CS 225
Data Structures

November 8 – Graph Implementation and Traversals
G Carl Evans
To study all of these structures:
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms
Key Ideas:
- Given a vertex, $O(1)$ lookup in vertex list
  - Implement with a hash table, etc
- All basic ADT operations run in $O(m)$ time
Adjacency Matrix

Key Ideas:
- Given a vertex, $O(1)$ lookup in vertex list
- Given a pair of vertices (an edge), $O(1)$ lookup in the matrix
- Undirected graphs can use an upper triangular matrix
Adjacency List
Adjacency List

insertVertex(K key):
Adjacency List

removeVertex(Vertex v):

- u
  - v: d=2
  - w: d=3
- v
  - d=2
- w
  - z: d=1
- z

Diagram:

- vertices: u, v, w, z
- edges: a→u, b→u, c→u, d→u, d→w, w→z, z→d
- vertex degrees: d=2, d=3, d=1
Adjacency List

incidentEdges(Vertex v):

- incidentEdges(u):
  - a
  - c

- incidentEdges(v):
  - a
  - b

- incidentEdges(w):
  - b
  - c
  - d

- incidentEdges(z):
  - d

Vertex labels with their respective degrees:
- u (d=2)
- v (d=2)
- w (d=3)
- z (d=1)
Adjacency List

areAdjacent(Vertex v1, Vertex v2):
Adjacency List

insertEdge(Vertex v1, Vertex v2, K key):
<table>
<thead>
<tr>
<th>Expressed as O(f)</th>
<th>Edge List</th>
<th>Adjacency Matrix</th>
<th>Adjacency List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>n+m</td>
<td>n^2</td>
<td>n+m</td>
</tr>
<tr>
<td>insertVertex(v)</td>
<td>1</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>removeVertex(v)</td>
<td>m</td>
<td>n</td>
<td>deg(v)</td>
</tr>
<tr>
<td>insertEdge(v, w, k)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>removeEdge(v, w)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>incidentEdges(v)</td>
<td>m</td>
<td>n</td>
<td>deg(v)</td>
</tr>
<tr>
<td>areAdjacent(v, w)</td>
<td>m</td>
<td>1</td>
<td>min( deg(v), deg(w) )</td>
</tr>
</tbody>
</table>
**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 ....but it’s different:

- Ordered
- Obvious Start
- 

- 
- 
-
Traversal: BFS
Traversing a graph using BFS:

<table>
<thead>
<tr>
<th>v</th>
<th>d</th>
<th>P</th>
<th>Adjacent Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Traversal: BFS

- **Navigation Order:**
  - Level 0: A
  - Level 1: A, B, C
  - Level 2: A, C
  - Level 3: E
  - Level 4: G

- **Adjacent Edges Table:**

<table>
<thead>
<tr>
<th>d</th>
<th>p</th>
<th>Adjacent Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>A, C, B, D</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>A, B, C, E</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>A, C, B, A, D, E, F</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>A, D, A, C, F, H</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>C, E, B, C, G</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>C, F, C, D, G</td>
</tr>
<tr>
<td>3</td>
<td>E</td>
<td>E, G, E, F, H</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>D, H, D, G</td>
</tr>
</tbody>
</table>

- **Diagram:**

```
G - F - E - D - B - C - A
```

- **Traversal Order:**

```
A, B, C, D, E, F, G, H
```
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(v, w, DISCOVERY)
        setLabel(w, VISITED)
        q.enqueue(w)
      elseif getLabel(v, w) == UNEXPLORED:
        setLabel(v, w, CROSS)
BFS Analysis

Q: Does our implementation handle disjoint graphs? If so, what code handles this?
   • *How do we use this to count components?*

Q: Does our implementation detect a cycle?
   • *How do we update our code to detect a cycle?*

Q: What is the running time?
Running time of BFS

While-loop at :19?

For-loop at :21?
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(v, w, DISCOVERY)
        setLabel(w, VISITED)
        q.enqueue(w)
      elseif getLabel(v, w) == UNEXPLORED:
        setLabel(v, w, CROSS)
BFS Observations

Q: What is a shortest path from A to H?

Q: What is a shortest path from E to H?

Q: How does a cross edge relate to d?

Q: What structure is made from discovery edges?

<table>
<thead>
<tr>
<th>d</th>
<th>p</th>
<th>v</th>
<th>Adjacent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>A</td>
<td>C B D</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>A C E</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>C</td>
<td>B A D E F</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>D</td>
<td>A C F H</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>E</td>
<td>B C G</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>F</td>
<td>C D G</td>
</tr>
<tr>
<td>3</td>
<td>E</td>
<td>G</td>
<td>E F H</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>H</td>
<td>D G</td>
</tr>
</tbody>
</table>
BFS Observations

**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:

$$|d(u) - d(v)| = 1$$
Traversal: DFS
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(v, w, DISCOVERY)
        setLabel(w, VISITED)
        q.enqueue(w)
      elseif getLabel(v, w) == UNEXPLORED:
        setLabel(v, w, CROSS)
DFS(G):
  Input: Graph, G
  Output: A labeling of the edges on G as discovery and back 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:
      DFS(G, v)

DFS(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(v, w, DISCOVERY)
        setLabel(w, VISITED)
        DFS(G, w)
      elseif getLabel(v, w) == UNEXPLORED:
        setLabel(v, w, BACK)
Running time of DFS

Labeling:
- Vertex:
- Edge:

Queries:
- Vertex:
- Edge: