WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
POLITECHNIKA WROCŁAWSKA

KIERUNEK: INFORMATYKA TECHNICZNA

Struktury danych i złożoność obliczeniowa

PROJEKT 2

Autor: Konrad Pempera

Wtorek 11:15 TP

Wrocław, maj 2023

Spis treści

1 Cel, założenia i zakres projektu

Celem projektu było zaprojektowanie i zaimplementowanie struktur danych implementujących graf, implementacja wybranych algorytmów grafowych oraz zbadanie czasu obliczeń algorytmów w zależności od wykorzystanej struktury. Prace projektowe obejmowały następujące: problemy i algorytmy:

- wyznaczenia minimalnego drzewa rozpinającego - algorytm Prima oraz Kruskala,
- wyznaczenie najkrótszej drogi w grafie - algorytm Dijkstry oraz Forda-Bellmana,
- wyznaczenie maksymalnego przepływu - algorytm Forda Fulersona (gdzie ścieżki są wyszukiwane metodami przeszukiwania grafu wszerz oraz w głąb)

2 Przyjęte założenia, język programowania, środowisko przeprowadzenia testów

Przyjęte założenia

Podczas projektowania i implementacji przyjęto następujące założenia:

- algorytmy są reprezentowane w postaci listowej oraz macierzowej,
- dla każdej z reprezentacji stworzono osobny algorytm,
- zadanie jest zrealizowane w postaci jednego programu,
- wszystkie struktury alokowane są dynamicznie,
- przepustowość krawędzi jest liczbą całkowitą,
- program jest zrealizowany w formie konsolowej.

Język programowania i wykorzystane narzędzia

Program, który jest częścią projektu został napisany w języku C++. W celu zautomatyzowania przeprowadzenia testów efektywności struktur przygotowano dodatkowe funkcje, które pozwalały uruchomienie odpowiedniego algorytmu z wybranymi parametrami takimi jak gęstość, liczba powtórzeń czy liczba wierzchołków. Implementacje oraz testy struktur przeprowadzono w środowisku Visual Studio.

Środowisko przeprowadzenia testów

Badania przeprowadzono na komputerze Lenovo ThinkPad T540p z procesorem o taktowaniu 2.50 GHz oraz systemem operacyjnym Windows 10. Badania zostały przeprowadzone po kompilacji i buildowaniu kodu w trybie relase.

3 Projekt i implementacja grafu

Graf $G = (\mathcal{V}, \mathcal{E})$ ze zbiorem węzłów $\mathcal{V}$ oraz łuków $\mathcal{E}$ może być implementowany w pamięci komputera na wiele sposobów. Najczęściej grafy implementuje się w postaci macierzy sąsiedztwa oraz w postaci list następników. W reprezentacji macierzowej graf składający się z V węzłów reprezentowany jest w postaci macierzy kwadratowej W o rozmiarze $V \times V$. Element $W[i, j]$ tej macierzy równy jest wadze krawędzi od węzła i do węzła j lub równy symbolowi „specjalnemu" gdy taka krawędź nie istnieje. W reprezentacji listowej, każdemu węzłowi u przyporządkowana jest lista zawierająca pary (v, w), gdzie v oznacza węzeł do którego prowadzi krawędź u, natomiast w waga tej krawędzi. Zarówno reprezentacja listowa jak i reprezentacja macierzowa implementują graf skierowany. Graf nieskierowany można zaimplementować dodając dwie skierowane krawędzie.

Z punktu widzenia implementowanych algorytmów, projektowane struktury grafów powinny zwracać liczbę węzłów V oraz łuków E w grafie oraz umożliwiać

- (i) dodawanie krawędzi do grafu,
- (ii) iterowanie po następnikach węzła v (w dowolnej kolejności),
- (iii) zmianę wagi łuku (u, v) i łuku skierowanego w przeciwnym kierunku (v, u).

W przypadku reprezentacji macierzowej wykonanie (i) oraz (iii) wynosi $O(1)$, natomiast iterowanie po następnikach wynosi $O(V)$ niezależnie od liczby węzłów do których prowadzą krawędzie z danego węzła. W przypadku reprezentacji listowej wykonanie (i) wynosi $O(1)$, czas wykonania (ii) proporcjonalny jest do liczby następników węzła. Modyfikacja wagi łuku (v, u) podczas zmiany wagi łuku (u, v) można zrealizować w czasie $O(1)$ pamiętając referencję do łuku (v, u) w łuku (u, v) (szczegóły zostały opisane w implementacji algorytmu Forda-Fulkersona).

Ostatecznie każda krawędź reprezentowana jest przez trójkę $(src, dst, weight)$, natomiast element listy przez czwórkę $(edge, prev, next, reversed)$, gdzie $prev$, $next$ oznaczają referencję do elementu poprzedniego i następnego w liście, $reversed$ referencję do elementu listy zawierającego krawędź przeciwstawną, $e = (src, dst, weight)$ krawędź. Oczywiście nie wszystkie pola w/w struktur są wykorzysta w implementacji każdego z algorytmów.

W implementacji grafu w wersji macierzowej wykonanie w/w czynności odbywa się poprzez odwołanie są do elementów macierzy. Natomiast w przypadku reprezentacji listowej, w/w czynności realizowane są przy pomocy metod klasy implementującej listę, która została zrealizowana w poprzednim projekcie.

Przewagą reprezentacji listowej nad macierzową jest czynność iterowania po następnikach węzła. Czas wykonania tej czynności proporcjonalny jest do faktycznej liczby następników danego węzła, a nie jak w przypadku reprezentacji macierzowej liczby węzłów. Z kolei w przypadku wersji macierzowej dostęp do wag łuków wymaga odczytu jednego elementu. W przypadku wersji listowej, należy po odwołaniu się do elementu listy, odwołać się do obiektu reprezentującego krawędź i pola $weight$.

Z w/w względów reprezentacja macierzowa jest efektywniejsza czasowa w przypadku implementacji grafów gęstych natomiast listowa rzadkich co zostało potwierdzone w rezultatach badań algorytmów stanowiących integralną część projektu.

4 Metodologia przeprowadzenie eksperymentu

Celem badań eksperymentalnych jest zbadanie czasu wykonania algorytmów grafowych w zależności postaci implementacji grafu oraz od jego gęstości grafu. W poprzednim rozdziale omówiono najistotniejsze cechy obu reprezentacji. Określenie granicy stosowalności w/w postaci implementacji wymaga przeprowadzenia testów komputerowych.

Badania komputerowe przeprowadzono dla grafów o gęstości 1%, 5%, 10%, 25%, 50%, 75%, 99%. Dla liczby węzłów od 100 do 1000 węzłów (co 100 węzłów). Dla ustalonej liczby węzłów i gęstości grafu wygenerowano pięć przykładów. Każdy przykład uruchomiono trzykrotnie. Za czas wykonania algorytmu dla danego przykłady, przyjęto najkrótszy czas wykonania. Ostatecznie na podstawie czasów wykonania algorytmów dla ustalonej liczby węzłów i gęstości grafu wyznaczono wartość średnią. Dla każdego algorytmu i każdej postaci grafu średni czas działania w zależności od liczby węzłów w grafie i gęstości grafu zostały zilustrowane na wykresach.

5 Najkrótsza ścieżka w grafie

Algorytm Dijkstry(G, s)

Krok 1 dla każdego v w G wykonaj:

Krok 1.1 $D[v] = \infty$

Krok 1.2 $path[v] = -1$

Krok 1.3 $S \leftarrow v$

Krok 2 D[s] = 0

Krok 3 dopóki $S \neq \emptyset$:

Krok 3.1 $u \leftarrow$wierzchołek w S z najmniejszym $D[u]$

Krok 3.2 dla każdego następnika v węzła u znajdującego się w S:

Krok 3.2.1 $d = D[u] + w(u, v)$

Krok 3.2.2 jeżeli $d < D[v]$ to:

Krok 3.2.2.1 $D[v] = d$

Krok 3.2.2.2 $path[v] = u$

Krok 4 zwróć D, $path$

Rysunek 1: Algorytm Dijkstry

5.1 Opis problemu i algorytmów

W problemie wyznaczania najkrótszej drogi należy wyznaczyć najkrótszą drogę pomiędzy wyróżnionym węzłem s i wszystkimi pozostałymi węzłami. Obiektem badań eksperymentalnych były dwa algorytmy algorytm Dijkstry i Belmana-Forda. Pseudo kod algorytmu Dijkstry przedstawiono na Rysunku 1, natomiast Belmana-Forda na Rysunku 2.

Danymi wejściowymi algorytmów jest graf G oraz wyróżniony wierzchołek startowy s, efektem obliczeń jest wektor z długościami najkrótszych dróg D ($D[i]$ - długość najkrótszej drogi do węzła i) oraz wektor $path$ pozwalający odtworzyć najkrótszą ścieżkę do węzłów ($path[i]$ - wierzchołek poprzedzający węzeł i w najkrótszej ścieżce do tego węzła).

