Projektowanie Algorytmów i Metody Sztucznej Inteligencji	
Prowadzący *Mgr inż. Marta Emirsajłow*	**Termin** *Poniedziałek 15:15*
Imię, nazwisko, numer albumu,grupa *Mateusz Szlachetko 259370 Y03-51a*	**Data** *25 kwietnia 2022*
Temat ćwiczenia i nr *Projekt 2*	

1 Wstęp

1.1 Cel ćwiczenia

Celem ćwiczenia było zbadanie efektywności jednego z algorytmów rozwiązującego problem, znalezienia najkrótszej ścieżki pomiędzy wybranymi wierzchołkami w grafie, oraz implementacja grafów za pomocą listy sąsiedztwa i macierzy sąsiedztwa.

Tabela 1: Wybrany algorytm

Dijkstra

1.2 Opis problemu i algorytmu

Problem najkrótszej drogi (ścieżki) w grafie między dwoma wierzchołkami polega na znalezieniu w grafie ważonym najkrótszego połączenia pomiędzy tymi wierzchołkami.

Do rozwiązania tego problemu możemy użyć np. algorytmu Dijkstry.
Zasada działania:

Z danego grafu którego chcemy poddać działaniu algorytmu wybieramy jeden interesujący nas wierzchołek(source). Następnie tworzymy tablicę odległości pomiędzy źródłem i pozostałymi wierzchołkami. Inicjujemy ją na wartości nieskończone(dla źródła 0). Tworzymy kolejkę priorytetową, której priorytetem jest aktualnie wyliczona odległość od wierzchołka źródłowego. Dopóki nie opróżnimy całej kolejki, usuwamy wierzchołek o najniższym priorytecie i dokonujemy relaksacji poprzez u dla sąsiednich wierzchołków.

Dla opisania dokładnej przebytej drogi w tablicy parent możemy przechowywać dla każdego wierzchołka numer jego bezpośredniego poprzednika na najkrótszej ścieżce, co pozwoli na odtworzenie pełnej ścieżki od źródła do każdego wierzchołka.

Dla wyszukania drogi tylko do konkretnego wierzchołka, wystarczy zrobić przerwanie w pętli, gdy już znajdziemy do niego ścieżkę.

Złożoność obliczeniowa algorytmu Dijkstry zależy od liczby wierzchołków "V" i krawędzi "E" w grafie (i powiązanej z dwoma powyższymi wartościami-gęstości). Na rząd złożoności wpływa implementacja kolejki priorytetowej.

- Implementacja kolejki poprzez zwykłą tablicę $O(V^2)$
- Implementacja kolejki poprzez kopiec binarny $O(ElogV)$
- Implementacja kolejki poprzez kopiec Fibonacciego $O(E + VlogV)$

Pierwszy wariant kolejki jest optymalny dla grafów gęstych, drugi jest szybszy dla grafów rzadkich, trzeci, szybszy niż drugi, ale skomplikowany, a zysk czasowy jest niewielki.

2 Eksperyment

Przebieg eksperymentu polegał na wywołaniu alogrytmu dla różnych ilości wierzchołków i gęstości w grafie, wyniki są uśrednione dla 100 losowo wygenerowanych grafów o danych parametrach. Dodatkowo dla zobrazowania wyżej opisanych właściwości algorytmu w zależności od sposobu implementowania kolejki, dla macierzy sąsiedztwa algorytm opiera się o kolejkę tablicową, a dla listy sąsiedztwa o kolejkę przy użyciu kopca. Same różnice w implementacji grafu, zmieniały tylko właściwości takie jak szybkość przeszukiwania i samo podejście do rozpisania algorytmu.

2.1 Tabele

Tabela 2: Wyniki dla gęstości=0,25

Rodzaj grafu	**Gęstość**	**Czas[mikrosekundy]**
Macierz	**25%**	
V		
10		1.92
50		27.5
100		98.94
500		2264.99
1000		8546.7
Lista	**25%**	
V		
10		0.92
50		13.05
100		41.53
500		1177.65
1000		10669.8

Tabela 3: Wyniki dla gęstości=0,5

Rodzaj grafu	**Gęstość**	**Czas[mikrosekundy]**
Macierz	**50%**	
V		
10		1.1
50		28.36
100		102.1
500		2287.83
1000		8819.43
Lista	**50%**	
V		
10		1.05
50		18.99
100		68.89
500		2377.83
1000		38916.9

