Back to Chapter
Previous Next

Path Compression Optimization - Disjoint Set

Report Issue

Path Compression Optimization - Disjoint Sets

In the previous implementation of the “disjoint set”, notice that to find the root node, we need to traverse the parent nodes sequentially until we reach the root node. If we search the root node of the same element again, we repeat the same operations. Is there any way to optimize this process?

The answer is yes! After finding the root node, we can update the parent node of all traversed elements to their root node. When we search for the root node of the same element again, we only need to traverse two elements to find its root node, which is highly efficient. So, how could we efficiently update the parent nodes of all traversed elements to the root node? The answer is to use “recursion”. This optimization is called “path compression”, which optimizes the `find` function.

Video Explanation

In this video, we'll talk about how the “path compression” optimization is implemented.

▶ Clarifying Notes

Algorithm

Here is the sample implementation of Path Compression.

C++ Java Python3 Copy Run Playground

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

    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) {
            root[rootY] = rootX;
        }
    }

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

Time Complexity

Time complexities shown below are for the average case, since the worst-case scenario is rare in practice.

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

Note: N is the number of vertices in the graph.

- As before, we need $O(N)$ time to create and fill the `root` array.
- For the `find`, `union`, and `connected` operations (the latter two operations both depend on the `find` operation), we need $O(1)$ time for the best case (when the parent node for some vertex is the root node itself). In the worst case, it would be $O(N)$ time when the tree is skewed. However, on average, the time complexity will be $O(\log N)$. Supporting details for the average time complexity can be found in Top-Down Analysis of Path Compression where R. Seidel and M. Sharir discuss the upper bound running time when path compression is used with arbitrary linking.

Space Complexity

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