Sprawozdanie 2

"Projektowanie algorytmów i metod sztucznej inteligencji"

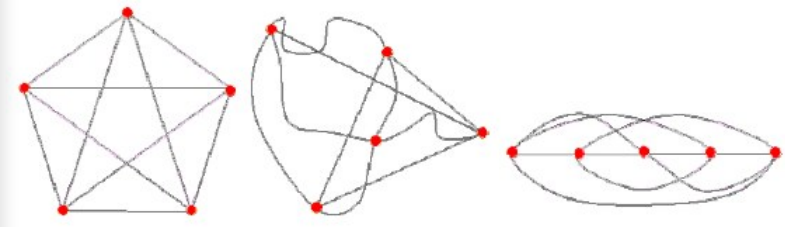
7 maja 2019

Temat projektu : Grafy
Autor : Przemysław Widz
Termin zajęć : ŚRODA 7:30 - 9:00
Prowadzący : Dr inż. Łukasz Jeleń

1 Wstęp teoretyczny

1.1 Definicja

Grafem nazywamy taką strukturę danych, która składa się z wierzchołków i krawędzi, przy czym poszczególne wierzchołki (zwane również węzłami) mogą być połączone krawędziami (skierowanymi lub nieskierowanymi) w taki sosób, iż każda krawędź zaczyna się i kończy w którymś z wierzchołków. Wierzchołki i krawędzie mogą być numerowane, etykietowane i nieść pewną dodatkową informację - w zależności od potrzeby modelu, do którego konstrukcji są wykorzystane. W porównaniu do drzew w grafach mogą występować pętle i cykle. Krawędzie mogą mieć wyznaczony kierunek (wtedy graf nazywamy skierowanym), mogą mieć przypisaną wagę (pewną liczbę), kolor, etykietę, np. odległość pomiędzy punktami w terenie, rodzaj połączenia.



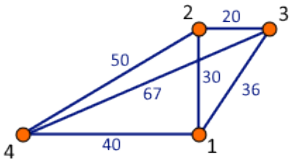
Rysunek 1: Przykłady graficznej reprezentacji różnych grafów

W ogólności dopuszcza się krawędzie wielokrotne i pętle (czyli krawędzie, które rozpoczynają i kończą się w tym samym wierzchołku).

1.2 Podział grafów

Istnieje wiele rodzajów grafów, które mogą mieć wiele interesujących właściwości. Grafy mogą być, np.:

- **etykietowane** - gdy wierzchołki przybierają pewne etykiety
- **ważone** - gdy krawędzie posiadają pewne wartości (wagi)
- **skierowane** - gdy możliwe jest przejście pomiędzy wierzchołkami tylko w jedną stronę (krawędź wtedy oznaczamy strzałką)
- **nieskierowane** – gdy możliwe jest przejście pomiędzy wierzchołkami w obydwie strony
- **spójne** - gdy istnieje ścieżka (droga) w grafie łącząca dowolne dwa wierzchołki
- **niespójne** - gdy takiej ścieżki (drogi) nie można wyznaczyć dla dowolnych dwóch wierzchołków
- **pełne** - gdy każde dwa wierzchołki w grafie są połączone ze sobą krawędzią
- **regularne** - czyli posiadające wszystkie wierzchołki tego samego stopnia, czyli z każdego wierzchołka grafu regularnego wychodzi dokładanie taka sama ilość krawędzi
- **i wiele innych**



Rysunek 2: Przykład graficznej reprezentacji grafu ważonego

1.3 Graf i jego reprezentacja

Graf w informatyce jest najbardziej skomplikowaną podstawową strukturą danych. Stosy, kolejki, listy i drzewa można traktować jako szczególne przypadki grafu lub grafy uproszczone. Grafy dostarczają nam potężne możliwości tworzenia różnego rodzaju relacji pomiędzy danymi, które można wiązać na wiele różnych sposobów, reprezentując lub modelując złożone zależności z otaczającego nas świata.

Jednymi z najpopularniejszych metod reprezentacji grafu w komputerze są :

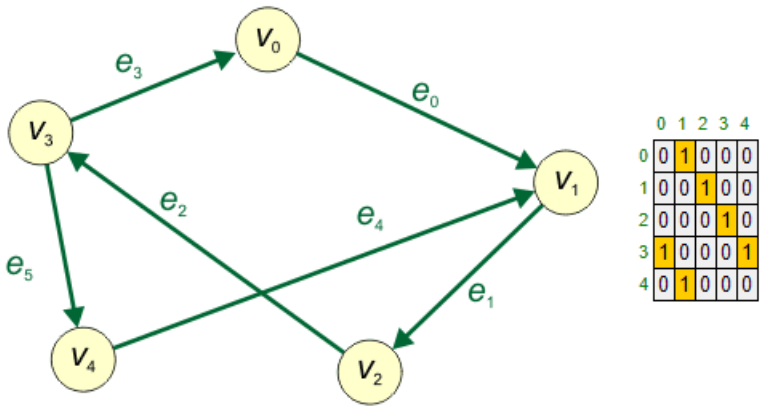
- **macierz sąsiedztwa**

  Macierz sąsiedztwa ma wymiary VxV. Idea jest prosta: jeżeli na "skrzyżowaniu" i-tego wiersza i j-tej kolumny znajduje się 1 to znaczy, że istnieje krawędź (i,j) (wychodząca z i-tego wierzchołka i wchodząca do j-tego); jeśli 0 to znaczy, że tej krawędzi nie ma.

1. złożoność pamięciowa: O(V^2)
2. przejrzenie wszystkich krawędzi: O(V^2)
3. przejrzenie następników/poprzedników danego wierzchołka: O(V)
4. sprawdzenie istnienia jednej krawędzi: O(1)

Wadą tej reprezentacji jest duża złożoność pamięciowa i czasy przejrzenia większej liczby krawędzi. Dla grafów gęstych jest to niezauważalne, ale dla grafów rzadkich tak.

Zaletą jest możliwość szybkiego sprawdzenia czy dana krawędź istnieje oraz prosta implementacja.



Rysunek 3: Graf w postaci macierzy sąsiedztwa

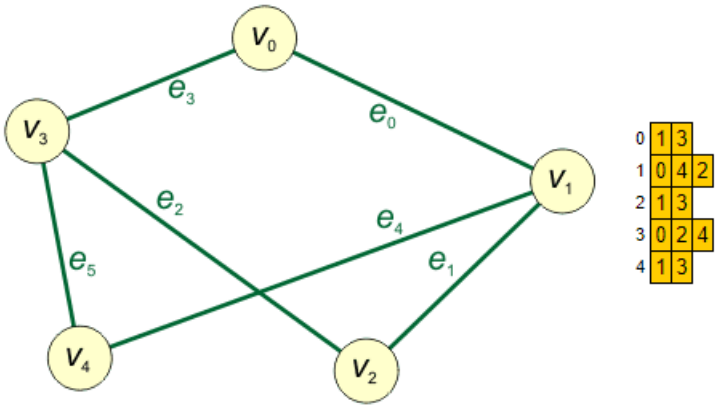
- **lista sąsiedztwa**

  Reprezentacja grafów przez listy sąsiedztwa, jak sama nazwa wskazuje, polega na trzymaniu dla każdego wierzchołka listy jego wszystkich sąsiadów (następników albo poprzedników).

1. złożoność pamięciowa: O(E)
2. przejrzenie wszystkich krawędzi: O(E)
3. przejrzenie następników/poprzedników danego wierzchołka: maksymalnie O(V) (bo tyle sąsiadów może mieć wierzchołek), ale średnio O(E/V) (trzeba po prostu przejrzeć całą listę sąsiadów, których średnio wierzchołek ma właśnie E/V)
4. sprawdzenie istnienia jednej krawędzi: j.w. (bo i tak trzeba przejrzeć listę dla pewnego wierzchołka)

Zaletą jest minimalna możliwa złożoność pamięciowa, szybkie przeszukiwanie krawędzi wychodzących z danego wierzchołka, stosunkowo łatwa implementacja.

Wadą jest jedynie długi czas sprawdzenia istnienia pojedyńczej krawędzi. Można go poprawić do O(log V) wprowadzając drzewa BST zamiast list, ale to skomplikuje implementację.

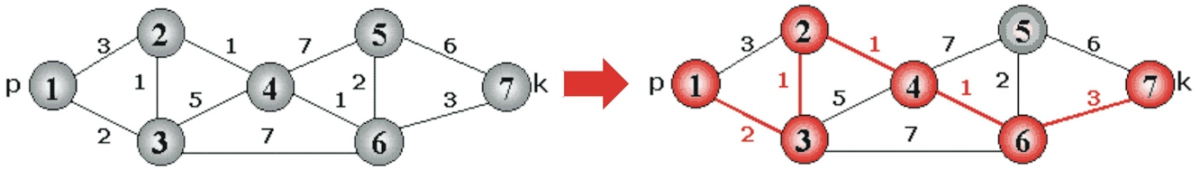


Rysunek 4: Graf w postaci listy sąsiedztwa

1.4 Problemy i algorytmy grafowe

- **Algorytm Dijkstry** znajduje najkrótszą drogę z wierzchołka p (zwanego źródłem lub początkiem) do wszystkich pozostałych wierzchołków grafu lub do wybranego wierzchołka k (zwanego ujściem lub końcem) w grafie, w którym wszystkim krawędziom nadano nieujemne wagi. Polega na przypisaniu wierzchołkom pewnych wartości liczbowych. Taką liczbę nazwijmy cechą wierzchołka. Cechę wierzchołka v nazwiemy stałą (gdy jest równa długości najkrótszej drogi z p do v) albo, w przeciwnym przypadku, tymczasową. Na początku wszystkie wierzchołki, oprócz p, otrzymują tymczasowe cechy . źródło p otrzymuje cechę stałą równą 0. Następnie wszystkie wierzchołki połączone krawędzią z wierzchołkiem p otrzymują cechy tymczasowe równe odległości od p. Potem wybierany jest spośród nich wierzchołek o najmniejszej cesze tymczasowej. Oznaczmy go v. Cechę tego wierzchołka zamieniamy na stałą oraz przeglądamy wszystkie wierzchołki połączone z v. Jeśli droga z p do któregoś z nich, przechodząca przez v ma mniejszą długość od tymczasowej cechy tego wierzchołka, to zmniejszamy tą cechę. Ponownie znajdujemy wierzchołek o najmniejszej

cesze tymczasowej i zamieniamy cechę tego wierzchołka na stałą. Kontynuujemy to postępowanie aż do momentu zamiany cechy wierzchołka k na stałą (czyli obliczenia długości najkrótszej drogi z p do k).



Rysunek 5: Działanie algorytmu Dijkstry

- **Algorytm Bellmana-Forda** Jeśli ścieżki grafu posiadają nieujemne wagi, to najlepszym rozwiązaniem tego problemu jest przedstawiony w poprzednim rozdziale algorytm Dijkstry. W niektórych zastosowaniach ścieżki mogą posiadać wagi ujemne. W takim przypadku musimy użyć nieco mniej efektywnego, lecz bardziej wszechstronnego algorytmu Bellmana-Forda. Algorytm tworzy poprawny wynik tylko wtedy, gdy graf nie zawiera ujemnego cyklu (ang. negative cycle), czyli cyklu, w którym suma wag krawędzi jest ujemna. Jeśli taki cykl istnieje w grafie, to każdą ścieżkę można "skrócić" przechodząc wielokrotnie przez cykl ujemny. W takim przypadku algorytm Bellmana-Forda zgłasza błąd.

