Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Temat: Mapa (10)

Autor	Karol Kadłubowski
Prowadzący	Dr inż. Adam Gudyś
Rok akademicki	2019/2020
Kierunek	Informatyka
Rodzaj studiów	SSI
Semestr	1
Termin laboratorium	Poniedziałek, 10:15 - 11:45
Sekcja	2
Termin oddania sprawozdania	2020-01-26

1. Treść zadania

Napisać program, który umożliwia znalezienie najkrótszej trasy miedzy dwoma miastami. Miasta połączone są drogami o pewnej długości. Drogi są jednokierunkowe. Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-d plik wejściowy z drogami

-t plik wejściowy z trasami do wyznaczenia

-o plik wynikowy z wyznaczony trasami

.

2. Analiza zadania

Zagadnienie przedstawia problem znalezienia najkrótszej drogi pomiędzy dwoma miastami zgodnie z trasami zapisanymi w pliku dróg.

2.1. Struktury danych

W programie do przedstawienia grafu wykorzystana jest lista sąsiedztwa składająca się z listy dwukierunkowej wierzchołków grafu, w której każdy wierzchołek ma swoją listę jednokierunkową krawędzi. Sąsiedztwo tutaj polega na tym, że poszczególne krawędzie grafu posiadają wskaźniki do sąsiadujących wierzchołków. Oprócz grafu istnieje także lista dwukierunkowa, służąca za kolejkę do relaksacji najkrótszej drogi między dwoma miastami. Lista sąsiedztwa jest optymalną strukturą do symulacji grafu, ponieważ oddaje zależności pomiędzy sąsiednimi wierzchołkami, które są istotą grafu.

2.2. Algorytmy

Po załadowaniu danych z pliku dróg bezpośrednio do listy sąsiedztwa, program przepisuje wszystkie wierzchołki z listy sąsiedztwa do kolejki. Następnie program pobiera kolejno trasy z pliku tras do wyznaczenia i szuka najkrótszej ścieżki za pomocą Algorytmu Dijkstry. Algorytm ten przeszukuje kolejne wierzchołki grafu, każdorazowo sprawdzając możliwość relaksacji najkrótszej trasy do sprawdzanego wierzchołka. Szybkość przejścia tego algorytmu dla każdej trasy wynosi średnio $O(n^2)$ w zależności od ilości wierzchołków w grafie. Po każdorazowym wykorzystaniu Algorytmu Dijkstry, trasa zapisywana jest do pliku wyjściowego oraz kolejka zostaje wyzerowana żeby przygotować się pod kolejną trasę do wyznaczenia.

3. Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików: wejściowych mapy trasy i zadanych tras, oraz wyjściowego po odpowiednich przełącznikach (odpowiednio: -d, -t, -o) w dowolnej kolejności np.:

```
Mapa.exe –d mapa.txt –t inputroutes.txt -o outputroutes.txt
```

Pliki powinny być plikami tekstowymi o rozszerzeniu .txt. Uruchomienie programu bez żadnego parametru lub z niewłaściwymi parametrami. Powoduje zwrócenie odpowiedniego komunikatu np.:

```
Mapa.exe –d mapa.txt –t inputroutes.txt
```

```
Niepoprawne parametry programu
```

W przypadku braku obecności pliku dróg lub zadanych tras mimo podania prawidłowych przełączników program wyświetli odpowiedni komunikat:

```
Brak pliku mapy
```

```
Brak pliku z zadanymi trasami
```

W każdym przypadku podania niewłaściwych danych, program kończy swoje działanie.

4. Specyfikacja wewnętrzna

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

4.1. Ogólna struktura programu

W funkcji głównej wywołana zostaje funkcja **switches,** która sprawdza czy program został poprawnie uruchomiony z wiersza poleceń, jeśli nie to wyświetlony zostaje stosowny komunikat. Następnie po sprawdzeniu pliku dróg uruchomiona zostaje funkcja **load**, która otwiera plik wejściowy i wczytuje dane do listy sąsiedztwa, a z niej funkcja **addtolist** wpisuje je do kolejki. Następnie program sprawdza plik wejściowy tras do wyznaczenia po czym w pętli wywołana zostaje funkcja **load2.** Dla każdego przebiegu pętli odpowiada ona za realizację Algorytmu Dijkstry, zapisywanie wyników do pliku wyjściowego oraz resetowania kolejki do ponownego użycia w następnym przebiegu pętli. Funkcja **load2** najpierw wczytuje miasto początkowe i końcowe po czym sprawdza czy oba miasta znajdują się w grafie. Jeśli nie to do pliku wydrukowany zostaje komunikat o nieprawidłowości zadania trasy. Następnie uruchomiona zostaje funkcja **dijkstra2**, która najpierw ustawia w kolejce wierzchołek początkowy korzystając z funkcji **searchinlist**, po czym kolejno w każdym przebiegu pętli szuka wierzchołka o najmniejszym koszcie dojścia ze zbioru jeszcze nieprzeszukanych za pomocą funkcji **searchfirstfalse** i **searchthelowest**. Gdy już znajdzie dany wierzchołek przenosi go ze zbioru „do przeszukania" do zbioru przeszukanych oraz uruchamia funkcję **cnv**, która przeszukuje listę krawędzi danego wierzchołka sprawdzając czy suma dystansu pokonanego obecnym torem jest mniejsza niż wartość zapisana w kolejce. Jeśli tak to wartość w kolejce zostaje zmieniona. Następuję kontynuacja funkcji **load2**, w kolejce wyszukane zostają wierzchołek początkowy i końcowy, po czym następuję rekurencyjny wypis do pliku wyjściowego za pomocą funkcji **printsol**. Jeśli nie udało się wyznaczyć trasy, ponieważ nie ma takiego połączenia dróg funkcja wydrukuje odpowiedni komunikat do pliku wyjściowego. Następnie kolejka jest resetowana do pierwotnego stanu za pomocą funkcji **resetlist**. W przypadku pustych plików wejściowych program wypisze stosowny komunikat. W przypadku uszkodzonych plików wejściowych, program pominie linie, które są uszkodzone, ale będzie kontynuował pracę programu oraz wczytywanie pozostałych linii.

4.2. Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

5. Testowanie

Program został przetestowany na różnego rodzaju plikach wejściowych. Pliki niepoprawne o wadliwych rekordach (ujemne wartości tras, litery w miejscu tras, niekompletne rekordy) czy pliki puste wyświetlają odpowiednie komunikaty. Maksymalna liczba w pliku zależy od kompilatora (typ int może być realizowany jako zmienna dwubajtowa lub czterobajtowa). Maksymalna wielkość pliku dróg, dla której udało się poprawnie uruchomić program to 48 KB. Większe pliki wywołują StackOverflow z uwagi na zastosowania rekurencyjne w programie. Program został sprawdzony pod kątem wycieków pamięci funkcją **_CrtDumpMemoryLeaks**.

6. Wnioski

Program do wyszukiwania najkrótszej trasy na podstawie zadanej mapy okazał się bardzo ciekawym do realizacji projektem. Na jego podstawie uświadomiłem sobie wadliwość rozwiązań rekurencyjnych dla większych ilości danych. Jednocześnie ciągłe czuwanie w trakcie pisania programu by pamięć w dynamicznych strukturach danych była zwalniana na bieżąco okazało się pokrzepiające, kiedy kończąc

projekt nie musiałem korygować żadnych wycieków pamięci. Największym wyzwaniem była jednak konfiguracja struktur dynamicznych, ponieważ trzeba było uwzględnić szczególne przypadki by listy działały prawidłowo.

7. Źródła

https://eduinf.waw.pl/inf/alg/001_search/0138.php

https://en.cppreference.com/w/

Mapa

Generated by Doxygen 1.8.16

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

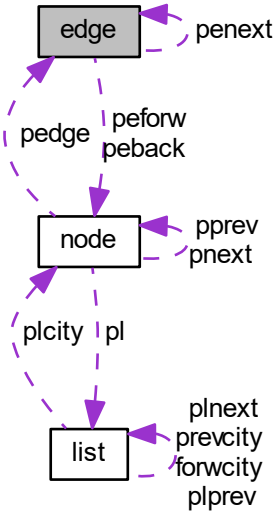
Chapter 3

Class Documentation

3.1 edge Struct Reference

`#include <structures.h>`

Collaboration diagram for edge:



Public Attributes

- type edistance
  waga krawędzi
- node * peback
  wskaźnik do węzła na początku krawędzi
- node * peforw
  wskaźnik do węzła na końcu krawędzi
- edge * penext
  wskaźnik na następną krawędź danego węzła

3.1.1 Detailed Description

Struktura reprezentująca listę krawędzi danego węzła.

