CSCI 241

Lecture 19
A3 Overview, Intro to Graphs
Announcements

- A3 has phases!
- Phase 1 will be released today or tomorrow.
- Phases 2 and 3 should be out by Tuesday.
Goals

- Understand the architecture of A3

- Know the definition of a graph and basic associated terminology:
  - Node/vertex; edge/arc; directed, undirected; adjacent; (in/out-)degree; path; cycle;
Goals (next time)

• Understand how to represent graphs and implement basic graph operations using:
  • Adjacency list
  • Adjacency matrix
A3 has 4 phases.
A3 has 4 phases.

It may sound scary.
A3 has 4 phases.

It may sound scary.

It isn’t so bad:

total lines of code is probably $\leq A2$

nothing here is as tricky as AVL rebalance

I’m giving you the unit tests.
A3 has 4 phases.

0. Write an ArrayList clone
A3 has 4 phases.

0. Write an ArrayList clone
   (done!)
A3 has 4 phases.

0. Write an ArrayList clone
   (done by Sunday!)
A3 has 3 phases.

1. Write a min-heap to implement a priority queue with operations:
   • boolean add(V value, P priority)
   • V peek();
   • V poll();

2. Write a hash table.

3. Use the hash table to augment the heap, making the following operations efficient:
   • boolean contains(V v);
   • void changePriority(V v, P newP);
Phase 3 - Hash your Heap

In Phase 1 Heap:

- `contains` requires searching the whole tree.
- `changePriority` requires searching the whole tree, then bubbling down or up.

```
4
/   \
6---5
|   |
21--8
|   |
22--38--55--10
    |       |
    20     19
```
Phase 3 - Hash your Heap

In Phase 3 Heap:

- Each node is stored in the heap and in a HashTable that tracks its index in the heap.

HashTable<$V$, Integer>:

<table>
<thead>
<tr>
<th>value</th>
<th>i (index in heap)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>8</td>
</tr>
<tr>
<td>35</td>
<td>6</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Heap: [4 6 5 21 8 14 35 22 38 55 10 20 19]
Phase 3 - Hash your Heap

In Phase 3 Heap:

boolean contains(V v):

true iff map contains key v

HashTable<V, Integer>:

<table>
<thead>
<tr>
<th>value</th>
<th>i (index in heap)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>8</td>
</tr>
<tr>
<td>35</td>
<td>6</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>19</td>
<td>12</td>
</tr>
</tbody>
</table>

Heap: [4 6 5 21 8 14 35 22 38 55 10 20 19]
Phase 3 - Hash your Heap

In Phase 3 Heap:

```java
void changePriority(V v, P newP):
    i = map.get(v);
    change priority of heap entry
    bubble it up or down
```

HashTable<V, Integer>:

<table>
<thead>
<tr>
<th>value</th>
<th>i (index in heap)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>8</td>
</tr>
</tbody>
</table>

Heap: [ 4 6 5 21 8 14 35 22 38 55 10 20 19 ]
Questions?
CSCI 241
Lecture 19
A3 Overview, Intro to Graphs
THESE AREN'T THE GRAPHS YOU'RE LOOKING FOR
Graph: a bunch of points connected by lines. The lines may have directions, or not.
This is a graph:

The internet's undersea world

The vast majority of the world's communications are not carried by satelite but an altogether older technology: cables under the earth's oceans. As a ship accidentally wipes out Aaxis's access, this map shows how we rely on collections of wires of less than 1cm diameter to link us all together.

Fibre-optic submarine cable systems

In service

Planned

Damaged

Cables with transmission capacity plus future planned capacity are shown in red.

Alexandria, Wednesday
A ship's anchor accidentally cut these cables, leaving the the USA and Europe without 20% of internet capacity.

Astra has an even worse problem: its 20 million users are growing by 80%.
The edges are made of these:
Social Networks

(before they were cool)

Republic of Letters
1650 to 1785 (136 years)

Locke’s (blue) and Voltaire’s (yellow) correspondence. Only letters for which complete location information is available are shown. Data courtesy the Electronic Enlightenment Project, University of Oxford.
Social Networks
(before they were cool)
The USA as a graph:

- Neighboring states are connected by edges.
Electrical circuit
A bigger electrical circuit
This is not a graph:

it is a cat.
This is a graph that can recognize cats.
Graphs: Abstract View

K_5

K_{3,3}
Graphs, Formally

- A directed graph (digraph) is a pair $(V, E)$ where:
  - $V$ is a (finite) set
  - $E$ is a set of ordered pairs $(u, v)$ where $u, v$ are in $V$
  - Often (not always): $u \neq v$ (i.e. no edges from a vertex to itself)

- An element in $V$ is called a vertex or node

- Elements in $E$ are called edges or arcs

- $|V| = \text{size of } V$ (traditionally called $n$)

- $|E| = \text{size of } E$ (traditionally called $m$)
An example directed graph

\[ V = \{A, B, C, D, E\} \]
\[ E = \{(A, C), (B, A), (B, C), (C, D), (D, C)\} \]
\[ |V| = 5 \]
\[ |E| = 5 \]
Graphs, Formally

• An **undirected graph** is a just like a digraph, but
  
  • E is a set of unordered pairs (u, v) where u, v are in V

  \[
  V = \{A, B, C, D, E\} \\
  E = \{\{A, C\}, \{B, A\}, \{B, C\}, \{C, D\}\} \\
  |V| = 5 \\
  |E| = 4
  \]

• An **undirected graph** can be converted to an equivalent **directed** graph:
  
  • Replace each undirected edge with two directed edges in opposite directions

• A **directed** graph can’t always be converted to an **undirected** graph.
Graph Terminology: Adjacency, Degree

- Two vertices are adjacent if they are connected by an edge.
- Nodes u and v are called the source and sink of the directed edge (u, v).
- Nodes u and v are endpoints of an edge (u, v) (directed or undirected).
- The outdegree of a vertex u in a directed graph is the number of edges for which u is the source.
- The indegree of a vertex v in a directed graph is the number of edges for which v is the sink.
- The degree of a vertex u in an undirected graph is the number of edges of which u is an endpoint.
Graph Terminology: Paths, Cycles

- A **path** is a sequence of vertices where each consecutive pair are adjacent.
- In a directed graph, paths must follow the direction of the edges (nodes must be ordered source then sink).
- A **cycle** is a path that ends where it started, e.g.: x, y, z, x
- A graph is **acyclic** if it has no cycles.