

Algorytmy i Struktury danych

Lista zadań 2 (tablice, listy, drzewa)

Pomocny przy wykonaniu tej listy jest plik `tree-2018.cc` - (drzewa BST - implementacja) do pobrania z serwisu <http://panoramx.ift.uni.wroc.pl>. Pobierz go i zapoznaj się z zawartością.

1. Ile trzeba porównań, by znaleźć element x w posortowanej tablicy t o rozmiarze n .
2. Rozważ trzy wersje znajdowania maksimum w tablicy `int maks(int t[], int n)`.
 - (a) iteracyjna - `{int x=a[--n]; while(n--){if(t[n]<x)x=t[n];} return x;}`
 - (b) rekurencyjna - znajdź maksimum $n-1$ elementów; porównaj je z ostatnim elementem
 - (c) rekurencyjna - podziel tablicę na dwie części; znajdź ich maksima; wybierz większe z nich.Ile pamięci wymaga każda z tych wersji? Ile porównań między elementami wykonuje?
3. Ile potrzeba mnożeń by obliczyć w standardowy sposób: (a) iloczyn skalarny dwóch wektorów rozmiaru n . (b) Wartość wielomianu stopnia n . (c) Iloczyn dwóch wielomianów stopnia n . (d) Iloczyn dwóch macierzy $n \times n$. (e) Wyznacznik macierzy $n \times n$.
4. (2 pkt.) Jak przesunąć cyklicznie elementy tablicy `int t[n]`; o k komórek w prawo tak by nie wykorzystywać dodatkowej pamięci (za wyjątkiem kilku zmiennych pomocniczych typu `int`): (a) jeśli liczby n i k są względnie pierwsze, (b) dla dowolnych n i k .
5. Napisz procedurę `void reverse(lnode*& L)`, która odwróci listę L modyfikując jedynie pola `next` elementów listy L oraz wskaźnik L .
6. Napisz procedurę `lnode* merge(lnode* L1, lnode* L2)`, która złączy posortowane listy $L1$ i $L2$ w jedną posortowaną listę, używając tylko dwóch dodatkowych wskaźników ($M(n) = \text{const}$). Ilość porównań nie powinna przekroczyć długości wynikowej listy.
7. Jakie drzewo powstanie po wstawieniu do pustego drzewa BST liczb od 1 do n w kolejności rosnącej? Jaka potem będzie głębokość drzewa? Ile porównań kluczy wykonano w trakcie tworzenia tego drzewa? Jaka jest złożoność w tego procesu w notacji O ?
8. Uzasadnij, że w każdym drzewie BST zawsze ponad połowa wskaźników (pól `left` i `right`) jest równa `NULL`.
9. Ile maksymalnie węzłów może mieć drzewo BST o głębokości h ? Jaka jest najmniejsza a jaka największa głębokość drzewa binarnego o n węzłach?
10. Napisz operacje dla drzewa BST (`find`, `insert`, `remove`) bez użycia rekurencji. Jak ich pesymistyczna złożoność czasowa $T(n)$ zależy od głębokości drzewa?
11. Implementacja usuwania węzła z drzewa binarnego działa wg następującego schematu:
 - (a) jeśli usuwany węzeł nie ma dzieci, to go usuwamy a odpowiedni wskaźnik zmieniamy na `NULL`.
 - (b) jeśli ma jedno dziecko, to go usuwamy, a odpowiedni wskaźnik w węźle rodzica zastępujemy wskaźnikiem na to dziecko.
 - (c) jeśli ma dwoje dzieci, to nie usuwamy tego węzła, lecz najmniejszy element w jego prawym poddrzewie, a dane i klucz tego elementu wpisujemy do węzła, który miał być usunięty.

Uzasadnij, dlaczego postępowanie wg punktu (c) nie psuje prawidłowego rosnącego porządku kluczy wypisywanych w porządku `inorder`.

12. Jak zmodyfikować operacje dla drzewa BST (`insert`, `remove`) bez użycia rekurencji, aby działały poprawnie dla drzewa o węzłach gdzie występuje też wskaźnik na ojca.
`struct node{int x; node *left; node *right; node *parent;};`
13. Napisz rekurencyjną procedurę `void inorder_do(node *t,void f(node*))`, która wykona funkcję `f` na każdym węźle drzewa `t` w kolejności `in order`.
14. Wiedząc, że `node` zawiera wskaźniki na rodzica `parent`, napisz nierekurencyjną wersję powyższej funkcji.
15. Napisz funkcję `int height(node *t)`, która wyliczy głębokość (długość najdłuższej gałęzi) drzewa BST. Jak zależy czas wykonania tej funkcji od ilości n węzłów drzewa i jego głębokości h ?
16. (2 pkt.) Zakładając, że w każdym węźle drzewa BST jest również wskaźnik na ojca, napisz klasę `iterator` oraz funkcje `iterator begin(node *t)` oraz `iterator end(node *t)`, które pozwolą wypisać wszystkie klucze z drzewa `t` za pomocą instrukcji:

```
for(iterator begin(t); i!=end(t); i++)  
    cout<< *i <<endl;
```

Jedyną składową (w części prywatnej) powinien być wskaźnik na bieżący węzeł.

17. (3 pkt.) Zmodyfikuj `iterator` drzewa BST, tak by nie korzystał z pola `parent`. Wskazówka: do części prywatnej iteratora dodaj stos elementów typu `node*` zawierający węzły, powyżej bieżącego.