Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

# Podstawy Programowania Komputerów

## Dijkstra

| | |
|---|---|
| autor | Paulina Czapla |
| prowadzący | dr inż. Artur Pasierbek |
| rok akademicki | 2019/2020 |
| kierunek | informatyka |
| rodzaj studiów | SSI |
| semestr | 1 |
| termin laboratorium | czwartek, 08:15 – 09:45 |
| sekcja | 41 |
| termin oddania sprawozdania | 2019-01-16 |

# 1 Treść zadania

Napisać program, do znajdowania najkrótszych ścieżek między zadanym wierzchołkiem grafu, a wszystkimi pozostałymi wierzchołkami tego grafu. Program wykorzystuje algorytm Dijkstry. Pierwszym plikiem wejściowym jest plik z grafem, drugim jest plik z numerami wierzchołków, dla których chcemy wyznaczyć najkrótsze odległości do pozostałych wierzchołków. W pliku wynikowym zostaną zapisane trasy o minimalnej długości dla zadanych wierzchołków.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

- `-g` plik wejściowy z grafem
- `-w` plik wejściowy z wierzchołkami
- `-o` plik wyjściowy z wynikami

Przykładowy plik z grafem:

```
3 -> 2 : 54.5
12 -> 3 : 4.5
2 -> 5 : 34.65
5 -> 3 : 2.4
3 -> 12 : 1.00
```

Przykładowy plik z wierzchołkami:

```
2
6
12
```

W pliku wynikowym zostaną zapisane trasy o minimalnej długości dla zadanych wierzchołków, np.

```
wierzcholek startowy: 2
2 -> 5 -> 3 : 37.05
2 -> 5 : 34.65
2 -> 5 -> 3 -> 12 : 38.05

wierzcholek startowy: 6
brak wierzcholka 6 w grafie
```

```
wierzcholek startowy: 12
12 -> 3 : 4.5
12 -> 3 -> 2 : 59.0
12 -> 3 -> 2 -> 5 : 93.65
```

# 2 Analiza zadania

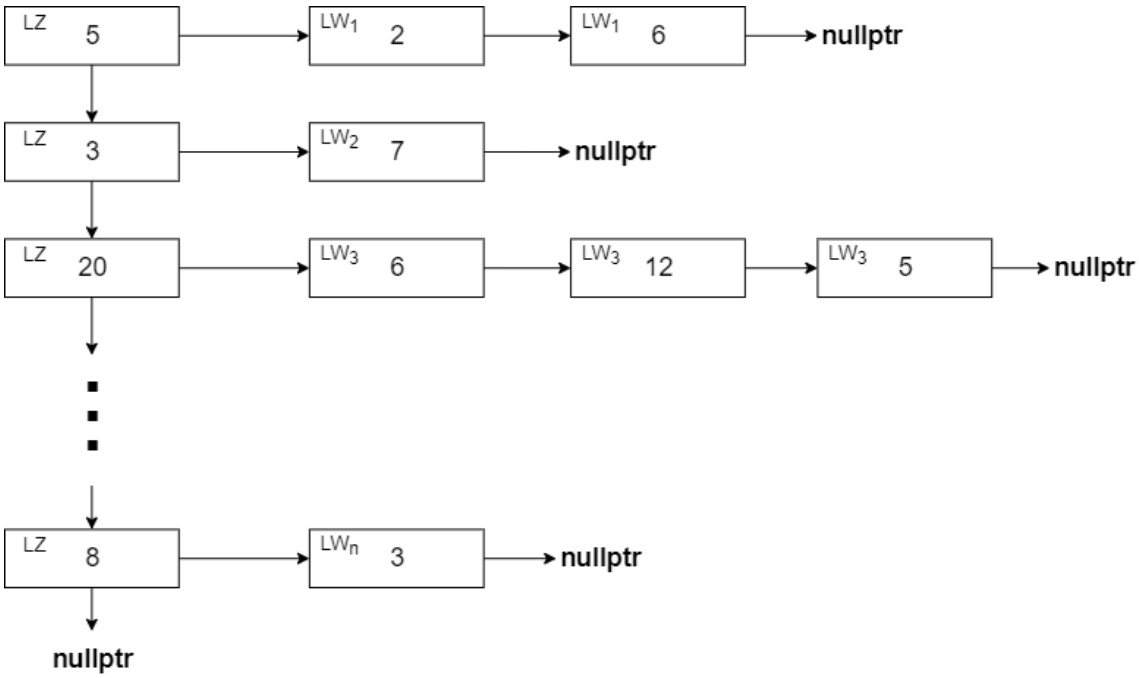
Zagadnienie przedstawia problem wybierania najkrótszej drogi między wierzchołkami przy użyciu algorytmu dijkstry oraz reprezentacji grafu odczytanego z pliku w programie.

## 2.1 Struktury danych

W projekcie wykorzystano strukturę listy list (listy sąsiedztwa) do reprezentacji grafu. Jednokierunkowa lista zewnętrzna, której elementami są struktury element_LZ, przechowuje numery wszystkich wierzchołków w grafie, najkrótszą drogę z wierzchołka początkowego (przy wpisywaniu grafu do listy, wartość ta jest równa maksymalnej wartości zmiennej typu int) oraz informację w postaci zmiennej typu bool o tym, czy dany wierzchołek został już odwiedzony (czy została do niego wyznaczona najkrótsza ścieżka), która początkowo jest równa false. Każdy element listy zewnętrznej posiada wskaźniki na swojego następnika oraz na pierwszy element listy wewnętrznej. Jednokierunkowe listy wewnętrzne, których elementami są struktury element_LW, przechowują numery wierzchołków sąsiednich w grafie dla danego wierzchołka z listy zewnętrznej oraz długość krawędzi od niego.

Użycie struktury listy list zdecydowanie ułatwia reprezentację oraz przeszukiwanie grafu do znajdowania najkrótszej ścieżki.

Na rysunku 1. została przedstawiona lista sąsiedztwa, której elementami listy zewnętrznej LZ są wierzchołki 5, 3, 20, ... ,8. Wierzchołkami sąsiednimi dla 5 są wierzchołki o numerach 2 i 6, które są elementami listy LW1.




Rysunek 1: Lista sąsiedztwa.

Ponadto, została użyta jeszcze jedna lista kierunkowa składająca się ze struktur sciezka. Zostają do niej zapisywane kolejno wierzchołki w ścieżce, od ostatniego do początkowego, przy czym każdy nowy węzeł staje się początkiem listy. Pomaga ona w ułożeniu wierzchołków w odpowiedniej kolejności oraz ułatwia zapis do pliku.

