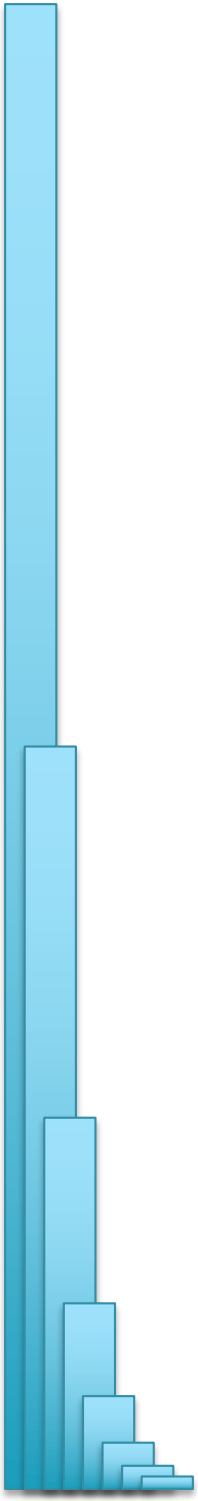
CS 600.226: Data Structures

Michael Schatz

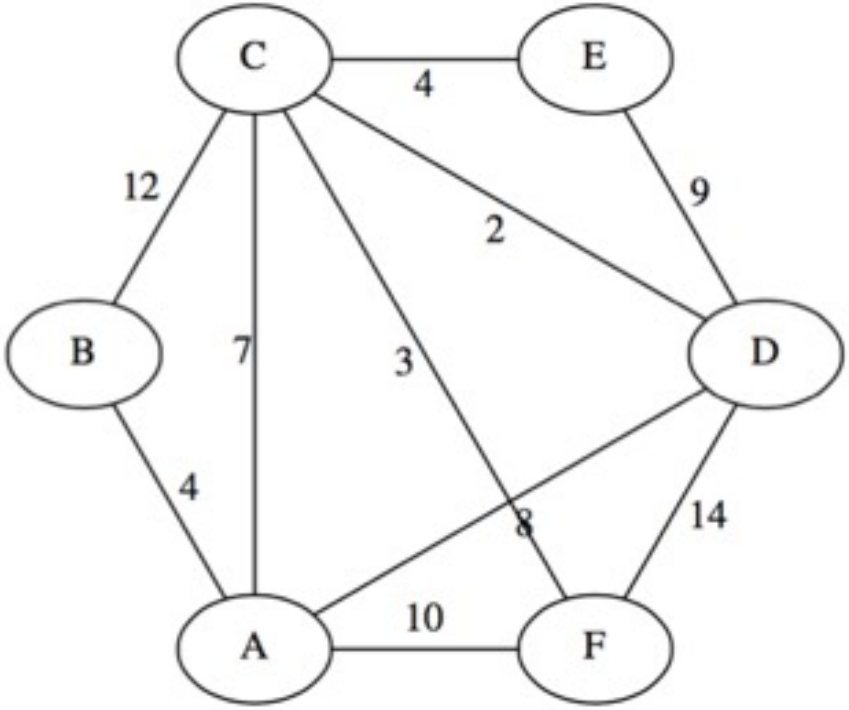
Dec 7, 2016

Lecture 39: Advanced Compiler Optimizations ;-)



Part 1: Minimum Spanning Trees

Long Distance Calling



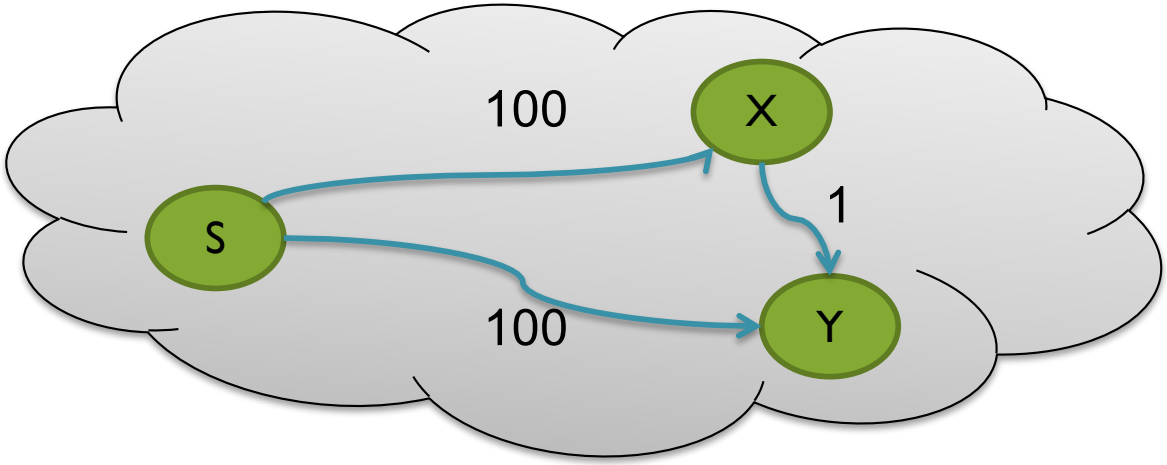
Supposes it costs different amounts of money to send data between cities A through F. Find the least expensive set of connections so that anyone can send data to anyone else.

Given an undirected graph with weights on the edges, find the subset of edges that (a) connects all of the vertices of the graph and (b) has minimum total costs

This subset of edges is called the minimum spanning tree (MST)

Removing an edge from MST disconnects the graph, adding one forms a cycle

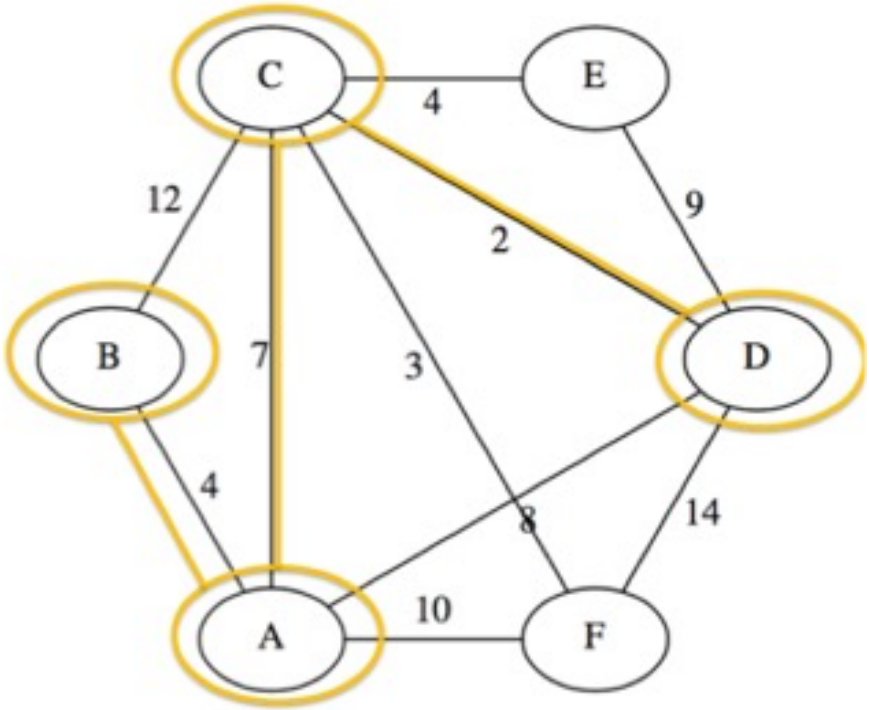
Dijkstra's != MST



Dijkstra's will build the tree S->X, S->Y
(tree visits every node with shortest paths from S to every other node)

but the MST is S->X, X->Y
(tree visits every node and minimizes the sum of the edges)

