Disjoint Sets with Arrays

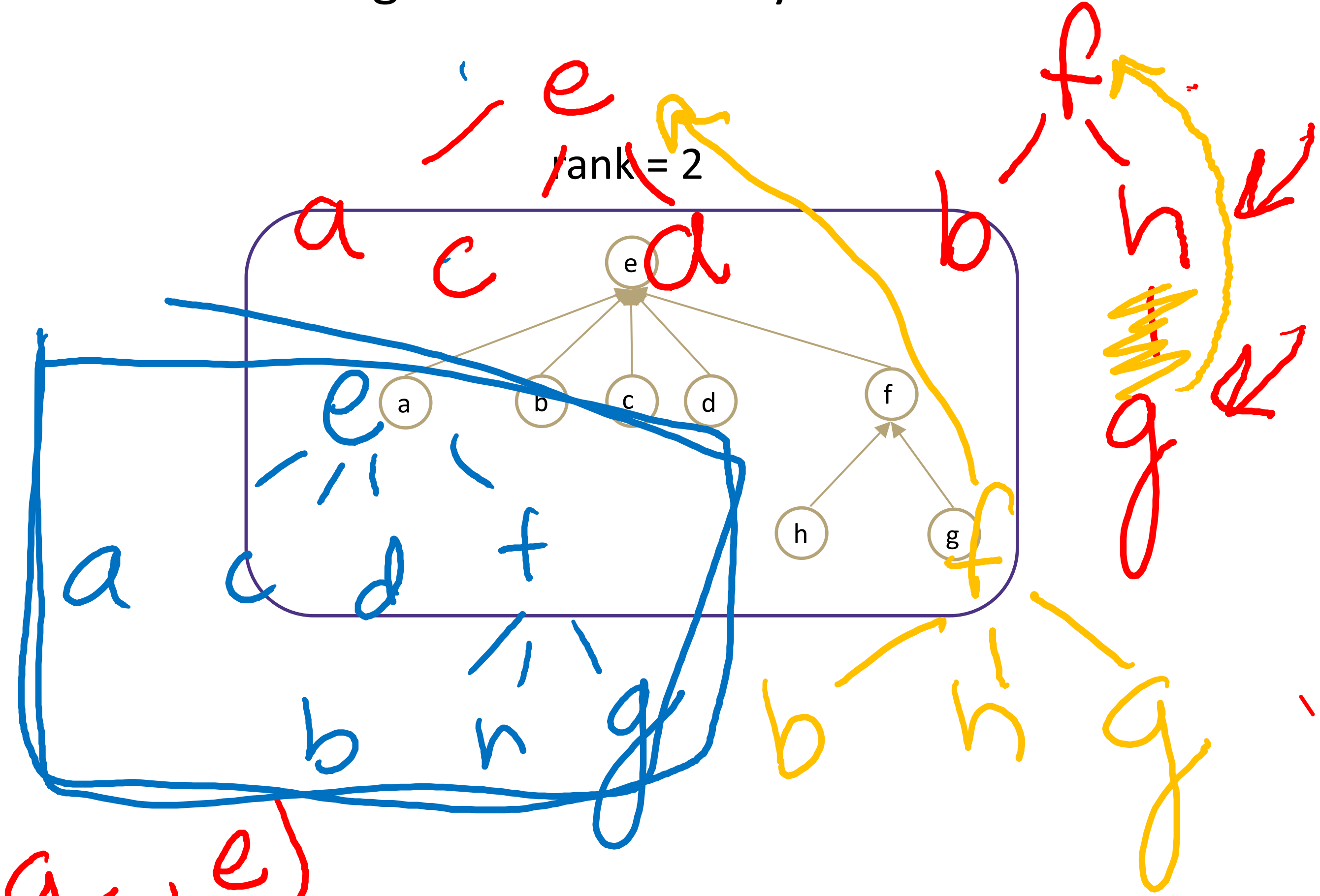
Data Structures and Algorithms

Warm Up

Using the union-by-rank and path-compression optimized implementations of disjoint-sets draw the resulting forest caused by these calls:

1. makeSet(a)
2. makeSet(b)
3. makeSet(c)
4. makeSet(d)
5. makeSet(e)
6. makeSet(f)
7. makeSet(h)
8. union(c, e)
9. union(d, e)
10. union(a, c)
11. union(g, h)
12. union(b, f)
13. union(g, f)
14. union(b, c)

union(g, e)



TreeDisjointSet<E>

state

```
Collection<TreeSet> forest
Dictionary<NodeValues,
NodeLocations> nodeInventory
```

behavior

```
makeSet(x)-create a new tree
of size 1 and add to our
forest
findSet(x)-locates node with x
and moves up tree to find root
union(x, y)-append tree with y
as a child of tree with x
```

Reminders:

- Union-by-rank: make the tree with the larger rank the new root, absorbing the other tree. If ranks are equal pick one at random, increase rank by 1
- Path-compression: when running findSet() update parent pointers of all encountered nodes to point directly to overall root
- Union(x, y) internally calls findSet(x) and findSet(y)

