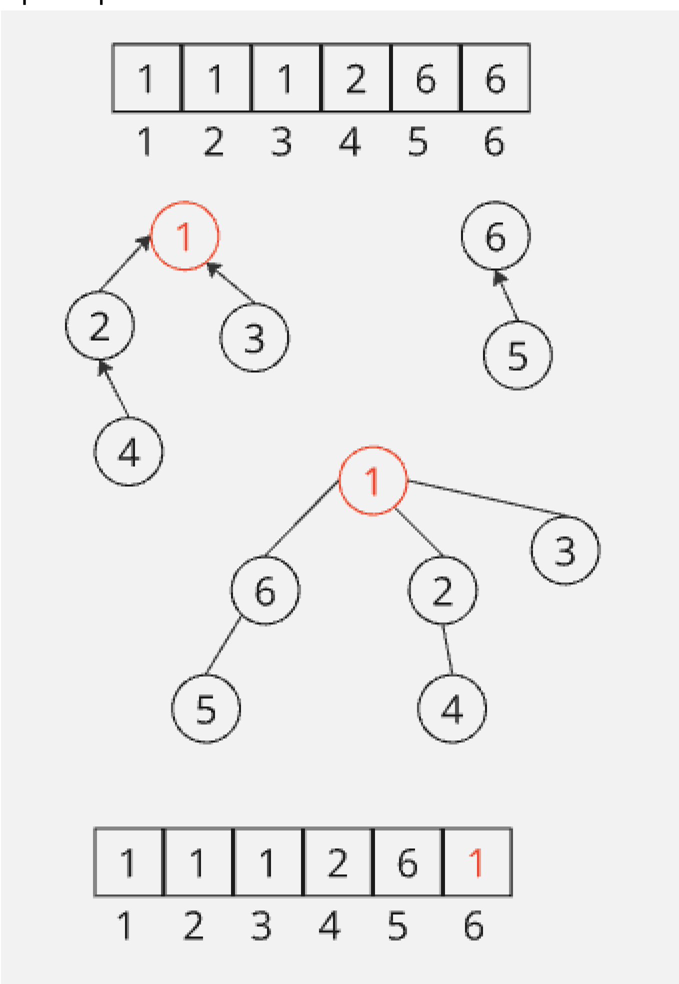
22. Disjont sets (union-find), complexity analysis.

Разделённые множества — это структура данных, которая используется для работы с коллекцией непересекающихся (дизъюнктных) множеств. Она поддерживает два основных операции:

1. **Find**: Определяет, к какому множеству принадлежит элемент.
2. **Union**: Объединяет два множества в одно.

