Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Połączenia autobusowe

---

autor	Szymon Skoczylas
prowadzący	dr inż. Bernard Wyrwoł
rok akademicki	2021/2022
kierunek	informatyka
semestr	1
termin laboratorium	wtorek 11:30 - 13:00
sekcja	14
termin oddania sprawozdania	2022-01-25

1 Treść zadania

Napisać program, który umożliwia znalezienie połączeń autobusowych między dwoma wybranymi miastami (dowolna liczba przesiadek, minimalny czas na przesiadkę 5min.), by czas przejazdu był jak najkrótszy. Miasta połączone są trasami (jednokierunkowe) przy czym dla każdej trasy podawany jest średni czas przejazdu (w minutach) oraz czasy odjazdów autobusów. Plik z danymi ma następującą postać (w każdej linii podana jest jedna trasa): miasto początkowe miasto końcowe średni czas przejazdu godziny odjazdów. Przykładowy plik z trasami (dla kilku wybranych tras)

Katowice Krakow 70 10:15 14:20 16:25
Krakow Tarnow 80 11:20 13:30 18:00
Tarnow Jaslo 60 12:30 15:00 20:00
Katowice Gliwice 30 9:15 12:30 15:45
Lodz Poznan 160 7:00 12:10 15:15
Gliwice Katowice 30 8:30 12:00 17:00
Katowice Czestochowa 70 9:50 14:00 20:00
Czestochowa Lodz 100 7:25 12:35 17:20
Lodz Torun 130 9:30 14:20 21:00
Krakow Katowice 70 7:30 11:00 14:25
Gliwice Wroclaw 130 8:00 14:00 18:00

Drugim plikiem wejściowym jest plik z trasami do wyznaczenia. Każda linia pliku zawiera jedną trasę w postaci: miasto początkowe miasto końcowe (podobnie jak w pliku z podanymi trasami, ale bez określania pozostałych parametrów). Wynikiem działania programu jest plik wyjściowy z wyznaczonymi trasami, tzn. podana jest nazwa trasy, całkowity czas podróży, liczba przesiadek, a potem szczegółowe dane dotyczące podróży, np.
Katowice --> Torun (czas: 7:15, przesiadki: 2)

Katowice (9:15) --> Czestochowa (11:00), przesiadka za 1:35
Czestochowa (12:35) --> Lodz (14:15), przesiadka za 0:05
Lodz (14:20) --> Torun (16:30)

W przypadku nie znalezienia połączenia w pliku zapisywany jest komunikat brak polaczenia.

2 Analiza zadania

Zagadnienie przedstawia problem znalezienia najszybszego połączenia “z miasta A do miasta B” z podanych możliwych przejazdów między miastami. Przejazdy są jednokierunkowe, dlatego program musi znaleźć wszystkie możliwe połączenia oraz sprawdzić, które z nich jest najszybsze.

2.1 Struktury danych

W programie wykorzystano zostały listy klas do przechowywania wartości otrzymanych z pliku. Pojedyncza linia z pliku została przechowana w klasie, a cała

Atrybuty prywatne

std::string	**m_departure** zmienna przechowująca informację o miecie, z którego odjeżdzamy Więcej...
std::string	**m_arrival** zmienna przechowująca informację o mieście docelowego Więcej...
int	**m_duration** {} zmiennna przechowująca informację o czasie jazdy z miasta, z którego odjeżdzamy do miasta docelowego Więcej...
std::array< std::string, 3 >	**m_time** zmienna przechowująca informację o godzinach, o których możemy odjechać z tego przystanku Więcej...

Następnie na podstawie tych danych została stworzona lista klasy przechowującej informacje o miastach oraz ich wszystkich możliwych przejazdach jednokierunkowych.

Atrybuty publiczne

std::string	**m_name** nazwa miasta Więcej...
std::vector< std::string >	**m_canTravelTo** miasta, do których możemy z tego miasta dojechać

W dalszej części programu wykorzystana jest lista łańcuchów znaków do przechowania informacji o najszybszym możliwym połączeniu (ta informacja zostaje przechowana w liście zawierającej nazwy wszystkich miast przez które będziemy przejeżdżać).

2.2 Algorytmy

Program wykorzystując informację przechowane w liście interpretującej połączenia między miastami wyszukuje rekurencyjnie wszystkie możliwe połączenia. Algorytm sprawdza, czy z danego miasta jesteśmy w stanie dojechać do miasta docelowego następnie dodaje ten wynik do listy.

Następnie ze sporządzonej listy wszystkich połączeń program wyszukuje najszybszej możliwej drogi (biorąc pod uwagę zarówno średni czas przejazdu między miastem, a miastem oraz czas oczekiwania na kolejny odjazd).

Na samym końcu program sumuje łączny czas przejazdu najszybszej drogi oraz zapisuje w pliku wyjściowym informacje o przejeździe zgodnie z opisanym w treści zadania schemacie.

3. Specyfikacja zewnętrzna

Program jest wykonywany z pliku wykonywalnego. Program wykorzystuje pliki zewnętrzne (zakłada, że pliki są poprawne). Nazwy gotowych plików są zapisane w kodzie źródłowym.

