Drzewa poszukiwań binarnych

Kacper Pawłowski

Streszczenie

W tej pracy przedstawię zagadnienia związane z drzewami poszukiwań binarnych. Przytoczę poszczególne operacje na tej strukturze danych oraz ich złożoność obliczeniową. Przy okazji zaznajomię czytelnika z najważniejszymi pojęciami Teorii Grafów dotyczącymi tematyki drzew. Na końcu podam przykłady struktur opartych na drzewach BST.

1 Wprowadzenie

Drzewa poszukiwań binarnych *(ang. Binary Search Tree)* są strukturą danych służącą do przechowywania informacji w sposób uporządkowany. Wymogiem jest, aby każdy węzeł miał przypisaną jakąś wartość (tzw. klucz). Przy wyszukiwaniu danego elementu w drzewie stosując porównania kluczy wiadomo, w którym miejscu należy go poszukiwać. Za pomocą operacji In-Order-Tree-Walk możliwe jest wyświetlenie elementów od najmniejszego do największego w czasie $\Theta(n)$. Każde drzewo zawiera element główny od którego rozpoczynana jest "wędrówka" po tej strukturze danych. Ten element nazywamy korzeniem. Pojęcie drzew należy do dziedziny matematyki nazywanej Teorią Grafów (wielu zalicza ją do Matematyki dyskretnej) zapoczątkowanej przez Eulera w XVIII wieku. Czytelnik zainteresowany podstawami teorii grafów warto, aby zaznajomił się z książką [Wil12].

Definicja 1.1 *Drzewo binarne*
Drzewo binarne to drzewo, w którym stopień każdego wierzchołka jest mniejszy równy od 3.

Definicja 1.2 *Drzewo*
Graf $\mathbb{T}$ jest drzewem wtedy i tylko wtedy, gdy $\mathbb{T}$ jest spójny i nie zawiera cykli.

Definicja 1.3 *Graf spójny*
Graf jest spójny wtedy i tylko wtedy, gdy istnieje droga łącząca każde dwa dowolne wierzchołki.

Definicja 1.4 *Stopień wierzchołka*
Stopień wierzchołka (oznaczany poprzez deg) to liczba krawędzi wychodzących z danego wierzchołka.

2 Operacje na drzewie

`INSERT(T,x)` Wstawianie elementu x do drzewa $\mathbb{T}$
Jeżeli element x jest większy równy od węzła y to należy wybrać prawego syna, w przeciwnym wypadku syna lewego.

Require: T - drzewo, x - element do wstawienia

```
1: curr ← root(T)
2: while curr ≠ NULL do
3:    father ← curr;
4:    if x >= curr.value then
5:       curr ← curr.right
6:    else
7:       curr ← curr.left
8:    end if
9: end while
10: if father ≠ NULL then
11:    if x >= father.value then
12:       father.right.value ← x
13:    else
14:       father.left.value ← x
15:    end if
16: else
17:    father.value ← x
18: end if
```

`DELETE(T,x)` Usuwanie elementu x z drzewa $\mathbb{T}$

W przypadku operacji DELETE należy najpierw znaleźć przy pomocy SEARCH element, który chce się usunąć. Sprawdzić, którym synem swojego ojca jest ten element i podstawić za usuwany element odpowiadającego syna tego elementu, następnie dla drugiego syna należy odnaleźć nowe miejsce w drzewie.

`SEARCH(T,x)` Wyszukiwanie elementu x w drzewie $\mathbb{T}$

Require: T - struktura drzewa, x - poszukiwany węzeł

```
1: node ← root(T)
2: while node do
3:    if x = node.value then
4:       return node
5:    else
6:       if x >= node.value then
7:          node ← node.right
8:       else
9:          node ← node.left
10:      end if
11:   end if
12: end while
```

`MINIMUM(T)` Znajdowanie najmniejszego elementu w drzewie $\mathbb{T}$

Require: T - struktura drzewa

```
1: node ← root(T)
```

```
2: while node.left ≠ NULL do
3:    node ← node.left
4: end while
5: return node
```

`MAXIMUM(T)` Znajdowanie największego elementu w drzewie $\mathbb{T}$

```
Require: T - struktura drzewa
1: node ← root(T)
2: while node.right ≠ NULL do
3:    node ← node.right
4: end while
5: return node
```

`INORDERTREEWALK(T)` Wyświetlenie elementów w drzewie od najmniejszego do największego

```
1: node ← T
2: if node ≠ NULL then
3:    if node.left ≠ NULL then
4:       INORDERTREEWALK(node.left)
5:    end if
6:    Wyswietl node.value
7:    if node.right ≠ NULL then
8:       INORDERTREEWALK(node.right)
9:    end if
10: end if
```

3 Złożoność obliczeniowa

Najbardziej pesymistycznym przypadkiem dla operacji wykonywanych na drzewie jest przypadek, w którym drzewo jest izomorficzne z grafem liniowym oraz korzeń posiada tylko jednego, lecz konkretnego dla danej operacji, syna (w przypadku niektórych operacji jest to wręcz sytuacja najkorzystniejsza).

