AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Drzewo BST
działające na stercie w języku C++

Autor:
Szymon Biel
Paweł Dudziak
Grzegorz Fiejtek

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Napisz program „drzewo BST" działającej na stercie w języku C++. Drzewo winno być zaimplementowana w klasie. Funkcjonalność (metod) drzewa:

- Dodaj element,
- Usuń element,
- Usuń całe drzewo,
- Szukaj drogi do podanego elementu,
- Wyświetl drzewo graficznie na ekranie, użytkownik wybiera metodę podczas wyświetlania (metody preorder, inorder, postorder) [oprogramuj wszystkie trzy],
- Zapis do pliku tekstowego wygenerowanego drzewa,

W drugiej klasie należy zaimplementować (metody) zapis do pliku i odczyt z pliku utworzonego drzewa BTS (plik musi być zapisany binarnie). Oprócz tego powinien mieć możliwość wczytania pliku tekstowego z cyframi, co 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 oraz odczytu i zapisu pliku. Program czeka na wybranie opcji. Nie zapomnij ustawić wyjścia z programu aby program można było poprawnie zamknąć.

Wszystkie utworzone klasy mają być zaimplementowane w oddzielnych plikach. Funkcja main także powinna być w osobnym pliku.

Celem zadania jest poznanie możliwości GitHuba w pracy nad projektem grupowym. Konto na GitHub jest utworzone. Należy utworzyć nowy projekt, który jest realizowany w dwuosobowych grupach.

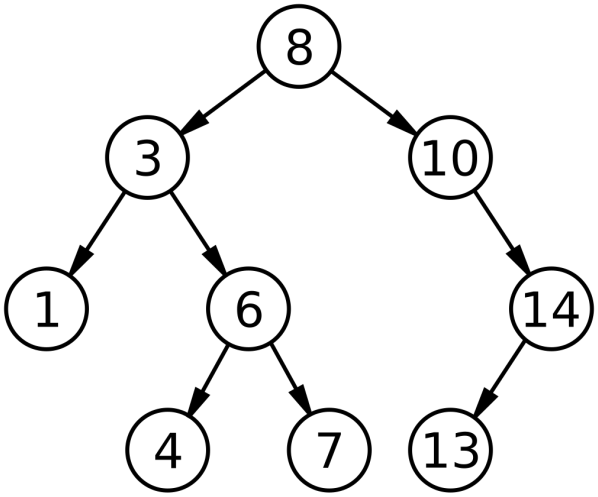
Przy oddawaniu projektu należy zaprezentować:

- Co najmniej 5 commit'ów (każdej osoby),
- Najpierw jedna osoba tworzy gałąź i po kilku comitach ją scala. Po scaleniu druga osoba tworzy gałąź i po kilku comitach ją scala do głównej gałęzi ,
- Obie osoby w grupie mają utworzyć nowe gałęzie (w jednym punkcie obie osoby zaczynają pracę równoległą), a po kilku comitach wykonują scalenie do głównej gałęzi,
- Co najmniej 6 konfliktów, które należy rozwiązać (3 jedna osoba, 3 druga osoba) przy scalaniu gałęzi (w wcześniejszych punktach),

2. Analiza problemu

2.1. Czym jest drzewo BST?

Binarne drzewo poszukiwań (ang. Binary Search Tree, BST) 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.



Rys. 2.1. Drzewo BST
[1]

Na rys. 2.1 przedstawiona jest przykładkowa wizualizacja drzewa BST.

2.2. Zastosowanie

Binarne drzewa poszukiwań są szeroko stosowane w informatyce i systemach przetwarzania danych, szczególnie tam, gdzie istotna jest efektywność wyszukiwania, wstawiania oraz usuwania danych. Przykłady zastosowań BST obejmują:

- Systemy baz danych – BST jest często używane jako podstawa do indeksowania danych w celu przyspieszenia operacji wyszukiwania. W przypadku dużych zbiorów danych, zrównoważone drzewa są szczególnie efektywne.
- Kompilatory i interpretery - Drzewa poszukiwań umożliwiają przechowywanie symboli i identyfikatorów, ułatwiając szybki dostęp do informacji o zmiennych, funkcjach i innych elementach składniowych.
- Struktury słowników – BST mogą służyć jako struktura do implementacji słowników, gdzie każdy klucz jest unikalny i powiązany z wartością. Dzięki temu możliwe jest szybkie sprawdzanie obecności klucza oraz przypisanej do niego wartości.
- Systemy rekomendacyjne – W niektórych przypadkach drzewa BST mogą być używane do filtrowania wyników lub porządkowania ich według wartości klucza, co ułatwia przeszukiwanie danych w oparciu o kryteria rangowe.

BST są stosowane wszędzie tam, gdzie kluczowe znaczenie ma szybki dostęp do posortowanych danych, a także efektywne zarządzanie ich aktualizacją.

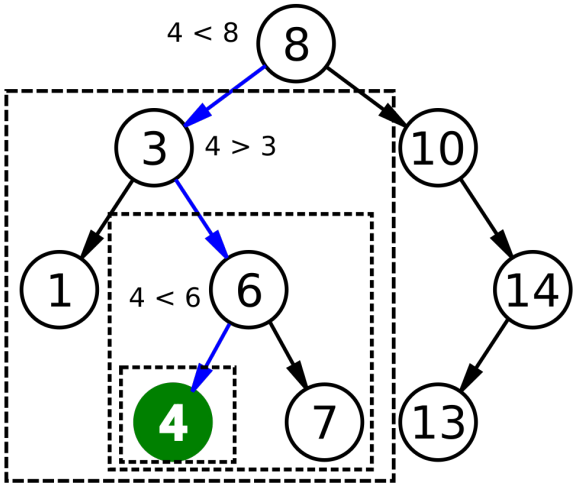
2.3. Sposób działania

BST przechowuje dane w postaci drzewiastej, gdzie każdy węzeł zawiera klucz, a także wskaźniki na lewe i prawe poddrzewo. Działanie BST opiera się na zasadzie porządku węzłów, gdzie:

- Lewe poddrzewo zawiera węzły o kluczach mniejszych od klucza węzła głównego.
- Prawe poddrzewo zawiera węzły o kluczach większych lub równych kluczowi węzła głównego.

