Graphs

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

```
0 1 a
1 2 b
0 2 c
2 3 d
```
Graph Implementation: Adjacency Matrix

```
\begin{array}{c c c c}
0 & 1 & a & \\
1 & 2 & b & \\
0 & 2 & c & \\
2 & 3 & d & \\
\end{array}
```

```
\begin{array}{c c c c}
0 & 1 & 2 & 3 \\
0 & 3 & 0 & 0 \\
1 & 0 & 0 & 0 \\
2 & - & - & 0 \\
3 & - & - & - \\
\end{array}
```
Graph Implementation: Edge List + ?
Graph Implementation: Adjacency List

- Vertices: u, v, w, z
- Edges: (u, a), (u, c), (v, a), (v, b), (w, b), (w, c), (w, d), (u, w), (v, w), (u, w), (w, z), (w, d)
- Degrees: d(u) = 2, d(v) = 2, d(w) = 3, d(z) = 1
Graph Implementation: Adjacency List
Adjacency List

```
insertVertex(K key):
```

![Diagram illustrating Adjacency List with vertices and edges labelled with distances]
Adjacency List

removeVertex(Vertex v):

- Adjacency List representation of a graph with vertices v, w, u, and z, and edges between them.
  - Edge labels indicate the distance (d) between the vertices.
Adjacency List

incidentEdges(Vertex v):

- For vertex u, incident edges are: u, v, a.
- For vertex v, incident edges are: v, w, b.
- For vertex w, incident edges are: w, c, d.
- For vertex z, incident edges are: z, d.
Adjacency List

areAdjacent(Vertex v1, Vertex v2):

Vertex u is adjacent to:
- v
- c

Vertex v is adjacent to:
- a
- b

Vertex w is adjacent to:
- b
- c
- d

Vertex z is adjacent to:
- c
- d

u has degree 2
v has degree 2
w has degree 3
z has degree 1
Adjacency List

\textbf{insertEdge}(Vertex v1, Vertex v2, K key):
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)
### Traversal: BFS

#### Adjacent 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></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

A

B

C

D

E

F

G

H

<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>A C E</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>B A D E F</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>A C F H</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>B C G</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>C D G</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>E F H</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>D G</td>
<td></td>
</tr>
</tbody>
</table>
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>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>

G H F E D B C A
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?

<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 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?
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):

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)
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

    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: