AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Implementacja drzewa rozpinającego w C++

Autor:
Jakub Piwko
Łukasz Nowak

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2023

Spis treści

1. Ogólne określenie wymagań

Celem tego zadania jest napisanie programu „drzewo BST" działającego na stercie w języku C++.
Drzewo powinno być zaimplementowane w klasie. Funkcjonalność (metod) drzewa:
• Dodaj element
• Usuń element
• Usuń całe drzewo
• Szukaj drogi do podanego elementu
• Wyświetl drzewo

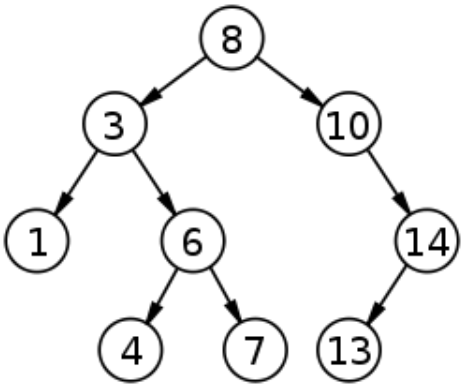
W drugiej klasie należy zaimplementować (metody):
•Zapis do pliku
•Odczyt z pliku utworzonego drzewa BTS (plik ma być zapisany binarnie).

Oprócz tego należy mieć możliwość wczytania pliku tekstowego z cyframi. Dzięki temu daje to możliwość zbudowania drzewa. Program powinien posiadać możliwość wczytania pliku z liczbami do drzewa pustego lub istniejącego.

Funkcja main powinna wyświetlać menu z opcjami drzewa, odczytu oraz zapisu pliku, a także opcję do wyłączenia programu. Program czeka na wybranie opcji. Wszystkie utworzone klasy mają być zaimplementowane w oddzielnych plikach w tym funkcja main.

2. Analiza problemu

Binarne drzewo poszukiwań (Binary Search Tree, BST) – 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 elementy znajdujące się poniżej ich: po lewej oraz po prawej, a także wskaźnik na element znajdujący się powyżej. Działanie przykładowego drzewa BST możemy zauważyć na rysunku 2.1.



Rys. 2.1. Przykładowe drzewo BST

Drzewo BST pozwala na szybkie i efektywne wyszukiwanie elementów, ponieważ elementy zależnie od ich wartości zostają utworzone po lewej lub prawej stronie głównego węzła:
- jeżeli element ma wartość mniejszą od głównego węzła znajdzie się on po jego lewej stronie
- jeżeli element ma wartość większą od głównego węzła znajdzie się on po jego prawej stronie

Ustawienie wartości w drzewie w ten sposób znacznie skraca czas wyszukiwania elementów o wybranych przez nas wartościach. Sposób rozmieszczenie tych wartości można zaobserwować na rysunku 2.1.

3. Projektowanie

Do wykonania tego zadania użyte zostało opgrogramowanie:
- Visual Studio Code
- Github + Git (w celu kontroli wersji naszego programu)
- Język Programowania C++
- Overleaf (LateX)

3.1. Visual Studio Code

Visual Studio Code (VS Code) to darmowy, Open-Source edytor tekstu stworzony przez firmę Microsoft. Jest to popularne narzędzie programistyczne, które oferuje wiele funkcji przydatnych dla programistów i deweloperów.
VS Code charakteryzuje się prostotą w dodawaniu rozszerzeń ułatwiających pracę z tym programem oraz wieloma typami projektów niezależnie od języka. Ma także wbudowany system współpracy z Git'em oraz GitHub'em.

3.2. Github + Git

Git to system kontroli wersji używany do śledzenia zmian w kodzie źródłowym i współpracy nad projektem programistycznym.

Kontrola Wersji: Git umożliwia programistom śledzenie zmian w kodzie źródłowym, przechowywanie historii zmian i przywracanie poprzednich wersji projektu.

Rozproszony System Kontroli Wersji: Git jest rozproszonym systemem, co oznacza, że każdy programista ma pełną kopię repozytorium na swoim komputerze, co ułatwia niezależne pracę i umożliwia pracę offline.

Branching i Merging: Git pozwala na tworzenie oddzielnych gałęzi (branch) projektu, co umożliwia pracę nad różnymi funkcjonalnościami niezależnie. Następnie można połączyć (merge) te gałęzie w jedną całość.

Współpraca: Git ułatwia współpracę zespołową, umożliwiając wielu programistom równoczesną pracę nad projektem i rozwiązywanie konfliktów wersji.

Platforma Niezależna: Git jest dostępny na różnych platformach (Windows,

macOS, Linux) i można go wykorzystywać w różnych narzędziach i środowiskach programistycznych.