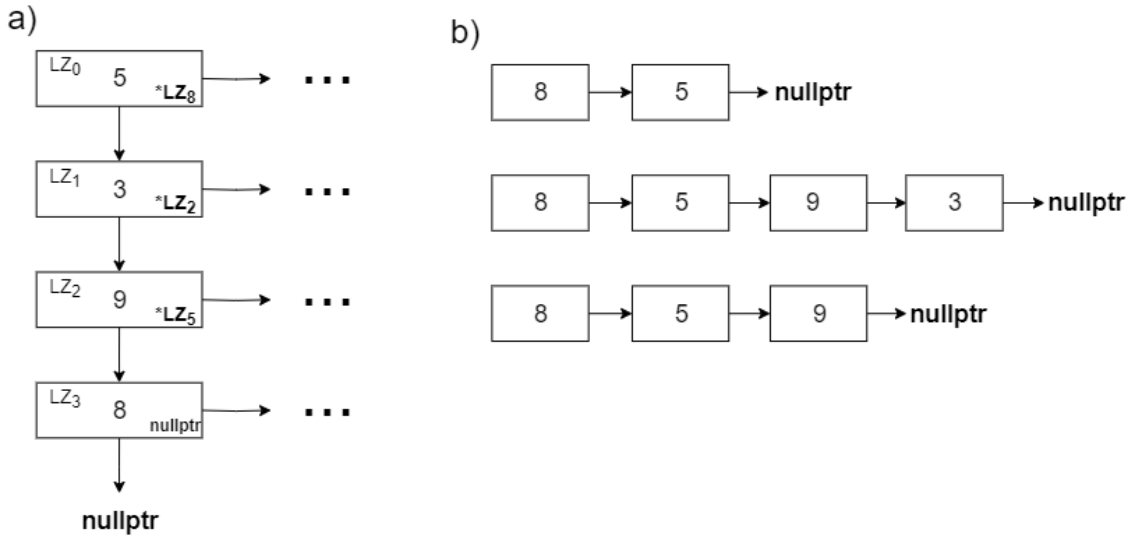
## 2.2 Algorytmy

Program odnajduje najkrótszą ścieżkę, od danego wierzchołka grafu do reszty wierzchołków, za pomocą algorytmu dijkstry. Początkowo droga do wierzchołka początkowego jest równa zero, natomiast do reszty wierzchołków ma maksymalną wartość zmiennej typu int. Zmienna **bool** odwiedzone jest dla nich równa **false**, a do wierzchołka początkowego **true**. Następnie algorytm sprawdza rekurencyjnie kolejnych sąsiadów wierzchołka, i dla każdego z nich (do którego droga z wierzchołka początkowego istnieje) zmienia wartość drogi na najkrótszą oraz przypisuje wskaźnik na poprzednika na ścieżce. Algorytm wykonuje się do momentu, aż nie zostaną odnalezione najkrótsze ścieżki do wszystkich dostępnych wierzchołków w grafie. Użycie algorytmu dijkstry było wymagane w projekcie. Szczegółowy opis działania funkcji, wykorzystującej ten algorytm w projekcie, znajduje się w załączonej dokumentacji.

Innym algorytmem wartym opisania, jest algorytm zawarty w funkcji wczytaj_sciezke . Wpisuje on do listy sciezka wierzchołek grafu oraz kolejno jego poprzedników, aż nie dotrze do wierzchołka początkowego, z którego wyznaczana jest ścieżka. Nowy element listy, staje się jej początkiem, co pozwala na ułożenie ścieżki w odpowiedniej kolejności do wpisania do pliku. Algorytm wykonuje się dla każdego wierzchołka z listy zewnętrznej, oprócz wierzchołka początkowego. Jeśli z danego wierzchołka nie ma drogi do wierzchołka początkowego, lista nie zostaje utworzona.

Na rysunku 2a została graficznie przedstawiona przykładowa, uproszczona lista zewnętrzna po wykonaniu algorytmu dijkstry. W lewym górnym rogu bloku, znajduje się numer elementu listy LZ, natomiast w prawym dolnym znajduje się wskaźnik na poprzednika na ścieżce. Wierzchołkiem początkowym, odczytanym z pliku, jest 8. Na rysunku 2b przestawione zostały listy ścieżki, które powstaną dla danego grafu i wierzchołka początkowego. W takiej kolejności wierzchołki zostaną wpisane do pliku, w odpowiedniej formie, co następuje po wpisaniu ścieżki do listy.

Po wpisaniu danych do pliku wynikowego, lista zostaje usunięta (pamięć zostaje zwolniona) i algorytm wykonuje się od nowa, aż nie zostaną sprawdzone wszystkie węzły listy zewnętrznej oprócz wierzchołka początkowego.




Rysunek 2: a) przykładowa lista zewnętrzna po zastosowaniu algorytmu dijkstry, b) listy ścieżki, które powstaną dla tego grafu.

# 3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików: wejściowego z grafem, wejściowego z wierzchołkami i wyjściowego po odpowiednich przełącznikach (odpowiednio: `-g` dla pliku z grafem `-w` dla pliku z wierzchołkami i `-o` dla pliku wyjściowego), np.

```
dijkstra.exe -g graf.txt -w wierzcholki.txt -o wynik.txt
dijkstra.exe -o wynik.txt -w wierzcholki.txt -g graf.txt
```

Przełączniki mogą być podane w dowolnej kolejności. Uruchomienie programu z niewystarczającą liczbą parametrów (mniejszą niż 7), lub z nieprawidłowymi przełącznikami, np.

```
dijkstra.exe -o wyjsciowy.txt -g graf.txt
dijkstra.exe -m wynik.txt -x wierzcholki.txt -g graf.txt
```

powoduje wyświetlenie krótkiej pomocy:

```
Podano nieprawidlowe argumenty.
dijkstra.exe    -  nazwa pliku
- g <nazwa.txt> -  plik wejsciowy z grafem
- w <nazwa.txt> -  plik wejsciowy z wierzcholkami
- o <nazwa.txt> -  plik wyjsciowy z wynikami
```

Pliki są plikami tekstowymi z rozszerzeniem .txt. Jeśli argumenty zostaną podane prawidłowo, ale plik nie będzie plikiem tekstowym, zostanie wyświetlony komunikat:

```
Nieprawidlowy format pliku. Pliki musza byc plikami tekstowymi.
```

Podanie nieprawidłowej nazwy pliku (plik nie istnieje) powoduje wyświetlenie odpowiedniego komunikatu:

```
Nie udalo sie otworzyc pliku <nazwa pliku>. Wprowadz dane jeszcze raz.
```

Jeśli pliki mają dopuszczalny format oraz otwierają się prawidłowo, ale mają błędne dane np.

