Politechnika Śląska Wydział Informatyki, Elektroniki i Informatyki

# Podstawy Programowania Komputerów

# Listonosz

| | |
|---|---|
| autor | Mateusz Górecki |
| prowadzący | dr inż. Paweł Foszner |
| rok akademicki | 2020/2021 |
| kierunek | informatyka |
| rodzaj studiów | SSI |
| semestr | 1 |
| termin laboratorium | poniedziałek, 10.30 – 12.45,<br>wtorek 12.00-14.15 |
| sekcja | 22 |
| termin oddania sprawozdania | 2020-11-03 |

# 1 Treść zadania

W swojej pracy, listonosz wyrusza z poczty, dostarcza przesyłki adresatom, by na końcu powrócić na pocztę. Listonosz musi przejść każdą ulicę w swoim rejonie co najmniej raz. Zadanie polega na wyznaczeniu najkrótszej jego drogi. Program uruchamiany jest z linii poleceń z użyciem następujących przełączników:

-i plik wejściowy (z układem ulic)
-o plik wyjściowy (do którego przekazana zostanie wyznaczona trasa)
-p punkt startowy (numer skrzyżowania)

# 2 Analiza zadania

Zagadnienie przedstawia tzw. “problem chińskiego listonosza” - polegającego na znalezieniu ścieżki zamkniętej (wracającej do wierzchołka początkowego), zawierającej każdą ulicę (krawędź grafu) co najmniej raz i mającej minimalną długość.

## 2.1 Struktury danych

W programie wykorzystano struktury biblioteki standardowej STL takie jak: vector, map oraz stack. Vector(wektor) zastosowano do przechowywania danych o grafie, ponieważ umożliwia on stały czas dostępu do każdego elementu. Map wykorzystano m. in. jako strukturę przechowywującą informacje o stopniu poszczególnych wierzchołków grafu. Stack(stos) został natomiast użyty w algorytmie sprawdzania spójności grafu.

## 2.2 Algorytmy

Do rozwiązana głównego zagadnienia programu wykorzystuje algorytm **Fleury’ego** odnajdującego cykl/drogę Eulera w grafie eulerowskim, jeśli podany graf nie jest grafem eulerowskim dopisujące dodatkowe nadmiarowe połączenia (z wykorzystaniem podejścia kombinatorycznego) o najmniejszym koszcie (wykorzystując algorytm Dijkstry).

# 3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików: wejściowego, wyjściowego oraz punkt startowy po odpowiednich przełącznikach: (-i dla pliku wejściowego, -o dla pliku wyjściowego oraz –p dla punktu startowego), np.

**listonosz -i przyklad.txt -o wyniki.txt -p 1**

**listonosz -o wyniki.txt -i przyklad.txt -p 2**

Pliki wejściowe/wyjściowe są plikami tekstowymi. Kolejność użycia przełączników jest jak wyżej w przykładzie pokazano dowolna. Przy uruchomieniu programu bez żadnych przełączników:

**listonosz**

wyświetlona zostanie krótka instrukcja, natomiast uruchomienie programu z nieprawidłowymi przełącznikami/parametrami spowoduje dodatkowo wyświetlenie odpowiedniego komunikatu, np.

**listonosz -i przyklad.txt -p**

**Nie odpowiednia ilosc argumentow (3/6).**

**Instrukcja :**

**Program uruchamiany jest z linii polecen z wykorzystaniem nastepujacych przelacznikow:**

**'-i' plik wejsciowy**

**'-o' plik wyjsciowy**

**'-p' punkt startowy(numer skrzyzowania)**

W przypadku wystąpienia błędu z plikiem wejściowym zostanie wyświetlony odpowiedni komunikat:

**Blad pliku wejsciowego, sprawdz czy istnieje i jest zgodny z zaleceniem :**

**Linia pliku: <nr skrzyzowania 1> <nr skrzyzowania 2> <Dlugosc ulicy> <Nazwa ulicy>**

Wyświetlane są także odpowiednie komunikaty dla niepoprawnych danych np..

1. Dla punktu startowego nie występującego w grafie:

   **Podany punkt startowy nie nalezy do grafu(siatka skrzyzowania).**

2. Dla grafu niespójnego:

   **Podany graf(siatka skrzyżowania) jest niespójny.**

# 4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. W programie rozdzielono interfejs (komunikację z użytkownikiem) od logiki aplikacji (wyznaczanie drogi dla listonosza – droga Eulera).

## 4.1 Ogólna struktura programu

W funkcji głównej wywoływana jest funkcja **Pobranie_Argumentow**, która przypisuje podane parametry do odpowiednich zmiennych, sprawdzając jednocześnie ich poprawność, w razie wystąpienia błędu wyświetla odpowiedni komunikat i przerywa działanie programu. Następnie wywoływana jest funkcja **Pobranie_Danych**. Funkcja ta jest odpowiedzialna za pobranie danych z pliku wejściowego do odpowiedniej zmiennej (**vector<krawendz> - opisana w załączniku**) i zamkniecie pliku (przy wystąpieniu jakiegokolwiek błędu zwraca wartość **false**, w razie ich braku zwraca **true**). Kolejną wywoływaną funkcją jest funkcja **Stopnie** zwracająca zmienną typu **map<int,int>** przechowywującą stopnie poszczególnych wierzchołków wczytanego grafu.

Następnymi wywoływanymi funkcjami są funkcje **Czy_Nalezy_Do_Grafu** i **Czy_spojny**, które kolejno sprawdzają czy podany przez użytkownika punkt startowy należy do grafu oraz czy podany przez niego graf jest spójny, obie te funkcje zwracają wartości **true** dla poprawnego/pozytywnego wykonania oraz **false** dla błędnego/negatywnego (co skutkuje wyświetleniem odpowiedniego komunikatu oraz zakończeniem programu).

Kolejną wywoływaną funkcją jest funkcja **Sprawdz_Euler** sprawdzająca czy można w podanym grafie wyznaczyć drogę/cykl Eulera. Funkcja ta zwraca **true** jeśli jest to możliwe lub **false** jeśli wymagane jest dodanie nadmiarowych/dodatkowych krawędzi do grafu.

Jeśli funkcja **Sprawdz_Euler** zwróci **true** wywoływana jest funkcja **Droga_Eulera** generująca przy pomocy algorytmu **Fleury'ego** drogę Eulera. Jeśli jednak funkcja **Sprawdz_Euler** zwróci **false** to najpierw wywoływana jest funkcja **Naprawa** odpowiedzialna za dopisanie do grafu dodatkowych nadmiarowych krawędzi (w najbardziej optymalny sposób - wykorzystuje ona m. in. algorytm Dijkstry) i dopiero po jej wykonaniu jest wywoływana funkcja **Droga_Eulera**.

Ostatnią wywoływaną funkcją jest funkcja **Przekaz_Wynik** zapisująca wynik do pliku wynikowego (w razie wystąpienia błędu wyświetla odpowiedni komunikat).

## 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.

Pliki wejściowe niepoprawne (zawierające błędne/podane w błędnej kolejności dane, niezgodne ze specyfikacją) powodują wyświetlenie odpowiedniego komunikatu.

Program został przetestowany przy użyciu wielu różnie skomplikowanych danych wejściowych (np. wymagających generowania większej ilości dodatkowych połączeń w celu wyznaczenia drogi Eulera). Wraz ze wzrostem liczby nieparzystych skrzyżowań (wierzchołków grafu) ilość sprawdzanych kombinacji (dla 8 jest ich 105, dla 10 jest ich 945, a dla 12 już 10395) co powoduje, że program dla mniej sprzyjających przypadków wykonuję się coraz dłużej.

# 6 Wnioski

Program Listonosz był jak na pierwszy mój projekt, programem dość skomplikowanym wymagającym zaimplementowania trzech algorytmów (**Dijkstry**, **Fleury'ego** oraz **algorytmu sprawdzania spójności grafu**) oraz obsługi dodawania nadmiarowych połączeń i sprawdzania wszystkich możliwych kombinacji (podejściem **kombinatorycznym**, której jak wyżej w testach opisałem może przyczynić się do spowolnienia działania programu) w poszukiwaniu tej najoptymalniejszej (tu wykorzystując wyżej wspomniana **Dijkstrę**).

Łatwiejszą częścią projektu okazało się implementowanie pobierania argumentów oraz praca z plikami (wejściowymi/wynikowymi) oraz tworzenie dokumentacji (podczas której miałem okazję lepiej zapoznać się z możliwościami m. in. **Doxygena**).

Uważam, że cała praca na tym projektem jak i na laboratoriach przyczyniła się znaczniej poprawy moich umiejętności w programowaniu w c++, co mam nadziej, że pomoże mi przy pracy z przyszłymi projektami.

# 7 Literatura

1. Algorytm_Fleury'ego
2. Algorytm_Dijkstry
3. eduinf.waw.pl - opisanie/lista kroków
4. www.cplusplus.com/
5. Omówienie "problemu chińskiego listonosza"

# Listonosz

1.0

Wygenerowano przez Doxygen 1.8.20

# Rozdział 1

# Indeks klas

## 1.1 Lista klas

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

# Rozdział 2

# Indeks plików

## 2.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

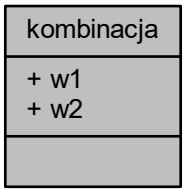
# Rozdział 3

# Dokumentacja klas

## 3.1 Dokumentacja struktury kombinacja

`#include <struktury.h>`

Diagram współpracy dla kombinacja:




### Atrybuty publiczne

- int w1
- int w2

### 3.1.1 Opis szczegółowy

Struktura opisująca kombinacje (para zmiennych typu int).

### 3.1.2 Dokumentacja atrybutów składowych

#### 3.1.2.1 w1

```
int kombinacja::w1
```

pierwsza liczba w kombinacji (calkowita)

#### 3.1.2.2 w2

```
int kombinacja::w2
```

druga liczba w kombinacji (calkowita)

Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

## 3.2 Dokumentacja struktury krawendz

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

Diagram współpracy dla krawendz:


### Atrybuty publiczne

- int w1
- int w2
- double dl
- string Nazwa = ""
- bool usunienta = false

### 3.2.1 Opis szczegółowy

Struktura opisująca krawędz grafu (pojedyńczą ulicę).

### 3.2.2 Dokumentacja atrybutów składowych

#### 3.2.2.1 dl

```
double krawendz::dl
```

długość krawędzi(ulicy)

#### 3.2.2.2 Nazwa

```
string krawendz::Nazwa = ""
```

nazwa krawędzi(ulicy)

#### 3.2.2.3 usunienta

```
bool krawendz::usunienta = false
```

parametr bool wskazujacy czy krawędź została już wykorzystana

#### 3.2.2.4 w1

```
int krawendz::w1
```

pierwszy wieszchołek

#### 3.2.2.5 w2

```
int krawendz::w2
```

drugi wieszchołek

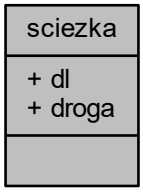
Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

## 3.3 Dokumentacja struktury sciezka

`#include <struktury.h>`

Diagram współpracy dla sciezka:




### Atrybuty publiczne

- double dl
- vector< int > droga

### 3.3.1 Opis szczegółowy

Struktura opisująca ścieżkę/drogę w grafie.

### 3.3.2 Dokumentacja atrybutów składowych

#### 3.3.2.1 dl

`double sciezka::dl`

długość całej ścieżki

#### 3.3.2.2 droga

`vector<int> sciezka::droga`

vektor zawierający kolejne wierzchołki ścieżki

Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

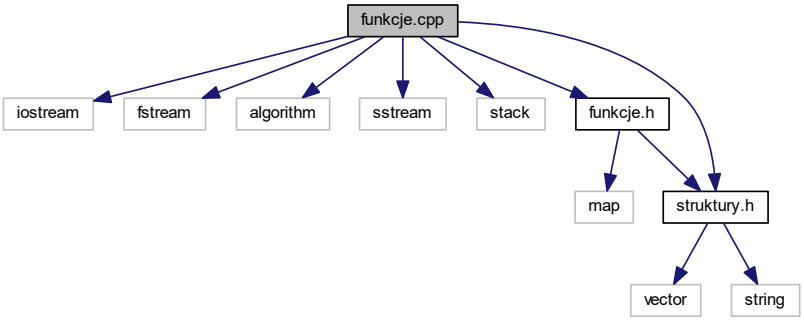
# Rozdział 4

# Dokumentacja plików

## 4.1 Dokumentacja pliku funkcje.cpp

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <sstream>
#include <stack>
#include "funkcje.h"
#include "struktury.h"
```

Wykres zależności załączania dla funkcje.cpp:




### Funkcje

- bool Pobranie_Argumentow (int &ile, char **arg, string &strIN, string &strOUT, int &poczatek)
- vector< int > Sasiad (int &u, vector< krawendz > &g)
- bool Pobranie_Danych (string &str, vector< krawendz > &Tgraf)
- map< int, int > Stopnie (vector< krawendz > &vec)
- bool Sprawdz_Euler (map< int, int > &m)
- bool Czy_spojny (vector< krawendz > &g, map< int, int > &m)
- vector< int > Droga_Eulera (vector< krawendz > g, map< int, int > m, int &p)
- double Dlugosc (vector< krawendz > &g, int &u, int &w)

- sciezka Dijkstra (vector< krawendz > &g, map< int, int > &m, int &v, int &k)
- vector< vector< kombinacja > > Generowanie_Kombinacji (int &k, vector< int > &vec)
- void Dopisz (vector< krawendz > &g, map< int, int > &m, vector< kombinacja > &rozw)
- void Naprawa (vector< krawendz > &g, map< int, int > &m)
- string Nazwa_ulicy (vector< krawendz > &g, int &x, int &y)
- bool Czy_Nalezy_Do_Grafu (map< int, int > &m, int &v)
- void Przekaz_Wynik (vector< krawendz > &g, string &strOUT, vector< int > &DE)

### 4.1.1 Dokumentacja funkcji

#### 4.1.1.1 Czy_Nalezy_Do_Grafu()

```
bool Czy_Nalezy_Do_Grafu (
            map< int, int > & m,
            int & v )
```

Funkcja sprawdza czy wierzchołek należy do grafu

**Parametry**

| | |
|---|---|
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |
| *v* | sprawdzany wierzchołek |

**Zwraca**

Jeśli sprawdzany wierzchołek należy do grafu funkcja zwraca wartość true, w przeciwnym razie zwraca wartość false.

#### 4.1.1.2 Czy_spojny()

