COL351
Assignment 1

Mallika Prabhakar, 2019CS50440
Sayam Sethi, 2019CS10399

August 2021

Contents

1 Question 1

Let G be an edge-weighted graph with n vertices and m edges satisfying the condition that all the edge weights in G are distinct.

1.a Unique MST

Question 1.a

Question. *Prove that G has a unique MST.*

Proof. We will prove this by induction on the size of G using an idea similar to Kruskal's algorithm discussed in the class.
Hypothesis:

$$h(n) : \forall\ G = (V, E) :\ |V| = n \implies MST(G)\ is\ unique \tag{1}$$

Base case: $n = 1$ is true since there is no edge and $MST(G) = (V, \phi)$ is unique.

Induction Step: Assume $h(n-1)$ is true for $n \geq 2$, now for $h(n)$:
(Note: This proof assumes each edge to be an unordered pair of vertices)

Consider Kruskal's algorithm,

Algorithm 1 Recursive MST Routine – Kruskal's algorithm

procedure MST(G)
    $e_0 \leftarrow (x, y)$ be edge with least weight
    $H \leftarrow G$
    remove x, y from H and add new vertex z
    for all v such that v is neighbour of x or y **do**
        add (v, z) to H
        $wt(v, z) \leftarrow \min(wt(v, x), wt(v, y))$
        if $wt(v, x) < wt(v, y)$ **then**
            $map(v, z) \leftarrow (v, x)$
        else
            $map(v, z) \leftarrow (v, y)$
        end if
    end for
    $T_H \leftarrow MST(H)$
    $T_G \leftarrow (V, \{e_0\})$
    for all $e \in T_H$ **do**
        if e is not incident on z **then**
            add e to T_G
        else
            add $map(e)$ to T_G
        end if
    end for
    return T_G
end procedure

In the above algorithm, it is clear that H has $n-1$ vertices. Thus, by our assumption, $h(n-1)$ is true and hence T_H is unique. Also, we know that T_G is a valid MST, from the correctness of Kruskal's algorithm. Now, assume by contradiction that T_G is not unique. Then there exists an MST, say $T' \neq T_G$.

Claim 1.1. *e_0 cannot be in T'*

Proof. This is because, if e_0 were in T', then $T \setminus \{e_0\} \neq T' \setminus \{e_0\}$ and thus, there would be two different MSTs for H which would be a contradiction to our assumption. Thus, $e_0 \notin T'$. □

Consider the path from x to y in T'. Since $e_0 = (x, y)$ is not present in T', there exists a different path, say $P = (f_1, f_2 \cdots, f_k)$ where $f_i \in E(T'), 1 \leq i \leq k$. We know that

$wt(f_i) > wt(e_0), 1 \le i \le k$.
Swap any of the f_i with e_0 and let the subgraph formed be T'', i.e., $T'' = T' \setminus \{f_i\} \cup \{e_0\}$.
We know T'' is a spanning tree of G since $V(T'') = V(G)$ and there are no cycles formed on performing the swap operation (this can be proven using contradiction as discussed in the lecture).
Now, consider the weight of T'':

$$\begin{aligned} & wt(T'') = wt(T') - wt(f_i) + wt(e_0) \\ \implies & wt(T'') < wt(T') \end{aligned} \tag{2}$$

We have shown that the total weight of T'' is lesser than the weight of T'. However, this is a contradiction to the fact that T' is the MST of G. Thus our assumption that T_G is not the unique MST of G was wrong. Therefore, $h(n)$ is true.

This completes the induction and the proof that *if all edge weights in a graph are distinct, then its MST is unique.* □

1.b Algorithm Sketch

Question 1.b

Question. *If it is given that G has at most $n+8$ edges, then design an algorithm that returns a MST of G in $O(n)$ running time.*

Solution. The idea is to use the previous result along with the fact that the number of edges to be removed to form a spanning tree is atmost $(n+8)-(n-1) = 9$, assuming that G was initially connected (else no MST exists). The algorithm is as follows:

Algorithm 2 Compute MST for 1.b

procedure MST(G)
  if $|E(G)|$ equals $|V(G)| - 1$ **then**
    return G ▷ since G is acyclic and hence a tree
  end if
  $C \leftarrow findCycle(G)$
  $e \leftarrow$ edge with largest weight in C
  remove e from G
  $T_G \leftarrow MST(G)$
  return T_G
end procedure

The procedure *findCycle* calls a DFS function on G which uses graph colouring and returns the first cycle it finds:

Algorithm 3 *findCycle*

```
procedure findCycle(G)
    v ← any vertex of G
    colour ← map of vertices initialised to zero
    parent ← map of vertices initialised to null
    (u, v) ← dfs(G, v, colour, parent, null)
                        ▷ returns the bottommost and topmost vertex of the cycle
    C ← empty array of edges
    add (u, v) to C
    while u ≠ v do
        add (u, parent(u)) to C
        u ← parent(u)
    end while
    return C
end procedure
```

The *DFS* function looks as follows:

Algorithm 4 Identify cycle using colouring and DFS

```
procedure DFS(G, v, colour, parent, p)
    parent(v) ← p
    colour(v) ← 1
    for all u such that u is neighbour of v in G do
        if colour(u) is 2 then
            return (u, v)
        else if colour(u) is 0 then
            value ← dfs(G, u, colour, parent, v)
            if value is not null then
                return value
            end if
        end if
    end for
    colour(v) ← 2
end procedure
```

□

2 Question 2

2.a two point one

2.b two point two

file for 2b

3 Question 3

3.a three point one

file for 3a

3.b three point one

file for 3b