Disjoint Sets

CS 225 - Fall 2025

Mattox Beckman

Objectives

- Use the up-tree data structure to model disjoint sets
- Use path compression to make this structure very efficient

Properties

Properties of Disjoint Sets

Let $s(a, b)$ indicate that a and b are in the same set.

- If $s(a, b)$ then $\forall x.s(a, x) \rightarrow s(b, x)$.

Properties

Properties of Disjoint Sets

Let $s(a, b)$ indicate that a and b are in the same set.

- If $s(a, b)$ then $\forall x.s(a, x) \rightarrow s(b, x)$.

Operations

- Declare that a and b are in the same set
- Check if a and b are in the same set
- Count how many elements are in a set

Representatives

What do these groups have in common?

- Lutherans
- Jacobites
- Marxists
- Platonists
- Dumbledor's Army

Up Trees

Representation

Use an integer vector `ds` to record the representatives

- `ds[i] == -1` means i is a root
- `ds[i] == n`, $n \geq 0$ means i's parent is n
- `ds[i] == n`, $n < 0$ means i is root
- ... and that the set contains $|n|$ elements

Find

Find(n)

- Keep following parents until you reach a root.

```
1 int find(vector<int> ds, int n) {
2     if (ds[n] < 0)
3         return n;
4     else
5         return find(ds,ds[n]);
6 }
```

Find

Find(n)

- Keep following parents until you reach a root.

```
1 int find(vector<int> ds, int n) {
2     if (ds[n] < 0)
3         return n;
4     else
5         return find(ds,ds[n]);
6 }
```

Find

Find(n)

- Keep following parents until you reach a root.

```
1 int find(vector<int> ds, int n) {
2     if (ds[n] < 0)
3         return n;
4     else
5         return find(ds,ds[n]);
6 }
```

Find

Find(n)

- Keep following parents until you reach a root.

```
1 int find(vector<int> ds, int n) {
2     if (ds[n] < 0)
3         return n;
4     else
5         return find(ds,ds[n]);
6 }
```

Find

Find(n)

- Keep following parents until you reach a root.

```
1 int find(vector<int> ds, int n) {
2     if (ds[n] < 0)
3         return n;
4     else
5         return find(ds,ds[n]);
6 }
```

Same Set?


```
1 a_rep = find(a);
2 b_rep = find(b);
3 if (a_rep == b_rep)
4      cout << "Same!" << endl;
5 else
6      cout << "Not Same!" << endl;
```

Union


```
 1 int union(vector<int> ds, int a,int b) {
 2     int ar = find(ds,a);
 3     int br = find(ds,b);
 4
 5     if (ar==br) return ds[ar];
 6
 7     if (ds[ar] < ds[br])  // we want br to be the largest set
 8         swap(ar,br);
 9
10     ds[br] += ds[ar];
11     ds[ar] = br;
12
13     return ds[br];
14 }
```

Union


```
 1 int union(vector<int> ds, int a,int b) {
 2     int ar = find(ds,a);
 3     int br = find(ds,b);
 4
 5     if (ar==br) return ds[ar];
 6
 7     if (ds[ar] < ds[br])  // we want br to be the largest set
 8         swap(ar,br);
 9
10     ds[br] += ds[ar];
11     ds[ar] = br;
12
13     return ds[br];
14 }
```

Union


```
 1 int union(vector<int> ds, int a,int b) {
 2     int ar = find(ds,a);
 3     int br = find(ds,b);
 4
 5     if (ar==br) return ds[ar];
 6
 7     if (ds[ar] < ds[br])  // we want br to be the largest set
 8         swap(ar,br);
 9
10     ds[br] += ds[ar];
11     ds[ar] = br;
12
13     return ds[br];
14 }
```

Union


```
 1 int union(vector<int> ds, int a,int b) {
 2     int ar = find(ds,a);
 3     int br = find(ds,b);
 4
 5     if (ar==br) return ds[ar];
 6
 7     if (ds[ar] < ds[br])  // we want br to be the largest set
 8         swap(ar,br);
 9
10     ds[br] += ds[ar];
11     ds[ar] = br;
12
13     return ds[br];
14 }
```

Union


```
 1 int union(vector<int> ds, int a,int b) {
 2     int ar = find(ds,a);
 3     int br = find(ds,b);
 4
 5     if (ar==br) return ds[ar];
 6
 7     if (ds[ar] < ds[br])  // we want br to be the largest set
 8         swap(ar,br);
 9
10     ds[br] += ds[ar];
11     ds[ar] = br;
12
13     return ds[br];
14 }
```

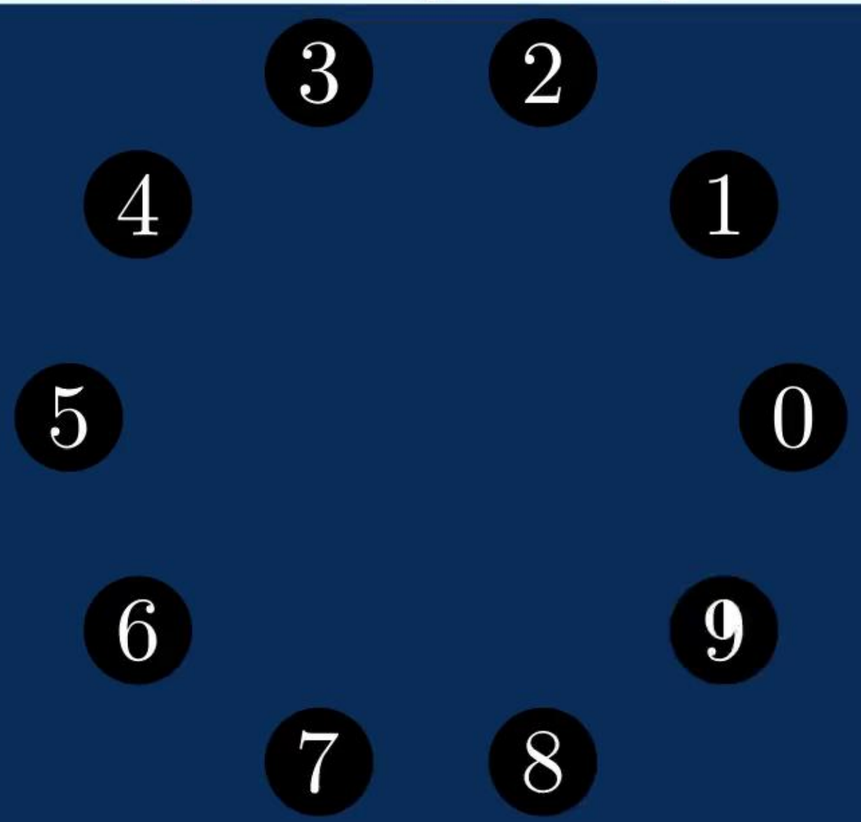
Union


```
 1 int union(vector<int> ds, int a,int b) {
 2     int ar = find(ds,a);
 3     int br = find(ds,b);
 4
 5     if (ar==br) return ds[ar];
 6
 7     if (ds[ar] < ds[br])  // we want br to be the largest set
 8         swap(ar,br);
 9
10     ds[br] += ds[ar];
11     ds[ar] = br;
12
13     return ds[br];
14 }
```

Example Run

Demo 1

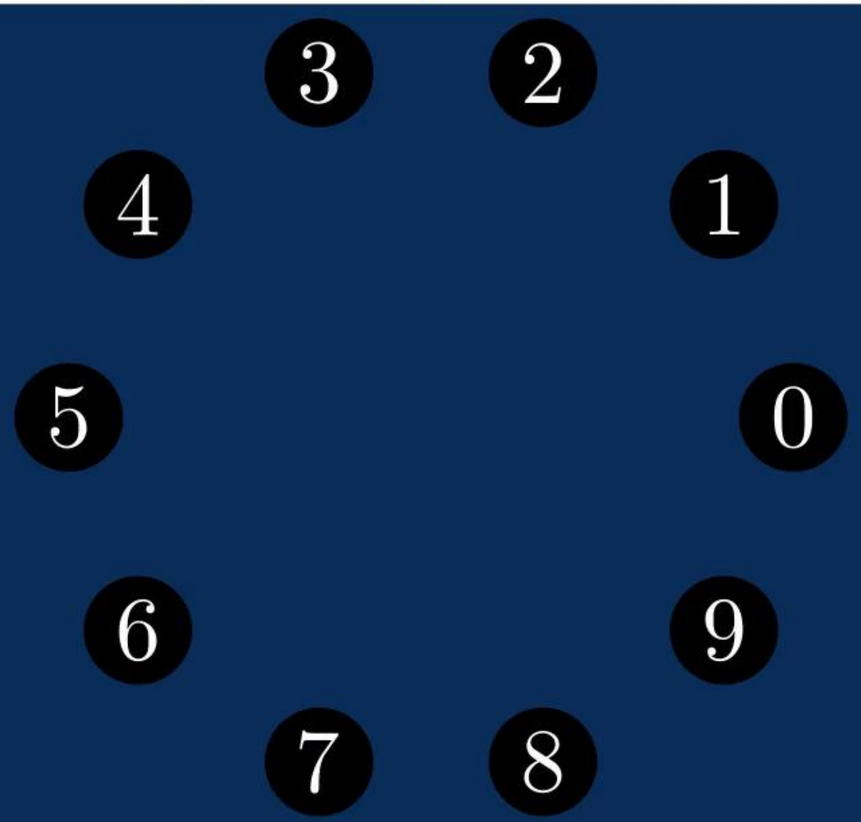
0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



