PROJEKT2

Sprawozdanie

Projektowanie algorytmów i metod sztucznej inteligencji
Mgr inż. Marta Emirsajłow
Poniedziałek 13.15-15.00

Nikodem Iwin (252928)
10.05.2021

1. Wstęp

Jako drugi projekt z przedmiotu Projektowanie algorytmów i metod sztucznej inteligencji należało zaimplementować reprezentacje grafów ważonych w postaci macierzowej i listowej, a następnie zaimplementować algorytm Dijkstry, który służy rozwiązywaniu problemu najkrótszej ścieżki oraz sprawdzić jego efektywność w oparciu o uśrednione testy 100 losowych grafów.

Grafy miały następujące liczby wierzchołków (*V*): 10, 50, 100, 500, 1000.

Gęstości grafów były następujące (*D*): 25%, 50%, 75%, 100%.

2. Opis algorytmów

1) Graf w postaci macierzowej

Macierz sąsiedztwa można sprowadzić do tablicy dwuwymiarowej, której indeksy kolumn i wierszy świadczą o połączeniu wierzchołków, a wartość danej komórki przedstawia wagę krawędzi.

Złożoność obliczeniowa dla mojej reprezentacji grafu poprzez postać macierzową wynosi $O(V^2)$, gdzie V opisuje liczbę wierzchołków. Złożoność mojego algorytmu taka sama jak teoretyczna.

2) Graf w postaci listowej

Lista sąsiedztwa możemy porównać do tablicy jednowymiarowej, która dla każdej komórki zawiera paczkę danych, w której znajdują się informacje o wierzchołach i wagach krawędzi.

Grafy reprezentowany poprzez listę sąsiedztwa w odróżnieniu od formy macierzowej często zajmują mniej miejsca i zużywają mniej zasobów. Jest to szczególnie widoczne dla grafów rzadkich lub grafów o niewielkiej ilości krawędzi.

Ich złożoność obliczeniowa wynosi $O(E+V)$, gdzie E oznacza ilość krawędzi, a V oznacza liczbę wierzchołków.

Złożoność obliczeniowa dla mojej reprezentacji grafu poprzez postać listową wynosi $O(V^2)$, co wynika z faktu iż nie poruszam się po krawędziach, tylko sprawdzam czy istnieją.

3) Algorytm Dijkstry

Algorytm Dijkstry pozwala na rozwiązaniu najkrótszej ścieżki, którą należy przebyć z punktu *A* do punktu *B.* Algorytm składa się z paru konkretnych kroków:

I. Stworzenie tablicy relacji miedzy punktem startowym, a resztą wierzchołków.
II. Utworzenie kolejki priorytetowej ze wszystkimi wierzchołkami, której priorytetem jest waga odległość między elementem początkowym. a innym.
III. Wykonywania dwóch kroków IV i V dopóki kolejka nie jest pusta
IV. Usuń z kolejki element o najmniejszym priorytecie.
V. Dla każdego sąsiada elementu usuniętego zmień wagę na mniszą.
VI. Gdy skończy się kolejka powinniśmy otrzymać tabelę utworzoną w punkcie I, którą zawiera najkrótsze odległości.

Złożoność obliczeniowa dla implementacji wynosi teoretycznie *O(E+logV)*, gdzie *E* oznacza ilość krawędzi, a *V* oznacza liczbę wierzchołków. W moim programie złożoność wynosi $O(V^2)$. Wynika to z tego samego faktu iż algorytm sprawdza czy istnieje krawędź, a nie porusza się po nich.

3. Omówienie przebiegu eksperymentów oraz wyniki

1) Warunki przebiegu eksperymentów

Całość programu została napisana w Visual Studio Code. Eksperymenty zostały wykonane na moim prywatnym komputerze o procesorze Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz 3.90 GHz i 10,0 GB pamięci RAM.

Czas liczony był dzięki bibliotece zewnętrznej „chrono". Pomiar odbywał się poprzez odmierzanie każdorazowego czasu trwania samego algorytmu Djikstry. Następnie pomiary oraz ich uśredniony czas były zapisywane w utworzonym pliku tekstowym. Czas mierzony był w mikrosekundach.

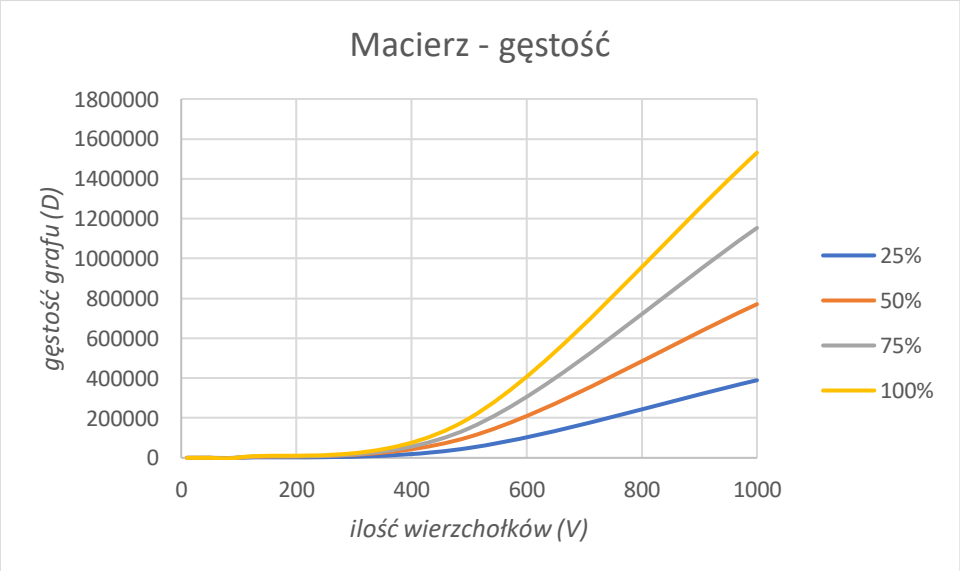
Każdy eksperyment odbył się na tym samym kodzie przy podobnym użyciu procesora i pamięci. Wszystkie warunki pracy programów były określane bezpośrednio z jego menu.

