Lecture 14 CS2630 Sept 16

Generic Greedy algorithm is basically Kruskal's MST in disguise

Boruvka's algorithm

For every vertex, we find the closest vertex and add that edge to the MST.

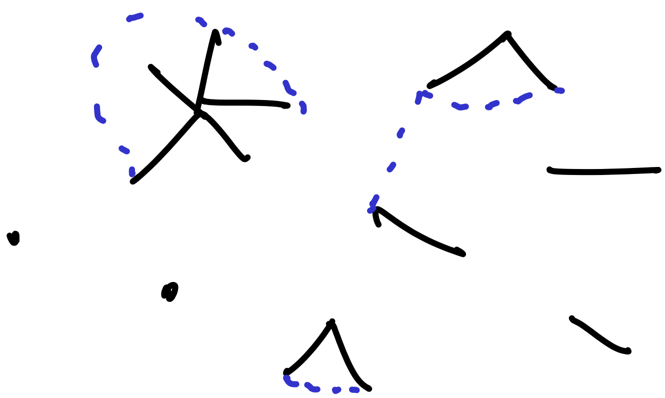
- It is easy to parallelize
- There is linear time randomized algorithm for MST based on Borüvka's algorithm

The time complexity of the greedy algorithm is dependent on

data structure dependent

1. how quickly we can detect independence
2. If so, actually update

In the context of MST we want to detect cycles due to the addition of an edge



For any edge (u,v) if u, v belong to different components we can add else discard

Find(u) Find(v)

return the component # of u/v

Process of adding (u,v) is Union

join component of u and v

Data Structure for Union Find

We have a family of (disjoint) subsets $\{S_i\}$ Given any element x we want to return j such that

n: total # elements

$x \in S_j$ Find(x)

$S_k \leftarrow S_{i_1} \cup S_{i_2}$ Union

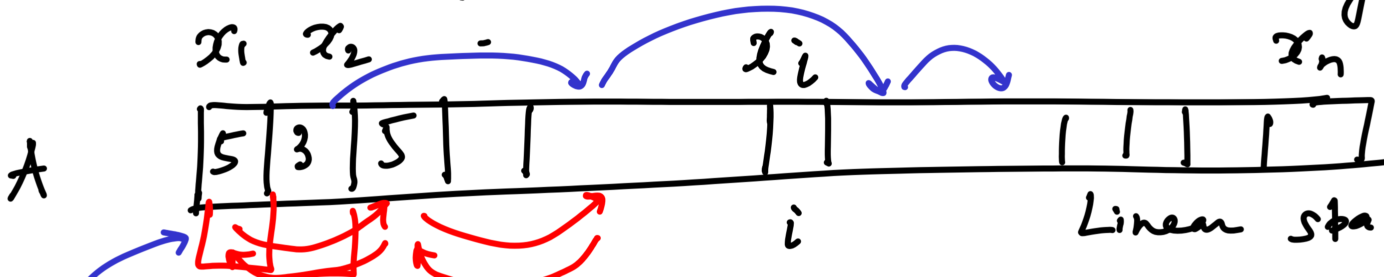
We can rename the union in the way that it suits us

