Problem komiwojażera:  
opis rozwiązania przy pomocy symulowanego wyżarzania dla grafu spójnego skierowanego.

Autorzy:  
Tomasz Kogut  
Mateusz Mielnicki

Konsultacja merytoryczna:  
dr inż. Sebastian Kozłowski

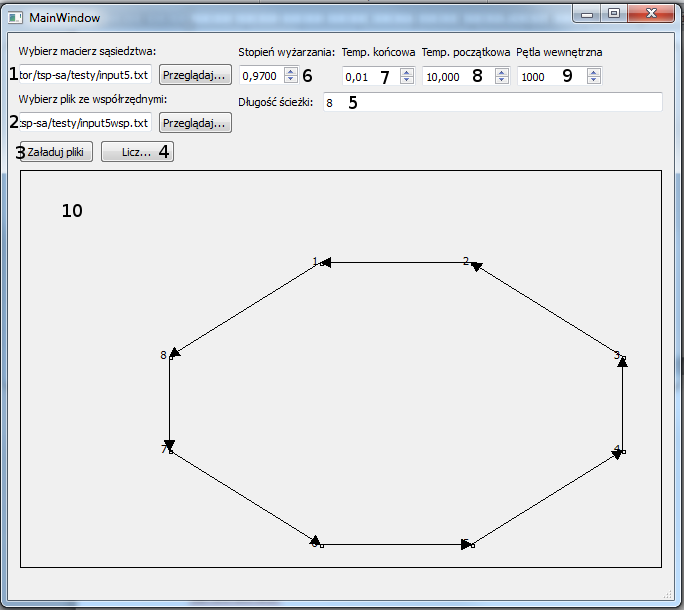
**Streszczenie**

**Poniższy dokument jest opisem realizacji projektu z przedmiotu „Grafy i sieci”. Tematem projektu było zaimplementowanie programu na komputer osobisty PC, który wczyta graf miast z pliku i rozwiąże problem komiwojażera używając symulowanego wyżarzania. W dokumencie czytelnik znajdzie szczegóły implementacyjne oraz opis zastosowania języka c++ oraz biblioteki QT do rozwiązania zadania. Przeprowadzamy również analizę działania programu w zależności od liczby miast i połączeń w grafie.**

**26 maja 2011**

# Opis graficznego interfejsu użytkownika

## Elementy interfejsu



Ilustracja 1: Zrzut ekranu z programu

Ilustracja [1] prezentuje zrzut ekranu z wyniku działania programu dla 8 miast wraz z nałożonymi numerami, które są referencją dla dostępnych opcji.

1. Lokalizacja pliku z macierzą sąsiedztwa.
2. Plik ze współrzędnymi miast (opcjonalny).
3. Przycisk ładujący dane wejściowe.
4. Przycisk uruchamiający działanie algorytmu.
5. Pole wynikowe prezentujące wynik działania algorytmu – długość znalezionej ścieżki.
6. Poziom wyżarzania używany przy obliczeniach.
7. Temperatura końcowa, której osiągnięcie zatrzymuje algorytm.
8. Temperatura początkowa na poziomie której algorytm zaczyna działanie.
9. Liczba iteracji dla stałego poziomu temperatury.
10. Płotno obrazujące wprowadzone współrzędne miast. Po i w trakcie działania algorytmu, obrazuje również ścieżki między miastami wraz z zaznaczonym kierunkiem.

## Wykonanie programu

Aby wykorzystać program do wykonania obliczeń należy wykonać następującą sekwencje czynności:

* Wskazać lokalizacje pliku tekstowego zawierającego macierz sąsiedztwa.
* (Opcjonalnie) Wskazać plik zawierający współrzędne miast. Jeżeli plik nie zostanie wskazany, współrzędne zostaną wylosowane.
* Ustalić żądane parametry algorytmu: stopień wyżarzania, temperaturę początkową, temperaturę końcową i pętlę wewnętrzną.
* Kliknąć w przycisk „Załaduj pliki”, co powinno nanieść miasta na płótno wraz z numerami.
* Kliknąć w przycisk licz co uruchomi algorytm.

W trakcie liczenia płótno będzie na bieżąco aktualizowane siecią dróg, które były rozważane w trakcie działania algorytmu.

W momencie gdy algorytm się zakończy, sieć dróg ustabilizuje się na najlepszej znalezionej trasie i pojawi się liczba oznaczająca długość trasy.

# Szczegóły implementacyjne

## Architektura programu

Lista klas i struktur:

* Graficzne:
  + MainWindow – reprezentuje widoczny dla użytkownika interfejs. Steruje uruchomieniem algorytmu oraz prezentuje wynik
  + Draw – reprezentuje płótno, które obrazuje rozmieszczenie miast oraz obliczoną ścieżkę.
* Funkcjonalne
  + InputReader – reprezentuje obiekt odpowiedzialny za wczytywanie danych wejściowych i dostarczanie ich obiektowej reprezentacji do pozostałych obiektów.
  + TSPSolver – reprezentuje obiekt obliczeniowy, który zawiera algorytmy potrzebne do przeprowadzenia wszystkich obliczeń.
* Pomocnicze
  + CityPosition – klasa reprezentująca położenie miasta we współrzędnych kartezjańskich
  + RoadPart – struktura reprezentująca połączenie między dwoma miastami. Jest to uporządkowana para identyfikatorów miast odpowiadająca wierszom w macierzy sąsiedztwa.
  + ComputeThread – obiekt reprezentujący wykonanie algorytmu w oddzielnym wątku. Służy do uruchamiania algorytmu oraz odbierania i przekazywania do okna graficznego.

## Opis algorytmów obliczających trasę.

### Dane wstępne

Za całość obliczeń odpowiada klasa TSPSolver, która zawiera metody dokonujące kolejnych kroków obliczeń. Do działania potrzebuje następujących informacji:

* Macierzy sąsiedztwa.
* Liczby miast.
* Początkową temperaturę.
* Końcowa temperatura.
* Stopień wyżarzania.

### Generacja trasy początkowej

Po podaniu danych wstępnych TSPSolver generuje trasę początkową:

* Start następuje poprzez wylosowanie jednego z wierzchołków.
* Wszystkie miasta zostają oznaczone w pomocniczej tablicy jako nieodwiedzone.
* Metodą zachłanną przechodzimy do kolejnego sąsiada czyli wybierane jest miasto, które znajduje się najbliżej.
* Miasto które opuszczamy oznaczane jest w tablicy jako odwiedzone.
* Jeżeli wszyscy sąsiedzi byli już odwiedzeni to poprzez wyszukiwanie w głąb szukamy jeszcze nieodwiedzone miasta i powtarzamy algorytm.
* Algorytm działa dopóki licznik nieodwiedzonych miast nie spadnie do 1.
* Jeżeli licznik odwiedzonych miast spadnie do 1 to oznacza, że wszystkie miasta zostały odwiedzone.
* Szukamy drogi powrotnej do punktu wylosowanego na początku, od którego startowaliśmy poprzez wyszukiwanie w głąb.

Jeżeli nie jesteśmy w stanie znaleźć wierzchołka który nie został jeszcze odwiedzony poprzez przeszukiwanie w głąb a licznik jest > 1 to znaczy, że podany graf nie jest możliwy do obejścia i algorytm się kończy.

### Generacja sąsiedztwa.

Mati działaj ☺

### Symulowanie wyżarzanie.

Mati działaj!

# Opis danych wejściowych i wyjściowych

## Generacja danych testowych

Ponieważ aby przetestować działanie algorytmu należy wygenerować szereg grafów, które będą:

* spełniać warunek możliwości odwiedzenia wszystkich miast,
* będą różnych, często znacznych rozmiarów,

cały proces należy zautomatyzować. Do programu dołączany jest skrypt w napisany w języku Python. Skrypt *„generate\_matrix.py”* przyjmuje dwa parametry, pierwszy mówiący o liczbie miast a drugi liczbę połączeń między miastami.

Aby zapewnić warunek, że wszystkie miasta można obejść i wrócić do punktu wyjścia, skrypt łączy w pierwszym kroku w cykl wszystkie miasta. Następnie pozostała żądana liczba połączeń jest generowana losowo.

Skrypt drukuje losową macierz sąsiedztwa na standardowe wyjście w formacie zgodnym z tym, który program rozwiązujący problem komiwojażera przyjmuje na wejście.

Liczba miast nie ma narzuconego ograniczenia wynikającego ze specyfiki programu. Jedynym ograniczeniem jest moc obliczeniowa komputera.

Liczba połączeń między miastami zależy od liczby miast „n”, która należy do przedziału: <n,n(n-1)>. Dolne ograniczenie wynika z potrzeby zapewnienia cyklu. Górne ograniczenie daje graf pełny, gdzie każde miasto ma połączenie z innym.

Program testowano na maksymalnie 500 wierzchołkach, gdzie każde miasto miało połączenie z co drugim miastem przy 2GB pamięci.

## Wyjście

Program po wykonaniu obliczeń prezentuje graficzną reprezentacje znalezionej trasy odmalowaną na płótnie oraz jedną liczbę, sumę wag krawędzi.

# Analiza działania programu

Działaj ☺

Spis treści.

[1. Opis graficznego interfejsu użytkownika 2](#_Toc294212189)

[1.1. Elementy interfejsu 2](#_Toc294212190)

[1.2. Wykonanie programu 3](#_Toc294212191)

[2. Szczegóły implementacyjne 4](#_Toc294212192)

[2.1. Architektura programu 4](#_Toc294212193)

[2.2. Opis algorytmów obliczających trasę. 4](#_Toc294212194)

[2.2.1. Dane wstępne 4](#_Toc294212195)

[2.2.2. Generacja trasy początkowej 4](#_Toc294212196)

[2.2.3. Generacja sąsiedztwa. 5](#_Toc294212197)

[2.2.4. Symulowanie wyżarzanie. 5](#_Toc294212198)

[3. Opis danych wejściowych i wyjściowych 6](#_Toc294212199)

[4. Analiza działania programu 7](#_Toc294212200)