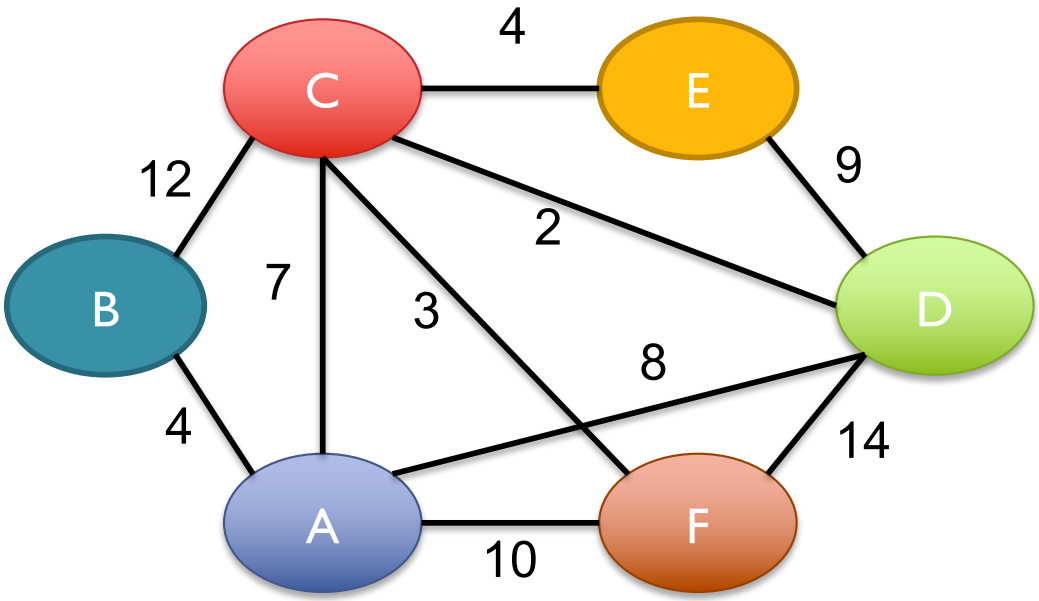
Prim's Algorithm



Prim's Algorithm Sketch

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and add it to the tree.
3. Repeat step 2 (until all vertices are in the tree).