4. Specyfikacja wewnętrzna

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

4.1 Ogólna struktura programu

W funkcji głównej deklarowane są pliki wejściowe oraz plik wyjściowy (łącznie ze ścieżkami do nich). Następnie deklarowane są zmienne potrzebne na rzecz działania programu. W kolejnych krokach program:

- interpretuje plik rozkładu jazdy i zapisuje go w liście klasy *FileLine* (lista *timetable*)
- interpretuje informacje z listy *timetable* i tworzy listę *cities* składającą się z kopii klasy *City* (informacja o miastach i ich połączeniach)
- wyszukuje wszystkich możliwych połączeń dla pożądanej trasy z przeczytanego pliku wejściowego przy użyciu funkcji *FindRoute*
- Sprawdza, czy trasa jest możliwa do wyznaczenia jeśli tak to program wykonuje zadania przedstawione poniżej, jeśli nie zapisuje w pliku wyjściowym informacje o braku takiej trasy

- Wyszukuje najszybszej trasy przy użyciu funkcji *FastestRoad* i liczy czas tej trasy przy użyciu funkcji *GetTotalTime*
- Zapisuje w pliku wyjściowym informacje o najszybszej trasie w pliku wyjściowym zgodnie z formatem podanym w treści zadania

4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

5. Testowanie

Program został przetestowanych dla różnych konfiguracji pliku wejściowego z zapisanymi drogami (rozkład jazdy). Program jest wadliwy ze względu na algorytm wyszukujący wszystkie połączenie danej trasy.

Podejrzenie:
Algorytm nie bierze pod uwagę wszystkich możliwych wariantów, jeśli napotka pierwsze miasto, które spełnia warunek, przez co część możliwości tras jest ignorowana i prowadzi do zakłamanych wariantów trasy, a czasem do **nie wyszukania trasy wcale**. Program wymaga wyłącznie naprawienia tego jednego algorytmu. Reszta programu działa zgodnie z oczekiwaniami. Poprawiona funkcja nie będzie skomplikowana do zaaplikowania w kodzie ze względu na blokowy podział programu na funkcje. Wystarczającym rozwiązaniem będzie napisanie funkcji, która znajdzie wszystkie możliwe warianty trasy i zapisaniem tej trasy w postaci listy łańcuchów znaków.

Program nie sprawdza poprawności plików wejściowych (pliki podawane są w kodzie; autor zakłada, że program został stworzony do pracy na poprawnych plikach).

Kod został napisany z myślą o wyciekach pamięci (zostały zastosowane inteligentne wskaźniki).

6. Wnioski

Program znajdujący najszybsze połączenia jest programem średnio skomplikowanym. Najbardziej wymagające okazało się napisanie algorytmu wyszukującego wszystkie możliwe warianty trasy “z miasta A do miasta B”. Przedstawiona funkcja rekurencyjna była wiele razy modyfikowana, została nawet napisana dodatkowa funkcja, która miała na celu skorygowanie błędnych wyników, lecz mimo to nie udało się uzyskać oczekiwanych rezultatów. Dla niektórych tras program działa poprawnie, niektórych tras wcale nie wyszukuje.

Bardzo ucząca była interpretacja podanych informacji z pliku i zapisanie ich na różne typy danych. Projekt pomógł w obyciu się z językiem programowania C++ (praca na projekcie podzielonym na plik źródłowy, plik nagłówkowy etc.), poznaniu możliwości nowych standardów języka (w tym przypadku użycie wiązania strukturalnego dla standardu c++17) oraz w pisaniu kodu w formie programowania obiektowego (wykorzystaniu klas, ich metod etc.).

Projekt PPK

Wygenerowano przez Doxygen 1.9.3

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 klasy City

Klasa zawierająca informację o nazwie miasta oraz wszystkich miast do których możemy z tego miasta dojechać

`#include <functions.h>`

Metody publiczne

- void SetName (const std::string &str)

  Funkcja ustawiająca nazwę miasta.
- void CanTravel (const std::string &str)

  Funkcja, która dopisuje do vectora zawierającego informację o miastach, do których możemy dojechać kolejne z miast.
- void ClearVec ()

  Funkcja usuwająca wartości w vectorze m_canTravelTo.

Atrybuty publiczne

- std::string m_name

  nazwa miasta
- std::vector< std::string > m_canTravelTo

  miasta, do których możemy z tego miasta dojechać

3.1.1 Opis szczegółowy

Klasa zawierająca informację o nazwie miasta oraz wszystkich miast do których możemy z tego miasta dojechać

Definicja w linii 75 pliku functions.h.

3.1.2 Dokumentacja funkcji składowych

3.1.2.1 CanTravel()

```
void City::CanTravel (
            const std::string & str )  [inline]
```

Funkcja, która dopisuje do vectora zawierającego informację o miastach, do których możemy dojechać kolejne z miast.

Parametry

str	zmienna zawierające nazwę miasta, do którego możemy dojechać

Definicja w linii 88 pliku functions.h.

3.1.2.2 ClearVec()

```
void City::ClearVec ( )  [inline]
```

Funkcja usuwająca wartości w vectorze m_canTravelTo.

Definicja w linii 91 pliku functions.h.

3.1.2.3 SetName()

```
void City::SetName (
            const std::string & str )  [inline]
```

Funkcja ustawiająca nazwę miasta.

Parametry

str	zmienna która zawiera nazwę miasta

Definicja w linii 84 pliku functions.h.

3.1.3 Dokumentacja atrybutów składowych

3.1.3.1 m_canTravelTo

```
std::vector<std::string> City::m_canTravelTo
```

miasta, do których możemy z tego miasta dojechać

Definicja w linii 79 pliku functions.h.