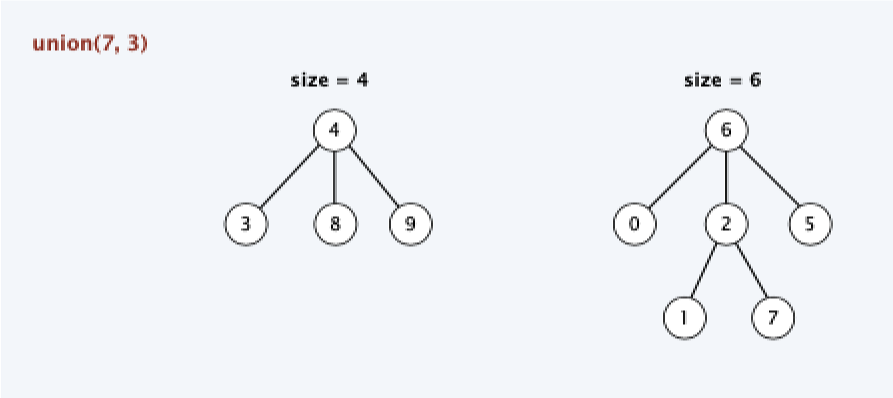
Link-by-size

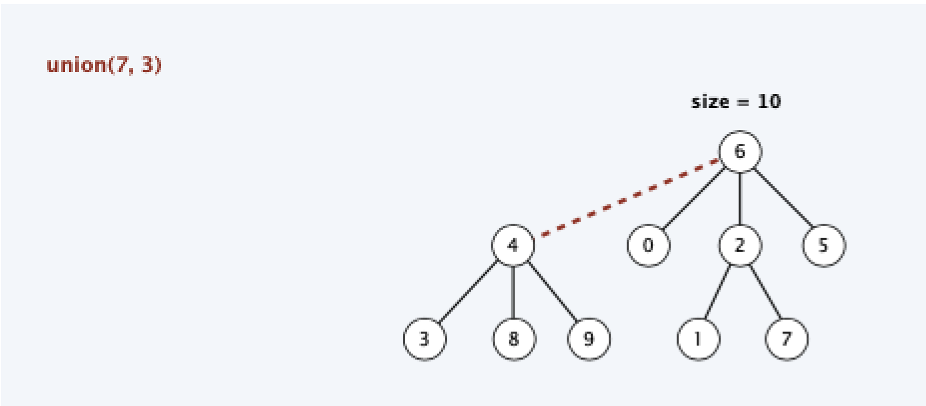
Представителем выбирается элемент из множества, где больше элементов.
Пример:



Имеем 2 множества и хотим их объединить. Представителем выберем единицу. На массиве это сказывается так, что в 6 меняется на 1.

Пример:





Теорема 1:
При использовании link-by-size, для каждого r представителя выполняется следующее неравенство:

$$\text{size}(r) \geq 2^{\text{height}(r)}$$

Доказательство:
Докажем по индукции.
Базовый случай: дерево с одним node имеет size = 1 и height = 0
Предположим, что верно для первых i size-ов. Докажем для s size-а.
Высота r можем увеличиться только если его связывать с деревом, у которого height(r) <= height(s).

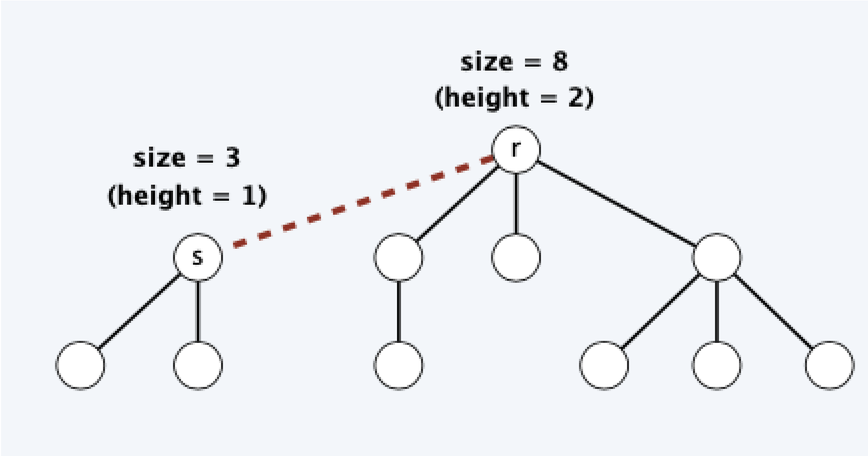
1. случай:

   $$height(r) > height(s)$$

   Тогда

   $$\text{size}'(r) \geq \text{size}(r) \geq (\text{предположение индукции})\ 2^{\text{height}(r)} = 2^{\text{height}'(r)}$$

   где size'() — это размер полученного множества.



Предположим, что size(r) < k. и выполняется size(r) >= 2^(height(r)).
Докажем, что если соединить 2 множества size(r) + size(s) = k, то
size'(r) >= 2^(height'(r)).
Не нарушая общности, будем считать, что size(r) > size(s), т.е. получим, что представителем полученного множества будет r.
тк height(r) > height(s), то высота нового дерева никак не изменится, т.е. height'(r) = height(r). теперь оценим размеры. size'(r) > size(r) > 2^(height(r)) = 2^(height'(r)). те доказали первый случай

