Politechnika Śląska

Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Infrastruktura Drogowa

Autor:	Agnieszka Góral
Prowadzący:	dr inż. Paweł Foszner
Rok akademicki:	2020/2021
Kierunek:	informatyka
Rodzaj studiów:	SSI
Semestr:	1
Termin laboratorium:	poniedziałek 10:30 – 12:45 wtorek 12:00 – 14:14
Sekcja:	22
Termin oddania	2020-11-08

1 . Treść zadania:

Napisz program, który zaproponuje optymalny (najtańszy) sposób połączenia miast siecią drogową. Zaproponowana sieć drogowa musi umożliwić przemieszczanie się pomiędzy dowolnie wybranymi miastami. Połączenia między miastami są zapisane w pliku w następujący sposób:

<miasto 1> <miasto2> <cena>

W wyniki działania programu zostanie utworzony plik zawierający zaproponowaną strukturę sieci w formacie:

<miasto 1> <miasto 2> <cena>

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

-i plik wejściowy
-o plik wyjściowy

2 . Analiza zadania:

Zadanie przedstawia problem połączenia sieci miast, które pobieramy z pliku tekstowego siecią drogową. Szukamy takiego połączenia miast, które będzie najtańsze. Musimy pamiętać, że każde miasto nie musi być połączone z każdym, niektóre mogą być połączone za pośrednictwem innego z miast.

2.1 Struktury danych:

W programie wykorzystana została stworzona przeze mnie struktura „węzeł”, która przechowuje:

- nazwę miasta węzłowego (zmienną typu string);
- mapę połączeń (listę par, która przechowuje zmienne typu string oraz double), składającą się z nazwy miasta docelowego i ceny połączenia miasta węzłowego i danego miasta;
- zmienną typu bool, która używana jest podczas algorytmu do sprawdzania czy dane miasto węzłowe zostało już odwiedzone i czy zostały już sprawdzone połączenia z tym miastem;
- zmienną typu double, która znajduje zastosowanie przy użyciu algorytmu Dijkstry.

- Parę, która składa się ze zmiennej typu string oraz double, która będzie wykorzystywana podczas algorytmu i będzie potrzebna do przechowania najtańszego połączenie z danego węzła wraz z jego nazwą.

Struktura ta ilustruje jedno miasto węzłowe (część grafu) oraz przechowuje wszystkie miasta, z którymi możemy je połączyć oraz ceny tych połączeń.

2.2 Algorytmy:

Program szuka najtańszych połączeń pomiędzy węzłami za pomocą zmodyfikowanego algorytmu Dijkstry. Najpierw algorytm wszystkim cenom połączeń przyporządkowuje wartkość maksymalną dla danego typu zmiennej (w tym przypadku double). Następnie szuka, czy istnieje jakieś tańsze połączenie do węzła (miasta). Tańsze połączenia są zapamiętywane przez węzeł. Później sprawdza, czy miasto zostało odwiedzone i sprawdza najtańsze połączenia z nim. Algorytm wykona się dla każdego węzła i w ten sposób znajdzie najtańsze połączenia.

3 . Specyfikacja zewnętrzna:

Program jest uruchamiany z linii poleceń, za pomocą następujących przełączników:

-i nazwa_pliku_wejściowego.txt
-o nazwa_pliku_wyjściowego.txt

Użytkownik musi pamiętać, że pliki muszą posiadać rozszerzenie „.txt”. Jeśli użytkownik poda błędne parametry lub program nie będzie w stanie otworzyć pliku to w konsoli pojawi się komunikat informujący o błędzie oraz dokładna instrukcja wprowadzania danych.

4 . Specyfikacja wewnętrzna:

Program został zrealizowany zgodnie z paradygmatem strukturalnym. W programie rozdzielono interfejs (komunikacji z użytkownikiem) od logiki aplikacji.

4.1 Ogólna struktura programu

Program za pomocą funkcji *czytanie_z_wiersza_polecen* pobiera parametry, które zostały podane przez użytkownika i sprawdza ich poprawność. Gdy wprowadzone parametry są niepoprawne to wyświetlany jest komunikat o błędzie, wraz z instrukcją obsługi parametrów. Następnie, jeśli parametry zostały wczytane poprawnie, czyli jeśli funkcja *czytanie_z_wiersza_polecen* zwróciła wartość *true* to funkcja *zczytaj_z_pliku* pobiera dane z pliku, który użytkownik wskazał za pomocą parametrów i umieszcza je w strukturze (po zakończonej operacji zamyka plik), funkcja ta zwraca wektor węzłów (listę miast wraz z ich możliwymi połączeniami). Następnie lista miast jest przekazywana to funkcji *algorytm* (która wykorzystuje funkcje: *przygotuj_dane*, *relaksacja_krawedzi* oraz *znajdz_najblizsze_miasto*). W tej funkcji program sprawdza po kolei miasta i szuka najtańszych połączeń pomiędzy nimi. Jako ostania jest wywoływana fukcja *zwróć_do_pliku*, która zapisuje wynik działania

programu do pliku, a następnie go zamyka. Przed zakończeniem działania programu pamięć jest zwalniana aby zapobiec wyciekom pamięci.

4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

5 . Testowanie

Program został przetestowany na różnych plikach wejściowych. Włącznie z plikami, w których nie wszystkie dane są prawidłowe (część linijek jest pusta, w pliku znajdują się losowe znaki lub brakuje jakiejś danej). W takim przypadku program zlicza ilość linijek pustych oraz tych zapisanych w niepoprawny sposób, a następnie po odczytaniu pozostałych linijek (tych, w których dane są zapisane w sposób poprawny, jeśli takie istnieją) wyświetla komunikat o stanie pliku wejściowego np.:

„|| Liczba wszystkich linijek: 30 || Liczba pustych linijek: 5 || Liczba niepoprawnie sformatowanych linijek: 2 ||”

Podczas testowania został wyeliminowany przypadek, gdy w pliku wejściowym znajduje się graf rozłączny (np. w pliku znajdują się dwie grupy miast ale nie są one ze sobą w żaden sposób połączone) oraz dodano opcję wyświetlania komunikatu błędu, gdy użytkownik próbuje podać dwa razy ten sam parametr lub wprowadza niepoprawne parametry.

Program został przetestowany pod kątem wycieków pamięci. Pomimo dużych rozmiarów plików na jakich program był testowany, nie pojawił się komunikat o naruszeniu ochrony pamięci.

6 . Wnioski :

Zadane zadanie zostało zrealizowane w całości. Najtrudniejszym etapem zadania okazało się ułożenie algorytmu oraz stworzenie struktury, która miała by obrazować połączenia. Podczas tworzenia programu poszerzyłam swoją wiedzę z zakresu tworzenia struktur oraz funkcji, gdyż korzystałam w tym celu z wielu zewnętrznych źródeł.

Literatura:

- „C++ Przewodnik Dla Początkujących” Alex Allain
- „Podstawy Programowania w Języku C++” Józef Zieliński

Drogi

Wygenerowano przez Doxygen 1.8.20

Rozdział 1

Indeks klas

1.1 Lista klas

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

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury wezel

`#include <struktury.h>`

Atrybuty publiczne

- std::string nazwa
- std::map< wezel *, double > polaczenia
- double cena_od_miasta_poczatkowego
- bool odwiedzono = false
- std::pair< wezel *, double > najtansze_polaczenie_z_wezla

3.1.1 Opis szczegółowy

Struktora, która ilustuje miasto węzłowe (pojednyńcze miasto) i wszystkie możliwe połączenia z nim oraz ceny.

Struktura przechowuje nazwę miasta węzłowego i wszystkie możliwe połączenia w postaci mapy (listy par) miast, z którymi miasto węzłowe jest połączone oraz ceny tych połączeń). Struktura ta później jest wykorzystywana przy algorytmie.

3.1.2 Dokumentacja atrybutów składowych

3.1.2.1 cena_od_miasta_poczatkowego

`double wezel::cena_od_miasta_poczatkowego`

Zmienna, która jest wykorzystywana przy algorytmie, jest to cena od miasta do węzła

3.1.2.2 najtansze_polaczenie_z_wezla

```
std::pair<wezel*, double> wezel::najtansze_polaczenie_z_wezla
```

Zmienna, która przechowuje parę, która przechowuje najtańsze połączenie

3.1.2.3 nazwa

```
std::string wezel::nazwa
```

Nazwa miasta węzłowego

3.1.2.4 odwiedzono

```
bool wezel::odwiedzono = false
```

Zmienna, która służy do sprawdzenia czy dane miasto zostało już odwiedzone

3.1.2.5 polaczenia

```
std::map<wezel*, double> wezel::polaczenia
```

Mapa, która przechowuje listę par, w tym przypadku para to nazwa miasta, z którym miasto węzłowe jest połączone oraz cena połączenia tych miast

Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku funkcje.cpp

```
#include "funkcje.h"
#include "struktury.h"
#include <limits.h>
```

Funkcje

- bool czytanie_z_wiersza_polecen (int arg, char ∗argv[ ], string &nazwa_pliku_wejsciowego, string &nazwa↩ _pliku_wyjsciowego)
- wezel ∗ zwroc_miasto (string &nazwa_szukanego_miasta, vector< wezel ∗ > &miasta_wezlowe)
- vector< wezel ∗ > zczytaj_z_pliku (string &nazwa)
- void przygotuj_dane (vector< wezel ∗ > &lista_miast)
- void relaksacja_krawedzi (wezel ∗miasto, wezel ∗miasto_poczotkowe)
- wezel ∗ znajdz_najblizsze_miasto (wezel ∗obecny_wezel, vector< wezel ∗ > &lista_miast)
- void algorytm (vector< wezel ∗ > &lista_miast, wezel ∗miasto_poczatkowe)
- void zwroc_do_pliku (vector< wezel ∗ > &lista_miast, string &nazwa_pliku_wyjsciowego)

4.1.1 Dokumentacja funkcji

4.1.1.1 algorytm()

```
void algorytm (
            vector< wezel * > & lista_miast,
            wezel * miasto_poczatkowe )
```

Funkcja wykorzystująca algorytm Dijkstry.

Wykorzystuje funkcję przygotuj_dane(), relaksacja_krawedzi() oraz znajdz_najblizsze_miasto(). Algorytm działa, dla każego miasta i jak wykryje mniejszą cenę to zapisuje z jakeigo połączenia ona pochodzi.

Parametry

lista_miast	zmienna typu vector<wezel*>, która zawiera listę miast
miasto_poczatkowe	zmienna typu wskaźnik na wezel

Zwraca

zwraca miasto, z którym połączenie jest najtańsze.

4.1.1.2 czytanie_z_wiersza_polecen()

```
bool czytanie_z_wiersza_polecen (
            int arg,
            char * argv[],
            string & nazwa_pliku_wejsciowego,
            string & nazwa_pliku_wyjsciowego )
```

Funkcja, która zajmuje się zczytywaniem parametrów z wiersza poleceń.

Funkcja sprawdza poprawność podanych parametrów i zwraca prawdę (jeśli są poprawne) lub fałsz (jeśli są błędne)

Parametry

arg	ilość argumentów przekazanych
argv[]	wskaźnik na tablicę parametrów
nazwa_pliku_wejsciowego	do funkcji przekazujemy nazwę pliku tekstowego (wejściowego), zmienna typu string
nazwa_pliku_wyjsciowego	do funkcji przekazujemy nazwę pliku tekstowego (wyjściowego), zmienna typu string

Zwraca

jeźli miasto już znajduje się na liście to zwraca nullptr, jeśli miasto jeszcze nie znajduje się na liście to zwraca jego nazwę

4.1.1.3 przygotuj_dane()

```
void przygotuj_dane (
            vector< wezel * > & lista_miast )
```

Funkcja, która przygotowywuje dane, aby później można było wykorzystać je w algorytmie.