2 Cele projektu

Celem projektu jest poznanie struktur danych jakimi są grafy oraz rozwiązanie za pomocą jednego z powyższych algorytmów problemu najkrótszej drogi (scieżki) w grafie między dwoma wierzchołkami, czyli znalezienie najkrótszego połączenia pomiędzy tymi punktami. Każda krawędź generowana pseudolosowo w programie posiada nieujemną wagę, zatem za najkrótszą drogę uznano połączenie, o najmniejszym koszcie zsumowanych wag. W zadaniu za cel postawiono sobie wyznaczenie najkrótszych scieżek łączących wszystkie wierchołki z wierzchołkiem startowym. Aby rozwiązać ten problem w programie wykorzystano algorytm Dijkstry. Grafy pseudolosowo generowane w programie to grafy nieskierowane, spójne , reprezentowane w postaci macierzy sąsiedztwa oraz listy sąsiedztwa.

Celem projektu jest również przeprowadzenie badań dotyczących średniego czasu wykonywania się wybranego algorytmu w zależności od wybranej reprezentacji grafu w pamięci komputera : za pomocą macierzy sąsiedztwa oraz listy sąsiedztwa. Badania te przeprowadzone zostaną dla 5 różnych ilości wierzchołków grafu oraz dla różnych gęstości grafu :

1. 25 %
2. 50 %
3. 75 %
4. 100 % (graf pełny)

3 Wyniki przeprowadzonych badań

3.1 Tabelki z czasem działania algorytmu Dijkstry dla reprezentacji grafu w postaci macierzy oraz listy sąsiedztwa

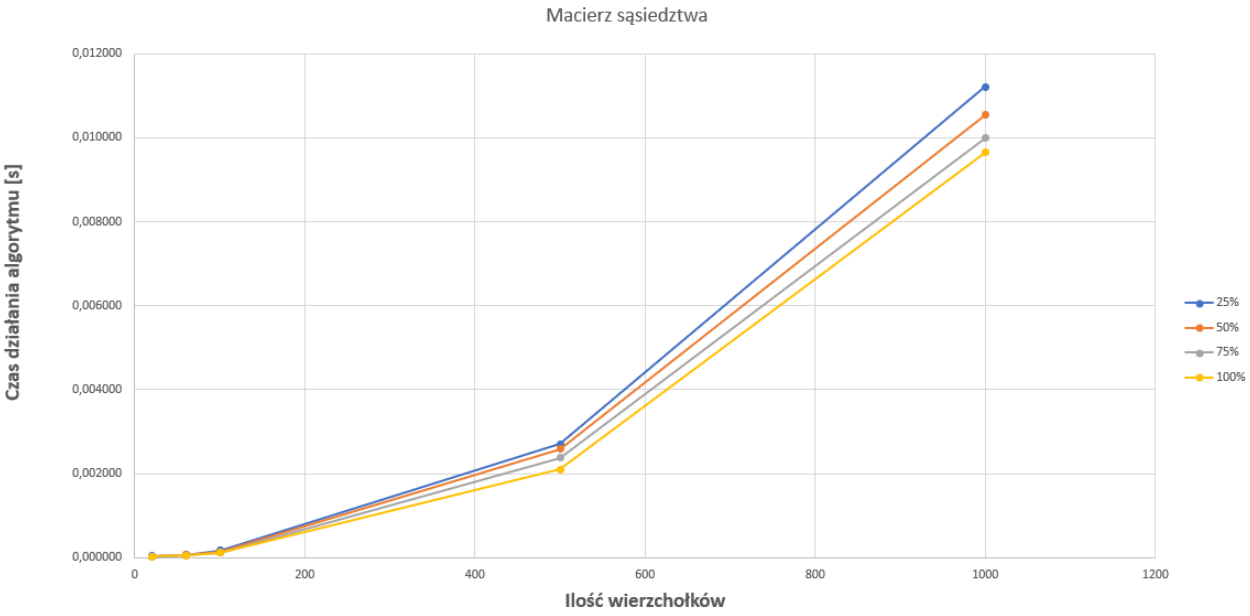
Gęstość \ Rozmiar		20	60	100	500	1000
25	%	0,000036	0,000070	0,000173	0,002706	0,011213
50	%	0,000026	0,000067	0,000129	0,002583	0,010540
75	%	0,000015	0,000050	0,000116	0,002374	0,009985
100	%	0,000034	0,000049	0,000117	0,002101	0,009647

