Borysionek Malwina
14 II 2016
wersja 2.

Problem komiwojażera
DOKUMENTACJA

Program polega na znajdowaniu w przybliżeniu najbardziej optymalnej drogi podróżującego komiwojażera w zadanej liczbie iteracji.

Problem komiwojażera dotyczy znalezienia najkrótszej trasy przechodzącej przez wszystkie podane miasta i postanowiłam go rozwiązać z wykorzystaniem grafu Hamiltona oraz algorytmów genetycznych.

Graf Hamiltona jest specjalnym rodzajem grafu, w którym istnieje ścieżka przechodząca przez każdy wierzchołek dokładnie raz.
Jeśli taka ścieżka się zamyka – jest cyklem – nazywamy ją **cyklem Hamiltona**.

Program składa się z następujących modułów:
*** main** – uruchamia wszystkie funkcje

Użytkownik na samym początku musi zdecydować, czy chce wygenerować graf połaczeń pomiędzy miastami, czy skorzystać z grafu zapisanego w pliku.
Druga opcja jest bardzo pożyteczna w przypadku, gdy chcemy wielokrotnie testować działanie programu (np. zmieniajac liczbę iteracji i badając jak wpływa na wynik) dla takich samych odległości pomiędzy miastami i porównywać wyniki.

*** complete_graph.py** – moduł z klasą grafu pełnego

Graf pełny będzie w przypadku tego programu oznaczał połączenia między miastami (zakładamy, że między każdymi dwoma miastami ze zbioru istnieje połączenie), wagi krawędzi oznaczają odległości pomiędzy miastami.
Dla ułatwienia zakładamy, że między każdym miastem istnieje połączenie, więc program działa na grafie pełnym ważonym.
Wagi są liczbami całkowitymi, lecz można to zmienić w module complete_graph.py, zamieniając linijkę

```
w = randint(1, weight)
```

na:

```
w = randint(1, weight) + random()
```

oraz dodać rozsądne zaokrąglenia, gdzie jest to wymagane.
Aby lepiej zilustrować działanie programu, wykonywane są konwersje liczb do znaków z tablicy ASCII, więc w wypadku, gdy chcemy obejrzeć wypisywane przez program komunikaty, liczba wierzchołków nie może być zbyt duża.

Reprezentacją grafu pełnego w tym przypadku jest słownik + listy.
Tworząc obiekt klasy CompleteGraph podajemy jako argument obowiązkowo liczbę wierzchołków.
Drugim argumentem jest 'dict_graph', czyli możemy utworzyć obiekt klasy CompleteGraph ze stworzonego ręcznie słownika, zaś w przypadku wpisania "g" dla zadanej liczby wierzchołków zostanie wygenerowany graf pełny z losowymi wagami krawędzi, jeśli nie podamy nic zadziała wartość domyślna None i zostanie wygenerowany graf pusty, do którego później będzie można dodawać wierzchołki i krawędzie poprzez funkcje 'add_vertex' i 'add_edge'.

W klasie są zaimplementowane podstawowe metody do generowania grafu pełnego, funkcja odczytująca wagę przypisaną do krawędzi między dwoma danymi wierzchołkami, funkcja do wypisywania grafu, oraz funkcje: zapisująca graf do pliku i odczytująca graf z pliku.

* **hamilton_cycle.py** – moduł z klasą cyklu Hamiltona

Klasa CyklHamiltona reprezentuje ciąg wierzchołków ze zbioru wierzchołków, co symbolizuje trasę, każdy wierzchołek oznacza miasto.
Taki cykl musi zawierać w sobie wszystkie wierzchołki-miasta ze zbioru, każdy dokładnie raz.
Cykl reprezentowany jest za pomocą listy.
Przy tworzeniu obiektu możemy podać w parametrze H_C inny cykl, wtedy zostanie on skopiowany, możemy również stworzyć cykl pustych wierzchołków (jest to potrzebne przy algorytmach genetycznych, dokładniej - przy krzyżowaniu).
Klasa posiada metodę do sumowania cyklu, czyli określania długości trasy, gdy chce się przejść wszystkie miasta w podanej kolejności.
Długość cyklu jednocześnie oznacza jak dobry jest, im cykl krótszy, tym jest lepszy.
Takie założenie zostało przyjęte przy realizacji funkcji porównującej cykle (__cmp__).

*** genetic_algorithms** – moduł, zawierający funkcje z algorytmami genetycznymi

Algorytmy genetyczne są specjalną klasą algorytmów, które wyszukują najlepsze rozwiązania problemu poprzez przeszukiwanie przestrzeni alternatywnych rozwiązań.
Wprowadza się w nich pewną startową bazę rozwiązań, tzw. populację i poprzez modyfikowanie jej, odpowiednie ewolucje, dąży się do uzyskania najlepszego wyniku.

Funkcja '**tournament**' pozwala na selekcję użytkowników.
Za jej pomocą wybiera się najlepiej rokującą część populacji.
Wybieranie uczestników turnieju polega na losowaniu z populacji zadanej liczby uczestników turnieju oraz następnie wybranie spośród tych osobników najlepszego pod względem długości trasy, czyli wygrywa osobnik, którego trasa jest najkrótsza.
Zostaje on dodany do grupy osobników, która będzie się krzyżować i stworzy nową populację.

Krzyżowanie, zawarte w funkcji **'crossing'** polega w skrócie na wymianie podtras pomiędzy osobnikami.
Najłatwiej jest to wytłumaczyć na przykładzie.