```
bool Czy_spojny (
            vector< krawendz > & g,
            map< int, int > & m )
```

Funkcja sprawdza spójność grafu.

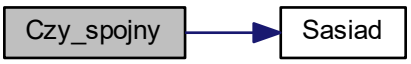
**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |

**Zwraca**

Jeśli graf jest spójny to zwraca wartość true, jeśli nie to zwraca wartość false.

Oto graf wywołań dla tej funkcji:




#### 4.1.1.3 Dijkstra()

```
sciezka Dijkstra (
            vector< krawendz > & g,
            map< int, int > & m,
            int & v,
            int & k )
```

Funkcja wykorzystując algorytm Dijkstry do wyznaczenia najkrótrzego połączenia dwóch wierzchołków grafu.

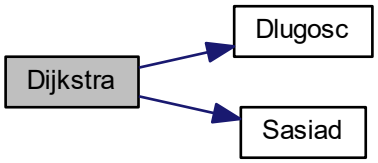
**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |
| *v* | początkowy wierzchołek |
| *k* | końcowy wierzchołek |

**Zwraca**

Zwraca zmienną typu 'sciezka' przechowywującą długość najkrótrzego połączenia oraz wektro kolejnych wierzchołków tego połączenia.

Oto graf wywołań dla tej funkcji:



#### 4.1.1.4 Dlugosc()

```
double Dlugosc (
            vector< krawendz > & g,
            int & u,
            int & w )
```

Funkcja zwracająca długość krawędzi łączącej podane wierzchołki w podanym grafie.

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *w* | wierzchołek grafu |
| *u* | wierzchołek grafu |

**Zwraca**

Zwraca długość szukanej krawędzi o ile istnieje w przeciwnym razie zwróci 0;

#### 4.1.1.5 Dopisz()

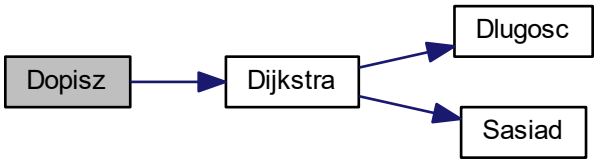
```
void Dopisz (
            vector< krawendz > & g,
            map< int, int > & m,
            vector< kombinacja > & rozw )
```

Funkcja dodaje/dopisuje do grafu podane połączenia(w postaci odpowiednich krawędzi).

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |
| *rozw* | wektor kombinacji wierzchołków dla których chcemy dopisac połączenia |

Oto graf wywołań dla tej funkcji:




#### 4.1.1.6 Droga_Eulera()

```
vector<int> Droga_Eulera (
            vector< krawendz > g,
            map< int, int > m,
            int & p )
```

Funkcja wykorzystując algorytm Fleury'ego wyznacza drogę/obwód Eulera.

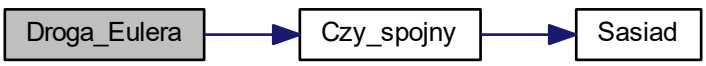
**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |
| *p* | zmienna przechowująca punkt początkowy |

**Zwraca**

Zwraca drogę w postaci wektora kolejnych pokonywanych wierzchołków.

Oto graf wywołań dla tej funkcji:



#### 4.1.1.7 Generowanie_Kombinacji()

```
vector<vector<kombinacja> > Generowanie_Kombinacji (
            int & k,
            vector< int > & vec )
```

Funkcja generuje możliwe kombinacje zbioru(kolejnych liczb całkowitych) o podanej wielkości w pary.

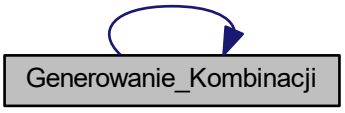
**Parametry**

| | |
|---|---|
| *k* | wielkość podanego zbioru |
| *vec* | podany zbiór |

**Zwraca**

Zwraca wektor zawierający kolejne kombinacje przedstawione w postaci wektorów kombinacji(par).

Oto graf wywołań dla tej funkcji:




#### 4.1.1.8 Naprawa()

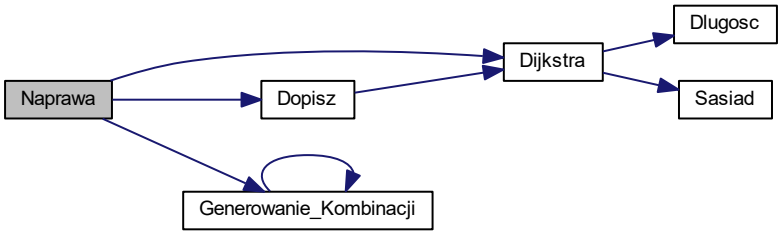
```
void Naprawa (
            vector< krawendz > & g,
            map< int, int > & m )
```

Funkcja odpowiednio zmienia graf(dopisuje nadmiarowe krawędzie), by móc wyznaczyć w nim drogę/obwód Eulera

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |

Oto graf wywołań dla tej funkcji:




#### 4.1.1.9 Nazwa_ulicy()

