

Algorytmy i struktury danych

Wykład II – wyszukiwanie

Paweł Rembelski

PJWSTK

4 października 2009



- 1 Wyszukanie wskazanego elementu bez uporządkowania
 - Algorytm Find
- 2 Wyszukanie elementu maksymalnego
 - Algorytm FindMax
- 3 Wyszukanie elementu 2-go co do wielkości
 - Algorytm Find2nd
 - Algorytm Turniej – szkic
- 4 Wyszukanie elementów minimalnego i maksymalnego
 - Algorytm FindMinMax
 - Algorytm RecMinMax
- 5 Wyszukiwanie i otoczka wypukła
 - Wstęp
 - Algorytm Jarvisa

Wyszukanie wskazanego elementu bez uporządkowania

Problem, struktura i specyfikacja algorytmu

Problem

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

Struktura dla algorytmu

Struktura dla algorytmu Alg: $\langle \mathbb{N}, +, \neq \rangle$.

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

Wyszukanie wskazanego elementu bez uporządkowania

Algorytm Find

Rozwiązanie problemu – **algorytm Find**:

```
1  int Find(int T[], int n, int x) {  
    | WP : T jest niepustym wektorem  
    | różnych liczb naturalnych,  $n \in \mathbb{N}^+$ ,  $|T| = n$ ,  $\exists! (0 \leq i < n) (T[i] = x)$   
2    int idx:=0;  
3  
4    while (T[idx]≠x)  
5        idx:=idx+1;  
6  
7    return idx;  
8 }  
    | WK : Find(T, n, x) = idx, gdzie T[idx] = x.
```

Poprawność algorytmu Find

- 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 $Find(T, n, x) = idx, T[idx] = x$,
- warunek stopu: z warunku początkowego $\exists! (0 \leq i < n) (T[i] = x)$ i $n \in \mathbb{N}^+$. Zmienna idx inicjalizowana wartością 0 jest inkrementowana z każdą iteracją pętli o 1, stąd po i iteracjach pętli $idx = i$, czyli $T[idx] = x$, co kończy działanie algorytmu.

Zadanie. Podaj postać niezmiennik uzasadniającego poprawność częściową algorytmu Find.

Złożoność algorytmu Find

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

Pytanie. Czy prawdziwe jest stwierdzenie $T(Find, n) = \Theta(n)$?

Twierdzenie

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

* powyższe twierdzenie jest prawdziwe w klasycznym modelu obliczeniowym deterministycznej maszyny Turinga. Nie jest ono jednak spełnione np. w modelu kwantowej maszyny Turinga (zob. *algorytm Grover'a*, 1996).

Wyszukanie elementu maksymalnego

Problem, struktura i specyfikacja algorytmu

Problem

Podać algorytm $Alg(T, n)$ znajdujący indeks elementu maksymalnego w niepustym n -elementowym wektorze różnych liczb naturalnych T .

Struktura dla algorytmu

Struktura dla algorytmu Alg: $\langle \mathbb{N}, +, < \rangle$.

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 : $Alg(T, n) = idxMax$, gdzie $\forall (0 \leq i < n) (T[i] \leq T[idxMax])$.

Wyszukanie wskazanego elementu bez uporządkowania

Algorytm FindMax

Rozwiązanie problemu – **algorytm FindMax**:

```

1  int FindMax(int T[], int n) {←—————| WP : T jest niepustym wektorem
                                     różnych liczb naturalnych,  $n \in \mathbb{N}^+$ ,  $|T| = n$ 
2      int i:=1, idxMax:=0;
3
4      while (i<n) {
5          if (T[idxMax]<T[i]) idxMax:=i;
6          i:=i+1;
7      }
8      return idxMax;←—————| WK :  $Alg(T, n) = idxMax$ , gdzie
                                      $\forall (0 \leq i < n) (T[i] \leq T[idxMax])$ 
9  }
```