Wylosowaliśmy indeksy krzyżowania:
index1 = 2
index2 = 6

oraz dwóch rodziców:

parent1:
['E', 'D', 'F', 'G', 'I', 'H', 'A', 'J', 'L', 'C', 'K', 'B']
parent2:
['D', 'F', 'G', 'I', 'A', 'J', 'H', 'C', 'B', 'K', 'L', 'E']

Zaznaczone wierzchołki to te obejmowane przez indeksy, które zostaną wstawione do dzieci na taki sam zakres indeksów, z pierwszego rodzica odpowiednio do pierwszego dziecka, z drugiego rodzica do drugiego dziecka, jak poniżej

child1 - parent1:
[None, None, 'F', 'G', 'I', 'H', 'A', None, None, None, None, None]
child2 - parent2:
[None, None, 'G', 'I', 'A', 'J', 'H', None, None, None, None, None]

Później wystarczy uzupełnić dalszą część trasy, weźmy za przykład child1.
Jest to krzyżowanie między rodzicem pierwszym i drugim, z pierwszego rodzica wzięliśmy podtrasę ['F', 'G', 'I', 'H', 'A'], więc resztę musimy uzupełnić z rodzica numer 2.
Przypomnijmy jego trasę:

parent2:
['D', 'F', 'G', 'I', 'A', 'J', 'H', 'C', 'B', 'K', 'L', 'E']

Zaznaczmy w nim na zielono wierzchołki, które już posiada child1, a na czerwono te, których w child1 jeszcze nie ma:

parent2:
['D', 'F', 'G', 'I', 'A', 'J', 'H', 'C', 'B', 'K', 'L', 'E']

Teraz wpisujemy po kolei w miejsca 'None' dziecka wierzchołki, których jeszcze nie posiada.

child1 - parent2:
['D', 'J', 'F', 'G', 'I', 'H', 'A', 'C', 'B', 'K', 'L', 'E']

Dla drugiego dziecka postępujemy analogicznie i otrzymujemy:

child2 - parent1:

['E', 'D', 'G', 'I', 'A', 'J', 'H', 'F', 'L', 'C', 'K', 'B']

Analogiczna operacja jest przeprowadzana, aby utworzyć child3, które jest skrzyżowaniem parent1 i parent3. Następnie bierzemy z populacji kolejnych 3 rodziców i znów przeprowadzamy między nimi operację krzyżowania.
Operacja jest przeprowadzana, dopóki nie zostanie stworzona populacja o zadanej na początku liczebności.

Funkcja **'mutation'** polega na wylosowaniu z populacji osobnika i zmutowaniu go. Powtarza się to dla określonego wcześniej procenta populacji, nie może być on zbyt wielki, aby nie zaburzyć wyników krzyżowania, czyli dla populacji o wielkości 100 osobników, jeśli współczynnik mutacji będzie wynosił np. 0,05 to zostanie zmutowanych 5 osobników. W szczególności nie jest wskazane, aby współczynnik mutacji był większy od współczynnika krzyżowania.
Ma on tylko delikatnie wpływać na populację, aby dopuścić do niej gatunki, z których być może powstaną nowe, lepsze rozwiązania, a zostały pominięte w turniejach i krzyżowaniu.

Operacja mutacji polega u mnie na wymianie losowych dwóch wierzchołków w osobniku. Faktycznie będzie to wymiana dwóch miast w trasie.

*** shortest_hamilton_cycle** – moduł szukający najkrótszego cyklu hamiltonowskiego w grafie

Zawiera funkcję 'number_one_from_population', która wyznacza najlepszego osobnika z populacji oraz funkcję 'shortest_hamilton_cycle', która uruchamia wszystkie algorytmy genetyczne.

Poprzez zmianę wartości zmiennych o nazwach jak poniżej, możemy ustawić parametry algorytmów genetycznych.

population_size = 100
iterations = 15
tournament_participants_number = 5
crossing_rate = 0.4
mutation_rate = 0.05

Na początku funkcja losuje pewną startową populację cykli Hamiltona (populacja jest po prostu listą tych cykli), w każdym cyklu wierzchołki są rozmieszczone całkowicie losowo, imituje to startową populację możliwych tras komiwojażera, gdzie w każdej trasie miasta są rozmieszczone losowo.

Dla ustalonej przez nas w parametrze 'iterations' liczby iteracji następują operacje turnieje+krzyżowanie+mutacja, które powodują utworzenie nowej populacji na miejsce starej, dokładnie tyle razy, ile iteracji będzie posiadała pętla.
Po wykonaniu tych operacji populacja ulega wyraźnej zmianie, ewoluuje i zmienia się,

dlatego po każdej iteracji wyniki zostają wypisane na ekran oraz zapisane do pliku.

*** nearest_neighbour_algorithm.py – moduł z implementacją algorytmu najbliższych sąsiadów**
Algorytm polega na wylosowaniu miasta startowego i szukaniu miasta, które leży najbliżej, następnie dodaje je do wierzchołków odwiedzonych.
Dla znalezionego miasta powtarzamy procedurę poszukiwania leżącego najbliżej miasta, pomijając te, które już znajdują się w odwiedzonych.
Kontynuujemy aż do momentu, gdy wszystkie miasta zostaną odwiedzone.

*** W plikach population1, population2 itp. znajdują się wypisane przez program populacje wraz z numerem iteracji po której zostały utworzone.**
Dzięki temu można porównać zmianę, która się dokonywała.

Przykładowo, dla 25 miast oraz 15 iteracji-pokoleń, przy liczebności populacji - 100 osobników, mamy przykładowy wycinek populacji:

population1.txt (1. pokolenie - losowe)

```
['K', 'U', 'C', 'Y', 'N', 'D', 'V', 'S', 'W', 'F', 'B', 'E', 'I', 'X', 'O', 'M', 'L', 'H', 'A', 'J', 'Q', 'P', 'T', 'G', 'R'] 1118
['T', 'W', 'G', 'B', 'O', 'L', 'I', 'K', 'E', 'S', 'A', 'H', 'M', 'P', 'U', 'R', 'Y', 'N', 'F', 'D', 'X', 'V', 'J', 'Q', 'C'] 1040
['A', 'P', 'K', 'S', 'L', 'W', 'B', 'H', 'T', 'E', 'M', 'D', 'O', 'G', 'U', 'Q', 'V', 'C', 'X', 'N', 'I', 'R', 'Y', 'J', 'F'] 1105
['E', 'X', 'L', 'C', 'W', 'P', 'T', 'V', 'S', 'F', 'Y', 'Q', 'G', 'M', 'U', 'N', 'H', 'B', 'I', 'A', 'O', 'R', 'K', 'D', 'J'] 1103
['U', 'H', 'G', 'O', 'I', 'N', 'K', 'S', 'V', 'A', 'C', 'T', 'B', 'J', 'W', 'P', 'D', 'Y', 'F', 'L', 'X', 'E', 'M', 'R', 'Q'] 1005
['L', 'G', 'B', 'S', 'A', 'I', 'X', 'P', 'V', 'F', 'U', 'Y', 'N', 'D', 'T', 'H', 'O', 'M', 'Q', 'E', 'R', 'K', 'C', 'W', 'J'] 1165
['D', 'O', 'U', 'P', 'J', 'X', 'H', 'L', 'V', 'A', 'M', 'F', 'N', 'S', 'C', 'T', 'W', 'G', 'Q', 'K', 'R', 'B', 'I', 'Y', 'E'] 1093
['T', 'Q', 'M', 'B', 'X', 'O', 'H', 'L', 'Y', 'V', 'G', 'F', 'U', 'E', 'R', 'C', 'K', 'I', 'N', 'J', 'D', 'A', 'P', 'W', 'S'] 1146
['O', 'F', 'A', 'X', 'G', 'S', 'L', 'R', 'K', 'Q', 'E', 'T', 'J', 'P', 'N', 'H', 'I', 'V', 'Y', 'U', 'C', 'W', 'D', 'B', 'M'] 1168
['H', 'Y', 'R', 'J', 'K', 'D', 'A', 'N', 'V', 'G', 'M', 'C', 'T', 'F', 'X', 'L', 'S', 'P', 'B', 'O', 'Q', 'U', 'I', 'E', 'W'] 1126
['E', 'X', 'I', 'F', 'U', 'C', 'K', 'Y', 'D', 'J', 'A', 'O', 'S', 'V', 'W', 'T', 'Q', 'B', 'P', 'L', 'N', 'R', 'H', 'G', 'M'] 1173
['C', 'Y', 'U', 'R', 'S', 'T', 'I', 'W', 'D', 'B', 'Q', 'L', 'H', 'F', 'O', 'P', 'X', 'N', 'G', 'K', 'V', 'A', 'J', 'M', 'E'] 1059
['P', 'J', 'E', 'C', 'Q', 'O', 'D', 'B', 'W', 'M', 'Y', 'S', 'H', 'N', 'L', 'F', 'R', 'A', 'K', 'X', 'V', 'T', 'G', 'U', 'I'] 1065
['E', 'X', 'H', 'W', 'J', 'Q', 'K', 'O', 'Y', 'R', 'C', 'U', 'L', 'T', 'F', 'P', 'V', 'D', 'M', 'B', 'S', 'N', 'A', 'I', 'G'] 1099
['T', 'D', 'G', 'S', 'C', 'L', 'F', 'H', 'U', 'A', 'R', 'V', 'X', 'Q', 'I', 'W', 'N', 'P', 'B', 'J', 'Y', 'K', 'E', 'M', 'O'] 1161
['B', 'G', 'R', 'C', 'Q', 'A', 'N', 'D', 'W', 'H', 'I', 'E', 'P', 'T', 'J', 'L', 'K', 'F', 'V', 'X', 'S', 'M', 'Y', 'U', 'O'] 1200
['U', 'M', 'T', 'F', 'O', 'G', 'V', 'J', 'R', 'K', 'X', 'L', 'E', 'H', 'D', 'C', 'A', 'Q', 'Y', 'N', 'P', 'W', 'I', 'S', 'B'] 900
['O', 'L', 'P', 'N', 'T', 'Q', 'M', 'C', 'F', 'E', 'X', 'K', 'H', 'B', 'S', 'I', 'D', 'R', 'A', 'Y', 'U', 'V', 'J', 'W', 'G'] 895
['W', 'E', 'O', 'K', 'B', 'I', 'U', 'G', 'R', 'A', 'L', 'Y', 'H', 'N', 'D', 'M', 'X', 'C', 'V', 'S', 'F', 'P', 'J', 'Q', 'T'] 1265
['R', 'F', 'C', 'L', 'Y', 'T', 'M', 'K', 'N', 'B', 'D', 'Q', 'O', 'P', 'X', 'V', 'W', 'J', 'S', 'G', 'E', 'U', 'H', 'I', 'A'] 1169
['U', 'C', 'T', 'J', 'G', 'L', 'M', 'P', 'B', 'A', 'W', 'S', 'D', 'R', 'X', 'I', 'N', 'H', 'V', 'Y', 'F', 'Q', 'O', 'E', 'K'] 1176
['K', 'U', 'C', 'S', 'W', 'D', 'R', 'M', 'I', 'H', 'O', 'P', 'J', 'L', 'Q', 'V', 'Y', 'G', 'N', 'A', 'F', 'T', 'X', 'B', 'E'] 1236
['D', 'C', 'I', 'J', 'B', 'G', 'H', 'N', 'M', 'R', 'Q', 'T', 'S', 'K', 'F', 'V', 'E', 'U', 'Y', 'A', 'P', 'L', 'X', 'O', 'W'] 1177
```

5 pokoleń dalej zauważamy już znaczną różnicę:

population6.txt

```
['O', 'L', 'Q', 'C', 'X', 'B', 'W', 'F', 'R', 'E', 'T', 'V', 'M', 'N', 'K', 'H', 'P', 'A', 'J', 'S', 'Y', 'I', 'D', 'U', 'G'] 778
['O', 'L', 'B', 'G', 'M', 'T', 'K', 'E', 'Q', 'C', 'H', 'S', 'I', 'N', 'X', 'J', 'W', 'P', 'D', 'Y', 'F', 'R', 'A', 'U', 'V'] 851
['O', 'L', 'P', 'H', 'X', 'F', 'Y', 'N', 'I', 'S', 'V', 'A', 'C', 'B', 'T', 'J', 'W', 'U', 'D', 'G', 'Q', 'K', 'R', 'M', 'E'] 840
['O', 'L', 'Q', 'C', 'X', 'B', 'W', 'F', 'R', 'E', 'T', 'V', 'M', 'N', 'G', 'P', 'K', 'H', 'S', 'I', 'D', 'A', 'Y', 'U', 'J'] 782
['O', 'L', 'B', 'G', 'P', 'Q', 'M', 'C', 'K', 'H', 'S', 'I', 'N', 'T', 'X', 'E', 'J', 'W', 'D', 'Y', 'F', 'R', 'A', 'U', 'V'] 841
['U', 'Y', 'I', 'D', 'L', 'H', 'F', 'O', 'P', 'X', 'V', 'A', 'N', 'S', 'C', 'T', 'W', 'G', 'Q', 'K', 'R', 'B', 'J', 'M', 'E'] 838
['M', 'U', 'I', 'D', 'L', 'H', 'F', 'O', 'P', 'X', 'V', 'A', 'N', 'S', 'C', 'T', 'W', 'G', 'R', 'B', 'K', 'E', 'J', 'Y', 'Q'] 866
['O', 'L', 'Q', 'C', 'X', 'B', 'W', 'F', 'R', 'T', 'V', 'N', 'K', 'H', 'P', 'A', 'S', 'Y', 'I', 'D', 'U', 'G', 'J', 'M', 'E'] 832
['Y', 'U', 'I', 'D', 'L', 'H', 'F', 'O', 'P', 'X', 'V', 'A', 'N', 'S', 'C', 'T', 'W', 'G', 'R', 'B', 'Q', 'K', 'J', 'M', 'E'] 741
['Y', 'J', 'B', 'M', 'T', 'K', 'E', 'U', 'I', 'D', 'L', 'H', 'F', 'O', 'P', 'X', 'V', 'A', 'N', 'S', 'C', 'W', 'G', 'R', 'Q'] 850
['D', 'U', 'I', 'Y', 'L', 'F', 'O', 'P', 'H', 'X', 'N', 'S', 'V', 'A', 'C', 'B', 'T', 'J', 'W', 'G', 'Q', 'K', 'R', 'M', 'E'] 861
['U', 'I', 'H', 'O', 'A', 'N', 'S', 'R', 'P', 'D', 'Y', 'F', 'L', 'C', 'T', 'V', 'W', 'G', 'Q', 'K', 'X', 'B', 'J', 'M', 'E'] 876
['U', 'M', 'T', 'F', 'O', 'G', 'V', 'J', 'R', 'K', 'X', 'L', 'E', 'H', 'S', 'C', 'A', 'Q', 'Y', 'N', 'P', 'W', 'I', 'D', 'B'] 900
['K', 'A', 'H', 'P', 'O', 'F', 'Q', 'L', 'W', 'G', 'M', 'Y', 'N', 'C', 'J', 'I', 'R', 'U', 'V', 'D', 'S', 'T', 'B', 'X', 'E'] 917
['U', 'I', 'O', 'V', 'C', 'T', 'G', 'Q', 'R', 'W', 'D', 'Y', 'F', 'N', 'K', 'H', 'P', 'A', 'J', 'S', 'L', 'X', 'B', 'M', 'E'] 905
['Y', 'U', 'I', 'D', 'L', 'O', 'W', 'F', 'R', 'E', 'T', 'V', 'H', 'P', 'X', 'A', 'N', 'S', 'C', 'G', 'Q', 'K', 'B', 'J', 'M'] 854
['C', 'J', 'B', 'G', 'M', 'T', 'K', 'E', 'U', 'A', 'W', 'S', 'D', 'R', 'X', 'I', 'N', 'H', 'V', 'Y', 'Q', 'O', 'F', 'P', 'L'] 865
['Y', 'P', 'L', 'H', 'F', 'O', 'X', 'U', 'I', 'D', 'V', 'A', 'N', 'S', 'C', 'T', 'W', 'G', 'Q', 'K', 'R', 'B', 'J', 'M', 'E'] 893
```

Spójrzmy na dalsze iteracje:

population13.txt

```
['L', 'Q', 'C', 'B', 'E', 'O', 'W', 'F', 'R', 'T', 'V', 'H', 'P', 'X', 'M', 'K', 'J', 'A', 'N', 'S', 'Y', 'U', 'I', 'D', 'G'] 795
['O', 'L', 'Q', 'X', 'W', 'F', 'R', 'E', 'M', 'N', 'K', 'H', 'P', 'J', 'V', 'A', 'C', 'B', 'T', 'S', 'Y', 'U', 'I', 'D', 'G'] 675
['Y', 'I', 'D', 'U', 'L', 'O', 'F', 'P', 'H', 'K', 'E', 'X', 'N', 'S', 'V', 'A', 'B', 'T', 'J', 'Q', 'C', 'W', 'G', 'R', 'M'] 675
['O', 'L', 'Q', 'X', 'B', 'F', 'R', 'E', 'T', 'V', 'M', 'H', 'P', 'N', 'S', 'C', 'G', 'U', 'A', 'W', 'Y', 'I', 'D', 'J', 'K'] 736
['O', 'L', 'Q', 'C', 'B', 'W', 'F', 'E', 'R', 'T', 'V', 'H', 'P', 'X', 'A', 'N', 'M', 'K', 'J', 'S', 'Y', 'U', 'I', 'D', 'G'] 715
['O', 'U', 'I', 'D', 'L', 'Q', 'M', 'K', 'N', 'W', 'F', 'R', 'E', 'V', 'T', 'S', 'C', 'G', 'A', 'Y', 'H', 'P', 'X', 'B', 'J'] 747
['O', 'L', 'Q', 'M', 'K', 'E', 'U', 'X', 'R', 'T', 'V', 'H', 'P', 'W', 'A', 'N', 'S', 'C', 'B', 'F', 'J', 'Y', 'I', 'D', 'G'] 685
['I', 'Y', 'H', 'D', 'U', 'L', 'F', 'R', 'O', 'P', 'X', 'B', 'T', 'J', 'Q', 'K', 'M', 'V', 'A', 'N', 'S', 'C', 'W', 'G', 'E'] 663
['O', 'L', 'Q', 'M', 'K', 'E', 'U', 'A', 'W', 'C', 'X', 'B', 'F', 'R', 'V', 'N', 'T', 'J', 'H', 'P', 'S', 'Y', 'I', 'D', 'G'] 738
['Y', 'U', 'I', 'D', 'L', 'H', 'F', 'O', 'P', 'X', 'V', 'A', 'N', 'S', 'C', 'T', 'W', 'G', 'R', 'B', 'Q', 'K', 'J', 'M', 'E'] 741
['Y', 'U', 'I', 'D', 'L', 'N', 'F', 'O', 'R', 'H', 'S', 'V', 'A', 'C', 'B', 'T', 'J', 'W', 'G', 'Q', 'P', 'X', 'K', 'E', 'M'] 828
['Y', 'U', 'I', 'D', 'T', 'L', 'A', 'O', 'F', 'X', 'B', 'P', 'V', 'H', 'N', 'S', 'C', 'W', 'G', 'R', 'Q', 'K', 'J', 'M', 'E'] 756
['L', 'Q', 'C', 'B', 'E', 'O', 'W', 'F', 'R', 'T', 'V', 'H', 'P', 'X', 'A', 'N', 'M', 'K', 'J', 'S', 'Y', 'U', 'I', 'D', 'G'] 646
['Y', 'I', 'D', 'U', 'L', 'N', 'S', 'C', 'R', 'T', 'J', 'H', 'F', 'O', 'P', 'X', 'V', 'A', 'W', 'B', 'G', 'Q', 'K', 'M', 'E'] 695
['Y', 'I', 'H', 'D', 'K', 'E', 'U', 'L', 'F', 'O', 'P', 'X', 'N', 'S', 'V', 'A', 'C', 'B', 'T', 'J', 'W', 'G', 'Q', 'R', 'M'] 747
['O', 'L', 'Q', 'M', 'T', 'K', 'E', 'U', 'A', 'C', 'X', 'B', 'F', 'R', 'V', 'N', 'W', 'G', 'H', 'P', 'J', 'S', 'Y', 'I', 'D'] 675
['Y', 'I', 'H', 'D', 'U', 'L', 'F', 'O', 'P', 'X', 'B', 'T', 'J', 'Q', 'S', 'M', 'V', 'A', 'N', 'K', 'C', 'W', 'G', 'R', 'E'] 636
['Y', 'L', 'E', 'M', 'I', 'H', 'D', 'K', 'Q', 'U', 'F', 'O', 'P', 'X', 'N', 'S', 'V', 'A', 'B', 'T', 'J', 'C', 'W', 'G', 'R'] 681
['Y', 'I', 'H', 'D', 'U', 'L', 'F', 'O', 'P', 'X', 'B', 'T', 'J', 'S', 'M', 'V', 'A', 'N', 'K', 'C', 'W', 'G', 'Q', 'R', 'E'] 658
['Q', 'C', 'R', 'E', 'L', 'O', 'F', 'H', 'P', 'X', 'B', 'T', 'V', 'M', 'N', 'K', 'A', 'Y', 'W', 'G', 'J', 'S', 'U', 'I', 'D'] 835
['Y', 'U', 'I', 'D', 'R', 'L', 'F', 'N', 'S', 'V', 'A', 'C', 'B', 'T', 'J', 'W', 'G', 'Q', 'O', 'H', 'P', 'X', 'K', 'E', 'M'] 743
['Y', 'I', 'D', 'U', 'L', 'F', 'O', 'P', 'X', 'V', 'A', 'B', 'T', 'H', 'E', 'S', 'C', 'W', 'J', 'M', 'N', 'Q', 'G', 'R', 'K'] 773
['Y', 'I', 'D', 'O', 'U', 'L', 'F', 'P', 'X', 'B', 'T', 'J', 'S', 'M', 'V', 'A', 'N', 'Q', 'H', 'E', 'C', 'W', 'G', 'R', 'K'] 704
['Y', 'I', 'D', 'U', 'T', 'L', 'A', 'F', 'X', 'O', 'P', 'V', 'H', 'N', 'S', 'C', 'W', 'G', 'B', 'E', 'R', 'J', 'Q', 'K', 'M'] 929
```

