Sebastian Łągiewski nr.indeksu 226173

Struktury Danych i Złożoność Obliczeniowa

Zadanie projektowe nr 2

Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera.

1. Wstęp

Reprezentacje grafu w pamięci komputera:

- **Macierz sąsiedztwa**

  Tablica dwuwymiarowa(V*V), której elementy pod indeksami [i][j] to wagi krawędzi i-j. W przypadku jej braku, wartość pod indeksami jest równa max(int) (∞) . Przejrzenie sąsiadów wierzchołka V ma złożoność czasową O[V].

- **Listy sąsiedztw**

  Tablica składająca się z list sąsiedztw. Pod danym indeksem [i] tablicy znajdują się struktury krawędzi wychodzących z wierzchołka umieszczone w liście jednokierunkowej z biblioteki STL. Każda struktura ma w sobie sąsiada wierzchołka oraz wagę krawędzi znajdującej się między nimi. Złożoność czasowa przejrzenia sąsiadów wierzchołka i wynosi (pesymistycznie) O(V), gdzie V to ilość wierzchołków grafu.

Algorytmy:

- **Algorytm Prima – Wyznaczanie minimalnego drzewa rozpinającego**

  Największy wpływ na złożoność czasową algorytmu mają operacje na kolejce priorytetowej z biblioteki STL: dodawanie krawędzi nieodwiedzonych oraz pobieranie krawędzi o najmniejszej wadze. W związku z tym złożoność algorytmu będzie wynosiła O(E * logV), gdzie E to ilość krawędzi w grafie.

- **Algorytm Dijkstry – Wyznaczanie najkrótszej ścieżki w grafie.**

  Implementując kolejkę priorytetową jako tablicę odległości, każda operacja pobrania z niej krawędzi o najmniejszej wadze ma złożoność O(V). Ponieważ wykonywanych jest V takich operacji to ich łączny czas wykonania wynosi $O(V^2)$. Każda krawędź jest badana tylko raz dzięki tablicy visited. Dodatkowo łączna liczba iteracji pętli for (dla relaksacji) wynosi E. Złożoność czasowa wynosi więc $O(E + V^2)$ $(= O(V^2))$.

2. Plan eksperymentu

- Pomiar czasu dokonywany był za pomocą narzędzia QueryPerformanceTimer.
- Liczba krawędzi do wygenerowania obliczana jest według wzoru na maksymalną ilość krawędzi dla danego rodzaju grafu (skierowany/nieskierowany) pomnożoną przez gęstość (%). Generowanie grafu rozpoczynam od jego inicjalizacji – utworzenia prymitywnego grafu spójnego, poprzez połączenie kolejnych wierzchołków grafu(1 z 2, 2 z 3 itd.) i wylosowaniu wag ze znacznie szerszego

zakresu niż pozostałym krawędziom, w ceku utrudnienia zadania algorytmowi Dijkstry. Następnie losuję pozostałe krawędzie i dopóki ich ilość nie będzie równa zadanej przypisuję im wagę z zakresu <0;2*V> (liczby całkowite) unikając generowania pętli.

- Dla algorytmu Prima pomiary zostały wykonane dla następującej ilości wierzchołków: {40, 80 ,120, 160, 200}. Dla każdej z tych wielkości wykonywane były pomiary dla czterech gęstości: {25%, 50%, 75%, 99%}. Dla każdego zestawu pomiary były wykonywane 100 razy, a wynik zaokrąglano. Dla algorytmu Dijkstry pomiary zostały wykonane dla następujących ilości wierzchołków: {400, 800, 1200, 1600, 2000}. Wybór wartości dla alg. Dijkstry podyktowany był zbyt szybkim wykonywaniem się algorytmu dla mniejszych wartości, co wiązało się z niską dokładnością pomiaru czasu.