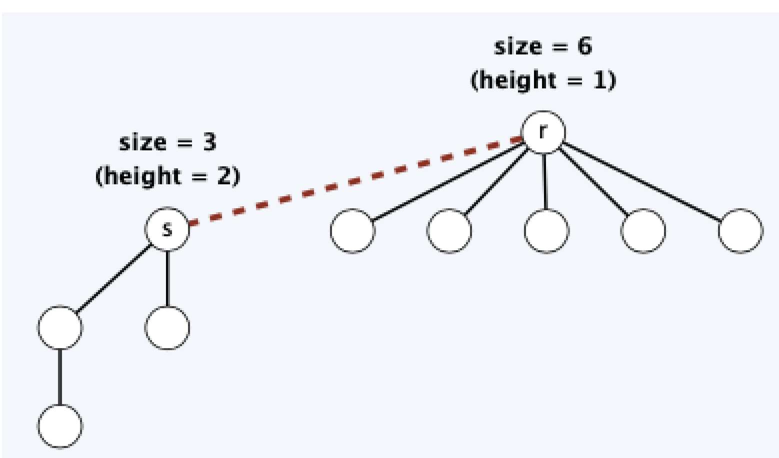
---

2. случай:

$$height(r) \leq height(s)$$

Тогда

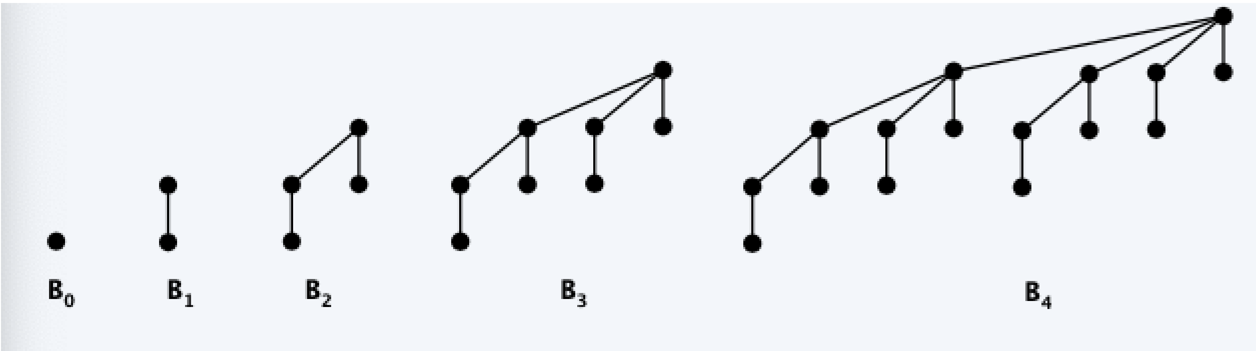
$$size'(r) = size(r) + size(s) \geq \text{(link-by-size) } 2size(s) \geq \text{(предположение индукции) } 2 * 2^{\text{height(s)}}$$



В данном случае высота полученного дерева будет height(s) + 1.

Теорема 2: Используя link-by-size число операций find занимает O(log n) времени, а union - O(1) (если представитель множества уже найден).

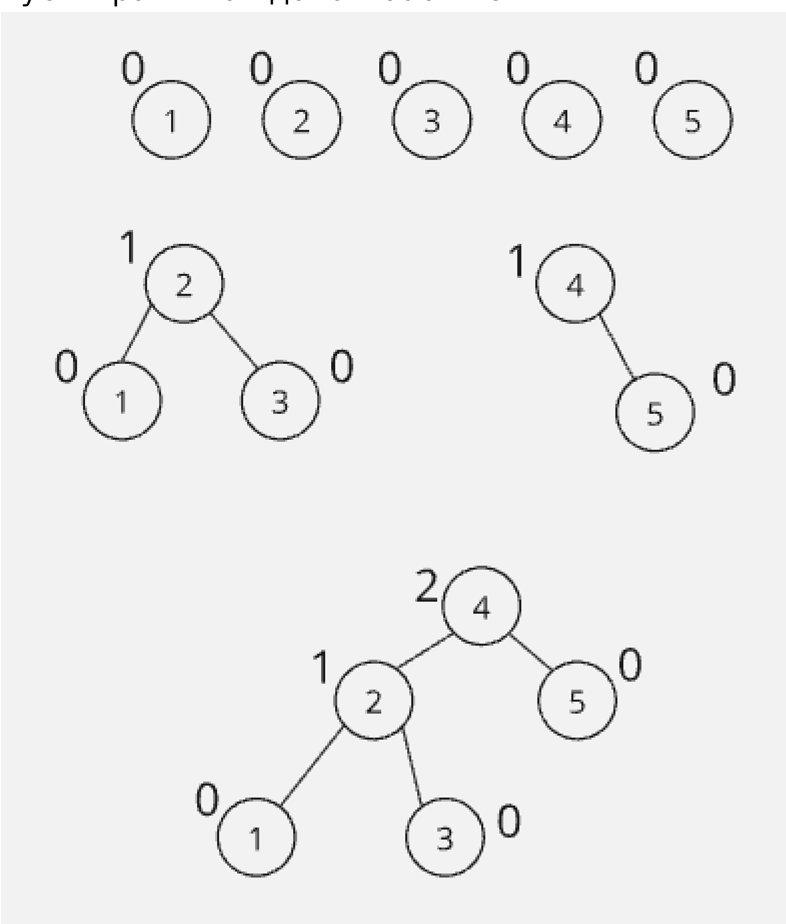
Теорема 3: Используя link-by-size, дерево с n node-ами может иметь высоту = lg n.