population20.txt

```
['Y', 'I', 'D', 'U', 'L', 'F', 'O', 'P', 'X', 'V', 'A', 'B', 'T', 'H', 'E', 'S', 'C', 'W', 'G', 'R', 'J', 'Q', 'K', 'M', 'N'] 649
['L', 'Q', 'C', 'W', 'F', 'R', 'E', 'O', 'P', 'X', 'B', 'T', 'J', 'S', 'Y', 'U', 'D', 'M', 'V', 'H', 'N', 'A', 'G', 'I', 'K'] 654
['L', 'Q', 'C', 'B', 'E', 'O', 'W', 'F', 'R', 'T', 'V', 'H', 'P', 'X', 'A', 'N', 'M', 'K', 'J', 'S', 'Y', 'U', 'I', 'D', 'G'] 646
['Y', 'U', 'A', 'F', 'I', 'D', 'M', 'T', 'L', 'O', 'P', 'X', 'V', 'H', 'N', 'S', 'C', 'W', 'G', 'R', 'B', 'Q', 'K', 'J', 'E'] 700
['K', 'E', 'U', 'I', 'D', 'M', 'T', 'L', 'O', 'F', 'P', 'X', 'V', 'H', 'N', 'S', 'R', 'C', 'W', 'A', 'G', 'B', 'J', 'Q', 'Y'] 647
['Y', 'I', 'H', 'D', 'U', 'L', 'F', 'O', 'P', 'X', 'B', 'T', 'J', 'S', 'V', 'A', 'N', 'M', 'E', 'Q', 'C', 'W', 'G', 'R', 'K'] 649
['Y', 'U', 'F', 'O', 'B', 'W', 'G', 'R', 'E', 'T', 'V', 'H', 'P', 'X', 'A', 'N', 'M', 'K', 'J', 'S', 'I', 'D', 'L', 'C', 'Q'] 725
['Y', 'I', 'D', 'U', 'L', 'T', 'J', 'F', 'O', 'P', 'X', 'V', 'A', 'H', 'N', 'S', 'C', 'W', 'G', 'R', 'B', 'Q', 'K', 'M', 'E'] 598
['B', 'G', 'E', 'U', 'Y', 'W', 'L', 'O', 'H', 'F', 'R', 'T', 'I', 'D', 'M', 'P', 'X', 'V', 'N', 'S', 'A', 'K', 'C', 'J', 'Q'] 677
['I', 'Y', 'H', 'D', 'U', 'L', 'F', 'P', 'X', 'B', 'T', 'J', 'S', 'M', 'V', 'A', 'N', 'Q', 'O', 'C', 'W', 'G', 'R', 'K', 'E'] 564
['L', 'Q', 'C', 'W', 'F', 'R', 'E', 'O', 'P', 'X', 'B', 'T', 'J', 'S', 'Y', 'U', 'I', 'D', 'M', 'V', 'H', 'N', 'A', 'G', 'K'] 564
['Y', 'I', 'D', 'U', 'L', 'T', 'J', 'K', 'O', 'H', 'B', 'Q', 'P', 'X', 'A', 'N', 'S', 'M', 'V', 'F', 'C', 'W', 'G', 'R', 'E'] 579
['Y', 'I', 'W', 'D', 'U', 'L', 'F', 'O', 'P', 'N', 'K', 'X', 'B', 'T', 'J', 'Q', 'S', 'M', 'V', 'A', 'C', 'H', 'G', 'R', 'E'] 601
['L', 'K', 'E', 'U', 'R', 'H', 'F', 'O', 'P', 'X', 'V', 'S', 'J', 'Q', 'M', 'T', 'B', 'A', 'N', 'C', 'W', 'G', 'Y', 'I', 'D'] 678
['Y', 'I', 'D', 'U', 'L', 'F', 'O', 'P', 'X', 'V', 'A', 'B', 'T', 'H', 'N', 'K', 'J', 'Q', 'S', 'M', 'C', 'W', 'G', 'R', 'E'] 609
['O', 'L', 'Q', 'X', 'W', 'F', 'R', 'T', 'V', 'K', 'H', 'B', 'Y', 'U', 'A', 'N', 'S', 'C', 'G', 'P', 'E', 'M', 'J', 'I', 'D'] 620
['Y', 'U', 'I', 'D', 'M', 'T', 'L', 'O', 'H', 'F', 'P', 'X', 'V', 'S', 'J', 'Q', 'B', 'A', 'N', 'C', 'W', 'G', 'R', 'K', 'E'] 595
['Y', 'I', 'D', 'U', 'L', 'T', 'J', 'F', 'O', 'P', 'X', 'V', 'H', 'S', 'M', 'Q', 'B', 'A', 'N', 'C', 'W', 'K', 'G', 'R', 'E'] 580
['Y', 'I', 'D', 'U', 'L', 'T', 'J', 'F', 'O', 'P', 'X', 'V', 'A', 'H', 'N', 'M', 'S', 'C', 'W', 'G', 'B', 'Q', 'R', 'K', 'E'] 638
['Y', 'I', 'D', 'U', 'L', 'H', 'F', 'P', 'X', 'B', 'T', 'J', 'S', 'M', 'V', 'A', 'N', 'Q', 'O', 'C', 'E', 'W', 'G', 'R', 'K'] 625
['R', 'I', 'D', 'U', 'M', 'T', 'L', 'O', 'F', 'P', 'X', 'V', 'H', 'N', 'S', 'C', 'W', 'A', 'G', 'B', 'J', 'Q', 'Y', 'K', 'E'] 608
['O', 'B', 'G', 'T', 'K', 'E', 'U', 'A', 'W', 'F', 'P', 'X', 'V', 'H', 'Y', 'I', 'D', 'L', 'N', 'S', 'C', 'J', 'Q', 'R', 'M'] 586
['Y', 'I', 'D', 'U', 'B', 'E', 'M', 'T', 'L', 'O', 'F', 'P', 'X', 'V', 'H', 'N', 'S', 'C', 'W', 'A', 'G', 'R', 'J', 'Q', 'K'] 550
['Y', 'I', 'D', 'U', 'L', 'T', 'J', 'F', 'O', 'P', 'X', 'V', 'A', 'H', 'N', 'S', 'B', 'M', 'Q', 'C', 'W', 'G', 'R', 'K', 'E'] 644
['Y', 'I', 'D', 'U', 'L', 'T', 'J', 'F', 'O', 'P', 'X', 'V', 'A', 'H', 'M', 'S', 'Q', 'B', 'N', 'C', 'W', 'K', 'G', 'R', 'E'] 659
```

population30.txt (populacja, na której kończymy)

```
['Y', 'I', 'D', 'C', 'K', 'E', 'U', 'W', 'F', 'P', 'X', 'V', 'A', 'L', 'O', 'B', 'H', 'N', 'J', 'Q', 'S', 'M', 'T', 'G', 'R'] 455
['I', 'D', 'B', 'E', 'M', 'T', 'V', 'F', 'H', 'L', 'N', 'J', 'S', 'O', 'Y', 'U', 'P', 'X', 'C', 'W', 'A', 'G', 'R', 'Q', 'K'] 570
['T', 'K', 'U', 'A', 'W', 'P', 'X', 'V', 'F', 'Y', 'I', 'D', 'L', 'N', 'O', 'B', 'H', 'S', 'M', 'J', 'Q', 'C', 'G', 'R', 'E'] 588
['Y', 'I', 'D', 'U', 'B', 'E', 'M', 'T', 'L', 'X', 'V', 'A', 'O', 'F', 'P', 'H', 'N', 'S', 'C', 'W', 'G', 'R', 'J', 'Q', 'K'] 555
['Y', 'I', 'D', 'U', 'L', 'H', 'F', 'O', 'P', 'X', 'V', 'A', 'M', 'T', 'S', 'J', 'Q', 'B', 'N', 'C', 'W', 'G', 'R', 'K', 'E'] 578
['O', 'B', 'T', 'K', 'U', 'A', 'W', 'P', 'X', 'V', 'F', 'C', 'Y', 'I', 'D', 'L', 'N', 'S', 'H', 'G', 'E', 'M', 'J', 'Q', 'R'] 597
['B', 'T', 'U', 'K', 'F', 'A', 'W', 'P', 'X', 'V', 'H', 'L', 'N', 'J', 'S', 'O', 'Y', 'I', 'D', 'C', 'G', 'E', 'M', 'Q', 'R'] 529
['Y', 'I', 'D', 'U', 'L', 'C', 'K', 'E', 'W', 'F', 'P', 'X', 'V', 'A', 'O', 'B', 'H', 'N', 'J', 'Q', 'S', 'M', 'T', 'G', 'R'] 593
['Y', 'I', 'D', 'C', 'K', 'E', 'U', 'W', 'F', 'P', 'X', 'V', 'A', 'L', 'O', 'B', 'H', 'N', 'J', 'Q', 'S', 'M', 'T', 'G', 'R'] 533
['O', 'L', 'Q', 'W', 'F', 'R', 'T', 'K', 'P', 'X', 'V', 'H', 'S', 'B', 'Y', 'U', 'A', 'N', 'C', 'G', 'E', 'M', 'J', 'I', 'D'] 583
['Y', 'I', 'D', 'U', 'B', 'E', 'M', 'T', 'L', 'O', 'F', 'P', 'X', 'V', 'H', 'N', 'S', 'C', 'W', 'A', 'G', 'R', 'J', 'Q', 'K'] 550
['Y', 'I', 'D', 'U', 'L', 'E', 'M', 'T', 'F', 'O', 'P', 'X', 'V', 'A', 'B', 'H', 'N', 'Q', 'K', 'J', 'S', 'C', 'W', 'G', 'R'] 536
['Y', 'I', 'D', 'U', 'L', 'F', 'O', 'P', 'X', 'V', 'A', 'M', 'T', 'B', 'H', 'N', 'J', 'Q', 'K', 'S', 'C', 'W', 'G', 'R', 'E'] 561
['O', 'B', 'T', 'K', 'U', 'A', 'W', 'P', 'H', 'L', 'N', 'J', 'S', 'X', 'V', 'F', 'Y', 'I', 'D', 'M', 'Q', 'C', 'G', 'R', 'E'] 629
['Y', 'I', 'D', 'U', 'B', 'E', 'M', 'T', 'L', 'O', 'F', 'P', 'X', 'V', 'H', 'N', 'S', 'C', 'W', 'A', 'J', 'Q', 'G', 'R', 'K'] 596
['Y', 'I', 'D', 'U', 'L', 'F', 'O', 'P', 'X', 'V', 'A', 'B', 'T', 'H', 'N', 'K', 'J', 'Q', 'S', 'M', 'C', 'W', 'G', 'R', 'E'] 609
['T', 'E', 'A', 'W', 'Y', 'I', 'D', 'K', 'U', 'F', 'P', 'X', 'V', 'L', 'O', 'B', 'H', 'N', 'J', 'Q', 'S', 'M', 'C', 'G', 'R'] 624
['T', 'U', 'A', 'W', 'P', 'X', 'V', 'F', 'Y', 'I', 'D', 'L', 'N', 'O', 'B', 'H', 'S', 'M', 'J', 'Q', 'C', 'G', 'R', 'K', 'E'] 604
['Y', 'I', 'D', 'U', 'L', 'F', 'O', 'P', 'X', 'V', 'A', 'M', 'T', 'H', 'S', 'J', 'Q', 'B', 'N', 'C', 'W', 'G', 'R', 'K', 'E'] 615
['O', 'B', 'T', 'K', 'U', 'A', 'W', 'P', 'X', 'V', 'F', 'H', 'Y', 'I', 'D', 'L', 'N', 'S', 'M', 'J', 'Q', 'C', 'G', 'R', 'E'] 595
['O', 'L', 'W', 'C', 'R', 'P', 'X', 'V', 'Y', 'I', 'D', 'F', 'K', 'E', 'U', 'A', 'B', 'H', 'N', 'M', 'J', 'Q', 'S', 'G', 'T'] 479
```

