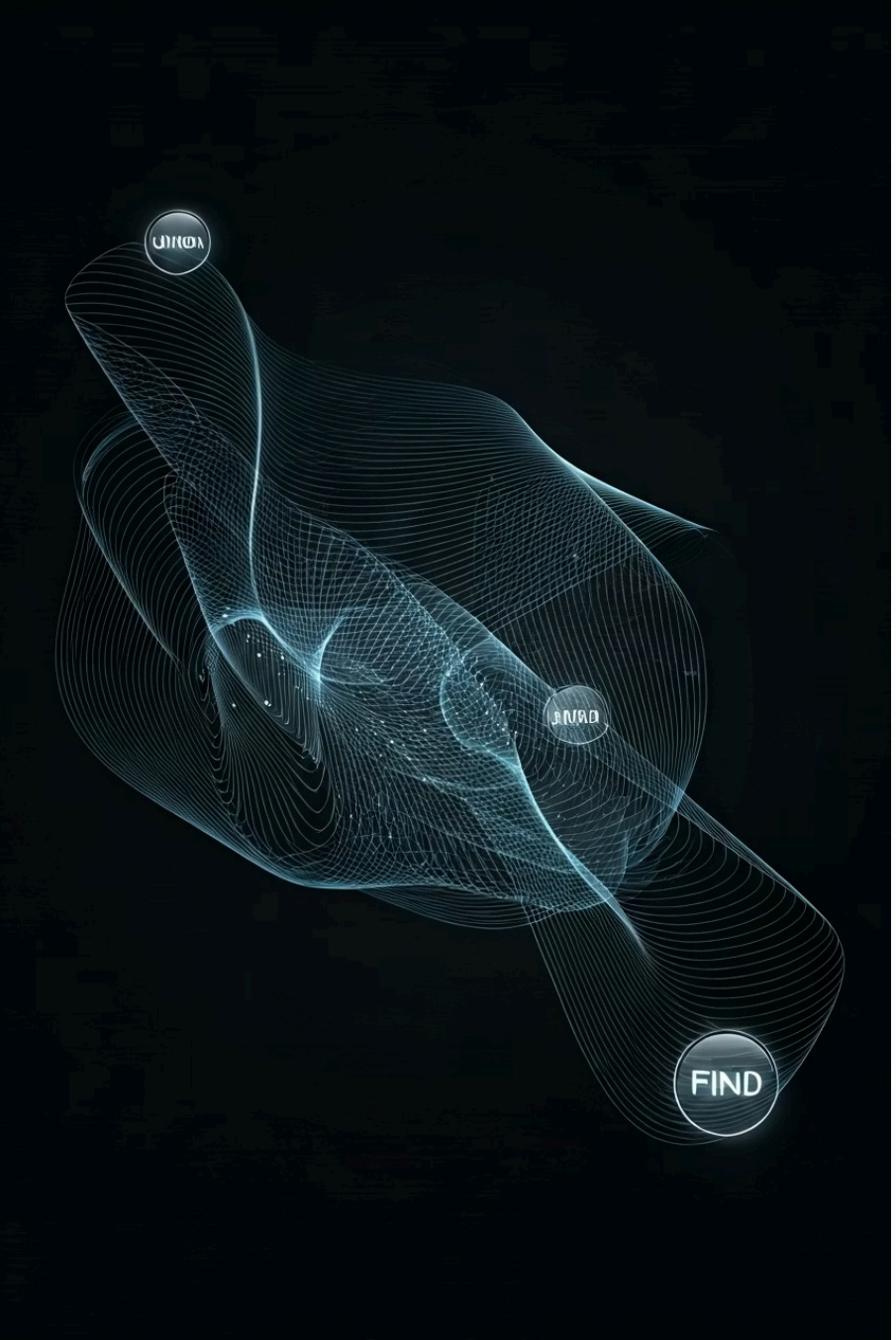
