数据结构 A
作业 11 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：图
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

用 Dijkstra 算法求一个带权有向图 G 中从顶点 0 出发的最短路径，在算法执行的某时刻，$S=\{0,2,3,4\}$，下一步选取的目标顶点可能是（　　）
A. 顶点 2
B. 顶点 3
C. 顶点 4
D. 顶点 7

答案：D

解析：Dijkstra 算法中，一般会把节点分为 S 和 T 两个集合部分，S 是已经被更新过的，**并且不会被再次更新的那些节点**，而 T 是还未更新的节点，Dijkstra 算法需要在 T 里面找到离源点最近的那一个更新至 S，并利用这个节点更新 T 中的距离。

因此只有顶点 7 有可能被作为这个新点被更新，而 $2,3,4$ 已经在 S 集合里，没有更近的点可以更新它们了。

习题 2 单选题

若一个有向图中的顶点不能排成一个拓扑序列，则可断定该有向图（　　）
A. 是个有根有向图
B. 是个强连通图
C. 含有多个入度为 0 的顶点
D. 含有顶点数目大于 1 的强连通分量

答案：D

解析：仍然使用排除法。先考虑拓扑序列的定义，一般而言是指**有向无环图**。也就是说，**不能有环**，但可以不连通（分成多块），也可以有多个源点（入度为 0 的点）。

按照定义，依次检查选项：

A：不能排成拓扑序列的图可以是无根的有向图，图出入度为 0 的点不一定唯一。

B：如果图强连通，说明图里任意两个点可以互相到达。这样的条件太苛刻，而当一个图无法拍成拓扑序列时，只需要有自环以外的环即可。

C：含有多个入度为 0 的顶点与图是否能拓扑无关。

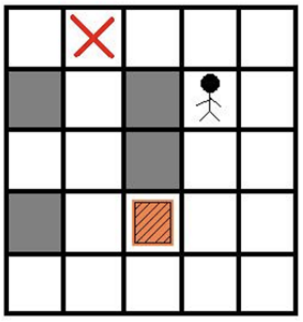
D：顶点数目等于 1 的强连通分量表示自环，不影响拓扑排序结果；而如果有顶点数目大于 1 的强连通分量，说明有环，此时就不能完成拓扑排序前后序依赖的梳理了。

习题 3 推箱子

【问题描述】

推箱子是一个很经典的游戏，今天我们来玩一个简单版本。在一个 $n \times m$ 的房间里有一个箱子和一个搬运工，搬运工的工作就是把箱子推到指定的位置。注意，搬运工只能推箱子而不能拉箱子，因此如果箱子被推到一个角上（如下图所示），那么箱子就不能再被移动了，如果箱子被推到一面墙上，那么箱子只能沿着墙移动。

现在给定房间的结构，箱子的位置，搬运工的位置和箱子要被推去的位置，请你计算出搬运工至少要推动箱子多少格。



【输入形式】

输入数据的第一行是两个正整数 n 和 m（$2 \leq n, m \leq 7$），代表房间的大小，然后是一个 n 行 m 列的矩阵，代表房间的布局，其中 0 代表空的地板，1 代表墙，2 代表箱子的起始位置，3 代表箱子要被推去的位置，4 代表搬运工的起始位置。

【输出形式】

对于每组测试数据，输出搬运工最少需要推动箱子多少格才能将箱子推到指定位置，如果不能推到指定位置则输出 -1。

【样例输入】

```
5 5
0 3 0 0 0
1 0 1 4 0
```

0 0 1 0 0
1 0 2 0 0
0 0 0 0 0
【样例输出】
4
【样例说明】
对于输入的房间布局和箱子、搬运工起始位置，搬运工最少需要推动箱子 4 格才能将箱子推到指定位置。测试数据存放在 in.txt 文件中。
【评分标准】
该题目有 10 个测试用例，每通过一个测试得 10 分。

