Struktury Danych i Złożoność obliczeniowa - projekt 2

Mikołaj Chmielecki

7 czerwca 2021 r.

Autor:	Mikołaj Chmielecki
Nr albumu:	252723
Prowadzący:	mgr inż. Antoni Sterna
Temat:	Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera.
Termin zajęć:	poniedziałek TP 11:15
Termin oddania:	7 czerwca 2021 r.

Spis treści

1 Wstęp

Zadaniem projektowym była implementacja oraz dokonanie pomiaru czasu działania wybranych algorytmów grafowych rozwiązujących następujące problemy:

- wyznaczenie minimalnego drzewa rozpinającego - algorytm Kruskala oraz algorytm Prima,
- wyznaczenie najkrótszej ścieżki w grafie - algorytm Dijkstry oraz algorytm Bellmana-Froda,
- wyznaczenie maksymalnego przepływu - algorytm Forda-Fulkersona w dwóch wariantach - przeszukiwanie wszerz (algorytm Edmondsa-Karpa) oraz przeszukiwanie w głąb.

2 Implementacja

Implementacja projektu spełnia następujące założenia:

- do implementacji został użyty język programowania C++ oraz środowisko IDE Visual Studio 2019,
- program został wyeksportowany do pliku wykonywalnego SDiZO_2.exe zwanego dalej programem,
- grafy zapisywane są w pamięci komputera na dwa sposoby - macierz sąsiedztwa oraz listy sąsiedztwa,
- każdy algorytm to osobna klasa implementująca interfejs `IAlgorytm`, co ułatwia dokonanie pomiaru czasów trwania algorytmów,
- każdy problem posiada własne menu, w którym użytkownik może dokonać wyboru sposobu rozwiązania i typ reprezentacji grafu,
- istnieje możliwość wczytania grafu do programu z pliku tekstowego,
- program domyślnie sprawdza plik "dane.txt", który znajduje się w katalogu uruchomieniowym programu, jednak istnieje możliwość zmiany ścieżki pliku z danymi,
- liczby w pliku z danymi mogą być rozdzielone dowolną liczbą znaków białych,
- pierwsza linia wczytywanego pliku powinna zawierać następujące informacje: liczbę krawędzi, liczbę wierzchołków, numer wierzchołka startowego oraz numer wierzchołka końcowego,
- w kolejnych liniach powinny znajdować się krawędzie oznaczone numerem wierzchołka początkowego i końcowego oraz wagą/przepustowością,
- program jest odporny na wszelkiego rodzaju błędy danych wejściowych, o czym informuje stosownymi komunikatami,
- program posiada opcję włączenia testów, jednak trwa to łącznie kilka godzin i w tym czasie nie ma możliwości korzystania z pozostałych funkcji programu,
- program oddany do oceny został zbudowany w trybie Release i w takiej postaci wykonywano na nim eksperymenty z pomiarem czasu.

2.1 Reprezentacje grafów

Reprezentacje grafów implementują wspólny interfejs `IGraf`, dzięki czemu każdy algorytm wystarczyło zaimplementować raz dla obu reprezentacji. Ten interfejs umożliwia pobranie wszystkich krawędzi oraz pobranie wszystkich sąsiadów danego wierzchołka, co było najczęściej wykorzystywane w zaimplementowanych algorytmach.

2.1.1 Macierz sąsiedztwa

Macierz sąsiedztwa to tablica dwuwymiarowa o wymiarach VxV. Na przecięciu w danych komórkach przechowywana jest waga/przepustowość krawędzi. W rzędach zapisane są wierzchołki startowe. W kolumnach zapisane są wierzchołki końcowe. Tablica dwuwymiarowa jest zaimplementowana w postaci tablicy tablic dynamicznych z poprzedniego zadania projektowego.

2.1.2 Listy sąsiedztwa

Listy sąsiedztwa to struktury danych, przechowujące sąsiadów oraz wagi/przepustowości każdej krawędzi dla każdego wierzchołka. List jest tyle ile jest wierzchołków w grafie. Listy przechowywane są w tablicy dynamicznej. Każdy element listy przechowuje nr sąsiada oraz wagę/przepustowość krawędzi.

2.2 Tablica

Klasa `Tablica` to zmodyfikowana tablica dynamiczna z poprzedniego zadania projektowego. Została zaimplementowana w sposób generyczny po to, by mogła przechowywać różne typy danych. Dodatkowo posiada metodę sortowania, która jako argument przyjmuje wskaźnik na funkcję porównując elementy tablicy. Algorytm sortowania to algorytm sortowania szybkiego `quicksort`.

2.3 Lista

Klasa `Lista` to zmodyfikowana lista dwukierunkowa z poprzedniego zadania projektowego. Została zaimplementowana w sposób generyczny po to, by mogła przechowywać różne typy danych.

2.4 Kolejka priorytetowa

Kolejka priorytetowa została zaimplementowana w postaci kopca binarnego. Przechowuje obiekty klasy `Struktura`, które są wykorzystywane przez 3 algorytmy. Na początku zawsze znajduje się struktura o najmniejszej wartości pola `klucz`. Zakłada się, że w momencie utworzenia kolejki, jako parametr przekazany jest już utworzony kopiec. Rzeczywiście tak jest, ponieważ w danych algorytmach wszystkie wierzchołki posiadają klucze o wartości ∞, a jedynie wierzchołek startowy posiada klucz o wartości 0. Pozwala na naprawę kopca od zadanego wierzchołka w górę, gdyż w danych algorytmach klucze mogą tylko zmniejszać wartość. Przechowuje tablicę indeksów, która pozwala na natychmiastowy dostęp do zadanego wierzchołka znajdującego się w kopcu.

2.5 Algorytmy

Wszystkie algorytmy implementują interfejs `IAlgorytm`, który udostępnia takie funkcje jak:

- `uruchom()` - uruchamia działanie algorytmu,
- `getWynik()` - zwraca wynik dziania algorytmu w postaci tekstowej,
- `inicjalizuj()` - inicjalizuje algorytm dla danego grafu wejściowego,
- `zwolnij()` - zwalnia pamięć zarezerwowaną podczas pojedynczego uruchomienia algorytmu po to, by można było uruchomić algorytm ponownie dla innego grafu.

2.5.1 Algorytm Bellmana-Forda

Algorytm znajdujący najkrótszą ścieżkę w grafie od zadanego wierzchołka do wszystkich pozostałych wierzchołków. Dopuszcza krawędzie o ujemnych wagach. W momencie znalezienia cyklu ujemnego zwraca błąd do użytkownika.

2.5.2 Algorytm Dijkstry

Algorytm znajdujący najkrótszą ścieżkę w grafie od zadanego wierzchołka do wszystkich pozostałych wierzchołków. Nie dopuszcza krawędzi o ujemnych wagach. Brak tej ogólności sprawia, że ma lepszą złożoność czasową niż algorytm Bellmana-Forda. Korzysta z kolejki priorytetowej.

2.5.3 Algorytm Prima

Algorytm znajdujący minimalne drzewo rozpinające. Graf wejściowy traktuje jako nieskierowany. Z racji tego, że w trakcie swojego dziania pobiera wszystkich sąsiadów danego wierzchołka, to krawędzie w reprezentacji grafu są zdublowane, co ułatwia wyszukiwanie sąsiadów. Korzysta z kolejki priorytetowej.

2.5.4 Algorytm Kruskala

Algorytm znajdujący minimalne drzewo rozpinające. Graf wejściowy traktuje jako nieskierowany. Z racji tego, że w trakcie swojego dziania pobiera wszystkie krawędzie grafu, to krawędzie w reprezentacji grafu nie są zdublowane, co powoduje brak powielania tych samych krawędzi podczas ich zwracania. Korzysta z algorytmu szybkiego sortowania tablicy. Korzysta również z zaawansowanej implementacji zbiorów rozłącznych w postaci struktur, które nazywają się "lasami". Początkowo każdy wierzchołek należy do osobnego drzewa. W momencie dodawania krawędzi do MST, drzewa są ze sobą łączone. Jest to rozwiązanie efektywniejsze niż naiwna implementacja zbiorów rozłączonych poprzez tablicę i odpowiednie "kolorowanie" wierzchołków.

2.5.5 Algorytm Forda-Fulkersona

Algorytm znajdujący maksymalny przepływ w sieci przepływowej. Zakłada, że przepływy mają wartość większą niż 0. Korzysta z kolejki zaimplementowanej w postaci listy dwukierunkowej. Jest realizowany w dwóch wariantach. Pierwszym wariantem jest wariant z przeszukiwaniem w szerz (BFS) - jest to algorytm Edmondsa-Karpa. W tym przypadku kolejka jest typu FIFO. Drugim wariantem jest wariant z przeszukiwaniem w głąb (DFS). Jest on nieco gorszy, niż poprzedni wariant, gdyż znajdowane ścieżki rozszerzające są dłuższe. W tym przypadku kolejka jest typu LIFO (stos). Algorytm Forda-Fulkersona preferuje, krótkie ścieżki rozszerzające, co sprawia, że przeszukiwanie w głąb nie jest rozwiązaniem optymalnym, gdyż znajduje ono zazwyczaj ścieżki długie. Zaletą przeszukiwania w głąb jest fakt, iż potrzebuje ono mniejszą przestrzeń pamięciową do zapamiętywania wierzchołków pozostałych do odwiedzenia. Algorytm Forda-Fulkersona wprowadza do obu reprezentacji grafu dodatkową informację o aktualnym przepływie przez daną krawędź. Jest to niezbędne do obliczania przepustowości rezydualnej ścieżki.

2.6 Wczytywanie danych z pliku tekstowego

Program wczytuje dane z pliku tekstowego. Użytkownik może wyświetlić oraz zmienić ścieżkę wczytywanych danych w menu `Dane`. Domyślnie ścieżka ma wartość "dane.txt" i jest zapisana w postaci względnej, tzn. plik z danymi powinien znajdować się w folderze uruchomieniowym. Pierwsza linia wczytywanego pliku powinna zawierać następujące informacje: liczbę krawędzi, liczbę wierzchołków, numer wierzchołka startowego oraz numer wierzchołka końcowego. Dla problemu minimalnego drzewa rozpinającego krawędzie traktowane są jako nieskierowane oraz pomijane są numery wierzchołka startowego oraz końcowego. Dla problemu najkrótszej ścieżki krawędzie traktowane są jako skierowane i pomijany jest wierzchołek końcowy. Dla problemu maksymalnego przepływu krawędzie traktowane są jako skierowane oraz jest nałożone ograniczenie na przepustowości krawędzi - powinny być one większe od 0. W kolejnych liniach powinny znajdować się krawędzie oznaczone numerem wierzchołka początkowego i końcowego oraz wagą/przepustowością, Poszczególne wartości mogą być rozdzielone dowolną liczbą znaków białych. Bazowa część implementacji klasy odpowiadającej za wczytywanie danych oparta jest o materiał zamieszczony w sekcji "Literatura"[3]. Program jest odporny na wszystkie możliwości

wprowadzenia błędnych wartości, oprócz braku spójności grafu, co jest wykrywane dopiero w momencie uruchomienia danego algorytmu.

2.7 Pomiar czasu

Pomiar czasu wykorzystywany jest podczas przeprowadzania testów czasów operacji. Pomiar czasu ma dokładność rzędu mikrosekund i jest oparty na pobieraniu stanu licznika przed rozpoczęciem wykonywania operacji oraz po jej zakończeniu. Dzieląc otrzymany wynik przez częstotliwość tego licznika i mnożąc przez odpowiedni czynnik otrzymujemy czas w mikrosekundach[4]. Każdy algorytm implementuje interfejs `IAlgorytm`, który zawiera funkcję uruchamiającą działanie algorytmu. Dzięki temu dysponując zestawem algorytmów, możemy je testować za pośrednictwem tej samej funkcji. Pomiar czasu sprawdza czas działania algorytmów nie wliczając w to przygotowywanie wyników końcowych, które są przedstawiane użytkownikowi w postaci tekstowej.

2.8 Generowanie grafów losowych

Klasa `LosowyGraf` posiada możliwość generowania losowych grafów. Losowość oparta jest na generatorze licz pseudolosowych[2]. Sposób generowania grafów losowych dla każdego problemu nieco się różnił z uwagi na inne założenia dotyczące grafów poddawanych działaniom algorytmów. Dla problemu minimalnego drzewa rozpinającego, na początku wybieramy wierzchołek centralny i tworzymy krawędzie łączące go ze wszystkimi pozostałymi wierzchołkami, a następnie uzupełniamy graf do zadanej gęstości. Losowe wagi zawierają się w przedziale [-1000000,1000000]. Dla problemu najkrótszej ścieżki, tworzymy ścieżki o losowej długości od wierzchołka startowego do wszystkich pozostałych wierzchołków. Następnie uzupełniamy graf do zadanej gęstości. Dla algorytmu Bellmana-Forda losowe wagi przyjmują wartości [-1000000,1000000], a dla algorytmu Dijkstry [0,1000000]. Dla problemu maksymalnego przepływu tworzymy losową liczbę ścieżek o losowej długości od wierzchołka startowego do wierzchołka końcowego. Następnie uzupełniamy graf do zadanej gęstości. Przepustowości krawędzi przyjmują wartości [1,1000000]. Na czas testów zakładamy, że wierzchołkiem startowym jest wierzchołek o numerze 0, a wierzchołkiem końcowym jest wierzchołek o maksymalnym numerze. Podczas uzupełniania grafu do zadanej gęstości wybieramy losową krawędź, poprzez wylosowanie wierzchołka startowego i końcowego, a następnie sprawdzamy czy ta krawędź jest jeszcze pusta. W momencie kiedy nie jest pusta, to iterujemy po macierzy sąsiedztwa do momentu, aż znajdziemy krawędź pustą. Dzięki takiemu podejściu mamy pewność, że dla dużej gęstości grafu koniec końców znajdziemy krawędź pustą. Grafy losowe są generowane w postaci macierzy sąsiedztwa. Kiedy istnieje potrzeba wygenerowania grafu losowego w postaci list sąsiedztwa, to generujemy graf w postaci macierzy sąsiedztwa, a następnie konwertujemy macierz sąsiedztwa na listy sąsiedztwa.

3 Złożoność obliczeniowa

Złożoność obliczeniowa definiowana jest jako liczba zasobów potrzebnych do przeprowadzenia algorytmu. Wyróżniamy dwa rodzaje złożoności obliczeniowej: złożoność pamięciowa oraz czasowa. Przeprowadzając eksperymenty badamy złożoność czasową, zatem przedstawiam złożoności czasowe teoretyczne dla zaimplementowanych algorytmów:

Problem	Algorytm	Złożoność czasowa
Minimalne drzewo rozpinające	Algorytm Prima	$O(E \cdot \log V)$
	Algorytm Kruskala	$O(E \cdot \log E)$
Najkrótsza ścieżka	Algorytm Dijkstry	$O(E \cdot \log V)$
	Algorytm Bellmana-Forda	$O(E \cdot V)$
Maksymalny przepływ	Algorytm Edmondsa-Karpa	$O(V \cdot E^2)$
	Algorytm Forda-Fulkersona DFS	$O(E \cdot f_{max})$

Tablica 1: Teoretyczne złożoności czasowe[6]

4 Eksperymenty

Klasa `Testy` odpowiada za przeprowadzenie przeprowadzenie pomiarów czasów działania zaimplementowanych algorytmów dla różnych gęstości grafów oraz dwóch różnych reprezentacji grafów. Wyniki testów są zapisywane w pliku tekstowym w formacie ".csv". W pierwszej lini pliku tekstowego jest zapisany testowany algorytm, gęstość grafu oraz rodzaj reprezentacji. Jest to przydatne podczas automatycznego tworzenia wykresów i tabel. Testy były wykonywane dla 10 różnych reprezentatywnych rozmiarów grafów, co sprawia, że wyniki obrazują zależność złożoności czasowej od rozmiaru problemu. Dla problemu minimalnego drzewa rozpinającego oraz najkrótszej ścieżki w grafie przyjęto, że maksymalna liczba wierzchołków w grafie będzie wynosić 500. Ze względu na dużą złożoność czasową problemu maksymalnego przepływu przyjęto, że dla tego problemu losowe grafy będą posiadały maksymalnie 50 wierzchołków. Algorytmu były testowane na grafach o następujący gęstościach: 25%, 50%, 75%, 99%. Algorytmu były testowane dla reprezentacji macierzy sąsiedztwa oraz list sąsiedztwa.

4.1 Założenia techniczne

Po uruchomieniu testów program pracuje przez kilka godzin. Testy były przeprowadzane na laptopie o następujących parametrach:

- system operacyjny: Windows 10 Pro uruchomiony w trybie awaryjnym z dostępem do konsoli,
- procesor: Intel Core i5-4300M,
- częstotliwość taktowania: 2,6 GHz,
- pamięć RAM: 16 GB.

4.2 Generowanie wykresów i tablic

Wykonując testy otrzymałem 48 różnych zestawów danych. Generowanie wykresów dla takiej liczby różnych zestawów danych w standardowych narzędziach (np. Excel) okazałoby się dosyć problematyczne. Dlatego skorzystałem z języka programowania Python oraz dedykowanego narzędzia do pracy z danymi "MatPlotLib". Napisałem skrypt generujący w sposób automatyczny wykresy oraz tabele z danych zawartych w plikach z rozszerzeniem ".csv" zawartych w katalogu uruchomieniowym skryptu. Skrypt automatycznie dostosowuje jednostkę, tak aby żadna wartość nie przekraczała 1000 oraz przybliża czasy operacji do 3 cyfr znaczących. Najmniejszą akceptowalną jednostką jest mikrosekunda, a wynik nie jest konwertowany na nanosekundy, ponieważ pomiar czasu ma rozdzielczość mikrosekundową[4].

