1 Treść zadania

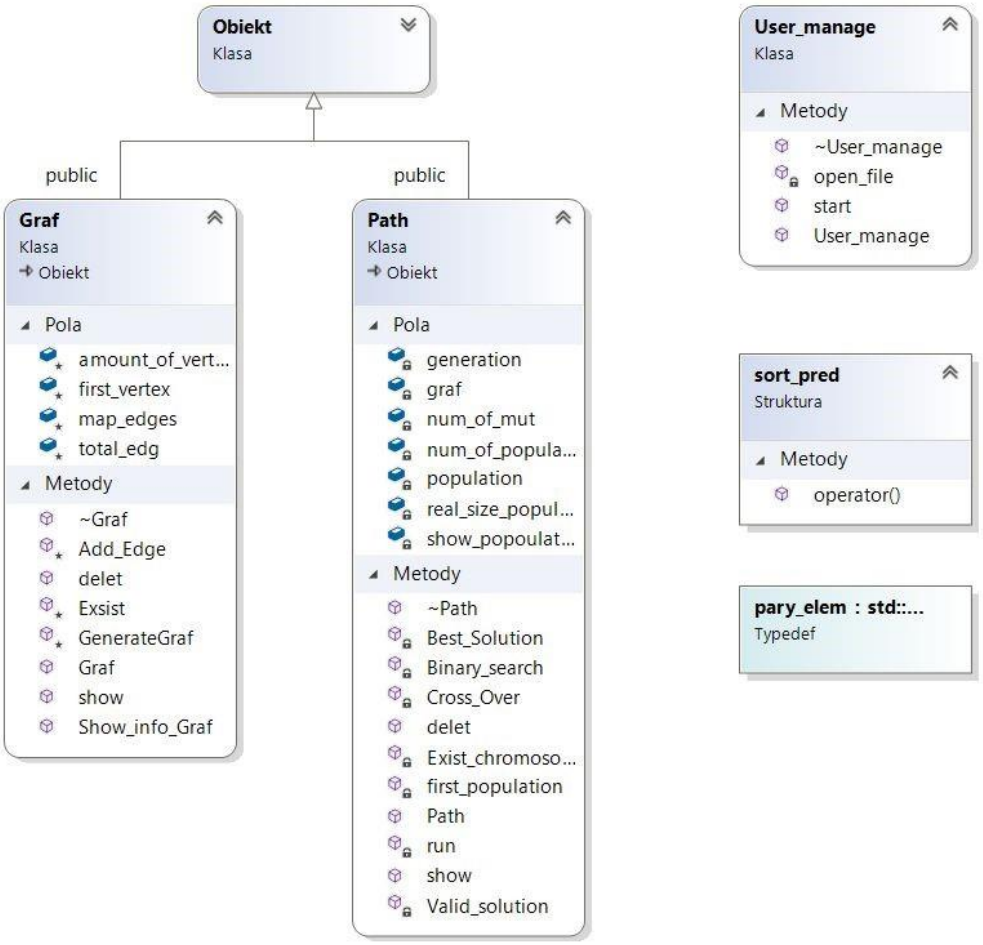
Napisać program który rozwiąże problem komiwojażera za pomocą algorytmu genetycznego. Przyjęte zostały tutaj pewne założenia w celu sprecyzowania zasad działania programu :

- Każdy wierzchołek w grafie jest połączony ze wszystkimi pozostałymi wierzchołkami.
- Komiwojażer przechodzi przez każdy wierzchołek tylko raz.
- Droga jaką wyznaczy program zaczyna się w wybranym punkcie np.“A” oraz się w nim kończy.

2 Struktury Danych

W programie wykorzystano podział na klasy oraz zastosowano polimorfizm. Logika aplikacji została odseparowana od komunikacji z użytkownikiem. Dodatkowo użyto mapę par, w której punkty pomiędzy którymi występuje krawędź są kluczem a wartości są odległością pomiędzy nimi.

3 Specyfikacja



Poniższy obrazek przedstawia poglądowo diagram klas użytych w programie.

Program jest uruchamiany z linii poleceń jednak bez argumentów wejściowych ponieważ możliwe są dwa warianty:

Pierwszy to automatyczna generacja grafu oraz przeprowadzenie całego algorytmu na losowo wygenerowanym grafie. Program oczekuje w tym momencie podania poniższych wartości :

- Liczbę wierzchołków
- Liczbę generacji (ilość iteracji dla algorytmu genetycznego)
- Współczynnik mutacji (ilość ścieżek, w których dojdzie do mutacji)

Drugi wariant odczytuje z pliku tekstowego krawędzie grafu oraz prosi użytkownika o podanie następujących danych :

- Liczbę wierzchołków
- Wartość pierwszego wierzchołka

Odczytanie z pliku odbywa się za pomocą metody **`open_file()`** która odczytuje linijka po linijce plik tekstowy oraz przerywa pracę programu jeśli użytkownik poda złą ścieżkę.

```
25;1000;3
0;1;1;
0;2;3;
0;3;4;
0;4;5;
1;2;1;
1;3;4;
1;4;8;
2;3;5;
2;4;1;
3;4;2;
```

przykładowa zawartość pliku tekstowego

W pierwszej linijce pliku tekstowego należy podać odpowiednio ilość krawędzi, ilość generacji oraz współczynnik mutacji.

---

4 Typy zdefiniowane w programie

W programie zdefiniowano następujące klasy:

```
class Obiekt
{
public:
     virtual void show()=0;// wypisz virtual
     void delet() = 0;//usuń

     virtual ~Obiekt() {};// dekonstruktor
};
```

Jest to nadrzędna klasa czysto wirtualna, jej metody są nadpisane przez Klasy dziedziczące.

```
      class Graf : public Obiekt
{
protected:
     //std::vector<Point>punkty;
 int amount_of_verticles; // liczba wierzcholkow int total_edg;
     // ilosc krawedzi int first_vertex; //pierwszy wierzcholek
     std::map<std::pair<int,    int>,    int>map_edges;    void
     Add_Edge(int X, int Y, int weight);// dodaje krawędź
dodaj do protected void GenerateGraf();// generuje losowy graf
     int Exsist(int x, int y);//sprawdza czy istnieje już
jakis graf
public:
        Graf(int number, int first, bool random_graph = false);
//konstruktor
  virtual  void  show();      void  Show_info_Graf();//pokazuje
     informacje o grafie   void delet() override {};  ~Graf()
     {};  friend class Path; //przyjaźn po to żeby sciezka miała
dostep do prywatnych zmiennych friend
     class User_manage;
};
```

Jest to klasa pochodna od klasy Obiekt, przechowywane są tutaj wartości potrzebne do opisania Grafu, takie jak liczba wierzchołków, ilość krawędzi grafu, wartość pierwszego wierzchołka, mapa krawędzi, z których zbudowany jest graf, oraz metody takie jak `void Add_Edge(int X, int Y, int weight);` która dodaje kolejne krawędzie do mapy, `void GenerateGraf();` generująca losowy graf w przypadku wybrania pierwszego wariantu programu.
`int Exsist(int x, int y);` sprawdzająca czy istnieje już jakiś graf.

```
class Path : public Obiekt
{
     Graf* graf;//obiekt typu graf
     std::vector<pary_elem>population; // każdy element pary
to jeden punkt vector i koszt(odleglosc) int
     num_of_populacji;//size of population int
     real_size_population;//real sizee int
     generation;//amount of generations int
     num_of_mut;//mutiation rate bool
     show_popoulation;//flag to show population
 void first_population();//generuje początkową populację int
     Valid_solution(std::vector<int>& solution);
     voidCross_Over(std::vector<int>&parent1,std::vector<int>&
parent2); void Binary_search (std::vector <int>& child, int
total_cost ); void run(); int Best_Solution(); bool
     Exist_chromosome(const std::vector<int>& v);
public:

     Path(Graf* graf, int amount_of_population, int
generations, int mutation_rate, bool show_population = true);
       virtual void show() ;
     void delet() override
     {};
     ~Path() {}; friend class
     Graf; friend class
     User_manage;
};
```

Jest to druga klasa pochodna klasy Obiekt, przechowywane są w niej zmienne opisujące kolejne ścieżki oraz zmienne użyte w procesie wybierania najkrótszej drogi algorytmem genetycznym. `void first_population();` generuje początkową populację. `int Valid_solution(std::vector<int>& solution);` z kolei ta metoda zapisuje do kontenera set otrzymany wektor i sprawdza czy nie zawiera zdublowanych wartości oraz czy wszystkie elementy są połączone poprawnie w wyniku tej metody zwrócone zostaje całkowity koszt ścieżki lub wartość -1 jeżeli metoda ta wykryje błąd w ścieżce. `voidCross_Over (std::vector <int> &parent1 ,std::vector<int>& parent2);` Przeprowadza krzyżowanie się ścieżek ze sobą w efekcie czego powstają nowe.

