```java
/*  (FAST Dijkstra's Algorithm below)
 * Dijkstra's Algorithm in an accepted solution on UVaOJ
 ()using
 * Binary Min Heap or a Priority Queue
 *
 * */
import java.util.*;

class fast_Dijkstra_Alg{
    static Node[] G;
    static int M;
    static class Node {
        List<Edge> adj;
        int n;
        public Node(int N){
            adj = new ArrayList<Edge>();
            n=N;
        }
    }
    static class Edge{
        int to, weight;
        public Edge(int t, int w){
            to=t;
            weight = w;
        }
    }
    static class QueueItem implements Comparable<QueueItem> {
        int v, dist;
        public QueueItem(int v, int dist) {
            this.v = v; this.dist = dist;
        }
        public int compareTo(QueueItem q) {
            return this.dist - q.dist;
        }
    }
    public static void addEdge(int u,int v, int w){
        G[u].adj.add(new Edge(v,w));
        G[v].adj.add(new Edge(u,w));
    }
    public static void makeGraph(int n){
```

```
40          G = new Node[n];
41          for(int i =0; i<n; i++){
42              G[i]=new Node(i);
43          }
44      }
45      public static int dijkstra(int s, int t, int n)
46      {
47          PriorityQueue<QueueItem> pq = new
  PriorityQueue<QueueItem>();
48          pq.offer(new QueueItem(s,0));
49          int[] tdis = new int[n];
50          Arrays.fill(tdis, Integer.MAX_VALUE);
51          tdis[s]=0;
52          int[] from = new int[n];
53          Arrays.fill(from, 0);
54          while (!pq.isEmpty()){
55              QueueItem c = pq.poll();
56              List<Edge> E = G[c.v].adj;
57              if(E.size()>0){
58                  for(Edge e : E)
59                  {
60                      if(tdis[e.to]> tdis[c.v]+e.weight){
61                          tdis[e.to] = Math.min(tdis[e.to],
  tdis[c.v]+e.weight);
62                          from[e.to] = c.v;
63                          pq.offer(new
  QueueItem(e.to,tdis[e.to]));
64                      }
65                  }
66              }
67          }
68          return tdis[t];
69      }
70      public static void main(String[] args){
71          Scanner scan = new Scanner(System.in);
72          int K = scan.nextInt();
73          for(int i =0; i<K; i++){
74              int N = scan.nextInt();
75              M = scan.nextInt();
76              int S = scan.nextInt();
```

```
77              int T = scan.nextInt();
78              makeGraph(N);
79              for(int z =0; z<M; z++){
80                  int u = scan.nextInt();
81                  int v = scan.nextInt();
82                  int w = scan.nextInt();
83                  addEdge(u,v,w);
84              }
85//             for(int z =0; z<N; z++){
86//                 System.out.println(G[z].adj.size());
87//             }
88              int temp = -1;
89              temp = dijkstra(S,T,N);
90              if(temp!=Integer.MAX_VALUE && M!=0)
91              {
92                  System.out.println("Case #"+(i+1)+": "+temp);
93              }
94
95              else {
96                  System.out.println("Case #"+(i+1)+":
  "+"unreachable");
97
98              }
99          }
100         scan.close();
101     }
102 }
103
```