作业 HW4 实验报告

姓名　　学号　　日期：2024 年 12 月 1 日

1. 涉及数据结构和相关背景

本次实验内容主要围绕图这一数据结构展开。

题目涉及了图的构建（邻接矩阵和邻接表，我大多采用邻接表的结构）、图的遍历（bfs 和 dfs）、最短路径算法（比如 Dijkstra 算法）、最小生成树算法（比如 Prim 算法）等内容。

2. 实验内容

2.1 图的遍历

2.1.1 问题描述

给定一个无向图，用 dfs 和 bfs 找出图的所有连通分量。

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

输入：第 1 行输入 2 个整数 n m，分别表示顶点数和边数，空格分割；后面 m 行，每行输入边的两个顶点编号，空格分割。

输出：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{v11 v12 ...}{v21 v22 ..}... 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

对于 20%的数据，有 0<n<=15；对于 40%的数据，有 0<n<=100；对于 100%的数据，有 0<n<=1000；对于所有数据，0.5n<=m<=1.5n，不考虑输入错误。

2.1.3 数据结构设计

```
struct arcnode { //弧（arc）结点
    int adjvex;             //邻结点域 /nextarc下一条弧 info数据域存权值，本题无权值
    arcnode* nextarc;
};

typedef struct vertex {//vertex结点--数组下标就是结点名称
    bool vis = 0;  //标记是否被访问
    arcnode* firstarc = NULL;//当前顶点出发的第一条弧
}*AList;

struct ALGraph {//采用邻接表形式构建图
    AList vertices;//vertices变量被声明为一个alist指针类型的数组--顶点用一个一维数组存储
    int vexnum, arcnum; //顶点总数vexnum；弧总数arcnum
};
```

2.1.4 功能说明（函数、类）

```
//函数功能--构建每个顶点对应的单链表
void goujianlist(ALGraph& graph, int src, int dst)
{
    arcnode* v1 = graph.vertices[src].firstarc;// 获取起始顶点 src 的第一个邻接点链表头指针
    arcnode* v2 = new(nothrow) arcnode;// 为新边节点分配内存

    if (v1不是NULL) {
        while (v1->nextarc)
            v1 = v1->nextarc;// 找到链表的最后一个节点
        v1->nextarc = v2;// 将新节点 v2 添加到链表的末尾
    }
    else// 如果起始顶点 src 还没有邻接点链表
        graph.vertices[src].firstarc = v2; // 将新节点 v2 设为链表的头节点

    v2->nextarc = NULL;// 设置新节点 v2 的 nextarc 为 NULL
    return;
}

//函数功能--深度优先搜索算法
void dfs(ALGraph& graph, int vex)
{
    arcnode* v = graph.vertices[vex].firstarc;// 获取顶点 vex 的第一个邻接点

    for (; v; v = v->nextarc) {// 遍历顶点 vex 的所有邻接点
        if (!graph.vertices[v->adjvex].vis) {// 检查当前邻接点是否已被访问过
            graph.vertices[v->adjvex].vis = 1;
            cout << ' ' << v->adjvex;
            dfs(graph, v->adjvex);//递归调用dfs，继续搜索当前邻接点的邻接点
        }
    }
    return;
}

//函数功能--广度优先算法
void bfs(ALGraph& graph, int vex) //bfs肯定要用队列的
{
    arcnode* p = graph.vertices[vex].firstarc;
    queue<int> vexlist; //定义了一个队列名，用于存储待访问的顶点

    while (!vexlist.empty()) {// 当队列不为空时，循环
        int tmp = vexlist.front();
        arcnode* p = graph.vertices[tmp].firstarc;// 获取当前顶点的第一个邻接点
        vexlist.pop();// 从队列中取出一个顶点（出队）
        for (; p; p = p->nextarc) {// 遍历当前顶点的所有邻接点
            if (!graph.vertices[p->adjvex].vis) {// 检查当前邻接点是否已被访问过
                graph.vertices[p->adjvex].vis = 1;// 标记当前邻接点为已访问
                cout << ' ' << p->adjvex;// 输出当前邻接点的编号（或进行其他处理）
                vexlist.push(p->adjvex); // 将当前邻接点加入队列，以便后续访问
            }
        }
    }
}
```

2.1.5 调试分析（遇到的问题和解决方法）

调试过程描述：（本题是课上讲过的基础题目，没什么遇到的问题，在此仅描述调试过程）

1. 整体写完程序后运行，根据编译器报的错误修改程序中的语法错误。
2. 生成可执行文件，根据题目中给出的测试数据生成文件，利用输入输出重定向的方法，检测程序的输出结果是否正确。以此来验证函数逻辑是否正确。

2.1.6 总结和体会

本题涉及了图上的基本操作，即 bfs、dfs 和图的构建，为后续题目的解决奠定了基础。通过本题，我回顾了这些基本操作，加深了我对图的理解和运用。

本题没什么难点和易错点，主要考察的是对基本操作的掌握程度。

2.2 小世界现象

2.2.1 问题描述

给一个社交网络图，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

```
struct arcnode { //弧（arc）结点
    int adjvex;              //邻结点域 /nextarc 下一条弧 info数据域存权值，本题无权值
    arcnode* nextarc;
};

typedef struct vertex {//vertex结点--数组下标就是结点名称
    bool vis = 0;  //标记是否被访问
    arcnode* firstarc = NULL;//当前顶点出发的第一条弧
}*AList;

struct ALGraph {//采用邻接表形式构建图
    AList vertices;//vertices变量被声明为一个alist指针类型的数组--顶点用一个一维数组存储
    int vexnum, arcnum; //顶点总数vexnum；弧总数arcnum
};

struct node {
    int vex;//顶点编号 vex 和深度 depth
    int depth;
    node(int v, int d) {
        vex = v;
        depth = d;
    }
};
```

2.2.4 功能说明（函数、类）

```
//函数功能--构建每个顶点对应的单链表
void goujianlist(ALGraph& graph, int src, int dst)
{
    arcnode* v1 = graph.vertices[src].firstarc;// 获取起始顶点 src 的第一个邻接点链表头指针
    arcnode* v2 = new(nothrow) arcnode;// 为新边节点分配内存

    if (v1不是NULL) {
        while (v1->nextarc)
            v1 = v1->nextarc;// 找到链表的最后一个节点
        v1->nextarc = v2;// 将新节点 v2 添加到链表的末尾
    }
    else// 如果起始顶点 src 还没有邻接点链表
        graph.vertices[src].firstarc = v2; // 将新节点 v2 设为链表的头节点

    v2->nextarc = NULL;// 设置新节点 v2 的 nextarc 为 NULL
    return;
}
```

```
void bfs(ALGraph& graph, int vex) //bfs肯定要用队列的
{
    arcnode* p = graph.vertices[vex].firstarc;
    queue<node> vexlist; //定义了一个队列，用于存储待访问的顶点
    vexlist.push(node{ vex,0 });
    while (!vexlist.empty()) {// 当队列不为空时，循环
        node tmp = vexlist.front();// 获取当前顶点的第一个邻接点
        vexlist.pop();
        //基本的bfs搜索，已在前一题中解释
        arcnode* p = graph.vertices[tmp.vex].firstarc;
        for (; p; p = p->nextarc) {
            if (!graph.vertices[p->adjvex].vis) {
                graph.vertices[p->adjvex].vis = 1;
                ++cnt;
                vexlist.push(node(p->adjvex, tmp.depth + 1));
            }
        }

        if (tmp.depth >= 6) {//如果当前深度大于或等于 6，则停止搜索
            break;
        }
    }
}
```

2.2.5 调试分析（遇到的问题和解决方法）

调试过程：

1. 整体写完程序后运行，根据编译器报的错误修改程序中的语法错误。
2. 生成可执行文件，根据题目中给出的测试数据生成文件，利用输入输出重定向的方法，检测程序的输出结果是否正确。以此来验证函数逻辑是否正确。

遇到的问题：在程序调试过程中，犯了很“经典”的错误--输入时顶点编号从 1 开始，但程序中顶点数组的下标从 0 开始。　解决方法：通读整个程序，对用到的下标逻辑重新梳理了一遍。

2.2.6 总结和体会

本题在上一道题目 bfs 的基础上，增加了一个 if (tmp.depth >= 6)的判断。如果将这个判断条件挖掘出来，本道题就可以在上一道题的基础上轻松做出。

本题锻炼了我触类旁通的能力，可以在已有题目的基础上快速做出新的题目。题目本身只要理解了题目含义，就没什么难点和易错点了（哦对，注意数组下标的转换，容易出错）。

2.3 村村通

2.3.1 问题描述

N 个村庄，从 1 到 N 编号，现在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。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

```
struct {
    int adjvex; //记录该边依附的在U中的顶点
    int lowcost;//存放在V-U中各个顶点到集合U中的当前最小权值
}closedge[100];
```

2.3.4 功能说明（函数、类）

```
int get_lowcost(int n)
{
    int tmp = 1000; //最假设大路径长度
    int vex = 0;//从顶点0开始，事实上其他也可以
    for (int i = 0; i < n; ++i) {// 遍历所有顶点
        if (closedge[i].lowcost != -1 && closedge[i].lowcost < tmp) {
            tmp = closedge[i].lowcost;
            vex = i;//记录这个顶点，便于加入到U中去
        }
    }
    return vex;//返回具有最小未访问最短路径的顶点的索引

}

//函数功能--利用prim算法求最小生成树的总权重
void prim_min(int n)//prim求最短路径
{
    //closedge数组用于存储到最小生成树中每个顶点的最短边）
    closedge[0].lowcost = -1;//从0号村庄开始，当然也可以设为其他的，但0号最简单
    for (int i = 1; i < n; ++i) {
        closedge[i] = { 0,graph[0][i] };  //得到其他点和0号村庄之间的路径
    }
    int sum = 0;
    for (int i = 1; i < n; ++i) {// 循环，构建最小生成树
        int j = get_lowcost(n);//调用 get_lowcost(n) 函数，返回具有最小未访问最短路径的顶点的索引，存储在 j 中
        sum = sum + closedge[j].lowcost;
        closedge[j].lowcost = -1;//// 标记该顶点已包含在最小生成树中

        //更新点集
        for (int m = 0; m < n; ++m) {
            if (graph[j][m] < closedge[m].lowcost) {
                closedge[m] = { j,graph[j][m] };
            }
        }
    }
```

2.3.5 调试分析（遇到的问题和解决方法）

调试过程：

1. 整体写完程序后运行，根据编译器报的错误修改程序中的语法错误。
2. 生成可执行文件，根据题目中给出的测试数据生成文件，利用输入输出重定向的方法，检测程序的输出结果是否正确。以此来验证函数逻辑是否正确。

遇到的问题：一开始，我将 int graph[100][100];放在了 main 函数里，然后编译器报了

这样类似的 warning（这是在矩阵源题目里遇到的，和本次遇到了相同的问题）。经过请教高程沈坚老师，得知这是由于在 main 函数里局部变量所占空间太大。（但就算不管这个问题，直接提交到 oj 系统也能 100 分通过。。。）　　解决方法：将此数组开成全局变量即可。（虽然全局变量不推荐使用，但单个程序使用了也没什么毛病。）

2.3.6 总结和体会

通过此题目，让我遇到了编译过程中一个不常见的 warning，丰富了我 debug 的经历。然后此题目考查了 prim 最短路径算法，通过此题，加深了我对 prim 算法的理解。

2.4 给定条件下构造矩阵

2.4.1 问题描述

给一个正整数 k 同时给：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。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。

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 { //弧（arc）结点
    int adjvex;             //邻结点域 /nextarc下一条弧 info数据域存权值，本题无权值
    arcnode* nextarc;
};

typedef struct vertex {//vertex结点--数组下标就是结点名称
    int indegree = 0;  //标记入度结点
    arcnode* firstarc = NULL;//当前顶点出发的第一条弧
}*AList;

struct ALGraph {//采用邻接表形式构建图
    AList vertices;//vertices变量被声明为一个alist指针类型的数组--顶点用一个一维数组存储
    int vexnum, arcnum; //顶点总数vexnum；弧总数arcnum
};
```

2.4.4 功能说明（函数、类）

```
//函数功能--构建每个顶点对应的单链表
void goujianlist(ALGraph& graph, int src, int dst)
{
    arcnode* v1 = graph.vertices[src].firstarc;// 获取起始顶点 src 的第一个邻接点链表头指针
    arcnode* v2 = new(nothrow) arcnode;// 为新边节点分配内存

    if (v1不是NULL) {
        while (v1->nextarc)
            v1 = v1->nextarc;// 找到链表的最后一个节点
        v1->nextarc = v2;// 将新节点 v2 添加到链表的末尾
    }
    else// 如果起始顶点 src 还没有邻接点链表
        graph.vertices[src].firstarc = v2; // 将新节点 v2 设为链表的头节点

    v2->nextarc = NULL;// 设置新节点 v2 的 nextarc 为 NULL
    return;
}
```

```
//函数功能--拓扑排序
int tuopopaixu(ALGraph & G, int* array)
//输入：G - 图的邻接表表示；array - 用于存储拓扑排序结果的数组；
//输出：若能成功进行拓扑排序，则返回1；否则，返回0
{
    int a = 0;// 用于记录已排序的顶点数目
    while (1) {
        int i = -1;
        for (i = 1; i <= G.vexnum; i++)// 查找入度为0的顶点
            if (G.vertices[i].indegree == 0)
                break;

        // 若未找到入度为0的顶点，则跳出循环
        if (i > G.vexnum)
            break;      //没有入度为0的点了

        a++;// 记录找到的顶点，并更新相关顶点的入度
        array[i] = a;

        G.vertices[i].indegree=-1;  //置为-1
        for (arcnode* p = G.vertices[i].firstarc; p; p = p->nextarc)// 更新所有邻接顶点的入度
            G.vertices[p->adjvex].indegree--;
    }
    // 检查是否所有顶点都已排序
    if (a == G.vexnum)
        return 1;
    else
        return 0;
}
```

2.4.5 调试分析（遇到的问题和解决方法）

调试过程：

1. 整体写完程序后运行，根据编译器报的错误修改程序中的语法错误。
2. 生成可执行文件，根据题目中给出的测试数据生成文件，利用输入输出重定向的方法，检测程序的输出结果是否正确。以此来验证函数逻辑是否正确。

遇到的问题：1.首先就是遇到了跟上一道题一样的问题--局部变量占用空间太多.

2.第一版程序写完后，总是出现只能处理一个图的情况。 解决办法：根据和同学探讨程序逻辑，发现是由于顶点下标设置错误，更改后即可。

2.4.6 总结和体会

本题是在数学问题的背景下考察拓扑排序。通过本题，使我加深了对拓扑排序算法的理解，同时提高了用算法解决具体问题的能力。

本题个人认为最大的难点在于对题目的理解。第一遍读完题目，我都不知道要干什么。通过

阅读力扣题解，辅助画图，才发现原来是考察拓扑排序。只要 get 到本题是考察拓扑排序，整个题目的解决思路就非常清晰明了了。

2.5 小马吃草

2.5.1 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，请计算每一匹小马需要走过的最短距离。

2.5.2 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；

对于 100%的数据，1<=N、R<=1000，1<=w_i<=100000；

对于所有数据，N-1<=M<=2N，1<=H<=N。

输出：输出共 R 行，表示 R 匹小马需要走过的最短距离

2.5.3 数据结构设计

```
struct arcnode { //弧（arc）结点
    int adjvex;             //邻结点域 /nextarc下一条弧 info数据域存权值，本题有权值
    arcnode* nextarc;
    int val;//权值
};

typedef struct vertex {//vertex结点--数组下标就是结点名称
    bool vis = 0;  //标记是否被访问
    arcnode* firstarc = NULL;//当前顶点出发的第一条弧
}*AList;

struct ALGraph {//采用邻接表形式构建图
    AList vertices;//vertices变量被声明为一个alist指针类型的数组--顶点用一个一维数组存储
    int vexnum, arcnum; //顶点总数vexnum；弧总数arcnum
};
```

2.5.4 功能说明（函数、类）

```
//函数功能--构建每个顶点对应的单链表
void goujianlist(ALGraph& graph, int src, int dst,int val)
{
    arcnode* v1 = graph.vertices[src].firstarc;
    arcnode* v2 = new(nothrow) arcnode;
    if (!v2)
        exit(-1);
    v2->adjvex = dst;
    //这一步是添加权值，其他均为构建邻接表的基础操作，前面几道题已经阐述过
    v2->val = val;

    if (v1) {
        while (v1->nextarc)
            v1 = v1->nextarc;
        v1->nextarc = v2;
    }
    else
        graph.vertices[src].firstarc = v2;

    v2->nextarc = NULL;
    return;
}
```

```
//函数功能-- dijkstra求带权无向图的最短路径
void dijkstra(ALGraph graph, int* dis, int* vis, int s)
{
    dis[s] = 0;//// 源节点到自身的距离为0
    priority_queue<node> q;
    q.push({ 0, s });
    //Dijkstra算法主要的循环步骤
    while (!q.empty()) {
        int u = q.top().vex;//从 q 中取出距离最小的节点 u（使用q.top()获取，并随后调用q.pop()移除）
        q.pop();
        if (vis[u])
            continue;
        vis[u] = true;
        // 更新节点u的所有邻接节点的最短距离
        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 });
            }
        }
    }
}
```

2.5.5 调试分析（遇到的问题和解决方法）

调试过程：

1. 整体写完程序后运行，根据编译器报的错误修改程序中的语法错误。
2. 生成可执行文件，根据题目中给出的测试数据生成文件，利用输入输出重定向的方法，检测程序的输出结果是否正确。以此来验证函数逻辑是否正确。

遇到的问题：调试过程中，结果总是出错。 解决方法：再次详细检查了程序的每一步后，发现是有两个变量命名冲突了。。。。得到的教训是，当程序稍微长一点的时候，最好每定义一个变量都把变量名稍微记一下，以免出错。

2.5.6 总结和体会

本题和之前题目不同的是，本题是带权值的无向图。因此不再用 prim 算法，而是使用 dijkstra 算法。通过本题，加深了我对无向图带权值和不带权值的理解，提高了我处理图这一数据结构时的能力。

当然本题的难点就是将实际问题和 dijkstra 算法联系起来，并熟悉基本的 dijkstra 算法实现代码。

3. 实验总结

对于本次实验，我作出了以下总结：

1. 首先通过本次的实验，在解决实际问题的基础上，加深了我对图上基本操作的理解，同时也提高了我对这些基本操作进行代码实现优化的能力。
2. 同时通过有限的几道题目能明显感觉到图这一数据结构涉及众多基本的知识和算法，需要再进一步通过做题巩固对这些基础知识的掌握。
3. 在调试过程中遇到了一些没有见过的 warning 问题。通过解决这些问题，个人 debug 的能力也得到了提高。

总之，通过此次实验，我收获良多，但也仍旧存在一些不足，需要课下进一步地理解和巩固对相关知识的理解。