Algorytmy i struktury danych

Wykład III – wyszukiwanie c.d.

Paweł Rembelski

PJWSTK

16 października 2009

1. Podział względem mediany
   - Algorytm Partition
   - Algorytm Split – szkic
2. Wyszukanie elementu k-tego co do wielkości
   - Algorytm Hoare'a
   - Algorytm 5-tek – szkic
3. Wyszukanie wskazanego elementu z uporządkowaniem
   - Algorytm Skoki co k
   - Algorytm wyszukiwania binarnego
   - Algorytm wyszukiwania interpolacyjnego – szkic

Dla uproszczenia w poniższym wykładzie rozważamy algorytmy wyszukiwania w zbiorze liczb naturalnych z relacją $\leq$. Należy jednak pamiętać, że opisane rozwiązania są poprawne w dowolnym uniwersum U z relacją porządku liniowego $\preceq$.

Podział względem mediany

Problem, struktura i specyfikacja algorytmu

Problem

Niech T będzie niepustym n-elementowym wektorem liczb naturalnych. Podać algorytm $Alg(T, n)$ dzielący elementy wektora T tak, że $\forall (0 \leq i < idx)(T[i] < T[idx])$ oraz $\forall (idx < j \leq n-1)(T[i] > T[idx])$, dla wybranego $0 \leq idx < n$. Element $T[idx]$ będziemy nazywać *medianą podziału* a rezultat podziału *wektorem podzielonym dla wektora* T.

Struktura dla algorytmu

Struktura dla algorytmu Alg: standardowa struktura liczb naturalnych.

Specyfikacja algorytmu

Specyfikację algorytmu Alg stanowi para $\langle WP, WK \rangle$, gdzie warunki początkowy i końcowy są postaci kolejno:

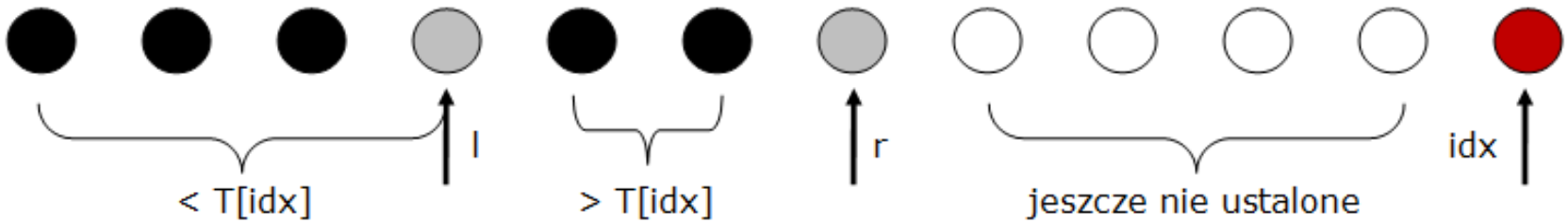
- WP : T jest niepustym wektorem różnych liczb naturalnych, $n \in \mathbb{N}^+$, $|T| = n$,
- WK : $Alg(T, n) = idx$, gdzie idx jest finalną pozycją mediany podziału wektora T w wektorze podzielonym T'.

Podział względem mediany

Algorytm Partition

Pomysł. Niech $l = -1$, $r = 0$, $idx = n - 1$:

- znaczenie zmiennych indeksujących:
  - zmienna l, wszystkie elementy wektora T na pozycjach $0, 1, \ldots, l$ są mniejsze od mediany podziału (tzw. *część młodsza* podziału),
  - zmienna r, wszystkie elementy wektora T na pozycjach $l + 1, l + 2, \ldots, r - 1$ są większe od mediany podziału (tzw. *część starsza* podziału),
- dopóki $r < idx$ powtórz następujące działanie:
  - jeżeli $T[r] < T[idx]$
    - jeżeli $r > l + 1$ zamień $T[l + 1]$ z $T[r]$,
    - zwiększ $l + 1$ o jeden,
  - zwiększ r o jeden,
- jeżeli $l + 1 < idx$ zamień $T[idx]$ z $T[l + 1]$ i przypisz $idx := l + 1$



Zadanie. Przedstaw działanie algorytmu Partition dla wektora wejściowego postaci $T = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9]$.

Rozwiązanie problemu – **algorytm Partition**:

```
1  int Partition(int T[], int n) {
2    int l:=-1, r:=0, idx:=n-1;
3
4    while (r<idx) {
5      if (T[r]<T[idx]) {
6        if (r>l+1)
7          SWAP(T,l+1,r);
8
9        l:=l+1;
10     }
11
12     r:=r+1;
13   }
14   if (l+1<idx) { SWAP(T,l+1,idx); idx:=l+1; }
15
16   return idx;
17 }
```

T jest niepustym wektorem różnych liczb naturalnych, $n \in \mathbb{N}^+$, $|T| = n$,

$WK : Alg(T, n) = idx$, gdzie idx jest finalną pozycją mediany podziału wektora T w wektorze podzielonym T'.

Poprawność algorytmu Partition

- poprawność częściowa: niezmiennikiem pętli w rozważanym algorytmie jest formuła

$$\forall (0 \leq i \leq l) (T[i] < T[idx]), \forall (l+1 \leq j < r) (T[j] > T[idx]).$$

Po wyjściu z pętli prawdą jest, że $r \geq idx$ (a dokładniej $r = idx = n-1$), stąd

$$\forall (0 \leq i \leq l) (T[i] < T[idx]), \forall (l+1 \leq j < idx) (T[j] > T[idx]).$$

Ostatecznie po wykonaniu instrukcji warunkowej z wiersza 14 zachodzi

$$\forall (0 \leq i < idx) (T[i] < T[idx]), \forall (idx < j \leq n-1) (T[j] > T[idx]).$$

- warunek stopu: z warunku początkowego $n \in \mathbb{N}^+$, stąd po inicjalizacji zmiennych $idx := n-1$ zachodzi $idx \in \mathbb{N}^+$. Zmienna r inicjalizowana wartością 0 jest inkrementowana z każdą iteracja pętli o 1, czyli po idx iteracjach pętli $r = idx$, co kończy działanie algorytmu.

Złożoność czasowa algorytmu Partition – wariant I

- operacja dominująca: porównanie elementów rozważanego uniwersum,
- złożoność czasowa: $T(n) = n - 1 = \Theta(n)$.

Złożoność czasowa algorytmu Partition – wariant II

- operacja dominująca: przestawienie elementów rozważanego uniwersum,
- średnia złożoność czasowa: $A(n) = \Theta\left(\frac{n}{2}\right)$, zakładając $Pr(SWAP(T, l+1, r)) = \frac{1}{2}$, dla każdego $0 \leq r < n-1$,
- pesymistyczna złożoność czasowa: $W(n) = (n-2) + 1 = n - 1 = \Theta(n)$.

Zadanie. Podaj przykłady wektorów wejściowych długości n, dla których realizacja algorytmu Partition będzie odpowiadała kolejno przypadkowi optymistycznemu oraz pesymistycznemu.

Złożoność pamięciowa algorytmu Partition

Złożoność pamięciową algorytmu Partition można oszacować przez $\Theta(1)$.

Podział względem mediany

Algorytm Split – szkic

Pomysł. Niech $l = 1$, $r = n - 1$, $idx = 0$:

- znaczenie zmiennych indeksujących:
  - zmienna l, wszystkie elementy wektora T na pozycjach $1, 2, \ldots, l-1$ są mniejsze od mediany podziału (tzw. część młodsza podziału),
  - zmienna r, wszystkie elementy wektora T na pozycjach $r+1, r+2, \ldots, n-1$ są większe od mediany podziału (tzw. część starsza podziału),
- dopóki $l \leq r$ powtórz następujące działanie:
  - dopóki $l \leq r$ i $T[r] > T[idx]$, zmniejsz r o jeden,
  - dopóki $l \leq r$ i $T[l] < T[idx]$, zwiększ l o jeden,
  - jeżeli $l < r$ zamień $T[l]$ z $T[r]$, zmniejsz r o jeden, zwiększ l o jeden,
- jeżeli $r > 0$ zamień $T[idx]$ z $T[r]$ i przypisz $idx := r$

Zadanie. Przedstaw działanie algorytmu Split dla wektora wejściowego postaci $T = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9]$.

Złożoność czasowa algorytmu Split – wariant I

- operacja dominująca: porównanie elementów rozważanego uniwersum,
- złożoność czasowa: $T(n) = n - 1 = \Theta(n)$.

Złożoność czasowa algorytmu Split – wariant II

- operacja dominująca: przestawienie elementów wektora T,
- pesymistyczna złożoność czasowa: $W(n) = \left\lfloor \frac{n-1}{2} \right\rfloor + 1 = \Theta\left(\frac{n}{2}\right)$.

Zadanie. Podaj dokładne oszacowanie złożoności średniej algorytmy Split mierzonej liczbą przestawień elementów wektora wejściowego.

Zadanie. Podaj przykłady wektorów wejściowych długości, dla których realizacja algorytmu Split będzie odpowiadała przypadkowi kolejno optymistycznemu oraz pesymistycznemu.

Złożoność pamięciowa algorytmu Split

Złożoność pamięciową algorytmu Partition można oszacować przez $\Theta(1)$.

Wyszukanie elementu k-tego co do wielkości

Problem, struktura i specyfikacja algorytmu

Problem

Podać algorytm $Alg(T, n)$ znajdujący indeks elementu k-tego co do wielkości zapisanego w niepustym, n-elementowym wektorze różnych liczb naturalnych T.

Struktura dla algorytmu

Struktura dla algorytmu Alg: standardowa struktura liczb naturalnych.

Specyfikacja algorytmu

Specyfikację algorytmu Alg stanowi para $\langle WP, WK \rangle$, gdzie warunki początkowy i końcowy są postaci kolejno:

- WP : T jest niepustym wektorem różnych liczb naturalnych, $n, k \in \mathbb{N}^+$, $n \geq k$ $|T| = n$,
- WK : $Alg(T, n, x) = idx$, gdzie idx jest indeksem elementu k-tego co do wielkości zapisanego w wektorze T.

Wyszukanie elementu k-tego co do wielkości

Algorytmy Hoare'a

Pytanie. Czy k-krotne wykonanie algorytm FindMax z usunięciem $k-1$ kolejnych elementów maksymalnych jest poprawnym rozwiązaniem dla problemu wyszukania w uniwersum uporządkowanym? Jeżeli tak, to:

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

Pomysł. Powtarzaj rekurencyjnie następujący schemat działania z **ustaloną procedurą podziału** względem mediany:

- podziel stosując procedurę podziału elementy aktualnie rozważanego fragmentu wektora względem mediany $T[idx]$ na część młodszą i część starszą,
- jeżeli część starsza po podziale liczy dokładnie $k-1$ elementów, to rozwiązaniem jest $T[idx]$, 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 wektora wejściowego postaci $T = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9]$ i $k = 5$.

