Zadanie

Zaprojektuj strukturę danych, która umożliwia wydajne wykonywanie następujących operacji na dynamicznym zbiorze S złożonym z liczb całkowitych, z których każda ma przypisaną wagę całkowitoliczbową:

Ini(S):: S := ∅; {*operacja wykonywana tylko raz na samym początku*}

Insert(i,w):: wstaw do zbioru S liczbę i, której waga wynosi w; jeśli i jest już w zbiorze, to zmień wagę i na w

Interval(i):: podaj parę l, p taką, że $l \leq i \leq p$ oraz dla każdego $k \in \{l, l+1, \ldots, p\}$ mamy $k \in S$, $l-1 \notin S$, $p+1 \notin S$
{*operacja ta jest wykonywana tylko dla i ∈ S; polega na wyznaczeniu maksymalnego przedziału kolejnych liczb całkowitych ze zbioru S, do którego należy i*}

Weight(i):: podaj sumę wag elementów z przedziału Interval(i)
{ operacja ta jest wykonywana tylko dla i ∈ S }

Zaprezentujemy dwa rozwiązania. Pierwsze jest koncepcyjnie trudniejsze, ale pozwala lepiej zrozumieć wzbogacanie i strukturę drzew BST. Drugie rozwiązanie jest koncepcyjnie łatwiejsze, ale wymaga użycia dwóch drzew.

Rozwiązanie 1

Elementy zbioru S przechowujemy w zrównoważonym drzewie wyszukiwani binarnych (np. AVL, czerwono-czarnym) względem ich wartości. Dodatkowo z każdym węzłem pamiętamy jego wagę. To oczywiście wystarczałoby do wykonywania operacji Insert w czasie O(log |S|), ale mamy jeszcze dwie dodatkowe operacje Interval oraz Weight. W tym celu wzbogacimy nasze drzewo. Każdy węzeł w drzewie będzie miał trzy dodatkowe atrybuty:

w_sum – suma wag elementów w poddrzewie

el_count – liczba elementów (węzłów) w poddrzewie

min_el – minimalny element w poddrzewie (ostatni na skrajnie lewej ścieżce w tym poddrzewie)

max_el – maksymalny element w poddrzewie (ostatni na skrajnie prawej ścieżce w tym poddrzewie)

Po wstawieniu nowego elementu należy przejść w górę drzewa i poprawić wartości atrybutów na ścieżce do korzenia. Zauważmy, że wartość w_sum zmieni się w każdym węźle o różnicę nowej i starej wagi elementu i, argumentu Insert. Gdy i nie ma w drzewie, do w_sum dodajemy wagę i. Po zaktualizowaniu wartości atrybutów przywracamy zrównoważenie drzewa za pomocą rotacji. Zauważmy, że aktualizacja wartości atrybutów przy pojedynczej rotacji możliwa jest do wykonania w czasie stałym. Zatem Insert(i,w) wykonujemy w czasie O(log |S|).

Operacja Interval(i) polega na znalezieniu maksymalnego podzbioru zbioru S składającego się z kolejnych liczb całkowitych zawierającego i. Podzbiór wyznaczamy wskazując odpowiednio l i p – najmniejszy i największy element w tym podzbiorze. Innymi słowy szukamy maksymalnego przedziału liczb całkowitych [l,p], które należą do S. Po pierwsze i musi należeć do zbioru S. Wyszukujemy węzeł v zawierający i. Gdy takiego węzła nie ma odpowiadamy, że przedział [l,p] jest pusty. Pokażemy teraz, w jaki sposób znaleźć najmniejsze takie l w zbiorze S, że l, l+1, …, i są w S. Symetrycznie szukamy największe takie p, że i, i+1, …, p są w S. Pomocna będzie funkcja RightInterval(v) (podobnie LeftInterval(v)), która w poddrzewie o korzeniu v znajduje najmniejsze takie l' (największe tkaie p'), że w tym poddrzewie znajdują się elementy l', l'+1, …, v.max_el (v.min_el, v.min_el+1, …, p'). Funkcję RightInterval(v) można zapisać rekurencyjnie jak następuje:

```
RightInterval(v)::
{
   consider one of the cases{
      v nie ma dzieci: return v.el;
      v ma tylko prawe dziecko: if v.prawy.max_el = v.el + v.prawy.el_count then return v.el
                                    else return RightInterval(v.prawy);
      v ma tylko lewe dziecko: if v.lewy.max_el + 1 = v.el then return RightInterval(v.lewy)
                                    else return. v.el;
      v ma dwoje dzieci: if v.prawy.max_el = v.el + v.prawy.el_count then
                                    if v.lewy.max_el + 1 = v.el then return RightInterval(v.lewy)
                                    else return v.el
                                 else return RightInterval(v.prawy)
}
```

Czas działania RightInterval(v) wynosi O(wys. poddrzewa o korzenu v).

Żeby obliczyć teraz lewy koniec l przedziału zawierającego i rozważamy kolejno węzły v_0 = v, v_1, v_2, … na ścieżce z v do korzenia drzewa oraz takie, że v_0.el = i > v_1.el > v_2.el ….

Jeśli v nie ma lewego dziecka, to bierzemy l = i. Jeśli v ma lewe dziecko oraz v.lewy.min_el + v.lewy.el_count = i, to bierzemy l = v.min_el. Jeśli v jest korzeniem całego drzewa, kończymy obliczanie l. W przeciwnym razie sprawdzamy, czy lewy koniec przedziału nie może być mniejszy. W tym celu idziemy do węzła v_1. Jeśli okazało się jednak, że lewy.min_el + v.lewy.el_count < i, to lewy koniec przedziału zawierającego i jest po prostu w poddrzewie o korzeniu v. Gdy v.lewy.max_el + 1 < i, to i jest tym lewym końcem. W przeciwnym przypadku lewym końcem jest RightInterval(v.lewy). Nasze obliczenia kończymy w węźle v.

Załóżmy teraz, że przetworzyliśmy węzły v_0, v_1, …, v_{j-1}, jesteśmy w węźle v_j oraz że wszystkie elementy z przedziału [v_{j-1}.min_el,i] są w zbiorze S (w drzewie).

Terazsprawdzamy, czy lewy koniec przedziału zawierającego i może być mniejszy. Musimy rozważyć kilka przypadków:

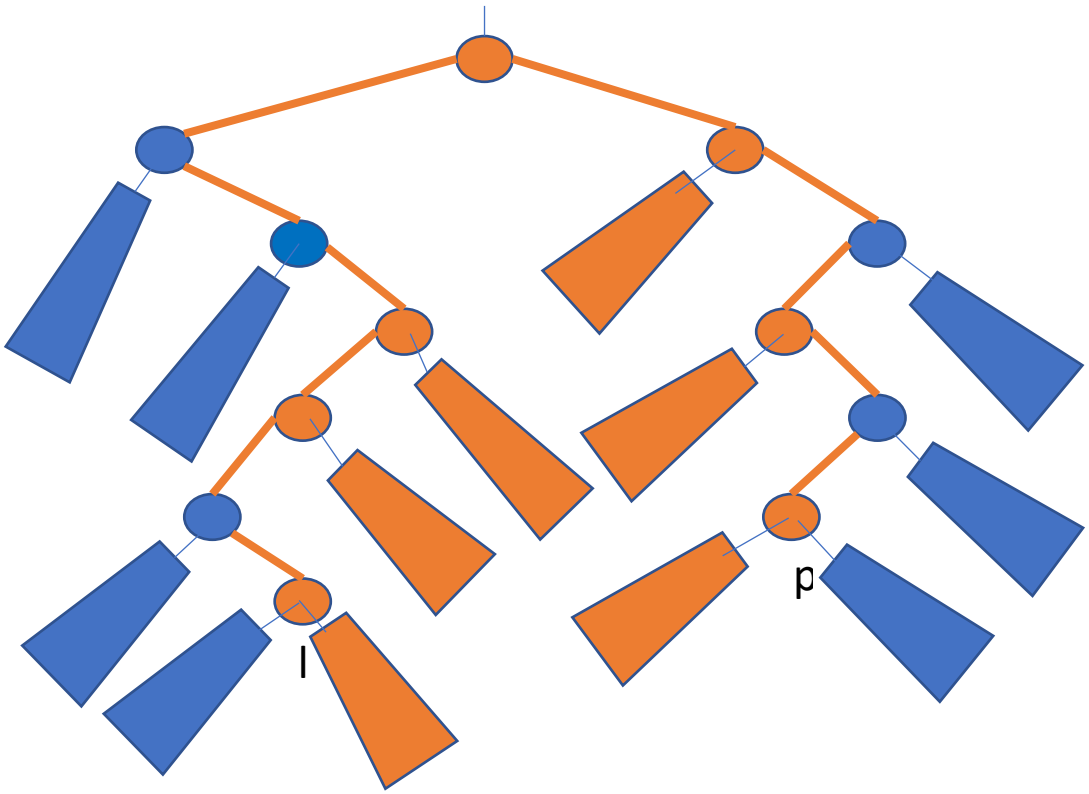
v_j.el + 1 < v_{j-1}.min_el: wynikiem jest l = v_{j-1}.min_el

