Algoritmusok és adatszerkezetek I. vizsga, 2015.12.15.

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

1. Szemléltessük a kupacrendezést az alábbi tömbre!
`< 2; 9; 1; 3; 4; 6 >`
Minden lesüllyesztés előtt jelöljük a csúcs mellett egy kis körbe tett sorszámmal, hogy ez a rendezés során a hányadik lesüllyesztés, akkor is, ha az aktuális lesüllyesztés nem mozdítja el a csúcsban lévő kulcsot! Minden valódi lesüllyesztés előtt jelöljük a lesüllyesztés irányát és útvonalát, s utána rajzoljuk újra a fát! (15p)

2. Adott a { [(2) 4 ({6} 8 {10})] 12 [14 (16)] } AVL fa. Rajzoljuk le a fát a 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 csúcsok egyensúlyait is, a szokásos módon! Rajzoljuk le a hat általános kiegyensúlyozási séma közül azokat, amiket alkalmaztunk! (15p)

3. Valósítsuk meg rendezetlen, statikus tömb segítségével az előadásról ismert **PrSor** (elsőbbségi sor) osztályt! Legyen a maxKivesz() függvény futási ideje lineáris, míg a konstruktor, a prSorba(x) eljárás, a max(), a tele_e() és az üres_e() függvény metódusok futási ideje $\Theta(1)$. (Ötlet: A maximum helyét folyamatosan tartsuk nyilván!) (20p)

4. Az `L` pointer egy FKCL (fejelemes, kétirányú, ciklikus, láncolt lista) fejelemére mutat. A lista nem üres, kezeléséhez felhasználhatók az előadásról ismert Elem2 osztály műveletei. Írjuk meg a MaxVégére(L) eljárást, ami a lista legnagyobb kulcsú elemét a lista végére fűzi! A program a listán csak egyszer menjen végig! $T(L) \in O(hossz(L))$. (15p)

5. Adjuk meg az összefésülő rendezés (merge sort) struktogramját egyszerű láncolt listákra! Adjuk meg a „szétvág" eljárást is, az „összefésül" eljárást és a lista hosszát kiszámító függvényt viszont nem kell részletezni! Mekkora lesz a műveletigény? Röviden indokoljuk állításunkat! (20p)

6. Mondjuk ki az összehasonlító rendezések műveletigényének alsó korlátjára vonatkozó két alaptételt! Bizonyítsuk be a maximális műveletigényre vonatkozót! Mi a jelentősége ezeknek a tételeknek? (15p)

Algoritmusok és adatszerkezetek I. vizsga, 2015.12.22.

Az eljárásokat és függvényeket megfelelően elnevezett és paraméterezett struktogramok segítségével adjuk meg! Ne feledkezzünk meg a referencia paraméterek szükség szerinti jelöléséről sem! A változókat alapértelmezésben a struktogramra vonatkozóan lokálisnak tekintjük.

1. A bináris fa fogalmát ismertnek feltételezve, mondjuk ki a kupac definícióját! Szemléltessük az alábbi kupacra a 9, majd az **eredmény kupacra** a 8 beszúrásának műveletét! `< 8; 8; 6; 6; 5; 2; 3; 1; 5; 4 >`. Szemléltessük az **eredeti kupacra** a maxKivesz eljárás kétszeri végrehajtását! Minden művelet után rajzoljuk újra a fát! (15p)

2. A bináris fa fogalmát ismertnek feltételezve, definiáljuk a bináris keresőfa fogalmát! Írjuk meg a maxKivesz(t, max) utasítással meghívható, ciklust nem tartalmazó eljárást, ami a t bináris keresőfa maximális kulcsát max-ba másolja, majd a megfelelő csúcsot törli a fából! A felszabaduló memóriát adjuk vissza a szabad területnek! $T(h) \in O(h)$, ahol $h = h(t)$ (15p)

3. A bináris keresőfa fogalmát ismertnek feltételezve, mondjuk ki az AVL fa meghatározásához szükséges definíciókat!

Adott az { [(1) 2 (3 {4})] 6 [(7) 8] } AVL fa. Rajzoljuk le a fát a 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 csúcsok egyensúlyait is, a szokásos módon! Rajzoljuk le a hat általános kiegyensúlyozási séma közül azokat, amiket alkalmaztunk! (20p)

