Projektowanie Efektywnych Algorytmów

Projekt

16.11.2021

256423 Maciej Radziszewski

(2) Held-Karp

1. Treść zadania

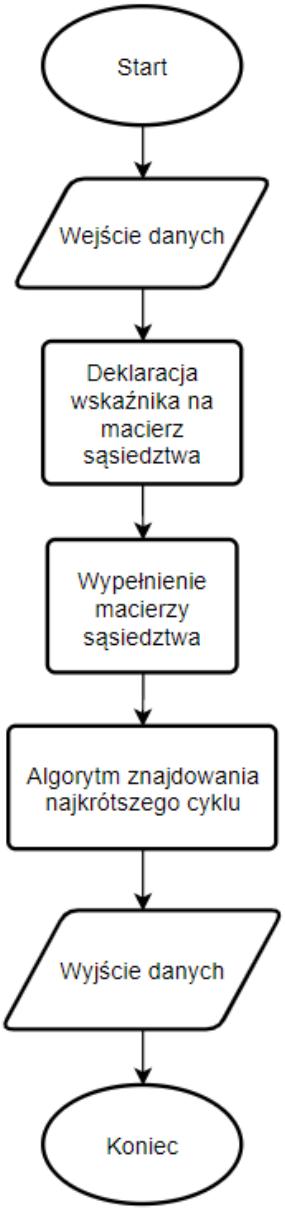
Zadane polega na implementacji i zbadaniu efektywności algorytmu Helda-Karpa rozwiązującego problem komiwojażera w wersji programowania dynamicznego. Problem komiwojażera polega na odnalezieniu minimalnego lub maksymalnego (w zależności od wymagań) cyklu Hamiltona w grafie. Graf, który jest brany pod uwagę musi być pełny, oraz jego krawędzie muszą mieć wagi.

2. Metoda

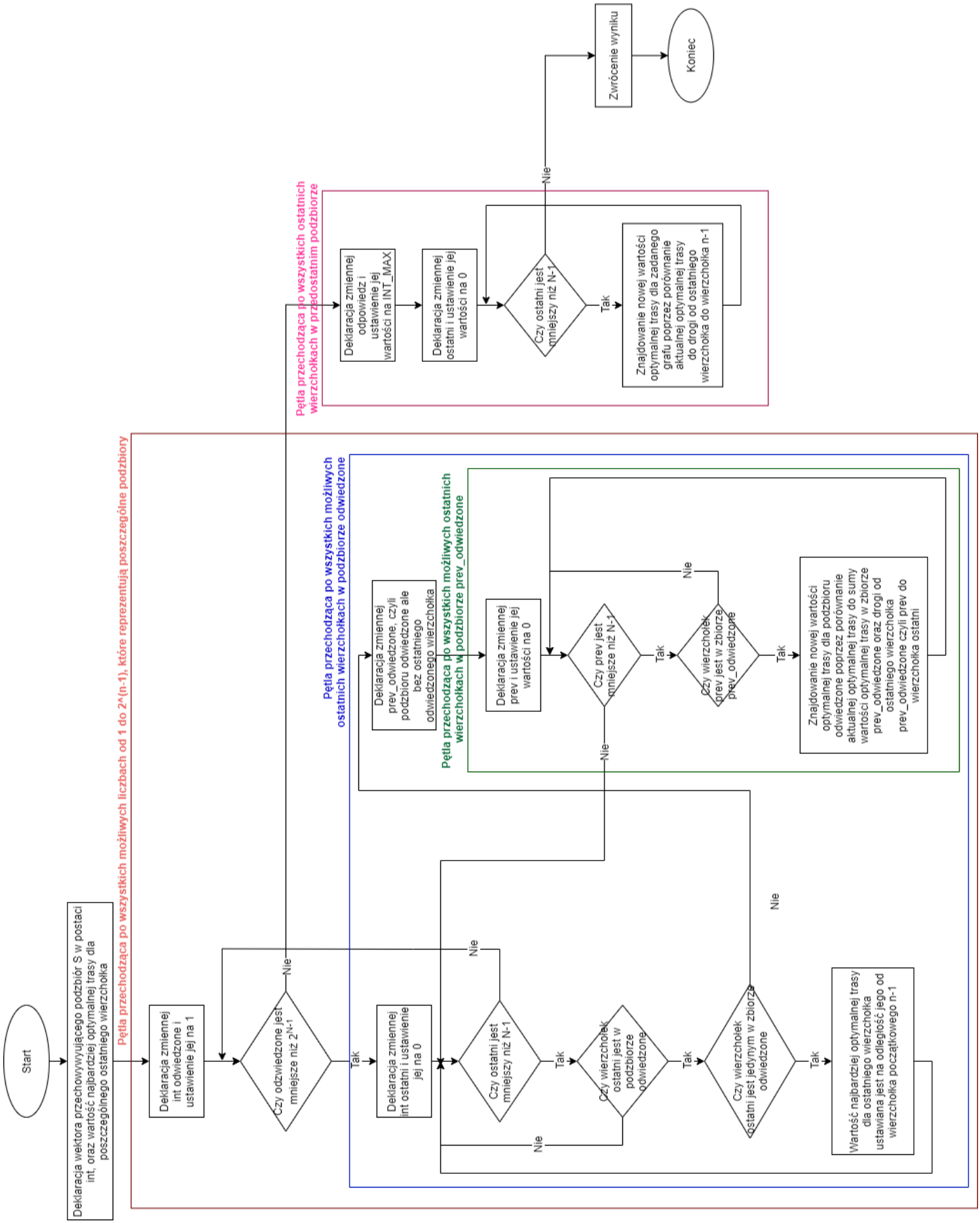
Problem przedstawiano w treści zadania rozwiązywany jest metodą programowania dynamicznego. Została ona opracowana w 1957 roku przez Richarda Ernesta Bellmana. Metoda określa podejście opierające się na przekształceniu zadania w wieloetapowy proces obejmujący rozwiązanie podproblemów, na które podzielony jest główny problem. Wyniki rozwiązań pomniejszych problemów są zapisywane, dzięki czemu przy rozwiązywaniu podproblemu większego dostępne jest już rozwiązanie podproblemów mniejszych w nim zawartych. Każdy kolejny pomniejszy problem rozwiązywany jest w ten sam sposób. Na końcu otrzymujemy optymalny wynik, ponieważ jesteśmy pewni, że na każdym poprzednim etapie wynik także był optymalny.

3. Algorytm

W implementacji został wykorzystany algorytm Helda-Karpa opierający się na wyżej wymienionej metodzie. Algorytm ten został opracowany przez Richarda Ernesta Bellmana, Michaela Helda, oraz Richarda Manninga Karpa. Polega on na znajdowaniu najkrótszej ścieżki dla podzbiorów wierzchołków, a następnie na używaniu tego wyniku w celu opracowania najkrótszej ścieżki dla coraz większych podzbiorów, aż do momentu dojścia do zbioru wszystkich wierzchołków, gdzie obliczana jest wartość najkrótszego cyklu Hamiltona dla zadanego grafu. W implementacji wykorzystano metodę przechowywania podzbioru odwiedzonych wierzchołków w ciągu bitów zapisywanym jako liczba całkowita. Algorytm polega na przejściu po wszystkich możliwych podzbiorach które reprezentowane są liczbami całkowitymi od 1 do 2^{n-1} i znajdowania dla każdego z nich najkrótszej możliwej ścieżki, innej w zależności od tego, który wierzchołek był ostatnim odwiedzonym. Na rysunku 1. przedstawiono schemat blokowy programu, a na rysunku 2. schemat blokowy algorytmu.



Rys. 1. – Schemat blokowy programu



Rys. 2. – Schemat blokowy algorytmu

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu wybrano następujący zestaw instancji:

1.tsp15.txt, war. optymalna: 291; http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/peastud/tsp

2.ulysses16.txt, war. optymalna: 6774; http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php

3.br17.txt, war. optymalna: 39; http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php

4.ulysses22.txt, war. optymalna: 6929; http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php

5.gr24.txt, war. optymalna: 1272; http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php

5.fri26.txt, war. optymalna: 937; http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu od wielkości instancji. W przypadku algorytmu realizującego rozwiązanie problemu za pomocą programowania dynamicznego nie występowały parametry programu, które mogły mieć wpływ na czas i jakość uzyskanego wyniku. W związku z tym procedura badawcza polegała na uruchomieniu programu sterowanego plikiem inicjującym .INI.

Dla każdej instancji czas mierzony był 5 razy. Pomiary zostały wykonane na komputerze ze specyfikacją: Procesor AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz. Zainstalowana pamięć RAM 8,00 GB (dostępne: 7,37 GB). Typ systemu 64-bitowy system operacyjny, procesor x64. Do pomiaru czasu została użyta biblioteka <chrono>, wraz z narzędziem high_resolution_clock. Fragment kodu w języku C++ wykonujący pomiar czasu:

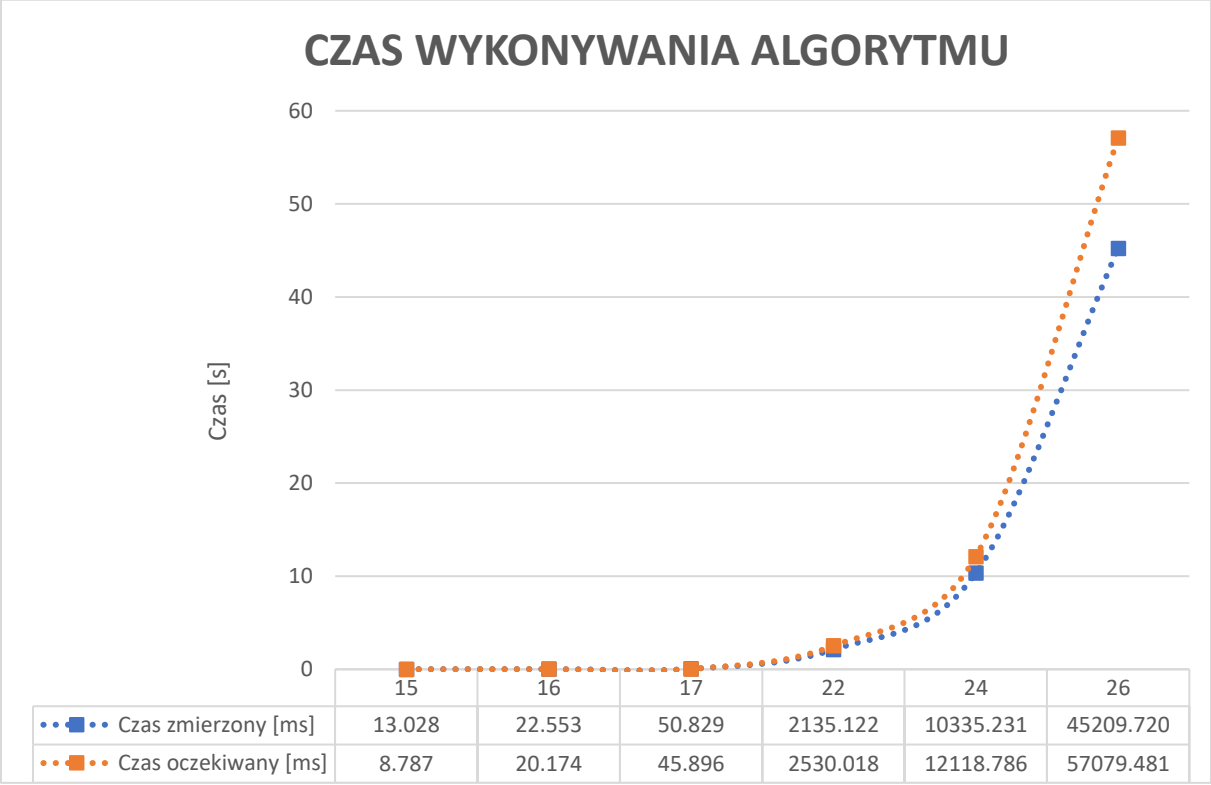
```
auto start = high_resolution_clock::now();

suma_cyklu = heldkarp(num_wierzch, matrix);

auto stop = high_resolution_clock::now();

auto duration = duration_cast<nanoseconds>(stop - start);

float time = (duration.count())/(1e+9);

cout << "Time [ms] : " << std::fixed << std::setprecision(3) << time*1000
```

6. Wyniki

Wyniki zgromadzone zostały w pliku: Wykres.xlsx. Zostały przeprowadzone także testy na instancjach asynchronicznych, jednak nie dostrzeżono różnic w czasie wykonania i w alokowanej pamięci.

Pomiar czasu dla każdej instancji:

Nazwa pliku	Średnia czasu [ms]
tsp15.txt	13.028
ulysses16.txt	22.553
br17.txt	50.829
ulysses22.txt	2 135.122
gr24.txt	10 335.231
fri26.txt	45 209.720



Rys 3. – Wykres czasu wykonywania algorytmu w zależności od wielkości instancji

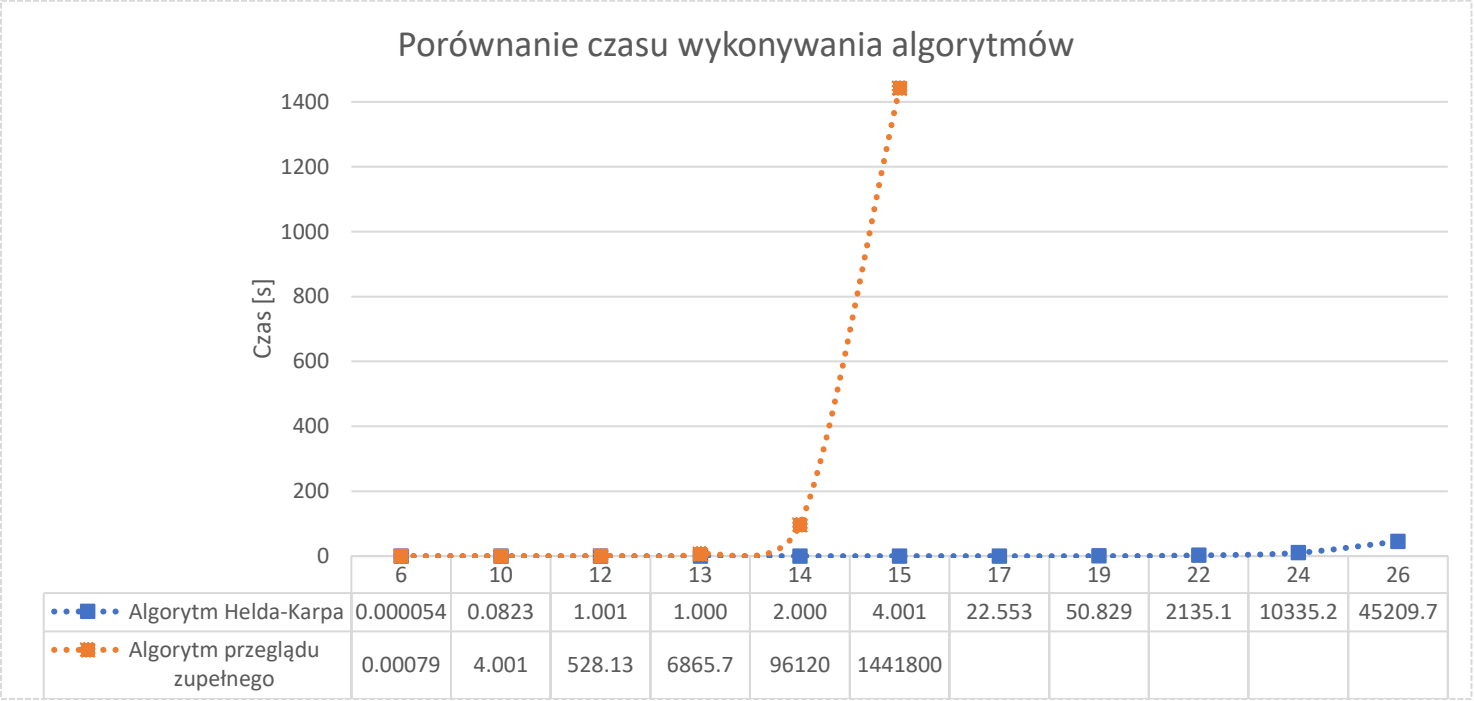
Rezerwowana pamięć:

Nazwa pliku	Rezerwowana pamięć [MB]
tsp15.txt	1.88
ulysses16.txt	4.00
br17.txt	8.50
ulysses22.txt	352.00
gr24.txt	1 536.00
fri26.txt	6 656.00

7. Analiza wyników i wnioski

W celu porównania wyników zadania drugiego zostały zamieszczone wyniki uzyskane w zadaniu pierwszym:

Nazwa pliku	Średnia czasu [ms]
tsp_6_1.txt,	0.000833
tsp_6_2.txt,	0.000906
tsp_10.txt,	4.917450
tsp_12.txt,	573.332620
tsp_13.txt,	7 054.432900
tsp_14.txt,	97 519.344000
tsp_15.txt,	1 426 995.175000



Rys 4. – Wykres porównujący czasu wykonywania algorytmów w zależności od wielkości instancji

Porównując wyniki można dostrzec, że algorytm Helda-Karpa wykonuje zadanie znacznie szybciej, niż algorytm przeglądu zupełnego. Na wykresie (Rysunek 3.) została przedstawiona krzywa wzrostu czasu (niebieska) względem wielkości instancji. Krzywa ta ma charakter wykładniczy, jednak nie rośnie tak drastycznie jak w algorytmie przeglądu zupełnego (Rysunek 4.). Algorytm ma złożoność obliczeniową $O(2^{N-1} \cdot (N-1)^2)$. Taka złożoność przypłacona jest faktem, że początkowym wierzchołkiem jest N-1. Złożoność została wykorzystana w celu obliczenia oczekiwanych wartości czasu w zależności od wielkości instancji. Krzywa opisująca te wyniki została nałożona na wykres. Głównymi atutami algorytmu Helda-Karpa, jest prędkość odnajdywania zawsze optymalnego rozwiązania. Problemem jednak jest złożoność pamięciowa. Algorytm przy instancji o wielkości 29 rezerwuje ilości pamięci na tyle duże, że komputer odmawia współpracy. Dzieje się tak pomimo zapisywania kolejnych podzbiorów w ciągach bitów zapisywanych w postaci liczb całkowitych. Podsumowując, atutami algorytmu Helda-Karpa jest niewielka złożoność obliczeniowa w stosunku do algorytmu przeglądu

zupełnego, oraz pewność zawsze optymalnego wyniku. Przypłacone to jest jednak złożonością pamięciową, która nie pozwala na wyszukiwanie najlepszej ścieżki w instancjach o wielkości powyżej pewnej wartości, w zależności od dostępnej pamięci.