DS [Day – 3]

UID: 24MCI10204

Name: Rahul Saxena

Branch: 24MCA – AI & ML

Question 1: Given a Binary Search Tree (BST), find the Kth smallest element in the tree.

Code:

```
class Node {
   int data;
   Node left, right;
   Node(int value) {
      data = value;
      left = right = null;
   }
}
public class KthSmallestBST {
   static int count = 0;
   static int answer = -1;
   static void findKthSmallest(Node root, int k) {
      if (root == null)
         return;
      findKthSmallest(root.left, k);
      count++;
      if (count == k) {
         answer = root.data;
         return;
      }
      findKthSmallest(root.right, k);
   }
   public static void main(String[] args) {
      Node root = new Node(5);
      root.left = new Node(3);
      root.right = new Node(7);
      root.left.left = new Node(2);
      root.left.right = new Node(4);
      root.right.right = new Node(8);
      int k = 3;
      findKthSmallest(root, k);
      if (answer != -1)
         System.out.println("The " + k + "rd smallest element is: " + answer);
      else
         System.out.println("K is larger than number of nodes in the tree.");
   }
}
```

Question 2: You are given a directed weighted graph where the weights on the edges change over time. Each edge (u, v) has an associated weight function f(t), where t is the time at which the edge is traversed. You need to find the shortest path from a given source node to a destination node at a specified time T.

The edge weight changes periodically (e.g., it could represent traffic conditions or road congestion that varies over time). You need to calculate the shortest path from the source to the destination at a specific time, considering the dynamic nature of the edge weights. Write a program in c/c++ to perform said method.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;
const int MAX = 1e9;
struct Edge {
   int to;
   int baseWeight;
   int fluctuation;
   int period;
   int getWeight(int t) {
      int mod = t % period;
      if (mod < period / 2)
         return baseWeight + fluctuation;
      else
         return baseWeight;
   }
};
struct Node {
   int id;
   int time;
   bool operator>(const Node& other) const {
      return time > other.time;
   }
};
int findShortestTime(int n, vector<vector<Edge>>& graph, int src, int dest, int startTime) {
   vector<int> minTime(n, MAX);
   priority_queue<Node, vector<Node>, greater<Node>> pq;
   minTime[src] = startTime;
   pq.push({src, startTime});
   while (!pq.empty()) {
      Node curr = pq.top();
      pq.pop();
      int u = curr.id;
      int currTime = curr.time;
      if (u == dest)
         return currTime;
      if (currTime > minTime[u])
         continue;
```

```
        for (int i = 0; i < graph[u].size(); i++) {
            Edge e = graph[u][i];
            int travelTime = e.getWeight(currTime);
            int newTime = currTime + travelTime;
            if (newTime < minTime[e.to]) {
                minTime[e.to] = newTime;
                pq.push({e.to, newTime});
            }
        }
    }
    return -1;
}
int main() {
    int n = 5;
    vector<vector<Edge>> graph(n);
    Edge e01;
    e01.to = 1;
    e01.baseWeight = 2;
    e01.fluctuation = 3;
    e01.period = 10;
    graph[0].push_back(e01);
    Edge e12;
    e12.to = 2;
    e12.baseWeight = 5;
    e12.fluctuation = 0;
    e12.period = 1;
    graph[1].push_back(e12);
    Edge e03;
    e03.to = 3;
    e03.baseWeight = 8;
    e03.fluctuation = 0;
    e03.period = 1;
    graph[0].push_back(e03);
    Edge e32;
    e32.to = 2;
    e32.baseWeight = 2;
    e32.fluctuation = 0;
    e32.period = 1;
    graph[3].push_back(e32);
    Edge e24;
    e24.to = 4;
    e24.baseWeight = 1;
    e24.fluctuation = 2;
    e24.period = 5;
    graph[2].push_back(e24);
    int start = 0, end = 4, startTime = 0;
    int result = findShortestTime(n, graph, start, end, startTime);
    if (result != -1)
        cout << "Minimum time to reach destination is: " << result << endl;
    else
        cout << "Destination can't be reached from source." << endl;
    return 0;}
```