Sample Solution

Assignment 4

CSCI 3110 — Summer 2018

Analysis

Finding E_1. Finding the cheapest edge incident to each vertex $v \in G$ requires us to inspect all edges incident to v, remembering the cheapest one seen so far, and finally adding the cheapest edge at the end of the scan to E_1. Given an adjacency-list representation of G, we can scan the list of edges incident to each vertex $v \in G$ in $O(\deg(v))$ time. Summing this cost over all vertices gives a cost of $O(\sum_{v \in G} \deg(v)) = O(m)$ for the total cost of constructing E_1 (because $\sum_{v \in G} \deg(v) = 2m$).

A small detail to take care of is the fact that an edge may be the cheapest edge incident to both its endpoints (in fact, this is true for at least one edge). We need to ensure that each such edge is added to E_1 by at most one of its endpoints. There are several ways to do this:

1. The first option is to assume that our adjacency list representation has room for a constant amount of auxiliary information to be associated with each edge (by allocating additional space for each edge record in the edge list that can be used whichever way the user sees fit). In this case, we store with every edge a flag that indicates whether it has been added to E_1 already. We start by scanning the edge list in $O(m)$ time and setting this flag to false for every edge. This takes $O(m)$ time. Then we run the above algorithm. When a vertex v is about to add its cheapest incident edge to E_1, it first checks the edge's flag to see whether it has already been added to E_1. If so, v does nothing; otherwise, it adds the edge to E_1 and sets its flag.

2. If we cannot associate arbitrary information with the edges directly in the adjacency list, then it is reasonable to assume that each edge has a unique ID between 0 and $m-1$. In this case, we can simulate the previous strategy by creating a bit vector of length m whose bits are all 0 initially. Initializing this vector takes $O(m)$ time. Then, when we are about to add an edge with ID e, we inspect the eth bit in the bit vector. If it is 0, we set it to 1 and add the edge to E_1; otherwise, we do nothing because the edge has already been added to E_1.

3. If edges do not have IDs but every vertex has an ID between 0 and $n-1$ (something we have to assume for the construction of G' anyway), then we can initially just add edges to E_1 without checking whether they have already been added to E_1. This potentially adds each edge to E_1 twice. We use the integer sorting algorithm mentioned in my email to sort the edges in E_1 by their endpoints in $O(|E_1|) = O(m)$ time. This stores multiple copies of the same edge consecutively, so a single scan of this sorted edge list suffices to discard all but one copy of each edge.

Constructing G'. To construct G', we need to first compute the connected components of H, which takes $O(n+m) = O(m)$ time. Here we opt for a representation of the connected components $H_1, \dots, H_k$ that labels the vertices of G so that every vertex in H_i receives the label $i-1$. Constructing the vertex

set of G' then amounts to adding k vertices in $O(k) = O(m)$ time and storing a pointer to the ith vertex in the ith slot of a lookup array V.

Next we scan the edge set of G. For each edge (u, v) such that u and v have different component labels, we retrieve the vertices u' and v' corresponding to these components from the lookup table and add the corresponding edge (u', v') with $\text{orig}((u', v')) = (u, v)$ to G'. This takes constant time per edge of G, $O(m)$ time in total, because identifying u' and v' requires two accesses into V and adding the edge (u', v') to G', given u' and v', takes constant time.

At this point, G' may have multiple edges connecting the same endpoints u' and v', one per edge in G whose endpoints belong to the connected components of H represented by u' and v'. We need to eliminate these duplicate edges and keep only the cheapest one among them. To do this, we use the integer sorting algorithm to sort the edges of G' by their endpoints in $O(m)$ time. This stores all edges with the same endpoint consecutively, so a single scan of this sorted edge list suffices to discard all but the cheapest edge between every pair of endpoints. This scan also takes $O(m)$ time.

Constructing E_2. Given the edge set E'' of an MST of G', we only need to scan this edge list and add $\text{orig}(e)$ to E_2 for each edge $e \in E''$. This takes constant time per edge in E'', $O(|E''|) = O(m)$ time in total, because accessing $\text{orig}(e)$ for each edge $e \in E''$ amounts to dereferencing a pointer.

The number of vertices in G'. Since G is connected, every vertex of G has at least one incident edge in G and thus chooses one of these edges to be added to E_1. Thus, every vertex of H also has an incident edge in H, which shows that every connected component of H has at least two vertices. Let $H_1, \ldots, H_k$ be the connected components of H, and let n_i be the number of vertices in H_i. Since these components are vertex-disjoint and H has the same vertex set as G, we have $\sum_{i=1}^{k} n_i = n$. Since $n_i \geq 2$ for all $1 \leq i \leq k$, this gives

$$\begin{aligned} \sum_{i=1}^{k} 2 &\leq \sum_{i=1}^{k} n_i \\ 2k &\leq n \\ k &\leq \frac{n}{2}, \end{aligned}$$

that is, H has at most $n/2$ connected components. Since G' has one vertex per connected component of H, this shows that G' has at most $n/2$ vertices.

In the analysis below, we need to apply this argument inductively, that is, if G_i is the input graph to the ith recursive call the algorithm makes, then we need to show that G_i has at most half as many vertices as the input G_{i-1} of the previous recursive call. Since our proof that G' has at most $n/2$ vertices uses the fact that G is connected, we can apply this proof inductively only if G' is also connected. This is what we show next. Assume for the sake of contradiction that G' is not connected. Then there exists a cut (U', W') of G' (that is, $\emptyset \subset U' \subset V(G')$ and $W' = V(G') \setminus U'$) such that no edge in G' has one endpoint in U' and one endpoint in W'. Each vertex $v \in V(G')$ represents a connected component of H; let V_v be the set of vertices in this component. Then define $U = \bigcup_{v \in U'} V_v$ and $W = \bigcup_{v \in W'} V_v$. (U, W) is a cut of G. Since G is connected, there exists an edge $(u, w) \in G$ such that $u \in U$ and $w \in W$. Let H_u be the connected component of H that contains u and let H_w be the connected component of H that contains w. Let u' be the vertex of G' that represents H_u and let w' be the vertex of G' that represents H_w. Then $u' \in U'$ and $w' \in W'$ and, by the definition of G', the edge (u', w') is an edge of G', a contradiction. This shows that G' is connected.

The total running time. Each recursive call constructs E_1 and G' and, from an MST of G', E_2. Thus, each recursive call takes $O(m)$ time. Let $G = G_1, \ldots, G_t$ be the input graphs of the recursive calls the algorithm makes. Then G_1 has n vertices. We also proved that, for $i > 1$, G_i has at most half as many vertices as G_{i-1}. Thus, by induction, G_i has at most $n/2^{i-1}$ vertices. Since the algorithm returns without making any further recursive calls when the input graph has only one vertex, G_{t-1} has at least two vertices, that is $2 \le |V(G_{t-1})| \le n/2^{t-2}$. This gives

$$\begin{aligned}\frac{n}{2^{t-1}} &\ge 1\\ n &\ge 2^{t-1}\\ 1 + \lg n &\ge t,\end{aligned}$$

that is, the algorithm makes at most $1 + \lg n$ recursive calls. Since the running time per recursive call is $O(m)$, the total running time of the algorithm is thus $O(m \lg n)$.

Correctness

As stated in the assignment question, if no two edges of G have the same weight, G has exactly one minimum spanning tree T. We do not prove this here but use this fact.

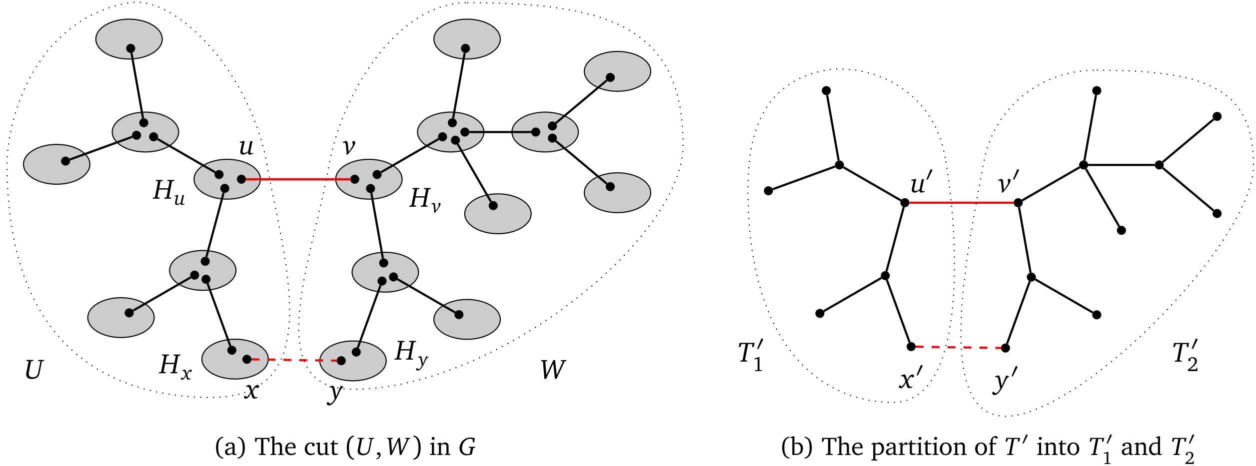
Edges in E_1 belong to T. Every edge $(u, v) \in E_1$ is the cheapest edge incident to at least one of its endpoints, say v. Thus, it is the cheapest edge that crosses the cut $(\{v\}, V(G) \setminus \{v\})$. Therefore, by the Cut Theorem, there exists a minimum spanning tree T' that includes the edge (u, v). However, T is the only MST of G, so $T' = T$ and $(u, v) \in T$.

