

Disjoint Sets

CS 225 - Fall 2026

Mattox Beckman

-
- Use the up-tree data structure to model disjoint sets
 - Use union by size to keep the trees shallow
 - Use path compression to make this structure very efficient

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 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

What do these groups have in common?

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

Up Trees

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(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(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 }
```

Returns the representative

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 }
```

Returns the representative
Negative means root...

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 }
```

Returns the representative
Negative means root...
...so n is its own rep

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 }
```

Returns the representative
Negative means root...
...so n is its own rep

Otherwise ask the parent



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



```
1 int unite(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])  
8         swap(ar,br);  
9  
10    ds[br] += ds[ar];  
11    ds[ar] = br;  
12  
13    return ds[br];  
14 }
```



```
1 int unite(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])  
8         swap(ar,br);  
9  
10    ds[br] += ds[ar];  
11    ds[ar] = br;  
12  
13    return ds[br];  
14 }
```

Find both representatives



```
1 int unite(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])  
8         swap(ar,br);  
9  
10    ds[br] += ds[ar];  
11    ds[ar] = br;  
12  
13    return ds[br];  
14 }
```

Find both representatives

Already the same set

```
1 int unite(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])  
8         swap(ar,br);  
9  
10    ds[br] += ds[ar];  
11    ds[ar] = br;  
12  
13    return ds[br];  
14 }
```

Find both representatives

Already the same set

ds[r] is -size: br is now bigger



```
1 int unite(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])  
8         swap(ar,br);  
9  
10    ds[br] += ds[ar];  
11    ds[ar] = br;  
12  
13    return ds[br];  
14 }
```

Find both representatives

Already the same set

ds[r] is -size: br is now bigger

Add sizes; small points to big



```
1 int unite(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])  
8         swap(ar,br);  
9  
10    ds[br] += ds[ar];  
11    ds[ar] = br;  
12  
13    return ds[br];  
14 }
```

Find both representatives

Already the same set

ds[r] is -size: br is now bigger

Add sizes; small points to big

Return -(new size)

Example Run

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

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

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

union(1,3)

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

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

union(1,3)



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

union(1,3)

union(1,4)



Predict: which root
ends up on top?

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

union(1,3)

union(1,4)



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

union(1,3)

union(1,4)

union(5,9)

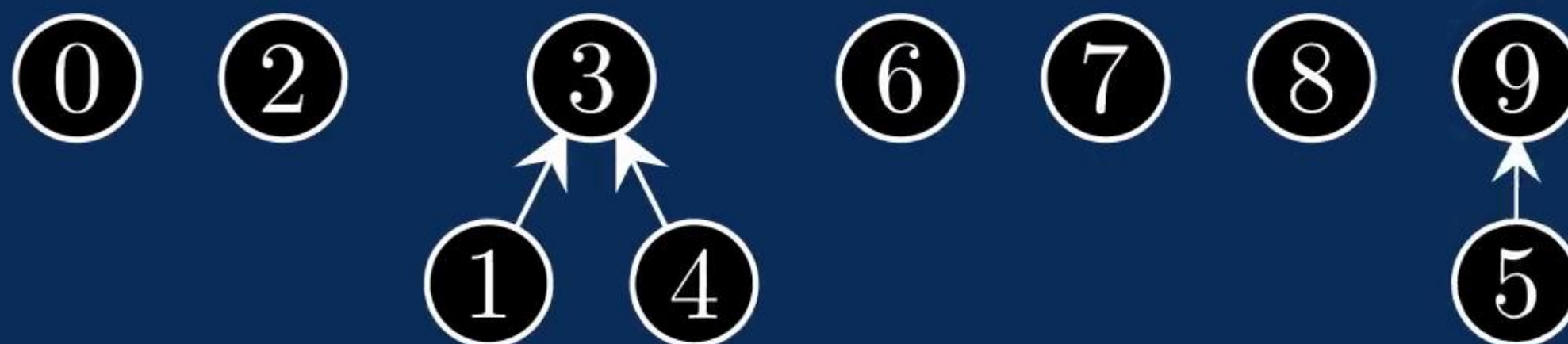


⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -1	⁷ -1	⁸ -1	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	--------------------	--------------------

union(1,3)

union(1,4)

union(5,9)



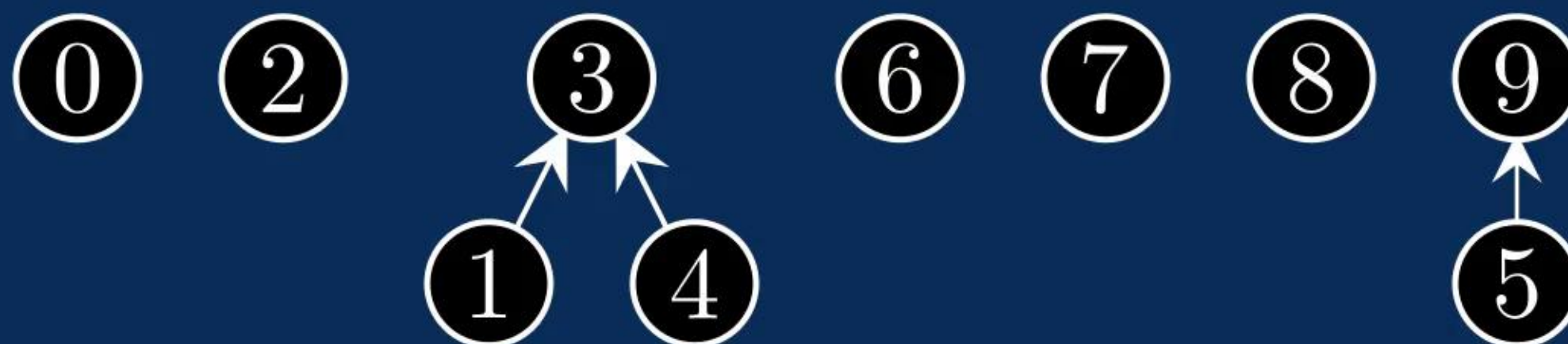
⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -1	⁷ -1	⁸ -1	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	--------------------	--------------------

union(1,3)

union(1,4)

union(5,9)

union(8,6)



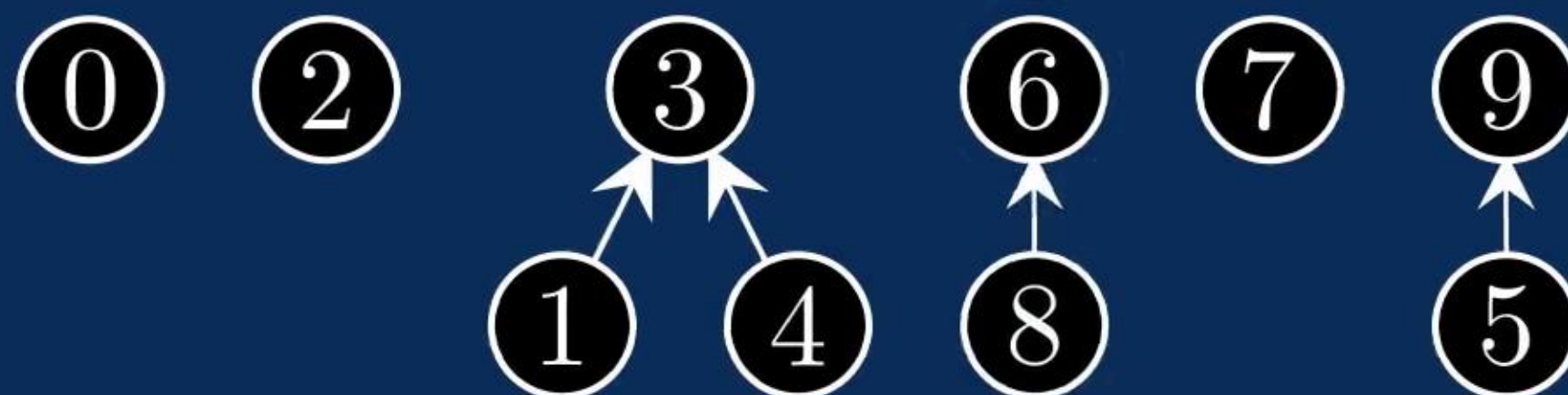
⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -2	⁷ -1	⁸ 6	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	-------------------	--------------------

union(1,3)

union(1,4)

union(5,9)

union(8,6)



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

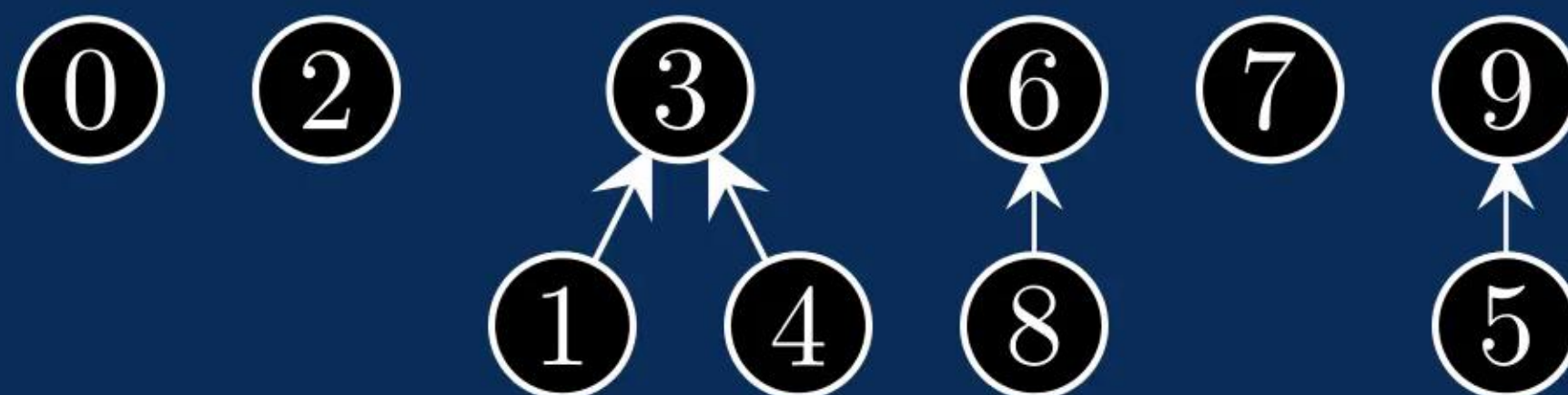
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



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

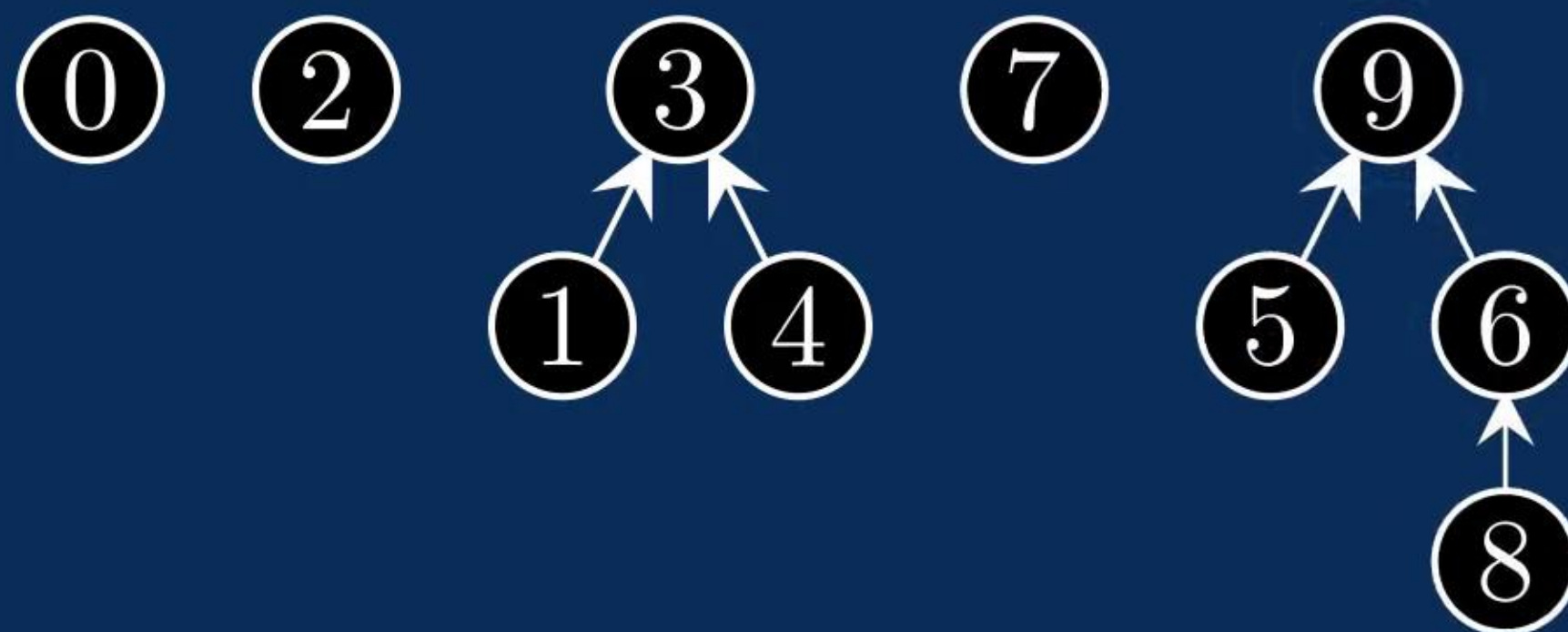
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



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

union(1,3)

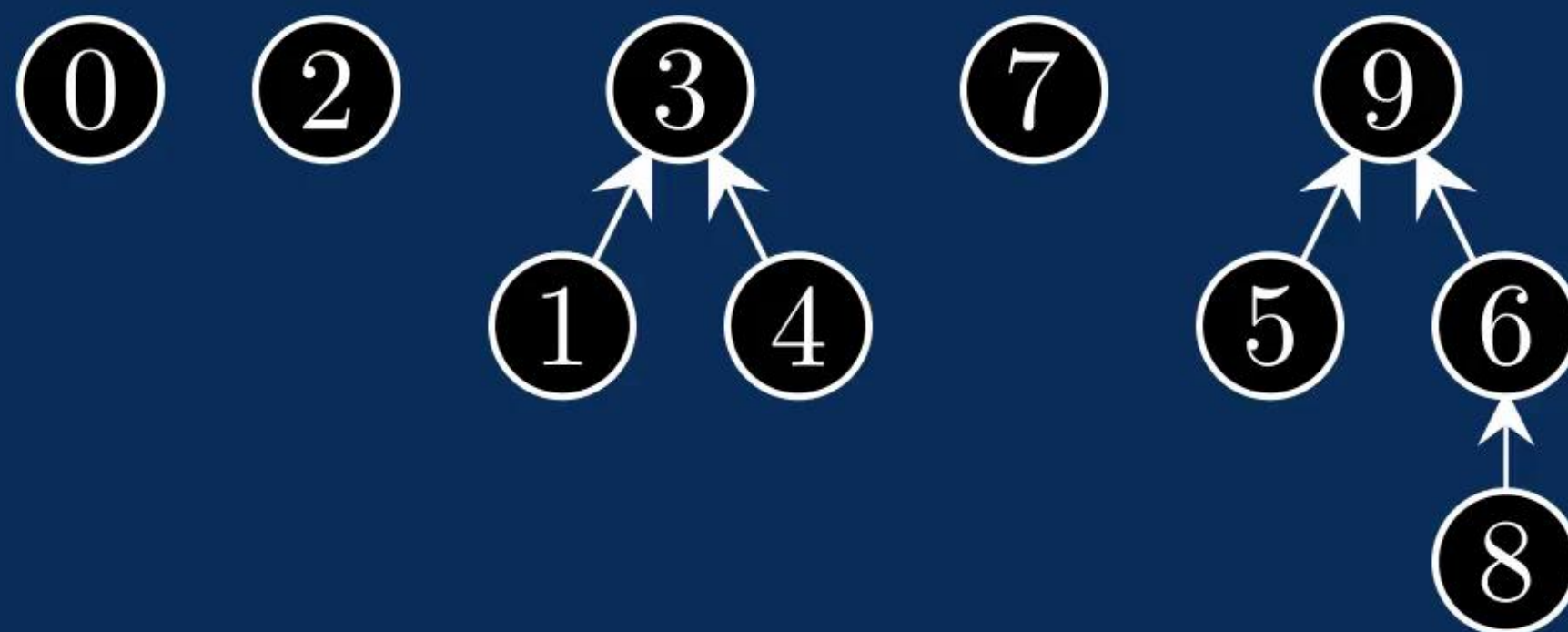
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Predict: what will
ds look like?

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

union(1,3)

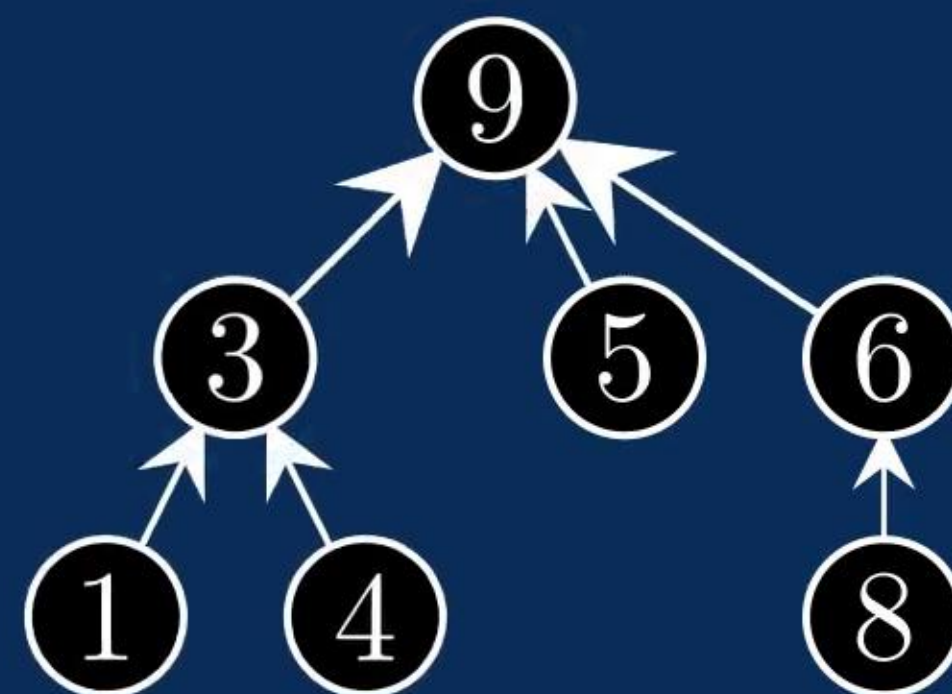
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



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

union(1,3)

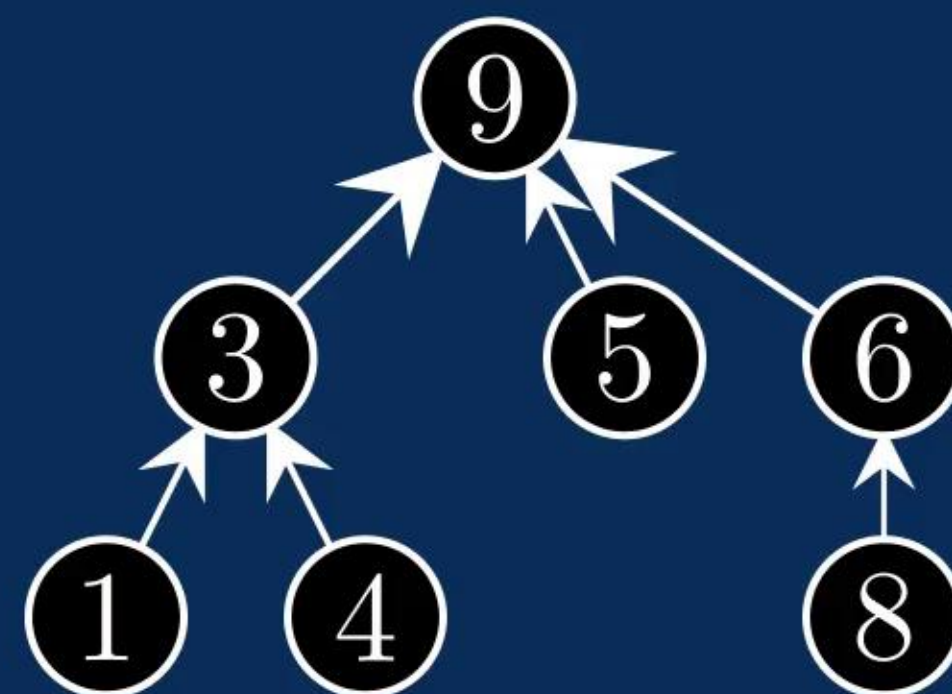
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Union by Size

Naive: `ds[ar] = br`

Union by size

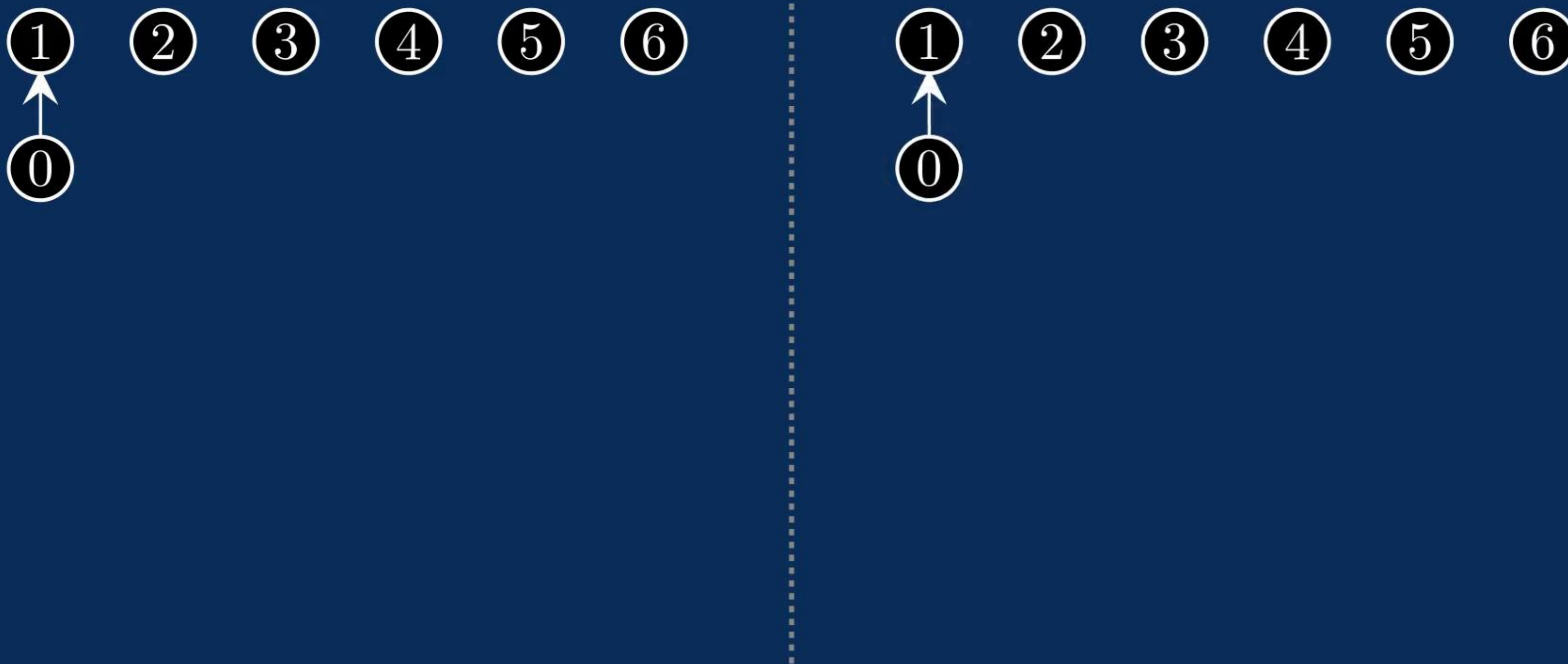
① ② ③ ④ ⑤ ⑥

① ② ③ ④ ⑤ ⑥

Naive: `ds[ar] = br`

Union by size

`union(0,1)`



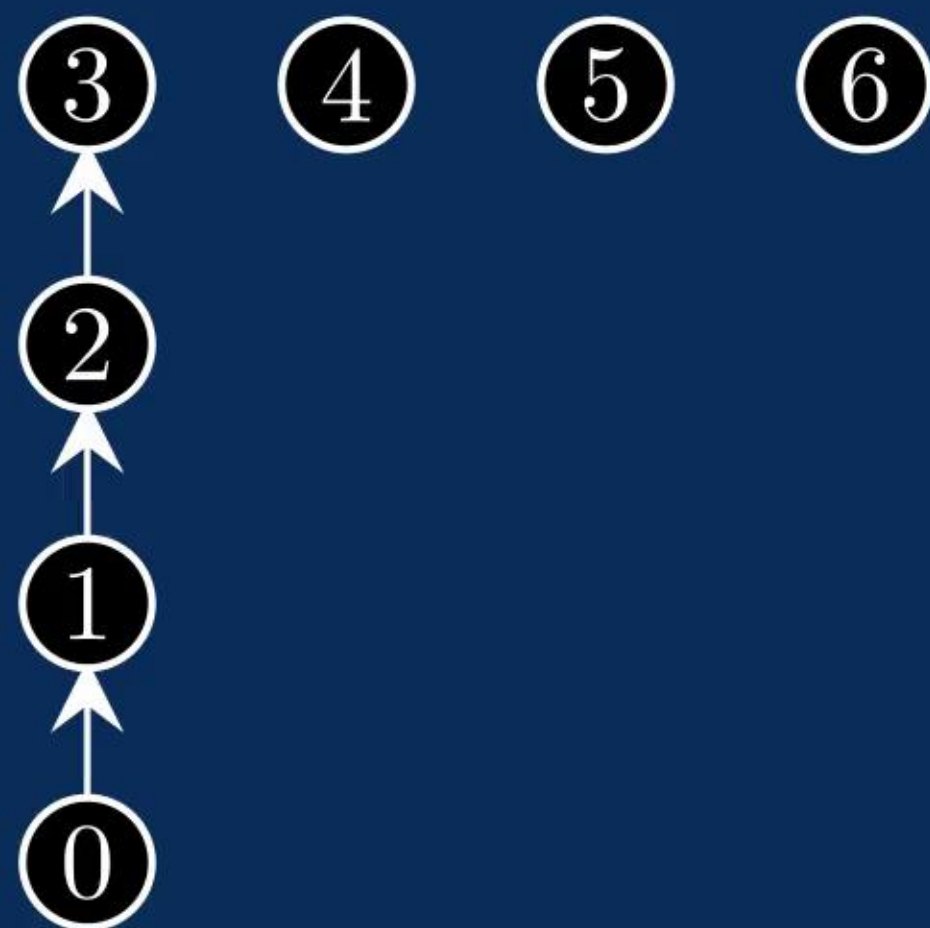
Naive: $ds[ar] = br$



$union(0,2)$

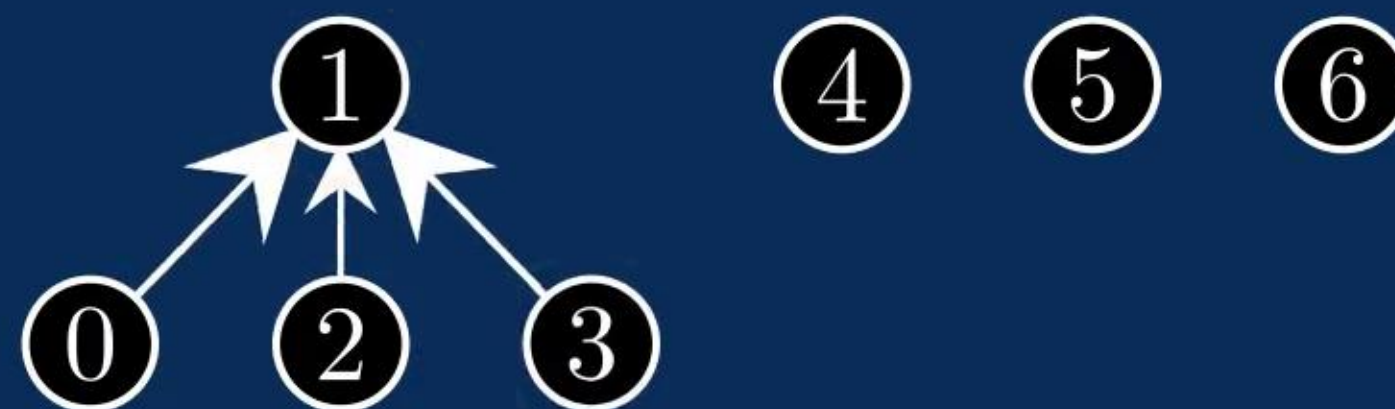


Naive: $ds[ar] = br$

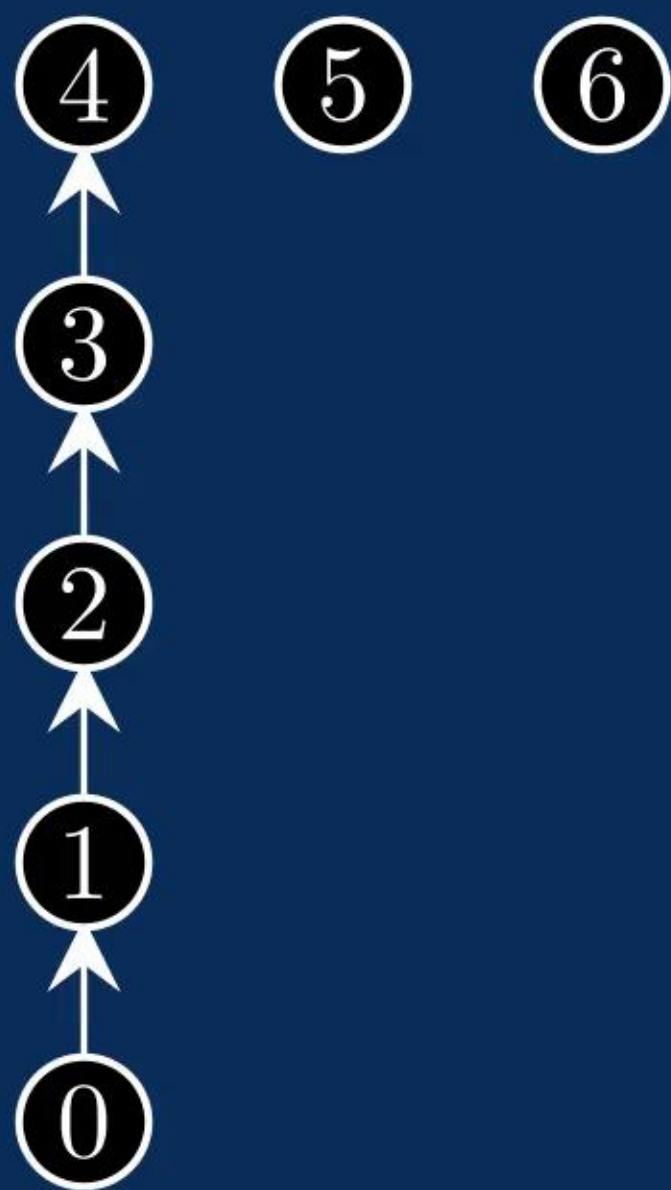


$union(0,3)$

Union by size

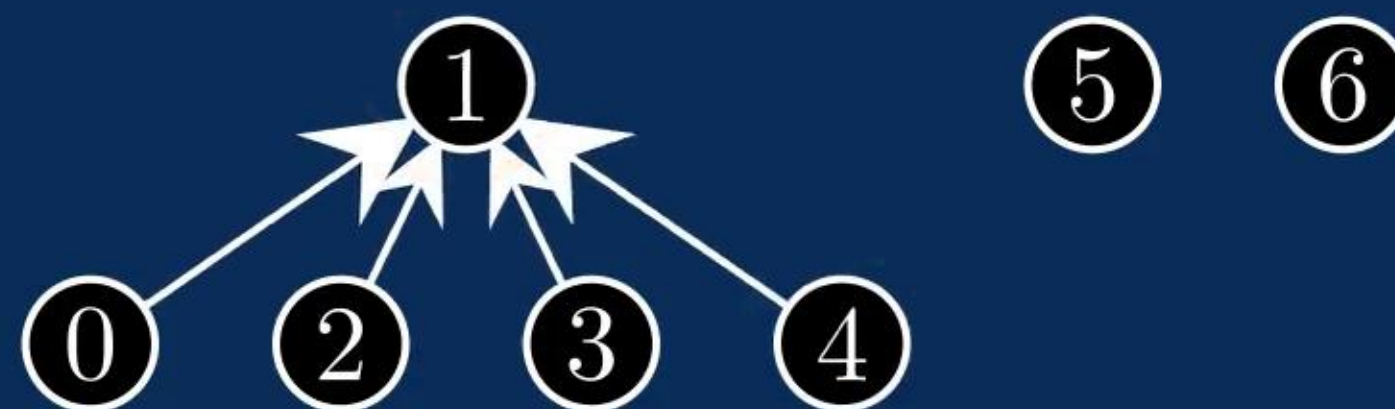


Naive: $ds[ar] = br$

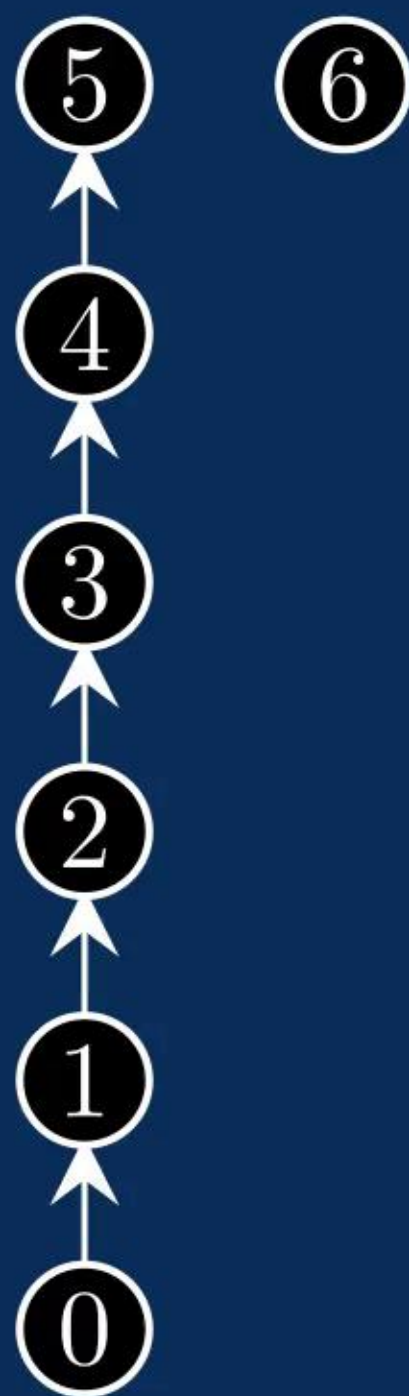


$union(0,4)$

Union by size

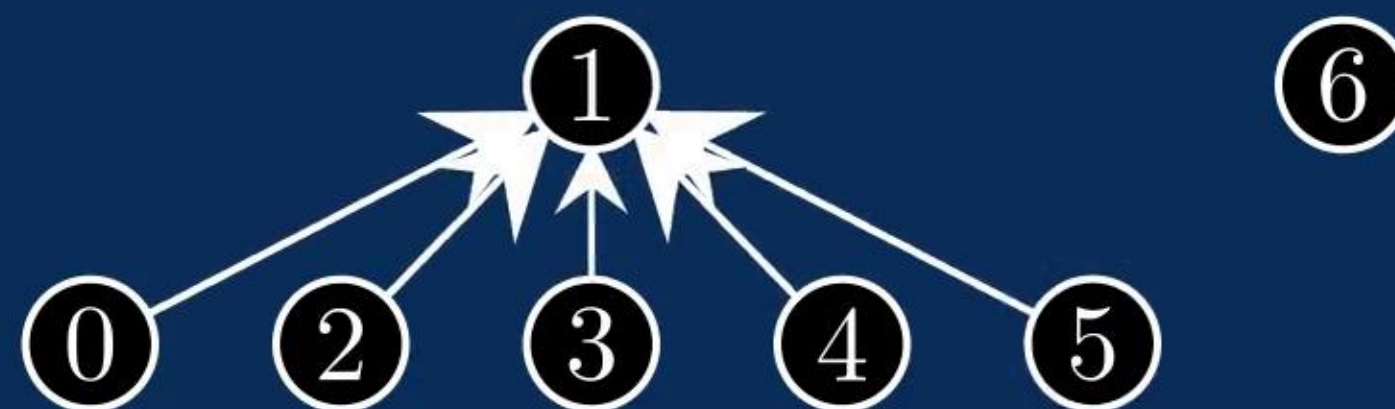


