Algorytmy i Struktury danych

Lista zadań 3 (rekurencja i drzewa)

1. W pliku jest $n = 10^6$ liczb całkowitych. Ile potrzeba pamięci i dodawań by sprawdzić, która z sum $k = 1000$ kolejnych liczb jest największa? Czy potrafisz zrobić tak, aby całkowity rozmiar utworzonych zmiennych był mniejszy niż 20-bajtów, niezależnie od wartości n i k?

2. Napisz nierekurencyjną procedurę `int poziom(BSTnode * t, int klucz)`, której wynikiem jest poziom w drzewie `t`, na którym występuje `klucz`. Wynik 0 oznacza brak klucza w drzewie, 1 - klucz w korzeniu, 2 - w dziecku korzenia itd.

3. Jaką informację trzeba dodać do każdego węzła, by w czasie $O(\log n)$ losować z drzewa klucz z prawdopodobieństwem proporcjonalnym do tego klucza. Klucze większe mają mieć większą szansę na wylosowanie.

4. Ile porównań wykona algorytm `insertion_sort` w wersji z wartownikiem, jeśli dane $(a_1, ..., a_n)$ o rozmiarze `n` zawierają `k` inwersji. Liczba inwersji to liczba takich par (i, j), że $i < j$ i $a_i > a_j$.

5. Jaka jest maksymalna liczba porównań, procedury `insertion_sort` w wersji z wartownikiem, dla danych rozmiaru 5.

6. Niech $K(n)$ oznacza ilość różnych kształtów drzew binarnych o n węzłach.
   (a) Znajdź wzór rekurencyjny wyrażający $K(n)$ przez $\{K(i) : i < n\}$.
   (b) Napisz procedurę rekurencyjną, która używa tego wzoru.
   (c) Napisz procedurę nierekurencyjną, która oblicza po kolei wyrazy ciągu $K(n)$ i zapisuje je w tablicy. Przy obliczaniu kolejnych wyrazów, korzysta w poprzednio zapisanych wyników.
   (d) Uruchom programy (b) i (c) dla $n = 1000$ lub większego.

7. Jaką dodatkową informację należy przechowywać w każdym węźle drzewa binarnego, by łatwo znajdować medianę zawartych w nim elementów? Napisz implementację funkcji `BTSnode* ity(BSTnode *t, int i)`, korzystającą z tego dodatkowego pola, która będzie działała w czasie $O(\log n)$ dla drzew zrównoważonych. Jak należy zmienić rekurencyjne wersje procedur `insert` i `remove`, by infornacja ta była automaycznie uaktualniana.

8. Algorytm `quick_sort` nie jest stabilny. Znajdź jak najmniejsze dane, dla których algorytm zmieni względną kolejność jednakowych kluczy.

9. Napisz procedurę `void reverse(node*& n)`, która odwróci listę jednokierunkową, której pierwszy element jest wskazywany przez `n`. Nie może ona wykonywać przydzielać ani zwalniać pamięci. Ma tylko zmienić pola `next` w każym węźle i ustawić `n` na (dotychczasowy) ostatni element.

10. Napisz procedurę `insertion_sort(node*& n))` - sortowanie przez wstawianie działające na liście jednokierunkowej.

11. Napisz procedurę `merge_sort(node*& n))` - sortowanie przez zlaczanie działające na liście jednokierunkowej, nie używaj rekurencji.