5 Wyniki eksperymentów

5.1 Problem minimalnego drzewa rozpinającego

Kruskal

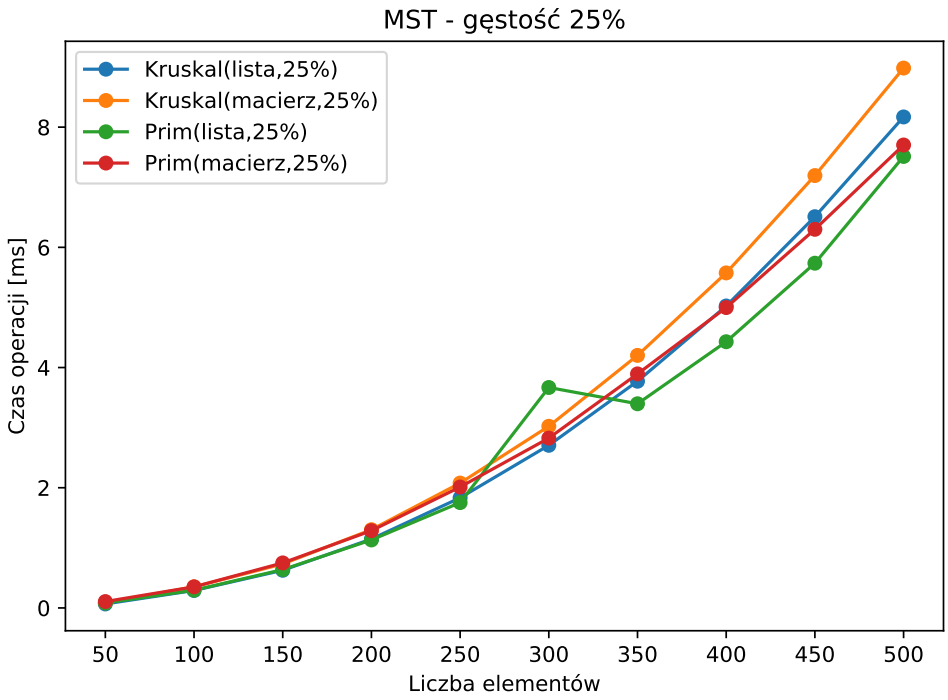
Liczba wierzchołków	(lista,25%) [ms]	(lista,50%) [ms]	(lista,75%) [ms]	(lista,99%) [ms]	(macierz,25%) [ms]	(macierz,50%) [ms]	(macierz,75%) [ms]	(macierz,99%) [ms]
50	0.0665	0.132	0.196	0.254	0.088	0.156	0.215	0.274
100	0.29	0.546	0.828	1.1	0.351	0.586	0.87	1.13
150	0.627	1.28	1.96	2.63	0.732	1.36	2.02	2.66
200	1.15	2.38	3.65	4.92	1.3	2.48	3.74	4.94
250	1.83	3.82	5.92	7.98	2.08	4.0	6.03	7.99
300	2.71	5.68	8.79	11.9	3.02	5.91	8.94	11.8
350	3.77	7.91	12.4	16.8	4.2	8.23	12.5	16.5
400	5.02	10.6	16.6	22.6	5.57	11.0	16.6	22.1
450	6.51	13.9	21.7	30.0	7.19	14.1	21.4	28.6
500	8.17	17.8	27.9	38.5	8.98	17.7	27.0	36.7

Tablica 2: Wyniki pomiarów czasu - Kruskal

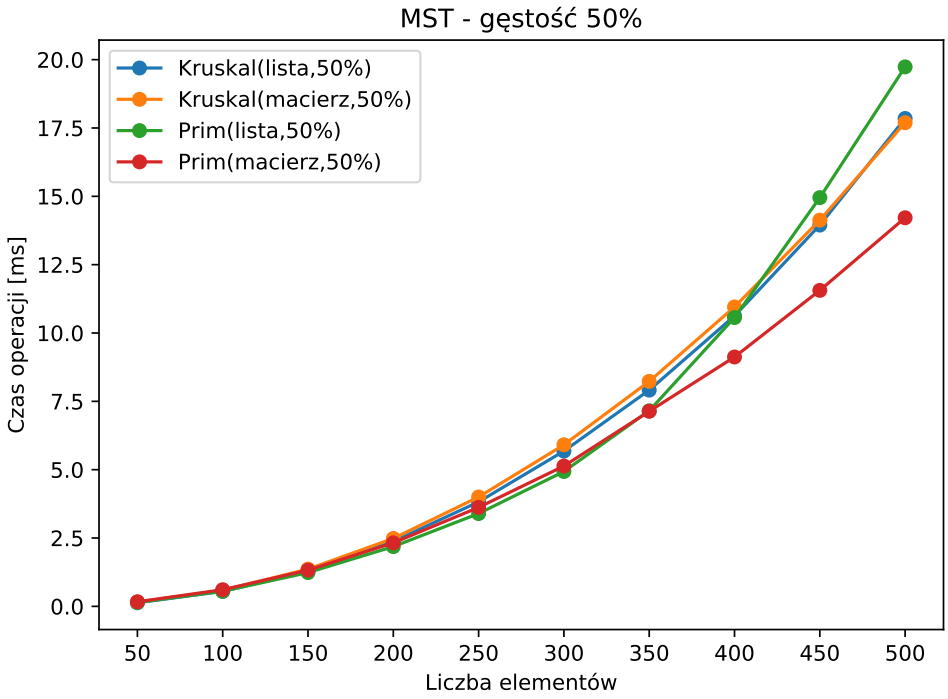
Prim

Liczba wierzchołków	(lista,25%) [ms]	(lista,50%) [ms]	(lista,75%) [ms]	(lista,99%) [ms]	(macierz,25%) [ms]	(macierz,50%) [ms]	(macierz,75%) [ms]	(macierz,99%) [ms]
50	0.0823	0.145	0.208	0.266	0.106	0.168	0.223	0.272
100	0.295	0.552	0.821	1.06	0.351	0.608	0.837	1.03
150	0.638	1.23	1.82	2.39	0.749	1.32	1.85	2.33
200	1.13	2.19	3.23	4.24	1.29	2.32	3.26	4.09
250	1.75	3.39	5.12	7.13	2.01	3.62	5.11	6.41
300	3.67	4.93	8.04	12.3	2.83	5.13	7.24	9.1
350	3.4	7.15	12.8	18.9	3.89	7.14	9.97	12.5
400	4.43	10.6	18.5	26.6	5.0	9.12	12.8	16.6
450	5.74	15.0	25.1	34.8	6.3	11.6	16.3	20.7
500	7.51	19.7	32.3	44.1	7.7	14.2	20.1	25.5

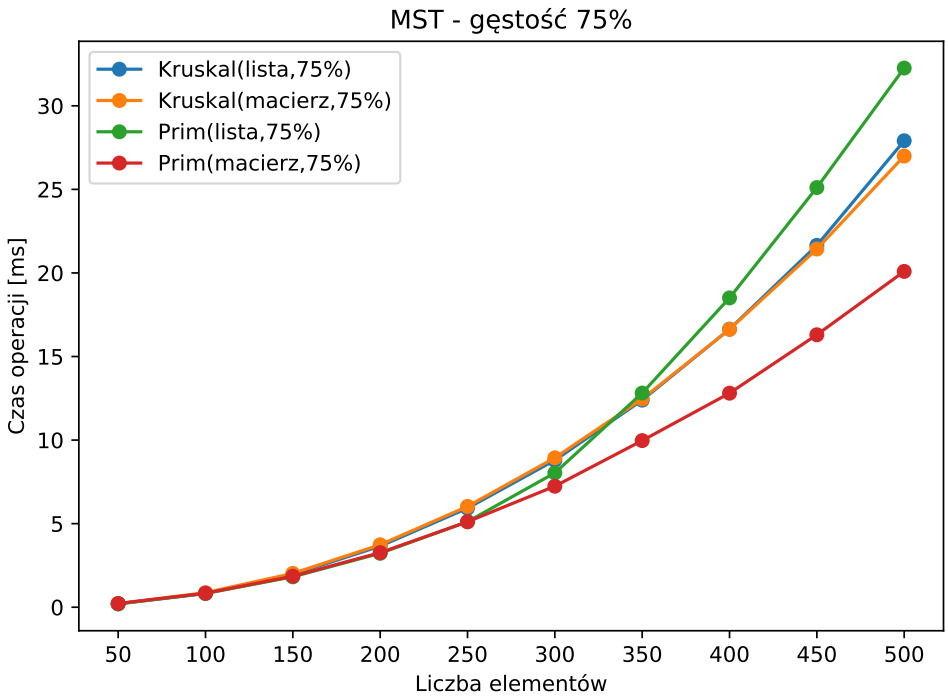
Tablica 3: Wyniki pomiarów czasu - Prim



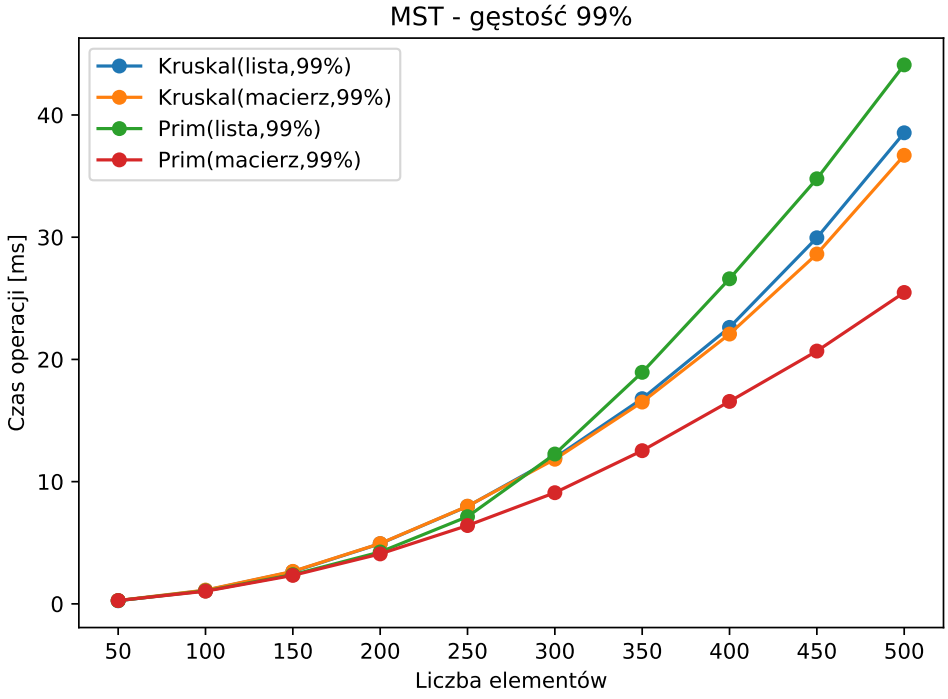
Rysunek 1: Wykres dla gęstości 25%



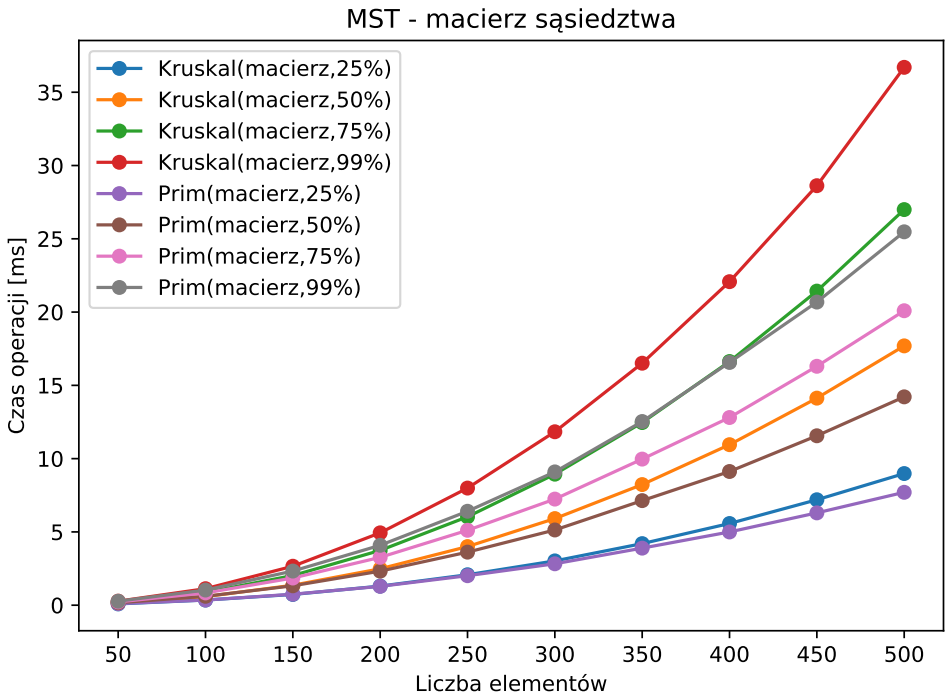
Rysunek 2: Wykres dla gęstości 50%



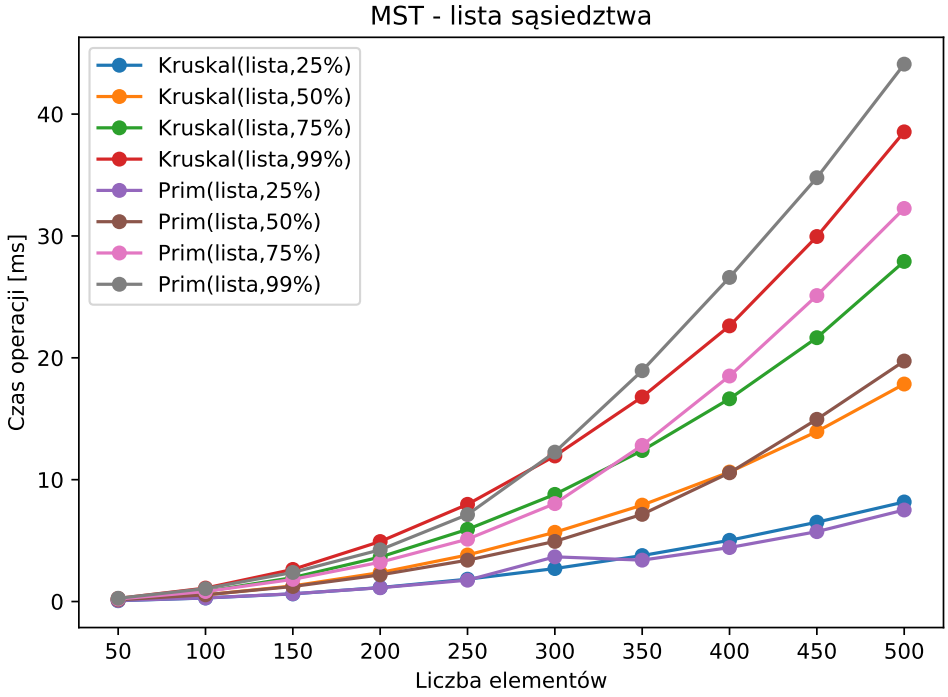
Rysunek 3: Wykres dla gęstości 75%



Rysunek 4: Wykres dla gęstości 99%



Rysunek 5: Wykres dla macierzy sąsiedztwa



Rysunek 6: Wykres dla list sąsiedztwa

5.1.1 Wnioski

Można zauważyć, że dla dużych gęstości mniejsze czasy operacji występują dla reprezentacji macierzy sąsiedztwa, zarówno dla algorytmu Prima, jak i dla algorytmu Kruskala. Dla małych gęstości różnice w czasie działania pomiędzy tymi dwoma algorytmami są słabozauważalne. Na wykresach, dla macierzy sąsiedztwa oraz list sąsiedztwa można zauważyć, że dla grafów o większych gęstościach czasy operacji są większe, co wynika wprost z teoretycznych złożoności czasowych. Z tabeli teoretycznych złożoności czasowych 1 wynika, że złożoność algorytmu Kruskala jest nieco większa niż algorytmu Prima. Jest to jednak słabozauważalne i to tylko na niektórych wykresach. Prawdopodobnie wynika to z faktu, że liczby wierzchołków testowanych grafów są dosyć niskie, a złożoność czasowa prawidłowo określa czas wykonywania algorytmu dla rozmiaru problemu dążącego do nieskończoności. Ponadto ta różnica w teoretycznej złożoności czasowej wynika z innego parametru logarytmowanego. Funkcja logarytm jest funkcją bardzo wolno rosnącą co dla małych liczb wierzchołków i krawędzi słabo różnicuje te dwa algorytmy.

