Лабораторна робота №1

Звіт

Фіалко Ярина Краснянький Тимур

Посиланна на GitHub:

https://github.com/T1M0UT/DescreteMath

Мета експерименту

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

Експеримент проведено на Lenovo 320-15IKB (Type 80XL, 80YE) з такими характеристиками: 8GB RAM, мінамільна тактова чистота: 2.30GHz, ядер: 2, потоків 4, операційна систера: 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 V
           if edge[0] in node_1:
12 V
                 for node_2 in nodes:
13 V
                     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
21
          graph (nx.Graph): a grapth (class networkx.Graph)
22
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)
         EdgeDataView([(1, 3, {}), (3, 2, {})])
25
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 V
           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.

```
import networkx as nx
1
2
3
      def prim(graph):
4
5
         Args:
        graph (nx.Graph): a grapth (class networkx.Graph)
6
7
         a graph: a minimum spanning tree with minimum weight using Prim's algorithm (class networkx.Graph)
8
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())
         start_node = nodes[0]
14
         n = 0
15
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]
            tree.add_edge(choice[0], choice[1])
25
             graph.remove_edge(choice[0], choice[1])
26
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 допомогої якої запинуємо кількість вершин, ймовірність проведення вершини та кількість ітерацій. У нашому експерименті ми використовувати 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: ")
           completeness = int(input(">>> "))
50
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)
               end = time.time()
67
               total pr += end - start
68
69
70
           prim time = total pr/iterations
71
           kruskal time = total kr/iterations
72
73
           print("Prim's algorithm: " + str(prim_time))
           print("Kruskal's algorithm: " + str(kruskal time))
74
           print("Difference: " + str(prim time - kruskal time))
75
```

		кала	
Кількість вершин	ймовірність провести ребро	Час	Фото
10	0.25	0.0001704025268554687 6	
10	0.5	0.00016001224517822265	3
10	0.75	0.00013998031616210938	
10	1	0.0001299953460693359	5 6 9 2

		Алгоритм Крас	кала
Кількість вершин	Ймовірність провести ребро	Час	Фото
20	0.25	0.00032015562057495115	3 12 15 10 10 11 11 13 19 10
20	0.5	0.000390579700469970 7	16 0 18 13 11 10 15 15 15 15 15 15 15 15 15 15 15 15 15
20	0.75	0.0004006481170654297	3 17 18 0 5 15 10 10 15 15 15 15 15 15 15 15 15 15 15 15 15
20	1	0.000479540824890136 7	18 13 9 16 5 4 13 10 10 10 10 10 10 10 10 10 10 10 10 10

		Алгоритм Крас	кала
Кількість вершин	Ймовірність провести ребро	Час	Фото
50	0.25	0.0013505744934082032	38 43 23 1 48 23 10 25 27 35 30 24 37 35 35 31 32 45 35 35 31 31 41 32 22 43 45 36 31 31 41 32 22 43 45 36 31 31 41 32 32 34 34 35 36 31 31 41 32 32 34 34 35 36 31 31 31 31 31 31 31 31 31 31 31 31 31
50	0.5	0.0016893887519836427	$ \begin{array}{cccccccccccccccccccccccccccccccccccc$
50	0.75	0.0022502899169921874	29 5 20 20 20 20 20 20 20 20 20 20
50	1	0.002510337829589844	21 22 24 16 34 20 17 37 26 10 89 22 22 23 30 30 32 12 9 25 11 14 11 24 16 34 25 4 19 46 18 41 26 5 4 46 18 41 27 28 33 28 15 15

		:кала	
Кількість вершин	Ймовірність провести ребро	Час	Фото
100	0.25	0.004120500087738037	80 33 34 16 55 91 40 17 18 19 19 19 19 19 19 19 19 19 19
100	0.5	0.006540465354919434	$ \begin{array}{cccccccccccccccccccccccccccccccccccc$
100	0.75	0.009361851215362548	35 70 16 39 16 39 16 39 16 39 17 1 14 1 15 16 18 18 18 18 18 18 18 18 18 18 18 18 18
100	1	0.011758456230163574	$\begin{array}{cccccccccccccccccccccccccccccccccccc$

	Алгоритм Краскала			
Кількість вершин	Ймовірність провести ребро	Час	Фото	
200	0.25	0.016690828800201417	112 55 1969 24 20 197 26 198 198 198 198 198 198 198 198 198 198	
200	0.5	0.03045832395553589	150-46 19402 150-46 19402 15	
200	0.75	0.04013873815536499		
200	1	0.05177566051483154	152 20 21 11 11 21 12 12 12 12 12 12 12 12 12	

	Алгоритм Краскала			
Кількість вершин	Ймовірність провести ребро	Час	Фото	
250	0.25	0.03036566972732544	1297 169 350 191 1285 248 5128 5131 1285 248 5128 5131 1285 248 5128 5131 1285 248 5128 5131 1285 248 5128 5131 1285 248 5128 5131 1285 248 5128 5131 1285 248 5128 5131 1285 248 5128 5128 5128 5128 5128 5128 5128 512	
250	0.5	0.04507545709609986	1960 27 21088 68693 1832 20 177625 100 168 1533 177625 100 168 1533 177625 100 168 1533 177625 100 168 1533 177625 100 100 100 100 100 100 100 100 100 10	
250	0.75	0.06043470144271851	106 54 120 15 16 12 20 15 16 16 17 18 19 10 10 10 10 10 10 10 10 10 10	
250	1	0.08459551095962524	1200 1100	

		Алгоритм При	има
Кількість вершин	Ймовірність провести ребро	Час	Фото
10	0.25	0.0002499151229858398	
10	0.5	0.0002499151229858398 5	
10	0.75	0.0004199671745300293	
10	1	0.000419688224792480 46	

	Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото	
20	0.25	0.0011396408081054688	16 11 12 13 13 14 15 10 10 10 10 10 10 10 10 10 10 10 10 10	
20	0.5	0.0016298818588256837		
20	0.75	0.0022510838508605956		
20	1	0.002889118194580078		

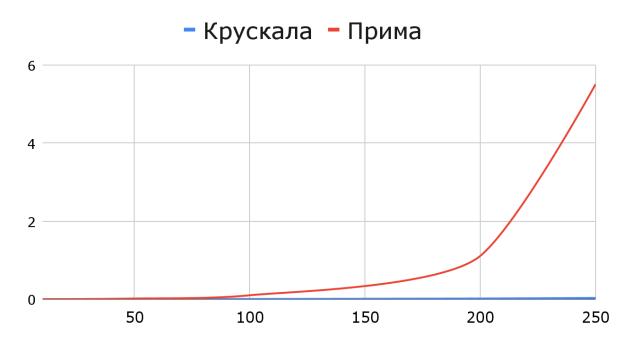
		Алгоритм При	има
Кількість вершин	Ймовірність провести ребро	Час	Фото
50	0.25	0.01637404203414917	27 17 18 4 11 ¹³ 12 23 5 2 30 0 72 31 10 60 43 4 28 37 30 71 - 33 24 13 45 6 3 35 36 40 45 7 7 38
50	0.5	0.026116492748260497	36 25 31 15 0 41 3 3 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 3 3 40 20 20 3 40 20 20 3 40 20 20 3 40 20 20 20 20 20 20 20 20 20 20 20 20 20
50	0.75	0.0349141788482666	
50	1	0.04598390579223633	23

	Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото	
100	0.25	0.09977493047714234	10 13 4 58 7 50 57 53 53 53 53 53 53 53 53 53 53 53 53 53	
100	0.5	0.2009675121307373	85 53 57 15 53 57 17 55 18 26 30 24 53 12 60 13 14 61 12 13 14 14 14 14 14 14 14 14 14 14 14 14 14	
100	0.75	0.3992867279052734	78 54 63 41 75758 71 21 56 33 22 57 73 13 15 56 22 35 14 56 18 8055 92 483 19 19 18 8055 93 483 19 19 18 8055 93 483 19 19 18 8055 94 83 19 19 18 8055 95 19 19 18 8055 96 17 19 18 18 8055 97 18 18 18 18 18 18 18 18 18 18 18 18 18	
100	1	0.5306085443496704	255 48 625 286 265 28 21 399 51 80 265 28 25 25 25 25 80 275 28 25 25 25 25 25 25 25 25 25 25 25 25 25	

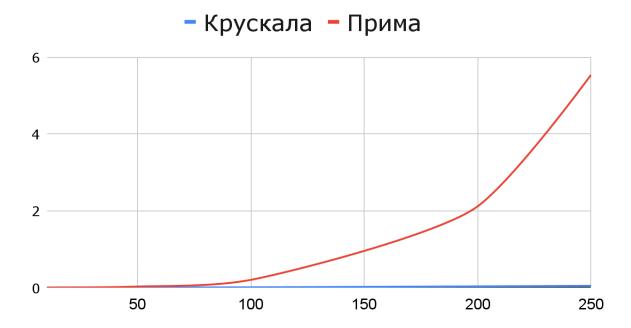
	Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото	
200	0.25	1.1126050400733947	100 107 108 107 108 107 108 109 109 109 109 109 109 109 109 109 109	
200	0.5	2.1144519233703614	1709 187 70953 187 70953 188 96 187 188 98 188 188 188 188 188 188 188 188	
200	0.75	3.4164995098114015	20049 20049	
200	1	4.460636661052704		

	Алгоритм Прима			
Кількість вершин	Ймовірність провести ребро	Час	Фото	
250	0.25	5.5188414978981015	141-6 56-21 111-17 17 18 18 18 19 19 19 19 19 19 19 19 19 19 19 19 19	
250	0.5	5.5390751171112065	241 603 145 103 1287 12 12 71 1685 2 42 145 166 10 10 10 10 10 10 10 10 10 10 10 10 10	
250	0.75	6.851164722442627	2286 100 12 17 26	
250	1	10.074769048690795	100 130 110 110 110 110 110 110 110 110	

0.25



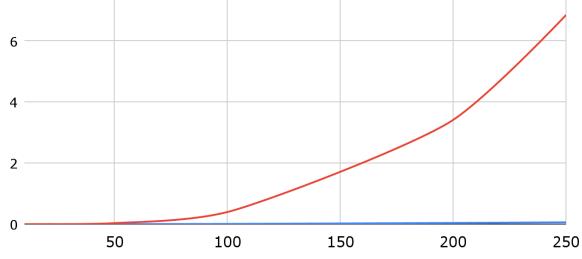




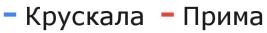


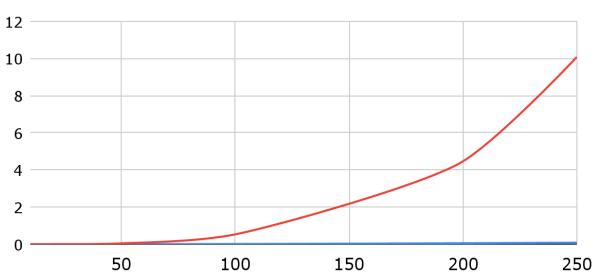
8 -





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, алгорим 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*log(n)). Алгоритм Крускала варто використовувати для знаходження каркасу в деревах з вершинами, що мають низький степінь.