

Programowanie Komputerów II

Temat: Problem komiwojażera

Autor projektu:

Kierunek:

Semestr:

Grupa:

Prowadzący projekt:

Maciej Pałach

Teleinformatyka

III

2

dr inż. Tomasz Moroń

1. Temat

Napisać program, który będzie rozwiązywał problem komiwojażera. Powinien on działać na podstawie algorytmu genetycznego. Program musi być napisany w języku C++ zgodnie z paradygmatem programowania obiektowego.

2. Analiza i podstawy projektu

2.1. Struktury, algorytmy oraz inne elementy wymagane w programie

W programie zostały wykorzystane :

- kontenery oraz algorytmy STL, głównie kontener vector (do przechowywania elementów populacji) oraz map (krawędzie grafu)
- przeciążone operatory (porównania, indeksowania, przypisania)
- wyjątki, użyte w przypadku wpisania błędnych danych
- dziedziczenie
- polimorfizm

2.2. Opis działania programu

Program poszukuje najkrótszej ścieżki zawierającej wszystkie punkty stworzonego przez nas grafu. Najpierw wczytujemy dane dotyczące grafu, czyli liczbę punktów, punkt startowy (miejsce gdzie zaczyna podróż komiwojażer), liczbę krawędzi oraz które punkty są ze sobą połączone (mapa krawędzi). Następnie uruchamiamy algorytm genetyczny który poprzez mutacje osobników z dostępnej populacji tworzy nowe obiekty, decyduje czy są one lepsze czy słabsze od tych już istniejących i na tej podstawie odrzuca najslabsze obiekty z populacji. Po zakończeniu działania algorytmu zapisze nam się populacja zawierająca „najmocniejsze osobniki” czyli zbiór najkrótszych dostępnych ścieżek.

3. Specyfikacja zewnętrzna

3.1. Tworzenie grafu

Aby stworzyć graf do analizy musimy w kompilatorze stworzyć nowy obiekt klasy Graf. W tym celu należy wpisać poniższą komendę:

```
Graf * test_graf = new Graf(5, 0);
```

Liczby znajdujące się w nawiasie odpowiadają kolejno: Liczbie punktów z których będzie składać się nasz graf(5) oraz punktowi w którym komiwojażer zaczyna swoją podróż(0)

Drugim i ostatnim już etapem jest stworzenie krawędzi (ścieżek) po których będzie poruszał się nasz komiwojażer. W tym celu mamy do dyspozycji dwie klasy. Jedną do stworzenia krawędzi zwykłej (przejście po niej możliwe jest w obie strony) oraz drugą do stworzenia krawędzi

kierunkowej (przejście możliwe jest tylko w jednym kierunku). Aby to zrobić w kompilatorze wpisujemy następujące komendy:

```
zwykla krawedz_testowa1;
```

```
krawedz_testowa1.dodaj_krawedz(0, 1, 1, test_graf);
```

Parametry w nawiasie to kolejno: Punkt pierwszy (jeden koniec krawędzi), punkt drugi (drugi koniec krawędzi), koszt przejścia po krawędzi oraz nazwę grafu do którego będzie należeć stworzona krawędź.

Dla krawędzi kierunkowej jedyne co musimy zrobić to zmienić nazwę klasy na „kierunkowa”, a resztę wpisujemy analogicznie. Należy pamiętać jednak, że w tym przypadku ważna jest w jakiej kolejności podamy punkty, ponieważ będziemy mogli się poruszać tylko w kierunku punktu drugiego (z punktu pierwszego do drugiego)

```
kierunkowa krawedz_testowa2;
```

```
krawedz_testowa2.dodaj_krawedz(0, 1, 1, test_graf);
```

3.2. Uruchomienie algorytmu genetycznego

Gdy stworzyliśmy już graf który chcemy przeanalizować pod względem najkrótszej ścieżki musimy uruchomić algorytm genetyczny. Aby to zrobić w kompilatorze tworzymy obiekt typu AlgorytmGenetyczny. W tym celu wpisujemy następujące komendy:

```
AlgorytmGenetyczny testowy_algorytm(test_graf, 10, 100, 5, true);
```

```
testowy_algorytm.rozpocznij_algorytm();
```

