Disjoint Set Union (C++)

Sets can make friends too!

#csspree Online

Disjoint Set

Disjoint Set

`Group of sets with no common elements.`

Disjoint Set

Group of sets with no common elements.

```
{1, 3, 4}  {5, 6}  {2, 7, 11}

     {12, 15}  {9, 13}
```

Disjoint Set

Group of sets with no common elements.

{1, 3, 4} {5, 6} {2, 7, 11}

{12, 15} {9, 13}

No sets overlap

Valid √

Disjoint Set

`Group of sets with no common elements.`

{1, 3, 4} {5, 6} {2, 7, 11}

{12, 15} {9, 13}

No sets overlap

Valid √

{1, 3, 4} {5, 6} {2, 9, 11}

{4, 15} {9, 13}

Disjoint Set

`Group of sets with no common elements.`

`{1, 3, 4}  {5, 6}  {2, 7, 11}`

`{12, 15}  {9, 13}`

No sets overlap

Valid √

`{1, 3, 4}  {5, 6}  {2, 9, 11}`

`{4, 15}  {9, 13}`

Sets have common element(s)

Invalid ×

Disjoint Set Union

Disjoint Set Union

`Things it does:`

Disjoint Set Union

```
Things it does:

1. Find(u) // which set u belongs to
```

Disjoint Set Union

```
Things it does:

1. Find(u) // which set u belongs to
2. Union(u, v) // merge sets of u and v
```

Disjoint Set Union

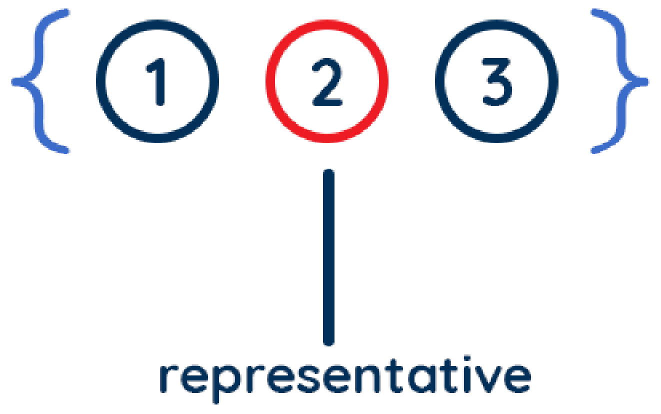
```
Things it does:

1. Find(u) // which set u belongs to
2. Union(u, v) // merge sets of u and v



Some trivial operations...
3. MakeSet(u) // initialize u as a set
4. IsSameSet(u, v) // check if u and v belongs to same set (isFriend(u, v))
```

Disjoint Set Union

```
How it does:
```

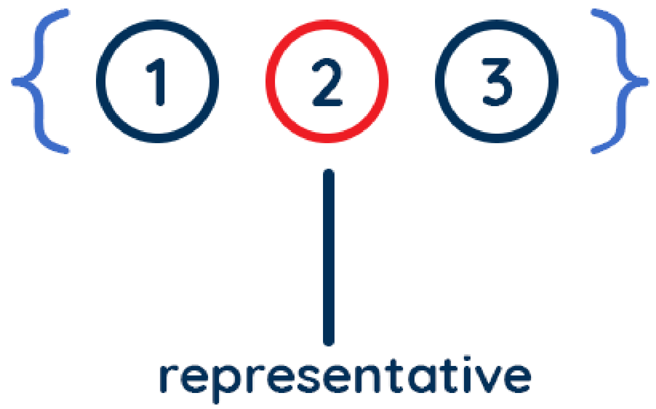
Disjoint Set Union

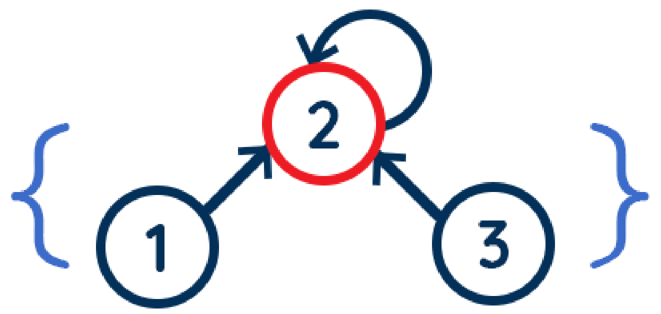
```
How it does:

1. [Identification] : Every set has a representative
                      (one of the elements of the set)
```



Disjoint Set Union

```
How it does:

1. [Identification] : Every set has a representative
                     (one of the elements of the set)
2. [Relation] :       Elements are connected via parent-child
                     relation
```



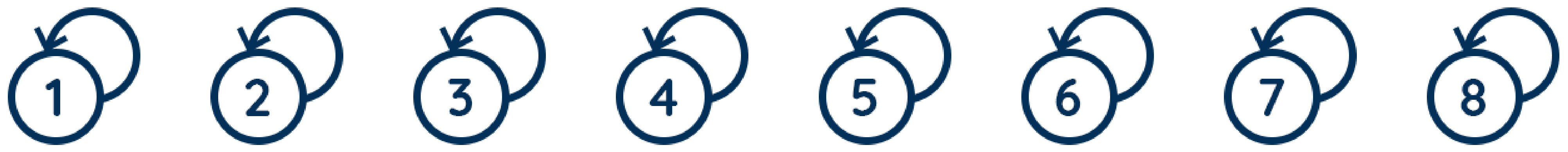


2 is parent of 1, 3 and itself

Simulation

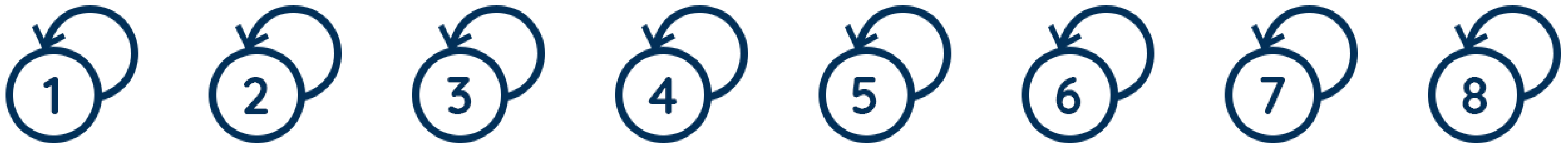
Simulation

Simulation



initially, every element is parent of itself

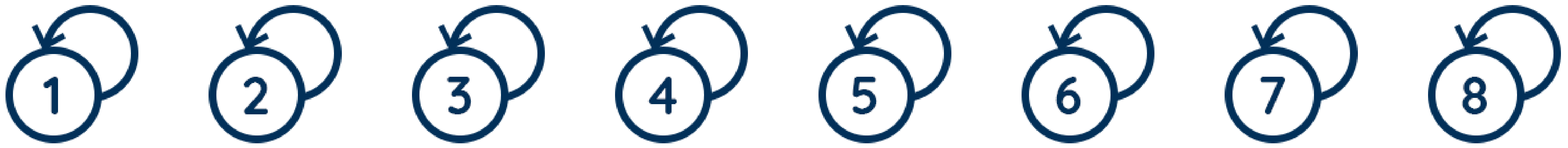
Simulation



parent[] =

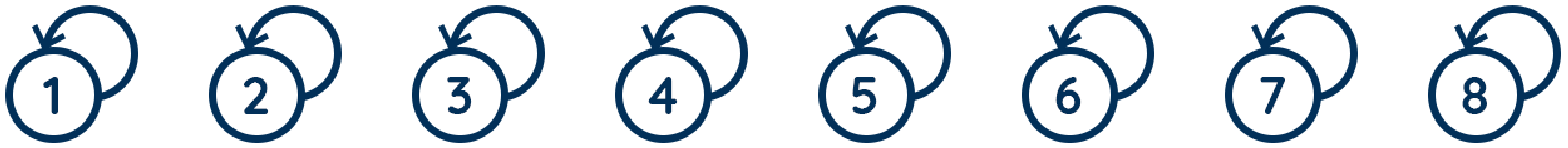
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

Simulation



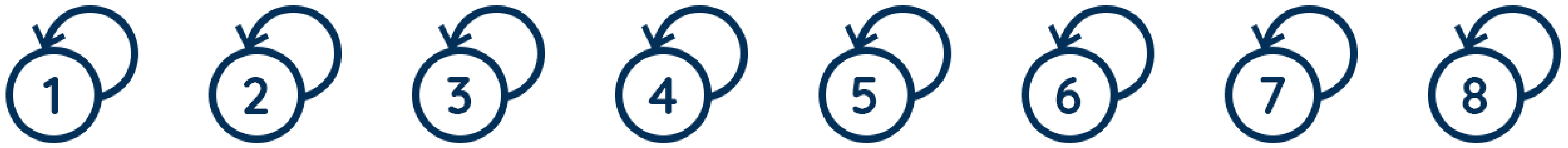
```
Union(1, 5)
```

Simulation



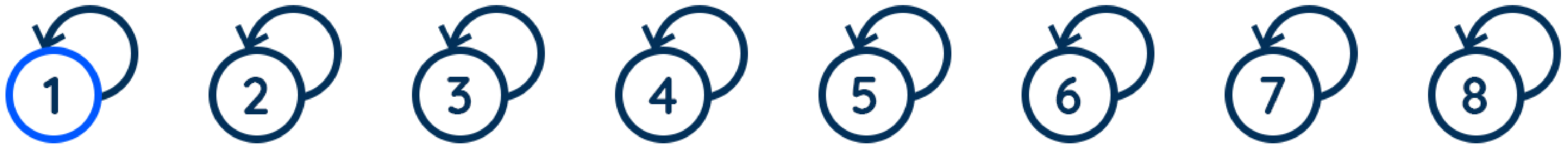
```
Union(1, 5)

p = Find(1) // find parent/representative of 1’s set
q = Find(5) // find parent/representative of 5’s set
```

Simulation



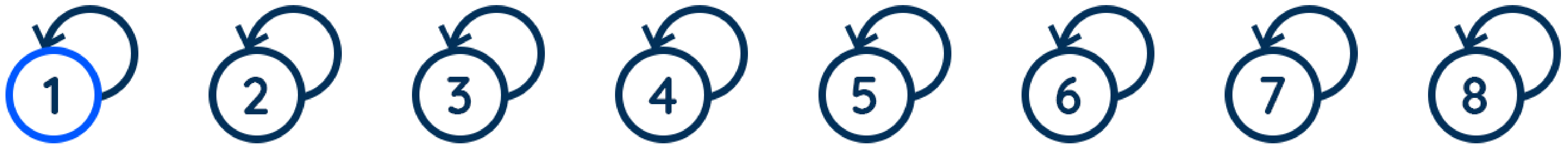
```
Union(1, 5)

p = Find(1) // find parent/representative of 1’s set
q = Find(5) // find parent/representative of 5’s set

If the parents (p and q) are not same, they are in different set.
In this case, make p the parent of q (or vice versa).
```

