6;2;7 \mid 9\rangle$
$\to \langle 3;1;6;2 \mid 7;9\rangle \to \langle 3;1;2 \mid 6;7;9\rangle$
$\to \langle 2;1 \mid 3;6;7;9\rangle \to \langle 1 \mid 2;3;6;7;9\rangle$
$\to \langle 1;2;3;6;7;9\rangle$

Írjunk struktogramot erre a – maximumkiválasztásos rendezés néven közismert – algoritmusra MaxKivRend(F) néven! Mi lesz a fő ciklus invariánsa? Miért elég csak az első $(n-1)$ elemre lefuttatni? Adjuk meg az $MT(n)$ és $mT(n)$ függvényeket a minimumkiválasztásos rendezésre az előadásról ismert Θ-jelöléssel!

R.7. Tekintsük az F fejpointerű fejelemes lista rendezését a következő algoritmussal! Először megkeressük a lista minimális elemét, majd átfűzzük a fejelem után. Ezután megkeressük a második legkisebb elemét és átfűzzük az első minimum után. Folytassuk ezen a módon a lista első $(n-1)$ elemére! Pl.:

$\langle 3;9;7;1;6;2\rangle \to \langle 1 \mid 9;7;3;6;2\rangle$
$\to \langle 1;2 \mid 7;3;6;9\rangle \to \langle 1;2;3 \mid 7;6;9\rangle$
$\to \langle 1;2;3;6 \mid 7;9\rangle \to \langle 1;2;3;6;7 \mid 9\rangle$
$\to \langle 1;2;3;6;7;9\rangle$

Írjunk struktogramot erre a – minimumkiválasztásos rendezés néven közismert – algoritmusra MinKivRend(F) néven! Mi lesz a fő ciklus invariánsa? Miért elég csak az első $(n-1)$ elemre lefuttatni? Adjuk meg az $MT(n)$ és $mT(n)$ függvényeket a minimumkiválasztásos rendezésre az előadásról ismert Θ-jelöléssel!

R.8. Az L pointer egy egyszerű láncolt lista elejére mutat. (A lista üres is lehet). Írjuk meg a quicksort(L) eljárást, ami a listát monoton növekvően rendezi $O(|L| * log(|L|))$ minimális és $O(|L|^2)$ maximális műveletigénnyel. **Algoritmus**: Ha a lista nem üres, akkor az első elemnél kisebb-vagy-egyenlő és a nála nagyobb elemekből külön-külön, egy-egy segédlistát képezünk. Az így kapott két listát külön-külön, rekurzívan, quicksort-tal rendezzük és az eredményeket, középen az eredetileg első listaelemmel, rendezetten egymás után fűzzük.

R.9. Az L pointer egy egyszerű láncolt lista elejére mutat. (A lista üres is lehet). Írjuk meg a unionSort(L) eljárást, ami a listát szigorúan monoton növekvően rendezi $O(|L| * log(|L|))$ maximális műveletigénnyel úgy, hogy közben az elemduplikációkat törli (`delete`). **Algoritmus**: Ha a listának legalább két eleme van, akkor elfelezzük, az így kapott két listát külön-külön, unionSort-tal rendezzük és az eredményeket rendezetten uniózzuk. A lista felezést most úgy kell végrehajtani, hogy páros sorszámú (a második, negyedik stb.) listaelemeket egy másik listába tesszük, míg a páratlan sorszámúak (az első, harmadik stb.) az első listában maradnak. (A lista hosszát ne számoljuk ki!)

R.10. Lássuk be, hogy a kupacrendezés minimális futási ideje $\Theta(n)$, ami például akkor áll elő, amikor az input tömb minden eleme egyenlő!

R.11. A bináris keresőfák mintájára valósítsuk meg a Halmaz osztályt egy üresre inicializáló konstruktorral, továbbá a fv keres(k), beszúr(k), töröl(k), fv min(), minKivesz(min), fv max(), maxKivesz(max), műveletekkel, de a reprezentációhoz (1) rendezetlen tömböket, (2) növekvően rendezett tömböket, (3) csökkenően rendezett tömböket, (4) rendezetlen listákat, (5) növekvően rendezett listákat, (6) csökkenően rendezett listákat használva úgy, hogy minden kulcs csak egyszer szerepeljen! A hat rep-

rezentáció mindegyikére gondoljuk meg, hogy mely műveletekre tudnánk az $MT(n) \in \Theta(n)$ hatékonyságot javítani? Mennyire?

R.12. Az előadásról ismert módon illusztrálja a kupacrendezés (heapsort) működését az alábbi tömbre! A tömbben reprezentált fastruktúrát minden lesüllyesztés után rajzoljuk újra!
$A = \langle 31; 15; 52; 23; 68; 83; 71; 24; 92; 47 \rangle$.

Vermek

V.1. A gyakorlatról ismert módon illusztrálja a posztfix (azaz „lengyel") formára hozó algoritmus működését az alábbi aritmetikai kifejezésre! (Akkor rajzoljuk újra a vermet, amikor kiveszünk belőle egy vagy több elemet!)
`x*y+z*(x+1)/(y-5*(x+z))`

V.2. A gyakorlatról ismert módon illusztrálja a posztfix (azaz „lengyel") forma kiértékelő algoritmus működését az alábbi aritmetikai kifejezésre az `x=6, y=7, z=9` helyettesítési értékekkel! (Akkor rajzoljuk újra a vermet, amikor kiveszünk belőle egy vagy több elemet!)
`xyzx3-/-*z1-/3y*+z-`

V.3. Hozzuk postfix formára a 2+3*((4+8)/2)-6*3/2 kifejezést, majd értékeljük ki az így kapott lengyel formát, a gyakorlatról ismert algoritmusok segítségével! A vermet minden kipakolás után kell újra lerajzolni.

V.4. Az F szövegfájlban kerek zárójelekkel helyesen zárójelezett szöveg van. A zárójelpárok egymásba ágyazottan is előfordulhatnak. Írjunk programot, amely kiírja a zárójelpárok pozícióját! Pl.
`F = "Maga(san) (maradt (a(z)) ((al)(ma)fa) alatt)."`
esetén az alábbi eredmény adódik.
`(5,9) (21,23) (19,24) (27,30) (31,34) (26,37) (11,44)`

V.5. Tegyük fel, hogy a „verem” adattípusban az adatszerkezetet egyszerű láncolt listával ábrázoltuk! (Ld. gyakorlat!) Írjuk meg a verem destruktorát, azaz a listát törlő eljárást! A felszabaduló listaelemeket a szokásos `delete` művelettel töröljük. $T(n) \in \Theta(n)$ legyen! (Feltesszük, hogy $T_{\text{delete}} \in \Theta(1)$.)

V.6. Írjuk meg a Verem osztályt dinamikusan allokált tömbbel! Ha a verembe művelet úgy találja, hogy már tele van a tömb, cserélje le kétszer akkorára! Ügyeljünk a nagyságrendileg optimális várható futási időkre

Sorok

S.1. Tegyük fel, hogy a „Sor” adattípusban az adatszerkezetet olyan egyirányú, nemciklikus láncolt listával ábrázoltuk, amelynek a végén őrszem (végelem) van! (Ld. gyakorlat!) Írjuk meg a sor destruktorát, azaz a listát törlő eljárást! A felszabaduló listaelemeket a szokásos `delete` művelettel töröljük. $T(n) \in \Theta(n)$ legyen! (Feltesszük, hogy $T_{\text{delete}} \in \Theta(1)$.)

S.2. Tegyük fel, hogy a „Sor” adattípusban az adatszerkezetet egyirányú, őrszem elemmel is ellátott ciklikus láncolt listával ábrázoltuk! A sort azonosító pointer az őrszemre mutat. Adjuk meg a következő öt műveletet: üres sor konstruktora, a sor ürességének ellenőrzése, a sorba(x) és a sorból() metódusok, a sor destruktora – a gyakorlatról ismert módon! A sor destruktorára $T(n) \in \Theta(n)$ a többi műveletre $T(n) \in \Theta(1)$ legyen, ahol n a sor hossza. (Feltesszük, hogy $T_{\text{new}} \in \Theta(1)$ és $T_{\text{delete}} \in \Theta(1)$.)

Elsőbbségi (prioritásos) sorok

E.1. Adott az $A = \langle 8; 6; 8; 5; 4; 5; 2; 3; 4; 2 \rangle$ tömb, ami egy bináris maximum-kupacot reprezentál. (Feltehető, hogy ez egy nagyobb tömb első néhány eleme.) Szemlétessük az előadásról ismert módon a 9, a 8, a 7 és a 3 beszúrását a fenti kupacba! Mindegyik esetben a fent adott kupacra alkalmazzuk a műveletet!

E.2. Adott az $A = \langle 8; 6; 8; 5; 4; 5; 2; 3; 4; 2\rangle$ tömb, ami egy bináris maximum-kupacot reprezentál. Szemlétessünk az előadásról ismert módon sorban három maxKivesz műveletet erre a kupacra!

E.3. Adott az $A = \langle 8; 8; 6; 7; 6; 5; 2; 3; 4; 5\rangle$ tömb, ami egy bináris maximum-kupacot reprezentál. Szemlétessünk az előadásról ismert módon sorban három maxKivesz műveletet erre a kupacra!

E.4. Adott az $A[k..m]$ tömb, amiben szintfolytonosan ábrázoltunk egy kváziteljes (majdnem teljes), balra tömörített bináris fát. Tegyük fel, hogy k és m egész számok, továbbá $i \in k..m$, azaz $A[i]$ a fa egy tetszőleges csúcsa. Adjuk meg az alábbi függvényeket:
- $bal(i : k..m) : \mathbb{Z}$, ahol $A[bal(i)]$ az $A[i]$ bal gyereke (feltesszük, hogy $A[i]$ nem levélcsúcs).
- $lev(i : k..m) : \mathbb{L}$, ami akkor ad vissza igaz értéket, ha $A[i]$ a fa levele.
- $jobb(i : k..m) : \mathbb{Z}$, ahol $A[jobb(i)]$ az $A[i]$ jobb gyereke (feltesszük, hogy $A[i]$-nek van jobb gyereke).
- $vanjobb(i : k..m) : \mathbb{L}$, ami akkor ad vissza igaz értéket, ha $A[i]$-nek van jobb gyereke.
- $gy(i : k..m) : \mathbb{L}$, ami akkor ad vissza igaz értéket, ha $A[i]$ a fa gyökere.
- $sz(i : k..m) : \mathbb{Z}$, ahol $A[sz(i)]$ az $A[i]$ szülője (feltesszük, hogy $A[i]$ nem a fa gyökere).

E.5. Az $A[k..n]$ egész számok tömbje, $n \geq m \geq k-1$, $A[k..m]$ bináris maximumheap (A *heap* magyarul *piramis* illetve *kupac*, $m = k-1$ esetén a heap üres). A bináris maximumheap a tömbben szintfolytonosan ábrázolt kváziteljes (majdnem teljes), balra tömörített bináris fa, ahol minden csúcs $\geq$ mint a gyerekei. Írjuk meg az insIntoHeap$(A[], k, n, m, x)$ függvényt, ami beszúrja x-et a kupacba! A függvény aszerint tér vissza igaz ill. hamis értékkel, hogy sikeres vagy sikertelen volt-e a beszúrás. (Volt-e még az $A[k..n]$ tömbben szabad hely.) A beszúrást a fa magasságával

arányos műveletigénnyel hajtsuk végre!

E.6. Az $A[k..n]$ egész számok tömbje, $n \geq m \geq k-1$, $A[k..m]$ bináris maximumheap (A *heap* magyarul *piramis* illetve *kupac*, $m = k-1$ esetén a heap üres). A bináris maximumheap a tömbben szintfolytonosan ábrázolt kváziteljes (majdnem teljes), balra tömörített bináris fa, ahol minden csúcs $\geq$ mint a gyerekei. Írjuk meg a RemoveMaxFromHeap$(A[], n, m, k, x)$ függvényt, ami kiveszi, a kupacból törli, és x-ben visszaadja a kupac maximális elemét! A függvény aszerint tér vissza igaz ill. hamis értékkel, hogy sikeres vagy sikertelen volt-e a művelet (Volt-e még a kupacban elem.) A műveleltet a fa magasságával arányos műveletigénnyel hajtsuk végre!

E.7. Az $A[0..(m-1)]$ egész számok tömbje, k pozitív egész szám, $m \geq n \geq 0$, $A[0..(n-1)]$ k-áris maximumheap (A *heap* magyarul *piramis* illetve *kupac*). A k-áris maximumheap a tömbben szintfolytonosan ábrázolt kváziteljes, balra tömörített k-áris fa, ahol minden csúcs $\geq$ mint a gyerekei. Írjuk meg az insIntoHeap$(A[], n, m, k, x)$ függvényt, ami beszúrja x-et a kupacba! A függvény aszerint tér vissza igaz ill. hamis értékkel, hogy sikeres vagy sikertelen volt-e a beszúrás. (Volt-e még az $A[0..(m-1)]$ tömbben szabad hely.) A beszúrást a fa magassága és k szorzatával arányos műveletigénnyel hajtsuk végre! (Az i indexű csúcs gyermekei a $k*i+1, \ldots, k*i+k$ indexű csúcsok.)

E.8. Az $A[0..(m-1)]$ egész számok tömbje, k pozitív egész szám, $m \geq n \geq 0$, $A[0..(n-1)]$ k-áris maximumheap (A *heap* magyarul *piramis* illetve *kupac*). A k-áris maximumheap a tömbben szintfolytonosan ábrázolt kváziteljes, balra tömörített k-áris fa, ahol minden csúcs $\geq$ mint a gyerekei. Írjuk meg a RemoveMaxFromHeap$(A[], n, m, k, x)$ függvényt, ami kiveszi, a kupacból törli, és x-ben visszaadja a kupac maximális elemét! A függvény aszerint tér vissza igaz ill. hamis értékkel, hogy sikeres

vagy sikertelen volt-e a művelet (Volt-e még a kupacban elem.) A műveletet a fa magassága és k szorzatával arányos műveletigénnyel hajtsuk végre! (Az i indexű csúcs gyermekei a $k*i+1, \ldots, k*i+k$ indexű csúcsok.)

E.9. Valósítsuk meg rendezetlen tömb segítségével az előadásról ismert PrSor osztályt! Legyen a maxKivesz() függvény futási ideje lineáris, míg a prSorba(x) eljárás és a max() függvény futási ideje $\Theta(1)$. (Ötlet: A maximum helyét folyamatosan tartsuk nyilván!)

E.10. Valósítsuk meg rendezetlen, egyszerű láncolt lista segítségével az előadásról ismert PrSor osztályt! Legyen a maxKivesz() függvény futási ideje lineáris, míg a prSorba(x) eljárás és a max() függvény futási ideje $\Theta(1)$. (Ötlet: A maximum helyét folyamatosan tartsuk nyilván!)

Láncolt listák

L.1. Az L pointer egy monoton növekvően rendezett egyszerű láncolt lista első elemére mutat ($L \neq \oslash$).

Írjuk meg a duplDel(L, D) eljárást, ami a duplikált adatoknak csak az első előfordulását hagyja meg! $T(n) \in O(n)$, ahol n az L lista hossza (n a program számára nem adott). A lista tehát szigorúan monoton növekvő lesz. A feleslegessé váló elemekből hozzuk létre a D pointerű, monoton csökkenő, egyszerű láncolt listát!

L.2. Az F pointer egy monoton növekvően rendezett nemüres FKCL fejelemére mutat. A D pointer egy üres FKCL fejelemére mutat. A listák elemtípusa az ismert Elem2 típus.

Írjuk meg a duplDel(F, D) eljárást, ami a duplikált adatoknak csak az első előfordulását hagyja meg! $T(n) \in O(n)$, ahol n az F lista hossza (n a program számára nem adott). A lista tehát szigorúan monoton növekvő lesz. A feleslegessé váló elemekből hozzuk

létre a D fejpointerű, monoton növekvő FKCL-t! (A fejelem már megvan.)

L.3.1. Az $A[1..n]$ egész számok tömbje.

Írjuk meg a listaba$(A[1..n], F)$ eljárást, ami előállítja az $A[1..n]$ tömb által reprezentált absztrakt sorozat láncolt ábrázolását az F fejelemes listában. A szükséges listaelemeket a gyakorlatról ismert `new Elem` művelet segítségével nyerjük. Feltesszük, hogy ha nincs elég memória, akkor a `new` művelet null pointert ad vissza. Ebben az esetben írjuk ki azt, hogy „Memory Overflow!", aztán azonnal fejezzük be az eljárást! Ilyenkor az F listában csak azok az elemek legyenek, amelyeket sikeresen generáltunk! $T_{\text{listaba}}(n) \in O(n)$ legyen! (Feltesszük, hogy $T_{\text{new}} \in \Theta(1)$.)

L.3.2. Az $A[1..n]$ egész számok tömbje.

Írjuk meg a listaba$(A[1..n], L)$ eljárást, ami előállítja az $A[1..n]$ tömb által reprezentált absztrakt sorozat láncolt ábrázolását az L egyszerű listában. A szükséges listaelemeket a gyakorlatról ismert `new Elem` művelet segítségével nyerjük. Feltesszük, hogy ha nincs elég memória, akkor a `new` művelet null pointert ad vissza. Ebben az esetben írjuk ki azt, hogy „Memory Overflow!", aztán azonnal fejezzük be az eljárást! Ilyenkor az L egyszerű listában csak azok az elemek legyenek, amelyeket sikeresen generáltunk! $T_{\text{listaba}}(n) \in O(n)$ legyen! (Feltesszük, hogy $T_{\text{new}} \in \Theta(1)$.)

L.4. Az $A[1..n]$ egész számok tömbje.

Írjuk meg a listába$(A[1..n], F)$ eljárást, ami előállítja az $A[1..n]$ tömb által reprezentált absztrakt sorozat láncolt ábrázolását az F fejpointerű FKCL-ben. (A hívás pillanatában $F = \oslash$.) A szükséges listaelemeket a gyakorlatról ismert `new Elem2` művelet segítségével nyerjük. Feltesszük, hogy ha nincs elég memória, akkor a `new` művelet null pointert ad vissza. Ebben az esetben írjuk ki azt, hogy „Memory Overflow!", aztán azonnal fejezzük be az eljárást! Ilyenkor az F listában csak azok az elemek legyenek,

amelyeket sikeresen generáltunk! A „listába($A[1..n], F$)” eljárásra $T(n) \in O(n)$ legyen! (Feltesszük, hogy $T_{\text{new}} \in \Theta(1)$.)

L.5. Adott a $P[1..n]$ pointer tömb, amelynek elemei egyszerű láncolt listákat azonosítanak. (Mindegyik vagy $\oslash$ pointer, vagy egy lista első elemére mutat.)

Írjuk meg az összefűz($P[1..n], L$) eljárást, ami a listákat sorban egymás után fűzi az L egyszerű listába. $T(m, n) \in O(m+n)$, ahol m az eredeti listák összhossza (m a program számára ismeretlen).

L.6. Az L pointer egy rendezetlen, egyszerű láncolt lista első elemére mutat. Feltehető, hogy a lista nem üres.

Írjuk meg a MaxRend(L) eljárást, ami a listát monoton növekvően, L hosszától független, konstans mennyiségű memóriafelhasználással, maximumkiválasztással rendezi! $T(n) \in O(n^2)$, ahol n az L lista hossza (n a program számára nem adott).

