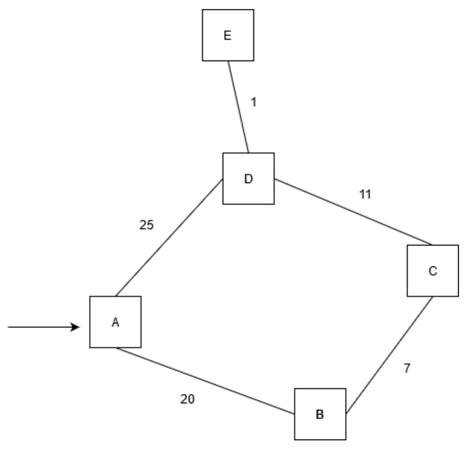
ADS 2021: Week 8 Solutions

Solutions for week 8 of Algorithms and Data Structures.



1 - Green Order:

Prim: A-B , B-C, C-D, D-E

Kruskal: D-E, B-C, C-D, A-B

Dijkstra (edges that are actually added): A-B, A-D, B-C, D-E

Dijkstra (the order of all edges considered): A-B, A-D, (B-A), B-C, (D-A), (D-C), D-E, (E-D), (C-B), (C-D)

4.3.1 - Green The MST does not depend on the absolute value of the weights but only on whether one weight is greater than the other or not, so we wish to prove that adding a positive constant to all weights or multiplying all weights by a positive constant does not change the order of the weights.

Mathematically we have:

$$x < y \Leftrightarrow c \cdot x < c \cdot y, \quad \forall c > 0$$

And:

$$x < y \Leftrightarrow c + x < c + y, \quad \forall c \in \mathbb{R}$$

So multiplying all weights by a positive constant or adding a positive constant (or any constant) to the weights does not change the order of the weights and will as such have no impact on the MST.

4.3.7 - Green One could multiply all weights by -1 and run Prim's or Kruskal's algorithm, which would return a maximum spanning tree.

(Alternatively, one could modify Algorithm 4.9 to initialize distTo as Double.NEGATIVE_INFINITY, turn all inequalities and use an IndexMaxPQ as pq. This would also return a maximum spanning tree but the above solution is preferred.)

4.3.12 - Green Does the shortest edge have to belong to the MST? Yes, consider Kruskal's algorithm. It always considers the shortest edge first and does not at any point overrule this, so it will always include this edge. Since all edges have distinct ways, the MST is unique so any other way of computing the MST must also include said edge.

Can its longest edge belong to the MST? Yes. Consider the graph in the first exercise. If we edit the edge D-E to have weight 100, it would still belong to the MST even though it is the longest edge.

Does a min-weight edge on every cycle have to belong to the MST? No, consider the following graph:

```
A - B
| X |
C - D
```

With edges having weights:
A-B = 1
B-D = 2
A-D = 3
A-C = 4
C-D = 5
B-C = 6

The MST is formed by A-B, B-D and A-C. There is a cycle, A-C-D-A, with the shortest edge being A-D. This is however not in the MST even though it is the shortest edge in a cycle.

4.4.1 - Green False. Adding a constant to every edge weight will add more weight to the paths that are composed of more edges, possibly changing the solution to the single-source shortest-paths problem.

4.3.3 - Yellow If a graph's edges all have distinct weights there is only one way to arrange the smallest V - 1 edge weights in increasing order when constructing the minimum spanning tree (either with Prim's or Kruskal's algorithm). Therefore, the MST is unique.

4.3.4 - Yellow Counterexample:

Consider a graph with 4 vertices, A, B, C, and D, and edges A-B, B-C and C-D with weight 1 and edge A-D with weight 4. The edge weights are clearly not distinct, yet the MST is unique.

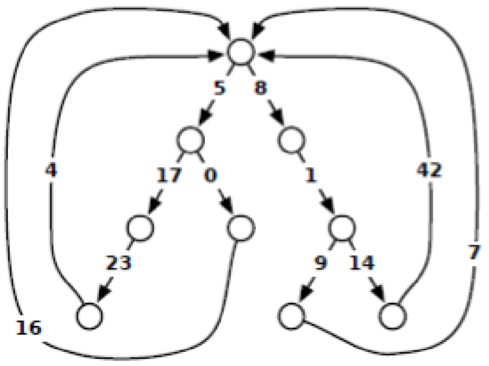
4.3.14 - Yellow First we check in linear time whether the deleted edge, was in the MST and if not, we just leave the MST as is. If it is, however, and say that the it is the edge from v to u, we can do a BFS traversal in the MST from v and mark all vertices, which is done in O(E+V) time, and since the number of edges is at least V-1 (as the graph is connected) we have $E + V \leq 2E - 1$, so the running time is still proportional to E. We know that deleting one edge from an MST results in exactly two subtrees, so we are interested in finding the edge with the smallest weight that connects the two subtrees. We do this by making a linear search through the edges and keeping track of the smallest edge with one marked and one unmarked vertex as we go through them.

Note: A previous version stated the following linearithmic solution: We can achieve this by adding all edges of G to a minPQ, and use this to find and edge that connects a marked vertex with an unmarked vertex.

Since this is a series of operations that are all proportional to E in running time, the overall process will also be proportional to E.

4.4.31 - Yellow First we identify the endpoints in the graph by going through all vertices and finding the two with degree 1. Then we arbitrarily choose one of these endpoints as a starting point, and traverse through the graph and for each vertex we store the accumulated weight from the starting point to that vertex in an array (We can use that for every vertex the accumulated weight is the weight of the previous vertex and the edge leading to the current vertex). Since identifying endpoints and computing the weights are both done in linear time, the entire preprocessing is also in linear time. To return the shortest distance between to vertices we just need to get fetch the accumulated weight and return the absolute value of one subtracted from the other.

Exam from May 31st 2012, question 4 A looped tree is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight.



A looped tree.

4.A - Green It is ...

4.B - Yellow The running time of Dijkstras algorithm is E Log(N), where N is the number of nodes and E is the number of edges. In a regular tree the number of edges is always N-1. A looped tree is just a regular tree with exactly one added edge per leaf, and the number of leaves in a binary tree is $\frac{N+1}{2}$. So the running time of using Dijkstras algorithm on a looped tree in terms of N is $(N - 1 + \frac{N+1}{2})Log(N) = O(NLog(N))$

4.C - Red A faster algorithm to find the shortest path from u to v can be done in linear time. First we need to identify the root of the looped tree. If this is not given we can iterate through the vertices to find the one with in-degree > 1, which can be done in linear time. Then we can reverse all edges in the graph in linear time and start at v and work backwards, keeping track of the accumulative weight, until we reach either u or the root. If we find u, then we have found the only path from u to v and we are done in linear time. If we end up at the root, then we need to find the shortest path from u to the root. We can then do a recursive function (using the unreversed tree) that examines each path from u to the root resulting in exactly one recursive call per node in the subtree that has u as root, which must be less than the overall number of nodes in the looped tree (since if u was the root of the looped, we would have been done in the previous step).

Since we do something linear, something linear and something linear we have an upper bound of the running time that is $O(N) + O(N) + O(N) = O(N)$ which is better than Dijkstras algorithm.