ALGORYTMY I STRUKTURY DANYCH
WYKŁAD II (materiały pomocnicze)

Problem wyszukania

Polsko Japońska Wyższa Szkoła Technik Komputerowych

Warszawa, 9 listopada 2008

Plan wykładu:

- uniwersum nieuporządkowane:
  - algorytm sekwencyjny,
  - algorytm „turniej” dla problemu 2-go co do wielkości,
  - algorytm rekurencyjny dla problemu min-max,
- uniwersum uporządkowane:
  - algorytm „skoki co k”
  - algorytm binarny,
  - algorytm interpolacyjny.

<u>Plan wykładu c.d:</u>

- problem k-tego co do wielkości:
  - rozwiązanie „naiwne”,
  - algorytm Hoare,
  - algorytm Bluma-Floyda-Pratta-Rivesta-Trajana,

Uniwersum nieuporządkowane

(algorytm sekwencyjny)

Uniwersum nieuporządkowane – algorytm sekwencyjny

Zadanie (problem wyszukania). Niech A będzie tablicą $n \geq 1$ różnych liczb naturalnych. Podaj algorytm, który wyznaczy indeks tablicy i, gdzie $0 \leq i < n$ taki, że $A[i] = 0$ (zakładamy, że poszukiwany element znajduje się w tablicy A).

Rozwiązanie. Algorytm sekwencyjny:

```
int FindIndex(int A[n],int n) { // wp:  n ≥ 1
   int i;

   for (i=0;i<n;i++)
      if (A[i]==0) return i; // wk:  A[i] = 0
}
```

Pytanie. Jaka jest złożoność czasowa algorytmu FindIndex w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu FindIndex w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu FindIndex?

Twierdzenie. *Algorytm FindIndex jest optymalnym rozwiązaniem problemu wyszukania w uniwersum nieuporządkowanym.*

Uniwersum nieuporządkowane

(algorytm „turniej” dla problemu 2-go co do wielkości)

Uniwersum nieuporządkowane – algorytm „turniej”

Zadanie (problem 2-go co do wielkości). Niech A będzie tablicą n różnych liczb naturalnych, gdzie $n = 2^k$ i $k \in \mathbb{N}^+$. Podaj algorytm, który wyznaczy drugi co do wielkości element tablicy A.

Rozwiązanie. Algorytm sekwencyjny:

```
int Find2nd(int A[n],int n) { // wp:  n = 2^k i k ∈ N+
   int i,max=Max(A[0],A[1]),sec=Min(A[0],A[1]);

   for (i=2;i<n;i++)
      if (A[i]>max) { sec=max; max=A[i];}
      else if (A[i]>sec) sec=A[i];

   return sec // wk:  sec jest drugim co do wielkości elementem tablicy A
}
```

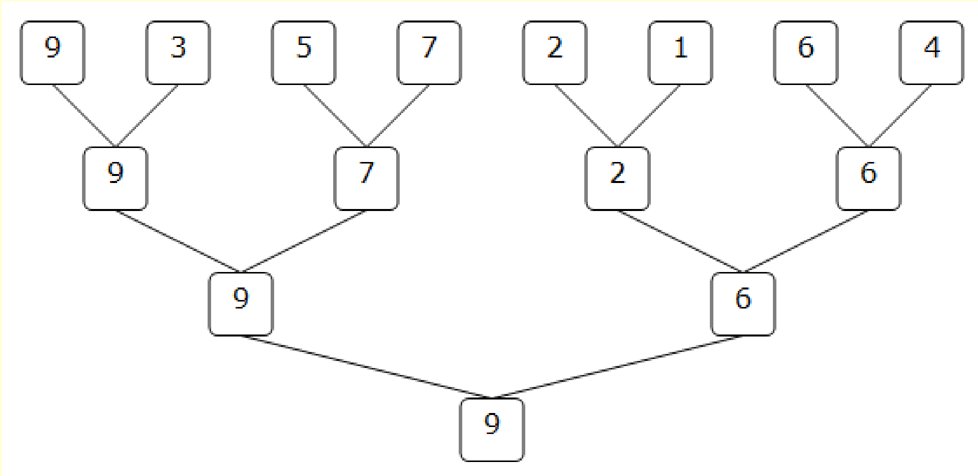
Pytanie. Jaka jest złożoność czasowa algorytmu Find2nd w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu Find2nd w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu Find2nd?

