

Wstęp do programowania

Algorytmy na tablicach

Paweł Daniluk

Wydział Fizyki

Jesień 2014



Dwadzieścia pytań

Zasady

- 1 Osoba 1 wymyśla hasło z ustalonej kategorii.
- 2 Osoba 2 zadaje pytania, na które odpowiedź jest “tak” albo “nie”.
- 3 Po dwudziestym pytaniu należy podać hasło.
- 4 Nie wolno łgać.

Tablice

Język naturalny

Wektory:

$$\mathbf{v} = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{bmatrix}$$

Macierze:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix}$$

Ciągi:

$$F_{0,1} = 1$$

$$F_{n+1} = F_n + F_{n-1}$$

Tablice

Python – listy

```
>>> a = [1, 2, 3]
>>> a[0]
1
>>> a[1] = a[0] + a[2]
>>> a
[1, 4, 3]
>>> a[3] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Dodawanie elementów

```
>>> a.append(5)
>>> a
[1, 4, 3, 5]
```

Uwaga

Elementy są numerowane od 0.

Tablice

Python – tablice wielowymiarowe

```
>>> b = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> b
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> b[1][1]
5
>>> b[0]
[1, 2, 3]
>>> b[2]
[7, 8, 9]
```

Tablica wielowymiarowa jest w istocie listą list, zatem wiersze mogą być różnej długości.

Istnieją inne implementacje tablic, które są wydajniejsze w szczególnych zastosowaniach (np. w obliczeniach numerycznych).

Wycinki (ang. *slices*)

Ujemne indeksy

$a[-1]$ – oznacza ostatni element listy

$a[-i]$ – oznacza i -ty element od końca (miły fakt $-i \equiv_N N - i$)

Wycinki

$a[i:j]$ – oznacza listę zawierającą elementy od i -tego do $j - 1$ -szego włącznie

i i j mogą być ujemne. Można je również pominąć.

Wycinki z krokiem

$a[i:j:k]$ – j.w. ale co k -ty element

Problem

Dana jest posortowana tablica (lista) T liczb całkowitych długości N oraz liczba a . Określić, czy i na której pozycji a występuje w tablicy.

Wyszukiwanie

Problem

Dana jest posortowana tablica (lista) T liczb całkowitych długości N oraz liczba a . Określić, czy i na której pozycji a występuje w tablicy.

Rozwiązanie naiwne

Przeglądamy elementy tablicy po kolei.

Wyszukiwanie

Problem

Dana jest posortowana tablica (lista) T liczb całkowitych długości N oraz liczba a . Określić, czy i na której pozycji a występuje w tablicy.

Rozwiązanie naiwne

Przeglądamy elementy tablicy po kolei.

Dobry pomysł

- 1 Sprawdzamy czy $T[\lfloor \frac{N}{2} \rfloor] \leq a$.
- 2 Jeśli nie – $T' \leftarrow T[0..\lfloor \frac{N}{2} \rfloor]$
- 3 Jeśli tak – $T' \leftarrow T[\lfloor \frac{N}{2} \rfloor..N]$
- 4 Teraz wystarczy znaleźć a w T' .
- 5 Kończymy, gdy T' liczy jeden element.

Wyszukiwanie c.d.

Wydajność wyszukiwania naiwnego

W najgorszym przypadku poszukiwany element jest na końcu tablicy. Zanim do niego dojdziemy wykonamy N porównań.

Wydajność wyszukiwania binarnego

Po każdym kroku T staje się dwa razy krótsza. Zatem wykonamy $\lceil \log_2 N \rceil + 1$ porównań.

To duża różnica

N	$\lceil \log_2 N \rceil + 1$
1	1
10	5
100	8
1000	11
1000000	21
1000000000	31

Diabeł tkwi w szczegółach

- Należy zagwarantować, że tablica zmniejsza się w każdym kroku, co może być kłopotliwe dla małych N .
- Można sprawdzać, czy $T[\lfloor \frac{N}{2} \rfloor] = a$ i w tym momencie od razu przerywać, albo poczekać, aż T' będzie miało 0 lub 1 element i wtedy sprawdzić.
- Możliwa jest implementacja iteracyjna lub rekurencyjna.

Sortowanie

Problem

Dana jest tablica T długości N zawierająca liczby całkowite.
Uporządkować liczby w kolejności od najmniejszej do największej.

Jest wiele algorytmów sortowania, które różnią się wydajnością, stopniem skomplikowania i innymi subtelными cechami.

Sortowanie przez scalanie – (ang. *merge sort*)

Problem

Dane są dwie posortowane tablice T_1 i T_2 o długości N . Zbudować posortowaną tablicę T zawierającą elementy z obu tablic (T_1 i T_2).

Pomysł

Można porównywać pierwsze elementy T_1 i T_2 . Mniejszy z nich odcinać i wpisywać do T .

Sortowanie przez scalanie – (ang. *merge sort*) c.d.

Schemat rozwiązania – przechodzenie po dwóch tablicach

