AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Drzewo BST działające na stercie w języku C++

Autor:
Mateusz Smaga
Kamil Trzópek

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Zadanie w proejkcie jest napisanie programu przedstawiające strukturę „drzewa BST" działającego 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).Funkcja main powinna wyświetlać menu z opcjami drzewa oraz odczytu i zapisu pliku. Program czeka na wybranie opcji.Wszystkie utworzone klasy mają być zaimplementowane w oddzielnych plikach. Funkcja main także powinna być w osobnym pliku.

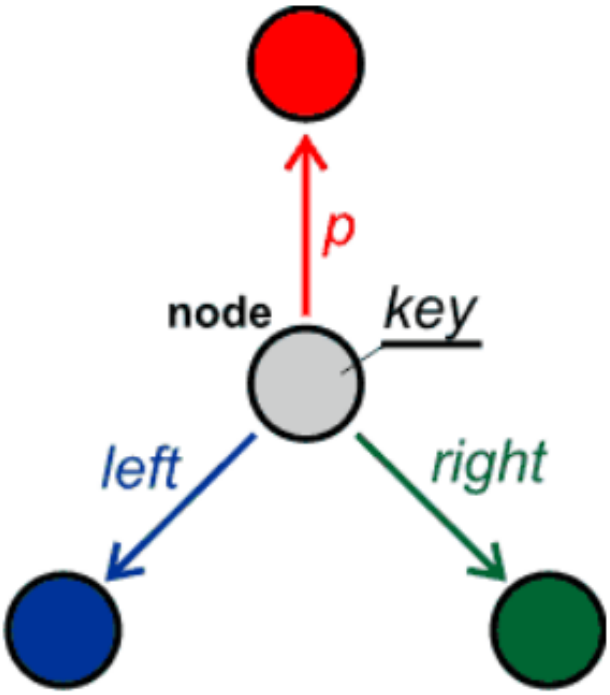
Przy oddawaniu projektu należy zaprezentować:

- - 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

Drzewa poszukiwań binarnych - BST

Drzewo poszukiwań binarnych BST (ang. Binary Search Tree) jest dynamiczną strukturą danych zbudowaną z węzłów (ang. node). Każdy węzeł może posiadać dwóch potomków (left - lewy i right - prawy) oraz jednego przodka (p - parent)(2.1). Z każdym węzłem dodatkowo związany jest klucz (key) (link do strony z której pochadzą grafiki, oraz informacje [1])

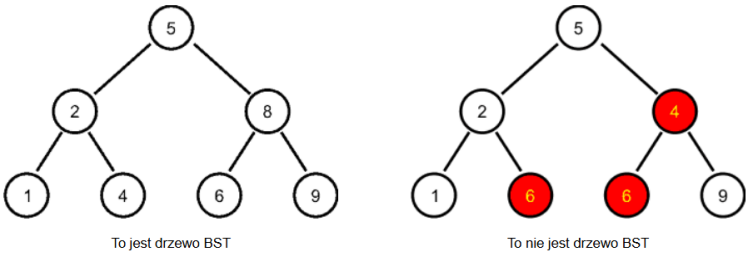


Rys. 2.1. node schemat

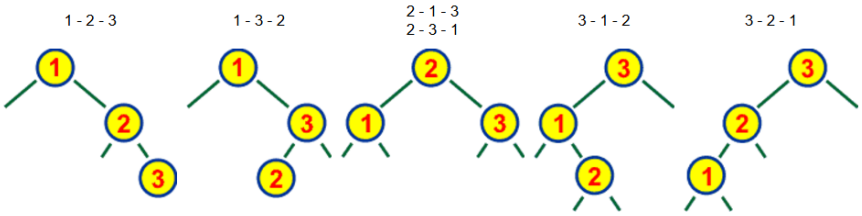
Dla każdego węzła w drzewie BST zachodzą następujące własności 2.2:

- Wartości kluczy węzłów leżących w prawym poddrzewie węzła są większe lub równe wartości klucza danego węzła.
- Wartości kluczy węzłów leżących w lewym poddrzewie węzła są mniejsze lub równe wartości klucza danego węzła.

Ważnym jest że zależności od kolejności wprowadzania danych do drzewa BST mogą powstać różne konfiguracje węzłów 2.3.



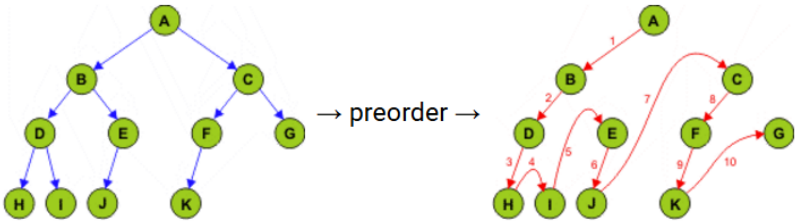
Rys. 2.2. node schemat



Rys. 2.3. node róznica

Przedstawienie działania metod sotrowań drzewa BTS: (postorder,inorder, preorder) (strona referencyjna: [2])

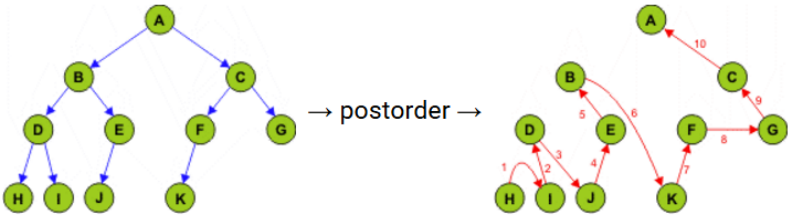
Preorder to metoda przeglądania drzewa binarnego, w której odwiedza się węzeł bieżący przed jego lewym i prawym poddrzewem (kolejność: węzeł -¿ lewe poddrzewo -¿ prawe poddrzewo). Stosuje się ją często w rekursywnych algorytmach przetwarzania drzew, odwiedzając każdy węzeł dokładnie raz (rys: 2.4).



Rys. 2.4. preorder

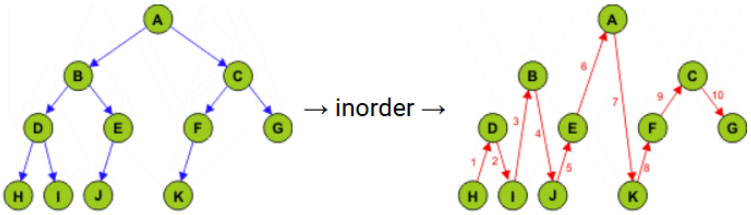
Postorder to metoda przeglądania drzewa binarnego, w której odwiedza się najpierw lewe poddrzewo, potem prawe poddrzewo, a na końcu bieżący węzeł (kolejność: lewe poddrzewo -¿ prawe poddrzewo -¿ węzeł). Ta metoda jest używana m.in. do usuwania drzewa lub oceny wyrażeń w notacji polskiej odwrotnej (rys: 2.5).