Uniwersum nieuporządkowane – algorytm „turniej”

Idea algorytmu „turniej”. Zbuduj drzewo turnieju zgodnie z zasadą „przechodzi tylko wygrywający”, np. dla $A = [9, 3, 5, 7, 2, 1, 6, 4]$ i $n = 8$ otrzymujemy:

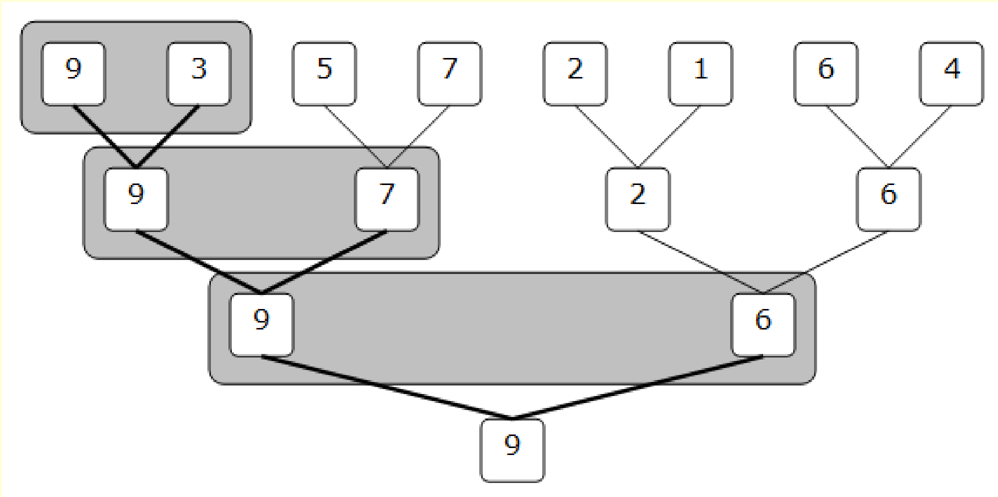


Pytanie. Ile porównań elementów tablicy A jest niezbędnych do zbudowania drzewa turnieju w rozważanym przykładzie (tj. $n = 8$) i w przypadku ogólnym?

Pytanie. Jak najmniejszym kosztem można znaleźć element 2-gi co do wielkości?

Uniwersum nieuporządkowane – algorytm „turniej”

Wniosek. Element 2-gi co do wielkości jest jednym z tych, które „przegrały” z elementem największym, czyli:



Pytanie. Jaka jest złożoność czasowa i pamięciowa algorytmu „turniej” w rozważanym przykładzie i w przypadku ogólnym?

Twierdzenie. *Algorytm „turniej” jest optymalnym rozwiązaniem dla problemu wyszukania 2-go co do wielkości elementu w uniwersum nieuporządkowanym.*

Uniwersum nieuporządkowane

(algorytm rekurencyjny dla problemu min-max)

Uniwersum nieuporządkowane – algorytm rekurencyjny dla problemu min-max

Zadanie (problem min-max). Niech A będzie tablicą n liczb naturalnych, gdzie $n = 2^k$ i $k \in \mathbb{N}^+$. Podaj algorytm, który wyznaczy element minimalny i maksymalny w tablicy A.

Rozwiązanie. Algorytm sekwencyjny:

```
(int,int) FindMinMax_1(int A[n],int n) { // wp: n = 2^k i k ∈ N+
   int i,min=Min(A[0],A[1]),max=Max(A[0],A[1]);

   for (i=2;i<n;i++) {
      if (A[i]<min) min=A[i];
      if (A[i]>max) max=A[i];
   }

   return (min,max); // wk: min = min(A), max = max(A)
}
```

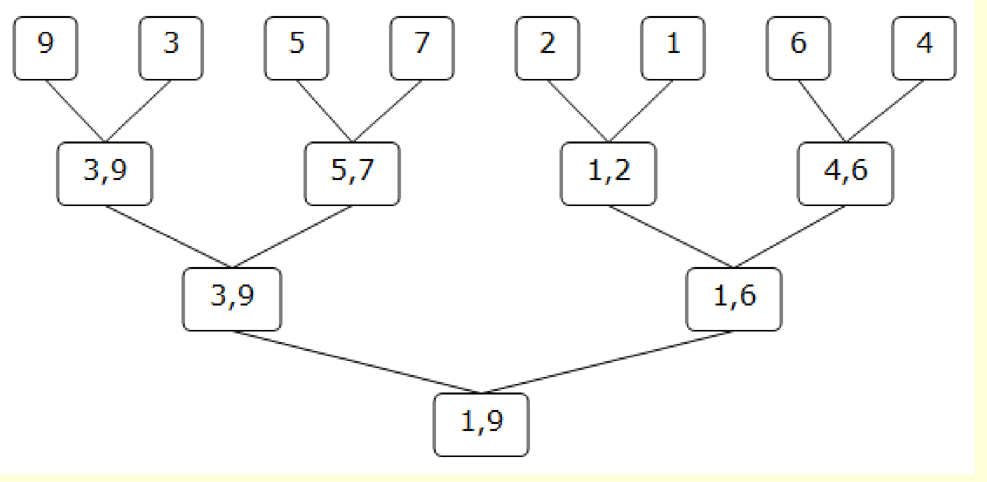
Pytanie. Jaka jest złożoność czasowa algorytmu FindMinMax_1 w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu FindMinMax_1 w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu FindMinMax_1?

Uniwersum nieuporządkowane – algorytm rekurencyjny dla problemu min-max

Idea algorytmu rekurencyjnego. Zbuduj zmodyfikowane drzewo turnieju zgodnie z zasadą „przechodzą tylko elementy minimalny i maksymalny”, np. $A = [9, 3, 5, 7, 2, 1, 6, 4]$ i $n = 8$ otrzymujemy:



Pytanie. Ile porównań elementów tablicy A jest niezbędnych do zbudowania zmodyfikowanego drzewa turnieju w rozważanym przykładzie (tj. $n = 8$)?

Uniwersum nieuporządkowane – algorytm rekurencyjny dla problemu min-max

Rozwiązanie. Algorytm rekurencyjny:

```
(int,int) FindMinMax_2(int A[n],int l,int r) { // wp: n = 2^k i k ∈ N+
   int min,max;
   (int,int) result1,result2;

   if (r-l==1)
      if (A[r]>A[l]) return (A[l],A[r]);
      else return (A[r],A[l]);
   else {
      result1=FindMinMax_2(A,l,(l+r) div 2);
      result2=FindMinMax_2(A,((l+r) div 2)+1,r);

      if (result1[0]<result2[0]) min=result1[0] else min=result2[0];
      if (result1[1]>result2[1]) max=result1[1] else max=result2[1];

      return (min,max); // wk: min = min(A[l], A[l+1], ..., A[r]),
                        //     max = max(A[l], A[l+1], ..., A[r])
   }
}
```

Uniwersum nieuporządkowane – algorytm rekurencyjny dla problemu min-max

Poprawność częściowa algorytmu. Dowód przez indukcję ze względu na k, gdzie $n = 2^k$ i $k \in \mathbb{N}^+$.

- baza indukcji: dla $k = 1$, tj. $n = 2$ zachodzi $r - l = 1$, zatem wykonany jest pierwszy warunek instrukcji warunkowej

  ```
  if (A[r]>A[l]) return (A[l],A[r]);
  else return (A[r],A[l]);
  ```

  stąd wynik algorytmu jest poprawny,

- założenie indukcyjne: dla $k = p$, gdzie $p \geq 1$, wynik algorytmu FindMinMax dla tablicy A rozmiaru $n = 2^p$ jest poprawny,

- teza indukcyjna: dla $k = p + 1$, gdzie $p \geq 1$, wynik algorytmu FindMinMax dla tablicy A rozmiaru $n = 2^{p+1}$ jest poprawny,

Uniwersum nieuporządkowane – algorytm rekurencyjny dla problemu min-max

- dowód tezy: dla $k > 1$, tj. $n > 2$ zachodzi $r - l > 2$, zatem wykonany jest drugi warunek instrukcji warunkowej, którego początkowe instrukcje są postaci

```
result1=FindMinMax_2(A,l,(l+r) div 2);
result2=FindMinMax_2(A,((l+r) div 2)+1,r);
```

Korzystając z założenia indukcyjnego zachodzi kolejno

$$
\begin{aligned}
result1[0] &= \min\left(A[l], A[l+1], \ldots, A\left[\left\lfloor\frac{l+r}{2}\right\rfloor\right]\right),\\
result1[1] &= \max\left(A[l], A[l+1], \ldots, A\left[\left\lfloor\frac{l+r}{2}\right\rfloor\right]\right),\\
result2[0] &= \min\left(A\left[\left\lfloor\frac{l+r}{2}\right\rfloor+1\right], A\left[\left\lfloor\frac{l+r}{2}\right\rfloor+2\right], \ldots, A[r]\right),\\
result2[1] &= \max\left(A\left[\left\lfloor\frac{l+r}{2}\right\rfloor+1\right], A\left[\left\lfloor\frac{l+r}{2}\right\rfloor+2\right], \ldots, A[r]\right),
\end{aligned}
$$

gdzie $\left\lfloor\frac{l+r}{2}\right\rfloor - l = r - \left\lfloor\frac{l+r}{2}\right\rfloor + 1 = n$ i $n = 2^p$.

Uniwersum nieuporządkowane – algorytm rekurencyjny dla problemu min-max

- dowód tezy c.d.: następnie wykonane są instrukcje warunkowe

```
if (result1[0]<result2[0]) min=result1[0] else min=result2[0];
if (result1[1]>result2[1]) max=result1[1] else max=result2[1];

return (min,max);
```

Stąd

$$\begin{aligned} min &= \min\left(A\left[l\right], A\left[l+1\right], \ldots, A\left[r\right]\right), \\ max &= \max\left(A\left[l\right], A\left[l+1\right], \ldots, A\left[r\right]\right), \end{aligned}$$

gdzie $r - l = 2n = 2 \cdot 2^p = 2^{p+1}$, co kończy dowód.

Wniosek. Algorytm FindMinMax_2 jest częściowo poprawnym rozwiązaniem problemu min-max.

Pytanie. Jak uzasadnić całkowitą poprawność algorytmu FindMinMax_2 dla problemu min-max?

Uniwersum nieuporządkowane – algorytm rekurencyjny dla problemu min-max

Złożoność czasowa. Niech $T(n)$ będzie liczbą operacji porównań jakie wykonuje algorytm FindMinMax_2 dla danych rozmiaru n, wtedy:

$$T(n) = \begin{cases} 1 & \text{dla } n = 2 \\ 2T\left(\frac{n}{2}\right) + 2 & \text{dla } n > 2 \end{cases},$$

czyli dla $n = 2^k$ i $k \in \mathbb{N}^+$

$$T\left(2^k\right) = \begin{cases} 1 & \text{dla } k = 1 \\ 2T\left(2^{k-1}\right) + 2 & \text{dla } k > 1 \end{cases},$$

i ostatecznie $T\left(2^k\right) = \frac{3}{2}2^k - 2$, czyli $T(n) = \frac{3}{2}n - 2$.

Uzasadnienie. Dla $k = 1$ mamy $T\left(2^1\right) = T(2) = \frac{3}{2}2 - 2 = 1$ co stanowi bazę indukcji. Załóżmy, że dla $k > 1$ zachodzi $T\left(2^k\right) = \frac{3}{2}2^k - 2$, wtedy dla $k + 1$ mamy $T\left(2^{k+1}\right) = 2T\left(2^k\right) + 2$ i na podstawie założenia

$$T\left(2^{k+1}\right) = 2\left(\frac{3}{2}2^k - 2\right) + 2 = 3 \cdot 2^k - 2 = \frac{3}{2}2^{k+1} - 2$$

co kończy dowód indukcyjny. Stąd dla $n = 2^k$ i $k \in \mathbb{N}^+$ zachodzi $T(n) = \frac{3}{2}n - 2$.

Uniwersum nieuporządkowane – algorytm rekurencyjny dla problemu min-max

Pytanie. Jaka jest złożoność czasowa algorytmu FindMinMax_2 w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu FindMinMax_2 w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu FindMinMax_2?

Twierdzenie. *Algorytm rekurencyjny FindMinMax_2 jest optymalnym rozwiązaniem dla problemu min-max w uniwersum nieuporządkowanym.*

Zadanie (*). Podaj algorytm sekwencyjny dla problemu min-max, którego średnia złożoność czasowa będzie istotnie mniejsza niż złożoność czasowa algorytmu sekwencyjnego FindMinMax_1.

Uniwersum uporządkowane

(algorytm „skoki co k”)

Uniwersum uporządkowane – algorytm „skoki co k”

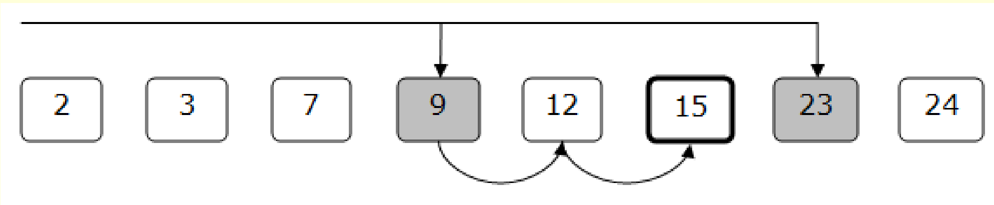
Zadanie (problem wyszukania). Niech A będzie uporządkowaną tablicą $n \geq 1$ różnych liczb naturalnych. Podaj algorytm, który wyznaczy indeks tablicy i, gdzie $0 \leq i < n$ taki, że $A[i] = x$ i $x \in \mathbb{N}$ (zakładamy, że poszukiwany element znajduje się w tablicy A).

Pytanie. Czy algorytm sekwencyjny FindIndex jest poprawnym rozwiązaniem dla problemu „książki telefonicznej”? Jeżeli tak, to:

- jaka jest złożoność czasowa algorytmu tego algorytmu w przypadku średnim?
- jaka jest złożoność czasowa algorytmu tego algorytmu w przypadku pesymistycznym?

Idea algorytmu „skoki co k”. Porównujemy liczbę x z co k-tym elementem tablicy A poczynając od elementu k-tego, tj. $A[k], A[2k], A[3k], \ldots$. Proces przerywamy wtedy, gdy $A[ik] > x$, dla pewnego $i \in \mathbb{N}$. Sekwencyjnie przeglądamy $k-1$ elementy
$A[ik-k], A[ik-(k-1)], \ldots, A[ik-1]$.

Przykład. Szukamy indeksu liczby 15 w tablicy $A = [2, 3, 7, 9, 12, 15, 23, 24]$ dla $k = 3$.



Uniwersum uporządkowane – algorytm „skoki co k”

Rozwiązanie. Algorytm „skoki co k”:

```
int Skoki(int A[n],int n,int k,int x) {
   // wp:  n ≥ 1 i A[0] < A[1] < ... < A[n-1] i x ∈ A
   int i=0;

   while (i<n AND x≥A[i]) i=i+k; // nz:  x ≥ A[0], A[1], ..., A[i-k]

   i=i-k;
   while (A[i]!=x) i=i+1;

   return i; // wk:  A[i] = x
}
```

Pytanie. Jaka jest złożoność czasowa algorytmu „skoki co k” w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu „skoki co k” w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu „skoki co k” ?

Pytanie. Dla jakiej wartości parametru k algorytm „skoki co k” ma najmniejszą złożoność pesymistyczną?