Rozwiązanie problemu – **algorytm Hoare'a**:

```
1  int Hoare(int T[], int n, int k) {
2    int idx;
3
4    idx:=Rozdziel(T,n);
5
6    if (n-idx=k) return idx;
7    else
8      if (n-idx>k)
9        if (n-idx-1>0) return idx+1+Hoare(T[idx+1...n-1],n-idx-1,k);
10     else
11       if (idx>0) return Hoare(T,[0...idx-1],idx,k-(n-idx));
12 }
```

WP : T jest niepustym wektorem różnych liczb naturalnych, $n, k \in \mathbb{N}^+$, $n \geq k$ $|T| = n$

procedura Split albo Partition

WK : $Hoare\,(T, n, x) = idxK$, gdzie $idxK$ jest indeksem elementu k-tego co do wielkości zapisanego w wektorze T.

WK : $Hoare\,(T, n, x) = idxK$, gdzie $idxK$ jest indeksem elementu k-tego co do wielkości zapisanego w wektorze T.

Poprawność algorytmu Hoare'a

- poprawność częściowa: dla $n = 1$ poprawność częściowa algorytmu Hoare'a wynika z warunku początkowego $n, k \in \mathbb{N}^+$, $n \geq k$, tj. $k = 1$ i poprawności częściowej algorytmu podziału Rozdziel (np. metoda Partition). Wtedy $n - idx = 1$ i $T[idx]$ jest 1-szym co do wielkości elementem aktualnie rozważanego fragmentu wektora wejściowego. Dla $n > 1$ i po zastosowaniu algorytmu Rozdziel jeżeli $n - idx = k$, to $T[idx]$ jest k-tym co do wielkości elementem aktualnie rozważanego fragmentu wektora wejściowego, co kończy zejście rekurencyjne algorytmu. W przeciwnym przypadku jeżeli $n - idx > k$ to w wierszu 9 rekurencyjnie szukamy indeksu (powiększonego o wartość idx) elementu k-tego co do wielkości w wektorze $T[idx + 1 \ldots n - 1]$ długości $n - idx - 1$ (czyli w tzw. części starszej podziału) bądź dla $n - idx > k$ w wierszu 11 rekurencyjnie szukamy indeksu elementu $(k - n - idx)$-tego co do wielkości w wektorze $T[0 \ldots idx - 1]$ długości idx (czyli w tzw. części młodszej podziału).
- warunek stopu: ponieważ $n \in \mathbb{N}^+$ i ciąg kolejnych rozmiarów aktualnie rozważanego fragmentu wektora wejściowego jest ciągiem ściśle malejącym, to po co najwyżej n wywołaniach rekurencyjnych algorytmu Hoare'a przestają być prawdziwe warunki $(n - idx - 1 > 0)$ oraz $(idx > 0)$ z wierszy kolejno 9 i 11 , co kończy zejście rekurencyjne rozważanej procedury.

Złożoność algorytmu Hoare'a

- operacja dominująca: porównanie elementów rozważanego uniwersum,
- średnia złożoność czasowa: zakładamy, że rozkład elementów n-elementowego wektora T jest losowy, szukamy elementu k-tego co do wielkości, procedura rozdzielania została zaimplementowana zgodnie z metodą Partition albo Split, wtedy $A(n,k)$ wynosi:

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

czyli*

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

---

* rozwiązanie równania w Dodatku A.

Złożoność algorytmu Hoare'a c.d.

- pesymistyczna złożoność czasowa: zakładamy, że elementy n-elementowego wektora T są uporządkowane rosnąco, szukamy elementu 1-szego 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

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

- złożoność pamięciowa: $O(n)$ z uwzględnieniem kosztów rekursji ($O(1)$ w przeciwnym przypadku).

---

* jaki jest układ danych wejściowych dla przypadku pesymistycznego wykonania algorytmu Hoare, jeżeli procedura rozdzielania została zaimplementowana zgodnie z metodą Partition?

Wyszukanie elementu k-tego co do wielkości

Algorytmy 5-tek – szkic

Pomysł. Powtarzaj rekurencyjnie następujący schemat działania, gdzie n jest rozmiarem aktualnie rozważanego fragmentu wektora T:

- jeżeli $n \leq 5$, to posortuj fragment wektora i wybierz element $(n - (k-1))$-ty,
- jeżeli $n > 5$, to:
  - podziel aktualnie rozważany fragment wektora T na kolejne 5-tki elementów, niech k będzie liczbą otrzymanych piątek,
  - rekurencyjnie wyszukaj $\lceil k/2 \rceil$-gi element z median rozważanych piątek (tj. medianę median 5-tek), niech będzie to $T[idx]$,
  - rekurencyjnie wykonaj działanie analogiczne do kroku algorytmu Hoare'a i elementu dzielącego $T[idx]$.

Wniosek

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

Fakt

Złożoność czasowa algorytmu 5-tek w przypadku średnim jest co najwyżej liniowa.

