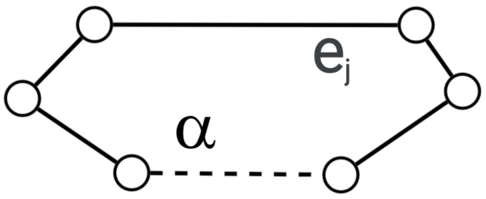
Aman Gupta
2019CS10673
Arpit Chauhan
2019CS10332

Assignment - 1
COL351 - Analysis and Design of Algorithms
Term: Fall 2021
Due Date: 5th September, 2021

Problem 1(a)
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. Prove that G has a unique MST.

Solution: Given a graph G(V,E) such that, all the edges e $\in$ E are unique. We need to prove that there exists a unique minimum spanning tree of G.
We will use **contradiction** to prove the claim.
Claim 1: Let us assume that there exists two different MSTs of G, namely, T_1(V,E_1) & T_2(V,E_2). Since, T_1 & T_2 are two spanning trees they have the same set of vertices but have different edge sets $E_1 \neq E_2$.
Consider a set J = $E_1 \cup E_2$,
let α be the least weight edge in J, which belongs to only one of T_1 & T_2. Since all edges are distinct and the two trees are different, there exists only one such edge α.
Without loss of generality let us assume that $\alpha \in T_1$ and hence $\alpha \notin T_2$.
Lets add the edge α to the tree T_2, Since trees are maximal connected adding an additional edge will form a cycle, therefore the new graph T' = $T_2 \cup \{\alpha\}$ contains a cycle C.
S = $\{e_1, e_2, e_3, ...e_i\}$ be the edges of the cycle C excluding α. Hence, S $\subseteq E_2$

Assertion: There exists an edge in S that has higher weight than α.
We can prove this by contradiction.
Let us assume that all edges in S have lesser weight than α.
By the way of choosing edge α we know that all the edges having weight less than α, belongs to both the tree T_1 & T_2. Since all the edges of S have weights less than α, therefore, S $\in E_1$.
But S $\cup \{\alpha\}$ forms the cycle C, and are also contained in the tree T_1. This leads to a contradiction and hence our initial claim that all edges in S have lesser weight than α was false.
Hence the assertion is proved.



Let $e_j \in$ S be the edge having higher weight than α.
Consider T" = $T_2 \setminus \{e_j\} \cup \{\alpha\}$,
T" is a spanning tree, because it does not contain cycle C as edge e_j is removed and is also connected as any path going through the edge e_j can be replaced by a path going through edge α.
Since T" has a less total weight than T_2 (weight(α) <weight(e_j)), therefore T_2 cannot be a MST of G. Therefore by contradiction, claim 1 is false and hence there cannot be two different MSTs of a graph, in which all the edges are distinct.

Problem 1(b)
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. 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 following is the pseudo code to find the MST of graph G, with n vertices and at most n+8 edges, in O(n) run time.

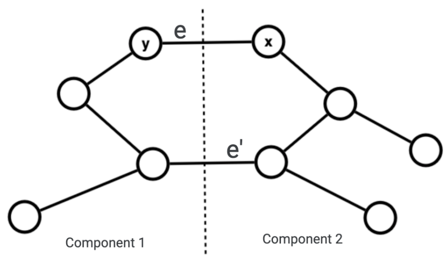
Algorithm 1: Finding MST of graph G

```
1 Perform BFS starting from any vertex of the graph G, to ensure that the graph is connected. If
    the number of vertices in the BFS tree is less than n, then the graph is not connected.
2 Perform DFS starting from any vertex of the graph G to detect a cycle, indicated by the presence
    of a back edge during traversal.
3 If a cycle is detected then find all the edges of that cycle using back edge and parents array.
    Remove the edge with highest weight form the graph G. Then repeat the step 2 & 3 again.
4 If no cycle is detected then terminate the process and return the remaining graph as it the
    required MST of G.
```

Proof of Correctness
We will prove the correctness of the algorithm to provide the MST by the following claims.

