COL351 Assignment - 1

Aditya (2019CS50471)
Ajay (2019CS10323)

3^{rd} September, 2021

Contents

1 Minimum Spanning Tree

1.a Unique MST

Proof by Contradiction:
Let M_1 and M_2 be two MST's of G.
Let E be the set of edges which are in exactly one of M_1 or M_2.
1) If $|E| = \phi$, then both M_1 and M_2 have same edges. So, $M_1 = M_2$.
2) If $|E| \neq \phi$:
Let e be the edge in E with minimum weight. Without loss of generality, let e belongs to M_1. e = (x, y) $\in M_1$. let P be the path from x to y in M_2.
Claim: There will be an edge in Path P, which is not contained in M_1.
Proof: Let all the edges in P are in M_1. We know that M_1 also contains e. So, P.e forms a cycle. So, there should be atleast one edge in P that is not on M_1. Let that edge be e_1.
$e_1 \in E$ (because $e_1 \in M_2$ and $e_1 \notin M_1$).
weight(e) < weight(e_1) (because e is the minimum weight edge in E and given that all edge weights are distinct).
let M = $(M_2 \cup e) \setminus e_1$.
M is a spanning tree and weight(M) < weight(M_2). This implies that M_2 is not a MST.
This is a contradiction to our assumption that M_2 is MST.
So, there exists a unique MST if edge weights of a graph are distinct.

1.b Algorithm for MST

Algorithm Sketch:

```
Repeat the process 9 times:
    Find a cycle C in G
    Find the edge e with maximum weight in the cycle C using modified DFS
    If there is no such cycle, we found MST
    Or else Remove the edge e from G
```

Pseudo Code:

```
fun findMST(G):
    for i from 0 to 8:
        Arbitrarily choose any node as root
        edge e = modifiedDFS(e, root)
        if e = NULL:
            break
        G = G \ e
    return G
Maintain an array "parent", which stores the parent of each node in dfs.
fun modifiedDFS(G, root):
    visited[root] = true
    for each vertex v in adj(root):
        if visited[v] = false:
            set parent[v] = root
            edge e = detectCycle(G, v)
            if e is not equal to NULL
                return e
        if visited[v] = true and parent[v] is not equal to root:
            let z = LeastCommonAncestor(v, root)
            edge e1 = maximum weighted edge in path(v, z)
            edge e2 = maximum weighted edge in path(root, z)
            edge e = (v, root)
            edge ans = max(e1, e2, e)
            return ans
    return NULL
```

In function modifiedDFS, we are finding an edge with maximum weight cycle. So, if we come across a node which is already visited, we found a cycle. But u - v - u can't be a cycle. So, we are maintaining a parent array to check this condition. If we find a cycle between (u, v), we have to find the maximum weighted edge in the cycle. So, we find the least common ancestor of u and v. let w be the least common ancestor. We find the edge e1 with maximum weight in path (u, w) and edge e2 with maximum weight in path (v, w). Maximum weighted edge in the cycle will be maximum(e1, e2, edge(u, v)). modifiedDFS function, returns the edge with maximum weight in a cycle.

Time Complexity:

In each iteration of function findMST, we are finding edge with maximum weight in a cycle using modifiedDFS function. modifiedDFS function takes O(n) time, because:
1) We visit at most all the vertices which takes O(n).
2) If we find a cycle, we compute leastCommonAncestor which takes O(n). We compute maximum weighted edge in paths of cycle, which takes O(n). We do this step only once. This takes O(n).
3) So, modifiedDFS takes O(n) + O(n) = O(n) time.
We are iterating 9 times in findMST function, so total time complexity will be 9*O(n) = O(n).
So, this program takes O(n) time to compute MST.

Proof of Correctness:

In each step, we are finding a cycle and removing the maximum weighted edge in that cycle. We are repeating this until there is no cycle. So, the graph will be a spanning tree at the end.
Claim: G will be Minimum spanning Tree at the end.
Proof: Proof by contradiction.
In each iteration, we are removing maximum weighted edge in a cycle. Let this edge be e. Assume that there is an MST M with edge e in it. Let e = (x, y). Let P be another path between (x, y) in G (since x and y are vertices in a cycle). There will be an edge in P, that is not in M (because if all edges in P are in M, them P.e forms a cycle but M is MST). let that edge be e_1. We know that weight of every edge in P is less than weight of e, because e is the maximum weighted edge in cycle. So, weight(e_1) < weight(e). If we replace e_1 with e in MST, we get another spanning tree M_1. Also weight(M_1) < weight(M). So, M is not a spanning tree. This is a contradiction. So, there can't be any MST with e (maximum weighted edge of a cycle) in it.

2 Huffman Encoding

2.a Huffman encoding for Fibonacci numbers

Let A = [a_1, a_2,.....,a_n] be n symbols with frequencies F = [f_1, f_2,.....,f_n] where f_1, f_2,.....,f_n are first n Fibonacci numbers.
Claim 1: $f_n \leq f_1 + f_2 + ... + f_{n-1} \leq f_{n+1}$ for n ≥ 2.
Proof: By Induction
Base Case: $f_1 = 1$, $f_2 = 1$, $f_3 = 2$. So, $f_2 \leq f_1 \leq f_3$. So, our claim is true for base case.
Assume that $f_k \leq f_1 + f_2 + ... + f_{k-1} \leq f_{k+1}$
Induction Step:
let $x = f_1 + f_2 + ... + f_{k-2} + f_{k-1} + f_k$
$\Rightarrow x = f_1 + f_2 + ... + f_{k-2} + f_{k+1}$

$$\Rightarrow f_1 + f_2 + ... + f_{k-2} + f_{k-1} + f_k \geq f_{k+1} \tag{1}$$

From Hypothesis, $f_1 + f_2 + ... + f_{k-1} \leq f_{k+1}$
$\Rightarrow f_1 + f_2 + ... + f_{k-1} + f_k \leq f_{k+1} + f_k$
$\Rightarrow f_1 + f_2 + ... + f_{k-1} + f_k \leq f_{k+2}$

$$\Rightarrow f_1 + f_2 + ... + f_{k-2} + f_{k-1} + f_k \leq f_{k+2} \tag{2}$$

From (1) and (2), our claim is true for all n ≥ 2.

At each step of huffman encoding, we take two lettes a_i, a_j with least frequencies f_i, f_j and replace them with a new symbol $\tilde{a}$ with frequency $\tilde{f} = f_i + f_j$.

Claim 2: After k steps of Huffman encoding, F will become [$f_1 + f_2 + ... + f_{k+1}$, f_{k+2},.....,f_n] for k ≥ 1
Proof: By Induction
Base Case: In 1st step, a_1, a_2 have least frequency f_1, f_2. So, replace a_1 and a_2 with $\tilde{a_1}$ and freq($\tilde{a_1}$) = $f_1 + f_2$. So, after 1st step F = [$f_1 + f_2$, f_3,...f_n]
Assume that after p steps, F = [$f_1 + f_2 + ... + f_{p+1}$, f_{p+2},.....,f_n]
Induction step:
From Hypothesis, after pth step F = [$f_1 + f_2 + ... + f_{p+1}$, f_{p+2},.....,f_n]. From claim 1, $f_1 + f_2 + ... + f_{p+1}$, f_{p+2} are minimum frequencies in F. So, in (p + 1)th step, we replace them with a new frequency $f_1 + f_2 + ... + f_{p+2}$. So, F = [$f_1 + f_2 + ... + f_{p+2}$, f_{p+3},.....,f_n].
So, our claim is true for all k ≥ 1.

Using Claim 2, we can construct binary tree for Huffman encoding. The final binary tree will be:

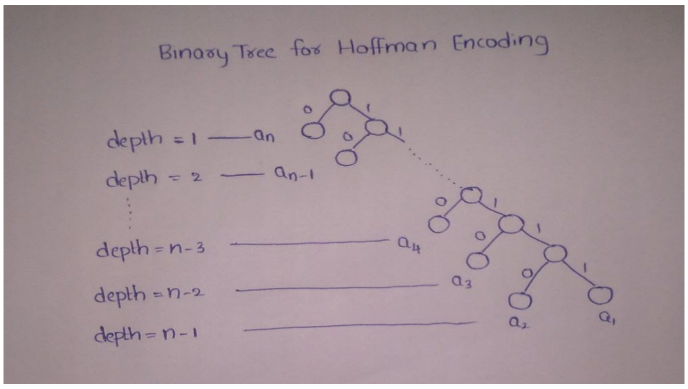

Figure 1: Binary Tree for Hoffman encoding of symbols whose frequencies are fibonacci numbers

For k > 1, the depth of symbol a_k will be (n + 1) - k and the Huffman encoding for symbol a_k will be:

$$\underbrace{111....}_{n-k \ \ times} \ \ 0 \quad (for \ \ k > 1)$$

For k = 1, depth of a_1 will be n - 1 and Huffman encoding will be:

$$\underbrace{111....}_{n-1 \ \ times} \quad (for \ \ k = 1)$$

The Huffman encoding for two symbols with frequencies 1 are:

$$\underbrace{111....}_{n-1 \ \ times}$$

$$\underbrace{111....}_{n-2 \ \ times} \ \ 0$$

2.b Huffman Encoding and Fixed Length Encoding

Let A = [a_1, a_2,.....,a_n] be n symbols with frequencies F = [f_1, f_2,.....,f_n]. Without loss of generality assume that F is sorted in increasing order. Given that the maximum character frequency is strictly less than twice the minimum character frequency. This implies that 2*f_1 > f_n.
We need to prove that Huffman encoding is same as fixed length encoding. In Fixed length encoding all the symbols are at the same depth from the root node. So, it is enough to prove that Huffman encoding forms a binary tree in which all the symbols are at same depth from root node.

At each step of Huffman encoding, we take two lettes a_i, a_j with least frequencies f_i, f_j and replace them with a new symbol $\tilde{a}$ with frequency $\tilde{f} = f_i + f_j$.

Claim - 1: Initially F = [f_1, f_2,.....,f_n]. After the first k steps, F = [f_{2k+1}, f_{2k+2},....,f_n, $f_1 + f_2$, $f_3 + f_4$,...,$f_{2k-1} + f_{2k}$] for 1 ≤ k ≤ n/2 and n = 2^m.
Proof: By Induction
Base Case: k = 1. Initially f_1, f_2 are minimum frequencies, so we replace them with $f_1 + f_2$. Since, 2*$f_1 > f_n \Rightarrow f_1 + f_2 > f_n$. So, after 1st step F = [f_3,....,f_n, $f_1 + f_2$]. So, base case is true.
Assume that Claim is true for k = i.
Induction Step: k = i + 1. From Induction Hypothesis, after ith step F = [f_{2i+1}, f_{2i+2},....,f_n, $f_1 + f_2$, $f_3 + f_4$,...,$f_{2i-1} + f_{2i}$]. f_{2i+1} and f_{2i+2} are the minimum frequencies in F, because f_{2i+1} ≤ f_{2i+2} ≤ ...≤ f_n (since we assumed that F is sorted) and $f_n \leq f_1 + f_2 \leq .... \leq f_{2i-1} + f_{2i}$ (from the given constraint 2*$f_1 > f_n$). So, in (k + 1)th step, we replace f_{2i+1}, f_{2i+2} with $f_{2i+1} + f_{2i+2}$. So, F = [f_{2i+3}, f_{2i+4},....,f_n, $f_1 + f_2$, $f_3 + f_4$,...,$f_{2i+1} + f_{2i+2}$]. So, our claim is true for all 1 ≤ k ≤ n/2.

Claim - 2: P(i) = Huffman encoding forms a binary tree with 2^i symbols are at same depth from root node is true for i ≥ 1.
Proof by Induction:
Base Case: i = 1. Let there are two symbols a_1 and a_2 with frequencies f_1 and f_2. So, the encoding will be 0 and 1 for a_1 and a_2. Both the symbols are at same depth from root node. So, P(1) is true.
Assume that P(k) is true i.e., Huffman encoding of 2^k symbols forms a binary tree in which all the symbols are at a same depth from the root node.
Induction Step:
Let there are 2^{k+1} symbols [a_1, a_2, a_3,....,a_n] with frequencies [f_1, f_2, f_3,....,f_n], n = 2^{k+1}.
From claim - 1, we can say that after n/2 steps, F' = [$f_1 + f_2$, $f_3 + f_4$,.....,$f_{n-1} + f_n$].
Now, the number of symbols is n/2 = 2^k. Also, $f_1 + f_2 \leq f_3 + f_4 \leq .... \leq f_{n-1} + f_n$ (since $f_1 \leq f_2 \leq ... \leq f_n$). Also, 2*($f_1 + f_2$) > $f_{n-1} + f_n$, because 2*$f_2 \geq$ 2*$f_1 > f_n > f_{n-1}$. So, the 2^k symbols with corresponding frequencies F' will be at same depth from root (from induction hypothesis).
Also, we can observe that depth(a_i) = 1 + depth of corresponding parent in whose frequency is in F'. We know that depth of all the symbols with corresponding frequencies in F' are same. So, depth of all the a_i's (1 ≤ i ≤ n) will be same.

3 Graduation Party of Alice

3.a Party Invitees

We use a greedy approach to solve the problem. If we find any person who knows less than 5 people or who has less than 5 unknown people (similar to knowing more than (n - 5) people), we remove the person from the list.

Algorithm Sketch:

Consider n people as n nodes and if two people know each other, add an edge between those two nodes. A graph G is formed.

Repeat the process until the G is not changed:

    Iterate over all the vertices
    If degree of a vertex < 5 or degree of vertex $>$ n - 5, remove the vertex from G

Pseudo Code:

```
#array "invited" of size n, stores whether a particular person is invited or not.
#If invited[i] = 1, then person i is invited, else person i is  not invited.
#Initially everyone is invited.
#Variable "flag" checks whether there is any change in the list of invitees from
#previous iteration to present iteration.
#If there is no change, we found the answer.
set all values of invited to 1.
flag = 1
while(flag = 1):
    flag = 0
    for each vertex v in G:
        if degree(v) < 5 or degree(v) > n - 5:
            removeVertex(v, G)
            flag = 1
            invited[v] = 0
            break
return invited

fun removeVertex(v, G):
    for each u in adj(v):
        adj(u).remove(v)
    G = G \ {v}
```

Proof of Correctness:

Let $I = I_1$ be the initial set of persons. Let I_2, I_3,...be the set of invitees after each iteration. I_k is obtained from I_{k-1} by removing a person in I_k.

Claim: let opt(I) be a set of valid invitees. We will prove that $opt(I) \subseteq I_k$ for all $k \geq 1$.

Proof: By Induction:

Base case: $opt(I) \subseteq I_1$, because I_1 is the entire set of invitees.

Assume that $opt(I) \subseteq I_j$

Induction Step:

I_{j+1} is formed from I_j by removing person p. Person p is removed by our algorithm from I_j because p is not satisfying either of the given two constraints, i.e., p knows less than 5 others in I_j or p has less than 5 unknown persons in I_j. From Induction Hypothesis, $opt(I) \subseteq I_j$. This implies that, p knows less than 5 others in opt(I) or p has less than 5 unknown persons in opt(I). Since, opt(I) is a valid set of persons, p is not contained in opt(I). So, $opt(I) \subseteq I_{j+1}$. So, out claim is true for all $k \geq 1$.

We terminate this algorithm when there is no invalid person in G. So, this implies that we get maximum subset of the persons who satisfies both the constraints.

This algorithm terminates because the no of persons is finite and we can remove at most n persons.

Time Complexity:

In each iteration, we are removing atmost one vertex. In total, we can remove atmost n vertices. So, outer loop runs O(n) times. In inner loop, we are iterating over all the vertices and if we find a vertex which is not satisfying the condition we remove that vertex. Removing all the vertices takes O(m) time overall. So, the total time complexity will be O(n*m). m = O(n^2). So, total time complexity will be O(n^3)

3.b Party And Dinner Tables

Algorithm :

```
Step - 1: Sorting the list of ages
create array A of size 90.
for age in ages:
    A[age - 10]++
create array sortedAges.
for i from 0 to 89:
    for j from 0 to A[i]:
        append (i + 10) to sortedAges
Step - 2:
set tables = 0
while sortedAges is not empty:
    set a to the first element of sortedAges
    remove first element from sortedAges
    set addedAges = 1
    while addedAges < 10 and sortedAges[0] - a <= 10:
        remove first element from sortedAges
    tables++
return tables
```

Proof:
let J be the list of ages of people that are getting invited to the party and opt(J) be the optimal number of tables required for them to get seated with the given conditions.let J_0 be the minimum age. We define overlap(J_0) as list of elements in J whose value is less than or equal to $2J_0$. If the number of elements in J whose value is less than or equal to $2J_0$ is greater than 10, we take only the minimum 10 elements into overlap(J_0) (because it is given that maximum no of people on each table is 10). Let J' = J \ overlap(J_0).
Claim - 1: There exist an optimal solution with overlap(J_0) in one table.
Proof: Let A be an optimal solution and let T be the table in which J_0 is placed. Let J_i be a person in overlap(J_0) but not in T. Let J_i be placed in table T'.
1) There exists a person in T (who is not in overlap(J_0)) whose age is greater than or equal to J_i because overlap(J_0) has persons with minimum ages. Let that person in T be J_k. We place J_k in T' and J_i in T. This won't break the conditions because no of persons in each table is same and age difference between person among each table won't exceed 10 because $J_i \leq J_k$.
2) If there is no such person, then | T | < 10, so we can simply remove J_i from T' and place it in T.
So, there exists an optimal solution with overlap(J_0) in one table.
Claim - 2: opt(J) = opt(J') + 1
Proof:
1) opt(J) $\leq$ opt(J') + 1:
Let A' be the optimal solution of J'. We can seat the persons in overlap(J_0) in one table because overlap(J_0) satisfies the conditions that $|J_0| \leq 10$ and maximum age difference among people in overlap(J_0) is at most 10.
J = J' $\cup$ overlap(J_0) and J' $\cap$ overlap(J_0) = ϕ. We are placing people of J' in opt(J) tables and people in overlap(J_0) in one table. So, this is a solution for J. So, opt(J) $\leq$ opt(J') + 1.
2) opt(J') $\leq$ opt(J) - 1 From claim - 1, there exists an optimal solution with overlap(J_0) in one table T. Remove the table T, and all the remaining people belong to J' since J' = J \ overlap(J_0). We are placing all the people in J' in opt(J) - 1 table. So, opt(J') $\leq$ opt(J) - 1.
Time Complexity:
Sorting takes O(n_0) time (where n_0 is the number of invited people). Then, in step-2 we are looking at each element once and then we are removing it from the list. So, we do O(n_0) time in step-2. So, overall time complexity will be O(n_0)