Minimum Spanning Trees[*]

Bang Ye Wu   Kun-Mao Chao

1 Introduction

Suppose you have a business with several branch offices and you want to lease phone lines to connect them with each other. Your goal is to connect all your offices with the minimum total cost. The resulting connection should be a spanning tree since if it is not a tree, you can always remove some edges without losing the connectivity to save money.

A minimum spanning tree (MST) of a weighted graph G is a spanning tree of G whose edges sum to minimum weight. In other words, a minimum spanning tree is a tree formed from a subset of the edges in a given undirected graph, with two properties: (1) it spans the graph, i.e., it includes every vertex in the graph, and (2) it is a minimum, i.e., the total weight of all the edges is as low as possible.

The minimum spanning tree problem is always included in algorithm textbooks since (1) it arises in many applications, (2) it is an important example where greedy algorithms always deliver an optimal solution, and (3) clever data structures are necessary to make it work efficiently.

What is a minimum spanning tree for the weighted graph in Figure 1? Notice that a minimum spanning tree is not necessarily unique.

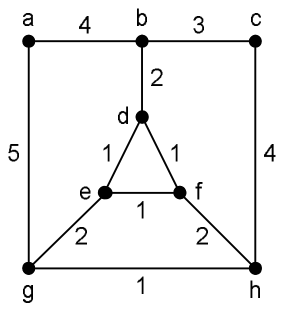


Figure 1: A weighted graph.

Figure 2 gives four minimum spanning trees, where each of them is of total weight 14. These trees can be derived by growing the spanning tree in a *greedy* way.

Before exploring the MST algorithms, we state some important facts about spanning trees. Let $G + e$ denote the graph obtained by inserting edge e into G.

Lemma 1: Any two vertices in a tree are connected by a unique path.

[*]An excerpt from the book "Spanning Trees and Optimization Problems," by Bang Ye Wu and Kun-Mao Chao (2004), Chapman & Hall/CRC Press, USA.

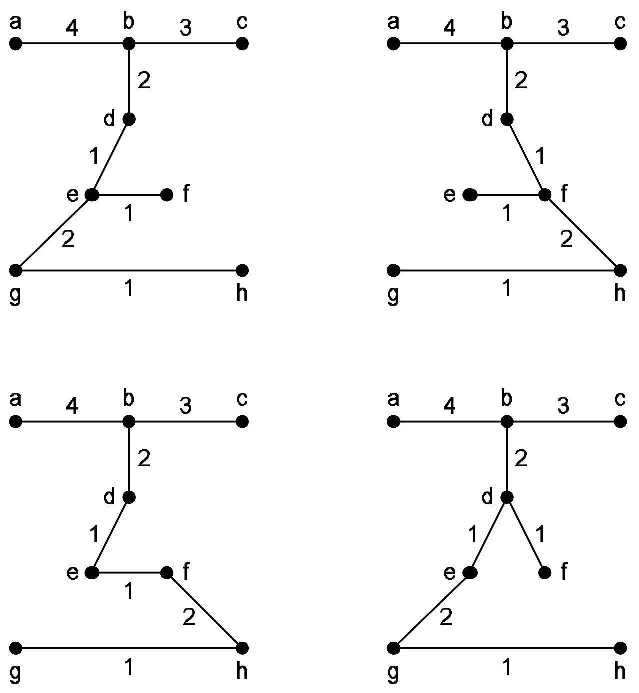


Figure 2: Some minimum spanning trees.

Proof: Since a tree is connected, any two vertices in a tree are connected by at least one simple path. Let T be a tree, and assume that there are two distinct paths P_1 and P_2 from vertex u to vertex v. There exists an edge $e = (x, y)$ of P_1 that is not an edge of P_2. It can be seen that $(P_1 \cup P_2) - e$ is connected, and it contains a path P_{xy} from vertex x to vertex y. But then $P_{xy} + e$ is a cycle. This is a contradiction. Thus, there can be at most one path between two vertices in a tree. $\square$

Lemma 2: Let T be a spanning tree of a graph G, and let e be an edge of G not in T. Then $T + e$ contains a unique cycle.

Proof: Let $e = (u, v)$. Since T is acyclic, each cycle of $T + e$ contains e. Moreover, X is a cycle of $T + e$ if and only if $X - e$ is a path from u to v in T. By Lemma 1, such a path is unique in T. Thus $T + e$ contains a unique cycle. $\square$

In this chapter, we shall examine three well-known algorithms for solving the minimum spanning tree problem: Borůvka's algorithm, Prim's algorithm, and Kruskal's algorithm. They all exploit the following fact in one way or another.

