#33: Graph Vocabulary + Implementation
April 8, 2019 · Wade Fagen-Ulmschneider

Motivation:
Graphs are awesome data structures that allow us to represent an enormous range of problems. To study these problems, we need:
1. A common vocabulary to talk about graphs
2. Implementation(s) of a graph
3. Traversals on graphs
4. Algorithms on graphs

Graph Vocabulary
Consider a graph \( G \) with vertices \( V \) and edges \( E \), \( G=(V,E) \).

- Incident Edges: \( I(v) = \{ (x, v) \in E \} \)
- Degree(v): \( |I| \)
- Adjacent Vertices: \( A(v) = \{ x : (x, v) \in E \} \)
- Path(\( G_2 \)): Sequence of vertices connected by edges
- Cycle(\( G_i \)): Path with a common begin and end vertex.
- Simple Graph(\( G \)): A graph with no self loops or multi-edges.
- Subgraph(\( G \)): \( G' = (V', E') \):
  \( V' \in V \), \( E' \in E \), and \( (u, v) \in E \rightarrow u \in V' \), \( v \in V' \)

Graphs that we will study this semester include:
- Complete subgraph(G)
- Connected subgraph(G)
- Connected component(G)
- Acyclic subgraph(G)
- Spanning tree(G)

Size and Running Times
Running times are often reported by \( n \), the number of vertices, but often depend on \( m \), the number of edges.

For arbitrary graphs, the minimum number of edges given a graph that is:

- Not Connected:
  - Minimally Connected*:

The maximum number of edges given a graph that is:

- Simple:

- Not Simple:

The relationship between the degree of the graph and the edges:

---

Proving the Size of a Minimally Connected Graph

**Theorem:** Every minimally connected graph \( G=(V, E) \) has \( |V|-1 \) edges.

**Proof of Theorem**
Consider an arbitrary, minimally connected graph \( G=(V, E) \).

**Lemma 1:** Every connected subgraph of \( G \) is minimally connected.
*(Straightforward proof by contradiction left for you; remember that graph \( G \) is a minimally connected graph in this problem.)*
Suppose \(|V| = 1\):

**Definition:**

**Theorem:**

**Inductive Hypothesis:** For any \(j < |V|\), any minimally connected graph of \(j\) vertices has \(j-1\) edges.

Suppose \(|V| > 1\):

1. Choose any vertex: 

2. Partitions:

3. Count the edges:

|\(E_G| =

...by application of our IH and Lemma #1, every component \(C_k\) is a minimally connected subgraph of \(G\)...

|\(E_G| =

### Graph ADT

<table>
<thead>
<tr>
<th>Data</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Vertices</td>
<td><code>insertVertex(K key)</code>;</td>
</tr>
<tr>
<td>2. Edges</td>
<td><code>insertEdge(Vertex v1, Vertex v2, K key)</code>;</td>
</tr>
<tr>
<td>3. Some data structure maintaining the structure between vertices and edges.</td>
<td><code>removeVertex(Vertex v)</code>;</td>
</tr>
<tr>
<td></td>
<td><code>removeEdge(Vertex v1, Vertex v2)</code>;</td>
</tr>
<tr>
<td></td>
<td><code>incidentEdges(Vertex v1, Vertex v2)</code>;</td>
</tr>
<tr>
<td></td>
<td><code>areAdjacent(Vertex v1, Vertex v2)</code>;</td>
</tr>
<tr>
<td></td>
<td><code>origin(Edge e)</code>;</td>
</tr>
<tr>
<td></td>
<td><code>destination(Edge e)</code>;</td>
</tr>
</tbody>
</table>

### Graph Implementation #1: Edge List

<table>
<thead>
<tr>
<th>Vert. Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
</tr>
<tr>
<td>v</td>
</tr>
<tr>
<td>w</td>
</tr>
<tr>
<td>z</td>
</tr>
</tbody>
</table>

### Operations:

- `insertVertex(K key)`;
- `removeVertex(Vertex v)`;
- `areAdjacent(Vertex v1, Vertex v2)`;
- `incidentEdges(Vertex v)`;

### CS 225 – Things To Be Doing:

1. mp_mazes EC+7 due tonight; final due date on Monday, Apr. 15
2. lab_dict released this week; due on Sunday, Apr. 14
3. Final programming exam next week (Apr. 18 – Apr. 21)
4. Daily POTDs are ongoing!