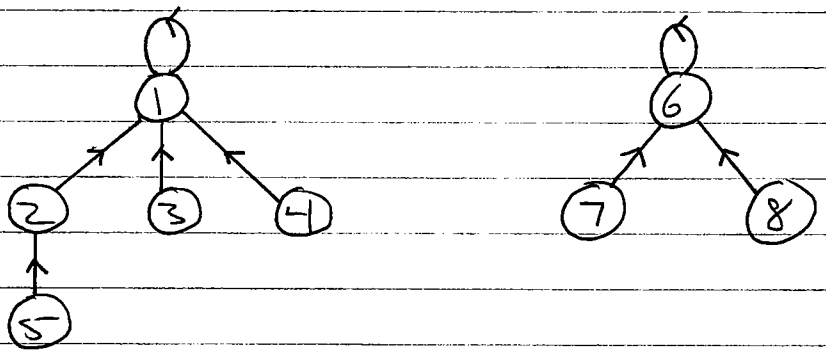
(21.3) Disjoint Set Forests

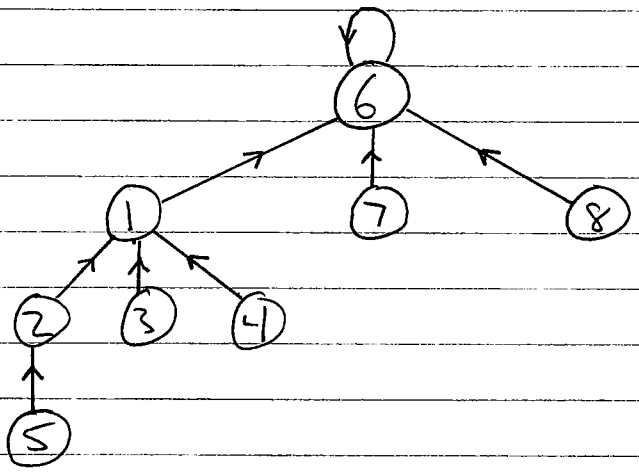
Another implementation strategy is to represent each set in $\mathcal{S}$ by a rooted tree, i.e. $\mathcal{S}$ is a <u>disjoint set forest</u>.

Each node points only to its parent. The root of each tree is it's own parent, and is it's set's representative.

<u>Ex</u>



Result of Union(3, 8):



<u>Convention</u>: Union(x, y) causes Rep[x] to point to Rep[y].

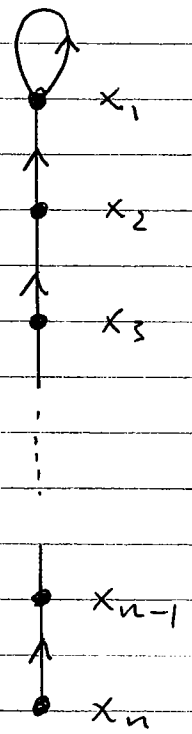
MakeSet CREATES A TREE WITH JUST ONE NODE WHICH POINTS TO ITSELF. Union CAUSES THE ROOT OF ONE TREE TO POINT TO THE ROOT OF ANOTHER. BOTH OPERATIONS HAVE COST $\Theta(1)$.

FindSet FOLLOWS PARENT POINTERS UNTIL THE ROOT IS REACHED. THE NODES VISITED ON THIS PATH TOWARD THE ROOT CONSTITUTE A FIND PATH. THE COST OF FindSet IS Θ(length-of-findPath).

IT IS POSSIBLE USING THESE OPERATIONS TO CREATE A TREE WHICH IS SIMPLY A LINEAR LINKED LIST.

EX.

MakeSet(x_1)
⋮
MakeSet(x_n)
Union(x_n, x_{n-1})
Union(x_{n-1}, x_{n-2})
⋮
Union(x_3, x_2)
Union(x_2, x_1)



THIS SEQUENCE IS NOT BAD IN ITSELF (RUN TIME $\Theta(2n-1) = \Theta(m)$), BUT IT CREATES A TREE IN WHICH FindSet will RUN slowly (i.e. IN $\Theta(n)$ TIME.)