Bezpieczeństwo: Repozytoria Git są zabezpieczone i chronione, co zapewnia bezpieczne przechowywanie kodu źródłowego.

Społeczność: Git ma ogromną społeczność użytkowników i jest szeroko używany w całym świecie, co oznacza, że dostępne są liczne źródła wsparcia i dokumentacji.

3.3. Język programowania C++

Jest używany w wielu dziedzinach, od tworzenia oprogramowania do gier, aplikacji biznesowych, systemów wbudowanych po rozwiązywanie algorytmicznych problemów. Jest językiem, który wymaga od programistów solidnej wiedzy i umiejętności, ale oferuje dużą elastyczność i kontrolę nad kodem źródłowym.
C++ jest językiem kompilowanym, co oznacza, że przekształca kod źródłowy w efektywny kod maszynowy, co sprawia, że jest szybki i wydajny.
C++ jest językiem programowania obiektowego, co oznacza, że wspiera tworzenie obiektów, które zawierają dane i funkcje operujące na tych danych. To podejście ułatwia projektowanie i zarządzanie kodem.

4. Implementacja

Implementacja programu opisującego Drzewo BST, która działa na stercie, skupia się na utworzeniu 2 klas oraz pliku main:
Zadaniem klasy 1 jest wykonywanie podstawowych operacji na drzewie BST:
- Tworzenie drzewa
- Dodawanie elementów
- Usuwanie elementów
- Usuwanie drzewa
- Wyświetlanie drzewa

Zadaniem klasy 2 jest:
- Zapisywanie Drzewa BST do pliku
- Odczytywanie zapisanych Drzew BST z pliku

Zadaniem pliku main jest:
Obsługa całego programu poprzez utworzone do tego menu, które pozwalać nam będzie na wykonywanie wszystkich z wcześniej opisanych funkcji oraz możliwość wyjścia z programu.
Oto ogólny opis implementacji programu:

4.1. Klasa Tree:

Program rozpoczyna się od zdefiniowania klasy `Tree`, która reprezentuje Drzewo BST. W klasie tej są przechowywane wskaźniki na wartość (`data`) element po lewej (`left`) oraz element po prawej (`right`), które wskazują na elementy znajdujące się poniżej po lewej oraz po prawej w drzewie. Klasa ta zawiera metody do operacji na Drzewie BST.

```
1     // Klasa Tree
2
3     class Tree {
4 public:
5     int data;
6     Tree* left;
7     Tree* right;
8
9     // Konstruktor
```

```
10      Tree(int value) : data(value), left(nullptr), right(nullptr) {}
11
12  };
```

Listing 1. Klasa Tree

4.2. Dodawanie elementów:

Funkcja zaczyna od sprawdzenie czy drzewo BST jest puste, jeżeli jest ono puste to tworzone jest nowe drzewo, jeżeli drzewo posiada inne elementy wprowadzona przez nas wartość porównywana jest z głównym węzłem. W zależności od tego czy jego wartość jest mniejsza lub większa od nowej wartości jest ona wprowadzana w odpowiednie dla niej miejsce. Kod wykonujący tę funkcję można zaobserwować na listingu 2.

```
1     // Dodawanie elementu do drzewa BST
2     void insertNode(Tree*& root, int value) {
3     if (root == nullptr) {
4         root = new Tree(value);
5     } else {
6         if (value < root->data) {
7             insertNode(root->left, value);
8         } else if (value >= root->data) {
9             insertNode(root->right, value);
10        }
11    }
12    }
```

Listing 2. Dodawanie elementów

4.3. Usuwanie elementów:

Funkcja zaczyna od sprawdzenia czy drzewo BST nie jest puste, w przypadku gdy jest puste funkcja kończy prace. Jeżeli drzewo nie jest puste zaczyna się przeszukiwanie drzewa w celu znalezienia elementu o wartości przez nas podanej, po znalezieniu elementu wydarzy się jeden z 2 możliwych przypadków.

Przypadek 1:

Element jest liściem (nie posiada elementów pod sobą) lub element posiada nie więcej niż jeden element poniżej siebie. W tym przypadku element jest usuwany, a jego miejsce zastępuje element poniższy (jeżeli taki istnieje).

