Assignment-1 Submission

Indian Institute of Technology Delhi

COL351: Analysis and Design of Algorithms

Name: Dishant Dhiman Entry Number: 2019CS10347 Group: 03

Name: Deepanshu Entry Number: 2019CS50427 Group: 04

1 Minimum spanning tree

1.1 Graph with distinct edges has unique MST

Proof by contradiction

Assume two MSTs T_1 and T_2 with n vertices and edges as f_1, f_2 .... f_{n-1} and g_1, g_2 .... g_{n-1} respectively such that $f_1 < f_2 <$.... f_{n-1} and $g_1 < g_2 <$.... g_{n-1}.

Assumption:

Let the first edge to differ in the ordering differ at i^{th} position. Without Loss of generality, we can assume that $f_i < g_i$. Upon adding f_i to T_2, a cycle is formed because adding an edge to tree results in a cycle.

Case-1:

Assume f_i is not the largest weight edge in the cycle formed. Then. if g_k is the edge with the largest weight. then removing g_k, we get a tree $T_2 \cup \{\mathrm{f}_i\} \setminus \{\mathrm{g}_k\}$. The resultant tree has smaller weight than T_2 which is not possible because T_2 is MST.

Case-2:

Assume f_i is the largest weighted edge in the cycle formed. So, cycle is formed by

$$f_{\mathrm{i}} \cup \{g_1, g_2, ...g_{\mathrm{i\text{-}1}}\}$$

which is same as

$$f_{\mathrm{i}} \cup \{f_1, f_2, ...f_{\mathrm{i\text{-}1}}\}$$

because $f_j = g_j \; \forall$ j $<$ i. This is not true because T_1 does not contain a cycle as T_1 is a tree.

Hence we arrive at a contradiction and our assumption is false. Thus, a graph with distinct edges has a unique MST.

Algorithm 1: MST of a tree with n vertices and n+8 edges

Result: Write here the result
V(G) = set of vertices $\{v_1, v_2 ... v_{n-1}\}$ in G;
T(G) = DFS Tree of v_i for any i;
foreach *remaining edge e(u,v)* $\notin T(G)$ **do**
    Let e_1, e_2, ... ,e_k *be the edges in the path from u to v in T(G);*
    m = max(e_1, e_2, ... ,e_k*);*
    if $|m| > |e|$ **then**
        T(G) = T(G) $\cup \{e\} \setminus \{m\}$*;*
    end
end
return T(G);

1.2 MST in O(n) time for n+8 edges

Lemma 1 *The size of tree decreases at every step.*

Proof: At every stage, we add an edge and a cycle is formed. From the cycle, we remove the edge with the maximum weight which can be the edge that we just added or some other edge. In any case, either the weight stays the same(when the edge that was inserted in removed) or decreases(when edge with more weight is removed).

Lemma 2 *The resultant tree obtained is MST.*

Proof: In the tree that is obtained in the end, the edges that are not the part of the tree are rejected by the algorithm. i.e. they are the largest edges in the cycle that they form. So, on addition of any edge, the edge itself is removed hence no tree can be formed that has a smaller weight than the current tree. Since for distinct edges, MST is unique(1.1), the given tree is the MST.

1.2.1 For getting path from u to v

While generating the tree from DFS, we can maintain parent-child relation link for each node.

By doing so, we can find path from any two node in O(m) time without applying DFS again and again. Given parent-child relation, we would have to only find only the common ancestors of the nodes. This can be achieved in O(m) time.

After swapping of each edges, we might have to update the parent- child relation of some nodes in the cycle. This is also possible with DFS in O(m) time.
Thus, we can find the path between any two vertices in O(m) time.

1.2.2 Proof of termination

Proof:

After the formation of tree, the algo will run for time equal to number of edges of graph not in the tree i.e

$$n + 8 - (n - 1) = 9$$

Thus, there are 9 edges in total that are inserted one by one to the tree and checked whether they satisfy the condition.
Hence, after the 9 edges, the algorithm terminates.

1.2.3 Time Complexity:

We are doing following tasks repeatedly:

1. Creating the DFS tree for any random vertex. This step takes O(m) time.
2. Then it loops for all the remaining 9 edges in the tree in the tree.
3. It runs the path finding logic to get the path from one end point of the edge to another in a tree and removes the maximum. This process takes O(m) time and is repeated 9 times for each edge.

Hence, the final time complexity is

$$O(m) + 9 * O(m) = 10 * O(m) = O(m) = O(n)$$

2 Huffman Encoding

2.1 Optimal Binary Encoding for fibonacci numbers

Optimal Huffman Encoding for the given example of 8 fibonacci numbers is:

h	0
g	1 0
f	1 1 0
e	1 1 1 0
d	1 1 1 1 0
c	1 1 1 1 1 0
b	1 1 1 1 1 1 0
a	1 1 1 1 1 1 1

Generalizing the above example, for n fibonacci numbers, the i^{th} fibonacci number has an encoding of $2^{n-i+1} - 2$ for i > 1 and $2^{n-1} - 1$ for i $= 1$.

Lemma 3

$$F_{n+2} = \sum_{n=0}^{n} F_n + 1$$

Proof: Proof by induction.

Assume a fibonacci series such that $F_0 = 0$ and $F_1 = 1$.

Base case: n = 0

F_2 = F_0 + 1 = 0 + 1 = 1

Induction hypothesis:

let $F_{i+2} = \sum_{n=0}^{i} F_j + 1$

For n = i+3:

$F_{i+3} = F_{i+1} + F_{i+2} = \sum_{n=0}^{i+1} F_j + 1$

Hence, the proposition is true for n = i+3 whenever we assume induction hypothesis.

Therefore, by principle of mathematical induction, given proposition is true for all n.

Hence proved by induction.

Lemma 4 *The number that is formed after adding the minimum frequency nodes takes second position because the sum is 1 less than the second fibonacci number.*

Proof: Proof by induction

Base case:

The first two terms, i.e. 0 and 1. The new node is 1 and it is less than the second term by 1.

Induction hypothesis:

Let the node formed by sum of first i elements of the series come at the second place.

For n = i+1:

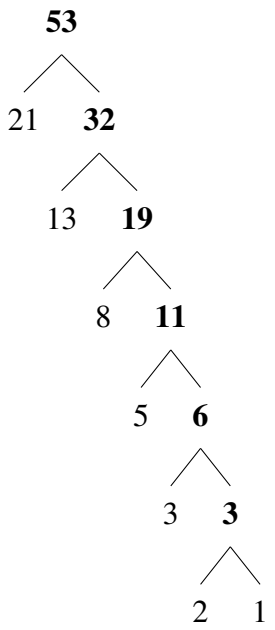
The series becomes:

$$F_{i+1}, \sum_{n=0}^{i} F_j, F_{i+2}, ...$$

On picking the minimum two elements, we get $\sum_{n=0}^{i+1} F_j$ which is $F_{j+3} - 1$ from the lemma proved above. Hence, one less than the second element. Hence, the proposition is true for n = i+1 whenever we assume induction hypothesis.

Therefore, by principle of mathematical induction, given proposition is true for all n.

Hence proved by induction.



2.2 Compressing a file with 16-bit characters

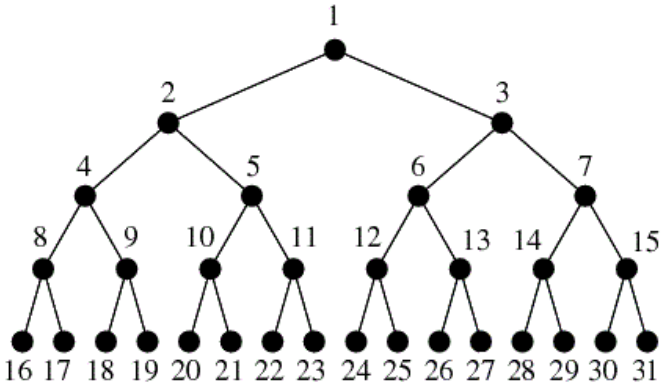
Since there are 16-bit characters, there are 2^{16} combinations of characters which need to be encoded. Using the algorithm discussed in class to find the huffman encoding of the characters, We add the minimum two frequencies and replace them with another node.
let the frequencies of these 2^16 characters is given in the ascending order as

$$F_1, F_2, F_3, ..., F_{2^{16}}$$

it is given that

$$2 * F_{min} > F_{max}$$

which implies that upon applying the algorithm to the above problem, the sum of the minimum two frequencies will result in a node with the maximum frequency. Thus, adding the nodes together 2^8 times, we get 2^8 pairs that are formed after adding these terms. We can continue this process and in the end, be left with only one node. From this node, we can open the tree, which will open in both the directions, to give us a balanced binary tree.



3 Graduation Party of Alice

3.1 Finding the largest number of people for the party

Algorithm 2: Find the largest subset of vertices of graph with n vertices such that deg(v) is greater than 5 and less than n-5.

```
Insert all the vertices in an AVL tree T comparing them based on their degrees.;
while (deg(smallest(T)) < 5 or deg(largest(T)) > n-5) and |V(G)| > 0 do
    if deg(smallest(T)) < 5 then
        u = smallest(T);
        remove u from T;
        update the degree of neighbours of v in T;
        remove u from G;
        update the degree of neighbours of v in G;
        n–;
    end
    if deg(largest(T)) > n-5 then
        u = largest(T);
        remove u from T;
        update the degree of neighbours of v in T;
        remove u from G;
        update the degree of neighbours of v in G;
        n–;
    end
end
return V(G);
```

3.1.1 Proof of correctness

Proof by induction

Let f(i) = Algorithm gives correct result if i vertices are to be removed for optimal answer.