Theorem 3: Let $F_1, F_2, \ldots, F_k$ be a spanning forest of G, and let (u, v) be the smallest of all edges with only one endpoint $u \in V(F_1)$. Then there is an optimal one containing (u, v) among all spanning trees containing all edges in $\cup_{i=1}^{k} E(F_i)$.

Proof: By contradiction. Suppose that there is a spanning tree T of G with $\cup_{i=1}^{k} E(F_i) \subseteq E(T)$, and $(u, v) \notin E(T)$, which is smaller than all spanning trees containing $\cup_{i=1}^{k} E(F_i) \cup \{(u, v)\}$. By Lemma 2, $T + (u, v)$ contains a unique cycle. Since $v \notin V(F_1)$, this cycle contains some vertices outside F_1. Thus there exists an edge (u', v'), different from (u, v), on this cycle such that $u' \in V(F_1)$

and $v' \notin V(F_1)$. Since this edge is no smaller than (u,v), and does not belong to $\cup_{i=1}^{k} E(F_i)$, $T+(u,v)-(u',v')$ is a new spanning tree with total weight no more than T. This is a contradiction. It follows that there is an optimal one containing (u,v) among all spanning trees containing all edges in $\cup_{i=1}^{k} E(F_i)$. □

2 Borůvka's Algorithm

The earliest known algorithm for finding a minimum spanning tree was given by Otakar Borůvka back in 1926. In a Borůvka step, every *supervertex* selects its smallest adjacent edge. These edges are added to the MST, avoiding cycles. Then the new supervertices, i.e., the connected components, are calculated by contracting the graph on the edges just added to the MST. This process is repeated until only one supervertex is left. In other words, there are $n-1$ edges contracted. The union of these edges gives rise to a minimum spanning tree.

Algorithm: Borůvka
Input: A weighted, undirected graph $G = (V, E, w)$.
Output: A minimum spanning tree T

```
    $T \leftarrow \emptyset$
    while $|T| < n-1$ do
        $F \leftarrow$ a forest consisting of the smallest edge incident to
            each vertex in $G$
        $G \leftarrow G \backslash F$
        $T \leftarrow T \cup F$
```

Figure 3 illustrates the execution of the Borůvka algorithm on the graph from Figure 1. In Figure 3(a), each vertex chooses the smallest incident edge without causing cycles. In Figure 3(b), vertices a, b, c, d, e, and f are contracted into one supervertex, and vertices g and h are contracted into the other supervertex. In Figure 3(d), these two supervertices are contracted into one supervertex. All the contracted edges constitute a minimum spanning tree as shown in Figure 3(e).

Notice that each Borůvka step reduces the number of vertices by a factor of at least two. Therefore the **while** loop will be executed at most $O(\log n)$ times. In each iteration, all the contraction can be done in $O(m)$ time. In total, the Borůvka algorithm has a running time of $O(m \log n)$.

3 Prim's Algorithm

Prim's algorithm was conceived by computer scientist Robert Prim in 1957. It starts from an arbitrary vertex, and builds upon a single partial minimum spanning tree, at each step adding an edge connecting the vertex nearest to but not already in the current partial minimum spanning tree. It grows until the tree spans all the vertices in the input graph. This strategy is greedy in the sense that at each step the partial spanning tree is augmented with an edge that is the smallest among all possible neighboring edges.

Algorithm: Prim

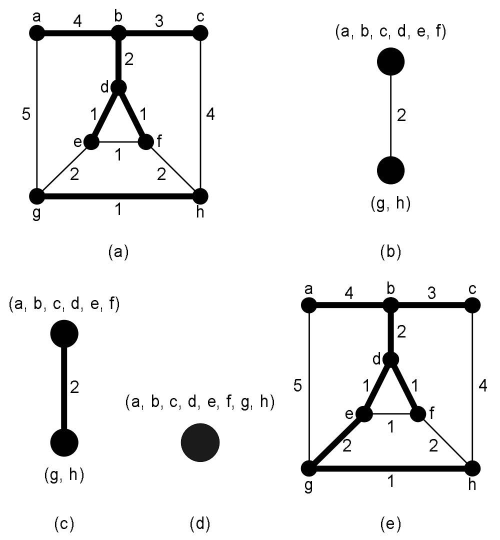


Figure 3: The execution of the BORŮVKA algorithm on the graph from Figure 1.

Input: A weighted, undirected graph $G = (V, E, w)$.
Output: A minimum spanning tree T.

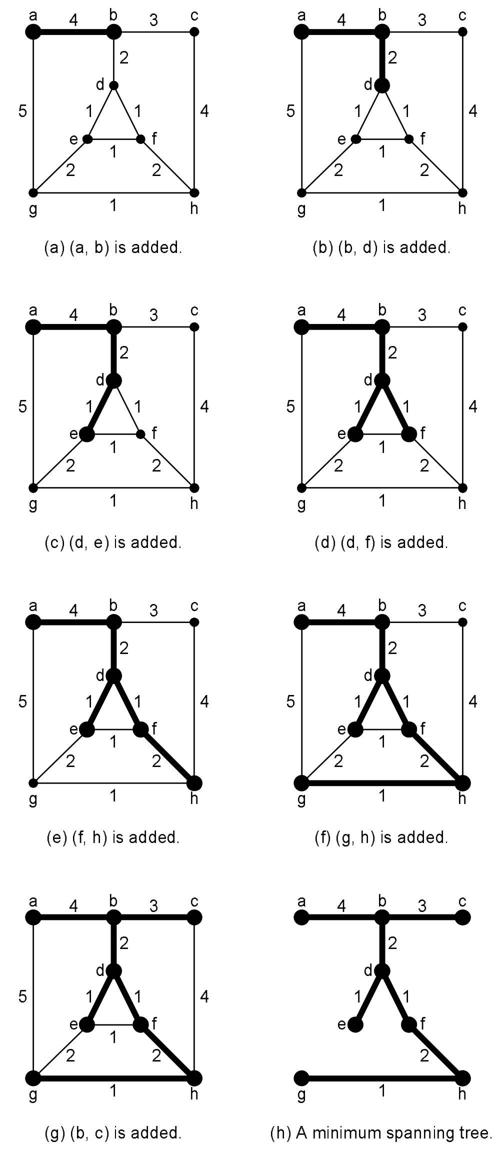
```
    $T \leftarrow \emptyset$
    Let $r$ be an arbitrarily chosen vertex from $V$.
    $U \leftarrow \{r\}$
    while $|U| < n$ do
        Find $u \in U$ and $v \in V - U$ such that the edge $(u, v)$ is a smallest
            edge between $U$ and $V - U$.
        $T \leftarrow T \cup \{(u, v)\}$
        $U \leftarrow U \cup \{v\}$
```

Figure 4 illustrates the execution of the PRIM algorithm on the graph from Figure 1. It starts at vertex a. Since (a, b) is the smallest edge incident to a, it is included in the spanning tree under construction (see Figure 4(a)). In Figure 4(b), (b, d) is added because it is the smallest edge between $\{a, b\}$ and $V - \{a, b\}$. When there is a tie, as in the situation in Figure 4(c), any smallest edge would work well. Proceed this way until all vertices are spanned. The final minimum spanning tree is shown in Figure 4(h).

Prim's algorithm appears to spend most of its time finding the smallest edge to grow. A straightforward method finds the smallest edge by searching the adjacency lists of the vertices in V; then each iteration costs $O(m)$ time, yielding a total running time of $O(mn)$. By using binary heaps, this can be improved to $O(m \log n)$. By using Fibonacci heaps, Prim's algorithm runs in $O(m + n \log n)$ time. Interested readers should refer to the end of this chapter for further improvements.

4 Kruskal's Algorithm

Kruskal's algorithm was given by Joseph Kruskal in 1956. It creates a forest where each vertex in the graph is initially a separate tree. It then sorts all the edges in the graph. For each edge (u, v)



(a) (a, b) is added.
(b) (b, d) is added.
(c) (d, e) is added.
(d) (d, f) is added.
(e) (f, h) is added.
(f) (g, h) is added.
(g) (b, c) is added.
(h) A minimum spanning tree.

Figure 4: The execution of the PRIM algorithm on the graph from Figure 1.

in sorted order, we do the following. If vertices u and v belong to two different trees, then add (u, v) to the forest, combining two trees into a single tree. It proceeds until all the edges have been processed.

Algorithm: KRUSKAL
Input: A weighted, undirected graph $G = (V, E, w)$.
Output: A minimum spanning tree T.

```
    Sort the edges in $E$ in nondecreasing order by weight.
    $T \leftarrow \emptyset$
    Create one set for each vertex.
    for each edge $(u, v)$ in sorted order do
        $x \leftarrow$ FIND($u$)
        $y \leftarrow$ FIND($v$)
        if $x \neq y$ then
            $T \leftarrow T \cup \{(u, v)\}$
            UNION($x, y$)
```

Figure 5 illustrates the execution of the KRUSKAL algorithm on the graph from Figure 1. Initially, every vertex is a tree in the forest. Let the sorted order of the edges be $\langle(d, e)$, (g, h), (e, f), (d, f), (b, d), (e, g), (f, h), (b, c), (a, b), (c, h), $(a, g)\rangle$. Since (d, e) joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees (see Figure 5(a)). Next we consider (g, h). Vertex g and vertex h belong to two different trees, thus (g, h) is added to the forest as shown in Figure 5(b). In Figure 5(d), when (d, f) is processed, both d and f belong to the same tree, therefore we do nothing for this edge. The final minimum spanning tree is shown in Figure 5(h).

Sorting the edges in nondecreasing order takes $O(m \log m)$ time. The total running time of determining if the edge joins two distinct trees in the forest is $O(m\alpha(m, n))$ time, where α is the functional inverse of Ackermann's function defined in [17]. Therefore the asymptotic running time of Kruskal's algorithm is $O(m \log m)$, which is the same as $O(m \log n)$ since $\log m = \Theta(\log n)$ by observing that $m = O(n^2)$ and $m = \Omega(n)$.

5 Applications

Minimum spanning trees are useful in constructing networks, by describing the way to connect a set of sites using the smallest total amount of wire. Much of the work on minimum spanning trees has been conducted by the communications company.

5.1 Cable TV

One example is a cable TV company laying cable to a new neighborhood. If it is constrained to bury the cable only along certain paths, then there would be a graph representing which points are connected by those paths. Some of those paths might be more expensive, because they are longer, or require the cable to be buried deeper. A spanning tree for that graph would be a subset of those paths that has no cycles but still connects to every house. There might be several spanning trees possible. A minimum spanning tree would be one with the lowest total cost.

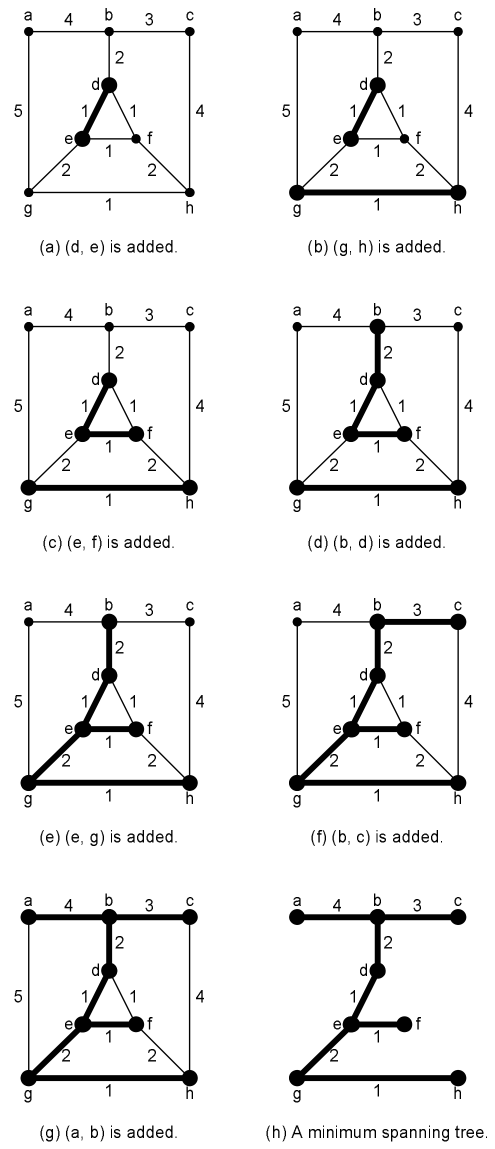


Figure 5: The execution of the Kruskal algorithm on the graph from Figure 1.

5.2 Circuit design

In the design of electronic circuitry, it is often necessary to wire some pins together in order to make them electrically equivalent. A minimum spanning tree needs the least amount of wire to interconnect a set of points.

