Algorithm

Homework 4

CHANG LIU

chang.liu@jhu.edu

April 10, 2012

1

If all of the graph’s edges have distinct weights, then when we add edge while creating the MST, we will always add the edge with the least weight which is also distinct in the graph. Thus we will only have one unique MST. We can prove it by introducing contradiction:

Suppose there are two different MSTs of G, say T1 and T2. Let e = E(v, x) be the min weight edge of G that is in one of T1 or T2, but not both. Let's suppose e is in T1. Adding e to T2 creates a cycle C. There is at least one edge, say f, in C that is not in T1 (otherwise T1 would be cyclic). By our choice of e, $w(e) \leq w(f)$. Since all of the edge weights are distinct, $w(e) < w(f)$. Now, replacing f with e in T2 yields a new spanning tree with weight less than that of T2, which contradicts the minimality of T2. Thus we have only one MST.

2

We can get the maximum spanning tree by modifying the Kruskal's algorithm: we add the edge with maximum weight instead of minimum weight for each iteration.

```
MAX-ST (G, w)
1   A = $\emptyset$
2   for each vertex $v \in G.V$
3       MAKE-SET(v)
4   sort the edges of G.E into non-increasing order by weight w
5   for each edge $(u, v) \in G.E$, taken in non-increasing order by weight
6       if FIND-SET(u) $\neq$ FIND-SET(v)
7           $A = A \cup \{u, v\}$
8           UNION(u, v)
9   return A
```

3

1. Apply the strongly connected component algorithm to find the different component in the graph.
2. Apply Kruskal’s algorithm on each of the component to find the MST separately.
3. Then we can combine all these trees we got from each component to form the minimum spanning forest.

4

5

For the sake of contradiction, suppose $T \bigcap H$ is contained in none of the MST of H. Let's say edge $e \in T \bigcap H$, then e should not be in any MST of H. Also $T - T \bigcap H$ should be the safe edge of the graphic G, assume the safe edges are E. Then $(T - T \bigcap H) \bigcap E$ will be the MST, but T is a MST of graph G, thus the contradiction occurs, and the statement is true.

6

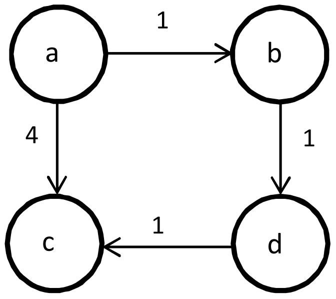
(1) This statement is ture.
Since there are more than |V|-1 edges in the G, there are must be a circle in the graph G, a graph with circle should obviously not be a part of the MST. Thus the statement is false.

(2) This statement is true.
As the MST algorithm adds the edge with their weights in nondecreasing order, an egde with the unique heaviest weight will always be the last to add to the MST if applicable. While in G the edge with the heaviest weight is in a cycle thus this edge will never be add into the MST as the other 3 edges will be add to the MST first. Also assume e is the edge with heaviest weight and is in the MST, replacing it with any other edges in the cycle will yield to a better MST. Thus the statement is true.

(3) This statement is true.
Let's say the edge with the minimum weight is e=E(v, x), also T is any minimum spanning tree does not contain e. Then if we add e to the T, it will create a cycle. Then if we remove any edge in this cycle, T will still be a spanning tree which even has a better minimum weight, as e's weight is less than any other edges in the original T. Thus the edge with the lightest weight will be the part of any MST. The statement is true.

7

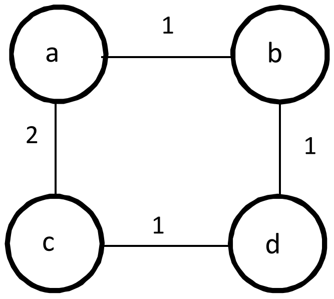
The statement is false.
A counterexample is showed as the following:



The shortest path for the above graph G is {a -> b -> d -> c}. If we add a constant weight to each edge, says 2, then the weight of path {a -> b -> d -> c} becomes 9, whereas the weight of e(a, c) becomes 6. So after add the constant weight to each edges in G, the shortest path becomes w{a -> c} = 6 instead of w{a -> b -> d -> c} = 9. Thus the statement is wrong.

8

The statement is false.
We can find a counterexample, consider a graph G as following:



The MST of graph G will be {a, b, d, c} and the path between V(a, c) is {a -> b -> d -> c}, weight 3. But the shortest path between V(a, c) is {a -> c}, weight 2 rather than {a -> b -> d -> c}. Thus the statement is false.

9

Both Prim and Kruskal's algorithm will still work if we allow negative value in the weights.

In the Kruskal's algorithm, the safe edge adds the current subset of a MST is always the least weight edge between 2 components, thus an edge with negative weight won't affect the algorithm as the algorithm will simply add the edge regardless whether it is positive or not.

In the Prim's algorithm, the safe edge add to the subset is the edge with least weight connecting the subset (which is a tree) to a vertex that is not in the current tree. As the algorithm only picks the edge with the least weight among all the valid edges, thus the negative value won't affect the algorithm.

10

```
DIJKSTRA-MOD(G, w, s)
1   INITIALIZE-SINGLE-SOURCE(G, s)
2   $S = \emptyset$
3   $Q = G.V$
4   $u = EXTRACT - MIN(Q)$
5   $S = S \cup \{u\}$
6       for each vertex $v \in G.Adj[u]$
7       RELAX(u, v, w)
8   while $Q \neq \emptyset$
9       $u = EXTRACT - MAX(Q)$
10      $S = S \cup \{u\}$
11      for each vertex $v \in G.Adj[u]$
12          RELAX(u, v, w)
```

This modified algorithm will not produce the path with maximum weight. This algorithm may perform infinite loop in the line 11 – 12. Because the RELAX() function will perform the add action on the weights and when there's a cycle in the graph, it will become an infinite loop.

11

To check whether there is an edge that can be removed while still leaving the graph connected, we need to find whether there's a cycle exists in the graph including that given edge.

Suppose the edge we need to check is e(u, v), and the graph is G, and {G-e} stands for the graph G without the edge e.

We just need to start from u, and search through {G - e} using DFS to see if we can reach v (find a path to v). If there's a path from vertex u to vertex v in the graph without edge e, then e must be contained in a cycle or there should not exist a path from u to v without e. Then we know that we can safely remove e from the graph while leaving the graph connected. This algorithm runs in linear time because DFS runs in linear time.

12

Prove by induction:
When n = 1, we only have one connected graph, thus the graph will have at least |V| – 1 edges to make the graph connected.

When n = k, assume the statement holds, there will be at least $|V|_k - k$ edges.

When n = k + 1, there will be one more unconnected graph as a component in the whole graph, thus we will have $|V|_k - k + |V| - 1 = |V|_{k+1} - (k + 1)$ edges. Thus the statement holds for n = k + 1.

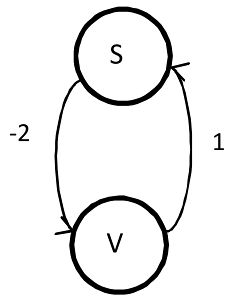
13

```
BELLMAN-FORD-MOD(G, w, s)
**for** $i = 1$ **to** $|G.V| - 1$
    **for** each edge $(u, v) \in G.E$
        $w = w(u) + w(v) + w(u, v)$
        RELAX(u, v, w)
**for** each edge $(u, v) \in G.E$
    **if** $v.d > u.d + w(u, v) + w(u) + w(v)$
        **return** FALSE
**return** TRUE
```

14

15

Dijkstra’s algorithm will NOT work under this case. We can give a counterexample as following:



The example above will make the Dijkstra’s algorithm fall into infinite loop in the RELAX() function, thus the Dijkstra’s algorithm will not work in this case.

16

(1) Prove by induction:

When n = 1, there is only one vertex in the graph, thus the degree of that vertex is 0 since no edges is there in the graph. Thus degree(v) = 0 = 2 |E|.

When n = 2, assume u, v are the 2 vertices in the graph, we have degree(u) = 1 and degree(v) = 1 since there is only 1 edge connect this 2 vertices. Thus degree(u) + degree(v) = 2 = 2|E|, the statement holds.

When n = k, assume the statement holds, we have vertices {v_1, v_2, v_3, … v_k} and m edges such that $\sum_{v_i \in V} d(v_i) = 2m$.

When n = k + 1, we will add v_{k+1} which will connect to n nodes into the set { v_1, v_2, v_3, … v_k }, we will have d(v_{k+1}) = n and d({v_1, v_2, v_3, … v_k}) = d({v_1, v_2, v_3, … v_k}) + n. Thus $\sum_{v_i \in V} d(v_i) = 2m + d(v_{k+1}) + \Delta d(\{v_1, v_2, v_3, \dots v_k\}) = 2m + n + n = 2(m + n) = 2|E|$, the statement holds for n = k + 1, thus is true.

(2) The sum of the vertices with even degree is obvious even. Let's assume the number of vertices with odd degree is not even (that is, odd). Under this assumption, the sum of the degree of all the vertices with odd degree is odd still odd. Thus the sum of the degree with all the vertices in the graph is even + odd = odd, this is contradict with what we had just proved in the problem (1), as the sum 2|E| should be an even number. Thus the number of vertices with odd degree is even.

17

Given a connected undirected graph G, we can always perform DFS on G to get a DFS tree. And there must be at least a vertex in the tree act as a leaf node. We can always remove the vertex in the leaf node of the DFS tree to make the graph G connected, because all the other vertices could still be visited from the source (root node) in the DFS tree as we only removed the leaf node. Thus the statement is true.

18

We can first perform a topological sort on the DAG, and then search to see if there is a path for every consecutive pair of the vertices, says (v_n, v_{n+1}). If we can reach the last element in the end, then there is a path that touches every vertex exactly once.