

WYBÓR  $k$ -TEGO ELEMENTU

## 1 Definicja problemu

*Dane:*  $T[1..n]$  - ciąg elementów należących do zbioru liniowo uporządkowanego,  
 $k$  - liczba z przedziału  $\langle 1, n \rangle$ .

*Wynik:*  $k$ -ty co do wielkości element ciągu  $T$

**ZAŁOŻENIE** (nie zmniejszające ogólności): wszystkie elementy w  $T$  są różne.

**MODEL OBLICZEŃ:** na elementach ciągu  $T$  dokonujemy jedynie porównań.

## 2 Szczególne przypadki

- $k = 1$ .  
 Konieczna i wystarczająca liczba porównań  $= n - 1$ .
- $k = 2$ .

**Twierdzenie 1** W tym przypadku potrzeba i wystarcza  $n - 2 + \lceil \log n \rceil$  porównań.

Dowód(szkic)

$\Rightarrow$

Konstruujemy algorytm, działający w  $\lceil \log n \rceil$  rundach: w 1-szej rundzie porównywane są pary elementów  $\langle T[2k-1], T[2k] \rangle$  (dla  $k = 1, \lfloor \frac{n}{2} \rfloor$ ). "Zwycięzcy" tych porównań (oraz element  $T[n]$  - w przypadku nieparzystego  $n$ ) przechodzą do następnej rundy. W kolejnych rundach postępujemy analogicznie.

Oczywiście ostatnia runda wyznaczy element największy w  $T$ . Do tej pory algorytm wykona  $n - 1$  porównań. Można to łatwo udowodnić, jeśli na działanie tego algorytmu popatrzeć jako na drzewo binarne o  $n$  liściach: liście tego drzewa etykietujemy elementami  $T[i]$ , a każdy wierzchołek wewnętrzny - większą spośród etykiet jego synów. Tak więc wierzchołki wewnętrzne odpowiadają porównaniom wykonanym przez algorytm.

Znalezienie 2-ego co do wielkości elementu sprowadza się teraz do znalezienia największego elementu spośród tych, które były porównywane z elementem największym. Ponieważ elementów tych jest  $\lceil \log n \rceil$ , wystarczy teraz  $\lceil \log n \rceil - 1$  porównań.

$\Leftarrow$

Rozważmy dowolny algorytm  $\mathcal{A}$  wyznaczający 2-gi co wielkości element. Zauważmy, że  $\mathcal{A}$  wyznacza jednocześnie element największy. Niech bowiem  $X = T[j]$  będzie rozwiązaniem podanym przez  $\mathcal{A}$ . Elementem największym jest ten, z którym  $X$  "przegrał" porównanie. Element taki musi istnieć (w przeciwnym razie  $\mathcal{A}$  podałby  $T[j]$  jako rozwiązanie także dla takich danych, w których  $T[j]$  zwiększylibyśmy dowolnie, a pozostałe elementy pozostawilibyśmy bez zmian) i to dokładnie jeden (w przeciwnym razie  $X$  nie byłby drugim elementem). Tak więc  $\mathcal{A}$  musi wykonać  $n - 1$  porównań, by wyznaczyć element największy. Porównania te nie wnoszą żadnej informacji o wzajemnej relacji pomiędzy elementami, które jedynie z nim przegrały porównanie (a  $X$  jest największym z tych elementów). Tak więc nasz dowód sprowadza się do pokazania, że w najgorszym przypadku element największy bierze udział w co najmniej  $\lceil \log n \rceil$  porównaniach wykonywanych przez  $\mathcal{A}$ . To będzie treścią zadania na ćwiczenia (rozwiązanie możesz znaleźć w ([7], s.212).

## 3 Przypadek ogólny

### 3.1 Algorytm deterministyczny

IDEA Stosujemy metodę Dziel i Zwycięzaj. Rozdzielamy  $T$  na dwa podzbiory:  $U$  i  $V = T \setminus U$ , takie że wszystkie elementy  $U$  są mniejsze od wszystkich elementów  $V$ . Teraz porównanie  $k$  z mocą  $U$  pozwala określić, w którym ze zbiorów znajduje się szukany element. W ten sposób redukujemy problem do problemu szukania elementu w zbiorze mniejszym.

**Procedure** *SELECTION*( $k, T$ )

1. if  $|T|$  małe then *sort*( $T$ ) & return ( $T[k]$ )
2.  $p \leftarrow$  jakiś element z  $T$
3.  $U \leftarrow$  elementy  $T$  mniejsze od  $p$
4. if  $k \leq |U|$  then return ( *SELECTION*( $k, U$ ) )  
else return (*SELECTION*( $k - |U|, T \setminus U$ ))

UWAGA: W powyższej procedurze zbiory  $T$  i  $U$  wygodnie jest pamiętać w tablicach.

