Politechnika
Wrocławska,
Wydział Informatyki
i Telekomunikacji
Semestr IV Rok II

Zadanie projektowe nr 2

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

Wykonał:
Patryk Ignasiak 263889
Gr. Wtorek 15:15 P
Prowadzący:
Mgr. Inż. Antoni Sterna
Przedmiot:
Struktury danych i złożoność obliczeniowa – Projekt

Wrocław, 27 Maj 2023

Spis treści

Wstęp

W tym projekcie została przeprowadzona analiza porównawcza algorytmów grafowych. Rozważone zostały następujące problemy:

- Wyznaczenie minimalnego drzewa rozpinającego
  - Algorytm Kruskala – O(ElogV)
  - Algorytm Prima – O(ElogV)
- Wyznaczenie najkrótszej ścieżki w grafie
  - Algorytm Dijkstry – O(ElogV)
  - Algorytm Bellmana-Forda – O(VE)
- Wyznaczenie maksymalnego przepływu
  - Algorytm Forda-Fulkersona – O(V3)

W celu ułatwienia implementacji wierzchołki w grafach są numerowane w sposób ciągły od 0. Natomiast za brak krawędzi, przyjęta została umowna wartość ∞, a jej wartość jest równa INT32_MAX.

Opis eksperymentu

Wszystkie testy zostały przeprowadzone dla 20 wielkości oraz 4 gęstości, na dwóch reprezentacjach.

- Wielkości grafów: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120 ,130, 140, 150, 160, 170, 180, 190, 200.
- Gęstości grafów: 25%, 50%, 75%, 99%.
- Reprezentacje grafów: lista sąsiadów, macierz sąsiedztwa.

Czasy testów były mierzone z wykorzystaniem „CHRONO” z dokładnością do nanosekund i zapisywane w tablicy typu „long long” po zakończeniu wszystkich testów, z wykorzystaniem metody „saveTimeTestResult” pochodzącej z klasy „Data”, średnie czasy (czyli podzielone przez 20) zostały zapisane w plikach „.txt”. Następnie z wykorzystaniem programu „Excel” uzyskane wyniki zostały obrobione.

Po załadowaniu losowo wygenerowanych danych do struktury testy czasu odbywały się w sposób przedstawiony poniżej:

```
auto start = chrono::steady_clock::now(); //zapisanie czasu startowego
mstObj->kruskalOnList(graph);            //testowana procedura
auto end = chrono::steady_clock::now();   //zapisanie czasu końcowego
auto duration = end - start;             //obliczenie czasu trwania

//zapisanie do tablicy
times[z][0] = 0; //0 - lista, 1 - macierz
times[z][1] = vertices[i]; //ilość wierzchołków
times[z][2] = filling[j]; //ilość gęstość
//Kruskal
times[z][3]+=chrono::duration_cast<chrono::microseconds>(duration).count();
```

Poniżej zostały przedstawione wyniki pomiarów. Pomiary przeprowadzone zostały na laptopie Lenovo ThinkPad T14. Podstawowe parametry:

- RAM: 32GB

- Procesor: AMD Ryzen 7 PRO 4750U

Opis metody generowania grafów

Jest to autorska metoda, która na początek wylicza maksymalną ilość krawędzi jaka mogłaby wystąpić w grafie w zależności czy jest to graf do wyznaczenia MST czy nie. Następnie, ilość ta jest mnożona przez % wypełnienia w celu uzyskania odpowiedniej ilości krawędzi.

Generowane są wierzchołki startowy i końcowy w celu zachowania spójności struktury plików z danymi. Po tym przechodzimy do generowania drzewa rozpinającego, a następnie pozostałych krawędzi.

Generowanie krawędzi polega na wylosowaniu wierzchołków oraz wagi krawędzi. Następnie sprawdzamy czy taka krawędź już istnieje, jeśli tak to losujemy odnowa, jeśli nie to dodajemy do pliku.

```
void Data::generateData(int vertices, int filling, string filename, int min, int max, bool isMST) {
    srand(time(NULL));
    int maxEdges = 0;
    for (int i = vertices - 1; i > 0; i--) {
         maxEdges += i;
    }

    if (!isMST) {
        maxEdges *= 2;
    }

    int edges = maxEdges * filling / 100;
    int startVertice = rand() % vertices;
    int endVertice = rand() % vertices;
    while(startVertice == endVertice) {
        endVertice = rand() % vertices;
    }

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> distValue(min, max);
    uniform_int_distribution<> distVertice(0, vertices-1);

    //Tworzymy plik i zapisujemy rozmiar
    ofstream file(filename);

    file << edges << " " << vertices << " " << startVertice << " " << endVertice << endl;
```

```
    file.close();



    //otwieramy plik do nadpisania
    //i zapisujemy generowane dane
    file.open(filename, ios_base::app);

    int** edgesMemory = new int*[edges];
    int k = 0;
    //drzewo rozpinające
    for (int i = 0; i < vertices - 1; i++){
        file << i << " " << i + 1 << " " << distValue(gen) << endl;

        int* edge = new int[2];
        edge[0] = i;
        edge[1] = i+1;
        edgesMemory[k] = edge;
        k++;
    }
    //pozostałe krawędzie
    for (int i = 0; i < edges - vertices + 1; i++) {
                    int firstVertice = distVertice(gen);
                    int secondVertice = distVertice(gen);
                    while (firstVertice == secondVertice) {
                                secondVertice = distVertice(gen);
                    }

                    bool isUnique = true;

        for (int j = 0; j < k; j++) {
            if (isMST) {
                if ((firstVertice == edgesMemory[j][0] && secondVertice == edgesMemory[j][1])
                        || (secondVertice == edgesMemory[j][0] && firstVertice == edgesMemory[j][1])){
                    isUnique = false;
                }
            }
            else {
                if (firstVertice == edgesMemory[j][0] && secondVertice == edgesMemory[j][1]) {
                    isUnique = false;
                }
            }
        }
        if (isUnique) {
            file << firstVertice << " " << secondVertice << " " << distValue(gen) << endl;

            int* edge = new int[2];
            edge[0] = firstVertice;
            edge[1] = secondVertice;
            edgesMemory[k] = edge;
            k++;
        }else {
            i--;
        }
    }
    file.close();
}
```

Wyznaczenie minimalnego drzewa rozpinającego

Tabela czasów – Algorytm Kruskala

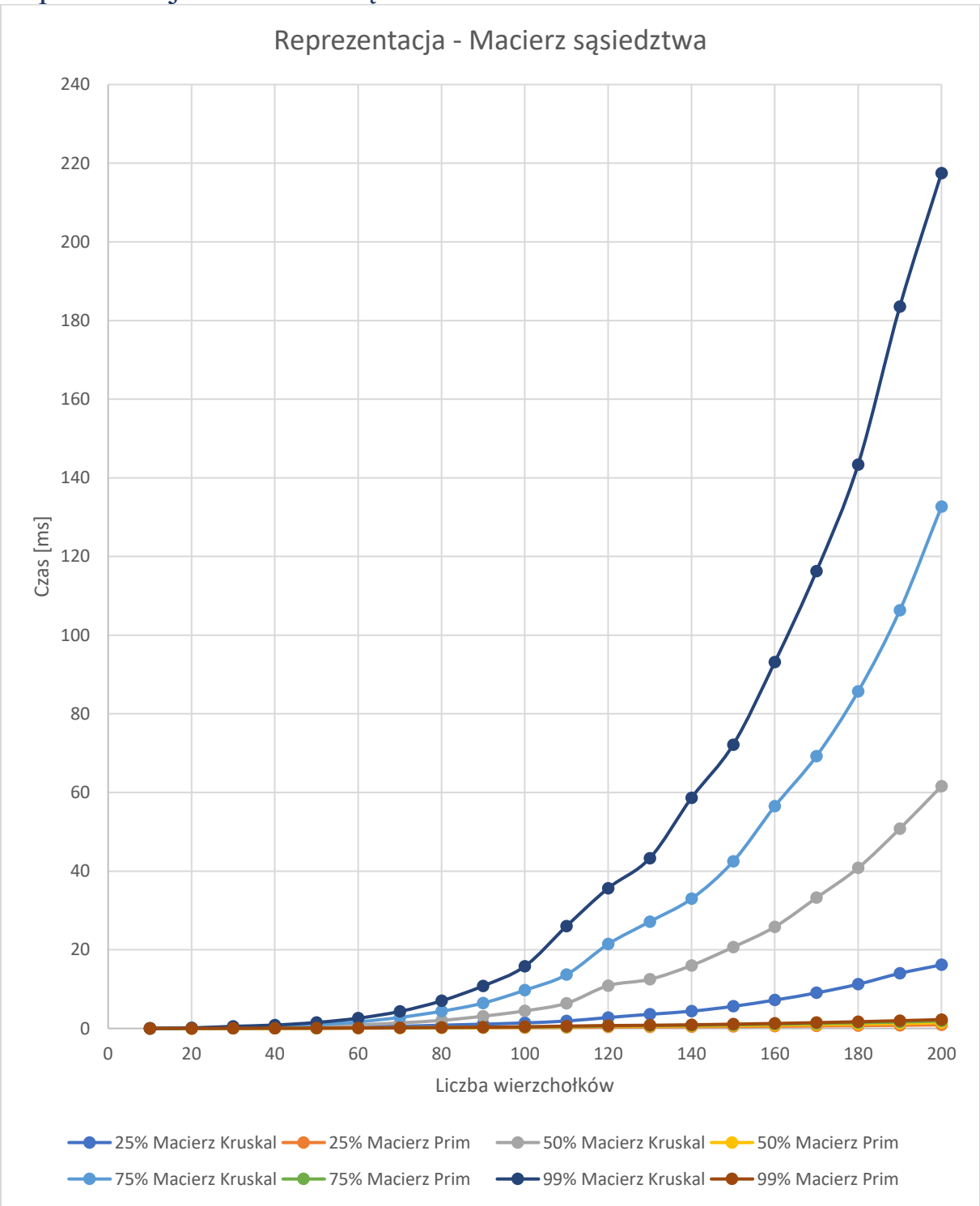
Ilość wierzchołków	Gęstość - 25%		Gęstość - 50%		Gęstość - 75%		Gęstość - 99%	
	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]
10	0,007	0,005	0,014	0,010	0,019	0,014	0,029	0,021
20	0,034	0,025	0,071	0,058	0,113	0,089	0,185	0,146
30	0,100	0,074	0,275	0,218	0,465	0,372	0,618	0,538
40	0,244	0,209	0,500	0,406	0,786	0,604	1,047	0,867
50	0,383	0,288	0,765	0,595	1,130	0,982	1,760	1,525
60	0,536	0,424	1,021	0,891	1,941	1,689	2,909	2,611
70	0,737	0,559	1,542	1,419	3,075	2,800	4,718	4,329
80	0,979	0,829	2,267	2,057	4,554	4,374	7,207	7,016
90	1,245	1,093	3,370	3,114	6,579	6,450	10,843	10,822
100	1,543	1,414	4,696	4,475	9,699	9,731	16,715	15,805
110	2,058	1,886	6,815	6,402	13,827	13,666	25,971	26,004
120	2,871	2,780	11,039	10,877	23,455	21,440	36,073	35,652
130	3,771	3,606	13,272	12,484	27,940	27,159	44,643	43,290
140	4,789	4,391	17,459	16,013	33,431	32,971	57,016	58,606
150	5,919	5,625	21,069	20,678	42,594	42,455	75,259	72,129
160	7,430	7,219	25,548	25,829	55,427	56,482	94,506	93,139
170	8,996	9,044	33,192	33,224	70,915	69,237	120,566	116,263
180	11,421	11,253	40,069	40,811	87,978	85,674	150,345	143,372
190	13,939	14,015	50,670	50,792	104,224	106,291	185,431	183,510
200	17,470	16,169	62,146	61,608	131,458	132,666	221,536	217,486

Tabela czasów – Algorytm Prima

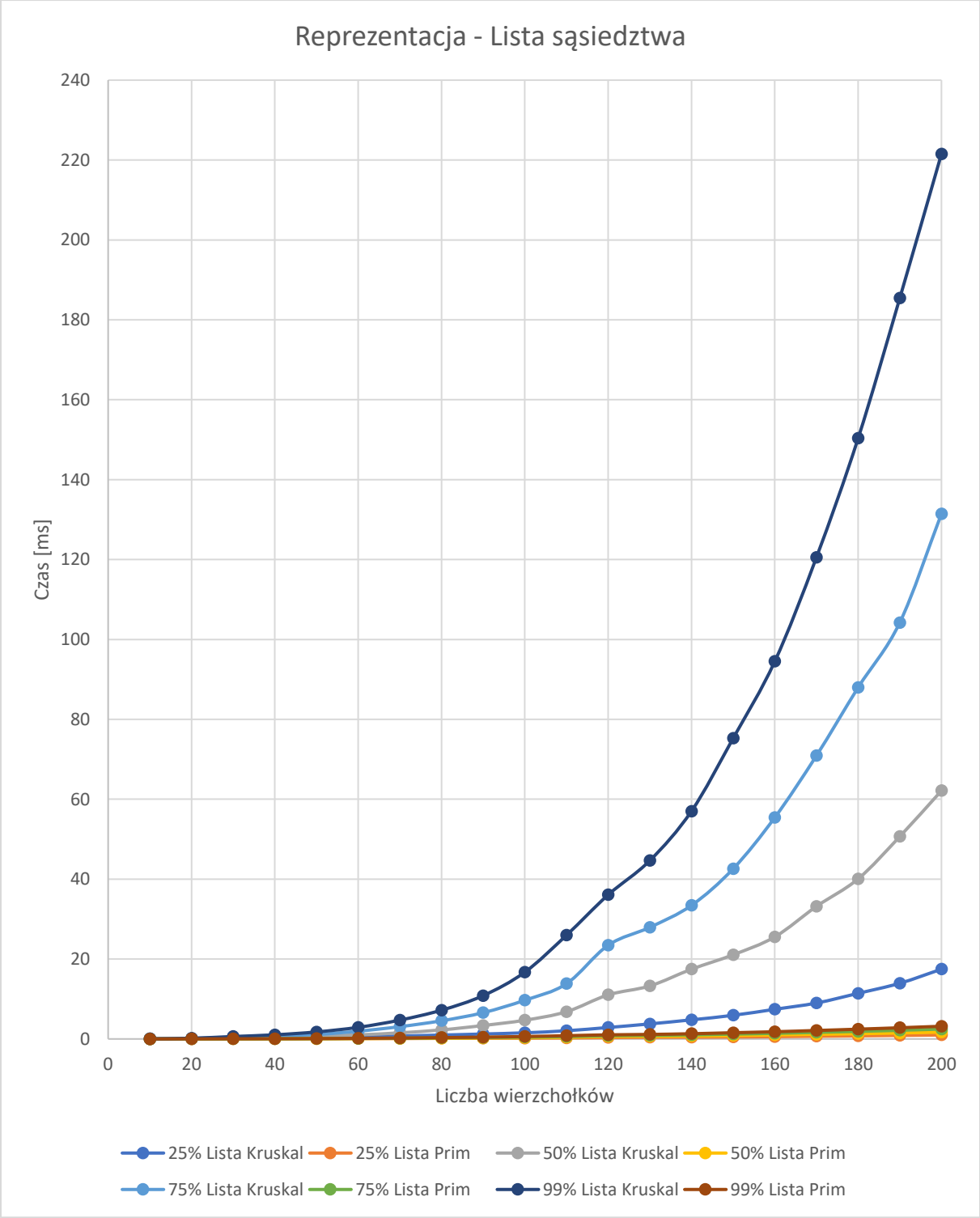
Ilość wierzchołków	Gęstość - 25%		Gęstość - 50%		Gęstość - 75%		Gęstość - 99%	
	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]
10	0,005	0,005	0,005	0,005	0,005	0,004	0,005	0,005
20	0,012	0,012	0,015	0,016	0,015	0,016	0,020	0,017
30	0,023	0,027	0,029	0,032	0,037	0,039	0,046	0,037
40	0,041	0,045	0,050	0,052	0,057	0,057	0,066	0,062
50	0,059	0,066	0,073	0,077	0,083	0,083	0,103	0,092
60	0,087	0,089	0,099	0,108	0,132	0,124	0,156	0,129
70	0,109	0,110	0,141	0,150	0,193	0,170	0,237	0,186
80	0,148	0,158	0,185	0,194	0,277	0,233	0,338	0,254
90	0,181	0,187	0,261	0,257	0,353	0,302	0,460	0,332
100	0,208	0,215	0,332	0,315	0,472	0,376	0,614	0,422
110	0,256	0,267	0,424	0,395	0,602	0,471	0,820	0,593
120	0,344	0,349	0,648	0,585	0,824	0,660	1,037	0,730
130	0,412	0,406	0,665	0,601	0,916	0,733	1,137	0,815
140	0,444	0,447	0,746	0,658	1,034	0,803	1,311	0,931
150	0,501	0,489	0,870	0,757	1,216	0,938	1,566	1,096
160	0,580	0,560	1,014	0,873	1,411	1,083	1,810	1,275
170	0,675	0,635	1,171	0,998	1,649	1,254	2,119	1,481
180	0,773	0,712	1,329	1,131	1,892	1,435	2,451	1,712
190	0,881	0,803	1,501	1,283	2,171	1,643	2,804	1,977
200	0,995	0,904	1,722	1,481	2,475	1,900	3,182	2,248

Wykresy – w zależności od reprezentacji grafu

Reprezentacja – macierz sąsiedztwa

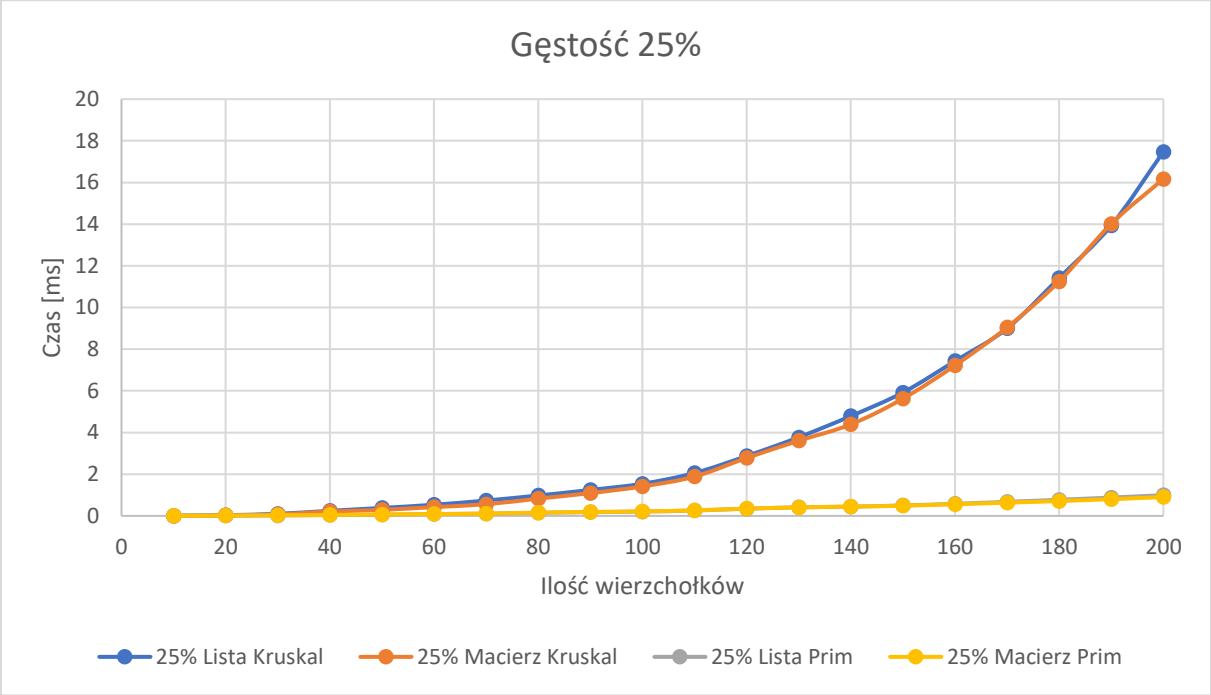


Reprezentacja – lista sąsiadów

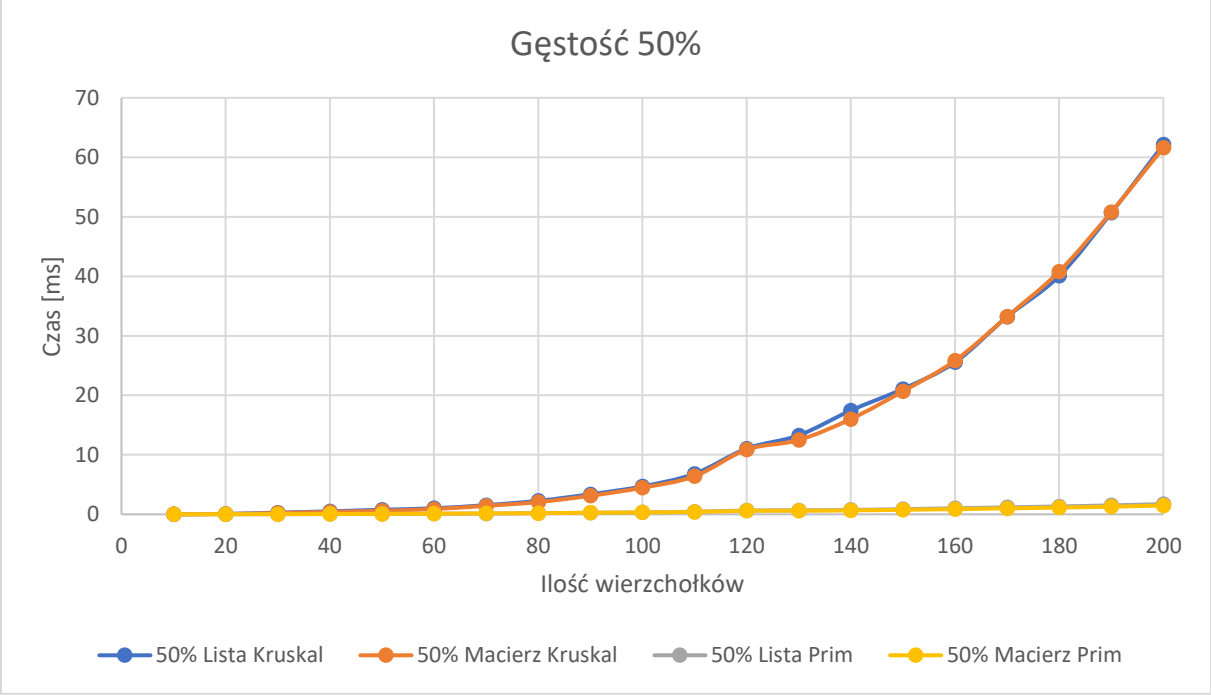


Wykresy – w zależności od gęstości grafu

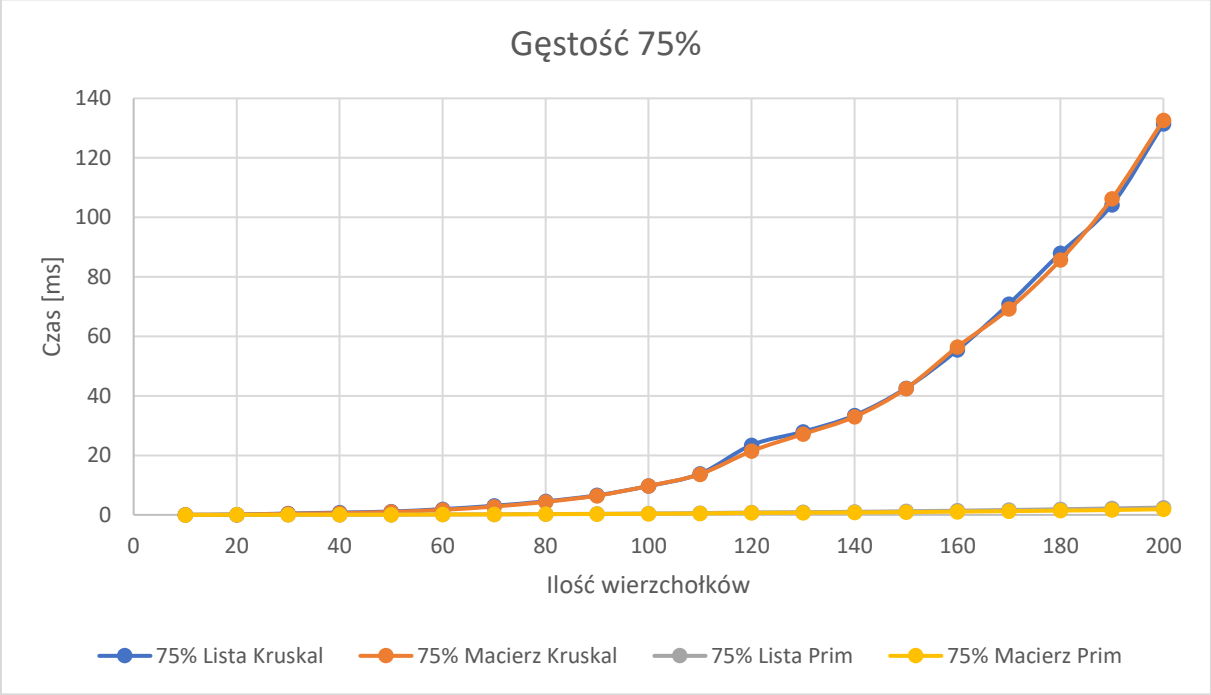
Gęstość - 25%



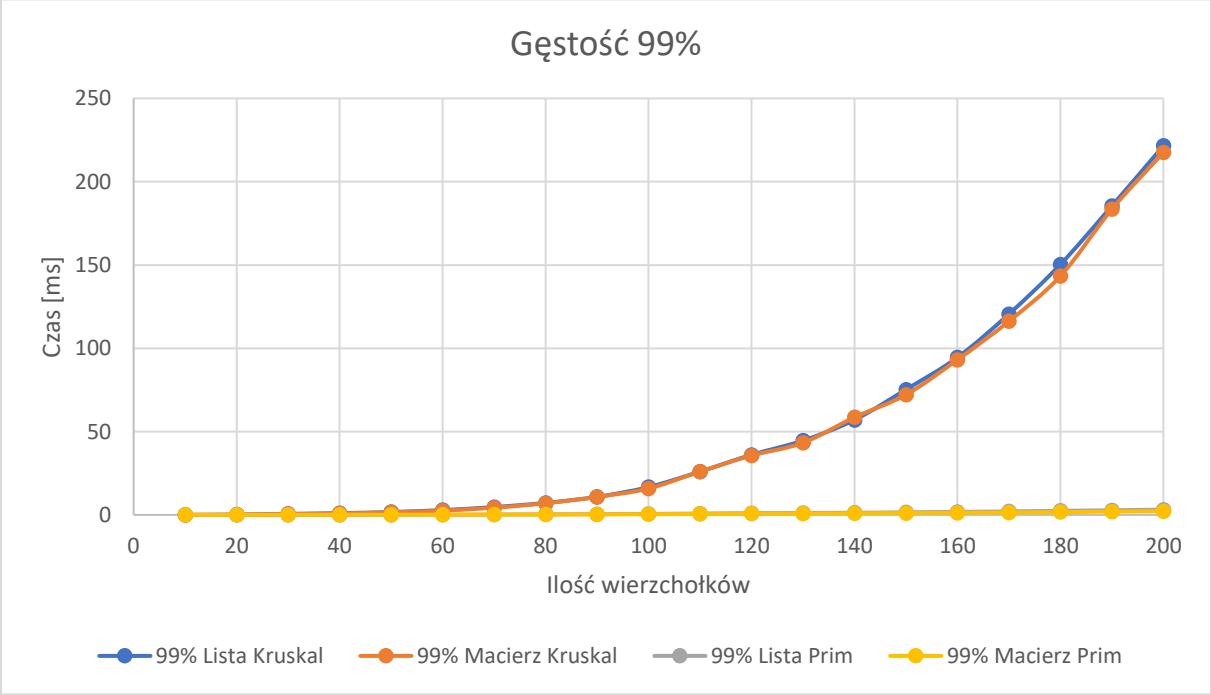
Gęstość - 50%



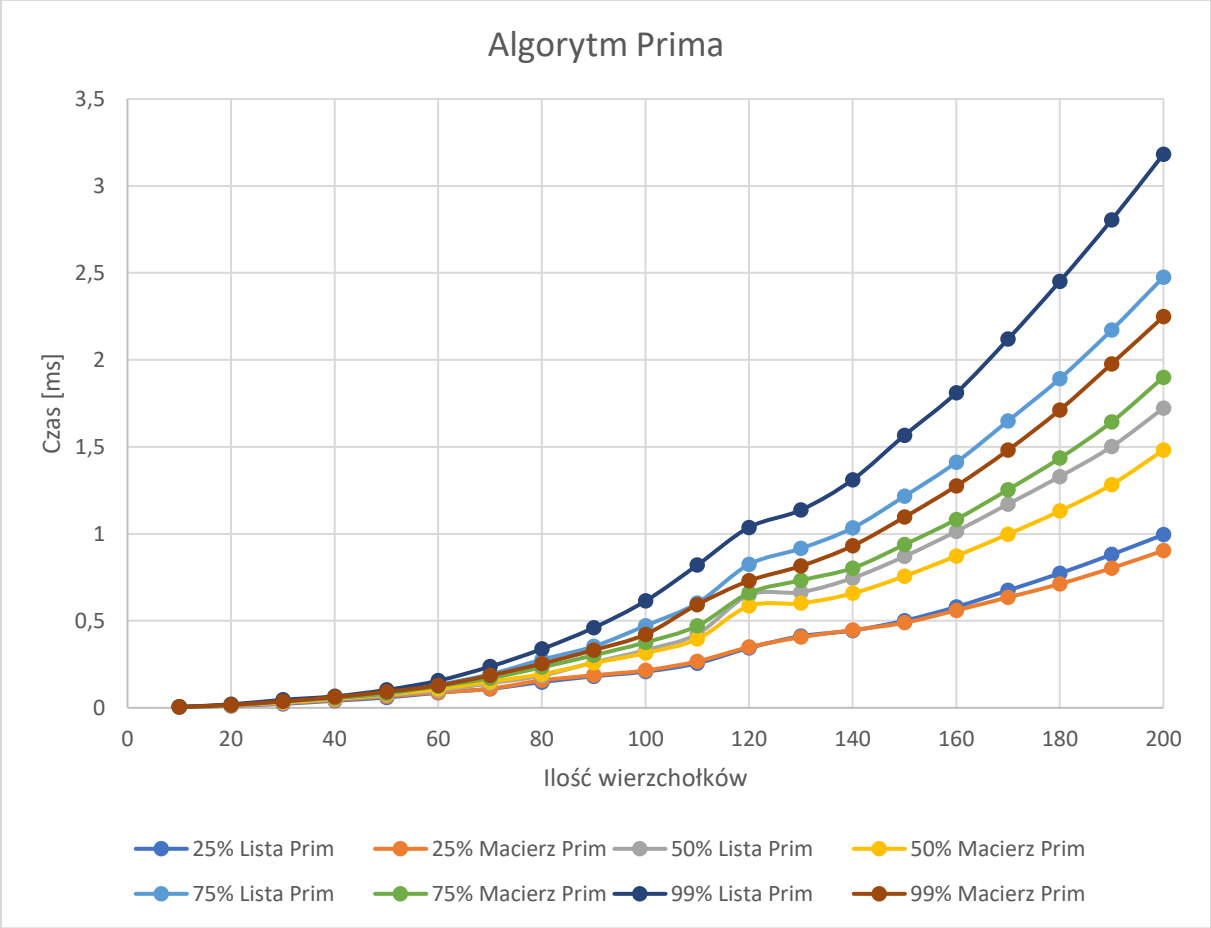
Gęstość - 75%



Gęstość - 99%



Dodatkowy wykres – Algorytm Prima



Podsumowanie

Jak można zauważyć na powyższych wykresach, algorytm Kruskala jest znacznie wolniejszy niż algorytm Prima. Z założeń teoretycznych wynika, że oba te algorytmy mają złożoność O(ElogV). Tak znacząca różnica może wynikać ze sposobu implementacji algorytmu Kruskala, ponieważ na początku tworzę listę krawędzi co ma złożoność $O(V^2)$ co znacząco mogło spowolnić ten algorytm.

Możemy również zauważyć, iż algorytm Kruskala jest ściśle powiązany z gęstością grafu (im większa gęstość, tym więcej krawędzi), tym większy czas wykonywania algorytmu. Natomiast, jeżeli chodzi o typ reprezentacji, to nieznacznie szybsza okazała się macierz sąsiedztwa. Większe różnice natomiast możemy zauważyć w algorytmie Prima, im większa gęstość grafu, tym reprezentacja macierzowa wypada znacząco lepiej. Algorytm Prima podobnie jak Kruskala jest powiązany z ilością krawędzi, jednakże jego czasy rosną wolniej. Można po tym wywnioskować, iż jego użyteczna złożoność wynosi O(VlogV), co jest mniejsze niż złożoność teoretyczna O(ElogV).

Wyznaczenie najkrótszej ścieżki w grafie

Tabela czasów – Algorytm Dijkstry

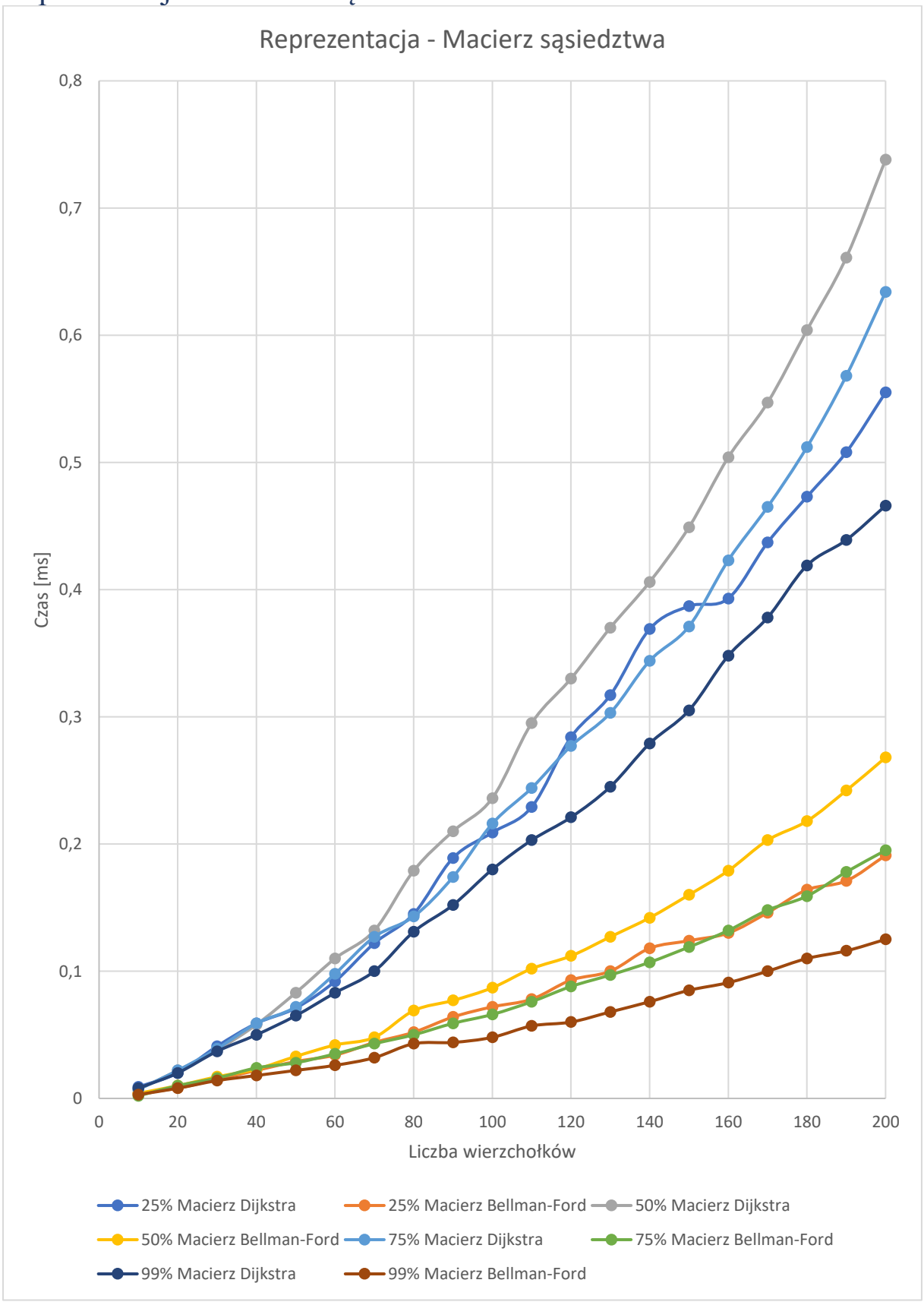
Ilość wierzchołków	Gęstość - 25%		Gęstość - 50%		Gęstość - 75%		Gęstość - 99%	
	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]
10	0,010	0,009	0,009	0,007	0,009	0,007	0,008	0,008
20	0,021	0,020	0,024	0,022	0,024	0,022	0,023	0,020
30	0,044	0,041	0,041	0,038	0,048	0,039	0,044	0,037
40	0,067	0,059	0,065	0,058	0,070	0,059	0,066	0,050
50	0,087	0,071	0,089	0,083	0,090	0,072	0,090	0,065
60	0,102	0,092	0,120	0,110	0,119	0,098	0,116	0,083
70	0,136	0,122	0,144	0,132	0,154	0,127	0,149	0,100
80	0,163	0,145	0,190	0,179	0,194	0,143	0,211	0,131
90	0,197	0,189	0,237	0,210	0,227	0,174	0,234	0,152
100	0,229	0,209	0,254	0,236	0,272	0,216	0,292	0,180
110	0,249	0,229	0,308	0,295	0,323	0,244	0,359	0,203
120	0,294	0,284	0,364	0,330	0,395	0,277	0,441	0,221
130	0,331	0,317	0,403	0,370	0,457	0,303	0,523	0,245
140	0,383	0,369	0,463	0,406	0,545	0,344	0,661	0,279
150	0,412	0,387	0,535	0,449	0,627	0,371	0,812	0,305
160	0,426	0,393	0,600	0,504	0,727	0,423	0,890	0,348
170	0,472	0,437	0,667	0,547	0,833	0,465	0,994	0,378
180	0,517	0,473	0,759	0,604	0,951	0,512	1,170	0,419
190	0,568	0,508	0,875	0,661	1,080	0,568	1,277	0,439
200	0,618	0,555	0,972	0,738	1,247	0,634	1,426	0,466

Tabela czasów – Algorytm Bellmana-Forda

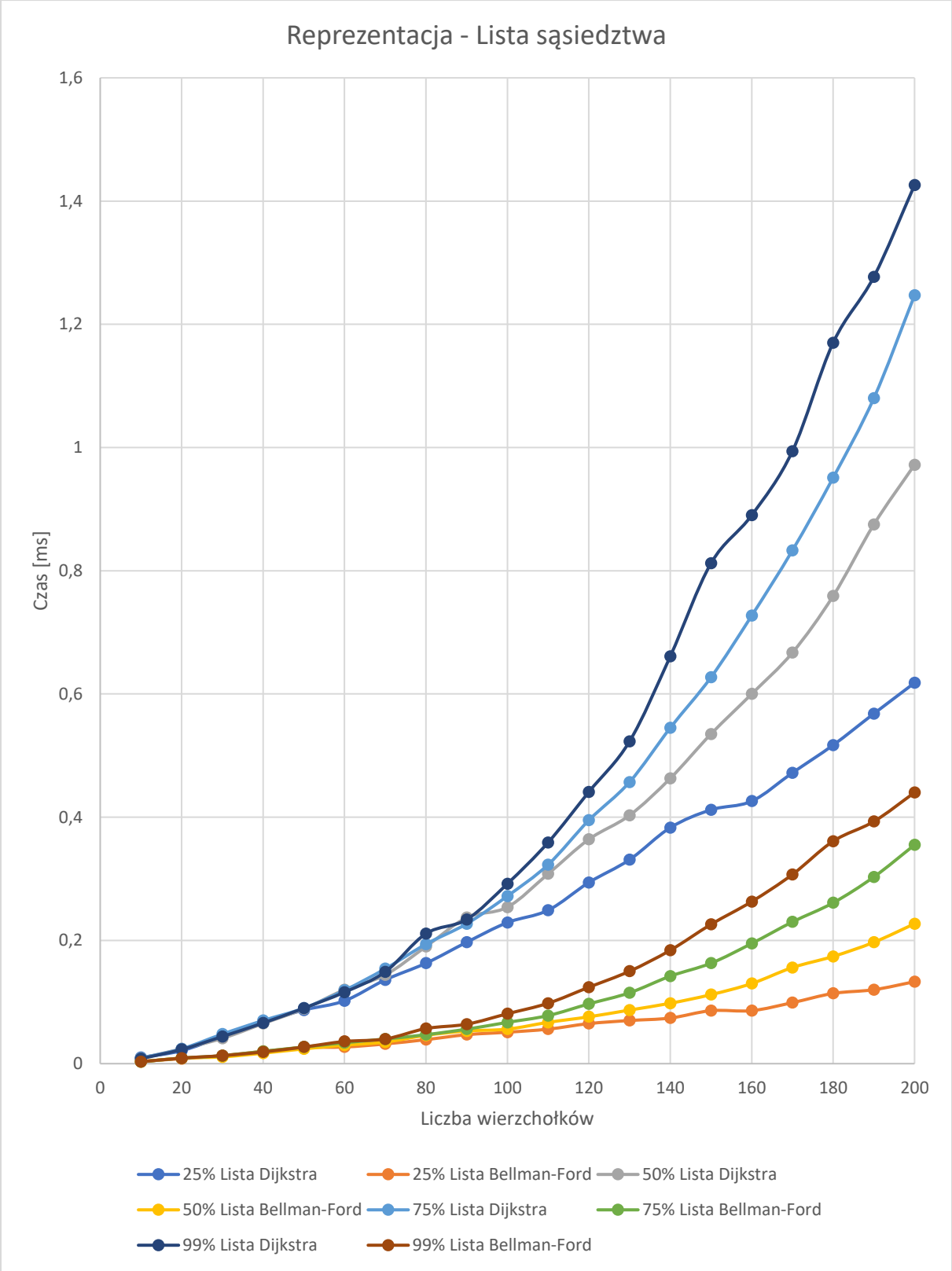
Ilość wierzchołków	Gęstość - 25%		Gęstość - 50%		Gęstość - 75%		Gęstość - 99%	
	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]
10	0,004	0,004	0,003	0,004	0,003	0,002	0,003	0,003
20	0,008	0,008	0,009	0,010	0,009	0,010	0,009	0,008
30	0,013	0,015	0,011	0,017	0,013	0,016	0,013	0,014
40	0,020	0,022	0,017	0,023	0,020	0,024	0,019	0,018
50	0,026	0,029	0,024	0,033	0,027	0,028	0,027	0,022
60	0,027	0,034	0,030	0,042	0,034	0,035	0,036	0,026
70	0,032	0,044	0,035	0,048	0,040	0,043	0,040	0,032
80	0,039	0,052	0,046	0,069	0,047	0,050	0,057	0,043
90	0,047	0,064	0,053	0,077	0,056	0,059	0,064	0,044
100	0,051	0,072	0,056	0,087	0,067	0,066	0,081	0,048
110	0,056	0,078	0,067	0,102	0,078	0,076	0,098	0,057
120	0,065	0,093	0,076	0,112	0,097	0,088	0,124	0,060
130	0,070	0,100	0,087	0,127	0,115	0,097	0,150	0,068
140	0,074	0,118	0,098	0,142	0,142	0,107	0,184	0,076
150	0,086	0,124	0,112	0,160	0,163	0,119	0,226	0,085
160	0,086	0,130	0,130	0,179	0,195	0,132	0,263	0,091
170	0,099	0,146	0,156	0,203	0,230	0,148	0,307	0,100
180	0,114	0,164	0,174	0,218	0,261	0,159	0,361	0,110
190	0,120	0,171	0,197	0,242	0,303	0,178	0,393	0,116
200	0,133	0,191	0,227	0,268	0,355	0,195	0,440	0,125

Wykresy – w zależności od reprezentacji grafu

Reprezentacja – macierz sąsiedztwa

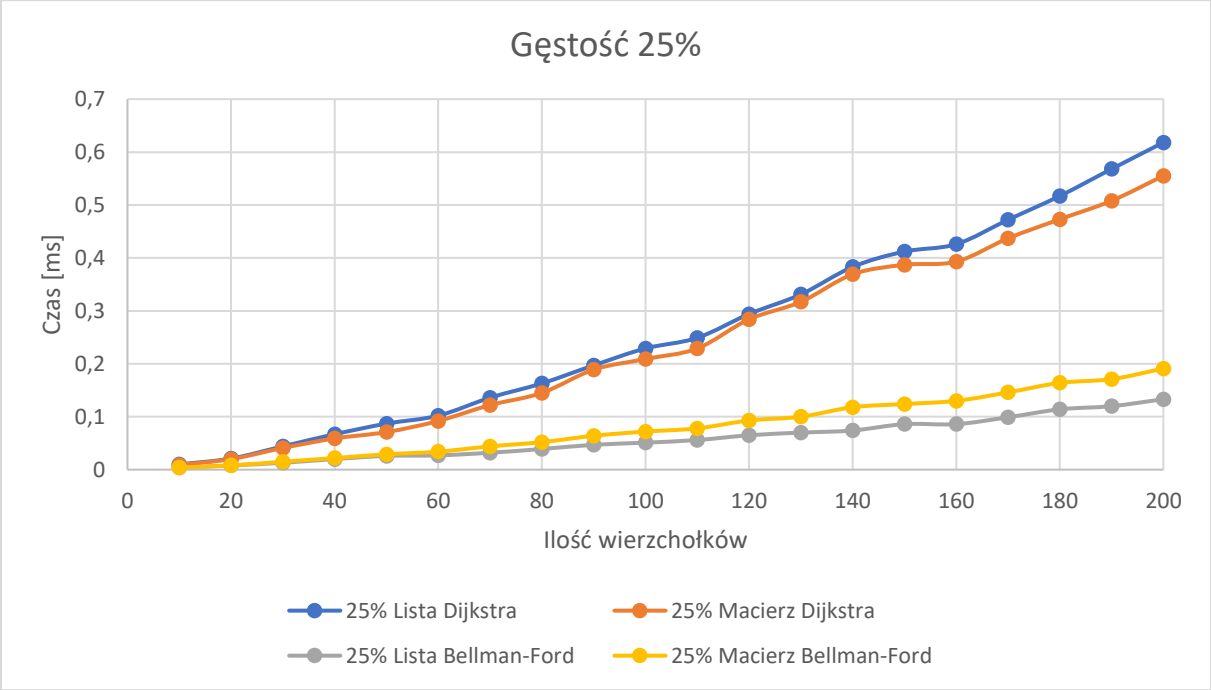


Reprezentacja – lista sąsiadów

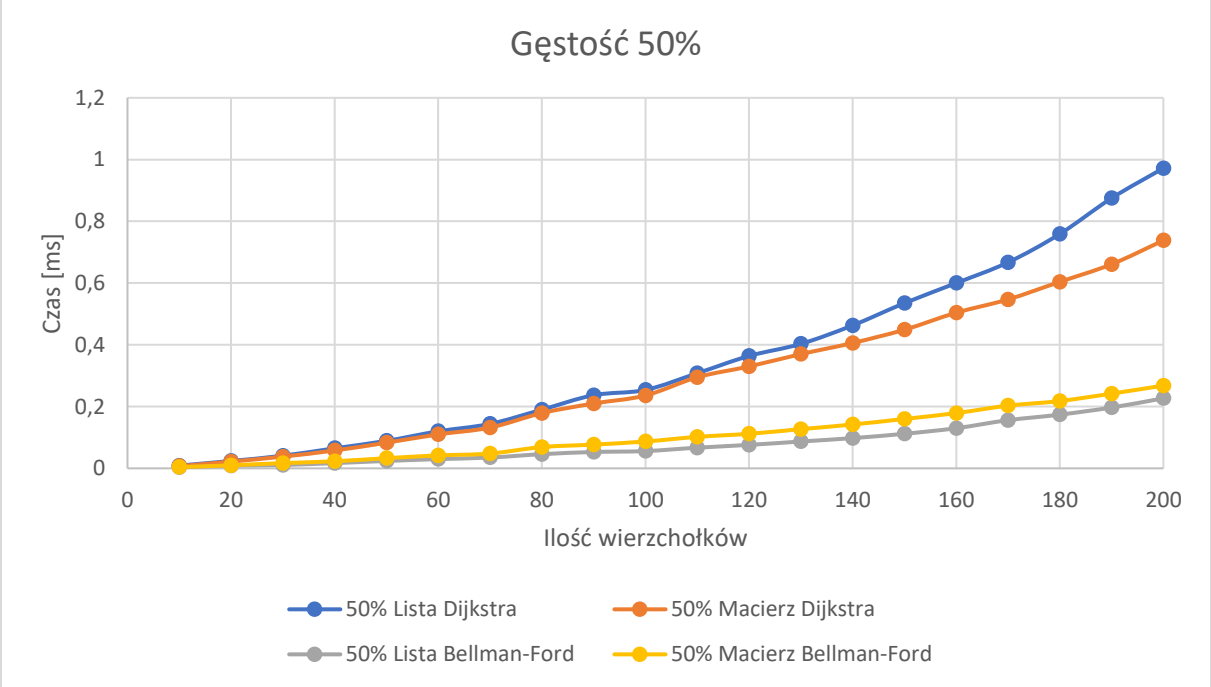


Wykresy – w zależności od gęstości grafu

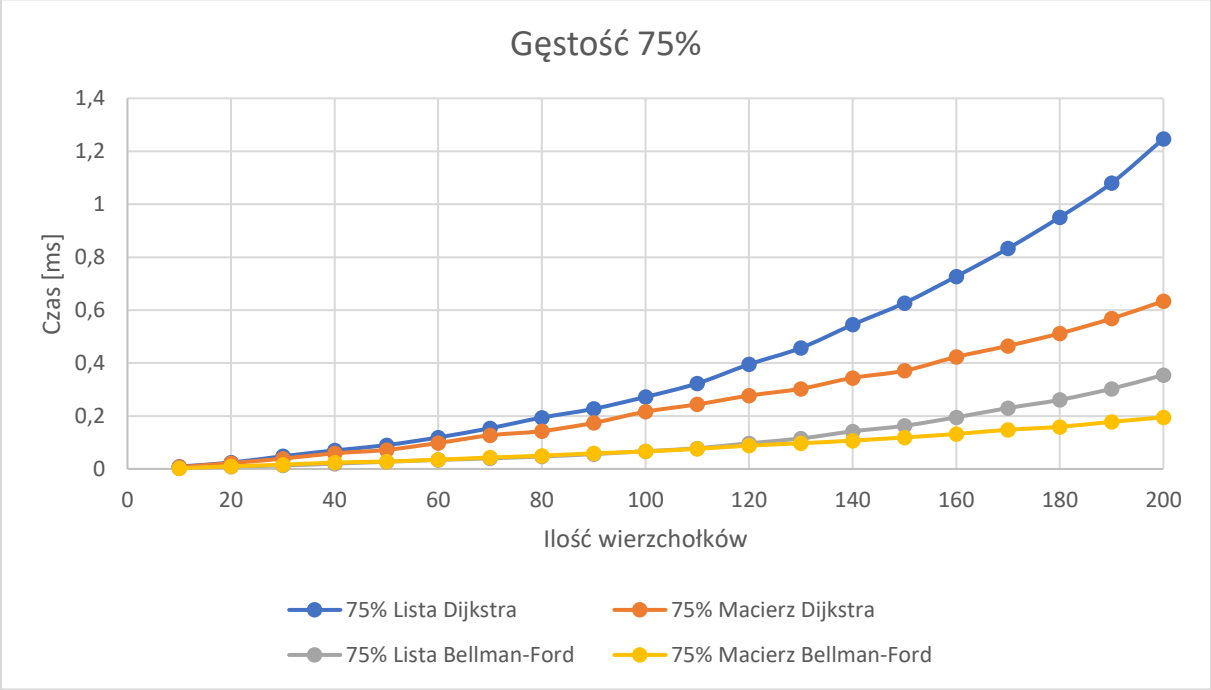
Gęstość - 25%



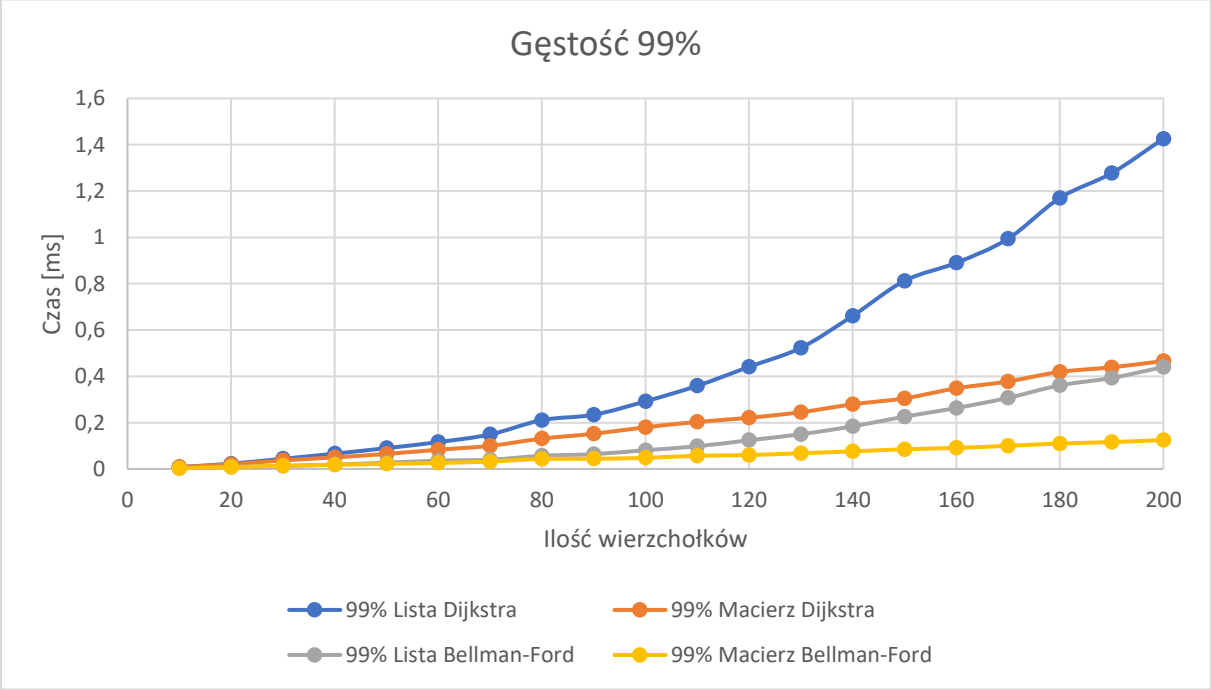
Gęstość - 50%



Gęstość - 75%



Gęstość - 99%



Podsumowanie

Jak można zauważyć na powyższych wykresach oba algorytmy znacznie lepiej współpracuje z reprezentacją macierzową. Może to być związane ze sposobem implementacji obu struktur. Dodatkowo po dokładnej analizie możemy zauważyć interesującą zależność, dokładnie analizując jak algorytmy zachowują się we współpracy z macierzą sąsiedztwa. Mianowicie najgorzej wypada gęstość

grafu 50%, a najlepiej 99%, natomiast 25% i 75% są pomiędzy nimi i mają zbliżone czasy. Związane to może być również ze sposobem implementacji obu reprezentacji oraz sposobem działania algorytmów.

Warto również zwrócić uwagę, iż algorytm Dijkstry wraz ze wzrostem liczby krawędzi w reprezentacji macierzowej grafu rośnie praktycznie liniowo, natomiast w reprezentacji listy sąsiadów ucieka silnie do góry. Tej zależności nie widać jedynie w grafach o małej gęstości. W algorytmie Bellmana-Forda również możemy zauważyć interesującą zależność dla gęstości <=50% reprezentacja grafu za pomocą listy sąsiadów jest bardziej wydajna niż macierzowa. Natomiast przy większych gęstościach dzieje się podobnie jak w algorytmie Dijkstry.

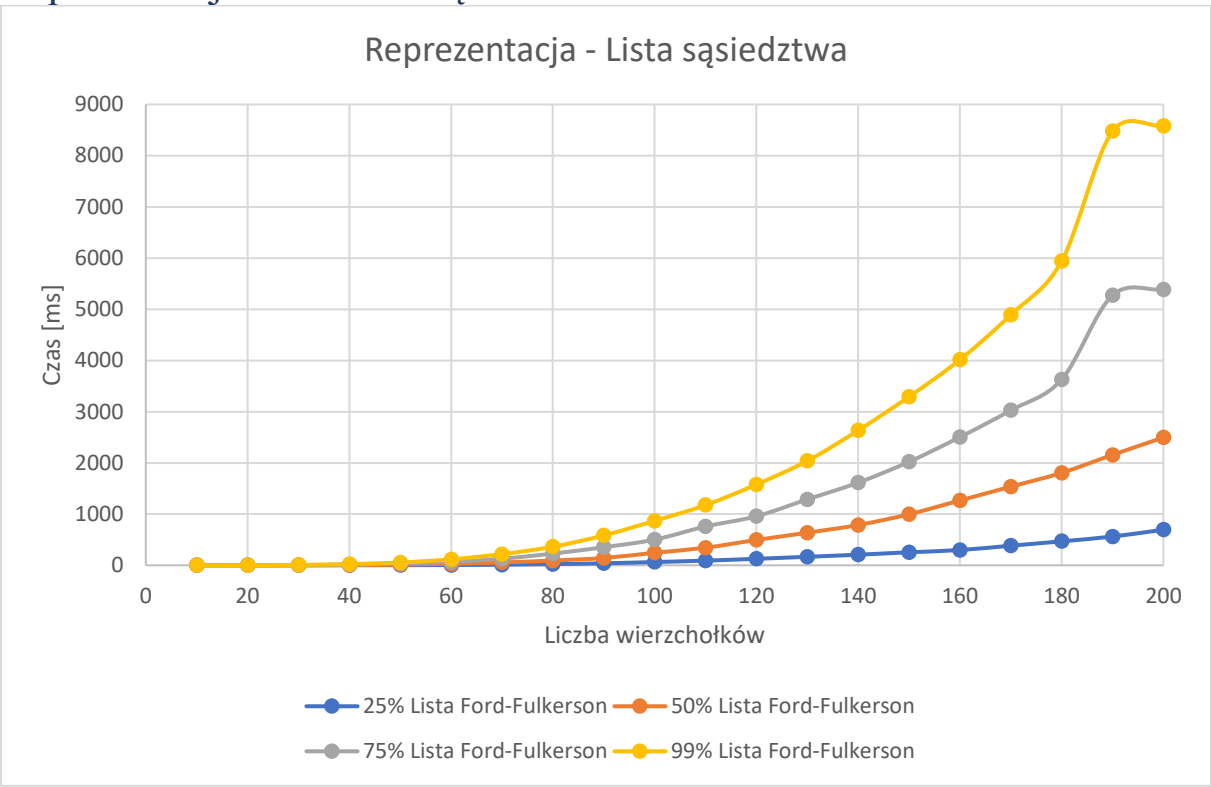
Wyznaczenie maksymalnego przepływu w grafie

Tabela czasów – Algorytm Forda-Fulkersona

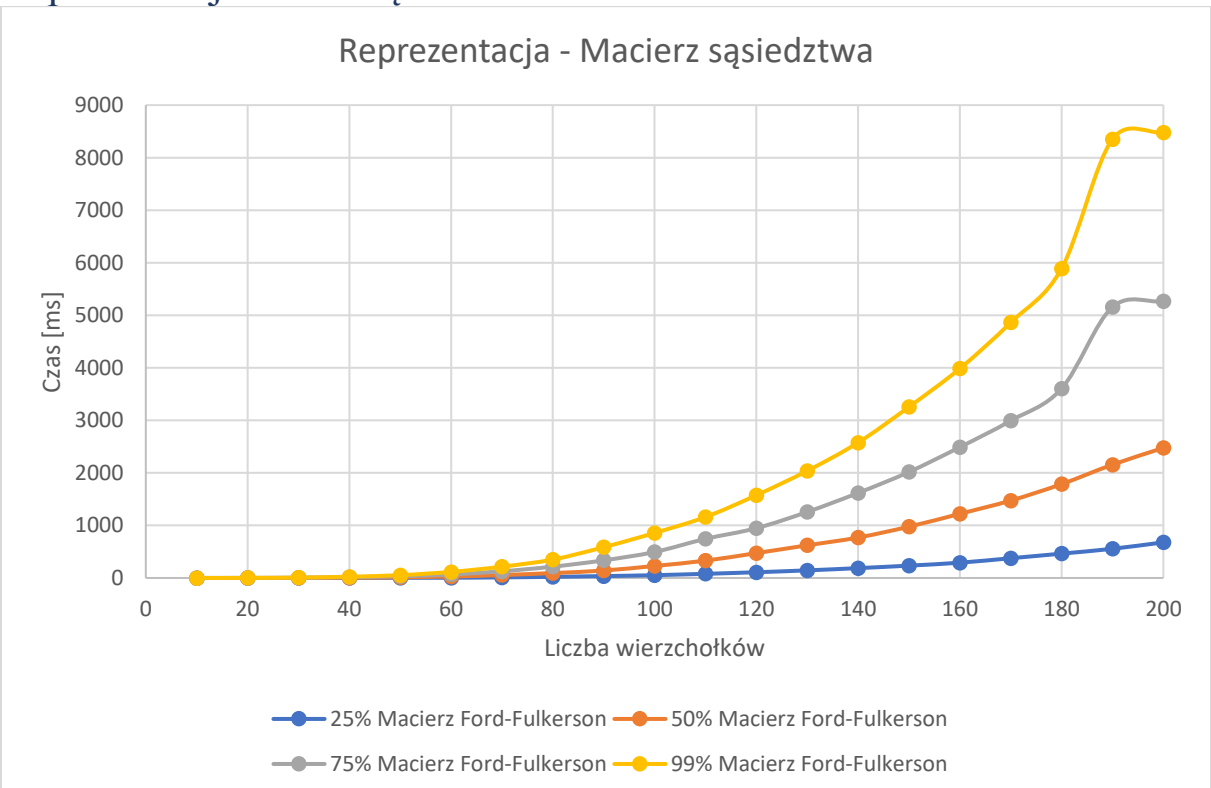
Ilość wierzchołków	Gęstość - 25%		Gęstość - 50%		Gęstość - 75%		Gęstość - 99%	
	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]	Lista [ms]	Macierz [ms]
10	0,004	0,004	0,020	0,015	0,041	0,041	0,076	0,077
20	0,061	0,059	0,380	0,338	0,810	0,731	1,500	1,403
30	0,358	0,319	1,236	1,184	4,896	4,785	7,311	6,999
40	1,622	1,584	5,404	5,159	15,189	15,021	24,258	24,274
50	3,551	3,463	13,929	13,578	34,273	34,069	54,204	53,599
60	5,280	4,882	28,895	28,072	67,127	67,500	117,890	115,700
70	12,082	11,187	56,293	55,464	128,966	127,707	218,243	214,796
80	23,012	21,930	93,411	93,929	229,240	215,395	362,789	351,141
90	39,928	36,210	143,367	141,767	353,591	335,737	583,076	585,372
100	63,254	53,096	245,026	229,277	501,192	494,564	868,371	857,215
110	91,825	79,430	343,318	330,187	757,607	744,243	1 175,485	1 159,796
120	126,744	109,033	495,309	473,321	957,552	947,420	1 578,332	1 575,073
130	166,583	144,966	636,095	621,686	1 287,020	1 256,791	2 038,225	2 039,339
140	206,859	187,479	786,542	769,688	1 614,039	1 618,308	2 633,482	2 576,240
150	254,565	235,175	996,755	976,640	2 020,579	2 017,597	3 292,451	3 256,248
160	298,485	291,006	1 265,873	1 221,434	2 502,868	2 489,877	4 013,942	3 986,146
170	382,994	376,140	1 535,651	1 473,550	3 026,676	2 993,792	4 891,159	4 862,440
180	468,533	463,774	1 805,150	1 786,988	3 627,051	3 603,924	5 938,700	5 889,564
190	561,995	555,657	2 156,422	2 156,922	5 271,596	5 155,965	8 476,817	8 353,110
200	694,911	677,967	2 495,705	2 475,036	5 387,220	5 265,384	8 579,536	8 479,732

