A Review of Major Data Structures So Far

<table>
<thead>
<tr>
<th>Array-based</th>
<th>List/Pointer-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Sorted Array</td>
<td>- Singly Linked List</td>
</tr>
<tr>
<td>- Unsorted Array</td>
<td>- Doubly Linked List</td>
</tr>
<tr>
<td>- Stacks</td>
<td>- Skip Lists</td>
</tr>
<tr>
<td>- Queues</td>
<td>- Trees</td>
</tr>
<tr>
<td>- Hashing</td>
<td>- BTree</td>
</tr>
<tr>
<td>- Heaps</td>
<td>- Binary Tree</td>
</tr>
<tr>
<td>- Priority Queues</td>
<td>- Huffman Encoding</td>
</tr>
<tr>
<td>- UpTrees</td>
<td>- kd-Tree</td>
</tr>
<tr>
<td>- Disjoint Sets</td>
<td>- AVL Tree</td>
</tr>
</tbody>
</table>

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($G_3$)**: Sequence of vertices connected by edges

- **Cycle($G_1$)**: Path with a common begin and end vertex.

- **Simple Graph($G$)**: A graph with no self loops or multi-edges.

Subgraph($G$): $G' = (V', E')$:
$$V' \in V, E' \in E, \text{ and } (u, v) \in E \Rightarrow u \in V', v \in V'$$

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**:

The relationship between the degree of the graph and the edges:
Proving the Size of a Minimally Connected Graph

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

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

Suppose \(|V| = 1:\)
Definition:

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

Suppose \(|V| > 1:\)
1. Choose any vertex:
2. Partitions:
   - \( C_0 := \)
   - \( C_k, k = [1...d] := \)
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>\texttt{insertVertex(K key);}</td>
</tr>
<tr>
<td>2. Edges</td>
<td>\texttt{insertEdge(Vertex v1, Vertex v2, K key);}</td>
</tr>
<tr>
<td>3. Some data structure maintaining the structure between vertices and edges.</td>
<td>\texttt{removeVertex(Vertex v);} \texttt{removeEdge(Vertex v1, Vertex v2);} \texttt{incidentEdges(Vertex v);} \texttt{areAdjacent(Vertex v1, Vertex v2);} \texttt{origin(Edge e);} \texttt{destination(Edge e);}</td>
</tr>
</tbody>
</table>

Graph Implementation #1: Edge List

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

Operations:
- \texttt{insertVertex(K key):}
- \texttt{removeVertex(Vertex v):}
- \texttt{areAdjacent(Vertex v1, Vertex v2):}
- \texttt{incidentEdges(Vertex v):}

CS 225 – Things To Be Doing:

1. \texttt{mp_traversal} due today.
2. Daily POTDs are ongoing!