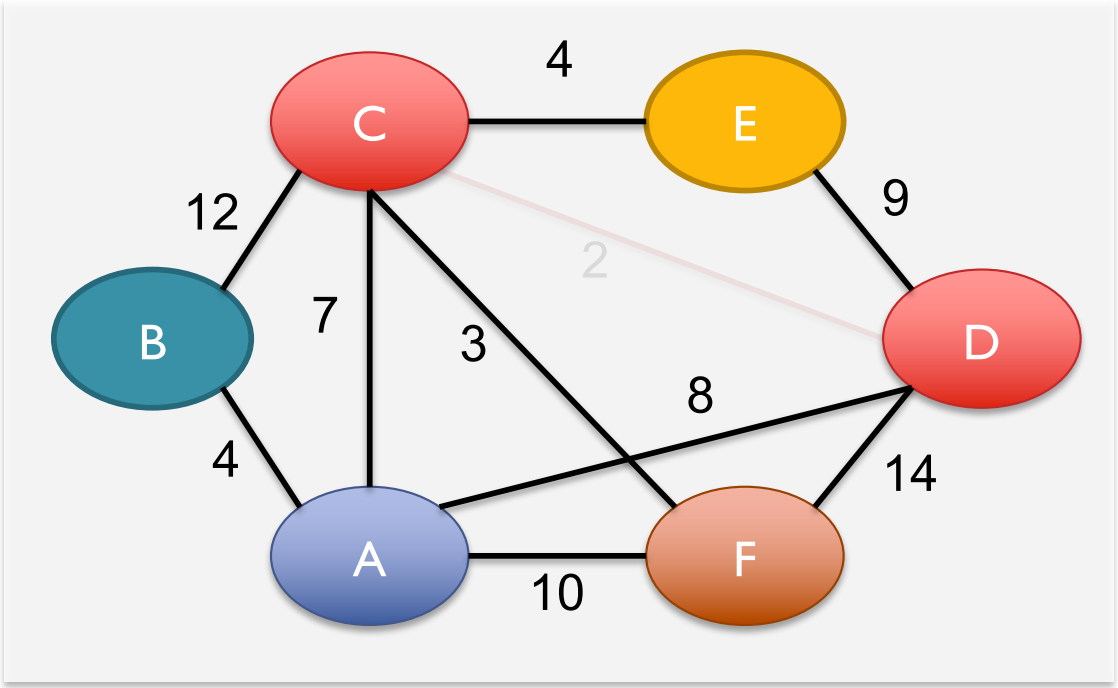
Kruskal's Algorithm



Kruskal's Algorithm Sketch

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

Kruskal's Algorithm

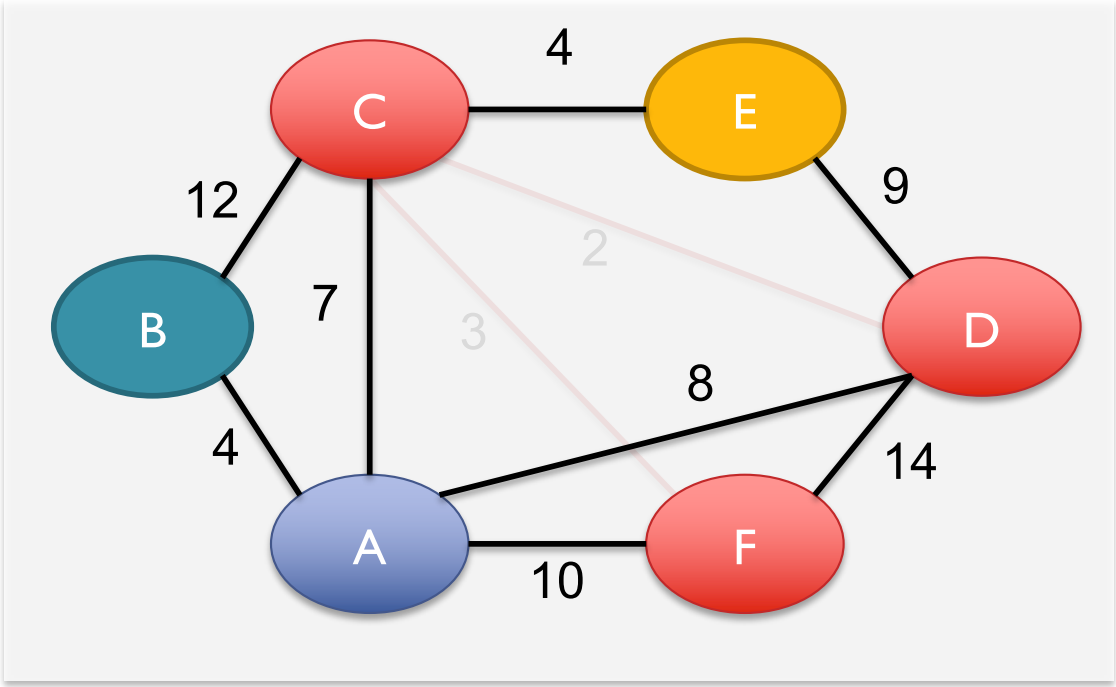


Tinting is just to make it easier to look at

Kruskal's Algorithm Sketch

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

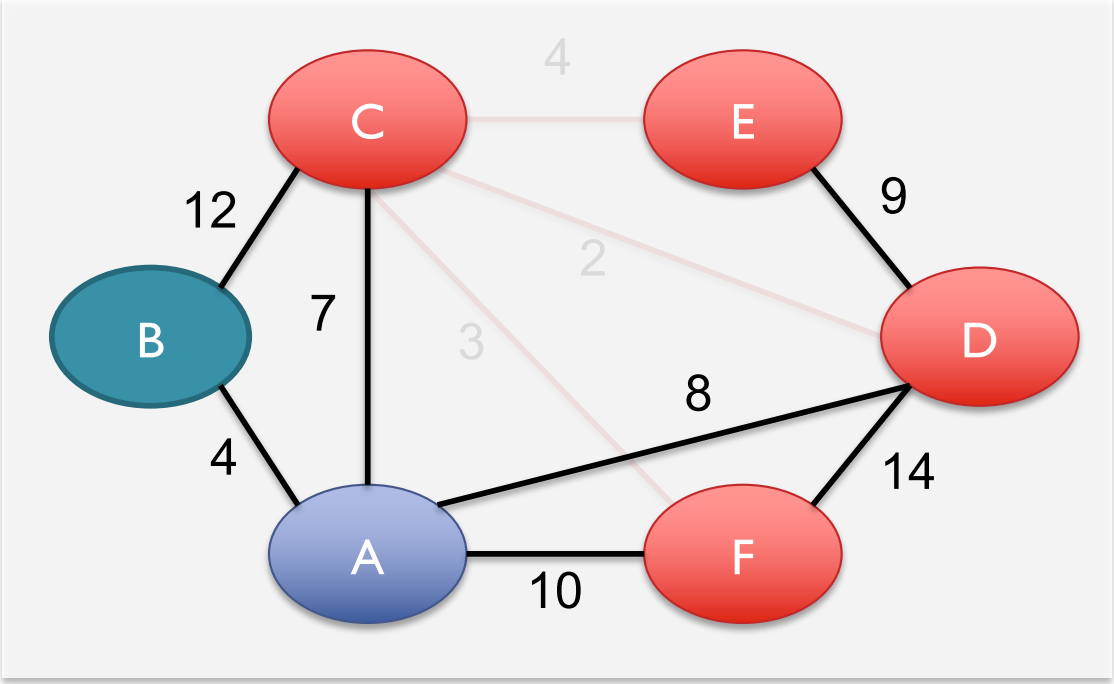
Kruskal's Algorithm



Kruskal's Algorithm Sketch

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

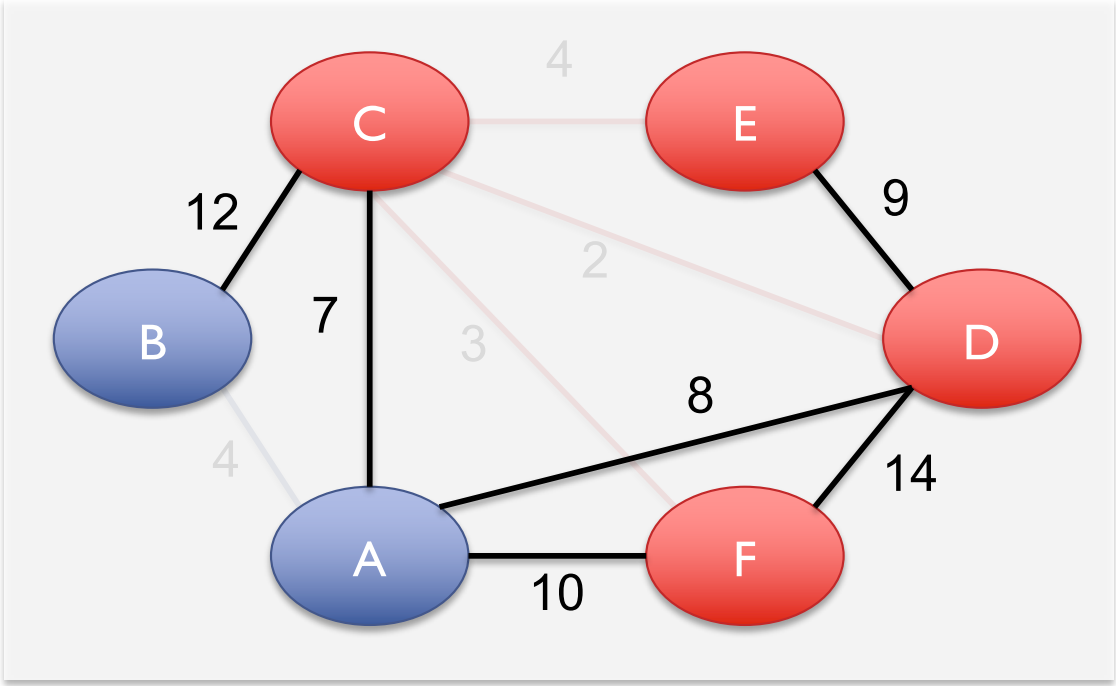
Kruskal's Algorithm



Kruskal's Algorithm Sketch

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

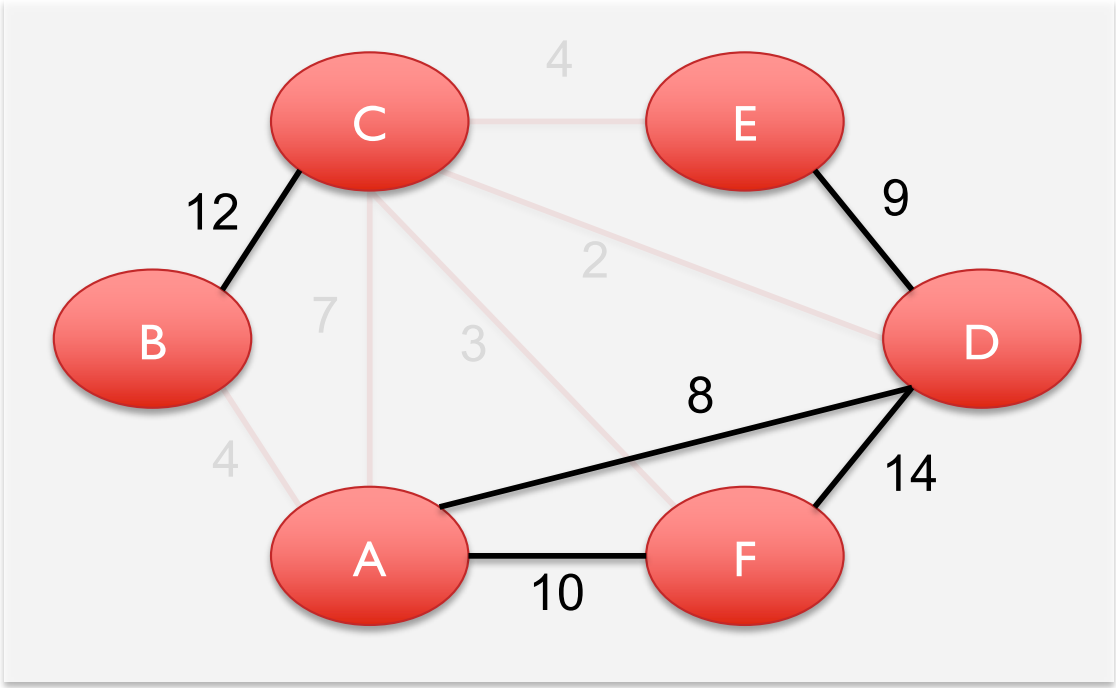
Kruskal's Algorithm



Kruskal's Algorithm Sketch

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

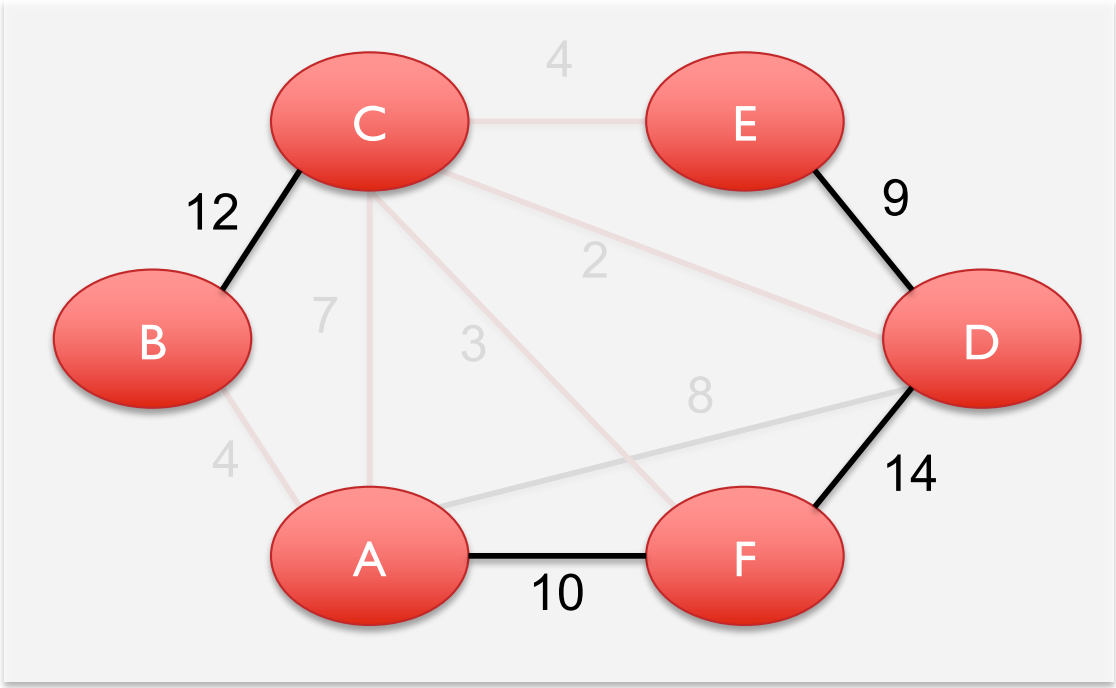
Kruskal's Algorithm



Kruskal's Algorithm Sketch

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

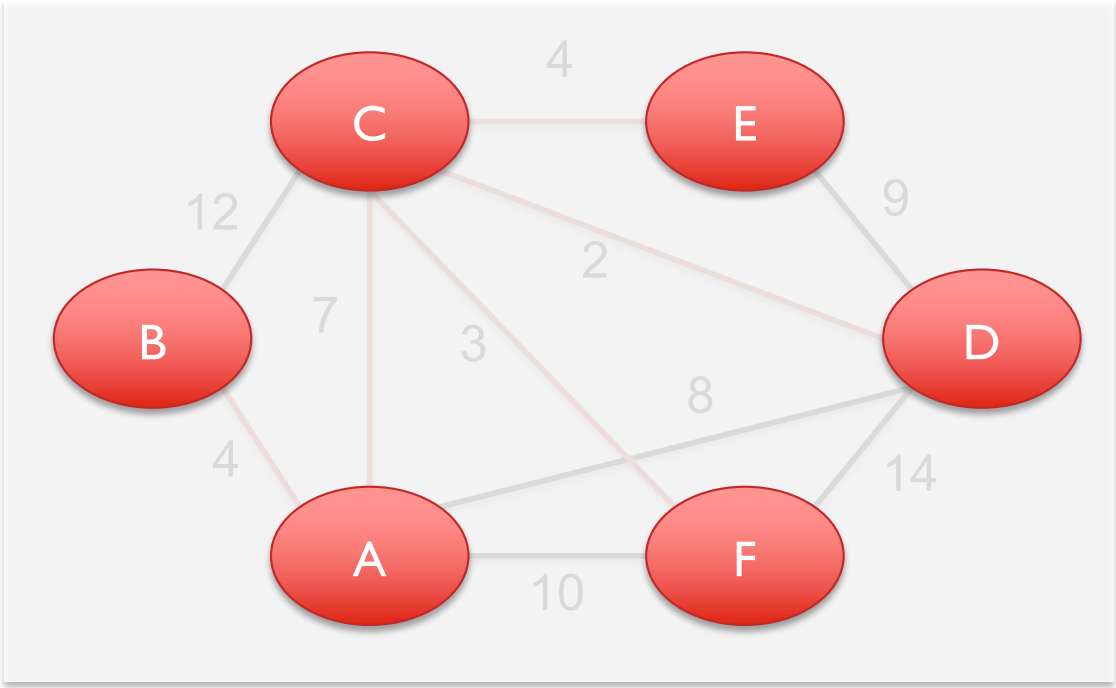
Kruskal's Algorithm



Kruskal's Algorithm Sketch

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

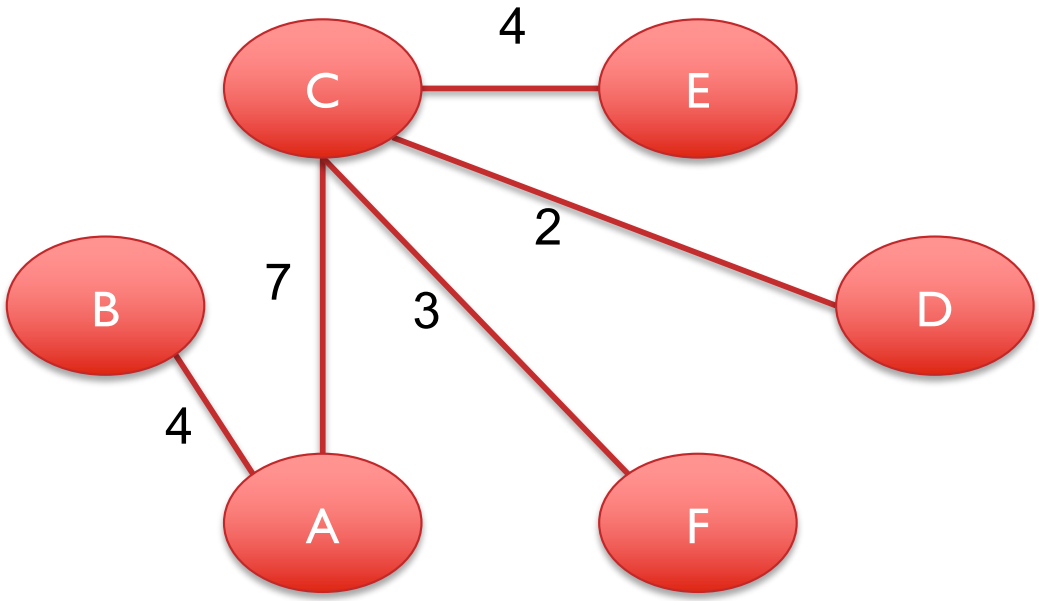
Kruskal's Algorithm



Kruskal's Algorithm Sketch

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

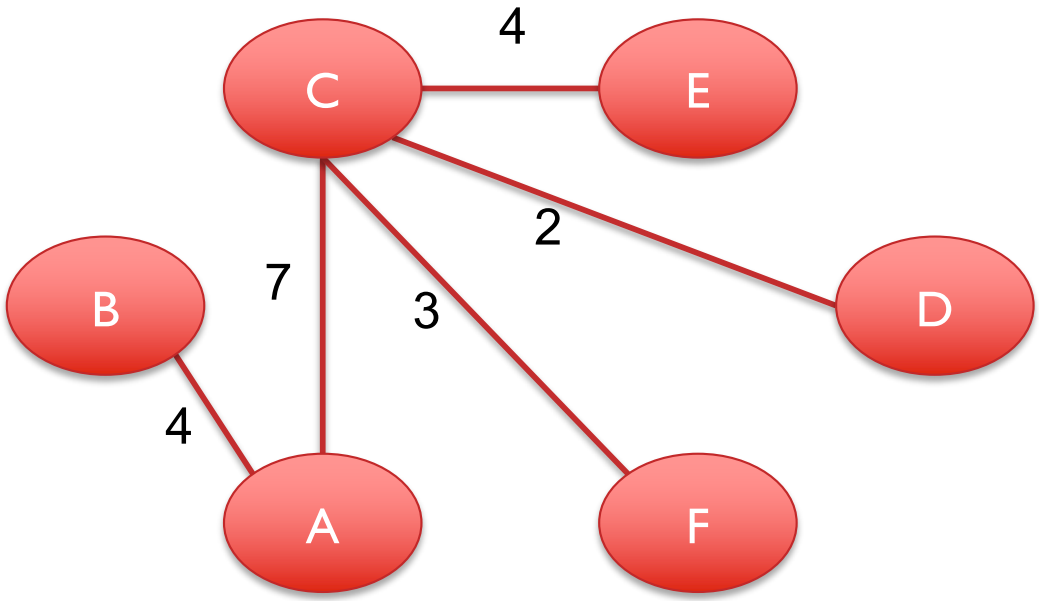
Kruskal's Algorithm



Kruskal's Algorithm Sketch

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

Kruskal's Algorithm



Running time:

Kruskal's Algorithm Sketch

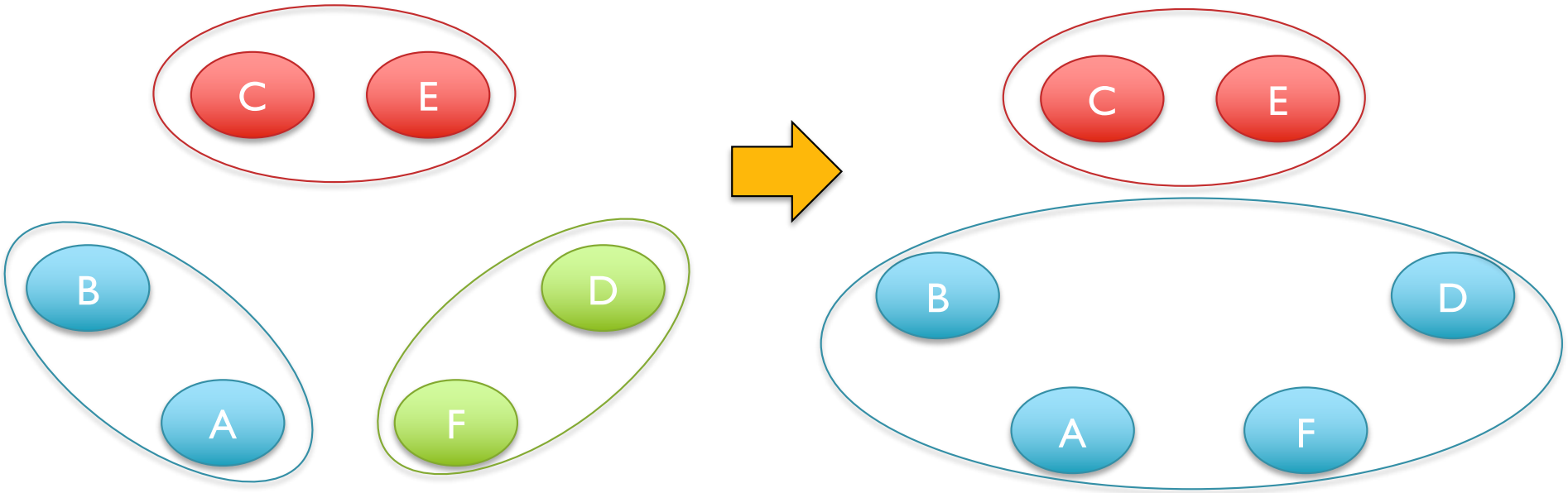
1. Create a forest F (a set of trees), where each vertex in the grap separate tree
2. Create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning
    1. remove an edge with minimum weight from S
    2. if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

Easy: O(|V|)

Easy: O(|E|lg|E|)

Hmm???

Disjoint Sets



Disjoint Subset: every element is assigned to a single subset

Problem: Determine which elements are part of the same subset as sets are dynamically joined together

Two main methods:
Find: Are elements x and y part of the same set?
Union: Merge together sets containing elements x and y

Quick Find

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` are connected if they have the same id.

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	9	9	6	6	7	8	9

5 and 6 are connected
2, 3, 4, and 9 are connected

Find. Check if `p` and `q` have the same id.

id[3] = 9; id[6] = 6
3 and 6 not connected

Union. To merge components containing `p` and `q`, change all entries with `id[p]` to `id[q]`.

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	6	6	6	6	6	7	8	6

problem: many values can change

union of 3 and 6
2, 3, 4, 5, 6, and 9 are connected

https://www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf

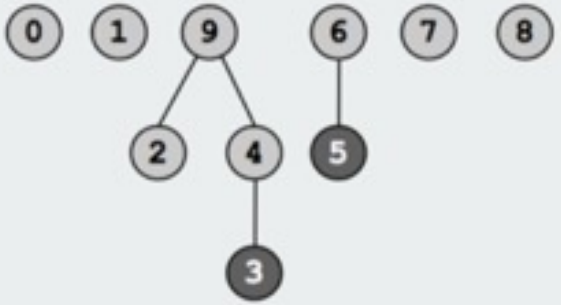
Quick Union

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

keep going until it doesn't change

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	4	9	6	6	7	8	9



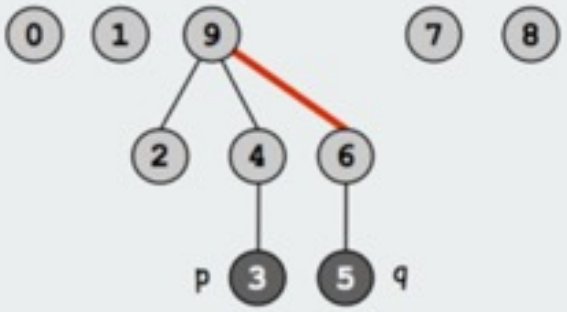
3's root is 9; 5's root is 6

Find. Check if p and q have the same root.

Union. Set the id of q's root to the id of p's root.

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	4	9	6	9	7	8	9

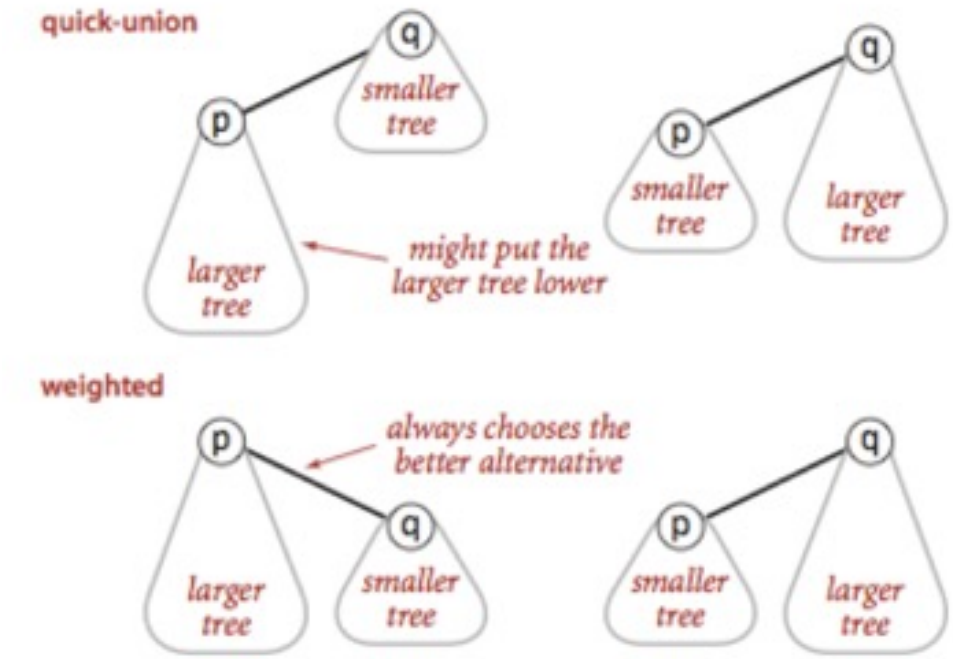
only one value changes



Improved Quick-Union: Weighting

Weighted quick-union.

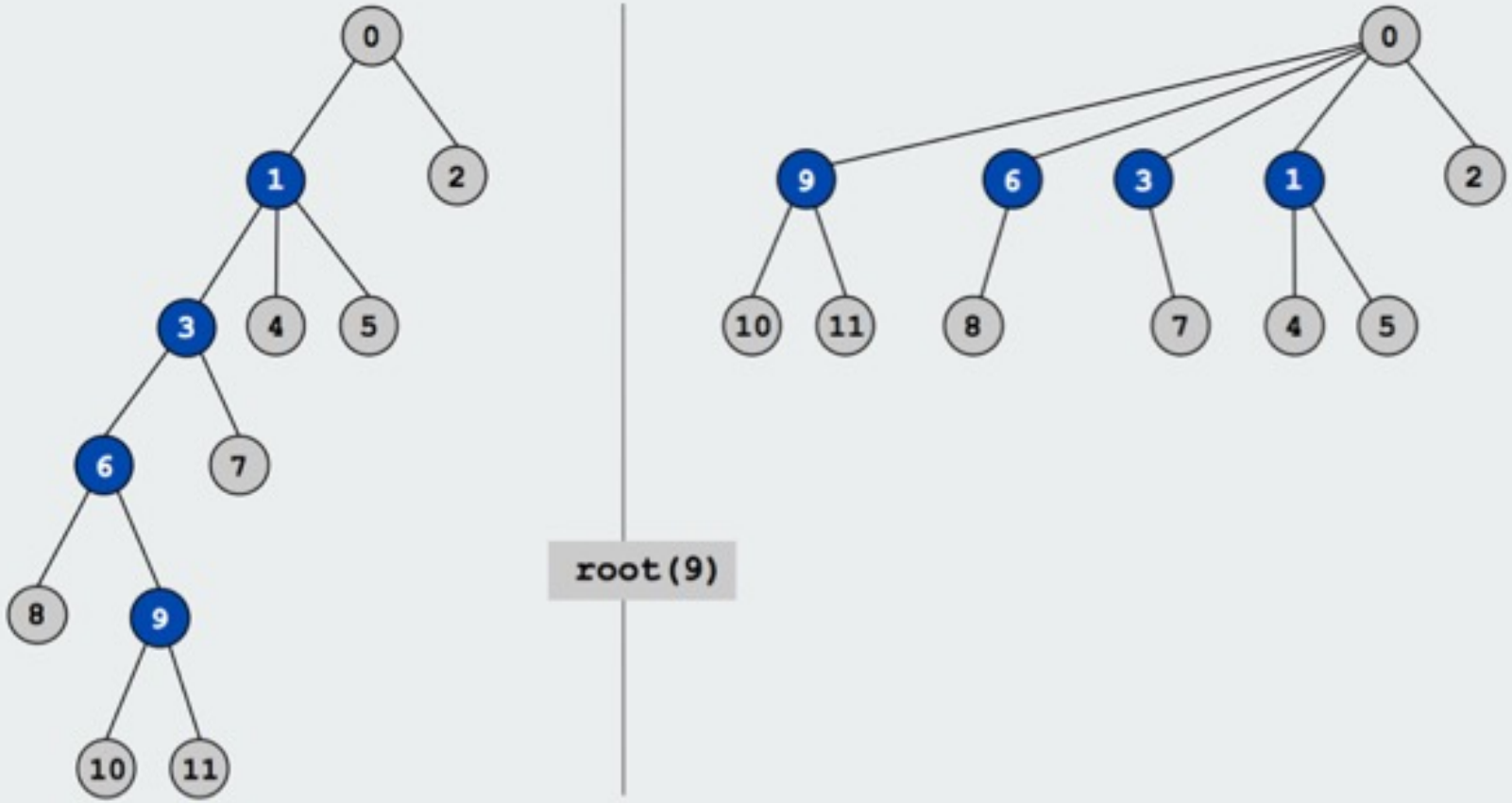
- Modify quick-union to avoid tall trees.
- Keep track of size of each component.
- Balance by linking small tree below large one.



Weighted quick-union

Improvement 2: Path Compression

Path compression. Just after computing the root of `i`, set the `id` of each examined node to `root(i)`.



Weighted Path Compression Analysis

Theorem. Starting from an empty data structure, any sequence of M union and find operations on N objects takes $O(N + M \lg^* N)$ time.

- Proof is very difficult.
- But the algorithm is still simple!

number of times needed to take the lg of a number until reaching 1

Linear algorithm?

- Cost within constant factor of reading in the data.
- In theory, WQUPC is not quite linear.
- In practice, WQUPC is linear.

because lg* N is a constant in this universe

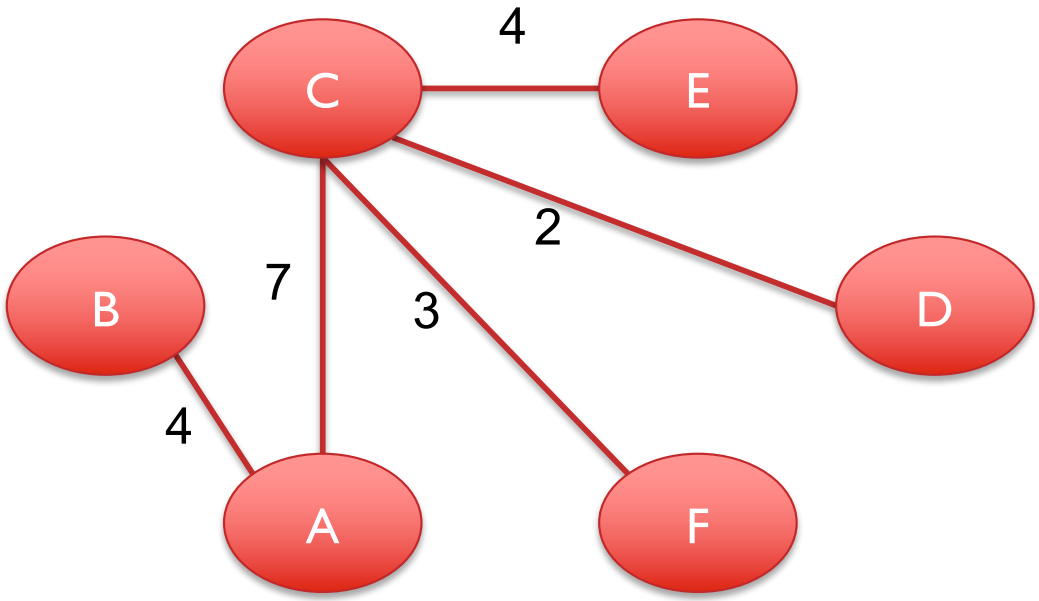
x	$\lg^* x$
$(-\infty, 1]$	0
$(1, 2]$	1
$(2, 4]$	2
$(4, 16]$	3
$(16, 65536]$	4
$(65536, 2^{65536}]$	5

Much bigger than #atoms in the universe

Amazing fact:

- In theory, no linear linking strategy exists

Kruskal's Algorithm



Kruskal's Algorithm Sketch

Using Weighted-Path Compression Quick Union (aka Union-Find) each find or union in O(lg*|V|) time. Total time is O(|E|lg|E| + Elg*|V|) => O(|E|lg|E|)

3. while S is nonempty and F is not yet spanning

If the edges are already in sorted order (or use a linear time sorting algorithm such as counting sort), reduce total time to O(|E|lg*|V|)

in the graph is a

s then add it to the

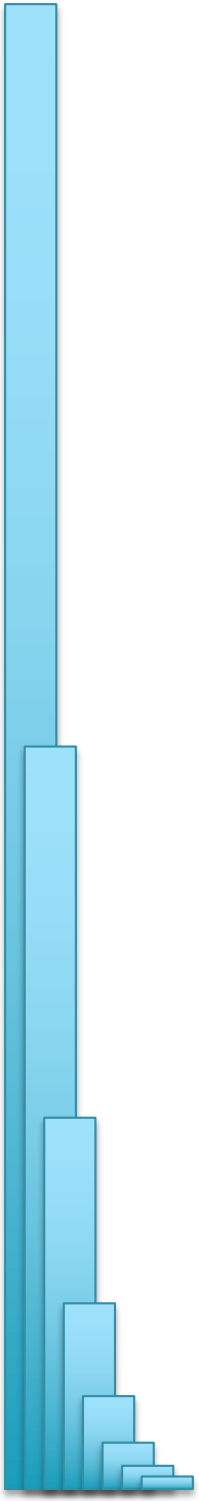
Running time:

Easy: O(|V|)

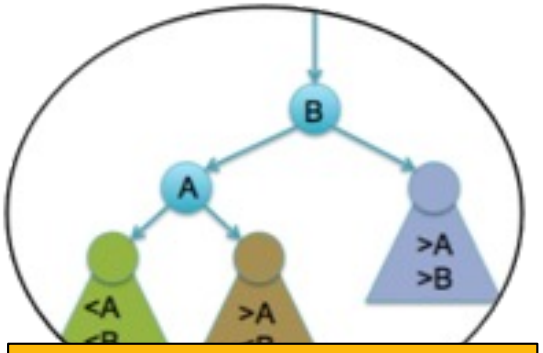
Easy: O(|E|lg|E|)

Hmm???

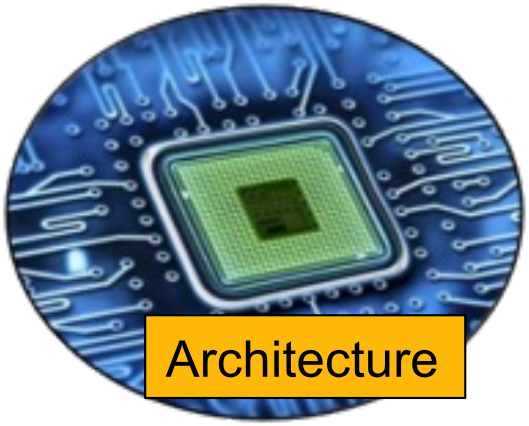
Never underestimate the power of the LOG STAR!!!!