Naive: $ds[ar] = br$



$union(0,5)$

Union by size



Naive: $ds[ar] = br$

$union(0,6)$

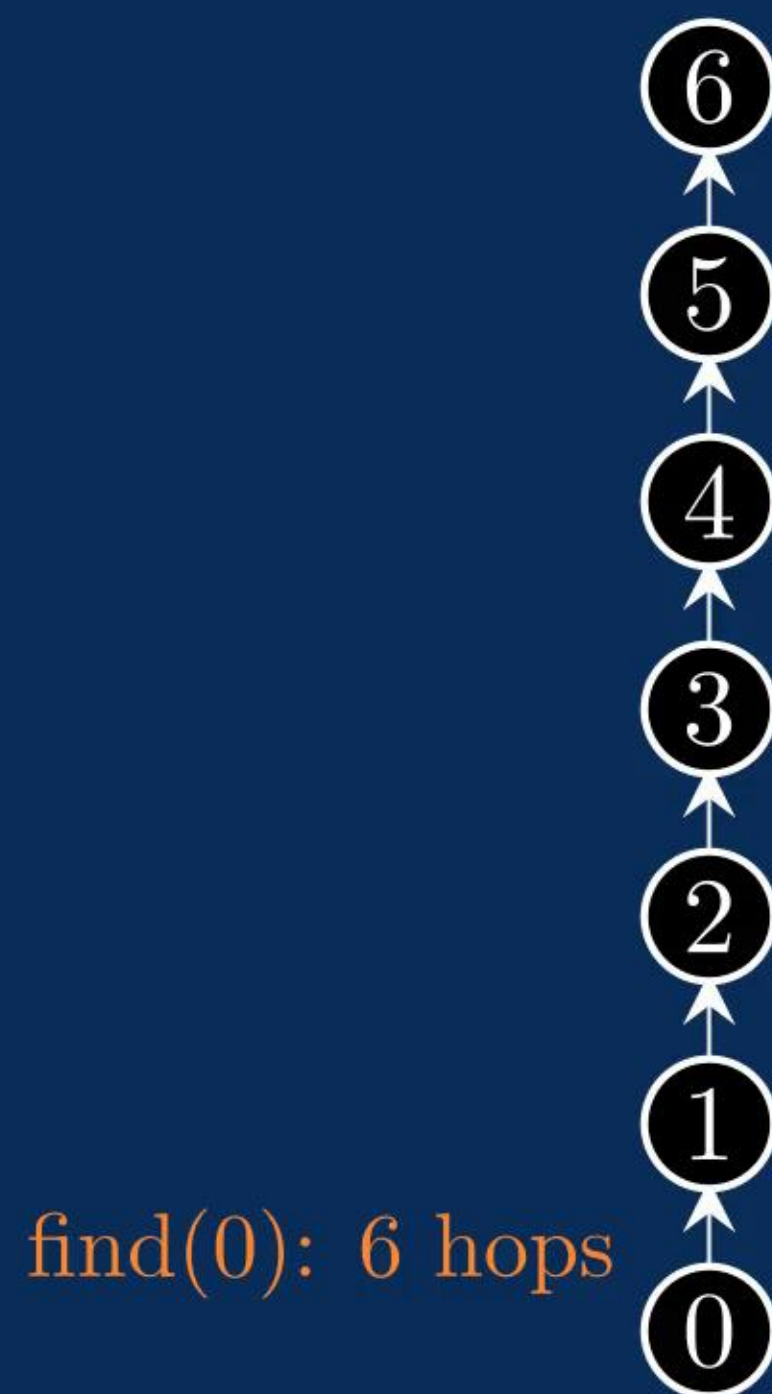
Union by size



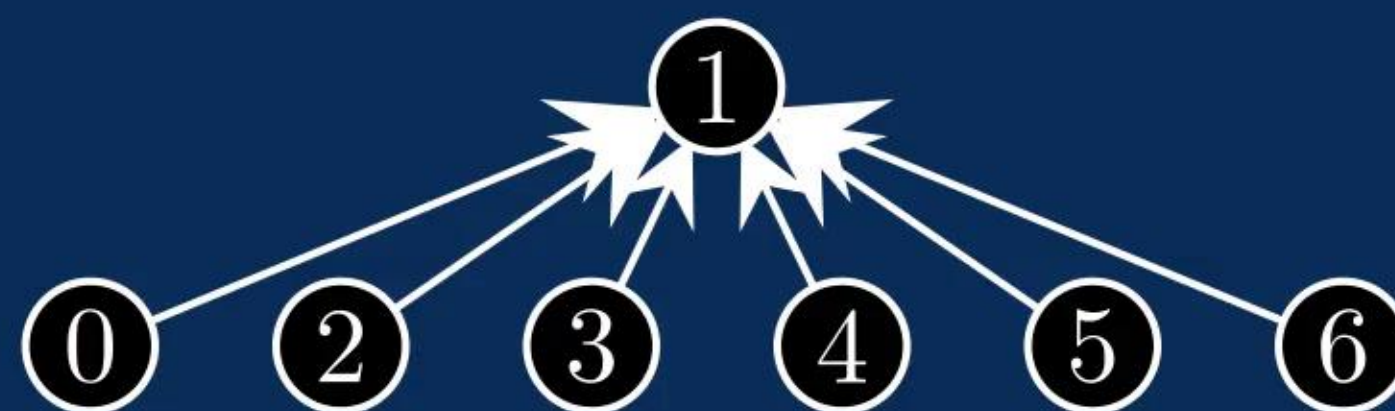
Naive: $ds[ar] = br$

$union(0,6)$

Union by size



$find(0)$: 6 hops



$find(0)$: 1 hop

Claim: with union by size, height is at most $\log_2 n$

Claim: with union by size, height is at most $\log_2 n$

A node gets deeper only when its root gets a parent

- That only happens when its tree joins one *at least as big*
- So the size of its tree at least *doubles*

Claim: with union by size, height is at most $\log_2 n$

A node gets deeper only when its root gets a parent

- That only happens when its tree joins one *at least as big*
- So the size of its tree at least *doubles*

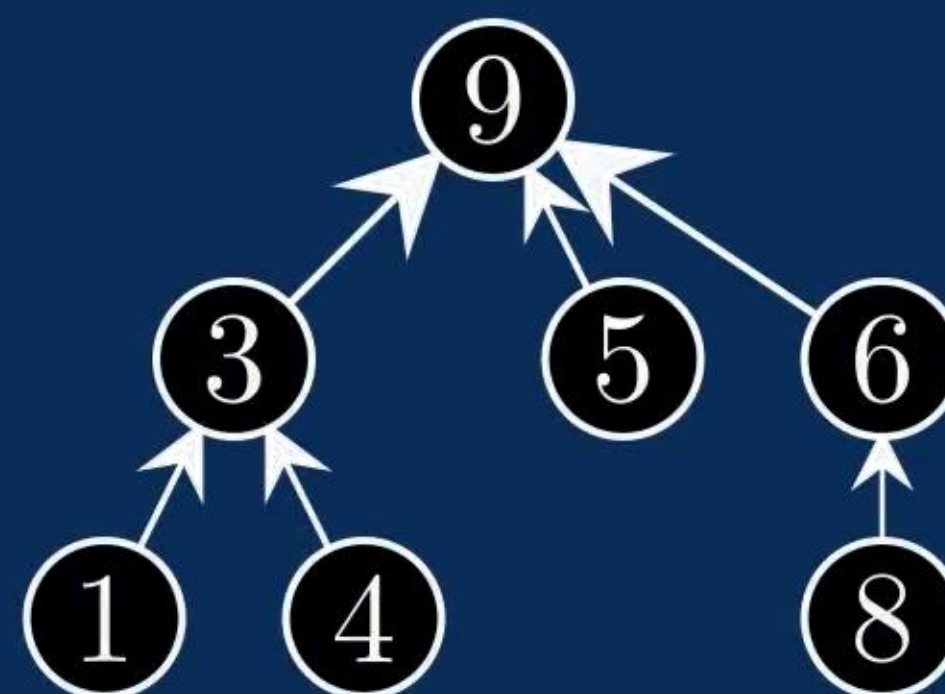
Sizes can double at most $\log_2 n$ times

- So `find` is $\mathcal{O}(\log n)$

Path Compression

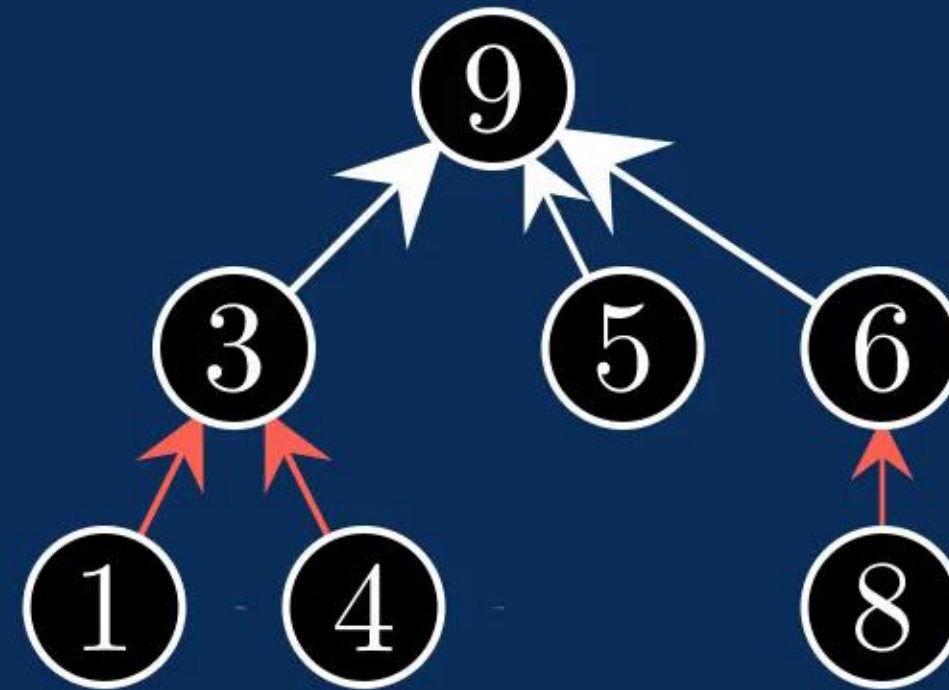
⁰ -1	¹ 3	² -1	³ 9	⁴ 3	⁵ 9	⁶ 9	⁷ -1	⁸ 6	⁹ -7
--------------------	-------------------	--------------------	-------------------	-------------------	-------------------	-------------------	--------------------	-------------------	--------------------

① ② ⑦



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

① ② ⑦



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

① 0 ② 2 ③ 7



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

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

We already visit every node on the path...

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

① ② 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 }
```

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

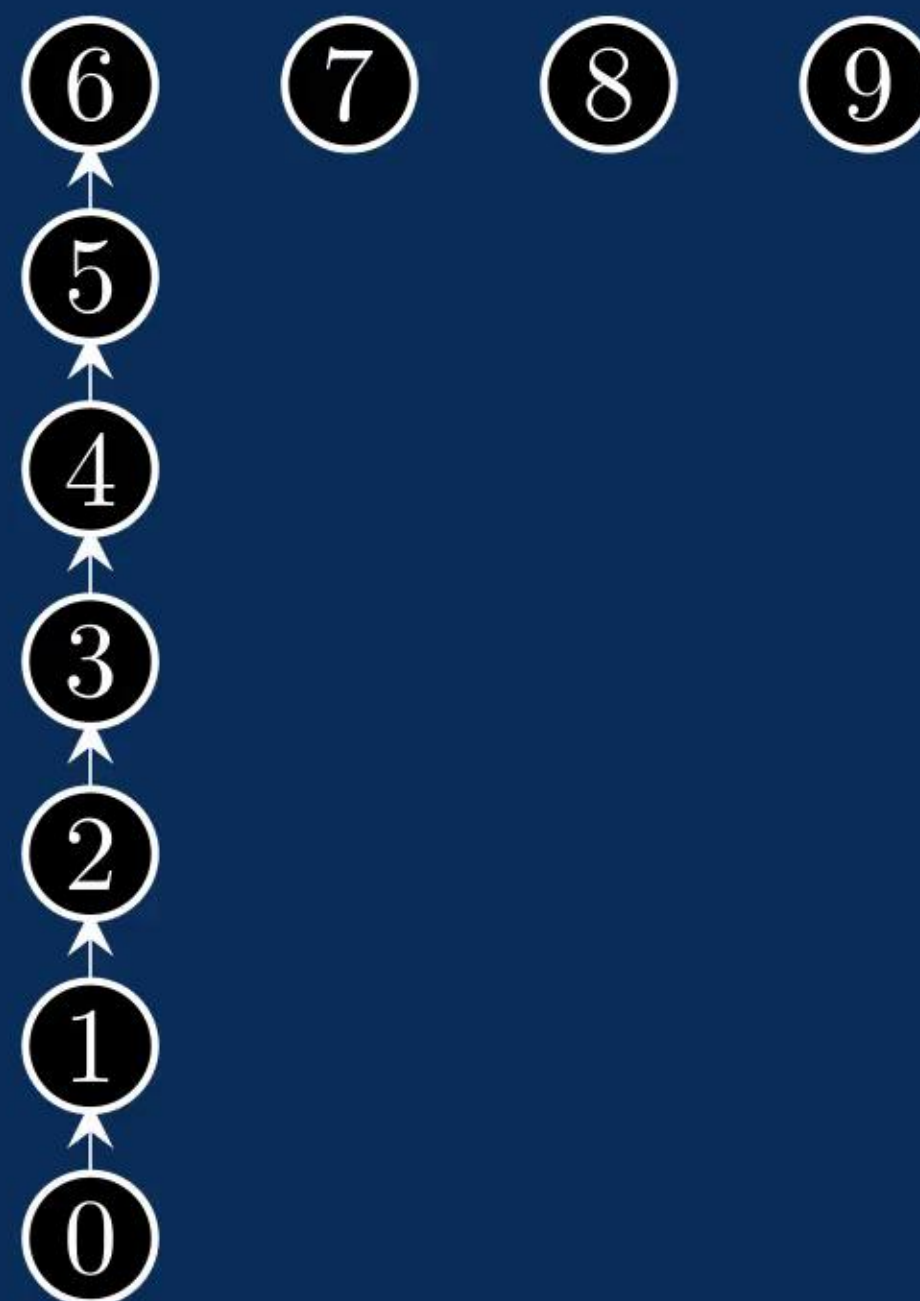
① ② ⑦