Przypadek 2:

Element posiada dwa elementy poniższe. W tym przypadku potrzebna jest nam metoda do wybranie elementu poniższego o najmniejszej wartości. W naszym przypadku jest to funkcja findNode jak widzimy na listingu 3, przy jej użyciu wybieramy mniejszy z elementów, a następnie usuwamy wyszukany element i zastępujemy go elementem znalezionym przez findNode.

```
1    // Znajdowanie najmniejszego z potomkow usuwanego wezla |
   Potrzebne do metody deleteNode
2    Tree* findNode(Tree* node) {
3        while (node->left != nullptr) {
4            node = node->left;
5        }
6        return node;
7        }
8
9    // Usuwanie elementu drzewa BST po wartosci
10   Tree* deleteNode(Tree* root, int value) {
11       if (root == nullptr) {
12           return root;
13       }
14       if (value < root->data) {
15           root->left = deleteNode(root->left, value);
16       } else if (value > root->data) {
17           root->right = deleteNode(root->right, value);
18       } else {
19
20       // Przypadek 1: Wezel ma co najwyzej jedno dziecko
21       if (root->left == nullptr) {
22           Tree* temp = root->right;
23           delete root;
24           return temp;
25       } else if (root->right == nullptr) {
26           Tree* temp = root->left;
27           delete root;
28           return temp;
29       }
30
31       // Przypadek 2: Wezel ma dwoje dzieci
32       Tree* temp = findNode(root->right);
33       root->data = temp->data;
34       root->right = deleteNode(root->right, temp->data);
35   }
```

```
36      return root;
37      }
```

Listing 3. Usuwanie elementów

4.4. Usuwanie całego drzewa:

Funkcja ta zaczyna od sprawdzenia czy drzewo BST nie jestu puste, jeżeli jest puste funkcja się kończy, a w przeciwnym przypadku zaczyna od usunięcia wszystkich węzłów po lewej stronie, następnie usuwa węzły po prawej stronie i na sam koniec usuwa węzeł główny. Kod tej funkcji możemy zaobserwować na listingu 4.

```
1       // Usuwanie drzewa BST przy uzyciu deleteTree || Problem z
        dodawaniem elementu po usunieciu drzewa
2       void deleteTree(Tree* root) {
3       if (root == nullptr) {
4           return;
5       }
6
7       // Najpierw usun lewe poddrzewo
8       deleteTree(root->left);
9
10      // Nastepnie usun prawe poddrzewo
11      deleteTree(root->right);
12
13      // Ustawienie wezla na pusty
14      delete root;
15      root = nullptr;
16      }
```

Listing 4. Usuwanie całego drzewa

4.5. Wyświetlanie drzewa:

Funkcja ta używana jest do Wyświetlania wszystkich elementów drzewa za pomocą InOrder. Funkcja zaczyna od sprawdzenia czy drzewo nie jest puste, jeżeli jest puste kończy ona swoją prace, a w przeciwnym przypadku zaczyna od wypisanie elementów znajdujących się po lewej stronie węzła głównego, następnie wypisuje węzeł główny, a na koniec wypisuje elementy znajdujące się po prawej stronie węzła głównego. Kod tej funkcji możemy zaobserwować na listingu 5.

```
    // Wyswietlenie drzewa BST przy uzyciu InOrder
    void printInOrder(Tree* root) {
    if (root == nullptr) {
        return;
    }

    // Najpierw wypisz lewe poddrzewo
    printInOrder(root->left);

    // Nastepnie wypisz wartosc biezacego wezla
    cout << root->data << " ";

    // Na koniec wypisz prawe poddrzewo
    printInOrder(root->right);
    }  return;
    }
```

Listing 5. Wyświetlanie drzewa

4.6. Testowanie w funkcji main:

Implementacja klasy jest testowana w funkcji `main`, gdzie tworzona jest instancja klasy `Tree`, a następnie wywoływane są różne operacje na drzewie, takie jak dodawanie, usuwanie i wyświetlanie elementów.

5. Wnioski

5.1. Github + Git

Narzędzia takie jak oprogramowanie kontroli wersji znacznie ułatwia proces prowadzenia projektu oraz monitorowania zmian w kodzie bez przeglądania każdego pliku po kolei. Pozwala ono na wygodne kontynuowanie pracy bez obawy o utracie postępu w razie wystąpienia niespodziewanych problemów.

5.2. Drzewo BST

Drzewo binarne to struktura danych używana w informatyce i matematyce. Jest to rodzaj drzewa, w którym każdy węzeł może mieć co najwyżej dwóch potomków: lewego i prawego. Drzewa binarne są szeroko wykorzystywane w algorytmach i strukturach danych do rozwiązywania różnych problemów. W drzewie binarnym elementy mniejsze od głównego węzła zostają umieszczone po jego lewej, a elementy większe po jego prawej, rozmieszczenie elementów w ten sposób znacznie przyśpiesza czas wyszukiwania elementów w drzewie binarnym.
Drzewa binarne są często używanymi strukturami z powodu uporządkowania danych w konkretny sposób co pozwala skrócić czas poszukiwania o ponad połowę względem struktur typu lista czy tabela.

Spis rysunków

Spis tabel

Spis listingów