Algorithms and Data Structures for Data Science
Graphs

CS 277
Brad Solomon

November 1, 2021

Department of Computer Science
Exam 2

Average (before partial credit):
Final Project Mid-Project Check-ins

Mid-project check-ins start next week (November 8th-12th)!

Proposal resubmissions due November 5th

To sign up for a mid-project meeting time:

Learning Objectives

Define graph vocabulary

Discuss graph implementation / storage strategies

Define key graph functions and discuss implementation details
In Review: Data Structures

Array
- Sorted Array
- Unsorted Array
  - Stacks
  - Queues

Linked
- Linked List
- Trees
  - Binary Tree
    - Huffman Encoding
  - BST
  - AVL Tree
The Internet 2003


Map of the entire internet; nodes are routers; edges are connections.

Example from Carl Evans
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
“Rush Hour” Solution

Figure from Christopher Wolfram
Idea from Cinda Heeren
“Stanford Bunny”
Greg Turk and Mark Levoy (1994)
To study all of these structures:
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms
Graph Vocabulary

$G = (V, E)$

Vertex:

Edges:

Cardinality:
Graph Vocabulary

$G = (V, E)$

Degree:

Adjacent:

Path:

Cycle:
Connected Graphs
Graph Efficiency

Graph efficiency has two factors:

\( n \), the number of vertices

\( m \), the number of edges.

Minimum Edges:

Maximum Edges:

\[ \sum_{v \in V} deg(v) = \]
Graph Implementation: Edge List

\(|V| = n, |E| = m\)

Vertex Storage:

Edge Storage:
Graph Implementation: Edge List  \(|V| = n, |E| = m\)

insertVertex(v):

removeVertex(v):

|V| = n, |E| = m
Graph Implementation: Edge List

$|V| = n, |E| = m$

**insertEdge(u, v):**

**removeEdge(u, v):**
Graph Implementation: Edge List  \(|V| = n, |E| = m\)

getEdges(u):

areAdjacent(u, v):

Graph Implementation: Edge List  \(|V| = n, |E| = m\)

Pros:

Cons:
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></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

insertVertex(v):

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

removeVertex(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>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

insertEdge(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>-</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

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

```
getEdges(u):
```

<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>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- u
- v
- w
- z
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>-</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

Pros:

Cons:

<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>
Adjacency List

Vertex Storage:

Edge Storage:
Adjacency List

Vertex Storage:

Edge Storage:
insertVertex(v):

Adjacency List
Adjacency List

\[ \text{removeVertex}(v): \]

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

**insertEdge(u, v):**

- **u**
- **v**
- **w**
- **z**

Node distances:
- **u** \( d=2 \)
- **v** \( d=2 \)
- **w** \( d=3 \)
- **z** \( d=1 \)
Adjacency List

removeEdge(u, v):

u
v
w
z
Adjacency List

getEdges(u):

u → v → w → z

v → u → w → z

w → u → v → z

z → w → v → u
Adjacency List

areAdjacent(u, v):
Adjacency List

Pros:

Cons:
<table>
<thead>
<tr>
<th></th>
<th>Edge List</th>
<th>Adjacency Matrix</th>
<th>Adjacency List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expressed as O(f)</td>
<td></td>
<td></td>
<td></td>
</tr>
<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>deg(v)</td>
</tr>
<tr>
<td>insertEdge(u, v)</td>
<td>1</td>
<td>1</td>
<td>1*</td>
</tr>
<tr>
<td>removeEdge(u, v)</td>
<td>m</td>
<td>1</td>
<td>min( deg(u), deg(v) )</td>
</tr>
<tr>
<td>getEdges(v)</td>
<td>m</td>
<td>n</td>
<td>deg(v)</td>
</tr>
<tr>
<td>areAdjacent(u, v)</td>
<td>m</td>
<td>1</td>
<td>min( deg(u), deg(v) )</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

-