Dowód częściowej poprawności algorytmu **metodą niezmienników**:

```

1  int FindMax(int T[], int n) {←—————| T jest niepustym wektorem
                                   różnych liczb naturalnych,  $n \in \mathbb{N}^+$ ,  $|T| = n$ 
2  int i:=1, idxMax:=0;←—————|  $T[idxMax] \leq T[idxMax]$ 
                                    $\Rightarrow \forall (0 \leq j < i) (T[j] \leq T[idxMax])$ 
3  ←—————| ustalenie niezmiennika:  $\forall (0 \leq j < i) (T[j] \leq T[idxMax])$ 
4  while (i<n) {←—————| weryfikacja niezmiennika:  $\forall (0 \leq j < i) (T[j] \leq T[idxMax])$ 
5    if (T[idxMax]<T[i]) idxMax:=i;←—————|  $\forall (0 \leq j \leq i) (T[j] \leq T[idxMax])$ 
6    i:=i+1;←—————| odtworzenie niezmiennika:  $\forall (0 \leq j < i) (T[j] \leq T[idxMax])$ 
7  } ←—————|  $(NZ \wedge \neg(i < n)) \Rightarrow \forall (0 \leq j < i) (T[j] \leq T[idxMax])$ 
                                    $\Rightarrow \forall (0 \leq j < n) (T[j] \leq T[idxMax])$ 
8  return idxMax;←—————| Alg (T, n) = idx, gdzie  $\forall (0 \leq i < n) (T[i] \leq T[idx])$ 
9  }
```

Wniosek

Algorytm FindMax jest częściowo poprawny względem specyfikacji $\langle WP, WK \rangle$.

Zadanie. Udowodnij przez indukcję własność stopu algorytmu FindMax.

Złożoność algorytmu FindMax

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

Twierdzenie

Algorytm FindMax jest optymalnym rozwiązaniem postawionego problemu.

Wyszukanie elementu 2-go co do wielkości

Problem, struktura i specyfikacja algorytmu

Problem

Podać algorytm $Alg(T, n)$ znajdujący indeks elementu 2-go co do wielkości w n -elementowym wektorze różnych liczb naturalnych T , gdzie $n \geq 2$.

Struktura dla algorytmu

Struktura dla algorytmu Alg: $\langle \mathbb{N}, +, <, > \rangle$.

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, $n \in \mathbb{N} \setminus \{0, 1\}$, $|T| = n$,
- $WK : Alg(T, n) = idxSec$, gdzie $\exists! (0 \leq i < n) (T[i] > T[idxSec])$.

Wyszukanie elementu 2-go co do wielkości

Algorytm Find2nd

Rozwiązanie problemu – algorytm Find2nd:

```

1  int Find2nd(int T[], int n) {←—————| WP : T jest niepustym wektorem
                                   różnych liczb naturalnych,  $n \in \mathbb{N} \setminus \{0, 1\}$ ,  $|T| = n$ ,
2      int i:=2, idxMax, idxSec;
3
4      if (T[0]<T[1]) { idxMax:=1; idxSec:=0; }
5      else { idxMax:=0; idxSec:=1 }
6
7      while (i<n) {
8          if (T[i]>T[idxMax]) { idxSec:=idxMax; idxMax:=i; }
9          else if (T[i]>T[idxSec]) idxSec:=i;
10
11         i:=i+1;
12     }
13
14     return idxSec;←—————| WK : Find2nd(T, n) = idxSec,
                                   gdzie  $\exists! (0 \leq i < n) (T[i] > T[idxSec])$ 
15 }

```

Poprawność algorytmu Find2nd

- poprawność częściowa: tuż po inicjalizacji zmiennych w wierszach 4-5 prawdą jest, że zmienna $idxSec$ wskazuje na 2-gi co do wielkości element w ciągu dwuelementowym $T[0], T[1]$. Niezmiennikiem pętli jest formuła NZ :
 $\exists!(0 \leq j < i)(T[j] > T[idxSec])$. Po wykonaniu instrukcji warunkowej w wierszach 8-9 prawdą jest, że $\exists!(0 \leq j \leq i)(T[j] > T[idxSec])$. Zatem po wykonaniu instrukcji $i := i + 1$ zachodzi $\exists!(0 \leq j < i)(T[j] > T[idxSec])$ – **odtworzenie niezmiennika NZ** . Po zakończeniu pętli iteracyjnej mamy $i = n$, stąd $\exists!(0 \leq i < n)(T[i] > T[idxSec])$,
- warunek stopu: z warunku początkowego $n \in \mathbb{N} \setminus \{0, 1\}$. Zmienna i inicjalizowana wartością 2 jest inkrementowana z każdą iteracją pętli o 1, stąd po $n - 2$ iteracjach pętli $i = n$, co kończy działanie algorytmu.

Złożoność algorytmu Find2nd

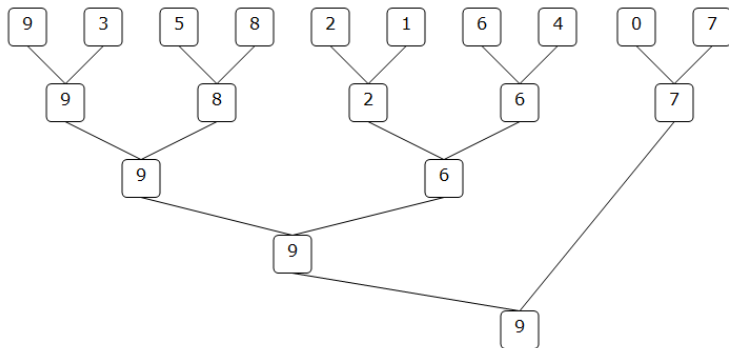
- operacja dominująca: porównanie elementów rozważanego uniwersum,
- średnia złożoność czasowa: $A(n) = \frac{2n}{2} + \frac{n}{2} \pm c = \Theta(n)$, gdzie $c \leq 2$ jest pewną stałą, zakładając $Pr(T[i] > T[idxMax]) = \frac{1}{2}$, dla każdego $0 \leq i < n$,
- pesymistyczna złożoność czasowa: oraz $W(n) = 2n \pm c = \Theta(n)$, gdzie $c \leq 2$ jest pewną stałą,
- złożoność pamięciowa: $O(1)$.

Pytanie. Czy algorytm Find2nd jest optymalnym rozwiązaniem dla rozważanego problemu?

Wyszukanie elementu 2-go co do wielkości

Algorytm Turniej – szkic

Pomysł. Zbuduj drzewo turnieju zgodnie z zasadą „przechodzi tylko wygrywający”, np. dla $T = [9, 3, 5, 8, 2, 1, 6, 4, 0, 7]$ i $n = 10$ otrzymujemy:



Pytanie. Ile porównań elementów tablicy T jest niezbędnych do zbudowania drzewa turnieju dla rozważanego przykładu (tj. $n = 10$) i w przypadku ogólnym?

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

Wniosek

Element 2-gi co do wielkości jest jednym z tych, które „przegrały” z elementem maksymalnym. Tych elementów jest co najwyżej $\lceil \lg n \rceil$ i wśród nich należy szukać elementu maksymalnego np. stosując metodę FindMax.

Złożoność algorytmu Turniej

- operacja dominująca: porównanie elementów rozważanego uniwersum,
- pesymistyczna złożoność czasowa:
$$W(\text{Turniej}, n) = n - 1 + \lceil \lg n \rceil - 1 = n + \lceil \lg n \rceil - 2,$$
- złożoność pamięciowa: $O(n)$ ze względu na konieczność zapisywania rezultatów pojedynków.

Twierdzenie

Algorytm Turniej jest **optymalnym z dokładnością co do stałej** w przypadku pesymistycznym rozwiązaniem rozważanego problemu, tj. złożoność każdego innego optymalnego w przypadku pesymistycznym algorytm dla postawionego problemu jest rzędu $W(\text{Turniej}, n) + c = n + \lceil \lg n \rceil - 2 + c$, gdzie c jest nieujemną stałą naturalną.

Wyszukanie elementów minimalnego i maksymalnego

Problem, struktura i specyfikacja algorytmu

Problem

Podać algorytm $Alg(T, n)$ znajdujący indeks elementów minimalnego i maksymalnego w n -elementowym wektorze różnych liczb naturalnych T , gdzie $n = 2k$ i $k \in \mathbb{N}^+$.

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, $n = 2k$, $k \in \mathbb{N}^+$,
 $|T| = n$,
- WK : $Alg(T, n) = (idxMin, idxMax)$, gdzie $\forall (0 \leq i < n) (T[idxMin] \leq T[i])$,
 $\forall (0 \leq i < n) (T[idxMax] \geq T[i])$.

Pytanie. Czy sekwencyjne zastosowanie algorytmów FindMax i analogicznego FindMin jest poprawnym rozwiązaniem rozważanego problemu? Jeżeli tak, to jaka jest dokładna złożoność czasowa takiego rozwiązania?

Wyszukanie elementów minimalnego i maksymalnego

Algorytm FindMinMax

Rozwiązanie problemu – **algorytm FindMinMax:**

```

1  (int, int) FindMinMax(int T[], int n) { $\leftarrow$ ————— | WP : T jest niepustym wektorem
                                różnych liczb naturalnych,  $n = 2k$ ,  $k \in \mathbb{N}^+$ ,  $|T| = n$ ,
2      int i:=0;
3      (int, int) s;
4
5      for (i:=0; i<n/2; i:=i+1)
6          if (T[i]>T[(n-1)-i])
6              SWAP(T,i,(n-1)-i); $\leftarrow$ ————— | operacja zamiany elementów wektora
7
8      s.idxMin:=FindMin(T[0..n/2-1],n/2); $\leftarrow$ ————— | procedura wyszukania elementu
                                minimalnego analogiczna do omówionej metody FindMax
9      s.idxMax:=FindMax(T[n/2..n-1],n/2);
10
11     return s; $\leftarrow$ ————— | WK :  $FindMinMax(T, n) = (idxMin, idxMax)$ , gdzie
                                 $\forall (0 \leq i < n) (T[idxMin] \leq T[i]), \forall (0 \leq i < n) (T[idxMax] \geq T[i])$ .
12 }

```

Zadanie. Uzasadnij poprawność całkowitą algorytmu FindMinMax.

Złożoność algorytmu Jarvisa

- operacja dominująca: porównywanie elementów rozważanego uniwersum,
- złożoność czasowa metody pomocniczej FindMin: $T(\text{FindMin}, n) = \frac{n}{2} - 1$,
- złożoność czasowa metody pomocniczej FindMax: $T(\text{FindMax}, n) = \frac{n}{2} - 1$,
- złożoność czasowa algorytmu Jarvisa:

$$T(\text{FindMinMax}, n) = \frac{n}{2} + T(\text{FindMin}, n) + T(\text{FindMax}, n) = \frac{3}{2}n - 2,$$

- złożoność pamięciowa algorytmu Jarvisa: $O(1)$.

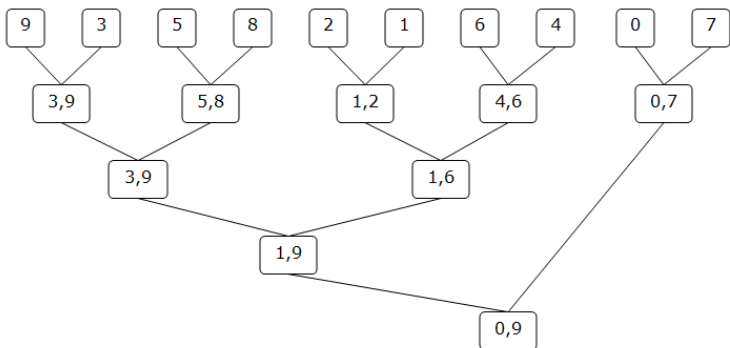
Twierdzenie

Algorytm FindMinMax jest **optymalnym z dokładnością co do stałej** rozwiązaniem rozważanego problemu, tj. złożoność każdego innego optymalnego algorytmu dla postawionego problemu jest rzędu $T(\text{FindMinMax}, n) + c = \frac{3}{2}n - 2 + c$, gdzie c jest nieujemną stałą naturalną.

Wyszukanie elementów minimalnego i maksymalnego

Algorytm RecMinMax

Pomysł. Zbuduj rekurencyjnie zmodyfikowane drzewo turnieju zgodnie z zasadą „przechodzą tylko elementy minimalny i maksymalny”, rezultat znajdziemy w korzeniu drzewa, np. dla $T = [9, 3, 5, 8, 2, 1, 6, 4, 0, 7]$ i $n = 10$ otrzymujemy:



Konstruujemy procedurę rekurencyjną $RecMinMax(T, l, r)$, gdzie l i r to kolejno indeks lewego i prawego krańca aktualnie analizowanego fragmentu wektora T . Dalej załóżmy bez straty ogólności, że $(r - l + 1) = n = 2^k$ i $k \in \mathbb{N}^+$.

