Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 23 Issue 1 Version 1.0 Year 2023
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Comparison of Primand Kruskal's Algorithm

By Rohit Maurya & Rahul Sharma
Ajeenkya D Y Patil University

Abstract- The goal of this research is to compare the performance of the common Prim and the Kruskal of the minimum spanning tree in building up super metric space. We suggested using complexity analysis and experimental methods to evaluate these two methods. After analysing daily sample data from the Shanghai and Shenzhen 300 indexes from the second half of 2005 to the second half of 2007, the results revealed that when the number of shares is less than 100, the Kruskal algorithm is relatively superior to the Prim algorithm in terms of space complexity; however, when the number of shares is greater than 100, the Prim algorithm is more superior in terms of time complexity. A spanning tree is defined in the glossary as a connected graph with non-negative weights on its edges, and the challenge is to identify a maz weight spanning tree. Surprisingly, the greedy algorithm yields an answer. For the problem of finding a min weight spanning tree, we propose greedy algorithms based on Prim and Kruskal, respectively. Graham and Hell provide a history of the issue, which began with Czekanowski's work in 1909. The information presented here is based on Rosen.

Keywords: kruskal, prim's, graph, minimal spanning trees, complexity.

GJCST-C Classification: FOR Code: 080201

COMPARISONOFPRIMANDKRUSKALSALGORITHM

Comparison of Primand Kruskal's Algorithm

Rohit Maurya [α] & Rahul Sharma [σ]

Abstract- The goal of this research is to compare the performance of the common Prim and the Kruskal of the minimum spanning tree in building up super metric space. We suggested using complexity analysis and experimental methods to evaluate these two methods. After analysing daily sample data from the Shanghai and Shenzhen 300 indexes from the second half of 2005 to the second half of 2007, the results revealed that when the number of shares is less than 100, the Kruskal algorithm is relatively superior to the Prim algorithm in terms of space complexity; however, when the number of shares is greater than 100, the Prim algorithm is more superior in terms of time complexity. A spanning tree is defined in the glossary as a connected graph with non-negative weights on its edges, and the challenge is to identify a maz weight spanning tree. Surprisingly, the greedy algorithm yields an answer. For the problem of finding a min weight spanning tree, we propose greedy algorithms based on Prim and Kruskal, respectively. Graham and Hell provide a history of the issue, which began with Czekanowski's work in 1909. The information presented here is based on Rosen.

Keywords: kruskal, prim's, graph, minimal spanning trees, complexity.

I. Introduction

A minimum spanning tree of an undirected graph can be readily obtained using Prim or Kruskal's classical algorithms. To enumerate all spanning trees in an undirected graph, a number of algorithms have been suggested. These algorithms' main worries are good time and space complexities. A minimum spanning tree of an undirected graph can be readily obtained using Prim or Kruskal's classical algorithms. A number of algorithms have been suggested to count all of an object's spanning trees.

A spanning tree of a connected graph can be constructed including all the vertices with minimum possible no of edges. If there are n vertices in the graph, then each spanning tree has n-1 edges. A connected weighted graph where all the vertices are interlinked by some weighted edges can contain multiple numbers of spanning trees.A minimum spanning tree of an undirected graph can be easily obtained using Prim or Kruskal's classical algorithms. A number of algorithms for enumerating all spanning trees in an undirected graph have been proposed. These algorithms' main concerns are good time and space complexities. The majority of algorithms generate spanning trees by utilising some fundamental cut or circuit. The cost of the tree is not considered during the generation process. This paper presents an algorithm for generating spanning trees of a graph in decreasing cost order. New opportunities emerge by generating spanning trees in increasing cost order. This method can be used to find the second smallest or, more broadly, the k-th smallest spanning tree.The smallest spanning tree satisfying some additional constraints can be found by checking whether these constraints are satisfied at each generation. Our algorithm is based on Murty's (1967) algorithm, which enumerates all solutions to an assignment problem in increasing cost order. The complexities of time and space are discussed.

The network is undirected. These algorithms' main worries are good time and space complexities. The goal of this research is to compare the performance of the common Prim and the Kruskal of the minimum spanning tree in constructing super metric space. To evaluate these two methods, we suggested using complexity analysis and experimental methods. After analysing daily sample data from the Shanghai and Shenzhen 300 indexes from the second half of 2005 to the second half of 2007, the results revealed that when the number of shares is less than 100, Kruskal algorithm is relatively superior to Prim algorithm in terms of space complexity; however, when the number of shares is greater than 100, Prim algorithm is more superior in terms of time complexity.

II. Prim's Algoritms

VojtchJarnak, a Czech mathematician, created the Prim algorithm in 1930. Robert C. Prim rediscovered it in 1957, and Edsger W. Dijkstra republished it in 1959. To determine the minimal spanning tree (MST) of a given linked weighted graph, Prim's algorithm uses a greedy approach. When the graph is dense, this algorithm is recommended. When there are many edges in a graph, the graph is said to be dense. Only undirected linked graphs can use this approach, and there must not be any edges with a negative edge weight. The algorithm is pretty effective in this situation. There will always be a shortest path because there are no cycles with nonnegative weights.

It begins by choosing a random vertex to serve as the tree's root. Then itThe shortest edge from any vertex in the tree to the new vertex is added in order to extend the tree, as is the edge closest to the current vertex. Once all vertices have been added to the tree, the procedure ends.

Author α α: Ajeenkya D Y Patil University, Pune, Maharashtra, India.
e-mail: rohit221107@gmail.com

III. The Following are the Steps to find the Minimum Spaning Tree using Prim's Algoritms

1. If a graph has loops and parallel edges, remove those loops and parallel edges.
2. Select any node at random, labelling it with a distance of 0 and all other nodes as. The chosen node is considered the current node and has been visited. All other nodes are assumed to be unvisited.
3. Locate all unvisited nodes that are currently linked to the current node. Determine the distance between the unvisited nodes and the current node.
4. Label each vertice with its corresponding weight to the current node, but relabel a node if it is less than the previous label value. The nodes are labelled with their weights each time; keep track of the path with the smallest weight.
5. Color over the current node to indicate that it has been visited. We don't need to look at a vertex again once we've visited it.
6. Among all unvisited nodes, find the one with the lowest weight to the current node; mark this node as visited and treat it as the current working node.
7. Repeat steps 3–5 until all nodes have been visited.

Prim's Algoritms:

```
PRIM (Graph, c, t)
P ěN[Graph]
For every m  P
  do key[m] ě Ğ
  key[t] ě 0
  [t]ěnull
  while P Į Ø
do m ě EXTRACTMINNODE(P)
for every n  Adjacent[m]
do if n  P and c (m,n ) < key[n]

[n] ě m
key[n]  ě  c(m,n)
```

IV. Kruskal's Algoritms

This algorithm, designed by Joseph Kruskal, was published for the first time in the Proceedings of the American Mathematical Society in 1956. The algorithm begins by creating an ordered set of edges by weights and proceeds through the ordered set by adding an edge to the partial MST if the new edge does not form a cycle. The algorithm takes a greedy approach, in which it finds the path with the least weight in each iteration stage and includes it in the growing spanning tree.

Sort all edges of the given graph in increasing order using Kruskal's algorithm. If the newly added edge does not form a cycle, it continues to add new edges and nodes to the MST. It selects the minimum weighted edge first, followed by the maximum weighted edge. Thus, in order to find the optimal solution, it makes a locally optimal choice in each step. As a result, this is a Greedy Algorithm.

V. The Steps for Detemining MST using Kuruskal Algoritms are as Follow

Sort the edges in non-descending order of weight.

Choose the smallest edge. Check to see if it forms a cycle with the spanning tree that has been formed thus far. Include this edge if the cycle is not formed. Otherwise, throw it away.

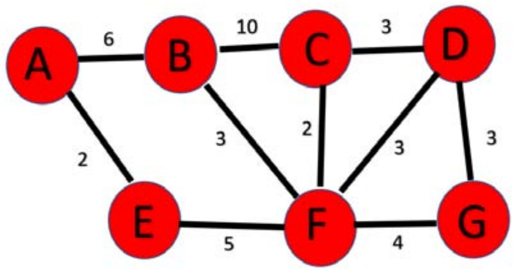
Step 2 should be repeated until the spanning tree has (V-1) edges.