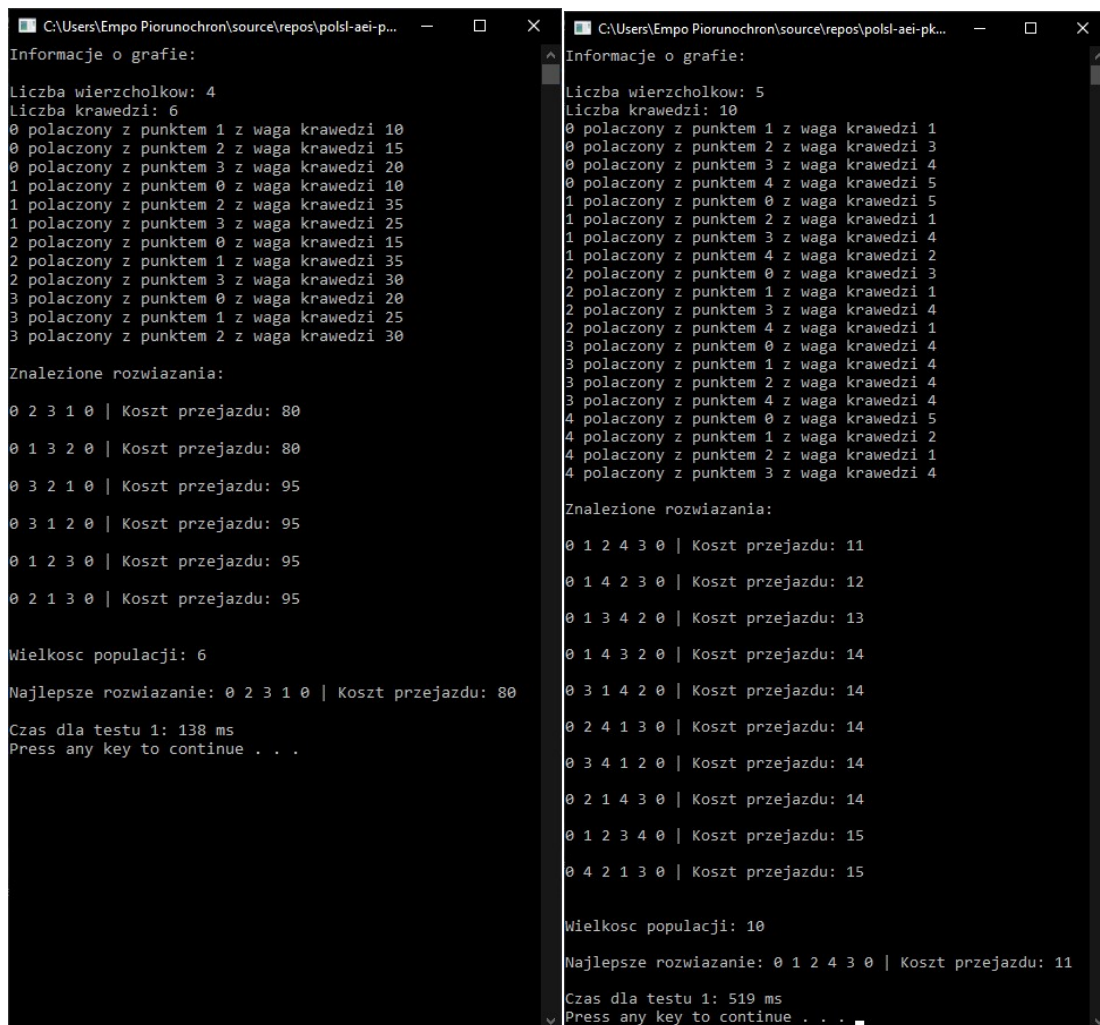
Parametry które należy wpisać przy tworzeniu nowego obiektu to: nazwa grafu który chcemy przeanalizować, wielkość populacji którą chcemy utrzymywać, liczbę generacji, współczynnik mutacji oraz czy chcemy wyświetlić końcową populację na ekranie (true/false).