Base Case: i=0

No vertex is to be removed. Thus $\forall$ v $\in$ V(G): deg(v) $\geq$ 5 and deg(v) $\leq$ n-5. In that case, the while loop condition will be false and the algorithm will return whole V(G) which should be the correct output for i = 0.

Induction hypothesis:

The algorithm gives correct output for all i = 0, 1, ... k.

For i = k + 1

Since k + 1 vertices are to be removed, the flow will go into while loop. Let v be the vertex removed. Now this removal may result in k components a_1, a_2, ... a_k. Now in all the

components, sum of all vertices to be removed is k. Thus, each connected component contains not more than k vertices to be removed.
Using induction hypothesis, the algorithm will run correctly on all the connected components.
Thus, algorithm gives correct output for n = i + 1 whenever induction hypothesis is satisfied.
Thus from principle of mathematical induction, algorithm gives correct output for all n.

Note: For final answer, take union of all answers i.e:

$$Ans(G) = Ans(a_1) \cup Ans(a_2) \cup ....Ans(a_k)$$

3.1.2 Proof of termination:

In every iteration, we are doing one of the following tasks:

1. Terminating the algorithm if none of the vertex has less than 5 or greater than n-5 degrees.
2. Terminating the algorithm if no vertex is left in the graph.
3. Removing at least one vertex from the graph.

Since graph will have finite vertices, the algorithm is bound to terminate.

3.1.3 Time Complexity:

We are doing following tasks repeatedly:

1. Initial generation of AVL tree will take O(nlogn) time to add all the vertices.
2. We are carrying out search for least and largest element in AVL tree to get the vertices with lowest and highest degree. This operation takes O(logn) time for each iteration.
3. Removing vertex from T also takes O(logn time). Removing vertex from graph takes O(deg(v)) time for each vertex v.
4. Updating the degrees will overall take O(sum of degrees*logn) time which is O(2*m*logn) = O(mlogn).
5. Updating the neighbours in graph will take O(sum of degree) time in total which is O(2*m) = O(m).

Hence, the final time complexity is:

$$O(nlogn) + O(mlogn) + O(m) = O((n + m)logn)$$

3.2 Table arrangement

3.2.1 Idea

Sort the ages using **bucket sort** in O(n_0) time. Assign people to tables. If number of people exceed 10 or the age difference between min and max exceeds 10, increment the number of tables by 1. Count the number of tiles.

3.2.2 Correctness of algorithm:

Proof:
Proof by contradiction.
Let the optimal solution to this problem be k_1 tables and the algorithm gave k_2 tables as result. If there are k_1 tables in optimal solution, then the people sitting on the tables must be such that age difference is less than 10 and there are no more that 10 people on the table.

If there are overall less tables involved, then there should exist a person p_1 such that was sitting on a different table in the algorithm's solution but he was sitting with someone in the optimal solution. If p_1 was sitting with someone in optimal solution, this means that the age difference and well as the number of people on that table, both conditions meet. If these conditions are satisfied, then our algorithm would have not assigned a separate table to p_1. Hence we arrive at a contradiction that p_1 cannot sit on some other position in optimum solution.
Thus, no such p_1 exists and the optimum solution and the algorithm's solution are same.

3.2.3 Time Complexity:

We are doing following tasks repeatedly:

1. Bucket sort takes O(n_0) time and O(100) space because it is given that the age varies in the range(10,99).

2. After sorting, the traversal of the array formed takes O(100) time.

Hence the total time complexity of the code is:

$$O(n_0) + O(100) = O(n_0)$$

Algorithm 3: Find the minimum number of tables to fulfil the needs of the guests

```
Result: Write here the result
array = [0 for i in range(100)];
bucketSort(ages, array);
tableCount = 0;
currentTable = 0;
i = 0;
for i in range(20, 99) do
    if array[i] != 0 then
        break;
    end
end
smallestAge = i;
while i < 100 do
    if array[i] == 0 then
        i++;
        continue;
    end
    if i - smallestAge > 10 then
        currentTable = 0;
        tableCount++;
        smallestAge = i;
    end
    currentTable = currentTable + array[i];
    while currentTable > 10 do
        tableCount++;
        currentTable -= 10;
        smallestAge = i;
    end
    i++;
end
if currentTable > 0 then
    tableCount++;
end
return tableCount;
```