Algoritmusok I. / 2. zh minta feladatok (2011)

A listaelemek `struct Elem{ int adat; Elem *mut; }`, a fák csúcsai `struct Elem2{int adat; Elem2 *bal, *jobb;}` (nincs „szülő pointer") vagy
`struct Elem3{int adat; Elem3 *bal, *jobb, *sz;}` (van „szülő pointer") típusúak. Az eljárásokat és függvényeket megfelelően elnevezett és paraméterezett struktogramok segítségével adjuk meg. A változókat alapértelmezésben a struktogramra vonatkozóan lokálisnak tekintjük. Kerüljük a felesleges memória allokálásokat (new)! Ahol a feladat az algoritmus leírását is tartalmazza, ott a megoldás az algoritmus implementációja legyen!

Definíciók:

A t bináris fa egy csúcsa méret szerint kiegyensúlyozott, **a.ha** a két részfájának a mérete legfeljebb eggyel tér el egymástól.

A t bináris fa méret szerint kiegyensúlyozott, **a.ha** minden csúcsa méret szerint kiegyensúlyozott.

1. Adott az `A[0..n-1]` egész számok tömbje. Írjuk meg a `toBinTree(A[],n,t)` eljárást, ami előállítja az `Elem2 *t`, méret szerint kiegyensúlyozott bináris fát $T(n) \in O(n)$ maximális műveletigénnyel, ahol a fa Inorder bejárása sorban az `A[0..n-1]` 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.

2. Írjuk meg az előző programot abban az esetben, ha a fa csúcsaiban „szülő pointerek" is vannak, azaz az előállítandó fa `Elem3 *t` típusú.

3. Az `A[0..m-1]` egész számok tömbje, `k` pozitív egész szám, `m>=n>=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.)

4. Egy nemüres láncolt bináris fa csúcsaiban `sz` szülő pointerek is vannak. A `p` pointer a fa egy csúcsára mutat. Írjuk meg a `inorder_prev(p)` eljárást, ami a `p` pointert az inorder bejárás szerint megelőző elemre állítja. Ha ilyen nincs, a `p` pointer a `null` értéket vegye fel. $T(t) \in O(h(t))$, ahol $h(t)$ a `*p` csúcsot tartalmazó t fa magassága. (A program számára t nem adott.)

5. Az `L` pointer egy egyszerű láncolt lista elejére mutat, `n` a lista hossza. (A lista üres is lehet). Írjuk meg az `mergeSort(L,n)` eljárást, ami a listát monoton növekvően rendezi $T(n) \in O(n*log(n))$ maximális műveletigénnyel és $M(n) \in O(log(n))$ maximális tárigénnyel. **Algoritmus**: Ha a listának legalább két eleme van, akkor középen elfelezzük, az így kapott két listát külön-külön, rekurzív `mergeSort` hívással rendezzük, és az eredményeket rendezetten összefésüljük. A rendezett összefésülés az elemduplikációkat megtartja.

6. 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 a nála 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.

7. 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 csúcsok egyensúlyait is, a szokásos módon!

8. 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.

9. Implementáljuk a Sor ADT üresre inicializáló, a sor végéhez fűző és a sor első elemét kivevő műveleteit, ha a sort egyirányú, láncolt listával ábrázoltuk úgy, hogy a lista első és utolsó elemére is egy-egy pointer mutat. A lista utolsó eleme üres, így az üres sor is tartalmaz egy listaelemet. Mindegyik művelet $O(1)$ műveletigényű legyen!