"Dolina Meksyku"-AiSD

Jacek Kulma
Konrad Leszczyński

18 stycznia 2023r.

Spis treści

1 Wprowadzenie

W ramach naszego projektu przenieśliśmy się w czasie do około XIV w. do malowniczej doliny Meksyku wówczas jeszcze położonej nad rozległym jeziorem. Wokół tegoż jeziora znajdowały się miasta, które wspólnie handlowały. Niestety nie wszystkie miasta miały ze sobą porozumienie handlowe. Te, które jednak miały wysyłały fregaty handlowe, aby kursowały od jednego miasta do drugiego. Aby jednak rozwój handlu był bardziej prężny, przywódcy zdecydowali, że opracowana zostanie jedna trasa wiodąca przez wszystkie miasta wokół jeziora, tak żeby każda para kolejno odwiedzanych miast miała porozumienie handlowe. Problemem stała się jednak konstrukcja takiej trasy, aby i czas podróży fregaty był najkrótszy. I w tym momencie rozpoczęło się nasze wyzwanie.
Celem naszego zadania projektowego było napisanie algorytmu, który dla losowej liczby $n \leqslant 20$ miast oraz losowej liczby m dwustronnych porozumień handlowych opracowywał najkrótszą trasę przechodzącą przez wszystkie miasta wokół jeziora oraz podawał czas jej przepłynięcia.

2 Implementacja grafu

W naszym programie zaimplementowaliśmy sieć połączeń handlowych między poszczególnymi miastami w Dolinie Meksyku jako graf ważony nieskierowany, w którym wierzchołki odpowiadają miastom, krawędzie porozumieniom handlowym między tymi miastami a wagi czasowi przepłynięcia z jednego miasta do drugiego.
W naszym programie posłużyliśmy się programowaniem obiektowym. Wierzchołki oraz graf ważony nieskierowany zaimplementowaliśmy jako połączone ze sobą klasy obiektów o podanych niżej metodach.

2.1 Klasa $Vertex$

2.1.1 Metody

- **init(key)** - tworzy nowy obiekt typu vertex (wierzchołek), który posiada następujące parametry :
  - **id=key** - wskaźnik na konkretny wierzchołek;
  - **connectedTo={}** - pusty słownik, w którym klucze to sąsiadzi rozważanego wierzchołka(obiekty typu Vertex), a wartości to wagi połączenia między nimi;

- **visited=False** - wskaźnik logiczny, który informuje czy wierzchołek został odwiedzony w algorytmie.

- **addNeighbor(nbr,weight=0)** - funkcja, która do słownika sąsiadów(connectedTo) wstawia nowy wierzchołek i wagę połączenia, czyli "nbr:weight";
- **getConnections()** - zwraca listę wierzchołków(obiektów typu Vertex) sąsiadujących ze słownika connectedTo;
- **getId()** - zwraca Id wierzchołka;
- **getWeight(nbr)** - zwraca wagę połączenia z wierzchołkiem nbr ze słownika connectedTo;
- **removeNeighbours()** - usuwa wszystkich sąsiadów wierzchołka poprzez zastąpienie słownika connecetedTo pustym słownikiem.

2.2 Klasa *Graph*

2.2.1 Metody

- **init()** - tworzy obiekt będący grafem ważonym, nieskierowanym o parametrach:
  - **vertlist={}** - pusty słownik wierzchołków grafu, w którym klucze to Id wierzchołków, a wartości to obiekty Vertex(Id);
  - **numVertices=0** - liczba wierzchołków w grafie równa na początku 0;
- **addVertex(key)** - powiększa parametr numVertices o 1 oraz do słownika vertlist wstawia element "key:Vertex(key)";
- **getVertex(n)** - zwraca ze słownika vertlist wierzchołek(obiekt typu Vertex) o Id=n;
- **addEdge(f,t,weight=0)** - funkcja wykonuje dwie rzeczy:
  - Jeśli wierzchołek o Id=f lub wierzchołek o Id=t nie znajdują się w słowniku vertlist, wówczas funkcja dodaje je do grafu przy pomocy funkcji addVertex()
  - Do słownika sąsiadów wierzchołka t dodawany jest element "Vertex(f):weight" przy pomocy funkcji addNeighbor(). Symetrycznie wykonywane jest to samo dla wierzchołka f.
- **getVertices()** - zwraca listę składajacą się z Id wierzchołków grafu.

3 Algorytm i jego implementacja

3.1 Opis algorytmu

Rozwiązaniem problemu jest ścieżka hamiltona, tzn. ścieżka przechodząca przez każdy wierzchołek dokładnie raz, o najkrótszym łącznym czasie podróży.
W celu znalazienia takiej ścieżki stosujemy algorytm przechodzenia grafu w głąb, zapamiętując po drodze wszystkie możliwe ścieżki oraz czasy potrzebne do ich pokonania. Na końcu wybieramy najszybszą z nich.

3.2 Implementacja

3.2.1 Zmienne pomocnicze

- **used** - liczba odwiedzonych wierzchołków. Jeśli równa się ona liczbie wszystkich wierzchołków w grafie, to znaleźliśmy ścieżkę hamiltona.
- **cost** - zmienna przechowująca łączny koszt (czas potrzebny do przebycia) obecnie rozważanej ścieżki. Początkowo przyjmuje wartość 0.
- **cheapest_cost** - ta zmienna przechowuje koszt (czas) potrzebny do przebycia najszybciej, do tej pory znalezionej, ścieżki hamiltona. Na początku przyjmuje wartość ∞. Jeśli po przejsciu wszytskich wierzchołków grafu okaże się, że **cost** $<$ **cheapest_cost**, to zmienna **cheapest_cost** przyjmuje wartość zmiennej **cost**.
- **path** - lista wierzchołków tworzących obecnie rozważaną ścieżkę. Początkowo jest pusta i stopniowo dodajemy do niej kolejne wierzchołki.
- **best_path** - lista wierzchołków tworzących najkrótszą ścieżkę hamiltona. Na początku jest ona pusta. Przypisujemy do niej listę **path**, gdy końcowy koszt rozważanej ścieżki jest mniejszy od kosztu najlepszej, do tej pory znalezionej, trasy.
- **weights** - lista czasów połączeń pomiędzy kolejnymi wierzchołkami w obecnej rozważanej ścieżce. Na starcie **weights** $= \{0\}$ - koszt połączenia pierwszego rozważanego wierzchołka z nim samym jest zerowy.

3.2.2 Kod

Poniższy algorytm znajduje najszybszą ściezkę hamiltona w grafie G, rozpoczynając poszukiwania w zadanym wierzchołku. W celu znalezienia najlepszej trasy algorytm należy wywołać dla każdego wierzchołka i wybrać najlepszy z wyników.

```
begin hamiltonian_path_list(int ver)
    Visited(ver):=true;
    path[used]:=ver;
    used:=used+1;
    for all w ∈ Neighbours(v) do
        if ¬Visited(w) then
            weights[used]:=koszt połączenia ver z w;
            hamiltonian_path_list(w) fi;
        if used=|G| then
            for i=0:|G|-1 do
                cost:=cost+weights[i];
            if cost < cheapest_cost then
                cheapest_cost:=cost;
                best_path:=path; fi;
            cost:=0; fi;
    Visited(ver):=false;
    path[used]:=puste;
    weights[used]:=puste;
    used:=used-1;
    return (best_path, cheapest_cost);
end hamiltonian_path_list;
```

4 Złożoność czasowa

Algorytm szukania ścieżki hamiltona jest problemem NP-zupełnym. W ogólności nie wiadomo czy problem tej klasy daje się rozwiązać w czasie wielomianowym lub logarytmicznym.
Przy szukaniu najkrótszej ścieżki hamiltona w grafie ważonym, w najbardziej pesymistycznym przypadku, mamy do czynienia z problemem komiwojażera. Dzieje się tak, gdy rozważany graf jest grafem pełnym. W tej sytuacji liczba kombinacji (wszystkich ścieżek hamiltona) wynosi $\frac{(n-1)!}{2}$. Zatem ten problem ma złożoność czasową typu silnia.
Możemy zauważyć, że dla $n = 20$ ilość kombinacji to $\frac{19!}{2} \approx 6 \times 10^{16}$, co sprawia, że rozwiązanie tego problemu, w jakimkolwiek sensownym czasie staje się praktycznie niewykonalne.

5 Wyniki i testy

Niech n oznacza liczbę wierzchołków grafu G. Dla $3 \leqslant n \leqslant 10$ wylosowaliśmy po 100 grafów i zbadaliśmy średni czas potrzebny do przebycia najkrótszej trasy. Dodatkowo sprawdziliśmy w ilu przypadkach powstały grafy nieposiadające żadnej ścieżki hamiltona. Uzyskane wyniki przedstawia poniższa tabela.

n	średni koszt najlepszej trasy	przypadki bez rozwiązania
3	9,3	46
4	13,2	48
5	17,3	43
6	18,2	47
7	19,2	42
8	23,2	38
9	25,8	36
10	28,9	29

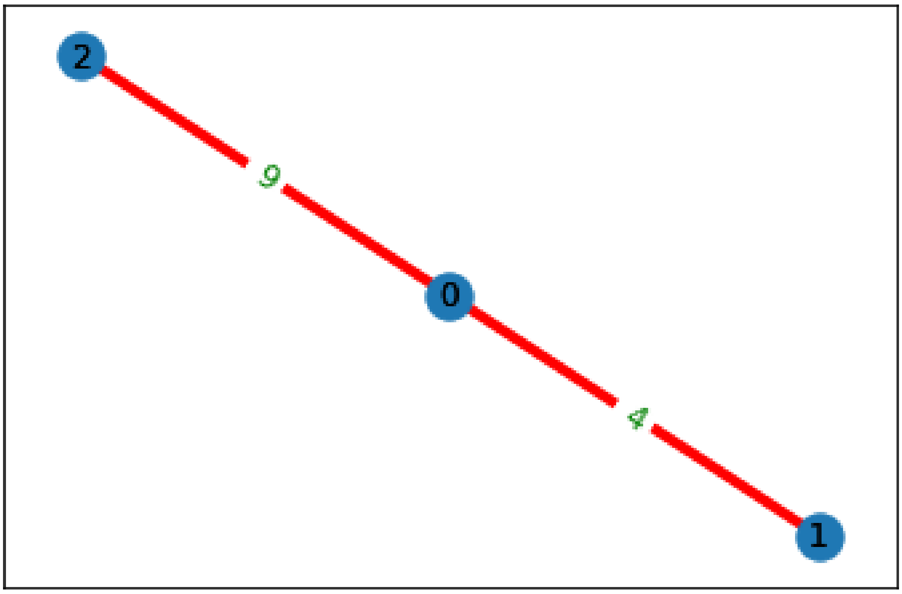
Jak można było się spodziewać, średni koszt najszybszej trasy rośnie, gdy zwiększamy ilość wierzchołków w grafie. Liczba przypadków bez rozwiązania (grafów bez ani jednej scieżki hamiltona) zachowuje się zupełnie na odwrót. Testów dla $n > 10$ nie wykonujemy ze względu na złożoność czasową algorytmu.

6 Zwizualizowane przykłady

Do wizualizacji sporządzanych przykładów i testów posłużyliśmy się zewnętrznym pakietem NetworkX służącym m.in. do badania wykresów, grafów i sieci.

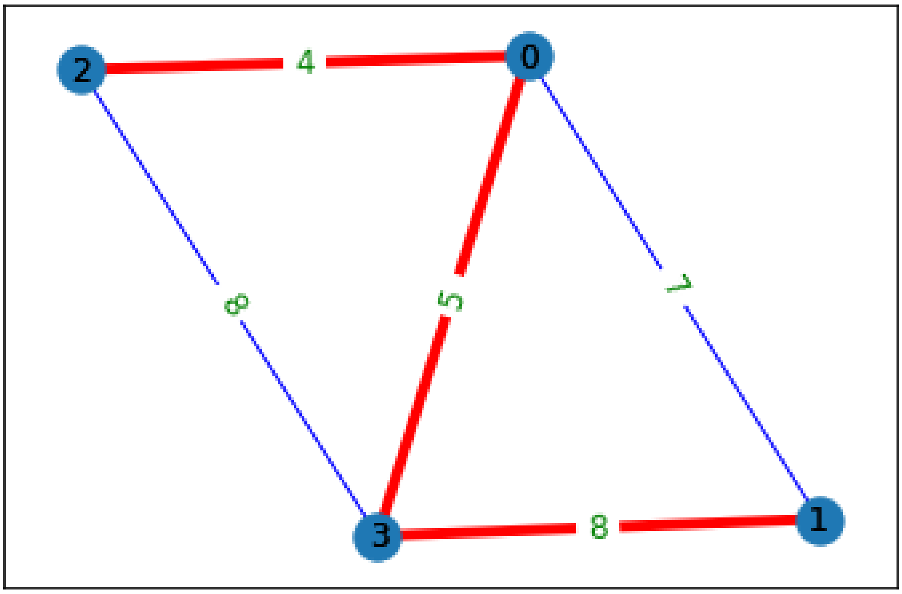
6.1 Przykład 1

Liczba wierzchołków:3
Liczba krawędzi:2
Najkrótsza trasa:(1,0,2)
Czas najkrótszej trasy:13



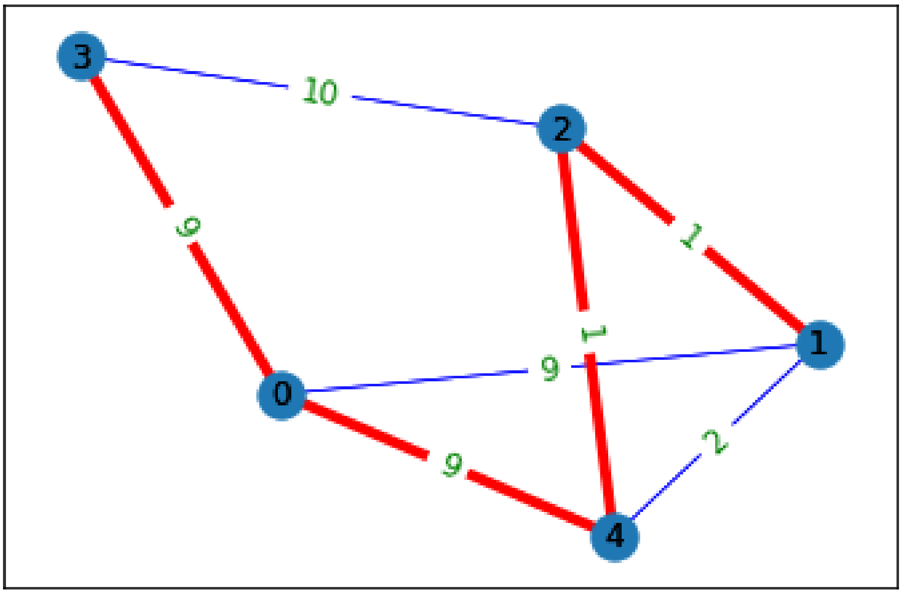
6.2 Przykład 2

Liczba wierzchołków:4
Liczba krawędzi:5
Najkrótsza trasa:(1,3,0,2)
Czas najkrótszej trasy:17



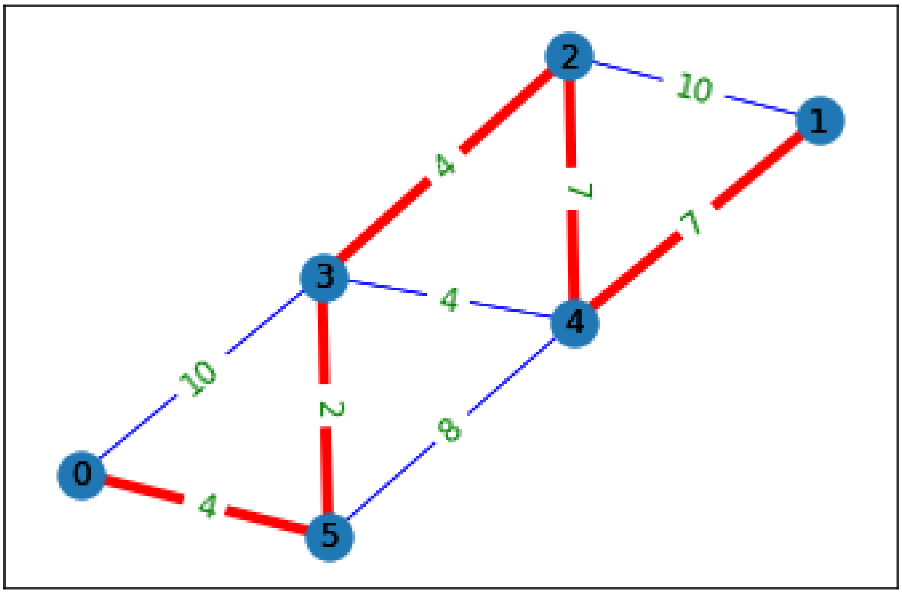
6.3 Przykład 3

Liczba wierzchołków:5
Liczba krawędzi:7
Najkrótsza trasa:(1,2,4,0,3)
Czas najkrótszej trasy:20



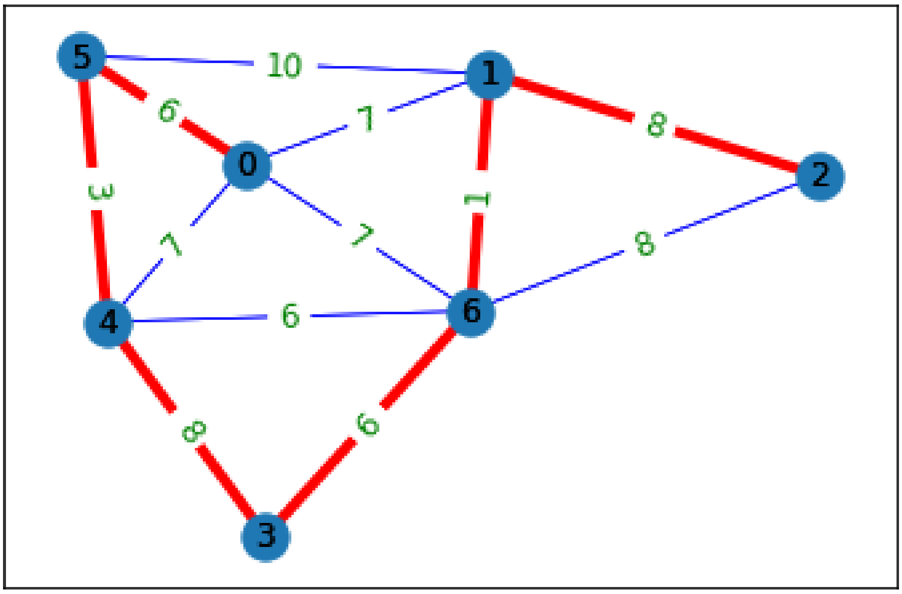
6.4 Przykład 4

Liczba wierzchołków:6
Liczba krawędzi:9
Najkrótsza trasa:(0,5,3,2,4,1)
Czas najkrótszej trasy:24



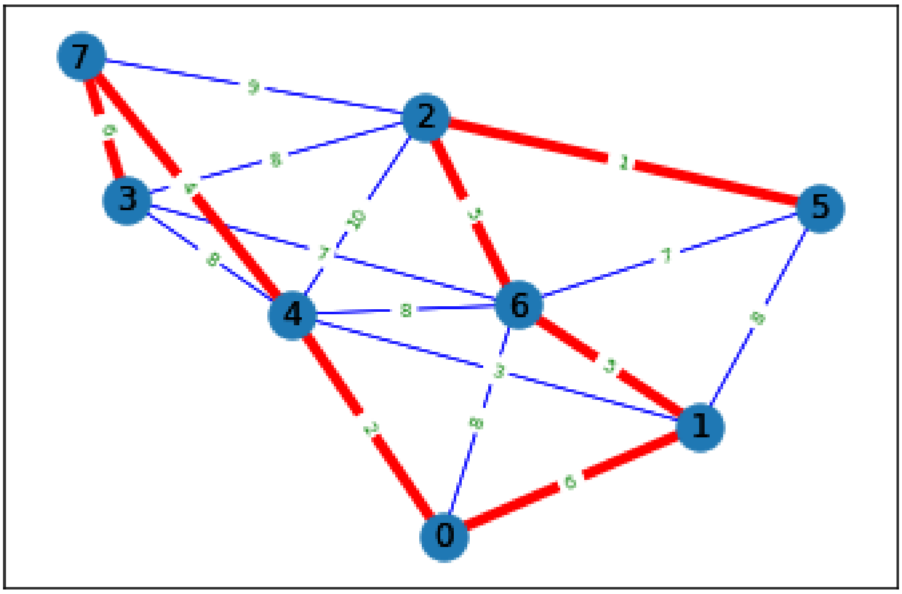
6.5 Przykład 5

Liczba wierzchołków:7
Liczba krawędzi:12
Najkrótsza trasa:(0,5,4,3,6,1,2)
Czas najkrótszej trasy:32



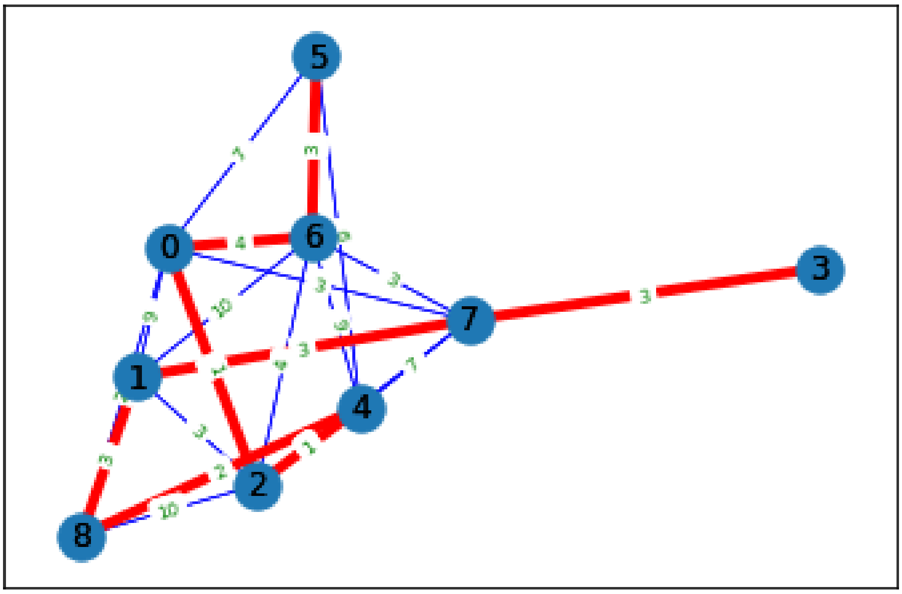
6.6 Przykład 6

Liczba wierzchołków:8
Liczba krawędzi:17
Najkrótsza trasa:(3, 7, 4, 0, 1, 6, 2, 5)
Czas najkrótszej trasy:29



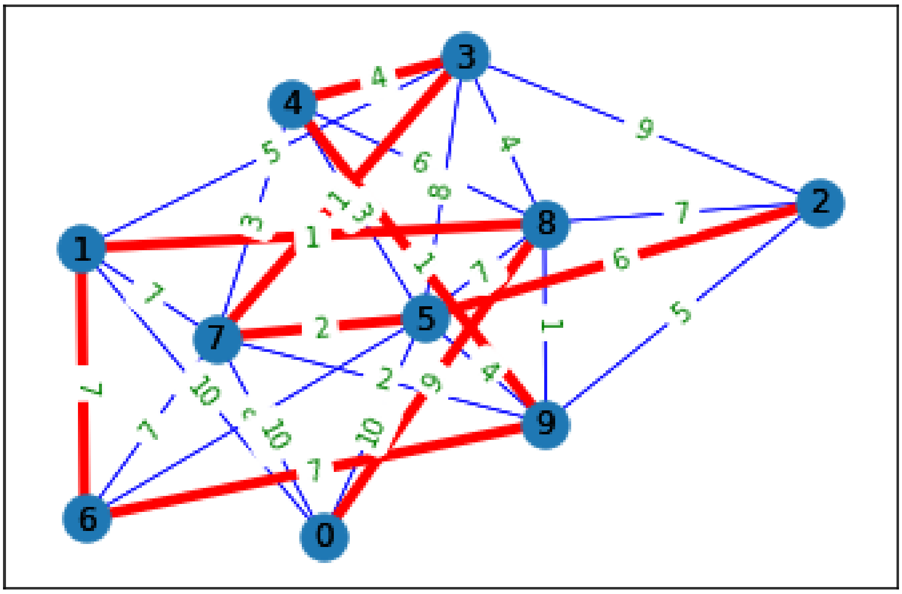
6.7 Przykład 7

Liczba wierzchołków:9
Liczba krawędzi:21
Najkrótsza trasa:(3, 7, 1, 8, 4, 2, 0, 6, 5)
Czas najkrótszej trasy:20



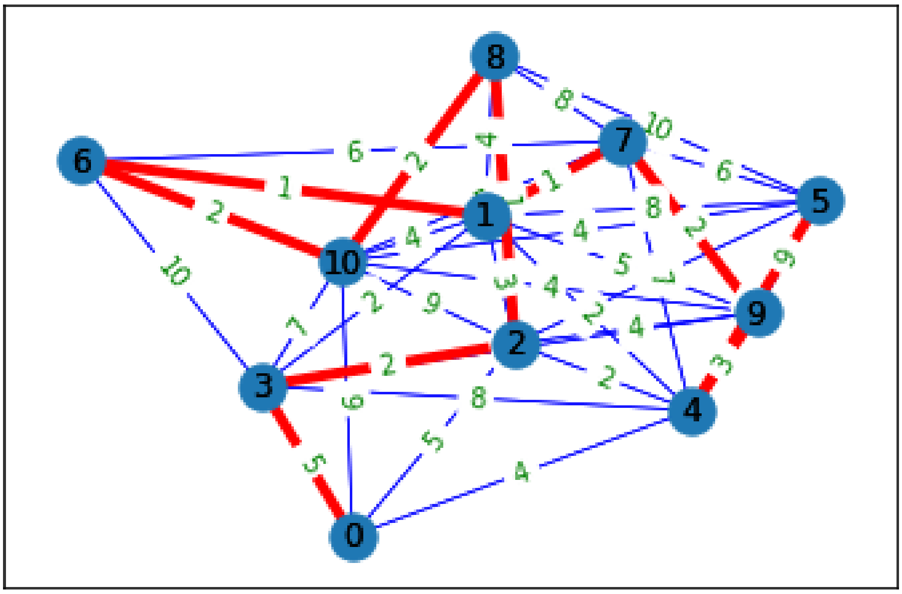
6.8 Przykład 8

Liczba wierzchołków:10
Liczba krawędzi:28
Najkrótsza trasa:(0, 8, 1, 6, 9, 4, 3, 7, 5, 2)
Czas najkrótszej trasy:38



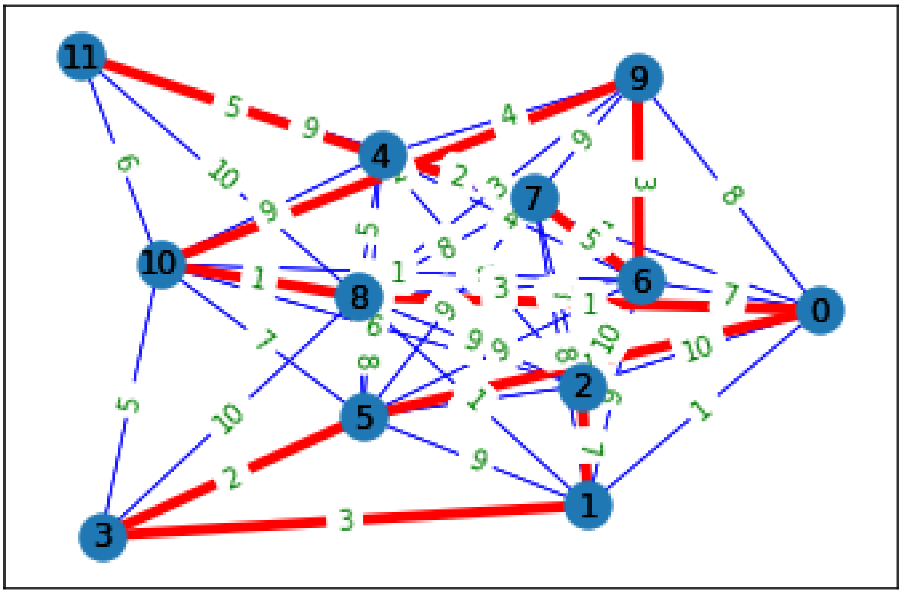
6.9 Przykład 9

Liczba wierzchołków:11
Liczba krawędzi:37
Najkrótsza trasa:(0, 3, 2, 8, 10, 6, 1, 7, 9, 4, 5)
Czas najkrótszej trasy:21



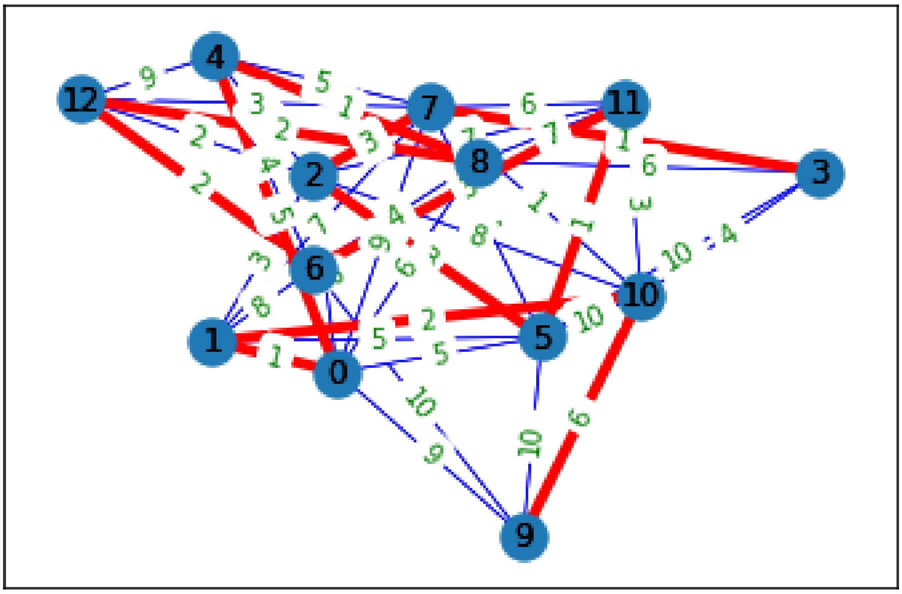
6.10 Przykład 10

Liczba wierzchołków:12
Liczba krawędzi:45
Najkrótsza trasa:(2, 1, 3, 5, 0, 8, 10, 9, 6, 7, 4, 11)
Czas najkrótszej trasy:32



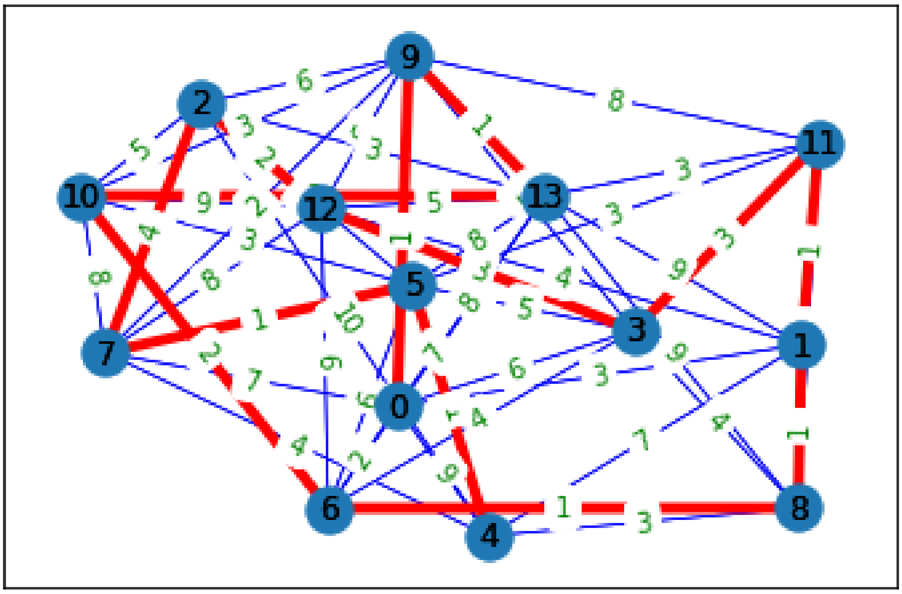
6.11 Przykład 11

Liczba wierzchołków:13
Liczba krawędzi:41
Najkrótsza trasa:(3, 7, 2, 5, 11, 6, 12, 8, 4, 0, 1, 10, 9)
Czas najkrótszej trasy:30



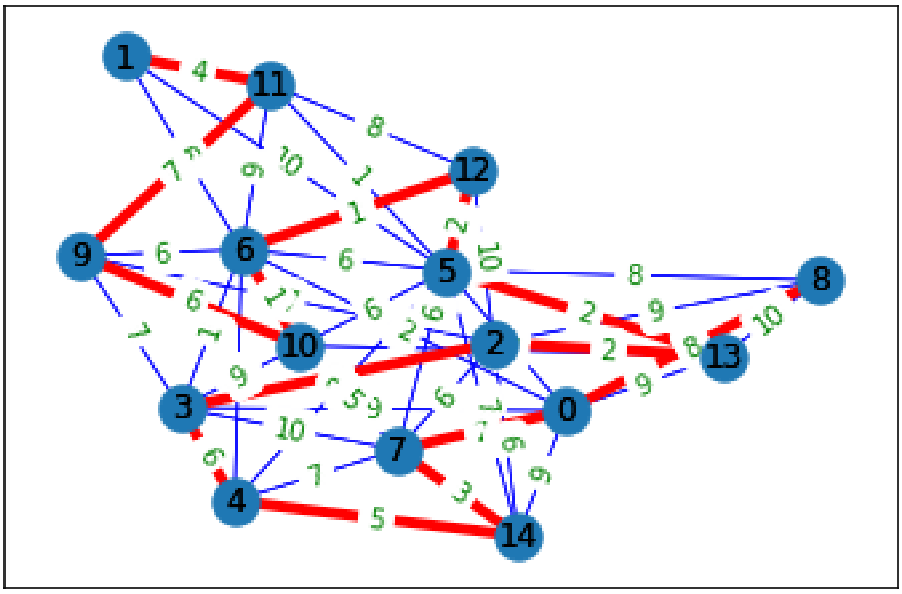
6.12 Przykład 12

Liczba wierzchołków:14
Liczba krawędzi:50
Najkrótsza trasa:(0, 9, 13, 10, 6, 8, 1, 11, 3, 12, 2, 7, 5, 4)
Czas najkrótszej trasy:23



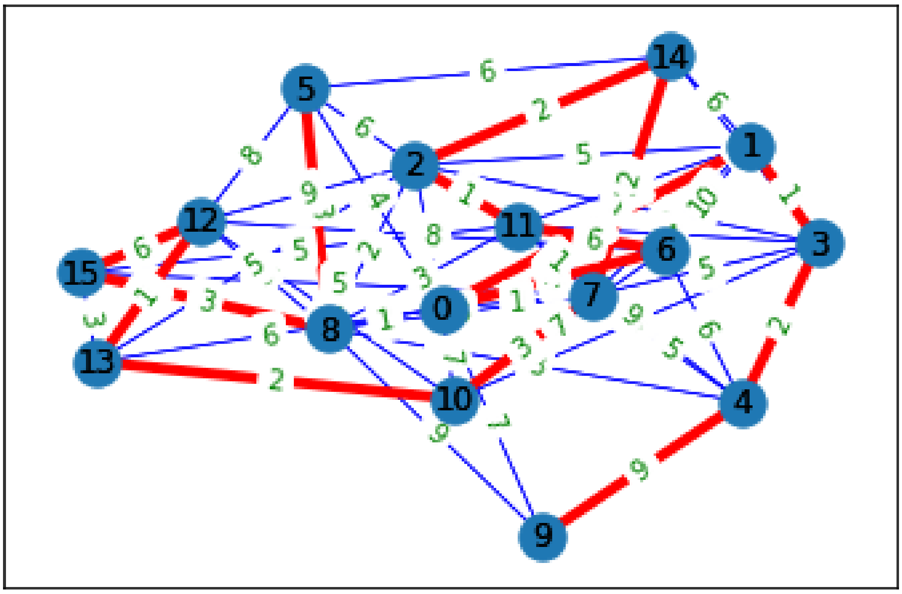
6.13 Przykład 13

Liczba wierzchołków:15
Liczba krawędzi:45
Najkrótsza trasa:(1, 11, 9, 10, 6, 12, 5, 13, 2, 3, 4, 14, 7, 0, 8)
Czas najkrótszej trasy:50



6.14 Przykład 14

Liczba wierzchołków:16
Liczba krawędzi:53
Najkrótsza trasa:(5, 8, 15, 12, 13, 10, 7, 14, 2, 11, 6, 0, 1, 3, 4, 9)
Czas najkrótszej trasy:46



Literatura

[1] Lech Banachowski, Krzysztof Diks, Wojciech Rytter (2021) *Algorytmy i Struktury Danych*, Wydawnictwo Naukowe PWN.

[2] Anna Radzikowska (2022) *Algorytmy i struktury danych, Wykład 4: Algorytmy Grafowe I*, MiNI PW, r.ak. 2022/2023.

[3] Wikipedia, Problem komiwojażera:
https://pl.wikipedia.org/wiki/Problem_komiwoja%C5%BCera

[4] Wikipedia, Problem NP-zupełny:
https://pl.wikipedia.org/wiki/Problem_NP-zupe%C5%82ny