Inorder to metoda przeglądania drzewa binarnego, w której odwiedza się najpierw lewe poddrzewo, następnie bieżący węzeł, a na końcu prawe poddrzewo (ko-



Rys. 2.5. postorder

lejność: lewe poddrzewo -¿ węzeł -¿ prawe poddrzewo). Stosuje się ją, gdy chce się uzyskać elementy drzewa w kolejności rosnącej, np. w drzewach poszukiwań binarnych (BST) (rys: 2.6).



Rys. 2.6. inorder

```
1
2
3  int BSTminkey(BSTNode * root)
4  {
5    BSTNode * x = root;
6
7    while((x->left) x = x->left;
8
9    return x->key;
10 }
```

Listing 1. BTS

Wyszukiwanie rozpoczynamy od korzenia drzewa BST, przechodzimy przez poszczególne węzły drzewa BST zaawsze wybierając lewo (listninig:1)

```
1
2  struct Node {
3      int key;
4      Node* left;
5      Node* right;
```

```
6       Node(int k) : key(k), left(nullptr), right(nullptr) {}
7   };
8
9   Node* insert(Node* root, int key) {
10      if (!root)
11          return new Node(key);
12      if (key < root->key)
13          root->left = insert(root->left, key);
14      else
15          root->right = insert(root->right, key);
16      return root;
17  }
```

Listing 2. BTS

Załorzenie konstrukcji wsawiania do drzewa BTS (listninig:2)

```
1
2   Node* search(Node* root, int key) {
3       if (!root || root->key == key)
4           return root;
5       if (key < root->key)
6           return search(root->left, key);
7       return search(root->right, key);
8   }
```

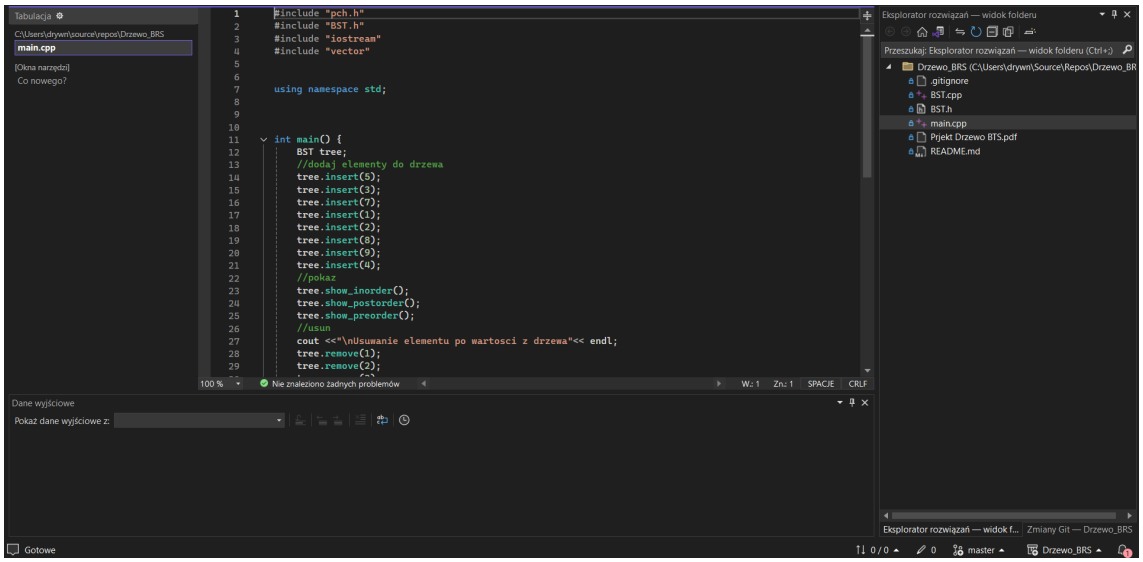
Listing 3. BTS

Przykładowa konsterukcja kodu do Wyszukiwania w dzrewie BTS 3

3. Projektowanie

Program do projektu został napisany w języku programowania C++, przy użyciu edytora Visual Studio Code 3.1. Repozytorium projektu zostało utworzone w serwisie Github (link [3]), także dokumentacja została wygenerowana automatycznie przy użyciu "Doxywizard"obsługiwana przez program doxygen. (link do pobrania doxygen [4]) (link Visual Studio: [5])

Program został wykonany w edytorze kodu visual studio 2022 3.1. Edytor został podpięty do repetytorium git (sklonowany), oraz edytowany i zapisywany krok po kroku.



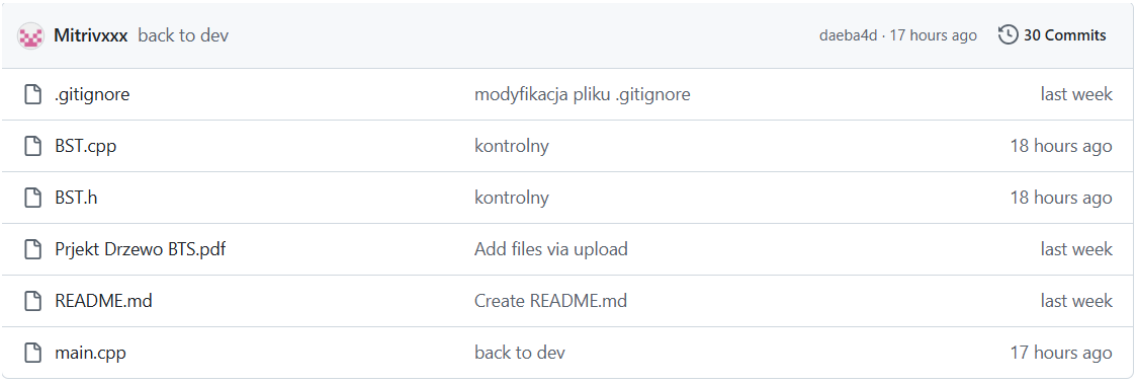
Rys. 3.1. visual 1

Projekt Drzewa BTS został wykonany wspolnie za pomocą githuba na róznych gałęziach pozwalających na wspolną pracę takich jak master (całość już działającego kodu) 3.2, dev (gałąź zawierającą całośc zmian przed omówieniem, oraz po)3.3, latex (osobna gałąź przechowująca kod latex,doxygen, oraz pdf z dokumentacją, oraz zmiany w niej wprowadzane) 3.4

Całość konstrukcji repetytorium można zabaczyć za zrzucie 3.5, przedstawiające powiazania i konflikty między dev a master, oraz odosobinienie gałęzi latex.