Aby algorytm był efektywny, musimy zagwarantować, że zbiory  $U$  i  $T \setminus U$  są istotnie mniej liczne od zbioru  $T$ . W tym celu zbiór  $T$  dzielimy na 5-cioelementowe grupy. W każdej grupie wybieramy medianę (tj. środkowy element) a następnie rekurencyjnie wybieramy medianę tych median.

**function** *Pseudomed*( $T$ )

1. Podziel  $T$  na rozłączne podzbiory 5-cioelementowe  $C_j$  ( $j = 1, \dots, \lceil |T|/5 \rceil$ )  
{jeśli  $|T|$  nie dzieli się przez 5, to  $C_{\lceil |T|/5 \rceil}$  zawiera mniej niż 5 elementów }
2. for  $i \leftarrow 1$  to  $\lceil |T|/5 \rceil$  do  $s_i \leftarrow \text{adhocmed}(C_i)$
3.  $S \leftarrow \{s_i \mid i = 1, \dots, \lceil |T|/5 \rceil\}$
4. return (*SELECTION*( $\lceil \frac{|S|}{2} \rceil, S$ ))

**Twierdzenie 2** Jeśli w procedurze *SELECTION* wybór elementu  $p$  dokonywany jest funkcją *Pseudomed*, to procedura *SELECTION* wyznacza  $k$ -ty element ciągu  $T$  w czasie  $O(n)$ .

Pomijamy dowód poprawności procedury *Selection*. W oszacowaniu jej kosztu kluczowym punktem jest następujący lemat:

**Lemat 1** Jeśli element  $p$  został wybrany funkcją *Pseudomed*, to każdy ze zbiorów  $U$  i  $T \setminus U$  zawiera nie mniej niż  $\frac{3}{10}n - 4$  elementy.

DOWÓD: Istnieje co najmniej (a przy założeniu, że wszystkie elementy w  $T$  są różne - dokładnie)  $\lceil \frac{1}{2} \lceil n/5 \rceil \rceil$  grup, których mediana jest nie mniejsza od  $p$ . W każdej z tych grup, poza ostatnią, znajdują się co najmniej 3 elementy nie mniejsze od  $p$  (takimi są na pewno elementy większe od mediany w danej grupie). Tak więc elementów nie mniejszych od  $p$  jest co najmniej  $3(\lceil \frac{1}{2} \lceil n/5 \rceil - 1)$ . W tym wliczony jest  $p$ . Ponieważ zakładamy, że wszystkie elementy są różne, więc elementów większych od  $p$  jest co najmniej  $\frac{3}{10}n - 4$ .

W podobny sposób pokazujemy, że co najmniej tyle samo jest elementów mniejszych od  $p$ . □

**Koszt Procedury *Selection*.** Niech  $T$  będzie funkcją ograniczającą z góry ten koszt. Bez zmniejszenia ogólności możemy założyć, że  $T$  jest monotoniczną funkcją niemalejącą. Ponieważ koszt wszystkich operacji poza rekurencyjnymi wywołaniami *Selection* można ograniczyć funkcją liniową otrzymujemy następującą nierówność rekurencyjną:

$$T(n) \leq T(\lceil n/5 \rceil) + T(7n/10 + 4) + O(n) \quad \text{dla odpowiednio dużych } n.$$

Można łatwo sprawdzić, że  $T(n)$  jest  $O(n)$ .

UWAGI:

- Stałą ukrytą w Twierdzeniu 2 pod "dużym  $O$ " można oszacować z góry przez 5.43.
- Obecnie najszybszy asymptotycznie algorytm wykonuje  $2.9442n + o(n)$  porównań ([3]).
- Dolna granica - każdy algorytm deterministyczny wykonuje co najmniej  $2n$  porównań.

## 3.2 Algorytmy zrandomizowane

### 3.2.1 Algorytm Hoare'a

W procedurze *Selection* element  $p$  wybieramy w sposób losowy. W ten sposób otrzymujemy algorytm o małej średniej liczbie porównań (choć oczywiście nadal liniowej) ([5],[2]).

Zauważ podobieństwo tego algorytmu do zrandomizowanego *Quicksortu*. Podobnie jak w tamtym algorytmie element rozdzielający można wybierać inaczej, np. jako medianę spośród losowej próbki kilku elementów zbioru  $T$  ([4]).