W algorytmie Dijksry wykorzystywany jest zbiór S zawierający wierzchołki na którym wykonywane są następujące czynności:

- (i) dodanie wierzchołka v ($S \leftarrow v$),
- (ii) odczyt następnika węzła u jeżeli znajduje się w S,
- (iii) usunięcie węzła u o najmniejszej wadze z S ($u \leftarrow S$),
- (iv) aktualizacja wagi węzła znajdującego się S.

Do implementacji zbioru S w algorytmie wykorzystano w niewielkim stopniu zmodyfikowany kopiec, który był obiektem badań projektowych w poprzednim projekcie. Modyfikacja polegała na przyporządkowaniu każdemu węzłowi trzech atrybutów tj. numeru, wagi oraz pozycji. Bezpośrednio w tablicy implementującej kopiec K pamiętane są numery węzłów, w dodatkowej tablicy *weights* pamiętane są wagi, natomiast w kolejnej *positions* pozycje węzłów w tablicy implementującej kopiec. Porównywanie węzłów w K odbywa się za pośrednictwem wartości w *weights*. Każda zmiana pozycji węzłów w K wywołuje aktualizację *positions*. Wymienione wyżej operacje nie zwiększają złożoności obliczeniowej

Algorytm Belmana-Forda(G, s)

Krok 1 dla każdego v w G wykonaj:

Krok 1.1 $D[v] = \infty$

Krok 1.2 $path[v] = -1$

Krok 2 $D[s] = 0$

Krok 3 dla $i = 1$ do $V - 1$

Krok 3.1 dla każdego węzła u w grafie G

Krok 3.1.1 dla każdego następnika v węzła u:

Krok 3.1.1.1 $d = D[u] + w(u, v)$

Krok 3.1.1.2 jeżeli $d < D[v]$ to:

Krok 3.1.1.2.1 $D[v] = d$

Krok 3.1.1.2.2 $path[v] = u$

Krok 4 dla każdego węzła u w grafie G

Krok 4.1 dla każdego następnika v węzła u:

Krok 4.1.1 jeżeli $D[v] > D[u] + w(u, v)$ to

Krok 4.1.1.1 zwróć -1

Krok 5 zwróć D, $path$

Rysunek 2: Algorytm Belmanna - Forda

podstawowych funkcji kopca. Dodatkowo umożliwiają: wyznaczenie pozycji węzła w kopcu i sprawdzenie czy się w nim znajduje w czasie $O(1)$. Ostatecznie wykonanie czynności (i,ii,iv) zajmuje $O(lnV)$, natomiast (ii) zajmuje $O(1)$.

W Kroku 1 algorytmu Dijkstry najbardziej czasochłonnym krokiem jest krok 1.1. Ze względu na uporządkowanie kopca S zajmuje on $O(lnV)$. Ostatecznie oszacowanie czasu wykonania całego kroku 1 wynosi $O(V \cdot lnV)$. Krok 3 algorytmu Dijkstry wykonywany jest V - krotnie. Wyznaczenia węzła u w kroku 3.1 wynosi $O(lnV)$, natomiast liczba wykonań kroku 3.2 równa jest liczbie następników tego węzła. Najdłużej trwającym krokiem jest aktualizacja wartości D, która wiąże się uporządkowaniem kopca S i zajmuje $O(lnV)$. Ostatecznie oszacowanie czas wykonania całego kroku 3 wynosi $O((V + E) \cdot lnV)$.

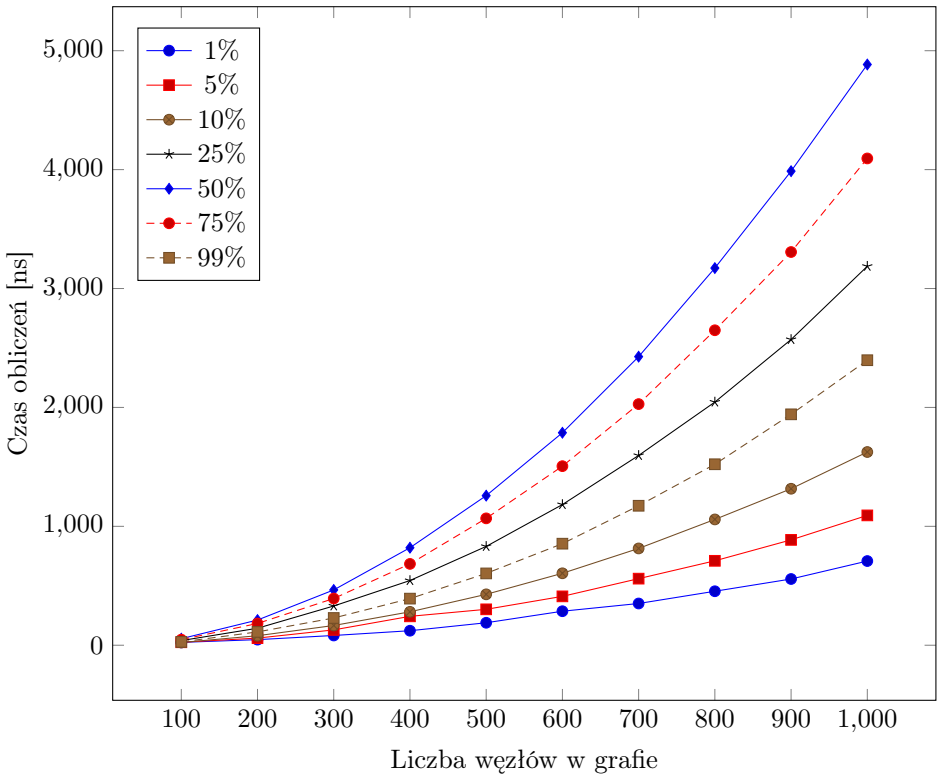
Podobnie jak w algorytmie Dijkstry liczba wykonań kroku 3.1.1 w algorytmie Belmana-Forda równa jest liczbie następników tego węzła. Zatem wykonanie kroku 3.1.1 zajmuje $O(E)$. Ostatecznie wykonanie kroku 3 zajmuje $O(V \cdot E)$. Z podobnej analizy wynika, że wykonanie kroku 4 wynosi $O(E)$.

Algorytm Dijkstry może być stosowany wyłącznie do grafów nie zawierających krawędzi o ujemnych wagach. Algorytm Belmana-Forda może być zastosowany do grafów o dowolnych wagach, przy czym w przypadku wystąpienie w grafie cykli o ujemnej długości (dla takich grafów nie można wyznaczyć najkrótszych ścieżek) zwraca wartość -1 (Krok 4. algorytmu).

5.2 Wyniki eksperymentu komputerowego

Wyniki pomiaru czasu działania algorytmów w zależności od liczby węzłów grafie i gęstości grafu przedstawiono na:

- algorytm Dijkstry w reprezentacji macierzowej - Rysunek 3
- algorytm Dijkstry w reprezentacji listowej - Rysunek 4



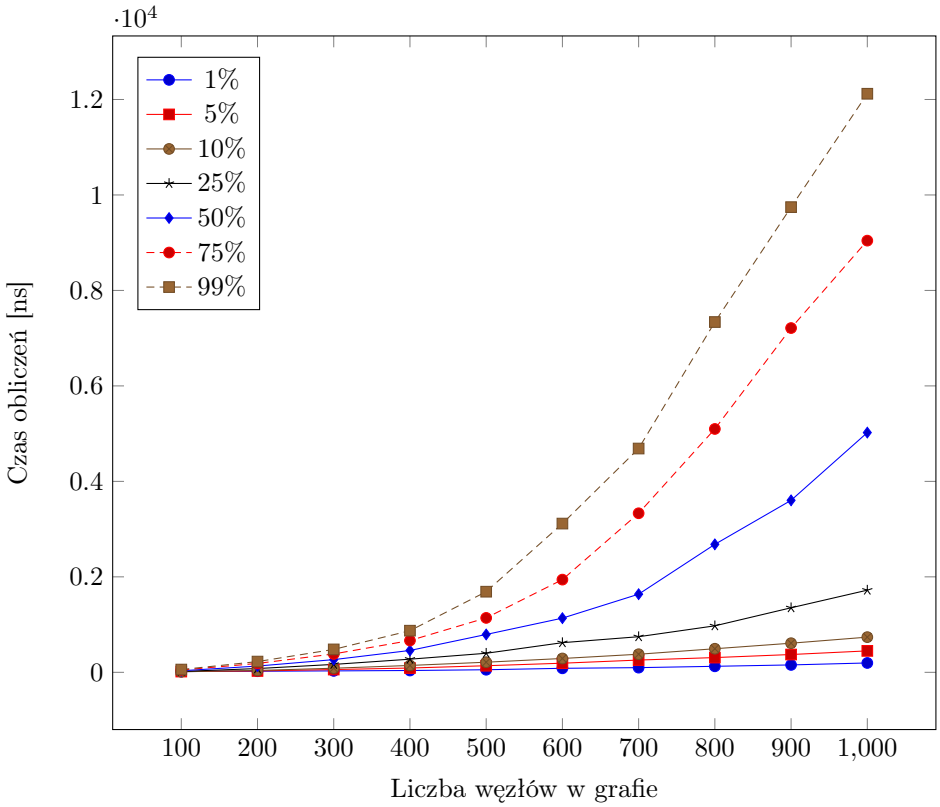
Rysunek 3: Algorytm Dijkstry - reprezentacja macierzowa grafu