[Felveszünk egy rendezett, kezdetben üres segédlistát. (Így inicializáljuk.) A rendezés menetekből áll. Minden menetben kiválasztjuk a rendezetlen lista legnagyobb elemét, és átfűzzük a rendezett lista elejére.]

L.7. Az L pointer egy FKCL fejelemére mutat.

Írjuk meg a MaxRend(L) eljárást, ami a listát monoton növekvően, maximum kiválasztásos rendezéssel, helyben rendezi, ahol $T(n) \in O(n^2)$ (n az L lista hossza, a program számára *nem* adott).

Algoritmus: A lista sorban egy rendezetlen és egy rendezett szakaszra bomlik. A rendezett rész kezdetben üres. A rendezett szakaszban az elemek monoton növekvően helyezkednek el, és a legkisebb elem is nagyobb vagy egyenlő, mint a rendezetlen rész maximuma. A rendezetlen rész maximális elemét minden menetben átfűzzük a rendezett szakasz elejére. Ha a rendezetlen résznek már csak legfeljebb egy eleme van, kész vagyunk.

L.8. Az $L1, L2$ pointerek egy-egy szigorúan monoton FKCL fejelemére mutatnak.

Írjuk meg a unio($L1, L2$) eljárást, ami az $L1$ lista elemei közé fésüli az $L2$ lista, $L1$ listán nem szereplő elemeit, azaz átfűzi őket rendezett módon az $L1$ elemei közé! Az $L2$ listában így a két lista, mint halmaz metszetének elemei maradnak, az $L1$ listában pedig előáll a két lista, mint halmaz uniója. Minkét lista szigorúan monoton növekvő marad. $T(n_1, n_2) \in O(n_1 + n_2)$, ahol n_1 és n_2 a két lista hossza (a program számára *egyik sem* adott).

(Végigmegyünk egyszer az $L1$ listán, és minden elemhez, tőle balra átfűzzük a másik listáról a nála kisebb elemeket. Végül a fejelemtől balra fűzzük az $L2$ listának az eredeti $L1$-belieknél nagyobb elemeit. Az $L2$ listán is csak egyszer menjünk végig!)

L.9. Az L pointer egy FKCL fejelemére mutat. $P[1..k]$ definiálatlan pointer tömb, m definiálatlan egész szám.

Írjuk meg a növBont($L, P[1..k], m$) eljárást, ami meghatározza és m-ben visszadja az L-beli, monoton növekvően rendezett szakaszok számát. Ezen szakaszok első elemeire az eljárás végrehajtásának eredményeként sorban a $P[1..k]$ tömb $P[1..m]$ résztömbjének elemei mutatnak. $T(n) \in O(n)$, ahol n az L lista hossza (a program számára *nem* adott).

Feltehető, hogy $P[1..k]$-nak elég sok eleme van, azaz az eljárás által kiszámolt m értékre $k \geq m$. A szigorúan monoton csökkenő részeket egyelemű monoton növekvő szakaszok sorozatának tekintjük. Pl. $\langle 5, 6, 4, 2, 1, 3\rangle$ monoton növekvő szakaszai $[\langle 5, 6\rangle, \langle 4\rangle, \langle 2\rangle, \langle 1, 3\rangle]$, ahol $m = 4$. Az eljárás az L listát nem változtatja meg, csak m-et és $P[1..m]$-et állítja be.

Bináris fák

Definíciók:

A t bináris fa egy csúcsa méret szerint kiegyensúlyozott $\Longleftrightarrow$ a két részfájának a mérete legfeljebb eggyel tér el egymástól.

A t bináris fa méret szerint kiegyensúlyozott $\Longleftrightarrow$ minden csúcsa méret szerint kiegyensúlyozott.