Demo 1

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

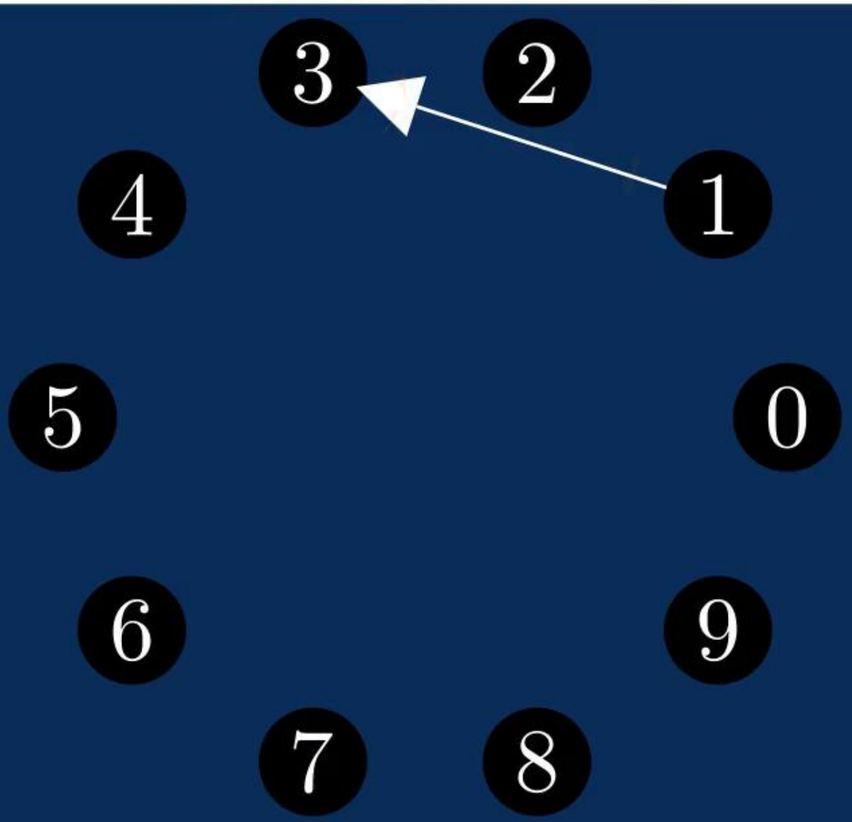
union(1,3)



Demo 1

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-2	-1	-1	-1	-1	-1	-1

union(1,3)

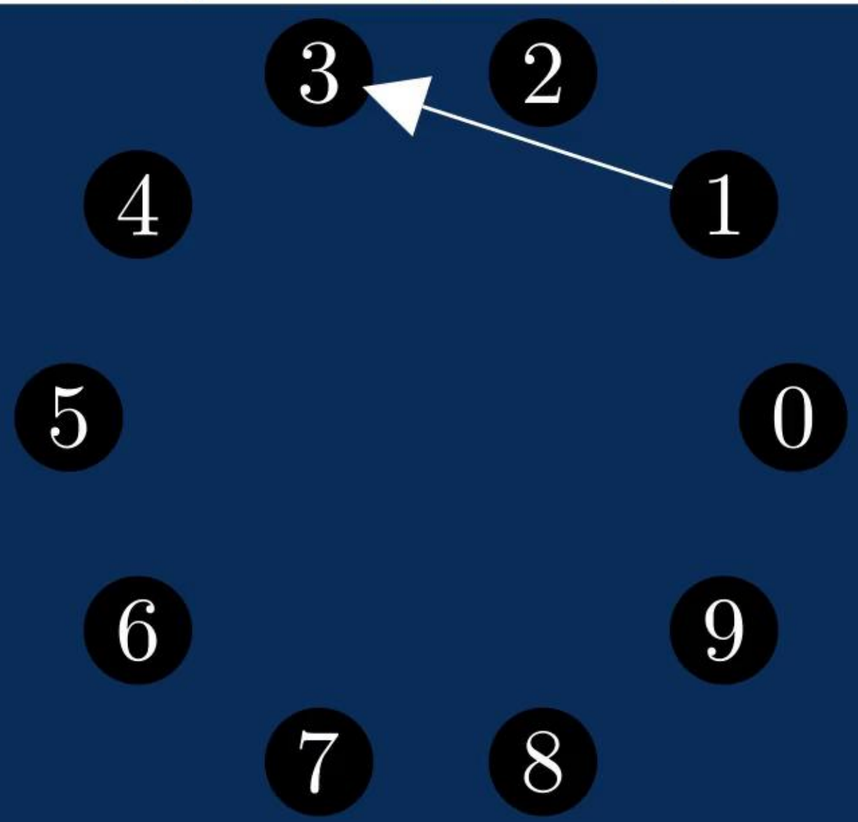


Demo 1

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-2	-1	-1	-1	-1	-1	-1

union(1,3)

union(1,4)

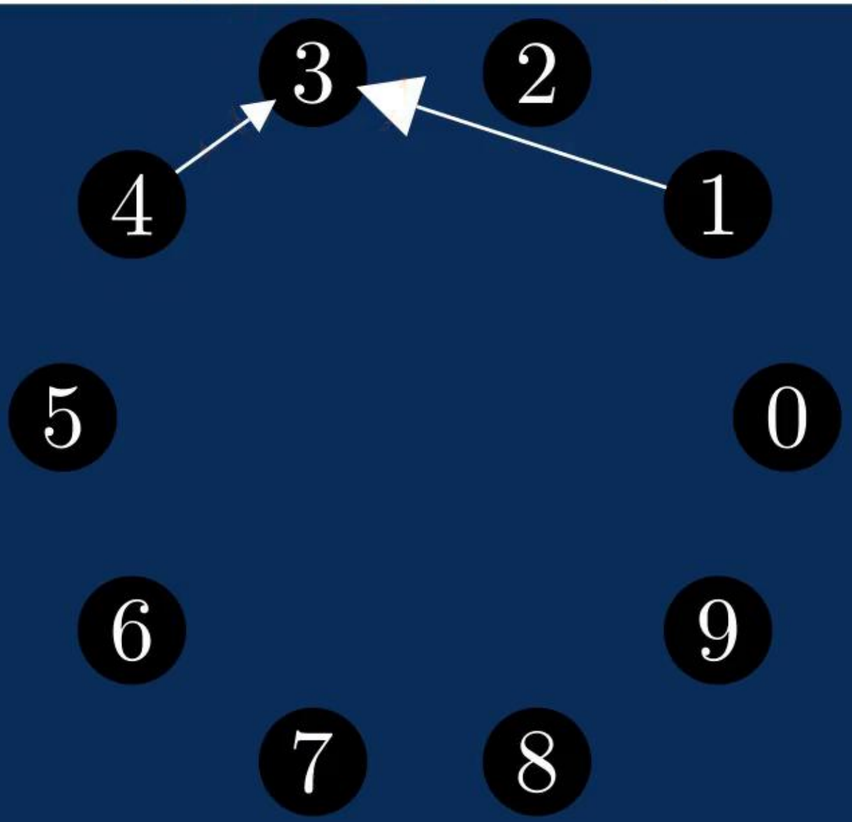


Demo 1

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	-1	-1	-1	-1	-1

union(1,3)

union(1,4)



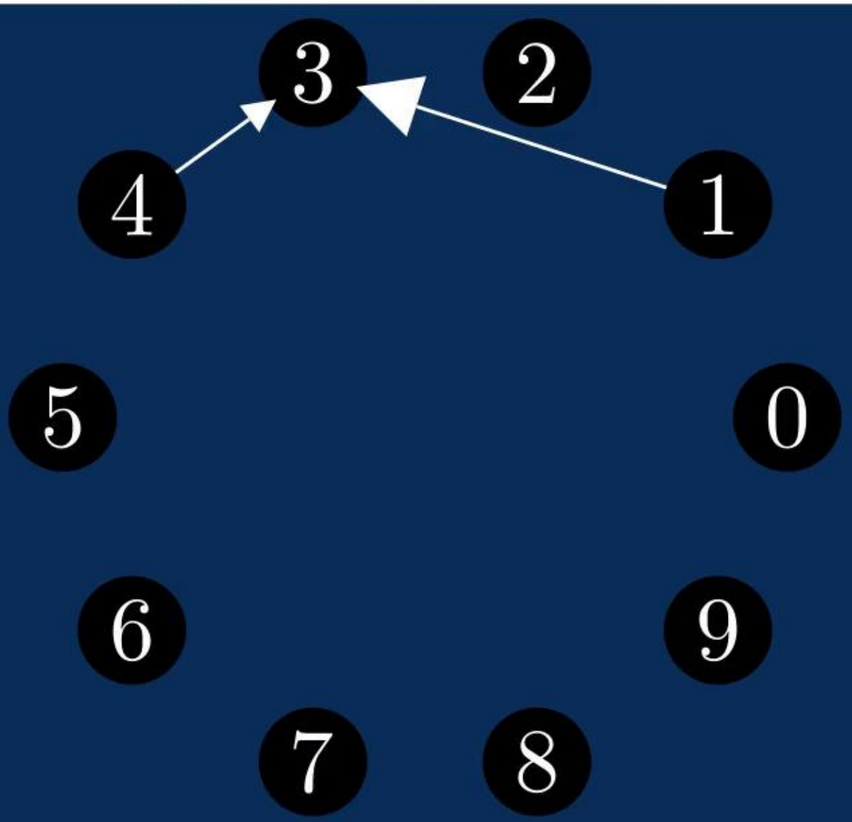
Demo 1

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	-1	-1	-1	-1	-1

union(1,3)

union(1,4)

union(5,9)



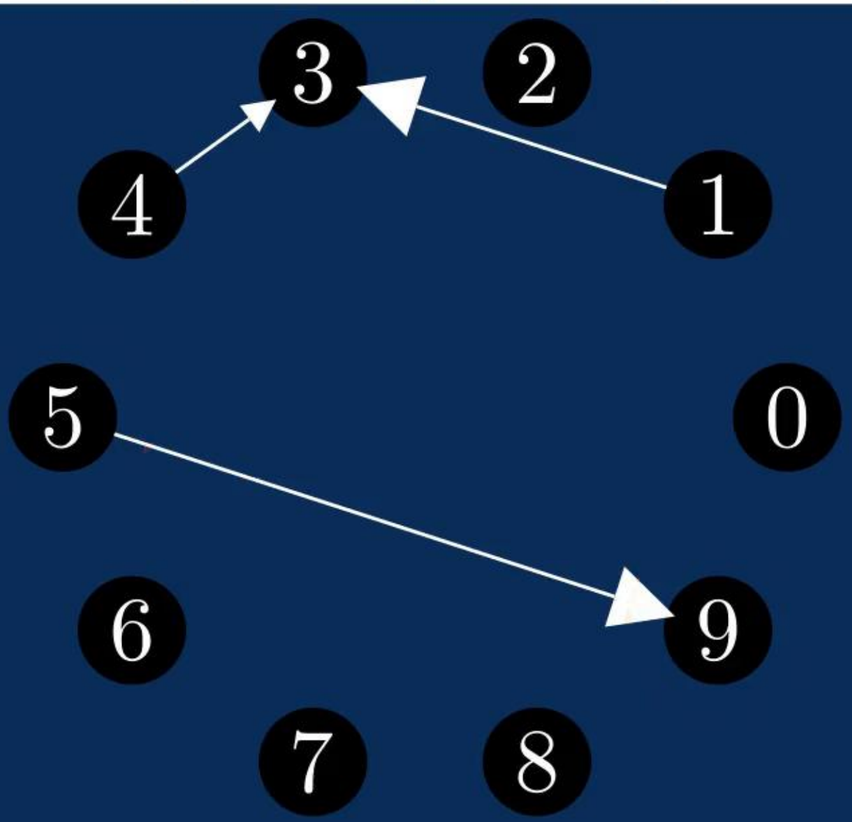
Demo 1

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	-1	-1	-1	-2

union(1,3)

union(1,4)

union(5,9)



Demo 1

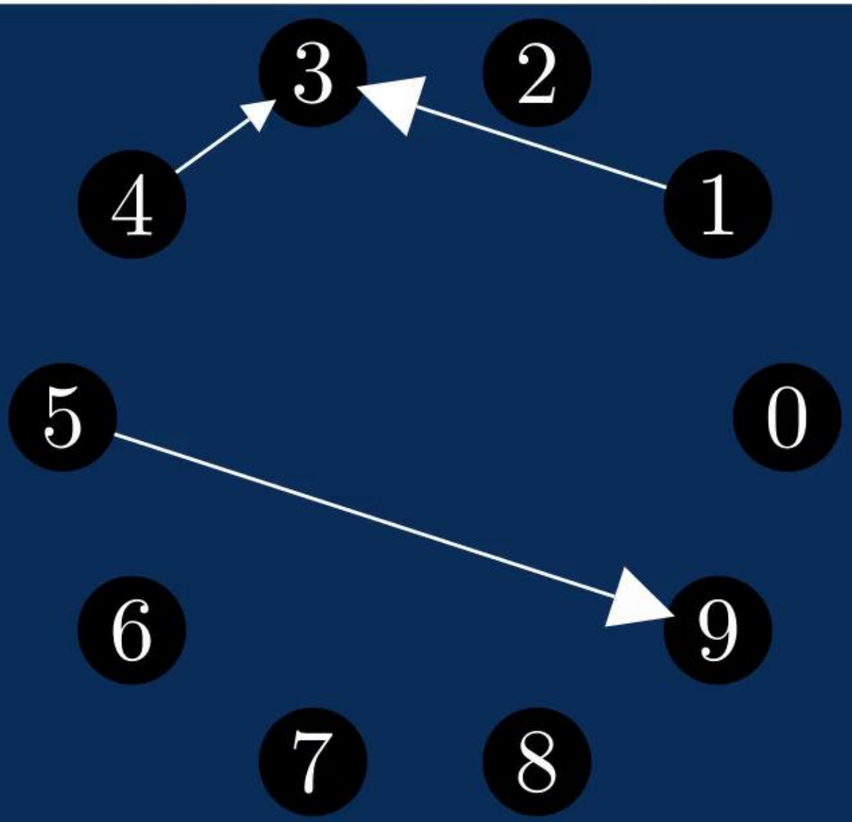
0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	-1	-1	-1	-2

union(1,3)

union(1,4)

union(5,9)

union(8,6)



Demo 1

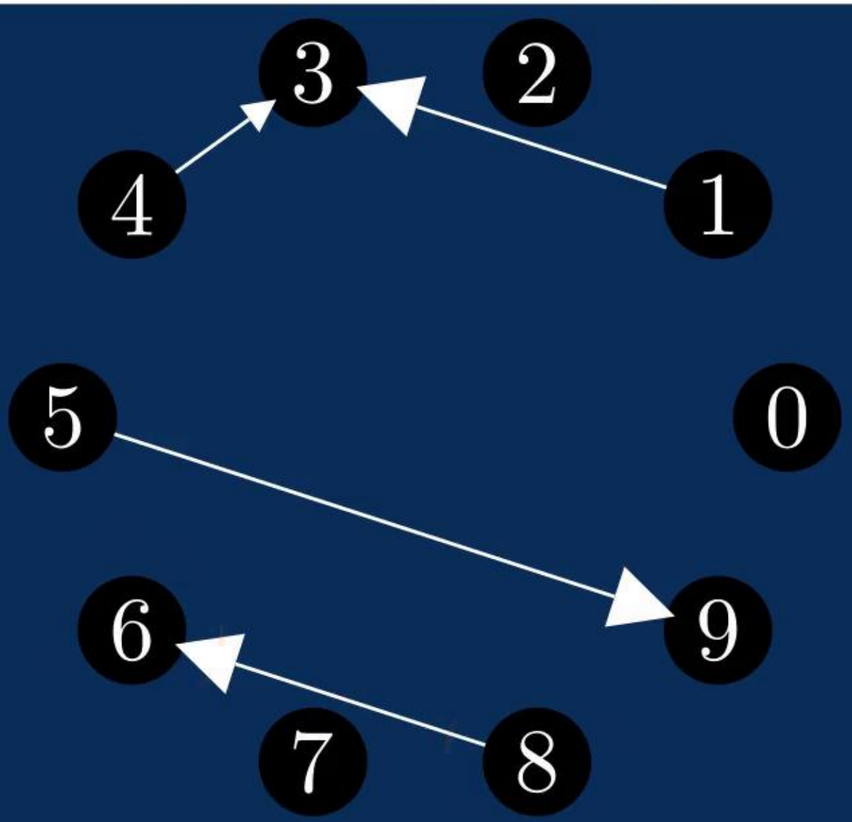
0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	-2	-1	6	-2

union(1,3)

union(1,4)

union(5,9)

union(8,6)



Demo 1

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	-2	-1	6	-2

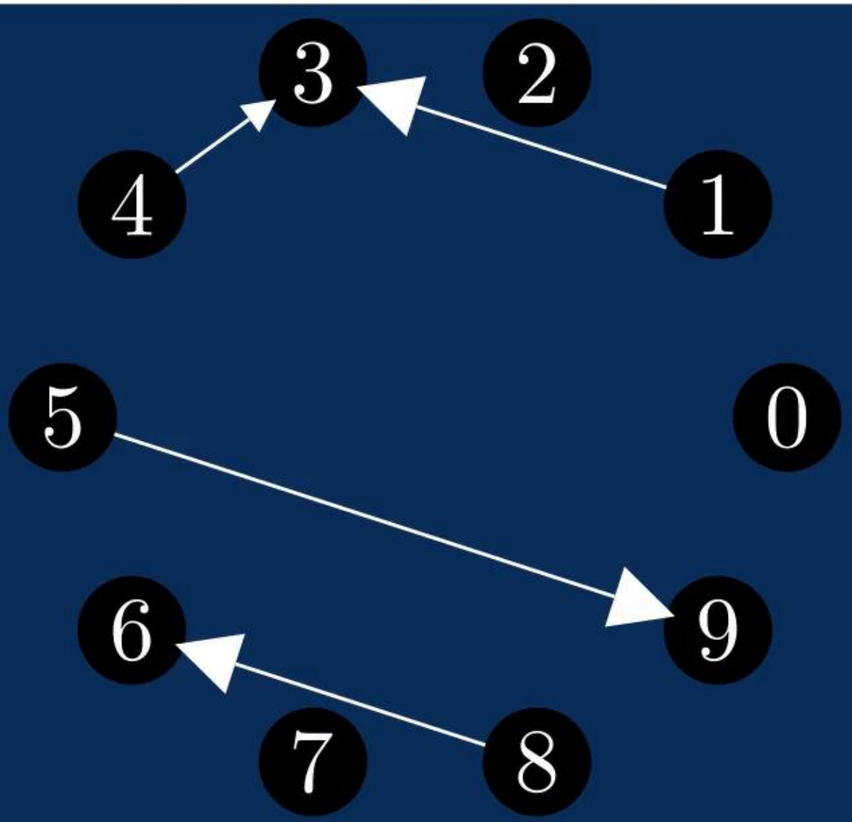
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



Demo 1

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	9	-1	6	-4

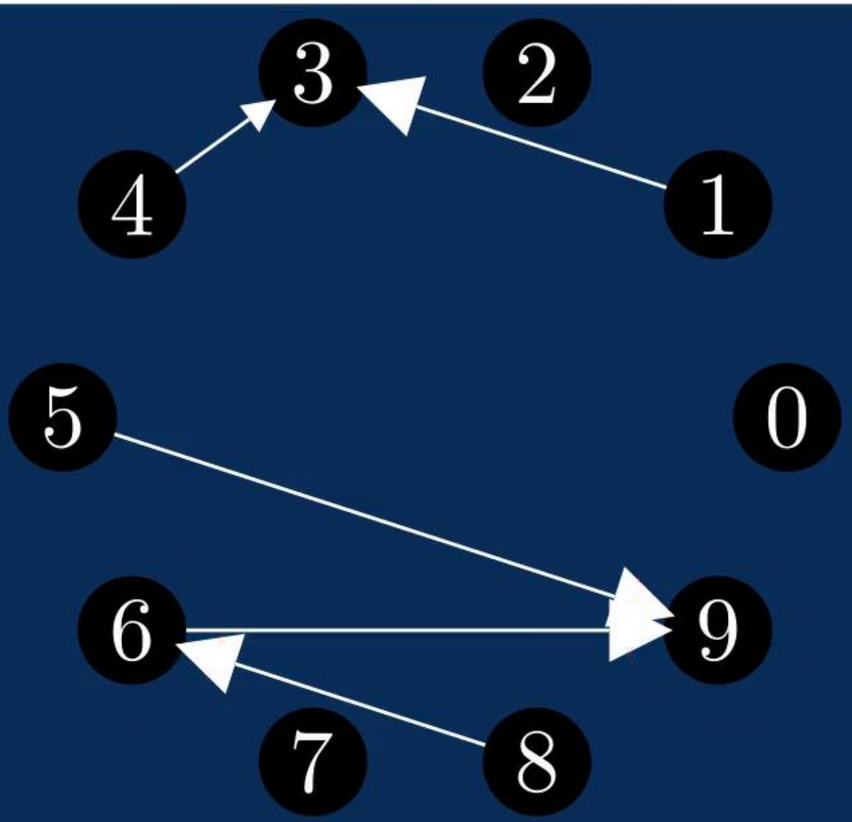
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



Demo 1

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	9	-1	6	-4

union(1,3)

