Sprawozdanie:

Projektowanie Efektywnych Algorytmów

Dmytro Verkhovsky 231844

**Prowadzący:** dr inż. Dariusz Banasiak

# Zadanie projectowe numer 1

# Problem Komiwojazera dla programowania dynamicznego

13.11.2017

## Wstęp

* 1. Opis zadania projektowego

Należy zaimplementować algorytm programowania dynamicznego dla problemu komiwojażera oraz dokonać testów polegających na pomiarze czasu działania algorytmy w zależności od wielkości.

1.2 Wstęp teoretyczny

1.2.1 Problem komiwojażera

Problem komiwojażera jest zagadnieniem optymalizacyjnym, polegającym na znalezieniu minimalnego cyklu Hamiltona w pełnym graﬁe ważonym. Nazwa pochodzi od typowej ilustracji problemu, przedstawiającej go z punktu widzenia wędrownego sprzedawcy (komiwojażera): dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość / cena podróży / czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej / najtańszej / najszybszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej się w określonym punkcie.

1.2.2 Programowanie dynamiczne

Programowanie dynamiczne jest techniką lub strategią projektowania algorytmów, stosowaną przeważnie do rozwiązywania zagadnień optymalizacyjnych. Jest alternatywą dla niektórych zagadnień rozwiązywanych za pomocą algorytmów zachłannych. Programowanie dynamiczne opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów. Zagadnienia odpowiednie dla programowania dynamicznego cechuje to, że zastosowanie do nich metody siłowej (ang. brute force) prowadzi do ponadwielomianowej liczby rozwiązań podproblemów, podczas gdy sama liczba różnych podproblemów jest wielomianowa. Ponieważ jednak uzyskiwany algorytm zazwyczaj wymaga pamięci (i czasu) proporcjonalnego do iloczynu maksymalnych wartości wszystkich parametrów, stosowanie większej ilości parametrów niż 3-4 rzadko bywa praktyczne. Klucz do zaprojektowania algorytmu tą techniką leży w znalezieniu równania rekurencyjnego opisującego optymalną wartość funkcji celu dla danego problemu jako funkcji optymalnych wartości funkcji celu dla podproblemów o mniejszych rozmiarach. Programowanie dynamiczne znajduje optymalną wartość funkcji celu dla całego zagadnienia, rozwiązując podproblemy od najmniejszego do największego i zapisując optymalne wartości w tablicy. Pozwala to zastąpić wywołania rekurencyjne odwołaniami do odpowiednich komórek wspomnianej tablicy i gwarantuje, że każdy podproblem jest rozwiązywany tylko raz. Rozwiązanie ostatniego z rozpatrywanych podproblemów jest na ogół wartością rozwiązania zadanego zagadnienia. Niejednokrotnie stosowanie techniki programowania dynamicznego daje w rezultacie algorytm pseudowielomianowy. Programowanie dynamiczne jest jedną z bardziej skutecznych technik rozwiązywania problemów NP-trudnych. Niejednokrotnie może być z sukcesem stosowana do względnie dużych przypadków problemów wejściowych, o ile stałe występujące w problemie są stosunkowo nieduże. Na przykład, w przypadku dyskretnego zagadnienia plecakowego jako parametry dynamiczne w metodzie programowania dynamicznego należy przyjąć rozmiar kolejno rozpatrywanych podzbiorów elementów oraz rozmiar plecaka, zmieniający się od 0 do wartości B danej w problemie.

*Metoda programowania dynamicznego* została opracowana przez *Richarda Bellmana* w połowie dwudziestego wieku. Metoda programowania dynamicznego ma zastosowanie do rozwiązywania tzw. problemów bez pamięci, spełniających *własność Markowa*(Mówimy, że wieloetapowy proces decyzyjny ma *własność Markowa*, jeżeli po dowolnej liczbie decyzji, np.k, wpływ pozostałych etapów procesu decyzyjnego na wartość funkcji celu f zależy tylko od stanu procesu przy końcu k-tego etapu i od decyzji następnych)

### **Zasada optymalności Bellmana**

Dla wieloetapowego procesu decyzyjnego z własnością Markowa strategia

optymalna ma tę własność, że jakikolwiek byłby stan początkowy i decyzja

początkowa, pozostałe decyzje muszą tworzyć strategię optymalną z punktu

widzenia stanu wynikłego z pierwszej decyzji.

Klasycznym problemem dotyczącym DP dla podzbiorów jest znany **problem komiwojażera** (TSP).

Ten problem jest klasycznym problemem NP-zupełnym, i ma złożonośc *O*((2^*N)*∗(*N^*2)).

1.2.3 Maski bitowe

Maska bitowa - specyficzne dane które urzywamy do maskowania - wybór pojedynczych bitów lub pól z kilku bitów z binarnego ciągu lub liczby.

Operacje te działają na ciągach bitowych w ten sposób, że bity na odpowiadających sobie miejscach są poddawane „standardowym” operacjam logicznym - koniunkcji (AND), alternatywie (OR) i alternatywie wykluczającej (XOR), gdzie :

- AND to &

Jest to operacja dwuargumentowa. W językach programowania lub w pseudokodzie zapisujemy ją tak:

**A** *and* **B** lub **a** *&* **b**  
Wynikowy bit jest ustawiany na 1 tylko wówczas gdy obydwa bity argumentów ustawione są na 1.

- OR to |

Jest to operacja dwuargumentowa W językach programowania lub w pseudokodzie zapisujemy ją tak:

**A** *or* **B** lub **A** ***|*** **B**  
Wynikowy bit jest ustawiany na 1 tylko wówczas gdy przynajmniej jeden bit jakiegoś z argumentów ustawiony jest na 1.

|  |
| --- |
|  |

- XOR to ^

Jest to operacja dwuargumentowa, zwana też alternatywą wykluczającą W językach programowania lub w pseudokodzie zapisujemy ją tak:

**A** *xor* **B**  lub **A** *^* **B**  
Wynikowy bit jest ustawiany na 1 tylko wówczas gdy dokładnie jeden bit jakiegoś z argumentów ustawiony jest na 1.

* 1. Rozpatrywanie algorytmu dla konkretnego pliku

| 0 49 79 |

| 60 0 91 |

| 87 8 0 |

Powyższa matryca przedstawia problem dla trzech miast.

W rozwiązaniu :

V= zbiór wszysktich wierzchołków

A=podzbiór V

=rozmiar najkrótszej drogi z do przechodzącej przez każdy wierzchołek A dokładnie raz

Zatem w przykładzie V={- reprezentuje zbiór,

[] -reprezentuje drogę

Zbiór V-{} zawiera wszystkie wierzchołki oprócz oraz i ma zastosowanie zasada optymalności , możemy stwierdzić:

Długość trasy minimalnej = minimum(D[)

2<=j<=n

I ogólnie dla i!=1 oraz nie należącego do A

D [=minimum(D[]+ W[i][j]) jeżeli A!=Ø

D Ø][ W[1][j]

Określmy optymalną trasę dla grafu z przykładu.

D[ Ø][]=49;

D Ø][=79;

Teraz rozważamy wszystkie zbiory zawierające jeden elemnt:

D[ ]= 49+91=140

Podobnie:

D[ ]= 79+8=87;

Na końcu liczymy długość optymalnej trasy:

D[{}][]=min(W[2][1]+ D[ ],W[3][1]+ D[ ])=min(147,227)=147.

Minimalny koszt 147.

Optymalna scziezka znalieżona.

1->3->2->1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Wirzchołek koncowy | | |
|  | P | 0[001] | 1[010] | 2[100] |
| zbior wierzchołków | 1[001] | 0 | 9999 | 9999 |
| 2[010] | 9999 | 9999 | 9999 |
| 3[011] | 9999 | 49 | 9999 |
| 4[100] | 9999 | 9999 | 9999 |
| 5[101] | 9999 | 9999 | 79 |
| 6[110] | 9999 | 9999 | 9999 |
| 7[111] | 9999 | 87 | 140 |

W pierwszy podzbiór ustawiamy 0 przy stworzeniu zbiorów.W drugi podzbiór nie wchodzimy bo nie rozpoczyna się od pierwszego wierzchołka.W trzeczim podzbiórze mamy współbieżność na pierwszym i drugim bitu. Nic nie wpisujemy w pierwszym bo żadna liczba dodatnia nie jest mniejsza od 0. Też mamy w tym podzbiórze współbieżność na drugim bitu. Wyszykujemy najmniejszy element i wpisujemy jego do tej komórki. W czwarty podzbiór nie wchodzimy jak i w drugi bo nie rozpoczyna się od pierwszego wierzchołka. W piątym podzbiorze mamy współbieżność na pierwszym i ostatnim bitu. Nic nie wpisujemy w pierwszym, jak poprzednio, bo żadna liczba dodatnia nie jest mniejsza od 0. Też mamy w tym podzbiórze współbieżność na ostatnim bitu. Wyszykujemy najmniejszy element i wpisujemy jego do tej komórki. Szósty podzbiór nie zaczyna się od pierwszego wierszhołka przez to nie wchodzimy do niego. I ostatni, śiódmy podzbiór, ma współbieżność na wszystkich bitach. Jak i poprzednio nie wchodzimy na pierwszy. Za tym wyszukujemy najmniejsze wartośći w drugim i trzecim bicie odpowiednio.

Po wykonaniu tej częśći algorytmu wykonujemy wyszukiwanie najmnejszego kosztu szceżki między wszystkimi miastami. W ostatnim podzbiórze znajdują się sumy kosztów przechodzenia między miastami opróć wrocenia w początkowe miasto. Dodajemy odpowidni koszt i szukamy najmniejszy z tych kosztów,

który i zapisujemy jako minimalny koszt.

Ostatnia część naszego algorytmu to wyszukiwanie minimalnej szieżki. W tej częśći algorytmu już po wybraniu minimalnego kosztu znajdujemy i wpisujemy minimalnu ścieżku. Patszymy na to jak my otrzymali konkretny element z sumy dwóch poprzednich operacij i kiedy elemnt i suma dwóch poprzednich elementów jest równa to rozpatrujemy już poprzedni element. I tak do tego czasu póki nam nie zastaną same licby, które nie stworzone z podzbiuru.

**2. Plan eksperymentu**

- Czasy poszczególnych operacji mierzone są przy pomocy funkcji zawartych w bibliotece **<chrono>** będącej częścią standardu C++11.

- Będziemy też wczytywać już przygotowane dane z strony zamieszczonej w pliku.

-Test jaki mierzy czas dla rożnej ilości miast jest przeprowadzony 100 dla konkretnego rozmiaru miasta.

**3. Wyniki eksperymentu**

Wyniki czasowe dla roznych ilosci miast

|  |  |
| --- | --- |
| ilość miast | Czas,s |
| 2 | 1,48E-05 |
| 7 | 5,02E-05 |
| 13 | 0,004738 |
| 17(br17.atsp) | 0,110605 |
| 20 | 1,123475 |
| 21(gr17.tsp) | 3,967244 |
| 22 | 5,444132 |

Pierwszy wykres zroniony dla wszystkich danych, ale na nim nie są widoczne skoki czasowe dla małych liczb miast. Dwa poniższych wykresy to jest podzieliony górny żeby te skoki czasowe były widoczniejsze.

|  |  |
| --- | --- |
| ilość miast | Czas,s |
| 2 | 1,48E-05 |
| 7 | 5,02E-05 |
| 13 | 0,004738 |

|  |  |
| --- | --- |
| ilość miast | Czas,s |
| 13 | 0,004738 |
| 17(br17.atsp) | 0,110605 |
| 20 | 1,123475 |
| 21(gr17.tsp) | 3,967244 |
| 22 | 5,444132 |

|  |  |
| --- | --- |
| ilość miast | Czas,s |
| 17(gr17.tsp) | 0,208331 |
| 20 | 1,123475 |
| 21(gr17.tsp) | 3,967244 |
| 22 | 5,444132 |

4. Wnioski

W tym projekcie zostały wykonane pomiary dla algorytmu wykonanego za pomoca programownia dynamicznego dla problemu Komiwojazera .

Na wykresach jakie zostały przedstawione można zobaczyć różnicę pomiędzy czasem wykonania algorytmów dla roznej ilosci miast, czym wiecej miast tym dłuzej robi sie program, bo potrzebuje wiecej pamieci do przechowywania danych. Należy pamiętać o tym że mogą wynikać rozbieżności powiązane z niedokładności pomiaru czasu w systemie Windows, różnej zajętości procesora podczas wykonywania algorytmów.