**1.5 案例研究：union-find算法**

**1.5.1 动态连通性**

编程：动态连通性问题

当程序从输入中读取了整数对p和q时，如果已知的所有整数对都不能说明p和q是相连的，那么则将这一对整数写入到输出中。

定义分量类UF和union()，find()，connected()和count()方法。

**1.5.2 实现**

quick-find算法：UF包含一个数组，一种方法是保证当且仅当id[p]等于id[q]时p和q是连通的。

如果id[p]和id[q]不同时，需要遍历整个数组，将所有和id[q]相等的值变为id[p]的值。

quick-find无法处理大型问题，因为对于每一对输入union()都需要扫描整个id[]数组。

quick-union算法

编程：quick-union算法实现

每个触点所对应的id[]元素都是同一个分量中的另一个触点的名称（也可能是它自己），我们将这种联系称为链接。

find方法寻找到根触点，即链接指向自己的触点。

当且仅当分别由两个触点开始的这个多层到达同一个根触点时，它们存在于同一个连通分量中。

union很简单，只需要将一个根触点链接到另一个根触点即可。因此这个算法叫quick-union。

id[]用父链接的方式表示了一片森林。

加权quick-union算法

编程：quick-union算法实现

记录每一棵树的大小并总是将较小的树连接到较大的树上。我们称它为加权算法。

quick-union算法能够保证对数级别的性能。

对于N个触点，加权quick-union算法构造的森林中的任意结点的深度最多为lgN。

路径压缩的加权quick-union算法是最优的算法。

**1.5.3 展望**