作业 HW4 实验报告

姓名：郑博远 学号：2154312 日期：2022 年 11 月 17 日

1. 涉及数据结构和相关背景

本次作业涉及到的数据结构为图。图 (graph) 是一个二元组。其中 $V(G)$ 是非空集，称为点集 (vertex set)，对于 V 中的每个元素，称其为顶点 (vertex) 或节点 (node)，简称点；$E(G)$ 为 $V(G)$ 各结点之间边的集合，称为边集 (edge set)。常用 $G = (V, E)$ 表示图。当 V, E 都是有限集合时，称 G 为有限图。当 V 或 E 是无限集合时，称 G 为无限图。图有多种，包括无向图 (undirected graph)，有向图 (directed graph)，混合图 (mixed graph) 等。

若 G 为无向图，则 E 中的每个元素为一个无序二元组 (u, v)，称作无向边 (undirected edge)，简称边 (edge)，其中 $u, v \in V$。设 $e = (u, v)$，则 u 和 v 称为 e 的端点 (endpoint)。

若 G 为有向图，则 E 中的每一个元素为一个有序二元组 (u, v)，有时也写作 $u \to v$，称作有向边 (directed edge) 或弧 (arc)，在不引起混淆的情况下也可以称作边 (edge)。设 $e = u \to v$，则此时 u 称为 e 的起点 (tail)，v 称为 e 的终点 (head)，起点和终点也称为 e 的端点 (endpoint)。并称 u 是 v 的直接前驱，v 是 u 的直接后继。

本次实验涉及到用邻接矩阵与邻接表的方式存储图，用深度优先搜索 (DFS) 和广度优先搜索 (BFS) 遍历图，以及普里姆算法求最小生成树。在应用上，本次作业涉及到了拓扑排序、AOV-网、AOE-网来解决问题。同时，小马吃草一题涉及到优先队列优化的 Djikstra 算法求最短路径。

2. 实验内容

2.1 图的遍历

2.1.1 问题描述

本题给定一个无向图，用邻接表作存储结构，用深度优先搜索（DFS）和广度优先搜索（BFS）找出图的所有连通子集。

所有顶点用 0 到 n-1 表示，搜索时总是从编号最小的顶点出发。使用邻接矩阵存储，或者邻接表（使用邻接表时需要使用尾插法）。

2.1.2 基本要求

第 1 行输出 DFS 的结果；

第 2 行输出 BFS 的结果。

连通子集输出格式为{v_{11} v_{12} ...} {v_{21} v_{22} ..}... 连通子集内元素之间用空格分割，子集之间无空格，'{' 和子集内第一个数字之间、'}' 和子集内最后一个元素之间、子集之间均无空格。

2.1.3 数据结构设计

（本题中使用邻接表存储图）

```
struct ArcNode{
    int adjvex;          //该弧的终点
    ArcNode* nextarc;    //后继的弧
    //本题中边无权重
};

typedef struct VNode {
    //数组下标当作结点名称
    bool vis = false;
    ArcNode* firstarc = NULL;
}* AdjList;

struct ALGraph {
    AdjList vertices;    //存储所有的结点
```

```
    int vexnum, arcnum;
};
```

2.1.4 功能说明（函数、类）

```
/**
 * @brief 在图中添加边
 * @param graph 操作的图
 * @param src 边的起点
 * @param dst 边的终点
*/
void AddArc(ALGraph& graph, int src, int dst)


/**
 * @brief 深度优先搜索遍历图
 * @param graph 遍历的图
 * @param vex 当前结点
*/
void DFS(ALGraph& graph, int vex)


/**
 * @brief 广度优先搜索遍历图
 * @param graph 遍历的图
 * @param vex 当前结点
*/
void BFS(ALGraph& graph, int vex)
```

2.1.5 调试分析

作为本次作业的第一题，本题考查对图的两种遍历方式，题目难度比较低。在调试过程中，出现了两个小问题：

1. 向添加边、深度优先搜索、广度优先搜索函数中传参时，graph 变量忘记传递引用，导致所作出的修改并未被记录。使用引用即可；

2. 查找子集时遍历每一个结点，若其还未被访问则进入 DFS 开始搜索。由于 DFS 函数的写法是在调用之前将 vis 置 1 而非进入函数后置 1，因此在首次次调用进入 DFS 之前，要额外把该节点 vis 设为 true，否则会再次搜索到该位置。

