INVOLVE 2:4(2009)

Minimum spanning trees

Pallavi Jayawant and Kerry Glavin

(Communicated by Arthur T. Benjamin)

The minimum spanning tree problem originated in the 1920s when O. Borůvka identified and solved the problem during the electrification of Moravia. This graph theory problem and its numerous applications have inspired many others to look for alternate ways of finding a spanning tree of minimum weight in a weighted, connected graph since Borůvka's time. This note presents a variant of Borůvka's algorithm that developed during the graph theory course work of undergraduate students. We discuss the proof of the algorithm, compare it to existing algorithms, and present an implementation of the procedure in Maple.

1. Introduction

Minimum spanning trees (MSTs) have long been of interest to mathematicians because of their many applications. Most commonly, cable and communications companies can represent the task of connecting every house in a network in the least expensive way possible as an MST problem. In this case, the cost of laying cables between houses corresponds to the weights of the edges. There are analogous applications to transportation networks, such as determining the least expensive method of connecting a number of islands or bodies of land. For more applications, see [Wu and Chao 2004; Graham and Hell 1985].

Many algorithms have been developed over the years to find MSTs efficiently. The problem originated in the 1920s when O. Borůvka identified and solved the problem during the electrification of Moravia. However, the language of graph theory is not used to describe the algorithm in his papers from 1926 [Borůvka 1926a; 1926b] which have been translated recently into English [Nešetřil et al. 2001]. In the 1950s, many people contributed to the MST problem. Among them were R. C. Prim and J. B. Kruskal, whose algorithms are very widely used today. The algorithm known as Prim's algorithm was in fact discovered earlier by V. Jarník in 1930. A history of the MST problem appears in [Graham and Hell 1985; Nešetřil et al. 2001; Milková 2007; Nešetřil 1997]. In this note we present a variant

MSC2000: 05C85, 68R10.

Keywords: minimum spanning tree, graph algorithm, graph theory, Maple.

of Borůvka's algorithm and compare it to the algorithms given by Borůvka, Prim and Kruskal which have been central to the history of the problem.

In Section 2, we introduce the graph theory terminology used in this note. We outline the steps of our algorithm in Section 3, provide an illustrative example in Section 4, and prove the algorithm works as intended in Section 5. Section 6 highlights the differences between our algorithm and the work of Borůvka, Prim, and Kruskal. Finally, Section 7 discusses the Maple$^{\text{TM}}$ implementation of our algorithm.

2. Terminology

We use the following terminology throughout this note. An *undirected graph* G consists of a set of vertices, denoted by $V(G)$, and a set of unordered pairs of vertices called edges, denoted by $E(G)$. Since we focus on undirected graphs, henceforth we use the word graph to mean an undirected graph. If there exists a path from each vertex u to every other vertex v in G, we call G a *connected graph*. In a *weighted graph*, a real number (usually positive) is assigned to each edge and is called the *weight of the edge*. An example of a connected, weighted graph is provided in Figure 1. The sequence of edges $\{1,2\}$, $\{2,4\}$, $\{4,5\}$ and $\{5,1\}$ is called a *cycle*. There are many cycles in the graph in Figure 1. The *ends* of the edge $\{1,2\}$ are the vertices 1 and 2 and its weight is 1, that is, $w(\{1,2\}) = 1$.

A *minimum spanning tree* (MST) T in a connected, weighted graph G is a connected, acyclic subgraph of G with minimum total weight. To further clarify this definition, we use Figure 2 to explain the various graph theory terms embedded in an MST. The top left diagram shows a tree in G. A *tree* is a connected graph which is acyclic, that is, it has no cycles. A graph with multiple connected components such that each component is a tree is called a *forest*.

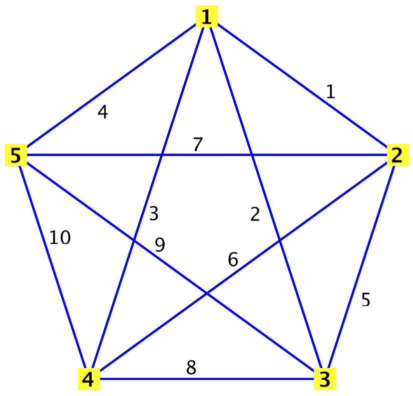

Figure 1. A connected, weighted graph G.

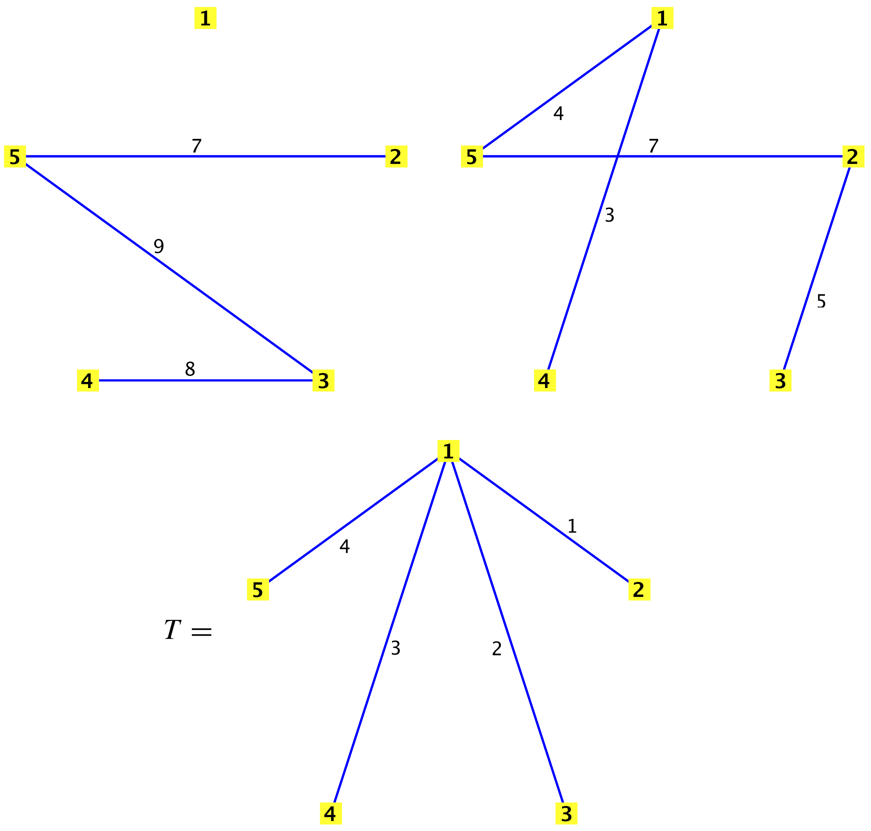

Figure 2. Clockwise from top left: a tree in G; a spanning tree in G; a minimum spanning tree T in G.

The top right diagram represents a spanning tree in G, that is, a tree with vertex set equal to $V(G)$. Finally, the bottom diagram shows a minimum spanning tree in G, which is a spanning tree with minimum total weight. The total weight of a tree is the sum of the weights of all the edges in the tree. The weight of the minimum spanning tree T in Figure 2 is $w(T) = 10$.

3. Steps of the algorithm

First, we establish the input and output for the algorithm, in addition to any necessary notation. The input is a weighted, connected graph G and the output is a spanning tree in G with minimum total weight, which we will call H. We start with H having no vertices and edges. We then construct H by adding vertices and edges as we go through the steps of the algorithm. In this procedure, we assume that there is an ordering of the vertices in $V(G)$. This is quite standard in a computer algebra system such as Maple. We designate the number of vertices in G as n.

3.1. ***Identify incident edge with smallest weight.*** For each vertex v_i for i from 1 to n, identify the edge incident to v_i with the smallest weight. In the case of multiple edges with the same weight, identify only one of these edges. If the ends of this edge are not already in H, then add the edge to H. Otherwise, do not make any changes to H. This ensures that H does not contain any cycles.

At the end of this step, H may contain one or more connected components. If there is only one connected component, then the procedure is finished. If the number of connected components of H is greater than one, then the procedure continues in the next step.

3.2. ***If necessary, create one connected component.*** If H consists of more than one connected component, then evaluate the weights of all edges connecting the distinct components of H. Add the edge with the smallest weight to H. In the case of multiple edges with the same weight, add only one of these edges.

Repeat this step until H has just one connected component.

4. An illustrative example

To demonstrate the two separate steps involved in this algorithm, we will build a minimum spanning tree H in the weighted graph shown in Figure 3. This is a complete bipartite graph, denoted by $K_{3,3}$.

The first step of the algorithm calls for an evaluation of the weights of the edges incident to each vertex. Starting with vertex 1, we evaluate the weights of the edges $\{1, 4\}$, $\{1, 5\}$, and $\{1, 6\}$. We note that $w(\{1, 6\}) = 6$ and this is the edge of smallest weight at vertex 1. Since neither vertex 1 nor vertex 6 is already part of H, the

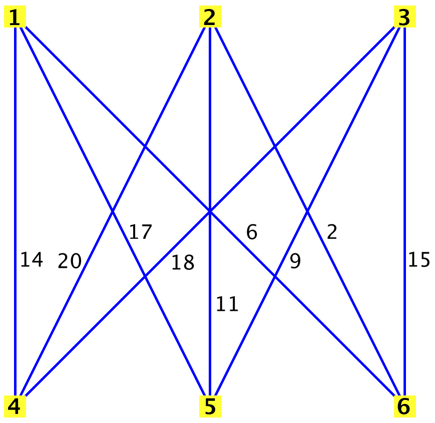

Figure 3. A weighted complete bipartite graph $K_{3,3}$.

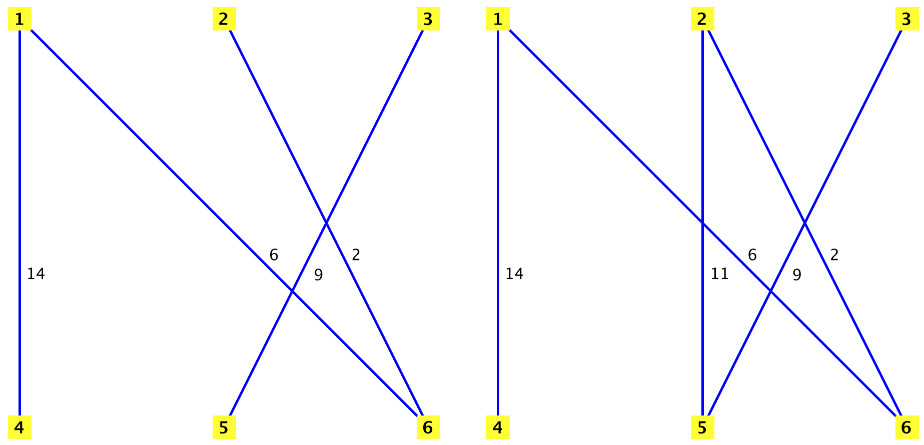

Figure 4. The generation of a minimum spanning tree H in a weighted $K_{3,3}$. We begin with the smallest edge incident to vertex 1, namely edge $\{1, 6\}$ in Figure 3. At the end of the first step of the algorithm, H is the graph shown on the left. At completion, H is the minimum spanning tree shown on the right.

edge $\{1, 6\}$ is the first addition to H. Even though it is obvious that the ends of the first edge added are not already part of the MST, this check is extremely important in later iterations to guarantee that H does not contain any cycles.

Once the algorithm runs through the entire first step, we have two distinct components to H. Figure 4, left, shows these components. The second step of the algorithm is necessary in this case to achieve a connected graph.

The second step begins with an evaluation of the weights of all edges that connect these two distinct components. The algorithm evaluates the weights of edges $\{3, 4\}$, $\{3, 6\}$, $\{2, 5\}$, and $\{1, 5\}$ and finds that edge $\{2, 5\}$ has the smallest weight. Thus this edge is added to H. We know that the addition of this edge will not create any cycles by the nature of distinct components. Now that H contains just one connected component with no cycles, we have our final desired result, as shown in Figure 4, right. The weight of the MST is $w(H) = 42$.

5. Proof of the algorithm

The proof of our algorithm uses similar techniques as the existing proofs for other MST algorithms, including the work of both Prim and Kruskal [Wilson and Watkins 1990; Rosen 2007]. In order to prove that the final output of the algorithm, H, is a minimum spanning tree, it is necessary to prove two separate properties: H is a spanning tree of the weighted, connected graph G; and H is of minimum weight.

5.1. ***H is a spanning tree of G.*** First, we show that H is a tree, that is, H is both connected and acyclic. The second step of the algorithm (3.2) guarantees that edges will be added to H until it has only one connected component. There are two different parts of the algorithm to consider when determining if H contains a cycle. In the first step (3.1), we do not add any edges whose ends are already in H. This prevents the creation of any cycles. In the second step (3.2), we only add edges connecting distinct components. There is no way to create a cycle in H by adding an edge that connects distinct components.

Second, we show that H spans G, that is, we show $V(H) = V(G)$. The first step of the algorithm, the loop for each vertex v_i for i from 1 to n, ensures that H contains all vertices of G.

5.2. ***H is of minimum weight.*** Suppose M is a minimum spanning tree in G. We know $w(M) \leq w(H)$ and M and H are both subgraphs of G. Our goal is to transform M into H in a way that shows $w(H) \leq w(M)$. This implies $w(H) = w(M)$ and proves that H is of minimum weight.

A tree on n vertices has $n-1$ edges. We name the edges in H according to the order in which they were added by the algorithm: $e_1, e_2, \ldots, e_{n-1}$. Assume e_1, $e_2, \ldots, e_{k-1}$ are all in M as well. So $e_k = \{u, v\}$ is the first edge in H that we find is not also in M. We want to add e_k to M and delete an edge from it such that the resulting subgraph L is a spanning tree of G and $w(L) \leq w(M)$. We know that there must be a path between u and v in M since the MST is both connected and spanning. Thus the addition of $e_k = \{u, v\}$ to M creates a cycle C in M. Let e be the other edge incident to u in C.

To select the edge to delete so that we obtain L, we now consider two cases: either e_k was added to H during the first step (3.1) or e_k was added during the second step (3.2).

If e_k was added to H during the first step, it must be true that e_k is the edge of smallest weight at one of its endpoints. Without loss of generality, assume that e_k is the edge of smallest weight incident to u. We then delete the edge e from M to obtain L. We know that $w(e) \geq w(e_k)$ because e_k is an edge incident to u with the smallest weight and hence $w(L) \leq w(M)$.

If e_k was added to H during the second step, let K be the subgraph of H to which e_k was added. Then we know that u and v must have been in distinct components of K. Note that K is also a subgraph of M because all the edges added to H before e_k are in M as well. If we start traversing the cycle C along the edge e, then we must reach an edge f such that its ends are in distinct components of K and we delete f from M to obtain L. Again, $w(f) \geq w(e_k)$ because the algorithm adds an edge of smallest weight that connects distinct components of K, and hence $w(L) \leq w(M)$.

Since M is of minimum weight, $w(L) = w(M)$ and thus L is a minimum spanning tree of G. We repeat the process of adding and deleting an edge with M replaced by L. We continue in this way until we get H.

6. Comparison with other algorithms

We now compare our algorithm to the three algorithms by Prim, Kruskal and Borůvka "that have played a central role in the history of the MST problem" as stated in [Milková 2007].

Prim's algorithm [1957] generates a minimum spanning tree by identifying an edge with minimum weight incident to the initial vertex and spreading the tree from this edge. At every iterative step, the algorithm finds and adds the edge of smallest weight such that one vertex is already part of the MST and the other vertex is not. The procedure is complete once the vertex set of the tree is equal to the vertex set of the original graph, that is, the tree spans all of the vertices. Prim's algorithm allows for just one tree at any given step. By comparison, there may be multiple trees, or a forest, that are ultimately connected in our algorithm.

We use graphs created in Maple to highlight the differences between our algorithm and Prim's algorithm. Figure 5 shows MSTs in a complete graph on six vertices, denoted by K_6, with each edge weighted 5. The left diagram shows the MST generated by the `spantree` procedure in Maple, which uses Prim's algorithm. The Maple implementation of our algorithm created the MST shown in the diagram on the right.

