AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

...Drzewo BST...

Autor:
Józef Czelusta
Seweryn Homoncik

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Celem pracy jest stworzenie programu implementującego drzewo BST (ang. Binary Search Tree, czyli drzewo wyszukiwania binarnego) w języku C++. Projekt zakłada, że struktura BST zostanie zaimplementowana jako klasa z różnorodnymi metodami pozwalającymi na manipulowanie drzewem, zapisywanie jego stanu do pliku oraz odczyt z pliku. Program ten umożliwi użytkownikowi operacje na strukturze danych w przejrzystym i intuicyjnym interfejsie tekstowym.

1.1. Cel i założenia projektu

Projekt ma na celu stworzenie wydajnego i funkcjonalnego narzędzia do pracy z drzewem BST. Struktura BST jest często wykorzystywana w różnych algorytmach i aplikacjach związanych z wyszukiwaniem, dzięki temu, że umożliwia szybkie wyszukiwanie, dodawanie oraz usuwanie elementów. Program powinien więc nie tylko spełniać funkcjonalność standardowych operacji BST, takich jak dodawanie, usuwanie i przeszukiwanie, ale również umożliwiać zapisywanie aktualnego stanu drzewa do pliku, zarówno tekstowego, jak i binarnego. Wymagana jest także funkcja pozwalająca na wczytanie drzewa z pliku, co daje możliwość szybkiego odtworzenia jego struktury z poprzedniego stanu.

Dodatkowo, użytkownik będzie mógł wybrać metodę wyświetlania drzewa na ekranie, korzystając z metod preorder, inorder i postorder, co pozwala na przejrzenie struktury w różnych układach.

1.2. Przewidywane wyniki i funkcjonalności

- Dodawanie elementów Implementacja powinna umożliwiać dodanie elementu do drzewa w odpowiednim miejscu. Oczekuje się, że dodawanie elementów będzie przebiegać zgodnie z zasadami BST, tzn. wartości mniejsze od korzenia będą dodawane do lewego poddrzewa, a większe do prawego.

- Usuwanie elementów Program powinien pozwolić na usunięcie wybranego elementu, a następnie odpowiednie zaktualizowanie struktury drzewa. Przewiduje się, że usuwanie będzie obsługiwało przypadki, w których element jest liściem, ma jedno dziecko, lub ma dwoje dzieci.

- Usuwanie całego drzewa Funkcja ta powinna zwalniać pamięć przeznaczoną na wszystkie węzły drzewa, co zapobiega wyciekom pamięci. Oczekuje się, że po wywołaniu tej funkcji drzewo będzie całkowicie wyczyszczone.

- Szukaj drogi do podanego elementu Program ma umożliwiać wyznaczenie drogi do zadanego elementu poprzez podanie kolejnych wartości węzłów od korzenia do poszukiwanego elementu. Funkcja ta powinna zwracać odpowiednią ścieżkę w postaci listy wartości.

- Wyświetlanie drzewa w różnych porządkach Przewiduje się, że użytkownik będzie mógł wybrać metodę wyświetlania drzewa spośród preorder, inorder oraz postorder, co pozwoli mu lepiej zrozumieć strukturę i ułożenie elementów w drzewie. Każda z tych metod ma swoje zastosowanie:

  Preorder – do przeglądania struktury drzewa od korzenia do liści.

  Inorder – do wyświetlania elementów drzewa w porządku rosnącym.

  Postorder – przydatne przy usuwaniu całej struktury.

- Zapis do pliku tekstowego i binarnego Drzewo powinno mieć możliwość zapisu do pliku tekstowego oraz binarnego. Zapis w formacie binarnym ma zapewnić szybszy zapis i odczyt dużych struktur drzewa, natomiast zapis tekstowy ułatwia podgląd stanu drzewa w edytorach tekstowych.

- Odczyt z pliku Program powinien umożliwiać wczytanie drzewa z pliku tekstowego zawierającego liczby, co umożliwi odbudowanie struktury drzewa od podstaw na podstawie danych wejściowych.

- Interfejs użytkownika w funkcji main Program będzie posiadał menu główne, które wyświetli dostępne opcje operacji na drzewie BST oraz na plikach. Użytkownik będzie mógł wybrać operacje związane z manipulacją drzewem oraz zapis i odczyt drzewa do/z pliku. Menu będzie intuicyjne i pozwoli użytkownikowi na łatwą nawigację między funkcjonalnościami. Dodatkowo przewidziano opcję zamknięcia programu.

1.3. Podsumowanie przewidywanych efektów

W wyniku realizacji projektu powstanie wszechstronny program implementujący drzewo BST w C++, który umożliwi użytkownikowi przeprowadzanie podstawowych operacji na drzewie oraz zapis i odczyt struktury z pliku. Rozbudowana funkcjonalność oraz możliwość zapisu i wczytania drzewa z plików tekstowych i binarnych uczyni program wszechstronnym i użytecznym narzędziem do pracy z danymi w strukturze BST.

2. Analiza problemu

2.1. Sposób działania programu implementującego drzewo BST

Program implementujący drzewo BST w języku C++ pozwala na wykonanie szeregu operacji na drzewie: dodawanie, usuwanie, wyszukiwanie, zapisywanie do pliku oraz wczytywanie z pliku. Działa on zgodnie z poniższymi zasadami:

Dodawanie elementów do drzewa:

- Program rozpoczyna od dodania elementu do korzenia drzewa, jeśli jest ono puste.
- Każdy nowy element jest porównywany z wartością bieżącego węzła:
- Jeśli jest mniejszy, algorytm przechodzi do lewego dziecka.
- Jeśli jest większy, przechodzi do prawego dziecka.
- Proces powtarza się rekurencyjnie aż do znalezienia odpowiedniego, pustego miejsca dla nowego węzła.
- Dodanie elementu ma złożoność czasową O(log n) dla drzewa zrównoważonego, ale może być O(n) dla drzewa niezrównoważonego (np. w przypadku, gdy elementy są dodawane w kolejności rosnącej lub malejącej). Usuwanie elementów z drzewa:
- Aby usunąć element, program wyszukuje go w drzewie.
- Jeśli węzeł nie ma dzieci, jest usuwany bezpośrednio.
- Jeśli węzeł ma jedno dziecko, to dziecko zastępuje usunięty węzeł.
- W przypadku węzła z dwoma dziećmi, algorytm wyszukuje najmniejszy element w jego prawym poddrzewie, który zastępuje wartość usuwanego węzła (metoda "znajdź minimum").
- Usuwanie elementu w BST może być złożone czasowo, zależnie od głębokości drzewa.

  Wyszukiwanie drogi do danego elementu

- Program przeszukuje drzewo, poruszając się od korzenia w dół, zgodnie z wartością poszukiwanego elementu.

- Podczas wyszukiwania zapisywane są wartości odwiedzanych węzłów, tworząc ścieżkę do szukanego elementu.
- Jeśli program nie znajdzie elementu, zwraca pustą ścieżkę.

  Wyświetlanie drzewa w różnych porządkach (preorder, inorder, postorder):

- Preorder: odwiedza węzeł, a następnie jego lewe i prawe poddrzewo.
- Inorder: odwiedza lewe poddrzewo, węzeł, a następnie prawe poddrzewo (daje uporządkowany zbiór elementów).
- Postorder: odwiedza najpierw lewe i prawe poddrzewo, a na końcu węzeł.
- Wyświetlanie w porządku inorder pozwala na przedstawienie drzewa jako uporządkowanej listy elementów.
- Zapisywanie i wczytywanie drzewa do/z pliku:
- Program umożliwia zapis drzewa zarówno do pliku tekstowego, jak i binarnego.
- W pliku tekstowym zapisywane są wartości węzłów w wybranym porządku (preorder, inorder lub postorder).
- W pliku binarnym struktura drzewa jest zachowywana, co pozwala na jego dokładne odtworzenie przy wczytywaniu.
- Wczytywanie drzewa z pliku tekstowego lub binarnego umożliwia odbudowę drzewa, przy czym program sprawdza istnienie pliku i tworzy drzewo na podstawie zapisanych wartości.

  Program jest wyposażony w interfejs konsolowy, który umożliwia użytkownikowi wybór operacji na drzewie. Użytkownik może wybrać, czy chce dodać element, usunąć element lub całe drzewo, wyszukać element, wyświetlić drzewo w wybranym porządku, zapisać je do pliku lub je z niego wczytać.

