Lista 7 zadanie 3

June 1, 2020

1 Wstęp

1.1 Binary Search Tree

BST (Binary Search Tree) jest to dynamiczna struktura danych będąca drzewem binarnym, w którym lewe poddrzewo każdego węzła zawiera wyłącznie elementy o kluczach mniejszych niż klucz węzła a prawe poddrzewo zawiera wyłącznie elementy o kluczach nie mniejszych niż klucz węzła. Węzły, oprócz klucza, przechowują wskaźniki na swojego lewego i prawego syna oraz na swojego ojca.

1.2 Binary heap

Binary heap (kopiec binary, zwany też stertą) – tablicowa struktura danych reprezentująca drzewo binarne, którego wszystkie poziomy z wyjątkiem ostatniego muszą być pełne. W przypadku, gdy ostatni poziom drzewa nie jest pełny, liście ułożone są od lewej do prawej strony drzewa. Wyróżniamy dwa rodzaje kopców binarnych: kopce binarne typu max w których wartość danego węzła niebędącego korzeniem jest zawsze mniejsza niż wartość jego rodzica oraz kopce binarne typu min w których wartość danego węzła niebędącego korzeniem jest zawsze większa niż wartość jego rodzica.

2 Rozwiązanie

Operacje:

- Insert (wstawiamy klucz jako nowy węzeł najniższym poziomie tak, aby była spełniona własność BST i jeśli trzeba, to odtwarzamy strukturę zbalansowanego drzewa)
- Minimum (idziemy "w lewo" aż napotkamy na węzeł nie mający synów)
- Delete (znajdujemy klucz o zadanej wartości, następnie usuwamy go z drzewa, podpinając dzieci Node'a pod jego rodzica)

wykorzystuja standardowa implementacje BST (spełniają one warunki kolejki priorytetowej). Optymalna złożoność obliczeniowa tych funkcji to $O(\log n)$. (korzystając z implementacji drzew czerwono czarnych albo drzew AVL).

Operacja ExtractMin analogicznie do Minimum, przy czym nalezy wykorzystać też operację Delete (*ekstracja* elementu z drzewa). Złożonośc obliczeniowa wynosi $O(\log n)$, ponieważ taka jest złożonośc operacji Min jak również Delete.

W przypadku DecreaseKey wykorzystywane są operacje Delete i Insert (usunięcie elementu i wstawienie nowego ze zmniejszonym priorytetem). Złożoność obliczeniowa tych operacji wynosi $O(\log n)$.

Operacja Union w obu przypadkach połączy dwie zadane struktury w jedną. W przypadku binary heapa mamy doczynienia z tablicami (standardowa implementacja). Tworzymy więc nową tablicę wielkości n+m (suma wielkości pomniejszych tablic) i przekopiowujemy do niej wartości z tablic n i m elementowej, co daje algorytm o złożoności liniowej. Na koniec na nowoutworzonej tablicy wykonujemy operację BuildHeap o koszcie O(n+m)

Dla drzew BST również możemy otrzymać taką złożoność. W tym celu tworzymy posortowane tablice otrzymane w wyniku Inorder Tree Traversal, a następnie budujemy z nich zbalansowane BST. Algorytm tej operacji otrzymuje na wejściu posortowaną tablicę a następnie: Czas działania inorder tree traversal jest liniowy względem liczby węzłów

1. Bierzemy środkowy element posortowanej tablicy i ustawiamy go jako root.
2. Rekurencyjnie (póki wejściowa tablica ma długość > 1):
   (a) Wywołujemy tę funkcję dla prawego dziecka, biorąc za input prawą połowę wejściowej tablicy
   (b) Wywołujemy tę funkcję dla lewego dziecka, biorąc za input lewą połowę wejściowej tablicy

Złożoność tego algorytmu w tym przypadku (jak również i dla kopca) wynosi asymptotycznie $O(n)$. // Gdy mamy łącznie n+m elementów, to łączny koszt to O(n+m)

3 Złożoności obliczeniowe

Złożoności obliczeniowe zarówno dla kopca jak i BST są asympototycznie takie same dla operacji:

- Insert
- Min
- ExtractMin
- DecreaseKey

- Delete

Złożoność w przypadku obu implementajcji wynosi $O(\log n)$ Dla Union w obu implemetacjach – $O(n)$.
Róznica występuje w przypadku operacji Minimum ($O(\log n)$ dla BST i $O(1)$ dla kopca binarnego.
Ważną obserwacją jest także to, że kopiec binarny możemy bardzo efektywnie zaimplementować przy użyciu "zwykłej" tablicy, a implementacja zbalansowanych drzew BST - choć asymptotycznie równie efektywna - jednak wymaga większej liczby operacji, a co za tym idzie, bardziej skomplikowanej implementacji. (te stałe ukryte w notacji O-duże są większe - weźmy np. RB-drzewa i operacje Insert czy Delete - tam dochodzi dodatkowy narzut związany z utrzymywaniem zbalansowanej struktury)
Poza tym n-elementowy kopiec binarny jesteśmy w stanie zbudować z losowej tablicy w czasie liniowym (por. rozwiązanie zadania 2), a utworzenie odpowiedniego drzewa binarnego wymaga czasu O(n log n).

Na korzyść drzewa przemawiają natomiast fakty, że wyszukiwanie elementu jest szybsze ($O(\log n)$ zamiast $O(n)$ dla binary heap), jak również wyświetlanie posortowanych elementów (dla BST $O(\log n)$, dla binary heap – $O(n \log n)$.

Obie struktury przy implementacji kolejki priorytetowej mają swoje wady i zalety, stąd przy konkretnej implementacji należy zwracać uwagę na których feature'ach nam bardziej zależy.