COSC-311 Sample Midterm Questions

Note: There will be four problems on the actual midterm. The problems on this handout are meant to give you a sense of the types of questions I might ask. This study guide is not comprehensive: there are topics we have covered in class that are not represented in the sample problems. Everything we have done in class or on homework is fair game for the midterm.

1 Runtime Analysis

(a) What does it mean for $f(n)$ to be big-O of $g(n)$?

(b) Here is a recurrence:

$$T(n) = 2T(n/3) + 4n$$

Prove by induction that $T(n) = O(n)$.

2 Stooge Sort

Consider the following sorting algorithm:

```
Stoogesort(A, i, j){
   if A[i] > A[j]
      swap(A[i], A[j])
   if (j - i + 1) > 2
         t = (j - i + 1) / 3
         Stoogesort(A, i, j-t)
         Stoogesort(A, i+t, j)
         Stoogesort(A, i, j-t)
   return A
}
```

(a) Write a recurrence for the runtime of this algorithm.

(b) Is Stooge Sort more or less efficient than Insertion Sort? Why? You may want to use the Master Theorem, draw a recursion tree, or use some other strategy to reason about the asymptotic runtime of Stooge Sort.

3 Knapsacks

In the Fractional Knapsack Problem, we had a set of n items, where each item i has value v_i and weight w_i. We also had a knapsack that could hold up to total weight W. Our goal was to maximize the value stored in the knapsack without exceeding the weight limit.

(a) What was the algorithm we used to select what fraction of each item i, p_i, to put in the knapsack?

(b) Now suppose that we must set $p_i = 0$ or $p_i = 1$ for each item; that is, we must take all of the item or none of it. This is called the 0-1 Knapsack Problem. Give a counterexample that shows that the algorithm you gave in part (a) no longer is guaranteed to find the optimal solution.

(c) Write a recurrence for a dynamic programming algorithm that finds the optimal solution to the 0-1 Knapsack Problem.

4 Minimum Spanning Trees

Ryan's cool new Minimum Spanning Tree algorithm works as follows on a graph with n vertices:

- Initialize set $T = \emptyset$ and $S = u$, where u is a randomly chosen vertex.
- Iterate $n - 1$ times: Let v be the vertex most recently added to S. Consider all edges from v to some other vertex x that has not yet been added to S. Among those edges, let (v, w) be the edge of lowest cost. Add edge (v, w) to T and add w to S.
- If v doesn't have any neighbors not already in S, go back to the second-most-recently added vertex, and keep backtracking until you find a vertex that has at least one neighbor not already in S. Use that vertex instead of v in this iteration.

Sadly this algorithm does not work.

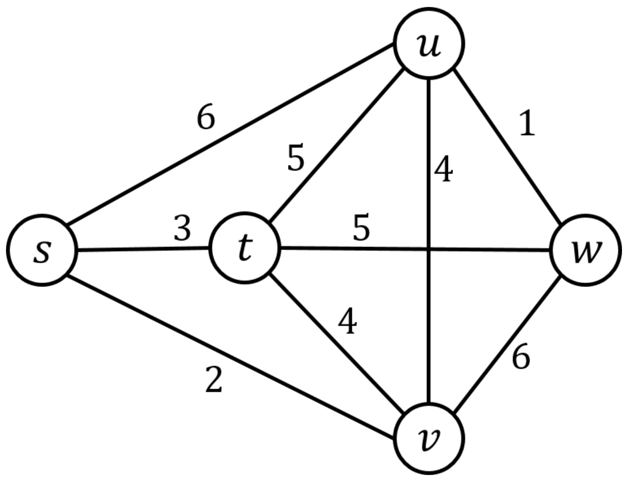
(a) State the theorem that tells us which edges are safe to add to a partial MST T.

(b) Explain why Ryan's algorithm doesn't fit the criteria for this theorem.

(c) Give an example of a graph with at least 3 vertices for which Ryan's algorithm returns the wrong answer. What does the algorithm do? What is the correct MST?

5 More Minimum Spanning Trees

Here is a graph:



(a) Draw the minimum spanning tree that would result from running Prim's algorithm on this graph, starting at vertex s. List the order in which edges are added to the tree.

(b) Draw the minimum spanning tree that would result from running Kruskal's algorithm on this graph. List the order in which edges are added to the tree.

6 Scheduling

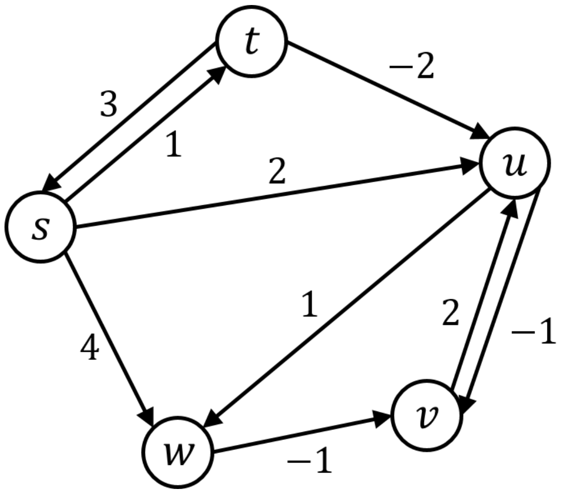
In the Interval Scheduling problem, we were given a set of jobs, where each job i required our resource starting at time $a(i)$ and ending at time $f(i)$, and only one job could be scheduled to use the resource at a time. If two jobs conflict, we must discard one of them. Our goal was to come up with a subset of jobs $\mathcal{S}$ such that there are no conflicts among the jobs in $\mathcal{S}$, and $|\mathcal{S}|$ is maximized.

(a) Dana suggests the following greedy algorithm: let $x(i) = f(i) - a(i)$ be the *size* of job i, and let $n(i)$ be the number of jobs conflicting with job i. Add to $\mathcal{S}$ the job i with the smallest $x(i) * n(i)$, then throw out any jobs conflicting with i and recur on the remaining jobs. Give a counterexample that shows that Dana's algorithm does not always produce the optimal solution.

(b) What is the correct greedy algorithm to solve this problem optimally?

7 Shortest Paths

Here is a graph:



(a) Run the Bellman-Ford algorithm to find the shortest path cost from s to every other node in the graph. Show the table that you fill in while running the algorithm

(b) How would you use the table you filled in part (a) to reconstruct the actual paths that produce the shortest path cost?