5.2 Problem najkrótszej ścieżki

Dijkstra

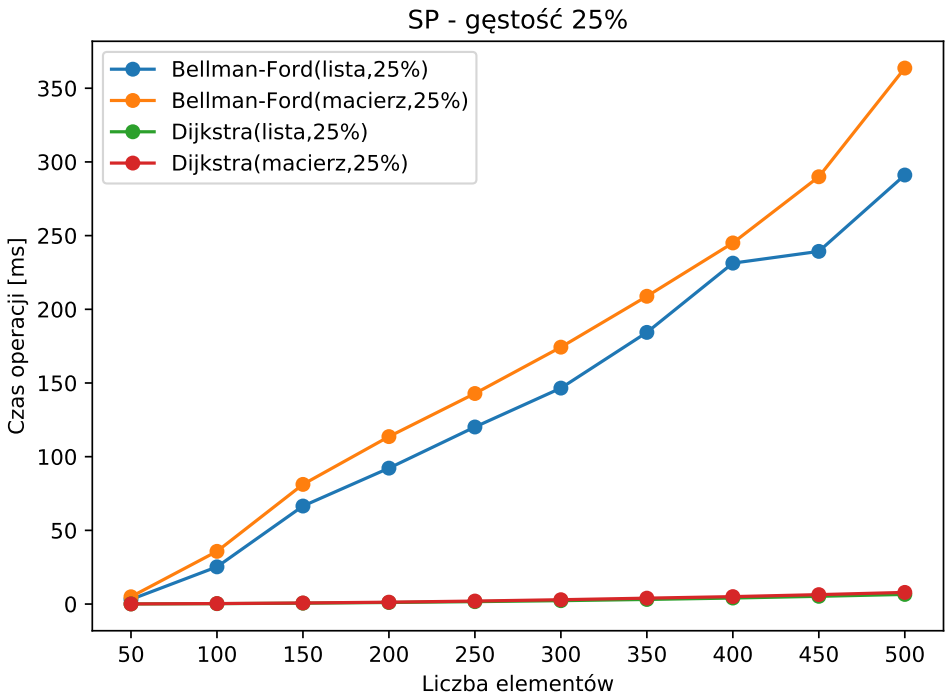
Liczba wierzchołków	(lista,25%) [ms]	(lista,50%) [ms]	(lista,75%) [ms]	(lista,99%) [ms]	(macierz,25%) [ms]	(macierz,50%) [ms]	(macierz,75%) [ms]	(macierz,99%) [ms]
50	0.0755	0.136	0.198	0.253	0.0897	0.156	0.222	0.252
100	0.272	0.515	0.759	0.994	0.335	0.608	0.844	0.995
150	0.595	1.14	1.69	2.22	0.73	1.32	1.9	2.21
200	1.05	2.02	2.99	3.93	1.28	2.34	3.32	3.98
250	1.63	3.13	4.67	6.22	2.03	3.69	5.12	6.12
300	2.3	4.52	6.86	9.11	2.94	5.22	7.43	9.08
350	3.12	6.24	9.41	12.4	4.0	7.07	10.0	12.3
400	4.09	8.24	12.3	16.2	5.05	9.28	13.2	16.0
450	5.25	10.5	15.6	20.5	6.39	11.7	16.5	20.3
500	6.54	12.9	19.3	25.2	7.9	14.5	20.6	25.1

Tablica 4: Wyniki pomiarów czasu - Dijkstra

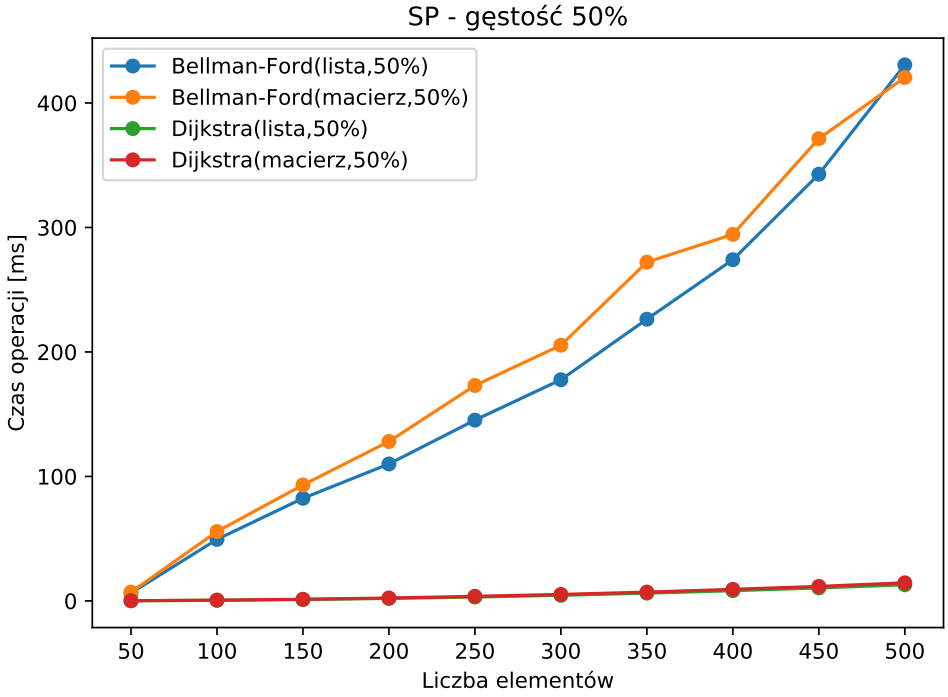
Bellman-Ford

Liczba wierzchołków	(lista,25%) [ms]	(lista,50%) [ms]	(lista,75%) [ms]	(lista,99%) [ms]	(macierz,25%) [ms]	(macierz,50%) [ms]	(macierz,75%) [ms]	(macierz,99%) [ms]
50	3.13	6.37	9.38	12.3	4.95	7.08	9.84	12.2
100	25.2	49.4	61.6	72.6	35.8	55.7	62.8	72.1
150	66.5	82.4	97.4	115.0	81.2	93.1	100.0	116.0
200	92.2	110.0	135.0	165.0	114.0	128.0	150.0	165.0
250	120.0	145.0	187.0	217.0	143.0	173.0	188.0	216.0
300	147.0	178.0	248.0	284.0	174.0	205.0	240.0	273.0
350	184.0	226.0	295.0	393.0	209.0	272.0	284.0	325.0
400	231.0	274.0	343.0	431.0	245.0	294.0	348.0	420.0
450	239.0	343.0	409.0	612.0	290.0	371.0	406.0	458.0
500	291.0	431.0	486.0	598.0	364.0	421.0	479.0	565.0

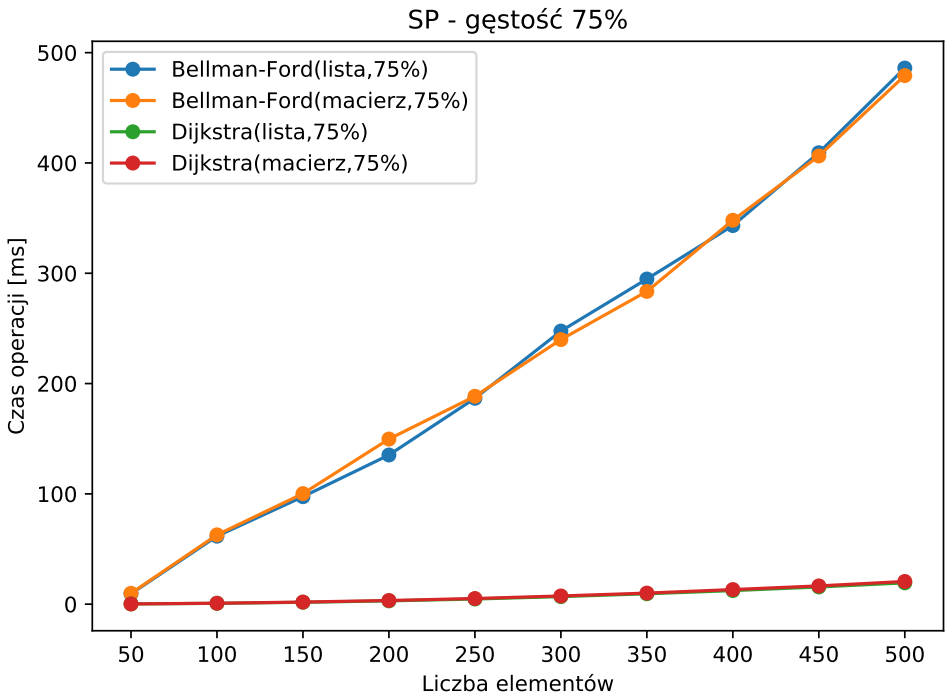
Tablica 5: Wyniki pomiarów czasu - Bellman-Ford



