

Politechnika Śląska  
Wydział Informatyki Elektroniki i Informatyki

# Podstawy Programowania Komputerów

## Mapa (18)

---

Autor	Mateusz Grabarczyk
Prowadzący	dr. Inż. Adam Gudyś
Rok akademicki	2020/2021
Kierunek	Informatyka
Rodzaj studiów	SSI
Semestr	1
Termin laboratoriów	Poniedziałek, 10:30-12:45 Środa, 8:00-10:15
Sekcja	11
Termin oddania sprawozdania	

---

# 1 Treść zadania

Napisać program, który umożliwia znalezienie najkrótszej trasy między dwoma miastami. Miasta połączone są drogami o pewnej długości. Drogi są jednokierunkowe. Plik mapy dróg ma następującą postać: W każdej linii podana jest jedna droga:

<miasto początkowe> <miasto końcowe> <odległość>

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-d plik wejściowy z drogami

-t plik wejściowy z trasami do wyznaczenia

-o plik wynikowy z wyznaczonymi trasami

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej instrukcji.

## 2 Analiza zadania

Zagadnienie przedstawia problem znajdowania najkrótszej drogi między drogami podanymi w pliku tekstowym.

### 2.1 Struktury danych

W programie wykorzystano macierz sąsiedztwa do przechowywania wartości. Macierz sąsiedztwa jest dynamiczną tablicą dwuwymiarową. Taka struktura umożliwia łatwy dostęp do danych.

### 2.2 Algorytmy

Program znajduje najkrótsze drogi między podanymi miastami poprzez wykorzystanie algorytmu Dijkstry. Wykorzystując zwykłą implementację poprzez zwykłą tablicę, otrzymujemy algorytm o złożoności  $O(V^2)$ . Algorytm służy do znajdowania najkrótszej ścieżki z pojedynczego źródła w grafie o nieujemnych wagach krawędzi.

## 3 Specyfikacja zewnętrzna

Program uruchamiany jest z linii poleceń. Należy przekazać do programu nazwy plików: dwóch wejściowych oraz jednego wyjściowego po odpowiednich przełącznikach (odpowiednio -d dla pliku wejściowego z drogami, -t dla pliku wejściowego z trasami do wyznaczenia i -o plik wyjściowy wynikowy z wyznaczonymi trasami np. :

Program -d drogi.txt -t trasy.txt -o wyniki.txt

Pliki są plikami tekstowymi. Przełączniki mogą być podane w dowolnej kolejności. Urochomienie programu bez przynajmniej jednego parametru powoduje wyświetlenie krótkiej instrukcji : “Podane argumenty sa złe”.

## 4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym.

### 4.1 Ogólna struktura programu

Na początku sprawdzamy, czy argumenty programu są prawidłowe. Jeśli program nie został wywołany prawidłowo, zostaje wypisana krótka instrukcja. W funkcji głównej tworzony jest vector miast, które są obecne w grafie oraz graf za pomocą tablicy dynamicznej dwuwymiarowej, o wielkości vectora miast. Następnie graf jest wypełniany wartością 0. Funkcja *zapisanie\_do\_grafu* sprawdza, czy graf udało się wypełnić. Jeśli nie, to program zostaje zakończony za pomocą `return -1`. Następnie przy użyciu funkcji *pobranie\_tras* pobieramy do vectora par (string, string) trasy do wyznaczenia. Jeśli vector będzie pusty, wtedy program zostaje zakończony za pomocą `return -1`. W przed ostatnim kroku przy użyciu instrukcji warunkowej sprawdzamy, czy oba miasta z każdego elementu vectora par są w vectorze z miastami obecnymi w grafie. Jeśli tak, wtedy wywołujemy funkcję *dijkstra* odpowiednimi argumentami. Funkcja ta znajduje najkrótsze ścieżki do wszystkich pozostałych miast, a następnie zapisuje do pliku wyjściowego z wyznaczonymi trasami daną trasę z miasta startowego do końcowego. Jeśli nie, wtedy do pliku wyjściowego wypisywane są dwa podane miasta oraz krótka instrukcja, że trasy nie da się wyznaczyć. Na samym końcu zwalniamy pamięć grafu.