4. Az L_1 és L_2 pointerek két egyszerű láncolt listát azonosítanak. Írjuk meg az összefűz(L_1, L_2) eljárást, ami $MT(n) \in \Theta(n)$ és $mT(n) \in \Theta(1)$ $(n = |L_1|)$ műveletigénnyel az L_1 lista után fűzi az L_2 listát! (15p)

5. Adjuk meg a beszúró rendezés optimalizált változatának struktogramját (beszúró_rendezés$(A[1..n])$)! Mi a fő ciklus invariánsa? Adjuk meg a minimális, a maximális és az átlagos műveletigényt! Indokoljuk állításainkat! (20p)

6. Tegyük fel, hogy a függvényeink a nemnegatív egész számok halmazáról a nemnegatív valós számok halmazára képeznek, és g is egy ilyen függvény! Adjuk meg a $\Theta(g)$, az $O(g)$ és az $\omega(g)$ függvényhalmazok definícióját!
6.a, Igaz-e, hogy $2^{n+1} \in \Theta(2^n)$. Miért? **6.b,** Igaz-e, hogy $2^{2n} \in O(2^n)$. Miért? **6.c,** Igaz-e, hogy $2^{2n} \in \omega(2^n)$. Miért? (15p)

Név: ..... *Neptun kód:* .....

Algoritmusok és adatszerkezetek I. vizsga, 2016.01.05.

Az eljárásokat és függvényeket megfelelően elnevezett és paraméterezett struktogramok segítségével adjuk meg! Ne feledkezzünk meg a referencia paraméterek szükség szerinti jelöléséről sem! A változókat alapértelmezésben a struktogramra vonatkozóan lokálisnak tekintjük.

1. Mi a rendezési feladat fogalma? Mekkora a beszúró rendezés műveletigénye? Szemléltessük a beszúró rendezést (insertion sort) a következő vektorra! `< 7; 4; 1; 4; 3; 8; 9 >`. Szemléltessük az előbbi vektorra az összefésülő rendezést (mergesort) is! Egyenlőtlen vágás esetén a bal oldali részvektor legyen eggyel rövidebb! Mekkora az összefésülő rendezés műveletigénye? Érdemes-e az előbbi gyors és lassú rendezéseket egyetlen rendezésben egyesíteni? Hogyan? Miért? (20p)

2. A d-edfokú B+ fák leveleinek milyen tulajdonságait ismeri? – Adott a { | (2 4) 8 (8 10 12) 14 (14 16) 18 (20 22) | 24 | (24 26 28) 30 (30 32) | } negyedfokú B+ fa. Rajzoljuk le a fát! Szemléltessük az előadáson elhangzott algoritmus szerint a 18, a 25 és a 9 beszúrását, **mindhárom esetben az eredeti fára**! (20p)

3. Az L_1, L_2 pointerek egy-egy szigorúan monoton növekvő FKCL (fejelemes, kétirányú, ciklikus, láncolt lista) fejelemére mutatnak. A listák kezeléséhez felhasználhatók az előadásról isnert Elem2 osztály műveletei. **Írjuk meg** a különbség(L_1, L_2) eljárást, ami az L_1 lista elemei közül törli az L_2 listán is szereplő elemeket! Az L_2 lista változatlan, de az L_1 is szigorúan monoton növekvő marad. Mindkét listán legfeljebb egyszer menjünk végig! A felszabaduló listaelemeket adjuk vissza a szabad területnek! $MT(n_1, n_2) \in O(n_1 + n_2)$, $mT(n_1, n_2) \in O(min(n_1, n_2))$, ahol n_1 az L_1, n_2 az L_2 lista hossza. (20p)

4. A bináris fa fogalmát ismertnek feltételezve, definiáljuk a bináris keresőfa fogalmát! **Írjuk meg** a beszúr(t, k, s) – ciklust nem tartalmazó – $T(h) \in O(h)$ hatékonyságú rekurzív eljárást, ami megpróbál beszúrni a t bináris keresőfába egy k kulcsú csúcsot (akkor tudja beszúrni, ha nem talál ilyet), és az s, logikai típusú paraméterben visszaadja, hogy sikeres volt-e a beszúrás! A fa csúcsai Csúcs típusúak, azaz szülő pointert nem tartalmaznak. Igaz-e, hogy a fenti beszúr eljárásra $mT(h) \in \Theta(1)$? Miért? (20p)

5. Bizonyítsuk be a következő állítást! Tetszőleges n csúcsú és h magasságú bináris fára $n - 1 \geq h \geq \lfloor \lg n \rfloor$. Mikor lesz $h = n - 1$ és miért? Bizonyítsuk be, hogy majdnem teljes bináris fák esetén a $h = \lfloor \lg n \rfloor$ egyenlőség teljesül! (20p)

Név: ..... *Neptun kód:* .....

Algoritmusok és adatszerkezetek I. vizsga, 2016.01.12.

Az eljárásokat és függvényeket megfelelően elnevezett és paraméterezett struktogramok segítségével adjuk meg! Ne feledkezzünk meg a referencia paraméterek szükség szerinti jelöléséről sem! A változókat alapértelmezésben a struktogramra vonatkozóan lokálisnak tekintjük.

1. Írjuk le struktogram formában a gyorsrendezés (quicksort) programját! Szemléltessük a programban megadott szétvág/helyrevisz függvény/eljárás működését a következő vektorra! `< 4; 9; 8; 1; 3; 7; 4 >`. Mekkora a gyorsrendezés műveletigénye? Érdemes-e a gyorsrendezést és a beszúró rendezést egyetlen rendezésben egyesíteni? Hogyan? Miért? (20p)

2. Adott az { [(1 2) 3 (4 5)] 6 [(6 7) 8 (8 9 10) 11 (12 13) 14 (14 15)] } negyedfokú B+ fa. Rajzoljuk le a fát! Szemléltessük az ismert algoritmus szerint a 8, a 13 és az 1 törlését, **mindhárom esetben az eredeti fára**! (20p)

3. Az L_1, L_2 pointerek egy-egy monoton növekvő FL (fejelemes, egyirányú, nemciklikus, láncolt lista) fejelemére mutatnak. **Írjuk meg** az összefésül(L_1, L_2) eljárást, ami az L_1 lista elemei közé fésüli az L_2 lista elemeit, azaz átfűzi őket rendezett módon az L_1 elemei közé! (Végül egyetlen lépésben az L_1 lista végére fűzi az L_2 listának az eredeti L_1-belieknél $\geq$ elemeit, ha vannak ilyenek.) Az L_1 lista monoton növekvő marad, L_2 üres lesz. Mindkét listán legfeljebb egyszer menjünk végig! $MT(n_1, n_2) \in O(n_1 + n_2)$, $mT(n_1, n_2) \in O(min(n_1, n_2))$, ahol n_1 az L_1, n_2 az L_2 hossza. (20p)

4. A bináris fa fogalmát ismertnek feltételezve, definiáljuk a bináris keresőfa fogalmát! **Írjuk meg** a gykTöröl(t) – ciklust nem tartalmazó – $T(h) \in O(h)$ hatékonyságú rekurzív eljárást, ami a t nemüres bináris keresőfából törli a gyökércsúcsban található kulcsot a megfelelő csúccsal együtt! (Nem biztos, hogy a ténylegesen törölt csúcs t gyökércsúcsa.) A felhasznált eljárások, függvények kódját is részletezzük! A felszabaduló memóriát adjuk vissza a szabad területnek! A fa csúcsai Csúcs típusúak, azaz szülő pointert nem tartalmaznak. Igaz-e, hogy a gykTöröl eljárásra $mT(h) \in \Theta(1)$? Miért? (20p)

5. Bizonyítsuk be a következő állítást! Tetszőleges n csúcsú nemüres kiegyensúlyozott bináris fa h magasságára $h \leq 1,45 \lg n$. (Elég a bizonyítás vázlata: az $\langle n_h \rangle$ és az $\langle F_h \rangle$ sorozatok jelentése és megadása, összefüggés a két sorozat között, n-re a kezdeti és a végső alsó becslés, ebből pedig h-ra felső becslés.) (20p)

Név: ..... *Neptun kód:* .....

Algoritmusok és adatszerkezetek I. vizsga, 2016.01.19.

1. Egy bináris fa mikor *szigorúan bináris*? Mikor *teljes*? Mikor *majdnem teljes*? Ez utóbbi mikor *balra tömörített*, és mikor *kupac*? Szemléltessük az alábbi vektorban ábrázolt kupacra a 7, majd az **eredmény kupacra** a 8 beszúrásának műveletét! $\langle 7; 5; 7; 5; 3; 4; 6; 2; 1; 2\rangle$. Szemléltessük az **eredeti kupacra** a maxKivesz eljárás végrehajtását! Mindkét beszúrás és a maxKivesz végrehajtása után is rajzoljuk újra a fát! (20p)

2. A bináris keresőfákat ismertnek feltételezve, mondjuk ki az AVL fa meghatározásához szükséges definíciókat! Rajzoljuk le a következő AVL fát a 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 csúcsok egyensúlyait is, a szokásos módon! Rajzoljuk le a hat általános kiegyensúlyozási séma közül azokat, amiket alkalmaztunk! (20p)

3. Egy láncolt lista mikor *egyszerű*? Az L pointer egy nemüres, egyszerű lista első elemére mutat. Írjuk meg a **szétoszt**(L, L_1, L_2) eljárást, ami L első elemét a helyén hagyja, míg – a megfelelő listaelemek átfűzésével – a többi, az elsőnél kisebb-egyenlő kulcsú elemből az L_1, a nagyobb kulcsú elemekből pedig az L_2 listát építi fel. Az L lista végül egyelemű lesz. Az L_1 és L_2 egyszerű listákban az elemek sorrendje tetszőleges, akár az eredeti sorrendjük fordítottja is lehet (a program így a legegyszerűbb). $T(n) \in \Theta(n)$, ahol n az L lista eredeti hossza. A program az L listát csak egyszer dolgozza fel! (20p)

4. A bináris fa fogalmát ismertnek feltételezve, mit értünk a bináris keresőfa alatt? Adott a **gykTöröl**(f), $T(h(f)) \in O(h(f))$ hatékonyságú eljárás, ami az f nemüres bináris keresőfából törli a gyökércsúcsban található kulcsot a megfelelő csúccsal együtt. (Ezt nem kell megírni.) Ennek segítségével írjuk meg a **töröl**(t, k, s) – ciklust nem tartalmazó – szintén $T(h(t)) \in O(h(t))$ hatékonyságú rekurzív eljárást, ami az s, logikai típusú paraméterben visszaadja, hogy talált-e a t bináris keresőfában k kulcsú csúcsot, és ha talált, a kulcsot és a megfelelő csúcsot törli a fából. A fát kizárólag a gykTöröl(f) eljárás segítségével szabad módosítani. A fa csúcsai Csúcs típusúak, azaz szülő pointert nem tartalmaznak. (20p)

5. Adjuk meg a **kupacrendezés**$(A[1..n])$ és segédeljárásai struktogramjait! Igaz-e, hogy $MT(n) \in \Theta(n \lg n)$? Miért? Igaz-e, hogy $mT(n) \in \Theta(n)$? Miért? (20p)

Név: ..... *Neptun kód:* .....

Algoritmusok és adatszerkezetek I. vizsga, 2016.01.21.

1. Egy bináris fa mikor *szigorúan bináris*? Mikor *teljes*? Mikor *majdnem teljes*? Ez utóbbi mikor *balra tömörített*, és mikor *kupac*? Szemléltessük a kupacrendezést a következő tömbre! – $\langle 3; 9; 8; 2; 4; 6; 7; 5\rangle$ – Minden lesüllyesztés előtt jelöljük a csúcs mellett egy kis körbe tett sorszámmal, hogy ez a rendezés során a hányadik lesüllyesztés; akkor is, ha az aktuális lesüllyesztés nem mozdítja el a csúcsban lévő kulcsot! Minden valódi lesüllyesztés előtt jelöljük a lesüllyesztés irányát és útvonalát, s utána rajzoljuk újra a fát! A szemléltetést elég addig a pillantig elvégezni, amíg a vektor utolsó három eleme a végső helyére kerül. (20p)

2. A bináris fákat ismertnek feltételezve, mondjuk ki az AVL fa meghatározásához szükséges definíciókat! Rajzoljuk le a következő AVL fát a 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 csúcsok egyensúlyait is, a szokásos módon! Rajzoljuk le a hat általános kiegyensúlyozási séma közül azokat, amiket alkalmaztunk! (20p)

3. Az $A[1..m]$ tömb első n elemében egy bináris kupacot tárolunk $(m > 0 \wedge 0 \leq n \leq m)$.

Írjuk meg a **kupacba**$(A[1..m], n, x)$ függvényt, ami beteszi a kupacba x-et! Akkor ad vissza **igaz** logikai értéket, ha a művelet sikeres volt. Próbáljunk meg minél hatékonyabb algoritmust írni! Mekkora a műveletigénye? Miért? (20p)

4. L_1 és L_2 is egy-egy fejelemes, egyirányú, nemciklikus láncolt lista fejelemére mutató pointer. Az L_2 lista kezdetben üres.

