# Intro to Algorithms HW6

Jiayao Wu jwu86

Mar 24th 2016

## 1 Problem 1

We can use proof by contradiction. Suppose the MST of graph $G$ contains the edge $e$ that has the maximum weight among other edges in cycle $C$. Then by deleting the edge $e$, the MST is split into 2 sub-trees. Then there is an edge $e' \neq e$ so that it reconnects the 2 sub-trees that each contain the end points of $e$. Since we know the weight of $e' < e$ because $e$ has the maximum weight, so this new tree we formed has less weight than the original MST, so this is a contradiction because the original MST is not the actual MST due to the fact that there is another tree that has smaller weight.

# 2 Problem 2

**Algorithm overview:**

Suppose the new edge $e$ has two end points $u$ and $n$. Use a graph traversal technique to traverse through the $T_{mst}$ to find a unique path that goes from $u$ to $n$. After find the path, we can find the longest edge along this path. Then compare this longest edge with $e$ to determine if $T_{mst}$ is also the MST for the new graph $G'$. If the longest edge's weight is larger than $W(e)$—weight of $e$, then this means there is a shorter path by replacing the longest edge with the new edge in $T_{mst}$ which means $T_{mst}$ is not the MST of graph $G'$. On the other hand, if the longest edge's weight is smaller or equal to $W(e)$, then this means $T_{mst}$ is still the MST of graph $G'$.

We can use a slight variation to BFS to find the unique path from $u$ to $n$.

**Pseudo Code:**

Let $Q$ be a queue

**Algorithm 1: Init(G, u)**

For all $v \in V$
    $v.parent$ = null
    $v.color$ = white
Finish for
$u.color$ = grey
enqueue $(Q, u)$

**Algorithm 2: BFS(G, u, n)**

Init$(G, u)$
while ($Q$ is not empty)
    $a$ = dequeue $(Q)$
    For all $x \in$Adj[$a$] given from $T_{mst}$
        if ($x.color$ = white)
            $x.parent$ = a
            $x.color$ = grey
            If $x = n$
                Terminate the while loop
            enqueue $(Q, x)$
    $a.color$ = black
Finish while loop
Comparison(G, u, n)

**Algorithm 3: Comparison(G, u, n)**

Now, start from $n$, trace back the parents till the parent is $u$. Meanwhile, keep track of $e_{max}$ accounts for the weight of the edge that has the max length

while ($p \neq u$ )
    $p = n.parent$
    If ($W(p, n) > e_{max}$)
        $e_{max} = W(p, n)$

$n = n.parent$
if $e_{max} > W(e)$
    return false
else
    return true

**Proof of correctness:**

$T_{mst}$ is a tree, so it does not contain any cycle. When we add an edge to the graph without adding extra vertices and if this edge is added to the $T_{mst}$ as well, this means there is a cycle called $M$ formed in $T_{mst}$. Since we proved in problem 1 that the largest edge in a cycle cannot exist in MST, then here within $M$, the largest edge cannot be in MST either. Furthermore, since $M$ contains the new edge $e$, so if weight of $e$ is the largest in $M$, then there is no need to change the original $T_{mst}$ and the original $T_{mst}$ is still the MST for $G'$. However, if the edge that has the maximum weight in $M$ is not edge $e$, then we need to replace that one with $e$ to have a shorter path, thus original $T_{mst}$ is not the MST for $G'$. This can be further easily proved by contradiction.

Assume $T'_{mst}$ is the new MST for $G'$ and $T'_{mst} = T_{mst}$, for the sake of contradiction, assume that our algorithm returns false. However, since we know $T'_{mst} = T_{mst}$, so the new added edge $e$ must has the largest weight among other edges in cycle $M$ as proven in previous paragraph, so the algorithm should return true. This contradicts with our assumption. Therefore, if $T'_{mst} = T_{mst}$, the algorithm must return true. Similarly, we can prove that if $T'_{mst} \neq T_{mst}$, then the algorithm should return false.

In addition, since this algorithm uses a slight variation to the BFS and as we already proved BFS is correct in class, then this algorithm is correct. The only difference is that at the end, I checked if $x = n$, if so, we terminate the whole while loop. This is so because we want the path from $u$ to $n$ and apparently that if $x = n$, we have touched our desired end point.

Lastly, algorithm 3 works correctly also because the parent pointer of each vertex stores where it is coming from in the BFS algorithm. So, if we start from the last vertex $n$ and trace back, we can reach $u$ in the end.

**Running Time Analysis**

The Algorithm 1 takes $O(\mid V \mid)$ because it's going over all the vertices.

In Algorithm 2, the step of Init$(G, u)$ takes $O(\mid V \mid)$ as proved above. Then in the while loop, since I am only using the Adjacency matrix of $T_{mst}$, $T_{mst}$ contains all the vertices, then it would be $O(\sum_{a \in V} deg(a)) = O(\mid V \mid)$. Afterwards, we proceed to Algorithm 3 when we trace back the parents from $n$, it's again at most taking $V$ steps because there are only $V$ vertices.

Therefore, as we sum up all the parts, the running time is just $O(\mid V \mid)$.