Data Structures and Algorithms

Lecture 29

Aniket Basu Roy

2026-04-08 Wed

Contents

1 Agenda **2**
1.1 Time Complexity of Kruskal's Algorithm 2

2 Cycle Detection in Kruskal's **2**

3 Disjoint-Set Union (Union-Find) ADT **2**
3.1 Operations 2
3.2 Kruskal's Revisited 2

4 Forest Implementation **3**
4.1 Basic (unoptimized) 3

5 Optimization 1: Union by Rank **3**
5.1 Lemma 4
5.2 Corollary 4

6 Optimization 2: Path Compression **4**

7 Combined: Union by Rank + Path Compression **5**
7.1 Theorem (Tarjan 1975) 5
7.2 The Inverse Ackermann Function 5

8 Time Complexity of Kruskal's Algorithm **5**

9 Comparison: Kruskal vs. Prim-Jarník **5**

10 Questions **6**

1 Agenda

1.1 Time Complexity of Kruskal's Algorithm

- Cycle detection and connected components
- Disjoint-Set Union (Union-Find) ADT
- Forest implementation
- Optimization 1: Union by Rank
- Optimization 2: Path Compression
- Time complexity of Kruskal's

2 Cycle Detection in Kruskal's

Kruskal's must decide: does adding (u, v) form a cycle?

Key observation: (u, v) forms a cycle in the current forest A iff u and v are in the **same connected component** of A.

We need a data structure that maintains connected components under edge insertions.

3 Disjoint-Set Union (Union-Find) ADT

Maintain a collection of disjoint sets, each with a **representative** element.

3.1 Operations

- MAKE-SET(x): create a new singleton set $\{x\}$.
- FIND-SET(x): return the representative of the set containing x.
- UNION(x, y): merge the sets containing x and y into one.

3.2 Kruskal's Revisited

```
KRUSKAL(G, w)
1.  A = {}
2.  for each vertex v in G.V
3.      MAKE-SET(v)
4.  sort edges in E by weight (non-decreasing)
```

```
5.  for each edge (u, v) in E  (in sorted order)
6.      if FIND-SET(u) != FIND-SET(v)
7.          A.insert((u, v))
8.          UNION(u, v)
9.  return A
```

Total Union-Find calls: n MAKE-SET, $2m$ FIND-SET, $\leq n-1$ UNION.

4 Forest Implementation

Represent each set as a **rooted tree**:

- x.parent: parent of x (root satisfies x.parent $= x$).
- FIND-SET(x): follow parent pointers to the root.
- UNION(x, y): attach the root of one tree under the root of the other.

4.1 Basic (unoptimized)

```
MAKE-SET(x)
1.  x.parent = x

FIND-SET(x)
1.  if x != x.parent
2.      return FIND-SET(x.parent)
3.  return x

UNION(x, y)
1.  FIND-SET(x).parent = FIND-SET(y)
```

Worst case: repeated UNION creates a chain of length n. FIND-SET costs $O(n)$; a sequence of m operations costs $O(m \cdot n)$.

5 Optimization 1: Union by Rank

Always attach the **shorter** tree under the **taller** one.

x.rank: upper bound on the height of the subtree rooted at x.

```
MAKE-SET(x)
1.  x.parent = x
```

```
2.  x.rank   = 0

LINK(x, y)          // x and y are roots
1.  if x.rank > y.rank
2.      y.parent = x
3.  else
4.      x.parent = y
5.      if x.rank == y.rank
6.          y.rank = y.rank + 1

UNION(x, y)
1.  LINK(FIND-SET(x), FIND-SET(y))
```

5.1 Lemma

With union by rank, the tree rooted at x has at least $2^{x.\text{rank}}$ nodes.

5.2 Corollary

Maximum rank is $\lfloor \log n \rfloor$, so FIND-SET costs $O(\log n)$. Total for m operations: $O(m \log n)$.

6 Optimization 2: Path Compression

During FIND-SET, make every node on the root path point **directly** to the root.

```
FIND-SET(x)
1.  if x != x.parent
2.      x.parent = FIND-SET(x.parent)   // path compression
3.  return x.parent
```

```
Before FIND-SET(d):          After FIND-SET(d):
   a                            a
   |                           / | \
   b                          b  c  d
   |
   c
   |
   d
```

Path compression alone (without union by rank) gives $O(m \log n)$ amortized.

7 Combined: Union by Rank + Path Compression

7.1 Theorem (Tarjan 1975)

A sequence of m MAKE-SET, FIND-SET, UNION operations on n elements costs

$$O(m \cdot \alpha(n))$$

total time, where α is the **inverse Ackermann function**.

7.2 The Inverse Ackermann Function

The Ackermann function $A(k, k)$ grows extremely fast (faster than any tower of exponentials). $\alpha(n) = \min\{k \mid A(k, k) \geq n\}$ grows correspondingly slowly:

$$\alpha(n) \leq 4 \quad \text{for all } n \leq A(4,4) = 2^{2^{2^{2^{16}}}}.$$

For all practical purposes: $\alpha(n)$ behaves like a constant.

8 Time Complexity of Kruskal's Algorithm

Step	Cost
Initialization (n MAKE-SET)	$O(n\,\alpha(n))$
Sorting m edges	$O(m \log m)$
$2m$ FIND-SET + $n-1$ UNION	$O(m\,\alpha(n))$
Total	$O(m \log m)$

Since $m \leq n^2$: $\log m \leq 2 \log n$, so $O(m \log m) = O(m \log n)$.

$$\boxed{T_{\text{Kruskal}} = O(m \log n)}$$

The bottleneck is **sorting**, not the Union-Find operations.

9 Comparison: Kruskal vs. Prim-Jarník

Algorithm	Data Structure	Time Complexity
Kruskal	Union-Find (forest)	$O(E \log V)$
Prim-Jarník	Array	$O(V^2)$
Prim-Jarník	Binary min-heap	$O(E \log V)$

- For **sparse** graphs ($E = O(V)$): all $O(E \log V)$ variants are equivalent.
- For **dense** graphs ($E = \Theta(V^2)$): Prim-Jarník with array gives $O(V^2)$, beating Kruskal's $O(V^2 \log V)$.

10 Questions

- Prove: with union by rank, a tree containing k nodes has rank $\leq \lfloor \log k \rfloor$.
- Why does path compression not change the rank values?
- If edges are already sorted (e.g., integer weights in $[1..W]$ with $W = O(V)$), what is the running time of Kruskal's?