```
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 }
```

...so point each one
directly at the root

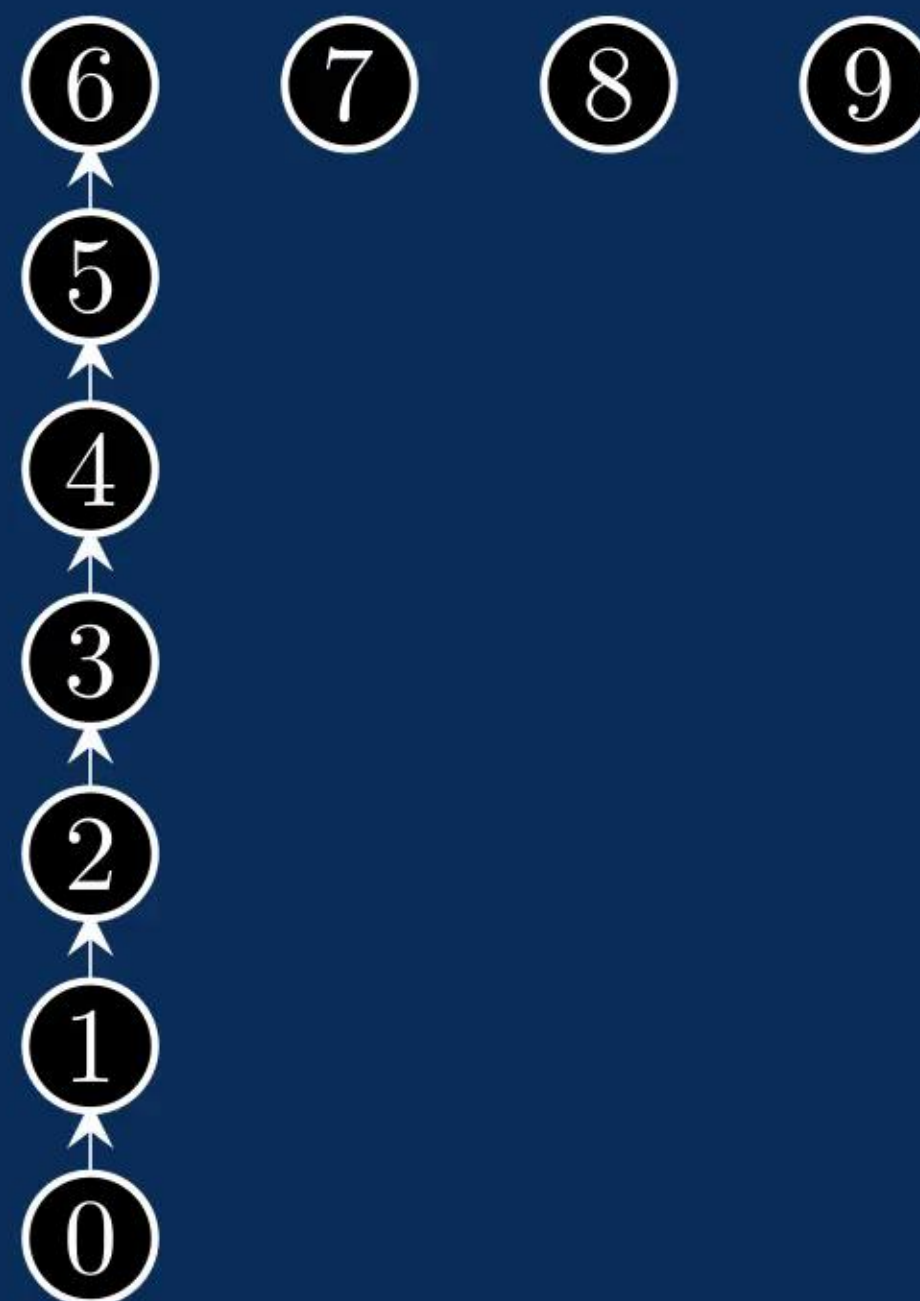
⁰ 1	¹ 2	² 3	³ 4	⁴ 5	⁵ 6	⁶ -7	⁷ -1	⁸ -1	⁹ -1
-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	--------------------	--------------------	--------------------	--------------------



(Remember naive union?)

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

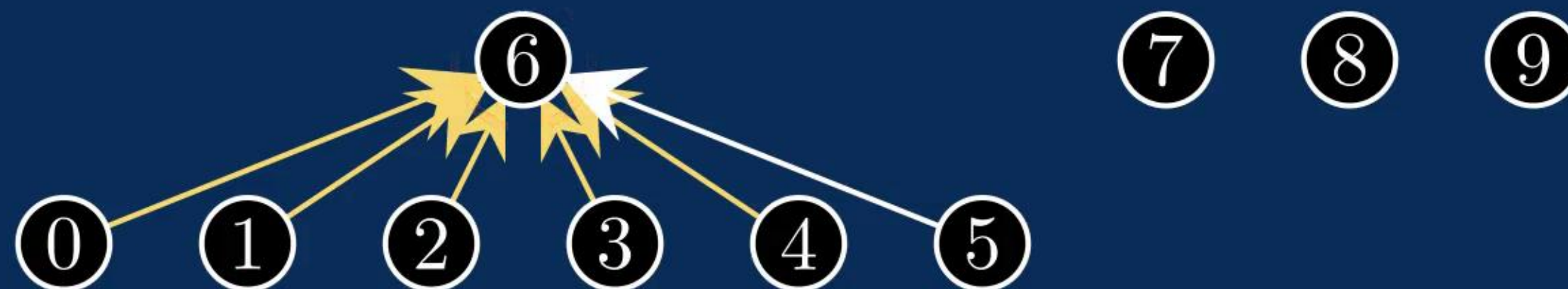
find(0)



(Remember naive union?)

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

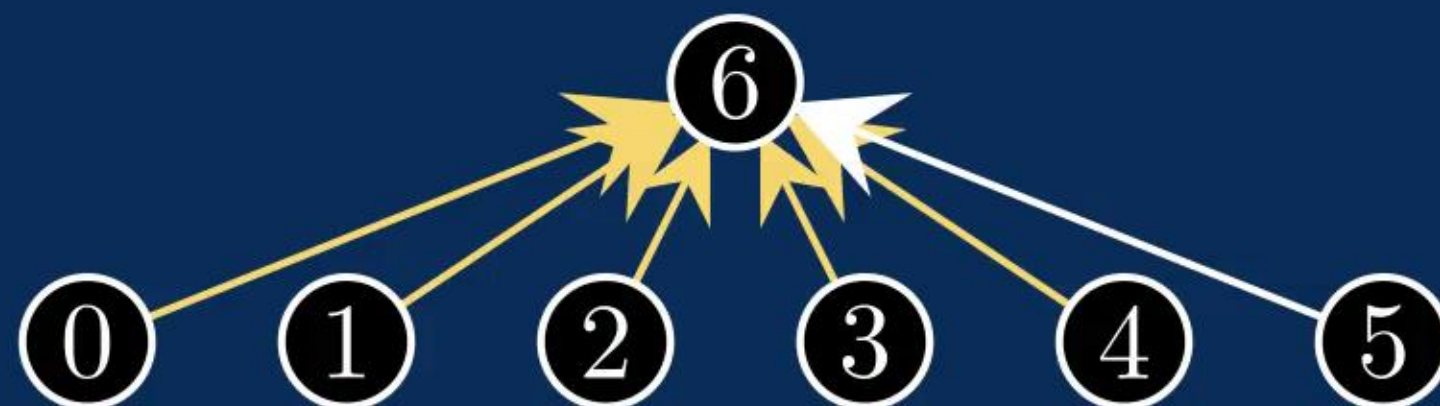
find(0)



(Remember naive union?)

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

find(0)



(Remember naive union?)

Next find(0): 1 hop

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

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

Yellow = changed by compression

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

union(1,3)

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

Yellow = changed by compression

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

union(1,3)



Yellow = changed by compression

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

union(1,3)

union(1,4)



Yellow = changed by compression

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)



Yellow = changed by compression

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)



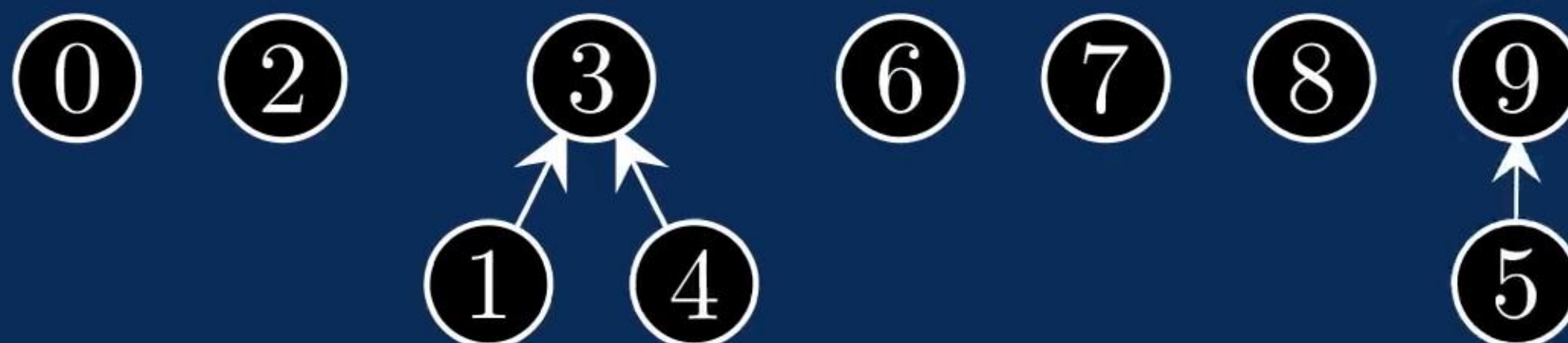
