CS 170 HW 5

Due on 2018-09-30, at 9:59 pm

1 (★) Study Group

List the names and SIDs of the members in your study group.

2 (★★) Updating a MST

You are given a graph $G = (V, E)$ with positive edge weights, and a minimum spanning tree $T = (V, E')$ with respect to these weights; you may assume G and T are given as adjacency lists. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\hat{w}(e)$. You wish to quickly update the minimum spanning tree T to reflect this change, without recomputing the entire tree from scratch. There are four cases. In each, give a description of an algorithm for updating T, a proof of correctness, and a runtime analysis for the algorithm. Note that for some of the cases these may be quite brief.

(a) $e \notin E'$ and $\hat{w}(e) > w(e)$

(b) $e \notin E'$ and $\hat{w}(e) < w(e)$

(c) $e \in E'$ and $\hat{w}(e) < w(e)$

(d) $e \in E'$ and $\hat{w}(e) > w(e)$

3 (★★) Finding MSTs by Deleting Edges

Consider the following algorithm to find the minimum spanning tree of an undirected, weighted graph $G(V, E)$. For simplicity, you may assume that no two edges in G have the same weight.

```
procedure FindMST(G(V, E))
    E' ← E
    for Each edge e in E in decreasing weight order do
        if G(V, E' − e) is connected then
            E' ← E' − e
    return E'
```

Show that this algorithm outputs a minimum spanning tree of G.

4 (★★) Minimum Spanning k-Forest

Given a graph $G(V, E)$ with nonnegative weights, a spanning k-forest is a cycle-free collection of edges $F \subseteq E$ such that the graph with the same vertices as G but only the edges in F has at most k connected components. The weight of a k-forest F is the total weight of all edges in F. The minimum spanning k-forest is defined as the spanning k-forest of minimum weight.

Note that when $k = 1$, this is equivalent to the minimum spanning tree. For simplicity, in this problem you may assume that all edges in G have distinct weights.

(a) Define a j-partition of a graph G to be a list of j sets of vertices $\Pi = \{S_1, S_2 \dots S_j\}$ such that every vertex in G appears in exactly one of these sets. Define an edge (u, v) to be crossing a j-partition $\Pi = \{S_1, S_2 \dots S_j\}$ if the set in Π containing u and the set in Π containing v are different sets. For example, one 3-partition of the graph with vertices $\{A, B, C, D, E\}$ is $\Pi = \{\{A, B\}, \{C\}, \{D, E\}\}$, and an edge from A to C would cross this partition.

Show that for any k'-partition Π of a graph G, if $k' > k$ then the lightest edge crossing Π must be in the minimum spanning k-forest of G.

(b) Give an efficient algorithm for finding the minimum spanning k-forest. Your solution should include the algorithm description, proof of correctness, and runtime analysis.

5 (★★) Parallelizable Prim's

Given an undirected, weighted graph $G(V, E)$, consider the following algorithm to find the minimum spanning tree. This algorithm is similar to Prim's, except rather than grow out a spanning tree from one vertex, it tries to grow out the spanning tree from every vertex at the same time.

procedure FINDMST($G(V, E)$)
  $T \leftarrow \emptyset$
  while T is not a spanning tree **do**
    Let $S_1, S_2 \dots S_k$ be the components of the graph with vertices V and edges T
    For each i, let e_i be the minimum-weight edge with exactly one endpoint in S_i
    $T \leftarrow T \cup \{e_1, e_2, \dots e_k\}$
  return T

For simplicity, in the following parts you may assume that no two edges in G have the same weight.

(a) Show that this algorithm finds a minimum spanning tree.

(b) Give an exact upper bound (that is, an upper bound without using Big-O notation) on the worst-case number of iterations of the while loop in one run of the algorithm.

(c) Using your answer to the previous part, give an upper bound on the runtime of this algorithm.

6 (★) Huffman Coding

In this question we will consider how much Huffman coding can compress a file F of m characters taken from an alphabet of $n = 2^k$ characters x_0, x_1, ... , x_{n-1} (each character appears at least once).

(a) Let $S(F)$ represent the number of bits it takes to store F without using Huffman coding (i.e., using the same number of bits for each character). Represent $S(F)$ in terms of m and n.

(b) Let $H(F)$ represent the number of bits used in the optimal Huffman coding of F. We define the *efficiency* $E(F)$ of a Huffman coding on F as $E(F) := S(F)/H(F)$. For each m and n describe a file F for which $E(F)$ is as small as possible.

(c) For each m and n describe a file F for which $E(F)$ is as large as possible. How does the largest possible efficiency increase as a function of n? Give you answer in big-O notation.

7 (★★) Sum of Products

This question guides you through writing a proof of correctness for a greedy algorithm. You have n computing jobs to perform, with job i requiring t_i units of CPU time to complete. You also have access to n machines that you can assign these jobs to. Since the machines are in high demand, you can only assign one job to any machine. The jth machine costs c_j dollars for each unit of CPU time it spends running a job, so assigning job i to machine j will cost you $t_i \cdot c_j$ dollars (each job takes the same amount of CPU time to complete, regardless of which machine is used). Your goal is to find an assignment of jobs to machines that minimizes the total cost.

Assume the jobs and machines are sorted and have distinct runtimes/costs, i.e. $t_1 > t_2 > \ldots > t_n$ and $c_1 > c_2 > \ldots > c_n$.

(a) What assignment of jobs to machines minimizes the total cost? (Hint: What machine should we assign the longest job to? What machine should we assign the second longest job to? It might help to solve a small example by hand first.)

(b) Given an assignment of jobs to machines, consider the following modification: If there is a pair of jobs i, j such that job i is assigned to machine i', job j is assigned to machine j', and $t_i > t_j$ and $c_{i'} > c_{j'}$, instead assign job i to machine j' and job j to machine i'. Show that this modification decreases the total cost of an assignment.

(c) Use part b to show that the assignment you chose in part a has the minimum total cost (Hint: Show that for any assignment other than the one you chose in part a, you can apply the modification in part b. Conclude that the assignment you chose in part a is the optimal assignment.)

8 (★★★) Preventing Conflict

A group of n guests shows up to a house for a party, but some pairs of guests are enemies. There are two rooms in the house, and the host wants to distribute guests among the rooms, breaking up as many pairs of enemies as possible. The guests are all waiting outside the house and are impatient to get in, so the host needs to assign them to the two rooms quickly, even if this means that it's not the best possible solution. Come up with a linear-time algorithm that breaks up at least half as many pairs of enemies as the best possible solution. Provide a proof of correctness and a runtime analysis as well.

You can treat the input as an undirected graph $G = (V, E)$ where each vertex in V represents a guest and (u, v) is in E if u and v are enemies.