COL351-Assignment1

Pratyush Saini 2019CS10444
Prakhar Aggarwal 2019CS50441

5th August 2021

1 Question 1

1.1 Part a

Proof by contradiction:
Assume for the sake of contraction that G contains two Minimum Spanning Trees T1 and T2. Enumerate the edges of T1 and T2 as follows:

$$E(T1) = \{\text{e1, e2, e3, ... , em}\}, \text{ where e1} < \text{e2} < \text{...} < \text{em}$$
$$E(T2) = \{\text{g1, g2, g3, ... , gm}\}, \text{ where g1} < \text{g2} < \text{...} < \text{gm}$$

Let g_k be the minimum edge in $T_2 \setminus T_1$ and Let e_k be the minimum edge in $T_1 \setminus T_2$.
Without loss of generality assume that wt(e_k) < wt(g_k). By the definition of MST, we can say that $T_2 \cup \{e_k\}$ contains a cycle C, which passes through edge e_k. Let u be any edge of cycle C that is not contained in T_1. At least one such edge must exist, because T_1 is a tree. (We may or may not have u = g_k). Because $e_k \in T_1$, we can say that u $\neq e_k$ and thus, u $\in T_2 \setminus T_1$. Now consider the spanning tree T = T_2 + $\{e_k\}$ - {u}. wt(T) = wt(T_2) + wt(e_k) - wt(u) $\leq$ wt(T_2). Now, since T_2 is a minimum spanning tree, we conclude that T is also a minimum spanning Tree. It automatically follows that wt(e_k) = wt(u), which contradicts our assumption that all the edges of G are unique.

1.2 Part b

Algorithm 1 Minimum Spanning Tree

2 Question 2

2.1 Part a

Fibonacci sequence follows the relation $F_{n+1} = F_n + F_{n-1}$
We consider the following algorithm for constructing Huffman tree from given Fibonacci Sequence.

- Create leaf nodes for each character in priority queue(min heap). The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root.
- Extract two nodes with the minimum frequency from the min heap.
- Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
- Repeat the above steps till only one node is left in the min heap.

Claim:

$$\sum_{i=1}^{n} a_k < a_{k+2} \tag{1}$$

Proof: Assume it is true for n, Add a_{n+1} to both the sides, we can show that it is satisfied for n+1.
Claim:

2.2 Part b

3 Question 3

3.1 Part a

3.1.1 Algorithm

We are given n people and a list of pairs of people who know each other. Mathematically, we can formulate the problem as:
Given: A graph G with n vertices and m edges, we need to find the largest subset S(V,E) of G satisfying the following contraints:

- $\forall u \in S(V), \exists$ at least 5 v $\in S(V)$ such that (u,v) $\in S(E)$
- $\exists$ at least 5 w $\in S(V)$ such that (u,w) $\notin S(E)$

Algorithm 2 Yet to be decided

```
procedure ADJACENCY_LIST(Edges, N)
    adjList ← [{}, {}, ... {}]                    ▷ list of unordered_set (length N)
    for i from 1 to size(Edges) do
        adjList[Edges[i][0]].insert(Edges[i][1])
        adjList[Edges[i][1]].insert(Edges[i][0])
    end for
    return adjList
end procedure
```

3.1.2 Running Time

Time Complexity of the algorithm = Time Complexity of the Adjacency_List procedure + Time Complexity of the While loop + Time Complexity of the for loop

- Time Complexity of the Adjacency_List procedure is O(length(Edges)) i.e. O(n^2) (considering input to be a dense graph)
- Time Complexity of the last for loop is O(n)
- While Loop
  - The while loop will run atmost n times, and, in each iteration, the first for loop runs O(n) times: n*O(n) -> O(n^2)
  - For every person added to the **NewlyRemoved** list, the algorithm runs a loop over the **adjList** list (size n) and removes the edge(if present) incident on the person in the **NewlyRemoved** list
  - The loop over the **adjList** helps to remove all such edges and it takes O(1) time (unordered_set) to remove 1 such edge
  - So, we can insert atmost n persons to the **NewlyRemoved** list, and, for each such person, we will loop over the **adjList** (size n), removal of an edge takes O(1) time
  - Therefore, Time Complexity of the while loop will be: O(n^2) + n*O(n)*O(1) -> O(n^2)
- Overall Time Complexity: O(n^2) + O(n) + O(n^2) -> O(n^2)