Rozwiązanie problemu – algorytm RecMinMax:

```

1  (int,int) RecMinMax(int T[], int l, int r) { ←————— | WP : T jest niepustym
      wektorem różnych liczb naturalnych,  $(r - l + 1) = n = 2^k$ ,  $k \in \mathbb{N}^+$ ,  $|T| = n$ ,
2      int idxMin, idxMax;
3      (int,int) sl, sr;
4
5      if (r-l=1)
6          if (T[r]>T[l]) return (l,r) else return (r,l);
7
8      else {
9          sl:=RecMinMax(T,l,(l+r)/2);
10         sr:=RecMinMax(T,((l+r)/2)+1,r);
11
12         if (T[sl.idxMin]<T[sr.idxMin]) idxMin:=sl.idxMin else idxMin:=sr.idxMin;
13         if (T[sl.idxMax]<T[sr.idxMax]) idxMax:=sl.idxMax else idxMax:=sr.idxMax;
14
15         return (idxMin,idxMax); ←————— | WK :  $RecMinMax(T,l,r) = (idxMin, idxMax)$ , gdzie
       $\forall (l \leq i \leq r) (T[idxMin] \leq T[i]), \forall (l \leq i \leq r) (T[idxMax] \geq T[i])$ .
16     }
17 }

```

Dowód częściowej poprawności algorytmu przez **indukcję** względem $k \in \mathbb{N}^+$, gdzie $n = 2^k$:

- **baza indukcji**: dla $k = 1$, tj. $n = 2$ zachodzi $r - l = 1$, zatem wykonany jest pierwszy warunek zewnętrznej instrukcji warunkowej, czyli
`if (T[r]>T[l]) return (1,r) else return (r,1);`
 Stąd algorytm RecMinMax jest częściowo poprawny dla $k = 1$ i $n = 2$,
- **założenie indukcyjne**: dla $0 \leq i < k$ i $n = 2^i$ algorytm RecMinMax jest częściowo poprawny,
- **teza indukcyjna**: dla $k > 1$ i $n = 2^k$ algorytm RecMinMax jest częściowo poprawny.

Dowód tezy indukcyjnej

Dla $k > 1$, tj. $n > 2$ mamy $r - l > 2$, zatem wykonany jest drugi warunek zewnętrznej instrukcji warunkowej, czyli kolejno instrukcje w wierszach 9-10. Stąd i na podstawie założenia indukcyjnego zachodzi

$$sl.idxMin = \text{indeks elem. } \min \left(\left\{ T[l], T[l+1], \dots, T \left[\left\lfloor \frac{l+r}{2} \right\rfloor \right] \right\} \right),$$

$$sl.idxMax = \text{indeks elem. } \max \left(\left\{ T[l], T[l+1], \dots, T \left[\left\lfloor \frac{l+r}{2} \right\rfloor \right] \right\} \right),$$

Dowód tezy indukcyjnej c.d.

oraz

$$sr.idxMin = \text{indeks elem. min} \left(\left\{ T \left[\left\lfloor \frac{l+r}{2} \right\rfloor + 1 \right], T \left[\left\lfloor \frac{l+r}{2} \right\rfloor + 2 \right], \dots, T[r] \right\} \right),$$

$$sr.idxMax = \text{indeks elem. max} \left(\left\{ T \left[\left\lfloor \frac{l+r}{2} \right\rfloor + 1 \right], T \left[\left\lfloor \frac{l+r}{2} \right\rfloor + 2 \right], \dots, T[r] \right\} \right).$$

Następnie wykonane są instrukcje warunkowe

```

12     if (T[sl.idxMin]<T[sr.idxMin]) idxMin:=sl.idxMin else idxMin:=sr.idxMin;
13     if (T[sl.idxMax]<T[sr.idxMax]) idxMax:=sl.idxMax else idxMax:=sr.idxMax;
14
15     return (idxMin,idxMax);

```

Stąd

$$idxMin = \text{indeks elem. min} (\{ T[l], T[l+1], \dots, T[r] \}),$$

$$idxMax = \text{indeks elem. max} (\{ T[l], T[l+1], \dots, T[r] \}),$$

gdzie $r - l = n$ i $n = 2^k$, dla $k > 1$, co kończy dowód.**Zadanie.** Udowodnij przez indukcję własność stopu algorytmu RecMinMax.

Złożoność czasowa algorytmu RecMinMax

Niech dalej $T(n)$ będzie liczbą operacji porównań elementów rozważanego uniwersum, jakie wykonuje algorytm RecMinMax dla danych rozmiaru n (ponownie zakładamy, że $n = 2^k$ i $k \in \mathbb{N}^+$), 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 $k \in \mathbb{N}^+$

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

i ostatecznie* $T(2^k) = \frac{3}{2}2^k - 2$, stąd $T(n) = \frac{3}{2}n - 2$ *

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

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

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

Wniosek

Z twierdzenia o optymalności algorytmu FindMinMax i wyprowadzonej z powyższego równania rekurencyjnego postaci zwartej na $T(n)$ wynika, że algorytm RecMinMax jest optymalnym rozwiązaniem postawionego problemu.