3.3. Elementy do „wyświetlenia” grafu

Gdy chcemy zobaczyć jak wygląda nasz graf (wypisać liczbę punktów, krawędzi oraz wszystkie połączenia) przydatne mogą okazać się następujące komendy:

```
test_graf->info_graf();           //wyswietlamy informacje o grafie
test_graf->wyswietl_graf();       //wyswietlamy "mape" polaczen
```

4. Efekty działania programu.



```
Informacje o grafie:
Liczba wierzchołkow: 4
Liczba krawedzi: 6
0 polaczony z punktem 1 z waga krawedzi 10
0 polaczony z punktem 2 z waga krawedzi 15
0 polaczony z punktem 3 z waga krawedzi 20
1 polaczony z punktem 0 z waga krawedzi 10
1 polaczony z punktem 2 z waga krawedzi 35
1 polaczony z punktem 3 z waga krawedzi 25
2 polaczony z punktem 0 z waga krawedzi 15
2 polaczony z punktem 1 z waga krawedzi 35
2 polaczony z punktem 3 z waga krawedzi 30
3 polaczony z punktem 0 z waga krawedzi 20
3 polaczony z punktem 1 z waga krawedzi 25
3 polaczony z punktem 2 z waga krawedzi 30

Znalezione rozwiazania:
0 2 3 1 0 | Koszt przejazdu: 80
0 1 3 2 0 | Koszt przejazdu: 80
0 3 2 1 0 | Koszt przejazdu: 95
0 3 1 2 0 | Koszt przejazdu: 95
0 1 2 3 0 | Koszt przejazdu: 95
0 2 1 3 0 | Koszt przejazdu: 95

Wielkosc populacji: 6
Najlepsze rozwiazanie: 0 2 3 1 0 | Koszt przejazdu: 80
Czas dla testu 1: 138 ms
Press any key to continue . . .

Informacje o grafie:
Liczba wierzchołkow: 5
Liczba krawedzi: 10
0 polaczony z punktem 1 z waga krawedzi 1
0 polaczony z punktem 2 z waga krawedzi 3
0 polaczony z punktem 3 z waga krawedzi 4
0 polaczony z punktem 4 z waga krawedzi 5
1 polaczony z punktem 0 z waga krawedzi 5
1 polaczony z punktem 2 z waga krawedzi 1
1 polaczony z punktem 3 z waga krawedzi 4
1 polaczony z punktem 4 z waga krawedzi 2
2 polaczony z punktem 0 z waga krawedzi 3
2 polaczony z punktem 1 z waga krawedzi 1
2 polaczony z punktem 3 z waga krawedzi 4
2 polaczony z punktem 4 z waga krawedzi 1
3 polaczony z punktem 0 z waga krawedzi 4
3 polaczony z punktem 1 z waga krawedzi 4
3 polaczony z punktem 2 z waga krawedzi 4
3 polaczony z punktem 4 z waga krawedzi 5
4 polaczony z punktem 0 z waga krawedzi 2
4 polaczony z punktem 1 z waga krawedzi 1
4 polaczony z punktem 3 z waga krawedzi 4

Znalezione rozwiazania:
0 1 2 4 3 0 | Koszt przejazdu: 11
0 1 4 2 3 0 | Koszt przejazdu: 12
0 1 3 4 2 0 | Koszt przejazdu: 13
0 1 4 3 2 0 | Koszt przejazdu: 14
0 3 1 4 2 0 | Koszt przejazdu: 14
0 2 4 1 3 0 | Koszt przejazdu: 14
0 3 4 1 2 0 | Koszt przejazdu: 14
0 2 1 4 3 0 | Koszt przejazdu: 14
0 1 2 3 4 0 | Koszt przejazdu: 15
0 4 2 1 3 0 | Koszt przejazdu: 15

Wielkosc populacji: 10
Najlepsze rozwiazanie: 0 1 2 4 3 0 | Koszt przejazdu: 11
Czas dla testu 1: 519 ms
Press any key to continue . . .
```

Zdj1 - wynik dla grafu testowego 3

Zdj2 - wynik dla grafu testowego 1

5. Podsumowanie

Zostały spełnione wszystkie wymagania projektu: Projekt został napisany z paradygmatami programowania obiektowego. Wykorzystuje on dziedziczenie oraz polimorfizm wraz z wymaganymi elementami poznanymi podczas laboratoriów. Najcięższym elementem przy tworzeniu programu była poprawna implementacja algorytmu genetycznego. Poznane algorytmy i struktury oraz użycie klas znacząco ułatwiły napisanie poprawnego projektu.

6. Specyfikacja wewnętrzna

My Project

Generated by Doxygen 1.8.17

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 Class Documentation	5
3.1 AlgorytmGenetyczny Class Reference	5
3.1.1 Detailed Description	5
3.1.2 Constructor & Destructor Documentation	5
3.1.2.1 AlgorytmGenetyczny()	6
3.1.3 Member Function Documentation	6
3.1.3.1 BinarySearch()	6
3.1.3.2 czy_chromosom()	6
3.1.3.3 pobierz_wielkosc_populacji()	6
3.1.3.4 pokaz_populacje()	6
3.1.3.5 polaczenie()	7
3.1.3.6 poprawne_rozwiazanie()	7
3.1.3.7 rozpocznij_algorytm()	7
3.1.3.8 rozwiazanie()	7
3.2 Graf Class Reference	7
3.2.1 Detailed Description	8
3.2.2 Constructor & Destructor Documentation	8
3.2.2.1 Graf()	8
3.2.3 Member Function Documentation	8
3.2.3.1 czy_krawedz()	9
3.2.3.2 dodaj_krawedz()	9
3.2.3.3 info_graf()	9
3.2.3.4 pobierzPunkt()	9
3.2.3.5 wyswietl_graf()	9
3.2.4 Friends And Related Function Documentation	9
3.2.4.1 AlgorytmGenetyczny	10
3.3 kierunkowa Class Reference	10
3.3.1 Detailed Description	10
3.3.2 Member Function Documentation	10
3.3.2.1 dodaj_krawedz()	11
3.4 sortowanie Class Reference	11
3.4.1 Detailed Description	11
3.4.2 Member Function Documentation	11
3.4.2.1 operator()	11
3.5 zwykła Class Reference	12
3.5.1 Detailed Description	12
3.5.2 Member Function Documentation	12

3.5.2.1 dodaj_krawedz()	12
-------------------------	----

Index	13
--------------	-----------

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AlgorytmGenetyczny	5
Graf	7
kierunkowa	10
zwykla	12
sortowanie	11

Chapter 2

Class Index

2.1 Class List

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

AlgorytmGenetyczny	5
Graf	7
kierunkowa	10
sortowanie	11
zwykla	12

Chapter 3

Class Documentation

3.1 AlgorytmGenetyczny Class Reference

```
#include <AlgorytmGenetyczny.h>
```

Public Member Functions

- int [pobierz_wielkosc_populacji](#) () const
- void [ustaw_wielkosc_populacji](#) (int wielkosc_populacji)
- int [pobierz_rzeczywista_wielkosc_populacji](#) () const
- void [ustaw_rzeczywista_wielkosc_populacji](#) (int rzeczywista_wielkosc_populacji)
- int [pobierz_generacje](#) () const
- void [ustaw_generacje](#) (int generacje)
- int [pobierz_wspolczynnik_mutacji](#) () const
- void [ustaw_wspolczynnik_mutacji](#) (int wspolczynnik_mutacji)
- bool [pobierz_czy_pokazac_populacje](#) ()
- void [ustaw_czy_pokazac_populacje](#) (bool czy_pokazac_populacje)
- [AlgorytmGenetyczny](#) ([Graf](#) *graf, int wielkosc_populacji, int generacje, int wspolczynnik_mutacji, bool czy↔
_pokazac_populacje=true)
- void [pokaz_populacje](#) ()
- void [polaczenie](#) (std::vector< int > &rodzic1, std::vector< int > &rodzic2)
- void [BinarySearch](#) (std::vector< int > &dziecko, int calkowity_koszt)
- void [rozpocznij_algorytm](#) ()
- int [poprawne_rozwiazanie](#) (std::vector< int > &rozwiazanie)
- int [rozwiazanie](#) ()
- bool [czy_chromosom](#) (const std::vector< int > &Punkt)