2.1.6 总结和体会

在本题中，我体会了图最基础的遍历——广度优先搜索（BFS）与深度优先搜索（DFS），并且使用邻接表这一数据结构来进行存储。建图时，为了让输出的集合顺序与题目给定相同，采用了尾插而非头插法。若设图中点数为 n，边数为 m：则 DFS 与 BFS 的时间复杂度均为 O(n+m)；因为要额外用 vis 变量标记是否访问过，所以两种算法空间复杂度均为 O(n)。本题较基础，我完成得比较顺利。

2.2 小世界现象

2.2.1 问题描述

六度空间理论又称小世界理论。理论通俗地解释为："你和世界上任何一个陌生人之间所间隔的人不会超过 6 个人，也就是说，最多通过五个人你就能够认识任何一个陌生人。"

假如给你一个社交网络图，请你对每个节点计算符合"六度空间"理论的结点占结点总数的百分比。

说明：由于浮点数精度不同导致结果有误差，请按 float 计算。

2.2.2 基本要求

输入：第 1 行给出两个正整数，分别表示社交网络图的结点数 N（1<N≤2000，表示人数）、边数 M（≤33×N，表示社交关系数）。

随后的 M 行对应 M 条边，每行给出一对正整数，分别是该条边直接连通的两个结点的编号（节点从 1 到 N 编号）。

输出：对每个结点输出与该结点距离不超过 6 的结点数占结点总数的百分比，精确到小数点后 2 位。每个结节点输出一行，格式为"结点编号:（空格）百分比%"。

2.2.3 数据结构设计

（本题中使用邻接表存储图）

```
struct ArcNode{
    int adjvex;          //该弧的终点
    ArcNode* nextarc;    //后继的弧
    //本题中边无权重
};

typedef struct VNode {
    //数组下标当作结点名称
    bool vis = false;
    ArcNode* firstarc = NULL;
}* AdjList;

struct ALGraph {
    AdjList vertices;    //存储所有的结点
    int vexnum, arcnum;
};
```

2.2.4 功能说明（函数、类）

```
/**
 * @brief 在图中添加边
 * @param graph 操作的图
 * @param src 边的起点
 * @param dst 边的终点
*/
void AddArc(ALGraph& graph, int src, int dst)



/**
 * @brief 广度优先搜索遍历，统计距vex不超过6的点
 * @param graph 遍历的图
 * @param vex 当前结点
*/
void BFS(ALGraph& graph, int vex)
```

2.2.5 调试分析

起初错误地使用深度优先搜索 DFS 来统计距离某点 vex 距离不超过 6 的点。

访问过的每个点都被标记，这会导致距离计算的错误。如下方图 1 所示，在计算结点 1 的“六度空间”时，由结点 2 开始深度优先搜索直至结点 8 超过距离 6 返回，此时结点 8 被打上 vis 标记。当遍历到结点 1 的下一个相邻结点 8 时，由于结点 8 的 vis 已经置 false，其不会被记入结点 1 的“六度空间”内。

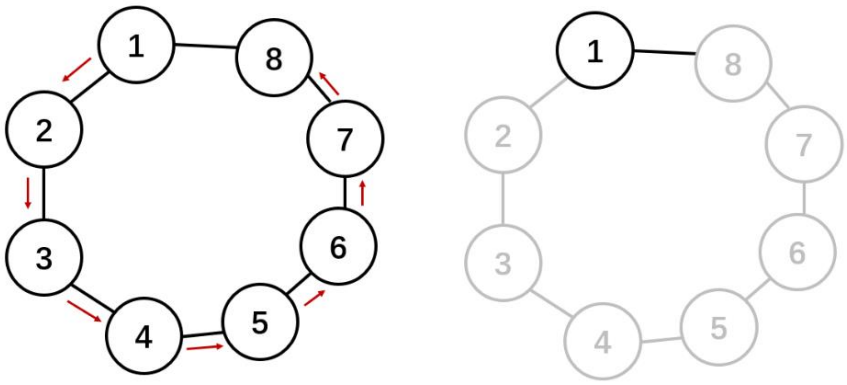

图 1　使用 DFS 导致结点 8 被错判距离

实际上，本题的场景十分符合广度优先搜索的特点。由于 BFS 每次都尝试访问同一层的节点，如果同一层都访问完了再访问下一层；因此，BFS 算法找到的路径是从起点开始的最短合法路径。换言之，这条路径所包含的边数最小。也就是说 b 遍历到的结点顺序即为其到当前正在分析的 vex 的距离，当遍历的深度大于 6 时，则之后的点距离均大于 6，可以直接结束遍历。

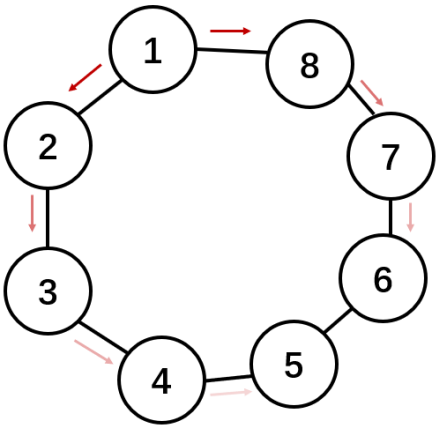

图 2　使用 BFS 遍历图

2.2.6 总结和体会

本题在图的建立与图的遍历方式上与第一小题图的遍历无差，均使用了邻接表方式存储图、BFS 广度优先搜索方式遍历图。在本题中我主要体会了 BFS 与 DFS 两种算法的不同特点：DFS 每次都尝试向更深的节点走，BFS 每次都尝试访问同一层的节点，如果同一层都访问完了再访问下一层。因此，BFS 算法找到的路径是从起点开始的最短合法路径，即路径所包含的边数最小。对于本题，使用广度优先搜索时，根据到起点的边数关系依次进行遍历，这样既不会出现使用 DFS 时误判的情况，也可以在距离大于 6 时直接结束遍历。若设图中点数为 n，边数为 m：则 BFS 的时间复杂度为 O(n+m)；因为要额外用 vis 变量标记是否访问过，所以 BFS 算法空间复杂度为 O(n)。

2.3 村村通

2.3.1 问题描述

N 个村庄，从 1 到 N 编号，现在请你修建一些路使得任何两个村庄都彼此连通。我们称两个村庄 A 和 B 是连通的，当且仅当在 A 和 B 之间存在一条路，或者存在一个村庄 C，使得 A 和 C 之间有一条路，并且 C 和 B 是连通的。

已知在一些村庄之间已经有了一些路，您的工作是再兴建一些路，使得所有的村庄都是连通的，并且新建的路的长度是最小的。

2.3.2 基本要求

输入：第一行包含一个整数 n（3<=n<=100），表示村庄数目。

接下来 n 行,每行 n 个非负整数，表示村庄 i 和村庄 j 之间的距离。距离值在[1,1000]之间。

接着是一个整数 m，后面给出 m 行，每行包含两个整数 a,b,(1<=a<b),表示

在村庄 a 和 b 之间已经修建了路。

输出：输出一行，仅有一个整数，表示为使所有的村庄连通，要新建公路的长度的最小值。

2.3.3 数据结构设计

（本题中使用邻接矩阵存储图）

```
#define MAX_VEX_NUM 100
int graph[MAX_VEX_NUM][MAX_VEX_NUM];   //邻接矩阵

//普里姆算法的辅助数组
struct closedge{
    int adjvex;       //到当前结点最短的V-U中的结点
    int lowcost;      //连接边的权，当前加入V-U的代价
};
```

2.3.4 功能说明（函数、类）

```
/**
 * @brief 求U中距离V-U最小的结点下标
 * @param ce 存放V-U结点信息的数组
 * @return U中距离V-U最小的结点下标
*/
int minimum(closedge* ce)

/**
 * @brief 普里姆算法求最小生成树
 * @param graph 存放图的邻接矩阵
 * @param n 点的数量
*/
void MiniSpanTree_PRIM(int graph[MAX_VEX_NUM][MAX_VEX_NUM], int n)
```

2.3.5 调试分析

本题完成地比较顺利，调试时只遇到了一个小问题。

题目给定了一些已经建好了的两村庄之间的路。在往邻接矩阵添加题目给定路时（如村庄 i、j 已有路，则 graph[i][j] = 0），我忘记了本题中的图是无向图，要将双向的边 graph[i][j]与 graph[j][i]都置零。修改后即通过本题。

2.3.6 总结和体会

本题中由于数据量比较小，n_{max}仅到 100 的数量级，我使用邻接矩阵的方式来存储本题中的图结构。本题是最小生成树算法的运用；在建立最小生成树的过程中，我选择用普里姆算法建树。在不使用二叉堆优化的情况下，其时间复杂度为 $O(n^2+m)$（图中结点数为 n，边数为 m）。需要注意的是，题目给定了一些建好的路径，而他们所连接的点有可能在初始状态属于同一连通分量，也可能属于不同连通分量。经过思考，我决定将书本中加入 U 的点 lowcost 变为 0 改为 lowcost 变为-1，而将题目给定的联通两村庄的路边权记为 0。这样在建树的过程中，这些路一定会被加入到最终的 U 集合，且不会增加总 cost。如下方图 3 所示，是对题目样例输入的修改后普里姆算法示意图。

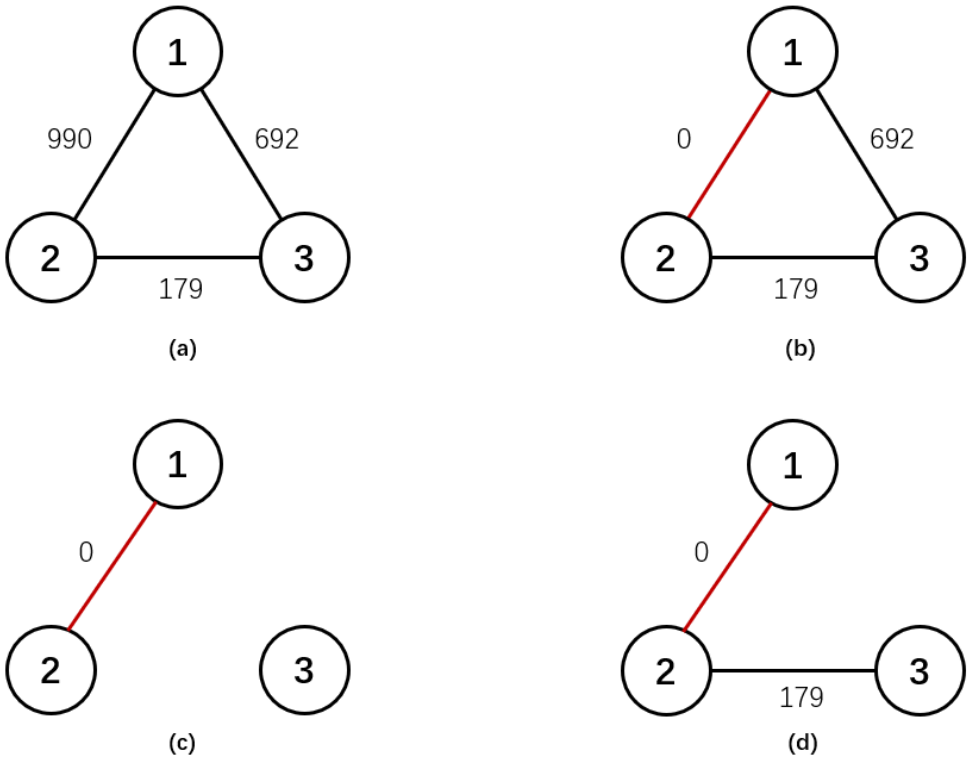

图 3　对题目样例输入的普里姆算法示意图

2.4 给定条件下构造矩阵

2.4.1 问题描述

给你一个正整数 k ，同时给你：

一个大小为 n 的二维整数数组 rowConditions ，其中 rowConditions[i] = [abovei, belowi] 和一个大小为 m 的二维整数数组 colConditions ，其中 colConditions[i] = [lefti, righti] 。两个数组里的整数都是 1 到 k 之间的数字。

你需要构造一个 k x k 的矩阵，1 到 k 每个数字需要恰好出现一次。剩余的数字都是 0 。矩阵还需要满足以下条件：

对于所有 0 到 n - 1 之间的下标 i ，数字 abovei 所在的行必须在数字 belowi 所在行的上面。

对于所有 0 到 m - 1 之间的下标 i ，数字 lefti 所在的列必须在数字 righti 所在列的左边。

返回满足上述要求的矩阵，题目保证若矩阵存在则一定唯一；如果不存在答案，返回一个空的矩阵。

2.4.2 基本要求

输入：第一行包含 3 个整数 k、n 和 m。

接下来 n 行，每行两个整数 abovei、belowi，描述 rowConditions 数组；

接下来 m 行，每行两个整数 lefti、righti，描述 colConditions 数组。

输出：如果可以构造矩阵，打印矩阵；否则输出-1。

矩阵中每行元素使用空格分隔。

2.4.3 数据结构设计

```
struct ArcNode {
    int adjvex;              //该弧的终点
```

```
    ArcNode* nextarc;    //后继的弧
    //本题中边无权重
};

typedef struct VNode {
    int in_degree = 0;       //额外存放入度
    ArcNode* firstarc = NULL;
}*AdjList;

struct ALGraph {
    AdjList vertices;    //存放图的结点
    int vexnum, arcnum;
};
```

2.4.4 功能说明（函数、类）

```
/**
 * @brief 在图中添加边
 * @param graph 操作的图
 * @param src 边的起点
 * @param dst 边的终点
*/
void AddArc(ALGraph& graph, int src, int dst)


/**
 * @brief 拓扑排序
 * @param G 欲排序的图
 * @param arr 排序序列存储的数组
 * @return 是否可以拓扑排序
*/
bool TopologicalSort(ALGraph& G, int* arr)
```

2.4.5 调试分析

我将横纵的关系分别建图进行两次拓扑排序，将排序的结果存入两个一维数组 row、col 之中（在每个结点对应下标位置存储其序号值）。然后，再开一个二维数组存储最后的矩阵，即 map[row[i]][col[i]] = i。最后将矩阵逐行输出即可。但是在开二维数组时，我忽略了下标从 1 开始；导致当数据大小为 400 时会产生数组越界。将数组再开大一些，或使用动态内存申请即可解决问题。

2.4.6 总结和体会

本题中，我主要练习了拓扑排序。在给定一系列偏序关系的前提下，其目标是将所有结点排序，使得排在前面的结点不能依赖于排在后面的结点。初始状态下，栈 S 装着所有入度为 0 的点，L 是一个空列表。每次从 S 中取出一个点，然后将其相连的所有边删除，并减少另一端结点的入度。不断重复以上过程，直到栈 S 为空。检查图中是否存在任何边，如果有则这个图一定有环路，否则 L 中顶点的顺序就是拓扑排序的结果。对于有 n 个顶点和 e 条弧的有向图而言，求各个顶点入度时间复杂度为 O(e)，建立入度顶点栈的时间复杂度为 O(n)；排序过程中每个结点进出栈各一次，入度减 1 操作执行 e 次，总的时间复杂度为 O(n+e)。

2.5 必修课

2.5.1 问题描述

某校的计算机系有 n 门必修课程。学生需要修完所有必修课程才能毕业。

每门课程都需要一定的学时去完成。有些课程有前置课程，需要先修完它们才能修这些课程；而其他课程没有。 不同于大多数学校，学生可以在任何时候进行选课，且同时选课的数量没有限制。

现在校方想要知道：

从入学开始，每门课程最早可能完成的时间（单位：学时）；

对每一门课程，将该课程的学时增加 1 是否会延长入学到毕业的最短时间。

2.5.2 基本要求

输入：第一行，一个正整数 n，代表课程的数量。

接下来 n 行，每行若干个整数：

- 第一个整数为 t_i ，表示修完该课程所需的学时。

・第二个整数为 c_i，表示该课程的前置课程数量。

・接下来 c_i 个互不相同的整数，表示该课程的前置课程的编号。

该校保证，每名入学的学生，一定能够在有限的时间内毕业。

输出：输出共 n 行，第 i 行包含两个整数：

・第一个整数表示编号为 i 的课程最早可能完成的时间。

・第二个整数表示，如果将该课程的学时增加 1，入学到毕业的最短时间是否会增加。如果会增加则输出 1，否则输出 0。

每行的两个整数以一个空格隔开。

2.5.3 数据结构设计

```
struct ArcNode {
    int adjvex;          //该弧的终点
    ArcNode* nextarc;    //后继的弧
};

typedef struct VNode {
    int in_degree = 0;
    int val;                 //所需学时
    int max_preval = 0;      //前驱结点的累计最长时间
    vector<int> pre;         //影响该结点的前驱结点(可能多个相同)
    ArcNode* firstarc = NULL;
}*AdjList;

struct ALGraph {
    AdjList vertices;
    int vexnum, arcnum;
};
```

2.5.4 功能说明（函数、类）

```
/**
 * @brief 在图中添加边
 * @param graph 操作的图
 * @param src 边的起点
 * @param dst 边的终点
*/
void AddArc(ALGraph& graph, int src, int dst)
```

```
/**
 * @brief 拓扑排序
 * @param G 欲排序的图
 * @return 是否可以拓扑排序
*/
bool TopologicalSort(ALGraph& G)

/**
 * @brief 深搜求某结点是否是关键结点
 * @param graph 搜索的图
 * @param cur 当前所在结点
 * @param tgt 欲搜索的节点
 * @return 是否是影响总时长的关键结点
*/
bool dfs_relevance(ALGraph graph, int cur, int tgt)
```

2.5.5 调试分析

总体编写很顺利，除了起初忘记更新影响下一结点的前驱最大值以外没有遇到其他问题，很迅速地通过了。

2.5.6 总结和体会

本题的关键在于求关键路径，即对最早毕业时间有影响的路径和路径上的结点。由于完成本体时课上还未介绍到关键路径 AOE-网，我没有用边来存储课程的课时，而是采用对 AOV-网进行改造的方式，用结点来存储课程的课时，具体方法如下：每个结点代表一门课程，记录其学时。同时用一个变量维护这门课程开始前所需的最长时间，一个 vector 存储提供这个最长时间的前驱结点（可能有多个，对应多个关键路径）。将所有出度为零的结点（不是其他课程的前驱课程）连接到同一个汇点上。拓扑排序遍历之后，其实可以视作建立了一个以汇点为根节点的树。树上所有点都是关键活动，对最终的毕业时间均有所影响。因此遍历这颗树，其中存在的结点即是影响最终最早毕业时间的关键节点。学习了 AOE-

网之后，我也了解到了在拓扑排序、逆拓扑排序更新最早开始时间与最迟开始时间的方法来求关键活动。

2.6 小马吃草

2.6.1 问题描述

假设无向图 G 上有 N 个点和 M 条边，点编号为 1 到 N，第 i 条边长度为 w_i，其中 H 个点上有可以食用的牧草。另外有 R 匹小马，第 j 匹小马位于点 start_j，需要先前往任意一个有牧草的点进食牧草，然后前往点 end_j，请你计算每一匹小马需要走过的最短距离。

2.6.2 基本要求

输入：第一行两个整数 N、M，分别表示点和边的数量。

接下来 M 行，第 i 行包含三个整数 x_i,y_i,w_i，表示从点 x_i 到点 y_i 有一条长度为 w_i 的边，保证 x_i≠y_i。

接下来一行有两个整数 H 和 R，分别表示有牧草的点数量和小马的数量。

接下来一行包含 H 个整数，为 H 个有牧草的点编号。

接下来 R 行，第 j 行包含两个整数 start_j 和 end_j，表示第 j 匹小马起始位置和终点位置。

题目保证两个点之间一定是连通的，并且至少有一个点上有牧草

输出：输出共 R 行，表示 R 匹小马需要走过的最短距离。

2.6.3 数据结构设计

```
#define MAX_NODE_NUM 1005
int map[MAX_NODE_NUM][MAX_NODE_NUM];  //存储两点最短距离的矩阵
int vis[MAX_NODE_NUM][MAX_NODE_NUM];  //存储该点是否访问的矩阵

struct ArcNode {
```

```
    int adjvex;          //该弧的终点
    ArcNode* nextarc;    //后继的弧
    int val;             //边权
};

typedef struct VNode {
    //本题数组下标就当作是结点名称
    bool vis = false;
    ArcNode* firstarc = NULL;
}*AdjList;

struct ALGraph {
    AdjList vertices;
    int vexnum, arcnum;
};
```

2.6.4 功能说明（函数、类）

```
/**
 * @brief 在图中添加边
 * @param graph 操作的图
 * @param src 起点
 * @param dst 终点
 * @param val 边权
*/
void AddArc(ALGraph& graph, int src, int dst, int val)

/**
 * @brief Dijkstra求两点最短路
 * @param graph 图
 * @param dis 存放最终dis的数组
 * @param vis 存放是否访问过的数组
 * @param s 所求和其他点距离的某节点
*/
void dijkstra(ALGraph graph, int* dis, int* vis, int s)
```

2.6.5 调试分析

1. 起初使用未被优化的 Dijkstra 算法，对于每一颗草求它到其他点的最短距离，时间复杂度为 $O(n^2)$，因此求图中所有草时间复杂度为 $O(n^3)$，这会导致 TLE。使用优先队列优化之后，对于每一个点求到其他点最短距离时间复杂度为

O(nlogn)，总的时间复杂度为 $O(n^2logn)$，即可通过本题；

2. 优先队列默认是队首元素最大，而 Dijkstra 每次要取出的是当前 dis 最小的元素。因此在运算符重载时，要重载相反的符号。起初忽略了这一点，导致了程序计算结果的错误，更正后即可。

2.6.6 总结和体会

```
void dijkstra(ALGraph graph, int* dis, int* vis, int s)
{
    memset(vis, 0, MAX_NODE_NUM * sizeof(int));
    memset(dis, 0x3f, MAX_NODE_NUM * sizeof(int));
    dis[s] = 0;
    priority_queue<node> q;
    q.push({ 0, s });
    while (!q.empty()) {
        int u = q.top().vex;
        q.pop();
        if (vis[u])
            continue;
        vis[u] = true;
        for (ArcNode* p = graph.vertices[u].firstarc; p; p = p->nextarc) {
            if (dis[p->adjvex] > dis[u] + p->val) {
                dis[p->adjvex] = dis[u] + p->val;
                q.push({ dis[p->adjvex], p->adjvex });
            }
        }
    }
}
```

图 4 使用 priority_queue 优化的 Dijkstra 算法

本题考察了最短路径的求法。若使用 Floyd 算法，可以之间求出图中任意两个结点之间的最短路，最后再遍历一次所有草丛即可得到小马吃草最短的路径。但 Floyd 算法的时间复杂度为 $O(n^3)$，在 10 的 3 次方数量级的数据量下会 TLE。我的想法是使用 Dijkstra 计算每一个草丛到其他所有点的最短路（开两个二维数组 dis 与 vis，其中第一个维度表示不同的草丛，每一个一维数组都相当于单个 Dijikstra 算法的 dis 与 vis 数组）。Dijkstra 算法将结点分成两个集合：已确定最短路长度的点集（记为 S 集合）的和未确定最短路长度的点集（记为 T

集合)。初始化 dis[s] = 0（s 为当前正在求最短路径的草丛），其他点的 dis 均为∞（此处记作 0x3f3f3f3f)。然后重复操作：从 T 集合中，选取一个最短路长度最小的结点，移到 S 集合中。对那些刚刚被加入 S 集合的结点的所有出边执行松弛操作。直到 T 集合为空，算法结束。通过优先队列优化之后，时间复杂度能够达到 $O(n^2logn)$，即可通过本题。

3. 实验总结

这次作业中，我巩固了对图这一重要数据结构的掌握。在图的存储方面，我在不同题目中使用了邻接矩阵、邻接表两种方式来存储，体会了二者的区别与各自的优缺点。在图的遍历上，使用了深度优先搜素（DFS）和广度优先搜索（BFS），并在“六度空间”一题中体会了广度优先搜索算法的特点。在“村村通”一题中，我体会了使用普里姆算法实现建立的最小生成树。

在有向无环图的应用方面，我在构造矩阵一题中巩固了对 AOV-网以及拓扑排序的掌握；同时对 AOV-网进行改造，通过自己的思考解决了“必修课”一题。在课上学习 AOE-网后，我也思考了通过 AOE-网来解决该题的方法，了解了关键活动、关键路径的求法。求最短路径方面，我在“小马吃草”一题中重温了 Floyd 与 Dijikstra 算法求最短路，并通过优先队列的方式优化时间复杂度至 O(nlogn)。

总体来说，本次作业中我巩固了对已有的图的存储、遍历、求最短路径等方面知识的掌握；同时，我也学习了 AOV-网、AOE-网的相关应用，收获颇丰。我希望通过这次作业的练习，在未来能更加熟练掌握使用已学过的这些数据结构。