Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Czerwono-Czarni

autor	Aleksander Augustyniak
prowadzący	mgr inż. Marek Kokot
rok akademicki	2019/2020
kierunek	informatyka
rodzaj studiów	SSI
semestr	1
termin laboratorium	piątek, 11:45 – 13:15
sekcja	22
termin oddania sprawozdania	2020-01-27

1 Treść zadania

Napisać program sortujący liczby rzeczywiste w pewnym zbiorze. Liczby podawane są w dość specyficzny sposób. Liczba może być dodana lub usunięta ze zbioru. Dodanie liczb jest realizowane przez komendę `add`, po której może wystąpić jedna lub więcej liczb (rozdzielonych białymi znakami). Komenda `remove` usuwa podaną po niej liczbę (lub liczby rozdzielone białymi znakami) ze zbioru. Komenda `print` powoduje wypisane liczb zawartych w zbiorze w porządku rosnącym. Po komendzie tej można podać nazwę pliku, wtedy wartości zostaną zapisane do tegoż pliku zamiast na standardowe wyjście. Komenda `graph` wypisuje drzewo w postaci graficznej – głębsze poziomy drzewa są wypisywane z coraz większym wcięciem. Dodatkowo wartości w węzłach czarnych są wypisywane w nawiasach kwadratowych, np. `[13]`, w węzłach czerwonych – w nawiasach okrągłych, np. `(13)`. Podobnie jak w przypadku komendy print po komendzie graph można podać nazwę pliku do zapisu. Jeżeli komendy `print` i `graph` są użyte do zapisu do pliku, możliwe jest poprzedzenie nazwy pliku znakiem `+`. Wtedy plik nie zostanie nadpisany, ale nowa treść zostanie dopisana na końcu pliku, nie niszcząc dotychczasowej jego zawartości. Znak `%` rozpoczyna komentarz do końca linii. Każda komenda jest zapisana w osobnej linii. Jeżeli zostanie podana niepoprawna komenda, program ignoruje ją.

Przykładowy plik wejściowy:

```
% przykladowy plik wejsciowy
add -3.14 43.9 4
graph % wypisane z wcieciami i z oznaczeniami kolorow wezlow
remove 4
print % wypisane na ekran
add 3.45 -0.32
print test-1.txt % wypisanie do pliku

add 490 32.3

print % na ekran
print + test-1.txt % dopisanie do pliku
```

Po komendzie `graph` zostanie wypisane drzewo:

```
   (-3.14)
[4]
   (42.9)
```

Program uruchamiany jest z linii poleceń z wykorzystaniem jednego prze-

łącznika:

`-i` plik wejściowy

Do przechowywania liczb należy wykorzystać drzewa czerwono-czarne. Jest to warunek sine qua non.

2 Analiza zadania

Zadanie skupia się wokół problemu sortowania liczb rzeczywistych, zapisanych w pliku o podanej przez użytkownika nazwie. Struktura danych powinna wykorzystywać dynamiczne alokowanie pamięci, w celu szybszego wykonywania operacji na danych. Jednym z warunków przystąpienia do napisania programu jest umiejętność korzystania z pamięci w ten sposób, aby nie powodować wycieku danych.

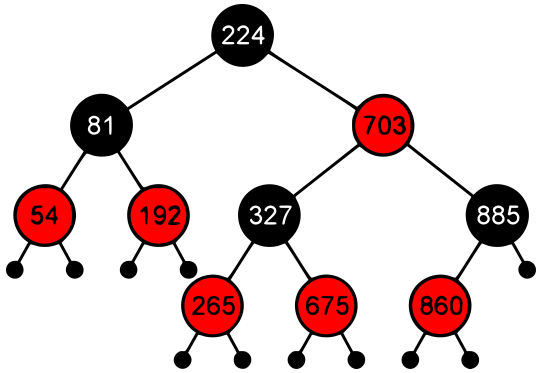
Do zrealizowania zadania potrzebne jest zarezerwowanie jednego bitu pamięci dla każdego z węzłów zawartych w strukturze danych, w celu opisania koloru danego węzła.

2.1 Struktury danych

Program wykorzystuje drzewo czerwono–czarne do przechowywania wartości. Wartość jest przypisywana węzłowi o odpowiednim kolorze. Węzeł może mieć jednego, dwóch potomków lub ich nie mieć. W lewym poddrzewie wybranego węzła występują wartości mniejsze od wartości przechowywanej przez *węzeł–rodzic*, zaś wartości niemniejsze są umieszczone w prawym poddrzewie węzła. Rysunek 1 przedstawia wizualizację drzewa czerwono–czarnego.

Podstawowe zasady panujące w drzewie czerwono–czarnym: [1]

1. Każdy z węzłów jest czerwony lub czarny.
2. Korzeń drzewa jest czarny.
3. Wszystkie liście (NIL) są czarne.
4. Jeżeli węzeł jest czerwony, to oba jego potomki są czarne.
5. Każda ścieżka od wybranego węzła do liścia zawiera tę samą ilość czarnych węzłów.



Rysunek 1: Wizualizacja drzewa czerwono–czarnego przechowującego liczby rzeczywiste. Węzły reprezentują wartości wprowadzone w następującej kolejności: 327, 54, 224, 81, 703, 885, 192, 860, 265, 675.

Powyższe zasady – opisujące wzajemny stosunek węzłów drzewa – pomagają zapanować nad wartościami przechowywanymi w węzłach oraz równoważyć jego wysokość. Pozwala to na skrócenie czasu poszukiwania, usuwania oraz dodawania wartości do struktury.

2.2 Algorytmy

Dodawanie wartości do drzewa czerwono–czarnego jest realizowane podobnie, jak w drzewie poszukiwań binarnych, lecz dodatkowo program wykonuje lokalne operacje na węzłach, takie jak: rotacje w lewo/prawo oraz zmiana kolorów węzłów, by przywrócić właściwości drzewa czerwono–czarnego.

Gwarantowana złożoność obliczeniowa dla operacji dodawania, wyszukiwania, usuwania węzłów wynosi średnio, jak i w najgorszym przypadku $O(\log n)$, gdzie n oznacza ilość wszystkich elementów znajdujących się w strukturze danych. [3] Złożoność obliczeniowa algorytmów rotacji w lewo, w prawo oraz zmiany kolorów węzłów również wynosi $O(\log n)$. [2]

By zrealizować operacje wypisywania i usuwania wartości węzłów, program rekurencyjnie przechodzi przez wszystkie węzły drzewa.

3 Specyfikacja zewnętrzna

Aby uruchomić program, należy wprowadzić do linii poleceń, w odpowiedniej kolejności:

```
nazwa_programu -i plik_wejściowy
```

Jeżeli program nie został prawidłowo uruchomiony, pojawi się stosowny ko-

munikat z ewentualną instrukcją prawidłowego uruchomienia programu.

4 Specyfikacja wewnętrzna

Program realizuje pojedyncze komendy usuwając z wczytanego łańcucha znakowego niepotrzebne znaki, np. znaki białe, znak rozpoczęcia komentrza (i znaki znajdujące się bezpośrednio za nim, aż do końca linii).

W trakcie zczytywania pojedynczych wartości program decyduje o tym, czy wczytana wartość jest prawidłowa, tzn. czy nie zawiera znaków nie będących cyframi. Ponadto, jeżeli funkcja zczytująca dane dostanie informację o liczbie, wykraczającej poza zakres określony dla jej typu, liczba ta zostanie zignorowana, a program dalej będzie realizowany.

4.1 Ogólna struktura programu

Funkcja główna sprawdza, czy ilość wprowadzonych parametrów do linii poleceń jest większa od dwóch. Jeżeli nie – program kończy się – w przeciwnym wypadku zostaje uruchomiona funkcja ReadFromFile, przyjmująca jeden plik wejściowy. Funkcja ta zczytuje – linia po linii – każde polecenie i wywołuje odpowiednie podfunkcje: Add, Delete, Print, Graph, itd. – w zależności od znajdującej się w pliku wejściowym komendy.

Program – po wczytaniu wszystkich komend – dealokuje całą zajętą przez drzewo pamięć i zamyka się.

4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji znajduje się w załączniku.

5 Testowanie

Program został przetestowany na plikach:

- typowych, poprawnych – w których komendy następowały po sobie w różnym porządku;
- nietypowych i poprawnych – o znacznej ilości danych;
- niepoprawnych – wynikiem tego było ignorowanie źle wypisanych komend i liczb zawierających znaki inne niż zarezerwowane dla cyfr.

Przetestowane zostały również funkcje wykorzystujące rekurencyjne przechodzenie przez węzły.

W tabeli 1 przedstawiono wyniki testów sprawdzających zapotrzebowanie pamięci na określoną ilość wprowadzonych danych:

Program został również sprawdzony pod kątem wycieków pamięci.

6 Uzyskane wyniki

ilość liczb dodanych do struktury	pamięć przydzielona procesowi
10000	3 MB
20000	6 MB
30000	9 MB
50000	15 MB
75000	18.4 MB
100000	25.2 MB
250000	60.6 MB
500000	120 MB
8000000	1.9 GB

Tablica 1: Zestawienie przybliżonej ilości zapotrzebowania na pamięć, potrzebnej do dodania do struktury określonej ilości danych.

7 Wnioski

Program wykorzystujący drzewa czerwono–czarne jako strukturę danych nie jest programem łatwym do napisania. Szczególną trudność sprawiło mi obsłużenie sytuacji usuwania wartości z drzewa, które wymagało przywrócenia jego własności. 2.1

W napisaniu projektu pomogło mi dobre nastawienie, solidna, codzienna praca. Przed przystąpieniem do projektu pisałem wiele innych programów, które były zlecone do wykonania przez prowadzącego na laboratorium. Robiłem również dodatkowe zadania, pozwalające uzyskać odpowiednią sprawność w kodowaniu. Cały wysiłek, który przeznaczyłem przez ostatni semestr na programowanie, zaowocowało zdobytą przeze mnie wiedzą, z której będę mógł w odpowiednim momencie skorzystać.

Niestety, nie udało mi się przetestować programu na maszynach z innym systemem operacyjnym niż Windows 10.

Literatura

[1] Thomas Cormen, Charles Leiserson, Ronalds Rivest, Clifford Stein. *„13". Introduction to Algorithms (3rd ed.).* 2009.

[2] John Morris. „red–black trees". data structures and algorithms., 1998.

[3] James Paton. Red-black trees.

Dodatek
Szczegółowy opis typów i funkcji

Czerwono-Czarni

1.00

Wygenerowano przez Doxygen 1.8.16

1 Indeks klas 1

1.1 Lista klas . . . 1

2 Indeks plików 3

2.1 Lista plików . . . 3

3 Dokumentacja klas 5

3.1 Dokumentacja struktury TreeNode . . . 5

3.1.1 Opis szczegółowy . . . 5

3.1.2 Dokumentacja atrybutów składowych . . . 5

3.1.2.1 _Color . . . 6

3.1.2.2 left . . . 6

3.1.2.3 parent . . . 6

3.1.2.4 right . . . 6

3.1.2.5 value . . . 6

4 Dokumentacja plików 7

4.1 Dokumentacja pliku funkcje.cpp . . . 7

4.1.1 Dokumentacja funkcji . . . 8

4.1.1.1 Add() . . . 8

4.1.1.2 Add_Recovery() . . . 8

4.1.1.3 ChangeToColor() . . . 10

4.1.1.4 Deallocate() . . . 10

4.1.1.5 Delete() . . . 10

4.1.1.6 Delete_Recovery() . . . 11

4.1.1.7 FileGraph() . . . 11

4.1.1.8 FilePrint() . . . 11

4.1.1.9 Find() . . . 12

4.1.1.10 GetParent() . . . 12

4.1.1.11 GetSibling() . . . 12

4.1.1.12 Graph() . . . 13

4.1.1.13 GraphToFile() . . . 13

4.1.1.14 lookForSign() . . . 13

4.1.1.15 Print() . . . 14

4.1.1.16 PrintToFile() . . . 14

4.1.1.17 ReadFromFile() . . . 14

4.1.1.18 rotate_left() . . . 14

4.1.1.19 rotate_right() . . . 15

4.1.1.20 StartCase() . . . 15

4.1.1.21 Successor() . . . 15

4.2 Dokumentacja pliku funkcje.h . . . 16

4.2.1 Dokumentacja funkcji . . . 17

4.2.1.1 Add() . . . 17

4.2.1.2 Add_Recovery() . . . 17
4.2.1.3 ChangeToColor() . . . 18
4.2.1.4 Deallocate() . . . 18
4.2.1.5 Delete() . . . 18
4.2.1.6 Delete_Recovery() . . . 19
4.2.1.7 FileGraph() . . . 19
4.2.1.8 FilePrint() . . . 19
4.2.1.9 Find() . . . 20
4.2.1.10 GetParent() . . . 20
4.2.1.11 GetSibling() . . . 20
4.2.1.12 Graph() . . . 21
4.2.1.13 GraphToFile() . . . 21
4.2.1.14 lookForSign() . . . 21
4.2.1.15 Print() . . . 22
4.2.1.16 PrintToFile() . . . 22
4.2.1.17 ReadFromFile() . . . 22
4.2.1.18 rotate_left() . . . 23
4.2.1.19 rotate_right() . . . 23
4.2.1.20 StartCase() . . . 23
4.2.1.21 Successor() . . . 24
4.3 Dokumentacja pliku main.cpp . . . 24
4.3.1 Dokumentacja funkcji . . . 24
4.3.1.1 main() . . . 24
4.4 Dokumentacja pliku struktury.h . . . 25
4.4.1 Dokumentacja definicji typów . . . 25
4.4.1.1 T . . . 25
4.4.2 Dokumentacja typów wyliczanych . . . 25
4.4.2.1 Color . . . 25

Indeks **27**

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

TreeNode
Struktura węzła drzewa czerwono-czarnego.
5

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

funkcje.cpp 7
funkcje.h 16
main.cpp 24
struktury.h 25

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury TreeNode

Struktura węzła drzewa czerwono-czarnego.

`#include <struktury.h>`

Atrybuty publiczne

- Color _Color {}
  Kolor typu wyliczeniowego, określający kolor węzła.
- T value
  Wartość typu domyślnego T, przechowywana w węźle.
- TreeNode * left {}
  Wskaźnik na węzeł lewego poddrzewa.
- TreeNode * right {}
  Wskaźnik na węzeł prawego poddrzewa.
- TreeNode * parent {}
  Wskaźnik na węzeł-rodzic poddrzewa.

3.1.1 Opis szczegółowy

Struktura węzła drzewa czerwono-czarnego.

3.1.2 Dokumentacja atrybutów składowych

3.1.2.1 _Color

```
Color TreeNode::_Color {}
```

Kolor typu wyliczeniowego, określający kolor węzła.

3.1.2.2 left

```
TreeNode* TreeNode::left {}
```

Wskaźnik na węzeł lewego poddrzewa.

3.1.2.3 parent

```
TreeNode* TreeNode::parent {}
```

Wskaźnik na węzeł-rodzic poddrzewa.

3.1.2.4 right

```
TreeNode* TreeNode::right {}
```

Wskaźnik na węzeł prawego poddrzewa.

3.1.2.5 value

```
T TreeNode::value
```

Wartość typu domyślnego T, przechowywana w węźle.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku funkcje.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include "struktury.h"
```

Funkcje

- TreeNode ∗ GetParent (TreeNode ∗node)

  Funkcja znajdująca i zwracająca rodzica podanego węzła.
- TreeNode ∗ GetSibling (TreeNode ∗node)

  Funkcja znajdująca i zwracająca rodzeństwo podanego węzła.
- void ChangeToColor (TreeNode ∗node, Color color)

  Funkcja zmieniająca kolor węzła na podany.
- void rotate_left (TreeNode ∗&node)

  Lokalna zmiana struktury poddrzewa (w lewo).
- void rotate_right (TreeNode ∗&node)

  Lokalna zmiana struktury poddrzewa (w lewo).
- void Add_Recovery (TreeNode ∗z)

  Funkcja przywracająca własności drzewa Czerwono-Czarnego po wykonaniu operacji dodawania nowego węzła do drzewa.
- void Add (TreeNode ∗&root, const T &value)

  Funkcja dodająca do drzewa węzeł o podanej wartości.
- TreeNode ∗ Find (TreeNode ∗root, const T &value)

  Funkcja szukająca w drzewie węzła o podanej wartości. Jeżeli węzeł zostanie odnaleziony funkcja zwraca wskaźnik na niego - w przeciwnym wypadku zwracany jest nullptr.
- TreeNode ∗ Successor (TreeNode ∗node)

  Funkcja odnajdująca następcę danego węzła. Jeżeli węzeł zostanie odnaleziony, funkcja zwraca wskaźnik na niego, w przeciwnym wypadku zwracany jest nullptr.
- void StartCase (TreeNode ∗node, bool &isCase2)

  Funkcja obejmująca przestawienie odpowiednich wskaźników i dealokowanie odpowiedniej pamięci - pamięci węzła y.
- void Delete_Recovery (TreeNode ∗&node)

Funkcja przywracająca własności drzewa czerwono-czarnego z widoku usuniętego wcześniej węzła-następcy węzła przeznaczonego do usunięcia.

- void Delete (TreeNode ∗&root, const T &value)

  Funkcja odnajduje węzeł o podanej wartości, usuwa go i dealokuje zaalokowaną wcześniej pamięć.
- void Print (TreeNode ∗root)

  Funkcja przyjmująca jako parametr korzeń drzewa. Wypisuje wartości wszystkich węzłów drzewa w kolejności in-order.
- void Graph (TreeNode ∗root, int indent=0)

  Funkcja wypisuje wartości wszystkich węzłów drzewa w kolejności rosnącej z uwzględnieniem wysokości drzewa i kolorów poszczególnych węzłów.
- void Deallocate (TreeNode ∗&root)

  Funkcja całkowicie usuwa drzewo i dealokuje zajętą przezeń pamięć.
- void PrintToFile (TreeNode ∗root, ofstream &file)
- void GraphToFile (TreeNode ∗root, ofstream &file, int indent=0)
- size_t lookForSign (const std::string &line, char searched)

  Funkcja odnajdująca pozycję podanego znaku w łańcuchu znakowym. Używana jest w funkcji ReadFromFile, by usunąć z pliku wszystkie komentarze, puste linie ora linie, które zawierają niepoprawnie wprowadzone komendy.
- void FilePrint (TreeNode ∗&root, const std::string &file_name, const bool &toOverwrite)

  Funkcja wypisująca do pliku o podanej nazwie wartości wszystich węzłów drzewa w porządku rosnącym.
- void FileGraph (TreeNode ∗&root, const std::string &file_name, const bool &toOverwrite)

  Funkcja wypisująca do pliku o podanej nazwie wizualizację wszystkich węzłów znajdujących się w drzewie, z uwzględnieniem ich koloru, np: [33] - węzeł czarny o wartości 33, (13) - węzeł czerwony o wartości 13.
- void ReadFromFile (const std::string &file_name)

  Funkcja odczytująca plik "linia po linii" i wykonująca odpowiednie polecenia.

4.1.1 Dokumentacja funkcji

4.1.1.1 Add()

```
void Add (
            TreeNode *& root,
            const T & value )
```

Funkcja dodająca do drzewa węzeł o podanej wartości.

Parametry

root	wskaźnik na korzeń drzewa.
T	wartość.

4.1.1.2 Add_Recovery()

```
void Add_Recovery (
            TreeNode * z )
```

Funkcja przywracająca własności drzewa Czerwono-Czarnego po wykonaniu operacji dodawania nowego węzła do drzewa.

Parametry

z	wskaźnik na węzeł drzewa.

4.1.1.3 ChangeToColor()

```
void ChangeToColor (
            TreeNode * node,
            Color color )
```

Funkcja zmieniająca kolor węzła na podany.

Parametry

node	wskaźnik na węzeł drzewa, którego kolor będzie zmieniony.
color	kolor, na jaki węzeł powinien być zmieniony (o ile węzeł istnieje).

4.1.1.4 Deallocate()

```
void Deallocate (
            TreeNode *& root )
```

Funkcja całkowicie usuwa drzewo i dealokuje zajętą przezeń pamięć.

Parametry

root	wskaźnik na korzeń drzewa.

4.1.1.5 Delete()

```
void Delete (
            TreeNode *& root,
            const T & value )
```

Funkcja odnajduje węzeł o podanej wartości, usuwa go i dealokuje zaalokowaną wcześniej pamięć.

Parametry

root	wskaźnik na korzeń drzewa.
value	wartość węzła do usunięcia.

4.1.1.6 Delete_Recovery()

```
void Delete_Recovery (
            TreeNode *& node )
```

Funkcja przywracająca własności drzewa czerwono-czarnego z widoku usuniętego wcześniej węzła-następcy węzła przeznaczonego do usunięcia.

Parametry

node	węzeł drzewa czerwono-czarnego.

4.1.1.7 FileGraph()

```
void FileGraph (
            TreeNode *& root,
            const std::string & file_name,
            const bool & toOverwrite )
```

Funkcja wypisująca do pliku o podanej nazwie wizualizację wszystkich węzłów znajdujących się w drzewie, z uwzględnieniem ich koloru, np: [33] - węzeł czarny o wartości 33, (13) - węzeł czerwony o wartości 13.

Parametry

root	wskaźnik na korzeń drzewa.
file_name	pełna nazwa pliku, do którego mają być wypisywane wszystkie wartości węzłów drzewa.
toOverwrite	parametr określający czy wartości w pliku mają być nadpisane.

4.1.1.8 FilePrint()

```
void FilePrint (
            TreeNode *& root,
            const std::string & file_name,
            const bool & toOverwrite )
```

Funkcja wypisująca do pliku o podanej nazwie wartości wszystich węzłów drzewa w porządku rosnącym.

Parametry

root	wskaźnik na korzeń drzewa.
file_name	pełna nazwa pliku, do którego mają być wypisywane wszystkie wartości węzłów drzewa.
toOverwrite	parametr określający czy wartości w pliku mają być nadpisane.

4.1.1.9 Find()

```
TreeNode* Find (
            TreeNode * root,
            const T & value )
```

Funkcja szukająca w drzewie węzła o podanej wartości. Jeżeli węzeł zostanie odnaleziony funkcja zwraca wskaźnik na niego - w przeciwnym wypadku zwracany jest nullptr.

Parametry

root	wskaźnik na korzeń drzewa.
value	wartosc.

Zwraca

węzeł o podanej wartości lub nullptr.

4.1.1.10 GetParent()

```
TreeNode* GetParent (
            TreeNode * node )
```

Funkcja znajdująca i zwracająca rodzica podanego węzła.

Parametry

node	wskaźnik na odpowiedni węzeł drzewa.

Zwraca

rodzic podanego węzła lub nullptr.

4.1.1.11 GetSibling()

```
TreeNode* GetSibling (
            TreeNode * node )
```

Funkcja znajdująca i zwracająca rodzeństwo podanego węzła.

Parametry

node	wskaźnik na odpowiedni węzeł drzewa.

Zwraca

rodzeństwo węzła lub nullptr.

4.1.1.12 Graph()

```
void Graph (
            TreeNode * root,
            int indent = 0 )
```

Funkcja wypisuje wartości wszystkich węzłów drzewa w kolejności rosnącej z uwzględnieniem wysokości drzewa i kolorów poszczególnych węzłów.

Parametry

root	wskaźnik na korzeń drzewa.
indent	wcięcie - wyrażane w ilości znaków białych.

4.1.1.13 GraphToFile()

```
void GraphToFile (
            TreeNode * root,
            ofstream & file,
            int indent = 0 )
```

4.1.1.14 lookForSign()

```
size_t lookForSign (
            const std::string & line,
            char searched )
```

Funkcja odnajdująca pozycję podanego znaku w łańcuchu znakowym. Używana jest w funkcji ReadFromFile, by usunąć z pliku wszystkie komentarze, puste linie ora linie, które zawierają niepoprawnie wprowadzone komendy.

Parametry

line	łańcuch znakowy.
searched	znak, którego pozycja ma zostać odnaleziona.

Zwraca

pozycję odnalezionego znaku (jeżeli nie wystąpił - zwracana jest długość podanego łańcucha znakowego).

4.1.1.15 Print()

```
void Print (
            TreeNode * root )
```

Funkcja przyjmująca jako parametr korzeń drzewa. Wypisuje wartości wszystkich węzłów drzewa w kolejności in-order.

Parametry

root	wskaźnik na korzeń drzewa.

4.1.1.16 PrintToFile()

```
void PrintToFile (
            TreeNode * root,
            ofstream & file )
```

4.1.1.17 ReadFromFile()

```
void ReadFromFile (
            const std::string & file_name )
```

Funkcja odczytująca plik "linia po linii" i wykonująca odpowiednie polecenia.

Parametry

file_name	pełna nazwa pliku.

4.1.1.18 rotate_left()

```
void rotate_left (
            TreeNode *& node )
```

Lokalna zmiana struktury poddrzewa (w lewo).

Parametry

node	wskaźnik na węzeł drzewa.

Ostrzeżenie

prawe poddrzewo węzła musi istnieć!

4.1.1.19 rotate_right()

```
void rotate_right (
            TreeNode *& node )
```

Lokalna zmiana struktury poddrzewa (w lewo).

Parametry

node	wskaźnik na węzeł drzewa.

Ostrzeżenie

lewe poddrzewo węzła musi istnieć!

4.1.1.20 StartCase()

```
void StartCase (
            TreeNode * node,
            bool & isCase2 )
```

Funkcja obejmująca przestawienie odpowiednich wskaźników i dealokowanie odpowiedniej pamięci - pamięci węzła y.

Parametry

node	wskaźnik na węzeł drzewa.
isCase2	sprawdzenie, czy istnieje potrzeba dalszego przywracania własności drzewa czerwono-czarnego.

4.1.1.21 Successor()

```
TreeNode* Successor (
            TreeNode * node )
```

Funkcja odnajdująca następcę danego węzła. Jeżeli węzeł zostanie odnaleziony, funkcja zwraca wskaźnik na niego, w przeciwnym wypadku zwracany jest nullptr.

Parametry

node	wskaźnik na węzeł drzewa.

Zwraca

następca węzła.

4.2 Dokumentacja pliku funkcje.h

```
#include <string>
#include "struktury.h"
```

Funkcje

- void Add (TreeNode ∗&root, const T &value)

  Funkcja dodająca do drzewa węzeł o podanej wartości.
- void Add_Recovery (TreeNode ∗z)

  Funkcja przywracająca własności drzewa Czerwono-Czarnego po wykonaniu operacji dodawania nowego węzła do drzewa.
- void rotate_left (TreeNode ∗&node)

  Lokalna zmiana struktury poddrzewa (w lewo).
- void rotate_right (TreeNode ∗&node)

  Lokalna zmiana struktury poddrzewa (w lewo).
- TreeNode ∗ Find (TreeNode ∗root, const T &value)

  Funkcja szukająca w drzewie węzła o podanej wartości. Jeżeli węzeł zostanie odnaleziony funkcja zwraca wskaźnik na niego - w przeciwnym wypadku zwracany jest nullptr.
- void StartCase (TreeNode ∗node, bool &isCase2)

  Funkcja obejmująca przestawienie odpowiednich wskaźników i dealokowanie odpowiedniej pamięci - pamięci węzła y.
- TreeNode ∗ Successor (TreeNode ∗node)

  Funkcja odnajdująca następcę danego węzła. Jeżeli węzeł zostanie odnaleziony, funkcja zwraca wskaźnik na niego, w przeciwnym wypadku zwracany jest nullptr.
- void Delete (TreeNode ∗&root, const T &value)

  Funkcja odnajduje węzeł o podanej wartości, usuwa go i dealokuje zaalokowaną wcześniej pamięć.
- TreeNode ∗ GetParent (TreeNode ∗node)

  Funkcja znajdująca i zwracająca rodzica podanego węzła.
- TreeNode ∗ GetSibling (TreeNode ∗node)

  Funkcja znajdująca i zwracająca rodzeństwo podanego węzła.
- void ChangeToColor (TreeNode ∗node, Color color)

  Funkcja zmieniająca kolor węzła na podany.
- void Deallocate (TreeNode ∗&root)

  Funkcja całkowicie usuwa drzewo i dealokuje zajętą przezeń pamięć.
- void Delete_Recovery (TreeNode ∗&node)

  Funkcja przywracająca własności drzewa czerwono-czarnego z widoku usuniętego wcześniej węzła-następcy węzła przeznaczonego do usunięcia.

- void Print (TreeNode ∗root)

  Funkcja przyjmująca jako parametr korzeń drzewa. Wypisuje wartości wszystkich węzłów drzewa w kolejności in-order.
- void Graph (TreeNode ∗root, int indent=0)

  Funkcja wypisuje wartości wszystkich węzłów drzewa w kolejności rosnącej z uwzględnieniem wysokości drzewa i kolorów poszczególnych węzłów.
- void ReadFromFile (const std::string &file_name)

  Funkcja odczytująca plik "linia po linii" i wykonująca odpowiednie polecenia.
- void FilePrint (TreeNode ∗&root, const std::string &file_name, const bool &toOverwrite)

  Funkcja wypisująca do pliku o podanej nazwie wartości wszystich węzłów drzewa w porządku rosnącym.
- void FileGraph (TreeNode ∗&root, const std::string &file_name, const bool &toOverwrite)

  Funkcja wypisująca do pliku o podanej nazwie wizualizację wszystkich węzłów znajdujących się w drzewie, z uwzględnieniem ich koloru, np: [33] - węzeł czarny o wartości 33, (13) - węzeł czerwony o wartości 13.
- void PrintToFile (TreeNode ∗root, std::ofstream &file)

  Główny algorytm służący wypisywaniu do pliku wszystkich wartości węzłów drzewa w kolejności rosnącej.
- void GraphToFile (TreeNode ∗root, std::ofstream &file, int indent=0)

  Główny algorytm służący wypisywaniu do pliku wizualizacji pozycji wszystkich węzłów w drzewie czerwono-czarnym.
- size_t lookForSign (const std::string &line, char searched)

  Funkcja odnajdująca pozycję podanego znaku w łańcuchu znakowym. Używana jest w funkcji ReadFromFile, by usunąć z pliku wszystkie komentarze, puste linie ora linie, które zawierają niepoprawnie wprowadzone komendy.

4.2.1 Dokumentacja funkcji

4.2.1.1 Add()

```
void Add (
            TreeNode *& root,
            const T & value )
```

Funkcja dodająca do drzewa węzeł o podanej wartości.

Parametry

root	wskaźnik na korzeń drzewa.
T	wartość.

4.2.1.2 Add_Recovery()

```
void Add_Recovery (
            TreeNode * z )
```

Funkcja przywracająca własności drzewa Czerwono-Czarnego po wykonaniu operacji dodawania nowego węzła do drzewa.

Parametry

z	wskaźnik na węzeł drzewa.

4.2.1.3 ChangeToColor()

```
void ChangeToColor (
            TreeNode * node,
            Color color )
```

Funkcja zmieniająca kolor węzła na podany.

Parametry

node	wskaźnik na węzeł drzewa, którego kolor będzie zmieniony.
color	kolor, na jaki węzeł powinien być zmieniony (o ile węzeł istnieje).

4.2.1.4 Deallocate()

```
void Deallocate (
            TreeNode *& root )
```

Funkcja całkowicie usuwa drzewo i dealokuje zajętą przezeń pamięć.

Parametry

root	wskaźnik na korzeń drzewa.

4.2.1.5 Delete()

```
void Delete (
            TreeNode *& root,
            const T & value )
```

Funkcja odnajduje węzeł o podanej wartości, usuwa go i dealokuje zaalokowaną wcześniej pamięć.

Parametry

root	wskaźnik na korzeń drzewa.
value	wartość węzła do usunięcia.

4.2.1.6 Delete_Recovery()

```
void Delete_Recovery (
            TreeNode *& node )
```

Funkcja przywracająca własności drzewa czerwono-czarnego z widoku usuniętego wcześniej węzła-następcy węzła przeznaczonego do usunięcia.

Parametry

node	węzeł drzewa czerwono-czarnego.

4.2.1.7 FileGraph()

```
void FileGraph (
            TreeNode *& root,
            const std::string & file_name,
            const bool & toOverwrite )
```

Funkcja wypisująca do pliku o podanej nazwie wizualizację wszystkich węzłów znajdujących się w drzewie, z uwzględnieniem ich koloru, np: [33] - węzeł czarny o wartości 33, (13) - węzeł czerwony o wartości 13.

Parametry

root	wskaźnik na korzeń drzewa.
file_name	pełna nazwa pliku, do którego mają być wypisywane wszystkie wartości węzłów drzewa.
toOverwrite	parametr określający czy wartości w pliku mają być nadpisane.

4.2.1.8 FilePrint()

```
void FilePrint (
            TreeNode *& root,
            const std::string & file_name,
            const bool & toOverwrite )
```

Funkcja wypisująca do pliku o podanej nazwie wartości wszystich węzłów drzewa w porządku rosnącym.

Parametry

root	wskaźnik na korzeń drzewa.
file_name	pełna nazwa pliku, do którego mają być wypisywane wszystkie wartości węzłów drzewa.
toOverwrite	parametr określający czy wartości w pliku mają być nadpisane.

4.2.1.9 Find()

```
TreeNode* Find (
            TreeNode * root,
            const T & value )
```

Funkcja szukająca w drzewie węzła o podanej wartości. Jeżeli węzeł zostanie odnaleziony funkcja zwraca wskaźnik na niego - w przeciwnym wypadku zwracany jest nullptr.

Parametry

root	wskaźnik na korzeń drzewa.
value	wartosc.

Zwraca

węzeł o podanej wartości lub nullptr.

4.2.1.10 GetParent()

```
TreeNode* GetParent (
            TreeNode * node )
```

Funkcja znajdująca i zwracająca rodzica podanego węzła.

Parametry

node	wskaźnik na odpowiedni węzeł drzewa.

Zwraca

rodzic podanego węzła lub nullptr.

4.2.1.11 GetSibling()

```
TreeNode* GetSibling (
            TreeNode * node )
```

Funkcja znajdująca i zwracająca rodzeństwo podanego węzła.

Parametry

node	wskaźnik na odpowiedni węzeł drzewa.

Zwraca

rodzeństwo węzła lub nullptr.

4.2.1.12 Graph()

```
void Graph (
            TreeNode * root,
            int indent = 0 )
```

Funkcja wypisuje wartości wszystkich węzłów drzewa w kolejności rosnącej z uwzględnieniem wysokości drzewa i kolorów poszczególnych węzłów.

Parametry

root	wskaźnik na korzeń drzewa.
indent	wcięcie - wyrażane w ilości znaków białych.

4.2.1.13 GraphToFile()

```
void GraphToFile (
            TreeNode * root,
            std::ofstream & file,
            int indent = 0 )
```

Główny algorytm służący wypisywaniu do pliku wizualizacji pozycji wszystkich węzłów w drzewie czerwono-czarnym.

Parametry

root	wskaźnik na korzeń drzewa.
file	strumień plikowy.
indent	wcięcie - wyrażane w ilości znaków białych.

4.2.1.14 lookForSign()

```
size_t lookForSign (
            const std::string & line,
            char searched )
```

Funkcja odnajdująca pozycję podanego znaku w łańcuchu znakowym. Używana jest w funkcji ReadFromFile, by usunąć z pliku wszystkie komentarze, puste linie ora linie, które zawierają niepoprawnie wprowadzone komendy.

Parametry

line	łańcuch znakowy.
searched	znak, którego pozycja ma zostać odnaleziona.

Zwraca

pozycję odnalezionego znaku (jeżeli nie wystąpił - zwracana jest długość podanego łańcucha znakowego).

4.2.1.15 Print()

```
void Print (
            TreeNode * root )
```

Funkcja przyjmująca jako parametr korzeń drzewa. Wypisuje wartości wszystkich węzłów drzewa w kolejności in-order.

Parametry

root	wskaźnik na korzeń drzewa.

4.2.1.16 PrintToFile()

```
void PrintToFile (
            TreeNode * root,
            std::ofstream & file )
```

Główny algorytm służący wypisywaniu do pliku wszystkich wartości węzłów drzewa w kolejności rosnącej.

Parametry

root	wskaźnik na korzeń drzewa.
file	strumień plikowy.

4.2.1.17 ReadFromFile()

```
void ReadFromFile (
            const std::string & file_name )
```

Funkcja odczytująca plik "linia po linii" i wykonująca odpowiednie polecenia.

Parametry

file_name	pełna nazwa pliku.

4.2.1.18 rotate_left()

```
void rotate_left (
            TreeNode *& node )
```

Lokalna zmiana struktury poddrzewa (w lewo).

Parametry

node	wskaźnik na węzeł drzewa.

Ostrzeżenie

prawe poddrzewo węzła musi istnieć!

4.2.1.19 rotate_right()

```
void rotate_right (
            TreeNode *& node )
```

Lokalna zmiana struktury poddrzewa (w lewo).

Parametry

node	wskaźnik na węzeł drzewa.

Ostrzeżenie

lewe poddrzewo węzła musi istnieć!

4.2.1.20 StartCase()

```
void StartCase (
            TreeNode * node,
            bool & isCase2 )
```

Funkcja obejmująca przestawienie odpowiednich wskaźników i dealokowanie odpowiedniej pamięci - pamięci węzła y.

Parametry

node	wskaźnik na węzeł drzewa.
isCase2	sprawdzenie, czy istnieje potrzeba dalszego przywracania własności drzewa czerwono-czarnego.

4.2.1.21 Successor()

```
TreeNode* Successor (
            TreeNode * node )
```

Funkcja odnajdująca następcę danego węzła. Jeżeli węzeł zostanie odnaleziony, funkcja zwraca wskaźnik na niego, w przeciwnym wypadku zwracany jest nullptr.

Parametry

node	wskaźnik na węzeł drzewa.

Zwraca

następca węzła.

4.3 Dokumentacja pliku main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include "funkcje.h"
```

Funkcje

- int main (int argc, char ∗argv[ ])

4.3.1 Dokumentacja funkcji

4.3.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

4.4 Dokumentacja pliku struktury.h

Komponenty

- struct TreeNode

  Struktura węzła drzewa czerwono-czarnego.

Definicje typów

- typedef double T

  Szablon typu zmiennej (domyślnie: double).

Wyliczenia

- enum Color { Color::red, Color::black }

  Typ wyliczeniowy, opisujący kolor węzła.

4.4.1 Dokumentacja definicji typów

4.4.1.1 T

```
typedef double T
```

Szablon typu zmiennej (domyślnie: double).

4.4.2 Dokumentacja typów wyliczanych

4.4.2.1 Color

```
enum Color  [strong]
```

Typ wyliczeniowy, opisujący kolor węzła.

Wartości wyliczeń

red	
black	

Indeks

_Color
TreeNode, 5

Add
funkcje.cpp, 8
funkcje.h, 17
Add_Recovery
funkcje.cpp, 8
funkcje.h, 17

black
struktury.h, 25

ChangeToColor
funkcje.cpp, 10
funkcje.h, 18
Color
struktury.h, 25

Deallocate
funkcje.cpp, 10
funkcje.h, 18
Delete
funkcje.cpp, 10
funkcje.h, 18
Delete_Recovery
funkcje.cpp, 11
funkcje.h, 19

FileGraph
funkcje.cpp, 11
funkcje.h, 19
FilePrint
funkcje.cpp, 11
funkcje.h, 19
Find
funkcje.cpp, 12
funkcje.h, 20
funkcje.cpp, 7
Add, 8
Add_Recovery, 8
ChangeToColor, 10
Deallocate, 10
Delete, 10
Delete_Recovery, 11
FileGraph, 11
FilePrint, 11
Find, 12
GetParent, 12
GetSibling, 12
Graph, 13
GraphToFile, 13
lookForSign, 13
Print, 14
PrintToFile, 14
ReadFromFile, 14
rotate_left, 14
rotate_right, 15
StartCase, 15
Successor, 15
funkcje.h, 16
Add, 17
Add_Recovery, 17
ChangeToColor, 18
Deallocate, 18
Delete, 18
Delete_Recovery, 19
FileGraph, 19
FilePrint, 19
Find, 20
GetParent, 20
GetSibling, 20
Graph, 21
GraphToFile, 21
lookForSign, 21
Print, 22
PrintToFile, 22
ReadFromFile, 22
rotate_left, 23
rotate_right, 23
StartCase, 23
Successor, 24

GetParent
funkcje.cpp, 12
funkcje.h, 20
GetSibling
funkcje.cpp, 12
funkcje.h, 20
Graph
funkcje.cpp, 13
funkcje.h, 21
GraphToFile
funkcje.cpp, 13
funkcje.h, 21

left
TreeNode, 6
lookForSign
funkcje.cpp, 13
funkcje.h, 21

main
    main.cpp, 24
main.cpp, 24
    main, 24

parent
    TreeNode, 6
Print
    funkcje.cpp, 14
    funkcje.h, 22
PrintToFile
    funkcje.cpp, 14
    funkcje.h, 22

ReadFromFile
    funkcje.cpp, 14
    funkcje.h, 22
red
    struktury.h, 25
right
    TreeNode, 6
rotate_left
    funkcje.cpp, 14
    funkcje.h, 23
rotate_right
    funkcje.cpp, 15
    funkcje.h, 23

StartCase
    funkcje.cpp, 15
    funkcje.h, 23
struktury.h, 25
    black, 25
    Color, 25
    red, 25
    T, 25
Successor
    funkcje.cpp, 15
    funkcje.h, 24

T
    struktury.h, 25
TreeNode, 5
    _Color, 5
    left, 6
    parent, 6
    right, 6
    value, 6

value
    TreeNode, 6