Drzewo przedziałowe

Autor: Tomasz Idziaszek

Drzewo przedziałowe jest jedną z ważniejszych struktur danych w arsenale olimpijczyka wykorzystywaną w rozwiązaniach wielu zadań. Niemniej jednak informacji na jego temat próżno szukać w standardowych podręcznikach do algorytmów i publikacjach naukowych. To struktura danych typowo olimpijska: nadaje się do zastosowań, w których znane jest ograniczenie górne na rozmiar przechowywanych danych i w których zależy nam na prostocie implementacji. W podręcznikach częściej znajdziemy opisy zrównoważonych drzew binarnych, takich jak drzewa czerwono-czarne czy drzewa AVL.

Artykuł ten jest próbą zebrania i uporządkowania zagadnień dotyczących drzewa przedziałowego, które dotychczas były rozproszone w niebieskich książeczkach, ale zawiera również treści, które do tej pory nie doczekały się opisu w literaturze.

Struktury dla uporządkowanego multizbioru

Zaczniemy od najprostszego zastosowania drzewa przedziałowego, a mianowicie zaimplementujemy za jego pomocą strukturę danych umożliwiającą przechowywanie uporządkowanego multizbioru liczb całkowitych i wykonywanie na nim określonych operacji. W zależności od rodzaju potrzebnych nam operacji taką strukturę danych można zrealizować w mniej lub bardziej prosty sposób. Z tego też powodu przedstawimy aż cztery możliwe implementacje: za pomocą tablicy, kontenerów ze standardowej biblioteki C++ oraz dwóch szczególnych przypadków drzewa przedziałowego – drzewa licznikowego i drzewa potęgowego.

Multizbiór to uogólnienie zbioru, w którym wartości mogą się powtarzać. Przykładowo $\{1, 2, 4, 4, 4, 5\}$ jest multizbiorem, w którym występują wartości 1, 2, 4 i 5, przy czym wartość 4 występuje trzykrotnie. Multizbiór ten ma 6 elementów. Liczbę wystąpień wartości nazywamy jej krotnością (tak więc wartość 4 ma krotność 3, wartość 1 ma krotność 1, a wartość 3 ma krotność 0). Najbardziej podstawowe operacje, które potrzebujemy wykonywać, żeby w ogóle mówić o multizbiorze liczb, to:

- $init(S)$: utworzenie pustego multizbioru S;
- $insert(S, x)$: wstawienie nowego elementu o wartości x do multizbioru S;
- $remove(S, x)$: usunięcie jednego elementu o wartości x z multizbioru S;
- $count(S, x)$: podanie, ile elementów o wartości x znajduje się w multizbiorze S;
- $size(S)$: podanie liczby wszystkich elementów w multizbiorze S.

Reprezentacja tablicowa

Na początek przyjmijmy założenie, że wszystkie możliwe wartości, które możemy wstawiać do multizbioru, są liczbami całkowitymi z przedziału od 1 do n, przy czym wartość n jest na tyle nieduża, że pozwala nam użyć struktury danych wykorzystującej $O(n)$ komórek pamięci.

Najprostsza implementacja multizbioru polega na trzymaniu tablicy, w której zapisujemy krotności poszczególnych wartości multizbioru. Niech c_x oznacza krotność

wartości x (czyli liczbę elementów o wartości x) w multizbiorze S. Operacja $init(S)$ tworzy n-elementową tablicę $t[1\,..\,n]$, początkowo wypełnioną zerami. Będziemy utrzymywali niezmiennik, że $t[x] = c_x$ dla $1 \leqslant x \leqslant n$. Będziemy też osobno pamiętali wartość $C = \sum_{1 \leqslant i \leqslant n} c_i$.

Operacja $insert(S, x)$ zwiększa o jeden wartości $t[x]$ oraz C. Analogicznie operacja $remove(S, x)$ zmniejsza $t[x]$ oraz C o jeden. Mamy również $count(S, x) = t[x]$ oraz $size(S) = C$. Wszystkie te operacje działają w czasie stałym.

Kontenery z biblioteki standardowej

Jeśli multizbiór S jest niepusty, to możemy chcieć wykonywać dodatkowe operacje polegające na znajdowaniu wartości spełniających pewne kryteria:

- $minimum(S)$, $maximum(S)$: znalezienie najmniejszej lub największej wartości w multizbiorze S;
- $pred(S, x)$, $succ(S, x)$: znalezienie wartości poprzedzającej x (największej mniejszej od x) lub następującej po x (najmniejszej większej od x) w multizbiorze S.

W przypadku implementacji tablicowej takie operacje będą wymagały w pesymistycznym przypadku czasu $\Theta(n)$.

Ale jeśli programujemy w języku C++, to do zaimplementowania multizbioru możemy wykorzystać kontener `multiset` dostępny w standardowej bibliotece języka:

```
#include <set>
using namespace std;
multiset<int> S;
```

Podstawowe operacje przedstawione poprzednio wyglądają następująco:

Operacja	Kod
$init(S)$	(automatycznie)
$insert(S, x)$	`S.insert(x);`
$remove(S, x)$	`S.erase(S.find(x));`
$count(S, x)$	`return S.count(x);`
$size(S)$	`return S.size();`

Warto zwrócić uwagę, że operacja usunięcia to nie jest po prostu `S.erase(x)`, bo takie wywołanie usunie *wszystkie* wystąpienia wartości x z multizbioru.

Wszystkie te operacje (oprócz ostatniej) działają w czasie $O(\log C)$, gdzie C jest liczbą elementów w multizbiorze S. Operacja $size(S)$ działa w czasie stałym. Możemy też wykonać cztery nowe operacje, z których każda działa w czasie $O(\log C)$:

Operacja	Kod
$minimum(S)$	`return *S.begin();`
$maximum(S)$	`auto it = S.end();` `return *--it;`
$succ(S, x)$	`return *S.upper_bound(x);`
$pred(S, x)$	`auto it = S.lower_bound(x);` `return *--it;`