```
string Nazwa_ulicy (
            vector< krawendz > & g,
            int & x,
            int & y )
```

Funkcja zwracająca nazwe krawędzi łączącej podane wierzchołki w podanym grafie.

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *x* | wierzchołek grafu |
| *y* | wierzchołek grafu |

**Zwraca**

Zwraca nazwe szukanej krawędzi o ile istnieje w przeciwnym razie zwróci ("");

#### 4.1.1.10 Pobranie_Argumentow()

```
bool Pobranie_Argumentow (
            int & ile,
            char ** arg,
            string & strIN,
            string & strOUT,
            int & poczatek )
```

Funkcja przekazująca parametry funkcji main do odpowiednich zmiennych.

**Parametry**

| | |
|---|---|
| *ile* | liczba parametrów przekazanych funkcji main |
| *arg* | tablica napisów reprezentujących parametry funkcji main |
| *strIN* | zmienna do której zostanie przekazany parametr po fladze '-i' (ścieżka do pliku wejściowego) |
| *strOUT* | zmienna do której zostanie przekazany parametr po fladze '-o' (ścieżka do pliku wyjściowego) |
| *poczatek* | zmienna do której zostanie przekazany parametr po fladze '-p' (punkt startowy) |

**Zwraca**

Jeśli operacja przebiegnie poprawnie zwraca wartość true, w razie wystąpienia jakiegokolwiek błędu wyświetla odpowiedni komunikat i zwraca wartość false.

#### 4.1.1.11 Pobranie_Danych()

```
bool Pobranie_Danych (
            string & str,
            vector< krawendz > & Tgraf )
```

Funkcja pobiera dane z pliku wejściowego i przekazuje je do odpowiedniej zmiennej.

**Parametry**

| | |
|---|---|
| *str* | ścierzka do pliku wejściowego |
| *Tgraf* | zmienna, do którego przekazane zostaną dane z plik wejściowego |

**Zwraca**

Jeśli operacja przebiegnie poprawnie zwraca wartość true, w razie wystąpienia błędu zwraca wartość false.

#### 4.1.1.12 Przekaz_Wynik()

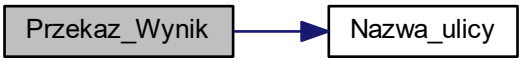
```
void Przekaz_Wynik (
            vector< krawendz > & g,
            string & strOUT,
            vector< int > & DE )
```

Funkcja przekazuje rozwiązanie do pliku wynikowego

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *strOUT* | ścieżka do pliku wynikowego |
| *DE* | zmienna przechowująca w wektorze drogę/obwód Eulera w postaci kolejnych wierzchołków grafu |

Oto graf wywołań dla tej funkcji:




#### 4.1.1.13 Sasiad()

```
vector<int> Sasiad (
            int & u,
            vector< krawendz > & g )
```

Funkcja zwraca numery sąsiadujących wierzchołków.

**Parametry**

| | |
|---|---|
| *u* | numer wierzchołka, którego sąsiadów szukamy |
| *g* | graf w któreym szukamy sąsiadów wierzchołka u |

**Zwraca**

Zwraca wektor przechowujący numery sąsiadujących wierzchołków.

#### 4.1.1.14 Sprawdz_Euler()

```
bool Sprawdz_Euler (
            map< int, int > & m )
```

Funkcja sprawdza czy dla danego grafu można wyznaczyc drogę Eulera.

**Parametry**

| | |
|---|---|
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |

**Zwraca**

Jeśli jest możliwe wyznaczenie drogi Eulera to funkcja zwraca true, jeśli nie ma takiej możliwości zwraca wartość false.

#### 4.1.1.15 Stopnie()

```
map<int, int> Stopnie (
            vector< krawendz > & vec )
```

Funkcja wyznacza stopień każdego wierzchołka grafu.

**Parametry**

| | |
|---|---|
| *vec* | zmienna przechowywująca graf |

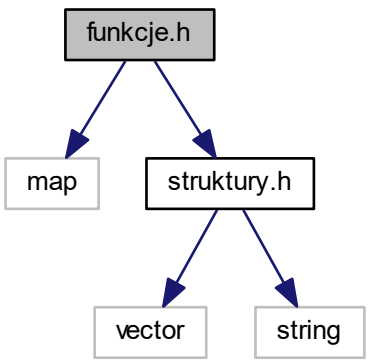
**Zwraca**

Zwraca zmienną typu 'map' przechwywującą dla każdego wierzchołka grafu jego stopień.

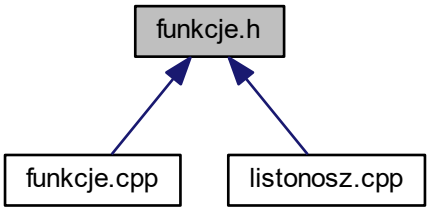
## 4.2 Dokumentacja pliku funkcje.h

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

Wykres zależności załączania dla funkcje.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:




## Funkcje

- bool Pobranie_Argumentow (int &ile, char **arg, string &strIN, string &strOUT, int &poczatek)
- vector< int > Sasiad (int &u, vector< krawendz > &g)
- bool Pobranie_Danych (string &str, vector< krawendz > &Tgraf)
- map< int, int > Stopnie (vector< krawendz > &vec)
- bool Sprawdz_Euler (map< int, int > &m)
- bool Czy_spojny (vector< krawendz > &g, map< int, int > &m)
- vector< int > Droga_Eulera (vector< krawendz > g, map< int, int > m, int &p)
- double Dlugosc (vector< krawendz > &g, int &u, int &w)
- sciezka Dijkstra (vector< krawendz > &g, map< int, int > &m, int &v, int &k)
- vector< vector< kombinacja > > Generowanie_Kombinacji (int &k, vector< int > &vec)
- void Dopisz (vector< krawendz > &g, map< int, int > &m, vector< kombinacja > &rozw)
- void Naprawa (vector< krawendz > &g, map< int, int > &m)
- string Nazwa_ulicy (vector< krawendz > &g, int &x, int &y)
- bool Czy_Nalezy_Do_Grafu (map< int, int > &m, int &v)
- void Przekaz_Wynik (vector< krawendz > &g, string &strOUT, vector< int > &DE)

### 4.2.1 Dokumentacja funkcji

#### 4.2.1.1 Czy_Nalezy_Do_Grafu()