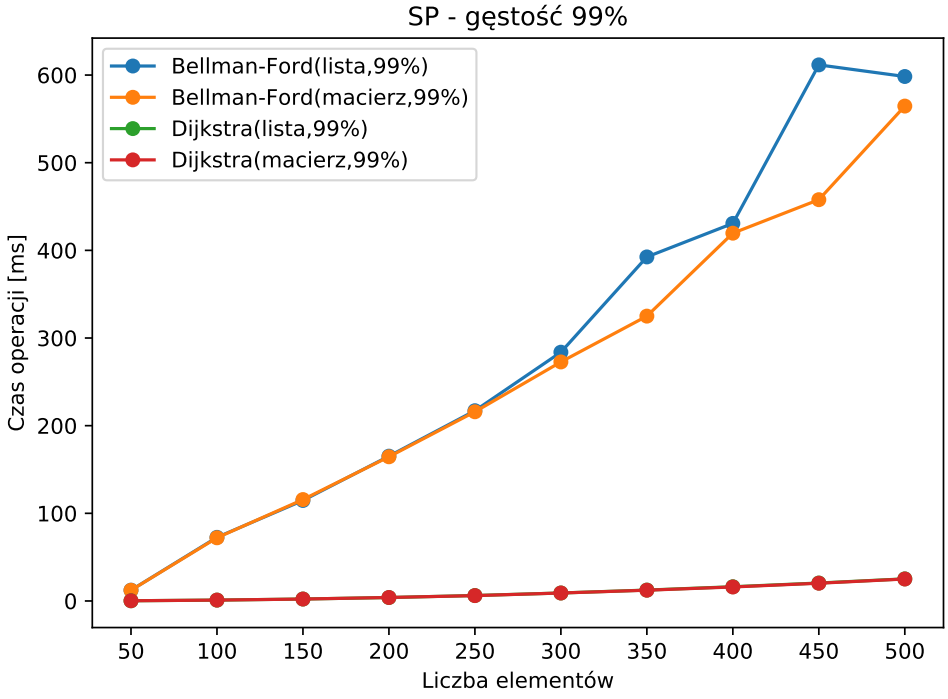
Rysunek 7: Wykres dla gęstości 25%



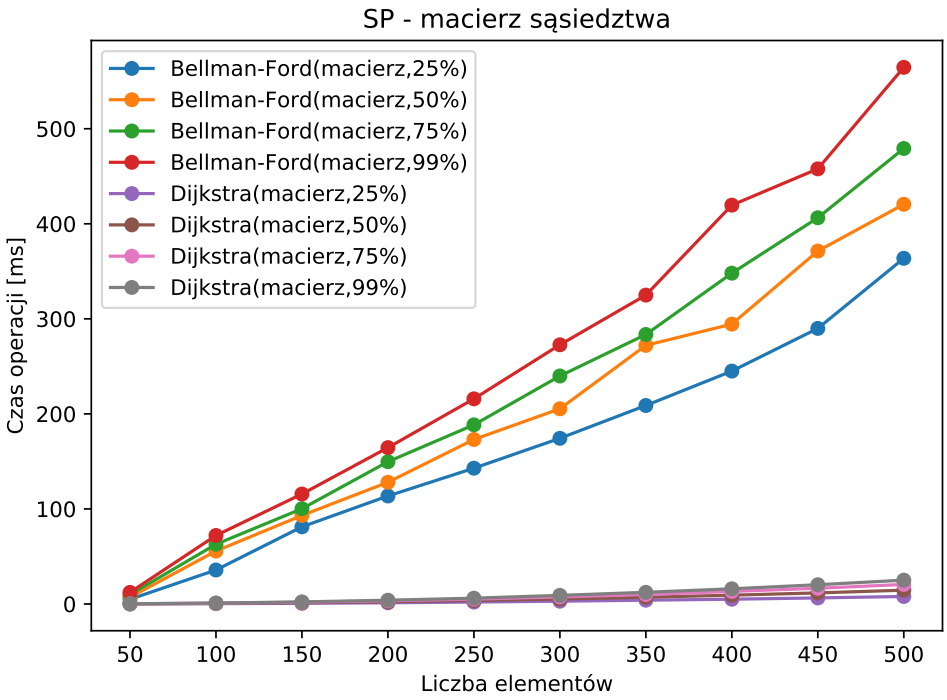
Rysunek 8: Wykres dla gęstości 50%



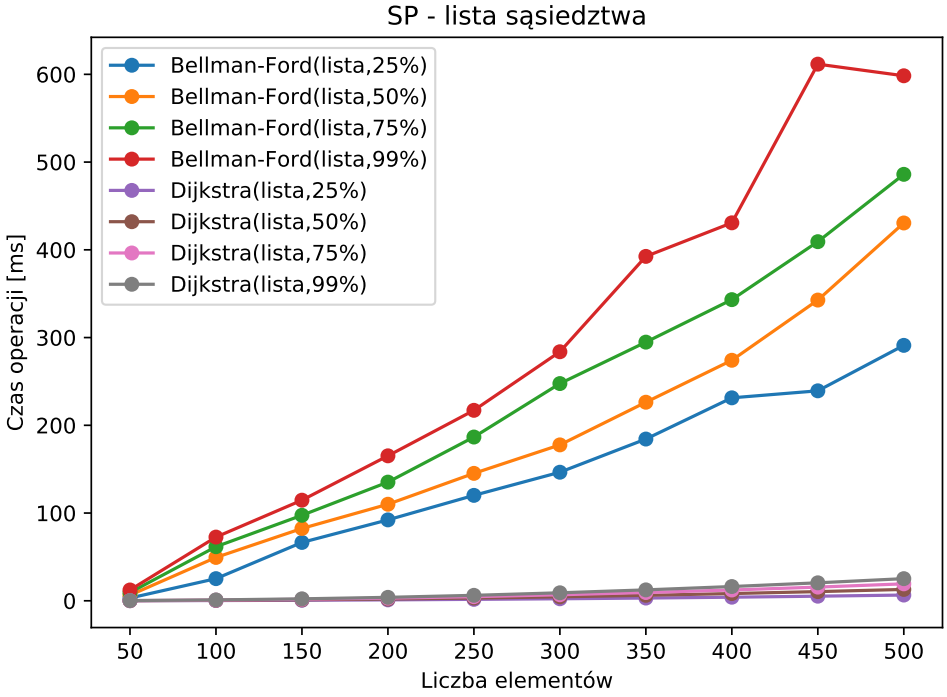
Rysunek 9: Wykres dla gęstości 75%



Rysunek 10: Wykres dla gęstości 99%



Rysunek 11: Wykres dla macierzy sąsiedztwa



Rysunek 12: Wykres dla list sąsiedztwa

5.2.1 Wnioski

Z uwagi na ogólność algorytmu Bellmana-Forda, dopuszczającą ujemne wagi krawędzi, rośnie złożoność czasowa tego algorytmu. Jest to zauważalne na wszystkich wykresach. Co więcej, dominacja algorytmu Dijkstry wynika z faktu, iż z wykresów możemy odczytać średnią złożoność czasową. Natomiast dla algorytmu Bellmana-Forda bardzo często był realizowany pesymistyczny przypadek. Z uwagi na to, że dla algorytmu Bellmana-Forda wagi krawędzi były losowane z przedziału [-1000000,1000000] zgodnie z rozkładem równomiernym, to bardzo często występowały cykle ujemne. Algorytm Bellmana-Forda kiedy graf nie posiada cyklu ujemnego to pętlę główną kończy po mniejszej liczbie iteracji. Natomiast dla grafu, który posiada cykl ujemny, pętla główna wykona się tyle razy, ile jest wierzchołków w grafie, bez wcześniejszego przerwania. Algorytm Bellmana-Forda testowałem również dla grafów o wagach nieujemnych i wtedy różnica również występowała, jednak nie była ona, aż tak znacząca.

5.3 Problem maksymalnego przepływu

Ford-Fulkerson-BFS

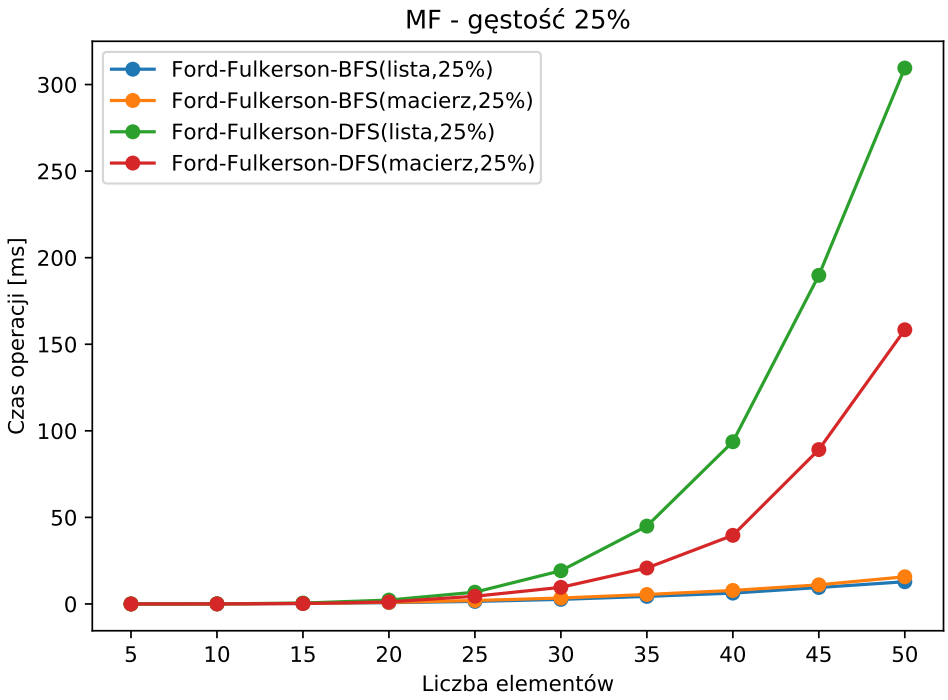
Liczba wierzchołków	(lista,25%) [ms]	(lista,50%) [ms]	(lista,75%) [ms]	(lista,99%) [ms]	(macierz,25%) [ms]	(macierz,50%) [ms]	(macierz,75%) [ms]	(macierz,99%) [ms]
5	0.00711	0.0192	0.0202	0.018	0.01	0.0135	0.0189	0.0207
10	0.0606	0.168	0.272	0.291	0.0661	0.179	0.283	0.31
15	0.283	0.599	0.828	0.968	0.33	0.789	1.1	1.22
20	0.756	1.49	2.03	2.33	0.897	1.92	2.56	2.84
25	1.46	2.95	3.93	4.5	1.99	3.69	4.79	5.44
30	2.64	5.03	7.07	8.01	3.42	6.26	8.37	9.34
35	4.37	8.31	10.9	12.7	5.47	9.43	13.0	14.1
40	6.26	12.2	17.2	19.0	7.81	14.3	19.3	21.6
45	9.5	17.5	24.1	28.2	11.0	20.3	26.4	30.3
50	12.8	24.2	33.6	38.3	15.7	27.7	36.8	41.8

