《数据结构》课程实践报告

院、系	计算机学院	年级专业	21 计科	姓名	方浩楠	学号	2127405048
实验布置日期	2022-11-01		提交日期			成绩	

课程实践实验 8：图的实现和应用

一、问题描述及要求

一、 无向图的实现
（1）创建一个无向图的邻接矩阵和邻接表结构;
（2）在邻接矩阵结构下对该图进行深度优先搜索;
（3）在邻接表结构下对该图进行广度优先搜索。
二、无向网的最小生成树（可选）
（1）创建一个无向网的邻接矩阵表示;
（2）求其最小生成树并输出。
三、骑士周游问题（可选）
在一个国际象棋棋盘上，一个棋子“马”（骑士），按照“马走日”的规则，从一个格子出发，走遍所有棋盘格恰好一次，一个这样的走棋序列称为一次“周游”。利用图的搜索算法，给出一个周游序列。

二、概要设计

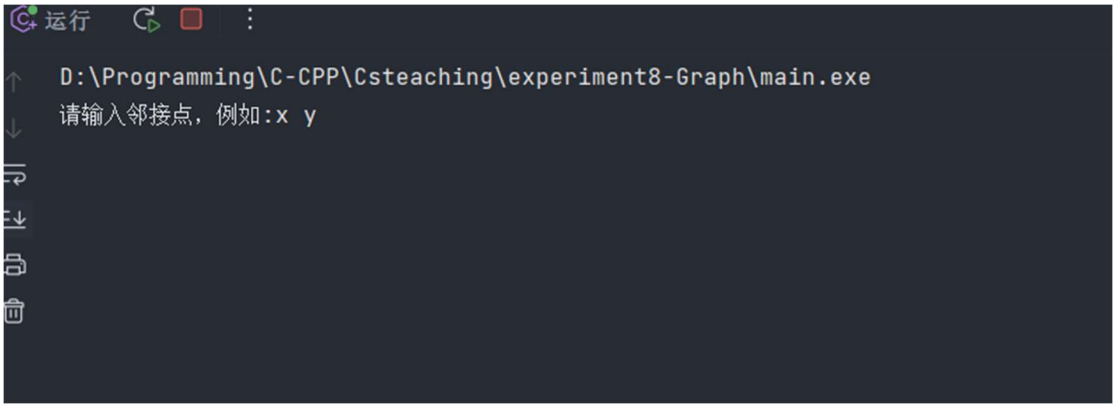
（1）对实验内容的理解

实验要求实现无向图的不同的存储方式，并且在不同存储方式的基础上分别实现 BFS 和 DFS

（2）功能列表

1、创建无向图的邻接矩阵
2、创建无向图的邻接表
3、在邻接矩阵结构下实现 DFS
4、在邻接表结构下实现 BFS

（3）程序运行的界面设计



三、详细设计

```
Class MGraph {            //创建无向图
    MGraph(DataType a[], int n, int e);      //创建图的邻接矩阵结构
    void DFS();              //调用递归函数实现深度优先搜索
    void DFS(int v, int visited[MaxSize]);       //递归实现深度优先搜索
}
Class ALGraph{
    ALGraph(DataType a[], int n, int e);         //创建图的邻接表
    ~ALGraph();      //  | Vertex | firstEdge |  析构函数
    void BFSTraverse();                          //实现广度优先搜索
}
```

三、实验结果

测试输入:

0 1
1 2
2 3
3 4
4 5
5 6
6 1

期望输出:
BFS: ABCDEF
DFS:ABCDEF
实际输出:

```
D:\Programming\C-CPP\Csteaching\experiment8-Graph\main.exe
请输入邻接点，例如:x y
0 1
1 2
2 3
3 4
4 5
5 6
6 1
邻接矩阵DFS:A B C D E F
请输入邻接点，例如:x y
0 1
1 2
2 3
3 4
4 5
5 6
6 1
邻接表BFS:A B C D E F
```

五、实验分析与探讨

测试结果分析:
若图采用了领接矩阵的方式存储，则时间复杂度$T(n) = O(n^2)$，空间复杂度$S(n) = O(n^2)$，其中 n 为顶点的个数
若图采用了领接表的方式存储，则时间复杂度$T(n) = O(e)$，空间复杂度$S(n) = O(n + e)$，其中 n 为顶点的个数，e 为边的个数

六、小结

本次实验让我了解到了图的不同的存储方式，以及不同存储方式在进行 BFS 和 DFS 时的差异。

本次实验对最小生成树了解的不够，以后会在此方面多加了解

附录：源代码

实验环境：gcc.exe (Rev4, Built by MSYS2 project) 12.2.0
Clion 2022.3.1
源代码:
(1) main.cpp

```
#include <iostream>

using namespace std;

struct EdgeNode
{
    int adjvex;
    EdgeNode *next;
};

template<typename DataType>
struct VertexNode
{
    DataType vertex;
    EdgeNode *firstEdge;
};

const int MaxSize = 10;

template<typename DataType>
class MGraph
{
public:
    MGraph(DataType a[], int n, int e);

    void DFS();

    void DFS(int v, int visited[MaxSize]);

private:
    DataType vertex[MaxSize];
    int edge[MaxSize][MaxSize] = {0};
    int vertexNum, edgeNum;
};

template<typename DataType>
MGraph<DataType>::MGraph(DataType a[], int n, int e) {
```

```
    int i, j, k;
    vertexNum = n;
    edgeNum = e;
    cout << "请输入邻接点，例如:x y" << endl;
    for (i = 0; i < vertexNum; i++){
        vertex[i] = a[i];
    }
    for (k = 0; k < edgeNum; k++) {
        cin >> i >> j;
        edge[i][j] = 1;
        edge[j][i] = 1;
    }
}

template<typename DataType>
void MGraph<DataType>::DFS() {
    int visited[MaxSize] = {};
    for (int v = 0; v < vertexNum; v++)
        if (visited[v] == 0){
            DFS(v, visited);
        }
    cout << endl;
}

template<typename DataType>
void MGraph<DataType>::DFS(int v, int visited[MaxSize]) {
    cout << vertex[v] << " ";
    visited[v] = 1;
    for (int i = 0; i < vertexNum; i++){
        if (edge[v][i] == 1 && visited[i] == 0){
            DFS(i, visited);
        }

    }
}

template<typename DataType>
class ALGraph
{
public:
    ALGraph(DataType a[], int n, int e);

    ~ALGraph();
```

```
    void BFS();

private:
    VertexNode<DataType> adjlist[MaxSize];
    int vertexNum, edgeNum;
    int visited[MaxSize] = {0};
};

template<typename DataType>
ALGraph<DataType>::ALGraph(DataType a[], int n, int e) {
    int i, j, k;
    cout << "请输入邻接点，例如:x y" << endl;
    EdgeNode *s;
    vertexNum = n;
    edgeNum = e;
    for (i = 0; i < vertexNum; i++) {
        adjlist[i].vertex = a[i];
        adjlist[i].firstEdge = nullptr;
    }
    for (k = 0; k < edgeNum; k++) {
        cin >> i >> j;
        s = new EdgeNode;
        s->adjvex = j;
        s->next = adjlist[i].firstEdge;
        adjlist[i].firstEdge = s;
    }
}

template<typename DataType>
ALGraph<DataType>::~ALGraph() {
    EdgeNode *p, *q;
    for (int i = 0; i < vertexNum; i++) {
        p = q = adjlist[i].firstEdge;
        while (p != nullptr) {
            p = p->next;
            delete q;
            q = p;
        }
    }
}

template<typename DataType>
void ALGraph<DataType>::BFS() {
    int v = 0;
```

```
    int w, j, Q[MaxSize];
    int front = -1, rear = -1;
    EdgeNode *p = nullptr;
    cout << adjlist[v].vertex << " ";
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear) {
        w = Q[++front];
        p = adjlist[w].firstEdge;
        while (p != nullptr) {
            j = p->adjvex;
            if (visited[j] == 0) {
                cout << adjlist[j].vertex << " ";
                visited[j] = 1;
                Q[++rear] = j;
            }
            p = p->next;
        }
    }
    cout << endl;
}

int main() {
    char ch[] = {'A', 'B', 'C', 'D', 'E', 'F'};
    MGraph<char> MG(ch, 6, 7);
    cout << "邻接矩阵 DFS:";
    MG.DFS();
    ALGraph<char> ALG(ch, 6, 7);
    cout << "邻接表 BFS:";
    ALG.BFS();
    system("pause");
}
```