# DSU

Disjoint Set Union

# Basic Problem

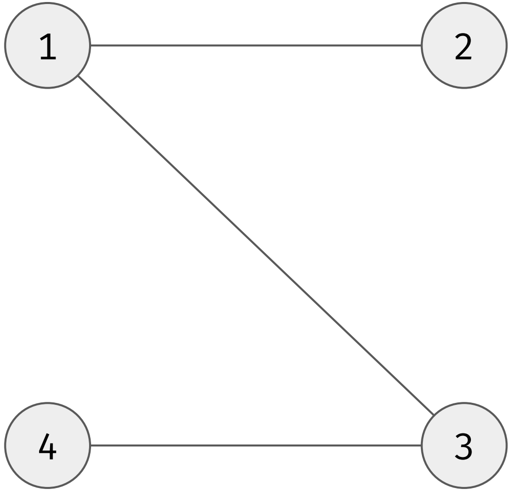
We have an undirected graph with N nodes and 0 edges. Process Q queries of the following types in order:

- Add an edge between u and v
- Check whether u and v are in the same connected component

$1 <= N, Q <= 2×10^5$

# Example Input/Output

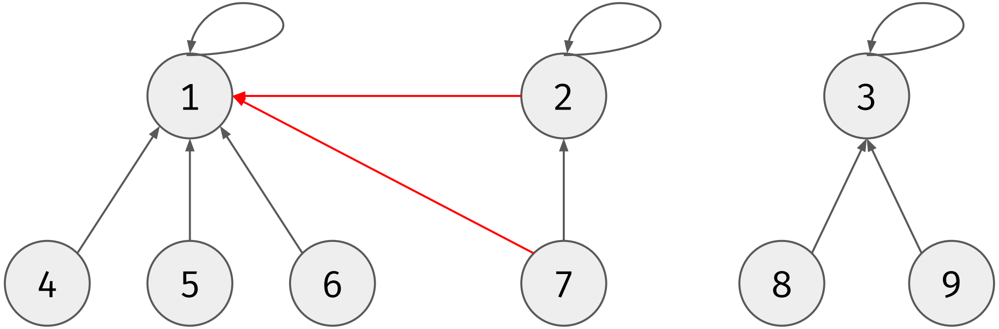
```
N = 4, Q = 7
CHECK 1 2      NO
UNION 1 2
UNION 3 4
CHECK 1 2      YES
CHECK 2 3      NO
UNION 1 3
CHECK 2 4      YES
```



# Observations

- We only care about connectivity - the edges in a connected component don’t actually matter
  - E.g. (1)--(2)--(3) is effectively the same as (1)--(3)--(2)
- If we assign a “representative” node to each connected component, then we can quickly identify them
  - We can point all nodes in a component to the representative
  - When we join components, point each node in the first to the representative of the second
  - Too slow if we do this naively

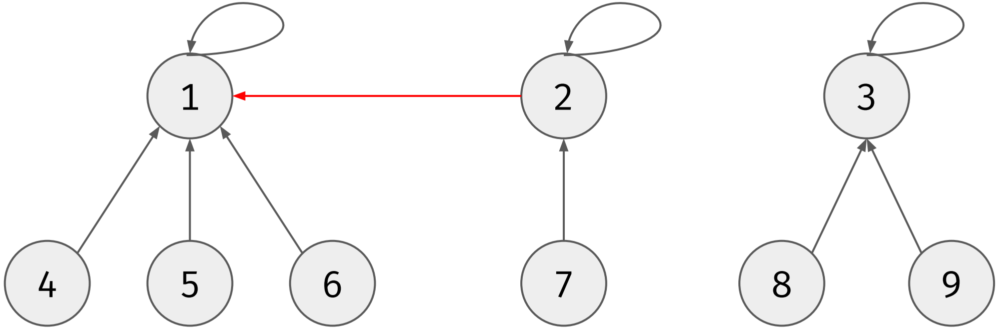
```
UNION 4 7:
```



# Optimization 1

- When we join components, we point **only the representative** of the first component to the representative of the second
- This way, we can just follow a path to get a representative (call this `FIND`)
- `UNION` is now O(`FIND`)
- Still too slow without further optimizations
  - What happens when we have "`UNION x x-1`" for each x from 2 to N?
  - "`FIND 1`" will take O(N) time

`UNION 4 7:`



# Optimization 2 (Union by Rank/Path Balancing)

- Point the representative of the **smaller component** to the bigger component
- `FIND` complexity is now O(log N)
- See Wikipedia for a proof

# Optimization 3 (Path Compression)

- When traversing the graph to find a representative, point each visited node to its parent’s parent
- This speeds up future `FIND` queries
- `FIND` complexity is now O(α(N)), where α is the Inverse-Ackermann function
  - α(N) grows very slowly and is effectively constant

(You don’t need union by rank if you use path compression)

# Code

```
int cmp[100001];

int find(int A) {
    while (A ≠ cmp[A]) cmp[A] = cmp[cmp[A]], A = cmp[A];
    return A;
}

void onion(int A, int B) { cmp[find(A)] = find(B); }

int main() {
    iota(cmp + 1, cmp + n + 1, 1);
}
```

I use onion because union is a reserved keyword in C++

std::iota fills cmp with 1, 2, ..., N because we want each node to point to itself

# We can store additional information too!

Size of the component, number of edges in the component, etc.

We can merge this information in `UNION`

# Example Problem - COCI 2020 Sjekira

You should remember this problem: https://oj.uz/problem/view/COCI20_sjekira

Solution sketch:

- It’s optimal to “isolate” the hardest node (i.e. chop all of its incident edges)
- Querying the maximum hardness in trees that can be cut is inconvenient, so we process the chopped edges backwards (i.e. join trees by adding edges)
- Use DSU to find the maximum hardness in the trees we join

# DSU Code for Sjekira

```
int find(int A) {
    while (cmp[A] != A) cmp[A] = cmp[cmp[A]], A = cmp[A];
    return A;
}

void onion(int A, int B) {
    A = find(A), B = find(B);
    if (A == B) return;
    ans += hardness[A] + hardness[B];
    hardness[B] = max(hardness[B], hardness[A]);
    cmp[A] = B;
}
```

Notice how we can store additional information about components

# Other Cool Things You Can Do With DSU

- Minimum spanning trees
  - A tree that connects all nodes and has the minimum sum of edge weights
  - E.g. COCI 2020 Odašiljači
- Checking whether a graph is bipartite
  - Basically checking whether there exists an odd cycle in the graph
- DSU with rollback
  - Undo `UNION` queries
  - You can’t use path compression, so you have to use union by rank
  - E.g. APIO 2019 Bridges
- DSU tree
  - Useful for finding all nodes reachable after a certain `UNION` query
  - E.g. IOI 2018 Werewolf

# Practice Problems (Roughly Ordered; No MST)

| | | |
|---|---|---|
| USACO 2018 | Mootube | http://www.usaco.org/index.php?page=viewproblem2&cpid=789 |
| SAPO 2019 | Jump | https://saco-evaluator.org.za/cms |
| Croatian OI 2015 | Kovanice | https://oj.uz/problem/view/COI15_kovanice |
| Baltic OI 2016 | Park | https://oj.uz/problem/view/BOI16_park |
| USACO 2020 | Favorite Colors | http://www.usaco.org/index.php?page=viewproblem2&cpid=1042 |
| APIO 2020 | Swapping Cities | https://oj.uz/problem/view/APIO20_swap |
| IOI 2018 | Werewolf | http://oj.uz/problem/view/IOI18_werewolf |
| USACO 2019 | Valleys | http://www.usaco.org/index.php?page=viewproblem2&cpid=950 |
| APIO 2019 | Bridges | https://oj.uz/problem/view/APIO19_bridges |
| SAPO 2017 | Stargazing | https://saco-evaluator.org.za/cms |
| JOISC 2017 | Port Facility | https://oj.uz/problem/view/JOI17_port_facility |