Sergio E. Garcia Tapia
Algorithms by Sedgewick and Wayne (4th edition) [SW11]
January 16, 2025

4.3: Minimum Spanning Trees

Exercise 1. Prove that you can rescale the weights by adding a positive constant to all of them or by multiplying them by a positive constant without affecting the MST.

Solution.

Proof. Suppose G is a connected graph. Then G admits an MST. Let T be any MST of G.

Let G_c be the graph obtained by adding the positive constant c to the weight of every edge of G. Since the endpoints of the edges remain unchanged, it follows that G_c is still connected, and thus it admits a spanning tree. In particular, if T_c is the tree whose edges are the same as T but with weights increased by c, then T_c is still a spanning tree. To see that it is an MST for G_c, let T_c' be any other tree spanning tree of G_c. Let $e_{1,c}, \ldots, e_{V-1,c}$ be the edges of T_c and $f_{1,c}, \ldots, f_{V-1,c}$ be the edges of T_c'. By definition of G_c, there are edges e_k and f_k of G satisfying

$$\begin{aligned} w(e'_{k,c}) &= w(e_k) + c, \\ w(f'_{k,c}) &= w(f_k) + c, \end{aligned}$$

where $1 \leq k \leq V-1$. In particular, the our definition of T_c says that $e_1, \ldots, e_{V-1}$ are the edges of T, an MST of G. Similarly, since T_c' is a spanning tree of G_c, the edges $f_1, \ldots, f_{V-1}$ are the same edges but with lower weight. The upshot is that they form a spanning tree T' of G. By definition of an MST, we know that

$$\begin{aligned} w(T) &\leq w(T') \\ \sum_{k=1}^{V-1} w(e_k) &\leq \sum_{k=1}^{V-1} w(f_k) \\ (V-1)\cdot c + \sum_{k=1}^{V-1} w(e_k) &\leq (V-1)\cdot c \sum_{k=1}^{V-1} w(f_k) \\ \sum_{k=1}^{V-1} (w(e_k) + c) &\leq \sum_{k=1}^{V-1} (w(f_k) + c) \\ \sum_{k=1}^{V-1} w(e_k') &\leq \sum_{k=1}^{V-1} w(f_k') \\ w(T_c) &\leq w(T_c') \end{aligned}$$

We conclude that if T is a minimum spanning tree of G, the T_c is a minimum spanning tree of G_c, so the MST is unaffected when all edge weights are increased by a positive constant. A similar argument works when multiplying by a positive constant. □

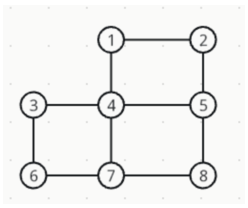

Figure 1: Undirected edge-weighted graph with all equal weights for Exercise 1.

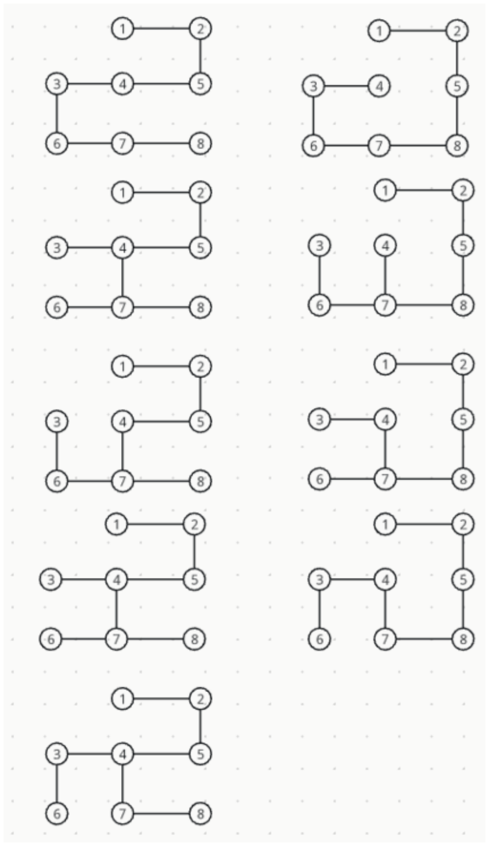

Figure 2: Some MSTs for the weighted graph in Figure 1.

