CS2800 Design and Analysis of Algorithms
Problem Set #7 March 13, 2023

1. Let $\mathcal{S} = \{a, b, c\}$ be an alphabet with frequencies $f[a]$, $f[b]$ and $f[c]$. Which of the following codes are valid Huffman codes, and why? If they are valid, give values for $f[a]$, $f[b]$ and $f[c]$ that will lead to that particular Huffman code.
   (a) $C(a) = 0, C(b) = 10, C(c) = 11$.
   (b) $C(a) = 0, C(b) = 1, C(c) = 00$.
   (c) $C(a) = 10, C(b) = 01, C(c) = 00$.

   Solution:

   Part (a): This can be constructed using Huffman coding with frequencies $f(a) = 0.5$, $f(b) = 0.25$, $f(c) = 0.25$.

   Part (b): This is not a prefix code.

   Part (c): The tree given by the code is not a full binary tree, and hence the code is not optimal.

2. Prove the following two properties of Huffman codes.
   (a) Show that if there is a letter in an alphabet with frequency more than $2/5$, then any Huffman code for that alphabet contains a letter that is encoded by a single bit.

   Solution: The crucial observation is that there are at most two letters in the alphabet whose frequencies are more than $2/5$. Therefore, these two will never be considered together in any step of Huffman coding. If they are put as siblings at the leaf level, then this means there is another letter with at least as high a frequency as these two, and we know that is impossible. So, one of the largest two must be encoded at the final step of Huffman coding, and will be encoded using a single bit.

   (b) Show that if every letter in an alphabet has frequency less that $1/3$, then for any Huffman code for that alphabet, there is no letter that is encoded by a single bit.

   Solution: Suppose that a letter i with is encoded using a single bit, and all letters have frequencies less than $1/3$. In the second-to last step, there were three characters. The reason i was not considered should be that both the other characters had frequencies at most $f(i)$. But then, the frequencies don't add up to 1.

3. Let $G(V, E)$ be a weighted graph with a weight function $w : E \to \mathbb{R}$. Instead of the minimum spanning tree, suppose that we are interested in the maximum spanning tree which is a

spanning tree with the maximum weight among all spanning trees. Does this problem have a greedy algorithm? If yes, design the algorithm and prove its correctness. If no, then explain why.

Solution:

The question is to describe and prove a greedy choice property for constructing maximum spanning trees.

Greedy choice: Let $G(V, E, w)$ be any connected, weighted, undirected graph with distinct edge weights. Let $S \subseteq V$ be any subset of vertices. If $e = (u, v) \in E(S, \overline{S})$ is the edge of maximum weight in the cut, then e is present in the maximum spanning tree of G

Prove this by imitating the proof of the cut-property we saw in class.

4. We will see an alternate algorithm to find an MST in a connected undirected graph $G(V, E)$ using the following observation.
   (a) Prove the following statement: Let C be any cycle in the graph $G(V, E)$, and let e be the heaviest edge in C. Then there is an MST that does not contain e.
   (b) How will you use the above observation to design an algorithm to construct an MST?
   (c) What is the running time of your algorithm?

Solution:

Part (a) involves an exchange argument. Read the reverse-delete algorithm described in Kleinberg and Tardos.

5. Suppose that you are given a graph $G(V, E)$ with edge weights $w : E \to \mathbb{R}$ and an MST T of G. An update is given where an edge $e \in E$ has its weight modified from $w(e)$ to $w(e)'$. Describe an efficient algorithm to update the MST T to obtain a new MST. Clearly explain why your algorithm is correct.

Solution:

- If $e \in T$ and weight of e is decreased, then the MST does not change.
- If $e \notin T$ and weight of e is increased, then the MST does not change.
- If $e \in T$ and weight of e increased, then remove e, and use the cut-property to find the edge to be added to create a spanning tree.

- If $e \notin T$ and weight of e decreased. Add e to T, and remove the maximum weight edge in the cycle containing e.

Prove these statements using the cut-property and Part (a) from Problem 4.

6. Design an algorithm to find the second smallest (in weight) spanning tree in a graph $G(V, E, w)$.

   If you are given G and an MST T, then there is a linear-time algorithm to get the second smallest spanning tree. For this exercise, any polynomial-time algorithm will suffice.

   Hint: The second smallest spanning tree will differ from an MST in only one edge. Prove this and use it to design your algorithm.

Solution:

Let T' be the second smallest spanning tree, and suppose that it differs from T in at least two edges. Let (u, v) be the min-weight edge in $T - T'$. Thus $T' \cup \{(u, v)\}$ contains a cycle. Furthermore, since $|T - T'| \geq 2$, we have $|T' - T| \geq 2$ (since $|T| = |T'| = n - 1$), and hence there exists an edge $(u', v') \in T' - T$ in the cycle containing (u, v).

First note that $w(u', v') > w(u, v)$. To see this, assume that $w(u', v') < w(u, v)$. The graph $T \cup \{(u', v')\}$ contains a cycle, and let (u'', v'') be an edge in the cycle that is in T but not in T'. Such an edge must exist because otherwise T' will contain a cycle. If $w(u', v') < w(u'', v'')$, then $T \cup \{(u', v')\} - \{(u'', v'')\}$ will have smaller weight than T and that will contradict the fact the T was an MST. Therefore, $w(u', v') > w(u'', v'')$. But, this would mean $w(u'', v'') < w(u, v)$ contradicting the fact that (u, v) was the minimum weight edge in $T - T'$.

If $w(u', v') > w(u, v)$, then $T'' = T' - \{(u', v')\} \cup \{(u, v)\}$ is a spanning tree of weight lesser than T'. Furthermore, T'' differs in T' in exactly one edge, and $|T - T''| < |T - T'|$. Therefore T'' is different from T and this argument contradicts the fact that T' was the second smallest spanning tree.

7. In this question, we will try to show that for certain classes of graphs, like planar graphs, Boruvka's algorithm actually runs in time $O(n)$. To that end, let's look at a slightly modified version of the algorithm.

   First, for a weighted graph $G(V, E, w)$, a *contraction* of an edge $e = (u, v)$ (denoted by G/e) gives a new graph where the vertices u and v are replaced by a new vertex, all edges incident on u or v are now incident on the new vertex, and the parallel edges to the new vertex is replaced by the minimum-weight edge incident on it.

   Now, we can think of Boruvka's algorithm as first finding the minimum weight edges in each step, contracting all of them to create a new graph and finding an MST in the new graph G'.

(a) Show that if T' is the MST of G', and $G' = G/e$, then $T' \cup \{e\}$ is the MST of G.

Hint: This is very similar in nature to the recursive construction we saw for Huffman coding in class. Use similar ideas to prove the statement.

> **Solution:** Let $\widetilde{T}$ be an MST containing e. Take the spanning tree $\widetilde{T_e}$ of G/e given by $\widetilde{T}$. Therefore, $w(T') \leq w(\widetilde{T_e})$. Then, $w(T) = w(T') + w(e) \leq w(\widetilde{T_e}) + w(e) = w(\widetilde{T})$.

(b) Show that given an edge e, we can construct the adjacency list representation of G/e in time $O(n+m)$ where n is the number of vertices in G and m is the number of edges.

> **Solution:** To use this in the next section, we need slightly stronger statement: That we can create the adjacency list of the graph after one step of Boruvka's algorithm in $O(n+m)$ time. For this, create the forest with the edges chosen by one step of Boruvka's algorithm, and label each of the components of the forest with a BFS/DFS. Now for each label, maintain an array indexed by labels of other components. For label i, let its array be A_i. The entry $A_i[j]$ contains the minimum weight edge between the labels i and j. Additionally, we can maintain a set S_i that stores the labels to which i has an edge. Now for each edge $e = (u, v)$, first check if labels of u and v are identical. If they are, then they are not part of the graph for the next step of Boruvka's algorithm. Otherwise, if $A_u[v] = \infty$, then set $A_u[v]$ to be $w(u, v)$ and add label of v to the set S_u. Otherwise, if $A_u[v]$ is greater than $w(u, v)$, then modify accordingly. This was we can create the adjacency list of the graph after one step of Boruvka's algorithm in $O(n+m)$ time.

Call a class of graphs *nice*[1] if the following conditions hold:

- For any edge e, G/e is nice.
- A nice graph G on n vertices has $O(n)$ edges.

(c) Use Parts (a) and (b) to show the MST of a nice graph G on n vertices can be constructed using Boruvka's algorithm in $O(n)$ time.

> **Solution:** Each time, the size of the graph reduces by $1/2$. It takes $O(n)$-time to create the new graph after one step of Boruvka's algorithm if the original graph had n vertices. Thus the total running time to obtain the MST is $O(n+n/2+n/4+\ldots+1) = O(n)$.

[1] For instance, planar graphs are nice.