Projektowanie efektywnych algorytmów
projekt nr 1

Piotr Józefek 272311
grupa 8 poniedziałek, 15:15 - 16:55

Wprowadzenie

Problem komiwojażera (TSP) jest zagadnieniem optymalizacji kombinatorycznej, w którym zadaniem jest znalezienie najkrótszej możliwej ścieżki odwiedzającej każde z zadanych miast dokładnie raz i powracającej do miasta początkowego.

TSP znajduje zastosowanie w różnych dziedzinach, takich jak logistyka, robotyka, oraz optymalizacja tras w telekomunikacji, co czyni go jednym z częściej analizowanych problemów NP-trudnych.

Złożoność przeglądu zupełnego: $O(n!)$

Złożoność programowania dynamicznego: $O(n^2 \cdot 2^n)$

Opis implementacji algorytmu

Przegląd zupełny

- int** tab - macierz sąsiedztwa przechowująca wagi krawędzi pomiędzy wierzchołkami, gdzie każdy element tab[i][j] reprezentuje koszt przejścia między wierzchołkiem i oraz j.
- int* path - tablica, która przechowuje najlepszą ścieżkę znalezioną w danym momencie, czyli kolejność wierzchołków z minimalnym kosztem.
- int* word - pomocnicza tablica przechowująca aktualnie generowaną permutację wierzchołków do wyznaczania ścieżek komiwojażera.

Programowanie dynamiczne

- int** tab - macierz sąsiedztwa przechowująca koszty przejść między wierzchołkami; element tab[i][j] reprezentuje koszt przejścia z wierzchołka i do j.
- vector<vector<int>> dp - tablica dynamiczna przechowująca minimalne koszty dotarcia do wierzchołków z określonymi maskami odwiedzonych wierzchołków.

- vector<vector<int>> parent - tablica dynamiczna do śledzenia poprzednich wierzchołków w optymalnej ścieżce
- int* path - tablica, która przechowuje najlepszą ścieżkę znalezioną w danym momencie

Przykład praktyczny

Programowanie dynamiczne Helda-Karpa

Przykładowe miasta
0: Miasto A
1: Miasto B
2: Miasto C
3: Miasto D

Macierz wag

	0	1	2	3
0	0	10	15	20
1	5	0	35	25
2	10	30	0	15
3	20	25	30	0

Krok 1: Inicjalizacja

Inicjalizujemy dp[1][0] na 0, co oznacza, że koszt podróży zaczynając od miasta A (wierzchołek 0) i odwiedzając tylko to miasto wynosi 0. Tablica parent jest również początkowo pusta.

Krok 2: Przeglądanie masek

Algorytm wykorzystuje maski bitowe do reprezentowania zestawu odwiedzanych wierzchołków. W przypadku czterech wierzchołków (N=4) mamy 16 różnych masek (od 0 do 15).

Główna pętla (Obliczanie minimalnych kosztów)

1. Maska 1 (0001): Tylko wierzchołek 0 (Miasto A) odwiedzony.

    - Brak obliczeń.
2. Maska 3 (0011): Wierzchołki 0 i 1 (Miasto A i B).
    - Obliczamy koszt podróży z 0 do 1:
        - dp[3][1] = dp[1][0] + tab[0][1] = 0 + 10 = 10.
        - Ustawiamy parent[3][1] = 0.
3. Maska 5 (0101): Wierzchołki 0 i 2 (Miasto A i C).
    - Obliczamy koszt:
        - dp[5][2] = dp[1][0] + tab[0][2] = 0 + 15 = 15.
        - Ustawiamy parent[5][2] = 0.
4. Maska 7 (0111): Wierzchołki 0, 1 i 2 (Miasto A, B i C).
    - Obliczamy koszt przejścia z 1 do 2:
        - dp[7][2] = dp[3][1] + tab[1][2] = 10 + 35 = 45.
        - Ustawiamy parent[7][2] = 1.
    - Obliczamy koszt przejścia z 2 do 1:
        - dp[7][1] = dp[5][2] + tab[2][1] = 15 + 30 = 45.
        - Ustawiamy parent[7][1] = 2.
5. Maska 15 (1111): Wszystkie wierzchołki (A, B, C, D).
    - Obliczamy koszt powrotu do 0 z każdego wierzchołka (B, C, D).

Krok 3: Obliczenie minimalnego kosztu powrotu

W ostatniej pętli, dla każdego wierzchołka końcowego, obliczamy całkowity koszt powrotu do A:

- Dla B: cost = dp[15][1] + tab[1][0]
- Dla C: cost = dp[15][2] + tab[2][0]
- Dla D: cost = dp[15][3] + tab[3][0]

Zapisujemy minimalny koszt i ostatni wierzchołek.

Krok 4: Odtwarzanie ścieżki

Na podstawie tablicy parent, odtwarzamy ścieżkę w odwrotnej kolejności, zaczynając od ostatniego wierzchołka, aż dotrzemy do wierzchołka startowego.

Wyniki:

- Minimalny koszt: 80
- Ścieżka: A → B → D → C → A

Plan eksperymentu

Założeniem eksperymentu jest przeprowadzenie symulacji algorytmów rozwiązujących problem komiwojażera.
Przyjęte rozmiary to: 8, 9 , 10, 11, 12, 13, 14

Dla dokładności pomiarów każdy algorytm został wykonany po 100 razy, wyjątkiem było przeszukiwanie zupełne które z powodu długiego czasu wykonywania było wykonane 20 razy .

Do mierzenia czasu została wykorzystana biblioteka <chrono> oraz funkcja high_resolution_clock. Dla każdego algorytmu został zapisany czas rozpoczęcia oraz zakończenia działania algorytmu. Na ich podstawie został obliczony średni czas wykonania.

Generowanie polega na operacjach na macierzy sąsiedztwa, a następnie przypisaniu odpowiednich wartości do struktur grafu:

- Macierz sąsiedztwa zawierająca liczbę wierzchołków jest tworzona dynamicznie, gdzie każdy element tablicy tab[i][j] oznacza wagę krawędzi między wierzchołkami i i j. Dla krawędzi, które nie istnieją przypisana jest wartość -1, natomiast pozostałym krawędziom losowana jest waga z zadanego przedziału.

- Struktury ścieżki obejmują tablice path oraz finalpath, które przechowują odpowiednio ścieżkę oraz wynik końcowy algorytmu. Struktury te są dynamicznie inicjalizowane po usunięciu poprzednich danych.

Na początku generowane są losowe wartości wag krawędzi pomiędzy wierzchołkami, po czym przypisywane są do odpowiednich miejsc w macierzy sąsiedztwa, aby utworzyć pełny graf z losowymi kosztami przejść między wierzchołkami. Została wykorzystana do tego klasa mt19937(generator) i uniform_real_distribution z biblioteki <random>.

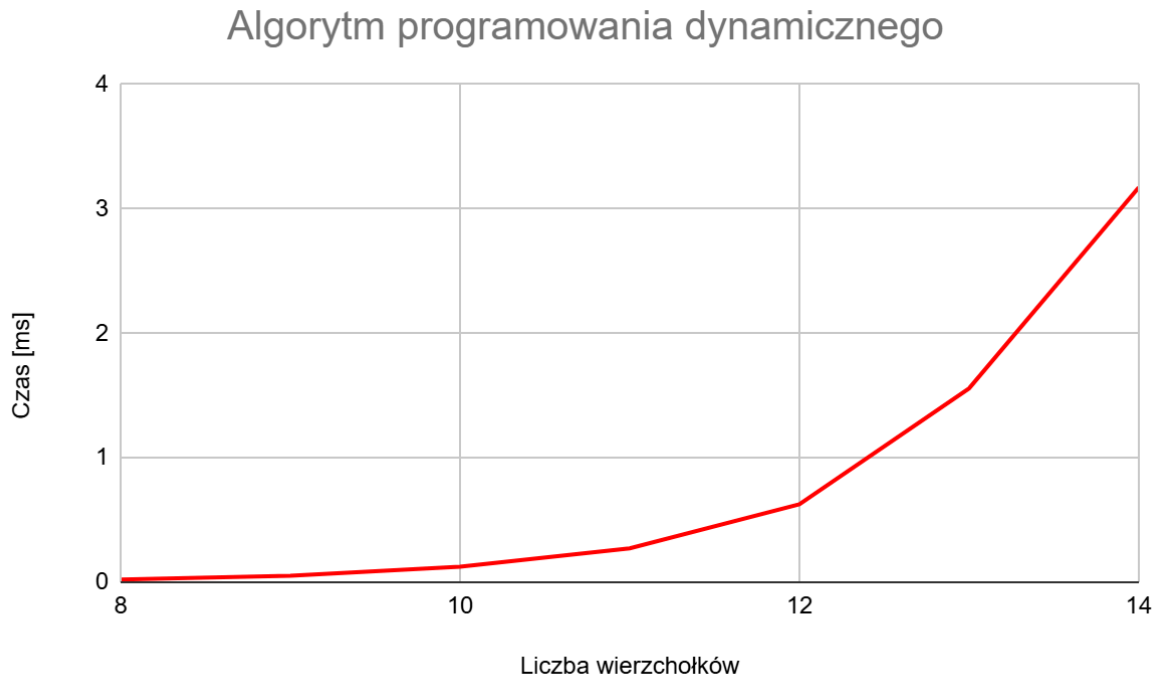
Wyniki eksperymentu

Liczba wierzchołków	Czas trwania przeglądu zupełnego [ms]
8	0.61615
9	5.8225
10	59.7585
11	676.332
12	8393.23
13	115296
14	1773710

Liczba wierzchołków	Czas trwania programowania dynamicznego [ms]
8	0.0214
9	0.05078
10	0.12365
11	0.27185
12	0.62564
13	1.55733
14	3.17156

Przegląd zupełny

Programowanie dynamiczne



Wnioski:

Literatura:

- http://algorytmy.ency.pl/artykul/problem_komiwojazera
- http://algorytmy.ency.pl/artykul/algorytm_helda_karpa
- https://www.erainformatyki.pl/programowanie/problem-komiwojazera.html
- https://en.wikipedia.org/wiki/Travelling_salesman_problem