Jak widać, gwałtowne zmiany w pokoleniach dokonują się głównie w pierwszych iteracjach, od pewnego momentu zmiany mogą stawać się nieznaczne.
Zależy to również w jakim zakresie znajdują się wagi, różne wyniki otrzymamy, gdy odległości są z przedziałów (0; 50), a całkiem inne będzie porównanie dla wag (0; 300).

Po przejściu wszystkich iteracji program wybiera najlepszego osobnika z ostatniej populacji, którego możemy uznać, że najlepsze rozwiązanie znalezione w rozsądnym czasie. Gdybyśmy nie ustalili liczby iteracji pętla działałaby w nieskończoność, tworząc kolejne populacje, które, zakładamy, stawałyby się coraz lepsze.

WYNIKI PROGRAMU

Obserwując wyniki programu można powiedzieć, że faktycznie wraz ze wzrostem liczby iteracji osobniki stają się coraz lepsze, trasy są krótsze.
Musimy mieć na uwadze, że czasami ewolucja może pójść w złą stronę i w wyniku krzyżowania czy mutacji zmodyfikować całkiem dobre rozwiązania, które były nawet lepsze niż ostateczny wynik, ale jest to ryzyko, które możemy śmiało podjąć, korzystając z algorytmów genetycznych, gdyż wynik, który otrzymamy prawdopodobnie i tak będzie wystarczająco zadowalający.

PORÓWNANIE Z ALGORYTMEM NAJBLIŻSZYCH SĄSIADÓW

Manipulując ilością miast możemy porównać wyniki zwracane przez algorytmy genetyczne i algorytm najbliższych sąsiadów.

Np. dla 10 miast i tego samego grafu, otrzymujemy wyniki jak poniżej:

```
GENETIC ALGORITHMS:
The best cycle is ['J', 'F', 'C', 'B', 'I', 'A', 'G', 'H', 'D', 'E']  with length  164.

NEAREST NEIGHBOUR ALGORITHM:
['H', 'B', 'I', 'E', 'F', 'C', 'J', 'G', 'D', 'A'] , length:  197
```

```
GENETIC ALGORITHMS:
The best cycle is ['D', 'A', 'H', 'G', 'E', 'J', 'F', 'C', 'B', 'I']  with length  143.

NEAREST NEIGHBOUR ALGORITHM:
['I', 'B', 'H', 'F', 'C', 'J', 'G', 'E', 'D', 'A'] , length:  180
```

Dla 15 miast:

```
GENETIC ALGORITHMS:
The best cycle is ['H', 'A', 'E', 'F', 'M', 'J', 'I', 'D', 'G', 'B', 'L', 'K', 'O', 'N', 'C']  with length  211.

NEAREST NEIGHBOUR ALGORITHM:
['L', 'G', 'B', 'C', 'H', 'N', 'A', 'E', 'F', 'O', 'K', 'D', 'I', 'J', 'M'] , length:  259
```

```
GENETIC ALGORITHMS:
The best cycle is ['I', 'J', 'E', 'F', 'C', 'H', 'K', 'D', 'N', 'A', 'M', 'O', 'L', 'G', 'B']  with length  184.

NEAREST NEIGHBOUR ALGORITHM:
['F', 'C', 'H', 'N', 'L', 'G', 'B', 'E', 'I', 'J', 'M', 'K', 'A', 'D', 'O'] , length:  250
```

```
GENETIC ALGORITHMS:
The best cycle is ['M', 'I', 'J', 'E', 'F', 'C', 'H', 'N', 'A', 'D', 'O', 'L', 'G', 'K', 'B']  with length  173.

NEAREST NEIGHBOUR ALGORITHM:
['H', 'N', 'L', 'G', 'B', 'C', 'E', 'F', 'O', 'A', 'M', 'I', 'J', 'D', 'K'] , length:  274
```

Jak widać tutaj algorytmy genetyczne okazują się w większości przypadków lepsze, choć może wystąpić przypadek, gdy algorytm najbliższego sąsiada zadziała lepiej.