- algorytm Belmanna - Forda w reprezentacji macierzowej - Rysunek 5
- algorytm Belmanna - Forda w reprezentacji listowej - Rysunek 6

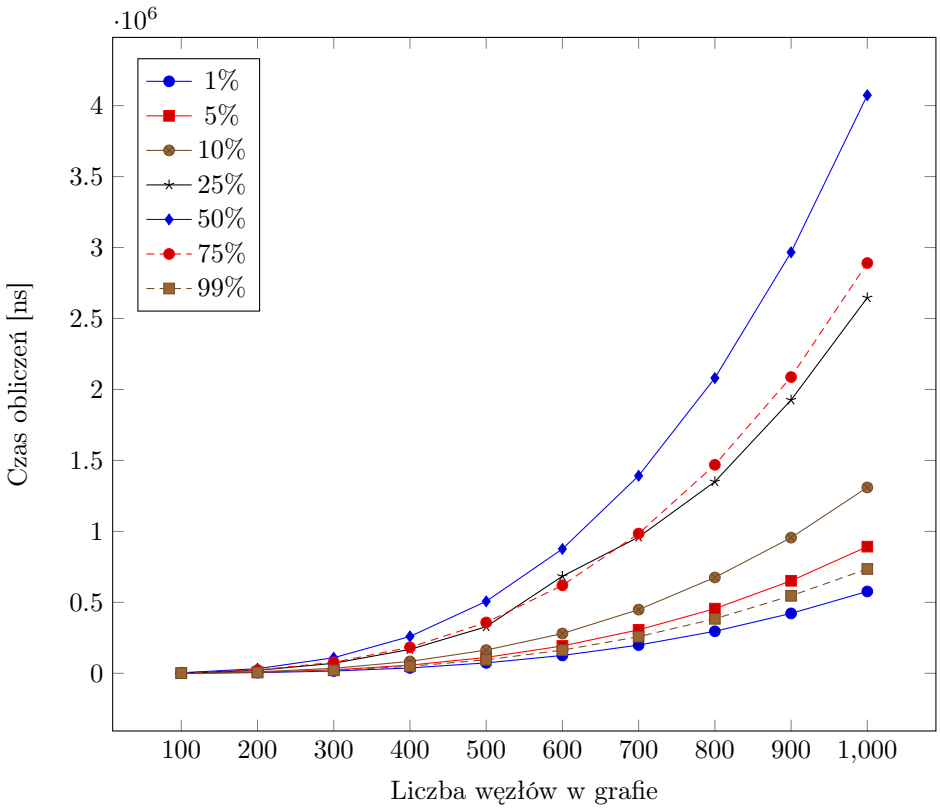
Szczegółowe czasy dotyczące problemu znajdowania najkrótszej ścieżki zostały umieszczone w dodatku w Tabelach 1, 2, 3, 4.

Porównując algorytm Dijikstry oraz Belmanna-Forda, które dotyczą problemu wyszukiwania najkrótszej drogi, można zauważyć, że algorytm Dijkstry jest znacznie szybszy od algorytmu Belmanna - Forda. Dla przykładu, czas wykonania algorytmu przy 500 węzłach i gęstości 50% w przypadku algorytmu Dijkstry jest 100 razy krótszy niż w przypadku algorytmu Belmanna - Forda.

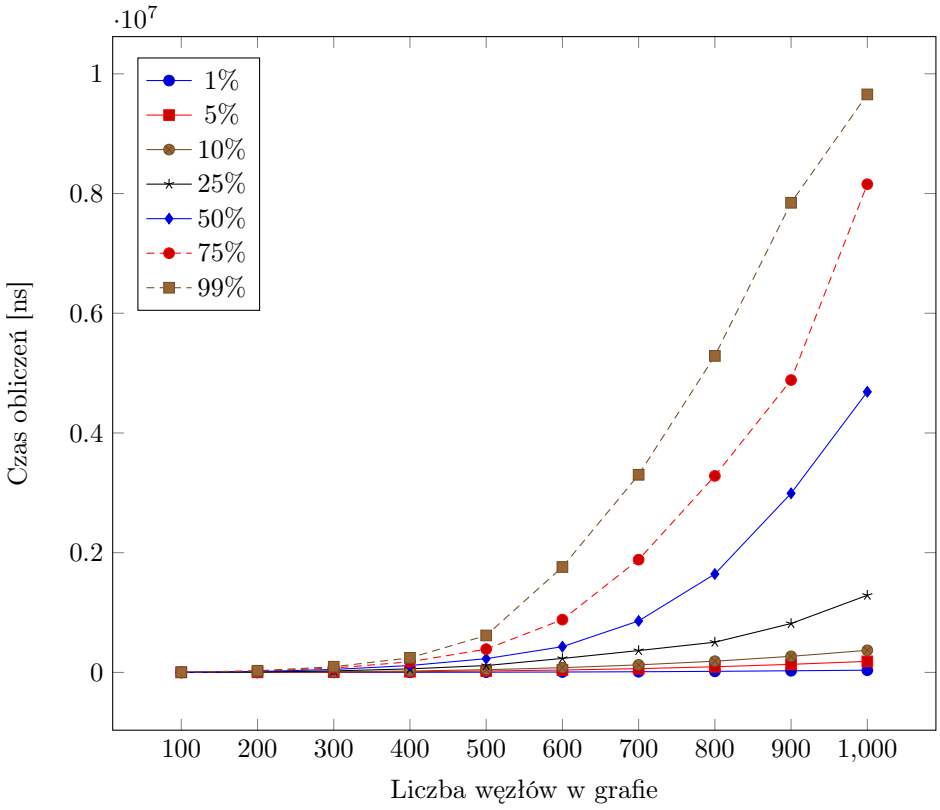
Na Rysunku 7 przedstawiono czasy wykonywania algorytmu Dijkstry dla małych gęstości grafu i o różnych implementacji grafu. Analizując ten wykres, można dostrzec, że reprezentacja listowa dla gęstości grafu do 10% jest bardziej efektywna czasowo niż reprezentacja macierzowa.



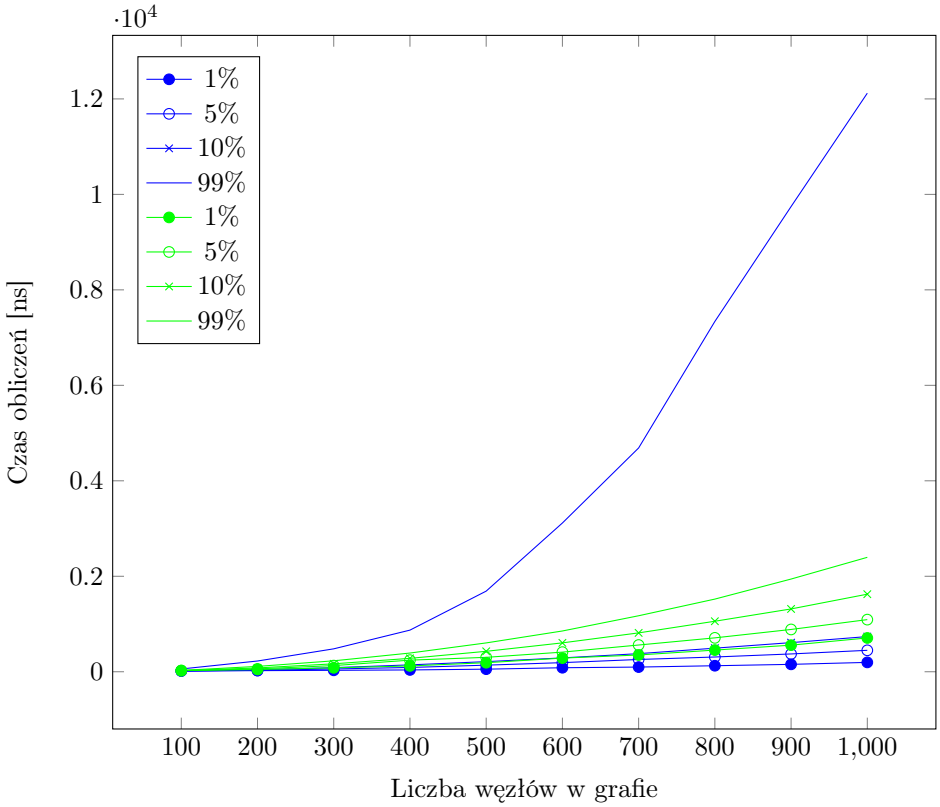
Rysunek 4: Algorytm Dijkstry - reprezentacja listowa grafu



Rysunek 5: Algorytm Belmanna - Forda - reprezentacja macierzowa



Rysunek 6: Algorytm Belmanna - Forda - reprezentacja listowa



Rysunek 7: Algorytm Dijkstry - porównanie czasów obliczeń dla grafów rzadkich (niebieski - implementacja listowa , zielony - implementacja macierzowa)