Fakt

Złożoność czasowa algorytmu 5-tek w przypadku pesymistycznym jest co najwyżej liniowa.

Fakt

Złożoność czasowa w przypadku pesymistycznym algorytmu 5-tek przestaje być rzędu liniowego, gdy zamiast piątek elementów będziemy analizowali jednostki mniejszego rozmiaru, np. 3-ki elementów.

Fakt

Złożoność czasowa w przypadku pesymistycznym algorytmu 5-tek przestaje być rzędu liniowego, gdy zamiast piątek elementów będziemy analizowali k-ki, dla k dostatecznie bliskiego n.

Wyszukanie wskazanego elementu z uporządkowaniem

Problem, struktura i specyfikacja algorytmu

Problem

Podać algorytm $Alg\,(T, n, x)$ znajdujący indeks elementu x zapisanego w niepustym, **uporządkowanym** n-elementowym wektorze różnych liczb naturalnych T.

Struktura dla algorytmu

Struktura dla algorytmu Alg: standardowa struktura liczb naturalnych.

Specyfikacja algorytmu

Specyfikację algorytmu Alg stanowi para $\langle WP, WK \rangle$, gdzie warunki początkowy i końcowy są postaci kolejno:

- WP : T jest niepustym, uporządkowanym wektorem różnych liczb naturalnych, $n \in \mathbb{N}^+$, $|T| = n$, $\exists!\,(0 \leq i < n)\,(T\,[i] = x)$,
- WK : $Alg\,(T, n, x) = idx$, gdzie $T\,[idx] = x$.

Wyszukanie wskazanego elementu

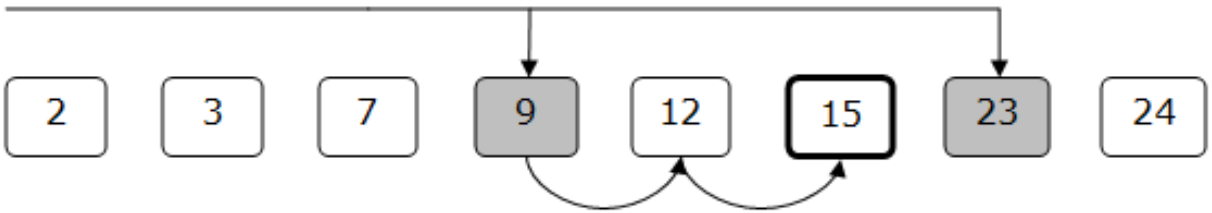
Algorytm Skoki co k

Pytanie. Czy algorytm sekwencyjny FindIndex jest poprawnym rozwiązaniem dla problemu wyszukania w uniwersum uporządkowanym? Jeżeli tak, to:

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

Pomysł. Porównujemy liczbę x z co k-tym elementem wektora T poczynając od elementu k-tego, tj. $T[k], T[2k], T[3k], \ldots$. Proces przerywamy wtedy, gdy $T[ik] > x$, dla pewnego $i \in \mathbb{N}$. Sekwencyjnie przeglądamy $k-1$ elementów $T[ik-k], T[ik-(k-1)], \ldots, T[ik-1]$.

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



Rozwiązanie problemu – **algorytm JumpSearch**:

```
1  int JumpSearch(int T[], int n, int k, int x) {
2    int i:=0;
3
4    while ((i<n) AND (x≥A[i])) i:=i+k;
5
6    if (i≥n)
7      return Find(T[i-k...n-1],n-k);
8    else
9      return Find(T[i-k...i-1],k);
10 }
```

T jest niepustym uporządkowanym wektorem różnych liczb naturalnych, $n \in \mathbb{N}^+$, $|T| = n$, $0 < k \leq n$, $\exists! (0 \leq i < n)(T[i] = x)$,

$WK : JumpSearch(T, n, k, x) = idx$, gdzie $T[idx] = x$

$WK : JumpSearch(T, n, k, x) = idx$, gdzie $T[idx] = x$

Poprawność algorytmu JumpSearch

- poprawność częściowa: tuż po inicjalizacji zmiennej i w wierszu 2 prawdą jest, że $x \geq T[0], T[1], \ldots, T[i-k]$, dla dowolnego k. Niezmiennikiem pętli jest formuła $NZ : x \geq T[0], T[1], \ldots, T[i-k]$. Po wykonaniu instrukcji dozoru pętli mamy $x \geq T[0], T[1], \ldots, T[i]$. Zatem po wykonaniu instrukcji $i := i + k$ zachodzi $x \geq T[0], T[1], \ldots, T[i-k]$ – **odtworzenie niezmiennika** NZ. Po zakończeniu pętli iteracyjnej $i \geq n$ albo $x < T[i]$ oraz $x \geq T[0], T[1], \ldots, T[i-k]$. Załóżmy dalej, że:
  - $i \geq n$ i $x \geq T[0], T[1], \ldots, T[i-k]$, stąd x jest elementem wektora $T[i-k \ldots n-1]$,
  - $x < T[i]$ i $x \geq T[0], T[1], \ldots, T[i-k]$, stąd x jest elementem wektora $T[i-k \ldots i-1]$,

  Dalej indeks owego elementu wyszukujemy w wektorze $T[i-k \ldots n-1]$ albo $T[i-k \ldots i-1]$ stosując algorytm Find, którego poprawność została wykazana na Wykładzie II.
- warunek stopu: z warunku początkowego $0 < k \leq n$ gdzie $n \in \mathbb{N}^+$. Zmienna i inicjalizowana wartością 0 jest inkrementowana z każdą iteracja pętli o k, stąd po co najwyżej $\lceil \frac{n}{k} \rceil$ iteracjach pętli $i \geq n$, co kończy działanie właściwej części algorytmu. Warunek stopu dla procedury pomocniczej Find został wykazanych na Wykładzie II.