Выполяется lg n операций.

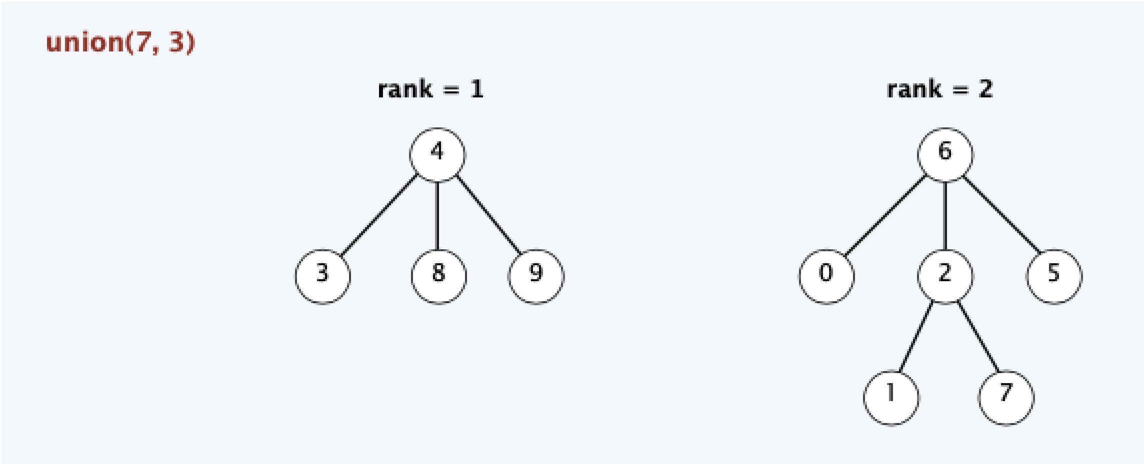
Link-by-rank

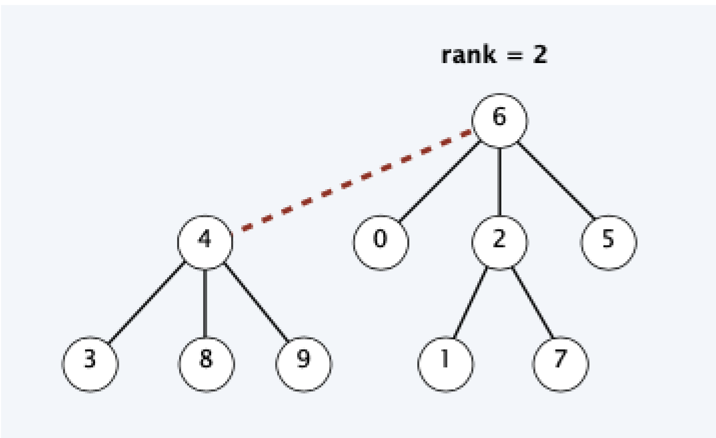
Пусть ранк каждого node = 0.



У всех изначально ранк = 0. потом в каждом множестве какой-то элемент выбирается представителем, и его ранк = 1 (тк сначала все были = 0). Когда уже соединяем 2 множества, вновь выбирается представитель и его ранк увеличивается на 1, только если изначально ранки были равны.

Пример:





Ранк нового дерева не меняется, тк он изначально был больше.
В данном случае, ранк = высоте (это не всегда так).

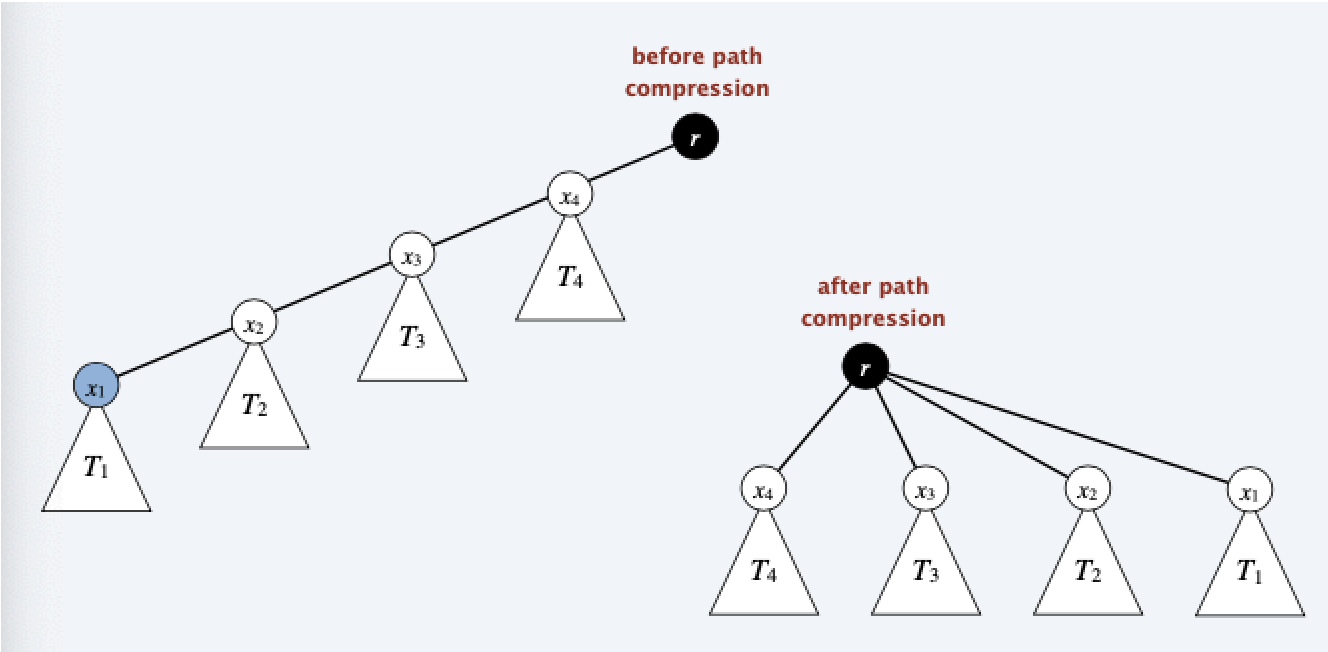
Свойства

1. Если x не представитель, то rank(x) < rank(parent(x))
   Док: Ранк k можно получить соединив два дерева с ранками k-1
2. Если x не представитель, то rank(x) ниогда больше не изменится
3. Если parent(x) меняется, то rank(parent(x)) увеличивается
4. Если у какого-то узла ранк = k, то число узлов в поддереве этого узла >= 2^k
   Док: докажем индукцией. При k = 0 очевидно. Предположим, что верно при k-1, докажем для k. Тк node с ранком = k можно получить лишь соединив 2 дерева с ранками k-1, и в обоих деревьях количество узлов >= 2^(k-1), соединив деревья получим 2^k
5. Число действий <= [lg n].
6. Для любого целого r>=0, есть <=n/(2^r) node-ов с ранком r.