Algorytm Kruskal(G)

Krok 1 dla każdej krawędzi e w G wykonaj:

Krok 1.1 $S \leftarrow e$

Krok 2 $Makeset(V, rank, parent)$

Krok 3 $cost = 0$

Krok 4 dopóki S nie jest puste:

Krok 4.1 $e = (src, dest, weight) \leftarrow$ krawędź z S o najmniejszej wadze

Krok 4.2 jeżeli $findset(parent, src)! = findset(parent, dest)$:

Krok 4.2.1 $Union(parent, rank, src, dest)$

Krok 4.2.2 $cost = cost + weight$

Krok 4.2.3 $path[dest] = src$

Krok 5 return $cost$, $path$

Rysunek 8: Algorytm Kruskala

6 Minimalne drzewo rozpinające

6.1 Opis problemu i algorytmów

W problemie wyznaczenia minimalnego drzewa rozpinającego należy wyznaczyć drzewo rozpinające graf (zawierające wszystkie węzły) o minimalnej sumie wag krawędzi. Obiektem badań eksperymentalnych były dwa algorytmy algorytm Kruskala i Prima. Pseudo kod algorytmu Kruskala przedstawiono na Rysunku 8. Algorytm Prima jest algorytmem którego schemat jest identyczny z algorytmem Dijkstry, różni się on jedynie realizacją kroku 3.2.2 oraz wyliczeniem kosztu (sumy krawędzi) drzewa.

Danymi wejściowymi algorytmów jest graf G. Algorytm zwraca koszt drzewa oraz oraz wektor $path$ pozwalający odtworzyć drzewo tj. ($path[i]$ - wierzchołek do którego prowadzi krawędź od węzeł i w drzewie).

W algorytmie Kruskala do implementacji zbioru S wykorzystano zmodyfikowany kopiec w którym do węzła przyporządkowana została trójka $(src, dst, weight)$. Wierzchołki kopca porównywane były względem wartości $weight$. Kolejnym ważnym elementem algorytmu Kruskala jest stwierdzenie czy łuk łączy dwa węzły należące do różnych drzew. W tym celu zaimplementowano następujące funkcje:

- Makeset - dla każdego wierzchołka przyporządkowuje rodzica w postaci swojego numery oraz przyporządkowuje rangę 0,
- Findset - przyporządkowuje wierzchołkowi numer korzenia, do którego należy, jednocześnie kompresuje ścieżkę,
- Union - łączy dwa zbiory i modyfikuje rangę zbioru końcowego.

Jest to jedna z najszybszych metod implementujących operacje na zbiorach rozłącznych.

Wykonanie kroku 1 zajmuje łącznie $O(E \cdot lnE)$. Wykonanie kroku 2 zajmuje $O(V)$. Wykonanie funkcji FindSet oraz Union nie jest gorsze od $O(lnE)$, natomiast kroku 4.1 wynosi $O(lnE)$. Ostatecznie oszacowanie czasu wykonania kroku 4 wynosi $O(E \cdot lnE)$. Analiza złożoności obliczeniowej algorytmu Prima jest podobna do algorytmu Dijkstry.

6.2 Eksperymenty komputerowe

Wyniki pomiaru czasu działania poszczególnych algorytmów w zależności od liczby węzłów grafie i gęstości grafu przedstawiono na:

Algorytm Prim(G)

Krok 3.2.2 $d = w(u, v)$

Krok 3 $cost = 0$

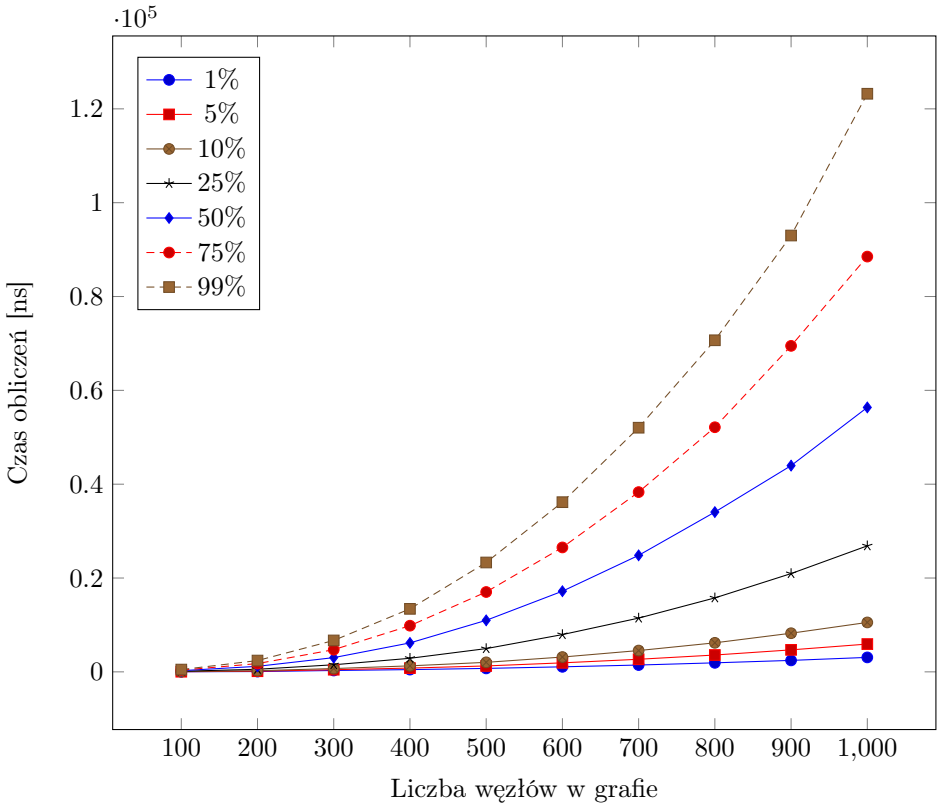
Krok 3.1 dla każdego węzła u w grafie G

Krok 3.1 $cost = cost + w(path[u], u)$

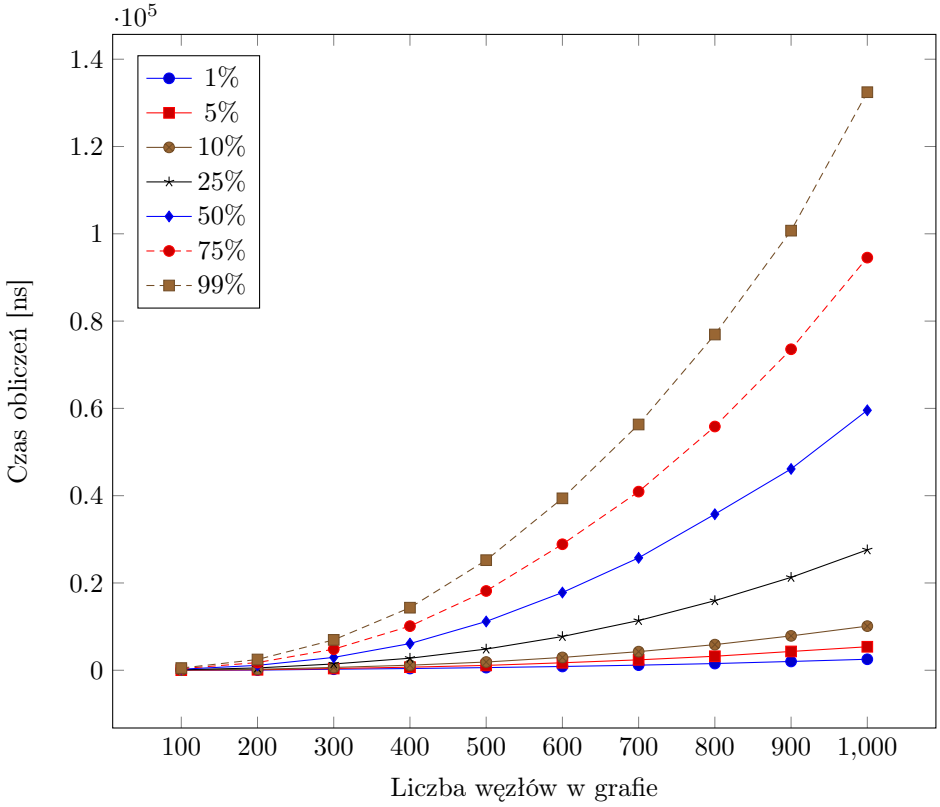
Rysunek 9: Algorytm Prima

- algorytm Kruskala w reprezentacji macierzowej - Rysunek 10,
- algorytm Kruskala w reprezentacji listowej - Rysunek 11,
- algorytm Prima w reprezentacji macierzowej - Rysunek 12,
- algorytm Prima w reprezentacji listowej - Rysunek 13.

