ECSE215L: DATA STRUCTURES USING C++ CODATHON - III	
Date: **27-Nov-2021 8:30 AM**	No. of Questions: **1**
Proposed Time: **120 Minutes**	Skills Tested: **Binary Search trees Insertion, Graphs Representation and Traversal**

FRIEND RECOMMENDER SYSTEM

Students are given a project of friend recommender system, in which they need to recommend the most appropriate friendship match to the user. The students collectively suggested the following approach to be followed for the development of an efficient and effective friend recommender system:

Initial System: The system in its initial stage contains N number of registered users, each of certain age. Each user is represented in the system by his age A. [Assume all N registered users have distinct/ unique age].

Data Structure Creation: In order to give a random start to the system and for keeping the same age group people closer and clustered, binary search tree is suggested to be implemented. For this, pick the incoming user age and insert it in the binary search tree according to his/her age. This process is repeated until no more users arrive and one finally gets a binary search tree with N nodes and their linkages [according to the binary search tree properties].

Now, as tree is a graph, so, treat the above obtained binary search tree as an undirected and unweighted graph, where each undirected link/edge present between two users represent that they are friends and directly connected.

Generating Recommendations: With the help of above obtained minimally connected graph, the recommendations are generated as follows:

For a user of 'A' age:

- Make a list of all friends of friends of A [First, second, third and so on… indirect connections], those users who are not directly connected to him and hence may be recommended as friends.
- Then calculate the **absolute age difference** between the user's age 'A' and the age of each of these indirect users of the list.
- Then **arrange** the list of indirect linked users in **non-decreasing order based on their age difference**, and then their actual age [if necessary] i.e., in case of a tie, a younger user is preferred.
- The first user in this arranged, indirectly linked users list is recommended as the most appropriate friend. Along with the recommended user age, the number of the indirect connection is also showed to the user as an explanation. For example:
    - If recommended user is A->Friend->Friend: then number of the indirect connection is 1.
    - If recommended user is A->Friend->Friend->Friend: then number of the indirect connection is 2.
    - If recommended user is A->Friend->Friend->Friend->Friend: then number of the indirect connection is 3.
    - And So On..

You need to implement the above code, along with the following inputs.

Input Constraints:
5 <= A <= 100
5 <= N <= 20

Sample Input:
First Line: User for whom you need to generate the recommendation
Second Line: Number of N registered users in sequence
Next N Lines: Sequence of N registered users
Sample Output:
First Line: Age of the most appropriately recommended indirectly-connected user
Second Line: Number of the indirect connection

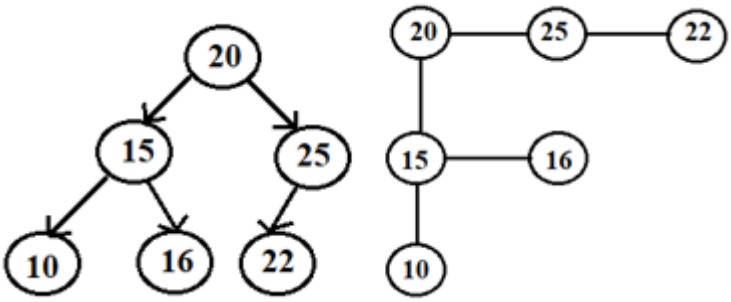
Sample Test Case:

Input:
20 --> User for whom you need to generate the recommendation
6 --> N = 6; Number of N registered users in sequence [Queue]
20 --> Queue Random Sequence of 6 registered users [20, 15, 25, 10, 16, 22]
15
25
10
16
22

Output:
22 --> Age of the most appropriately recommended indirectly-connected user
1 --> Number of the indirect connection

Explanation:

Sequence Generation [Already Given as Input]: [20, 15, 25, 10, 16, 22]
Data Structure Creation: Created Binary Search Tree from the above sequence and corresponding unweighted-undirected Graph:



Generating Recommendations: For User of age 20:

User Age	Status
10	
16	
15	Friend
22	
25	Friend

------>

User Age (X)	Age Diff. abs(X- 20)
10	10
16	4
22	2

ARRANGE ------>

User Age (X)	Age Diff. abs(X- 20)
22	2
16	4
10	10

User 22 is recommended. And as he is 20 --> Friend (25) --> Friend (22), So, number of the indirect connection is 1, i.e., 1st indirect connection. Hence the Output.

Test Cases:

1	20 6 20 15 25 10 16 22	22 1
2	30 5 10 20 30 40 50	10 1
3	20 7 70 60 50 40 30 20 10	40 1
4	100 10 100 90 80 70 60 50 40 30 20 10	80 1
5	10 10 100 90 80 70 60 50 40 30 20 10	30 1
6	45 12 50 35 45 41 40 44 47 46 30 55 51 80	44 1
7	55 8 50 35 45 40 30 55 51 80	45 2
8	16 8 16 20 25 22 19 18 21 17	17 3
9	21 10 29 20 21 15 22 16 27 28 14 30	16 2
10	100 12 100 90 30 5 35 15 40 20 45 25 50 55	55 6
11	40 20 40 29 41 25 33 60 20 31 36 55 15 30 32 35 38 54 34 37 53 42	38 3
12	41 20 40 29 41 25 33 60 20 31 36 55 15 30 32 35 38 54 34 37 53 42	42 4

Code:

```
#include <bits/stdc++.h>
using namespace std;
struct node{
   int data;
   struct node* left;
   struct node* right;
   struct node* parent;
};
typedef struct node node;
struct node* Add(struct node* root, int n){
   if(root == NULL){
      root = (node*)malloc(sizeof(node));
      root->data = n;
      root->left = NULL;
      root->right = NULL;
      root->parent = NULL;
   }
   else{
      node* temp = root;
      while(1){
         if(n<temp->data){
            if(temp->left!=NULL)    temp = temp->left;
            else{
               node* temp1 = (node*)malloc(sizeof(node));
               temp1->data = n;
               temp1->left = NULL;
               temp1->right = NULL;
               temp->left = temp1;
               temp1->parent = temp;
               break;
            }
         }
         else{
            if(temp->right!=NULL)   temp = temp->right;
            else{
               node* temp1 = (node*)malloc(sizeof(node));
               temp1->data = n;
               temp1->left = NULL;
               temp1->right = NULL;
               temp->right = temp1;
               temp1->parent = temp;
               break;
            }
         }
      }
   }
   return root;
}
vector<int> friends[100];
void bfs(node* root, int n ){
   queue<pair<node*,int>> q;
   bool visit[1000];
   for(int i=0;i<1000;i++) visit[i] = false;
```

```
    visit[root->data] = true;
    if(root->parent!=NULL){
        q.push({root->parent,0});
        visit[root->parent->data] = true;
    }
    if(root->left!=NULL){
        visit[root->left->data] = true;
        q.push({root->left,0});
    }
    if(root->right!=NULL){
        q.push({root->right,0});
        visit[root->right->data] = true;
    }
    while(!q.empty()){
        pair<node*,int> temp = q.front();
        q.pop();
        visit[temp.first->data] = true;
        friends[temp.second].push_back(temp.first->data);
        if(temp.first->parent!=NULL && !visit[temp.first->parent->data]){
            q.push({temp.first->parent,temp.second+1});
        }
        if(temp.first->left!=NULL && !visit[temp.first->left->data]){
            q.push({temp.first->left,temp.second+1});
        }
        if(temp.first->right!=NULL && !visit[temp.first->right->data]){
            q.push({temp.first->right,temp.second+1});
        }
    }
}
node* src = NULL;
void find(node* root, int n){
    if(n == root->data)
        src = root;
    else{
        if(n<root->data)
            find(root->left,n);
        else {
            find(root->right,n);
        }
    }
}
void print(node* root){
    if(root== NULL)
        return;
    else{
        print(root->left);
        cout << root->data << " ";
        print(root->right);
    }
}
int main() {
    struct node* root = NULL;
    int a;  cin >> a;
    int n;  cin >> n;
    int minD = INT_MAX;
```

```
    int minNum, distance;
    for(int i=0;i<n;i++){
        int x;  cin >> x;
        root = Add(root,x);
    }
    find(root,a);
    bfs(src,minNum);
    for(int i=1;i<100;i++){
        int k = friends[i].size();
        for(int j=0;j<k;j++){
            int val = abs(friends[i][j]-a);
            if(val<minD){
                minD = val;
                minNum = friends[i][j];
                distance = i;
            }
            else if(val == minD && friends[i][j]<minNum){
                minNum = friends[i][j];
                distance = i;
            }
        }
    }
    cout << minNum << "\n" << distance << endl;
    return 0;
}
```