基本解法

Java代码

```
class Solution {
    public int[] dist; //dist存储从源节点k到各个节点的最短距离
    public int networkDelayTime(int[][] times, int N, int K) {
        //构建有权图的邻接表adj
        ArrayList<ArrayList<int[]>> adj = new ArrayList<ArrayList<int[]>>(N+1);
        for(int i=0; i <= N; i++)  adj.add(new ArrayList<int[]>());
        for(int i=0; i < times.length; i++)
            adj.get(times[i][0]).add(new int[]{times[i][1], times[i][2]});
        dist = new int[N+1];
        //初始化最短距离为无穷大
        for (int node = 1; node <= N; ++node) dist[node] = Integer.MAX_VALUE;
        //从起始节点K开始dfs
        dfs(adj, K, 0);
        //寻找最短路径中的最大值
        int ans = 0;
        for (int node = 1; node <= N; ++node){
            if (dist[node] == Integer.MAX_VALUE) return -1;
            ans = Math.max(ans, dist[node]);
        }
        return ans;
    }
    public void dfs(ArrayList<ArrayList<int[]>> adj, int node, int elapsed) {
        //剪枝：若之前存在更快到达node节点的路径，本次无需继续扩展该节点
        if (elapsed >= dist[node]) return;
        dist[node] = elapsed;
        for(int j=0; j<adj.get(node).size(); j++){
            int[] vert=adj.get(node).get(j);
            dfs(adj, vert[0], elapsed + vert[1]);
        }
    }
}
```

优化解法

Java代码

```
class Solution {
    public int networkDelayTime(int[][] times, int N, int K) {
        //构建有权图的邻接表adj
        ArrayList<ArrayList<int[]>> adj = new ArrayList<ArrayList<int[]>>(N+1);
        for(int i=0; i <= N; i++)  adj.add(new ArrayList<int[]>());
        for(int i=0; i < times.length; i++)
            adj.get(times[i][0]).add(new int[]{times[i][1], times[i][2]});
        //dist存储从源节点k到各个节点的最短距离，初始化最短距离为无穷大
        int[] dist = new int[N+1];
        for (int node = 1; node <= N; ++node)
            dist[node] = Integer.MAX_VALUE;
        dist[K] = 0;
```

```
        boolean[] visited = new boolean[N+1];
        while (true) {
            int candNode = -1;
            int candDist = Integer.MAX_VALUE;
            for (int i = 1; i <= N; ++i) {
                if (!visited[i] && dist[i] < candDist) {
                    candDist = dist[i];
                    candNode = i;
                }
            }
            if (candNode < 0) break;
            visited[candNode] = true;
            for(int j=0; j<adj.get(candNode).size(); j++){
                int[] info =adj.get(candNode).get(j);
                dist[info[0]] = Math.min(dist[info[0]], dist[candNode] +
info[1]);
            }
        }
        int ans = 0; //寻找最短路径中的最大值并返回
        for (int node = 1; node <= N; ++node){
            if (dist[node] == Integer.MAX_VALUE) return -1;
            ans = Math.max(ans, dist[node]);
        }
        return ans;
    }
}
```

C++代码

```
class Solution{
public:
    int networkDelayTime(vector<vector<int>>& times, int N, int K) {
        //构建邻接矩阵
        vector<vector<int>> graph(N + 1, vector<int>(N + 1, -1));
        for (const auto& time : times) {
            graph[time[0]][time[1]] = time[2];
        }
        //dist 数组
        vector<int> distance(N + 1, -1);
        for (int i = 1; i <= N; i++) {
            distance[i] = graph[K][i];
        }

        distance[K] = 0;
        //D集合
        vector<bool> visited(N + 1, false);
        visited[K] = true;

        for (int i = 1; i <= N - 1; i++) {
            int minDistance = numeric_limits<int>::max();
            int minIdx = 1;
            // 遍历所有节点，找到离K最近的节点
            for (int j = 1; j <= N; j++) {
                if (!visited[j] && distance[j] != -1 && distance[j] <
minDistance) {
                    minDistance = distance[j];
                    minIdx = j;
```

```
                }
            }
            visited[minIdx] = true;

            // 根据刚刚找到的最短距离节点，通过该节点更新K节点于其他节点的距离
            for (int j = 1; j <= N; j++) {
                if (graph[minIdx][j] != -1) { // reachable
                    if (distance[j] != -1) {
                        distance[j] = min(distance[j], distance[minIdx] +
graph[minIdx][j]);
                    } else {
                        distance[j] = distance[minIdx] + graph[minIdx][j];
                    }
                }
            }
        }
        int maxDistance = 0;
        for (int i = 1; i <= N; i++) {
            if(distance[i] == -1) {
                return -1;
            }
            maxDistance = max(maxDistance, distance[i]);
        }
        return maxDistance;
    }
};
```

Python代码

```
class Solution(object):
    def networkDelayTime(self, times, N, K):
        #建立邻接表
        graph = collections.defaultdict(list)
        for u, v, w in times:
            graph[u].append((v, w))

        #dist矩阵
        dist = {node: float('inf') for node in xrange(1, N+1)}
        dist[K] = 0
        #D集合
        visited = [False] * (N+1)

        while True:
            cand_node = -1
            cand_dist = float('inf')
            #寻找下一个可以加入D的节点
            for i in xrange(1, N+1):
                if not visited[i] and dist[i] < cand_dist:
                    cand_dist = dist[i]
                    cand_node = i

            if cand_node < 0: break
            visited[cand_node] = True
            for nei, d in graph[cand_node]:
                dist[nei] = min(dist[nei], dist[cand_node] + d)

        ans = max(dist.values())
```

```
        return ans if ans < float('inf') else -1
```