```
i=0
j=0

while i<N and j<N:
    if T1[i]<T2[j]:
        ...
        i=i+1
    else:
        ...
        j=j+1
```

Sortowanie przez scalanie – (ang. *merge sort*) c.d.

Ten algorytm można stosować wielokrotnie dzieląc sortowaną tablicę na tablice długości 1 i scalając je parami.

Uwaga

W taki sposób można sortować dane, które nie mieszczą się w pamięci.

Sortowanie bąbelkowe (*bubble sort*)

Pomysł

Polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

Przydatne schematy – iterowanie dopóki zachodzą zmiany

```
changed=True
```

```
while changed:  
    changed=False  
    ...
```


Sortowanie bąbelkowe (*bubble sort*)

Przydatne schematy – zamiana wartości

```
tmp=T[ i ]  
T[ i]=T[ j ]  
T[ j]=tmp
```

albo

```
T[ i ],T[ j ] = T[ j ],T[ i ]
```

Sortowanie przez wstawianie (*insertion sort*)

Pomysł

- 1 Utwórz zbiór T' .
- 2 Weź dowolny element z T .
- 3 Wyciągnięty element porównuj z kolejnymi elementami z T póki nie napotkasz elementu większego.
- 4 Wyciągnięty element wstaw w miejsce, gdzie skończyłeś porównywać.
- 5 Jeśli zostały elementy w T wróć do punktu 2.

Przydatny schemat – działanie w miejscu

Jeżeli w pkt. 2 zawsze będziemy brali pierwszy element T , to możemy budować T' w miejscu, które było zajmowane przez rozważony początek tablicy T .

Sortowanie przez wstawianie (*insertion sort*) c.d.

Przydatny schemat – znajdowanie pierwszego elementu niemniejszego niż a w posortowanej tablicy T

$i=0$

```
while i < N and T[i] < a:  
    i = i + 1
```

$i < N \Rightarrow T[i] \geq a$

Sortowanie przez wstawianie (*insertion sort*) c.d.

Przydatny schemat – wstawianie wartości a do tablicy długości N w miejsce i -te

```
j=N+1
```

```
while j>i:
```

```
    T[j]=T[j-1]
```

```
    j=j-1
```

```
# W tym momencie i==j
```

```
T[j]=a
```

Problem

Lista T wypełniona zerami i jedynkami reprezentuje ciąg n urn w których znajdują się żetony białe (0) i czerwone (1). Podać algorytm, który zamieniając żetony miejscami doprowadzi do sytuacji, w której wszystkie żetony białe znajdują się na lewo od czerwonych.

Pomysł

Należy przesuwac się indeksem c od początku tablicy, zaś indeksem b od końca. Intencją jest utrzymywanie następującego niezmiennika: wszystkie elementy tablicy o indeksach mniejszych od c są czerwone, zaś większych od b są białe. Indeksy c i b będą się do siebie zbliżać i ostatecznie, gdy c będzie równe b , tablica będzie uporządkowana.

Wyszukiwanie mediany

Problem

Dana jest tablica T długości N (nieuporządkowana). Znaleźć k -ty w porządku rosnącym element (dla mediany $k = \lfloor \frac{n}{2} \rfloor$).

Pomysł

Można wziąć dowolny element a i podzielić tablicę na elementy mniejsze równe od a i większe od a (flaga polska). k -ty element jest w jednej z uzyskanych części (w zależności od k i ich liczności).

Wyszukiwanie mediany c.d.

Schemat rozwiązania

