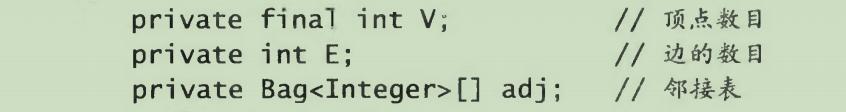
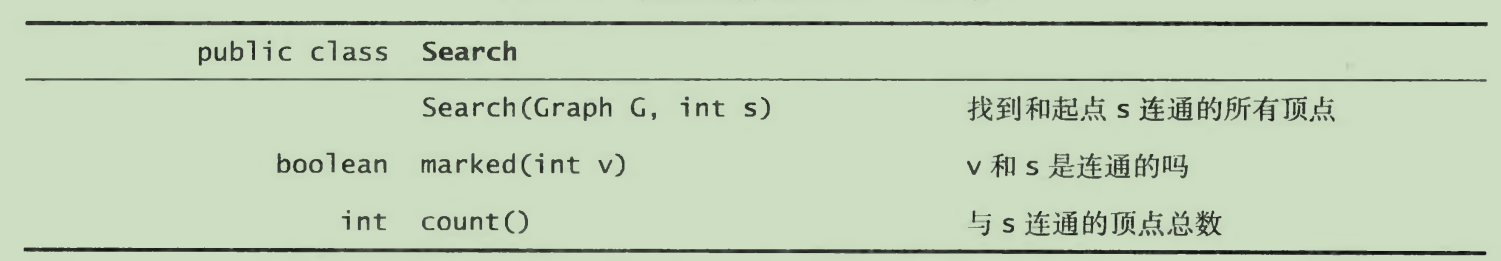
连通分量定义

在无向图中,如果从顶点vi到顶点vj有路径,则称vi和vj连通.如果图中任意两个顶点之间都连通,则称该图为连通图,否则,将其中的较大连通子图称为连通分量.[无向图](https://baike.baidu.com/item/%E6%97%A0%E5%90%91%E5%9B%BE)的极大连通子图称为的**连通分量**

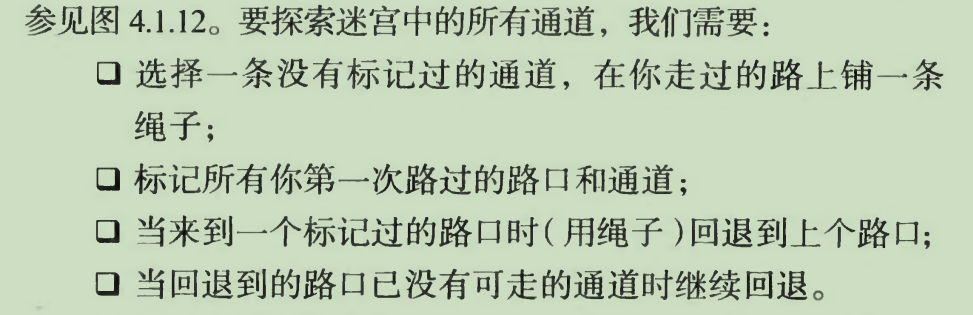


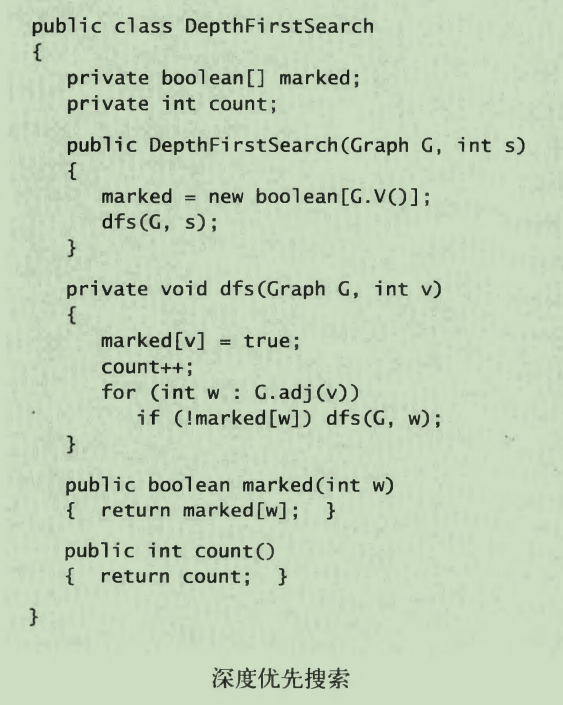


深度优先搜索

问题1 有多少点与起点相连接（count），有哪些点与起点相连接（ marked）

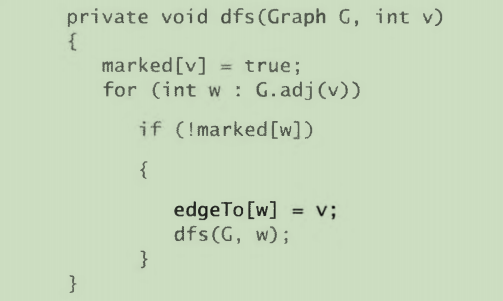
算法逻辑 我们注意寻找点的顺序，其实是一个堆的逻辑，先进后出，依靠先进后出数据结构，来实现先顺着起点往后找跟他相关联的点，如果下一个节点的所有临接点都被查找过，就返回到上一个节点，找上一个节点剩下的临接点，所以先进后出，这也是整个算法的逻辑思路。





问题2 起点到目的点的路径，表示出来

我们用edgeTo来保存深度搜索路径



、

**连通分量**

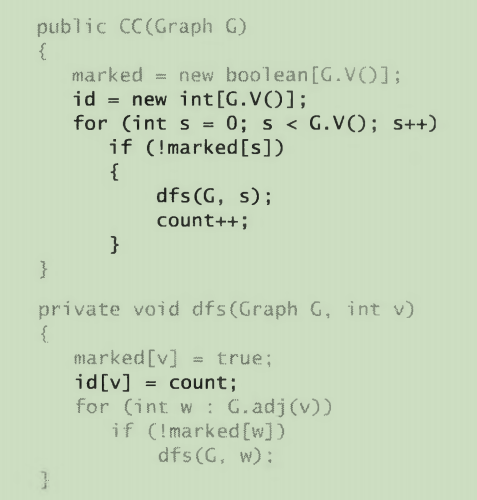
**问题**

找出图中存在的连通分量

**逻辑**

从图中第0个顶点开始（总循环），使用深度搜索，深度搜索会把该点的所有相连点标记为第1个连通分量，标记的点就会在（总循环）跳过，没有被标记的点，就开始重复进行深度搜索，标记为第二个连通分量

**代码展示**



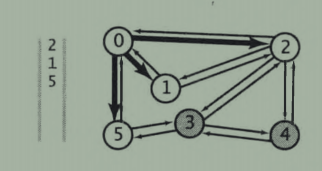
5 广度优先搜索

问题 ：单点最短路径。给定一幅图和一个起点S,回答“从 S 到给定目的顶点V 是否存在一条路径？如果有，找出其中最短的那条（所含边数少）。”等类似问题。

逻辑思路

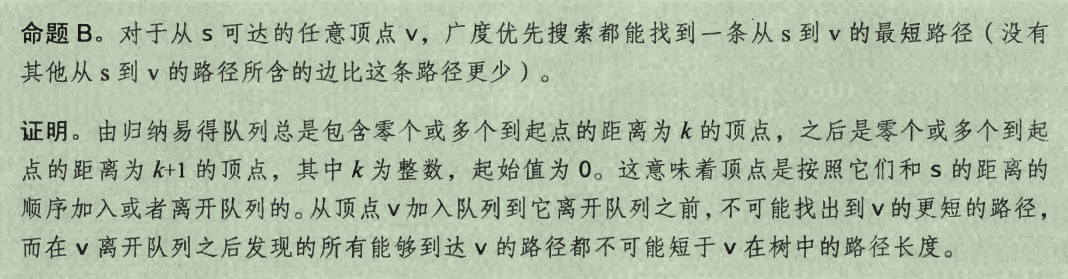
正如其名，广度意味着从起点开始把它相关联的点动放入到队列中，接着下一步就是把队列中的点的相关联的点存入到队列中，以此类推，与深度优先不同，他是先进先出，用的是队列的数据结构。



