作业 HW4 实验报告

姓名：何正潇 学号：1950095 日期：2021 年 12 月 13 日

1. 涉及数据结构和相关背景

本章涉及的数据结构乃是数据结构中比较重要的图，有着广泛的应用。其中蕴含的数学意义和实际意义比较重要。

2. 实验内容

4.1 图的存储结构

4.1.1 问题描述

图是一种描述多对多关系的数据结构，图中的数据元素称作顶点，具有关系的两个顶点形成的一个二元组称作边或弧，顶点的集合V和关系的集合R构成了图，记作G=（V,R）。图又分成有向图，无向图，有向网，无向网。图的常用存储结构有邻接矩阵、邻接表、十字链表、邻接多重表。图的基本操作包括图的创建、销毁、添加顶点、删除顶点、插入边、删除边、图的遍历。
本题练习邻接矩阵和邻接表的创建。

4.1.2 基本要求

第1行输入一个数字1~4，1为有向图,2为有向网,3为无向图,4为无向网;
第2行输入2个整数n m，分别表示顶点数和边数，空格分割
第3行为n个字符的序列，一个字符表示一个顶点
后面m行，若前面选择的是图，则每行输入边的两个顶点字符，空格分割，若是网，则每行输入弧的两个顶点字符和弧的权值，空格分割

输出

第1行输出顶点向量，顶点字符以一个空格分割
接着n行n列，输出邻接矩阵，每个数字占4位
接着n行，输出邻接表

4.1.3 数据结构设计

```
struct edgeNode
{
    int end;
    int weight;
    edgeNode* next;
    edgeNode(int e, int w, edgeNode* n = NULL)
    {
        end = e;
        weight = w;
        next = n;
    }
};
```

4.1.4 功能说明（函数、类）

```
#define _CRT_SECURE_NO_WARNINGS
#include<string.h>
#include<map>
#include<string>
#include<stack>
#include<iostream>
#include<iomanip>
struct edgeNode
{
int end;
int weight;
edgeNode* next;
edgeNode(int e, int w, edgeNode* n = NULL)
{
end = e;
weight = w;
next = n;
}
};
struct verNode
{
```

```
    char ver;
    edgeNode* head;
    verNode(edgeNode* h = NULL)
    {
    head = h;
    }
    };
    using namespace std;
    int main()
    {
    verNode* list=NULL;
    int choice;
    int m, n;
    int** matrix;
    char* point;
    cin >> choice;
    cin >> m >> n;
    matrix = new int* [m];
    for(int i=0;i<m;i++)
    matrix[i]= new int[m];
    point = new char[m];
    list = new verNode[m];
```

```
        for(int i=0;i<m;i++)
        for (int j = 0; j < m; j++)
        {
        matrix[i][j] = 0;
        }
        /*choice 1 为临接矩阵的表示方法,
        choice 2 为临接矩阵带权的表示方法
        Choice3 为邻接表的表示方法
        Choice4 为邻接表带权的表示方法

        */
        switch (choice)
        {
        case 1:
        for (int i = 0; i < m; i++)
        {
        cin >> point[i];
        list[i].ver = point[i];
        }
        for (int i = 0; i < n; i++)
        {
        char temp1, temp2;
```

```
        cin >> temp1 >> temp2;
        int sign1, sign2;
        for (int j = 0; j < m; j++)
        {
        if (point[j] == temp1)
        sign1 = j;
        if (point[j] == temp2)
        sign2 = j;
        }
        matrix[sign1][sign2] = 1;
        list[sign1].head = new edgeNode(sign2, 1,
    list[sign1].head);

        }
        break;
        case 2:
        for (int i = 0; i < m; i++)
        {
        cin >> point[i];
        list[i].ver = point[i];
        }
        for (int i = 0; i < n; i++)
```

```
        {
        char temp1, temp2;
        int weight = 0;
        cin >> temp1 >> temp2>>weight;
        int sign1, sign2;
        for (int j = 0; j < m; j++)
        {
        if (point[j] == temp1)
        sign1 = j;
        if (point[j] == temp2)
        sign2 = j;
        }
        matrix[sign1][sign2] = weight;
        list[sign1].head = new edgeNode(sign2, weight,
list[sign1].head);

        }
        break;
        case 3:
        for (int i = 0; i < m; i++)
        {
        cin >> point[i];
```

```
        list[i].ver = point[i];
        }
        for (int i = 0; i < n; i++)
        {
        char temp1, temp2;
        cin >> temp1 >> temp2;
        int sign1, sign2;
        for (int j = 0; j < m; j++)
        {
        if (point[j] == temp1)
        sign1 = j;
        if (point[j] == temp2)
        sign2 = j;
        }
        matrix[sign1][sign2] = 1;
        matrix[sign2][sign1] = 1;
        list[sign1].head = new edgeNode(sign2, 1,
list[sign1].head);
        list[sign2].head = new edgeNode(sign1, 1,
list[sign2].head);

        }
```

```
        break;
        case 4:
        for (int i = 0; i < m; i++)
        {
        cin >> point[i];
        list[i].ver = point[i];
        }
        for (int i = 0; i < n; i++)
        {
        char temp1, temp2;
        int weight = 0;
        cin >> temp1 >> temp2 >> weight;
        int sign1, sign2;
        for (int j = 0; j < m; j++)
        {
        if (point[j] == temp1)
        sign1 = j;
        if (point[j] == temp2)
        sign2 = j;
        }
        matrix[sign1][sign2] = weight;
        matrix[sign2][sign1] = weight;
```

```
        list[sign1].head = new edgeNode(sign2, weight,
list[sign1].head);
        list[sign2].head = new edgeNode(sign1, weight,
list[sign2].head);
        }
        break;
        }
        for (int i = 0; i < m; i++)
        cout << point[i] << ' ';
        cout << endl;
        for (int i = 0; i < m; i++)
        {
        for (int j = 0; j < m; j++)
        {
        cout << setw(4) << matrix[i][j];
        }
        cout << endl;
        }
        for (int i = 0; i < m; i++)
        {
        edgeNode* n = list[i].head;
        cout << list[i].ver << "-->";
```

```
        while (n != NULL)
        {
        if (choice == 1 || choice == 3)
        {
        cout << n->end << " ";
        }
        else
        {
        cout << n->end <<","<<n->weight<<" ";
        }
        n = n->next;
        }
        cout << endl;
        }


        }
```

4.1.5 调试分析（遇到的问题和解决方法）

这道题是图数据结构最基本的知识点应用，基本没有碰到问题。

4.1.6 总结和体会

我通过这道题了解了图数据结构的基本知识，基本熟悉了图的基本存储方法，为接下来学习图的进阶应用打下了比较好的基础。

4.2 图的遍历

4.2.1 问题描述

图的遍历

描述

本题给定一个无向图，用邻接表作存储结构，用dfs和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）。

4.2.2 问题要求

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<=10000；
对于所有数据，0.5n<=m<=1.5n，保证输入数据无错。
下载p107_data.cpp，编译运行以生成随机测试数据

4.2.3 数据结构设计

直接简单的邻接矩阵

4.2.4 功能说明（函数、类）

/深度搜索

```
    void dfs(int** matrix, bool* visited,int number,const int m,int depth)
    {
    if (depth == 0)
    cout << number;
    else
    cout << " " << number;
    visited[number] = true;
    for (int i = 0; i < m; i++)
```

```
        {
        if (matrix[number][i] != 0)
        {
        if (visited[i] == false)
        dfs(matrix, visited, i, m,depth+1);
        else
        continue;
        }
        }
        }
        void bfs(int** matrix, bool* visited, const int
    m)
        {
        queue<int>a;
        for (int i = 0; i < m; i++)
        {
        if (visited[i] == true)
        continue;
        cout << "{";
        a.push(i);
        int num = 0;
        while (!a.empty())
```

```
        {
        int current = a.front();
        a.pop();
        if (visited[current] == true)
        {
        continue;
        }
        // cout << current << " ";
        visited[current] = 1;
        for(int i=0;i<m;i++)
        if (matrix[current][i] != 0)
        {
        if (visited[i] == false)
        {
        a.push(i);
        }

        }
        // if (!a.empty())
        if(num!=0)
        cout << " " << current;
        if(num==0)
```

```
        cout << current;
        num++;
        //else
        // cout << current;
        }
        cout << "}";
        }

        }
```

4.2.5 调试分析（遇到的问题和解决方法）

基本的困难在深度搜索的递归程序设计，但是没有遇到太多问题，因此很快就完成了这道程序的设计。

4.2.6 总结和体会

深度搜索和广度搜索可以说也是图的应用中比较重要的部分。只有掌握了这些图的基础部分知识，接下来学习图的进阶部分知识才有意义。

4.3 关键路径

4.3.1 问题描述

一个工程项目由一组子任务（或称活动）构成，子任务之间有的可以并行执行，有的必须在完成了其它一些子任务后才能执行，并且每个任务完成需要一定的时间。
对于一个工程，需要研究的问题是：
（1）由这样一组子任务描述的工程是否可行？
（2）若可行，计算完成整个工程需要的最短时间。
（3）这些任务中，哪些任务是关键活动（也就是必须按时完成的任务，否则整个项目就要延迟）。
现将这样一个工程项目用一个有向图表示，给定一组顶点，每个顶点表示任务之间的交接点（若任务2要在任务1完成后才可以开始，则这两任务之间必须有一个交接点，该点称作事件）。任务用有向边表示，边的起点是该任务可以开始执行的事件，终点是该任务已经完成的事件，边上的权值表示该任务完成需要执行的时间。
请你编写程序，回答上述三个问题。

4.3.2 基本要求

4.3.3 数据结构设计

```
struct edgeNode
{
    int end;
    int weight;
    edgeNode* next;
    edgeNode(int e, int w, edgeNode* n = NULL)
    {
        end = e;
        weight = w;
        next = n;
    }
};
struct verNode
{

    edgeNode* head;
    verNode(edgeNode* h = NULL)
    {
        head = h;
    }
};
```

邻接表

4.3.4 功能说明（函数、类）

```
int main()
{
verNode* list = NULL;
int m, n,current;
int count = 0;
queue<int>a;
stack<int>b;
```

```
        cin >> m >> n;
        int* ee = new(nothrow) int[m];
        if (ee == NULL)
        return -1;
        int* le = new(nothrow) int[m];
        if (le == NULL)
        return -1;
        for (int i = 0; i < m; i++)
        ee[i] = 0;
        list = new(nothrow) verNode[m];
        int count2=0;
        int* topo= new(nothrow)int[m];
        if (topo == NULL)
        return -1;
        for (int i = 0; i < m; i++)
        topo[i] = 0;
        int* degree = new (nothrow) int[m];
        if (degree == NULL)
        return -1;
        for (int i = 0; i < m; i++)
        degree[i] = 0;
        for (int i = 0; i < n; i++)
        {
        int temp1, temp2;
        int weight = 0;
        cin >> temp1 >> temp2 >> weight;
        list[temp1 - 1].head = new edgeNode(temp2 - 1, weight, list[temp1 -
1].head);
        }
        /*以上都是进行图的基本存储操作*/
        for (int i = 0; i < m; i++)
        {
        for (edgeNode* temp = list[i].head; temp != NULL; temp = temp->next)
        degree[temp->end]++;
        }
        for (int i = 0; i < m; i++)
        {
        if (degree[i] == 0)
        a.push(i);

        }
        /*进行拓扑排序，并对最早发生时间进行求解*/
        while (!a.empty())
```

```
        {
        current = a.front();
        b.push(current);
      for (edgeNode* temp = list[current].head; temp != NULL; temp = temp->next)
        if (ee[current] + temp->weight >= ee[temp->end])
        ee[temp->end] = ee[current] + temp->weight;
        count++;
        a.pop();
        for (edgeNode* temp = list[current].head; temp != NULL; temp = temp->next)
        if (--degree[temp->end] == 0)
        a.push(temp->end);
        }
        delete[]degree;
        if (count != m)
        {
        cout << "0" << endl;
        return 0;
        }
        int max = 0;
        for (int i = 0; i < m; i++)
        if (ee[i] >= max)
        {
        max = ee[i];
        }
        cout << max << endl;
        /*对最晚发生时间进行求解*/

        for (int i = 0; i < m; i++)
        le[i] = max;
        while(!b.empty())
        {
        current = b.top();
        b.pop();
        for (edgeNode* temp = list[current].head; temp != NULL; temp = temp->next)
        if (le[temp->end] - temp->weight <= le[current])
        le[current] = le[temp->end] - temp->weight;
        }
        /*进行遍历，输出所求序列*/
        for (int i = 0; i < m; i++) {
```

```
        for (edgeNode* p =list[i].head; p!=NULL; p = p->next) {
        int k = p->end;
        if (ee[i] + p->weight == le[k])
        cout << i + 1 << "->" << k + 1 << endl;
        }
        }
```

4.4 最短路径

4.4.1 题目描述

单源最短路径

描述

本题给出一张交通网络图，列出了各个城市之间的距离。请计算出从某一点出发到所有点的最短路径长度。

4.4.2 题目要求

输入

第一行包含三个整数n、m、s，分别表示n个顶点、m条无向边、出发点的编号。

接下来m行，每行包含三个整数ui、vi、wi，其中1<=ui,vi<=n, 1<=wi<=1000, 分别表示第i条无向边的出发点、目标点和长度。

顶点编号从1开始。

输出

一行，包含n个用空格分隔的整数，其中第i个整数表示从点s出发到点i的最短路径长度

（若s=i则最短路径长度为0，若从点s无法到达点i，则最短路径长度为2147483647，用INT_MAX表示）

4.4.3 数据结构设计

```
struct edgeNode
{
    int end;
    int weight;
    edgeNode* next;
    edgeNode(int e, int w, edgeNode* n = NULL)
    {
        end = e;
        weight = w;
        next = n;
    }
};
struct verNode
{

    edgeNode* head;
    verNode(edgeNode* h = NULL)
    {
        head = h;
    }
};
```

经典邻接表

4.4.4 功能说明（函数、类）

```
. struct edgeNode
. {
. int end;
. int weight;
. edgeNode* next;
. edgeNode(int e, int w, edgeNode* n = NULL)
. {
. end = e;
. weight = w;
. next = n;
. }
. };
. struct verNode
. {

.

. edgeNode* head;
```

```
        verNode(edgeNode* h = NULL)
        {
        head = h;
        }
        };
        struct length
        {
        int distance, position;
        };
        bool operator<(const length& v1, const length& v2) {
        return v1.distance > v2.distance;
        }
        /*迪杰斯塔拉算法*/
        int* Dijkstra(int start, int Noedge, int vertex, verNode* list)
        {
        priority_queue< length> p;
        int* distance = new int[vertex];
        int* prev = new int[vertex];
        bool* known = new bool[vertex];
        int min = Noedge;
        int u;
        for (int i = 0; i < vertex; i++)
        {
        known[i] = false;
        if (i != start - 1)
        distance[i] = Noedge;
        else
        {
        distance[i] = 0;
        length temp;
        temp.distance = distance[i], temp.position = i;
        p.push(temp);
        }

        }
        prev[start-1] = start-1;
        distance[start-1] = 0;
        while (!p.empty() )
        {
        if (known[p.top().position] == false)
        {
        min = p.top().distance;
        u = p.top().position;
```

```
        }
        p.pop();
        known[u] = true;
        for (edgeNode* temp = list[u].head; temp != NULL; temp = temp->next)
        {
        if (!known[temp->end] && distance[temp->end] > min + temp->weight)
        {
        distance[temp->end] = min + temp->weight;
        length temp1;
        temp1.distance = distance[temp->end], temp1.position = temp->end;
        if (known[temp1.position] == false)
        p.push(temp1);
        prev[temp->end] = u;
        }
        }
        }
        return distance;

        }
```

4.4.5 调试分析（遇到的问题和解决方法）

这道题整体来说最大的难点在于优先队列的使用，别的地方还是比较经典的算法，没有太大的问题。

4.4.6 总结和体会

这道题总体来说又是对于经典算法的演练与实践，对于最短路径的求解算法有了一个基本的了解，有助于我们对于接下来的知识进行进一步学习。当然可能存在一些还可以优化的地方，限于时间没有进行进一步优化。

4.5.1 小世界现象

4.5.1 题目描述

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。”如图1所示。

4.5.2 题目要求

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%”。

4.5.3 数据结构设计

```
struct edgeNode
{
    int end;
    int weight;
    edgeNode* next;
    edgeNode(int e, int w, edgeNode* n = NULL)
    {
        end = e;
        weight = w;
        next = n;
    }
};
struct verNode
{

    edgeNode* head;
    verNode(edgeNode* h = NULL)
    {
        head = h;
    }
};
```

经典邻接表

4.5.4 功能说明（函数、类）

/*进行广度搜索，并且把距离初始节点大于等于 6 的部分都舍去*/

```
int Bfs(verNode* list, int i, bool* judge, int m)
{
    queue<int>a;
    int count = 0;
```

```
            int layer = 0;
            judge[i] = true;
            a.push(i);
            int record1 = 1;
            int signal;
            while (layer <= 5)
            {
                signal = 0;
                for (int i = 0; i < record1; i++)
                {
                    int current = a.front();
                    a.pop();
                      for (edgeNode* temp = list[current].head; temp != NULL; temp =
temp->next)
                    {
                        if (judge[temp->end] == false)
                        {
                            a.push(temp->end);
                            judge[temp->end] = true;
                            signal = signal + 1;
                        }
                    }
                }
                record1 = signal;
                layer++;
            }
            int result = 0;
            for (int i = 0; i < m; i++)
            {
                if (judge[i] == true)
                    result++;
            }
            return result;

        }
        void refresh(bool* judge, int m)
        {
            for (int i = 0; i < m; i++)
            {
                judge[i] = false;
            }
        }
```

4.5.5 调试分析（遇到的问题和解决方法）

简单 bfs 的应用，最大的困难是百分比的数据处理，不过模仿了给与的测试程序的处理方式解决了这个问题。

4.5.6 总结和体会

这道题整体来说比较趣味，同时又是对于图相关知识的巩固，十分具有意义，对于 bfs 的进一步复习，当然这题用迪杰斯塔拉算法应该也是可以的。

2. 实验总结

总体来说，本次作业设计的算法题覆盖了所有数据结构基础部分的内容，比较好的让我们掌握了图的相关知识和应用。同时也为我们对于图的进一步学习打下了比较良好的基础。