Funkcja ceny (cena_od_misata_początkowego) ustawia na nieskończoność i zmienia status odwiedzenia na false. Ustawia im wartość maksymalną dla double.

Parametry

lista_miast	zmienna typu vector<wezel*>, która zawiera listę miast

4.1.1.4 relaksacja_krawedzi()

```
void relaksacja_krawedzi (
            wezel * miasto,
            wezel * miasto_poczotkowe )
```

Funkcja służąca do szukania czy istnieje jakieś tańsze połączenie do węzła. Tańsze połączenia są zapamiętywane przez węzeł.

Funkcja sprawdza ceny połączeń z węzła i jeżli znalazł krótszą trase to wpisuje tą trase jako połączenie.

Parametry

miasto	zmienna typu wskaźnik na wezel, przechowuje nawę miasta
miasto_poczatkowe	zmienna typu wskaźnik na wezel, przechowuje nawę miasta poczatkowego

4.1.1.5 zczytaj_z_pliku()

```
vector<wezel *> zczytaj_z_pliku (
            string & nazwa )
```

Funkcja służąca do odczytania danych z pliku oraz umieszczenia ich w strukturze. Wykorzystuje funkcję zwroc_↩ miasto.

Funkcja zczytuje dane z pliku za pomocą stringstream po linijce i sprawdza poprawność danych w danej linijce, jeśli dane są poprawne to sprawdza czy dane miasta istnieją już w strukturze, jeśli nie to dla danego miasta tworzy nowy węzeł

Parametry

nazwa	to zmienna typu string przechowująca nazwę pliku wejściowego wraz z jego rozszerzeniem

Zwraca

zwraca zmienną typu vector<wezel*> (wektor węzłów), która przechowuje listę miast węzłowych i ich połączenia z innymi mistami.

4.1.1.6 znajdz_najblizsze_miasto()

```
wezel* znajdz_najblizsze_miasto (
            wezel * obecny_wezel,
            vector< wezel * > & lista_miast )
```

Funkcja, która sprawdza czy miasto zostało odwiedzone a następnie sprawdza najtańsze połącznia z nim.

Funkcja służąca do szukania najbliższego (najtańszego) miasta połączonego z węzłem lub najtańszego połącznia.

Parametry

obecny_wezel	zmienna typu wskaźnik na wezel, miasto które obecnie sprawdzamy
lista_miast	zmienna typu vector<wezel*>, która zawiera listę miast

Zwraca

zwraca miasto, z którym połączenie jest najtańsze.

4.1.1.7 zwroc_do_pliku()

```
void zwroc_do_pliku (
            vector< wezel * > & lista_miast,
            string & nazwa_pliku_wyjsciowego )
```

Funkcja, która zwraca dane do pliku wyściowego.

Parametry

lista_miast	zmienna typu vector<wezel*>, która zawiera listę miast
nazwa_pliku_wyjsciowego	zmienna typu string, która przechowuje nazwę pliku wyjściowego

4.1.1.8 zwroc_miasto()

```
wezel* zwroc_miasto (
            string & nazwa_szukanego_miasta,
            vector< wezel * > & miasta_wezlowe )
```

Funkcja służąca do sprawdzenia, czy zczytane z pliku miasto istnieje już na liście miast.

Jeżli miasto istnieje już na liście to funkcja zwraca pusty wskaźnik, jeśli miasto jeszcze nie istnieje na liście miast to dopisuje je do listy.

