Algoritmusok és adatszerkezetek I. régebbi vizsgakérdések.

Ásványi Tibor – asvanyi@inf.elte.hu

2023. május 26.

A vizsgákon természetesen olyan kérdések is szerepelhetnek, amelyek a régebbi vizsgákon nem voltak. Az alábbiakban csupán mintákat kívánok adni.

Struktogram készítésekor a feladat része (1) a paraméterek típusának megadása, (2) a cím szerinti paraméterátvétel szükség szerinti jelölése, (3) *függvények esetében* a visszatérési típus megadása, (4) a láncolt adatszerkezetek elemtípusának (pl. E1, E2, Node) UML leírása.

(A tömbök, struktúrák, objektumok, sztringek, absztrakt adatszerkezetek [minden, ami nem skalár] csak cím szerint adhatók át, így ezeknél a paraméter átvételi módot nem jelöljük. A skalárok [számok, karakterek, logikai változók, pointerek] viszont alapértelmezésben érték szerint adódnak át. Itt a cím szerint átvett paramétereket – kizárólag a formális paraméter listán – jelölni kell.)

Az algoritmusok konkrét példákon való bemutatásánál a működést alapértelmezésben az előadáson tanultak szerint kell szemléltetni.

A vizsgákon négy összetett feladat van, ahol egy feladaton belül adott témakörhöz kapcsolódóan több feladattípushoz tartozó részfeladat lehet.

Feladattípusok:
(1) Lejátszós feladat, amiben egy tanult algoritmus működését kell szemléltetni adott inputra, az előadásról ismert módon.
(2) Tanult definíciók, tételek kimondása.
(3a) Tanult algoritmus által megoldott feladat,
(3b) az algoritmus struktogramja,
(3c) helyességének indoklása,
(3d) aszimptotikus hatékonysága (műveletigény, esetleg tárigény is),
(3e) aszimptotikus hatékonyságának kiszámítása.

(4) Tanult tétel bizonyítása.
(5) Kreatív feladat, amelyben adott feladat megoldására struktogramot kell írni.
(6) Kreatív bizonyítási és számítási feladatok.

A kreatív feladatok általában az előadásról vagy a gyakorlatról ismerős, és ott megoldott problémákhoz hasonló feladatok megoldását jelentik.

Négy (összetett) vizsgakérdés lesz. Mindegyik 25 pontot ér. A ponthatárok: 85 $\to$ jeles ; 70 $\to$ jó ; 55 $\to$ közepes ; 40 $\to$ elégséges.

1. Függvények aszimptotikus viselkedése

1. Tegyük fel, hogy $g : \mathbb{N} \to \mathbb{R}$, aszimptotikusan nemnegatív függvény!
1.a, Adjuk meg az $O(g)$ és az $\Omega(g)$ függvényhalmazok definícióját!
1.b, Milyen alapvető összefüggést ismer az $O(g)$, az $\Omega(g)$ és a $\Theta(g)$ függvényhalmazok között?
1.c, Igaz-e, hogy $(3n+4)^2 \in \Theta(n^2)$? Miért?
1.d, Igaz-e, hogy $n^n \in \Omega(2^n)$? Miért?
1.e, Igaz-e, hogy $1000n^2(\lg n) \in O(n^3)$? Miért?

2. Tegyük fel, hogy $g : \mathbb{N} \to \mathbb{R}$, aszimptotikusan nemnegatív függvény!
2.a, Adjuk meg az $O(g)$ és az $\Omega(g)$ függvényhalmazok definícióját!
2.b, Milyen alapvető összefüggést ismer az $O(g)$, az $\Omega(g)$ és a $\Theta(g)$ függvényhalmazok között?
2.c, Igaz-e, hogy $(2n+1)(3n-4) \in \Theta(n^2)$? Miért?
2.d, Igaz-e, hogy $2^n \in O(n!)$? Miért?
2.e, Igaz-e, hogy $(n \lg n) \in \Theta(n^2)$? Miért?

2. Összehasonlító rendezések

1.a, Osztályozza az előadásról ismert négy összehasonlító rendezést műveletigény $(MT(n), AT(n), mT(n))$ szempontjából!
1.b, Mondja ki az összehasonlító rendezések maximális műveletigényének alsó korlátjára vonatkozó alaptételt!
1.c, Bizonyítsa be a kulcsösszehasonlítások számának maximumára vonatkozó lemmát!
1.d, Mi a jelentősége ennek a tételnek?

2.a, Osztályozza az előadásról ismert négy összehasonlító rendezést műveletigény $(MT(n), AT(n), mT(n))$ szempontjából!

2.b, Mit tud mondani a rendezések minimális műveletigényének alsó korlátjáról? Miért?
2.c, Osztályozza az előadásról ismert négy összehasonlító tömbrendezést (maximális és minimális) tárigény szempontjából! (A tárigény = az algoritmus végrehajtásához használt temporális adatok számára + az alprogram hívások számára szükséges tárterület.)
2.d, A fenti négy rendezés közül melyeket javasolná egyirányú láncolt listák rendezésére is? Mit tud mondani ezen listarendezések tárigényéről?
2.e, A fenti négy rendezés közül melyiket javasolná előrendezett, kétirányú láncolt listák rendezésére? Miért?

2.1. Beszúró rendezés

1.a, Szemléltesse a beszúró rendezést az előadásról ismert módon az $\langle 5; 7; 6; 4; 8; 4 \rangle$ tömbre! Az utolsó beszúrást részletezze!
1.b, Adja meg a megfelelő struktogramot!
1.c, Számolja ki a struktogramjának megfelelő pontos $MT(n)$ és $mT(n)$ értékeket!
1.d, Mit jelentenek ezek az eredmények aszimptotikusan?

2.a, Mikor nevezünk egy rendezést stabilnak?
2.b, Adja meg fejelemes, egyirányú, nemciklikus listára (H1L) a beszúró rendezés struktogramját! A listát a *key* mezők szerint monoton növekvően kell rendezni. (A listaelemeknek a szokásos *key* és *next* mezőkön kívül más részei is lehetnek, de ezeket nem ismerjük.)
2.c, Hogyan biztosítottuk a fenti rendezés stabilitását?
2.d, Mekkora lesz a rendezés minimális és maximális műveletigénye? Miért?

3.a, Mikor nevezünk egy rendezést stabilnak?
3.b, Adja meg fejelemes, kétirányú, ciklikus listára (C2L) a beszúró rendezés struktogramját! A listát a *key* mezők szerint monoton növekvően kell rendezni. (A listaelemeknek a szokásos *key* és a két mutató mezőn kívül járulékos mezői is lehetnek, de ezeket nem ismerjük. A listát kizárólag az unlink(q), precede(q, r) és follow(p, q) eljárásokkal szabad módosítani, ahogy azt az előadáson tanultuk.)
3.c, Hogyan biztosítottuk a fenti rendezés stabilitását?
3.d, Mekkora lesz a rendezés minimális és maximális műveletigénye? Miért?

2.2. Összefésülő rendezés

1.a, Szemléltesse az összefésülő rendezés (mergesort) működését az előadásról

ismert módon az $\langle 4; 3; 5; 2; 1; 8; 3\rangle$ sorozatra! (Sem a szétvágásokat, sem az összefuttatásokat nem kell részletezni.)
1.b, Adja meg egyszerű láncolt listákra a rekurzív eljárás és a „cut” függvény struktogramját!
1.c, Mekkora a rendezés műveletigénye? Röviden indokolja állítását! (Csak a bizonyítás vázlatát kell leírni.)

2.a, Adja meg vektorokra a **mergeSort(**A**)** és segédeljárásai struktogramjait!
2.b, Mekkora lesz a műveletigénye és a tárigénye? Miért? (Csak vázlatos indoklást kérünk.)

3.a, Szemléltesse az összefésülő rendezés (merge sort) működését az előadásról ismert módon az $\langle 5; 3; 2; 1; 4; 9; 2\rangle$ sorozatra!
3.b, Adja meg egyszerű láncolt listákra az összefésülő rendezés (merge sort) merge($L1, L2$) függvényének struktogramját!
3.c, Mekkora a merge($L1, L2$) függvény műveletigénye? Miért?

4.a, Szemléltesse az összefésülő rendezés (merge sort) működését az előadásról ismert módon az $\langle 1; 9; 2; 5; 3; 2; 4\rangle$ sorozatra!
(Sem a szétvágásokat, sem az összefuttatásokat nem kell részletezni.)
4.b, Adja meg egyszerű láncolt listákra az összefésülő rendezés (merge sort) főeljárásának és rekurzív eljárásának struktogramját!
4.c, Mekkora a rendezés műveletigénye? Tárigénye?

5.a, Szemléltesse az összefésülő rendezés (mergesort) működését az előadásról ismert módon a $\langle 4; 5; 2; 3; 5; 7; 3; 9; 4\rangle$ sorozatra!
(Az utolsó összefésülésnél azt is jelezze, hogy az input elemei milyen sorrendben kerülnek az outputra!)
5.b, Egyszerű láncolt listákra a fenti rendezést úgy szeretnénk módosítani, hogy az azonos kulcsú elemekből csak egy-egy legyen az eredmény listán, így az szigorúan monoton növekvő legyen. A mergesort melyik eljárását/függvényét kell módosítanunk? Adja meg ennek az új struktogramját!
5.c, Mit tud mondani az újfajta összefésülő rendezés műveletigényéről és tárigényéről?

6.a, Szemléltesse az összefésülő rendezést az előadásról ismert módon a $\langle 7; 5; 1; 3; 8; 6; 3; 9; 4\rangle$ sorozatra! (Az utolsó összefésülésnél azt is jelezze, hogy az input elemei milyen sorrendben kerülnek az outputra!)
6.b, Adja meg egyszerű láncolt listákra (S1L) az összefésülő rendezésből (merge sort) a rekurzív eljárás struktogramját!
6.c, Tegyük fel, hogy a rendezendő lista hossza n. Hányszor fog meghívódni a rendezés során a rekurzív eljárás? Miért?

2.3. Kupacrendezés

1.a, Egy bináris fa mikor *szigorúan bináris*? Mikor *teljes*? Mikor *majdnem teljes*? Ez utóbbi mikor *balra tömörített*, és mikor *kupac*?
1.b, Szemléltesse a kupacrendezést a következő tömbre! – $\langle 3; 9; 8; 2; 4; 6; 7; 5\rangle$ – Minden lesüllyesztés előtt jelölje 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 jelölje a lesüllyesztés irányát és útvonalát! Minden olyan lesüllyesztés előtt rajzolja újra a fát, ami az aktuális ábrán már módosított csúcsokat érinti! Újrarajzoláskor adja meg a fát tartalmazó tömb pillanatnyi állapotát is!

2.a, Egy bináris fa mikor *szigorúan bináris*? Mikor *teljes*? Mikor *majdnem teljes*? Ez utóbbi mikor *balra tömörített*, és mikor *kupac*?
2.b, Szemléltesse a kupacrendezést a következő tömbre! – $\langle 5; 7; 6; 4; 2; 8; 9; 4; 3\rangle$ – Minden lesüllyesztés előtt jelölje 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 jelölje a lesüllyesztés irányát és útvonalát! Minden olyan lesüllyesztés előtt rajzolja újra a fát, ami az aktuális ábrán már módosított csúcsokat érinti! Újrarajzoláskor adja meg a fát tartalmazó tömb pillanatnyi állapotát is!

3.a, Adja meg a **heapSort**($A : \mathcal{T}[]$) és segédeljárásai struktogramjait!
3.b, Igaz-e, hogy $MT(n) \in \Theta(n \lg n)$? Miért? [Vegyük észre, hogy az indokláshoz elegendő, ha a kupaccá alakítás és az utána következő rész műveletigényére is durva felső becslést adunk, továbbá használjuk az összehasonlító rendezések (alsókorlát-elemzésre vonatkozó) alaptételét!]

2.4. Gyorsrendezés

1.a, Írja le az előadásról ismert formában a gyorsrendezés (quicksort) struktogramjait!
1.b, Szemléltesse a program „partition” függvényének működését a következő vektorra! $\langle 3; 4; 8; 7; 1; 2; 6; 4\rangle$.
1.c, Mekkora a gyorsrendezés műveletigénye?
1.d, Érdemes-e a gyorsrendezést és a beszúró rendezést egyetlen rendezésben egyesíteni? Hogyan? Miért?

2.a, Írjuk le az előadásról ismert formában a gyorsrendezés (quicksort) struktogramjait!

2.b, Szemléltessük a program „partition” függvényének működését a következő vektorra! $\langle 1; 2; 8; 7; 3; 2; 6; 3\rangle$.
2.c, Mekkora a gyorsrendezés műveletigénye?
2.d, Mekkora munkatárat igényel a gyorsrendezés (quicksort) alapváltozata a legjobb és a legrosszabb esetben? Miért?

3. Tervezze újra a quicksort eljárást egyszerű láncolt listákra! Adja meg a szükséges struktogramokat, ha a szétvágásnál a rendezendő lista első eleme a tengely (pivot)! Vegye figyelembe, hogy a listaelemekben – a *key* mezőn és a következő listaelemre mutató mezőn kívül – lehetnek számunkra ismeretlen, járulékos mezők is!

3. Absztrakt adattípusok

3.1. Verem

1. Adja meg az előadásról ismert módon a **Stack** osztály – tömbös reprezentációra alapozott – leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje?

2. A **TransparentStack** adattípus műveletei a szokásos jelentéssel és formában a **push, pop, isEmpty** verem műveletek, de a v.push(x) csak akkor teszi x-et v-be, ha éppen nincs benne (ha benne van, nem csinál semmit).

A **TransparentStack** elemtípusa az $1..n$ egész intervallum típus, és így legfeljebb n méretű vermekre van szükségünk, ahol n pozitív egész szám.

Adja meg a fenti típust megvalósító osztályt, a metódusokkal együtt, ahol n a konstruktor paramétere!

A konstruktor műveletigénye $\Theta(n)$, a többi metódus és a destruktor műveletigénye $\Theta(1)$ legyen!
{Használhat egy $b[1..n]$ logikai tömböt „$b[x] = true \iff x$ a veremben van” jelentéssel, ahol $x \in 1..n$.}

3. Adja meg az előadásról ismert **Stack** osztály – egyszerű láncolt listás reprezentációra alapozott – leírását, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! Mekkora az egyes műveletek aszimptotikus futási ideje?

3.2. Sor

1. Adja meg az előadásról ismert módon a **Queue** osztály – tömbös reprezentációra alapozott – leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje?

2. Adja meg az előadásról ismert **Queue** osztály – egyirányú, nem ciklikus, láncolt listás reprezentációra (S1L) alapozott – leírását, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! (Szüksége lesz arra, hogy a lista elejét és végét is közvetlenül elérje.) Mekkora az egyes műveletek aszimptotikus futási ideje?

3. Adja meg az előadásról ismert **Queue** osztály – egyirányú, ciklikus, fejelemes láncolt listás reprezentációra alapozott – leírását, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! A listát csak a fejelemre mutató pointeren keresztül szabad elérni. Mekkora az egyes műveletek aszimptotikus futási ideje?

3.3. Prioritásos sor

1. Tegyük fel, hogy a prioritásos sort tömbben, maximum kupaccal reprezentáljuk! Adja meg az előadásról ismert módon a **PrQueue** osztály leírását, a metódusok struktogramjaival együtt! (A lesüllyesztést nem kell leírni.) Mekkora az egyes műveletek aszimptotikus futási ideje?

2.a, Egy bináris fa mikor *szigorúan bináris*? Mikor *teljes*? Mikor *majdnem teljes*? Ez utóbbi mikor *balra tömörített*, és mikor *kupac*?
2.b, Szemléltesse az alábbi kupacra a 9, majd az **eredmény kupacra** a 8 beszúrásának műveletét! $\langle 8; 8; 6; 6; 5; 2; 3; 1; 5; 4 \rangle$.
2.c, Szemléltesse az **eredeti kupacra** a maxKivesz eljárás kétszeri végrehajtását! Minden művelet után rajzolja újra a fát!

3. Tegyük fel, hogy a prioritásos sort tömbben, rendezetlen módon reprezentáljuk! Adja meg az előadásról ismert módon a **PrQueue** osztály leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje, és miért?

4. Tegyük fel, hogy a prioritásos sort tömbben, monoton növekvően rendezett sorozatként tároljuk! Adja meg az előadásról ismert módon a **PrQueue** osztály leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje, és miért?

5. Tegyük fel, hogy a prioritásos sort fejelemes láncolt listával, rendezetlen módon reprezentáljuk! Adja meg az előadásról ismert **PrQueue** osztály leírását erre az esetre, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! Mekkora az egyes műveletek aszimptotikus futási ideje, és miért?

6. Tegyük fel, hogy a prioritásos sort fejelemes láncolt listával, monoton csökkenően rendezett sorozatként tároljuk! Adja meg az előadásról ismert **PrQueue** osztály leírását erre az esetre, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! Mekkora az egyes műveletek aszimptotikus futási ideje, és miért?

4. Láncolt listák

4.1. Egyszerű listák (S1L)

1. Az L_1 és L_2 pointerek két egyszerű láncolt listát azonosítanak. **Írja meg** az append(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!

2. Írja meg a reverse(L) eljárást, ami megfordítja az L egyszerű láncolt lista elemeinek sorrendjét! $T(n) \in \Theta(n)$, ahol n a lista hossza.

4.2. Fejelemes listák (H1L)

1. Az L_1, L_2 pointerek egy-egy szigorúan monoton növekvő H1L (fejelemes, egyirányú, nem ciklikus, láncolt lista) fejelemére mutatnak.

Írjuk meg az intersection(L_1, L_2) eljárást, ami az L_1 lista elemei közül törli azokat az elemeket, amelyek kulcsa nem szerepel az L_2 listán! Így az L_1 listán a két input lista metszete jön létre, míg az L_2 lista változatlan marad. Mindkét listán legfeljebb egyszer menjünk végig! Listaelemeket ne hozzunk létre!

$MT(n_1, n_2) \in O(n_1 + n_2)$ és $mT(n_1, n_2) \in O(n_1)$ legyen, ahol n_1 az L_1, n_2 az L_2 lista hossza.

2. Az L pointer egy nemüres, fejelemes láncolt lista fejelemére mutat.

Írjuk meg a **minElejére**(L) eljárást, ami az L lista legkisebb kulcsú elemét *átfűzi* az L lista elejére! A program a listán csak egyszer menjen végig! $T(n) \in \Theta(n)$, ahol n az L lista hossza. A listaelemeknek a key és a $next$ mezőkön kívül más részei is lehetnek, de ezeket nem ismerjük.

4.3. Egyszerű kétirányú listák (S2L)

1. Az L pointer egy rendezetlen, kétirányú, fejelem nélküli, nem ciklikus, láncolt listát azonosít.

Írjuk meg a del(L, k) eljárást, ami az L listából törli a k kulcsú listaelemeket! A listán legfeljebb egyszer menjünk végig! Listaelemet ne hozzunk létre, a feleslegessé váló listaelemeket viszont deallokáljuk!

Mekkora a fenti eljárás műveletigénye? Miért?

2. Az L pointer egy monoton növekvő kétirányú, fejelem nélküli, nem ciklikus, láncolt listát azonosít.

Írjuk meg a sortedInsert(L, k) eljárást, ami az L listába rendezetten beszúr egy k kulcsú listaelemet! A listán legfeljebb egyszer menjünk végig! Pontosan egy listaelemet hozzunk létre!

$MT(n) \in \Theta(n)$ és $mT(n) \in O(1)$ legyen, ahol n az L lista hossza.

4.4. Fejelemes, kétirányú, ciklikus listák (C2L)

1. Az L_1, L_2 pointerek egy-egy szigorúan monoton növekvő C2L (fejelemes, kétirányú, ciklikus, láncolt lista) fejelemére mutatnak. A listákat kizárólag az unlink(q), precede(q, r) és follow(p, q) eljárásokkal szabad módosítani, ahogy azt az előadáson tanultuk.

Írjuk meg a difference(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!

$MT(n_1, n_2) \in O(n_1 + n_2)$ és $mT(n_1, n_2) \in O(min(n_1, n_2))$ legyen, ahol n_1 az L_1, n_2 az L_2 lista hossza.

2. Az L_1, L_2 pointerek egy-egy szigorúan monoton növekvő C2L (fejelemes, kétirányú, ciklikus, láncolt lista) fejelemére mutatnak. A listákat kizárólag az unlink(q), precede(q, r) és follow(p, q) eljárásokkal szabad módosítani, ahogy azt az előadáson tanultuk.

Írjuk meg a unionIntersection(L_1, L_2) eljárást, ami az L_1 lista elemei közé átfűzi az L_2 listáról az L_1 listán eredetileg nem szereplő elemeket! Így az L_1 listán a két input lista uniója, míg az L_2 listán a metszetük jön létre, és mindkét lista szigorúan monoton növekvő marad. Mindkét listán legfeljebb egyszer menjünk végig! Listaelemeket ne hozzunk létre és ne is töröljünk!

$MT(n_1, n_2) \in O(n_1 + n_2)$ és $mT(n_1, n_2) \in O(n_2)$ legyen, ahol n_1 az L_1, n_2 az L_2 lista hossza.

5. Bináris fák

1.a A t : Node* típusú pointer egy láncoltan ábrázolt bináris fát azonosít. A fa csúcsaiban nincsenek „parent” pointerek. **Írjuk meg** a töröl(t) rekurzív eljárást, ami törli a t fa csúcsait, $\Theta(|t|)$ műveletigénnyel, a posztorder bejárás szerint! Rendelkezésre áll ehhez a **delete** p utasítás, ami a p pointer által mutatott csúcsot törli. A t fa végül legyen üres!

1.b A t_1, t_2 : Node* típusú pointerek egy-egy láncoltan ábrázolt bináris fát azonosítanak. A fák csúcsaiban nincsenek „parent" pointerek. A t_2 fa akkor fedi le a t_1 fát, ha t_1 üres, vagy ha egyikük sem üres, és a t_1 bal és jobb részfáját is lefedik a t_2 megfelelő oldali részfái. Egy fa megmetszése alatt bizonyos részfái törlését értjük. **Írjuk meg** az alámetsz(t_1, t_2) rekurzív eljárást, ami úgy metszi meg a t_1 fát, hogy a t_2 fa lefedje, de a t_1 fa a lehető legnagyobb maradjon! A t_1 feleslegessé váló részfáit töröljük a töröl(t) eljárás segítségével! $T(n) \in \Theta(n)$ legyen, ahol n a t_1 fa mérete.

2. A t : Node* típusú pointer egy láncoltan ábrázolt bináris fát azonosít, ami majdnem teljes és balra tömörített. A fa csúcsaiban nincsenek „parent" pointerek. **Írja meg** az szkiír(t, A, n) eljárást, ami $\Theta(|t|)$ műveletigénnyel a fa kulcsait szintfolytonosan az A tömbbe írja, és n-ben visszaadja a fa méretét! Feltehető, hogy a tömb elég nagy. Az eljárás a fát ne változtassa meg!

3. A t : Node* típusú pointer egy láncoltan ábrázolt bináris fát azonosít. A fa csúcsaiban nincsenek „parent" pointerek. **Írja meg** az szkiír(t) eljárást, ami $\Theta(|t|)$ műveletigénnyel szintenként írja ki a fa kulcsait úgy, hogy minden szint külön sorba kerül, azaz az első sorban csak a fa gyökércsúcsának kulcsa jelenik meg, a második sorban a gyerekei kulcsai, a harmadikban az unokáié stb. Mindegyik szintet balról jobbra írjuk ki! Az eljárás a fát ne változtassa meg!

4. Milyen bináris fa bejárásokat ismer?
4.a, Adja meg a struktogramjaikat!
4.b, Számolja ki a struktogramokhoz tartozó műveletigényeket!

5.1. Bináris keresőfák (és rendezőfák)

1.a, A bináris fa fogalmát ismertnek feltételezve, mondjuk ki a bináris keresőfa definícióját!
1.b, A t egy láncoltan ábrázolt bináris keresőfa gyökérpointere. A csúcsokban nincsenek „parent" pointerek. (A fa üres is lehet.) **Írja meg** az előadásról ismert módon az insert(t, k) eljárás rekurzív struktogramját, ami megkeresi a t fában a k kulcs helyét, és ha ott egy üres részfát talál, akkor az üres részfa helyére tesz egy új levélcsúcsot, k kulccsal.
1.c, Mekkora az insert(t, k) eljárás maximális, illetve minimális műveletigénye? Miért?

2.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!
2.b, Írja meg az insert(t, k, s) – ciklust **nem** tartalmazó – $MT(h) \in \Theta(h)$ hatékonyságú rekurzív eljárást ($h = h(t)$), ami megpróbál beszúrni a t bináris keresőfába egy k kulcsú csúcsot (akkor tudja beszúrni, ha a fában nem talál ilyet), és az s, logikai típusú paraméterben visszaadja, hogy sikeres volt-e a beszúrás! A fa csúcsai **Node** típusúak; szülő pointert nem tartalmaznak.
2.c, Igaz-e, hogy a fenti insert(t, k, s) eljárásra $mT(h) \in \Theta(1)$? Miért?

3.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!
3.b, Szemléltesse a remMin$(t, minp)$ eljárás hatását az alábbi bináris keresőfán!
{ [(1 {2}) 3 (4)] 5 [(6) 7 (8)] }
3.c, Írja meg a remMin$(t, minp)$ és a remMax$(t, maxp)$, $MT(h(t)) \in O(h(t))$ hatékonyságú rekurzív eljárásokat, ahol $t \neq \oslash$ bináris keresőfa! (A remMax$(t, maxp)$ eljáráshívás a $maxp$ pointert a t fa a legnagyobb kulcsú csúcsára állítja, és ezt a csúcsot el is távolítja a fából, ami továbbra is bináris keresőfa marad.) A fa csúcsai „parent” pointert nem tartalmaznak, számunkra ismeretlen, járulékos adatmezőket viszont tartalmazhatnak.
3.d, Igaz-e a fenti remMin$(t, minp)$ és remMax$(t, maxp)$ eljárásokra, hogy $mT(h(t)) \in \Theta(1)$? Miért?

