# 1334. Find the City With the Smallest Number of Neighbors at a Threshold Distance

```
/*
   Dijkstra algorithm
   Time complexity: O(V+V(ElogV)+V)=O(V+VElogV+V)=O(n^3)
   Extra space complexity: O(2VE+3V)
*/
typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;
typedef std::vector<vii> vvii;

class Solution {
   public:
      vvii graph;
   public:
      void build_graph(int n, std::vector<std::vector<int>>& edges){
         graph.resize(n);
         for(auto& edge: edges){
            graph[edge[0]].push_back({edge[1],edge[2]});
            graph[edge[1]].push_back({edge[0],edge[2]});
         }
      }
```

```
int dijkstra(int n,int start,std::vector<bool>& visited,int distanceThreshold){
        std::priority_queue<ii,vii,std::greater<ii>> q;
        q.push({0,start});
        while(!q.empty()){
            auto [dist,u]=q.top();
            q.pop();

            if(visited[u]) continue;
            visited[u]=true;
            for(auto& neighbor: graph[u]){
                int v=neighbor.first;
                int w=neighbor.second;
                int d=dist+w;
                if(d<=distanceThreshold) q.push({d,v});
            }
        }
        int cnt=-1;
        for(int i=0;i<n;++i) if(visited[i]) cnt++;
        return cnt;
    }

    int findTheCity(int n, std::vector<std::vector<int>>& edges, int distanceThreshold) {
        build_graph(n,edges);

        std::vector<int> citie_neighbors_count(n);
        for(int i=0;i<n;++i){
            std::vector<bool> visited(n,false);
            citie_neighbors_count[i]=dijkstra(n,i,visited,distanceThreshold);
        }

        int
min_number=*std::min_element(citie_neighbors_count.begin(),citie_neighbors_count.end());

        for(int i=n-1;i>=0;i--){
            if(citie_neighbors_count[i]==min_number) return i;
        }

        return -1;
    }
};
```