Kruskal Algoritms

```
Kruskal(Graph):
T = Empty;
For every node n ę G.N:
CreateSet(n)
For every path (m, n) ę G.E arranged by increasing
weights(m,n):
 if NewSet(m) Į NewSet(n):
T = T Ĥ {(m, n)}
 UNION (m, n)
return T
```

Problem Statement

Find the MST through Prim's Algorithm and Explain it step wise.

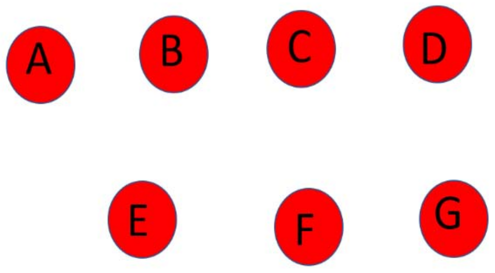


Solution:

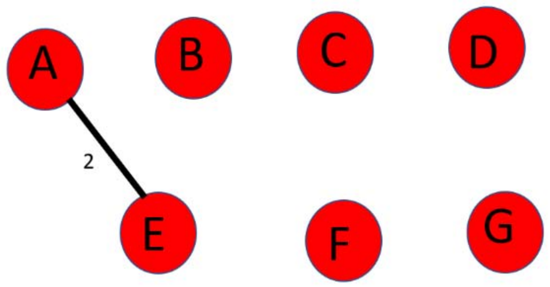
Step 1:

First write all edges weight in Ascending order:

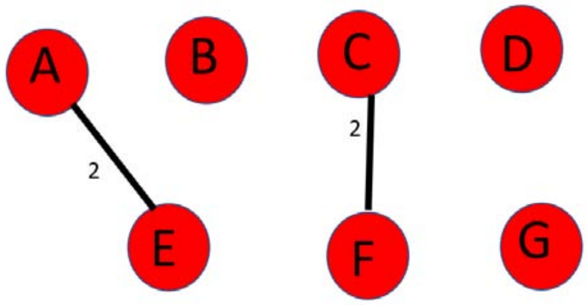
2,2,3,3,3,3,4,5,6,10



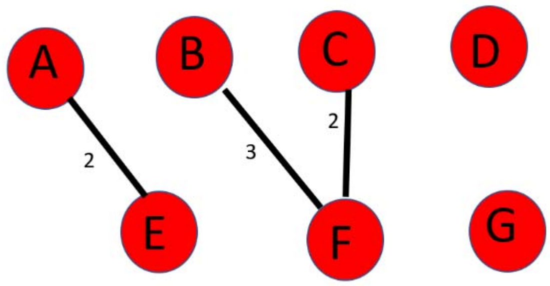
Step 2:



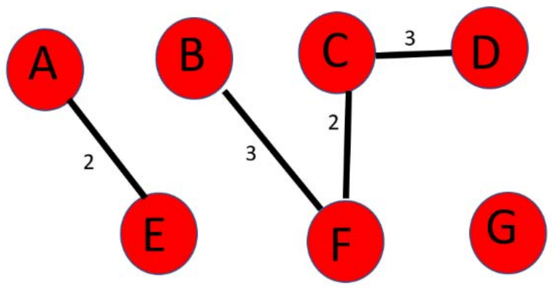
Step 3:



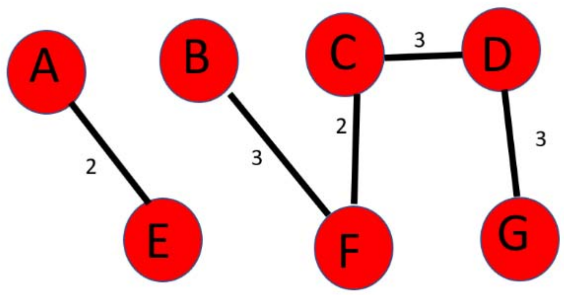
Step 4:



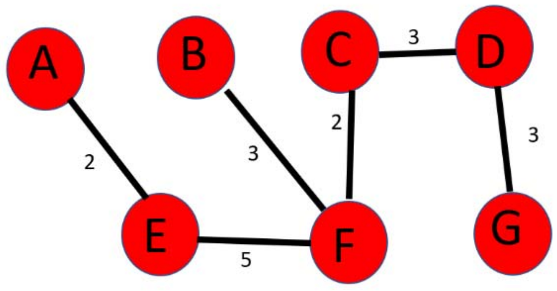
Step 5:



Step 6:

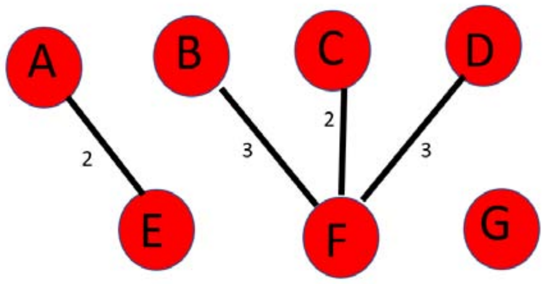


Step 7:

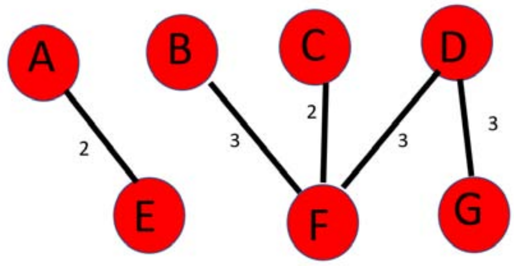


Total weight of MST is 18 and one more path or another MST

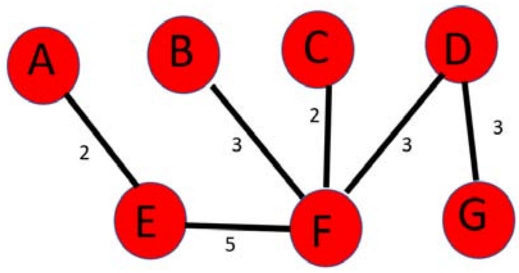
Step 8:



Step 9:



Step 10:



It also have weigth is 18 so both way are right.

VI. Find the Time Complexity of prims's Algorithm we Follow Step by step

1. Set up a minimum priority queue (heap) and add the first vertex to it.
2. Do the following while the queue is still not empty:
   a. Take the queue's minimum weight edge.
   b. If the edge connects a vertex that has not yet been visited, add it to the minimum spanning tree and mark the vertex as visited.

c. Add all adjacent edges of the newly visited vertex to the queue if they connect to unvisited vertices.

Assume the input graph contains V vertices and E edges. Prim's algorithm's time complexity can be calculated as follows:

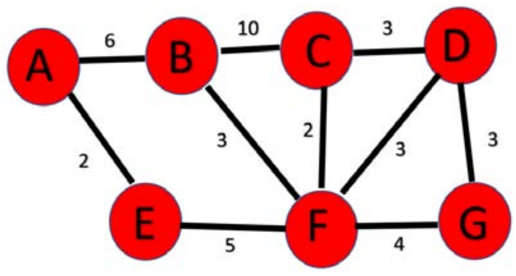
- It takes O(V) time to initialize the priority queue.
- It takes O(log V) time to extract the minimum weight edge from the queue.
- It takes constant time to check if a vertex has been visited.
- Adding an edge to the minimum spanning tree takes an infinite amount of time.
- It takes constant time to mark a vertex as visited.
- Because each edge can only be added to the queue once, adding adjacent edges to the queue takes O(E log V) time.

Prim's algorithm has a total time complexity of O((V-1) log V + E log V) because step 2 is repeated V-1 times. In practise, the term E log V dominates the time complexity, so Prim's algorithm has an overall time complexity of O. (E log V).

It should be noted that this analysis assumes that the priority queue is implemented using a binary heap. The time complexity may differ slightly if a Fibonacci heap or another data structure is used.

Problem Statement

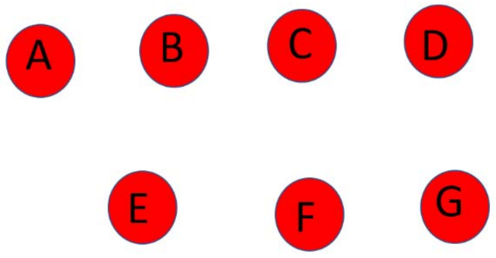
Find the MST through Kruskal's Algorithm and Explain it step wise.



Solution:

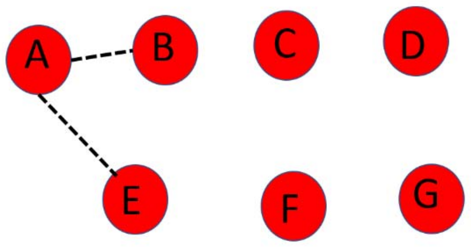
Step 1:

In the Kruskal algorithm first we assign first a node as head or starting point to start the finding MST and show all possible way from that node

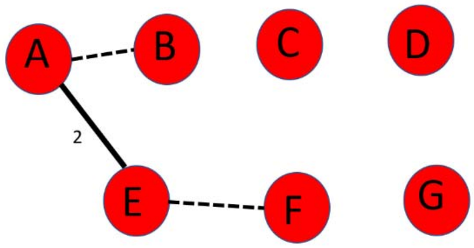


Here I take as a head or starting point node is A

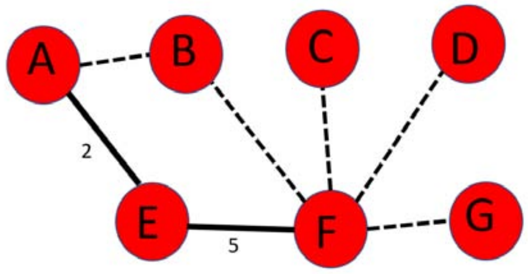
Step 2:



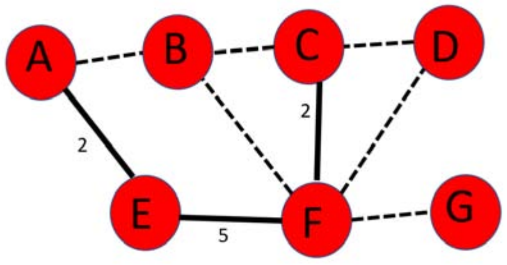
Step 3:



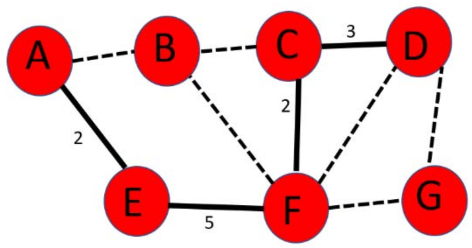
Step 4:



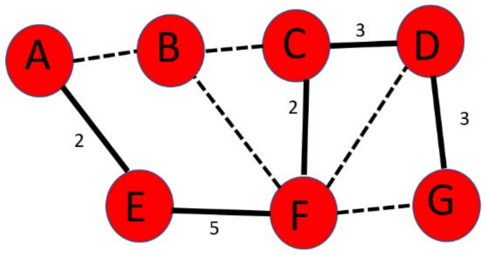
Step 5:



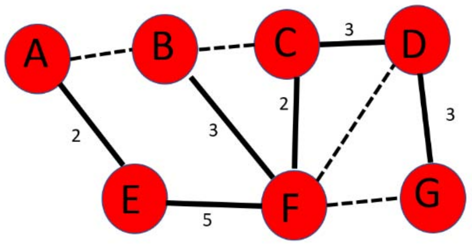
Step 6:



Step 7:

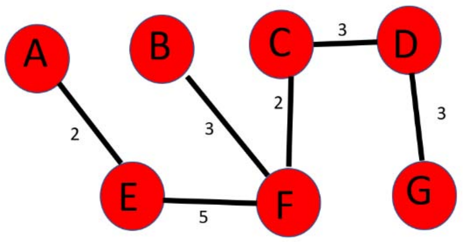


Step 8:



We get MST through it which weight is 18

Final view is this



VII. Find the time Complexity of Kruskal Algorithm we Follow Step by Step

1. Sort the edges in increasing weight order.
2. To keep track of connected components, create a disjoint-set data structure.
3. Do the following for each edge, in increasing order of weight:
   a. Add the edge to the minimum spanning tree and merge the two components if it connects two vertices that are not in the same connected component.
   b. Otherwise, throw away the edge.

