CSS.201.1 Algorithms

Instructor: Umang Bhaskar

TIFR 2024, Aug-Dec

Scribe: Soham Chatterjee

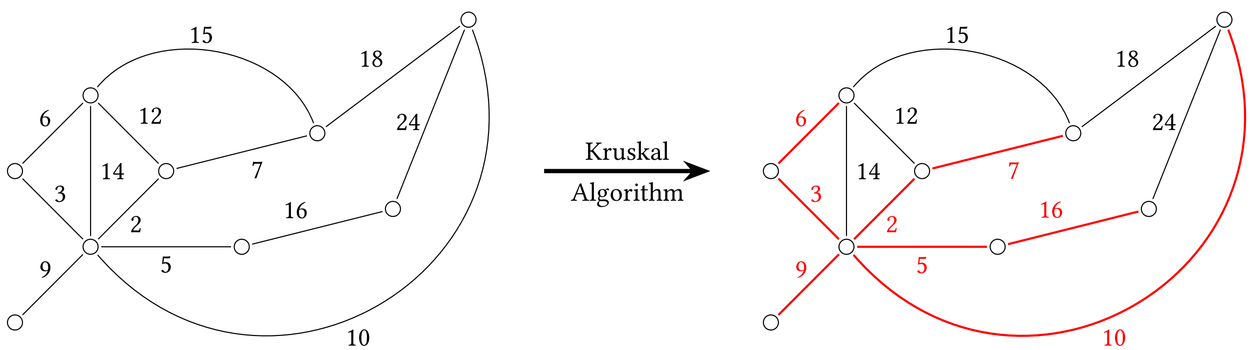
sohamchatterjee999@gmail.com
Website: sohamch08.github.io

Contents

Chapter 1

Kruskal Algorithm with Data Structure

1.1 Kruskal Algorithm



1.2 Data Structure 1: Array

1.3 Data Structure 2: Left Child Right Siblings Tree

1.4 Data Structure 3: Union Find

1.4.1 Analyzing the Union-Find Data-Structure

We call a node in the union-find data-structure a *leader* if it is the root of the (reversed) tree.

> **Lemma 1.4.1**
> Once a node stop being a leader (i.e. the node in top of a tree). it can never become a leader again.

Proof: A node x stops being a leader only because of the Union operation which made x child of a node y which is a leader of a tree. From this point on, the only operation that might change the parent pointer of x is the Find operation which traverses through x. Since path-compression only change the parent pointer of x to point to some other node y. Therefore the parent pointer of x will never become equal to itself i.e. x can never be a leader again. Hence once x stops being a leader it can never be a leader again. ■

Lemma 1.4.2

Once a node stop being a leader then its rank is fixed.

Proof: The rank of a node changes only by an Union operation. But the Union operation only changes the rank of nodes that are leader after the operation is done. Therefore once a node stops being a leader it's rank will not being changed by an Union operation. Hence once a node stop being a leader then its rank is fixed. ■

Lemma 1.4.3

Ranks are monotonically increasing in the reversed trees, as we travel from a node to the root of the tree.

Proof: To show that the ranks are monotonically increasing it suffices to prove that for all edge $u \to v$ in the data structure we have $\text{rank}(u) < \text{rank}(v)$. ■

Lemma 1.4.4

When a node gets rank k than there are at least $\geq 2^k$ elements in its subtree.

Corollary 1.4.5

For all vertices v, $v.rank \leq \lfloor \log n \rfloor$

Corollary 1.4.6

Height of any tree $\leq \lfloor \log_2 n \rfloor$

Lemma 1.4.7

The number of nodes that get assigned rank k throughout the execution of the Union-Find data-structure is at most $\frac{n}{2^k}$.

Define $N(r) = \#\text{vertices with rank at least } k$. Then by the above lemma we have $N(r) \leq \frac{n}{2^k}$.

Lemma 1.4.8

The time to perform a single find operation when we perform union by rank and path compression is $O(\log n)$ time.

We will show that we can do much better. In fact we will show that for m operations over n elements the overall running time is $O((n+m)\log^* n)$

Lemma 1.4.9

During a single Find(x) operation, the number of jumps between blocks along the search path is $O(\log^* n)$.

Lemma 1.4.10

At most $|Block(i)| \leq Tower(i)$ many Find operations can pass through an element x which is in the i^{th} block (i.e. $\text{index}_B(x) = i$) before $x.parent$ is no longer in the i^{th} block. That is $\text{index}_B(x.parent) > i$.

Lemma 1.4.11

There are at most $\frac{n}{Tower(i)}$ nodes that have ranks in the i^{th} block throughout the algorithm execution.

Lemma 1.4.12

The number of internal jumps performed, inside the i^{th} block, during the lifetime of UNION-FIND data structure is $O(n)$.

Theorem 1.4.13

The number of internal jumps performed by the UNION-FIND data structure overall $O(n \log^* n)$.

Theorem 1.4.14

The overall time spent on m FIND operations, throughout the lifetime of a Union-Find data structure defined over n elements is $O((n+m)\log^* n)$.