

Politechnika Śląska
Wydział Automatyki, Elektroniki i
Informatyki

Podstawy Programowania Komputerów

B-Drzewo

Autor	Kamil Szymański
Prowadzący	dr inż. Krzysztof Simiński
Rok akademicki	2020/2021
Kierunek	Informatyka
Rodzaj studiów	SSI
Semestr	1
Termin laboratoriów	Pon, 11:00-13:15 Śr, 13:00-15:15
Sekcja	71
Termin oddania	11/02/2021

1. Treść zadania

Napisać program sortujący liczby rzeczywiste w pewnym zbiorze. Do przechowywania liczb należy wykorzystać B-drzewo, którego rząd podany jest w pierwszym wierszu pliku. Liczby podawane są w ściśle określony 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 `print` powoduje wypisanie 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 pierwszy element każdego węzła poprzedzony jest symbolem `[a`, po ostatnim umieszczony jest symbol `]`. 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. Niepoprawne komendy są ignorowane. Przykładowy plik wejściowy:

```
% przykładowy plik wejściowy
1 % rząd B-drzewa
add 1 2 3 4 5 6 7
graph % wypisane B-drzewa
print % wypisane liczb na ekran
add 3.45 -0.32
print test-1.txt % wypisanie liczb do pliku
print + test-1.txt % dopisanie do pliku
Po komendzie graph zostanie wypisane drzewo:
```

```
    [7]
  6]
    5]
  [4
[3
    2]
  [1
```

Program uruchamiany jest z linii poleceń z wykorzystaniem jednego przełącznika:

`-i` plik wejściowy

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

Uwaga: Węzeł B-drzewa ma:

- k kluczy (liczb) pamiętanych w nim,
- $k + 1$ wskaźników do węzłów potomnych.

Rząd drzewa, czyli minimalny stopień drzewa, $t > 2$, co oznacza, że

- korzeń nie pustego drzewa musi mieć co najmniej jeden klucz,
- każdy węzeł niebędący korzeniem musi mieć co najmniej $t - 1$ kluczy (zatem ma co najmniej t potomków),
- węzeł może mieć co najwyżej $2t - 1$ kluczy (zatem ma co najwyżej $2t$ potomków)

2. Analiza zadania

Zadanie polega na implementacji struktury danych, zwanej „b-drzewo”.

2.1 Algorytmy i struktury danych

Owa struktura przechowuje wartości węzłów, rozmieszczonych w odpowiedni sposób. Każdy węzeł posiada potomka (oprócz liści) i rodzica (oprócz korzenia). Kolejne wartości są dodawane do poszczególnych węzłów, począwszy od góry. Gdy zostanie przekroczona maksymalna liczba wartości w danym węźle (jest to wartość równa dwukrotnej wartości stopnia drzewa), następuje **rozsadzenie** węzła. Utworzone zostają 2 potomki, lewy z mniejszymi wartościami od mediany i prawy w wartościach większymi. Mediana pozostaje w rodzicu. Każdy potomek oprócz korzenia musi zawierać minimalną wartość wartości, która wynosi tyle co rząd drzewa. Liście, tzn. najgłębsze węzły drzewa, muszą znajdować się na tym samym poziomie. Każdy węzeł zawiera **ilość potomków równą ilości wartości w węźle+1**. Drzewo to, z założenia, podlega **samo optymalizacji**. Musi być napisane w sposób **by** samo dotrzymywało wszystkich zasad.

3. Specyfikacja zewnętrzna

Program jest uruchamiany z poziomu linii poleceń. Aby uruchomić program, należy wpisać nazwę pliku wykonywalnego, przełącznik „-i” oraz nazwę pliku z danymi wejściowymi. Przykładowe wywołanie:

```
./main -i dane.txt
```

Jeśli użytkownik wprowadzi nieprawidłowe parametry uruchomienia, zostanie wyświetlona notka podpowiadająca jak prawidłowo uruchomić program.

4. Specyfikacja wewnętrzna

Program został napisany zgodnie z paradygmatem strukturalnym. Funkcje wykorzystywane w programie znajdują się w osobnym pliku funkcje.h i funkcje.cpp. Stworzone struktury zostały umieszczone w pliku struktury.h. Wszystkie funkcje i struktury służące do bezpośredniej obsługi i modyfikacji drzewa zostały umieszczone w pliku drzewo.h oraz drzewo.cpp.

