

Projekt PPK

Wygenerowano przez Doxygen 1.9.3

1 Indeks klas	1
1.1 Lista klas	1
2 Indeks plików	3
2.1 Lista plików	3
3 Dokumentacja klas	5
3.1 Dokumentacja klasy City	5
3.1.1 Opis szczegółowy	5
3.1.2 Dokumentacja funkcji składowych	5
3.1.2.1 CanTravel()	5
3.1.2.2 ClearVec()	6
3.1.2.3 SetName()	6
3.1.3 Dokumentacja atrybutów składowych	6
3.1.3.1 m_canTravelTo	6
3.1.3.2 m_name	7
3.2 Dokumentacja klasy FileLine	7
3.2.1 Opis szczegółowy	8
3.2.2 Dokumentacja funkcji składowych	8
3.2.2.1 ConvertHour()	8
3.2.2.2 GetCityName()	8
3.2.2.3 GetDuration()	9
3.2.2.4 GetFileLine()	9
3.2.2.5 GetHour()	9
3.2.2.6 GetNumberOfHoursOfDepartures()	9
3.2.3 Dokumentacja atrybutów składowych	10
3.2.3.1 m_arrival	10
3.2.3.2 m_departure	10
3.2.3.3 m_duration	10
3.2.3.4 m_time	10
4 Dokumentacja plików	11
4.1 Dokumentacja pliku Project/headers/functions.cpp	11
4.2 functions.cpp	11
4.3 Dokumentacja pliku Project/headers/functions.h	14
4.3.1 Dokumentacja funkcji	15
4.3.1.1 ConvertMinutesToHour()	15
4.3.1.2 Correction()	15
4.3.1.3 FastestRoad()	16
4.3.1.4 FillCityVec()	17
4.3.1.5 FindRoute()	17
4.3.1.6 GetTotalTime()	18
4.3.1.7 LookForNextDepart()	18

4.4 functions.h	19
4.5 Dokumentacja pliku Project/Project/src/main.cpp	20
4.5.1 Dokumentacja funkcji	20
4.5.1.1 main()	21
4.6 main.cpp	21
Indeks	25

Rozdział 1

Indeks klas

1.1 Lista klas

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

City	Klasa zawierająca informację o nazwie miasta oraz wszystkich miast do których możemy z tego miasta dojechać	5
FileLine	Klasa odpowiedzialna za przechowanie informacji pobranych z jednej linii z pliku rozkładu jazdy	7

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Project/headers/ functions.cpp	11
Project/headers/ functions.h	14
Project/Project/src/ main.cpp	20

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

<i>str</i>	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

<i>str</i>	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 mieście, z którego odjeżdżamy
- `std::string m_arrival`
zmienna przechowująca informację o mieście docelowego
- `int m_duration {}`
zmienna przechowująca informację o czasie jazdy z miasta, z którego odjeżdżamy 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

<i>n</i>	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

<i>boolean</i>	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

<i>str</i>	String, który zawiera jedną linijkę z pliku rozkładu jazdy
------------	--

Definicja w linii 44 pliku [functions.h](#).

Oto graf wywołań 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

<i>n</i>	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 mieście, z którego odjeżdżamy