`void Binary_search (std::vector <int>& child, int total_cost );` metoda ta jest używana w celu dodania nowej ścieżki powstałej w wyniku krzyżowania do wektora par population. `void run();` metoda ta jest odpowiedzialna za przeprowadzenie całego algorytmu programu wywołując w odpowiednich miejscach kolejne metody oraz sprawdzając wyniki przez nie zwracane. `int Best_Solution();` metoda która zwraca najlepszą ścieżkę w grafie. `bool Exist_chromosome(const std::vector<int>& v); public:` sprawdza czy istnieje już taka ścieżka w mapie krawędzi i zwraca true jeśli nie istnieje, w przeciwnym wypadku zwróci false.

```
struct sort_pred
{ bool operator()(const pary_elem& firstElem, const
pary_elem& secondElem)
     { return firstElem.second < secondElem.second; }
};
```

Jest to struktura użyta do posortowania par w wektorze, użyta w metodzie first population

---

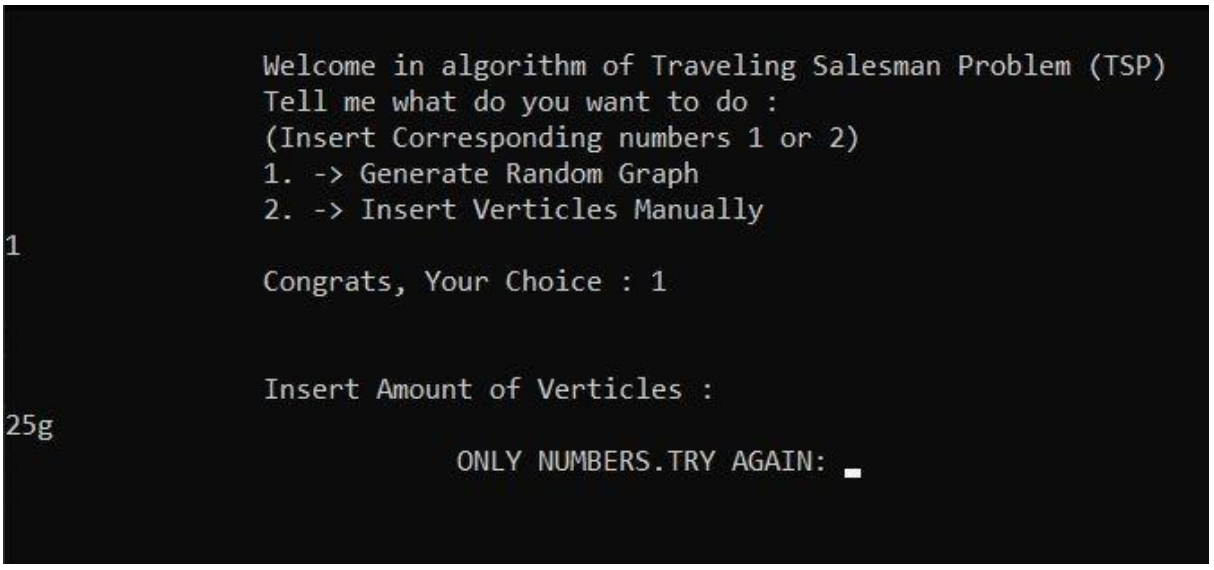
```
class User_manage
{ void open_file(Graf* graf2);
public:
     User_manage() {};
     void start();
     friend class Graf;
     friend class Path;
     ~User_manage() {};
};
```

Klasa używana do komunikacji z użytkownikiem, void start() rozpoczyna pracę programu od zapytania użytkownika o wszystkie potrzebne parametry, oraz w drugim wariancie o dokładną ścieżkę do pliku tekstowego. Jednocześnie metoda ta sprawdza poprawność danych wpisywanych przez użytkownika, a przy błędnym podaniu prosi o ponowne wpisanie danej wartości aż do skutku. później wywołuje konstruktor klasy Graf oraz wywołuje metodę run() klasy Path, która była opisana wyżej.

---

5 *Testowanie*

Program został przetestowany pod kątem wprowadzenia błędów przez użytkownika.Przykładem może być sytuacja początkowa w której użytkownik proszony jest o wprowadzenie danych liczbowych, w momencie podania niepoprawnych wartości np gdy zamiast “25” użytkownik poda “25g” program poprosi o ponowne wpisanie danej wielkości ze stosownym alarmem.



W momencie podania złej ścieżki do pliku tekstowego program kończy działanie z stosownym alertem: