

玩儿转数据结构

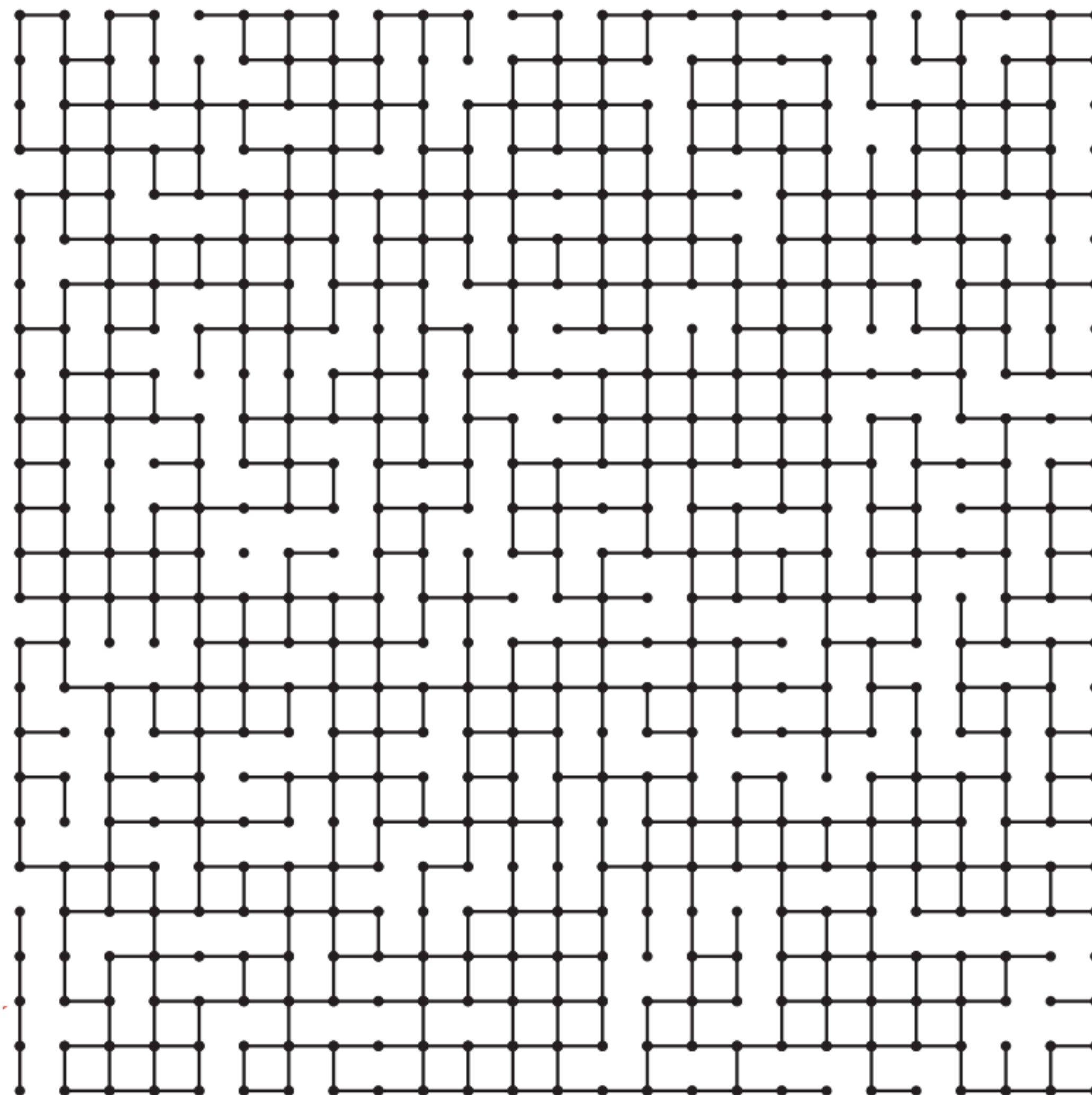
liuyubobobo

并查集 Union Find

一种很不一样的树形结构

连接问题 Connectivity Problem

连接问题



连接问题

网络中节点间的连接状态

- 网络是个抽象的概念： 用户之间形成的网络

数学中的集合类实现

连接问题和路径问题

比路径问题要回答的问题少

- 和堆作比较

并查集 Union Find

对于一组数据，主要支持两个动作：

- `union(p , q)`
- `isConnected(p , q)`

实践：并查集的接口设计

Quick Find

并查集 Union Find

对于一组数据，主要支持两个动作：

- `union(p , q)`
- `isConnected(p , q)`

并查集的基本数据表示

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1

并查集的基本数据表示

id	0	1	2	3	4	5	6	7	8	9
	0	1	0	1	0	1	0	1	0	1

并查集 Union Find

对于一组数据，主要支持两个动作：

- `union(p , q)`
- `isConnected(p , q)`

并查集 Union Find

对于一组数据，主要支持两个动作：

- `union(p , q)`
- `isConnected(p , q)`  `find(p) == find(q)`

Quick Find

id	0	1	2	3	4	5	6	7	8	9
	0	1	0	1	0	1	0	1	0	1

Quick Find 时间复杂度 $O(1)$

Quick Find 下的 Union

`union(1, 4)`

id	0	1	2	3	4	5	6	7	8	9
	0	1	0	1	0	1	0	1	0	1



Quick Find 下的 Union

id	0	1	2	3	4	5	6	7	8	9
	1	1	1	1	1	1	1	1	1	1

Quick Find 下的 Union 时间复杂度 $O(n)$

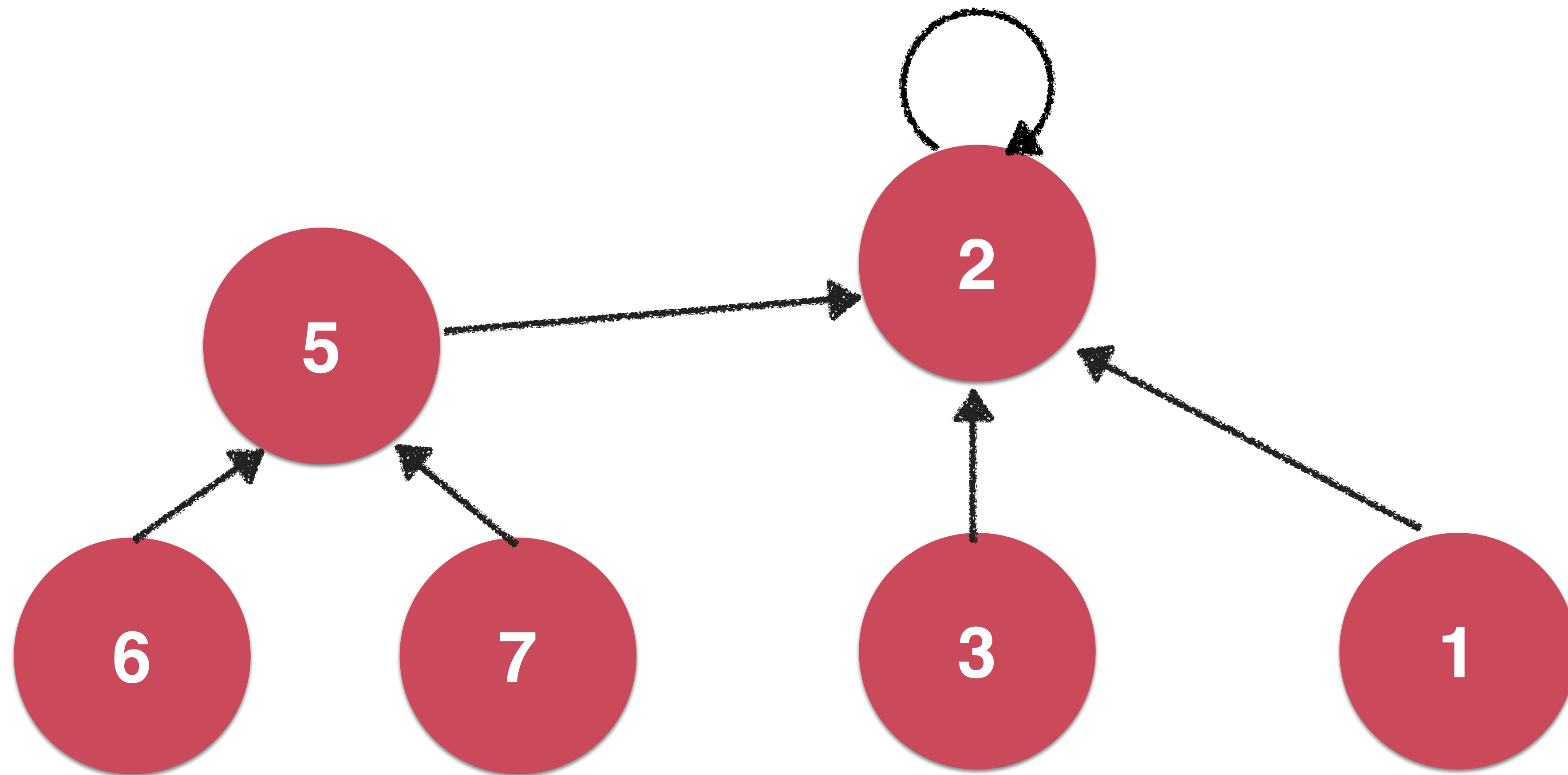
实践： Quick Find

Quick Find

- `unionElements(p , q)`  $O(n)$
- `isConnected(p , q)`  $O(1)$

Quick Union

将每一个元素，看做是一个节点



Quick Union 下的数据表示

parent

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

Quick Union



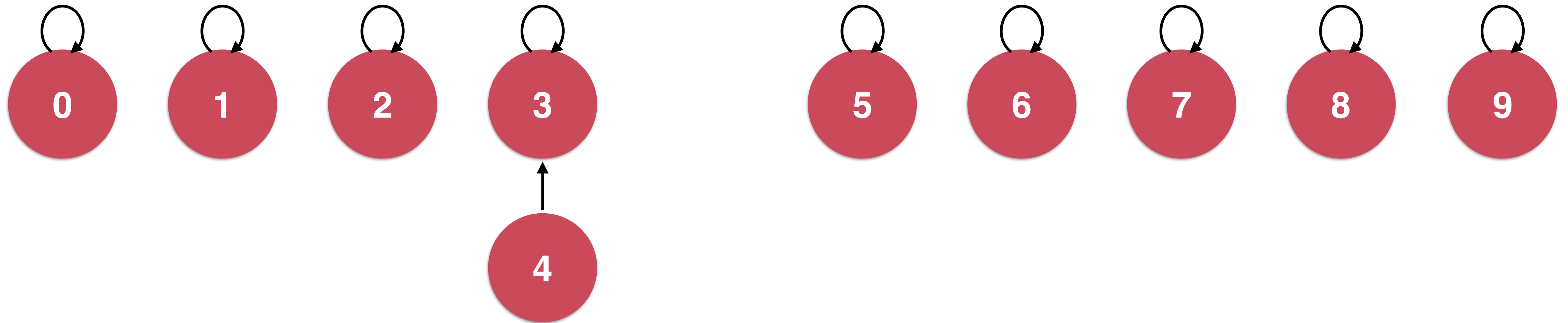
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

union 4 , 3



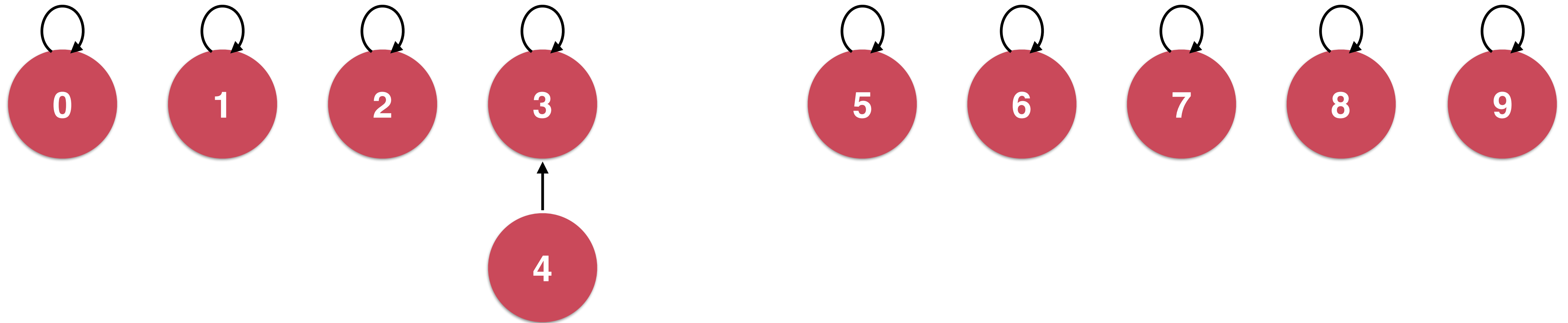
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

union 4 , 3



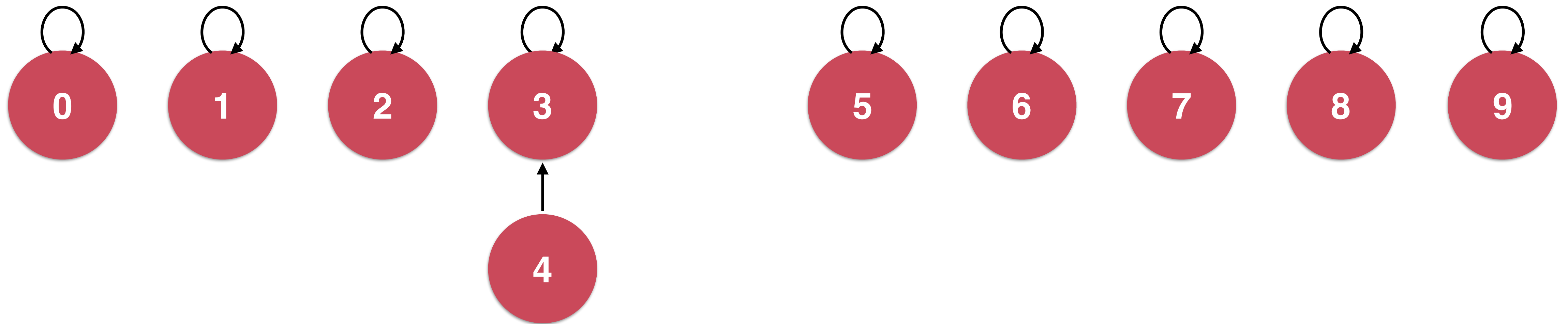
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

union 4 , 3



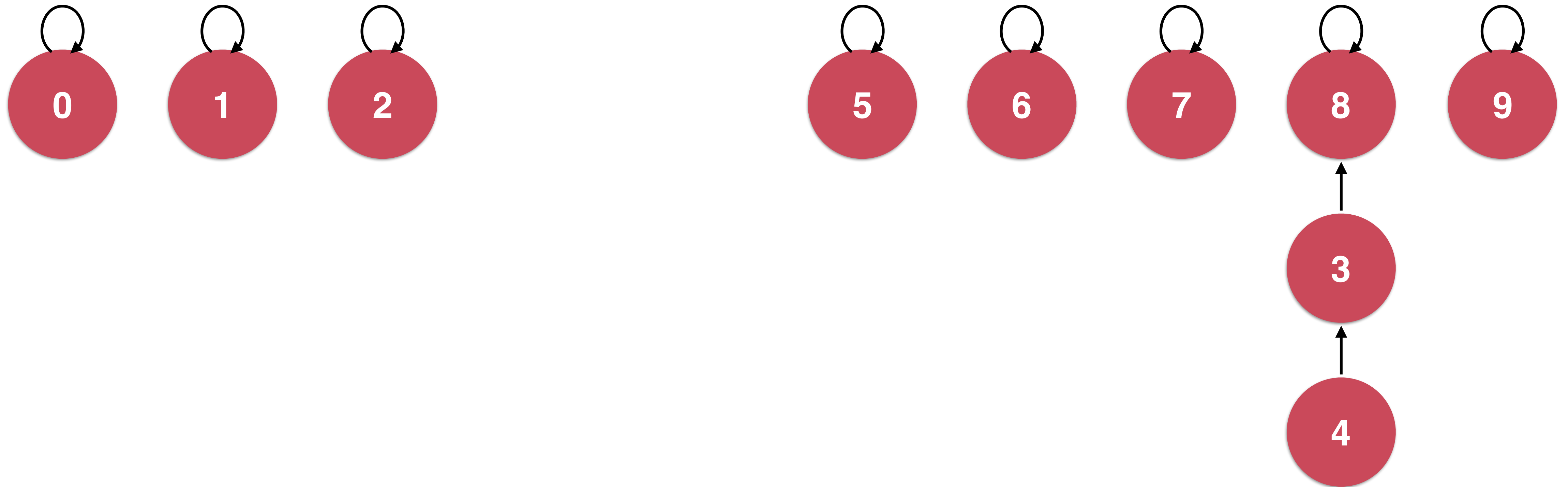
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	3	5	6	7	8	9

union 3 , 8



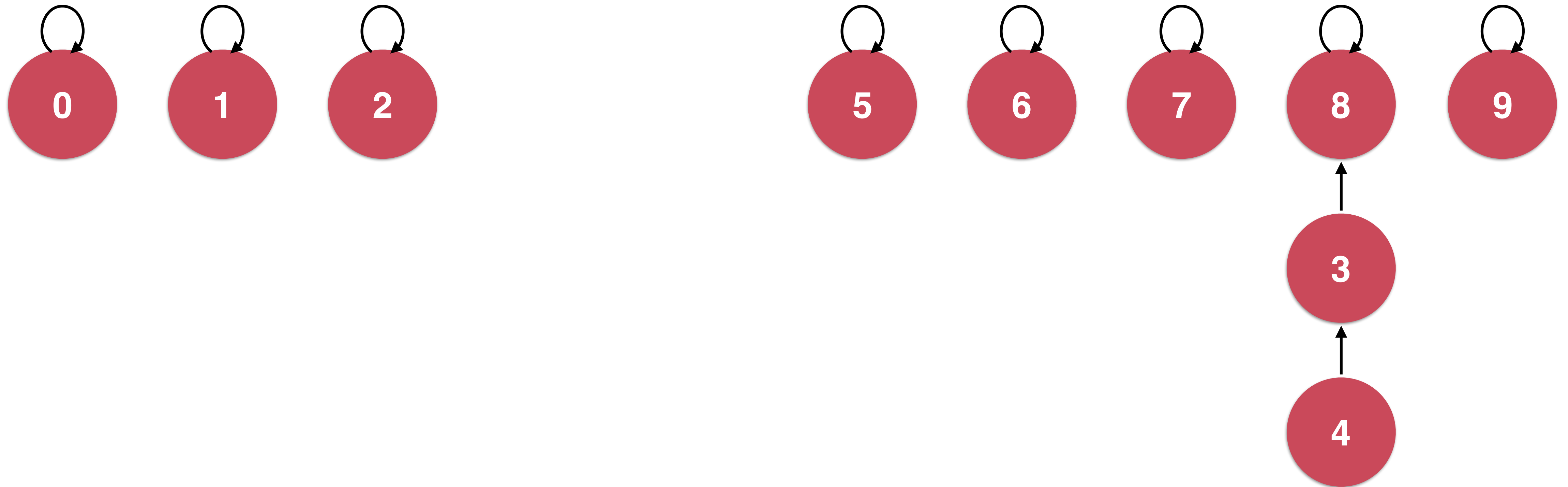
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	3	5	6	7	8	9

union 3 , 8



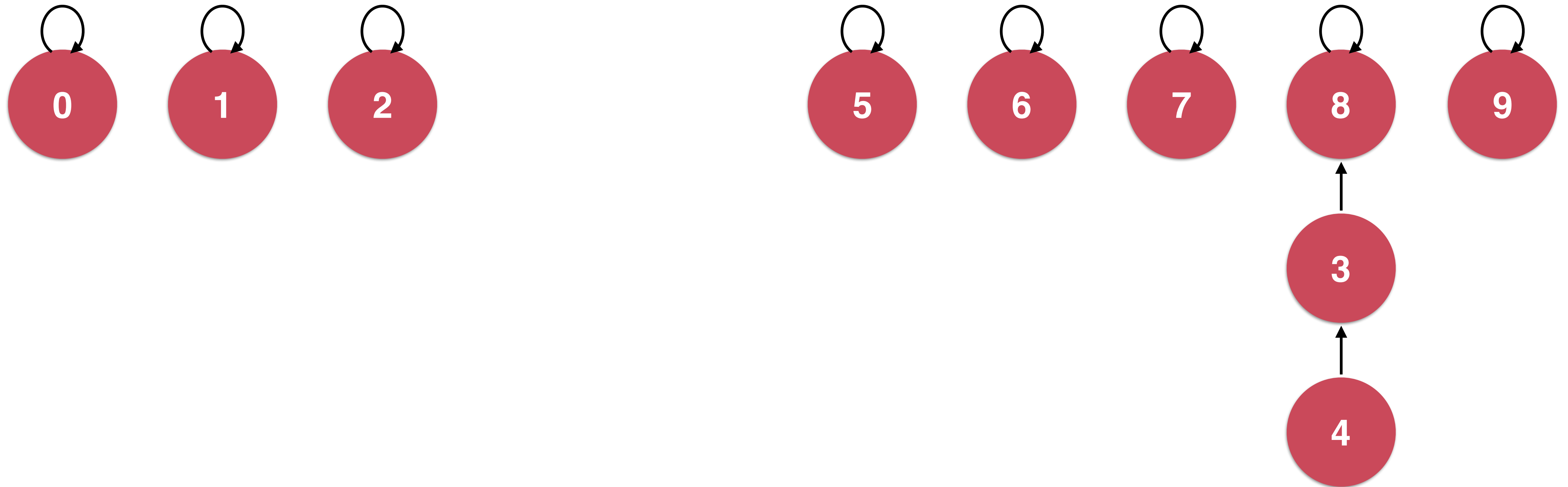
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	3	5	6	7	8	9

union 3 , 8



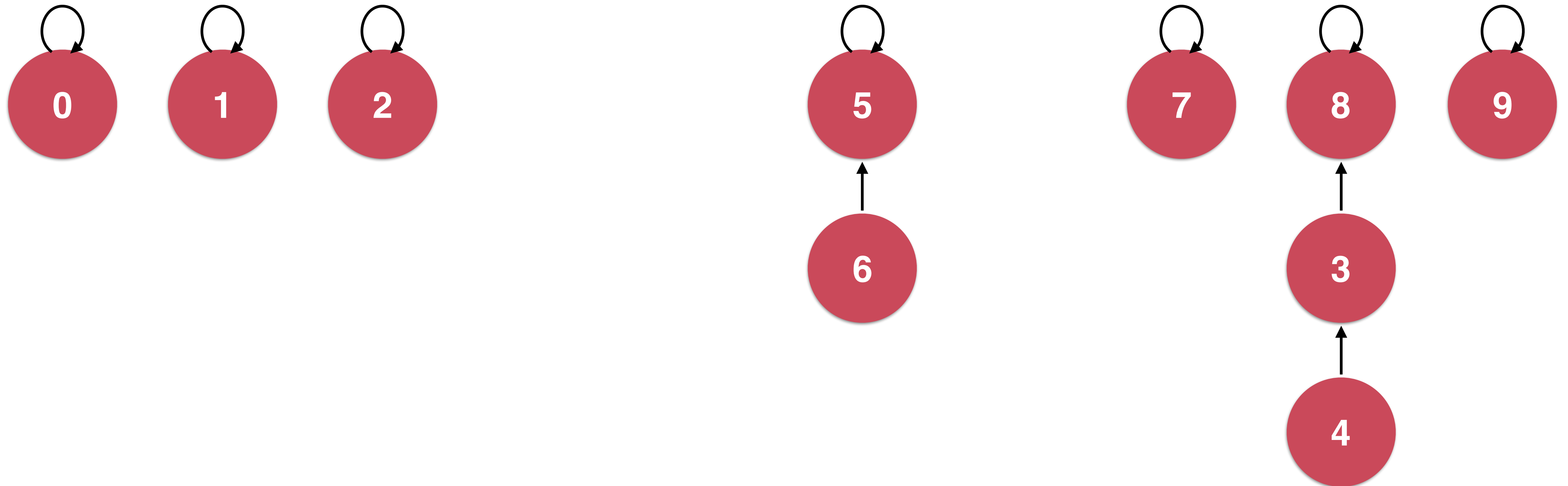
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	3	5	6	7	8	9

union 6 , 5



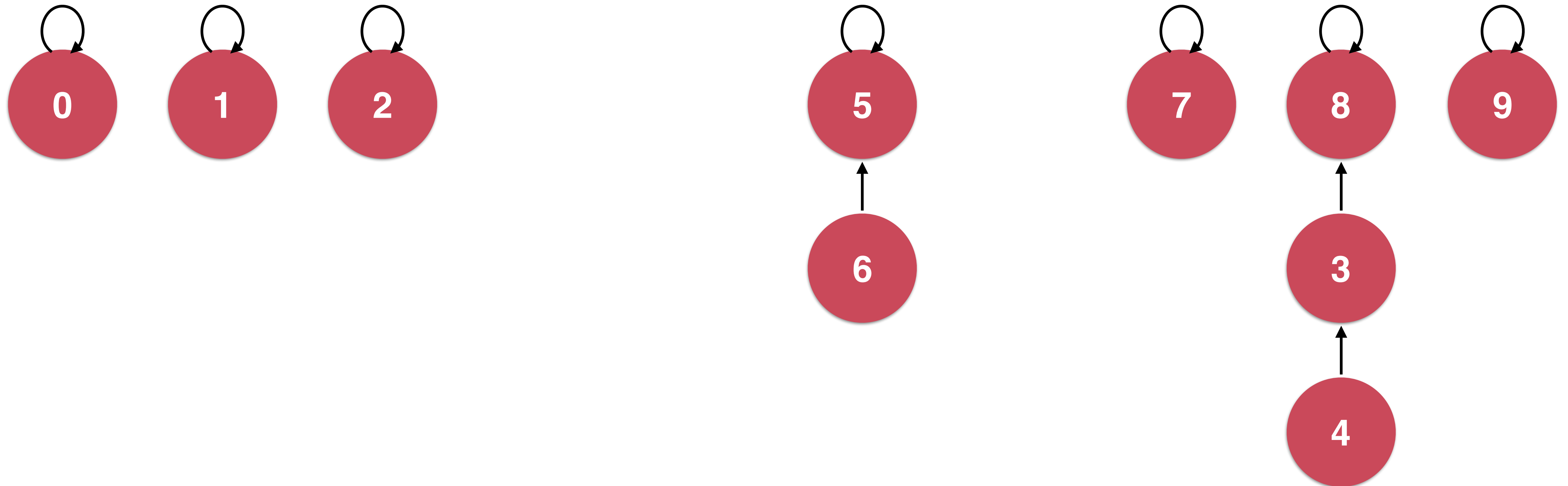
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	3	5	6	7	8	9

union 6 , 5



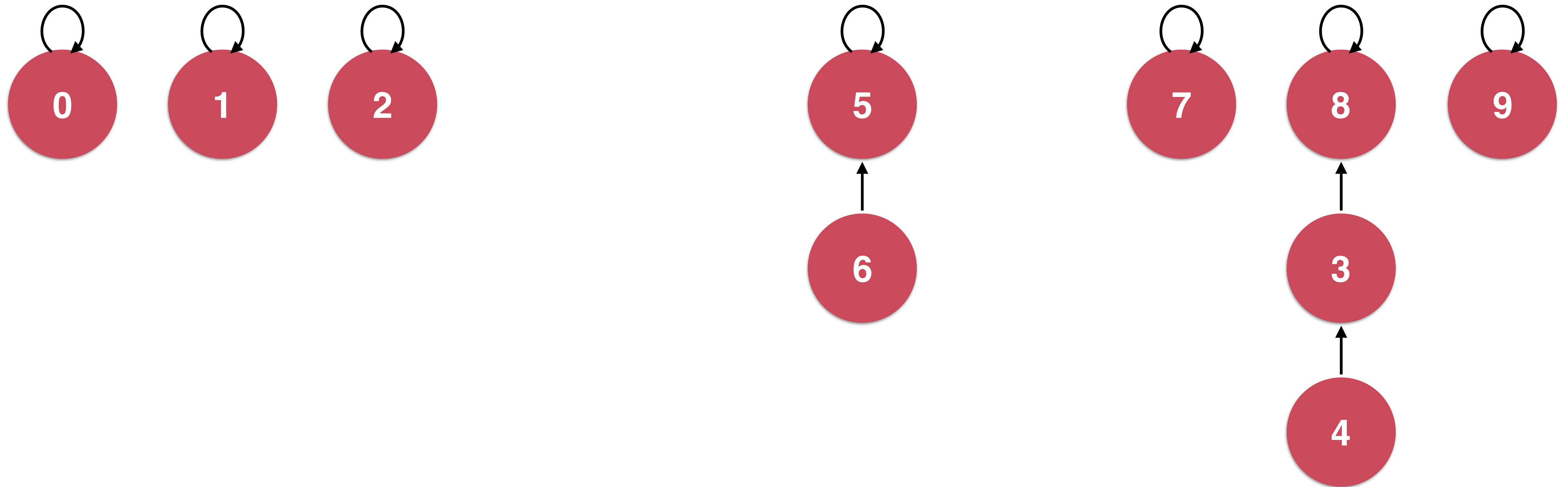
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	3	5	6	7	8	9

