Warmup Game

• Discuss/Design data structures for these tasks:

A. Model connectivity of airline flights & plan connections

B. Optimize traffic flow for rapid evacuation of a city

C. Represent possible moves in a state-based game like Cat’s Cradle or Rubik’s Cube

D. Represent friend relationships on Facebook or other social networks

E. Diagram the link connections between web pages

F. Design an electric power grid that can ensure delivery of power from source to users without overloading
Graphs: Representing Connectivity

• Examples involve connectivity in some form
  *(Identify how this applies in your case)*

• Two parts to the model:
  – **Vertex** (or **node**) is junction for connections
  – **Edge** is connection between two nodes
    ▪ Has two ends, called **head** and **tail**
    ▪ May be directed (one-way) or undirected (two-way)

• Both nodes and edges may have additional data associated with them
  – Cost, capacity, strength, id, etc.
Brainstorm:
Graphs for Representation

TECHIE CAREER TRACK FLOWCHART

START HERE

BRILLIANT, NAIVE TECHIE

GET PROMOTED TO MANAGEMENT.

WERE YOU SUCCESSFUL?

CONGRATS, YOU'RE NOW A CYNICAL, GRUMPY, OLD-SCHOOL TECHNICAL GURU.

YOU'RE NOT STERILE, BUT YOU WON'T BREED.

YOU'RE NOW STERILE FROM WORKING NEAR WEAPONS. YOU WON'T BREED.

HAVE YOU DONE TECH SUPPORT?

ARE YOU BITTER YET?

DO YOU HATE PEOPLE YET?

BECOME A CODER.

BECOME A DEFENSE CONTRACTOR.

YOU'RE A DANGER TO THE PUBLIC.

DO YOU WANT TO PLAY WITH NUCLEAR WEAPONS YET?

HAVE YOU WORKED WITH WINDOWS NT?

HAVE YOU BEEN A SYSADMIN?

NO

YES

COPYRIGHT (C) 2000 ILLIAD

HTTP://WWW.USERFRIENDLY.ORG/
Brainstorm:
Graphs for Representation

• Networks
  – Transportation
  – Communication
  – Transmission
• State diagrams & flow charts
  – Control systems
  – Automata
• Relationships
  – Disease transmission
  – Social networks
  – Semantic networks
Brainstorm: Graphs for Computation

• Physical connections
  – Distance
  – Connectivity
  – Shortest path

• Capacity
  – Maximum flow
  – Choke points

• Boundary Analysis
  – Coloring problems
  – Segmentation
Formal Definitions

• A graph $G$ consists of two sets $(V,E)$
  $V = \{v_0, v_1, v_2, \ldots\}$ are the vertices
  $E = \{e_0, e_1, e_2, \ldots\}$ are the edges
  $e_k = \{v_{Hk}, v_{Tk}\}$ if undirected, $(v_{Hk}, v_{Tk})$ if directed
  Sometimes written $e_{ij}$ where $i$ is head vertex and $j$ is tail

• Variations
  – Directed? $e_k = (v_{Hk}, v_{Tk})$
  – Undirected? $e_k = \{v_{Hk}, v_{Tk}\}$
  – Self-edges? $H_k == T_k$
  – Duplicate edges? $E$ is collection, not set
Related Definitions

• Number of edges touching a node is its **degree**
  – **In-degree**: number of incoming edges
  – **Out-degree**: number of outgoing edges
  – **Degree of a graph** is maximum degree of its edges

• Two nodes that share an edge are **neighbors**

• **Subgraph** is graph \((U,F)\) where \(U \subseteq V, F \subseteq E\)

• **Clique** is any subgraph whose nodes are all neighbors of each other
  – A clique with \(k\) members called a **\(k\)-clique**
Paths & Cycles

• A path is a sequence of edges $e_{p_0}, e_{p_1}, \ldots, e_{p_{s-1}}$ such that $\text{head}(e_{p_i}) = \text{tail}(e_{p_{i+1}})$
  – Two nodes are connected if a path includes both
  – Simple path visits each vertex no more than once
  – Cycle is path that starts and ends at same node: $\text{tail}(e_{p_0}) = \text{head}(e_{p_{s-1}})$

Acyclic graph is a graph with no cycles
Graphs & Trees

• A tree may be seen as a special form of graph
  – No cycles
  – One simple path from root to any node
• Generalization of tree: **Directed Acyclic Graph**
• **Embedded tree**: any subgraph that is a tree

Spanning tree: any subtree that includes all nodes of the original graph
Bipartite Graphs

• Nodes comprise two disjoint sets A & B such that $V = A \cup B$ and $A \cap B = 0$

• Also, all edges have one end in A and the other in B

• *Bipartite graph matching* seeks to find low-cost pairing between A and B
Graph Isomorphism

- Graphs $G, H$ are **isomorphic** if there exists a map $f: V_G \rightarrow V_H$ such that nodes $u, v$ in $V_G$ are neighbors in $G$ if and only if $f(u), f(v)$ are neighbors in $H$.

Which are isomorphic?

- $M, S, V, Z$
- $F, T$
- $K, X$
- $R, Q$
Activity

1. Draw a graph with 6 nodes and 10 edges
2. On new paper, describe your graph in words
3. Swap descriptions with a partner
   *Don’t let them see your diagram!*
4. Try to reproduce your partner’s graph from their description
5. Now compare your picture with the original
   *Is it isomorphic?*
6. What strategies helped to describe the graph?
Design Choices

• Sparse vs. Full Notation
  Full: Include an entry for every possible item
  Sparse: Include entries for present items only

• Ex: Primes between 1 and 16
  Full: NYYNYNYNNNYNNYNNN  ↩️ 16 bits
  Sparse: \{2, 3, 5, 7, 11, 13\}  ↩️ 24 bits

• Edge-Primary vs. Node-Primary
  Organize listing around which entity?
Adjacency Matrix

• Vertex-Primary Full Representation

<table>
<thead>
<tr>
<th></th>
<th>v₀</th>
<th>v₁</th>
<th>v₂</th>
<th>v₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>v₀</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>v₁</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>v₂</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>v₃</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Head

tail

How would self-edges be represented?
Duplicate edges?

What do you get when you multiply an adjacency matrix?

\[
\begin{bmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix} \times 
\begin{bmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix} = 
\begin{bmatrix}
1 & 1 & 0 & 2 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]
Write down the adjacency matrix for the graph shown below.
Edge Matrix

- Edge-Primary Semifull

<table>
<thead>
<tr>
<th>e</th>
<th>v₀</th>
<th>v₁</th>
<th>v₂</th>
<th>v₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>e₀</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e₁</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>e₂</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e₃</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>e₄</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>e₅</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Perhaps not as useful as the adjacency matrix
Your Turn

Write down the edge matrix for the graph shown below.

\[
\begin{array}{cccc}
\text{P} & \text{Q} & \text{R} & \text{S} \\
e_0 & 0 & 0 & -1 & 1 \\
e_1 & 0 & -1 & 1 & 0 \\
e_2 & 0 & 1 & 0 & -1 \\
e_3 & 0 & -1 & 0 & 1 \\
\end{array}
\]
Sparse Representation

• By vertices:

\[
\begin{align*}
v_0 &: v_1, v_2 \\
v_1 &: v_0, v_3 \\
v_2 &: v_1, v_3 \\
v_3 &: \emptyset
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>(v_0)</th>
<th>(v_1)</th>
<th>(v_2)</th>
<th>(v_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(v_0)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(v_1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(v_2)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(v_3)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Each line corresponds to a row

• By edges:

\[
\begin{align*}
e_0 &: v_0, v_1 \\
e_1 &: v_0, v_2 \\
e_2 &: v_1, v_0 \\
e_3 &: v_1, v_3 \\
e_4 &: v_2, v_1 \\
e_5 &: v_2, v_3
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>(v_0)</th>
<th>(v_1)</th>
<th>(v_2)</th>
<th>(v_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(e_0)</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(e_1)</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>(e_2)</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(e_3)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>(e_4)</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(e_5)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Java Representation

• Previous representations only show form
  – Nodes can have data (label, type, position, etc.)
  – Edges can have data (weight, capacity, strength)
• Need additional structure(s) to represent this
• Java will use generic class for nodes, edges
  – Vertex will have list of reference to its edges
  – Edge will have references to its head & tail nodes
  – Graph will keep master list of all edges & nodes
Data structure below represents this graph:

- Each node links to its edges
- Edges link back to their nodes

Note the pairs of connections between edges and nodes.
g.nodes.get(1).edges.get(0).data → ?
g.nodes.get(1).edges.get(0).data
Navigation

g.get(1).edges.get(0).data
g.nodes.get(1).edges.get(0).data
g.nodes[1].edges[0].data
g.nodes.get(1).edges.get(0).data
g.nodes.get(1).edges.get(0).data
g.nodes.get(1).edges.get(0).data
g.nodes.get(1).edges.get(0).data
g.nodes.get(1).edges.get(0).data
g.nodes.get(1).edges.get(0).data
g.nodes.get(1).edges.get(0).data
g.nodes.get(1).edges.get(0).data \rightarrow X
Your Turn

Diagram the memory structure to represent the graph pictured below.