CS330HW7

ras70

November 2017

Problem 3A

Let G' be the graph that results from adding $e = (u, v)$ to the graph G ($G' = G \cup \{e\}$). The added edge e could potentially be in the minimum spanning tree (MST) of G' or not. Therefore, the algorithm will add e to the MST of G and remove it if necessary. The step of the algorithm are as follows:

1. Form the set of edges S_c by adding e to T. In other words, $S_c = T \cup \{e\}$.
2. Identify the cycle C in S_c. To do such perform a depth first search (DFS) on the edges in S_c starting at vertex u, keeping track of each separate path and visited nodes. In a given path P, if a vertex is revisited let C be P.
3. Find the maximum weight edge e_w in C. In other words find the e_w such that $w(e_w) \geq w(e)$ for all $e \in C$.
4. Form the tree T' by removing e_w from S_c. In other words, $T' = S_c - \{e_w\}$. T' is the updated MST for G'.

For clarity pseudo-code is given below:

```
updateMST(T, G, e) {

    S[]
    S = G.add(e)  //note: this is set addition

    C = identifyCycle(S, e);

    maxEdge = findMaxEdge(C)

    T’ = S.remove(maxEdge)
    return T’
}

identifyCycle(S, e) {
```

```
    e = (u,v)
    cycle[]
    DFS_cycle(u, cycle)
    return cycle
}

DFS_cycle(u, cycle) {
     Mark u as visited
     Mark u as in stack
     FOR each edge (u, v)
        IF v is in stack
            (u,v) is a back edge, found a cycle
            cycle = stack
        IF v is not visited
            DFS_cycle(v, cylce)
      Mark u as not in stack.
}
```

Clarifying remarks

In the above code the method *identifyCycle* passes an empty set to the *DFS_cylce* method by reference. The *DFS_cylce* method fills the content of this set when it identifies a cycle.

Running Time Analysis

There are two steps in the algorithm that take non-constant time - identifying the cycle once e is added to T and finding the maximum weight edge in C. These steps correspond to steps 2 and 3 in Problem 3A above. Steps 1 and 4 are simple additions and subtractions from a set, which take constant time.

Identifying the cycle is implemented with a DFS on the edges in T_c. The running time of DFS is $O(V+E)$ where V and E are the number of vertices and edges respectively in the set being searched. T_c is formed by adding an edge to the MST T. Therefore, T_c has n vertices and $nedges$. Thus, the running time to identify the cycle is $O(n+n) = O(2n) = O(n)$.

Finding the maximum weight edge in the set C can be accomplished by iterating through all of C's edges and keeping track of the maximum. This takes time proportional to the size of C. Since C is a subset of T_c, which has n edges, the number of edges in C is upper bounded by n. Therefore, finding the edge of maximum weight in C takes $O(n)$ time.

Since identifying the cycle and finding the edge of maximum weight in the cycle both take $O(n)$ time the algorithm takes constant, $O(n)$ time to run.

Problem 3B

Let G' be the graph created by adding the edge e to G, that is $G' = G \cup \{e\}$. Let T be the prior calculated minimum spanning tree (MST) of G and let T' the tree returned by the algorithm. It needs to be proved that T' is an MST of G'.

Consider the set Q of *all* possible cuts $(S, \bar{S})$ of G'. It can be shown that T' contains the minimum edge of each cut in the set Q. Let W be the set of all possible cuts $(S, \bar{S})$ of the original graph G. Let c indicate *all* existent elements the set Q (all such cuts in G'). The cut c can relate to W in the following 2 ways:

1. c could be contained in W.
2. c could not be contained in W. However, in this case it must be true that $c = c_w \cup \{e\}$ for some cut c_w existent in W.

These are the only 2 possible cases because G' differs from G only in the edge e. Therefore, each cut in G' must be identical to a cut in G or contain only the additional edge e.

Case 1 - $c \in W$

Let it first be established that T must contain one edge, a, in cut c because it is a minimal spanning tree. Additionally, since c is contained in W, which considers the graph G that does not contain e, the edge e cannot be in c. Therefore, e must either be in the set S or $\bar{S}$. By algorithm design T' must identically be T or be equal to $T \cup \{e\}$ minus one edge r. Given this, and the fact that e is not in c, T and T' can only differ by at most one edge across the cut c.

Let's consider the case where T and T' do not differ over the cut. In other words, $a \in T'$. Since a must be a minimal cost edge across c (given Problem 2), T' contains the minimal cost edge across c.

Let's consider the case where T and T' vary by one edge - where T' contains the edge b across cut c. However, no such b can exist because the algorithm only adds e to T'. The edge e has been determined to not be across the cut. Therefore, this is not a valid case.

Thus, it has been shown that if $c \in W$ then T' must contain the minimal cost edge in the cut c.

Case 2 - $c \notin W$

In this case e is included in the set c - if not c would be in W. Given T is a MST of G, which does not contain e, T must contain a minimal edge a across the cut that is not e. There are two distinct cases to consider - when the algorithm chooses T' to include e and when it does not.

T′ contains e

If T' contains e then the algorithm added e. The algorithm originally creates a cycle with the addition of e. Therefore, there must be at least one other edge across the cut c. If this was not the case the cycle that includes e would not exist.

Let r be the edge that the algorithm removes from the cycle.

If r is not in the cut c then a (the minimal edge across the cut in T) cannot be r (the edge removed). Therefore, T' must still contain a. If $w(a) \leq w(e)$ then edge a in the cut is still minimal and if $w(e) \leq w(a)$ the new edge e is minimal. However, since T' contains both e and a it must still contain the minimal edge in the cut.

If r is in the cut then the algorithm can potentially remove a minimal cut. However, since the algorithm removes the edge in the cycle of maximum weight, $w(r) > w(e)$. Therefore, if r is a then the minimal edge *before* the addition of e is removed. However, in this case e is the minimal cost edge, which T' contains. If r is not a then T' contains both a and e since r is the only edge removed from T to make T'. Therefore, T still contains the minimal cost edge regardless if it is a or e since it contains both.

$\mathbf{T'}$ does not contain e

If T' does not contain e then per the algorithm T' must have the same elements as T. Therefore, since T contains the minimal edge across the cut (per the conclusion of Problem 2) then T' must also contain the minimal edge across the cut.

Thus, it has been shown that for all cuts c such that $c \notin W$ T' will still contain one of the minimum cost edges in the cut.

Above it has been shown that for every cut in G', T' contains an edge e that is one of the minimal cost edges across the cut. Therefore, per the conclusions of Problem 2, T' must be a minimal spanning tree of G'.