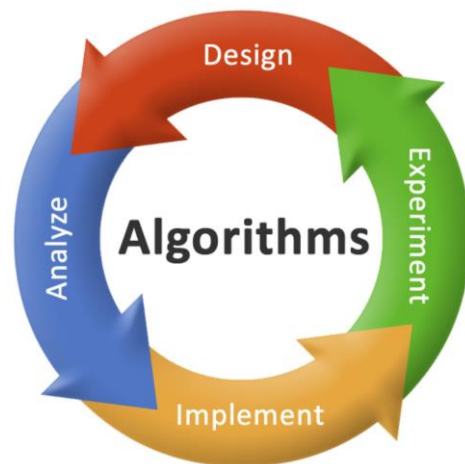


华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY



Algorithm Design and Analysis

算法设计与分析

Chapter 7: Union-find

张乾坤

不相交集合 (Disjoint-sets) 数据结构

目标：对一系列不相交的集合支持以下三种操作：

- Make-Set(x)：创建一个只包含一个元素x的新的集合
- Find(x)：返回一个包含x的集合中的代表元素
- Union(x,y)：将包含x和y的集合合并

参数：

- m=以上三种函数的调用次数
- n=集合中元素的数量

不相交集合 (Disjoint-sets) 数据结构

动态连通性 (Dynamic Connectivity) :

给定一个图G, 初始化为空集, 并支持以下三种操作:

不相交集合=连通分支

- Add-Node(u): 加入新的点u
- Add-Edge(u,v): 在u和v之间加入一条新的边
- Is-Connected(u,v): 判断u和v是否连通

1 Make-Set

1 Union

2 Find

不相交集合（Disjoint-sets）数据结构：应用

1964年提出该数据结构：

An Improved Equivalence Algorithm

BERNARD A. GALLER AND MICHAEL J. FISHER
University of Michigan, Ann Arbor, Michigan

An algorithm for assigning storage on the basis of EQUIVALENCE, DIMENSION and COMMON declarations is presented. The algorithm is based on a tree structure, and has reduced computation time by 40 percent over a previously published algorithm by identifying all equivalence classes with one scan of the EQUIVALENCE declarations. The method is applicable in any problem in which it is necessary to identify equivalence classes, given the element pairs defining the equivalence relation.

应用：

连通分支

Kruskal算法

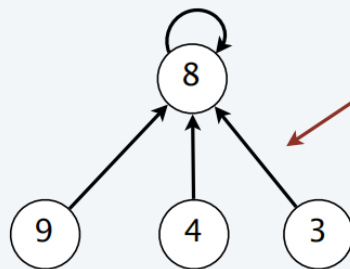
...

不相交集合 (Disjoint-sets) 数据结构

父链接 (Parent-link) 表示法: 把集合表示成一棵元素的树

- 每个元素都有一个显式的父指针
- 根节点指针指向自己, 是集合中的代表性元素
- Find(x): 找到包含x的树的根
- Union(x,y): 合并两棵树, 把一个根节点的指针指向另一个

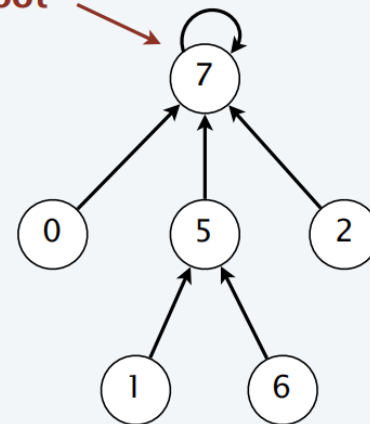
UNION(3, 5)



parent of 3 is 8



root

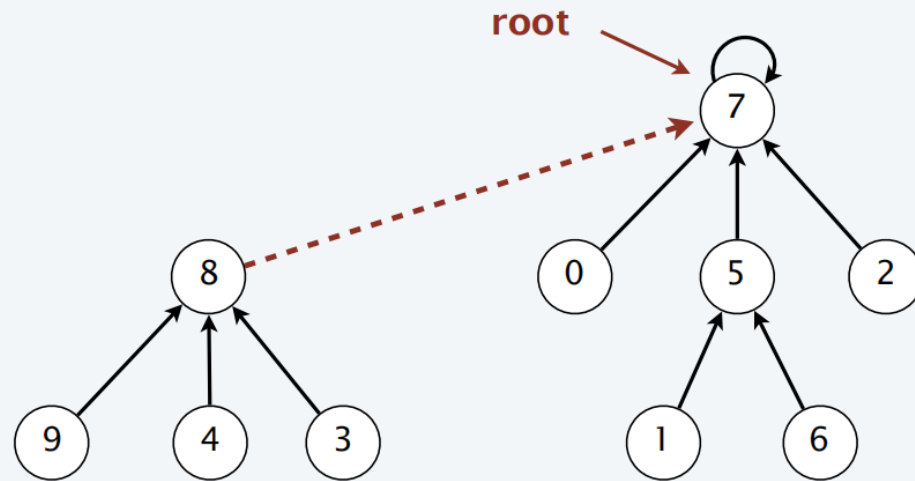


不相交集合 (Disjoint-sets) 数据结构

父链接 (Parent-link) 表示法: 把集合表示成一棵元素的树

- 每个元素都有一个显式的父指针
- 根节点指针指向自己, 是集合中的代表性元素
- Find(x): 找到包含x的树的根
- Union(x,y): 合并两棵树, 把一个根节点的指针指向另一个

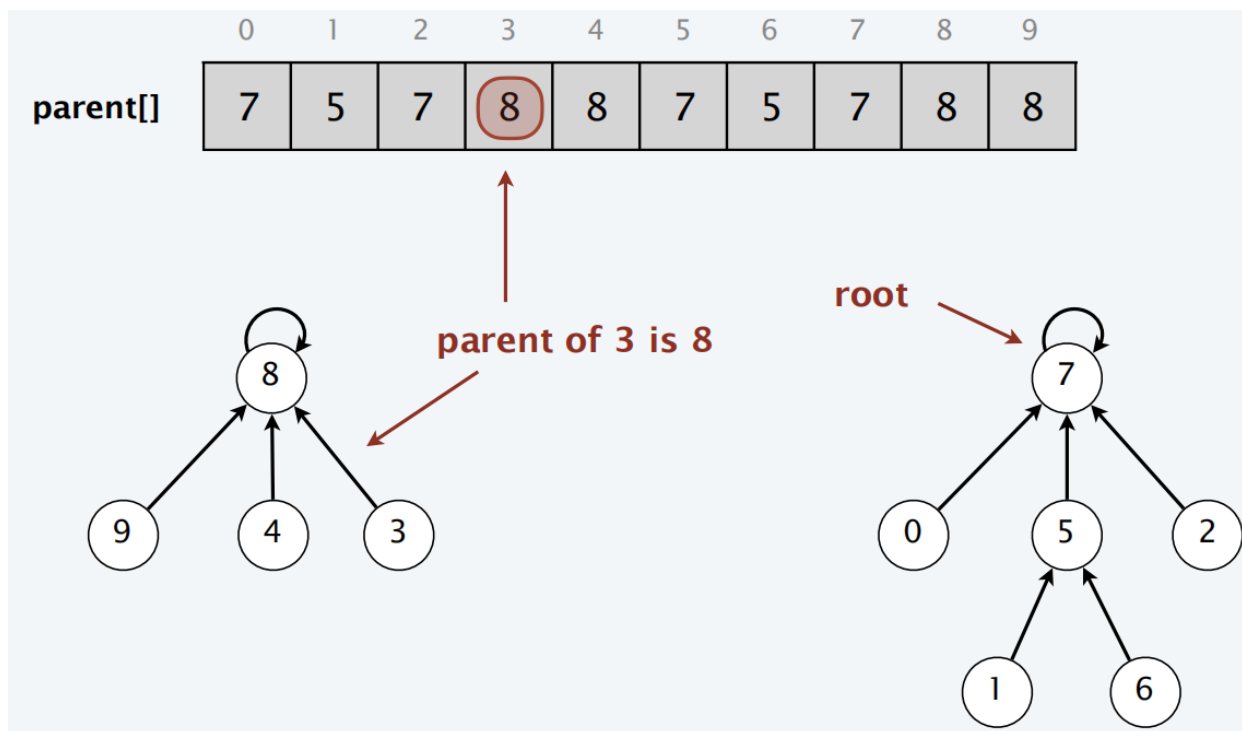
UNION(3, 5)



不相交集合 (Disjoint-sets) 数据结构

数组表示法：把集合表示成一棵元素的树

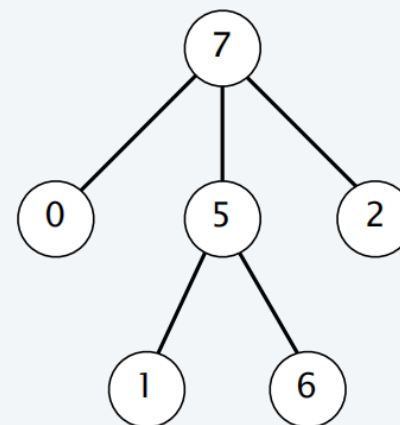
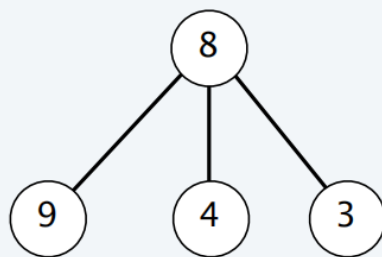
- 定义一个长度为n的数组parent[] 必须事先知道n的值
- $\text{parent}[i]=j$ ：元素i的父节点是元素j



Naïve Linking

将一棵树的根节点指向另一棵树的根节点

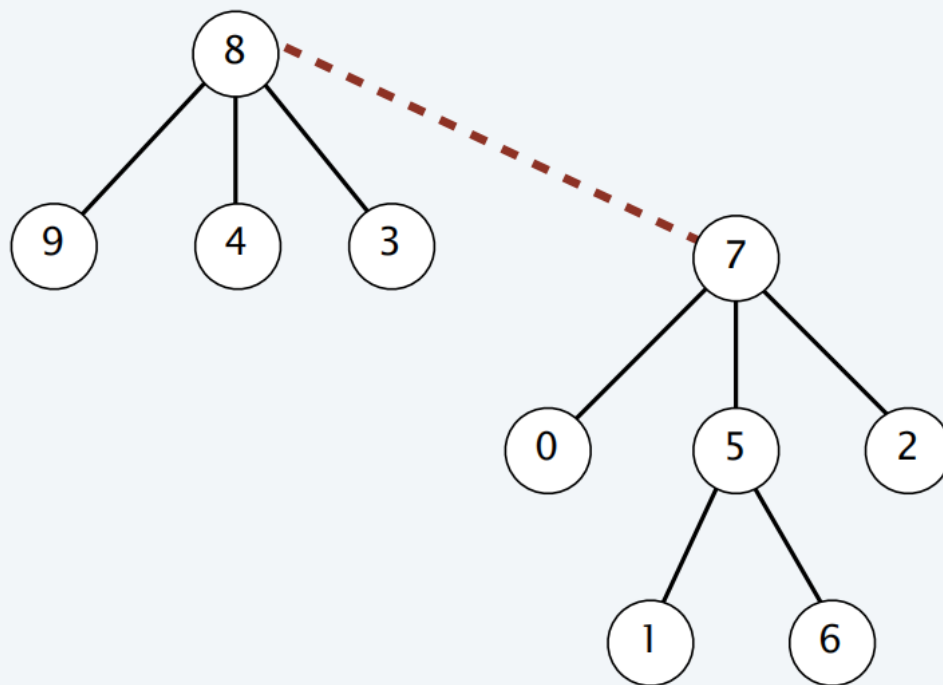
UNION(5, 3)



Naïve Linking

将一棵树的根节点指向另一棵树的根节点

UNION(5, 3)



Naïve Linking

将一棵树的根节点指向另一棵树的根节点

MAKE-SET(x)

$parent[x] \leftarrow x.$

FIND(x)

WHILE ($x \neq parent[x]$)

$x \leftarrow parent[x].$

RETURN $x.$

UNION(x, y)

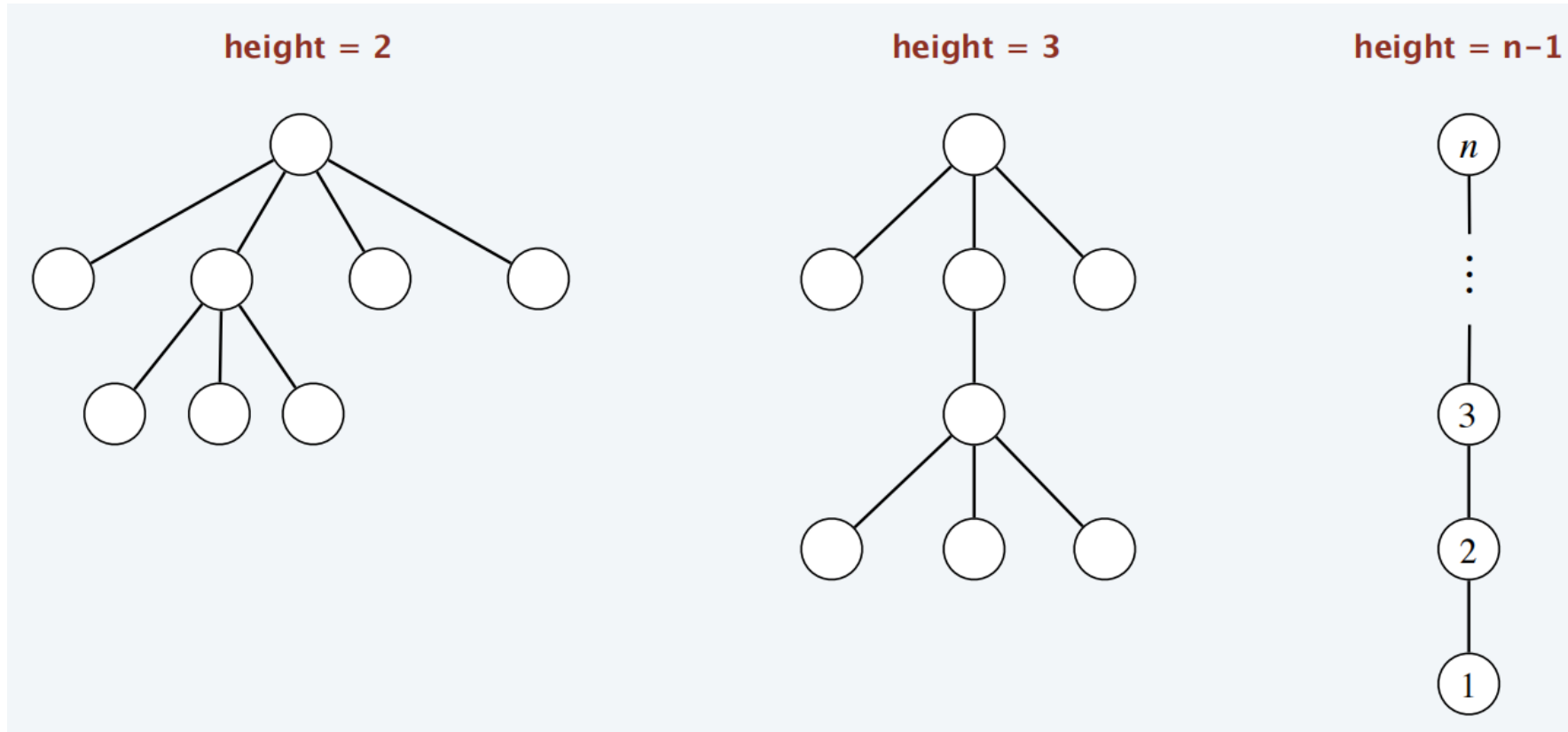
$r \leftarrow \text{FIND}(x).$

$s \leftarrow \text{FIND}(y).$

$parent[r] \leftarrow s.$

Naïve Linking: 分析

Union和Find操作在最差情况下都需要 $\Theta(n)$ 时间，其中 n 是元素个数

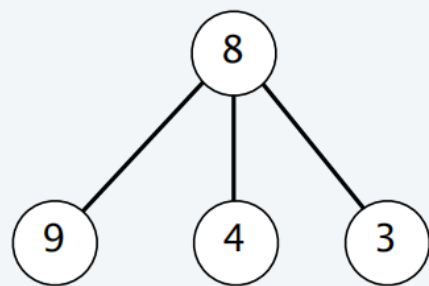


Link-by-size

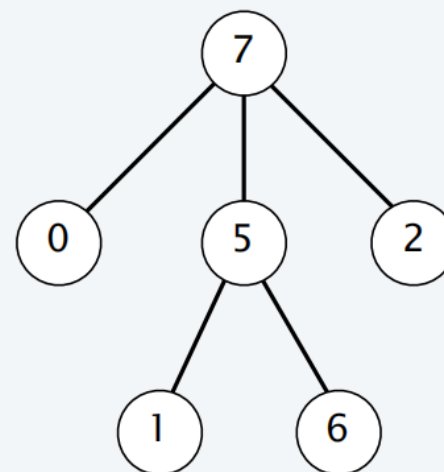
对每棵树记录树的大小，总是把小的树合并到大的树

UNION(5, 3)

size = 4



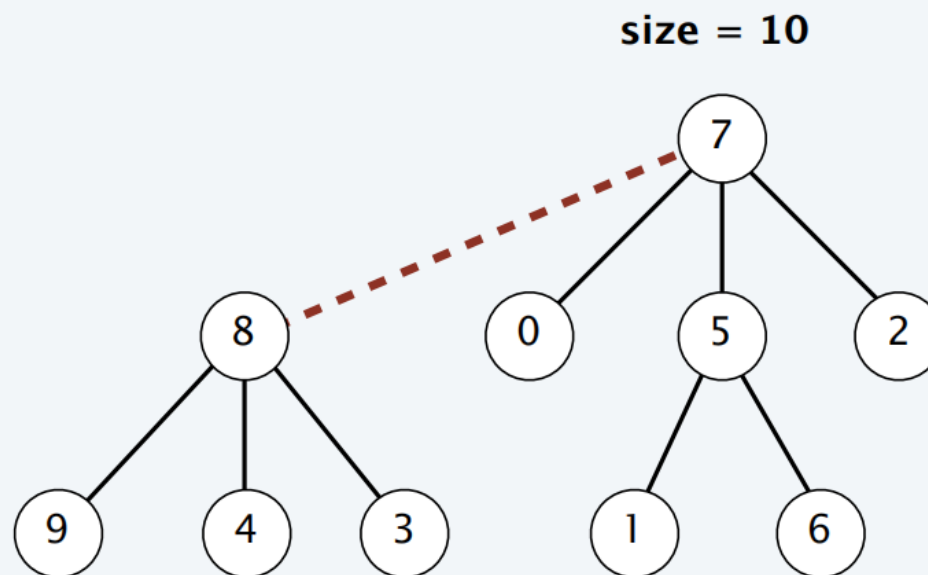
size = 6



Link-by-size

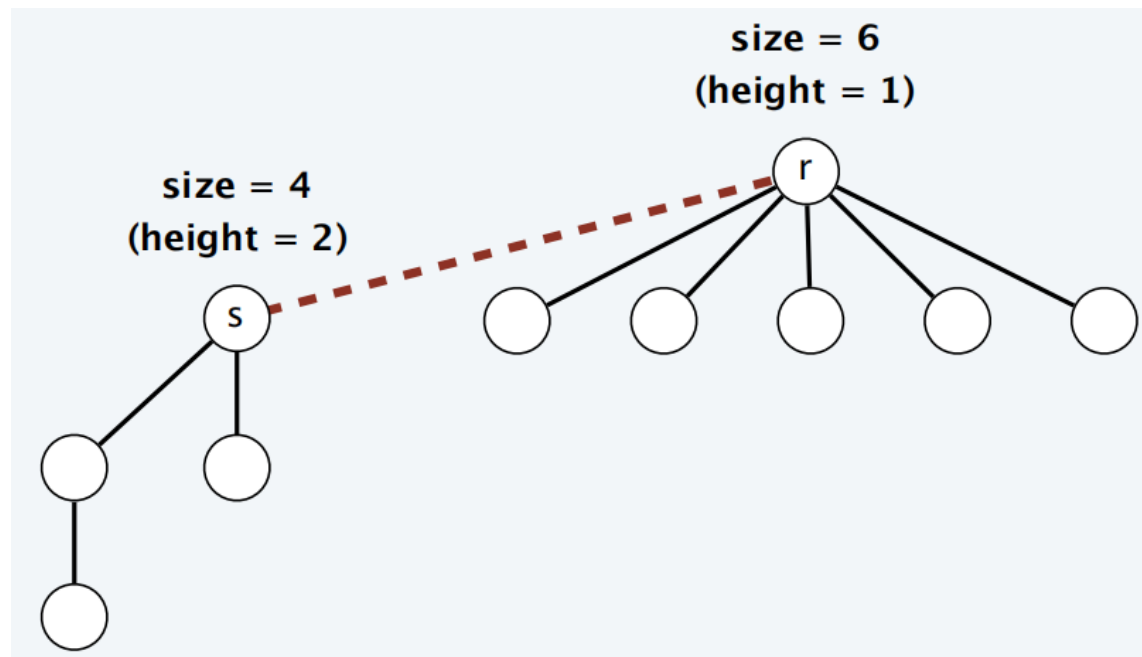
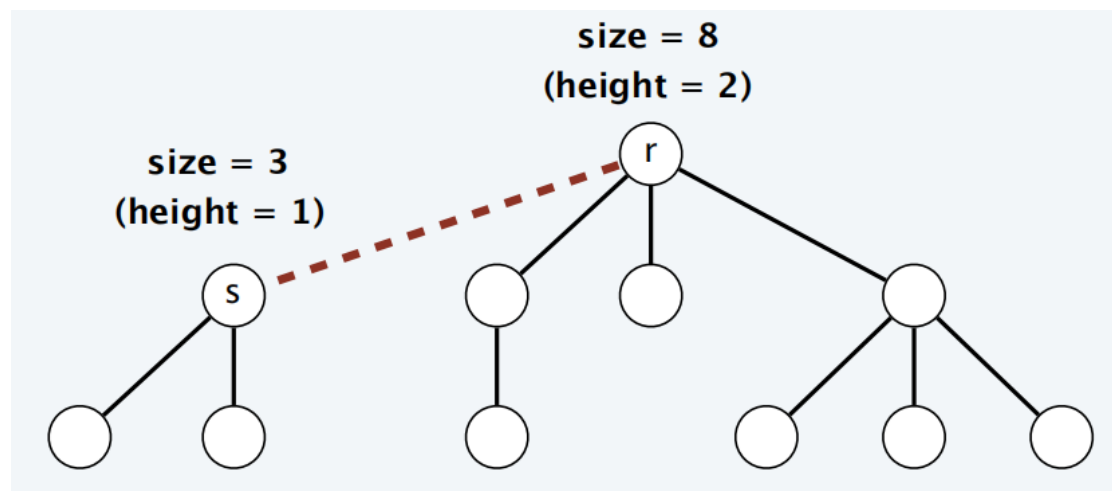
对每棵树记录树的大小，总是把小的树合并到大的树

UNION(5, 3)



Link-by-size: 分析

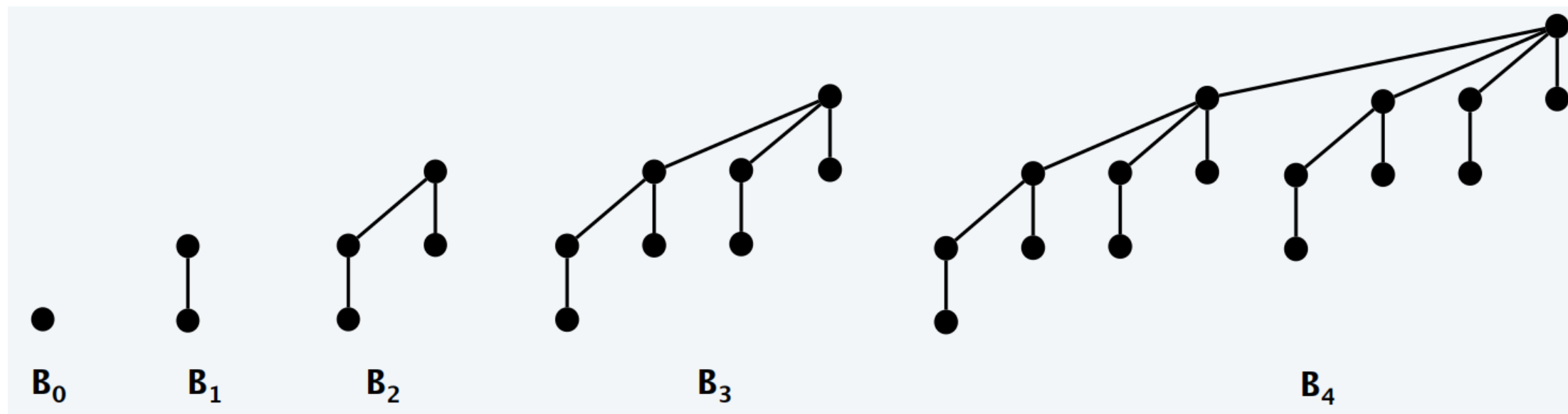
性质：对任意根节点 r , $\text{size}[r] \geq 2^{\text{height}(r)}$



Link-by-size: 分析

Union和Find操作在最差情况下都需要 $\Theta(\log n)$ 时间，其中 n 是元素个数

定理：根据Link-by-size策略，一个 n 个点的树可具有高度 $\log_2 n$

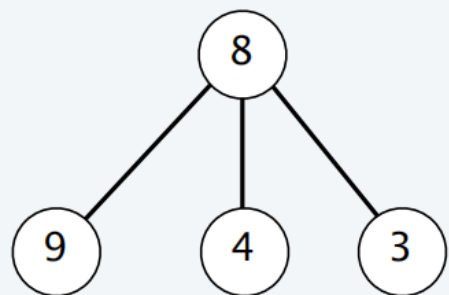


Link-by-rank

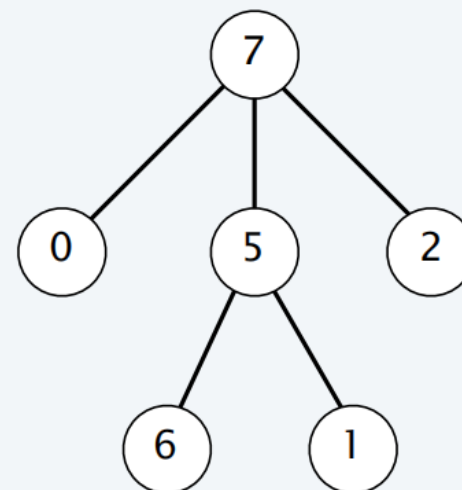
每个节点维护一个整数表示rank，将较小的rank指向较大的rank（例如rank=height）

UNION(5, 3)

rank = 1

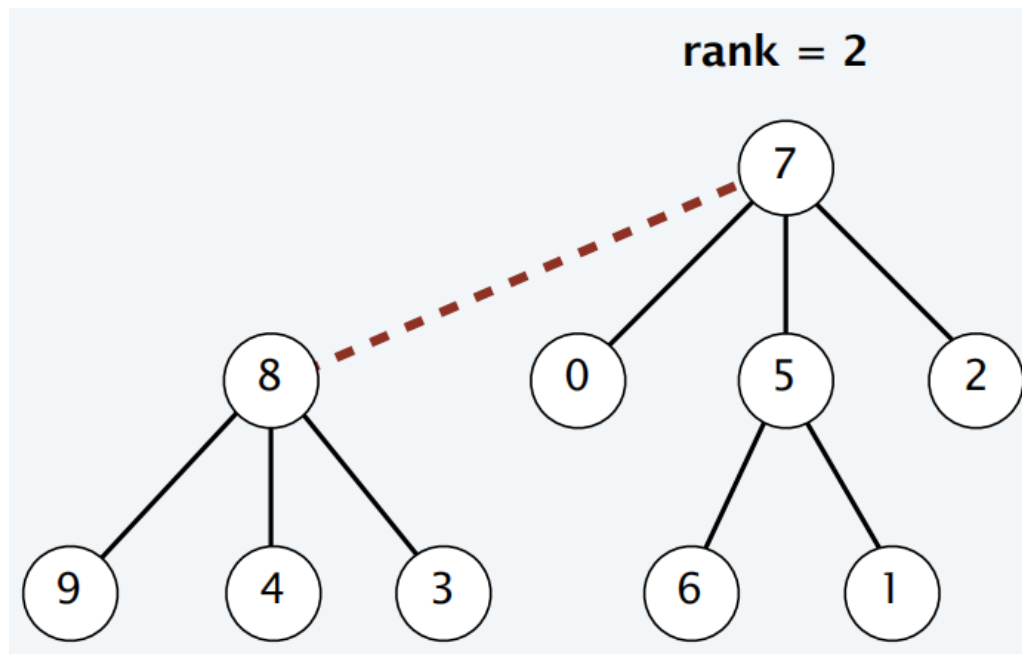


rank = 2



Link-by-rank

每个节点维护一个整数表示rank，将较小的rank指向较大的rank（例如rank=height）



Link-by-rank

每个节点维护一个整数表示rank，将较小的rank指向较大的rank（例如rank=height）

MAKE-SET(x)

$parent[x] \leftarrow x.$

$rank[x] \leftarrow 0.$

FIND(x)

WHILE ($x \neq parent[x]$)

$x \leftarrow parent[x].$

RETURN $x.$

UNION(x, y)

$r \leftarrow \text{FIND}(x).$

$s \leftarrow \text{FIND}(y).$

IF ($r = s$) RETURN.

ELSE IF ($rank[r] > rank[s]$)

$parent[s] \leftarrow r.$

ELSE IF ($rank[r] < rank[s]$)

$parent[r] \leftarrow s.$

ELSE

$parent[r] \leftarrow s.$

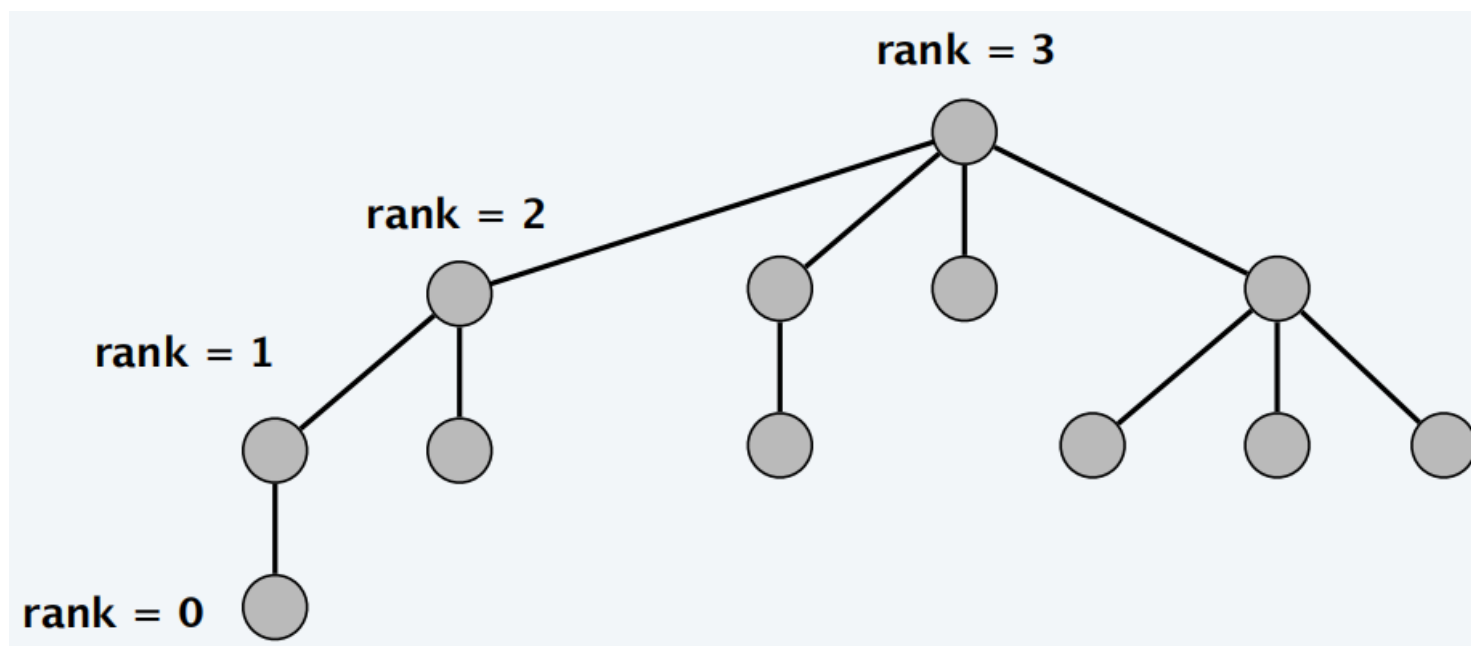
$rank[s] \leftarrow rank[s] + 1.$

Link-by-rank: 性质

性质1: 如果 x 不是根节点, 则 $\text{rank}[x] < \text{rank}[\text{parent}[x]]$

性质2: 如果 x 不是根节点, 则 $\text{rank}[x]$ 永不会再改变

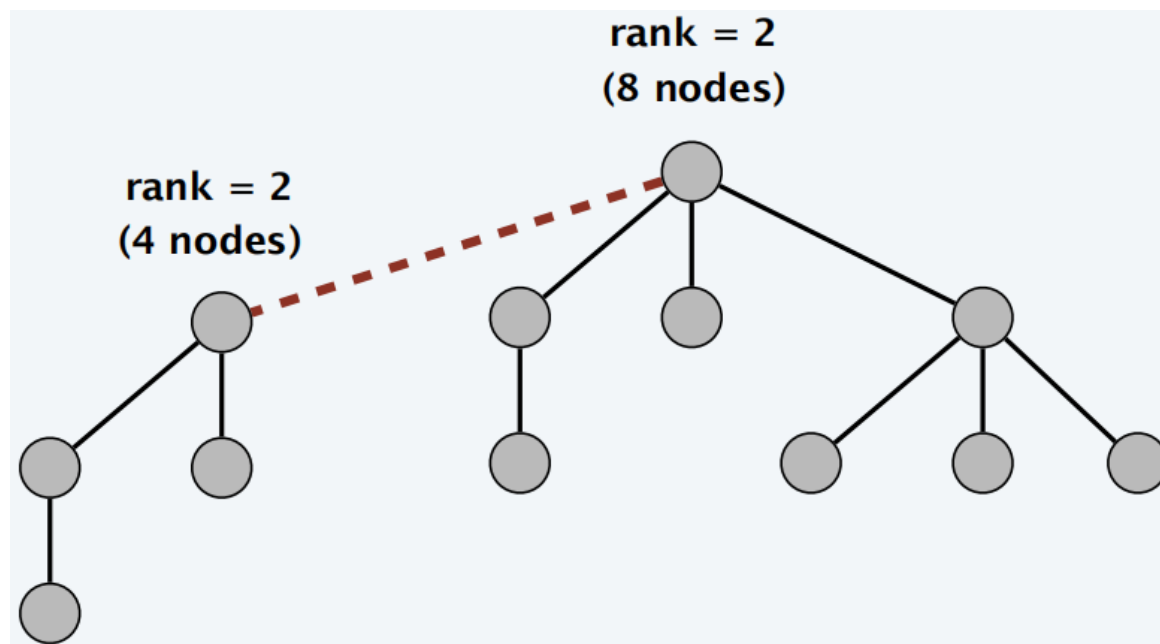
性质3: 如果 $\text{parent}[x]$ 改变, 则 $\text{rank}[\text{parent}[x]]$ 会严格增加



Link-by-rank: 性质

性质4: 任何rank是k的根节点的树至少有 2^k 个点

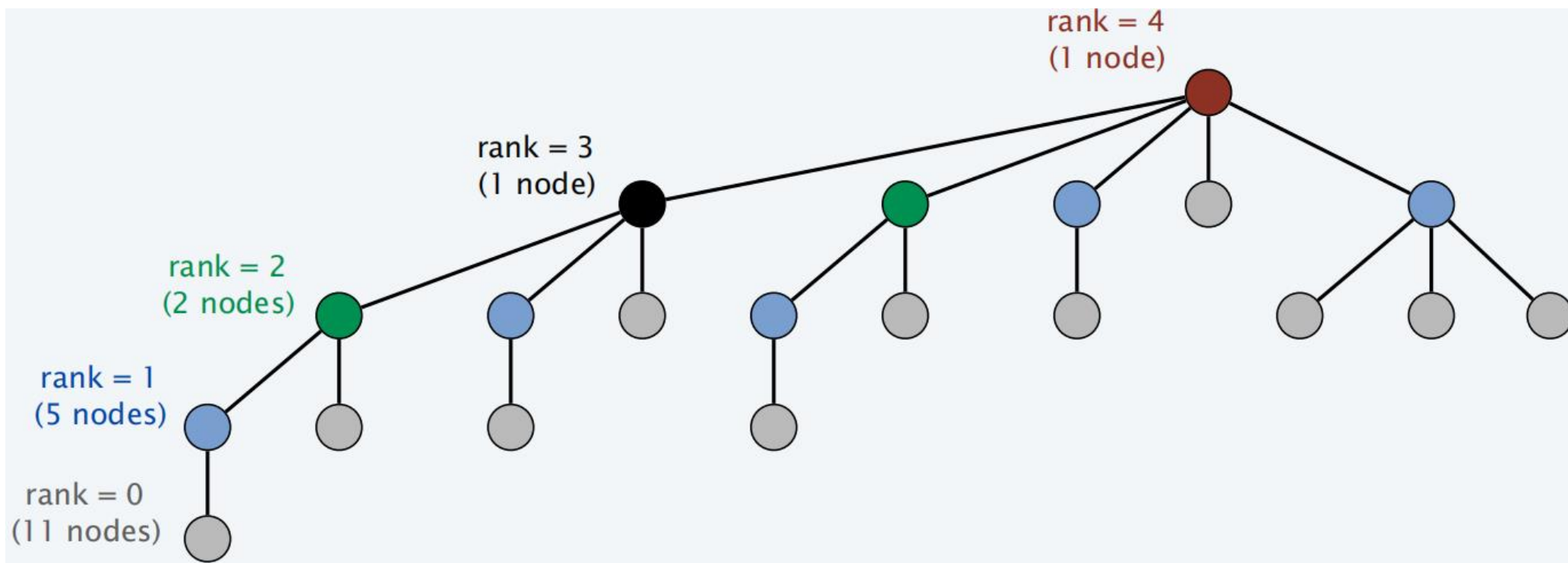
性质5: 节点的最高rank不会高于 $\lceil \log_2 n \rceil$



Link-by-rank: 性质

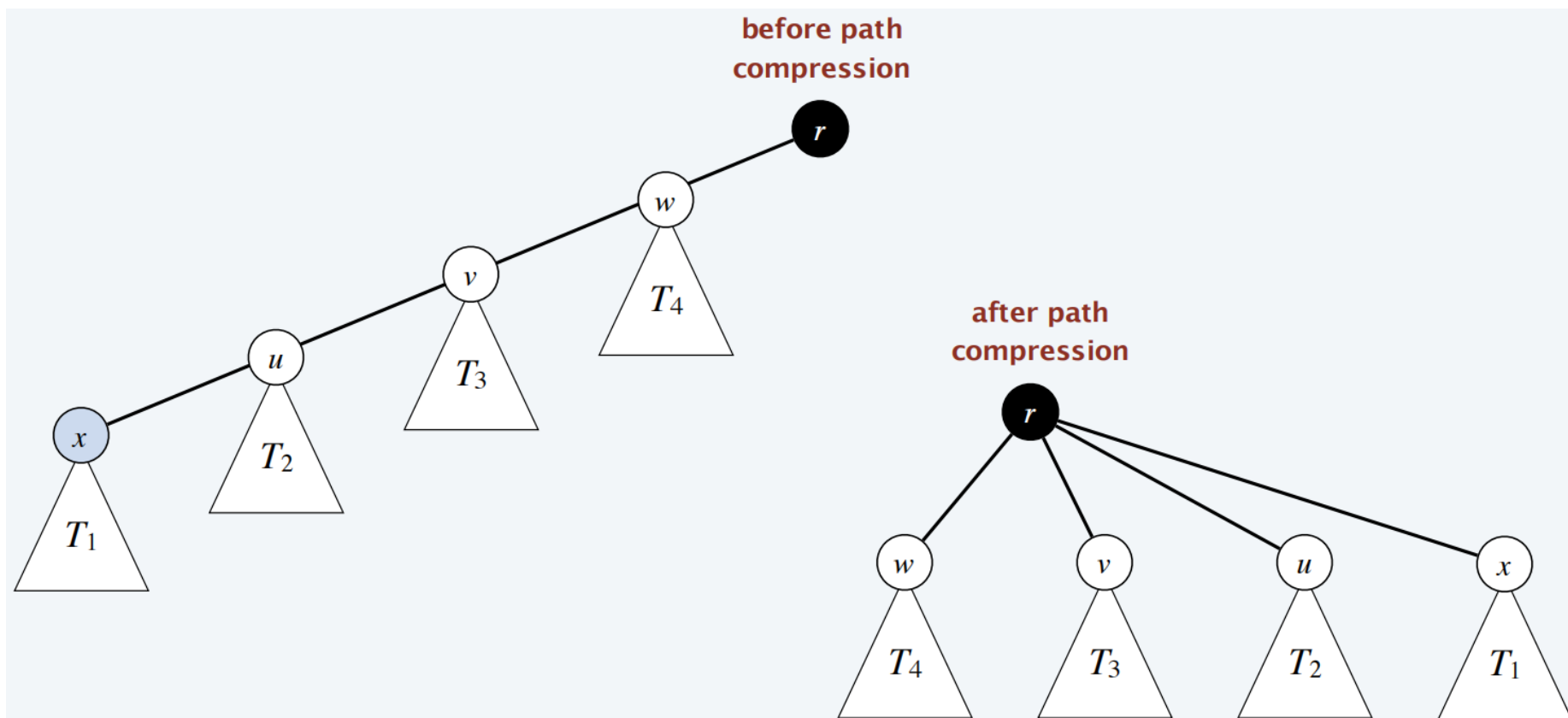
性质6: 对任意整数 k , rank是 k 的点的个数不超过 $n/2^k$

定理: Union和Find操作在最差情况下都需要 $\Theta(\log n)$ 时间, 其中 n 是元素个数



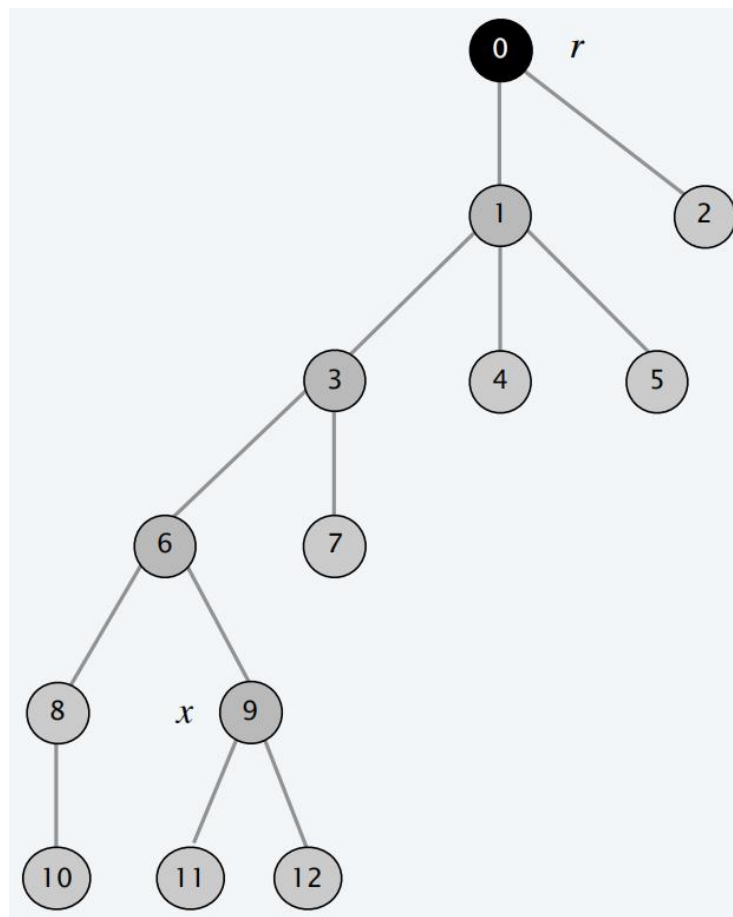
路径压缩Path compression

路径压缩：每次 $\text{find}[x]$ 时， x 所在树的根节点是 r ，把从 x 到 r 路径上的所有点都直接指向 r



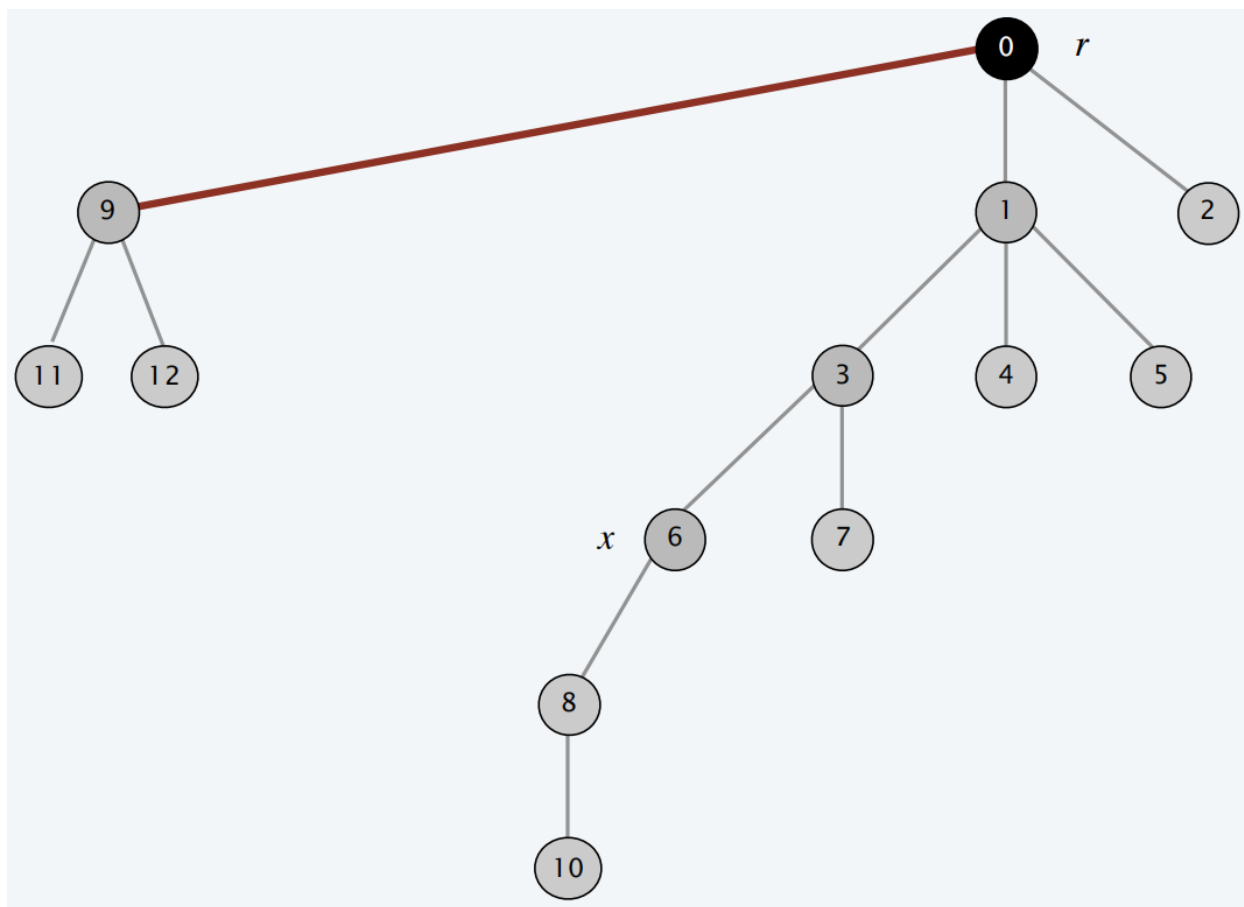
路径压缩Path compression

路径压缩：每次 $\text{find}[x]$ 时， x 所在树的根节点是 r ，把从 x 到 r 路径上的所有点都直接指向 r



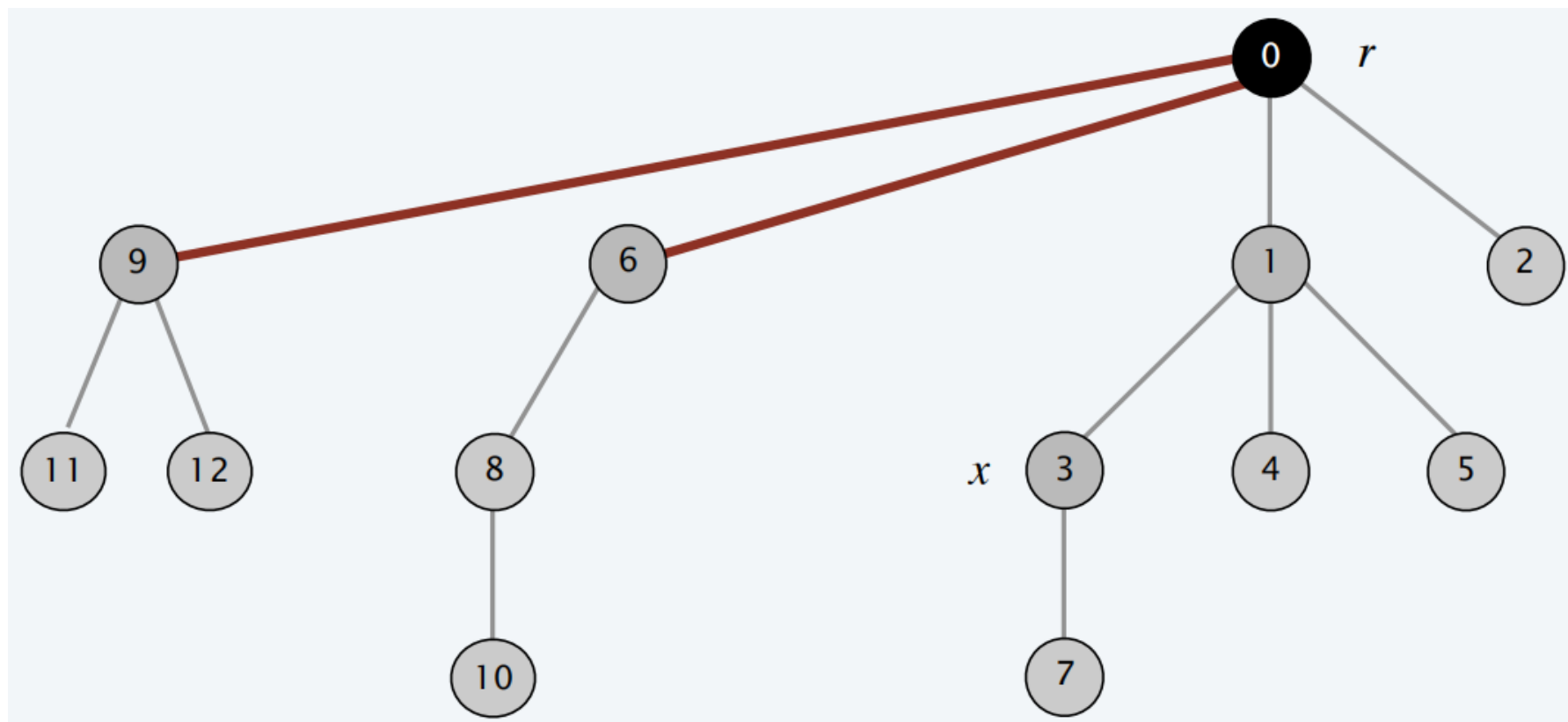
路径压缩Path compression

路径压缩：每次 $\text{find}[x]$ 时， x 所在树的根节点是 r ，把从 x 到 r 路径上的所有点都直接指向 r



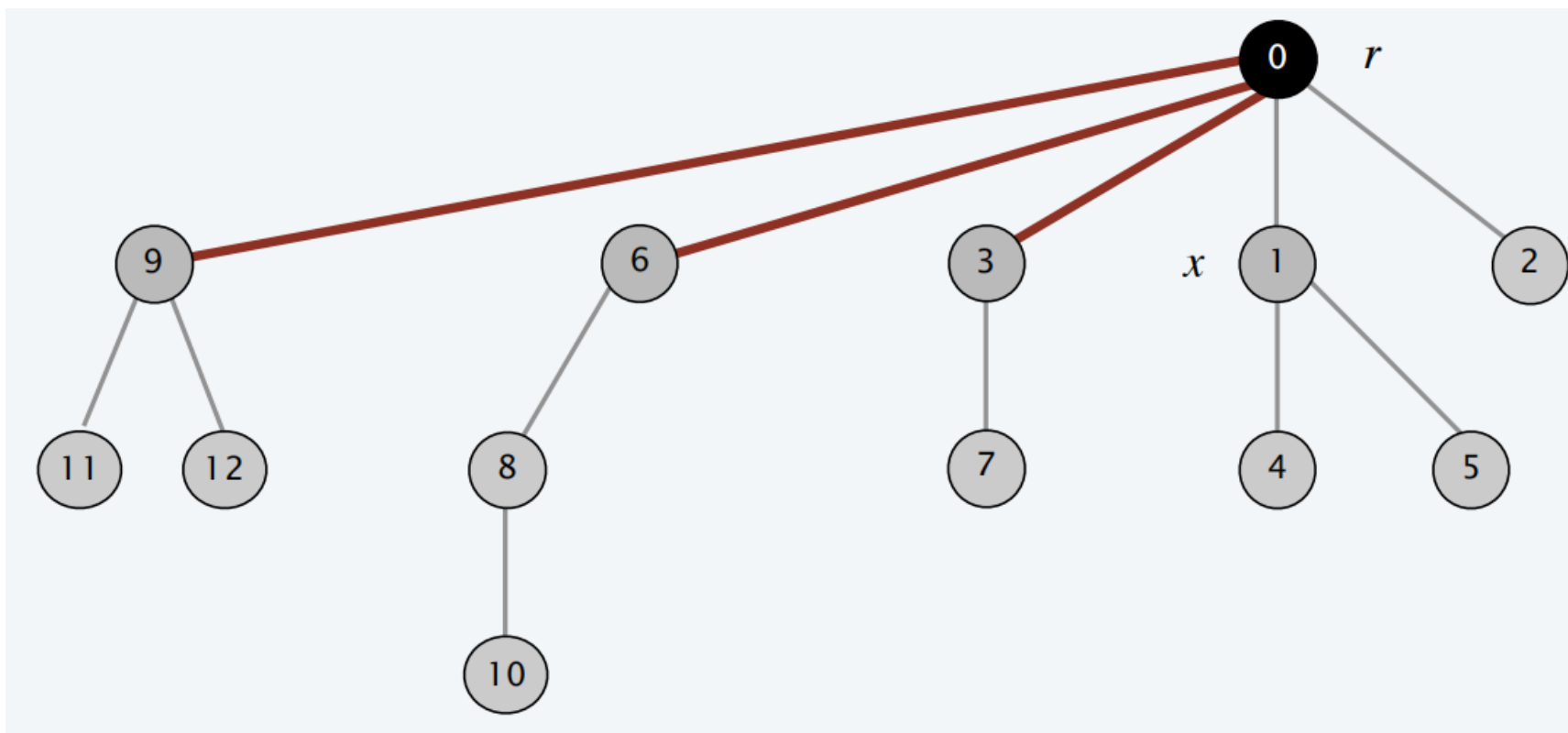
路径压缩Path compression

路径压缩：每次 $\text{find}[x]$ 时， x 所在树的根节点是 r ，把从 x 到 r 路径上的所有点都直接指向 r



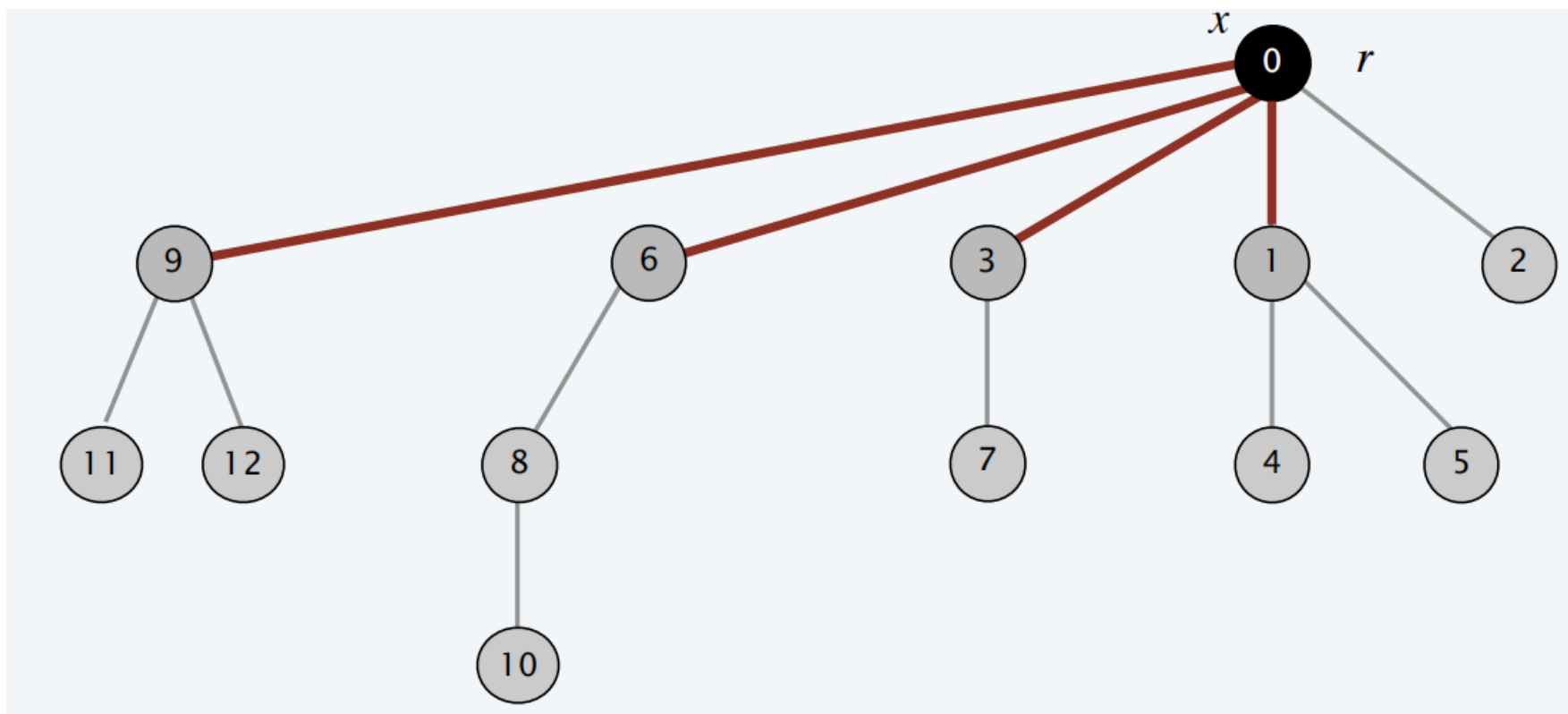
路径压缩Path compression

路径压缩：每次 $\text{find}[x]$ 时， x 所在树的根节点是 r ，把从 x 到 r 路径上的所有点都直接指向 r



路径压缩Path compression

路径压缩：每次 $\text{find}[x]$ 时， x 所在树的根节点是 r ，把从 x 到 r 路径上的所有点都直接指向 r



路径压缩Path compression

路径压缩：每次find[x]时，x所在树的根节点是r，把从x到r路径上的所有点都直接指向r


FIND(x)

IF ($x \neq \text{parent}[x]$)

$\text{parent}[x] \leftarrow \text{FIND}(\text{parent}[x]).$

RETURN $\text{parent}[x].$

this FIND implementation
changes the tree structure (!)



路径压缩Path compression

路径压缩+ naïve linking ([Tarjan–van Leeuwen 1984]) 从一个空的数据结构开始, 考虑 n 个元素和任意组合的 $m \geq n$ 个Make-Set、Union和Find操作, 路径压缩+naïve linking的时间复杂度是 $O(m \log n)$

证明: 不好证

路径压缩+ link-by-rank ([Tarjan–van Leeuwen 1984]) 从一个空的数据结构开始, 考虑 n 个元素和任意组合的 $m \geq n$ 个Make-Set、Union和Find操作, 路径压缩+naïve linking的时间复杂度是 $O(m \log^* n)$

证明: 可以证但是算了

Def. The **iterated logarithm** function is defined by:

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

n	$\lg^* n$
1	0
2	1
[3, 4]	2
[5, 16]	3
[17, 65536]	4
[65537, 2 ⁶⁵⁵³⁶]	5

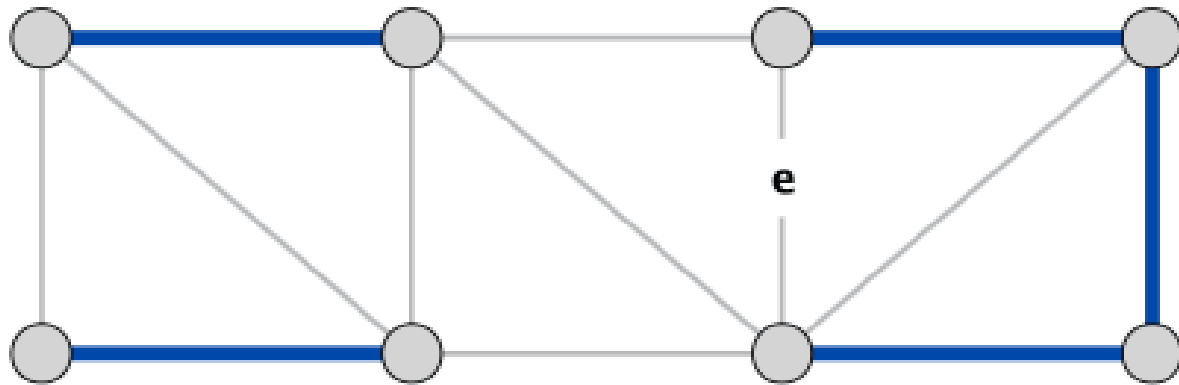
iterated lg function

Kruskal算法

按照代价升序考虑边：

- 若不产生环，便将其添加到树中

定理 Kruskal算法生成一棵MST



Kruskal算法：实现

定理 Kruskal算法可以实现以 $O(m \log m)$ 的时间复杂度运行。

- 按代价将边排序
- 使用并查集数据结构动态维护连接部分

KRUSKAL (V, E, c)

SORT m edges by cost and renumber so that $c(e_1) \leq c(e_2) \leq \dots \leq c(e_m)$.

$T \leftarrow \emptyset$.

FOREACH $v \in V$: **MAKE-SET**(v).

FOR $i = 1$ **TO** m

$(u, v) \leftarrow e_i$.

IF (**FIND-SET**(u) \neq **FIND-SET**(v))  are u and v in
 same component?

$T \leftarrow T \cup \{e_i\}$.

UNION(u, v).  make u and v in
 same component

RETURN T .