Homework 3 Solutions

Chris Wu

The following solution were prepared by me (Chris), so if you find a typos email me at wu@cs.nyu.edu and not the professor. One comment is that these solutions are *complete* and contain many steps that are written for explanatory purposes so don't worry if you didn't write everything here.

Question 1

Part a)

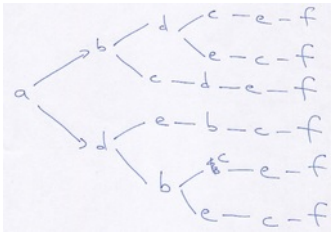
The orderings are

- abcdef
- abdcef
- abdecf
- adbcef
- adbecf
- adebcf

The way to envision topological orderings is akin to thinking about class prerequisites. One has to take pre-cal before cal which, in turn, is what you take before multivariable calculus. As such I'll refer to nodes as "tasks" and prerequisites as such or as "requirements".

So for the given graph, task a must be done first in any topological ordering. This is because every other task has a as a, possibly, indirect requirement. After choosing a, we are free to choose either b or d.

Here's the tricky part, let's say we choose b. d is still a viable option as well as, the now free, c. We can draw a tree of choices corresponding to this:



We notice that in part (a), we are really solving the problem of 2 lists of length 2 each (since we have no choice about the first and last tasks). So there are $\binom{4}{2} = 6$ choices, as we already knew.

Part b)

A linear list of nodes is a line of nodes with each node connected to their right neighbour (except the last one). How can we order two such lists with n_1 and n_2 nodes?

First, let's see what a topological ordering should look like. It will simply be a list of nodes from L_1 and L_2 mixed together. For example, if nodes from L_1 are denoted by 1 and nodes from L_2 are denoted by 2, then 11221121 is a unambiguous topological ordering. Why? Since L_1 is linear, all the 1's are the nodes from L_1 and their order is simply their order in the linear list. We have the same thing for L_2.

Now we've reduced the problem to something simpler. All we need to do now is count the number of different possible lists of this kind. So we want to know how many different lists there are of length $n_1 + n_2$ where we can choose the n_1 positions for the nodes of L_1. That's just $\binom{n_1+n_2}{n_1}$.

Question 2

For this question, we should make use of the standard DFS. Notice that we need to return a cycle (the edges) not just answer yes/no.

Consider the graph on 85 which depicts the DFS tree. Notice that the back edges signify the presence of a cycle. This is good but not enough to answer this question. We can also notice that when a back edge (say xy) is encountered, the DFS tree gives us a record how we got to y from x. So if we can keep track of the set of edges that got us here then we'd be set.

We'll augment the regular DFS with a stack and change DFS as follows: First, in DFS, whenever we're at a node x and visit a neighbour y, we'll push xy onto the stack. When we finish at a node and backtrack, we'll pop the an edge.

Second, when we reach a back edge (say xy), we stop. Then we pop the stack until we see the node x as an endpoint of some edge. The popped edges plus xy is our cycle.

Question 3

Part a)

We want to show that if G is a DAG then the algorithm returns the correct answer. I'll proceed by showing that if the answer returned is incorrect, then the graph had a cycle.

Assume the algorithm returns an invalid answer. Then the topological sort has a back edge (i.e. if the nodes of the topological order are lined left to right, then there is an edge that goes from right to left). Say that edge is (y, x) with y on the right.

Now, if there is a forwards path in the topological sort from x to y then we are done since that path and the back edge form a cycle. So let us assume there isn't one. This means that x and all it's descendants were explored by the DFS and assigned ranks before any of y ancestors. This means that y must have a lower number for a rank than x. This is a contradiction since we assume that y came after x in the topological ordering.

Part b)

Here's we make some small changes to the DFS in the homework. We change DFS(v) to be

$Rank(v) \leftarrow 0$
for For each u that is adjacent to v **do**
  if $Rank(u) = 0$ **then**
    Return and report a cycle
  end if
  if $Rank(u) < 0$ **then**
    DFS(u)
  end if
end for
$Rank(v) \leftarrow R$
$R \leftarrow R - 1$

Question 4

For both algorithms, if we have more than 1 choice for an edge, we choose the one that has lowest endpoint (if there is a tie we compare the other endpoints).

Kruskal's algorithm selects edges:

$$(1,3),(6,11),(10,11),(7,11),(11,12),(1,2),(3,8),(2,5),(3,4),(5,7),(9,10)$$

Prim's selects:

$$(1,3),(1,2),(3,8),(3,4),(2,5),(5,7),(7,11),(6,11),(10,11),(11,12),(9,10)$$

Notice that both trees are of weight 37, which is good otherwise something would be wrong. In fact, for this example, the trees are identical. This is not usual.

If we draw the run of both algorithms, we see that Kruskal's algorithm maintains a forest and unifies them over time. Prim's maintain one tree and adds edges until it is spanning.

There was initially an error in the pdf of homework 3 so if you got one without edges, get the new copy to see how this solution runs on that graph. If you didn't and assumed the graph was complete then it will be marked as such.

Question 5

Exercise 1 pg 188

This is true. Imagine that it were not true, i.e. that e^* was the cheapest edge but not in any MST. Let T be any MST (there's always at least one). Add e^* to it. You'll create a single cycle. Since e^* is cheapest then every other edge is more expensive. Take out of the most expensive one.

It can be shown that this new tree is cheaper and still a spanning tree. This contradicts the MST property of T. So e^* must belong to some MST.

Now, notice that we wanted to show that it belongs to *some* MST. In fact, this show that it must belong to *every* MST.

Exercise 2a pg 189

This is also true. Let G be the graph and G' be the graph with squared edges. If T is the MST for G, then there exists an ordering such that Kruskal's will output T. If we square all those edges, then that ordering is still valid and Kruskal's algorithm will return the same tree as the MST for G'.