Prawidłowa linia z krawędzią grafu:

```
2 -> 5 : 34.65
```

Przykłady błędnego opisu krawędzi grafu:

```
2 -> 5 : -34
2 - ddf : 6
tekst tekst tekst
2 ->  : 4.65
3 - 8 :
```

Przykład błędnego pliku z wierzchołkami:

```
4
wierzcholek
6
7
```

W przypadku błędnej zawartości pliku z grafem bądź wierzchołkami, zostanie wyświetlony komunikat:

```
Bledny plik!
```

Jeśli wszystkie parametry zostały podane prawidłowo, pliki się otwarły oraz są prawidłowe, to zostaje wyświetlona informacja:

```
Program zostal wykonany.
```

# 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 przelaczniki . Służy ona do obsługi przełączników, oraz sprawdza czy zostały podane prawidłowo. W przypadku nieprawidłowego wywołania programu, zostaje wypisany stosowny komunikat i program się kończy. W przelaczniki zostaje wywołana funkcja sprawdz_pliki , która sprawdza czy podane pliki są plikami tekstowymi. Jeśli nie, funkcja zwraca **false** i program się kończy. Kolejną wywoływaną funkcją jest szukaj_drogi . Na początku funkcja otwiera plik z wierzchołkami i sprawdza czy otwiera się on prawidłowo. Jeśli nie, funkcja zwraca **false**, w przeciwnym przypadku zostaje otwarty plik z grafem. Jeśli zostanie on otwarty prawidłowo, zostaje przekazany do funkcji wczytaj_graf . W przeciwnym przypadku, kończy się program. Funkcja wczytaj_graf wczytuje graf z pliku do listy sąsiedztwa. Sprawdzana jest poprawność pliku. Węzły listy zewnętrznej tworzone są za pomocą funkcji dodaj_elementLZ (tworzy węzeł) oraz dodaj_wezelLZ (dodaje nowy węzeł do istniejącej już listy). Połączenie pomiędzy listą zewnętrzną a wewnętrzną powstaje przy pomocy funkcji dodaj_polaczenie . Po wczytaniu grafu, plik zostaje zamknięty.

Jeśli plik z wierzchołkami bądź plik z grafem nie otworzył się prawidłowo, program się kończy po uprzednim wyświetleniu komunikatu.Po zamknięciu pliku z grafem, następuje odczyt numeru pierwszego wierzchołka z pliku z

wierzchołkami i zostaje on wyszukany w liście zewnętrznej za pomocą funkcji znajdzLZ. Zostaje również otwarty plik wyjściowy. Jeśli w grafie nie ma takiego wierzchołka, zostaje wywołana funkcja wczytaj_sciezke z odpowiednimi parametrami. Jeśli został on znaleziony, zostaje wykonany algorytm dijkstry za pomocą funkcji dijkstra , a następnie zostaje wywołana funkcja wczytaj_sciezke . W tej funkcji ścieżki z wierzchołka początkowego do reszty wierzchołków zapisywane są w liście sciezka , która tworzona jest za pomocą dodaj_wezel (tworzy nowy węzeł) oraz dodaj_wierzcholek (dodaje nowy węzeł do już istniejącej listy). Następnie wczytaj_sciezke wczytuje zawartość listy do pliku wyjściowego i usuwa list za pomocą usun_liste_wierzcholkow , aż nie zostaną wpisane wszystkie drogi z danego wierzchołka początkowego.

Po zakończeniu wczytywania danych do pliku, w funkcji szukaj_drogi zostaje wywołane wyczysc_dane, które resetuje dane w strukturze i przygotowuje ją do wykonania algorytmu dijkstry dla następnego wierzchołka. Po odczytaniu wszystkich wierzchołków, plik z wierzchołkami i plik wyjściowy zostają zamknięte. Na koniec zostaje wywołana funkcja usun_graf, która usuwa strukturę listy list, wykorzystanej do reprezentacji grafu, zwalniając zaalokowaną pamięć.
Ponadto, w przypadku błędnych danych w plikach wejściowych, lista ścieżki bądź lista sąsiedztwa zostają usuwane za pomocą odpowiednich funkcji.

## 4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączonej dokumentacji.

# 5 Testowanie

Program został przetestowany na wprowadzanie nieprawidłowych przełączników, nieprawidłowych nazw plików (nieistniejących w folderze) oraz plików w nieprawidłowym formacie. W każdym z tych przypadków, program kończy swoje działanie, wyświetlając stosowny komunikat o błędzie. Przy kompilacji program zawsze wyświetla komunikat o nieprawidłowej liczbie argumentów. Sprawdzone zostało również działanie na różnego rodzaju plikach. Projekt wykonuje się przy zarówno prawidłowych,typowych plikach wejściowych jak i nietypowych oraz skomplikowanych. Pliki niepoprawne (niezawierające liczb, zawierające liczby w niepoprawnym formacie niezgodne ze, specyfikacją np. ujemne wagi krawędzi) powodują zgłoszenie błędu, a plik wynikowy nie tworzy się.

Program został sprawdzony pod kątem wycieków pamięci.

# 6 Wnioski

Program do znajdowania najkrótszych ścieżek z danego wierzchołka początkowego do reszty wierzchołków w grafie przy użyciu algorytmu dijkstry jest programem prostym, chociaż wymaga samodzielnego zarządzania pamięcią. Najbardziej wymagającą częścią projektu okazało się zaprojektowanie odpowiedniej struktury do reprezentacji grafu oraz dopasowanie implementacji algorytmu dijkstry do tej struktury. Realizacja projektu była doskonałą okazją do poznania dynamicznych struktur danych w praktyce.

# Projekt zaliczeniowy z PPK - SSI

Wygenerowano przez Doxygen 1.8.16

# Rozdział 1

# Dijkstra

**Autor**

Paulina Czapla [SSI] [INF] [SEM1] [GR4]

**Data**

14.01.2020

# Rozdział 2

# Indeks klas

## 2.1 Lista klas

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

# Rozdział 3

# Indeks plików

## 3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

# Rozdział 4

# Dokumentacja klas

## 4.1 Dokumentacja struktury element_LW

`#include <struktury.h>`

### Atrybuty publiczne

- int wierzcholek

  *numer wierzchołka sąsiedniego*
- double wartosc

  *wartość drogi do danego wierzchołka*
- element_LW * nastepny

  *adres następnego elementu listy*

### 4.1.1 Opis szczegółowy

Węzeł listy wewnętrznej listy list.

### 4.1.2 Dokumentacja atrybutów składowych

#### 4.1.2.1 nastepny

```
element_LW* element_LW::nastepny
```

#### 4.1.2.2 wartosc

```
double element_LW::wartosc
```

#### 4.1.2.3 wierzcholek

```
int element_LW::wierzcholek
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

## 4.2 Dokumentacja struktury element_LZ

```
#include <struktury.h>
```

### Atrybuty publiczne

- int pierwszy_wierzcholek

  *numer wierzchołka w grafie*
- bool odwiedzone

  *flaga odwiedzenia wierzchołka*
- double droga

  *najkrótsza droga do danego wierzchołka z wierzchołka początkowego*
- element_LZ * poprzedni_wierzcholek

  *adres na poprzedni wierzchołek w ścieżce*
- element_LZ * nastepny

  *adres na następny element listy*
- element_LW * polaczenie

  *adres na pierwszy element listy wewnętrznej*

### 4.2.1 Opis szczegółowy

Węzeł listy zewnętrznej listy list.

### 4.2.2 Dokumentacja atrybutów składowych

#### 4.2.2.1 droga

```
double element_LZ::droga
```

#### 4.2.2.2 nastepny

```
element_LZ* element_LZ::nastepny
```

#### 4.2.2.3 odwiedzone

```
bool element_LZ::odwiedzone
```

#### 4.2.2.4 pierwszy_wierzcholek

```
int element_LZ::pierwszy_wierzcholek
```

#### 4.2.2.5 polaczenie

```
element_LW* element_LZ::polaczenie
```

#### 4.2.2.6 poprzedni_wierzcholek

```
element_LZ* element_LZ::poprzedni_wierzcholek
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

## 4.3 Dokumentacja struktury sciezka

```
#include <struktury.h>
```

### Atrybuty publiczne

- int wierzcholek

  *numer wierzchołka*
- double waga

  *długość ścieżki od początkowego do danego wierzchołka*
- sciezka * nastepny

  *adres następnego elementu w liście*

### 4.3.1 Opis szczegółowy

Węzeł listy wierzchołków najkrótszej ścieżki.

### 4.3.2 Dokumentacja atrybutów składowych

#### 4.3.2.1 nastepny

```
sciezka* sciezka::nastepny
```

#### 4.3.2.2 waga

```
double sciezka::waga
```

#### 4.3.2.3 wierzcholek

```
int sciezka::wierzcholek
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

# Rozdział 5

# Dokumentacja plików

## 5.1 Dokumentacja pliku dijkstra.cpp

```
#include "dijkstra.h"
#include "struktury.h"
```

### Funkcje

- bool przelaczniki (int argc, char ∗argv[ ])
- bool sprawdz_pliki (string plik1, string plik2, string plik3)
- sciezka ∗ dodaj_wezel (int wartosc, double waga)
- void dodaj_wierzcholek (sciezka ∗&glowa, int wartosc, double waga)
- void usun_liste_wierzcholkow (sciezka ∗&glowa)
- element_LZ ∗ dodaj_elementLZ (int wartosc)
- void dodaj_wezelLZ (element_LZ ∗&glowa, int wartosc)
- element_LZ ∗ znajdzLZ (int wartosc, element_LZ ∗p)
- element_LW ∗ dodaj_elementLW (double wartosc, int wezel)
- void dodaj_wezelLW (element_LW ∗&LW, int wierzcholek, double wartosc)
- void dodaj_polaczenie (element_LZ ∗&LZ, double waga, int wierzcholek, bool skierowana)
- void usun_graf (element_LZ ∗&poczatek)
- element_LZ ∗ wczytaj_graf (ifstream &plik)
- void wyczysc_dane (element_LZ ∗p)
- void dijkstra (element_LZ ∗&tmp_wierzcholek, int numer, element_LZ ∗glowa)
- bool szukaj_drogi (string wierzcholki, string graf, string wyniki)
- void wczytaj_sciezke (int numer, element_LZ ∗&glowa, bool czy_wystepuje, ofstream &plik)

### 5.1.1 Dokumentacja funkcji

#### 5.1.1.1 dijkstra()

```
void dijkstra (
            element_LZ *& tmp_wierzcholek,
            int numer,
            element_LZ * glowa )
```

Funkcja wykonująca algorytm dijkstry, wyszukuje najkrótszą drogę z podanego wierzchołka do reszty wierzchołków. Zostaje wywołana dla danego wierzchołka początkowego, który zostaje wyszukany w liście LZ. Początkowo w liście LZ wartość drogi do wierzchołka początkowego jest równa zero, natomiast do reszty wierzchołków jest równa maksymalnej wartości zmiennej typu int ("nieskonczonosc"). Następnie sprawdzany jest pierwszy element listy LW, wierzchołek zostaje wyszukany w liście LZ i odległość z wierzchołka początkowego zostaje przypisana do element↩ _LZ->droga, a wskaźnik na wierzchołek poprzedni (czyli wierzchołek początkowy) zostaje przypisany do element↩ _LZ->poprzedni_wierzchołek. Element_LZ-> odwiedzone zostaje zmieniony na true. W następnej kolejności, wywołana zostaje dla tego wierzchołka rekurencyjnie funkcja dijkstra. Przed przypisaniem wartości do element_LZ- >droga zostaje sprawdzone, czy wartość już tam zapisana nie jest nieskończonością, bądź czy nie jest większa od nowej. Jeśli tak, wartość zostaje przypisana. Funkcja dijkstra jest wywoływana rekurencyjnie tylko dla wierzchołków, które nie zostały jeszcze odwiedzone, bądź których droga została zmieniona. Algorytm wykonuje się, aż najkrótsze ścieżki z danego wierzchołka początkowego do reszty wierzchołków zostaną odnalezione.

**Parametry**

| | |
|---|---|
| *tmp_wierzcholek* | wskaźnik na wierzchołek początkowy przekazany przez referencję |
| *numer* | numer wierzchołka początkowego |
| *glowa* | wskaźnik na węzeł początkowy listy zewnętrznej LZ |

#### 5.1.1.2 dodaj_elementLW()

```
element_LW* dodaj_elementLW (
            double wartosc,
            int wezel )
```

Funkcja tworząca nowy węzeł listy jednokierunkowej (lista wewnętrzna listy list). Wpisuje do struktury podane wartości i zwraca wskaźnik na nowo powstały węzeł LW.

**Parametry**

| | |
|---|---|
| *wartosc* | długość krawędzi do danego wierzchołka |
| *wezel* | numer wierzchołka |

**Zwraca**

wskaźnik na nowy węzeł listy.

#### 5.1.1.3 dodaj_elementLZ()

```
element_LZ* dodaj_elementLZ (
            int wartosc )
```

Funkcja tworząca pierwszy węzeł listy jednokierunkowej LZ (lista zewnętrzna listy list). Lista ta przechowuje numery wszystkich wierzchołków w grafie (bez powtórzeń), zmienną odwiedzone, informującą czy dany wierzchołek został już odwiedzony podczas wykonywania algorytmu (początkowo false), zmienną droga przechowującą najmniejszą wartość drogi do danego wierzchołka (początkowo ma wartość nieskończoność, czyli największą wartość typu int) oraz wskaźniki na następny i poprzedni węzeł listy, poprzedni wierzchołek przy wyszukiwaniu ścieżki oraz wskaźnik na pierwszy element listy wewnętrznej LW. Do wszystkich wskaźników zostaje przypisany nullptr.

**Parametry**

| | |
|---|---|
| *wartosc* | (wierzcholek grafu) wartość, która ma zostać przypisana do p->pierwszy_wierzcholek |

**Zwraca**

funkcja zwraca wskaźnik na nowo dodany węzeł listy.

#### 5.1.1.4 dodaj_polaczenie()

