Intro to Algorithms HW 7

Greg Stewart

March 30, 2018

Q1.

Given an undirected, weight graph G, assume that no two edges have the same weight and prove that there is a unique MST for such a graph.

Let there be two different MSTs, M_1 and M_2. There is a set of edges which are contained in only of the two different MSTs, and from this set we consider the one with minimum weight, e_1, which is contained only in M_1. M_1 and M_2 are distinct, so $M_2 \cup e_1$ contains a cycle which has an edge that isn't in M_1, which we can call e_2. These two edges, e_1 and e_2, are different, and contained in only one of the two MSTs, so it must be true that the weight of e_1 is less than the weight of e_2.

Consider the spanning tree $S = M_2 \cup \{e_1\} \setminus \{e_2\}$. Based on what we already know, the total weight of S is less than that of M_2, which is a contradiction since M_2 is supposedly an MST. Therefore there must be a unique MST if all edges are weighted differently.

Q2.

Correctness.

First, the algorithm only deletes edges if they create a cycle in G—if they do not create a cycle they are not removed because it would disconnect the graph and no MST could be found. Since this algorithm will look at all edges in the graph, it is ensured that no cycles exist in the reduced graph. Thus the fully reduced graph is a spanning tree of G.

It must also be a minimum spanning tree. G must have a minimum spanning tree made up of some subset of E' which is itself a subset of E, and the set of remaining edges after each iteration in the algorithm. Let T be an MST, so $T \subseteq E'$. When an edge is deleted with this algorithm, there are two possible cases: the edge is in T or the edge is not in T. If it's not in T, then there are no issues. If the edge e is in T, then there's more to it. Deleting e causes T to become disconnected into two graphs T_1 and T_2, which, since e was deleted, must be connected by some path using edges from E'. So there is another cycle in E that is not in T, using some edge $f \in E'$. Then we have a new MST $T' = T - e + f$. Consider the weights of e and f, $w(e)$ and $w(f)$. There are three possibilities here. $w(e) > w(f)$ is impossible because the weight of T' must be less than that of T. $w(e) < w(f)$ is impossible because f would have been deleted before e and could not be part of T'. Thus we must have $w(e) = w(f)$, so T' is an MST. So, once all edges have been visited, we have $T' = E'$, so E' is the MST of G.

Running Time.

Sorting the edges can be done with some comparison sort, and will take $O(|E| \log |E|)$ time, and since $|E| = |V|^2$, this is $O(|E| \log |V|)$. There are $|E|$ iterations in the next part of the algorithm, where the only part contained that requires more than unit time is checking connectivity. This could be done with a BFS or DFS from some vertex in the graph, which takes linear time and would make the overall complexity $O(|E|^2)$.

Q3.

Given a weighted, undirected graph G and a subset of vertices $C \subseteq V$, describe a method to find a constrained MST such that vertices in C only appear as leaves in the MST.

First, construct a graph G' which contains the vertices not in C and the edges that connect them. Then, use Kruskal's algorithm to construct an MST from G', which we'll call T. If T cannot be constructed, then return that the MST we want is not possible. If it could be constructed, though, then continue. Create the set of edges c, v where $c \in C$ and $v \notin C$. Then, sort all the edges just created by their weights; let's call this sorted set of edges E'. Now, loop through all the edges (u, v) in E, from smallest weight to largest weight, and if the vertices are in different sets, we can add the edge to T and join the sets. This is similar to Kruskal's algorithm. After this is done, return T.

Kruskal's algorithm is $O(|E| \log |V|)$. The second part of the algorithm, where the MST is fully constructed using the set C, is $O(|E| \log |C|)$, which is less than or equal to the complexity of Kruskal's, so we still in the end have $O(|E| \log |V|)$.

Q4.

(a) *Show the frequencies of all characters sum to 1 for any n*

This is just a geometric series, so we can use a summation rule:

$$\begin{aligned}\sum_{i=1}^{n-1} \left(\frac{1}{2}\right)^i + \frac{1}{2^{n-1}} &= \frac{(1/2)(1-(1/2)^{n-1})}{1-1/2} + \frac{1}{2^{n-1}} \\ &= 1 - \frac{1}{2^{n-1}} + \frac{1}{2^{n-1}} \\ &= 1\end{aligned}$$

(b) *Show the huffman encoding for each character.* The huffman encoding for the most frequent character will be 0, then the second most frequent 10, the third most frequent 110, the fourth most frequent 1110, and so on, until the final two characters, which will be coded with $n-1$ bits each, where the one with the smaller index will be coded ending with 0 and having $n-2$ 1's before the 0, and the other will be coded as $n-1$ 1's.e.g., if c_i is the character index ordered by frequency, starting with the most frequent, then we have

c_i	code
c_1	0
c_2	10
c_3	110
c_4	1110
$\cdots$	$\cdots$
c_{n-1}	$11\cdots10$
c_n	$11\cdots11$

(c) *What is the expected number of bits per character?*

We need to derive an expression from

$$\sum_{i=1}^{n} l_i \cdot f_i$$

From (b), we can see that the length of the encoding for each character c_i is i, when ordered by frequency. Thus, $l_i = i$, so we can rewrite the above expression as

$$\sum_{i=1}^{n-1} i \cdot \frac{1}{2^i} + (n-1) \cdot \frac{1}{2^{n-1}}$$

since the last two characters have the same frequency and encoding length. Evaluated, this is

$$\begin{aligned}
\sum_{i=1}^{n-1} i \cdot \frac{1}{2^i} + (n-1) \cdot \frac{1}{2^{n-1}} &= \frac{1}{2^{n-1}}(2^n - n - 1) + \frac{n-1}{2^{n-1}} \\
&= \frac{2^n}{2^{n-1}} - \frac{n}{2^{n-1}} - \frac{1}{2^{n-1}} + \frac{n}{2^{n-1}} - \frac{1}{2^{n-1}} \\
&= \frac{2^n}{2^{n-1}} - \frac{2}{2^{n-1}} \\
&= \frac{2^n - 2}{2^{n-1}} \\
&= 2^{1-n}(2^n - 2) \\
&= 2 - 2^{2-n}
\end{aligned}$$