3. Sposób generowania grafów

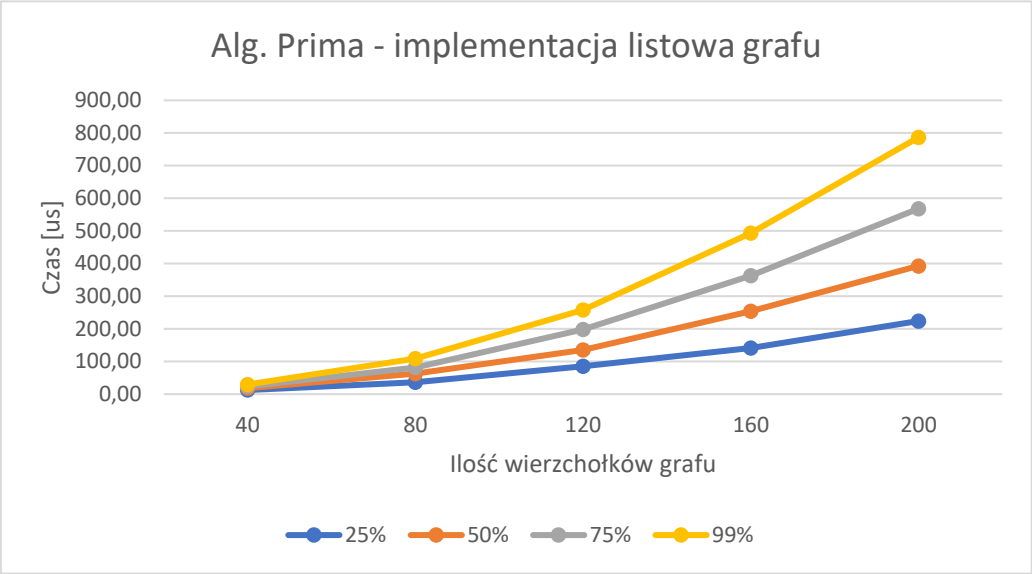
Pierwszym krokiem jest wyznaczenie za pomocą wzoru ilości krawędzi do wygenerowania dla zadanych parametrów. Następnie sprawdzany jest warunek czy dana gęstość zapewnia utworzenie grafu spójnego. Kolejnym krokiem jest inicjalizacja grafu (graf „prymitywny” 1->2, 2->3 …; dla reprezentacji macierzowej dodatkowo następuje zapełnienie jej wartościami nieskończoności) i dodanie ilości w ten sposób wygenerowanych krawędzi do obecnej ilości krawędzi. Po inicjalizacji wykonywana jest pętla, która trwa dopóki ilość obecnie wygenerowanych krawędzi jest mniejsza od zadanej. Następnie losowane są dwa wierzchołki za pomocą funnkcji rand() oraz wykonywane jest sprawdzenie, czy krawędź ta już istnieje oraz czy nie jest pętlą. Po spełnieniu warunków wykonywane jest losowanie wagi krawędzi, dodanie jej do struktury(macierz/lista) oraz zwiększenie licznika obecnie wygenerowanych krawędzi. Dla grafu nieskierowanego następuje dodatkowe utworzenie krawędzi w przeciwną stronę.

4. Wykresy

Algorytm Prima

Wykres zależności czasu wykonania algorytmu od ilości wierzchołków i gęstości dla poszczególnych implementacji:

-

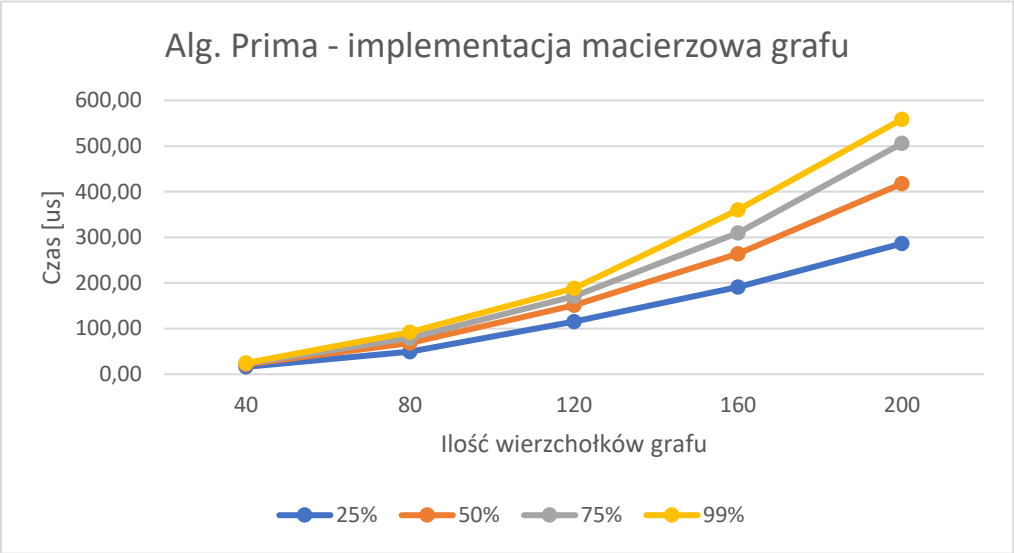


Wykres 1. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz gęstości grafu dla implementacji listowej

Dane (wyniki w [us])

		Ilość wierzchołków grafu				
		40	80	120	160	200
Gęstość	25%	12,55	36,22	85,78	141,16	223,95
	50%	18,22	62,56	135,25	254,05	392,75
	75%	23,56	81,39	198,39	363,22	568,01
	99%	29,60	108,91	257,89	493,66	786,43

Tabela 1. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz gęstości grafu dla implementacji listowej

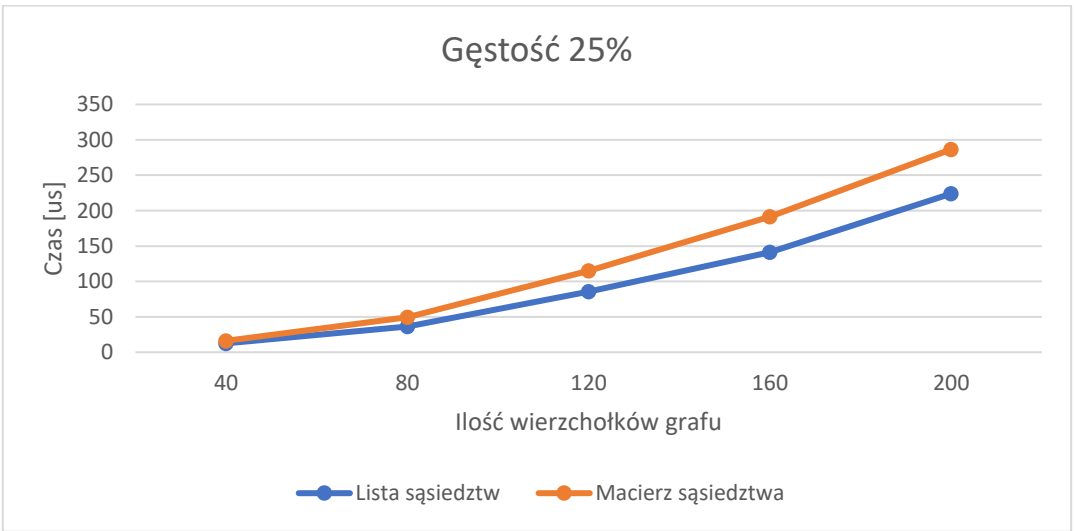


Wykres 2. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz gęstości grafu dla implementacji macierzowej

Dane (wyniki w [us]):

		Ilość wierzchołków grafu				
		40	80	120	160	200
Gęstość	25%	16,12	49,59	115,19	191,38	286,52
	50%	21,51	68,16	150,84	263,99	417,68
	75%	23,31	79,71	170,89	309,63	505,97
	99%	24,59	91,93	188,33	360,31	558,87

Tabela 2. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz gęstości grafu dla implementacji macierzowej



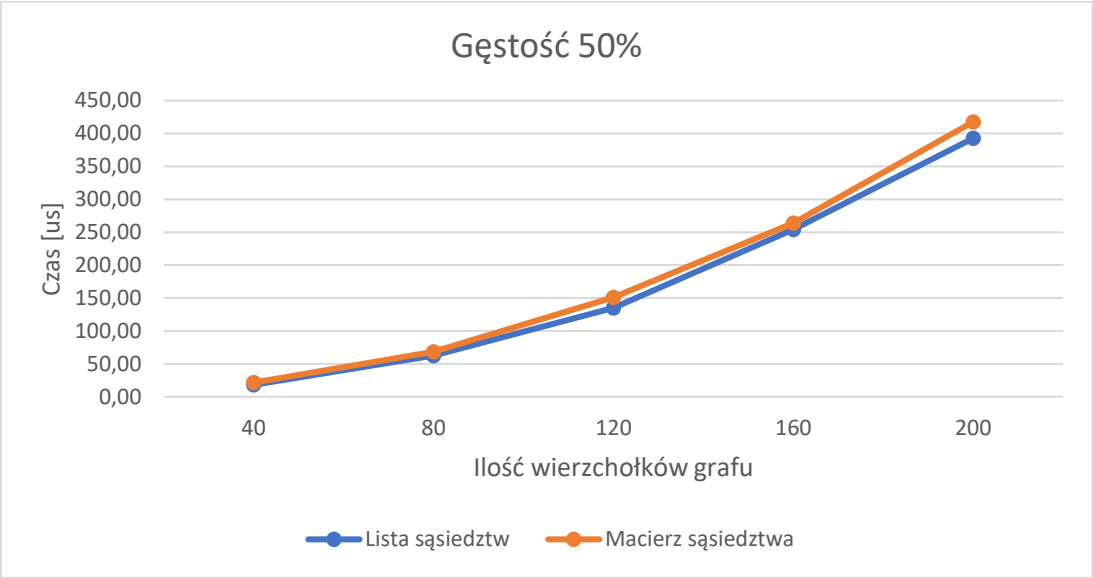
Wykres 3. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 25%

Dane (wyniki w [us]):

		Ilość wierzchołków grafu				
		40	80	120	160	200
Implementacja	Lista sąsiedztw	12,55	36,22	85,78	141,16	223,95
	Macierz sąsiedztwa	16,12	49,59	115,19	191,38	286,52

Tabela 3. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 25%

-

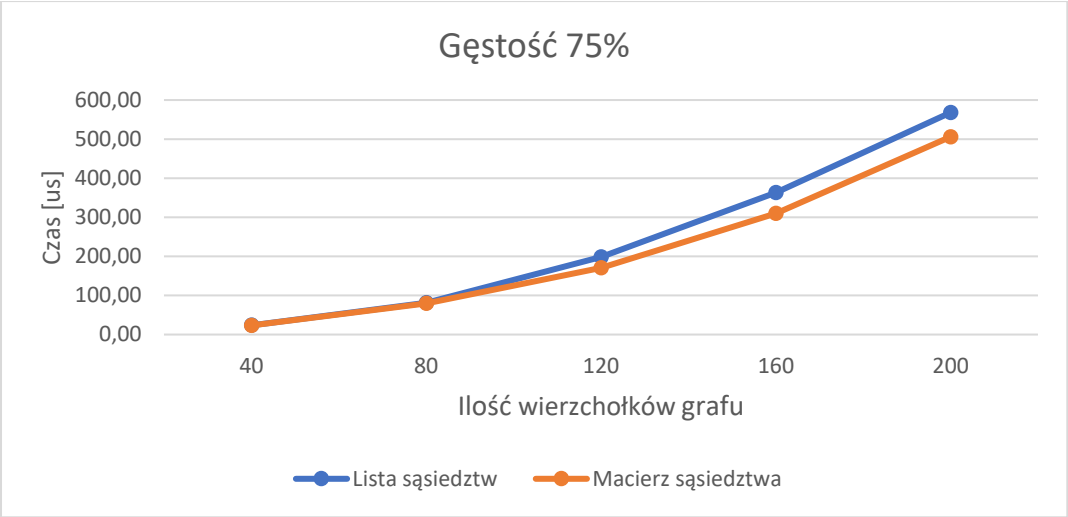


Wykres 4. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 50%

Dane (wyniki w [us]):

		Ilość wierzchołków grafu				
		40	80	120	160	200
Implementacja	Lista sąsiedztw	18,22	62,56	135,25	254,05	392,75
	Macierz sąsiedztwa	21,51	68,16	150,84	263,99	417,68

Tabela 4. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 50%

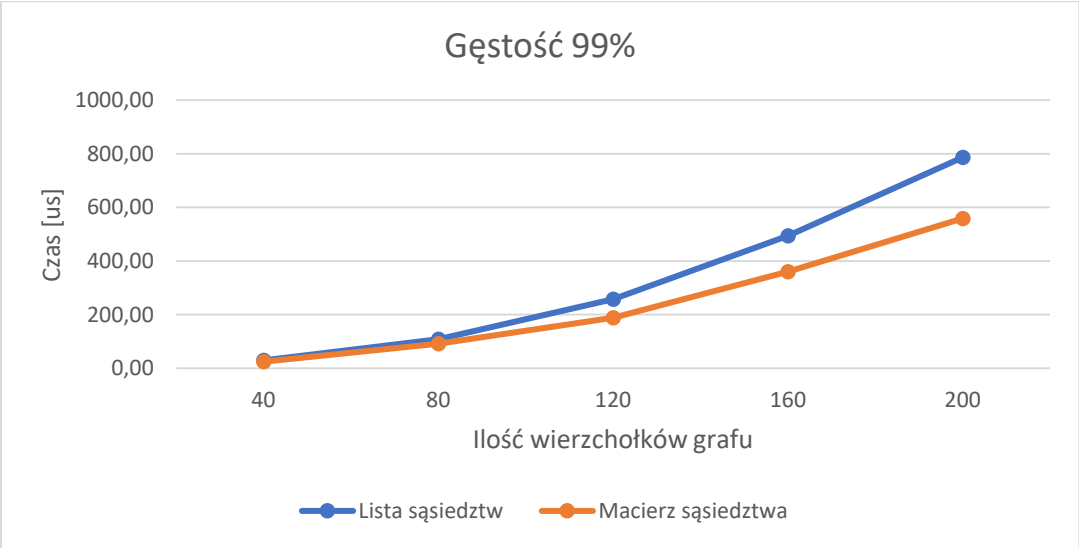


Wykres 5. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 75%

Dane (wyniki w [us]):

