

Лабораторна робота №1

Звіт

Фіалко Ярина
Краснянький Тимур

Посилання на GitHub:

<https://github.com/TIMOUT/DescreteMath>

Мета експерименту

Визначити ефективніший алгоритм зпоміж алгоритму Краскала та Прима. Програма приймає кількість вершин, ймовірність провести ребро від вершини до якоїсь іншої та кількість ітерацій та порівнює ефективність алгоритмів.

Експеримент проведено на Lenovo 320-15IKB (Type 80XL, 80YE) з такими характеристиками: 8GB RAM, мінімальна тактова частота: 2.30GHz, ядер: 2, потоків 4, операційна система: Windows 10.

Програмний код

Алгоритм Краскала

Алгоритм краскала `kruskal.py` складається з функції з алгоритмом

`kruskal()` та допоміжної **`combine_sets()`**, створює каркас переданого графа.

```
1  import networkx as nx
2
3  def combine_sets(nodes, edge):
4      """
5      >>> combine_sets([1], {2, 3, 4}, {5, 6}), (1, 5))
6      [{1, 5, 6}, {2, 3, 4}]
7      >>> combine_sets([1, 2, 3], {4}, {5, 6, 7}), (3, 4))
8      [{1, 2, 3, 4}, {5, 6, 7}]
9      """
10     for node_1 in nodes:
11         if edge[0] in node_1:
12             for node_2 in nodes:
13                 if edge[1] in node_2:
14                     new_node = node_1|node_2
15                     return [new_node] + [node for node in nodes if node != node_1 and node != node_2]
16
17
18 def kruskal(graph):
19     """
20     Args:
21     | graph (nx.Graph): a graph (class networkx.Graph)
22     Returns:
23     | a graph: a minimum spanning tree with minimum weight using Kruskal's algorithm (class networkx.Graph)
24     >>> kruskal(nx.Graph([(1,2,{ 'weight':7}), (1,3,{ 'weight':0}), (2,3,{ 'weight':1})])).edges(data=True)
25     EdgeDataView([(1, 3, {}), (3, 2, {})])
26     """
27     tree = nx.Graph()
28     edges = sorted(list(graph.edges(data=True)), key=lambda x: x[2]["weight"])
29     nodes = [{node} for node in graph.nodes()]
30     n = 0
31     for edge in edges:
32         for node in nodes:
33             if (edge[0] in node) and not (edge[1] in node):
34                 tree.add_edge(edge[0], edge[1])
35                 nodes = combine_sets(nodes, edge)
36                 n += 1
37             if n == len(graph.nodes()) - 1:
38                 return tree
39
40     return tree
41
```

Алгоритм Прима

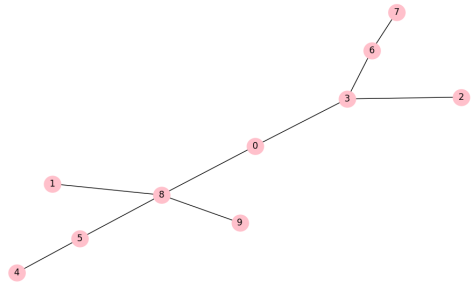
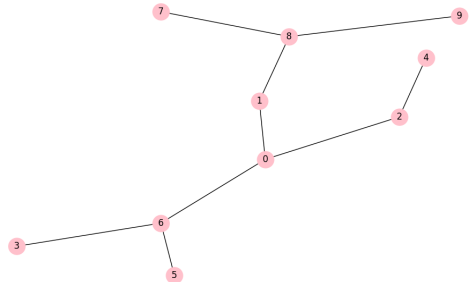
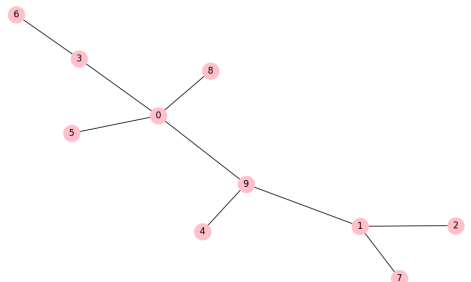
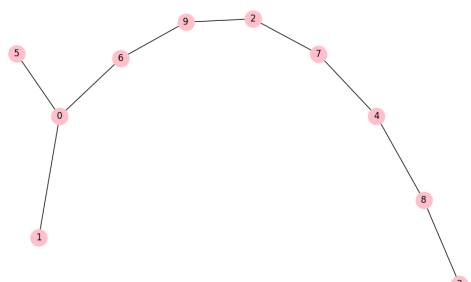
Алгоритм краскала **prim.py** складається з функції з алгоритмом **prim()**, створює каркас переданого графа.

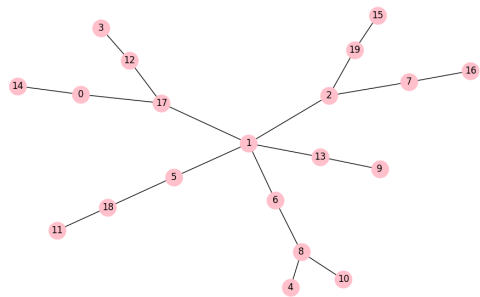
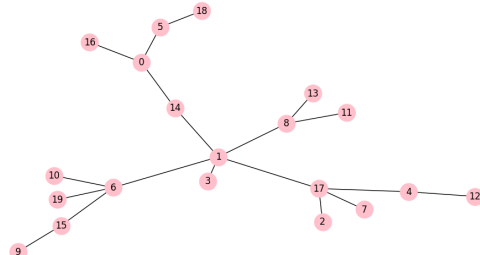
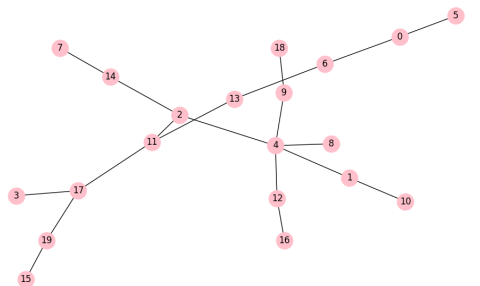
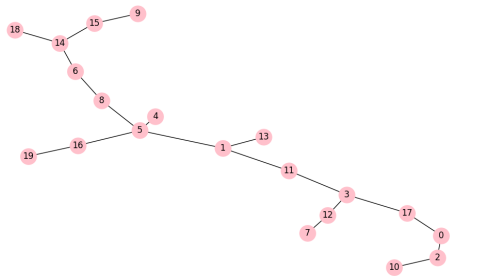
```
1 import networkx as nx
2
3 def prim(graph):
4     """
5     Args:
6     | graph (nx.Graph): a graph (class networkx.Graph)
7     Returns:
8     | a graph: a minimum spanning tree with minimum weight using Prim's algorithm (class networkx.Graph)
9     >>> prim(nx.Graph([(1,2,{ 'weight':7}),(1,3,{ 'weight':0}),(2,3,{ 'weight':1}]))).edges(data=True)
10    EdgeDataView([(1, 3, {}), (3, 2, {})])
11    """
12    tree = nx.Graph()
13    nodes = list(graph.nodes())
14    start_node = nodes[0]
15    n = 0
16    vertices = set([start_node])
17    while n < len(nodes) - 1:
18        incident = graph.edges(vertices, data=True)
19        choose_low = sorted(incident, key=lambda x: x[2]["weight"])
20        choice = choose_low[0][:2]
21        i = 0
22        while choice[0] in vertices and choice[1] in vertices:
23            i += 1
24            choice = choose_low[i][:2]
25        tree.add_edge(choice[0], choice[1])
26        graph.remove_edge(choice[0], choice[1])
27        choice = set(choice)
28        vertices |= choice
29        n += 1
30    return tree
31
```

Проведення експерименту

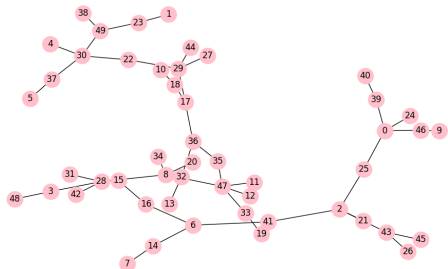
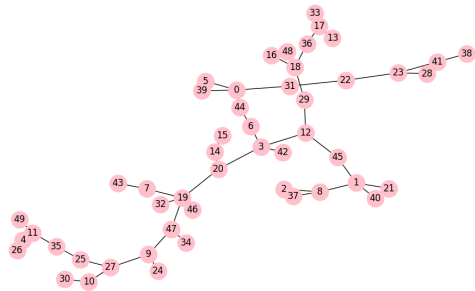
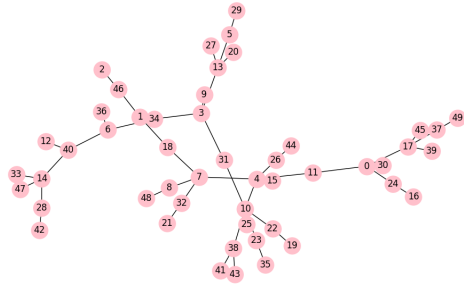
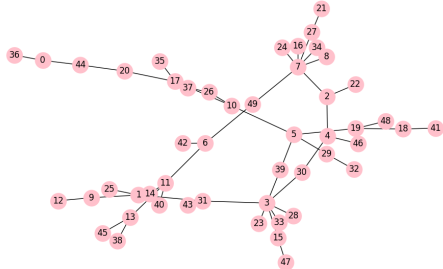
У модулі **graph_generator.py** міститься функція **main()** за допомогою якої запинуємо кількість вершин, ймовірність проведення вершини та кількість ітерацій. У нашому експерименті ми використовуємо 100 ітерацій.

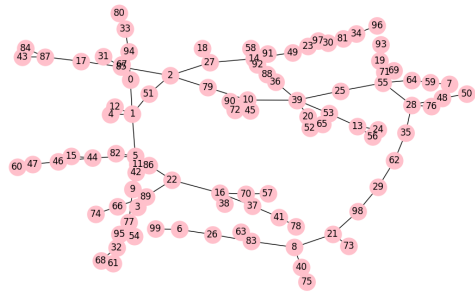
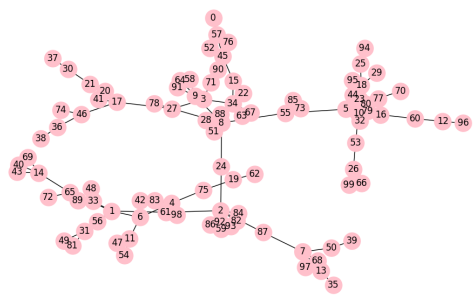
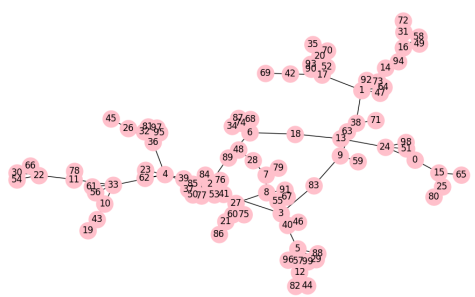
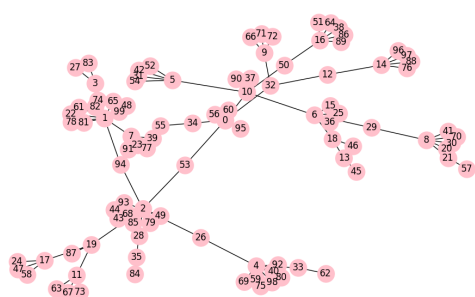
```
42     def main():
43         """
44         Tests efficiency of the algorithms.
45         """
46         print("Enter a number of nodes: ")
47         nodes = int(input(">>> "))
48
49         print("Enter completeness of the graph: ")
50         completeness = int(input(">>> "))
51
52         print("Enter number of iterations: ")
53         iterations = int(input(">>> "))
54
55         import time
56         total_kr = 0
57         total_pr = 0
58         for _ in range(iterations):
59             g = gnp_random_connected_graph(nodes, completeness)
60             start = time.time()
61             kruskal.kruskal(g)
62             end = time.time()
63             total_kr += end - start
64
65             start = time.time()
66             prim.prim(g)
67             end = time.time()
68             total_pr += end - start
69
70         prim_time = total_pr/iterations
71         kruskal_time = total_kr/iterations
72
73         print("Prim's algorithm: " + str(prim_time))
74         print("Kruskal's algorithm: " + str(kruskal_time))
75         print("Difference: " + str(prim_time - kruskal_time))
76
```

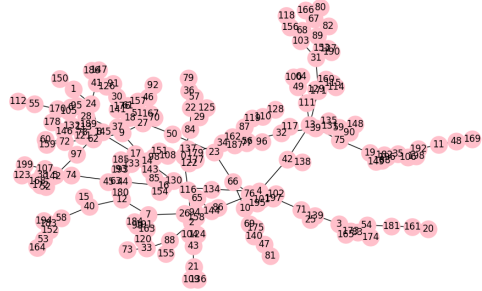
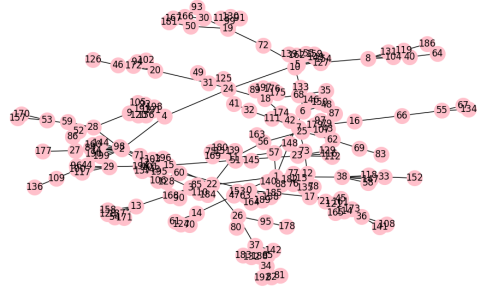
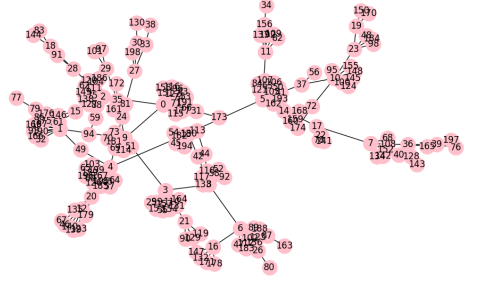
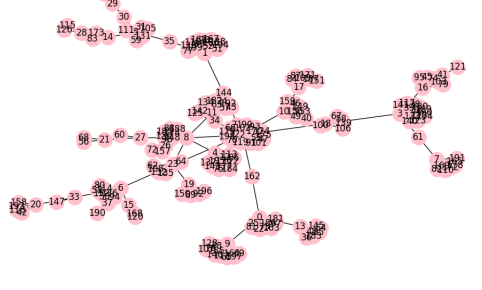
Алгоритм Краскала			
Кількість вершин	Ймовірність провести ребро	Час	Фото
10	0.25	0.00017040252685546876	
10	0.5	0.00016001224517822265	
10	0.75	0.00013998031616210938	
10	1	0.00012999534606933594	

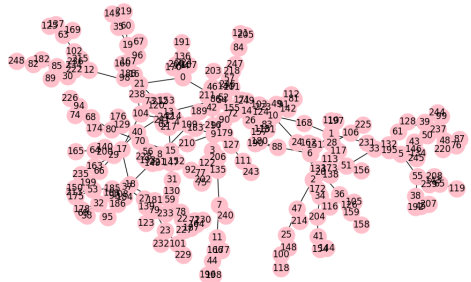
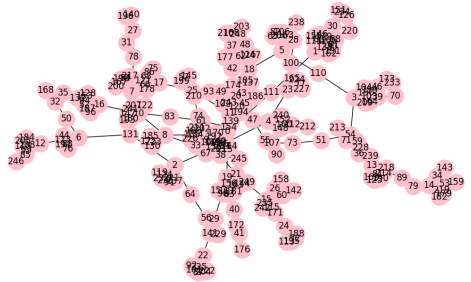
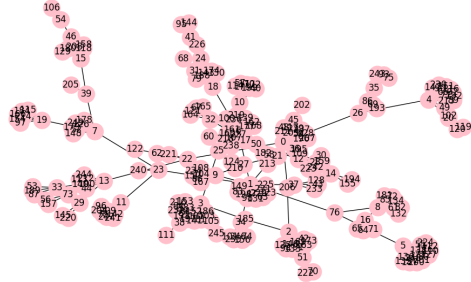
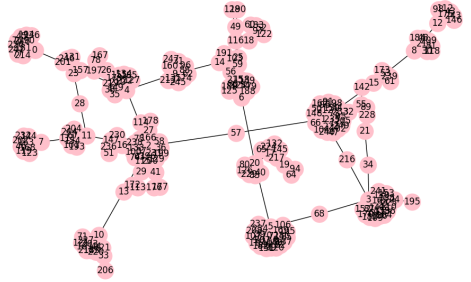
Алгоритм Краскала			
Кількість вершин	Ймовірність провести ребро	Час	Фото
20	0.25	0.00032015562057495115	
20	0.5	0.0003905797004699707	
20	0.75	0.0004006481170654297	
20	1	0.0004795408248901367	

Алгоритм Краскала

Кількість вершин	Ймовірність провести ребро	Час	Фото
50	0.25	0.0013505744934082032	
50	0.5	0.0016893887519836427	
50	0.75	0.0022502899169921874	
50	1	0.002510337829589844	

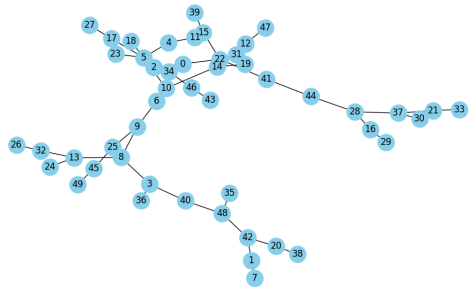
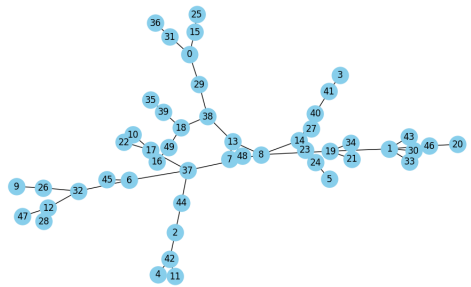
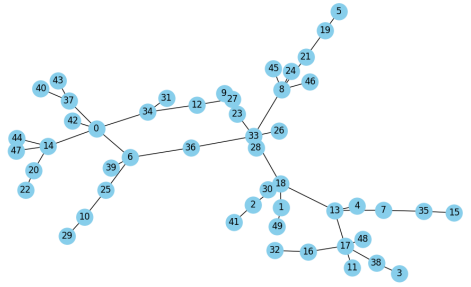
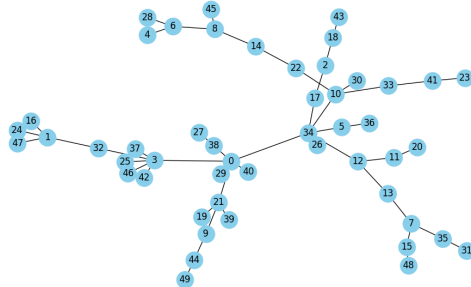
Алгоритм Краскала			
Кількість вершин	Ймовірність провести ребро	Час	Фото
100	0.25	0.004120500087738037	
100	0.5	0.006540465354919434	
100	0.75	0.009361851215362548	
100	1	0.011758456230163574	

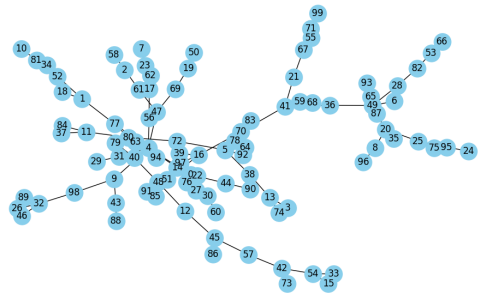
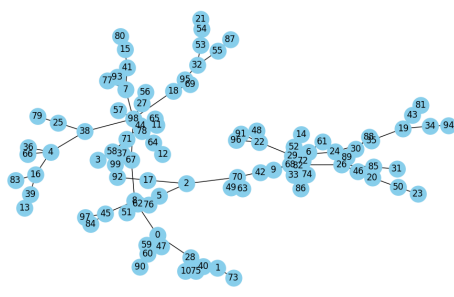
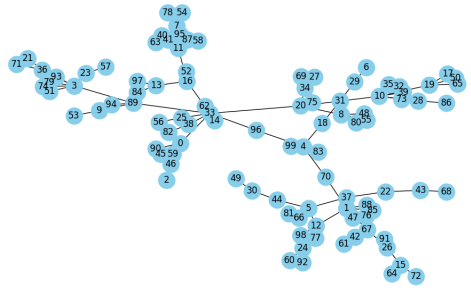
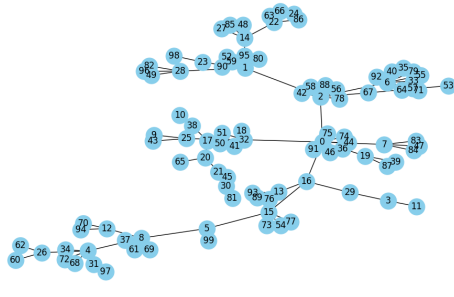
Алгоритм Краскала			
Кількість вершин	Ймовірність провести ребро	Час	Фото
200	0.25	0.016690828800201417	
200	0.5	0.03045832395553589	
200	0.75	0.04013873815536499	
200	1	0.05177566051483154	

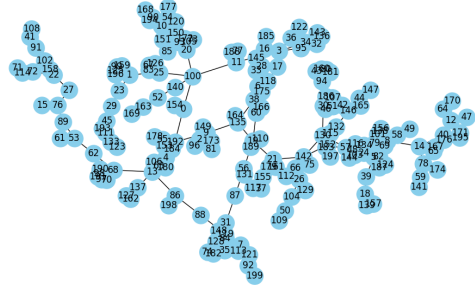
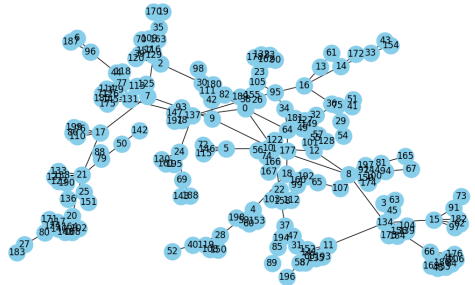
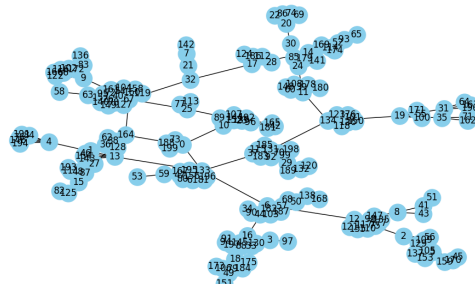
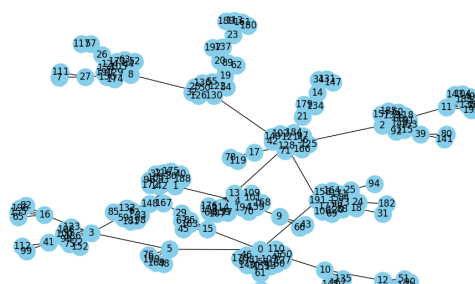
Алгоритм Краскала			
Кількість вершин	Ймовірність провести ребро	Час	Фото
250	0.25	0.03036566972732544	
250	0.5	0.04507545709609986	
250	0.75	0.06043470144271851	
250	1	0.08459551095962524	

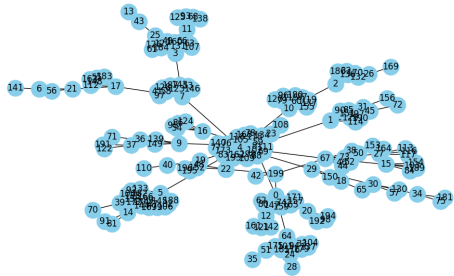
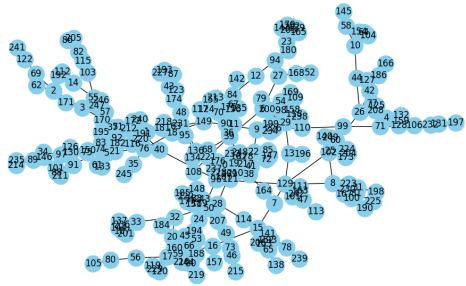
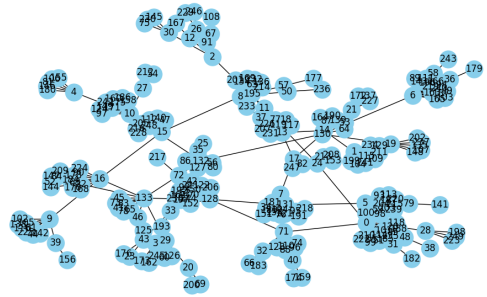
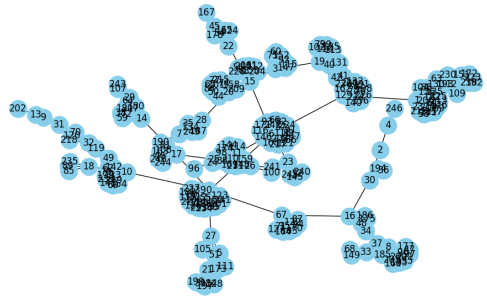
Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото
10	0.25	0.00024991512298583985	
10	0.5	0.00024991512298583985	
10	0.75	0.0004199671745300293	
10	1	0.00041968822479248046	

Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото
20	0.25	0.0011396408081054688	
20	0.5	0.0016298818588256837	
20	0.75	0.0022510838508605956	
20	1	0.002889118194580078	

Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото
50	0.25	0.01637404203414917	
50	0.5	0.026116492748260497	
50	0.75	0.0349141788482666	
50	1	0.04598390579223633	

Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото
100	0.25	0.09977493047714234	
100	0.5	0.2009675121307373	
100	0.75	0.3992867279052734	
100	1	0.5306085443496704	

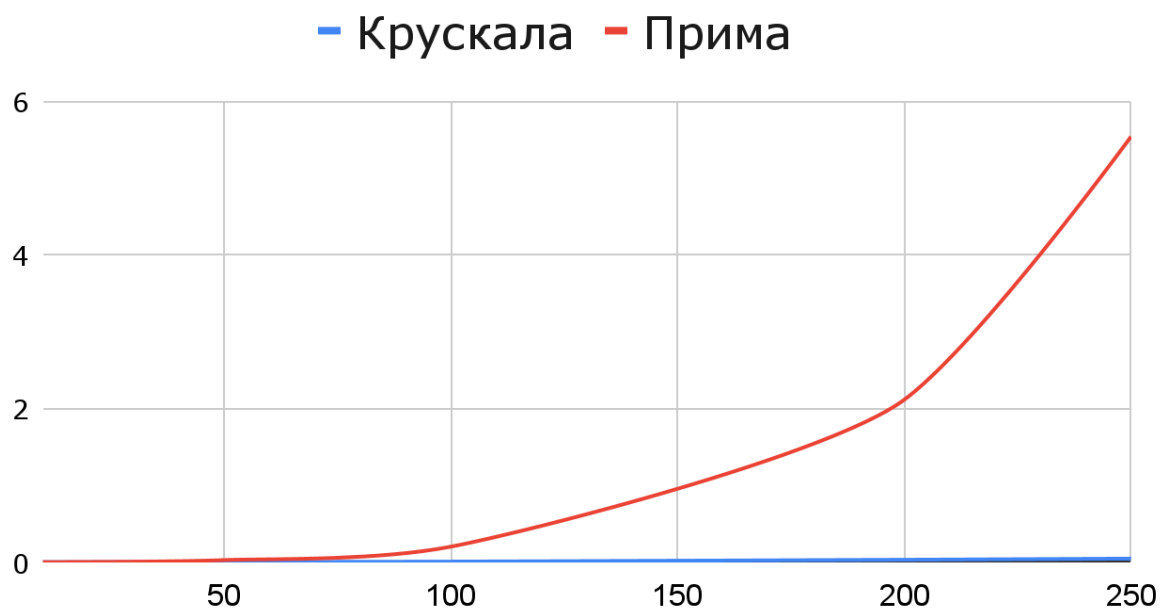
Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото
200	0.25	1.1126050400733947	
200	0.5	2.1144519233703614	
200	0.75	3.4164995098114015	
200	1	4.460636661052704	

Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото
250	0.25	5.5188414978981015	
250	0.5	5.5390751171112065	
250	0.75	6.851164722442627	
250	1	10.074769048690795	

0.25



0.5



0.75



1



Зверху подані графіки при заповненості 0.25, 0.5, 0.75 та 1.

Проаналізувавши їх, ми прийшли до висновку, що алгоритм Крускала показує найбільшу ефективність у всіх випадках.

Висновок

Отже, алгоритм Крускала виявився ефективнішим за алгоритм

Прима, особливо при графах, що мають більше 100 вершин.

Пояснюється це тим, що у алгоритма Крускала простіша реалізація.

Сортування ребер відбувається лише один раз, далі послідовно вибираємо ребра з мінімальною вагою, уникаючи утворення циклу.

Перевагою є й непотрібність пошуку інцидентних ребер та подальшого їх сортування складністю $O(n \cdot \log(n))$. Алгоритм

Крускала варто використовувати для знаходження каркасу в деревах з вершинами, що мають низький степінь.