Wrocław, 09.11.2019

Adrian Śnieżek
235984
Śr 17.05

Prowadzący dr inż. Jarosław Mierzwa

Sprawozdanie
Projektowanie efektywnych algorytmów
Projekt I

1. Wstęp teoretyczny

Problem komiwojażera jest to zagadnienie optymalizacyjne, którego celem jest znalezienie najkrótszej drogi między miastami, odwiedzając każde kolejne miasto dokładnie jeden raz i powrócenie do miasta początkowego.

Problem ten jest problemem NP-trudnym. Problem NP-trudny to taki problem dla którego znalezienie rozwiązania zwykle trwa a czasie ponad wielomianowym.

Do znalezienia dokładnego rozwiązania problemu zaimplementowałem dwa algorytmy:
- przegląd zupełny (brute force)
- metodę podziału i ograniczeń (branch&bound)

Oba algorytmy sprawdzają się dla niewielkich rozmiarów problemu, w tym oczywiście przegląd zupełny dla mniejszych instancji problemów niż metoda podziału i ograniczeń.

2. Złożoność obliczeniowa

Przegląd zupełny jest to algorytm, jak nazwa sugeruje, sprawdzający każda istniejąca możliwość. Wyznaczanie dróg dochodzi przez permutacje zbioru (miast) i dlatego też ten algorytm osiąga złożoność obliczeniową O(n!) gdzie „n" oznacza ilość miast

Dla algorytmu wykorzystującego metodę podziału i ograniczeń sytuacja wygląda trochę inaczej. Złożoność tego algorytmu jest uzależniona od zadanych danych. W optymistycznym przypadku gdzie cały czas będziemy poruszać się w dół jednej i tej samej gałęzi liczba wywołań rekurencji wynosi n, natomiast dla najbardziej pesymistycznego założenia złożoność obliczeniowa wyniesie O(n!) czyli dokładnie tak samo jak dla przeglądu zupełnego.

3. Opis działania programu

W programie zaimplementowałem klasę sterująca „Menu", która służy za interfejs użytkownika. Obiekt przy utworzeniu wyświetla opcje jakie użytkownik może wybrać. W zależności od wyborów wywoływane są poszczególne metody. Dostępne opcje to:

- wczytanie danych - następuje wyświetlenie plików tekstowych znajdujących się w folderze z danymi wejściowymi, użytkownik wpisuje nazwę pliku i wybrane dane zostają wczytane

- generowanie danych – użytkownik jest proszony o podanie nazwy pliku w którym dane będą zapisane, ilości miast dla których mają zostać wygenerowane dane oraz użytkownik decyduje czy losowe dane go satysfakcjonują i czy mają zostać wczytane jako aktualne

- zakładka Brute Foce – możliwość wyboru pojdyńczego testu lub serii testów oraz zapisanie wyników (drogi, kosztu i czasu) do pliku

- zakładka Branch and Bound – możliwość wyboru pojdyńczego testu lub serii testów oraz zapisanie wyników (drogi, kosztu i czasu) do pliku

- wyświetlenie aktualnych danych – wyświetlenie macierzy aktualnie załadowanych danych

- koniec – wyjście z programu

Klasa „menu" korzysta jednakże z drugiej klasy – „Read_data" , która odpowiedzialna jest za wczytanie danych z ustalonego pliku, przygotowanie danych pod konkretnie wybrany algorytm i na koniec przeprowadzenie algorytmu.

Klasa „Read_data” posiada metody takie jak:

- wypełnienie macierzy danymi z pliku

- przygotowanie danych (zmiana „0” na „-1” na przekątnej)

- przygotowanie danych pod określony algorytm

- wyliczanie granicy dla podanej macierzy

- kalkulowanie kosztu przejścia między miastami

- wykonanie danego z algorytmów

- zwrócenie informacji o wynikach z danego algorytmu

Dodatkowo w programie została stworzona klasa „Node” odpowiedzialna za przechowywanie informacji o wierzchołkach – miastach. Struktura ta przechowuje listę rodziców, koszt aktualnego wierzchołka, swój indeks, listę dzieci oraz macierz przejść.

4. Krokowe działanie algorytmu wykorzystującego metodę podziału i ograniczeń

Ustalenie miasta startowego i stworzenie wierzchołka. Wywołanie funkcji rekurencyjnej dla pierwszego wierzchołka

a) Zwiększenie iteracji wywołań funkcji

b) Zmiana wartości wierzchołka na odwiedzony

c) Generowanie możliwych dzieci dla aktualnego wierzchołka

d) Wyliczenie kosztów przejść dla każdego z dzieci

e) Umieszczenie stworzonych nowych wierzchołków w liście wierzchołków

f) Przeszukanie listy wierzchołków w celu znalezienia jeszcze nie odwiedzonego wierzchołka o najmniejszym koszcie przejścia

g) Wybranie wierzchołka i wywołanie rekurencyjne metody z argumentem wybranego wierzchołka

Algorytm kończy się w momencie znalezienia pełnej ścieżki, jeżeli koszt tej ścieżki jest najmniejszy z innych możliwie dostępnych

Przykład:

Macierz

[0, 20, 30, 31, 28, 40]
[30, 0, 10, 14, 20, 44]
[40, 20, 0, 10, 22, 50]
[41, 24, 20, 0, 14, 42]
[38, 30, 32, 24, 0, 28]
[50, 54, 60, 52, 38, 0]

Iteracja I:

Stworzenie pierwszego wierzchołka

Wierzchołek: 0
Koszt: 132
Dzieci: [1, 2, 3, 4, 5]
Rodzice: []

Wierzchołek odwiedzony

Stworzenie dzieci:

Wierzchołek: 1	Wierzchołek: 2	Wierzchołek: 3	Wierzchołek: 4
Koszt: 132	Koszt: 152	Koszt: 153	Koszt: 150
Dzieci: [2, 3, 4, 5]	Dzieci: [1, 3, 4, 5]	Dzieci: [1, 2, 4, 5]	Dzieci: [1, 2, 3, 5]
Rodzice: [0]	Rodzice: [0]	Rodzice: [0]	Rodzice: [0]

Wierzchołek: 5
Koszt: 156
Dzieci: [1, 2, 3, 4]
Rodzice: [0]

Wyszukanie kosztu minimalnego – wierzchołek 1

Iteracja II:

Aktualny wierzchołek zmieniony jako odwiedzony

Stworzenie dzieci:

Wierzchołek: 2	Wierzchołek: 3	Wierzchołek: 4	Wierzchołek: 5
Koszt: 132	Koszt: 154	Koszt: 148	Koszt: 168
Dzieci: [3, 4, 5]	Dzieci: [2, 4, 5]	Dzieci: [2, 3, 5]	Dzieci: [2, 3, 4]
Rodzice: [0, 1]	Rodzice: [0, 1]	Rodzice: [0, 1]	Rodzice: [0, 1]

Wyszukanie kosztu minimalnego – wierzchołek 2

Iteracja III:

Aktualny wierzchołek zmieniony jako odwiedzony

Stworzenie dzieci:

Wierzchołek: 3	Wierzchołek: 4	Wierzchołek: 5
Koszt: 132	Koszt: 159	Koszt: 168
Dzieci: [4, 5]	Dzieci: [3, 5]	Dzieci: [3, 4]
Rodzice: [0, 1, 2]	Rodzice: [0, 1, 2]	Rodzice: [0, 1, 2]

Wyszukanie kosztu minimalnego – wierzchołek 3

Iteracja IV:

Aktualny wierzchołek zmieniony jako odwiedzony

Stworzenie dzieci:

Wierzchołek: 4	Wierzchołek: 5
Koszt: 132	Koszt: 158
Dzieci: [5]	Dzieci: [4]
Rodzice: [0, 1, 2, 3]	Rodzice: [0, 1, 2, 3]

Wyszukanie kosztu minimalnego – wierzchołek 4

Iteracja V:

Aktualny wierzchołek zmieniony jako odwiedzony

Stworzenie dzieci:

Wierzchołek: 5
Koszt: 132
Dzieci: []
Rodzice: [0, 1, 2, 3, 4]

Wyszukanie kosztu minimalnego – wierzchołek 5

Iteracja VI:

Nieznalezienie dzieci oraz mniejszego kosztu – koniec algorytmu

5. Pomiary czasowe i środowisko pracy

Wykonywane pomiary zostały przeprowadzone na komputerze stacjonarnym o podzespołach:

- płyta główna ASUS Prime b350-plus

- procesor AMD Ryzen 7, 8 rdzeni, 3.65 GHz

- pamięć RAM CORSAIR Vengeance Lpx 16 GB 3000 MHz DDR4

Z wykorzystaniem system operacyjnego Windows 10 Education, pracując z językiem Python w środowisku programistycznym JetBrains PyCharm.

Założenia:

- w celu sprawdzenia uśrednionego czasu dokonywałem 150 wykonań algorytmu

- pierwsze 50 iteracji zostawało pomijanych pod względem czasowym

- program zlicza czas wykonań 100 ostatnich razy

- wynik jest uśredniany

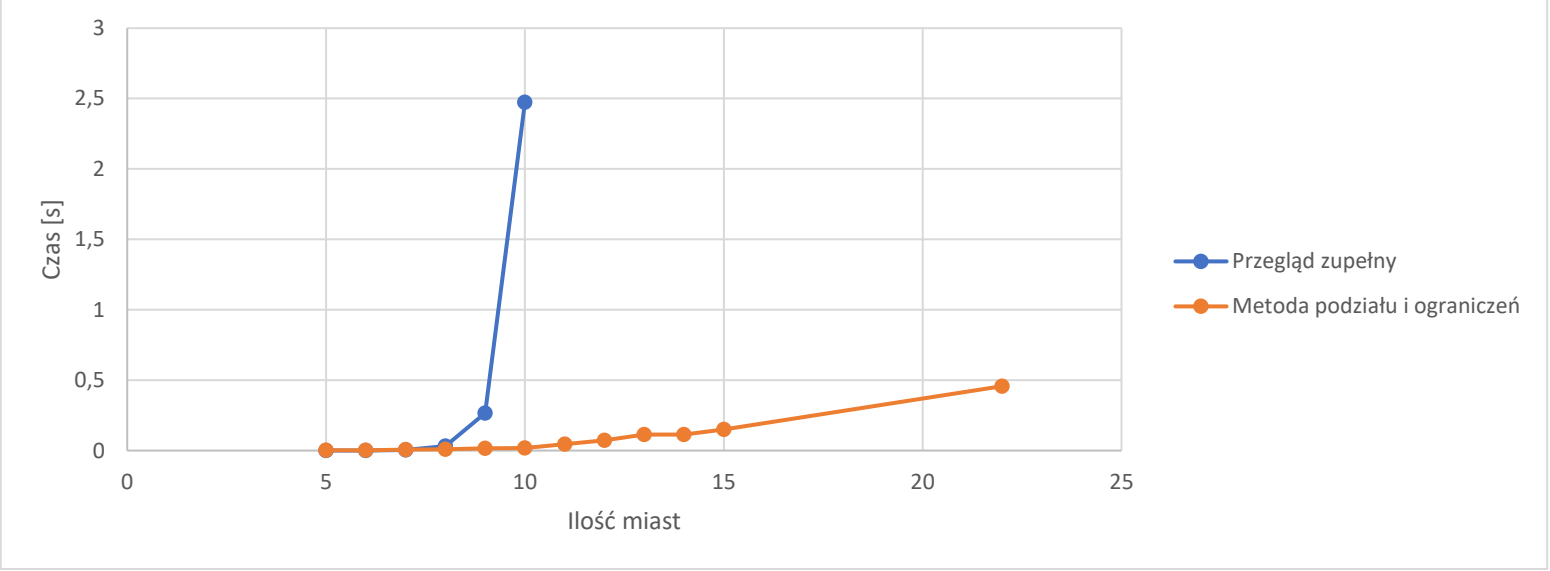
- jeżeli czas wykonywania przekraczał 5 minut test zostaje przerwany i zostaje zwrócona informacja o ilości procentowej jaka została wykonana

- ze względu na naturę przeglądu zupełnego i jego złożoność testy będą wykonywane dla mniejszych instancji problemów, a dla metody ograniczeń i podziałów na trochę większych

6. Wyniki

liczba miast	Przegląd zupełny	Metoda podziału i ograniczeń
5	0,000177437 [s]	0,001587928 [s]
6	0,000662559 [s]	0,001546962 [s]
7	0,004406065 [s]	0,006868252 [s]
8	0,031957356 [s]	0,009915214 [s]
9	0,265960846 [s]	0,015664578 [s]
10	2,472780921 [s]	0,017637079 [s]
11	25,3001819 [s]	0,044518857 [s]
12	297,7080343 [s]	0,072557191 [s]
13	-	0,112890433 [s]
14	-	0,112890433 [s]
15	-	0,150363414 [s]
22	-	0,45736632 [s]

Przedstawione wyniki w tabeli są wynikami uśrednionymi, z wyjątkiem dwóch pól.
Dla przeglądu zupełnego dla 11 miast zostało wykonane 11 iteracji w ciągu 5 minut i operacja została przerwana, a dla 12 miast została wykonana tylko jedna iteracja.



Powyższy wykres reprezentuje czas wykonywania algorytmu w zależności od ilości przeszukiwanych miast.

7. Wnioski

Porównywając oba algorytmy można zauważyć, że dla mniejszych ilości miast (5-7) algorytm przeglądu zupełnego działa szybciej. Spowodowane jest to różnicą w budowie algorytmów. Przegląd zupełny wykonuje się dysponując listą liczb z indeksami wierzchołków. Nie dokonuje redukcji macierzy i nie tworzy w czasie swojego działania innych obiektów. Algorytm metody ograniczeń i podziału wykonuje o wiele więcej operacji podczas jednorazowego wywołania rekurencyjnego. Dla wyższej ilości miast dochodzi do odwrócenia sytuacji. Pomimo większej ilości obliczeń podczas jednego wywołania, przegląd zupełny będzie zawsze musiał sprawdzić n! możliwości, zaś drugi algorytm swoich wywołań będzie miał, oczywiście w większości przypadków, zdecydowanie mniej co powoduje zauważalną różnicę czasową.

8. Bibliografia

https://en.wikipedia.org/wiki/Travelling_salesman_problem
https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/