```
void dodaj_polaczenie (
            element_LZ *& LZ,
            double waga,
            int wierzcholek,
            bool skierowana )
```

Funkcja tworząca listę wewnętrzną dla danego elementu listy zewnętrznej. Przypisuje adres na pierwszy element nowej listy wewnętrznej do LZ->polaczenie, tworząc tym samym połączenie między tymi dwoma listami. Jeśli dla danego węzła listy zewnętrznej nie ma jeszcze listy wewnętrznej, wywołana zostaje funkcja dodaj_elementLW z parametrami waga i wierzchołek. Jeśli dla danego węzła istnieją już elementy listy, zostaje wywołana funkcja dodaj_wezelLW z parametrami el_LW, wierzcholek i waga.

**Parametry**

| | |
|---|---|
| *LZ* | wskaźnik na podany element listy zewnętrznej listy list (będącej reprezentacją grafu) przekazany przez referencję |
| *waga* | długość krawędzi z wierzchołka, którego numer przechowywany jest w LZ do podanego jako parametr wierzchołka |
| *wierzcholek* | numer wierzcholka, do którego prowadzi krawędź |
| *skierowana* | true jeśli krawędź między wierzchołkami jest skierowana (prowadzi tylko w jedną stronę), false jeśli jest nieskierowana (prowadzi w obie strony) |

#### 5.1.1.5 dodaj_wezel()

```
sciezka* dodaj_wezel (
            int wartosc,
            double waga )
```

Funkcja tworząca nowy węzeł listy jednokierunkowej (sciezka). Lista ta przechowuje drogę (kolejne wierzchołki na drodze oraz odległość między pierwszym a ostatnim wierzchołkiem) między danym wierzchołkiem początkowym, a kolejnymi wierzchołkami grafu.

**Parametry**

| | |
|---|---|
| *wartosc* | wartość, która ma zostać przypisana do p->wierzcholek |
| *waga* | wartość, która ma zostać przypisana p->waga |

**Zwraca**

wskaźnik na nowo dodany węzeł listy.

#### 5.1.1.6 dodaj_wezelLW()

```
void dodaj_wezelLW (
            element_LW *& LW,
            int wierzcholek,
            double wartosc )
```

Funkcja dodająca na koniec listy wewnętrznej nowy wezeł. Sprawdza, czy wskaźnik na element listy, podany jako parametr, wskazuje na ostatni element. Jeśli nie, to przesuwa wskaźnik na następny element listy, aż nie trafi na LW->nastepny==nullptr. Wtedy zostaje wywołana funkcja dodaj_elementLW z parametrami wartosc i wezel przekazanymi do funkcji.

**Parametry**

| | |
|---|---|
| *LW* | wskaźnik na węzeł listy wewnętrznej podany przez referencję |
| *wierzcholek* | numer wierzchołka |
| *wartosc* | długość krawędzi do danego wierzchołka. |

#### 5.1.1.7 dodaj_wezelLZ()

```
void dodaj_wezelLZ (
            element_LZ *& glowa,
            int wartosc )
```

Funkcja dodająca na koniec listy LZ nowy węzeł.

**Parametry**

| | |
|---|---|
| *glowa* | wskaźnik przekazany przez referancję na pierwszy element listy |
| *wartosc* | liczba typu int (wierzcholek grafu), która ma zostać przypisana do p->pierwszy_wierzcholek |

#### 5.1.1.8 dodaj_wierzcholek()

```
void dodaj_wierzcholek (
            sciezka *& glowa,
            int wartosc,
            double waga )
```

Funkcja dodająca na początek nowy węzeł listy jednokierunkowej (sciezka). Nowo powstały węzeł, staje się głową listy.

**Parametry**

| | |
|---|---|
| *glowa* | wskaźnik przekazany przez referancję na pierwszy element listy |
| *wartosc* | wartość typu int, która zostaje przekazana do funkcji dodaj_wezel |
| *waga* | wartość typu double, która zostaje przekazana do funkcji dodaj_wezel |

#### 5.1.1.9 przelaczniki()

```
bool przelaczniki (
            int argc,
            char * argv[] )
```

Funkcja do obsługi przełączników. Sprawdza liczbę argumentów (typu string) wprowadzonych z linii poleceń. Jeśli jest ich 7 to sprawdza, czy przełączniki są prawidłowe, czyli czy jest to -g -w i -o. Następnie sprawdzane jest rozszerzenie plików (wywołana zostaje funkcja sprawdz_pliki) oraz poprawność otwierania się plików (funkcja szukaj↩ _drogi, która jeśli warunki zostaną spełnione, wykona cały program i zwróci true). W przypadku niespełnienia warunków, funkcja zwraca false.

**Parametry**

| | |
|---|---|
| *argc* | liczba argumentów |
| *argv* | wskaźnik na tablicę łańcuchów znakowych |

**Zwraca**

true albo false, w zależności czy warunki zostały spełnione.

#### 5.1.1.10 sprawdz_pliki()

```
bool sprawdz_pliki (
            string plik1,
            string plik2,
            string plik3 )
```

Funkcja sprawdzająca czy pliki są tekstowe (czy mają rozszerzenie .txt).

**Parametry**

| | |
|---|---|
| *plik1* | nazwa typu string pierwszego pliku |
| *plik2* | nazwa typu string drugiego pliku |
| *plik3* | nazwa typu string trzeciego pliku |

**Zwraca**

jeśli wszystkie pliki mają rozszerzenie .txt, funkcja zwraca true, w przeciwnym wypadku zwraca false.

#### 5.1.1.11 szukaj_drogi()

```
bool szukaj_drogi (
            string wierzcholki,
            string graf,
            string wyniki )
```

Funkcja otwiera plik z grafem i wywołuje funkcję wczytaj_graf oraz otwiera plik z wierzchołkami początkowymi, odczytuje każdy wierzchołek i jeśli istnieje on w grafie to wywołuje funkcje dijkstra i wczytaj_sciezke. Na koniec zamyka pliki i wywoluje funkcje usun_graf. Jeśli pliki otwierają się nieprawidłowo, funkcja zwraca wartość false i zwalnia zaalokowaną pamięć.

**Parametry**

| | |
|---|---|
| *wierzcholki* | nazwa pliku z wierzchołkami |
| *graf* | nazwa pliku z grafem |
| *wyniki* | nazwa pliku na wyniki |

**Zwraca**

true jeśli wszystkie pliki otwarły się poprawnie

#### 5.1.1.12 usun_graf()

