

图

代码题 0

题干

- 题目：
 - 图的存储结构
- 总结：
 - (1) 邻接表
 - 1. 邻接表的结构;
 - 2. 结点的结构;
 - 3. 边的结构;
 - (2) 邻接矩阵

解答：邻接表

```
typedef struct AGraph
{
    int vertexNum = 0; // 顶点的个数
    int edgeNum = 0; // 边的个数
    VNode adjlist[maxsize]; // 所有图中结点
}AGraph;

typedef struct VNode
{
    char data;
    ArcNode* firstArc;
}VNode;

typedef struct ArcNode
{
    int adjVertex; // 此边所指向的结点位置
    struct ArcNode* NextArc;
}ArcNode;
```

解答：邻接矩阵

```
typedef struct MGraph
{
    int vertexNum = 0;
    int arcNum = 0;
    int Edge[maxsize][maxsize]; // 边的信息
    char verticle[maxsize]; // 存储顶点中的值
```

```
}MGraph;
```

解答：邻接表的插入 - 无向图

```
AGraph* Insert(int vertexNum,int edgeNum)
{
    AGraph* g = new AGraph;
    g->vertexNum = vertexNum;
    g->edgeNum = edgeNum;

    for(int i =0;i<vertexNum;i++)
    {
        g->adjlist[i].firstArc = null;
    }

    for(int i = 0;i<edgeNum;i++)
    {
        int v1,v2;

        cin>>v1;
        cin>>v2;

        // 将边结点插在v1结点的边链表中
        ArcNode* newNode1 = new ArcNode;
        newNode1->adjVertex = v2;
        newNode1->NextArc = g->adjlist[v1].firstArc;
        g->adjlist[v1].firstArc = newNode1;

        // 将边结点插在v2结点的边链表中
        ArcNode* newNode2 = new ArcNode;
        newNode2->adjVertex = v1;
        newNode2->NextArc = g->adjlist[v2].firstArc;
        g->adjlist[v2].firstArc = newNode2;
    }

    return G;
}
```

代码题 1

题干

- 题目：

1、已知无向连通图 G 由顶点集 V 和边集 E 组成， $|E|>0$ ，当 G 中度为奇数的顶点个数为不大于 2 的偶数时， G 存在包含所有边且长度为 $|E|$ 的路径（称为 EL 路径）。设图 G 采用邻接矩阵存储，设计算法判断图中是否存在 EL 路径，若存在返回 1，否则返回 0。

- 总结：
 - 判断每个结点的边的度，将度求和后求模数；

解答

```
int IsEL(MGraph* G)
{
    int count = 0;
    int degree;

    for(int i = 0; i < G.vertexNum; ++i)
    {
        degree = 0;

        for(int j = 0; j < G.vertexNum; ++j)
        {
            degree += G.Edge[i][j];
        }

        if(degree % 2 == 1)
        {
            count++;
        }
    }

    // 不大于2的偶数
    if(count == 0 || count == 2)
    {
        return 1;
    }

    return 0;
}
```

代码题 2

题干

- 题目：
 - 2、图的广度优先遍历（BFS）

- 总结:
 - 第一步: 新建队列
 - 第二步: 初始化 `visit` 数组
 - 第三步: 访问开始结点
 - 第四步: 进入两层循环
 - 1. 外层循环利用队列, 访问所有结点 (`G->adjlist[v]`);
 - 2. 内层循环访问链表, 访问当前结点的所有边;

解答

```
// G 表示当前的图
// v 表示当前BFS遍历从哪一个顶点开始
// visit[] 表示当前顶点是否被访问过
void BFS(AGraph* G,int v,int visit[])
{
    int queue[maxsize];
    int rear = 0;
    int front = 0;

    // 初始化visit数组
    for(int i = 0;i<G->vertexNum;i++)
    {
        visit[i] = 0;
    }

    // 将当前结点入队, 访问当前结点
    queue[++rear] = v;
    visit[v] = 1;
    cout<< v << " ";

    // 当前队列不为空, 则选择结点、遍历
    while(rear!=front)
    {
        // 队头元素出队
        v = queue[++front];
        ArcNode* p = G->adjlist[v].firstArc;

        // 对当前队头结点的所有边遍历, 访问其visit数组, 如果未被遍历过, 则访问+输出
        while(p!=null)
        {
            // 如果当前边连接的结点还未访问
            // 注意, p是边结点指针, 要访问其对应的结点编号, 要是p->adjVertex
            if(visit[p->adjVertex] == 0)
            {
                visit[p->adjVertex] = 1;
                cout<< p->adjVertex << " ";
                queue[++rear] = p->adjVertex;
            }
            p = p->NextArc;
        }
    }
}
```

```
    }  
  }  
}
```

代码题 3

题干

- 题目：
 - **3、利用 BFS 求无向图的最短路径**
- 总结：
 - 和 BFS 基本一致，但在 BFS 基础上增加了一个 dist 数组，如果当前边所对应的结点尚未访问过，则其 dist 为当前 p 的距离加上 1；
 - 其实就是把 BFS 中，输出路径的动作替换为了更新 dist 数组；