Dzięki takiej strukturze można efektywnie wykonywać operacje wyszukiwania, wstawiania i usuwania danych:

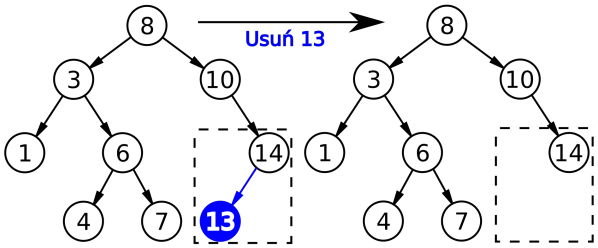
- Wyszukiwanie – Proces rozpoczyna się od korzenia i porównuje klucz szukanego elementu z kluczem bieżącego węzła. W zależności od wyniku porównania przeszukiwane jest lewe lub prawe poddrzewo, aż do znalezienia klucza lub stwierdzenia, że nie istnieje w drzewie.
  Rys 2.2 przedstawia graficznie schemat wyszukiwania klucza o wartości 4.



Rys. 2.2. Wyszukiwanie w drzewie BST
[2]

- Wstawianie – Nowy element dodawany jest do BST zgodnie z zasadą porządku węzłów. Proces wstawiania zaczyna się od korzenia i w zależności od wartości klucza wędruje w dół drzewa, aż znajdzie odpowiednie miejsce do osadzenia nowego węzła jako liścia.

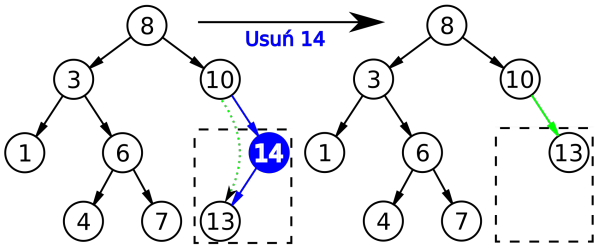
- Usuwanie – Usunięcie węzła w BST wymaga uwzględnienia jego położenia w drzewie. Możliwe są trzy przypadki:



Rys. 2.3. Usunięcie węzła który jest liściem

[3]

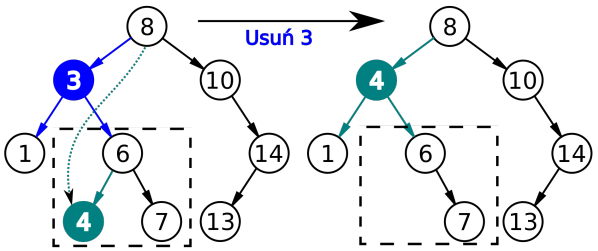
Węzeł jest liściem – zostaje usunięty bezpośrednio.
Na rys 2.3 schemat usunięcia węzła 13 który jest liściem.



Rys. 2.4. Usunięcie węzła z jednym synem

[4]

Węzeł ma jedno dziecko – dziecko węzła zajmuje jego miejsce.
Na rys 2.4 schemat usunięcia węzła 14 który ma jedno dziecko.



Rys. 2.5. Usunięcie węzła dwoma synami

[5]

Węzeł ma dwoje dzieci – węzeł zostaje zastąpiony przez swojego następnika (najmniejszy węzeł w prawym poddrzewie) lub poprzednika (największy węzeł w lewym poddrzewie).
Na rys 2.5 schemat usunięcia węzła 3 który ma dwujkę dzieci.

3. Projektowanie

3.1. Git

3.1.1. Co to git?

Git to rozproszony system kontroli wersji, który pozwala na zarządzanie zmianami w kodzie źródłowym lub innych plikach w projekcie. Umożliwia śledzenie historii zmian, współpracę wielu osób nad jednym projektem oraz tworzenie równoległych wersji projektu.

3.1.2. Zastosowanie

Git jest szeroko stosowany w programowaniu, projektach zespołowych oraz zarządzaniu dokumentacją, ponieważ umożliwia:

- Śledzenie zmian: Każda zmiana w plikach może być zapisana jako tzw. commit, co pozwala na odtworzenie dowolnej wcześniejszej wersji projektu.
- Współpracę: Git pozwala wielu osobom jednocześnie pracować nad tym samym projektem, synchronizując ich zmiany.
- Rozgałęzianie i scalanie: Git pozwala tworzyć różne wersje projektu i później scalić je w jedną całość.

3.1.3. Sposób działania

Git przechowuje zmiany w projekcie w repozytorium, które może znajdować się lokalnie lub zdalnie. Główne operacje wykonywane w Git to:

- Clone – skopiowanie całego repozytorium na komputer użytkownika.
- Commit – zapisanie zmian lokalnie, jako kolejnej wersji projektu.
- Push – przesłanie lokalnych zmian do zdalnego repozytorium.
- Pull – pobranie zmian ze zdalnego repozytorium do lokalnej kopii projektu.
- Branch – stworzenie nowej gałęzi (np. do testowania nowej funkcji).
- Merge – scalanie zmian z jednej gałęzi do innej.

3.1.4. Zastosowanie w projekcie

3.2. C++

3.2.1. Co to C++?

C++ to język programowania ogólnego przeznaczenia, który rozszerza język C o mechanizmy programowania obiektowego, a także inne zaawansowane funkcje.

3.2.2. Zastosowanie

C++ jest używany w wielu dziedzinach, w tym:

- Tworzenie systemów operacyjnych: np. fragmenty systemów takich jak Windows czy Linux.
- Aplikacje desktopowe i mobilne: edytory tekstu, programy graficzne, oprogramowanie inżynieryjne.
- Programowanie gier: ze względu na wydajność i możliwość pracy z grafiką oraz pamięcią.
- Systemy wbudowane: np. oprogramowanie dla mikrokontrolerów. Przetwarzanie wysokowydajne

3.2.3. Sposób działania

C++ umożliwia programowanie zarówno proceduralne, jak i obiektowe. Jego cechy obejmują:

- Programowanie obiektowe: C++ wprowadza klasy i obiekty, co pozwala na grupowanie danych oraz funkcji w struktury, co ułatwia zarządzanie złożonością kodu.
- Dziedziczenie: Możliwość tworzenia nowych klas na podstawie istniejących, co wspiera ponowne używanie kodu.
- Polimorfizm: Pozwala funkcjom i obiektom przyjmować różne formy w zależności od kontekstu.
- Zarządzanie pamięcią: W C++ mamy pełną kontrolę nad pamięcią, co pozwala na dynamiczne alokowanie i zwalnianie zasobów.

