图形结构属于多对多关系

**图的基本概念**

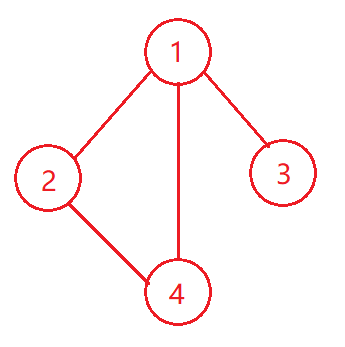
**图的定义**

我们使用 G=(V, E) 表示图G

V是顶点集合，如 V = { 1, 2, 5, 8 }

E是边集合，如 无向图的E = { (1, 2), (1, 3) }，有向图的E = { <1, 2>, <2, 1>, <2, 3> }

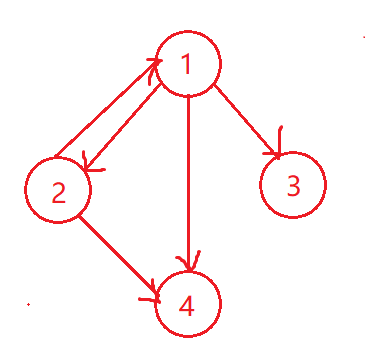
如下是无向图



其V={1, 2, 3, 4}

其E={(1, 2), (1, 3), (1, 4), (2, 4)}

如下是有向图

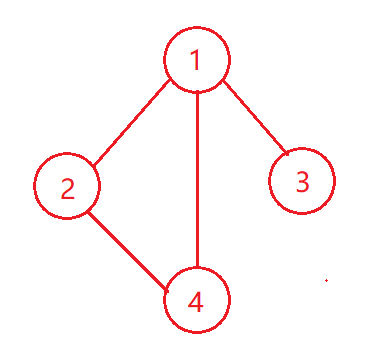


其V={1, 2, 3, 4}

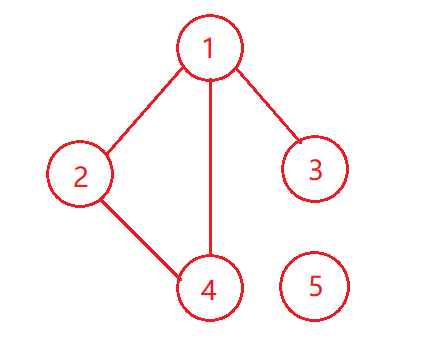
其E={<1, 2>, <1, 3>, <1, 4>, <2, 1>, <2, 4>}

**连通图**

无向图中，任意2个顶点都有路径相连起来，则改图称为连通图，如下为连通图



而如下不是连通图，因为1 2 3 4 和顶点5并不相连



**生成树**

具有n个顶点的连通图，有n-1条边将这些顶点连起来组成的图称为生成树，如

图2，3均是图1的生成树

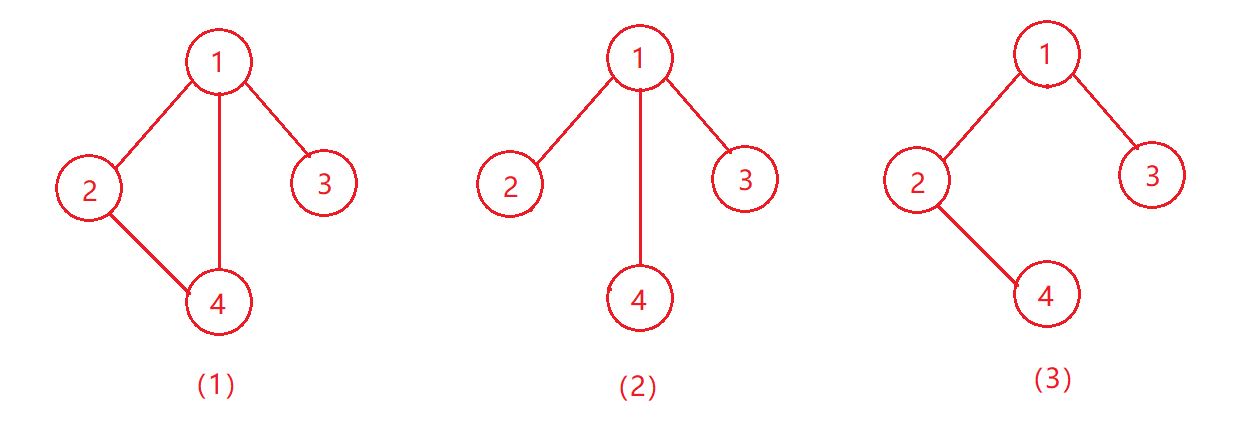


图2，3均为树状结构

**有权图**

为图的边加上权，即为有权图，权可以代表2点之间的长度，时间等

**图的存储**

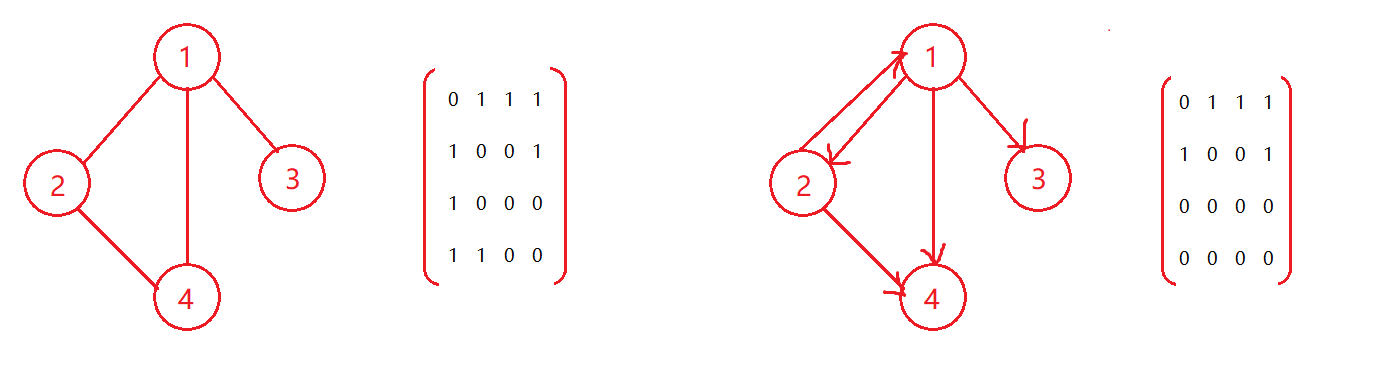
**数组表示法**

我们可以使用n\*n数组矩阵去表示具有n个顶点的图

1. 无权图

我们使用位矩阵，0（false）表示未相连，1（true）表示有相连

如下是无权图和其所对应的矩阵

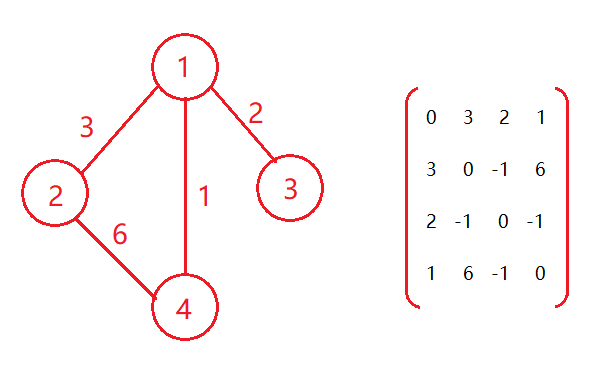


矩阵元素aij = 1 表示顶点i有连接到顶点j，你应该可以发现，无向图的矩阵是对称矩阵，因为i连接j就代表j连接i，所以我们只需保存右上角的数据即可

1. 有权图

我们可以使用int类型矩阵表示，-1表示未连接，0表示当前节点，>0表示有连接，其值表示权

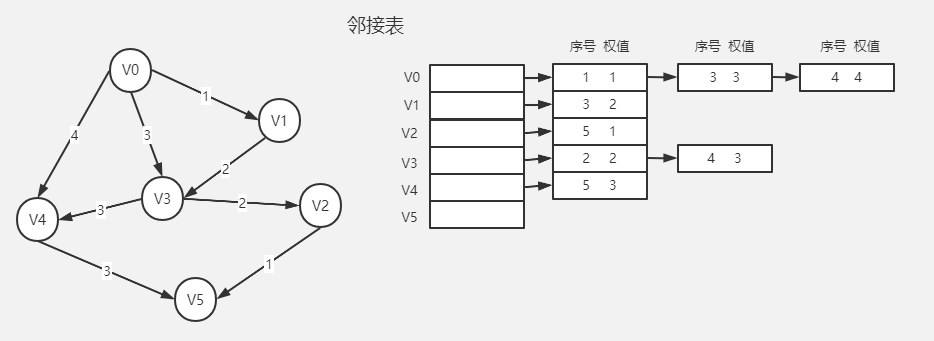
如下是有权图和其对应的矩阵



**邻接表表示法**

邻接表表示法使用链表去存储图

如下是图和邻接表的存储

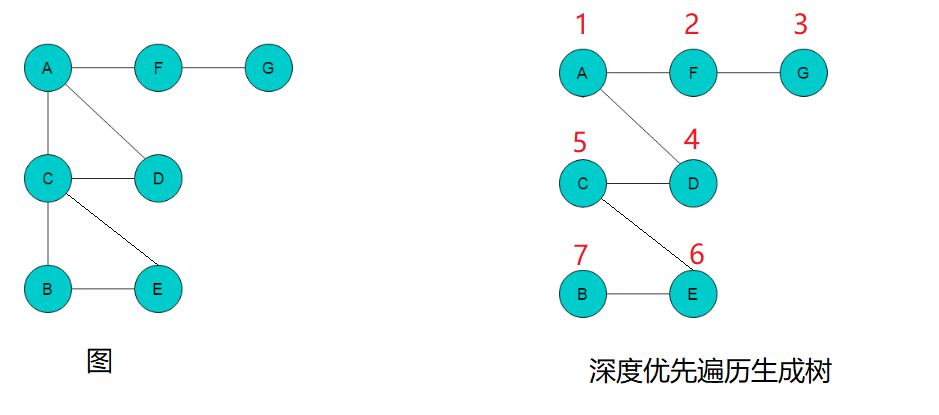


**图的遍历**

**深度优先遍历**

遍历很简单，就步废话了

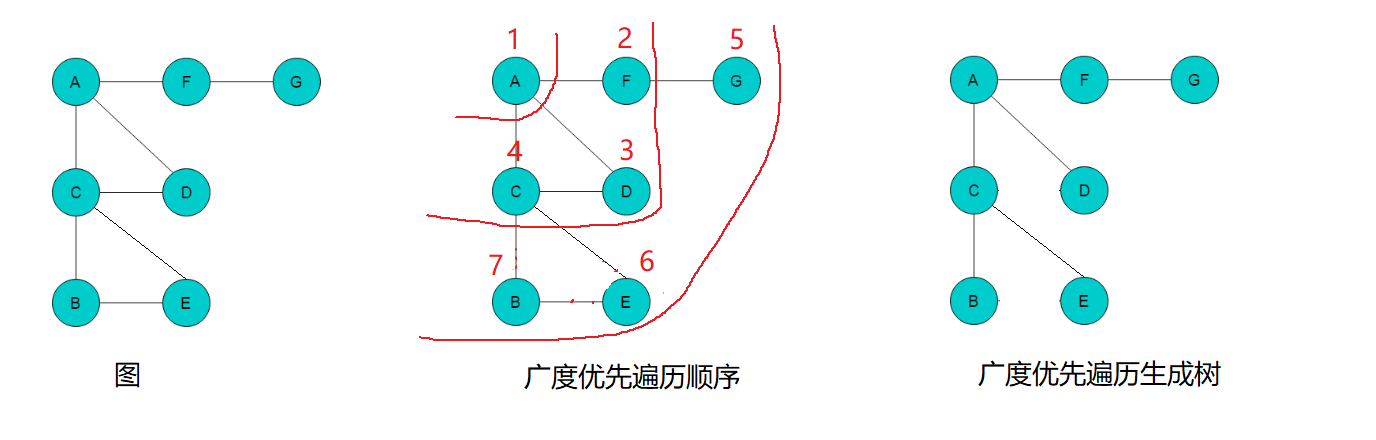
如下，其遍历顺序为A, F, G, D, C, E, B



根据其遍历顺序生成的树为右边的图

**广度优先遍历**

如下是广度优先遍历的过程



1. 初始时，将A放入队列
2. 遍历A的连接顶点，得到F D C，将F D C放入队列，A 移遍历，将A移出队列
3. 取队列的下一个，即F，遍历F得到G，将G放入队列，F已遍历，将F移除队列
4. 依此类推，我们就可以得到整个图的遍历顺序