Assume the input graph contains V vertices and E edges. Kruskal's algorithm's time complexity can be calculated as follows:

- It takes O(E log E) time to sort the edges.
- It takes O(V) time to initialise the disjoint-set data structure.
- Using the disjoint-set data structure, determining whether two vertices are in the same connected component takes O(log V) time.
- Using the disjoint-set data structure, merging two connected components takes O(log V) time.
- Adding an edge to the minimum spanning tree takes an infinite amount of time.
- It takes time to discard an edge.

Kruskal's algorithm has a total time complexity of O(E log E + V log V) because step 3 is repeated E times. In practise, the term E log E dominates the time complexity, so Kruskal's algorithm has an overall time complexity of O. (E log E).

It should be noted that this analysis assumes that the edges are sorted using a standard sorting algorithm such as quicksort or mergesort. The time complexity may differ slightly if a radix sort or another algorithm is used. Furthermore, the time complexity of the disjoint-set data structure is determined by the implementation used.

VIII. To Find the Space Complexity of Prim's and Kruskal's Algorithm

The Prim Algorithm:

- To store edges, the minimum priority queue (heap) requires O(E) space.
- The boolean array used to mark visited vertices takes up O(V) space.
- The minimum spanning tree takes up O(V) space.
- Prim's algorithm has a total space complexity of O(V + E).

The Kruskal Algorithm:

- The disjoint-set data structure used to keep connected components together takes up O(V) space.
- The array used to store the edges takes up O(E) space.
- The minimum spanning tree takes up O(V) space.
- Kruskal's algorithm has an overall space complexity of O(V + E).

It should be noted that the above space complexity calculations assume a standard implementation of each algorithm. However, depending on the implementation, the space complexity can vary. Additionally, the input graph may require space, but this is typically considered a separate factor and is not included in the algorithm's space complexity calculation.

IX. Comparision Between Prim's and Kruskal's Algorithm

1. Prim's algorithm is a greedy algorithm that starts with a single vertex and gradually adds edges to form a minimum spanning tree. Kruskal's algorithm is also a greedy algorithm that begins with the edge with the smallest weight and gradually adds edges to form a minimum spanning tree.
2. Time complexity: The time complexity of Prim's algorithm is O(E log V), where E is the number of edges and V is the number of vertices in the graph.

The time complexity of Kruskal's algorithm is O(E log E) or O(E log V), depending on the implementation.

3. Prim's algorithm has a space complexity of O(V), where V is the number of vertices in the graph. Kruskal's algorithm has a space complexity of O(E), where E is the number of edges in the graph.
4. Prim's algorithm always generates a connected minimum spanning tree. If the graph is not connected, Kruskal's algorithm can generate a forest of minimum spanning trees.
5. Edge selection: Prim's algorithm chooses the edges with the lowest weight among all those that connect the tree to a non-tree vertex. Kruskal's algorithm chooses edges based on the lowest weight among all edges that have yet to be added to the tree.
6. Performance: Prim's algorithm performs better on dense graphs (where E is close to V2), while Kruskal's algorithm performs better on sparse graphs (where E is much less than V2).

Overall, both the Prim and Kruskal algorithms are effective and widely used for determining minimum spanning trees. The algorithm chosen is determined by the characteristics of the input graph and the specific requirements of the problem.

Time Complication:

- The time complexity of Prim's algorithm is O(E log V), where E is the number of edges and V is the number of vertices in the graph. In dense graphs where E is close to V2, this complexity outperforms Kruskal's algorithm.
- In sparse graphs, where E is much less than V2, Kruskal's algorithm has a time complexity of O(E log E), which is better than Prim's algorithm.

Space Complexity:

- Prim's algorithm takes up O(V + E) space, and Kruskal's algorithm takes up O(V + E) space as well. As a result, the space complexity of both algorithms is comparable.

There are several alternative algorithms for determining the minimum spanning tree, each with its own time and space complexities:

- Boruvka's algorithm has a time complexity of O(E log V), making it faster in dense graphs than Prim's algorithm. However, it requires O(E log V) space, which is greater than that required by Prim's algorithm.
- Although the Reverse-Delete algorithm has an O(E2) time complexity, it performs well on sparse graphs. It has an O(V + E) space complexity.
- The time complexity of Randomized Prim's algorithm is similar to that of Prim's algorithm, but it can be faster in practise due to the randomised nature of its implementation.

Overall, the algorithm chosen is determined by the properties of the graph being processed. Prim's or Boruvka's algorithms may be preferable for dense graphs. Kruskal's algorithm or the Reverse-Delete algorithm may be preferable for sparse graphs. Randomized Prim's algorithm is another viable option in practise.

X. Conclusion

Finally, Prim's and Kruskal's algorithms are two well-known algorithms for determining the minimum spanning tree of a weighted, connected graph.

Prim's algorithm employs a greedy approach, beginning with a single vertex and expanding the minimum spanning tree one edge at a time. At each step, the algorithm maintains a priority queue to select the edge with the lowest weight. Prim's algorithm has a time complexity of O(E log V), where E is the number of edges and V is the number of vertices in the graph.

Kruskal's algorithm, on the other hand, employs a greedy approach but works by adding edges to the minimum spanning tree in increasing weight order while avoiding cycles. To keep connected components and check for cycles, the algorithm employs a disjoint-set data structure. Kruskal's algorithm has a time complexity of O(E log E), where E is the number of edges in the graph.

Both algorithms require O(V + E) space in terms of complexity.

Overall, the algorithm chosen is determined by the properties of the graph being processed. Prim's algorithm is preferable for dense graphs, whereas Kruskal's algorithm is preferable for sparse graphs. Other algorithms, such as Boruvka's algorithm and the Reverse-Delete algorithm, can also be used depending on the problem's specific requirements.

References Références Referencias

1. B.Hughes. Trees and ultra metric spaces: a categorical equivalence [J]. Advances in Mathematics, 2004, 189(1):148-191
2. M. J. Naylor, L.C.Rose, B. J.Moyle. Topology of foreign exchange markets using hierarchical structure methods [J]. Physica A: Statistical Mechanics and its Applications, 2007, 382(1):199–208.
3. J.G. Brida, W. A. Risso. Multidimensional minimal spanning tree: The Dow Jones case [J]. Physica A: Statistical Mechanics and its Applications, 2008, 387(21):5205-5210.
4. Martel. The expected complexity of Prim's minimum spanning tree algorithm [J]. Information Processing Letters, 2002 ,81(4):197-201.
5. Yang Guo Hui, Zhou Chun Guang. An algorithm for clustering gene expression data using minimum

spanning tree [J]. Journal of Computer Research and Development, 2003, 40(10):1431-1435.

6. Feixue Huang. Comparison of Prim and Kruskal on Shanghai and Shenzhen 300 Index hierarchical structure tree, 2009,237-240
7. Michael Laszlo and Sumitra Mukherjee, Member, IEE . Minimum Spanning Tree Partitioning Algorithm for Micro aggregation, July 2005 ,902-904
8. Peace Ayegba. A Comparative Study of Minimal Spanning Tree Algorithms, 2020
9. Jogamohan Medak. Review and Analysis of Minimum Spanning Tree Using Prim's Algorithm, 2018
10. Kenneth Sorensen. An Algorithm to Generate all Spanning Trees of a Graph in Order of Increasing Cost, 2005
11. Arogundade O.T. Prim Algorithm to Improving Local Access Network in Rural Areas, 2011
12. Harvey J. Greenberg. Greedy Algorithm for Minimum Spanning Tree, 1998