《算法与数据结构》实验报告

实验名称：图型数据结构及其应用

系　　别：计算机科学与技术学院

专　　业：计算机类

班　　级：21 级计算机类 1 班

姓　　名：侯文辉

学　　号：2021463030114

实验日期：　2022 年 6 月 10 日

2. 实验名称：图型数据结构及其应用

2. 实验目的：

通过实验达到：

⑴ 理解和掌握图的基本概念、基本逻辑结构；
⑵ 理解和掌握图的邻接矩阵存储结构、邻接链表存储结构；
⑶ 理解和掌握图的 DFS、BFS 遍历操作的思想及其实现；
⑷ 加深对堆栈、队列的概念及其典型操作思想的理解；
⑸ 掌握典型图操作算法的算法分析。

3. 对每个题目，包括：

设图结点的元素类型为 char，建立一个不少于 8 个顶点的带权无向图 G，实现以下图的各种基本操作的程序：

① 用邻接矩阵作为储结构存储图 G 并输出该邻接矩阵；
② 用邻接链表作为储结构存储图 G 并输出该邻接链表；
③ 按 DFS 算法输出图 G 中顶点的遍历序列；
④ 按 BFS 算法输出图 G 中顶点的遍历序列；
⑤主函数通过函数调用实现以上各项操作。

⑴ 所定义的数据结构：

邻接矩阵 1 个，邻接表 3 个

```
#define MAX_VEX 3 //最大顶点数

/**********邻接矩阵宏定义**********/
#define VexType char //顶点向量，类型可自定义，
#define AdjType int  //邻接矩阵,权值类型可自定义
```

```c
/**********邻接表宏定义**********/
#define InfoType int // 与边或弧相关的信息，如权值

typedef struct
{
    int vexnum, arcnum;            /*  图的当前顶点数和弧数  */
    VexType vexs[MAX_VEX];         /*  顶点向量组  */
    AdjType adj[MAX_VEX][MAX_VEX]; //邻接矩阵,
} MGraph;

/*  表结点类型定义  */
typedef struct LinkNode
{
    int adjvex;               // 邻接点在头结点数组中的位置(下
标)
    InfoType info;            // 与边或弧相关的信息，如权值
    struct LinkNode *nextarc; // 指向下一个表结点
} LinkNode;

/*  顶点结点类型定义  */
typedef struct VexNode
{
    VexType data;       // 顶点信息
    int indegree;       // 顶点的度，有向图是入度或出度或没有
    LinkNode *firstarc; // 指向第一个表结点
} VexNode;

/*  图的结构定义  */
typedef struct
{
    int vexnum;               //图的当前顶点数
    int visited[MAX_VEX];     //访问标志数组,for DFS、BFS
    VexNode AdjList[MAX_VEX]; //顶点表
} ALGraph;
```

(2) 主要操作算法思想或算法步骤:

邻接矩阵转换为邻接表：
首先初始化邻接表，之后每个顶点计算相连的边，加入 LinkNode，利用头插法加入邻接节点

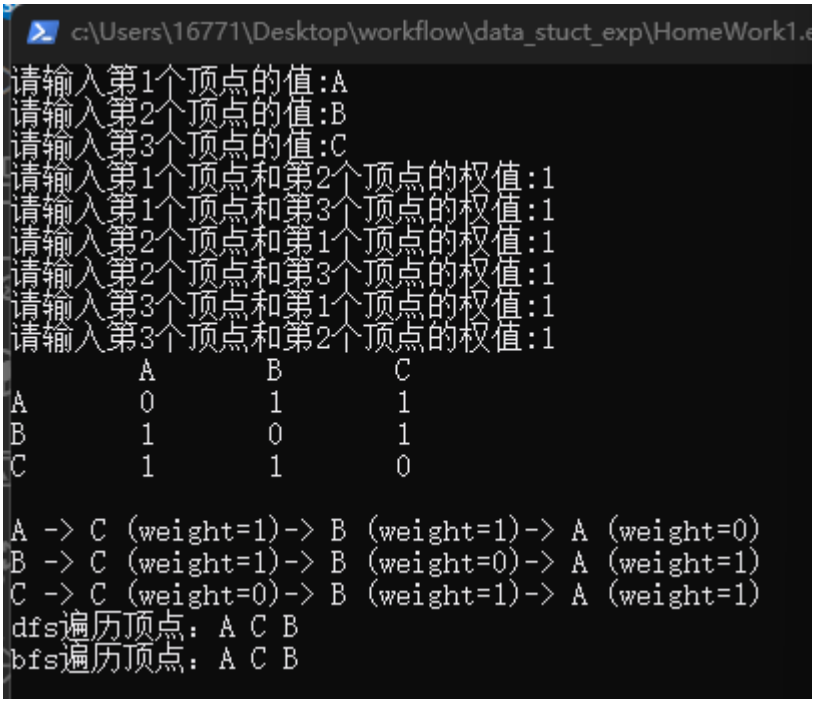
DFS 算法输出图 G 中顶点的遍历序列：
DFS 属于一路挖到底的，所以我们从初始节点开始，将其第一个邻接的点判断有无被访问过，没访问过的就开始递归，访问过的就下一个链表节点。

BFS 算法输出图 G 中顶点的遍历序列：
BFS 属于广泛搜索，与 DFS 比较，他并不会一开始就访问第一个节点开始递归，而是输出完该节点临近的节点后开始下一个节点继续广泛的扫描。

(3) 主要操作算法的算法分析:

邻接矩阵转换为邻接表：时间复杂度 O(n2),空间复杂度 O(n)
DFS 算法输出图 G 中顶点的遍历序列：时间复杂度 O(n),空间复杂度 O(n)
BFS 算法输出图 G 中顶点的遍历序列：时间复杂度 O(n),空间复杂度 O(n)

(4) 程序运行过程及结果（抓屏粘贴）:



4. 问题与总结:

对 BFS 和 DFS 的题目模板还是不够熟练，仍要多去刷一下题熟悉一下模板。

附件：代码

```
/************************** Dongguan-University of Technology
-ACE**************************
 * @file HomeWork1.c
 * @author pansyhou 侯文辉 (1677195845lyb@gmail.com)
 * @brief
    1. 用邻接矩阵作为储结构存储图 G 并输出该邻接矩阵；
    2. 用邻接链表作为储结构存储图 G 并输出该邻接链表；
    3. 按 DFS 算法输出图 G 中顶点的遍历序列；
    4. 按 BFS 算法输出图 G 中顶点的遍历序列；
    5. 主函数通过函数调用实现以上各项操作。
 * @version 0.1
```

```
 * @date 2022-06-01
 *
 * @copyright Copyright (c) 2022
 *
 ************************ Dongguan-University of Technology
-ACE*************************/
#include "stdio.h"
#include "stdlib.h"

#define MAX_VEX 3 //最大顶点数

/**********邻接矩阵宏定义**********/
#define VexType char //顶点向量，类型可自定义，
#define AdjType int  //邻接矩阵,权值类型可自定义

/**********邻接表宏定义**********/
#define InfoType int // 与边或弧相关的信息，如权值

typedef struct
{
    int vexnum, arcnum;            /*  图的当前顶点数和弧数  */
    VexType vexs[MAX_VEX];         /*  顶点向量组  */
    AdjType adj[MAX_VEX][MAX_VEX]; //邻接矩阵,
} MGraph;

/*  表结点类型定义   */
typedef struct LinkNode
{
    int adjvex;               // 邻接点在头结点数组中的位置(下
标)
    InfoType info;            // 与边或弧相关的信息，如权值
    struct LinkNode *nextarc; // 指向下一个表结点
} LinkNode;

/*  顶点结点类型定义   */
typedef struct VexNode
{
    VexType data;       // 顶点信息
```

```
    int indegree;       // 顶点的度，有向图是入度或出度或没有
    LinkNode *firstarc; // 指向第一个表结点
} VexNode;

/*  图的结构定义   */
typedef struct
{
    int vexnum;               //图的当前顶点数
    int visited[MAX_VEX];     //访问标志数组,for DFS、BFS
    VexNode AdjList[MAX_VEX]; //顶点表
} ALGraph;

/***********图的实现***********/

//邻接矩阵初始化
void InitGraph(MGraph *G)
{
    int i, j;
    G->vexnum = 0; //初始化顶点数
    G->arcnum = 0; //初始化弧数
    for (i = 0; i < MAX_VEX; i++)
    {
        for (j = 0; j < MAX_VEX; j++)
        {
            if (i == j)
                G->adj[i][j] = 0;
            else
                G->adj[i][j] = INT_MAX;//初始化为无穷大
        }
    }
}

//插入顶点
void InsertVertex(MGraph *G, VexType Vertex)
{
    if (G->vexnum > MAX_VEX)
    {
        printf("顶点已满");
```

```
        return;
    }
    G->vexs[G->vexnum++] = Vertex;
}

//插入边
void InsertEdge(MGraph *G, int v1, int v2, int weight)
{
    if (G->vexnum < v1 || G->vexnum < v2||v1<0||v2<0)
    {
        printf("顶点不存在");
        return;
    }
    G->adj[v1][v2] = weight;
    G->arcnum++;
}

//删除边
void DelEdge(MGraph *G, int v1, int v2)
{
    if(G->vexnum < v1 || G->vexnum < v2||v1<0||v2<0)//处理越
界
    {
        printf("参数错误");
        return;
    }
    if(G->adj[v1][v2]==INT_MAX||v1==v2)//处理越界
    {
        printf("该边不存在");
        return;
    }
    G->adj[v1][v2]=INT_MAX;
    G->arcnum--;
}

//图的创建
void CreatGraph(MGraph *G)
{
```

```
    //先插入顶点
    for(int i=0;i<MAX_VEX;i++)
    {
        printf("请输入第%d 个顶点的值:",i+1);
        scanf(" %c",&G->vexs[i]);
        InsertVertex(G,G->vexs[i]);
    }
    for(int i=0;i<G->vexnum;i++)
    {
        for(int j=0;j<G->vexnum;j++)
        {
            if(i==j)
                continue;//跳过对角线
            printf("请输入第%d 个顶点和第%d 个顶点的权
值:",i+1,j+1);
            scanf("%d",&G->adj[i][j]);
            InsertEdge(G,i,j,G->adj[i][j]);
        }
    }
}

//输出邻接矩阵
void PrintMgraph(MGraph *G)
{
    for(int i=0;i<G->vexnum+1;i++)
    {
        for(int j=0;j<G->vexnum+1;j++)
        {
            if(i==0&&j==0)printf("\t");
            else if(i==0&&j!=0)printf("%c\t",G->vexs[j-1]);//
输出一个好看的矩阵将第一行和第一列化为顶点的信息
            else if(i!=0&&j==0)printf("%c\t",G->vexs[i-1]);
            else {
                printf("%d\t",G->adj[i-1][j-1]);
            }
        }
        printf("\n");
    }
```

```
}
//输出邻接表
void PrintAlgraph(ALGraph *G)
{
    for(int i=0;i<G->vexnum;i++)
    {
        printf("%c ",G->AdjList[i].data);
        LinkNode *p=G->AdjList[i].firstarc;
        while (p!=NULL)
        {
            printf("->");
            printf(" %c
(weight=%d)",G->AdjList[p->adjvex].data,p->info);
            p=p->nextarc;
        }
        printf("\n");
    }
}

//邻接矩阵转换为邻接表
void MGraphToALGraph(MGraph *G, ALGraph *AL)
{
    int i, j;
    AL->vexnum = G->vexnum;
    //初始化邻接表
    for (i = 0; i < MAX_VEX; i++)
    {
        AL->AdjList[i].data = G->vexs[i];
        AL->AdjList[i].indegree = 0;         //初始化入度
        AL->AdjList[i].firstarc = NULL;      //第一个表结点先为
空
        AL->visited[i] = 0;                  //初始化访问标志
    }
    //每个顶点计算相连的边，加入 LinkNode，
    for (i = 0; i < G->vexnum; i++)
    {
        for (j = 0; j < G->vexnum; j++)
        {
```

```
            if (G->adj[i][j] != INT_MAX)
            {
                LinkNode *p = (LinkNode
*)malloc(sizeof(LinkNode));
                p->adjvex = j;                          //下
标
                p->info = G->adj[i][j];                 //权值
                //（头插法，将新建的 p 看作新的头，p->next 连接旧
头）
                p->nextarc = AL->AdjList[i].firstarc;   //插入
到链表头
                AL->AdjList[i].firstarc = p;            //更新
链表头
                AL->AdjList[j].indegree++;              //度
数++
            }
        }
    }
    return;
}

//dfs 算法遍历顶点
void DFS(ALGraph *G, int v)
{
    LinkNode *p;
    G->visited[v] = 1;
    printf("%c ", G->AdjList[v].data);
    p = G->AdjList[v].firstarc;
    while (p != NULL)
    {
        if (G->visited[p->adjvex] == 0)
            DFS(G, p->adjvex);
        p = p->nextarc;
    }
}

//清楚访问标志
void ClearVisited(ALGraph *G)
```

```
{
    for (int i = 0; i < G->vexnum; i++)
        G->visited[i] = 0;
}

//bfs 算法遍历顶点
void BFS(ALGraph *G, int v)
{
    LinkNode *p;
    int i;
    G->visited[v] = 1;
    printf("%c ", G->AdjList[v].data);
    p = G->AdjList[v].firstarc;
    while (p != NULL)
    {
        if (G->visited[p->adjvex] == 0)
        {
            G->visited[p->adjvex] = 1;
            printf("%c ", G->AdjList[p->adjvex].data);
        }
        p = p->nextarc;
    }
    for (i = 0; i < G->vexnum; i++)
    {
        if (G->visited[i] == 0)
            BFS(G, i);
    }
}

int main(void)
{
    MGraph *MGraph=malloc(sizeof(MGraph));
    ALGraph *ALGraph1=malloc(sizeof(ALGraph));

    //初始化邻接矩阵
    InitGraph(MGraph);
    //初始化邻接表
    CreatGraph(MGraph);
```

```
    //输出邻接矩阵
    PrintMgraph(MGraph);
    printf("\n");
    //转换为邻接表
    MGraphToALGraph(MGraph,ALGraph1);
    //输出邻接表
    PrintAlgraph(ALGraph1);
    //dfs 算法遍历顶点
    printf("dfs 遍历顶点：");
    DFS(ALGraph1,0);
    printf("\n");
    //清楚访问标志
    ClearVisited(ALGraph1);
    //bfs 算法遍历顶点
    printf("bfs 遍历顶点：");
    BFS(ALGraph1,0);
    printf("\n");
    return 1;
}
```