Exam 2 sample solutions

1. (a) False *(If the graph is connected, then all vertices are in the same component)*
   (b) True *(You proved this for homework)*
   (c) True *(Same as for an unweighted graph)*
   (d) False *(If so, there would be a cycle)*
   (e) True *(If you had a spanning tree T' for G' with smaller total weight than the edges of T, then dividing all the edge weights in half would give you a spanning tree for G with smaller total weight for than T)*
   (f) $O(m \log n)$
   (g) False *(Think about a triangle)*
   (h) m

2. Given $G = (V, E)$, construct a graph G^{rev} by reversing each edge in G. Note that there is a path from x to vertex v in G^{rev} if and only if there is a path from v to x in G. Perform a BFS on G^{rev} starting at node x. If all nodes are marked as "discovered" in the BFS, then x is a ground vertex; otherwise, it is not.

   Assume as always that G has n vertices and m edges. We can construct an adjacency list for G^{rev} in time $O(n + m)$ as follows:

```
Initialize an n-element array with an empty neighbor list for each vertex
For each edge (u, v) in G
  add u to the neighbor list for v
```

   The BFS on G^{rev} requires time $O(n + m)$ as well.

3.
   - Base step: When $S = \{s\}$, $Dist[s] = 0$ which is correct.
   - Induction step: Let $k > 0$ and assume as an induction hypothesis that when $|S| = k$, `Dist[u]` is the length of the shortest path from s to u, for all u in S. Let v be the next vertex added to S by the algorithm. To complete the induction step, we just need to show that `Dist[v]` is the length of the shortest path to v.

     So let P be any path P from s to v. Since $s \in S$ and $v \notin S$, this path must include some edge (x, y) for which $x \in S$ and $y \notin S$.

$$\begin{aligned} \text{Total length of } P &\geq Dist[x] + c(x, y) \text{ since edge weights are positive} \\ &\geq Dist[u] + c(u, v) \text{ by choice of } v \\ &= Dist[v] \end{aligned}$$

     Since P was an arbitrary path from s to v, this proves that $Dist[v]$ is the length of the shortest path from s to v.

4. Define a graph G in which V is the set of children and $\langle i, j \rangle$ is an edge if and only if i throws rocks at j. The basic algorithm is a slight modification of a topological sort: the first "row" consists of kids at whom no one throws rocks. We then remove those kids, and look for the remaining kids at whom no one throws rocks; that becomes the second row, and so on.

```
i = 1
While V is not empty
  Let R_i be the set of nodes with no incoming edges
  If R_i is empty
        return null
  Else
        V = V - R_i
        i = i + 1
return i
```

To get the runtime, we have to be a bit more precise about how we find the nodes with no incoming edges in each iteration. To do this, use an `IncomingCount` array as in the topological sort algorithm in the text. When removing an node, we reduce the `IncomingCount` for each adjacent vertex. If the incoming count goes down to 0, then that vertex goes in the next row.

```
Initialize an array IncomingCount[v] = (indegree of node v)
Put each node with indegree 0 in list R_1
count = size of R_1
i = 1
While count < n
  If R_i is empty
    return null
  Else
    For each node x in R_i
      For each outgoing edge (x, y)
        Decrease IncomingCount[y]
        If IncomingCount[y] is zero
          Put y into R_{i+1}
          count = count + 1
    i = i + 1
return i
```

Initializing the incoming counts and R_0 requires iteration over all edges, which is $O(n+m)$. The loop processes each edge once, which is also $O(n+m)$.

5. Let $e = (u, v)$. Let S be the set of nodes reachable from u in $T - \{e\}$. Find the edge $e' = (x, y)$ in G', of minimum weight, that has $x \in S$ and $y \in V - S$, and let $T' = T - \{e\} \cup \{e'\}$. T' is the desired MST for G'

To identify e', we can:

(a) Initialize an n-element boolean array `Found[w]` to false. This is $O(n)$.

(b) Perform a BFS of $T - \{e\}$, starting at node u, and set `Found[w]` to true for each reachable node w. This is $O(n + m)$.

(c) Iterate over all edges and find an edge (x, y) of lowest weight such that `Found[x]` is true and `Found[y]` is false. Again, this is $O(n + m)$

Thus the entire algorithm is $O(n + m)$.

6. Here is a greedy algorithm: Start with x_1. Find an interval that covers it. Since we are greedy, select the interval $[a, b]$ that covers x_1 plus as much other stuff as possible, i.e., has the largest right endpoint b. Then, find the next x_j that isn't covered yet, and repeat. More carefully:

```
Let A be initially empty.
While at least one of the given integers is still uncovered
  Let y be the smallest one not covered by an interval in A
  Among intervals that cover y, select the interval [a, b] with the
    largest right endpoint b, and add it to A
```

Proof of correctness: Let $J_1, J_2, \ldots, J_k$ be the intervals of A in the order they were added by our algorithm, and for each index, let y_i be the smallest integer y that was not covered by $J_1, \ldots, J_{i-1}$ as identified in the algorithm.

Let Opt denote any optimal solution, that is, Opt is a set of intervals of minimum size that covers every integer x. We need to show that Opt and A have the same size. If every interval $J_1, J_2, \ldots, J_k$ is also in Opt, then Opt is at least as large as A, so A is optimal. Otherwise, let i be the first index such that $J_1, \ldots, J_{i-1}$ are in Opt but J_i is not in Opt. In constructing A, J_i was chosen to cover the integer y_i, the leftmost integer not covered by $J_1, \ldots, J_{i-1}$. Opt must also include some interval I^* that covers y_i. Removing I^* from Opt and adding J_i cannot leave any integer x uncovered: every integer to the left of y_i is covered by $J_1, \ldots, J_{i-1}$, and every integer to the right that was covered by I^* is also covered by J_i, because among all intervals that cover y_i, J_i is the one with the largest right endpoint b. Thus the modified set $\text{Opt} - \{I^*\} \cup \{J_i\}$ is still optimal.

Since the above exchange can be iterated until there is no interval in A that is not in Opt, we can conclude that A and Opt have the same size.