Algorithm 3 Yet to be decided

```
procedure PeopleToInvite(Edges, N)                                   ▷ People who know each other
    adjList ← ADJACENCY_LIST(Edges, N)
    RemPeople ← N
    GuestToRemove ← True
    PeopleList ← [1, 1, ....., 1]                                                    ▷ length N
    while GuestToRemove = True do
        GuestToRemove = False
        NewlyRemoved ← []                  ▷ Will store the list of people removed in 1 iteration
        for i from 1 to N do
            if PeopleList[i] = 1 and (size(adjList[i]) < 5 or size(adjList) > RemPeople - 5) then
                GuestToRemove ← True
                PeopleList[i] ← 0
                RemPeople ← RemPeople −1
                NewlyRemoved.insert(i)
            end if
        end for

        if GuestToRemove = True then
            for i from 1 to N do
                if PeopleList[i] = 0 then
                    continue
                end if
                for j from 1 to size(NewlyRemoved) do
                    if adjList[i].find([NewlyRemoved[j]]) != adjList[i].end() then
                        adjList[i].erase(NewlyRemoved[j])            ▷ O(1) (unordered_set)
                    end if
                end for
            end for
        end if

    end while

    GuestList ← []                                                                ▷ Empty List

    for i from 1 to N do
        if PeopleList[i] = 1 then
            GuestList.insert(i)
        end if
    end for

    return GuestList                                                           ▷ Party Invitees
end procedure
```

3.1.3 Proof of Correctness

At each iteration, we are removing atleast 1 guest from Alice's invitee list. As the number of People (N) is finite, this guarantess the **termination of algorithm**. We are removing only those people who do not satisfy the contraints mentioned above i.e. only those people who could not be invited to the party. This implies the above algorithm always produces the largest subset of people who can be invited to the party.
Proof (Vague For Now):
Let H(V,E) be the optimal answer. So, $\exists$ u $\in$ H(V) such that u $\notin$ S(V) (Produced by the algorithm). This implies, PeopleList[u] $\geqslant$ 5 and PeopleList[u] $\leqslant$ size(H) - 5. But according to our algorithm, we will never remove such vertex.

Hence our algorithm produces the largest subset of G subject to the contraints :)

3.2 Part b

Given: Given n people and their respective age, we need to find minimum number of tables to sit them such that:

- Each table has a capactity of 10 people
- Age difference between members of same table should be atmost 10

Algorithm 4 Yet to be decided

```
procedure MinTables(PersonAge, N)                          ▷ Replacing n_0 with N
    Tables ← 0
    MaxAge ← 99
    MinAge ← 10
    AgeGap ← 10
    TableCap ← 10
    RemPeople ← N
    AgeList ← [0, 0, ....., 0]                             ▷ length = MaxAge - MinAge + 1
    for i from 1 to N do
        AgeList[PersonAge[i] - MinAge] ← AgeList[PersonAge[i] - MinAge] + 1
    end for

    currAge ← MinAge

    while currAge ⩽ MaxAge and RemPeople ⩾ 0 do

        while currAge ⩽ MaxAge and AgeList[currAge] = 0 do
            currAge ← currAge + 1
        end while

        if currAge > MaxAge then
            return Tables
        end if

        startAge ← currAge
        lastAge ← currAge
        peopleCount ← 0

        while peopleCount ⩽ TableCap and lastAge - startAge ⩽ AgeGap do
            peopleOfThisAge ← min(TableCap - peopleCount, AgeList[lastAge - MinAge])
            AgeList[lastAge - MinAge] ← AgeList[lastAge - MinAge]) - peopleofThisAge
            peopleCount ← peopleCount + peopleOfThisAge
            if AgeList[lastAge - MinAge] = 0 then
                lastAge ← lastAge + 1
            end if
            if lastAge > MaxAge then
                break
            end if
        end while

        RemPeople ← RemPeople - peopleCount
        Tables ← Tables + 1
        currAge ← lastAge
    end while

    return Tables                                          ▷ Minimum Tables
end procedure
```