Simulation



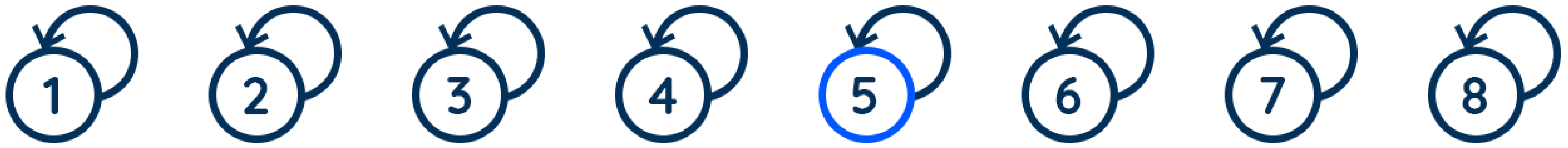
```
Union(1, 5)

p = Find(1)
```

Simulation



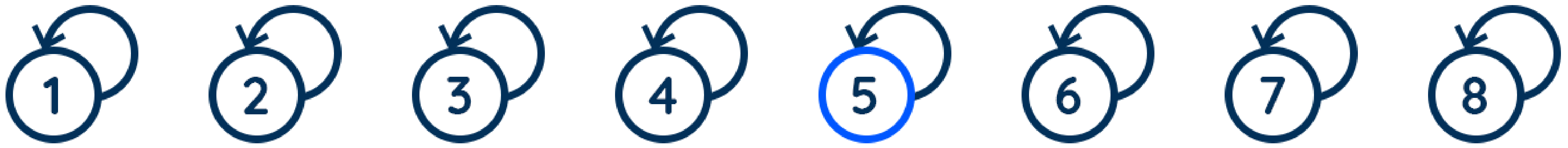
```
Union(1, 5)

p = Find(1) = 1
```

Simulation



```
Union(1, 5)

p = Find(1) = 1
q = Find(5)
```

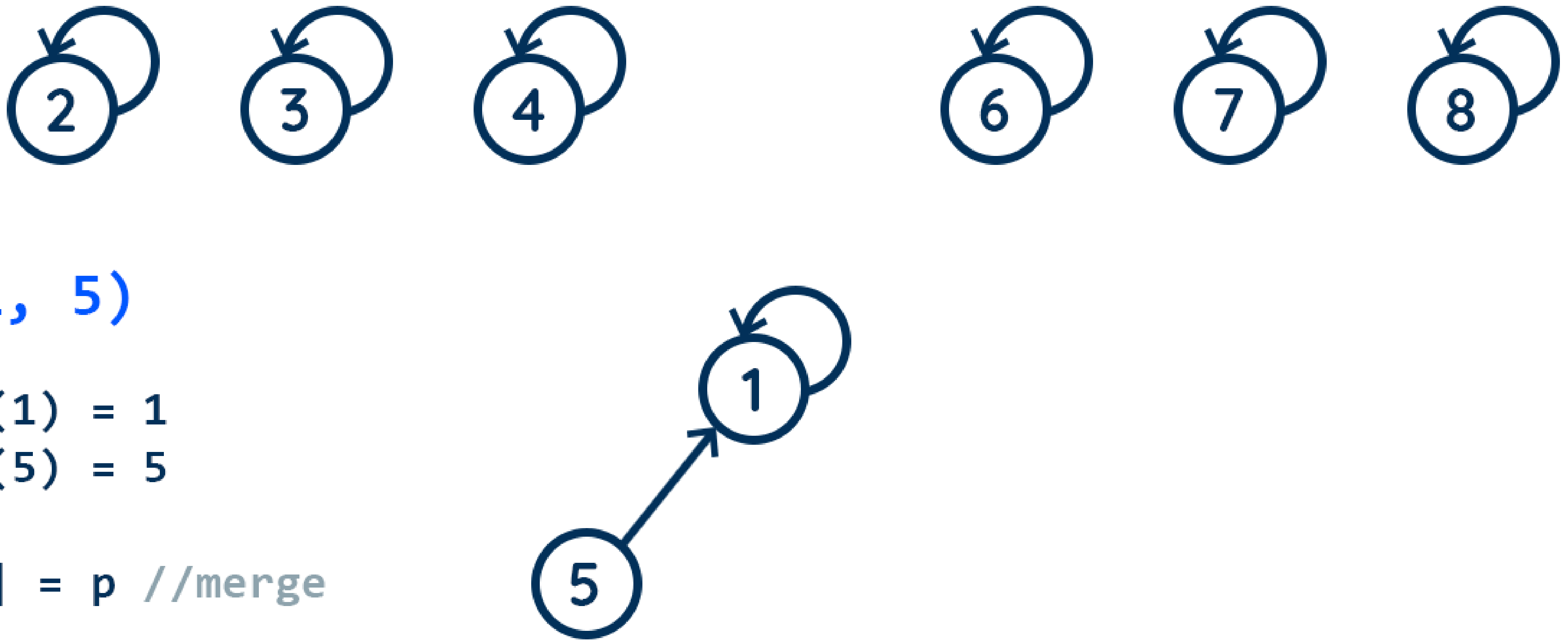
Simulation



```
Union(1, 5)
```

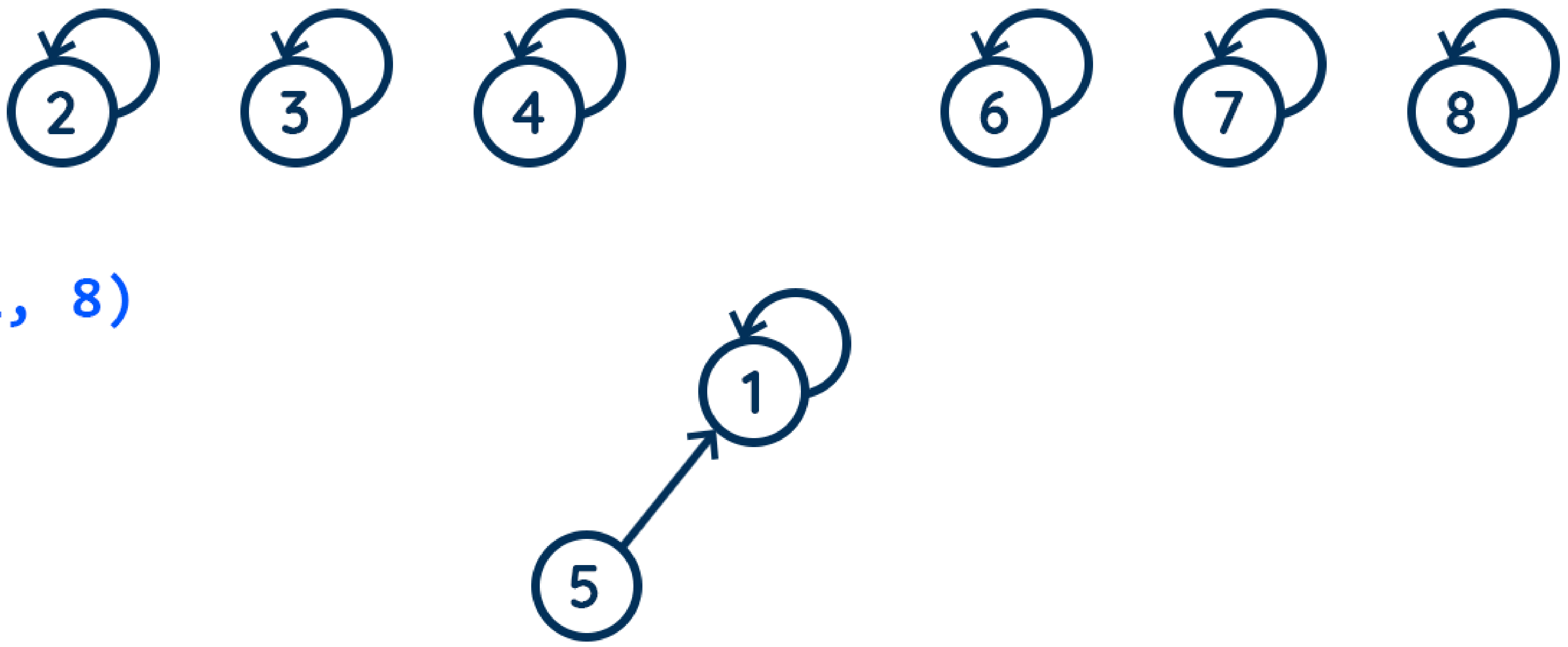
```
p = Find(1) = 1
q = Find(5) = 5
```

Simulation



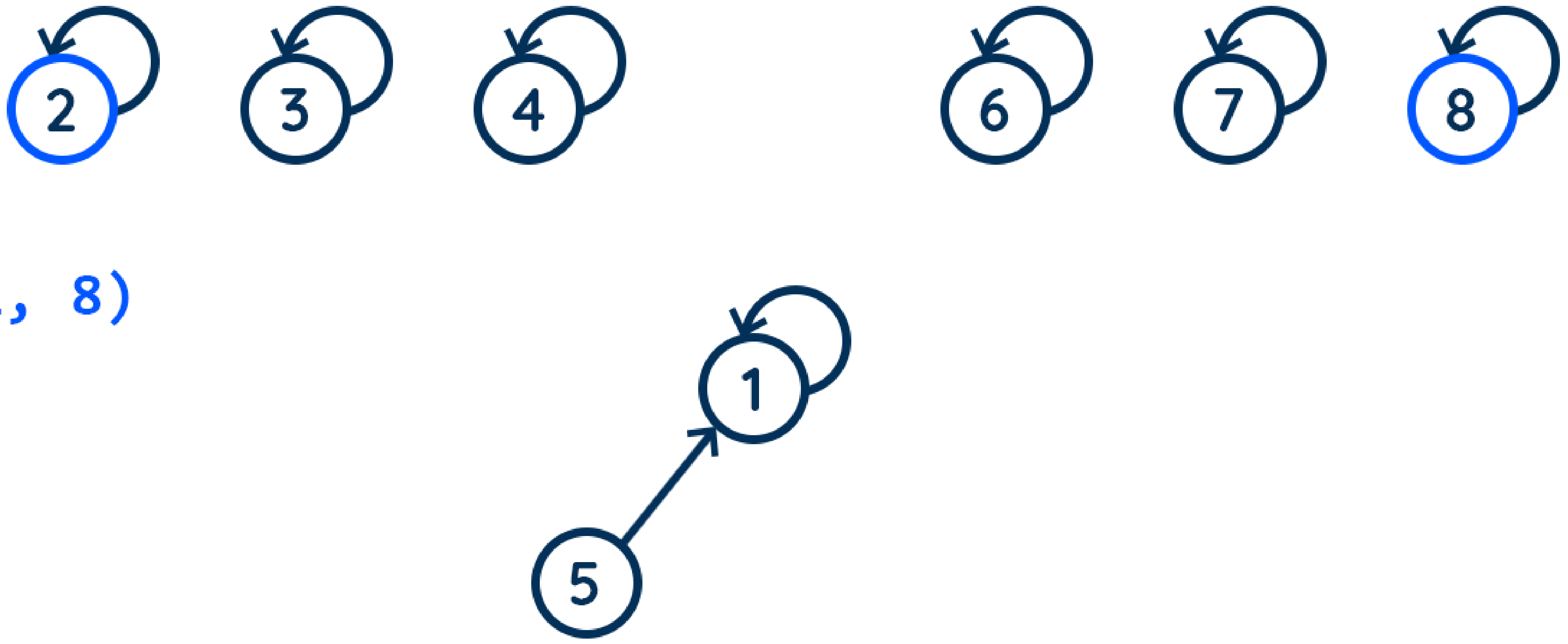
```
Union(1, 5)

p = Find(1) = 1
q = Find(5) = 5

parent[q] = p //merge
```