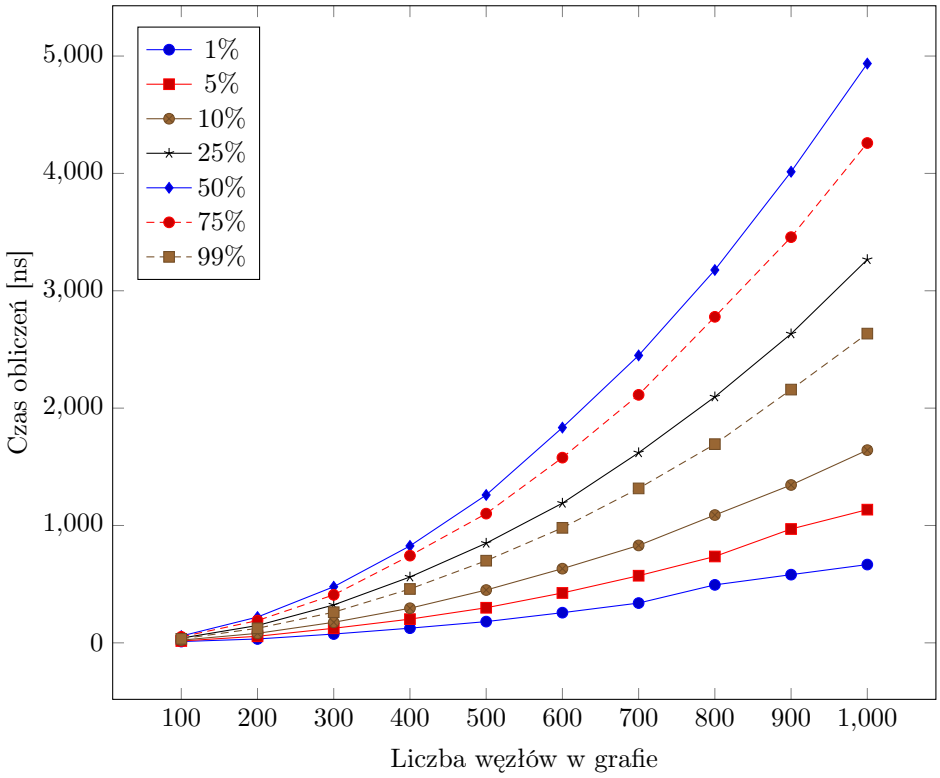
Szczegółowe czasy wykonywania algorytmów wyszukujących minimalne drzewo rozpinające w zależności od gęstości grafu i liczby jego wierzchołków zostały umieszczone w dodatku w Tabelach 5, 6, 7, 8. Analizując wykresy algorytmów dotyczących wyszukiwania minimalnych drzew rozpinających z łatwością można dostrzec, że algorytm Prima jest bardziej efektywny od Algorytmu Kruskala. Czas wykonywania algorytmu Prima jest około dwa razy krótszy. Dla przykładu, czas wykonywania algorytmu Prima dla gęstości grafu równej 50% i liczbie wierzchołków równej 1000 w reprezentacji listowej wynosi 8000ms, podczas gdy czas wykonywania algorytmu Kruskala dla takich samych danych wejściowych wynosi 20 000ms. Reprezentacja listowa obydwóch algorytmów cechuje się lepszą wydajnością w przypadku małych gęstości niż reprezentacja macierzowa. W odróżnieniu od algorytmu Prima, w przypadku algorytmu Kruskala nie ma większych różnic w czasach wykonania pomiędzy reprezentacjami. Na Rysunku 14 zestawiono czasy wykonania algorytmu Prima dla małych gęstości grafu i różnych reprezentacji grafu. Analizując dane, można zauważyć, że reprezentacja listowa dla gęstości grafu do 25% jest bardziej efektywna czasowo od reprezentacji macierzowej.



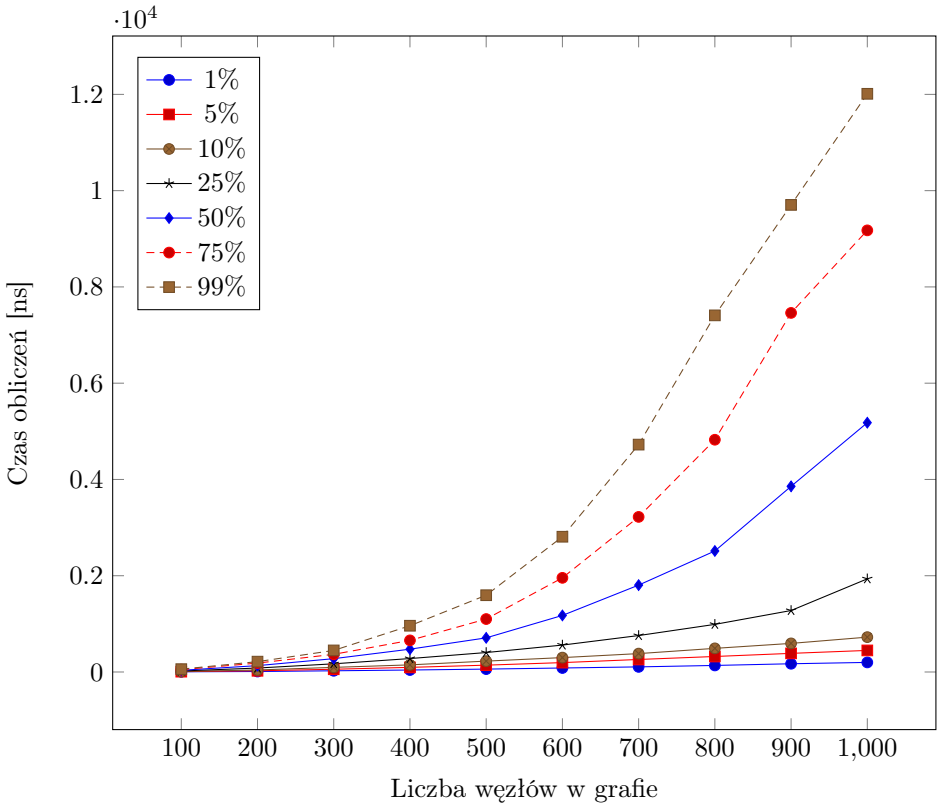
Rysunek 10: Algorytm Kruskala - reprezentacja macierzowa grafu



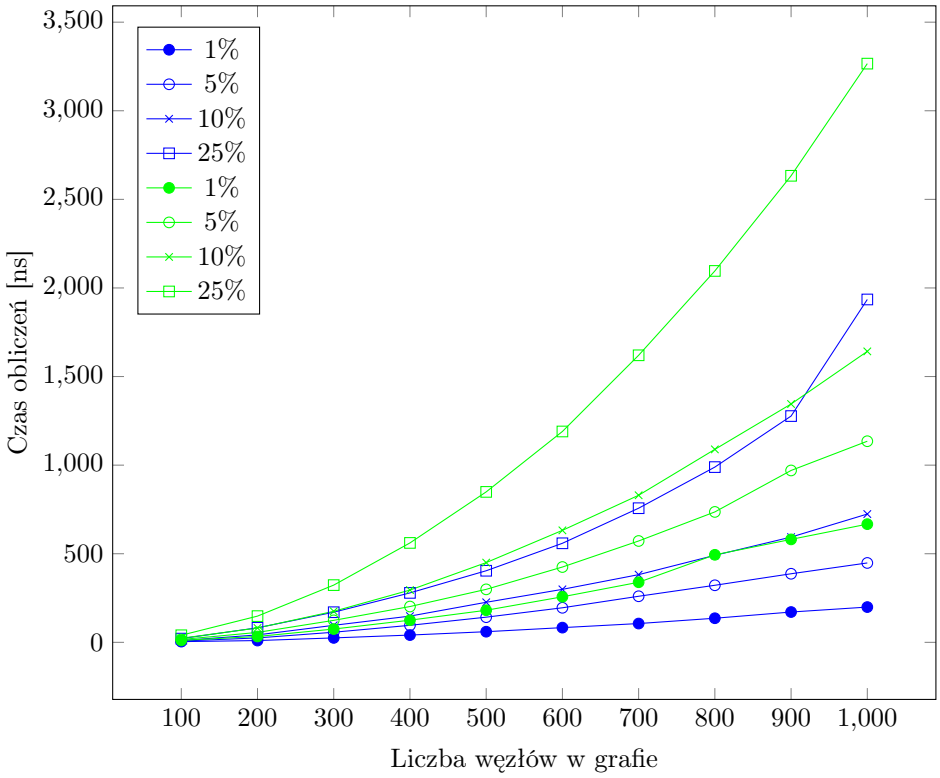
Rysunek 11: Algorytm Kruskala - reprezentacja listowa grafu



Rysunek 12: Algorytm Prima - reprezentacja macierzowa grafu



Rysunek 13: Algorytm Prima - reprezentacja listowa grafu



Rysunek 14: Algorytm Prima - czasów obliczeń dla grafów rzadkich
(niebieski - implementacja listowa, zielony - implementacja macierzowa)

7 Maksymalny przepływ w sieci

7.1 Opis problemu i algorytmów

W problemie maksymalnego przepływu w sieci należy znaleźć przepływ o maksymalnej wartoci z wyróżnionego wierzchołka początkowego s do wyróżnionego wierzchołka t. Sieć modeluje się w postaci grafu G gdzie wagi krawędzi określają przepustowości krawędzi. Obiektem badań był algorytm Forda-Fulkersona z dwoma metodami znajdowania ścieżki powiększającej tj. przez przegląd grafu wszerz i wgłąb.