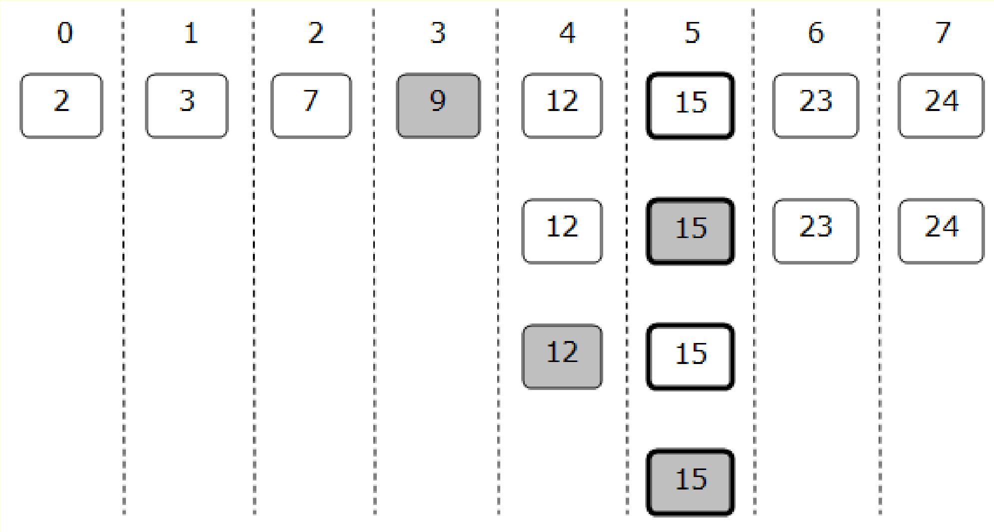
Uniwersum uporządkowane

(algorytm poszukiwań binarnych)

Uniwersum uporządkowane – algorytm poszukiwań binarnych

Idea algorytmu poszukiwań binarnych. Porównujemy liczbę x z m-tym elementem tablicy A, gdzie $m = \left\lfloor \frac{n}{2} \right\rfloor$, jeżeli $A[m] \geq x$, to powtarzamy podobne postępowanie dla tablicy $A[0], A[1], \ldots A[m]$, w przeciwnym przypadku powtarzamy podobne postępowanie dla tablicy $A[m+1], A[m+2], \ldots A[n-1]$. Jeżeli rozmiar aktualnie rozważanej tablicy jest równy 1, to poszukiwanym indeksem jest m.

Przykład. Szukamy indeksu liczby 15 w tablicy $A = [2, 3, 7, 9, 12, 15, 23, 24]$.



Uniwersum uporządkowane – algorytm poszukiwań binarnych

Rozwiązanie. Algorytm poszukiwań binarnych:

```
int BinSearch(int A[n],int n,int x) {
   // wp:  n ≥ 1 i A[0] < A[1] < ... < A[n − 1] i x ∈ A
   int l=0,r=n-1,m;

   while (r-l≥1) { // nz:  A[l] ≤ x ≤ A[r]
      m=(l+r) div 2;

      if (A[m]≥x) r=m;
      else l=m+1;
   }

   return l; // wk:  A[l] = x
}
```

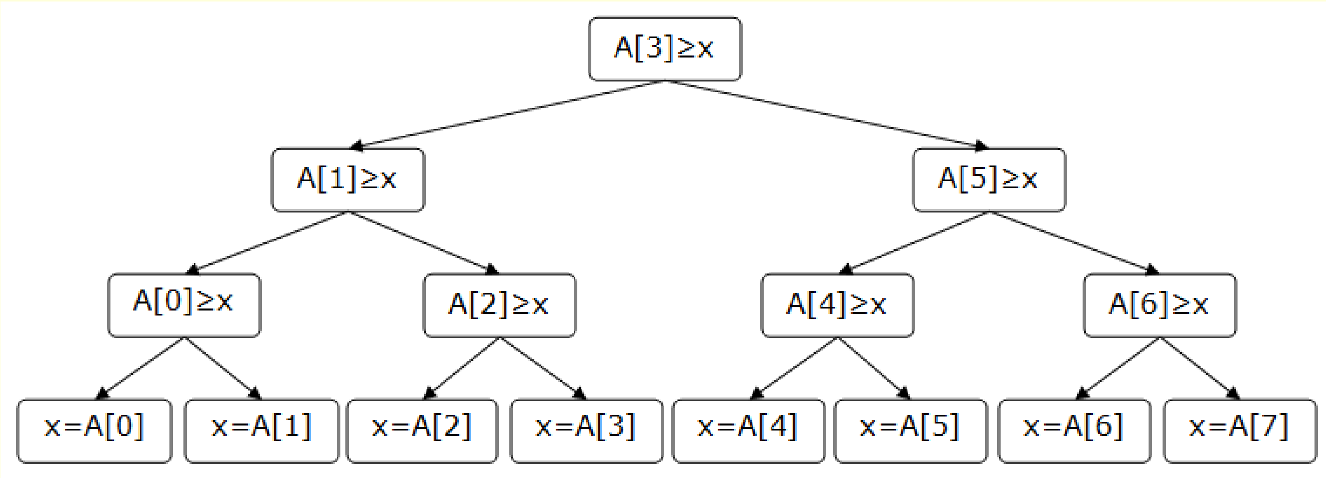
Pytanie. Jaka jest złożoność czasowa algorytmu BinSearch?

