数据结构与算法 第三次实验

学号：3722023220380　　姓名：宋浩元

一、 实验目的

1. 了解分别使用邻接矩阵与邻接表存储图的基础实现方法与原理，理解存储图的基本操作的代码编写方式
2. 学会灵活按照邻接矩阵与邻接表的存储方式自由编写存储结构代码
3. 在成功实现存图的基础上进而理解编写图的基本操作的基础实现方法，理解基本操作的代码编写方式
4. 通过实验探索图的深度优先搜索与广度优先搜索的方法，发现在操作实现上的不同

二、 实验内容

3-1 根据邻接矩阵实现图的基本操作，并设计图的深度优先搜索遍历算法和广度优先搜索遍历算法。

代码编写：

- 邻接矩阵定义

```c
#include<stdio.h>
#include<stdlib.h>
#define MAXVEX 100   //最大定点顶点数
#define INFINITF 65535  //用 65535来表示无穷
#define MAX 100
#define MAXSIZE 100
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
typedef int Boolean;    //Boolean是布尔类型，其值是TRUE或FALSE
Boolean visited[MAX];
typedef struct{
    char vexs[MAXVEX];  //顶点表
    int arc[MAXVEX][MAXVEX];  //邻接矩阵，可看作边表
    int numVertexes,numEdges;  //图中当前的顶点数
    int GraphType; //图的类型
}MGraph;
```

- 建图函数定义

```c
void CreateMGraph (MGraph *G)
{
    int i,j,k,w;
    printf("输入顶点数和边数：\n");
    scanf("%d %d",&G->numVertexes,&G->numEdges); //输入顶点数和边数
    fflush(stdin);
    for(i=0;i<G->numVertexes;i++)
    {
        printf("第%d个顶点：",i+1);
        scanf("%c",&G->vexs[i]);
        getchar();
    }
    for(i=0;i<G->numVertexes;i++)
        for(j=0;j<G->numVertexes;j++)
            G->arc[i][j]=INFINITF ; //邻接矩阵初始化
    for(k=0;k<G->numEdges;k++)
    {
        printf("输入边(vi,vj)上的上标i,下标j和权W:");
        scanf("%d %d %d",&i,&j,&w); //输入边(vi,vj)上的权W
        G->arc[i][j]=w;
        if(G->GraphType==0)
        G->arc[j][i]=G->arc[i][j];  //因为是无向图，矩阵对称
    }
}
```

- 输出邻接矩阵

```c
//输出邻接矩阵
void output(MGraph *G)
{
    int i,j,count=0;
    for (i=0;i<G->numVertexes;i++)
        printf("\t%c",G->vexs[i]);
    printf("\n");
    for(i=0;i<G->numVertexes;i++)
    {
        printf("%4c",G->vexs[i]);
        for(j=0;j<G->numVertexes;j++)
        {
            printf("\t%d",G->arc[i][j]);
            count++;
            if(count%G->numVertexes==0)
            printf("\n");
        }
    }
}
```

- 邻接矩阵的深度优先递归算法

```c
//邻接矩阵的深度优先递归算法
void DFS (MGraph G,int i)
{
    int j;
    visited[i]=TRUE;
    printf("%c ",G.vexs[i]);  //打印顶点
    for(j=0;j<G.numVertexes;j++)
    {
        if(G.arc[i][j]==1&&!visited[j])
        DFS(G,j);  //对未访问的邻接顶点递归调用
    }
}
```

```
void DFSTraverse(MGraph G)
{
    int i;
    for(i=0;i<G.numVertexes;i++)
        visited[i]=FALSE;  //初始化所有顶点状态都是未访问过状态
    for(i=0;i<G.numVertexes;i++)
        if(!visited[i])   //对未访问过的顶点调用DFS，若是连通图，只会执行一次
        DFS(G,i);
}
```

