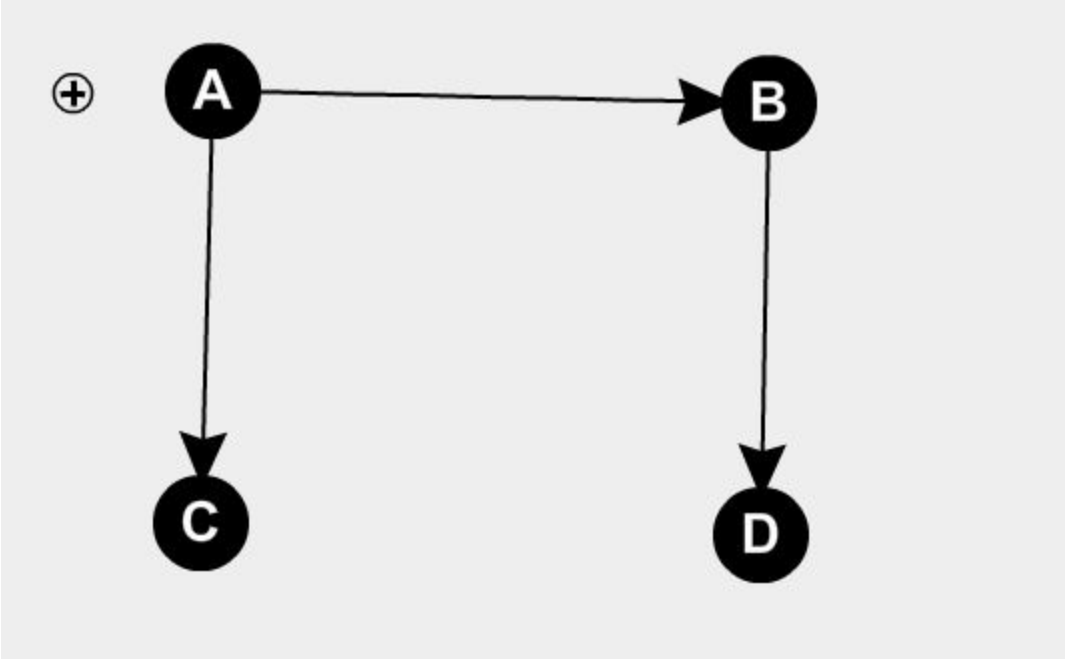
Stephen Chambers
CS858
April 4, 2016

Assignment 9 Writeup

1. Exercise 23.1–2 in CLRS.

Being a safe edge does not imply being a light edge. A safe edge is defined as an edge we can add without violating the minimum spanning tree invariant. A light edge is defined as the minimum cost edge across a cut. Consider the following poorly drawn example.



The edge [A,B] is a safe edge, but not a light edge. Therefore, being a safe edge does not also mean being a light edge. [B,D] is the light edge for this cut.

2. Exercise 23.1-9 in CLRS.

Suppose T' is not a minimum spanning tree. Let X' be the minimum spanning tree of graph G'. Note that the weight of T' is larger than the weight of X'. If X' is connected, add in the edges that are in T but not T' to X'. X' is now a spanning tree of the graph G. Because the larger weights of T' were not included in X', X' has a weight smaller than T, which was originally stated to be the minimum spanning tree of a graph G. This is false, so T' must be a minimum spanning tree. Proof by contradiction.

3. (Those in 858 only) Problem 23-4 in CLRS.

Part a:

This is the reverse of Kruskal's algorithm. Greedily taking the highest cost edge out of the minimal spanning tree is optimal. Each time an edge is removed from the connected graph, it is still a spanning tree. To prove that greedily taking the highest cost edge out of the connected graph results in the minimal spanning tree, consider a minimal spanning tree A. Another spanning tree A' can be made by taking out edge *x* and replacing it with edge *y*. Let the cost of *x* be larger than the cost of *y*. If *x* was taken out of the tree and replaced with *y*, A' would have a lower cost than A and would thus be the minimal spanning tree. Proof by contradiction.

<u>This algorithm solves for a minimal spanning tree.</u>

An efficient implementation would be to store the edges in a max heap, since we are only looking at one edge at a time. Additionally, we can store the final minimum spanning tree *T* in a max heap. Insertion into the heaps would take O(2E) time, and deleting from *T* would take O(logE) time. Popping off each edge would take O(E) time as well. To check if a graph was connected, the union find data structure could be used, which takes log(V) time.

<u>The time complexity of the algorithm above is (ElogE)</u>

Part b:

This would not create a minimum spanning tree. The edges are taken in arbitrary order, and the only check made is that they are T U e has no cycles. An edge with an extremely high cost could be added into the minimum spanning tree before other smaller cost edges are considered. To prove this, consider a spanning tree A. Another spanning tree A' can be made by taking out edge *x* and replacing it with edge *y*. Let the cost of *x* be larger than the cost of *y*. If *x* was taken out of the tree and replaced with *y*, A' would have a lower cost than A and would thus be the minimal spanning tree. Proof by contradiction.

<u>This algorithm does not solve for a minimal spanning tree.</u>

An efficient implementation would be to store the edges in a simple list, since we are taking them in arbitrary order. An array of integers would determine whether or not a vertex was visited by the spanning tree. When adding a new edge, check if both vertices were already visited. If they were, there is a cycle.

The time complexity of the algorithm above is O(E).

Part c:

This algorithm would create a minimum spanning tree. In some ways, it is similar to a combination of the first two algorithms. Every edge will be added to *T*, and maximum cost edges in cycles will be removed. When *T* has no cycles at the end of this algorithm, it will be a spanning tree. Additionally, because the maximum cost edge of a cycle was removed each time one was encountered, the resulting spanning tree will be minimal.

This algorithm does solve for a minimal spanning tree.

An efficient implementation would use a max heap for MST, and an array of integers for cycle detection as described in part b.

Therefore, the time complexity for this algorithm is O(ElogE).

7. What suggestions do you have for improving this assignment in the future?

I thought the second question was far trickier than the ‘grads only’ one. I would have grads do question 2 and undergrads do question 3.