Drzwo Binarne

Generated by Doxygen 1.15.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BST Class Reference

Klasa reprezentująca drzewo BST (Binary Search Tree).

`#include <bst.h>`

Public Member Functions

- BST ()
- ~BST ()
- void insertKey (int key)

  Wstawia nowy klucz do drzewa.
- void removeKey (int key)

  Usuwa klucz z drzewa.
- void clearTree ()

  Czyści całe drzewo.
- vector< int > getPathTo (int key) const

  Zwraca ścieżkę od korzenia do danego klucza.
- vector< int > getPreorder () const

  Zwraca elementy w kolejności preorder.
- vector< int > getInorder () const

  Zwraca elementy w kolejności inorder.
- vector< int > getPostorder () const

  Zwraca elementy w kolejności postorder.
- void printTreeGraphic2 (Node *node, int indent=0, int indentStep=4) const

  Rekursywne wyświetlenie drzewa w konsoli.
- void printTreeGraphic (int indentStep=4) const

  Wyświetlenie drzewa w konsoli.
- BST::Node * getRoot () const

  Zwraca wskaźnik do korzenia.

Static Public Member Functions

- static void printVectorTraversal (const vector< int > &v)

  Wypisuje wektor kluczy na ekran.

Friends

- class BSTFileHandler

  Klasa przyjaciół umożliwiająca dostęp do prywatnych elementów drzewa.

3.1.1 Detailed Description

Klasa reprezentująca drzewo BST (Binary Search Tree).

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BST()

```
BST::BST ()  [inline]
```

3.1.2.2 ∼BST()

```
BST::∼BST ()  [inline]
```

3.1.3 Member Function Documentation

3.1.3.1 clearTree()

```
void BST::clearTree ()
```

Czyści całe drzewo.

Usuwa całe drzewo BST.

Funkcja wywołuje prywatną metodę `clear(node*)`, która rekurencyjnie usuwa wszystkie węzły drzewa.

3.1.3.2 getInorder()

```
vector< int > BST::getInorder () const
```

Zwraca elementy w kolejności inorder.

Zwraca wektor kluczy w porządku inorder (L–K–P).

Metoda wywołuje prywatną funkcję rekurencyjną inorder(), która wypełnia przekazany wektor wartościami odczytanymi w kolejności inorder.

Returns

vector<int> Wektor zawierający klucze drzewa w porządku inorder.

3.1.3.3 getPathTo()

```
vector< int > BST::getPathTo (
            int key) const
```

Zwraca ścieżkę od korzenia do danego klucza.

Zwraca ścieżkę od korzenia do podanego klucza w drzewie.

Funkcja wykorzystuje prywatną funkcję findPath(), która rekurencyjnie przeszukuje drzewo w poszukiwaniu klucza i wypełnia wektor ścieżki.

Parameters

key	Klucz, do którego chcemy znaleźć ścieżkę.

Returns

vector<int> Wektor zawierający kolejność kluczy od korzenia do poszukiwanego klucza. Zwraca pusty wektor, jeśli klucz nie istnieje w drzewie.

3.1.3.4 getPostorder()

```
vector< int > BST::getPostorder () const
```

Zwraca elementy w kolejności postorder.

Zwraca elementy drzewa w kolejności postorder (Left -> Right -> Root).

Funkcja wykorzystuje prywatną funkcję postorder(), która rekurencyjnie przeszukuje drzewo i wypełnia wektor elementami w kolejności postorder.

Returns

vector<int> Wektor elementów drzewa w kolejności postorder.

3.1.3.5 getPreorder()

```
vector< int > BST::getPreorder () const
```

Zwraca elementy w kolejności preorder.

Zwraca elementy drzewa w kolejności preorder (Root -> Left -> Right).

Funkcja wykorzystuje prywatną funkcję preorder(), która rekurencyjnie przeszukuje drzewo i wypełnia wektor elementami w kolejności preorder.

Returns

vector<int> Wektor elementów drzewa w kolejności preorder.

3.1.3.6 getRoot()

```
BST::Node * BST::getRoot () const  [inline]
```

Zwraca wskaźnik do korzenia.

3.1.3.7 insertKey()

```
void BST::insertKey (
            int key)
```

Wstawia nowy klucz do drzewa.

Wstawia nowy klucz do drzewa BST.

Funkcja wywołuje prywatną funkcję insert(), która rekurencyjnie znajduje odpowiednie miejsce w drzewie i wstawia nowy węzeł.

Parameters

key	Klucz do wstawienia.

