CSCI 3100

Today: Graphs
Announcements

- Boeing scholarships
- HW due Monday
- HWO- back + posted
  (+ I think I fixed blackboard... )
A graph $G = (V, E)$ is an ordered pair of 2 sets:

$V = \text{vertices} = \{v_1, v_2, v_3, v_4\}$

$E = \text{edges} = \{v_1v_2, v_2v_3, v_3v_4\}$
Why? (my favorite!)
They model everything!

Examples

- social network
- roads
- connectivity
- sensor network
- communication
More defns:

\[ G \] is undirected if edges are unordered pairs

so \( \exists u, v \neq \exists v, u \)

\[ G \] is directed if edges are ordered pairs

so \( (u, v) \neq (v, u) \)
The degree of a vertex, \( d(v) \), is the number of adjacent edges.

A path \( P = v_1, \ldots, v_k \) is a set of vertices with \( \{ v_i, v_{i+1} \} \in E \) (or \( (v_i, v_{i+1}) \in E \) if directed).

A path is simple if all vertices are distinct.

A cycle is a path which is simple except \( v_i = v_k \).
Lemma: (degree sum formula)
\[ \sum_{v \in V} d(v) = 2|E| \]

Proof:
Consider 1 edge:
has 2 vertices in is connected to
\[ \Rightarrow \text{each edge contributes } +2 \text{ to sum on left side} \]
\[ \Rightarrow 2|E| \]

Why?
Size of G: 2 parameters:

\[ |V| = n \]
\[ |E| = m \]

How big can \( m \) be in terms of \( n \)?

# edges in a graph with \( n \) vertices:

\[ m \leq n \frac{(n-1)}{2} = \binom{n}{2} \]

\( K_n \) = all edges

trees: acyclic graph, connected with how many edges?

\[ m = n - 1 \]
Representing graphs

How do we make this data structure?

- arrays or lists
- matrix

more options
Adjacency (or vertex) lists:

\[ \begin{align*}
V_1 & : V_2, V_5 \\
V_2 & : V_1, V_3, V_5 \\
V_3 & : V_2, V_3, V_5 \\
V_4 & : \text{-----} \\
V_5 & : \text{-----}
\end{align*} \]

Size: \( \propto O(n+m) \)

Lookup: Time to check if \( v_i + v_j \) are nbsrs: \( O(n) \)
Implementation:

More buried data structures!

Could use:

- linked
- array

Issues w/ insertion, sorting, …
Adjacency Matrix

\[\begin{array}{ccccc}
& v_1 & v_2 & v_3 & v_4 & v_5 \\
v_1 & 0 & 1 & 0 & 0 & 1 \\
v_2 & 1 & 0 & 1 & 1 & 0 \\
v_3 & 0 & 1 & 0 & 0 & 0 \\
v_4 & 0 & 1 & 1 & 0 & 0 \\
v_5 & 1 & 1 & 0 & 0 & 0 \\
\end{array}\]

directed: use whole matrix

space: \(O(n^2)\)

check nbr: \(O(1)\)
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><strong>Space</strong></td>
<td>$\Theta(V^2)$</td>
<td>$\Theta(V + E)$</td>
<td>$\Theta(V + E)$</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>$\Theta(V^2)$</td>
<td>$\Theta(V + E)$</td>
<td>$\Theta(V + E)$</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>
**Definition:**
- **G is connected if** $\forall u, v$, **there exists a path from** $u$ **to** $v$.
- **The distance from** $u$ **to** $v$, $d(u, v)$, **is equal to the number of edges on the minimum** $u, v$-**path**.

*Graphs are unweighted*

$d(u, v) = 1$  
$d(v, v') = \infty$

$d(x, y) = 2$
Algorithms on graphs

Basic 1st question:
Given any 2 vertices, are they connected?
Also: What is their distance?

How to solve?

BFS
DFS
Suggestion:
Suppose we're in a maze searching for something. What do you do?

Depth FS:
go left until revisit
back up
+ try next leftmost

Pseudocode: two versions

**RecursiveDFS(ν):**
- if ν is unmarked
  - mark ν
- for each edge νw
  - RecursiveDFS(w)

**IterativeDFS(s):**
- Push(s) $O(1)$
- while the stack is not empty
  - $ν \leftarrow$ Pop $O(1)$
  - if $ν$ is unmarked
    - mark $ν$
    - for each edge $νw$
      - Push(w) $O(1)$

$O(m+n)$
total

Really, building a "tree":

DFS tree:
General traversal strategy's

**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

Q: Can we use a different "bag"?
BFS: use a queue

**TRAVVERSE(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 vs. DFS