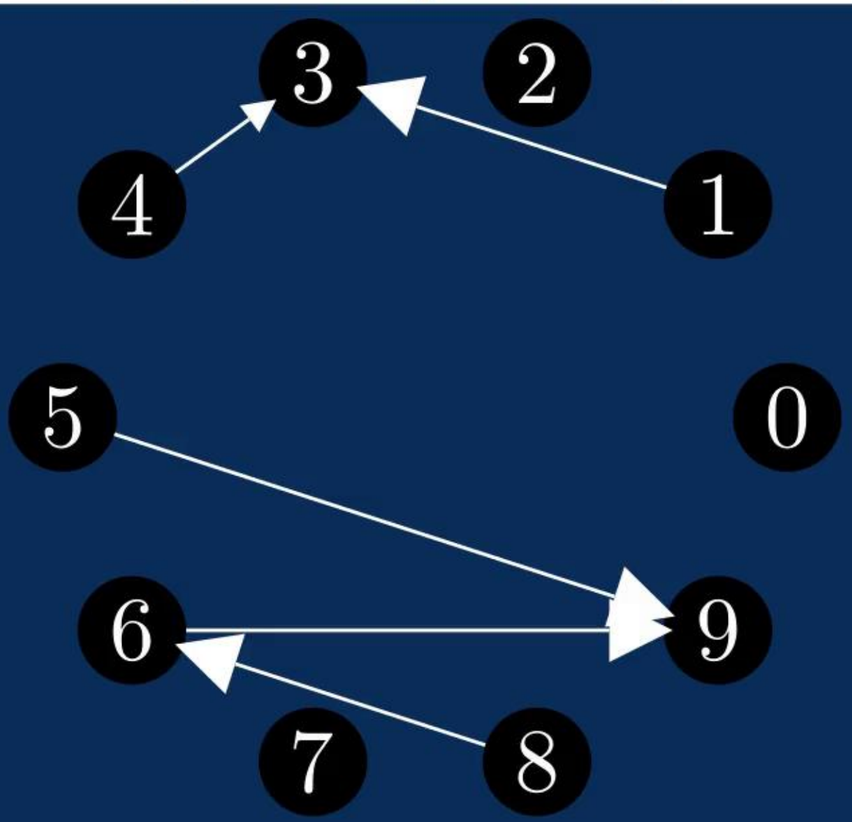
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Demo 1

0	1	2	3	4	5	6	7	8	9
-1	3	-1	9	3	9	9	-1	6	-7

union(1,3)

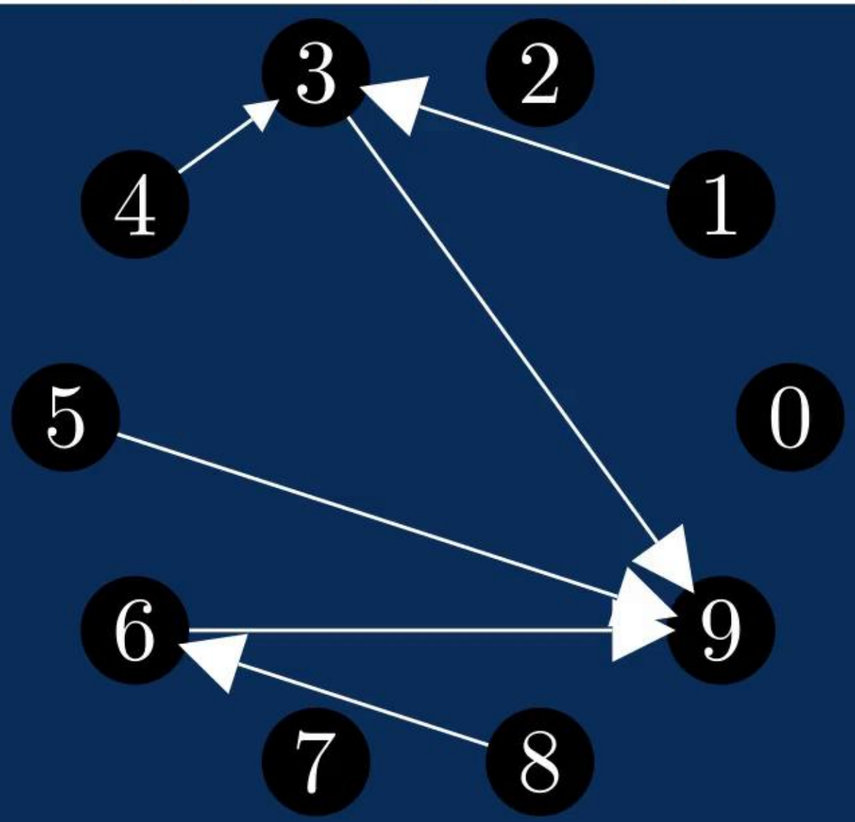
union(1,4)

union(5,9)

union(8,6)

union(8,5)

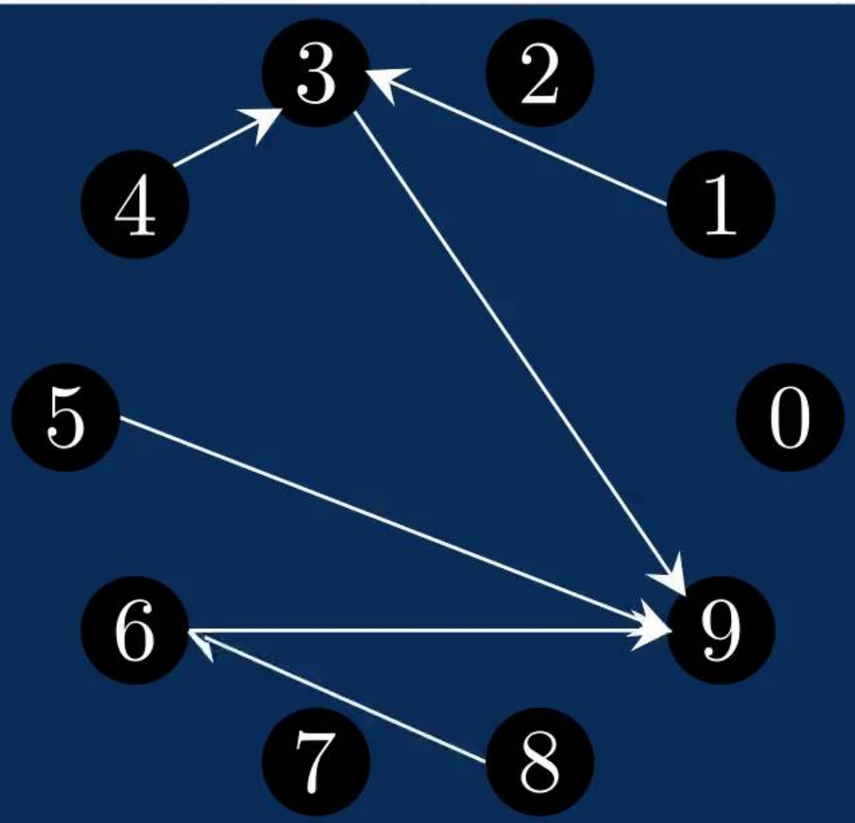
union(8,1)



Path Compression

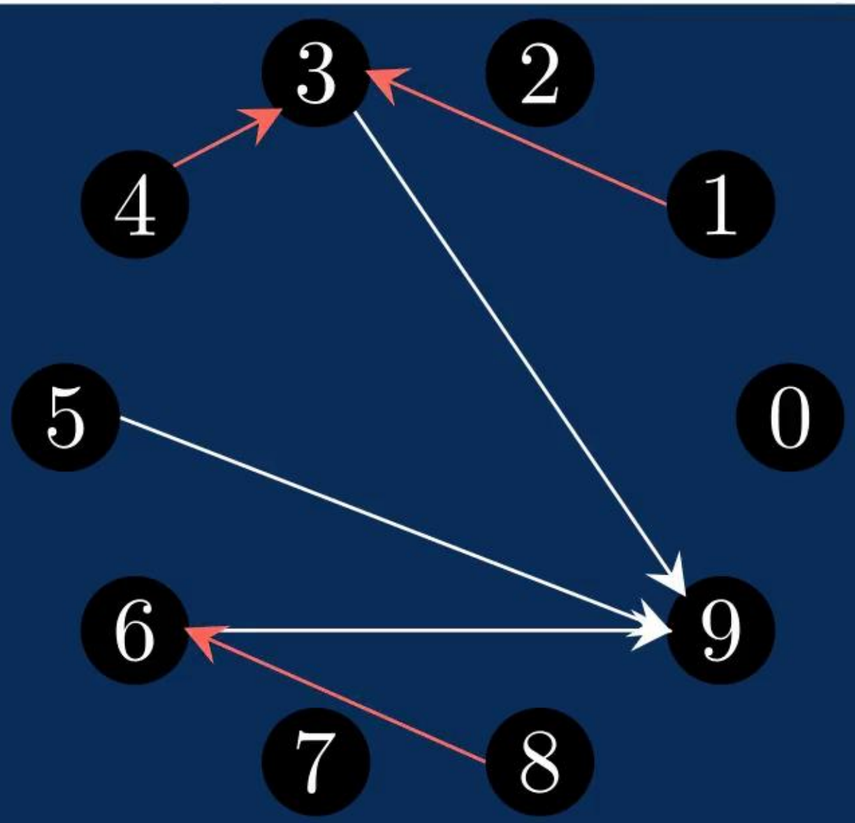
The Problem

0	1	2	3	4	5	6	7	8	9
-1	3	-1	9	3	9	9	-1	6	-7



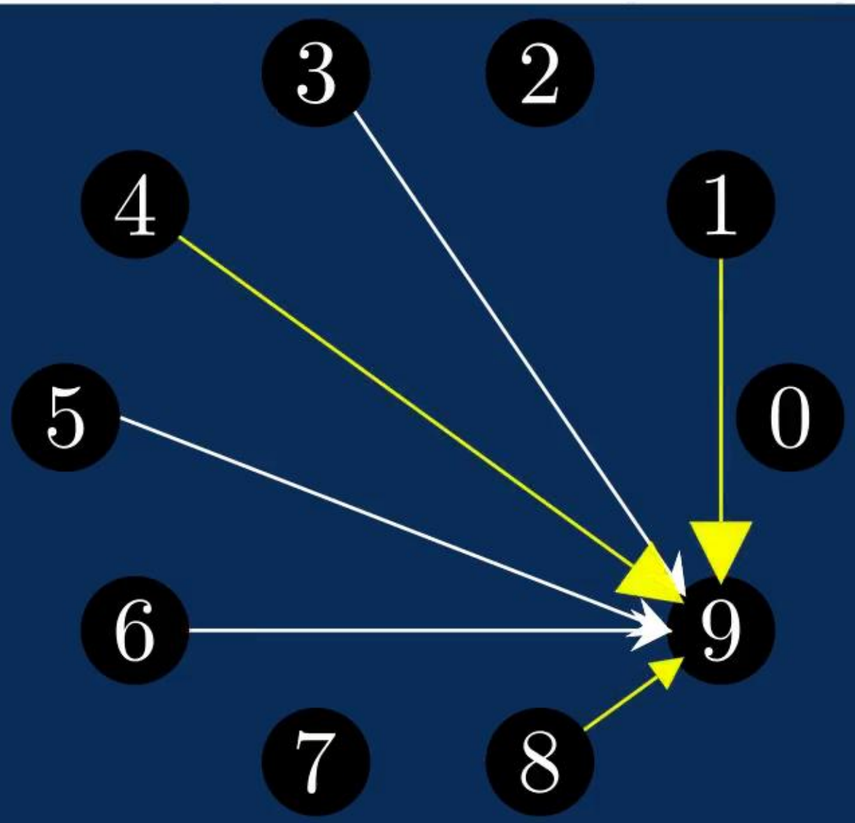
The Problem

0	1	2	3	4	5	6	7	8	9
-1	3	-1	9	3	9	9	-1	6	-7



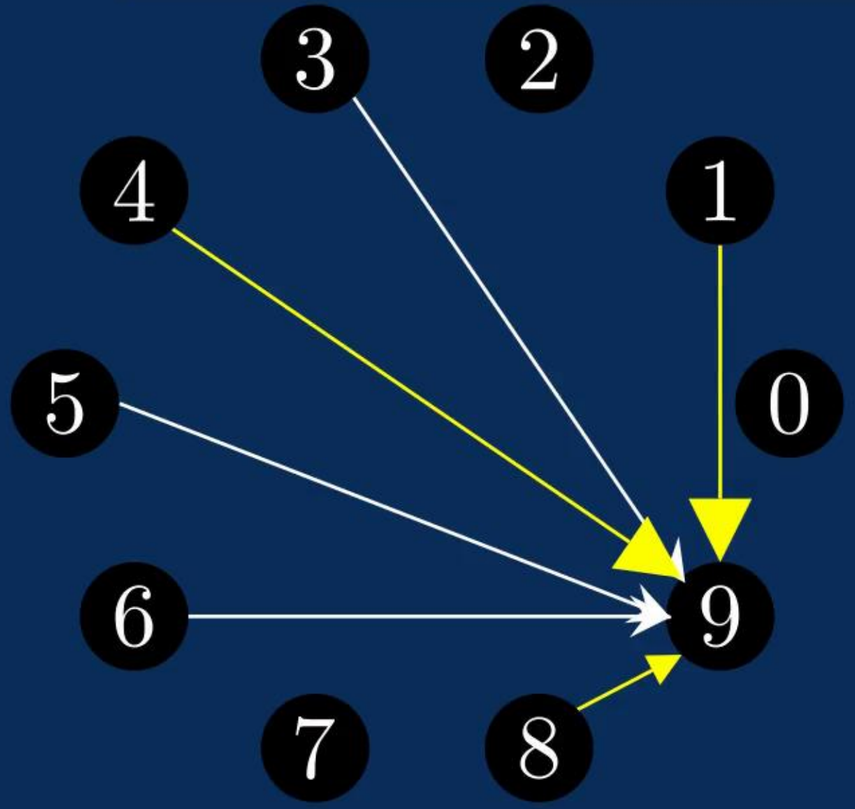
The Problem

0	1	2	3	4	5	6	7	8	9
-1	9	-1	9	9	9	9	-1	9	-7



The Problem

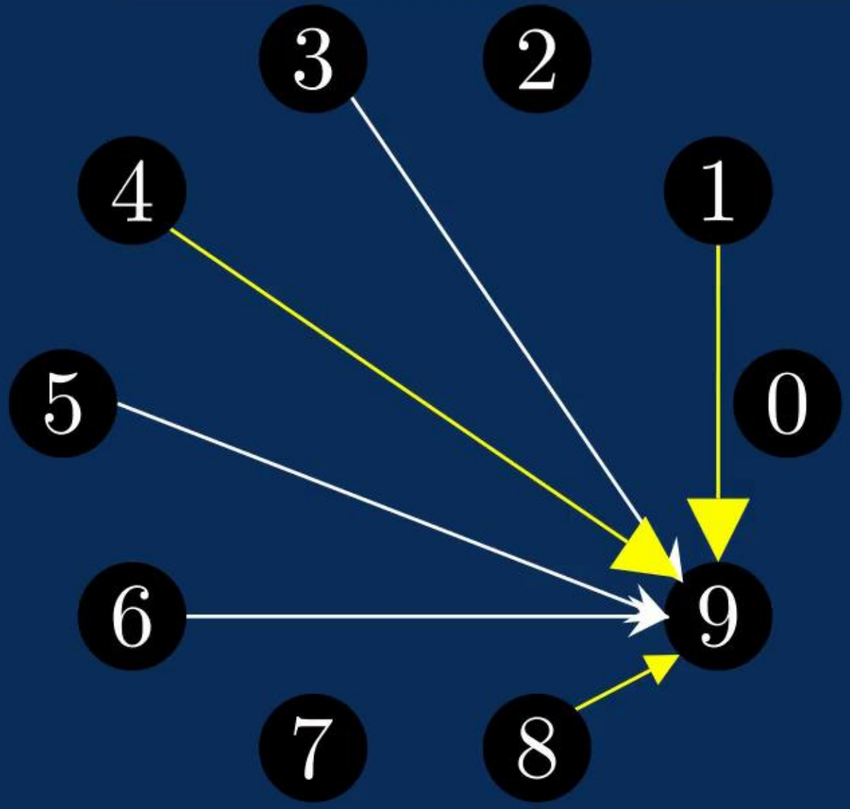
0	1	2	3	4	5	6	7	8	9
-1	9	-1	9	9	9	9	-1	9	-7




```
1 int find(vector<int> ds, int n) {
2     if (ds[n] < 0)
3         return -n;
4     else
5         return find(ds,ds[n]);
6 }
```

The Problem

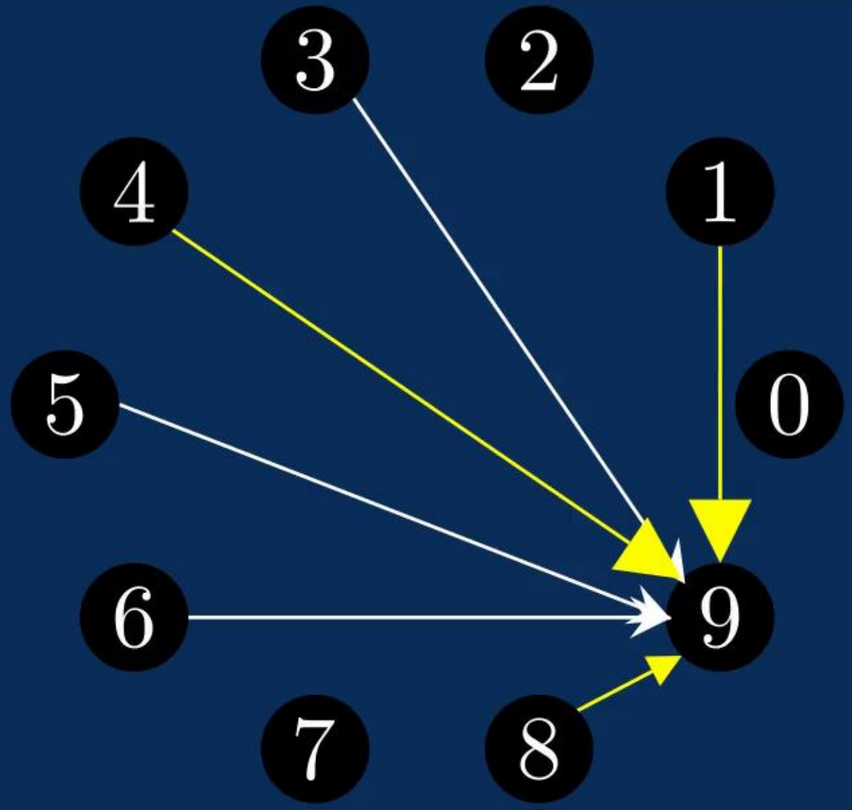
0	1	2	3	4	5	6	7	8	9
-1	9	-1	9	9	9	9	-1	9	-7




```
1 int find(vector<int> ds, int n) {
2     if (ds[n] < 0)
3         return -n;
4     else
5         return find(ds,ds[n]);
6 }
```

The Problem

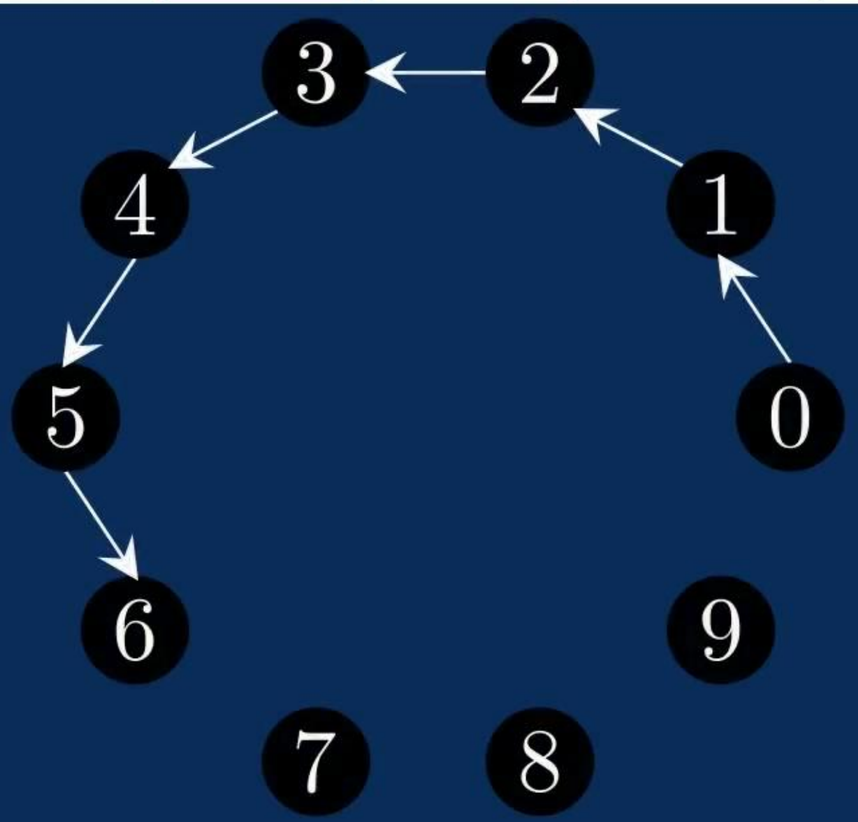
0	1	2	3	4	5	6	7	8	9
-1	9	-1	9	9	9	9	-1	9	-7




```
1 int find(vector<int> ds, int n) {
2     if (ds[n] < 0)
3         return -n;
4     else
5         return ds[n] = find(ds,ds[n]);
6 }
```