Złożoność algorytmu JumpSearch

- operacja dominująca: porównanie elementów rozważanego uniwersum,
- średnia złożoność czasowa: $A(n,k) = \frac{\left\lceil \frac{n}{k} \right\rceil}{2} + \frac{k-1}{2} = \Theta\left(\frac{n}{k} + k\right)$, zakładając $Pr(T[i] = x) = \frac{1}{n}$, dla każdego $0 \leq i < n$,
- pesymistyczna złożoność czasowa: $W(n,k) = \left\lceil \frac{n}{k} \right\rceil + k - 1 = \Theta\left(\frac{n}{k} + k\right)$,
- złożoność pamięciowa: $O(1)$.

Fakt

Optymalny rozmiar skoku dla algorytmu JumpSearch jest rzędu $\sqrt{n}$.

Uzasadnienie. Szukamy minimum funkcji zmiennej k postaci $f(k) = \frac{n}{k} + k - 1$, gdzie n jest pewną stałą. Ponieważ $f'(k) = -\frac{n}{k^2} + 1$, $f'(k) = 0 \Rightarrow k = \sqrt{n}$ i $f''\left(\sqrt{n}\right) > 0$, to $k = \sqrt{n}$ jest poszukiwanym minimum.

Wniosek

Złożoność czasowa algorytmu Skoki co k jest rzędu $\Theta\left(\sqrt{n}\right)$, dla danych rozmiaru n i skoku $k = \left\lfloor \sqrt{n} \right\rfloor$.

Wyszukanie wskazanego elementu

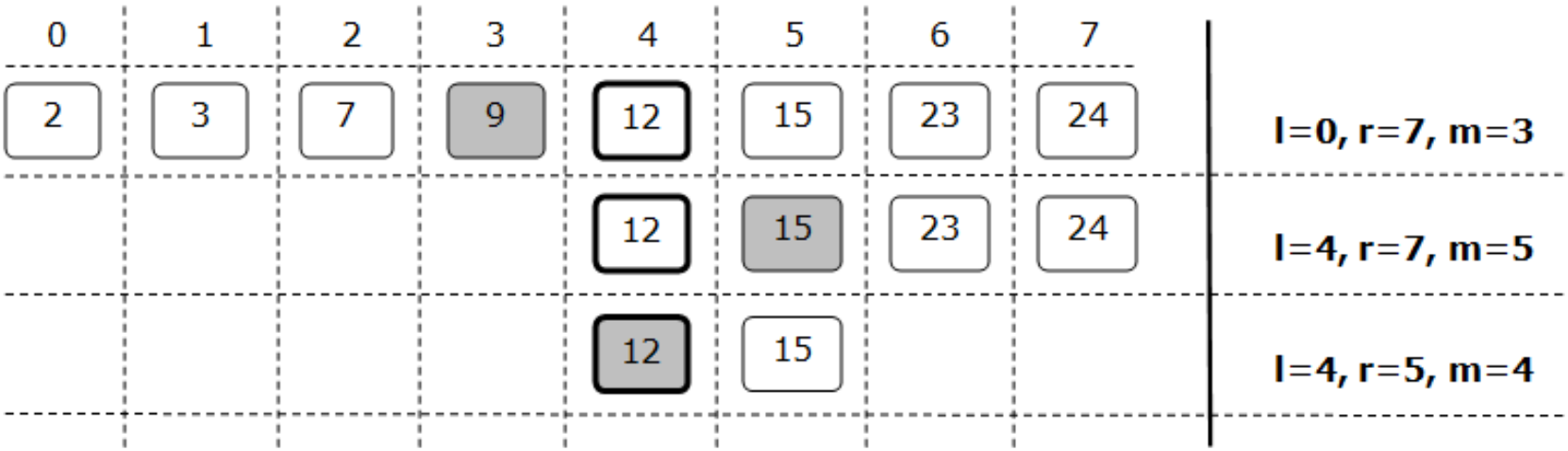
Algorytm wyszukiwania binarnego

Pomysł. Porównujemy liczbę x z m-tym elementem wektora T, gdzie $m = \left\lfloor \frac{n}{2} \right\rfloor$, jeżeli:

- $T[m] = x$, to przerywamy działanie algorytmu dla $Alg(T, n, x) = m$,
- $T[m] < x$, to powtarzamy podobne postępowanie dla fragmentu wektora $T[m+1 \ldots n-1]$ i $n := n - (m+1)$,
- $T[m] > x$, to powtarzamy podobne postępowanie dla fragmentu wektora $T[0 \ldots m-1]$ i $n := m$,

Algorytm kończy działanie, gdy rozmiar aktualnie rozważanego wektora T jest równy 1. Wtedy $Alg(T, n, x) = m$.