```
bool Czy_Nalezy_Do_Grafu (
            map< int, int > & m,
            int & v )
```

Funkcja sprawdza czy wierzchołek należy do grafu

**Parametry**

| | |
|---|---|
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |
| *v* | sprawdzany wierzchołek |

**Zwraca**

Jeśli sprawdzany wierzchołek należy do grafu funkcja zwraca wartość true, w przeciwnym razie zwraca wartość false.

#### 4.2.1.2 Czy_spojny()

```
bool Czy_spojny (
            vector< krawendz > & g,
            map< int, int > & m )
```

Funkcja sprawdza spójność grafu.

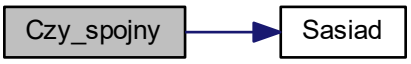
**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |

**Zwraca**

Jeśli graf jest spójny to zwraca wartość true, jeśli nie to zwraca wartość false.

Oto graf wywołań dla tej funkcji:




#### 4.2.1.3 Dijkstra()

```
sciezka Dijkstra (
            vector< krawendz > & g,
            map< int, int > & m,
            int & v,
            int & k )
```

Funkcja wykorzystując algorytm Dijkstry do wyznaczenia najkrótrzego połączenia dwóch wierzchołków grafu.

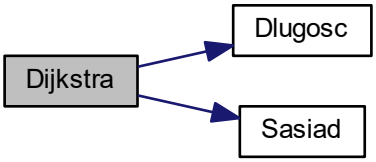
**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |
| *v* | początkowy wierzchołek |
| *k* | końcowy wierzchołek |

**Zwraca**

Zwraca zmienną typu 'sciezka' przechowywującą długość najkrótrzego połączenia oraz wektro kolejnych wierzchołków tego połączenia.

Oto graf wywołań dla tej funkcji:




#### 4.2.1.4 Dlugosc()

```
double Dlugosc (
            vector< krawendz > & g,
            int & u,
            int & w )
```

Funkcja zwracająca długość krawędzi łączącej podane wierzchołki w podanym grafie.

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *w* | wierzchołek grafu |
| *u* | wierzchołek grafu |

**Zwraca**

Zwraca długość szukanej krawędzi o ile istnieje w przeciwnym razie zwróci 0;

#### 4.2.1.5 Dopisz()

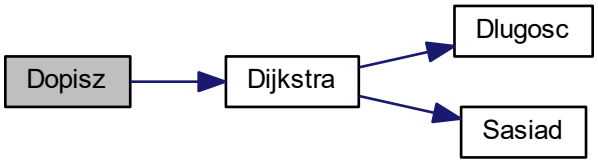
```
void Dopisz (
            vector< krawendz > & g,
            map< int, int > & m,
            vector< kombinacja > & rozw )
```

Funkcja dodaje/dopisuje do grafu podane połączenia(w postaci odpowiednich krawędzi).

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |
| *rozw* | wektor kombinacji wierzchołków dla których chcemy dopisac połączenia |

Oto graf wywołań dla tej funkcji:




#### 4.2.1.6 Droga_Eulera()

```
vector<int> Droga_Eulera (
            vector< krawendz > g,
            map< int, int > m,
            int & p )
```

Funkcja wykorzystując algorytm Fleury'ego wyznacza drogę/obwód Eulera.

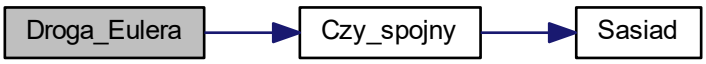
**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |
| *p* | zmienna przechowująca punkt początkowy |

**Zwraca**

Zwraca drogę w postaci wektora kolejnych pokonywanych wierzchołków.

Oto graf wywołań dla tej funkcji:




#### 4.2.1.7 Generowanie_Kombinacji()