Poniżej w tabelach zaprezentowane są uśrednione wyniki otrzymanych czasów (w mikrosekundach). Pierwsza tabela przedstawia pomiary dla grafów reprezentowanych poprzez macierz, a druga pomiary dla grafów reprezentowanych poprzez listę.

2) Tabele i wykresy dla grafów w postaci macierzy sąsiedztwa

Poniżej zaprezentowano tabelę zawierającą uśrednione pomiary czasów (w mikrosekundach) zwrócone poprzez program.

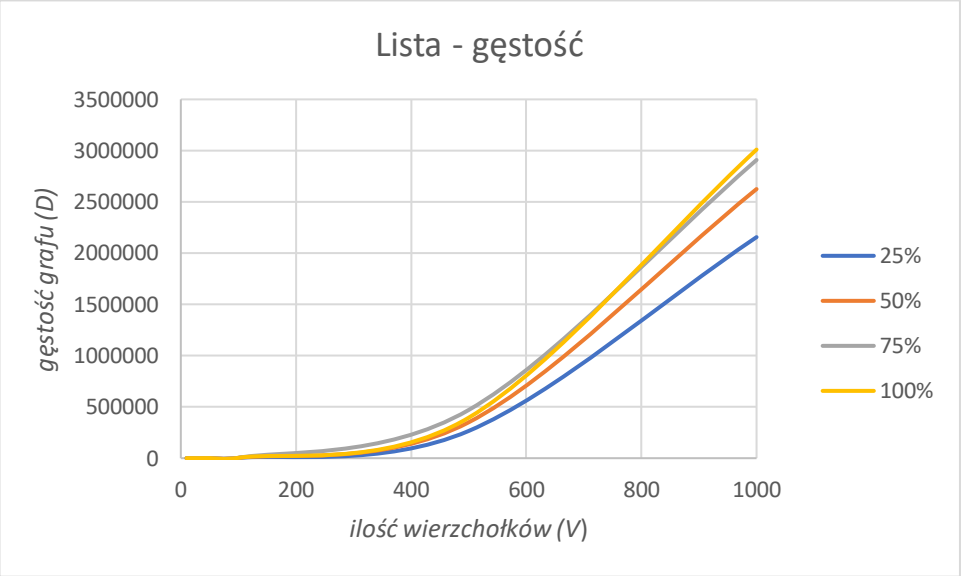
	10	**50**	**100**	**500**	**1000**
25%	0	119	629	49558	389452
50%	0	190	1144	104827	770856
75%	44	280	1414	148385	1153202
100%	11	209	1817	197856	1530557



3) Tabele i wykresy dla grafów w postaci listy sąsiedztwa

Poniżej zaprezentowano tabelę zawierającą uśrednione pomiary czasów (w mikrosekundach) zwrócone poprzez program.

	10	**50**	**100**	**500**	**1000**
25%	9	279	2599	265877	2156135
50%	29	509	3209	350075	2625204
75%	19	559	3382	470232	2908751
100%	39	599	3554	395230	3010101



4. Podsumowanie i wnioski

Pomiar czasy czasów otrzymane są w dużej mierze prawidłowe, lecz istnieje duża możliwość wystąpienia błędu pomiarowego uwarunkowanego wadliwością mojego procesora.
W plikach wynikowych pojawiały się czasy przedstawione jako 0, ale ten sam algorytm na mocniejszym układzie zwracał wartości wyrażane poprzez liczby, jednakże średnie wyniki były zbliżone.

Pomimo możliwych błędnych pomiarów możemy wyciągnąć następujące wnioski:

- Zgodnie z powyższymi danymi szybszy jest algorytm działający na reprezentacji macierzowej.

- Teoretycznie algorytm obsługujący postać listową powinien być szybszy dla małych grafów oraz mało zagęszczonych. Powyższe tabele pokazują, że w przypadku moich implementacji tak nie jest. Wynika to z faktu iż algorytm nie porusza się po krawędziach, lecz sprawdza za każdym razem czy dana krawędź istnieje. Powoduje to zmianę złożoności obliczeniowej na $O(V^2)$. Należy również zwrócić uwagę ze w implementacji listowej korzystano z kontenera *vector*, z którego dane są pobierane w dłuższym czasie niż w przypadku algorytmu obsługującego macierz, w której dostępy do kolejki są szybsze.

5. Literatura

- Instrukcja do Projektu2 zamieszczona na Microsoft Teams w zespole przedmiotowym
- https://eduinf.waw.pl/inf/alg/001_search/0124.php
- https://pl.wikipedia.org/wiki/Graf_sp%C3%B3jny
- https://pl.wikipedia.org/wiki/Reprezentacja_grafu
- https://pl.wikipedia.org/wiki/Graf_(matematyka)
- https://pl.wikipedia.org/wiki/Algorytm_Dijkstry
- https://www.youtube.com/watch?v=P7QLI3rLtwE – wstępne pojęcie czym są grafach
- https://www.youtube.com/watch?v=PVW8IHZruAw&t=43s – wizualizacja algorytmu Dijkstry