### 3.2.2 Algorytm *LazySelect*

IDEA Ze zbioru  $S$  wybieramy losowo próbkę  $R$ . Próbkę powinna być niezbyt liczna, by można było szybko ją posortować. Znajdujemy w  $R$  dwa elementy  $L$  i  $H$ , takie że z dużym prawdopodobieństwem podzbiór  $P \subseteq S$ , elementów większych od  $L$  i mniejszych od  $H$ , jest nieduży oraz szukany element należy do  $P$  (możemy go wówczas łatwo znaleźć po posortowaniu  $P$ ).

#### Algorytm *LazySelect*

1. Wybierz losowo, niezależnie, z powtórzeniami próbkę  $R$  złożoną z  $n^{\frac{3}{4}}$  elementów zbioru  $S$ .
2. Posortuj  $R$  w czasie  $O(n^{\frac{3}{4}} \log n)$ .
3.  $x \leftarrow kn^{-\frac{1}{4}}$ ;  $L \leftarrow R[l]$ ;  $H \leftarrow R[h]$   
gdzie  $l = \max\{\lfloor x - \sqrt{n} \rfloor, 1\}$  oraz  $h = \min\{\lfloor x + \sqrt{n} \rfloor, n^{\frac{3}{4}}\}$
4. Porównując  $L$  i  $H$  ze wszystkimi elementami z  $S$  oblicz:
  - $r_S(L) = \#\{y \in S \mid y < L\}$
  - $P \leftarrow \{y \in S \mid L \leq y \leq H\}$
5. Sprawdź czy:
  - $k$ -ty element zbioru  $S$  znajduje się w  $P$ , tj. czy  $r_S(L) < k \leq r_S(L) + |P|$ ,
  - $|P| \leq 4n^{\frac{3}{4}} + 2$ .Jeśli nie, to powtórz kroki 1-4.
6. Posortuj  $P$ .  
**return**  $(P_{(k-r_S(L)+1)})$ .

**Twierdzenie 3** Z prawdopodobieństwem  $em = 1 - O\left(\frac{1}{\sqrt[4]{n}}\right)$  *LazySelect* potrzebuje tylko jednej iteracji kroków 1-4 do znalezienia  $k$ -tego elementu zbioru  $S$ . Tak więc z prawdopodobieństwem  $em 1 - O\left(\frac{1}{\sqrt[4]{n}}\right)$  *LazySelect* zatrzymuje się po wykonaniu  $2n + o(n)$  porównań.

Dowód tego twierdzenia nie jest trudny, ale wykracza poza zakres wykładu. Można go znaleźć w książce [8] (str. 47-50). Zainteresowani będą mogli go poznać na wykładzie Algorytmy Zrandomizowane.

Teraz ograniczymy się do podania szkicu:

1. pokazujemy, że  $kn^{-\frac{1}{4}}$  jest wartością oczekiwaną liczby elementów w  $R$  nie większych od szukanego,
2. pokazujemy, że wariancja tej liczby jest mniejsza od  $\sqrt{n}$ ,
3. korzystamy z poniższej Nierówności Czebyszewa.

**Twierdzenie 4 (nierówność Czebyszewa)** *Jeśli  $X$  jest zmienną losową (o wartościach rzeczywistych) o wartości oczekiwanej  $\mu_X$  i odchyleniu standardowym  $\sigma_X$ . Wówczas*

$$\forall t \in \mathcal{R}_+ \quad \mathbf{P}\left[|X - \mu_X| \geq t\sigma_X\right] \leq \frac{1}{t^2}$$

## Literatura

- [1] M.Blum, R.W.Floyd, V.Pratt, R.L.Rivest, R.E.Tarjan, Time bounds for selection, *Journal of Computer and System Sciences*, 7(1973), 448–461.
- [2] T.Cormen, C.Leiserson, R.L.Rivest, *Introduction to Algorithms*, The MIT Press, 1990.
- [3] D.Dor, U.Zwick, Median selection requires  $O(2+\epsilon)n$  comparisons, Proceedings of the 37th FOCS, 1996, 125-134.
- [4] R.W.Floyd, R.L.Rivest, Expected time bounds for selection, *Communication of the ACM*, 18(1975), 165–172.
- [5] C.A.R.Hoare, Algorithm 63 (partition) and 65 (find), *Communication of the ACM*, 4(1961), 321–322.
- [6] R.M.Karp, Probabilistic recurrence relations, w: *Proceedings of the 23rd STOC*, 1991, 190–197.
- [7] D.E.Knuth, *The Art of Computer Programming*, vol. 3, Addison-Wesley, 1973.
- [8] R.Motwani, P.Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.