3.3. Visual Studio

3.3.1. Co to Visual Studio?

Visual Studio to zintegrowane środowisko programistyczne stworzone przez firmę Microsoft. Jest to narzędzie używane do tworzenia, testowania i debugowania aplikacji w różnych językach programowania, takich jak C++, Python, JavaScript, i wielu innych.

3.3.2. Zastosowanie

Visual Studio jest używane przez programistów do tworzenia różnych typów oprogramowania:

- Aplikacje desktopowe (Windows Forms, WPF),
- Aplikacje webowe (ASP.NET, HTML, CSS, JavaScript),
- Aplikacje mobilne (Xamarin, .NET MAUI),
- Gry (Unity, Unreal Engine),
- Usługi chmurowe (Microsoft Azure).

3.3.3. Sposób działania

Visual Studio ułatwia tworzenie oprogramowania, oferując szereg funkcji, które wspierają proces tworzenia kodu:

- Edytor kodu: Oferuje wsparcie dla składni wielu języków, podświetlanie kodu, autouzupełnianie, a także sugestie podczas pisania.
- Kompilator i debugger: Visual Studio pozwala kompilować kod do plików wykonywalnych i automatycznie wykrywać oraz naprawiać błędy w kodzie.
- IntelliSense: Zaawansowana funkcja podpowiedzi podczas pisania kodu, która analizuje kod i proponuje odpowiednie funkcje, metody czy zmienne.
- Integracja z systemami kontroli wersji: Visual Studio obsługuje Git, co pozwala zarządzać kodem, śledzić zmiany i współpracować z zespołem bezpośrednio z poziomu środowiska IDE.
- Debugowanie i profilowanie: Narzędzia do śledzenia błędów oraz wydajności kodu, które pomagają analizować i optymalizować aplikacje.

- Szablony projektów: Visual Studio oferuje szereg gotowych szablonów projektów, które ułatwiają rozpoczęcie pracy nad aplikacjami webowymi, desktopowymi czy mobilnymi.

3.4. Doxygen

3.4.1. Co to Doxygen?

Doxygen to narzędzie do automatycznego generowania dokumentacji technicznej z kodu źródłowego. Obsługuje wiele języków programowania, takich jak C, C++, Java, Python, Fortran, PHP i inne. Doxygen skanuje kod źródłowy i na podstawie specjalnych komentarzy generuje szczegółową dokumentację w formacie HTML, PDF, LaTeX, czy RTF.

3.4.2. Zastosowanie

Doxygen jest szeroko stosowany w projektach programistycznych, aby generować dokumentację bezpośrednio na podstawie kodu. Jest szczególnie popularny w zespołach pracujących z językami C i C++, ale także w innych środowiskach, które wymagają dokumentacji technicznej. Przykładowe zastosowania:

- Dokumentacja bibliotek i API: Umożliwia automatyczne tworzenie dokumentacji dla publicznych interfejsów, funkcji, klas, metod i zmiennych.
- Projekty open-source: Doxygen jest popularny w projektach open-source, ponieważ pozwala na łatwe udostępnienie dokumentacji użytkownikom i programistom pracującym nad kodem.
- Zarządzanie dużymi projektami: Dzięki automatycznemu generowaniu dokumentacji, Doxygen pomaga programistom zrozumieć struktury dużych i złożonych projektów.

3.4.3. Sposób działania

Doxygen działa poprzez analizowanie specjalnych komentarzy w kodzie, które są oznaczane za pomocą znaczników podobnych do tych w języku HTML. Po uruchomieniu Doxygen generuje dokumentację na podstawie tych komentarzy oraz samego kodu, opisując struktury takie jak klasy, funkcje, zmienne i ich relacje.

4. Implementacja

4.1. Plik bst.h

4.1.1. struct wezel

```
1    struct wezel {
2     wezel *pop;
3     wezel *lewy;
4     wezel *prawy;
5     int war;
6 };
```

Listing 1. struct wezel

W lini 2: Wskaźnik na rodzica danego węzła.
W lini 3: Wskaźnik na lewego potomka węzła.
W lini 4: Wskaźnik na prawego potomka węzła.
W lini 5: Wartość przechowywana w węźle.

4.1.2. class bst

```
1    class bst {
2 public:
3     wezel *korzen;
4     bst();
5     ~bst();
6     wezel *najmin(wezel *a);
7     wezel *najmax(wezel *a);
8     wezel *min(wezel *a);
9     wezel *max(wezel *a);
10    wezel *zwroc(int a);
11    void dodele(int a);
12    wezel *usunele(int d);
13    void usubst(wezel *korzen);
14    void szuk(int a);
15    void wyspre(wezel *korzen);
16    void wysin(wezel *korzen);
17    void wyspost(wezel *korzen);
18    void wczytaj();
19    void zapisz();
20 };
```

Listing 2. class bst

W lini 3: Wskaźnik na korzeń drzewa.
W public metody klasy opisane osobno w dokumentacji.

4.2. Plik bst.cpp

4.2.1. konstruktor

```
1 bst::bst(void) {
2     korzen = NULL;
3 }
```

Listing 3. konstruktor

W lini 2: Inicjalizuje drzewo ustawiając wskaźnik ‘korzen‘ na ‘NULL‘

4.2.2. destruktor

```
1 bst::~bst(void) {
2     delete korzen;
3 }
```

Listing 4. destruktor

W lini 2: Usuwa pamięć zajętą przez korzeń drzewa, co wywołuje destrukcję całego drzewa.

4.2.3. najmin

```
1 wezel *bst::najmin(wezel *a) {
2     wezel *b = a;
3     while(b->lewy) b = b->lewy;
4     return b;
5 }
```

Listing 5. najmin

W lini 2: Tworzy wskaźnik ‘b‘ i przypisuje do niego węzeł ‘a‘.
W lini 3: Przechodzi do lewego potomka, dopóki ‘b‘ ma lewego potomka.
W lini 4: Zwraca węzeł z najmniejszą wartością w poddrzewie ‘a‘.

4.2.4. najmax

```
1 wezel *bst::najmax(wezel *a) {
2     wezel *b = a;
3     while(b->prawy) b = b->prawy;
4     return b;
5 }
```

Listing 6. najmax

W lini 2: Tworzy wskaźnik ‘b‘ i przypisuje do niego węzeł ‘a‘.
W lini 3: Przechodzi do prawego potomka, dopóki ‘b‘ ma prawego potomka.
W lini 4: Zwraca węzeł z największą wartością w poddrzewie ‘a‘.

4.2.5. min

```
1 wezel *bst::min(wezel *a) {
2     if(a->lewy) return najmax(a->lewy);
3     wezel *b;
4     do {
5         b = a;
6         a = a->pop;
7     } while((a) && (a->prawy != b));
8     return a;
9 }
```

Listing 7. min

W lini 2: Jeśli ‘a‘ ma lewego potomka, zwraca największy węzeł w lewym poddrzewie.
W lini 5: Przypisuje ‘a‘ do ‘b‘.
W lini 6: Przechodzi do rodzica ‘a‘.
W lini 7: Dopóki ‘a‘ istnieje i ‘b‘ jest prawym potomkiem ‘a‘, kontynuuje.
W lini 8: Zwraca najmniejszy węzeł większy od ‘a‘

4.2.6. max

```
1 wezel *bst::max(wezel *a) {
2     if(a->prawy) return najmin(a->prawy);
3     wezel *b;
4     do {
5         b = a;
```

```
6           a = a->pop;
7       } while((a) && (a->lewy != b));
8       return a;
9   }
```

Listing 8. max

W lini 2: Jeśli ‘a‘ ma prawego potomka, zwraca najmniejszy węzeł w prawym poddrzewie.
W lini 5: Przypisuje ‘a‘ do ‘b‘.
W lini 6: Przechodzi do rodzica ‘a‘.
W lini 7: Dopóki ‘a‘ istnieje i ‘b‘ jest lewym potomkiem ‘a‘, kontynuuje.
W lini 8: Zwraca największy węzeł mniejszy od ‘a‘.

4.2.7. zwroc

```
1  wezel *bst::zwroc(int a) {
2      wezel *b = korzen;
3      while((b) && (b->war != a)) {
4          if(a < b->war) b = b->lewy;
5          else b = b->prawy;
6      }
7      return b;
8  }
```

Listing 9. zwroc

W lini 2: Ustawia wskaźnik ‘b‘ na korzeń drzewa.
W lini 3: Dopóki ‘b‘ istnieje i ‘b-¿war‘ nie jest równe ‘a‘, wykonuje pętlę.
W lini 4: Jeśli ‘a‘ jest mniejsze niż ‘b-¿war‘, przechodzi do lewego potomka.
W lini 5: W przeciwnym razie przechodzi do prawego potomka.
W lini 7: Zwraca wskaźnik ‘b‘ na znaleziony węzeł lub ‘NULL‘ jeśli nie znaleziono.

4.2.8. dodele

```
1  void bst::dodele(int a) {
2      wezel *b = new wezel;
3      wezel *c = korzen;
4      b->lewy = NULL;
5      b->prawy = NULL;
6      b->war = a;
```

```
7       if(!c) korzen = b;
8       else {
9           while(true) {
10              if(b->war < c->war) {
11                  if(!c->lewy) {
12                      c->lewy = b;
13                      break;
14                  }
15                  else c = c->lewy;
16              }
17              else {
18                  if(!c->prawy) {
19                      c->prawy = b;
20                      break;
21                  }
22                  else c = c->prawy;
23              }
24          }
25      }
26      b->pop = c;
27  }
```

Listing 10. dodele

W lini 2: Tworzy nowy węzeł ‘b‘.

W lini 3: Ustawia wskaźnik ‘c‘ na korzeń drzewa.

W lini 4: Ustawia lewego potomka ‘b‘ na ‘NULL‘.

W lini 5: Ustawia prawego potomka ‘b‘ na ‘NULL‘.

W lini 6: Ustawia wartość ‘war‘ węzła ‘b‘ na ‘a‘.

W lini 7: Jeśli drzewo jest puste, ustawia ‘b‘ jako korzeń.

W lini 10: Jeśli ‘b-¿war‘ jest mniejsze niż ‘c-¿war‘, przechodzi w lewo.

W lini 11: Jeśli lewy potomek ‘c‘ jest ‘NULL‘, ustawia ‘b‘ jako lewy potomek.

W lini 15: W przeciwnym razie kontynuuje w lewo.

W lini 17: Jeśli ‘b-¿war‘ jest większe lub równe ‘c-¿war‘, przechodzi w prawo.

W lini 18: Jeśli prawy potomek ‘c‘ jest ‘NULL‘, ustawia ‘b‘ jako prawy potomek.

W lini 22: W przeciwnym razie kontynuuje w prawo.

W lini 26: Ustawia ‘pop‘ nowego węzła ‘b‘ na ‘c‘.

4.2.9. usunele

```
1   wezel *bst::usunele(int d) {
2       wezel *a = zwroc(d);
```

```
3     wezel *b = a->pop;
4     wezel *c = NULL;
5     if((a->lewy) && (a->prawy)) {
6         if(rand() % 2)
7         c = usunele(min(a)->war);
8         else c = usunele(max(a)->war);
9         c->lewy = a->lewy;
10        if(c->lewy) c->lewy->pop = c;
11        c->prawy = a->prawy;
12        if(c->prawy) c->prawy->pop = c;
13    }
14    else {
15        if(a->lewy) c = a->lewy;
16        else c = a->prawy;
17    }
18    if(c) c->pop = b;
19    if(!b) korzen = c;
20    else {
21        if(b->lewy == a) b->lewy = c;
22        else b->prawy = c;
23    }
24    return a;
25 }
```

Listing 11. usunele

W lini 2: Znajduje węzeł a o wartości d.
W lini 3: Ustawia wskaźnik b na rodzica węzła a.
W lini 4: Inicjuje wskaźnik c jako NULL.
W lini 5: Sprawdza, czy a ma dwóch potomków.
W lini 6: Losowo wybiera usunięcie minimalnego lub maksymalnego węzła.
W lini 7: Usuwa węzeł o minimalnej wartości w poddrzewie a.
W lini 8: Usuwa węzeł o maksymalnej wartości w poddrzewie a.
W lini 9: Ustawia lewego potomka c na lewego potomka a.
W lini 10: Ustawia rodzica lewego potomka na c, jeśli istnieje.
W lini 11: Ustawia prawego potomka c na prawego potomka a
W lini 12: Ustawia rodzica prawego potomka na c, jeśli istnieje.
W lini 15: Jeśli a ma tylko lewego potomka, ustawia c na lewego potomka.
W lini 16: Jeśli a ma tylko prawego potomka, ustawia c na prawego potomka.
W lini 18: Ustawia rodzica c na b, jeśli c istnieje.
W lini 19: Jeśli b jest NULL, oznacza to, że usuwany węzeł jest korzeniem.
W lini 19: Ustawia c jako nowy korzeń.

W lini 21: Jeśli a jest lewym potomkiem b, ustawia c jako nowego lewego potomka.
W lini 22: Jeśli a jest prawym potomkiem b, ustawia c jako nowego prawego potomka.
W lini 24: Zwraca usunięty węzeł a.

4.2.10. usubst

```
1 void bst::usubst(wezel *a) {
2     if(a) {
3         usubst(a->lewy);
4         usubst(a->prawy);
5         usunele(a->war);
6     }
7 }
```

Listing 12. usubst

W lini 2: Sprawdza, czy a nie jest NULL.
W lini 3: Rekurencyjnie usuwa lewego potomka
W lini 4: Rekurencyjnie usuwa prawego potomka.
W lini 5: Usuwa aktualny węzeł a.

4.2.11. szuk

```
1 void bst::szuk(int a) {
2     wezel *b = korzen;
3     while((b) && (b->war != a)) {
4         std::cout << b->war << " ";
5         if(a < b->war) b = b->lewy;
6         else b = b->prawy;
7     }
8     std::cout << std::endl;
9 }
```

Listing 13. szuk

W lini 2: Ustawia wskaźnik b na korzeń drzewa.
W lini 3: Przechodzi przez drzewo, dopóki b istnieje i b-¿war nie jest równe a.
W lini 4: Wyświetla wartość bieżącego węzła.
W lini 5: Jeśli a jest mniejsze niż b-¿war, przechodzi w lewo.
W lini 6: Jeśli a jest większe lub równe b-¿war, przechodzi w prawo.

W lini 8: Kończy linię po zakończeniu pętli.

4.2.12. wyspre

```
1 void bst::wyspre(wezel *a) {
2     if (a) {
3         std::cout << a->war << " ";
4         wyspre(a->lewy);
5         wyspre(a->prawy);
6     }
7 }
```

Listing 14. wyspre

W lini 2: Sprawdza, czy a nie jest NULL.
W lini 3: Wyświetla wartość a w porządku przedrostnym.
W lini 4: Rekurencyjnie wyświetla lewego potomka.
W lini 5: Rekurencyjnie wyświetla prawego potomka.

4.2.13. wysin

```
1 void bst::wysin(wezel *a) {
2     if(a) {
3         wysin(a->lewy);
4         std::cout << a->war << " ";
5         wysin(a->prawy);
6     }
7 }
```

Listing 15. wysin

W lini 2: Sprawdza, czy a nie jest NULL.
W lini 3: Rekurencyjnie wyświetla lewego potomka.
W lini 4: Wyświetla wartość a.
W lini 5: Rekurencyjnie wyświetla prawego potomka.

4.2.14. wyspost

```
1 void bst::wyspost(wezel *a) {
2     if(a) {
```

```
3           wyspost(a->lewy);
4           wyspost(a->prawy);
5           std::cout << a->war << " ";
6       }
7   }
```

Listing 16. wyspost

W lini 2: Sprawdza, czy a nie jest NULL.
W lini 3: Rekurencyjnie wyświetla lewego potomka.
W lini 4: Rekurencyjnie wyświetla prawego potomka.
W lini 5: Wyświetla wartość a.

4.2.15. zapisz

```
1 void bst::zapisz() {
2     dopliku a(this->korzen);
3     a.zapisz();
4 }
```

Listing 17. zapisz

W lini 2: Tworzy obiekt dopliku inicjowany korzeniem drzewa.
W lini 3: Wywołuje metodę zapisu do pliku.

4.2.16. wczytaj

```
1 void bst::wczytaj() {
2     dopliku a(this->korzen);
3     a.wczytaj();
4     this->korzen = a.drzewo;
5 }
```

Listing 18. wczytaj

W lini 2: Tworzy obiekt dopliku inicjowany korzeniem drzewa.
W lini 3: Wywołuje metodę wczytywania z pliku.
W lini 4: Ustawia korzeń drzewa na załadowane drzewo.

4.3. Plik wczyt.h

4.3.1. class dopliku

```
class dopliku {
public:
  wezel *drzewo;
  std::fstream PRE;
  std::fstream IN;
  std::fstream POST;
  dopliku(wezel *a);
  void wczytajpre(wezel* a);
  void wczytajin(wezel* a);
  void wczytajpost(wezel* a);
  void zapisz();
  void zapiszpre(wezel* a);
  void zapiszin(wezel* a);
  void zapiszpost(wezel* a);
  void wczytaj();
  wezel *rdrzewo();
  int bintoint(int a);
  int inttobin(int a);
};
```

Listing 19. class dopliku

Tworzona jest klasa dopliku.
Jej metody opisane są osobno w dokumentacji.

4.4. Plik wczyt.cpp

4.4.1. dopliku

```
dopliku::dopliku(wezel *a) : drzewo(a) {}
```

Listing 20. dopliku

Konstruktor klasy dopliku, inicjalizuje wskaźnik drzewo na węzeł a.

