# 图

吴益强

C 版

日期：2022 年 4 月 30 日

## 1 图

### 1.1 图的定义

1) 图由顶点集合和边集合组成每条边连接两个不同顶点

2) 有向图：边有方向有起点和终点

3) 无向图：边没有方向

```
无向图连接顶点u,v 的边，记为 (u,v)
有向图连接顶点u,v 的边，记为 <u,v>
无向图中边(u,v) 存在，称 u,v 相邻 ,u,v 互为邻点
有向图中边<u,v> 存在，称 v 是 u 的邻点
```

4) 边只是逻辑上表示两个顶点有直接关系，边是直的还是弯的，边有没有交叉，都没有意义。

5) 无向图两个顶点之间最多一条边有向图两个顶点之间最多两条方向不同的边

### 1.2 图相关概念

```
1)顶点的度数：和顶点相连的边的数目。
2)顶点的出度：有向图中，以该顶点作为起点的边的数目
3)顶点的入度：有向图中，以该顶点作为终点的边的数目
4)顶点的出边：有向图中，以该顶点为起点的边
5)顶点的入边：有向图中，以该顶点为终点的边
6)路径：路径，比如在无向图G中，存在一个顶点序列Vp,Vi1,Vi2,Vi3…，Vim，Vq，使得(
    Vp,Vi1)，(Vi1,Vi2)，…,(Vim,Vq)均属于边集E(G)，则称顶点Vp到Vq存在一条路径。
    回路（环）：起点和终点相同的路径
7)简单路径：除了起点和终点可能相同外，其它顶点都不相同的路径
8)完全图：
完全无向图：任意两个顶点都有边相连
完全有向图：任意两个顶点都有两条方向相反的边
9)连通：如果存在从顶点 u 到顶点 v 的路径，则称 u 到 v 连通，或 u 可达 v 。无向
    图中， u 可达 v，必然 v 可达 u 。有向图中， u 可达 v ，并不能说明 v 可达 u
    。
10)连通无向图：图中任意两个顶点 u 和 v 互相可达。
11)强连通有向图：图中任意两个顶点 u 和 v 互相可达。
```

12)子图：从图中抽取部分或全部边和点构成的图
13)连通分量（极大连通子图）：无向图的一个子图，是连通的，且再添加任何一些原图中的顶点和边，新子图都不再连通。
14)强连通分量：有向图的一个子图，是强连通的，且再添加任何一些原图中的顶点和边，新子图都不再强连通。
15)带权图：边被赋予一个权值的图
16)网络：带权无向连通图

## 1.3 图的性质

1. 图的边数等于顶点度数之和的一半

2.n 个节点的连通图至少有 n-1 条边

3.n 个节点的，无回路的连通图就是一棵树，有 n 1 条边

## 1.4 图的表示方法（课上图示）

1、邻接矩阵

```
typedef struct GNode *PtrToGNode;
struct GNode{
    int Nv;  /* 顶点数 */
    int Ne;  /* 边数   */
    WeightType G[MaxVertexNum][MaxVertexNum]; /* 邻接矩阵 */
};
typedef PtrToGNode MGraph; /* 以邻接矩阵存储的图类型 */
```

2、邻接表

```
/* 邻接点的定义 */
typedef struct AdjVNode *PtrToAdjVNode;
struct AdjVNode{
    Vertex AdjV;        /* 邻接点下标 */
    PtrToAdjVNode Next; /* 指向下一个邻接点的指针 */
};
/* 顶点表头结点的定义 */
typedef struct Vnode{
    PtrToAdjVNode FirstEdge; /* 边表头指针 */
} AdjList[MaxVertexNum];     /* AdjList是邻接表类型 */

/* 图结点的定义 */
typedef struct GNode *PtrToGNode;
struct GNode{
    int Nv;     /* 顶点数 */
    int Ne;     /* 边数   */
    AdjList G;  /* 邻接表 */
};
typedef PtrToGNode LGraph; /* 以邻接表方式存储的图类型 */
```

## 1.5 图的遍历 DFS 和 BFS

1、DFS

从起点出发，走过的点要做标记，发现有没走过的点，就随意挑一个往前走，走不了就回退，此种路径搜索策略就称为深度优先搜索，简称深搜。

其实称为远度优先搜索更容易理解些。因为这种策略能往前走一步就往前走一步，总是试图走得更远。所谓远近或深度，就是以距离起点的步数来衡量的。

邻接矩阵的 DFS

```
#include <stdio.h>

typedef enum {false, true} bool;
#define MaxVertexNum 10  /* 最大顶点数设为10 */
#define INFINITY 65535   /*  设为双字节无符号整数的最大值65535*/
typedef int Vertex;      /* 用顶点下标表示顶点,为整型 */
typedef int WeightType;  /* 边的权值设为整型 */

typedef struct GNode *PtrToGNode;
struct GNode{
    int Nv;  /* 顶点数 */
    int Ne;  /* 边数   */
    WeightType G[MaxVertexNum][MaxVertexNum]; /* 邻接矩阵 */
};
typedef PtrToGNode MGraph; /* 以邻接矩阵存储的图类型 */
bool Visited[MaxVertexNum]; /* 顶点的访问标记 */

MGraph CreateGraph(); /* 创建图并且将Visited初始化为false；裁判实现，细节不表
    */

void Visit( Vertex V )
{
    printf(" %d", V);
}

void DFS( MGraph Graph, Vertex V, void (*Visit)(Vertex) );


int main()
{
    MGraph G;
    Vertex V;

    G = CreateGraph();
    scanf("%d", &V);
    printf("DFS from %d:", V);
    DFS(G, V, Visit);
```

```
    return 0;
}

/*DFS代码将被嵌在这里 */
```

DFS 代码:

```
void DFS( MGraph Graph, Vertex V, void (*Visit)(Vertex) )
/*第一个参数是传入二维数组的头地址（即图）
第二个参数是结点编号
第三个参数是传入Visit()这个函数的地址（可以自行百度函数是怎么作为参数传递的）*/
{
    /*从第V个顶点出发递归地深度优先遍历图G*/
    int i;
    Visited[V] = true;//标记为true，说明已经遍历过了
    Visit(V); //打印出V这个结点
    for(i = 0; i < Graph->Nv; i++) //遍历V的每个邻接点
    {
        if(Graph->G[V][i] == 1 && !Visited[i])
        /*Graph->G[V][i] == 1说明有结点，!Visited[i]为真，说明未遍历过*/
        {
           DFS(Graph, i, Visit); //递归调用DFS
        }
    }
}
```

这里给出裁判实现的代码:（可以把这段代码放到本地去运行）

```
#include<stdlib.h> //需要增加此头文件

MGraph CreateGraph() //创建图并且将Visited初始化为false
{
    int Nv, i, VertexNum;
    int v1, v2;
    Vertex V, W ;
    MGraph Graph;
    printf("请输入顶点个数: \n");
    scanf("%d", &VertexNum);
    Graph = (MGraph)malloc(sizeof(struct GNode));
    Graph->Nv = VertexNum;
    Graph->Ne = 0;
    for(V = 0; V < Graph->Nv; V ++) {
        for(W = 0; W < Graph->Nv; W ++) {
            Graph->G[V][W ] = INFINITY;
        }
    }
    printf("请输入边数: \n");
    scanf("%d", &Graph->Ne);
    if(Graph->Ne) {
```

```
        for(i = 0; i < Graph->Ne; i ++) {
            scanf("%d %d", &v1, &v2);
            Graph->G[v1][v2] = 1;
            Graph->G[v2][v1] = 1;
        }
    }

    return Graph;
}
```

2、BFS

1）选一个没有访问过的顶点入队列并标记其为访问过。如果找不到，遍历结束

2) 若队列不为空，取出队头顶点 x go to 3) 。若队列为空，go to 1)

3) 找出 x 的所有未访问过的邻点，将它们标记为访问过，并入队列

4)go to 2)

因为图可能不连通（有向图），或不是强连通（无向图），因此队列为空时，可能还有顶点未曾访问过

邻接表的 BFS

```
#include <stdio.h>

typedef enum {false, true} bool;
#define MaxVertexNum 10   /* 最大顶点数设为10 */
typedef int Vertex;       /* 用顶点下标表示顶点,为整型 */

/* 邻接点的定义 */
typedef struct AdjVNode *PtrToAdjVNode;
struct AdjVNode{
    Vertex AdjV;        /* 邻接点下标 */
    PtrToAdjVNode Next; /* 指向下一个邻接点的指针 */
};

/* 顶点表头结点的定义 */
typedef struct Vnode{
    PtrToAdjVNode FirstEdge; /* 边表头指针 */
} AdjList[MaxVertexNum];     /* AdjList是邻接表类型 */

/* 图结点的定义 */
typedef struct GNode *PtrToGNode;
struct GNode{
    int Nv;     /* 顶点数 */
    int Ne;     /* 边数   */
    AdjList G;  /* 邻接表 */
};
typedef PtrToGNode LGraph; /* 以邻接表方式存储的图类型 */
```

```
bool Visited[MaxVertexNum]; /* 顶点的访问标记 */

LGraph CreateGraph(); /* 创建图并且将Visited初始化为false; 裁判实现, 细节不表
    */

void Visit( Vertex V )
{
    printf(" %d", V);
}

void BFS ( LGraph Graph, Vertex S, void (*Visit)(Vertex) );

int main()
{
    LGraph G;
    Vertex S;

    G = CreateGraph();
    scanf("%d", &S);
    printf("BFS from %d:", S);
    BFS(G, S, Visit);

    return 0;
}

/* BFS的代码将被嵌在这里 */
```

BFS 代码:

```
void BFS ( LGraph Graph, Vertex S, void (*Visit)(Vertex) )
{
    int queue[11]; //定义一个队列
    int l = 0, r = 0;
    queue[r++] = S;
    Visit(S);
    Visited[S] = true;
    PtrToAdjVNode tmp;
    while(l != r) //队列不为空
    {
        tmp = Graph->G[queue[l++]].FirstEdge;
        while(tmp)
        {
            Vertex pos = tmp->AdjV;
            if(!Visited[pos])
            {
                Visit(pos);
                Visited[pos] = true;
                queue[r++] = pos;
```

```
            }
            tmp = tmp->Next;
        }
    }
}
```

这里给出裁判实现的代码:（可以把这段代码放到本地去运行）

```
#include<stdlib.h> //需要增加此头文件

LGraph CreateGraph()
{
    int i, k, VertexNum;
    Vertex V, E1, E2;
    LGraph Graph;
    printf("输入顶点的个数: ");
    scanf("%d", &VertexNum);
    Graph = (LGraph)malloc (sizeof(struct GNode));
    Graph->Nv = VertexNum;
    Graph->Ne = 0;
    for(V = 0; V < Graph->Nv; V ++) {
        Graph->G[V].FirstEdge = NULL;
    }
    printf("输入边的个数: ");
    scanf("%d", &(Graph->Ne));
    if(Graph->Ne) {
        for(i = 0; i < Graph->Ne; i ++) {
            scanf("%d %d", &E1, &E2);
            //插入边<E1, E2>
            PtrToAdjVNode NewNode;
            NewNode = (PtrToAdjVNode)malloc (sizeof(struct AdjVNode));
            NewNode->AdjV = E2;
            NewNode->Next = Graph->G[E1].FirstEdge;
            Graph->G[E1].FirstEdge = NewNode;
            //无向图, 所以还是插入边<E2, E1>
            NewNode = (PtrToAdjVNode)malloc (sizeof(struct AdjVNode));
            NewNode->AdjV = E1;
            NewNode->Next = Graph->G[E2].FirstEdge;
            Graph->G[E2].FirstEdge = NewNode;
        }
    }

    return Graph;
}
```

## 1.6 例题

1、迷宫问题

```
#include <stdio.h>
int n,ha,la,hb,lb;
char a[101][101];
int f=0;
int dx[4]={-1,1, 0,0};//上下左右
int dy[4]={ 0,0,-1,1};

void dfs(int x,int y)
{
    int i,xx,yy;
    if(f==1)return;
    if(x==hb&&y==lb){ f=1;return ; }
    for(i=0;i<4;i++)
    {
        xx=x+dx[i]; yy=y+dy[i];
        if(xx>=0&&xx<n&&yy>=0&&yy<n&&a[xx][yy]=='.'&&f==0)
        {
            a[xx][yy]='#';
            dfs(xx,yy);
            //a[xx][yy]='.';//这一行不能要，本题不需要回溯。若是加上这一行会导
                致超时。
        }
    }
}
int main()
{
    int i,j,k,t;
    scanf("%d",&t);
    for(k=0;k<t;k++)
    {
        scanf("%d",&n); getchar();
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
                scanf("%c",&a[i][j]);
            getchar();
        }
        scanf("%d%d%d%d",&ha,&la,&hb,&lb);

        /*printf("%d\n",n);
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
                printf("%c",a[i][j]);
```

```
            printf("\n");
        }
        printf("%d %d %d %d\n",ha,la,hb,lb);*/

        if(a[ha][la]=='#'||a[hb][lb]=='#') { printf("NO\n"); continue; }
        a[ha][la]='#';
        f=0;
        dfs(ha,la);
        if(f==1)printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

2、棋盘问题

```
1 #include <stdio.h>
 2 #include<string.h>
 3 #include<iostream>
 4 using namespace std;
 5 int qq[10][10],k,n,sum,pd[10],way;
 6 void look(int r)//处理第r行
 7 {
 8     if(way==k) {sum++;return;}
 9     if(r>n) return;
10     for(int i=1;i<=n;i++)//往第r行放棋子
11         if(qq[r][i]==1&&pd[i]==0)
12         {
13
14             pd[i]=1;way++;
15             look(r+1);
16             pd[i]=0;way--;
17         }
18     look(r+1);//第r行不放棋子
19 }
20 int main()
21 {
22     while(1)
23     {
24         way=0;sum=0;
25         scanf("%d%d",&n,&k);
26         if(n==-1&&k==-1) break;
27         memset(pd,0,sizeof(pd));
28         memset(qq,0,sizeof(qq));
29
30         for(int i=1;i<=n;i++)
31             for(int j=1;j<=n;j++)
32             {
```

```
33                  char t;
34                  cin>>t;
35                  if(t=='#') qq[i][j]=1;
36              }
37
38          look(1);
39          printf("%d\n",sum);
40      }
41      return 0;
42 }
```

### 1.7 DFS 剪枝

剪枝 1： 搭建过程中发现已建好的面积已经不小于目前求得的最优表面积，或者预见到搭完后面积一定会不小于目前最优表面积则停止搭建（最优性剪枝）

剪枝 2： 搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建可行性剪枝）

剪枝 3： 搭建过程中发现还没搭的那些层的体积，一定会超过还缺的体积，则停止搭建可行性剪枝）

剪枝 4： 搭建过程中发现还没搭的那些层的体积，最大也到不了还缺的体积，则停止搭建可行性剪枝）