

Disjoint Set

并查集

描述:

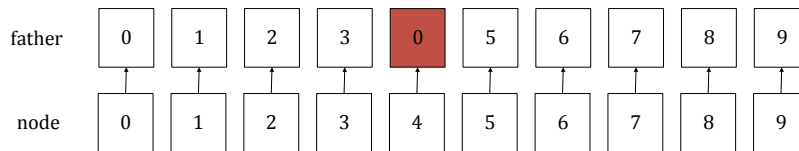
并查集是一种高效的用于检查成员分类的树形数据结构,用于对一组成员的集合进行分类、合并、查询,这些成员属于不同的“家庭”。每个成员具有一个父节点指针,以父节点是否相同来区分两个成员是否属于同一家庭。当集合中的所有成员最终分为 2 个家庭,则所有成员所属的父节点只有 2 种可能。并查集的时间复杂度为 $O(\alpha(n))$,在实际情况中 $\alpha(n) < 5$ 。

并查集的核心操作是查询父节点,这个操作实际上是查询祖宗节点。设 $father[x]$ 为 x 节点的父节点,当 $father[x] = x$ 时,称 x 为一个祖宗节点,设 $ancestor[x]$ 是 x 的祖宗节点。该方法可以压缩查询时搜索的节点数量,称为路径压缩技术。

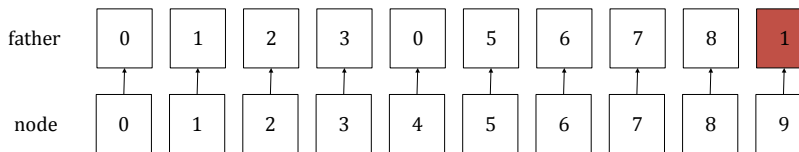
将下面的集合 $s = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,共 10 个成员,分成两个家庭 A 和 B。每个成员都有父节点,初始时所有成员的父节点都指向自己。如图所示:



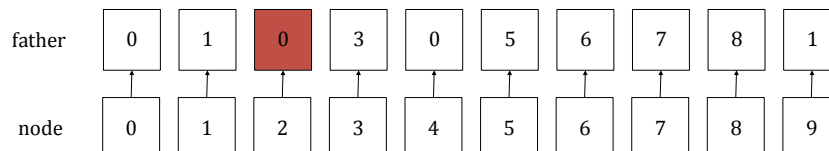
- (1) 声明 0 和 4 属于同一家庭,比较 4 和 0 的祖宗节点,因为 $ancestor[0] < ancestor[4]$,设置 $father[4] = ancestor[0] = 0$ (设置父节点的规则可以根据实际要求进行设计,但两个节点的其中一个的父节点必须设置为另一个节点的祖宗点),本文中我们取节点编号较小的祖宗节点作为另一个节点的父节点;



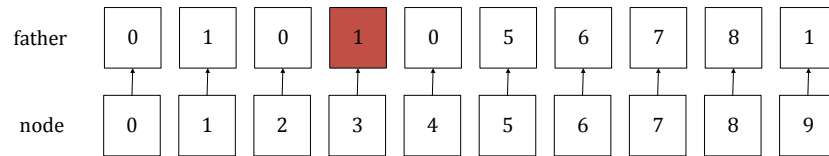
- (2) 声明 1 和 9 节点属于同一家庭,因为 $ancestor[1] < ancestor[9]$,设置 $father[9] = ancestor[1] = 1$;



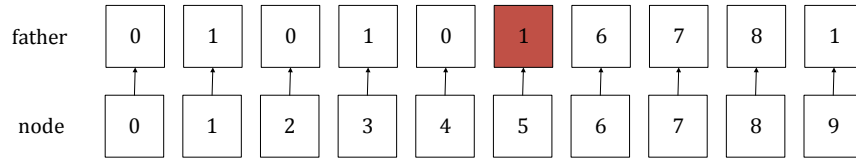
- (3) 声明 2 和 4 节点属于同一家庭,因为 $ancestor[2] > ancestor[4]$,设置 $father[2] = ancestor[4] = 0$;



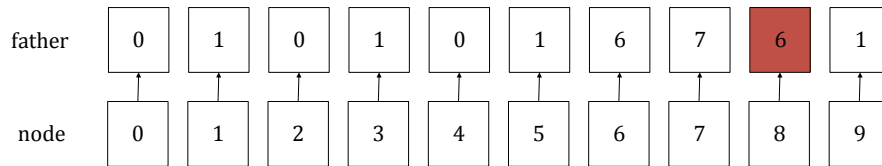
- (4) 声明 3 和 1 节点属于同一家庭,因为 $ancestor[3] > ancestor[1]$,设置 $father[3] = ancestor[1] = 1$;



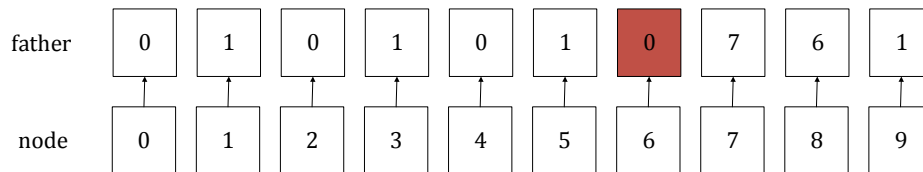
(5) 声明 3 和 5 节点属于同一家庭，因为 $ancestor[5] > ancestor[3]$ ，设置 $father[5] = ancestor[3] = 1$;



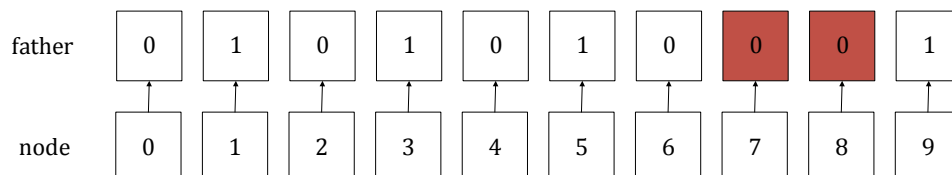
(6) 声明 6 和 8 节点属于同一家庭，因为 $ancestor[8] > ancestor[6]$ ，设置 $father[8] = ancestor[6] = 6$;



(7) 声明 6 和 2 节点属于同一家庭，因为 $ancestor[6] > ancestor[2]$ ，设置 $father[6] = ancestor[2] = 0$;



(8) 声明 7 和 8 节点属于同一家庭，由于 $father[8] \neq ancestor[8]$ ，更新 8 的父节点 $father[8] = ancestor[8] = 0$ ，因为 $ancestor[7] > ancestor[8]$ ，设置 $father[7] = ancestor[8] = 0$;



合并两节点 x 和 y 时，根据固定规则设置 $father[x] = ancestor[y]$ ；查询节点 x 的祖宗节点时，若 $father[x] \neq ancestor[x]$ 则设置 $father[x] = ancestor[x]$ 。并查集的合并、查询操作的时间复杂度接近 $O(1)$ 。