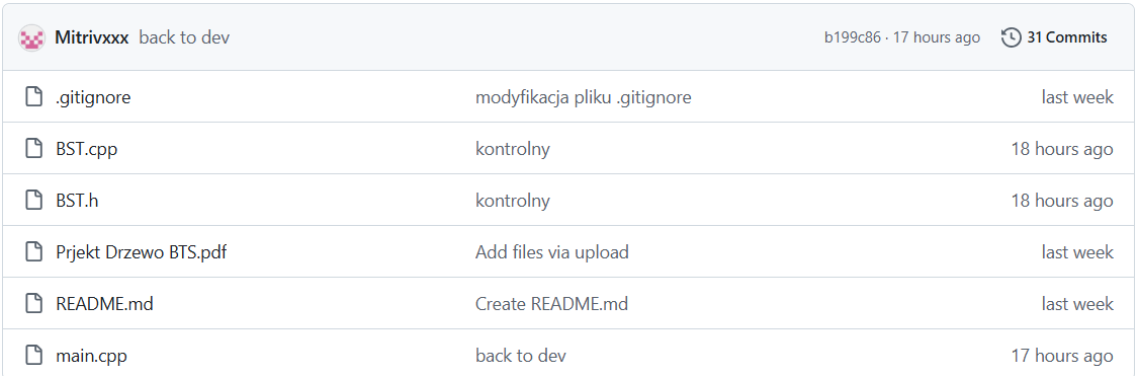
Przedstawienie przykładowego konfliktu (rys: 3.6) po przez narzędzie github, oraz przedstawienie jego roziwiązania i wpływu na wykres (rys: 3.7)

Poza tym do projektu została wygentowania dokumentacja za pomocą programu doxywizard (doxygen), wybieranie z czego ma zastać wgenerowana dokumntacja 3.8,



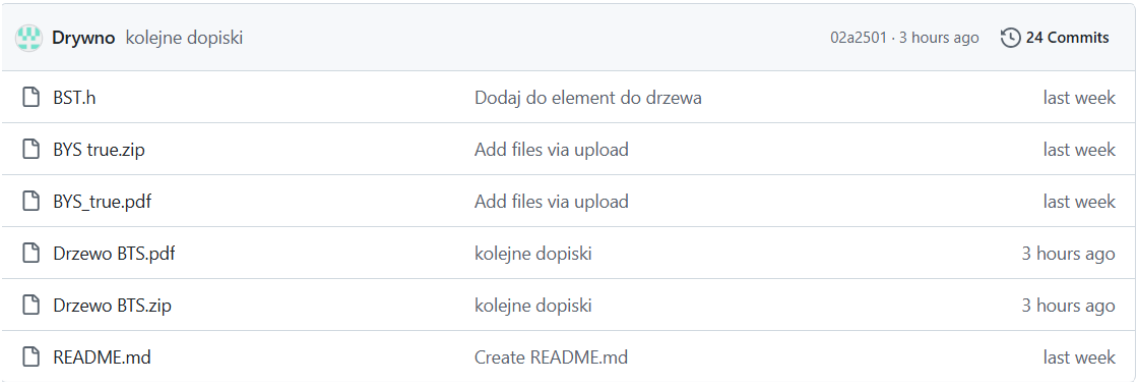
Mitrivxxx back to dev	daeba4d · 17 hours ago	30 Commits
.gitignore	modyfikacja pliku .gitignore	last week
BST.cpp	kontrolny	18 hours ago
BST.h	kontrolny	18 hours ago
Prjekt Drzewo BTS.pdf	Add files via upload	last week
README.md	Create README.md	last week
main.cpp	back to dev	17 hours ago

Rys. 3.2. github 1



Mitrivxxx back to dev	b199c86 · 17 hours ago	31 Commits
.gitignore	modyfikacja pliku .gitignore	last week
BST.cpp	kontrolny	18 hours ago
BST.h	kontrolny	18 hours ago
Prjekt Drzewo BTS.pdf	Add files via upload	last week
README.md	Create README.md	last week
main.cpp	back to dev	17 hours ago

Rys. 3.3. github 2

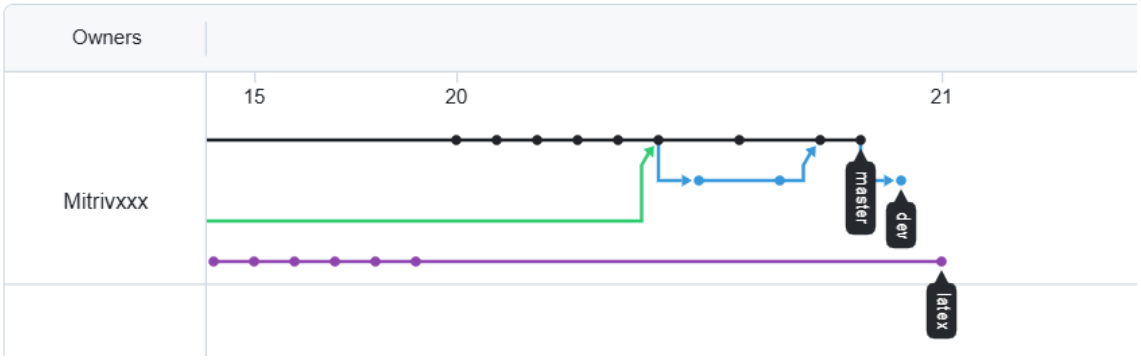


Drywno kolejne dopiski	02a2501 · 3 hours ago	24 Commits
BST.h	Dodaj do element do drzewa	last week
BYS true.zip	Add files via upload	last week
BYS_true.pdf	Add files via upload	last week
Drzewo BTS.pdf	kolejne dopiski	3 hours ago
Drzewo BTS.zip	kolejne dopiski	3 hours ago
README.md	Create README.md	last week