Pytanie. Jaka jest złożoność pamięciowa algorytmu BinSearch?

Uniwersum uporządkowane – algorytm poszukiwań binarnych

Twierdzenie. *Algorytm BinSearch jest optymalnym rozwiązaniem problemu wyszukania w uniwersum uporządkowanym.*

Uzasadnienie. Konstruujemy drzewo decyzyjne dla dowolnego algorytmu rozwiązującego problem wyszukania w uniwersum uporządkowanym. Np.



Ponieważ drzewo decyzyjne dla rozważanego problemu zawiera n liści i jest to drzewo binarne, to istnieje w takim drzewie co najmniej jedna ścieżka od korzenia do jednego z wierzchołków zewnętrznych, której długość wynosi co najmniej $\lfloor \lg n \rfloor$. Stąd każdy algorytm, działający przez porównania, dla rozważanego problemu w przypadku pesymistycznym wykona co najmniej $\lfloor \lg n \rfloor$ porównań. Zatem metoda BinSearch jest rozwiązaniem optymalnym.

Uniwersum uporządkowane

(algorytm poszukiwania interpolacyjnego)

Uniwersum uporządkowane – algorytm poszukiwania interpolacyjnego

Idea algorytmu poszukiwania interpolacyjnego. Zauważmy, że w przypadku gdy elementy tablicy A są rozłożone równomiernie na pewnym przedziale zbioru liczb naturalnych, to zachodzi następująca zależność

$$\frac{m-l}{r-l} \quad \approx \quad \frac{x-A[l]}{A[r]-A[l]},$$

stąd punkt podziału dla algorytmu binarnych poszukiwań możemy wyznaczyć dokładniej

$$m \quad = \quad l+\frac{(x-A[l])(r-l)}{A[r]-A[l]}.$$

Reszta postępowania jest identyczna jak w przypadku metody BinSearch.

Fakt. Złożoność pesymistyczną algorytmu wyszukiwania interpolacyjnego można oszacować przez $\Theta(n)$.

Fakt. Złożoność średnią algorytmu wyszukiwania interpolacyjnego można oszacować przez $O(\lg\lg n)$.

Pytanie. Czy złożoność pamięciowa algorytmu poszukiwania interpolacyjnego jest istotnie różna od złożoności pamięciowej algorytmu poszukiwań binarnych?

Problem k-tego co do wielkości

(rozwiązanie „naiwne”)

Problem k-tego co do wielkości – rozwiązanie „naiwne”

Zadanie (problem k-go co do wielkości). Niech A będzie tablicą n różnych liczb naturalnych, gdzie $n \geq k$ i $k \in \mathbb{N}^+$. Podaj algorytm, który wyznaczy k-ty co do wielkości element tablicy A.

Idea algorytmu „naiwnego”. Niech $i = 0$,

- k-krotnie powtórz następujące działanie:
  - wyszukaj element największy wśród elementów $A[i], A[i+1], \ldots, A[n-1]$, niech to będzie element $A[max]$,
  - zamień element $A[max]$ z elementem $A[i]$,
  - zwiększ i o jeden.
- rezultatem jest ostatni z wyszukanych elementów $A[max]$, tj. $A[i-1]$.

Zadanie. Przedstaw działanie algorytmu „naiwnego” dla następujących danych wejściowych:

$$A = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9], \ k = 5.$$

Pytanie. Jaka jest złożoność czasowa algorytmu „naiwnego” w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu „naiwnego” w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu „naiwnego”?

Problem k-tego co do wielkości – rozwiązanie „naiwne”

Rozwiązanie. Algorytm „naiwny”:

```
int FindKth(int A[n],int n,int k) { // wp: n ≥ k i k ∈ ℕ⁺
   int i=0,j,max;

   while (i<k) {
      max=i; // nz: A[0] > A[1] > ... > A[i-1] >
                  każdego elementu A[i], A[i+1], ..., A[n-1]

      for (j=max+1;j<n;j++)
         if (A[j]>A[max]) max=j;

      Swap(A[max],A[i]);
      // A[i] > każdego elementu A[i+1], A[i+2], ..., A[n-1]

      i=i+1; // nz: A[0] > A[1] > ... > A[i-1] >
                  każdego elementu A[i], A[i+1], ..., A[n-1]
   }

   return A[i-1]; // wk: A[i-1] jest k-tym co do wielkości elementem
}
```

Problem k-tego co do wielkości

(algorytm Hoare)

Problem k-tego co do wielkości – algorytm Hoare

Idea algorytmu Hoare. Powtarzaj rekurencyjnie następujący schemat działania:

- wybierz dowolny element aktualnie rozważanego fragmentu tablicy A, tzw. *medianę*, niech będzie to $A[m]$,
- rozdziel elementy aktualnie rozważanego fragmentu tablicy na elementy mniejsze od $A[m]$, tzw. *część młodsza* tablicy, oraz elementy większe od $A[m]$, tzw. *część starsza* tablicy,
- umieść element $A[m]$ w tablicy A tak aby poprawnie rozdzielał część młodszą od starszej,
- jeżeli część starsza liczy dokładnie $k-1$ elementów, to rozwiązaniem jest $A[m]$, zakończ działanie algorytmu, w przeciwnym przypadku
  - jeżeli część starsza liczy więcej niż $k-1$ elementów, to poszukaj rekurencyjnie k-tego co do wielkości elementu tylko w części starszej,
  - w przeciwnym przypadku poszukaj rekurencyjnie ($k-$liczba elementów w części starszej-1)-tego co do wielkości elementu tylko w części młodszej.

Zadanie. Przedstaw działanie algorytmu Hoare dla następujących danych wejściowych:

$$A = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9]\,,\ k = 5.$$

Przyjmij, że medianą jest zawsze pierwszy element aktualnie rozważanego fragmentu tablicy.

Problem k-tego co do wielkości – algorytm Hoare

Rozwiązanie. Algorytm Hoare:

```
int Rozdziel(A[n],int l,int r) {
...  // funkcja dokonuje rozdzielenia fragmentu tablicy A[l], A[l+1], ..., A[r]
        względem mediany wybieranej w ustalony sposób, wynikiem działania
        funkcji jest indeks elementu rozdzielającego po umieszczeniu
        na właściwej pozycji
}

int Hoare(int A[n],int k,int l,int r) { // wp:  n ≥ k i k ∈ ℕ+
   int m;

   m=Rozdziel(A,l,r);

   if (r-m==k-1) return A[m];
   else if (r-m>k-1) return Hoare(A,k,m+1,r);
      else return Hoare(A,k-(r-m)-1,l,m-1);
}
```

Problem k-tego co do wielkości – algorytm Hoare – podział ciągu względem mediany

Idea algorytmu Split. Niech $l = 1$, $r = n - 1$, $m = 0$:

- znaczenie zmiennych indeksujących:
  - zmienna l, wszystkie elementy tablicy na pozycjach $A[1], A[1], \ldots, A[l-1]$ są mniejsze od mediany,
  - zmienna r, wszystkie elementy tablicy na pozycjach $A[r+1], A[r+2], \ldots, A[n-1]$ są większe od mediany,
- dopóki $l < r$ powtórz następujące działanie:
  - dopóki $r \geq 0$ i $A[r] > A[m]$, zmniejsz r o jeden,
  - dopóki $l < r$ i $A[l] < A[m]$, zwiększ l o jeden,
  - jeżeli $l < r$ zamień $A[l]$ z $A[r]$, zmniejsz r o jeden, zwiększ l o jeden,
- zamień $A[m]$ z $A[r]$.

Problem k-tego co do wielkości – algorytm Hoare – podział ciągu względem mediany

Zadanie. Przedstaw działanie algorytmu Split dla następujących danych wejściowych:

$$A = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9] .$$

Pytanie. Jaka jest złożoność czasowa algorytmu Split w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu Split w przypadku pesymistycznym?

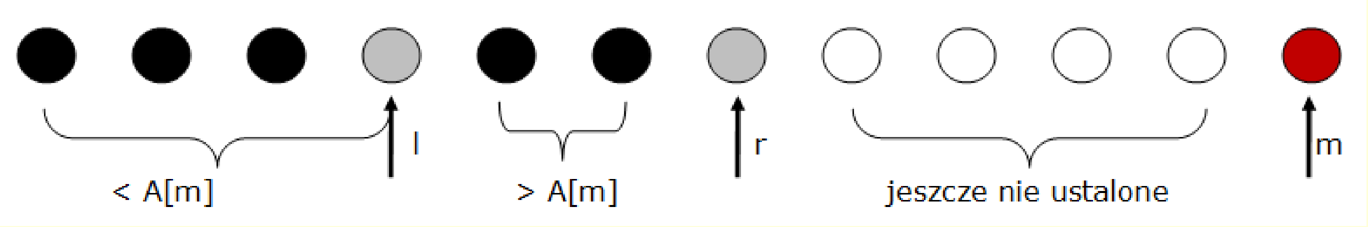
Pytanie. Jaka jest złożoność pamięciowa algorytmu Split?

Zadanie (*). Zaimplementuj algorytm Split w wersji iteracyjnej oraz rekurencyjnej i przedstaw wyniki porównania wydajności obu implementacji dla dostatecznie dużego zbioru danych wejściowych. Oszacuj empirycznie złożoność czasową obu rozwiązań.

Problem k-tego co do wielkości – algorytm Hoare – podział ciągu względem mediany

Idea algorytmu Partition. Niech $l = -1$, $r = 0$, $m = r - 1$:

- znaczenie zmiennych indeksujących:
  - zmienna l, wszystkie elementy tablicy na pozycjach $A[0], A[1], \ldots, A[l]$ są mniejsze od mediany,
  - zmienna r, wszystkie elementy tablicy na pozycjach $A[l+1], A[l+2], \ldots, A[r-1]$ są większe od mediany,
- dopóki $r < m$ powtórz następujące działanie:
  - jeżeli $A[r] < A[m]$ zamień $A[l+1]$ z $A[r]$, zwiększ $l+1$ o jeden,
  - zwiększ r o jeden,
- zamień $A[m]$ z $A[l+1]$.



