A Optimized "disjoint set" with Path Compression and Union by Rank

Report Issue

Optimized "disjoint set" with Path Compression and Union by Rank

This implementation of the "disjoint set" is optimized with both "path compression" and "union by rank".

Implementation

C++ Java Python3 Copy Run Playground

```
class UnionFind {
public:
    UnionFind(int sz) : root(sz), rank(sz) {
        for (int i = 0; i < sz; i++) {
            root[i] = i;
            rank[i] = 1;
        }
    }

    int find(int x) {
        if (x == root[x]) {
            return x;
        }
        return root[x] = find(root[x]);
    }

    void unionSet(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX != rootY) {
            if (rank[rootX] > rank[rootY]) {
                root[rootY] = rootX;
            } else if (rank[rootX] < rank[rootY]) {
                root[rootX] = rootY;
            } else {
                root[rootY] = rootX;
```

Time Complexity

	Union-find Constructor	Find	Union	Connected
Time Complexity	$O(N)$	$O(\alpha(N))$	$O(\alpha(N))$	$O(\alpha(N))$

Note: N is the number of vertices in the graph. α refers to the Inverse Ackermann function. In practice, we assume it's a constant. In other words, $O(\alpha(N))$ is regarded as $O(1)$ on average.

- For the `union-find` constructor, we need to create two arrays of size N each.
- When using the combination of union by rank and the path compression optimization, the `find` operation will take $O(\alpha(N))$ time on average. Since `union` and `connected` both make calls to `find` and all other operations require constant time, `union` and `connected` functions will also take $O(\alpha(N))$ time on average.

Space Complexity

We need $O(N)$ space to store the array of size N.