4.1 Ogólna struktura programu

W funkcji głównej zostaje wywołana funkcja odczytaj_dane_z_pliku. Funkcja ta odczytuje dane zawarte w pliku wejściowym, pomijając spacje i komentarze i zapisuje je do wektora. Następnie wektor przekazywany jest do funkcji wykonaj_komendy, która wykonuje po kolei każde żądanie zamieszczone w pliku.

4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

5. Testowanie

Program został przetestowany na różnego rodzaju plikach. Plik pusty nie zgłosi błędu, lecz spowoduje wypisanie pustego drzewa. Program został sprawdzony pod kątem wycieków pamięci.

6. Wnioski

Program implementujący b-drzewo okazał się programem dość skomplikowanym. Samo zarządzanie danymi zapisanymi w plikach, tzn odczytywanie, zapisywanie oraz nadpisywanie nie było trudne. Najbardziej wymagającym okazała się **optymalizacja** drzewa, która wymagała szeregu skomplikowanych funkcji modyfikujących je, gdy dochodziło np. do rozbicia węzłów. Trudnością okazał się również dobór wskaźników, gdyż początkowo po zastosowaniu niewłaściwych, dochodziło do ogromnych wycieków pamięci. Mimo wszystko napisanie tego programu przyczyniło się do pogłębienia mojej wiedzy z zakresu języka C++, w szczególności o doświadczenie praktyczne, czego nie dałoby się zastąpić żadnym egzaminem.

Dodatek

Szczegółowy opis typów i funkcji

B-Drzewo

Wygenerowano przez Doxygen 1.9.1

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 komenda	5
3.1.1 Opis szczegółowy	5
3.1.2 Dokumentacja atrybutów składowych	5
3.1.2.1 nazwa_pliku	5
3.1.2.2 params	6
3.1.2.3 rodzaj	6
3.2 Dokumentacja struktury wezel	6
3.2.1 Opis szczegółowy	6
3.2.2 Dokumentacja atrybutów składowych	7
3.2.2.1 czyKorzenDoZmiany	7
3.2.2.2 czyLisc	7
3.2.2.3 potomki	7
3.2.2.4 pRodzic	7
3.2.2.5 rzad	7
3.2.2.6 wartosci	7
3.2.2.7 wskaznik_na_nowy_korzen	7
4 Dokumentacja plików	9
4.1 Dokumentacja pliku drzewo.cpp	9
4.1.1 Dokumentacja definicji	10
4.1.1.1 debug	10
4.1.2 Dokumentacja funkcji	10
4.1.2.1 czy_za_duzo_w_wezle()	10
4.1.2.2 dodaj()	10
4.1.2.3 gdy_za_duzo_w_wezle()	11
4.1.2.4 podmien_wartosci()	11
4.1.2.5 podziel_wezel()	11
4.1.2.6 utworz_nowy_korzen_drzewa()	12
4.1.2.7 wsortuj_potomka()	12
4.1.2.8 wsortuj_wartosc()	12
4.1.2.9 wypisz()	13
4.1.2.10 zapisz_do_pliku()	13
4.1.2.11 znajdz_adres_potomka()	13
4.2 Dokumentacja pliku drzewo.h	14
4.2.1 Dokumentacja funkcji	14
4.2.1.1 czy_za_duzo_w_wezle()	14

4.2.1.2 dodaj()	15
4.2.1.3 gdy_za_duzo_w_wezle()	15
4.2.1.4 podmien_wartosci()	15
4.2.1.5 podziel_wezel()	16
4.2.1.6 utworz_nowy_korzen_drzewa()	16
4.2.1.7 wsortuj_potomka()	17
4.2.1.8 wsortuj_wartosc()	17
4.2.1.9 wypisz()	17
4.2.1.10 zapisz_do_pliku()	18
4.2.1.11 znajdz_adres_potomka()	18
4.3 Dokumentacja pliku funkcje.cpp	18
4.3.1 Dokumentacja definicji	19
4.3.1.1 debug	19
4.3.2 Dokumentacja funkcji	19
4.3.2.1 help()	19
4.3.2.2 odczytaj_komendy_z_pliku()	19
4.3.2.3 sprawdz_parametry_programu()	20
4.3.2.4 test_komend()	20
4.3.2.5 to_string()	20
4.3.2.6 wykonaj_komendy()	21
4.4 Dokumentacja pliku funkcje.h	21
4.4.1 Dokumentacja funkcji	21
4.4.1.1 help()	21
4.4.1.2 odczytaj_komendy_z_pliku()	22
4.4.1.3 sprawdz_parametry_programu()	22
4.4.1.4 test_komend()	22
4.4.1.5 wykonaj_komendy()	23
4.5 Dokumentacja pliku main.cpp	23
4.5.1 Dokumentacja definicji	23
4.5.1.1 debug	24
4.5.2 Dokumentacja funkcji	24
4.5.2.1 main()	24
4.5.2.2 test()	24
4.6 Dokumentacja pliku struktury.h	24
4.6.1 Dokumentacja typów wyliczanych	24
4.6.1.1 typ	24

Rozdział 1

Indeks klas

1.1 Lista klas

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

komenda	
Struktura komendy	5
wezel	
Struktura wezła drzewa	6

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

drzewo.cpp	9
drzewo.h	14
funkcje.cpp	18
funkcje.h	21
main.cpp	23
struktury.h	24

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury komenda

Struktura komendy.

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

Atrybuty publiczne

- **typ rodzaj**
- `std::vector< double > params`
rodzaj komendy
- `std::string nazwa_pliku`
parametry liczbowe

3.1.1 Opis szczegółowy

Struktura komendy.

3.1.2 Dokumentacja atrybutów składowych

3.1.2.1 nazwa_pliku

```
std::string komenda::nazwa_pliku
```

parametry liczbowe

3.1.2.2 params

```
std::vector<double> komenda::params
```

rodzaj komendy

3.1.2.3 rodzaj

```
typ komenda::rodzaj
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- **struktury.h**

3.2 Dokumentacja struktury wezel

Struktura wezła drzewa.

```
#include <drzewo.h>
```

Atrybuty publiczne

- `std::list< double > wartosci`
Wartosci w wezle.
- `std::list< std::shared_ptr< wezel > > potomki`
Potomki wezla.
- `std::weak_ptr< wezel > pRodzic`
Rodzic wezla.
- `std::shared_ptr< wezel > wskaznik_na_nowy_korzen = nullptr`
Wskaznik ktory przechowuje adres na nowy korzen, gdy dokonujemy jego zmiany.
- `bool czyLisc {true}`
Zmienna okreslajaca czy wezel jest lisciem.
- `bool czyKorzenDoZmiany {false}`
Zmienna okreslajaca czy dany wezel jest korzeniem ktory musimy zmienic na nowy.

Statyczne atrybuty publiczne

- `static double rzad`
Zmienna okreslajaca czy dany wezel jest korzeniem ktory musimy zmienic na nowy.

3.2.1 Opis szczegółowy

Struktura wezła drzewa.

3.2.2 Dokumentacja atrybutów składowych

3.2.2.1 czyKorzenDoZmiany

```
bool wezel::czyKorzenDoZmiany {false}
```

Zmienna określająca czy wezel jest lisciem.

3.2.2.2 czyLisc

```
bool wezel::czyLisc {true}
```

Wskaznik który przechowuje adres na nowy korzen, gdy dokonujemy jego zmiany.

3.2.2.3 potomki

```
std::list<std::shared_ptr< wezel> > wezel::potomki
```

Wartosci w wezle.

3.2.2.4 pRodzic

```
std::weak_ptr< wezel> wezel::pRodzic
```

Potomki wezla.

3.2.2.5 rzad

```
double wezel::rzad [static]
```

Zmienna określająca czy dany wezel jest korzeniem który musimy zmienić na nowy.

3.2.2.6 wartosci

```
std::list<double> wezel::wartosci
```

3.2.2.7 wskaznik_na_nowy_korzen

```
std::shared_ptr< wezel> wezel::wskaznik_na_nowy_korzen = nullptr
```

Rodzic wezla.

Dokumentacja dla tej struktury została wygenerowana z plików:

- drzewo.h
- drzewo.cpp

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku drzewo.cpp

```
#include <iostream>
#include <fstream>
#include "funkcje.h"
#include "struktury.h"
#include "drzewo.h"
#include <tuple>
```

Definicje

- #define **debug**(x) std::cerr << __FILE__ << " (" << __LINE__ << ") " << #x << " == " << (x) << std::endl

Funkcje

- void **dodaj**(std::shared_ptr< **wezel** > &pKorzen, const double wartosc)
- std::tuple< std::shared_ptr< **wezel** >, double, std::shared_ptr< **wezel** > > **podziel_wezel**(std::shared_ptr< **wezel** > &pWezel)
- void **podmien_wartosci**(std::shared_ptr< **wezel** > &pWezel, const double wartosc, std::shared_ptr< **wezel** > &pLewy, double mediana, std::shared_ptr< **wezel** > &pPrawy)
- void **gdz_za_duzo_w_wezle**(std::shared_ptr< **wezel** > &pWezel, double wartosc)
- std::shared_ptr< **wezel** > **utworz_nowy_korzen_drzewa**(std::shared_ptr< **wezel** > &pLewy, const double mediana, std::shared_ptr< **wezel** > &pPrawy)
- std::shared_ptr< **wezel** > **znajdz_adres_potomka**(const std::shared_ptr< **wezel** > pWezel, const double wartosc)
- bool **czy_za_duzo_w_wezle**(const std::shared_ptr< **wezel** > pWezel)
- void **wsortuj_potomka**(const std::shared_ptr< **wezel** > &potomek, std::list< std::shared_ptr< **wezel** > > &lista_potomkow)
- void **wsortuj_wartosc**(const double wartosc, std::list< double > &wartosci)
- void **wypisz**(const std::shared_ptr< **wezel** > &pKorzen, const int wciecie)
- void **zapisz_do_pliku**(std::ofstream &plik, const std::shared_ptr< **wezel** > &pKorzen, const int wciecie)

4.1.1 Dokumentacja definicji

4.1.1.1 debug

```
#define debug(
    x ) std::cerr << __FILE__ << " (" << __LINE__ << ") " << #x << " == " <<
(x) << std::endl
```

4.1.2 Dokumentacja funkcji

4.1.2.1 czy_za_duzo_w_wezle()

```
bool czy_za_duzo_w_wezle (
    const std::shared_ptr< wezel > pWezel )
```

Funkcja sprawdza czy w wezle nie ma zbyt wielu wartosci

Parametry

<i>pWezel</i>	Wezel do sprawdzenia
---------------	----------------------

Zwraca

true – za duzo wartosci w wezle, potrzebna modyfikacja, false – poprawna ilosc wartosci w wezle

4.1.2.2 dodaj()

```
void dodaj (
    std::shared_ptr< wezel > & pKorzen,
    const double wartosc )
```

Funkcja dodajaca wartosc do B-drzewa.

Parametry

<i>in, out</i>	<i>pKorzen</i>	Korzen drzewa (jezeli drzewo jest puste, ma wartosc nullptr)
	<i>wartosc</i>	Wartosc do dodania do drzewa

4.1.2.3 gdy_za_duzo_w_wezle()

```
void gdy_za_duzo_w_wezle (
    std::shared_ptr< wezel > & pWezel,
    double wartosc )
```

Funkcja ktora stanowi posrednik miedzy funkcja dodaj, a funkcjami ktore modyfikuja wezly

Parametry

in, out	pWezel	Wezel drzewa
	wartosc	Wartosc ktora spowodowala przeciazenie wezla

4.1.2.4 podmien_wartosci()

```
void podmien_wartosci (
    std::shared_ptr< wezel > & pWezel,
    const double wartosc,
    std::shared_ptr< wezel > & pLewy,
    double mediana,
    std::shared_ptr< wezel > & pPrawy )
```

Funkcja usuwa zbyt duzy wezel i wkłada na jego miejsce 2 nowe wezly (lewy i prawy) oraz mediane do rodzica

Parametry

in, out	pWezel	Rozbity wezel
	wartosc	Wartosc ktora wywolala rozbite
in, out	pLewy	Lewy wezel utworzony po rozbiciu oryginalnego wezla
	mediana	Mediana, czyli wartosc srodkowa, ktora idzie to rodzica wezla
in, out	pPrawy	Prawy wezel utworzony po rozbiciu oryginalnego wezla

4.1.2.5 podziel_wezel()

```
std::tuple<std::shared_ptr< wezel>, double, std::shared_ptr< wezel> > podziel_wezel (
    std::shared_ptr< wezel > & pWezel )
```

Funkcja dzieli zbyt duzy wezel na prawa i lewa wartosc oraz mediane i przekazuje do funkcji gdy_za_duzo_w_wezle, ktora decyduje co robic dalej

Parametry

in, out	pWezel	Wezel drzewa do podzielenia
---------	--------	-----------------------------

Zwraca

Lewy wezel
 Mediana
 Prawy wezel

4.1.2.6 utworz_nowy_korzen_drzewa()

```
std::shared_ptr< wezel> utworz_nowy_korzen_drzewa (
    std::shared_ptr< wezel > & pLewy,
    const double mediana,
    std::shared_ptr< wezel > & pPrawy )
```

Funkcja przyjmuje lewy i prawy wezel oraz mediane (po rozbiciu starego korzenia), tworzy nowy korzen, dodaje do niego lewe i prawe potomki, zwraca wskaznik na nowy korzen, który podmieniamy z oryginalnym wskaznikiem

Parametry

in, out	<i>pLewy</i>	Lewy wezel
	<i>mediana</i>	Mediana
in, out	<i>pPrawy</i>	Prawy wezel

Zwraca

Nowy korzen

4.1.2.7 wsortuj_potomka()

