**Question 5 Edible Sequence**

**Divyansh Mathur,210357**

**To check whether a sequence of given tree structure is a valid BFS or not.**

We know when we start BFS, we maintain a queue data structure and for a particular iteration we add all the child into the queue once done we pop the element.

Using the above criterion we can devise the following algorithm.

*Algorithm:*

Maintain an array no\_of\_child[n] such that no\_of\_child[i] has the no. of children of index i.

Maintain another array parent[n] such that parent[i] has the parent of index i.

And Let the given sequence be arr[n].

Maintain a queue Q and apply the following operations:

for a element in the sequence, if its parent is at the front of Q then ok, decrease no\_of\_child[front] and push the element into the Q, now if the no\_of\_child[front] is zero then pop the front.

If the parent is not at front then simply print not edible and return

At the end of all iterations if we reach end of loop print edible.

*Pseudocode:*

*Let the tree is given in form of adjacency list.*

*//no\_of\_child array*

*for i from 0 to n-1*

*no\_of\_child[i]=adj[i].size()-1; //-1 becuase one neighbour is parent*

*//parent array { run a dry BFS}*

*intitalise empty queue P and panrent[n] array as -1 (={-1}).*

*parent [0]=0;*

*while(q is not empty){*

*s=q.size();*

*node= front(q);*

*pop(q);*

*for i from 0 to s{*

*for(all elements 'ent' of adj[node]){*

*if(parent[ent]==-1) {*

*parent[ent]=node;*

*}*

*}*

*}*

*}*

*//final algo*

*Q.push(0)*

*if(arr[0]!=0) {*

*print "not edible"*

*return*

*}*

*for i from 1 to n-1{*

*if(parent[arr[i]]!=Q.front){*

*print "not edible"*

*return*

*}else{*

*Q.push(arr[i])*

*no\_of\_child[Q.front]--;*

*if(no\_of\_child[Q.front]==0) {*

*Q.pop*

*}*

*}*

*}*

*print "edible"*

*Time Complexity Analysis:*

Filling the array no\_of\_child takes O(n) time

Filling the parent array needs BFS hence O(V+E), given the no. of vertices is n and given the graph is tree E=n-1 hence O(2n-1)

Again we run a BFS-type-algorithm where we visit each node viewing each edge hence again O(2n-1)

Hence overall time complexity =O(n)