TWO HEURISTICS ARE USED TO IMPROVE THE RUN TIME OF SEQUENCES WHICH INCLUDE MANY FindSet OPERATIONS.

Union By RANK

FOR EACH NODE x, MAINTAIN A VALUE rank[x], WHICH IS AN UPPER BOUND ON THE HEIGHT OF x.

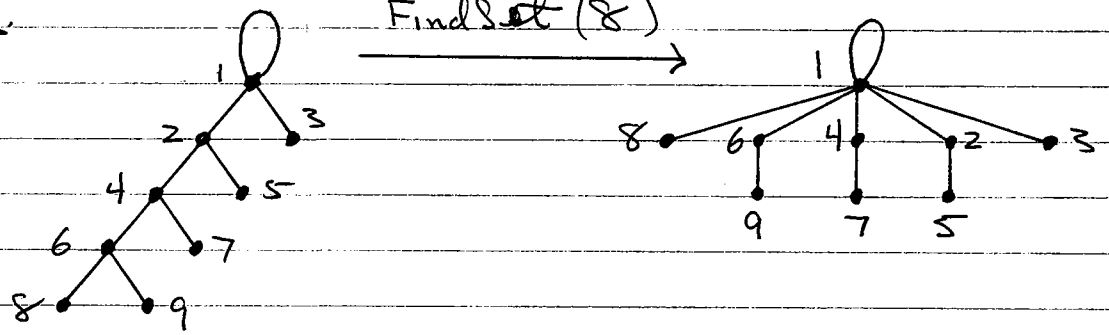
$$height(x) \leq rank[x]$$

MODIFY Union SO THAT THE ROOT WITH SMALLER RANK IS MADE TO POINT TO THE ROOT WITH LARGER RANK.

PATH COMPRESSION

THIS IS A MODIFICATION OF FindSet WHICH CAN REDUCE THE HEIGHT OF A TREE. IT INVOLVES TWO PASSES: PERFORM ONE PASS UP THE FIND PATH TO FIND THE ROOT, THEN MAKE ANOTHER PASS DOWN THE FIND PATH CAUSING EACH NODE ALONG THAT PATH TO POINT TO THE ROOT.

Ex.



FindSet(8) ALTERS THE TREE AS ABOVE, THEN RETURNS 1.

MakeSet(x)

1.) $p[x] \leftarrow x$
2.) $rank[x] \leftarrow 0$

Union(x, y) (USING UNION-BY-RANK)

1.) Link(FindSet(x), FindSet(y))

Link(x, y) (Pre: x, y BOTH ROOT NODES)

1.) if $rank[x] > rank[y]$
2.) $\quad p[y] \leftarrow x$
3.) else
4.) $\quad p[x] \leftarrow y$
5.) $\quad$ if $rank[x] = rank[y]$
6.) $\quad\quad rank[y] \leftarrow rank[y] + 1$

NOTE: AS LONG AS ONLY MakeSet AND Link OPERATIONS ARE PERFORMED, $rank[x]$ IS ACTUALLY EQUAL TO $height(x)$. THIS CAN BE PROVED BY INDUCTION ON THE NUMBER OF Link OPERATIONS PERFORMED.

FindSet(x) (USING PATH COMPRESSION)
1.) if $x \neq P[x]$
2.) $P[x] \leftarrow$ FindSet($P[x]$)
3.) return $P[x]$.

WE TRACE FindSet ON THE PRECEDING EXAMPLE:

FindSet(8): 1.) $8 \neq P[8]$
2.) $P[8] \leftarrow$ FindSet(6)
3.) return 1

FindSet(6) 1.) $6 \neq P[6]$
2.) $P[6] \leftarrow$ FindSet(4)
3.) return 1

FindSet(4) 1.) $4 \neq P[4]$
2.) $P[4] \leftarrow$ FindSet(2)
3.) return 1

FindSet(2) 1.) $2 \neq P[2]$
2.) $P[2] \leftarrow$ FindSet(1)
3.) return 1

FindSet(1) 1.) $1 = P[1]$
2.) ~~~
3.) return 1

NOTE THAT FindSet(x) MAY REDUCE THE HEIGHT OF THE TREE CONTAINING x, WHILE IT DOES NOT ALTER ANY RANKS. THUS FindSet OPERATIONS MAINTAIN THE INEQUALITY

$$height(x) \leq rank[x]$$

FOR ALL NODES x.

THE UNION-BY-RANK AND PATH COMPRESSION HEURISTICS WORK TOGETHER TO KEEP THE HEIGHTS OF TREES SMALL, WHICH LOWERS THE COST OF FindSet.

THEOREM

USING THE UNION-BY-RANK AND PATH COMPRESSION HEURISTICS, ANY SEQUENCE OF m MakeSet, Union, AND FindSet OPERATIONS, n OF WHICH ARE MakeSets, CAN BE PERFORMED IN TIME

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

WHERE $\alpha(n)$ IS AN EXTREMELY SLOW GROWING FUNCTION (DEFINED IN 21.4). IN ANY CONCEIVABLE APPLICATION $\alpha(n) \leq 4$, SO THE ABOVE RUN TIME IS EFFECTIVELY LINEAR, i.e. $O(m)$.

AMORTIZING THIS COST OVER ALL OPERATIONS GIVES $O(1)$ TIME PER OPERATION ON AVERAGE, WHICH IS VERY EFFICIENT.

BACK TO SEC. 23.2

RECALL THAT KRUSKAL'S ALGORITHM GROWS SEVERAL TREES SIMULTANEOUSLY UNTIL THEY MERGE INTO A SINGLE MWST. THE IMPLEMENTATION BELOW USES A DISJOINT SET DATA STRUCTURE TO MAINTAIN A COLLECTION OF VERTEX SETS. EACH SET IN THIS COLLECTION IS THE VERTEX SET OF ONE OF THE TREES UNDER CONSTRUCTION.

NOTE THE SIMILARITY OF THIS ALGORITHM TO THE ALGORITHM FOR DETERMINING CONNECTED COMPONENTS FROM 21.1.

```
KRUSKAL(G, w)
1.) A ← ∅
2.) for each v ∈ V
3.)     MakeSet(v)
4.) SORT E BY WEIGHT w (NON-DECREASING)
5.) for each (u,v) ∈ E   (IN SORTED ORDER)
6.)     if FindSet(u) ≠ FindSet(v)
7.)         A ← A ∪ {(u,v)}
8.)         Union(u,v)
9.) return A
```

ON EACH ITERATION OF LOOP 5-8 WE PICK THE LIGHTEST EDGE JOINING TWO DISTINCT TREES AND ADD THAT TO A. I.E. AMONGST ALL EDGES WHOSE ADDITION TO A WOULD NOT CREATE A CYCLE, WE PICK ONE OF MINIMUM WEIGHT. THE RESULTING SET $A \subseteq E$ CONSTITUTES THE EDGES OF A MWST IN G.

Run Time

Assume we use the disjoint set forest implementation with union-by-rank and path compression heuristics, which is the asymptotically fastest known.

Initialization of A in line 1 costs $O(1)$ time. Loop 2–3 does $|V|$ MakeSet operations and loop 5–8 $O(|E|)$ FindSet and Union operations. Thus the disjoint set operations have total cost $O((|V|+|E|)\cdot\alpha(|V|))$.

Now since G is connected (otherwise there is no spanning tree) we have $|E| \geq |V|-1$, whence $|V| \leq |E|+1 = O(|E|)$, so the disjoint set operations cost $O(|E|\alpha(|V|))$

The sort on line 4 costs $O(|E|\lg|E|)$ time (assuming we use MergeSort or HeapSort, or a similarly efficient sort.)

Also note that $\alpha(|V|) = O(\lg|V|) = O(\lg|E|)$, so the run time of Kruskal is

$$O(|E|\lg|E|)$$

But also since G is simple, $|E| < |V|^2$, so $\lg|E| = O(\lg|V|)$, and the run time of Kruskal can be expressed as

$$O(|E|\lg|V|)$$

which is the same as Prim.