CSE 373: Data Structures and Algorithms

More on disjoint sets

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...

Today

Revisiting Kruskal's algorithm (with union/find operations)

Array representation of a disjoint set

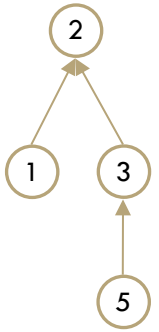
Review: Answering algorithm design questions

Set and union

Review

Set:

- Collection of elements.
- We want to identify all elements in a set with one value, so that findSet(x) == findSet(y) if x and y in one set.
- In tree structure, every node has one root.
- So use a tree structure for our set, and identified the set as the root element



Union of two elements is union of sets of those elements:

- Combine the two trees

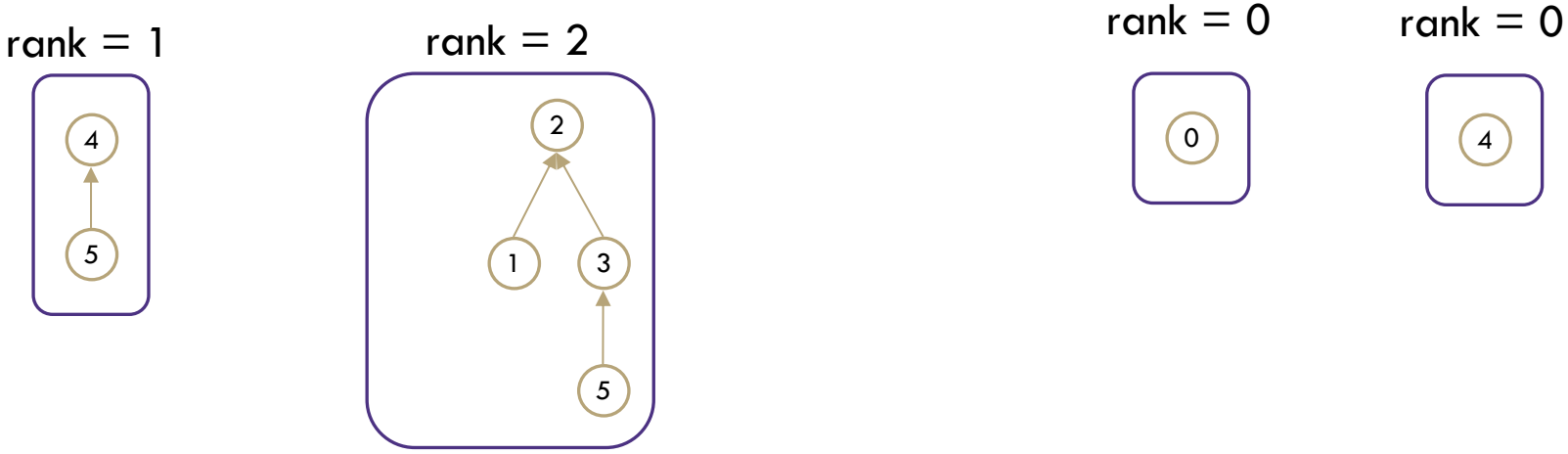
Improving union

Review

Problem: Trees can be unbalanced

Solution: Union-by-rank!

- let rank(x) be a number representing the upper bound of the height of x so rank(x) $\geq$ height(x)
- Keep track of rank of all trees
- When unioning make the tree with larger rank the root
- If it's a tie, pick one randomly and increase rank by one



Improving findSet()

