# STRUKTURY DANYCH I ZŁOŻONOŚĆ OBLICZENIOWA

## Zadanie projektowe nr 2

| **Temat:** |
| --- |
| Badanie efektywności algorytmów grafowych. |
| **Autor:** |
| Dawid Waligórski |
| **Termin zajęć:** |
| Wtorek, godz. 15.15, TP |

# Spis treści

# 1 Założenia projektowe:

## 1.1 Cele projektu:

Celem projektu była implementacja algorytmów grafowych wraz z niezbędnymi do ich działania strukturami danych w języku C++. Kluczowym było również dokonanie pomiaru czasów wykonywania się wspomnianych algorytmów na grafach o różnych reprezentacjach, rozmiarach (liczba wierzchołków) oraz gęstościach (liczba krawędzi). Badanymi algorytmami były:

- Algorytm Prima (problem wyznaczenia MST)
- Algorytm Kruskala (problem wyznaczenia MST)
- Algorytm Dijkstry (problem wyszukiwania najkrótszych ścieżek)
- Algorytm Bellmana-Forda (problem wyszukiwania najkrótszych ścieżek)

Tworzona aplikacja z konsolowym interfejsem użytkownika, poza możliwością przeprowadzenia eksperymentów miała oferować również opcję sprawdzenia poprawności zaimplementowanych algorytmów za pomocą dedykowanych menu.

Szczegółowe założenia tyczące się poszczególnych algorytmów zostały zawarte w odpowiadających im rozdziałach.

## 1.2 Grafy:

Założono, że wierzchołki grafu będą identyfikowane przez kolejne, nieujemne liczby całkowite (typ *unsigned int*). Do przechowywania wartości krawędzi wybrano znakowany, 4-bajtowy typ całkowity (*int*). W zależności od wybranego problemu mogły być one skierowane (najkrótsza ścieżka) lub nieskierowane (MST).

Badane algorytmy otrzymać miały swoje odpowiedniki dla dwóch różnych sposobów reprezentacji grafu. Pierwszy z nich zakładał użycie macierzy sąsiedztwa. Drugi korzystać miał ze zbioru połączonych list sąsiadów. W wypadku macierzy sąsiedztwa, zamiast tradycyjnej, binarnej informacji o istnieniu krawędzi między wierzchołkami, miała ona zawierać informację o wadze krawędzi. Brak połączenia w tej strukturze był oznaczany za pomocą specjalnej, zarezerwowanej wartości NO_CONNECTION (wartość minimalna typu

*int*).

Dodatkowo możliwym miało być wczytanie grafu z pliku (np. tekstowego). Pierwsza linia w takim pliku definiować miała liczbę wierzchołków, krawędzi, a także wierzchołki początkowy i końcowy (wykorzystywane przy niektórych algorytmach). W kolejnych liniach znajdować się miały dane na temat kolejnych krawędzi grafu w konwencji: wierzchołek początkowy, wierzchołek końcowy, waga.

# 2 Metoda pomiarowa:

## 2.1 Środowisko pomiarowe:

Pomiary przeprowadzano na komputerze stacjonarnym z zainstalowanym systemem Windows 10 (wersja 64-bitowa). Częstotliwość taktowania zegara procesora wynosiła 3,60GHz. Ilość pamięci operacyjnej wynosiła 16GB. W trakcie pomiarów na maszynie działały jedynie niezbędne do funkcjonowania systemu operacyjnego procesy. Aplikacja mierząca działała w domyślnym trybie przydziału zasobów komputera. Była ona skompilowana za pomocą narzędzia *g++* na poziomie optymalizacji O3.

## 2.2 Pomiary wykonywane w programie:

### 2.2.1 Narzędzie pomiarowe:

W pomiarach czasu wykorzystano specjalną klasę *Timer*. Instancja wspomnianej klasy działała na zasadzie stopera, mierząc czas pomiędzy wywołaniami swoich metod *start()* oraz *stop()*. Precyzja prowadzonego nią pomiaru była rzędu 0,1 mikrosekundy ($10^{-7}s$). Przykład użycia *Timer* został zamieszczony na rysunku **??**.

Rysunek 1: Przykład użycia klasy *Timer* do pomiaru czasu wykonania algorytmu Bellmana-Forda.

```
// Pomiar
timer.start();
graph->algorithmBellmanFord(start);
timer.stop();
```

### 2.2.2 Przebieg pomiaru:

Pomiary czasu wykonywania algorytmów zostały wkomponowane jako jedne z opcji możliwych do wyboru z poziomu aplikacji projektowej (rys. 2). W wypadku wybrania takiej opcji aplikacja dawała użytkownikowi opcję wyboru algorytmu, który miał zostać przebadany. Następnie wykonywany był właściwy pomiar, dzielący się na 40 faz. Każda faza wyróżniała się inną kombinacją parametrów używanego w pomiarach grafu. Owe parametry oraz ich dopuszczalne wartości zamieszczono w tabeli 1. W trakcie jednej fazy wykonywano łącznie 100 prób. Dla każdej z osobna generowano nową, losową instancję grafu (odpowiedniego dla danego algorytmu). Później za pomocą klasy *Timer* dokonywano pomiaru czasu wykonywania wybranego algorytmu. W końcu dane o próbie były zapisywane w specjalnej strukturze, która po zakończeniu się danej fazy była zapisywana do pliku **.csv* w formie jednego z jego z rekordów. Pola takiego rekordu została zamieszczona w tabeli 2.2.2.

Rysunek 2: Menu główne, zawierające opcję przeprowadzenia pomiarów.

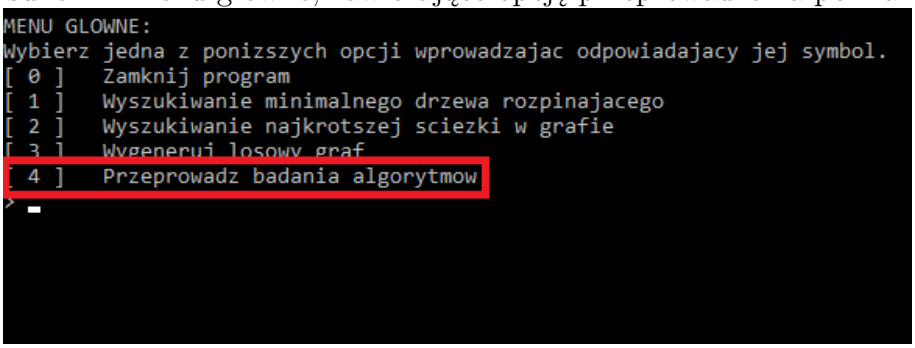


Tabela 1: Tabela zawierające parametry grafów używanych w poszczególnych fazach badania wraz z ich dopuszczalnymi wartościami.

| **Parametr:** | Reprezentacja | Rozmiar | Gęstość |
|---|---|---|---|
| **Opis:** | Sposób reprezentacji grafu. | Liczba wierzchołków. | Stosunek liczby krawędzi grafu do jego maksymalnej liczby krawędzi. |
| **Wartości:** | listowa<br>macierzowa | 100<br>200<br>300<br>400<br>500 | 25%<br>50%<br>75%<br>99% |

Tabela 2: Tabela zawierająca strukturę rekordu pliku **.csv*, zawierającego wyniki pomiarów.

| | |
|---|---|
| **Algorytm** | Inicjał odpowiadający badanemu algorytmowi. |
| **Reprezentacja** | Typ reprezentacji przebadanego grafu. |
| **Rozmiar** | Rozmiar przebadanego grafu. |
| **Gęstość** | Gęstość przebadanego grafu. |
| **Czas** | Zmierzony czas. |

### 2.2.3 Generowanie grafu:

Za generowanie losowych instancji grafów, używanych w badaniach odpowiedzialna była klasa *GraphGenerator*. Była ona paremetryzowana wartościami takimi jak:

- Typ reprezentacji grafu
- Liczba wierzchołków
- Informacja o skierowaniu krawędzi
- Gęstość grafu
- Informacja o dopuszczalności wystąpienia ujemnych wag krawędzi

Tak duża gama parametrów pozwoliła na generowanie grafów dobrze przystosowanych do każdego z zaimplementowanych algorytmów (np. zabraniając generowania się ujemnych krawędzi w wypadku algorytmu Dijkstry).

Najważniejszym i najbardziej wymagającym zadaniem powierzonym omawianej klasie było generowanie losowego zbioru krawędzi dla tworzonego grafu. Było ono wykonywane w dwóch etapach. W pierwszym generowane, a następnie umieszczane w tablicy były wszystkie, możliwe dla danego typu grafu (skierowany/nieskierowany) krawędzie. Drugi etap polegał na wybraniu z uzyskanej kolekcji krawędzi losowego podzbioru o wielkości podyktowanej przez parametr gęstości grafu. Odbywało się to poprzez kolejne losowania indeksu z wygenerowanej w etapie pierwszym tablicy, stale pomniejszanej o już wybrane krawędzie.