$S_{i_1} \leftarrow S_{i_1} \cup S_{i_2}$

Approach 1

Based on maintaining an array of the label of each element

label ← identity of set

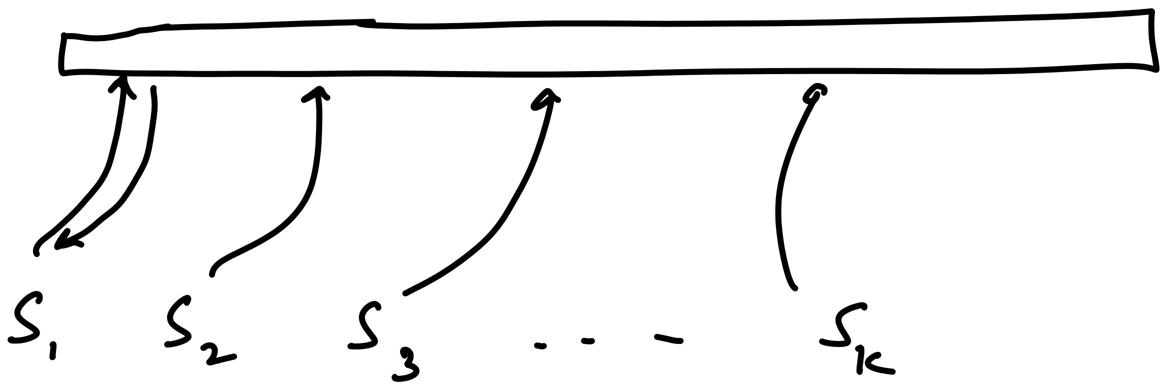


Linear space

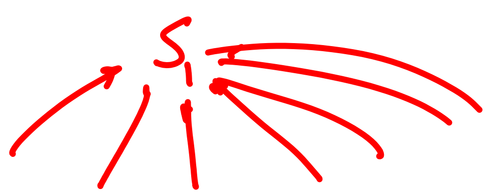
Find x_j : return $A[j]$ $O(1)$ time

SS Union

The elements of a set can be recovered by scanning the array : $O(n)$



Union S_1 S_2



Cost of union is the number of label changes $\leq$ size of the smaller set

$\leq O(n)$ (worst case)

What is cost of

n unions and $\underbrace{m}$ Finds

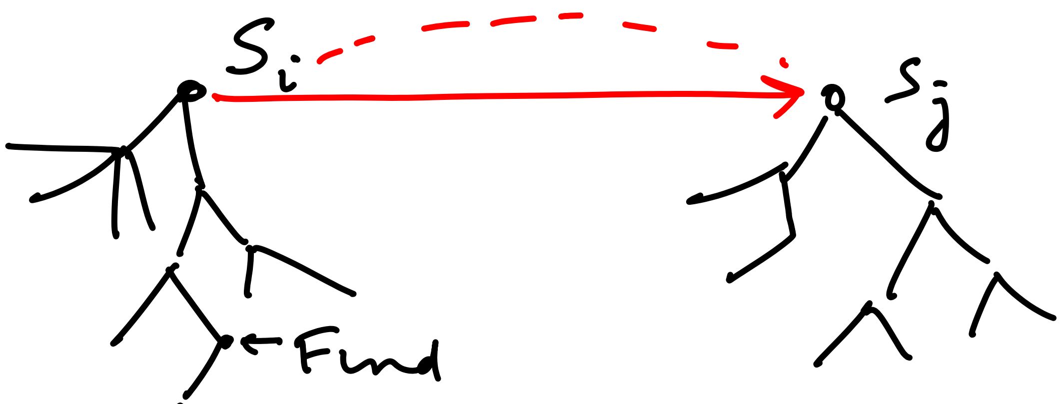
$O(n^2) + O(m)$

$O(n \log n)$

For any fixed element x, what is number of times, its label can change?

By changing the labels of the elements of the smaller set, the #label changes is bounded by $O(\log n) \Rightarrow$ $O(n \log n)$ for all elements $\Rightarrow$ cost of n unions is $O(n \log n)$

Tree representation of sets.



Cost of Find : length of the path to root : $O(n)$

Cost of Union : $O(1)$

Let us link the "smaller" tree to the "larger" tree. (height is more relevant)

<u>rank</u> of a tree : height

rank of a singleton node is 0

Two trees T_1, T_2 with ranks r_1, r_2

$r_1 < r_2$, if we make T_1 the child of the root of T_2 then $\text{rank}(T_2) = r_2$

If $r_1 = r_2$ then $\text{rank}(T_2) = r_2 + 1$

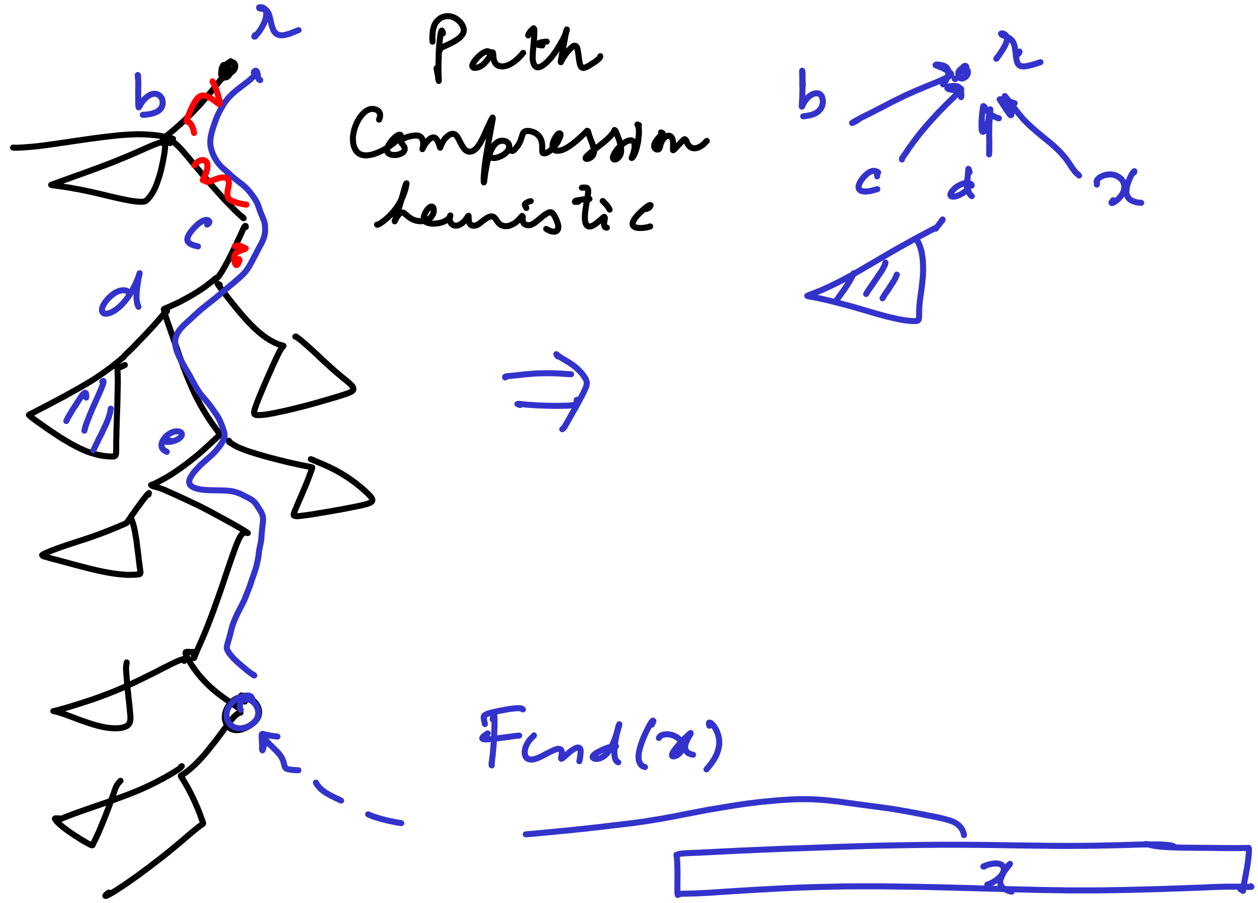
Union-heuristic : Make the tree with smaller rank the child of the root of the other tree.

<u>Claim</u> : The # of nodes in a tree with rank $r \geq 2^r$

Then cost of find opn is bound by $O(\log n)$

$\Rightarrow O(n + m \log n)$

Path Compression heuristic



To rigorously analyse the benefits

1. The rank function is preserved
2. The rank of an internal node remains fixed henceforth
3. Along any node $\rightsquigarrow$ root path the ranks are monotonically increasing.

$$r_{i_1} < r_{i_2} < r_{i_3} < r_{i_4} < \dots < r_{i_k} = \text{root}$$

The path compression heuristic gives us an $O(n \log^* n + m \log^* n)$ bound on m finds + n unions.

$$\log^* n = i \quad \text{if} \quad n \leq \left.2^{2^{2^{2}}}\right\} i$$

$$2^{100} < 2^{2^{2^{2^{2}}}} \qquad \log^*(2^{100}) < 4$$

$$\log^*\left(2^{2^{100}}\right) < 5$$