4.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!
4.b, Szemléltesse a delRoot(t) eljárás hatását az alábbi bináris keresőfán! (Törléskor, indeterminisztikus esetben a jobb részfa minimumát használjuk!)
{ [(1) 2 (3)] 4 [(5 {7}) 8 (9)] }
A remMin$(t, minp)$ eljárás struktogramját nem kell megadni.
4.c, Írja meg a delRoot(t), $MT(h(t)) \in O(h(t))$ hatékonyságú eljárást (ami a nemüres t bináris keresőfából törli a gyökércsúcsot)! A fa csúcsai „parent” pointert nem tartalmaznak, számunkra ismeretlen, járulékos adatmezőket viszont tartalmazhatnak.
4.d, Igaz-e a fenti delRoot(t) eljárásra, hogy $mT(h(t)) \in \Theta(1)$? Miért?

5.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!
5.b, Szemléltesse a delRoot(t) eljárás hatását az alábbi bináris keresőfán! (Törléskor, indeterminisztikus esetben a jobb részfa minimumát használjuk!)
{ [(2) 3] 4 [(5) 6 ({7} 8)] }
A delRoot(t) eljárás struktogramját nem kell megadni.

5.c, **Írja meg** a del(t, k), $MT(h(t)) \in O(h(t))$ hatékonyságú rekurzív logikai függvényt, ami megpróbálja törölni a láncoltan ábrázolt t bináris keresőfából a k kulcsú csúcsot! A del(t, k) függvény adja vissza, hogy sikeres volt-e a törlés! A fa csúcsai „parent” pointert nem tartalmaznak, számunkra ismeretlen, járulékos adatmezőket viszont tartalmazhatnak.
5.d, Igaz-e a fenti del(t, k) függvényre, hogy $mT(h(t)) \in \Theta(1)$? Miért?

6.a, A bináris fa fogalmát ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!
6.b, Adott a t bináris fa. A csúcsok kulcsai pozitív egész számok. Írja meg a bst(t) logikai függvényt; ami a t egyszeri (Inorder) bejárásával eldönti, hogy keresőfa-e! $MT(n) \in O(n)$, ahol $n = |t|$. $MS(h) \in O(h)$, ahol $h = h(t)$; MS pedig az algoritmus maximális munkatár-igényét jelöli. (A bejárást és eldöntést a megfelelően inicializált, rekurzív, bst(t, k) logikai segédfüggvény végezze, ami híváskor k-ban a t kulcsainál kisebb értéket vár, visszatéréskor pedig, amennyiben t nemüres keresőfa, a t-beli legnagyobb kulcsot tartalmazza! Ha t üres, akkor k-ban maradjon a függvényhívásnál kapott érték!)
6.c, Igaz-e az Ön által megfogalmazott bst(t) logikai függvényre, hogy $mT(h) \in O(h)$? Miért?

7.a, Bizonyítsa be, hogy tetszőleges, n csúcsú és h magasságú bináris fára az $n - 1 \geq h$ egyenlőtlenség teljesül!
7.b, Mikor lesz $h = n - 1$, és miért?
7.c, Bizonyítsa be, hogy $h \geq \lfloor \lg n \rfloor$, ha a bináris fa nemüres!
7.d, Bizonyítsa be, hogy $h = \lfloor \lg n \rfloor$, ha az előbbi fa *majdnem teljes*!
7.e, Lehetséges-e, hogy $h = \lfloor \lg n \rfloor$, holott a nemüres bináris fára a *majdnem teljesség* kritériuma nem teljesül? Miért?

8.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!
8.b, Milyen kapcsolat van a bináris keresőfák és az inorder bejárás között? (Indokolja is az állítást!)
8.c, A t : Node* típusú pointer egy láncoltan ábrázolt bináris keresőfát azonosít. **Írja meg** a printIncreasingly(t) főeljárást és a pI$(t, level)$ rekurzív segédeljárást, amiknek a segítségével – $\Theta(|t|)$ műveletigénnyel – ki tudja írni a fa kulcsainak szigorúan monoton növekvő sorozatát, és mindegyik kulcs mellett megjeleníti azt is, hogy az a fa hányadik szintjén található, mégpedig „(kulcs/szint)” alakban!

Pl. a { [2 (3)] 4 [(5) 6 ({7} 8)] } fára az eredmény:
(2/1) (3/2) (4/0) (5/2) (6/1) (7/3) (8/2)

Az eljárások a fát ne változtassák meg, és ne tartalmazzanak ciklust!

9.a, A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!
9.b, A t:Node* egy láncoltan ábrázolt bináris keresőfát azonosít. **Írja meg** az előadásról ismert módon az insert(t, k) eljárás rekurzív struktogramját!
9.c, Mekkora aszimptotikusan az $mT(n)$ és az $MT(n)$? Miért?

10.a, Mondja ki a bináris rendezőfa definícióját!
10.b, A t:Node* egy láncoltan ábrázolt bináris rendezőfát azonosít. **Írja meg** az insert(t, k) eljárás rekurzív, ciklust nem tartalmazó struktogramját, $MT(h) \in \Theta(h)$ műveletigénnyel!
10.c, Hogyan tudna a fenti eljárásra stabil rendezést építeni?

6. Rendezés lineáris időben

6.1. Edényrendezés a [0;1) intervallumon (bucket sort)

1.a, A $\langle$0,4; 0,82; 0,0; 0,53; 0,73; 0,023; 0,64; 0,7; 0,39; 0,203$\rangle$, tíz elemű listán mutassa be a *bucket sort* algoritmusát $[0; 1)$-beli kulcsokra!
1.b, Adja meg a bucket sort struktogramját!
1.c, Mekkora a minimális műveletigénye? Mekkora az átlagos műveletigénye, és milyen feltétellel? Hogyan tudnánk biztosítani, hogy a maximális műveletigénye $\Theta(n \lg n)$ legyen?

2.a, A $\langle$0,42; 0,16; 0,64; 0,39; 0,20; 0,89; 0,13; 0,79; 0,53; 0,71$\rangle$ listán mutassa be és magyarázza el az *bucket sort* algoritmusát $[0; 1)$-beli kulcsokra!
2.b, Mekkora a (szokásos, beszúró rendezéses változatának) minimális és maximális műveletigénye? Miért? Mekkora az átlagos műveletigénye?
2.c, Hogyan tudná a maximális műveletigényt aszimptotikusan csökkenteni?
2.d, Mit értünk *stabil rendezés* alatt? Hogyan tudná a *bucket sort*-ot stabil rendezéssé alakítani?

3. Adott az L egyszerű láncolt lista, aminek $n \geq 0$ eleme van. Minden elemének kulcsa a $[0; 1)$ intervalumon egyenletes eloszlás szerint választott érték. Írja meg a **bucketSort**(L, n) utasítással meghívható egyszerű edényrendezés struktogramját, $AT(n) \in \Theta(n)$, $MT(n) \in \Theta(n \lg n)$ műveletigénnyel és $S(n) \in O(n)$ tárigénnyel! Segédrendezésként felhasználható a megfelelő, ebben a félévben tanult, egyszerű láncolt listákat kulcsösszehasonlításokkal rendező eljárás. Ezt nem kell megírni, a kód többi részét viszont teljes részletességgel kérjük.

6.2. Leszámláló rendezés (counting sort)

1.a Adja meg a leszámláló rendezés előfeltételeit, struktogramját és aszimptotikus műveletigényét!
1.b, Szemléltesse a $\langle 30; 20; 11; 22; 23; 13\rangle$ négyes számrendszerbeli számok tömbjén, ha a kulcsfüggvény a baloldali számjegyet választja ki!
1.c, Minek kellett teljesülnie a bemenetre, és minek a rendezésre, hogy a fenti példában a végeredmény, mint számsor is rendezett lett? Hogyan biztosítottuk a rendezés e tulajdonságát?

6.3. Radix rendezés leszámláló rendezéssel

1.a, Mutassa be a *számjegypozíciós* (*Radix*) *rendezés* működését a következő, négyes számrendszerbeli számok tömbjén: $\langle 20; 02; 21; 01; 31; 20\rangle$! Az egyes menetekben *leszámláló rendezést* alkalmazzon!
1.b, Mekkora a fenti rendezés aszimptotikus műveletigénye, és miért?
1.c, A *leszámláló rendezés* – mint segédprogram – mely tulajdonságára épül a *Radix rendezés*? Mit jelent ez a tulajdonság az adott esetben?

2.a, Mutassa be a számjegypozíciós („Radix") rendezés működését a $\langle 11; 20; 10; 23; 21; 30\rangle$ négyes számrendszerbeli számok tömbjén! Az egyes menetekben *leszámláló rendezést* alkalmazzon!
2.b, Mekkora a Radix rendezés műveletigénye, és miért?
2.c, A leszámláló rendezés – mint segédprogram – mely tulajdonságára épül a Radix rendezés? Mit jelent ez a tulajdonság az adott esetben?

6.4. Radix rendezés szétválogatással

1.a, Mutassa be a számjegypozíciós („Radix") rendezés működését a $\langle 31; 20; 11; 23; 21; 10\rangle$ négyes számrendszerbeli számok listáján! Az egyes menetekben a megfelelő számjegy szerinti szétválogató rendezést alkalmazzon!
1.b, Mekkora a Radix rendezés műveletigénye, és miért?
1.c, A felhasznált rendezés – mint segédprogram – mely tulajdonságára épül a Radix rendezés? Mit jelent ez a tulajdonság az adott esetben?

2. Oldja meg az előző feladatot a $\langle 11; 20; 10; 23; 21; 30\rangle$ input listával!

7. Hasító táblák

7.1. Ütközés feloldása láncolással

1. A $T[0..(m-1)]$ hasító tábla rései kétirányú, nemciklikus, fejelem nélküli, rendezetlen láncolt listák pointerei. Adott a $k \bmod m$ hasító függvény. A kulcsütközéseket láncolással oldjuk fel. Mindegyik kulcs csak egyszer szerepelhet T-ben.