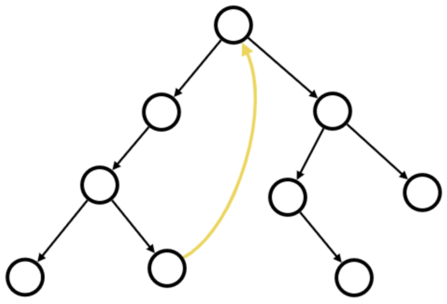
Claim1 Highest weight edge of a cycle cannot be part of any minimum spanning tree of G.
Proof We will prove the claim by contradiction.
Let us assume that there exists a MST T that contains edge e(x,y), which is the heaviest weight edge in cycle C.
If we remove edge e from the tree T, it will get disconnected into two components as a tree is minimal connected graph. Also there will be no path from node x to y, after removing edge e.
Since C is a cycle containing edge e, there must be another edge in C that has one vertex in each of the two components. Let this edge be e'.
Consider the graph T' = T \ {e} ∪ {e'},

Figure 1: Graph G



T' is a connected graph because all paths in T that were passing through e can now be replaced with paths passing through e'. T' also does not contain any cycle because after removing e we only added an edge e' that connects the two separated components.
Hence T' is a spanning tree of the graph G. Also, the total weight of T' is less than weight of T as e is the heaviest weight cycle in the cycle C so weight(e) > weight(e').
Hence, T cannot be a MST of G, our initial assumption was false, therefore the claim is true.

Figure 2: Using DFS tree to find cycle



https://www.overleaf.com/project/612f1440f6cc444f8b54ab1f The algorithm in each iteration finds a back edge to detect a cycle in the graph and removes the maximum edge weight of the cycle to proceed to a smaller graph. This process is continued till we find a graph with no cycle, which is the MST.

Run Time Analysis
The graph G contains m edges and n vertices.
Since according to the given condition in the problem. m $\leq$ n+8, Therefore m = O(n).
We know that time complexity of BFS or DFS on a graph G(V,E) is O($\|E\|+\|V\|$), in this case it will be O(m+n) = O(n) (As m is at most n+8).
The algorithm performs one BFS in the beginning to detect whether the graph is connected or not. Further the algorithm performs DFS m-(n-1) number of times. Because a tree with n vertices has n-1 edges. m-(n-1) $\leq$ 9, hence its performs a constant number of times.
Therefore, the overall complexity of the algorithm is **O(n)**.

Problem 2(a)
What is the optimal binary Huffman encoding for n letters whose frequencies are the first n Fibonacci numbers? (For example: The frequency vector F is [1, 1, 2, 3, 5, 8, 13, 21] for n = 8). What will be the encoding of the two letters with frequency 1, in the optimal binary Huffman encoding?

Solution: We are given the frequencies for n letters as F_i for $1 \leq i \leq n$ which satisfies the relation $F_k = F_{k-1} + F_{k-2}$ where $F_1 = 1$ and $F_2 = 1$.

Claim 1: For all positive integers k, $F_{k+2} = 1 + \sum_{i=1}^{k} F_i$
Proof: We will use induction on k to prove this claim. Let us check the base case, k = 1

$$1 + \sum_{i=1}^{1} F_i = 1 + F_1 = 1 + 1 = 2 = F_3$$

Hence, the base case is true. Now, let us assume that $F_{k+1} = 1 + \sum_{i=1}^{k-1} F_i$. We know that

$$\begin{aligned} F_{k+2} &= F_{k+1} + F_k \\ &= 1 + \sum_{i=1}^{k-1} F_i + F_k \\ &= 1 + \sum_{i=1}^{k} F_i \end{aligned} \tag{1}$$

Hence, the claim is proved by induction.

We will use Binary trees to represent Prefix Codes. For each letter, we start with an empty string and follow the path from the root to the leaf labeled x. Every time the path goes from a node to its left child, we append 0, otherwise 1. The resulting string of bits is taken as the encoding of x.

To obtain an optimal binary Huffman encoding, we greedily combine 2 nodes with the least frequencies (new node has frequency equal to their sum) and then recursively construct the prefix encoding. The algorithm terminates in n-1 steps because at each step the number of remaining nodes decreases by 1.

Claim 2: After kth recursive step during the Huffman's algorithm on these letters, the set of frequencies of remaining nodes will be $\{\sum_{i=1}^{k+1} F_i, F_{k+2}, F_{k+3}, \ldots F_n\}$.
Proof : The proof is by induction on k. The base case is trivially true because for k = 1 we get the initial frequency set $\{F_1, F_2, \ldots F_n\}$.
Now, let us assume that the after (k-1) recursive steps, the set of frequencies of remaining nodes is $\{\sum_{i=1}^{k} F_i, F_{k+1}, F_{k+2}, \ldots F_n\}$.
Since F_i follows $F_k = F_{k-1} + F_{k-2}$ and $F_i > 0$, $F_{k+1} \leq F_{k+2} \leq \ldots F_n$.
Using claim 1, $\sum_{i=1}^{k} F_i = F_{k+2} - 1 < F_{k+2}$
Hence, $\sum_{i=1}^{k} F_i$ and F_{k+1} are the least 2 frequencies, so they will be combined to form a new node with frequency ,

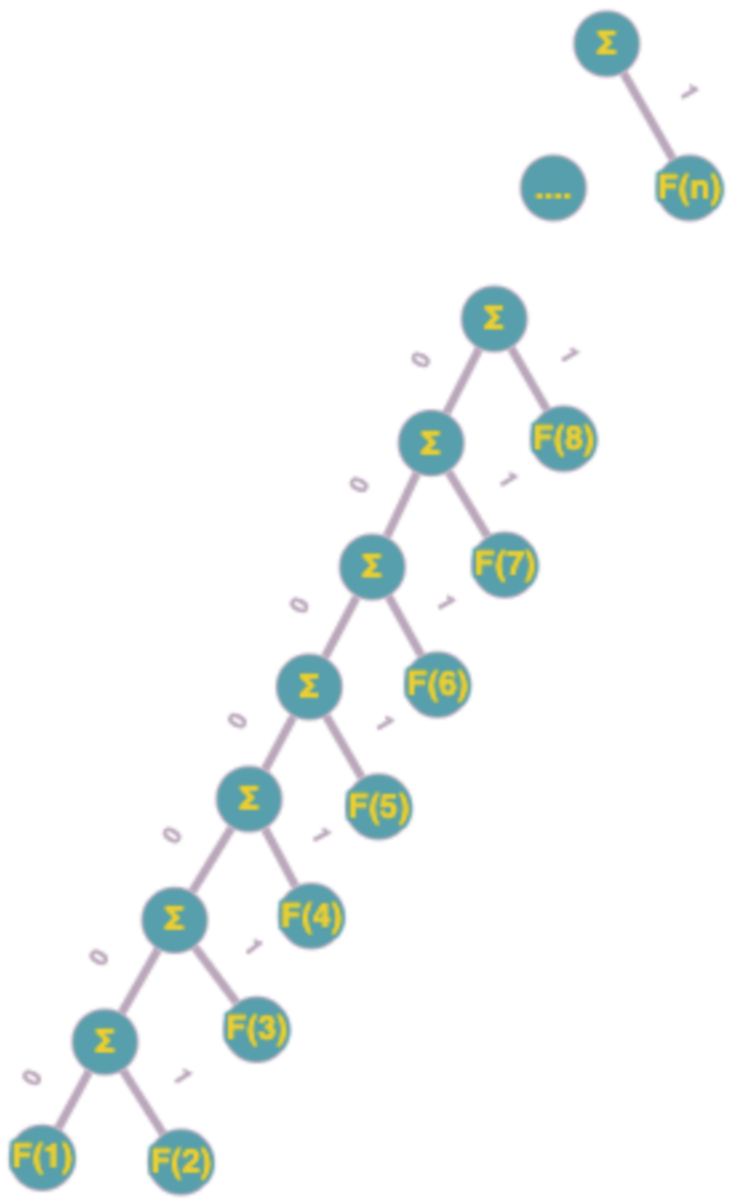
$$\sum_{i=1}^{k} F_i + F_{k+1} = \sum_{i=1}^{k+1} F_i$$

Thus, the new set of frequencies will be $\{\sum_{i=1}^{k+1} F_i, F_{k+2}, F_{k+3}, \ldots F_n\}$.
Hence, the claim is proved by induction.

Claim 2 suggests that one of the optimal solutions can be obtained from the following Binary tree where the k$^{\text{th}}$ (k>1) least frequent letter gets the encoding as $0^{n-k}1$ and the least frequent letter gets encoded as 0^{n-1}. Thus, the **two letters with frequency 1** get encoded as $0^{n-2}1$ and 0^{n-1}

Proof of correctness

At every recursive step, the newly formed node will be one among the two least frequency nodes using Claim 1 and Claim 2. Hence, a new level will be formed at each recursive step, thus obtaining the below binary tree. Hence, the Binary tree given below will yield the optimal binary Huffman encoding. This solution is not unique because at each level, the new node has the choice of being the left child or the right child which would change the encoding accordingly but the solution will remain optimal.



Binary tree that gives one of the optimal solution to the Encoding Problem

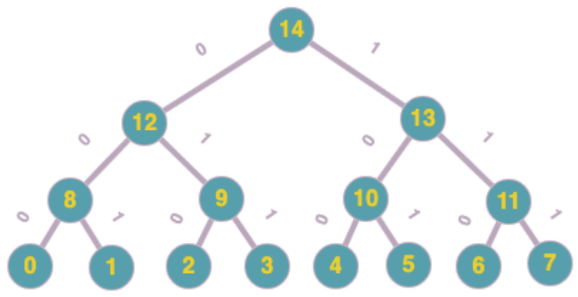
Problem 2(b)
Suppose you aim to compress a file with 16-bit characters such that the maximum character frequency is strictly less than twice the minimum character frequency. Prove that the compression obtained by Huffman encoding, in this case, is same as that of the ordinary fixed-length encoding.

Solution: Let the frequency vector be F $= \{F_1, F_2, F_3, \ldots F_n\}$ where $n = 2^{16}$. We are given that the maximum character frequency is strictly less than twice the minimum character frequency.
To obtain an optimal binary Huffman encoding, we greedily combine the 2 nodes with the least frequencies (new node has frequency equal to their sum) and then recursively construct the prefix encoding. The algorithm terminates in n-1 steps because at each step the number of remaining nodes decreases by 1.

Claim 1: If the maximum character frequency is strictly less than twice the minimum character frequency, then the sum of the least two frequencies is strictly greater than the maximum character frequency .
Proof: Let us say that any point during the algorithm, the frequency vector of the remaining nodes is $\{f_1, f_2, f_3, \ldots f_m\}$ in ascending order i.e. $f_1 \leq f_2 \leq \ldots f_m$.
$f_1 \leq f_2 \implies 2\text{f}_1 \leq f_1 + f_2$ (Adding f_1 on both sides)
But we are given that $f_m < 2f_1$, hence $f_m < f_1 + f_2$ and our claim is proved.