```
vector<vector<kombinacja> > Generowanie_Kombinacji (
            int & k,
            vector< int > & vec )
```

Funkcja generuje możliwe kombinacje zbioru(kolejnych liczb całkowitych) o podanej wielkości w pary.

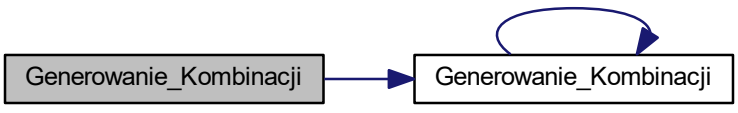
**Parametry**

| | |
|---|---|
| *k* | wielkość podanego zbioru |
| *vec* | podany zbiór |

**Zwraca**

Zwraca wektor zawierający kolejne kombinacje przedstawione w postaci wektorów kombinacji(par).

Oto graf wywołań dla tej funkcji:



#### 4.2.1.8 Naprawa()

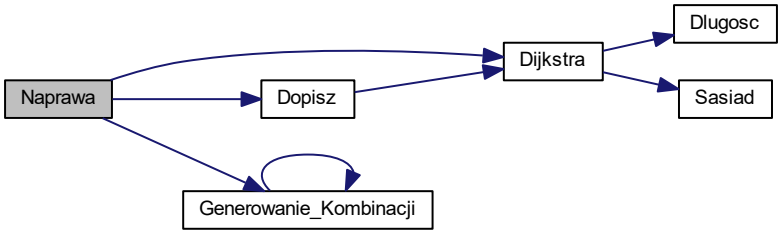
```
void Naprawa (
            vector< krawendz > & g,
            map< int, int > & m )
```

Funkcja odpowiednio zmienia graf(dopisuje nadmiarowe krawędzie), by móc wyznaczyć w nim drogę/obwód Eulera

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |

Oto graf wywołań dla tej funkcji:




#### 4.2.1.9 Nazwa_ulicy()

```
string Nazwa_ulicy (
            vector< krawendz > & g,
            int & x,
            int & y )
```

Funkcja zwracająca nazwe krawędzi łączącej podane wierzchołki w podanym grafie.

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *x* | wierzchołek grafu |
| *y* | wierzchołek grafu |

**Zwraca**

Zwraca nazwe szukanej krawędzi o ile istnieje w przeciwnym razie zwróci ("");

#### 4.2.1.10 Pobranie_Argumentow()

```
bool Pobranie_Argumentow (
            int & ile,
            char ** arg,
            string & strIN,
            string & strOUT,
            int & poczatek )
```

Funkcja przekazująca parametry funkcji main do odpowiednich zmiennych.

**Parametry**

| | |
|---|---|
| *ile* | liczba parametrów przekazanych funkcji main |
| *arg* | tablica napisów reprezentujących parametry funkcji main |
| *strIN* | zmienna do której zostanie przekazany parametr po fladze '-i' (ścieżka do pliku wejściowego) |
| *strOUT* | zmienna do której zostanie przekazany parametr po fladze '-o' (ścieżka do pliku wyjściowego) |
| *poczatek* | zmienna do której zostanie przekazany parametr po fladze '-p' (punkt startowy) |

**Zwraca**

Jeśli operacja przebiegnie poprawnie zwraca wartość true, w razie wystąpienia jakiegokolwiek błędu wyświetla odpowiedni komunikat i zwraca wartość false.

#### 4.2.1.11 Pobranie_Danych()

```
bool Pobranie_Danych (
            string & str,
            vector< krawendz > & Tgraf )
```

Funkcja pobiera dane z pliku wejściowego i przekazuje je do odpowiedniej zmiennej.

**Parametry**

| | |
|---|---|
| *str* | ścierzka do pliku wejściowego |
| *Tgraf* | zmienna, do którego przekazane zostaną dane z plik wejściowego |

**Zwraca**

Jeśli operacja przebiegnie poprawnie zwraca wartość true, w razie wystąpienia błędu zwraca wartość false.

#### 4.2.1.12 Przekaz_Wynik()

```
void Przekaz_Wynik (
            vector< krawendz > & g,
```

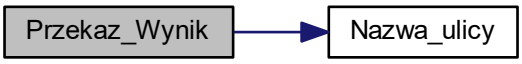
```
            string & strOUT,
            vector< int > & DE )
```

Funkcja przekazuje rozwiązanie do pliku wynikowego

**Parametry**

| | |
|---|---|
| *g* | zmienna przechowywująca graf |
| *strOUT* | ścieżka do pliku wynikowego |
| *DE* | zmienna przechowująca w wektorze drogę/obwód Eulera w postaci kolejnych wierzchołków grafu |

Oto graf wywołań dla tej funkcji:




#### 4.2.1.13 Sasiad()

