Sprawozdanie
Struktury danych i złożoność obliczeniowa

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

Politechnika Wrocławska
Dawid Szeląg 264008
Prowadzący: Mgr. inż. Antoni Sterna
05.06.2023r.

Politechnika
Wrocławska

Spis treści

1 Wstęp teoretyczny

Zadanie projektowe polegało na zaimplementowaniu oraz dokonaniu pomiaru czasu działania algorytmów takich jak:

- algorytm Prima oraz algorytm Kruskala - wyznaczanie minimalnego drzewa rozpinającego (MST),
- algorytm Dijkstry oraz algorytm Bellmana-Forda - wyznaczanie najkrótszej ścieżki w grafie,

dla następujących reprezentacji grafów:

- reprezentacja macierzowa (macierz wag - macierz sąsiedztwa w której zamiast wartości 0/1 wpisane są wagi krawędzi)
- reprezentacja listowa (połączone listy sąsiadów).

Przyjęte założenia w projekcie:

- waga (przepustowość) krawędzi jest liczbą całkowitą;
- wszystkie struktury danych są alokowane dynamicznie;
- graf wejściowy jest spójny;
- dane do testów są wygenerowane losowo;
- pomiary zależności czasu wykonywania algorytmów, wykonywano **100 razy**, następnie wyliczano z tego średnią arytmetyczną dla następujących gęstości grafu oraz ilości wierzchołków:

Ilości wierzchołków

1. 10	**3**. 40	**5**. 80	**7**. 150
2. 20	**4**. 60	**6**. 100	**8**. 200

Gęstości

1. 25%	**2**. 50%	**3**. 75%	**4**. 99%

1.1 Sposoby generowania liczb oraz liczenie czasu

Generowanie liczb pseudolosowych do wag krawędzi oraz losowania wierzchołków zostało zrealizowane za pomocą generatora mt19937 z biblioteki <random>.

```
//Genarator.h
#include <random>
#include <iostream>
using namespace std;

class Generator {
    random_device rd;
    mt19937 gen;
    uniform_int_distribution<> dist;
public:
    Generator();
    int getNumber();
    int getNumber(int min, int max);
};

//Genarator.cpp
Generator::Generator():rd(),gen(rd()), dist(0, INT32_MAX)
{}
int Generator::getNumber() {
    return dist(gen);
}
int Generator::getNumber(int min, int max) {
    dist.param(uniform_int_distribution<>::param_type(min,max));
    int valReturn = dist(gen);
    dist.param(uniform_int_distribution<>::param_type(INT32_MIN,INT32_MAX));
    return valReturn;}
}
```

Czas liczony był za pomocą funkcji QueryPerformanceCounter().

```
#include <windows.h>
#include <iostream>
#include <iomanip>
using namespace std;

Timer::Timer() : frequency(0), start(0), elapsed(0)
{
}

void Timer::run() {
    QueryPerformanceFrequency((LARGE_INTEGER *)&frequency);
    start = read_QPC();
}

void Timer::stop() {
    elapsed = read_QPC() - start;
}

long long int Timer::read_QPC()
{
    LARGE_INTEGER count;
    DWORD_PTR oldmask = SetThreadAffinityMask(GetCurrentThread(),
        0);
    QueryPerformanceCounter(&count);
    SetThreadAffinityMask(GetCurrentThread(), oldmask);
    return((long long int)count.QuadPart);
}

double Timer::getTimeMs()
{
    return (1000.0 * elapsed) /frequency;
}
```

2 Minimalne drzewo rozpinające

2.1 Algorytm Prima

2.1.1 Złożoność według literatury

Do sortowania krawedzi został wykorzystany kopiec typu minimum, co daje nam dzięki temu złożoność: $O(Elog(V))$.

2.1.2 Wyniki

Wyniki czasowe podane są w mikrosekundach

Macierz sąsiedztwa

Liczba wierzchołków	Prima 25%	Prima 50%	Prima 75%	Prima 99%
10	2,413	2,76	2,784	2,696
20	5,691	6,288	6,751	7,086
40	14,936	20,824	24,875	24,616
60	30,862	44,016	49,943	51,961
80	55,406	78,159	88,458	93,371
100	78,68	118,658	140,015	145,907
150	169,891	281,476	327,148	355,654
200	308,923	524,319	692,562	803,729

Rysunek 1: Wyniki - Macierz/Prima

Lista sąsiedztwa

Liczba wierzchołków	Prima 25%	Prima 50%	Prima 75%	Prima 99%
10	2,042	2,376	3,165	2,879
20	4,713	5,807	7,588	8,512
40	12,32	18,585	25,723	31,085
60	24,668	39,64	60,203	76,481
80	41,229	84,548	123,535	190,861
100	80,488	140,232	219,992	287,89
150	166,303	372,952	632,682	856,62
200	333,368	716,329	1442,79	1936,31

Rysunek 2: Wyniki - Lista/Prima

2.2 Algorytm Kruskala

2.2.1 Złożoność według literatury

Sortowanie krawedzi odbywa się poprzez kopcowanie, oraz wykorzystujemy zbiory rozłączne do wyszukiwania cyklów, co daje nam złożoność: $O(Elog(E))$.

2.2.2 Wyniki

Wyniki czasowe podane są w mikrosekundach

Macierz sąsiedztwa

Liczba wierzchołków	Kruskal 25%	Kruskal 50%	Kruskal 75%	Kruskal 99%
10	2,809	2,671	2,702	2,878
20	4,974	6,254	6,058	6,764
40	12,818	16,834	18,391	20,816
60	23,559	33,723	36,139	40,477
80	37,565	54,002	64,45	71,848
100	56,147	80,889	99,403	116,007
150	116,051	195,721	241,2	290,909
200	213,317	383,044	584,575	669,96

Rysunek 3: Wyniki - Macierz/Kruskal

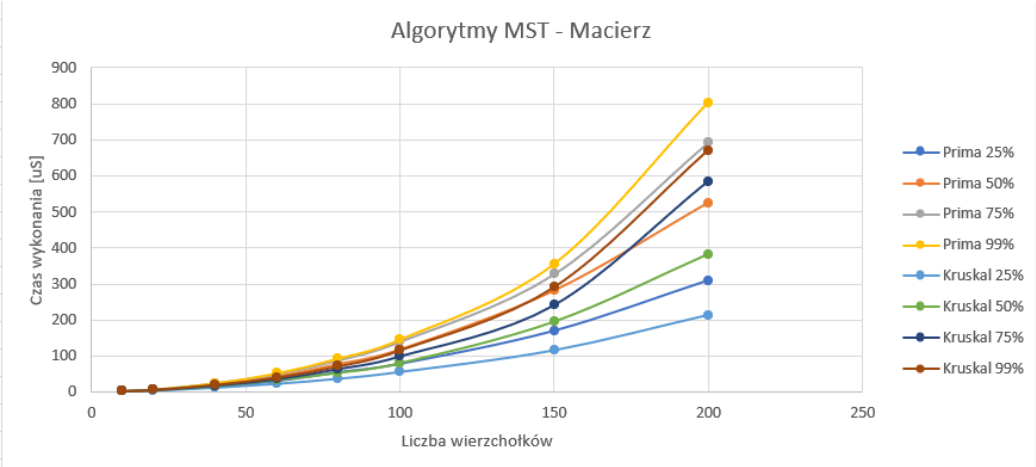
Lista sąsiedztwa

Liczba wierzchołków	Kruskal 25%	Kruskal 50%	Kruskal 75%	Kruskal 99%
10	3,49	3,809	4,578	4,328
20	7,303	9,668	12,155	13,748
40	22,265	30,551	41,692	48,932
60	43,973	68,826	90,899	117,374
80	73,131	124,928	182,173	244,382
100	137,98	236,924	342,115	441,852
150	368,029	640,752	887,913	1115,61
200	556,306	1036,8	1723,93	2298,16

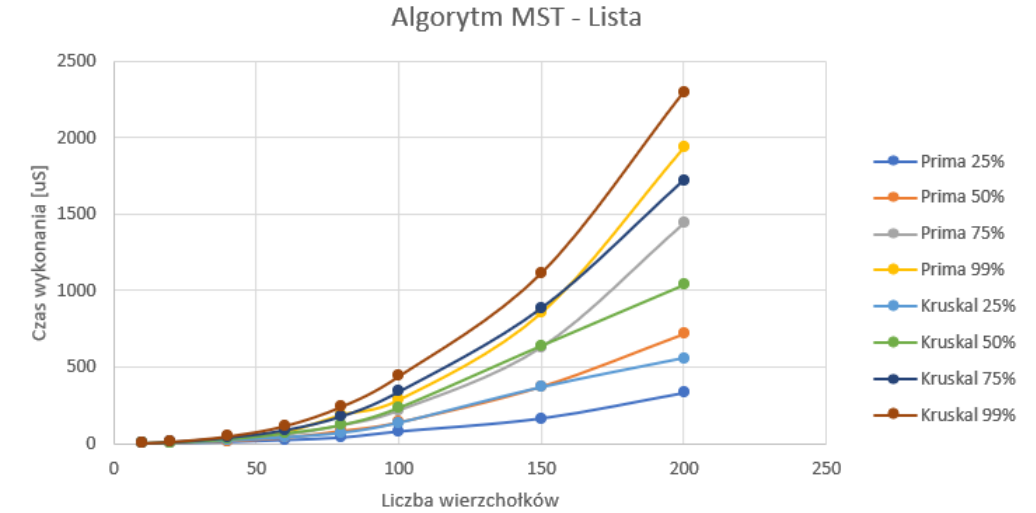
Rysunek 4: Wyniki - Lista/Kruskal

2.3 Wykresy

2.3.1 TYP 1

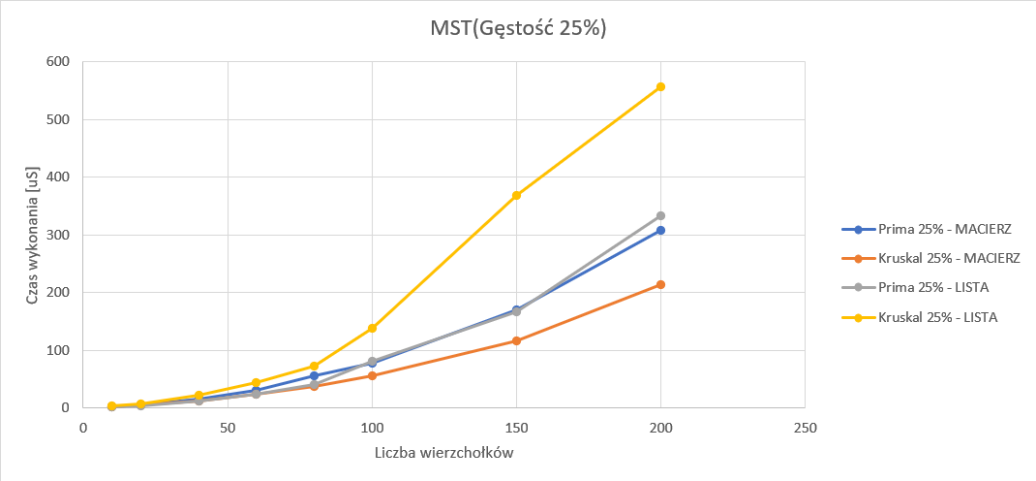


Rysunek 5: Wyniki - Macierz/MST/1

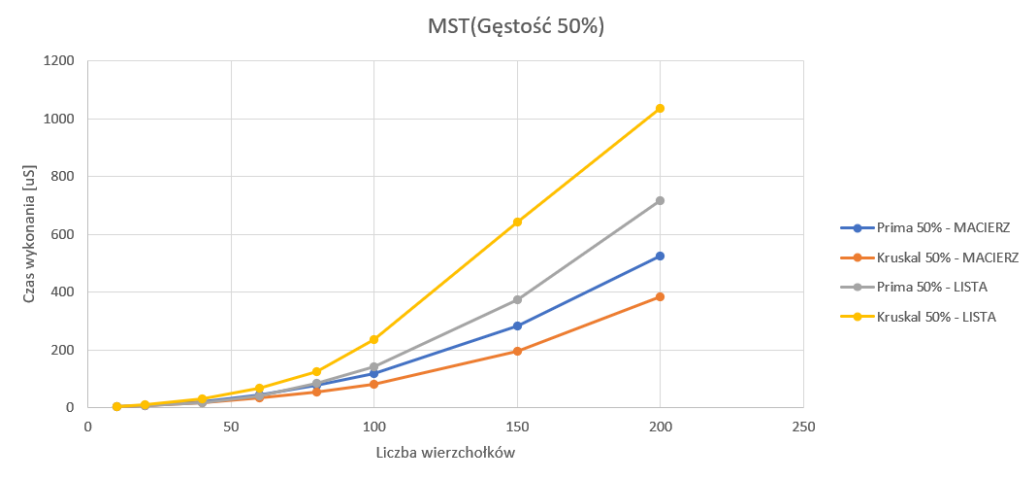


Rysunek 6: Wyniki - Lista/MST/1

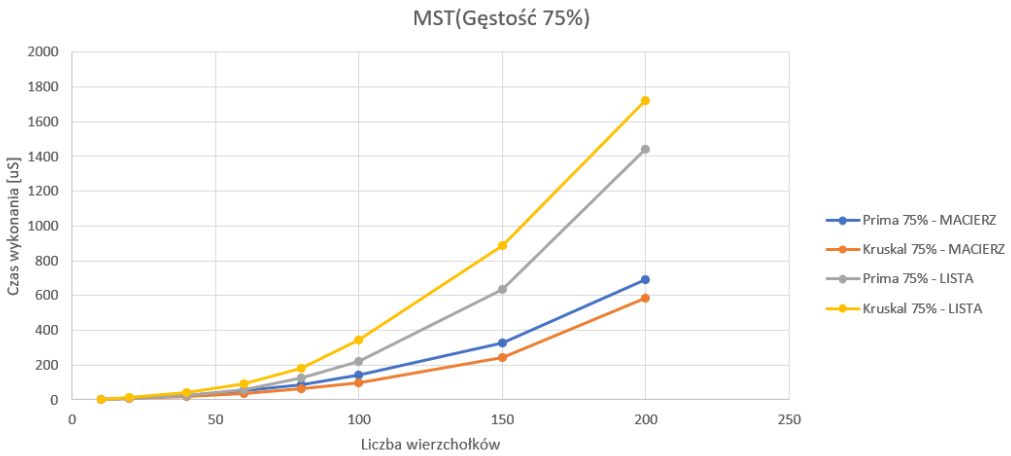
2.3.2 TYP 2



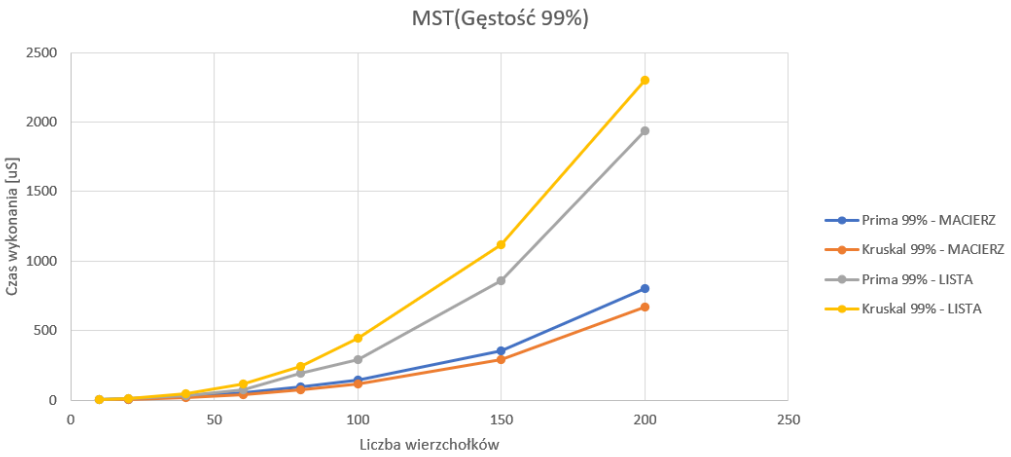
Rysunek 7: Wyniki - MST - 25%



Rysunek 8: Wyniki - MST - 50%



Rysunek 9: Wyniki - MST - 75%



Rysunek 10: Wyniki - MST - 99%

3 Wyznaczenie najkrótszej ścieżki

3.1 Algorytm Dijkstry

3.1.1 Złożoność według literatury

Do otrzymywania kolejnych nieodwiedzonych wierzchołków, o najmniejszym koszcie dotarcia wykorzystano kopiec typu minimum,stąd złożoność: $O((V+E)log(V))$. Wadą algorytmu jest możliwe błędne rozwiązywanie problemów dla ujemnych wag.

