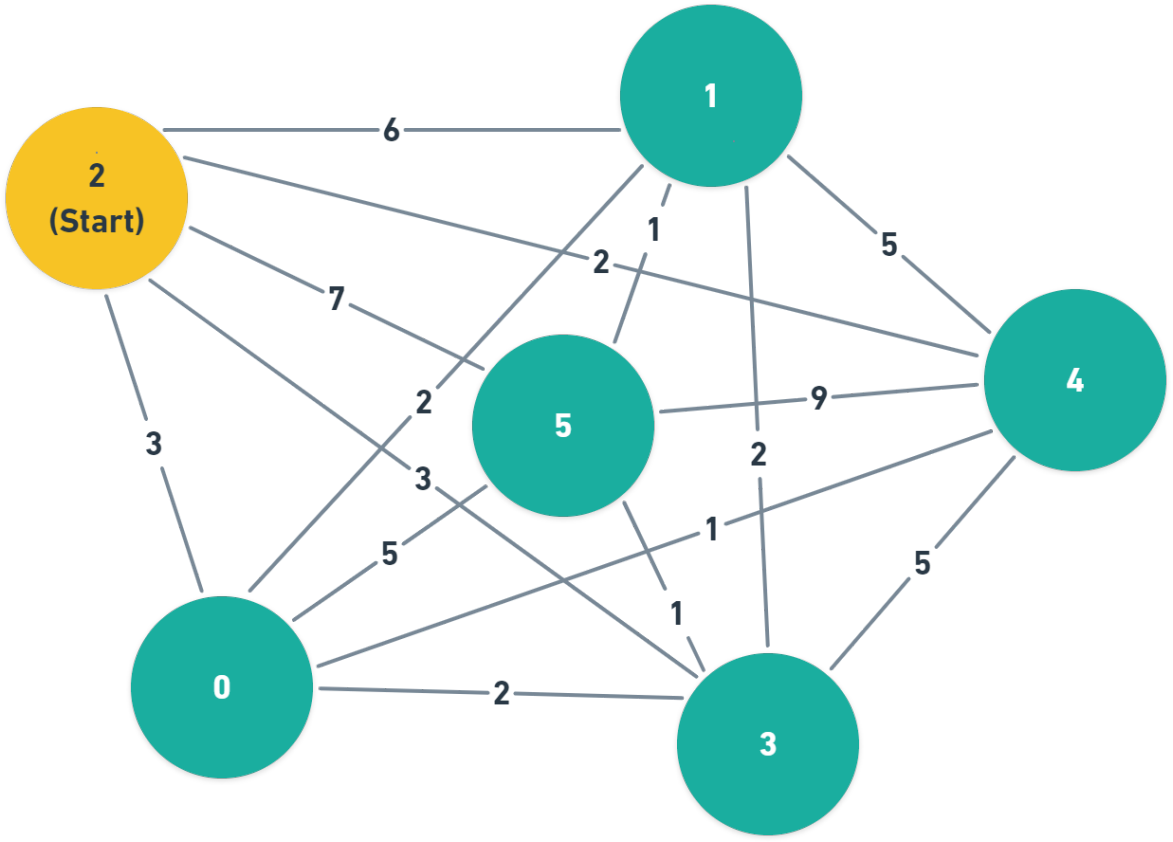
Damian Kaniewski 291565

Problem Komiwojażera

Problem polega na tym że, dane jest **N** miast, które komiwojażer ma odwiedzić, oraz odległość / cena podróży / czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej / najtańszej / najszybszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej się w określonym punkcie.

Przykładowy graf, na którym pracowałem, jest on spójny, każdy węzeł ma ścieżkę do każdego węzła:



Na krawędziach podane są koszty przejścia z danego węzła do kolejnego. Przyjąłem za punkt startowy węzeł nr. 2.

Rozwiązanie problemu Komiwojażera przedstawiam w języku C#.

Wykorzystane algorytmy:

- Algorytm najbliższego sąsiada,
- Algorytm symulowanego wyżarzania,

Aby rozwiązać problem Komiwojażera stworzyłem na podstawie grafu odpowiednią macierz sąsiedztwa:

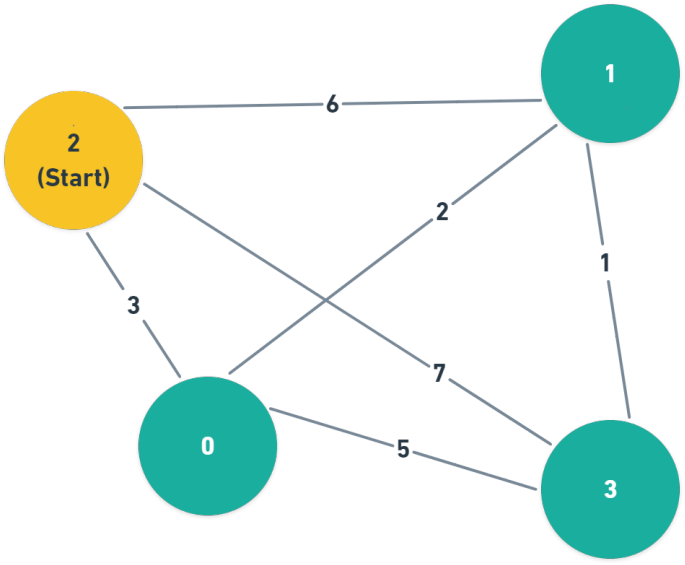
MACIERZ SĄSIEDZTWA

	0	1	2	3	4	5
0	0	2	3	2	1	5
1	2	0	6	2	5	1
2	3	6	0	3	2	7
3	2	2	3	0	5	1
4	1	5	2	5	0	9
5	5	1	7	1	9	0

W Macierzy sąsiedztwa znajduje się koszt.

Algorytm najbliższego sąsiada

W ramach przedstawienia działania Algorytmu wykorzystałem graf 4 węzłowy.



oraz jego macierz sąsiedztwa:

MACIERZ SĄSIEDZTWA

	0	1	2	3
0	0	2	3	5
1	2	0	6	1
2	3	6	0	7
3	5	1	7	0

Rozpatrujemy działanie algorytmu dla powyższego 4 węzłowego grafu spójnego:

```
startNode = 2
currentNode = startNode        2
minCost = inf                  2
visited[currentNode] = true
Oznaczamy węzeł startowy jako Odwiedzony

Pętla do zmiany węzła                    i = 0
  działa dopóki i < liczba węzłów - 1

  Pętla do rozpatrywania sąsiadów
    działa dopóki i < liczba węzłów - 1

    dla i = 0 i currentNode = 2   graph[2,0] = 3 <-- na podstawie Macierzy Sąsiedztwa
    if(visited[i] == false && graph[currentNode,i] < minCost      inf
    && graph[currentNode,i]  != 0)

      minCost = graph[currentNode,i]   minCost = 3;
      tmp = i                          tmp = 0
      dla i = 1 i i =3  i currentNode = 2 koszt jest wiekszy niz dla i = 0, dla i = 2, warunek visited[2] jest false,

    currentCost += graph[currentNode,tmp];   currentCost += 3
                                             dodajemy koszt przejscia do nowego wezla
    currentNode = tmp                  przesuwamy się do wezla 0
    visited[currentNode] = true;       oznaczamy wezel 0 jako odwiedzony
    minCost = int.MaxValue;            zmieniamy minCost na inf
```

Pętla do zmiany węzła **i = 1**

działa dopóki i < liczba węzłów - 1

Pętla do rozpatrywania sąsiadów

działa dopóki i < liczba węzłów - 1

dla i = 1 i currentNode = 0 graph[0,1] = 2 <-- na podstawie Macierzy Sąsiedztwa

inf

```
if(visited[i] == false && graph[currentNode,i] < minCost
&& graph[currentNode,i]  != 0)
```

```
minCost = graph[currentNode,i]   minCost = 2;
tmp = i   tmp = 1
```

dla i = 2 i i =3 i currentNode = 0 koszt jest wiekszy niz dla i = 1, dla i = 0, i = 2 warunek visited[i] jest false,

```
currentCost += graph[currentNode,tmp];
```

currentCost = 3 + 2 = 5

dodajemy koszt przejscia do nowego wezla

```
currentNode = tmp
```

przesuwamy się do wezla 1

```
visited[currentNode] = true;
```

oznaczamy wezel 1 jako odwiedzony

```
minCost = int.MaxValue;
```

zmieniamy minCost na inf

Pętla do zmiany węzła **i = 2**

działa dopóki i < liczba węzłów - 1

Pętla do rozpatrywania sąsiadów

działa dopóki i < liczba węzłów - 1

dla i = 3 i currentNode = 1 graph[1,3] = 1 <-- na podstawie Macierzy Sąsiedztwa

inf

```
if(visited[i] == false && graph[currentNode,i] < minCost
&& graph[currentNode,i]  != 0)
```

```
minCost = graph[currentNode,i]   minCost = 1;
tmp = i   tmp = 3
```

dla i = 0, i = 1, i = 2 warunek visited[i] jest false,

```
currentCost += graph[currentNode,tmp];
```

currentCost = 3 + 2 + 1 = 6

dodajemy koszt przejscia do nowego wezla

```
currentNode = tmp
```

przesuwamy się do wezla 3

```
visited[currentNode] = true;
```

oznaczamy wezel 3 jako odwiedzony

```
minCost = int.MaxValue;
```

zmieniamy minCost na inf

Pętla do zmiany węzła **i = 3**

działa dopóki i < liczba węzłów - 1

wszystkie wezly sa odwiedzone
currentCost = 3+2+1+0 = 6

```
endCost = currentCost +
graph[currentNode,
startNode];
```

wychodzimy z pentli dodajemy koszt przejscia z wezla w ktorym jestesmy do wezla startowego.

endCost = 6 + 7 = 13
sciezka to 2 -> 0 -> 1 -> 3 -> 2

Algorytm symulowanego wyżarzania

Rozpatrujemy działanie algorytmu dla 4 węzłowego grafu spójnego, przedstawionego na rysunku w zagadnieniu Algorytm Najbliższego Sąsiada.