3.1.3.2 m_name

```
std::string City::m_name
```

nazwa miasta

Definicja w linii 78 pliku functions.h.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Project/headers/functions.h

3.2 Dokumentacja klasy FileLine

Klasa odpowiedzialna za przechowanie informacji pobranych z jednej linii z pliku rozkładu jazdy.

```
#include <functions.h>
```

Metody publiczne

- std::string GetCityName (const std::string &boolean) const

  Funkcja, która zwraca nazwę miasta (odjazdu albo docelowego) zależnie od podanej wartości parametru boolean.
- int GetDuration () const

  Funkcja zwracająca czas jazdy z miasta odjazdu do miasta docelowego.
- int GetNumberOfHoursOfDepartures () const

  Funkcja zwracająca informację o liczbie godzin, o których możemy rozpocząć podróż do miasta docelowego.
- int GetHour (short n) const

  Funkcja zwracająca jedną z godzin, o której możemy rozpocząć podróż
- void GetFileLine (const std::string &str)

  Funkcja która na podstawie linii z pliku rozkładu jazdy zapisze wartości do zmiennych klasy.

Metody prywatne

- int ConvertHour (const short &n) const

  Funkcja, która zwraca daną godzinę w postaci liczby minut w godzinie (np. 1:20 to 80)

Atrybuty prywatne

- std::string m_departure

  zmienna przechowująca informację o miecie, z którego odjeżdzamy
- std::string m_arrival

  zmienna przechowująca informację o mieście docelowego
- int m_duration {}

  zmiennna przechowująca informację o czasie jazdy z miasta, z którego odjeżdzamy do miasta docelowego
- std::array< std::string, 3 > m_time

  zmienna przechowująca informację o godzinach, o których możemy odjechać z tego przystanku

3.2.1 Opis szczegółowy

Klasa odpowiedzialna za przechowanie informacji pobranych z jednej linii z pliku rozkładu jazdy.

Definicja w linii 14 pliku functions.h.

3.2.2 Dokumentacja funkcji składowych

3.2.2.1 ConvertHour()

```
int FileLine::ConvertHour (
            const short & n ) const  [inline], [private]
```

Funkcja, która zwraca daną godzinę w postaci liczby minut w godzinie (np. 1:20 to 80)

Parametry

n	zmienna na podstawie której funkcja wybiera godzinę

Definicja w linii 53 pliku functions.h.

Oto graf wywoływań tej funkcji:

3.2.2.2 GetCityName()

```
std::string FileLine::GetCityName (
            const std::string & boolean ) const  [inline]
```

Funkcja, która zwraca nazwę miasta (odjazdu albo docelowego) zależnie od podanej wartości parametru boolean.

Parametry

boolean	zmienna na podstawie której funkcja zwróci jedno z dwóch miast

Zwraca

Funkcja zwraca string zawierający nazwę miasta

Definicja w linii 26 pliku functions.h.

3.2.2.3 GetDuration()

```
int FileLine::GetDuration ( ) const  [inline]
```

Funkcja zwracająca czas jazdy z miasta odjazdu do miasta docelowego.

Definicja w linii 33 pliku functions.h.

3.2.2.4 GetFileLine()

```
void FileLine::GetFileLine (
            const std::string & str )  [inline]
```

Funkcja która na podstawie linii z pliku rozkładu jazdy zapisze wartości do zmiennych klasy.

Parametry

str	String, który zawiera jedną linijkę z pliku rozkładu jazdy

Definicja w linii 44 pliku functions.h.

Oto graf wywoływań tej funkcji:

3.2.2.5 GetHour()

```
int FileLine::GetHour (
            short n ) const  [inline]
```

Funkcja zwracająca jedną z godzin, o której możemy rozpocząć podróż

Parametry

n	zmienna na podstawie której funkcja zwróci godzinę

Definicja w linii 40 pliku functions.h.

Oto graf wywołań dla tej funkcji:

3.2.2.6 GetNumberOfHoursOfDepartures()

```
int FileLine::GetNumberOfHoursOfDepartures ( ) const  [inline]
```

Funkcja zwracająca informację o liczbie godzin, o których możemy rozpocząć podróż do miasta docelowego.

Definicja w linii 36 pliku functions.h.

3.2.3 Dokumentacja atrybutów składowych

3.2.3.1 m_arrival

```
std::string FileLine::m_arrival  [private]
```

zmienna przechowująca informację o mieście docelowego

Definicja w linii 18 pliku functions.h.

3.2.3.2 m_departure

```
std::string FileLine::m_departure  [private]
```

zmienna przechowująca informację o miecie, z którego odjeżdzamy

Definicja w linii 17 pliku functions.h.

3.2.3.3 m_duration

```
int FileLine::m_duration {}  [private]
```

zmiennna przechowująca informację o czasie jazdy z miasta, z którego odjeżdzamy do miasta docelowego

Definicja w linii 19 pliku functions.h.

3.2.3.4 m_time

```
std::array<std::string,3> FileLine::m_time  [private]
```

zmienna przechowująca informację o godzinach, o których możemy odjechać z tego przystanku

Definicja w linii 20 pliku functions.h.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Project/headers/functions.h

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku Project/headers/functions.cpp

`#include "functions.h"`
Wykres zależności załączania dla functions.cpp:

4.2 functions.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "functions.h"
00002
00003 #define MINUMUM_TIME_FOR_CHANGE 5
00004
00005 std::vector<City> FillCityVec(std::vector<FileLine> & timetable)
00006 {
00007     std::vector<City> vec;
00008     std::string temp;
00009     bool cond = 0;
00010     std::unique_ptr<City> city = std::make_unique<City>();
00011     for (auto it = timetable.begin(); it != timetable.end(); it++)
00012     {
00013         temp = it->GetCityName("departure");
00014         for (auto jt = vec.begin(); jt != vec.end(); jt++)
00015         {
00016             if (jt->m_name == temp)
00017                 cond = 1;
00018         }
00019         if (cond == 0)
00020         {
00021             city->SetName(temp);
00022             for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
00023             {
00024                 if (jt->GetCityName("departure") == temp)
00025                     city->CanTravel(jt->GetCityName("arrival"));
00026             }
00027             vec.push_back(*city);
00028             city->ClearVec();
00029         }
00030         cond = 0;
00031     }
00032     for (auto it = timetable.begin(); it != timetable.end(); it++)
00033     {
00034         cond = 1;
00035         temp = it->GetCityName("arrival");
00036         for (auto jt = vec.begin(); jt != vec.end(); jt++)
00037         {
00038             if (jt->m_name == temp)
00039                 cond = 0;
00040         }
00041         if (cond)
00042         {
00043             city->SetName(temp);
```

```
00044             vec.push_back(*city);
00045         }
00046     }
00047     return vec;
00048 }
00049
00054 bool IfDuplicates(const std::vector<std::string>& vec, const std::string& city)
00055 {
00056     std::string str;
00057     for (auto it = vec.begin(); it != vec.end(); it++)
00058     {
00059         str = *it;
00060         if (str == city) return true;
00061     }
00062     return false;
00063 }
00064
00065 void Correction(std::vector<std::vector<std::string>> & roads, const std::string & goal)
00066 {
00067     for (int i=0;i<roads.size();i++)
00068     {
00069         std::reverse(roads[i].begin(), roads[i].end());
00070         if (roads[i].back() != goal)
00071             roads[i].clear();
00072     }
00073
00074     auto end = roads.end();
00075     for (auto it = roads.begin(); it != end; it++)
00076         end = std::remove(it + 1, end, *it);
00077     roads.erase(end, roads.end());
00078
00079     auto it = roads.begin();
00080     for (int i = 0; i < roads.size(); i++)
00081     {
00082         if (roads[i].empty() == true)
00083         {
00084             it += i;
00085             roads.erase(it);
00086         }
00087     }
00088 }
00089 void FindRoute(std::string goal, const std::string & from,const std::vector<City>& cities,
      std::vector<std::string>& solution, std::vector<std::vector<std::string>>& solutions)
00090 {
00091     if (goal == from)
00092     {
00093         solution.push_back(from);
00094         if (!solution.empty()) solutions.push_back(solution);
00095         solution.clear();
00096         return;
00097     }
00098     for (auto it = cities.begin(); it != cities.end(); it++)
00099     {
00100         bool condition = IfDuplicates(solution, it->m_name);
00101         if (!condition)
00102         {
00103             for (auto jt = it->m_canTravelTo.begin(); jt != it->m_canTravelTo.end(); jt++)
00104             {
00105                 if (*jt == goal)
00106                 {
00107                     condition = IfDuplicates(solution, *jt);
00108                     if (!condition)
00109                     {
00110                         solution.push_back(goal);
00111                         FindRoute(it->m_name, from, cities, solution, solutions);
00112                     }
00113                 }
00114             }
00115         }
00116     }
00117     solution.clear();
00118 }
00119
00120 std::vector<std::string> FastestRoad(const std::vector<std::vector<std::string>& roads,const
      std::vector<FileLine> & timetable)
00121 {
00122     std::vector<std::string> temp;
00123     std::vector<std::string> sol;
00124     std::vector<FileLine>::iterator ptr;
00125     bool FirstCity = true;
00126     int hourOfDepartFromLastCity;
00127     int time{};
00128     int tempTime = 1441;
00129     for (auto it = roads.begin(); it != roads.end(); it++)
00130     {
00131         time = 0;
00132         temp = *it;
```

```
00133         for (size_t i = 0; i < temp.size() - 1; i++)
00134         {
00135             for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
00136             {
00137                 if ((jt->GetCityName("departure") == temp[i]) && (jt->GetCityName("arrival") == temp[i
      + 1]))
00138                 {
00139                     time += jt->GetDuration();
00140                     if (FirstCity)
00141                     {
00142                         hourOfDepartFromLastCity = jt->GetHour(0);
00143                         hourOfDepartFromLastCity += jt->GetDuration();
00144                         FirstCity = false;
00145                     }
00146                     else
00147                     {
00148                         for (int i = 0; i < jt->GetNumberOfHoursOfDepartures(); i++)
00149                         {
00150                             if (jt->GetHour(i) > (hourOfDepartFromLastCity+MINUMUM_TIME_FOR_CHANGE))
00151                             {
00152                                 time += jt->GetHour(i) - hourOfDepartFromLastCity;
00153                                 hourOfDepartFromLastCity = jt->GetHour(i);
00154                                 hourOfDepartFromLastCity += jt->GetDuration();
00155                                 goto foundNextHour;
00156                             }
00157                         }
00158                     }
00159                 }foundNextHour:;
00160             }
00161         }
00162         if (time < tempTime)
00163             sol = temp;
00164         tempTime = time;
00165         FirstCity = true;
00166     }
00167     return sol;
00168 }
00169
00170 std::string ConvertMinutesToHour(int & minutes)
00171 {
00172     std::string hours;
00173     int temp = minutes % 60;
00174     minutes -= temp;
00175     minutes /= 60;
00176     hours = std::to_string(minutes);
00177     hours += ':';
00178     if (temp < 10) hours+='0';
00179     hours += std::to_string(temp);
00180     return hours;
00181 }
00182
00183 int LookForNextDepart(const int & lastHour, const std::string & departCity, const std::string &
      arriveCity,const std::vector<FileLine> & timetable)
00184 {
00185     bool cond=false;
00186     for (auto it = timetable.begin(); it != timetable.end(); it++)
00187     {
00188         if ((it->GetCityName("departure") == departCity) && (it->GetCityName("arrival") ==
      arriveCity))
00189         {
00190             for (int i = 0; i < it->GetNumberOfHoursOfDepartures(); i++)
00191             {
00192                 cond = true;
00193                 if (lastHour < (it->GetHour(i) - MINUMUM_TIME_FOR_CHANGE))
00194                     return it->GetHour(i);
00195             }
00196             if (cond == true) return it->GetHour(0);
00197         }
00198     }
00199 }
00200
00201 int GetTotalTime(const std::vector<std::string>& road, const std::vector<FileLine>& timetable)
00202 {
00203     int departHour{};
00204     int arriveHour{};
00205     int changeTimeMinutes{};
00206
00207     int totalTime{};
00208
00209     for (auto it = road.begin(); it != road.end() - 1; it++)
00210     {
00211         if (it != road.end() - 2)
00212         {
00213             for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
00214             {
00215                 if ((jt->GetCityName("departure") == *it) && (jt->GetCityName("arrival") == *(it +
      1)))
```