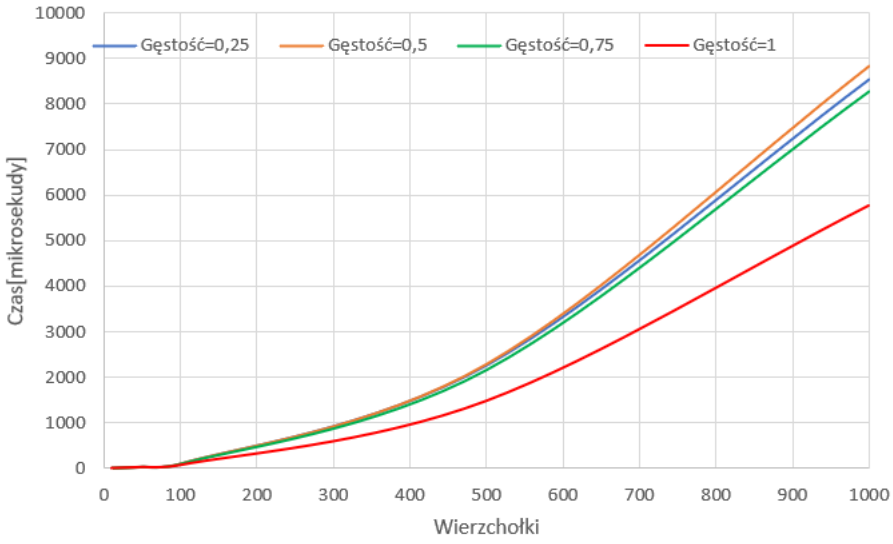
Tabela 4: Wyniki dla gęstości=0,75

Rodzaj grafu	**Gęstość**	**Czas[mikrosekundy]**
Macierz	**75%**	
V		
10		1
50		26.5
100		90.81
500		2162.54
1000		8287.27
Lista	**75%**	
V		
10		1.1
50		25.68
100		99.76
500		4672.54
1000		63928.1

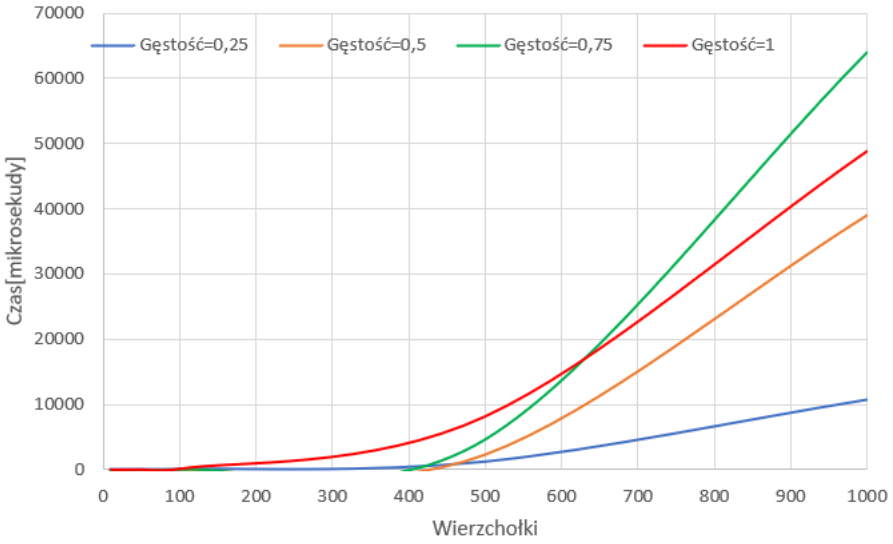
Tabela 5: Wyniki dla gęstości=1

Rodzaj grafu	**Gęstość**	**Czas[mikrosekundy]**
Macierz	**100%**	
V		
10		1.15
50		23.77
100		67.8
500		1478.87
1000		5780.23
Lista	**100%**	
V		
10		1.17
50		29.87
100		134.43
500		8137.1
1000		48723.5

2.2 Wykresy



Rysunek 1: Wykresy dla grafu macierzy sąsiedztwa



Rysunek 2: Wykresy dla grafu listy sąsiedztw

3 Podsumowanie

Program został napisany obiektowo, przy użyciu klas i struktur. Poza powyższymi eksperymentami, została zaimplementowana możliwość wczytywania grafu z pliku, zarówno dla reprezentacji za pomocą listy jak i macierzy, oraz zapis wyniku działania algorytmu Dijkstry do pliku(dodatkowa możliwość, wyświetlenie w terminalu).

Wnioski:

- Po wykonaniu testów poprawności, wyświetlaniu wyników działania algorytmu, program działa prawidłowo
- Różnice pomiędzy reprezentacją grafu za pomocą macierzy lub listy sąsiedztwa, przejawiają się w szybkości operacji wykonywanych na tych obiektach, ale i na zajmowanej pamięci.
- Dla badanych implementacji algorytmu Dijkstry, dla grafów o mniejszej gęstości lepiej działała kolejka zaimplementowana za pomocą kopca, dla grafów i większej gęstości ta ze zwykłą tablicą, co zgadza się z przewidywaniami.
- Dla zwiększającej się liczby wierzchołków czas wykonywania algorytmu wzrastał zgodnie z oczekiwaniami, dla kolejki kopcowej, przy większych ilościach wierzchołków, gęstości i co za tym idzie krawędzi, algorytm zaczynał działać już wyraźnie wolniej, jednak wszystko to odbywało się w czasie mniejszym niż sekunda, co nadal jest bardzo efektywne
- Na wyniki mógł mieć też wpływ przedział losowanych wag dla poszczególnych wierzchołków od 3 do 10.
- Same eksperymenty zajmowały dość dużo czasu, przez potrzebę tworzenia 100 losowych grafów, gdzie dla większych gęstości(bardzo dużej ilości krawędzi) i ilości wierzchołków, trzeba było wielokrotnie losować jeszcze nie połączone dotychczas krawędzie
- Dla usprawnienia, dla grafów pełnych metoda przypisywania wag, odbywała się po kolei bez losowania
- Wszystkie wyniki są charakterystyczne dla danego komputera na którym były wykonywane pomiary.