Demo

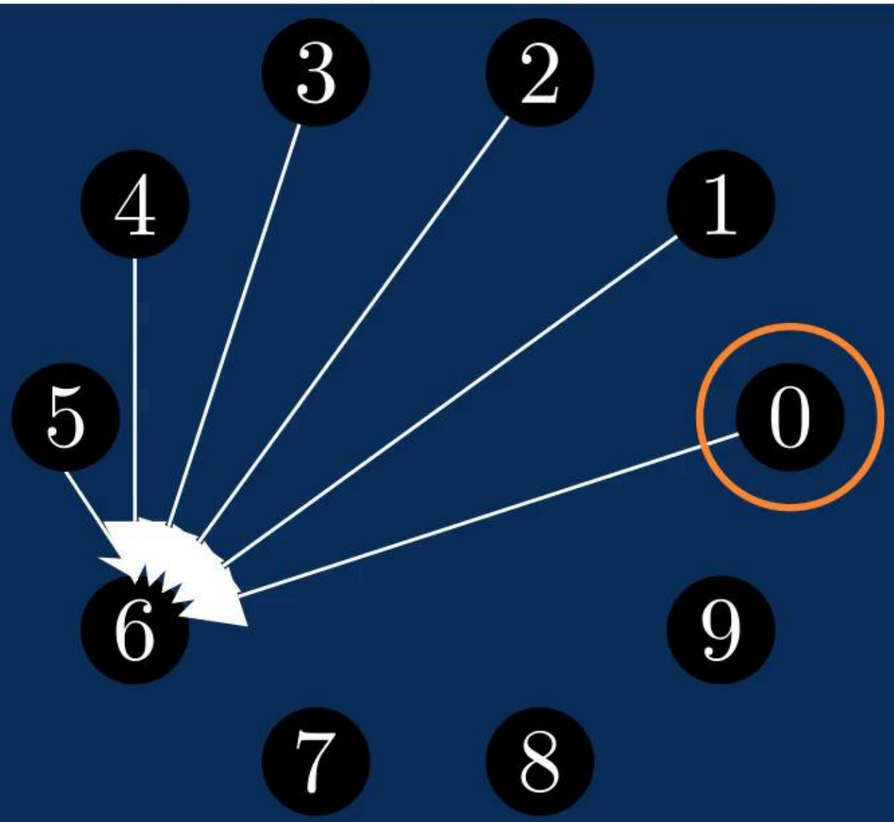
0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	-7	-1	-1	-1



Demo

0	1	2	3	4	5	6	7	8	9
6	6	6	6	6	6	-7	-1	-1	-1

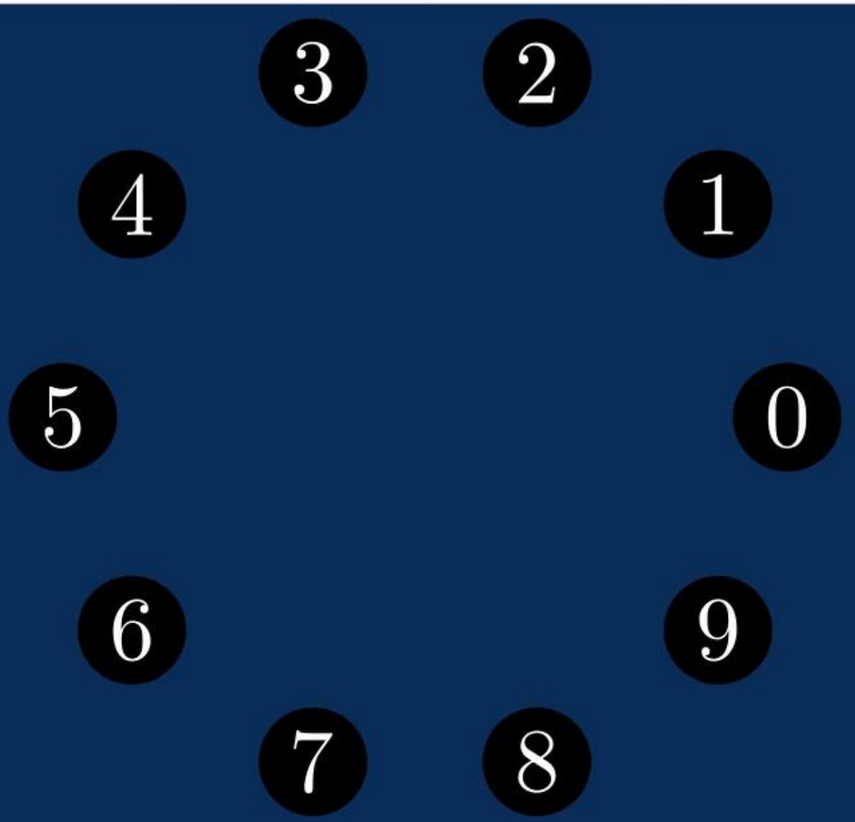
find(0)



Result

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

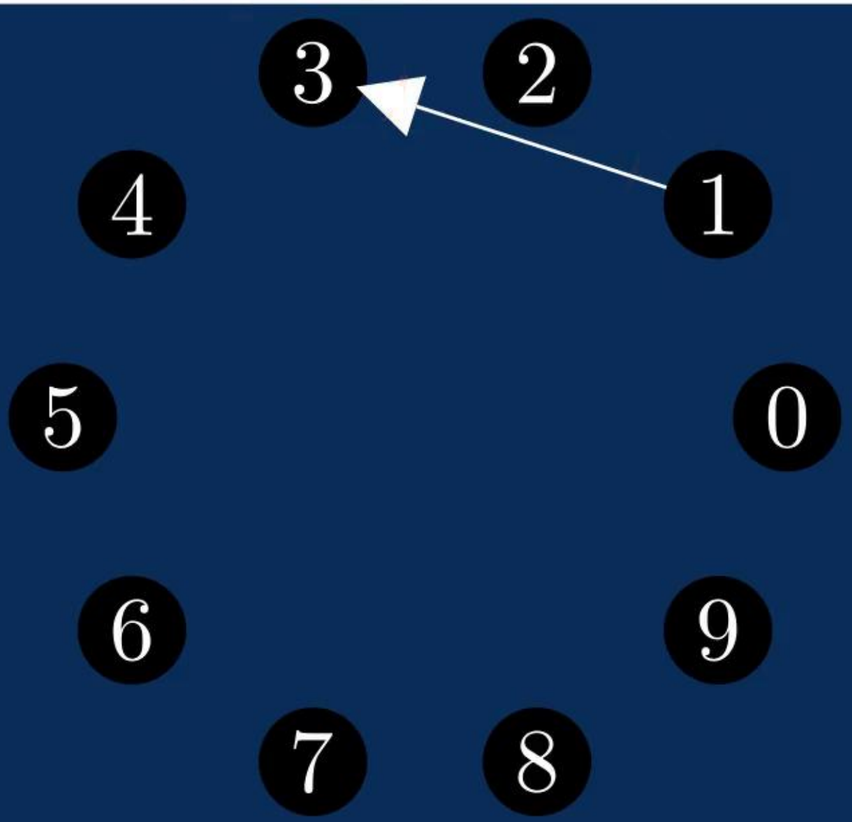
union(1,3)



Result

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-2	-1	-1	-1	-1	-1	-1

union(1,3)

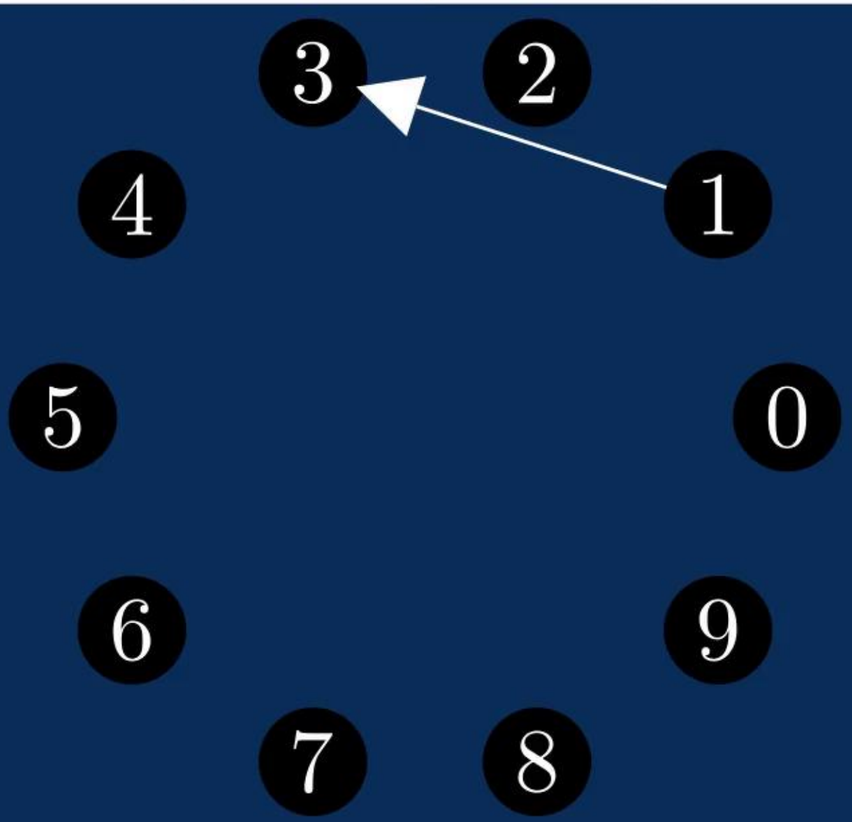


Result

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-2	-1	-1	-1	-1	-1	-1

union(1,3)

union(1,4)