Exercise 2. Draw all of the MSTs of the graph depicted in Figure 1 (all edge weights are equal).

Solution. Since the graph has $V = 8$ vertices, a spanning tree would have $V - 1 = 7$ edges. The graph is connected with $E = 10$ edges, so we need to pick 3 edges to remove. Thus, an upper bound for the number of minimum spanning trees is $\binom{10}{3}$. This upper-bound is conservative because we cannot just remove *any* edges, given that we also must ensure the graph remains connected (else it would not be a spanning tree). Figure 2 shows some, but not all of them.

Exercise 3. Show that if a graph's edges all have distinct weights, the MST is unique.

Solution.

Proof. Suppose T_1 and T_2 are MSTs of a graph G whose edge weights are all distinct. Let e be any edge in T_1, and suppose that e is not in T_2. If we add e to T_2, then a cycle

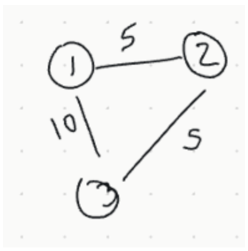

Figure 3: An undirected weighted tree whose edge weights are not all distinct.

containing e is formed. Let f be any other edge in that cycle, and add that edge to T_1, thus forming a cycle in T_1. By assumption, we know that e and f have distinct weights, so without loss of generality, suppose $w(e) < w(f)$.

If we remove f from T_2, then we eliminate the cycle that was formed by adding e to T_2, and the tree remains connected. However, now the weight of the resulting tree is $w(T_2) - w(e) + w(f)$, which is strictly smaller than the weight of $w(T_2)$ because $-w(e) + w(f) < 0$. But T_1 and T_2 are MSTs, so this contradicts their minimality.

We conclude that no such each e exists. That is, we do in fact have $e \in T_2$. Since e is an arbitrary edge, this hold for every edge of T_1. A symmetric arguments holds that shows that every edge of T_2 belongs to T_1, and hence, T_1 and T_2 are in fact the same tree. □

Exercise 4. Consider the assertion that an edge-weighted graph has a unique MST *only* if its edge weights are distinct. Give a proof or a counterexample.

Solution. The assertion is false. See Figure 3. There are three spanning trees:

- The tree with edges `1-2` and `1-3`, with a total weight of 15.
- The tree with edges `1-3` and `2-3`, with a total weight of 15.
- The tree with edges`1-2` and `2-3`, with a total weight of 10.

Though edges `1-2` Among these, only the last one with weight 10 is an MST, and it is the only MST. Note however that not all weights are distinct.

Exercise 5. Show that the greedy algorithm is valid even when the edge weights are not distinct.

Solution. The statement of the Greedy MST algorithm is:

Proposition. *The following method colors black all edges in the MST of any connected edge-weighted graph with V vertices: starting with all edges colored gray, find a cut with no black crossing edges, color its minimum-weight crossing edge black, and continue until $V - 1$ edges have been colored black.*

Proof. Suppose G is a connected, undirected weighted graph. Start with any cut of the graph, and let e_1 be *a* crossing edge of minimal weight. Note that we are not saying it is *the* crossing edge of minimal weight; there may be others with the same weight, but among them we pick e_1. By the cut property (**Proposition J**), it belongs to an MST of the graph. Suppose that T is an MST of the graph not containing e_1. By adding e_1 to

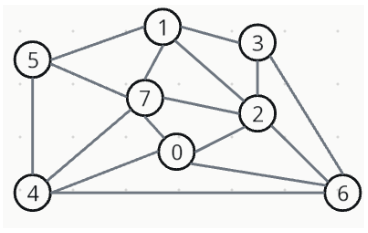

Figure 4: Undirected weighted graph for `tinyEWG.txt` (see Exercise 6, or page 604).