Definicja 3.1 *Graf liniowy*
Graf liniowy to graf powstały poprzez usunięcie dowolnej krawędzi z grafu cyklicznego (2-regularnego).

Dla operacji `MINIMUM` korzeń, który jest największym elementem drzewa w tym wypadku jest przypadkiem pesymistycznym o złożoności $O(n)$. Z kolei korzeń, który jest najmniejszym elementem drzewa, a więc nie posiada lewego syna, jest przypadkiem najkorzystniejszym. Analogicznie wygląda operacja `MAXIMUM`. Złożoność tych operacji dla danego drzewa jest rzędu $O(h)$, gdzie h jest wysokością drzewa.

Definicja 3.2 *Wysokość drzewa*
Wysokością drzewa nazywamy maksymalną liczbę wierzchołków, którą odwiedzamy wykonując tylko operacje przejścia na jednego z synów, od korzenia do danego liścia (bez liścia).

Własność 3.1 *Jeżeli graf* $\mathbb{T}$ *jest drzewem i zawiera* n *wierzchołków to* $\mathbb{T}$ *posiada* $n-1$ *krawędzi.*

W przypadku operacji wstawiania `INSERT` oraz operacji wyszukiwania `SEARCH` łatwo jest zauważyć, że podobnie jak w przypadku operacji `MINIMUM` oraz `MAXIMUM` złożoność pesymistyczna jest ograniczona poprzez $O(h)$, czyli w najbardziej pesymistycznym przypadku znów mamy $O(n)$ (a dla drzew pełnych $O(log_2 n)$). Nieco podobnie wygląda rzecz z operacją `DELETE`, gdyż wykorzystuje ona właściwie, z operacji bardziej czasochłonnych, tylko `SEARCH` oraz `INSERT`.

Oczekiwany wynik powyższych operacji to $O(log_2 n)$. Taki przypadek występuje dla drzew zrównoważonych i pełnych.

W przypadku operacji `INORDERTREEWALK` mamy do czynienia ze złożonością rzędu $\theta(n)$.

Fachowe dowody złożoności poszczególnych operacji można znaleźć na przykład w książce [CLRS12].

4 Możliwe ulepszenia i metody implementacji

Drzewa binarne mogą również zawierać mechanizmy odpowiedzialne za ich balansowanie. Przykładem takich drzew są drzewa czerwono-czarne (RB-Tree), w których po operacji INSERT są wykonane w razie potrzeby operacje rotacji (w lewo bądź prawo). W przypadku olbrzymiej ilości węzłów w drzewie warto, aby wysokość struktury danych była jak najmniejsza. W tym wypadku warto rozważyć implementację B-drzewa. Czytelnik zainteresowany tą tematyką znajdzie potrzebne informacje w [CLRS12].

Implementując drzewo binarne należy wpierw zastanowić się jak ta struktura ma być przechowywana w pamięci komputera. Najbardziej popularne są dwie metodyki implementacji. Jedna to dynamiczne alokowanie potrzebnej pamięci i wykorzystanie wskaźników, a druga nieco łatwiejsza, ale narzucająca pewne ograniczenia pamięciowa to metoda trzymania drzewa w tablicy. Wtedy elementem zerowym tablicy jest korzeń, lewy syn ma numer 1, prawy syn numer 2. Dzieci lewego syna to elementy 3 i 4 itd. Warto przy tym pamiętać, że element i może mieć dwóch synów oznaczanych jako $2i+1$ oraz $2i+2$. Ojcem elementu j jest element $\lfloor \frac{i-1}{2} \rfloor$.

5 Podsumowanie

Drzewa poszukiwań binarnych to jedna z podstaw przy studiowaniu algorytmów. Standardową implementację można ulepszać poprzez stosowanie wszelkich rotacji. Najważniejszą cechą drzew są ich własności logarytmiczne (występujące w

przypadkach oczekiwanych) związane z wysokością. Właśnie dzięki temu drzewa znacząco ułatwiają poszukiwania wybranych elementów. W zasadzie umożliwiają przechowywanie dowolnych danych, przy których stosuje się skalę porównawczą. Wykorzystując BST można oprogramować np. kolejkę priorytetową (ang. Priority Queue), czy strukturę umożliwiającą w czasie $\Theta(n)$ wyświetlenie wprowadzonych danych w porządku niemalejącym.

Literatura

[CLRS12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Wprowadzenie do algorytmów*. Wydawnictwo Naukowe PWN, 2012.

[Wil12] Robin J. Wilson. *Wprowadzenie do teorii grafów*. Wydawnictwo Naukowe PWN, 2012.