Írjuk meg a **páratlan_páros**(L_1, L_2) eljárást, ami az L_1 lista, eredetileg páros *sorszámú* elemeit **átfűzi** az L_2 listába! Az L_1 listában tehát, az eredetileg páratlan sorszámú elemei maradnak. A program minden listaelemet csak egyszer dolgozzon fel! $T(n) \in \Theta(n)$. A listaelemeknek csak a *mut* (mutató) mezőjéhez férünk hozzá. (20p)

5. Adjuk meg az **összefésülő_rendezés**$(A[1..n])$ és segédeljárásai struktogramjait! Igaz-e, hogy $MT(n) \in \Theta(n \lg n)$? Miért? (20p)

Név: ..... *Neptun kód:* .....

Algoritmusok és adatszerkezetek I. vizsga, 2016.01.26.

1. **(1.a)** Hozzuk postfix formára az „`5+8*(4/(9-7))-6*2/3`” kifejezést, és **(1.b)** értékeljük ki az „`58497-/*+62*3/-`” lengyel formát, a gyakorlatról ismert algoritmusokat szemléltetve! (Feltesszük, hogy mindegyik szám egyjegyű, és mindegyik operátornak két operandusa van.) A vermet minden egyes kipakolás után újra kell lerajzolni.

Mekkora a lengyel forma kiértékelő algoritmus műveletigénye, ha mindegyik operandus egyjegyű szám, és mindegyik operátornak két operandusa van? Miért? (20p)

2. Milyen magas egy B+ fa? Milyen kapcsolatban áll ez a keresés, a beszúrás és a törlés műveletigényével? Adott az
{ [(1 2) 3 (4 5)] 6 [(7 8) 9 (9 10) 11 (11 12 13) 14 (15 16)] }
negyedfokú B+ fa. Rajzoljuk le a fát! Szemléltessük az előadásról ismert algoritmus szerint a 11, a 15 és a 4 törlését, **mindhárom esetben az eredeti fára**! (20p)

3. Az $A[1..m]$ tömb első n elemében egy bináris kupacot tárolunk ($m > 0 \wedge 0 \leq n \leq m$).

Írjuk meg a **maxKivesz**$(A[1..m], n, x)$ függvényt, ami kiveszi a kupacból a legnagyobb elemét, és x-be teszi! Akkor ad vissza **igaz** logikai értéket, ha a művelet sikeres volt. Próbáljunk meg minél hatékonyabb algoritmust írni! Mekkora a műveletigénye? Miért? (20p)

4. Az L_1 pointer egy nemüres, fejelemes láncolt lista fejelemére mutat, az L_2 pointer pedig egy egyszerű láncolt lista elejére ($L_2 = \emptyset$ is lehet).

Írjuk meg a **maxÁtfűz**(L_1, L_2) eljárást, ami az L_1 lista legnagyobb kulcsú elemét *átfűzi* az L_2 lista elejére! A program az L_1 listán csak egyszer menjen végig! $T(n) \in \Theta(n)$, ahol n az L_1 lista hossza. A listaelemeknek a *kulcs* és a *mut* (mutató) mezőkön kívül más részei is lehetnek, de ezeket nem ismerjük. (A listaelemeket tehát nem tudjuk lemásolni.) (20p)

5. A bináris fákat ismertnek feltételezve, mondjuk ki az AVL fa meghatározásához szükséges definíciókat! Az AVL fa mérete és magassága között milyen összefüggést ismer? Mi ennek a jelentősége az AVL fa műveletei szempontjából? Melyek ezek a műveletek?

Adjuk meg az előadásról ismert **AVLbeszúr**(t, k, d) rekurzív eljárás struktogramját, ami a t AVL fába beszúrja a k kulcsot, a d logikai típusú paraméterben pedig visszadja, hogy a művelet hatására nőtt-e a fa magassága! A **balRészfaNőtt**(t, d) és a **jobbRészfaNőtt**(t, d) segédeljárásokat nem kell részletezni. (20p)