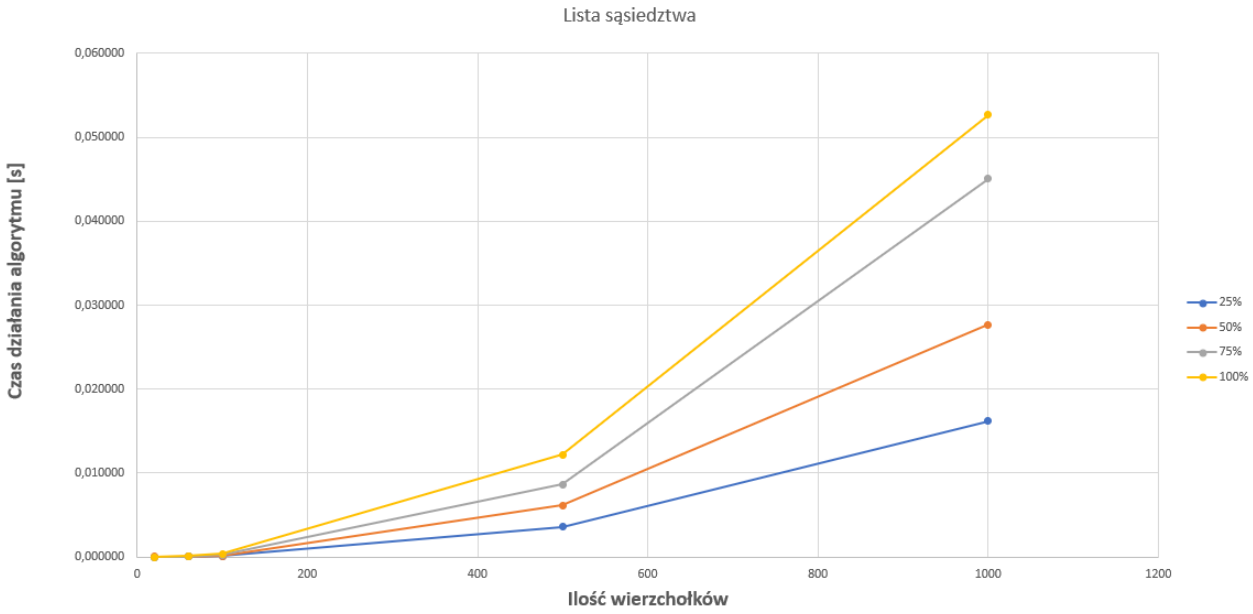
Tabelka 1 Czas działania algorytmu Dijkstry dla macierzy sąsiedztwa. Wyniki podano w sekundach.

Gęstość \ Rozmiar		20	60	100	500	1000
25	%	0,000018	0,000044	0,000088	0,003567	0,016223
50	%	0,000021	0,000051	0,000149	0,006135	0,027659
75	%	0,000016	0,000070	0,000271	0,008675	0,045057
100	%	0,000019	0,000133	0,000354	0,012219	0,052645

