

Actual Problem statement

BORUVKA'S ALGORITHM (Randomized linear-time MST Algo)

Minimum Weighted Vertex Cover (MWVC) problems:

*) By installing antivirus SW at the servers of the graph (MWVC), malware propagation among a computer network could be minimized with the minimum cost

*) By deploying IoT Digital Camera Node at vertices of MWVC, we could surveil all the road in a transport network with minimum cost

By:

- Palash Pratim Deka
(Emp ID: 22515038)
- Jai Prakash Toppo
(Emp ID: 22502302)

Time comparison between algorithms to create minimum spanning tree



Minimum spanning tree

A minimum spanning tree or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.



Methods to create a minimum spanning tree from a given graph

- Kruskal's algorithm

The Worst time complexity of this algorithm is $O(E \log V)$, where E is a number of edges and V is a number of vertices

- Boruvka's algorithm

Its time complexity is $O(E \log V)$, where E is the number of edges and V is the number of vertices.


- Prim's algorithm

The time complexity of the Prim's Algorithm is $O((V + E) \log V)$



Code procedure:

- Here we have implemented a code that generates a random graph
- It takes the value of nodes from user and creates a random value of edge(m)
- With the consideration of all points to create a graph it generates m random edges
- The all three MST algos, i.e kruskal, boruvka and prims have been implemented in different files

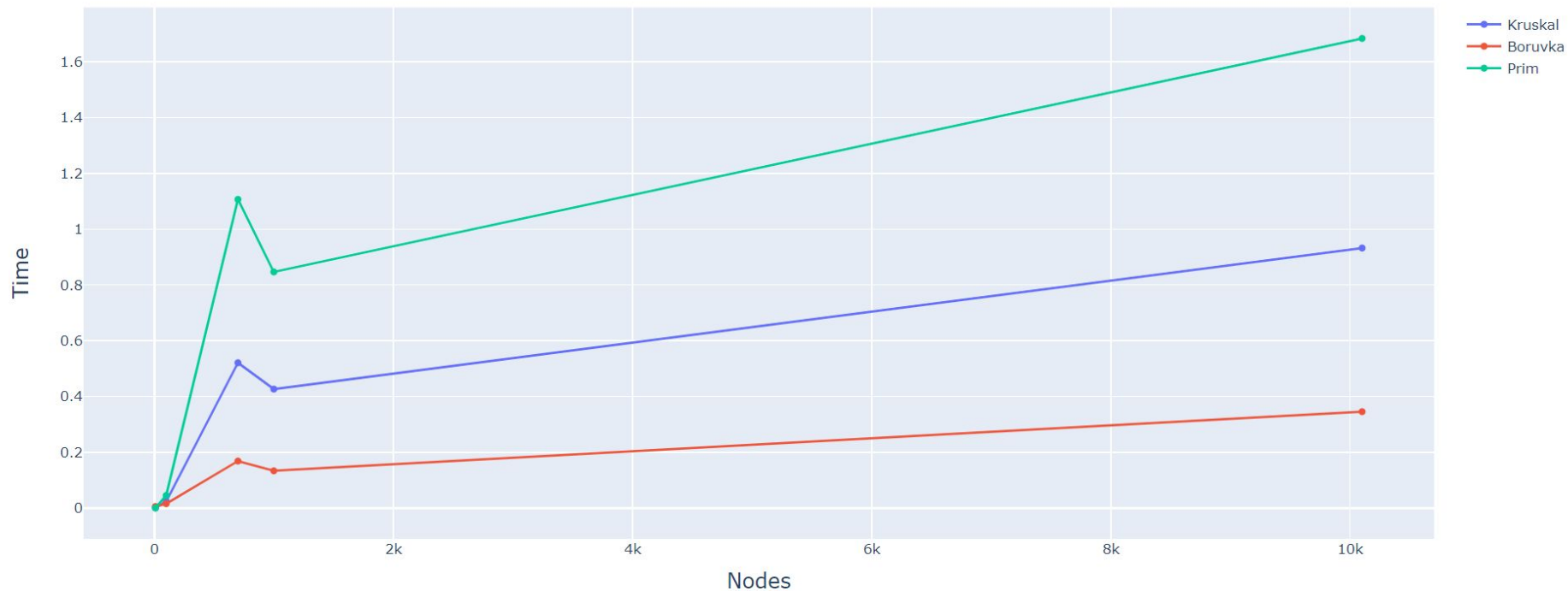
- 
- All the three algorithms separately generate the MST and store it in a newly generated output file for future reference
 - Inside the file, main.py, all the three algorithms are called and it measures the time taken by all three
 - It does the same for all the nodes provided by user
 - Later by using python library plotly, we can see the graph of No of nodes vs taken time for all the three algorithms
 - In the file storage.py, it updates the average time of all three algorithms that particular node has taken from the no of times it has ran till now, irrespective of no of edges.
 - The explanation of all the functions/variables can be found in the documentation file

Sample output of a single input

```
C:\Users\Palash Pratim Dea\Desktop\AAAAAA_codes_2022\Python Codes\Intern>python main.py
Enter nodes : 10 100 700 1000 10101
Nodes : 10      edges : 13
Nodes : 100     edges : 3831
Nodes : 700     edges : 63941
Nodes : 1000    edges : 49532
Nodes : 10101   edges : 97131
Kruskal times- [0.0007279000710695982, 0.024654899956658483, 0.5211181999184191, 0.4267601000610739, 0.9323551000561565
]
Boruvka times- [0.006145999999716878, 0.016080699861049652, 0.16887410008348525, 0.1339199000503868, 0.3458928000181913
4]
Prim times- [0.0007864001672714949, 0.045452099991962314, 1.106568600051105, 0.8466610999312252, 1.683521999977529]
10 7.0 0.0008433571950133358 0.004848885715806058 0.001014128593461854
100 3.0 0.016329133262236912 0.045599199986706175 0.030027166629830997
700 1.0 0.45261809998191893 0.14754529995843768 0.9619586998596787
1000 3.0 0.551322366654252 0.1843016998997579 0.9552015333902091
10101 3.0 0.5788291999294112 0.2611427332740277 0.994133000029251
```

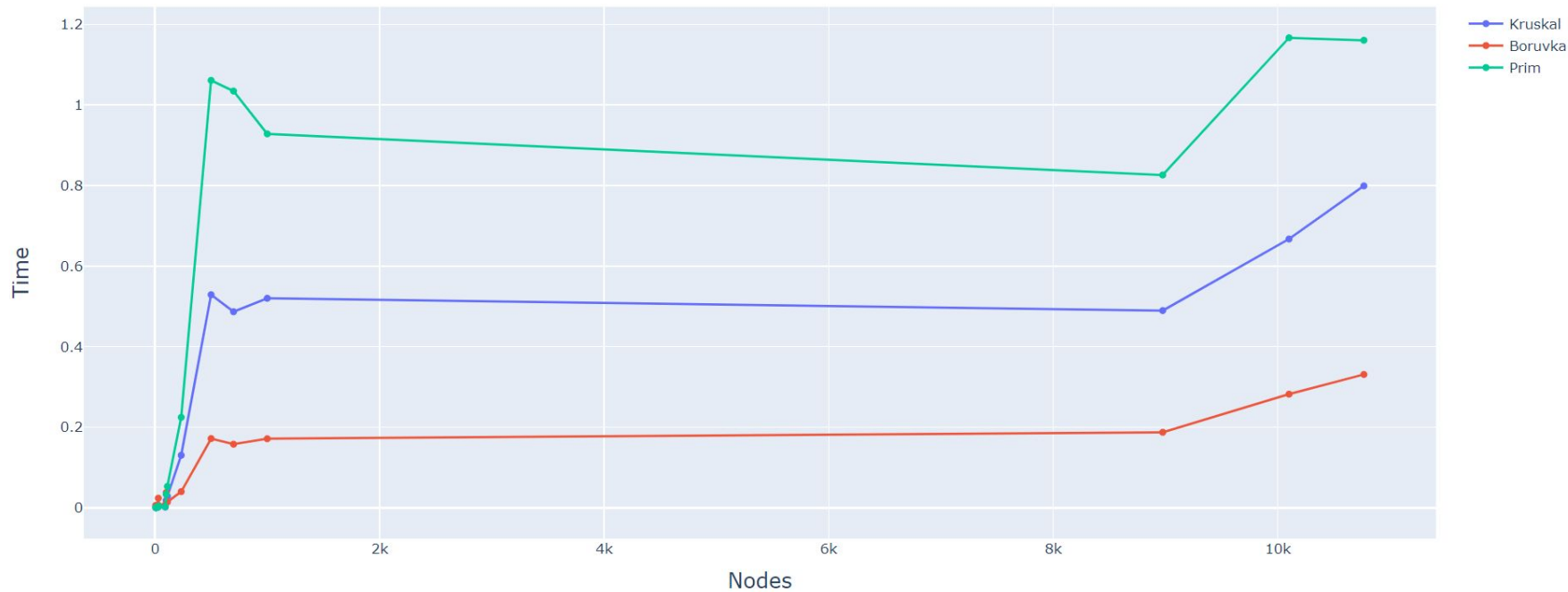
Graph for previous input

Kruskal vs Boruvka vs Prim



Average time graph

Kruskal vs Boruvka vs Prim avg time graph





Thank You