POLITECHNIKA WROCŁAWSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Projektowanie efektywnych algorytmów

Projekt 1

Implementacja i analiza efektywności algorytmu programowania dynamicznego dla problemu komiwojażera (TSP)

Termin zajęć: Czwartek, 15:15

Autorka:
Daria Jeżowska, 252731

Prowadzący zajęcia:
mgr inż. Antoni Sterna

Zadanie projektowe

Problem komiwojażera polega na znalezieniu minimalnego cyklu Hamiltona, czyli najkrótszej drogi pomiędzy wszystkimi wierzchołkami grafu, przy czym każdy wierzchołek może zostać odwiedzony tylko jeden raz i podróż musi się skończyć w punkcie startowym - rozwiązaniem problemu komiwojażera jest cykl o najmniejszej sumie wag krawędzi. Problem ten został sformułowany w 1859 roku i jest to NP-trudny problem, ważny informatyce teoretycznej.

Założenia projektowe

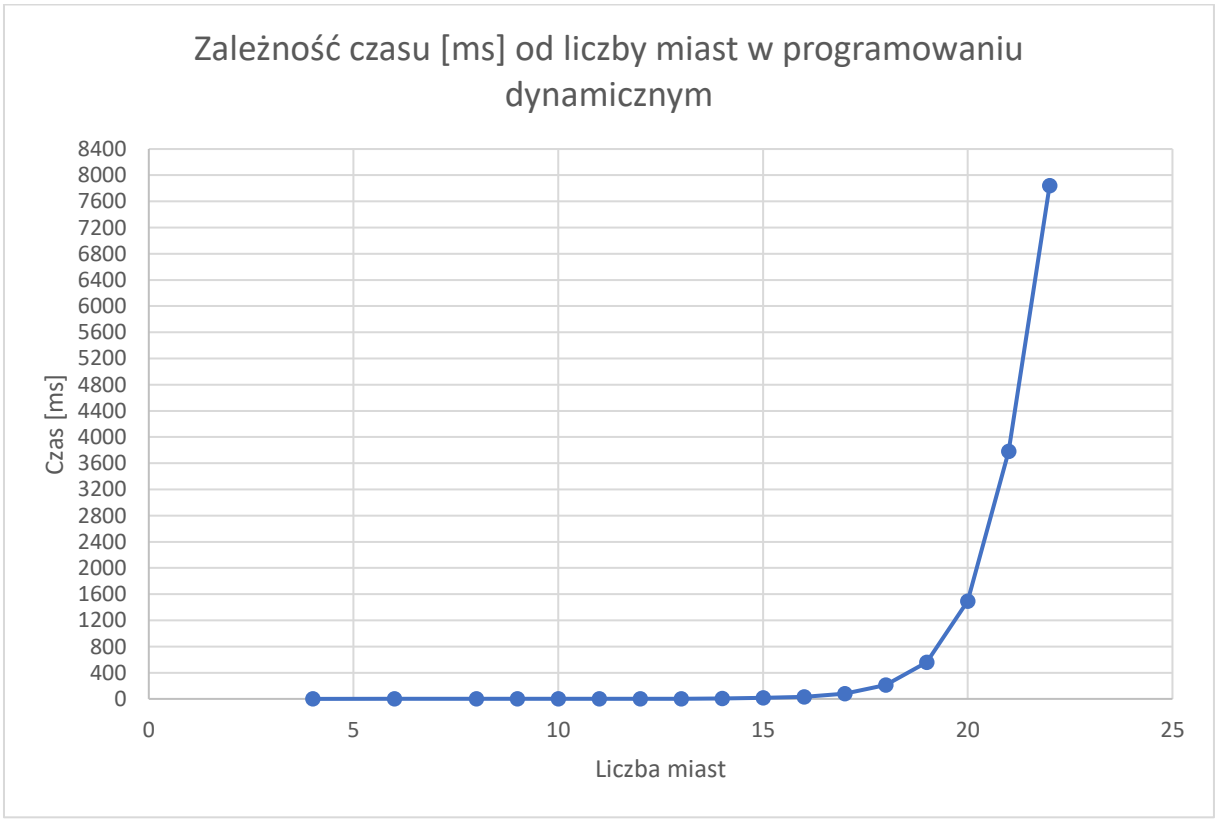
- używane struktury danych powinny być alokowane dynamicznie (w zależności od aktualnego rozmiaru problemu),
- program powinien umożliwić weryfikację poprawności działania algorytmu (wczytanie danych wejściowych z pliku tekstowego),
- po zaimplementowaniu i sprawdzeniu poprawności działania algorytmu należy
- dokonać pomiaru czasu jego działania w zależności od rozmiaru problemu *N* (badania należy wykonać dla minimum 7 różnych reprezentatywnych wartości *N*),
- dla każdej wartości *N* należy wygenerować po 100 losowych instancji problemu (w sprawozdaniu należy umieścić tylko wyniki uśrednione),
- implementacje algorytmów powinny być zgodne z obiektowym paradygmatem programowania,
- używanie „okienek" nie jest konieczne i nie wpływa na ocenę (wystarczy wersja konsolowa),
- kod źródłowy powinien być komentowany,

Algorytm

Aby rozwiązać ten problem został użyty algorytm Helda-Karpa, oparty na programowaniu dynamicznym. Złożoność pamięciowa tego problemu to $O(2^n n)$, a złożoność czasowa to $O(2^n n^2)$.
Metoda ta jest rekurencyjna i polega na znalezieniu najkrótszej ścieżki w danym podzbiorze – czyli rozpatrzeniu wszystkich podzbiorów zaczynając od najmniejszego. Przy rozwiązywaniu używamy problemów już wcześniej obliczonych. W implementacji są wykorzystywane maski bitowe, które usprawniają tworzenie podzbiorów dla określonego problemu. Przykładowo: dla zmiennej *visitedSoFar*, która przechowuje odwiedzone do tej pory wierzchołki wartość 13, czyli 1101 będzie oznaczać, że odwiedziliśmy już wierzchołki 0, 2 i 3 (2^0,2^2, 2^3). Dzięki operacjom bitowym możemy sprawdzić czy odwiedziliśmy już dany podzbiór czy nie.

Wyniki

Liczba miast	Czas [ms]
4	0,0006
6	0,0061
8	0,0348
9	0,0812
10	0,2128
11	0,4578
12	1,0042
13	2,5431
14	5,4202
15	15,2118
16	32,6944
17	79,992
18	211,893
19	559,317
20	1493,49
21	3782,31
22	7840,84



Wnioski

Dla każdego rozmiaru problemu wykonanych zostało 100 testów, a wyniki widoczne w tabelce i na wykresie są ich uśrednieniem. Jak widać czas rośnie dosyć gwałtownie, co by się zgadzało ze złożonością oczekiwaną $O(2^n n^2)$. Dla 22 danych średni czas to ~8 sekund, więc dla stu testów zajęło to ponad 13 minut. Patrząc na to, jak szybko rośnie czas wykonywania algorytmu, przetestowanie go dla większej ilości danych zajęłoby bardzo dużo czasu. Dodatkowo dla każdego miasta przechowujemy informację o zaliczonych wierzchołkach, gdzie są to wartości rzędu 2^n – dla kilkudziesięciu miast zajęłoby to znacznie więcej pamięci.

Źródła

- Travelling Salesman Problem – Dynamic Programming
- Held-Karp algorithm
- Dynamic programming
- Wykłady