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 odjeżdżamy 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 MINIMUM_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,
00090               std::vector<std::string>& solution, std::vector<std::vector<std::string>>& solutions)
00091 {
00092     if (goal == from)
00093     {
00094         solution.push_back(from);
00095         if (!solution.empty()) solutions.push_back(solution);
00096         solution.clear();
00097         return;
00098     }
00099     for (auto it = cities.begin(); it != cities.end(); it++)
00100     {
00101         bool condition = IfDuplicates(solution, it->m_name);
00102         if (!condition)
00103         {
00104             for (auto jt = it->m_canTravelTo.begin(); jt != it->m_canTravelTo.end(); jt++)
00105             {
00106                 if (*jt == goal)
00107                 {
00108                     condition = IfDuplicates(solution, *jt);
00109                     if (!condition)
00110                     {
00111                         solution.push_back(goal);
00112                         FindRoute(it->m_name, from, cities, solution, solutions);
00113                     }
00114                 }
00115             }
00116         }
00117         solution.clear();
00118     }
00119 }
00120 std::vector<std::string> FastestRoad(const std::vector<std::vector<std::string>>& roads, const
00121                                   std::vector<FileLine> & timetable)
00122 {
00123     std::vector<std::string> temp;
00124     std::vector<std::string> sol;
00125     std::vector<FileLine>::iterator ptr;
00126     bool FirstCity = true;
00127     int hourOfDepartFromLastCity;
00128     int time{};
00129     int tempTime = 1441;
00130     for (auto it = roads.begin(); it != roads.end(); it++)
00131     {
00132         time = 0;
00133         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             goto foundCity;
00232         }
00233     }foundCity;;
00234 }
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

<i>minutes</i>	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

<i>roads</i>	Vector przechowujący wszystkie drogi(które są vectorami stringów)
<i>goal</i>	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 odjeżdżamy
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 odjeżdżamy do miasta docelowego
    if (roads[i].back() != goal)
        roads[i].clear(); //jeśli droga nie kończy się na mieście docelowym usuwamy całą zawartość
        wektora
}
//Pod koniec funkcji elementy puste zostają usunięte z wektora dróg
```

4.3.1.2.2 Element usuwający duplikaty z wektora 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

<i>roads</i>	kontener zawierający wszystkie połączenia "z miasta A do miasta B"
<i>timetable</i>	kontener zawierający informację o wszystkich 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 wywołań 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

<i>timetable</i>	Gotowy kontener zawierający informację o każdej linii pliku rozkładu jazdy przechowywanej w postaci klasy
------------------	---

Definicja w linii 5 pliku [functions.cpp](#).

Oto graf wywołań 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

<i>goal</i>	zmienna przechowująca nazwę miasta docelowego
<i>from</i>	zmienna przechowująca nazwę miasta odjazdu
<i>cities</i>	kontener klas zawierający wszystkie połączenia między miastami
<i>solution</i>	zmienna tymczasowa, która jest potrzebna do poprawnego działania algorytmu, zawiera jedną drogę (wariacja połączeń miast)
<i>solutions</i>	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

<i>road</i>	dane połączenie "z miasta A do miasta B"
<i>timetable</i>	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

<i>lastHour</i>	informacja o godzinie, od której chcemy znaleźć kolejną godzinę odjazdu
<i>departCity</i>	miasto, z którego odjeżdżamy
<i>arriveCity</i>	miasto docelowe
<i>timetable</i>	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) &&
00061                                 (jt->GetCityName("arrival") == *(it + 1)))
00062                             {
00063                                 if (it == fastest.begin())
00064                                     departHour = jt->GetHour(0);
00065                                 else
00066                                     departHour = LookForNextDepart(arriveHour, *it, *(it + 1),
00067                                         timetable);
00068                                 arriveHour = departHour + jt->GetDuration();
00069                                 changeTimeMinutes = LookForNextDepart(arriveHour, *(it + 1), *(it
00070                                     + 2), timetable) - arriveHour;
00071
00072                                 auto temp1 = arriveHour;
00073                                 auto temp2 = departHour;
00074                                 auto temp3 = changeTimeMinutes;
00075                                 if (temp3 < 0) temp3 *= -1;
00076                                 changeTime = ConvertMinutesToHour(temp3);
00077
00078                                 toSolutionFile << jt->GetCityName("departure") << " (" <<
00079                                     ConvertMinutesToHour(temp2) <<
00080                                     " " << jt->GetCityName("arrival") << " (" <<
00081                                     ConvertMinutesToHour(temp1) <<
00082                                     ")", przesiadka za " << changeTime << '\n';
00083                                 goto foundCity;
00084                             }
00085                             }foundCity;;
00086                         }
00087                     else if (it == fastest.end() - 2)
00088                     {
00089                         for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
00090                         {
00091                             if ((jt->GetCityName("departure") == *it) &&
00092                                 (jt->GetCityName("arrival") == *(it + 1)))
00093                             {
00094                                 departHour = changeTimeMinutes + arriveHour;
00095                                 arriveHour = departHour + jt->GetDuration();
00096                                 toSolutionFile << jt->GetCityName("departure") << " (" <<
00097                                     ConvertMinutesToHour(departHour) <<
00098                                     " " << jt->GetCityName("arrival") << " (" <<
00099                                     ConvertMinutesToHour(arriveHour) << ")\n\n";
00100                                 goto loopEnd;
00101                             }
00102                             }loopEnd;;
00103                         }
00104                     }
00105                     else
00106                     {
00107                         for (auto jt = timetable.begin(); jt != timetable.end(); jt++)
00108                         {
00109                             if ((jt->GetCityName("departure") == *it) && (jt->GetCityName("arrival")
00110                                 == *(it + 1)))
00111                             {
00112                                 departHour = jt->GetHour(0);
00113                                 arriveHour = departHour + jt->GetDuration();
00114                                 toSolutionFile << jt->GetCityName("departure") << " (" <<
00115                                     ConvertMinutesToHour(departHour) <<
00116                                     " " << jt->GetCityName("arrival") << " (" <<
00117                                     ConvertMinutesToHour(arriveHour) << ")\n\n";
00118                             }
00119                         }
00120                     }
00121                 }
00122             }
00123             roads.clear();
00124         }
00125         else
00126             toSolutionFile << departCity << " --> " << arriveCity << " - DROGA NIE ISTNIEJE\n\n";

