**Disjoint sets**

* **Representations**
* **Union Algorithm**
* **Find Algorithm**

**Disjoint Set-Definition**

* A [disjoint-set data structure](http://en.wikipedia.org/wiki/Disjoint-set_data_structure) is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets.
* A disjoint-set data structure is a structure that maintains a collection S1, S2, S3, …, Sn of dynamic disjoint sets
* It is a group of sets where no item can be in more than one set.
* It is also called an **union-find data structure** as it support union and find operations on subsets.
* Two **sets** are said to be **disjoint sets** if they have no element in common.
* Equivalently, two **disjoint sets** are **sets** whose intersection is the empty **set**.
* For **example**, {1, 2, 3} and {4, 5, 6} are **disjoint sets**, while {1, 2, 3} and {3, 4, 5} are not **disjoint**.

S=1, 2, 3, 4, 5, 6

S1= {1}

S2= {2}

S3= {3}

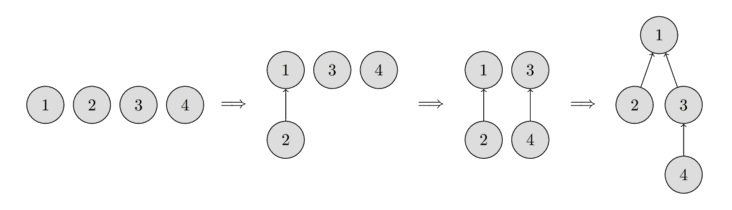
S4= {4}

S5= {5}

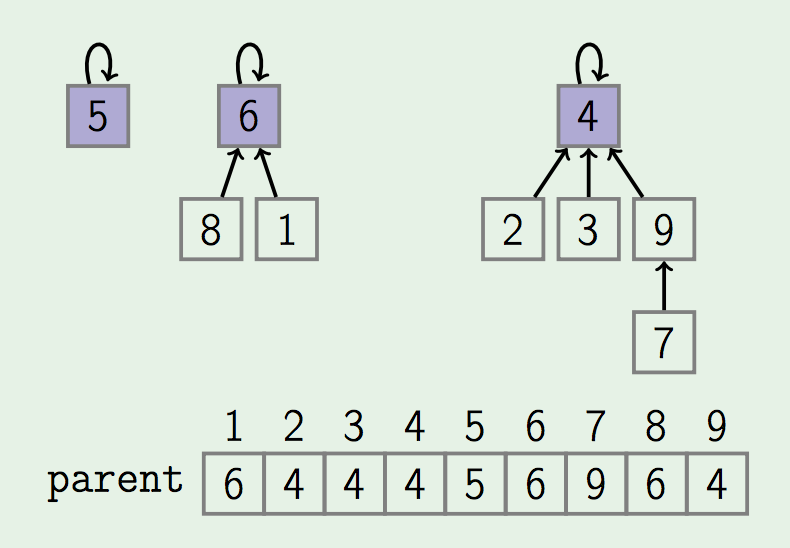
S6= {6}

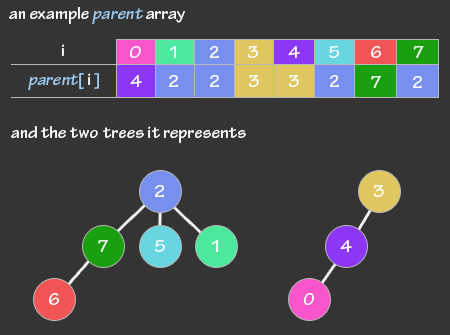
**Representations**

* We will store the sets in the form of trees: each tree will correspond to one set.
* In a data structure of disjoint sets every set contains a **representative**, which is one member of the set.
* The root of the tree will be the representative/leader of the set.
* In the following image you can see the representation of such trees.



* In the beginning, every element starts as a single set, therefore each vertex is its own tree.
* Then we combine the set containing the element 1 and the set containing the element 2.
* Then we combine the set containing the element 3 and the set containing the element 4.
* And in the last step, we combine the set containing the element 1 and the set containing the element 3.
* For the implementation this means that we will have to maintain an array parent that stores a reference to its immediate ancestor in the tree.
* All the information about the sets of elements will be kept in an array parent.





**Operations**

* **make\_set(v)**
* creates a new set consisting of the new element **v**
* **union\_sets(a, b)**
  + merges the two specified sets (the set in which the element **a** is located, and the set in which the element **b** is located)
* **find\_set (v)**
  + Returns the representative (also called leader) of the set that contains the element **v**.
  + This representative is an element of its corresponding set.
  + It is selected in each set by the data structure itself (and can change over time, namely after **union\_sets** calls).
  + This representative can be used to check if two elements are part of the same set or not. **a** and **b** are exactly in the same set, if **find\_set(a) == find\_set(b)**. Otherwise they are in different sets.

**Find ():** It is used to find in which subset a particular element is in and returns the representative of that particular set.

**Union ():** It merges two different subsets into a single subset and representative of one set becomes representative of other.

* To combine two sets (operation union\_sets (a, b)), we first find the representative of the set in which a is located, and the representative of the set in which b is located.
* If the representatives are identical, that we have nothing to do, the sets are already merged.
* Otherwise, we can simply specify that one of the representatives is the parent of the other representative - thereby combining the two trees.
* Finally the implementation of the find representative function (operation find\_set (v)): we simply climb the ancestors of the vertex v until we reach the root, i.e. a vertex such that the reference to the ancestor leads to itself. This operation is easily implemented recursively.

**void make\_set (int v)**

**{**

**parent[v] = v;**

**}**

**int find\_set (int v)**

**{**

**if (v == parent[v])**

**return v;**

**return find\_set (parent[v]);**

**}**

**void union\_sets (int a, int b) a=8,b=2**

**{**

**a = find\_set (a);//6**

**b = find\_set (b);//4**

**if (a! = b)**

**parent[b] = a;//parent[4]=6**

**}**