3.1.2 Member Data Documentation

3.1.2.1 edistance

```
type edge::edistance
```

waga krawędzi

3.1.2.2 peback

```
node* edge::peback
```

wskaźnik do węzła na początku krawędzi

3.1.2.3 peforw

```
node* edge::peforw
```

wskaźnik do węzła na końcu krawędzi

3.1.2.4 penext

```
edge* edge::penext
```

wskaźnik na następną krawędź danego węzła

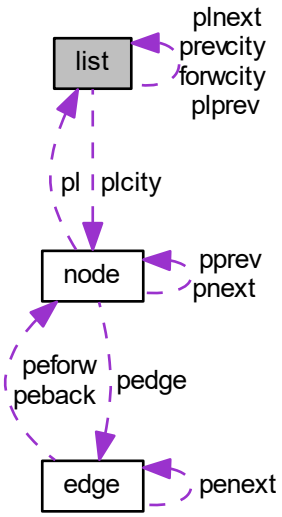
The documentation for this struct was generated from the following file:

- structures.h

3.2 list Struct Reference

`#include <structures.h>`

Collaboration diagram for list:



Public Attributes

- node ∗ plcity

  wskaźnik na węzeł grafu
- type cost

  obecny najniższy koszt dojścia do węzła
- list ∗ prevcity

  poprzedni węzeł w kolejce najkrótszej drogi
- bool qs

  zmienna reprezentuje przenoszenie węzła do zbioru już przeszukanych
- list ∗ plprev

  wskaźnik na poprzedni węzeł w kolejce
- list ∗ plnext

  wskaźnik na następny węzeł w kolejce
- list ∗ forwcity

  wskaźnik na następny węzeł w kolejce przy wyznaczaniu najkrótszej drogi

3.2.1 Detailed Description

Struktura reprezentująca kolejkę, w której zawarta jest najkrótsza trasa.

3.2.2 Member Data Documentation

3.2.2.1 cost

```
type list::cost
```

obecny najniższy koszt dojścia do węzła

3.2.2.2 forwcity

```
list* list::forwcity
```

wskaźnik na następny węzeł w kolejce przy wyznaczaniu najkrótszej drogi

3.2.2.3 plcity

```
node* list::plcity
```

wskaźnik na węzeł grafu

3.2.2.4 plnext

```
list* list::plnext
```

wskaźnik na następny węzeł w kolejce

3.2.2.5 plprev

```
list* list::plprev
```

wskaźnik na poprzedni węzeł w kolejce

3.2.2.6 prevcity

```
list* list::prevcity
```

poprzedni węzeł w kolejce najkrótszej drogi

3.2.2.7 qs

```
bool list::qs
```

zmienna reprezentuje przenoszenie węzła do zbioru już przeszukanych

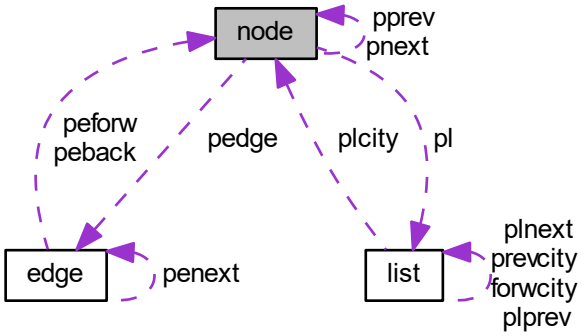
The documentation for this struct was generated from the following file:

- structures.h

3.3 node Struct Reference

```
#include <structures.h>
```

Collaboration diagram for node:



Public Attributes

- string city
  nazwa węzła
- edge * pedge
  wskaźnik na listę krawędzi węzła
- node * pprev
  wskaźnik na poprzedni węzeł
- node * pnext
  wskaźnik na następny węzeł
- list * pl
  wskaźnik na miejsce w kolejce węzła

3.3.1 Detailed Description

Struktura reprezentująca listę węzłów grafu.

3.3.2 Member Data Documentation

3.3.2.1 city

```
string node::city
```

nazwa węzła

3.3.2.2 pedge

```
edge* node::pedge
```

wskaźnik na listę krawędzi węzła

3.3.2.3 pl

```
list* node::pl
```

wskaźnik na miejsce w kolejce węzła

3.3.2.4 pnext

```
node* node::pnext
```

wskaźnik na następny węzeł

3.3.2.5 pprev

```
node* node::pprev
```

wskaźnik na poprzedni węzeł

The documentation for this struct was generated from the following file:

- structures.h

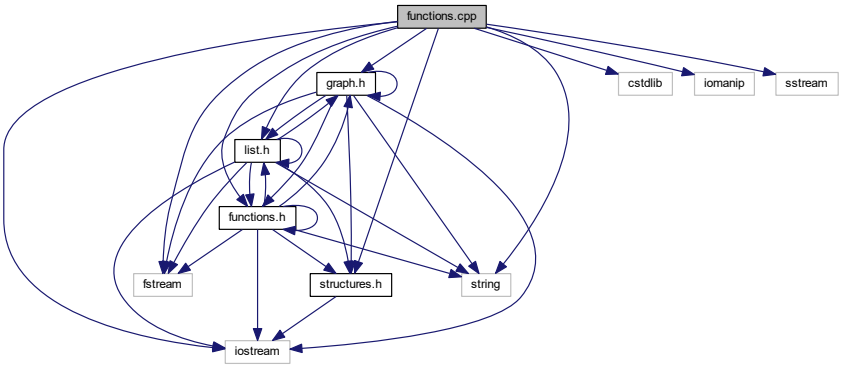
Chapter 4

File Documentation

4.1 functions.cpp File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
#include <iomanip>
#include <sstream>
#include "graph.h"
#include "structures.h"
#include "list.h"
#include "functions.h"
```

Include dependency graph for functions.cpp:



Functions

- bool switches (int argc, char *argv[ ], string *f1, string *f2, string *f3)
- list * searchfirstfalse (list *plhead)
- list * searchthelowest (list *plpointer)
- void cnv (list *plhead, list *plpointer)
- void dijkstra2 (node *phead, node *ptail, list *&plhead, list *&pltail, const string &arrival, const string &destination)

- bool correctroute (list ∗plarr, list ∗pldest)
- void printroute (ofstream &file3, list ∗plpointer, list ∗plnextp)
- void printsol (ofstream &file3, list ∗plarr, list ∗pldest)
- bool load (ifstream &file, node ∗&phead, node ∗&ptail)
- bool load2 (ifstream &file2, ofstream &file3, node ∗&phead, node ∗&ptail, list ∗&plhead, list ∗&pltail)

4.1.1 Function Documentation

4.1.1.1 cnv()

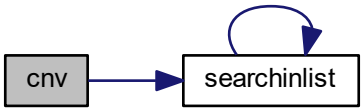
```
void cnv (
            list * plhead,
            list * plpointer )
```

Funkcja składowa funkcji dijkstra2, która odpowiada za relaksacje najkrótszych ścieżek.

Parameters

plhead	głowa kolejki
plpointer	wskaźnik na węzeł, z którego będą pobierane kolejne krawędzie grafu

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.1.2 correctroute()