T, we create a cycle that contains at least one other edge f which crossing the same cut as e_1. By the minimality of e_1, we must have $w(e_1) \leq w(f)$, and since T is the MST, the possibility $w(e_1) < w(f)$ is impossible (it would imply a spanning tree of smaller which, contradicting the minimality of T). We conclude $w(e_1) = w(f)$. Thus, replacing f by e_1, we still have an MST, one which e_1 belongs to. We color e_1 black.

Now suppose that we have colored k edges black, where $1 \leq k < V-1$, and by the cut property they all belong to *an* (not necessarily *the*) MST of the graph. Since $k < V-1$, the tree consisting of the k black-colored edges is not spanning, meaning at least one vertex v is not an endpoint of any of the black-colored edges. In particular, there must be a cut with no black crossing edges. For example, one such cut consists of precisely the endpoints of the k black-colored edges. Since G is connected, there is at least one crossing edge. Let e_{k+1} be *any* crossing edge of minimal weight. Let T_k be an MST containing all k black-colored edges. If e_{k+1} does not belong to T_k, then add e_k to T_k to form a cycle. Let f_{k+1} be the other edge in the cycle, which must also cross the same cut as e_{k+1}. As earlier, the minimality of e_{k+1} ensures that $w(e_{k+1}) \leq w(f_{k+1})$. The minimality of T_k assures us that the possibility $w(f_{k+1}) \leq w(e_{k+1})$. We conclude that $w(e_{k+1}) = w(f_{k+1})$, so we can replace f_{k+1} by e_{k+1} and the resulting tree is still a minimal spanning tree. We color e_k black.

Now if $V-2$ edges have been colored black, then applying this argument again results in a spanning tree consisting of only the $V-1$ black-colored edges, which is the MST. □

Exercise 6. Give the MST of the weighted graph obtained by deleting vertex 7 from `tinyEWG.txt` (see page 604, and Figure 4).

```
8
16
4  5  0.35
4  7  0.37
5  7  0.28
0  7  0.16
1  5  0.32
0  4  0.38
2  3  0.17
1  7  0.19
0  2  0.26
1  2  0.36
1  3  0.29
2  7  0.34
6  2  0.40
3  6  0.52
```

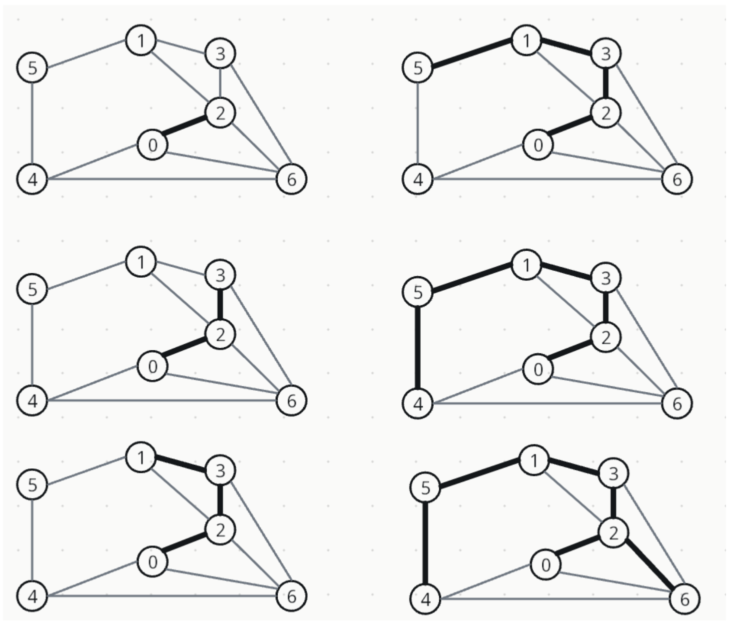

Figure 5: Applying the Greedy MST algorithm to the tree in Figure 4 after removing vertex 7.

```
6  0  0.58
6  4  0.93
```

Solution. If we delete vertex 7 and its associated edge, then the remaining edges are:

```
4  5  0.35
1  5  0.32
0  4  0.38
2  3  0.17
0  2  0.26
1  2  0.36
1  3  0.29
6  2  0.40
3  6  0.52
6  0  0.58
6  4  0.93
```

Figure 5 shows my application of the greedy MST algorithm to find the resulting MST when removing vertex 7.