```
void usun_graf (
            element_LZ *& poczatek )
```

Funkcja usuwająca graf, przedstawiony za pomocą struktury listy listy. Dla każdego elementu listy zewnętrznej LZ, funkcja usuwa elementy połączonej z nią listy wewnętrznej LW (o ile istnieje), po czym po przypisaniu do zmiennej tymczasowej wskaźnika na następny węzeł LZ, zostaje usunięty dany węzeł LZ, tak długo aż lista zewnętrzna ma elementy.

**Parametry**

| | |
|---|---|
| *poczatek* | wskaźnik przekazany przez referancję na początek listy zewnętrznej. |

#### 5.1.1.13 usun_liste_wierzcholkow()

```
void usun_liste_wierzcholkow (
            sciezka *& glowa )
```

Funkcja usuwająca kolejno wszystkie elementy listy (ścieżki).

**Parametry**

| | |
|---|---|
| *glowa* | wskaźnik przekazany przez referencję na pierwszy element listy. |

#### 5.1.1.14 wczytaj_graf()

```
element_LZ* wczytaj_graf (
            ifstream & plik )
```

Funkcja wczytująca z podanego pliku graf do listy list (lista zewnętrzna (jej elementami jest struktura element_LZ), której każdy węzeł wskazuje na listę wewnętrzną (jej elementami jest struktura element_LW). Jeśli dane w pliku są nieprawidłowe, funkcja zwraca nullptr. Jeśli oczytany wierzchołek nie znajduje się jeszcze w grafie, wywołana zostaje dla niego funkcja dodaj_elementLZ, w przeciwnym przypadku, wierzchołek jest wczytywany do istniejącej listy LZ za pomocą funkcji dodaj_wezelLZ. W zależności od tego, czy krawędź jest skierowana, wierzchołki sąsiednie zostają wpisane do listy wewnętrznej LW. Połączenie między listą zewnętrzną a wewnętrzną jest dodawane za pomocą funkcji dodaj_polaczenie.

**Parametry**

| | |
|---|---|
| *plik* | referencja na plik z grafem |

**Zwraca**

wskaźnik na pierwszy element listy zewnętrznej LZ. Jeśli dane są nieprawidłowe, zwraca nullptr.

#### 5.1.1.15 wczytaj_sciezke()

```
void wczytaj_sciezke (
            int numer,
            element_LZ *& glowa,
            bool czy_wystepuje,
            ofstream & wyniki )
```

Funkcja wczytująca najpierw ścieżkę z wierzchołka początkowego do pierwszego wierzchołka grafu do listy ścieżka, a następnie wpisująca tą ścieżkę do pliku wyjściowego.

**Parametry**

| | |
|---|---|
| *numer* | numer wierzchołka początkowego |
| *glowa* | wskaźnik na węzeł początkowy listy zewnętrznej LZ |
| *czy_wystepuje* | zmienna typu bool informująca o tym czy wierzchołek występuje w grafie |
| *wyniki* | referencja na plik wyjściowy na wyniki. |

#### 5.1.1.16 wyczysc_dane()

```
void wyczysc_dane (
            element_LZ * p )
```

Funkcja czyści/resetuje dane na liście zewnętrznej po zastosowaniu algorytmu dijkstry.

**Parametry**

| | |
|---|---|
| *p* | wskaźnik na początek listy zewnętrznej LZ |

#### 5.1.1.17 znajdzLZ()

```
element_LZ* znajdzLZ (
            int wartosc,
            element_LZ * p )
```

Funkcja szukająca w LZ (liście zewnętrznej listy list będącej reprezentacją grafu) podanego wierzchołka.

**Parametry**

| | |
|---|---|
| *wartosc* | poszukiwany wierzcholek w liście |
| *p* | wskaźnik na pierwszy element listy |

**Zwraca**

wskaźnik na element listy, w której znajduje się dany wierzchołek. Jeśli w liście nie ma takiego wierzchołka, funkcja zwraca nullptr.

## 5.2 Dokumentacja pliku dijkstra.h

```
#include <iostream>
#include <fstream>
#include "struktury.h"
```

## Definicje

- #define DIJKSTRA_H

## Funkcje

- bool przelaczniki (int argc, char ∗argv[ ])
- bool sprawdz_pliki (string plik1, string plik2, string plik3)
- sciezka ∗ dodaj_wezel (int wartosc, double waga)
- void dodaj_wierzcholek (sciezka ∗&glowa, int wartosc, double waga)
- void usun_liste_wierzcholkow (sciezka ∗&glowa)
- element_LZ ∗ dodaj_elementLZ (int wartosc)
- void dodaj_wezelLZ (element_LZ ∗&glowa, int wartosc)
- element_LZ ∗ znajdzLZ (int wartosc, element_LZ ∗p)
- element_LW ∗ dodaj_elementLW (double wartosc, int wezel)
- void dodaj_wezelLW (element_LW ∗&LW, int wierzcholek, double wartosc)
- void dodaj_polaczenie (element_LZ ∗&LZ, double waga, int wierzcholek, bool skierowana)
- void usun_graf (element_LZ ∗&poczatek)
- element_LZ ∗ wczytaj_graf (ifstream &plik)
- void wyczysc_dane (element_LZ ∗p)
- void dijkstra (element_LZ ∗&tmp_wierzcholek, int numer, element_LZ ∗glowa)
- bool szukaj_drogi (string wierzcholki, string graf, string wyniki)
- void wczytaj_sciezke (int numer, element_LZ ∗&glowa, bool czy_wystepuje, ofstream &wyniki)

### 5.2.1 Dokumentacja definicji

#### 5.2.1.1 DIJKSTRA_H

```
#define DIJKSTRA_H
```

### 5.2.2 Dokumentacja funkcji

#### 5.2.2.1 dijkstra()

```
void dijkstra (
            element_LZ *& tmp_wierzcholek,
            int numer,
            element_LZ * glowa )
```

Funkcja wykonująca algorytm dijkstry, wyszukuje najkrótszą drogę z podanego wierzchołka do reszty wierzchołków. Zostaje wywołana dla danego wierzchołka początkowego, który zostaje wyszukany w liście LZ. Początkowo w liście LZ wartość drogi do wierzchołka początkowego jest równa zero, natomiast do reszty wierzchołków jest równa maksymalnej wartości zmiennej typu int ("nieskonczonosc"). Następnie sprawdzany jest pierwszy element listy LW, wierzchołek zostaje wyszukany w liście LZ i odległość z wierzchołka początkowego zostaje przypisana do element↩ _LZ->droga, a wskaźnik na wierzchołek poprzedni (czyli wierzchołek początkowy) zostaje przypisany do element↩ _LZ->poprzedni_wierzchołek. Element_LZ-> odwiedzone zostaje zmieniony na true. W następnej kolejności, wywołana zostaje dla tego wierzchołka rekurencyjnie funkcja dijkstra. Przed przypisaniem wartości do element_LZ- >droga zostaje sprawdzone, czy wartość już tam zapisana nie jest nieskończonością, bądź czy nie jest większa od nowej. Jeśli tak, wartość zostaje przypisana. Funkcja dijkstra jest wywoływana rekurencyjnie tylko dla wierzchołków, które nie zostały jeszcze odwiedzone, bądź których droga została zmieniona. Algorytm wykonuje się, aż najkrótsze ścieżki z danego wierzchołka początkowego do reszty wierzchołków zostaną odnalezione.

**Parametry**

| | |
|---|---|
| *tmp_wierzcholek* | wskaźnik na wierzchołek początkowy przekazany przez referencję |
| *numer* | numer wierzchołka początkowego |
| *glowa* | wskaźnik na węzeł początkowy listy zewnętrznej LZ |

#### 5.2.2.2 dodaj_elementLW()