```
void wsortuj_potomka (
    const std::shared_ptr< wezel > & potomek,
    std::list< std::shared_ptr< wezel >> & lista_potomkow )
```

Funkcja dodaje potomka do listy potomkow, wsadzając go w odpowiednie miejsce, tak by lista była posortowana rosnaco wg pierwszych wartosci potomkow

Parametry

in, out	<i>potomek</i>	Potomek do wsortowania
in, out	<i>lista_potomkow</i>	Lista do której wsortujemy potomka

4.1.2.8 wsortuj_wartosc()

```
void wsortuj_wartosc (
```

```
const double wartosc,  
std::list< double > & wartosci )
```

Funkcja dodaje wartosc do listy i ja sortuje

Parametry

	<i>wartosc</i>	Dodawana wartosc
<i>in, out</i>	<i>wartosci</i>	Lista wartosci do ktorej dodajemy

4.1.2.9 wypisz()

```
void wypisz (  
    const std::shared_ptr< wezel > & pKorzen,  
    const int wciecie )
```

Funkcja wypisująca graf drzewa na standardowe wyjście

Parametry

<i>in, out</i>	<i>pKorzen</i>	Korzen drzewa
	<i>wciecie</i>	Wspolczynnik wciecia

4.1.2.10 zapisz_do_pliku()

```
void zapisz_do_pliku (  
    std::ofstream & plik,  
    const std::shared_ptr< wezel > & pKorzen,  
    const int wciecie )
```

Funkcja zapisująca graf drzewa do pliku

Parametry

<i>in, out</i>	<i>plik</i>	Strumien wyjsciowy do pliku
<i>in, out</i>	<i>pKorzen</i>	Korzen drzewa
	<i>wciecie</i>	Wspolczynnik wciecia

4.1.2.11 znajdz_adres_potomka()

```
std::shared_ptr< wezel> znajdz_adres_potomka (  
    const std::shared_ptr< wezel > pWezel,  
    const double wartosc )
```

Funkcja znajduje adres potomka do ktorego kierujemy wartosc

Parametry

<i>pWezel</i>	Rodzic, wsrod ktorego potomkow szukamy takiego, do ktorego ma trafic wartosc
<i>wartosc</i>	Wartosc dla ktorej szukamy potomka

Zwraca

Szukany potomek

4.2 Dokumentacja pliku drzewo.h

```
#include <vector>
#include <string>
#include <memory>
#include <list>
```

Komponenty

- struct **wezel**

Struktura wezla drzewa.

Funkcje

- void **dodaj** (std::shared_ptr< **wezel** > &pKorzen, const double wartosc)
- void **wypisz** (const std::shared_ptr< **wezel** > &pKorzen, const int wciecie)
- void **zapisz_do_pliku** (std::ofstream &pplik, const std::shared_ptr< **wezel** > &pKorzen, const int wciecie)
- void **wsortuj_wartosc** (const double wartosc, std::list< double > &wartosci)
- bool **czy_za_duzo_w_wezle** (const std::shared_ptr< **wezel** > pWezel)
- void **gdz_za_duzo_w_wezle** (std::shared_ptr< **wezel** > &pWezel, double wartosc)
- std::tuple< std::shared_ptr< **wezel** >, double, std::shared_ptr< **wezel** > > **podziel_wezel** (std::shared_ptr< **wezel** > &pWezel)
- void **podmien_wartosci** (std::shared_ptr< **wezel** > &pWezel, const double wartosc, std::shared_ptr< **wezel** > &pLewy, double mediana, std::shared_ptr< **wezel** > &pPrawy)
- std::shared_ptr< **wezel** > **utworz_nowy_korzen_drzewa** (std::shared_ptr< **wezel** > &pLewy, const double mediana, std::shared_ptr< **wezel** > &pPrawy)
- std::shared_ptr< **wezel** > **znajdz_adres_potomka** (const std::shared_ptr< **wezel** > pWezel, const double wartosc)
- void **wsortuj_potomka** (const std::shared_ptr< **wezel** > &potomek, std::list< std::shared_ptr< **wezel** > > &lista_potomkow)

4.2.1 Dokumentacja funkcji

4.2.1.1 czy_za_duzo_w_wezle()