2.2. Zastosowanie algorytmu drzewa BST

Jednym z popularnych zastosowań algorytmu drzewa BST (Binary Search Tree) jest implementacja struktury danych dla wyszukiwania, wstawiania oraz usuwania elementów w sposób efektywny, zachowując uporządkowanie elementów. Przykładami takiego wykorzystania są:

- Bazy danych: W strukturach baz danych często stosuje się drzewa wyszukiwań binarnych lub ich modyfikacje (np. drzewa AVL, drzewa czerwono-czarne) do organizowania indeksów i sprawnego dostępu do rekordów. Indeksowanie baz danych z pomocą drzewa BST pozwala na szybkie wyszukiwanie danych, które mogą być uporządkowane według kluczy (np. identyfikatorów użytkowników, nazwisk lub numerów produktów).

- Systemy plików: W niektórych systemach plików lub systemach zarządzania pamięcią, stosuje się BST w celu organizacji plików lub segmentów pamięci, co umożliwia szybki dostęp i zarządzanie danymi.

- Kompilatory: Drzewa BST są stosowane w kompilatorach do zarządzania tablicami symboli (ang. symbol tables), gdzie przechowuje się informacje na temat zmiennych, funkcji i ich atrybutów. Wyszukiwanie w takim drzewie jest szybkie, co ułatwia analizę i przetwarzanie kodu.

- Filtrowanie danych i autouzupełnianie: Drzewa BST mogą być wykorzystane do implementacji funkcji autouzupełniania (ang. autocomplete) lub filtrowania danych na podstawie słów kluczowych. Wprowadzone frazy mogą być dynamicznie dodawane do drzewa, a wyszukiwanie umożliwia znalezienie pasujących słów w uporządkowanym zbiorze.

- Aplikacje wyszukujące: Drzewa BST sprawdzają się w systemach wyszukiwania, takich jak wyszukiwarki internetowe, gdzie umożliwiają szybkie sortowanie i filtrowanie wyników.

- Struktury indeksowe w aplikacjach graficznych i gier komputerowych: W grach komputerowych i aplikacjach graficznych BST są wykorzystywane do optymalizacji dostępu do obiektów, np. drzewo przechowuje obiekty według ich pozycji na ekranie lub ich priorytetów.

3. Projektowanie

3.1. Opis środowiska GitHub

GitHub[1] to platforma hostingowa, która umożliwia przechowywanie i zarządzanie kodem źródłowym projektów przy użyciu systemu kontroli wersji Git. Pozwala na współpracę nad projektami, śledzenie zmian w kodzie oraz synchronizowanie pracy. Dzięki GitHub, każdy członek zespołu może tworzyć oddzielne "gałęzie"(branch), na których wprowadza zmiany, a następnie łączy je z głównym projektem poprzez "pull requesty"po zaakceptowaniu przez innych.

GitHub jest kluczowym narzędziem w pracy zespołów open-source, gdzie kod projektów jest publicznie dostępny, a każdy użytkownik może zaproponować zmiany lub aktualizacje. Jest także platformą społecznościową, gdzie deweloperzy mogą tworzyć profile, śledzić innych programistów oraz przeglądać i uczyć się z publicznie dostępnych repozytoriów.

3.2. Język programowania - C++

C++ to jeden z najpopularniejszych i najbardziej wszechstronnych języków programowania, który jest używany w szerokim zakresie dziedzin, od systemów operacyjnych po aplikacje multimedialne, gry komputerowe, a także oprogramowanie naukowe. Język ten został zaprojektowany jako rozszerzenie języka C, wprowadzając paradygmaty programowania obiektowego, co czyni go bardziej elastycznym i umożliwia tworzenie bardziej złożonych programów i aplikacji.

