Projektowanie Efektywnych Algorytmów
Przegląd zupełny
Programowanie dynamiczne

Jakub Klawon

Listopad 2023

Spis treści

1 Wstęp teoretyczny

Tematem projektu jest przygotowanie programu do badania efektywności algorytmów: przeglądu zupełnego (ang. brute force) i programowania dynamicznego (ang. dynamic programming) w rozwiązywaniu problemu komiwojażera (ang. traveling salesman problem).

1.1 Przegląd zupełny

Algorytm przeglądu zupełnego polega na wygenerowaniu wszystkich możliwych permutacji i policzeniu trasy dla każdej z nich. Złożoność tego algorytmu to $O(n!)$.

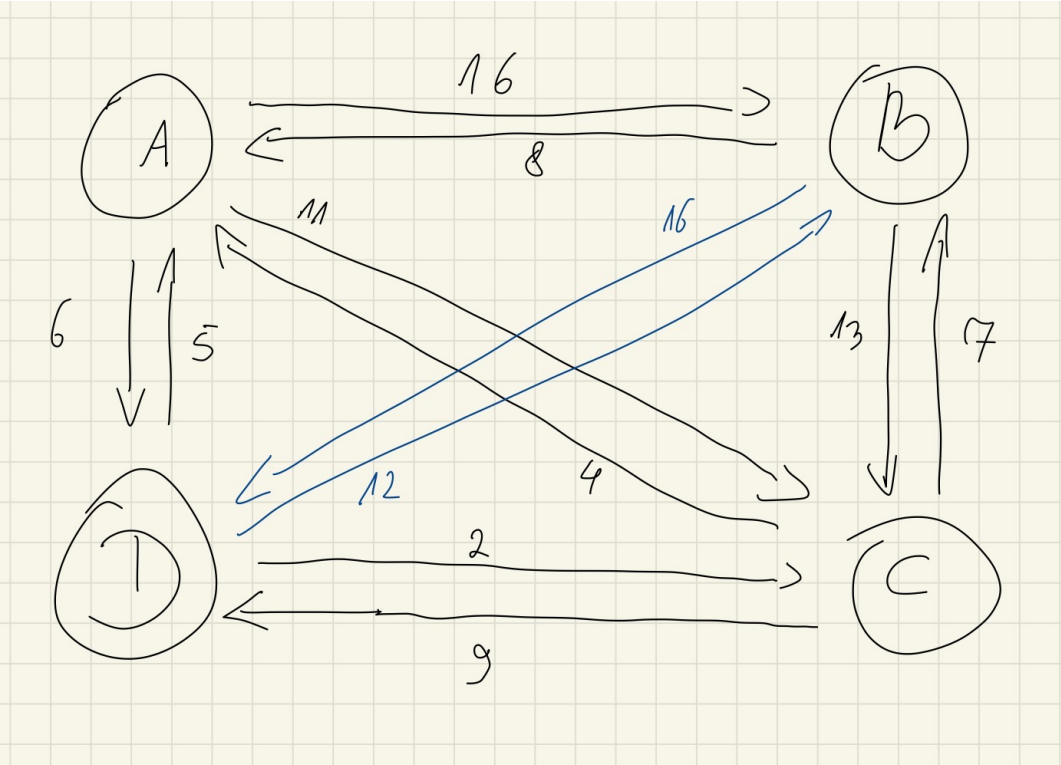
1.2 Programowanie dynamiczne

Programowanie dynamiczne polega na rekurencyjnym obliczaniem poddrzew drzewa możliwych ścieżek. Złożoność tego algorytmu to $O(n^2 * 2^n)$.

2 Przykład praktyczny programowania dynamicznego

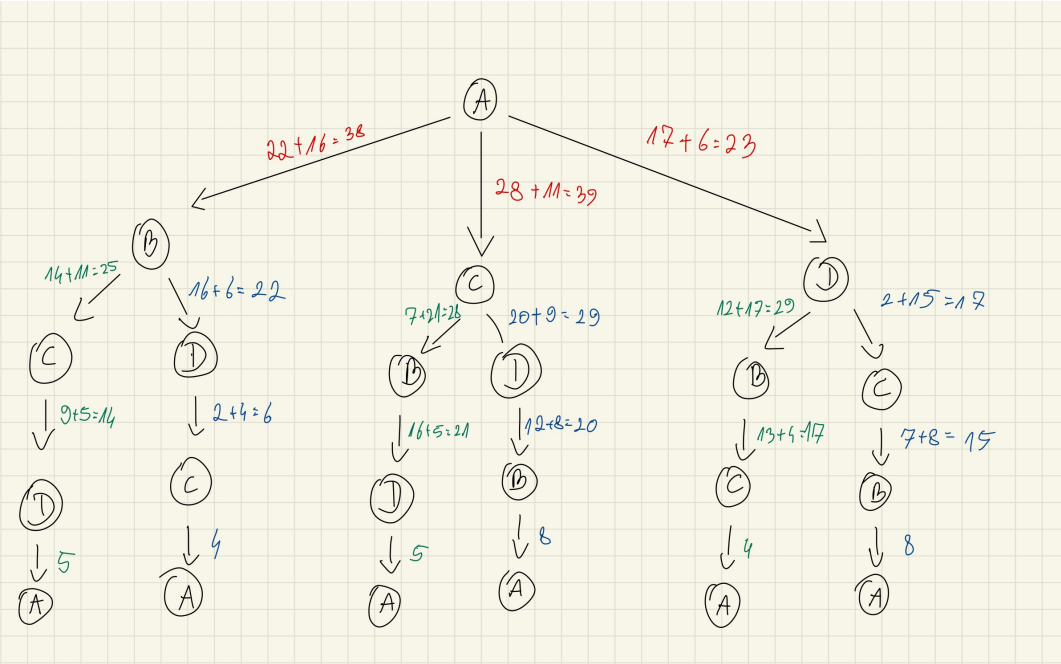
2.1 Dane

$$\left|\begin{array}{cccc} 0 & 16 & 11 & 6 \\ 8 & 0 & 13 & 16 \\ 4 & 7 & 0 & 9 \\ 5 & 12 & 2 & 0 \end{array}\right|$$



Rysunek 1: Graficzne przedstawienie danych

2.2 Opis



Rysunek 2: Graficzne przedstawienie opisu

1. Pętla zaczyna się od 0 (miasto A)
2. Funkcja wywołuje się dla trasy A -> B + wynik
3. Trasa B -> C + wynik
4. Trasa C -> D + wynik
5. Trasa D -> A zwraca wynik = 5
6. Trasa B -> C zwraca wynik 14 + 11 = 25, porównuje go z najmniejszym dotychczasowym i jako, że jest mniejszy, oznacza go jako najmniejszy dotychczasowy
7. Trasa D -> C + wynik
8. Trasa C -> A zwraca wynik = 4
9. Trasa B -> D zwraca wynik 16 + 6 = 22, porównuje go z najmniejszym dotychczasowym i jako, że jest mniejszy, oznacza go jako najmniejszy dotychczasowy
10. Trasa A -> B zwraca wynik 22 + 16 = 38, który zostaje oznaczony jako najmniejszy dotychczasowy
11. Pętla przechodzi o 1
12. Funkcja wywołuje się dla trasy A -> C + wynik
13. Trasa C -> B + wynik
14. Trasa B -> D + wynik
15. Trasa D -> A zwraca wynik = 5

16. Trasa C -> B zwraca wynik 7 + 21 = 28, porównuje go z najmniejszym dotychczasowym i jako, że jest mniejszy, oznacza go jako najmniejszy dotychczasowy
17. Trasa D -> B + wynik
18. Trasa B -> A zwraca wynik = 4
19. Trasa B -> D zwraca wynik 20 + 9 = 29, porównuje go z najmniejszym dotychczasowym i jako, że jest większy, nie zmienia wyniku
20. Trasa A -> C zwraca wynik 28 + 11 = 39, porównuje go z najmniejszym dotychczasowym i jako, że jest większy, nie zmienia wyniku
21. Pętla przechodzi o 1
22. Funkcja wywołuje się dla trasy A -> D + wynik
23. Trasa D -> B + wynik
24. Trasa B -> C + wynik
25. Trasa C -> A zwraca wynik = 4
26. Trasa D -> B zwraca wynik 12 + 17 = 29, porównuje go z najmniejszym dotychczasowym i jako, że jest mniejszy, oznacza go jako najmniejszy dotychczasowy
27. Trasa C -> B + wynik
28. Trasa B -> A zwraca wynik = 4
29. Trasa D -> C zwraca wynik 2 + 15 = 17, porównuje go z najmniejszym dotychczasowym i jako, że jest mniejszy, oznacza go jako najmniejszy dotychczasowy
30. Trasa A -> D zwraca wynik 17 + 6 = 23, który zostaje oznaczony jako najmniejszy dotychczasowy
31. Funkcja zwraca ostateczny wynik jakim jest 23

3 Opis implementacji algorytmu

3.1 Przegląd zupełny

1. Funkcja najpierw generuje wszystkie możliwe permutacje, dodając je do tablicy dynamicznej jednowymiarowej dwukierunkowej zmiennych typu string.
2. Następnie rozpoczyna liczenie dodając kolejno trasy między elementami danej trasy. Jednocześnie po policzeniu decyduje, czy wynik jest mniejszy od najniższego i aktualizuje zwracany wynik.
3. Po obliczeniu wyniku danej permutacji, funkcja usuwa ją z tablicy dynamicznej.

3.2 Programowanie dynamiczne

1. Funkcja zaczyna od początkowego miasta oznaczając je jako odwiedzone zmieniając odpowiednią pozycję w masce binarnej
2. Rozpoczynając od pierwszego sąsiada wywołuje się do momentu aż maska binarna będzie miała wszystkie pozycje zmienione na "1"
3. Kiedy dotrze do ostatniego nieodwiedzonego miasta, zwraca trasę od niego do miasta początkowego "x"
4. Każde miasto odwiedzone po drodze zwraca trasę x dodając do niej trasy między sobą
5. W momencie, kiedy funkcja ma wybór między trasami do przekazania, sprawdza czy dany koszt jest mniejszy od aktualnie najmniejszego

6. Funkcja zapamiętuje również wyniki pewnych tras przechowując je w tablicy dwuwymiarowej o wielkości (1 « n) (przesunięcie bitowe o n pozycji, np. dla n = 4, 1 « n = 16) na n - "memo"(z ang. memoization - zapamiętywanie). Używa do tego aktualnej maski oraz pozycji, jeśli element nie jest równy -1, funkcja zwraca zapamiętaną wartość.

7. Do zapamiętywania przebytej trasy, funkcja używa tablicy dwuwymiarowej "path", która na podstawie maski oraz aktualnej pozycji, przechowuje wartości następnego miasta.

4 Plan eksperymentu

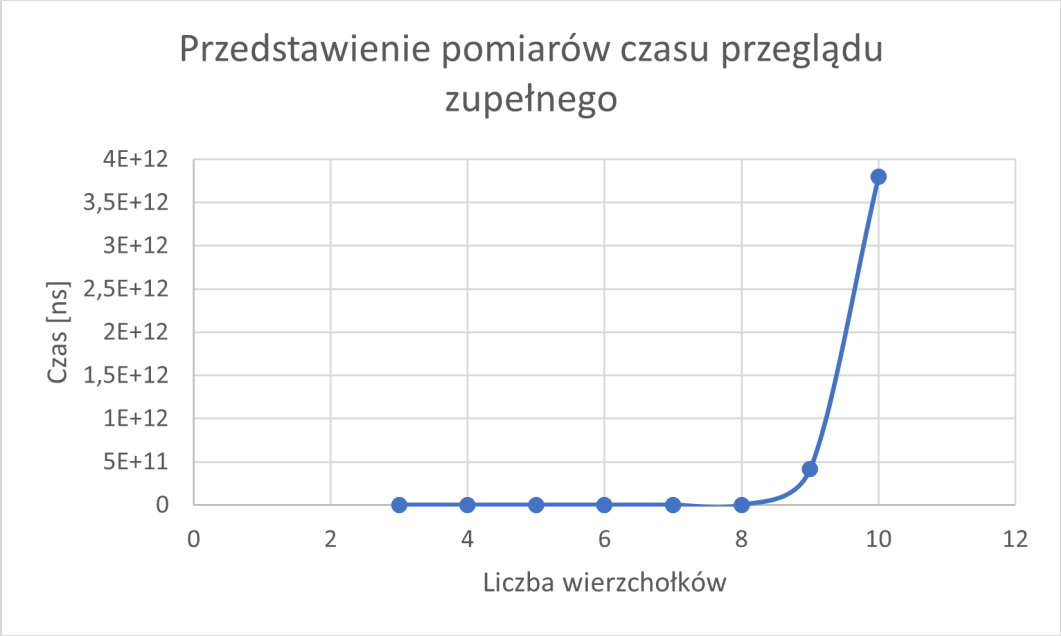
Dane generowane są losowo z zakresu wartości od 1 do 2000. Klasa odpowiadająca za przechowywanie macierzy tworzy dane i przechowuje w tablicy dwuwymiarowej. Tworzone jest 50 różnych zestawów, który poddaje się danemu algorytmowi w 10 iteracjach. Z 50 wyników tworzy się średnią, którą przekazuje się do tablicy wyników. Na koniec program przedstawia średnią z wyników, którą wraz z danymi, zapisuje do pliku informując o wielkości zestawów.

5 Wyniki eksperymentu

5.1 Przegląd zupełny

Liczba wierzchołków	Uśredniony czas [ns]
3	90,8
4	321,4
5	2546
6	17221,8
7	61513,4
8	474073
9	4,13E+11
10	3,80E+12

Tabela 1: Uśrednione pomiary czasów dla przeglądu zupełnego

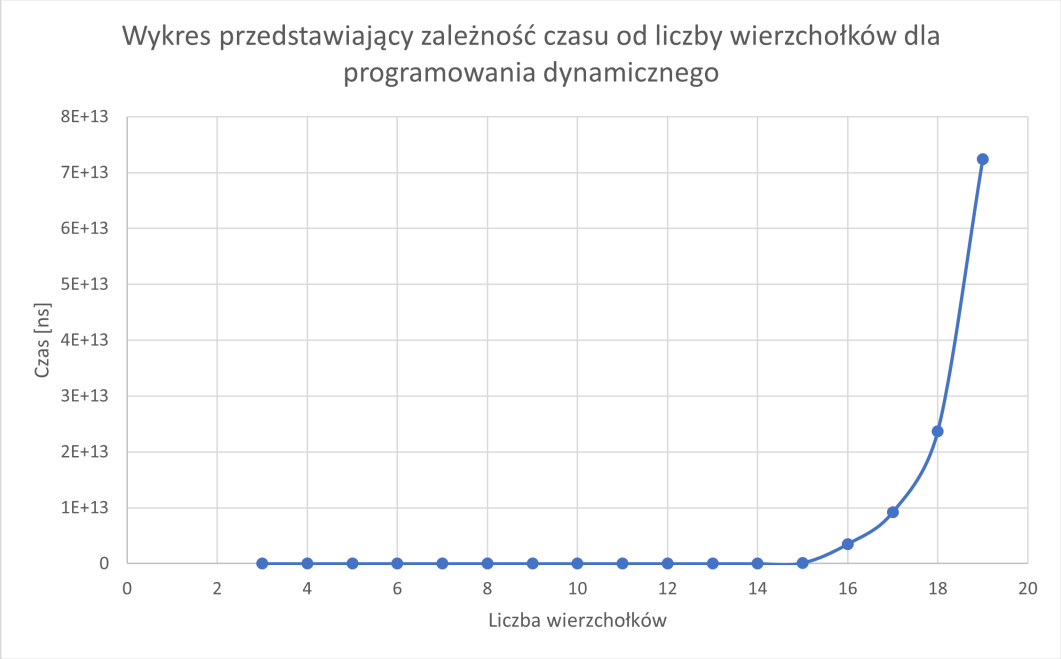


Rysunek 3: Wykres przedstawiający uśrednione pomiary czasów dla przeglądu zupełnego

5.2 Programowanie dynamiczne

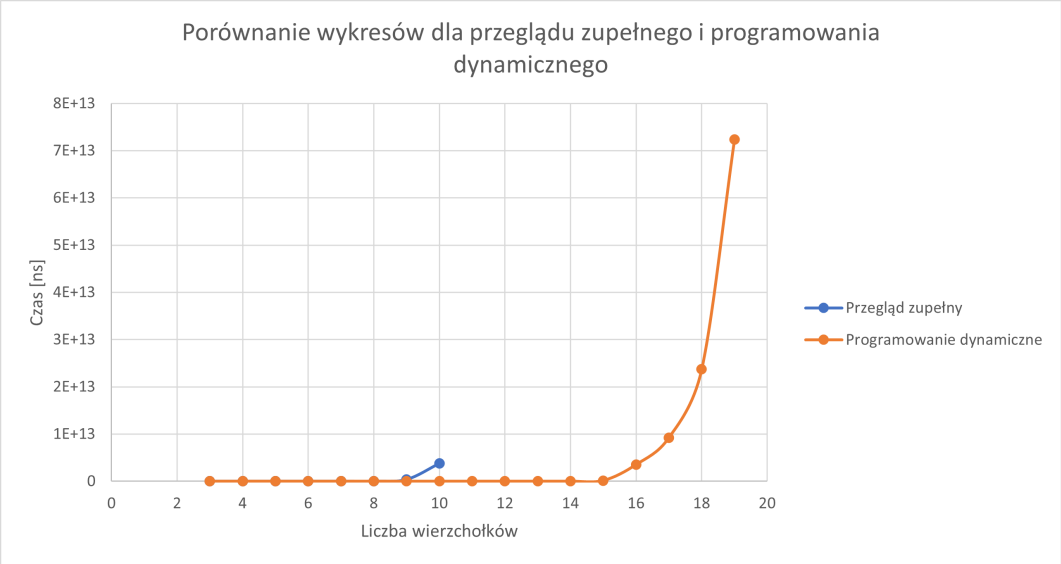
Liczba wierzchołków	Uśredniony czas [ns]
3	99,8
4	246
5	886,2
6	2203,4
7	4475,8
8	23194,2
9	67433,6
10	198537
11	589983
12	1,30E+06
13	2,78E+06
14	6,54E+06
15	1,51E+11
16	3,52E+12
17	9,23E+12
18	2,37E+13
19	7,24E+13

Tabela 2: Uśrednione pomiary czasów dla programowania dynamicznego

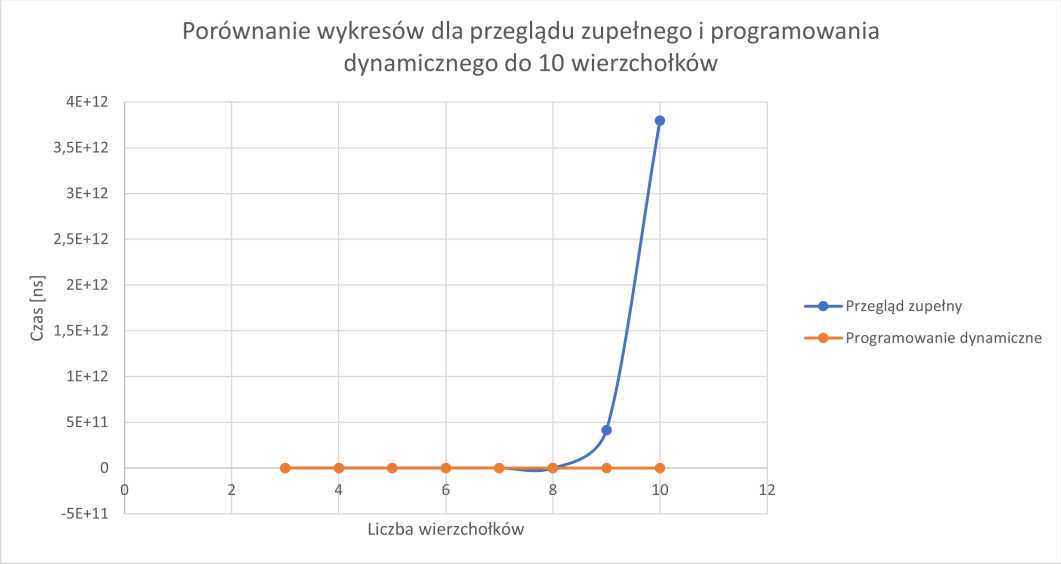


Rysunek 4: Wykres przedstawiający uśrednione pomiary czasów dla programowania dynamicznego

5.3 Porównanie przeglądu zupełnego i programowania dynamicznego



Rysunek 5: Porównanie wykresów dla przeglądu zupełnego i programowania dynamicznego



Rysunek 6: Porównanie wykresów dla przeglądu zupełnego i programowania dynamicznego do 10 wierzchołków

6 Wnioski

6.1 Przegląd zupełny

Wykres przedstawiający czas wykonania algorytmu w zależności od liczby wierzchołków pokazuje, że implementacja algorytmu pokrywa się z wartością teoretyczną $O(n!)$. Maksymalna liczba wierzchołków, dla której udało się wykonać pomiary to 10. Przy 11 wierzchołkach program po godzinie dalej nie kończył pracy. Dla sprawdzenia maksymalnej liczby wierzchołków, dla której wykona się algorytm, wygenerowałem kilka losowych macierzy i wykonałem pojedyncze badania bez mierzenia czasu. Jednakże dla 11 wierzchołków, ponownie musiałem przerwać program po godzinie bez odpowiedzi.

6.2 Programowanie dynamiczne

Wykres przedstawiający czas wykonania algorytmu w zależności od liczby wierzchołków pokazuje, że implementacja algorytmu pokrywa się z wartością teoretyczną $O(n^2 * 2^n)$. Maksymalna liczba wierzchołków, dla której udało się wykonać pomiary to 19. Przy 20 wierzchołkach program po 2 godzinach dalej nie kończył pracy, aż wreszcie poinformował o braku pamięci do dalszego działania. Maksymalna ilość wierzchołków, dla której algorytm działał to 26.