Simulation



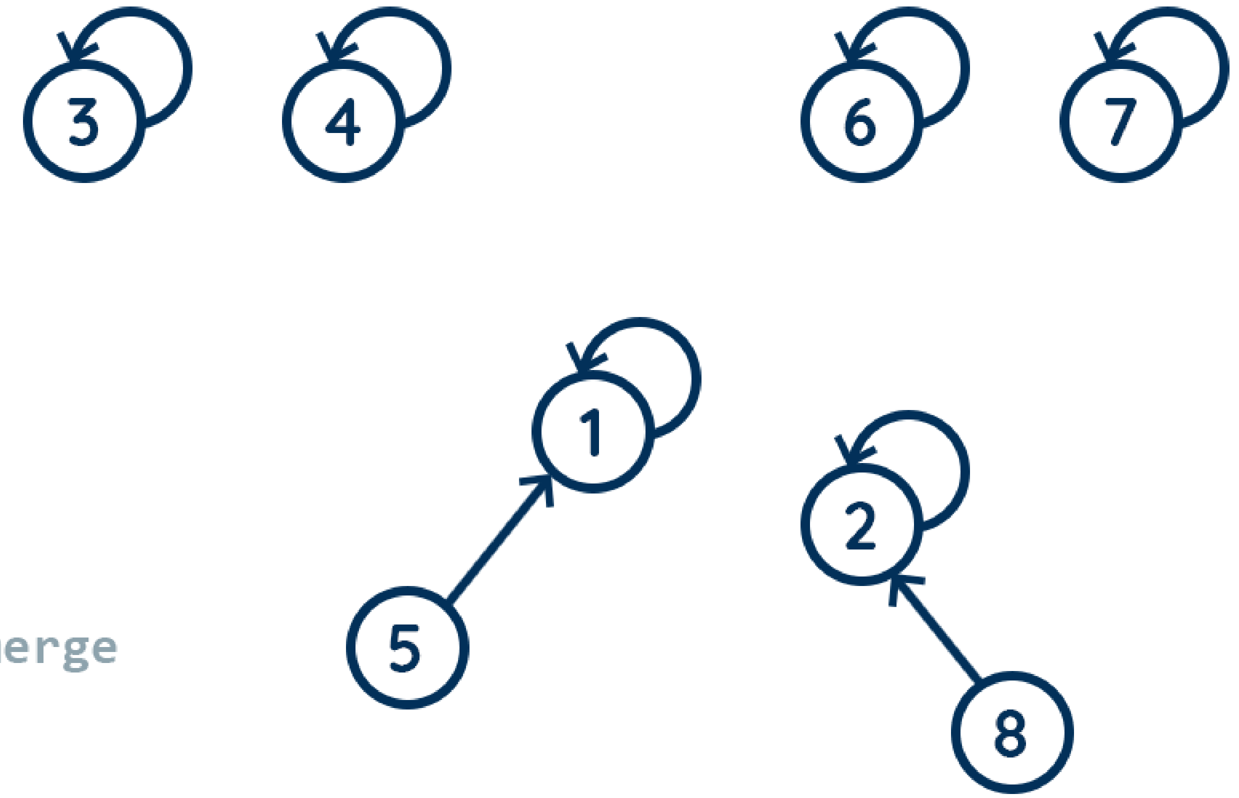
```
Union(2, 8)
```

Simulation



```
Union(2, 8)

p = 2
q = 8
```

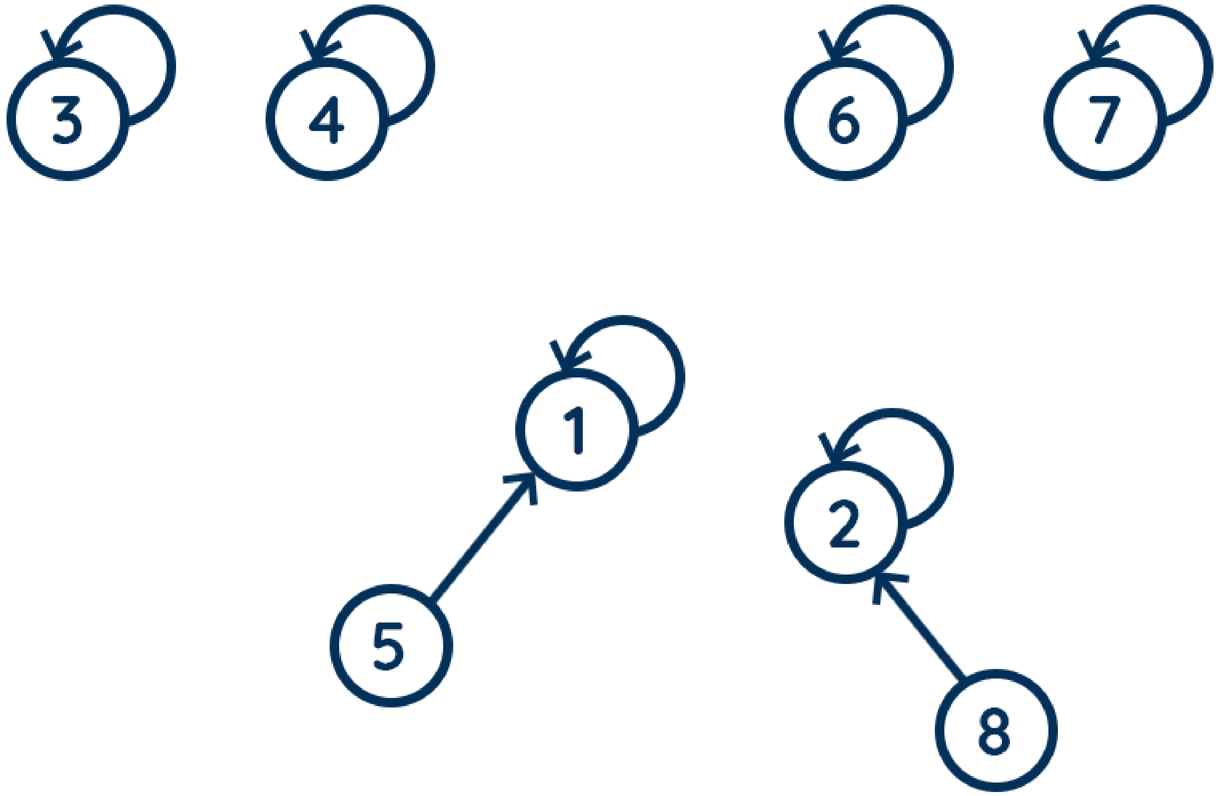
Simulation



```
Union(2, 8)

p = 2
q = 8

parent[8] = 2 //merge
```

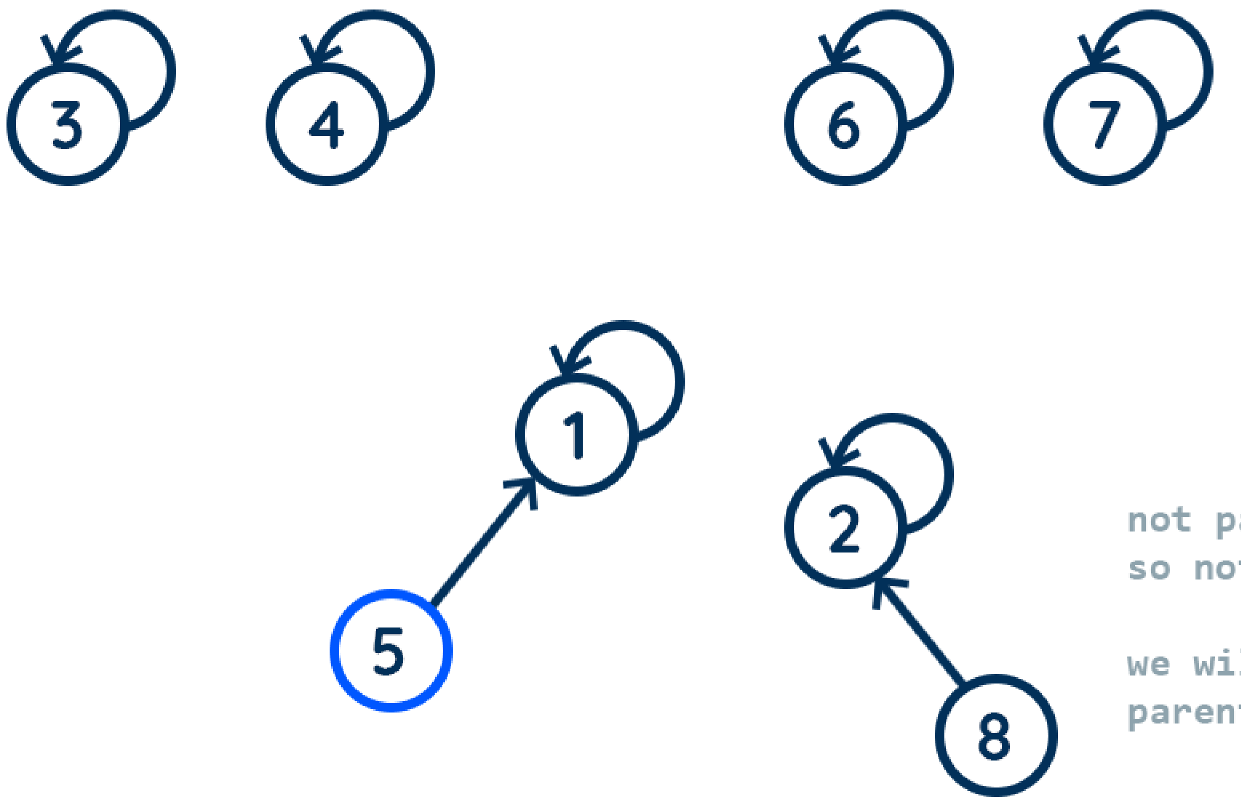
Simulation

`Union(5, 8)`



Simulation

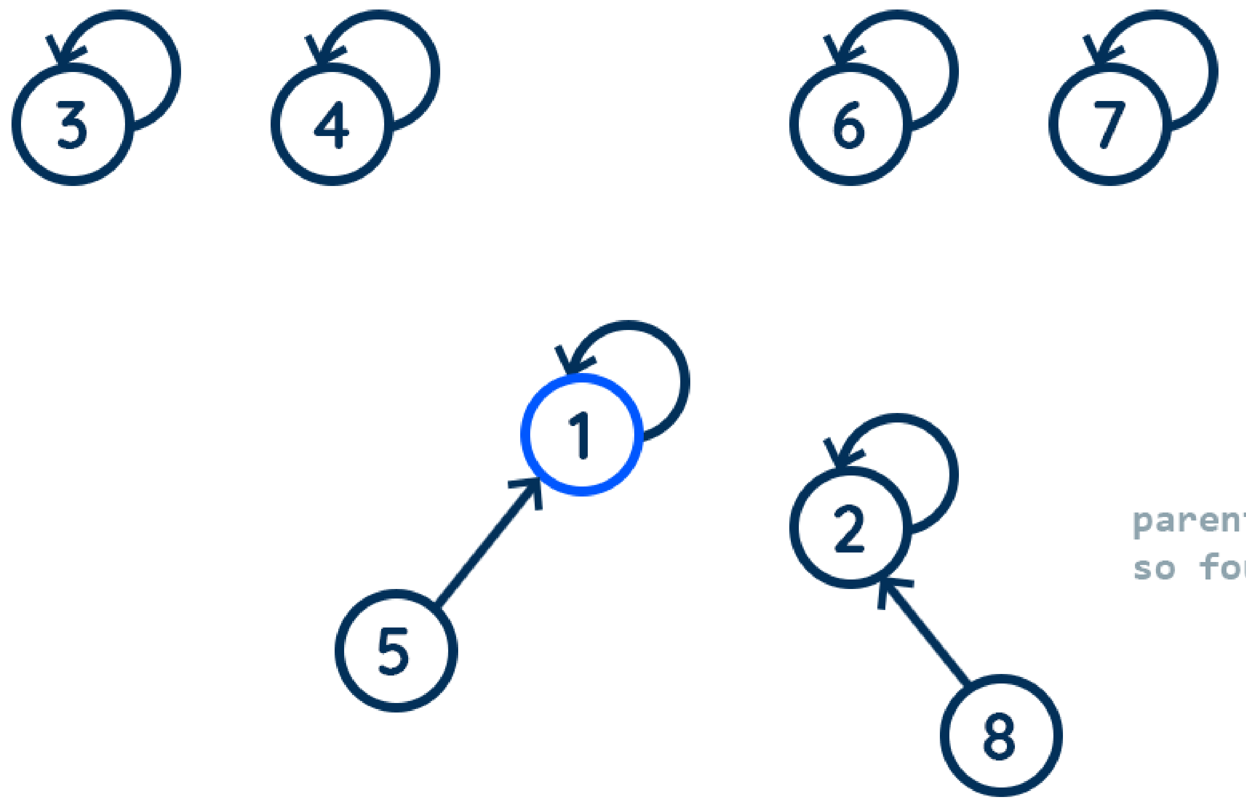
`Union(5, 8)`



```
not parent of itself
so not representative

we will move to its
parent
```

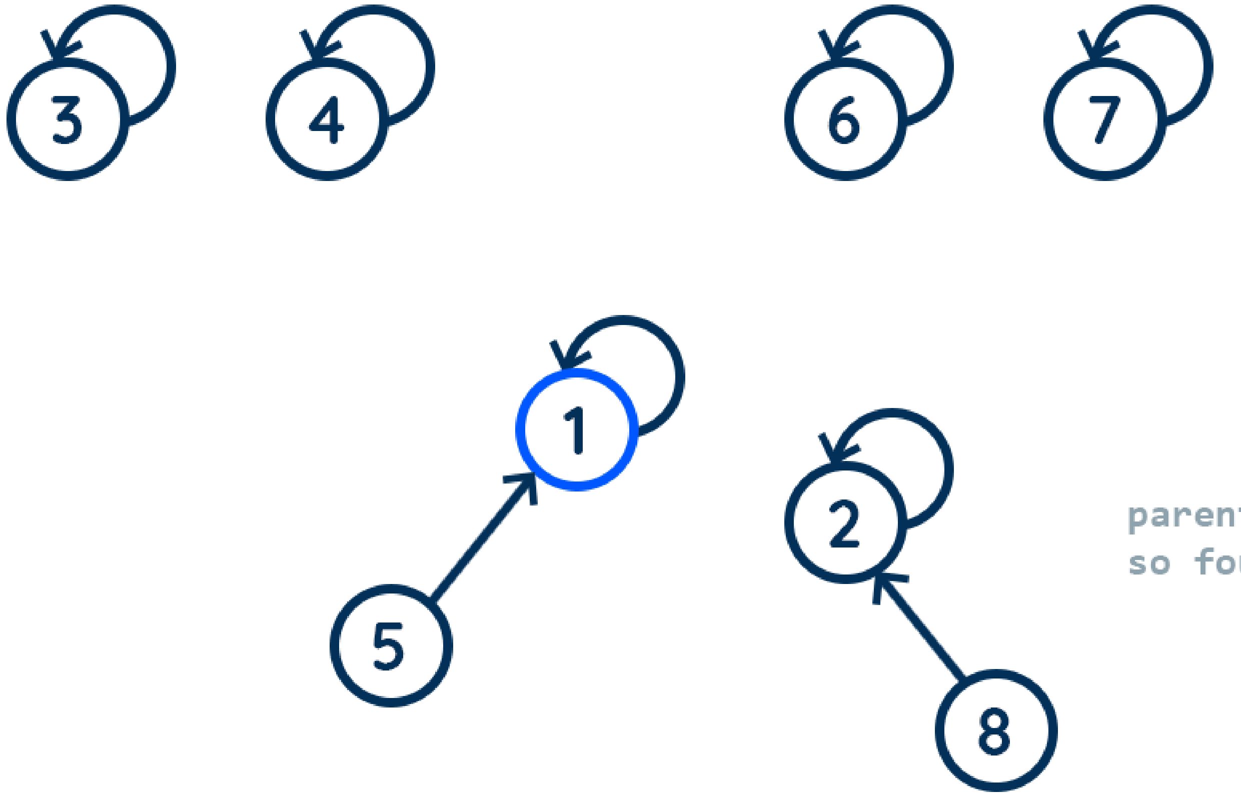
Simulation

`Union(5, 8)`



```
parent of itself
so found representative
```

Simulation

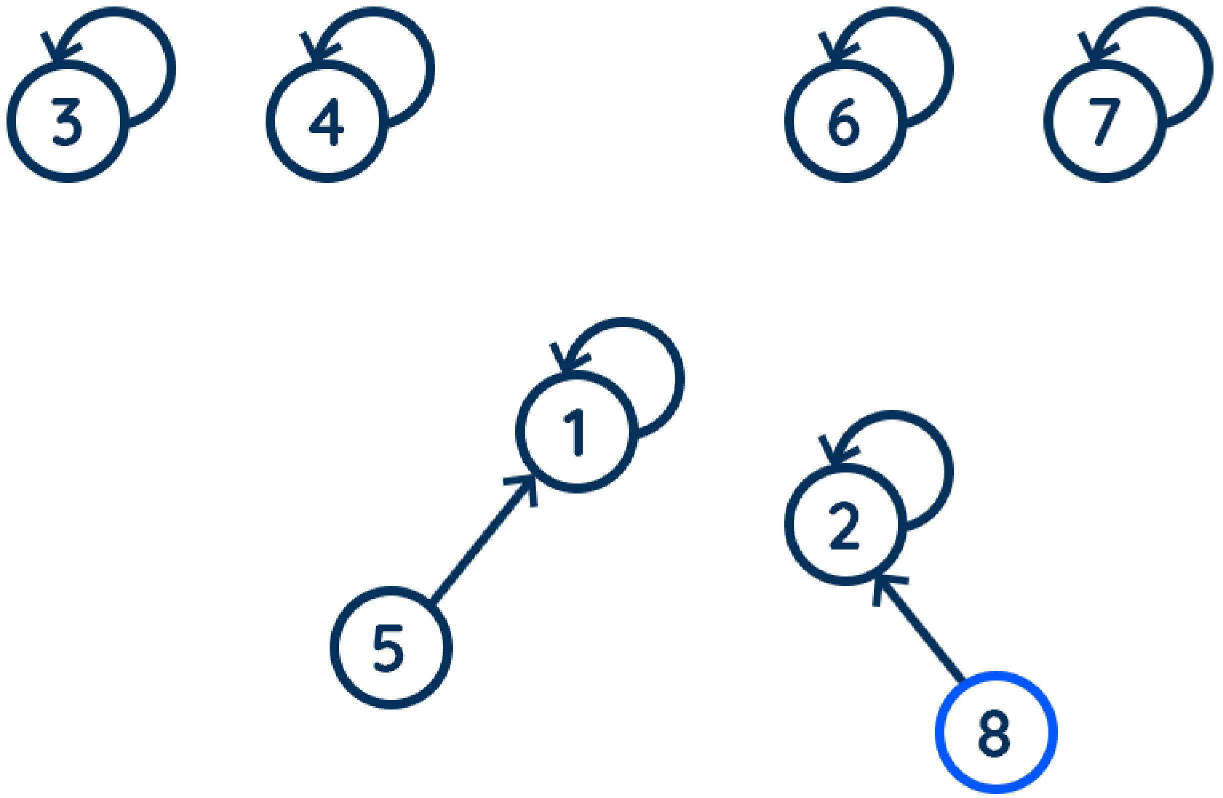


```
Union(5, 8)

p = 1
```

```
parent of itself
so found representative
```

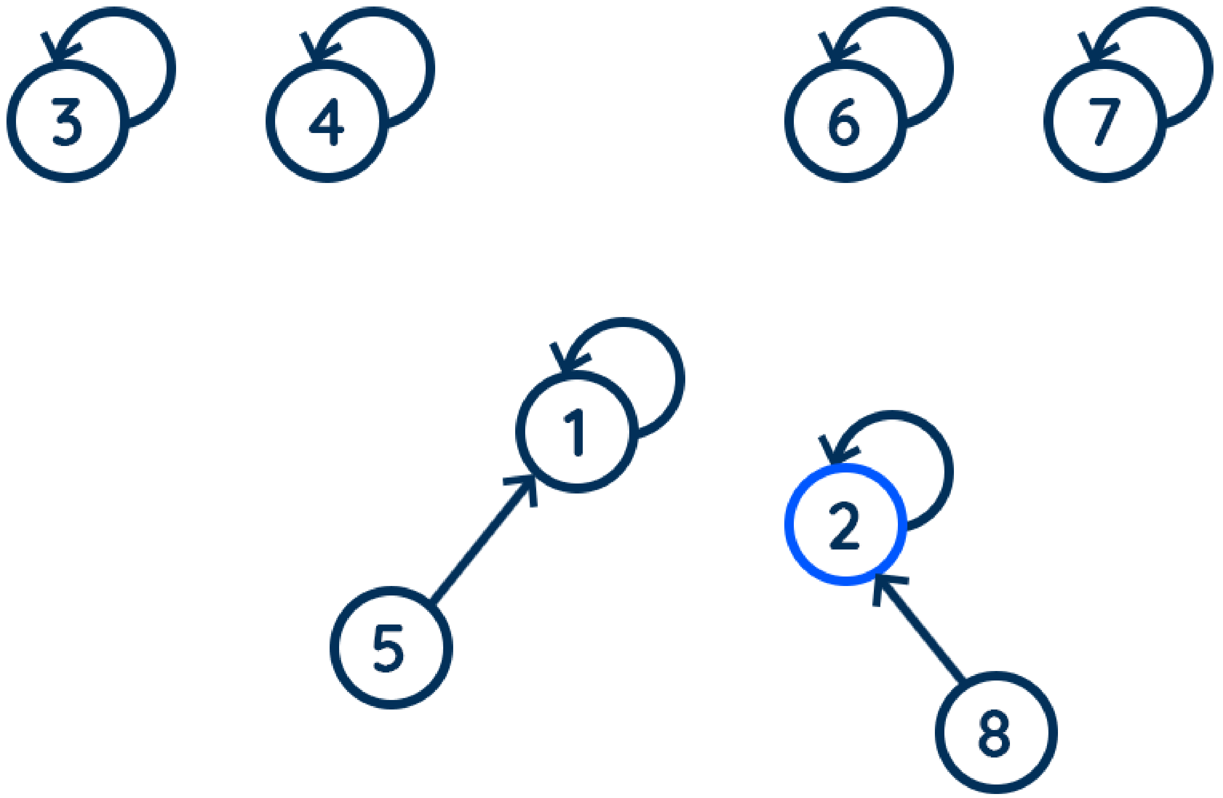
Simulation

```
Union(5, 8)

p = 1
```



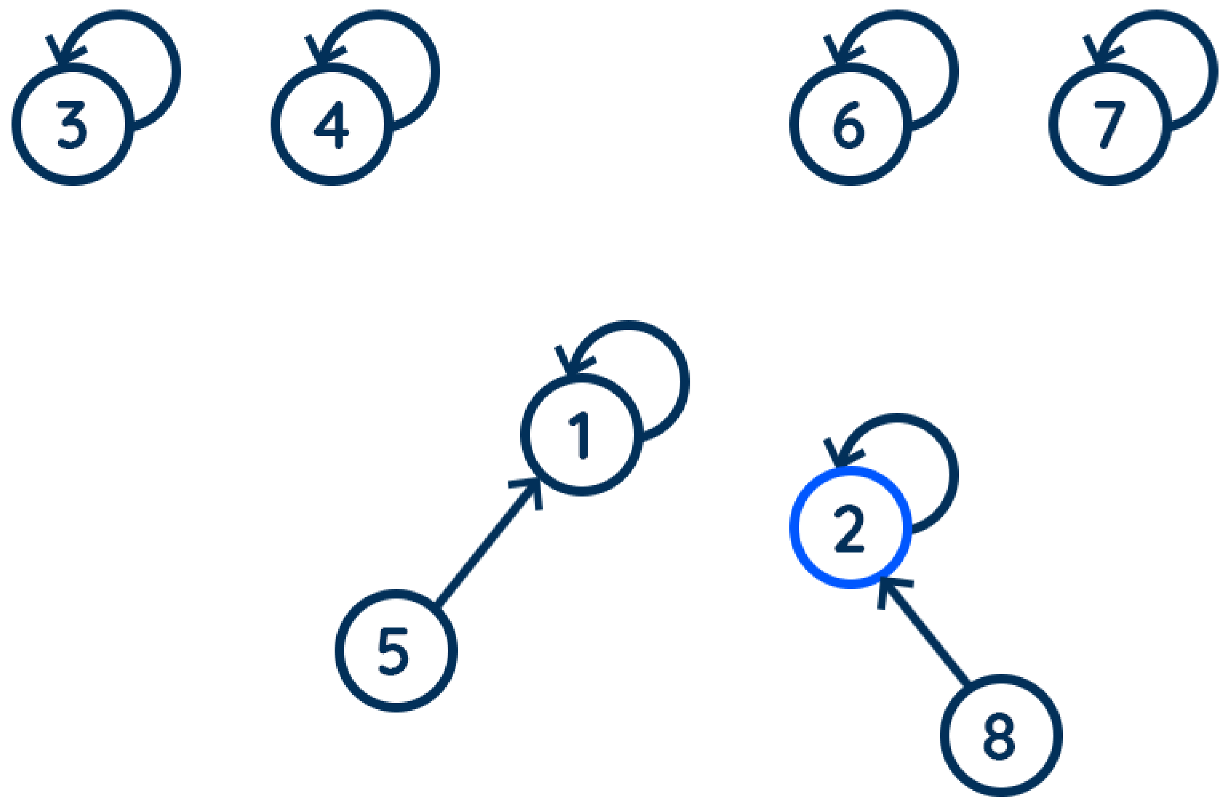
Simulation

```
Union(5, 8)

p = 1
```

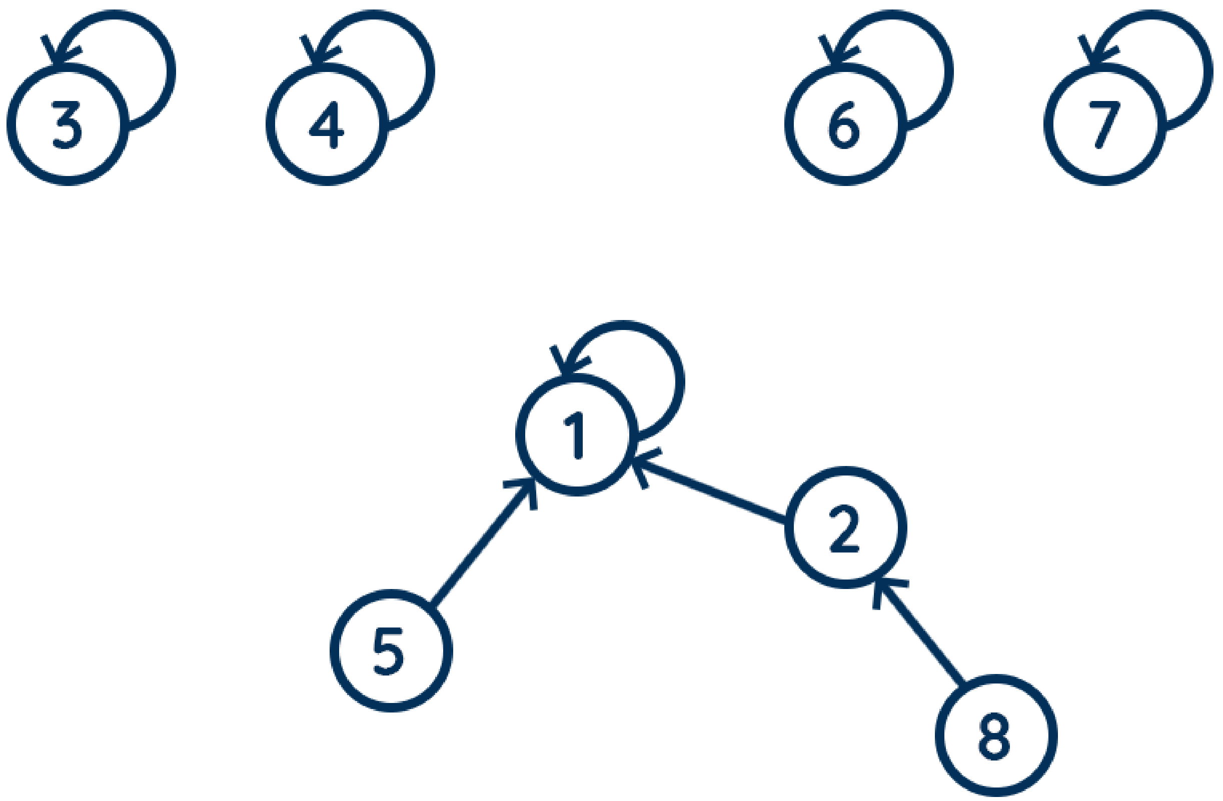


Simulation

```
Union(5, 8)

p = 1
q = 2
```



Simulation

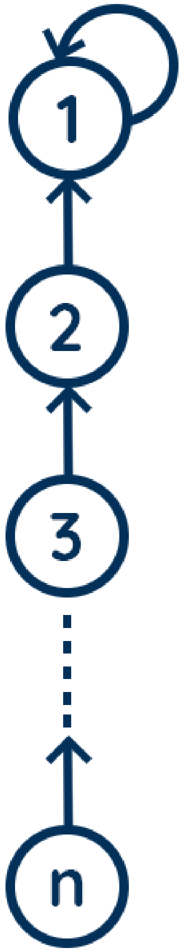


Union(5, 8)

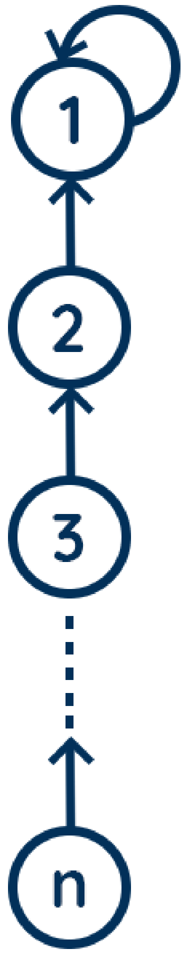
```
p = 1
q = 2

parent[2] = 1
```

Optimization

Optimization

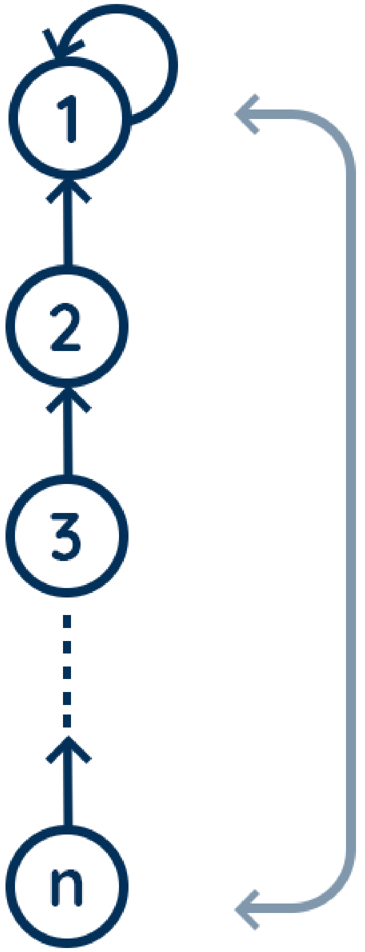


Optimization



```
Find(n)  //returns parent of n’s set
```

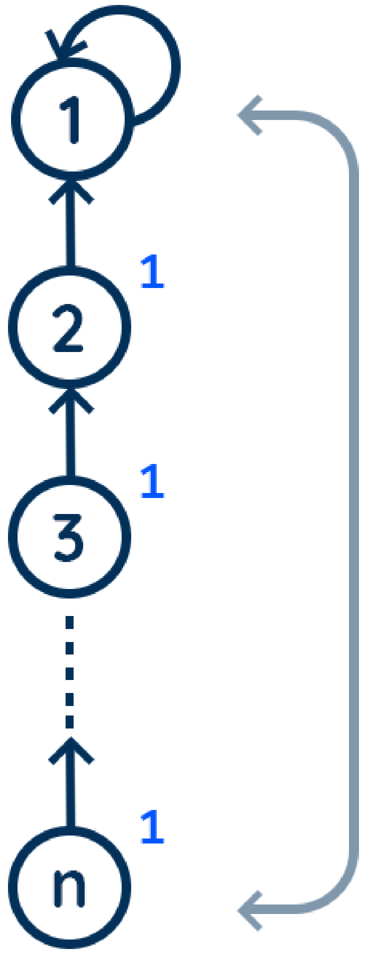
Optimization



```
Find(n)  //returns parent of n's set
```

```
traverses length n
every time we call Find(n)
```

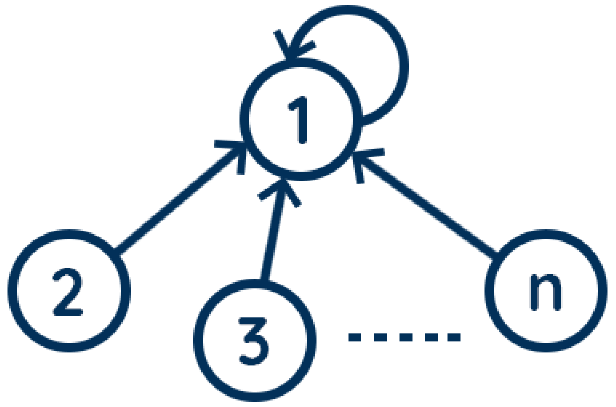
Optimization



```
Find(n)   //returns parent of n's set
```

```
traverses length n
every time we call Find(n)

we can set
parent[u] = Find(parent[u])
```

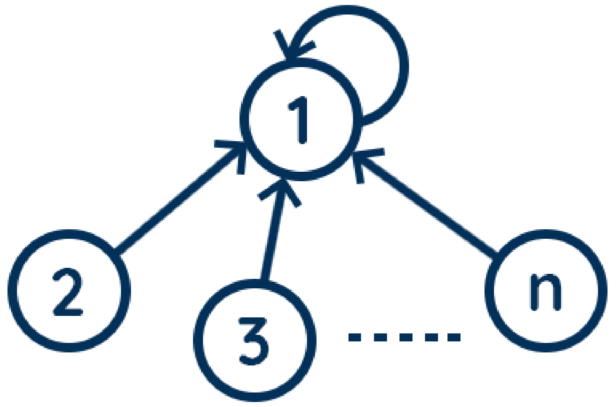
Optimization



`Find(n)` `//returns parent of n’s set`

```
traverses length n
every time we call Find(n)

we can set
parent[u] = Find(parent[u])

so next time we need to know
parent of u we have to call
once!
```

Optimization



this technique is called
path compression

`Find(n)` //returns parent of n's set

traverses length n
every time we call Find(n)

we can set
`parent[u] = Find(parent[u])`

so next time we need to know
parent of u we have to call
once!