Jakub Balcerzak

Projektowanie Efektywnych Algorytmów

Termin zajęć: czwartek godz. 7:30

Prowadzący:

Zadanie projektowe 1

- sprawozdanie

1. Opis zadania projektowego:

W projekcie dokonano implementacji algorytmu programowania dynamicznego dla problemu komiwojażera (ang. TSP).

2. Wstęp teoretyczny:

Problem komiwojażera jest problem optymalizacyjnym. Dany jest w nim graf pełny z N wierzchołkami, które reprezentują miasta, a także o ważonych krawędziach, wskazujących koszt połączenia pomiędzy dwoma poszczególnymi miastami. Należy zaznaczyć, że przez koszt rozumiemy tutaj przykładowo: długość drogi, czas przebycia odcinka lub wydatki na benzynę związane z przebyciem danego fragmentu drogi.

Graf może być skierowany, wówczas mamy do czynienia z asymetrycznym problemem komiwojażera lub nieskierowany, wówczas problem jest symetryczny.

Tytułowy komiwojażer podróżuje przez N miast, każde odwiedzając dokładnie jeden raz oraz wracając na koniec do miasta początkowego. Algorytm ma na celu odnalezienie drogi o minimalnym koszcie, co oznacza, że problem komiwojażera jest tożsamy ze znalezieniem minimalnego cyklu w grafie Hamiltona.

Do rozwiązywania problemów optymalizacyjnych służy m.in. programowanie dynamiczne. Jednak słowo ‘programowanie’ oznacza w tym kontekście bardziej tabelaryczną metodę rozwiązania problemu.

Problem (podobnie, jak przy strategii „dziel i zwyciężaj”) rozwiązywany jest przez złożenie rozwiązań podproblemów. Programowanie dynamiczne może być zastosowane, gdy podproblemy są niezależne, co oznacza, że mogą one zawierać te same podpodproblemy.

Proces projektowania algorytmu programowania dynamicznego można scharakteryzować w czterech wydzielonych etapach:

1. Scharakteryzowanie struktury optymalnego rozwiązania.
2. Rekurencyjne zdefiniowanie kosztu optymalnego rozwiązania.
3. Obliczenie wartości optymalnego rozwiązania metodą wstępującą.
4. Skonstruowanie optymalnego rozwiązania na podstawie wyników wcześniejszych obliczeń.

Ww. metoda wstępująca oznacza rozpoczęcie rozwiązywania problemu od najmniejszych podproblemów, poprzez coraz większe. Wykorzystując przy tym zapamiętane rozwiązania mniejszych podproproblemów. Te zapamiętane rozwiązania są przechowywane w tablicy, stąd określenie „tabelarycznej metody rozwiązywania problemu”.

Na koniec należy wymienić dwie własności, które musi posiadać problem, aby jego rozwiązanie dało osiągnąć się na drodze programowania dynamicznego:

- Optymalną podstrukturę – rozwiązanie optymalne jest funkcją optymalnych rozwiązań podproblemów.
- Własność wspólnych problemów – algorytm rekurencyjny rozwiązuje wielokrotnie te same podproblemy.

Należy dodatkowo zapewnić realizację procesu spamiętywania, który zapamiętuje rozwiązania podproblemów w dodatkowej tablicy. Wypełnianie tej tablicy zaszyte jest w algorytmie rekurencyjnym.

W przypadku zastosowania programowania dynamicznego do rozwiązania problemu komiwojażera skonstruowano algorytm Helda-Karpa (Bellmana-Helda-Karpa). Algorytm ten posiada złożoność czasową: $O(n^2 2^n)$ oraz pamięciową: $O(n2^n)$. Pomimo złożoności wykładniczej sprawdza się on lepiej, niż algorytm brut force, który sprawdza wszystkie permutacje, a jego złożoność wynosi: O(n!)

Złożoność pamięciowa algorytmu Helda-Karpa wiąże się z wykorzystaniem tablicy, w której zapamiętywane są rozwiązania podproblemów. Jej rozmiar wynosi $[n]x[2^n]$, gdzie n to liczba miast. Wewnątrz tego algorytmu rozpatrywane są wszystkie miasta oraz wszystkie kombinacje, których jest odpowiednio n oraz $n2^n$. Stąd złożoność obliczeniowa.

Sam algorytm działa następująco. Załóżmy, że mamy 1,2,3,…, n miast. Odległości pomiędzy nimi są dodatnie oraz oznaczamy je, jako $d_{i,j}$ – waga dojścia z i do j. C(S, p) oznaczać będzie dojście do miasta początkowego (przyjmijmy miasto 1) z p, przechodząc przy tym przez zbiór wierzchołków S.

$C(S, p) = d_{p,1}$ dla $|S| = 1$

$C(S, p) = \min_{x \varepsilon S \setminus \{p\}} \{C(S \setminus \{p\}, x) + d_{p,xi}\}$

Na końcu rozwiązujemy problem: $\min_{x \varepsilon \{2,3,\ldots,n\}} \{C(\{2, 3,\ldots, n\}, x) + d_{1,x}\}$

3. Przykład działania algorytmu programowania dynamicznego:

Przyjmijmy następujące odległości pomiędzy miastami:

	1	2	3
1	0	2	7
2	5	0	2
3	8	1	0

Zaczynamy od najmniejszych podproblemów:

$C(\phi,2) = d_{2,1} = 5$

$C(\phi,3) = d_{3,1} = 8$

Powiększamy zbiór o jeden element:

$C(\{3\}, 2) = \min\{C(\phi, 3) + d_{2,3}\} = \min\{8 + 2\} = 10$

$C(\{2\}, 3) = \min\{C(\phi, 2) + d_{3,2}\} = \min\{5+ 1\} = 6$

Powiększamy zbiór S o kolejny element, dochodzimy do przypadku ostatniego, zatem rozwiązujemy problem:

$\min_{x \varepsilon \{2,3\}} \{C(\{2\},3) + d_{1,3}, C(\{3\}, 2) + d_{1,2}) = \min\{6 + 7, 10 + 2\} = \min\{13, 12\} = 12$ dla x = 2.

Odtwarzając rozwiązanie otrzymujemy drogę: 1→2→3→1. jej koszt wynosi 12.

4. Opis implementacji:

Wszystkie wymienione poniżej struktury zostały zaimplementowane w projekcie i są dynamicznie alokowane:

- graf – reprezentowany przez macierz sąsiedztwa, rozmiaru NxN,
- tablica spamiętująca – jej funkcje opisano w punkcie 2, rozmiar $Nx2^N$,
- tablica odtwarzająca drogę – rozmiar $Nx2^N$.

Do obliczenia minimalnego cyklu służy rekurencyjna funkcja countPath() zwracająca całkowity koszt drogi. Z kolei displayPath() odpowiada za odtworzenie drogi i wyświetlenie jej w oknie konsoli. Pozostałe funkcje, jak i wykorzystane zmienne zostały odpowiednio opatrzone komentarzem w kodzie źródłowym.

5. Plan eksperymentu:

W programie zaimplementowano funkcje:

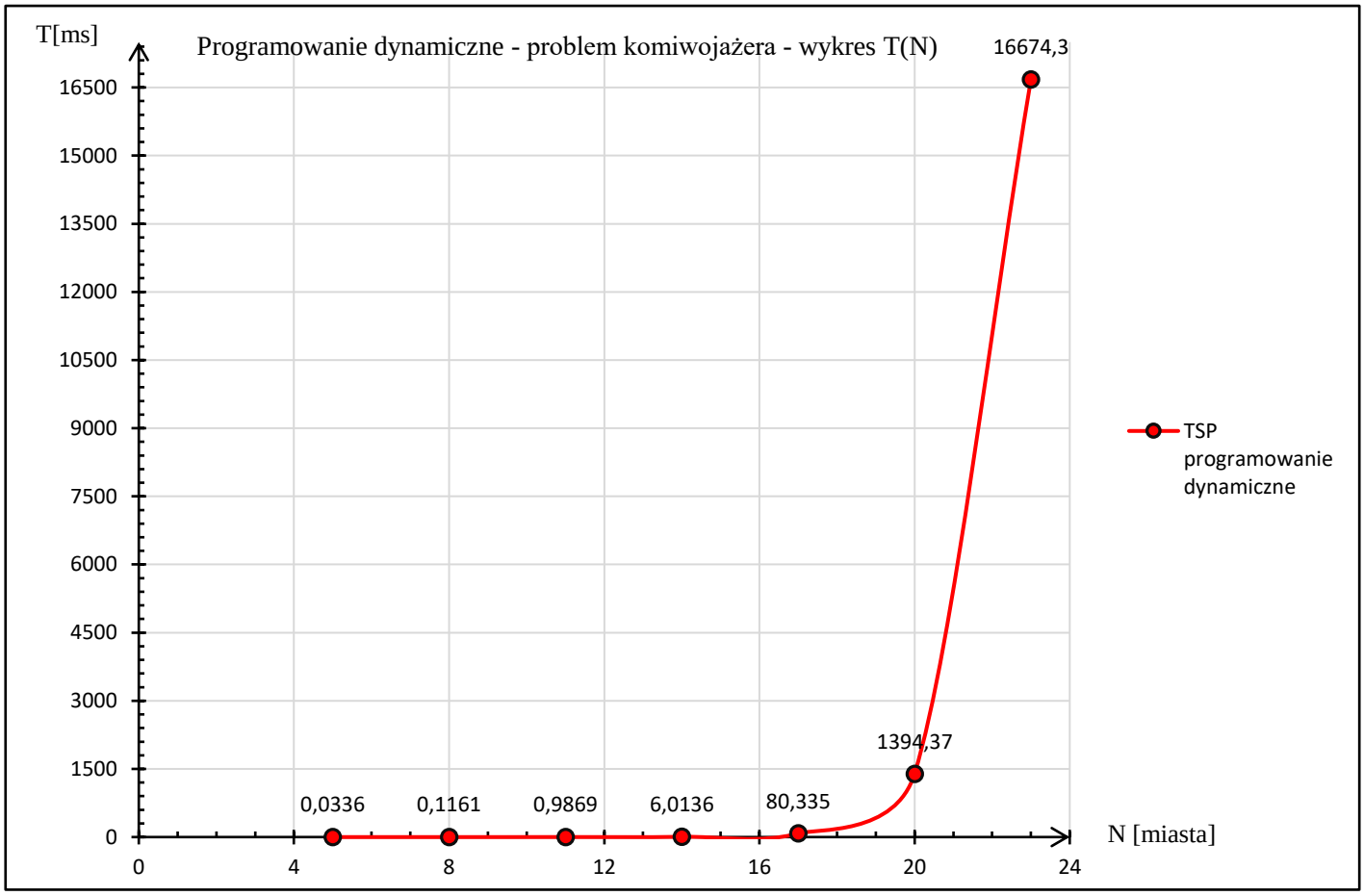
- wczytującą dane testowe z pliku .txt w formacie zgodnym z podanym w instrukcji
- generującą graf – do funkcji przekazywana jest liczba miast N, macierz sąsiedztwa jest zapełniania liczbami z zakresu <1,50000> o rozkładzie jednostajnym. Na przekątnej macierzy są 0, które oznaczają brak połączenia pomiędzy miastami. Zakres został przyjęty do 50000, gdyż maksymalna macierz, którą można było utworzyć posiadała rozmiar 23x23, co oznacza 23*23-23 = 506 pól do wypełnienia. W związku z czym uznano przyjęty zakres za wystarczający.

Pomiary były wykonywane dla procesu obliczania długości minimalnego cyklu. Każdy pomiar dla danej wielkości grafu (N) został powtórzony stukrotnie, a następnie uśredniony. Wyniki pomiarów prezentowane są w punkcie 6. niniejszego sprawozdania. Pomiary wykonywano wg instrukcji projektowej.

Pomiary, a także powyższe funkcje, zostały opracowane w oparciu o materiały pomocnicze ze strony przedmiotu.

6. Pomiary:

N	t [µs]	t [ms]	t [s]
5	33,6	0,0336	0,0000336
8	116,1	0,1161	0,0001161
11	986,9	0,9869	0,0009869
14	6013,6	6,0136	0,0060136
17	80335,0	80,3350	0,0803350
20	1394370,0	1394,3700	1,3943700
23	16674300,0	16674,3000	16,6743000



7. Wnioski:

Programowanie dynamiczne, choć lepsze w sensie złożoności czasowej od rozwiązania siłowego, dalej ma jedną wadę: jego złożoność nie jest wielomianowa, lecz wykładnicza, co prezentuje wykres w punkcie 6. Dlatego też wymieniane w punkcie 2. złożoności algorytmu Helda-Karpa są tak samo nieefektywne w ujęciu czasowym, jak i pamięciowym.

Podczas przeprowadzania testów nie można było zbadać przypadku większego od 23 miast. Wynika to z dużej złożoności pamięciowej, która to uniemożliwiała. Należy zwrócić tu uwagę na przyrost wykorzystania pamięci wraz ze wzrostem rozmiaru problemu, co dla wybranych wartości N prezentuje poniższa tabela:

N	pamięć
8	1 MB
14	3 MB
20	178 MB
23	2 GB

Stosunek pamięci dla dwudziestu miast do czternastu (178 MB / 3 MB) wynosi 59.333, co oznacza blisko 60-krotny wzrost jej wykorzystania. Porównując dwa ostatnie miasta widzimy, że wzrost ten jest 11-krotny (2 GB / 178 MB). Po mimo niemożliwości przetestowania algorytmu dla N większych, niż 23 można prognozować dalszy wzrost

wykorzystania pamięci. Zebrane dane potwierdzają dużą teoretyczną złożoność pamięciową dla wykorzystanego algorytmu.

Korzystając z tabeli z pomiarami czasowymi, umieszczonej w punkcie 6. sprawozdania możemy obliczyć, w jak szybkim tempie wzrastał czas wykonywania się algorytmu. I tak:

- $N_{14} / N_{11} \approx 6{,}09$
- $N_{17} / N_{14} \approx 13{,}36$
- $N_{20} / N_{17} \approx 17{,}36$
- $N_{23} / N_{20} \approx 11{,}96$
- $N_{23} / N_{17} \approx 207{,}56$

Prezentowane powyżej stosunki czasów wykonywania się algorytmu dowodzą jego dużej złożoności czasowej, a tym samym jego nieefektywności.

Podsumowując, zebrane dane tj. pomiary prezentowane w tabelach i na wykresach, a także analiza ich dla zaimplementowanego algorytmu pozwalają stwierdzić, że jego złożoności odpowiadają wiedzy teoretycznej (złożonościom teoretycznym) prezentowanej w punkcie 2.