Algorytmy i Struktury Danych, 10. ćwiczenia

2019-12-04

Spis treści

1 Pokazać, że przy pomocy rotacji można zawsze przejść z jednego BST do drugiego **1**

2 Drzepce **1**

3 ASD Zadania — wzbogacanie struktur danych **2**

1 Pokazać, że przy pomocy rotacji można zawsze przejść z jednego BST do drugiego

Za pomocą rotacji możemy dowolne drzewo BST zamienić na listę (i na odwrót). Wstarczy tak długo jak drzewo zawiera węzeł z lewym synem, wykonać na nim (i lewym synie) prawą rotację.

2 Drzepce

`https://en.wikipedia.org/wiki/Treap`

Danych jest n par liczb całkowitych, które się parami różnią na każdej pozycji. Pierwsze elementy para to klucze, drugie to priorytety.

- Wykaż, że istnieje dokładnie jedno drzewo binarne, które jest drzewem wyszukiwań binarnych ze względu na klucze i jednocześnie kopcem typu MAX ze względu na priorytety.
- Niech T będzie wyszukiwań drzewem binarnych ze względu na klucze. Opisz konstrukcję ciągu rotacji o długości O(n), które należy wykonać, żeby przekształcić drzewo T w drzewo BST, które będzie jednocześnie kopcem binarnym typu MAX ze względu na priorytety.
- Zaproponuj algorytm, który w czasie liniowym znajdzie ciąg rotacji, o którym mowa w poprzednim punkcie

Rozwiązanie: Istnieje tylko jedno takie drzewo:

- korzeń jest jednoznacznie wyznaczony przez parę o największym priorytecie,

- lewo poddrzewo jest tworzone rekurencyjnie (węzły o kluczach mniejszych niż korzeń),
- analogicznie prawe poddrzewo (węzły o kluczach większych niż korzeń)

Każde drzewo da się przekształcić za pomocą $O(n)$ rotacji do listy. Niech T oryginalne drzewo BST, T' drzewo docelowe. Przekształcamy za pomocą $O(n)$ rotacji T do L, a następnie za pomocą $O(n)$ rotacji z L do T'.

Drzewo T' można skonstruować w czasie $O(n)$ za pomocą n zapytań RMQ (range minimum query).

3 ASD Zadania — wzbogacanie struktur danych

Zadanie 3.24

Zaprojektuj strukturę danych umożliwiającą wykonywanie w czasie $O(\log n)$ następujących operacji na zbiorze S:

- $makeset(S)$ :: $S := \emptyset$
- $insert((x, y), S)$ :: $S := S \cup \{(x, y)\}$
- $minx(S)$ :: usunięcie z S pary (x, y) o najmniejszej pierwszej składowej,
- $miny(S)$ :: usunięcie z S pary (x, y) o najmniejszej drugiej składowej,
- $searchx(x, S)$ :: wyznaczenie takiej pary $(a, b) \in S$, że $x = a$,
- $searchy(y, S)$ :: wyznaczenie takiej pary $(a, b) \in S$, że $y = b$.

Rozwiązanie: dwa drzewa AVL (jedno ze współrzędnymi x, drugie ze współrzędnymi y), dodatkowo każdy węzeł trzyma dowiązanie do odpowiadającego mu węzła w drugim drzewie.

Zadanie 3.25

Zaprojektuj strukturę danych umożliwiającą wykonywanie w czasie $O(\log n)$ następujących operacji na zbiorze S:

- $construct(S)$ :: utworzenie ciągu pustego S,
- $insert(S, x)$ :: $S := S \cup \{(x)\}$,
- $delete(S, x)$ :: $S := S - \{(x)\}$,
- $search(S, x)$ :: sprawdzenie, czy x znajduje się w zbiorze S,
- $elem(S, i)$ :: wyznaczenie i–tego co do wielkości elementu zbioru S,
- $numb(S, x)$ :: wyznaczenie numeru elementu x w zbiorze S (względem wielkości).

Rozwiązanie: drzewo AVL z atrybutami rozmiar poddrzewa.

Zadanie 3.26

Zaprojektuj strukturę danych do wykonywania ciągów następujących operacji (dla elementów x pochodzących z dowolnego zbioru liniowo uporządkowanego):

- $initialization$ :: $S_i = \emptyset$ dla $i = 1, 2, \ldots, n$,
- $insert(S_i, x)$ :: $S_i := S_i \cup \{(x)\}$, pod warunkiem, że x nie występuje w żadnym zbiorze S_j, $1 \leq j \leq n$,
- $deletemin(S_i)$ :: usunięcie ze zbioru S_i najmniejszego elementu,
- $find(x)$ :: wyznaczenie numeru zbioru do którego należy element x.

Rozwiązanie: S_i jako zwykłe kopce, dodatków utrzymujemy słownik par $(x, numerzbioru)$

Zadanie 3.27

Zaprojektuj strukturę danych umożliwiającą wykonywanie w czasie $O(\log n)$ następujących operacji na ciągu S:

- $construct(S)$ :: utworzenie ciągu pustego S,
- $insert(S, i, x)$ :: wstawienie x na i-te miejsce w ciągu S, tzn. $S_i = x$ pod warunkiem, że $i \leq |S| + 1$,
- $sum(S, i, j)$ :: obliczenie sumy $\sum_{k=i}^{j} S_k$,

Rozwiązanie: AVL z dodatkowym atrybutem suma elementów poddrzewa.

Zadanie 3.28

Rozwiązanie: AVL z atrybutem rozmiar poddrzewa.

Zadanie 3.29

Zaprojektuj strukturę danych umożliwiającą wykonywanie w czasie $O(\log n)$ następujących operacji na zbiorze S zawierającym przedziały liczb rzeczywistych $[l, r]$:

- $empty(S)$ :: $S = \emptyset$,
- $add(S, I)$ :: $S = S \cup \{I\}$,
- $delete(S, I)$ :: $S = S - \{I\}$,
- $is(S, x)$ :: sprawdzenie czy element x należy do jakiegoś przedziału w zbioru S;
- $intersect(S, I)$ :: sprawdzenie czy przedział I ma niepuste przecięcie z jakimś przedziałem należącym do S.

Rozwiązanie: Utrzymujemy słownik z parami (x, z) (gdzie x to liczba rzeczywista, a z +1 lub -1). Dodatkowo każdy węzeł ma dodatkowy atrybut *suma* oznaczający sumę wartości z w poddrzewie. Możemy w takim drzewie w czasie $O(\log n)$ obliczyć $sum(q)$ oznaczającą sumę wszystkich atrybutów z par (x, z), takich, że $x \leq q$.

- $add(S, I)$ – dodajemy do słownika pary $(l, +1)$ i $(r, -1)$,
- $delete(S, I)$ – usuwamy ze słownika pary $(l, +1)$ i $(r, -1)$,
- $is(S, x)$ – jeśli słownik zawiera pary $(x, +1)$ lub $(x, -1)$ to zwracamy $true$, wpp. obliczamy $sum(x)$ i jeśli suma jest > 0 to zwracamy $true$, jeśli $sum(x) \leq 0$, to zwracamy $false$.
- $intersect(S, I)$, jeśli $is(S, l)$ lub $is(S, r)$ to zwracamy $true$, jeśli istnieje w słowniu para (x, z), t.że $l \leq x \leq r$, to zwracamy $true$, wpp zwracamy $false$.