- 创建队列定义

```
typedef struct{
    int data[MAXSIZE];
    int front;     //头指针
    int rear; //尾指针，若队列不空，指向队列尾元素的下一个位置
}SqQueue;
//初始化一个空队列
int InitQueue(SqQueue *Q)
{
    Q->front=0;
    Q->rear=0;
    return OK;
}
int QueueEmpty(SqQueue Q)
{
    if(Q.rear==Q.front)
        return TRUE;
    else
        return FALSE;
}
int EnQueue(SqQueue *Q,int e)
{
    if ((Q->rear+1)%MAXSIZE == Q->front);  //队列满的判断
        return ERROR;
    Q->data[Q->rear]=e;  //将元素e赋值给队尾
    Q->rear=(Q->rear+1)%MAXSIZE; //rear指针向后移一位置,若到最后则转到数组头
    return OK;
}
// 循环队列出队列操作
int DeQueue(SqQueue *Q,int *e)
{
    if (Q->front == Q->rear)
        return ERROR; //队列空的判断
    *e = Q->data[Q->front];  //将队头元素赋值给e
    Q->front=(Q->front+1)%MAXSIZE; //front指针向后移一位置
    return OK;
}
```

- 邻接矩阵的广度遍历操作

```
121  void BFSTraverse(MGraph G)
122  {
123      int i,j;
124      SqQueue Q;
125      for(i=0;i<G.numVertexes;i++)
126          visited[i]=FALSE;
127      InitQueue(&Q);  //初始化一辅助用的队列
128      for(i=0;i<G.numVertexes;i++)  //对每个顶点做循环
129      {
130          if(!visited[i])  //若是未访问过就处理
131          {
132              visited[i]=TRUE;  //设置当前顶点访问过
133              printf("%c ",G.vexs[i]);  //打印顶点
134              EnQueue(&Q,i);   //将此顶点入队列
135              while(!QueueEmpty(Q)) //若当前队列不为空
136              {
137                  DeQueue(&Q,&i); //将队中元素出队列，赋值给i
138                  for(j=0;j<G.numVertexes;j++)
139                  {
140                      //判断其他结点若与当前顶点存在且未访问过
141                      if(G.arc[i][j] == 1&& !visited[j])
142                      {
143                          visited[j]=TRUE;  //将找到的此顶点标记为已访问
144                          printf("%c ",G.vexs[j]);  //打印顶点
145                          EnQueue(&Q,j); //将找到的此顶点入队列
146                      }
147                  }
148              }
149          }
150      }
151  }
```

- 主函数设计

```
152  int main()
153  {
154      MGraph G;
155      int i,j;
156      printf("输入图的类型（无向图0/有向图1）：");
157      scanf("%d",&G.GraphType);
158      CreateMGraph (&G);
159      printf("邻接矩阵数据如下：\n");
160      output(&G);
161      printf("\n");
162      printf("图的深度优先遍历如下:\n");
163      DFSTraverse(G);
164      printf("\n图的广度优先遍历如下：\n");
165      BFSTraverse(G);
166      return 0;
167  }
```

3-2 根据邻接表实现图的基本操作，并设计图的深度优先搜索遍历算法和广度优先搜索遍历算法。

代码编写：

- 定义邻接矩阵类型

```
//定义邻接矩阵类型
typedef struct{
    int no;           //顶点编号
    InfoType info;    //顶点其他信息
}VertexType;         //顶点类型
typedef struct{             //图的定义
    int edges[MAXV][MAXV]; //邻接矩阵
    int vexnum,arcnum;      //顶点数，弧数
    VertexType vexs[MAXV]; //存放顶点信息
}MGraph;                    //图的邻接矩阵类型
```

- 定义邻接表类型

```
//定义邻接表类型
typedef struct ANode{          //弧的结点结构类型
    int adjvex;                //该弧的终点位置
    struct ANode *nextarc;      //指向下一条弧的指针
    InfoType info;              //弧的相关信息,用来存放权值
}ArcNode;
typedef int Vertex;
typedef struct Vnode{       //邻接表头结点的类型
    Vertex data;            //顶点信息
    ArcNode *firstarc;      //指向第一条弧
}VNode;
typedef VNode AdjList[MAXV]; //AdjList是邻接表类型
typedef struct{
    AdjList adjlist;      //邻接表
    int n,e;              //图中顶点数n和边数
}ALGraph;                 //图的邻接表类型
```

- 将邻接矩阵 g 转换邻接表 G

```
//将邻接矩阵g转换邻接表G
void MatToList(MGraph g,ALGraph *&G){
    int i,j,n=g.vexnum;                //n为顶点数
    ArcNode *p;
    G=(ALGraph *)malloc(sizeof(ALGraph));
    for(i=0;i<n;i++)                      //给邻接表中所有头结点的指针域置
        G->adjlist[i].firstarc=NULL;
    for (i=0;i<n;i++)                    //检查邻接矩阵中的每个元素
        for (j=n-1;j>=0;j--)
            if (g.edges[i][j]!=0)        //邻接矩阵的当前元素不为0
            {
                p=(ArcNode *)malloc(sizeof(ArcNode));  //创建一个结点*p
                p->adjvex=j;
                p->info=g.edges[i][j];
                p->nextarc=G->adjlist[i].firstarc;   //将*p链接到链表后面
                G->adjlist[i].firstarc=p;
            }
            G->n=n; G->e=g.arcnum;
}
```

- 将邻接表 G 转换为邻接矩阵 g

```
57  //将邻接表G转换为邻接矩阵g
58  void ListToMat(ALGraph *G,MGraph &g){
59      int i,j,n=G->n;
60      ArcNode *p;
61      for(i=0;i<n;i++)          //g.edges[i][j]赋初值0
62          for(j=0;j<n;j++)
63              g.edges[i][j]=0;
64      for (i=0;i<n;i++)
65      {
66          p=G->adjlist[i].firstarc;
67          while(p!=NULL){
68              g.edges[i][p->adjvex]=p->info;
69              p=p->nextarc;
70          }
71      }
72      g.vexnum=n; g.arcnum=G->e;
73
74  }
```

- 输出邻接矩阵 g

```
76  //输出邻接矩阵g
77  void DispMat(MGraph g){
78      int i,j;
79      for (i=0;i<g.vexnum;i++)
80      {
81          for(j=0;j<g.vexnum;j++)
82              if(g.edges[i][j]==INF)
83                  printf("%3s","∞");
84              else
85                  printf("%3d",g.edges[i][j]);
86          printf("\n");
87      }
88
89  }
```

- 输出邻接表 G

```
91  //输出邻接表G
92  void DispAdj(ALGraph *G){
93      int i;
94      ArcNode *p;
95      for (i=0;i< G->n;i++)
96      {
97          p=G->adjlist[i].firstarc;
98          if(p!=NULL) printf("%3d: ",i);
99          while (p!=NULL)
100         {
101             printf("%3d",p->adjvex);
102             p=p->nextarc;
103         }
104         printf("\n");
105
106     }
107
108 }
```

- 连通图的深度优先遍历

```c
int visited[MAXV];
//连通图的深度优先遍历
void DFS(ALGraph *G,int v){ //G指向某个邻接表，v是起始顶点
    ArcNode *p;
    visited[v]=1;                       //已访问，则置1
    printf("%3d",v);                    //输出被访问顶点的编号
    p=G->adjlist[v].firstarc;           //p指向顶点v的第一条弧的弧头结点
    while (p!=NULL)
    {
        if(visited[p->adjvex]==0)       //若p->adjvex顶点未访问，递归访问它
            DFS(G,p->adjvex);
        p=p->nextarc;                   //p指向顶点v的下一条弧的弧头结点
    }

}
```

- 连通图的深度优先遍历(非递归)

```c
//连通图的深度优先遍历(非递归)
void DFS1(ALGraph *G,int v){
    ArcNode *p;
    ArcNode *St[MAXV];
    int top=-1,w,i;
    for(i=0;i< G->n;i++)
        visited[i]=0;
    printf("%3d",v);
    visited[v]=1;
    top++;
    St[top]=G->adjlist[v].firstarc;
    while (top > -1)
    {
        p=St[top]; top--;
        while (p!=NULL)
        {
            w=p->adjvex;
            if (visited[w]==0)
            {
                printf("%3d",w);
                visited[w]=1;
                top++;
                St[top]=G->adjlist[w].firstarc;
                break;
            }
            p=p->nextarc;
        }

    }
    printf("\n");
}
```

- 连通图的广度优先遍历

```c
//连通图的广度优先遍历
void BFS(ALGraph *G,int v){ //G指向某个邻接表,v是起始顶点
    ArcNode *p;
    int queue[MAXV],front=0,rear=0;       //定义循环队列并初始化
    int visited[MAXV];                    //定义存放结点的访问标志的数组
    int w,i;
    for(i=0;i< G->n;i++) visited[i]=0;    //访问标志数组初始化
    printf("%3d",v);                      //输出被访问顶点的编号
    visited[v]=1;                         //置已访问标记
    rear=(rear+1)%MAXV;
    queue[rear]=v;                       //v进队
    while (front!=rear)                   //若队列不空时循环
    {
        front=(front+1)%MAXV;
        w=queue[front];                  //出队并赋给w
        p=G->adjlist[w].firstarc;        //找与顶点w邻接的第一个顶点
        while (p!=NULL)
        {
            if (visited[p->adjvex]==0)  //若当前邻接顶点未访问
            {
                printf("%3d",p->adjvex); //访问相邻顶点
                visited[p->adjvex]=1;    //置该顶点已被访问的标志
                rear=(rear+1)%MAXV;      //该顶点进队
                queue[rear]=p->adjvex;
            }
            p=p->nextarc;               //找下一个邻接顶点
        }
    }
    printf("\n");
}
```

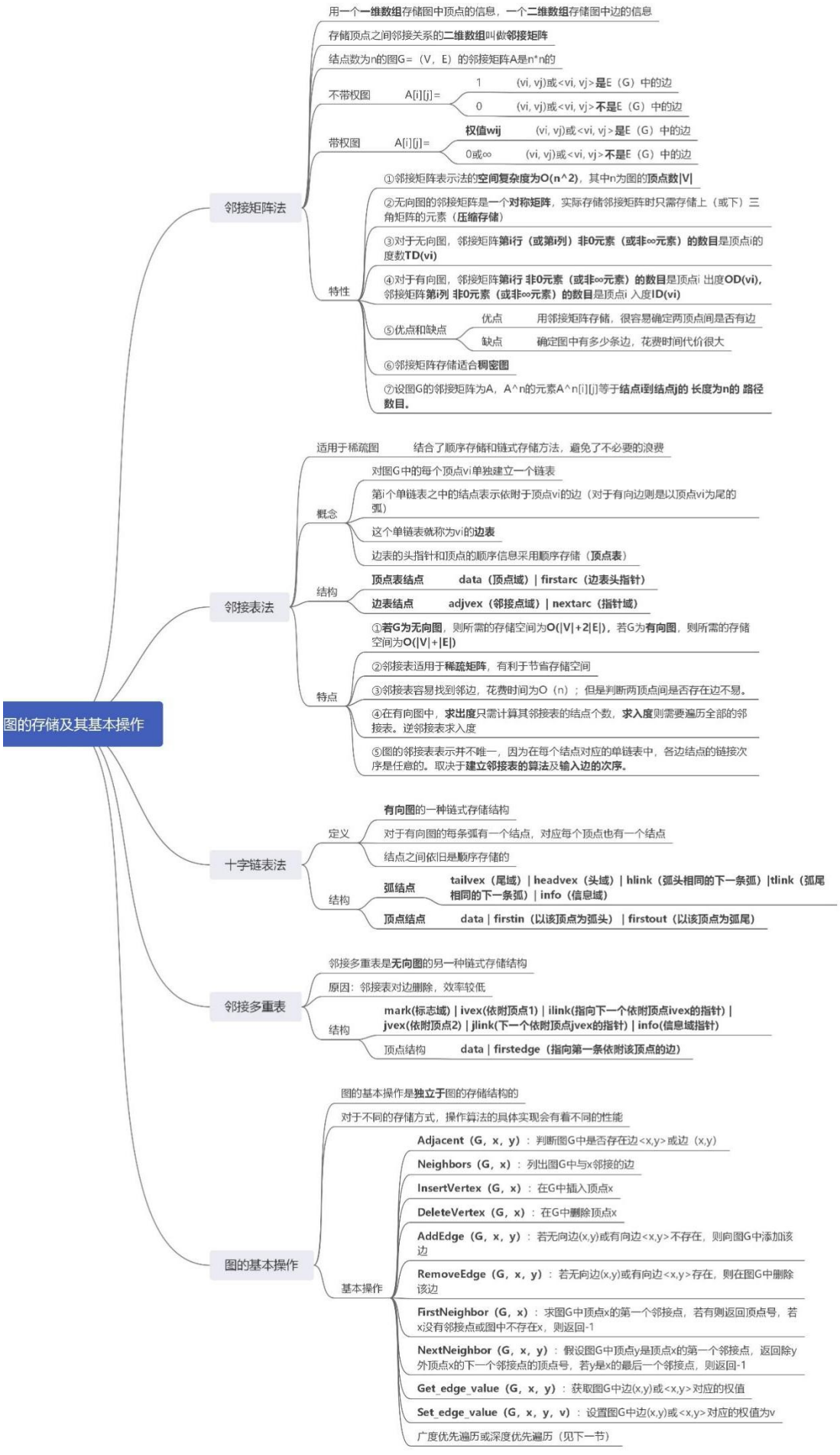
- **主函数编写**

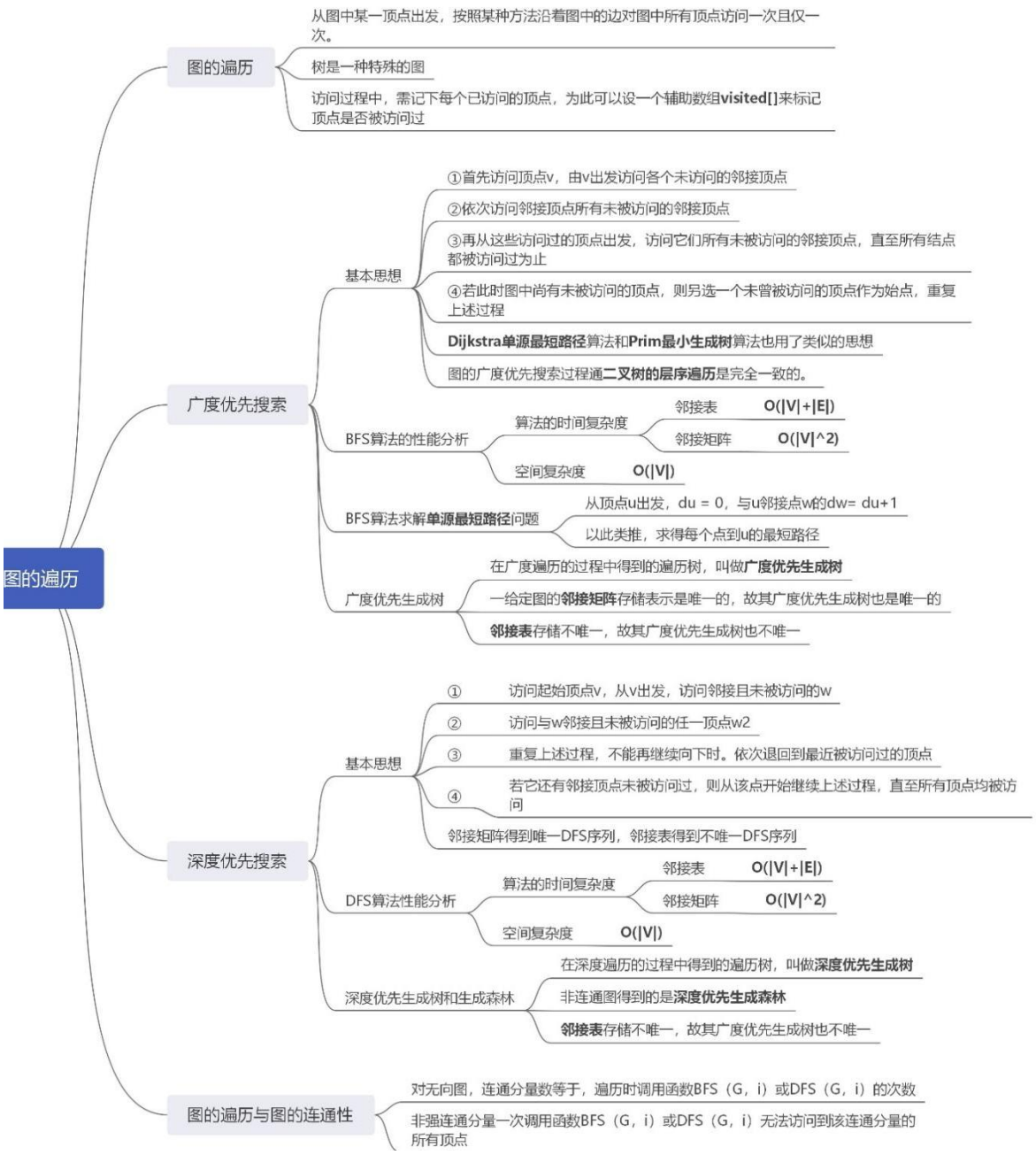
```c
int main()
{
    int i,j;
    MGraph g;
    ALGraph *G;
    int A[6][6]={
        {0,5,0,5,5,0},
        {0,0,4,0,4,0},
        {0,0,0,0,0,9},
        {0,0,0,0,0,0},
        {0,0,5,7,0,7},
        {0,0,0,0,0,0}
    };

    g.vexnum=6; g.arcnum=10;
    for (i=0;i<g.vexnum;i++)
        for(j=0;j<g.vexnum;j++)
            g.edges[i][j]=A[i][j];
    printf("\n");
    printf(" 有向图G的邻接矩阵: \n");
    DispMat(g);

    G=(ALGraph *)malloc(sizeof(ALGraph));
    printf(" 图G的邻接矩阵转换为邻接表: \n");
    MatToList(g,G);
    DispAdj(G);

    printf("深度优先遍历: \n");
    DFS(G,0);
    printf("\n");
    printf("广度优先遍历: \n");
    BFS(G,0);
    printf("\n");
}
```

三、 主要算法流程图

3-1 与 3-2 算法流程

(图片太小可见目录下的 3-1.2 邻接矩阵与邻接表.png)


图的存储及其基本操作
邻接矩阵法
用一个一维数组存储图中顶点的信息，一个二维数组存储图中边的信息
存储顶点之间邻接关系的二维数组叫做邻接矩阵
结点数为n的图G=（V，E）的邻接矩阵A是n*n的
不带权图
A[i][j]=
1
(vi, vj)或<vi, vj>是E（G）中的边
0
(vi, vj)或<vi, vj>不是E（G）中的边
带权图
A[i][j]=
权值wij
(vi, vj)或<vi, vj>是E（G）中的边
0或∞
(vi, vj)或<vi, vj>不是E（G）中的边
特性
①邻接矩阵表示法的空间复杂度为O(n^2)，其中n为图的顶点数|V|
②无向图的邻接矩阵是一个对称矩阵，实际存储邻接矩阵时只需存储上（或下）三角矩阵的元素（压缩存储）
③对于无向图，邻接矩阵第i行（或第i列）非0元素（或非∞元素）的数目是顶点i的度数TD(vi)
④对于有向图，邻接矩阵第i行 非0元素（或非∞元素）的数目是顶点i 出度OD(vi)，邻接矩阵第i列 非0元素（或非∞元素）的数目是顶点i 入度ID(vi)
⑤优点和缺点
优点
用邻接矩阵存储，很容易确定两顶点间是否有边
缺点
确定图中有多少条边，花费时间代价很大
⑥邻接矩阵存储适合稠密图
⑦设图G的邻接矩阵为A，A^n的元素A^n[i][j]等于结点i到结点j的 长度为n的 路径数目。
邻接表法
适用于稀疏图
结合了顺序存储和链式存储方法，避免了不必要的浪费
概念
对图G中的每个顶点vi单独建立一个链表
第i个单链表之中的结点表示依附于顶点vi的边（对于有向边则是以顶点vi为尾的弧）
这个单链表就称为vi的边表
边表的头指针和顶点的顺序信息采用顺序存储（顶点表）
结构
顶点表结点
data（顶点域）| firstarc（边表头指针）
边表结点
adjvex（邻接点域）| nextarc（指针域）
特点
①若G为无向图，则所需的存储空间为O(|V|+2|E|)，若G为有向图，则所需的存储空间为O(|V|+|E|)
②邻接表适用于稀疏矩阵，有利于节省存储空间
③邻接表容易找到邻边，花费时间为O（n）；但是判断两顶点间是否存在边不易。
④在有向图中，求出度只需计算其邻接表的结点个数，求入度则需要遍历全部的邻接表。逆邻接表求入度
⑤图的邻接表表示并不唯一，因为在每个结点对应的单链表中，各边结点的链接次序是任意的。取决于建立邻接表的算法及输入边的次序。
十字链表法
定义
有向图的一种链式存储结构
对于有向图的每条弧有一个结点，对应每个顶点也有一个结点
结点之间依旧是顺序存储的
结构
弧结点
tailvex（尾域）| headvex（头域）| hlink（弧头相同的下一条弧）|tlink（弧尾相同的下一条弧）| info（信息域）
顶点结点
data | firstin（以该顶点为弧头）| firstout（以该顶点为弧尾）
邻接多重表
邻接多重表是无向图的另一种链式存储结构
原因：邻接表对边删除，效率较低
结构
mark(标志域) | ivex(依附顶点1) | ilink(指向下一个依附顶点ivex的指针) | jvex(依附顶点2) | jlink(下一个依附顶点jvex的指针) | info(信息域指针)
顶点结构
data | firstedge（指向第一条依附该顶点的边）
图的基本操作
图的基本操作是独立于图的存储结构的
对于不同的存储方式，操作算法的具体实现会有着不同的性能
基本操作
Adjacent（G，x，y）：判断图G中是否存在边<x,y>或边（x,y）
Neighbors（G，x）：列出图G中与x邻接的边
InsertVertex（G，x）：在G中插入顶点x
DeleteVertex（G，x）：在G中删除顶点x
AddEdge（G，x，y）：若无向边(x,y)或有向边<x,y>不存在，则向图G中添加该边
RemoveEdge（G，x，y）：若无向边(x,y)或有向边<x,y>存在，则在图G中删除该边
FirstNeighbor（G，x）：求图G中顶点x的第一个邻接点，若有则返回顶点号，若x没有邻接点或图中不存在x，则返回-1
NextNeighbor（G，x，y）：假设图G中顶点y是顶点x的第一个邻接点，返回除y外顶点x的下一个邻接点的顶点号，若y是x的最后一个邻接点，则返回-1
Get_edge_value（G，x，y）：获取图G中边(x,y)或<x,y>对应的权值
Set_edge_value（G，x，y，v）：设置图G中边(x,y)或<x,y>对应的权值为v
广度优先遍历或深度优先遍历（见下一节）



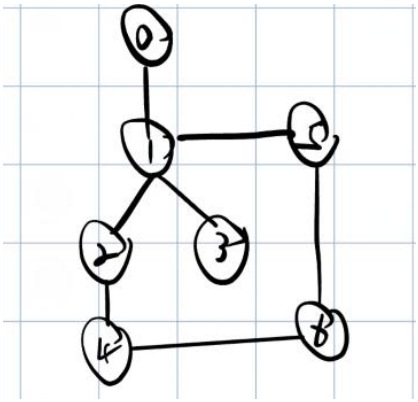
四、 实验结果：

（结合截图说明算法的输入输出）

1、关于 3-1 的输入与输出：

```
输入图的类型（无向图0/有向图1）：0
输入顶点数和边数：
7 7
第1个顶点：0
第2个顶点：1
第3个顶点：2
第4个顶点：3
第5个顶点：4
第6个顶点：5
第7个顶点：6
输入边(vi,vj)上的上标i,下标j和权W:0 1 1
输入边(vi,vj)上的上标i,下标j和权W:1 2 1
输入边(vi,vj)上的上标i,下标j和权W:1 3 1
输入边(vi,vj)上的上标i,下标j和权W:2 4 1
输入边(vi,vj)上的上标i,下标j和权W:4 6 1
输入边(vi,vj)上的上标i,下标j和权W:5 6 1
输入边(vi,vj)上的上标i,下标j和权W:1 5 1
邻接矩阵数据如下：
        0       1       2       3       4       5       6
   0    65535   1       65535   65535   65535   65535   65535
   1    1       65535   1       1       65535   1       65535
   2    65535   1       65535   65535   1       65535   65535
   3    65535   1       65535   65535   65535   65535   65535
   4    65535   65535   1       65535   65535   65535   1
   5    65535   1       65535   65535   65535   65535   1
   6    65535   65535   65535   65535   1       1       65535

图的深度优先遍历如下:
0 1 2 4 6 5 3
图的广度优先遍历如下：
0 1 2 3 4 5 6
--------------------------------

Process exited after 44.18 seconds with return value 0
请按任意键继续. . .
```

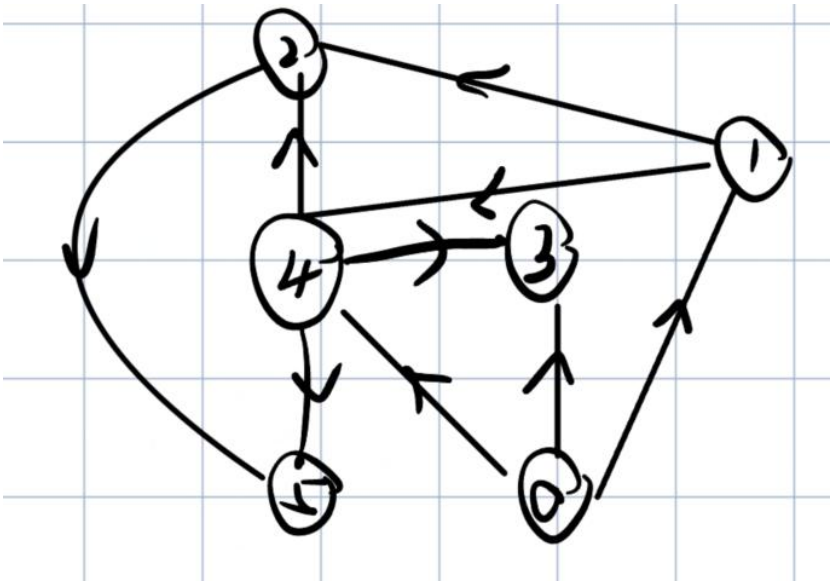
在实际运行中，我创建了如下的图，并实现了正确的遍历：



2、关于 3-2 的输入与输出

```
有向图G的邻接矩阵:
 0  5  0  5  5  0
 0  0  4  0  4  0
 0  0  0  0  0  9
 0  0  0  0  0  0
 0  0  5  7  0  7
 0  0  0  0  0  0
图G的邻接矩阵转换为邻接表:
 0:   1  3  4
 1:   2  4
 2:   5

 4:   2  3  5

深度优先遍历:
 0  1  2  5  4  3
广度优先遍历:
 0  1  3  4  2  5


--------------------------------
Process exited after 0.04274 seconds with return value 0
请按任意键继续. . .
```

在实际运行中，我创建了如下的图，并实现了正确的遍历:



五、 实验小结（即总结本次实验所得到的经验与启发等）:

在本次实验中，我尝试具体运用了图，在实体机的实验中我能够更深刻地理解对这一部分数据结构的执行方式与特点，并且在编写代码的过程中，我通过不断的调试去寻找语句之间的问题和不足，在潜移默化中提高了我的代码编写能力，这是一次完成效果良好的实验!