Złożoność pamięciowa algorytmu RecMinMax

Złożoność pamięciowa algorytmu RecMinMax z uwzględnieniem kosztów rekursji jest asymptotycznie równa wysokości zmodyfikowanego drzewa turnieju, tj. $S(n) = \Theta(\lg n)$.

Wyszukiwanie i otoczka wypukła

Wstęp

Własność

Niech $p_0 = (x_0, y_0)$, $p_1 = (x_1, y_1)$ oraz $p_2 = (x_2, y_2)$ będą dowolnymi punktami płaszczyzny $\mathbb{N} \times \mathbb{N}$ oraz $\det(p_0, p_1, p_2)$ wyznacznikiem takim, że

$$\det(p_0, p_1, p_2) = (x_1 - x_0)(y_2 - y_0) - (y_1 - y_0)(x_2 - x_0),$$

wtedy, jeżeli:

- $\det(p_0, p_1, p_2) > 0$ wtedy punkt p_2 leży po lewej stronie prostej przechodzącej przez punkty p_0, p_1 ,
- $\det(p_0, p_1, p_2) = 0$ wtedy punkty p_0, p_1 oraz p_2 są współliniowe,
- $\det(p_0, p_1, p_2) < 0$ wtedy punkt p_2 leży po prawej stronie prostej przechodzącej przez punkty p_0, p_1 .

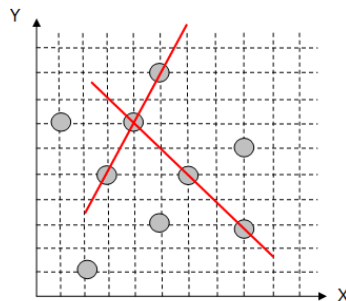
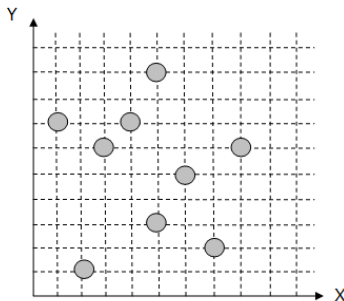
Definicja

Niech p_0 będzie wybranym punktem płaszczyzny $\mathbb{N} \times \mathbb{N}$. Ustalamy relację porządku częściowego $\preceq_{p_0} \subseteq (\mathbb{N} \times \mathbb{N})^2$ taką, że $p_1 \preceq_{p_0} p_2$ wtedy i tylko wtedy, gdy $\det(p_0, p_1, p_2) \geq 0$ dla dowolnych punktów $p_1, p_2 \in \mathbb{N} \times \mathbb{N}$. Relację \preceq_{p_0} będziemy nazywali *porządkiem kątowym*.

Wniosek

Niech Q będzie dowolnym zbiorem punktów płaszczyzny $\mathbb{N} \times \mathbb{N}$. Jeżeli dla dowolnego punktu p_0 dowolne dwa różne punkty p_1, p_2 należące do dziedziny relacji \preceq_{p_0} nie są współliniowe (tj. $\det(p_0, p_1, p_2) \neq 0$), to relacja \preceq_{p_0} jest relacją porządku liniowego w zbiorze Q .

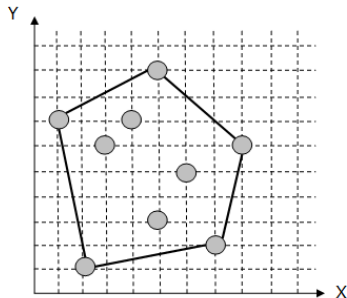
Przykład. Zbiory punktów, w których relacja porządku kąowego jest/nie jest relacją porządku liniowego.



Definicja

Niech $Q \subset \mathbb{N} \times \mathbb{N}$ będzie zbiorem punktów płaszczyzny, z których każde trzy różne punkty **nie są współliniowe**. *Otoczką wypukłą* zbioru Q nazywamy zbiór $CH(Q) \subseteq Q$ taki, że wielokąt wypukły o wierzchołkach ze zbioru punktów $CH(Q)$ zawiera wewnątrz wszystkie punktu zbioru $Q \setminus CH(Q)$ oraz $CH(Q)$ jest najmniejszym takim zbiorem.

Przykład. Zbiór punktów Q oraz otoczka wypukła $CH(Q)$ owego zbioru.



Wyszukiwanie i otoczka wypukła

Problem, struktura i specyfikacja algorytmu

Problem

Podać algorytm $Alg(P, n)$ znajdujący liczbę punktów tworzących otoczkę wypukłą niepustego n -elementowego zbioru różnych i trójkami niewspółliniowych punktów Q płaszczyzny $\mathbb{N} \times \mathbb{N}$, zapisanego w wektorze (tablicy) punktów

$$P = [(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})].$$

Struktura dla problemu

Rozszerzenie standardowej struktury dla algorytmu Alg: zbiór punktów płaszczyzny $\mathbb{N} \times \mathbb{N}$, z wyróżnioną relacją \preceq porządku kąowego.

Specyfikacja algorytmu

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

- WP : P jest niepustym wektorem różnych i trójkami niewspółliniowych punktów, $n \in \mathbb{N}^+, |P| = n$,
- WK : $Alg(P, n) = |CH(Q)|$, gdzie Q jest zbiorem reprezentowanym przez wektor P .

Wyszukiwanie i otoczka wypukła

Algorytm Jarvisa

int FindStart((int,int) P[], int n)

- warunek początkowy *WP* : P jest niepustym wektorem różnych i trójkami niewspółliniowych punktów, $n \in \mathbb{N}^+$, $|P| = n$,
- warunek końcowy *WK* : $FindStart(P, n) = idx$, gdzie $P[idx]$ jest punktem o najmniejszej współrzędnej y a w razie niejednoznaczności kolejno najmniejszej współrzędnej x .

```

1  int FindStart((int,int) P[], int n) { ←————— | WP
2      int i, idx:=0;
3
4      for (i:=1; i<n; i:=i+1)
5          if ((P[idx].y>P[i].x) OR
6              ((P[idx].y=P[i].y) AND ((P[idx].x>P[i].x)))
7              idx:=i;
8
9      return idx; ←————— | WK
10 }
```

Zadanie. Uzasadnić całkowitą poprawność procedury FindStart.

```
int FindNext((int,int) P[], int n, (int,int) p)
```

- warunek początkowy WP : P jest niepustym wektorem różnych i trójkami niewspółliniowych punktów, $n \in \mathbb{N}^+, |P| = n, \exists (0 \leq i < n) (P[i] = p)$,
- warunek końcowy: WK : $FindNext(P, n, p) = idx$, gdzie $P[idx] \neq p$ jest punktem najmniejszym w sensie relacji porządku kąowego \preceq_p (tj. względem punktu p).

```

1  int FindNext((int,int) P[], int n, (int,int) p) { ← WP
2    int i, idx:=RandomDiff(Q,n,p), ← indeks dowolnego elementu wektora P
                                     różnego od elementu p
3      first=idx;
4
5      for (i:=1; i<n; i:=i+1)
6        if ((i!=first) AND (P[i]!=p) AND (P[i]⊆pP[idx]))
7          idx:=i;
8
9      return idx; ← WK
10 }
```

Zadanie. Uzasadnić całkowitą poprawność procedury FindNext.

Rozwiązanie problemu – algorytmu Jarvisa:

```

1  int Jarvis((int,int) P[], int n) {←—————| WP : P jest niepustym wektorem
                                   różnych i trójkami niewspółliniowych punktów,  $n \in \mathbb{N}^+$ ,  $|P| = n$ 
2      int size:=0, start:=FindStart(P,n), tmp:=start;
3
4      do {
5          size:=size+1;
6
7          tmp:=FindNext(P,n,T[tmp]);
8      } while (tmp!=start)
9
10     return size;←—————| WK :  $Jarvis(P, n) = |CH(Q)|$ ,
                                   gdzie Q jest zbiorem reprezentowanym przez wektor P
11
12 }
```