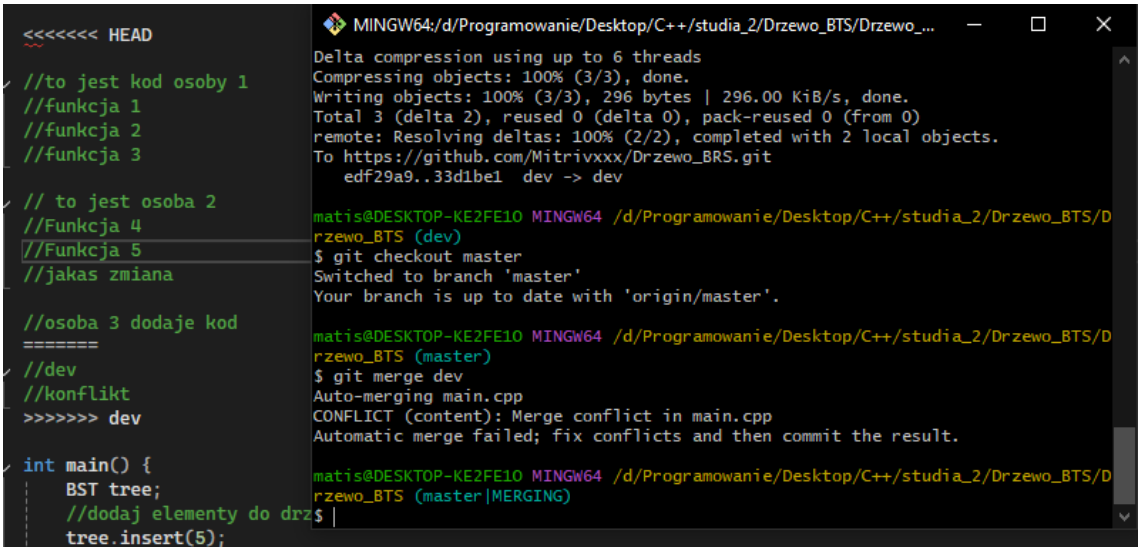
Rys. 3.4. github 3

jaki typ dokumentacji na zstac wygenetowany 3.9, oraz gdzie ma ona zostać zapisana po generacji (w razie braku jest to miejsce z którego bierzemy dane) 3.10

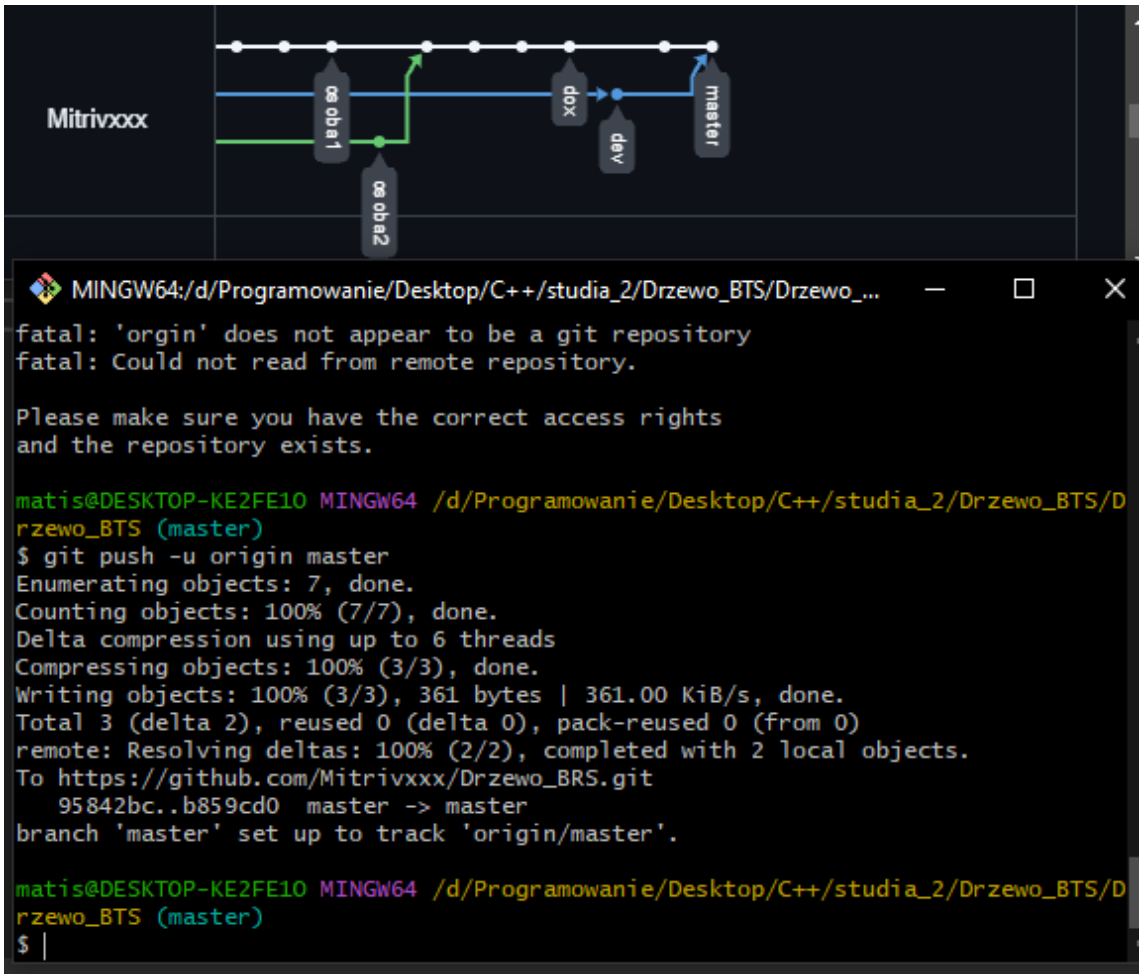
Gotowy i wygernerowany dokument w doxygen (rys: 3.11)