```
vector<int> Sasiad (
            int & u,
            vector< krawendz > & g )
```

Funkcja zwraca numery sąsiadujących wierzchołków.

**Parametry**

| | |
|---|---|
| *u* | numer wierzchołka, którego sąsiadów szukamy |
| *g* | graf w któreym szukamy sąsiadów wierzchołka u |

**Zwraca**

Zwraca wektor przechowujący numery sąsiadujących wierzchołków.

#### 4.2.1.14 Sprawdz_Euler()

```
bool Sprawdz_Euler (
            map< int, int > & m )
```

Funkcja sprawdza czy dla danego grafu można wyznaczyc drogę Eulera.

**Parametry**

| | |
|---|---|
| *m* | zmienna przechowywująca dla każdego wierzchołka grafu jego stopień |

**Zwraca**

Jeśli jest możliwe wyznaczenie drogi Eulera to funkcja zwraca true, jeśli nie ma takiej możliwości zwraca wartość false.

#### 4.2.1.15 Stopnie()

```
map<int, int> Stopnie (
            vector< krawendz > & vec )
```

Funkcja wyznacza stopień każdego wierzchołka grafu.

**Parametry**

| | |
|---|---|
| *vec* | zmienna przechowywująca graf |

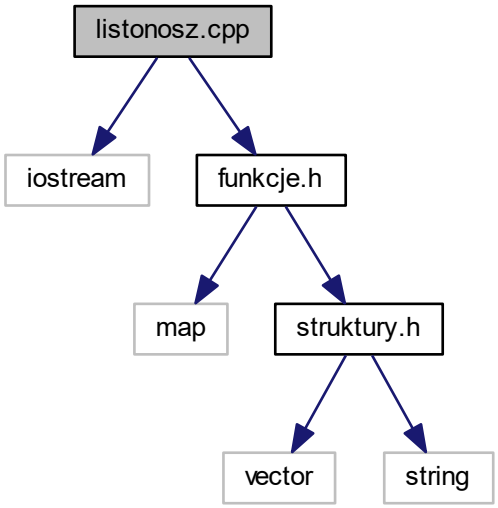
**Zwraca**

Zwraca zmienną typu 'map' przechwywującą dla każdego wierzchołka grafu jego stopień.

## 4.3 Dokumentacja pliku listonosz.cpp

```
#include <iostream>
#include "funkcje.h"
```

Wykres zależności załączania dla listonosz.cpp:




## Funkcje

- int main (int ile, char **arg)

### 4.3.1 Dokumentacja funkcji

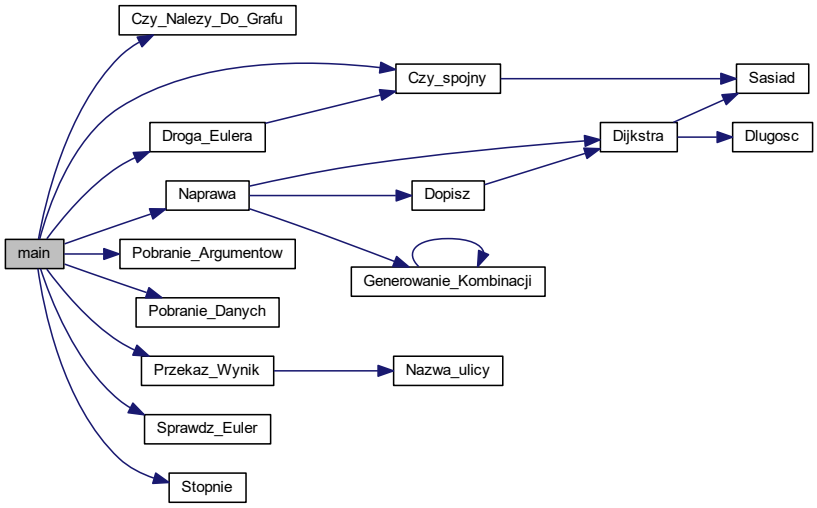
#### 4.3.1.1 main()

```
int main (
            int ile,
            char ** arg )
```

**Parametry**

| | |
|---|---|
| *ile* | liczba parametrów wywołania funkcji main |
| *arg* | tablica napisów reprezentujących parametry |

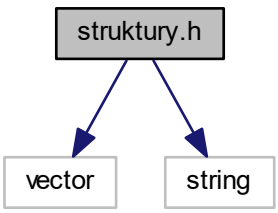
Oto graf wywołań dla tej funkcji:



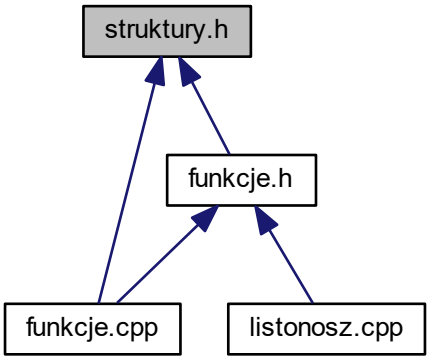

## 4.4 Dokumentacja pliku struktury.h

```
#include <vector>
#include <string>
```

Wykres zależności załączania dla struktury.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:




## Komponenty

- struct krawendz
- struct sciezka
- struct kombinacja

# Indeks