5.3 Islands connection

Suppose we have a group of islands that we wish to link with bridges so that it is possible to travel from one island to any other in the group. Further suppose that the government wishes to spend the minimum amount on this project. The engineers are able to calculate a cost for a bridge linking each possible pair of islands. The set of bridges that will enable one to travel from any island to any other at the minimum cost to the government is the minimum spanning tree.

5.4 Clustering gene expression data

Minimum spanning trees also provide a reasonable way for clustering points in space into natural groups. For example, Ying Xu and his coworkers [19] describe a new framework for representing a set of multi-dimensional gene expression data as a minimum spanning tree. A key property of this representation is that each cluster of the gene expression data corresponds to one subtree of the MST, which rigorously converts a multi-dimensional clustering problem to a tree partitioning problem. They have demonstrated that, although the inter-data relationship is greatly simplified in the MST representation, no essential information is lost for the purpose of clustering. They observe that there are two key advantages in representing a set of multi-dimensional data as an MST. One is that the simple structure of a tree facilitates efficient implementations of rigorous clustering algorithms, which otherwise are highly computationally challenging. The other is that it can overcome many of the problems faced by classical clustering algorithms since an MST-based clustering does not depend on detailed geometric shape of a cluster. A new software tool called EXCAVATOR, which stands for "EXpression data Clustering Analysis and VisualizATiOn Resource," has been developed based on this new framework. The clustering results on the gene expression data (1) from yeast *Saccharomyces cerevisiae*, (2) in response of human fibroblasts to serum, and (3) of *Arabidopsis* in response to chitin elicitation are very promising.

5.5 MST-based approximations

In the traveling salesperson problem (TSP), we are given a complete undirected graph G that has weight function w associated with each edge, and we wish to find a tour of G with minimum weight. This problem has been shown to be NP-hard even when the weight function satisfies the triangle inequality, i.e., for all three vertices $x, y, z \in V$, $w(x, z) \leq w(x, y) + w(y, z)$. The triangle inequality arises in many practical situations. It can be shown that the following strategy delivers an approximation algorithm with a ratio bound of 2 for the traveling salesperson problem with triangle inequality. First, find a minimum spanning tree T for the given graph. Then double the MST and construct a tour T'. Finally, add shortcuts so that no vertex is visited more than once, which is done by a preorder tree walk. The resulting tour is of length no more than twice of the optimal. It can also be shown that an MST-based approach also provides a good approximation for the Steiner tree problems.

6 Summary

We have briefly discussed three well-known algorithms for solving the minimum spanning tree problem: Borůvka's algorithm, Prim's algorithm, and Kruskal's algorithm. All of them work in a "greedy" fashion.

For years, many improvements have been made for this classical problem. We close this chapter by sketching one possible extension to these three basic algorithms. First apply the contraction step in Borůvka's algorithm for $O(\log\log n)$ time. This takes $O(m\log\log n)$ time since each Borůvka's contraction step takes $O(m)$ time. After these contraction steps, the number of the supervertices of the contracted graph is at most $O(n/2^{\log\log n}) = O(n/\log n)$. Then apply Prim's algorithm to the contracted graph, which runs in time $O(m + (n/\log n)\log n) = O(m + n)$. In total, this hybrid algorithm solves the minimum spanning tree problem in $O(m\log\log n)$ time.

Bibliographic Notes and Further Reading

The history of the *minimum spanning tree* (MST) problem is long and rich. An excellent survey paper by Ronald Graham and Pavol Hell [9] describes the history of the problem up to 1985. The earliest known MST algorithm was proposed by Otakar Borůvka [1], a great Czech mathematician, in 1926. At that time, he was considering an efficient electrical coverage of Bohemia, which occupies the western and middle thirds of today's Czech Republic. In the mid-1950s when the computer age just began, the MST problem was attacked again by several researchers. Among them, Joseph Kruskal [14] and Robert Prim [16] gave two commonly used textbook algorithms. Both of them mentioned Borůvka's paper. In fact, Prim's algorithm was a rediscovery of the algorithm by the prominent number theoretician Vojtěch Jarník [10].

Textbook algorithms run in $O(m\log n)$ time. Andrew Chi-Chih Yao [20], and David Cheriton and Robert Tarjan [4] independently made improvements to $O(m\log\log n)$. By the invention of Fibonacci heaps, Michael Fredman and Robert Tarjan [7] reduced the complexity to $O(m\beta(m,n))$, where $\beta(m,n) = \min\{i | \log^i n \le m/n\}$. In the worst case, $m = O(n)$ and the running time is $O(m\log^* m)$. The complexity was further lowered to $O(m\log\beta(m,n))$ by Harold N. Gabow, Zvi Galil, Thomas H. Spencer, and Robert Tarjan [8].

On the other hand, David Karger, Philip Klein, and Robert Tarjan [11] gave a randomized linear-time algorithm to find a minimum spanning tree in the restricted random-access model. If the edge costs are integer and the models allow bucketing and bit manipulation, Michael Fredman and Dan Willard [6] gave a deterministic linear-time algorithm.

Given a spanning tree, how fast can we verify that it is minimum? Robert Tarjan [18] gave an almost linear-time algorithm by using path compression. János Komlós [13] showed that a minimum spanning tree can be verified in linear number of comparisons, but with nonlinear overhead to decide which comparisons to make. Brandon Dixon, Monika Rauch, and Robert Tarjan [5] gave the first linear-time verification algorithm. Valerie King [12] proposed a simpler linear-time verification algorithm. All these methods use the fact that a spanning tree is a minimum spanning tree if and only if the weight of each nontree edge (u, v) is at least the weight of the heaviest edge in the path in the tree between u and v.

It remains an open problem whether a linear-time algorithm exists for finding a minimum spanning tree. Bernard Chazelle [2] took a significant step towards a solution and charted out a new line of attack. His algorithm runs in $O(m\alpha(m,n))$ time, where α is the functional inverse of Ackermann's function defined in [17]. The key idea is to compute suboptimal independent sets in a nongreedy fashion, and then progressively improve upon them until an optimal solution is reached.

An approximate priority queue, called a *soft heap* [3], is used to construct a suboptimal spanning tree, whose quality is progressively refined until a minimum spanning tree is finally produced.

Seth Pettie and Vijaya Ramachandran [15] established that the algorithmic complexity of the minimum spanning tree problem is equal to its decision-tree complexity. They gave a deterministic, comparison-based MST algorithm that runs in $O(T^*(m,n))$, where $T^*(m,n)$ is the number of edge-weight comparisons needed to determine the MST. Because of the nature of their algorithm, its exact running time is unknown. The source of their algorithm's mysterious running time, and its optimality, is the use of precomputed "MST decision trees" whose exact depth is unknown but nonetheless provably optimal. A trivial lower bound is $\Omega(m)$; and the best upper bound is $O(m\alpha(m,n))$ [2].

References

[1] O. Borůvka. O jistém problému minimálním (about a certain minimal problem). *Práca Moravské Přirodovědecké Společnosti*, 3:37–58, 1926. (In Czech.).

[2] B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *J. ACM*, 47:1028–1047, 2000.

[3] B. Chazelle. The soft heap: an approximate priority queue with optimal error rate. *J. ACM*, 47:1012–1027, 2000.

[4] D. Cheriton and R. E. Tarjan. Finding minimum spanning trees. *SIAM J. Comput.*, 5:724–742, 1976.

[5] B. Dixon, M. Rauch, and R. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM J. Comput.*, 21:1184–1192, 1992.

[6] M. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48:424–436, 1994.

[7] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615, 1987.

[8] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:109–122, 1986.

[9] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Ann. Hist. Comput.*, 7:43–57, 1985.

[10] V. Jarník. O jistém problému minimálním (about a certain minimal problem). *ráca Moravské Přirodovědecké Společnosti*, 6:57–63, 1930.

[11] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42:321–328, 1995.

[12] V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18:263–270, 1997.

[13] J. Komlós. Linear verification for spanning trees. *Combinatorica*, 5:57–65, 1985.

[14] J. B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. Amer. Math. Soc.*, 7:48–50, 1956.

[15] S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49:16–34, 2002.

[16] R. C. Prim. Shortest connection networks and some generalizations. *Bell. Syst. Tech. J.*, 36:1389–1401, 1957.

[17] R. E. Tarjan. Efficiency of a good but not linear set-union algorithm. *J. ACM*, 22:215–225, 1975.

[18] R. E. Tarjan. Applications of path compressions on balanced trees. *J. ACM*, 26:690–715, 1979.

[19] Y. Xu, V. Olman, and D. Xu. Clustering gene expression data using a graph-theoretic approach: An application of minimum spanning trees. *Bioinformatics*, 18:536–545, 2002.

[20] A. Yao. An $O(|E| \log\log |V|)$ algorithm for finding minimum spanning trees. *Inf. Process. Lett.*, 4:21–23, 1975.