```

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


Indeks

CanTravel
 City, [5](#)

City, [5](#)
 CanTravel, [5](#)
 ClearVec, [6](#)
 m_canTravelTo, [6](#)
 m_name, [6](#)
 SetName, [6](#)

ClearVec
 City, [6](#)

ConvertHour
 FileLine, [8](#)

ConvertMinutesToHour
 functions.h, [15](#)

Correction
 functions.h, [15](#)

FastestRoad
 functions.h, [16](#)

FileLine, [7](#)
 ConvertHour, [8](#)
 GetCityName, [8](#)
 GetDuration, [8](#)
 GetFileLine, [9](#)
 GetHour, [9](#)
 GetNumberOfHoursOfDepartures, [9](#)
 m_arrival, [10](#)
 m_departure, [10](#)
 m_duration, [10](#)
 m_time, [10](#)

FillCityVec
 functions.h, [17](#)

FindRoute
 functions.h, [17](#)

functions.h
 ConvertMinutesToHour, [15](#)
 Correction, [15](#)
 FastestRoad, [16](#)
 FillCityVec, [17](#)
 FindRoute, [17](#)
 GetTotalTime, [18](#)
 LookForNextDepart, [18](#)

GetCityName
 FileLine, [8](#)

GetDuration
 FileLine, [8](#)

GetFileLine
 FileLine, [9](#)

GetHour
 FileLine, [9](#)

GetNumberOfHoursOfDepartures
 FileLine, [9](#)

GetTotalTime
 functions.h, [18](#)

LookForNextDepart
 functions.h, [18](#)

m_arrival
 FileLine, [10](#)

m_canTravelTo
 City, [6](#)

m_departure
 FileLine, [10](#)

m_duration
 FileLine, [10](#)

m_name
 City, [6](#)

m_time
 FileLine, [10](#)

main
 main.cpp, [20](#)

main.cpp
 main, [20](#)

Project/headers/functions.cpp, [11](#)
Project/headers/functions.h, [14](#), [19](#)
Project/Project/src/main.cpp, [20](#), [21](#)

SetName
 City, [6](#)