



# Disjoint-set DS

- Структура непересекающихся множеств это структура данных, которая позволяет администрировать множество элементов, разбитое на непересекающиеся подмножества.
- Каждому подмножеству назначается его представитель — элемент этого подмножества.
- Абстрактная структура данных определяется множеством трёх операций: { `U n i o n` , `F i n d` , `M a k e S e t` }



# MakeSet

- MakeSet(X) — вносит в структуру новый элемент X, создает для него множество размера 1 из самого себя.
- // Make Set -- Create a singleton set containing vertex x

```
template <class Element>
inline void make_set(Element x)
{
    put(parent, x, x);
    typedef typename property_traits<RankPA>::value_type R;
    put(rank, x, R());
}
```



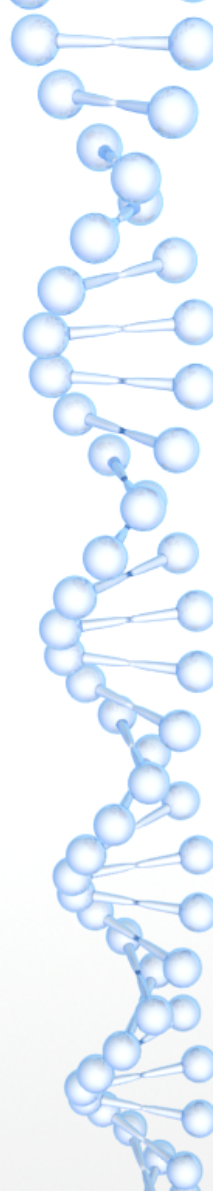
# Find

- Find(X) — возвращает идентификатор множества, которому принадлежит элемент X. В качестве идентификатора выбирается один элемент из этого множества — представитель множества.
- Гарантируется, что для одного и того же множества представитель будет возвращаться один и тот же.
- ```
// Find-Set - returns the  
Element representative of the set  
// containing Element x and  
applies path compression.  
template <class Element>  
inline Element  
find_set(Element x)  
{  
    return rep(parent, x);  
}
```



# Unite

- $\text{Unite}(X, Y)$  — объединяет два множества, в которых лежат элементы  $X$  и  $Y$ , в одно новое.
- ```
// Union-Set - union the two sets containing vertex x and y
inline void union_set(Element x, Element y)
{
    link(find_set(x), find_set(y));
}
```



```
MakeSet(1);  
MakeSet(2);  
MakeSet(3);  
MakeSet(4);  
MakeSet(5);
```

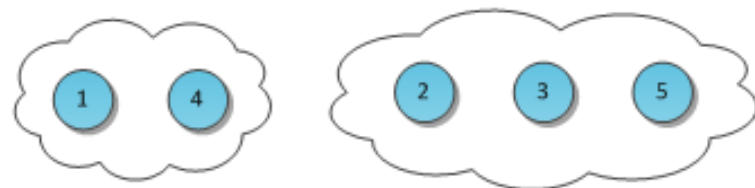
```
Find(4) = 4
```

```
Unite(1, 4);  
Unite(3, 5);
```

```
Find(4) = 4  
Find(1) = 4  
Find(2) = 2
```

```
Unite(5, 2);
```

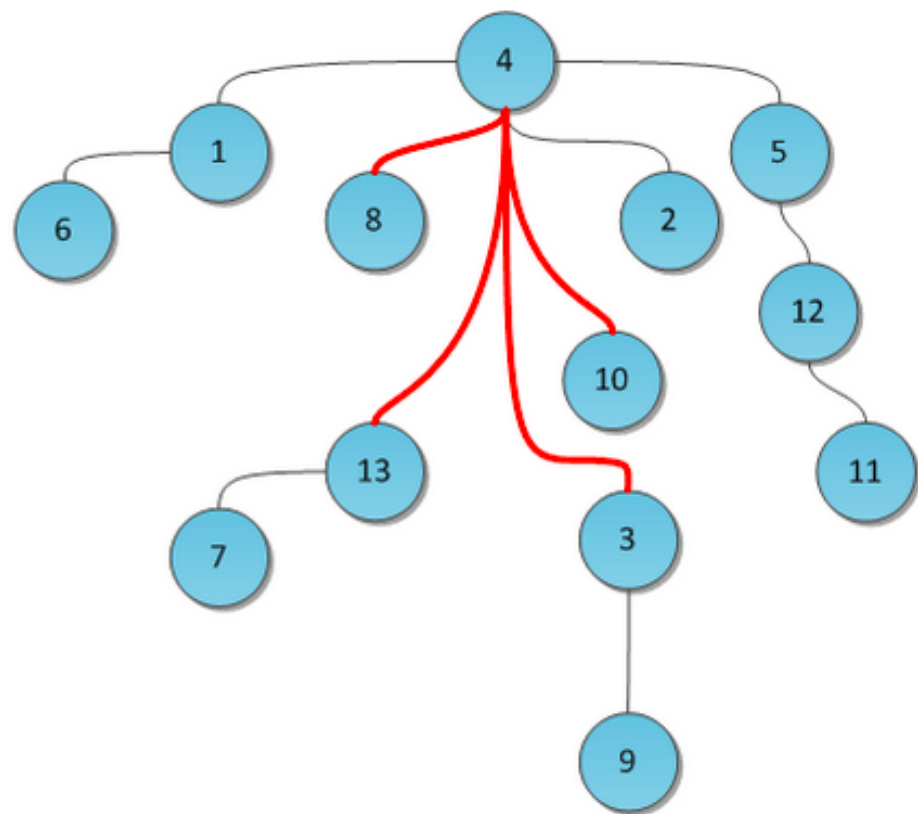
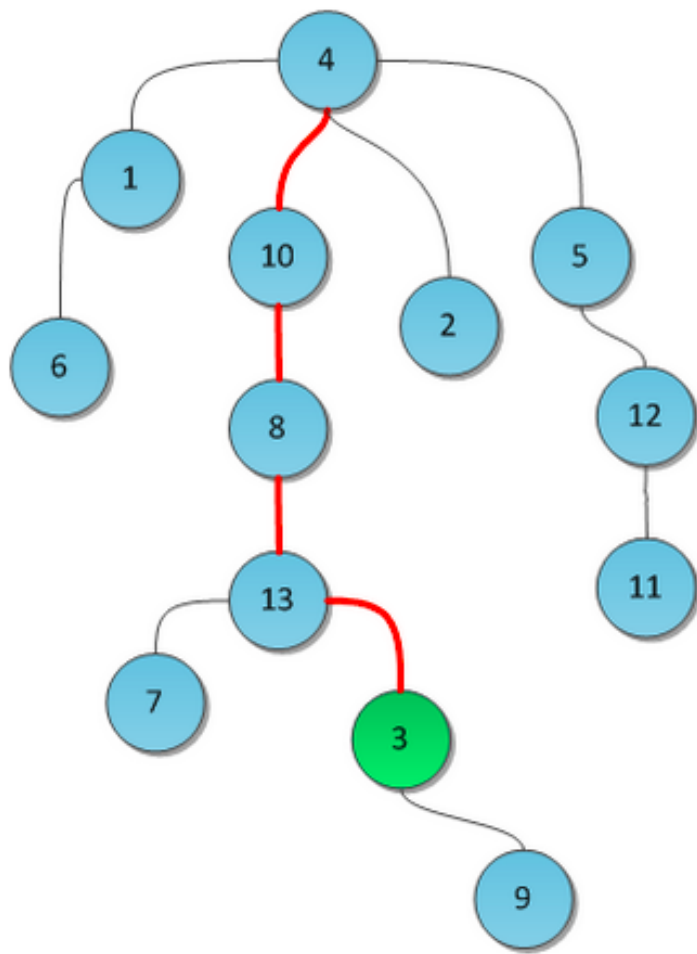
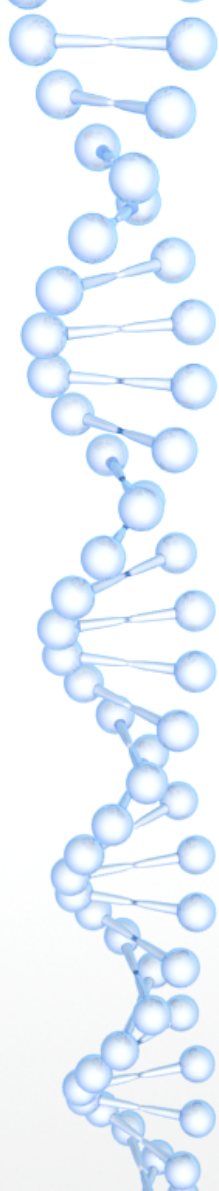
```
Find(5) = 2  
Find(3) = 2
```





# Path Compression

- Сжатие путей (path compression) подразумевает следующее: после того, как представитель таки будет найден, мы для каждой вершины по пути от  $X$  к корню изменим предка на этого самого представителя. То есть фактически переподвесим все эти вершины вместо длинной ветви непосредственно к корню. Таким образом, реализация операции Find становится двухпроходной.

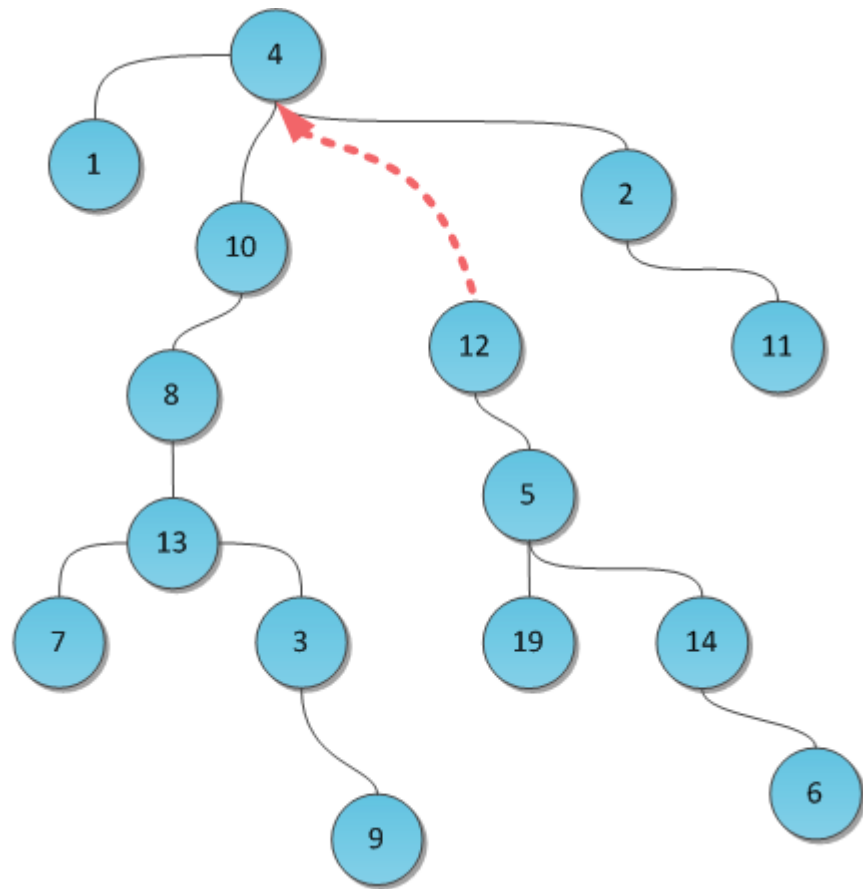
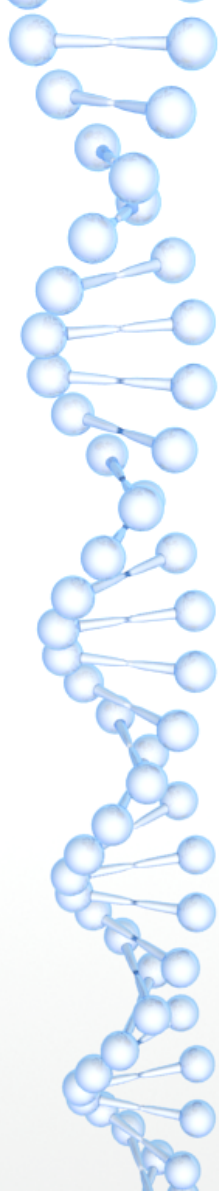




# Массив Rank

- В нем для каждого дерева будет храниться верхняя граница его высоты — то есть длиннейшей ветви в нем. Поэтому для каждого корня в массиве Rank будет записано число, гарантированно больше или равное высоте его дерева.