### 4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji jest zawarty w załączniku.

## 5 Testowanie

Program został przetestowany na różnego rodzaju plikach. Program pomija linie w plikach wejściowych, które są błędnie sformatowane (np. W pliku z drogami będzie o jedno za mało słowo, czy nie będzie odległości między miastami). Jeśli któryś z plików wejściowych będzie pusty, wtedy program zakończy się przy pomocy *return -1*; Jeśli podany plik wynikowy nie istnieje, program automatycznie utworzy plik o takiej nazwie i prześle do niego wyniki działania programu. Program został sprawdzony pod kątem wycieków pamięci przy użyciu Valgrind.

```

mateusz@mateusz-Latitude-E7240:~/377c082e-gr11-repo/projekt/MapaPPK/MapaPPK$ valgrind ./main -d drogi.txt -o zapisane.txt -t DrogiDoWyznaczenia.txt
==67397== Memcheck, a memory error detector
==67397== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==67397== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==67397== Command: ./main -d drogi.txt -o zapisane.txt -t DrogiDoWyznaczenia.txt
==67397==
0. Tarnow
1. Krosno
2. Warszawa
3. Torun
4. Lodz
5. Czestochowa
6. Katowice
7. Krakow
8. Gliwice
9. Wroclaw
10. Walbrzych
11. Poznan

0 80 250 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
250 0 0 100 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 70 0 0 0 0 0 0 0 0
0 0 0 100 0 0 0 0 0 0 0
0 0 0 0 75 0 80 30 0 0 0
100 0 0 0 0 80 0 0 0 0 0
0 0 0 0 0 30 0 0 130 0 0
0 0 0 0 0 0 0 0 30 140
0 0 0 0 0 0 0 0 0 0 0
0 0 100 0 0 0 0 0 0 0 0

==67397==
==67397== HEAP SUMMARY:
==67397==    in use at exit: 0 bytes in 0 blocks
==67397==   total heap usage: 297 allocs, 297 frees, 366,045 bytes allocated
==67397==
==67397== All heap blocks were freed -- no leaks are possible
==67397==
==67397== For lists of detected and suppressed errors, rerun with: -s
==67397== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

## 6 Wnioski

Program do wyznaczania najkrótszej drogi między miastami jest programem prostym, lecz wymaga bardzo dobrej znajomości i zrozumienia algorytmu, na którym się opiera. Najbardziej wymagającym okazało się stworzenie grafu, na którym można w miarę prosto oprzeć algorytm Dijkstry. Nie łatwym zadaniem było też zapisanie wyników do pliku wyjściowego, ponieważ trzeba było użyć rekurencji. Laboratorium pomogło mi w obsłudze różnych struktur danych, których użyłem, przykładowo wektorów par. Ostatnią rzeczą, która była już tylko szukaniem dziury w programie, było wyłapywanie błędów i ich naprawa.

## **Dodatek**

# Szczegółowy opis typów i funkcji

## Projekt Mapa

Wygenerowano przez Doxygen 1.8.20





<b>1 Indeks plików</b>	<b>1</b>
1.1 Lista plików	1
<b>2 Dokumentacja plików</b>	<b>3</b>
2.1 Dokumentacja pliku drogi.txt	3
2.2 Dokumentacja pliku DrogiDoWyznaczenia.txt	3
2.3 Dokumentacja pliku funkcje.cpp	3
2.3.1 Dokumentacja funkcji	4
2.3.1.1 dijkstra()	4
2.3.1.2 drukuj_sciezke()	4
2.3.1.3 pobranie_tras()	5
2.3.1.4 vector_z_miastami()	5
2.3.1.5 zapisanie_do_grafu()	5
2.3.1.6 zwroc_indeks()	6
2.3.1.7 zwroc_najblizsze_miasto()	6
2.4 Dokumentacja pliku funkcje.h	7
2.4.1 Dokumentacja funkcji	8
2.4.1.1 dijkstra()	8
2.4.1.2 drukuj_sciezke()	8
2.4.1.3 pobranie_tras()	9
2.4.1.4 vector_z_miastami()	9
2.4.1.5 zapisanie_do_grafu()	9
2.4.1.6 zwroc_indeks()	10
2.4.1.7 zwroc_najblizsze_miasto()	10
2.5 Dokumentacja pliku main.cpp	10
2.5.1 Dokumentacja funkcji	11
2.5.1.1 main()	11
2.6 Dokumentacja pliku zapis.txt	11
2.7 Dokumentacja pliku zapisane.txt	11
<b>Indeks</b>	<b>13</b>



# Rozdział 1

## Indeks plików

### 1.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

<a href="#">funkcje.cpp</a>	.....	3
<a href="#">funkcje.h</a>	.....	7
<a href="#">main.cpp</a>	.....	10



## Rozdział 2

# Dokumentacja plików

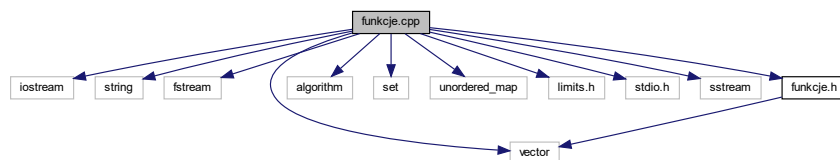
### 2.1 Dokumentacja pliku drogi.txt

### 2.2 Dokumentacja pliku DrogiDoWyznaczenia.txt

### 2.3 Dokumentacja pliku funkcje.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <algorithm>
#include <set>
#include <unordered_map>
#include <limits.h>
#include <stdio.h>
#include <sstream>
#include "funkcje.h"
```

