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>

Implementations and Use Cases

Ex. 1:

Ex. 2:

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>
</table>

BFS Graph Traversal

Ex. 2:

**Pseudocode for BFS**

```python
# Pseudocode for BFS

BFS(G):
    Input: Graph, G
    Output: A labeling of the edges on G as discovery and cross edges
    foreach (Vertex v : G.vertices()):
        setLabel(v, UNEXPLORED)
    foreach (Edge e : G.edges()):
        setLabel(e, UNEXPLORED)
    foreach (Vertex v : G.vertices()):
        if getLabel(v) == UNEXPLORED:
            BFS(G, v)

BFS(G, v):
    Queue q
    setLabel(v, VISITED)
    q.enqueue(v)
    while !q.empty():
        v = q.dequeue()
        foreach (Vertex w : G.adjacent(v)):
            if getLabel(w) == UNEXPLORED:
                setLabel(w, DISCOVERY)
                setLabel(v, w, VISITED)
                q.enqueue(w)
            elseif getLabel(v, w) == UNEXPLORED:
                setLabel(v, w, CROSS)
```

A
B
C
D
E
F
G
H
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?
   a. How can we modify our code to store update a private member variable cycleDetected_?

3. What is the running time of our algorithm?

4. What is the shortest path between A and H?

5. What is the shortest path between E and H?
   a. What does that tell us about BFS?

6. What does a cross edge tell us about its endpoints?

7. What structure is made from discovery edges in G?

Big Ideas: Utility of a BFS Traversal

Obs. 1: Traversals can be used to count components.
Obs. 2: Traversals can be used to detect cycles.
Obs. 3: In BFS, d provides the shortest distance to every vertex.
Obs. 4: In BFS, the endpoints of a cross edge never differ in distance, d, by more than 1: |d(u) - d(v)| = 1

Depth First Search – A Modification to BFS

Two types of edges:

1. 

2. 

Running Time of DFS:

Labeling:
- Vertex:
- Edge:

Queries:
- Vertex:
- Edge:

CS 225 – Things To Be Doing:

1. Programming Exam C starts Tuesday 4/17
2. MP6 due tonight, Monday, April 16th
3. lab_graphs available Wednesday
4. Daily POTDs are ongoing!