数据结构实验报告——实验八

学号：_20201050331___姓名：_黄珀芝_得分：_____

一、实验目的

1、复习图的逻辑结构、存储结构及基本操作；

2、掌握邻接矩阵、邻接表及图的创建、遍历；

3、了解图的应用

二、实验内容

1、（必做题）假设图中数据元素类型是字符型，请采用邻 接矩阵实现图的以下基本操作：

（1）构造图（包括有向图、有向网、无向图、无向网）；

（2）根据深度优先遍历图；

（3）根据广度优先遍历图；

三、数据结构及算法描述

数据结构：非线性结构：图；存储结构：链表；

1、图的邻接矩阵的存储方式是用两个数组来实现的，一个一维数组存储顶点信息，一个二维数组存储线（无向图）或弧（有向图）的信息。

2、无向图的邻接矩阵，两个顶点有边则为 1，否则，为 0；因为是无向图 arc[i][j] = arc[j][i]，所以矩阵为对称矩阵，对角线为自己到自己的边，邻接矩阵中，行之和或者列之和都为各顶点度的总数。

3、无向网图和无向图差不多，就是加了权值，两个顶点之间无边的话距离是∞。

如果是有向图，邻接矩阵就不是对称矩阵了。

算法描述：前提知识：

1、 图和网的区别：网是带权值的图

有向和无向的区别：有向直接标出谁指向谁，无向是有向的特例，<v1,v2>有弧，说明<v2,v1>也有弧。

构图：

① 确定顶点数，弧数，是否有权值

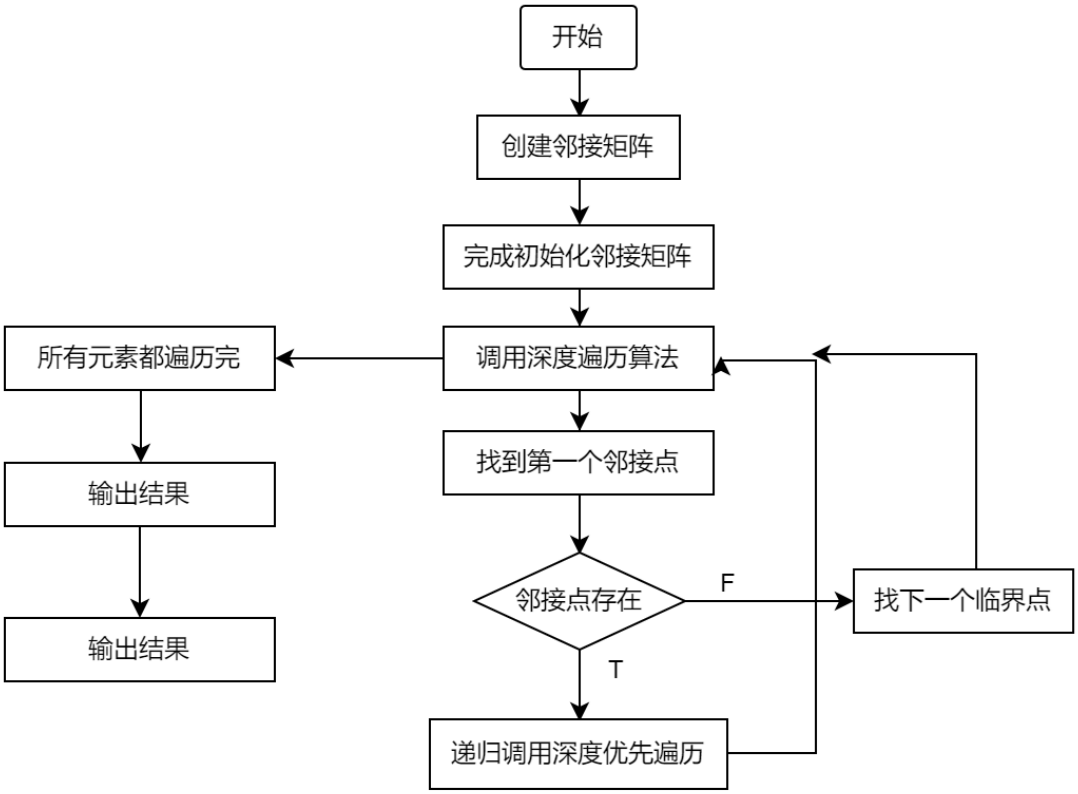
② 输入每个顶点，弧<弧尾，弧头>，权值

③ 若是无向，则需实现弧<v2,v1>与<v1,v2>的同置

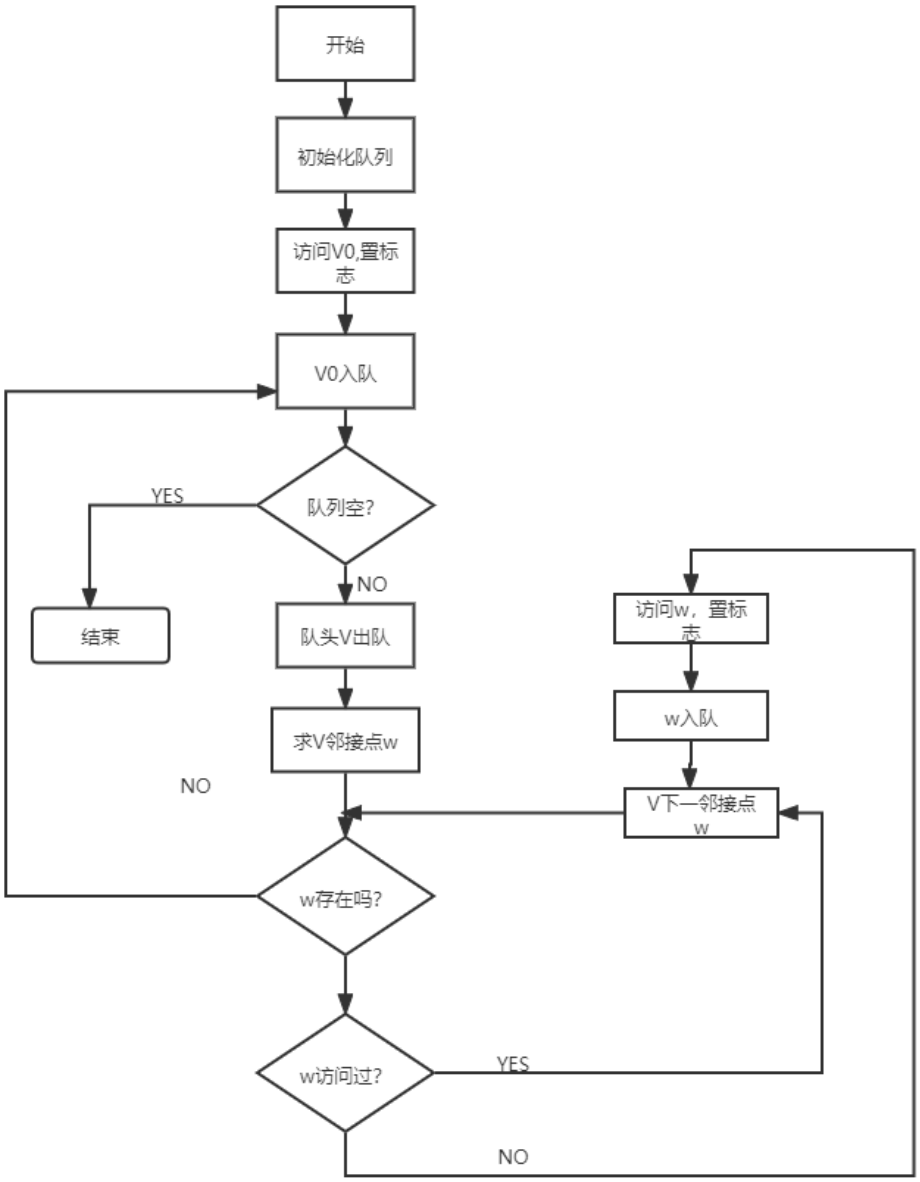
2、图的深度优先搜索遍历类似于树的先根遍历，沿着初始顶点出发的一条路径，尽可能深入地前进，直到所有顶点被访问完；用 visited[] 来存储顶点的访问情况，初始时所有顶点皆为未访问 FALSE，访问一个顶点之后就被标记为已访问 TRUE。

（1）先输入顶点个数以及边数，按照边的数量，依次输入边依附的两个顶点；
（2）构造邻接矩阵的时候，用 1 表示两个顶点相连接，用无穷表示两个顶点未连接；
（3）深度优先遍历图的时候，先找到开始结点，然后找到与此结点相邻的第一个结点，进一步以此结点为开始结点递归；
（4）若没有邻接点，回溯，直至所有结点都被访问；
（5）广度优先遍历，依次访问与开始结点相邻的结点，然后以开始结点相邻的结点进行广度优先遍历。

四、详细设计



Bfs:



五、程序代码

（由于程序过长，请老师查看文件中包含的源代码，这里只放重要部分）

```
#define MaxVertexNum 50
#define QueueSize 50

typedef enum{FALSE,TRUE}define;

typedef char VertexType;   //顶点是字符型
typedef int EdgeType;     //边是整型

define visited[MaxVertexNum];
typedef struct Arcshuzhu{
```

```
    EdgeType adj;/*顶点关系类型，无权图中 1 是相邻 0 是不相邻；带权图中为权值*/
    VertexType *info;
    }Adj[MaxVertexNum][MaxVertexNum];


typedef struct

{
      VertexType vexs[MaxVertexNum];   //顶点向量
      Adj arcs; //邻接矩阵
      int vexnum,arcnum;
      int kind;     //图中当前的顶点数和边数

}Graph;

/*  有向图的建立*/
void CreateDGGraph(Graph *G)
{

      int i,j,k;
      char ch1[5],ch2[5];
      printf("请输入顶点数和边数(输入格式为:顶点数,边数):");
      scanf("%d,%d",&(G->vexnum),&(G->arcnum));
      printf("请输入顶点名称(输入格式为:a,b,c...)：");
      for(i=0;i<G->vexnum;i++)
      {
          getchar();
            scanf("%c",&(G->vexs[i]));
      }
    for(i=0;i<G->vexnum;i++)
        {
         for(j=0;j<G->vexnum;j++)
               {
                 G->arcs[i][j].adj=0;

                 G->arcs[i][j].info=0;
               }
        }
           printf("请输入每条边对应的两个顶点名称(输入格式为:a,b):\n");
           for(k=0;k<G->arcnum;k++)
           {
                 printf("请输入第%d 条边的两个顶点名称：",k+1);
                 scanf("%s%s",ch1,ch2);
```

```
                i=LocateVex(G,ch1);
                i=LocateVex(G,ch2);
                G->arcs[i][j].adj=1;
            }
}

/*  深度优先遍历  */
void Depth(Graph *G,int i)
{
    int j;
     printf("%c\n",G->vexs[i]);     //访问顶点 vi
     visited[i]=TRUE;
     for(j=0;j<G->vexnum;j++)                  //依次搜索 vi 邻接点
          if(G->arcs[i][j]==1 && !visited[j])
               Depth(G,j);
}

void Depthsearch(Graph *G)
{
     int i;
     for(i=0;i<G->vexnum;i++)
          visited[i]=FALSE;
     for(i=0;i<G->vexnum;i++)
          if(!visited[i])
               Depth(G,i);
}

/*广度优先遍历*/
typedef struct
{
     int front;
     int rear;
     int count;
     int data[QueueSize];
}AQueue;

void InitQueue(AQueue *Q)
{
     Q->front=Q->rear=0;
     Q->count=0;
}

int QueueEmpty(AQueue *Q)
{
```

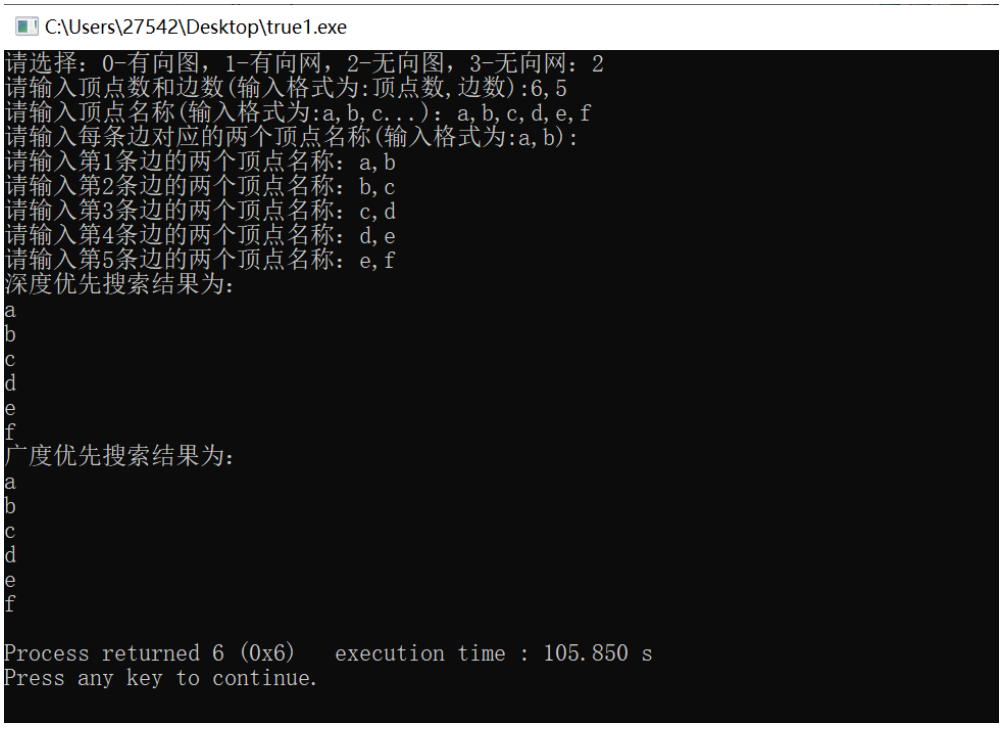
```
	return Q->count=QueueSize;

}

int QueueFull(AQueue *Q)
{

	return Q->count==QueueSize;

}

void EnQueue(AQueue *Q,int x)
{
	if (QueueFull(Q))
		printf("Queue overflow");
	else
	{
		Q->count++;
		Q->data[Q->rear]=x;
		Q->rear=(Q->rear+1)%QueueSize;
	}
}

int DeQueue(AQueue *Q)
{
	int temp;
	if(QueueEmpty(Q))
	{
		printf("Queue underflow");
		return NULL;
	}
	else
	{
		temp=Q->data[Q->front];
		Q->count--;
		Q->front=(Q->front+1)%QueueSize;
		return temp;
	}
}

void Breadth(Graph *G,int k)
{
	int i,j;
	AQueue Q;
```

```
    InitQueue(&Q);
    printf("%c\n",G->vexs[k]);
    visited[k]=TRUE;
    EnQueue(&Q,k);
    while (!QueueEmpty(&Q))
    {
        i=DeQueue(&Q);
        for (j=0;j<G->vexnum;j++)
            if(G->arcs[i][j]==1&&!visited[j])
            {
                printf("%c\n",G->vexs[j]);
                visited[j]=TRUE;
                EnQueue(&Q,j);
            }
    }
}


void Breadthsearch(Graph *G)
{
    int i;
    for (i=0;i<G->vexnum;i++)
        visited[i]=FALSE;
    for (i=0;i<G->vexnum;i++)
        if (!visited[i])
            Breadth(G,i);
}

void main()
{
    Graph *G;
   G=(Graph *)malloc(sizeof(Graph));
   printf("请选择：0-有向图，1-有向网，2-无向图，3-无向网：");
   scanf("%d",&G->kind);
   switch(G->kind)
   {
   case 0:
    CreateDGGraph(G);
    break;
   case 1:
    CreateDNGraph(G);
    break;

   case 2:
```

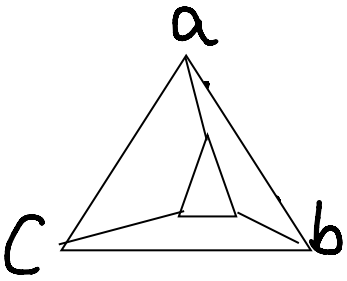
```
        CreateUDGGraph(G);
        break;

        case 3:
        CreateUDNGraph;
        break;

        default:
              break;
      }
      getchar();
        printf("深度优先搜索结果为：");
        printf("\n");
        Depthsearch(&G);
        printf("广度优先搜索结果为：");
        printf("\n");
        Breadthsearch(&G);
}
```

六、测试和结果

1、六边形 abcdef



2、




```
请输入顶点数和边数(输入格式为:顶点数,边数):6,9
请输入顶点名称(输入格式为:a,b,c...): a,b,c,d,e,f
请输入每条边对应的两个顶点名称(输入格式为:a,b):
请输入第1条边的两个顶点名称: a,b
请输入第2条边的两个顶点名称: a,c
请输入第3条边的两个顶点名称: b,c
请输入第4条边的两个顶点名称: a,d
请输入第5条边的两个顶点名称: b,e
请输入第6条边的两个顶点名称: c,f
请输入第7条边的两个顶点名称: d,e
请输入第8条边的两个顶点名称: e,f
请输入第9条边的两个顶点名称: d,f
深度优先搜索结果为:
a
b
c
f
e
d
广度优先搜索结果为:
a
b
c
d
e
f

Process returned 6 (0x6)   execution time : 118.882 s
Press any key to continue.
```

七、用户手册

1、按照文字提示，对应四个序号选择所想要的图的种类；

2、依次输入顶点数、边数、顶点名称、每条边的顶点名称、每条边的权值；

3、本程序没有输入报错提示，若输入的图形在现实中是不存在的、无法画出的，则会直接退出程序，程序无法进行，需要用户重新开始运行程序从头开始输入（需要改进，耗时）；

4、若输入的图形正确，程序自动给出 bfs 和 dfs 算法下的图的遍历；

5、所有输入的数据与数据之间都由“,”相隔，这个“,”是英文中的逗号。