v_j.el + 1 = v_{j-1}.min_el:

> v_j nie ma lewego dziecka: jeśli v_j jest korzeniem całego drzewa, to kończymy z l = v_j.el, w przeciwnym razie idziemy do v_{j+1}
>
> v_j ma lewe dziecko:
>
> > v_j.lewy.min_el + v_j.lewy.el_count = v_j.el: jeśli v_j jest korzeniem drzewa to kończymy z l = v_j.lewy.min_el, w przeciwnym razie idziemy do $v_{j+1;}$
> >
> > v_j.lewy.min_el + v_j.lewy.el_count < v_j.el:
> >
> > > l' := v_j.el; jeśli v_j.lewy.max_el + 1 = l' to l' := RightInterval(v_j.lewy);
> > >
> > > kończymy z l = l'

Węzłów do rozpatrzenia jest O(log |S|). Tylko dla co najwyżej jednego z nich wywołujemy RightInterval. Stąd poszukiwanie lewego końca l zabiera czas O(log |S|). W takim samym czasie znajdujemy prawy koniec p.

Jeśli znamy lewy i prawy koniec maksymalnego przedziału z S zawierającego i, to sumę wag na elementach tego przedziału liczymy podobnie jak maksimum z podciągu o indeksach od l do p z przykładu w wykładu:



Liczymy sumę wag elementów w pomarańczowych węzłach i wartości w_sum z korzeni pomarańczowych poddrzew.

Rozwiązanie 2

Zbiór S reprezentujemy za pomocą dwóch drzew zrównoważonych A i B. W drzewie A przechowujemy elementy zbioru S wraz z wagami. W drzewie B przechowujemy maksymalne przedziały elementów ze zbioru S wraz z sumą wag elementów w przedziałach. Przedziały w drzewie są uporządkowane względem lewych końców (równe dobrze można wziąć prawe końce). Z pomocą drzewa A łatwo sprawdzamy, czy element i jest w zbiorze oraz aktualizujemy jego wagę. Jeśli element i jest w S, to z pomocą drzewa B znajdujemy przedział zawierający i oraz zmieniamy sumę wag elementów w tym przedziale o różnicę miedzy nową i starą wagą elementu i. Jeżeli i nie było w S, to do drzewa A wstawiamy element i raz z wagą, tworzymy nowy przedział [a=i,b=i] o sumie wag s równej wadze i. Następnie wykonujemy, co następuje:

- jeśli w B jest przedział [a',i-1] o wadze s', to usuwamy ten przedział z B i za a bierzemy a' natomiast s przyjmuje wartość s + s';

- jeżeli w B jest przedział [i+1,b'] o wadze s'', to usuwamy ten przedział z B i b ustawiamy na b' natomiast za s bierzemy s + s'';

- wstawiamy przedział [a,b] wraz z sumą wag jego elementów s do B.

Wykonanie operacji Interval(i) oraz Weight(i) polega na wyszukaniu w B przedziału zawierającego i.

Czas działania poszczególnych operacji wynosi O(log |S|) – koszt operacji na zrównoważonym drzewie wyszukiwani binarnych.