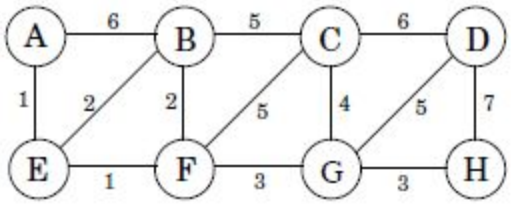
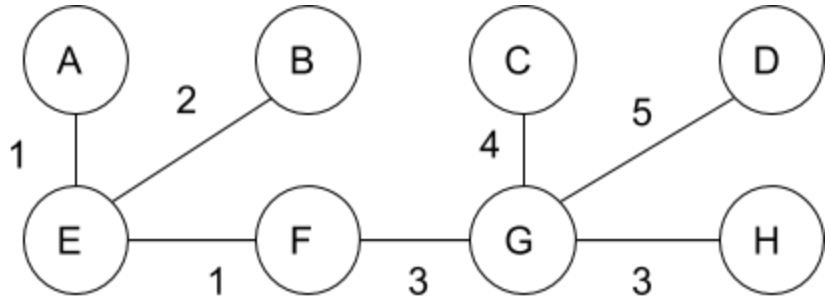
Aaron Taylor
Collaborators: John Fantell, Samm Katcher
Mohammed Zaki
Introduction to Algorithms
25 March 2017

Homework #5

1. **Q5.1**



a. The minimum cost of the tree is 19.



b. It has 2 minimum spanning trees. Edge EB could be substituted for FB.
c. Suppose Kruskal's algorithm is run on this graph. In what order are the edges added to the MST? For each edge in this sequence, give a cut that justifies its addition.

Note: A cut vertex is a vertex that when removed (with its boundary edges) from a graph creates more components than previously in the graph.

A cut edge is an edge that when removed (the vertices stay in place) from a graph creates more components than previously in the graph.

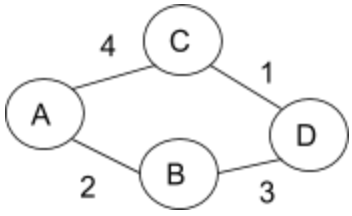
Edge	**Weight**	**Cut**
(A,E)	1	(ABCD,EFGH)
(E,F)	1	(ABCDE,FGH)
(E,B)	2	(AEFGH,BCD)
(F,G)	3	(ABE,CDFGH)

(G,H)	3	(ABCEF,GDH)
(G,C)	4	(ABEFGH,CD)
(G,D)	5	(ABCEFGH,D)

2. **Q5.3**
    - **a.** This all depends if G itself is a cycle. If G is a cycle then every node is reachable from each node in the graph. To solve this question, we could use a linear time cycle finding algorithm like Bellman-Ford O(VE). If we find a cycle and there are still edges left, we can then remove an edge while keeping G connected.
    - b. If we wanted the algorithm to be O(|V|), the graph would need to have to have no cycles and still be connected, leading to a tree structure. |E| for a tree would be |V|-1. This would lead to the exploration of at most |V| edges in the edge graph, leading to a runtime of |V|.
3. **Q5.5**

    Consider an undirected graph G = (V,E) with nonnegative edge weights we ≥ 0. Suppose that you have computed a minimum spanning tree of G, and that you have also computed shortest paths to all nodes from a particular node s ∈ V . Now suppose each edge weight is increased by 1: the new weights are w′e = we + 1. Does the MST or shortest path change?
    - a. If each edge weight increased by 1,then new MST weight = (old MST weight +|V|). The order in which the edges would be found would be the same if Kruskal's algorithm were applied.
    - b. The shortest path would change. If all edge weights were increased by one, the original path with the lowest cost would increase by |V|. Running Dijkstra's algorithm, the same path would be found but it would have a higher cost.
4. **Q5.9 a and g**
    - a. This claim is false. For example, if the heaviest edge is the only edge that connects a node to the graph, then the heaviest edge must be part of the MST.
    - b. This claim isn't necessarily true. While a shortest path algorithm like Dijkstra's are greedy like MST algorithms, they form their path/tree structures in slightly different ways. Dijkstra's algorithm looks for the shortest edge leading out of a node. MST algorithms look for the next shortest edge connected to a tree.

      Ex:

      :

      Shortest Path: Edges = {(A,B),(B,D),(A,C)},Weight: 9

      MST: Edges = {(A,B),(B,D),(C,D)},Weight: 6

5. **Q5.24**
    a. First, construct a graph G' using the subset of vertices V-U and the corresponding edges . Next we can run an MST algorithm like Prim's or Kruskal's on G' to get an MST. We will call the resulting MST T'. If T' cannot be created then G' nor T' can the lightest spanning tree. With T' created, we can then iterate through all vertices u in G' and using the makeset(u) function described in Chapter 5, sort the rank of each vertex based on weight. After, iterate through all edges (u,v) in G'. If two vertices u and v are disjoint, merge them using the union function described in Chapter 5 with tree T'. At the end we will return the MST of G' and T'.
6. **Q6.4**
    a. Let W[l] be the problem of decomposing s[·] into a sequence of valid words.
    W will be an array of booleans
    W[i]: W[i] is true is s[1..i] is a valid sequence.
    W[i] is false if anything else occurs
    If we want to find W[n], we need to solve our subpoblems. For our subproblem,s[1..i] is valid if s[1..j] is a valid sequence of words and s[j + 1...i] is a valid word. Basically our method is determining if a sequence of words consists smaller valid sequences. Using recursion we can break each sequence into subproblems and solve each.

```
Let W[1...n]
for k in range(1,n):
  W[k] = false #Initially set each Boolean to false
  for j in range(1,k):
    #determine value based on sub problem
    #if the word is in the dictionary and can be paired with
     others, it is valid
    if W[j-1] and dict(s[l+1...k]):
      W[j] = true
```

7. **Q6.8**
    a. The problem can be solved recursively in O(mn) time. We can break each string down into substrings using two indices, i for x and j for y. The problem can be represented as such:
    Let m and n be the lengths of strings x and y respectively such that 1 <= i <= m and 1 <= j <= n.
    LCS(i,j) = **{** LCS(i - 1, j - 1) + 1 if x_i == y_j**}**
    { 0 if x_i != y_j **}**
    Base Cases: LCS(0,0) = 0
    LCS(i,0) = 0
    LCS(0,j) = 0

Reasoning: The longest common substring between x and y must include the previous xth and yth indices that are the same. That is why the result will be 1 + LCS(i - 1,j - 1). If they are different, the result will be 0. Because we will have m*n subproblems and the work done during each problem will be constant, the resulting
runtime will be O(mn).

8. **Q6.14**