Wrocław, 03.06.2014r.

Damian Golonka, 200649

Mateusz Karbowiak, 200684

Prowadzący: mgr inż. Agata Rusoń

**Struktury danych i złożoność obliczeniowa**

**Projekt 3** – Implementacja i analiza efektywności algorytmów optymalnych o pseudowielomianowej złożoności obliczeniowej dla wybranych problemów kombinatorycznych.

1. **Wstęp**

Celem projektu implementacja oraz dokonanie pomiaru czasu działania algorytmów dla następujących problemów kombinatorycznych:

* Dyskretny problem plecakowy
* Asymetryczny problem komiwojażera

Dla dyskretnego problemu plecakowego opracowaliśmy:

* Algorytm oparty na programowaniu dynamicznym

Dla asymetrycznego problemu komiwojażera opracowaliśmy:

* Metodę przeglądu zupełnego (brute force)
* Algorytm zachłanny

Czasy działania algorytmów:

* Problem plecakowy – programowanie dynamiczne: 
* Problem plecakowy – przegląd zupełny: 
* Problem plecakowy – algorytm zachłanny: 
* Problem komiwojażera – bruteforce: *O(n!)*
* Problem komiwojażera – algorytm zachłanny: 

Czasy działania algorytmów zostały opracowane na podstawie książki:   
'Wprowadzenie do algorytmów'[1].

1. **Plan eksperymentu**

Wielkość struktur ustalana jest dynamicznie. Problem plecakowy jest jednym z bardziej znanych problemów optymalizacyjnych. Problem ten można porównać do problemu złodzieja który znalazł N towarów i i-ty przedmiot jest wart ci oraz waży wi. Złodziej chce zabrać jak najlepszy łup, ale nie może zabrać więcej niż B kilogramów. Przedmioty nie mogą być ułamkami.

Do mierzenia czasów operacji użyliśmy funkcji, którą w swoim dokumencie zamieścił   
dr inż. Jarosław Mierzwa[2]. Została zmodyfikowana, aby wyświetlać czas w milisekundach.

***Dyskretny problem plecakowy:***

Do każdej operacji utworzyliśmy pętlę, która wykonywała się 100 razy. Czasy uśredniliśmy. W każdej pętli generowana jest nowa instancja. Badania wykonaliśmy dla 5 różnych liczb przedmiotów N, a dla każdej liczby N wykonaliśmy 3 różne badania dla pojemności plecaka B. To pozwoliło nam uzyskać wykresy ukazujące złożoność naszych struktur. Czasy liczyliśmy dla N=1000, 5000, 10000, 20000, 50000 i B = 1000, 3000, 5000. Dane do programu można wczytać z pliku. W pierwszej linii wpisujemy: pojemność\_plecaka liczba\_przedmiotów(oddzielone spacją). W drugiej i kolejnych dla każdego przedmiotu: waga wartość.

**Specyfikacja laptopa, na którym wykonywane były testy:**

Procesor: Intel i5-3210 @ 2,50GHz

Pamięć RAM: 8GB

System: Windows 7 64-bit

***Asymetryczny problem komiwojażera:***

Dane do eksperymentu można wczytywać z pliku. W pierwszej linii podawana jest ilość miast, w kolejnych natomiast, koszty podróży do kolejnych wierzchołków – każdy wiersz to kolejne miasto.

Obliczenia wykonywaliśmy dla 5 różnych ilości miast. W pętli generowaliśmy 100 instancji dla pary liczba miast N – algorytm, po czym wyniki uśredniliśmy.

Specyfikacja komputera, na którym wykonywane były obliczenia:

Procesor: Intel Xeon E3 – 1230 V2 @ 3.30 GHz

Pamięć: 8 GB

System: Windows 7 64-bit

Program potrzebny do badania podstawowych struktur danych został napisany w języku C++ w programie Visual Studio 2013.

1. **Dyskretny problem plecakowy**

**3.1 Programowanie dynamiczne**

Metoda programowania dynamicznego polega na znalezieniu rekurencyjnej zależności między rozwiązaniami pod problemów różnego poziomu. W algorytmie wykorzystujemy vector z biblioteki stl.

W1,…..,wn oznacza wagi przedmiotów, a c1,…cn oznacza wartości. Algorytm musi zmaksymalizować sumę wartości przedmiotów, zachowując sumy ich wagi mniejszej bądź równej Niech A(i, j) będzie największą możliwą wartością, która może być otrzymana przy założeniu wagi mniejszej bądź równej j.



Dla pustego plecaka rozwiązanie jest równe zero. Obliczenie A(i) wymaga sprawdzenia n przedmiotów.

Złożoność tego problemu nie neguje faktu, że problem plecakowy jest NP zupełny, ponieważ W, w przeciwieństwie do n, nie jest proporcjonalne do rozmiaru danych wejściowych dla problemu.

Jeśli chcemy dodać do rozwiązania i-ty element, który jest cięższy od dopuszczalnej pojemności plecaka, to ten ruch jest ruchem nieopłacalnym i nadal rozwiązanie składa się z przedmiotów już wybranych. Jeśli ten przedmiot, po dodaniu, ma masę nie większa niż ładowność plecaka to zaczynamy sprawdzanie dwóch przypadków:

* wybrany zbiór zawiera dany element
* wybrany zbiór nie zawiera danego elementu

**Tabela wyników**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **B** | **N** | | | | |
| **1000** | **5000** | **10000** | **20000** | **50000** |
| **1000** | 3,499196 | 17,26806 | 35,06951 | 67,91895 | 170,988 |
| **3000** | 10,58293 | 51,7267 | 113,6633 | 201,2735 | 499,0388 |
| **5000** | 17,26223 | 83,85941 | 171,3236 | 334,8547 | 840,6319 |

**Wnioski**

Z przedstawionych wykresów widać ładną zależność problemu od czasu jego wykonania. Im problem stawał się większy, bardziej skomplikowany, czas algorytmu również wydłużał się. Na wykresie widzimy że problem rośnie liniowo, co jest zgodne z logiką. Wydaje nam się, że taki algorytm wykonuje się odpowiednio długo, tzn. czas jest w sam raz dla danego problemu.

**3.2 Przegląd zupełny**

W algorytmie tym sprawdzamy na początku wszystkie istniejące podzbiory zbioru n przedmiotów, a następnie odrzucamy te, których waga sumaryczna przekracza pojemność plecaka. W tym momencie złożoność obliczeniowa jest duża. Na koniec algorytm porównuje ze sobą wszystkie podzbiory. Złożoność takiego algorytmu wynosi O(2n). Jest to bardzo nieefektywna metoda. Złożoność ta wynika ciągu zero jedynkowego na n polach. Jest to najprostsza metoda wyszukiwania.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N - przed** | **5** | **7** | **9** | **11** | **13** | **14** | **15** | **16** |
| **Czas [ms]** | 0,068558 | 1,78866 | 34,8983 | 412,129 | 1757,2175 | 39877,9 | 187908 | 858055 |
|  |  |  |  |  |  | ~1min | 3min | 14 min |

**Wnioski**

Jak widać na wykresie, algorytm ten naprawdę jest nieefektywny. Uzyskane czasy zgadzają się z teorią i złożonością tego algorytmu. Algorytm dla coraz to większej liczby n, wykonuje się 2^n razy dłużej. Nasz algorytm nie był w stanie przetworzyć liczby przedmiotów większej niż 17, dla której czas oczekiwania przekroczył 30 min. Dla liczby N równej 16 było to już 14 minut, co udało się zmierzyć. Różna pojemność plecaka nie miała większego wpływu na czas działania algorytmu. Największy problem w tym algorytmie jest to, że przegląda wszystkie możliwości. Nasza pętla wykonywała się *pow(2,n)* razy, co sprawiało, że algorytm trwa bardzo długo. Algorytm w wersji dynamicznej jest bardziej dopracowany dlatego jest o wiele lepszy do problemu plecakowego.

1. **Asymetryczny problem komiwojażera**

# **Opis**

Wyobraźmy sobie komiwojażera, który podróżuje od miasta do miasta, sprzedając tam swoje produkty lub zawierając różne oferty handlowe. Wyrusza z miasta rodzinnego, po czym jego trasa przebiega dokładnie jeden raz przez każde z miast. Na koniec komiwojażer powraca do swojego miasta rodzinnego. Z oczywistych powodów chce, aby trasa podróży była najkrótszą ze wszystkich możliwych tras. W ten sposób powstaje problem wędrującego komiwojażera (ang. TSP - Travelling Salesman Problem).

W terminologii grafów miasta są wierzchołkami grafu, a trasy pomiędzy nimi to krawędzie z wagami. Waga krawędzi może odpowiadać odległości pomiędzy miastami połączonymi tą krawędzią, czasowi podróży lub kosztom przejazdu - zależy, co chcemy w podróżny komiwojażera zminimalizować. Trasa komiwojażera jest cyklem przechodzącym przez każdy wierzchołek grafu dokładnie jeden raz - jest to zatem [cykl Hamiltona](http://edu.i-lo.tarnow.pl/inf/utils/002_roz/ol025.php).

**Przegląd zupełny (bruteforce)**

Metoda bruteforce polega na znalezieniu permutacji wszystkich miast, wyliczenia sumy kosztów ścieżek między nimi i wybraniu najmniejszej. Ma dużą złożoność obliczeniową, czas wykonania może szybko zmienić się od kilkudziesięciu sekund do stuleci.

W założeniach projektu trzeba było zbadać średnie czasy dla >10 miast, lecz już dla 12 eksperyment trwał bardzo długo. Zmniejszyliśmy zakres i uśredniliśmy czasy dla 5, 8 i 10 miast.

|  |  |  |  |
| --- | --- | --- | --- |
| **Liczba miast [N]** | 5 | 8 | 10 |
| **Czas [ms]** | 5 | 43,97 | 4038,77 |

**Wnioski**

Algorytm ten jest nieefektywny, zajmuje bardzo dużo czasu przy większej ilości miast, lecz przy kilku – kilkunastu jesteśmy w stanie prawie idealnie podać najbardziej optymalną ścieżkę dla komiwojażera.

**Algorytm zachłanny**

W algorytmie zachłannym nie dokonujemy permutacji miast. Program na bieżąco sprawdza, którą trasę obrać, kierując się wytyczną – jak najtaniej. Skraca to ogromnie czas obliczeń w porównaniu do metody bruteforce. Dla liczby miast <1000 otrzymywaliśmy wyniki oscylujące w okolicach 0 ms. Wykres można było utworzyć dla ponad 1000 miast.

|  |  |  |  |
| --- | --- | --- | --- |
| **Liczba miast [N]** | 1000 | 5000 | 7000 |
| **Czas [ms]** | 1,18 | 47,28 | 93,68 |

**Wnioski**

O ile czasy drastycznie zmalały i można tym algorytmem zbadać o wiele większą ilość miast, tak wyniki nie były już zadowalające. Wydawać by się mogło, że kryterium najtańszej drogi od ostatnio odwiedzonego miasta brzmi rozsądnie, jednak nie zawsze oznacza to optymalną trasę. Czasem metodą bruteforce można było otrzymać bardziej optymalną ścieżkę.

1. **Bibliografia**

[1] Thomas H. Cormen. *Wprowadzenie do algorytmów. Wydanie czwarte.*

Warszawa: WNT, 2001. ISBN 83-204-2665-0.

[2] http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pamsi/debug\_and\_time.pdf