```
bool czy_za_duzo_w_wezle (
    const std::shared_ptr< wezel > pWezel )
```

Funkcja sprawdza czy w wezle nie ma zbyt wielu wartosci

Parametry

<i>pWezel</i>	Wezel do sprawdzenia
---------------	----------------------

Zwraca

true – za duzo wartosci w wezle, potrzebna modyfikacja, false – poprawna ilosc wartosci w wezle

4.2.1.2 dodaj()

```
void dodaj (
    std::shared_ptr< wezel > & pKorzen,
    const double wartosc )
```

Funkcja dodająca wartosc do B-drzewa.

Parametry

<i>in, out</i>	<i>pKorzen</i>	Korzen drzewa (jezeli drzewo jest puste, ma wartosc nullptr)
	<i>wartosc</i>	Wartosc do dodania do drzewa

4.2.1.3 gdy_za_duzo_w_wezle()

```
void gdy_za_duzo_w_wezle (
    std::shared_ptr< wezel > & pWezel,
    double wartosc )
```

Funkcja ktora stanowi posrednik miedzy funkcja dodaj, a funkcjami ktore modyfikuja wezly

Parametry

<i>in, out</i>	<i>pWezel</i>	Wezel drzewa
	<i>wartosc</i>	Wartosc ktora spowodowała przeciążenie wezła

4.2.1.4 podmien_wartosci()

```
void podmien_wartosci (
    std::shared_ptr< wezel > & pWezel,
    const double wartosc,
    std::shared_ptr< wezel > & pLewy,
```

```
double mediana,
std::shared_ptr< wezel > & pPrawy )
```

Funkcja usuwa zbyt duży węzeł i wkłada na jego miejsce 2 nowe węzły (lewy i prawy) oraz medianę do rodzica

Parametry

in, out	<i>pWezel</i>	Rozbity węzeł
	<i>wartosc</i>	Wartość która wywołała rozbięcie
in, out	<i>pLewy</i>	Lewy węzeł utworzony po rozbięciu oryginalnego węzła
	<i>mediana</i>	Mediana, czyli wartość środkowa, która idzie to rodzica węzła
in, out	<i>pPrawy</i>	Prawy węzeł utworzony po rozbięciu oryginalnego węzła

4.2.1.5 podziel_wezel()

```
std::tuple<std::shared_ptr< wezel>, double, std::shared_ptr< wezel> > podziel_wezel (
    std::shared_ptr< wezel > & pWezel )
```

Funkcja dzieli zbyt duży węzeł na prawa i lewą wartość oraz medianę i przekazuje do funkcji `gdy_za_duzo_w_wezle`, która decyduje co robić dalej

Parametry

in, out	<i>pWezel</i>	Węzeł drzewa do podzielenia
---------	---------------	-----------------------------

Zwraca

Lewy węzeł
Mediana
Prawy węzeł

4.2.1.6 utworz_nowy_korzen_drzewa()

```
std::shared_ptr< wezel> utworz_nowy_korzen_drzewa (
    std::shared_ptr< wezel > & pLewy,
    const double mediana,
    std::shared_ptr< wezel > & pPrawy )
```

Funkcja przyjmuje lewy i prawy węzeł oraz medianę (po rozbięciu starego korzenia), tworzy nowy korzeń, dodaje do niego lewe i prawe potomki, zwraca wskaźnik na nowy korzeń, który podmieniamy z oryginalnym wskaźnikiem

Parametry

in, out	<i>pLewy</i>	Lewy węzeł
	<i>mediana</i>	Mediana
in, out	<i>pPrawy</i>	Prawy węzeł

Zwraca

Nowy korzen

4.2.1.7 wsortuj_potomka()

```
void wsortuj_potomka (
    const std::shared_ptr< wezel > & potomek,
    std::list< std::shared_ptr< wezel >> & lista_potomkow )
```

Funkcja dodaje potomka do listy potomkow, wsadzajac go w odpowiednie miejsce, tak by lista byla posortowana rosnaco wg pierwszych wartosci potomkow

Parametry

in, out	<i>potomek</i>	Potomek do wsortowania
in, out	<i>lista_potomkow</i>	Lista do ktorej wsortujemy potomka

4.2.1.8 wsortuj_wartosc()

```
void wsortuj_wartosc (
    const double wartosc,
    std::list< double > & wartosci )
```

Funkcja dodaje wartosc do listy i ja sortuje

Parametry

	<i>wartosc</i>	Dodawana wartosc
in, out	<i>wartosci</i>	Lista wartosci do ktorej dodajemy

4.2.1.9 wypisz()

