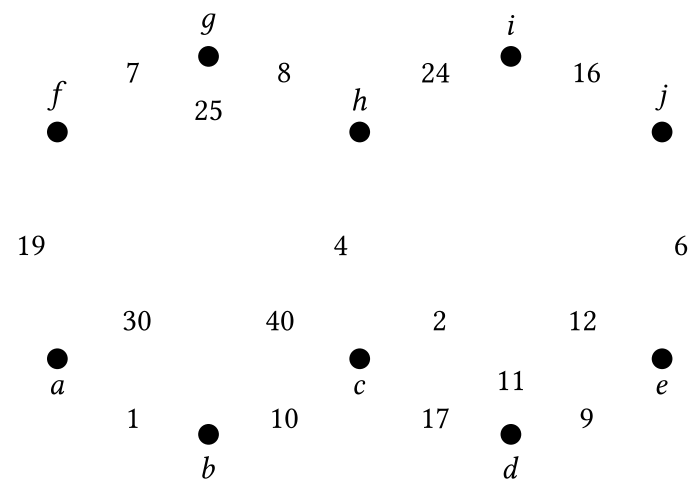
CS2800: Design and Analysis of Algorithms
Quiz #2
March 21, 2023

Max Marks: 20
Duration: 1 hour

1. Consider the following graph.



   (a) **(2 marks)** Give the sequence in which edges are added to the MST when Prim's algorithm is used in the graph starting with vertex b.

   (b) **(2 marks)** Give the sequence in which edges are added to the MST when Kruskal's algorithm is used in the graph.

   Solution:

   (a) The sequence is (a, b), (b, c), (c, j), (c, h), (j, e), (h, g), (f, g), (e, d), (j, i).

   (b) The sequence is (a, b), (c, j), (c, h), (j, e), (f, g), (g, h), (d, e), (b, c), (j, i).

2. **(5 marks)** Given a connected, undirected, weighted graph $G(V, E, w)$ where the edge weights are distinct, and an edge $e \in E$, give an efficient algorithm to check if e is present in the MST of G.

   Solution: Consider the graph $G'(V, E')$ where all the edges with weight less than $w(e)$ is present, and nothing else. We can prove the following: The edge $e = (u, v)$ is present in the MST iff there is no path from u to v in G'.

   $(\Rightarrow)$ Suppose the edge $e = (u, v)$ is present in the MST T, and there is a path from u to v consisting of edges whose weight is less than e. Then, $T - \{e\}$ consists of two trees. Take the path P in G' from u to v. There is some edge e' that goes from one tree to the other.

But, then $T - \{e\} + \{e'\}$ is a spanning tree of smaller weight, contradicting the fact that T was an MST.

($\Leftarrow$) Suppose that there is no path from u to v in G'. Let T be the MST of G, and $e \notin T$. Then, there is a path from u to v in T. Consider $T + \{e\}$. This contains exactly one cycle, and at least one of the edges in the cycle have weight more than $w(e)$. If we take any such edge e', then $T - \{e'\} + \{e\}$ is a spanning tree of smaller weight, contradicting the fact that T was an MST.

Thus, the algorithm is to construct $G'(V, E')$, and run BFS/DFS from u to check if v is reachable from u. This runs in time $O(|V| + |E|)$.

3. **(5 marks)** Suppose there are n jobs $J_1, J_2, \ldots, J_n$ such that job J_i requires t_i time to complete. Each of the jobs have a priority value; you can assume that the values are all distinct. For every pair of jobs J_i and J_ℓ, there is a switching time after i finishes and before ℓ starts. This could be different for different pairs of jobs. You want to do job J_k first, but unfortunately someone has already started the job J_s. You are only allowed to schedule a job lower in priority than the current job that is running. Give an efficient algorithm to check if the job J_k can ever be scheduled? If so, give an efficient algorithm to find the shortest time after the start of J_s that J_k can be scheduled.

Solution: Construct a graph G where the vertices are the n jobs. There is an edge from J_i to J_ℓ if ℓ has a lower priority that i and the weight of the edge is the switching time. Each vertex (job i) has a weight which is the time t_i for it to complete.

Since the priorities are all distinct, this graph G is a DAG because if there is a cycle $u_1, u_2, \ldots, u_r$, then priorities $p(u_1) < p(u_2) < \cdots < p(u_k) < p(u_1)$ which is not possible.

The graph G has weights for vertices, but this can be converted to an edge-weighted graph G' as follows: replace each vertex v with two copies v_0 and v_1. Add an edge from v_0 to v_1 with weight equal to the time for the corresponding job to finish. Now, all incoming edges to v are connected to v_0, and all the outgoing edges are connected to v_1. If G is a DAG, then G' is also a DAG for the same reason as mentioned above.

