Sprawozdanie - Projektowanie efektywnych algorytmów

Maksim Birszel nr indeksu 241353

6 listopada 2019

Zadanie 1 - Metoda przeszukiwania zupełnego oraz podziału i ograniczeń

Prowadzący:
Dr. Inż
Zbigniew Buchalski

Termin zajęć:
Wtorek 9:15

1 Wstęp teoretyczny

Problem komiwojażera (ang. travelling salesman problem, TSP) – zagadnienie optymalizacyjne, polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym.
Nazwa pochodzi od typowej ilustracji problemu, przedstawiającej go z punktu widzenia wędrownego sprzedawcy (komiwojażera): dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość / cena podróży / czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej / najtańszej / najszybszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej się w określonym punkcie. Jest on zaliczany to klasy problemów NP-trudnych.
Problem dzielimy na symetryczny oraz asymetryczny.
Symetryczny - odległości z miasta A do B są równe odległością z B do A
Asymetryczny - odległości z A do B różnią się od odległości z B do A

2 Przegląd zupełny - brute force

Przegląd zupełny (bruteforce, metoda siłowa) – metoda nieefektywna obliczeniowo (ale optymalna, gdyż znajduje rozwiązanie najlepsze)
Algorytm polega na sprawdzeniu wszystkich dostępnych ścieżek, a następnie wybraniu najkrótszej z nich.

Według literatury złożoność obliczeniowa takiego rozwiązania to O(n!).

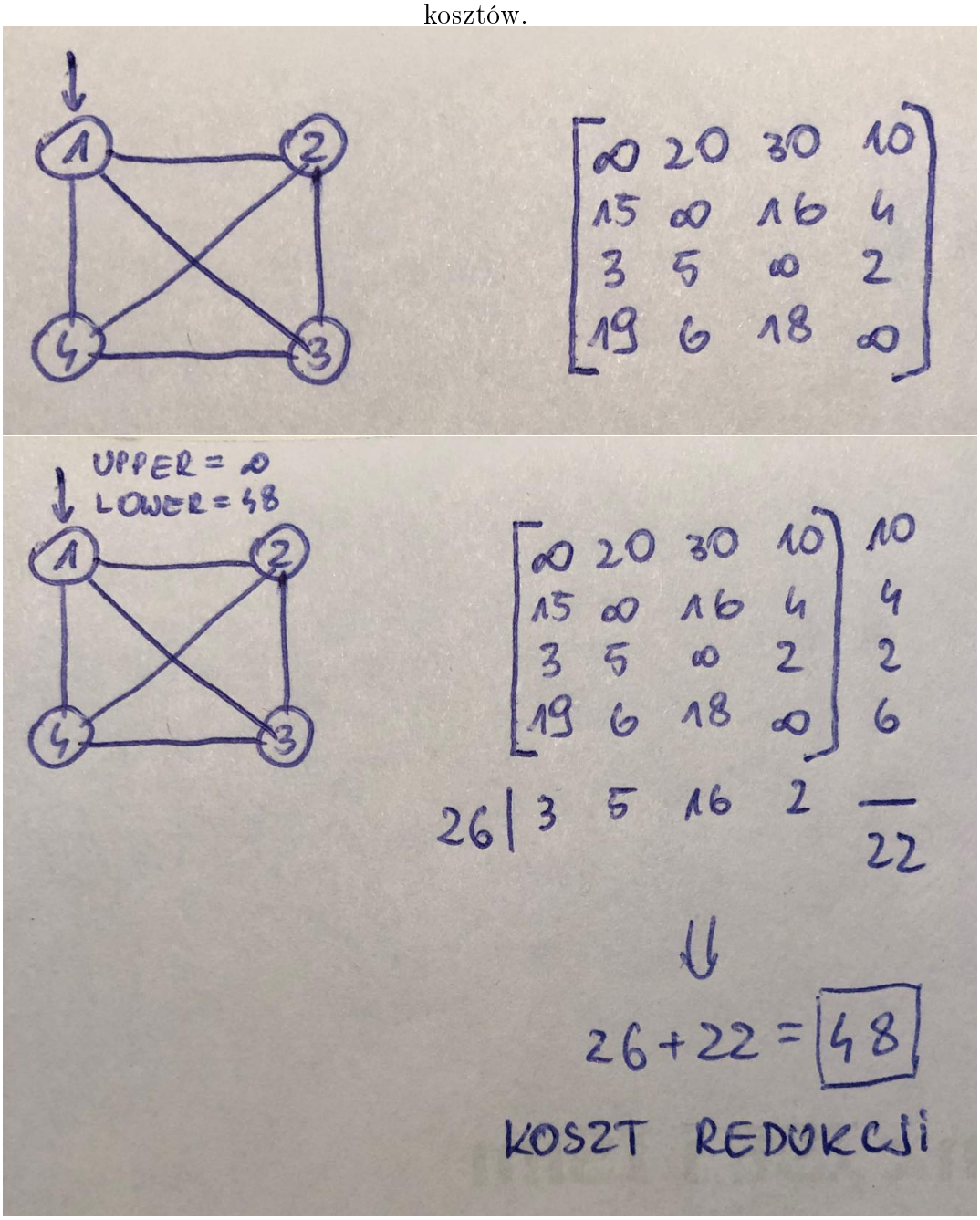
3 Podział i ograniczenia - Branch & Bound

Metoda podziału i ograniczeń (ang. branch-and-bound) opiera się na przeszukiwaniu (najczęściej w głąb) drzewa reprezentującego przestrzeń rozwiązań problemu. Stosowane w tej metodzie odcięcia redukują liczbę przeszukiwanych węzłów (wykładniczą względem rozmiaru instancji)
Metoda jest skomponowana, z grubsza rzecz ujmując z dwóch podstawowych procedur:
rozgałęzianie (ang. branching) - dzielenie zbioru rozwiązań reprezentowanego przez węzeł na rozłączne podzbiory, reprezentowane przez następników tego węzła
ograniczanie (ang. bounding) - pomijanie w przeszukiwaniu tych gałęzi drzewa, o których wiadomo, że nie zawierają optymalnego rozwiązania w swoich liściach

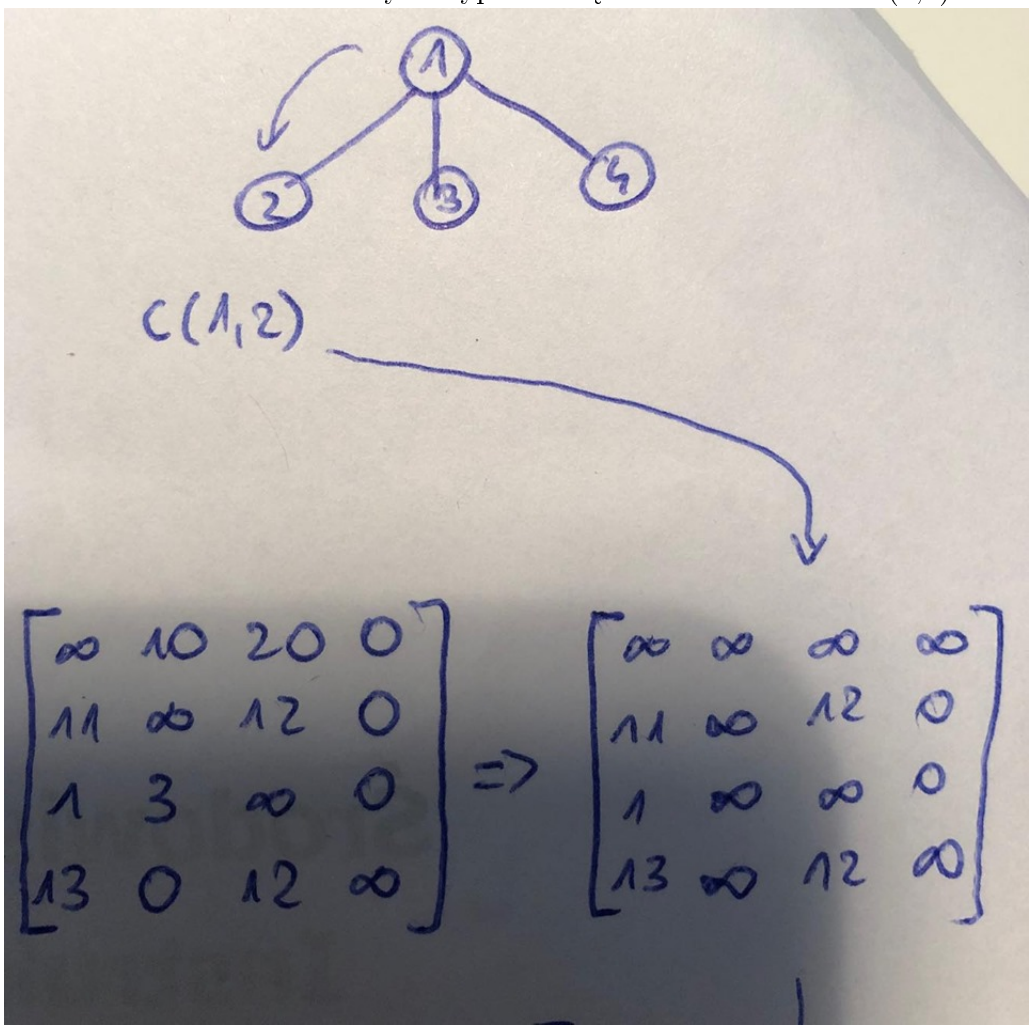
Według literatury najgorsza możliwa złożoność obliczeniowa takiego rozwiązania to O(n!), czyli taka sama jak brute force. Jest to jednak dla najgorsze przypadku, dlatego w większości z nich będzie ona lepsza.

3.1 Przykład praktyczny

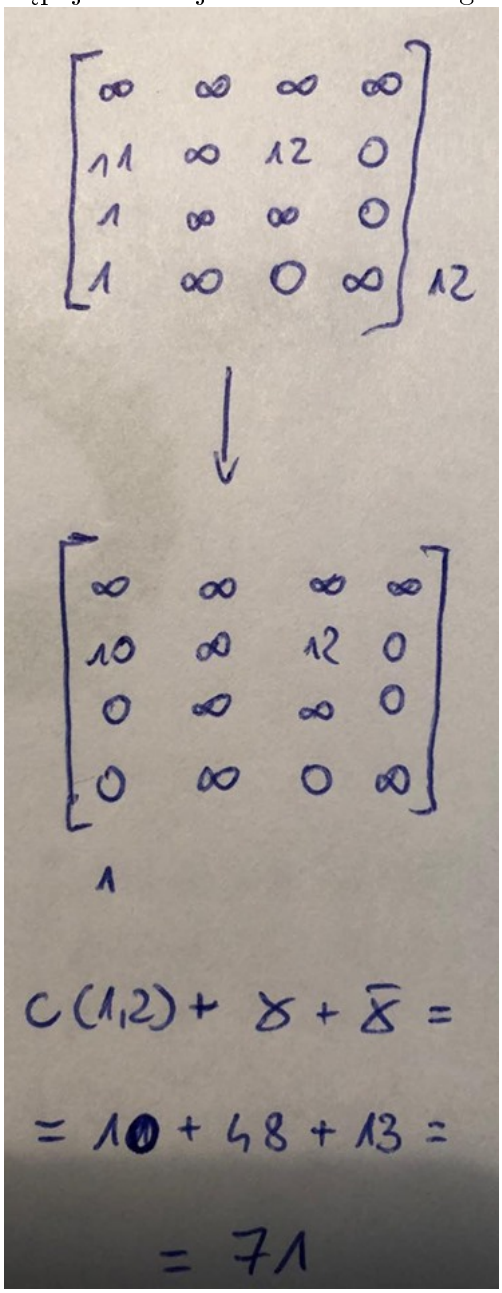
Zdjęcie przedstawia nasz graf z poszczególnymi ścieżkami oraz macierz kosztów.



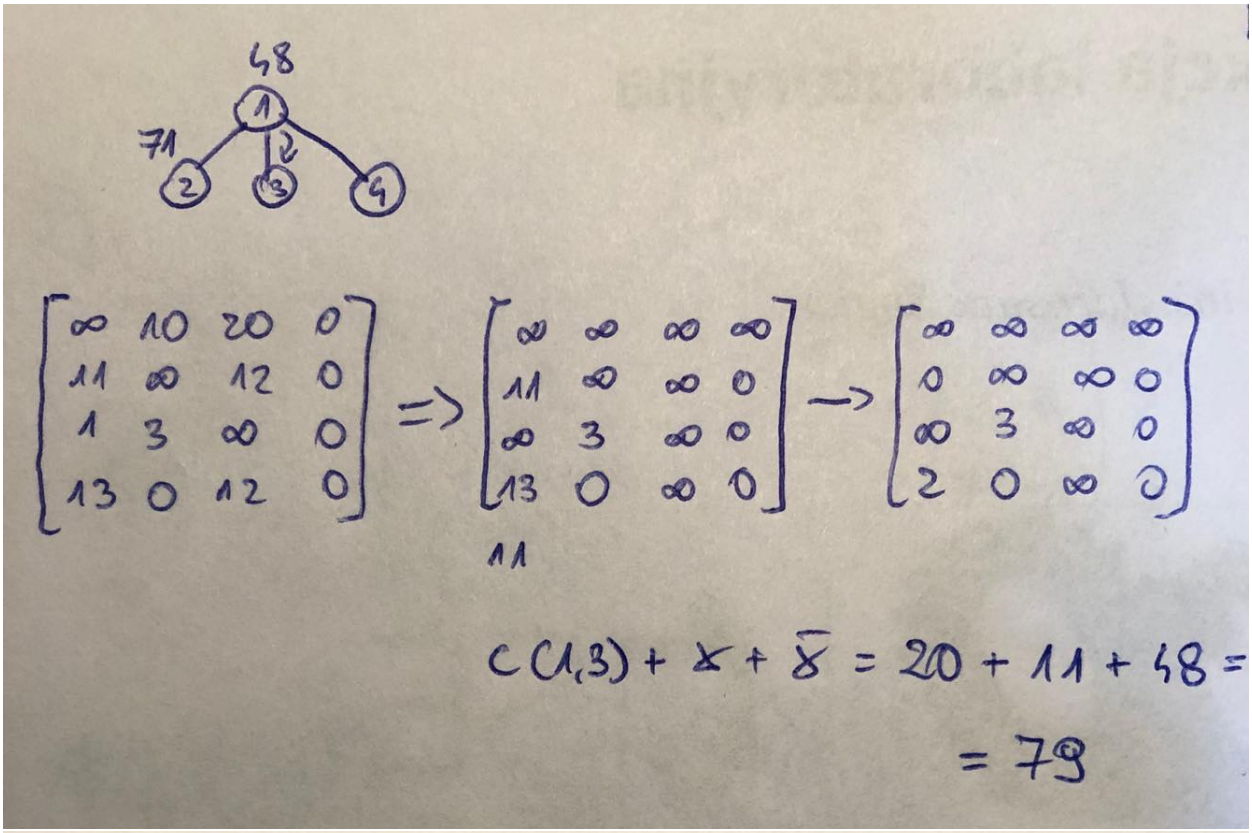
W pierwszym kroku obliczamy dolne ograniczenie dla naszego wierzchołka startowego oraz redukujemy macierz kosztów. Następnie obliczamy wartości ograniczenia dla poszczególnych wierzchołków majacych bezposrednie polaczenie z wierzcholkiem startowym oraz redukujemy ich macierze kosztów. W tym wypadku będzie to wierzchołek c(1,2).

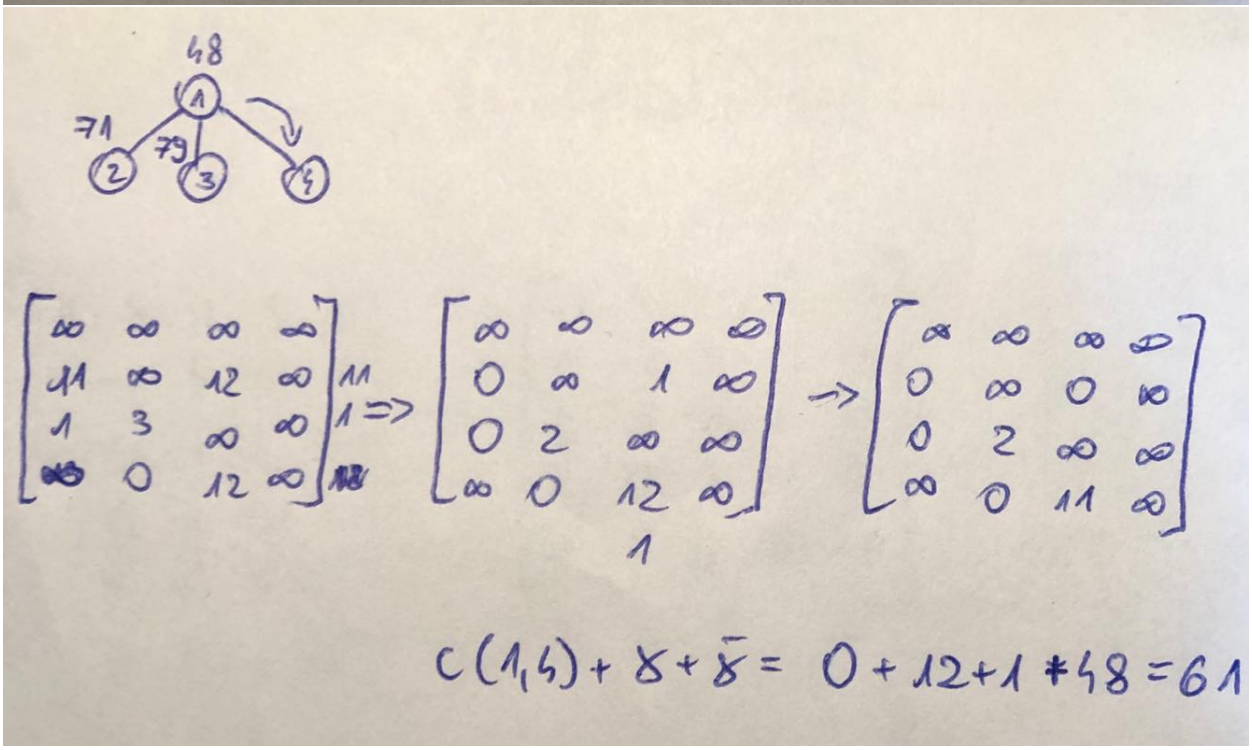


Tutaj następuje redukcja oraz obliczenie ograniczenia.

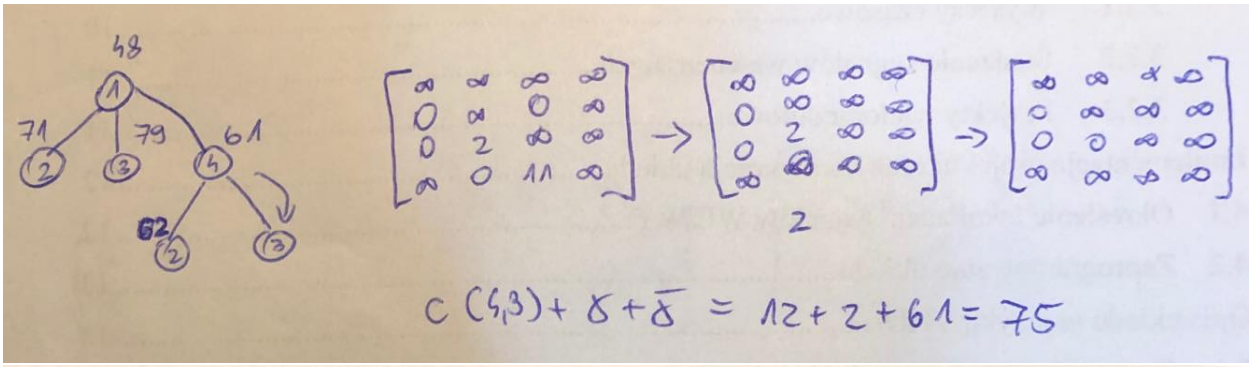


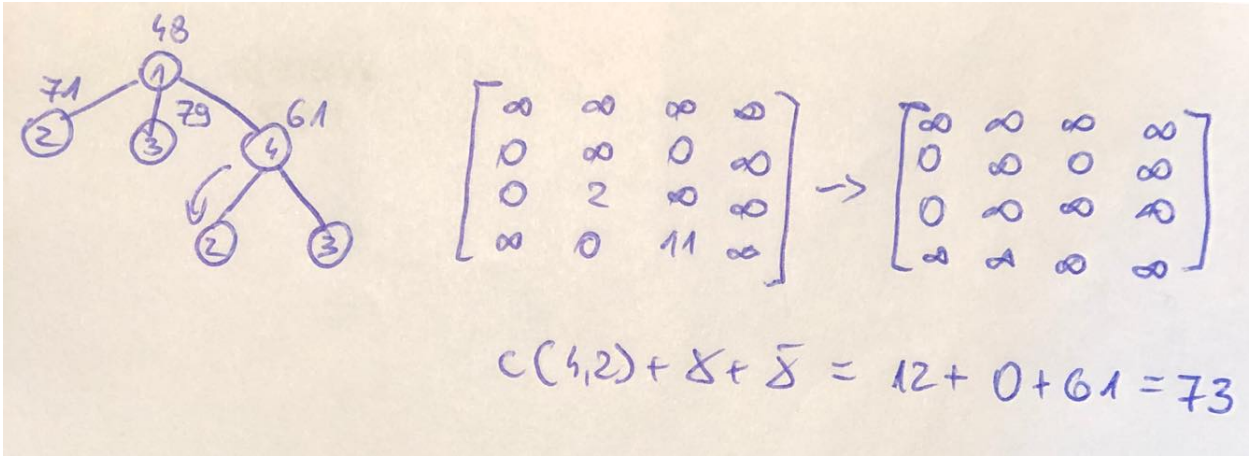
Powtórzenie poprzednich czynności dla kolejnych dwóch wierzchołków



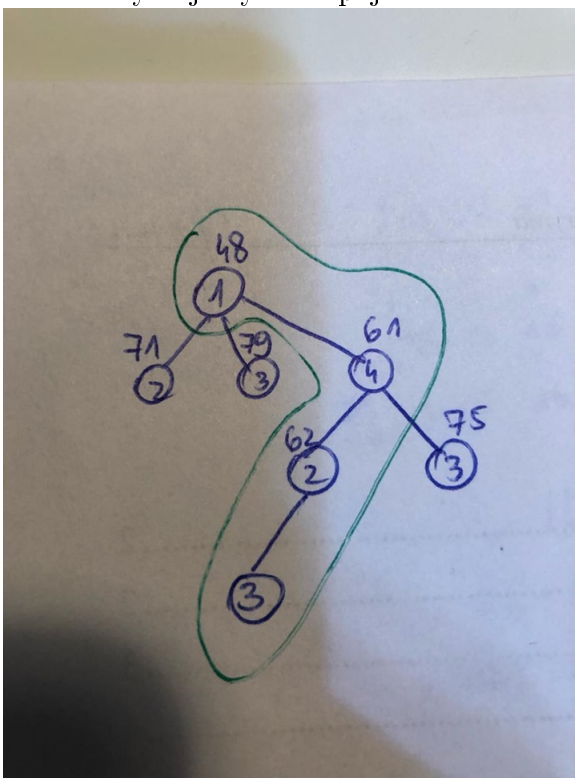


Następnie wybierany jest ten wierzchołek którego ograniczenie dolne jest najniższe (w tym wypadku wierzcholek nr 4) i jest on rozwijany w taki sam sposób jak wcześniej.





Spośród nich jest wybierany również ten o najnizszym koszcie i w taki sposob otrzymujemy nastepujace rozwiazanie:



3.2 Opis implementacji algorytmu

Początkowa redukcja macierzy odbywa się w następujący sposób:

```
for (int i = 0; i < matrixSize; i++)
{
for (int j = 0; j < matrixSize; j++)
{
if (matrix[i][j] == -1)
continue;

if (matrix[i][j] < currentMinimum)
currentMinimum = matrix[i][j];
}

rowMinimum.push_back(currentMinimum);
currentMinimum = INT_MAX;
}

for (int i = 0; i < matrixSize; i++)
{
for (int j = 0; j < matrixSize; j++)
{
if (matrix[i][j] == -1)
continue;

matrix[i][j] -= rowMinimum[i];
}
}

for (int a = 0; a < rowMinimum.size(); a++)
rowBound += rowMinimum[a];
```

W danym przykładzie redukowana jest macierz po wierszach. W podobny sposób odbywa się to podczas redukcji po kolumnach.

Następnie redukowane są kolejne macierze kolejnych wierzchołków oraz obliczane jest ograniczenie każdego z nich:

```
int rowBound = 0;
int columnBound = 0;
//vector<int> rowMinimum;
int* rowMinimum = new int[matrixSize];
//vector<int> columnMinimum;
int* columnMinimum = new int[matrixSize];
int currentMinimum = INT_MAX;
```

```
int bound = matrix[beginVertex][endVertex];

// inicjalizacja kolumny endVertex, wiersza beginVertex oraz c(endVertex, beginVertex)
for (int i = 0; i < matrixSize; i++)
{
matrix[i][endVertex] = -1;
matrix[beginVertex][i] = -1;
}
matrix[endVertex][beginVertex] = -1;

for (int i = 0; i < matrixSize; i++)
{
if (i == beginVertex)
{
rowMinimum[i] = 0;
continue;
}

for (int j = 0; j < matrixSize; j++)
{
if (matrix[i][j] == -1)
continue;

if (matrix[i][j] < currentMinimum)
currentMinimum = matrix[i][j];

if (currentMinimum == 0)
break;
}

rowMinimum[i] = currentMinimum;
currentMinimum = INT_MAX;

for (int a = 0; a < matrixSize; a++)
columnBound += columnMinimum[a];

bound += columnBound + rowBound;
}
```

W dalszej części rekurencyjnie zostaje wywołana funkcja ComputePath(), która wywołuje poszczególne operacje na macierzach dla najmniejszych wierzcholkow:

```
// poczatkowa redukcja macierzy kosztów
StartMatrixReducing();
```

```
// tablica przechowujaca aktualna sciezke
int* currentPath = new int[matrixSize];

// tablica przechowujaca odwiedzone wierzcholki
int* visited = new int[matrixSize + 1];
for (int k = 0; k < matrixSize + 1; k++)
visited[k] = 0;

// zmienna przechowujaca wage najmniejszego wierzcholka
int lowestBound = INT_MAX;
int lowestVertex = 0;
int iteration = matrixSize;

// algorytm
for (int x = 0; x < matrixSize; x++)
{
for (int i = 1; i < iteration; i++)
{
int** tempMatrix = CopyMatrix(matrix);

// jesli nie znajdzie elementu i w tabeli visited
if (!FindElement(visited, i))
{
int bound = matrixReducing(currentTreeLevel, i, tempMatrix);

if (bound < lowestBound)
{
lowestBound = bound;
lowestVertex = i;
}
}

delete[] tempMatrix;
}
visited[currentTreeLevel] = lowestVertex;
currentTreeLevel += 1;
iteration -= 1;
}
```

3.3 Plan eksperymentu

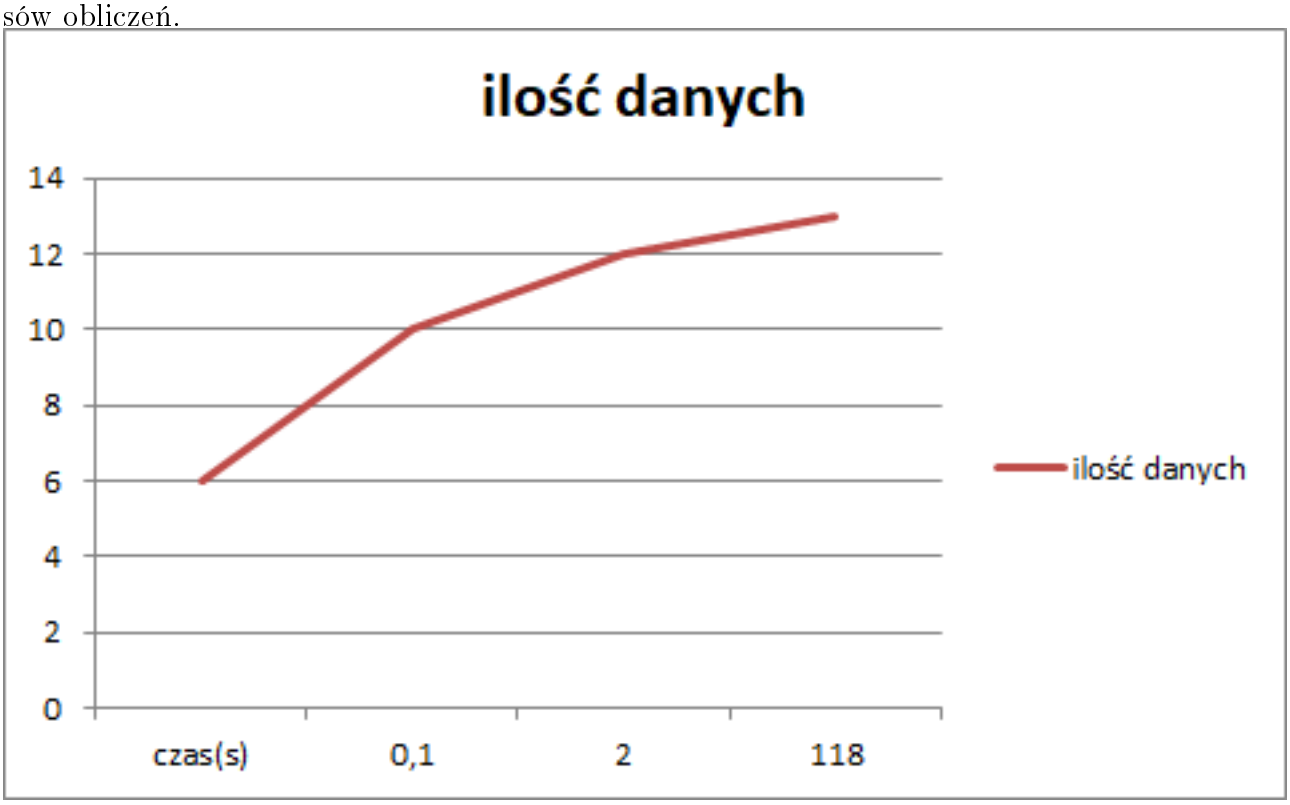
Dane oraz rozmiary testowanych danych pochodziły ze strony Pana Jarosława Mierzwy. Podobnie użyta metoda pomiaru czasu - funkcja QueryPerformanceCounter.

3.4 Wyniki eksperymentu

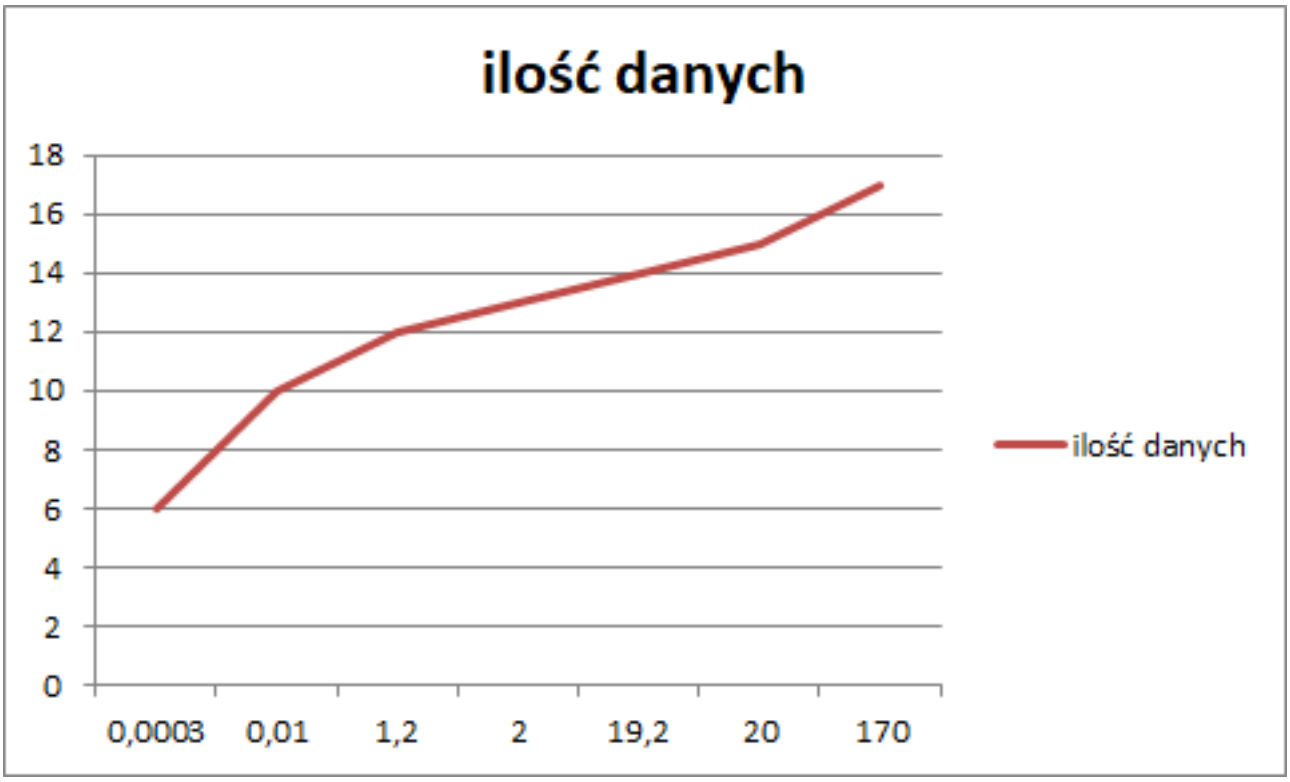
Czas obliczenia najkrótszej ścieżki w sekundach w zależności od ilości miast dla:

3.5 Brute force

W metodzie brute force została uwzględniona w obliczeniach maksymalna instancja problemu wielkości 13, gdyż większe wymagały zbyt dużych czasów obliczeń.



3.6 B & B



4 Wnioski

Jak widać po powyższych obliczeniach, metoda brute force jest w miare efektywna dla problemów instancji 10 i mniejszych. Każda powyższa zajmuje znacząco za dużą ilość czasu. Być może jest to spowodowane słabą implementacją (użycie rekurencyjnych funkcji klasy vector) oraz mocą obliczeniową komputera, na którym algorytm był testowany.
W przypadku B & B, widać że algorytm działa dużo szybciej podczas testowania większych instancji problemu. Rozwijanie tylko jednego wierzchołka na każdym poziomie poddrzewa znacząco zmniejsza czas obliczeń. Nawet mimo wielu operacji na macierzach, wielokrotnym: przeglądaniu, kopiowaniu, zamianie elementów oraz wielu innych jest to dużo szybsze niż przeglądanie wszystkich możliwych opcji.