- Podstawowe cechy języka C++: Wprowadza klasy, dziedziczenie, polimorfizm, abstrakcję oraz enkapsulację. Programowanie obiektowe pozwala na organizowanie kodu w struktury zwane obiektami, co ułatwia zarządzanie złożonymi aplikacjami. C++ jest językiem kompilowanym, co oznacza, że kod źródłowy jest przekształcany do postaci kodu maszynowego przed wykonaniem. Dzięki temu aplikacje napisane w C++ charakteryzują się dużą wydajnością, co czyni język idealnym do tworzenia aplikacji, gdzie szybkość działania jest krytyczna, jak np. w grach, symulacjach czy systemach wbudowanych. Język ten daje programistom pełną kontrolę nad pamięcią i zasobami systemowymi. Umożliwia ręczne zarządzanie pamięcią za pomocą operatorów new i delete, co pozwala na optymalizację zużycia zasobów, ale także wprowadza ryzyko błędów, takich jak wycieki pamięci.

[1] https://github.com/

C++ wspiera różne style programowania, w tym programowanie proceduralne, obiektowe, a także funkcyjne, co sprawia, że jest bardzo elastycznym językiem. Programiści mogą wybrać najlepszy sposób rozwiązywania problemu, dostosowując paradygmat do specyfiki aplikacji. Oferuje bogaty zestaw bibliotek standardowych (Standard Template Library, STL), które zawierają gotowe implementacje struktur danych (np. wektory, listy, mapy) oraz algorytmy (np. sortowanie, wyszukiwanie). STL umożliwia programistom szybkie tworzenie wydajnych aplikacji bez konieczności pisania kodu od podstaw. Programy napisane w C++ mogą być uruchamiane na różnych systemach operacyjnych, takich jak Windows, Linux czy macOS, dzięki dostosowanym kompilatorom i narzędziom dostarczanym przez różne platformy. Jest używany w tworzeniu aplikacji na urządzenia mobilne, komputery stacjonarne, a także w systemach wbudowanych.

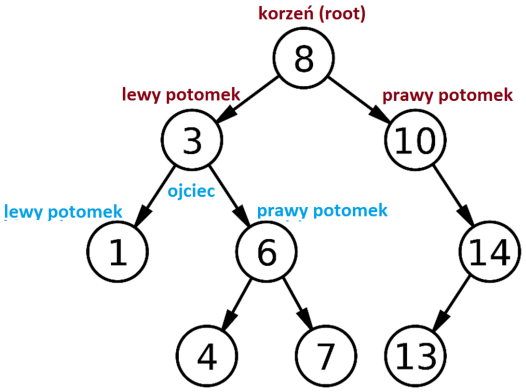
- Zalety C++:
  - Wydajność – Język ten pozwala na tworzenie bardzo wydajnych aplikacji, dzięki kompilacji do kodu maszynowego i bezpośredniemu dostępowi do pamięci.

  - Kontrola nad sprzętem – C++ umożliwia bezpośrednią manipulację pamięcią i rejestrami, co jest kluczowe w programowaniu systemów i aplikacji wbudowanych.

  - Wszechstronność – C++ jest stosowane w bardzo wielu dziedzinach: od gier komputerowych po oprogramowanie używane w przemyśle kosmicznym.

3.3. Wygląd drzewa BST

Drzewo składa się z węzłów (nodes). Każdy z nich posiada co najwyżej dwóch następników. Drzewo posiada tzw. „węzeł nadrzędny” (root). Jego następniki są nazywane węzłami potomnymi (dziecko, potomek).



Rys. 3.1. Przykładowe drzewo BST[1]

4. Implementacja

4.1. Plik main.cpp

W pliku main.cpp znajduje się główny program, który umożliwia interakcję z użytkownikiem i wykonanie operacji na drzewie BST. Program wyświetla menu z różnymi opcjami, takimi jak dodawanie elementów, usuwanie elementów, szukanie ścieżki do elementu, wyświetlanie drzewa w różnych porządkach (preorder, inorder, postorder), zapisywanie i wczytywanie drzewa z plików tekstowych oraz binarnych.

4.2. Plik BST.h

Plik BST.h zawiera definicję klasy DrzewoBST, która implementuje drzewo BST. Klasa ta zawiera:

- Struktura Wezel – reprezentuje węzeł drzewa, który przechowuje wartość oraz wskaźniki na lewego i prawego potomka.
- Prywatne i publiczne metody:
  - Dodawanie elementów: dodaj() i dodajRekurencyjnie() rekurencyjnie dodają elementy do drzewa.

