Disjoint Set Union (Union-Find) in DSA: Beginner to Advanced Guide

Subtitle: Learn how to efficiently manage disjoint sets for connectivity, cycle detection, and MST problems.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Disjoint Set Union (DSU), also called Union-Find, is a data structure used to **keep track of elements divided into disjoint sets**. It's essential for **connectivity checks, detecting cycles in graphs, and Kruskal's algorithm for MSTs**.

Step 1: Core Concepts

- **MakeSet(x):** Initialize element x as its own set
- **Find(x):** Return the representative (root) of the set containing x
- **Union(x, y):** Merge sets containing x and y
- **Optimizations:**
  - **Path Compression:** Flatten the tree during Find to speed up future queries
  - **Union by Rank / Size:** Always attach smaller tree under larger tree

Why it matters: DSU operations are almost **O(1) amortized**, making it ideal for large graphs.

Step 2: Implementation in JavaScript

1. Simple DSU

```
class DSU {
  constructor(n){
    this.parent = Array(n).fill(0).map((_,i)=>i);
    this.rank = Array(n).fill(0);
  }

  find(x){
    if(this.parent[x] !== x) this.parent[x] = this.find(this.parent[x]); // path compression
    return this.parent[x];
```

```
  }

  union(x, y){
    let rootX = this.find(x);
    let rootY = this.find(y);
    if(rootX === rootY) return false; // already connected
    // union by rank
    if(this.rank[rootX] < this.rank[rootY]) this.parent[rootX] = rootY;
    else if(this.rank[rootX] > this.rank[rootY]) this.parent[rootY] = rootX;
    else { this.parent[rootY] = rootX; this.rank[rootX]++; }
    return true;
  }
}
```

Step 3: Example Usage

1. Cycle Detection in Undirected Graph

```
let edges = [[0,1],[1,2],[2,0]];
let dsu = new DSU(3);
let hasCycle = false;

for(let [u,v] of edges){
  if(!dsu.union(u,v)) {
    hasCycle = true;
    break;
  }
}
console.log("Cycle exists:", hasCycle);
```

2. Kruskal’s Algorithm for MST

- Sort edges by weight
- Use DSU to check if adding an edge forms a cycle
- Add edge if no cycle → builds MST

Step 4: Practical Exercises

1. Count connected components in an undirected graph
2. Implement Kruskal's algorithm for given weighted graph
3. Merge sets and find representative for multiple queries
4. Check connectivity after a sequence of union operations

Step 5: Common Mistakes

- Forgetting path compression → slower performance
- Not handling union by rank/size → taller trees → inefficiency
- Confusing root with parent during operations
- Forgetting to initialize sets for all elements

Step 6: Practice Plan for DSU

Day 1:

- Implement DSU, basic union/find → 3–5 problems

Day 2:

- Cycle detection, connected components → 5 problems

Day 3:

- Kruskal's MST, edge-based queries → 5 problems

Day 4:

- Mixed DSU problems, optimizations → 5–7 problems

Key Takeaways

- DSU efficiently handles **disjoint sets and connectivity queries**
- Optimizations like **path compression** and **union by rank** are crucial
- Essential for **graph algorithms, MST, and network problems**
- Practice helps in **internalizing union/find logic and optimizations**

Visit haas.dev for more DSU guides, problem sets, and interview preparation resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app