CS 225
Data Structures

March 30 – Graph Implementation
G Carl Evans
Graphs

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

![Graph Diagram]
Graph Implementation: Adjacency Matrix
Graph Implementation: Adjacency Matrix

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

Adjacency Matrix:

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

The adjacency matrix for the given graph is shown below. Each row and column corresponds to a vertex in the graph. The adjacency matrix represents the connections between vertices, where a non-zero entry indicates an edge between the vertices. In this case:

- The entry in row 1, column 2 is 1, indicating an edge between vertices u and v.
- The entry in row 2, column 3 is 1, indicating an edge between vertices v and w.
- The entry in row 3, column 4 is 1, indicating an edge between vertices w and z.

The adjacency matrix is:

```
  u  v  w  z
u  0  1  0  0
v  1  0  1  0
w  0  1  0  1
z  0  0  1  0
```
Graph Implementation: Adjacency Matrix

**insertVertex(K key):**
Graph Implementation: Adjacency Matrix

removeVertex(Vertex v):

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

incidentEdges(Vertex v):
Graph Implementation: Adjacency Matrix

areAdjacent(Vertex v1, Vertex v2):

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

```
  u  v  w  z
u  0  0  0
v  -  0  0
w  -  0  0
z  -  -  -
```
Graph Implementation: Adjacency Matrix

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

```
<table>
<thead>
<tr>
<th></th>
<th>v</th>
<th>w</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td></td>
<td>v</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>v</td>
<td></td>
<td>w</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td></td>
<td>v</td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>z</td>
<td></td>
<td>w</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>0</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></td>
</tr>
</tbody>
</table>
```
Graph Implementation: Edge List

- Vertices: u, v, w, z
- Edges: (u, v), (v, w), (u, w), (w, z), (u, c), (v, b), (w, d)

Diagram:
- u connected to v, w via edges a, c
- v connected to w via edge b
- w connected to z via edge d
- u connected to w via edge c
Adjacency List

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>2</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
</tr>
<tr>
<td>w</td>
<td>3</td>
</tr>
<tr>
<td>z</td>
<td>1</td>
</tr>
</tbody>
</table>
Adjacency List

- u connects to a, c
- v connects to a, b
- w connects to b, c, d
- z connects to c, d

- u connects to a
- v connects to a
- w connects to b, c, d
- z connects to c, d
Adjacency List

```
adjList = {
    'u': ['a', 'c'],
    'v': ['b', 'd'],
    'w': ['a', 'c', 'b', 'd'],
    'z': ['d'],
}
```

**insertVertex**(K key):
- Insert vertex 'a' with degree 2
- Insert vertex 'b' with degree 2
- Insert vertex 'c' with degree 3
- Insert vertex 'd' with degree 1
Adjacency List

removeVertex(Vertex v):

- Remove vertex v from the graph.
- Update the adjacency lists of all vertices connected to v.
- Adjust the degrees of the vertices affected by v's removal.
- Ensure the graph remains connected and acyclic.

Diagram shows the process visually with vertex removal and updates to adjacency lists.
Adjacency List

incidentEdges(Vertex v):

- u
  - a
  - c
- v
  - a
  - b
- w
  - b
  - c
  - d
- z
  - d

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

d=2

- v
  - d=2
- w
  - d=3
- z
  - d=1
Adjacency List

areAdjacent(Vertex v1, Vertex v2):

adjacent vertices are connected by edges.
Adjacency List

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

- **u** connected to **v**, **w**, and **c**
- **v** connected to **u** and **b**
- **w** connected to **v**, **b**, **c**, and **d**
- **z** connected to **w** and **d**

Vertex and edge properties:
- **u** connected to **v** with `d=2`
- **v** connected to **w** with `d=2`
- **w** connected to **z** with `d=3`
- **z** connected to **d** with `d=1`
<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