Klasówka 1 z ASD
19.XI.2015

Zadanie 1 [5 punktów]

[2 punkty] Zaproponuj algorytm, optymalny ze względu na liczbę porównań, sortujący ciąg x[1], x[2], …, x[7], parami różnych liczb całkowitych oraz taki, że dla każdego i = 1, 2, 3, x[i] < x[2i] oraz x[i] < x[2i+1].

[3 punkty] Udowodnij optymalność swojego rozwiązania.

Zadanie 2 [6 punktów]

Ile wynosi maksymalna liczba zamian w fazie budowy kopca (od dołu) dla 2-uporządkowanego ciągu a[1..n], gdzie $n = 2^k-1$, dla pewnego $k > 0$? Odpowiedź uzasadnij.

Zadanie 3 [9 punktów]

Rozważamy dynamicznie zmieniający się ciąg $A=<a_1, a_2, \ldots, a_n>$, parami różnych n liczb całkowitych. Na ciągu A dozwolona jest jedyna operacja NaPoczątek(i), $1 \leq i \leq n$, która przesuwa element a_i na początek A.

Przykład
Dla $A=<4,1,3,5,2>$, po wykonaniu NaPoczatek(3) dostajemy $A=<3,4,1,5,2>$.

Interesuje nas ciąg operacji NaPoczątek o minimalnej długości, których wykonanie posortuje A. Nazwijmy go *minimalnym ciągiem sortującym.*

a) [3 punkty] Zaprojektuj algorytm, który w czasie O(n) wyznacza pierwszy element minimalnego ciągu sortującego.

b) [2 punkty] Zaprojektuj efektywny algorytm wyznaczający cały minimalny ciąg sortujący.

c) [4 punkty] Udowodnij poprawność swoich rozwiązań.

Dokonaj analizy złożoności czasowej zaproponowanych algorytmów.

Rozwiązania

Zadanie 1

Porządek na x, to porządek kopcowy. x[1] jest najmniejsze. Pozostałe elementy można uporządkować na 2*2*( ) sposobów – lewe i prawe poddrzewa można uporządkować na dwa sposoby każde, a w ciągu wynikowym elementy z lewego poddrzewa można rozmieścić na ( ) sposobów. Razem mamy 80 sposobów. Stąd drzewo decyzyjne sortujące ciągi x musi mieć wysokość co najmniej 7. Sortowanie za pomocą 7 porównań jest proste. Sortujemy osobno poddrzewa, a następnie scalamy dwa ciągi uporządkowane, każdy o długości 3.

Zadanie 2

Udowodnimy, że maksymalna liczba zamian wynosi 2^{k-1}-1. Liczba ta jest realizowana przez ciąg 2-uporządkowany, w którym wszystkie elementy z pozycji nieparzystych są większe od wszystkich elementów z pozycji parzystych. Dowód przeprowadzimy indukcją po k. Dla k = 1, 2 w oczywisty sposób maksymalne liczby zamian wynoszą odpowiednio 0 i 1. Rozważmy k > 2. Wówczas wysokość kopca wynosi k-1. Niech r będzie korzeniem. Z 2-uporządkowania wynika, że klucz z korzenia nigdy nie będzie zamieniany z kluczem z prawego syna. Z założenia indukcyjnego maksymalna liczba zamian w prawym poddrzewie wynosi 2^{k-2}-1. Załóżmy teraz, że klucz z korzenia jest zamieniany z kluczem z lewego syna. Z 2-uporządkowania wynika, że musi to być oryginalny klucz z tego węzła (przed budową lewego podkopca). Zatem b.o. możemy przed wszystkimi zamianami zamienić klucz z korzenia z kluczem z jego lewego syna. Jeśli teraz przenumerujemy węzły w lewym poddrzewie od 1 (odejmując 1 od numeru każdego węzła), to pytamy się o maksymalną liczbę zamian w budowie kopca dla 2-uporzadkowanego ciągu długości 2^{k-1}-1 – z złożenia indukcyjnego jest ich 2^{k-2} – 1. Tak więc razem mamy 2^{k-2} – 1 + 2^{k-2} – 1 + 1 = 2^{k-1}-1 zamian.

Zadanie 3

W tym zadaniu najtrudniejsza jest część a). Wystarczy zauważyć, że element maksymalny nie jest przesuwany. Jeśli drugi co do wielkości jest przed nim, to też nie jest przesuwany. Jeśli trzeci co do wielkości nie jest, to też nie jest przesuwany, itd. Zatem pierwszym przesuwanym elementem jest największy taki, od którego nie zaczyna się podciąg rosnący kolejnych co do wielkości elementów ciągu, kończący się elementem największym. Znalezienie takiego elementu jest bardzo proste. Szukamy maksimum przeglądając ciąg od strony lewej do prawej. Niech maks będzie największą dotychczas znalezioną wartością i m_1, m_2, …, m_k będą wartościami, które przyjmuje maks. Wówczas szukanym element jest element poprzedzający m1 w ciągu posortowanym. Znalezienie takiego elementu zajmuje czas liniowy – zapamiętujemy największy z elementów spośród największych pomiędzy m_i oraz m_{i+1}.

W części drugiej sortujemy cały ciąg i wyznaczamy m_1, m_2, …, m_k. Następnie dla każdego z pozostałych elementów liczymy ile jest elementów większych od niego, położonych na prawo i różnych od m_1, …, m_k. Stosujemy algorytm, jak przy obliczaniu wektora inwersji.

Napoczątek dla elementu e wykonujemy z parametrem równym pozycja elementu w ciągu wejściowym plus policzona liczba elementów większych.