```
00216                 {
00217                     if (it == road.begin())
00218                         departHour = jt->GetHour(0);
00219                     else
00220                         departHour = LookForNextDepart(arriveHour, *it, *(it + 1), timetable);
00221                     arriveHour = departHour + jt->GetDuration();
00222                     changeTimeMinutes = LookForNextDepart(arriveHour, *(it + 1), *(it + 2), timetable)
      - arriveHour;
00223                     if (changeTimeMinutes < 0)
00224                     {
00225                         auto temp = LookForNextDepart(arriveHour, *(it + 1), *(it + 2), timetable);
00226                         int minutesInDay = 1440;
00227                         changeTimeMinutes = minutesInDay - arriveHour + temp;
00228                     }
00229                     totalTime += jt->GetDuration() + changeTimeMinutes;
00230 
00231 
00232                     goto foundCity;
00233                 }
00234             }foundCity:;
00235         }
00236         else if (it == road.end() - 2)
00237         {
00238             for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
00239             {
00240                 if ((jt->GetCityName("departure") == *it) && (jt->GetCityName("arrival") == *(it +
      1)))
00241                 {
00242                     totalTime += jt->GetDuration();
00243                     return totalTime;
00244                 }
00245             }
00246         }
00247     }
00248 }
```

4.3 Dokumentacja pliku Project/headers/functions.h

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>
#include <memory>
#include <array>
```

Wykres zależności załączania dla functions.h: Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:

Komponenty

- class FileLine

  Klasa odpowiedzialna za przechowanie informacji pobranych z jednej linii z pliku rozkładu jazdy.
- class City

  Klasa zawierająca informację o nazwie miasta oraz wszystkich miast do których możemy z tego miasta dojechać

Funkcje

- std::vector< City > FillCityVec (std::vector< FileLine > &timetable)

  Funkcja, która na podstawie vectora klas FileLine tworzy vector klas City.
- void FindRoute (std::string goal, const std::string &from, const std::vector< City > &cities, std::vector< std::string > &solution, std::vector< std::vector< std::string > > &solutions)

  Funkcja, która szuka wszystkie możliwe drogi "z miasta A do miasta B".

- void Correction (std::vector< std::vector< std::string > > &roads, const std::string &goal)

  Funkcja korekcyjna dla funkcji FindRoute(). Jest potrzebna do optymalizacji wyników uzyskanych dzięki funkcji FindRoute.
- std::vector< std::string > FastestRoad (const std::vector< std::vector< std::string > > &roads, const std::vector< FileLine > &timetable)

  Funkcja, która na podstawie kontenera roads, znajdzie najszybszą z dróg.
- std::string ConvertMinutesToHour (int &minutes)

  Funkcja, która na podstawie liczby minut podanej w postaci int wygeneruje napis zawierający informacje o tym jaka to jest godzina.
- int LookForNextDepart (const int &lastHour, const std::string &departCity, const std::string &arriveCity, const std::vector< FileLine > &timetable)

  Funkcja szukająca następnej godziny odjazdu "z miasta A do miasta B" względem podanej godziny.
- int GetTotalTime (const std::vector< std::string > &road, const std::vector< FileLine > &timetable)

  Funkcja, która na podstawie podanej drogi liczy łączny czas jazdy (razem z czasem oczekiwania na odjazd)

4.3.1 Dokumentacja funkcji

4.3.1.1 ConvertMinutesToHour()

```
std::string ConvertMinutesToHour (
            int & minutes )
```

Funkcja, która na podstawie liczby minut podanej w postaci int wygeneruje napis zawierający informacje o tym jaka to jest godzina.

Parametry

minutes	liczba minut

Definicja w linii 170 pliku functions.cpp.

Oto graf wywoływań tej funkcji:

4.3.1.2 Correction()

```
void Correction (
            std::vector< std::vector< std::string > > & roads,
            const std::string & goal )
```

Funkcja korekcyjna dla funkcji FindRoute(). Jest potrzebna do optymalizacji wyników uzyskanych dzięki funkcji FindRoute.

Parametry

roads	Vector przechowujący wszystki drogi(które są vectorami stringów)
goal	Parametr zawierający nazwę miasta docelowego Vector przechowujący wszystkie drogi aktualnie zawiera parę niechcianych wyników. Wszystkie drogi, które nie kończą się miastem docelowym zostaną usunięte z vectora

4.3.1.2.1 Element usuwający

```
      //aktualnie wszystkie drogi są w kolejności od miasta docelowego do miasta, z którego odjeżdzamy
for (int i=0;i<roads.size();i++)
{
    std::reverse(roads[i].begin(), roads[i].end());  //odwracamy miasta i teraz są w kolejności od miasta, z
       którego odjeżdzamy do miasta docelowego
    if (roads[i].back() != goal)
        roads[i].clear();   //jeśli droga nie kończy się na mieście docelowonym usuwamy całą zawartość
       vectora
}
//Pod koniec funkcji elementy puste zostają usunięte z vectora dróg
```

4.3.1.2.2 Element usuwający duplikaty z vectora dróg

```
auto end = roads.end();
for (auto it = roads.begin(); it != end; it++)
    end = std::remove(it + 1, end, *it);
roads.erase(end, roads.end());
```

Zobacz również

FindRoute()

Definicja w linii 65 pliku functions.cpp.

