Projektowanie Efektownych Algorytmów
Projekt
07/11/2023

259113 Hubert Bełkot

(2)Held-Karp

Spis treści	strona

1. Sformułowanie Zadania

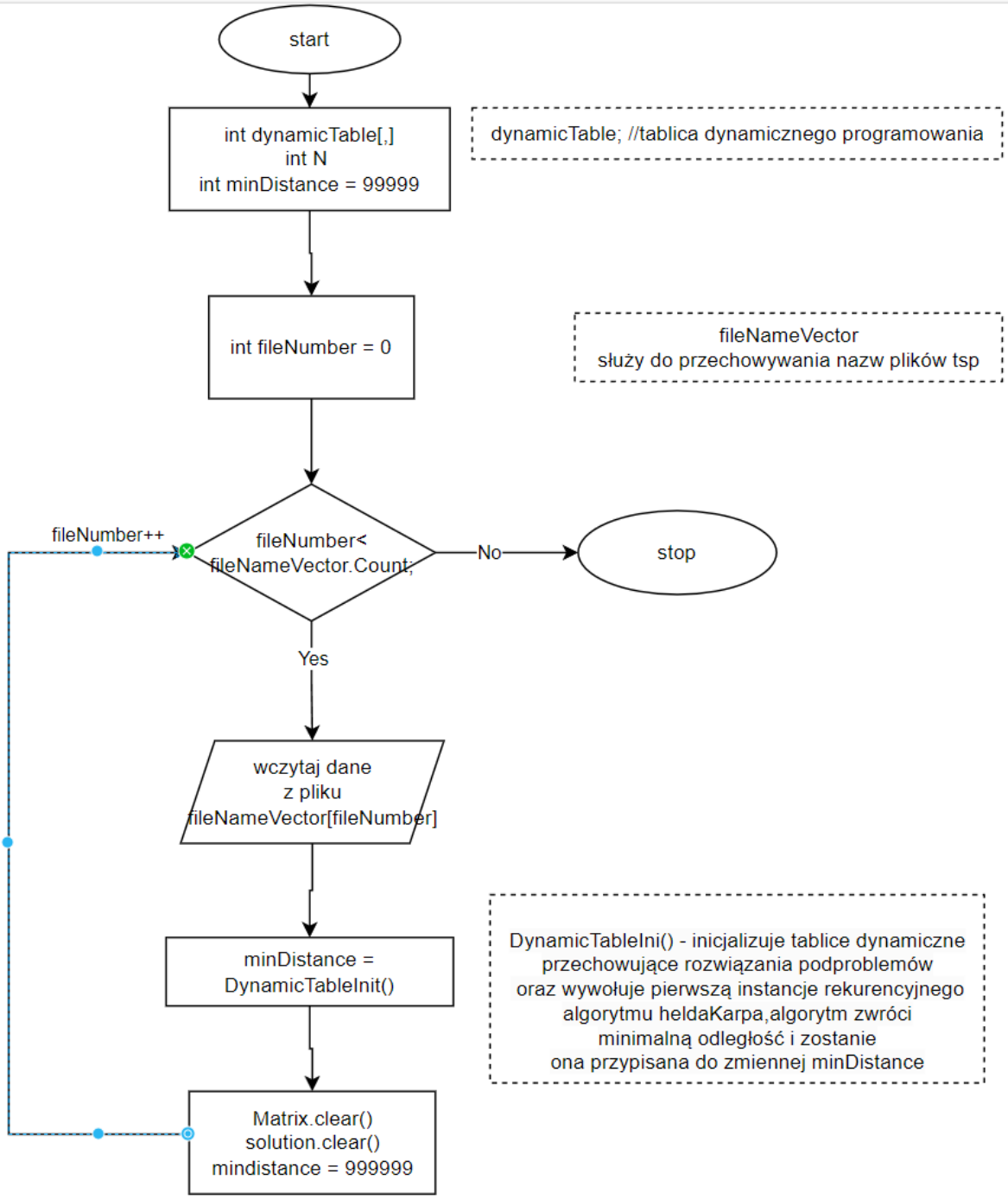
Zadanie polega na opracowaniu i zaimplementowaniu algorytmu Helda-Karpa, rozwiązującego problem komiwojażera(eng. tsp- travelling salesman problem). Problem polega na znalezieniu w grafie minimalnego cyklu, w którym każdy wierzchołek jest odwiedzany dokładnie raz (cykl Hamiltona). Należy zbadać złożoność czasową oraz zależność zużycia pamięci w zależności od wielkości instancji.

2. Metoda

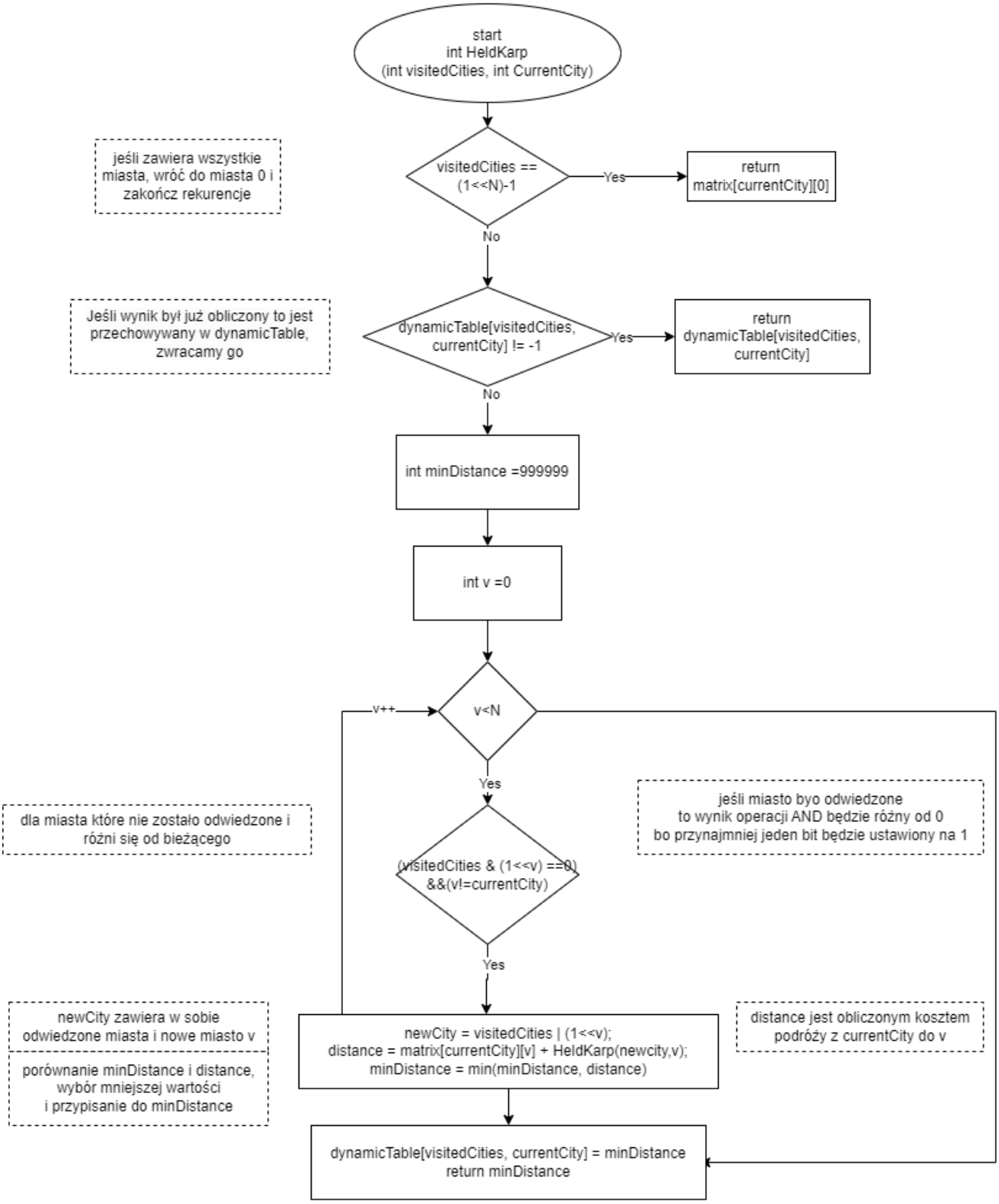
Algorytm Helda-Karpa jest oparty na programowaniu dynamicznym, czyli polega na rozwiązywaniu podproblemów i zapamiętywaniu ich wyników. Dzięki programowaniu dynamicznemu unikamy zbędnych obliczeń w przypadku natrafienia na ten sam podproblem. Podstawowe kroki algorytmu to:

1. Inicjalizacja: w pierwszym kroku, przygotowujemy dynamiczną tabelę, w której będziemy przechowywać obliczone wyniki odległości między różnymi miastami(wierzchołkami grafu). Tabela będzie rosła w miarę postępowania algorytmu.
2. Przypadek bazowy: w kolejnym etapie rozważamy najprostsze przypadki, czyli te dla których mamy tylko dwa miasta. Obliczamy odległość między miastami, co jest naszym punktem wyjścia, a wyniki tych działań zostaną wykorzystane do dalszych obliczeń.
3. Rekurencja: teraz rozważamy bardziej złożone podproblemy. Iterujemy przez wszystkie możliwe podzbiory miast (poza pierwszym miastem). Dla każdego podzbioru obliczamy optymalną trasę. Wybieramy tą, która minimalizuje łączną odległość trasy.
4. Zapamiętywanie wyników: w miarę obliczania optymalnych tras dla różnych podproblemów, zapisujemy wyniki w tabeli dynamicznej. To pozwala uniknąć wielokrotnego obliczania tych samych podproblemów.
5. Przeszukiwanie wsteczne: po zakończeniu obliczeń, możemy skonstruować optymalną trasę, rozpoczynając od końcowego miasta, korzystając z tabeli dynamicznej, cofamy się wybierając kolejne miasta które należy odwiedzić, aby uzyskać optymalną trasę. Po dotarciu do początkowego miasta, kończymy trasę.

3. Algorytm



Rysunek 1: schemat działania głównej części programu



Rysunek 2: schemat funkcji głównej HeldKarp

4. Dane testowe

Do sprawdzenia poprawności algorytmu posłużyły następujące zestawy danych:

- Tsp6_2.txt
- Tsp_10.txt

Do wykonania pomiarów wykorzystano:

- tsp_6_2.txt
- tsp_10.txt
- tsp_12.txt
- tsp_13.txt
- tsp_14.txt
- tsp_15.txt
- gr17.tsp.txt
- gr21.tsp.txt

Ze źródeł:
- http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/
- Teaching (uni-heidelberg.de)

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu od wielkości instancji, czyli złożoność czasową oraz zużycie pamięci od wielkości instancji(złożoność pamięciową). Na początku procedury badawczej uruchamialiśmy plik ini (format_pliku: nazwa_instancji liczba_wykonań rozwiązanie optymalne [ścieżka_optymalna] nazwa_pliku.csv).

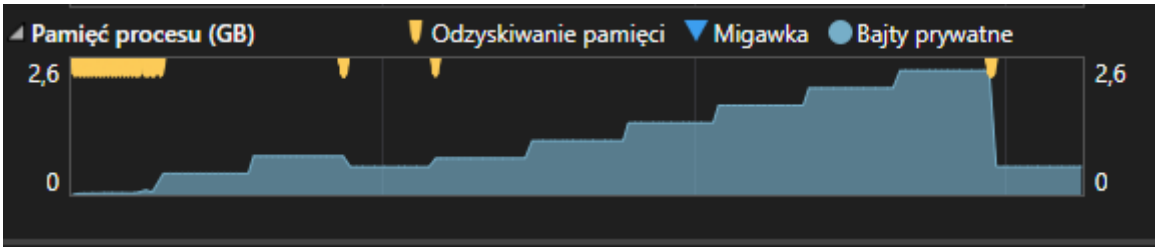
Treść pliku tsp.ini:
tsp_6_1.txt 100 132 [0 1 2 3 4 5 0]
tsp_6_2.txt 100 80 [0 5 1 2 3 4 0]
tsp_10.txt 100 212 [0 3 4 2 8 7 6 9 1 5 0]
tsp_12.txt 100 264 [0 1 8 4 6 2 11 9 7 5 3 10 0]
tsp_13.txt 50 269 [0 10 3 5 7 9 11 2 6 4 8 1 12 0]
tsp_14.txt 20 282 [0 10 3 5 7 9 13 11 2 6 4 8 1 12 0]
tsp_15.txt 10 291 [0 10 3 5 7 9 13 11 2 6 4 8 14 1 12 0]
gr17.tsp.txt 5 2085 [0 15 11 8 4 1 9 10 2 14 13 16 5 7 6 12 3]
gr21.tsp.txt 5 2707 [0 6 7 5 15 4 8 2 1 20 14 13 12 17 9 16 18 19 10 3 11 0]

output.csv

Każda instancja rozwiązywana była zgodnie z przyjętą liczbą jej wykonań, dla przykłądu tsp_10.txt została wykonana 100 razy. Do pliku wyjściowego był zapisywany: jako nagłówek, dane odczytywane z pliku .init dla konkretnej instancji, pod spodem czas wykonania [µs], optymalne rozwiązanie oraz ścieżka. Poniżej fragment pliku wyjściowego zapisanego w formacie .csv dla tsp_10.txt:

tsp_10.txt	100	212 [0 3 4 2 8 7 6 9 1 5 0]
360,3	212	[0 3 4 2 8 7 6 9 1 5 0]
360,9	212	[0 3 4 2 8 7 6 9 1 5 0]
354,6	212	[0 3 4 2 8 7 6 9 1 5 0]
357,4	212	[0 3 4 2 8 7 6 9 1 5 0]
362,7	212	[0 3 4 2 8 7 6 9 1 5 0]
356,4	212	[0 3 4 2 8 7 6 9 1 5 0]
357,8	212	[0 3 4 2 8 7 6 9 1 5 0]
357,1	212	[0 3 4 2 8 7 6 9 1 5 0]
353,6	212	[0 3 4 2 8 7 6 9 1 5 0]
354,2	212	[0 3 4 2 8 7 6 9 1 5 0]
354,6	212	[0 3 4 2 8 7 6 9 1 5 0]
356,8	212	[0 3 4 2 8 7 6 9 1 5 0]

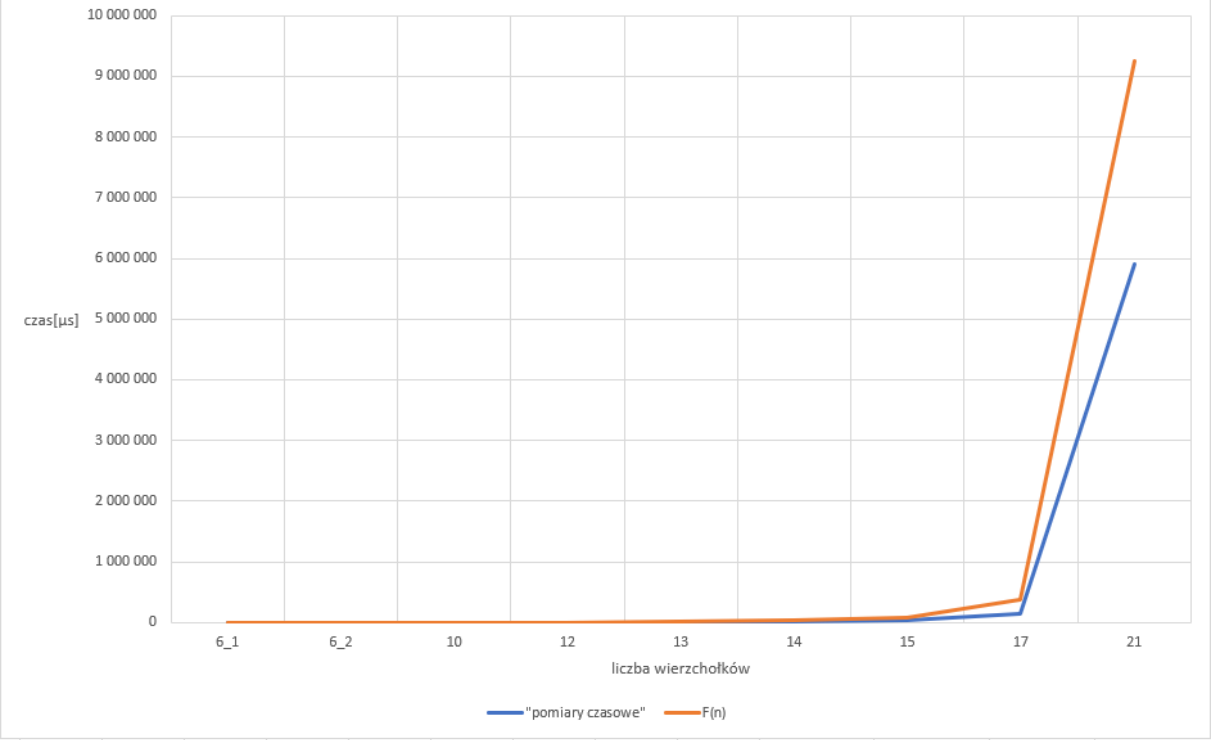
Do zbadania zależności wykorzystania pamięci od wielkości instancji była monitorowana na bieżąco przy pomocy systemowego Monitora zasobów(rysunek 3)



Rysunek 3: Zrzut ekranu przedstawiający pomiar wykorzystanej pamięci.

6. Wyniki

Wyniki zostały zgromadzone w pliku outputPEA.xlsx, opracowane z wykorzystaniem programu MS Excel. Na załączonych rysunku przedstawiona jest zależność czasu wykonywania się instancji(rysunek4). Na kolejnym rysunku (rysunek 5) została porównana złożoność czasowa algorytmu Held-Karp'a oraz Brute-Force'a dla przedstawionych wyżej instancji.

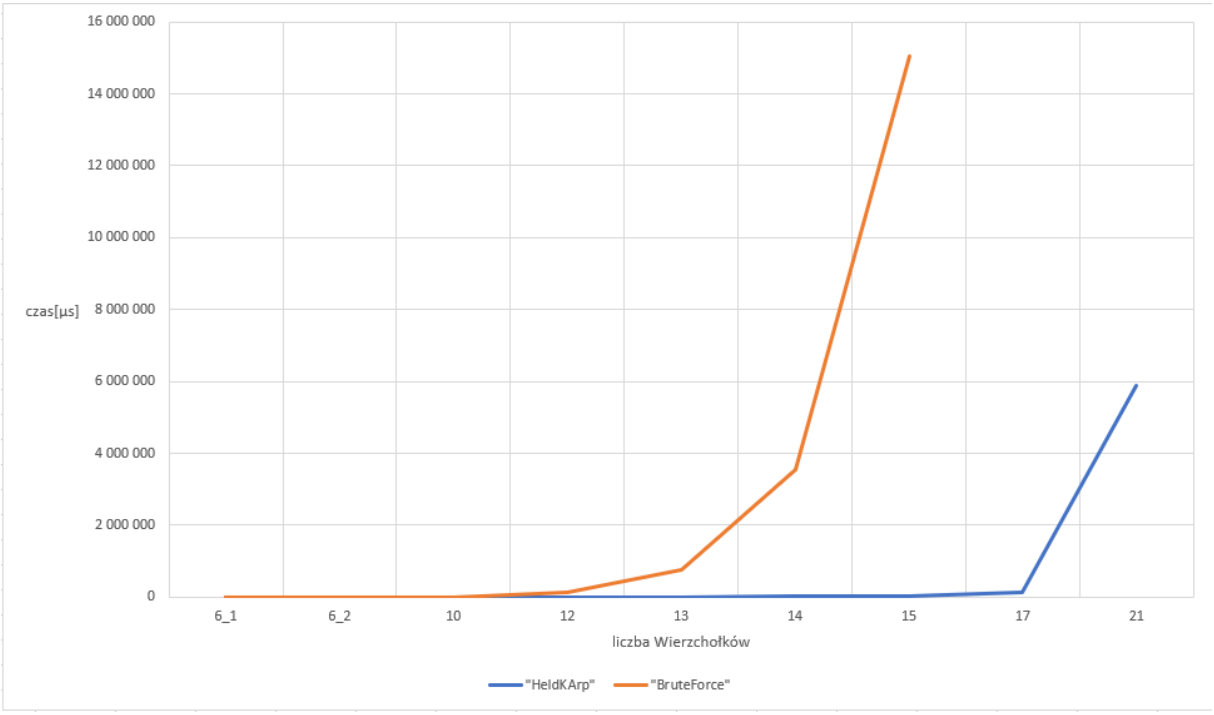


Rysunek 4: zależność czasu działania algorytmu od wielkości instancji

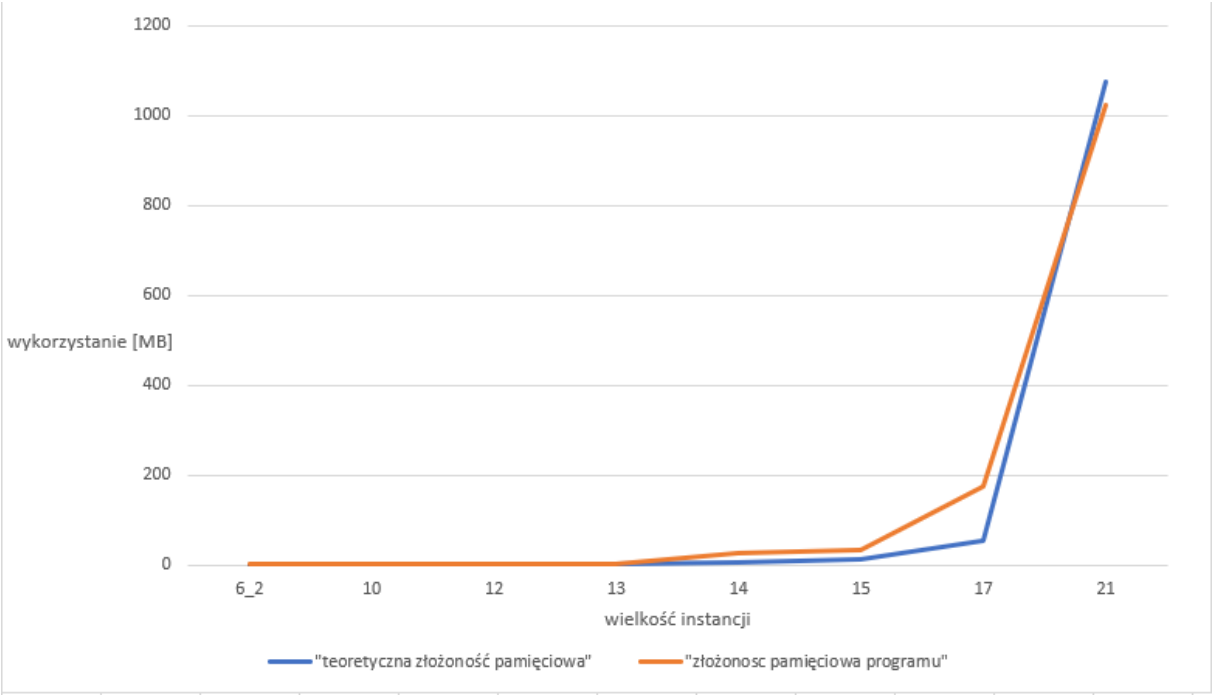
6_1	6_2	10	12	13	14	15	17	21
6	56	318	1781	4417	10814	31532	145912	5895838

Tabela1: tabela wykorzystana do tworzenia wykresu „pomiary czasowe" przedstawionego na „rysunek 4".

Tabela zawiera średnią czasów dla wszystkich mierzonych instancji.



Rysunek 5: porównanie złożoności czasowej algorytmu „BruteForce” oraz „HeldKarp”



Rysunek 6: Zależność zbadanej złożoności pamięciowej od teoretycznej, która wynosi $O(n*2^n)$.

7. Analiza wyników

Analizując rysunek 3, gdzie porównany jest wykres złożoności czasowej naszego algorytmu z wykresem funkcji: $f(n)=n^2*2^n$, możemy stwierdzić, że badanie potwierdza naszą teoretyczną złożoność czasową dla algorytmu Held-Karp'a, która wynosi $O(n^2*2^n)$.

Analizując porównanie złożoności czasowej algorytmów „BruteForce" oraz „HeldKarp"(rysunek 5), możemy wskazać, że algorytm HeldKarp'a pozwala na rozwiązywanie problemu komiwojażera dla większych instancji znacznie szybciej.

Na rysunku 6 zbadaliśmy zależność wykorzystanej pamięci w zależności do badanej instancji. Jak widać wykresy są bardzo zbliżone do siebie wyglądem co potwierdza teoretyczną złożoność obliczeniową.