Disjoint Set systems
MakeSet(x)
Find(x)
Union(x,y)

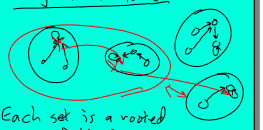
Ex: connected components in a graph

1. For each vertex v, call MakeSet(v)
2. Add edges one by one. At each stage, the sets are the connected components:
For each edge e=(u,v):
Union(u,v)

(at any point, to see if two vertices are in the same connected component, this is Find(u)=Find(v))

Implementation

Disjoint set forest



Each set is a rooted tree of its items.
Each item points to its parent, except the root, which points to itself and the representative

MakeSet(x):
a.parent := a

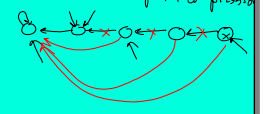
Find(a)
if a ≠ a.parent
return Find(a.parent)
else
return a

Union(x, y)
a := Find(x)
b := Find(y)
a.parent := b

Union-by-rank: each item x has a rank attribute which is an upper bound on the height of the subtree rooted at x.

MakeSet(a)
a.parent := a
a.rank := 0
Find(x) — same as before
Union(x, y)
a := Find(x)
b := Find(y)
if a.rank < b.rank
a.parent := b
else if a.rank > b.rank
b.parent := a
else // a.rank == b.rank
arbitrary choice → a.parent := b
b.rank := b.rank + 1

So far: x.rank equals the height of its subtree.
One more trick: path compression



MakeSet, Union (by_rank)
same as before.

Find(x)
if x ≠ x.parent
x.parent := Find(x.parent)
end if
return x.parent

rank is not necessarily equal to the height of the subtree if path compression is used, but it is still an upper bound on the height, so Union-by-rank still works efficiently.

Work union-by-rank & path compression
m operations, including
n MakeSets