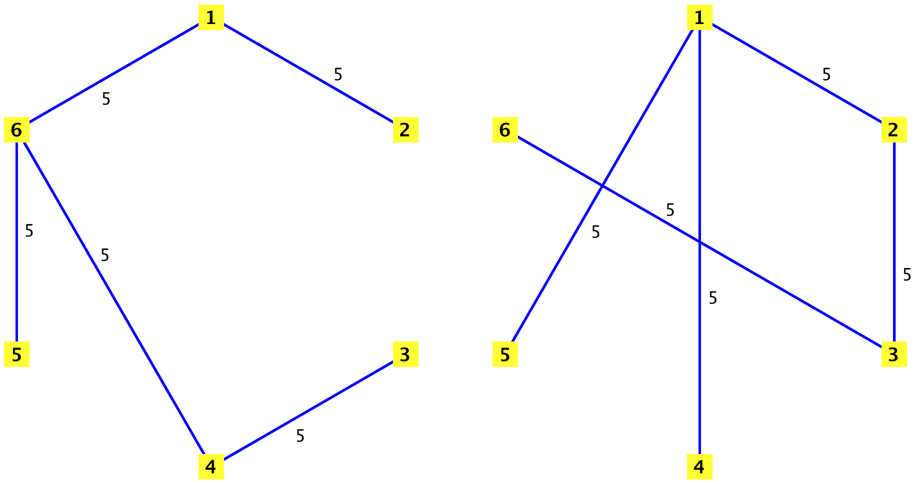

Figure 5. MSTs in K_6 with each edge weighted 5. The one on the left was created by Prim's algorithm, the one on the right by our algorithm.

Kruskal's algorithm [Kruskal 1956] is a "greedy" algorithm that constructs a minimum spanning tree by adding edges with minimum weight as long as doing so does not form a cycle. For detailed steps of the algorithm, see [Wilson and Watkins 1990]. Thus, Kruskal's algorithm chooses edges of smallest weight from the entire graph at every iteration whereas our algorithm identifies an edge with minimum weight at each vertex and later between two connected components.

Borůvka's algorithm [Borůvka 1926a; 1926b; Nešetřil et al. 2001] is a recursive algorithm which at every recursive step repeats the first step of the algorithm on a new graph formed by contraction. The first step of Borůvka's algorithm and the first step of our algorithm are identical. Thus, at the end of the first step, there is a set of chosen edges that may not form a single connected component. Borůvka's algorithm then forms a new graph by contraction as follows. Each connected component is replaced by a single vertex. All edges connecting vertices in the same connected component are eliminated. All edges between two distinct components are eliminated except for the edge with the smallest weight. If edges do not have distinct weights, a tie-breaking procedure is used to retain only one edge between two distinct components. Borůvka's algorithm then repeats the first step on this newly formed graph. The recursion continues until only one vertex remains in the contracted graph; that is, until the chosen edges form only one connected component. The set of all the edges chosen each time the first step is executed constitutes an MST. The contraction to form the new graph and the subsequent recursion of Borůvka's algorithm are replaced in our algorithm with the iterative process of joining the connected components obtained at the end of the first step with edges of minimum weight.

Thus of the three algorithms mentioned here, our algorithm is most similar to Borůvka's algorithm. The output of our algorithm may differ from the output of Borůvka's algorithm because both depend on the particular tie-breaking procedures used when all edges do not have distinct weights.

7. Maple implementation of algorithm

Our implementation of the algorithm takes advantage of the `networks` package in Maple. This package contains many commands useful in graph theory, a number of which we will discuss later in this section. The Maple implementation of the first step of the algorithm (3.1) is shown on the next page

The implementation employs a number of useful commands. In general, the `networks` package makes it easy to work with both vertices and edges in a graph. The `incident` command returns a set of the edges incident to a vertex v_i. The `eweight` command gives the weights of the edges in a graph. Both of these commands are important when we create the list of the weights of all edges incident to vertex v_i. The nops command counts the number of elements in its argument, and the member command tests if an element belongs to a set or list. While nops and member are not specific to the networks package, both of these commands play a large role in the procedure as well.

```
pathset:={}; # This is where we will keep track of edges in our MST.
for i from 1 to nops(vertices(G)) do
  listofweights:=[];
  currentedge:={};
  for j in ends(incident(i,G),G) do
    listofweights:=[op(listofweights),eweight(edges({j[1],j[2]},G)[1],G)];
  end do;
  # In this loop, we create a list of the weights (called "listofweights")
  # of all edges incident to vertex v_i.
  smallest:=listofweights[1];
     for k from 2 to nops(listofweights) do
         if (listofweights[k]<smallest) then smallest:=listofweights[k]
         end if;
     end do;
     # Here, we identify the smallest value in "listofweights".
  for x in ends(incident(i,G),G) do
    if eweight(edges({x[1],x[2]},G)[1],G)=smallest then
       currentedge:=currentedge union {x};
    end if;
  end do;
  # We match the smallest value to the edge(s) with this weight and add it
  # to a set called "currentedge".
  found:=0;
  foundvertex:=0;
  for j in pathset do
      if (member(currentedge[1][1],j) and currentedge[1][1]<>foundvertex)
       then found:=found+1:foundvertex:=currentedge[1][1]
      end if;
      if found=2 then break end if;
      if (member(currentedge[1][2],j) and currentedge[1][2]<>foundvertex)
       then found:=found+1:foundvertex:=currentedge[1][2]
      end if;
      if found=2 then break end if;
  end do;
  if found=2 then pathset:=pathset
      else pathset:=pathset union {currentedge[1]}
  end if;
  # If the ends of the edge incident to v_i with the smallest weight
  # are already part of pathset, then pathset remains the same.
  # Otherwise, we add just one edge to pathset.
end do;
```

Maple implementation of the first step (3.1) of our algorithm (# introduces a comment line). The input is a weighted, connected graph G.

Following the first step of the procedure, we begin to build H by inserting vertices 1 to n and the edges from `pathset` into H. If H has more than one component, the second part of the procedure starts by identifying all edges in G that connect distinct components in H, as shown below. These edges are then added to a set called `connectingedges`. The `components` command, which identifies the components of a graph as a set of sets, is especially valuable in this part of the procedure.

```
possibleedges:=ends(G) minus pathset;
connectingedges:={};
for r in possibleedges do
  d:=nops(components(H));
  addedge(r,H);
    if nops(components(H))<d
      then connectingedges:=connectingedges union {r}
    end if;
  delete(edges({r[1],r[2]},H),H);
end do;
```

The implementation of the rest of the second step (3.2) is similar to the procedure for the first step. After establishing the set `connectingedges`, we evaluate the weights of the edges, identify the smallest value, and add the associated edge to H. This step is repeated until H contains just one connected component.

Now we take a look at the complexity of our implementation. Let n be the number of vertices in G and m the number of edges in G. If we assume that each of the Maple command runs in unit time, then our implementation runs in time $O(mn)$. This could be improved with a more efficient sorting procedure in each step. For a discussion of the complexity of MST algorithms and recent work on MSTs see [Wu and Chao 2004; Graham and Hell 1985; Cheriton and Tarjan 1976].

8. Conclusion

Due to the numerous applications of minimum spanning trees to communications and transportation networks, it is important to have efficient algorithms to find minimum spanning trees in weighted, connected graphs. Borůvka, Jarník, Prim, and Kruskal, among others, have made important contributions to this area of graph theory. We have presented an algorithm that is a variant of the original solution by Borůvka and unlike the proof by Borůvka, we have provided a proof of the algorithm using the language of modern graph theory. The running time of the implementation could be improved and we hope the reader will try to do so.

Acknowledgments

The Graph Algorithms course is an elective course offered by the Department of Mathematics at Bates College and it aims to teach students the basics of graph

theory (with an emphasis on algorithms) and computer programming skills. Two thirds of class time is devoted to learning graph theory and one third is spent learning basic programming skills through the use of the `networks` package in the computer algebra system Maple. In the class of Fall 2007, after learning the definitions and basic properties of trees and MSTs, we discussed briefly the algorithms by Prim and Kruskal in class. The assignment for the next class was to write the pseudocode for an algorithm (not necessarily one of the algorithms we had discussed in class) to find an MST in a weighted, connected graph. The next class began with one of the students (the second author) presenting the first step of an algorithm to the whole class. As a class, we then completed the algorithm to produce the desired result. The second author and another student implemented the algorithm in Maple as their final exam project, with help from the instructor (the first author).

The second author was supported by a Hoffman Research Support Grant, a Bates College program funded by an endowment established by the Hoffman Foundation, during our work on this note in Summer 2008 when we wrote the proof of the algorithm. We thank Emmanuel Drabo, Philip Greengard, Alex Jorge, Binit Malla, Dylan Mogk, Razin Mustafiz, Duane Pelz and Dan Perry — the Graph Algorithms class of Fall 2007 for the class discussion to complete the second step of the algorithm. We are particularly grateful to Dan Perry for teaming up with the second author to implement the algorithm in Maple. We would also like to thank Eric Towne of the Department of Mathematics at Bates College for conducting the labs through which the students learned to program in Maple and for helpful comments on this note.

References

[Borůvka 1926a] O. Borůvka, "O jistém problému minimálním", *Práce Mor. Přírodověd., Spol. v Brně* **3** (1926), 37–58.

[Borůvka 1926b] O. Borůvka, "Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí", *Elektrotechnický obzor* **15** (1926), 153–154.

[Cheriton and Tarjan 1976] D. Cheriton and R. E. Tarjan, "Finding minimum spanning trees", *SIAM J. Comput.* **5**:4 (1976), 724–742. MR 56 #4783 Zbl 0358.90069

[Graham and Hell 1985] R. L. Graham and P. Hell, "On the history of the minimum spanning tree problem", *Ann. Hist. Comput.* **7**:1 (1985), 43–57. MR 86g:68005 Zbl 0998.68003

[Kruskal 1956] J. B. Kruskal, Jr., "On the shortest spanning subtree of a graph and the traveling salesman problem", *Proc. Amer. Math. Soc.* **7** (1956), 48–50. MR 17,1231d Zbl 0070.18404

[Milková 2007] E. Milková, "The minimum spanning tree problem: Jarník's solution in historical and present context", pp. 309–316 in *6th Czech-Slovak International Symposium on Combinatorics, Graph Theory, Algorithms and Applications*, edited by P. Hliněný et al., Electron. Notes Discrete Math. **28**, Elsevier, Amsterdam, 2007. MR 2324010

[Nešetřil 1997] J. Nešetřil, "A few remarks on the history of MST-problem", *Arch. Math.* (*Brno*) **33**:1-2 (1997), 15–22. MR 98g:01065 Zbl 0909.05022

[Nešetřil et al. 2001] J. Nešetřil, E. Milková, and H. Nešetřilová, "Otakar Borůvka on minimum spanning tree problem: translation of both the 1926 papers, comments, history", *Discrete Math.* **233**:1-3 (2001), 3–36. MR 2002f:05053

[Prim 1957] R. C. Prim, "The shortest connecting network and some generalizations", *Bell System Tech. J.* **36** (1957), 1389–1401.

[Rosen 2007] K. H. Rosen, *Discrete mathematics and its applications*, McGraw-Hill, New York, 2007. Zbl 0691.05001

[Wilson and Watkins 1990] R. J. Wilson and J. J. Watkins, *Graphs: An introductory approach*, Wiley, New York, 1990. MR 91b:05001 Zbl 0712.05001

[Wu and Chao 2004] B. Y. Wu and K.-M. Chao, *Spanning trees and optimization problems*, Discrete Mathematics and its Applications, Chapman & Hall/CRC, Boca Raton, FL, 2004. MR 2004i:90008 Zbl 1072.90047

Received: 2009-03-01 Accepted: 2009-05-02

pjayawan@bates.edu *Department of Mathematics, Bates College, Lewiston, ME 04240, United States*

kerry.glavin@gmail.com *Department of Mathematics, Bates College, Lewiston, ME 04240, United States*