CS 225
Data Structures

April 12 – Graphs
G Carl Evans
In Review: Data Structures

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

**Linked**
- Doubly Linked 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

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

Graphs
The Internet 2003
*The OPTE Project (2003)*
Map of the entire internet; nodes are routers; edges are connections.
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

“Rule of 7”
Unknown Source
Presented by Cinda Heeren, 2016
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/
“Stanford Bunny”
Greg Turk and Mark Levoy (1994)
HAMLET

TROILUS AND CRESSIDA
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):
\[ G' = (V', E') \]
\[ V' \in V, E' \in E, \text{ and } (u, v) \in E' \Rightarrow u \in V', v \in V' \]

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:

\[
\sum_{v \in V} \text{deg}(v) = \]

\[
\]
Connected Graphs
Proving the size of a minimally connected graph

**Theorem:**
Every connected graph $G=(V, E)$ has at least $|V|-1$ edges.
**Thm:** Every connected graph $G=(V, E)$ has at least $|V|-1$ edges.

**Proof:** Consider an arbitrary, connected graph $G=(V, E)$. 
Suppose $|V| = 1$:

**Definition:** A connected graph of 1 vertex has 0 edges.

**Theorem:** $|V|-1$ edges $\implies 1-1 = 0$. 
Inductive Hypothesis: For any $j < |V|$, any connected graph of $j$ vertices has at least $j-1$ edges.
Suppose $|V| > 1$:

1. Choose any vertex:

2. Partition:
Suppose $|V| > 1$:

3. Count the edges
Graph ADT

Data:
- Vertices
- Edges
- Some data structure maintaining the structure between vertices and edges.

Functions:
- insertVertex(K key);
- insertEdge(Vertex v1, Vertex v2, K key);
- removeVertex(Vertex v);
- removeEdge(Vertex v1, Vertex v2);
- incidentEdges(Vertex v);
- areAdjacent(Vertex v1, Vertex v2);
Graph Implementation Idea

\[ \begin{align*}
&V & \rightarrow & W & \rightarrow & Z \\
&u & \rightarrow & v & \rightarrow & w & \rightarrow & z
\end{align*} \]
Graph Implementation: Edge List

Vertex Collection:

Edge Collection:
Graph Implementation: Edge List

**insertVertex(K key):**

**removeVertex(Vertex v):**

```
  u   v   a
  v   w   b
  w   c
  z   d
```
Graph Implementation: Edge List

incidentEdges(Vertex v):

areAdjacent(Vertex v1, Vertex v2):

G.incidentEdges(v1).contains(v2)
Graph Implementation: Edge List

`insertEdge(Vertex v1, Vertex v2, K key):`

```
  u   v   a
  v   w   b
  u   w   c
  w   z   d
```
Graph Implementation: Adjacency Matrix

```
  u  v  a
  v  w  b
  w  z  d
```

```
<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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Graph Implementation: Adjacency Matrix

```
  u  v  w  a
  v  w  b
  w  z  d
```

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

The diagram above illustrates a graph with vertices labeled u, v, w, and z. The edges connect these vertices as follows:
- u to v
- u to c
- v to w
- w to z
- w to d
- u to b

The adjacency matrix below represents the connections in the graph:

<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>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>0</td>
<td>0</td>
<td>1</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(K key):

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

**removeVertex(Vertex 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>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>v</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>w</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>z</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Graph Implementation: Adjacency Matrix

incidentEdges(Vertex v):

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</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(Vertex v1, Vertex v2):
Graph Implementation: Adjacency Matrix

```plaintext
insertEdge(Vertex v1, Vertex v2, K key):
```

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td></td>
<td></td>
<td>u</td>
</tr>
<tr>
<td>v</td>
<td></td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>w</td>
<td>v</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>u</td>
<td>w</td>
<td>c</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></td>
<td>u</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>-</td>
<td></td>
<td>v</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td></td>
<td>u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
Graph Implementation: Edge List

- The graph consists of vertices u, v, w, and z.
- Edges are represented as pairs: (u, v), (v, w), (u, w), (w, z), and (u, c).

Diagram:
- u connected to a, b, and c.
- v connected to b and w.
- w connected to b, d, and c.
- z connected to d.

Edge List:
- (u, v, a)
- (v, w, b)
- (u, w, c)
- (w, z, d)
Adjacency List

u → a, c
v → a, b
w → b, c, d
z → d

a, c ∈ u
a, b ∈ v
b, c, d ∈ w
b, c, d ∈ z

u, v, a
v, w, b
u, w, c
w, z, d
Adjacency List

```
insertVertex(K key):
```

![Diagram of an adjacency list representation of a graph with vertices u, v, w, z and edges a-b, a-c, b-d, w-z, w-d, u-w, v-w, and v-d, with key values associated with each vertex and edge.]
Adjacency List

removeVertex(Vertex v):

u

v

w

z

d=2

d=2

d=3

d=1

u

v

w

z

a

b

c

d

u

v

w

z

a

b

c

d

u

v

w

z

a

b

c

d
Adjacency List

incidentEdges(Vertex v):

\[
\begin{array}{c}
\text{incidentEdges}(u) = \{v, w, c\} \\
\text{incidentEdges}(v) = \{a, b\} \\
\text{incidentEdges}(w) = \{b, c, d\} \\
\text{incidentEdges}(z) = \{w, z, d\}
\end{array}
\]
Adjacency List

areAdjacent(Vertex v1, Vertex v2):
Adjacency List

insertEdge(Vertex v1, Vertex v2, K key):
<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>Space</td>
<td>$n+m$</td>
<td>$n^2$</td>
<td>$n+m$</td>
</tr>
<tr>
<td>insertVertex($v$)</td>
<td>$1$</td>
<td>$n$</td>
<td>$1$</td>
</tr>
<tr>
<td>removeVertex($v$)</td>
<td>$m$</td>
<td>$n$</td>
<td>$\text{deg}(v)$</td>
</tr>
<tr>
<td>insertEdge($v$, $w$, $k$)</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>removeEdge($v$, $w$)</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>incidentEdges($v$)</td>
<td>$m$</td>
<td>$n$</td>
<td>$\text{deg}(v)$</td>
</tr>
<tr>
<td>areAdjacent($v$, $w$)</td>
<td>$m$</td>
<td>$1$</td>
<td>$\min(\text{deg}(v), \text{deg}(w))$</td>
</tr>
</tbody>
</table>
**Traversal:**

**Objective:** Visit every vertex and every edge in the graph.

**Purpose:** Search for interesting sub-structures in the graph.

We’ve seen traversal before ....but it’s different:

- Ordered
- Obvious Start

- 
- 
- 
-
Traversal: BFS
Graph ADT

Data:
- Vertices
- Edges
- Some data structure maintaining the structure between vertices and edges.

Functions:
- `insertVertex(K key);`
- `insertEdge(Vertex v1, Vertex v2, K key);`
- `removeVertex(Vertex v);`
- `removeEdge(Vertex v1, Vertex v2);`
- `incidentEdges(Vertex v);`
- `areAdjacent(Vertex v1, Vertex v2);`
Graph Implementation Idea
Graph Implementation: Edge List

Vertex Collection:

Edge Collection:
Graph Implementation: Edge List

insertVertex(K key):

removeVertex(Vertex v):
Graph Implementation: Edge List

incidentEdges(Vertex v):

areAdjacent(Vertex v1, Vertex v2):

G.incidentEdges(v1).contains(v2)
Graph Implementation: Edge List

insertEdge(Vertex v1, Vertex v2, K key):

```
  u   v   a
  v   w   b
  w   u   c
  z   w   d
```
Graph Implementation: Adjacency Matrix
Graph Implementation: Adjacency Matrix

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

```
insertVertex(K key):
```

```
    u  v  a
   ____________
   |           |
   |           |
   |           |
   |___________|
   w  b  c
```

```
    u  v  w  z
   ____________
   |           |
   |           |
   |           |
   |___________|
   v  -   0
```

```
    u  v  w  z
   ____________
   |           |
   |           |
   |           |
   |___________|
   w  -   0
```

```
    u  v  w  z
   ____________
   |           |
   |           |
   |           |
   |___________|
   z  -   -
```
Graph Implementation: Adjacency Matrix

removeVertex(Vertex 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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Graph Implementation: Adjacency Matrix

incidentEdges(Vertex 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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
incidentEdges:
- u to v: 0
- u to a: 0
- u to c: 0
- v to w: 0
- v to b: 0
- w to u: 0
- w to c: 0
- w to z: 0
- z to w: 0
```
Graph Implementation: Adjacency Matrix

\[ \text{areAdjacent(Vertex v1, Vertex v2):} \]

<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>v</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>v</td>
<td>w</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>u</td>
<td>w</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>w</td>
<td>z</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>
Graph Implementation: Adjacency Matrix

insertEdge(Vertex v1, Vertex v2, K key):
Graph Implementation: Edge List

- Vertices: u, v, w, z
- Edges: (u, v), (v, w), (u, w), (w, z)
Adjacency List
Adjacency List

insertVertex(K key):

V

W

Z

a

b

c

d

\( u \)

\( v \)

\( w \)

\( z \)

\( a \)

\( b \)

\( c \)

\( d \)

\( u \)

\( v \)

\( a \)

\( v \)

\( w \)

\( b \)

\( u \)

\( w \)

\( c \)

\( w \)

\( z \)

\( d \)
Adjacency List

removeVertex(Vertex v):

V: a, b, c, d
W: u, v, w, z

u → {a, c}
v → {a, b}
w → {b, c, d}
z → {d}

u = 2
v = 2
w = 3
z = 1
Adjacency List

incidentEdges(Vertex v):

\[
\begin{align*}
\text{incidentEdges}(v) = & \{a, b, c, d\} \\
\text{incidentEdges}(w) = & \{a, b, c, d\} \\
\text{incidentEdges}(z) = & \{a, b, c, d\}
\end{align*}
\]
Adjacency List

areAdjacent(Vertex v1, Vertex v2):

```plaintext
u  v  a
v  w  b
w  z  c
z  d  
```
Adjacency List

insertEdge(Vertex v1, Vertex v2, K key):
<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>Space</td>
<td>$n+m$</td>
<td>$n^2$</td>
<td>$n+m$</td>
</tr>
<tr>
<td>$\text{insertVertex}(v)$</td>
<td>1</td>
<td>$n$</td>
<td>1</td>
</tr>
<tr>
<td>$\text{removeVertex}(v)$</td>
<td>$m$</td>
<td>$n$</td>
<td>$\text{deg}(v)$</td>
</tr>
<tr>
<td>$\text{insertEdge}(v, w, k)$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\text{removeEdge}(v, w)$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\text{incidentEdges}(v)$</td>
<td>$m$</td>
<td>$n$</td>
<td>$\text{deg}(v)$</td>
</tr>
<tr>
<td>$\text{areAdjacent}(v, w)$</td>
<td>$m$</td>
<td>1</td>
<td>$\min(\text{deg}(v), \text{deg}(w))$</td>
</tr>
</tbody>
</table>
Traversal:

**Objective:** Visit every vertex and every edge in the graph.

**Purpose:** Search for interesting sub-structures in the graph.

We’ve seen traversal before ....but it’s different:

- Ordered
- Obvious Start
Traversal: BFS