Proceedings of the 5th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/19/20231012

The analysis and application of Prim algorithm, Kruskal algorithm, Boruvka algorithm

Jiayu Chen

Beijing Normal University - Hong Kong Baptist University United International College, Zhuhai, China, 519087

maxchenjiayu@163.com

Abstract. Minimum spanning tree has many applications in real life. For example, the government needs to build roads between many cities. Therefore, it is necessary to find the plan with the shortest path to save the cost. The problem is essentially generating a minimal spanning tree, and it require a suitable algorithm to find the minimum spanning tree. In this paper, the author analyzes the structure and time complexity of the Prim algorithm, the Kruskal algorithm and the Boruvka algorithm. Through this research, the author finds Prim algorithm is suitable for dense graphs. The Kruskal algorithm can generate the minimum spanning tree in sparse tree. And the Boruvka algorithm is suitable for graphs that have some special characters. Based on the above conclusions, the author gives some suggestions for urban highway network planning.

Keywords: minimum spanning tree, Prim algorithm, Kruskal algorithm, Boruvka algorithm, graph.

1. Introduction

Minimum spanning tree is the spanning tree that has the minimum weight in an undirected connected weight graph. It can be used in many fields. Minimum spanning trees have many important applications. For example, the government need to build roads between several cities, so that any two cities can be connected by road (but not necessarily directly, as long as the other city can be reached indirectly by road). And the government needs to find the plan with the shortest total distance to save cost. This requires finding the minimum spanning tree with weights.

At present, there are three mainstream algorithms for generating minimum spanning trees, which are respectively the Prim algorithm, the Kruskal algorithm and the Boruvka algorithm. This paper will study and compare the characteristics of these three algorithms and give some application suggestions in traffic fields.

2. Analysis and application of Prim algorithm, Kruskal algorithm, Boruvka algorithm

2.1. Prim algorithm

2.1.1. The description of Prim algorithm. It is a kind of algorithm to find the minimum spanning tree, which was first found by Vojtěch Jarník. It was also independently discovered by American computer

DOI: 10.54254/2755-2721/19/20231012

scientist Robert Prim in 1957. It was discovered again in 1959 by Etzger Dikoscher. Therefore, sometimes the Prim algorithm is also called the DJP algorithm [1].

Here are the steps of Prim algorithm [2].

Step 1:

Choose any vertex r and suppose it as the root of minimum. V is the set of all vertices. Set V_{all} is the set of all vertices in the graph. E is the set of all edges in the graph. Make $V = \{r\}$ and $E = \{\emptyset\}$.

Step 2:

Find the edge with the minimum weight which one of its vertex is in set V and the other vertex is in set $V_{all} \backslash V$. Then this edge should be added into E.

Step 3:

If $V_{all} \backslash V = \emptyset$, then the loop will stop. And the minimum spanning tree (V, E) would output. If $V_{all} \backslash V \neq \emptyset$, go back to step 2.

2.1.2. The prove of Prim algorithm. The Prim algorithm can be proved by induction.

Before the prove steps, it needs an underlying conclusion: in a connected graph G, for each point, the edge with the smallest weight connected to itself must belong to G's spanning tree. This conclusion can be proved by reduction to absurdity. Suppose vertex v is any vertex of the graph and edge e is the edge with the smallest weight that is connected v. If any other edge that connects with v is chosen to compose the spanning tree. Then the weight of the spanning tree is definitely larger than the minimum spanning tree, which has an edge e. Therefore, the conclusion is true.

Make set V to store the vertex. Set V_{all} is the set of all vertices in the graph. Set E is the set of edges. Choose any vertex as the root. Add it to V.

Now $|V| = 1$. Choose the edge which has the smallest weight. Add this edge into E. In the meantime, add the other vertex, which is connected by the edge. Because of the conclusion that was proved just now, tree (V, E) is a part of the minimum spanning tree.

Then when $|V| = k\ (1 \leq k \leq V_{all})$, suppose tree (V, E) is a part of the minimum spanning tree. Here (V, E) can be regarded as a vertex because it has some of the same attributes with vertex. They are all part of the minimum spanning tree. And they all have some edges that connect with each other. When $|V| = k + 1$, choose the edge in $V_{all} \backslash V$ to add into E. According to the conclusion above, now (V, E) is still part of the minimum spanning tree. Hence, the spanning tree that is obtained by Prim algorithm is the minimum spanning tree.

2.1.3. Time complexity of Prim algorithm. The average time complexity of the heap optimization Prim algorithm can be understood as the sum of the time complexity of these part:

Step 1: Time complexity of initializing the heap is $O(|V|)$.

Step 2: Ttime complexity of removing the smallest edge from the heap is $O(\log |V|)$. Each edge needs to be operated once, so time complexity of this step is $O(|E| \log |V|)$.

Step 3: Time complexity of adding a new vertex to the heap is $O(\log |V|)$. Each vertex needs to perform an operation, so the time complexity of this step is $O(|V| \log |V|)$.

The sum is $O(|V| + |E| \log |V| + |V| \log |V|)$. For this reason, the total time complexity is $O(|E| \log |V|)$.

2.2. Kruskal algorithm

2.2.1. The description of Kruskal algorithm. Joseph Kruskal find Kruskal Algorithm in 1956. It is a kind of greedy algorithm [3].

Here are the specific steps of the Kruskal Algorithm [4].

Step 1:

Sort the edges by their weight. Add them into E_{all} by sorting. Build a set E to store the edges that form the minimum spanning tree. Set V stores all vertices in the graph.

DOI: 10.54254/2755-2721/19/20231012

Step 2:

Then choose the edge with the smallest weight from $E_{all} \backslash E$. If the edge that is chosen does not form a cycle with the edges in E, add the edge into E.

Step 3:

If $E_{all} \backslash E$ is empty, stop and return (V, E). If $E_{all} \backslash E$ is not empty, go back to Step 2.

2.2.2. The prove of Kruskal algorithm. Suppose T is the spanning tree that is generate by the Kruskal algorithm, and T' is the minimum spanning tree of graph G. Suppose $w(e)$ is the weight of edge e and $W(T)$ is the total weight of T. If it can be proved that $W(T) = W(T')$, then it is proved that Kruskal can get the minimum spanning tree.

Here the author uses contradiction to prove it.

Suppose $W(T) \neq W(T')$, then there absolutely is an edge which is in T and it is not in T'. Suppose there are k edges like this. And e is the edge in k edges which has minimum weight.

e is not in T'. It means that adding e into T will generate a circle.

Then find the edge e' in the circle that has the minimum weight, and it is only in T'.

If $w(e) = w(e')$, then the Kruskal algorithm can get the minimum spanning tree.

If $w(e) > w(e')$, it is impossible because the Kruskal algorithm would choose the edge with a smaller weight.

If $w(e) < w(e')$, it is also impossible because it means $W(T) < W(T')$.

In conclusion, the Kruskal algorithm can get the minimum spanning tree.

2.2.3. Time complexity of Kruskal algorithm. Sorting edges takes $O(|E| \log |E|)$, where $|E|$ is the number of edges in the graph.

Initializing the disjoint set needs $O(|V|)$, where $|V|$ is the number of vertices in the graph.

For each edge, we perform a find operation and two union operations on the disjoint set. Each find or union operation takes $O(\log|V|)$ with a balanced union-find data structure. Therefore, the total time complexity of all these operations is $O(|E| \log |V|)$.

Finally, constructing the MST takes $O(|E|)$ time, since we are adding $|E|$ edges to the MST.

Overall, the average time complexity of the Kruskal algorithm is $O(|E| \log|E|)$. In worst case, E can be as large as $|V|^2$, so the time complexity can be $O(|V|^2 \log |V|)$, but this is rare in practice. Note that time complexity of the Kruskal algorithm can be further improved by using some special data structures such as Fibonacci heaps or priority queues, which can reduce the sorting time to $O(|E| + |V| \log |V|)$ or even $O(|E| + \log |V|)$ in some cases.

2.3. Boruvka algorithm

2.3.1. The description of Boruvka algorithm. The Boruvka Algorithm is an old algorithm. When Otakar Boruvka put forward the algorithm in 1926 [5], the modern general-purpose computer had not even been invented. But with the development of computer science and mathematics, people found that this algorithm had some unique advantages.

Here are the steps for the Boruvka algorithm to find a minimum spanning tree [6].

Step 1:

Initialization: Treat each vertex as a separate connected component.

Step 2:

For each component, find the minimum edge that is adjacent to that component. These edges are called “light edges”.

Step 3:

For each component, merge it with the component connected by its light edge. This will reduce the number of components.

Step 4:

Repeat step 2 and step 3 until there is only one connected component left.

DOI: 10.54254/2755-2721/19/20231012

2.3.2. The prove of Boruvka algorithm. It can be proved by induction.

Base case: For a graph with a vertex set of size 1, its minimum spanning tree has only one vertex, so the Boruvka algorithm can certainly find the minimum spanning tree correctly.

Induction hypotheses: the Boruvka algorithm can correctly find the minimum spanning tree of any graphs with a vertex set size less than k. Now consider a graph with a vertex set size of k.

Induction step: First, the Boruvka algorithm treats each vertex as a separate connected component and finds the minimum edge for each connected component. These minimum edges divide the graph into multiple connected components. Since the size of each connected component is less than k, according to the induction hypothesis, we can find minimum spanning tree of each connected component.

Now we shrink these connected components into a single vertex to form a new graph. In this new graph, we can consider minimum spanning tree of each original connected component as an edge. Since the original graph is connected, the new graph is also connected. Furthermore, since the number of edges in the minimum spanning tree of each connected component does not exceed $k-1$, the number of edges in new graph does not exceed $k-1$.

Therefore, according to the induction hypothesis, the Boruvka algorithm can find the minimum spanning tree of the new graph. To acquire the set of edges from the original graph, we can expand the minimum spanning tree edges of the new graph. These edges make up the original graph's spanning tree, which in accordance with a Boruvka algorithm property, is the original graph's smallest spanning tree.

In summary, the Boruvka algorithm can correctly find the minimum spanning tree of any connected graph or undirected graph.

2.3.3. Time complexity of Boruvka algorithm. Firstly, the outer loop repeats n times, where n is the number of vertices in the graph. The inner loop processes the neighbors of each vertex, so it scans each edge at most twice, once starting from each vertex and once ending at each vertex. Therefore, the number of times the inner loop is executed is $2|E|$.

In each iteration of the inner loop, we need to search for the minimum element in the cheapest array, which takes $O(|V|)$ time. Therefore, the total time for the inner loop is $O(|V||E|)$.

After each iteration of the inner loop, we need to merge the connected components into larger connected components. In the worst case, initially all vertices are isolated, so each vertex is a connected component. After every iteration, the number of connected components will halve. Therefore, the number of iterations required to merge all the connected components is logarithmic in V.

Therefore, the time complexity of Boruvka algorithm is $O(n|V||E|\log|V|) = O(|E|\log|V|)$.

3. The comparison of the Prim algorithm, Kruskal algorithm and Boruvka algorithm and the suggestions on traffic field

3.1. The character of the Prim algorithm, Kruskal algorithm and Boruvka algorithm

3.1.1. Advantages of the Prim algorithm, Kruskal algorithm and Boruvka algorithm. The advantages of the Prim algorithm: when the graph is a dense graph ($|E|$ is closed to $|V|^2$), time complexity of the Prim algorithm is $O(|V|^2)$, which is relatively quicker [7].

The advantages of the Kruskal algorithm: when it works with sparse graphs, the time complexity is $O(|E|\log|E|)$, which is relatively outstanding [8].

The advantages of the Boruvka algorithm: in graphs that have some special attributes, the Boruvka algorithm performs excellently. The graphs with some special attributes refer to the connectivity and weight distribution of the graph. For example, when the graph has a relatively uniform weight distribution, the Boruvka algorithm can quickly converge to the minimum spanning tree because it selects all edges with the minimum weight at each round of edge selection. Kruskal and Prim

DOI: 10.54254/2755-2721/19/20231012

algorithms may require more comparison operations when processing graphs with a uniform weight distribution because they need to select the minimum edge at each step. In addition, the Boruvka algorithm is also suitable for some large graphs with a lot of edges and vertices. In this situation, the parallel computing method of the Boruvka algorithm has advantages over the linear calculation methods of the Prim algorithm and the Kruskal algorithm [9].

3.1.2. Disadvantages of the Prim algorithm, Kruskal algorithm and Boruvka algorithm. The disadvantages of the Prim algorithm: when the Prim algorithm deals with sparse graphs ($|E| \ll |V|^2$), it is less efficient. And it requires heap optimization to achieve, which increases the complexity of the implementation [10]. The algorithm may not be suitable for some special cases.

The disadvantages of the Kruskal algorithm: the Kruskal algorithm needs to sort the edges by weights firstly. For dense graphs, the Kruskal algorithm might be slower than the Prim algorithm [11].

Disadvantages of the Boruvka algorithm: the Boruvka algorithm needs to deal with every connected component, which might take a lot of time.

3.2. Some suggestions on traffic fields

This part will give some suggestions in the field of transportation in combination with the description in the previous part.

Minimum spanning tree has a wide use range in real-life. It can be used to plan cables, a network of pipes for drinking water or natural gas, or the establishment of urban transportation networks.

Here, the author uses establishing urban transportation networks as an example. Some places have rugged terrain, which means that there are many places that are not suitable for building roads. Relatively, the map of transportation routes in these countries will be sparse graph. In this case, the Kruskal algorithm can help planners find transportation networks that connect every city faster. On the contrary, for some places with flat terrain, which means there are more planned routes to select, the Prim algorithm would be more suitable for planning the road network design in these countries. If the distances between cities in a region are similar, the Boruvka algorithm is more suitable.

Of course, the Prim algorithm, Kruskal algorithm and Boruvka algorithm are essentially greedy algorithms. They can only help users get a local, optimal solution. In many cases, users still need to convert the local optimal solution into a global optimal solution based on the actual situation.

4. Conclusion

This paper studies the basic structure and time complexity of the three algorithms and summarizes several different situations applicable to the three algorithms. The Prim algorithm is suitable for dense graphs. The Kruskal algorithm performs better in sparse graphs. The Boruvka algorithm is more appropriate for some graphs that have special properties. Based on the above research, the author gives suggestions for traffic route planning in different situations in reality.

This paper mainly discusses the Prim algorithm, Kruskal algorithm, and Boruvka algorithm, and does not cover more algorithms for generating minimal spanning trees. In the future, additional case studies may be carried out after field research or after obtaining more access to literature.

References

[1] Iqbal, M., Siahaan, A. P. U., Purba, N. E., & Purwanto, D. (2017). Prim's Algorithm for Optimizing Fiber Optic Trajectory Planning. Int. J. Sci. Res. Sci. Technol, 3(6), pp. 504-509.

[2] Medak, J. (2018). Review and Analysis of Minimum Spanning Tree Using Prim's Algorithm. International Journal of Computer Science Trends and Technology (IJCST),Volume 6.

[3] Berger, R., Dubuisson, S., & Gonzales, C. (2012). Fast multiple histogram computation using Kruskal's algorithm. In 2012 19th IEEE International Conference on Image Processing, pp. 2373-2376. IEEE.

[4] Greenberg, H. J. (1998). Greedy algorithms for minimum spanning tree. University of Colorado at Denver.

DOI: 10.54254/2755-2721/19/20231012

[5] Mariano, A., Proenca, A., & Sousa, C. D. S. (2015). A generic and highly efficient parallel variant of boruvka's algorithm. In 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pp. 610-617. IEEE.

[6] Bergantiños, G., & Vidal-Puga, J. (2011). The folk solution and Boruvka's algorithm in minimum cost spanning tree problems. Discrete applied mathematics, 159(12), pp. 1279-1283.

[7] Jarvis, J. P., & Whited, D. E. (1983). Computational experience with minimum spanning tree algorithms. Operations Research Letters, 2(1), pp. 36-41.

[8] Kershenbaum, A., & Van Slyke, R. (1972). Computing minimum spanning trees efficiently. In Proceedings of the ACM annual conference, 1, pp. 518-527.

[9] Deng, J., Wang, W., Quan, S., Zhan, R., & Zhang, J. (2022). Hierarchical Superpixel Segmentation for PolSAR Images Based on the Boruvka Algorithm. Remote Sensing, 14(19), p. 4721.

[10] Huang, F., Gao, P., & Wang, Y. (2009). Comparison of Prim and Kruskal on Shanghai and Shenzhen 300 Index hierarchical structure tree. In 2009 International Conference on Web Information Systems and Mining, pp. 237-241. IEEE.

[11] Paryati and Salahddine Krit. (2021). The Implementation of Kruskal's Algorithm for Minimum Spanning Tree in a Graph. MATEC Web of Conferences, 348.