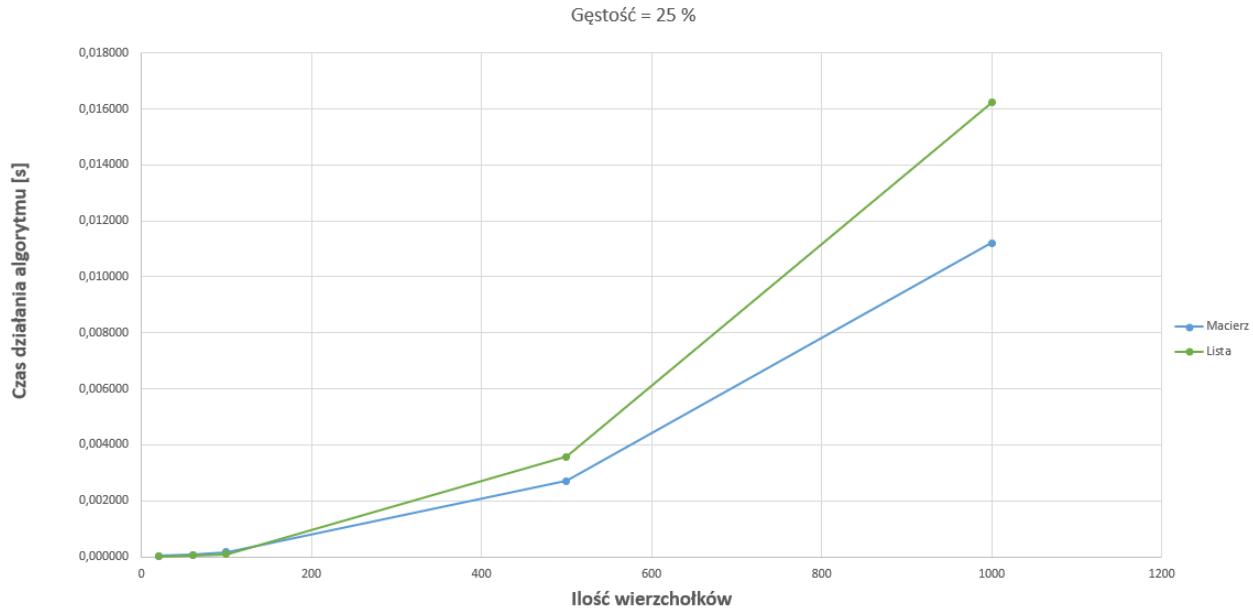
Tabelka 2 Czas działania algorytmu Dijkstry dla listy sąsiedztwa. Wyniki podano w sekundach.

3.2 Wykresy przedstawiające czas działania algorytmu Dijkstry dla reprezentacji grafu w postaci macierzy oraz listy sąsiedztwa dla różnych gęstości oraz rozmiarów grafu

