实验六 图的操作

一、 实验目的

1. 掌握图的基本存储方法；
2. 掌握有关图的操作算法并用高级语言编程实现；
3. 熟练掌握图的两种搜索路径的遍历方法；
4. 提交 OJ 系统进行验证。

二、 实验要求

1. 认真阅读并理解教材上相关操作函数。
2. 正确编写本程序并能上机运行。
3. 必须完成：
   1) 建立图的邻接矩阵存储
   2) 邻接矩阵存储方式下的深度优先遍历
   3) 邻接矩阵存储方式下的广度优先遍历
   4) 邻接表存储方式下的深度优先遍历
   5) 邻接表存储方式下的广度优先遍历

三 程序框架参考如下

第 1 题 邻接矩阵存储图的遍历

题目描述：邻接矩阵是图的一种存储结构，深入理解其实现的含义。阅读主函数、邻接矩阵存储的图的输出、求顶点在图中的位置等相关代码，完成邻接矩阵连通的**无向图**的创建、深度优先搜索、广度优先搜索。
输入：输入包括多行，每行代表一定的含义 输入 1，输出图中的顶点信息和邻接矩阵； 输入 2，代表**创建无向图**，然后输入 n 和 m，表示图的顶点和边的数量，接着输入 n 个字符表示顶点的意义，然后输入 m 行，每行由 x 和 y 构成，代表顶点 x 和 y 有一条边相连； 输入 3，代表深度遍历图，后面输入字符 x，表示从顶点 x 进行深度优先遍历； 输入 4，代表广度遍历图，后面输入字符 x，表示从顶点 x 进行广度优先遍历。
输出：按照输入的顺序依次输出相关信息。
输入样例： 2 4 4 a b c d a b a c b c c d

1
3 a
4 c

输出样例

abcd
0110
1010
1101
0010
abcd
cabd

程序框架

```
#include <iostream>
using namespace std;

#define MVNum 20     //假设当前顶点数最多为 20 个
int visited[MVNum];//用来存放当前顶点是否遍历过

//*****定义邻接矩阵****
typedef char VerTexType;
typedef int ArcType;
typedef struct
{
    VerTexType vexs[MVNum];
    ArcType arcs[MVNum][MVNum];
    int vexnum,arcnum;
}AMGraph;

//输出图的邻接矩阵
void PrintAMGraph(AMGraph MG){
    int i,j;
    if(MG.vexnum<=0)
        return;
    for(i=0; i<MG.vexnum; i++)
        cout<<MG.vexs[i];
    cout<<endl;

    for(i=0; i<MG.vexnum; i++){
        for(j=0; j<MG.vexnum; j++)
            cout<<MG.arcs[i][j];
        cout<<endl;
    }
}

//若 G 中存在 v,则返回该顶点在图中位置,否则返回-1
```

```
int LocateVex(AMGraph G, char v)
{
    int i;
    for(i=0; i<G.vexnum; i++)
    {
        if(G.vexs[i]==v)
            return i;
    }
    return -1;
}

//返回顶点 v 的第一个临接点位置,否则返回-1
int FisrtAdjVertex(AMGraph G, int v)
{
    //--------补充代码--Start------

    //--------补充代码--End-------
}

//返回顶点 v 相对于 w 的下一个临接点位置,否则返回-1
int NextAdjVertex(AMGraph G, int v, int w)
{
    //--------补充代码--Start------

    //--------补充代码--End-------
}

//邻接矩阵存储方式建立无向图
void CreateAMGraph(AMGraph &G)
{
    /*
    step 1.输入图中顶点总数与边的总数
    step 2.输入图中顶点信息，存放到 G.vexs[i]数组中
    step 3.初始化邻接矩阵中所有值为 0
    step 4.输入边的信息，修改邻接矩阵中相对应的元素值
    */
    //--------补充代码--Start------

    //--------补充代码--End-------
}

//从第 i 个顶点开始深度遍历
void DFSAMGraph(AMGraph G,int i)
{
    //--------补充代码--Start------
```

```
    //--------补充代码--End-------
}

//从第 i 个顶点开始广度遍历
void BFSAMGraph(AMGraph G,int i)
{
    //--------补充代码--Start------

    //--------补充代码--End-------
}
/*
4 4
a b c d
a b
a c
b c
c d
*/
//主函数
int main()
{
    int i,select,vex;
    char start;
    AMGraph MG;
    MG.vexnum=MG.arcnum=0;

    while(cin>>select)
    {
        if(select==1){//输出图
            PrintAMGraph(MG);
        }
        else if(select==2){//建立无向图
            CreateAMGraph(MG);
        }
        else if(select==3){//深度遍历
            getchar();
            cin>>start;//遍历的起始顶点
            vex = LocateVex(MG, start);
            for(i=0; i<MG.vexnum; i++)
                visited[i] = 0;
            DFSAMGraph(MG,vex);
            cout<<endl;
        }
        else if(select==4){//广度遍历
            getchar();
            cin>>start;//遍历的起始顶点
```

```
                vex = LocateVex(MG, start);
                for(i=0; i<MG.vexnum; i++)
                    visited[i] = 0;
                BFSAMGraph(MG,vex);
                cout<<endl;
            }
        }
        return 0;
}
```

第 2 题 邻接表存储图的应用

题目描述：邻接表是图的一种存储结构，深入理解其实现的含义。阅读主函数、邻接表存储的图的输出、求顶点在图中的位置等相关代码，完成邻接表有向网的创建、深度优先搜索和应用求解。

输入：输入包括多行，每行代表一定的含义

输入 1，代表**创建有向网**，然后输入 n 和 m，表示图的顶点和边的数量，接着输入 n 个字符表示顶点的意义，然后输入 m 行，每行由 x、y 和 w 构成，代表顶点 x 和 y 有一条权值为 w(>0)的边；
输入 2，代表深度遍历图，后面输入字符 x，表示从顶点 x 进行深度优先遍历所有连通的节点；
输入 3，代表求解整个网络的权值之和；
输入 4，代表出度大于入度的顶点的个数。

输出：按照输入的顺序依次输出相关信息。

输入样例：

```
1
4 4
a b c d
a b 5
b c 6
c a 7
a d 8
2 b
3
4
```

输出样例

```
bcad
26
1
```

程序框架

```
#include <iostream>
using namespace std;

#define MaxInt 32767        //代表无穷大
#define MVNum 10            //假设当前顶点数最多为 10 个
typedef char VerTexType;    //结点数据类型
int visited[MVNum];         //用来存放当前顶点是否遍历过
```

```
//******定义邻接表******
typedef struct ArcNode{                  //边结点
    int adjvex;                          //邻接点在数组中的位置
    struct ArcNode *nextarc;    //指向下一个边结点的指针
    int weight;                          //边的权重(>0)
}ArcNode;
typedef struct VNode{              //表头结点
    VerTexType data;
    ArcNode *firstarc;
}VNode, AdjList[MVNum];
typedef struct{                          //邻接表
    AdjList vexs;
    int vexnum,arcnum;
}ALGraph;

//若 G 中存在 v,则返回该顶点在图中位置,否则返回-1
int LocateVex(ALGraph G, char v){
    int i;
    for(i=0; i<G.vexnum; i++)
    {
        if(G.vexs[i].data==v)
            return i;
    }
    return -1;
}

//建立邻接表存储方式的无向图
void CreateALGraph(ALGraph &G)
{
    /*  1.输入图中顶点总数与边的总数
        2.输入图中顶点信息
        3.输入边的信息，建立边结点，插入到相对应数组元素后的边表中（注：无向图中每条边需要插入两个边结点）。*/
    int i,j,k;
    char v1,v2;
    ArcNode *p, *q;
    int w;

    cin>>G.vexnum>>G.arcnum;
    for(i=0; i<G.vexnum; i++)
    {
        cin>>G.vexs[i].data;
        G.vexs[i].firstarc=NULL;
    }
    //--------补充代码--Start------
```

```
    //--------补充代码--End------
}

//深度遍历邻接表
void DFSALGraph(ALGraph G,int i)
{
    //--------补充代码--Start------


    //--------补充代码--End------
}

//计算图中所有边的权值之和
int GetGraphWeight(ALGraph G)
{
    //--------补充代码--Start------


    //--------补充代码--End------
}

//计算图中出度大于入度的结点个数
int GetNode(ALGraph G)
{
    //--------补充代码--Start------

    //--------补充代码--End------
}

//主函数
int main()
{
    int i,select,vex;
    char start;
    ALGraph G;
    G.vexnum=G.arcnum=0;

    while(cin>>select)
    {
        if(select==1){//建立有向网
            CreateALGraph(G);
        }
        else if(select==2){//深度遍历
            cin>>start;//遍历的起始顶点
            vex = LocateVex(G, start);
```

```
                for(i=0; i<G.vexnum; i++)
                    visited[i] = 0;
                DFSALGraph(G,vex);
                cout<<endl;
            }
            else if(select==3)//求权和
                cout<<GetGraphWeight(G)<<endl;
            else if(select==4)//求结点数
                cout<<GetNode(G)<<endl;
        }
        return 0;
}
```