  - Usuwanie elementów: usun() i usunRekurencyjnie() usuwają elementy z drzewa, z uwzględnieniem przypadków, gdy węzeł ma jedno lub dwoje dzieci.

  - Wyszukiwanie elementów: szukajSciezki() znajduje ścieżkę do danego elementu w drzewie.

  - Wyświetlanie drzewa: Metody wyswietlPreOrder(), wyswietlInOrder(), i wyswietlPostOrder() służą do wyświetlania drzewa w odpowiednich porządkach.

- Zapis i odczyt do pliku: Metody zapiszDoPliku() i wczytajZPliku() umożliwiają zapis i odczyt drzewa w formacie tekstowym, natomiast zapiszDoPlikuBinarnie() i wczytajZPlikuBinarnie() obsługują zapis i odczyt w formacie binarnym.

4.3. Plik BST.cpp

W pliku BST.cpp znajdują się definicje metod klasy DrzewoBST. Kluczowe operacje to:

- Dodawanie: Funkcja dodajRekurencyjnie() dodaje nowy węzeł do drzewa w odpowiednie miejsce na podstawie porównania wartości.

- Usuwanie: Funkcja usunRekurencyjnie() usuwa węzeł, a jeśli węzeł ma dwoje dzieci, zastępuje go najmniejszym węzłem z prawego poddrzewa (tzw. "successor").

- Wyszukiwanie ścieżki: szukajSciezkiRekurencyjnie() rekurencyjnie przeszukuje drzewo, aby znaleźć ścieżkę do zadanego elementu.

- Wyświetlanie drzewa: Metody wyswietlPreOrder(), wyswietlInOrder(), i wyswietlPostOrder() są rekurencyjnymi funkcjami, które odwiedzają węzły drzewa w odpowiednich porządkach.

- Zapis do pliku: W metodach zapiszDoPliku() i zapiszDoPlikuBinarnego() drzewo jest zapisywane w formacie tekstowym i binarnym, odpowiednio przy pomocy rekurencyjnych funkcji, które przechodzą przez węzły drzewa.

- Odczyt z pliku: Funkcja wczytajZPlikuBinarnie() odczytuje dane z pliku binarnego i odbudowuje drzewo.

  Dodawanie i usuwanie: Główne operacje na drzewie polegają na rekurencyjnym dodawaniu i usuwaniu węzłów, co jest typowe dla struktury BST. Usuwanie węzła, który ma dwoje dzieci, jest obsługiwane przez zastąpienie go najmniejszym elementem z prawego poddrzewa. Wyświetlanie drzewa: Drzewo może być wyświetlane w trzech różnych porządkach (preorder, inorder, postorder), co ułatwia jego analizowanie. Zapis i odczyt z pliku: Program umożliwia zapis i odczyt zarówno w formacie tekstowym, jak i binarnym, co jest przydatne do trwałego przechowywania drzewa i jego późniejszego wczytywania.

4.4. Pliki ZapisOdczyt.h i ZapisOdczyt.cpp

Te pliki zawierają funkcje odpowiedzialne za zapis i odczyt drzewa z plików binarnych.

- zapiszDrzewoBinarne() – zapisuje drzewo do pliku w formacie binarnym.
- wczytajDrzewoBinarne() – wczytuje dane z pliku binarnego i dodaje elementy do drzewa.

```
void DrzewoBST::dodaj(int wartosc) {
    korzen = dodajRekurencyjnie(korzen, wartosc);
}

DrzewoBST::Wezel* DrzewoBST::dodajRekurencyjnie(Wezel* wezel, int
    wartosc) {
    if (!wezel) return new Wezel(wartosc);
    if (wartosc < wezel->wartosc) {
        wezel->lewy = dodajRekurencyjnie(wezel->lewy, wartosc);
    }
    else if (wartosc > wezel->wartosc) {
        wezel->prawy = dodajRekurencyjnie(wezel->prawy, wartosc);
    }
    return wezel;
}
```