4.4.2. zapisz

```
void dopliku::zapisz(){
    PRE.open("PRE.txt", std::ios::out | std::ios::trunc);
    if (!PRE.is_open()) {
```

```
4           std::cout << "Nie ma dostepu do pliku PRE.txt" << std::endl
    ;
5           return;
6       }
7       zapiszpre(drzewo);
8       PRE.close();
9
10      IN.open("IN.txt", std::ios::out | std::ios::trunc);
11      if (!IN.is_open()) {
12          std::cout << "Nie ma dostepu do pliku IN.txt" << std::endl;
13          return;
14      }
15      zapiszin(drzewo);
16      IN.close();
17
18      POST.open("POST.txt", std::ios::out | std::ios::trunc);
19      if (!POST.is_open()) {
20          std::cout << "Nie ma dostepu do pliku POST.txt" << std::
    endl;
21          return;
22      }
23      zapiszpost(drzewo);
24      POST.close();
25  }
```

Listing 21. zapisz

W lini 2: Otwarcie pliku PRE.txt do zapisu.

W lini 3: Sprawdzenie, czy plik został poprawnie otwarty.

W lini 4: Informacja o błędzie otwarcia pliku.

W lini 5: Zakończenie funkcji, jeśli plik nie został otwarty.

W lini 7: Wywołanie funkcji zapiszpre do zapisania drzewa w pre-order.

W lini 8: Zamknięcie pliku PRE.txt.

W lini 10: Otwarcie pliku IN.txt do zapisu.

W lini 11: Sprawdzenie, czy plik został poprawnie otwarty.

W lini 12: Informacja o błędzie otwarcia pliku.

W lini 13: Zakończenie funkcji, jeśli plik nie został otwarty.

W lini 15: Wywołanie funkcji zapiszin do zapisania drzewa w in-order.

W lini 16: Zamknięcie pliku IN.txt.

W lini 18: Otwarcie pliku POST.txt do zapisu.

W lini 19: Sprawdzenie, czy plik został poprawnie otwarty.

W lini 20: Informacja o błędzie otwarcia pliku.

W lini 21: Zakończenie funkcji, jeśli plik nie został otwarty.

W lini 23: Wywołanie funkcji zapiszpost do zapisania drzewa w post-order.
W lini 24: Zamknięcie pliku POST.txt.

4.4.3. wczytajpre

```
1  void dopliku::wczytajpre(wezel* a) {
2      wezel* b = new wezel;
3      wezel* c = drzewo;
4      b->lewy = NULL;
5      b->prawy = NULL;
6      int t;
7      PRE >> t;
8      b->war = bintoint(t);
9      if (!c) drzewo = b;
10     else {
11         while (true) {
12             if (b->war < c->war) {
13                 if (!c->lewy) {
14                     c->lewy = b;
15                     break;
16                 }
17                 else c = c->lewy;
18             }
19             else {
20                 if (!c->prawy) {
21                     c->prawy = b;
22                     break;
23                 }
24                 else c = c->prawy;
25             }
26         }
27     }
28     b->pop = c;
29     if (!PRE.eof()) {
30         wczytajpre(a);
31     }
32 }
```

Listing 22. wczytajpre

W lini 2: Tworzy nowy węzeł b.
W lini 3: Ustawia wskaźnik c na korzeń drzewa.
W lini 4: Ustawia lewego potomka b na NULL.

W lini 5: Ustawia prawego potomka b na NULL.
W lini 6: Deklaruje zmienną t do przechowywania danych z pliku.
W lini 7: Wczytuje wartość z pliku PRE.txt.
W lini 8: Przekształca wartość t z binarnej do dziesiętnej i przypisuje do b-¿war.
W lini 9: Jeśli drzewo jest puste, ustawia b jako korzeń.
W lini 10: Jeśli drzewo nie jest puste, szuka odpowiedniego miejsca.
W lini 12: Jeśli wartość b-¿war jest mniejsza od c-¿war, przechodzi w lewo.
W lini 13: Jeśli lewy potomek c jest NULL, ustawia b jako lewy potomek.
W lini 17: W przeciwnym razie kontynuuje w lewo.
W lini 19: Jeśli wartość b-¿war jest większa lub równa c-¿war, przechodzi w prawo.
W lini 20: Jeśli prawy potomek c jest NULL, ustawia b jako prawy potomek.
W lini 24: W przeciwnym razie kontynuuje w prawo.
W lini 28: Ustawia pop nowego węzła b na c.
W lini 29: Jeśli nie osiągnięto końca pliku, wywołuje funkcję rekurencyjnie.

4.4.4. wczytajin

```
1  void dopliku::wczytajin(wezel* a) {
2      wezel* b = new wezel;
3      wezel* c = drzewo;
4      b->lewy = NULL;
5      b->prawy = NULL;
6      int t;
7      IN >> t;
8      b->war = bintoint(t);
9      if (!c) drzewo = b;
10     else {
11         while (true) {
12             if (b->war < c->war) {
13                 if (!c->lewy) {
14                     c->lewy = b;
15                     break;
16                 }
17                 else c = c->lewy;
18             }
19             else {
20                 if (!c->prawy) {
21                     c->prawy = b;
22                     break;
23                 }
24                 else c = c->prawy;
```

```
            }
        }
    }
    b->pop = c;
    if (!IN.eof()) {
        wczytajin(a);
    }
}
```

Listing 23. wczytajin

Metoda ta działa na tej samej zasadzie co metoda 4.4.3 , zmienia się tylko plik.

4.4.5. wczytajpost

```
void dopliku::wczytajpost(wezel* a) {
    wezel* b = new wezel;
    wezel* c = drzewo;
    b->lewy = NULL;
    b->prawy = NULL;
    int t;
    POST >> t;
    b->war = bintoint(t);
    if (!c) drzewo = b;
    else {
        while (true) {
            if (b->war < c->war) {
                if (!c->lewy) {
                    c->lewy = b;
                    break;
                }
                else c = c->lewy;
            }
            else {
                if (!c->prawy) {
                    c->prawy = b;
                    break;
                }
                else c = c->prawy;
            }
        }
    }
    b->pop = c;
    if (!POST.eof()) {
        wczytajpost(a);
```

```
31      }
32  }
```

Listing 24. wczytajpost

Metoda ta działa na tej samej zasadzie co metoda 4.4.3 , zmienia się tylko plik.

