Competitive Programming Club

Meeting 3

Announcements

1. Fallen Star Tour
    a. Tomorrow 12:00-12:30 pm at Jacob's hall
    b. React with a thumbs up in #general in our Slack chat if you are coming!
2. CALICO Contest
    a. The California Informatics Competition (CALICO) is a contest hosted by UC Berkeley
    b. Competition this Saturday
    c. Registration ends tomorrow: https://calico.cs.berkeley.edu/
    d. If interested to compete with our club, react in #general channel

Union find
(Disjoint Set Union)

Motivation

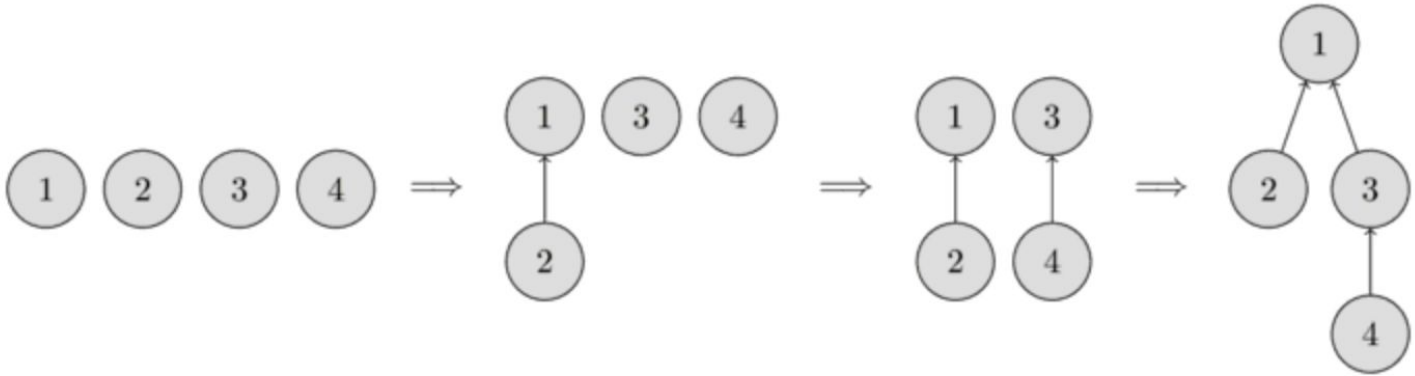
Union particularly useful for managing and solving problems related to network connectivity, such as determining whether two nodes are in the same connected component or performing components merging in a network.

Union Find Basics

Objective: Find if there is a path between two node

- **Initialization**: Sets up an array where each element points to itself, indicating that each element forms its own disjoint set.
- **Find**: Returns the root representative of the set containing the specified element.
- **Union**: Merges the sets containing two different elements by linking one representative root to the other.
- https://cp-algorithms.com/data_structures/disjoint_set_union.html

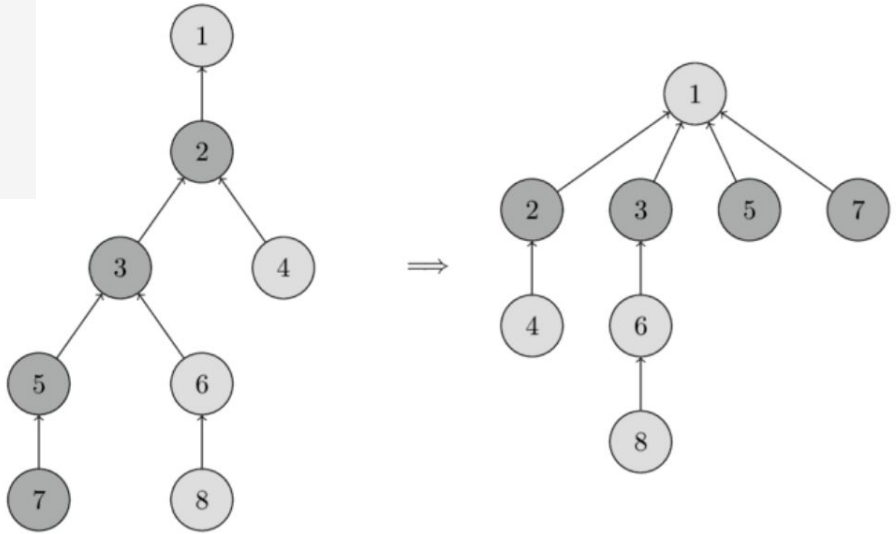
```
void make_set(int v) {
    parent[v] = v;
}

int find_set(int v) {
    if (v == parent[v])
        return v;
    return find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b)
        parent[b] = a;
}
```


1
2
3
4
1
3
4
2
1
3
2
4
1
2
3
4

Path compression optimization O(NlogN)

Path compression

```
int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}
```



Union find Optimization #2 O(NlogN)

Union by size / rank

- change the union_set operation
  - change which tree gets attached to the other one.
  - naive implementation the second tree always got attached to the first one. Potentially leading to chains of O(N). With this optimization we will avoid this by choosing very carefully which tree gets attached
- Most popular Heuristics
  - the first approach we use the size of the trees as rank
  - second one we use the depth of the tree (more precisely, the upper bound on the tree depth, because the depth will get smaller when applying path compression).

Here is the implementation of union by size:

```
void make_set(int v) {
    parent[v] = v;
    size[v] = 1;
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (size[a] < size[b])
            swap(a, b);
        parent[b] = a;
        size[a] += size[b];
    }
}
```

By depth

```
void make_set(int v) {
    parent[v] = v;
    rank[v] = 0;
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}
```

Use both optimization O(α(n))

- α(n) is the inverse Ackermann function, which grows very slowly. In fact it grows so slowly, that it doesn't exceed 4 for all reasonable n (approximately $n < 10^{600}$)

Example Problem

You are given Graph with n nodes, m edges, not necessarily connected

Now, you are given Q queries, with 2 types:

1. Delete an edge (x, y)
2. Answer the query if node q1 and node q2 are connected

Output the answer to the query

Q, n, m <= 10^5

Practice Problems

https://vjudge.net/contest/624456

Password: ucsd_icpc