答案：

```cpp
#include <iostream>
#include <fstream>
#include <cstring>
#include <queue>
using namespace std;
#define MAXN 10
#define INF 0x3f3f3f3f
struct QNode          //队中元素的类型
{ int x, y;           //搬运工的坐标
    int bx, by;       //箱子的坐标
    int step;         //推箱子步数
};
int dx[]={1,-1,0,0};         //x方向偏移量
int dy[]={0,0,1,-1};         //y方向偏移量
int grid[MAXN][MAXN];
int len[MAXN][MAXN][MAXN][MAXN];  //搬运工的坐标和箱子的坐标
int m,n;
bool Judgeperson(QNode p)     //判断位置p中搬运工的坐标是否有效
{   return p.x>=0 && p.x<m && p.y>=0 && p.y<n
            && grid[p.x][p.y]!=1 && p.step < len[p.x][p.y][p.bx][p.by];
}
bool Judgebox(QNode p)       //判断位置p中箱子的坐标是否有效
{   return p.bx>=0 && p.bx<m && p.by>=0 && p.by<n && grid[p.bx][p.by]!=1
            && p.step<len[p.x][p.y][p.bx][p.by];
}
int BFS(QNode st)         //从st开始搜索
{   queue<QNode> qu;
    qu.push(st);
    len[st.x][st.y][st.bx][st.by]=0;
    int ans=INF;
```

```
    while (!qu.empty())
    {   QNode p=qu.front(); qu.pop();    //出队一个元素p
        if(grid[p.bx][p.by]==3)      //找到箱子的目标位置
        {   ans=min(ans,p.step);
            continue;        //队不空时继续搜索
        }
        for(int di=0;di<4;di++)
        {   QNode np=p;       //搬运工向前走一步
            np.x+=dx[di];
            np.y+=dy[di];
            if (Judgeperson(np))      //判断搬运工走一步是否合法
            { if(np.x==np.bx && np.y==np.by) //搬运工和箱子重合
                {   np.bx+=dx[di];     //箱子沿着di方位走一步
                    np.by+=dy[di];
                    np.step++;      //推动箱子一次
                    if (Judgebox(np))    //判断箱子前进是否合法
                    {   len[np.x][np.y][np.bx][np.by]=np.step;
                        qu.push(np);
                    }
                }
                else         //搬运工和箱子尚未重合
                {   len[np.x][np.y][np.bx][np.by]=np.step;
                    qu.push(np);
                }
            }
        }
    }
    return ans;
}
int main()
{
    ifstream inFile;
    inFile.open("in.txt", ios::in);
    if(!inFile){
        cout << "error open in.txt!" << endl;
    }

    QNode st;          //初始状态

    memset(len,0x3f,sizeof(len));     //len所有元素初始化为INF
    while (!inFile.eof()) {
        inFile >> m >> n;
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++) {
                inFile >> grid[i][j];
                if (grid[i][j] == 4)      //搬运工的初始位置
                {
```

```
78                      st.x = i;
79                      st.y = j;
80                      st.step = 0;
81                  } else if (grid[i][j] == 2)     //箱子的初始位置
82                  {
83                      st.bx = i;
84                      st.by = j;
85                  }
86              }
87          int ans = BFS(st);
88          if (ans == INF) ans = -1;
89          printf("%d\n", ans);
90      }
91      return 0;
92  }
```

解析：本题主要考察的是广度优先搜索 BFS。本题有两个关键变量，工人位置坐标和箱子位置坐标。题目中最多一共只会出现 $(nm)^2$ 种状态，也就是箱子行数 $\times$ 箱子列数 $\times$ 工人行数 $\times$ 工人列数。

因此我们对于每个状态，记录从开始状态到它的距离。每次搜索时，检查工人是否能向四个方向（上下左右）移动，移动是否会影响箱子位置，移动的方向用 $(0,1),(0,-1),(1,0),(-1,0)$ 表示。

可以用 *len*[工人行][工人列][箱子行][箱子列] 来存储从出发点到这个状态需要经过的**最小步数**，如果更新相邻点时发现可以做到以更小的步数到达这个点，就更新这个点的新距离。

本题主要复杂之处在于**检查工人是否能向四个方向移动，移动是否会影响箱子位置**，需要较为细致的逻辑判断。要注意输入时的数据应当以合适的格式存储。

习题 4 找最小费用环

【问题描述】

杭州有 n 个景区，景区之间有一些双向的路来连接，现在万先生想找一条旅游路线，这个路线从 A 点出发并且最后回到 A 点，假设经过的路线为 v_1，v_2，…，v_k，v_1，那么必须满足 $k>2$，就是说除了出发点以外至少要经过两个其他不同的景区，而且不能重复经过同一个景区。现在万先生需要你帮他找一条这样的路线，并且花费越少越好。

【输入形式】

第一行是两个整数 n 和 m（$n \le 100$，$m \le 1000$），代表景区的个数和道路的条数。接下来的 m 行每行包括 3 个整数 a，b，c，代表 a 和 b 之间有一条通路，并且需要花费 c 元（$c \le 100$）。

【输出形式】
对于每个测试用例，如果能找到这样一条路线的话，输出花费的最小值。如果找不到的话，输出”It's impossible.”。
【样例输入】
3 3
1 2 1
2 3 1
1 3 1
【样例输出】
3
【样例说明】
输入的数据包含 3 个景区和 3 条道路，可以找到一条旅游路线，其最小代价为 3。测试数据存放在 in.txt 文件中。
【评分标准】
该题目有 10 个测试用例，每通过一个测试得 10 分。

答案：

```cpp
#include<iostream>
#include <fstream>
#include<algorithm>
using namespace std;
#define INF 0x3f3f3f3f
typedef long long LL;
int n,m;
LL mat[105][105];            //邻接矩阵数组
LL A[105][105];             //存放两顶点之间最短路径长度的A数组
LL Floyd()              //Floyd求最小环长度
{ LL ans=INF;
    for(int k=1;k<=n;k++)
    { for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                if(i!=k && j!=k && i!=j)       //保证i,j,k不相同，环中至少3个顶点
                    ans=min(ans,A[j][i]+mat[i][k]+mat[k][j]); //求最小环长
        for(int i=1;i<=n;i++)          //更新A[i][j]
            for(int j=1;j<=n;j++)
                A[i][j]=min(A[i][j],A[i][k]+A[k][j]);
    }
    return ans;
}
int main()
{
```

```
25      ifstream inFile;
26      inFile.open("in.txt", ios::in);
27      if(!inFile){
28          cout << "error open in.txt!" << endl;
29          return 1;
30      }
31      if(!inFile.eof())
32      {
33          inFile >> n >> m;
34          for(int i=1;i<=n;i++)          //mat和A初始化
35              for(int j=1;j<=n;j++)
36                  A[i][j]=mat[i][j]=INF;
37          int a,b;
38          LL c;
39          for(int i=1;i<=m;i++)
40          { inFile >> a >> b >> c;
41              mat[a][b]=min(mat[a][b],c);       //取最小值
42              mat[b][a]=mat[a][b];
43              A[a][b]=A[b][a]=mat[a][b];
44          }
45          LL ans=Floyd();
46          if (ans==INF)
47              printf("It's impossible.\n");
48          else
49              printf("%lld\n",ans);
50      }
51      inFile.close();
52      return 0;
53  }
```

解析：Floyd 算法主要是用中间点更新其余点之间的最短路的，每当一个点作为中间承接点 k 更新时，最短路径上就会有一个点被安放在指定的位置，直到最短路径上所有点都归位。

而 Floyd 并没有避免掉“自环”、“二元环”这两种情况，所以我们要手动避免 Floyd 更新出环的情况。避免的方式就是当且仅当 $i \neq j, j \neq k$ 时更新，这样就一定不会更新出小于三个点的环了。此外也要排除自环的情况，这样最后每个点的 $A[i][i]$ 就是题目所求“环路”的结果了。

习题 5 图最短路径算法

【问题描述】

给定 n 个村庄，如果村庄 i 与 j 之间有路联通，则将 i，j 之间连上边，边的权值 W_{ij} 表示这条路的长度。

现在选定一个村庄建医院，设计一个算法求出该医院应该建在哪个村庄，才能

使距离医院最远的村庄到医院的路程达到最短。

【输入形式】

第一行输入村庄个数 N（$3 < N < 10$）

之后输入邻接矩阵，如果 i，j 之间没有直接的通路，W_{ij} 为 0

W_{11}，W_{12}，$\ldots W_{1n}$

$\ldots$

W_{n1}，W_{n2}，$\ldots W_{nn}$

【输出形式】

输出选定的村庄的序号

【样例输入】

```
4
0 3 4 0
3 0 0 2
4 0 0 1
0 2 1 0
```

【样例输出】

```
2
```

答案：

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int A[15][15];
int main()
{

    ifstream inFile;
    inFile.open("in.txt", ios::in);
    if(!inFile){
        cout << "error open in.txt!" << endl;
        return 1;
    }
    if(!inFile.eof())
    {
        int n;
        inFile >> n;
        for(int i=1;i<=n;i++)          //A输入的初始化
            for(int j=1;j<=n;j++)
            {
                inFile>>A[i][j];
```

```
                if(A[i][j]==0)       //说明不连通
                    A[i][j]=1e9;
            }
        for(int k=1;k<=n;k++)        //Floyd算法
            for(int i=1;i<=n;i++)
                for(int j=1;j<=n;j++)
                    if(A[i][j]>A[i][k]+A[k][j])
                        A[i][j]=A[i][k]+A[k][j];
        int mn=1e9,mnn=0;//记录全局最大值中的最小值
        for(int i=1;i<=n;i++)
        {
            int mx=0;
            for(int j=1;j<=n;j++)
                if(A[i][j]>mx)
                    mx=A[i][j];
            if(mx<mn)
            {
                mn=mx;
                mnn=i;
            }
        }
        cout<<mnn;
    }
    inFile.close();
}
```

解析：最简单的做法仍然是 Floyd，因为它可以找到任意两点之间的距离，此时只需要判断到这个点最远的村庄距离，并进行比较就可以了。

对于 $\max_{1\le j\le n} F[i][j]$，要找到使这个值最小的 i，就是本题的答案。

Dijkstra 和 Bellman-Ford 等算法也可以做，但是稍显麻烦。每次仍然是对每个点求单源最短路，并记录最大值和更新。

总结

题目：图

日期：2024 年 6 月 18 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：对 Dijkstra 的概念掌握，概念和代码并重，都要能够理解。

习题 2：根据拓扑排序的原理，进行排除和判断。

习题 3：推箱子是一道有一定难度的复杂题目，以 BFS 的思路为基础，记录路径的长度和更新。题目主要障碍在于判断是否能够通行。有同学代码存在雷同，给这部分同学本题扣掉了一半的分数。

习题 4：考察对 Floyd 算法的应用和改进，不要只局限于传统的算法，要理解其本质。

习题 5：本题重点在于计算出最短路之后如何对其合理应用，并高效处理计算结果。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。