union 6 , 5



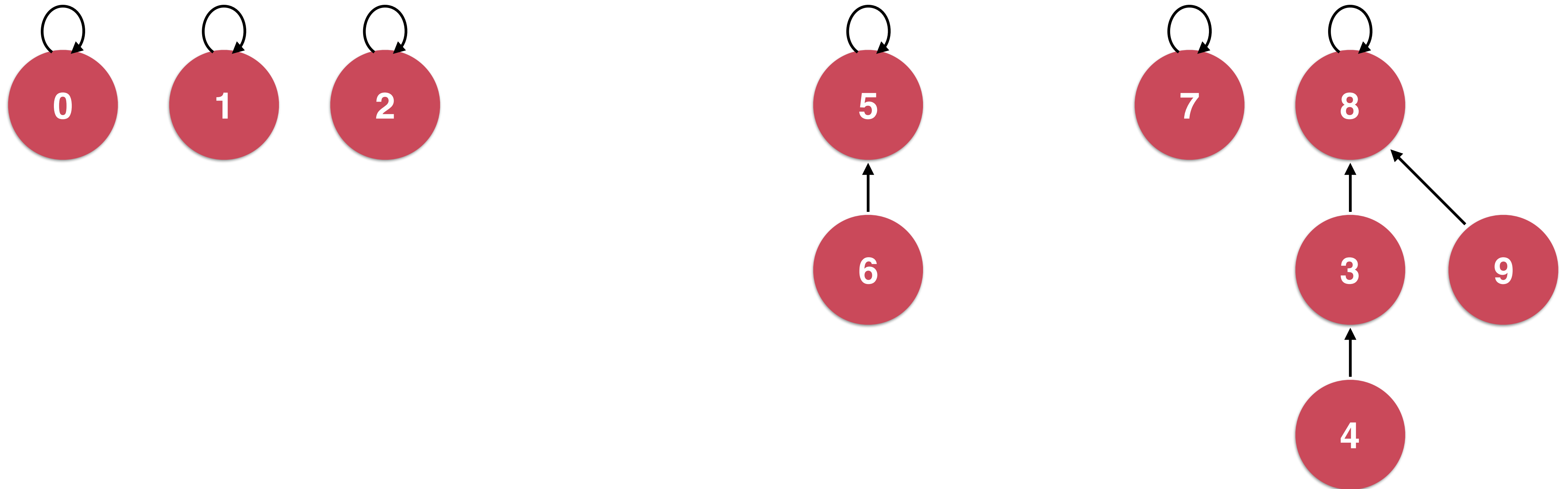
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	3	5	5	7	8	9

union 9 , 4



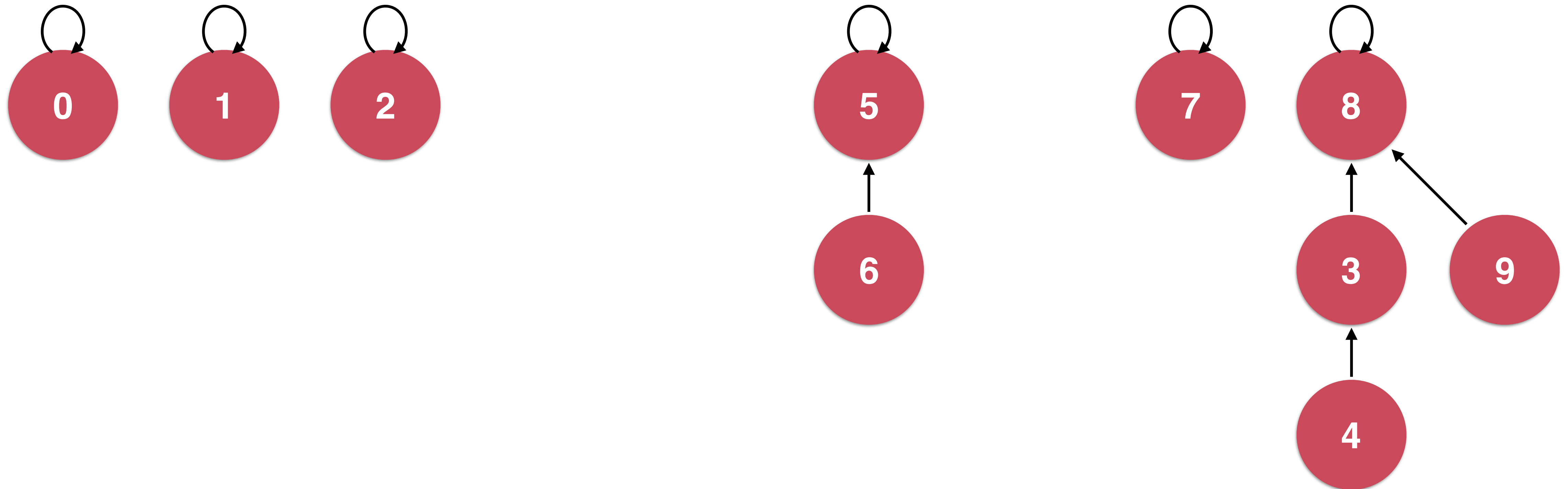
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	3	5	5	7	8	9

union 9 , 4



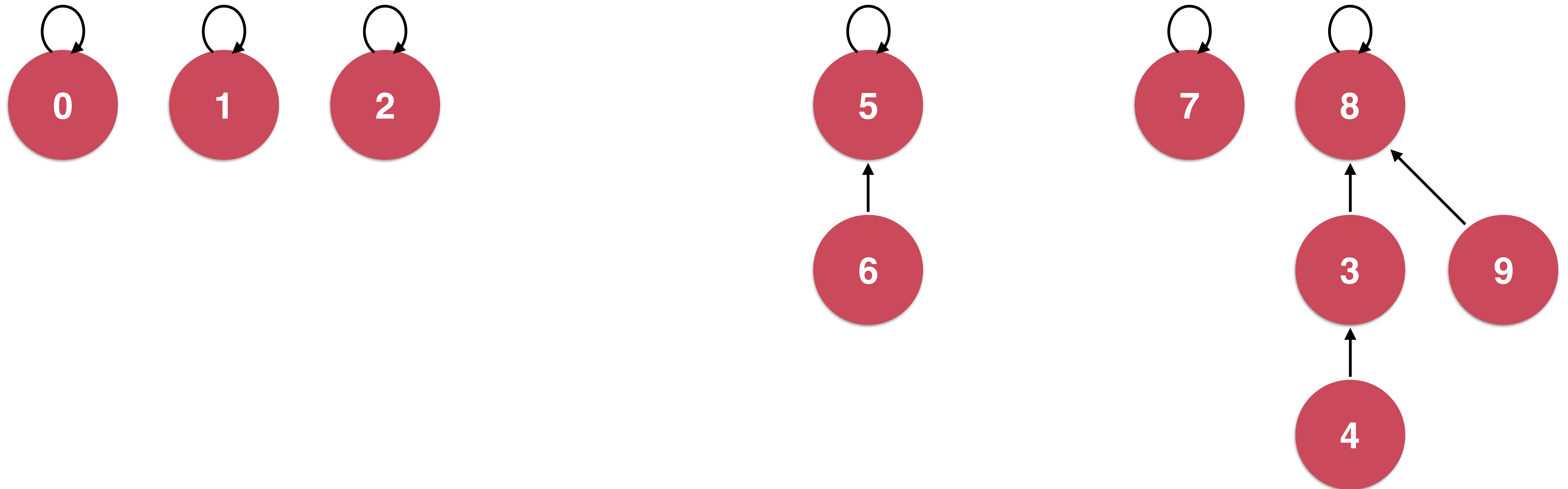
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	3	5	5	7	8	9

union 9 , 4



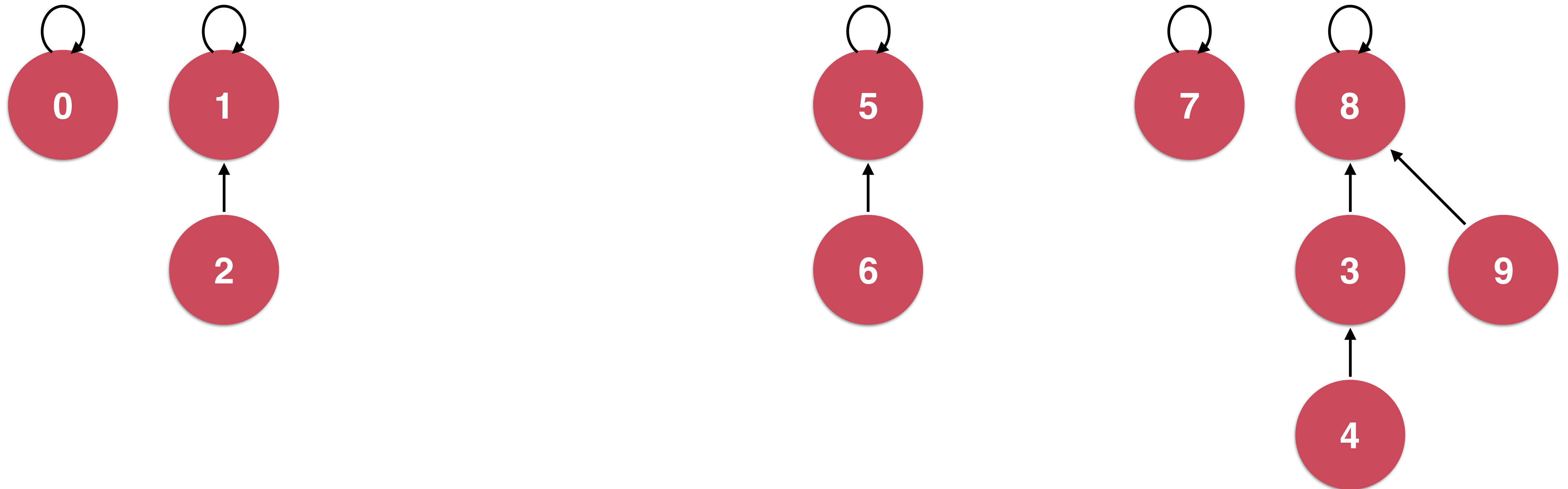
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	3	5	5	7	8	8

union 2 , 1



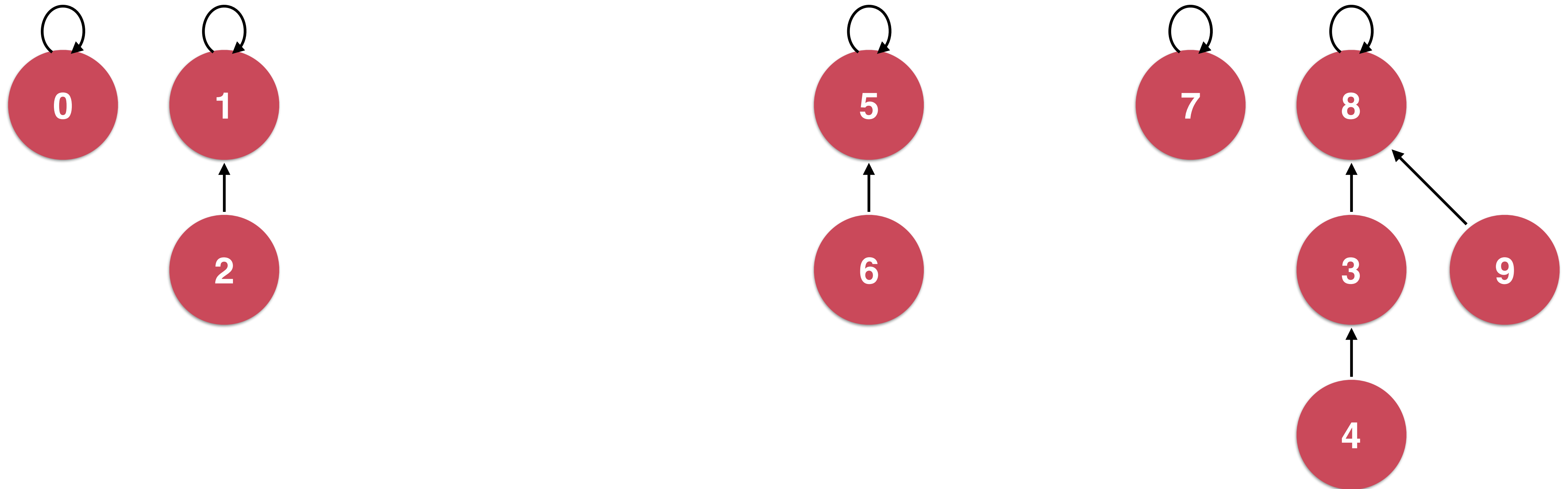
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	3	5	5	7	8	8

union 2 , 1



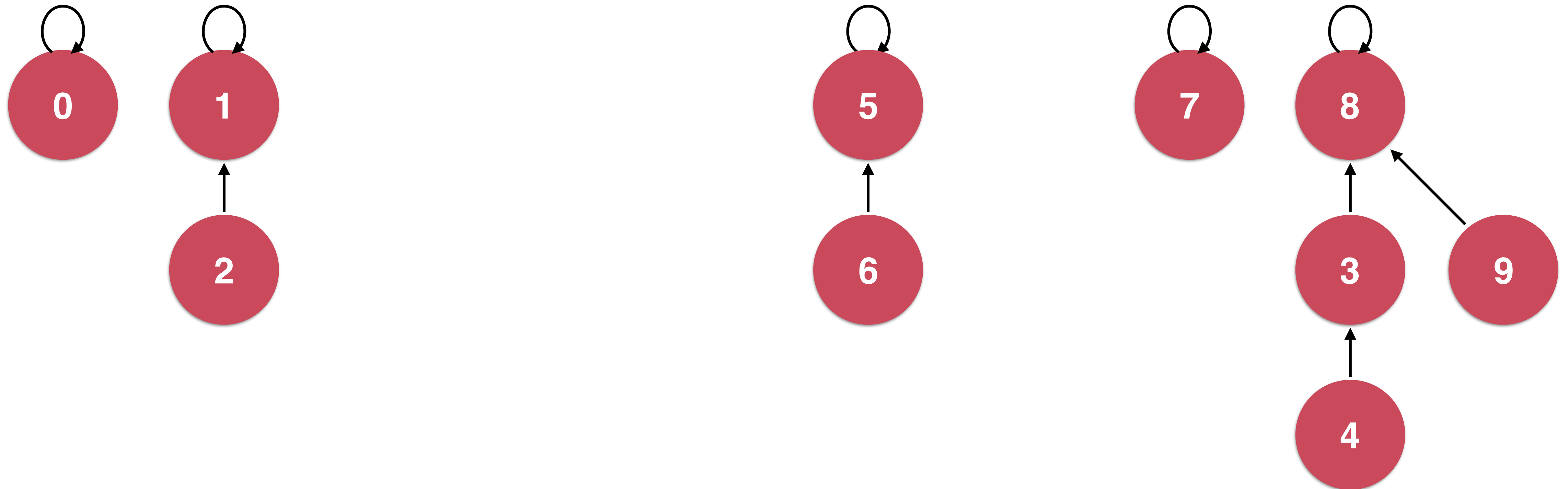
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	3	5	5	7	8	8

union 2 , 1



parent	0	1	2	3	4	5	6	7	8	9
	0	1	1	8	3	5	5	7	8	8

union 5 , 0



parent	0	1	2	3	4	5	6	7	8	9
	0	1	1	8	3	5	5	7	8	8

union 5 , 0



parent

0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

union 5 , 0



parent

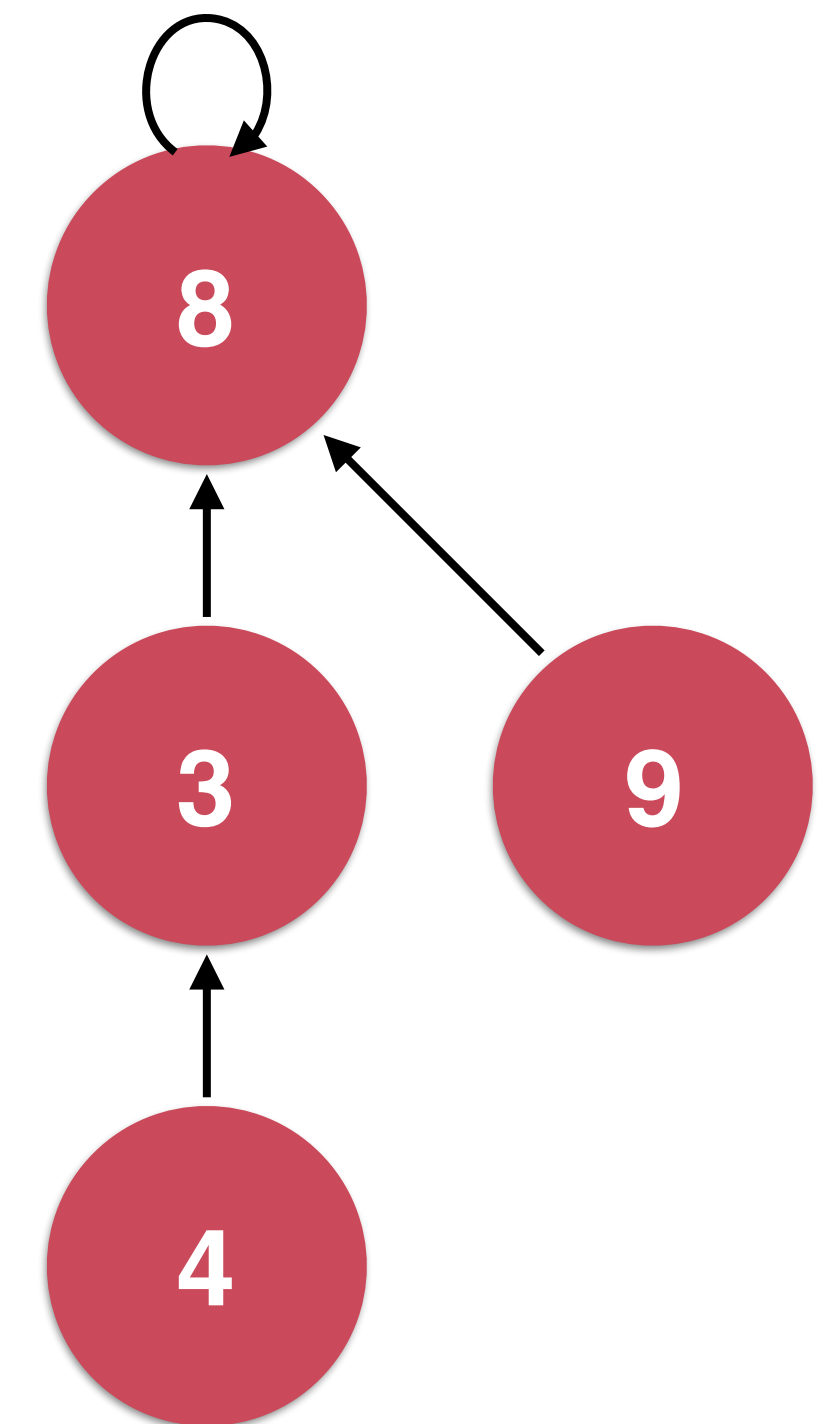
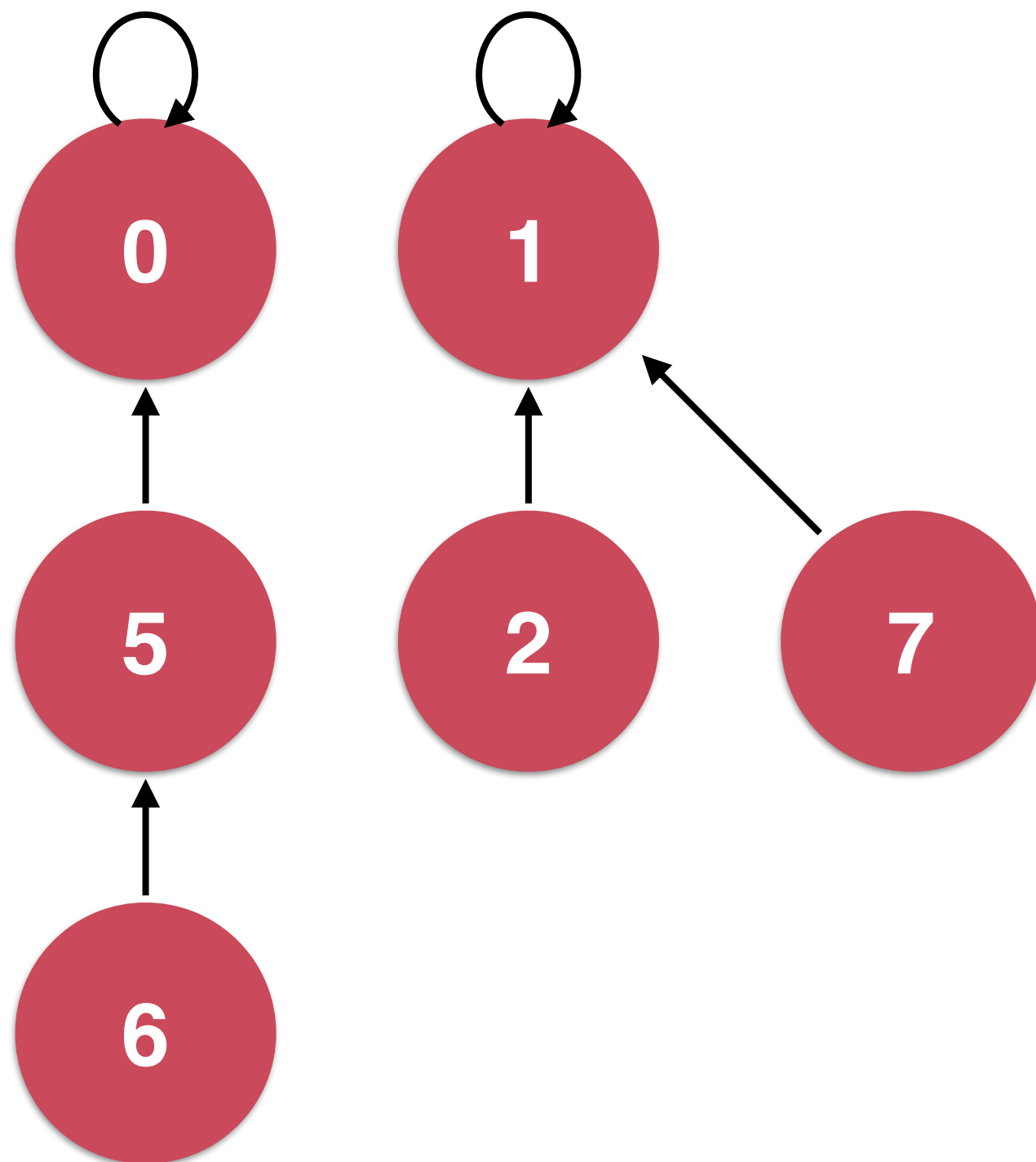
0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	7	8	8

union 7 , 2



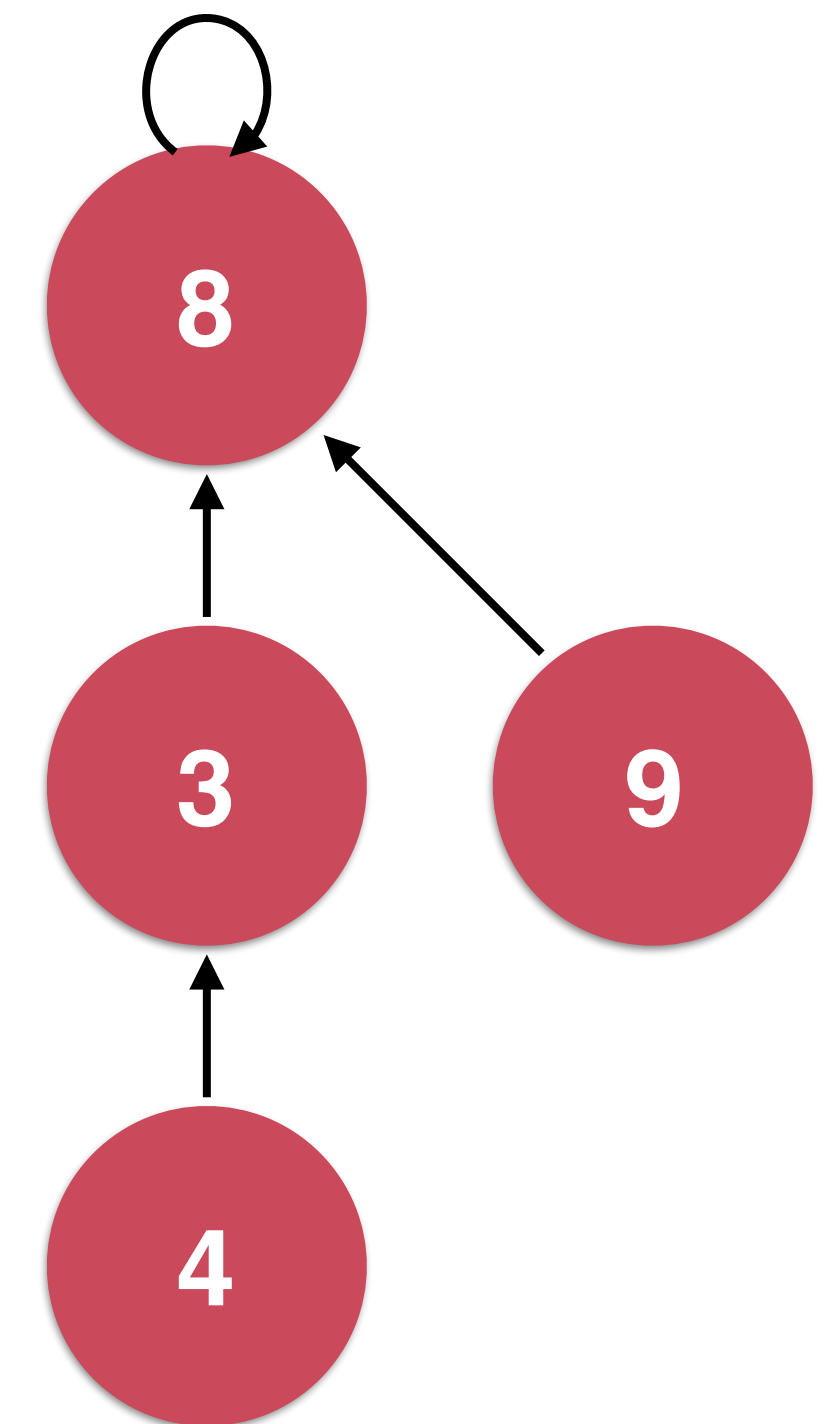
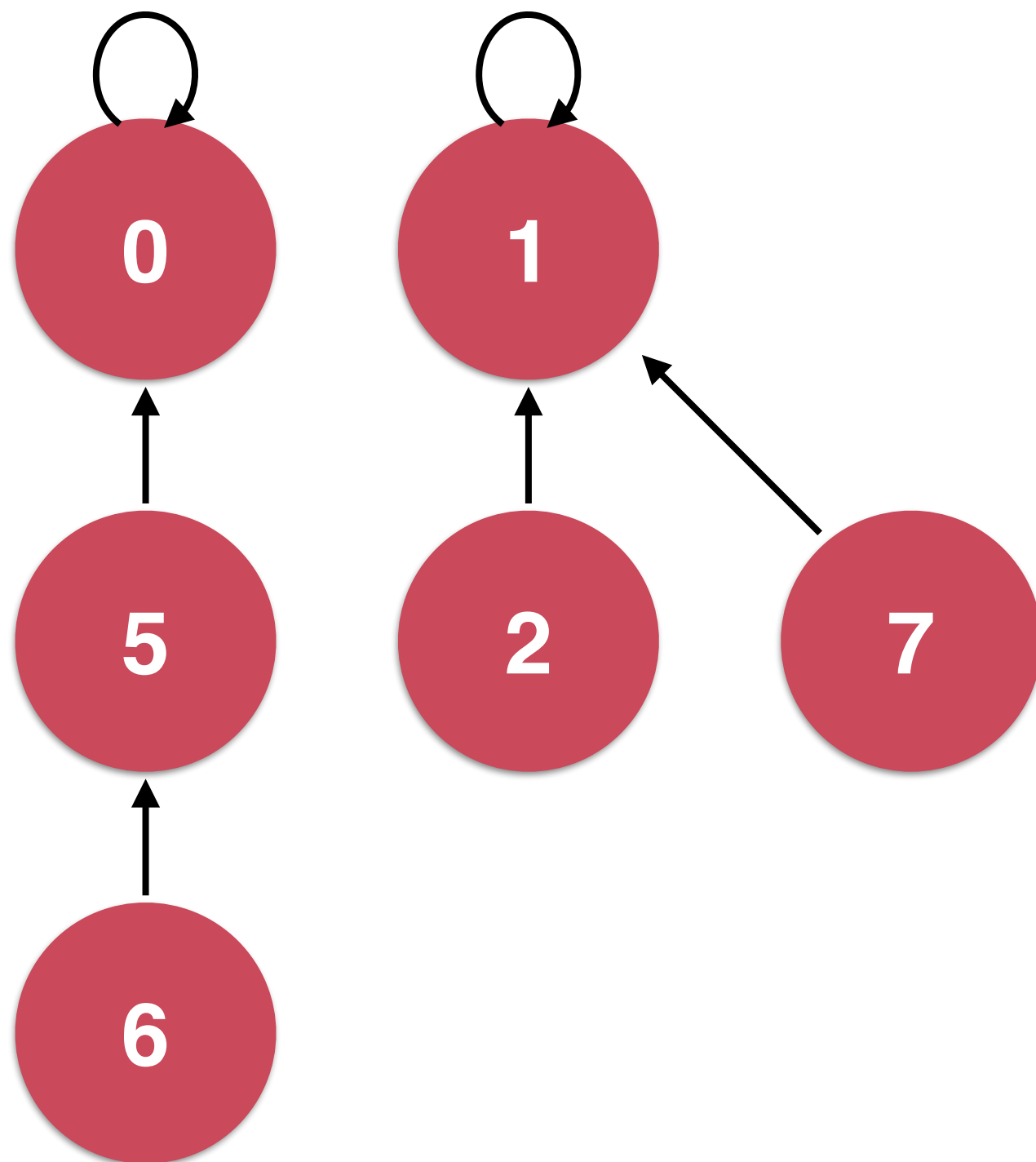
parent	0	1	2	3	4	5	6	7	8	9
	0	1	1	8	3	0	5	7	8	8