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



Rozwiązanie problemu – **algorytm BinSearch**:

```
1  int BinSearch(int T[], int n, int x) {<-------| T jest niepustym, uporządkowanym
                wektorem różnych liczb naturalnych, n ∈ ℕ⁺, |T| = n, ∃!(0 ≤ i < n)(T[i] = x),
2    int l:=0, r:=n-1, idx:=(l+r)/2;
3
4    while (T[idx]≠x) {
5      if (Tab[idx]<x)
6        l:=idx+1;
7      else
8        r:=idx-1;
9
10     idx:=(l+r)/2;
11   }
12
13   return idx;<-----------------------------| WK : BinSearch(T, n, x) = idx, gdzie T[idx] = x
14 }
```

Poprawność algorytmu FindBin

- poprawność częściowa: wynika bezpośrednio z warunku początkowego i zaprzeczenia dozoru pętli iteracyjnej $(\neg T[idx] \neq x) \Rightarrow T[idx] = x$, stąd prawdziwy jest warunek końcowy $BinSearch(T, n, x) = idx,\ T[idx] = x$,
- warunek stopu: z warunku początkowego $\exists!\,(0 \leq i < n)\,(T[i] = x)$ i $n \in \mathbb{N}$. Tuż po inicjalizacji zmiennych prawdą jest, że $T[l] \leq x \leq T[r]$ oraz $0 \leq r - l \leq n$. Po pierwszej iteracji pętli (z uwzględnieniem instrukcji warunkowej) zachodzi $T[l] \leq x \leq T[r]$ oraz $0 \leq r - l \leq \left\lfloor \frac{n}{2} \right\rfloor$. Po drugiej $T[l] \leq x \leq T[r]$ oraz $r - l \leq \left\lfloor \frac{n}{2^2} \right\rfloor$ i analogicznie po i-tej iteracji prawdziwy jest warunek $T[l] \leq x \leq T[r]$ oraz $r - l \leq \left\lfloor \frac{n}{2^i} \right\rfloor$. Stąd po co najwyżej $\lg n + 1$ iteracjach pętli spełniona będzie zależność $0 \leq r - l \leq \left\lfloor \frac{n}{2^{\lg n + 1}} \right\rfloor$, czyli $0 \leq r - l \leq \left\lfloor \frac{n}{2n} \right\rfloor$ i ostatecznie$0 \leq r - l \leq \left\lfloor \frac{1}{2} \right\rfloor$. Zatem $0 \leq r - l \leq 0$ i z tego $l = r$ oraz $T[l] = x = T[r]$, co kończy działanie algorytmu.

Złożoność algorytmu BinSearch

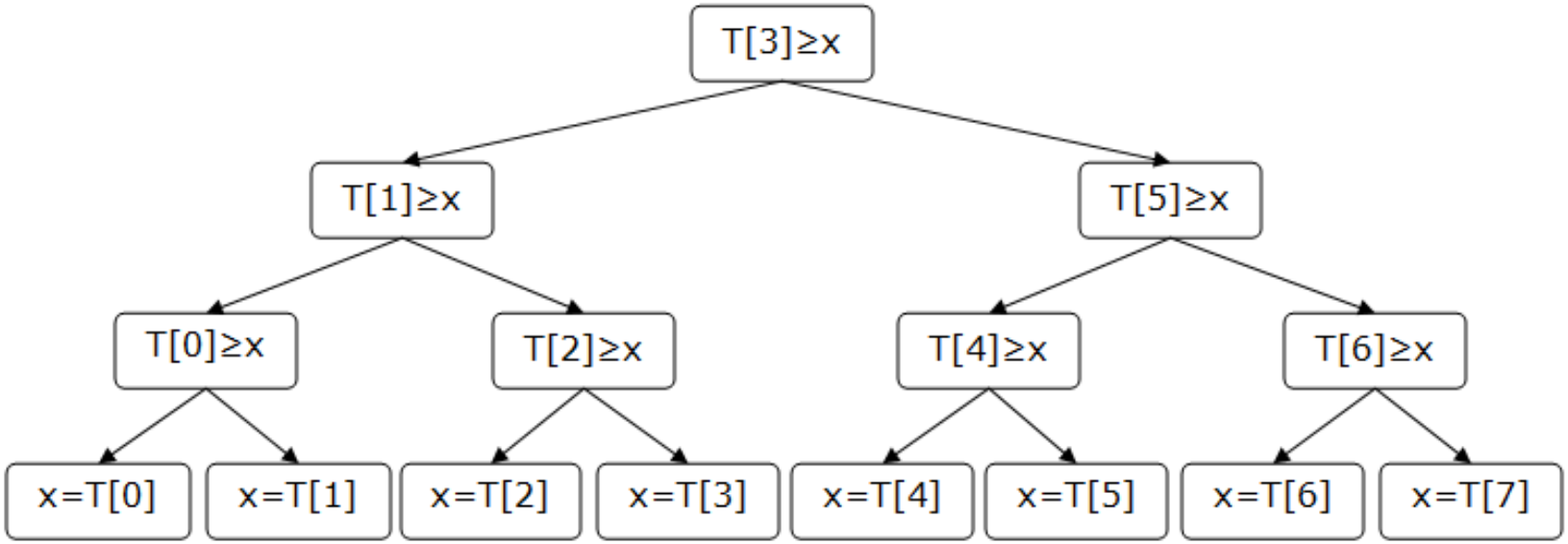
- operacja dominująca: porównanie elementów rozważanego uniwersum,
- średnia złożoność czasowa: $A(n) = \Theta(\lg n)$, zakładając $Pr(T[i] = x) = \frac{1}{n}$, dla każdego $0 \leq i < n$,
- pesymistyczna złożoność czasowa: $W(n) = 2\lfloor \lg n \rfloor + 1 = \Theta(\lg n)$,
- złożoność pamięciowa: $O(1)$.