Exercise 7. How would you find a *maximum* spanning tree of an edge-weighted graph?

Solution. I would apply a version of the greedy MST algorithm that colors black the crossing edge of maximal weight (instead of the crossing edge of minimal weight).

Exercise 8. Prove the following, known as the *cycle property*: Given any cycle in an edge-weighted graph (all edge weights distinct), the edge of maximum weight in the cycle does not belong to the MST of the graph.

Solution.

Proof. Let G be a connected undirected edge-weighted graph whose edge weights are all distinct. If G has a cycle $\mathtt{v}_0\mathtt{v}_1\cdots\mathtt{v}_k\cdots\mathtt{v}_0$, let $\mathtt{e}_i = \mathtt{v}_{i-1}\mathtt{v}_i$, where $1 \le i \le k$, and $\mathtt{e}_{k+1} = \mathtt{v}_k\mathtt{v}_0$. Without loss of generality (perhaps after renaming), say that $\mathtt{e}_1$ is the edge of maximum weight in the cycle. Given a cut with no black crossing edges for which $\mathtt{e}_1$ is a crossing edge, I claim that some other edge in the cycle must also cross the same cut.

Let A and B be nonempty subsets of the vertices that comprise the cut, with $\mathtt{v}_0 \in A$ and $\mathtt{v}_1 \in B$. If edges $\mathtt{e}_i$ for $2 \le i \le k+1$ do not cross the cut defined by A and B, then both endpoints belong to the same side of the cut; that is, they both belong to A or they both belong to B. Since $\mathtt{v}_1 \in B$, this implies $\mathtt{v}_2 \in B$, which implies that $\mathtt{v}_i \in B$ for $2 \le i \le k$, But since $\mathtt{e}_{k+1}$ is not a crossing edge either, and since $\mathtt{v}_k \in B$, this implies that $\mathtt{v}_0 \in B$. This is impossible because $\mathtt{v}_0 \in A$, and $A \cap B = \emptyset$ since A and B form a partition.

The contradiction implies that there is some edge $\mathtt{e}_j$ in the cycle distinct from $\mathtt{e}_1$ that crosses the same cut as $\mathtt{e}_1$. By the cut property, the crossing edge of minimum weight belongs to the MST. That crossing edge cannot be $\mathtt{e}_1$, because $\mathtt{e}_j$ is a crossing edge of smaller weight. Since the cut under consideration for which $\mathtt{e}_1$ is a crossing edge is arbitrary, we conclude that there is no such for which $\mathtt{e}_1$ would be the minimum crossing edge, and hence, it does not belong to the MST. □

Exercise 9. Implement the constructor for `EdgeWeightedGraph` that reads an edge-weighted graph from the input stream, by suitably modifying the constructor from `Graph` (see page 526).

Solution. See `com.segarciat.algs4.ch4.sec3.ex09`.

Exercise 10. Develop an `EdgeWeightedGraph` implementation for dense graphs that uses an adjacency matrix (two-dimensional array of weights) representation. Disallow parallel edges.

Solution. See `com.segarciat.algs4.ch4.sec3.ex10`.

Exercise 11. Determine the amount of memory used by `EdgeWeightedGraph` to represent a graph with V vertices and E edges, using the memory-cost model of **Section 1.4**.

Solution. First, `EdgeWeightedGraph` requires 16 bytes of object overhead, 4 bytes for its `int E` field, 4 bytes for its `int V` field, and 8 bytes for its reference to `Bag<Edge>[] adj`. Thus we begin with a flat cost of 32 bytes. Next, the array `adj` itself requires 24 bytes as flat cost, so now the flat cost is 56 bytes. The array `adj` contains V references to `Bag<Edge>` objects, each reference is 8 bytes, so that is $8V$ bytes. Now, each `Bag<Edge>` object requires 16 bytes of object overhead, 4 bytes for `int size`, 8 bytes for the reference to the `Node first`, and 4 bytes of padding, which is 32 bytes. Since there are V of them, this means an additional $32V$ bytes, so now we have $32 + 40V$ total so far. Now for each `Edge`, we require a `Node`, which has 16 bytes of overhead, 8 bytes for the reference to its enclosing class, 8 bytes for the reference to `next`, and 8 bytes for the reference to the item field, for a total of 40 bytes. Now, the item referred to by each node is an `Edge`. Moreover, since each edge `v-w` belongs to both the adjacency list of `v` and the adjacency

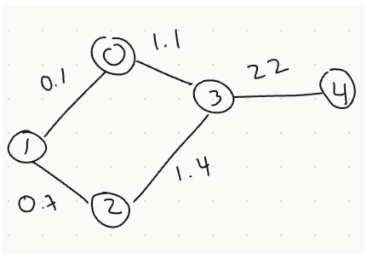

Figure 6: A weighted graph whose heaviest edge must belong to the MST.

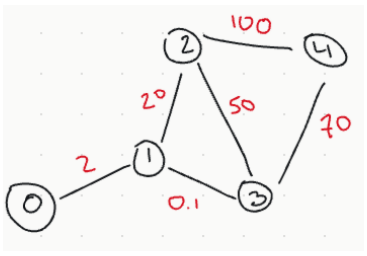

Figure 7: A weighted graph with a min-weight edge in a cycle.

list of `w`, the 40 byte cost is doubled to 80. However, only one actual edge object exists (both nodes refer to the same one). An `Edge` requires 16 bytes of object overhead, 4 bytes for its `int v` field, 4 bytes for its `int w` field, and 8 bytes for its `double weight` field, for a total of 32 bytes. Hence, each edge requires a total of $40 + 40 + 32 = 112$ bytes.

Now the overall cost is $56 + 40V + 112E$ bytes.

Exercise 12. Suppose that a graph has distinct edge weights. Does its lightest edge have to belong to the MST? Can its heaviest edge belong to the MST? Does a min-weight edge on every cycle have to belong to the MST? Prove your answer to each question or give a counterexample.

Solution. The lightest edge must belong to the MST. Here's my proof:

Proof. Let `v-w` be the lightest edge of the tree. Define a cut by the singleton set containing `v` (and its complement). Then `v-w` is a crossing edge for this cut. Since it is the lightest edge of the graph, it follows that it is minimum crossing edge for this cut, and hence, it must belong to the MST, by the cut property (**Proposition J**). □

Yes, the heaviest edge may belong to the MST. Consider the graph on Figure 6. We see that `3-4` is the heaviest edge, but it is also the only edge to vertex `4`. Therefore, any spanning tree must contain `4`, including the MST.

Figure 7 shows an example of a graph G with a weighted edge `2-3` that has minimum weight in the cycle `2-3-4`. However, `2-3` does not belong to the MST. To see this, note that since all the weights of G are distinct, the MST is unique. If we apply Prim's algorithm starting at vertex `0`, we would add `0-1`, followed by `1-2`, followed by `1-2`, followed by `3-4`, completing the MST.

Exercise 13. Give a counterexample that shows why the following strategy does not necessarily find the MST: "Start with any vertex as a single-vertex MST, the add `V-1` edges to it, always taking next a min-weight edge incident to the vertex most recently added to the MST."

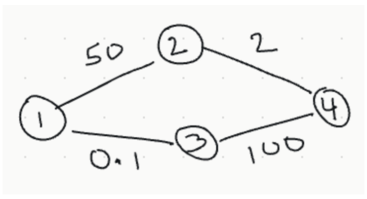

Figure 8: Counterexample for Exercise 13.

Solution. The problem is we cannot guarantee that the min-weight edge incident on the most recently added vertex is necessarily the best way to connect the vertex on the other endpoint of the edge to the tree, If we apply the algorithm to the graph depicted in Figure 8, starting at vertex `1`, we would pick edge `1-3` first, and since `3` is the most recently added vertex, we need to pick the minimum weight edge incident to `3`. The only edge incident to `3` that is not in the tree is `3-4`, and then finally we would add edge `4-2`. However, this is not the MST. In particular, the spanning tree consisting of edges `1-3`, `1-2`, and `2-4` is the MST.

Exercise 14. Given an MST for an edge-weighted graph G, suppose that an edge in G that does not disconnect G is deleted. Describe how to find an MST for the new graph in time proportional to E.

Solution. Let T be the MST of G in question, and let `e` be the edge removed from G. First, we check whether `e` belongs to T. If not, then T is still a spanning tree of the new graph. Moreover, if T' were a spanning tree of smaller weight for the new graph, then it would be a spanning tree of G also, which would contradict the minimality of T.

Now suppose `e` = `v-w` does belong to T. Then removing edge `e` from G disconnects T, resulting in two distinct connected components for T. Let $T_{\mathtt{e}}$ be the graph that results when removing `e` from T. We can create a new graph consisting of the edges in $T_{\mathtt{e}}$, and we can use depth-first search to identify the components of $T_{\mathtt{e}}$. This takes time proportional to $V - 2 + V$ because $T_{\mathtt{e}}$ has V vertices and $V - 2$ edges. Note that `v` and `w` belong to different components. Now we can iterate through all edges in G, at each point inspecting the endpoints, keeping track only of the ones who have one endpoint in the same component as `v` and another in the same component as `v-w`. The lightest such edge is the one we must add, completing the MST.

Exercise 15. Given an MST for an edge-weighted graph G and a new edge e with weight w, describe how to find an MST of the new graph in time proportional to V.

Solution. We add e to the MST, which creates a unique cycle containing e. We use DFS to find the edges in the cycle in time proportional to $2V$ (because the MST with e graph has V edges and V vertices). We remove the heaviest edge in the cycle, which may be e or a different edge.

Exercise 16. Given an MST for an edge-weighted graph G and a new edge e, write a program that determines the range of weights for which e is in an MST.

Solution. This is an implementation of the previous exercise. After adding `e` to the given MST, we find the cycle containing `e`. The `e` must have a weight no larger than the largest wait of the remaining edges in the cycle.

