CS 170 Homework 5 (Optional)

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

2 Encoding emergencies

The basic intuition behind Huffman's algorithm (i.e., that frequent words have short encodings and infrequent words have long encodings) is at work in the English language. The words "I", "me", "you", "he", and "she" are all short, while the word "velociraptor" is quite long.

However, words like "fire!", "danger!", or "high-voltage!" are not short because they are frequent. Instead, time is precious in situations when these words are used.

To make things theoretical, suppose we have a file containing words numbered $1, \ldots, m$ occurring with frequencies $f(1), \ldots, f(m)$ and suppose that for each word i, the cost per bit of encoding of the word is $c(i)$, so that if we find a prefix code where word i has encoding length $l(i)$, the total cost of our encoding will be $\Sigma_{i=1}^{m} f(i) \cdot c(i) \cdot l(i)$.

Show how to modify Huffman's algorithm in order to compute the prefix code of minimum total cost.

3 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. For simplicity, you may assume that no two edges have the same weight (this applies to both w and $\hat{w}$).

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

4 Job allocation

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 j'th 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) Describe the assignment of jobs to machines that minimizes the total cost (no proof necessary).

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.

5 Preventing conflict

A group of n guests shows up at a house for a party, but the host knows that m pairs of these guests are enemies (a guest can be enemies with multiple other guests). 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 $O(n+m)$-time algorithm that breaks up at least half of the pairs of enemies.

Please provide an algorithm description, a proof of correctness, and a runtime analysis.

6 A divide and conquer algorithm for MST?

Is the following algorithm correct? If so, prove it. Otherwise, give a counterexample and **explain why it doesn't work**.

procedure FindMST(G: graph on n vertices)
If $n = 1$ return the empty set
$T_1 \leftarrow$ FindMST(G_1: subgraph of G induced on vertices $\{1, \ldots, n/2\}$)
$T_2 \leftarrow$ FindMST(G_2: subgraph of G induced on vertices $\{n/2 + 1, \ldots, n\}$)
$e \leftarrow$ cheapest edge across the cut $\{1, \ldots, \frac{n}{2}\}$ and $\{\frac{n}{2} + 1, \ldots, n\}$.
Return $T_1 \cup T_2 \cup \{e\}$.

7 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 $x_0, x_1, \ldots, x_{n-1}$ (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 $E(F) := S(F)/H(F)$. Among all files with m characters and alphabet size n, describe a file F for which $E(F)$ is as small as possible.

(c) Among all files with m characters and alphabet size 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 your answer in big-O notation.

