#35: Adjacency List + BFS
November 16, 2018 · Wade Fagen-Ulmschneider

## Graph Implementation #3: Adjacency List

![Graph Diagram]

### Vertex List

<table>
<thead>
<tr>
<th>Vertex</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 on an Adjacency Matrix implementation:

- **insertVertex(K key):**

- **removeVertex(Vertex v):**

- **incidentEdges(Vertex v):**

- **areAdjacent(Vertex v1, Vertex v2):**

- **insertEdge(Vertex v1, Vertex v2, K key):**

### Running Times of Classical Graph Implementations

<table>
<thead>
<tr>
<th></th>
<th>Edge List</th>
<th>Adj. Matrix</th>
<th>Adj. List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space</strong></td>
<td>n+m</td>
<td>n²</td>
<td>n+m</td>
</tr>
<tr>
<td><strong>insertVertex</strong></td>
<td>1</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td><strong>removeVertex</strong></td>
<td>m</td>
<td>n</td>
<td>deg(v)</td>
</tr>
<tr>
<td><strong>insertEdge</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>removeEdge</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>incidentEdges</strong></td>
<td>m</td>
<td>n</td>
<td>deg(v)</td>
</tr>
<tr>
<td><strong>areAdjacent</strong></td>
<td>m</td>
<td>1</td>
<td>min( deg(v), deg(w) )</td>
</tr>
</tbody>
</table>

### Big Picture Ideas: Comparing Implementations

**Q:** If we consider implementations of simple, connected graphs, what relationship between \( n \) and \( m \)?

- On connected graphs, is there one algorithm that underperforms the other two implementations?

...what if our graph is sparse and not connected?

**Q:** Is there clearly a single best implementation?

- Optimized for fast construction:

- Optimized for areAdjacent operations:
Graph Traversal

Objective: Visit every vertex and every edge in the graph.
Purpose: Search for interesting sub-structures in the graph.

We've seen traversal before – this is different:

<table>
<thead>
<tr>
<th>BST</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="BST Diagram" /></td>
<td><img src="image2.png" alt="Graph Diagram" /></td>
</tr>
</tbody>
</table>

BFS Graph Traversal:

```
Pseudocode for BFS
1  BFS(G):
2      Input: Graph, G
3      Output: A labeling of the edges on G as discovery and cross edges
4      foreach (Vertex v : G.vertices()):
5          setLabel(v, UNEXPLORED)
6      foreach (Edge e : G.edges()):
7          setLabel(e, UNEXPLORED)
8      foreach (Vertex v : G.vertices()):
9          if getLabel(v) == UNEXPLORED:
10             BFS(G, v)
11  BFS(G, v):
12      Queue q
13      setLabel(v, VISITED)
14      q.enqueue(v)
15      while !q.empty():
16          v = q.dequeue()
17          foreach (Vertex w : G.adjacent(v)):
18              if getLabel(w) == UNEXPLORED:
19                  setLabel(v, w, DISCOVERY)
20                  setLabel(w, VISITED)
21                  q.enqueue(w)
22              elseif getLabel(v, w) == UNEXPLORED:
23                  setLabel(v, w, CROSS)
```

BST Graph Observations

1. Does our implementation handle disjoint graphs? How?
   a. How can we modify our code to count components?

2. Can our implementation detect a cycle? How?

---

CS 225 – Things To Be Doing:

1. Programming Exam C is different than usual schedule:
   Exam: Sunday, Dec 2 – Tuesday, Dec 4
2. lab_dict on-going; due on Tuesday, Nov. 27
3. MP6 EC+3 due tonight; final due date on Monday, Nov. 26
4. No POTDs over break (next one after today is Monday, Nov. 26)