This week only: Lab room and OH Changes

Friday April 14th: AE3’s Celebration of Teaching in 1306 Everitt

Our lab will be in 2101 Everitt instead!

Office Hour Changes: My OH will be Friday between 3:15 and 4:15
There will not be OH on Thursday April 13th!
Lab Feedback

Still read them and appreciate feedback

lab_huffman needs work in the future in both presentation and content

lab_trees and lab_avl were both highly rated
Learning Objectives

Review edge list and adjacency matrix graph implementations

Introduce adjacency list implementation

Discuss the strengths and weaknesses of each implementation
Graphs

Given a roster of students for each class, build a graph which tracks whether there are at least three students in common between two classes.

What is a vertex?

What is an edge?

Are the edges directed or undirected?

Are the edges weighted or unweighted?
Graph ADT

Find
getVertices() — return the list of vertices in a graph
getEdges(v) — return the list of edges that touch the vertex v
areAdjacent(u, v) — returns a bool based on if an edge from u to v exists

Insert
insertVertex(v) — adds a vertex to the graph
insertEdge(u, v) — adds an edge to the graph

Remove
removeVertex(v) — removes a vertex from the graph
removeEdge(u, v) — removes an edge from the graph
Graph Implementation: Edge List $|V| = n, |E| = m$

The equivalent of an ‘unordered’ data structure

**Vertex Storage:**

Not stored at all (recovered from edges)
or
An unordered list of vertices

**Edge Storage:**

An unordered list of edges (as tuples)
[or equivalent]
Graph Implementation: Edge List

How would our data structure change if...

Edges are weighted:

\[ u \rightarrow v \]
\[ v \rightarrow w \]
\[ w \rightarrow z \]
Graph Implementation: Edge List

How would our data structure change if…

Edges are directed:
Graph Implementation: Adjacency Matrix

Vertex Storage:

Edge Storage:

<table>
<thead>
<tr>
<th></th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>W</td>
<td>V</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Graph Implementation: Adjacency Matrix

getVertices():

<table>
<thead>
<tr>
<th></th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Graph Implementation: Adjacency Matrix

getEdges(v):

```
Graph Implementation: Adjacency Matrix

u
w
v

getEdges(v):

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Graph Implementation: Adjacency Matrix

areAdjacent(u, v):

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Graph Implementation: Adjacency Matrix

**insertVertex(v):**

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Graph Implementation: Adjacency Matrix

insertEdge(u, v):

\[
\begin{array}{cccc}
\text{u} & \text{v} & \text{w} & \text{z} \\
\hline
\text{u} & 0 & 1 & 1 & 0 \\
\text{v} & 1 & 0 & 1 & 0 \\
\text{w} & 1 & 1 & 0 & 1 \\
\text{z} & 0 & 0 & 1 & 0 \\
\end{array}
\]
Graph Implementation: Adjacency Matrix

removeVertex(v):

removeEdge(u, v):

\[
\begin{array}{c|cccc}
  & u & v & w & z \\
\hline
  u & 0 & 1 & 1 & 0 \\
  v & 1 & 0 & 1 & 0 \\
  w & 1 & 1 & 0 & 1 \\
  z & 0 & 0 & 1 & 0 \\
\end{array}
\]
Graph Implementation: Adjacency Matrix

Pros:

Cons:
Graph Implementation: Adjacency Matrix

How would our data structure change if...

Edges are directed:

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Graph Implementation: Adjacency Matrix

How would our data structure change if...

Edges are weighted:

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Adjacency List

Vertex Storage:
Edge Storage:
Adjacency List

Vertex Storage:

Edge Storage:
Adjacency List

getVertices():

u
v
w
z

u
d=2
v
d=2
w
d=3
z
d=1

v
w
u
z
z
Adjacency List

getEdges(v):

\[\text{Adjacency List}\]

\[\begin{align*}
\text{u} & \rightarrow \text{v} \\
\text{v} & \rightarrow \text{w} \\
\text{w} & \rightarrow \text{z} \\
\text{z} & \rightarrow \text{w}
\end{align*}\]
Adjacency List

areAdjacent(u, v):

![Graph Diagram]

- u
  - v
    - w
  - w
    - v
    - u
    - z
  - z
    - d=1
    - d=3
    - d=2

- v
  - u
    - w
  - d=2
Adjacency List

insertVertex(v):

u

v

w

z

d=2
d=2
d=3
d=1
Adjacency List

```
removeVertex(v):
```

![Graph representation]
Adjacency List

insertEdge(u, v):

- u
- v
- w
- z

u → v → w
v → u → w
w → v → u
z → z
Adjacency List

removeEdge(u, v):

V
W
Z

u
v
w
z

d=2
d=3
d=1
Adjacency List

Pros:

Cons:
Adjacency List

How would our data structure change if…

Edges are directed:
Adjacency List

How would our data structure change if…

Edges are weighted:
<table>
<thead>
<tr>
<th>Expressed as $O(f)$</th>
<th>Edge List</th>
<th>Adjacency Matrix</th>
<th>Adjacency List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space</strong></td>
<td>n+m</td>
<td>n$^2$</td>
<td>n+m</td>
</tr>
<tr>
<td><strong>insertVertex(v)</strong></td>
<td>1*</td>
<td>n*$^*$</td>
<td>1*</td>
</tr>
<tr>
<td><strong>removeVertex(v)</strong></td>
<td>m**</td>
<td>n</td>
<td>deg(v)</td>
</tr>
<tr>
<td><strong>insertEdge(u, v)</strong></td>
<td>1</td>
<td>1</td>
<td>1*</td>
</tr>
<tr>
<td><strong>removeEdge(u, v)</strong></td>
<td>m</td>
<td>1</td>
<td>min( deg(u), deg(v) )</td>
</tr>
<tr>
<td><strong>getEdges(v)</strong></td>
<td>m</td>
<td>n</td>
<td>deg(v)</td>
</tr>
<tr>
<td><strong>areAdjacent(u, v)</strong></td>
<td>m</td>
<td>1</td>
<td>min( deg(u), deg(v) )</td>
</tr>
</tbody>
</table>
Next week: Traversals

There is no clear order in a graph (even less than a tree!)

How can we systematically go through a complex graph in the fewest steps?