INFO DLA ARTI'ego: POWRZUCAJ PRZYKLADY Z CORMEN'a BO JA NIE MAM MOZLIWOSCI, DZIEKI

Algorytmy sortowania by Paweł „Gothar" Pekról

Definicja. Sortowanie to uporządkowanie zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru. Przykład: sortowanie względem wartości każdego elementu (sortowanie liczb, słów)

Algorytmy sortowania:

1. Sortowanie przez wstawianie
2. Sortowanie przez kopcowanie
3. Sortowanie szybkie (Quicksort)
4. Sortowanie w czasie liniowym:
   4.1 Sortowanie przez zliczanie
   4.2 Sortowanie pozycyjne
   4.3 Sortowanie kubełkowe
5. Sortowanie bąbelkowe

1. Sortowanie przez wstawianie
   1.1 Złożoność obliczeniowa O(n^2)
   1.2 Dane wejściowe: Ciąg n liczb $< a_1, a_2, \dots, a_n >$
   1.3 Wynik: Permutacja (zmiana kolejności) $< a_1', a_2', \dots, a_n' >$ ciągu wejściowego taka, że $a_1' \leq a_2' \leq \dots \leq a_n'$
   1.4 Pseudokod:
   INSERTION-SORT(A)
   $for\ j \leftarrow to\ length[A]$
   $\quad do\ key \leftarrow A[j]$
   $\qquad Wstaw\ A[j] w\ posortowany\ ciąg\ A[1..\ j-1].$
   $\qquad i \leftarrow j-1$
   $\qquad while\ i > 0\ i\ A[i] > key$
   $\qquad\quad do\ A[i+1] \leftarrow A[i]$
   $\qquad\quad i \leftarrow i-1$
   $\qquad A[i+1] \leftarrow key$
   1.5 Schemat działania
      1. Utwórz zbiór elementów posortowanych i przenieś do niego dowolny element ze zbioru nieposortowanego.
      2. Weź dowolny element ze zbioru nieposortowanego.
      3. Wyciągnięty element porównuj z kolejnymi elementami zbioru posortowanego póki nie napotkasz elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
      4. Wyciągnięty element wstaw w miejsce gdzie skończyłeś porównywać.
      5. Jeśli zbiór elementów nieuporządkowanych jest niepusty wróć do punkt 2.
   1.6 Uwagi

1. Wydajny dla danych wstępnie posortowanych
2. Wydajny dla zbiorów o niewielkiej liczebności
3. Stabilny

2. Sortowanie przez kopcowanie

2.1 Złożoność obliczeniowa: $O(n * \log n)$

2.2 Dane wejściowe: Tablica A

2.3 Wynik: Posortowana tablica A

2.4 Pseudokod:

HEAPSORT(A)

BUILD-MAX-HEAP(A)
$for\ i \leftarrow length[A]\ downto\ 2$
  $do\ zamień\ A[1] \leftrightarrow A[i]$
    $heap-size[A] \leftarrow heap-size[A]-1$
    MAX-HEAPIFY(A,1)

BUILD-MAX-HEAP(A)

$heap-size[A] \leftarrow length[A]$
$for\ i \leftarrow \left\lfloor \frac{length[A]}{2} \right\rfloor downto\ 1$
  do MAX-HEAPIFY(A,1)

MAX-HEAPIFY(A,i)   i – indeks w tablicy A

$l \leftarrow LEFT(i)$
$r \leftarrow RIGHT(i)$
$if\ l \leq heap-size[A]\ i\ A[l] > A[i]$
  $then\ largest \leftarrow l$
  $else\ largest \leftarrow i$
$if\ r \leq heap-size[A] i\ A[r] > A[largest]$
  $then\ largest \leftarrow r$
$if\ largest \neq i$
  $zamień\ A[i] \leftrightarrow A[largest]$
    MAX-HEAPIFY(A, largest)

LEFT(i)

  $return\ 2i$

RIGHT(i)

$return\ 2i+1$

2.5 Uwagi

1. http://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie
2. http://en.wikipedia.org/wiki/Heapsort
3. „Wprowadzenie do algorytmów” T.H. Cormen s. 124 – 140

2.6 Przykład

3. Sortowanie szybkie (Quicksort)

3.1 Złożoność obliczeniowa: $O(n * \log n)$ (Pesymistyczna: $O(n^2)$)
3.2 Dane wejściowe: Tablica A, p – indeks początkowy w tablicy A, r – indeks końcowy w tablicy A
3.3 Wynik: Posortowana tablica A
3.4 Pseudokod:

QUICKSORT(A, p, r)
1. $if\ p < r$
2. $\quad then\ q \leftarrow PARTITION(A,p,r)$
3. $\quad QUICKSORT(A,p,q-1)$
4. $\quad QUICKSORT(A,q+1,r)$

Aby posortować całą tablicę wywołujemy $QUICKSORT(A,1,length[A])$

PARTITION(A, p, r)
1. $x \leftarrow A[r]$
2. $i \leftarrow p-1$
3. $for\ j \leftarrow p\ to\ r-1$
4. $\quad do\ if\ A[j] \leq x$
5. $\qquad then\ i \leftarrow i+1$
6. $\qquad\quad zamie\acute{n}\ A[i] \leftrightarrow A[j]$
7. $zamie\acute{n}\ A[i+1] \leftrightarrow A[r]$
8. $return\ i+1$

3.5 Schemat działania
1. Wybieramy pewien element tablicy tzw. element osiowy
2. Przenosimy na początek tablicy wszystkie elementy mniejsze od wybranego elementu osiowego
3. Przenosimy na koniec tablicy wszystkie elementy większe od wybranego elementu osiowego
4. Wstawiamy wybrany element w miejsce „pomiędzy” elementami mniejszymi i większymi
5. Sortujemy osobno początek i koniec tablicy (uwaga rekursja!)
6. Jeżeli podzielona tablica nie jest jednoelementowa wracamy do 1.