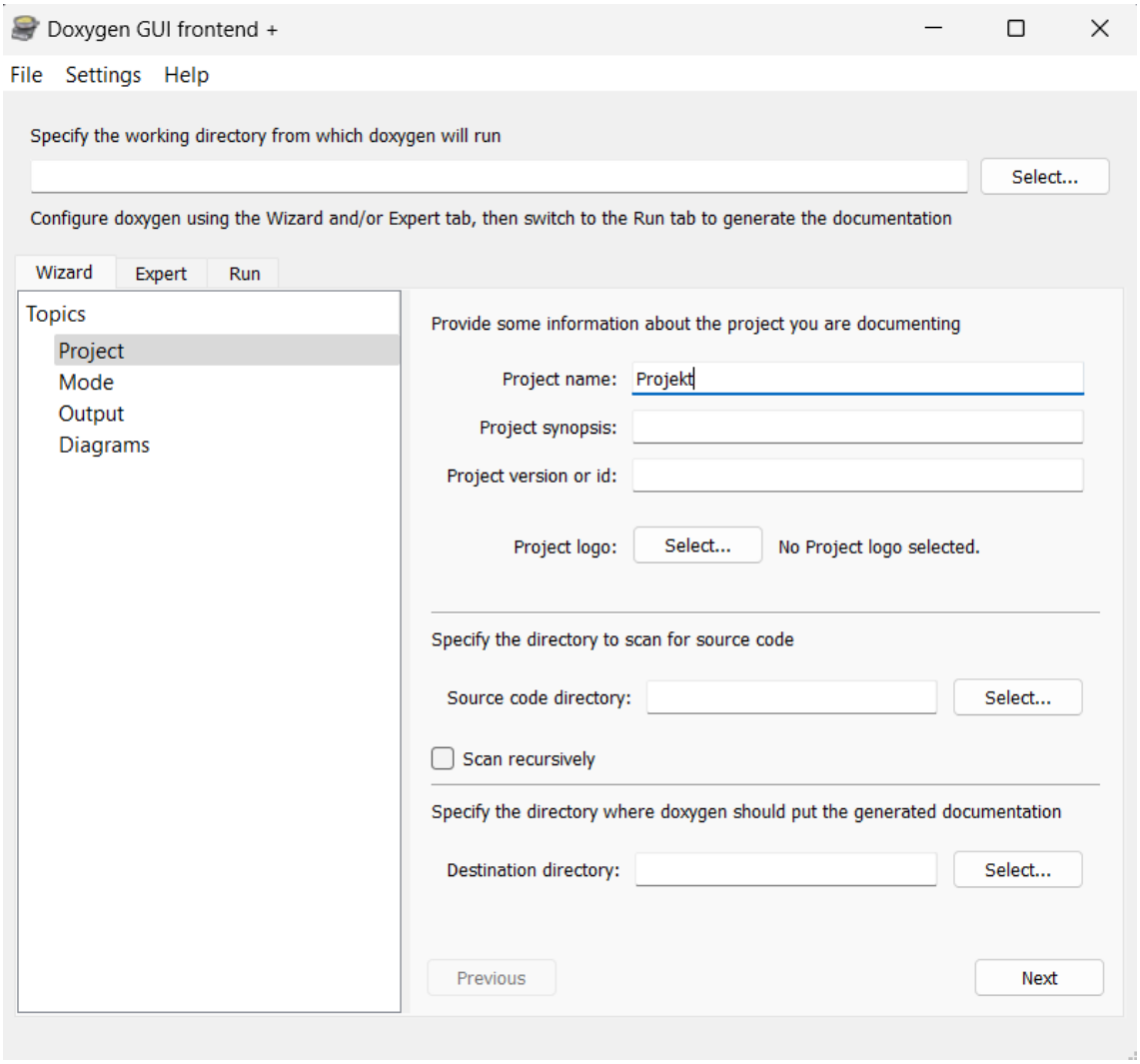
Rys. 3.5. github wykres drzewa



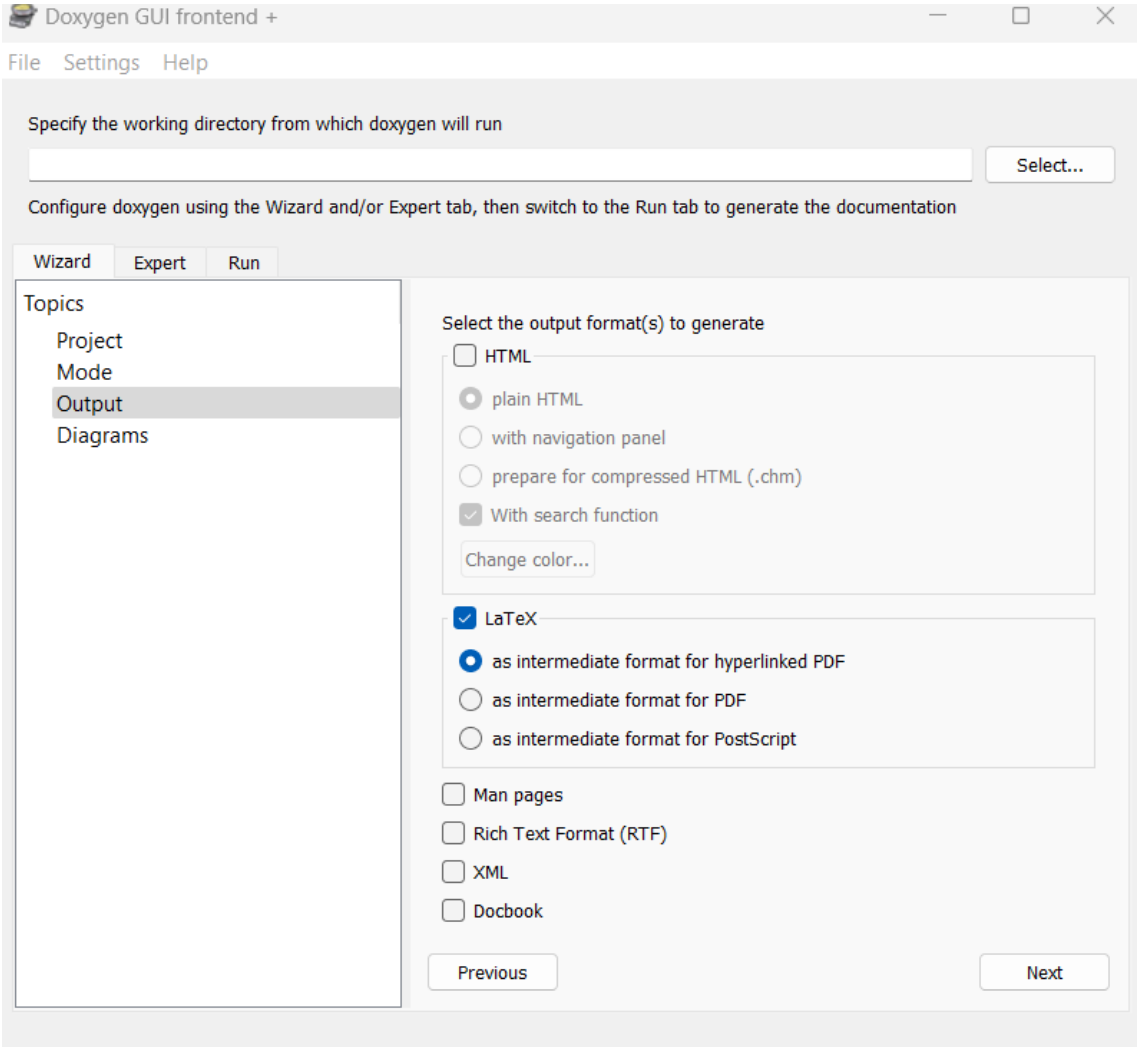
Rys. 3.6. konflikt



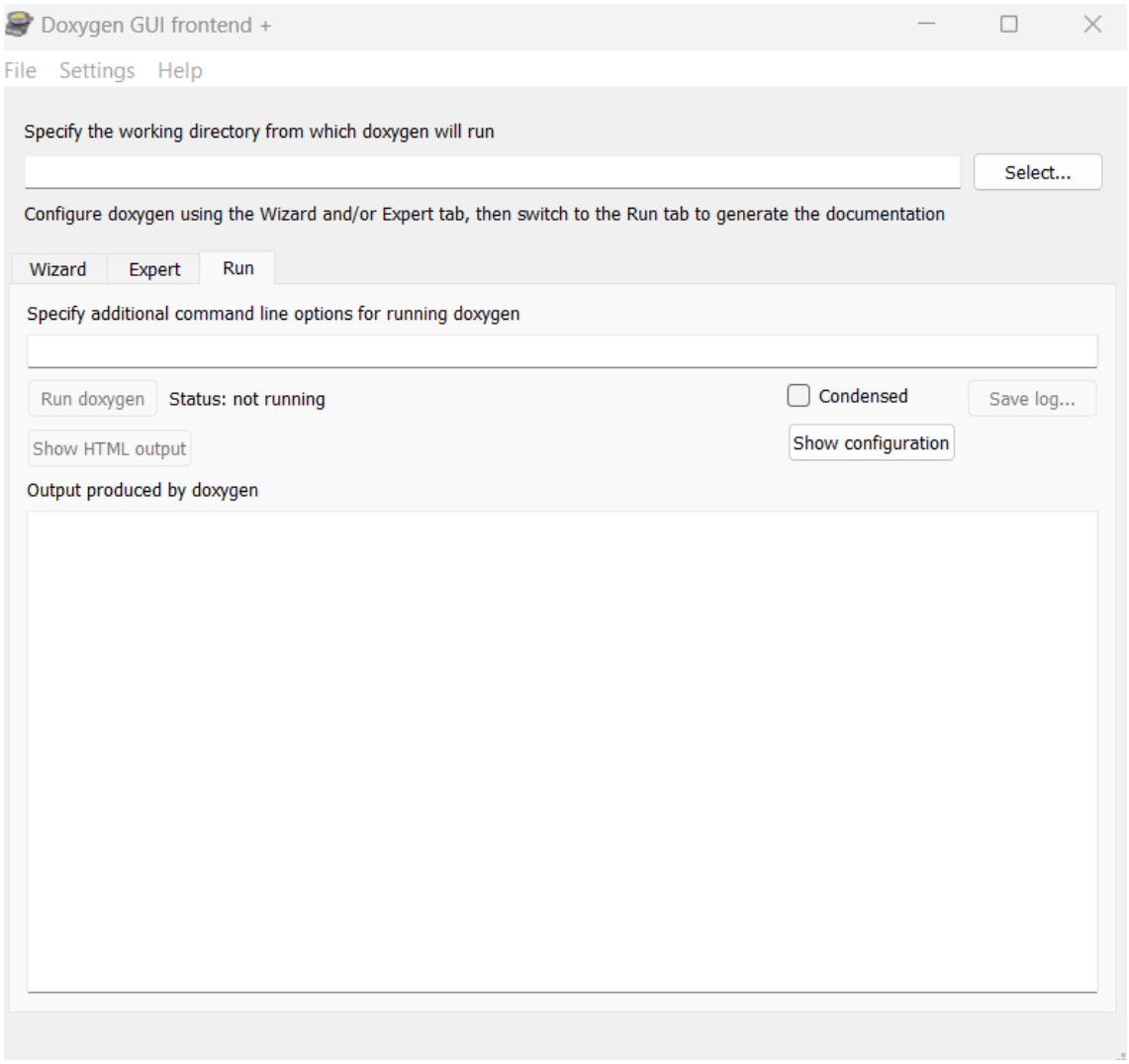
Rys. 3.7. konflikt rozwiązanie


Doxygen GUI frontend +
File Settings Help
Specify the working directory from which doxygen will run
Select...
Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard Expert Run
Topics
Project
Mode
Output
Diagrams
Provide some information about the project you are documenting
Project name: Projekt
Project synopsis:
Project version or id:
Project logo: Select... No Project logo selected.
Specify the directory to scan for source code
Source code directory: Select...
Scan recursively
Specify the directory where doxygen should put the generated documentation
Destination directory: Select...
Previous
Next

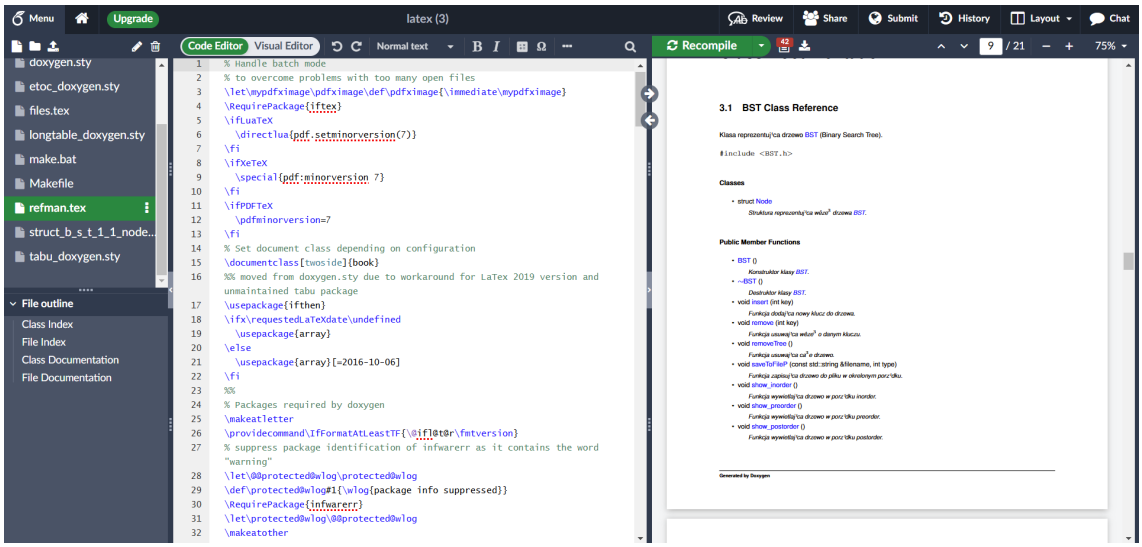
Rys. 3.8. dox1


Doxygen GUI frontend +
File
Settings
Help
Specify the working directory from which doxygen will run
Select...
Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Select the output format(s) to generate
HTML
plain HTML
with navigation panel
prepare for compressed HTML (.chm)
With search function
Change color...
LaTeX
as intermediate format for hyperlinked PDF
as intermediate format for PDF
as intermediate format for PostScript
Man pages
Rich Text Format (RTF)
XML
Docbook
Previous
Next