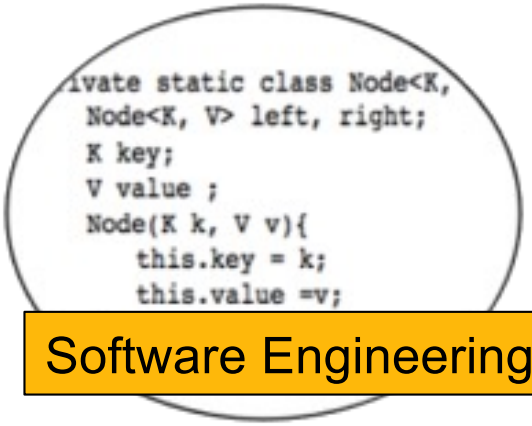
Part 2: Final Thoughts



Abstract Data Types

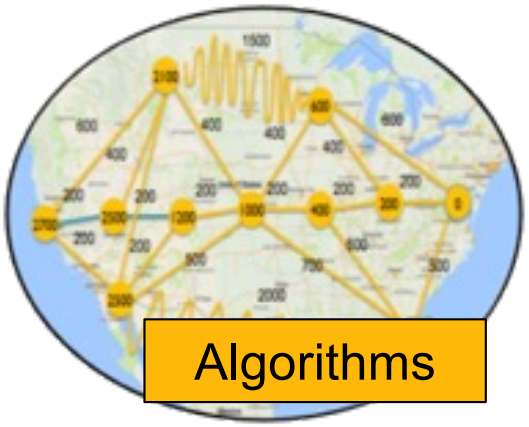


Architecture

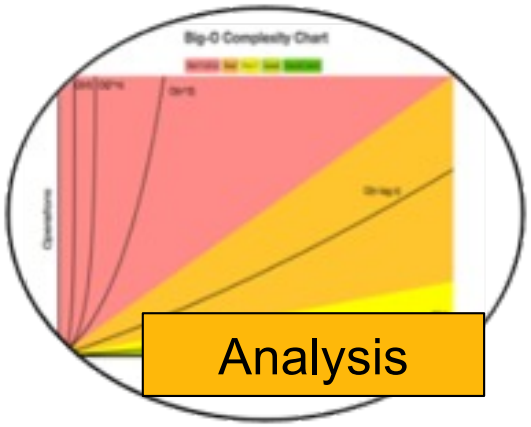


Software Engineering

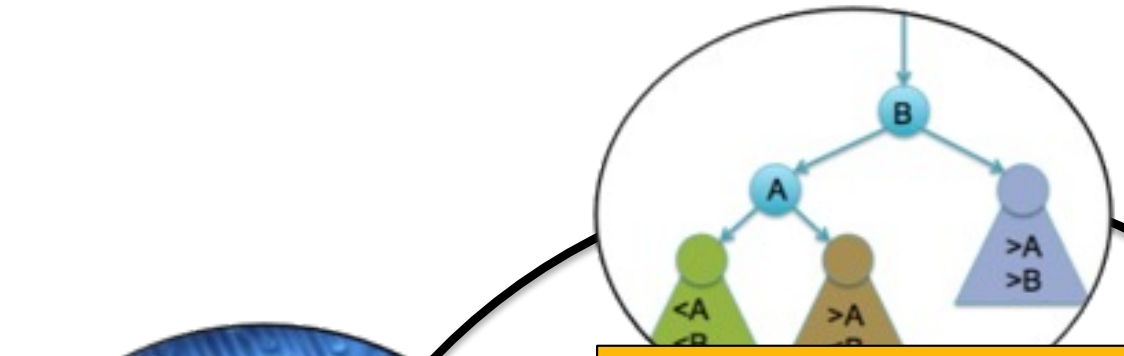
CS 600.226



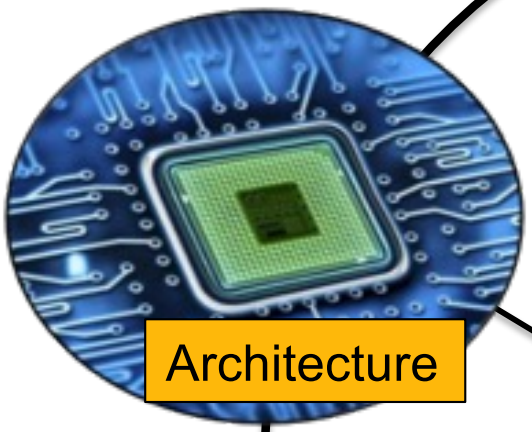
Algorithms



Analysis



Abstract Data Types

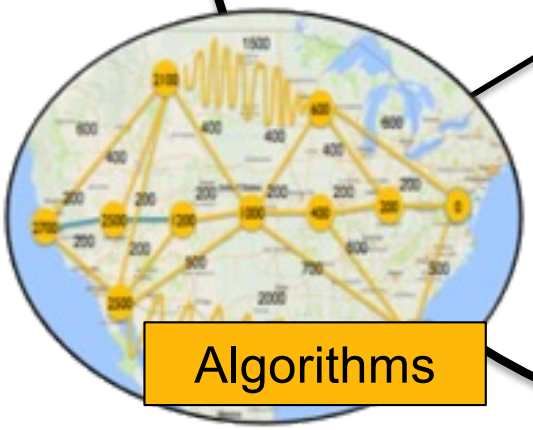


Architecture

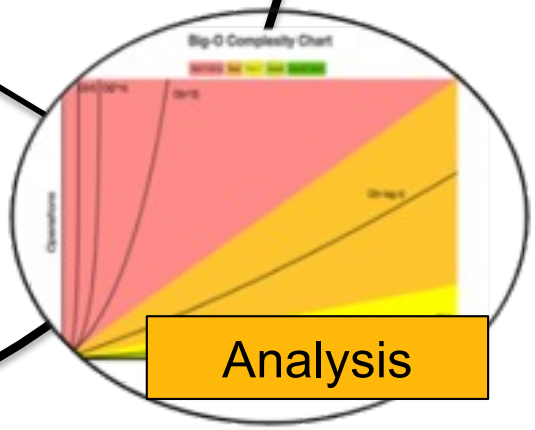
```
ivate static class Node<K,
 Node<K, V> left, right;
 K key;
 V value ;
 Node(K k, V v){
    this.key = k;
    this.value =v;
```