```
bool correctroute (
            list * plarr,
            list * pldest )
```

Funkcja sprawdzająca najkrótszą trasę.

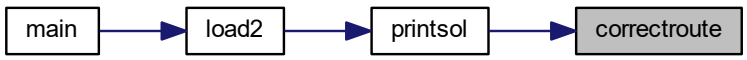
Parameters

plarr	wskaźnik na węzeł początkowy trasy
pldest	wskaźnik na węzeł końcowy trasy

Returns

Funkcja zwraca TRUE jeśli trasa jest niemożliwa do wyznaczenia, a FALSE jeśli udało się wyznaczyć trasę.

Here is the caller graph for this function:



4.1.1.3 dijkstra2()

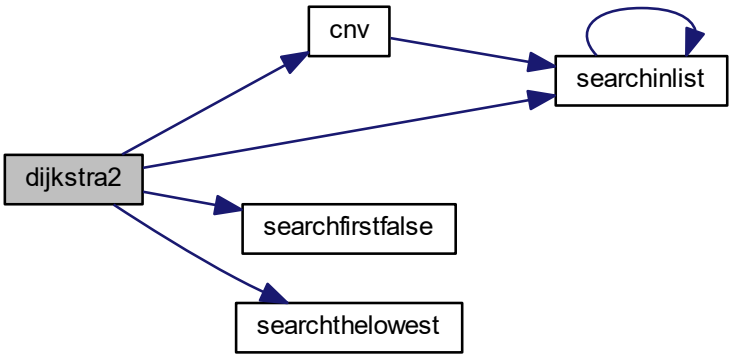
```
void dijkstra2 (
            node * phead,
            node * ptail,
            list *& plhead,
            list *& pltail,
            const string & arrival,
            const string & destination )
```

Funkcja odpowiadająca za realizację Algorytmu Dijkstry,w tym przenoszenia ze zbioru "do przeszukania" do zbioru "przeszukanych".

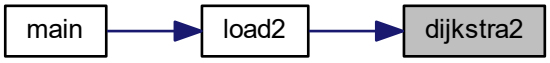
Parameters

phead	głowa listy, w której zwarty jest graf
ptail	ogon listy
plhead	głowa kolejki
pltail	ogon kolejki
arrival	węzeł początkowy trasy
destination	węzeł końcowy trasy

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.4 load()

```
bool load (
            ifstream & file,
            node *& phead,
            node *& ptail )
```

Funkcja sprawdzająca poprawność pliku mapy.

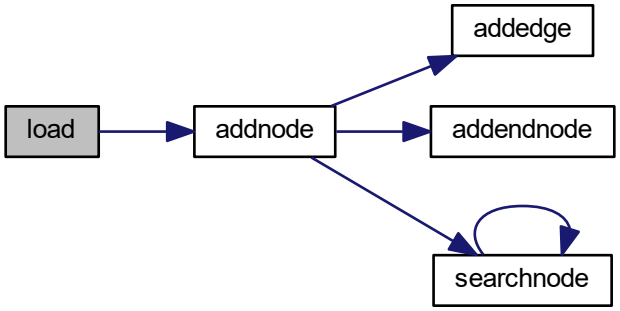
Parameters

file	plik wejściowy mapy
phead	głowa listy węzłów (grafu)
ptail	ogon listy węzłów

Returns

Funkcja zwraca TRUE jeśli plik z mapami został wczytany poprawnie, albo FALSE jeśli nie udało się go wczytać.

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.1.5 load2()

```
bool load2 (
            ifstream & file2,
            ofstream & file3,
            node *& phead,
            node *& ptail,
            list *& plhead,
            list *& pltail )
```

Funkcja odpowiadająca za odczyt z pliku zadanych tras oraz stwierdzenie jego poprawności, uruchamia też Algorytm Dijkstry i wpisuje do pliku wyjściowego.

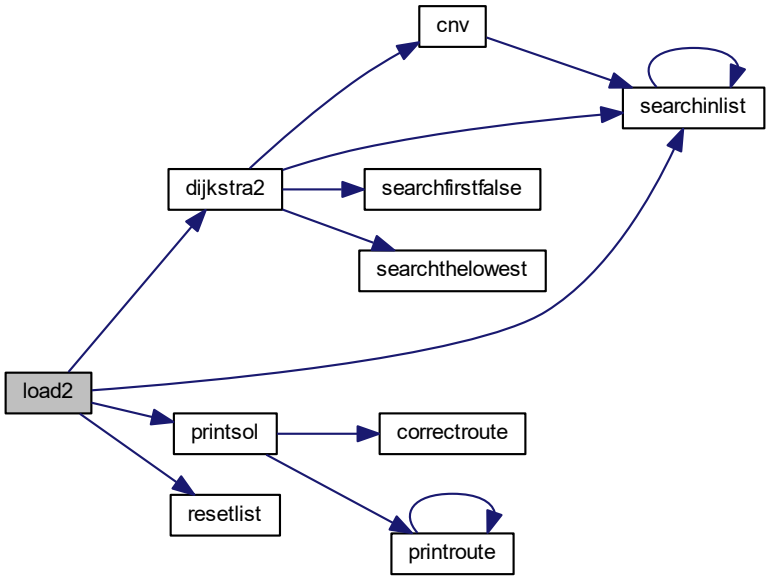
Parameters

file2	plik wejściowy z zadanymi trasami
file3	plik wyjściowy z wyznaczonymi trasami
phead	głowa listy węzłów(grafu)
ptail	ogon listy węzłów
plhead	głowa kolejki
pltail	ogon kolejki

Returns

Funkcja zwraca TRUE jeśli plik z zadanymi trasami został wczytany poprawnie, albo FALSE jeśli nie udało się go wczytać.

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.1.6 printroute()

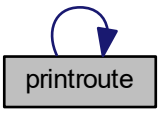
```
void printroute (
            ofstream & file3,
            list * plpointer,
            list * plnextp )
```

Funkcja składowa funkcji printroute, drukująca najkrótszą trasę do pliku.

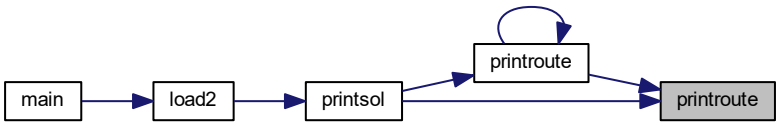
Parameters

file3	plik wyjściowy z wyznaczonymi trasami
plpointer	wskaźnik na obecny element najkrótszej trasy w kolejce
plnextp	wskaźnik na następny element najkrótszej trasy w kolejce

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.7 printsol()

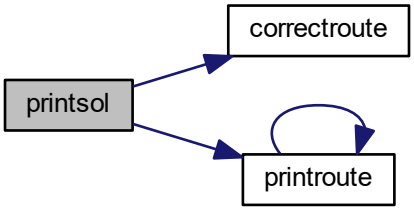
```
void printsol (
            ofstream & file3,
            list * plarr,
            list * pldest )
```

Funkcja drukująca trasę do pliku.

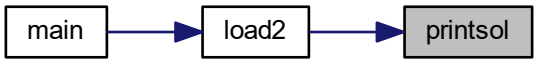
Parameters

file3	plik wyjściowy z wyznaczonymi trasami
plarr	węzeł początkowy
pldest	węzeł końcowy

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.8 searchfirstfalse()

```
list* searchfirstfalse (
            list * plhead )
```

Funkcja składowa funkcji dijkstra2, która wyszukuje pierwszy wierzchołek ze zbioru nieprzeszukanych wierzchołków.

Parameters

plhead	głowa kolejki

Returns

Funkcja zwraca wskaźnik na wyszukany element kolejki.

Here is the caller graph for this function:

4.1.1.9 searchthelowest()

```
list* searchthelowest (
            list * plpointer )
```

Funkcja składowa funkcji dijkstra2, która wyszukuje element o najmniejszym koszcie dojścia ze zbioru nieprzeszukanych elementów.

Parameters

plpointer	wskaźnik na pierwszy element ze zbioru nieprzeszukanych elementów w kolejce

Returns

Funkcja zwraca adres elementu kolejki ze zbioru nieprzeszukanych o najmniejszym koszcie dojścia.

Here is the caller graph for this function:

4.1.1.10 switches()