W algorytmie Forda-Fulkersona główne obliczenia odbywają się na tzw. grafie rezidualnym R. Graf R składa się z tego samego zbioru węzłów oraz łuków zgodnych z łukami w G o wadze równej przepustowości krawędzi (wadze krawędzi w G) pomniejszonej o aktualny przepływ przez tą krawędź oraz krawędzi skierowanej przeciwnie o wadze równej wartości przepływu przez tą krawędź. W celu znalezienia ścieżki od węzła s do węzła t (wzdłuż, której można zwiększyć przepływ) można wykorzystać algorytm przeglądu grafu wgłąb lub wszerz operujący na grafie R. W algorytmach tych brane są pod uwagę tylko krawędzie o niezerowej wadze.

Danymi wejściowymi algorytmu jest graf G oraz węzły s i t. Algorytm zwraca maksymalny przepływ w siei z węzła s do t. Schemat algorytmu Forda-Fulkersona przedstawiono na Rysunku 15.

W implementacji macierzowej ścieżka powiększająca implementowana jest w postaci wektora $path$ w którym $path[i]$ zawiera węzeł poprzedzający węzeł i w tej ścieżce (wyznaczonej w Kroku 2.3). W konstrukcji grafu rezidualnego R (Krok 1), dodanie pary łuków do grafu R odpowiadającej łukowi (u, v) w grafie G polega na przypisaniu $R.T[u, v] = G.T[u, v]$. Równie prosta jest aktualizacja wag krawędzi w krokach 2.4.1 oraz 2.4.2.

W przypadku implementacji listowej ścieżka implementowana jest w postaci wektora $path$, w którym $path[i]$ zawiera krawędź prowadzącą do węzła i w ścieżce powiększającej. Dodanie pary łuków do grafu R realizowane jest przez utworzenie dwóch elementów list zawierających odpowiednio krawędzie skierowane krawędzie o wagach $w(u, v)$ i 0, przypisanie wzajemnych referencji oraz dodanie do list następników odpowiednich węzłów. Wówczas, modyfikację wartości wag krawędzi realizowaną w krokach

Ford-Fulkerson(G, s, t)

Krok 1 Zbuduj graf rezidualny R dla zerowego przepływu w sieci G

Krok 2 Wykonaj

Krok 2.1 znajdź ścieżkę powiększającą *path* w R

Krok 2.2 jeżeli nie istniej to przerwij

Krok 2.3 wyznacz najmniejszą wagę *minweight* krawędzi w *path*

Krok 2.4 dla każdej krawędzi (u, v) ze ścieżki *path* wykonaj

Krok 2.4.1 $w(u, v) = w(u, v) - minweight$

Krok 2.4.2 $w(v, u) = w(v, u) + = minweight$

Krok 3 $F = 0$

Krok 4 dla każdego następnika x węzła t grafu R

Krok 4.1 $F = F + w(t, x)$

Krok 5 zwróć F

Rysunek 15: Algorytm Forda - Fulkersona

2.4.1 i 2.4.2 dla węzła v można zrealizować w czasie $O(1)$ przez podstawienia:

- path[v].reversed.dge.weight += minweight,
- path[v].edge.weight -= minweight.

Wykonanie kroku 1 zajmuje $O(V+E)$. Wyznaczenie ścieżki powiększającej metodą przeglądu wszerz lub wgłąb (Krok 2.1) zajmuje $O(E)$, natomiast czas wyznaczenia minimalnej wagi krawędzi w ścieżce (Krok 2.3) oraz uaktualnienie wag realizowane w krokach 2.4.1 i 2.4.2 proporcjonalny jest do liczby łuków w ścieżce która nie jest większa od E. Ostatecznie oszacowanie wykonania jednej iteracji kroku 2 wynosi $O(E)$.

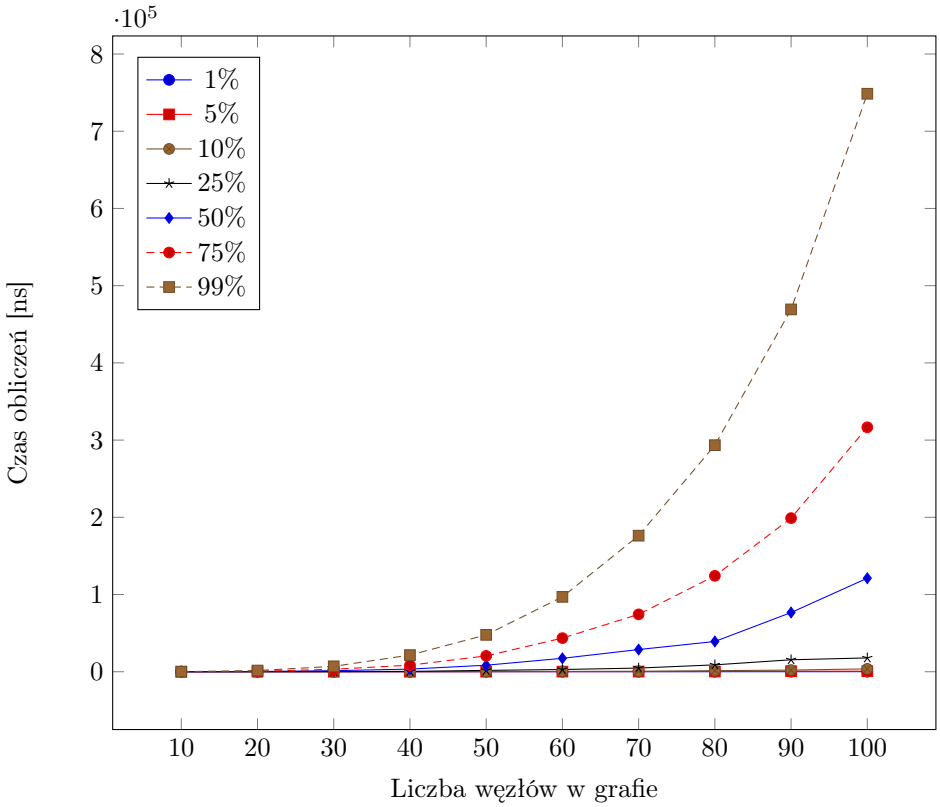
Niestety, w ogólnym przypadku, nie jesteśmy w stanie na podstawie liczby węzłów i/lub łuków oszacować liczby ścieżek powiększających znajdowanych w podczas działania algorytmu w kroku 2.1.

7.2 Eksperymenty komputerowe

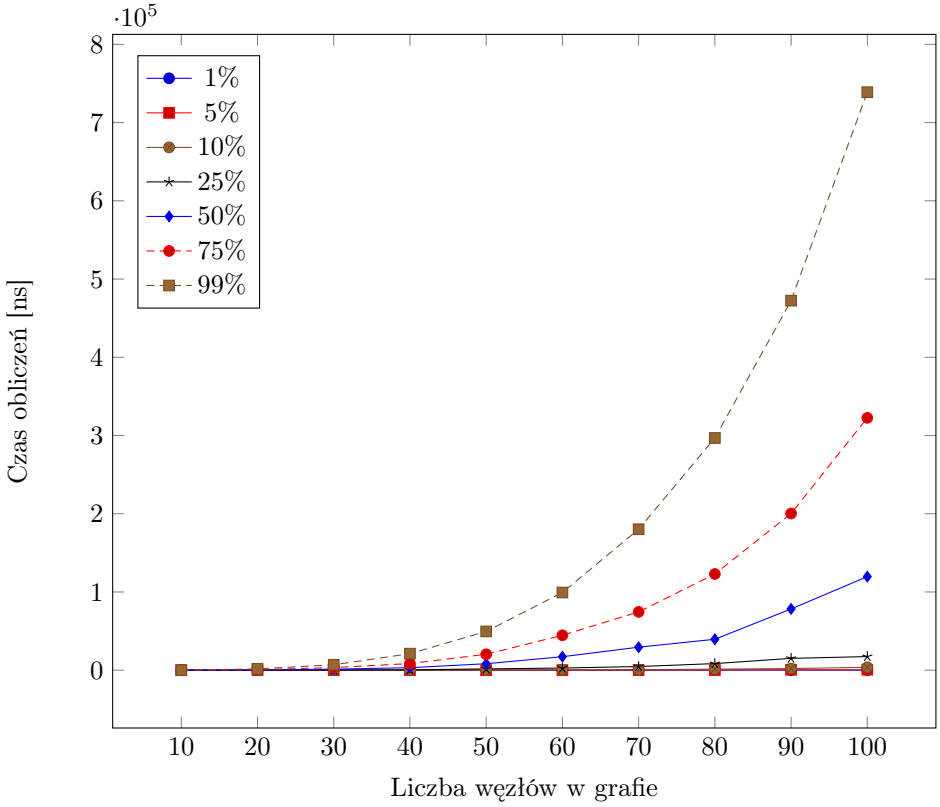
Wyniki pomiaru czasu działania poszczególnych algorytmów w zależności od liczby węzłów grafie i gęstości grafu przedstawiono na:

- algorytm Forda - Fulkersona w reprezentacji macierzowej oraz z wykorzystaniem algorytmu przeszukiwania grafu wgłąb - Rysunek 16,
- algorytm Forda - Fulkersona w reprezentacji macierzowej oraz z wykorzystaniem algorytmu przeszukiwania grafu wszerz - Rysunek 17,
- algorytm Forda - Fulkersona w reprezentacji listowej oraz z wykorzystaniem algorytmu przeszukiwania grafu wgłąb - Rysunek 18,
- algorytm Forda - Fulkersona w reprezentacji listowej oraz z wykorzystaniem algorytmu przeszukiwania grafu wszerz - Rysunek 19.

