Mateusz Krupa 256280

Projektowanie efektywnych algorytmów
Asymetryczny Problem Komiwojażera
Algorytm Branch & Bound

Spis treści

1 Wstęp teoretyczny

Problem komiwojażera (eng. TSP) to zagadnienie polegające na znalezieniu najkrótszej drogi w grafie nieskierowanym w którym znamy wszystkie wierzchołki oraz krawędzie. Dzięki tym informacjom jesteśmy w stanie wyznaczyć różne drogi w zależności o interesujące nas kryteria w projekcie skupimy się na znalezieniu najkrótszej drogi od miasta początkowego (czyli 0) poprzez wszystkie pozostałe miasta i powrót do miasta początkowego.

1.1 Algorytm Branch & Bound

Algorytm ten można przetłumaczyć na podziel i ogranicz [1]. Polega na przeszukaniu drzewa poprzez dzielenie drzewa na mniejsze podzbiory rozwiązań oraz ograniczenie czyli pominięcie gałęzi które wiemy, że nie zawierają optymalnego rozwiązania. Algorytm jest dokładny, dla przeszukiwania dużych zbiorów może być wolny oraz w zależności od implementacji potrzebować sporej ilości pamięci.

2 Opis działania algorytmu z przykładem

Dla przykładu użyjemy macierz:

$$\begin{bmatrix} -1 & 3 & 1 & 5 & 8 \\ 3 & -1 & 6 & 7 & 9 \\ 1 & 6 & -1 & 4 & 2 \\ 5 & 7 & 4 & -1 & 3 \\ 8 & 9 & 2 & 3 & -1 \end{bmatrix}$$

Pierwszym krokiem jest wyszukanie najmniejszej wartości w każdym wierszu a następnie zredukowanie macierzy o te wartości czyli o kolejno: 1,3,1,3,2.

$$\begin{bmatrix} -1 & 2 & 0 & 4 & 7 \\ 0 & -1 & 3 & 4 & 6 \\ 0 & 5 & -1 & 3 & 1 \\ 2 & 4 & 1 & -1 & 0 \\ 6 & 7 & 0 & 1 & -1 \end{bmatrix}$$

Następnie powtarzamy czynności tylko dla kolumn i dla najmniejszych wartości czyli: 0,2,0,1,0 w sytuacji gdy występuje 0 nic się nie dzieje. Poniżej przedstawiona jest już macierz po całej redukcji.

$$\begin{bmatrix} -1 & 0 & 0 & 3 & 7 \\ 0 & -1 & 3 & 3 & 6 \\ 0 & 3 & -1 & 2 & 1 \\ 2 & 2 & 1 & -1 & 0 \\ 6 & 5 & 0 & 0 & -1 \end{bmatrix}$$

Teraz sumujemy wszystkie wartości o które zminimalizowaliśmy macierz.

$$(1 + 3 + 1 + 3 + 2) + (0 + 2 + 0 + 1 + 0 = 13$$

Wynikiem działania jest minimalny droga.
Kolejnym krokiem będzie obliczenie miasta do którego mamy przejść (w tym przykładzie 0-1). Użyjemy do tego zredukowanej macierzy. Macierz modyfikujemy dodatkowo w następujący sposób, wiersz miasta odwiedzonego zamieniamy na -1 (A), kolumnę miasta które zamierzamy odwiedzić

również zamieniamy na -1(B) oraz krawędź z miasta które planujemy odwiedzić do miasta już odwiedzonego zmieniamy na -1(C).

$$A = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ 0 & -1 & 3 & 3 & 6 \\ 0 & 3 & -1 & 2 & 1 \\ 2 & 2 & 1 & -1 & 0 \\ 6 & 5 & 0 & 0 & -1 \end{bmatrix}, \; B = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ 0 & -1 & 3 & 3 & 6 \\ 0 & -1 & -1 & 2 & 1 \\ 2 & -1 & 1 & -1 & 0 \\ 6 & -1 & 0 & 0 & -1 \end{bmatrix}, \; C = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 3 & 3 & 6 \\ 0 & -1 & -1 & 2 & 1 \\ 2 & -1 & 1 & -1 & 0 \\ 6 & -1 & 0 & 0 & -1 \end{bmatrix}$$

Redukujemy macierz jak poprzednio i sumujemy o ile ją zredukowaliśmy. W tym przypadku to tylko 3. Dodatkowo dodajemy do tego minimalny aktualny koszt.

$$3 + 13 = 16$$

Wynik to minimalny koszt z miasta początkowego do miasta 1 (licząc od 0). Algorytm ten powtarzamy dla pozostałych wierzchołków i otrzymujemy.

$$0 - 1 = 16 \quad 0 - 2 = 16 \quad 0 - 3 = 18 \quad 0 - 4 = 22$$

Z otrzymanych dróg wybieramy najkrótszą drogę i powtarzamy algorytm zapamiętując minimalne drogi do tych miast, ponieważ przy następnym wybieraniu najkrótszej drogi musimy rozważyć również te wybierając najkrótszą drogę, w przypadku równych dróg wybieramy drogę prowadzącą przez większą liczbę miast. Postępujemy w ten sposób do momentu kiedy przejdziemy już przez wszystkie miasta.

3 Opis implementacji i plan eksperymentu

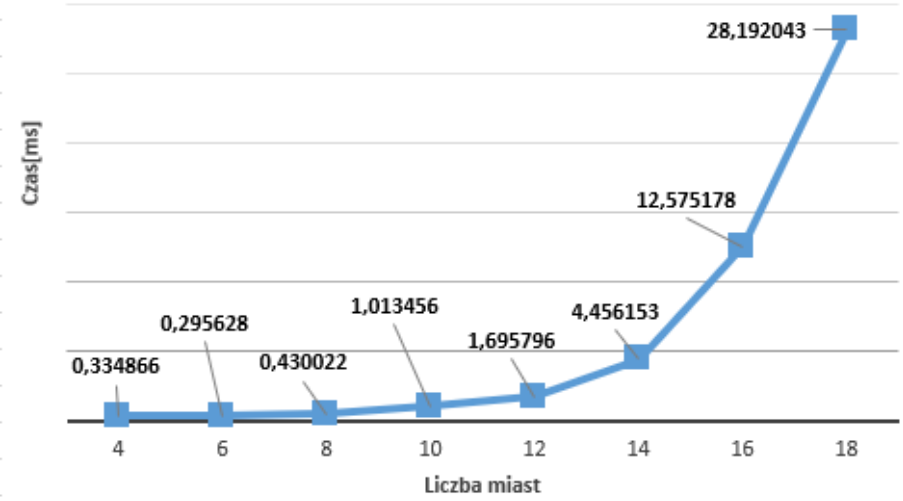
Do przechowywania wierzchołków używam kolejki priorytetowej [2]. Aby zoptymalizować zużycie pamięci przed rozpoczęciem; dodatkowo używam algorytm który szacuje drogę. Algorytm działa podobnie jak B&B jednak nie zapamiętuje wszystkich wierzchołków tylko bada aktualnie dostępne wierzchołki i wybiera najkrótszy. Na przykład dla 13 miast zamiast 161 zapamiętuje tylko 1. Do pomiaru czasu używam klasy Stopwatch która zlicza takty czasomierza w podstawowym mechanizmie czasomierza za pomocą licznika wydajności o wysokiej rozdzielczości. Liczba badanych miast to: 4,6,8,12,14,16 i 18 dla każdej instancji po 100 razy generuję losowo macierz i czas średni wykonania algorytmu wpisuję w tabelę.

4 Wyniki eksperymentu i wnioski

4.1 Testowanie

Liczba miast	Czas [ns]	Czas[ms]
4	334866	0,334866
6	295628	0,295628
8	430022	0,430022
10	1013456	1,013456
12	1695796	1,695796
14	4456153	4,456153
16	12575178	12,575178
18	28192043	28,192043

Rysunek 1: Zmierzony czas w nanosekundach oraz milisekundach



Rysunek 2: Wykres zależności czasu od ilości miast

```
Podaj ilosc miast

Ilosc: 32
Droga pokonana: 0-23-9-13-30-10-16-1-8-26-19-6-17-18-4-21-29-2-3-14-31-11-27-22-7-5-12-28-25-24-15-20-0
Minimalny koszt: 133
Czas: 23933,6336[ms]
Droga pokonana: 0-21-31-25-26-6-9-10-24-7-17-30-14-8-12-19-27-23-1-28-29-20-22-4-15-3-11-13-5-2-18-16-0
Minimalny koszt: 175
Czas: 49003,9932[ms]
Droga pokonana: 0-29-21-30-23-25-31-2-6-7-27-22-9-10-19-4-8-20-5-28-11-16-14-12-26-18-13-15-17-24-1-3-0
Minimalny koszt: 193
Czas: 52862,2135[ms]
Droga pokonana: 0-12-30-20-2-4-3-9-14-5-16-21-7-29-26-19-10-23-31-22-17-18-8-1-6-28-13-27-15-11-25-24-0
Minimalny koszt: 160
Czas: 9367,2026[ms]
Droga pokonana: 0-4-27-9-24-11-15-16-25-2-18-23-5-30-12-31-10-13-20-6-19-8-26-17-29-21-3-14-28-7-1-22-0
Minimalny koszt: 150
Czas: 1938,8498[ms]
Droga pokonana: 0-14-11-25-29-20-12-31-22-2-4-10-27-17-7-6-28-3-30-26-1-13-16-21-19-5-24-8-18-9-15-23-0
Minimalny koszt: 158
Czas: 2218,3152[ms]
Droga pokonana: 0-1-7-18-26-12-19-24-29-11-13-30-20-23-15-16-2-28-3-25-4-10-17-9-22-5-21-31-27-14-6-8-0
Minimalny koszt: 157
Czas: 16629,2189[ms]
Droga pokonana: 0-25-21-11-18-15-16-2-1-23-8-22-3-9-14-30-17-29-10-31-6-20-27-5-12-26-24-7-13-4-28-19-0
Minimalny koszt: 155
Czas: 10786,8551[ms]
Droga pokonana: 0-23-28-1-20-14-27-19-6-11-16-7-13-31-2-24-12-15-3-25-22-4-17-30-21-29-9-26-18-5-8-10-0
Minimalny koszt: 154
Czas: 557,4529[ms]
Droga pokonana: 0-11-6-25-19-10-18-2-5-16-9-1-22-29-15-8-4-28-20-17-3-26-13-21-7-27-12-30-24-31-23-14-0
Minimalny koszt: 136
Czas: 27051,8489[ms]
```

Rysunek 3: Droga, minimalny koszt oraz czas [ms] dla losowo wygenerowanych 32 miast

4.2 Wnioski

Dla badanych wielkości instancji algorytm sprawnie radził sobie z wyznaczaniem najkrótszej drogi. Z wykresu wynika, że algorytm jest zależny od wielkości instancji ponieważ za każdym razem dla interesujących nas wierzchołków musimy zredukować macierz i wyliczyć minimalny koszt. Podczas testów łatwo było zauważyć, że również od złożoności problemu i nawet dla mniejszych instancji z powodu braku pamięci nie jesteśmy w stanie otrzymać rozwiązania. Podczas testowania wiedząc o ograniczeniach pamięci sprawdziłem dla jakiej instancji algorytm jest w miarę sprawnie wyliczyć rozwiązanie, (rysunek 3) dla mojego komputera to 32 miasta, przy większych instancjach również był w stanie wyliczyć rozwiązanie jednak już dużo częściej zdarzało się że nie dysponuję wystarczającą ilością pamięci.

5 Literatura

[1] $https://www.ii.uni.wroc.pl/\ prz/2011lato/ah/opracowania/met_podz_ogr.opr.pdf$
[2] $https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp$