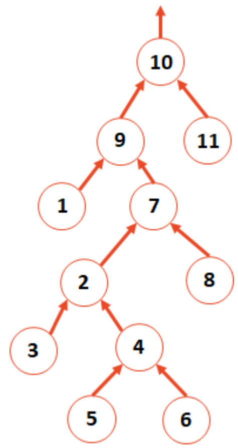
#32: Disjoint Sets Finale + Graphs Intro

April 5, 2019 · *Fagen-Ulmschneider, Zilles*

Smart Union Options:

- Union by Height (root := **-h - 1**)
- Union by Size (root := **-n**)
- Union by Rank (root := #union ops)



In <u>all</u> smart unions:

....height of UpTree: _____________.

How do we improve this?

DisjointSets.cpp (partial)

```
int DisjointSets::find(int i) {
  if ( arr_[i] < 0 ) { return i; }
  else { return _find( arr_[i] ); }
}
```

DisjointSets.cpp (partial)

```
void DisjointSets::unionBySize(int root1, int root2) {
  int newSize = arr_[root1] + arr_[root2];

  // If arr_[root1] is less than (more negative), it is the
  // larger set; we union the smaller set, root2, with root1.
  if ( arr_[root1] < arr_[root2] ) {
    arr_[root2] = root1;
    arr_[root1] = newSize;
  }
  // Otherwise, do the opposite:
  else {
    arr_[root1] = root2;
    arr_[root2] = newSize;
  }
}
```

Running Time:

- Worst case running time of find(k)**:**
- Worst case running time of union(r1, r2), given roots:
- New function: "Iterated Log":

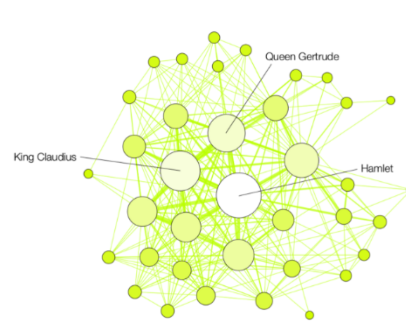
  log*(n) :=

- Overall running time:
  - A total of **m** union/find operation runs in:

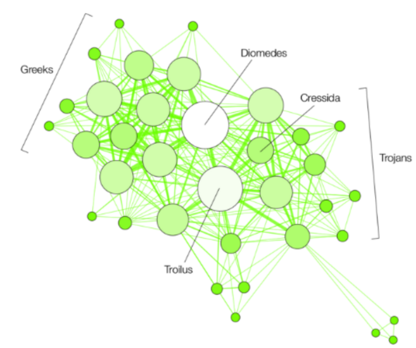
A Review of Major Data Structures so Far

Array-based	List/Pointer-based
- Sorted Array	- Singly Linked List
- Unsorted Array	- Doubly Linked List
- Stacks	- Trees
- Queues	- BTree
- Hashing	- Binary Tree
- Heaps	- Huffman Encoding
- Priority Queues	- kd-Tree
- UpTrees	- AVL Tree
- Disjoint Sets	

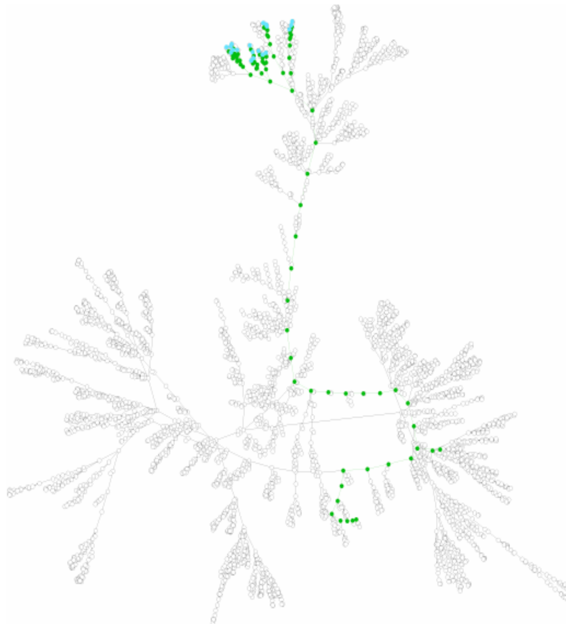
An Introduction to Graphs

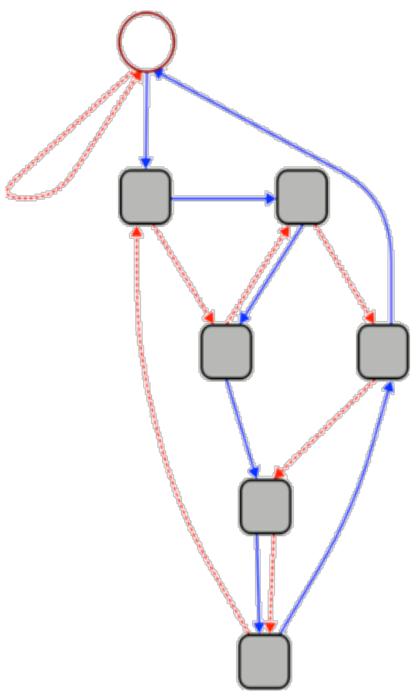


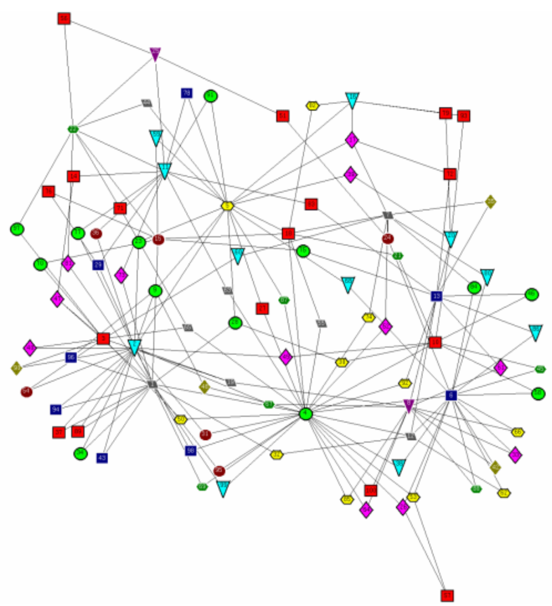
HAMLET



TROILUS AND CRESSIDA



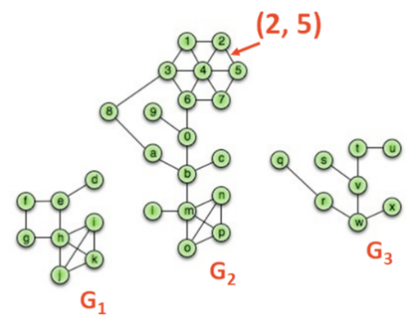




Motivation:
Graphs are awesome data structures that allow us to represent an enormous range of problems. To study these problems, we need:

1. A common vocabulary to talk about graphs
2. Implementation(s) of a graph
3. Traversals on graphs
4. Algorithms on graphs

Graph Vocabulary
Consider a graph **G** with vertices **V** and edges **E**, **G=(V,E)**.



Incident Edges:
I(v) = { (x, v) in E }

Degree(v): **|I|**

Adjacent Vertices:
A(v) = { x : (x, v) in E }

Path(G_2): Sequence of vertices connected by edges

Cycle(G_1): Path with a common begin and end vertex.

Simple Graph(G): A graph with no self loops or multi-edges.

Subgraph(G): **G' = (V', E')**:
V' ∈ V, E' ∈ E, and (u, v) ∈ E → u ∈ V', v ∈ V'

CS 225 – Things To Be Doing:

1. Theory Exam 3 is ongoing!
2. lab_heap due Sunday, April 7th
3. MP6 released; Extra Credit +7 deadline April 8th
4. Daily POTDs are ongoing!