Zadanie. Podaj dokładne oszacowanie średniej złożoności czasowej algorytmu BinSearch.

Twierdzenie

Algorytm BinSearch jest optymalnym w sensie średnim i pesymistycznym rozwiązaniem postawionego problemu.

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



Ponieważ drzewo decyzyjne 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 FindBin jest rozwiązaniem optymalnym.

Wyszukanie wskazanego elementu

Algorytm wyszukiwania interpolacyjnego – szkic

Pomysł. Zauważmy, że w przypadku gdy elementy wektora T są rozłożone równomiernie na pewnym przedziale zbioru liczb naturalnych, to zachodzi następująca zależność

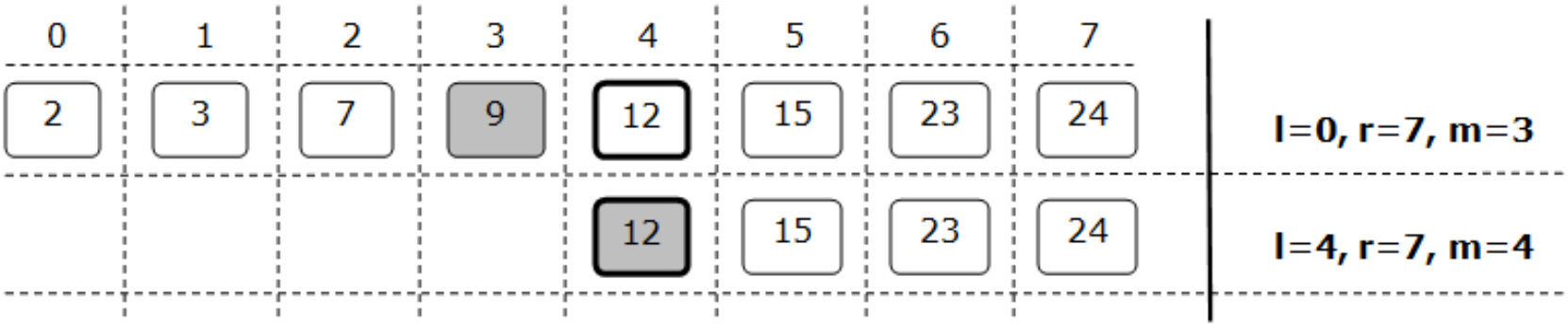
$$\frac{idx - l}{r - l} \approx \frac{x - T[l]}{T[r] - T[l]},$$

stąd punkt podziału dla algorytmu wyszukiwania binarnego możemy wyznaczyć dokładniej

$$idx = l + \frac{(x - T[l])(r - l)}{T[r] - T[l]}.$$

Dalej postępujemy analogicznie jak w algorytmie BinSerach.

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



Fakt

Średnią złożoność czasową algorytmu wyszukiwania interpolacyjnego można oszacować przez $O(\lg \lg n)$.

Fakt

Pesymistyczną złożoność czasową algorytmu wyszukiwania interpolacyjnego można oszacować przez $\Theta(n)$.

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

Dodatek A – rozwiązanie równania $A(Hoare, n, k)$

Twierdzimy, że $A(n,k) = O(n)$, dla

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

Dowód przez **indukcję** ze względu na n, gdzie $n \in \mathbb{N}^+$ oraz $c > 0$:

- baza indukcji: dla $n = 1$ zachodzi $A(n,k) = 0 \leq cn$,
- założenie indukcyjne: dla $1 \leq m < n$ zachodzi $A(m,k) \leq cm$,
- teza indukcyjna: dla $n > 1$ zachodzi $A(n,k) \leq cn$.

Dowód tezy indukcyjnej

Niech $\alpha = \sum_{i=1}^{n-k} A(n-i,k) + \sum_{i=n-k+2}^{n} A(i-1, k-(n-i)-1)$, wtedy z założenia indukcyjnego

$$\begin{aligned}
\alpha &\leq \sum_{i=1}^{n-k} c(n-i) + \sum_{i=n-k+2}^{n} c(i-1) \\
&= c\left(\sum_{i=1}^{n-k}(n-i) + \sum_{i=n-k+2}^{n}(i-1)\right) \\
&\leq c\left(\sum_{i=1}^{\lceil \frac{n}{2} \rceil}(n-i) + \sum_{i=\lfloor \frac{n}{2} \rfloor+1}^{n}(i-1)\right) \\
&\leq c\left(2 \cdot \frac{n(n-1)}{2} - 2 \cdot \frac{\frac{n}{2}\left(\frac{n}{2}-1\right)}{2}\right) = \\
&= c\left(n(n-1) - \frac{n}{2}\left(\frac{n}{2}-1\right)\right) = \frac{3c}{4}n^2 - \frac{c}{2}n.
\end{aligned}$$

Dowód tezy indukcyjnej c.d.

Stąd dla $n > 1$

$$\begin{aligned} A(n,k) &= n - 1 + \frac{1}{n}\alpha \\ &\leq n - 1 + \frac{1}{n}\left(\frac{3c}{4}n^2 - \frac{c}{2}n\right) \\ &= n + \frac{3c}{4}n - \frac{c}{2} - 1 \\ &\leq n + \frac{3c}{4}n = \left(1 + \frac{3c}{4}\right)n, \end{aligned}$$

czyli $A(n,k) \leq cn$, dla $c \geq 4$. Zatem $A(n,k) = O(n)$, co kończy dowód.

Literatura

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Wprowadzenie do algorytmów*, WNT 2004.
2. L. Banachowski, K. Diks, W. Rytter, *Algorytmy i struktury danych*, WNT 1996.
3. A. V. Aho, J. E. Hopcroft, J. D. Ullman, *Algorytmy i struktury danych*, Helion 2003.
4. A. Dańko, T. L. Lee, G. Mirkowska, P. Rembelski, A. Smyk, M. Sydow, *Algorytmy i struktury danych – zadania*, PJWSTK 2006.
5. R. Sedgewick, *Algorytmy w C++*, RM 1999.
6. N. Wirth, *Algorytmy + struktury danych = programy*, WNT 1999.
7. A. Drozdek, D. L. Simon, *Struktury danych w języku C*, WNT 1996.
8. D. Harel, *Rzecz o istocie informatyki – Algorytmika*, WNT 2001.

Zadania ćwiczeniowe

1. Zaimplementuj algorytm Partition.
2. Zaimplementuj algorytm Split.
3. Przeprowadź doświadczalnie analizę porównawczą efektywności algorytmów Partition i Split względem liczby następujących operacji dominujących:
    1. operacja porównania elementów wektora wejściowego,
    2. operacja przestawienia elementów wektora wejściowego..
4. Zaimplementuj algorytm Hoare'a z wybraną procedurą podziału.
5. Zaimplementuj algorytm 5-tek z wybraną procedurą podziału.
6. Udowodnij, że złożoność czasowa w przypadku pesymistycznym algorytmu 5-tek przestaje być rzędu liniowego, gdy zamiast piątek elementów będziemy analizowali 3-ki elementów.
7. Przeprowadź doświadczalnie analizę porównawczą efektywności algorytmów Hoare'a i 5-tek dla wybranej procedury podziału.
8. Zaimplementuj algorytm Skoki co k.
9. Zaimplementuj algorytm wyszukiwania binarnego.
10. Zaimplementuj algorytm wyszukiwania interpolacyjnego.
11. Przeprowadź doświadczalnie analizę porównawczą efektywności algorytmów JumpSearch i BinSearch.
12. Zaproponuj algorytm rekurencyjny (oparty na schemacie „dziel i zwyciężaj"), przybliżonego wyznaczania miejsca zerowego dowolnej ściśle rosnącej funkcji typu $f : \mathbb{R} \longrightarrow \mathbb{R}$. Uzasadnij poprawność algorytmu i oszacuj jego złożoność.
13. Podaj przykład danych wejściowych rozmiaru n, dla których:
    1. algorytm wyszukiwania binarnego wykona istotnie więcej porównań niż algorytm Skoki co k, dla optymalnego k,

2. algorytm wyszukiwania sekwencyjnego wykona w przybliżeniu tyle samo porównań co algorytm Skoki co k, dla optymalnego k,
3. algorytm wyszukiwania sekwencyjnego wykona w przybliżeniu tyle samo porównań co algorytm wyszukiwania binarnego,
4. algorytm wyszukiwania interpolacyjnego wykona $\Theta(n)$ porównań.

14. Rozważmy zmodyfikowaną wersję algorytmu Skoki co k, dla $k = \lfloor \sqrt{n} \rfloor$, gdzie n jest długością ciągu wejściowego. W miejscu sekwencyjnego przeglądania $k-1$ ostatnich elementów wywołujemy rekurencyjnie algorytm Skoki co k, dla $k = \left\lfloor \sqrt{\lfloor \sqrt{n} \rfloor} \right\rfloor$. Postępowanie powtarzamy rekurencyjnie dopóki $k > 1$, w p.p. znaleźliśmy rozwiązanie. Zaimplementuj rozważany algorytm. Oszacuj możliwie dokładnie złożoność czasową i pamięciową metody.

15. Zaproponuj algorytm, który dla danego n-elementowego wektora różnych liczb naturalnych i danej liczby naturalnej x, wyznaczy wszystkie pary liczb (p, q), takie że $T[p] \cdot T[q] = x$. Przedyskutuj różne rozwiązania i oszacuj ich złożoność:

   1. gdy ciąg jest dowolny,
   2. gdy ciąg jest uporządkowany.

16. Niech S oraz T będą uporządkowanymi rosnąco, n elementowymi wektorami liczb naturalnych i niech k będzie daną liczbą naturalną. Zaproponuj możliwie efektywny algorytm wyznaczający liczby i oraz j takie, że $S[i] + T[j] = k$. Uzasadnij poprawność rozwiązania i oszacuj jego koszt.

17. Niech S oraz T będą uporządkowanymi malejąco wektorami złożonymi odpowiednio z n oraz k liczb naturalnych. Zaproponuj możliwie efektywny algorytm wyznaczający medianę ciągu, powstałego z połączenia tablic S oraz T. Algorytm nie powinien wykorzystywać dodatkowej pamięci (poza pamięcią potrzebną do przechowywania danych).