3.1.3.8 printTreeGraphic()

```
void BST::printTreeGraphic (
            int indentStep = 4) const
```

Wyświetlenie drzewa w konsoli.

Wyświetla drzewo BST w formie graficznej w konsoli.

Funkcja wywołuje rekurencyjną metodę printTreeGraphic2 dla korzenia drzewa. Jeśli drzewo jest puste, wyświetla odpowiedni komunikat.

Parameters

indentStep	Ilość spacji używana do wcięć dla kolejnych poziomów drzewa.

3.1.3.9 printTreeGraphic2()

```
void BST::printTreeGraphic2 (
            Node * node,
            int indent = 0,
            int indentStep = 4) const
```

Rekursywne wyświetlenie drzewa w konsoli.

Rekurencyjnie wyświetla drzewo BST w formie graficznej w konsoli.

Funkcja wywołuje się rekurencyjnie dla prawego i lewego dziecka węzła, stosując wcięcia w zależności od poziomu drzewa.

Parameters

node	Wskaźnik na aktualny węzeł drzewa.
indent	Aktualny poziom wcięcia dla tego węzła.
indentStep	Ilość spacji dodawanych przy przechodzeniu na niższy poziom drzewa.

3.1.3.10 printVectorTraversal()

```
void BST::printVectorTraversal (
            const vector< int > & v)  [static]
```

Wypisuje wektor kluczy na ekran.

Wyświetla zawartość wektora reprezentującego przegląd drzewa (preorder, inorder, postorder).

Funkcja służy do wygodnego drukowania kolejności węzłów drzewa w konsoli.

Parameters

v	Wektor kluczy drzewa do wyświetlenia.

3.1.3.11 removeKey()

```
void BST::removeKey (
            int key)
```

Usuwa klucz z drzewa.

Usuwa węzeł o podanym kluczu z drzewa BST.

Funkcja wywołuje prywatną funkcję remove() rozpoczynając od korzenia drzewa.

Parameters

key	Klucz węzła, który ma zostać usunięty.

3.1.4 Friends And Related Symbol Documentation

3.1.4.1 BSTFileHandler

```
friend class BSTFileHandler  [friend]
```

Klasa przyjaciół umożliwiająca dostęp do prywatnych elementów drzewa.

The documentation for this class was generated from the following files:

- bst.h
- clearTree.cpp
- getInorder.cpp
- getPathTo.cpp
- getPostorder.cpp
- getPreorder.cpp
- insertKey.cpp
- printTreeGraphic.cpp
- printTreeGraphic2.cpp
- printVectorTraversal.cpp
- removeKey.cpp

3.2 BSTFileHandler Class Reference

Klasa obsługująca zapis i odczyt drzewa BST do/z plików.

```
#include <bst_file_handler.h>
```

Public Member Functions

- BSTFileHandler (BST &tree)

  Konstruktor przyjmujący referencję do drzewa BST.
- void saveToBinary (const std::string &filename) const

  Zapis drzewa do pliku binarnego w kolejności preorder.
- void saveToBinary (const std::string &filename, int mode) const

  Zapis drzewa do pliku binarnego w wybranej kolejności.
- void loadFromBinary (const std::string &filename)

  Wczytanie drzewa z pliku binarnego (czyści drzewo przed wczytaniem).
- void loadFromText (const std::string &filename, bool append=true)

  Wczytanie drzewa z pliku tekstowego.
- void saveToText (const std::string &filename, int method) const

  Zapis drzewa do pliku tekstowego w wybranej kolejności.

3.2.1 Detailed Description

Klasa obsługująca zapis i odczyt drzewa BST do/z plików.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 BSTFileHandler()

```
BSTFileHandler::BSTFileHandler (
            BST & tree)
```

Konstruktor przyjmujący referencję do drzewa BST.

Konstruktor klasy BSTFileHandler.

Parameters

tree	Referencja do obiektu BST
tree	Referencja do drzewa BST, na którym będą wykonywane operacje.

3.2.3 Member Function Documentation

3.2.3.1 loadFromBinary()

```
void BSTFileHandler::loadFromBinary (
            const std::string & filename)
```

Wczytanie drzewa z pliku binarnego (czyści drzewo przed wczytaniem).

Wczytuje drzewo z pliku binarnego i wstawia elementy do drzewa.

Parameters

filename	Nazwa pliku
filename	Nazwa pliku binarnego.

3.2.3.2 loadFromText()

```
void BSTFileHandler::loadFromText (
            const std::string & filename,
            bool append = true)
```

Wczytanie drzewa z pliku tekstowego.

Wczytuje liczby z pliku tekstowego i dodaje je do drzewa.

Parameters

filename	Nazwa pliku
append	Jeśli true, dodaje do istniejącego drzewa; jeśli false, czyści drzewo
filename	Nazwa pliku tekstowego.
append	Czy dopisywać do istniejącego drzewa (true), czy wyczyścić je (false).

3.2.3.3 saveToBinary() [1/2]

```
void BSTFileHandler::saveToBinary (
            const std::string & filename) const
```

Zapis drzewa do pliku binarnego w kolejności preorder.

Parameters

filename	Nazwa pliku

3.2.3.4 saveToBinary() [2/2]

```
void BSTFileHandler::saveToBinary (
            const std::string & filename,
            int mode) const
```

Zapis drzewa do pliku binarnego w wybranej kolejności.

Zapisuje drzewo do pliku binarnego w wybranej metodzie zapisu.

Parameters

filename	Nazwa pliku

mode	1=preorder, 2=inorder, 3=postorder
filename	Nazwa pliku binarnego.
mode	Metoda zapisu (1 = preorder, 2 = inorder, 3 = postorder).

3.2.3.5 saveToText()

```
void BSTFileHandler::saveToText (
            const std::string & filename,
            int method) const
```

Zapis drzewa do pliku tekstowego w wybranej kolejności.

Zapisuje drzewo do pliku tekstowego w wybranej kolejności.

Parameters

filename	Nazwa pliku
method	1=preorder, 2=inorder, 3=postorder
filename	Nazwa pliku tekstowego.
method	Metoda zapisu (1 = preorder, 2 = inorder, 3 = postorder).

The documentation for this class was generated from the following files:

- bst_file_handler.h
- bst_file_handler.cpp

Chapter 4

File Documentation

4.1 bst.h File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <cstring>
#include <sstream>
#include <algorithm>
```

Classes

- class BST

  Klasa reprezentująca drzewo BST (Binary Search Tree).

4.2 bst.h

Go to the documentation of this file.

```
00001 #ifndef BST_H
00002 #define BST_H
00003
00004 #include <iostream>
00005 #include <memory>
00006 #include <stack>
00007 #include <iterator>
00008 #include <fstream>
00009 #include <iomanip>
00010 #include <vector>
00011 #include <cstring>
00012 #include <sstream>
00013 #include <algorithm>
00014
00015 using namespace std;
00016
00020 class BST {
00021     friend class BSTFileHandler;
00022
00023 private:
```

```
00027     struct Node {
00028         int key;
00029         Node* left;
00030         Node* right;
00035         Node(int k) : key(k), left(nullptr), right(nullptr) {}
00036     };
00037
00038     Node* root;
00039
00046     Node* insert(Node* node, int key) {
00047         if (!node) return new Node(key);
00048         if (key < node->key)
00049             node->left = insert(node->left, key);
00050         else if (key > node->key)
00051             node->right = insert(node->right, key);
00052         return node;
00053     }
00054
00061     Node* remove(Node* node, int key) {
00062         if (!node) return nullptr;
00063         if (key < node->key) node->left = remove(node->left, key);
00064         else if (key > node->key) node->right = remove(node->right, key);
00065         else {
00066             if (!node->left) {
00067                 Node* r = node->right;
00068                 delete node;
00069                 return r;
00070             } else if (!node->right) {
00071                 Node* l = node->left;
00072                 delete node;
00073                 return l;
00074             } else {
00075                 Node* succParent = node;
00076                 Node* succ = node->right;
00077                 while (succ->left) {
00078                     succParent = succ;
00079                     succ = succ->left;
00080                 }
00081                 node->key = succ->key;
00082                 node->right = remove(node->right, succ->key);
00083             }
00084         }
00085         return node;
00086     }
00087
00092     void clear(Node* node) {
00093         if (!node) return;
00094         clear(node->left);
00095         clear(node->right);
00096         delete node;
00097     }
00098
00104     void preorder(Node* node, vector<int>& out) const {
00105         if (!node) return;
00106         out.push_back(node->key);
00107         preorder(node->left, out);
00108         preorder(node->right, out);
00109     }
00110
00116     void inorder(Node* node, vector<int>& out) const {
00117         if (!node) return;
00118         inorder(node->left, out);
00119         out.push_back(node->key);
00120         inorder(node->right, out);
00121     }
00122
00128     void postorder(Node* node, vector<int>& out) const {
00129         if (!node) return;
00130         postorder(node->left, out);
00131         postorder(node->right, out);
00132         out.push_back(node->key);
00133     }
00134
00142     bool findPath(Node* node, int key, vector<int>& path) const {
00143         if (!node) return false;
00144         path.push_back(node->key);
00145         if (node->key == key) return true;
00146         if (key < node->key) {
00147             if (findPath(node->left, key, path)) return true;
00148         } else {
00149             if (findPath(node->right, key, path)) return true;
00150         }
00151         path.pop_back();
00152         return false;
00153     }
00154
00160     void serializePreorder(Node* node, ostream& fout) const {
```

```
00161         if (!node) {
00162             char marker = 0;
00163             fout.write(&marker, sizeof(marker));
00164             return;
00165         }
00166         char marker = 1;
00167         fout.write(&marker, sizeof(marker));
00168         fout.write(reinterpret_cast<const char*>(&(node->key)), sizeof(node->key));
00169         serializePreorder(node->left, fout);
00170         serializePreorder(node->right, fout);
00171     }
00172
00173 public:
00174     BST() : root(nullptr) {}
00175     ~BST() { clearTree(); }
00176
00178     void insertKey(int key);
00179
00181     void removeKey(int key);
00182
00184     void clearTree();
00185
00187     vector<int> getPathTo(int key) const;
00188
00190     vector<int> getPreorder() const;
00191
00193     vector<int> getInorder() const;
00194
00196     vector<int> getPostorder() const;
00197
00199     static void printVectorTraversal(const vector<int>& v);
00200
00202     void printTreeGraphic2(Node* node, int indent = 0, int indentStep = 4) const;
00203
00205     void printTreeGraphic(int indentStep = 4) const;
00206
00208     BST::Node* getRoot() const { return root; }
00209 };
00210
00211 #endif // BST_H
```

4.3 bst_file_handler.cpp File Reference

```
#include "bst_file_handler.h"
#include <fstream>
#include <iostream>
#include <sstream>
```

4.4 bst_file_handler.h File Reference

```
#include <string>
#include <vector>
#include "bst.h"
```

Classes

- class BSTFileHandler

  Klasa obsługująca zapis i odczyt drzewa BST do/z plików.

4.5 bst_file_handler.h

Go to the documentation of this file.

```
00001 #ifndef BST_FILE_HANDLER_H
00002 #define BST_FILE_HANDLER_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include "bst.h"
00007
00011 class BSTFileHandler {
00012 public:
00017     BSTFileHandler(BST& tree);
00018
00023     void saveToBinary(const std::string& filename) const;
00024
00030     void saveToBinary(const std::string& filename, int mode) const;
00031
00036     void loadFromBinary(const std::string& filename);
00037
00043     void loadFromText(const std::string& filename, bool append = true);
00044
00050     void saveToText(const std::string& filename, int method) const;
00051
00052 private:
00053     BST& treeRef;
00054
00060     void saveNode(std::ofstream& out, BST::Node* node) const;
00061
00066     void loadNode(std::ifstream& in);
00067 };
00068
00069 #endif
```

4.6 clearTree.cpp File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

4.7 getInorder.cpp File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

4.8 getPathTo.cpp File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

4.9 getPostorder.cpp File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

4.10 getPreorder.cpp File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

4.11 insertKey.cpp File Reference

```
#include <iostream>
#include <memory>
```

```
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

4.12 main.cpp File Reference

```
#include <iostream>
#include <cmath>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
#include "bst_file_handler.h"
```

Functions

- int main ()

  Główny punkt wejścia programu.

4.12.1 Function Documentation

4.12.1.1 main()

```
int main ()
```

Główny punkt wejścia programu.

Program umożliwia interaktywną obsługę drzewa BST: dodawanie/usuwanie elementów, wyszukiwanie ścieżki, wyświetlanie drzewa oraz zapis/odczyt do/z plików binarnych i tekstowych.

Returns

int Zwraca 0 po normalnym zakończeniu programu.

< Dodawanie elementu

< Usuwanie elementu

< Usuwanie całego drzewa

< Wyszukiwanie ścieżki do elementu

< Wyświetlanie drzewa w wybranej kolejności

< Wyświetlanie drzewa graficznie

< Zapis do pliku binarnego

< Wczytanie z pliku binarnego

< Wczytanie z pliku tekstowego

< Zapis do pliku tekstowego

< Zakończenie programu

4.13 printTreeGraphic.cpp File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

4.14 printTreeGraphic2.cpp File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

4.15 printVectorTraversal.cpp File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

4.16 removeKey.cpp File Reference

```
#include <iostream>
#include <memory>
#include <stack>
#include <iterator>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include "bst.h"
```

Index