Name - Sachin Singh
Roll NO - MT2022094

Q⇒1 Validate Binary Search tree.

Approach :> we need to iterate Left and Right Subtree simultaneously and need to check whether the tree following BST Conditions.

-> If we go Left of the tree recursively then we need to check "root->left->data" should be in the range of (mini, root->data.

-> Similarly on going Right we need to check "root->right->data should be in the range of (root->data, maxi);

-> If all recursive calls goes affirmative then Return 1 else return -1;

Code :>
```
bool Solve(Node *Root, int &mini, int &maxi)
{
    if(root==NULL) return true;
    if(root->data >= maxi || root->data <= mini)
        return false;
    return Solve(root->left, mini, root->data) && Solve(root->right->data, maxi);
}
bool isBST(Node *root) {
    int mini = INT_MIN;
    int maxi = INT_MAX;
    return Solve(root, mini, maxi);
}
```

TC : O(N)

SC : O(Height of BST)

Name : Sachin Singh
RollNo: MT2022004
TAKE : Home – Part - 2

Q⇒2) Subarray with elements greater than varying threshold.

1) Create two array for next smaller & pictures smaller for every element of nums.
2) Finding the length of every element of nums from it's previous smaller to next smaller.
3) Divide the threshold by length.
4) If the length is greater than quotient return the length.
5) else return -1.

```
int Solve (int nums[], int th)
{
  int n = nums.length;
  int next_small = new int [n];
  int Prev_small = new int [n];
  Stack<Integer> st = new Stack<>();
  st.Push(0);
  for (i=1; i<n; i++)
  {
  while (!st.isempty() && nums[st.peek()] >= nums[i])
  st.pop();
  if (st.size() != 0)
```

```
        Prev.Small[i] = St.Peek();
        St.Push(i);
    }
    St = new stack<>();
    St.Push(n-1);
    for(int i=n-2; i>=0; i--)
    {
        while(!St.isEmpty() && nums[St.Peek()] >= nums[i])
            St.POP();
        if(St.Size() != 0)
            next.Small[i] = St.Peek();
        St.Push(i);
    }
    for(i=0; i<n; i++)
    {
        int len = next.Small[i] - Prev.Small[i] - 1;
        if((double)len < nums[i])
            return len;
    }
    return -1;
}
```

TC : O(N)

SC : O(N)

(35) Oliver and the Game

⇒ This can be solved by Euler's Tour.

i) if x and y are not in the same subtree then directly return 'NO', as it cannot be traversed.

ii) If bob is going towards the king's mansion check if bob is in the subtree of oliver return Yes. (type = 0).

iii) If bob is going away from the king's mansion checks if oliver is in the subtree of bob return Yes else 'No' (type 2)

```
int te[10001], td[10001];
main ()
{
  // Construct the graph with (n+1) nodes ;
     dfs ( 1, graph) ;
  // Now check for the above conditions ;
  if ( ! subtree (x,y) && ! subtree (y,x) )
        return ('NO') ;
  if ( type == 0)
{     if (subtree (y,x) )
          return YES ;
      else
          return NO ;
}
```

```
    else if (type == 1)
  {
    if (subtree (x,y))
        return YES;
    else
        return NO;
  }

}
    int times = 1;
  dfs(int src, graph)
{
  inTime[src] = timer++;
  for (int x : g[src])
    {
      if (!inTime[g[x][i]])
        {
          dfs(g[x][i])
          times++;
        }
    }
    outTime[src] = timer++;
}

    subtree (int x, int y)
    {
      if (inTime[x] > inTime[y] && outTime[x] < outTime[y])
          return true;
      return false;
    }
```

Q⇒5) Course - Schedule - II

⇒ If we observe, we can see that this problem can be solved by Topological sort.