Wykres zależności załączania dla funkcje.cpp:



## Funkcje

- void **drukuj\_sciezke** (int \*p, int \*\*odl, vector< string > miasta, int m, string nazwa\_pliku)  
*Drukowanie sciezki rekurencyjnie od miasta startowego do miasta koncowego.*
- vector< string > **vector\_z\_miastami** (string nazwa\_pliku)  
*Pobiera miasta obecne w grafie.*

- `vector< pair< string, string > >` [pobranie\\_tras](#) (const string &nazwa\_pliku)  
*Pobranie tras, które trzeba wyznaczyć.*
- `int` [zwroc\\_indeks](#) (vector< string > miasta, string miasto)  
*Funkcja zwracająca indeks miasta w wektorze 'miasta'.*
- `bool` [zapisanie\\_do\\_grafu](#) (string nazwa\_pliku, vector< string > miasta, int \*\*graf)  
*Funkcja zapisująca połączenia między miastami do grafu.*
- `int` [zwroc\\_najblizsze\\_miasto](#) (int \*odleglosci, bool \*sprawdzone, vector< string > miasta)  
*Funkcja zwracająca indeks miasta z najniższą odległością.*
- `void` [dijkstra](#) (string m\_start, string m\_koniec, vector< string > miasta, int \*\*graf, string nazwa\_pliku\_wyjsc)  
*Główny algorytm odpowiadający za wyznaczenie najkrótszych dróg z miasta startowego do reszty miast.*

## 2.3.1 Dokumentacja funkcji

### 2.3.1.1 dijkstra()

```
void dijkstra (
    string m_start,
    string m_koniec,
    vector< string > miasta,
    int ** graf,
    string nazwa_pliku_wyjsc )
```

Główny algorytm odpowiadający za wyznaczenie najkrótszych dróg z miasta startowego do reszty miast.

#### Parametry

<i>m_start</i>	miasto startowe
<i>m_koniec</i>	miasto końcowe
<i>miasta</i>	wektor miast obecnych w grafie
<i>graf</i>	macierz sąsiedztwa (graf)
<i>nazwa_pliku_wyjsc</i>	nazwa pliku wyjściowego, do którego zapiszemy trasy

### 2.3.1.2 drukuj\_sciezke()

```
void drukuj_sciezke (
    int * p,
    int ** odl,
    vector< string > miasta,
    int m,
    string nazwa_pliku )
```

Drukowanie ścieżki rekurencyjnie od miasta startowego do miasta końcowego.

**Parametry**

<i>p</i>	lista poprzedników
<i>odl</i>	graf z odleglosciami
<i>miasta</i>	vector miast w grafie
<i>m</i>	miasto koncowe
<i>nazwa_pliku</i>	nazwa pliku wyjsciowego, do ktorego zapisujemy dana trase

**2.3.1.3 pobranie\_tras()**

```
vector<pair<string, string> > pobranie_tras (
    const string & nazwa_pliku )
```

Pobranie tras, ktore trzeba wyznaczyc.

**Parametry**

<i>nazwa_pliku</i>	nazwa pliku z trasami, ktore trzeba wyznaczyc
--------------------	---

**Zwraca**

wektor par string string, ktorego pierwszym elementem pary jest miasto startowe, a drugim koncowe

**2.3.1.4 vector\_z\_miastami()**

```
vector<string> vector_z_miastami (
    string nazwa_pliku )
```

Pobiera miasta obecne w grafie.

**Parametry**

<i>nazwa_pliku</i>	nazwa pliku z drogami
--------------------	-----------------------

**Zwraca**

set z miastami obecnymi w grafie

**2.3.1.5 zapisanie\_do\_grafu()**

```
bool zapisanie_do_grafu (
    string nazwa_pliku,
```

```
vector< string > miasta,
int ** graf )
```

Funkcja zapisująca połączenia między miastami do grafu.

#### Parametry

<i>nazwa_pliku</i>	nazwa pliku tekstowego z miastami
<i>miasta</i>	vector miast obecnych w grafie
<i>graf</i>	graf prezentujący połączenia między miastami

#### Zwraca

true or false, w zależności, czy plik jest pusty czy nie

### 2.3.1.6 zwroc\_indeks()

```
int zwroc_indeks (
    vector< string > miasta,
    string miasto )
```

Funkcja zwracająca indeks miasta w wektorze 'miasta'.

#### Parametry

<i>miasta</i>	wektor z miastami obecnymi w grafie
<i>miasto</i>	miasto, którego indeks chcemy otrzymać

#### Zwraca

indeks miasta w wektorze 'miasta'

### 2.3.1.7 zwroc\_najblizsze\_miasto()

```
int zwroc_najblizsze_miasto (
    int * odleglosci,
    bool * sprawdzone,
    vector< string > miasta )
```

Funkcja zwracająca indeks miasta z najniższą odległością.

#### Parametry

<i>odleglosci</i>	tablica zawierająca odległości
<i>sprawdzone</i>	tablica bool zawierająca wierzchołki (sprawdza czy są odwiedzone czy też nie)
<i>miasta</i>	vector miast (służy tutaj do wyznaczenia ilości wierzchołków w grafie)



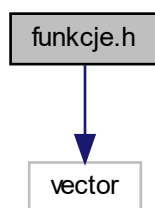
Zwraca

zwraca indeks miasta z najniższą odległością (waga)

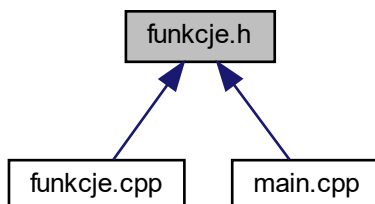
## 2.4 Dokumentacja pliku funkcje.h

```
#include <vector>
```

Wykres zależności załączania dla funkcje.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



### Funkcje

- `vector< string > vector\_z\_miastami` (string nazwa\_pliku)  
*Pobiera miasta obecne w grafie.*
- `vector< pair< string, string > > pobranie\_tras` (const string &nazwa\_pliku)  
*Pobranie tras, które trzeba wyznaczyć.*
- `int zwroc\_indeks` (vector< string > miasta, string miasto)  
*Funkcja zwracająca indeks miasta w wektorze 'miasta'.*
- `bool zapisanie\_do\_grafu` (string nazwa\_pliku, vector< string > miasta, int \*\*graf)  
*Funkcja zapisująca połączenia między miastami do grafu.*
- `int zwroc\_najblizsze\_miasto` (int \*odleglosci, bool \*sprawdzone, vector< string > miasta)  
*Funkcja zwracająca indeks miasta z najniższą odległością.*

- void `dijkstra` (string `m_start`, string `m_koniec`, vector< string > `miasta`, int `**graf`, string `nazwa_pliku_wyjsc`)  
*Główny algorytm odpowiadający za wyznaczenie najkrótszych dróg z miasta startowego do reszty miast.*
- void `drukuj_sciezke` (int `*p`, int `**odl`, vector< string > `miasta`, int `m`, string `nazwa_pliku`)  
*Drukowanie ścieżki rekurencyjnie od miasta startowego do miasta końcowego.*

## 2.4.1 Dokumentacja funkcji

### 2.4.1.1 `dijkstra()`

```
void dijkstra (
    string m_start,
    string m_koniec,
    vector< string > miasta,
    int ** graf,
    string nazwa_pliku_wyjsc )
```

Główny algorytm odpowiadający za wyznaczenie najkrótszych dróg z miasta startowego do reszty miast.

#### Parametry

<code>m_start</code>	miasto startowe
<code>m_koniec</code>	miasto końcowe
<code>miasta</code>	wektor miast obecnych w grafie
<code>graf</code>	macierz sąsiedztwa (graf)
<code>nazwa_pliku_wyjsc</code>	nazwa pliku wyjściowego, do którego zapiszemy trasy

### 2.4.1.2 `drukuj_sciezke()`

```
void drukuj_sciezke (
    int * p,
    int ** odl,
    vector< string > miasta,
    int m,
    string nazwa_pliku )
```

Drukowanie ścieżki rekurencyjnie od miasta startowego do miasta końcowego.

#### Parametry

<code>p</code>	lista poprzedników
<code>odl</code>	graf z odległościami
<code>miasta</code>	vector miast w grafie
<code>m</code>	miasto końcowe
<code>nazwa_pliku</code>	nazwa pliku wyjściowego, do którego zapisujemy daną trasę

### 2.4.1.3 pobranie\_tras()

```
vector<pair<string, string> > pobranie_tras (
    const string & nazwa_pliku )
```

Pobranie tras, które trzeba wyznaczyć.

#### Parametry

<i>nazwa_pliku</i>	nazwa pliku z trasami, które trzeba wyznaczyć
--------------------	---

#### Zwraca

wektor par string string, którego pierwszym elementem pary jest miasto startowe, a drugim końcowe

### 2.4.1.4 vector\_z\_miastami()

```
vector<string> vector_z_miastami (
    string nazwa_pliku )
```

Pobiera miasta obecne w grafie.

#### Parametry

<i>nazwa_pliku</i>	nazwa pliku z drogami
--------------------	-----------------------

#### Zwraca

set z miastami obecnymi w grafie

### 2.4.1.5 zapisanie\_do\_grafu()

```
bool zapisanie_do_grafu (
    string nazwa_pliku,
    vector< string > miasta,
    int ** graf )
```

Funkcja zapisująca połączenia między miastami do grafu.

#### Parametry

<i>nazwa_pliku</i>	nazwa pliku tekstowego z miastami
<i>miasta</i>	vector miast obecnych w grafie
<i>graf</i>	Wygenerowany przez Dorygen graf prezentujący połączenia między miastami

**Zwraca**

true or false, w zaleznosci, czy plik jest pusty czy nie

**2.4.1.6 zwroc\_indeks()**

```
int zwroc_indeks (
    vector< string > miasta,
    string miasto )
```

Funkcja zwracajaca indeks miasta w wektorze 'miasta'.

