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 ← (x, y) be edge with least weight
    H ← 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) ← min(wt(v, x), wt(v, y))
        if wt(v, x) < wt(v, y) then
            map(v, z) ← (v, x)
        else
            map(v, z) ← (v, y)
        end if
    end for
    T_H ← MST(H)
    T_G ← (V, {e_0})
    for all e ∈ 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, $e_0 \notin T'$. $\square$

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

$wt(f_i) > wt(e_0), 1 \leq i \leq k$.
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 $m - (n-1)$ which 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)
    while $|E(G)| > |V(G)| - 1$ **do**
        $C \leftarrow findCycle(G)$
        $e \leftarrow$ edge with largest weight in C
        remove e from G
    end while
    return 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*

```
1:  procedure findCycle(G)
2:      v ← any vertex of G
3:      colour ← map of vertices initialised to zero
4:      parent ← map of vertices initialised to null
5:      (u, v) ← DFS(G, v, colour, parent, null)
6:                          ▷ DFS returns the bottommost and topmost vertex of the cycle
7:      if DFS returned null then
8:          return null
9:      end if
10:     C ← empty array of edges
11:     add (u, v) to C
12:     while u ≠ v do
13:         add (u, parent(u)) to C
14:         u ← parent(u)
15:     end while
16:     return C
17: 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                    ▷ there is a forward edge from v to u
            return (u, v)
        else if colour(u) is 0 then                               ▷ u is unvisited
            value ← dfs(G, u, colour, parent, v)
            if value is not null then                ▷ cycle found in subtree of u
                return value
            end if
        end if
    end for
    colour(v) ← 2
    return null                                   ▷ no cycle found in subtree of v
end procedure
```

We will assume without proof that the *DFS* (Algorithm 4) function is correct and it takes $O(n+m)$ time since the algorithm is a standard one and has been discussed in the lecture.

Now consider the function *findCycle* (Algorithm 3), lines $3, 4$ take $O(n)$ time and line

5 takes $O(n+m)$ time. The while loop (lines 9–12) traverses up from u to v and each iteration takes $O(1)$ time. Therefore the entire while loop completes in $O(n)$ time (since the graph has n vertices and hence the loop cannot run for more than n iterations). Therefore the total time complexity of *findCycle* is $O(n+m)$.

We will now prove termination and compute complexity of Algorithm 2, which contains the code for computing MST:
Termination: The while loop terminates when $|E| = |V|-1$, that is, when the graph is a tree (since it assumes that the graph is connected). In each iteration of the while loop, *findCycle* returns a valid cycle since $|E| > |V| - 1$ and the graph is connected. After having found the cycle, we remove the edge with largest weight from G and therefore $|E|$ reduces by 1. Since $|V|$ remains constant, the while loop terminates after a finite number of steps.
Time Complexity: The while loop runs for exactly $m-(n-1)$ iterations, which is $O(m-(n-1)) = O((n+8)-(n-1)) = O(1)$ for the given constraints. Each iteration of the while loop calls *findCycle* which runs in $O(n+m) = O(n+(n+8)) = O(n)$. Finding the edge with least weight is $O(n)$ since a cycle cannot have more than n edges. Removing this edge from G can be implemented in as worse as $O(n)$ (better implementations in $O(1)$ and $O(\log(n))$ time exist but this won't change the complexity of the algorithm as we will show). Thus, each iteration of the while loop takes $O(n)$ time and the total time complexity of Algorithm 2 is:

$$\begin{aligned} T(MST) &= O(\text{iterations of while loop} \times \text{complexity of each iteration}) \\ &= O(O(1) \times O(n)) = O(n) \end{aligned} \tag{3}$$

Correctness: We now proceed to prove the correctness of the algorithm, using the following claim,

Claim 1.2. *If a cycle has edges of distinct weights, the edge with the largest weight can not be a part of any MST*

Proof. Let us assume by contradiction that the claim is false, then there exists an MST, say T such that the largest edge of cycle C (with distinct weights) is present in T. Let that edge be $e = (x, y)$. Now consider the paths from x to y in G. There exists atleast another path from x to y, which is exactly equal to $C \setminus \{e\}$, call it P. Consider the edge in P which is not in T, say $f = (p, q)$. We know such an edge exists since T is acyclic. Now, consider $T' = T \setminus \{e\} \cup \{f\}$. We will now prove that T' is a spanning tree using the following claim:

Claim 1.3. *Consider any edge $m = (a, b)$ in G which is not in T (spanning tree of G). Let $n = (u, v)$ be any edge on the unique path from a to b in T. Then on swapping m with n in T, we get another spanning tree of G.*

Proof. If $path_T(u, a)$ does not exist in T, then swap u, v (for ease of notation). Consider the graph $T \setminus \{n\}$. Define set

$$S = \{(c, d) \mid path_T(c, d) = \{k_1, k_2, \ldots, u, v, \ldots, k_l\}\} \tag{4}$$

All pair of vertices in this set are disconnected since all paths in the tree are unique. Now, consider the path

$$P_{T'} = \{c = k_1, k_2, \ldots, u\} \cup path_T(u, a) \cup path_T(b, v) \cup \{\ldots, d = k_l\}, \forall (c, d) \in S \quad (5)$$

Now, consider the graph $T' = T \setminus \{n\} \cup \{m\}$. All paths given by $P_{T'}$ are present in T' and thus, all pairs of vertices in S are connected in T'. Since all other paths are the same in T and T', T' is connected. Since $|E(T')| = |V(T')| - 1$, T' is a tree and also a spanning tree of G. This completes the proof of the claim. □

Thus, from Claim 1.3, we know that T' is an MST of G. Consider the weight of T':

$$\begin{aligned} & wt(T') = wt(T) - wt(e) + wt(f) \\ \implies & wt(T') < wt(T), \text{ since } wt(e) > wt(f) \end{aligned} \quad (6)$$

This is a contradiction to the fact that T is an MST of G. Therefore our assumption that Claim 1.2 is incorrect was wrong. This proves the correctness of Claim 1.2. □

Now consider Algorithm 2. In each iteration of the algorithm, we remove the largest edge of a cycle from G. Let the new graph obtained be G'. Therefore our algorithm transforms the problem from G to G'. We need to show that both graphs have the same MST.
From Claim 1.2, we know that e cannot be in any MST of G and from Question 1.a, we know that the MST of G will be unique. Therefore, the MST of G and $G' = G \setminus \{e\}$ will be the same. This completes the proof of correctness of Algorithm 2. □

2 Question 2

2.a Optimal Huffman Encoding

Question 1.b

Question. *What is the optimal binary Huffman encoding for n letters whose frequencies are the first n Fibonacci numbers? What will be the encoding of the two letters with frequency 1, in the optimal binary Huffman encoding?*

Solution. We begin by observing the property of Fibonacci numbers:

$$\begin{aligned} & f_n = f_{n-1} + f_{n-2} \ \forall n \geq 3 \\ \text{and, } & f_1 = f_2 = 1 \end{aligned} \quad (7)$$

We are given an alphabet $A = (a_1, a_2, \ldots, a_n)$ such that it has a frequency vector $F = (f_1, f_2, \ldots, f_n)$.Before finding the encoding, consider the sum of first k Fibonacci numbers, call it s_k:

$$\begin{aligned}
&s_k = f_1 + f_2 + \cdots + f_{n-2} + f_{n-1} + f_n \\
\implies &s_k = f_1 + f_2 + \cdots + f_{n-2} + f_{n+1} \\
\implies &s_k = s_{k-2} + f_{n+1} \\
\implies &s_k - s_{k-2} = f_{n+1} \qquad (8)
\end{aligned}$$

On performing telescopic summation over Equation 8 (for $k > 2$), we get the following:

$$\begin{aligned}
& s_k - \cancel{s_{k-2}} = f_{k+1} \\
+ \quad & s_{k-1} - \cancel{s_{k-3}} = f_k \\
+ \quad & \cancel{s_{k-2}} - \cancel{s_{k-4}} = f_{k-1} \\
& \vdots \\
+ \quad & \cancel{s_4} - s_2 = f_5 \\
+ \quad & \cancel{s_3} - s_1 = f_4 \\
\implies \quad & s_k + s_{k-1} - s_2 - s_1 = s_{k+1} - f_3 - f_2 - f_1 \\
\implies & (s_k + 1) + (s_{k-1} + 1) = (s_{k+1} + 1)
\end{aligned} \qquad (9)$$

This Equation 9 takes a form similar to Equation 7 and thus, $s_k + 1 = f_m$ for some m. On substituting value of $k = 1$:

$$\begin{aligned}
&s_1 + 1 = f_m \\
\implies &f_m = 2 \\
\implies &m = 3 \\
\implies &s_k + 1 = f_{k+2} \\
\implies &s_k = f_{k+2} - 1 \qquad (10)
\end{aligned}$$

Now consider the Huffman tree for $|A| = n$. Each of the frequency f_i $(1 \leq i \leq n-2)$ is less than f_n and sum of all frequencies f_i $(1 \leq i \leq n-2)$, i.e., $s_{k-2} = f_n - 1$ is less than f_n. We also know that a_i is merged at the same time or before a_j for any $i < j$. From this, we can formulate the merging strategy with the help of the following inductive claim:

Claim 2.1. *The optimal Huffman tree for A with frequency vector F is constructed in a way such that $(a_1, a_2, \ldots, a_i)$ is merged in the first $i-1$ steps $\forall i : 1 \leq i \leq n$.*

Proof. **Base case:** $i = 1$ is trivially true since a_1 is a leaf node and is *merged* in 0 merges.
Induction Step: Assume the claim is true for $i-1$. After $i-2$ merges, $(a_1, a_2, \ldots, a_{i-1})$ have been merged into $tree(a_1, a_2, \ldots, a_{i-1})$, and the frequency vector will be as follows,

$$\begin{aligned}
&F = (f_1 + f_2 + \cdots + f_{i-1}, f_i, f_{i+1}, \ldots, f_n) \\
&F = (s_{i-1}, f_i, f_{i+1}, \ldots, f_n) \\
&F = (f_{i+1} - 1, f_i, f_{i+1}, \ldots, f_n), \text{ from Equation 10} \qquad (11)
\end{aligned}$$

It is easy to see that the least two frequencies in the frequency vector are $f_i, f_{i+1} - 1$ which correspond to a_i and $tree(a_1, a_2, \ldots, a_{i-1})$. Therefore the $(i-1)^{\text{th}}$ merge will merge these two into $tree(a_1, a_2, \ldots, a_i)$.
We have shown that a_i is merged in the $(i-1)^{\text{th}}$ step and from induction we know that $(a_1, a_2, \ldots, a_{i-1})$ are merged before $(i-1)$ steps and thus, $(a_1, a_2, \ldots, a_i)$ are merged in $i-1$ steps. This completes the induction and proves the claim. $\square$

Therefore, from Claim 2.1, we know that a_n is merged in the last step (which is the $(n-1)^{\text{th}}$ step) and hence it is encoded using a single bit. We can now inductively define the encoding for each alphabet (for $n > 1$):

Claim 2.2. *a_i is encoded as $\underbrace{11\ldots1}_{n-i \text{ times}}0$ for $n \geq i > 1$ and a_1 is encoded as $\underbrace{11\ldots1}_{n-1 \text{ times}}$*

Proof. For $i > 1$, we will prove the claim using induction.
Base case: From Claim 2.1, we know that a_n will be merged in the last step and thus it is encoded using a single bit, we can choose this bit to be 0 and thus $enc(a_n) = \underbrace{11\ldots1}_{n-n \text{ times}}0 = 0$ and the claim is true for n.
Induction Step: Assume the claim is true for $i+1$, i.e., $enc(a_{i+1}) = \underbrace{11\ldots1}_{n-(i+1) \text{ times}}0$.
From the proof of the previous claim, we know that a_{i+1} and $tree(a_1, a_2, \ldots, a_i)$ are siblings and thus, the encoding of the root of $tree(a_1, a_2, \ldots, a_i)$ will be $\underbrace{11\ldots1}_{n-i \text{ times}}$.
From the base case, we know that a_n is encoded using a single bit with respect to the root of the tree. Therefore, with respect to $tree(a_1, a_2, \ldots, a_i)$, we know that a_i is encoded using a single bit. Let that bit be 0. We then have the complete encoding of a_i as:

$$\begin{aligned} enc(a_i) &= enc(tree(a_1, a_2, \ldots, a_i)).0 \quad (\text{. denotes concatenation}) \\ &= \underbrace{11\ldots1}_{n-i \text{ times}}0 \end{aligned} \tag{12}$$

This completes the induction for $i > 1$ and we now show the correctness of the claim for $i = 1$.
We know that a_1 and a_2 are siblings in the Huffman tree and thus they differ in their representation in exactly the last bit. Therefore, $enc(a_1) = \underbrace{11\ldots1}_{n-1 \text{ times}}$. This completes the proof of the claim. $\square$

Thus, we have computed the optimal Huffman encoding for the alphabet $A = (a_1, a_2, \ldots, a_n)$ which has frequency vector as $F = (f_1, f_2, \ldots, f_n)$ and we restate Claim 2.2:
In the optimal Huffman encoding for A with frequency F such that $|A| = n$, *a_i is encoded as $\underbrace{11\ldots1}_{n-i \text{ times}}0$ for $n \geq i > 1$ and a_1 is encoded as $\underbrace{11\ldots1}_{n-1 \text{ times}}$* (and for $n = 1$, $a_n = a_1 = 0$ trivially). $\square$

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