		Ilość wierzchołków grafu				
		40	80	120	160	200
Implementacja	Lista sąsiedztw	23,56	81,39	198,39	363,22	568,01
	Macierz sąsiedztwa	23,31	79,71	170,89	309,63	505,97

Tabela 5. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 75%



Wykres 6. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 99%

Dane (wyniki w [us]):

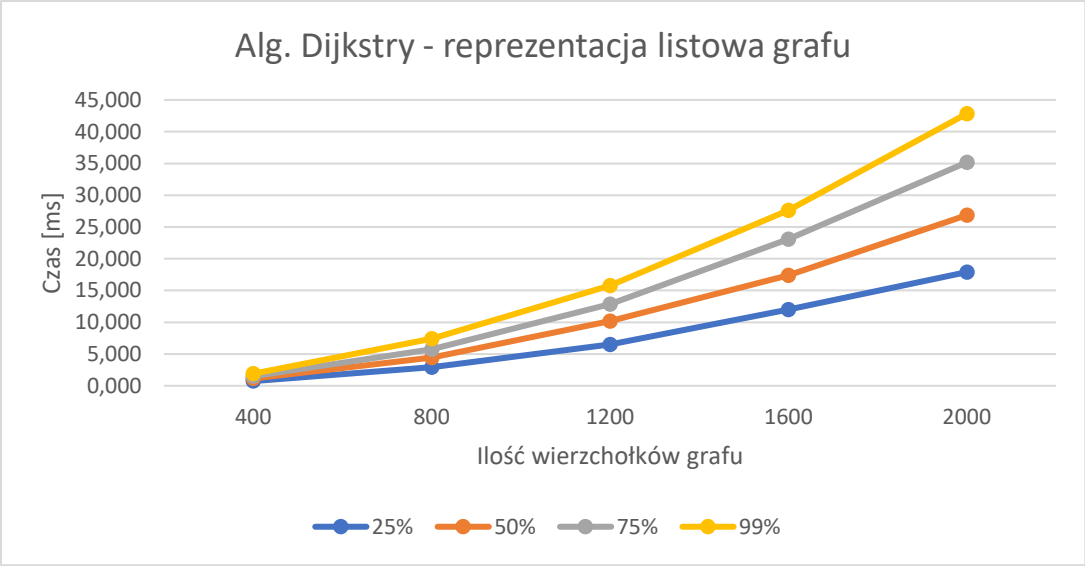
		Ilość wierzchołków grafu				
		40	80	120	160	200
Implementacja	Lista sąsiedztw	29,60	108,91	257,89	493,66	786,43
	Macierz sąsiedztwa	24,59	91,93	188,33	360,31	558,87

Tabela 6. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 99%

Algorytm Dijkstry

Wykres zależności czasu wykonania algorytmu od ilości wierzchołków i gęstości dla poszczególnych implementacji:

-

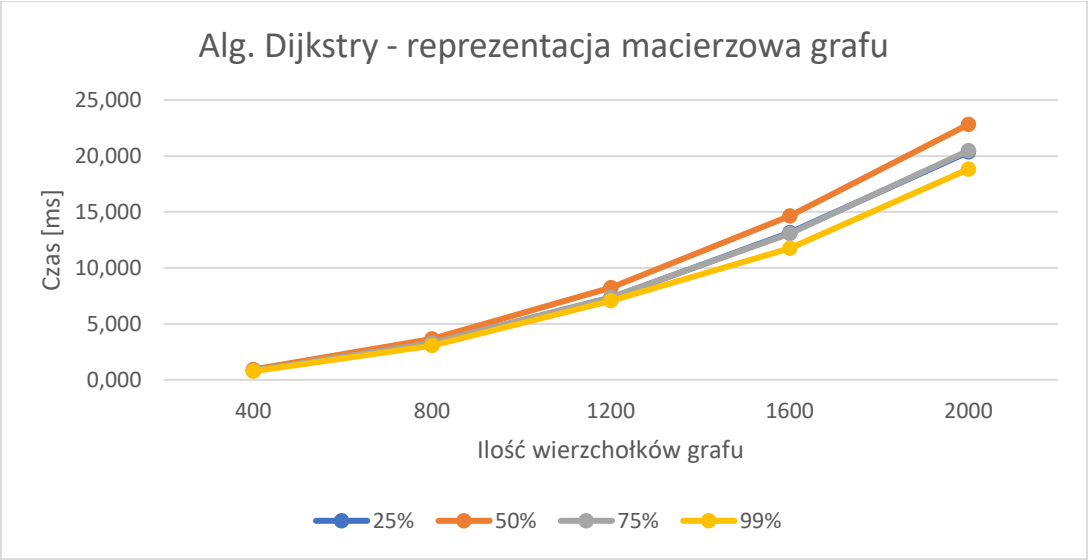


Wykres 7. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz gęstości grafu dla implementacji listowej

Dane (wyniki w [ms]):

		Ilość wierzchołków grafu				
		400	800	1200	1600	2000
Gęstość	25%	0,733	2,902	6,508	11,983	17,893
	50%	1,145	4,441	10,158	17,409	26,857
	75%	1,497	5,741	12,876	23,086	35,173
	99%	1,902	7,401	15,791	27,598	42,829

Tabela 7. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz gęstości grafu dla implementacji listowej

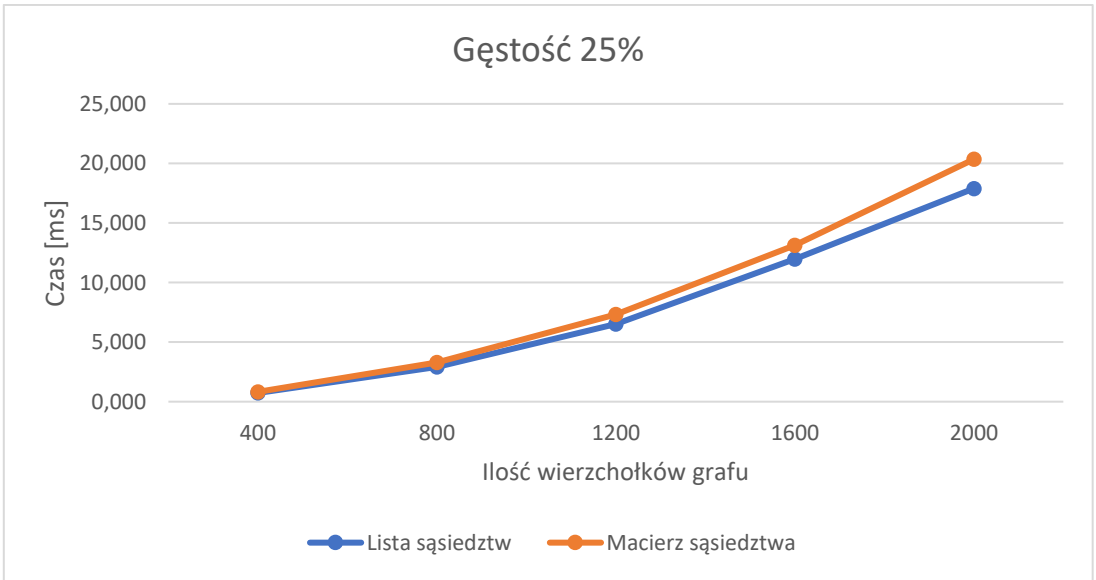


Wykres 8. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz gęstości grafu dla implementacji macierzowej

Dane (wyniki w [ms]):

		Ilość wierzchołków grafu				
		400	800	1200	1600	2000
Gęstość	25%	0,825	3,286	7,312	13,137	20,350
	50%	0,921	3,662	8,237	14,632	22,820
	75%	0,825	3,279	7,373	13,070	20,471
	99%	0,764	3,065	7,056	11,744	18,826

Tabela 8. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz gęstości grafu dla implementacji macierzowej



