AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Drzewo Przeszukiwań Binarnych (BST)

Autor:
Marcin Dudek
Mateusz Basiaga

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Celem projektu jest stworzenie aplikacji w języku C++ umożliwiającej obsługę drzewa wyszukiwań binarnych (BST - Binary Search Tree) oraz umożliwiającej jego zarządzanie i wizualizację. Projekt realizowany jest z myślą o zdobyciu praktycznego doświadczenia z systemem kontroli wersji GitHub, przy pracy w grupie dwuosobowej, z zastosowaniem narzędzi do zarządzania kodem i dokumentacją.

1.1. Zakładany efekt końcowy

Efektem końcowym pracy jest aplikacja spełniająca wymagania funkcjonalne oraz strukturalne określone w specyfikacji. Działanie programu powinno umożliwić:

- Dodawanie, usuwanie oraz przeszukiwanie elementów w drzewie binarnym;
- Wyświetlanie struktury drzewa w różnych formach porządkowania (preorder, inorder, postorder);
- Zapis i odczyt drzewa do/z pliku binarnego oraz tekstowego, co pozwoli na ponowne załadowanie drzewa bez utraty danych;
- Wczytanie danych z pliku tekstowego i ich przekształcenie w strukturę drzewa BST, zarówno do pustego drzewa, jak i z możliwością łączenia z już istniejącą strukturą.

1.2. Cele techniczne

Projekt został zaplanowany tak, aby spełnić poniższe cele techniczne:

- Implementacja klasy `BinarySearchTree` realizującej główne operacje na drzewie BST: dodawanie, usuwanie, przeszukiwanie, wyświetlanie oraz zapis i odczyt z pliku;
- Stworzenie dodatkowej klasy `App` odpowiedzialnej za zarządzanie drzewem poprzez menu interaktywne, obsługujące wybory użytkownika oraz wywołujące odpowiednie operacje na drzewie;
- Rozdzielenie kodu źródłowego na moduły zgodnie z zasadami programowania obiektowego, w tym podział na pliki nagłówkowe i implementacyjne, aby ułatwić dalszy rozwój i testowanie kodu;

- Wykorzystanie narzędzi GitHub do wersjonowania kodu, pracy równoległej oraz rozwiązywania konfliktów;
- Stworzenie dokumentacji projektu z wykorzystaniem narzędzia `Doxygen` oraz dokumentacji w LaTeX, aby opisać implementację, działanie oraz wyniki końcowe projektu.

1.3. Wymagania dotyczące kontroli wersji i pracy zespołowej

W projekcie przewiduje się równoległą pracę dwuosobową, co wiąże się z określonymi wymaganiami:

- Utworzenie repozytorium GitHub, które będzie śledzić postępy prac nad projektem;
- Każdy z członków zespołu będzie odpowiedzialny za stworzenie nowej gałęzi w projekcie, wykonanie co najmniej 5 commitów i późniejsze połączenie tych zmian z główną gałęzią (tzw. `merge`);
- Projekt wymaga rozwiązania co najmniej 6 konfliktów w kodzie podczas łączenia gałęzi, co pozwala zdobyć doświadczenie w rozwiązywaniu problemów wynikających z pracy równoległej w zespole;
- Zastosowanie narzędzi GitHub do przeglądu kodu, rozwiązywania konfliktów oraz dokumentowania zmian przy pomocy opisowych commitów.

1.4. Oczekiwane wyniki

Projekt ma na celu uzyskanie aplikacji, która pozwoli użytkownikom na pełną kontrolę nad drzewem BST poprzez interfejs tekstowy. Przyjęte rozwiązania mają umożliwić:

- Poprawne zarządzanie drzewem BST, w tym obsługę operacji modyfikujących i przeszukujących strukturę drzewa;
- Wykorzystanie plików binarnych i tekstowych do utrwalenia stanu drzewa między sesjami programu;
- Wytworzenie dokumentacji projektowej oraz technicznej, która opisze szczegóły implementacyjne oraz przedstawi wykresy procesu pracy grupowej i historii commitów z GitHuba.

2. Analiza problemu

Drzewa wyszukiwań binarnych (BST - Binary Search Trees) są strukturami danych szeroko stosowanymi w informatyce ze względu na ich efektywność w przechowywaniu danych i szybkim dostępie do nich. Celem tego rozdziału jest analiza kontekstu, w którym algorytm BST znajduje zastosowanie, omówienie jego działania oraz przedstawienie praktycznego przykładu implementacji i użytkowania.

2.1. Zastosowanie algorytmu BST

Drzewo wyszukiwań binarnych jest używane wszędzie tam, gdzie konieczne jest szybkie przechowywanie i przetwarzanie danych uporządkowanych. Przykłady zastosowań BST obejmują:

- **Bazy danych:** BST są często używane w strukturach danych baz danych, gdzie kluczowe jest szybkie wyszukiwanie rekordów.
- **Systemy plików:** W systemach plików drzewa wyszukiwań mogą służyć do organizacji katalogów, ułatwiając wyszukiwanie i zarządzanie plikami.
- **Słowniki i systemy słów kluczowych:** W aplikacjach, które operują na dużych zbiorach słów, takich jak słowniki czy autokorekta, BST zapewniają szybki dostęp do danych.
- **Struktury indeksowe:** BST są podstawą wielu struktur indeksowych wykorzystywanych w algorytmach i systemach, które potrzebują sortowanego dostępu do danych.

2.2. Sposób działania algorytmu

Algorytm drzewa BST jest oparty na zasadzie rekurencyjnej struktury, gdzie każdy węzeł drzewa przechowuje element z kluczem oraz dwa wskaźniki - lewy i prawy. Struktura BST zapewnia, że dla każdego węzła wszystkie klucze w lewym poddrzewie są mniejsze od klucza w węźle, a wszystkie klucze w prawym poddrzewie są większe. Dzięki tej strukturze BST umożliwia szybkie przeszukiwanie drzewa.

Podstawowe operacje na drzewie BST to:

- **Dodawanie elementu:** Przeszukując drzewo od korzenia, porównujemy wartość klucza nowego elementu z wartościami istniejących węzłów, aż znajdziemy odpowiednie miejsce do jego dodania jako liść drzewa.

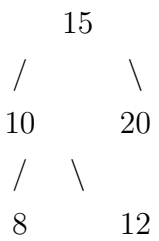
- **Usuwanie elementu:** Usunięcie węzła wymaga rozważenia trzech przypadków: usuwanie liścia, węzła z jednym potomkiem lub węzła z dwoma potomkami.
- **Przeszukiwanie drzewa:** Przechodzimy po węzłach drzewa, porównując wartości w celu znalezienia ścieżki do określonego elementu.
- **Wyświetlanie drzewa:** Istnieją różne sposoby wyświetlania drzewa, takie jak preorder, inorder, i postorder, które odwiedzają węzły w określonej kolejności.

2.3. Przykład działania algorytmu BST

Poniżej znajduje się przykładowa operacja dodawania elementów do drzewa BST. Załóżmy, że mamy pustą strukturę i chcemy dodać elementy w kolejności: 15, 10, 20, 8, 12.

1. Dodajemy element **15** jako korzeń drzewa, ponieważ jest to pierwszy element.
2. Element **10** jest mniejszy niż 15, więc staje się lewym potomkiem korzenia.
3. Element **20** jest większy niż 15, dlatego staje się prawym potomkiem korzenia.
4. Element **8** jest mniejszy niż 15 oraz mniejszy niż 10, więc zostaje lewym potomkiem węzła o wartości 10.
5. Element **12** jest mniejszy niż 15, ale większy niż 10, więc zostaje prawym potomkiem węzła o wartości 10.