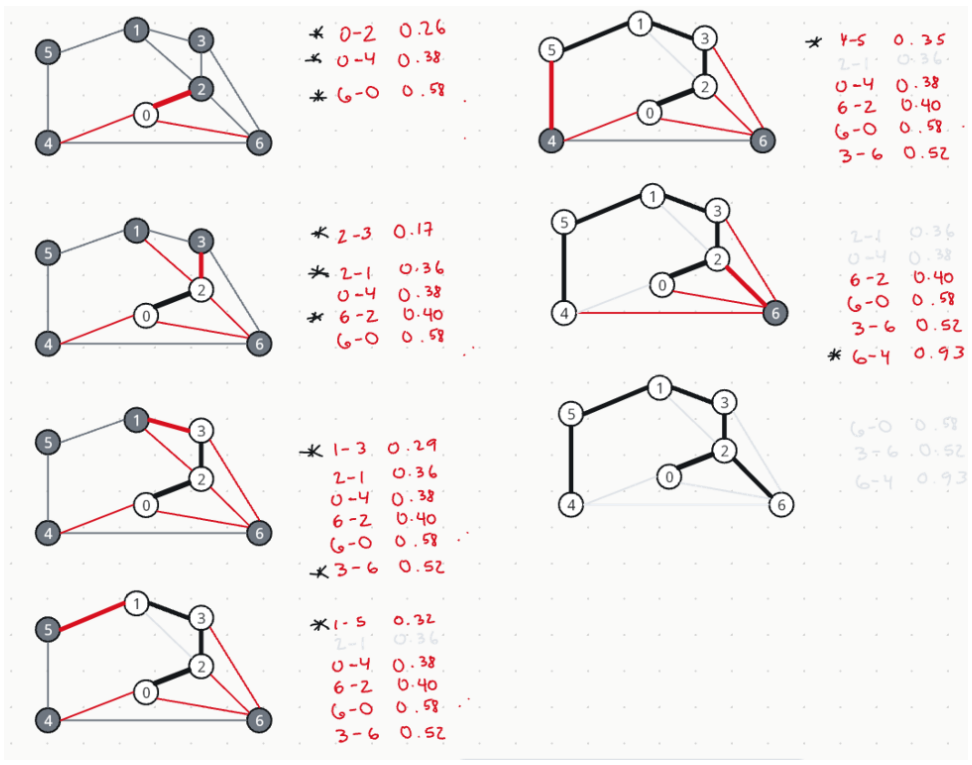

Figure 9: Trace of `LazyPrimMST` for **Exercise 4.3.18**.

See `com.segarciat.algs4.ch4.sec3.ex16`.

Exercise 17. Implement `toString()` for `EdgeWeightedGraph`.

Solution. I did this as part of Exercise 10.

Exercise 18. Give traces that show the process of computing the MST of the graph defined in **Exercise 4.3.6** with the lazy version of Prim's algorithm, the eager version of Prim's algorithm, and Kruskal's algorithm.

Solution. See Figure 9 for the trace of `LazyPrimMST`, Figure 10 for the trace of (eager) `PrimMST`, and Figure 11 for the trace of `KruskalMST`.

Exercise 19. Suppose that you implement `PrimMST` but instead of using a priority queue to find the next vertex to add to the tree, you can through all V entries in the `distTo[]` array to find the non-tree vertex with the smallest weight. What would be the order of growth for the running time for a graph with V vertices and E edges? When would this method be appropriate, if ever? Defend your answer.

Solution. Searching the array for the next vertex to visit entails a search for the minimum, which takes time proportional to V. Since this occurs until all vertices are visited, we need to do this V times. When a vertex is visited, all edges are examined. Therefore, the order of growth of the running time is $V^2 + E$. This might be acceptable for a dense graph, where $E \approx V^2$.