Listing 1. Metoda dodawania węzła

```
bool DrzewoBST::usun(int wartosc) {
    korzen = usunRekurencyjnie(korzen, wartosc);
    return korzen != nullptr;
}

DrzewoBST::Wezel* DrzewoBST::usunRekurencyjnie(Wezel* wezel, int
    wartosc) {
    if (!wezel) return nullptr;
    if (wartosc < wezel->wartosc) {
        wezel->lewy = usunRekurencyjnie(wezel->lewy, wartosc);
    }
    else if (wartosc > wezel->wartosc) {
        wezel->prawy = usunRekurencyjnie(wezel->prawy, wartosc);
    }
    else {
        if (!wezel->lewy) {
            Wezel* temp = wezel->prawy;
            delete wezel;
```

```
18              return temp;
19          }
20          else if (!wezel->prawy) {
21              Wezel* temp = wezel->lewy;
22              delete wezel;
23              return temp;
24          }
25          Wezel* minWezel = znajdzMin(wezel->prawy);
26          wezel->wartosc = minWezel->wartosc;
27          wezel->prawy = usunRekurencyjnie(wezel->prawy, minWezel->
    wartosc);
28      }
29      return wezel;
30  }
```

Listing 2. Metoda usuwania węzła

```
1  void DrzewoBST::wyswietlPreOrder(Wezel* wezel) {
2      if (wezel) {
3          std::cout << wezel->wartosc << " ";
4          wyswietlPreOrder(wezel->lewy);
5          wyswietlPreOrder(wezel->prawy);
6      }
7  }
```

Listing 3. Metoda wyświetlania drzewa w pre order

```
1  void DrzewoBST::wyswietlInOrder(Wezel* wezel) {
2      if (wezel) {
3          wyswietlInOrder(wezel->lewy);
4          std::cout << wezel->wartosc << " ";
5          wyswietlInOrder(wezel->prawy);
6      }
7  }
```

Listing 4. Metoda wyświetlania drzewa w in order

```
1  void DrzewoBST::wyswietlPostOrder(Wezel* wezel) {
2      if (wezel) {
3          wyswietlPostOrder(wezel->lewy);
4          wyswietlPostOrder(wezel->prawy);
5          std::cout << wezel->wartosc << " ";
6      }
7  }
```

Listing 5. Metoda wyświetlania drzewa w post order

5. Wnioski

5.1. Operacje na drzewie BST

Rekurencja jest kluczowym narzędziem w implementacji operacji na drzewie BST, zwłaszcza w przypadkach takich jak dodawanie, usuwanie czy wyszukiwanie ścieżek do elementów. Stosowanie rekurencji umożliwia eleganckie rozwiązanie problemów związanych z nawigacją w strukturze drzewa, gdzie każde poddrzewo można traktować jako nową instancję tego samego problemu.

Operacje na drzewie BST, takie jak dodawanie, usuwanie, czy wyszukiwanie, mają złożoność czasową O(log n) w przypadku zrównoważonych drzew. Jednak w przypadku drzew zdegenerowanych (np. zbudowanych w postaci listy), złożoność tych operacji może wynieść O(n). Zrozumienie tej zależności pomaga w optymalizacji i dostosowywaniu algorytmów do realnych scenariuszy, np. w kontekście zapewnienia zrównoważenia drzewa.

Implementacja operacji zapisywania i wczytywania drzewa do/z pliku (zarówno tekstowego, jak i binarnego) pokazała, jak można zrealizować trwałe przechowywanie danych. Przydatność tej funkcji staje się jasna w kontekście przechowywania dużych struktur danych, których zawartość może być zapisana na dysku i później odczytana, co jest istotne w wielu aplikacjach, np. bazach danych.

5.2. Podsumowanie

Projekt pozwala na głębsze zrozumienie podstawowej struktury danych, jaką jest drzewo binarne poszukiwań (BST). Zastosowanie tej struktury w różnych operacjach, takich jak dodawanie, usuwanie, wyszukiwanie czy wyświetlanie elementów w różnych porządkach (preorder, inorder, postorder) pozwala na lepsze zrozumienie, jak działa organizacja danych w formie drzewa i jak można przechowywać oraz manipulować danymi w sposób hierarchiczny.

Bibliografia

[1] *Źródło rysunku drzewa BST*. URL: https://zielonybuszmen.github.io/2017/02/20/binarne-drzewo-poszukiwan-binary-search-tree-bst/ (term. wiz. 14. 11. 2024).

Spis rysunków

Spis listingów