Problem k-tego co do wielkości – algorytm Hoare – podział ciągu względem mediany

Zadanie. Przedstaw działanie algorytmu Partition dla następujących danych wejściowych:

$$A = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9]\,.$$

Pytanie. Jaka jest złożoność czasowa algorytmu Partition w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu Partition w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu Partition?

Zadanie (**). Przedstaw i zaimplementuj rozszerzoną wersję algorytmu Partition, podziału względem mediany, dowolnej n-elementowej tablicy na trzy rozłączne fragmenty kolejno mniejsze, równe oraz większe od mediany.

Problem k-tego co do wielkości – algorytm Hoare – złożoność

Przypadek pesymistyczny. Elementy n-elementowej tablicy A posortowane są rosnąco, szukamy elementu pierwszego co do wielkości, procedura rozdzielania została zaimplementowana zgodnie z metodą Split, wtedy:

$$W(n) = \begin{cases} 0 & \text{dla } n = 1 \\ n - 1 + W(n-1) & \text{dla } n > 1 \end{cases},$$

czyli

$$\begin{aligned} W(n) &= n - 1 + W(n-1) = n - 1 + n - 2 + W(n-2) = \ldots = \\ &= \ldots = n - 1 + n - 2 + \ldots + 0 = \frac{n(n-1)}{2} = \Theta\left(n^2\right). \end{aligned}$$

Pytanie. Jaki jest układ danych wejściowych w przypadku pesymistycznym, dla algorytmu Hoare, jeżeli procedura rozdzielania została zaimplementowana zgodnie z metodą Partition?

Problem k-tego co do wielkości – algorytm Hoare – złożoność

Przypadek średni. Rozkład elementów n-elementowej tablicy A jest jednorodny, szukamy elementu k-tego co do wielkości, procedura rozdzielania została zaimplementowana zgodnie z metodą Split albo Partition, wtedy:

$$A(n,k) = \begin{cases} 0 & \text{dla } n = 1 \\ n - 1 + \frac{1}{n}\left(\sum_{i=1}^{n-k} A(n-i,k) + \sum_{i=n-k+2}^{n} A(i-1, k-(n-i)-1)\right) & \text{dla } n > 1 \end{cases},$$

czyli

$$A(n,k) = O(n).$$

Pytanie. Jaka jest złożoność pamięciowa algorytmu Hoare w przypadku pesymistycznym a jaka w przypadku średnim?

Zadanie (**). Czy i jak zmieni się złożoność algorytmu Hoare, jeżeli za operację dominującą przyjmiemy przestawianie elementów tablicy wejściowej A?

Problem k-tego co do wielkości

(algorytm Bluma-Floyda-Pratta-Rivesta-Trajana)

Problem k-tego co do wielkości – algorytm Bluma-Floyda-Pratta-Rivesta-Trajana

Idea algorytmu BFPRT. Powtarzaj rekurencyjnie następujący schemat działania, gdzie n jest rozmiarem aktualnie rozważanego fragmentu tablicy A:

- jeżeli $n \leq 5$, to posortuj fragment tablicy i wybierz element $(n - (k - 1))$-ty,
- jeżeli $n > 5$, to:
  - podziel aktualnie rozważany fragment tablicy A na kolejne „piątki elementów”,
  - rekurencyjnie wyszukaj $A[m] = \lceil m/2 \rceil$-gi element z median rozważanych piątek, gdzie m jest liczbą piątek,
  - rekurencyjnie wykonaj działanie zgodne z algorytmem Hoare dla elementu dzielącego $A[m]$.

Wniosek. Rekurencyjny schemat doboru mediany gwarantuje na każdym kroku działania algorytmu dla aktualnie rozważanego fragmentu tablicy A rozmiaru d, że co najmniej $\frac{d}{4}$ elementy są odpowiednio mniejsze i większe od mediany $A[m]$.

Problem k-tego co do wielkości – algorytm Bluma-Floyda-Pratta-Rivesta-Trajana

Fakt. Złożoność czasowa algorytmu BFPRT w przypadku średnim jest co najwyżej liniowa.

Fakt. Złożoność czasowa algorytmu BFPRT w przypadku pesymistycznym jest co najwyżej liniowa.

Fakt. Złożoność czasowa w przypadku pesymistycznym algorytmu BFPRT przestaje być rzędu liniowego, gdy zamiast piątek elementów będziemy analizowali np. trójki elementów.

Fakt. Złożoność czasowa w przypadku pesymistycznym algorytmu BFPRT przestaje być rzędu liniowego, gdy zamiast piątek elementów będziemy analizowali l-tki, dla l dostatecznie bliskiego n.

Zadanie (***). Udowodnij, że złożoność czasowa w przypadku pesymistycznym algorytmu BFPRT w wariancie trójek elementów jest rzędu większego niż liniowy. Oszacuj ten rząd.