B.1. Írjuk meg a toBinTree($A[1..n], t$) utasítással meghívható eljárást, ami előállítja a t:Csúcs*, méret szerint kiegyensúlyozott bináris fát $MT(n) \in O(n)$ maximális műveletigénnyel, ahol a fa Inorder bejárása sorban az $A[1..n]$ elemeit adja. (A bejárást nem kell megírni.) **Algoritmus**: A tömb középső eleme lesz a fa gyökere, a középső elemtől balra és jobbra levő két résztömbből pedig, rekurzívan, a megfelelő oldali részfákat generáljuk. Üres (rész)tömbből természetesen üres fa lesz. A szükséges csúcsokat az előadásról és a gyakorlatról ismert **new** utasítással allokáljuk!

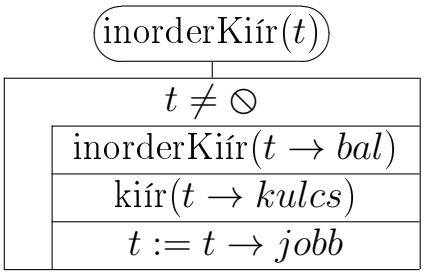
B.2. Írjuk meg az előző programot abban az esetben, ha a fa csúcsaiban „szülő pointerek" is vannak, azaz az előállítandó t fa Csúcs3* típusú.

B.3. Egy nemüres láncolt bináris fa csúcsaiban sz szülő pointerek is vannak. A p pointer a fa egy csúcsára mutat. Írjuk meg az inorder_prev(p), inorder_next(p), preorder_prev(p), preorder_next(p), postorder_prev(p), postorder_next(p), függvényeket, ami a $(*p)$ csúcs inorder/preorder/postorder bejárás szerinti megelőzőjének/rákövetkezőjének címét adja vissza! Ha ilyen nincs, a $\oslash$ extremális címet adja vissza! $MT(h) \in O(h)$, ahol h a $(*p)$ csúcsot tartalmazó bináris fa magassága. (A program számára h nem adott.)

B.4. Egy bináris fa *méret szerint kiegyensúlyozott*, ha tetszőleges nemüres részfája bal és jobb részfájának mérete legfeljebb eggyel térhet el. Bizonyítsuk be, hogy a méret szerint kiegyensúlyozott bináris fák halmaza a majdnem teljes bináris fák halmazának valódi részhalmaza!

B.5. Bizonyítsuk be, hogy ha egy majdnem teljes, balra tömörített bináris fát, szintfolytonosan az $A[0..(m-1)]$, nullától indexelt tömbbel ábrázoljuk, akkor $A[i]$ gyerekei $A[2i+1]$ illetve $A[2i+2]$, a $2i+1 < m$, illetve a $2i+2 < m$ feltétellel, $A[j]$ szülője pedig $j > 0$ esetén $A[\lfloor \frac{j-1}{2} \rfloor]$!

B.6. Bizonyítsuk be, hogy az alábbi program a t *bináris keresőfa* kulcsait szigorúan monoton növekvő sorrendben írja ki! (Ötlet: teljes indukció t mérete vagy magassága szerint.)



Bizonyítsuk be azt is, hogy ha a fenti program a t *bináris fa* kulcsait szigorúan monoton növekvő sorrendben írja ki, akkor az *keresőfa*!

B.7. Írjuk meg $t \neq \oslash$, láncoltan ábrázolt bináris keresőfa esetén a maximális kulcs kiolvasása / kivétele műveleteket! (A csúcsok nem tartalmaznak „szülő" pointert.) Mekkora lesz a futási idő? Miért? Írjuk át az előadásról ismert bináris keresőfa műveleteket, és a fenti műveleteket is szülő pointeres csúcsok esetére! Próbáljuk meg a nemrekurzív programokat rekurzívvá, a rekurzívakat nemrekurzívvá átírni, megtartva a futási idők nagyságrendjét!

B.8. Igaz-e, hogy véletlen építésű bináris keresőfák esetén a keres(t, k), beszúr(t, k), töröl(t, k), min(t), minKivesz$(t, minp)$, max(t), maxKivesz$(t, maxp)$, műveletek bármelyikére $mT(n) \in \Theta(1)$, ahol n a t bináris keresőfa mérete?

