Projektowanie Efektywnych Algorytmów

Projekt

19/10/2021

252690 Karolina Nogacka

(1) Brute force

spis treści	***strona***

1. **Sformułowanie zadania:**

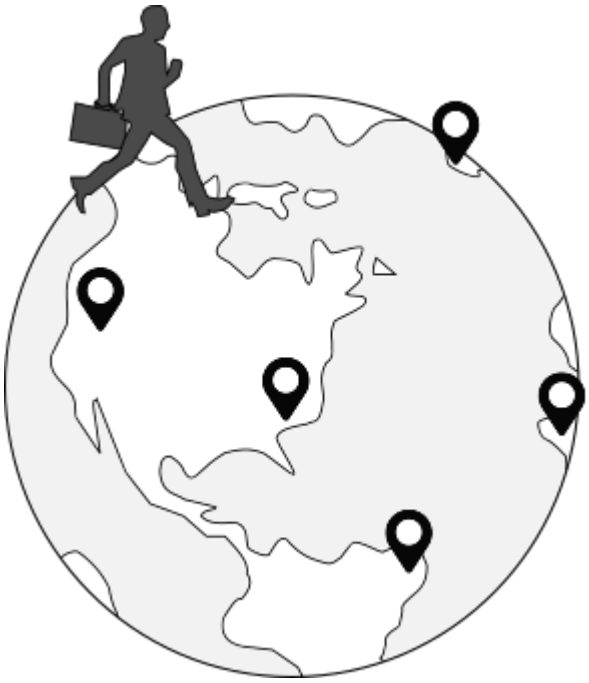
   Zadanie polega na stworzeniu algorytmu znajdującego rozwiązanie problemu komiwojażera opartego o metodę przeglądu zupełnego (Brute force). Algorytm będzie znajdował najbardziej optymalne rozwiązanie, lecz jedynie dla stosunkowo małych instancji problemu.

   Problem komiwojażera (TSP - Traveling salesman problem):

   Zakładając, że ktoś (np. kurier, sprzedawca, listonosz) ma zbiór miast/domów do odwiedzenia szukamy drogi, która zawierać będzie wszystkie wyżej wymienione miejsca i będzie drogą najkrótszą (aby listonosz nie musiał się nachodzić).

   Innymi słowy problem ten jest zagadnieniem optymalizacyjnym , polegającym na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Wszystkich cykli w takim grafie będzie n! (w związku z czym złożoność dla większych grafów będzie szybko rosła), pewne z nich będą w rzeczywistości tymi samymi cyklami, zaczynającymi się jednak w różnych węzłach grafu.

   Spodziewane ograniczenie górne funkcji czasu to **O(n!).**

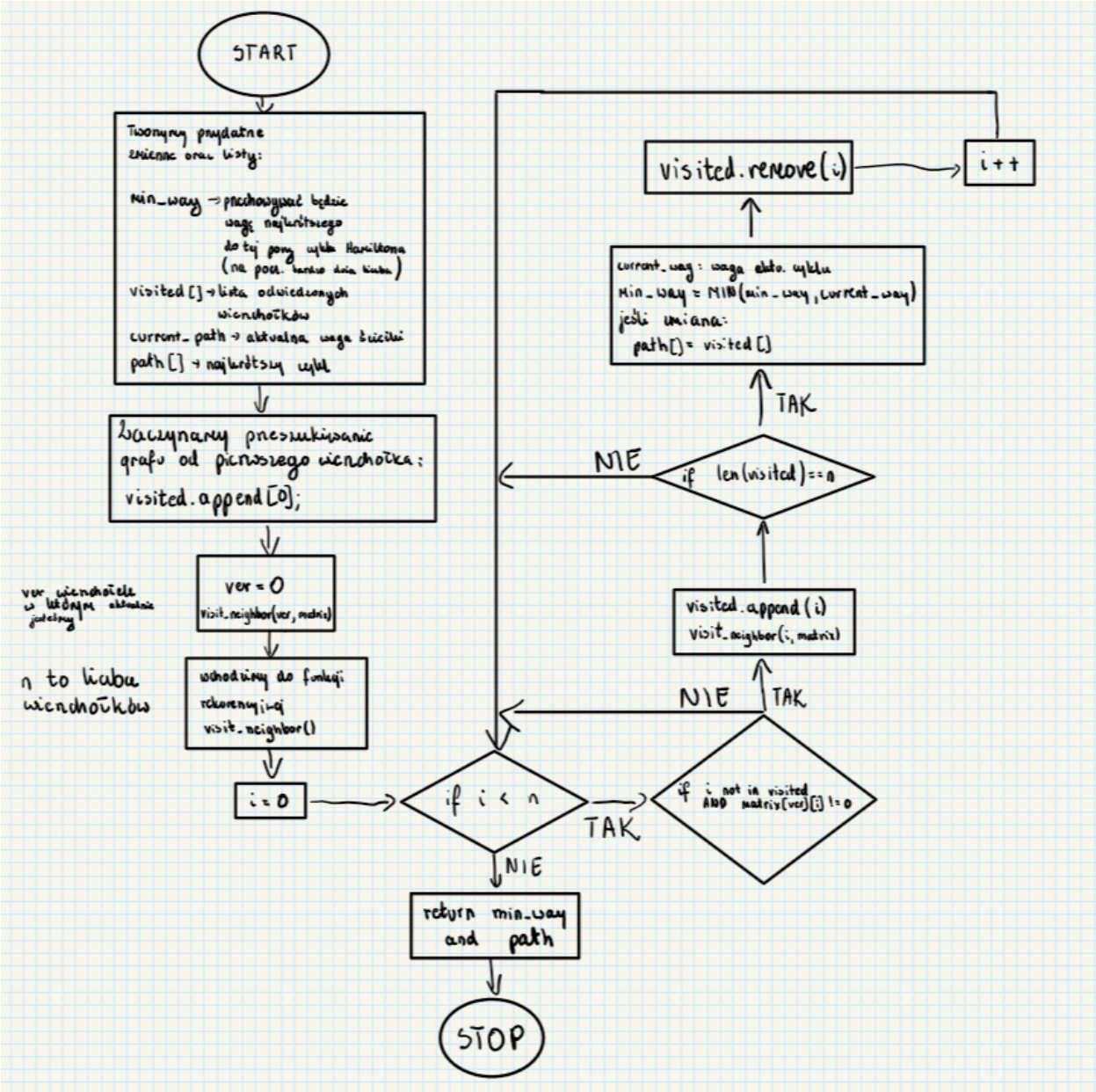


2. Metoda:

Użyta w tym etapie metoda, brute force (przeglądu zupełnego), polega na wygenerowaniu każdego możliwego, dopuszczalnego rozwiązania problemu i wybraniu z nich tego które dany problem rozwiązuje. Jest ona czaso- i zasobo-chłonna. Za jej pomocą zawsze uzyskamy poprawne wyniki, jednak jedynie dla stosunkowo małych instancji, ponieważ w praktyce dla dużych instancji nie wystarczy zasobów. Algorytmy oparte na niej są jednak stosunkowo proste w implementacji.

3. Algorytmy:

Dla problemu komiwojażera metoda przeglądu zupełnego sprowadza się do wygenerowania wszystkich możliwych cykli Hamiltona i wybraniu tej o najmniejszej wadze. Zaczynając od pierwszego węzła odwiedzamy jego sąsiadów (DFS - rekurencyjne) dodając ich do listy odwiedzonych, tak aby móc wyeliminować ich powtarzanie. Kiedy na liście znajdą się wszystkie węzły liczymy wagę cyklu i porównujemy z obecnie najkrótszą. Następnie cofamy się o jeden węzeł i sprawdzamy czy ma nieodwiedzonych sąsiadów (cofając się na pełnej liście, nie będzie miał, więc cofamy się o kolejny, sprawdzając jego innych sąsiadów). W ten sposób wygenerujemy wszystkie cykle.



Rysunek 1. Schemat blokowy algorytmu znajdującego rozwiązanie problemu komiwojażera, opartego na metodzie przeglądu zupełnego.

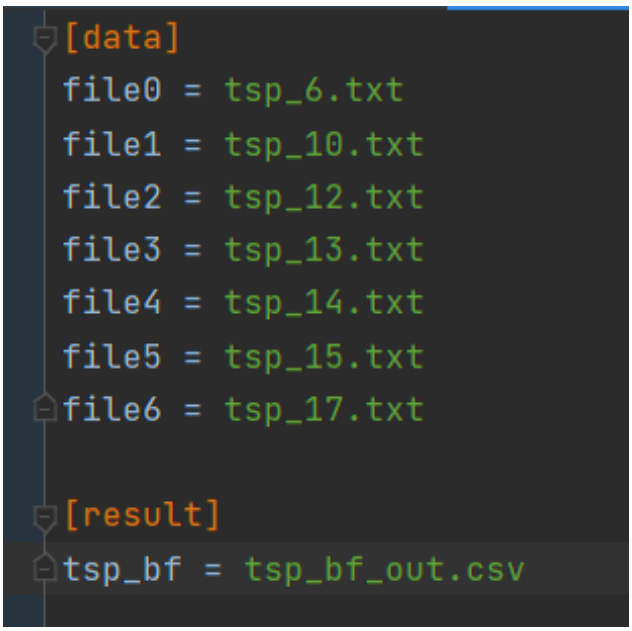
4. Dane wejściowe i wyjściowe

Co zawiera program badawczy:

- Plik wykonywalny zad1.exe
  Aby działał poprawnie musi być w jednym katalogu z plikiem config.ini i pliki tekstowe z grafami.

- Pliki tsp_6_1.txt, tsp_6_2.txt, tsp_10.txt, tsp_12.txt, itd.
  Zawierają grafy w postaci macierzy sąsiedztwa, które będziemy badać.
  Poza tymi grafami w programie wygenerowano także grafy o pośredniej ilości węzłów, dla uzupełnienia danych otrzymanych z analiz.

- Plik inicjujący config.ini



Rysunek 2. Zawartość pliku config.ini

Sekcja [data] zawiera nazwę plików wejściowych z grafami.
Sekcja [result] zawiera nazwę pliku wyjściowego

- Skrypty adjacency_matrix.py, adjacency_list.py, incidence_matrix.py:

Zawierają implementację kolejno: grafów w postaci macierz sąsiedztwa, listy sąsiedztwa i macierzy incydencji.

- Skrypty main.py, my_writer.py:

Skrypt main.py jest głównym plikiem programu, zawiera on wywołania metod z innych plików. my_writer.py zapewnia poprawny zapis do plików wynikowych.

- Plik tsp_bf_out-analiza.csv:

Zawiera zbiorcze dane i wykresy.

5. Procedura badawcza

- **Kolejność wykonywanych badań:**
    - Pobieranie danych z pliku inicjującego.
    - Wczytanie grafów z plików wejściowych
    - Uruchomienie badań dla grafów

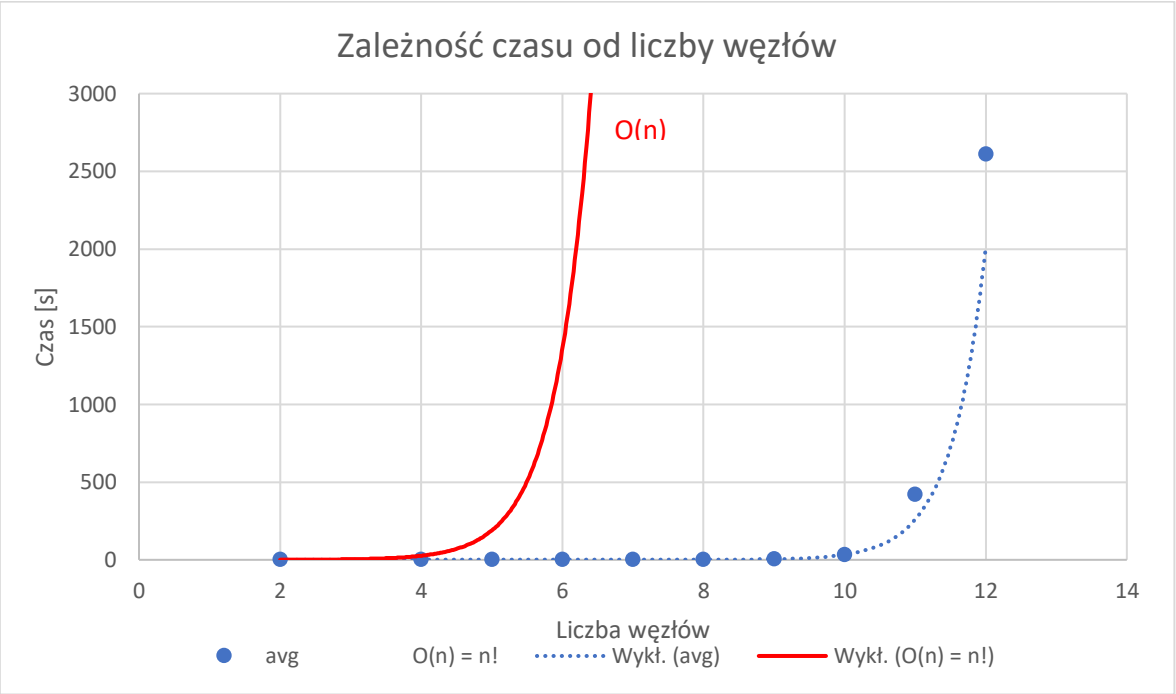
Wynikiem działania programu (zapisywanym do pliku) jest czas znalezienia rozwiązania dla każdej instancji, długość najkrótszego cyklu, oraz sam cykl (w postaci listy węzłów).

Program za każdym razem dopisuje aktualne wyniki do wyżej wymienionego pliku wyjściowego. Był uruchamiany kilka razy, aby zebrać więcej wyników. W pliku z analizami zostały wyciągnięte średnie czasu dla poszczególnych instancji, a następnie sporządzony wykres, pokazujący górne ograniczenie funkcji czasu przez O(n!).

W moim przypadku program zdołał policzyć wynik dla instancji max 12 węzłowej.

- **Specyfikacja sprzętu:**
    a. Procesor Intel i7-10510U, 1.80GHz – 2.30 GHz
    b. 16,0 GB pamięci ram
    c. System Windows 10 Home Edition

6. Wyniki



Rysunek 2. Zależność czasu pracy algorytmu w zależności od liczby węzłów instancji.

instance_size	result	time							avg	path
2	20	0	0	0	0	0	0	0,000194	2,77E-05	[0, 1, 0]
4	38	0	0	0	0	0	0	0,00023	3,29E-05	[0, 1, 2, 3, 0]
5	42	0	0	0	0	0	0	0,000765	0,000109	[0, 1, 4, 3, 2, 0]
6	58	0,015771	0,015986	0,015739	0,015673637	0,015604	0,015565	0,006937	0,014468	[0, 1, 4, 5, 2, 3, 0]
7	54	0,137625	0,093641	0,110668	0,093731403	0,104492	0,110656	0,035667	0,098069	[0, 3, 2, 5, 4, 1, 6, 0]
8	96	0,72611	0,683478	0,640484	0,579415798	0,468589	0,556713	0,321254	0,568006	[0, 1, 2, 3, 4, 5, 7, 6, 0]
9	79	2,475928	5,244052	5,772368	5,35684514	2,956686	4,710844	2,902714	4,202777	[0, 1, 2, 3, 4, 8, 5, 7, 6, 0]
10	212	29,17854	31,51299	25,40408	28,98297644	28,37647	30,44122	52,19327	32,29851	[0, 3, 4, 2, 8, 7, 6, 9, 1, 5, 0]
11	251	415,0429				430,8394	413,7388		419,8737	[0, 1, 8, 4, 6, 2, 9, 7, 5, 3, 10, 0]
12	264						2609,066		2609,066	[0, 1, 8, 4, 6, 2, 11, 9, 7, 5, 3, 10, 0]

Rysunek 3. Wyniki pomiaru czasu, znalezionych cykli dla grafów. Na żółto grafy ze strony, pozostałe to grafy własne.

instance_size	result	time	circle
6	132	0,019979	[0, 1, 2, 3, 4, 5, 0]
6	80	0,017145	[0, 5, 1, 2, 3, 4, 0]
10	212	30,44122	[0, 3, 4, 2, 8, 7, 6, 9, 1, 5, 0]
11	251	413,7388	[0, 1, 8, 4, 6, 2, 9, 7, 5, 3, 10, 0]
12	264	2609,066	[0, 1, 8, 4, 6, 2, 11, 9, 7, 5, 3, 10, 0]

Rysunek 4. Wyniki dla grafów ze strony dr Mierzwy.

7. Analiza wyników i wnioski

Algorytm skonstruowany tą metodą daje poprawne wyniki (*Rysunek 4.*), zgadzające się z tymi na stronie prowadzącego.

Tak jak się spodziewano czas dla tak skonstruowanego algorytmu (opartego o metodę brute force) rośnie wykładniczo. Jak na *Rysunku 2.* ograniczany jest od góry przez funkcję O(n!).

Nie jest to optymalna metoda dla dużych instancji. Już tutaj czas dla instancji z 12 węzłami jest dość duży w kontekście testowania programu w domowych warunkach.

Jeśli poszukuje się rozwiązania zawsze najlepszego i wielkość instancji nie jest duża, a także nie zależy nam na zasobach (mamy dużo czasu), to możemy z niej korzystać.

Dla bardziej rozbudowanych przykładów lepsze okażą się najprawdopodobniej inne metody.