Lecture 16 The Union-Find Data Structure

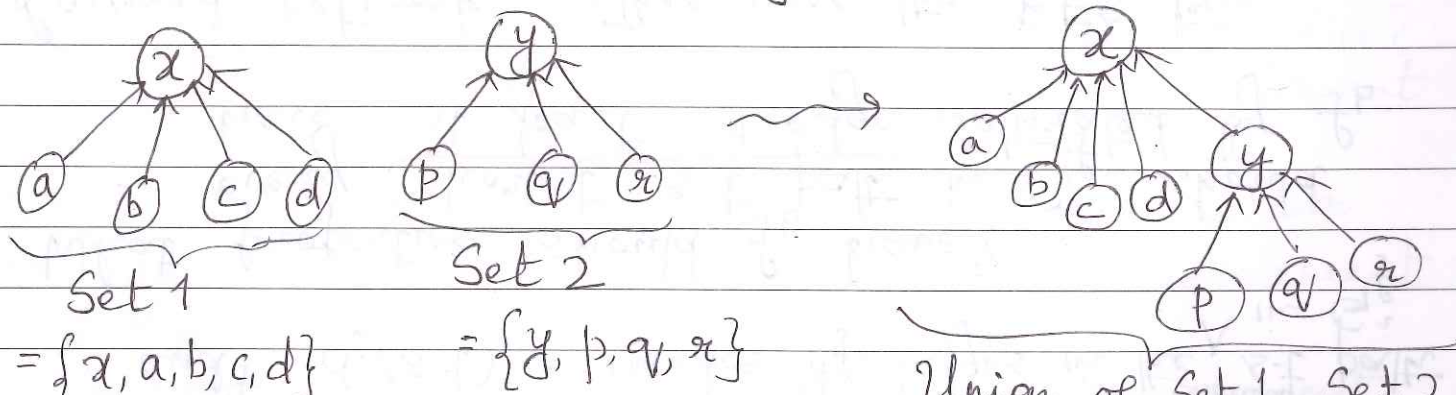
We have a universe of n elements (in the MST problem, these are the vertices). The universe is partitioned into components, i.e., disjoint sets.

- Given a pair of elements (u, v), we want to know if u and v are in the same component or in different components.
- If they are in different components, then we want to merge these 2 components into the same component.
  * FIND(u) ⇝ we have m such operations
  * Union(C, C') ⇝ we have $n-1$ such operations.

Last lecture we saw a solution that performs FIND in $O(1)$ time and Union in $O(\log n)$ amortized time.

Can we perform Union more efficiently?

- nodes in a set point to a common location containing the representative or leader of this set. Then we can perform Union in $O(1)$ time as follows:

