【华为OD机考 统一考试机试C卷】电脑病毒感染（C++ Java JavaScript Python）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷） 。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多 ，按照之前的经验C卷D卷部分考题会复用A卷/B卷题，博主正积极从考过的同学收集C卷和D卷真题，可以查看下面的真题目录。

真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为OD机考C卷和D卷真题华为OD机考C卷和D卷题解

题目描述

一个局域网内有很多台电脑，分别标注为 0 ~ N-1 的数字。相连接的电脑距离不一样，所以感染时间不一样，感染时间用 t 表示。

其中网络内一台电脑被病毒感染，求其感染网络内所有的电脑最少需要多长时间。如果最后有电脑不会感染，则返回-1。

给定一个数组 times 表示一台电脑把相邻电脑感染所用的时间。

如图：path[i] = {i, j, t} 表示：电脑 i->j，电脑 i 上的病毒感染 j，需要时间 t。

输入描述

第一行输入一个整数N， 表示局域网内电脑个数 N， 1 ≤ N ≤ 200 ;

第二行输入一个整数M ,表示有 M 条网络连接;

接下来M行 ,每行输入为 i , j , t 。表示电脑 i 感染电脑j 需要时间 t 。 （1 ≤ i , j ≤ N）

最后一行为病毒所在的电脑编号。

输出描述

输出最少需要多少时间才能感染全部电脑，如果不存在输出 -1

用例

输入	**4** **3** **2 1 1** **2 3 1** **3 4 1** **2**
输出	2
说明	第一个参数：局域网内电脑个数N，1 ≤ N ≤ 200； 第二个参数：总共多少条网络连接 第三个 2 1 1 表示2->1时间为1 第六行：表示病毒最开始所在电脑号2

解题思路

Bellman-Ford算法是一种用于在加权图中找到从单个源点到所有其他顶点的最短路径的算法。它能够处理带有负权边的图，这是它与Dijkstra算法的主要区别。然而，如果图中存在负权回路，即一个总权重为负的环路，Bellman-Ford算法可以检测到这种情况。

算法的工作原理如下：

1. **初始化距离数组**：算法开始时，除了源点（在上面的代码中是变量 `K` ）的距离被初始化为0以外，所有顶点的距离都被设置为无穷大（在上面的代码中是 `INF` ）。
2. **松弛操作**：算法会进行 `N-1` 次迭代，其中 `N` 是图中顶点的数量。在每次迭代中，算法会遍历所有的边，并尝试更新每条边的目标顶点的距离。如果通过当前边到达目标顶点的距离小于已知的最短距离，则更新该顶点的最短距离。这个过程称为松弛操作。
3. **检测负权回路**：在 `N-1` 次迭代之后，算法会再次遍历所有的边，检查是否还能进行松弛操作。如果可以，这意味着图中存在负权回路，因为最短路径应该已经在前面的 `N-1` 次迭代中被确定下来。

在下面的代码中， `networkDelayTime` 函数实现了Bellman-Ford算法：

- `times` 数组包含了图中所有的边，其中每个元素是一个三元组 `[u, v, w]` ，表示从顶点 `u` 到顶点 `v` 的边，其权重为 `w` 。
- `dist` 数组用于存储从源点 `K` 到每个顶点的最短距离。
- 在 `for` 循环中，算法遍历所有边，并对每条边执行松弛操作。如果 `dist[u] + w < dist[v]` ，则更新 `dist[v]` 为 `dist[u] + w` 。
- 在所有顶点的最短距离被计算出来后，算法找出最长的最短距离，即 `maxWait` ，这是感染所有电脑所需的最少时间。

- 如果有顶点的距离仍然是无穷大，这意味着有些顶点无法从源点 `K` 到达，函数返回 `-1` 。

Bellman-Ford算法的时间复杂度是 `O(VE)` ，其中 `V` 是顶点的数量， `E` 是边的数量。这使得它在稠密图中效率较低，但它是处理带有负权边的图的有效算法。

C++

```cpp
#include <iostream>
#include <vector>
#include <climits>
#include <algorithm>

using namespace std;
// 计算感染所有电脑所需的最少时间的函数
int networkDelayTime(vector<vector<int>>& times, int N, int K) {
    const int INF = INT_MAX / 2; // 定义无穷大的值，用于初始化距离数组
    vector<int> dist(N, INF); // 存储从源电脑到其他所有电脑的最短感染时间
    dist[K] = 0; // 源电脑的感染时间为0

    // 使用Bellman-Ford算法更新所有电脑的最短感染时间
    for (int i = 0; i < N; ++i) {
        for (const auto& time : times) {
            int u = time[0], v = time[1], w = time[2];
            // 如果可以通过电脑u感染到电脑v，并且时间更短，则更新电脑v的感染时间
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
            }
        }
    }

    // 找出所有电脑中最长的感染时间
    int maxWait = 0;
    for (int i = 0; i < N; ++i) {
        // 如果有电脑的感染时间仍为无穷大，表示该电脑不可被感染，返回-1
        if (dist[i] == INF) return -1;
        // 更新最长的感染时间
        maxWait = max(maxWait, dist[i]);
    }

    // 返回感染所有电脑所需的最少时间
    return maxWait;
```

```
}

int main() {
    int N, connections;
    cin >> N >> connections; // 电脑的数量和网络连接的数量
    vector<vector<int>> times(connections, vector<int>(3)); // 存储每个连接和对应的感染时间
    for (int i = 0; i < connections; ++i) {
        // 读取每个连接的信息，将电脑编号减1转换为从0开始的索引
        cin >> times[i][0] >> times[i][1] >> times[i][2];
        times[i][0]--; // 感染源电脑编号
        times[i][1]--; // 被感染电脑编号
    }
    int initial;
    cin >> initial; // 初始被感染的电脑编号，转换为从0开始的索引
    initial--;

    // 输出感染所有电脑所需的最少时间
    cout << networkDelayTime(times, N, initial) << endl;
    return 0;
}
```

Java

```
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt(); // 电脑的数量
        int connections = sc.nextInt(); // 网络连接的数量
        int[][] times = new int[connections][3]; // 存储每个连接和对应的感染时间
        for (int i = 0; i < connections; i++) {
            // 读取每个连接的信息，将电脑编号减1转换为从0开始的索引
            times[i][0] = sc.nextInt() - 1; // 感染源电脑编号
            times[i][1] = sc.nextInt() - 1; // 被感染电脑编号
            times[i][2] = sc.nextInt(); // 感染所需时间
        }
        int initial = sc.nextInt() - 1; // 初始被感染的电脑编号，转换为从0开始的索引
        sc.close(); // 关闭输入流
```

```java
        // 输出感染所有电脑所需的最少时间
        System.out.println(networkDelayTime(times, N, initial));
    }

    // 计算感染所有电脑所需的最少时间的函数
    public static int networkDelayTime(int[][] times, int N, int K) {
        final int INF = Integer.MAX_VALUE / 2; // 定义无穷大的值，用于初始化距离数组
        int[] dist = new int[N]; // 存储从源电脑到其他所有电脑的最短感染时间
        Arrays.fill(dist, INF); // 初始化所有感染时间为无穷大
        dist[K] = 0; // 源电脑的感染时间为0

        // 使用Bellman-Ford算法更新所有电脑的最短感染时间
        for (int i = 0; i < N; i++) {
            for (int[] time : times) {
                int u = time[0], v = time[1], w = time[2];
                // 如果可以通过电脑u感染到电脑v，并且时间更短，则更新电脑v的感染时间
                if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                }
            }
        }

        // 找出所有电脑中最长的感染时间
        int maxWait = 0;
        for (int i = 0; i < N; i++) {
            // 如果有电脑的感染时间仍为无穷大，表示该电脑不可被感染，返回-1
            if (dist[i] == INF) return -1;
            // 更新最长的感染时间
            maxWait = Math.max(maxWait, dist[i]);
        }

        // 返回感染所有电脑所需的最少时间
        return maxWait;
    }
}
```

javaScript

```javascript
const readline = require('readline');

// 创建readline接口实例
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

const lines = []; // 用于存储输入的每一行

// 监听line事件来获取每一行输入
rl.on('line', (line) => {
  lines.push(line);
}).on('close', () => {
  // 当输入完成后，处理lines数组中的数据
  const N = parseInt(lines[0], 10); // 电脑的数量
  const connections = parseInt(lines[1], 10); // 网络连接的数量
  const times = []; // 存储每个连接和对应的感染时间

  for (let i = 0; i < connections; i++) {
    const line = lines[i + 2].split(' ').map(Number);
    // 读取每个连接的信息，将电脑编号减1转换为从0开始的索引
    times.push([line[0] - 1, line[1] - 1, line[2]]); // 感染源电脑编号，被感染电脑编号，感染所需时间
  }

  const initial = parseInt(lines[connections + 2], 10) - 1; // 初始被感染的电脑编号，转换为从0开始的索引

  // 计算感染所有电脑所需的最少时间
  console.log(networkDelayTime(times, N, initial));
});

// 计算感染所有电脑所需的最少时间的函数
function networkDelayTime(times, N, K) {
  const INF = Number.MAX_SAFE_INTEGER / 2; // 定义无穷大的值，用于初始化距离数组
  const dist = new Array(N).fill(INF); // 存储从源电脑到其他所有电脑的最短感染时间
  dist[K] = 0; // 源电脑的感染时间为0

  // 使用Bellman-Ford算法更新所有电脑的最短感染时间
  for (let i = 0; i < N; i++) {
    for (const [u, v, w] of times) {
```

```
      // 如果可以通过电脑u感染到电脑v，并且时间更短，则更新电脑v的感染时间
      if (dist[u] + w < dist[v]) {
        dist[v] = dist[u] + w;
      }
    }
  }

  // 找出所有电脑中最长的感染时间
  let maxWait = 0;
  for (let i = 0; i < N; i++) {
    // 如果有电脑的感染时间仍为无穷大，表示该电脑不可被感染，返回-1
    if (dist[i] === INF) return -1;
    // 更新最长的感染时间
    maxWait = Math.max(maxWait, dist[i]);
  }

  // 返回感染所有电脑所需的最少时间
  return maxWait;
}
```

Python

```
import sys

# 计算感染所有电脑所需的最少时间的函数
def network_delay_time(times, N, K):
    INF = float('inf')  # 定义无穷大的值，用于初始化距离数组
    dist = [INF] * N  # 存储从源电脑到其他所有电脑的最短感染时间
    dist[K] = 0  # 源电脑的感染时间为0

    # 使用Bellman-Ford算法更新所有电脑的最短感染时间
    for _ in range(N):
        for u, v, w in times:
            # 如果可以通过电脑u感染到电脑v，并且时间更短，则更新电脑v的感染时间
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w

    # 找出所有电脑中最长的感染时间
    max_wait = max(dist)
    # 如果有电脑的感染时间仍为无穷大，表示该电脑不可被感染，返回-1
```

```
    return max_wait if max_wait < INF else -1


N = int(input())
connections = int(input()) # 电脑的数量和网络连接的数量
times = []  # 存储每个连接和对应的感染时间
for _ in range(connections):
    # 读取每个连接的信息，将电脑编号减1转换为从0开始的索引
    u, v, w = map(int, input().split())
    times.append((u - 1, v - 1, w))  # 感染源电脑编号，被感染电脑编号，感染所需时间
initial = int(input()) - 1  # 初始被感染的电脑编号，转换为从0开始的索引

# 输出感染所有电脑所需的最少时间
print(network_delay_time(times, N, initial))
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师