Summary of the "disjoint set" data structure

Report Issue

The main idea of a "disjoint set" is to have all connected vertices have the same parent node or root node, whether directly or indirectly connected. To check if two vertices are connected, we only need to check if they have the same root node.

The two most important functions for the "disjoint set" data structure are the `find` function and the `union` function. The `find` function locates the root node of a given vertex. The `union` function connects two previously unconnected vertices by giving them the same root node. There is another important function named `connected`, which checks the "connectivity" of two vertices. The `find` and `union` functions are essential for any question that uses the "disjoint set" data structure.

Implementation of the "disjoint set"

C++ Java Python3 Copy

```cpp
class UnionFind {
public:
    // Constructor of Union-find. The size is the length of the root array.
    UnionFind(int sz) : root(sz);
    int find(int x);
    void unionSet(int x, int y);
    bool connected(int x, int y);
};
```

find function of the "disjoint set"

The "disjoint set" mainly uses the `find` function to find the root node of a given vertex.

- A basic implementation of the `find` function:

C++ Java Python3 Copy

```cpp
int find(int x) {
    while (x != root[x]) {
        x = root[x];
    }
    return x;
}
```

- The `find` function – optimized with path compression:

C++ Java Python3 Copy

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

union function of the "disjoint set"

The "disjoint set" mainly uses the `union` function to connect two vertices, `x`, and `y`, by equating their root node.

- A basic implementation of the `union` function:

C++ Java Python3 Copy

```cpp
void unionSet(int x, int y) {
    int rootX = find(x);
    int rootY = find(y);
```

```
    if (rootX != rootY) {
        root[rootY] = rootX;
    }
}
```

- The `union` function – Optimized by union by rank:

C++ Java Python3 Copy

```
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;
            rank[rootX] += 1;
        }
    }
}
```

connected function of the “disjoint set”

The connected function checks if two vertices, `x` and `y`, are connected by checking if they have the same root node. If `x` and `y` have the same root node, they are connected. Otherwise, they are not connected.

C++ Java Python3 Copy

```
bool connected(int x, int y) {
    return find(x) == find(y);
}
```

Tips for using the “disjoint sets” data structure in solving LeetCode problems

The code for the disjoint set is highly modularized. You might want to become familiar with the implementation. I would highly recommend that you understand and memorize the implementation of “disjoint set with path compression and union by rank”.

Finally, we strongly encourage you to solve the exercise problems using the abovementioned implementation of the “disjoint set” data structure. Some of these problems can be solved using other data structures and algorithms, but we highly recommend that you practice solving them using the “disjoint set” data structure.