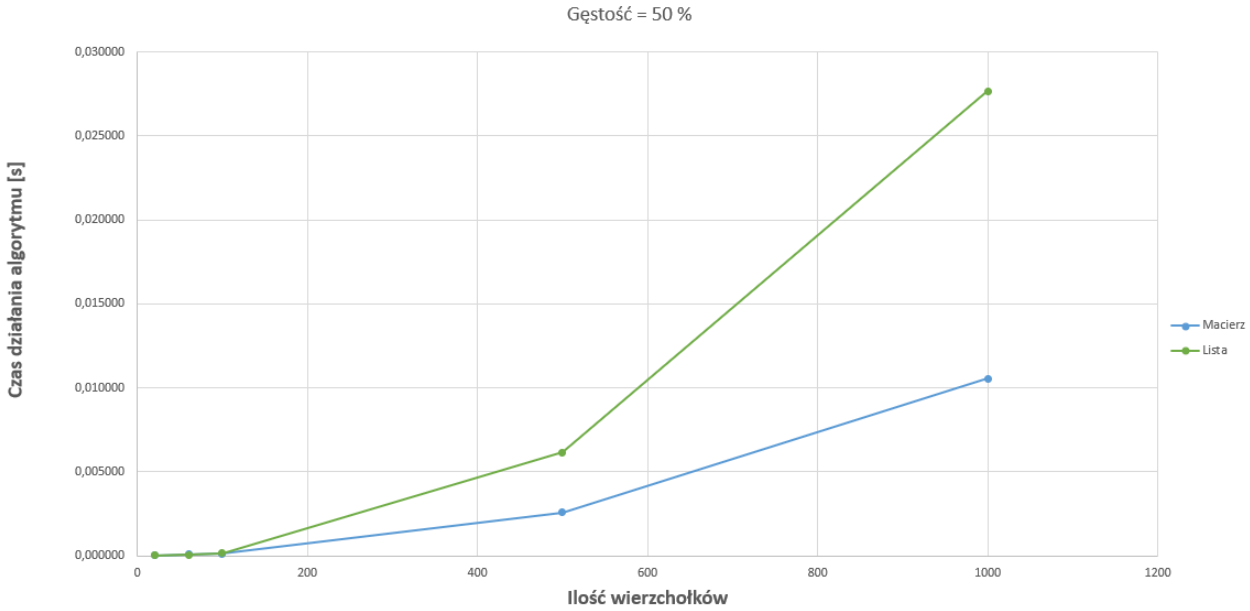




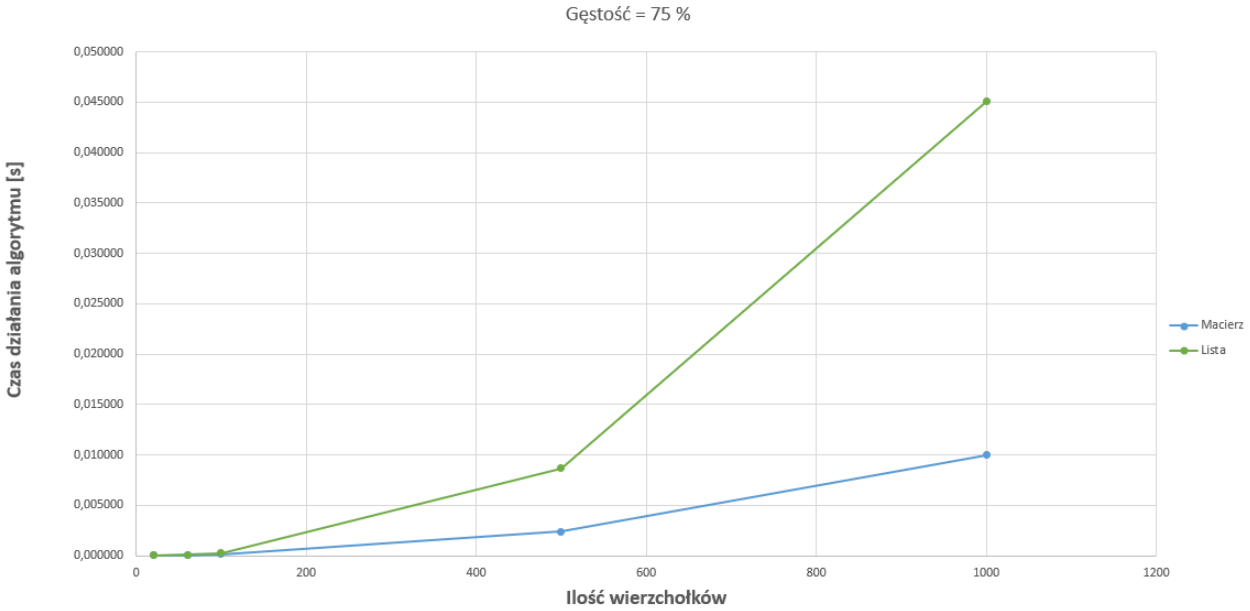
3.3 Porównanie czasu działania algorytmu Dijkstry dla reprezentacji grafu w postaci macierzy oraz listy sąsiedztwa dla różnych gęstości oraz rozmiarów grafu