Suppose we consider a directed edge between a course's we consider a prerequisite and the course, we see that if the ordering of courses can be done than there is no cyclic dependency forming.

```
int [] find (int n, int a[][] )
{
  // create the graph
  List < List <integer>> g = new AL <> ();
  For(i=0; i<n ; i++)
      g.add (new ArrayList <>() );
  int edges = a. length ;
  for (int i = 0 ; i <edges ; i++)
  {
    int v = a[i][0];
    int u = a[i][1];
    g . get(u).add(v);
  }
// Calculate the indegree for each vertex.
  for (i=0 ; i<n ; i++)
  {
      for (int it : g.get(i) )
          indegree[it] ++;
  }
// Apply Topological sort.
```

```
Queue<Integer> q ;
for (int i=0 ; i<n ; i++)
 { if (indegree[i] ==0)
      q.add(i) ;
 }
List <Integer> topo ;
while (!q.isEmpty())
{  int node = q.Peek();
    q.remove();
     topo.add (node);
   for (int it : g.get(node))
   {  indegree[it]--;
    if (indegree[it]==0) q.add(it);
   }
}
if (topo.size() == n) return true ;
 return false ;
}
```

TC : O(V+E)

SC : O(V)

Page No.
Date

Q⇒65 UVA 722 LAXBS

→ Approach : Just start on the given cell in the Problem description & sum the number of connected water files in the four possible directions. This can be done by DFS

Pseudo Code – dfs(a,b); int x[4]={0,0,-1,1}
int y[4]={1,-1,0,0}

```
dfs(int i, int j)
{
if(i<0 || i>=n || j<0 || j>=m || vis[i][j] ||
   graph[i][j] == '1')
   return 0;
vis[i][j] = true;
int ans = 1;
for(int k=0; k<4; k++)
{
   ans += dfs(i+x[k], j+x[k]);
return ans;
}
```

TC : O(V+E)
SC : O(V), recursion stack space.

Page No.
Date

Q⇒80 UVA 10199 – Tourist Guide

Approach –: We have to find the articulation point of the given graph

Pseudo-Code –: Do DFS traversal of the given Graph,

i) In DFS, maintain a parent array for each vertex.

ii) To check if vertex V is the root of the DFS Tree & it has two children. For every vertex, count children. If the currently visited vertex V is root (parent[v] != NULL) and has more than 2 childs, print it.

iii) If U is not the root of the DFS tree & it has a child V such that no vertex in the subtree rooted with V has a back edge to one of the ancestors in DFS tree of U maintain an array disc[] to store the discovery time of vertices.

iv) For every vertex U, Find the earliest vertex that can be reached from the subtree rooted with U. So we maintain low[], low[v] = min(disc[u], disc[w]), where w is an ancestor of V & there is a back edge from some descendant of v to u.

TC : O(V+E) For DFS

SC : O(V+E) for visited array's

Page No.
Date

(95) UVA 10397 – Connect the Campus.

Approach :- It's a minimum spanning tree question we first set the distance b/w the nodes of our connected cable to 0 & then use the kruskal's algorithm for good.

Pseudo Code :

```
F := ∅
for each v ∈ G.V do
    MAKE-SET(v)
for each (u,v) in G.E ordered by weight
(u,v), increasing do
    if FIND-SET(u) ≠ FIND-SET(v) then
        F := F ∪ {(u,v)}
        UNION(FIND-SET(u), FIND-SET(v))
return F
```

TC : $O(E \log V)$

SC : $O(V+E)$

(110) UVA 11492 Babel

Approach : Find the shortest path from each of the initial language words to each of the target language words and take the one with minimum cost.

Instead of doing path from each initial to each target, start with smallest words in each language as initial and target. Change initial & target based on lowest Yes in cost. First found in the min. path This can be done by Dijikstra's Algo.

Pseudo-Code :

```
dijkstra(G, S)
For each vertex V in G
    distance[v] ← infinite
    Prev[v] ← null
  if V != S, add V to Priority Queue Q.

distance[S] ← 0
while Q is not empty
    U ← extract min from Q
  for each unvisited neighbour V of U
    temp ← dis[u] + edge(u, v)
  if temp < distance[v]
        distance[v] ← temp
        Prev[v] ← u
  return dist[v];
```

TC : O(Elog V) , SC : O(V)