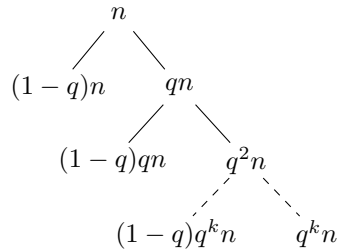


Algorytmy i struktury danych

Lista zadań 5

Zadanie 1

Udowodnij, że jeśli dla pewnego ustalonego q , takiego że $\frac{1}{2} < q < 1$, podczas sortowania szybkiego, procedura `partition`, na każdym poziomie rekurencji podzieli elementy tablicy w stosunku $q : (1 - q)$ to algorytm wykona się w czasie $O(n \log n)$. Wskazówka: udowodnij, że głębokość rekurencji nie przekroczy $-\log n / \log q$ i zaniedbaj błędy zaokrągleń do wartości całkowitych.



$$q^k n = 1$$

$$q^k = \frac{1}{n}$$

$$\log q^k = \log n^{-1}$$

$$k \log q = -\log n$$

$$k = -\frac{\log n}{\log q}$$

Zadanie 2

Ile porównań (zapisz wyniki w notacji O) wykona algorytm `quicksort` z procedurą `partition` w wersji Hoare'a, a ile w wersji z procedurę `partition` w wersji Lomuto dla danych: (a) posortowanych rosnąco, (b) posortowanych malejąco, (c) o identycznych kluczach?

Zadanie 3

Napisz wzór na numer kubelka, do którego należy wrzucić liczbę x w sortowaniu kubelkowym, jeśli kubelków jest n , a elementy tablicy mieszczą się przedziale (a, b) . Numeracja zaczyna się od 0.

$$k = \left\lfloor \frac{n}{b-a} \cdot (x-a) \right\rfloor$$

$$a \leq x < b$$

$$0 \leq x-a < b-a$$

$$0 \leq \frac{x-a}{b-a} < 1$$

$$\left\lfloor \frac{x-a}{b-a} \cdot n \right\rfloor$$

Zadanie 4

Dla jakich danych sortowanie metodą kubełkową ma złożoność $O(n^2)$?

Dla danych rozłożonych bardzo nierównomiernie których większość trafi do jednego kubełka, gdyż wywołany będzie na nich algorytm sortowania przez wstawianie o złożoności $O(n^2)$.

Zadanie 5

Jak obliczyć k -tą od końca cyfrę w liczby x ? Jak obliczyć ilość cyfr liczby x ? Przyjmujemy układ dziesiętny. Jak wyniki zmieniają się w układzie pozycyjnym gdzie różnych cyfr jest m a ich wartości x należą do przedziału $0 \leq x < m^n$?

$$x_k = \left\lfloor \frac{x}{10^k} \right\rfloor \bmod 10$$

$$x_k = \left\lfloor \frac{x}{m^k} \right\rfloor \bmod m$$

$$n = \lceil \log_{10} x \rceil + 1$$

$$n = \lceil \log_m x \rceil + 1$$

Zadanie 6

Posortuj metodą sortowania pozycyjnego liczby: 101, 345, 103, 333, 432, 132, 543, 651, 791, 532, 987, 910, 643, 641, 12, 342, 498, 987, 965, 322, 121, 431, 350. W pisemnym rozwiązaniu pokaż, jak wygląda zawartość kolejek, za każdym razem, gdy tablica wyjściowa jest pusta i wszystkie liczby znajdują się w kolejkach, oraz jak wygląda tablica wyjściowa, za każdym razem, gdy sortowanie ze względu na kolejną cyfrę jest już zakończone.