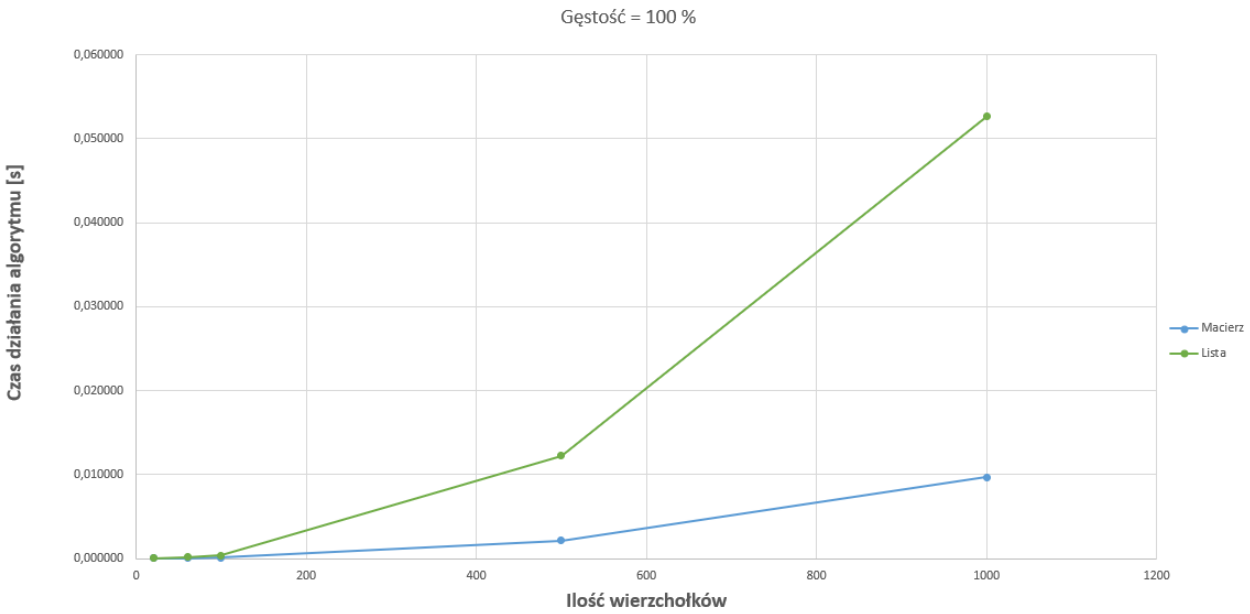
Wykres dla gęstości grafu 25%.



Wykres dla gęstości grafu 50%.



Wykres dla gęstości grafu 75%.



Wykres dla gęstości grafu 100%.

4 Wnioski

1. Algorytm Dijkstry działa dość szybko.
2. Dla reprezentacji grafu w postaci macierzy sąsiedztwa najlepsze wyniki otrzymujemy dla gęstości grafu 100%, tj. dla grafu pełnego, natomiast dla reprezentacji grafu w postaci listy sąsiedztwa dla grafu o gęstości 25%.
3. Jak widać na zamieszczonych w pkt. 3.3 wykresach algorytm Dijkstry wykonuje się szybciej dla reprezentacji grafu w postaci macierzy sąsiedztwa dla wszystkich przypadków gęstości grafu. Może to wynikać z różnic w implementacji tego algorytmu dla macierzy sąsiedztwa oraz listy sąsiedztwa w programie.
4. Analizując wykresy z pkt 3.3 można też zauważyć, że algorytm Dijkstry dla reprezentacji macierzowej grafu jest wraz ze wzrostem gęstości grafu coraz to szybszy w porównaniu do reprezentacji listowej (coraz większa różnica w czasie działania algorytmu). Można zatem stwierdzić, że algorytm ten dla coraz to większej gęstości grafu jest wydajniejszy dla macierzy sąsiedztwa w porównaniu z reprezentacją grafu w postaci listy sąsiedztwa.

Literatura

[1] https://pl.wikipedia.org/wiki/Graf_(matematyka)

[2] https://eduinf.waw.pl/inf/alg/001_search/0122.php

[3] Cormen T., Leiserson C.E., Rivest R.L., Stein C., Wprowadzenie do algorytmów, WNT

[4] http://home.agh.edu.pl/ horzyk/lectures/pi/ahdydpiwykl9.html

[5] http://www.algorytm.org/klasyczne/grafy-i-ich-reprezentacje.html

[6] http://www.cs.put.poznan.pl/arybarczyk/GrafReprezentacje.htm

[7] http://lukasz.jelen.staff.iiar.pwr.edu.pl/styled-2/page-2/index.php