3.1.2 Wyniki

Wyniki czasowe podane są w mikrosekundach

Macierz sąsiedztwa

Liczba wierzchołków	Dijsktra 25%	Dijsktra 50%	Dijsktra 75%	Dijsktra 99%
10	2,92	3,264	3,018	2,704
20	5,435	6,784	6,862	6,321
40	13,116	18,487	18,291	16,914
60	24,215	34,421	35,864	32,215
80	39,627	61,411	59,39	65,563
100	58,161	83,804	83,508	87,087
150	119,624	199,926	275,664	320,693
200	231,021	431,471	566,934	710,995

Rysunek 11: Wyniki - Macierz/Dijkstra

Lista sąsiedztwa

Liczba wierzchołków	Dijsktra 25%	Dijsktra 50%	Dijsktra 75%	Dijsktra 99%
10	3,492	3,262	2,798	3,035
20	4,783	6,449	6,856	8,101
40	12,887	17,436	22,169	27,037
60	27,638	45,947	64,143	83,364
80	52,72	98,178	138,654	188,826
100	78,367	156,513	257,056	401,234
150	189,149	430,199	682,128	963,75
200	455,688	731,727	1249,04	1758,86

Rysunek 12: Wyniki - Lista/Dijkstra

3.2 Algorytm Bellmana-Forda

3.2.1 Złożoność według literatury

W najgorszym wypadku algorytm musi odwiedzić wszystkie krawędzie w grafie V razy (gdzie V jest liczbą wierzchołków), co daje nam złożoność: $O(VE)$. Zaletą algorytmu jest możliwosc wychwytywania ujemnych cyklów oraz rozwiązywanie problemów dla ujemnych wag (bez cyklu).

3.2.2 Wyniki

Wyniki czasowe podane są w mikrosekundach

Macierz sąsiedztwa

Liczba wierzchołków	Bellman-Ford 25%	Bellman-Ford 50%	Bellman-Ford 75%	Bellman-Ford 99%
10	4,636	5,324	6,049	6,567
20	8,555	12,801	14,935	18,643
40	37,988	35,701	57,853	79,371
60	45,888	90,827	133,992	207,439
80	90,901	172,316	277,185	427,784
100	171,647	340,058	569,259	780,384
150	434,468	886,108	1424,71	1985,24
200	896,348	1815,74	2864,16	5505,71

Rysunek 13: Wyniki - Macierz/Bellman-Ford

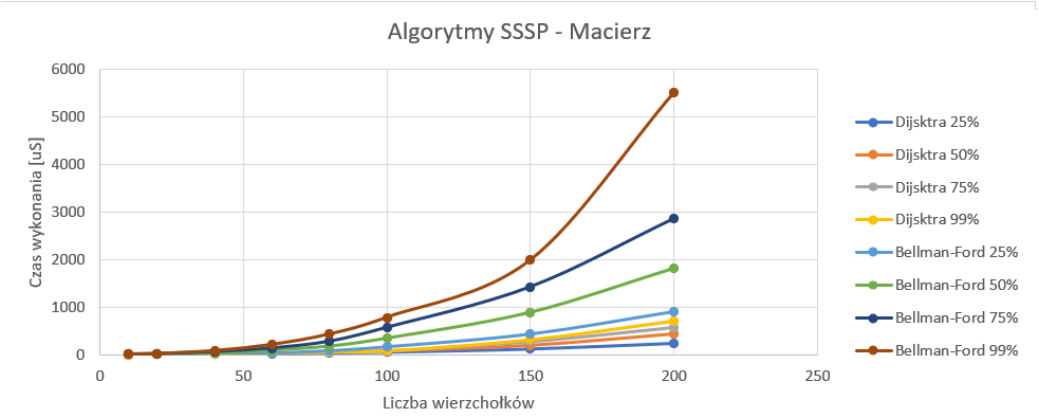
Lista sąsiedztwa

Liczba wierzchołków	Bellman-Ford 25%	Bellman-Ford 50%	Bellman-Ford 75%	Bellman-Ford 99%
10	4,68	5,769	6,92	8,394
20	9,609	15,701	17,727	25,063
40	44,319	40,905	70,882	106,94
60	56,349	130,721	236,306	425,568
80	180,003	343,278	570,948	848,533
100	260,786	670,136	1099,16	1386,56
150	752,176	1693,01	2496,81	3543,79
200	1597,32	3255,23	5135,67	7408,87

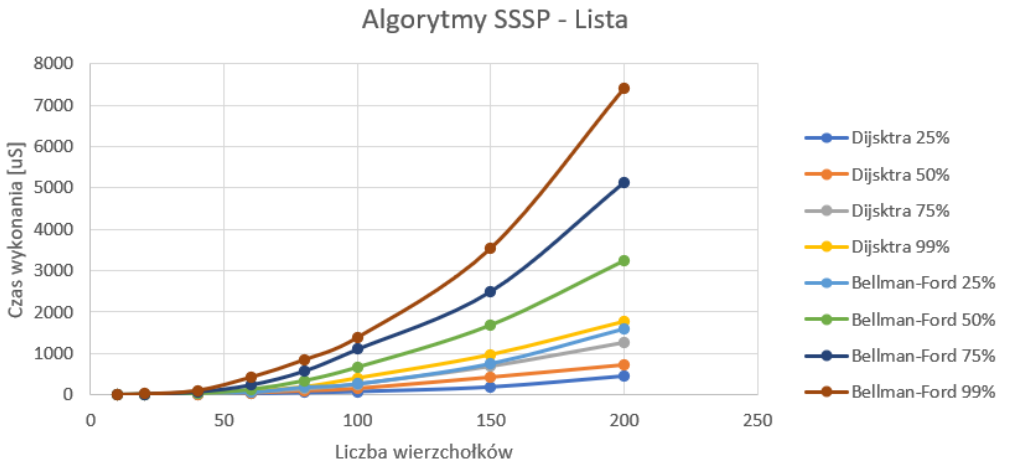
Rysunek 14: Wyniki - Lista/Bellman-Ford

3.3 Wykresy

3.3.1 TYP 1

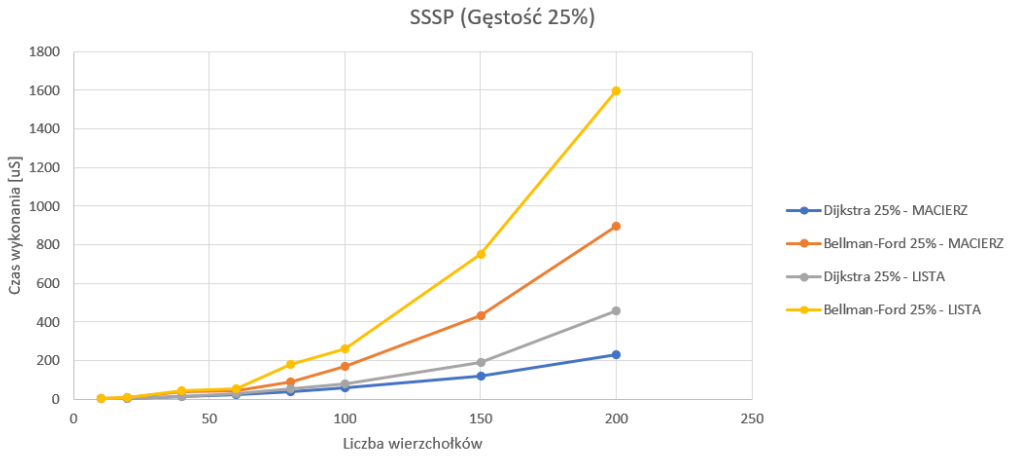


Rysunek 15: Wyniki - Macierz/SSSP/1

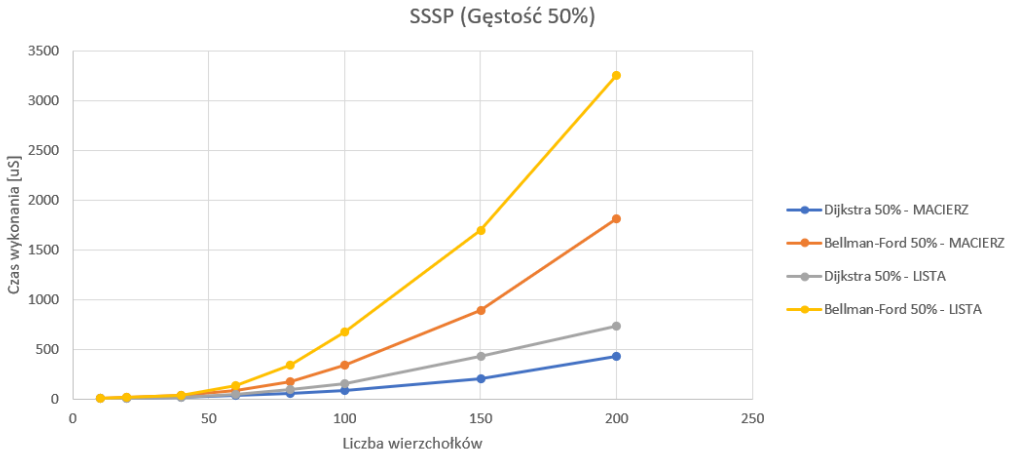


Rysunek 16: Wyniki - Lista/SSSP/1

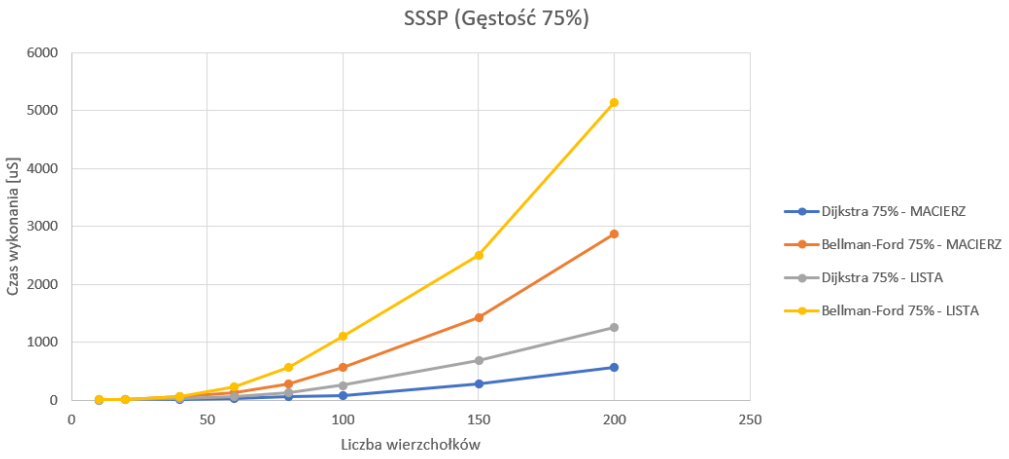
3.3.2 TYP 2



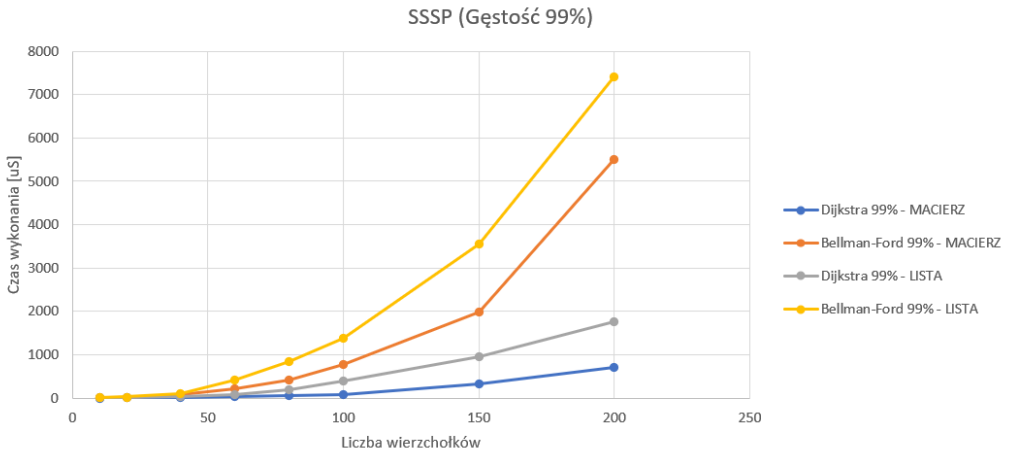
Rysunek 17: Wyniki - SSSP - 25%



Rysunek 18: Wyniki - SSSP - 50%



Rysunek 19: Wyniki - SSSP - 75%



Rysunek 20: Wyniki - SSSP - 99%

4 Podsumowanie oraz wnioski

Złożoności zgadzają się z teoretecznymi założeniami.

- Algorytm Kruskala był nieco wolniejszy dla reprezentacji listowej od algorytmu Primy.
- Dla algorytmów MST, reprezentacja listowa grafu sprawdziła się gorzej niż macierzowa.

- Dla algorytmów wyznaczania najkrótszych ścieżek, reprezentacja listowa grafu sprawdziła się podobnie dobrze jak macierzowa.
- Algorytm Bellman-Forda jest wolniejszy od algorytmu Diijsktry.
- Im bardziej gęsty graf, tym większa różnica pomiędzy czasami wykonania algorytmu Bellmana-Forda, a algorytmu Dijkstry.

  Wnioksujemy z tego, że dla MST lepiej wykorzystywać reprezentacje macierzową, a do wyznaczania najkrótszych ścieżek najlepiej wykorzystać algorytm Dijkstry (jeśli oczywiście nie mamy ujemnych krawędzi. Wówczas wtedy zostaje nam tylko Bellman-Ford).

5 Literatura

1. https://eduinf.waw.pl/inf/ - mgr Jerzy Wałaszek I LO w Tarnowie
2. https://visualgo.net
3. Wprowadzenie do algorytmów - Cormen Thomas H., Leiserson Charles E., Rivest Ronald L, Clifford Stein