```
b=0
e=N-1

while b < e:
    p=pivot(T, b, e)

    if p>k:
        e=p-1
    else:
        b=p

return T[b]
```

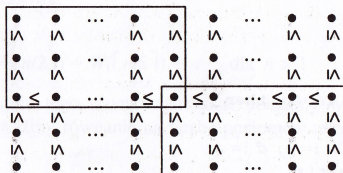
Wyszukiwanie mediany c.d.

Uwaga

To rozwiązanie przy pechowym losowaniu elementów może być powolne.

Istnieje rozwiązanie lepsze – algorytm
Bluma-Floyda-Pratta-Rivesta-Tarjana

Elementy, które
są na pewno \leq
elementu
dzielącego



Elementy, które
są na pewno \geq
elementu
dzielącego

Rys. 2.26. Podział ciągu względem mediany median

Segment o maksymalnej sumie

Problem

Dana jest tablica T liczb całkowitych długości N . Obliczyć maksymalną sumę segmentu (wycinka) tej tablicy. Przyjmujemy, że segment pusty ma sumę 0.

Przydatny schemat – obliczanie sumy przy pomocy akumulatora

```
akumulator=0
for i in l:
    akumulator = akumulator + i
```

Dodawanie można zastąpić dowolną inną operacją.

Segment o maksymalnej sumie c.d.

Przydatny schemat – wykorzystywanie wyników pośrednich

Podczas liczenia sumy wszystkich elementów tablicy, akumulator przyjmuje wartości sum kolejnych prefiksów.

Przydatna sztuczka w tym problemie – sumy prefiksowe

$$P(k) = \sum_{i=0}^{k-1} T_i$$

$$\sum_{i=s}^{e-1} T_i = P(e) - P(s)$$

$$\max_{0 \leq s \leq e \leq N} \sum_{i=s}^{e-1} T_i = \max_{0 \leq s \leq e \leq N} P(e) - P(s) = \max_{0 \leq e \leq N} P(e) - \min_{0 \leq s \leq e} P(s)$$

Zadania

- 1 Generowanie ciągu N losowych liczb całkowitych.
- 2 Generowanie monotonicznie rosnącego ciągu liczb N całkowitych.
- 3 Wyszukiwanie binarne.
- 4 Scalanie dwóch uporządkowanych list.
- 5 Sortowanie bąbelkowe.
- 6 Sortowanie przez wstawianie.
- 7 Flaga polska.
- 8 Segment o maksymalnej sumie.

Zadania

- 1 Generowanie ciągu N losowych liczb całkowitych.
- 2 Generowanie monotonicznie rosnącego ciągu liczb N całkowitych.
- 3 Wyszukiwanie binarne.
- 4 Scalanie dwóch uporządkowanych list.
- 5 Sortowanie bąbelkowe.
- 6 Sortowanie przez wstawianie.
- 7 Flaga polska.
- 8 Segment o maksymalnej sumie.

Generator liczb pseudolosowych

```
>>> import random
>>> random.randint(0,1)
0
>>> random.randint(0,1)
1
>>> random.randint(0,1)
0
```

[http://bioexploratorium.pl/wiki/Wstęp_do_programowania_2014z](http://bioexploratorium.pl/wiki/Wst%C4%99p_do_programowania_2014z)

Omówienie algorytmów sortujących z animacjami

<http://www.sorting-algorithms.com>