```
void wypisz (
    const std::shared_ptr< wezel > & pKorzen,
    const int wciecie )
```

Funkcja wypisujaca graf drzewa na standardowe wyjscie

Parametry

in, out	<i>pKorzen</i>	Korzen drzewa
	<i>wciecie</i>	Wspolczynnik wciecia

4.2.1.10 zapisz_do_pliku()

```
void zapisz_do_pliku (
    std::ofstream & plik,
    const std::shared_ptr< wezel > & pKorzen,
    const int wciecie )
```

Funkcja zapisująca graf drzewa do pliku

Parametry

in, out	<i>plik</i>	Strumień wyjściowy do pliku
in, out	<i>pKorzen</i>	Korzeń drzewa
	<i>wciecie</i>	Współczynnik wcięcia

4.2.1.11 znajdz_adres_potomka()

```
std::shared_ptr< wezel> znajdz_adres_potomka (
    const std::shared_ptr< wezel > pWezel,
    const double wartosc )
```

Funkcja znajduje adres potomka do którego kierujemy wartość

Parametry

<i>pWezel</i>	Rodzic, wśród którego potomków szukamy takiego, do którego ma trafić wartość
<i>wartosc</i>	Wartość dla której szukamy potomka

Zwraca

Szukany potomek

4.3 Dokumentacja pliku funkcje.cpp

```
#include <iostream>
#include <vector>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <list>
#include <algorithm>
#include "funkcje.h"
#include "struktury.h"
#include "drzewo.h"
```

Definicje

- `#define debug(x) std::cerr << __FILE__ << " (" << __LINE__ << ") " << #x << " == " << (x) << std::endl`

Funkcje

- `std::vector< komenda > odczytaj_komendy_z_pliku (const std::string &nazwa)`
- `std::string to_string (const typ &t)`
- `void test_komend (const std::vector< komenda > &ciag_komend)`
- `void help (const std::string &nazwa_pliku_wykonywalnego)`
- `bool sprawdz_parametry_programu (int argc, char *params[], std::string &nazwa_pliku)`
- `void wykonaj_komendy (std::shared_ptr< wezel > &pKorzen, const std::vector< komenda > &ciag_komend)`

4.3.1 Dokumentacja definicji

4.3.1.1 debug

```
#define debug(
    x ) std::cerr << __FILE__ << " (" << __LINE__ << ") " << #x << " == " <<
(x) << std::endl
```

4.3.2 Dokumentacja funkcji

4.3.2.1 help()

```
void help (
    const std::string & nazwa_pliku_wykonywalnego )
```

Funkcja wyświetlająca notkę przypominającą jak uruchomić program

Parametry

in, out	<i>nazwa_pliku_wykonywalnego</i>	Nazwa pliku wykonywalnego
---------	----------------------------------	---------------------------

4.3.2.2 odczytaj_komendy_z_pliku()

```
std::vector< komenda> odczytaj_komendy_z_pliku (
    const std::string & nazwa )
```

Funkcja odczytuje z pliku wejściowego komendy wraz z parametrami, pomijając komentarze

Parametry

<i>in, out</i>	<i>Nazwa</i>	strumienia pliku wejściowego
----------------	--------------	------------------------------

Zwraca

Ciąg komend

4.3.2.3 sprawdz_parametry_programu()

```
bool sprawdz_parametry_programu (
    int argc,
    char * params[],
    std::string & nazwa_pliku )
```

Parametry

	<i>argc</i>	Pierwszy parametr main
	<i>params</i>	Drugi parametr main
<i>in, out</i>	<i>nazwa_pliku</i>	Jezeli poprawne parametry wywołania programu, to tutaj będzie odczytana nazwa pliku.

Zwraca

true – wszystko OK, parametry poprawne, false – parametry błędne

4.3.2.4 test_komend()

```
void test_komend (
    const std::vector< komenda > & ciag_komend )
```

Testowe wypisanie na ekran nazwy każdej komendy (i jej parametrów)

Parametry

<i>in, out</i>	<i>ciag_komend</i>	Komendy zebrane z pliku wejściowego
----------------	--------------------	-------------------------------------

4.3.2.5 to_string()

```
std::string to_string (
```

```
const typ & t )
```

4.3.2.6 wykonaj_komendy()

```
void wykonaj_komendy (
    std::shared_ptr< wezel > & pKorzen,
    const std::vector< komenda > & ciag_komend )
```

Funkcja wykonująca komendy zebrane z pliku wejściowego

Parametry

in, out	<i>pKorzen</i>	Korzen drzewa
in, out	<i>ciag_komend</i>	Komendy zebrane z pliku wejściowego

4.4 Dokumentacja pliku funkcje.h

```
#include <vector>
#include <string>
#include <memory>
#include "struktury.h"
#include "drzewo.h"
```

Funkcje

- std::vector< **komenda** > **odczytaj_komendy_z_pliku** (const std::string &nazwa)
- void **test_komend** (const std::vector< **komenda** > &ciag_komend)
- bool **sprawdz_parametry_programu** (int argc, char *params[], std::string &nazwa_pliku)
- void **help** (const std::string &nazwa_pliku_wykonywalnego)
- void **wykonaj_komendy** (std::shared_ptr< **wezel** > &pKorzen, const std::vector< **komenda** > &ciag_komend)

4.4.1 Dokumentacja funkcji

4.4.1.1 help()

```
void help (
    const std::string & nazwa_pliku_wykonywalnego )
```

Funkcja wyświetlająca notkę przypominającą jak uruchomić program

Parametry

<i>in, out</i>	<i>nazwa_pliku_wykonywalnego</i>	Nazwa pliku wykonywalnego
----------------	----------------------------------	---------------------------

4.4.1.2 odczytaj_komendy_z_pliku()

```
std::vector< komenda> odczytaj_komendy_z_pliku (
    const std::string & nazwa )
```

Funkcja odczytuje z pliku wejsciowego komendy wraz z parametrami, pomijajac komentarze

Parametry

<i>in, out</i>	<i>Nazwa</i>	strumienia pliku wejsciowego
----------------	--------------	------------------------------

Zwraca

Ciag komend

4.4.1.3 sprawdz_parametry_programu()

```
bool sprawdz_parametry_programu (
    int argc,
    char * params[],
    std::string & nazwa_pliku )
```

Parametry

	<i>argc</i>	Pierwszy parametr main
	<i>params</i>	Drugi parametr main
<i>in, out</i>	<i>nazwa_pliku</i>	Jezeli poprawne parametry wywolania programu, to tutaj bedzie odczytana nazwa pliku.

Zwraca

true – wszystko OK, parametry poprawne, false – parametry bledne

4.4.1.4 test_komend()

```
void test_komend (
    const std::vector< komenda > & ciag_komend )
```

Testowe wypisanie na ekran nazwy kazdej komendy (i jej parametrow)

Parametry

in, out	<i>ciag_komend</i>	Komendy zebrane z pliku wejsciowego
---------	--------------------	-------------------------------------

4.4.1.5 wykonaj_komendy()

```
void wykonaj_komendy (
    std::shared_ptr< wezel > & pKorzen,
    const std::vector< komenda > & ciag_komend )
```

Funkcja wykonujaca komendy zebrane z pliku wejsciowego

Parametry

in, out	<i>pKorzen</i>	Korzen drzewa
in, out	<i>ciag_komend</i>	Komendy zebrane z pliku wejsciowego

4.5 Dokumentacja pliku main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <memory>
#include <algorithm>
#include <numeric>
#include <random>
#include "funkcje.h"
#include "struktury.h"
#include "drzewo.h"
```

Definicje

- #define **debug**(x) std::cerr << __FILE__ << " (" << __LINE__ << ") " << #x << " == " << (x) << std::endl

Funkcje

- void **test** ()
- int **main** (int ile, char *params[])

4.5.1 Dokumentacja definicji

4.5.1.1 debug

```
#define debug(  
    x ) std::cerr << __FILE__ << " (" << __LINE__ << ") " << #x << " == " <<  
(x) << std::endl
```

4.5.2 Dokumentacja funkcji

4.5.2.1 main()

```
int main (  
    int ile,  
    char * params[] )
```

4.5.2.2 test()

```
void test ( )
```

4.6 Dokumentacja pliku struktury.h

```
#include <vector>  
#include <string>  
#include <memory>  
#include <list>
```

Komponenty

- struct **komenda**
Struktura komendy.

Wyliczenia

- enum class **typ** {
 rzad, **add**, **graph**, **print**,
 append }
Typ wyliczeniowy określający rodzaje komend.

4.6.1 Dokumentacja typów wyliczanych

4.6.1.1 typ

```
enum typ [strong]
```

Typ wyliczeniowy określający rodzaje komend.

Wartości wyliczeń

rzad	
add	Rzad drzewa.
graph	Komenda dodaj.
print	Komenda wypisz drzewo na standardowe urządzenie wyjściowe.
append	Komenda wypisz wartosci drzewa, lub zapisz graf do pliku. Inaczej print +, czyli dopisanie do pliku