解答

```
void BFS(AGraph* G,int v,int visit[],int dist[])  
{  
    int queue[maxsize];  
    int rear = 0;  
    int front = 0;  
  
    // 初始化visit数组  
    for(int i = 0;i<G->vertexNum;i++)  
    {  
        visit[i] = 0;  
    }  
  
    // 初始化dist数组  
    for(int i = 0;i<G->vertexNum;i++)  
    {  
        dist[i] = INT16_MAX;  
    }  
  
    // 将当前结点入队，将其距离设为0  
    queue[++rear] = v;  
    visit[v] = 1;  
    dist[v] = 0;  
  
    // 当前队列不为空，则选择结点、遍历  
    while(rear!=front)  
    {  
        // 队头元素出队  
        v = queue[++front];  
        ArcNode* p = G->adjlist[v].firstArc;  
  
        // 对当前队头结点的所有边遍历，访问其visit数组，如果未被遍历过，则访问+输出  
        while(p!=null)
```

```

    {
        // 如果当前边连接的结点还未访问
        if(visit[p->adjVertex] == 0)
        {
            visit[p->adjVertex] = 1;

            // 将和当前v结点相连的结点的距离，在v的基础上+1
            dist[p->adjVertex] = dist[v] +1 ;
            queue[++rear] = p->adjVertex;
        }
        p = p->NextArc;
    }
}
}

```

代码题 4

题干

- 题目：

• 4、图的深度优先遍历 (DFS)

- 总结：

- 和树的递归类似，只不过在树里面、对左右子树的递归是直接写两个函数，分别传入左右孩子；
- 而 **DFS** 中是需要使用一个循环，分别对与当前结点连接边的链表中，对连接的结点进行 **DFS**；

解答

```

// visit数组已经初始化
// vertex表示DFS的开始点
void DFS(AGraph* G,int v,int visit[])
{
    // 输出顶点的编号，表示对其遍历
    cout<<v;
    visit[v] = 1;
    ArcNode* p = G->adjlist[v].firstArc;

    while(p!=null)
    {
        // 当前节点还未被访问，对其递归
        if(visit[p->adjVertex]==0)
        {
            DFS(G,p->adjVertex,visit[]);
        }

        p = p->netxArc;
    }
}

```

```
}  
}
```

代码题 5

题干

- 题目:

5、图采用邻接表存储，设计算法判断 i 和 j 结点之间是否有路径（以下全是邻接表存储）

- 总结:

- 以 i 结点为起始点，进行一次 DFS。如果 j 结点和 i 之间有路径，则 j 结点肯定会被访问，即 `visit[j]==1`，如果 `visit[j]==0`，则表示无路径

解答

```
// visit数组已经初始化  
bool IsPath(AGraph* G,int i,int j)  
{  
    int visit[maxsize];  
  
    for(int i =0;i<G->vertexNum;i++)  
    {  
        visit[i] = 0;  
    }  
  
    DFS(G,i,visit[]);  
    if(visit[j]==1)  
    {  
        return 1;  
    }  
    else  
    {  
        return 0;  
    }  
}  
  
void DFS(AGraph* G,int vertex,int visit[])  
{  
    // 输出顶点的编号，表示对其遍历  
    cout<<vertex;  
    visit[v] = 1;  
    ArcNode* p = G->adjlist[vertex].firstArc;  
  
    while(p!=null)  
    {  
        // 当前节点还未被访问，对其递归  
        if(visit[p->adjVertex]==0)  
        {
```

```

        DFS(G,p.adjVertex,visit[]);
    }

    p = p->netxArc;
}
}

```

代码题 6

题干

- 题目：
 - **6、设计算法判断无向图是否是一棵树**
- 总结：
 - 树的两个条件：
 - (1) 是一个连通图
 - (2) 结点数 - 边数 = 1
 - 因此，可以用一个 `vm` 来计算当前图 DFS 遍历到的节点个数，如果等于结点数，则表示为连通图；
 - 每进入一次 DFS，表示遍历到一个结点，将 `vm++`

解答

```

void DFS(AGraph* G,int vertex,int& vm,int visit[])
{
    visit[vertex] = 1;
    vm++;
    ArcNode* p = G->adjlist[vertex].firstArc;

    while(p!=null)
    {
        if(visit[p->adjVertex] == 0)
        {
            DFS(G,p->adjVertex,vm,visit[]);
        }
        p = p->nextArc;
    }
}

bool IsTree(AGraph G)
{
    int vn = 0;
    int visit[maxsize];
    for(int i =0;i<G->vertexNum;i++)
    {
        visit[i] = 0;
    }
}

```

```

DFS(G,0,vm,visit[]);

if((vm == G->vertexNum) && (G->vertexNum - 1 == G->edgeNum))
{
    return true;
}
else
{
    return false;
}
}

```

代码题 7

题干

- 题目：

7、设计算法，求无向连通图距顶点 v 最远的一个结点（即路径长度最大）

- 总结：

- 对顶点 v 进行 BFS，最后一个出队的元素就是最远的元素；

解答

```

// G 表示当前的图
// v 表示当前BFS遍历从哪一个顶点开始
// visit[] 表示当前顶点是否被访问过
void BFS(AGraph* G,int v,int visit[])
{
    int queue[maxsize];
    int rear = 0;
    int front = 0;

    // 初始化visit数组
    for(int i = 0;i<G->vertexNum;i++)
    {
        visit[i] = 0;
    }

    // 将当前结点入队，访问当前结点
    queue[++rear] = v;
    visit[v] = 1;
    cout<< v << " ";

    // 当前队列不为空，则选择结点、遍历
    while(rear!=front)
    {
        // 队头元素出队
    }
}

```

```

v = queue[++front];
ArcNode* p = G->adjlist[v].firstArc;

// 对当前队头结点的所有边遍历，访问其visit数组，如果未被遍历过，则访问+输出
while(p!=null)
{
    // 如果当前边连接的结点还未访问
    // 注意，p是边结点指针，要访问其对应的结点编号，要是p->adjVertex
    if(visit[p->adjVertex] == 0)
    {
        visit[p->adjVertex] = 1;
        cout<< p->adjVertex << " ";
        queue[++rear] = p->adjVertex;
    }
    p = p->NextArc;
}

return v;
}

```

代码题 8

题干

- 题目：
 - 8、写出深度优先遍历的非递归算法
- 总结：
 - 使用 visit 数组来记录结点是否被访问过，使用栈来存储一次 DFS 路径上的结点；
 - 第一步：初始化 visit 和栈
 - 第二步：将当前结点访问并入栈
 - 第三步：进入循环
 - 外循环：是否栈空，如果不为空，则取出栈顶元素；
 - 内循环：寻找当前结点所连接的边中，未被访问的结点，将其访问并入栈

解答

```

void DFS(AGraph* G,int v)
{
    int visit[maxsize];
    for(int i =0;i<G->vertexNum;i++)
    {
        visit[i] = 0;
    }

    int stack[maxsize];
    int top = -1;
}

```

```

cout<< v <<" ";
visit[v] = 1;
stack[++top] = v;

while(top!=-1)
{
    // 已经入栈过的元素，都是已访问过的，因此出站后不需要对v进行cout
    v = stack[top--];
    ArcNode* p = G->adjlist[v].firstArc;

    while(p!=null && visit[p->adjVertex] == 1)
    {
        p = p->nextArc;
    }

    // 跳出内层循环后，p有两种可能，一种是找到了一个和v相连的结点中、尚未被访问过的结
    // 点；
    // 还有一种可能，便是p走到了边链表的尾部，当前p为null，则此时要出栈、返回DFS的上
    // 一层

    if(p == null)
    {
        top--;
    }
    else
    {
        cout<<p->vertexNum<<" ";
        visit[p->vertexNum] = 1;
        stack[++top] = p->vertexNum;
    }
}
}

```

代码题 10

题干

- 题目：
 - 10、求无向图的连通分量个数：
- 总结：
 - DFS 的次数即为连通分量个数；

解答

```

int Fun(AGraph* G)
{
    int count = 0;
    int visit[maxsize];

    for(int i =0;i<G->vertexNum;i++)

```

```

{
    visit[i] = 0;
}

for(int i =0;i<G->vertexNum;i++)
{
    if(visit[i] == 0)
    {
        DFS(G,v,visit[]);
        count++;
    }
}

return count;
}

```

代码题 11

题干

- 题目：
 - **11、邻接表转化成邻接矩阵**
- 总结：
 - 遍历邻接表所有点，对其边链表进行访问，转化为邻接矩阵；

解答

```

void Convert(MGraph g1,AGraph* g2)
{
    g1.vertexNum = g2->vertexNum;
    g1.edgeNum = g2->vertexNum;

    for(int i =0;i<g2->vertexNum;i++)
    {
        ArcNode* p = g2->adjlist[i].firstArc;
        while(p!=null)
        {
            g1.Edge[i][p->adjVertex] = 1;
            p = p->nextArc;
        }
    }
}

```

代码题 13

题干

- 题目：

• 13、邻接矩阵转化成邻接表：

- 总结：

-

解答

```
void Fun(MGraph g1, AGraph* g2)
{
    g2->vertexNum = g1.vertexNum;
    g2->edgeNum = g1.vertexNum;

    for(int i = 0; i < g1.vertexNum; i++)
    {
        g2->adjlist[i] = null;
    }

    for(int i = 0; i < g1.vertexNum; i++)
    {
        for(int j = 0; j < g1.vertexNum; j++)
        {
            if(g1.Edge[i][j] == 1)
            {
                ArcNode* p = new ArcNode;
                p->adjVertex = j;
                p->nextArc = g2->adjlist[i].firstArc;
                g2->adjlist[i].firstArc = p;
            }
        }
    }
}
```

获取笔记