1.a, Írja meg az ins(T, k, a):0..2 értékű függvényt, ami beszúrja a hasító táblába a (k, a) kulcs-adat párt! Ha a táblában már volt k kulcsú elem, a beszúrás meghiúsul, és a 2 hibakódot adja vissza. Különben, ha nem tudja már a szükséges listaelemet allokálni, az 1 hibakódot adja vissza. (Feltesszük, hogy a **new** művelet, ha sikertelen, akkor $\oslash$ pointert ad vissza.) Az ins() művelet akkor ad vissza 0 kódot, ha sikeres volt a beszúrás. Ilyenkor az új listaelemet a megfelelő lista elejére szúrja be.

1.b, Mi a kitöltöttségi hányados? Milyen aszimptotikus becslést tud adni a fenti művelet minimális, átlagos és maximális futási idejére? Miért?

2. A $T[0..(m-1)]$ hasító tábla rései kétirányú, nemciklikus, fejelem nélküli, rendezetlen láncolt listák pointerei. Adott a $k \bmod m$ hasító függvény. A kulcsütközéseket láncolással oldjuk fel. Mindegyik kulcs csak egyszer szerepelhet T-ben.

(2.a) Írja meg a *search*(T, k) függvényt, ami visszaadja a T-beli, k kulcsú listaelem címét, vagy a $\oslash$ pointert, ha ilyen nincs!

(2.b) Írja meg a *del*(T, p) eljárást, ami törli a T hasító táblából (és deallokálja is) a p pointer által mutatott listaelemet!

(2.c) Mi a kitöltöttségi hányados? Milyen aszimptotikus becslést tudunk adni a fenti műveletek minimális, átlagos és maximális futási idejére? Miért?

7.2. Nyílt címzés

1. A $T[0..(m-1)]$ hasító táblában a kulcsütközést nyílt címzéssel oldjuk fel.

1.a, Mit értünk *kitöltöttségi hányados*, *próbasorozat* és *egyenletes hasítás* alatt? Mit tudunk a keresés és a beszúrás során a próbák várható számáról?

1.b, Mekkora egy sikertelen keresés várható hossza 80%-os kitöltöttség esetén, ha nincs törölt rés? Egy sikeres keresésé ennél több vagy kevesebb? Miért?

1.c, Legyen most $m = 11$, $h_1(k) = k \bmod m$, és alkalmazzon lineáris próbát! Az alábbi műveletekre szemléltesse a hasító tábla változásait az előadásról ismert módon! Szúrja be a táblába sorban a következő kulcsokat: 10; 22; 31; 4; 15; 28; 16; 26; 62; ezután törölje a 16-ot, majd próbálja megkeresni a 27-et és a 62-t, végül pedig szúrja be a 27-et! Szemléltesse a hasító tábla változásait az előadásról ismert módon!

2. A $T[0..(m-1)]$ hasító táblában a kulcsütközést nyílt címzéssel oldjuk fel.

2.a, Mit értünk *kitöltöttségi hányados* és *egyenletes hasítás* alatt? Mit tudunk a keresés és a beszúrás során a próbák várható számáról?

2.b, Mekkora egy sikertelen keresés várható hossza 90%-os kitöltöttség esetén, ha nincs törölt rés? Egy sikeres keresésé ennél több vagy kevesebb? Miért?
2.c, Legyen most $m = 7$. Nyílt címzést és kettős hasítást alkalmazunk a $h_1(k) = k \bmod m$, $\quad h_2(k) = 1 + (k \bmod (m-2))$ hasító függvények segítségével. Az alábbi műveletek mindegyikére adja meg a *próbasorozatát* $\langle \ldots \rangle$ alakban! Szemléltesse a hasító tábla változásait az előadásról ismert módon! Szúrja be a táblába sorban a következő kulcsokat: 22; 31; 4; 28; 15; 14; 30; ezután törölje a 14-et, majd próbálja megkeresni a 38-at és a 30-at, végül pedig szúrja be a 27-et!

3.a, Adott egy $m = 7$ méretű, üres hasító tábla. Nyílt címzést és kettős hasítást alkalmazunk a $h_1(k) = k \bmod m$, $\quad h_2(k) = 1 + (k \bmod (m-2))$ hasító függvények segítségével. Az alábbi műveletekre szemléltesse a hasító tábla változásait az előadásról ismert módon (i-beszúrás, s-keresés, d-törlés)! i37, i45, i19, i72, i33, d19, s12, i33, d33, i33.
3.b, Magyarázza meg, mi a szerepe nyílt címzés esetén a *foglalt*, az *üres* és a *törölt* réseknek, a beszúrás, a keresés és a törlés során!
3.c, Mi a kitöltöttségi hányados? Milyen becslést tudunk adni az egyes műveletigényekre, és milyen feltétellel?

4. Adott egy $m = 11$ méretű üres hasító tábla. Nyílt címzést és kettős hasítást alkalmazunk a „$h_1(k) = k \bmod m$" és a „$h_2(k) = 1 + (k \bmod (m-1))$" hasító függvények segítségével.
4.a, Az alábbi műveletekre szemléltesse a hasító tábla változásait az előadásról ismert módon! Szúrja be a táblába sorban a következő kulcsokat: 12; 17; 23; 105. Ezután törölje a 23-at, majd próbálja megkeresni a 133-at, végül pedig szúrja be a 133-at!
4.b, Magyarázza meg, mi a különbség nyílt címzés esetén a *foglalt*, az *üres* és a *törölt* rések között, az alkalmazható műveletek szempontjából!