4.4.6. zapiszpre

```
1  void dopliku::zapiszpre(wezel* a) {
2      if (a) {
3          PRE << " " << inttobin(a->war);
4          zapiszpre(a->lewy);
5          zapiszpre(a->prawy);
6      }
7  }
```

Listing 25. zapiszpre

W lini 2: Sprawdza, czy węzeł a nie jest NULL.
W lini 3: Zapisuje wartość węzła a w formacie binarnym do pliku PRE.txt.
W lini 4: Rekurencyjnie zapisuje lewego potomka.
W lini 5: Rekurencyjnie zapisuje prawego potomka.

4.4.7. zapiszin

```
1  void dopliku::zapiszin(wezel* a) {
2      if (a) {
3          zapiszin(a->lewy);
4          IN << " " << inttobin(a->war);
5          zapiszin(a->prawy);
6      }
7  }
```

Listing 26. zapiszin

Metoda ta działa na tej samej zasadzie co metoda 4.4.6 , zmienia się tylko plik.

4.4.8. zapiszpost

```
1  void dopliku::zapiszpost(wezel* a) {
2      if (a) {
3          zapiszpost(a->lewy);
4          zapiszpost(a->prawy);
```

```
        POST << " " << inttobin(a->war);
    }
}
```

Listing 27. zapiszpost

Metoda ta działa na tej samej zasadzie co metoda 4.4.6 , zmienia się tylko plik.

4.4.9. wczytaj

```
void dopliku::wczytaj() {
    std::cout << "\n PRE-1, IN-2, POST-3 \n wybierz z ktorej metody ma byc wczytane:";
    int x;
    std::cin >> x;
    if (x == 1) {
        PRE.open("PRE.txt", std::ios::in); // Open file once here
        if (!PRE.is_open()) {
            std::cout << "Nie ma dostepu do pliku PRE.txt" << std::endl;
            return;
        }
        wczytajpre(drzewo);
        PRE.close(); // Close file after traversal is complete
    }
    else if (x == 2) {
        IN.open("IN.txt", std::ios::in); // Open file once here
        if (!IN.is_open()) {
            std::cout << "Nie ma dostepu do pliku IN.txt" << std::endl;
            return;
        }
        wczytajin(drzewo);
        IN.close(); // Close file after traversal is complete
    }
    else if (x == 3) {
        POST.open("POST.txt", std::ios::in); // Open file once here
        if (!POST.is_open()) {
            std::cout << "Nie ma dostepu do pliku POST.txt" << std::endl;
            return;
        }
        wczytajpost(drzewo);
        POST.close(); // Close file after traversal is complete
    }
```

```
32      else {
33          std::cout << "Wprowadzony zly znak, wprowadz jeszcze raz";
34          wczytaj();
35      }
36  }
```

Listing 28. wczytaj

W lini 2: Wyświetla użytkownikowi dostępne opcje wyboru metody wczytywania danych.
W lini 3: Zmienna x, która będzie przechowywać wybór użytkownika.
W lini 4: Odczytuje wybór użytkownika z konsoli.
W lini 5: Jeśli użytkownik wybierze metodę PRE (pre-order):
W lini 6: Otwiera plik PRE.txt w trybie odczytu.
W lini 7: Sprawdza, czy plik został poprawnie otwarty.
W lini 8: Wyświetla komunikat o błędzie, jeśli plik nie jest dostępny.
W lini 9: Zatrzymuje dalsze wykonywanie funkcji.
W lini 11: Wywołuje funkcję ‘wczytajpre()‘, aby wczytać dane w porządku pre-order.
W lini 12: Zamknięcie pliku PRE.txt po zakończeniu operacji.
W lini 14: Jeśli użytkownik wybierze metodę IN (in-order):
W lini 15: Otwiera plik IN.txt w trybie odczytu.
W lini 16: Sprawdza, czy plik został poprawnie otwarty.
W lini 17: Komunikat o błędzie, jeśli plik jest niedostępny.
W lini 18: Zatrzymuje dalsze wykonywanie funkcji.
W lini 20: Wywołuje funkcję ‘wczytajin()‘, aby wczytać dane w porządku in-order.
W lini 21: Zamknięcie pliku IN.txt po zakończeniu operacji.
W lini 23: Jeśli użytkownik wybierze metodę POST (post-order):
W lini 24: Otwiera plik POST.txt w trybie odczytu.
W lini 25: Sprawdza, czy plik został poprawnie otwarty.
W lini 26: Komunikat o błędzie, jeśli plik jest niedostępny.
W lini 27: Zatrzymuje dalsze wykonywanie funkcji.
W lini 29: Wywołuje funkcję ‘wczytajpost()‘, aby wczytać dane w porządku post-order.
W lini 30: Zamknięcie pliku POST.txt po zakończeniu operacji.
W lini 32: Jeśli użytkownik wprowadzi niepoprawną opcję:
W lini 33: Wyświetla komunikat o błędzie.
W lini 34: Rekurencyjnie wywołuje funkcję, aby umożliwić ponowny wybór.

4.4.10. inttobin

```
1  int dopliku::inttobin(int a) {
2      int b;
3      std::string c;
4      while (a) {
5          c = (a % 2 ? "1" : "0") + c;
6          a /= 2;
7      }
8      b = std::stoi(c);
9      return b;
10 }
```

Listing 29. inttobin

W lini 2: Deklaruje zmienną b, która będzie przechowywać wynik.
W lini 3: Zmienna pomocnicza c do przechowywania binarnej reprezentacji.
W lini 4: Pętla, która konwertuje liczbę całkowitą na binarną.
W lini 5: Dodaje '1' lub '0' do c w zależności od reszty z dzielenia przez 2.
W lini 6: Dzieli a przez 2.
W lini 8: Konwertuje wynik w stringu na liczbę całkowitą.
W lini 9: Zwraca wynik konwersji.