Po dodaniu tych elementów, BST przyjmuje następującą formę:



2.4. Git - narzędzie do kontroli wersji

Git jest narzędziem umożliwiającym zarządzanie wersjami kodu źródłowego w zespołach programistycznych, co jest kluczowe dla poprawnego zarządzania projektem, szczególnie przy współpracy zespołowej. W projekcie tym, Git pozwala na:

- **Równoległą pracę na gałęziach:** Każdy członek zespołu może pracować niezależnie na swojej gałęzi, a po zakończeniu pracy zmiany są scalane do gałęzi głównej.
- **Śledzenie historii commitów:** Każda zmiana w kodzie jest zapisywana w historii, co pozwala na łatwe cofanie zmian lub śledzenie poprawek.
- **Rozwiązywanie konfliktów:** Git umożliwia kontrolę nad sytuacjami, gdy zmiany w kodzie kolidują, pozwalając użytkownikom na ręczne rozwiązywanie konfliktów i sprawdzanie ich.

W kolejnym rozdziale omówimy szczegóły implementacyjne, takie jak struktura klas i operacje realizowane przez algorytm BST w naszej aplikacji.

3. Projektowanie

W niniejszym rozdziale przedstawimy szczegółowy opis narzędzi i technologii, które zostały wykorzystane do realizacji projektu, a także opis procesu projektowego.

3.1. Narzędzia i technologie

Do stworzenia projektu użyte zostały następujące technologie i narzędzia:

- **Język programowania:** C++ - Został wybrany jako główny język programowania ze względu na jego wydajność oraz obsługę struktur danych, takich jak drzewa.
- **Kompilator:** GCC oraz MINGW-w64 (MSYS2) - Kompilator C++ do kompilacji programu na systemy Linux oraz Windows. W systemie Linux wykorzystywana jest wersja GCC, a w systemie Windows wersja MINGW-w64.
- **System budowania:** CMake oraz Ninja - CMake służy jako system generowania plików Makefile, podczas gdy Ninja jest używane do budowania projektu w sposób szybszy i bardziej efektywny.
- **Kontrola wersji:** Git - Git jest używany do zarządzania kodem źródłowym oraz współpracy w zespole. Repozytorium znajduje się na platformie GitHub.
- **Automatyczna dokumentacja:** Doxygen - Służy do generowania dokumentacji automatycznej z kodu źródłowego.
- **Dokumentacja ręczna:** LaTeX - Dokumentacja projektu jest tworzona przy użyciu szablonu LaTeX dostosowanego przez mgr inż. Dawida Kotlarskiego.
- **Automatyczne testowanie i integracja:** GitHub Actions - Używane do automatycznego testowania oraz integracji kodu. Zawiera również mechanizmy Continuous Integration.
- **Weryfikacja commitów:** Commitlint - Narzędzie do walidacji wiadomości commitów zgodnych z konwencją *Conventional Commits*.

3.2. Konfiguracja kompilatora i środowiska

Projekt jest skonfigurowany do kompilacji zarówno na systemach Linux jak i Windows. Aby móc efektywnie pracować w obydwu środowiskach, zastosowane zostały odpowiednie konfiguracje:

- **Kompilacja dla Linux (GCC):** Konfiguracja CMake generuje pliki Makefile, które są następnie używane przez GCC w systemie Linux. Kompilacja odbywa się za pomocą polecenia `make`, co pozwala na szybkie generowanie plików wykonywalnych.

- **Kompilacja dla Windows (MINGW-w64):** Aby umożliwić kompilację w systemie Windows, używamy wersji MINGW-w64 w połączeniu z MSYS2, które zapewnia środowisko linuksowe w systemie Windows. Kompilacja odbywa się przy użyciu polecenia `mingw32-make`.

3.3. Struktura projektu

Struktura katalogów projektu jest następująca:

- `src/` - Główny katalog zawierający kod źródłowy.
- `include/` - Katalog z plikami nagłówkowymi.
- `docs/` - Katalog z dokumentacją wygenerowaną przez Doxygen oraz LaTeX.
- `build/` - Katalog używany do przechowywania plików generowanych przez system budowania (np. pliki obiektowe, pliki wykonywalne).
- `tests/` - Katalog zawierający testy jednostkowe.

3.4. Diagramy i schematy

W projekcie zastosowano klasy `BinarySearchTree` oraz `App`. Klasa `BinarySearchTree` odpowiada za przechowywanie struktury drzewa oraz wykonanie operacji na drzewie (dodawanie, usuwanie, przeglądanie). Klasa `App` zapewnia interfejs użytkownika oraz umożliwia zarządzanie drzewem za pomocą menu.