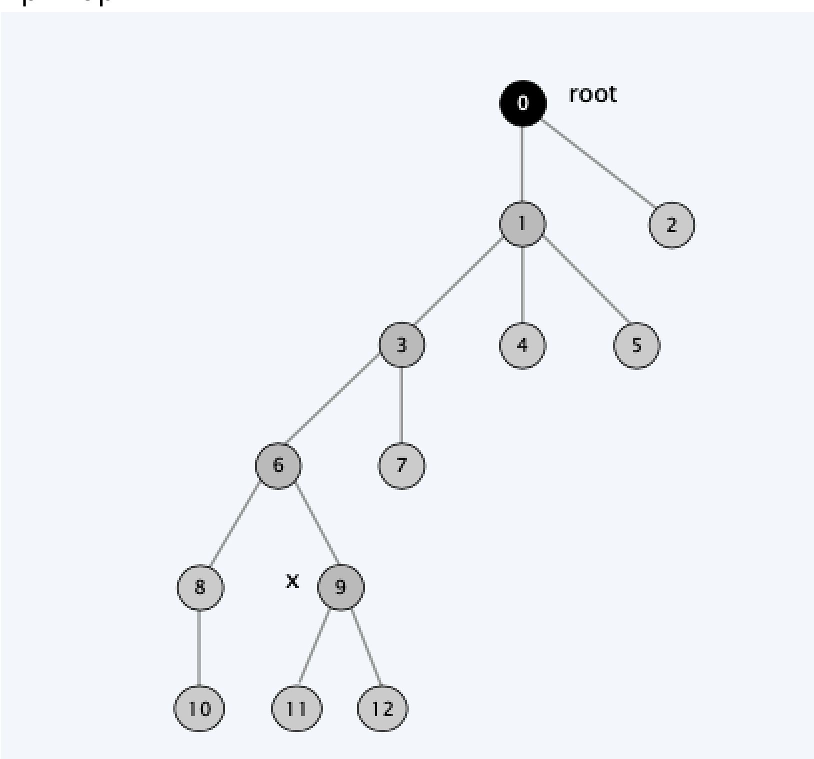
Теорема 2 также здесь справедлива.

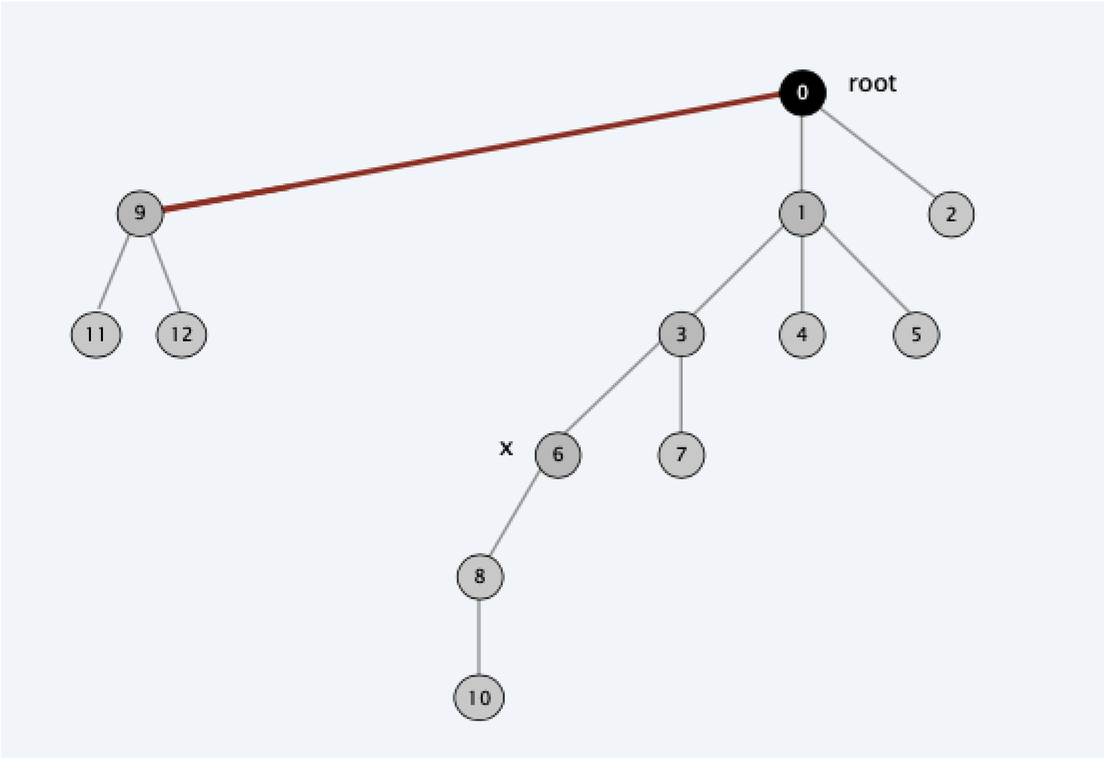
Path compression

После нахождения представителя r дерева, содержащего x, поменяем родительский указатель всех node-ов на пути прямо на r.

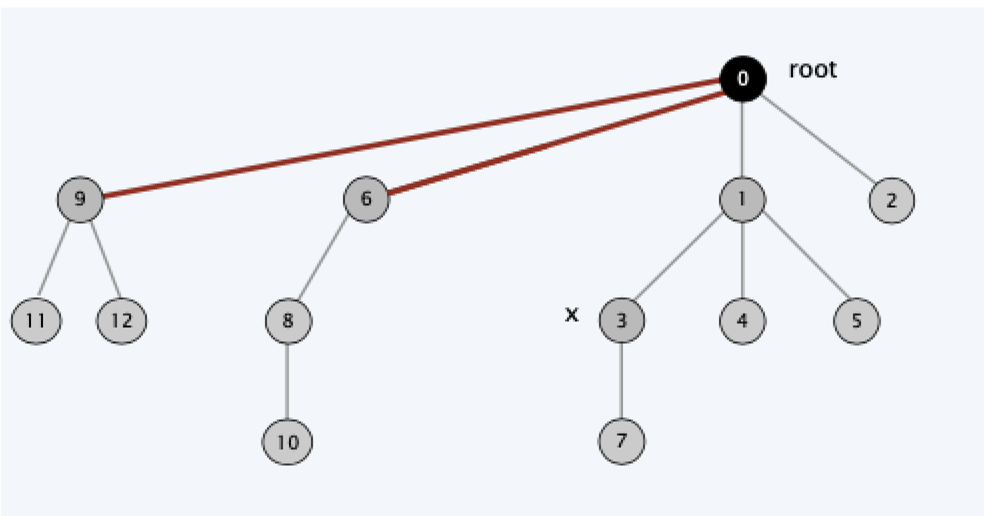


Пример:

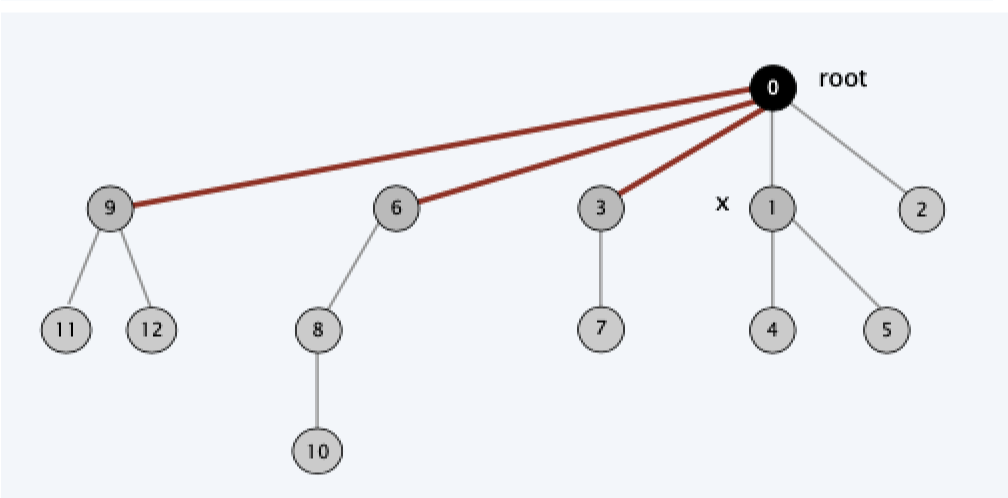




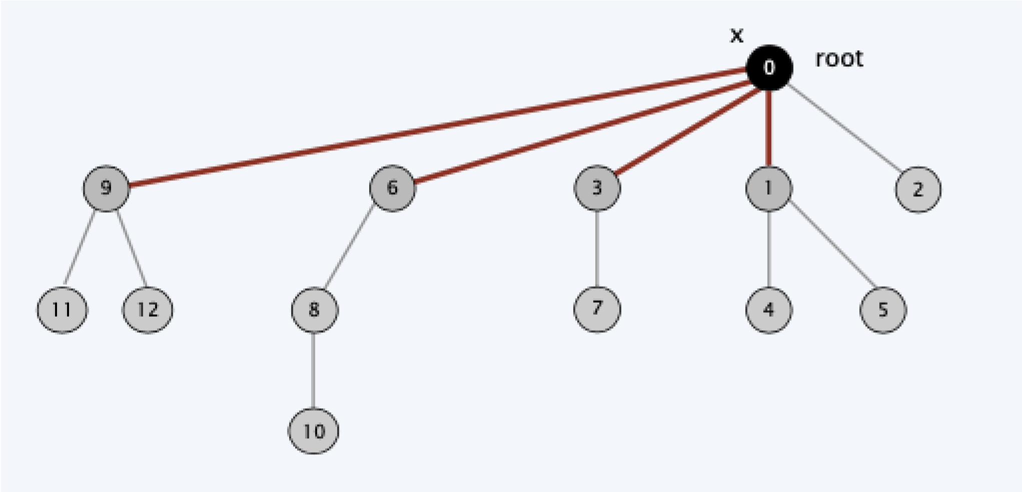
root
0
9
1
2
11
12
3
4
5
x
6
7
8
10



root
0
9
6
1
2
11
12
8
x
3
4
5
10
7



root
0
9
6
3
x
1
2
11
12
8
7
4
5
10



При link-by-rank path compression никак не меняет ранки.
Теперь ранк это не высота.
Все те свойства также сохраняются.

Итеративный логарифм - то число логарифмов, которое нужно применить на n, чтобы получить 1.

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

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

iterated lg function

Кроме того, итеративный логирифм - это обратная функция итеративной степени
Итеративная степень:

$$2 \uparrow n = 2^{2^{2^{\cdots 2}}} \quad \text{(n раз)}$$

$$2 \uparrow 0 = 1$$

$$2 \uparrow 1 = 2$$

$$2 \uparrow 2 = 4$$

$$2 \uparrow 3 = 16$$

Теорема:
Если на произвольном union find последовательно выполнить n операций find,

то число операций не будет превосходить

$$n \cdot \lg^* n$$

Те амортизировано это константа.

Не знаю это надо или нет, то пусть будет

Theorem. [Fischer 1972] Link-by-size with path compression performs any intermixed sequence of $m \geq n$ FIND and $n - 1$ UNION operations in $O(m \log \log n)$ time.

Theorem. [Hopcroft-Ullman 1973] Link-by-size with path compression performs any intermixed sequence of $m \geq n$ FIND and $n - 1$ UNION operations in $O(m \log^* n)$ time.

Theorem. [Tarjan 1975] Link-by-size with path compression performs any intermixed sequence of $m \geq n$ FIND and $n - 1$ UNION operations in $O(m\, \alpha(m, n))$ time, where $\alpha(m, n)$ is a functional inverse of the Ackermann function.

Ackermann function. A computable function that is not primitive recursive.

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

Inverse Ackermann function.

$$\alpha(m,n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) \geq \log_2 n\}$$

Definition.

$$\alpha_k(n) = \begin{cases} 1 & \text{if } n = 1 \\ \lceil n/2 \rceil & \text{if } k = 1 \\ 1 + \alpha_k(\alpha_{k-1}(n)) & \text{otherwise} \end{cases}$$