March 29 – Graph Implementations
G Carl Evans
Project Proposal – Problems

• Unclear and incomplete deliverables

• Lacking required details
Project Proposal - Deliverables

• It must be clear what each deliverable is and all the related properties with that deliverable. That is the academic reference for a given deliverable must be clear not a list of papers at the end.

• Deliverables must be significant and independent
  • Dijkstra’s and A* are both fine but Dijkstra’s is a subset of A* so you cannot do both as two different deliverables.
Project Proposals - Details

• Must explicitly describe how any data maps to the values your algorithm needs

  ✓ If the algorithm works on a graph must say what are the nodes, what are the edges, and if it has weights what are the weights. This must come from something specific not random.

  ✓ If the algorithm uses a heuristic what that heuristic is and why it makes sense

  ✓ What language anything is in if not in C++
Proposal Extension

Proposal Question reopened until end of the day on Friday March 31st.
Graphs

To study all of these structures:
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms
Graph Implementation: Edge List

![Graph Diagram]

- Vertices: 0, 1, 2, 3
- Edges: (0, 1), (1, 2), (0, 2), (2, 3)
- Labels: a, b, c, d
Graph Implementation: Adjacency Matrix
Graph Implementation: Edge List + ?
Graph Implementation: Adjacency List
Graph Implementation: Adjacency List
Adjacency List

insertVertex(K key):

```
  u  v  a
  u  w  b
  u  w  c
  w  z  d
```
Adjacency List

removeVertex(Vertex v):

U = \{u, v, w, z\}
D = \{a, b, c, d\}

E = \{(u, v), (v, w), (w, z), (u, w), (w, c), (c, d)\}
N = \{(u, a), (u, c), (v, a), (v, b), (w, b), (w, c), (w, d), (z, d)\}

T = \{(u, v, a), (v, w, b), (w, z, d)\}
Adjacency List

incidentEdges(Vertex v):
Adjacency List

areAdjacent(Vertex v1, Vertex v2):
Adjacency List

insertEdge(Vertex v1, Vertex v2, K key):
BFS(G):
1. Input: Graph, G
2. Output: A labeling of the edges on G as discovery and cross edges
3. foreach (Vertex v : G.vertices()):
4.     setLabel(v, UNEXPLORED)
5. foreach (Edge e : G.edges():)
6.     setLabel(e, UNEXPLORED)
7. foreach (Vertex v : G.vertices()):
8.     if getLabel(v) == UNEXPLORED:
9.         BFS(G, v)

BFS(G, v):
10. Queue q
11. setLabel(v, VISITED)
12. q.enqueue(v)
13. while !q.empty():
14.     v = q.dequeue()
15.     foreach (Vertex w : G.adjacent(v)):
16.         if getLabel(w) == UNEXPLORED:
17.             setLabel(v, w, DISCOVERY)
18.             setLabel(w, VISITED)
19.             q.enqueue(w)
20.     elseif getLabel(v, w) == UNEXPLORED:
21.         setLabel(v, w, CROSS)
Traversal: 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>
```
Traversals: 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>0</td>
<td>-</td>
<td>C B D</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>A C E</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>B A D E F</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>A C F H</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td>B C G</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td>C D G</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td>E F H</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td>D G</td>
</tr>
</tbody>
</table>

Diagram of the graph with vertices labeled A, B, C, D, E, F, G, and H, and edges connecting these vertices.
Traversals: 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>0</td>
<td></td>
<td>C B D</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>A</td>
<td>A C E</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>A</td>
<td>B A D E F</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>A</td>
<td>A C F H</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>C</td>
<td>B C G</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>C</td>
<td>C D G</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>E</td>
<td>E F H</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>D</td>
<td>D G</td>
</tr>
</tbody>
</table>
```
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?
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)
Running time of BFS

While-loop at :19?

For-loop at :21?
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>v</th>
<th>d</th>
<th>P</th>
<th>Adjacent Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>-</td>
<td>C B D</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>A</td>
<td>A C E</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>A</td>
<td>B A D E F</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>A</td>
<td>A C F H</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>C</td>
<td>B C G</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>C</td>
<td>C D G</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>E</td>
<td>E F H</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>D</td>
<td>D G</td>
</tr>
</tbody>
</table>

Diagram:

```
A
 /|
/ \
B - C - D
 \
 \
E - F
 \
G - H
```
BFS Observations

**Obs. 1:** BFS can be used to count components.

**Obs. 2:** BFS 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)
Traversal: DFS
Traversal: DFS

- Discovery Edge
- Back Edge
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

def 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: