**Politechnika Wrocławska**

**Wydział Elektroniki**

PROJEKT Z ZARZĄDZANIA W SYSTEMACH I SIECIACH KOMPUTEROWYCH

Zastosowanie algorytmów wielowątkowych do rozwiązywania Symetrycznego Problemu Komiwojażera

|  |  |
| --- | --- |
| Autor:  Katarzyna Brzezińska, 235879  Eliza Kalata, 235543 | Prowadzący zajęcia:  Dr inż. Robert Wójcik, W4/K-9 |
|  | Ocena pracy: |

Spis treści

[Spis rysunków 3](#_Toc26746732)

[Spis tabel 4](#_Toc26746733)

[Spis listingów 5](#_Toc26746734)

[1. Wstęp 6](#_Toc26746735)

[1.1. Cel projektu 6](#_Toc26746736)

[1.2. Zakres projektu 6](#_Toc26746737)

[2. Sformułowanie problemu 7](#_Toc26746738)

[2.1. Podstawowe założenia 7](#_Toc26746739)

[2.2. Zastosowany algorytm i metody rozwiązywania 7](#_Toc26746740)

[2.3. Analiza złożoności obliczeniowej problemu 7](#_Toc26746741)

[2.4. Przykład praktyczny algorytmu wykorzysującego programowanie dynamiczne: 8](#_Toc26746742)

[2.5. Technologie i narzędzia implementacji 9](#_Toc26746743)

[2.6. Harmonogram prac 9](#_Toc26746744)

[3. Projekt 10](#_Toc26746745)

[3.1. Zastosowane technologie i narzędzia projektowe 10](#_Toc26746746)

[3.2. Struktura programu 10](#_Toc26746747)

[3.3. Koncepcja działania algorytmu 10](#_Toc26746748)

[3.4. Diagram klas 10](#_Toc26746749)

[3.5. Struktura danych wejściowych 11](#_Toc26746750)

[3.6. Struktura wyników 11](#_Toc26746751)

[4. Implementacja systemu 12](#_Toc26746752)

[4.1. Wybrane klasy 12](#_Toc26746753)

[4.2. Realizacja algorytmu 12](#_Toc26746754)

[4.3. Mechanizm odczytu danych 13](#_Toc26746755)

[4.4. Zapis i prezentacja wyników 14](#_Toc26746756)

[5. Testowanie i ocena wydajności 16](#_Toc26746757)

[5.1. Poprawność 16](#_Toc26746758)

[5.2. Analiza czasów 17](#_Toc26746759)

[5.2.1. Porównanie czasów 17](#_Toc26746760)

[5.2.2. Weryfikacja z najlepszym uzyskanym wynikiem 17](#_Toc26746761)

[5.3. Wnioski z testów 18](#_Toc26746762)

[6. Podsumowanie 20](#_Toc26746763)

[Literatura 21](#_Toc26746764)

# Spis rysunków

[Rysunek 1 Diagram Klas 10](#_Toc26742008)

[Rysunek 2 Macierz kosztów z pliku tekstowego 11](#_Toc26742009)

[Rysunek 3 Funkcj dynamic implementująca algorytm wykorzystujące programowanie dynamiczne 13](#_Toc26742010)

[Rysunek 4 Metoda pobierz klasy K 14](#_Toc26742011)

[Rysunek 5 Widok prezentacji wyników 15](#_Toc26742012)

[Rysunek 6.Wykres zależności czasu od liczby wątków dla 17 miast 17](#_Toc26742013)

[Rysunek 7.Wykres zależności czasu od liczby wątków dla 20 miast 17](#_Toc26742014)

[Rysunek 8Wykres procentowego spadku czasu wykonywania dla 17 miast 18](#_Toc26742015)

[Rysunek 9 Wykres procentowego spadku czasu wykonywania dla 20 miast 18](#_Toc26742016)

# Spis tabel

[Tabela 1 Harmonogram 9](#_Toc26742017)

[Tabela 2 Uśrednienia wyników dla 17 miast 16](#_Toc26742018)

[Tabela 3 Uśrednienia wyników dla 20 miast 16](#_Toc26742019)

# Spis listingów

# Wstęp

## Cel projektu

Celem projektu było stworzenie oraz implementacja systemu realizującego obliczenia i znajdującego rozwiązanie problemu komiwojażera [1] przy użyciu wątków. Kryterium oceny jakości trasy jest suma wag krawędzi, tworzących w grafie pełen cykl Hamiltona [2].

## Zakres projektu

Swoim zakresem obejmuje projekt aplikacji desktopowej realizującej zarządzanie użyciem wątków w celu przyspieszenia obliczeń wykonywanego algorytmu na problemie komiwojażera. Projekt ma za zadanie umożliwiać zmianę liczby wątków a także instancji problemu – liczby miast. Otrzymane wyniki pomiarowe zostaną poddane analizie, a także porównane i zweryfikowany zgodnie z przyjętymi założeniami projektowymi.

# Sformułowanie problemu

## Podstawowe założenia

Problem optymalizacyjny: Jak powinien zaplanować trasę podróży komiwojażer, aby w sumie przebył możliwie najkrótszą drogę?

Problem komiwojażera jest problemem NP – zupełnym. Mając pełny graf skierowany G=(V,E;w) ,gdzie |V| =n , a w(e) wyznacza wagę krawędzi e. Problem polega na znalezieniu cyklu Hamiltona o najmniejszej wadze w G.

Symetryczny Problem Komiwojażera będzie rozwiązywany przy uwzględnieniu następujących założeń [3]:

* zbudowanie ważnego, skierowanego grafu, którego wierzchołki są miastami;
* połączenie pary wierzchołków krawędziami – odcinkiem trasy, który będzie odzwierciedlał odległość pomiędzy dwoma miastami;
* przyjęcie za punkt początkowy trasy wierzchołek (miasto) o numerze 1;
* krawędzie opatrzone są dodatnimi wagami;
* odwiedzenie wszystkich miast odpowiada cyklowi, który przechodzi przez każdy wierzchołek grafu dokładnie raz, tworząc pełny cykl Hamiltona o minimalnej sumie wag krawędzi;

## Zastosowany algorytm i metody rozwiązywania

Zastosowanym algorytmem do rozwiązania problemu zarządzania zasobami w systemach i  sieciach komputerowych w problemie komiwojażera jest Programowania Dynamicznego [1]. Jet to technika projektowania algorytmów polegająca na rozwiązywaniu podproblemów i zapamiętywaniu ich wyników. Rozwiązanie przebiega poprzez dzielenie problemu na mniejsze podproblemy. Na mniejszych instancjach wykonywane są obliczenia, których wyniki są zapisywane w tabeli, dzięki czemu w przypadku natrafienia na ten sam podproblem nie istnieje konieczność ponownego rozwiązywania. W celu usprawnienia obliczeń, zostaną zastosowane wątki do rozwiązywania poszczególnych instancji.

## Analiza złożoności obliczeniowej problemu

Zastosowaną metodą rozwiązania problemu jest wykorzystanie Programowania Dynamicznego [1] do implementacji algorytmu. Programowanie Dynamiczne jest metodą rozwiązywania zadań optymalizacyjnych, które na mocy pewnych własności, można sformułować jako poszukiwanie ciągu decyzji. Zostanie zastosowany algorytm Bellmana - Helda-Karpa ma złożoność czasową O(n^2 \*2^n) [5].

## Przykład praktyczny algorytmu wykorzysującego programowanie dynamiczne:

matrix(0,{a,b})-zaczynamy trasę od 0 i może przejść przez a do b lub przez b do a.

Dodaje wagę drogi powrotnej do wierzchołka początkowego w naszym przypadku jest nim 0

matrix ( 2, {3} ) =  (2,3) + matrix (3, 0 )     4+10=14

matrix ( 3, {2} ) =  (3,2) + matrix (2, 0 )     1+14=15

matrix ( 1, {3} ) =  (1,3) + matrix (3, 0 )     6+10=16

matrix ( 3, {1} ) =  (3,1) + matrix (1, 0 )     28+26 = 54

matrix ( 1, {2} ) =  (1,2) + matrix (2, 0 )     25+14 = 39

matrix ( 2, {1} ) =  (2,1) + matrix (1, 0 )     37+26=63

matrix (1, {2,3} ) = minimum of

= { (1,2) + matrix (2, {3} ) 25+14=39

= { (1,3) + matrix {3, {2} ) 6+15=21

matrix (2, {1,3} ) = minimum of

= { (2,1) + matrix (1, {3} ) 37+16=53

= { (2,3) + v {3, {1} ) 4+54=58

v (3, {1,2} ) = minimum of

= { (3,1) + matrix (1, {2} ) 28+39=67

= { (3,2) + matrix {3, {1} ) 1+54=55

matrix( 0, {1,2,3} ) = minimum of

= { (0,1) + matrix (1, {2,3} ) 33+21=54

= { 0,2) + matrix (2, {1,3} ) 6+53=59

= { (0,3) + matrix (3, {1,2} ) 15+55=70

Tras:0->1->3->2->0, waga: 54

## Technologie i narzędzia implementacji

Program desktopowy realizujący algorytm zostanie napisany w języku C++, przy wykorzystaniu bibliotek STL [6]. Projekt będzie realizowany w środowisku Microsoft Visual Studio 2013 [7].

## Harmonogram prac

Tabela 1 Harmonogram

|  |  |
| --- | --- |
| Numer zajęć: | Zakres prac |
| 1 | Opis założeń projektu |
| 2 | Implementacja algorytmów |
| 3 | Zrobienie testów i przedstawienie wyników |
| 4 | Napisanie sprawozdania |

# Projekt

## Zastosowane technologie i narzędzia projektowe

Projekt został wykonany w środowisku programistycznym Visual Studio 13, napisany w języku C++.

## Struktura programu

Ponieważ za zadanie mieliśmy zrealizowanie pojedynczego algorytmu, postawiłyśmy na prostą strukturę oprogramowania. Stworzone są dwie klasy: K i Pr.

## Koncepcja działania algorytmu

Algorytm wykorzystuje maske bitów oraz rekurencje do znalezienia optymalnego rozwiązania, co w naszym przypadku jest drogą o jak najmniejszym koszcie przebycia.

## Diagram klas

Obraz zawierający zrzut ekranu

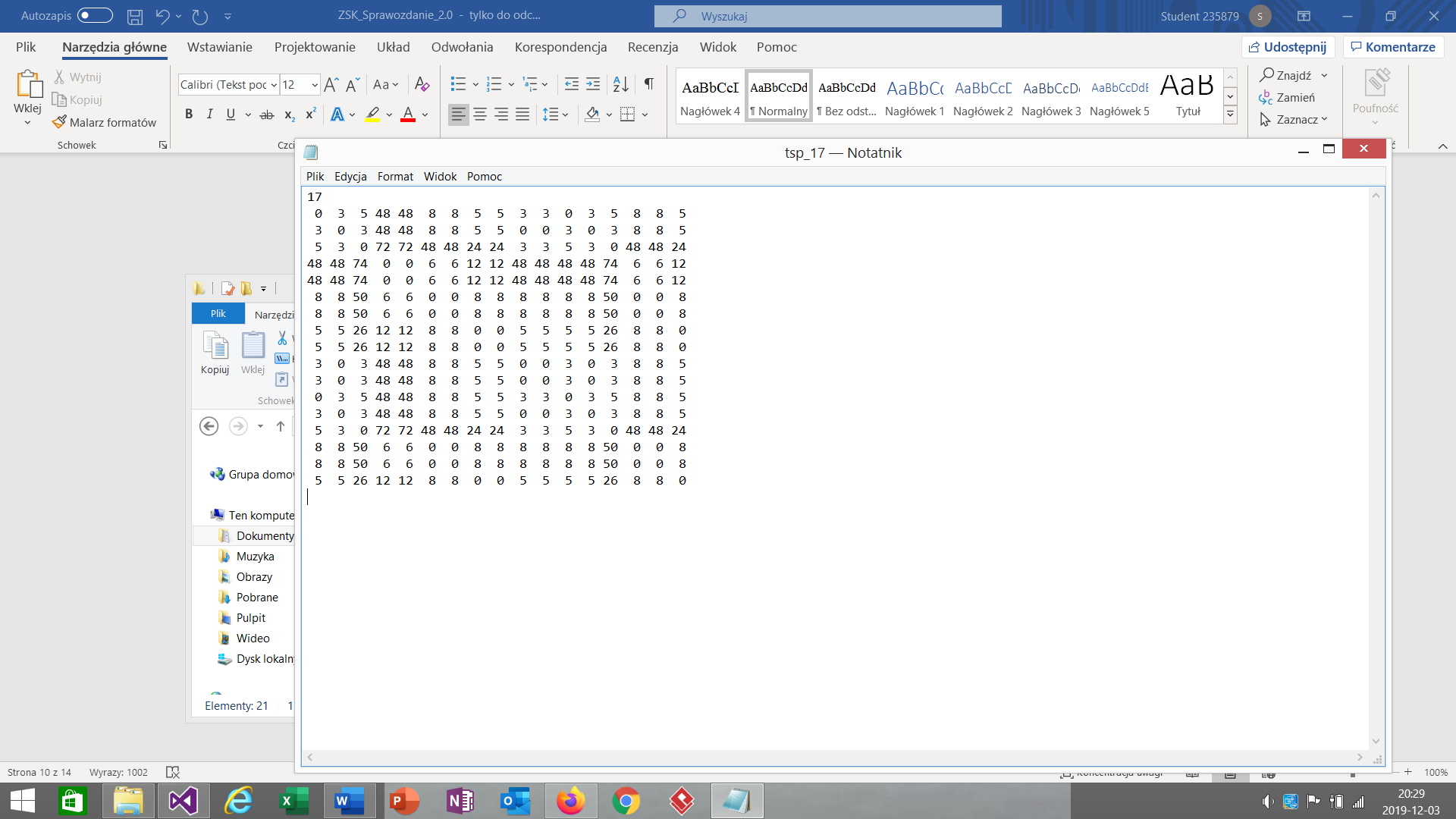
Opis wygenerowany automatycznie

Rysunek 1 Diagram Klas

## Struktura danych wejściowych

Dane wejściowe są pobierane z zewnętrznych plików tekstowych, gdzie pierwszy wiersz zawiera tylko liczbę miast, w kolejnych wierszach przedstawiona jest macierz kosztów, dane na przekątnej mają wartość 0.

Dane wczytywane są do zmiennej *std::vector<vector<int>> matrix* w funkcji *pobierz*.



Rysunek 2 Macierz kosztów z pliku tekstowego

## Struktura wyników

Dane wyjściowe, czyli miasta w odpowiedniej kolejności są wynikiem pracy funkcji *dynamic* umieszczane w zmiennej *std::vector<vector<int>> k* , a później w wyniku działania funkcj *d* wyświetlane w oknie konsoli.

# Implementacja systemu

Klasa Pr implementuje wykorzystanie wątków, które jako parametr przyjmują klasę K wraz z naszą funkcją docelową - dynami. Liczba wątków została ustawiona w sposób stały, przy użyciu predyrektywy define, w celu zmiany liczby wątków należy zmienić ja z poziomu kodu.

Funkcja std::chrono z biblioteka STL odpowiada za wykonywanie pomiarów czasowych dla zadanych parametrów

## Wybrane klasy

**Klasa Dynamic**

Odpowiada za realizację algorytm programowania dynamicznego oraz rozwiązanie wczytanego

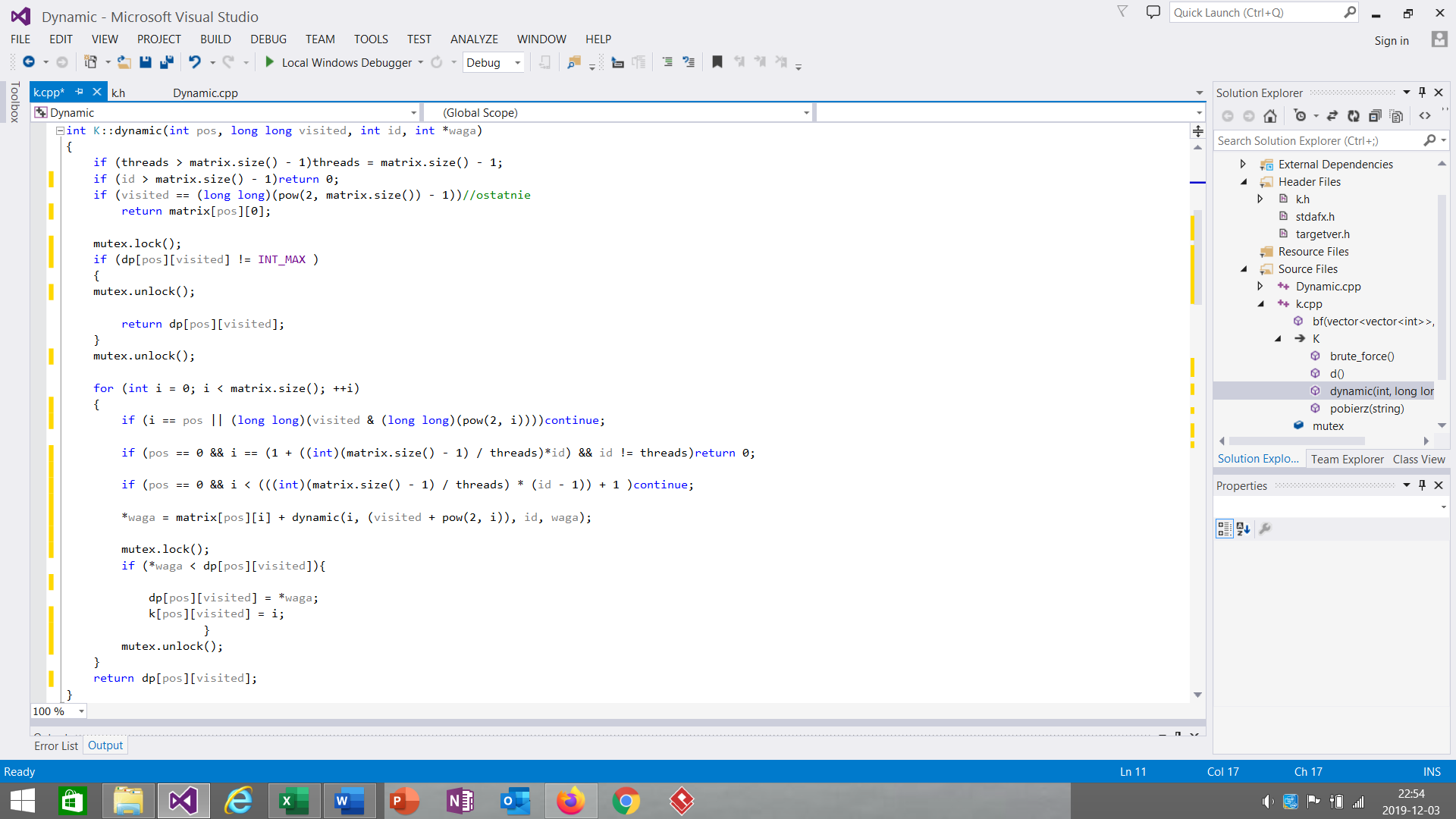
**Klasa Pr**

Jest odpowiedzialna za przechowywanie poszczególnych wag oraz numerów identyfikacyjnych watków.

## Realizacja algorytmu

Implementacja algorytmu programowania dynamicznego znajduje się w funkcji dynamic wywoływane jest dla stałego wybranego wierzchołek o numerze 1.

Warto zaznaczyć problem współdzielenia zasobów występujący w tej funkcji, z tego powodu wykorzystujemy mechanizm mutex, pozwalający na dostęp do zasobu tylko jednemu z wątków, gdy inne stoją w kolejce

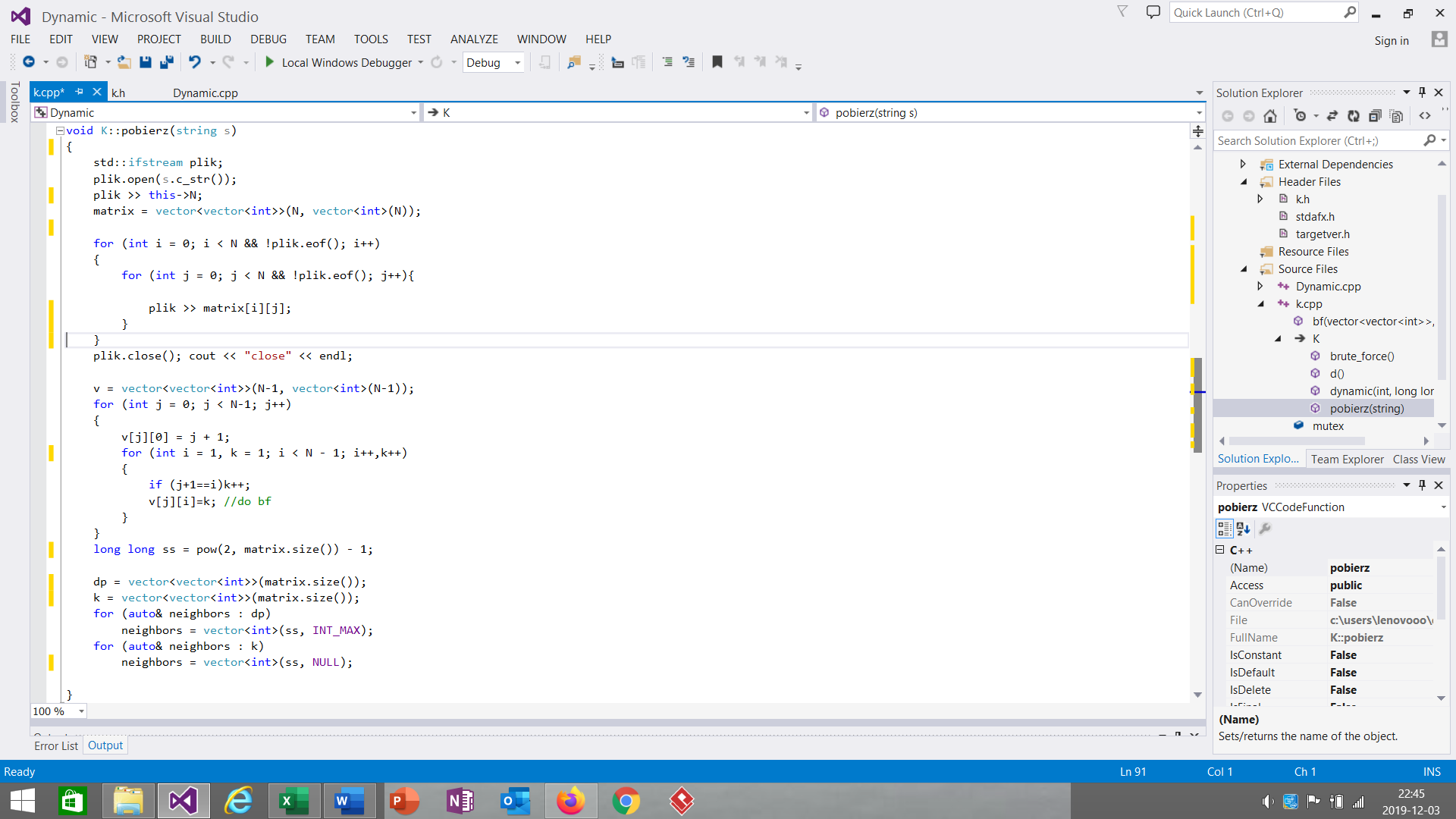


Rysunek 3 Funkcj dynamic implementująca algorytm wykorzystujące programowanie dynamiczne

## Mechanizm odczytu danych

Pliki z informacjami o liczbie miast oraz wagach pomiędzy nimi są wczytywane z pliku tekstowego przy użyciu funkcji *pobierz* będącej metodą klasy *Dynamic*.

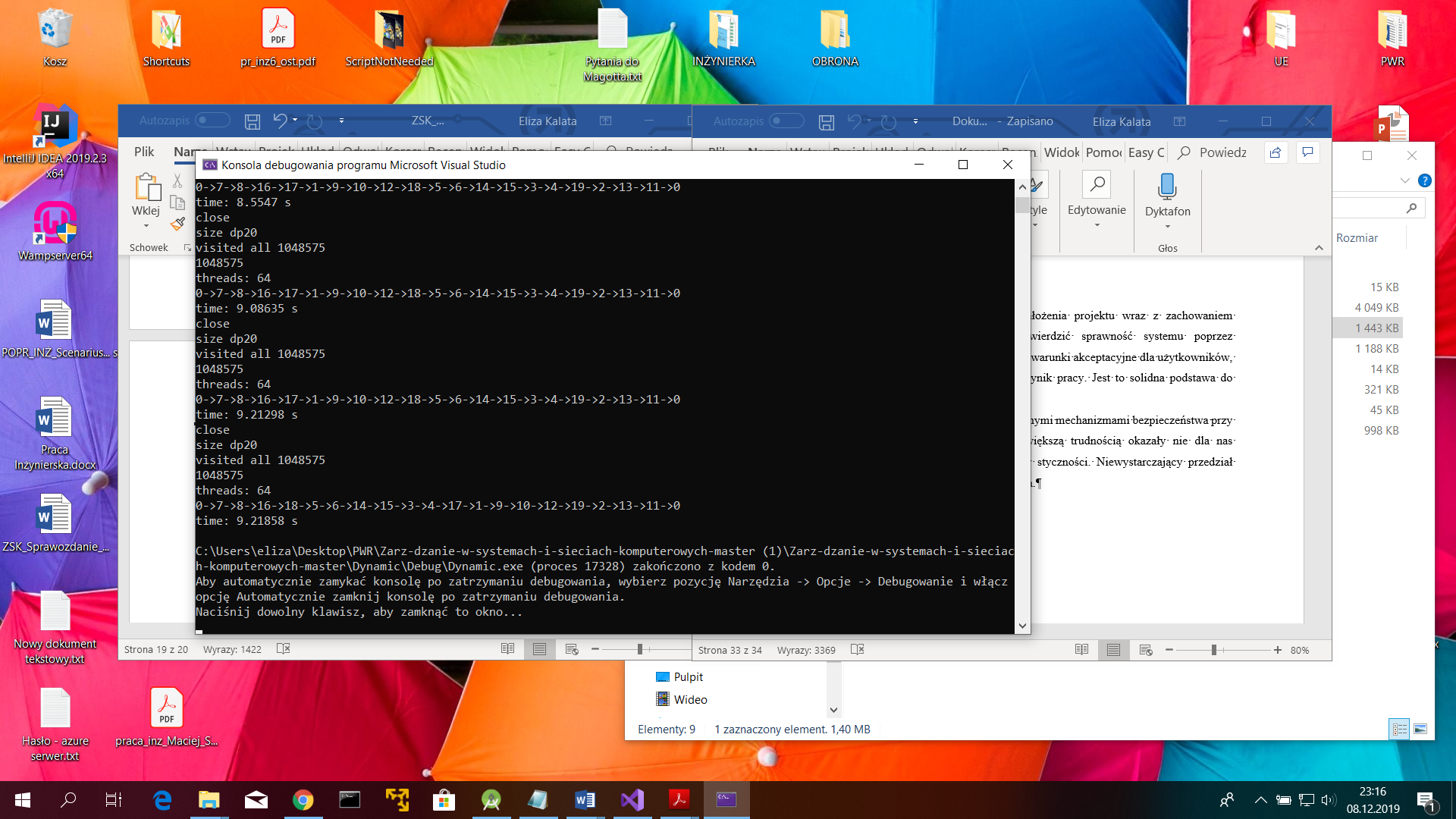
Dane wczytywane są do zmiennej matrix, która jest wektorem wektorów przechowujących zmienne typu *int (std:: vector<vector<int>>).*



Rysunek 4 Metoda pobierz klasy K

## Zapis i prezentacja wyników

Po wykonaniu obliczeń wyniki prezentowane są w konsoli. Size to rozmiar macierz, czyli liczba miast w problemie komiwojażera. Poźniej mamy informację o tym, że udało się odwiedzić wszystkie miasta. Następnie mamy threads – liczbę zastosowanych wątków przy wykonywanych obliczeniach. Wiersz niżej widzimy najkrótszą drogę – numery to poszczególne miasta, a strzałki wytaczają kolejność. Na samym końcu wyświetlany jest czas przebiegu operacji w sekundach z dokładnością do 5 miejsc po przecinku.



Rysunek 5 Widok prezentacji wyników

# Testowanie i ocena wydajności

## Poprawność

W celu zachowania niezależności pomiarów od błędów grubych dokonano uśrednienia. Przeprowadzono pomiary dla 10 instancji, a następnie wyniki zsumowano oraz uśredniono, co zaprezentowano w tabeli 2 i 3.

Tabela 2 Uśrednienia wyników dla 17 miast



Tabela 3 Uśrednienia wyników dla 20 miast



## Analiza czasów

### Porównanie czasów

Rysunek 6.Wykres zależności czasu od liczby wątków dla 17 miast

Rysunek 7.Wykres zależności czasu od liczby wątków dla 20 miast

### Weryfikacja z najlepszym uzyskanym wynikiem

W celu weryfikacji wyników oraz porównania w jakim stopniu użycie wątków pozwoliło na zwiększenie szybkości dokonywanych obliczeń przygotowano wykresy zestawienia. Ukazują one procentowy spadek stosunek czasu dokonywanych obliczeń dla określonej liczby wątków do czasu dla pojedynczego wątki.

Rysunek 8Wykres procentowego spadku czasu wykonywania dla 17 miast

Rysunek 9 Wykres procentowego spadku czasu wykonywania dla 20 miast

## Wnioski z testów

Skonfigurowany system został zweryfikowany pod kątem poprawności i efektywności działania. Porównywanie czasu działania algorytmów zostało zrealizowane przy użyciu biblioteki „chrono” STL [9]. Wraz ze wzrostem liczby wątków czas wykonywania się zmniejszał, aż do pewnego punktu przegięcia, w którym liczba wątków nie miała już znaczenia, a wręcz wątki zaczęły sobie ‘przeszkadza’ powodując delikatny spadek w czasie wykonywania. Na wykresie wiać najpierw tendencje wzrostową a później stały poziom, z niewielkim spadkiem co obrazuje wcześniej opisaną zależność.

# Podsumowanie

Dzięki zastosowanej technologii oraz wiedzy z poprzednich zajęć udało nam się pomyślnie i w pełni zrealizować projekt. Otrzymane wyniki potwierdzają słuszność zrównoleglania obliczeń, pozwalając na oszczędność czasową nawet do 70% z zastosowaniem naszego algorytmy Programowania Dynamicznego.

# Literatura

[1] Cormen T., Leiseron C., Rivest R., Wprowadzenie do algorytmów, WNT, Warszawa, 2001.

[2] https://eduinf.waw.pl/inf/alg/001\_search/0136.php

[3] http://www.mini.pw.edu.pl/MiNIwyklady/grafy/prob-komiw.html

[4] http://wazniak.mimuw.edu.pl/index.php?title=Z%C5%82o%C5%BCono%C5%9B%C4%87\_obliczeniowa/Wyk%C5%82ad\_5:\_Problemy\_NP-zupe%C5%82ne

[5] http://algorytmy.ency.pl/artykul/algorytm\_helda\_karpa

[6] http://cpp0x.pl/kursy/Kurs-STL-C++/Wstep-podstawowe-informacje-o-STL-u/115

[7] https://docs.microsoft.com/pl-pl/visualstudio/productinfo/vs2013-sysrequirements-vs

[8] http://cpp0x.pl/kursy/Kurs-New-Curses-C++/19

[9] https://en.cppreference.com/w/cpp/chrono