Krusgal's alg. for MST and disjoint sets

Assume we have a data structure of disjoint sets

$$\mathbb{S} = \{S_i\} \text{ such that } S_i \cap S_j = \phi$$

and supports the following operations.

- Make-Set (x) : $\mathbb{S} \leftarrow \mathbb{S} \cup \{\{x\}\}$
- Union (x, y) : $\mathbb{S} \leftarrow \mathbb{S} - \{S_1, S_2\} \cup \{S_1 \cup S_2\}$
  where $x \in S_1$ and $y \in S_2$.
- Find-Set (x) : returns a unique object that represents S
  where $x \in S$.

Krusgal's alg. for MST.

$T \leftarrow \emptyset$

for each $v \in V$

  do Make_Set(v)

sort E by increasing edge weight

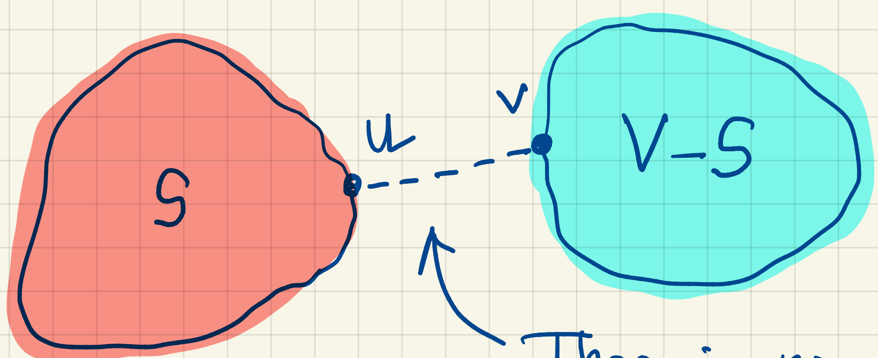
for each $(u,v) \in E$ (in sorted order)

  do if Find-Set$(u) \neq$ Find-Set(v) $\triangleright$ (u,v) make no cycle

    then $T \leftarrow T \cup \{(u,v)\}$

      Union (u,v) $\triangleright$ u & v now in same component

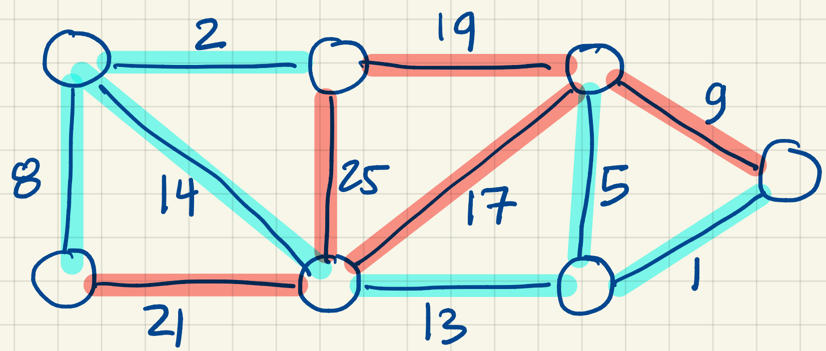
Why does it work? (previous proof)



$S = \{x \mid \text{Find\_Set}(u) = \text{Find-Set}(x)\}$

There is no smaller weight edge that crosses the cut (due to sorted order)

Example:



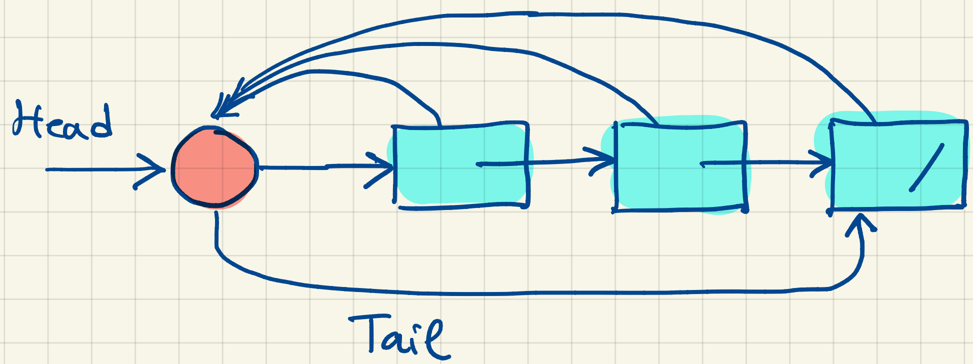
Time :

- Sorting: $\Theta(E \log E) = \Theta(E \log V)$ since graph is connected
  $= O(E \log V)$ in general since $E = O(V^2)$
  ($E = \Omega(V)$ if graph is connected)
- $\Theta(V)$ Make-Sets
- $\Theta(E)$ Find-Sets
- $O(V)$ Unions (exactly $|V|-1$)

We can implement m set operations on n elements in $O(m \cdot \alpha(m,n))$ time, where α grows very slowly. $\alpha(m,n) \leq 4$ if $m, n = 10^{80}$.

Disjoint_Set Implementations

- Each Set in a linked List of its elements. Each element point back to "Head" of List, which is the "representative" of the list.



- Using this implementation, Make_Set (x) and Find-Set (x) can be done in $O(1)$ time.

- Union (x, y) takes more time: "Copy" set of x into that of y, and update pointers. Using y's tail we can quickly identify where to append. This takes time proportional to size of x's set.

Worst case Scenario:

		# pointer updates
Union (x_1, x_2)	:	1
Union (x_2, x_3)	:	2
Union (x_3, x_4)	:	3
⋮		
Union (x_{n-1}, x_n)	:	$n-1$
		$\Theta(n^2)$ time.

- Improvement: Append smaller set to larger one (store size is set header)

A single Union operation can still take $\Theta(n)$ time; e.g. if both sets have $\frac{n}{2}$ elements.

- But m set operations, in which n are Make_Sets, will take $O(m + n\lg n)$ time. When an elem's back pointer is updated, its set at least doubles in size. This can happen at most $\lg n$ times per element.

Amortized analysis: Average time, but no probability

$n \leqslant m$

m operation take $O(m + n\lg n) = O(m\log n)$

So $O(\lg n)$ per operation.

We say each operation take $O(\lg n)$ amortized time.

Note: Average, but no bad sequences. Some operations will take more than $O(\lg n)$, but every sequence of m operations takes at most $O(m\lg n)$ time.

Amortized Analysis	Average-Case Analysis
No probability	May involve prob.
Obtain average performance in the worst-case	Average performance
No bad sequences	Possible bad sequences

Techniques for Amortized Analysis

- Aggregate analysis
- Accounting method
- Potential method.

What we did in disjoint sets was aggregate analysis.

Example: Stack with an added operation.

Push(S, x) : Push x onto stack

Pop(S) : Pop top of stack and return popped object

Multipop(S, k) : Remove $\min(|S|, k)$ objects from top.

Multipop(S, k)
    while not stack-empty(S) and $k \neq 0$
        do Pop(S)
            $k \leftarrow k - 1$

Running time of Multipop is $O(\min(|S|, k))$
which means in the worst-case it's $O(n)$

Therefore, a sequence of n operations takes $O(n^2)$ time.

Aggregate Analysis

Get a better bound by considering the entire sequence of n operations.

Claim: Any sequence of n stack operations take $O(n)$ time.

- Any object can be popped at most once after it's pushed.

The # times a pop is called on a non-empty stack (including those in a multipop) is at most equal to # pushes.

- Given n operations that result in m pops from within multipop, the running time is $O(n+m)$

But $m \le n$, so $O(2n) = O(n)$.

Each operation runs in $O(1)$ amortized time

Aggregate: "Treat every operation the same way in terms of time"