Rys. 3.9. dox2



Rys. 3.10. dox3



Rys. 3.11. doxygen dokument

4. Implementacja

Implementacja całego drzewa zawarta w pliku BTS.h (listing: 4) pozwalająca na wywoływanie m.in: odawanie (insert), usuwanie (remove), znajdowanie minimalnego węzła (findMin), sortowanie drzewa.

```
1
2  class BST {
3  private:
4      struct Node {
5          Node* p;         // Wska nik na rodzica
6          Node* left;      // Wska nik na lewego potomka
7          Node* right;     // Wska nik na prawego potomka
8          int key;         // Klucz (warto Ż  w Źz Ća)
9
10         // Konstruktor dla u Ćatwienia inicjalizacji w Źz Ć w
11         Node(int k) : p(nullptr), left(nullptr), right(nullptr),
       key(k) {}
12     };
13     Node* root;      //wskaznik na korzen drzewa
14
15     Node* insert(Node* current, int key);        //dodaj do drzewa
16     Node* remove(Node* current, int key); // Rekurencyjna funkcja
       usuwania
17     Node* findMin(Node* current);
18     void removeTree(Node* current); // usun cale drzewo
19
20
21     //ukladanie drzewa
22     void inorder(Node* root);
23     void preorder(Node* current);
24     void postorder(Node* current);
25
26 public:
27     // Konstruktor i destruktor
28     BST();
29     ~BST();
30     // Funkcja dodaj ca element do drzewa
31     void insert(int key);
32
33     //Funkcja usuwaj ca element
34     void remove(int key);
35     void removeTree();
36     //Funkcja szukajaca drogi do podaniego elementu
```

```
37
38      //niewiem o co chodzi
39
40      //Zapisz do pliku txt drzewo
41
42
43
44      // Funkcja wy Żwietlaj ca drzewo w porz dku inorder
45      void show_inorder();
46      void show_preorder();
47      void show_postorder();
48
49
50  };
```

Listing 4. BTS szkielet

Przedstawianie wywołania konstrukcji drzewa BTS znajdujące się w pliku main.cpp po przez metode tree dla fubnkcji BTS (listing: 5)

```
1
2   int main() {
3       BST tree;
4       //dodaj elementy do drzewa
5       tree.insert(5);
6       tree.insert(3);
7       tree.insert(7);
8       tree.insert(1);
9       tree.insert(2);
10      tree.insert(8);
11      tree.insert(9);
12      tree.insert(4);
13      //pokaz
14      tree.show_inorder();
15      tree.show_postorder();
16      tree.show_preorder();
17      //usun
18      cout <<"\nUsuwanie elementu po wartosci z drzewa"<< endl;
19      tree.remove(1);
20      tree.remove(2);
21      tree.remove(3);
22      tree.show_inorder();
23      cout << "\nUsun cale drzwo" << endl;
24      tree.removeTree();
25      tree.show_inorder();
26
```

```
27
28      return 0;
29  }
```

Listing 5. BTS wywołaj

Konstruktor, oraz destruktor klasy BTS, oraz podpięte bibliotek (iostream, vector), pliki (pch.h, oraz BST.h) (listing: 6)

```
1
2   #include "pch.h"
3   #include "BST.h"
4   #include "iostream"
5   #include "vector"
6
7   using namespace std;
8
9
10  // Konstruktor klasy BST
11  BST::BST() : root(nullptr) {}
12
13  // Destruktor klasy BST
14  BST::~BST() {}
```

Listing 6. BTS konstrukcja

Dodawanie do drzewa BTS (listing:7). Funkcja BST::insert wstawia nowy klucz do drzewa BST, tworząc nowy węzeł, gdy bieżący węzeł (current) jest pusty, lub rekurencyjnie wywołując się na lewym lub prawym poddrzewie w zależności od wartości klucza, aktualizując odpowiednio wskaźniki rodzica, a na końcu zwraca bieżący węzeł.

```
1
2   // dodaj do drzewa
3   BST::Node* BST::insert(Node* current, int key) {
4       if (current == nullptr) {
5           return new Node(key); // Tworzymy nowy w Źze Ć
6       }
7
8       if (key < current->key) {
9           Node* leftChild = insert(current->left, key);
10          current->left = leftChild;
11          leftChild->p = current; // Ustawiamy rodzica
12      }
13      else if (key > current->key) {
14          Node* rightChild = insert(current->right, key);
15          current->right = rightChild;
```

```
        rightChild->p = current; // Ustawiamy rodzica
    }

    return current; // Zwracamy bie   cy  w Źze Ć
}
```

Listing 7. BTS dodaj

Usuwanie z drzewa BTS (listing: 8) Funkcja BST::remove usuwa węzeł o podanym kluczu z drzewa BST, zaczynając od sprawdzenia, czy węzeł istnieje, następnie rekurencyjnie przeszukując lewe lub prawe poddrzewo, a po znalezieniu klucza usuwa węzeł w zależności od jego struktury: jeśli jest liściem, po prostu go usuwa, jeśli ma jedno dziecko, zastępuje go tym dzieckiem, a jeśli ma dwoje dzieci, znajduje jego następnika (najmniejszy węzeł w prawym poddrzewie), kopiuje jego klucz, a następnie usuwa następnika z drzewa.

```

BST::Node* BST::remove(Node* current, int key) {
    if (current == nullptr) {
        return nullptr; // Klucz nie znaleziony
    }

    if (key < current->key) {
        current->left = remove(current->left, key);
    }
    else if (key > current->key) {
        current->right = remove(current->right, key);
    }
    else {
        // Klucz znaleziony
        if (current->left == nullptr && current->right == nullptr)
   {
            // W Źze Ć jest li Żciem
            delete current;
            return nullptr;
        }
        else if (current->left == nullptr) {
            // W Źze Ć ma jedno dziecko (prawe)
            Node* temp = current->right;
            delete current;
            return temp;
        }
        else if (current->right == nullptr) {
            // W Źze Ć ma jedno dziecko (lewe)
            Node* temp = current->left;
```

```
29              delete current;
30              return temp;
31          }
32          else {
33              // W Źze Ć ma dwoje dzieci
34              Node* minNode = findMin(current->right); // Znajd
    nast Źpnika
35              current->key = minNode->key;            // Przepisz
    klucz
36              current->right = remove(current->right, minNode->key);
    // Usu  nast Źpnika
37          }
38      }
39      return current;
```

