Algoryty i Struktury Danych 2012

Lista 2

1. Przyjmij następującą definicję węzła drzewa BST:

```
struct node
{   int key;
    node* left;
    nide * right;
    node(int k0,node* l0=NULL, r0=NULL):key(k0),left(l0),right(r0){}
};
```

   Napisz procedury:

   (a) `void insert(node*& t,int x)` – wstawia x do drzewa t.

   (b) `int min(node* t)` – zwraca minimalny klucz w t.

   (c) `node* find (node* t, int x)` – zwraca wskaźnik na wezeł zawierający klucz x lub NULL gdy nie ma takiego węzła.

   (d) `int sum(node* l)` – zwraca sumę kluczy w drzewie t.

   (e) `int inorder(node* t)` – wypisuje klucze drzewa t w porządku `in order`.

   (f) `int inorder(node* t,void(*f)(int))` – wywołuje procedurę `*f` dla wszytkich kluczy drzewa t w porządku `in order`.

   (g) `void remove(node*& t,int x)` – usuwa klucz x z do drzewa t (o ile go znajdzie).

   (h) `void destroy(node* t)` – usuwa wsysztkie klucze z drzewa t i zwalnia całą pamięć.

   (i) `node* copy(node* t)` – zwraca kopię drzewa t. (rekurencja w porządku `post order`).

   Punkty (a)-(d) wykonaj w dwóch wersjach: rekurencyjnej i bez użycia rekurencji.

2. Napisz iterator do powyższej implementacji drzewa binarnego.

3. Zapoznaj się z implementacją węzła klasy deque `http://kicia.ift.uni.wroc.pl/algorytmy/struktury/deque.cc` i napisz iterator do tej listy.

4. Jaka jest minimalna liczba porównań potrzebna, by znaleźć: (a) największy, (b) drugi co do wielkości, (c) k-ty co do wielkości element (początkowo nieposortowanej) tablicy n liczb?

5. Pewiem algorytm dzieli tablicę rozmiaru n na dwie równe części, wykonuje n^2 operacji i wywołuje się rekurencyjnie dla każdej z tych części. Jaka jest złożoność tego algorytmu? Co by było, gdyby wykonywał n zamiast n^2 operacji?
   Wskazówka: do obliczeń przyjmij, że n jest potęgą dwójki.