Warm Up

Using the union-by-rank and path-compression optimized implementations of disjoint-sets draw the resulting forest caused by these calls:

```
1.makeSet(a)
2.makeSet(b)
3.makeSet(c)
4.makeSet(d)
5.makeSet(e)
6.makeSet(f)
7.makeSet(g)
8.makeSet(h)
9.union(c, e)
10.union(d, e)
11.union(a, c)
12.union(g, h)
13.union(b, f)
14.union(g, f)
15.union(b, c)
```

https://courses.cs.washington.edu/courses/cse373/18sp/files/slides/disjoint_set_warmup.pdf

TreeDisjointSet<E>

state

`Collection<TreeSet> forest`

`Dictionary<NodeValues, NodeLocations> nodeInventory`

behavior

`makeSet(x)`-create a new tree of size 1 and add to our forest

`findSet(x)`-locates node with x and moves up tree to find root

`union(x, y)`-append tree with y as a child of tree with x

Reminders:

- Union-by-rank: make the tree with the larger rank the new root, absorbing the other tree. If ranks are equal pick one at random, increase rank by 1
- Path-compression: when running findSet() update parent pointers of all encountered nodes to point directly to overall root
- Union(x, y) internally calls findSet(x) and findSet(y)

Administrivia

Monday	Tuesday	Wednesday	Thursday	Friday
5/21 Disjoint Sets		5/23 Implementing Disjoint Sets	5/24 Interview Prep	5/25 P vs NP HW 6 due HW 7 out
5/28 Memorial Day		5/30 Final Review	5/31 Final Review	6/1 Tech Interview Prep HW 7 due
	6/5 Final @ 8:30am			

Sorry, Kasey's email is DEEP
Want a meeting? Email me this week for times next week
Have ANY grading questions/concerns, email Kasey by this weekend
TA lead review TBA
Alternative testing time TBA

Optimized Disjoint Set Runtime

makeSet(x)

Without Optimizations **O(1)**

With Optimizations **O(1)**

findSet(x)

Without Optimizations **O(n)**

With Optimizations **Best case: O(1) Worst case: O(logn)**

union(x, y)

Without Optimizations **O(n)**

With Optimizations **Best case: O(1) Worst case: O(logn)**

Kruskal's

t_m = time to make MSTs
t_f = time to find connected components
t_u = time to union

```
KruskalMST(Graph G)
   initialize each vertex to be a connected component    O(V*t_m)
   sort the edges by weight                                O(ElogE) / O(ElogV)
   foreach(edge (u, v) in sorted order){
      if(u and v are in different components){          O(V*t_u+E*t_f)
         add (u,v) to the MST
         Update u and v to be in the same component
      }
   }
```

$t_m = O(1)$
$t_f = O(\log V)$
$t_u = O(\log V)$

```
KruskalMST(Graph G)
   initialize a disjointSet, call makeSet() on each vertex    O(V)
   sort the edges by weight                                     O(ElogV)
   foreach(edge (u, v) in sorted order){                        O(E)
      if(findSet(u) != findSet(v)){                             O(logV)
         add (u,v) to the MST
         union(u, v)                                            O(logV)
      }
   }
```

$O(V + E\log V + E\log V)$

Aside: $O(V + E\log V + E)$ if you apply ackermann

```
KruskalMST(Graph G)
   initialize a disjointSet, call makeSet()
on each vertex
   sort the edges by weight
   foreach(edge (u, v) in sorted order){
      if(findSet(u) != findSet(v)){
         add (u,v) to the MST
         union(u, v)
      }
   }
```

```
KruskalMST(Graph G)
   initialize a disjointSet, call makeSet()
on each vertex
   sort the edges by weight
   foreach(edge (u, v) in sorted order){
      if(findSet(u) != findSet(v)){
         union(u, v)
      }
   }
```

Implementation

Use Nodes?

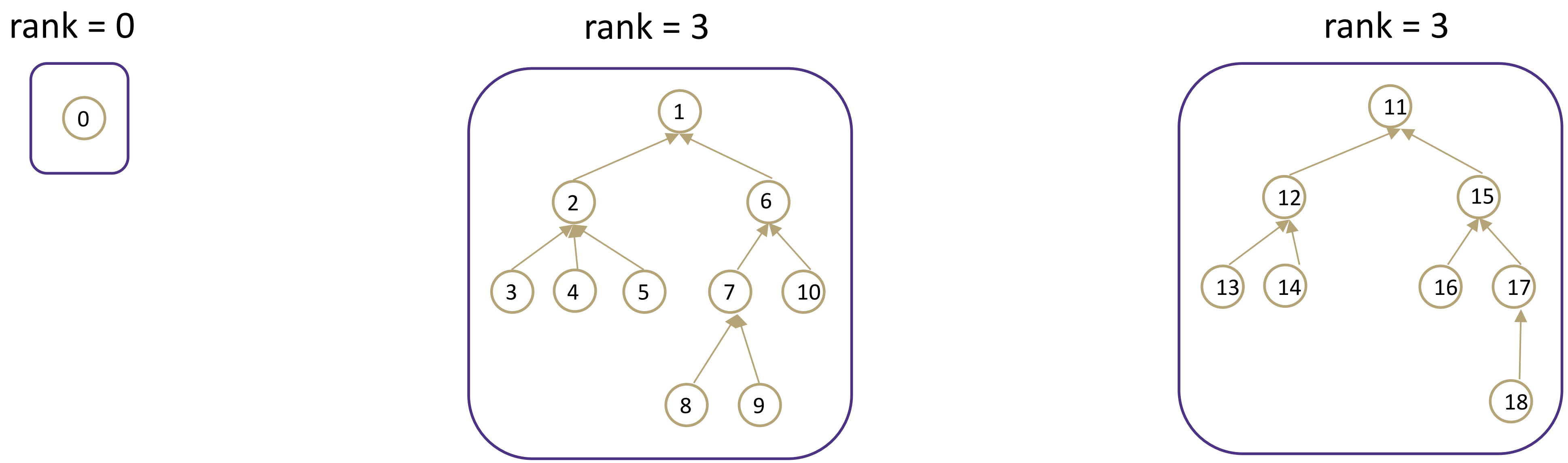
In modern Java (assuming 64-bit JDK) each object takes about 32 bytes

- int field takes 4 bytes
- Pointer takes 8 bytes
- Overhead ~ 16 bytes
- Adds up to 28, but we must partition in multiples of 8 => 32 bytes

Use arrays instead!

- Make index of the array be the vertex number
  - Either directly to store ints or representationally
  - We implement makeSet(x) so that **we** choose the representative
- Make element in the array the index of the parent

Array Implementation

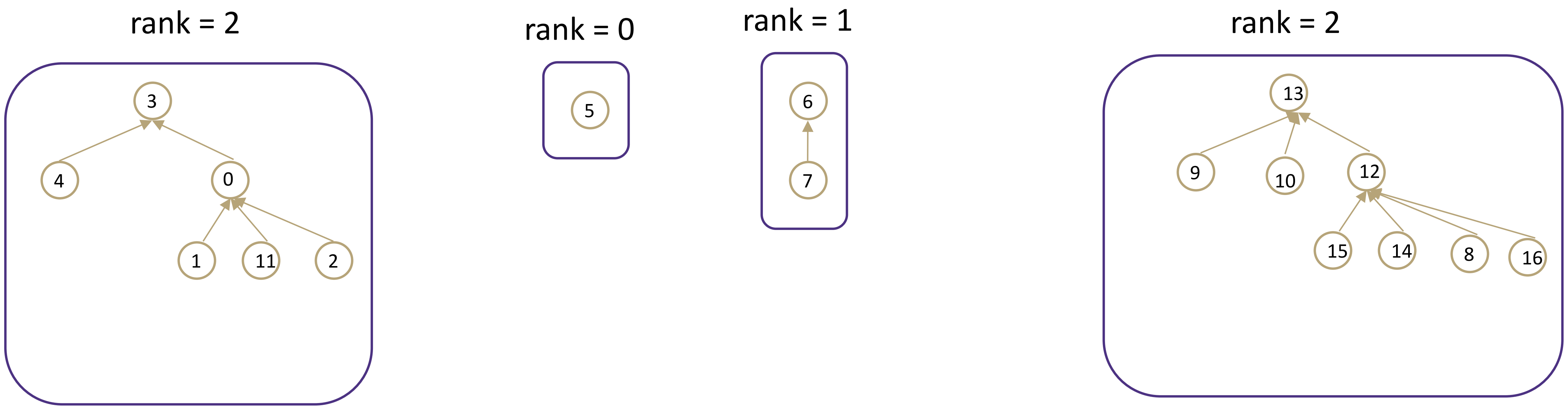


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
-1	-4	1	2	2	2	1	6	7	7	7	-4	11	12	12	11	15	15	17

Store (rank * -1) - 1

Each "node" now only takes 4 bytes of memory instead of 32

Practice



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	0	0	-3	3	-1	-2	6	12	13	13	0	13	-3	12	12	12

Array Method Implementation

makeSet(x)

add new value to array with a rank of -1

findSet(x)

Jump into array at index/value you’re looking for, jump to parent based on element at that index, continue until you hit negative number

union(x, y)

findSet(x) and findSet(y) to decide who has larger rank, update element to represent new parent as appropriate

Graph Review

Graph Definitions/Vocabulary
- Vertices, Edges
- Directed/undirected
- Weighted
- Etc…

Graph Traversals
- Breadth First Search
- Depth First Search

Finding Shortest Path
- Dijkstra's

Topological Sort

Minimum Spanning Trees
- Primm's
- Kruskal's

Disjoint Sets
- Implementing Kruskal's

Interview Prep

Treat it like a standardized test
- Cracking the Coding Interview
- Hackerrank.com
- Leetcode.com

Typically 2 rounds

Tech screen

"on site" interviews

4 general types of questions
- Strings/Arrays/Math
- Linked Lists
- Trees
- Hashing
- Optional: Design

It's a conversation!
1. T – Talk
2. E – Examples
3. B – Brute Force
4. O – Optimize
5. W – Walk through
6. I - Implement
7. T – Test