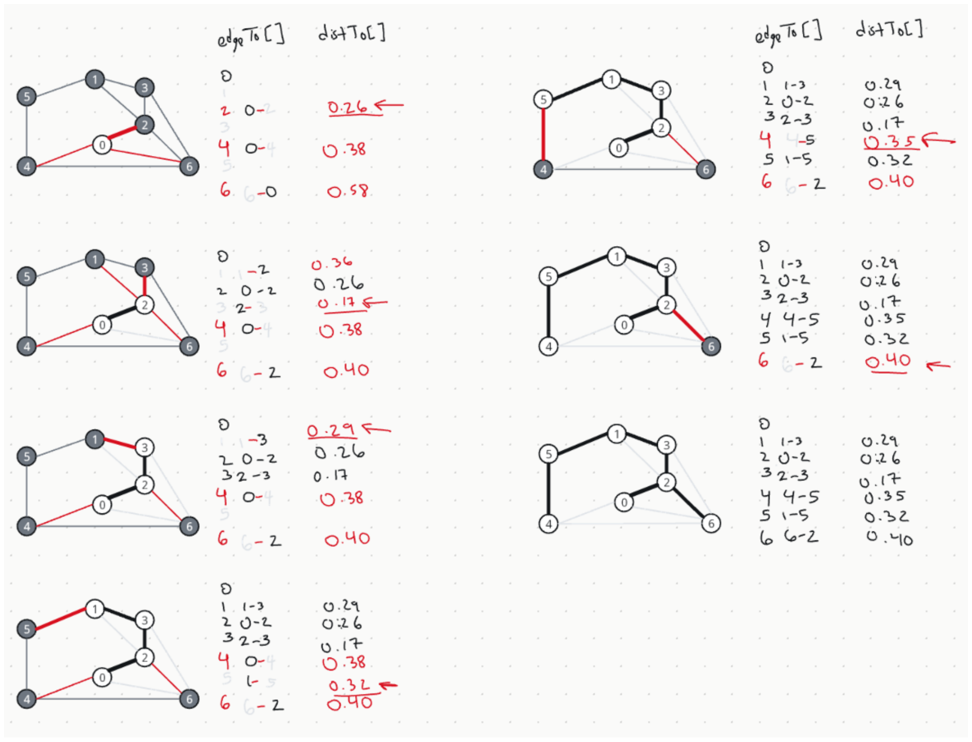

Figure 10: Trace of (eager) `PrimMST` for **Exercise 4.3.18**.

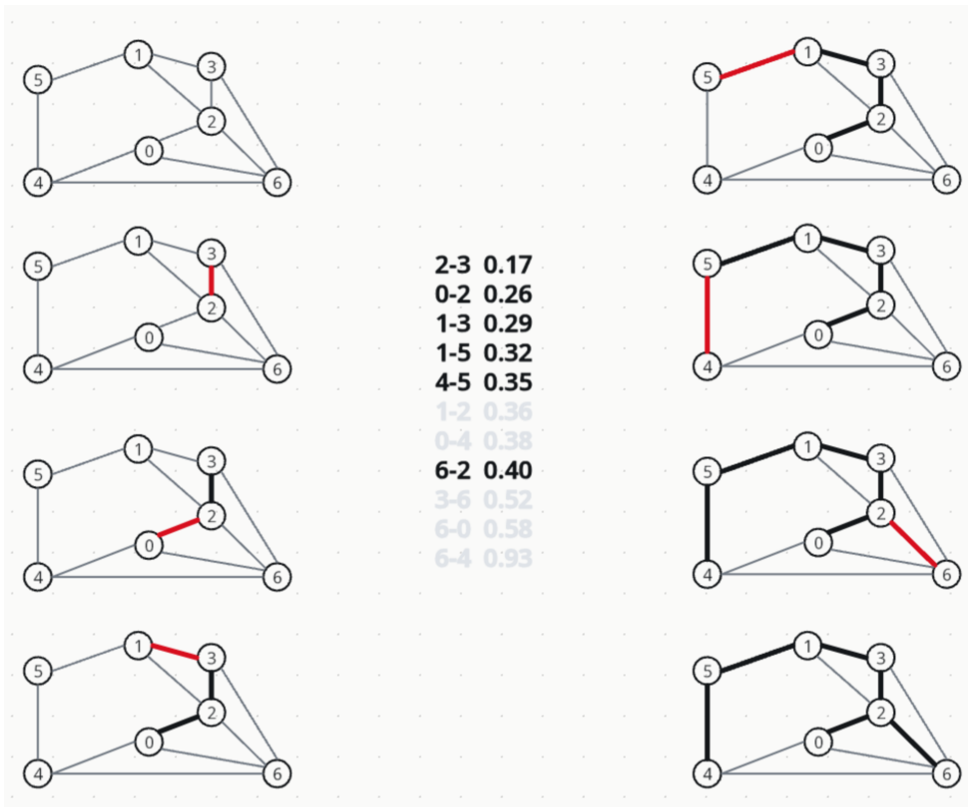

Figure 11: Trace of (eager) `KruskalMST` for **Exercise 4.3.18**.

References

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. 4th ed. Addison-Wesley, 2011. ISBN: 9780321573513.