Politechnika Wrocławska

Projektowanie efektywnych algorytmów

Prowadzący: Dr inż. Jarosław Mierzwa

Zadanie 1

Temat: Implementacja i analiza efektywności algorytmu podziału i ograniczeń i programowania dynamicznego.

Termin zajęć: PT 15:15 – 16:55

Autor sprawozdania: Karol Pastewski 252798

1. Wstęp teoretyczny

Rozpatrywanym problemem jest problem komiwojażera (*ang. travelling salesman problem, TSP*) w wersji asymetrycznej. Problem ten jest zagadnieniem optymalizacyjnym, polegającym na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Innymi słowami musimy znaleźć najkrótszą trasę przez wszystkie punkty, a następnie wrócić do początkowego punktu. W asymetrycznym TSP odległości między dwoma punktami mogą być różne.

Oszacowanie złożoności obliczeniowej dla TSP:

- Algorytm przeglądu zupełnego – O(N!)
- Algorytm podziału i ograniczeń – O(N!)

,gdzie N – wielkość rozpatrywanych punktów dla TSP

2. Przykład praktyczny

2.1. Algorytm podziału i ograniczeń

Załóżmy taką macierz sąsiedztwa:

	A	B	C	D
A	-1	5	12	8
B	9	-1	4	2
C	5	15	-1	7
D	7	3	10	-1

Krok 0: Ustawiamy finalCost = ∞.
Krok 1: Obliczamy lowerBound (suma minimalnych dróg), czyli lowerBound = 5 + 2 + 5 + 3 = 15
Krok 2: Zaczynając od punktu A tworzymy połączenie z kolejnym wierzchołkiem, czyli B i sprawdzamy, czy lowerBound jest mniejsze od finalCost (lowerBound = 15 < finalCost).
Krok 3: Jest, więc rozwijamy ścieżkę AB do kolejnego nieodwiedzonego punktu, czyli C. Sprawdzamy, czy lowerBound jest mniejsze od finalCost (lowerBound = 17 < finalCost).
Krok 4: Jest, więc rozwijamy ścieżkę ABC do kolejnego nieodwiedzonego punktu, czyli D. Sprawdzamy, czy lowerBound jest mniejsze od finalCost (lowerBound = 19 < finalCost).
Krok 5: Jest, więc rozwijamy ścieżkę ABCD, ale jest to ostatni poziom drzewa, więc wybieramy punkt początkowy, czyli A. Sprawdzamy, czy currentCost jest mniejsze od finalCost (currentCost = 23 < finalCost).
Krok 6: Jest, więc zmieniamy finalCost = currentCost.
Krok 7: Rozwijamy ścieżkę AB do kolejnego nieodwiedzonego punktu, czyli D. Sprawdzamy, czy lowerBound jest mniejsze od finalCost (lowerBound = 15 < finalCost = 23).
Krok 8: Jest, więc rozwijamy ścieżkę ABD do kolejnego nieodwiedzonego punktu, czyli C. (lowerBound = 22 < finalCost).
Krok 9: Łączymy ścieżkę ABDC z A i sprawdzamy currentCost = 22 < finalCost = 23.
Krok 10: Zmieniamy finalCost = currentCost.
Krok 11: Zaczynając od punktu A tworzymy połączenie z kolejnym wierzchołkiem, czyli C i sprawdzamy (lowerBound = 22 == finalCost = 22).
Krok 12: Nie przechodzimy dalej tą ścieżką, więc sprawdzamy kolejny punkt, czyli D. (lowerBound = 18 < finalCost = 22).
Krok 13: Rozwijamy ścieżkę AD do punktu B. (lowerBound = 18 < finalCost = 22).
Krok 14: Rozwijamy ścieżkę ADB do punktu C. (lowerBound = 20 < finalCost = 22).
Krok 15: Łączymy ścieżkę ADBC z punktem A i sprawdzamy currentCost = 20 < finalCost.
Krok 16: Zmieniamy finalCost = currentCost.
Krok 17: Rozwijamy ścieżkę AD do punktu C. (lowerBound = 25 > finalCost = 20).
Krok 18: Nie przechodzimy dalej tą ścieżką. Sprawdziliśmy wszystkie obiecujące ścieżki, więc wypisujemy najkrótszą ścieżkę oraz jej koszt.

Min ścieżka : ADBCA
Koszt: 20

3. Opis implementacji algorytmu

3.1. Algorytm przeglądu zupełnego

Wykorzystuje tablicę dynamiczną do przechowywania drogi. Algorytm działa rekurencyjnie, dopóki nie znajdzie wszystkich permutacji dostępnych dróg.

3.2. Algorytm podziału i ograniczeń

Zaimplementowany algorytm jest w wersji Depth First (najpierw w głąb) i przeszukuje kolejne gałęzie rekurencyjnie. do przechowywania obecnie przebytej drogi wykorzystuje tablicę dynamiczną.

Dla każdego punktu jest wyszukiwany najmniejszy koszt do sąsiadującego punktu. Początkowe dolne ograniczenie jest sumą tych minimalnych kosztów. Dla kolejnych rozpatrywanych gałęzi dolne ograniczenie jest sumą kosztu obecnie przebytej drogi oraz minimalnego kosztu do sąsiadującego punktu dla każdego nieodwiedzonego na obecnej drodze punktu.

4. Plan eksperymentu

4.1. Rozmiar używanych struktur danych

Odległości pomiędzy poszczególnymi punktami są przechowywane w tabeli dynamicznej o wielkości N na N, gdzie N to liczba rozpatrywanych punktów.

Obliczane optymalne ścieżki są przechowywane w tabeli dynamicznej o wielkości N + 1, gdzie N to liczba rozpatrywanych punktów.

Klasyczna implementacja algorytmu podziału i ograniczeń w wersji najpierw w głąb używa do przechowywania kolejnych rozpatrywanych gałęzi stosu, ale w tej wersji zastosowano rekurencję do przeszukiwania kolejnych gałęzi.

Rozpatrywana wielkość N dla poszczególnych algorytmów:

- Algorytm przeglądu zupełnego – $N \in \langle 6; 14 \rangle$
- Algorytm podziału i ograniczeń - $N \in \langle 6; 23 \rangle$

4.2. Sposób generowania danych

```
10  int **ArrayClass::createArray(int **array, int N) {
11      random_device rd;
12      mt19937 gen(rd());
13      uniform_int_distribution<int> distribution(1, 99);
14      auto random = bind(distribution, gen);
15      array = new int *[N];
16      for (int i = 0; i < N; i++) {
17          array[i] = new int[N];
18          for (int j = 0; j < N; j++) {
19              if (i == j) array[i][j] = -1;
20              else {
21                  array[i][j] = random();
22              }
23          }
24      }
25      return array;
26  }
```

Rysunek 1 Fragment kodu generujący dane

Pomiar czasowy jest sprawdzany dla 100 różnych instancji. Wartości są losowane za pomocą maszyny losującej mt19937 i są w przedziale od 1 do 99. Wartości po przekątnej są ustawiane na -1.

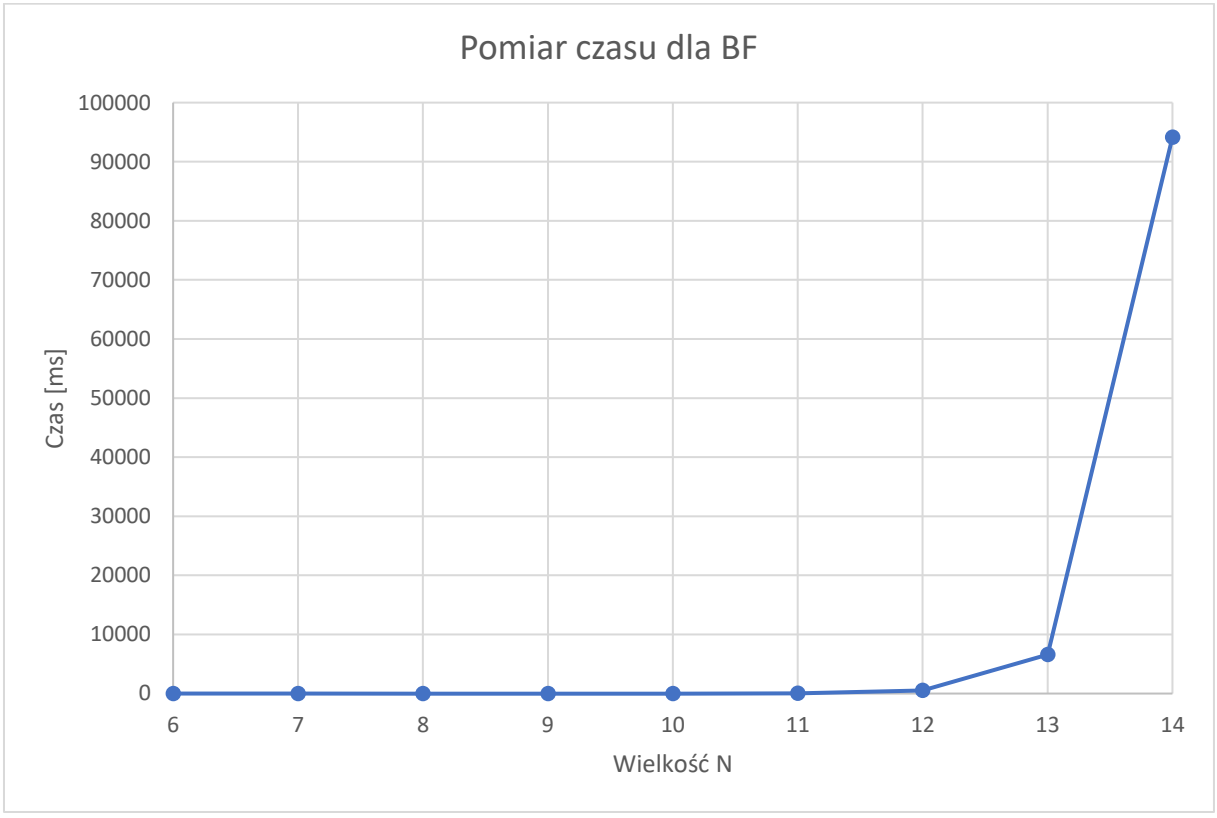
4.3. Metoda pomiaru czasu

Pomiar czasu został zmierzony funkcją QueryPerformanceCounter(). Zwracany stan licznika przez tą funkcję jest następnie dzielony przez częstotliwość otrzymaną z funkcji QueryPerformanceFrequency(), aby otrzymać pomiar w jednostkach mikrosekundowych czasu. Przedstawione wyniki na wykresach są w milisekundach z dokładnością do trzech miejsc po przecinku.

5. Wyniki eksperymentów

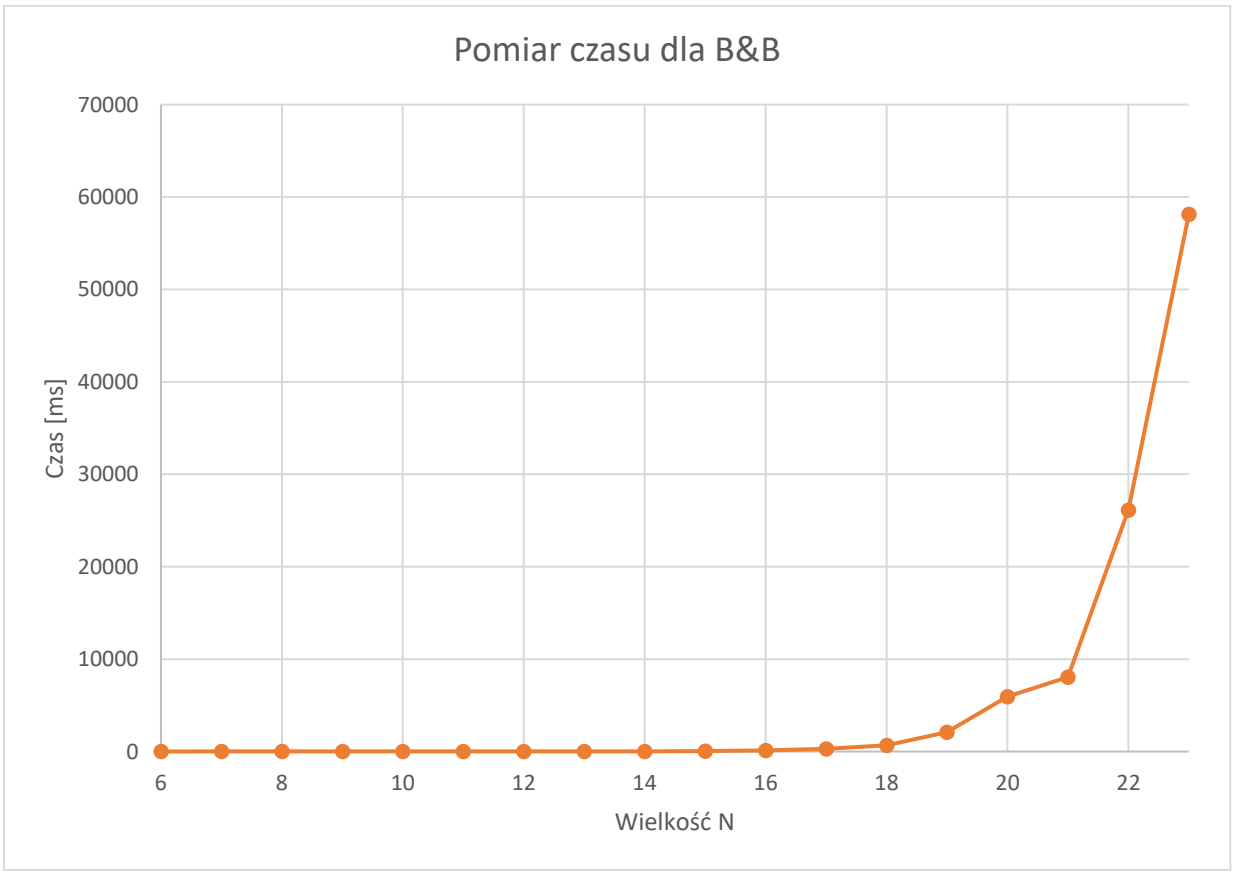
5.1. Wyniki pomiarów algorytmu zupełnego

N	Czas [ms]
6	0,001
7	0,009
8	0,060
9	0,508
10	4,428
11	46,120
12	522,492
13	6 585,972
14	94 137,580



5.2. Wyniki pomiarów algorytmu podziału i ograniczeń

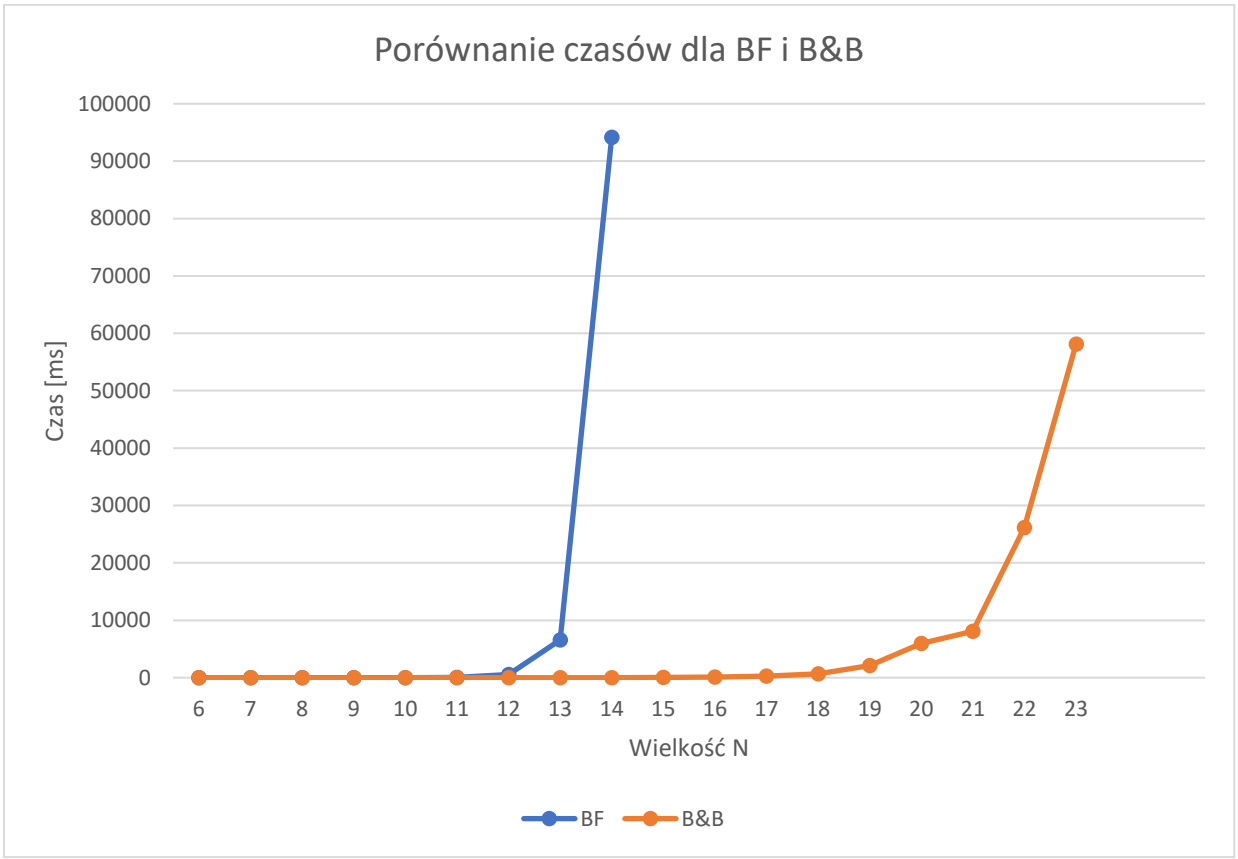
N	Czas [ms]	Indywidualne pomiary poniżej 120s [%]
6	0,006	100
7	0,018	100
8	0,047	100
9	0,120	100
10	0,389	100
11	1,111	100
12	2,858	100
13	7,027	100
14	15,584	100
15	50,622	100
16	112,597	100
17	278,895	100
18	665,959	100
19	2 108,163	100
20	5 938,538	100
21	8 057,258	100
22	26 145,655	98
23	58 123,673	85



5.3. Porównanie wyników wszystkich algorytmów

Brak danych oznacza, że algorytm potrzebował średnio więcej niż 120s do wykonania obliczeń.

N	Zupełny [ms]	Podziału i ograniczeń [ms]
6	0,001	0,006
7	0,009	0,018
8	0,06	0,047
9	0,508	0,120
10	4,428	0,389
11	46,12	1,111
12	522,492	2,858
13	6 585,972	7,027
14	94 137,58	15,584
15	Brak danych	50,622
16	Brak danych	112,597
17	Brak danych	278,895
18	Brak danych	665,959
19	Brak danych	2 108,163
20	Brak danych	5 938,538
21	Brak danych	8 057,258
22	Brak danych	26 145,655
23	Brak danych	58 123,673



6. Wnioski

Maksymalne N, dla którego średni czas jest mniejszy od 120s:

- Algorytm przeszukiwania zupełnego – N = 14
- Algorytm podziału i ograniczeń – N = 24

Pomimo tego, że teoretyczna złożoność obu algorytmów jest taka sama, to algorytm podziału i ograniczeń jest szybszy of przeszukiwania zupełnego. Jest to spowodowane samym działaniem algorytmu, który ogranicza nam rozpatrywaną przestrzeń do gałęzi, które wyglądają na obiecujące (mogą prowadzić do optymalnego rozwiązania).

Należy zaznaczyć, że wyniki dla poszczególnych instancji dla przeszukiwania zupełnego były bardzo do siebie zbliżone, bo za każdym razem program musiał wykonać i sprawdzić wszystkie permutacje ścieżki. W algorytmie podziału i ograniczeń wyniki czasowe były mocno zależne od rodzaju instancji, co dobrze obrazuje procentowa wartość pomiarów o czasie poniżej 120s dla N = 23. Widzimy, że 15% wyników było nawet dwukrotnie większych od otrzymanej średniej ze wszystkich 100 instancji. Algorytm podziału i ograniczeń jest średnio szybszy od przeszukiwania zupełnego, ale nie musi być, dla indywidualnej instancji.

7. Źródła

- https://en.wikipedia.org/wiki/Travelling_salesman_problem
- https://en.wikipedia.org/wiki/Brute-force_search
- https://en.wikipedia.org/wiki/Branch_and_bound
- http://lcm.csa.iisc.ernet.in/dsa/node187.html