Poprawność algorytmu Jarvisa

- poprawność częściowa: metoda pomocnicza FindStart ustala pierwszy wierzchołek poszukiwanej otoczki wypukłej $start$. Niech dalej α oznacza liczbę „odwiedzin” elementów wektora P . Niezmiennikiem pętli jest formuła: $NZ : size = \alpha - 1$. Stąd po wykonaniu instrukcji $size := size + 1$ prawdą jest, że $size = \alpha$. Procedura FindNext wyznacza kolejny wierzchołek otoczki wypukłej kątowno najbliższy aktualnemu wierzchołkowi tmp . Zatem po wykonaniu instrukcji $tmp := FindNext(P, n, tmp)$ prawdą jest $size = \alpha - 1$ – **odtworzenie niezmiennika NZ**. Po zakończeniu pętli iteracyjnej mamy $tmp = start$, tj. pierwszy wierzchołek otoczki wypukłej $start$ odwiedzony był 2-krotnie. Z własności geometrycznych otoczki wypukłej i procedury FindNext wynika, że każdy inny element wektora T (tj. wierzchołek zbioru Q) odwiedzony był co najwyżej jeden raz. Stąd liczba wierzchołków zbioru Q tworzących otoczkę wypukłą $CH(Q)$ jest równa dokładnie $\alpha - 1 = size$, czyli $Jarvis(P, n) = size = |CH(Q)|$.
- warunek stopu: ponieważ zbiór Q jest zbiorem skończonym i $CH(Q) \subseteq Q$ oraz z powyższego rozumowania każdy wierzchołek zbioru Q inny niż $start$ odwiedzany jest co najwyżej jeden raz, to ciąg wywołań metody FindNext jest także skończony dla dowolnego wektora P długości n reprezentującego zbiór Q .

Złożoność algorytmu Jarvisa

- operacja dominująca: porównywanie elementów rozważanego uniwersum względem relacji \preceq ,
- złożoność czasowa metody pomocniczej FindStart: $T(\text{FindStart}, n) = \Theta(1)$,
- złożoność czasowa metody pomocniczej FindNext: $T(\text{FindNext}, n) = \Theta(n)$,
- złożoność czasowa algorytmu Jarvisa:

$$T(\text{Jarvis}, n) = T(\text{FindStart}, n) + k \cdot (T(\text{FindNext}, n) + O(1)) = \Theta(kn),$$

gdzie k jest liczbą punktów tworzących otoczkę wypukłą zbioru Q ,

- złożoność pamięciowa algorytmu Jarvisa: $O(1)$,

Pytanie. Jaka jest pesymistyczna złożoność czasowa algorytmu Jarvisa?

Pytanie. Jaka jest średnia złożoność czasowa algorytmu Jarvisa?


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 Find.
- 2 Zaimplementuj algorytm FindMax.
- 3 Zaimplementuj algorytm Find2nd.
- 4 Zaimplementuj algorytm Turniej.
- 5 Zaproponuj algorytm wyszukania elementu 3-go co do wielkości w n -elementowym wektorze różnych liczb naturalnych T , gdzie $n \geq 3$, bazujący na metodzie turniejowej. Uzasadnij poprawność algorytmu i oszacuj jego złożoność.
- 6 Zaimplementuj algorytm FindMinMax.
- 7 Zaimplementuj algorytm rekurencyjny RecMinMax, dla:
 - 1 rozmiaru danych wejściowych $n = 2^k$ i $k \in \mathbb{N}^+$,
 - 2 rozmiaru danych wejściowych $n = 2k$ i $k \in \mathbb{N}^+$.
- 8 Przeprowadź doświadczalnie analizę porównawczą efektywności algorytmów FindMinMax i RecMinMax.
- 9 Zaimplementuj następujący algorytm równoczesnego wyszukiwania minimum i maksimum w n -elementowym wektorze różnych liczb naturalnych T , gdzie $n \geq 2$:
 - 1 dla każdej pary kolejnych elementów wektora ustawiam minimum na pozycji parzystej, a maksimum na pozycji nieparzystej,
 - 2 wyszukaj sekwencyjnie elementu minimalnego na pozycjach parzystych,
 - 3 wyszukaj sekwencyjnie elementu maksymalnego na pozycjach nieparzystych.

Uzasadnij poprawność algorytmu i oszacuj jego złożoność. Czy podana metoda jest optymalnym rozwiązaniem rozważanego problemu?

- 10 Zaimplementuj algorytm Jarvisa.
- 11 Zaproponuj i zaimplementuj algorytm sprawdzania, czy dana relacja binarna w zbiorze $\{1, 2, \dots, n\}$, zadana w postaci n -elementowego wektora T par postaci (x, y) jest:
 

- 1 zwrotna,
- 2 symetryczna,
- 3 przechodnia.

Uzasadnij poprawność algorytmu i oszacuj jego złożoność.