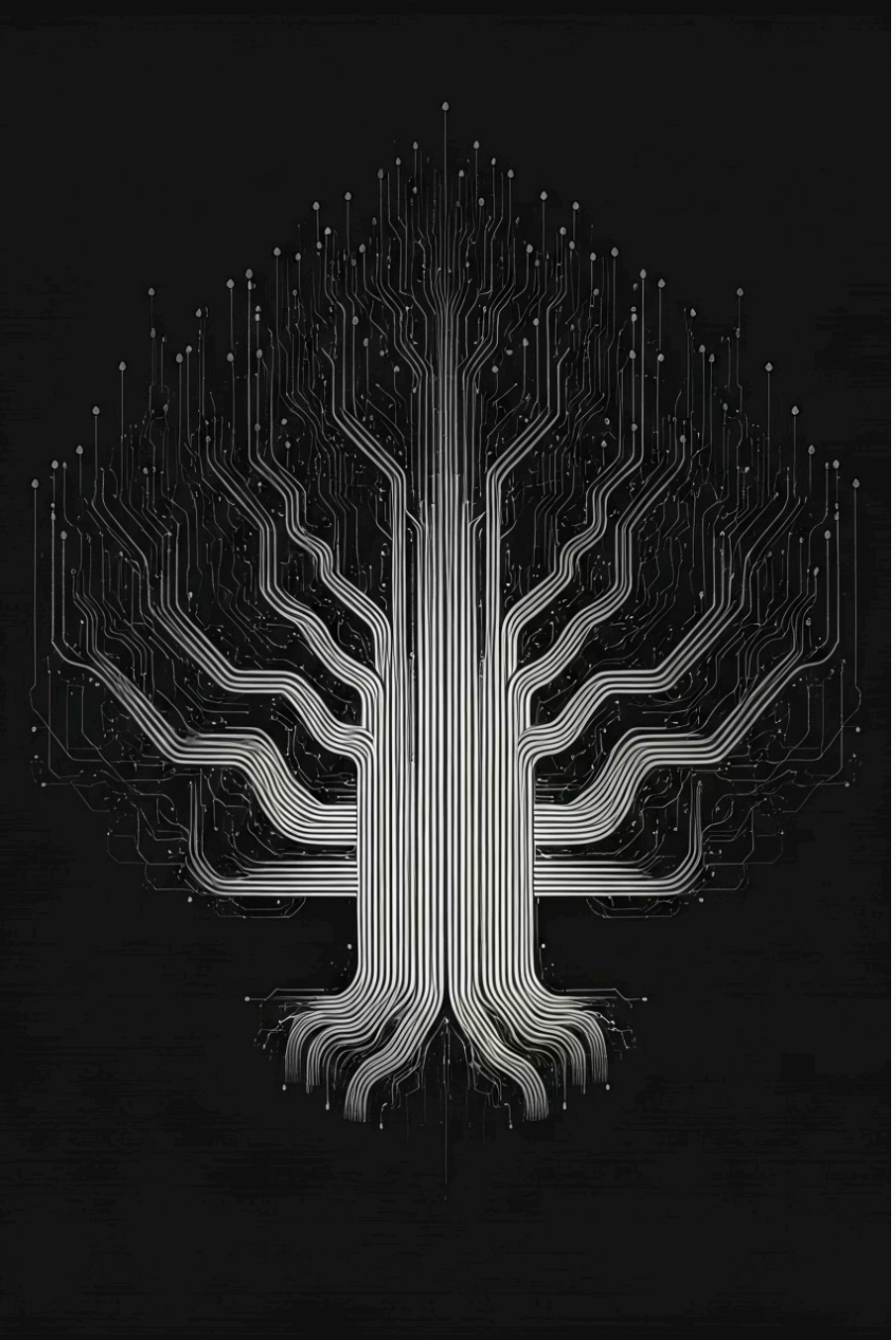


Optimizing Union and Find Operations

This presentation explores key optimizations for the Union-Find data structure. We will cover union by size, union by rank, and path compression, all crucial for efficient Disjoint Set algorithms.



What Is the Union-Find Data Structure?

Connectivity

Maintains disjoint sets and supports 'find' and 'union' operations.

Real-World Uses

Used in connectivity problems, clustering, and Kruskal’s MST algorithm.

Tree Structure

Elements are identified by parent pointers, forming a forest of trees.

Basic (Naive) Union and Find

Find Operation

The 'find' operation walks up parent pointers to reach the root of the tree.

Union Operation

The 'union' operation attaches one tree randomly under another's root.

Performance Issue

Trees can grow very tall in the worst-case scenarios. This results in an O(N) time complexity per operation.

The Problem with Tall Trees

Slow Find Operations

Excessively tall trees significantly slow down 'find' operations.

Degeneration

Repeated unions can cause the tree structure to degenerate into linked lists.

Optimization Need

Effective optimizations are essential to keep the tree structures flat and efficient.

Optimizing Union: Union by Size

Attach Smaller to Larger

Always attach the root of the smaller tree to the root of the larger tree.

Track Sizes

Maintain the size of each rooted tree using a dedicated 'size' array.

Minimize Height

This strategy ensures the maximum tree height remains as minimal as possible.

Alternative: Union by Rank

Union by rank is another effective optimization strategy, similar to union by size but focusing on tree depth or height.

Compare Ranks

Attach the tree with the smaller rank under the root of the tree with the higher rank.

Estimate Height

The 'rank' estimates the height or depth of the tree structure.

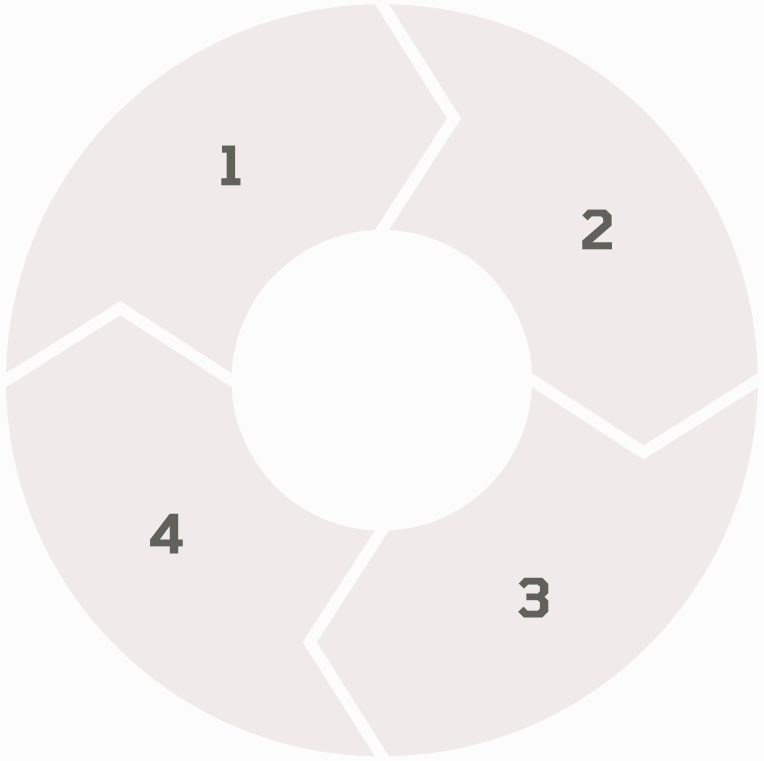
Equal Ranks

If ranks are equal, choose an arbitrary root and increment its rank by one.

Visual Example: Union by Size vs Rank

Initial State

Consider elements 2, 4, and 5 as individual sets.



Union(2, 4)

Illustrates how 2 and 4 are united, showing the height change.

Benefits

Observe how tree heights are maintained lower, leading to faster future 'finds'.

Union(4, 5)

Further demonstrates the union, contrasting size/rank with naive union.

Optimizing Find: Path Compression

Path compression is a powerful technique applied during the 'find' operation to significantly flatten the tree.

1

Traversal

During find(x), traverse from x to the root.

2

Re-parenting

Make every node on the path point directly to the root.

3

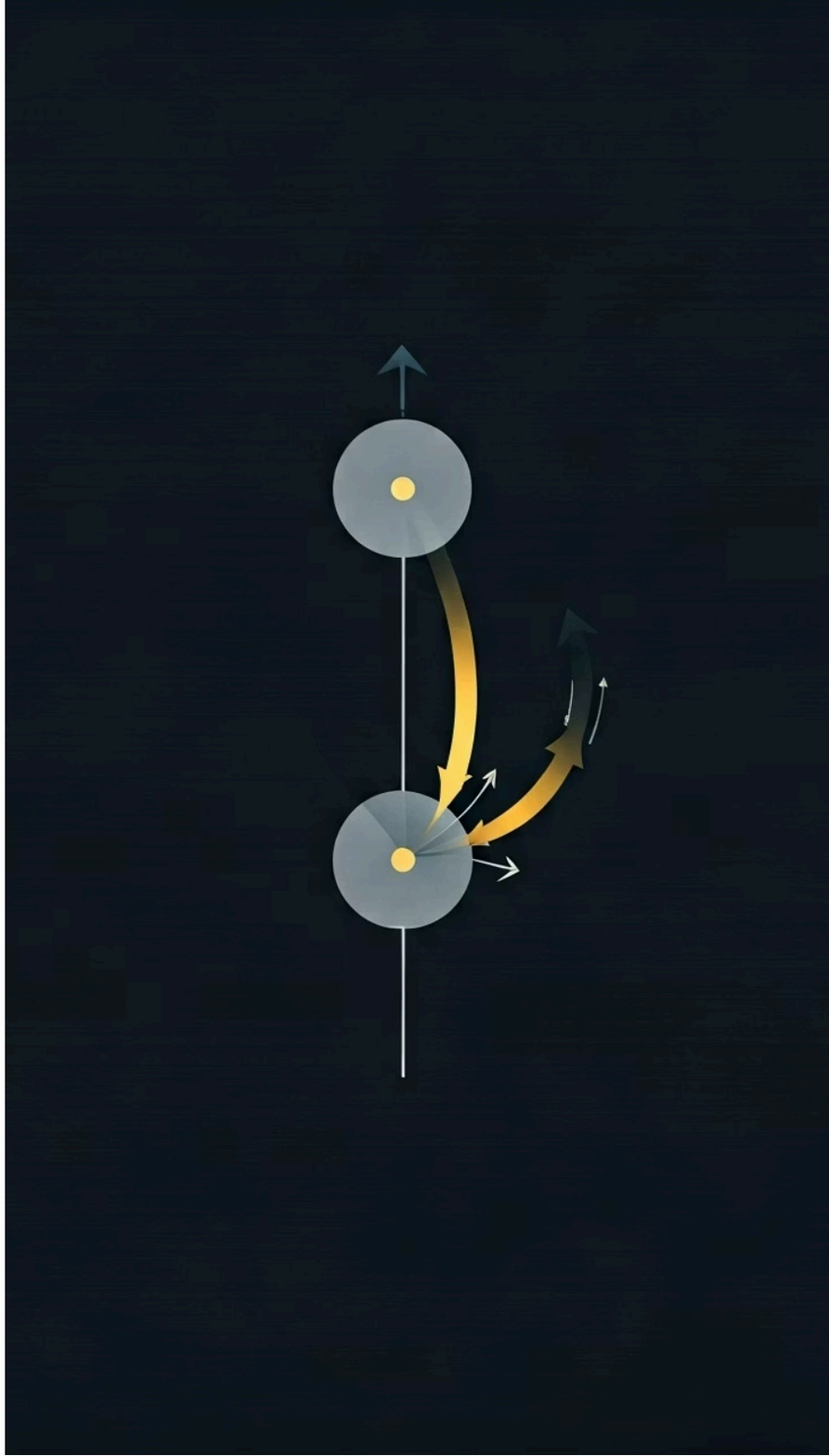
Flattening

This flattens the tree, drastically reducing future lookup times.

4

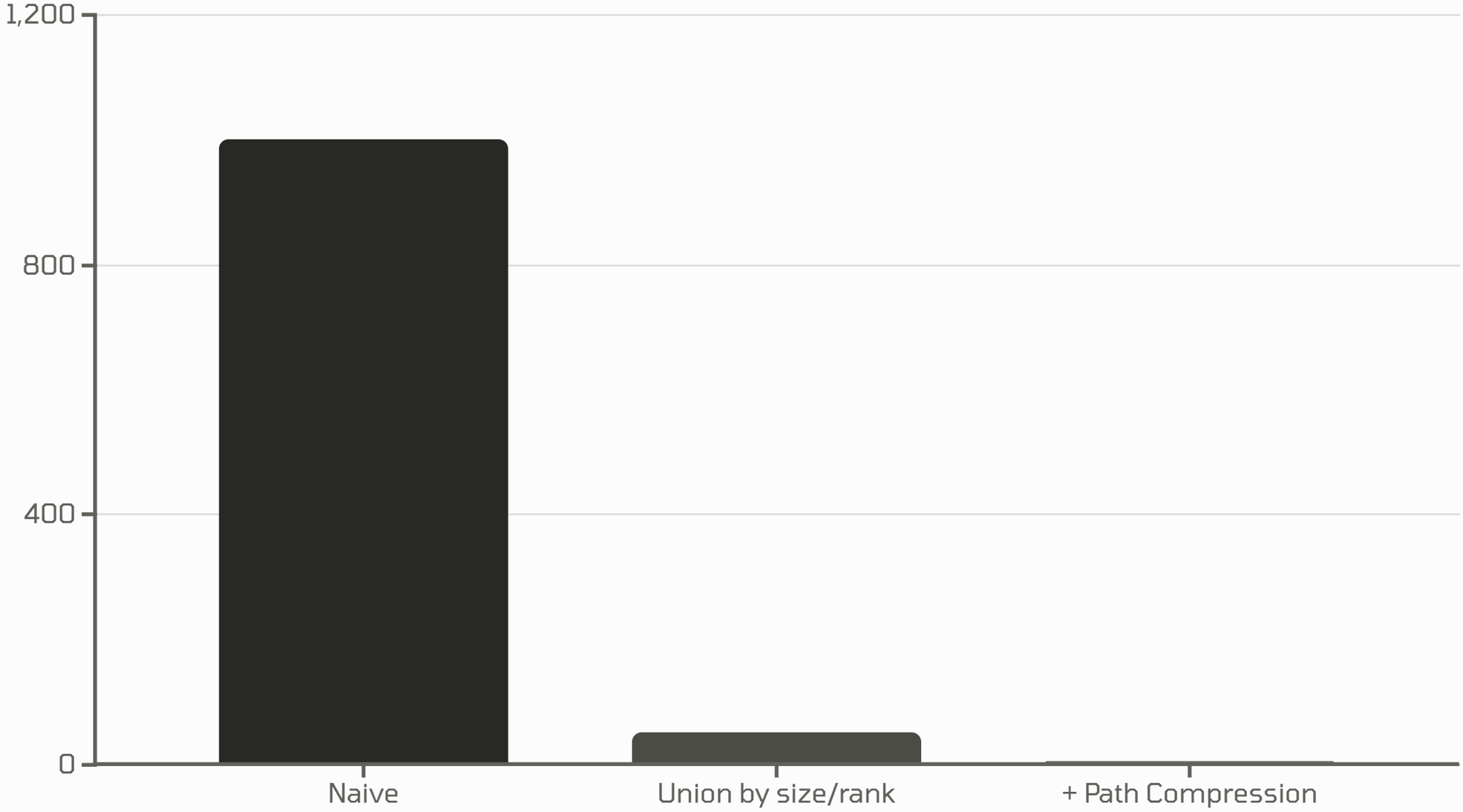
Implementation

Can be implemented using recursive or iterative re-parenting.



Time Complexity With Optimizations

The combination of union by size/rank and path compression yields impressive performance.



The amortized complexity becomes nearly O(1) per operation. Specifically, it's O(α(N)), where α is the inverse Ackermann function. Practically, α(N) is less than 5 for any feasible N on modern computers.

Applications and Summary



- Applications: Used in Kruskal's MST, dynamic connectivity, and network clustering.
- Efficiency: Optimizations make Union-Find practical for huge datasets.
- Key Takeaway: Union by size/rank and path compression are essential for high-speed Union-Find operations.