Edges in E_2 belong to T. Consider an edge $(u, v) \in E_2$. Then there exists an edge $(u', v') \in E''$ such that $(u, v) = \text{orig}((u', v'))$ (see Figure 1). As illustrated in the figure, the removal of this edge splits the MST T' of G' computed by the recursive call on G' into two subtrees T'_1 and T'_2 such that $u' \in T'_1$ and $v' \in T'_2$. Let (U, W) be the cut of G such that $v \in U$ if and only if v belongs to a connected component of H whose corresponding vertex belongs to T'_1; otherwise, $v \in W$. Again, this cut is illustrated in Figure 1. We prove that (u, v) is the cheapest edge that crosses this cut. This implies once again that there exists an MST of G that contains (u, v) and, since T is the only MST of G, that $(u, v) \in T$.

So assume there exists a cheaper edge (x, y) that crosses the cut (U, W). Let H_x and H_y be the components of H that contain x and y, respectively, and let x' and y' be the vertices of G' that represent these components. Again, see Figure 1 for an illustration. Then w.l.o.g. $x' \in T'_1$ and $y' \in T'_2$. Moreover, since $x \in H_x$ and $y \in H_y$, we have $w((x', y')) \le w((x, y)) < w((u, v)) = w((u', v'))$. Thus, by adding the edge (x', y') to T' and removing the edge (u', v') from the resulting cycle, we obtain a spanning tree T'' of G' with $w(T'') = w(T') + w((x', y')) - w((u', v')) < w(T')$, a contradiction because T' is an MST of G'. This finishes the proof.

E_1 and E_2 are disjoint and have total size $n - 1$. To show that $E_1 \cap E_2 = \emptyset$, observe that, by the definition of H, every edge in E_1 has both its endpoints in the same component of H and, by the definition of E_2, every edge in E_2 has its endpoints in different components of H. Since no edge of G can simultaneously satisfy both conditions, this shows that every edge of G belongs to at most one of the two sets E_1 and E_2, that is, $E_1 \cap E_2 = \emptyset$.

To bound the size of $E_1 \cup E_2$, we first need to show that H contains no cycles. Assume that H does contain some cycle C. Then let $e = (u, v)$ be the heaviest edge in C and let e_u and e_v be the other two



(a) The cut (U,W) in G

(b) The partition of T' into T'_1 and T'_2

Figure 1: Illustration of the proof that every edge in E_2 belongs to T.

edges in C incident to u and v, respectively (see Figure 2). Since $w(e) \geq w(e_u)$, $w(e) \geq w(e_v)$, and no two edges of G have the same weight, we have $w(e) > w(e_u)$ and $w(e) > w(e_v)$. Thus, both u and v have a cheaper incident edge than e, that is, $e \notin E_1$, a contradiction.

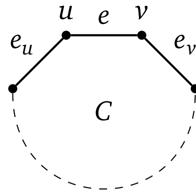


Figure 2: Illustation of the proof that H has no cycle.

Now let $m_1 = |E_1|$. Since H contains no cycles, this implies that H has $n-m_1$ connected components, that is, G' has $n-m_1$ vertices. The MST T' of G' thus has $n-m_1-1$ edges. For each edge of T', we add one edge to E_2. Moreover, for two edges $e_1 = (u,v) \in T'$ and $e_2 = (x,y) \in T'$ with $e_1 \neq e_2$, we cannot have $u = x$ and $v = y$ (because then they would be the same edge). This implies that $\mathrm{orig}(e_1)$ and $\mathrm{orig}(e_2)$ cannot have their endpoints in the same connected components of H, that is, $\mathrm{orig}(e_1) \neq \mathrm{orig}(e_2)$. Therefore, $|E_2| = n-m_1-1$. This shows that $|E_1 \cup E_2| = |E_1| + |E_2| = m_1 + n - m_1 - 1 = n-1$ (because $E_1 \cap E_2 = \emptyset$).

$(V, E_1 \cup E_2)$ is an MST of G. We have shown that every edge in $E_1 \cup E_2$ belongs to the MST T of G and that $|E_1 \cup E_2| = n-1$. Since T has $n-1$ edges, this implies that $E_1 \cup E_2$ is *exactly* the edge set of T, that is, the tree $(V, E_1 \cup E_2)$ computed by the algorithm is T.

A final note. This algorithm is called Borůvka's algorithm.