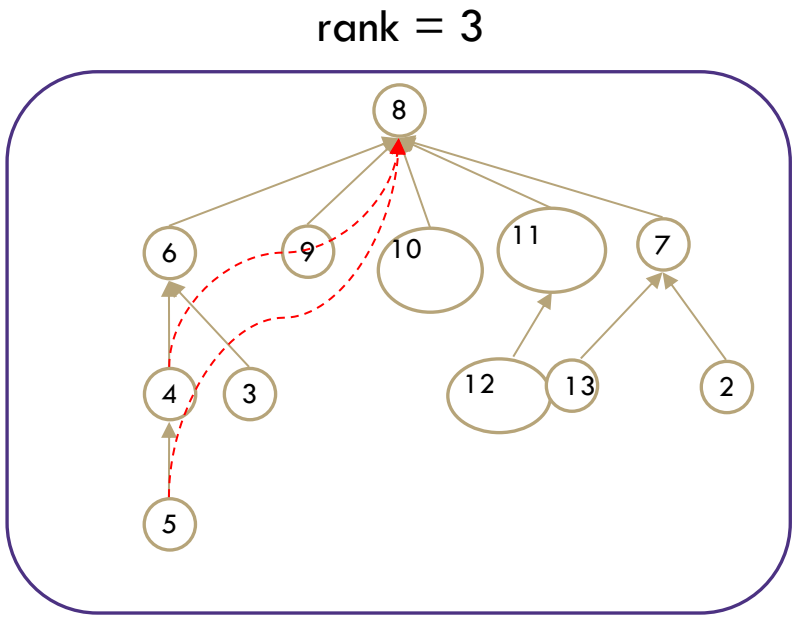
Review

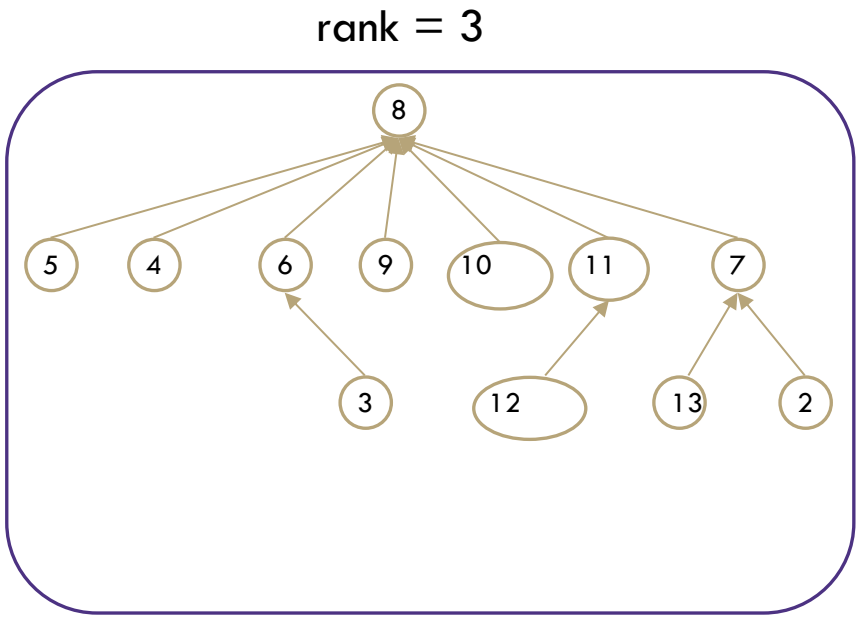
Problem: Every time we call findSet() you must traverse all the levels of the tree to find representative

Solution: Path Compression

- Collapse tree into fewer levels by updating parent pointer of each node you visit
- Whenever you call findSet() update each node you touch to point directly to overallRoot

```
findSet(5)
```





Optimized disjoint set runtime

makeSet(x)

Without Optimizations	**O(1)**
With Optimizations	**O(1)**

findSet(x)

Without Optimizations	**O(n)**
With Optimizations	**Worst case: O(logn)**

union(x, y)

Without Optimizations	**O(n)**
With Optimizations	**Worst case: O(logn)**

Worksheet question 1

```
1:  function Kruskal(Graph G)
2:      initialize each vertex to be a component
3:      sort all edges by weight
4:      for each edge (u, v) in sorted order do
5:          if u and v are in different components then
6:              add edge (u,v) to the MST
7:              update u and v to be in the same component
8:          end if
9:      end for
10: end function
```

Worksheet question 1

```
1:  function Kruskal(Graph G)
2:      initialize a disjoint set; call makeSet() on each vertex
3:      sort all edges by weight
4:      for each edge (u, v) in sorted order do
5:          if findSet(u) ≠ findSet(v) then
6:              add edge (u,v) to the MST
7:              union(u, v)
8:          end if
9:      end for
10: end function
```

Implementation

Use Nodes?

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

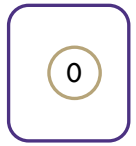
- int field takes 4 bytes
- Pointer takes 8 bytes
- Overhead ~ 16 bytes
- Adds up to 28, but we must partition in multiples of 8 => 32 bytes

Use arrays instead!

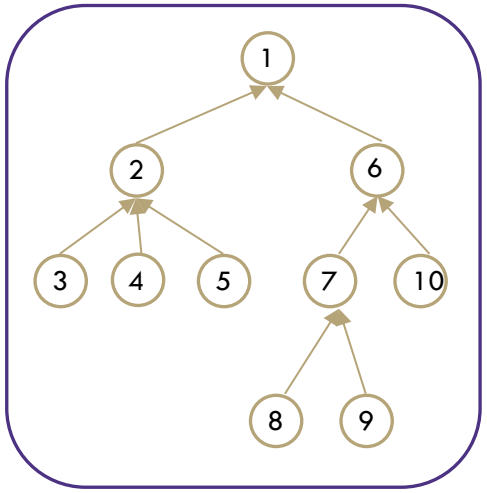
- Make index of the array be the vertex number
  - Either directly to store ints or representationally
  - We implement makeSet(x) so that **we** choose the representative
- Make element in the array the index of the parent

Array implementation

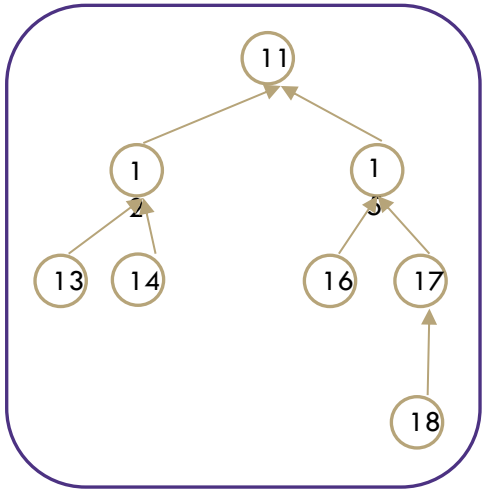
rank = 0

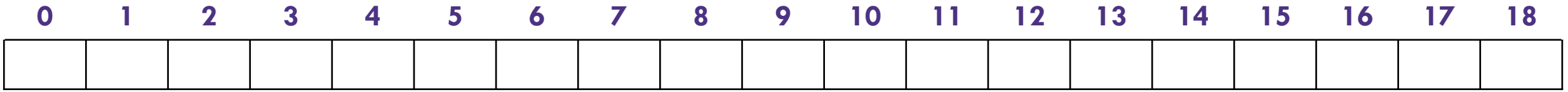


rank = 3



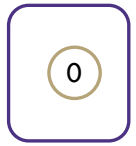
rank = 3



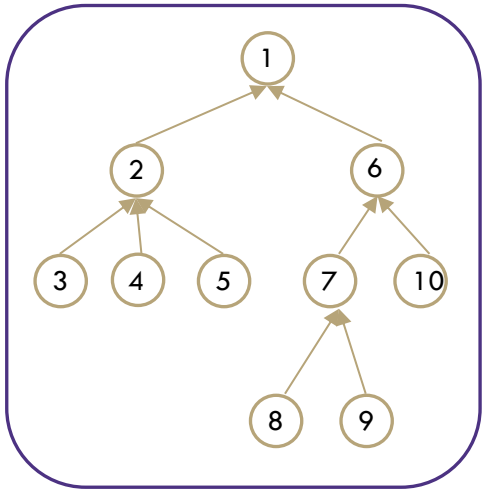


Array implementation

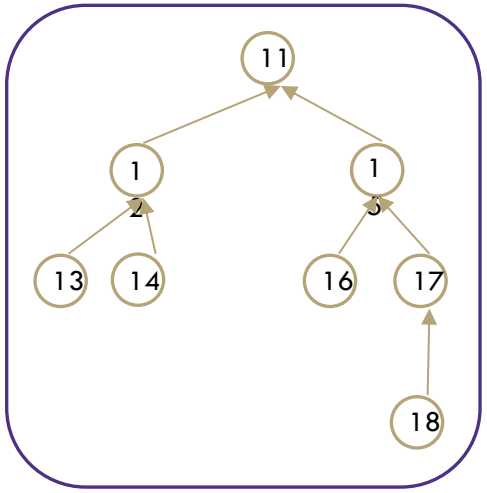
rank = 0



rank = 3



rank = 3



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	6	-4	11	12	12	11	15	15	17

Store (rank * -1) - 1

Worksheet question 2

Consider the following disjoint set. Assume that (from left) the first tree has rank 3, the second has rank 0, the third has rank 1, and the last tree has rank 1.

Write the array representation of this disjoint set in the array below.

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

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, Strongly connected components

Minimum Spanning Trees
- Primm's
- Kruskal's

Disjoint Sets
- Implementing Kruskal's

Next week

Monday: Guest lecture on Technical Interviews

Wednesday: P vs. NP

Thursday (Section): Final review session

Friday: Final review session

Sunday (tentative): TA lead extra review session

Topics covered in final

- Everything we learned in the class

- Final is cumulative with more focus on topics we covered after midterm.
  - So there will be questions on the topics we covered before midterm as well.

- Topics not covered in final
  - B-Trees
  - Java generics and Java interfaces
  - Java Syntax

Advice:

- Make use of exams from previous terms. A practice exam will be posted on Monday.

- Review section handouts for how to write good answers (particularly algorithm design questions)

- Finish HW7 early so you can focus on Final.

Worksheet question 3

Suppose you are given a connected graph G. Describe how you would figure out if the graph has a cycle. (Answer in at most 3-4 sentences.)

Worksheet question 3

Suppose you are given a connected graph G. Describe how you would figure out if the graph has a cycle. (Answer in at most 3-4 sentences.)

Run DFS but keep track of vertices in the stack.

If we hit a vertex that is already in the stack, then there is a cycle in the graph.

(Another solution is using topological sort, which is a similar idea, but it works only for directed graphs.)