```
bool switches (
            int argc,
```

```
            char * argv[],
            string * f1,
            string * f2,
            string * f3 )
```

Funkcja sprawdza poprawność parametrów programu.

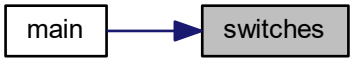
Parameters

argc	ilość przyjmowanych parametrów z konsoli
argv[]	wartości parametrów
f1	nazwa pliku wejściowego mapy
f2	nazwa pliku wejściowego zadanych tras
f3	nazwa pliku wyjściowego znalezionych tras

Returns

Funkcja zwraca TRUE kiedy któryś z przełączników jest wadliwy, a FALSE jeśli przełączniki zostały podpięte prawidłowo.

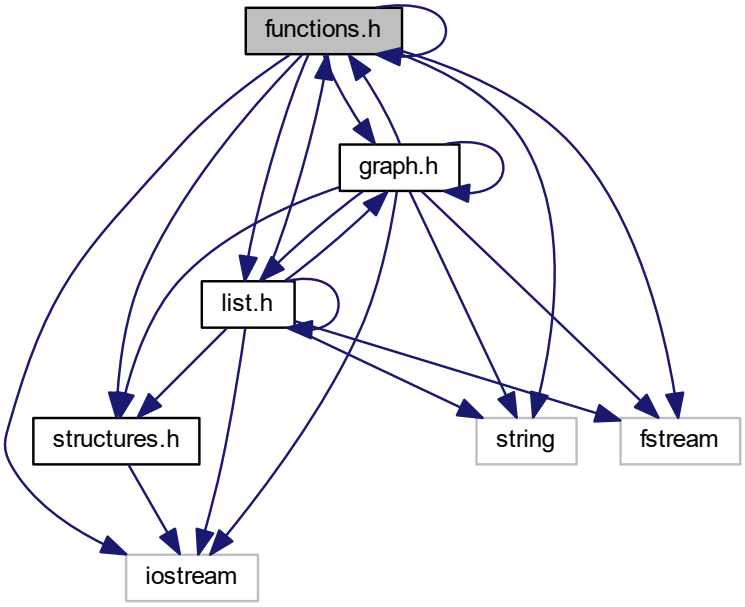
Here is the caller graph for this function:



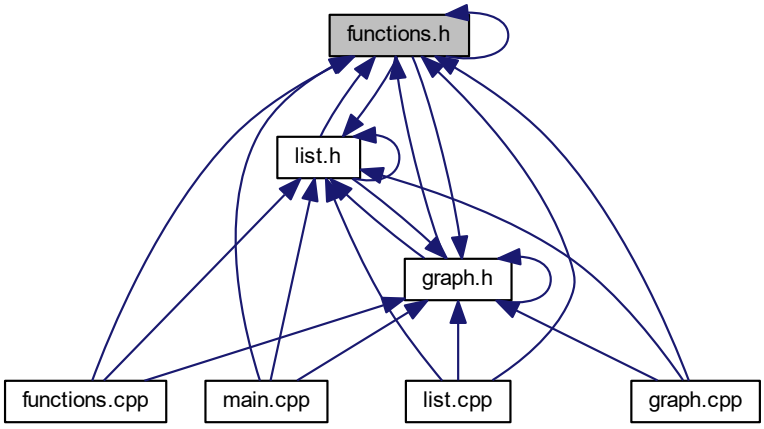
4.2 functions.h File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include "graph.h"
#include "structures.h"
#include "list.h"
#include "functions.h"
```

Include dependency graph for functions.h:



This graph shows which files directly or indirectly include this file:



Functions

- bool switches (int argc, char *argv[ ], string *f1, string *f2, string *f3)
- list * searchfirstfalse (list *plhead)
- list * searchthelowest (list *plpointer)

- void cnv (list ∗plhead, list ∗plpointer)
- void dijkstra2 (node ∗phead, node ∗ptail, list ∗&plhead, list ∗&pltail, const string &arrival, const string &destination)
- bool correctroute (list ∗plarr, list ∗pldest)
- void printroute (ofstream &file3, list ∗plpointer, list ∗plnextp)
- void printsol (ofstream &file3, list ∗plarr, list ∗pldest)
- bool load (ifstream &file, node ∗&phead, node ∗&ptail)
- bool load2 (ifstream &file2, ofstream &file3, node ∗&phead, node ∗&ptail, list ∗&plhead, list ∗&pltail)

4.2.1 Function Documentation

4.2.1.1 cnv()

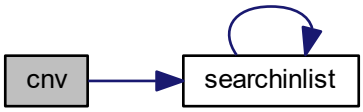
```
void cnv (
            list * plhead,
            list * plpointer )
```

Funkcja składowa funkcji dijkstra2, która odpowiada za relaksacje najkrótszych ścieżek.

Parameters

plhead	głowa kolejki
plpointer	wskaźnik na węzeł, z którego będą pobierane kolejne krawędzie grafu

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.1.2 correctroute()

```
bool correctroute (
            list * plarr,
            list * pldest )
```

Funkcja sprawdzająca najkrótszą trasę.

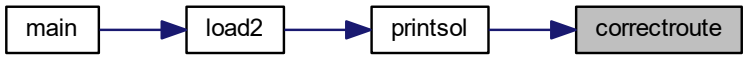
Parameters

plarr	wskaźnik na węzeł początkowy trasy
pldest	wskaźnik na węzeł końcowy trasy

Returns

Funkcja zwraca TRUE jeśli trasa jest niemożliwa do wyznaczenia, a FALSE jeśli udało się wyznaczyć trasę.

Here is the caller graph for this function:



4.2.1.3 dijkstra2()

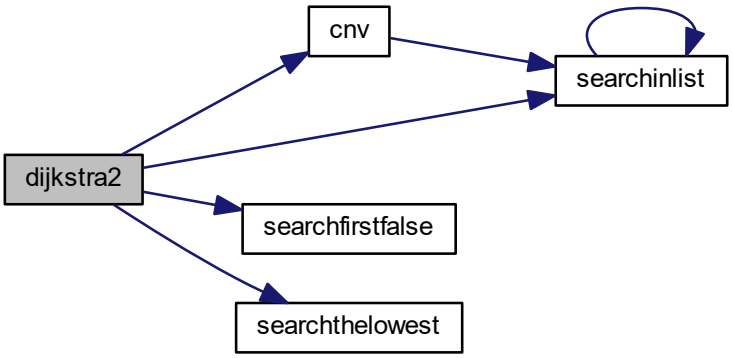
```
void dijkstra2 (
            node * phead,
            node * ptail,
            list *& plhead,
            list *& pltail,
            const string & arrival,
            const string & destination )
```

Funkcja odpowiadająca za realizację Algorytmu Dijkstry,w tym przenoszenia ze zbioru "do przeszukania" do zbioru "przeszukanych".

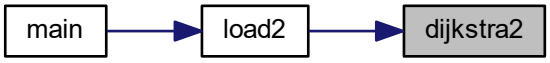
Parameters

phead	głowa listy, w której zwarty jest graf
ptail	ogon listy
plhead	głowa kolejki
pltail	ogon kolejki
arrival	węzeł początkowy trasy
destination	węzeł końcowy trasy

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.1.4 load()

```
bool load (
            ifstream & file,
            node *& phead,
            node *& ptail )
```

Funkcja sprawdzająca poprawność pliku mapy.

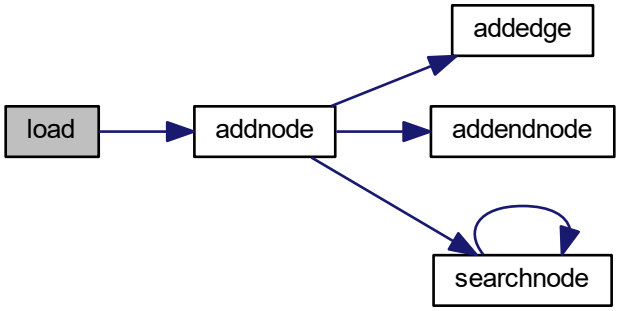
Parameters

file	plik wejściowy mapy
phead	głowa listy węzłów (grafu)
ptail	ogon listy węzłów

Returns

Funkcja zwraca TRUE jeśli plik z mapami został wczytany poprawnie, albo FALSE jeśli nie udało się go wczytać.

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.1.5 load2()

```
bool load2 (
            ifstream & file2,
            ofstream & file3,
            node *& phead,
            node *& ptail,
            list *& plhead,
            list *& pltail )
```

Funkcja odpowiadająca za odczyt z pliku zadanych tras oraz stwierdzenie jego poprawności, uruchamia też Algorytm Dijkstry i wpisuje do pliku wyjściowego.

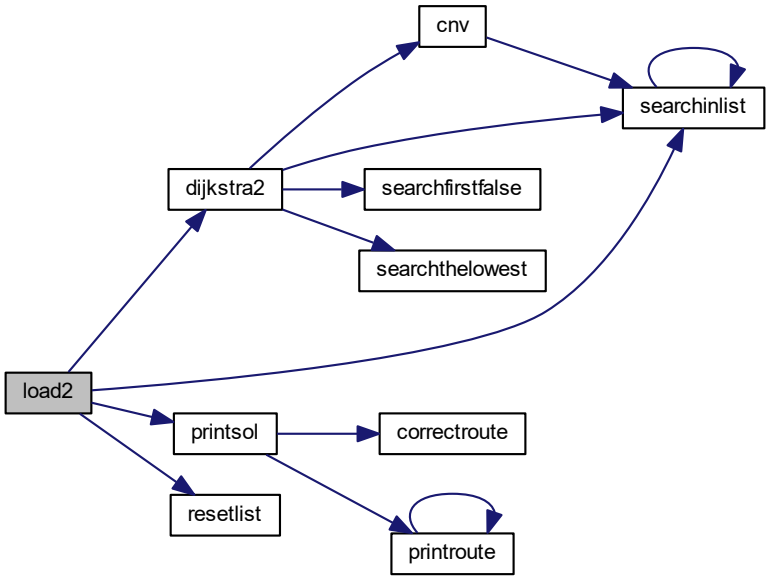
Parameters

file2	plik wejściowy z zadanymi trasami
file3	plik wyjściowy z wyznaczonymi trasami
phead	głowa listy węzłów(grafu)
ptail	ogon listy węzłów
plhead	głowa kolejki
pltail	ogon kolejki

Returns

Funkcja zwraca TRUE jeśli plik z zadanymi trasami został wczytany poprawnie, albo FALSE jeśli nie udało się go wczytać.

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.1.6 printroute()

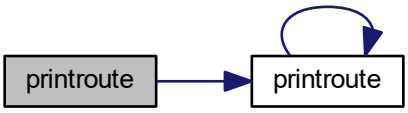
```
void printroute (
            ofstream & file3,
            list * plpointer,
            list * plnextp )
```

Funkcja składowa funkcji printroute, drukująca najkrótszą trasę do pliku.

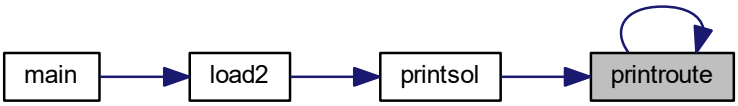
Parameters

file3	plik wyjściowy z wyznaczonymi trasami
plpointer	wskaźnik na obecny element najkrótszej trasy w kolejce
plnextp	wskaźnik na następny element najkrótszej trasy w kolejce

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.1.7 printsol()

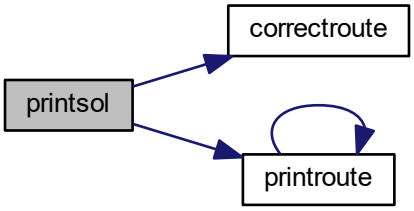
```
void printsol (
            ofstream & file3,
            list * plarr,
            list * pldest )
```

Funkcja drukująca trasę do pliku.

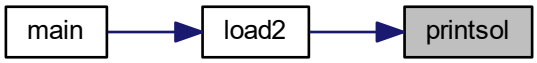
Parameters

file3	plik wyjściowy z wyznaczonymi trasami
plarr	węzeł początkowy
pldest	węzeł końcowy

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.1.8 searchfirstfalse()

```
list* searchfirstfalse (
            list * plhead )
```

Funkcja składowa funkcji dijkstra2, która wyszukuje pierwszy wierzchołek ze zbioru nieprzeszukanych wierzchołków.

Parameters

plhead	głowa kolejki

Returns

Funkcja zwraca wskaźnik na wyszukany element kolejki.

Here is the caller graph for this function:

4.2.1.9 searchthelowest()

```
list* searchthelowest (
            list * plpointer )
```

Funkcja składowa funkcji dijkstra2, która wyszukuje element o najmniejszym koszcie dojścia ze zbioru nieprzeszukanych elementów.

Parameters

plpointer	wskaźnik na pierwszy element ze zbioru nieprzeszukanych elementów w kolejce

Returns

Funkcja zwraca adres elementu kolejki ze zbioru nieprzeszukanych o najmniejszym koszcie dojścia.

Here is the caller graph for this function:

4.2.1.10 switches()

```
bool switches (
            int argc,
```

```
            char * argv[],
            string * f1,
            string * f2,
            string * f3 )
```

Funkcja sprawdza poprawność parametrów programu.

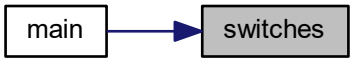
Parameters

argc	ilość przyjmowanych parametrów z konsoli
argv[]	wartości parametrów
f1	nazwa pliku wejściowego mapy
f2	nazwa pliku wejściowego zadanych tras
f3	nazwa pliku wyjściowego znalezionych tras

Returns

Funkcja zwraca TRUE kiedy któryś z przełączników jest wadliwy, a FALSE jeśli przełączniki zostały podpięte prawidłowo.

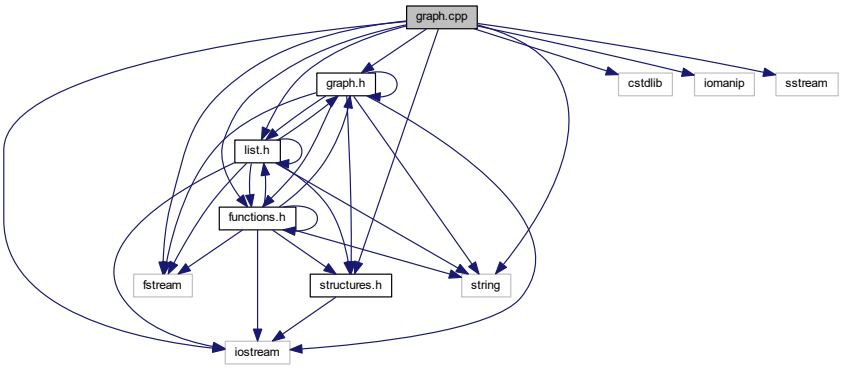
Here is the caller graph for this function:



4.3 graph.cpp File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
#include <iomanip>
#include <sstream>
#include "graph.h"
#include "structures.h"
#include "list.h"
#include "functions.h"
```

Include dependency graph for graph.cpp:



Functions

- void addnode (const type distance, const string &city1, const string &city2, node ∗&phead, node ∗&ptail)
- void addendnode (const string &city, node ∗&ptail)
- void addedge (const type distance, node ∗&peback, node ∗&peforw, edge ∗&pehead)
- node ∗ searchnode (node ∗phead, const string &city)
- void deletealledges (edge ∗&pehead)
- void deleteallnodes (node ∗&phead, node ∗&ptail)

4.3.1 Function Documentation

4.3.1.1 addedge()

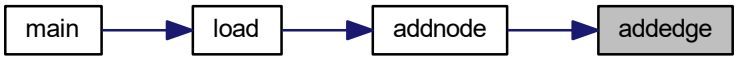
```
void addedge (
            const type distance,
            node *& peback,
            node *& peforw,
            edge *& pehead )
```

Funkcja dodaje krawędź grafu, połączoną z odpowiednimi węzłami.

Parameters

distance	waga krawędzi
peback	węzeł początkowy krawędzi
peforw	węzeł końcowy krawędzi
pehead	początek listy krawędzi danego węzła

Here is the caller graph for this function:



4.3.1.2 addendnode()

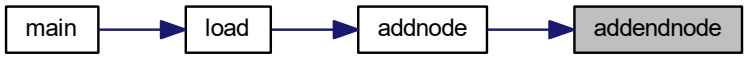
```
void addendnode (
            const string & city,
            node *& ptail )
```

Funkcja składowa funkcji addnode, która dodaje wierzchołek grafu na koniec listy.

Parameters

city	węzeł grafu
ptail	ogon listy

Here is the caller graph for this function:



4.3.1.3 addnode()

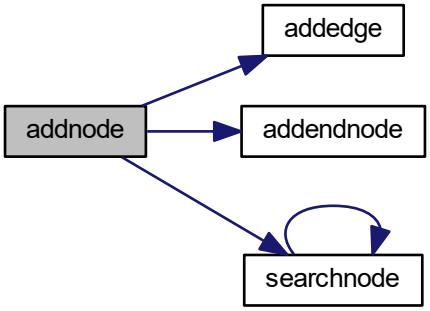
```
void addnode (
            const type distance,
            const string & city1,
            const string & city2,
            node *& phead,
            node *& ptail )
```

Funkcja dodaje wierzchołek grafu.

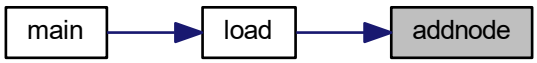
Parameters

distance	waga krawędzi
city1	węzeł początkowy krawędzi
city2	węzeł początkowy krawędzi
phead	początek listy mieszczącej graf
ptail	koniec listy mieszczącej graf

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.4 deletealledges()

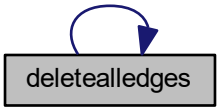
```
void deletealledges (
            edge *& pehead )
```

Funkcja usuwa listę krawędzi danego węzła.

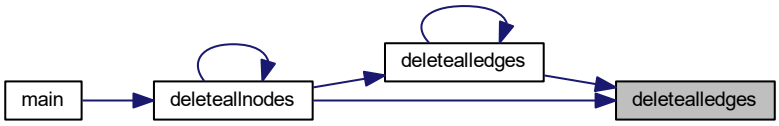
Parameters

pehead	początek listy krawędzi

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.5 deleteallnodes()

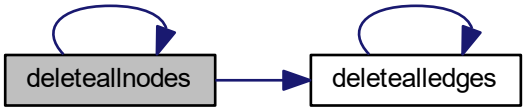
```
void deleteallnodes (
            node *& phead,
            node *& ptail )
```

Funkcja usuwa listę węzłów.

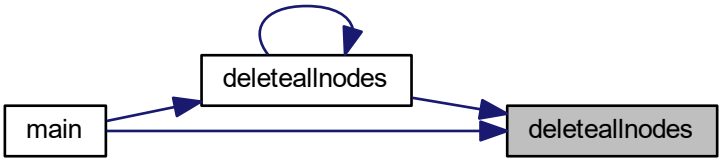
Parameters

phead	głowa listy węzłów
ptail	ogon listy węzłów

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.6 searchnode()

```
node* searchnode (
            node * phead,
            const string & city )
```

Funkcja wyszukuje wierzchołek w grafie.

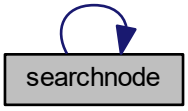
Parameters

phead	pierwszy element listy
city	szukany element listy

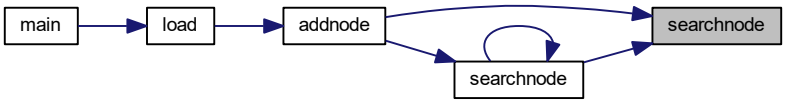
Returns

Funkcja zwraca adres danego elementu w liście.

Here is the call graph for this function:



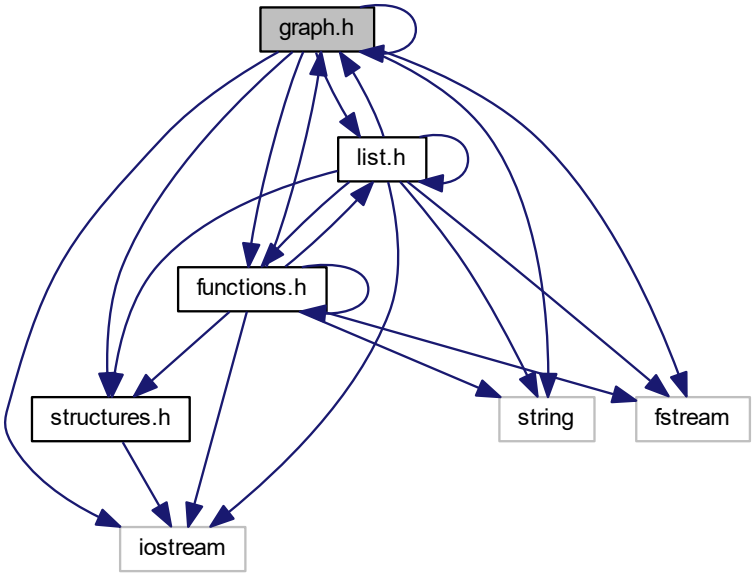
Here is the caller graph for this function:



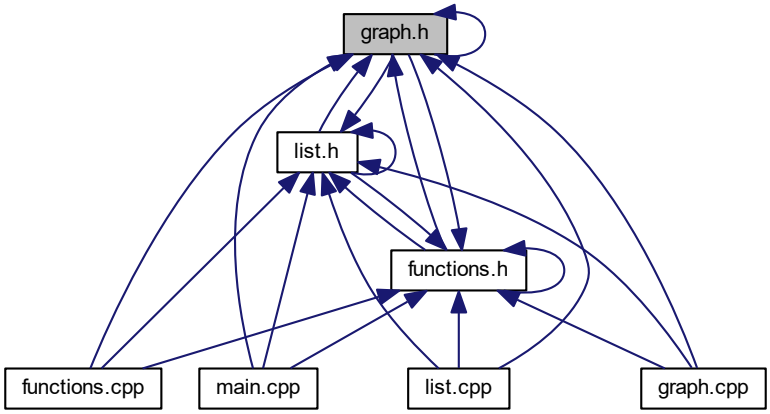
4.4 graph.h File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include "graph.h"
#include "structures.h"
#include "list.h"
#include "functions.h"
```

Include dependency graph for graph.h:



This graph shows which files directly or indirectly include this file:



Functions

- void addnode (const type distance, const string &city1, const string &city2, node ∗&phead, node ∗&ptail)
- void addendnode (const string &city, node ∗&ptail)
- void addedge (const type distance, node ∗&peback, node ∗&peforw, edge ∗&pehead)
- node ∗ searchnode (node ∗phead, const string &city)
- void deletealledges (edge ∗&pehead)
- void deleteallnodes (node ∗&phead, node ∗&ptail)

4.4.1 Function Documentation

4.4.1.1 addedge()

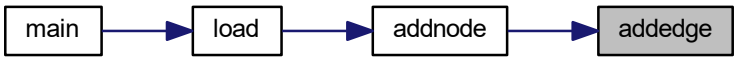
```
void addedge (
            const type distance,
            node *& peback,
            node *& peforw,
            edge *& pehead )
```

Funkcja dodaje krawędź grafu, połączoną z odpowiednimi węzłami.

Parameters

distance	waga krawędzi
peback	węzeł początkowy krawędzi
peforw	węzeł końcowy krawędzi
pehead	początek listy krawędzi danego węzła

Here is the caller graph for this function:



4.4.1.2 addendnode()

```
void addendnode (
            const string & city,
            node *& ptail )
```

Funkcja składowa funkcji addnode, która dodaje wierzchołek grafu na koniec listy.

Parameters

city	węzeł grafu
ptail	ogon listy

Here is the caller graph for this function:

4.4.1.3 addnode()

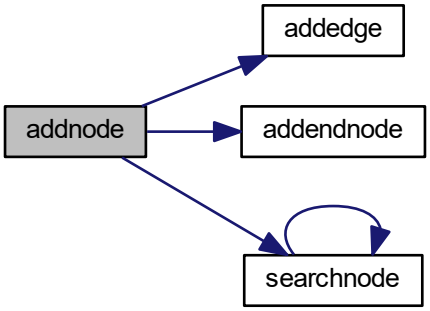
```
void addnode (
            const type distance,
            const string & city1,
            const string & city2,
            node *& phead,
            node *& ptail )
```

Funkcja dodaje wierzchołek grafu.

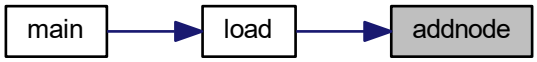
Parameters

distance	waga krawędzi
city1	węzeł początkowy krawędzi
city2	węzeł początkowy krawędzi
phead	początek listy mieszczącej graf
ptail	koniec listy mieszczącej graf

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.1.4 deletealledges()

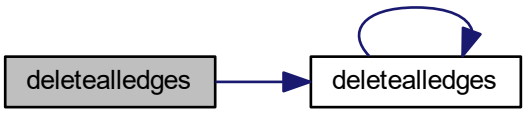
```
void deletealledges (
            edge *& pehead )
```

Funkcja usuwa listę krawędzi danego węzła.

