同，所以需要移走该节点，返回到前一个节点，即当前节点前面的一个节点，并将它作为新的当前节点，同时转到第3步。

这一算法是依次将每一个节点作为一棵树的根节点，并进行深度优先搜索。如果再次碰到已经遇到过的节点，那么就算找到了一个环。如果从任何给定的节点出发的弧都被穷举了，那么就回溯到前面的节点。如果回溯到根并且不能再深入下去，那么从当前节点出发的子图中就不包含任何环。如果所有的节点都是如此，那么整个图就不存在环，也就是说系统不存在死锁。

为了验证一下该算法是如何工作的，我们对图6-5a运用该算法。算法对节点次序的要求是任意的，所以可以选择从左到右、从上到下进行检测，首先从R节点开始运行该算法，然后依次从A、B、C、S、D、T、E、F开始。如果遇到了一个环，那么算法停止。

我们先从R节点开始，并将L初始化为空表。然后将R添加到空表中，并移动到惟一可能的节点A，将它添加到L中，变成L=[R，A]。从A我们到达S，并使L=[R，A，S]。S没有出发的弧，所以它是条死路，迫使我们回溯到A。既然A没有任何没有标记的出发弧，我们再回溯到R，从而完成了以R为起始点的检测。

现在我们重新以A为起始点启动该算法，并重置L为空表。这次检索也很快就结束了，所以我们又从B开始。从B节点我们顺着弧到达D，这时L=[B，T，E，V，G，U，D]。现在我们必须随机选择。如果选S点，那么走进了死胡同并回溯到D。接着选T并将L更新为[B，T，E，V，G，U，D，T]，在这一点上我们发现了环，算法结束。

这种算法远不是最佳算法，较好的一种算法参见（Even,1979）。但毫无疑问，该实例表明确实存在检测死锁的算法。

6.4.2 每种类型多个资源的死锁检测

如果有多种相同的资源存在，就需要采用另一种方法来检测死锁。现在我们提供一种基于矩阵的算法来检测从P_1到P_n这n个进程中的死锁。假设资源的类型数为m，E_1代表资源类型1，E_2代表资源类型2，E_i代表资源类型i($1 \leqslant i \leqslant m$)。$E$是现有资源向量（existing resource vector），代表每种已存在的资源总数。比如，如果资源类型1代表磁带机，那么E_1=2就表示系统有两台磁带机。

在任意时刻，某些资源已被分配所以不可用。假设A是可用资源向量（available resource vector），那么A_i表示当前可供使用的资源数（即没有被分配的资源）。如果仅有的两台磁带机都已经分配出去了，那么A_1的值为0。

现在我们需要两个数组：C代表当前分配矩阵（current allocation matrix），R代表请求矩阵（request matrix）。C的第i行代表P_i当前所持有的每一种类型资源的资源数。所以，C_{ij}代表进程i所持有的资源j的数量。同理，R_{ij}代表P_i所需要的资源j的数量。这四种数据结构如图6-6所示。

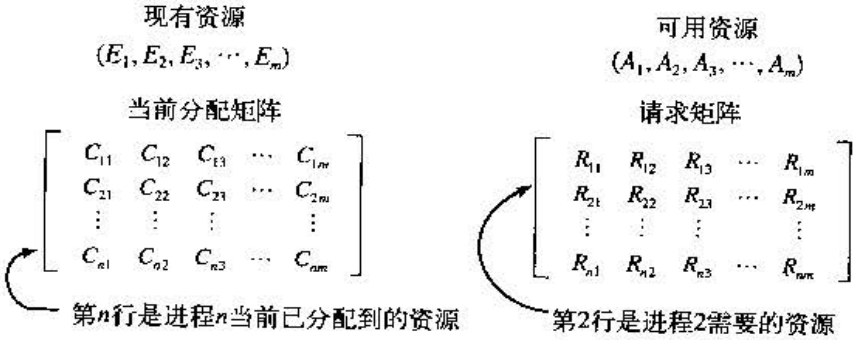

图6-6 死锁检测算法所需的四种数据结构

这四种数据结构之间有一个重要的恒等式。具体地说，某种资源要么已分配要么可用。这个结论意味着：

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

换言之，如果我们将所有已分配的资源j的数量加起来再和所有可供使用的资源数相加，结果就是该类资源的资源总数。