|  |  |
| --- | --- |
| Jakub Majewski 238902 | Data oddania: 12.06.2018r.  Termin zajęć: wtorek TN godz. 13:15  Prowadzący zajęcia: Dr inż. Dariusz Banasiak |

**SPRAWOZDANIE**

**Struktury danych i złożoność obliczeniowa  
Projekt 2**

**I. Cel projektu.**

1. Zrealizować dwie reprezentacje grafów:

a) reprezentacja macierzowa

b) reprezentacja listowa

2. Zrealizować dwa algorytmy wyznaczania minimalnego drzewa rozpinającego (MST):

a) algorytm Prima

b) algorytm Kruskala

3. Zrealizować dwa algorytmy wyznaczania najkrótszej ścieżki w grafie:

a) algorytm Dijkstry

b) algorytm Bellmana-Forda

4. Zrealizować metodę generującą graf dowolnej, wcześniej wymienionej, reprezentacji.

5. Zmierzyć czasy realizacji powyższych algorytmów dla kilku wariantów grafów.

+ sporządzić odpowiednie wykresy

**II. Złożoność obliczeniowa poszczególnych problemów w literaturze.**

Oznaczenia:

E – liczba krawędzi w grafie

V – liczba wierzchołków w grafie

1. Złożoność algorytmu Prima:

Czasowa: **O(E \* logV)**

Czasowa: **O(E + V \* logV)** przy zastosowaniu kopca Fibonacciego

2. Złożoność algorytmu Kruskala:

Czasowa: **O(E + logV)**

3. Złożoność algorytmu Dijkstry:

Czasowa: **O(E + logV)**

Pamięciowa: **O(E + V \* logV)** przy zastosowaniu kopca Fibonacciego

4. Złożoność algorytmu Bellmana-Forda:

Czasowa: **O(V \* E)**

Pamięciowa: **O(V)**

**III. Plan eksperymentu.**

1. Wszystkie metody zewnętrzne przyjmują jako argumenty wartości dla wierzchołków od 1 w górę.

2. Graf wewnątrz indeksuje wierzchołki od 0.

3. Liczba wierzchołków w grafie jest równa największemu indeksowi wierzchołka.

4. Wszystkie algorytmy i programy zostały napisane w języku C++.

5. Do pomiaru czasu została użyta biblioteka standardowa <chrono>.

6. Wewnątrz algorytmów został wykorzystany wektor, którego maksymalna pojemność (capacity) jest zawsze równa aktualnej liczbie przechowywanych elementów. Wyjątkiem jest szybki wektor, który został wykorzystany w metodach generowania grafów, a jego maksymalny rozmiar po osiągnięciu limitu zwiększa się zawsze dwukrotnie.

7. Dla wygody w pisaniu kodu, wszystkie metody, napisanych klas, zostały umieszczone w plikach nagłówkowych.

8. Algorytm generowania grafu generuje tylko grafy spójne, nieskierowane. Wynika to z faktu, że niektóre algorytmy nie działają dla grafów skierowanych.

**IV. Algorytm generowania grafu.**

Do generowania grafów został wykorzystany następujący algorytm:

Argumenty:

V - liczba wierzchołków w generowanym grafie.

max - krawędzie będą miały generowane losowo wagi całkowitoliczbowe z przedziału <1, max>.

density - gęstość generowanego grafu

void generateDoubleGraph(V, max, density):  
1. Wyczyść graf

2. Stwórz 2\*(V-1) krawędzi:

1<->2, 2<->3, 3<->4 ... V-1<->V (bez łączenia V<->1)

3. Stwórz tablicę pairs zawierającą pary liczb całkowitych reprezentujące przyszłe możliwe krawędzie. Pary mają następujące wartości:

(1,3)(1,4) ... (1,V)(2,4)(2,5) ... (2,V) (3,1) ... (3,V) ... (V-2, V)

4. Oblicz maksymalną liczbę krawędzi w grafie nieskierowanym Emax = V\*(V-1)/2

5. Oblicz liczbę dodatkowych krawędzi E tak, aby po ich dodaniu gęstość grafu zgadzała się z density.

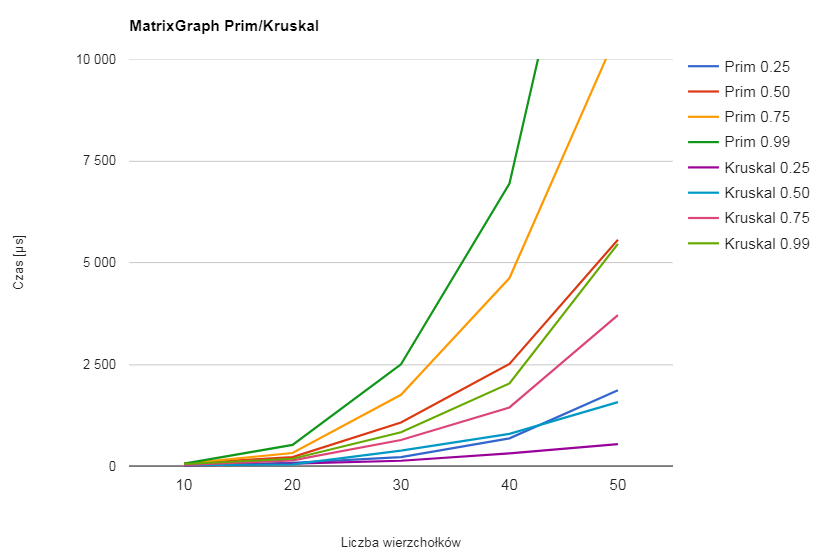
E = density\*Emax-(V-1)

6. Jeżeli E > 0:

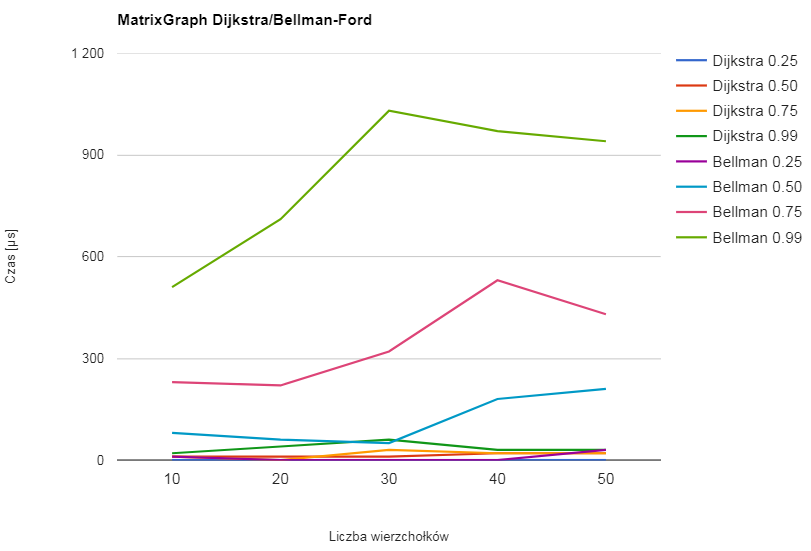
1. Przetasuj tablicę pairs Emax razy.

2. Dodaj pierwsze E krawędzi do grafu.

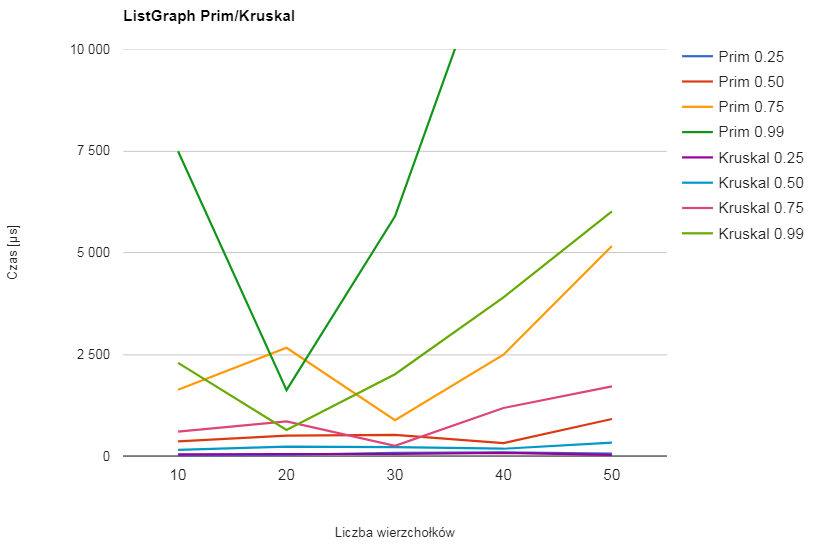
**IV. Wykresy**

****

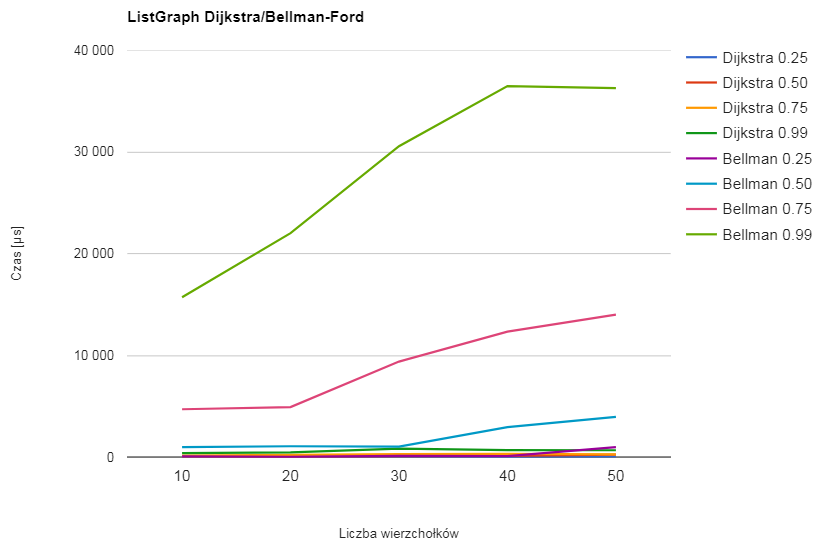
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Liczba  wierz. | Prim 25% [µs] | Prim 50% [µs] | Prim 75% [µs] | Prim 99% [µs] | Kruskal 25% [µs] | Kruskal 50% [µs] | Kruskal 75% [µs] | Kruskal 99% [µs] |
| 10 | 20 | 60 | 60 | 60 | 10 | 20 | 30 | 50 |
| 20 | 80 | 220 | 320 | 520 | 60 | 40 | 140 | 180 |
| 30 | 220 | 1070 | 1750 | 2500 | 130 | 380 | 640 | 830 |
| 40 | 680 | 2510.01 | 4620 | 6940.01 | 310 | 790 | 1440 | 2030 |
| 50 | 1860.03 | 5560.04 | 10670 | 18560.9 | 540 | 1570 | 3710.04 | 5460.04 |



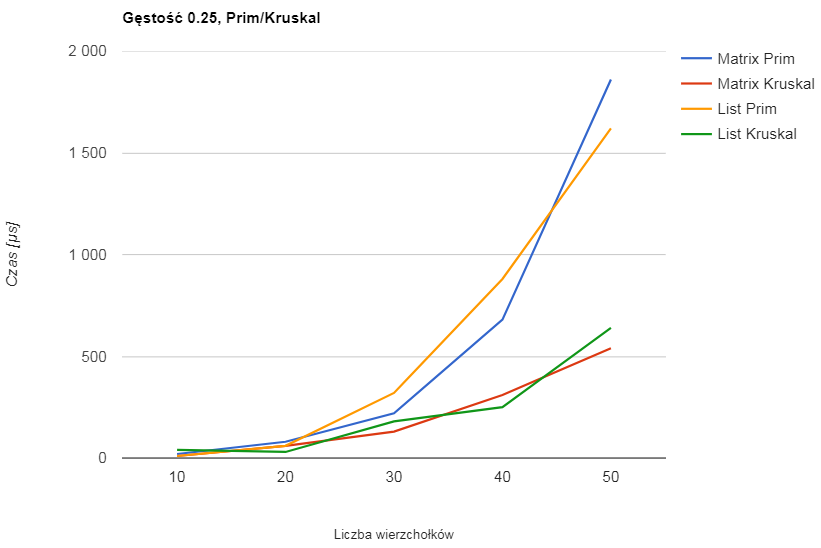
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Liczba  wierz. | Dijkstra 25% [µs] | Dijkstra 50% [µs] | Dijkstra 75% [µs] | Dijkstra 99% [µs] | Bellman 25% [µs] | Bellman 50% [µs] | Bellman 75% [µs] | Bellman 99% [µs] |
| 10 | 0 | 10 | 10 | 20 | 10 | 80 | 230 | 510 |
| 20 | 0 | 10 | 0 | 40 | 0 | 60 | 220 | 710 |
| 30 | 0 | 10 | 30 | 60 | 0 | 50 | 320 | 1030 |
| 40 | 0 | 20 | 20 | 30 | 0 | 180 | 530 | 970 |
| 50 | 0 | 20 | 20 | 30 | 30 | 210 | 430 | 940 |



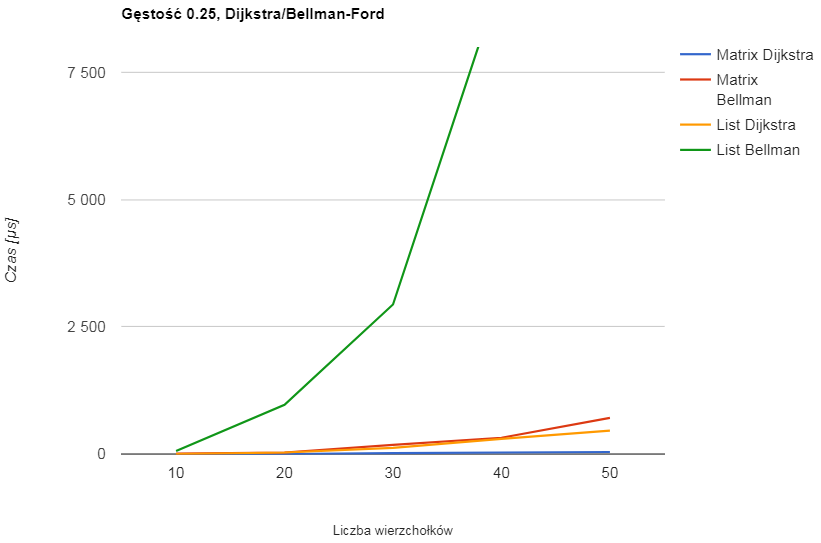
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Liczba  wierz. | Prim 25% [µs] | Prim 50% [µs] | Prim 75% [µs] | Prim 99% [µs] | Kruskal 25% [µs] | Kruskal 50% [µs] | Kruskal 75% [µs] | Kruskal 99% [µs] |
| 10 | 10 | 360 | 1630 | 7490 | 40 | 150 | 600 | 2290 |
| 20 | 30 | 500 | 2660 | 1620 | 50 | 230 | 850 | 640 |
| 30 | 80 | 520 | 880 | 5890 | 50 | 220 | 250 | 2010 |
| 40 | 90 | 320 | 2490 | 13330.1 | 80 | 180 | 1180.01 | 3900 |
| 50 | 60 | 910 | 5160 | 18571 | 30 | 330 | 1710 | 6010.04 |



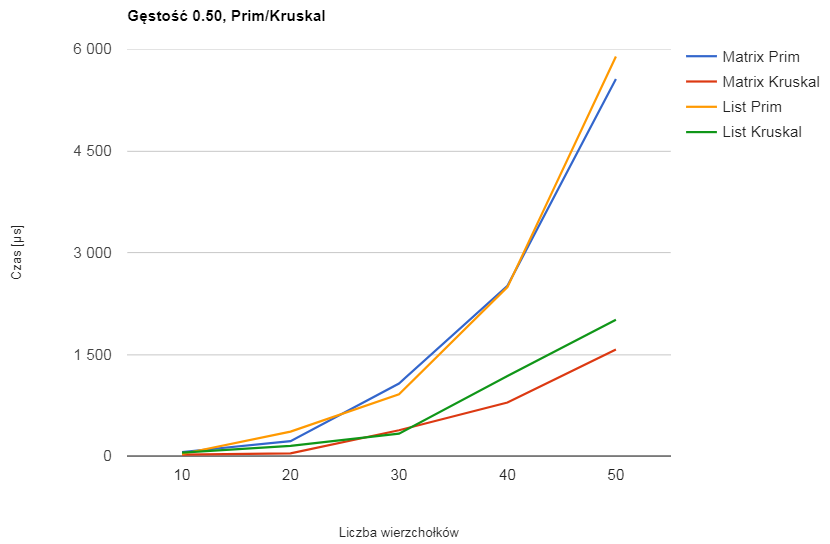
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Liczba  wierz. | Dijkstra 25% [µs] | Dijkstra 50% [µs] | Dijkstra 75% [µs] | Dijkstra 99% [µs] | Bellman 25% [µs] | Bellman 50% [µs] | Bellman 75% [µs] | Bellman 99% [µs] |
| 10 | 0 | 40 | 140 | 390 | 60 | 970 | 4700 | 15710.1 |
| 20 | 10 | 50 | 210 | 460 | 10 | 1060 | 4910 | 22001 |
| 30 | 20 | 60 | 300 | 830 | 110 | 1030 | 9380.04 | 30551.1 |
| 40 | 0 | 120 | 310 | 690 | 100 | 2940 | 12330 | 36461.6 |
| 50 | 30 | 230 | 270 | 670 | 970 | 3950 | 13990 | 36251.7 |



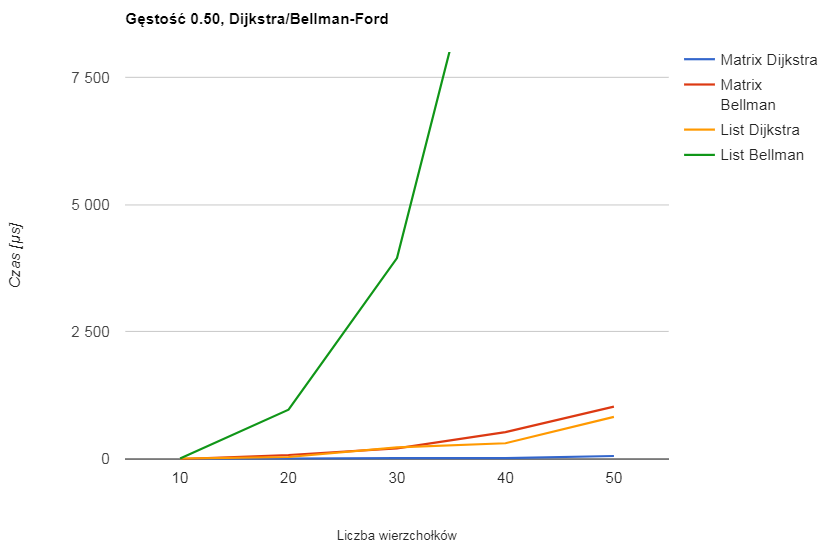
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba wierz. | Matrix Prim [µs] | Matrix Kruskal [µs] | List Prim [µs] | List Kruskal [µs] |
| 10 | 20 | 10 | 10 | 40 |
| 20 | 80 | 60 | 60 | 30 |
| 30 | 220 | 130 | 320 | 180 |
| 40 | 680 | 310 | 880 | 250 |
| 50 | 1860.03 | 540 | 1620 | 640 |



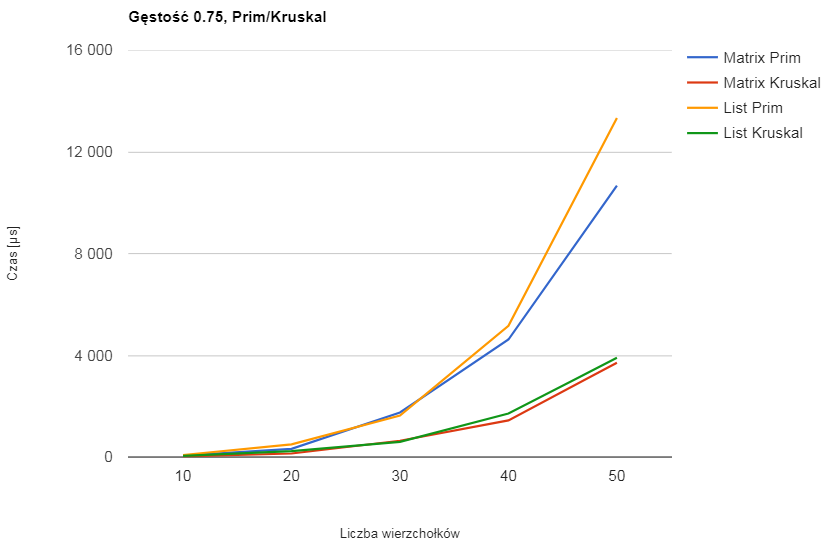
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba wierz. | Matrix Dijkstra [µs] | Matrix Bellman [µs] | List Dijkstra [µs] | List Bellman [µs] |
| 10 | 20 | 10 | 10 | 40 |
| 20 | 80 | 60 | 60 | 30 |
| 30 | 220 | 130 | 320 | 180 |
| 40 | 680 | 310 | 880 | 250 |
| 50 | 1860.03 | 540 | 1620 | 640 |



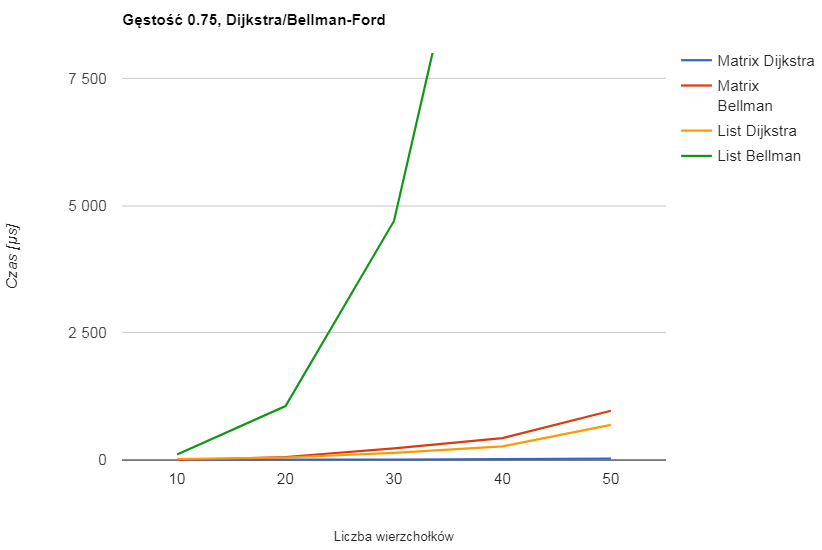
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba wierz. | Matrix Prim [µs] | Matrix Kruskal [µs] | List Prim [µs] | List Kruskal [µs] |
| 10 | 60 | 20 | 30 | 50 |
| 20 | 220 | 40 | 360 | 150 |
| 30 | 1070 | 380 | 910 | 330 |
| 40 | 2510.01 | 790 | 2490 | 1180 |
| 50 | 5560.04 | 1570 | 5890 | 2010 |



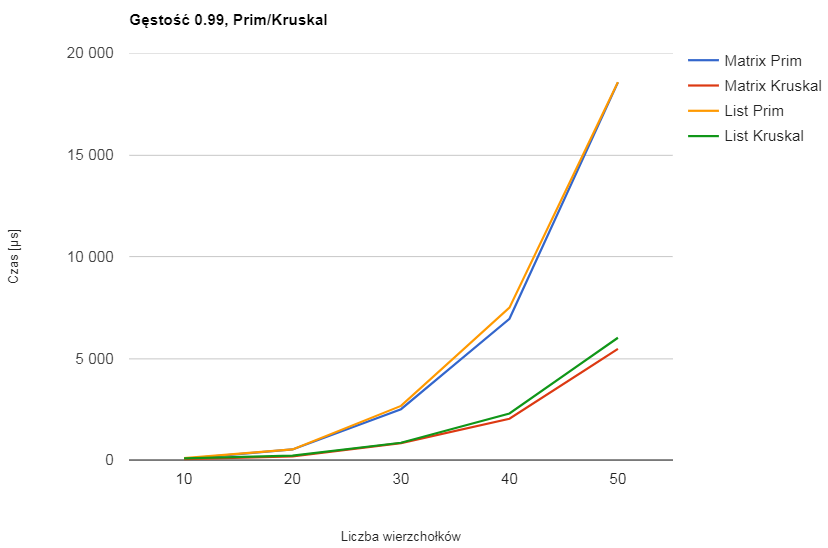
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba wierz. | Matrix Dijkstra [µs] | Matrix Bellman [µs] | List Dijkstra [µs] | List Bellman [µs] |
| 10 | 0 | 0 | 10 | 10 |
| 20 | 10 | 80 | 40 | 970 |
| 30 | 20 | 210 | 230 | 3950 |
| 40 | 20 | 530 | 310 | 12330 |
| 50 | 60 | 1030 | 830 | 30551.1 |



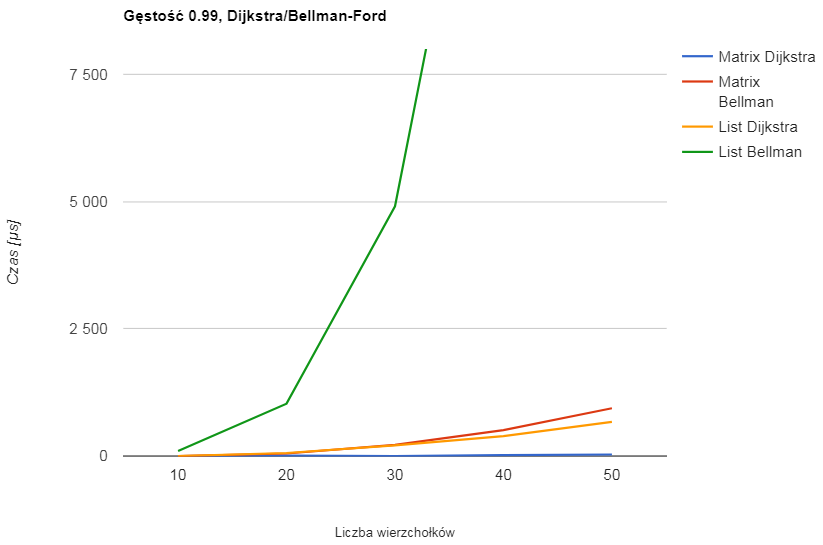
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba wierz. | Matrix Prim [µs] | Matrix Kruskal [µs] | List Prim [µs] | List Kruskal [µs] |
| 10 | 60 | 30 | 80 | 50 |
| 20 | 320 | 140 | 500 | 230 |
| 30 | 1750 | 640 | 1630 | 600 |
| 40 | 4620 | 1440 | 5160 | 1710 |
| 50 | 10670 | 3710.04 | 13330.1 | 3900 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba wierz. | Matrix Dijkstra [µs] | Matrix Bellman [µs] | List Dijkstra [µs] | List Bellman [µs] |
| 10 | 0 | 0 | 20 | 110 |
| 20 | 10 | 60 | 50 | 1060 |
| 30 | 10 | 230 | 140 | 4700 |
| 40 | 20 | 430 | 270 | 13990 |
| 50 | 30 | 970 | 690 | 36461.6 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba wierz. | Matrix Prim [µs] | Matrix Kruskal [µs] | List Prim [µs] | List Kruskal [µs] |
| 10 | 60 | 50 | 90 | 80 |
| 20 | 520 | 180 | 520 | 220 |
| 30 | 2500 | 830 | 2660 | 850 |
| 40 | 6940.01 | 2030 | 7490 | 2290 |
| 50 | 18560.9 | 5460.04 | 18571 | 6010.04 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba wierz. | Matrix Dijkstra [µs] | Matrix Bellman [µs] | List Dijkstra [µs] | List Bellman [µs] |
| 10 | 0 | 0 | 0 | 100 |
| 20 | 10 | 50 | 60 | 1030 |
| 30 | 0 | 220 | 210 | 4910 |
| 40 | 20 | 510 | 390 | 15710.1 |
| 50 | 30 | 940 | 670 | 36251.7 |

**V. Wnioski**

1. Algorytm Bellmana-Forda w porównaniu do algorytmu Dijkstry jest bardzo wolnym algorytmem. Jego jedyną przewagą jest wspieranie ujemnych wag i możliwość wykrycia ujemnego cyklu.

2. Mimo, że wewnątrz algorytmu został zastosowany bardzo wolny wektor większość w przypadku algorytmu Dijkstry w ogóle tego nie widać.

3. W przypadku algorytmów wyznaczających minimalne drzewo rozpinające, sama reprezentacja grafu nie wpływa znacząco na rezultat. Wyniki są w miarę podobne w obydwu przypadkach.

4. W przypadku algorytmów wyznaczających najkrótszą ścieżkę w grafie, reprezentacja grafu ma bardzo duży wpływ na końcowy rezultat. Największą różnicę można zaobserwować przy algorytmie Bellmana-Forda realizowanym na grafach o dwóch różnych reprezentacjach.

**VI. Źródła**

* en.wikipedia.org