Algoritmusok I. gyakorló feladatok (2024)

Talán van még itt néhány feladat, amit be kellene tenni az ea-jegyzetbe.

A listaelemek az előadásról ismert E1, illetve E2 típusúak. A láncoltan ábrázolt bináris fák csúcsai ugyanígy Node, illetve Node3 típusúak.

Az eljárásokat, függvényeket és metódusokat megfelelően elnevezett és paraméterezett struktogramok segítségével rajzoljuk le! Adjuk meg a paraméterek típusát és paraméterátvételi módot is! 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.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.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 reprezentáció 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 Stack (verem) adattípusban az adatszerkezetet egyszerű láncolt listával (S1L) ábrázoltuk! 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. Tegyük fel, hogy a standard bemeneten adott egy szöveg, amiben a szavakat vesszővel választottuk el, és a szavakon kívül csak a ',' karakter szerepel a szövegben. Írjunk struktogramot, amiben csak karakter és Stack típusú változók lehetnek, és ami a szavakat fordított sorrendben írja ki a standard outputra! (elég lesz két verem)! A beolvasáshoz a get(&c:char):$\mathbb{B}$ függvényt, a kiíráshoz a put(c:char) eljárást használjuk, a szokásos jelentéssel!

Sorok

S.1. Valósítsuk meg a Queue adattípust két verem (Stack) segítségével! Feltehető, hogy a verem műveletei, a konstrutort és a destruktort is beleértve, $\Theta(1)$ futási időt igényelnek. A rem() maximális műveletigénye $\Theta(n)$ lesz, ahol n a sor hossza. A többi műveletnél tartsuk a $\Theta(1)$ futási időt! Lássuk be, hogy az add() és rem() műveletek együttes átlagos műveletigénye így is $\Theta(1)$ marad!

S.2. Tegyük fel, hogy a Queue 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, az add(x) és a reml() 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 remMax() 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 remMax() műveletet erre a kupacra!

E.4. Adott az $A[k..m]$ tömb, amiben szintfolytonosan ábrázoltunk egy teljes (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).
- $lev(i : k..m) : \mathbb{B}$, 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).
- $vanjobb(i : k..m) : \mathbb{B}$, ami akkor ad vissza igaz értéket, ha $A[i]$-nek van jobb gyereke.
- $gy(i : k..m) : \mathbb{B}$, 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; $m = k - 1$ esetén a heap üres.

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 O(a fa magassága) műveletigénnyel hajtsuk végre!

E.6. Az $A[k..n]$ egész számok tömbje, $n \geq m \geq k - 1$, $A[k..m]$ bináris maximumheap.

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 O(a fa magassága) 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. 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 O(fa magassága és k szorzata) 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 O(fa magassága és k szorzata) műveletigénnyel hajtsuk végre!

E.9. Valósítsuk meg rendezetlen tömb segítségével az előadásról ismert PrQueue osztályt! Legyen a remMax() függvény futási ideje lineáris, míg az add(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 PrQueue osztályt! Legyen a remMax() függvény futási ideje lineáris, míg az add(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 C2L 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.2. Lásd még az előadásjegyzet feladatait!

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($A[0..n), t$) eljárást, ami előállítja a t:Node*, méret szerint kiegyensúlyozott bináris fát $MT(n) \in O(n)$ maximális műveletigénnyel, ahol a fa Inorder bejárása sorban az $A[0..n)$ 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!

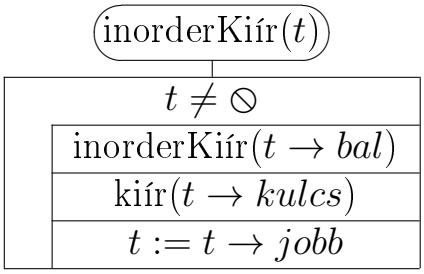
B.2. Írjuk meg az előző programot abban az esetben, ha a fa csúcsaiban „szülő pointerek" is vannak, azaz az előállítandó t fa Csúcs3* típusú.

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!

1. Rendezés lineáris időben

1.1. Rendezzük számjegypozíciós listarendezéssel a 013, 200, 010, 321, 213, 201 négyes számrendszerbeli számok egyszerű láncolt listáját! Mutassuk be a fenti példán az algoritmus működését! Tegyük meg ezt a 31, 21, 30, 22, 01, 23 listára is!

1.2. Rendezzük számjegypozíciós (Radix) rendezéssel a 013, 200, 010, 321, 213, 201 négyes számrendszerbeli számok tömbjét! Az egyes menetekben leszámláló rendezést alkalmazzunk! Mutassuk be a fenti példán az algoritmus működését! Tegyük meg ezt a 31, 21, 30, 22, 01, 23 tömbre is!

1.3. Az `A[1..n]` vektort szeretnénk rendezni. A vektor elemtípusának kulcsfüggvénye $\varphi = (\varphi_k, \ldots \varphi_2, \varphi_1)$ alakú, és az elemeket a fenti, összetett kulcsfüggvény szerint lexikografikusan rendezve, monoton növekvően kell sorbarakni.

Adott a `rendezi(A[1..n],i,B[1..n])` eljárás, amely az `A[1..n]` tömböt a φ_i kulcsfüggvény szerint, rendezéstartó módon, monoton növekvően a `B[1..n]` tömbbe rendezi, ahol $T(n,k) \in O(f(n))$. (Ezt nem kell megírni!)

Írjuk meg a `rendez(A[1..n])` eljárást, amely megoldja az eredeti rendezési feladatot, ahol $T(n,k) \in O(k * f(n))$.

1.4. Írjuk meg a `negPoz(A[1..n])` eljárást, ami az `A[1..n]` vektor elejére rendezi a negatív elemeket, a végére pedig a nemnegatívakat! $T(n) \in O(n)$, $M(n) \in O(1)$.

