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:

* Trzeba wpisać

Czasy działania algorytmów:

* Problem plecakowy – programowanie dynamiczne: 
* Problem plecakowy – przegląd zupełny: 
* Problem plecakowy – algorytm zachłanny: 
* Problem komiwojażera –
* Problem komiwojażera –
* Problem komiwojażera –

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: wartość waga.

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

Procesor: Intel i5-3210 @ 2,50GHz

Pamięć RAM: 8GB

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. **~~Wyznaczanie najkrótszej ścieżki w grafie~~**

**~~4.1 Algorytm Dijkstry~~**

~~Algorytm ten należy do grupy algorytmów zachłannych. Warunkiem działania algorytmu jest podanie mu grafu który nie posiada krawędzi o ujemnej wadze.~~

~~Działanie algorytmu:~~

* ~~Utworzenie tablicy D odległości od źródła dla wierzchołków grafu~~
* ~~Utworzenie zbioru wszystkich wierzchołków Q~~
* ~~Pobranie ze zbioru, wierzchołka o największej odległości od wybranego wierzchołka s, którego odległość od źródła wynosi 0~~
* ~~Usunięcie wierzchołka najbliższego źródła~~
* ~~Dla sąsiada wierzchołka v dokonaj relaksacji poprzez u~~

**~~4.2 Algorytm Bellmana-Forda~~**

~~Algorytm również służy do rozwiązywania problemów w wyznaczaniu najkrótszej ścieżki w grafie. Ma on jednak przewagę nad Dijkstry, ponieważ nie musi operować na grafach, które nie posiadają krawędzi o wadze ujemnej. Pamiętać trzeba jedynie o tym, że graf nie może posiadać cykli o ujemnej długości. Algorytm ten posługuje się metodą relaksacji.~~

~~Działanie algorytmu:~~

* ~~Macierz A dla każdej pary wierzchołków zawiera wagę krawędzi (u,v)~~
* ~~Jeśli krawędź nie istnieje ti przyjmujemy, że jej waga wynosi ~~
* ~~W każdym kroku obliczamy górne oszacowanie odległości s do wszystkich pozostałych wierzchołków.~~
* ~~Po stwierdzeniu, że D(v)>D(u)+A(u,v) -> polepszamy aktualne oszacowanie~~
* ~~Algorytm kończymy, gdy żadnego oszacowania nie można już poprawić~~

**~~Reprezentacja macierzowa~~**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **~~Wierz.~~** | **~~Dijkstry 0,3~~** | **~~Dijkstry 0,5~~** | **~~Dijkstry 0,75~~** | **~~Dijkstry 0,9~~** | **~~Bellman 0,3~~** | **~~Bellman 0,5~~** |
| **~~400~~** | ~~0,00375~~ | ~~0,007~~ | ~~0,013~~ | ~~0,01825~~ | ~~3,72675~~ | ~~5,14175~~ |
| **~~1000~~** | ~~0,0135938~~ | ~~0,020625~~ | ~~0,0247813~~ | ~~0,027~~ | ~~28,848~~ | ~~43,3441~~ |
| **~~1600~~** | ~~0,0399492~~ | ~~0,0407031~~ | ~~0,0474727~~ | ~~0,055625~~ | ~~76,4745~~ | ~~128,3~~ |
| **~~2200~~** | ~~0,0898379~~ | ~~0,0906187~~ | ~~0,103309~~ | ~~0,115078~~ | ~~160,803~~ | ~~267,73~~ |
| **~~2800~~** | ~~0,148605~~ | ~~0,157452~~ | ~~0,170539~~ | ~~0,195135~~ | ~~290,159~~ | ~~485,043~~ |

**~~Reprezentacja listowa~~**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **~~Wierz.~~** | **~~Dijkstry 0,3~~** | **~~Dijkstry 0,5~~** | **~~Dijkstry 0,75~~** | **~~Dijkstry 0,9~~** | **~~Bellman 0,3~~** | **~~Bellman 0,5~~** |
| **~~400~~** | ~~0,00175~~ | ~~0,0035~~ | ~~0,00325~~ | ~~0,00525~~ | ~~45,7578~~ | ~~77,4789~~ |
| **~~1000~~** | ~~0,0142188~~ | ~~0,0208125~~ | ~~0,0306562~~ | ~~0,0352813~~ | ~~98,158~~ | ~~153,7756~~ |
| **~~1600~~** | ~~0,0511523~~ | ~~0,0851016~~ | ~~0,118457~~ | ~~0,13816~~ | ~~156,72645~~ | ~~278,4486~~ |
| **~~2200~~** | ~~0,128519~~ | ~~0,208763~~ | ~~0,305182~~ | ~~0,368395~~ | ~~278,4687~~ | ~~467,3369~~ |
| **~~2800~~** | ~~0,26494~~ | ~~0,433595~~ | ~~0,619398~~ | ~~0,744299~~ | ~~490,587~~ | ~~685,4056~~ |

**~~4.3 Wnioski~~**

~~Wykresy dla algorytmu Dijkstry i Forda-Bellmana zawarliśmy osobno, ponieważ jak widać czasy dla algorytmu Bellmana są bardzo duże w porównaniu do algorytmu Dijkstry. Gęstość grafu ma wpływ na algorytm w przypadku dużej liczby krawędzi. Rozkłady dla algorytmów wyznaczania najkrótszej ścieżki w grafie są bardzo podobnego do algorytmów MST. Nasza implementacja algorytmu Bellmana okazała się bardzo kiepska. Czasy są bardzo duże. Dla gęstości powyżej 50% nasza implementacja algorytmu Bellmana nie dała sobie rady. Czas wykonywania był bardzo duży a po 20 minutach pokazywał się błąd pamięci.~~

1. **~~Wnioski~~**

**~~Osobne wykresy dla każdej gęstości grafu:~~**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ~~Macierz~~ | | ~~Lista~~ | |
|  | **~~Kruskal 0,3~~** | **~~Prim 0,3~~** | **~~Kruskal 0,3~~** | **~~Prim 0,3~~** |
| ~~400~~ | ~~0,00431~~ | ~~0,00218~~ | ~~0,01725~~ | ~~0,00317~~ |
| ~~1000~~ | ~~0,0289963~~ | ~~0,0122492~~ | ~~0,280281~~ | ~~0,0394672~~ |
| ~~1600~~ | ~~0,079756~~ | ~~0,0257292~~ | ~~1,29216~~ | ~~0,113201~~ |
| ~~2200~~ | ~~0,155551~~ | ~~0,0513737~~ | ~~3,6849~~ | ~~0,274118~~ |
| ~~2800~~ | ~~0,254611~~ | ~~0,0857804~~ | ~~8,12224~~ | ~~0,47699~~ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ~~Macierz~~ | | ~~Lista~~ | |
|  | **~~Kruskal 0,5~~** | **~~Prim 0,5~~** | **~~Kruskal 0,5~~** | **~~Prim 0,5~~** |
| ~~400~~ | ~~0,0071~~ | ~~0,00552~~ | ~~0,029~~ | ~~0,00483~~ |
| ~~1000~~ | ~~0,0468591~~ | ~~0,0159765~~ | ~~0,492875~~ | ~~0,0532718~~ |
| ~~1600~~ | ~~0,138993~~ | ~~0,031887~~ | ~~2,19161~~ | ~~0,157847~~ |
| ~~2200~~ | ~~0,250917~~ | ~~0,0629729~~ | ~~6,3197~~ | ~~0,380502~~ |
| ~~2800~~ | ~~0,426552~~ | ~~0,115644~~ | ~~14,182~~ | ~~0,804793~~ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ~~Macierz~~ | | ~~Lista~~ | |
|  | **~~Kruskal 0,75~~** | **~~Prim 0,75~~** | **~~Kruskal 0,75~~** | **~~Prim 0,75~~** |
| ~~400~~ | ~~0,01016~~ | ~~0,01293~~ | ~~0,04275~~ | ~~0,00725~~ |
| ~~1000~~ | ~~0,0696702~~ | ~~0,0119619~~ | ~~0,738344~~ | ~~0,0684312~~ |
| ~~1600~~ | ~~0,194921~~ | ~~0,02505~~ | ~~3,42804~~ | ~~0,226749~~ |
| ~~2200~~ | ~~0,375621~~ | ~~0,057713~~ | ~~9,80063~~ | ~~0,576085~~ |
| ~~2800~~ | ~~0,641799~~ | ~~0,0975327~~ | ~~21,9526~~ | ~~1,15394~~ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ~~Macierz~~ | | ~~Lista~~ | |
|  | **~~Kruskal 0,9~~** | **~~Prim 0,9~~** | **~~Kruskal 0,9~~** | **~~Prim 0,9~~** |
| ~~400~~ | ~~0,01229~~ | ~~0,01056~~ | ~~0,0535~~ | ~~0,008~~ |
| ~~1000~~ | ~~0,0824803~~ | ~~0,0418416~~ | ~~0,913063~~ | ~~0,071628~~ |
| ~~1600~~ | ~~0,227654~~ | ~~0,0796028~~ | ~~4,18751~~ | ~~0,234413~~ |
| ~~2200~~ | ~~0,473629~~ | ~~0,1196606~~ | ~~12,0553~~ | ~~0,595442~~ |
| ~~2800~~ | ~~0,861797~~ | ~~0,1844827~~ | ~~26,6374~~ | ~~1,241283~~ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ~~Macierz~~ | | ~~Lista~~ | |
|  | **~~Dijkstry 0,3~~** | **~~Bellman 0,3~~** | **~~Dijkstry 0,3~~** | **~~Bellman 0,3~~** |
| ~~400~~ | ~~0,00375~~ | ~~3,72675~~ | ~~0,00175~~ | ~~45,7578~~ |
| ~~1000~~ | ~~0,0135938~~ | ~~28,848~~ | ~~0,0142188~~ | ~~98,158~~ |
| ~~1600~~ | ~~0,0399492~~ | ~~76,4745~~ | ~~0,0511523~~ | ~~156,72645~~ |
| ~~2200~~ | ~~0,0898379~~ | ~~160,803~~ | ~~0,128519~~ | ~~278,4687~~ |
| ~~2800~~ | ~~0,148605~~ | ~~290,159~~ | ~~0,26494~~ | ~~490,587~~ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ~~Macierz~~ | | ~~Lista~~ | |
|  | **~~Dijkstry 0,5~~** | **~~Bellman 0,5~~** | **~~Dijkstry 0,5~~** | **~~Bellman 0,5~~** |
| ~~400~~ | ~~0,007~~ | ~~5,14175~~ | ~~0,0035~~ | ~~77,4789~~ |
| ~~1000~~ | ~~0,020625~~ | ~~43,3441~~ | ~~0,0208125~~ | ~~153,7756~~ |
| ~~1600~~ | ~~0,0407031~~ | ~~128,3~~ | ~~0,0851016~~ | ~~278,4486~~ |
| ~~2200~~ | ~~0,0906187~~ | ~~267,73~~ | ~~0,208763~~ | ~~467,3369~~ |
| ~~2800~~ | ~~0,157452~~ | ~~485,043~~ | ~~0,433595~~ | ~~685,4056~~ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ~~Macierz~~ | | ~~Lista~~ | |
|  | **~~Dijkstry 0,75~~** |  | **~~Dijkstry 0,75~~** |  |
| ~~400~~ | ~~0,013~~ |  | ~~0,00325~~ |  |
| ~~1000~~ | ~~0,0247813~~ |  | ~~0,0306562~~ |  |
| ~~1600~~ | ~~0,0474727~~ |  | ~~0,118457~~ |  |
| ~~2200~~ | ~~0,103309~~ |  | ~~0,305182~~ |  |
| ~~2800~~ | ~~0,170539~~ |  | ~~0,619398~~ |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ~~Macierz~~ | | ~~Lista~~ | |
|  | **~~Dijkstry 0,9~~** |  | **~~Dijkstry 0,9~~** |  |
| ~~400~~ | ~~0,01825~~ |  | ~~0,00525~~ |  |
| ~~1000~~ | ~~0,027~~ |  | ~~0,0352813~~ |  |
| ~~1600~~ | ~~0,055625~~ |  | ~~0,13816~~ |  |
| ~~2200~~ | ~~0,115078~~ |  | ~~0,368395~~ |  |
| ~~2800~~ | ~~0,195135~~ |  | ~~0,744299~~ |  |

~~Jak widzimy na powyższych wykresach algorytm Kruskala wykonuje się najdłużej w reprezentacji macierzowej jak i w reprezentacji listowej. Najszybciej wykonuje się algorytm Prima, czyli algorytm MST.~~

~~Algorytm Dijkstry przy zwiększaniu gęstości odczuwa różnice powyżej 75%. Różnica ta jest znacząca jeżeli patrzymy na reprezentacje listową. Algorytm Forda-Bellmana po naszej implementacji wykonuje się najdłużej, co z jednej strony jest przewidujące, ponieważ jest to algorytm ogólny tzn. algorytm ten nie opera się na założeniu że wagi są nieujemne. Algorytm ten nie rozwiązuje jednak problemu cyklu nieujemnego. Stwierdza jedynie fakt, że taki cykl wystąpił i dalej nie może działać.~~

~~Przy zadaniu projektowym napotkaliśmy dużo kłopotów z generowanie grafu losowego. Trudne okazało się zachowanie spójność grafu. Jako, że zastosowaliśmy macierz sąsiedztwa, zyskaliśmy miejsce w pamięci, ponieważ macierz ta jest symetryczna dla grafów nie skierowanych. Powoduje to, że możemy przechowywać tylko połowę. W programie nie udało nam się dokończyć wczytywania z pliku.~~

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