Oto graf wywoływań tej funkcji:

4.3.1.3 FastestRoad()

```
std::vector< std::string > FastestRoad (
            const std::vector< std::vector< std::string > > & roads,
            const std::vector< FileLine > & timetable )
```

Funkcja, która na podstawie kontenera roads, znajdzie najszybszą z dróg.

Parametry

roads	kontener zawierający wszystkie połączenia "z miasta A do miasta B"
timetable	kontener zawierający informację o wszystkiech połączeniach (takie jak czas jazdy, godziny odjazdu etc.)

4.3.1.3.1 Algorytm sprawdzający, która z dróg jest najszybsza

```
     int hourOfDepartFromLastCity; // zmienna potrzebna do przechowania informacji o godzinie, o której
      podróżnik będzie w mieście, do którego przyjedzie
for (auto it = roads.begin(); it != roads.end(); it++)
   {
    time = 0; // zerowanie zmiennej, która służy do liczenia łącznego czasu podróży
    temp = *it;
    for (size_t i = 0; i < temp.size() - 1; i++)
    {
        for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
        {
            if ((jt->GetCityName("departure") == temp[i]) && (jt->GetCityName("arrival") == temp[i + 1]))
            {
                time += jt->GetDuration(); // dodanie do sumy czasu podróży szacownego czasu podróży "z
      miasta A do miasta B"
                if (FirstCity)
                {
                    hourOfDepartFromLastCity = jt->GetHour(0);     // jeżeli to pierwsze miasto z drogi
      zaczynamy od pierwszej godziny, o której możemy odjechać
                    hourOfDepartFromLastCity += jt->GetDuration(); // dodajemy to znalezionej godziny czas
      podróży
                    FirstCity = false;
                }
                else
                {
                    for (int i = 0; i < jt->GetNumberOfHoursOfDepartures(); i++)
                    {
                        if (jt->GetHour(i) > (hourOfDepartFromLastCity+MINUMUM_TIME_FOR_CHANGE)) // jeżeli
      to nie pierwsze miasto z drogi szukamy następnej
```

```
                    {                                                                                    //
    najwcześniejszej godziny (uwzględniając czas na przesiadkę)
                        time += jt->GetHour(i) - hourOfDepartFromLastCity;
                        hourOfDepartFromLastCity = jt->GetHour(i);
                        hourOfDepartFromLastCity += jt->GetDuration();
                        goto foundNextHour; // jeżeli następna godzina odjazdu została odnaleziona
    szukamy od razu godziny dla następnej pary miast
                    }
                }
            }
        foundNextHour:;
        }
    }
  }
  if (time < tempTime)
      sol = temp;
  tempTime = time; // dodatkowa zmienna potrzebna do porównania czasów
  FirstCity = true;
 }
```

Definicja w linii 120 pliku functions.cpp.

Oto graf wywoływań tej funkcji:

4.3.1.4 FillCityVec()

```
std::vector< City > FillCityVec (
             std::vector< FileLine > & timetable )
```

Funkcja, która na podstawie vectora klas FileLine tworzy vector klas City.

Parametry

timetable	Gotowy kontener zawierający informację o każdej linii pliku rozkładu jazdy przechowywanej w postaci klasy

Definicja w linii 5 pliku functions.cpp.

Oto graf wywoływań tej funkcji:

4.3.1.5 FindRoute()

```
void FindRoute (
             std::string goal,
             const std::string & from,
             const std::vector< City > & cities,
             std::vector< std::string > & solution,
             std::vector< std::vector< std::string > > & solutions )
```

Funkcja, która szuka wszystkie możliwe drogi "z miasta A do miasta B".

Parametry

goal	zmienna przechowująca nazwę miasta docelowego
from	zmienna przechowująca nazwę miasta odjazdu
cities	kontener klas zawierający wszystkie połączenia między miastami
solution	zmienna tymczasowa, która jest potrzebna do poprawnego działania algorytmu, zawiera jedną drogę (wariacja połączeń miast)
solutions	kontener, który zostanie zwrócony. Zawiera wszystkie połączenia "z miasta A do miasta B", które będą korygowane w funkcji Correction()

Funkcja jest rekurencyjna. Algorytm szuka wszystkich miast, które mogą dojechać do danego miasta, do którego chcemy dojechać. Następnie szuka wszystkich możliwych miast, z których możemy dojechać do miasta, które jest w stanie dojechać do pierwotnego miasta docelowego etc.

4.3.1.5.1 Element szukający pasujące miasta

```
for (auto it = cities.begin(); it != cities.end(); it++)
{
    bool condition = IfDuplicates(solution, it->m_name); //Funkcja IfDuplicates() zwraca true jeśli w
       vectorze, w którym zapisujemy naszą aktualną drogę
    if (!condition)                                      //znajduje się już miasto do którego chcemy
       dojechać
    {
        for (auto jt = it->m_canTravelTo.begin(); jt != it->m_canTravelTo.end(); jt++)
        {
            if (*jt == goal)
            {
                solution.push_back(goal);
                FindRoute(it->m_name, from, cities, solution, solutions);
            }
        }
    }
}
```

4.3.1.5.2 Warunek końca rekurencji

```
if (goal == from)
{
 solution.push_back(from);
 solutions.push_back(solution);
 solution.clear();
 return;
}
```

Funkcja zawiera w efekcie końcowym parę wyników, które nie są potrzebne dla działania programu i są korygowane w funkcji Correction()

Zobacz również

Correction()

Definicja w linii 89 pliku functions.cpp.