1.5. Adott az `A[1..n]` tömb, amelynek elemei k bites, nemnegatív egész számok. Írjuk meg a `Bincsere(A[1..n])` eljárást, ami a tömböt monoton növekvően, bináris cserével helyben rendezi, ahol $T(n,k) \in O(k * n)$.

Az alkalmazandó algoritmus: A legmagasabb helyiérték szerint kettéosztjuk a tömb elemeit, majd a résztömböket a második legmagasabb helyiérték szerint osztjuk ketté stb.

(**Ötlet:** Használjuk ehhez a `bcs(A[u..v],m)` eljárást, ami az `A[u..v]` résztömböt a legalacsonyabb helyiértékű `m` bit szerint rendezi.)

2. Hasító táblák

2.1. A `T[0..M-1]` hasító tábla rései (slot-jai) egyirányú, nemciklikus, fejelem nélküli, rendezetlen láncolt listák pointerei. Adott a `hash(kulcs) = kulcs mod M` tördelőfüggvény (hash-függvény). Feltehető, hogy a tábla kulcsok szerint egyértelmű.

2.1.a. Írjuk meg az `ins(T[],M,k,a)` : `0..2` értékű függvényt, ami beszúrja a `T[0..M-1]` hasító táblába a `(k,a)` kulcs-adat párt! Akkor és csak akkor ad vissza `0` értéket, ha sikeres volt a beszúrás. Ilyenkor az új listaelemet a megfelelő lista elejére szúrjuk be! Ha a táblában már volt `k` kulcsú elem, a beszúrás meghiúsul, és `2` hibakódot adjunk vissza. Különben, ha nem tudjuk már a szükséges listaelemet allokálni, `1` hibakódot adjunk vissza! Feltehető, hogy a szükséges memória allokátor művelet sikertelenség esetén $\oslash$ pointert ad vissza.

2.1.b. Írjuk meg a `search(T[],M,k)` függvényt, ami visszaadja a T-beli, `k` kulcsú elem címét, vagy a $\oslash$ pointert, ha ilyen nincs!

2.1.c. Írjuk meg a `del(T[],M,k)` logikai függvényt, ami törli a `T[0..M-1]` hasító táblából a `k` kulcsú adatot! Akkor és csak akkor ad vissza `true` értéket, ha sikeres volt a törlés, azaz a táblában volt `k` kulcsú elem. Ilyenkor a feleslegessé váló listaelemet deallokáljuk!

2.1.d Mutassuk be, mi történik, ha sorban beszúrjuk a hasító táblába a következő kulcsokat: 5; 28; 19; 15; 20; 33; 12; 17; 10. Az egyszerűség kedvéért legyen $M = 9$, és legyen a hasító függvény a következő: $h(k) = k \bmod 9$.

2.2. A `T[0..M-1]` hasító táblát nyílt címzéssel ábrázoltuk. Adott a $h_1(kulcs) = kulcs \bmod M$ tördelőfüggvény (hash-függvény). Feltehető, hogy a tábla kulcsok szerint egyértelmű.

2.2.a. Írjuk meg az `ins(T[],M,k,a)` : `0..2` értékű függvényt, ami beszúrja a `T[0..M-1]` hasító táblába a `(k,a)` kulcs-adat párt! Akkor és csak akkor ad vissza `0` értéket, ha sikeres volt a beszúrás. Ha a táblában már volt `k` kulcsú elem, a beszúrás meghiúsul, és

2 hibakódot adjunk vissza. Különben, ha a táblában már nincs szabad hely, 1 hibakódot adjunk vissza! A kulcsütközést lineáris próbával oldjuk fel! Ne feledkezzünk el az üres és a törölt rések (slot-ok) megkülönböztetéséről!

2.2.b. Írjuk meg a `search(T[],M,k,a)` logikai függvényt, megkeresi a `T[0..M-1]` hasító táblában a `k` kulcshoz tartozó `a` adatot! Akkor és csak akkor ad vissza `true` értéket, ha sikeres volt a keresés, azaz a táblában volt `k` kulcsú elem. (Sikertelen keresés esetén `a` definiálatlan marad.) A kulcsütközést lineáris próbával oldjuk fel! Ne feledkezzünk el az üres és a törölt rések (slot-ok) megkülönböztetéséről!

2.2.c. Írjuk meg a `del(T[],M,k)` logikai függvényt, ami törli a `T[0..M-1]` hasító táblából a `k` kulcsú adatot! Akkor és csak akkor ad vissza `true` értéket, ha sikeres volt a törlés, azaz a táblában volt `k` kulcsú elem. A kulcsütközést lineáris próbával oldjuk fel! Ne feledkezzünk el az üres és a törölt rések (slot-ok) megkülönböztetéséről!

2.3. Oldjuk meg a 2.2 feladatot négyzetes próba esetére! ($c_1 = c_2 = 1/2$)

2.4. Írjuk meg a 2.2 feladat algoritmusait kettős hash-elés esetére! A másodlagos hash függvény: $h_2(kulcs) = 1 + (kulcs \bmod (M - 1))$.

2.5. Mutassuk be a 2.1-2.4 feladatokban szereplő algoritmusok működését papíron, konkrét adatokkal, `M=11` esetén! (Mi lehet a baj a négyzetes próbával?) Vegyünk például egy kezdetben üres hasító táblát, és szúrjuk be egymás után a következő kulcsokat: 10; 22; 31; 4; 15; 28; 17; 88; 59. Ezután töröljük a 17-et, majd próbáljuk megkeresni a 18-at és az 59-et, végül pedig szúrjuk be a 18-at! Nyílt címzés esetén minden egyes művelethez adjuk meg a próbasorozatot is!