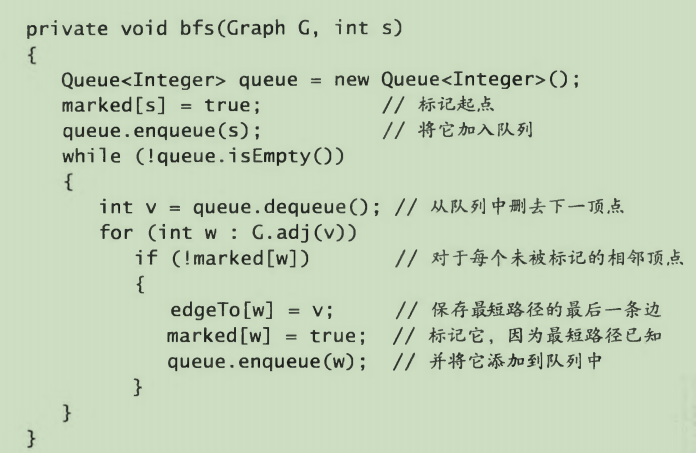




**上面的思路解释了整个算法流程，为什么这个可以找出最短的距离 思想如下**



**代码展示**



G.adj 表示边的关系 0（起点） 1 2 5

**二分图**

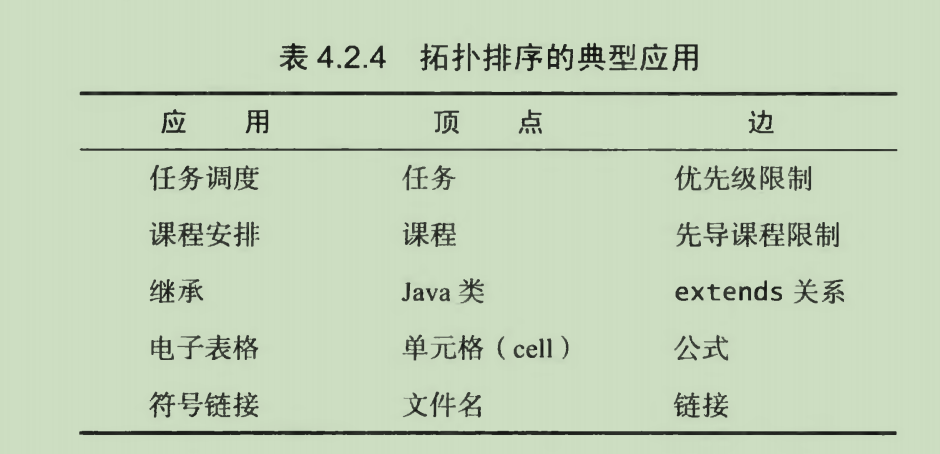
这个图我们一般运用于生活中，比如小李子演了很多电影，成龙也演了很多部电影，然后我们用图表示的时候是，一条边必须有演员人名和电影名，人名和电影名两个不同顶点就表示了二分图。

**有向图**

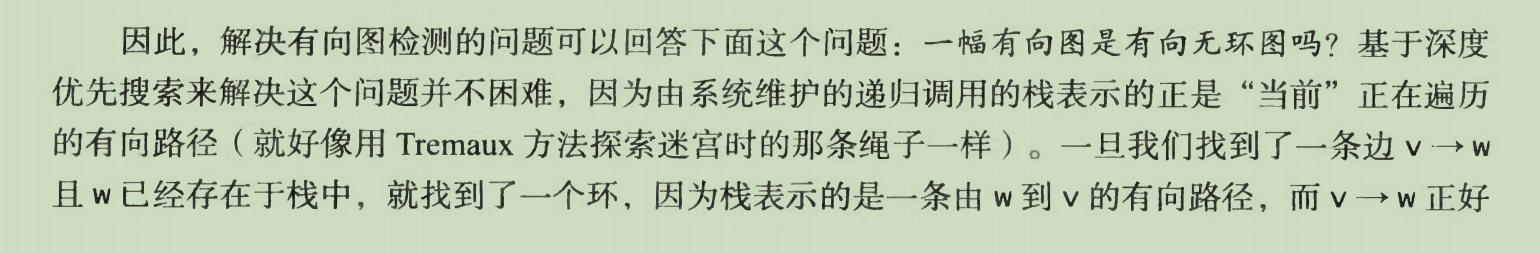
**拓扑排序**

给定一幅有向图，将所有的顶点排序，使得所有的有向边均从排在前面的元素指向排在后面的元素

下面的图片表示了我们处理这种有先后级的问题时候，比如课表安排，课程有一个先后顺序，如果课程之间出现相互依赖，比如一个环，就进入了死循环，无法找出一个先后顺序，所以这种图就不能出现环

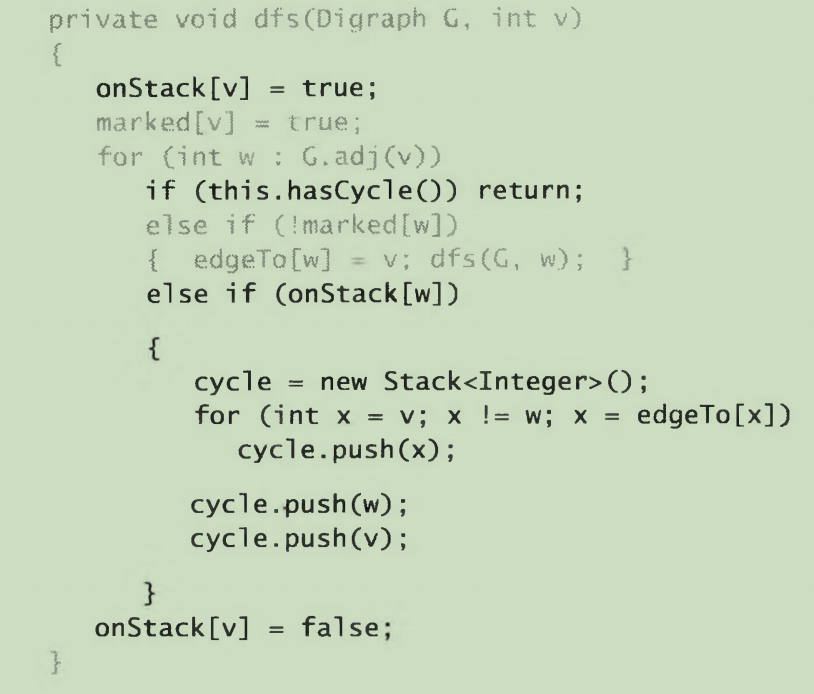


**逻辑**





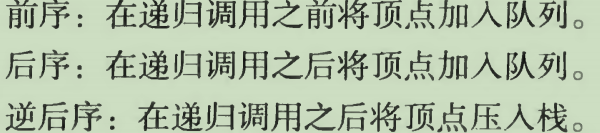
**代码展示**



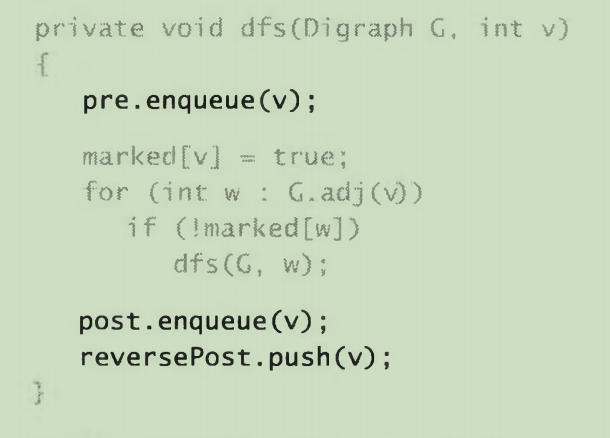
onstack表示顶点是否在栈上。

**解决无环的时候进行拓扑排序**

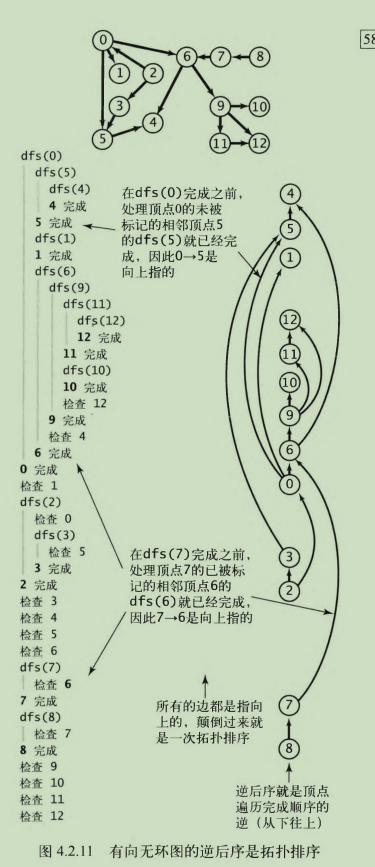
实际上我们已经见过一种拓扑排序的算法：只要添加一行代码，标准深度优先搜索程序就能完成这项任务！诞生了3种排列顺序



**代码演示**



下面图片展示了基于深度优先算法的后序排序，把后序排序的数字从队列中拿出来，然后用他们边的关系一 一连线就能够画出下面流程图



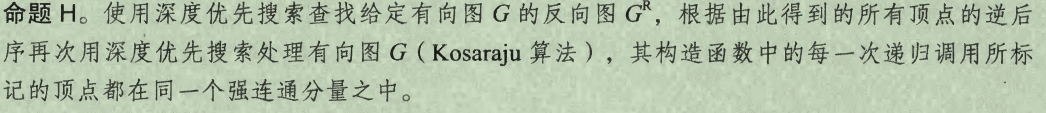
**有向图中的强连通性**

如果两个顶点V 和 W 是互相可达的，则称它们为强连通的。也就是说，既存在一条从V到 W 的有向路径，也存在一条从W 到 V 的有向路径。如果一幅有向图中的任意两个顶点都是强连通的，则称这幅有向图也是强连通的

Kosaraju 算法（属于**有向图中的强连通性**）

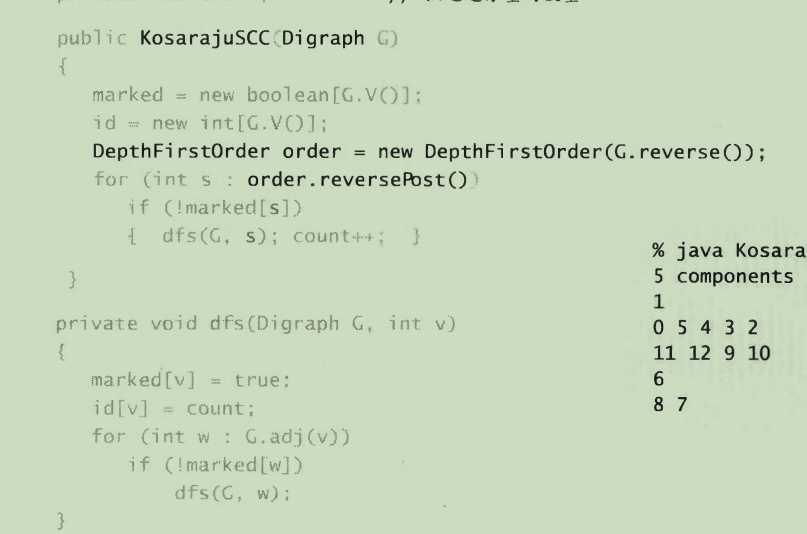
**思路**

与前面无向图的逻辑一样，不过访问点的顺序有所不同，使用的是前面拓扑排序的逆后排序（也就是下面代码中的reversePost的顺序

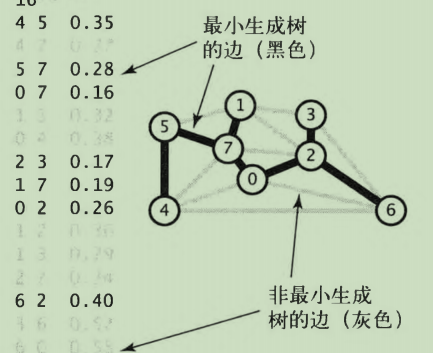


这个的整体思路是，我们假设按照正常图G进行深度搜索，S的顺序会在V前面，我们首先使用改图的反向图进行深度优先搜索，然后我们得知在V和S在同一个连通分量，所以有V到S的路径，因为是反向图所以实际上是S到V的路径，然后我们在对这个逆后序在正常图G进行深度搜索，逆后序中V在S前面，所以我们进行深度搜索，会先进行dfs（G，V），如果把S标记那么就找到了V到S的路，然后两个点强连通，其次我们这个逆序是一个单向连通分量，在这个基础上找出的连通分量就是真正的强连通分量。

**代码展示**

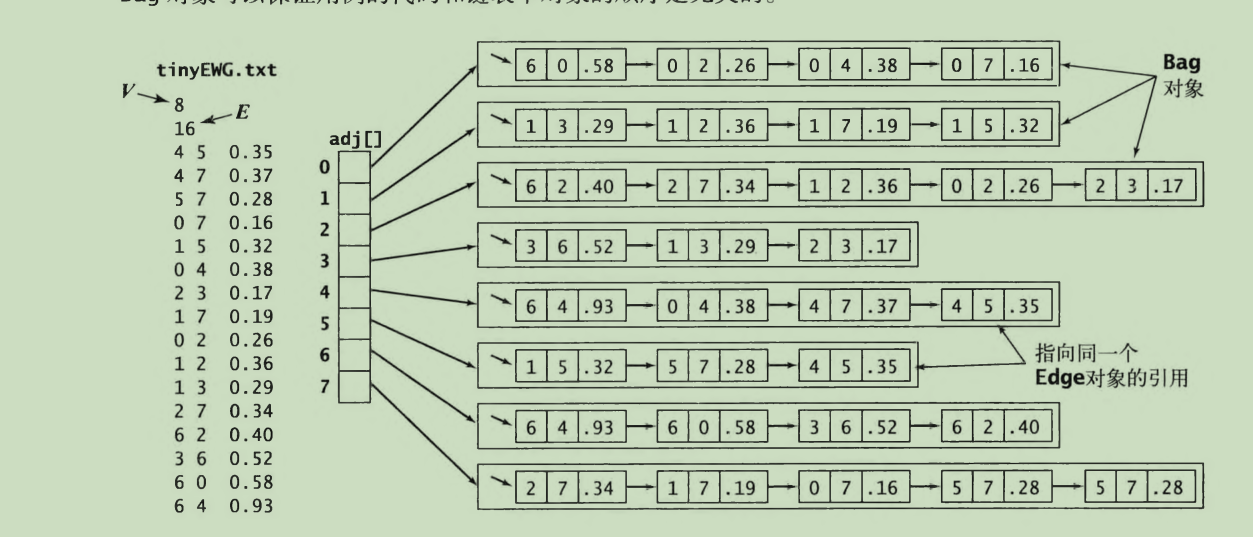


最小生成树



**定义**

图的生成树是它的一裸含有其所有顶点的无环连通子图。一幅加权无向图的最小生成树(M S T)是它的一裸权值（树中所有边的权值之和）最小的生成树。



我们对于最小生成树的数据结构的一种表示方式

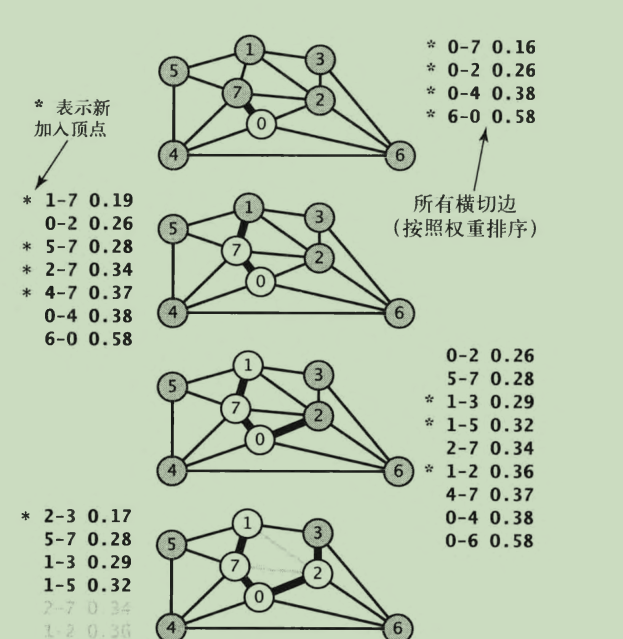
**最小生成树的方法**

**prim算法**

我们从一个点w开始，然后把上面的数据结构，点w所有链表放入到优先队列中，然后我们把w整条链表放入到优先队列中，也就是我们说的把w相关的边放入到优先队列中，从这些边中找出最短的那条边，让这条边失效，不参与下次比较，同时把最短的那条边的另一个顶点s，的所有边放入到优先队列中，然后再找出一条最短的边，同时防止成环

核心思想，每次找出距离树最近的那条边。

**优化 在于有些边不需要参与进来比较（p402没有理解）**



Kruskal 算法

我们要仔细学习的第二种最小生成树算法的主要思想是按照边的权重顺序（从小到大）处理它们，将边加人最小生成树中（图中的黑色边），加人的边不会与已经加入的边构成环，直到树中含有K-1条边为止。这些黑色的边逐渐由一片森林合并为一棵树，也就是图的最小生成树。这种计算方法被称为Kruskal 算法。

难点如何判断新加入的边是否会形成环，我们可以在加入v到w这条边之前，检查是否v和w连通。如果连通，再加上v到w这条边就会形成环，问题轻松解决。

**最短路径**

**默认**

**我们不会考虑这个图的最短路径存在环的问题，如果存在环，那么就会造成，这个最短距离可以绕着这个环一直旋转，距离无限放大，就不能成为最短距离。**

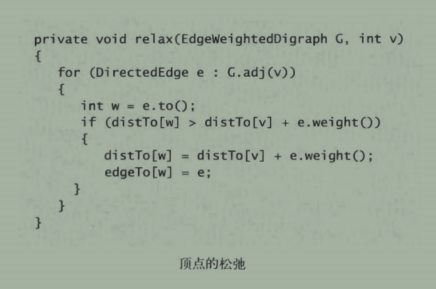
**边的松弛定义**

松弛意味着他松垮，所以他比最短边长，所以我们就要忽略他，这就是边的松弛，如果我们找到一个比这个边更短的边，我们就可以把原来那条边放弃，也就是让她松弛，不在使用，就是边的松弛成功。

在整个流程中edgeTO 和distTO数组显得最重要，edgeto 用于记录我们最短路径的边，比如 s到w边为e 所以我们级edgeto[w]为e，这样我们就可以记录整个最短路径，distTo这个数组是我们记录到某个顶点的最短距离，distTo[w]=10记录为当前到w最短的路径为10

edgeTo 和 distTo 数组直接支持 pathTo（）、hasPathTo（）和 distTo ()查询方法，只有在v 是从 s 可达的情况下，distTo [v ]才是有意义的，还已经约定，对于从s 不可达的顶点，distTo （） 方法都应该返回无穷大。在实现这个时，将distTo[ ] 中的所有元素都初始化为Double•POSITIVE.INFINITY，distTo [s ]则为0。最短路径算法会将从起点可达的顶点v 的 distTo [v ]设为一个有限值，这样就不必再用marked [ ] 数组来在图的搜索中标记可达的顶点，而是通 过检测distTo [ v ] 是 否 为 Doubl e\_POSITIVE.INFINITY 来实现 hasPathTo(v) 。对于 pathToO 方法，我们约定如果v 不是从起点可达的则返回null.

**核心思路代码**



**Dijkstra算法（有向无环图）**

**算法存在问题**

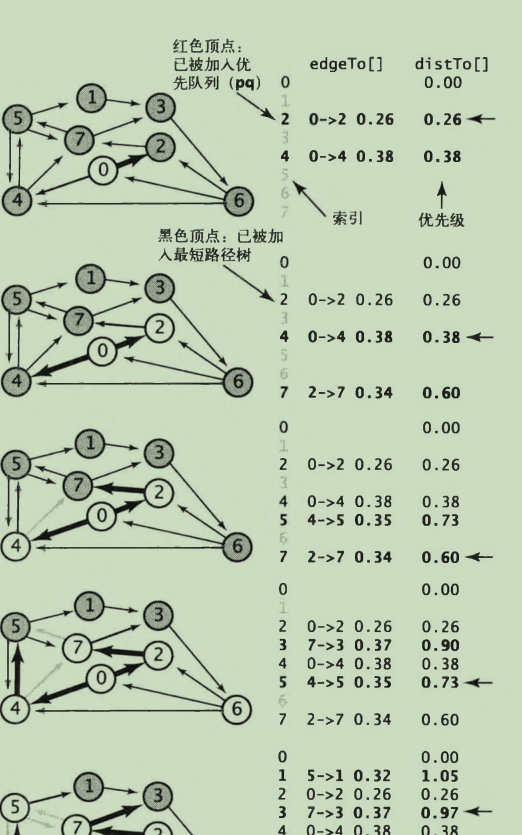
无法处理处理负权重的边

要实现Dijkstra算法，除了 distTo []和 edgeT0[ ] 数组之外还需要一条索引优先队列pq，以保存需要被放松的顶点并确认下一个被放松的顶点。

**核心思想**

的另一种方式就是将它和Prim算 法 相比较。两种算法都会用添加边的方式构造一棵树：Prim算法每次添加的都是离树最近的非树顶点，Dijkstra算法每次添加的都是离起点最近的非树顶点。

**这个算法的流程**



1.将顶点0 添加到树中，将顶点2 和 4 加入优先队列

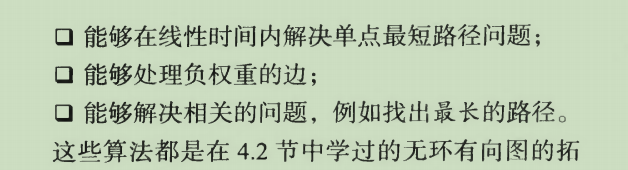
2.从优先队列中删除顶点2，将 0— 2 添加到树中，将顶点7 加入优先队列。

3.从优先队列中删除顶点4，将 0— 4 添加到树中，将顶点5 加入优先队列，边4——7 失效。

**这里我在看书的时候出现了重大错误理解（因为作者偷懒）**

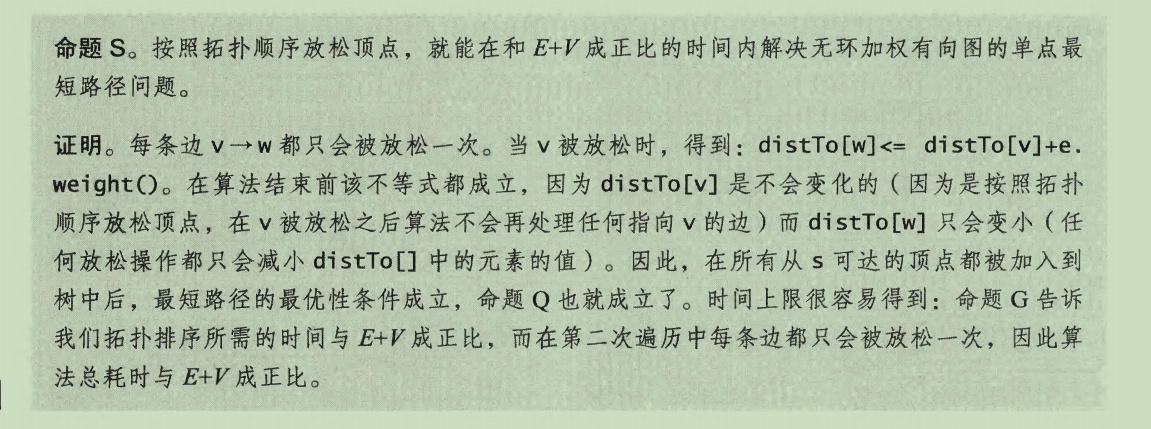
请看上面图片第三个图，第二图最右边箭头指向说明，队列中删除顶点4，接着肯定得找以4为起点边，我们从第三个图中看出，4到7这条边变成灰色了，我就在好奇为什么作者不用考虑他，难道作者可以肯定因为0到4这条边长于0到2，所以就可以知道0到4加4到7（在不知道4到7的距离）就长于 已有到7的最短距离（这里是按照算法流程，已经计算出目前到7的最短距离），我就在想为什么可以不用忽略4到7这条边，实际上把4到7这条边变成灰色，是因为作者以及确定了这条边松弛成功（也就是通过4到7这条路径大于现有的到7这个点的距离，所以他不用更新7这个点的距离，然后我们就把他失效，变成灰色，作者这里偷懒了，所以造成了错误理解，按照图片中的流程，他没有写出4到7，但实际上已经算过了，所以以后流程中，作者只是写出了最关键的流出，不是全部，为了让读者看起来简洁））

**基于拓扑排序算法的最短距离**

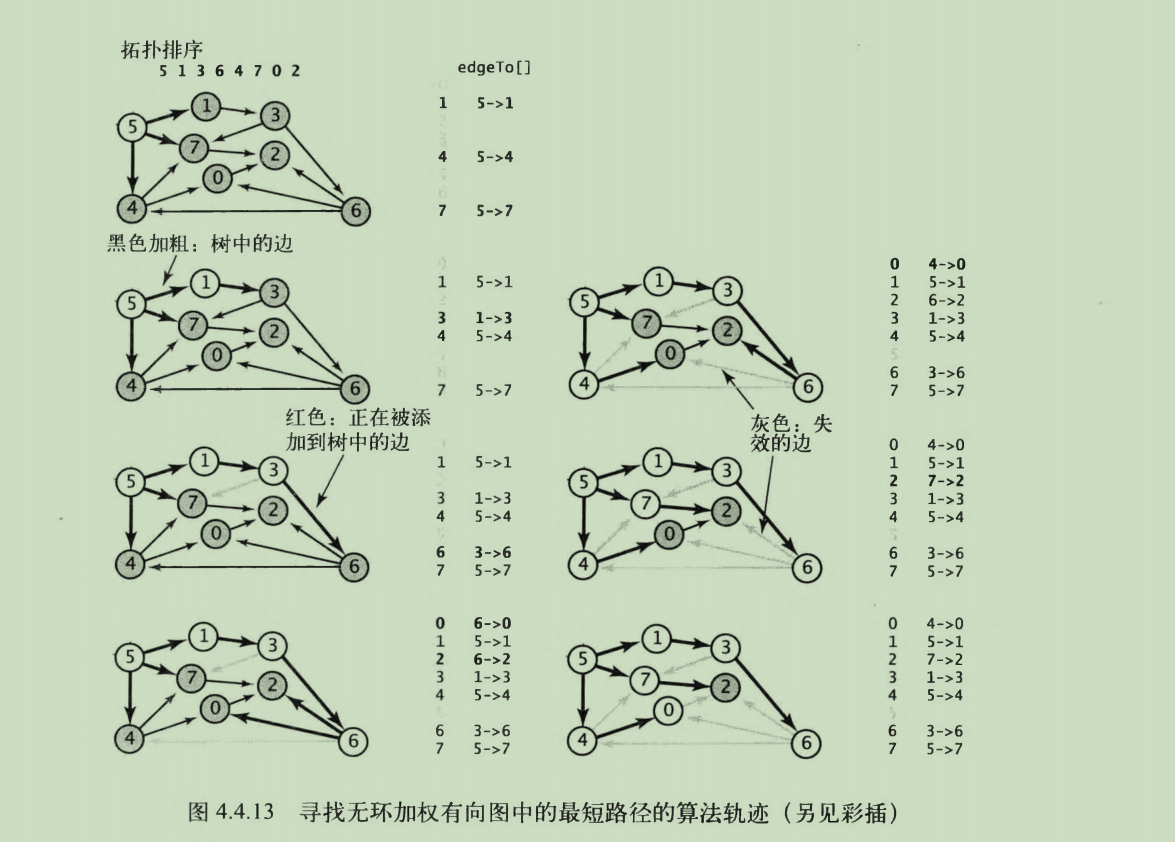


**思路**

拓扑排序顺序保证了我们在经历了拓扑排序某个顶点时，那个指向这个顶点w边都会被遍历一遍，所以这个点w的dist【w】的最小值都会得到确定。因为指向这个点的边都被遍历一遍，所以到这个点距离，就不会再被改变。



**算法顺序展示**



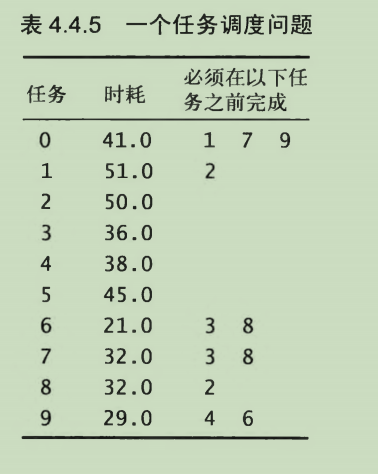
**最长距离**

**思路**

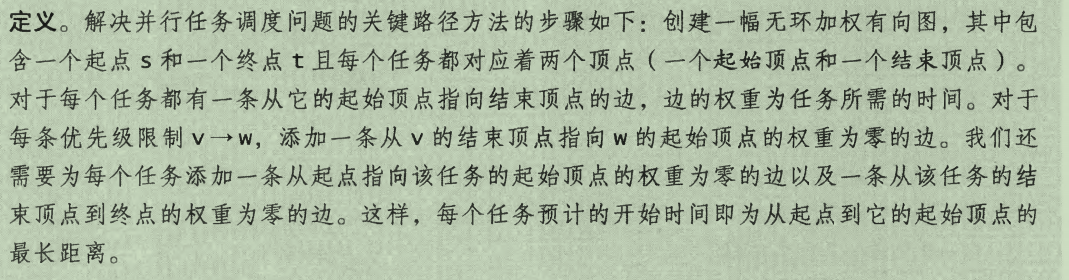
我们把图的边的权值添加负号，这样我们就可以利用上面的最短距离计算，实际上算出来的就是最长距离

**解决问题**

我们的拓扑排序是单核cpu，每次处理一个问题，时长就是所以时长的总和，这个我们是注重任务的先后顺序，但现在我们是多核cpu，可以同时处理多个任务，所以我们在意他的时长的时候就得重新寻找算法，我们可以使用最长距离解决这个问题，任务起点为w，结束的为v，我们把下面的任务关系变成一个图，然后我们可以找到多条路径到v，其中最长的那个距离，就是我们的任务最少时间，因为这个最长的时间路径都完成了，其他的路径肯定也完成了，这样我们所有的任务都完成了。我们称呼这条路径为关键路径。



**变成图的技巧**



**Bellman-Ford算法（看不懂）**