个进程和两种资源（打印机和绘图仪）的模型。横轴表示进程A执行的指令，纵轴表示进程B执行的指令。进程A在I_1处请求一台打印机，在I_3处释放，在I_2处请求一台绘图仪，在I_4处释放。进程B在I_5到I_7之间需要绘图仪，在I_6到I_8之间需要打印机。

图6-8中的每一点都表示出两个进程的连接状态。初始点为p，没有进程执行任何指令。如果调度程序选中A先运行，那么在A执行一段指令后到达q，此时B没有执行任何指令。在q点，如果轨迹沿垂直方向移动，表示调度程序选中B运行。在单处理机情况下，所有路径都只能是水平或垂直方向的，不会出现斜向的。因此，运动方向一定是向上或向右，不会向左或向下，因为进程的执行不可能后退。

当进程A由r向s移动穿过I_1线时，它请求并获得打印机。当进程B到达t时，它请求绘图仪。

图中的阴影部分是我们感兴趣的，画着从左下到右上斜线的部分表示在该区域中两个进程都拥有打印机，而互斥使用的规则决定了不可能进入该区域。另一种斜线的区域表示两个进程都拥有绘图仪，且同样不可进入。

如果系统一旦进入由I_1、I_2和I_5、I_6组成的矩形区域，那么最后一定会到达I_2和I_6的交叉点，这时就产生死锁。在该点处，A请求绘图仪，B请求打印机，而且这两种资源均已被分配。这整个矩形区域都是不安全的，因此决不能进入这个区域。在点t处惟一的办法是运行进程A直到I_4，过了I_4后，可以按任何路线前进，直到终点u。

需要注意的是，在点t进程B请求资源。系统必须决定是否分配。如果系统把资源分配给B，系统进入不安全区域，最终形成死锁。要避免死锁，应该将B挂起，直到A请求并释放绘图仪。

6.5.2 安全状态和不安全状态

我们将要研究的死锁避免算法使用了图6-6中的有关信息。在任何时刻，当前状态包括了E、A、C和R。如果没有死锁发生，并且即使所有进程突然请求对资源的最大需求，也仍然存在某种调度次序能够使得每一个进程运行完毕，则称该状态是安全的。通过使用一个资源的例子很容易说明这个概念。在图6-9a中有一个A拥有3个资源实例但最终可能会需要9个资源实例的状态。B当前拥有2个资源实例，将来共需要4个资源实例。同样，C拥有2个资源实例，还需要另外5个资源实例。总共有10个资源实例，其中有7个资源已经分配，还有3个资源是空闲的。

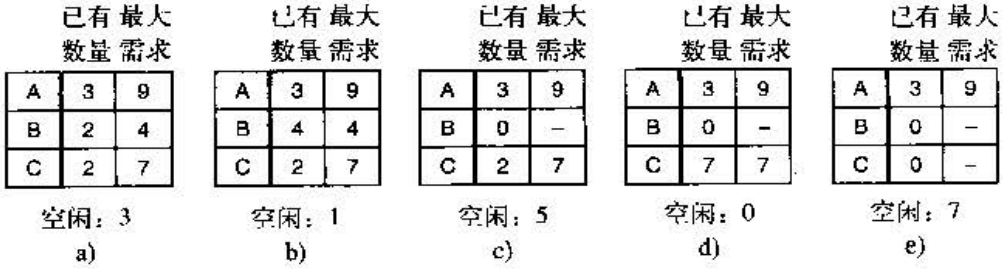

图6-9　说明a中的状态为安全状态

图6-9a的状态是安全的，这是由于存在一个分配序列使得所有的进程都能完成。也就是说，这个方案可以单独地运行B，直到它请求并获得另外两个资源实例，从而到达图6-9b的状态。当B完成后，就到达了图6-9c的状态。然后调度程序可以运行C，再到达图6-9d的状态。当C完成后，到达了图6-9e的状态。现在A可以获得它所需要的6个资源实例，并且完成。这样系统通过仔细的调度，就能够避免死锁，所以图6-9a的状态是安全的。

现在假设初始状态如图6-10a所示。但这次A请求并得到另一个资源，如图6-10b所示。我们还能找到一个序列来完成所有工作吗？我们来试一试。调度程序可以运行B，直到B获得所需资源，如图6-10c所示。

最终，进程B完成，状态如图6-10d所示，此时进入困境了。只有4个资源实例空闲，并且所有活动进程都需要5个资源实例。任何分配资源实例的序列都无法保证工作的完成。于是，从图6-10a到图6-10b的分配方案，从安全状态进入到了不安全状态。从图6-10c的状态出发运行进程A或C也都不行。回过头来再看，A的请求不应该满足。

值得注意的是，不安全状态并不是死锁。从图6-10b出发，系统能运行一段时间。实际上，甚至有