```
element_LW* dodaj_elementLW (
            double wartosc,
            int wezel )
```

Funkcja tworząca nowy węzeł listy jednokierunkowej (lista wewnętrzna listy list). Wpisuje do struktury podane wartości i zwraca wskaźnik na nowo powstały węzeł LW.

**Parametry**

| | |
|---|---|
| *wartosc* | długość krawędzi do danego wierzchołka |
| *wezel* | numer wierzchołka |

**Zwraca**

wskaźnik na nowy węzeł listy.

#### 5.2.2.3 dodaj_elementLZ()

```
element_LZ* dodaj_elementLZ (
            int wartosc )
```

Funkcja tworząca pierwszy węzeł listy jednokierunkowej LZ (lista zewnętrzna listy list). Lista ta przechowuje numery wszystkich wierzchołków w grafie (bez powtórzeń), zmienną odwiedzone, informującą czy dany wierzchołek został już odwiedzony podczas wykonywania algorytmu (początkowo false), zmienną droga przechowującą najmniejszą wartość drogi do danego wierzchołka (początkowo ma wartość nieskończoność, czyli największą wartość typu int) oraz wskaźniki na następny i poprzedni węzeł listy, poprzedni wierzchołek przy wyszukiwaniu ścieżki oraz wskaźnik na pierwszy element listy wewnętrznej LW. Do wszystkich wskaźników zostaje przypisany nullptr.

**Parametry**

| | |
|---|---|
| *wartosc* | (wierzcholek grafu) wartość, która ma zostać przypisana do p->pierwszy_wierzcholek |

**Zwraca**

funkcja zwraca wskaźnik na nowo dodany węzeł listy.

#### 5.2.2.4 dodaj_polaczenie()

```
void dodaj_polaczenie (
            element_LZ *& LZ,
            double waga,
            int wierzcholek,
            bool skierowana )
```

Funkcja tworząca listę wewnętrzną dla danego elementu listy zewnętrznej. Przypisuje adres na pierwszy element nowej listy wewnętrznej do LZ->polaczenie, tworząc tym samym połączenie między tymi dwoma listami. Jeśli dla danego węzła listy zewnętrznej nie ma jeszcze listy wewnętrznej, wywołana zostaje funkcja dodaj_elementLW z parametrami waga i wierzchołek. Jeśli dla danego węzła istnieją już elementy listy, zostaje wywołana funkacja dodaj_wezelLW z parametrami el_LW, wierzcholek i waga.

**Parametry**

| | |
|---|---|
| *LZ* | wskaźnik na podany element listy zewnętrznej listy list (będącej reprezentacją grafu) przekazany przez referencję |
| *waga* | długość krawędzi z wierzchołka, którego numer przechowywany jest w LZ do podanego jako parametr wierzchołka |
| *wierzcholek* | numer wierzcholka, do którego prowadzi krawędź |
| *skierowana* | true jeśli krawędź między wierzchołkami jest skierowana (prowadzi tylko w jedną stronę), false jeśli jest nieskierowana (prowadzi w obie strony) |

#### 5.2.2.5 dodaj_wezel()

```
sciezka* dodaj_wezel (
            int wartosc,
            double waga )
```

Funkcja tworząca nowy węzeł listy jednokierunkowej (sciezka). Lista ta przechowuje drogę (kolejne wierzchołki na drodze oraz odległość między pierwszym a ostatnim wierzchołkiem) między danym wierzchołkiem początkowym, a kolejnymi wierzchołkami grafu.

**Parametry**

| | |
|---|---|
| *wartosc* | wartość, która ma zostać przypisana do p->wierzcholek |
| *waga* | wartość, która ma zostać przypisana p->waga |

**Zwraca**

wskaźnik na nowo dodany węzeł listy.

#### 5.2.2.6 dodaj_wezelLW()

```
void dodaj_wezelLW (
            element_LW *& LW,
```

```
            int wierzcholek,
            double wartosc )
```

Funkcja dodająca na koniec listy wewnętrznej nowy wezeł. Sprawdza, czy wskaźnik na element listy, podany jako parametr, wskazuje na ostatni element. Jeśli nie, to przesuwa wskaźnik na następny element listy, aż nie trafi na LW->nastepny==nullptr. Wtedy zostaje wywołana funkcja dodaj_elementLW z parametrami wartosc i wezel przekazanymi do funkcji.

**Parametry**

| | |
|---|---|
| *LW* | wskaźnik na węzeł listy wewnętrznej podany przez referencję |
| *wierzcholek* | numer wierzchołka |
| *wartosc* | długość krawędzi do danego wierzchołka. |

#### 5.2.2.7 dodaj_wezelLZ()

```
void dodaj_wezelLZ (
            element_LZ *& glowa,
            int wartosc )
```

Funkcja dodająca na koniec listy LZ nowy węzeł.

**Parametry**

| | |
|---|---|
| *glowa* | wskaźnik przekazany przez referancję na pierwszy element listy |
| *wartosc* | liczba typu int (wierzcholek grafu), która ma zostać przypisana do p->pierwszy_wierzcholek |

#### 5.2.2.8 dodaj_wierzcholek()

```
void dodaj_wierzcholek (
            sciezka *& glowa,
            int wartosc,
            double waga )
```

Funkcja dodająca na początek nowy węzeł listy jednokierunkowej (sciezka). Nowo powstały węzeł, staje się głową listy.

**Parametry**

| | |
|---|---|
| *glowa* | wskaźnik przekazany przez referancję na pierwszy element listy |
| *wartosc* | wartość typu int, która zostaje przekazana do funkcji dodaj_wezel |
| *waga* | wartość typu double, która zostaje przekazana do funkcji dodaj_wezel |

#### 5.2.2.9 przelaczniki()