Wykresy – w zależności od reprezentacji grafu

Reprezentacja – macierz sąsiedztwa

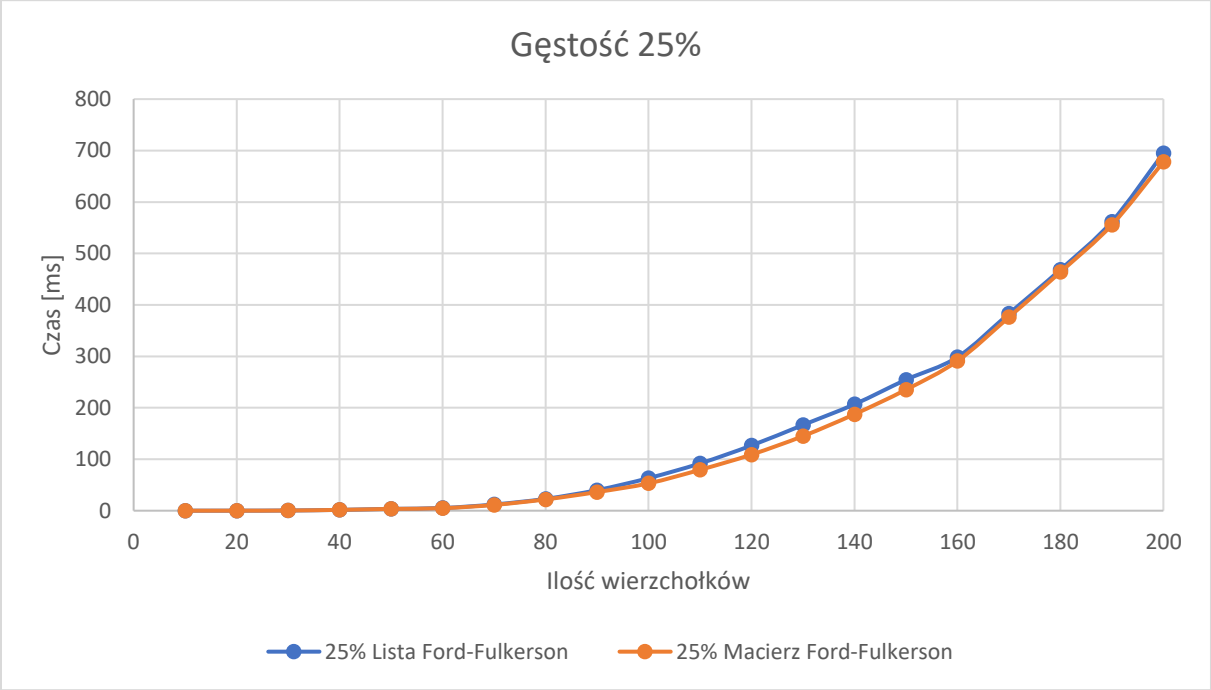


Reprezentacja – lista sąsiadów

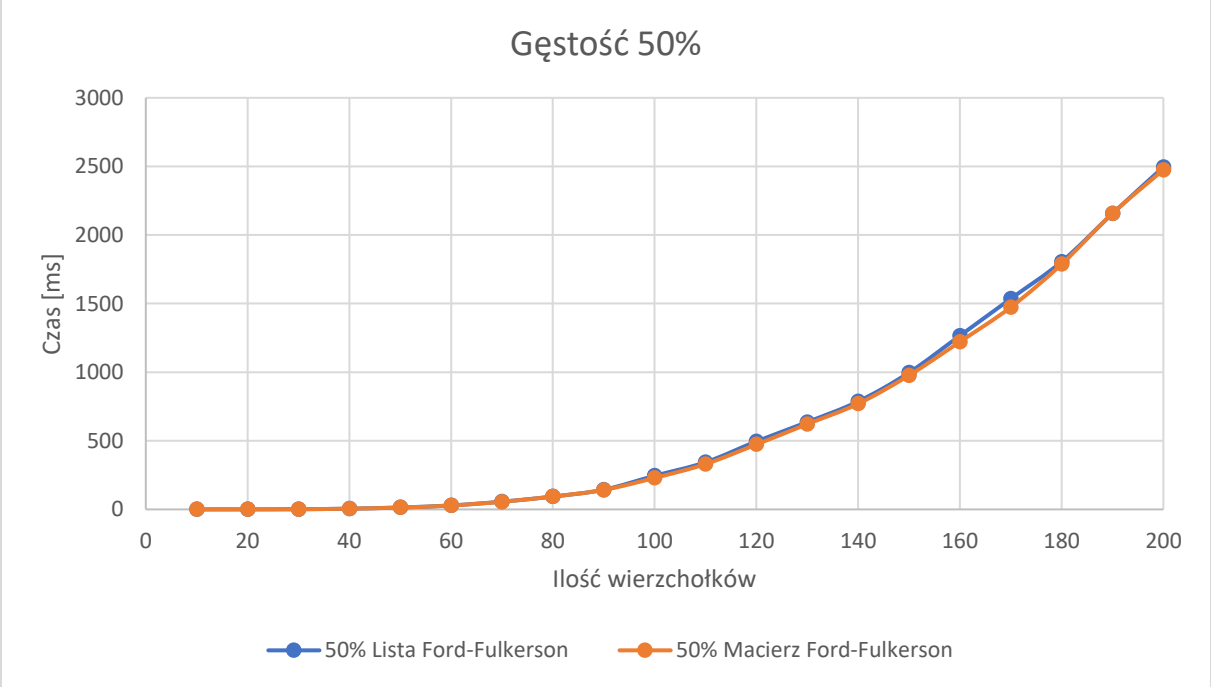


Wykresy – w zależności od gęstości grafu

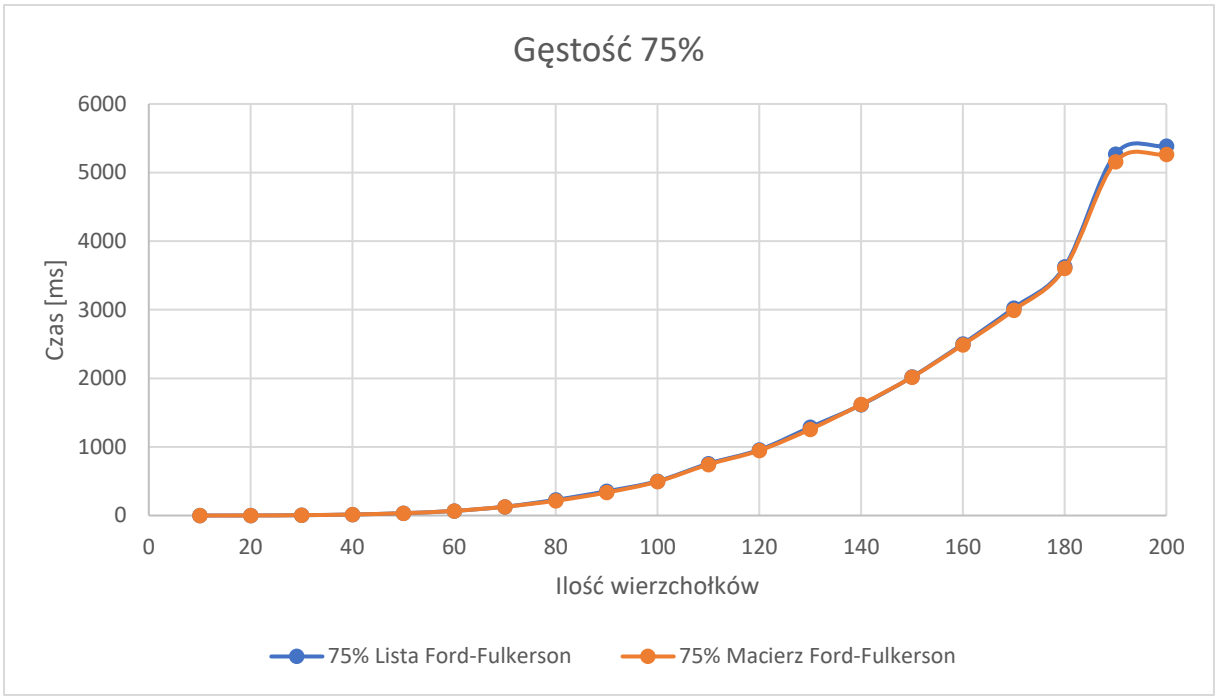
Gęstość - 25%



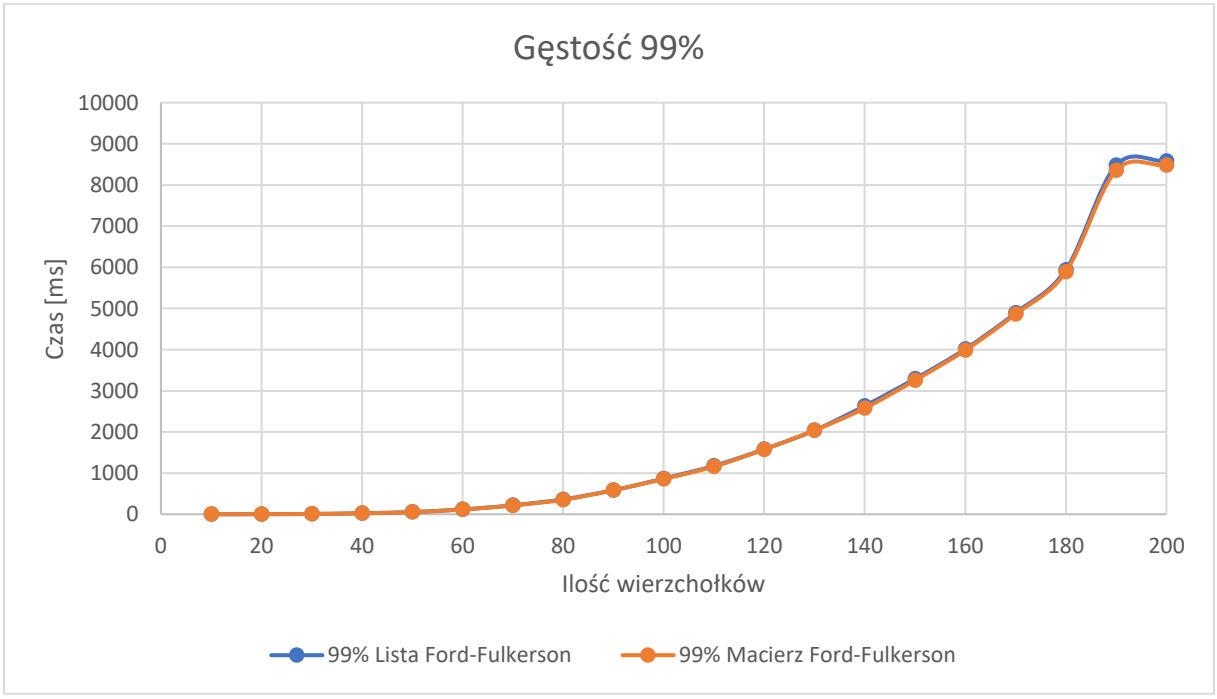
Gęstość - 50%



Gęstość - 75%


Gęstość 75%
Czas [ms]
6000
5000
4000
3000
2000
1000
0
0
20
40
60
80
100
120
140
160
180
200
Ilość wierzchołków
75% Lista Ford-Fulkerson
75% Macierz Ford-Fulkerson

Gęstość - 99%


Gęstość 99%
Czas [ms]
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
20
40
60
80
100
120
140
160
180
200
Ilość wierzchołków
99% Lista Ford-Fulkerson
99% Macierz Ford-Fulkerson

Podsumowanie

W algorytmie Forda-Fulkersona nie jesteśmy w stanie zobaczyć różnicy pomiędzy obiema reprezentacjami. Związane jest to z tym, że w trakcie trwania algorytmu nie korzystamy z nich. Na początku przenosimy dane do grafu rezydualnego, aby nie zniszczyć oryginalnych danych.

Warto jednak zauważyć, że dla większych gęstości, przy większych grafach występuje spłaszczenie wykresu. Związane jest to zapewne z wykorzystaniem tego grafu, w którym po użyciu krawędzi zmniejszamy jej przepływ i jednocześnie pozbywamy się niektórych dróg. a ogólnie wiadome jest to, iż maksymalnie możemy mieć V-1 dróg wychodzących z wierzchołka. Dlatego przy tak dużym wypełnieniu bardzo prawdopodobne jest, iż część tych dróg będzie bardzo krótkich przechodzących przez kilka wierzchołków.

Bibliografia

1. „Wprowadzenie do algorytmów” Clifford Stein, Ron Rivest i Thomas H. Cormen
2. http://jaroslaw.mierzwa.staff.iiar.pwr.edu.pl/sdizo/
3. https://en.cppreference.com/w/cpp/chrono/parse
4. https://eduinf.waw.pl/inf/alg/001_search/0146.php
5. http://www.algorytm.org/