Ewelina Matusiak

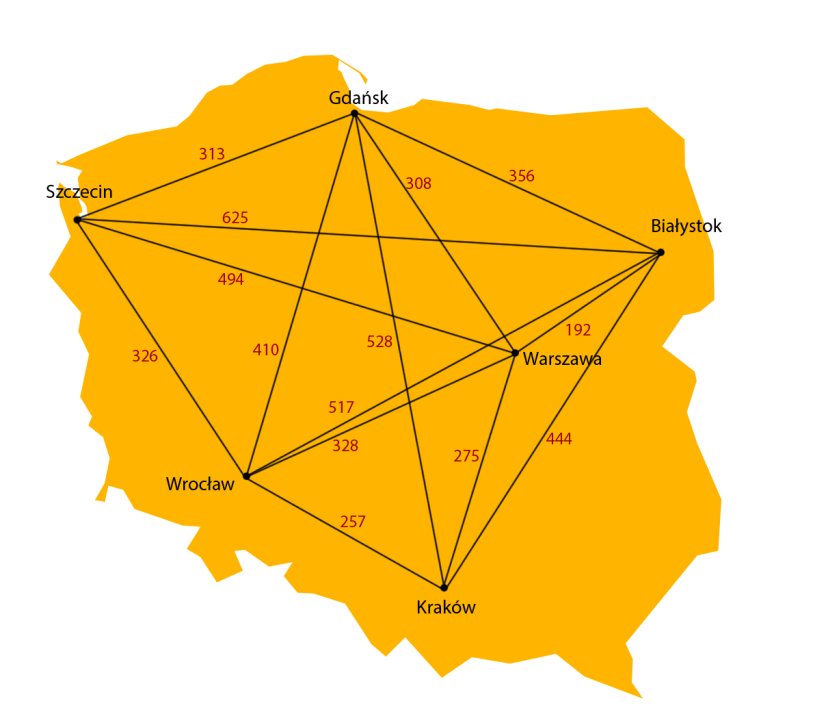
Sandra Tołkacz

Bartłomiej Sawa

**Algorytm rozwiązania problemu komiwojażera w dowolnym grafie spójnym, nieskierowanym (na podstawie DFS)**

**1. Wstęp**

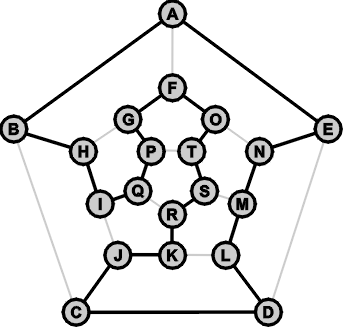
Problem komiwojażera opisujemy następująco: dane jest n miast, a każde dwa z nich połączone są drogami o pewnej długości. W jednym z miast znajduje się komiwojażer, który chce odwiedzić wszystkie w taki sposób, aby w każdym mieście znaleźć się dokładnie jeden raz, a na koniec wędrówki powrócić do miejsca startowego. Naszym celem jest znalezienie najkrótszej możliwej trasy dla komiwojażera. W życiu codziennym problem ten spotyka kierowcę tira, który musi wybrać najkrótszą drogę, aby objechać wszystkie miasta i dowieźć ładunek na czas. Można go również łatwo przedstawić w języku teorii grafów.



Rys 1. Przykładowa trasa dla kierowcy tira.

Dany jest graf G z krawędziami opisanymi dodatnimi wagami, które odpowiadają odległościom między miastami. Niech G(u, v) oznacza wagę krawędzi uv. Chcemy znaleźć w tym grafie cykl Hamiltona (czyli cykl przechodzący przez każdy wierzchołek dokładnie jeden raz) o minimalnej sumie wag krawędzi. Ponieważ cykl musi przechodzić przez wszystkie wierzchołki, więc wybór wierzchołka początkowego nie ma znaczenia.

**2. Znajdowanie cyklu Hamiltona**

****

Rys. 2 Cykl Hamiltona (A B H I Q P G F O T S R K J C D L M N E A)

Za pomocą odpowiednio zmodyfikowanej procedury DFS przeglądamy graf począwszy od wierzchołka nr 0. Przetwarzany wierzchołek umieszczamy na stosie. Następnie sprawdzamy, czy stos zawiera n wierzchołków. Jeśli tak, to otrzymaliśmy ścieżkę Hamiltona. W takim przypadku sprawdzamy, czy ostatni wierzchołek ścieżki jest połączony krawędzią z wierzchołkiem nr 0. Jeśli tak, to ścieżka tworzy cykl Hamiltona - wyprowadzamy jej zawartość ze stosu (dla cyklu dodatkowo dodajemy na końcu wierzchołek 0), usuwamy ostatni wierzchołek ze stosu i wycofujemy się na wyższy poziom rekurencji, gdzie będą rozważone inne ścieżki lub cykle Hamiltona.

Jeśli stos nie zawiera n wierzchołków, to rekurencyjnie wywołujemy procedurę DFS dla wszystkich nieodwiedzonych sąsiadów. Wierzchołek usuwamy ze stosu i kasujemy informację o jego odwiedzeniu, po czym wycofujemy się na wyższy poziom rekurencji.

W wyniku działania tej procedury otrzymamy zawsze parzystą ilość cykli Hamiltona. Dzieję się tak ponieważ graf jest spójny i nieskierowany oraz połowa wyników tego algorytmu się powtarza.

**2.1 Algorytm wyszukiwania ścieżek i cykli Hamiltona**

**Funkcja rekurencyjna DFS(n,graf,v,visited,S)**

**Wejście:**

* **n** -liczba wierzchołków w grafie (nєC)
* **graf** -zadany w dowolnie wybrany sposób
* **v** -wierzchołek grafu (vєC)
* **visited** -n-elementowa tablica odwiedzin
* **S** -stos wierzchołków

**Wyjście**

* Wszystkie ścieżki i cykle Hamiltona w grafie

**Elementy pomocnicze**

* **test** -zmienna logiczna do testowania cyklu lub ścieżek Hamiltona
* **u** -wierzchołek grafu (uєC)

**Lista Kroków:**

1: S.push(v) //umieszczamy na stosie

2: **Jeśli** S zawiera mniej niż n wierzchołków **to idź do** 10

3: test false //zakładamy brak cykli Hamiltona

4: **Dla** każdego sąsiada u wierzchołka v, **wykonuj** 5//przeglądamy sąsiadów

5: **Jeśli** u=0, **to** test true i **idź do** 6

6: **Jeśli** test = true, **to pisz** "Cykl Hamiltona: "

**inaczej pisz** "Ścieżka Hamiltona: "

7: **Pisz** zawartość S bez usuwania danych z S

8: **Jeśli** test = true, **to pisz** 0

9: **Idź do** 14

10: visited[v] true //oznaczamy wierzchołek jako odwiedzony

11: **Dla** każdego sąsiada u wierzchołka v, **wykonuj** 12//przeglądamy sąsiadów wierzchołka v

12: **Jeśli** visited[u] = false, **to** DFS(n,graf,v,visited,S) //dla sąsiadów nieodwiedzonych

wykonujemy wywołanie rekurencyjne

13: visited[v] false //wycofujemy się z wierzchołka

14: S.pop()//usuwamy wierzchołek ze stosu

15:**Zakończ**

Funkcję należy wywołać z pustym stosem S i wyzerowaną tablica visited[] jako DFS(n,graf,v,visited,S)

**3. Problem komiwojażera**

Należy znaleźć w grafie ścieżkę o najmniejszej sumie wag krawędzi, która przechodzi dokładnie jeden raz przez każdy wierzchołek i wraca do wierzchołka startowego. Jeśli liczba wierzchołków w grafie nie jest duża i graf nie jest grafem zupełnym, to rozwiązanie problemu komiwojażera możemy uzyskać algorytmem, który wyznacza wszystkie cykle Hamiltona, liczy sumę wag krawędzi i zwraca cykl o najmniejszej sumie wag. Rozwiązanie zawsze zwróci cykl optymalny, a nie przybliżony.

**3.1 Algorytm rozwiązujący problem komiwojażera dla małych grafów**

**Wejście**

* **n** -liczba wierzchołków w grafie (nєC)
* **graf** -zadany w dowolnie wybrany sposób
* **v** -wierzchołek bieżący (vєC)
* **v0** -wierzchołek startowy (v0єC)
* **d** -suma wag krawędzi cyklu Hamiltona (dєC)
* **dH** -pomocnicza suma wag krawędzi (dHєC)
* **S -**stos wierzchołków
* **SH** -pomocniczy stos wierzchołków
* **visited -**n-elementowa tablica logiczna odwiedzin.

**Wyjście**

* **S -**stos zawierający numery kolejnych wierzchołków cyklu Hamiltona
* **d** -suma wag krawędzi cyklu Hamiltona

**Elementy pomocnicze**

* **u -**wierzchołek (uєC)

**Lista kroków:**

1: SH.push(v) //odwiedzony wierzchołek dopisujemy do ścieżki

2: **Jeśli** SH nie zawiera n wierzchołków, **to idź do** 10 //jeśli brak ścieżki Hamiltona to przechodzimy do wyszukiwania

3: **Jeśli** nie istnieje krawędź z v do v0, **to idź do** 17 //jeśli ścieżka nie jest cyklem to ją odrzucamy

4: dHdH + waga krawędzi z v do v0 //uwzględniamy w sumie wagę ostatniej krawędzi cyklu

5: **Jeśli** dHd, **to idź do** 8

6: d dH

7: Skopiuj stos SH do S

8: dHdH - waga krawędzi z v do v0

9: **Idź do** 17

10: visited[v] true

11: **Dla** każdego sąsiada u wierzchołka v wykonuj 12...15

12: **Jeśli** visited[u] = true, **to następny obieg pętli** 11

13: dHdH + waga krawędzi z v do v0

14: TPS(n,graf,u,v0,d,dH,S,SH,visited)

15: dHdH - waga krawędzi z v do v0

16: visited[v] false

17: SH.pop()

18: **Zakończ**

**Lista kroków dla algorytmu głównego**

1: Utwórz i wyzeruj visited, S, SH

2: d, dH0

3: TPS(v0)

4: **Jeśli** S.empty() = false, **to pisz** S oraz d **inaczej pisz** "Brak cyklu Hamiltona"

5: **Zakończ**

**4. Program oraz instrukcja**

1. Aby obliczyć rozwiązanie dla grafu G, należy stworzyć plik tekstowy z macierzą wag grafu.

Dla grafu o 5 wierzchołkach plik powinien wyglądać tak:

0; a; b; c; d;

a; 0; e; f; g;

b; e; 0; h; i;

c; f; h; 0; j;

d; g; i; j; 0;

Każda liczba od A do J musi być liczbą rzeczywistą, a wartości liczbowe na głównej przekątnej równe 0.

2. Ponieważ zakres liczb należy do Int64 nie można liczyć rozwiązania dla grafów o ilości wierzchołków większej niż 20.

3. Ze względu na wiele czynników takich jak na przykład czasy odczytu/zapisu plików na dysku, szybkością procesora etc. program jest efektywny jedynie dla ilości wierzchołków mniejszej niż 12.

4. Program powinien być uruchamiany z dysku, na którym jest dużo wolnej przestrzeni. Do przechowywania plików tymczasowych korzysta z plików tekstowych, których wielkość potrafi dochodzić do kilkudziesięciu GB przy liczeniu grafów o wielu wierzchołkach.

5. W folderze dołączonym do programu znajduje się kilka wcześniej przygotowanych grafów, które są gotowe do policzenia.

**5. Wnioski**

Algorytm jest wydajny tylko dla grafów o małej ilości wierzchołków. Podstawowa wada to wykładnicza złożoność obliczeniowa, która uniemożliwia analizę większych grafów.