Recap:
- Normal-ish rest of the week
- HW due Wednesday
- Final worksheet posted
- Review next Monday in class
- Test next Wednesday @ 8am
- Sample finals tomorrow
A graph \( G = (V, E) \) is an ordered pair of 2 sets:

- \( V = \text{vertices} = \{v_1, v_2, v_3, v_4\} \)
- \( E = \text{edges} = \{e_{v_1, v_2}, e_{v_2, v_3}, e_{v_3, v_4}, e_{v_4, v_1}\} \)

**View:**

- Peterson graph
Representing graphs

How do we make this data structure?

- pointers! 

 like:

V

E

phso
Adjacency (or vertex) lists:

- $V_1 \sim V_2, V_5$
- $V_2 \sim V_1, V_3, V_5$
- $V_3 \sim$
- $V_4 \sim$
- $V_5 \sim$

Size: $n$ "lists," each size $\leq n-1$

Lookup: Time to check if $v_i \sim v_j$:

- $O(n)$
- (or $O(\log n)$)

Upper bound: $O(n+m)$
Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>V3</td>
<td></td>
<td></td>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>V4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>V5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Directed: need both "halves" of matrix

Space: $O(n^2)$

Check nbr: $O(1)$

$A[i][j]$
Which is better? Depends!

<table>
<thead>
<tr>
<th></th>
<th>Adjacency matrix</th>
<th>Standard adjacency list (linked lists)</th>
<th>Adjacency list (hash tables)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>Θ(V^2)</td>
<td>Θ(V + E)</td>
<td>Θ(V + E)</td>
</tr>
<tr>
<td>Time to test if ( uv \in E )</td>
<td>O(1)</td>
<td>O(1 + min{deg(u), deg(v)}) = O(V)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Time to test if ( u \rightarrow v \in E )</td>
<td>O(1)</td>
<td>O(1 + deg(u)) = O(V)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Time to list the neighbors of ( v )</td>
<td>O(V)</td>
<td>O(1 + deg(v))</td>
<td>O(1 + deg(v))</td>
</tr>
<tr>
<td>Time to list all edges</td>
<td>Θ(V^2)</td>
<td>Θ(V + E)</td>
<td>Θ(V + E)</td>
</tr>
<tr>
<td>Time to add edge ( uv )</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)*</td>
</tr>
<tr>
<td>Time to delete edge ( uv )</td>
<td>O(1)</td>
<td>O(deg(u) + deg(v)) = O(V)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

\[ α(n^2) \text{ space} \quad O(n+m) \]

Libraries: Boost, etc.
Dfn:
- $G$ is connected if for all $u, v$, there exists a path from $u$ to $v$.
- The distance from $u$ to $v$, $d(u, v)$, is equal to the sum of weights of edges on the minimum $u, v$-path (sum of weights).

$d(u, v') = 0 \Rightarrow G$ is connected.

$d(u, v) = 2$ implies $G$ is disconnected.
Algorithms on graphs

Basic 1st question:

Given any 2 vertices, are they connected?
Also: What is their distance?

How to solve?
Suppose we’re in a maze searching for something. What do you do?

Depth first search
- go as far as you can

Breadth first search
check nbs,
then their nbs, etc.
Pseudo-code: two versions

**RecursiveDFS(v):**
- if v is unmarked
  - mark v
  - for each edge vw
    - RecursiveDFS(w)

**IterativeDFS(s):**
- Push(s)
- while the stack is not empty
  - v ← Pop
  - if v is unmarked
    - mark v
    - for each edge vw
      - Push(w)

$O(m+n)$

Really, building a "tree":

DFS tree:

Stack: $v_1 v_3 v_5 v_6 v_7 v_8 v_9 v_10$
**General traversal strategy**

**TRVERSE(s):**
- put s into the bag
- while the bag is not empty
  - take \( v \) from the bag
  - if \( v \) is unmarked
    - mark \( v \)
    - for each edge \( vw \)
      - put \( w \) into the bag

**Q:** Can we use a different "bag"?

\[ \text{queue} \Rightarrow O(m+n) \]
BFS: use a queue

 Traverse(s):
 put s into the bag
 while the bag is not empty
 take v from the bag
 if v is unmarked
 mark v
 for each edge vw
 put w into the bag

 BFS tree:
BFS vs. DFS:

- Both do connectivity
- Both are $O(m+n)$ time (w/ either graph rep)

- Difference:
  What you are optimizing for.
Next time:
- directed searching
- weighted graphs