Parametry

nazwa_szukanego_miasta	to zmienna typu string, która przechwuje nazwę miasta, które zczytaliśmy z pliku i przekazujemy do funkcji
miasta_wezlowe	to zmienna typu vector<wezel*> (wektor wskaźników na węzeł), która zawiera listę miast węzłowych, które zostały już zczytane z pliku

Zwraca

jeśli miasto już znajduje się na liście to zwraca jego nazwę, jeśli miasto jeszcze nie znajduje się na liście to zwraca nullptr

4.2 Dokumentacja pliku funkcje.h

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

Funkcje

- bool czytanie_z_wiersza_polecen (int arg, char *argv[ ], string &nazwa_pliku_wejsciowego, string &nazwa↩ _pliku_wyjsciowego)
- wezel * zwroc_miasto (string &nazwa_szukanego_miasta, vector< wezel * > &miasta_wezlowe)
- vector< wezel * > zczytaj_z_pliku (string &nazwa)
- void przygotuj_dane (vector< wezel * > &lista_miast)
- void relaksacja_krawedzi (wezel *miasto, wezel *miasto_poczotkowe)
- wezel * znajdz_najblizsze_miasto (wezel *obecny_wezel, vector< wezel * > &lista_miast)
- void algorytm (vector< wezel * > &lista_miast, wezel *miasto_poczatkowe)
- void zwroc_do_pliku (vector< wezel * > &lista_miast, string &nazwa_pliku_wyjsciowego)

4.2.1 Dokumentacja funkcji

4.2.1.1 algorytm()

```
void algorytm (
            vector< wezel * > & lista_miast,
            wezel * miasto_poczatkowe )
```

Funkcja wykorzystująca algorytm Dijkstry.

Wykorzystuje funkcję przygotuj_dane(), relaksacja_krawedzi() oraz znajdz_najblizsze_miasto(). Algorytm działa, dla każego miasta i jak wykryje mniejszą cenę to zapisuje z jakeigo połączenia ona pochodzi.

Parametry

lista_miast	zmienna typu vector<wezel*>, która zawiera listę miast
miasto_poczatkowe	zmienna typu wskaźnik na wezel

Zwraca

zwraca miasto, z którym połączenie jest najtańsze.

4.2.1.2 czytanie_z_wiersza_polecen()

```
bool czytanie_z_wiersza_polecen (
            int arg,
            char * argv[],
            string & nazwa_pliku_wejsciowego,
            string & nazwa_pliku_wyjsciowego )
```

Funkcja, która zajmuje się zczytywaniem parametrów z wiersza poleceń.

Funkcja sprawdza poprawność podanych parametrów i zwraca prawdę (jeśli są poprawne) lub fałsz (jeśli są błędne)

Parametry

arg	ilość argumentów przekazanych
argv[]	wskaźnik na tablicę parametrów
nazwa_pliku_wejsciowego	do funkcji przekazujemy nazwę pliku tekstowego (wejściowego), zmienna typu string
nazwa_pliku_wyjsciowego	do funkcji przekazujemy nazwę pliku tekstowego (wyjściowego), zmienna typu string

Zwraca

jeźli miasto już znajduje się na liście to zwraca nullptr, jeśli miasto jeszcze nie znajduje się na liście to zwraca jego nazwę

4.2.1.3 przygotuj_dane()

```
void przygotuj_dane (
            vector< wezel * > & lista_miast )
```

Funkcja, która przygotowywuje dane, aby później można było wykorzystać je w algorytmie.

Funkcja ceny (cena_od_misata_początkowego) ustawia na nieskończoność i zmienia status odwiedzenia na false. Ustawia im wartość maksymalną dla double.

Parametry

lista_miast	zmienna typu vector<wezel*>, która zawiera listę miast

4.2.1.4 relaksacja_krawedzi()

```
void relaksacja_krawedzi (
            wezel * miasto,
            wezel * miasto_poczotkowe )
```

Funkcja służąca do szukania czy istnieje jakieś tańsze połączenie do węzła. Tańsze połączenia są zapamiętywane przez węzeł.

Funkcja sprawdza ceny połączeń z węzła i jeżli znalazł krótszą trase to wpisuje tą trase jako połączenie.

Parametry

miasto	zmienna typu wskaźnik na wezel, przechowuje nawę miasta
miasto_poczatkowe	zmienna typu wskaźnik na wezel, przechowuje nawę miasta poczatkowego

4.2.1.5 zczytaj_z_pliku()

```
vector<wezel *> zczytaj_z_pliku (
            string & nazwa )
```

Funkcja służąca do odczytania danych z pliku oraz umieszczenia ich w strukturze. Wykorzystuje funkcję zwroc_↩ miasto.

Funkcja zczytuje dane z pliku za pomocą stringstream po linijce i sprawdza poprawność danych w danej linijce, jeśli dane są poprawne to sprawdza czy dane miasta istnieją już w strukturze, jeśli nie to dla danego miasta tworzy nowy węzeł

Parametry

nazwa	to zmienna typu string przechowująca nazwę pliku wejściowego wraz z jego rozszerzeniem

Zwraca

zwraca zmienną typu vector<wezel*> (wektor węzłów), która przechowuje listę miast węzłowych i ich połączenia z innymi mistami.

4.2.1.6 znajdz_najblizsze_miasto()

```
wezel* znajdz_najblizsze_miasto (
            wezel * obecny_wezel,
            vector< wezel * > & lista_miast )
```

Funkcja, która sprawdza czy miasto zostało odwiedzone a następnie sprawdza najtańsze połącznia z nim.

Funkcja służąca do szukania najbliższego (najtańszego) miasta połączonego z węzłem lub najtańszego połącznia.

Parametry

obecny_wezel	zmienna typu wskaźnik na wezel, miasto które obecnie sprawdzamy
lista_miast	zmienna typu vector<wezel*>, która zawiera listę miast

Zwraca

zwraca miasto, z którym połączenie jest najtańsze.

4.2.1.7 zwroc_do_pliku()

```
void zwroc_do_pliku (
            vector< wezel * > & lista_miast,
            string & nazwa_pliku_wyjsciowego )
```

Funkcja, która zwraca dane do pliku wyściowego.

Parametry

lista_miast	zmienna typu vector<wezel*>, która zawiera listę miast
nazwa_pliku_wyjsciowego	zmienna typu string, która przechowuje nazwę pliku wyjściowego

4.2.1.8 zwroc_miasto()

```
wezel* zwroc_miasto (
            string & nazwa_szukanego_miasta,
            vector< wezel * > & miasta_wezlowe )
```

Funkcja służąca do sprawdzenia, czy zczytane z pliku miasto istnieje już na liście miast.

Jeżli miasto istnieje już na liście to funkcja zwraca pusty wskaźnik, jeśli miasto jeszcze nie istnieje na liście miast to dopisuje je do listy.

Parametry

nazwa_szukanego_miasta	to zmienna typu string, która przechwuje nazwę miasta, które zczytaliśmy z pliku i przekazujemy do funkcji
miasta_wezlowe	to zmienna typu vector<wezel*> (wektor wskaźników na węzeł), która zawiera listę miast węzłowych, które zostały już zczytane z pliku

Zwraca

jeśli miasto już znajduje się na liście to zwraca jego nazwę, jeśli miasto jeszcze nie znajduje się na liście to zwraca nullptr

4.3 Dokumentacja pliku infastruktura_drog.cpp

```
#include <iostream>
#include <vector>
#include "funkcje.h"
#include "struktury.h"
```

Funkcje

- int main (int arg, char *argv[ ])

4.3.1 Dokumentacja funkcji

4.3.1.1 main()

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

4.4 Dokumentacja pliku struktury.h

```
#include <string>
#include <iostream>
#include <vector>
#include <map>
#include <limits.h>
```

Komponenty

- struct wezel

Indeks