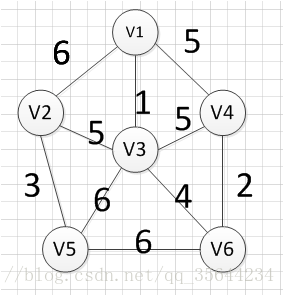
**无向图最小生成树**

参考：<https://blog.csdn.net/qq_35644234/article/details/59106779>

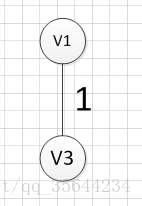
无向图中找出一颗生成树，该生成树的权加起来最小，即为最小生成树

**普里姆（Prim）算法**

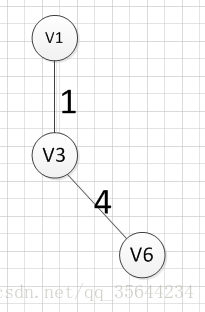
如下，从图中生成最小生成树



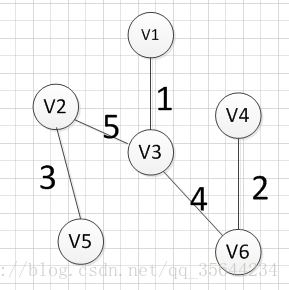
假设我们从顶点v1开始，所以我们可以发现（v1,v3）边的权重最小



接着在v1,v3所连接的边中找出能连接到新顶点且权最小的边，加入到树中



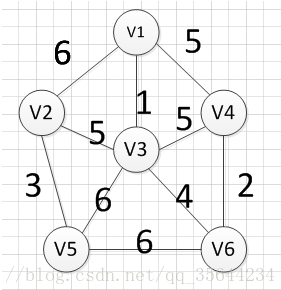
接着在v1，v3，v6所连接的边中找出能连接到新顶点且权最小的边，加入到树中，依此类推，得到的树为



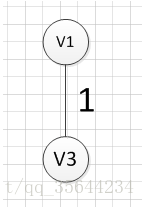
该树即为最小生成树

**克鲁斯卡（Kruskal）算法**

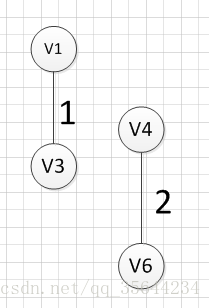
如下，从图中生成最小生成树



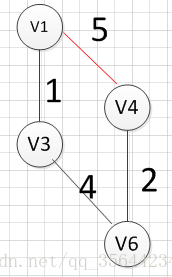
在途中找出权重最小的边



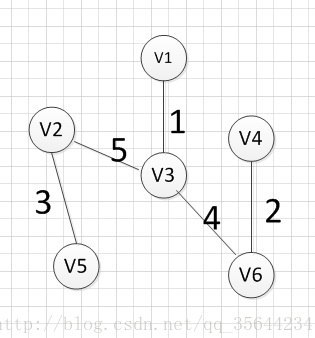
接在再寻找权重最小的边



接在再寻找权重最小的边，我们需要确保这些边不会形成环，即不能出现如下环路



如此循环，我们便可构建出最小生成树，如下

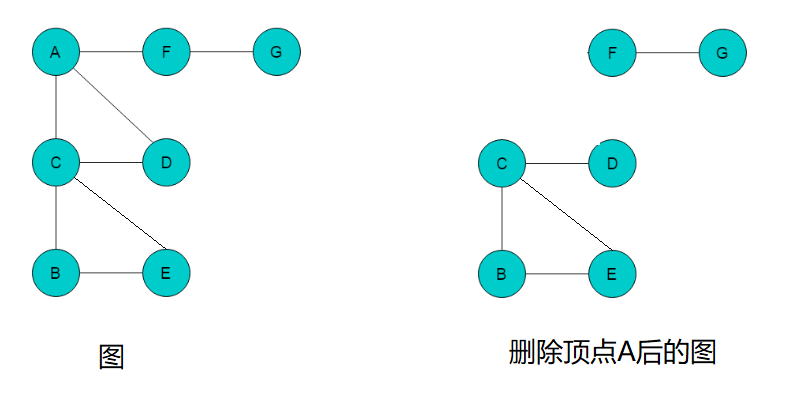


**无向图关节点和双连通图**

**关节点**

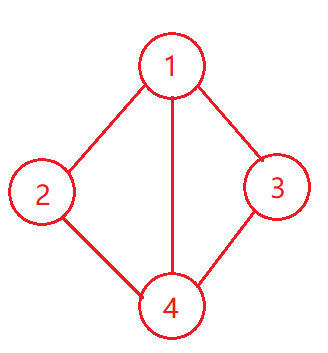
如果删除连通图的某个顶点v，使得连通图变成非连通图，则该顶点v为关节点

如下，A为关键点



**双连通图**

不存在关节点的图称为双连通图，如下图

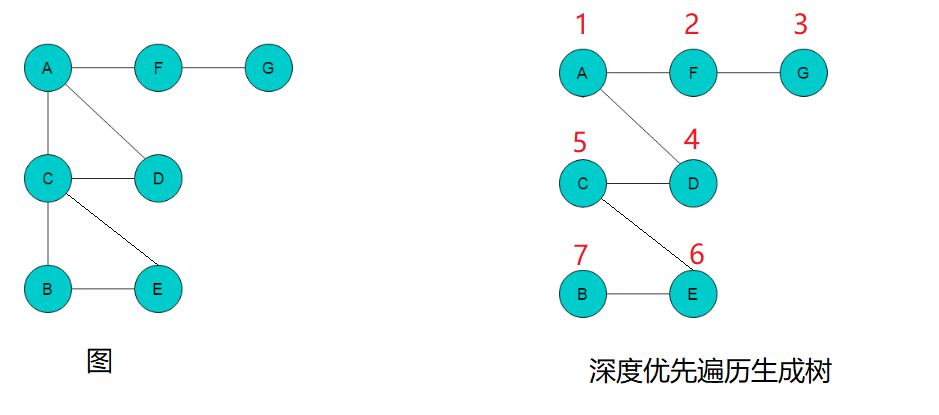


**寻找关节点**

利用深度优先遍历生成树

1. 如果顶点v是树的根，且顶点v有2个以上的子树，则顶点v是关节点
2. 如果顶点v不是树的根，顶点v的某个子树的所有顶点均不与顶点v的祖先有连接，则顶点v是关节点

如下图和其生成树，我们寻找其关节点



1. A是根，且A有2棵子树，所以A是关节点
2. F 不是根，其子树并没有任何一个顶点连接到祖先，所以F是关节点
3. D的子树有顶点C，顶点C有连接到其祖先，所以D不是关节点
4. C的子树没有任何一个顶点连接到祖先，所以C是关节点
5. ... 剩下的自己判断

**有向图AOV网和拓扑排序**

参考：<https://blog.csdn.net/qq_41713256/article/details/80805338>

拓扑排序用于再执行上有先后顺序的一系列活动，我们将这一系列的活动构建成图，该图称为AOV网，通过拓扑排序，我们可以给出则一系列活动的执行先后顺序

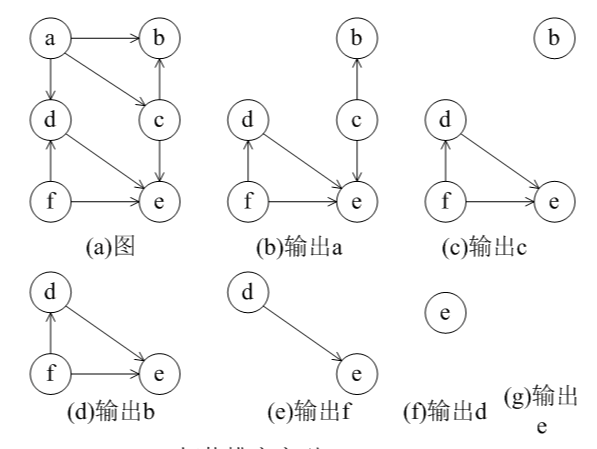
**拓扑排序**

如下示例

我们有活动a b c d e f，活动a可以直接执行，活动b要执行必须先执行活动a和c，活动c要执行必须先执行活动a，活动d要执行必须向执行活动a和f，活动e要执行必须向执行c和d和f，活动f可以直接执行

根据如上的描述，我们可以构建图（a），通过如下步骤找出活动的执行顺序

1. 找到没有入度的顶点（即没有被指向的顶点），输出该顶点，并移除该顶点
2. 重复步骤1直到所用顶点都输出
3. 输出顶点的顺序就是活动执行的讯息



如上，a没有入度，所以输出a，然后c b f d e

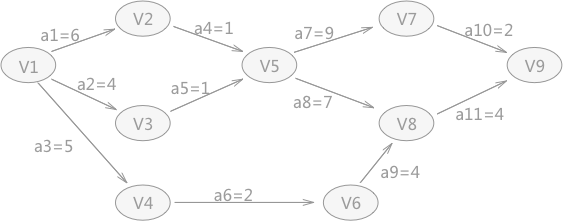
**有向图AOE网和关键路径**

参考：<http://c.biancheng.net/view/3417.html>

**AOE网**

AOE网使用边代表活动，边的权值代表活动所需的时长，使用顶点代表事件，活动是并发进行的，但活动之间有先后顺序

如下AOE网，活动a1需要6个工时，完成活动a1后进入v2可开始a4活动，完成a4和a5后进入v5，可开始a7和a8



**关键路径**

AOE网从起点到终点，最长的路径称为关键路径，如上AOE网，其中的一条关键路径为a1、a4、a7、a10，所需的工时为18，即项目至少需要18个工时才能完成

**求解关键路径算法**

以上面的例子为示例

1. 计算各个事件的最早发生事件ve(j)

我们使用一个数组ve代表各个事件的最早发生时间

1. 首先我们初始时在v1，所以 ve[0] 等于0

var ve = [0, null, null, null, null, null, null, null, null]

1. 执行完a1后可进入v2，执行完a2后可进入v3，执行完a3后可进入v4，所以v2、v3、v4早发生时间为v1的最早发生时间加上各个活动所需时间，为 v1+a1、v1+a2、v1+a3

var ve = [0, 6, 3, 5, null, null, null, null, null]

1. v5需要完成a4和a5后才可进入，所以v5的最早发生时间是完成a4和a5的最长时间，a4的完成时间是v2+a4=7，a5的完成时间是v3+a5=5，所以v5的值为7

var ve = [0, 6, 3, 5, 7, null, null, null, null]

1. 安装上面的方法，我们得到各个事件的最早发生时间

var ve = [0, 6, 3, 5, 7, 7, 16, 14, 18]

计算出ve(j)后，我们可以发现完成整个项目至少需要18个工时，但我们还并不知道关键路径包含哪些活动

1. 计算各个事件最晚什么时候发生vl(j)

我们使用数组vl保存各个事件的最晚发生事件

上面的计算中我们知道项目至少需要18个工时，所以我们假设v9的最晚发生事件为18，依此计算各个事件的最晚发生时间

1. 初始时，v9为18

var vl = [null, null, null, null, null, null, null, null, 18]

1. v7和v8的最晚发生时间为，v9 - a10和v9 - a11

var vl = [null, null, null, null, null, null, 16, 14, 18]

1. v5有2个活动，为确保项目按时完成，我们应该取最早（也就是最小）的时间，v7-a7=7，v8-a8=7，这量最晚发生时间都是11，取最小值也是11，所以v5为7

var vl = [null, null, null, null, 7, null, 16, 14, 18]

1. 按照如上方法，我们计算出vl

var vl = [0, 6, 6, 8, 7, 10, 16, 14, 18]

1. 计算各个活动的最早开始时间e(i)

各个活动的最早开始时间就是活动起点事件的最早发生时间，如 a1 的最早开始时间为 v1 即0，a7的最早开始时间为v5即7，所以最早发生时间如下

var e = [0, 0, 0, 6, 4, 5, 7, 7, 7, 16, 14]

1. 计算各个活动的最晚开始时间l(i)

各个活动的最晚开始时间等于活动的终点事件的最晚发生时间减去活动的时长，如v3的最晚发生时间是6，所以a2的最晚开始时间是v3 - a2 = 2

var l = [0, 2, 3, 6, 6, 8, 7, 7, 10, 16, 14]

1. 求出关键路径

至此，我们已求出4个数组

// 事件的最早发生事件

var ve = [0, 6, 3, 5, 7, 7, 16, 14, 18]

// 事件最晚什么时候发生

var vl = [0, 6, 6, 8, 7, 10, 16, 14, 18]

// 活动的最早开始时间

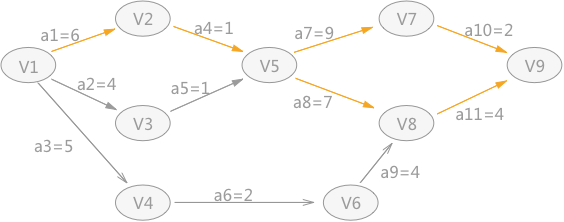
var e = [0, 0, 0, 6, 4, 5, 7, 7, 7, 16, 14]

// 活动的最晚开始时间

var l = [0, 2, 3, 6, 6, 8, 7, 7, 10, 16, 14]

当e[i] == l[i] 时，活动i便是一个关键活动，由关键活动组成的路径便是关键路径

如上示例中我们求出2条关键路径（黄色箭头标出）



注：或许我们可以一眼就看哪些时关键路径，哪些不是，但我们想要用编程去解决，如上是一个常用的思路，当然我们也可以使用其他思路去求解

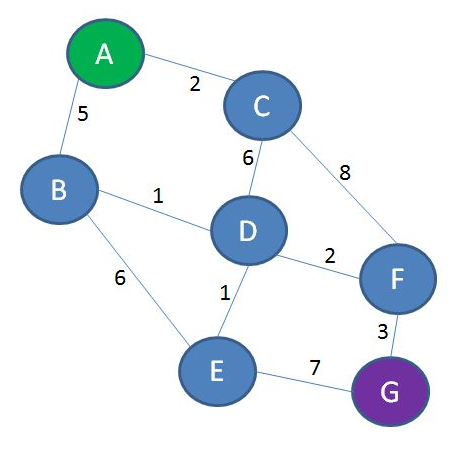
**最短路径**

**迪杰斯特拉（Dijkstra）算法**

参考：<https://blog.csdn.net/bjweimengshu/article/details/89090053>

Dijkstra算法用于求一个顶点到其他顶点的最短距离

我们通过如下示例了解Dijkstra算法，求顶点A到其他顶点的最短路径



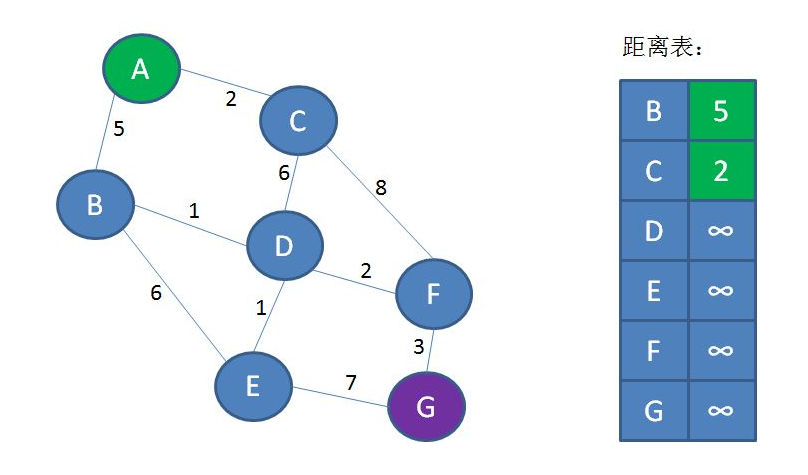
我们需要3个数组

Dist（距离表）：保存其他顶点到顶点A的最短距离

Find：保存已计算出最短距离的顶点

Path：保存最短距离的路径

1. 初始时，A到B C顶点的距离为2和5，其他顶点位置，所以距离表天上2和5



这时候我们还不能确定A到其他顶点的距离是否为最短，所以find如下

var find = {

    B: 0,

    C: 0,

    D: 0,

    E: 0,

    F: 0,

    G: 0

}

我们确定了BC的距离，所以最短路径如下

var path = {

    B: A,

    C: A,

    D: 0,

    E: 0,

    F: 0,

    G: 0

}

1. 从距离表中找出最短距离的顶点C，由于C在距离表中是最短，现在我们可以确定到C的最短距离就是2，所以更新find，将C更新为1

var find = {

    B: 0,

    C: 1,

    D: 0,

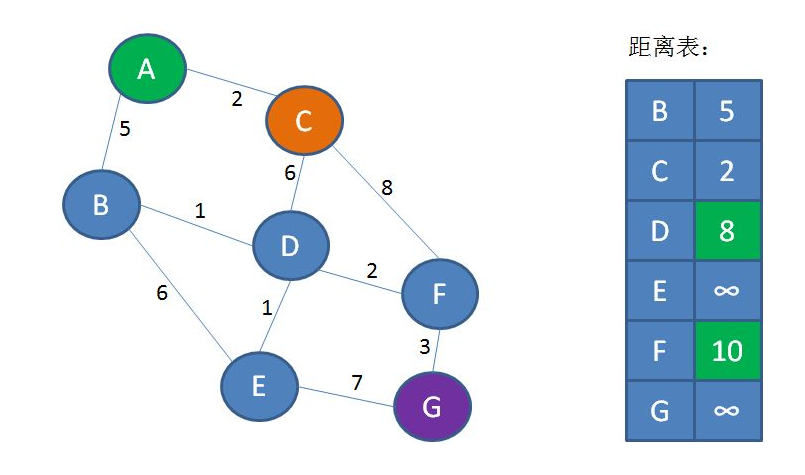
    E: 0,

    F: 0,

    G: 0

}

我们遍历C的边，得到D F的距离为8和10，将得出的距离和现有的距离比较，将较小的距离更新到表中



刚才我们更新了D F，所以我们更新路径表

var path = {

    B: A,

    C: A,

    D: C,

    E: 0,

    F: C,

    G: 0

}

1. 我们接着在未确定最短距离的顶点中找出最小距离的顶点B，由于B是最短距离，所以现在我们确定B的最短距离就是5，更新find

var find = {

    B: 1,

    C: 1,

    D: 0,

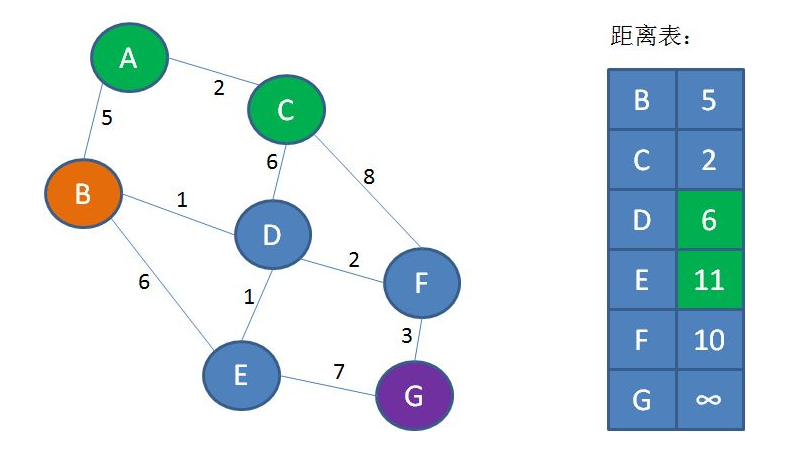
    E: 0,

    F: 0,

    G: 0

}

遍历B的所有边，计算出A到 D E 的距离为6和11，和距离表的值比较，将较小者更新到距离表中



刚刚我们更新了距离表的D E，这说明我们将DE指向B路径会更短，所以我们path

var path = {

    B: A,

    C: A,

    D: B,

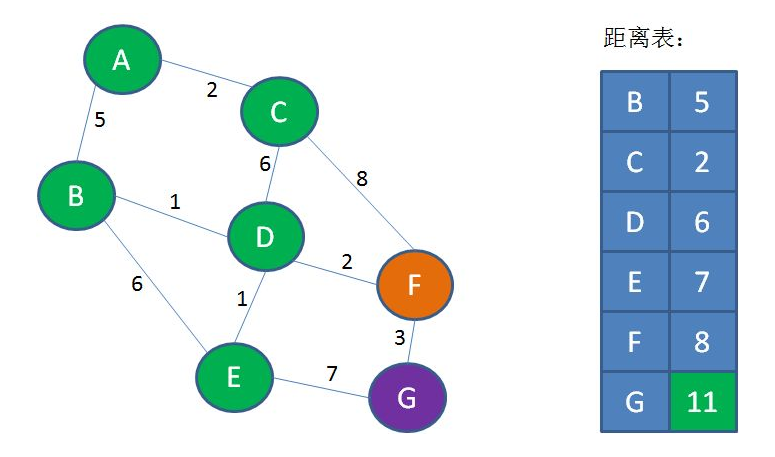
    E: B,

    F: C,

    G: 0

}

1. 循环上面的方法，我们最后的数组变为如下



var find = {

    B: 1,

    C: 1,

    D: 1,

    E: 1,

    F: 1,

    G: 1

}

var path = {

    B: A,

    C: A,

    D: B,

    E: D,

    F: D,

    G: F

}

我们可以知道A到G的最短为11，而A到G的最短路径为 G < F < D < B < A

**弗洛伊德（Floyd）算法**

如果我们要求解每一对顶点之间的最短距离，我们可以将执行n-1遍Dijkstra算法得出，但我们也可以使用Floyd算法求解，这里我就不做介绍了，给出一个地址自己看吧

<https://www.cnblogs.com/wangyuliang/p/9216365.html>