Projektowanie Efektywnych Algorytmów

Projekt 1: Rozwiązanie problemu komiwojażera przy pomocy programowania dynamicznego

Grupa:

Czwartek 17:05

Prowadzący:

Antoni Sterna

Kevin Pieprzak

225983

# Wstęp

Celem projektu było wykonanie programu, wykorzystującego algorytmy programowania dynamicznego do rozwiązania problemu komiwojażera.

# Specyfikacja techniczna

* Program wykonany w języku C++
* Struktury przechowujące dane alokowane są dynamicznie, zależnie od rozmiaru problemu
* Program posiada możliwość wczytywania danych z pliku, w celu weryfikacji poprawności
* Algorytmy zostały zaimplementowane zgodnie z paradygmatami programowania obiektowego
* Czas wykonania algorytmu mierzony był z dokładnością do mikrosekund, przy wykorzystaniu bibliotek systemowych

# Analiza problemu

Problem komiwojażera należy do klasy problemów NP-trudnych. Jest to optymalizacyjny problem, rozwiązaniem którego jest znalezienie minimalnego cyklu Hamiltona (ścieżki prowadzącej przez wszystkie wierzchołki grafu, powracając na końcu do wierzchołka początkowego) w grafie pełnym ważonym. W wersji asynchronicznej, odległości pomiędzy wierzchołkami mogą dodatkowo zależeć także od kierunku przejścia pomiędzy nimi. Główną trudnością w rozwiązaniu problemu jest znacząca liczba możliwych kombinacji.

# Opis algorytm

Programowanie dynamiczne (ang. dynamic programming) jest metodą rozwiązywania złożonych problemów, poprzez rozbicie ich na zbiór podproblemów o mniejszej złożoności, przy założeniu, że każdy podproblem rozważany jest jedynie raz, a wynik jego analizy przechowywany jest do wykorzystania w późniejszych obliczeniach. Dla problemu komiwojażera, najlepszym algorytmem wykorzystującym tę metodę, jest algorytm Helda-Karpa, posiadający złożoność czasową *O(n2 \* 2n)*. Jego wykonanie prezentuje się następująco:

1. Określamy funkcję koszt(S, p), gdzie S jest zbiorem wierzchołków, a p punktem kończącym ścieżkę, w następujący sposób:

* Jeżeli S = 1, to koszt(S, p) = wadze krawędzi określonej w macierzy jako d(0, p)
* Jeżeli S > 1, to koszt(S, p) = minx (koszt(S-{p}, x) + d(x, p))

1. Stosując ten algorytm, dla przykładowego zbioru czterech miast, w pierwszej iteracji otrzymamy:

* koszt({1}, 1) = d(0, 1)
* koszt({2}, 2) = d(0, 2)
* koszt({3}, 3) = d(0, 3)

1. Następnie, należy zacząć rozważać zbiory 2-elementowe, stosując ten sam wzór:

* koszt({1, 2}, 1) = min(koszt({2}, 2) + d(2, 1))
* koszt({1, 2}, 2) = min(koszt({1}, 1) + d(1, 2))
* koszt({1, 3}, 1) = min(koszt({3}, 3) + d(3, 1))
* koszt({1, 3}, 3) = min(koszt({1}, 1) + d(1, 3))
* koszt({2, 3}, 2) = min(koszt({3}, 3) + d(3, 2))
* koszt({2, 3}, 3) = min(koszt({2}, 2) + d(2, 3))

1. Kolejno, dla zbiorów 3-elementowych:

* koszt({1, 2, 3}, 1) = min(koszt({2, 3}, 2) + d(2, 1), koszt({2, 3}, 3) + d(3, 1))
* koszt({1, 2, 3}, 2) = min(koszt({1, 3}, 1) + d(1, 2), koszt({1, 3}, 3) + d(3, 2))
* koszt({1, 2, 3}, 3) = min(koszt({1, 2}, 1) + d(1, 3), koszt({1, 2}, 2) + d(2, 3))

1. Zbiory 3-elementowe to maksymalne zbiory, ponieważ rozważamy 4 wierzchołki, z których jeden jest wierzchołkiem startowym. Możemy więc wyznaczyć wzór końcowy:

min(koszt({1, 2, 3}, 1) + d(1, 0), koszt({1, 2, 3}, 2) + d(2, 0), koszt({1, 2, 3}, 3) + d(3, 0))

Obliczenie wartości otrzymanego wzoru pozwoli nam uzyskać wartość liczbową, reprezentującą minimalną wagę krawędzi, jakie tworzą cykl Hamiltona w zadanym grafie. Następnie, należy przejść po kolejnych iteracjach algorytmu, w celu uzyskania kolejności wierzchołków wyznaczających najkrótszą ścieżkę.

# Implementacja algorytmów

Algorytmy operują na zmiennych intege. Pomiar czasu wykonany jest przy pomocy biblioteki języka chrono mierzącej czas z dokładnością do nanosekund, przy wykorzystaniu pomiaru ilości ticków procesora. Do przechowywania czasów używana jest zmienna typu long, co pozwala zachować dużą dokładność pomiaru. Algorytmy zaimplementowane są w postaci klas, co zapewnia ich poprawne działanie w każdym środowisku uruchomieniowym i na każdym systemie operacyjnym.

# Wyniki pomiarów

|  |  |
| --- | --- |
| Liczba miast | Czas[mikrosekundy] |
| 2 | 43 |
| 4 | 147 |
| 6 | 786 |
| 8 | 4792 |
| 10 | 29519 |
| 12 | 164565 |
| 14 | 892385 |

# Wnioski

Algorytm wykorzystujący metodę programowania dynamicznego wykazuje się czasem wykonania rosnącym znacząco wolniej wraz ze wzrostem zbioru danych niż brute force. Nie zapewnia on jednak stuprocentowej poprawności wyniku, w przeciwieństwie do algorytmów typu brute force.