Claim 2: After each recursive step, the invariant that the maximum character frequency is strictly less than twice the minimum character frequency holds true.
Proof: We will prove this by induction. Base case is true because we are given that the claim is true initially. Let us assume that after k recursive steps, the frequency vector of the remaining nodes is $\{f_1, f_2, f_3, \ldots f_m\}$ in ascending order and Claim 2 holds true. Now as per the algorithm, we delete the nodes with frequency f_1 and f_2, replacing them with a new node of frequency $f_1 + f_2$. Claim 1 suggests that $f_1 + f_2 > f_m$, hence the new frequency vector will be $\{f_3, f_4, f_5, \ldots f_m, f_1 + f_2\}$ in ascending order.
Since $f_3 > f_1$ and $f_3 > f_2$, we get $2f_3 > f_1 + f_2$, thus the claim holds true after k+1 recursive steps as well. Hence, the claim is proved by induction.

Let us perform the algorithm in 16 stages. In the k^{th} stage, we perform 2^{16-k} recursive steps. We can verify that the total number of steps $= 2^{15} + 2^{14} + \cdots + 2^0 = 2^{16} - 1 = n - 1$ which is the exact number of steps required to build the binary tree that gives us the optimal Huffman encoding. In the 1^{st} stage, we start with 2^{16} nodes. After performing 2^{15} recursive steps, we obtain 2^{15} trees, each with 2 leaf nodes. The 2^{15} internal nodes have no frequency which is more than twice of the minimum frequency by Claim 2, hence we arrive at a similar situation as the initial one but with half the nodes. Hence in 16 stages, a Full binary tree of height 16 will be formed which yields the encoding same as ordinary fixed length encoding.



Full Binary Tree which gives the Huffman Encoding for 3-bit characters with same conditions

Claim 3: At the k$^{\text{th}}$ stage, $2^{16-(k-1)}$ nodes combine among themselves in 2^{16-k} recursive steps to form 2^{16-k} nodes at a new level with one less depth.
Proof: We will prove this by induction.Consider the base case with k=1. Claim 1 suggests that any new node formed during the stage will have frequency larger than all the initial nodes hence the 2^{16} nodes will combine among themselves to form 2^{15} nodes at a lower depth. Hence the base case is true.
Let the claim be true for k$^{\text{th}}$ stage. Claim 2 suggests that the invariant that the maximum character frequency is strictly less than twice the minimum character frequency still holds true. Hence, using claim 1, we can say that at the k+1$^{\text{th}}$ stage, the 2^{16-k} nodes will combine among themselves to form $2^{16-(k+1)}$ nodes at one less depth while performing Huffman's algorithm .Hence, the claim is true for k+1$^{\text{th}}$ stage as well. Thus the claim is proved by induction.

Correctness
By claim 3 since the number of nodes in the min heap gets halved at each stage, the algorithm will terminate in 16 stages with 2^d nodes at the d depth thus forming a full binary tree.
Hence proved that the compression obtained by Huffman encoding, in this case, is same as that of the ordinary fixed-length encoding.

Problem 3(a)
Alice wants to throw a graduation party and is deciding whom to call. She has n people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick a **largest** subset of n people, subject to two constraints: at the party, each person should have at least five other people whom they know and five other people whom they don't know. Present an efficient algorithm that takes as input the list of n people along with the list of pairs who know each other and outputs the best choice of party invitees. Give the running time in terms of n.

Solution: We can mathematically represent this question as a graph problem. With the following analogy:
Alice's friends represents nodes of Graph
Pairs that know each other represent edges of the Graph
Let G(V,E) be our graph, then the question requires us to find a maximal sub-graph, in which the minimum degree of a node is five and maximum degree is number of nodes in that sub-graph minus five.

Algorithm
1. Find degree of all the vertices of the graph.
2. Let the number of vertices in the graph be n. Remove all the vertices (and edges connected to them) which have degree less than 5 and the ones that have degree greater than n-5. If no vertex is removed algorithm terminates and we have arrived at our solution.
3. Recursively, solve the problem for the smaller graph.

Claim 1: At any recursive step, vertices having degree less than five cannot be a part of the solution
Proof: Vertices having degree less than five cannot be a part of the optimal graph because the problem asks that each vertex should have at least five neighbours (each person should have at least five other people whom they know) and number of neighbours cannot increase in further steps.

Claim 2: At any recursive step, vertices having degree more than number of vertices minus five cannot be a part of the solution
Proof: Vertices having a degree more than the number of vertices minus five cannot be a part of the optimal graph because the problem asks that each vertex should not be directly connected to at least five other vertices (each person should have at least five other people whom they don't know). Since the number of vertices decreases in further steps, the number of vertices to which a particular vertex is not directly connected to cannot increase.

Proof of termination
The algorithm removes at least one vertex in each step because if it does not then is has already been terminated. Therefore the total number of steps cannot be more than the number of vertices in the initial graph, that is, n. Since n is a finite number the algorithm will terminate in finite time.

Proof of Correctness
The algorithm will terminate only when the two required constraints are satisfied. Since we have given the proof of termination, the output of the algorithm will follow the given constraints.
In our algorithm, we remove only those vertices which can never be a part of the solution, by Claim 1 and Claim 2. Hence, the subset of people obtained at the termination of the algorithm will be the largest possible subset subject to two constraints: at the party, each person should have at least five other people whom they know and five other people whom they don't know.

Let the sub-graph obtained after the algorithm terminates be M and opt(G) denotes the optimal sub-graph of graph G.
Claim 3: M = opt(G)
The claim can be proved using the following claims:

M $\subseteq$ opt(G)........(i)
M is a sub-graph that satisfies the conditions that each vertex has degree at least five and at most $\|V\|$-5. opt(G) is maximal sub-graph that satisfies this condition, so by the condition of maximality claim (i) holds true.

opt(G) $\subseteq$ M ....(ii)
Using Claim 1 & 2, we can say that the vertices removed from G to form M, cannot be part of the optimal solution. Hence, the optimal solution of G is contained in M, so claim (ii) holds true.
Using the claims (i) & (ii) we can say that M is equal to opt(G).

Thus, the solution obtained by our algorithm is correct.

Implementation and Running Time of Algorithm

Let us represent the initial graph with **n** vertices and **m** edges using adjacency lists. First, we calculate the degree of each vertex which will just be the size of the corresponding adjacency lists. Building the graph takes O(m) time and then calculating the degrees takes O(n) time.
Then, in each recursive step, we make a linear pass through the vertices and remove the vertices with degree less than **5** or greater than $|V| - 5$ where V is the set of vertices present during that recursive step. When we remove a vertex and the edges incident on it, we decrease the degrees of the vertices in its adjacency list by 1, thus updating the degrees of all the remaining vertices. Since the initial number of edges is m, the number of steps taken for updating the degrees across all recursive steps is bound by 2m. Hence, this step takes overall O(m) time.
The number of recursive steps is bound by n because in each iteration we remove atleast 1 vertex and in each recursive step, we perform a linear search over the remaining vertices, hence the overall time complexity of this operation is bound by O(n^2).
Since m is bound by n^2, the overall time complexity of the algorithm comes out to be $\mathbf{O(n^2)}$.

Problem 3(b)
Suppose finally Alice invited n_0 out of her n friends to the party. Her next task is to set a minimum number of dinner tables for her friends under the constraint that each table has a capacity of ten people and the age difference between members of each dining group should be at most ten years. Present a greedy algorithm to solve this problem in $O(n_0)$ time assuming the age of each person is an integer in the range [10, 99].

Solution:

Sketch of Algorithm

We maintain a map that stores the number of individuals at a particular age. Then we iterate through ages from 10 to 99 to delegate tables with a greedy strategy. At each number where we find some people, we allocate tables to accommodate all the individuals at that age. If we don't have space in the previous table or if the age bandwidth has reached 10, then we start a new table from the point of requirement. Continuing this process till the end we find the minimum number of tables required.

Algorithm 2: Pseudo code to find the minimum number of tables required by Alice.

```
1 Make a Hash Map D, containing frequency of each age.
2 Maintain a variable S, which indicates that start of the current table. Initialise it to -1.
3 Iterate through the ages of 10 to 99
4     On each iteration, use D to check if people are present at that age. If yes, fill the current
          table, if there is space left. Else, start a new table at that age and repeat the process.
      Add 1 to the table count on starting a new table.
5     If at any iteration, the span of ages of current table exceeds 10, close that table.
6 Return the table count after the loop ends.
```

Time & Space Complexity

The algorithm performs the following operations:
1) Making the age-frequency Hash Map D takes $O(n_0)$ time as the number of guests is n_0 and O(1) space as hash map can be of at max 90 (99-10+1) size.
2) Iterations through all ages from 10 to 99 takes O(1) time because there are only 90 iteration steps and in each step constant amount of steps are performed. Space required is O(1) as we just need to maintain a few variables.
Hence, the overall algorithm runs in **$O(n_0)$ time** & **O(1) space**

Proof of Correctness

Let T be the table made by the algorithm for the lowest age person. Hence T either contains the ten least aged persons or the difference between the least and maximum age of T is ten.

Claim 1: There exists an optimal solution which contains table T.
Proof: Let Q = $\{T_1, T_2, T_3, \ldots\ldots T_k\}$ be an optimal solution. Let us assume that least age in the group be x and T_i be the table in the optimal distribution Q containing age x. The distribution of ages can be of the following types:

Case-1: There are less than ten people in the age group [x,x+10].
In this case the table T made by algorithm includes all the people in age group [x,x+10]. T_i can only either contain some or at max all the people of this age group. If T' does not contain some persons of that age group, we can take those persons out from their respective tables in Q and add them to T'. Therefore, Q $\setminus T_i \cup$ T is an optimal solution which contains T.

Case-2: There are more than ten people in the age group [x,x+10]
T contains the least ten age people of the age group. Let us assume that T_i contains only some of these least ten people.
Let T $\setminus$ T_i = $\{a_1, a_2, a_3...a_k\}$ (Such that $a_1 \leq a_2 \leq a_3$...)
and T_i $\setminus$ T = $\{b_1, b_2, b_3...b_k\}$ (Such that $b_1 \leq b_2 \leq b_3$...)
Since, T contains the least ten elements of this range therefore $a_1 \leq a_2 \leq a_3... \leq b_1 \leq b_2 \leq b_3...$
We can now safely exchange tables of each of a_i with b_i in the optimal solution Q because $a_i \leq b_i$ so anything contained in ten age span beyond a_i is also present in ten age span beyond b_i. In this way we will obtain an optimal solution Q' that contains table T.

Hence, we have proved the claim that there exists an optimal solution of the problem that contains the table T made by the algorithm.

Let J be the set of all the people's ages that are invited to Alice's party and opt(J) denoted the minimum number of tables required to arrange them. Consider reduced set of people J' = J $\setminus$ T, obtained after removing first table (which includes minimum age person) formed by the algorithm.

Claim 2: opt(J) = opt(J') + 1.
Proof: We will prove the claim by the following exchange arguments:
opt(J) $\leq$ opt(J') + 1 ....(i)
Since opt(J') is the optimal solution of the set J' and adding the table table T to it makes one of the solution of set J (J = J' $\cup$ T). By condition of optimality we can say that opt(J) is less than or equal to other solutions of J, therefore the condition (i) holds.
opt(J) $\geq$ opt(J') + 1 ....(ii)
Alternatively, opt(J') $\leq$ opt(J) - 1. Using Claim 1 we can say that there exists and optimal solution of J that contains the table T. Removing T from that optimal solution we obtain a solution for the set J'. Again, using the condition of optimality we can say that opt(J') is less than or equal to other solutions of J', therefor the condition (ii) holds.

Using (i) & (ii) we can say that claim 2 is true.

Hence, using Claim1 & Claim2 we can say that the the greedy strategy chooses appropriate table at each step and keeps reducing the set of ages of people until it reaches the optimal solution.