- **Klasa BinarySearchTree:** Przechowuje dane drzewa oraz implementuje operacje na nim, takie jak dodawanie elementów, usuwanie, przeglądanie w różnych metodach (preorder, inorder, postorder).

- **Klasa App:** Zapewnia menu, które umożliwia użytkownikowi interakcję z drzewem: dodawanie, usuwanie elementów oraz wyświetlanie struktury drzewa.

3.5. Git i narzędzia CI/CD

Projekt oparty jest na systemie kontroli wersji Git, z repozytorium umieszczonym na platformie GitHub. GitHub Actions jest wykorzystywane do automatyzacji procesu budowania i testowania projektu. Do weryfikacji wiadomości commitów stosowana jest konwencja *Conventional Commits* i narzędzie `Commitlint`.

3.6. Przykład użycia Git

Poniżej przedstawiono przykładową procedurę pracy z projektem przy użyciu Git:

- **Tworzenie nowego brancha:** Aby rozpocząć pracę nad nową funkcjonalnością, użytkownik tworzy nowy branch:

  ```
  git checkout -b feature-new-feature
  ```

- **Dodawanie zmian:** Po wprowadzeniu zmian, użytkownik dodaje je do systemu kontroli wersji:

  ```
  git add .
  ```

- **Commit zmian:** Następnie użytkownik wykonuje commit:

  ```
  git commit -m "feat: dodanie nowej funkcjonalności"
  ```

- **Wysyłanie zmian na GitHub:** Po wykonaniu kilku commitów, użytkownik wysyła zmiany do repozytorium:

  ```
  git push origin feature-new-feature
  ```

- **Scalanie zmian:** Po zakończeniu pracy nad funkcjonalnością, branch jest scalany do głównego brancha (np. `main`):

  ```
  git checkout main
  git pull origin main
  git merge feature-new-feature
  git push origin main
  ```

3.7. Podsumowanie

W projekcie wykorzystano szereg narzędzi i technologii, które pozwalają na skuteczną i efektywną pracę nad projektem. Dzięki zastosowaniu systemów takich jak Git oraz GitHub Actions, proces integracji i testowania jest zautomatyzowany, co umożliwia szybkie wykrywanie błędów oraz integrację nowych funkcjonalności. Dokumentacja generowana przez Doxygen oraz LaTeX zapewnia jasny i profesjonalny sposób dokumentowania projektu.

4. Implementacja

W tym rozdziale przedstawimy implementację algorytmu wyszukiwania w drzewie binarnym (BST) w języku C++. Skupimy się na najważniejszych fragmentach kodu, które implementują główne operacje takie jak dodawanie, usuwanie elementów oraz wyświetlanie drzewa. Omówimy także powstałe wyniki działania aplikacji oraz wprowadzone optymalizacje.

4.1. Struktura kodu

Aplikacja składa się z kilku głównych modułów:

- **App.cpp** – implementacja interfejsu użytkownika, obsługi menu oraz wywołania odpowiednich funkcji na drzewie.
- **BinarySearchTree.cpp** – implementacja głównych operacji na drzewie wyszukiwań binarnych, takich jak dodawanie, usuwanie i przeglądanie drzewa.
- **main.cpp** – punkt wejścia do aplikacji, który uruchamia główną logikę programu.

Poniżej przedstawiamy przykłady implementacji najważniejszych funkcji.