B.9. Írjunk olyan eljárást, ami egy szigorúan monoton növekvő vektorból majdnem teljes bináris keresőfa másolatot készít, $O(n)$ futási idővel!

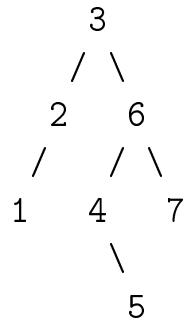
AVL fák

Jelölés: Az alábbi feladatokban a konkrét AVL fákat a bináris fák szokásos szöveges, azaz zárójeles reprezentációban adjuk meg.

Az üres fa reprezentációja az üres szöveg. Tetszőleges nemüres bináris fa szöveges formája

„(BalRészfaRepr GyökérKulcsa JobbRészfaRepr)”

alakú, tehát pl. a levélelemekhez tartozó részfák „(LevélKulcsa)” alakban jelennek meg. A könnyebb olvashatóság kedvéért a példákban, a részfák különböző szintjei szerint, háromféle zárójelpárt használunk: { }, [ ] és ( ). Pl.
{ [(1) 2] 3 [(4 {5}) 6 (7)] }
az alábbi fa zárójeles, azaz szöveges formája.



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

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

belső csúcsok egyensúlyait is, a szokásos módon! Rajzoljuk le a hat általános kiegyensúlyozási séma közül azokat, amiket alkalmaztunk!

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

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

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

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

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

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

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

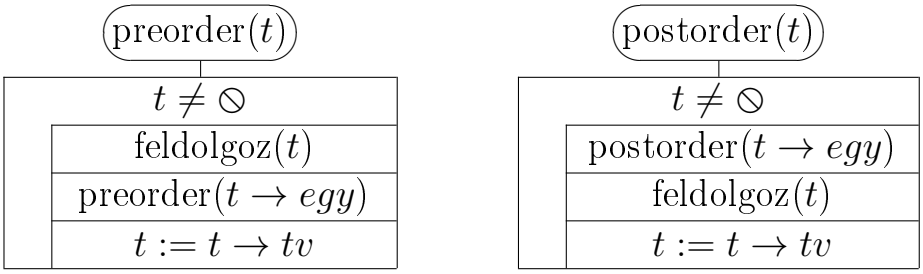
Általános fák

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

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

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

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



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

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

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

B+ fák

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

Megjegyzés:
- Egy negyedfokú B+ fa tetszőleges részfájának szöveges formája, ha a részfa nem egyetlen levélcsúcsból áll, a következő sémák valamelyikére illeszkedik:

[2] 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.

[3] preorder, inorder, postorder, szintfolytonos

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

+.1. Vegyük a legelső B+ fa illusztrációt a fent hivatkozott „B+ fa.pdf" fájlból! Szövegesen:
{ [(1 4) 9 (9 10) 11 (11 12)] 13 [(13 15) 16 (16 20 25)] }.
Szúrjuk be az 5-öt a fenti fába, majd a 6, 17, 8 és 7 számokat sorban az eredményül adódó B+ fákba! Rajzoljuk le az eredeti fát, majd a sorban az egyes beszúrások eredményeit!

+.2. Adott az alábbi B+ fa:
{ [2 4] 5 [6 8 10] 12 [12 14] 15 [16 18] }.
Szúrjuk be az 5 kulcsot a fenti fába! Rajzoljuk le az eredeti fát és a beszúrás eredményét is!

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

+.4. A t pointer egy d fokszámú, h magasságú B+ fa gyökércsúcsára mutat. Írjuk meg a printBplusTree0(t, d, h) eljárást, ami kiíratja a fát szöveges formában, csak kerek zárójeleket használva! A levelekben lévő kulcsokhoz tartozó, az adatrekordokra

mutató hivatkozásokat ebben az eljárásban nem vesszük figyelembe. $MT(n,d) \in O(n * d)$, ahol n a t fa csúcsainak száma (ami az eljárás számára nem adott).

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