3.1.1 Detailed Description

Klasa reprezentująca algorytm genetyczny

3.1.2 Constructor & Destructor Documentation

3.1.2.1 AlgorytmGenetyczny()

```
AlgorytmGenetyczny::AlgorytmGenetyczny (
    Graf * graf,
    int wielkosc_populacji,
    int generacje,
    int wspolczynnik_mutacji,
    bool czy_pokazac_populacje = true )
```

Konstruktor domyslny Konstruktor inicjalizujacy skladowe algorytmu genetycznego

3.1.3 Member Function Documentation

3.1.3.1 BinarySearch()

```
void AlgorytmGenetyczny::BinarySearch (
    std::vector< int > & dziecko,
    int calkowity_koszt )
```

Metoda uzywajaca poszukiwania binarnego do znalezienia najlepszego osobnika populacji (najkrotszej sciezki przejścia po wszystkich punktach)

3.1.3.2 czy_chromosom()

```
bool AlgorytmGenetyczny::czy_chromosom (
    const std::vector< int > & Punkt )
```

Metoda sprawdzajaca czy dany chromosom istnieje

3.1.3.3 pobierz_wielkosc_populacji()

```
int AlgorytmGenetyczny::pobierz_wielkosc_populacji ( ) const [inline]
```

Settery i gettery do pobierania i ustawiania parametrow algorytmu z zewnatrz klasy

3.1.3.4 pokaz_populacje()

```
void AlgorytmGenetyczny::pokaz_populacje ( )
```

Metoda wyswietlajaca na ekranie wynikowym populacje

Returns

Populacja: Elementy skladowe sie ze sciezki i kosztu przejścia po niej

< Pobieramy wektor

3.1.3.5 polaczenie()

```
void AlgorytmGenetyczny::polaczenie (
    std::vector< int > & rodzic1,
    std::vector< int > & rodzic2 )
```

Metoda robiaca crossover i mutacje pomiedzy osobnikami populacji

3.1.3.6 poprawne_rozwiazanie()

```
int AlgorytmGenetyczny::poprawne_rozwiazanie (
    std::vector< int > & rozwiazanie )
```

Metoda sprawdzajaca czy podane do niej rozwiazanie jest poprawne i czy takie juz nie wystepuje w populacji

3.1.3.7 rozpocznij_algorytm()

```
void AlgorytmGenetyczny::rozpocznij_algorytm ( )
```

Metoda rozpoczynajaca prace algorytmu genetycznego

Returns

Wynik koncowy dzialania programu

3.1.3.8 rozwiazanie()

```
int AlgorytmGenetyczny::rozwiazanie ( )
```

Metoda zwracajaca najlepsze rozwiazanie

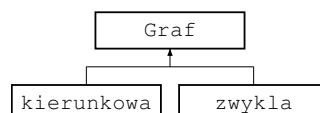
The documentation for this class was generated from the following files:

- C:/Users/Empo Piorunochron/source/repos/polsl-aei-pk2-tele/10fba932-gr21-repo/Projekt/komiwojazer/komiwojazer/AlgorytmGenetyczny.h
- C:/Users/Empo Piorunochron/source/repos/polsl-aei-pk2-tele/10fba932-gr21-repo/Projekt/komiwojazer/komiwojazer/AlgorytmGenetyczny.cpp

3.2 Graf Class Reference

```
#include <Graf.h>
```

Inheritance diagram for Graf:



Public Member Functions

- int [pobierzPunkt](#) () const
- void [ustawPunkt](#) (int Punkt)
- int [pobierzkrawedzie](#) () const
- void [ustawkrawedzie](#) (int krawedzie)
- int [pobierzpunkt_poczatkowy](#) () const
- void [ustawpunkt_poczatkowy](#) (int punkt_poczatkowy)
- [Graf](#) (int Punkt, int punkt_poczatkowy)
- virtual void [dodaj_krawedz](#) (int Punkt1, int Punkt2, int koszt, [Graf](#) *graf)
- void [wyswietl_graf](#) ()
- void [info_graf](#) ()
- int [czy_krawedz](#) (int Punkt1, int Punkt2)

Public Attributes

- std::map< std::pair< int, int >, int > [mapa_krawedzi](#)

Friends

- class [AlgorytmGenetyczny](#)

3.2.1 Detailed Description

Klasa reprezentująca graf

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Graf()

```
Graf::Graf (
    int Punkt,
    int punkt_poczatkowy )
```

Konstruktor domyslny Konstruktor inicjalizujacy skladowe grafu

3.2.3 Member Function Documentation

3.2.3.1 czy_krawedz()

```
int Graf::czy_krawedz (
    int Punkt1,
    int Punkt2 )
```

Metoda sprawdzająca czy krawedz z jednego punktu do drugiego istnieje

3.2.3.2 dodaj_krawedz()

```
virtual void Graf::dodaj_krawedz (
    int Punkt1,
    int Punkt2,
    int koszt,
    Graf * graf ) [inline], [virtual]
```

Metoda wirtualna dodająca krawedz do grafu

Reimplemented in [kierunkowa](#), and [zwykla](#).

3.2.3.3 info_graf()

```
void Graf::info_graf ( )
```

Metoda wyświetlająca na ekranie wynikowym podstawowe informacje dotyczące grafu czyli liczbę punktów oraz krawedzi

3.2.3.4 pobierzPunkt()

```
int Graf::pobierzPunkt ( ) const [inline]
```

Settery i gettery do pobierania i ustawiania parametrów grafu z zewnątrz klasy

3.2.3.5 wyswietl_graf()

```
void Graf::wyswietl_graf ( )
```

Metoda wyświetlająca na ekranie wynikowym wszystkie połączenia dostępne w grafie razem z kosztem przejścia po nich (mapa połączeń)

3.2.4 Friends And Related Function Documentation

3.2.4.1 AlgorytmGenetyczny

```
friend class AlgorytmGenetyczny [friend]
```

Uzyskanie dostępu do private klasy AlgorytmuGenetycznego

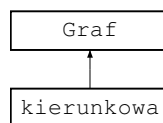
The documentation for this class was generated from the following files:

- C:/Users/Empo Piorunochron/source/repos/polsl-aei-pk2-tele/10fba932-gr21-repo/Projekt/komiwojazer/komiwojazer/Graf.↵h
- C:/Users/Empo Piorunochron/source/repos/polsl-aei-pk2-tele/10fba932-gr21-repo/Projekt/komiwojazer/komiwojazer/Graf.↵cpp

3.3 kierunkowa Class Reference

```
#include <kierunkowa.h>
```

Inheritance diagram for kierunkowa:



Public Member Functions

- virtual void [dodaj_krawedz](#) (int Punkt1, int Punkt2, int koszt, [Graf](#) *graf) override

Additional Inherited Members

3.3.1 Detailed Description

Klasa reprezentująca krawedz kierunkowa.

3.3.2 Member Function Documentation

3.3.2.1 dodaj_krawedz()

```
void kierunkowa::dodaj_krawedz (
    int Punkt1,
    int Punkt2,
    int koszt,
    Graf * graf ) [override], [virtual]
```

Konstruktor domyslny Metoda dodajaca krawedz do grafu

Reimplemented from [Graf](#).