Dla 30 miast wyniki już znacząco się różnią:

GENETIC ALGORITHMS:
The best cycle is ['K', 'Q', '[', 'G', ']', 'T', 'H', 'R', 'Y', 'A', 'B', 'P', 'C', 'W', '\\', 'Z', 'J', 'O', 'F', 'I', 'V', 'D', 'L', 'U', 'N', 'X', 'S', '^', 'E', 'M'] with length 474.

NEAREST NEIGHBOUR ALGORITHM:
['S', 'T', 'U', 'N', 'X', 'R', 'I', 'Z', 'J', 'D', '^', 'Q', 'H', ']', 'V', 'G', '[', 'B', 'W', 'K', '\\', 'C', 'E', 'M', 'P', 'A', 'O', 'F', 'Y', 'L'] , length: 321

GENETIC ALGORITHMS:
The best cycle is ['L', 'I', 'D', 'M', 'U', 'F', 'C', 'N', 'S', 'W', 'B', 'Q', 'H', ']', 'V', 'G', 'Y', 'J', 'X', 'A', 'O', 'R', '\\', 'Z', 'K', 'T', '^', 'E', '[', 'P'] with length 514.

NEAREST NEIGHBOUR ALGORITHM:
['H', 'T', 'U', 'S', 'X', 'R', 'I', 'Z', 'J', 'D', '^', 'Q', 'N', 'F', 'V', 'G', '[', 'B', 'W', 'K', '\\', 'C', 'E', 'M', 'P', 'A', 'O', 'Y', ']', 'L'] , length: 320

Tutaj lepszy okazuje się jednak algorytm najbliższego sąsiada.

Wszystko zależy również od zakresu odległości - wag, jakie losujemy.
W powyższych wynikach odległości między miastami były losowane z zakresu (0; 100).
Przyjmijmy (0; 300) i ilość miast 30:

GENETIC ALGORITHMS:
The best cycle is ['Q', 'E', 'B', 'Z', 'A', 'J', 'O', 'U', 'F', '[', 'P', 'H', 'S', 'L', 'N', 'Y', 'T', '\\', 'K', 'G', 'W', '^', 'V', ']', 'M', 'D', 'X', 'C', 'I', 'R'] with length 1776.

NEAREST NEIGHBOUR ALGORITHM:
['L', 'U', 'T', ']', '[', 'H', 'W', 'C', 'I', 'F', '\\', 'K', 'O', 'G', 'A', 'J', 'X', 'E', 'S', 'Z', '^', 'V', 'D', 'R', 'B', 'P', 'Y', 'M', 'N', 'Q'] , length: 1349

GENETIC ALGORITHMS:
The best cycle is ['G', 'Y', 'M', 'N', 'Z', 'A', 'J', 'B', 'X', 'P', '\\', 'V', 'W', 'U', 'D', 'T', 'F', 'O', 'C', 'I', ']', '[', 'H', 'Q', 'E', 'R', '^', 'L', 'S', 'K'] with length 1697.

NEAREST NEIGHBOUR ALGORITHM:
['W', 'C', 'H', 'L', 'U', 'T', ']', '[', 'E', 'S', 'Z', 'G', 'A', 'J', 'K', 'O', 'F', '\\', 'M', 'B', 'I', 'R', '^', 'V', 'D', 'P', 'Y', 'N', 'Q', 'X'] , length: 1268

Powyższe wyniki dotyczą przypadku, gdy mamy liczbę iteracji 30, podwoimy ją.
Dla tego samego grafu i 60 iteracji otrzymujemy już:

GENETIC ALGORITHMS:
The best cycle is ['U', 'O', 'F', 'B', 'Z', 'I', 'L', 'W', 'G', 'T', '\\', 'M', 'E', 'V', 'Q', 'A', 'Y', 'C', 'N', 'D', 'R', '[', 'J', 'X', 'P', 'S', 'K', 'H', '^', ']'] with length 1271.

NEAREST NEIGHBOUR ALGORITHM:
['[', 'H', 'L', 'U', 'T', ']', 'W', 'C', 'I', 'F', '\\', 'K', 'O', 'G', 'A', 'J', 'X', 'E', 'S', 'Z', '^', 'V', 'D', 'R', 'B', 'P', 'Y', 'M', 'N', 'Q'] , length: 1357

GENETIC ALGORITHMS:
The best cycle is ['L', 'W', 'P', 'S', 'Z', 'G', 'T', ']', 'N', 'Y', 'C', 'K', 'O', 'U', 'J', '\\', 'H', 'B', 'X', 'E', 'R', '[', '^', 'V', 'Q', 'A', 'I', 'F', 'M', 'D'] with length 944.

NEAREST NEIGHBOUR ALGORITHM:
['C', 'H', 'L', 'U', 'T', ']', '[', 'E', 'S', 'Z', 'G', 'A', 'J', 'K', 'O', 'F', '\\', 'W', 'D', 'V', 'Q', 'Y', 'B', 'I', 'R', '^', 'M', 'N', 'P', 'X'] , length: 1178

Na podstawie powyższych wyników można wywnioskować, że wynik algorytmu najbliższych sąsiadów w niektórych przypadkach może być lepszy od najlepszego osobnika, stworzonego przez algorytmy genetyczne w danym pokoleniu, lecz niewątpliwie w dalszych pokoleniach znajduje się optymalny wynik, lepszy od tego, który otrzymaliśmy przez algorytm najbliższych sąsiadów.

ŹRÓDŁA
Tworząc program, wspierałam się informacjami o grafach z Pana strony:

http://users.uj.edu.pl/~ufkapano/algorytmy/lekcja14/index.html

Korzystałam również z kilku ogólnych opisów algorytmów genetycznych, między innymi ze strony:

http://www.k0pper.republika.pl/geny.htm

W przypadku implementacji wspierałam się, oczywiście:

https://www.python.org/doc/