拉勾教育
—互联网人实战大学—

图中最短路径：网络延迟时间

中等/有向图、单源最短路、迪杰斯特拉算法

学习目标

拉勾教育
—互联网人实战大学—

- 理解图中最短路径的问题与类型
- 巩固图中深度优先搜索在不同问题中的应用
- 掌握图结构中的单源最短路径迪杰斯特拉算法

题目描述

有 N 个网络节点，标记为 1 到 N。

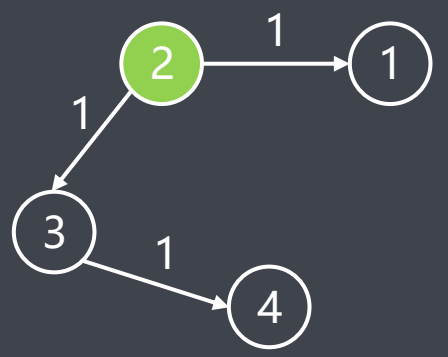
给定一个列表 times，表示信号经过有向边的传递时间。 times[i] = (u, v, w)，其中 u 是源节点，v 是目标节点， w 是一个信号从源节点传递到目标节点的时间。

现在，我们从某个节点 K 发出一个信号。需要多久才能使所有节点都收到信号？如果不能使所有节点收到信号，返回 -1。

输入：times = [[2,1,1],[2,3,1],[3,4,1]], N = 4, K = 2
输出：2

一. Comprehend 理解题意

网络延迟时间



- 题目中给定的样例可以被可视化成右图
- 我们需要找到从给定节点k发出的信号多久后可以到达所有节点
- 每条边都被赋予了长度（边权），即信号在该边传输所用的时间
- 观察：从节点k发出的信号到达节点a的最短时间就是在图中节点k到节点a最短路径的长度
- 题目所求的时间就是距离节点k最远的节点与节点k之间的最短距离

输入：times = [[2,1,1],[2,3,1],[3,4,1]], N = 4, K = 2

输出：2

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

额外信息

- 图中的节点个数N在[1, 100]之间
- 图中有向边的个数在[1, 6000]之间
- 图中有向边的边权在[0,100]之间

二. Choose 数据结构及算法思维选择

数据结构选择

- 经过前面的分析，我们可以将该问题建模为一个图上的最短路径问题，所以我们采用图作为我们的数据结构。
- 由于输入的节点个数不超过100个，边数不超过6000，邻接表或邻接矩阵都可以作为我们的选择，由于本题中的图是有权图，邻接矩阵和邻接表中还应存储额外的边权信息。
- 输入的数据是图中的节点个数、有向边信息与起始节点
- 输出的数据类型为一个整形值，表示所有节点收到信号需要的时间

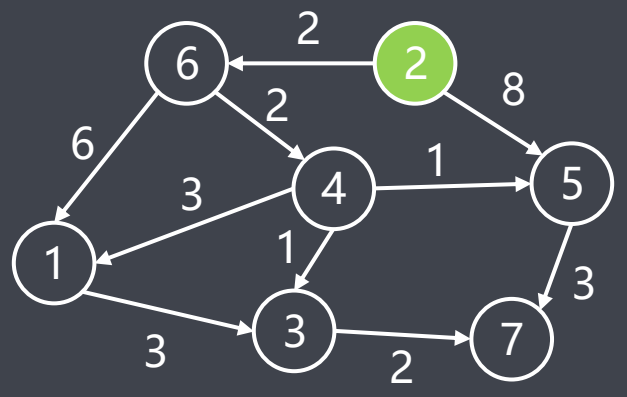
二. Choose 数据结构及算法思维选择

算法思维选择

- 我们已经将该问题建模成一个图中最短路径的问题，此题我们需要求解从给定节点出发到图中其他各个节点的最短距离。
- 该问题被称为图中的单源最短路径问题，我们可以使用上一节介绍的图中的深度优先搜索来完成这个任务，遍历从源节点出发到各个节点的所有路径，记录各条路径的长度

三. Code 基本解法及编码实现

解题思路剖析



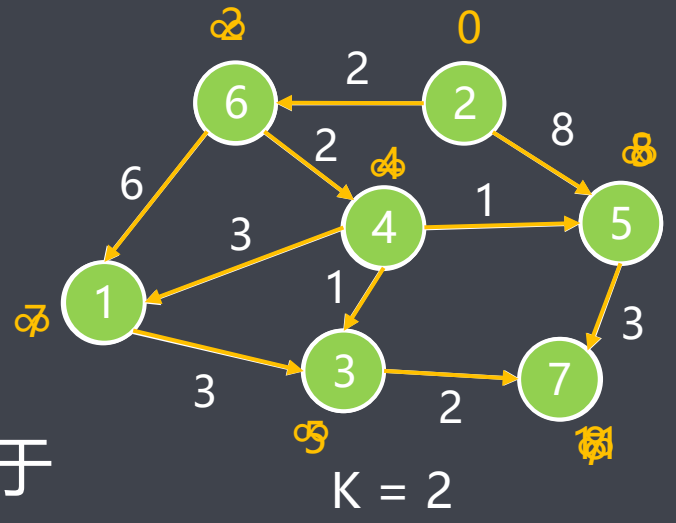
K = 2

- 深度优先搜索
- 在题目4.1中，我们只要获取节点间连通信息，无须扩展在之前任何一条路径中已经访问过的节点
- 此题中我们需要找到从起始节点出发的所有路径，若遇到节点只要未出现在本次的路径之中，无论是否访问过，我们都需要扩展

三. Code 基本解法及编码实现

解题思路剖析

- 使用数组记录当前从源节点出发到当前节点的最短距离，初始化为无穷大
- 从源节点出发，记录当前路径的长度，如果小于到达节点当前的最短距离，则进行更新
- 剪枝：如果当前节点的最短距离比当前路径长度要短，我们可以不用在本次继续扩展该节点
- DFS搜索完成，所需要的时间为7



三. Code 基本解法及编码实现

复杂度分析

- 时间复杂度
  - 我们需要根据输入的边信息建立邻接表，时间复杂度为O(|V|+|E|)
  - 图中的深度优先搜索算法，因为我们基本上需要遍历从K出发的所有路径，时间复杂度为$O(|V|^{|V|})$
  - 总时间复杂度为$O(|V|^{|V|}+|E|)$
- 空间复杂度
  - 空间消耗方面我们需要建立邻接表来存储图的连接关系以及最短距离矩阵，空间复杂度为O(|V|+|E|)

三. Code 基本解法及编码实现

Java编码实现

```java
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;
    dfs(adj, K, 0);    //从起始节点K开始dfs
    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;
}
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]);
    }
}
```

执行结果： 通过 显示详情 >

执行用时： 394 ms ，在所有 Java 提交中击败了 6.31% 的用户

内存消耗： 42.7 MB ，在所有 Java 提交中击败了 61.54% 的用户

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

是否存在无效代码或者无效空间消耗

是否有更好的算法思维

- 使用深度优先搜索基本上需要遍历所有从源节点出发的路径，是否存在更优的算法思路?
- 比如我们前面讲解过的贪心算法?
- 下面我们就介绍图论中单源最短路径的经典算法：迪杰斯特拉算法

四. Consider 思考更优解

关键知识点：迪杰斯特拉算法（Dijkstra's algorithm）

- 迪杰斯特拉算法，是由荷兰计算机科学家艾兹赫尔·戴克斯特拉在1956年发现的算法。戴克斯特拉算法使用类似广度优先搜索的方法解决赋权图的单源最短路径问题。
- 迪杰斯特拉算法的主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止。Dijkstra算法是很有代表性的最短路径算法，在很多专业课程中都作为基本内容有详细的介绍，如数据结构，图论，运筹学等等。注意该算法要求图中不存在负权边。

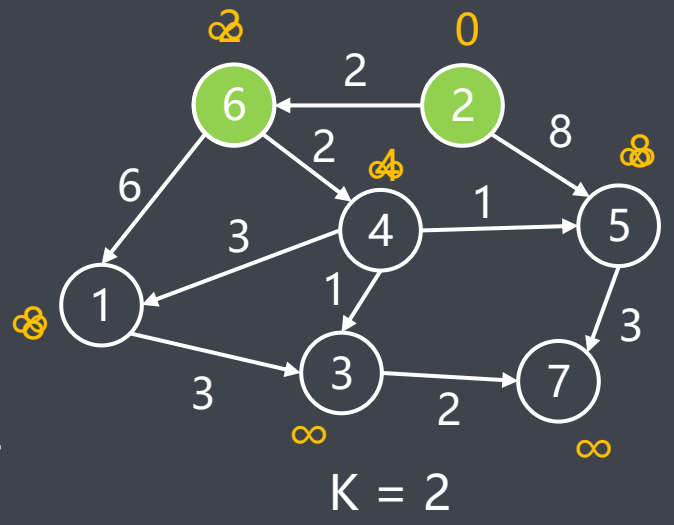
四. Consider 思考更优解

关键知识点：迪杰斯特拉算法（Dijkstra's algorithm）

- 迪杰斯特拉算法的主要思想
  - 定义一个节点集合D表示从源点k出发已经确定最短路径的前n近的节点
  - 比如初始D={k}，k到自己的距离为0，k一定是最近的节点
  - 那么下一个可以确定最短路径的节点（即第n+1近的节点）一定是与D集合中节点直接相连的节点
    - 反证法：如果第n+1近的节点p不与D中节点相邻，则从k出发到p至少会遇到一个不在D中的节点q，该节点到k的距离小于p，则p一定不是第n+1近的节点
  - 当D={k}时，也就是说，距离k最近的节点（除了k自己）一定是与k直接相连的节点，符合我们的直观感受