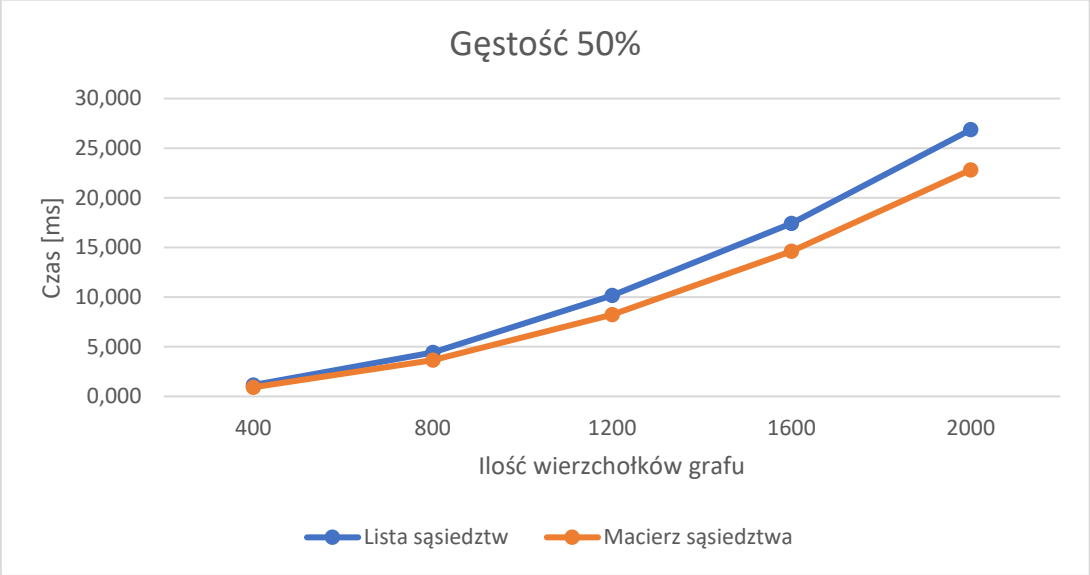
Wykres 9. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 25%

Dane (wyniki w [ms]):

		Ilość wierzchołków grafu				
		400	800	1200	1600	2000
Implementacja	Lista sąsiedztw	0,733	2,902	6,508	11,983	17,893
	Macierz sąsiedztwa	0,825	3,286	7,312	13,137	20,350

Tabela 9. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 25%

-

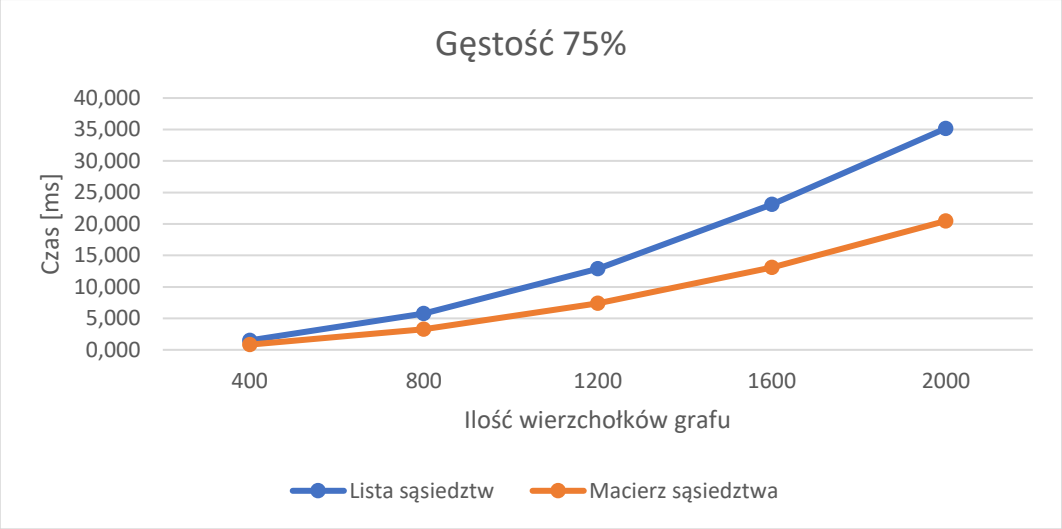


Wykres 10. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 50%

Dane (wyniki w [ms]):

		Ilość wierzchołków grafu				
		400	800	1200	1600	2000
Implementacja	Lista sąsiedztw	1,145	4,441	10,158	17,409	26,857
	Macierz sąsiedztwa	0,921	3,662	8,237	14,632	22,820