4.4.11. bintoint

```
1  int dopliku::bintoint(int a) {
2      int b = 0;
3      int c = 1;
4      std::string d = std::to_string(a);
5      for (int i = d.length() - 1; i >= 0; --i, c <<= 1) {
6          b += (d[i] - '0') * c;
7      }
8      return b;
9  }
```

Listing 30. bintoint

W lini 2: Zmienna do przechowywania wyniku w systemie dziesiętnym.
W lini 3: Zmienna pomocnicza do przesuwania bitów.
W lini 4: Zmienna pomocnicza do przechowywania binarnej reprezentacji.
W lini 5: Pętla przetwarzająca każdy bit.
W lini 6: Dodaje odpowiednią wartość do b.

W lini 8: Zwraca wynik konwersji na liczbę dziesiętną.

4.5. Plik main.cpp

4.5.1. zmienne

```
1 int main(int argc, char const *argv[]) {
2     int a = 0;
3     int b = 0;
4     bst e;
5     int r;
```

Listing 31. zmienne

W lini 2: Zmienna ‘a‘ do przechowywania wyboru użytkownika w menu.
W lini 3: Zmienna ‘b‘ kontrolująca pętlę menu, używana do zakończenia programu.
W lini 4: Tworzy obiekt klasy ‘bst‘ (drzewo binarne) o nazwie ‘e‘.
W lini 5: Zmienna ‘r‘ do przechowywania wartości elementu podawanego przez użytkownika.

4.5.2. switch

```
1 while (b == 0) {
2         std::cout << "\033[2J\033[1;1H";
3         std::cout << "Co chcesz zrobic? \n";
4         std::cout << "1 - Dodaj element do drzewa\n2 - Usun element
    z drzewa\n3 - Usun cale drzewo\n4 - Szukaj drogi do elementu\n5
    - Wyswietl drzewo\n6 - Zapisz do pliku\n7 - Wczytaj z pliku\n8
   - Zakoncz program \nPodaj operacje: ";
5         std::cin >> a;
6         switch (a) {
7         case 1:
8             std::cout << "Podaj element: ";
9             std::cin >> r;
10            e.dodele(r);
11            std::cout << "Udalo sie! \n";
12            system("pause");
13            break;
14        case 2:
15            std::cout << "Podaj element: ";
16            std::cin >> r;
17            e.usunele(r);
```

```
18              std::cout << "Udalo sie! \n";
19              system("pause");
20              break;
21          case 3:
22              e.usubst(e.korzen);
23              std::cout << "Udalo sie! \n";
24              system("pause");
25              break;
26          case 4:
27              std::cout << "Podaj element: ";
28              std::cin >> r;
29              e.szuk(r);
30              std::cout << "Udalo sie! \n";
31              system("pause");
32              break;
33          case 5:
34              int y;
35              std::cout << "\nPRE-1, IN-2, POST-3 \nWybierz z ktorej
    metody ma byc wypisane: ";
36              std::cin >> y;
37              switch (y) {
38              case 1:
39                  e.wyspre(e.korzen);
40                  break;
41              case 2:
42                  e.wysin(e.korzen);
43                  break;
44              case 3:
45                  e.wyspost(e.korzen);
46                  break;
47              default:
48                  std::cout << "Sprobuj jeszcze raz!";
49                  break;
50              }
51              std::cout << "\nUdalo sie! \n";
52              system("pause");
53              break;
54          case 6:
55              e.zapisz();
56              std::cout << "Udalo sie! \n";
57              system("pause");
58              break;
59          case 7:
60              e.wczytaj();
61              std::cout << "Udalo sie! \n";
```

```
62              system("pause");
63              break;
64          case 8:
65              b = 1;
66              break;
67          default:
68              std::cout << "Zla komenda, powtorz\n";
69              system("pause");
70              break;
71          }
72      }
73  }
```

Listing 32. switch

Switch pozwala na wybór akcji którą chcesz wykonać na drzewie.
Każda opcja wywołuje odpowiednią metodę i wyświetla komunikat czy sie udało.

5. Wnioski

Implementacja drzewa BST w języku C++ została zrealizowana pomyślnie, a wszystkie kluczowe funkcje, takie jak dodawanie, wyszukiwanie i usuwanie węzłów, działają poprawnie. Realizacja projektu pozwoliła na praktyczne pogłębienie wiedzy o strukturach danych oraz na rozwój umiejętności programowania w C++. Napotkane trudności przyczyniły się do lepszego zrozumienia działania drzew binarnych i opanowania zaawansowanych koncepcji programistycznych.

W trakcie pracy wykorzystano platformę GitHub, co umożliwiło zdobycie praktycznych umiejętności w zarządzaniu wersjami i współpracy na repozytoriach. Zastosowanie Git ułatwiło organizację kodu, śledzenie zmian oraz rozwiązywanie błędów. Dokumentację programu wygenerowano przy użyciu narzędzia Doxygen, co znacznie zwiększyło przejrzystość projektu i umożliwiło sprawniejszą pracę z kodem.

Projekt ten nie tylko wykazał korzyści płynące z użycia drzewa BST do efektywnego zarządzania danymi, ale także uwydatnił znaczenie narzędzi wspomagających proces tworzenia oprogramowania. Zdobyte doświadczenia będą pomocne w przyszłych projektach programistycznych oraz na dalszych etapach nauki i pracy zawodowej.

Bibliografia

[1] *Rysunek Wikipedia Drzewo BST*. URL: https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84#/media/Plik:Binary_search_tree.svg (term. wiz. 06. 11. 2024).

[2] *Rysunek Wikipedia Wyszukiwanie Drzewo BST*. URL: https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84#/media/Plik:Binary_search_tree_search_4.svg (term. wiz. 06. 11. 2024).

[3] *Rysunek Wikipedia Usunięcie węzła który jest liściem*. URL: https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84#/media/Plik:Binary_search_tree_delete_13.svg (term. wiz. 06. 11. 2024).

[4] *Rysunek Wikipedia Usunięcie węzła który ma jedno dziecko*. URL: https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84#/media/Plik:Binary_search_tree_delete_14.svg (term. wiz. 06. 11. 2024).

[5] *Rysunek Wikipedia Usunięcie węzła który ma dwoje dzieci*. URL: https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84#/media/Plik:Binary_search_tree_delete_3.svg (term. wiz. 06. 11. 2024).

Spis rysunków

Spis tabel

Spis listingów