Algorytmy grafowe - drugi projekt w ramach kursu Algorytmy i złożoność obliczeniowa

Piotr Koronczok
272955

13 czerwca 2024

Spis treści

1 Wprowadzenie

Celem niniejszego projektu jest zbadanie efektywności wybranych algorytmów grafowych oraz wpływu sposobu reprezentaji grafu w pamięci komputera.

- Wyznaczanie minimalnego drzewa rozpinającego (MST)
  - Prima
  - Kruskala
- Wyznaczanie najkrótszej ścieżki w grafie
  - Dijkstry
  - Bellmana-Forda

Algorytmy zostały zaimplementowane dla dwóch reprezentacji grafu w pamięci komputera.

- reprezentacja macierzowa -macierz incydencji
- reprezentacja listowa - lista następników

2 Oszacowanie złożoności problemu minimalnego drzewa rozpinającego MST

2.1 Algorytm Prima

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

Rysunek 1: Pseudokod algorytmu Prima. Źródło: Introduction to Algorithms, Third Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

Złożoność obliczeniowa to $O(ElogV)$

2.2 Algorytm Kruskala

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

Rysunek 2: Pseudokod algorytmu Kruskala. Źródło: Introduction to Algorithms, Third Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

Sortowanie przy odpowiedniej implementacji działa w czasie $O(E \log E)$. O złożoności tego algorytmu decyduje złożoność największa czyli złożoność sortowania zatem wynosi ona: $O(ElogE)$

3 Oszacowanie złożoności obliczeniowej problemu najkrótszej ścieżki w grafie

3.1 Algorytm Dijkstry

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

Rysunek 3: Pseudokod algorytmu Kruskala. Źródło: Introduction to Algorithms, Third Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

Kluczową dla złożoności całego algorytmu jest złożoność implementacji kolejki priorytetowej.

- dla implementacji poprzez zwykłą tablicę - $O(V^2)$
- dla implementacji kolejki poprzez kopiec - $O(E \log V)$
- dla implementacji kolejki poprzez kopiec Fibbonacciego - $O(E + V \log V)$

3.2 Algorytm Bellmana-Forda

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3       for each edge (u, v) ∈ G.E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7            return FALSE
8  return TRUE
```

Rysunek 4: Pseudokod algorytmu Bellmana-Forda. Źródło: Introduction to Algorithms, Third Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

Powyższy algorytm służy do wyznaczania najkrótszych ścieżek w grafach skierowanych. Działa on poprawnie nawet w przypadku grafów z krawędziami o ujemnych wagach w przeciwieństwie do algorytmu Dijkstry. Jego złożoność obliczeniowa to $O(VE)$.

4 Plan eksperymentu

Po wstępnych pomiarach zostały wybrane reprezentatywne rozmiary grafów.

Ilość wierzchołków
10
50
100
150
200
250
300
400

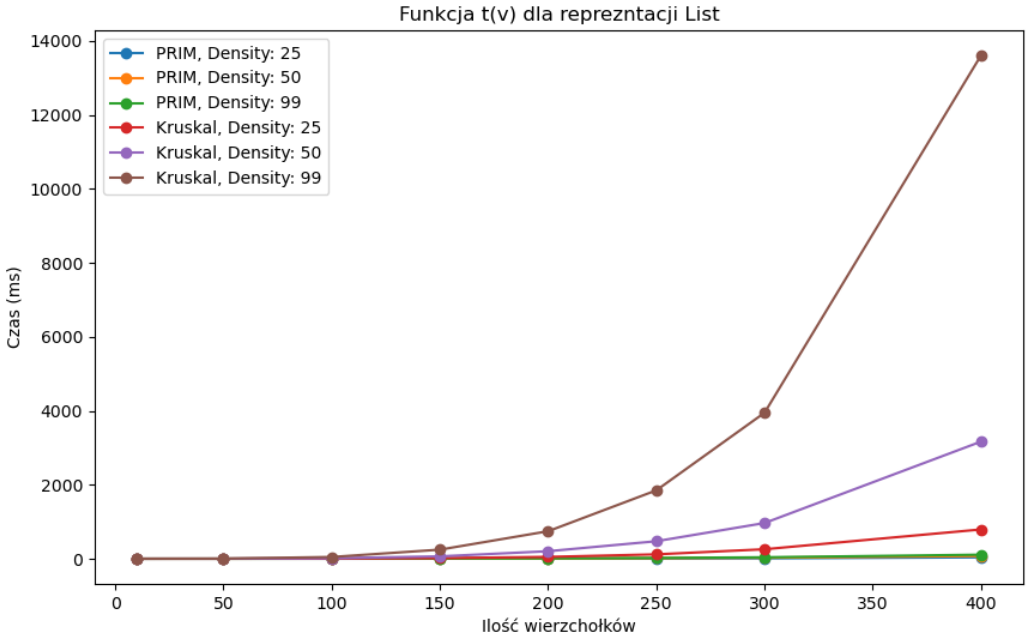
Tabela 1: Ilości wierzchołków dla badanych grafów

Zgodnie z zaleceniami algorytmy były testowane dla trzech gęstości grafów - 25, 50 oraz 99 procent. W celu uzyskania wiarygodnych wyników pomiar dla jednego zestawu danych tj. rodzaj algorytmu, rodzaj reprezentacji w pamięci, ilość wierzchołków, gęstość grafu, był wykonywany 50-krotnie.

Generacja danych odbywała się na bazie zadanych parametrów, a więc ilości wierzchołków oraz gęstości grafu. Początkowo generowane było minimalne drzewo rozpinające przy użyciu algorytmu Prima 1.

Program umożliwiający badanie napisany został w - *C++*. Pomiary wykonane na programie skompilowanym przy użyciu *g++*. Testowanie odbyło się na procesorze AMD Ryzen 5 7640U oraz Ubuntu 22.04.4 LTS, a pomiary czasu wykonywane były się przy użyciu std::chrono::high_resolution_clock. Wszystkie wartości czasu podane są w ms.

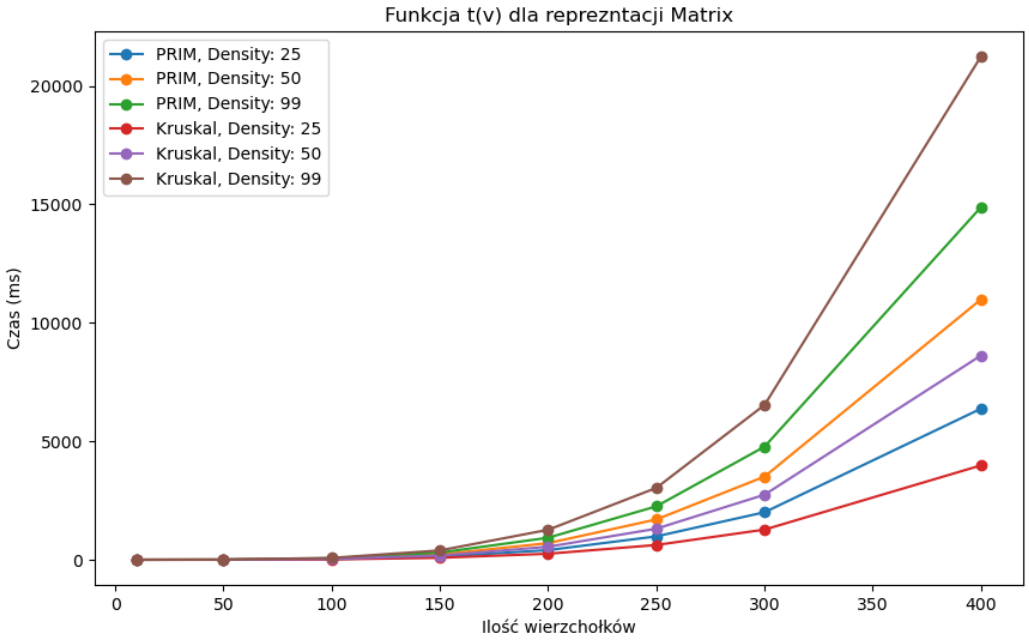
5 Algorytmy MST - wyniki



Rysunek 5: Algorytmy znajdujące MST dla reprezentacji listowej

Vertices	Prim (Density 25)	Prim (Density 50)	Prim (Density 99)	Kruskal (Density 25)	Kruskal (Density 50)	Kruskal (Density 99)
10	0.000881	0.001015	0.003042	0.001492	0.002242	0.007353
50	0.041057	0.067002	0.103570	0.202707	0.772528	3.045057
100	0.337953	0.631313	0.835154	3.258290	12.300739	47.431508
150	1.258308	2.254510	4.064355	16.290574	64.424967	245.623446
200	2.874523	7.124741	10.198710	49.471862	203.410194	744.177823
250	5.426099	16.312849	20.887840	118.802175	473.727413	1849.353796
300	10.775162	27.271733	39.018103	256.775339	965.821796	3947.608434
400	38.197063	74.569082	109.406834	792.066428	3166.891168	13605.783900

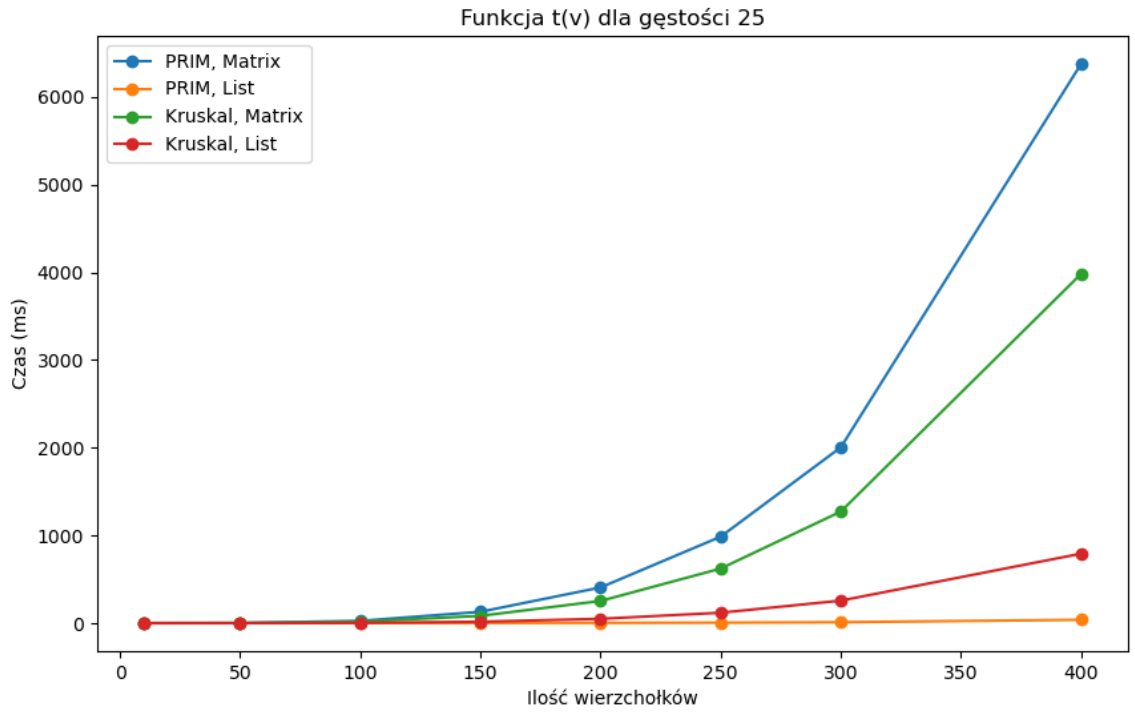
Tabela 2: Porównanie algorytmów MST w grafie dla reprezentacji listowej. Czas w ms.



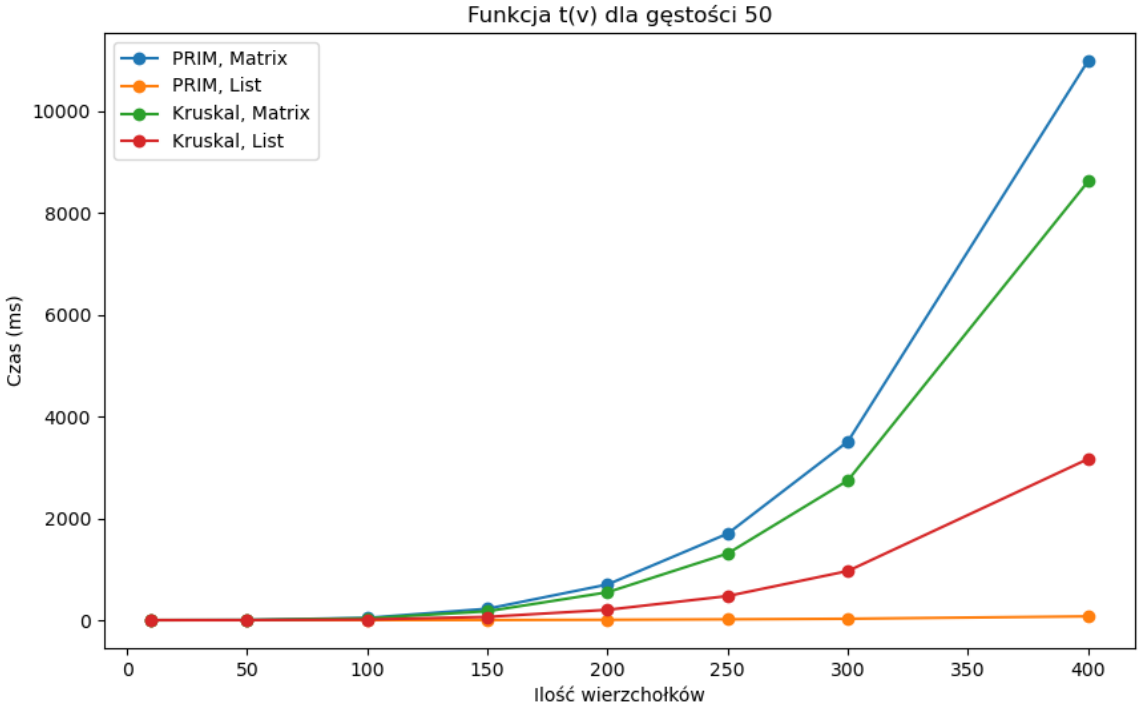
Rysunek 6: Algorytmy znajdujące MST dla reprezentacji macierzowej

Vertices	Prim (Density 25)	Prim (Density 50)	Prim (Density 99)	Kruskal (Density 25)	Kruskal (Density 50)	Kruskal (Density 99)
10	0.002562	0.003961	0.005364	0.001775	0.004095	0.008532
50	1.741119	2.962583	4.057634	1.084983	2.269647	5.053214
100	25.850628	44.417296	60.311258	15.966142	35.356805	77.212847
150	129.328170	224.093083	302.238542	83.017337	174.107399	392.836428
200	406.882406	701.212542	928.438672	252.945256	548.311853	1264.050288
250	986.151772	1700.420738	2254.308863	622.353250	1310.081458	3032.879500
300	2004.856680	3509.420041	4773.171710	1272.381766	2744.760445	6534.310529
400	6375.938138	10988.436861	14880.575704	3982.022749	8621.468096	21240.928910

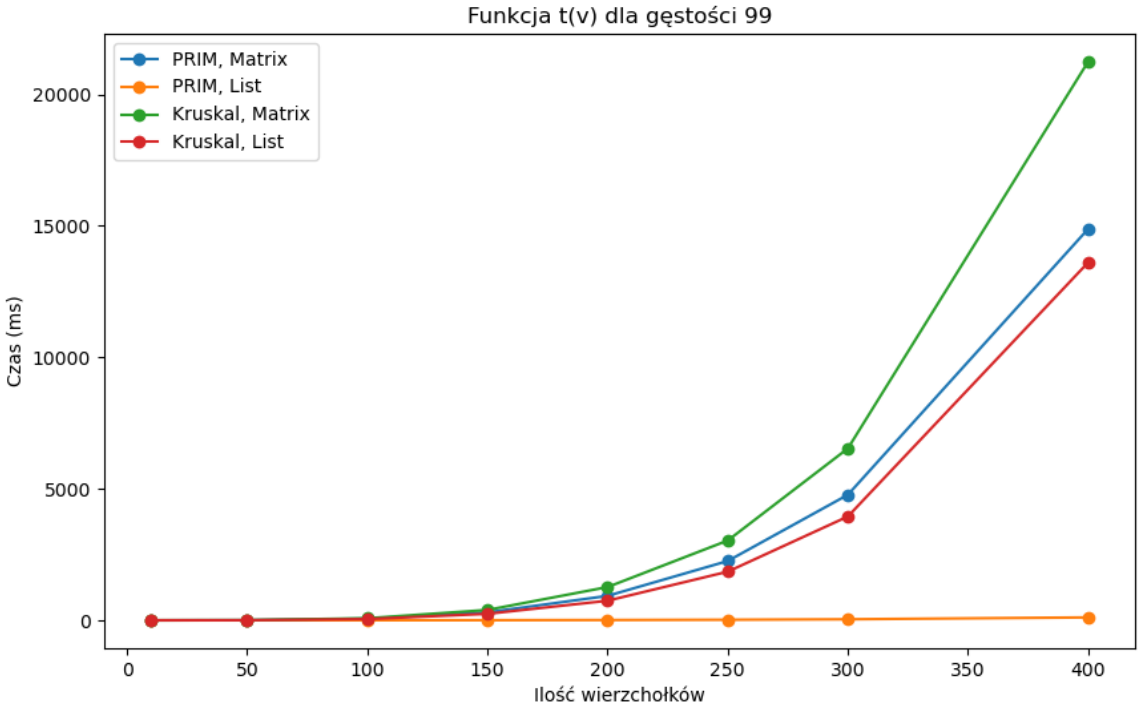
Tabela 3: Porównanie algorytmów MST w grafie dla reprezentacji macierzowej. Czas w ms.



Rysunek 7: Algorytmy znajdujące MST dla reprezentacji gęstości grafu równej 25%



Rysunek 8: Algorytmy znajdujące MST dla reprezentacji gęstości grafu równej 50%

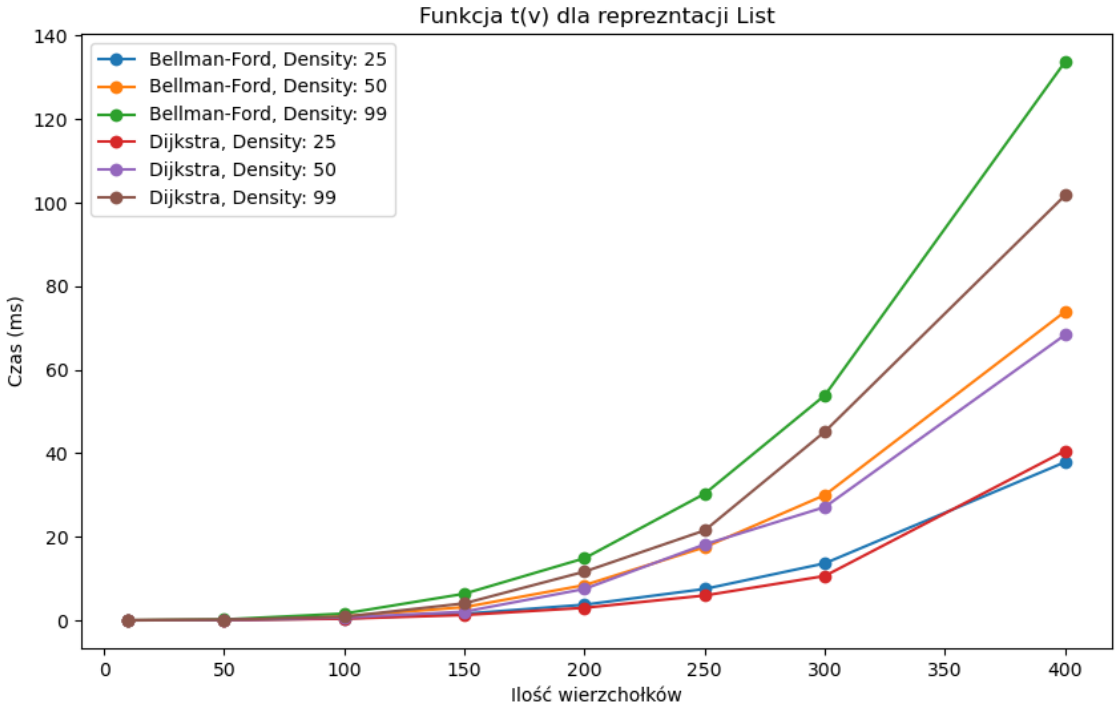


Rysunek 9: Algorytmy znajdujące MST dla reprezentacji gęstości grafu równej 99%

6 Algorytmy znajdujące najkrótszą ścieżkę - wyniki

Vertices	Dijkstra (Density 25)	Dijkstra (Density 50)	Dijkstra (Density 99)	Bellman-Ford (Density 25)	Bellman-Ford (Density 50)	Bellman-Ford (Density 99)
10	0.001595	0.000885	0.001225	0.000648	0.001064	0.001782
50	0.043678	0.065008	0.114842	0.055747	0.104294	0.200485
100	0.376653	0.652624	0.859063	0.446098	0.925455	1.610459
150	1.259431	2.063026	4.057855	1.561841	3.150466	6.365401
200	2.984720	7.511728	11.656426	3.740560	8.432870	14.863497
250	5.939165	18.175057	21.527958	7.500376	17.505872	30.310235
300	10.621136	27.119022	45.180896	13.643065	29.998218	53.877450
400	40.583698	68.365315	101.762535	37.880262	73.925236	133.759047

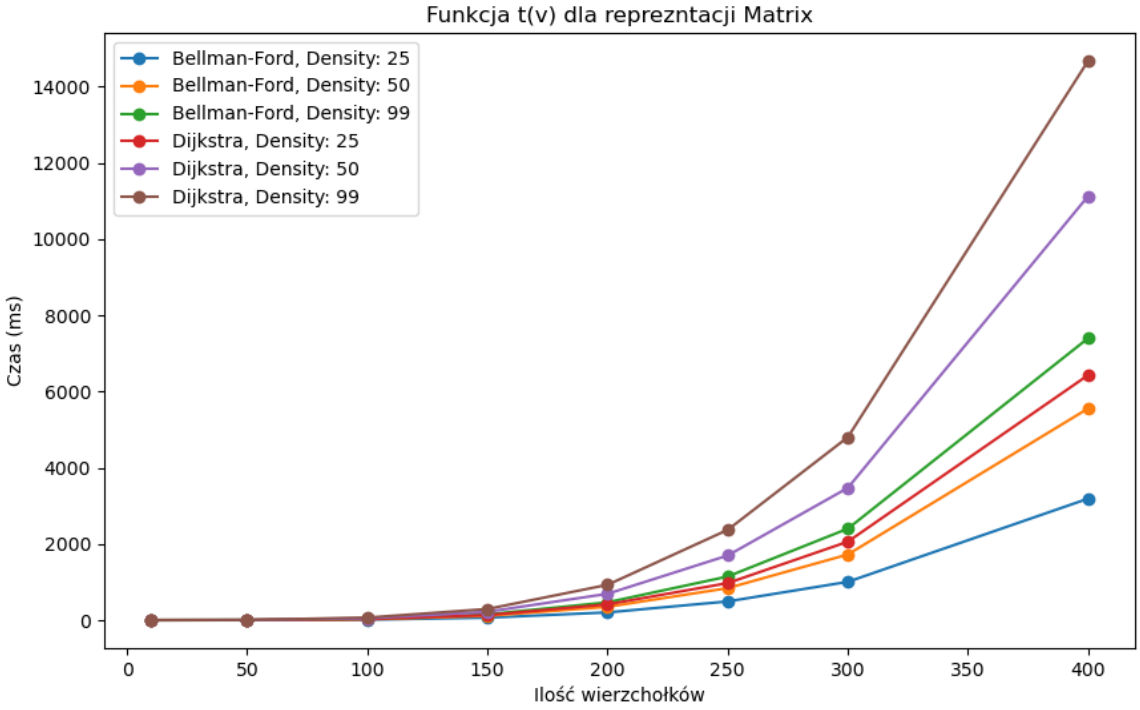
Tabela 4: Porównanie algorytmów MST w grafie dla reprezentacji macierzowej. Czas w ms.



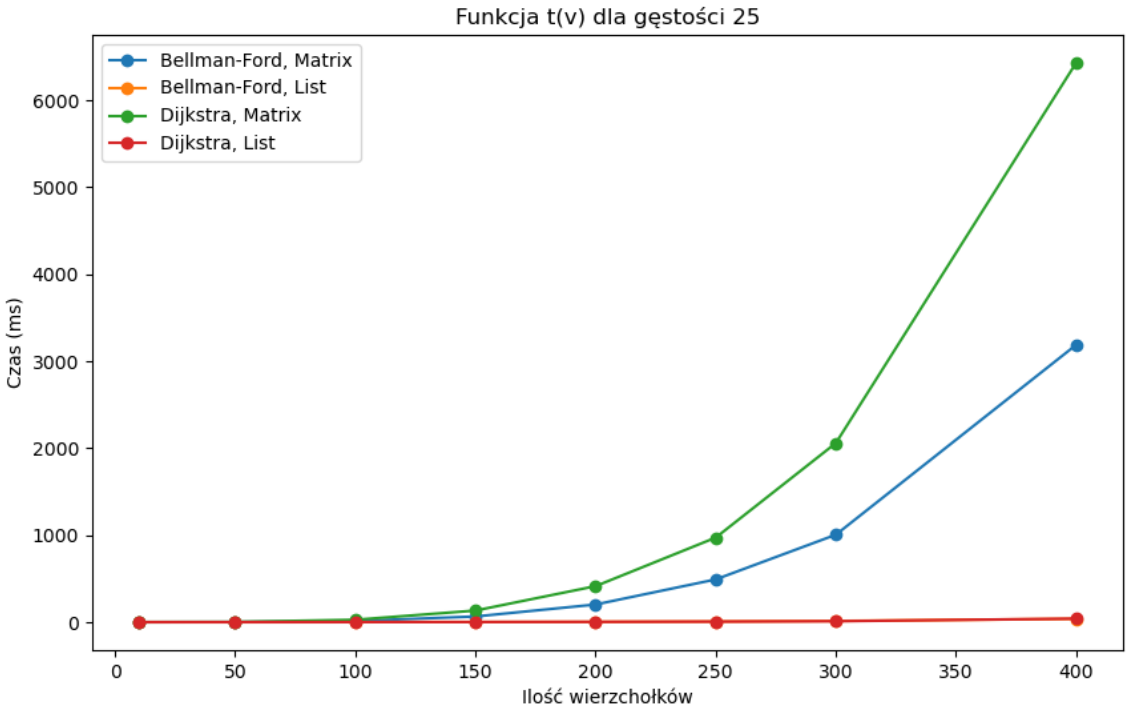
Rysunek 10: Algorytmy znajdujące najkrótszą ścieżkę dla reprezentacji listowej

Vertices	Dijkstra (Density 25)	Dijkstra (Density 50)	Dijkstra (Density 99)	Bellman-Ford (Density 25)	Bellman-Ford (Density 50)	Bellman-Ford (Density 99)
10	0.002853	0.003894	0.005049	0.012453	0.005030	0.003821
50	1.796482	3.127085	4.388912	0.886711	1.540570	2.158219
100	27.436665	47.822078	66.466823	13.063637	22.530415	30.618038
150	133.157596	219.505697	291.918619	65.032130	111.427242	152.049265
200	413.831232	691.293300	927.192526	202.751880	349.428738	467.687610
250	972.013908	1695.931070	2369.229925	491.201377	839.781068	1144.897094
300	2058.625699	3470.522899	4800.862895	1005.108342	1722.308320	2400.227937
400	6429.417355	11125.633465	14673.605086	3187.626709	5553.773210	7397.346581

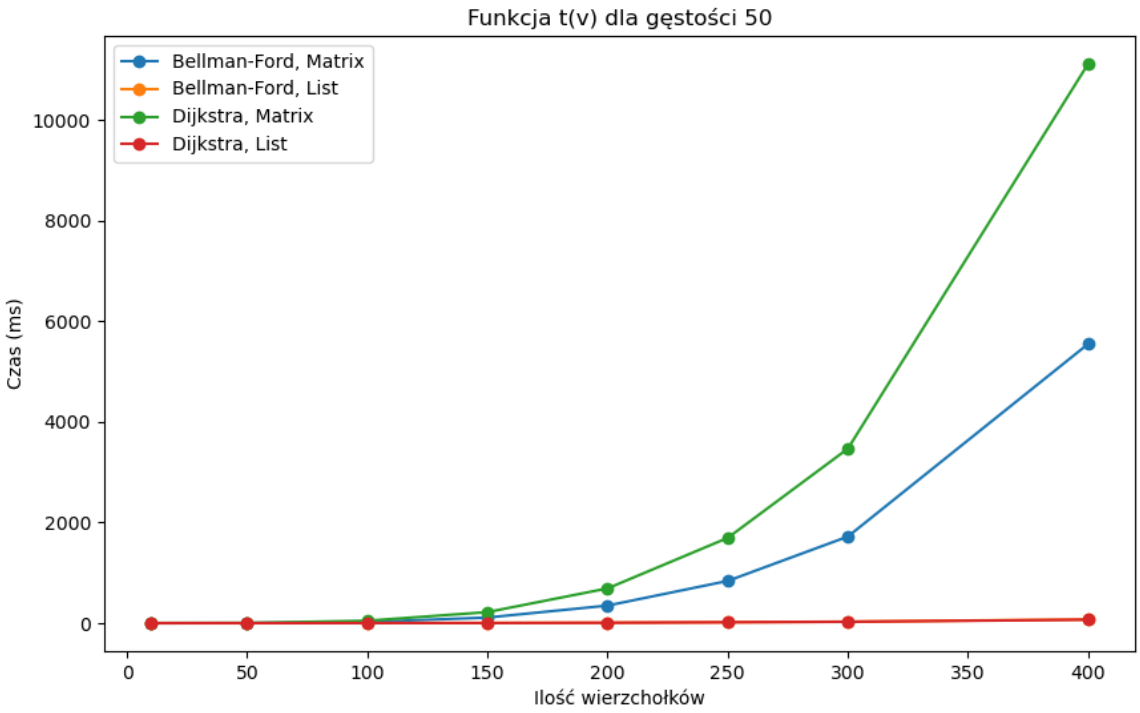
Tabela 5: Porównanie algorytmów najkrótszej ścieżki w grafie dla reprezentacji macierzowej. Czas w ms.



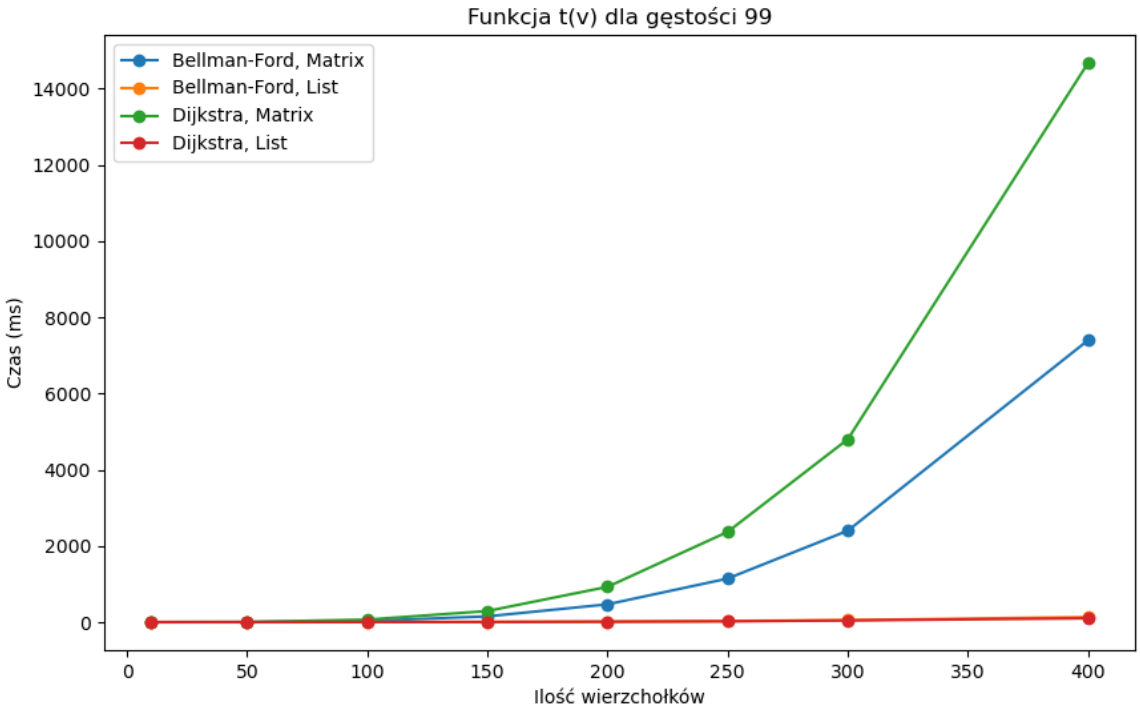
Rysunek 11: Algorytmy znajdujące najkrótszą ścieżkę dla reprezentacji macierzowej



Rysunek 12: Algorytmy znajdujące najkrótszą ścieżkę dla reprezentacji gęstości grafu równej 25%



Rysunek 13: Algorytmy znajdujące najkrótszą ścieżkę dla reprezentacji gęstości grafu równej 50%



Rysunek 14: Algorytmy znajdujące najkrótszą ścieżkę dla reprezentacji gęstości grafu równej 99%

7 Źródła

Materiały do kursu
Kruskal's algorithm
Incidence matrix