0 : (910, 350)
1 : (101, 651, 791, 641, 121, 431)
2 : (432, 132, 532, 12, 342, 322)
3 : (103, 333, 543, 643)
4 : ()
5 : (345, 965)
6 : ()
7 : (987, 987)
8 : (498)
9 : ()

(910, 350, 101, 651, 791, 641, 121, 431, 432, 132, 532, 12, 342, 322, 103, 333, 543, 643, 345, 965, 987, 987, 498)

0 : (101, 103)
1 : (910, 12)
2 : (121, 322)
3 : (431, 432, 132, 532, 333)
4 : (641, 342, 543, 643, 345)
5 : (350, 651)
6 : (965)
7 : ()
8 : (987, 987)
9 : (791, 498)

(101, 103, 910, 12, 121, 322, 431, 432, 132, 532, 333, 641, 342, 543, 643, 345, 350, 651, 965, 987, 987, 791, 498)

0 : (12)
1 : (101, 103, 121, 132)
2 : ()
3 : (322, 333, 342, 345, 350)
4 : (431, 432, 498)
5 : (532, 543)
6 : (641, 643, 651)
7 : (791)
8 : ()
9 : (910, 965, 987, 987)

(12, 101, 103, 121, 132, 322, 333, 342, 345, 350, 431, 432, 498, 532, 543, 641, 643, 651, 791, 910, 965, 987, 987)

Zadanie 7

Które z procedur sortujących:

- (a) insertionSort (przez wstawianie),
jest stabilny, gdyż każdy kolejny, zaczynając od początku tablicy, sortowany element jest wstawiany na koniec posortowanej części tablicy, więc w przypadku elementów o tej samej wartości, zachowana jest kolejność
- (b) quickSort (szybkie),
nie jest stabilny, przykładowo dla tablicy {2, 1, 1} już po pierwszym wywołaniu **partition** tracimy względną kolejność elementów o tej samej wartości, a kolejne wywołanie nie ma już nawet szansy jej przywrócić bo obejmie ono elementy o indeksach większych od 0
- (c) heapSort (przez kopcowanie),
nie jest stabilny, przykładowo dla tablicy {1, 1} sama budowa kopca nie dokona żadnych zamian elementów, ale już po pierwszym wyciągnięciu z kopca elementu o największej wartości czyli z indeksu zerowego tablicy tracimy względną kolejność elementów o tej samej wartości, gdyż jedynki zostaną wyjęte z kopca w tej samej kolejności w której były w tablicy ale trafiać będą na jej koniec przez co zamienią się miejscami
- (d) mergeSort (przez złączanie),
jest stabilny, procedura **merge** zachowuje względną kolejność elementów o tej samej wartości gdyż w przypadku porównywania dwóch elementów o tej samej wartości, wstawiany jest ten, który jest w tablicy pierwotnej na mniejszym indeksie
- (e) countingSort (przez zliczanie),
jest stabilny, ponieważ do tablicy wynikowej elementy są wstawiane po kolei z tablicy pierwotnej iterowanej od końca na kolejne dekrementowane indeksy na podstawie liczników
- (f) radixSort (pozycyjne),
jest stabilny, dzięki temu, że elementy umieszczane są kolejno w kolejkach FIFO z których później są wyciągane do tablicy pierwotnej
- (g) bucketSort (kubelkowe)
jest stabilny, ponieważ jest on połączeniem sortowania przez zliczanie i sortowania przez wstawianie, które oba są stabilne

są stabilne? W każdym przypadku uzasadnij stabilność lub znajdź konkretny przykład danych, dla których algorytm nie zachowa się stabilnie.

Zadanie 8

Napisz funkcję `void counting_sort(node* lista, int m)`; sortującą przez zliczanie listę linkowaną liczb całkowitych nieujemnych mniejszych od m . Procedura nie powinna usuwać ani tworzyć nowych węzłów, tylko sprytnie zmieniać pola `next` wykorzystując tylko $O(m)$ dodatkowej pamięci na wskaźniki.

Zadanie 9

(algorytm Hoare'a) Korzystając funkcji `int partition(int t[], int n)` znanej z algorytmu sortowania szybkiego napisz funkcję `int kty(int t[], int n)`, której wynikiem będzie k -ty co do wielkości element początkowo nieposortowanej tablicy `t`. Średnia złożoność Twojego algorytmu powinna wynieść $O(n)$.