union 7 , 2



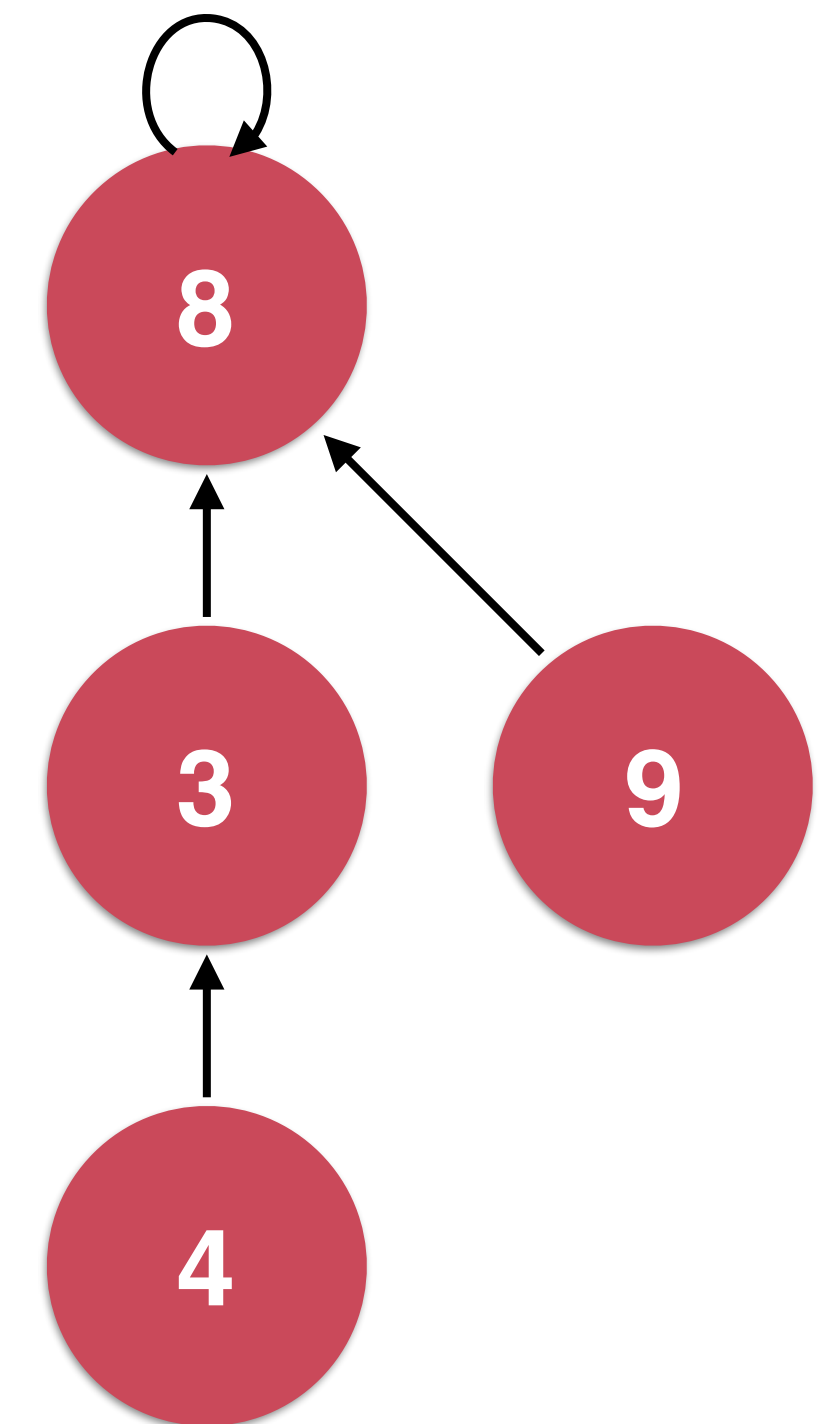
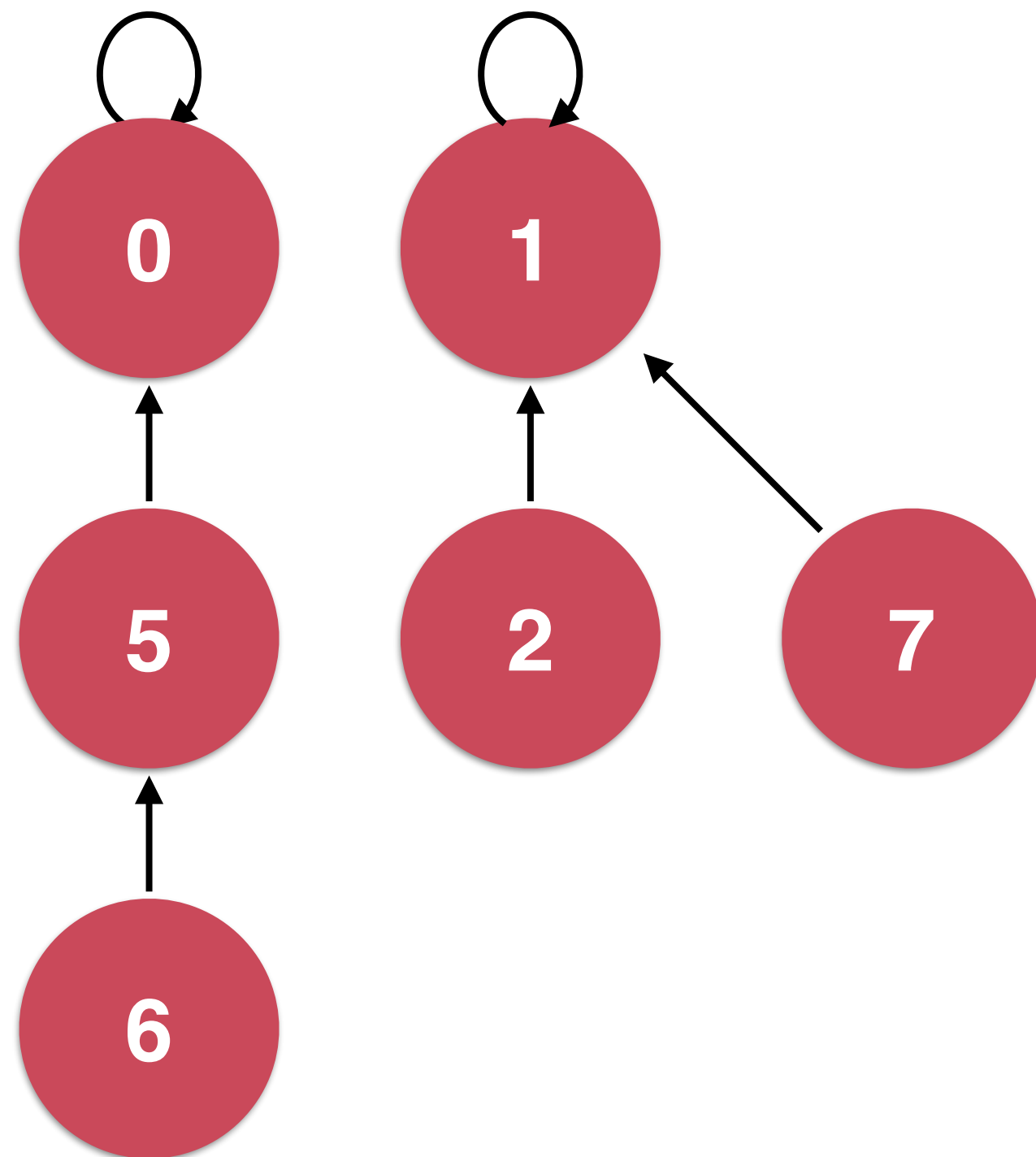
parent	0	1	2	3	4	5	6	7	8	9
	0	1	1	8	3	0	5	7	8	8

union 7 , 2



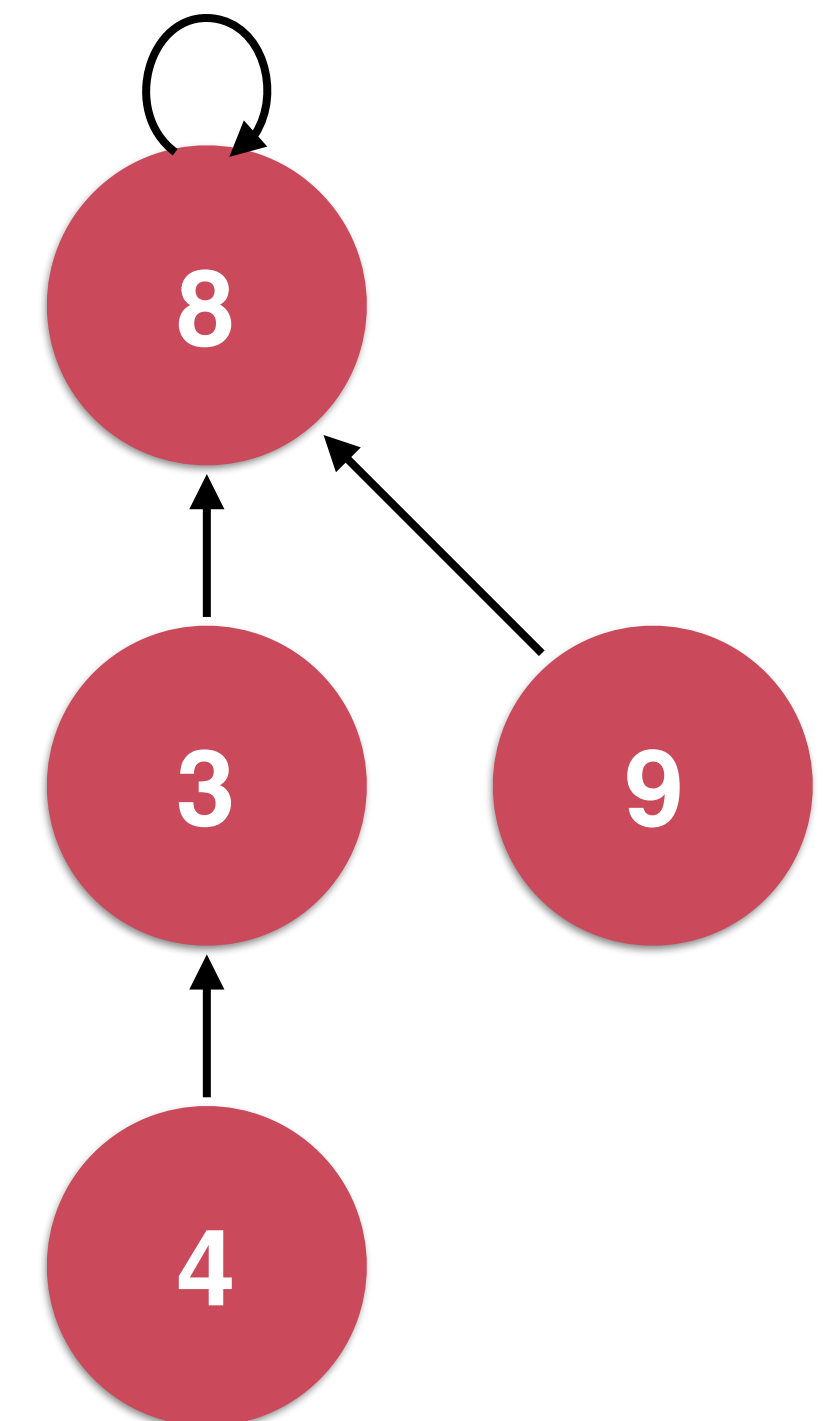
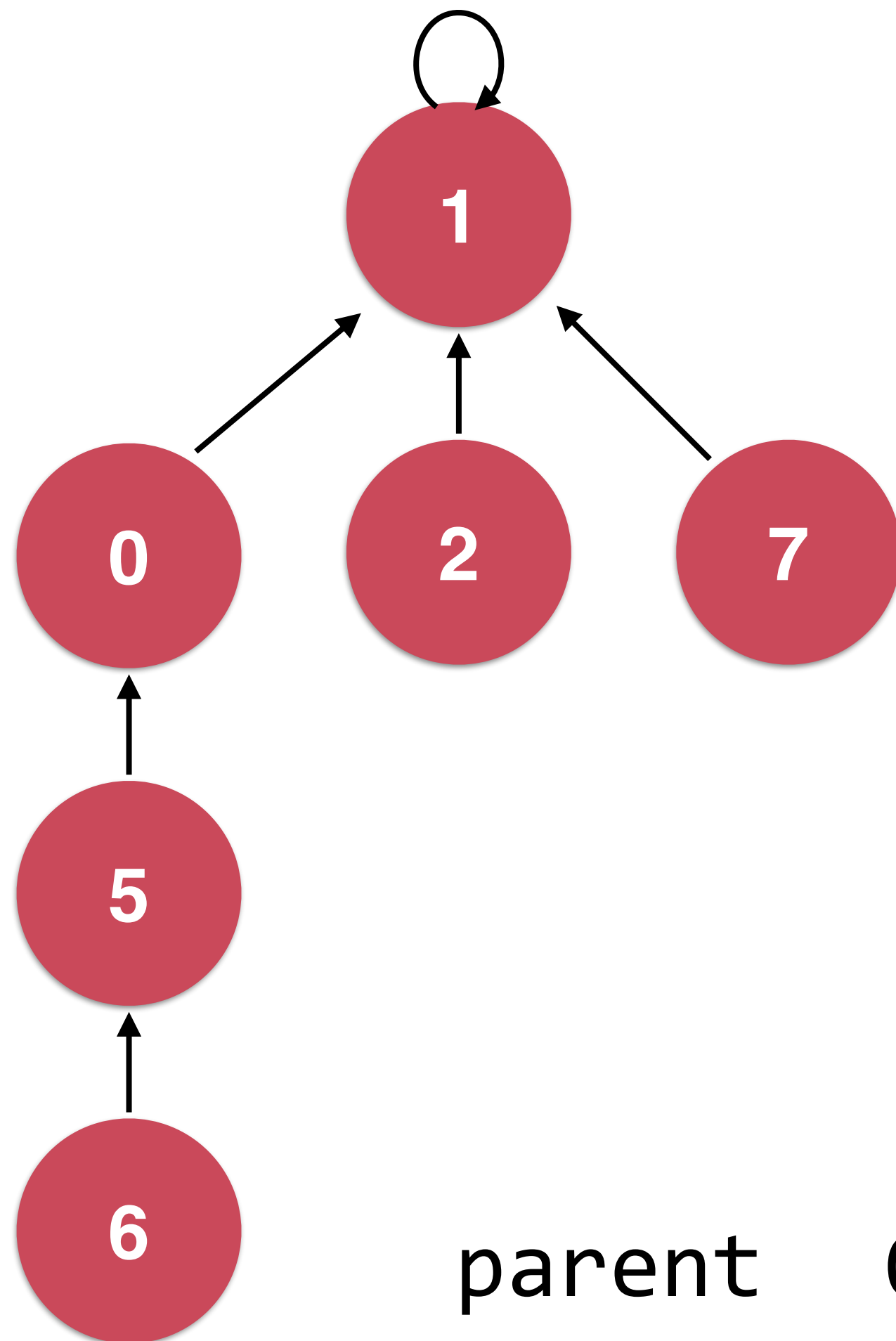
parent	0	1	2	3	4	5	6	7	8	9
	0	1	1	8	3	0	5	1	8	8

union 6 , 2



parent	0	1	2	3	4	5	6	7	8	9
	0	1	1	8	3	0	5	1	8	8

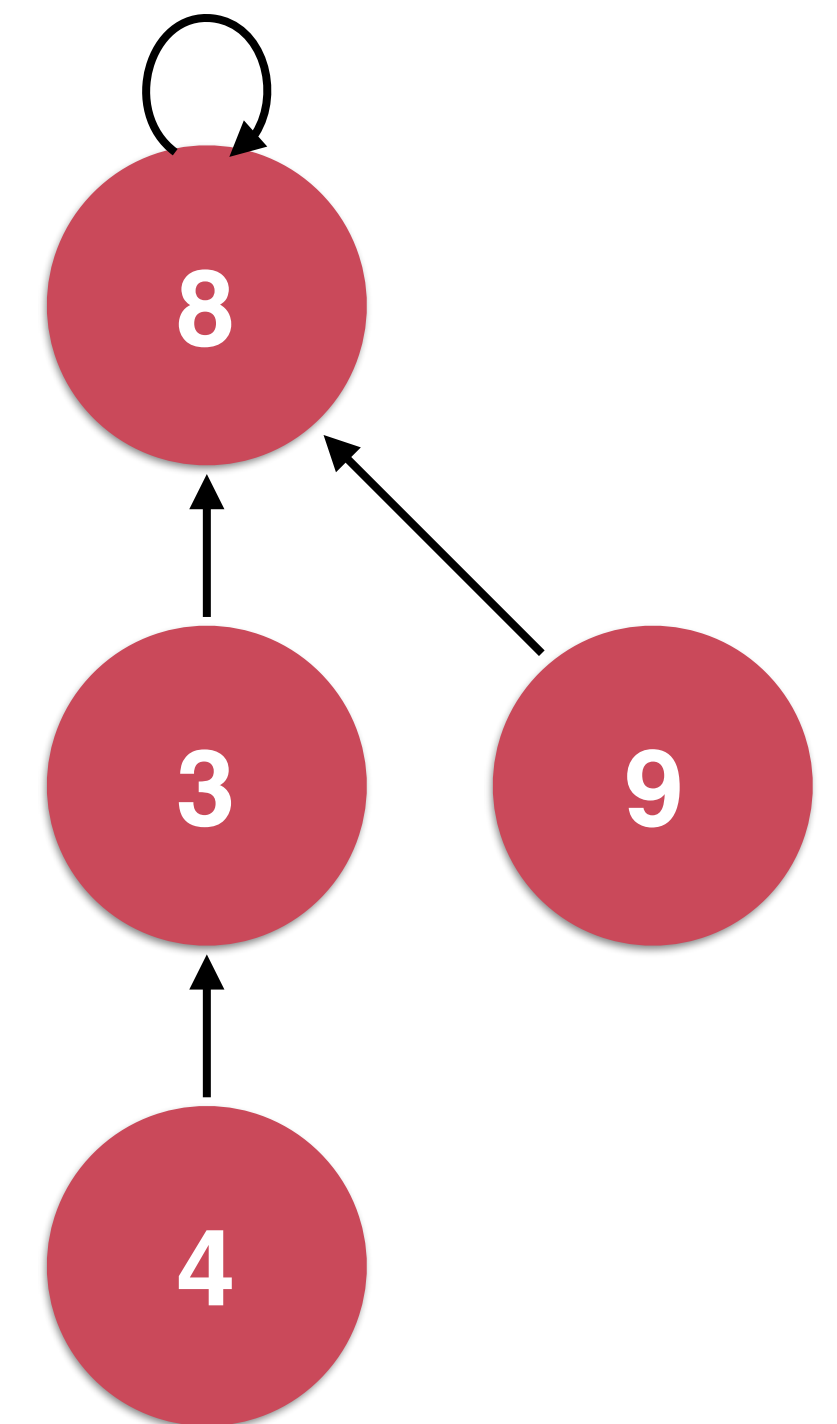
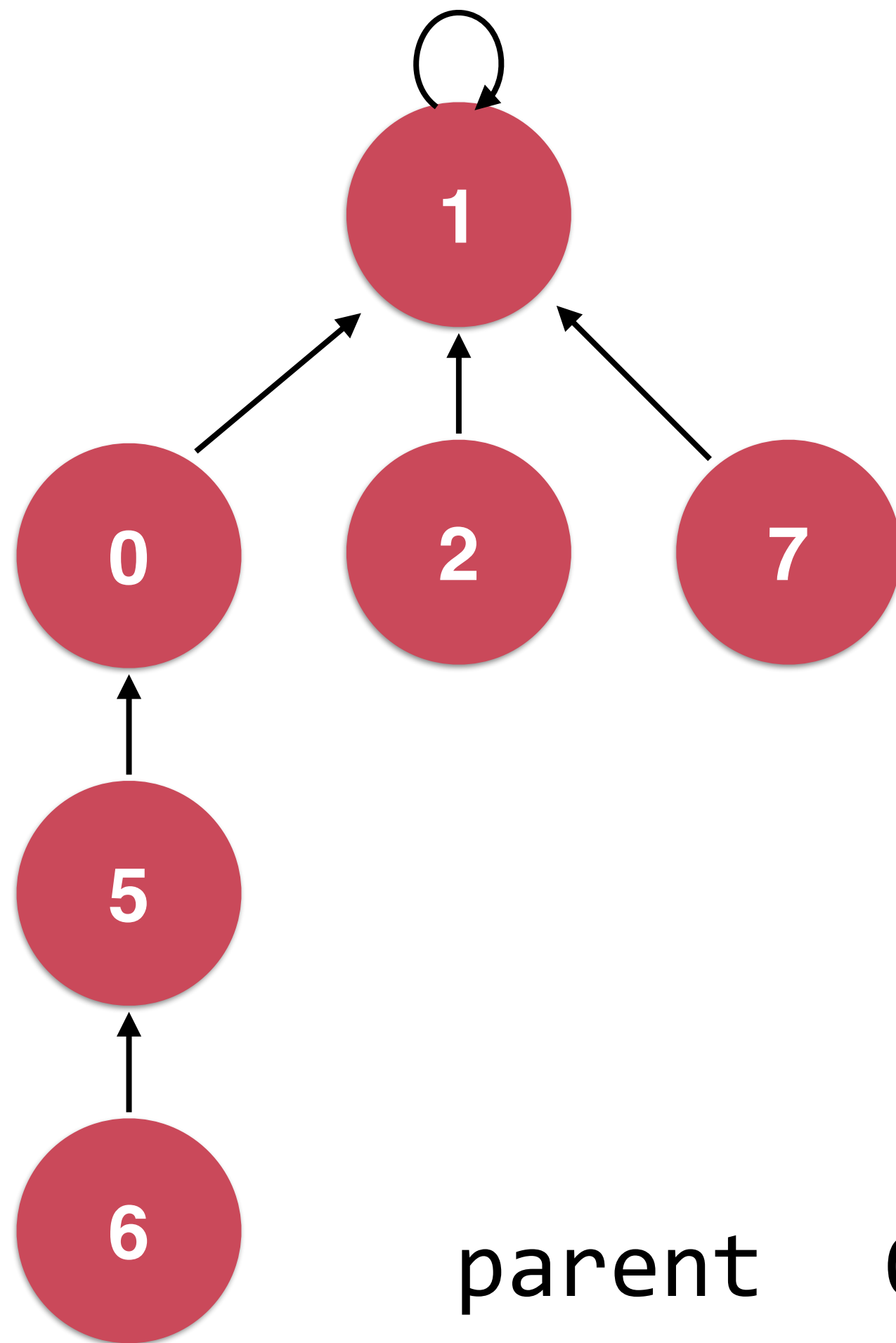
union 6 , 2



parent

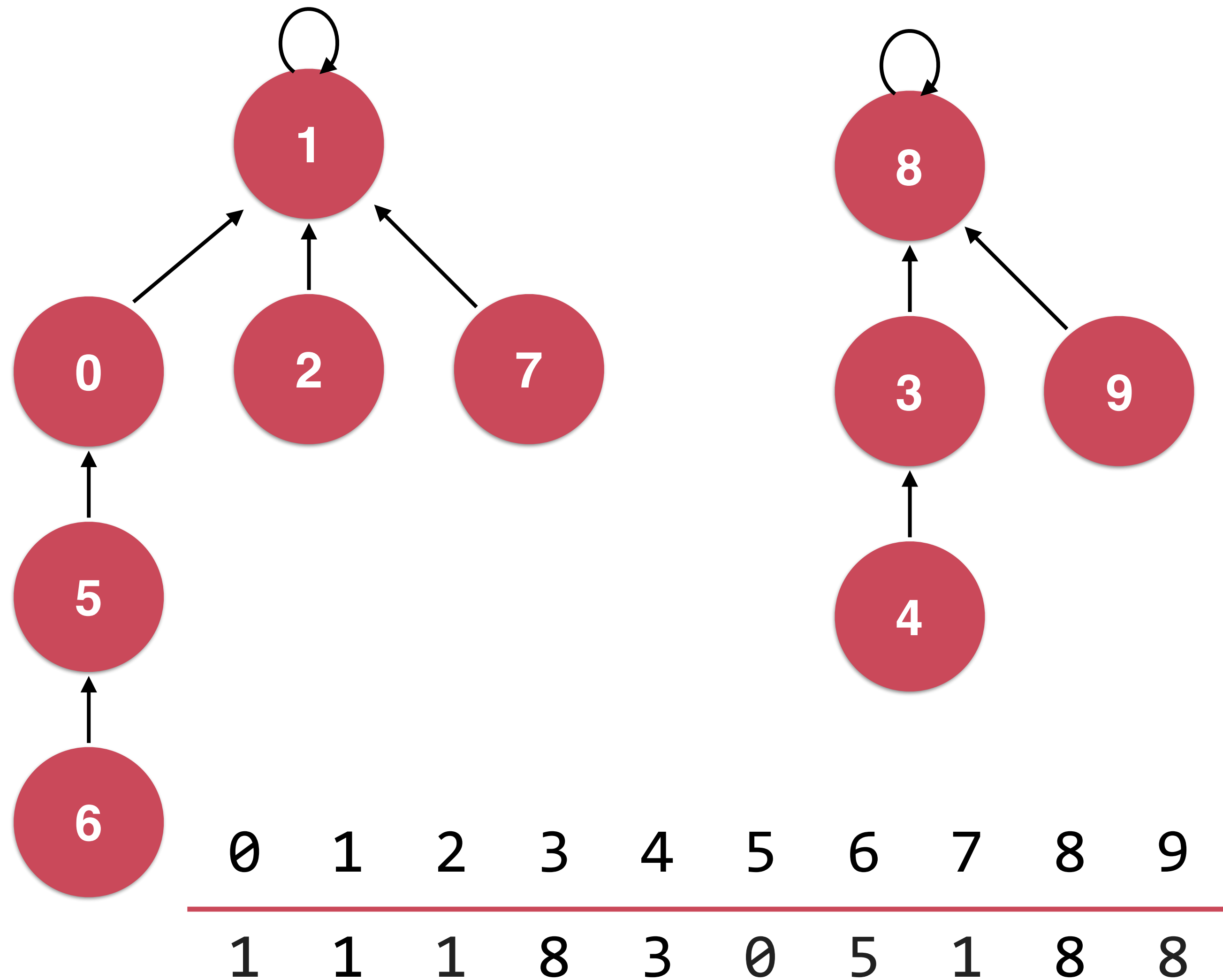
0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	1	8	8

union 6 , 2



parent	0	1	2	3	4	5	6	7	8	9
	1	1	1	8	3	0	5	1	8	8

并查集



实践：Quick Union 的并查集实现

并查集优化

实践：Quick Union 和 Quick Find 的时间效率比较

之前Quick Union的问题

Quick Union



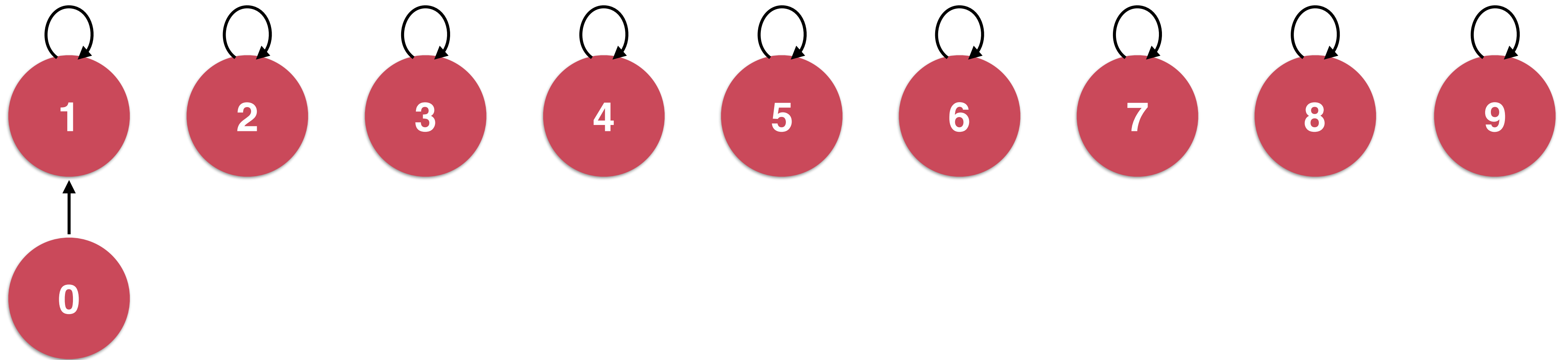
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

union 0 , 1



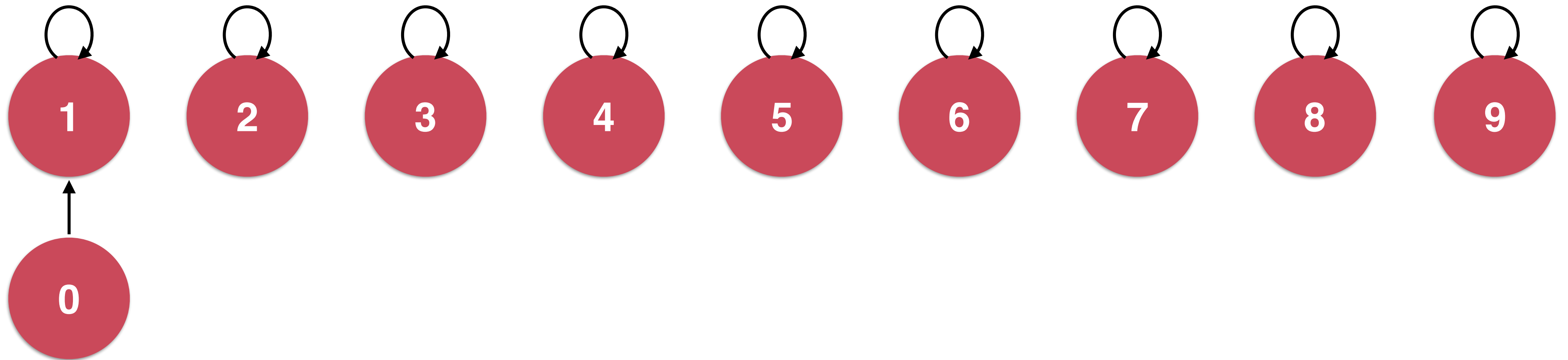
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

union 0 , 1



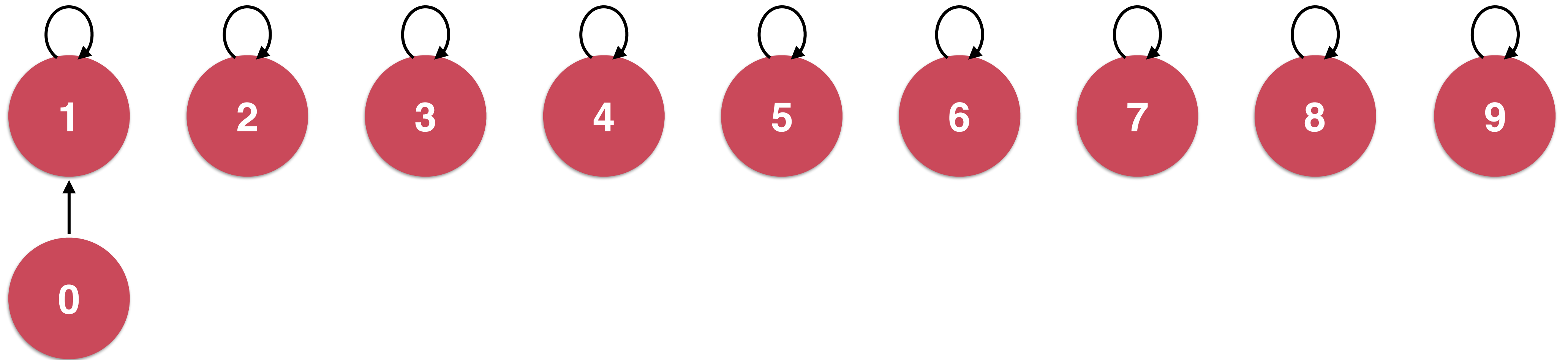
parent	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

union 0 , 1



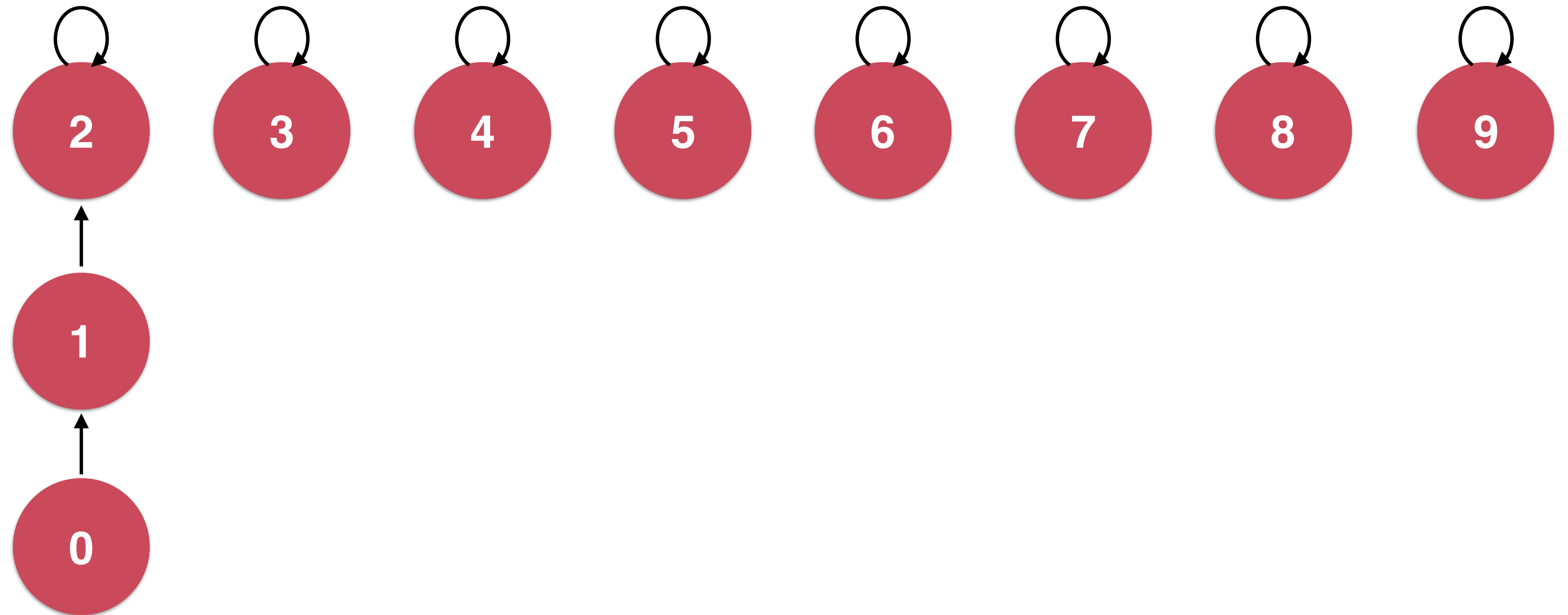
parent	0	1	2	3	4	5	6	7	8	9
	1	1	2	3	4	5	6	7	8	9

union 0 , 2



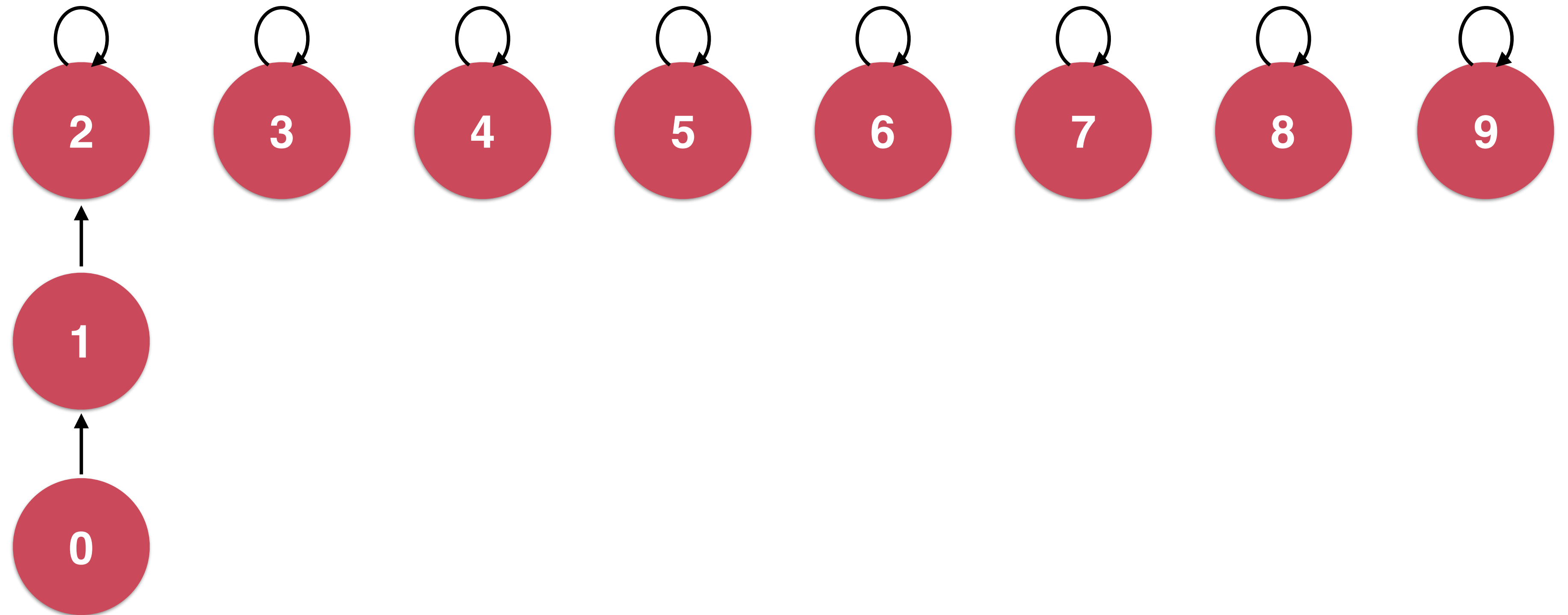
parent	0	1	2	3	4	5	6	7	8	9
	1	1	2	3	4	5	6	7	8	9

union 0 , 2



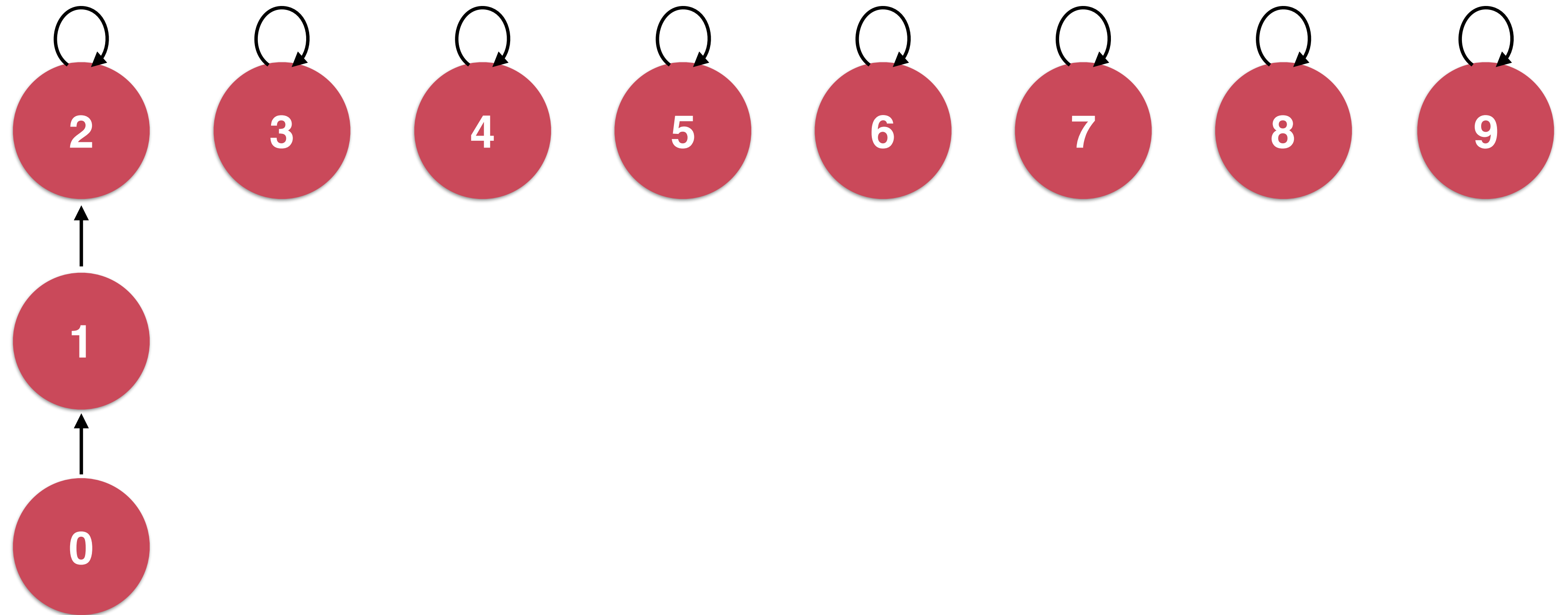
parent	0	1	2	3	4	5	6	7	8	9
	1	1	2	3	4	5	6	7	8	9

union 0 , 2



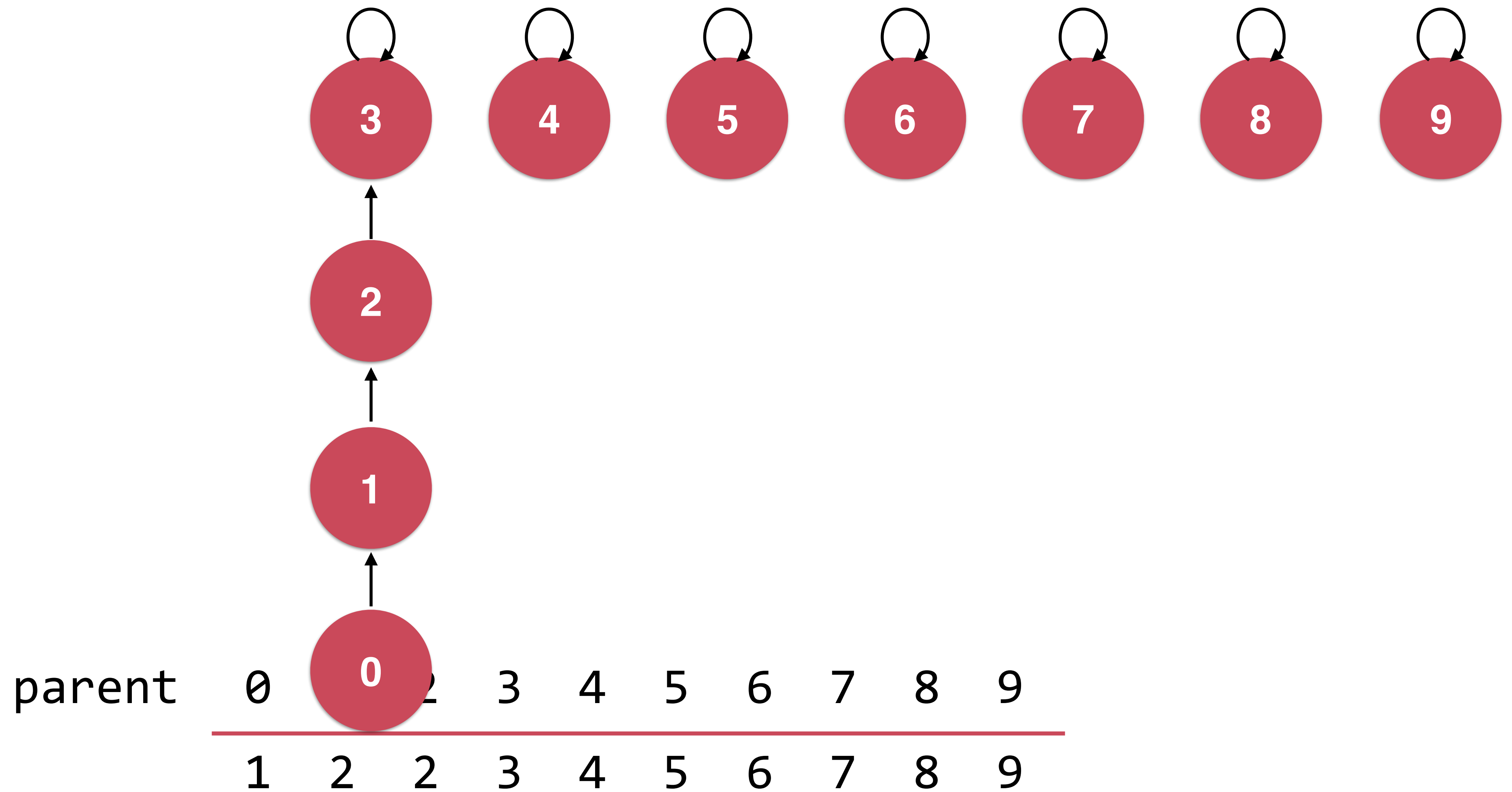
parent	0	1	2	3	4	5	6	7	8	9
	1	2	2	3	4	5	6	7	8	9

union 0 , 3

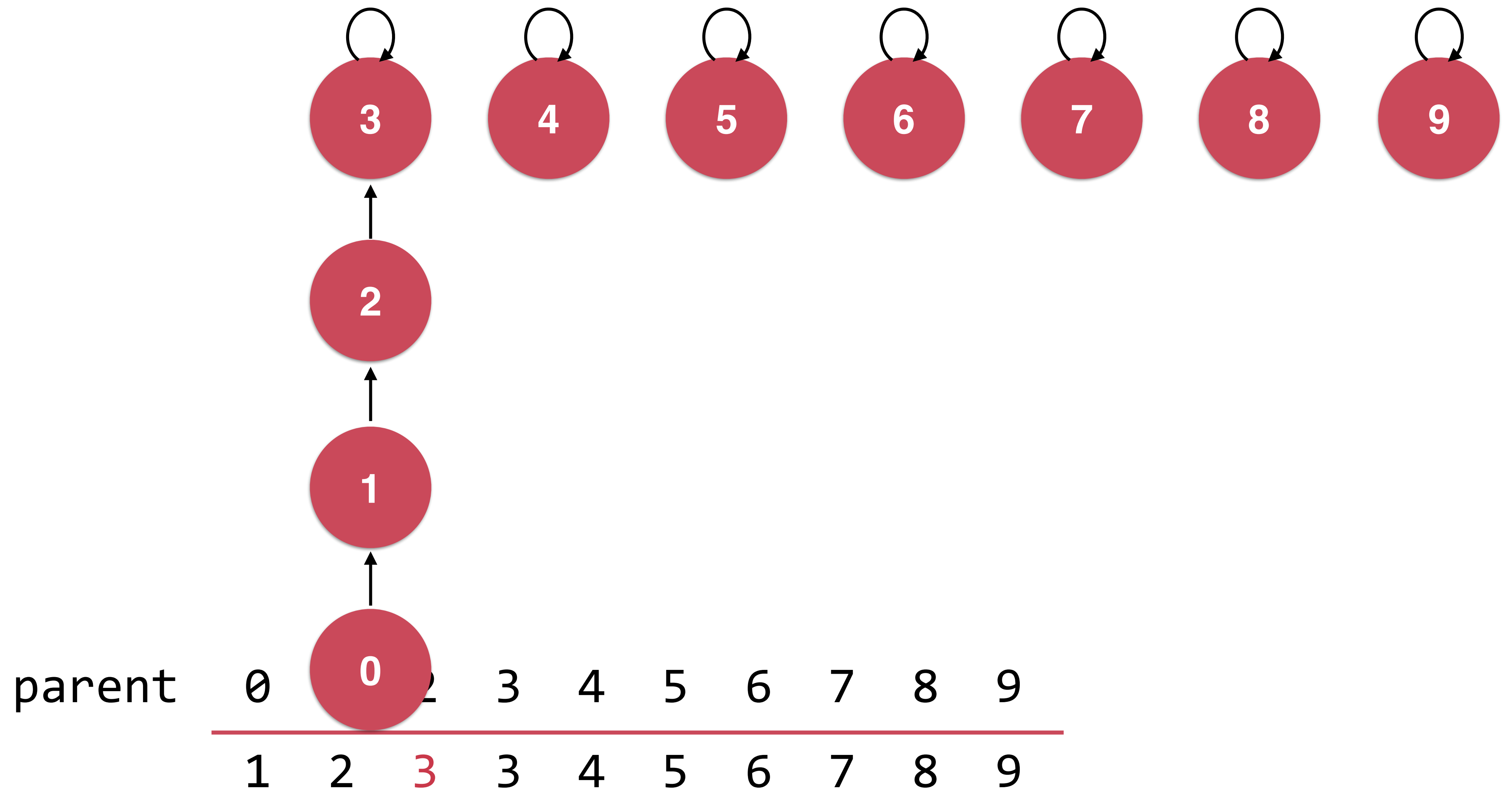


parent	0	1	2	3	4	5	6	7	8	9
	1	2	2	3	4	5	6	7	8	9

union 0 , 3

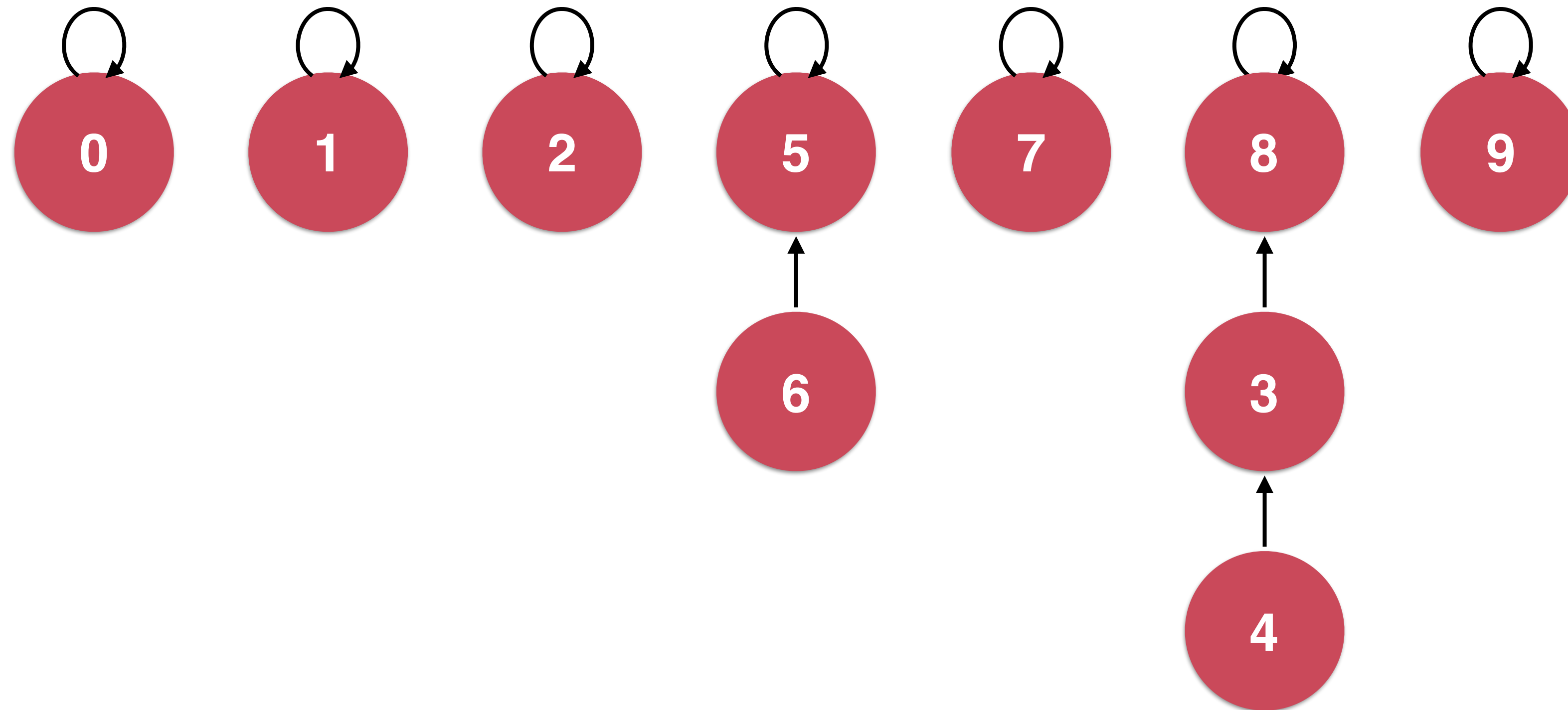


union 0 , 3



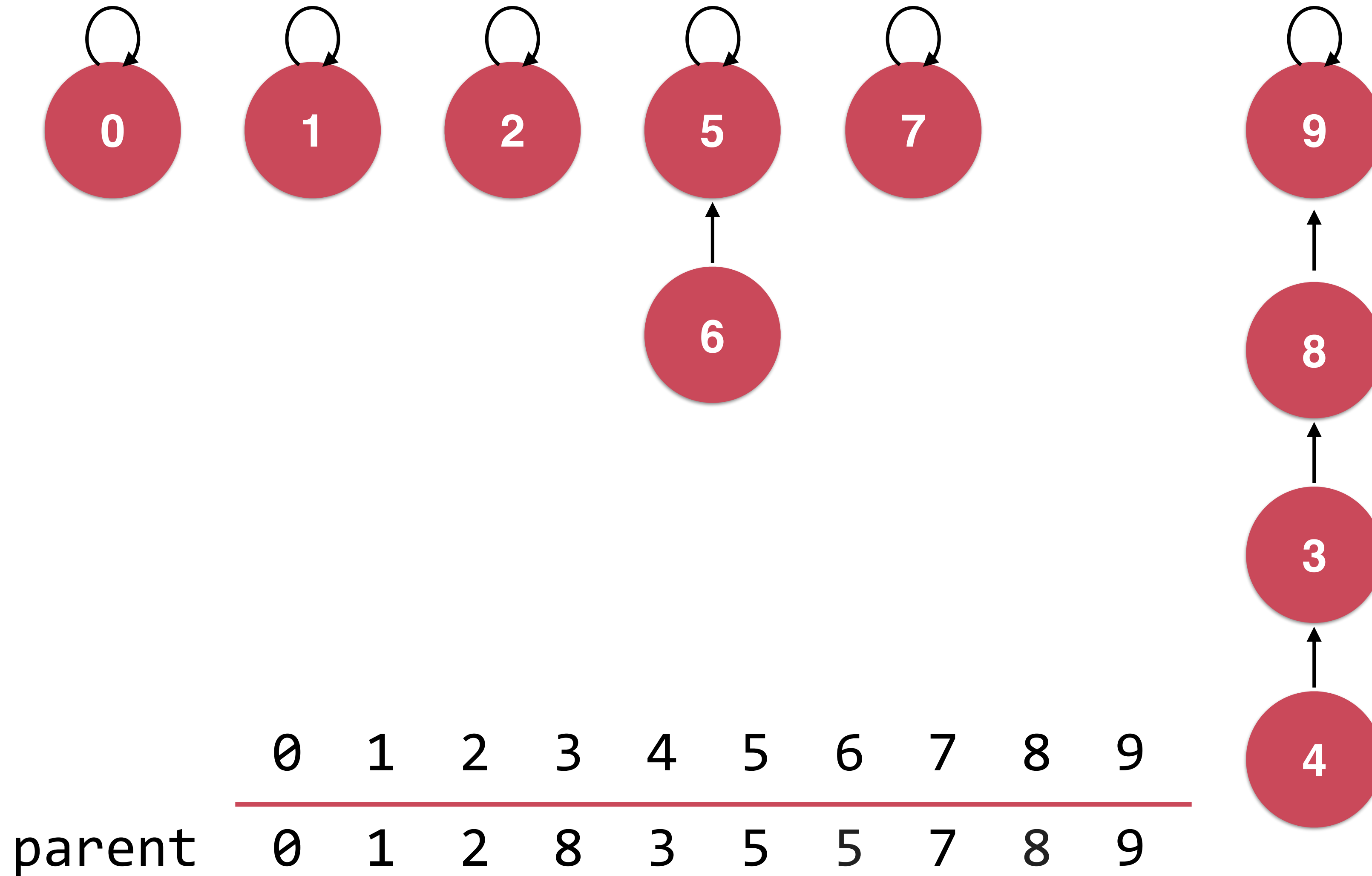
解决方案： 考虑size

union 4 , 9

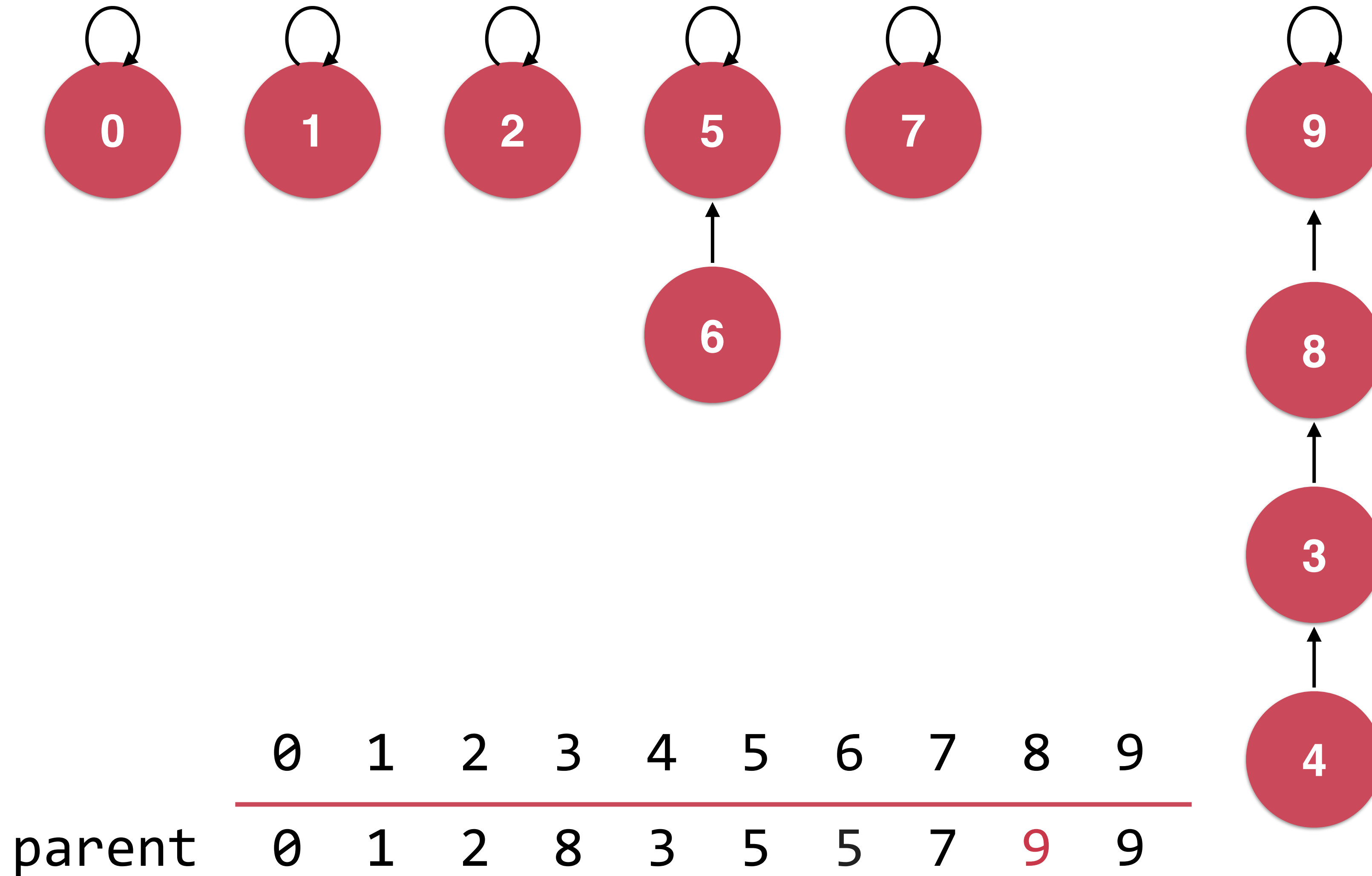


	0	1	2	3	4	5	6	7	8	9
parent	0	1	2	8	3	5	5	7	8	9

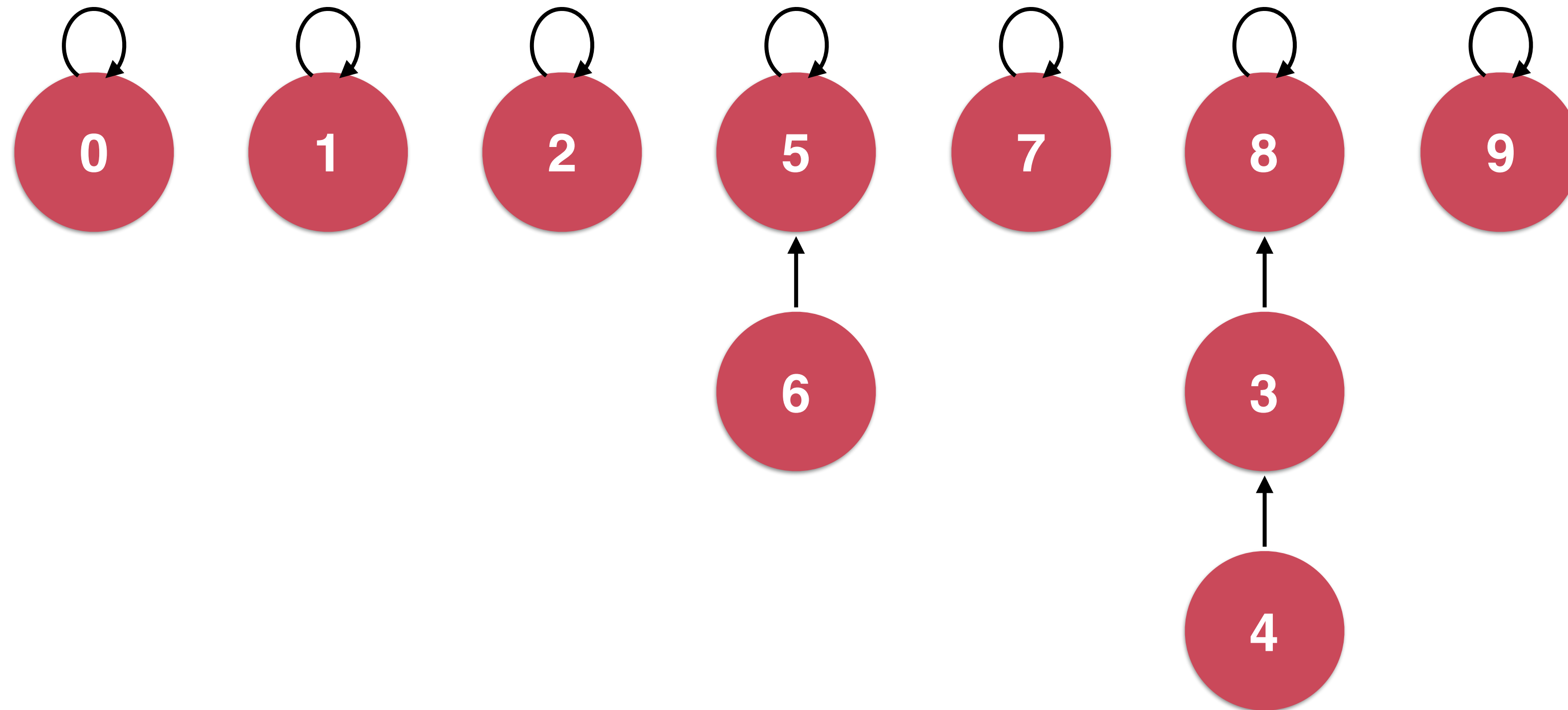
union 4 , 9



union 4 , 9

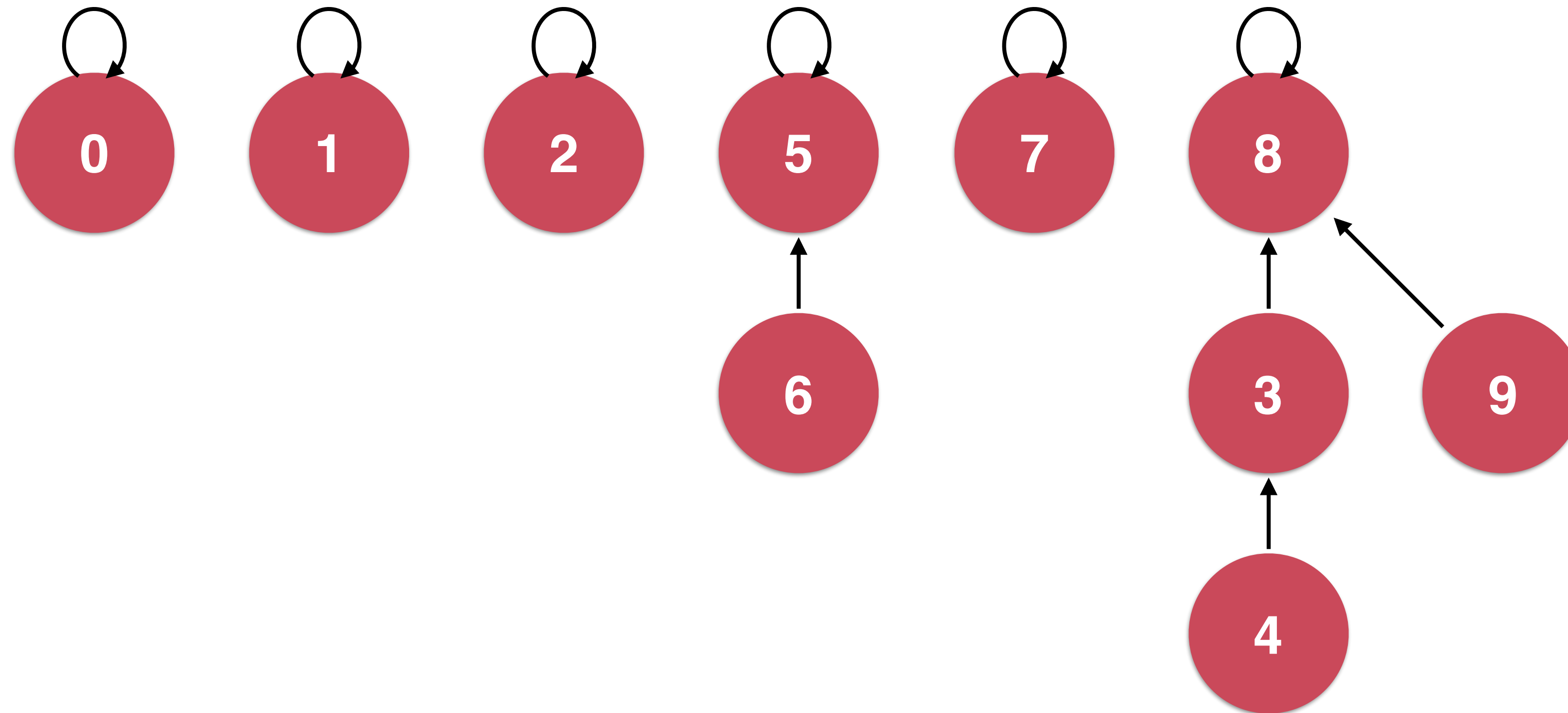


union 4 , 9



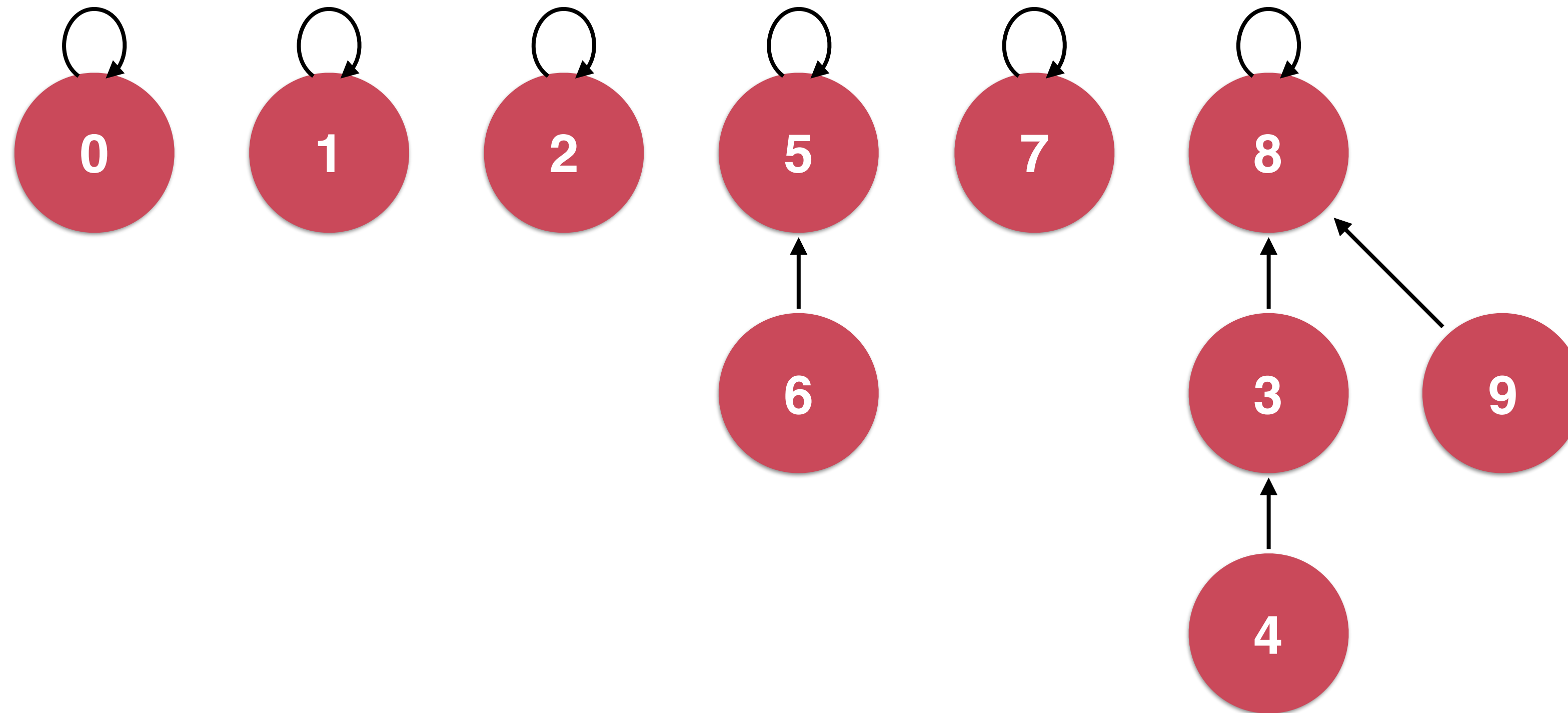
	0	1	2	3	4	5	6	7	8	9
parent	0	1	2	8	3	5	5	7	8	9

union 4 , 9



	0	1	2	3	4	5	6	7	8	9
parent	0	1	2	8	3	5	5	7	8	9

union 4 , 9



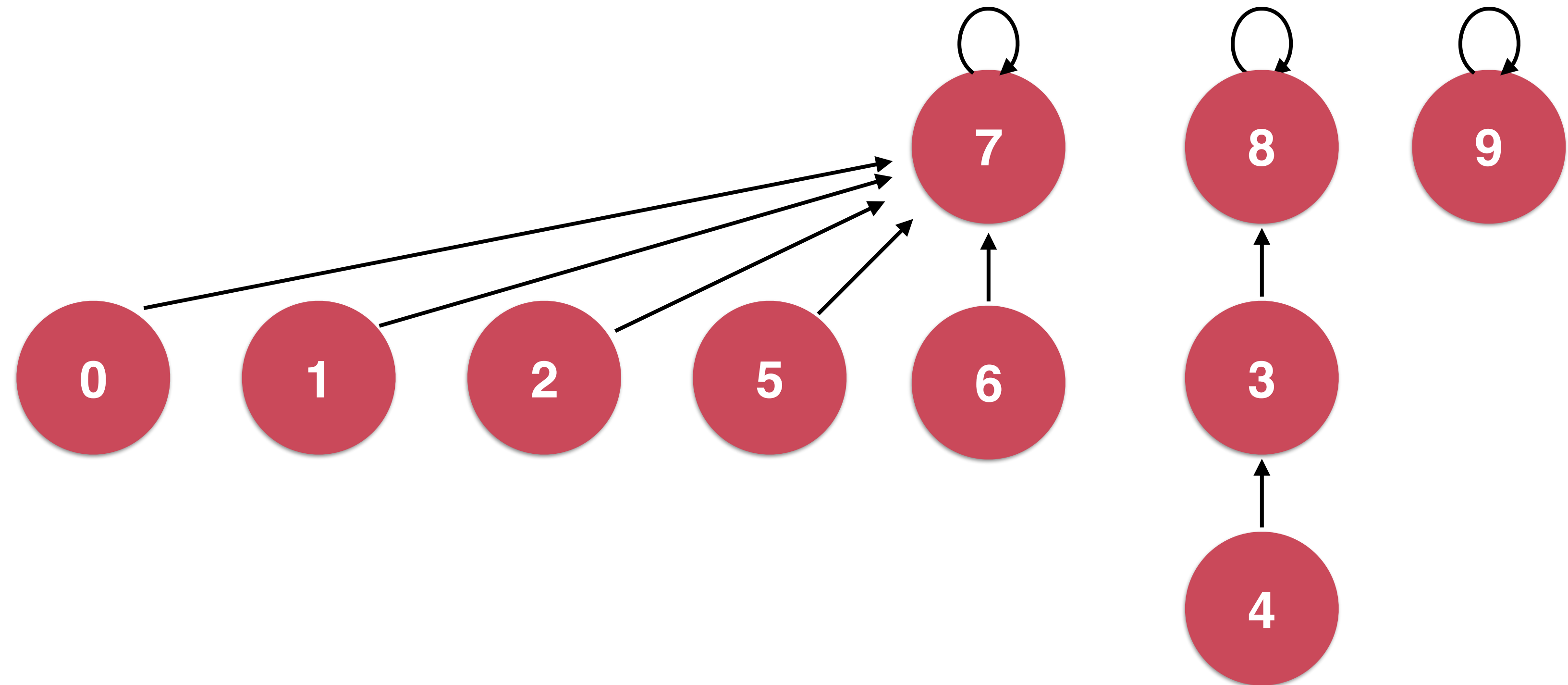
	0	1	2	3	4	5	6	7	8	9
parent	0	1	2	8	3	5	5	7	8	8

实践： 基于size的优化

实践： Quick Union 和优化后的时间效率比较

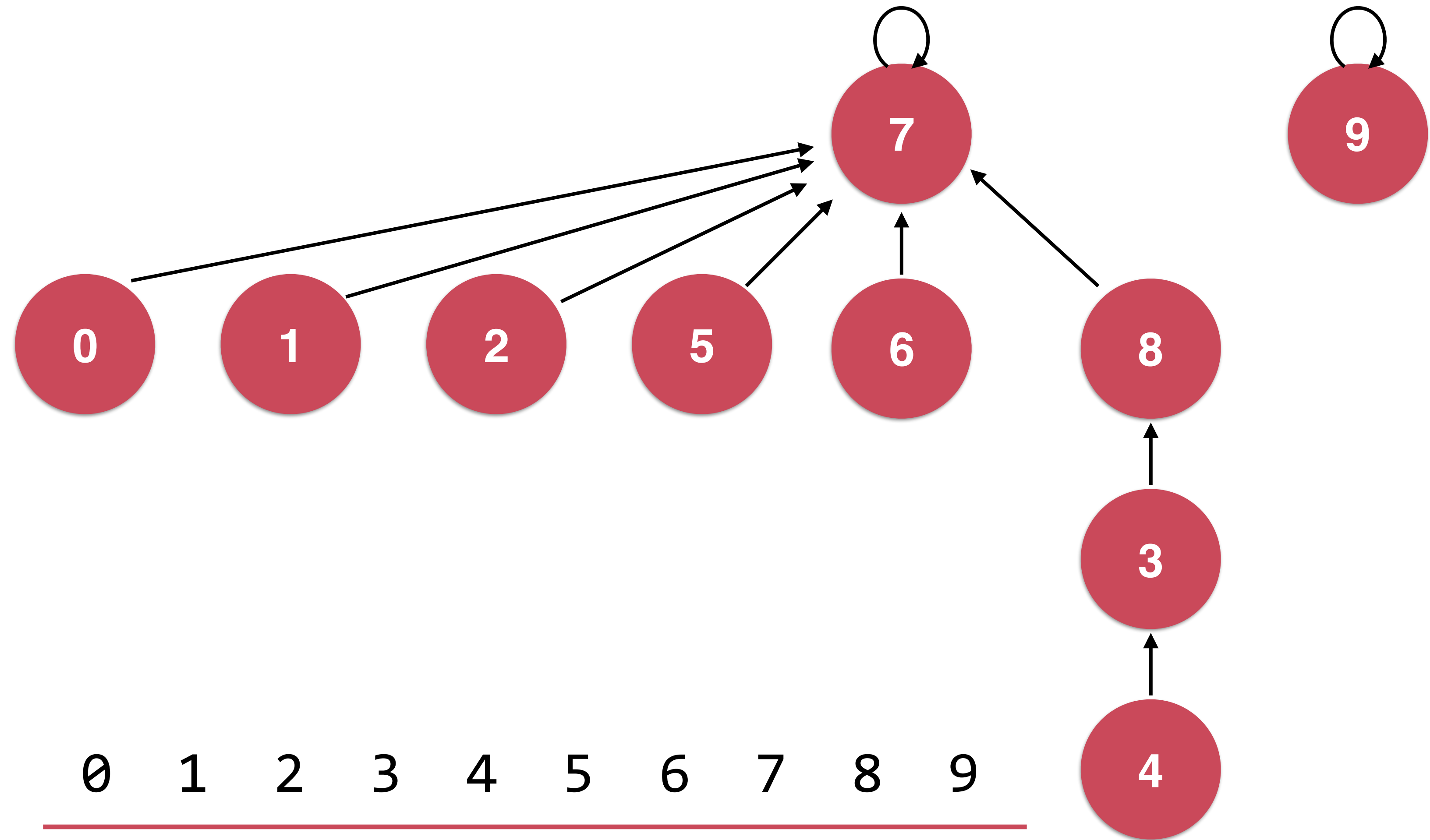
基于rank的优化

union 4 , 2



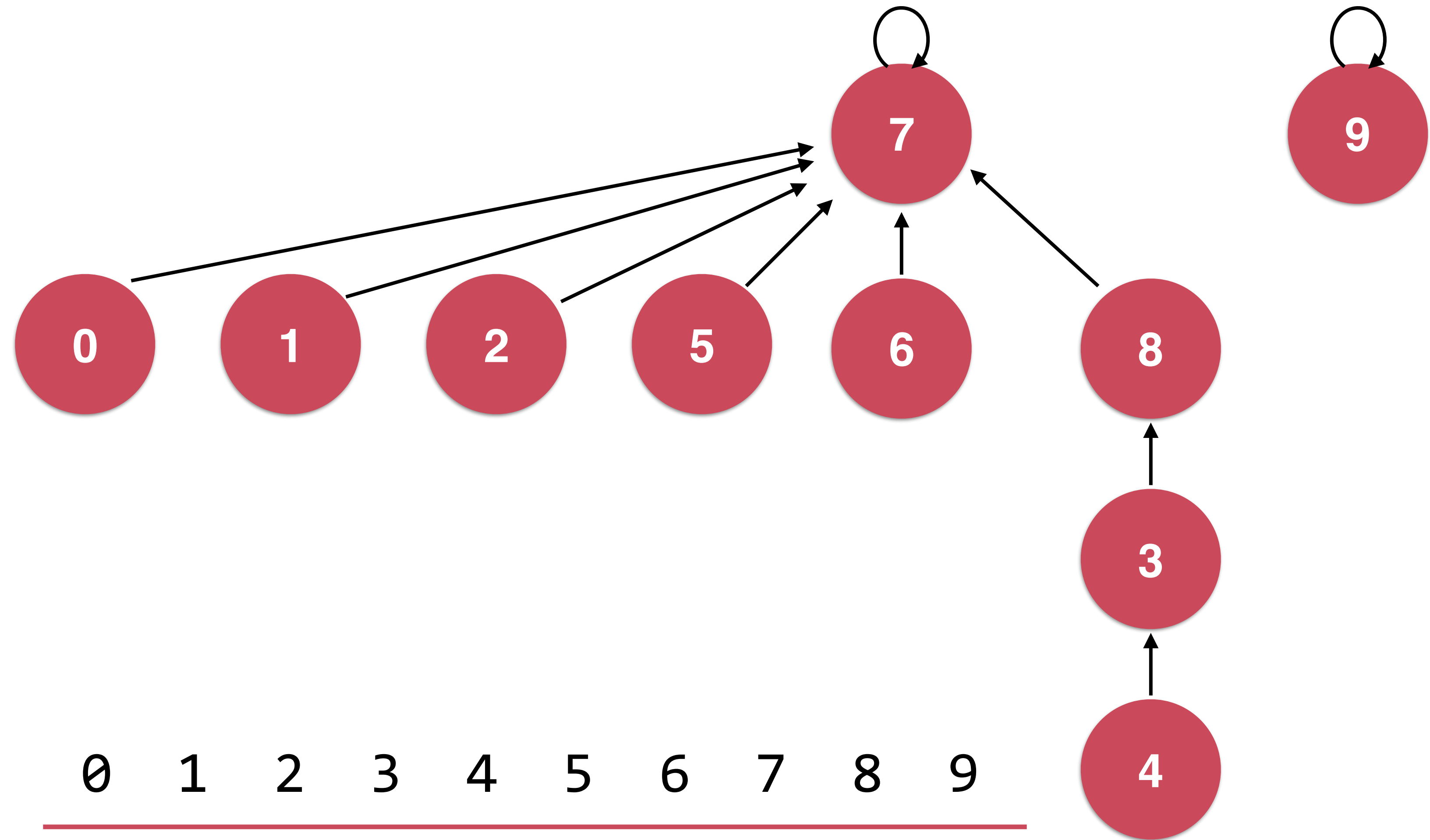
	0	1	2	3	4	5	6	7	8	9
parent	7	7	7	8	3	7	7	7	8	9

union 4 , 2



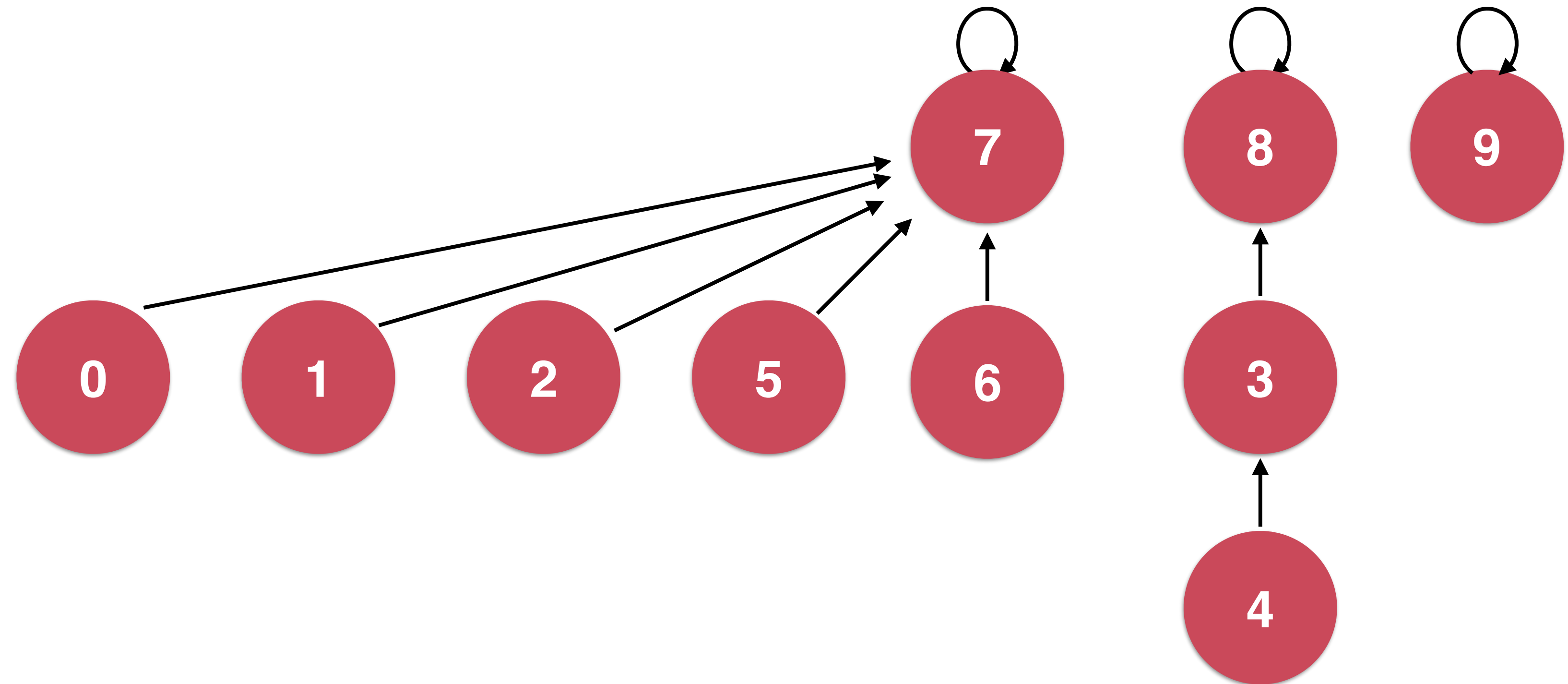
	0	1	2	3	4	5	6	7	8	9
parent	7	7	7	8	3	7	7	7	8	9

union 4 , 2



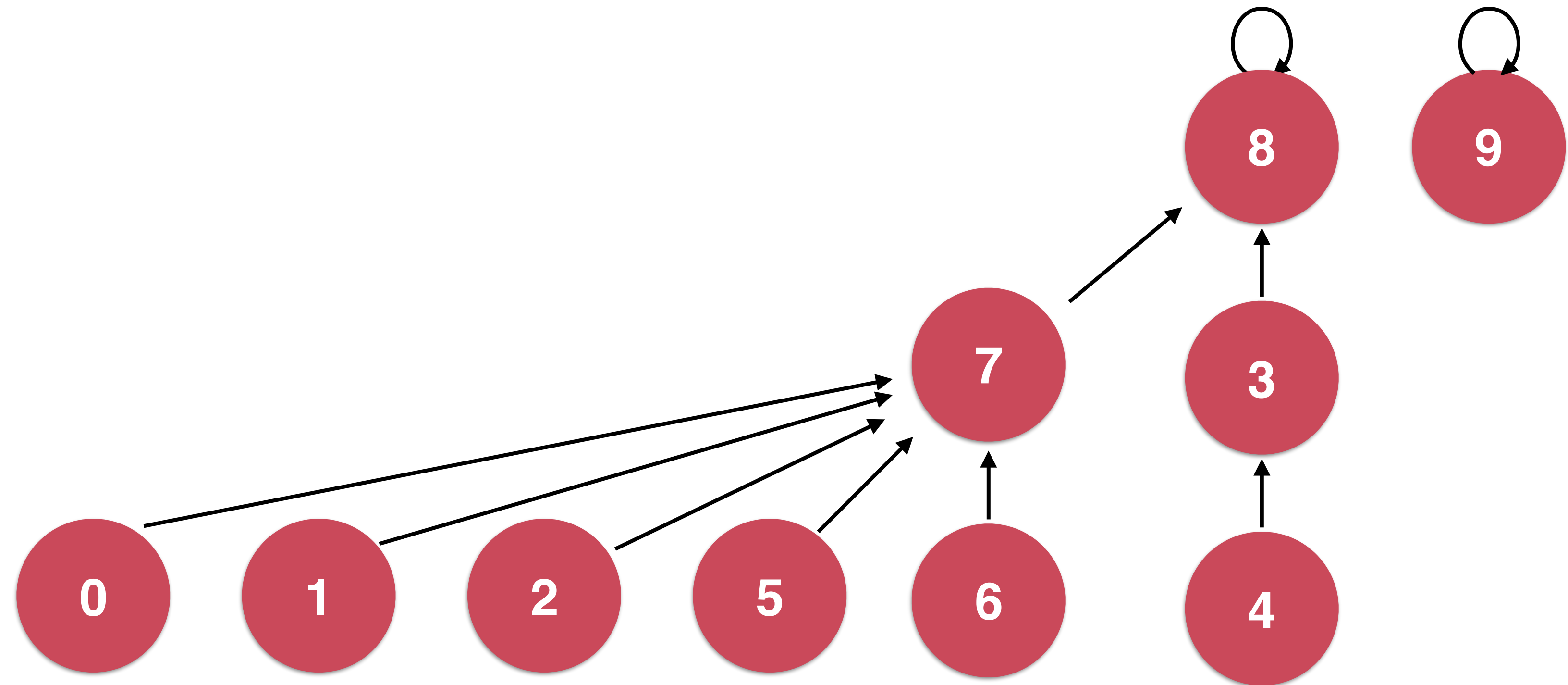
	0	1	2	3	4	5	6	7	8	9
parent	7	7	7	8	3	7	7	7	7	9

union 4 , 2



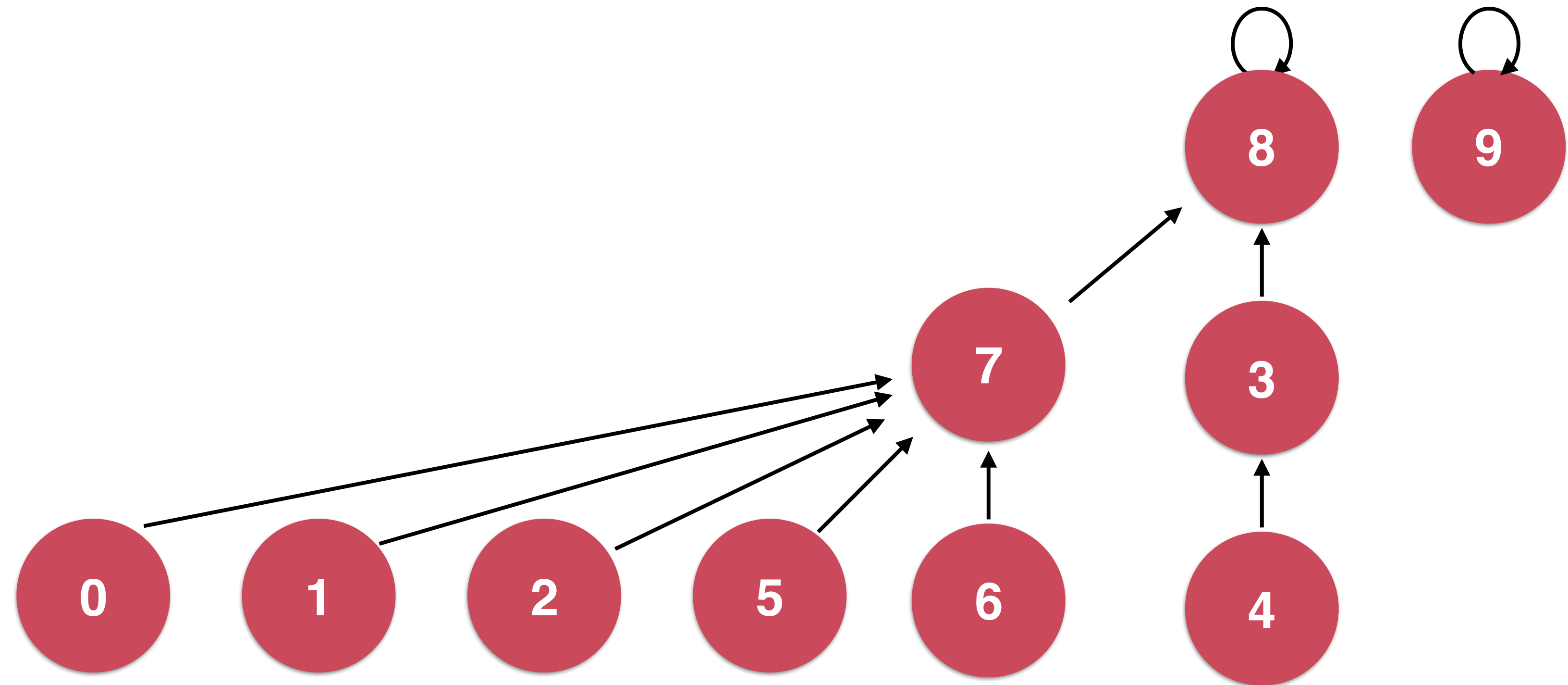
	0	1	2	3	4	5	6	7	8	9
parent	7	7	7	8	3	7	7	7	8	9

union 4 , 2



	0	1	2	3	4	5	6	7	8	9
parent	7	7	7	8	3	7	7	7	8	9

union 4 , 2



	0	1	2	3	4	5	6	7	8	9
parent	7	7	7	8	3	7	7	8	8	9

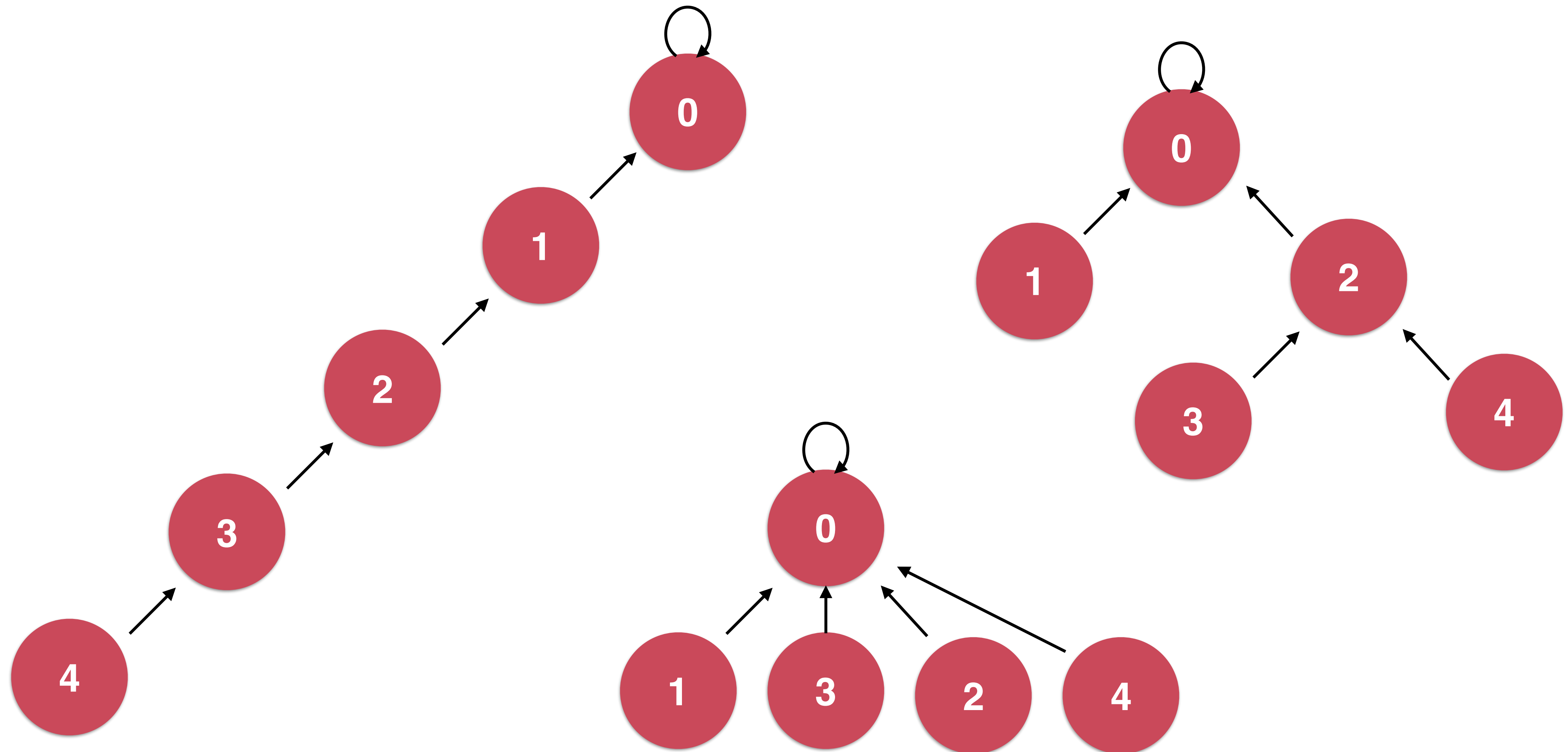
基于rank的优化

$\text{rank}[i]$ 表示根节点为 i 的树的高度

实践： 基于rank的优化

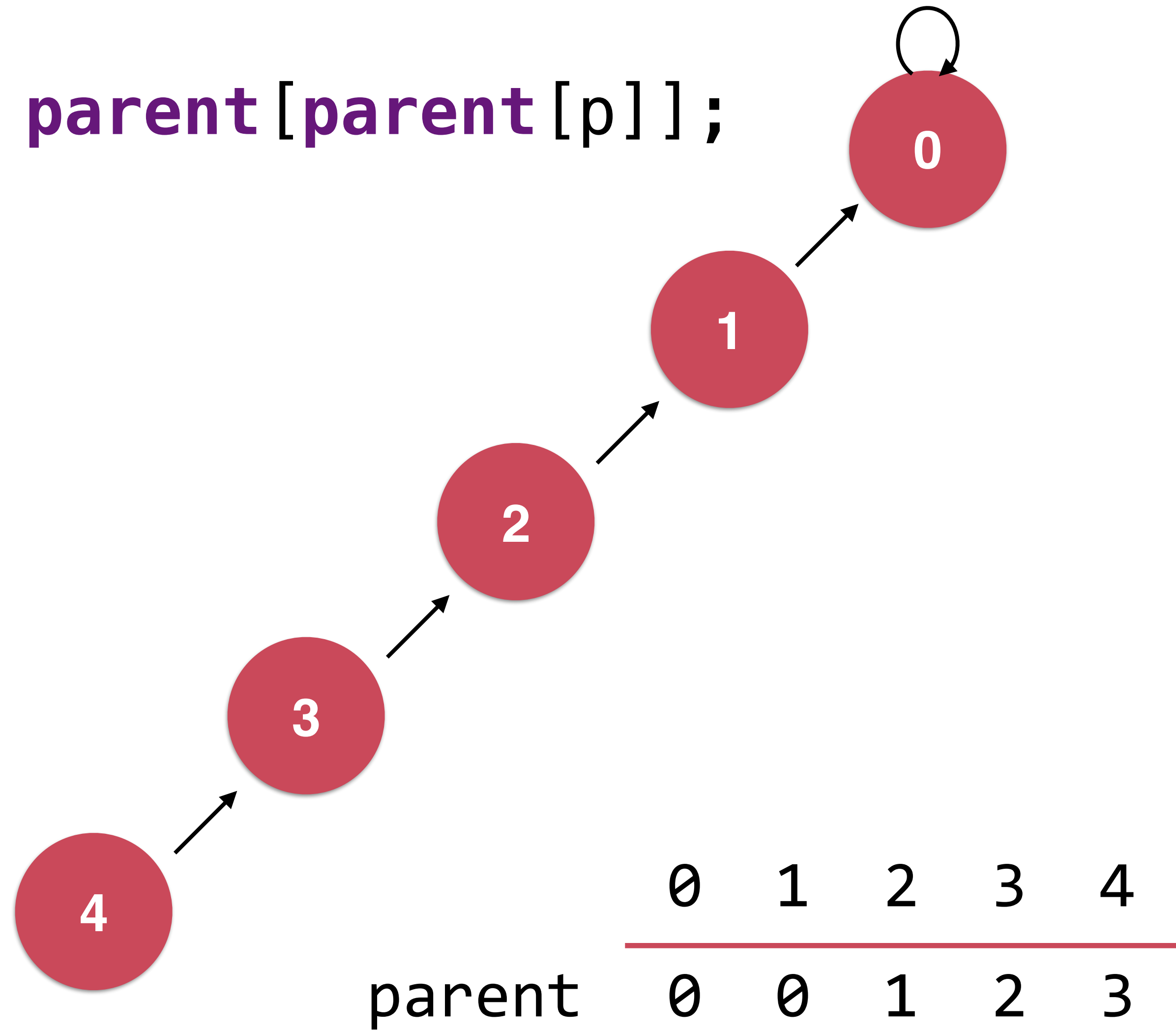
路径压缩 Path Compression

路径压缩



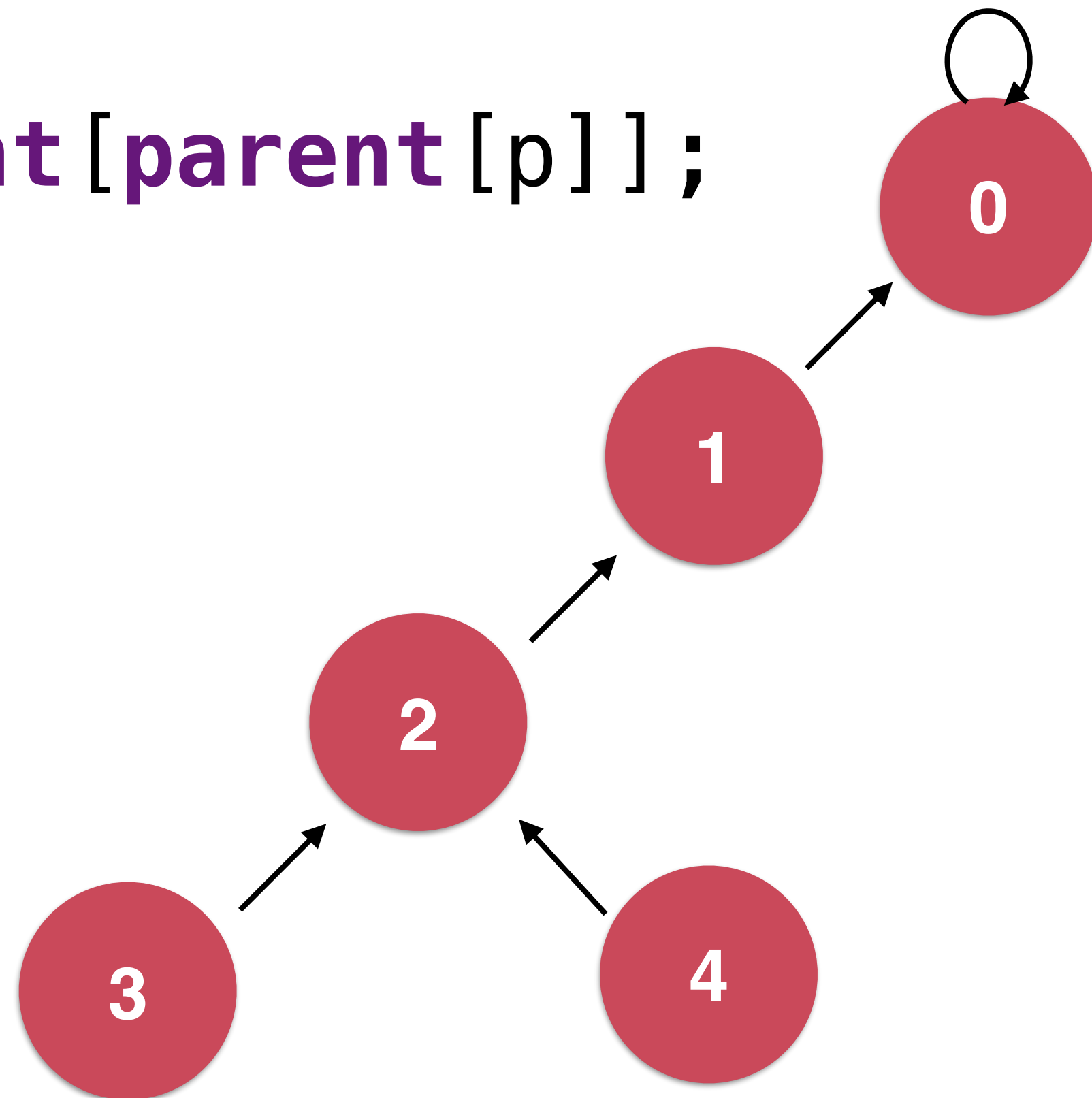
find 4

`parent[p] = parent[parent[p]];`



find 4

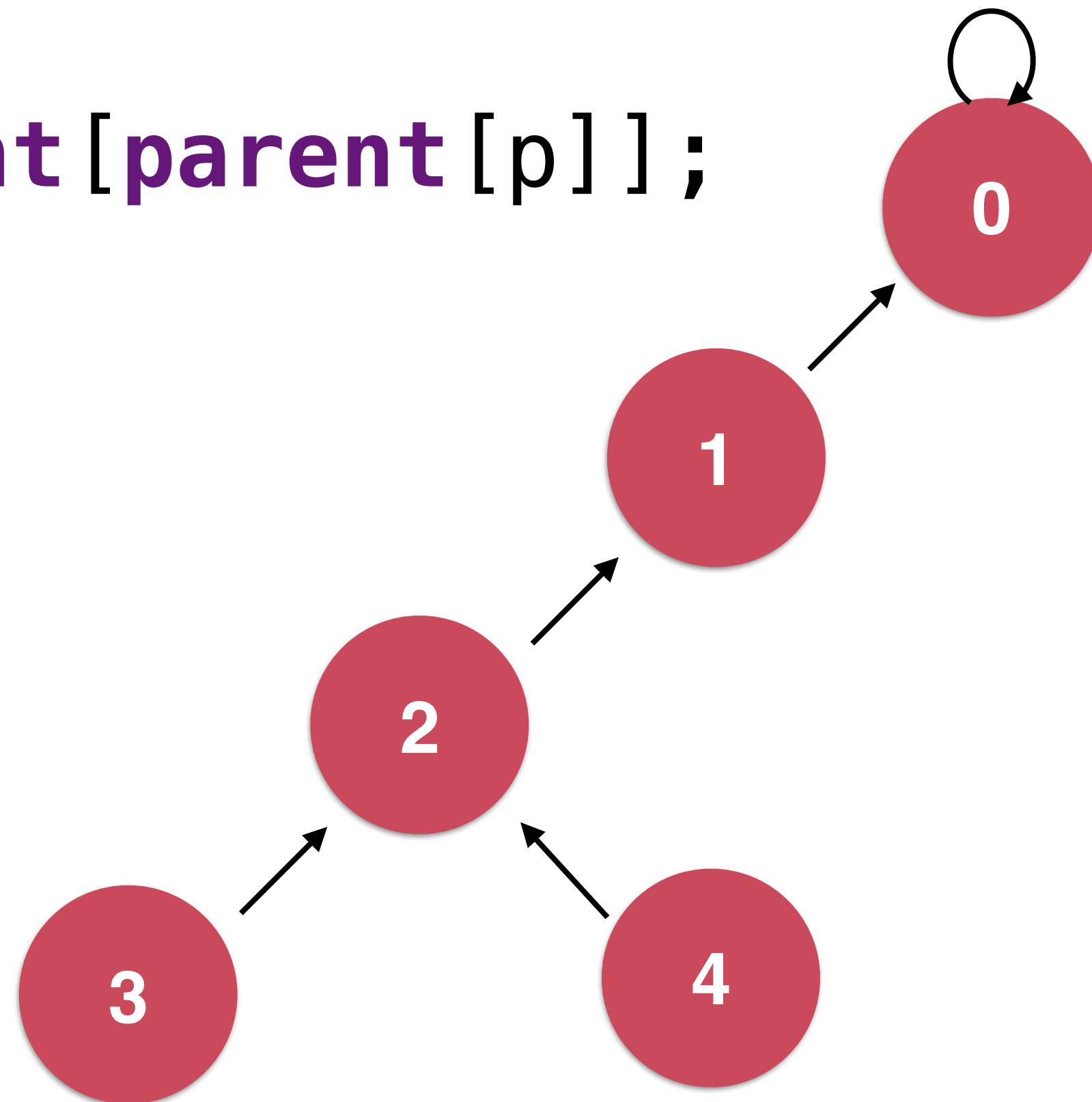
`parent[p] = parent[parent[p]];`



	0	1	2	3	4
parent	0	0	1	2	3

find 4

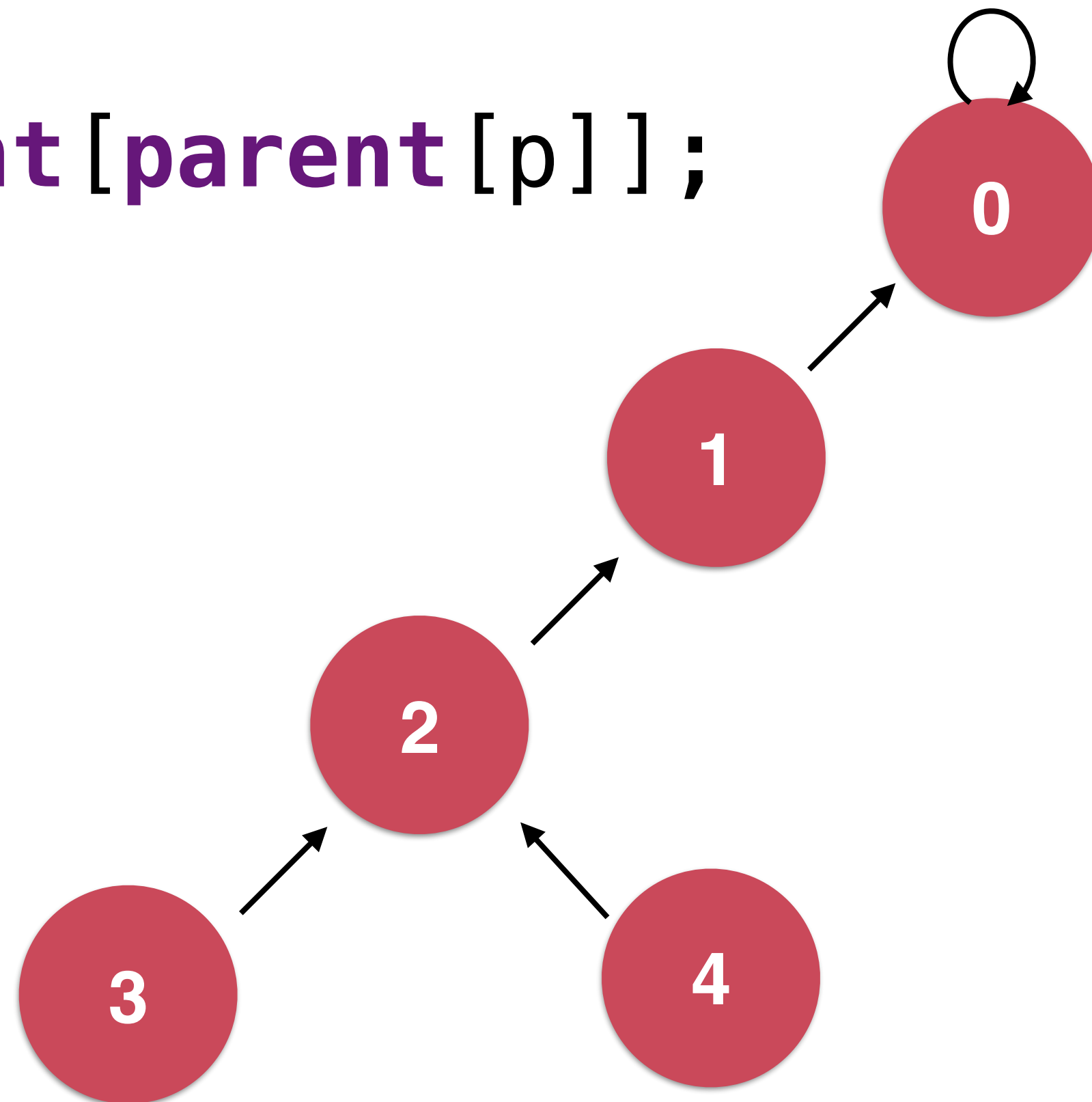
`parent[p] = parent[parent[p]];`



	0	1	2	3	4
parent	0	0	1	2	2

find 4

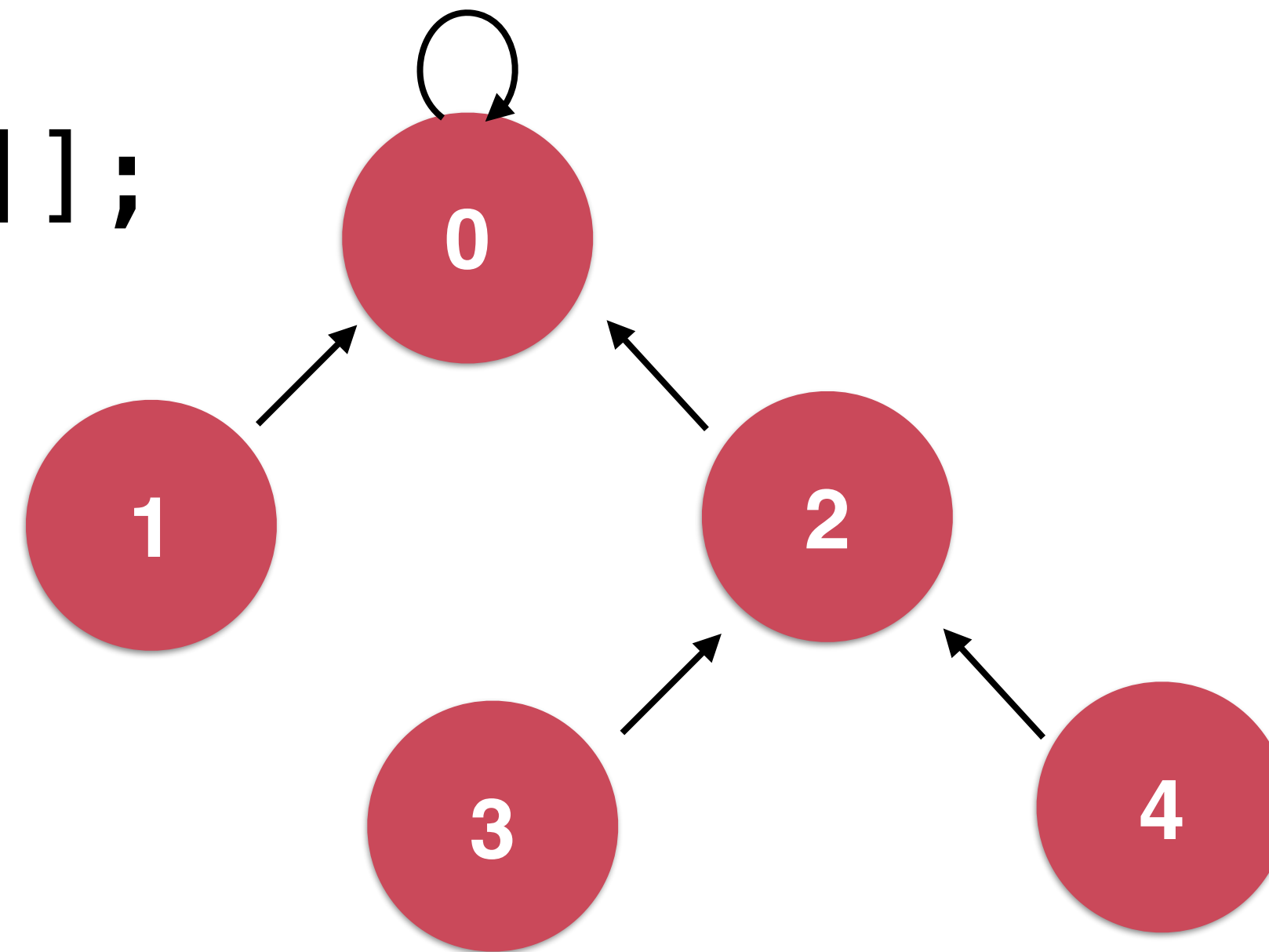
`parent[p] = parent[parent[p]];`



	0	1	2	3	4
parent	0	0	1	2	2

find 4

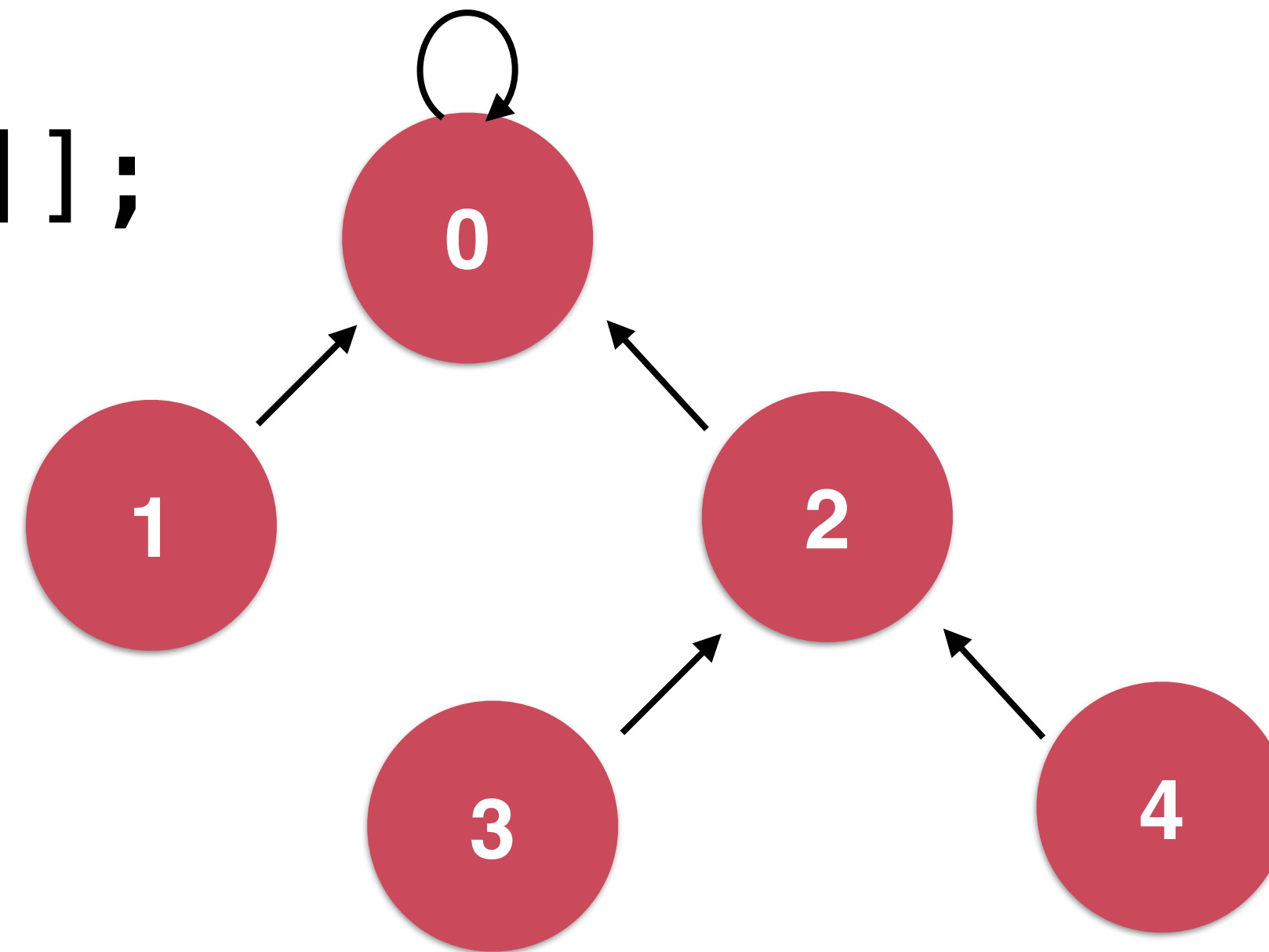
`parent[p] = parent[parent[p]];`



	0	1	2	3	4
parent	0	0	1	2	2

find 4

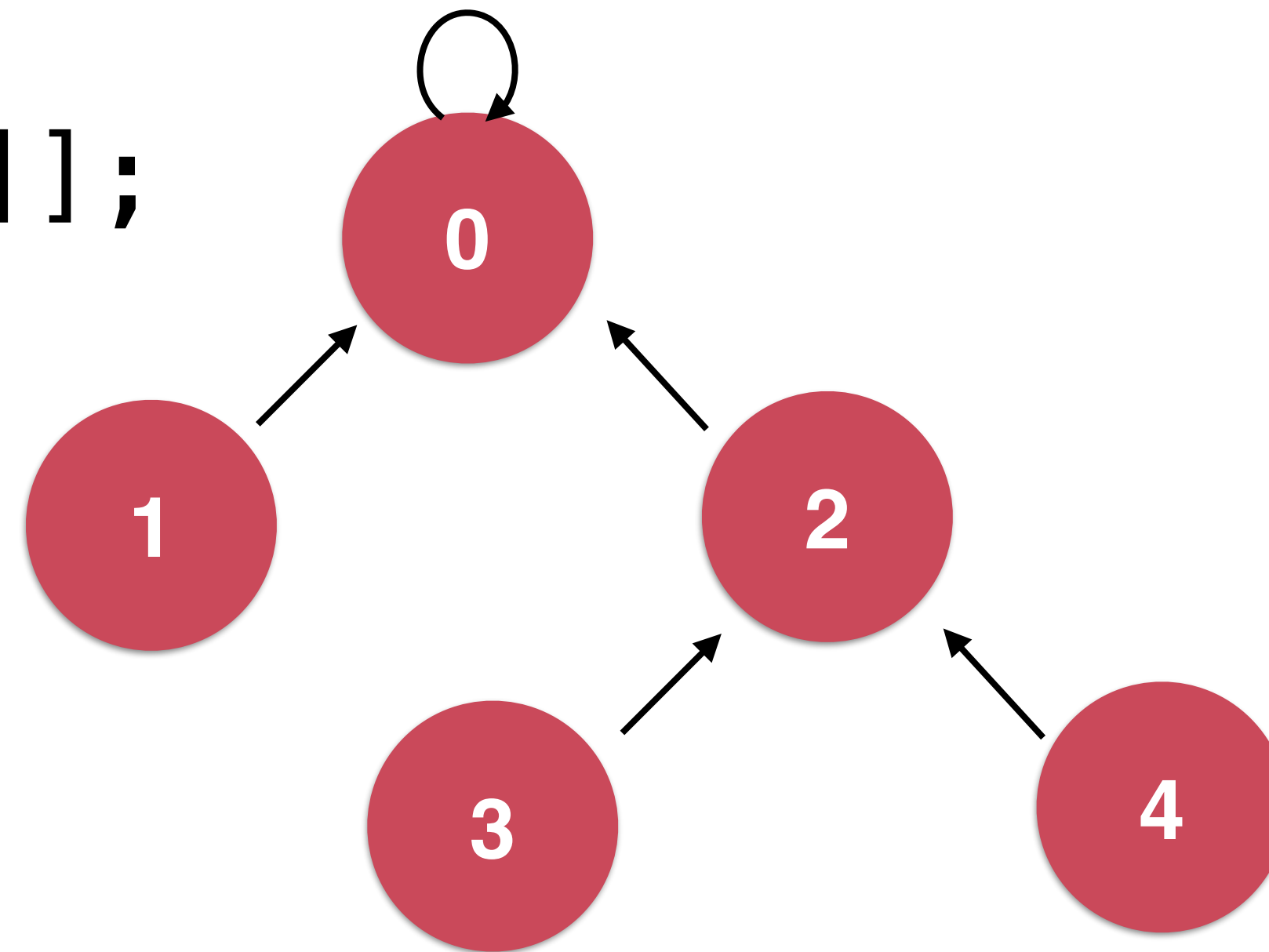
`parent[p] = parent[parent[p]];`



	0	1	2	3	4
parent	0	0	0	2	2

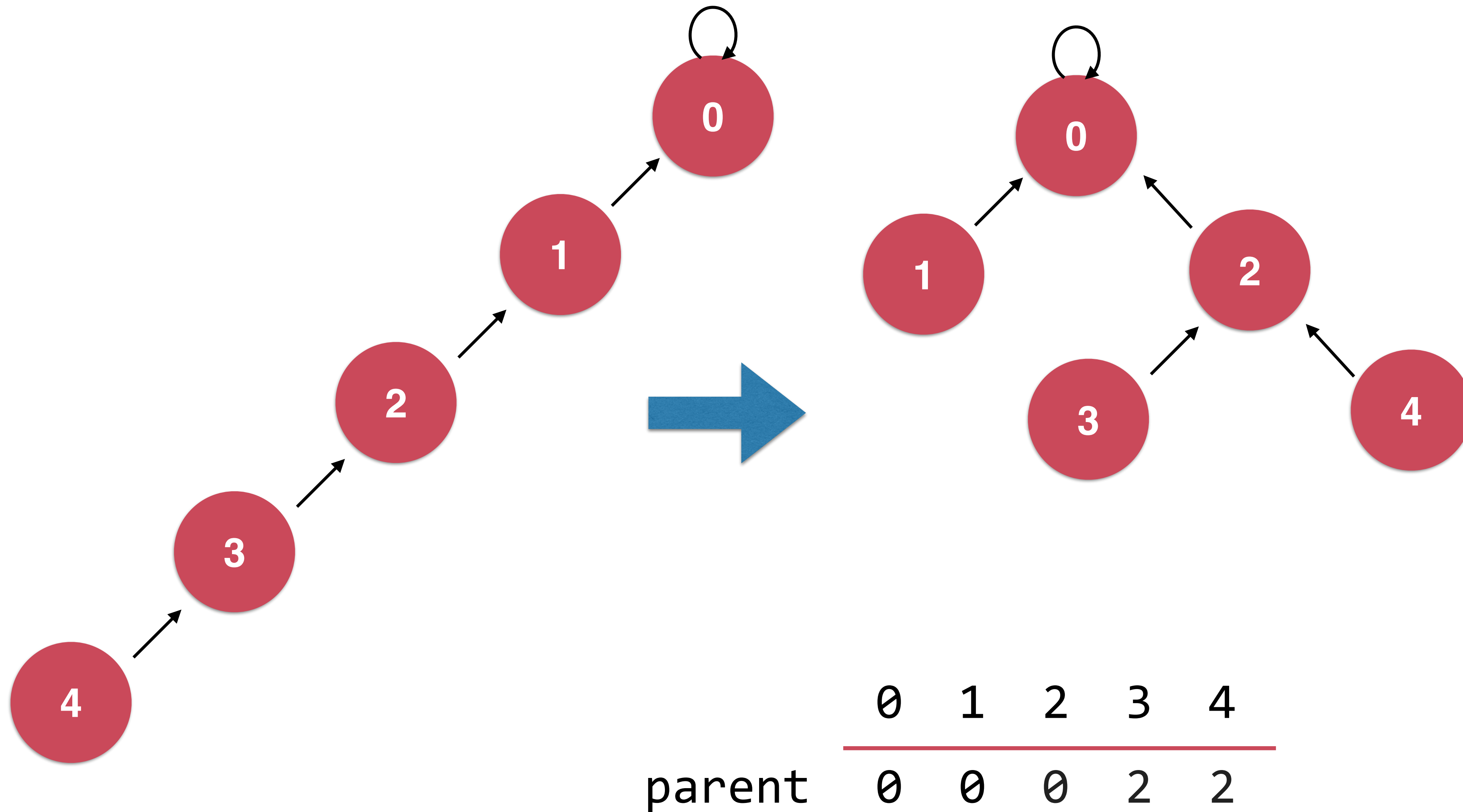
find 4

`parent[p] = parent[parent[p]];`



	0	1	2	3	4
parent	0	0	0	2	2

路径压缩 Path Compression

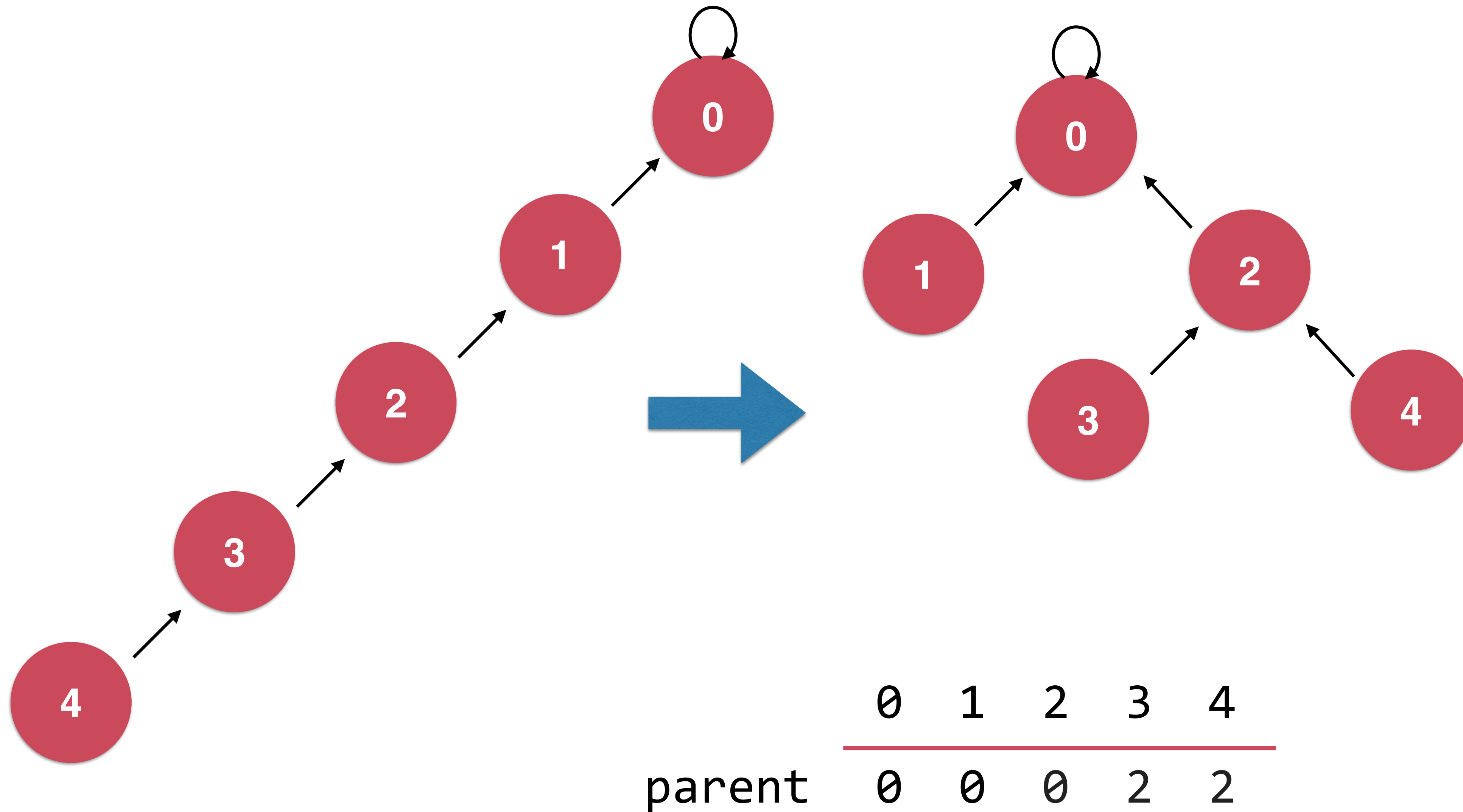


实践：路径压缩

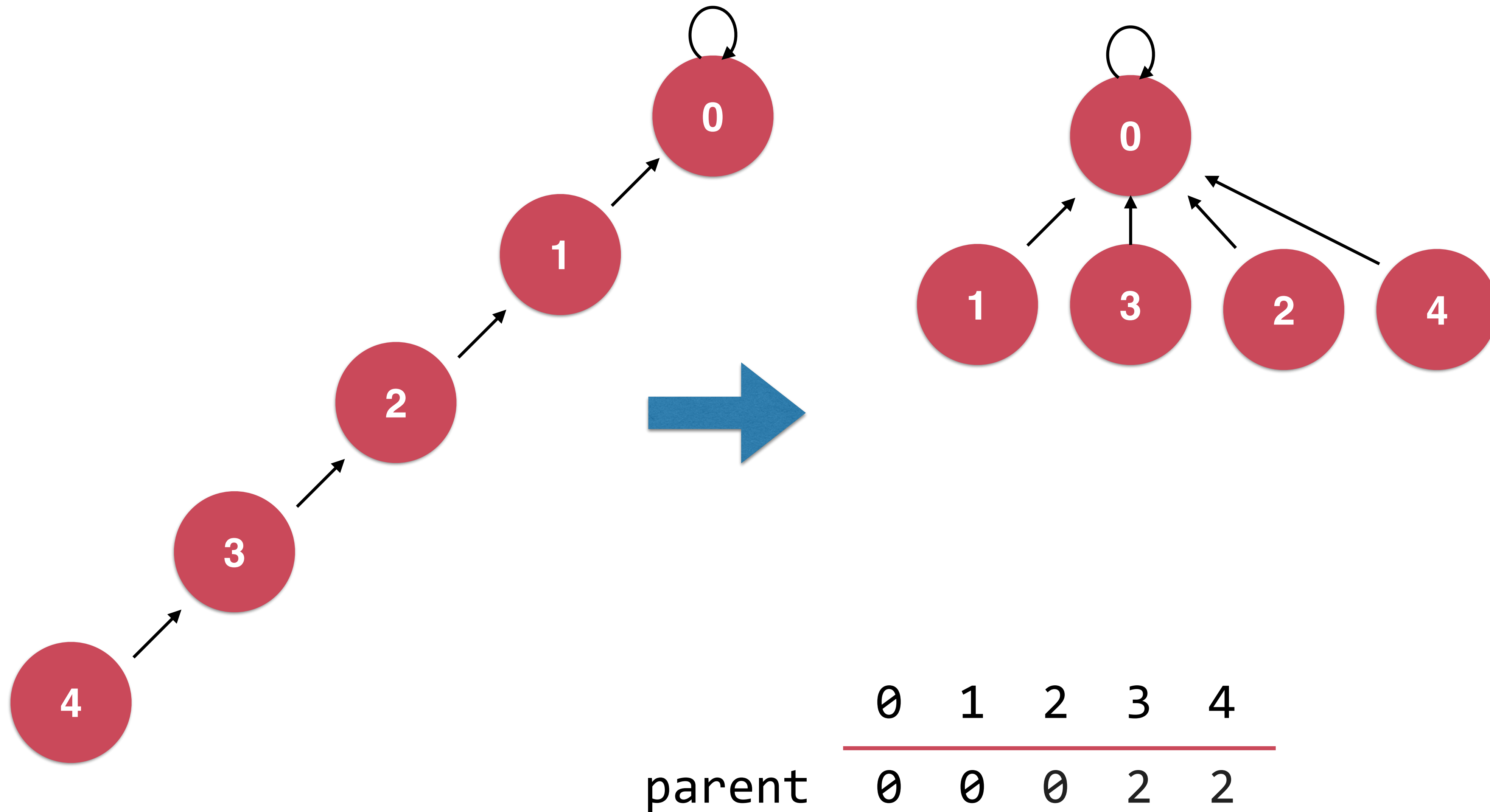
实践： 比较路径压缩和非路径压缩的效率

更多和并查集相关的话题

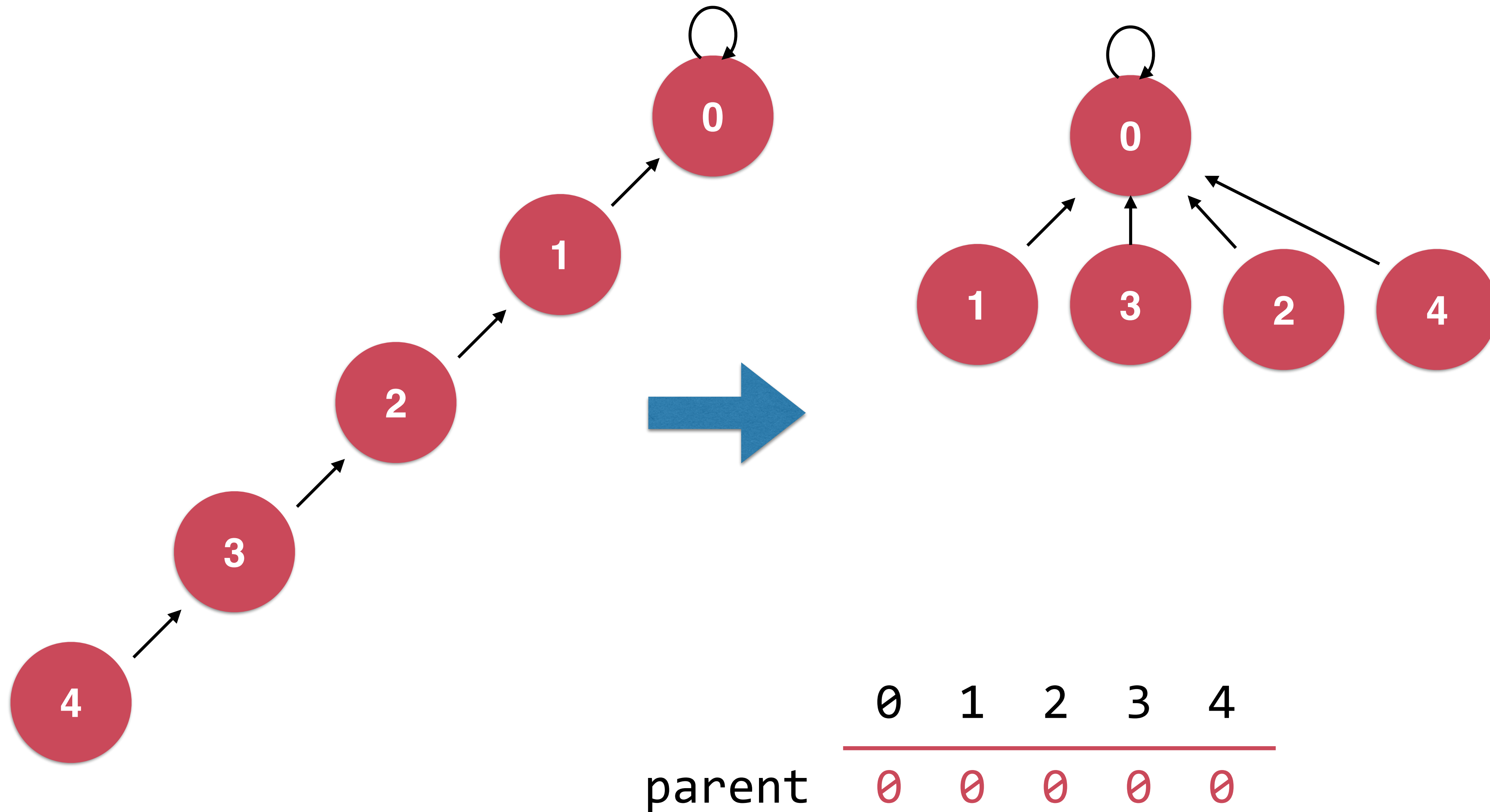
路径压缩 Path Compression



路径压缩 Path Compression

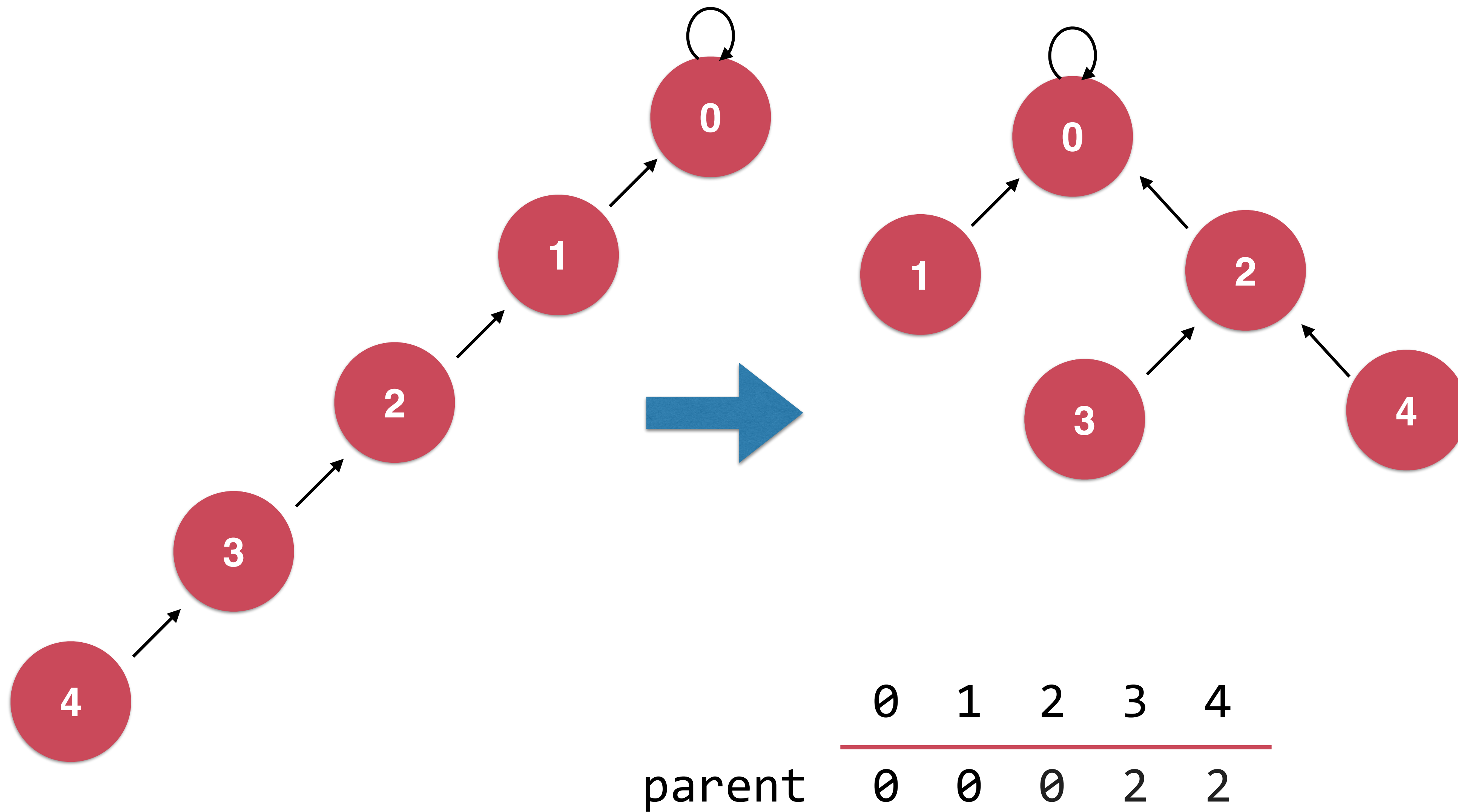


路径压缩 Path Compression

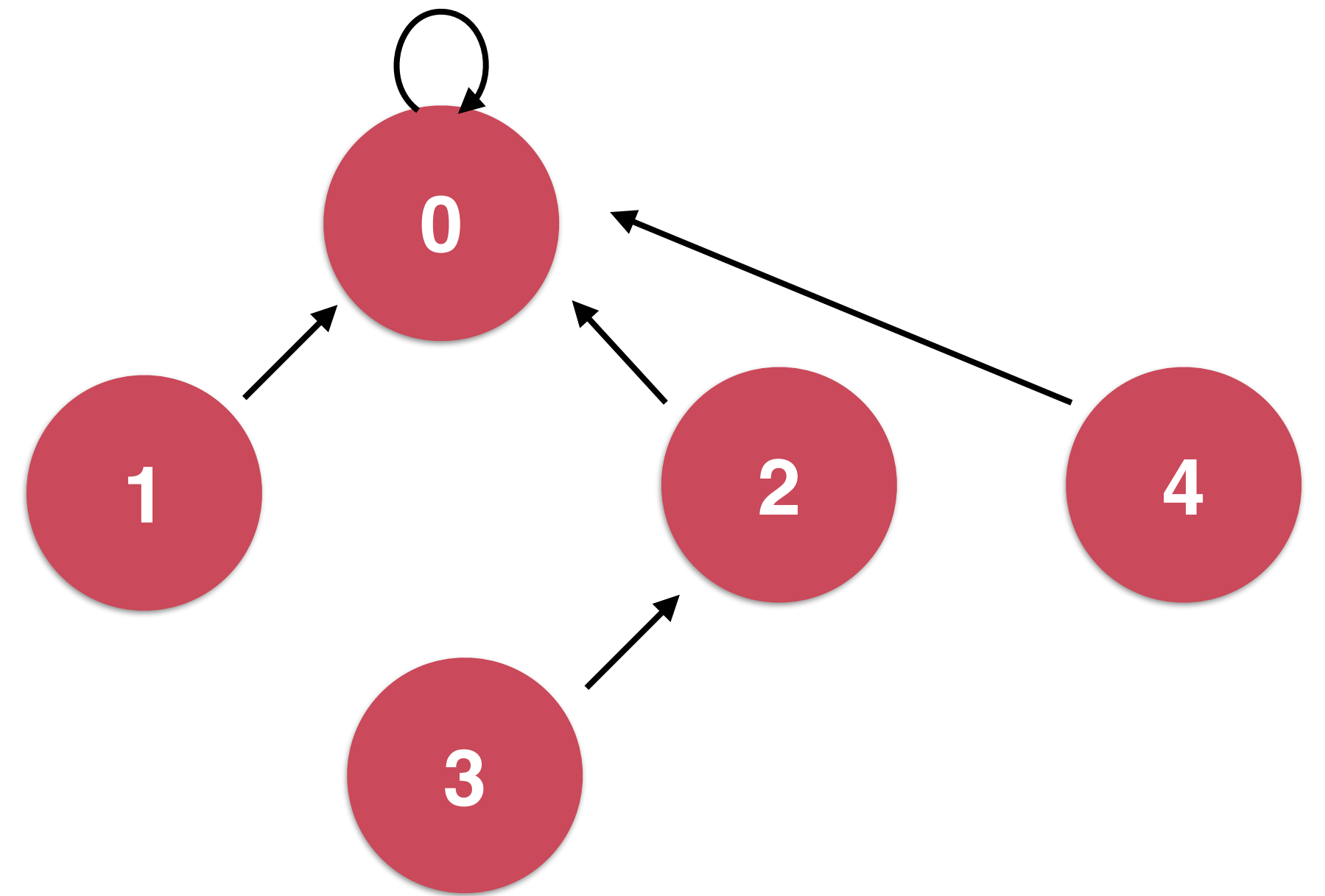
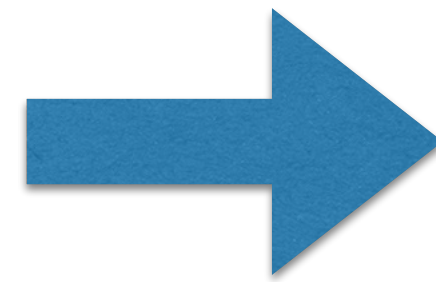
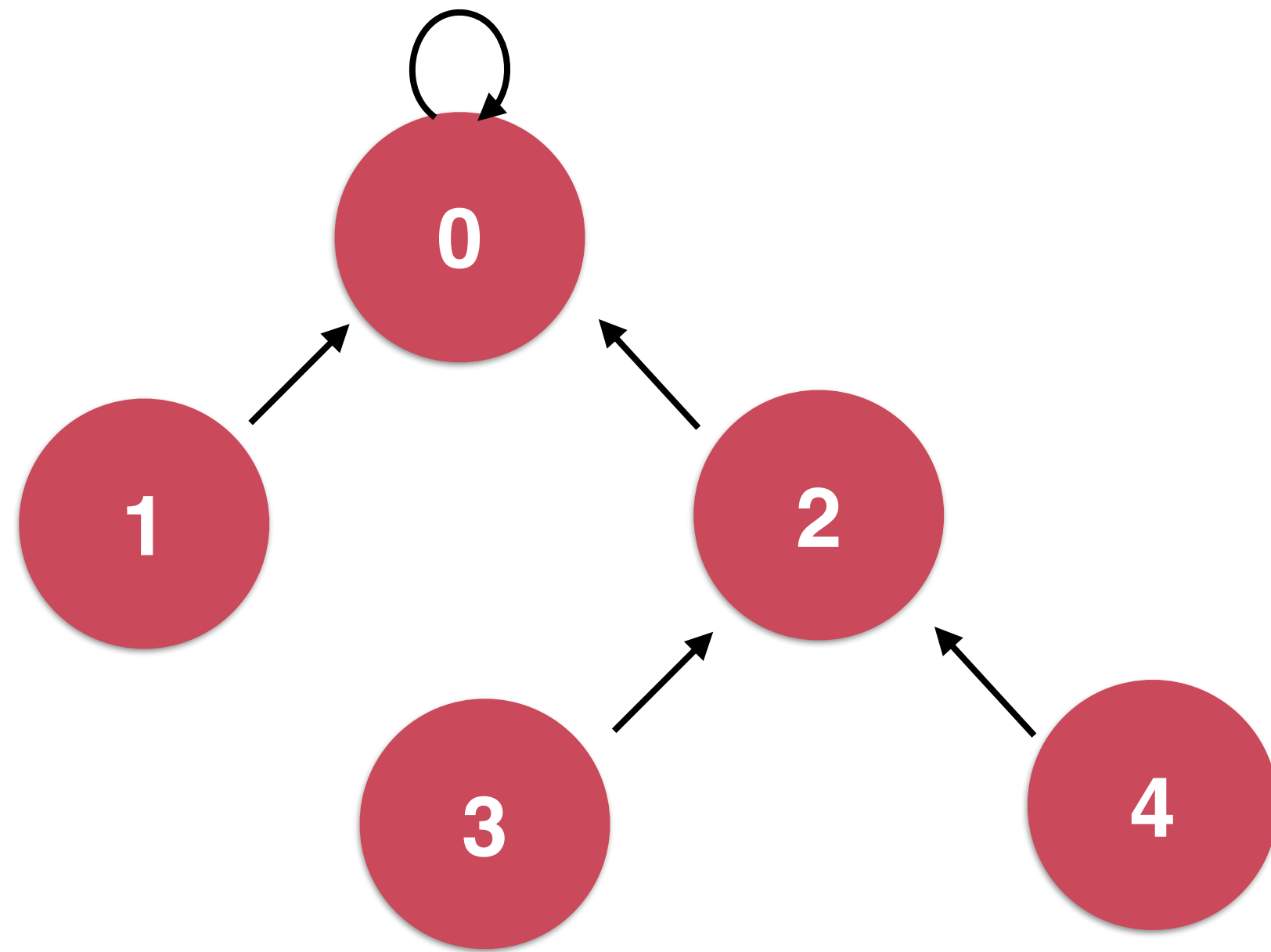


实践：递归的路径压缩

find 4

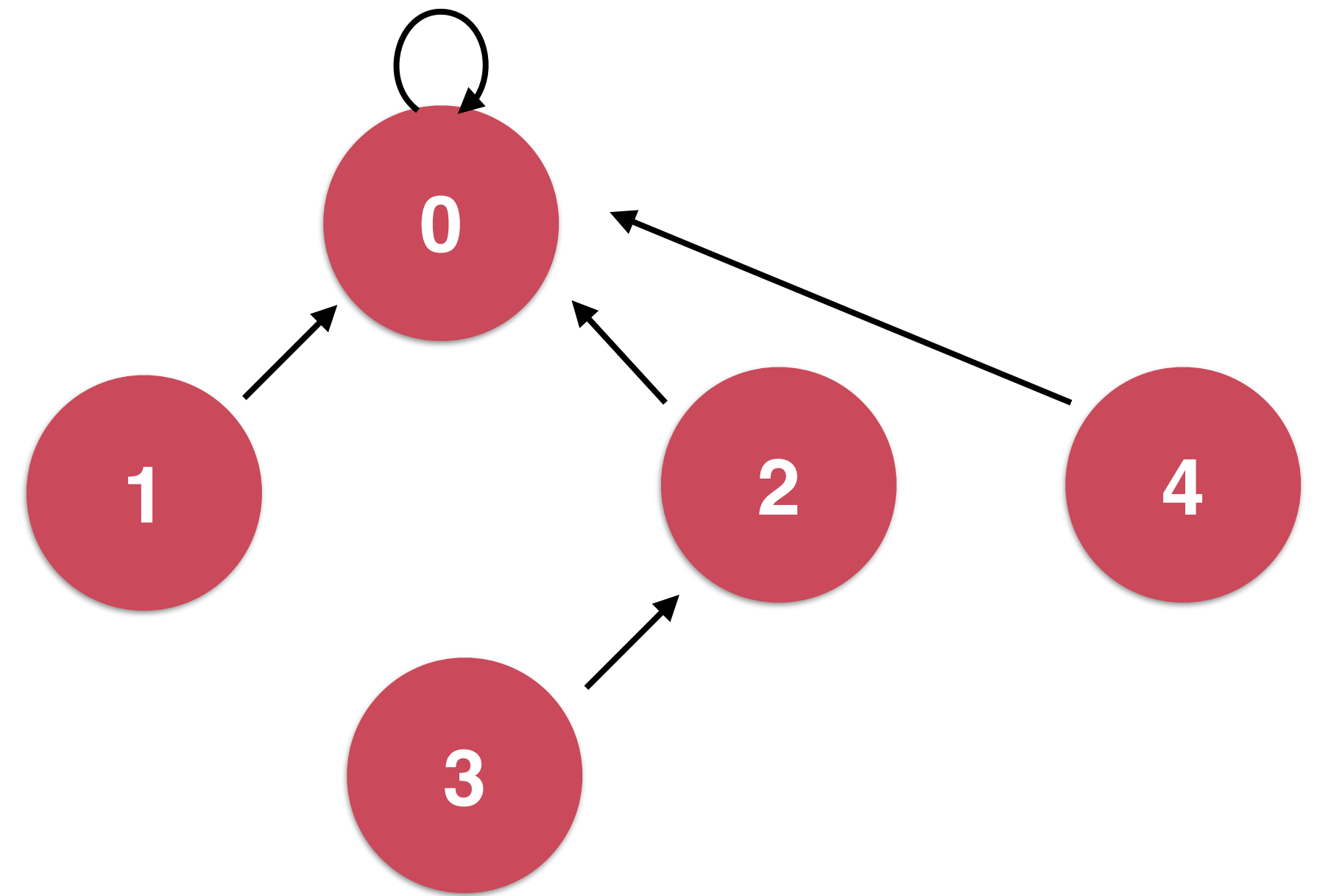
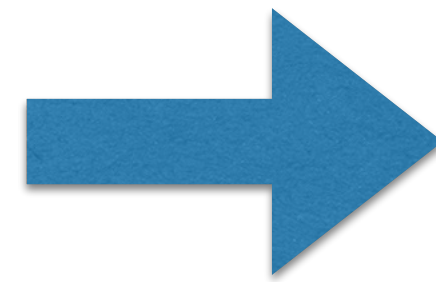
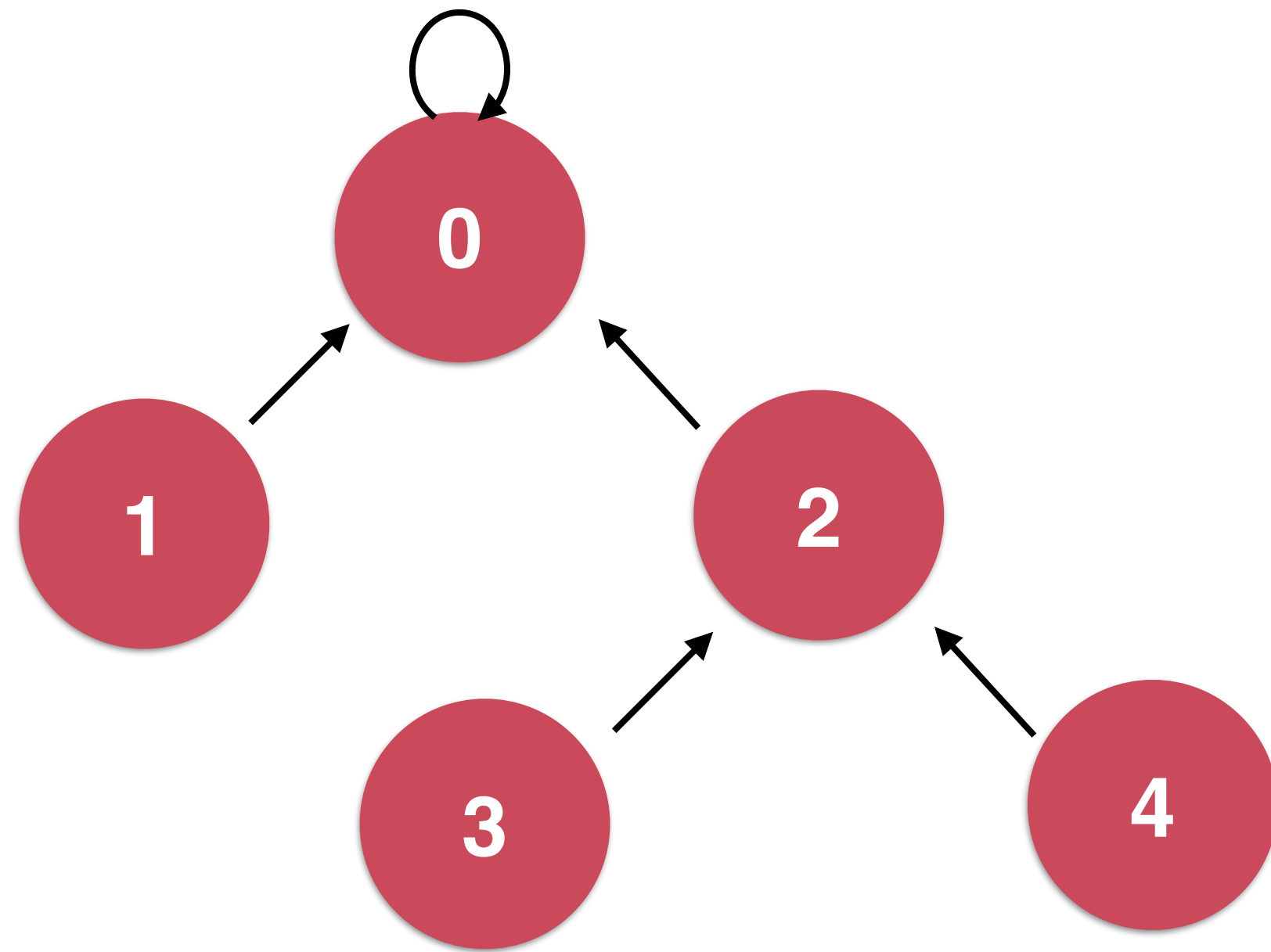


find 4



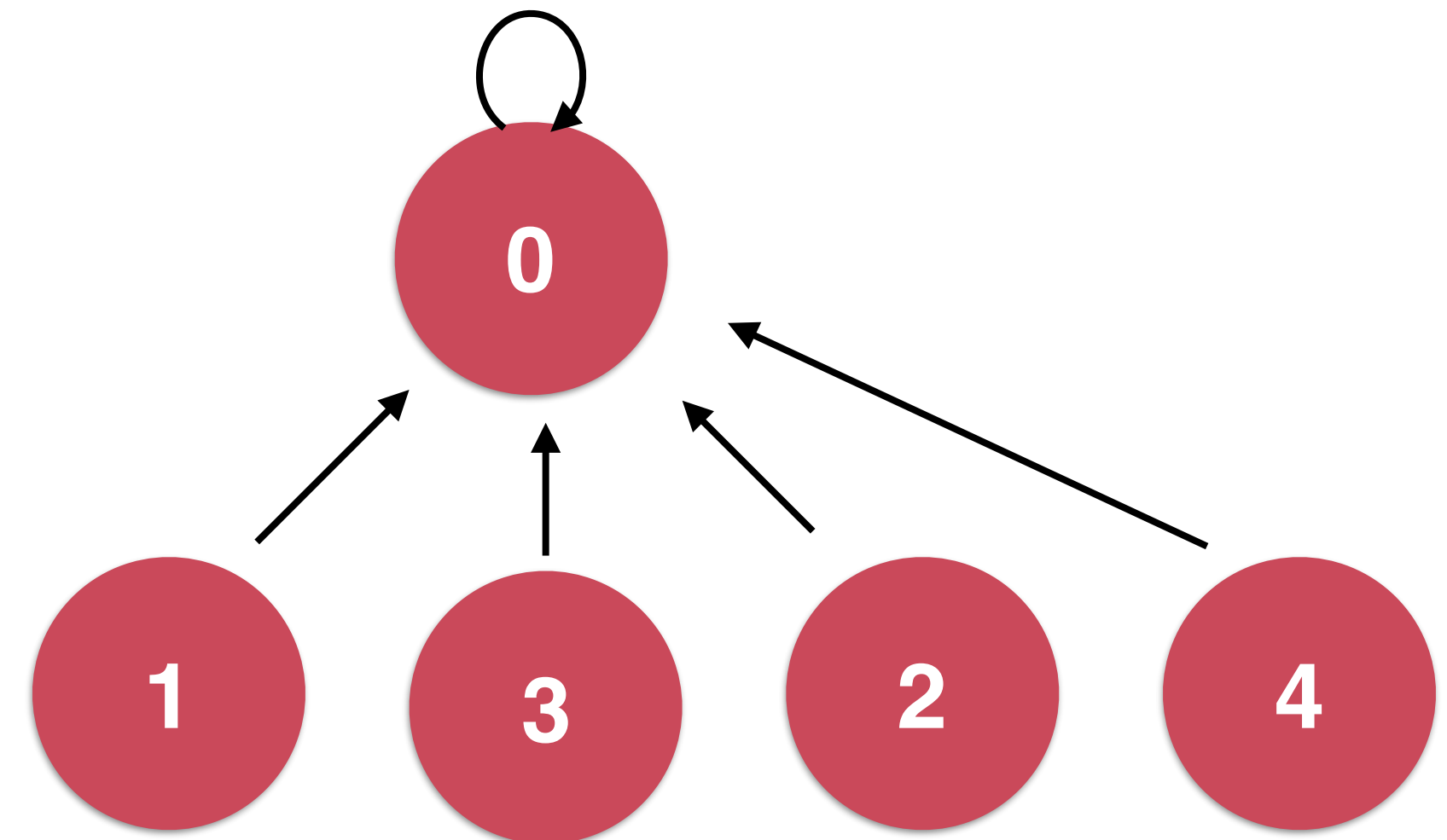
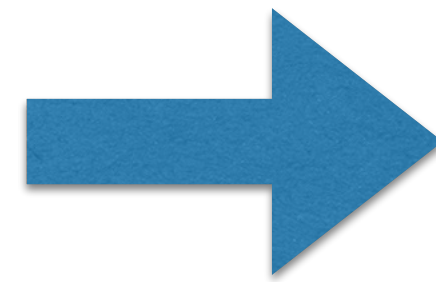
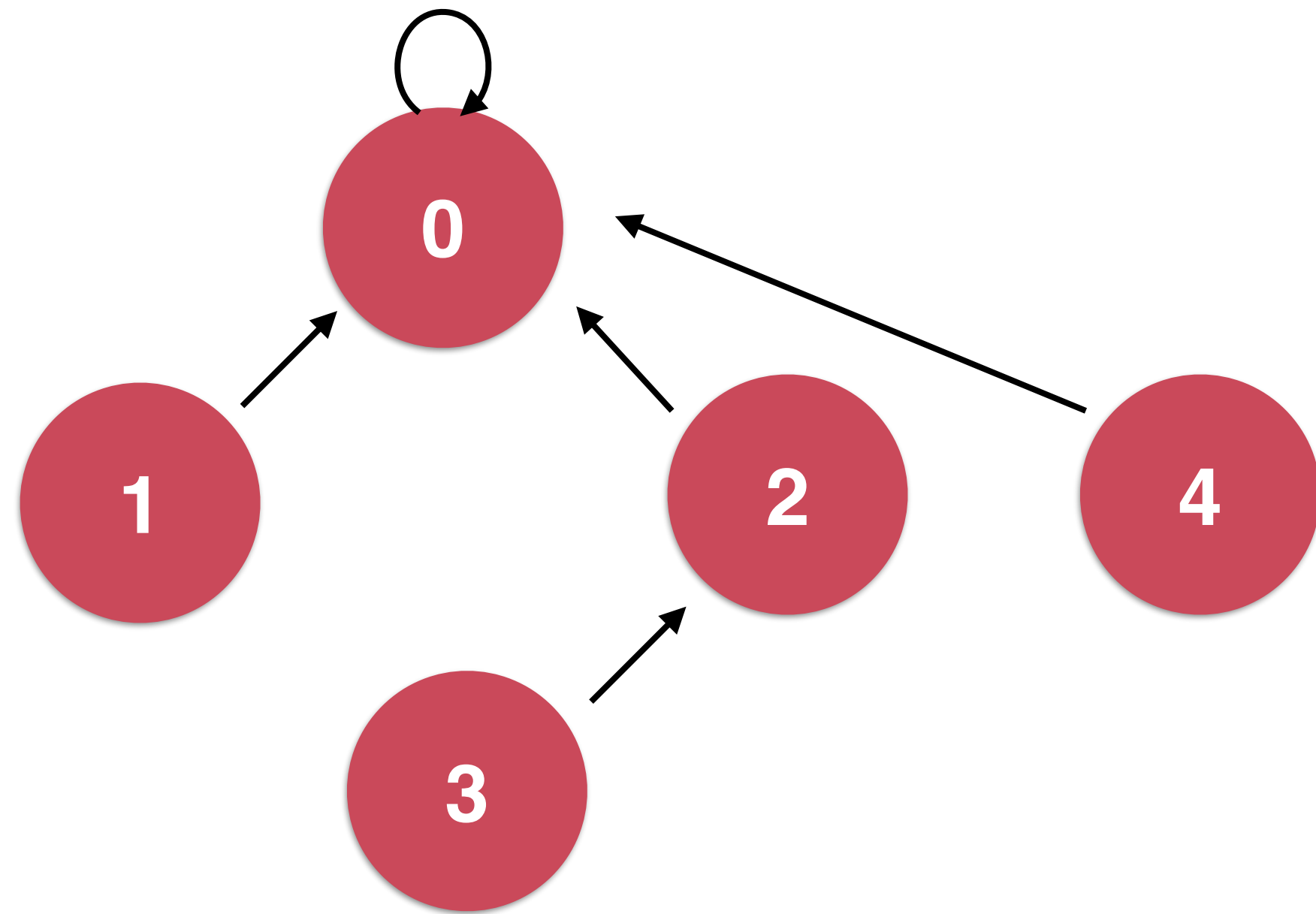
	0	1	2	3	4
parent	0	0	0	2	2

find 4



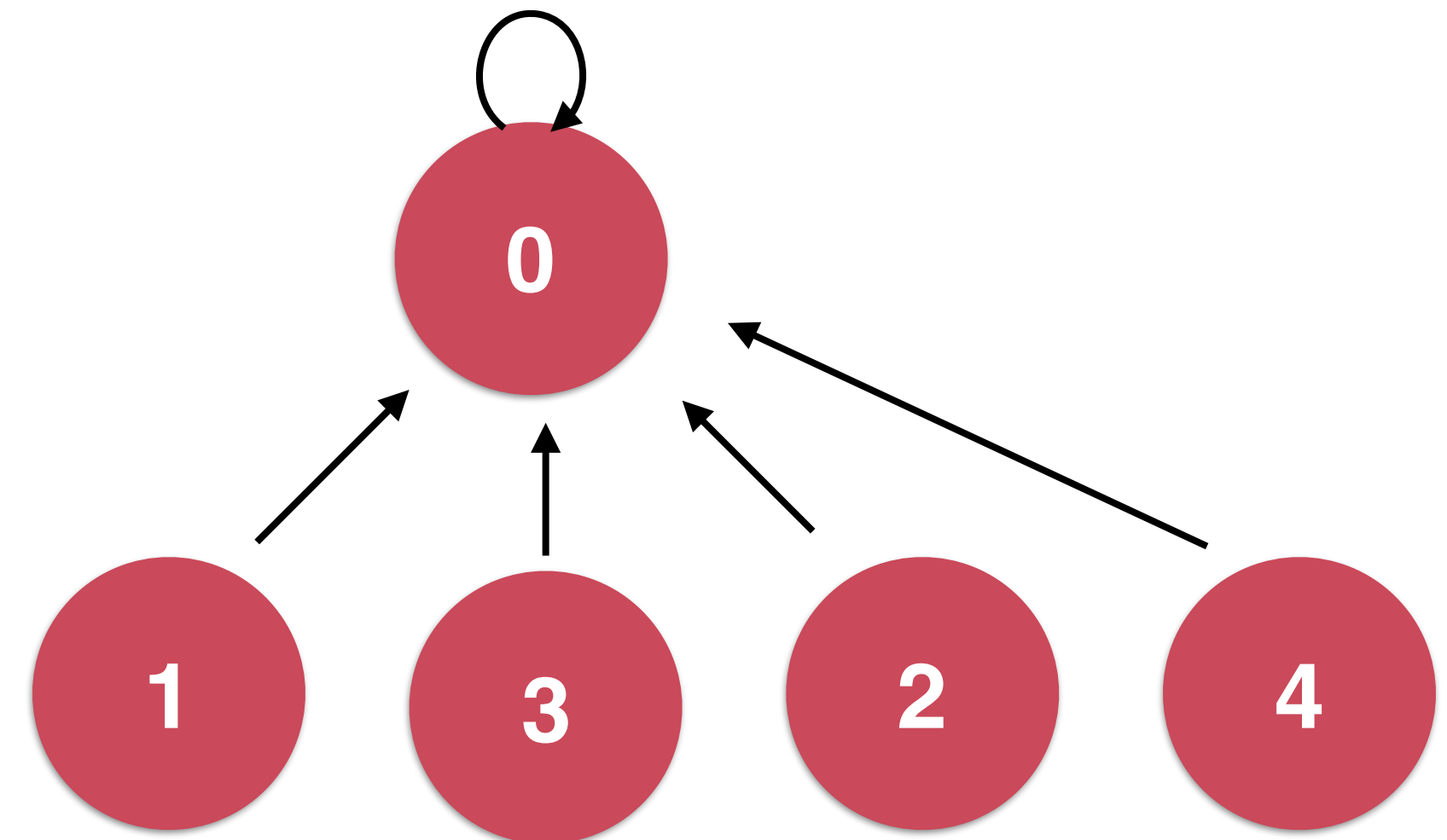
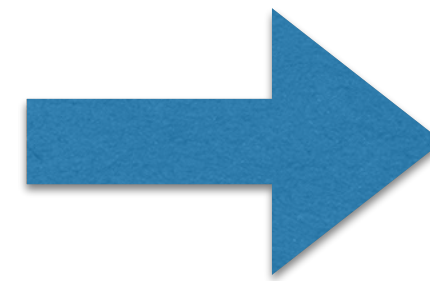
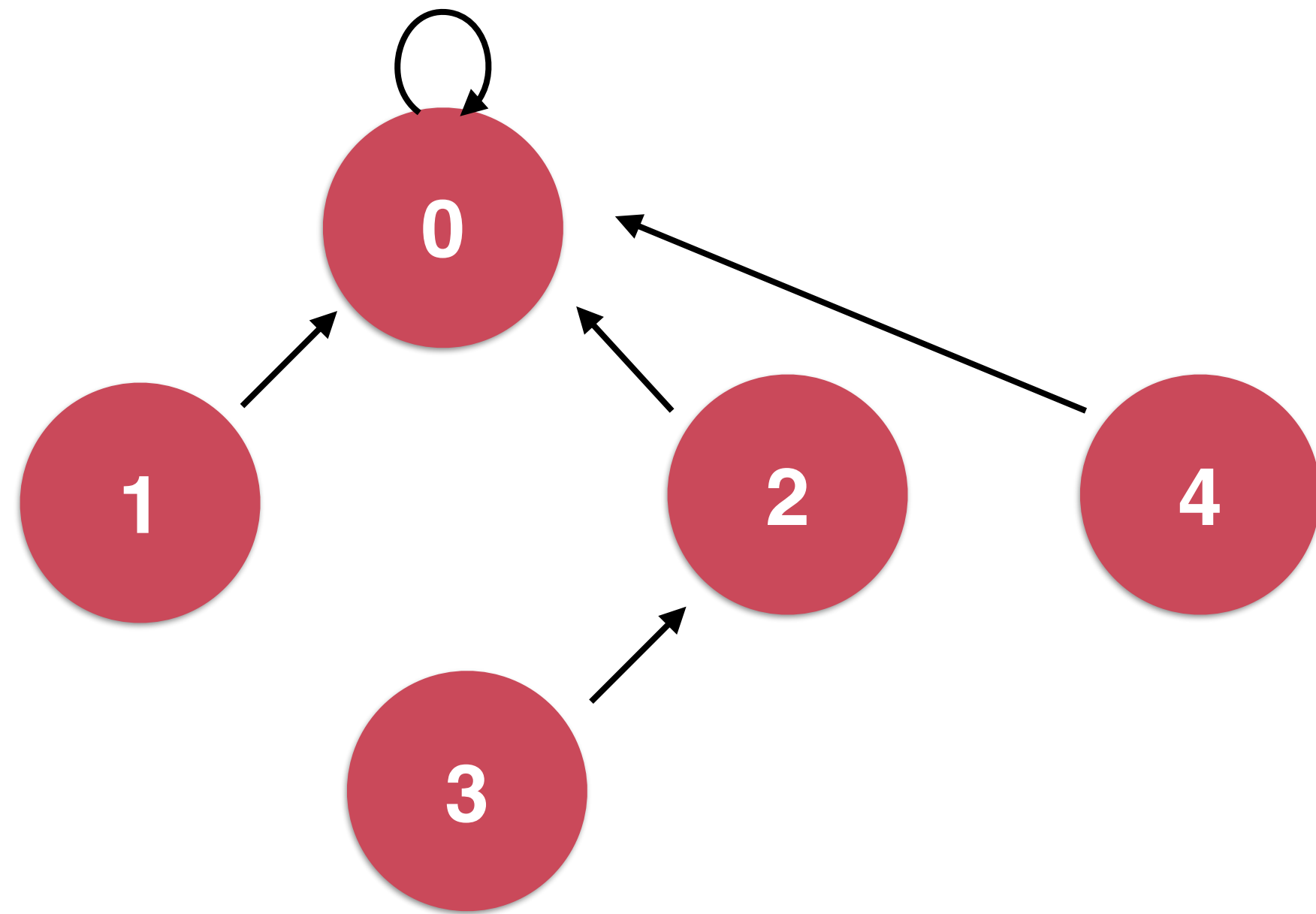
	0	1	2	3	4
parent	0	0	0	2	0

find 3



	0	1	2	3	4
parent	0	0	0	2	0

find 3



	0	1	2	3	4
parent	0	0	0	0	0

并查集的时间复杂度分析

$O(h)$

严格意义上： $O(\log^*n)$  iterated logarithm

并查集的时间复杂度分析

严格意义上： $O(\log^* n)$  iterated logarithm

$$\log^* n = \begin{cases} 0 & \text{if } (n \leq 1) \\ 1 + \log^* (\log n) & \text{if } (n > 1) \end{cases}$$

近乎是 $O(1)$ 级别的

更多Leetcode上Union Find相关的问题

并查集 Union Find

其他

欢迎大家关注我的个人公众号：是不是很酷



玩儿转数据结构

liuyubobobo