```
bool przelaczniki (
            int argc,
            char * argv[] )
```

Funkcja do obsługi przełączników. Sprawdza liczbę argumentów (typu string) wprowadzonych z linii poleceń. Jeśli jest ich 7 to sprawdza, czy przełączniki są prawidłowe, czyli czy jest to -g -w i -o. Następnie sprawdzane jest rozszerzenie plików (wywołana zostaje funkcja sprawdz_pliki) oraz poprawność otwierania się plików (funkcja szukaj↩ _drogi, która jeśli warunki zostaną spełnione, wykona cały program i zwróci true). W przypadku niespełnienia warunków, funkcja zwraca false.

**Parametry**

| | |
|---|---|
| *argc* | liczba argumentów |
| *argv* | wskaźnik na tablicę łańcuchów znakowych |

**Zwraca**

true albo false, w zależności czy warunki zostały spełnione.

#### 5.2.2.10 sprawdz_pliki()

```
bool sprawdz_pliki (
            string plik1,
            string plik2,
            string plik3 )
```

Funkcja sprawdzająca czy pliki są tekstowe (czy mają rozszerzenie .txt).

**Parametry**

| | |
|---|---|
| *plik1* | nazwa typu string pierwszego pliku |
| *plik2* | nazwa typu string drugiego pliku |
| *plik3* | nazwa typu string trzeciego pliku |

**Zwraca**

jeśli wszystkie pliki mają rozszerzenie .txt, funkcja zwraca true, w przeciwnym wypadku zwraca false.

#### 5.2.2.11 szukaj_drogi()

```
bool szukaj_drogi (
            string wierzcholki,
```

```
            string graf,
            string wyniki )
```

Funkcja otwiera plik z grafem i wywołuje funkcję wczytaj_graf oraz otwiera plik z wierzchołkami początkowymi, odczytuje każdy wierzchołek i jeśli istnieje on w grafie to wywołuje funkcje dijkstra i wczytaj_sciezke. Na koniec zamyka pliki i wywoluje funkcje usun_graf. Jeśli pliki otwierają się nieprawidłowo, funkcja zwraca wartość false i zwalnia zaalokowaną pamięć.

**Parametry**

| | |
|---|---|
| *wierzcholki* | nazwa pliku z wierzchołkami |
| *graf* | nazwa pliku z grafem |
| *wyniki* | nazwa pliku na wyniki |

**Zwraca**

true jeśli wszystkie pliki otwarły się poprawnie

#### 5.2.2.12 usun_graf()

```
void usun_graf (
            element_LZ *& poczatek )
```

Funkcja usuwająca graf, przedstawiony za pomocą struktury listy listy. Dla każdego elementu listy zewnętrznej LZ, funkcja usuwa elementy połączonej z nią listy wewnętrznej LW (o ile istnieje), po czym po przypisaniu do zmiennej tymczasowej wskaźnika na następny węzeł LZ, zostaje usunięty dany węzeł LZ, tak długo aż lista zewnętrzna ma elementy.

**Parametry**

| | |
|---|---|
| *poczatek* | wskaźnik przekazany przez referancję na początek listy zewnętrznej. |

#### 5.2.2.13 usun_liste_wierzcholkow()

```
void usun_liste_wierzcholkow (
            sciezka *& glowa )
```

Funkcja usuwająca kolejno wszystkie elementy listy (ścieżki).

**Parametry**

| | |
|---|---|
| *glowa* | wskaźnik przekazany przez referencję na pierwszy element listy. |

#### 5.2.2.14 wczytaj_graf()

```
element_LZ* wczytaj_graf (
            ifstream & plik )
```

Funkcja wczytująca z podanego pliku graf do listy list (lista zewnętrzna (jej elementami jest struktura element_LZ), której każdy węzeł wskazuje na listę wewnętrzną (jej elementami jest struktura element_LW). Jeśli dane w pliku są nieprawidłowe, funkcja zwraca nullptr. Jeśli oczytany wierzchołek nie znajduje się jeszcze w grafie, wywołana zostaje dla niego funkcja dodaj_elementLZ, w przeciwnym przypadku, wierzchołek jest wczytywany do istniejącej listy LZ za pomocą funkcji dodaj_wezelLZ. W zależności od tego, czy krawędź jest skierowana, wierzchołki sąsiednie zostają wpisane do listy wewnętrznej LW. Połączenie między listą zewnętrzną a wewnętrzną jest dodawane za pomocą funkcji dodaj_polaczenie.

**Parametry**

| | |
|---|---|
| *plik* | referencja na plik z grafem |

**Zwraca**

wskaźnik na pierwszy element listy zewnętrznej LZ. Jeśli dane są nieprawidłowe, zwraca nullptr.

#### 5.2.2.15 wczytaj_sciezke()

```
void wczytaj_sciezke (
            int numer,
            element_LZ *& glowa,
            bool czy_wystepuje,
            ofstream & wyniki )
```

Funkcja wczytująca najpierw ścieżkę z wierzchołka początkowego do pierwszego wierzchołka grafu do listy ścieżka, a następnie wpisująca tą ścieżkę do pliku wyjściowego.

**Parametry**

| | |
|---|---|
| *numer* | numer wierzchołka początkowego |
| *glowa* | wskaźnik na węzeł początkowy listy zewnętrznej LZ |
| *czy_wystepuje* | zmienna typu bool informująca o tym czy wierzchołek występuje w grafie |
| *wyniki* | referencja na plik wyjściowy na wyniki. |

#### 5.2.2.16 wyczysc_dane()

```
void wyczysc_dane (
            element_LZ * p )
```

Funkcja czyści/resetuje dane na liście zewnętrznej po zastosowaniu algorytmu dijkstry.

**Parametry**

| | |
|---|---|
| *p* | wskaźnik na początek listy zewnętrznej LZ |

#### 5.2.2.17 znajdzLZ()

```
element_LZ* znajdzLZ (
            int wartosc,
            element_LZ * p )
```

Funkcja szukająca w LZ (liście zewnętrznej listy list będącej reprezentacją grafu) podanego wierzchołka.

**Parametry**

| | |
|---|---|
| *wartosc* | poszukiwany wierzcholek w liście |
| *p* | wskaźnik na pierwszy element listy |

**Zwraca**

wskaźnik na element listy, w której znajduje się dany wierzchołek. Jeśli w liście nie ma takiego wierzchołka, funkcja zwraca nullptr.

## 5.3 Dokumentacja pliku main.cpp

```
#include "dijkstra.h"
#include "struktury.h"
```

### Funkcje

- int main (int argc, char ∗argv[ ])

### 5.3.1 Dokumentacja funkcji

#### 5.3.1.1 main()

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

## 5.4 Dokumentacja pliku struktury.h

### Komponenty

- struct sciezka
- struct element_LW
- struct element_LZ

### Definicje

- #define STRUKTURY_H

### 5.4.1 Dokumentacja definicji

#### 5.4.1.1 STRUKTURY_H

```
#define STRUKTURY_H
```

# Indeks