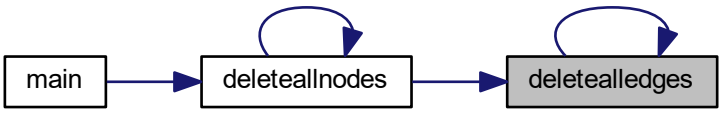
Parameters

pehead	początek listy krawędzi

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.1.5 deleteallnodes()

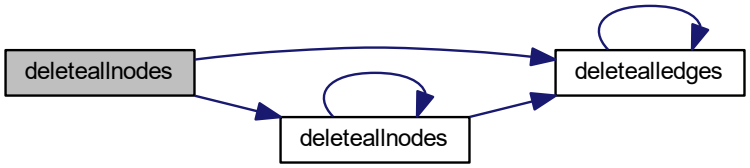
```
void deleteallnodes (
            node *& phead,
            node *& ptail )
```

Funkcja usuwa listę węzłów.

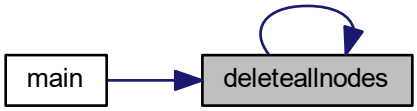
Parameters

phead	głowa listy węzłów
ptail	ogon listy węzłów

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.1.6 searchnode()

```
node* searchnode (
            node * phead,
            const string & city )
```

Funkcja wyszukuje wierzchołek w grafie.

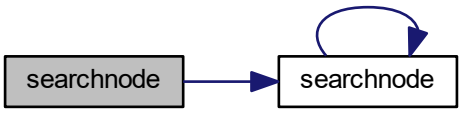
Parameters

phead	pierwszy element listy
city	szukany element listy

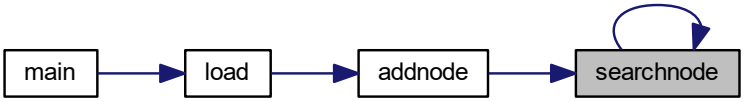
Returns

Funkcja zwraca adres danego elementu w liście.

Here is the call graph for this function:



Here is the caller graph for this function:



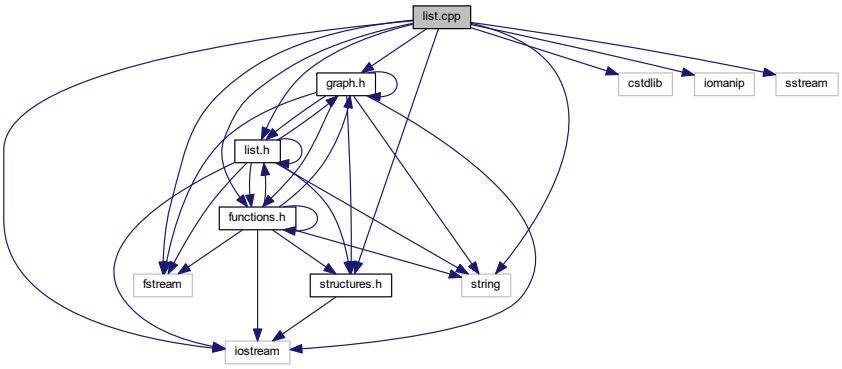
4.5 inputroutes.txt File Reference

4.6 list.cpp File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
#include <iomanip>
#include <sstream>
#include "graph.h"
#include "structures.h"
#include "list.h"
```

```
#include "functions.h"
```

Include dependency graph for list.cpp:



Functions

- void addtolist (node *plcity, list *&plhead, list *&pltail)
- list * searchinlist (list *plhead, const string &plcity)
- void deletelist (list *&plhead, list *&pltail)
- void resetlist (list *plhead)

4.6.1 Function Documentation

4.6.1.1 addtolist()

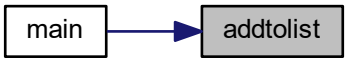
```
void addtolist (
            node * plcity,
            list *& plhead,
            list *& pltail )
```

Funkcja dodaje węzły grafu do kolejki według której odczytana zostanie najkrótsza trasa.

Parameters

plcity	głowa kolejki
pltail	ogon kolejki

Here is the caller graph for this function:



4.6.1.2 deletelist()

```
void deletelist (
            list *& plhead,
            list *& pltail )
```

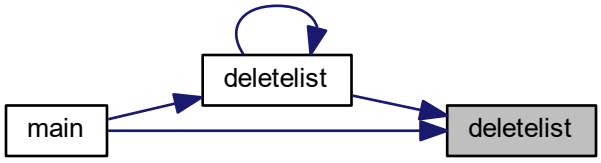
Funkcja kasuje kolejkę.

Parameters

plhead	głowa kolejki
ptail	ogon kolejki

Here is the call graph for this function:

Here is the caller graph for this function:



4.6.1.3 resetlist()

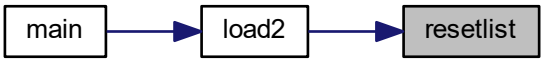
```
void resetlist (
            list * plhead )
```

Funkcja resetuje kolejkę do stanu bezpośrednio po wpisaniu do niej grafu.

Parameters

plhead	głowa kolejki

Here is the caller graph for this function:



4.6.1.4 searchinlist()

```
list* searchinlist (
            list * plhead,
            const string & plcity )
```

Funkcja wyszkuje dany element w kolejce.

Parameters

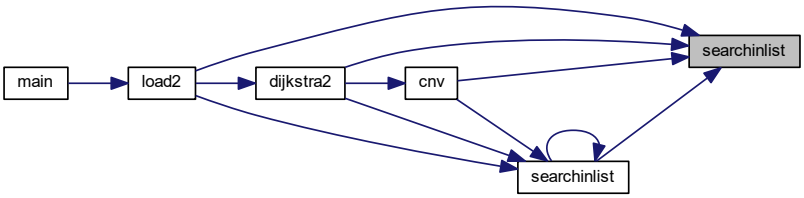
plhead	głowa kolejki
plcity	dany element grafu

Returns

Funkcja zwraca adres danego elementu kolejki.

Here is the call graph for this function:

Here is the caller graph for this function:

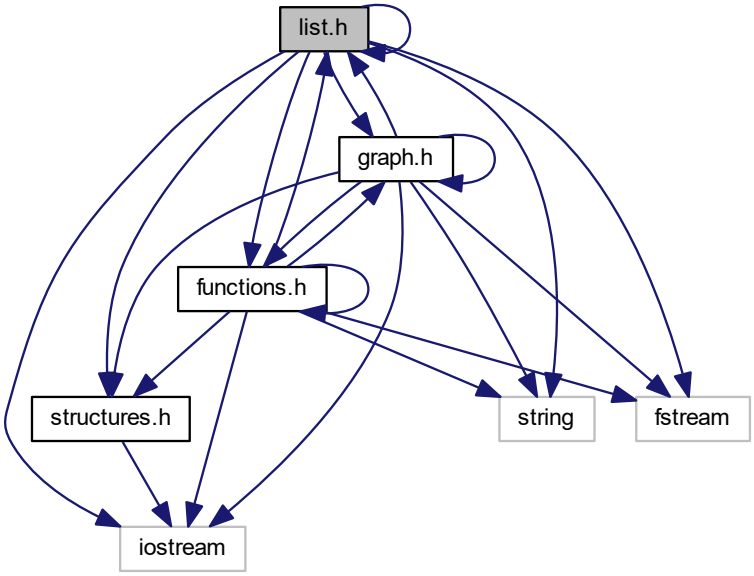


4.7 list.h File Reference

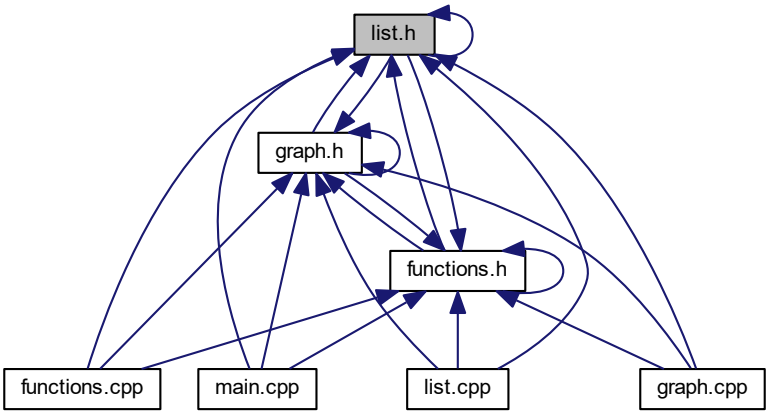
```
#include <iostream>
#include <string>
#include <fstream>
#include "graph.h"
#include "structures.h"
#include "list.h"
```

```
#include "functions.h"
```

Include dependency graph for list.h:



This graph shows which files directly or indirectly include this file:



Functions

- void addtolist (node *plcity, list *&plhead, list *&pltail)
- list * searchinlist (list *plhead, const string &plcity)
- void deletelist (list *&plhead, list *&pltail)
- void resetlist (list *plhead)

4.7.1 Function Documentation

4.7.1.1 addtolist()

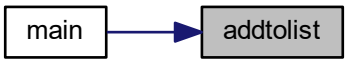
```
void addtolist (
            node * plcity,
            list *& plhead,
            list *& pltail )
```

Funkcja dodaje węzły grafu do kolejki według której odczytana zostanie najkrótsza trasa.

Parameters

plcity	głowa kolejki
pltail	ogon kolejki

Here is the caller graph for this function:



4.7.1.2 deletelist()

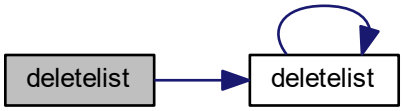
```
void deletelist (
            list *& plhead,
            list *& pltail )
```

Funkcja kasuje kolejkę.

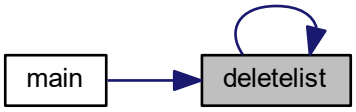
Parameters

plhead	głowa kolejki
ptail	ogon kolejki

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.1.3 resetlist()

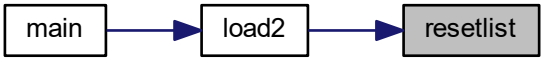
```
void resetlist (
            list * plhead )
```

Funkcja resetuje kolejkę do stanu bezpośrednio po wpisaniu do niej grafu.

Parameters

plhead	głowa kolejki

Here is the caller graph for this function:



4.7.1.4 searchinlist()

```
list* searchinlist (
            list * plhead,
            const string & plcity )
```

Funkcja wyszkuje dany element w kolejce.

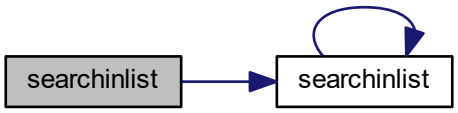
Parameters

plhead	głowa kolejki
plcity	dany element grafu

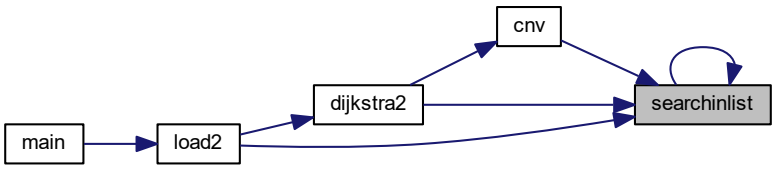
Returns

Funkcja zwraca adres danego elementu kolejki.

Here is the call graph for this function:



Here is the caller graph for this function:

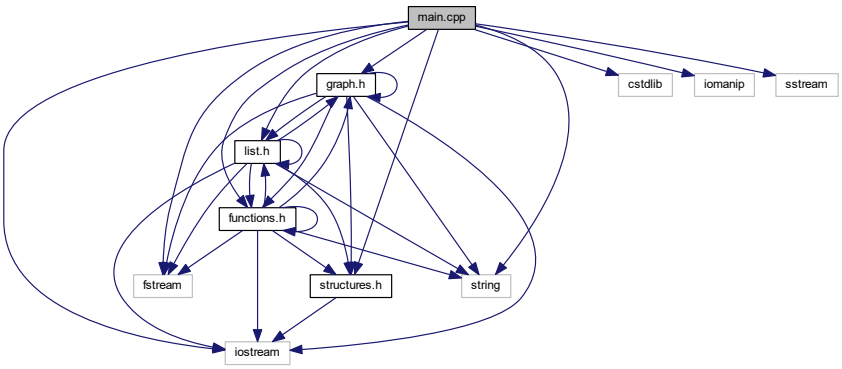


4.8 main.cpp File Reference

```
#include <iostream>
#include <string>
```

```
#include <fstream>
#include <cstdlib>
#include <iomanip>
#include <sstream>
#include "graph.h"
#include "structures.h"
#include "list.h"
#include "functions.h"
```

Include dependency graph for main.cpp:



Functions

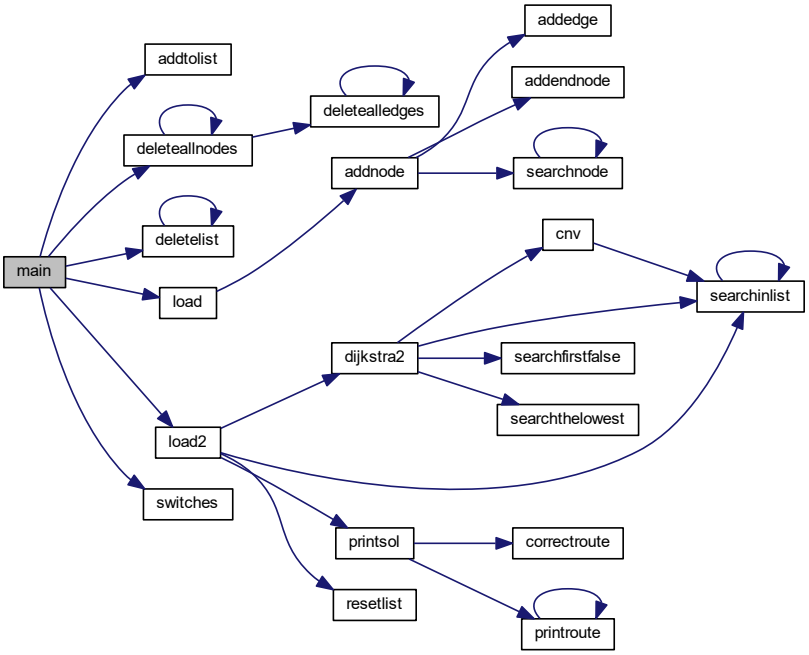
- int main (int argc, char ∗argv[ ])

4.8.1 Function Documentation

4.8.1.1 main()

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

Here is the call graph for this function:



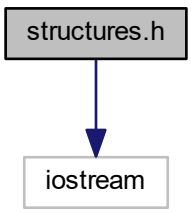
4.9 map.txt File Reference

4.10 outputroutes.txt File Reference

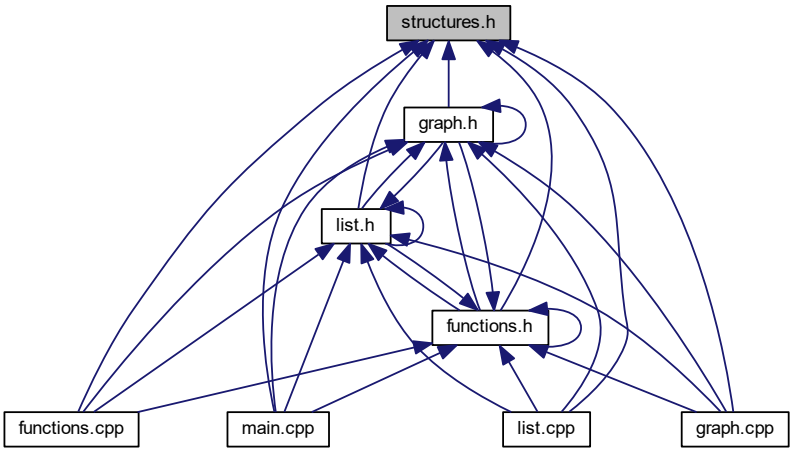
4.11 structures.h File Reference

```
#include <iostream>
```

Include dependency graph for structures.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct edge
- struct node
- struct list

Typedefs

- typedef int type

4.11.1 Typedef Documentation

4.11.1.1 type

```
typedef int type
```

Index