Software Engineering

CS 600.226

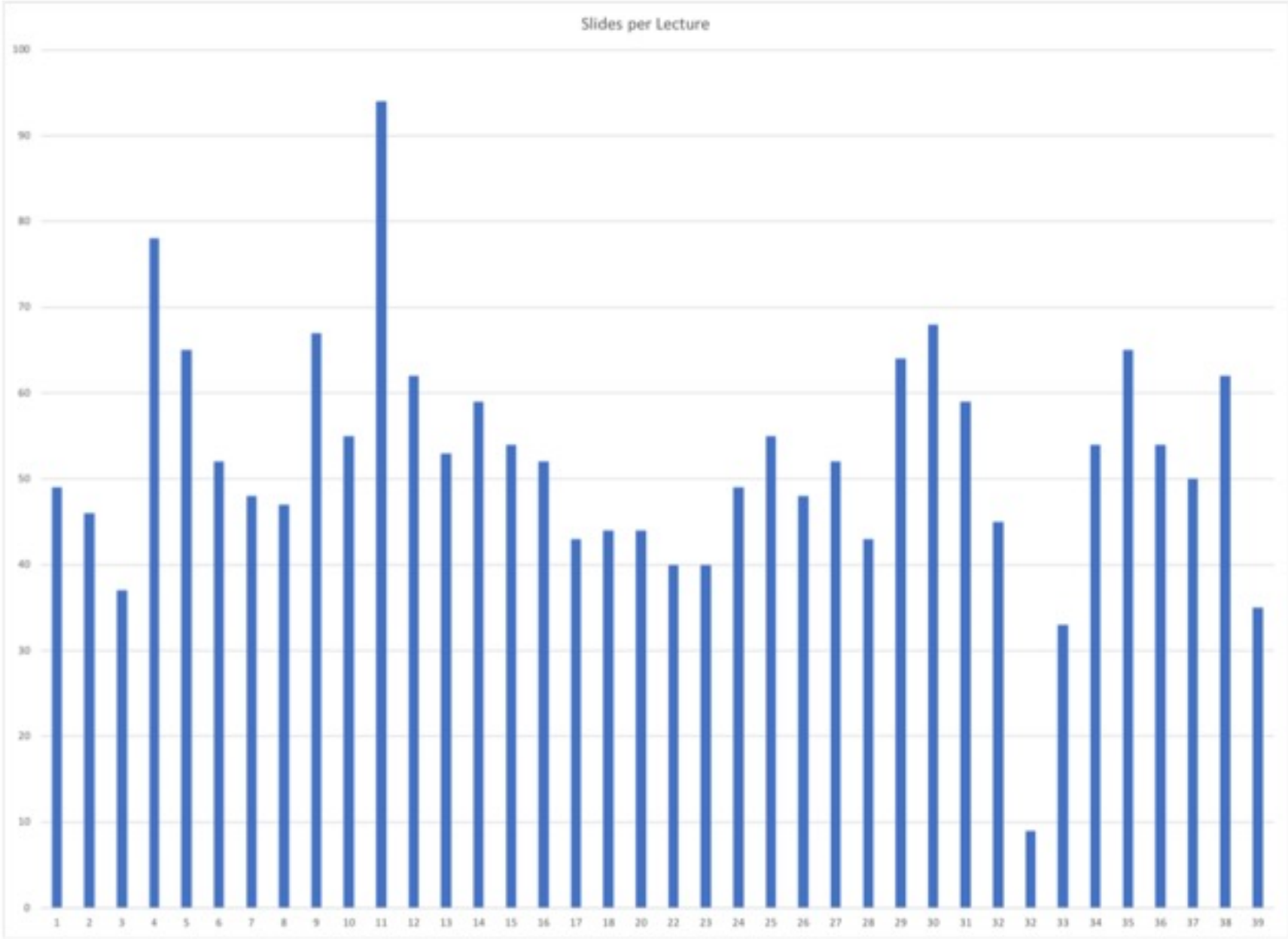


Algorithms



Analysis

Putting it all together



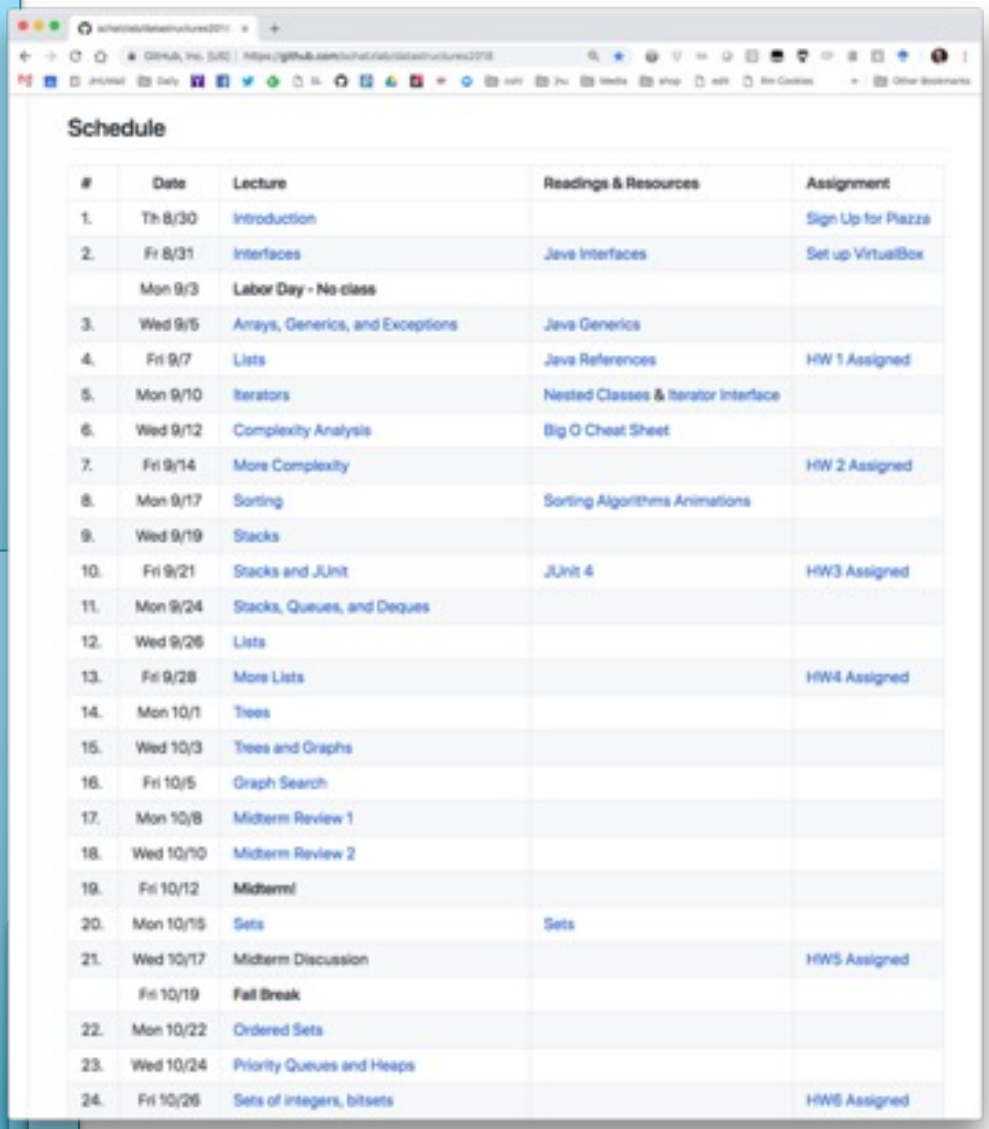
mean = 52.4 +/- 13.8 sum = 1974

Next Steps

Review Session: Tuesday Dec 11 @ 2pm in Malone G33/G35
Office Hours: Wednesday Dec 14 @ 3pm & by appointment

Final Exam: Thursday Dec 13 @ 9am Right here!

Resources



Schedule

#	Date	Lecture	Readings & Resources	Assignment
1.	Th 8/30	Introduction		Sign Up for Piazza
2.	Fr 8/31	Interfaces	Java Interfaces	Set up VirtualBox
	Mon 9/3	**Labor Day - No class**		
3.	Wed 9/5	Arrays, Generics, and Exceptions	Java Generics	
4.	Fri 9/7	Lists	Java References	HW 1 Assigned
5.	Mon 9/10	Iterators	Nested Classes & Iterator Interface	
6.	Wed 9/12	Complexity Analysis	Big O Cheat Sheet	
7.	Fri 9/14	More Complexity		HW 2 Assigned
8.	Mon 9/17	Sorting	Sorting Algorithms Animations	
9.	Wed 9/19	Stacks		
10.	Fri 9/21	Stacks and JUnit	JUnit 4	HW3 Assigned
11.	Mon 9/24	Stacks, Queues, and Deques		
12.	Wed 9/26	Lists		
13.	Fri 9/28	More Lists		HW4 Assigned
14.	Mon 10/1	Trees		
15.	Wed 10/3	Trees and Graphs		
16.	Fri 10/5	Graph Search		
17.	Mon 10/8	Midterm Review 1		
18.	Wed 10/10	Midterm Review 2		
19.	Fri 10/12	**Midterm!**		
20.	Mon 10/15	Sets	Sets	
21.	Wed 10/17	Midterm Discussion		HW5 Assigned
	Fri 10/19	**Fall Break**		
22.	Mon 10/22	Ordered Sets		
23.	Wed 10/24	Priority Queues and Heaps		
24.	Fri 10/26	Sets of integers, bitsets		HW6 Assigned

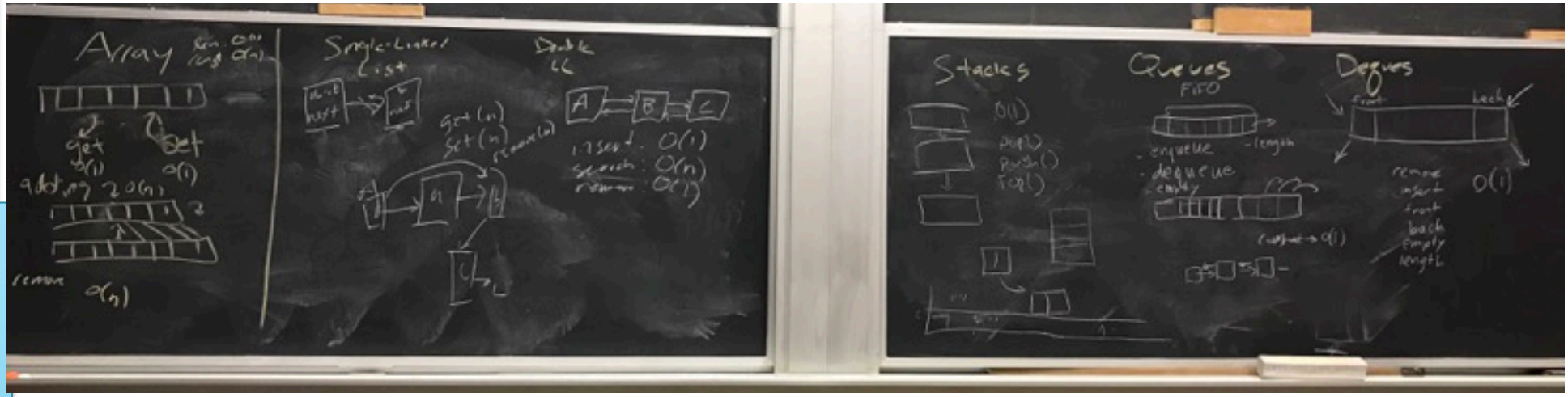
Algorithms and Data Structures
Lecture Notes for 601.226

Peter H. Fröhlich
phf@cs.jhu.edu

August 29, 2018

DRAFT

Midterm Review



*You are not responsible for knowing the k-d tree


DEWALT
EXPAND YOUR 18 VOLT SYSTEM
WITH BARE TOOLS FROM DEWALT
Makita
HAWKINGS
12V WEIGHT.


Become a TA!
Do Research!
Start a company!
EXPAND YOUR SYSTEM
18 VOLT
WITH BARE TOOLS FROM DEWALT
DEWALT

Thank you!

Final Review

Trees

- Array vs Node Implementation
- In-, pre-, post- order
- Binary Search Trees
- AVL Trees
- Heaps & Priority Queues
- Treaps

Graphs

- Adj. List, Adj. Matrix, Edge List
- DFS & BFS
- Shortest Path: Dijkstra's
- Topological Sorting
- MST: Prim's & Kruskal's

Sorting

- Binary Search
- HeapSort, MergeSort, QuickSort
- CountingSort
- O(n lg n) vs O(n) sorting

Sets & Maps

- Position-based vs Value-based
- Sparse Array
- Array Set vs List Set
- Self organizing
- Bit sets
- Bloom filter
- Union Find

Hash Tables

- Separate Chaining
- Linear/Quadratic/Double probing
- Cuckoo Hashing
- Hash Functions
- Hash Table Sizes

Strings

- Suffix Arrays
- BWT
- Run Length Encoding


Welcome to CS 600.226
https://github.com/schatzlab/datastructures2018

Good luck!