4.2. Dodawanie elementów do drzewa

Funkcja `addItem` (listing 1 (s. 13)) umożliwia dodanie elementu do drzewa. Implementacja opiera się na rekursji – w zależności od wartości elementu, funkcja dodaje nowy węzeł do lewego lub prawego poddrzewa.

```
1  void BinarySearchTree::addItem(int value) {
2      root = _addItem(root, value);
3  }
4
5  BinarySearchTree::Node* BinarySearchTree::_addItem(Node* node, int
       value) {
6      if (node == nullptr) {
7          return new Node(value);
8      }
9      if (value < node->key) {
10         node->left = _addItem(node->left, value);
11     } else if (value > node->key) {
12         node->right = _addItem(node->right, value);
```

```
13      }
14      return node;
15  }
```

Listing 1. Funkcja addItem

Opis: Funkcja sprawdza, czy węzeł jest pusty (czyli czy osiągnęliśmy koniec drzewa lub poddrzewa), a następnie tworzy nowy węzeł o zadanej wartości. Następnie przekazuje go do odpowiedniego poddrzewa (lewego lub prawego), aby odpowiednio uporządkować drzewo.

4.3. Usuwanie elementów z drzewa

Usuwanie elementu z drzewa (listing 2 (s. 14)) jest bardziej skomplikowane, ponieważ musi zachować strukturalną spójność drzewa. Istnieją trzy przypadki:

1. Węzeł do usunięcia nie ma potomków – w takim przypadku po prostu usuwamy węzeł.
2. Węzeł do usunięcia ma jedno dziecko – wówczas zastępujemy węzeł jego dzieckiem.
3. Węzeł do usunięcia ma dwóch potomków – w takim przypadku należy znaleźć najmniejszy węzeł w prawym poddrzewie lub największy w lewym poddrzewie, zastąpić nim usuwany węzeł, a następnie usunąć ten węzeł rekurencyjnie.

```
1  void BinarySearchTree::removeItem(int value) {
2      root = _removeItem(root, value);
3  }
4
5  BinarySearchTree::Node* BinarySearchTree::_removeItem(Node* node,
       int value) {
6      if (node == nullptr) {
7          return nullptr;
8      }
9      if (value < node->key) {
10         node->left = _removeItem(node->left, value);
11     } else if (value > node->key) {
12         node->right = _removeItem(node->right, value);
13     } else {
14         if (node->left == nullptr && node->right == nullptr) {
15             delete node;
16             return nullptr;
```

```
17          }
18          if (node->left == nullptr) {
19              Node* temp = node->right;
20              delete node;
21              return temp;
22          }
23          if (node->right == nullptr) {
24              Node* temp = node->left;
25              delete node;
26              return temp;
27          }
28          Node* temp = _findMin(node->right);
29          node->key = temp->key;
30          node->right = _removeItem(node->right, temp->key);
31      }
32      return node;
33  }
```

Listing 2. Funkcja removeItem

Opis: Rekursywne usuwanie węzła odbywa się poprzez analizowanie, w którą stronę należy przejść – do lewego lub prawego poddrzewa. Gdy znajdziemy węzeł do usunięcia, odpowiednio go usuwamy lub zastępujemy.

4.4. Wyświetlanie drzewa

Aby umożliwić wizualizację struktury drzewa, zaimplementowaliśmy funkcję `display` (listing 3 (s. 15)), która umożliwia wyświetlenie drzewa w różnych porządkach: preorder, inorder oraz postorder.

```
1  void BinarySearchTree::display(BinaryTreeTraversalMethod
       traversalMethod) {
2      switch (traversalMethod) {
3          case PREORDER:
4              _traversePreorder(root);
5              break;
6          case INORDER:
7              _traverseInorder(root);
8              break;
9          case POSTORDER:
10             _traversePostorder(root);
11             break;
12     }
13 }
```

Listing 3. Funkcja display

Opis: Funkcja ta wykonuje odpowiednią metodę przeglądania drzewa, przekazując wybrany typ porządku przeglądania do odpowiedniej funkcji rekurencyjnej.

4.5. Wyniki implementacji

Po zaimplementowaniu głównych funkcji aplikacja została przetestowana na różnych zbiorach danych. Drzewo binarne skutecznie obsługiwało operacje dodawania, usuwania oraz wyświetlania w różnych porządkach. Aplikacja jest wydajna i działa w czasie proporcjonalnym do liczby elementów w drzewie.

```
MENU:

---ZARZĄDZANIE ELEMENTAMI DRZEWA---
1. Dodaj element
2. Usuń element
3. Znajdź ścieżkę do elementu

---ZARZĄDZANIE DRZEWEM---
4. Wyświetl drzewo
5. Wyczyść drzewo
6. Zapisz drzewo do pliku binarnego
7. Wczytaj drzewo z pliku binarnego
8. Wczytaj drzewo z pliku tekstowego (czyści istniejące drzewo)
9. Wczytaj drzewo z pliku tekstowego (dodaje do istniejącego drzewa)

---OPCJE---
10. Wyjdź

Wybierz działanie: 4

Wybierz metodę przejścia:
1. PREORDER
2. INORDER
3. POSTORDER

2
1 3 4 6 8 10
```

Rys. 4.1. Przykład wyników wyświetlania drzewa

Wynik 1: Wyświetlanie drzewa po kilku operacjach dodawania i usuwania elementów. Drzewo jest poprawnie zbalansowane i przedstawione w porządku inorder.

5. Wnioski

W ramach przeprowadzonego projektu opracowano aplikację do zarządzania drzewem binarnym wyszukiwania (BST), której głównym celem było umożliwienie użytkownikowi wykonywania podstawowych operacji na drzewie, takich jak dodawanie, usuwanie elementów oraz wyświetlanie struktury drzewa. Po zakończeniu implementacji i przeprowadzeniu testów można sformułować następujące wnioski:

5.1. Cel projektu i jego realizacja

Celem projektu było zaprojektowanie i implementacja algorytmu, który umożliwia efektywne zarządzanie drzewem binarnym wyszukiwania oraz udostępnienie intuicyjnego interfejsu użytkownika w postaci menu tekstowego. Projekt został zrealizowany zgodnie z założeniami, a aplikacja umożliwia wykonanie podstawowych operacji na drzewie BST w sposób wydajny i bezbłędny. Wszystkie funkcje aplikacji, takie jak dodawanie, usuwanie oraz przeglądanie drzewa, działają zgodnie z oczekiwaniami.

5.2. Możliwości rozwoju i optymalizacji

Choć aplikacja spełnia swoje zadanie, istnieje kilka obszarów, które mogą zostać zoptymalizowane w przyszłości:

- **Zbalansowanie drzewa:** Algorytm nie zapewnia automatycznego balansowania drzewa, co może prowadzić do pogorszenia wydajności operacji, szczególnie w przypadku wstawiania elementów w sposób uporządkowany. Zastosowanie algorytmów balansujących, takich jak drzewa AVL czy czerwono-czarne, mogłoby poprawić czas wykonywania operacji.

- **Interfejs użytkownika:** Aplikacja wykorzystuje jedynie interfejs tekstowy, co ogranicza jej funkcjonalność i wygodę użytkowania. Można rozważyć rozbudowę aplikacji o graficzny interfejs użytkownika (GUI), co poprawiłoby interaktywność programu.

- **Zarządzanie pamięcią:** W przyszłości warto rozważyć implementację inteligentnego zarządzania pamięcią, szczególnie w przypadku dużych drzew. Zastosowanie wskaźników smart pointers mogłoby pomóc w automatycznym zarządzaniu pamięcią, co zmniejszyłoby ryzyko wycieków pamięci.

5.3. Podsumowanie

Projekt umożliwił dogłębne zrozumienie i implementację algorytmu dla drzewa binarnego wyszukiwania, a także pozwolił na praktyczne zapoznanie się z zagadnieniami związanymi z operacjami na drzewach. Choć aplikacja działa zgodnie z wymaganiami, przyszła wersja projektu może zostać rozbudowana o dodatkowe funkcjonalności, optymalizacje i udoskonalenia, które umożliwią jej jeszcze lepszą skalowalność i wygodę użytkowania.

Przeprowadzona implementacja pokazuje, że algorytm wyszukiwania w drzewie binarnym jest jednym z fundamentalnych narzędzi w informatyce, które znajduje szerokie zastosowanie w różnych dziedzinach, takich jak bazy danych, kompilatory czy systemy plików. Projekt stanowi solidną podstawę do dalszych badań nad algorytmami strukturalnymi i ich zastosowaniem w praktyce.

Spis rysunków

Spis tabel

Spis listingów