The documentation for this class was generated from the following files:

- C:/Users/Empo Piorunochron/source/repos/polsl-aei-pk2-tele/10fba932-gr21-repo/Projekt/komiwojazer/komiwojazer/kierunkowa.h
- C:/Users/Empo Piorunochron/source/repos/polsl-aei-pk2-tele/10fba932-gr21-repo/Projekt/komiwojazer/komiwojazer/kierunkowa.cpp

3.4 sortowanie Class Reference

```
#include <sortowanie.h>
```

Public Member Functions

- bool [operator\(\)](#) (const para &trasa1, const para &trasa2)

3.4.1 Detailed Description

Klasa reprezentujaca sortowanie

3.4.2 Member Function Documentation

3.4.2.1 operator()()

```
bool sortowanie::operator() (
    const para & trasa1,
    const para & trasa2 ) [inline]
```

Metoda zwracajaca posortowane dane pod wzgledem kosztu przejścia przez wszystkie punkty od najkrotszej do najdluzszej

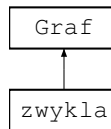
The documentation for this class was generated from the following file:

- C:/Users/Empo Piorunochron/source/repos/polsl-aei-pk2-tele/10fba932-gr21-repo/Projekt/komiwojazer/komiwojazer/sortowanie.h

3.5 zwykla Class Reference

```
#include <zwykla.h>
```

Inheritance diagram for zwykla:



Public Member Functions

- virtual void [dodaj_krawedz](#) (int Punkt1, int Punkt2, int koszt, [Graf](#) *graf) override

Additional Inherited Members

3.5.1 Detailed Description

Klasa reprezentująca krawedz kierunkowa.

3.5.2 Member Function Documentation

3.5.2.1 dodaj_krawedz()

```
void zwykla::dodaj_krawedz (
    int Punkt1,
    int Punkt2,
    int koszt,
    Graf * graf ) [override], [virtual]
```

Konstruktor domyslny Metoda dodajaca krawedz do grafu

Reimplemented from [Graf](#).

The documentation for this class was generated from the following files:

- C:/Users/Empo Piorunochron/source/repos/polsl-aei-pk2-tele/10fba932-gr21-repo/Projekt/komiwojazer/komiwojazer/zwykla.↔h
- C:/Users/Empo Piorunochron/source/repos/polsl-aei-pk2-tele/10fba932-gr21-repo/Projekt/komiwojazer/komiwojazer/zwykla.↔cpp

Index

- AlgorytmGenetyczny, [5](#)
 - AlgorytmGenetyczny, [5](#)
 - BinarySearch, [6](#)
 - czy_chromosom, [6](#)
 - Graf, [9](#)
 - pobierz_wielkosc_populacji, [6](#)
 - pokaz_populacje, [6](#)
 - polaczenie, [6](#)
 - poprawne_rozwiazanie, [7](#)
 - rozpocznij_algorytm, [7](#)
 - rozwiazanie, [7](#)
- BinarySearch
 - AlgorytmGenetyczny, [6](#)
- czy_chromosom
 - AlgorytmGenetyczny, [6](#)
- czy_krawedz
 - Graf, [8](#)
- dodaj_krawedz
 - Graf, [9](#)
 - kierunkowa, [10](#)
 - zwykla, [12](#)
- Graf, [7](#)
 - AlgorytmGenetyczny, [9](#)
 - czy_krawedz, [8](#)
 - dodaj_krawedz, [9](#)
 - Graf, [8](#)
 - info_graf, [9](#)
 - pobierzPunkt, [9](#)
 - wyswietl_graf, [9](#)
- info_graf
 - Graf, [9](#)
- kierunkowa, [10](#)
 - dodaj_krawedz, [10](#)
- operator()
 - sortowanie, [11](#)
- pobierz_wielkosc_populacji
 - AlgorytmGenetyczny, [6](#)
- pobierzPunkt
 - Graf, [9](#)
- pokaz_populacje
 - AlgorytmGenetyczny, [6](#)
- polaczenie
 - AlgorytmGenetyczny, [6](#)
- poprawne_rozwiazanie
 - AlgorytmGenetyczny, [7](#)
- rozpocznij_algorytm
 - AlgorytmGenetyczny, [7](#)
- rozwiazanie
 - AlgorytmGenetyczny, [7](#)
- sortowanie, [11](#)
 - operator(), [11](#)
- wyswietl_graf
 - Graf, [9](#)
- zwykla, [12](#)
 - dodaj_krawedz, [12](#)