3.6 Uwagi
1. Efektywność algorytmu związana jest bezpośrednio z wyborem elementu osiowego (pesymistyczna gdy zawsze wybieramy element najmniejszy lub największy, optymistyczna gdy wybrany element jest medianą)
2. http://en.wikipedia.org/wiki/Quicksort
3. http://pl.wikipedia.org/wiki/Sortowanie_szybkie
4. „Wprowadzenie do algorytmów” T.H. Cormen s. 141-160

3.7 Przykład

4. Sortowanie w czasie liniowym

4.1 Sortowanie przez zliczanie

4.1.1 Złożoność obliczeniowa: $O(n+k)$ n – liczebność zbioru, k – rozpiętość danych

4.1.2 Dane wejściowe: Tablica $A[1..n]$ z danymi do sortowania, Tablica $B[1..n]$ - będą w niej umieszczone posortowane dane wejściowe, Tablica $C[1..k]$ tymczasowa tablica pomocnicza

4.1.3 Wynik: Tablica B z posortowanymi danymi z tablicy A

4.1.4 Pseudokod:

COUNTING-SORT(A, B, k)

$for\ j \leftarrow 0\ to\ k$
$\quad do\ C[i] \leftarrow 0$
$for\ j \leftarrow 1\ to\ length[A]$
$\quad do\ C[A[j]] \leftarrow C[A[j]] + 1$
$\quad C[i]\ zawiera\ teraz\ liczbę\ elementów\ równych\ i.$
$for\ i \leftarrow 1\ to\ k$
$\quad do\ C[i] \leftarrow C[i] + C[i-1]$
$\quad C[i]\ zaiera\ teraz\ liczbę\ elementów\ mniejszych\ bądź\ równych\ i.$
$for\ j \leftarrow length[A]\ downto\ 1$
$\quad do\ B\left[C\left[A[j]\right]\right] \leftarrow A[j]$
$\qquad C[A[j]] \leftarrow C[A[j]] - 1$

4.1.5 Uwagi

1. Ograniczenie: Algorytm zakłada, że klucze elementów najeżą do skończonego zbioru, co ogranicza możliwości jego zastosowania

4.1.6 Przykład

4.2 Sortowanie pozycyjne

4.2.1 Złożoność obliczeniowa: $O(d(n+k))$ d – liczba cyfr w kluczach, n – długość tablicy, k – ilość różnych cyfr

4.2.2 Dane wejściowe: Tablica A (n – elementowa, każdy element ma d cyfr, cyfra na pozycji 1 najmniej znacząca, cyfra na pozycji d najbardziej znacząca)

4.2.3 Wynik: Posortowana tablica A

4.2.4 Pseudokod:

RADIX-SORT(A, d)

$for\ i \leftarrow 1\ to\ d$
$\quad do\ posortuj\ stabilnie\ tablicę\ A\ według\ pozycji\ i$

4.2.5 Uwagi

1. http://pl.wikipedia.org/wiki/Sortowanie_pozycyjne
2. http://en.wikipedia.org/wiki/Radix_sort
3. „Wprowadzenie do algorytmów” T.H. Cormen s. 167-170

4.2.6 Przykład

4.3 Sortowanie kubełkowe

4.3.1 Złożoność obliczeniowa: $O(n)$

4.3.2 Dane wejściowe: Tablica A

4.3.3 Wynik: Posortowana tablica A

4.3.4 Pseudokod:

BUCKET-SORT(A)

$n \leftarrow length[A]$

2. $for\ i\ \leftarrow 1\ to\ n$
3. $do\ wstaw\ A[i] na\ listę\ B[\lfloor nA[i] \rfloor]$
4. $for\ i\ \leftarrow 0\ to\ n-1$
5. $do\ posortuj\ tablicę\ B[i] przez\ wstawianie$
6. $połącz\ listy\ B[0], B[1], \dots, B[n-1] z\ zachowaniem\ kolejności$

4.3.5 Schemat działania

Sortowanie kubełkowe opiera się na triku polegającym na podziale przedziału $[0,1)$ na n podprzedziałów jednakowych rozmiarów (tzw. kubełków), a następnie „rozrzuceniu" n liczb wejściowych do kubełków, do których należą. Ponieważ liczby są rozłożone jednostajnie w przedziale $[0,1)$, więc oczekujemy, że w każdym z kubełków będzie ich niewiele. Aby otrzymać ciąg wynikowy, sortujemy najpierw liczby w każdym z kubełków, następnie wypisujemy je, przeglądając po kolei kubełki.

4.3.6 Uwagi

1. Dane wejściowe są liczbami rzeczywistymi wybieranymi losowa z przedziału $[0,1)$ zgodnie z rozkładem jednostajnym

4.3.7 Przykład

5. Sortowanie bąbelkowe

5.1 Złożoność obliczeniowa: $O(n^2)$

5.2 Dane wejściowe: Tablica A

5.3 Wynik: posortowana tablica A

5.4 Pseudokod:

BUBBLESORT(A)

1. $for\ i\ \leftarrow 1\ to\ length[A]$
2. $\quad do\ for\ j\ \leftarrow length[A]\ downto\ i+1$
3. $\quad\quad do\ if\ A[j] < A[j-1]$
4. $\quad\quad\quad then\ zamień\ A[j] \leftrightarrow A[j-1]$