五. Code 最优解思路及编码实现

解题思路剖析



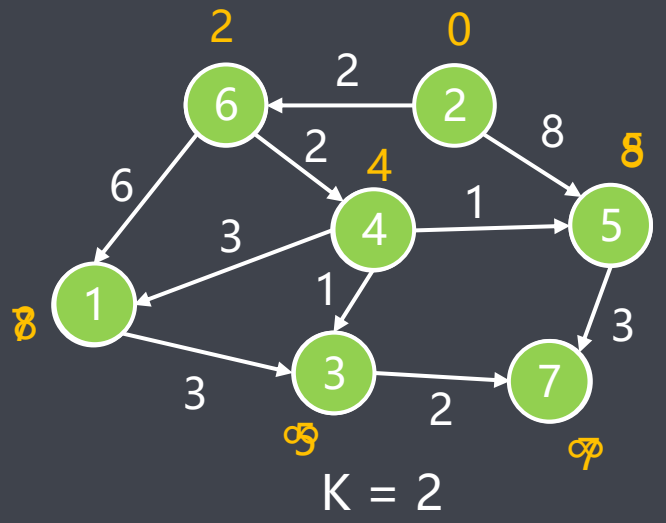
- 同样使用数组dist来记录当前从源节点出发到当前节点的最短距离，初始化为无穷大
- 初始D={2}，我们尝试来更新与2相连节点的距离

  dist[5] > dist[2] + 8 更新 dist[5] ← dist[2] + 8 = 8

  dist[6] > dist[2] + 2 更新 dist[6] ← dist[2] + 2 = 2
- 那么最近的节点一定在5与6之间，dist[5] > dist[6] 将6加入D
- 更新与6相连的节点的距离

五. Code 最优解思路及编码实现

解题思路剖析



- 此时D={2，6}，第二近的节点一定在 1,4,5中间
- 可以发现dist[4]最小，加入D
- 使用dist[4]更新相邻节点的dist值
- 此时D={2,6,4}，与D中节点直接相邻的节点有 1,3,5，比较他们的值，发现5和3的dist值都为最小值5，可以同时加入D
- 使用3和5的dist值尝试更新相邻的节点
- 此时D={2,6,4,3,5}，相邻的节点1,7，距离均为最小值7，加入D

所有节点都在D中，算法结束！

五. Code 最优解思路及编码实现

复杂度分析

- 时间复杂度
  - 建立邻接表的时间复杂度为O(|V|+|E|)
  - 迪杰斯特拉算法中每次我们需要找到未确定为最短路径的顶点中dist值最小的那个，如果我们每次遍历所有顶点，时间复杂度为$O(|V|^2)$，如果我们维护一个优先队列，时间复杂度为O((|E|+|V|)*log|V|)
  - 综上，总时间复杂度为$O(|V|^2)$或O((|E|+|V|)*log|V|)
- 空间复杂度
  - 空间消耗方面我们需要建立邻接表，空间复杂度为O(|V|+|E|)

五. Code 最优解思路及编码实现

Java编码实现

```java
public int networkDelayTime(int[][] times, int N, int K) {
    //构建有权图的邻接表adj （同基本解法）
    //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]; //即集合D
    while (true) {
        int candNode = -1;
        int candDist = Integer.MAX_VALUE;
        for (int i = 1; i <= N; ++i) {      //寻找下一个加入集合D的节点
            if (!visited[i] && dist[i] < candDist) {
                candDist = dist[i];
                candNode = i;
            }
        }
        if (candNode < 0) break; //到所有可达节点的最短路径都寻找完毕，退出循环
        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;
}
```

执行结果： 通过 显示详情 >

执行用时： 7 ms ，在所有 Java 提交中击败了 80.90% 的用户

内存消耗： 41.8 MB ，在所有 Java 提交中击败了 85.11% 的用户

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

1. 若输入的图为无向图，算法应如何修改?

延伸扩展

- 最短路径问题是图论中的经典问题
  - 生活中很多求最短路径、最短时间的问题都可以归结为该类问题
  - 高铁网络可以被看作一个有权图，连接了不同的火车站，很多规划路径的任务就可以被视作一个最短路径问题

总结

- 理解图中最短路径的问题与类型
- 巩固图中深度优先搜索在不同问题中的应用
- 掌握图结构中的单源最短路径迪杰斯特拉算法

课后练习

1. K 站中转内最便宜的航班（leetcode 787/中等）
2. 概率最大的路径（leetcode 1514/中等）
3. 迷宫ii（leetcode 505/中等）
4. 迷宫iii（leetcode 499/中等）

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容