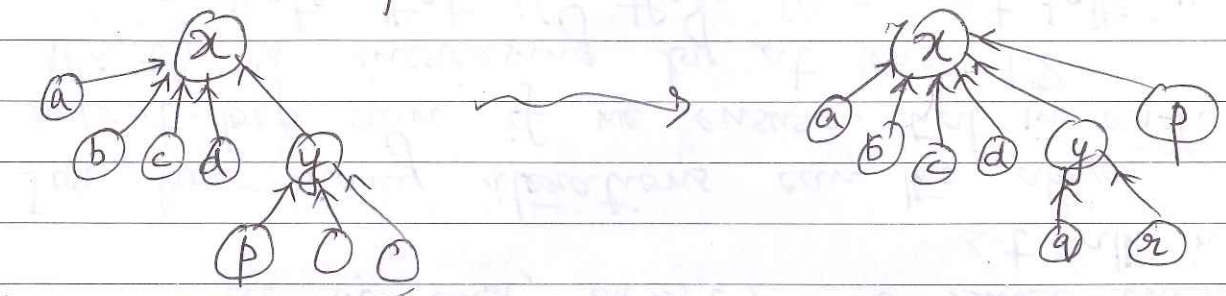


Set 1 = $\{x, a, b, c, d\}$ Set 2 = $\{y, p, q, r\}$ Union of Set 1, Set 2

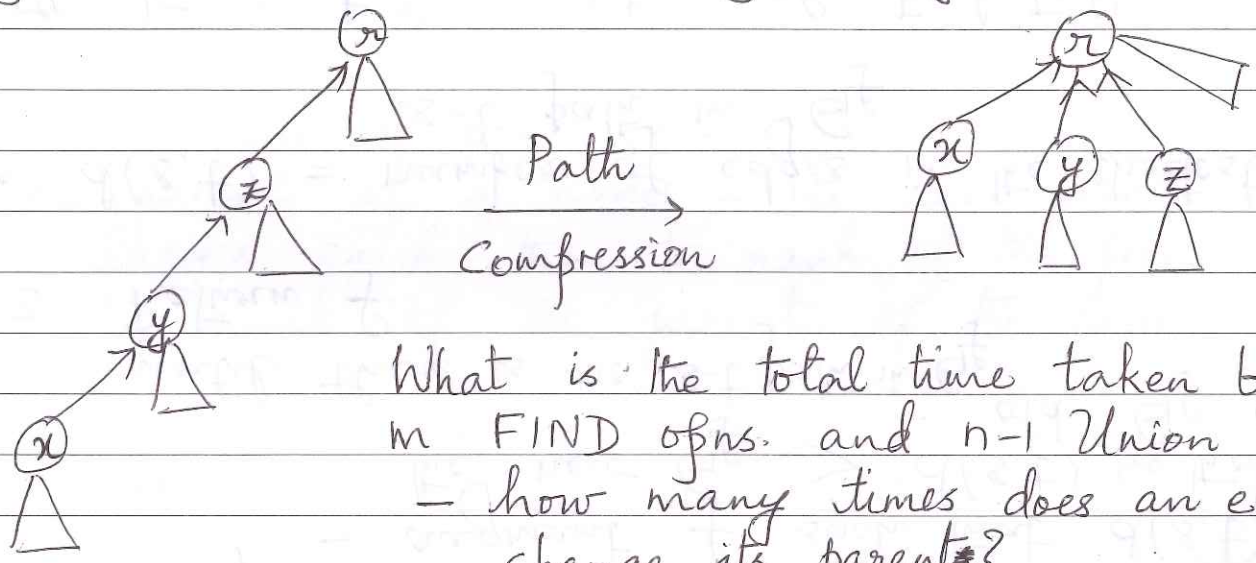
However the Union set has higher depth than Set 1 or Set 2 → this will make FIND more expensive. When we call FIND(p), we have to follow 2 parent pointers to know the leader of this set.

When we merge sets as indicated above, which tree becomes the parent?
- the one with larger height.

Improving find: Each find can also reduce the tree depth, at no extra cost. For instance,



Suppose we call FIND(p). After that, let us make p a child of x so that the next time we call FIND(p), it will take just 1 parent pointer traversal. More generally,



What is the total time taken by the m FIND ops. and n-1 Union ops?
- how many times does an element change its parent?

Let the height of an element be the number of edges in the longest path between this element and a descendant leaf.

* let us maintain "rank" of an element which is an upper bound on its height.

- When we do Union, link the root of lower rank to the root of higher rank.

The ranks are unchanged by this.
However if the 2 ranks are equal, then arbitrarily choose one root as the parent and increase its rank.
- Path compression does not change ranks.

Analysis of union by rank & FIND with path compression:

* a node with rank i has a subtree of size $\geq 2^i$. (why?)
* the parent of a node (call it x) has rank strictly larger than the rank of x.
* the final rank of a node is frozen when it ceases to be a root.

The number of rank i nodes $\leq n/2^i$ (why?)
So $i \leq \log n$.

Each unit of time spent in a FIND changes the parent of a node - the new parent has a larger rank than the rank of the previous parent. So the rank of the parent of the node keeps going up. A node changes parents $\leq \log n$ times. So all FIND operations take $O(m + n\log n)$ time.
[Please show this formally.] This is no better than what we had earlier.

An improved analysis

Partition ranks as follows: 0 | 1 | 2 3 | 4 ... 15 | 16 ... $2^{16}-1$ | ...
- here we put ranks $k \ldots 2^k - 1$ in one block.
- how many blocks are there?

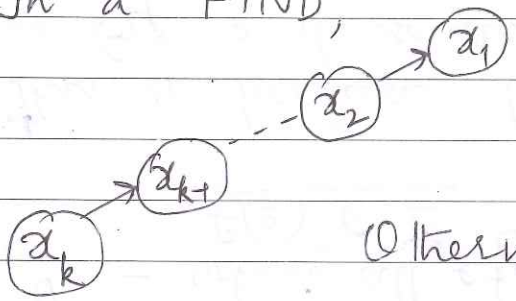
Let $\log^* n$ be the smallest t such that a tower of 2's of height t has value $\geq n$.

Ex: $\log^* 100 = 4$ since $2^{2^{2^2}} = 2^{16} > 100$, however $2^{2^2} < 100$.

Exercise. Show that the number of blocks is $1 + \log(\log^* n) = \log^* n$ ← this is a very slow growing function

Let us continue our improved analysis.

In a FIND,



colour an edge (x_i, x_{i+1}) on this path **blue** if $\text{block}(\text{rank}(x_{i+1})) \neq \text{block}(\text{rank}(x_i))$.

Otherwise colour this edge **red**.

The total **blue** work done by a single FIND operation $\leq \log^* n$. (since there are $\log^* n$ blocks)

So the total **blue** work done by m FIND operations $\leq m \log^* n$.

Red work. Consider a particular block $i \ldots 2^i - 1$.

The number of nodes in this block

$$\leq \frac{n}{2^i} + \frac{n}{2^{i+1}} + \ldots \leq \frac{2 \cdot n}{2^i}.$$

The number of traversals of red edges incident on x (of rank in $\{i, i+1, \ldots, 2^i - 1\}$) $\leq 2^i$ since each such traversal increases the rank of x's parent and the number of nodes in this block $\leq 2^i$.

The total **red** work done by all nodes in x's block $\leq$ (number of nodes in this block) $\cdot 2^i \leq 2n$.

The total **red** work done by all nodes $\leq 2n \log^* n$ (since there are $\log^* n$ blocks)

So the total work done for m FIND ops. and $(n-1)$ Union ops. is $O((m+n)\log^* n)$.

Primality Testing

Input: an odd integer $n \geq 3$.

Goal: Design an algorithm whose running time is $\text{poly}(\log n)$ to determine if n is prime or composite.

($\log n$ is the input length)

We will show a Monte Carlo algorithm — so our algo. is allowed to make mistakes, however we have to show error probability $\leq 1/4$

The following result, also known as little theorem of Fermat, will be very useful to us here.

Result If n is prime and $a \in \{1, \ldots, n-1\}$ then $a^{n-1} = 1 \pmod n$.

this means the remainder obtained when a^{n-1} is divided by n is 1.

For example, take $n = 5$.

Easy to check that $1^4 = 1 \pmod 5$, $2^4 = 1 \pmod 5$, $3^4 = 1 \pmod 5$, $4^4 = 1 \pmod 5$.

Proof of the above result

If n is prime then every $a \in \{1, \ldots, n-1\}$ has gcd 1 with n. Let $\mathbb{Z}_n^* = \{1, \ldots, n-1\}$. Please check that $\mathbb{Z}_n^*$ is a group wrt multiplication mod n.

For any $a \in \mathbb{Z}_n^*$, $\langle a \rangle = \{a, a^2, a^3, \ldots, \underset{\text{H}}{a^k}\}$ is a subgroup of $\mathbb{Z}_n^*$.

Fact (Lagrange's theorem) Let G be any finite group and let H be any subgroup of G. Then $|H|$ divides $|G|$.

Assuming this fact, we have k divides $n-1$,

i.e., $k \cdot t = n-1$. So $a^{n-1} = a^{kt} = 1 \pmod n$ since $a^k = 1 \pmod n$.

Proof of the above fact. If $H = G$ then done. Else let $a \notin H$. Observe that aH is disjoint from H — otherwise we have $ah_1 = h_2 \Rightarrow a = h_2 h_1^{-1}$ so $a \in H$, a contradiction.

We also have $|aH| = |H|$.

If $H \cup aH = G$ then $|H| = |G|/2$, so done.

Else let $b \notin H \cup aH$. Observe that bH is disjoint from $H \cup aH$. Also $|bH| = |H|$ and so on...

Our first attempt

1. Pick an $a \in \{1, 2, \ldots, n-1\}$ uniformly at random.
2. Compute $a^{n-1} \bmod n$.
   - if this is not 1 then return "composite" else return "prime".

Question. What is the time needed to compute $a^{n-1} \bmod n$? In other words, can we do it in $\text{poly}(\log n)$ time?

Let us analyze the above algo. What happens when n is prime? Observe that the algo. returns "prime".

What happens when n is composite?
- Suppose $\gcd(a, n) \neq 1$. What does the algo. return?
- Suppose $\gcd(a, n) = 1$.

For any n, let $\mathbb{Z}_n^*$ be the set of elements in $\{1, \ldots, n-1\}$ that are relatively prime to n.
That is, $\mathbb{Z}_n^* = \{a \in \{1, \ldots, n\} : \gcd(a, n) = 1\}$.

Exercise. Show that $\mathbb{Z}_n^*$ is a group wrt multiplication modulo n.

Date

We will continue this in the next lecture.