

Union-Find Disjoint Set

Khritish Kumar Behera

September 11, 2024

Introduction

Union-Find is a powerful data structure for representing disjoint sets efficiently. It's commonly used in graph algorithms like Kruskal's algorithm and finding connected components.

Core Operation

- **make_set(x):** Creates a new set containing only the element x.
- **find(x):** Returns the representative (or root) of the set containing x.
- **union(x, y):** Merges the sets containing x and y.

Implementation

- **Disjoint-set forests:** Each set is represented as a rooted tree. The root of a tree is the representative of the set.
- **Parent pointers:** Each element stores a pointer to its parent. The root's parent is itself.
- **Rank:** The rank of a node is an upper bound on the height of the subtree rooted at that node.

Optimizations

- **Union by Rank:** When merging two sets, the root with the smaller rank becomes the child of the root with the larger rank. This helps maintain balanced trees.
- **Path Compression:** During find, update the parent pointers of nodes along the path to the root to the root itself. This improves future find operations.

Algorithm

Combined Parent and Rank Array

A one-dimensional array parent where:

- If $\text{parent}[i] > 0$, then $\text{parent}[i]$ stores the parent of element i.
- If $\text{parent}[i] < 0$, then $\text{parent}[i]$ stores the rank of the set rooted at i.

Initialization

1. For each element i, set $\text{parent}[i] = i$ (initially, each element is its own parent).

Find(x)

1. **Base Case:** If $\text{parent}[x] < 0$, then x is the root of its set, so return x.
2. **Recursive Call:** Otherwise, recursively call $\text{Find}(\text{parent}[x])$ to find the root of the set containing x.
3. **Path Compression:** While returning from the recursive call, update $\text{parent}[x]$ to be the root directly.

Union(x, y)

1. **Find Roots:** Call $\text{Find}(x)$ and $\text{Find}(y)$ to find the roots root_x and root_y of the sets containing x and y, respectively.
2. **Check for Same Set:** If $\text{root}_x == \text{root}_y$, the elements are already in the same set.
3. **Merge Sets:**
 - If $\text{parent}[\text{root}_x] < \text{parent}[\text{root}_y]$:
 - Make root_x a child of root_y by setting $\text{parent}[\text{root}_x] = \text{root}_y$
 - Otherwise:
 - Make root_y a child of root_x by setting $\text{parent}[\text{root}_y] = \text{root}_x$
 - If $\text{parent}[\text{root}_x] == \text{parent}[\text{root}_y]$, increment the rank of the new root (either root_x or root_y) by 1.

Explanation

- **Negative Values:** A negative value in $\text{parent}[i]$ indicates that i is the root of a set, and the absolute value of $\text{parent}[i]$ represents the rank of the set.
- **Merging Sets:** When merging two sets, the set with the smaller rank becomes a child of the set with the larger rank. This helps maintain a balanced tree structure.
- **Path Compression:** The optional path compression optimization can significantly improve the performance of subsequent find operations by flattening the tree structure.

Pseudocode

```
MAKE_SET(n):
    parent array of size n
    for i in (0, n - 1):
        parent[i] = -1

FIND(x):
    if parent[x] < 0:
        return x
    parent[x] = FIND(parent[x])
    return parent[x]

UNION(x, y):
    root_x = parent[x]
    root_y = parent[y]

    if root_x != root_y:
        if parent[root_x] < parent[root_y]:
            # rank(root_x) > rank(root_y)
            parent[root_x] += parent[root_y]
            parent[root_y] = root_x
        else:
            parent[root_y] += parent[root_x]
            parent[root_x] = root_y
```

Time Complexity

Amortized time complexity: Both find and union have amortized time complexity of $O(\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function, which grows very slowly.

Applications

- Kruskal's algorithm: Finding the minimum spanning tree of a graph.
- Finding connected components: Identifying groups of connected vertices in a graph.
- Cycle detection: Detecting cycles in graphs.
- Maze solving: Finding a path through a maze.
- Implementing various graph algorithms: Disjoint-set forests are a fundamental building block for many graph algorithms.