Program MAPA

Bartosz Siwiaszczyk
Teleinformatyka semestr I 2022/2023
1.02.2023 r.

Program MAPA

Wprowadzenie

Program MAPA, to program umożliwiający znalezienie najkrótszej trasy między dwoma miastami. Drogi między tymi miastami połączone są jednokierunkowymi drogami, o ustalonej długości. Plik zawierający trasy oraz trasy do wyznaczenia ma ustaloną postać.

Wynikiem działania programu jest plik z wyznaczonymi trasami - podana jest nazwa trasy, jej długość, oraz kolejne jej odcinki.

Uruchamianie

Program uruchamiany jest z poziomu wiersza poleceń. Do uruchomienia wymaga trzech argumentów:

- -d (plik wejściowy z drogami)
- -t (plik wejściowy z trasami do wyznaczenia)
- -o (plik wynikowy z wyznaczonymi trasami)

Funkcje programu

- Powiadamianie o nieprawidłowym użyciu programu
- Wyznaczanie najkrótszej możliwej trasy między miastami
- Informowanie o niemożliwości wyznaczenia trasy
-

Dokumentacja klas

Dokumentacja struktury Road

`#include <mapa.h>`

Atrybuty publiczne

- string **city1**
- string **city2**
- int **distance**

Opis szczegółowy

Struktura zawierająca dane o drogach.

Parametry

city1	Zawiera nazwę początkowego miasta
city2	Zawiera nazwę nazwę miasta końcowego
distance	Zawiera odległość między miastem początkowym, a końcowym

Definicja w linii **15** pliku **mapa.h**.

Dokumentacja atrybutów składowych

string Road::city1

Definicja w linii **16** pliku **mapa.h**.

string Road::city2

Definicja w linii **17** pliku **mapa.h**.

int Road::distance

Definicja w linii **18** pliku **mapa.h**.

Dokumentacja plików

`#include "mapa.h"`

Funkcje

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

Dokumentacja funkcji

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

Funkcja main zawierająca:

- argumenty uruchamiania
- wywołanie funkcji readRoads, readRoutes, shortestPath oraz writeResults
- unordered_map zawierająca informacje o trasach
-
- **Parametry**

argc	liczba przełączników - gdy równa 1, to program jest uruchomiony bez przełączników
argv	argumenty wiersza polecen

Definicja w linii **37** pliku **Mapa.cpp**.

Mapa.cpp

```
00001
00027 #include "mapa.h"
00028
00037 int main(int argc, char* argv[]) {
00038     cout << "=== Program MAPA ===" << endl;
00039     string roadsFilename, routesFilename, outputFilename;
00040     if (argc == 1)
00041         cout << "Wprowadz w argumentach programu lokalizacje plikow zawierajace: -d
(plik z drogami) -t (plik z trasami) -o (plik wyjsciowy)" << endl << endl;
00042     else {
00043         for (int i = 1; i < argc; i++) {
00044             if (string(argv[i]) == "-d") {
00045                 roadsFilename = argv[i + 1];
00046             }
00047             else if (string(argv[i]) == "-t") {
00048                 routesFilename = argv[i + 1];
00049             }
00050             else if (string(argv[i]) == "-o") {
00051                 outputFilename = argv[i + 1];
00052             }
00053         }
00054     }
00055
00056     /* Wczytaj drogi i trasy z pliku */
00057     auto roads = readRoads(roadsFilename);
00058     auto routes = readRoutes(routesFilename);
00059
00060     unordered_map <string, vector <Road>> cityToRoads;
00061     for (auto road : roads) {
00062         cityToRoads[road.city1].push_back(road);
00063     }
00064
00065     /* Sprawdz trasy, a nastepnie zapisz wyniki do pliku */
00066     vector<pair<pair<string, string>, vector<Road>>> results =
shortestPath(routes, cityToRoads);
00067     writeResults(results, outputFilename);
00068
00069     return 0;
00070 }
```

Dokumentacja pliku mapa.h

```
#include <iostream>
#include <fstream>
#include <unordered_map>
#include <vector>
#include <queue>
#include <numeric>
```

Komponenty

- struct **Road**

Funkcje

- vector< **Road** > **readRoads** (string filename)
- vector< pair< string, string > > **readRoutes** (string filename)
- void **writeResults** (vector< pair< pair< string, string >, vector< **Road** > > > results, string filename)
- vector< pair< pair< string, string >, vector< **Road** > > > **shortestPath** (vector< pair< string, string > > routes, unordered_map< string, vector< **Road** > > cityToRoads)

Dokumentacja funkcji

vector< Road > readRoads (string *filename*)

Funkcja służąca do odczytywania dróg z pliku.

Parametry

filename	Zawiera nazwę pliku w którym znajdują się drogi.

Definicja w linii **25** pliku **mapa.h**.

vector< pair< string, string > > readRoutes (string *filename*)

Funkcja służąca do odczytywania tras z pliku.

Parametry

filename	Zawiera nazwę pliku w którym znajdują się trasy.

Definicja w linii **46** pliku **mapa.h**.

vector< pair< pair< string, string >, vector< Road > > > shortestPath (vector< pair< string, string > > *routes*, unordered_map< string, vector< Road > > *cityToRoads*)

Funkcja oparta na algorytmie Dijkstry służąca do znajdowania najkrótszej trasy między dwoma miastami.

Parametry

routes	Zawiera wektor pary stringów z trasami (nazwami dwóch miast).
cityToRoads	Zawiera typ unordered_map z informacjami o trasach.

Definicja w linii **95** pliku **mapa.h**.

void writeResults (vector< pair< pair< string, string >, vector< Road > > > *results*, string *filename*)

Funkcja służąca do zapisywania wyniku działania programu do pliku. Jeżeli program nie znalazł trasy pomiędzy dwoma miastami funkcja zwraca o tym informację.

Parametry

results	Zawiera docelowo wektor wyniku działania funkcji shortestPath.
filename	Zawiera nazwę pliku do którego będzie zapisywany wynik działania programu.

Definicja w linii **68** pliku **mapa.h**.

```
#include <iostream>
00002 #include <fstream>
00003 #include <unordered_map>
00004 #include <vector>
00005 #include <queue>
00006 #include <numeric>
00007 using namespace std;
00008
00015 struct Road {
00016     string city1;
00017     string city2;
00018     int distance;
00019 };
00020
00025 vector<Road> readRoads(string filename) {
00026     vector<Road> roads;
00027     ifstream input(filename);
00028     if (!input.is_open()) {
00029         cerr << "Nie udalo sie otworzyc pliku z drogami." << endl;
00030         return roads;
00031     }
00032
00033     string city1, city2;
00034     int distance;
00035     while (input >> city1 >> city2 >> distance) {
00036         roads.push_back({ city1, city2, distance });
00037     }
00038
00039     return roads;
00040 }
00041
00046 vector<pair<string, string>> readRoutes(string filename) {
00047     vector<pair<string, string>> routes;
00048     ifstream input(filename);
00049     if (!input.is_open()) {
00050         cerr << "Nie udalo sie otworzyc pliku z trasami." << endl;
00051         return routes;
00052     }
00053
00054     string city1, city2;
00055     while (input >> city1 >> city2) {
00056         routes.push_back({ city1, city2 });
00057     }
00058
00059     return routes;
00060 }
00061
00068 void writeResults(vector<pair<pair<string, string>, vector<Road>>> results, string
filename) {
00069     ofstream output(filename);
00070     if (!output.is_open()) {
00071         cerr << "Nie udalo sie otworzyc pliku wyjsciowego." << endl;
00072         return;
00073     }
00074     for (auto result : results) {
00075         auto route = result.first;
00076         auto roads = result.second;
00077         if (roads.empty()) {
00078             output << "Nie znaleziono trasy dla trasy: " << route.first << " --> "
<< route.second << endl;
00079         }
00080         else {
00081             output << "Trasa: " << route.first << " --> " << route.second << " ("
<< accumulate(roads.begin(), roads.end(), 0, [](int sum, Road road) { return sum +
road.distance; }) << " km):" << endl;
00082             for (auto road : result.second) {
```

```
00083                 output << road.city1 << " ---> " << road.city2 << " ---> " <<
road.distance << "km" << endl;
00084             }
00085         }
00086     }
00087 }
00088
00089
00095 vector<pair<pair<string, string>, vector<Road>>> shortestPath(vector<pair<string,
string>> routes, unordered_map<string, vector<Road>> cityToRoads) {
00096     vector<pair<pair<string, string>, vector<Road>>> results;
00097     for (auto route : routes) {
00098         unordered_map<string, int> distance;
00099         unordered_map<string, string> predecessor;
00100         priority_queue<pair<int, string>, vector<pair<int, string>>,
greater<pair<int, string>>> queue;
00101
00102         distance[route.first] = 0;
00103         queue.push({ 0, route.first });
00104
00105         // Dopóki kolejka nie jest pusta:
00106         while (!queue.empty()) {
00107             auto top = queue.top();
00108             queue.pop();
00109
00110
00111             auto city = top.second;
00112
00113             for (auto road : cityToRoads[city]) {
00114                 // Jeśli dotarliśmy do końcowego miasta, zapamiętaj wynik
00115                 if (road.city2 == route.second) {
00116                     distance[route.second] = distance[city] + road.distance;
00117                     predecessor[route.second] = city;
00118                     break;
00119                 }
00120
00121                 // Jeśli jeszcze nie odwiedziliśmy danego miasta:
00122                 if (distance.count(road.city2) == 0) {
00123                     distance[road.city2] = distance[city] + road.distance;
00124                     predecessor[road.city2] = city;
00125                     queue.push({ distance[road.city2], road.city2 });
00126                 }
00127             }
00128         }
00129
00130         // Jeśli udało się znaleźć trasę do końcowego miasta:
00131         if (predecessor.count(route.second) > 0) {
00132             vector<Road> result;
00133             auto currentCity = route.second;
00134             while (currentCity != route.first) {
00135                 result.push_back({ predecessor[currentCity], currentCity,
distance[currentCity] - distance[predecessor[currentCity]] });
00136                 currentCity = predecessor[currentCity];
00137             }
00138             reverse(result.begin(), result.end());
00139             results.push_back({ route, result });
00140         }
00141         else {
00142             results.push_back({ route, {}});
00143         }
00144     }
00145     return results;
00146 }
00147
```