# Быстродействие

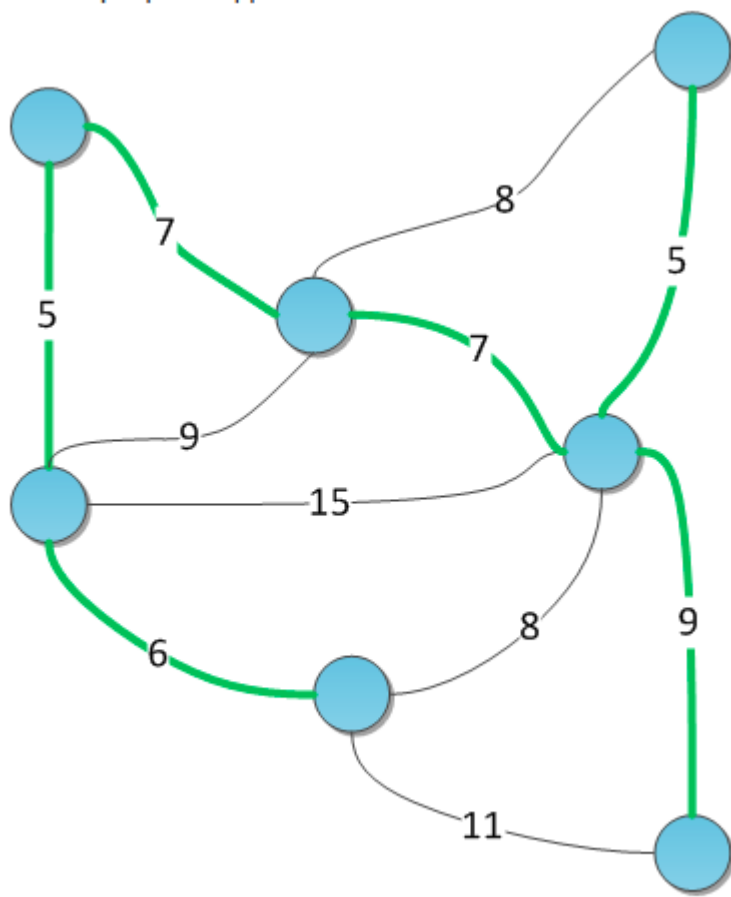
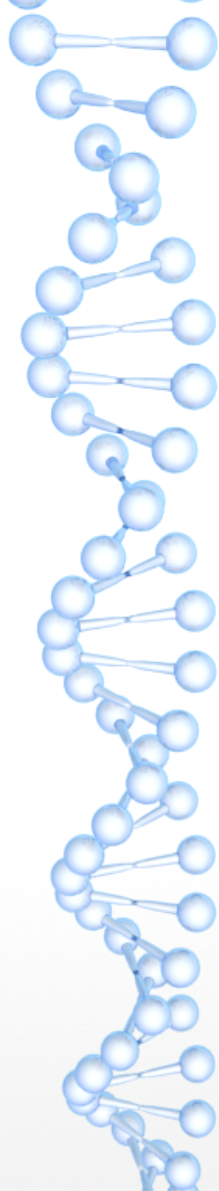
- Время работы как Find, так и Unite на лесе размера  $N$  есть  $O(\alpha(N))$ .
- Под  $\alpha(N)$  в математике обозначается обратная функция Аккермана, то есть, функция, обратная для  $f(N) = A(N, N)$ . Вообще, для всех мыслимых практических значений  $N$  обратная функция Аккермана от него не превысит 5. Поэтому её можно принять за константу и считать  $O(\alpha(N)) \cong O(1)$ .
- MakeSet( $X$ ) —  $O(1)$ .
- Find( $X$ ) —  $O(1)$  амортизированно.
- Unite( $X, Y$ ) —  $O(1)$  амортизированно.
- Расход памяти —  $O(N)$ .

$n \backslash m$	0	1	2	3	4
0	1	2	3	5	13
1	2	3	5	13	65533
2	3	4	7	29	$2^{65536} - 3$
3	4	5	9	61	$2^{2^{65536}} - 3$
4	5	6	11	125	$2^{2^{2^{65536}}} - 3$



# Остов минимального веса

- Дан неориентированный связный граф со взвешенными ребрами. Выкинуть из него некоторые ребра так, чтобы в итоге получилось дерево, причем суммарный вес ребер этого дерева должен быть наименьшим.
- Одно из известных мест, где встает эта задача (хотя и решается иначе) — блокирование избыточных связей в Ethernet-сети для избегания возможных циклов пакетов. Протоколы, созданные с этой целью, широко известны, причем половина серьезных модификаций в них сделана Cisco.



# Least Common Ancestor, LCA