**Parametry**

<i>miasta</i>	wektor z miastami obecnymi w grafie
<i>miasto</i>	miasto, ktorego indeks chcemy otrzymac

**Zwraca**

indeks miasta w wektorze 'miasta'

**2.4.1.7 zwroc\_najblizsze\_miasto()**

```
int zwroc_najblizsze_miasto (
    int * odleglosci,
    bool * sprawdzone,
    vector< string > miasta )
```

Funkcja zwracajaca indeks miasta z najnizsza odlegloscia.

**Parametry**

<i>odleglosci</i>	tablica zawierajaca odleglosci
<i>sprawdzone</i>	tablica bool zawierajaca wierzcholki (sprawdza czy sa odwiedzone czy tez nie)
<i>miasta</i>	vector miast (sluzy tutaj do wyznaczenia ilosci wierzcholkow w grafie)

**Zwraca**

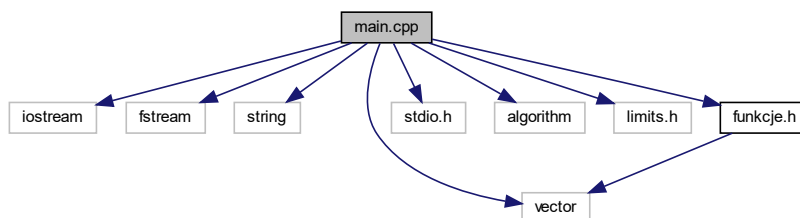
zwraca indeks miasta z najnizsza odlegloscia (waga)

**2.5 Dokumentacja pliku main.cpp**

```
#include <iostream>
#include <fstream>
```

```
#include <string>
#include <vector>
#include <stdio.h>
#include <algorithm>
#include <limits.h>
#include "funkcje.h"
```

Wykres zależności załączania dla main.cpp:



## Funkcje

- int `main` (int argc, char \*\*argv)

### 2.5.1 Dokumentacja funkcji

#### 2.5.1.1 main()

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

## 2.6 Dokumentacja pliku zapis.txt

## 2.7 Dokumentacja pliku zapisane.txt



# Indeks

- dijkstra
  - funkcje.cpp, [4](#)
  - funkcje.h, [8](#)
- drogi.txt, [3](#)
- DrogiDoWyznaczenia.txt, [3](#)
- drukuj\_sciezke
  - funkcje.cpp, [4](#)
  - funkcje.h, [8](#)
- funkcje.cpp, [3](#)
  - dijkstra, [4](#)
  - drukuj\_sciezke, [4](#)
  - pobranie\_tras, [5](#)
  - vector\_z\_miastami, [5](#)
  - zapisanie\_do\_grafu, [5](#)
  - zwroc\_indeks, [6](#)
  - zwroc\_najblizsze\_miasto, [6](#)
- funkcje.h, [7](#)
  - dijkstra, [8](#)
  - drukuj\_sciezke, [8](#)
  - pobranie\_tras, [9](#)
  - vector\_z\_miastami, [9](#)
  - zapisanie\_do\_grafu, [9](#)
  - zwroc\_indeks, [10](#)
  - zwroc\_najblizsze\_miasto, [10](#)
- main
  - main.cpp, [11](#)
- main.cpp, [10](#)
  - main, [11](#)
- pobranie\_tras
  - funkcje.cpp, [5](#)
  - funkcje.h, [9](#)
- vector\_z\_miastami
  - funkcje.cpp, [5](#)
  - funkcje.h, [9](#)
- zapis.txt, [11](#)
- zapisane.txt, [11](#)
- zapisanie\_do\_grafu
  - funkcje.cpp, [5](#)
  - funkcje.h, [9](#)
- zwroc\_indeks
  - funkcje.cpp, [6](#)
  - funkcje.h, [10](#)
- zwroc\_najblizsze\_miasto
  - funkcje.cpp, [6](#)
  - funkcje.h, [10](#)