Tablica 6: Wyniki pomiarów czasu - Ford-Fulkerson-BFS

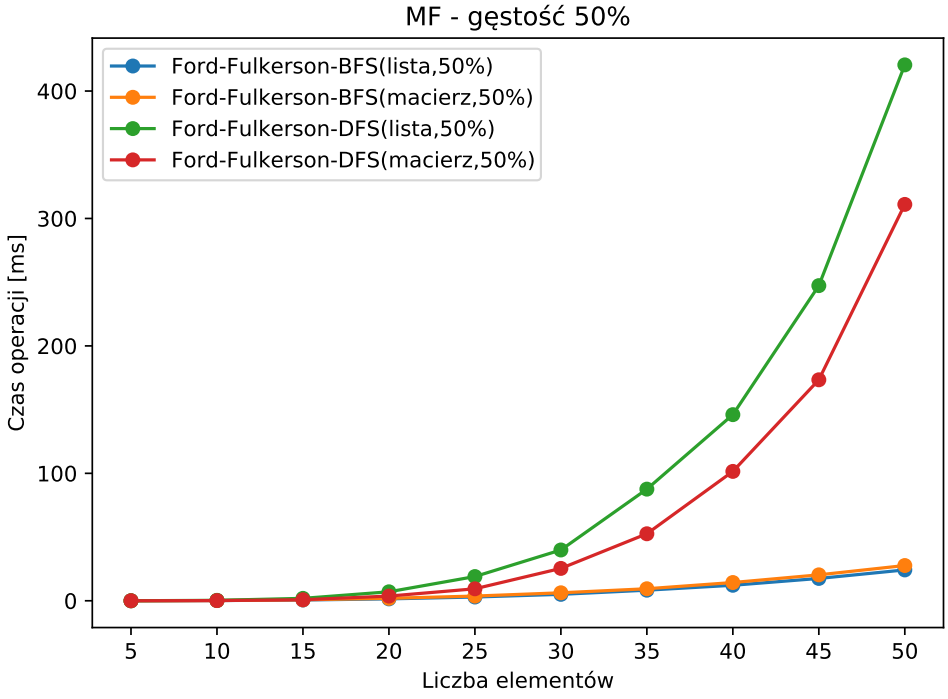
Ford-Fulkerson-DFS

Liczba wierzchołków	(lista,25%) [ms]	(lista,50%) [ms]	(lista,75%) [ms]	(lista,99%) [ms]	(macierz,25%) [ms]	(macierz,50%) [ms]	(macierz,75%) [ms]	(macierz,99%) [ms]
5	0.00688	0.0267	0.0309	0.0234	0.00692	0.0107	0.0185	0.0162
10	0.0584	0.337	0.8	0.808	0.0374	0.147	0.256	0.223
15	0.502	1.9	3.6	3.54	0.254	0.811	1.7	1.31
20	2.25	7.04	10.7	11.5	1.18	3.68	5.64	3.99
25	6.8	19.0	36.1	26.2	4.44	9.39	14.1	10.7
30	19.2	39.9	59.1	56.6	9.6	25.4	33.3	23.4
35	45.0	87.6	120.0	118.0	20.8	52.6	57.6	38.7
40	93.7	146.0	175.0	199.0	39.6	102.0	101.0	81.1
45	190.0	247.0	300.0	351.0	89.2	173.0	195.0	117.0
50	310.0	421.0	506.0	549.0	158.0	311.0	359.0	198.0

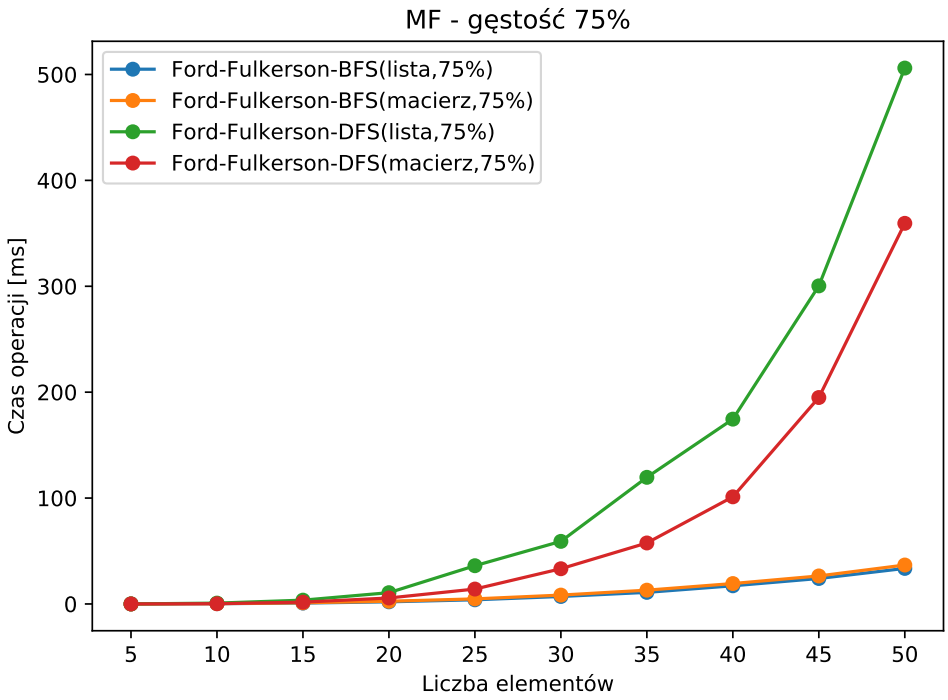
Tablica 7: Wyniki pomiarów czasu - Ford-Fulkerson-DFS



