CS 225
Data Structures

April 9 – Graphs Intro
Wade Fagen-Ulmschneider
Disjoint Sets Analysis

The *iterated log* function:

*The number of times you can take a log of a number.*

\[
\log^*(n) =
\begin{align*}
0 & , \quad n \leq 1 \\
1 + \log^*(\log(n)) & , \quad n > 1
\end{align*}
\]

What is \( \log^*(2^{65536}) \)?
Disjoint Sets Analysis

In an Disjoint Sets implemented with smart unions and path compression on find:

Any sequence of \( m \) union and find operations result in the worse case running time of \( O( \_________________ ) \), where \( n \) is the number of items in the Disjoint Sets.
In Review: Data Structures

<table>
<thead>
<tr>
<th>Array</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Sorted Array</td>
<td>- Doubly Linked List</td>
</tr>
<tr>
<td>- Unsorted Array</td>
<td>- Skip List</td>
</tr>
<tr>
<td>- Stacks</td>
<td>- Trees</td>
</tr>
<tr>
<td>- Queues</td>
<td>- BTree</td>
</tr>
<tr>
<td>- Hashing</td>
<td>- Binary Tree</td>
</tr>
<tr>
<td>- Heaps</td>
<td>- Huffman Encoding</td>
</tr>
<tr>
<td>- Priority Queues</td>
<td>- kd-Tree</td>
</tr>
<tr>
<td>- UpTrees</td>
<td>- AVL Tree</td>
</tr>
<tr>
<td>- Disjoint Sets</td>
<td></td>
</tr>
</tbody>
</table>
• Constant time access to any element, given an index $a[k]$ is accessed in $O(1)$ time, no matter how large the array grows

• Cache-optimized
  Many modern systems cache or pre-fetch nearby memory values due the “Principle of Locality”. Therefore, arrays often perform faster than lists in identical operations.
• **Efficient general search structure**
  Searches on the sort property run in $O(\lg(n))$ with Binary Search

• **Inefficient insert/remove**
  Elements must be inserted and removed at the location dictated by the sort property, resulting shifting the array in memory – an $O(n)$ operation
• Constant time add/remove at the beginning/end
  Amortized O(1) insert and remove from the front and of the array
  Idea: Double on resize

• Inefficient global search structure
  With no sort property, all searches must iterate the entire array; O(1) time
• First In First Out (FIFO) ordering of data
  Maintains an arrival ordering of tasks, jobs, or data

• All ADT operations are constant time operations
  enqueue() and dequeue() both run in $O(1)$ time
• Last In First Out (LIFO) ordering of data
  Maintains a “most recently added” list of data

• All ADT operations are constant time operations
  push() and pop() both run in O(1) time
In Review: Data Structures

Array
- Sorted Array
- Unsorted Array
  - Stacks
  - Queues
- Hashing
- Heaps
  - Priority Queues
- UpTrees
  - Disjoint Sets

List
- Doubly Linked List
- Skip List
- Trees
  - BTree
  - Binary Tree
    - Huffman Encoding
    - kd-Tree
    - AVL Tree
In Review: Data Structures

Array
- Sorted Array
- Unsorted Array
  - Stacks
  - Queues
- Hashing
- Heaps
  - Priority Queues
- UpTrees
  - Disjoint Sets

List
- Doubly Linked List
- Skip List
- Trees
  - BTree
  - Binary Tree
    - Huffman Encoding
    - kd-Tree
    - AVL Tree

Graphs
“When you're asked about kd-trees in an interview and Wade comes to mind:”
The Internet, 2003
The OPTE Project (2003)
Map of the entire internet; nodes are routers; edges are connections.
Who’s the real main character in Shakespearean tragedies?

*Martin Grandjean (2016)*

“Rush Hour” Solution
Unknown Source
Presented by Cinda Heeren, 2016
Wolfram|Alpha's "Personal Analytics" for Facebook
Generated: April 2013 using Wade Fagen-Ulmschneider’s Profile Data
This graph can be used to quickly calculate whether a given number is divisible by 7.

1. Start at the circle node at the top.
2. For each digit $d$ in the given number, follow $d$ blue (solid) edges in succession. As you move from one digit to the next, follow 1 red (dashed) edge.
3. If you end up back at the circle node, your number is divisible by 7.

3703
Conflict-Free Final Exam Scheduling Graph

Unknown Source
Presented by Cinda Heeren, 2016
Class Hierarchy At University of Illinois Urbana-Champaign
A. Mori, W. Fagen-Ulmschneider, C. Heeren

Graph of every course at UIUC; nodes are courses, edges are prerequisites

http://waf.cs.illinois.edu/discovery/class_hierarchy_at_illinois/
MP Collaborations in CS 225

Unknown Source

Presented by Cinda Heeren, 2016
Graphs

To study all of these structures:
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms
Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$

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.
Graph Vocabulary

$G = (V, E)$
$|V| = n$
$|E| = m$

Subgraph(G):
$G' = (V', E')$:
$V' \in V, E' \in E$, and
$(u, v) \in E \rightarrow u \in V', v \in V'$

Complete subgraph(G)
Connected subgraph(G)
Connected component(G)
Acyclic subgraph(G)
Spanning tree(G)
Running times are often reported by $n$, the number of vertices, but often depend on $m$, the number of edges.

How many edges? **Minimum edges:**

Not Connected:

Connected*:

**Maximum edges:**

Simple:

Not simple:

$$\sum_{v \in V} \deg(v) =$$