Listing 8. BTS usuń

Funckja układająca drzewo w sposób: inorder (listing: 9)

```
1
2 void BST::inorder(Node* root)
3 {
4     if (root != nullptr) {
5         inorder(root->left);
6         cout << root->key << " ";
7         inorder(root->right);
8     }
9 }
```

Listing 9. BTS inorder

Funckja układająca drzewo w sposób: preorder (listing: 10)

```
1
2 void BST::preorder(Node* current)
3 {
4     if (current != nullptr) {
5         cout << current->key << " "; // Przetw rz  bie  cy
    w Źze Ć
6         preorder(current->left);    // Przejd  do lewego poddrzewa
7         preorder(current->right);   // Przejd  do prawego
    poddrzewa
8     }
9 }
```

Listing 10. BTS preorder

Funckja układająca drzewo w sposób: postorder (listing: 11)

```
1
2  void BST::postorder(Node* current)
3  {
4      if (current != nullptr) {
5          postorder(current->left);   // Przejd  do lewego poddrzewa
6          postorder(current->right);  // Przejd  do prawego
       poddrzewa
7          cout << current->key << " "; // Przetw rz  bie   cy
       w Źze Ć
8      }
9  }
```

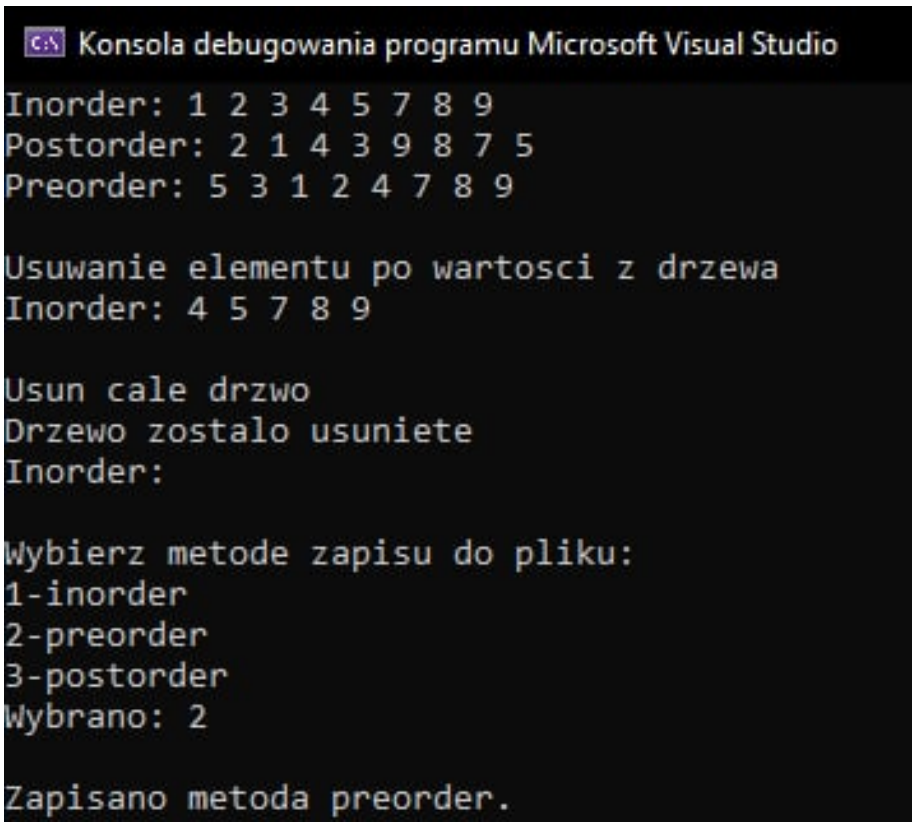
Listing 11. BTS postorder

Funkcje pomocnicze do wykononywania m.in sortowań: (listing: 12)

```
1
2  void BST::show_inorder() {
3      cout << "Inorder: ";
4      inorder(root);
5      cout << endl;
6  }
7  void BST::show_preorder(){
8      cout << "Preorder: ";
9      preorder(root);
10     cout << endl; // Dodanie nowej linii po wy Żwietleniu
11 }
12
13 void BST::show_postorder(){
14     cout << "Postorder: ";
15     postorder(root);
16     cout << endl; // Dodanie nowej linii po wy Żwietleniu
17 }
18 void BST::remove(int key) {
19     root = remove(root, key);
20 }
21
22 void BST::removeTree()
23 {
24     removeTree(root);
25     root = nullptr;
26 }
```

Listing 12. BTS

Przedstawienie działania kodu w konsoli visual Studio 2022 (obraz 4.1)



Rys. 4.1. Wynik dzialania kodu

5. Wnioski

Algorytm drzewa BST został pomyślnie zaimplementowany, a wszystkie metody działają prawidłowo (mam nadzieję). Praca nad implementacją sprawiła jedynie drobne trudności, które przyczyniły się do doskonalenia umiejętności programowania w języku C++ oraz lepszego zrozumienia struktury drzewa binarnego. Praktyczne zmagania z problemami pozwoliły także na rozwinięcie umiejętności analitycznego myślenia i debugowania kodu.

Praca w parach na GitHubie umożliwiła poznanie zasad współpracy na repozytoriach, co obejmowało zarządzanie wersjami kodu, rozwiązywanie konfliktów i organizację pracy zespołowej. Dzięki wykorzystaniu narzędzia Doxygen do generowania dokumentacji projektu, udało się lepiej uporządkować i opisać kod, co zwiększyło jego czytelność i ułatwiło współpracę.

GitHub okazał się bardzo przydatnym narzędziem, umożliwiającym m.in. śledzenie postępów prac, cofanie zmian oraz przechowywanie stabilnych wersji projektu. Wspólna praca na platformie pozwoliła na efektywną organizację zadań i szybsze rozwiązywanie problemów. W przyszłości umiejętności te znajdą zastosowanie zarówno na kolejnych kursach programistycznych, jak i w pracy zawodowej.

Bibliografia

[1] *Strona internetowa BTS*. URL: https://eduinf.waw.pl/inf/utils/002_roz/mp001.php.

[2] *Sortowania*. URL: https://eduinf.waw.pl/inf/alg/001_search/0109.php.

[3] *Strona git projektu*. URL: https://github.com/Mitrivxxx/Drzewo_BRS.

[4] *Strona internetowa Doxygen'a*. URL: https://doxygen.nl/.

[5] *Strona internetowa Visual Studio Code*. URL: https://code.visualstudio.com/.

Spis rysunków

Spis listingów