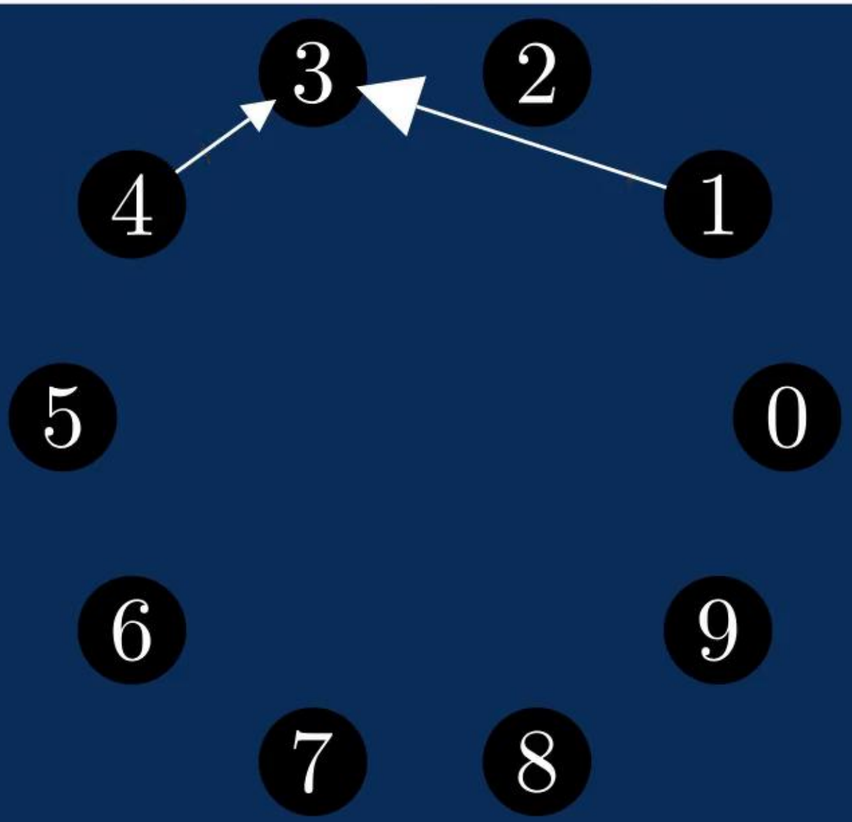


Result

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	-1	-1	-1	-1	-1

union(1,3)

union(1,4)



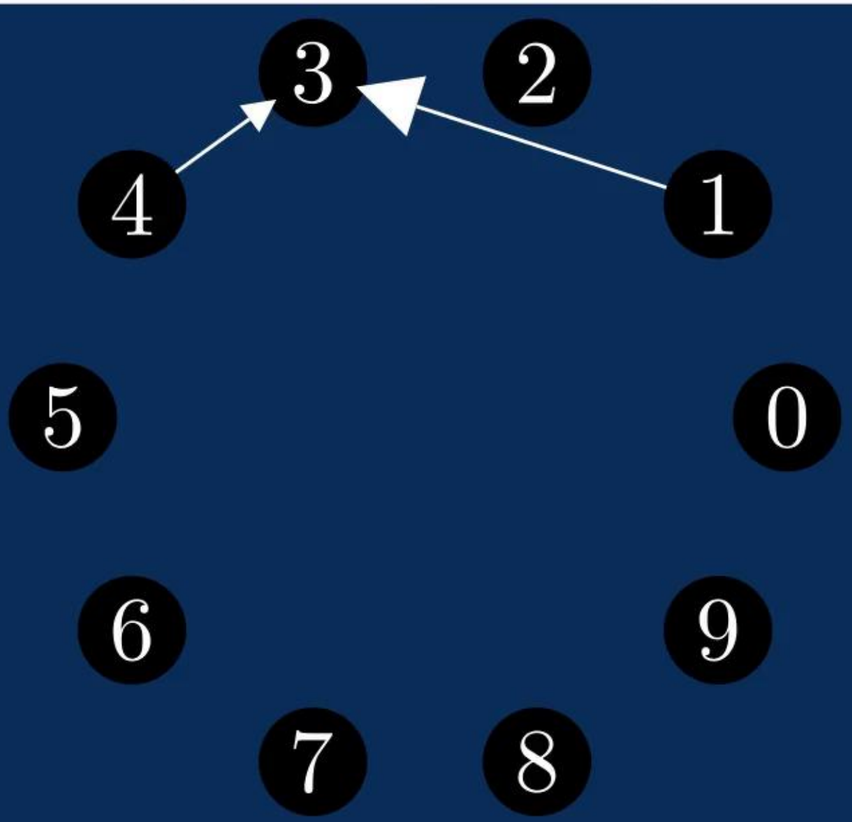
Result

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	-1	-1	-1	-1	-1

union(1,3)

union(1,4)

union(5,9)



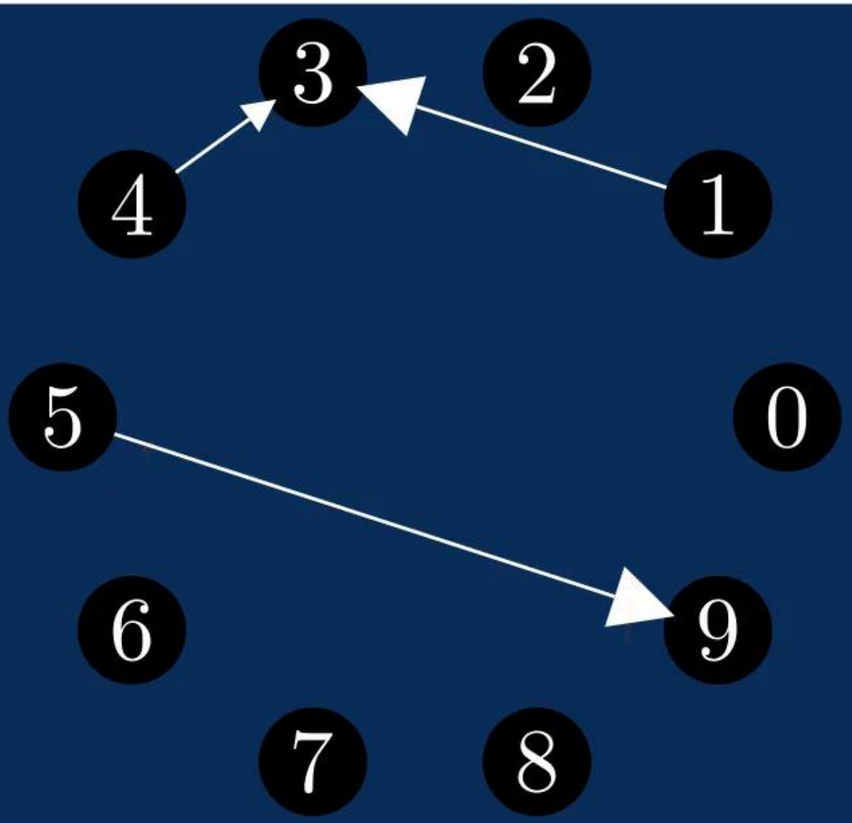
Result

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	-1	-1	-1	-2

union(1,3)

union(1,4)

union(5,9)



Result

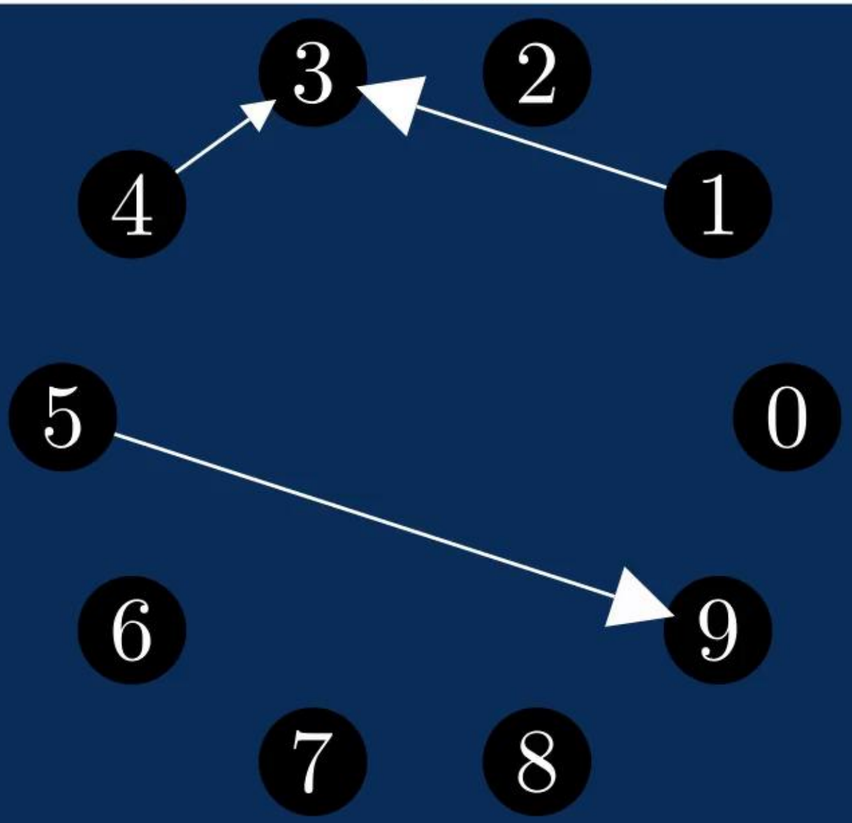
0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	-1	-1	-1	-2

union(1,3)

union(1,4)

union(5,9)

union(8,6)



Result

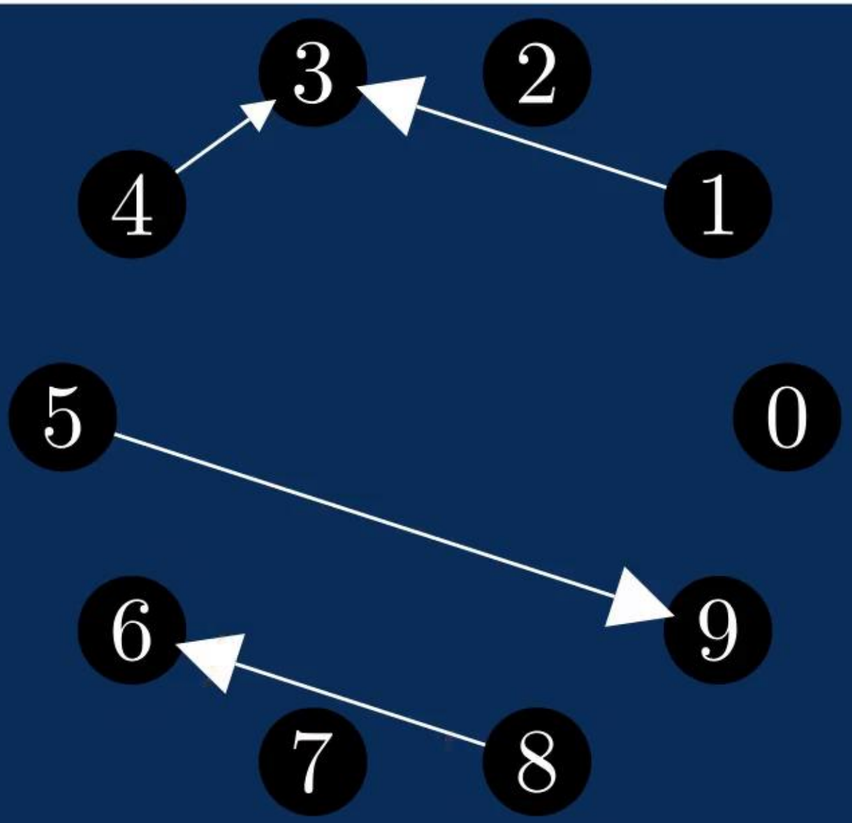
0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	-2	-1	6	-2

union(1,3)

union(1,4)

union(5,9)

union(8,6)



Result

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	-2	-1	6	-2

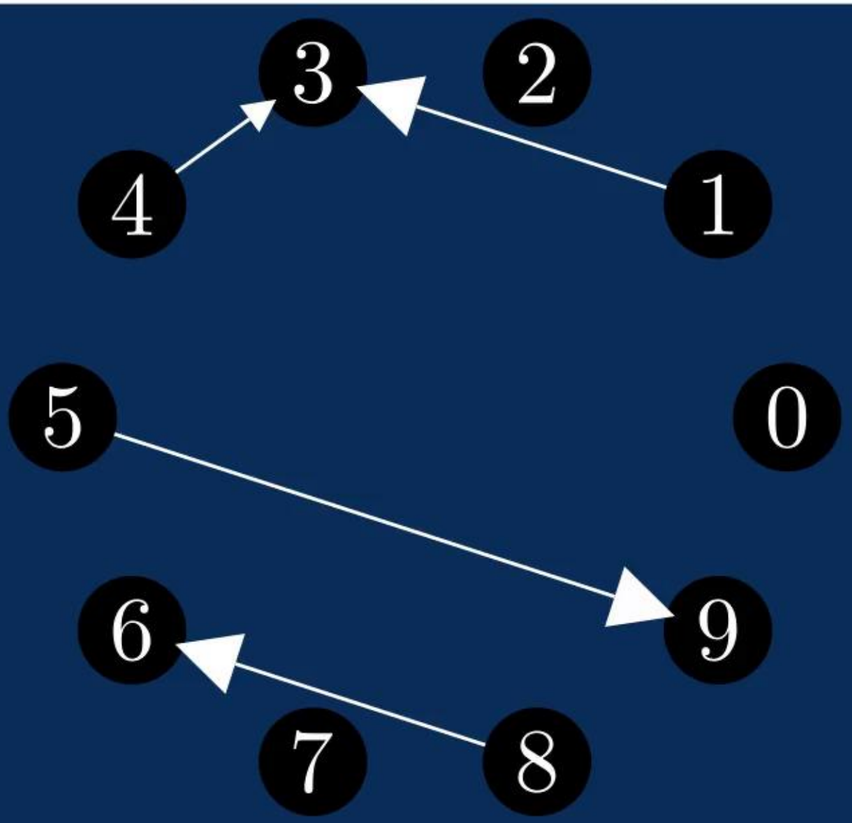
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



Result

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	9	-1	6	-4

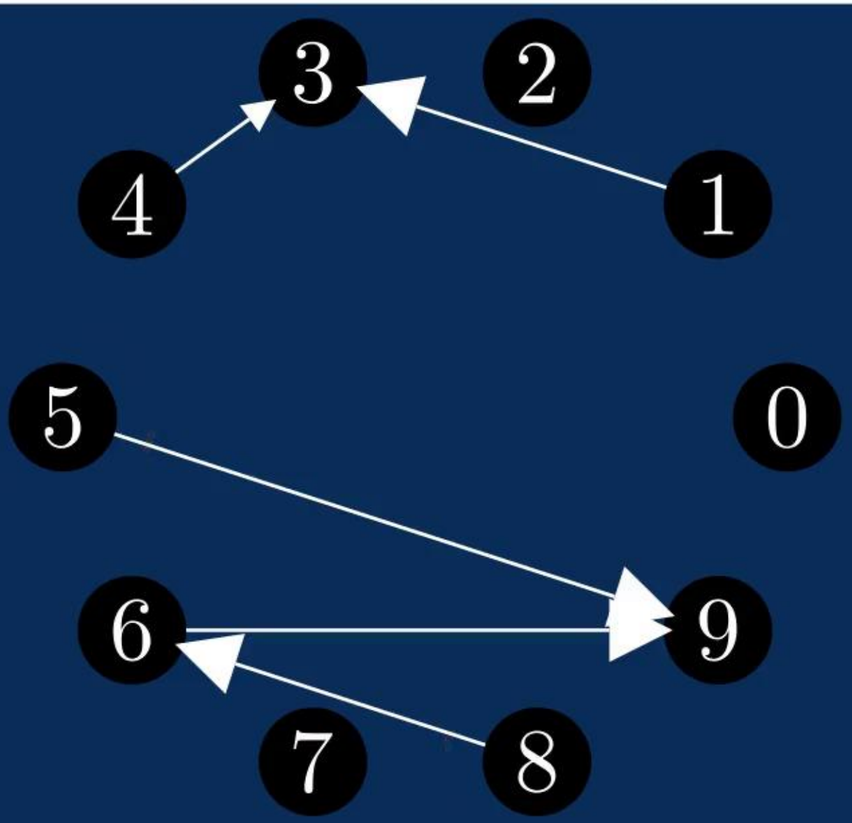
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



Result

0	1	2	3	4	5	6	7	8	9
-1	3	-1	-3	3	9	9	-1	6	-4

union(1,3)

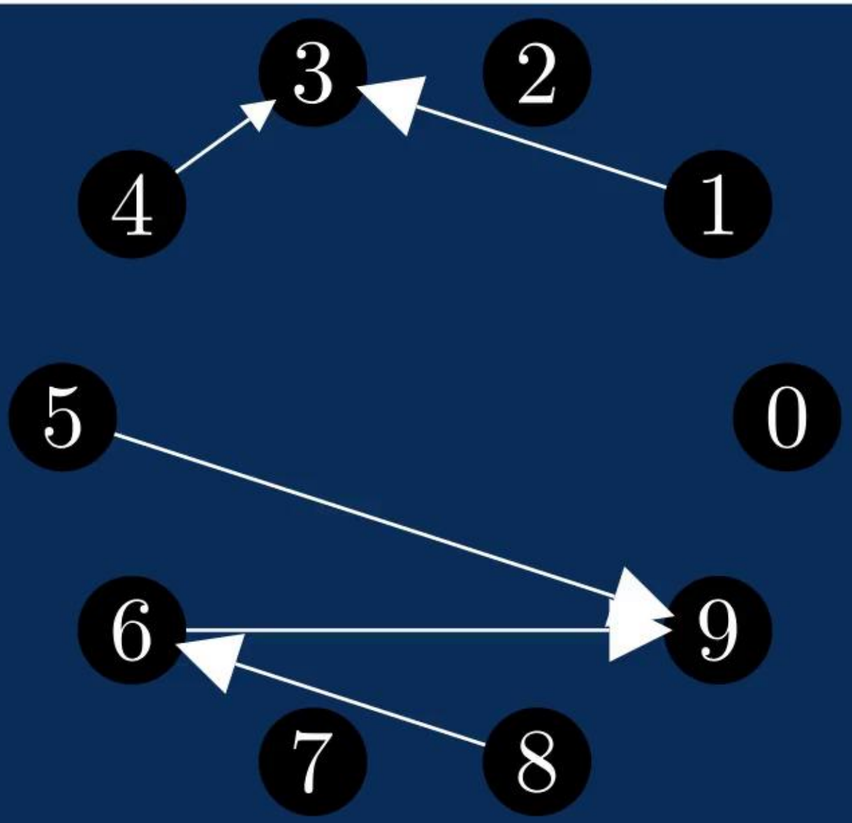
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Result

0	1	2	3	4	5	6	7	8	9
-1	3	-1	9	3	9	9	-1	9	-7

union(1,3)

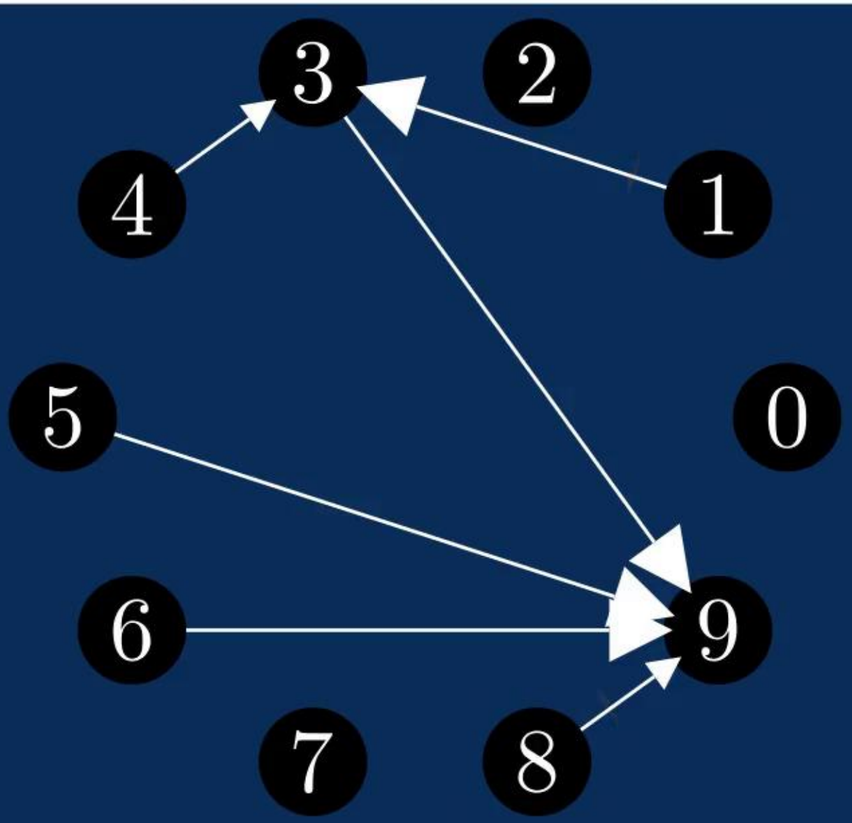
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Result

0	1	2	3	4	5	6	7	8	9
-1	9	-1	9	3	9	9	-1	9	-7

union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)

find(1)