图的介绍

> by ReActor

图作为离散数学中的一个重要内容，其是对现实世界中对象或者问题的一种抽象表示。

将实际问题转化为一个图的问题是解决问题的第一步。

1. 图的概念

图是点和边共同组成的集合，首先有一些点，然后是一部分边，这些边像道路一样，把点连接在一起。

当然边可能是单向的，也可能是无(双)向的。

$$\text{设有图 } G=(V,E)$$
$$V\text{为点集，}E\text{为边集}$$
$$|V|,|E| \text{ 分别表示点、边的数目}$$

> // Graph,Vertex,Edge

1.1 连通性

简单复习树中的定义：

- **路径：**顶点序列，使得其中每个顶点到该序列的下一个顶点有连边
- **简单路径：**一个没有重复顶点的路径称为**简单路径**
- **连通：** 两个顶点之间存在**路径**相连
- **环：**存在结点同时是**路径**的起点与终点

新的概念：

- **简单环：**除了第一个与最后一个顶点相同外，其他顶点均不相同；

默认讨论简单图，无重边、自环。

1.2 权重

一种抽象概念，代表着数据信息。

- **点权**：与点有关的数据信息；
- **边权：**与边有关的数据信息；
- **网络(网)：**每条边都带权的图成为网络，简称网；

数据结构书中的概念：

> 总之,问题所属的领域不同, 顶点和边或弧的实际意义也就不同。
> 在研究交通和通信问题 时,一个顶点可以表示一个城市,边或弧可以分别表示城市之间的道路或者通信线路,边或弧 上的权值可以表示道路的距离或者线路的造价; 在研究计划管理和工程进度时, 顶点可以表示 时刻, 弧表示一项工作以及该工作所需花费的时间; 在研究如何安排教学计划时, 顶点可以表 示课程,弧表示课程之间的选修关系; 在研究地图着色问题时 (不同地区着以不同颜色),一个 顶点表示一个地区,边表示两个地区的分界线 · · · · · · 因此, 灵活运用图的概念来描述问题是十分 重要的。

1.3 度

用两条有向边来模拟无向边。

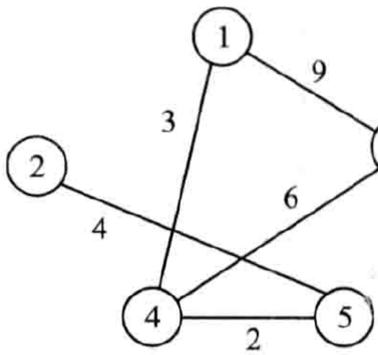
区分树与图中度概念的不同，因为树的通常以根区分上下方向；

- **结点的度(树)：** 结点拥有的子树(孩子)的数目；
- **度(TD)**：依附于该结点的边的数目；
- **入度(ID)**：指向，或者说以该结点为终点的边的数目；
- **出度(OD)**：指出，或者说以该结点为起点的边的数目；

设图$G=(V,E)$ 共有N 个点，M条边则：

$$TD(v_i)=ID(v_i)+OD(v_i)$$

$$2E=\sum_{i=1}^{n}TD(v_i)$$



1.4 图的类型

顶点与边的数目估计？开数组时候的估计；

无向图：$M \le \frac{N(N-1)}{2}$

有向图：$M \le N(N-1)$

- **完全图**：满边的无向图；
- **有向完全图**：满边的有向图；
- **有向无环图(DAG)**：Directed acyclic graph
- **稠密图**：边的数量接近完全图；
- **稀疏图**：边的数量较少；

1.5 子图

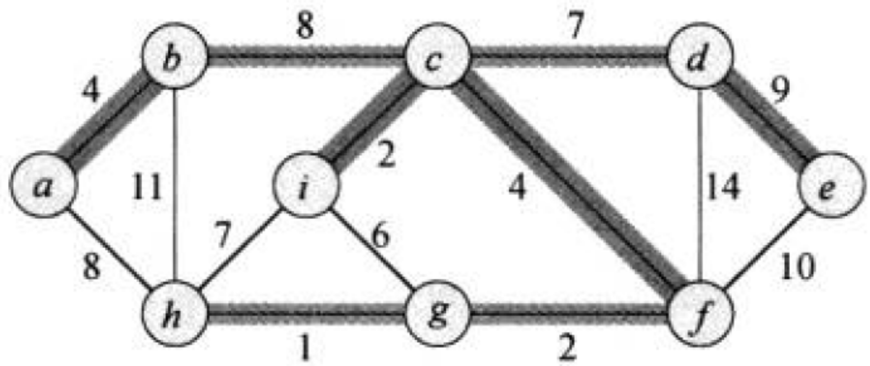
设图$G=(V,E)$ 与 $G'=(V',E')$，满足 $V'\subseteq V$，$E'\subseteq E$，则称 G' 为 G 的**一个子图**

- **连通图：** 无向图中，任意两个顶点可达；
- **连通分量：** 无向图的极大连通子图；
  - 显然连通图的连通分量仅有它本身；
  - 大是针对所包含结点的个数而言的；
- **强连通图：** 有向图中，任意两个顶点**相互**可达；
- **强连通分量：** 有向徒的极大强连通子图；

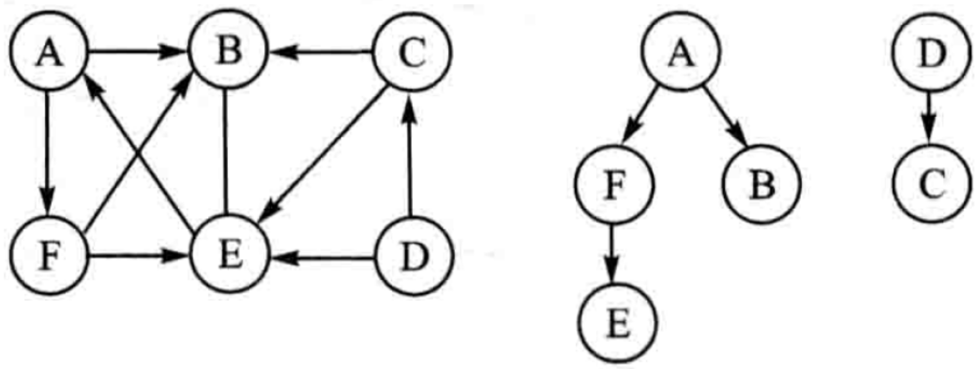
> 注意当提到连通图时，我们已经默认它是无向的，对强连通图也同理。

1.6 生成树

- **生成树**：在无向图上定义，连通图G的一个极小连通子图



- **生成森林**：在有向图上定义，由若干棵有向树构成。包含图中全部顶点，但只有足以构成若干**互不相交**的**有向树**的边。



2. 图的存储

2.1 邻接矩阵

多用于稠密图

人为地用序数下标表示顶点；

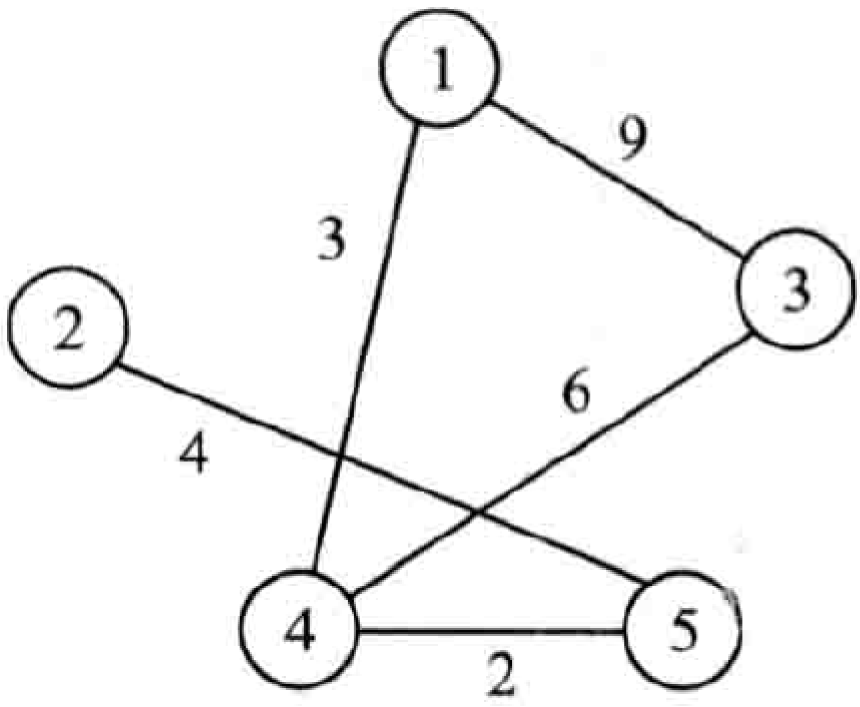
$$A[i][j]=\begin{cases}1 & v_i与v_j之间有边\\0 & v_i与v_j之间无边\end{cases}$$

$$A[i][j]=\begin{cases}w_{ij} & 顶点\ i\ 与顶点\ j\ 之间有边,且边上的权值为\ w_{ij}\\\infty & 顶点\ i\ 与顶点\ j\ 之间无边\end{cases}$$

邻接矩阵；

$$A1=\begin{bmatrix}0&1&1&1\\1&0&1&1\\1&1&0&1\\1&1&1&0\end{bmatrix},$$

$$A2=\begin{bmatrix}\infty&\infty&9&3&\infty\\\infty&\infty&\infty&\infty&4\\9&\infty&\infty&6&\infty\\3&\infty&6&\infty&2\\\infty&4&\infty&2&\infty\end{bmatrix}$$



CODE:

```
int A[maxn][maxn], n;
void ADJMatrix(int m)
{
    memset(A, 0x3f, sizeof(A));

    int w, u, v;
    for (int i = 1; i <= m;++i)
    {
        scanf("%d%d%d", &u, &v, &w);
        A[u][v] = w;
        // A[v][u]=w; 双向边
    }
}
```

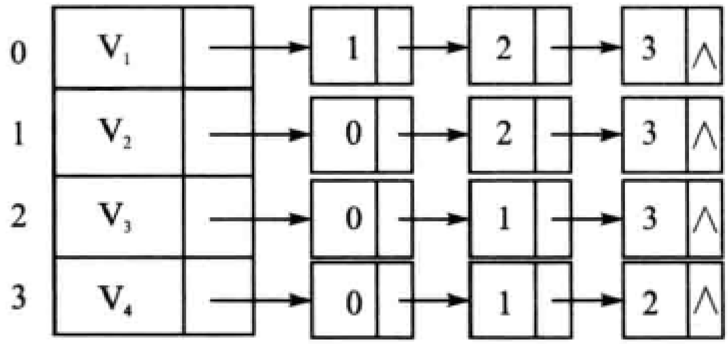
复杂度 $O(N^2+M)$

2.2 邻接表

常见于稀疏图，例如$M = O(2N)$ 的情况下。

每个顶点都对应一个链表，存储所有以该节点为起点的边。当需要访问一个结点的出边的时候，就枚举它对应的边表中所有的边。

图例：



(a) G_1的邻接表

实现上：

定义顶点，边两种结构体，分别作为表头，链表结点。

```
//邻接表的结构体定义
typedef struct edge
{
    int adjvex;
    int weight;
    struct edge *next;
} ELink;

typedef struct ver
{
    int data; //可选的结点数据信息
    ELink *link;
} VLink;
//邻接表的全局数据
VLink G[maxn];
int n, m;
//邻接表的成员函数
void addEdge(int u, int v, int w)
{//加边
    ELink *p = (ELink *)malloc(sizeof(ELink));
    p->adjvex = v - 1; //本质是从编号映射到数组下标
    p->weight = w;
    p->next = G[u - 1].link;
    G[u - 1].link = p;
    //小的优化
}

void initG()
{//初始化表头，在复用性上重要
    for (int i = 0; i < n; ++i)
    {
        G[i].data = i + 1;
```

```
        G[i].link = NULL;
    }
}

void ADJLIST(int n, int m)
{ // Adjacency List 邻接表，读入数据生成邻接表
    initG();
    int vi, vj, w;
    for (int i = 0; i < m;++i)
    {
        scanf("%d%d%d", &vi, &vj, &w);
        addEdge(vi, vj, w);
    }
}
```

3. 图的遍历：

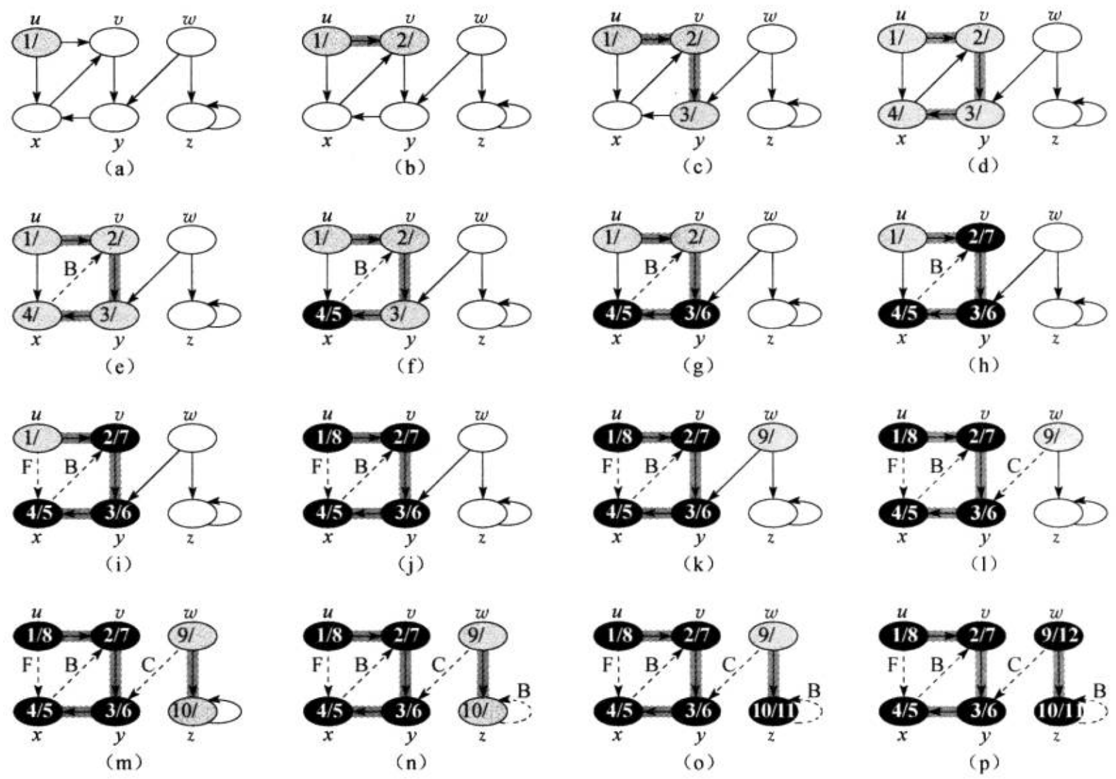
考虑一种师徒关系，如果A是B的师父，B是C的师父，那么我们认为A也是C的师父，并且是二代师父。

如果给定了n个人，m对师徒关系，我们怎样确定每个人i到底有多少徒弟？

3.1 深度优先搜索(DFS)

询问A是否有3个以上的200代徒弟？

尽可能早地访问更深处的结点：如果一个节点存在子节点，那么优先访问它。



```
int vis[maxn];//需要初始化

void DFS(int u)
{
    int v, w;
    vis[u] = 1;//标记访问
```

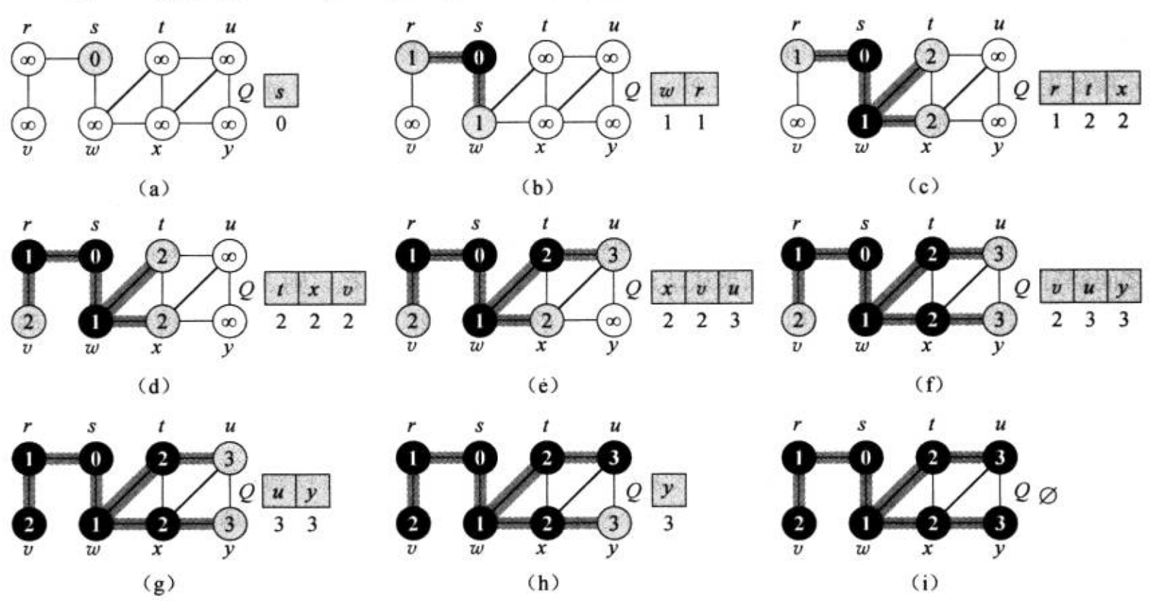
```
    for (ELink *p = G[u - 1].link; p != NULL; p = p->next)
    { //初始化指针为头链表，访问所有表中元素
        v = p->adjvex;
        w = p->weight;
        //do stuff 根据需要选择前后序遍历
        if(vis[v])
            continue;
        DFS(v);
    }
}

void DFS_G()
{
    memset(vis, 0, sizeof(vis));
    for (int i = 1; i <= n; ++i)
    {
        if (vis[i])
            continue;
        DFS(i);
    }
}
```

3.2 广度优先搜索(BFS)

询问A 三代以内的徒弟有多少个?

总是先访问层数更低的结点:

维护一个计划访问的结点的队列，如果一个节点存在子节点，不直接访问它，而是把他加入到队列的末尾。




```
int vis[maxn];

void BFS(int cur)
{
    //初始化队列，加入队头
    int q[maxn], front, rear, u, v, w;
    front = rear = 0;
    q[rear++] = cur;
    //进入队列访问
```

```
    while (front < rear)
    {
        u = q[front++];
        vis[u] = 1;
        for (ELink *p = G[u - 1].link; p != NULL; p = p->next)
        {
            v = p->adjvex;
            w = p->weight;
            // do stuff;
            if (vis[v])
                continue;
            q[rear++] = v;//加入队末尾
        }
    }
}

void BFS_G()
{
    memset(vis, 0, sizeof(vis));
    for (int i = 1; i <= n; ++i)
    {
        if (vis[i])
            continue;
        BFS(i);
    }
}
```

3.3 两种搜索的性质简介

复杂度：$O(V+E)$

通过标记，每个顶点最多被访问一次，每条边在所依附的顶点处被访问。

3.3.1 深度优先搜索

- DFS树：对一个节点DFS会得到一棵树，整体上会产生一个森林；
- DFS序：在DFS中访问到结点的时间顺序，用正整数表征
    - 通常在访问结点时标记$(u.d)$，也可以选择标记两次，第二次在退出结点时标记$(u.f)$
- 括号化结构：被访问的子树的时间戳一定被其父节点的时间戳包含，并且不同子树之间的结点必然不会交叉；
- 白色路径定理：分析DFS时结点的状态来得到更丰富的性质；

3.3.2 广度优先搜索

- 层数低的结点一定比层数高的结点优先被访问到；
- 队列中的结点层数的上下界恰好差1；
- BFS求最短路径；
- 树的直径；

4. 总结