The shortest time after J_s such that J_k can be scheduled is the shortest path in the graph G' from s to k. Since this is DAG, we can apply the linear-time algorithm taught in class. Dijkstra's algorithm is also correct, but will incur an additional $\log n$ factor.

4. We will prove a property of Huffman codes in the question.

   (a) **(3 marks)** Let T be any full binary tree with n leaves. Define the number $f(i)$ as follows: $f(i) = 2^{-d_i}$ if the leaf i^{th} leaf in T is at depth d_i.

   Prove the following:

$$\sum_{i=1}^{n} \frac{1}{2^{d_i}} = 1.$$

Solution: Proof by induction: Base case is when $n = 1$. Here the tree is a leaf node at depth equal to 0, and the statement is true.

Consider the case when there are n leaves. Let r be the root, and assume that the left subtree L of r has $n_1 < n$ leaves and the right subtree R of r has $n_2 < n$ leaves such that $n_1 + n_2 = n$. Since T is a full binary tree, both L and R are also full binary trees. If a leaf node has depth d_i in T, then it has depth $d_i - 1$ in L/R. By the induction hypothesis, we have

$$\sum_{i=1}^{n_1} \frac{1}{2^{d_i-1}} = 1, \text{ and}$$
$$\sum_{i=1}^{n_2} \frac{1}{2^{d_i-1}} = 1.$$

Combining the two, we get

$$\sum_{i=1}^{n} \frac{1}{2^{d_i}} = 1.$$

(b) **(3 marks)** Given a rooted tree T with n leaves (you are given the adjacency list of the tree together with the id of the root node), design an efficient algorithm to check if there exists frequencies $f(1), f(2), \ldots, f(n)$ such that a run of the Huffman coding algorithm on these frequencies produces the tree T. Give a clear justification of why the algorithm is correct.

Solution: Firstly, note that if T is not a full binary tree, then it cannot correspond to any Huffman code. We will now show that if T is a full binary tree, then it must correspond to some Huffman code.

The proof is once again by induction. Let T be a full binary tree. Suppose for a leaf at depth d, we assign the value 2^{-d} as in Part (a). Part (a) shows that these are valid frequency values. We need to show that starting from these frequencies, the Huffman coding algorithm will construct the tree T.

Consider two leaves ℓ_1, ℓ_2 at maximum depth d in T. Let T' be the tree where these two leaves are deleted, and the their parent is made a leaf. This new leaf has depth $d - 1$, and inductively, the frequencies as in Part (a) constructs this tree T'. Since d is the maximum depth, when we run Huffman coding algorithm, the frequencies corresponding to ℓ_1 and ℓ_2 can be considered first, and then inductively, T' will be constructed. Thus, the final tree for the frequencies given by Part (a) will be the tree T.

This means that a rooted binary T with n leaves can correspond to a Huffman code iff the tree is a full binary tree. So, the algorithm is to obtain the parent pointers of the tree from the adjacency list, and check if the tree T is full. For this, you need to recursively check if both the left and right subtrees are full. This gives an $O(n)$-time algorithm.

general Key for Quiz #2 MAR 24

Yadu Vasudev 1:52 PM

@all The answer key for the Quiz #2 is available here. These are not the only correct answers. Alternate correct answers will also get credit.

@all A few things to note about the answers 2:02 PM

1. **Question 2**: It is not sufficient to show that the largest weight edge in a cycle is not part of the MST, and then check for the path in a graph with smaller weights. For proving the correctness, you also need to prove the other direction - that if there is no path in the graph with smaller edge weights, then the edge given is present in the MST. Both the directions are necessary for full credit.
2. **Question 4**: In Part (b), you need to prove that if you start with the frequencies as in Part (a), Huffman coding will construct the tree T. This is required for proving the correctness of algorithm, even though the final algorithm is just checking if a given binary tree is a **full** binary tree (not **complete**).

@all A few additional points about the answers that we evaluated. 4:45 PM

Question 2

In Question 2, finding all cycles containing an edge in a naive way can potentially take exponential time. For instance, in K_n, for every edge (u, v), (u, v) with every subset of the remaining $n - 2$ vertices forms a cycle. So just checking if e is the max-weight edge in some cycle by going over all cycles, might not be efficient.

Similarly, checking if the edge e is a min-weight edge in some cut by iteratively constructing it will require $O(E)$-time per step to find the min-weight edge unless you use your data structures carefully. So, this could be slower than the naive algorithm of running Prim's/Kruskal's algorithm.

Question 4

In Question 4, the point of Part (b) is to show that every full binary tree corresponds to a Huffman code. **This was not done in class**. All we showed in class was that the binary tree corresponding to an optimal prefix code will be full, not the other way.