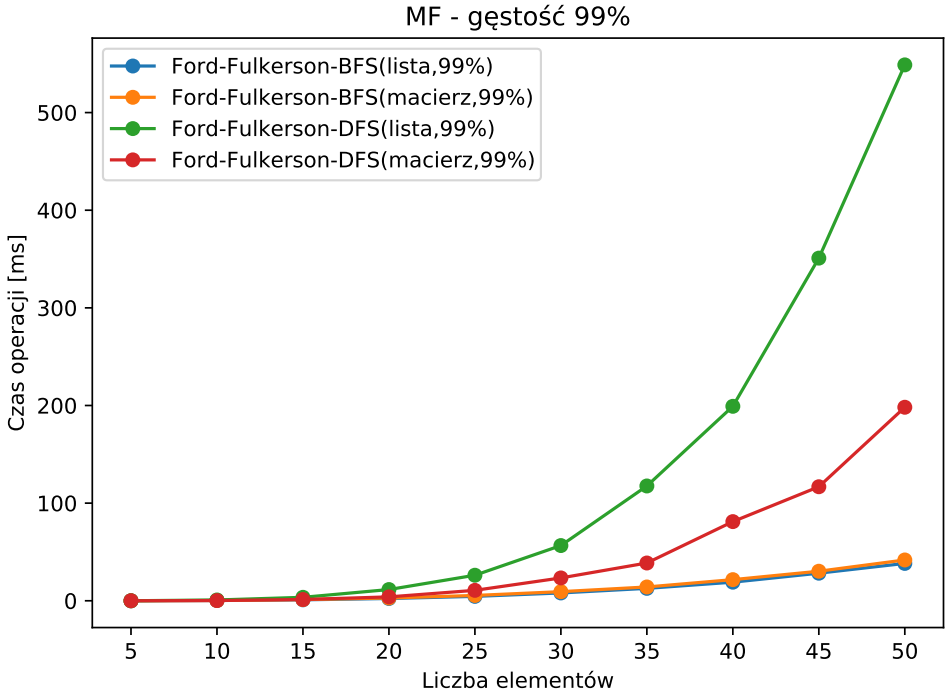
Rysunek 13: Wykres dla gęstości 25%



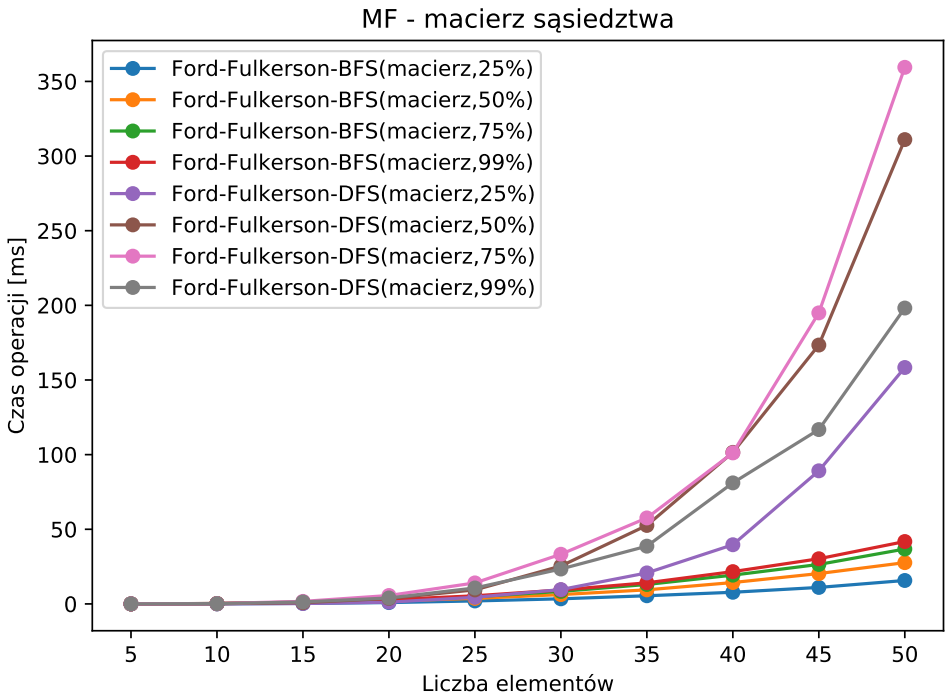
Rysunek 14: Wykres dla gęstości 50%



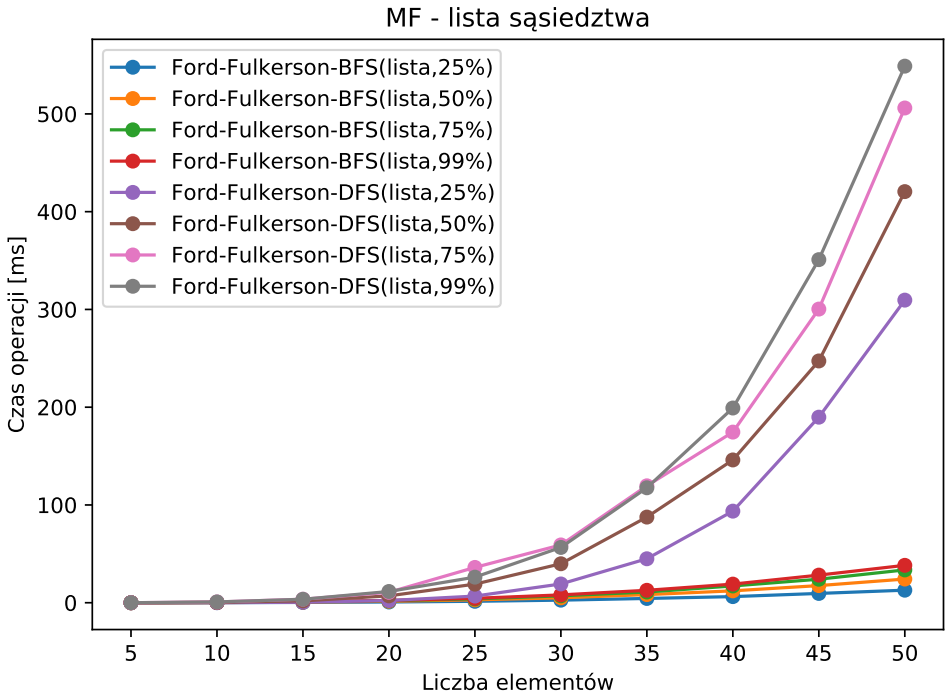
Rysunek 15: Wykres dla gęstości 75%



Rysunek 16: Wykres dla gęstości 99%



Rysunek 17: Wykres dla macierzy sąsiedztwa



Rysunek 18: Wykres dla list sąsiedztwa

5.3.1 Wnioski

Z wykresów czasowych odczytujemy, że znacznie efektywniejszym sposobem przeszukiwania jest przeszukiwanie wszerz. Jest to spowodowane tym, iż algorytm przeszukiwania wszerz znajduje najkrótsze ścieżki rozszerzające, co znacznie zmniejsza złożoność algorytmu. Złożoność algorytmu Forda-Fulkersona, przy wyszukiwaniu ścieżek w głąb, zmusiła mnie do zmniejszenia liczby wierzchołków w generowanych grafach losowych, by moć dokonać testów w realnym czasie. Jest to spowodowane tym, iż ścieżki rozszerzające wyszukiwane przez algorytm przeszukiwania w głąb, mogą na siebie bardzo często nachodzić i odwracać przepustowość rezydualną krawędzi. Przy przeszukiwaniu w głąb algorytm wyszukiwania ścieżki rozszerzającej może się wykonać maksymalnie tyle razy, ile wynosi maksymalny przepływ (przy każdej poprawie przepływ może zostać zwiększony minimalnie o wartość 1). Natomiast przy przeszukiwaniu wszerz mamy pewność, że algorytm wyszukiwania ścieżki rozszerzającej wykona się maksymalnie $V \cdot E$ razy. Stąd wynikają znacznie mniejsze czasy wykonywania algorytmu dla wariantu z przeszukiwaniem wszerz. Jedyną zaletą przeszukiwania w głąb jest mniejsza złożoność pamięciowa. Na powyższych wykresach bardzo zauważalna jest również dominacja reprezentacji macierzowej nad reprezentacją listową. Macierz sąsiedztwa zapewnia natychmiastowy dostęp do danej krawędzi, w momencie gdy znamy wierzchołek początkowy i końcowy.

6 Podsumowanie

Z powyższych rozważań wynika, że mając do rozwiązania jeden z przedstawionych problemów, należy dobrać odpowiedni algorytm oraz odpowiednią reprezentację grafu w pamięci komputera. Dla małych gęstości lepszym rozwiązaniem są listy sąsiedztwa, natomiast dla grafów gęstych lepszym rozwiązaniem są macierze sąsiedztwa. Macierze sąsiedztwa zapewniają natychmiastowy dostęp do krawędzi mając dany wierzchołek startowy i końcowy. Wprowadzają one jednak dosyć dużą nadmiarowość danych dla grafów rzadkich, gdyż nawet krawędzie puste, będą miały zarezerwowane miejsce w macierzy sąsiedztwa. Co więcej, warto zauważyć, że problemy minimalnego drzewa rozpinającego oraz najkrótszej ścieżki są znacznie mniej złożone niż problem maksymalnego przepływu. Można to było zauważyć podczas wykonywania testów. Byłem zmuszony zmniejszyć liczbę wierzchołków w generowanych grafach losowych dla problemu maksymalnego przepływu, aby móc wykonać testy w realnym czasie.
Należy zwrócić uwagę na kilka kwestii związanych z implementacją powyższego projektu. Klasy `Tablica` oraz `Lista` zostały zaimplementowane w sposób generyczny, tzn. mogą przechowywać dowolny typ danych. Ponadto klasa `Tablica` umożliwia sortowanie danych z wykorzystaniem efektywnego algorytmu sortowania szybkiego. Było ono wykorzystywane w wielu miejscach projektu, dzięki możliwości przechowywania różnych typów danych w tablicy. Sortowanie było wykorzystywane w algorytmie Kruskala oraz podczas generowania końcowego wyniku dla algorytmów Forda-Fulkersona, Prima oraz Kruskala (krawędzie ujęte w wyniku końcowym są posortowane względem numeru wierzchołka początkowego oraz końcowego). Kolejnym ciekawym punktem projektu jest kolejka priorytetowa zaimplementowana w postaci kopca, z możliwością natychmiastowego dostępu do danego wierzchołka zawartego w kopcu. Co więcej dwie różne reprezentacje grafów implementują wspólny interfejs `IGraf`, dzięki czemu każdy algorytm mógł zostać zaimplementowany jednokrotnie, przyjmując jako parametr interfejs do danej reprezentacji grafu.

Literatura

[1] mgr Jerzy Wałaszek, *Algorytmy i struktury danych.* Materiały dostępne na stronie internetowej `https://eduinf.waw.pl/inf/alg/001_search/index.php`

[2] mgr inż. Antoni Sterna, *Generowanie liczb pseudolosowych.* `http://staff.iiar.pwr.wroc.pl/antoni.sterna/sdizo/SDiZO_random.pdf`

[3] mgr inż. Antoni Sterna, *Odczyt danych z pliku.* `http://staff.iiar.pwr.wroc.pl/antoni.sterna/sdizo/SDiZO_file.pdf`

[4] mgr inż. Antoni Sterna, *Pomiar czasu.* `http://staff.iiar.pwr.wroc.pl/antoni.sterna/sdizo/SDiZO_time.pdf`

[5] dr inż. Jarosław Mierzwa, *Struktury danych i złożoność obliczeniowa.* Materiały dostępne na stronie internetowej `http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/`

[6] Thomas Cormen, *Wprowadzenie do algorytmów.* Wydawnictwo Naukowo-Techniczne Warszawa