Oto graf wywołań dla tej funkcji: Oto graf wywoływań tej funkcji:

4.3.1.6 GetTotalTime()

```
int GetTotalTime (
            const std::vector< std::string > & road,
            const std::vector< FileLine > & timetable )
```

Funkcja, która na podstawie podanej drogi liczy łączny czas jazdy (razem z czasem oczekiwania na odjazd)

Parametry

road	dane połączenie "z miasta A do miasta B"
timetable	zawierający informację o wszystkich połączeniach (takie jak czas jazdy, godziny odjazdu etc.)

Definicja w linii 201 pliku functions.cpp.

Oto graf wywołań dla tej funkcji: Oto graf wywoływań tej funkcji:

4.3.1.7 LookForNextDepart()

```
int LookForNextDepart (
            const int & lastHour,
```

```
            const std::string & departCity,
            const std::string & arriveCity,
            const std::vector< FileLine > & timetable )
```

Funkcja szukająca następnej godziny odjazdu "z miasta A do miasta B" względem podanej godziny.

Parametry

lastHour	informacja o godzinie, od której chcemy znaleść kolejną godzinę odjazdu
departCity	miasto, z którego odjeżdzamy
arriveCity	miasto docelowe
timetable	zawierający informację o wszystkich połączeniach (takie jak czas jazdy, godziny odjazdu etc.)

4.3.1.7.1 Algorytm szukający następnej godziny

```
bool cond=false;
   for (auto it = timetable.begin(); it != timetable.end(); it++)
   {
    if ((it->GetCityName("departure") == departCity) && (it->GetCityName("arrival") == arriveCity))
    {
        for (int i = 0; i < it->GetNumberOfHoursOfDepartures(); i++)
        {
            cond = true; // jeżeli funkcja nie skończy się po wykonaniu tej pętli to znaczy, że w danym dniu
       już nie ma odjazdów
            if (lastHour < (it->GetHour(i) - MINUMUM_TIME_FOR_CHANGE))
                return it->GetHour(i);
        }
        if (cond == true) return it->GetHour(0); // funkcja zwraca pierwszą godzinę, z której można odjechać
       (czyli trzeba czekać do następnego dnia)
    }
   }
```

Definicja w linii 183 pliku functions.cpp.

Oto graf wywoływań tej funkcji:

4.4 functions.h

Idź do dokumentacji tego pliku.

```
00001 #ifndef FUNCTIONS_H
00002 #define FUNCTIONS_H
00003
00004 #include <iostream>
00005 #include <fstream>
00006 #include <sstream>
00007 #include <string>
00008 #include <vector>
00009 #include <algorithm>
00010 #include <memory>
00011 #include <array>
00012
00014 class FileLine
00015 {
00016 private:
00017     std::string m_departure;
00018     std::string m_arrival;
00019     int m_duration{};
00020     std::array<std::string,3> m_time;
00021 public:
00022
00026     std::string GetCityName(const std::string & boolean) const
00027     {
00028         std::string city = boolean=="arrival"? m_arrival : m_departure;
00029         return city;
00030     }
00031
00033     int GetDuration() const { return m_duration; }
00034
00036     int GetNumberOfHoursOfDepartures() const { return m_time.size(); }
00037
00040     int GetHour(short n) const { return ConvertHour(n); }
00041
```

```
00044     void GetFileLine(const std::string& str)
00045     {
00046         std::stringstream ss;
00047         ss « str;
00048         ss » m_departure » m_arrival » m_duration » m_time[0] » m_time[1] » m_time[2];
00049     }
00050 private:
00053     int ConvertHour (const short & n) const
00054     {
00055         std::string hours;
00056         std::string minutes;
00057         if (m_time[n].size() == 5)
00058         {
00059             hours = m_time[n].substr(0, 2);
00060             minutes = m_time[n].substr(3, 2);
00061         }
00062         else
00063         {
00064             hours = m_time[n].substr(0, 1);
00065             minutes = m_time[n].substr(2, 2);
00066         }
00067         int h = std::stoi(hours);
00068         int m = std::stoi(minutes);
00069         h *= 60;
00070         return h + m;
00071     }
00072 };
00073
00075 class City
00076 {
00077 public:
00078     std::string m_name;
00079     std::vector<std::string> m_canTravelTo;
00080 public:
00081
00084     void SetName(const std::string & str) { m_name = str; }
00085
00088     void CanTravel(const std::string & str) { m_canTravelTo.push_back(str); }
00089
00091     void ClearVec() { m_canTravelTo.clear(); }
00092 };
00093
00097 std::vector<City> FillCityVec(std::vector<FileLine> & timetable);
00098
00138 void FindRoute(std::string goal, const std::string& from,const std::vector<City>& cities,
       std::vector<std::string>& solution, std::vector<std::vector<std::string>& solutions);
00139
00164 void Correction(std::vector<std::vector<std::string>& roads, const std::string& goal);
00165
00166
00213 std::vector<std::string> FastestRoad(const std::vector<std::vector<std::string>& roads, const
       std::vector<FileLine>& timetable);
00214
00218 std::string ConvertMinutesToHour(int& minutes);
00219
00242 int LookForNextDepart(const int & lastHour, const std::string& departCity, const std::string&
       arriveCity,const std::vector<FileLine>& timetable);
00243
00247 int GetTotalTime(const std::vector<std::string>& road, const std::vector<FileLine>& timetable);
00248
00249 #endif //FUNCTIONS_H
```

4.5 Dokumentacja pliku Project/Project/src/main.cpp

`#include "functions.h"`
Wykres zależności załączania dla main.cpp:

Funkcje

- int main ()

4.5.1 Dokumentacja funkcji

4.5.1.1 main()

```
int main ( )
```

< Plik rozkładu jazdy

< Plik z drogami, które chcemy znaleźć

< Plik wyjściowy

< Vector ze zinterpretowanymi liniami pliku rozkładu jazdy

< Vector wszystkich miast i ich połączeń

< szukanie wszystkich dróg

< korekta wyników funkcji FindRoute

< najszybsza trasa

< łączny czas podróży

Definicja w linii 3 pliku main.cpp.

Oto graf wywołań dla tej funkcji:

4.6 main.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "functions.h"
00002
00003 int main()
00004 {
00005     std::ifstream inFileTimetable("..\\data\\timetable.txt");
00006     std::ifstream inFileTravel("..\\data\\travel.txt");
00007     std::ofstream toSolutionFile("..\\data\\solution.txt");
00008     std::string line;
00009     FileLine fline;
00010     std::vector<FileLine> timetable;
00011     std::vector<City> cities;
00012     std::vector<City> tempCities;
00013
00014     while (std::getline(inFileTimetable, line))
00015     {
00016         fline.GetFileLine(line);
00017         timetable.push_back(fline);
00018     }
00019
00020
00021     struct { std::string a; std::string b; } traveler;
00022     auto& [departCity, arriveCity] = traveler;
00023
00024     std::stringstream ss;
00025     std::vector<std::string> temp;
00026     std::vector<std::string> fastest;
00027     std::vector<std::vector<std::string» roads;
00028     int time{};
00029     int totalTime{};
00030     int departHour{};
00031     int arriveHour{};
00032     int changeTimeMinutes{};
00033     std::string changeTime{};
00034
00035     while (std::getline(inFileTravel, line))
00036     {
00037         ss « line;
00038         ss » departCity » arriveCity;
00039         cities = FillCityVec(timetable);
00040         FindRoute(arriveCity, departCity, cities, temp, roads);
```

```
00041         Correction(roads, arriveCity);
00042         if (!roads.empty())
00043         {
00044             fastest = FastestRoad(roads, timetable);
00045             totalTime = GetTotalTime(fastest, timetable);
00046 
00047 
00048             toSolutionFile « departCity « " --> " « arriveCity « " (czas: " «
00049                 ConvertMinutesToHour(totalTime) « ", przesiadki: " « fastest.size() - 2 « ")\n" «
00050                 "-----------------------------------------------------\n";
00051 
00052                 for (auto it = fastest.begin(); it != fastest.end() - 1; it++)
00053                 {
00054                     if (fastest.size() != 2)
00055                     {
00056                         if (it != fastest.end() - 2)
00057                         {
00058                             for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
00059                             {
00060                                 if ((jt->GetCityName("departure") == *it) &&
      (jt->GetCityName("arrival") == *(it + 1)))
00061                                 {
00062                                     if (it == fastest.begin())
00063                                         departHour = jt->GetHour(0);
00064                                     else
00065                                         departHour = LookForNextDepart(arriveHour, *it, *(it + 1),
      timetable);
00066                                     arriveHour = departHour + jt->GetDuration();
00067                                     changeTimeMinutes = LookForNextDepart(arriveHour, *(it + 1), *(it
      + 2), timetable) - arriveHour;
00068 
00069                                     auto temp1 = arriveHour;
00070                                     auto temp2 = departHour;
00071                                     auto temp3 = changeTimeMinutes;
00072                                     if (temp3 < 0) temp3 *= -1;
00073                                     changeTime = ConvertMinutesToHour(temp3);
00074 
00075                                     toSolutionFile « jt->GetCityName("departure") « " (" «
      ConvertMinutesToHour(temp2) «
00076                                         ") --> " « jt->GetCityName("arrival") « " (" «
      ConvertMinutesToHour(temp1) «
00077                                         "), przesiadka za " « changeTime « '\n';
00078                                     goto foundCity;
00079                                 }
00080                             }foundCity:;
00081                         }
00082                         else if (it == fastest.end() - 2)
00083                         {
00084                             for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
00085                             {
00086                                 if ((jt->GetCityName("departure") == *it) &&
      (jt->GetCityName("arrival") == *(it + 1)))
00087                                 {
00088                                     departHour = changeTimeMinutes + arriveHour;
00089                                     arriveHour = departHour + jt->GetDuration();
00090 
00091                                     toSolutionFile « jt->GetCityName("departure") « " (" «
      ConvertMinutesToHour(departHour) «
00092                                         ") --> " « jt->GetCityName("arrival") « " (" «
      ConvertMinutesToHour(arriveHour) « ")\n\n";
00093                                     goto loopEnd;
00094                                 }
00095                             }loopEnd:;
00096                         }
00097                     }
00098                     else
00099                     {
00100                         for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
00101                         {
00102                             if ((jt->GetCityName("departure") == *it) && (jt->GetCityName("arrival")
      == *(it + 1)))
00103                             {
00104                                 departHour = jt->GetHour(0);
00105                                 arriveHour = departHour + jt->GetDuration();
00106 
00107                                 toSolutionFile « jt->GetCityName("departure") « " (" «
      ConvertMinutesToHour(departHour) «
00108                                     ") --> " « jt->GetCityName("arrival") « " (" «
      ConvertMinutesToHour(arriveHour) « ")\n\n";
00109                             }
00110                         }
00111                     }
00112                 }
00113             roads.clear();
00114         }
00115         else
00116             toSolutionFile « departCity « " --> " « arriveCity « " - DROGA NIE ISTNIEJE\n\n";
```

```
00117         ss.clear();
00118     }
00119 }
```

Indeks