Yellow = changed by compression

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

union(1,3)

union(1,4)

union(5,9)



Yellow = changed by compression

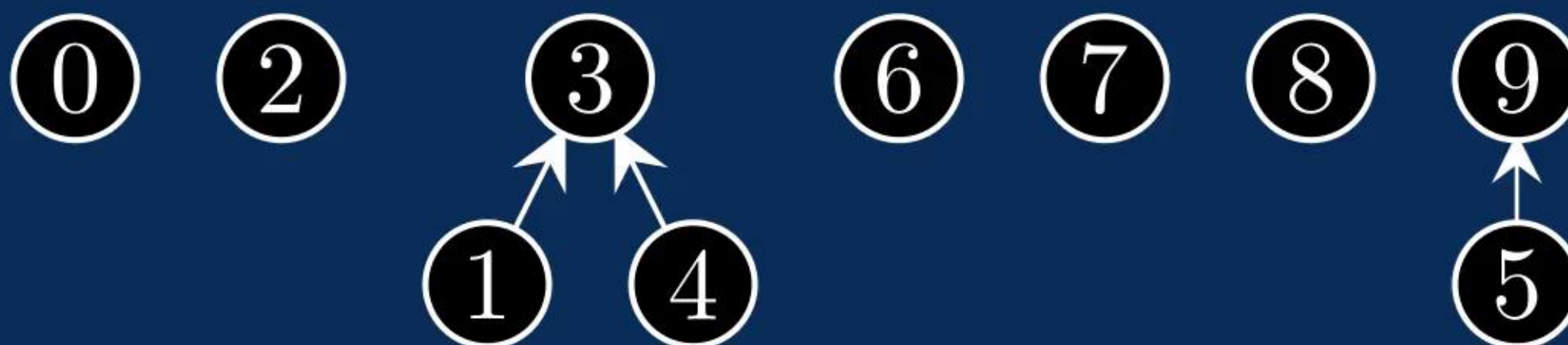
⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -1	⁷ -1	⁸ -1	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	--------------------	--------------------

union(1,3)

union(1,4)

union(5,9)

union(8,6)



Yellow = changed by compression

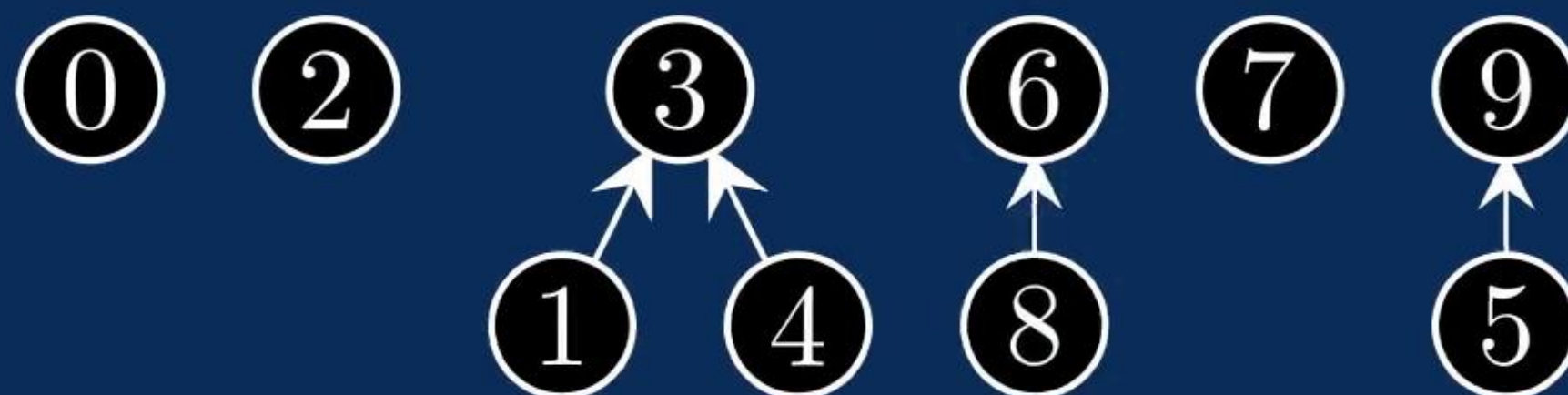
⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -2	⁷ -1	⁸ 6	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	-------------------	--------------------

union(1,3)

union(1,4)

union(5,9)

union(8,6)



Yellow = changed by compression

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

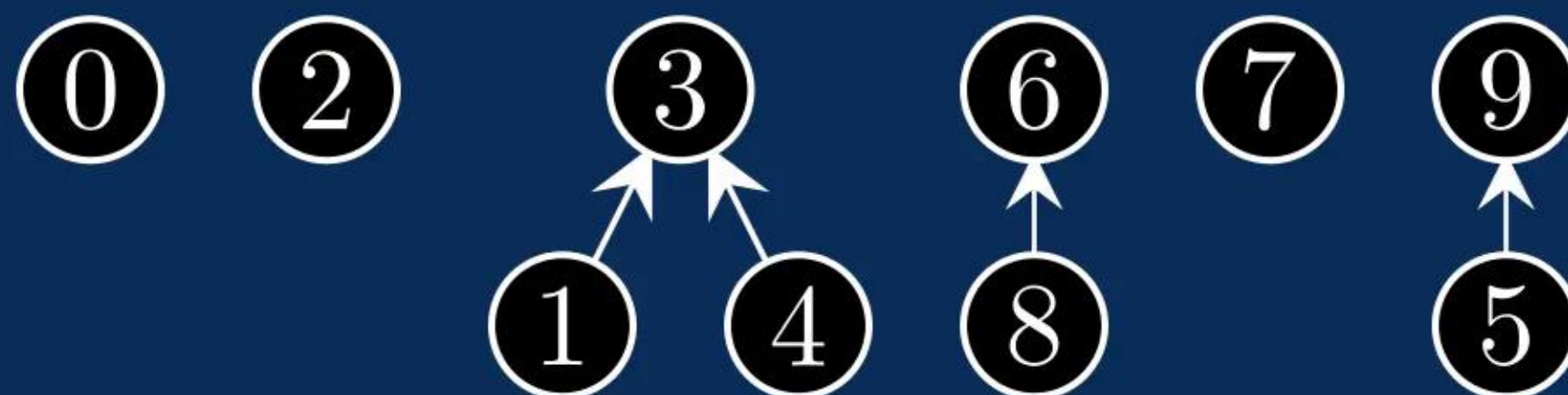
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



Yellow = changed by compression

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

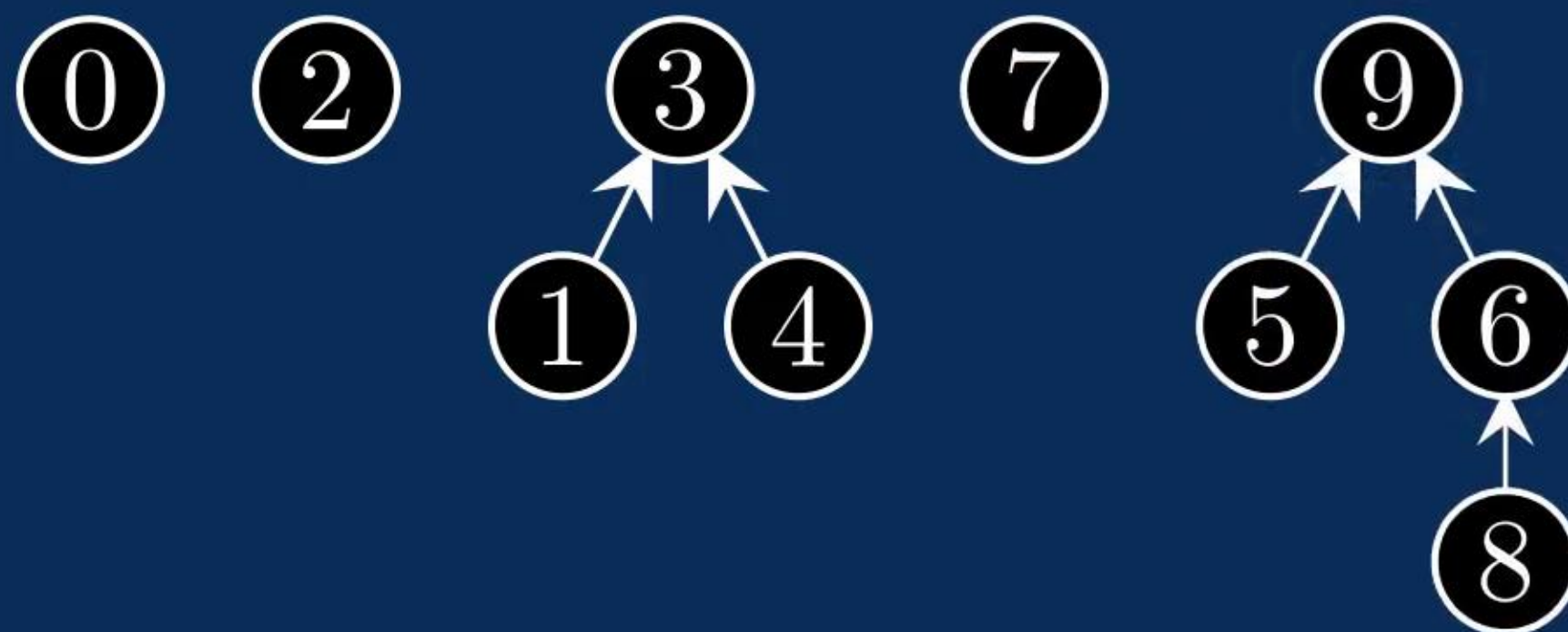
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



Yellow = changed by compression

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

union(1,3)

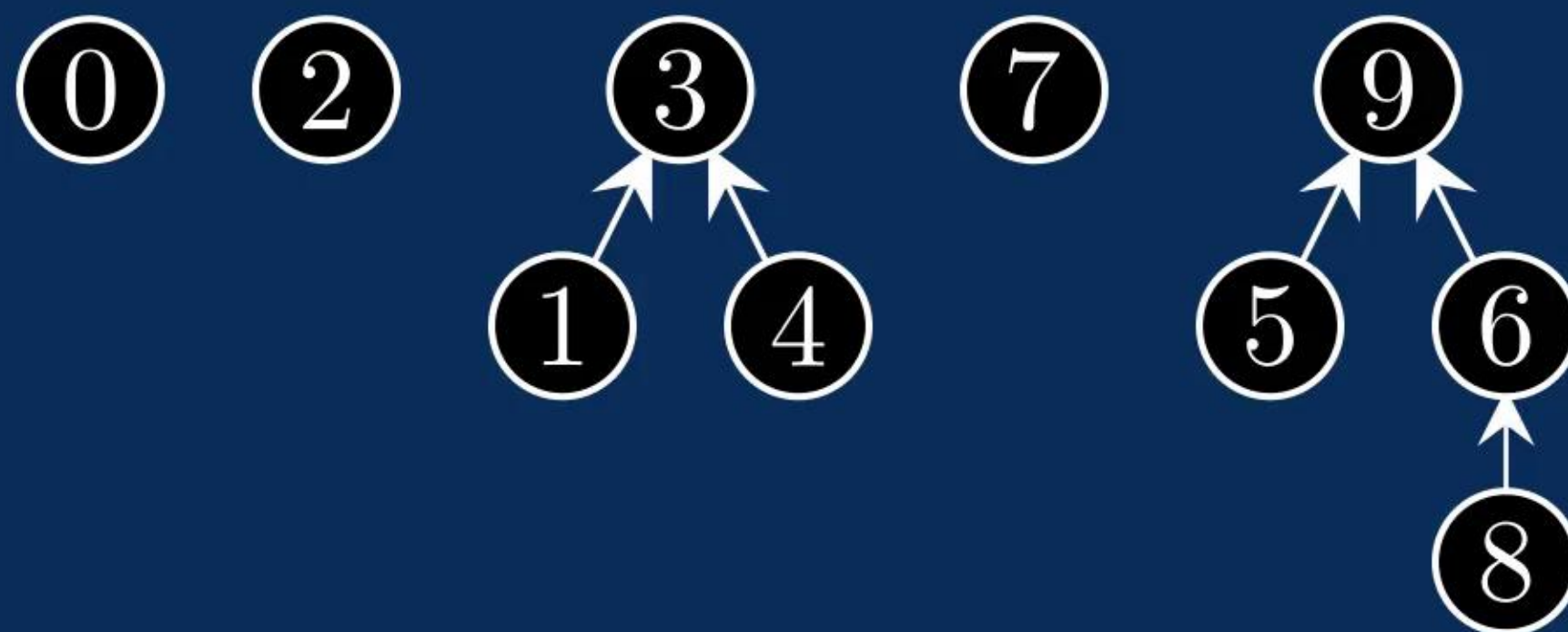
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Yellow = changed by compression

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

union(1,3)

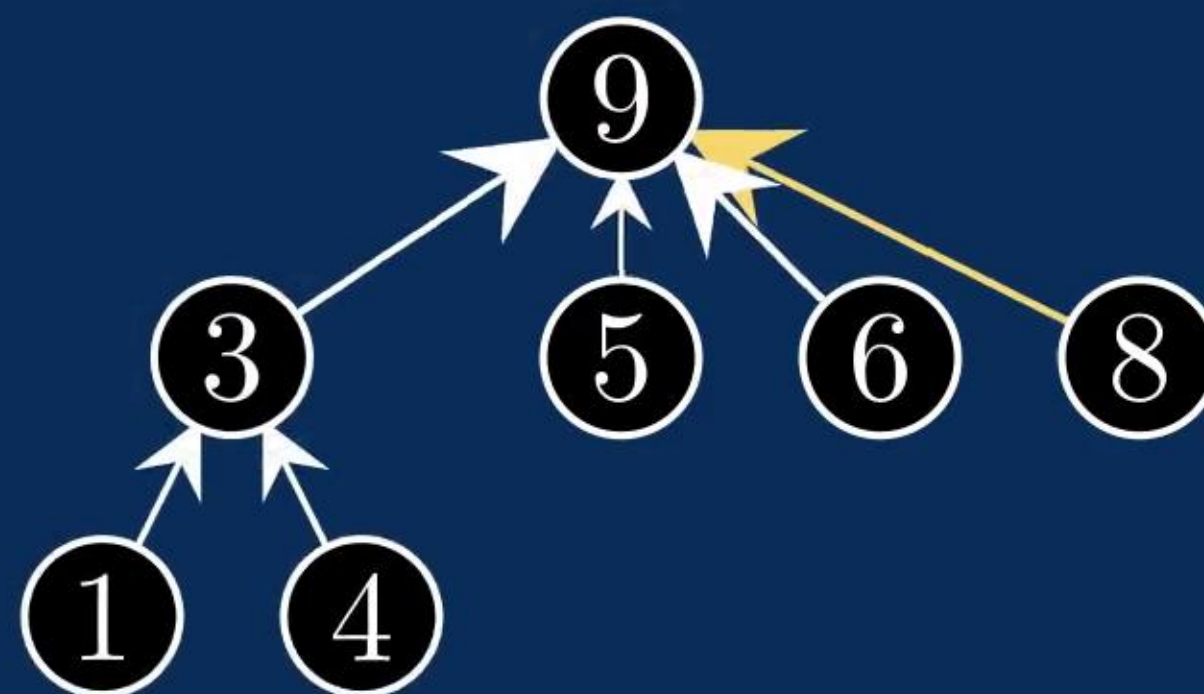
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Yellow = changed by compression

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)

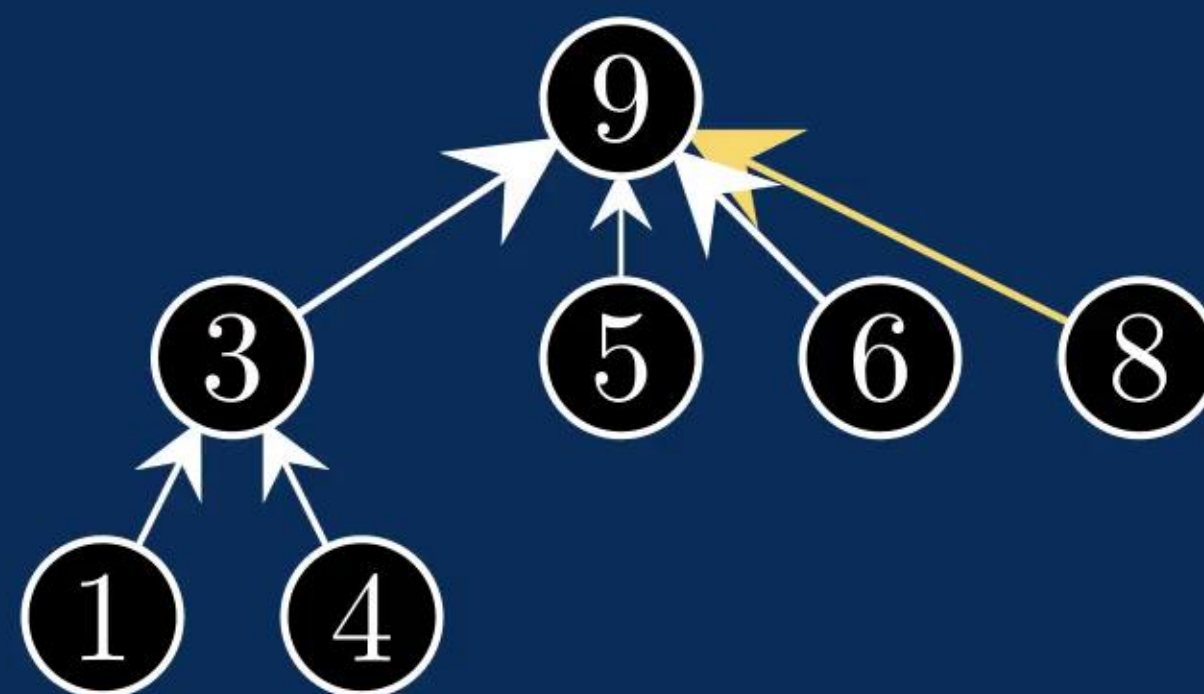
union(5,9)

union(8,6)

union(8,5)

union(8,1)

find(1)



Yellow = changed by compression

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)

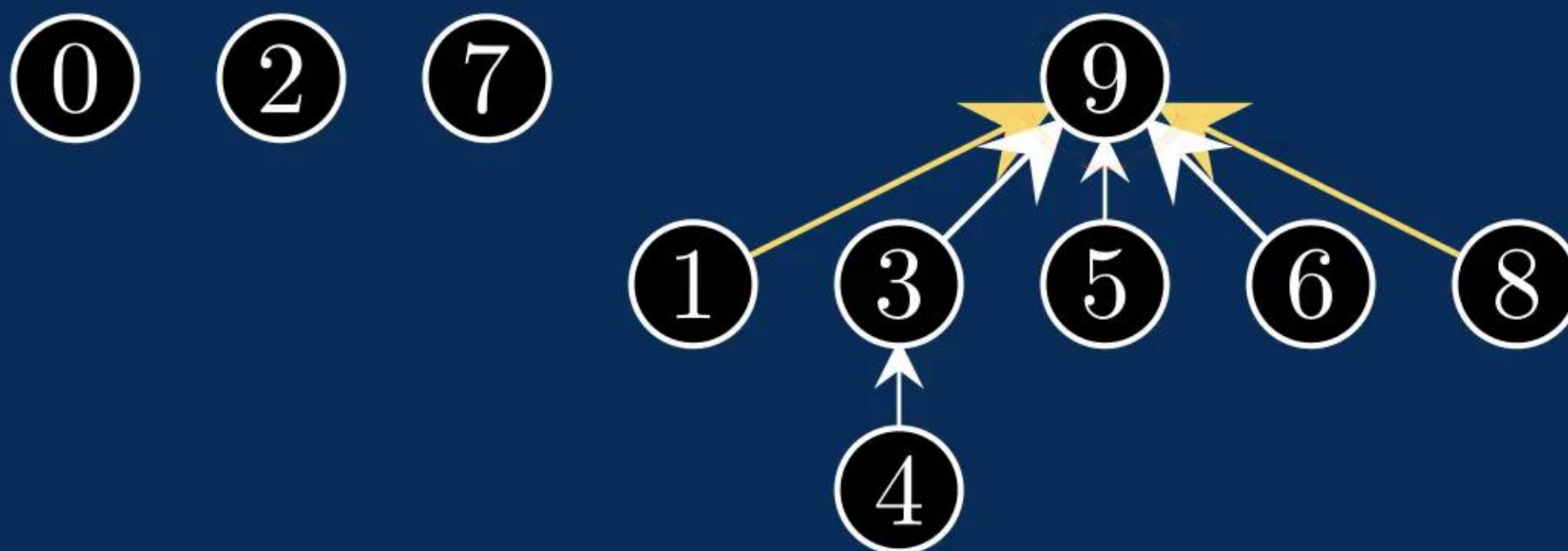
union(5,9)

union(8,6)

union(8,5)

union(8,1)

find(1)



Yellow = changed by compression

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)

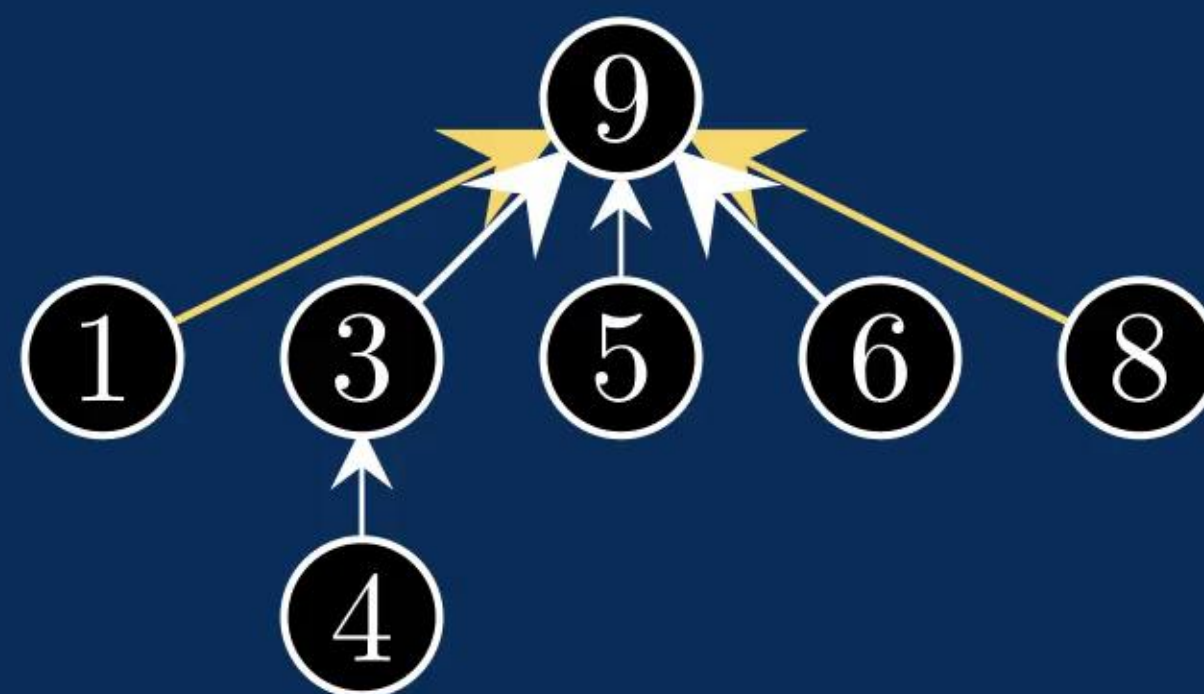
union(8,6)

union(8,5)

union(8,1)

find(1)

find(4)



Yellow = changed by compression

⁰	¹	²	³	⁴	⁵	⁶	⁷	⁸	⁹
-1	9	-1	9	9	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)

find(4)



Yellow = changed by compression

Strategy	Cost per operation

Strategy	Cost per operation
Naive union	$\mathcal{O}(n)$

Strategy	Cost per operation
Naive union	$\mathcal{O}(n)$
Union by size	$\mathcal{O}(\log n)$

Strategy	Cost per operation
Naive union	$\mathcal{O}(n)$
Union by size	$\mathcal{O}(\log n)$
Size + path compression	$\mathcal{O}(\alpha(n))$ amortized

Strategy	Cost per operation
Naive union	$\mathcal{O}(n)$
Union by size	$\mathcal{O}(\log n)$
Size + path compression	$\mathcal{O}(\alpha(n))$ amortized

α is the inverse Ackermann function: $\alpha(n) \leq 4$ for any n you will ever use.

Next time: we'll prove the slightly weaker $\mathcal{O}(\log^* n)$ bound.

(Tarjan's $\alpha(n)$ bound is beyond this course.)