Projektowanie Efektywnych Algorytmów

Zadanie projektowe nr 1

Mikołaj Pastuszek 259138

Czwartek 11:15

Prowadzący: Mgr inż. Antoni Sterna

1. Cel projektu

   Zadanie projektowe opierało się na implementacji oraz analizie efektywności algorytmu przeglądu zupełnego (BF) i podziału i ograniczeń (B & B) dla problemu komiwojażera (TSP).

2. Wstęp teoretyczny

   2.1. Opis problemu komiwojażera

   Biorąc pod uwagę zestaw miast i odległość między każdą parą miast, problem polega na znalezieniu najkrótszej możliwej trasy, która odwiedza każde miasto dokładnie raz i wraca do punktu początkowego. W projekcie rozwiązywaliśmy asymetryczny problem komiwojażera. Jest to trudniejsza wersja tego problemu gdyż długość ścieżki z punktu A do B i z B do A mogą się od siebie różnić.

   2.2. Opis algorytmów oraz określenie ich złożoności

   a) Przegląd zupełny

   Przegląd zupełny oblicza i porównuje wszystkie możliwe permutacje tras lub ścieżek w celu określenia najkrótszego unikalnego rozwiązania.

   Złożoność czasowa tego algorytmu to O(n!) gdzie n to liczba wierzchołków.

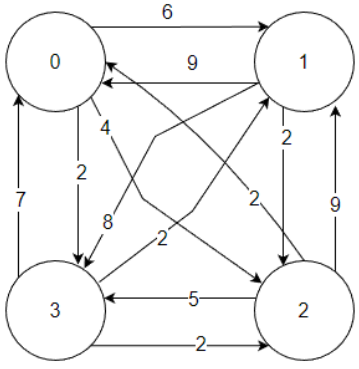
   b) Metoda podziału i ograniczeń

   Metoda podziału i ograniczeń składa się z systematycznego wyliczania rozwiązań kandydujących. Algorytm eksploruje gałęzie drzewa, które reprezentują podzbiory zbioru rozwiązań. Przed wyliczeniem rozwiązań kandydujących gałęzi, gałąź jest sprawdzana pod kątem górnych i dolnych oszacowanych granic optymalnego rozwiązania i jest odrzucana, jeśli nie może dać lepszego rozwiązania niż najlepsze znalezione dotychczas przez algorytm.

   Złożoność czasowa tego algorytmu to w najlepszym przypadku O(n) a w najgorszym O(n!) gdzie n to liczba wierzchołków.

3. Opis implementacji algorytmu oraz przykład praktyczny

   Wszystkie przykłady będą odbywały się na tym przykładzie:



3.1. Przegląd zupełny

Algorytm:

1. Przyjęcie początkowego wierzchołka jako 0. Drogi minimalnej jako maksymalną wartość zmiennej integer (2147483647).

2. Obliczenie dla każdej permutacji drogi i porównywanie jej z drogą minimalną. Gdy wartość drogi permutacji jest mniejsza od minimalnej to zamienić wartość minimalną oraz jej drogę.

3. Wypisanie ścieżki minimalnej

Dla przykładu:

```
Doszlo do zaminay. Obecne min to:
 Dla permutacji 0 1 2 3 0 droga to 20
Doszlo do zaminay. Obecne min to:
 Dla permutacji 0 1 3 2 0 droga to 18
 Dla permutacji 0 2 1 3 0 droga to 28
 Dla permutacji 0 2 3 1 0 droga to 20
Doszlo do zaminay. Obecne min to:
 Dla permutacji 0 3 1 2 0 droga to 8
 Dla permutacji 0 3 2 1 0 droga to 22
Minimalna droga to: 8
0 3 1 2 0
```

3.2. Metoda podziału i ograniczeń

Korzystając z tej metody należy wybrać strategię odwiedzania wierzchołków drzewa. Na potrzebę tego projektu wyróżnię dwie metody odwiedzania wierzchołków:

a) Least Cost (Best Search)

Graf jest przedstawiony w formie macierzy (dwuwymiarowej tablicy).

Węzły drzewa są reprezentowane przez strukturę która posiada 3 parametry: koszt drogi, wierzchołek oraz ścieżkę.

Kolejka priorytetowa wyznacza nam kolejność odwiedzania węzłów drzewa. Kolejka wyznacza kolejność na podstawie kosztu drogi.

Algorytm:

1.Funkcja znajduje UB.

2.Dodaje wierzchołek 0 do kolejki.

3.Dalsza część wykonuje się do momentu gdy kolejka priorytetowa nie będzie pusta.

4.Gdy długość ścieżki najlepszego węzła jest wielkości macierzy to oznacz że przeszedł już wszystkie wierzchołki. Wtedy sprawdzamy czy może się stać nowym UB. Nie ważne od wyniku węzeł i tak zostanie usunięty z kolejki

5.Następnie sprawdzamy czy LB najlepszego węzła jest mniejsza od obecnego UB. Gdyby była większa oznaczało by to że nie ma sensu dalej rozwijać tej gałęzi. Jeśli LB < HB to dodajemy dzieci badanego najlepszego węzła.

6.Usuwamy najlepszy węzeł

7.Gdy kolejka jest pusta następuje wypisanie ścieżki i jej kosztu

Dla przykładu:

```
Obecne UB to: 20
 Ze sciezka: 0 1 2 3 0
Zawartosc kolejki piorytetowej to: 0
Dodajemy dziedzi wezla z wierzcholkiem 0
Zawartosc kolejki piorytetowej to: 3 2 1
Dodajemy dziedzi wezla z wierzcholkiem 3
Zawartosc kolejki piorytetowej to: 2 2 1 1
Dodajemy dziedzi wezla z wierzcholkiem 2
Zawartosc kolejki piorytetowej to: 2 1 1 3 1
Zawartosc kolejki piorytetowej to: 1 1 3 1
Dodajemy dziedzi wezla z wierzcholkiem 1
Zawartosc kolejki piorytetowej to: 1 2 3 1
Dodajemy dziedzi wezla z wierzcholkiem 1
Zawartosc kolejki piorytetowej to: 2 2 3 1 3
Obecne UB to: 8
 Ze sciezka: 0 3 1 2 0
Zawartosc kolejki piorytetowej to: 2 3 1 3
Zawartosc kolejki piorytetowej to: 3 1 3
Zawartosc kolejki piorytetowej to: 1 3
Zawartosc kolejki piorytetowej to: 3
Droga to: 8
0 3 1 2 0
```

b) Depth First Search

Graf jest przedstawiony w formie macierzy (dwuwymiarowej tablicy).

Węzły drzewa są reprezentowane przez strukturę która posiada 3 parametry: koszt drogi, wierzchołek oraz ścieżkę.

Lista wyznacza nam kolejność odwiedzania węzłów drzewa.

1.Ustawiamy wartość drogi UB na 2147483647.

2.Dodaje wierzchołek 0 do kolejki.

3.Dalsza część wykonuje się do momentu gdy lista nie będzie pusta.

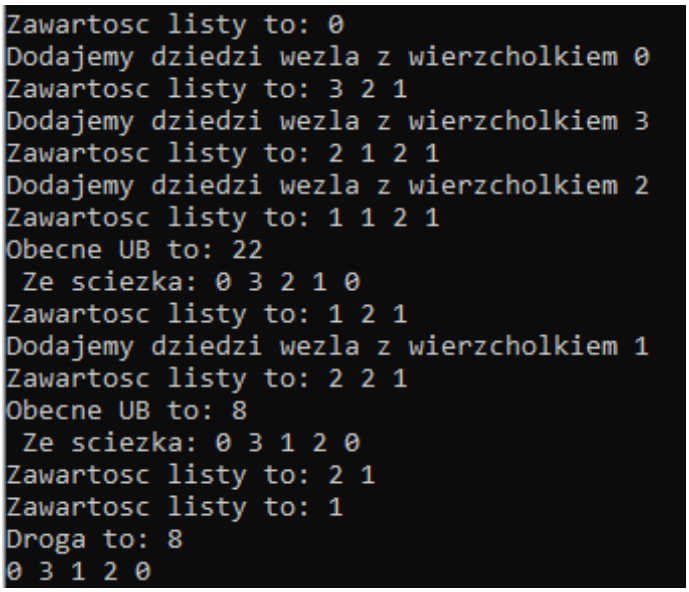
4.Gdy długość ścieżki pierwszego węzła w liście jest wielkości macierzy to oznacz że przeszedł już wszystkie wierzchołki. Wtedy sprawdzamy czy może się stać nowym UB. Nie ważne od wyniku węzeł i tak zostanie usunięty z kolejki

5.Następnie sprawdzamy czy LB pierwszego węzła w liście jest mniejsza od obecnego UB. Gdyby była większa oznaczało by to że nie ma sensu dalej rozwijać tej gałęzi. Jeśli LB < HB to dodajemy dzieci badanego najlepszego węzła.

6.Usuwamy najlepszy węzeł

7.Gdy kolejka jest pusta następuje wypisanie ścieżki i jej kosztu

Dla przykładu:



4. Plan eksperymentu

Dla każdej wartości N (rozmiaru problemu) zostanie wygenerowane po 100 losowych instancji problemu oraz obliczony średni czas rozwiązania dla poszczególnych algorytmów. Długości dróg będą losowane z zakresu <0,10000>

4.1. Używanie dane

Dla przeglądu zupełnego N = {5, 6, 7, 8, 9, 10, 11}

Dla metody podziału i ograniczeń N = {5, 7, 9, 11, 13, 15, 17}

4.2. Sposób generowanie danych

Do generowania macierzy została stworzona funkcja randomGraf() w klasie TestTime. Funckja przyjmuje dwa parametry: wielkość macierzy (ilość wierzchołków) oraz maksymalną długość drogi. Zwraca ona wskaźnik na daną macierz.

```
int** TestTime::randomGraf(int size, int maxRandom)
{
    int** matrix = new int* [size];
    for (int i = 0; i < size; i++)
        matrix[i] = new int[size];


    random_device rd; // non-deterministic generator
    mt19937 gen(rd()); // random engine seeded with rd()
    uniform_int_distribution<> dist(0, maxRandom); // distribute results between
                                                   // 1 and 1000000 inclusive

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (i!=j)
                matrix[i][j] = dist(gen); //loswanie liczby w zakresie <0,maxRandom)
            else
                matrix[i][j] = 0;
        }
    }
    return matrix;
}
```

Metoda była inspirowana kodem znajdującą się na stronie:

http://antoni.sterna.staff.iiar.pwr.wroc.pl/pea/PEA_random.pdf

4.3. Metoda pomiaru czasu

Do liczenia czasu dla każdego algorytmu została stworzona inna funkcja w klasie TestTime. Wszystkie jednak przyjmują te same parametry: wielkość macierzy (ilość wierzchołków), maksymalną długość drogi oraz ilość powtórzeń pomiaru czasu dla losowo wygenerowanej macierzy. Wypisują one uśredniony czas rozwiązania problemu w trzech dokładnościach: s, ns i µs. Funkcje te różnią się jedynie wywoływaną metodą dla której średni czas działania jest wyliczany. Przykładowa funkcja dla przeglądu zupełnego wraz z funkcją pomocniczą:

```
void TestTime::countTimeBF(int size, int maxRandom, int numberOfRepetitions)
{
    long long int frequency, start, elapsed = 0;
    QueryPerformanceFrequency((LARGE_INTEGER*)&frequency);

    for (int i = 0; i < numberOfRepetitions; i++)
    {
        BF* tsp = new BF(randomGraf(size, maxRandom), size);
        start = read_QPC();
        tsp->bf();
        elapsed += read_QPC() - start;
        delete tsp;
    }

    elapsed /= numberOfRepetitions;

    cout << "Time [s] = " << fixed << setprecision(3) << (float)elapsed /
        frequency << endl;
    cout << "Time [ms] = " << setprecision(0) << (1000.0 * elapsed) /
        frequency << endl;
    cout << "Time [us] = " << setprecision(0) << (1000000.0 * elapsed) /
        frequency << endl << endl;
}
```

```
long long int TestTime::read_QPC()
{
    LARGE_INTEGER count;
    QueryPerformanceCounter(&count);
    return((long long int)count.QuadPart);
}
```

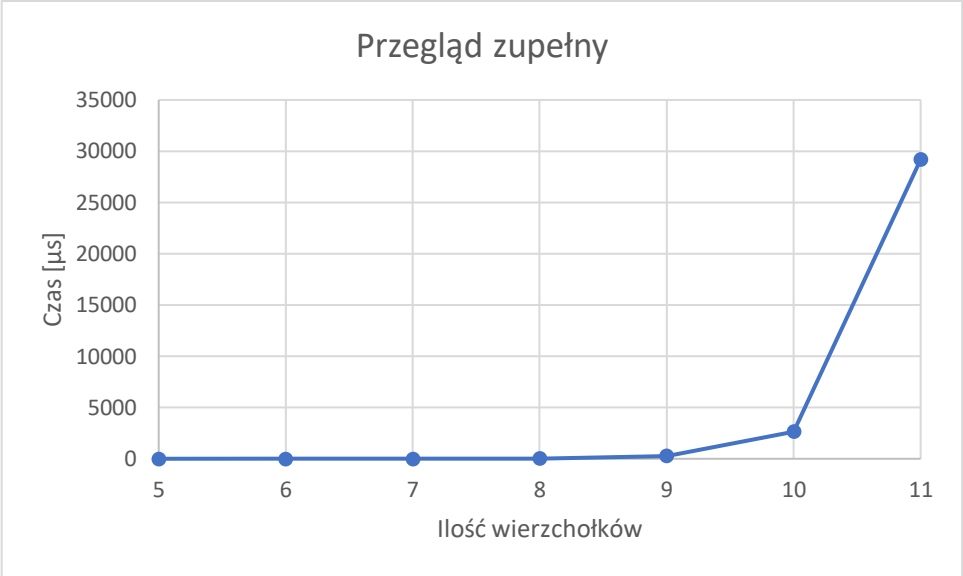
Metoda była inspirowana kodem znajdującą się na stronie:

http://antoni.sterna.staff.iiar.pwr.wroc.pl/pea/PEA_time.pdf

5. Wyniki eksperymentu (tabele i wykresy)

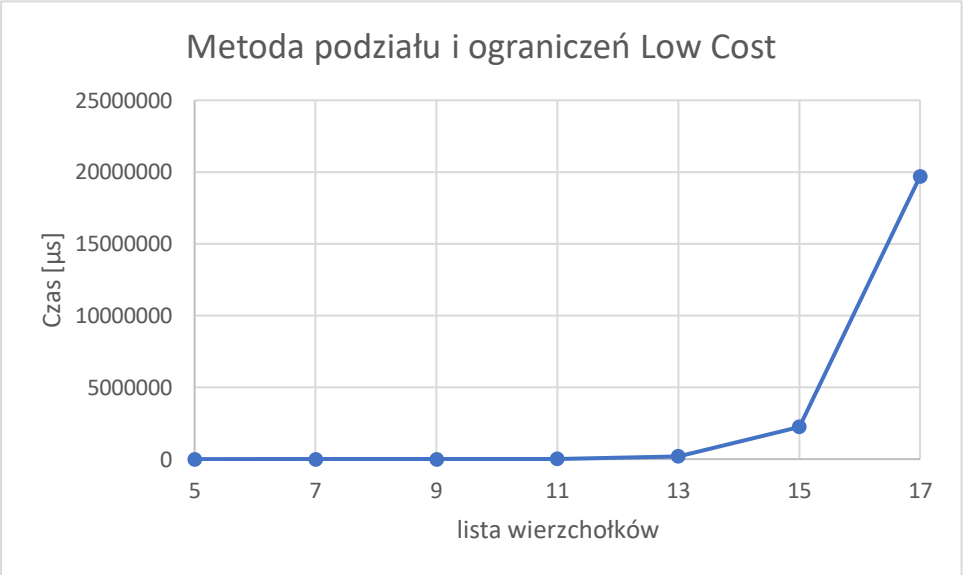
5.1. Przegląd zupełny

N	5	6	7	8	9	10	11
czas[µs]	1	5	7	36	276	2659	29178



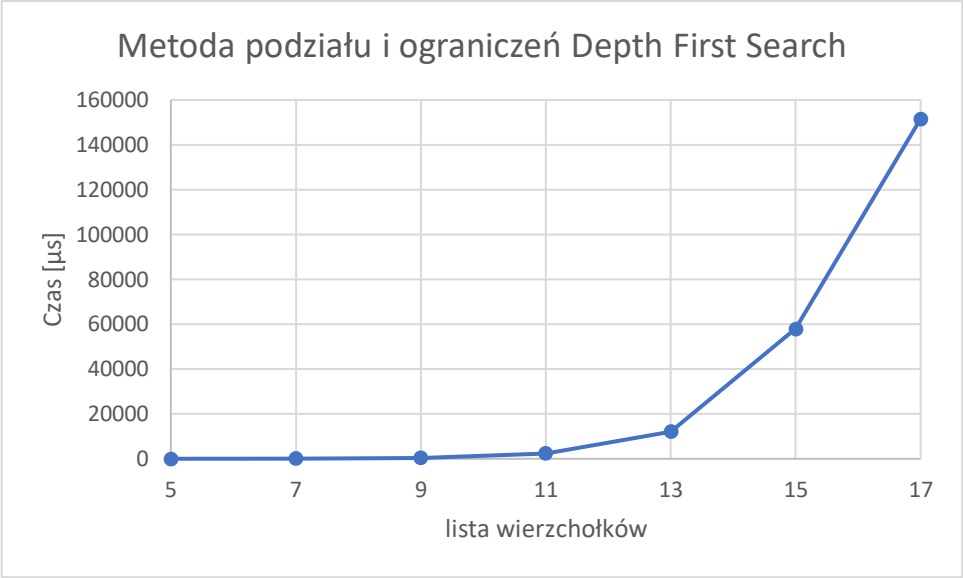
5.2. Metoda podziału i ograniczeń Least Cost (Best Search)

N	5	7	9	11	13	15	17
czas[µs]	12	153	1711	16099	193994	2238902	19696347



5.3. Metoda podziału i ograniczeń Depth First Search

N	5	7	9	11	13	15	17
czas[µs]	14	83	399	2352	12180	57960	151599



5.4. Wspólne zestawienie wyników dla metody podziału i ograniczeń

N	5	7	9	11	13	15	17
czas[µs] dla LC	12	153	1711	16099	193994	2238902	19696347
czas[µs] dla DFS	14	83	399	2352	12180	57960	151599

6. Wnioski

   Dla przeglądu zupełnego otrzymaliśmy wyniki zgodnie z oczekiwaną złożonością. Dla bardzo małych problemów (max. 9) okazał się najlepszy.

   Metoda podziału i ograniczeń ze strategią odwiedzania węzłów Best Search okazała się mało efektowna. Było to spowodowane średnią implementacją algorytmu. Miedzy innymi kolejka priorytetowa powinna np. porównywać LB dla każdego węzła a nie drogę przebytą, jak to było robione.

   Najlepszą metodą okazało się metoda podziału i ograniczeń ze strategią odwiedzania węzłów Depth First Search. Czasy były przyzwoite i w porównaniu do poprzedniej strategii nie pobierała ona dużo pamięci.

7. Kod źródłowy

   Kod źródłowy został umieszczony na dysku Google pod adresem:
   https://drive.google.com/drive/folders/16pUQAKRbLdTeZlQ1CZ56nWzkYNu8dtKz