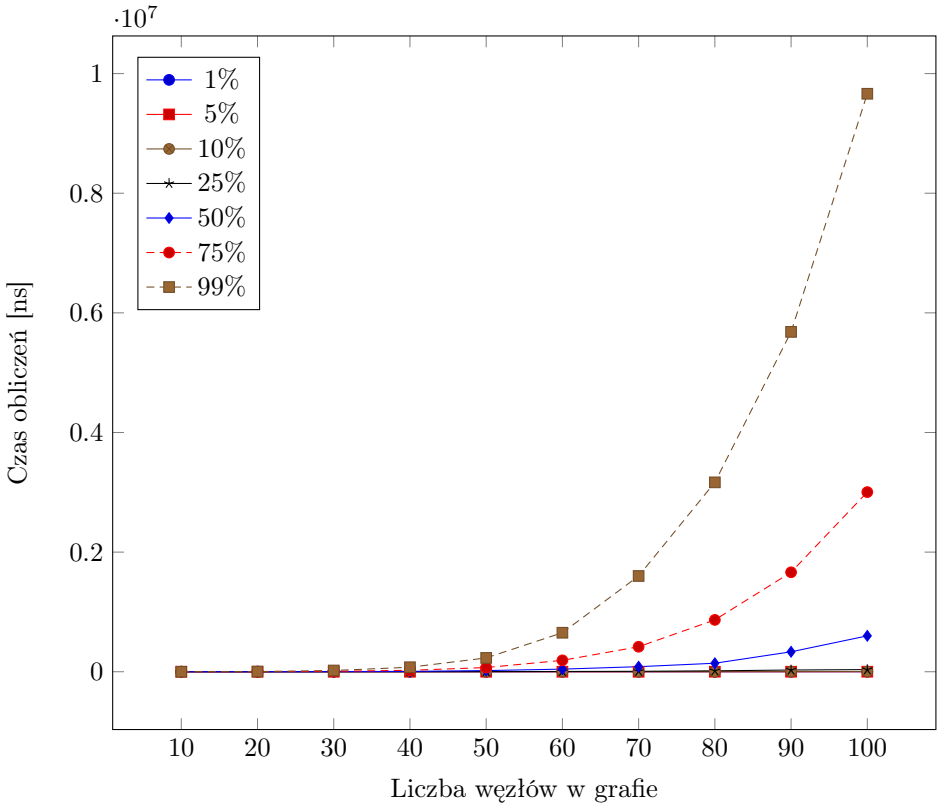
Szczegółowe czasy wykonywania algorytmów wyszukujących minimalne drzewo rozpinające w zależności od gęstości grafu i liczby jego wierzchołków zostały umieszczone w dodatku w Tabelach 9, 10, 11, 12.



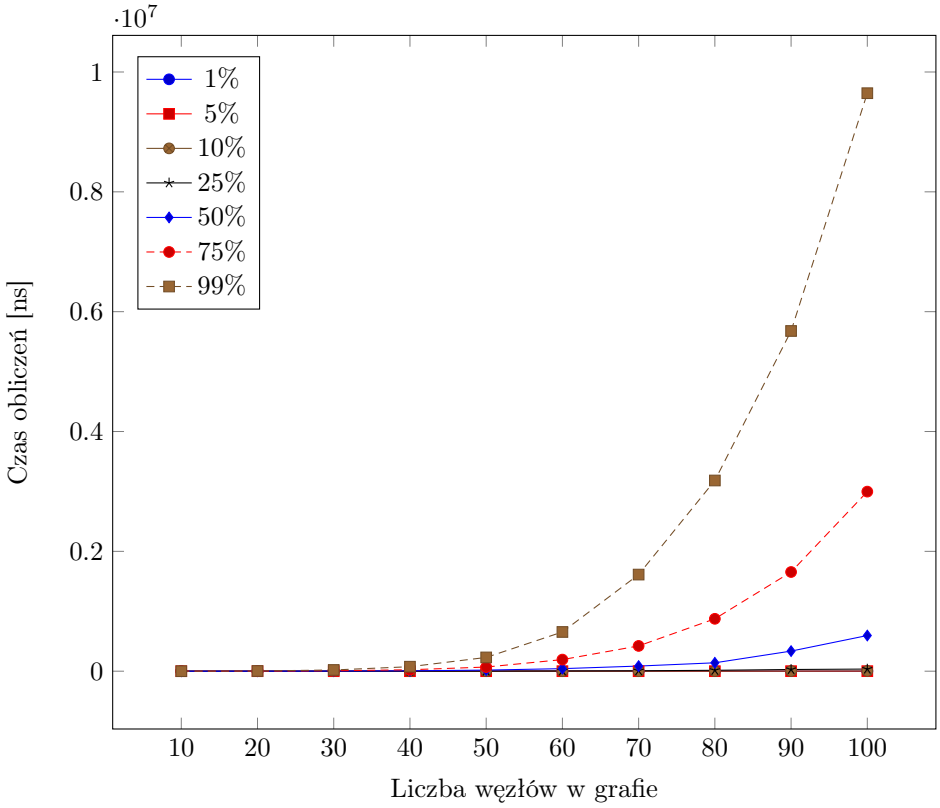
Rysunek 16: Algorytm Forda - Fulkersona z przeszukiwaniem grafu w głąb - reprezentacja macierzowa



Rysunek 17: Algorytm Forda - Fulkersona z przeszukiwaniem grafu wszerz - reprezentacja macierzowa



Rysunek 18: Algorytm Forda - Fulkersona z przeszukiwaniem grafu w głąb - reprezentacja listowa



Rysunek 19: Algorytm Forda - Fulkersona z przeszukiwaniem grafu wszerz - reprezentacja listowa

Analizując wykresy zawierające dane dotyczące pomiarów algorytmu Forda - Fulkersona w różnych implementacjach można zauważyć, że czasy wykonywania algorytmów w implementacji z użyciem algorytmu przeszukiwania grafu w głąb i wszerz są do siebie zbliżone. Spoglądając na Tabele 10 oraz 12 zawierające wyniki pomiarów można dostrzec niewielkie różnice tych czasów. Różnice w czasach wykonywania algorytmów można dostrzec w przypadku różnych reprezentacji, szczególnie w przypadkach, gdzie gęstość grafu jest większa niż 25%. W takich przypadkach reprezentacja macierzowa cechuje się lepszą wydajnością. Dla przykładu dla gęstości 99% i 100 wierzchołków średni czas wykonania algorytmu dla reprezentacji macierzowej (738847 ns), jest 13 razy krótszy od czasu wykonania tego algorytmu dla reprezentacji listowej (9646097 ns).

8 Podsumowanie

Z przeprowadzonych badań wynika, że reprezentacja macierzowa jest efektywniejsza strukturą reprezentującą graf dla gęstości powyżej 25%. Zależność ta dotyczy wszystkich testowanych algorytmów co pozwala wysunąć wniosek, iż macierz jest lepszą strukturą do przechowywania grafów o gęstości powyżej 25%. W praktyce bardzo dużo problemów grafowych dotyczy grafów rzadkich (bardzo rzadkich) np. problem wyznaczania najkrótszej drogi w połączeniach drogowych cechuje się bardzo dużą liczbą węzłów reprezentujących skrzyżowania dróg i niewielką liczbą łuków, które reprezentują drogi wychodzące z skrzyżowań. W Polsce mamy setki tysięcy skrzyżowań, gdzie zdecydowana większość, to takie gdzie liczba dróg wychodzących jest niewiększa niż cztery. Bardzo rzadko spotyka się skrzyżowania gdzie liczba dróg wychodzących jest większa niż 10.

Drugim istotnym wnioskiem, niewynikającym bezpośrednio z wyników prezentowanych w projekcie jest zależność czasu obliczeń od danych przykładów testowych. Złożoność obliczeniowa algorytmu Dijkstry wynosi $O((E+V)\cdot log_2 V)$. Dla grafu o gęstości 50% składającego się z $V = 1000$ węzłów spodziewana liczba elementarnych czynności wynosi w przybliżeniu $(V + 0.5 \cdot V^2) \cdot log_2 V \approx 5000000$, natomiast dla grafu pełnego wynosi $(V+V^2)\cdot log_2 V \approx 10000000$. Tymczasem faktyczna liczba elementarnych czynności wykonywanych na najbardziej czasochłonnej strukturze (kopcu) wynosiła odpowiednio ok. 26900 (gęstość 50%) i ok. 27400 (graf pełny). Wartości te stanowią tylko 0.538% oraz 0.274% wyznaczonej na podstawie złożoności obliczeniowej liczby zmian. Zatem wraz ze wzrostem gęstości grafu obserwuje się zmniejszenie frakcji faktycznej liczby zmian w stosunku do teoretycznej.

Najefektywniejszym algorytmem do wyszukiwania najkrótszej drogi w grafie jest algorytm Dijkstry. W przypadku wyszukiwania najmniejszego drzewa rozpinającego największą efektywnością cechuje się algorytm Prima. Najdłużej działającym algorytmem jest algorytm Forda-Fulkersona. Ze względu na długi czas obliczeń badania przeprowadzono na przykładach zawierających niewięcej niż 100 węzłów (przykłady zawierające 100 węzłów były najmniejszymi testowanymi w przypadku innych algorytmów).

9 Bibliografia

- https://www.algorytm.edu.pl/matura-informatyka/zlozonosc-algorytmu
- Cormen Thomas H., Leiserson Charles E., Rivest Ronald L, Clifford Stein, Wprowadzenie do algorytmów, Wydawnictwo Naukowe PWN, Warszawa, 2022.
- Materiały dydaktyczne doktora Jarosława Mierzwy
- https://home.math.uni.lodz.pl/horzel/wp-content/uploads/sites/3/2020/03/3_drzewa_RB_splay.pdf
- https://inf.ug.edu.pl/ pmp/Z/ASDwyklad/czczWUd.pdf
- https://en.cppreference.com/w/cpp/chrono

10 Dodatek

Tabela 1: Algorytm Dijkstry - reprezentacja macierzowa grafu

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
100	24	25	23	39	55	47	28
200	47	60	80	143	212	183	112
300	82	129	166	330	465	393	229
400	122	243	281	544	820	684	392
500	189	302	428	831	1258	1067	604
600	427	411	605	1183	1787	1506	854
700	351	560	814	1596	2427	2028	1173
800	454	710	1058	2045	3172	2649	1522
900	557	886	1316	2571	3987	3307	1942
1000	708	1092	1625	3187	4884	4094	2397

Tabela 2: Algorytm Dijkstry - reprezentacja listowa grafu

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
100	16	16	12	18	33	48	58
200	23	29	39	79	131	186	225
300	30	58	86	166	266	385	480
400	38	90	144	275	458	666	872
500	54	138	212	399	793	1140	1689
600	84	191	290	621	1135	1942	3116
700	98	257	379	746	1637	3332	4688
800	126	309	494	974	2681	5098	7338
900	155	373	610	1352	3603	7212	9745
1000	196	449	737	1721	5022	9043	12120

Tabela 3: Algorytm Belmanna - Forda - reprezentacja macierzowa

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
100	692	895	1247	2398	3774	2565	848
200	4697	7088	10342	20700	32329	22706	6307
300	16019	24150	35260	70497	109736	77375	21050
400	37731	57512	83713	168390	259705	183317	49197
500	73270	111769	163485	328117	506958	358088	95123
600	125684	192770	280211	683751	876165	620030	163478
700	198746	306219	448778	959943	1391727	984504	257913
800	296076	455419	675319	1350089	2080354	1469096	383910
900	421952	651922	955589	1925698	2966667	2087400	546150
1000	576758	892651	1309577	2646439	4073451	2890666	735073

Tabela 4: Algorytm Belmanna - Forda - reprezentacja listowa

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
100	28	97	249	626	1519	2719	3562
200	233	1027	2169	7210	14640	21312	27591
300	842	3792	9791	24634	49335	72141	94816
400	1940	11563	23197	59235	115193	177460	243025
500	3752	23226	45166	114646	229160	388686	617030
600	6560	39975	78963	233152	429856	881604	1760805
700	10988	62881	125994	364676	859912	1883541	3301486
800	17664	93990	187109	504457	1643699	3283167	5287886
900	27368	134843	268731	816380	2991759	4884029	7845663
1000	36768	184394	369010	1288401	4687861	8155320	9655778

Tabela 5: Algorytm Prima - reprezentacja macierzowa grafu

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
100	11	16	23	40	58	51	34
200	33	56	81	148	220	190	124
300	75	124	175	323	478	409	261
400	125	202	295	561	826	743	459
500	181	299	450	849	1260	1101	700
600	257	425	632	1190	1834	1578	980
700	339	572	830	1620	2449	2113	1316
800	494	736	1089	2096	3177	2778	1693
900	581	970	1345	2633	4014	3457	2158
1000	667	1135	1642	3266	4936	4259	2635

Tabela 6: Algorytm Prima - reprezentacja listowa grafu

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
100	4	7	10	21	35	49	59
200	10	25	42	82	132	185	215
300	25	57	96	170	277	367	448
400	41	96	148	279	474	657	961
500	60	142	226	404	709	1100	1595
600	83	195	299	559	1175	1956	2810
700	106	260	382	757	1805	3222	4724
800	136	322	491	989	2514	4825	7409
900	171	387	594	1277	3857	7460	9704
1000	199	448	724	1935	5179	9175	12012

Tabela 7: Algorytm Kruskala - reprezentacja macierzowa grafu

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
100	22	32	56	125	257	360	487
200	63	135	248	555	1177	1763	2408
300	290	457	705	1509	3034	4730	6695
400	478	804	1279	2890	6170	9839	13429
500	735	1260	2028	4955	10982	17020	23311
600	1073	1921	3158	7941	17191	26520	36167
700	1453	2682	4541	11457	24833	38316	52038
800	1923	3589	6188	15762	34055	52139	70675
900	2442	4686	8235	20946	43955	69483	93003
1000	3081	5922	10534	26857	56362	88511	123219

Tabela 8: Algorytm Kruskala - reprezentacja listowa grafu

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
100	14	24	46	115	240	366	501
200	41	111	219	532	1122	1764	2468
300	238	399	644	1444	2943	4831	6961
400	395	707	1165	2756	6111	10111	14336
500	606	1112	1885	4851	11174	18166	25223
600	863	1714	2939	7728	17808	28870	39392
700	1169	2389	4266	11388	25755	40927	56312
800	1535	3210	5873	15942	35756	55845	76926
900	2016	4308	7875	21280	46124	73549	100724
1000	2520	5383	10110	27590	59565	94541	132443

Tabela 9: Algorytm Forda - Fulkersona z przeszukiwaniem grafu w głąb - reprezentacja macierzowa

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
10	111	2	2	5	14	54	104
20	3	4	7	77	290	546	1554
30	5	8	41	157	1343	3217	7023
40	9	24	43	473	3411	8493	21552
50	18	68	283	1738	8442	20510	47766
60	17	167	421	2899	17406	43543	96974
70	24	228	689	4752	28737	74323	176206
80	35	321	1402	8914	39187	124204	293426
90	38	693	2139	15497	76671	198959	469250
100	84	749	3705	17892	121147	316599	748549

Tabela 10: Algorytm Forda - Fulkersona z przeszukiwaniem grafu w głąb - reprezentacja listowa

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
10	181	2	3	11	24	84	167
20	4	6	11	86	402	942	3222
30	5	12	49	195	2205	7149	20815
40	6	32	52	604	6392	24043	75353
50	14	58	261	2308	19180	69972	230758
60	13	119	360	4723	43833	190558	652086
70	17	150	586	7265	84638	417956	1600675
80	33	312	1245	15409	142198	866680	3166246
90	24	420	1945	28826	333386	1662949	5681976
100	40	449	3495	35596	599888	3003410	9662920

Tabela 11: Algorytm Forda - Fulkersona z przeszukiwaniem grafu wszerz - reprezentacja macierzowa

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
10	110	2	2	4	17	51	108
20	3	5	7	61	298	587	1662
30	5	10	40	170	1300	3350	6945
40	11	20	45	476	3213	8491	21138
50	13	65	300	1664	8266	20363	49546
60	16	163	419	2764	17242	44645	99276
70	21	156	705	4697	29423	74499	180148
80	26	281	1348	8434	39527	122968	296811
90	36	539	2092	15132	78312	200296	472394
100	42	622	3681	17390	119588	322582	738847

Tabela 12: Algorytm Forda - Fulkersona z przeszukiwaniem grafu wszerz - reprezentacja listowa

V \Gęstość	1%	5%	10%	25%	50%	75%	99%
10	174	3	3	9	32	75	163
20	3	7	12	74	397	966	3200
30	5	14	49	204	2105	7214	20506
40	7	24	51	601	6191	24137	74972
50	9	56	258	2181	18364	69198	229647
60	12	120	350	4246	42524	193485	654536
70	15	124	551	7139	85518	421783	1612465
80	19	250	1243	15090	140067	874273	3182618
90	23	372	1905	28383	335404	1655296	5677764
100	26	414	3516	35462	595229	2996900	9646097