Tabela 10. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 50%

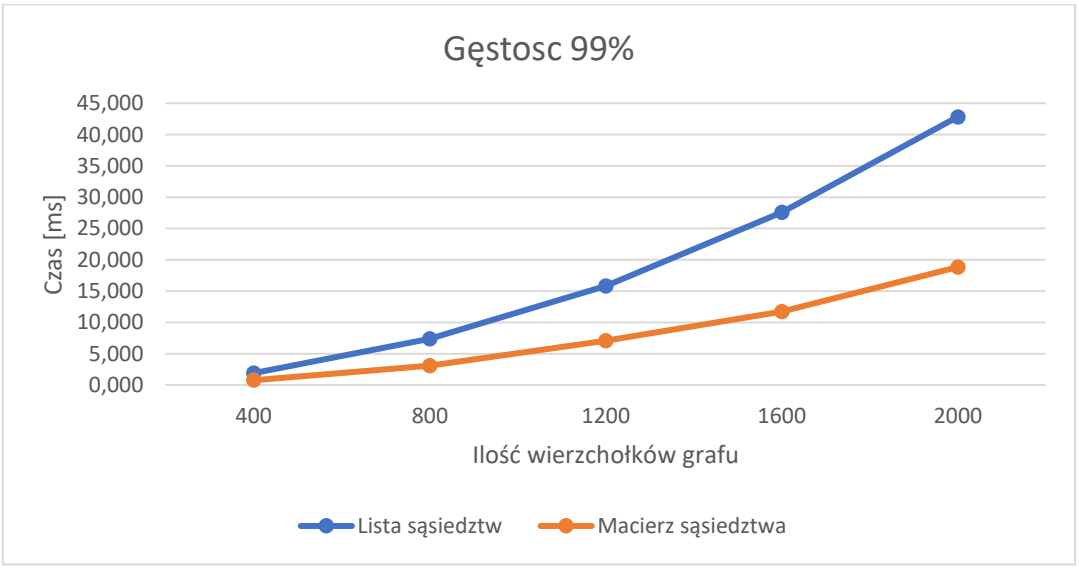


Wykres 11. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 75%

Dane (wyniki w [ms]):

		Ilość wierzchołków grafu				
		400	800	1200	1600	2000
Implementacja	Lista sąsiedztw	1,497	5,741	12,876	23,086	35,173
	Macierz sąsiedztwa	0,825	3,279	7,373	13,070	20,471

Tabela 11. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 75%



Wykres 12. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 99%

Dane (wyniki w [ms]):

		Ilość wierzchołków grafu				
		400	800	1200	1600	2000
Implementacja	Lista sąsiedztw	1,902	7,401	15,791	27,598	42,829
	Macierz sąsiedztwa	0,764	3,065	7,056	11,744	18,826

Tabela 12. Zależność czasu wykonania algorytmu od ilości wierzchołków oraz implementacji grafu dla gęstości 99%

Wnioski:

Możemy przyjąć, iż wyniki otrzymane w eksperymencie pokrywają się z teoretycznymi założeniami:

W algorytmie Prima widzimy jak czas wykonania zależny jest zarówno od ilości wierzchołków, a także gęstości grafu dla obu z reprezentacji. W tym przypadku dla mniejszych gęstości lepiej sprawdza się implementacja oparta na listach sąsiedztw ze względu na brak konieczności poruszania się po krawędziach nieistniejących (krawędzie o wartości równej ∞); Jednak im gęstszy oraz większy pod względem ilości wierzchołków jest dany graf, tym coraz bardziej widoczna jest przewaga implementacji opartej na macierzy sąsiedztw. Wiąże się to z brakiem konieczności „wchodzenia" w strukturę, by dodać potrzebne nam dane do kolejki priorytetowej (w tablicy do wszystkiego mamy dostęp za pomocą indeksów). Samo poruszanie się po tablicy jest również szybsze niż przeszukiwanie listy za pomocą iteratora.

W algorytmie Dijkstry zauważamy podobny trend. Tutaj macierz jest szybsza już przy gęstości na poziomie 50%. Jest to prawdopodobnie spowodowane koniecznością częstego odwoływania się do danych znajdujących się w strukturze podczas relaksacji oraz większą ilością wierzchołków użytych w tym algorytmie. Dodatkowo zauważamy, iż czas wykonania w implementacji macierzowej nie zależy od gęstości grafu. Jest to związane koniecznością przeszukiwania nawet tych indeksów dla których krawędź nie istnieje (ma wartość ∞). W reprezentacji listowej musimy zwrócić uwagę na gęstość, ponieważ w tym przypadku zwiększa się liczba krawędzi, które musimy przeszukać dla badanego w danym momencie wierzchołka (w danym przebiegu głównej pętli algorytmu).