



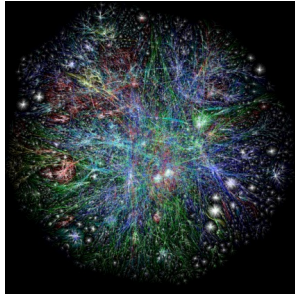
CS 225

Data Structures

March 25 – Graph Traversals

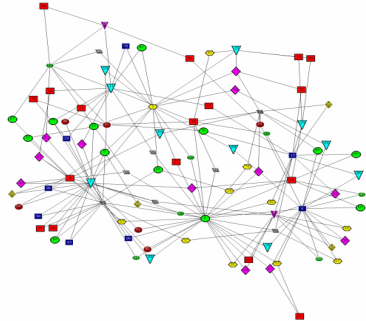
G Carl Evans

Graphs

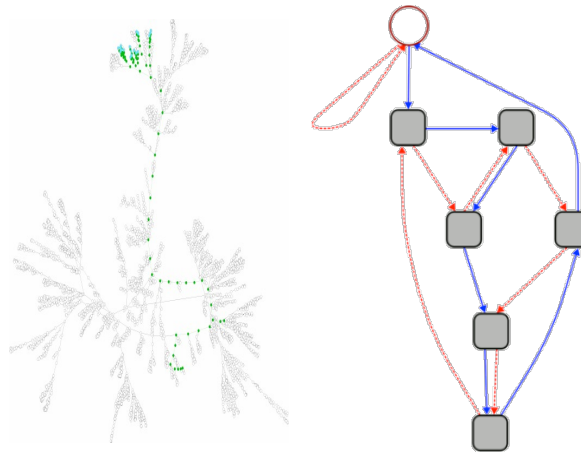
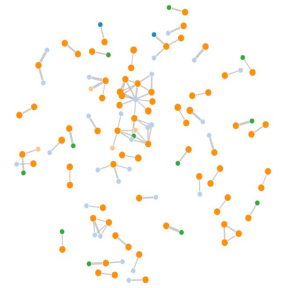
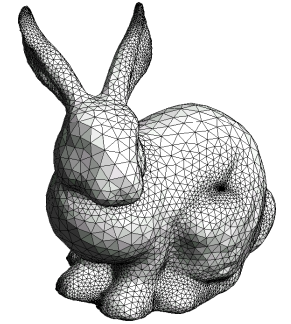


HAMLET

TROILUS AND CRESSIDA



- To study all of these structures:**
1. A common vocabulary
 2. Graph implementations
 3. Graph traversals
 4. Graph algorithms



```
heapify(int*, unsigned int):  
  push rbp  
  mov rsi, rbp  
  sub rbp, 20  
  mov dword ptr [rbp + 8], rsi  
  mov dword ptr [rbp + 12], esi  
  mov dword ptr [rbp + 16], 1  
  jmp .LBB_4
```

```
heapify(int*, unsigned int):  
  mov rax, qword ptr [rbp - 8]  
  mov ecx, dword ptr [rbp - 12]  
  mov ebx, ecx  
  mov rax, qword ptr [rax + 4*ebx]  
  mov rax, qword ptr [rbp - 8]  
  mov esi, dword ptr [rbp - 12]  
  shr esi, 1  
  mov ebx, esi  
  mov ecx, qword ptr [rax + 4*ebx]  
  jmp .LBB_3
```

```
heapify(int*, unsigned int):  
  mov rax, qword ptr [rbp - 8]  
  mov rax, qword ptr [rbp - 8]  
  mov ecx, qword ptr [rbp - 12]  
  mov ecx, qword ptr [rax + 4*ecx]  
  mov rax, qword ptr [rbp - 8]  
  mov esi, dword ptr [rbp - 12]  
  shr esi, 1  
  mov ebx, esi  
  mov ecx, qword ptr [rax + 4*ebx]  
  jmp .LBB_3
```

```
.LBB_3:  
  jmp .LBB_1
```

```
.LBB_4:  
  add rbp, 20  
  pop rbp  
  ret
```

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

```
14 BFS(G, v) :
15   Queue q
16   q.enqueue(v)
17
18   while !q.empty()
19     [v,p] = q.dequeue()
20     if( getLabel(v) == UNEXPLORED)
21       setLabel(v, VISITED)
22       setLabel(p, DISCOVERY)
23       foreach (Vertex w : G.adjacent(v)):
24         q.enqueue(w,v)
25     else
26       setLabel(p, CROSS)
27
```



BFS Analysis

Q: Does our implementation handle disjoint graphs?
If so, what code handles this?

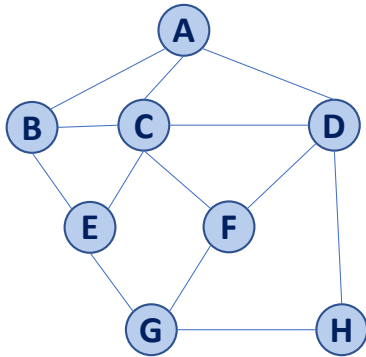
- *How do we use this to count components?*

Q: Does our implementation detect a cycle?

- *How do we update our code to detect a cycle?*

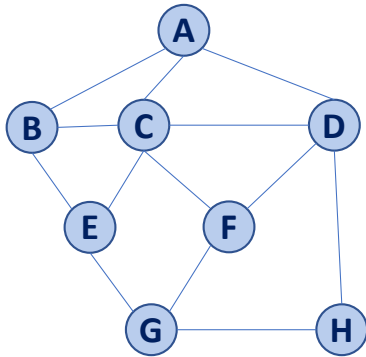
Q: What is the running time?

Traversal: BFS



v	d	P	Adjacent Edges
A			
B			
C			
D			
E			
F			
G			
H			

Traversal: BFS



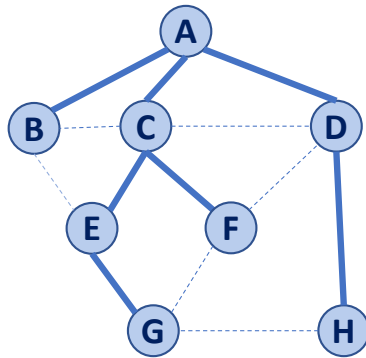
v	d	P	Adjacent Edges
A	0	-	C B D
B			A C E
C			B A D E F
D			A C F H
E			B C G
F			C D G
G			E F H
H			D G

A

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

```
14 BFS(G, v) :
15   Queue q
16   q.enqueue(v)
17
18   while !q.empty()
19     [v,p] = q.dequeue()
20     if( getLabel(v) == UNEXPLORED)
21       setLabel(v, VISITED)
22       setLabel(p, DISCOVERY)
23       foreach (Vertex w : G.adjacent(v)):
24         q.enqueue(w,v)
25     else
26       setLabel(p, CROSS)
27
```

Running time of BFS



While-loop at **:18?**

For-loop at **:23?**

v	d	P	Adjacent Edges
A	0	-	C B D
B	1	A	A C E
C	1	A	B A D E F
D	1	A	A C F H
E	2	C	B C G
F	2	C	C D G
G	3	E	E F H
H	2	D	D G



BFS Observations

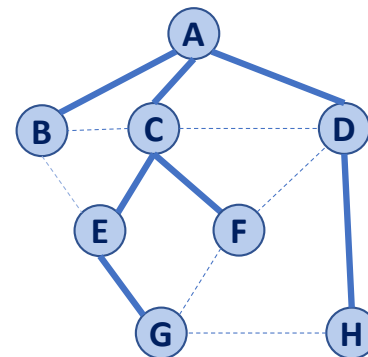
Q: What is a shortest path from **A** to **H**?

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?

v	d	P	Adjacent Edges
A	0	-	C B D
B	1	A	A C E
C	1	A	B A D E F
D	1	A	A C F H
E	2	C	B C G
F	2	C	C D G
G	3	E	E F H
H	2	D	D G





BFS Observations

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

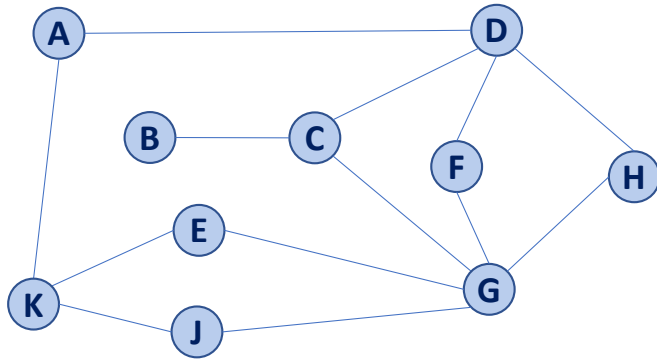
Obs. 2: BFS can be used to detect cycles.

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



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

```
14 BFS(G, v) :
15   Queue q
16   q.enqueue(v)
17
18   while !q.empty()
19       [v,p] = q.dequeue()
20       if( getLabel(v) == UNEXPLORED)
21           setLabel(v, VISITED)
22           setLabel(p, DISCOVERY)
23           foreach (Vertex w : G.adjacent(v)):
24               q.enqueue(w,v)
25       else
26           setLabel(p, CROSS)
27
```

```
1 DFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and back edges
5
6   foreach (Vertex v : G.vertices()):
7     setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9     setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11    if getLabel(v) == UNEXPLORED:
12      DFS(G, v)
```

```
14 DFS(G, v) :
15   Stack q
16   q.enqueue(v)
17
18   while !q.empty()
19     [v,p] = q.dequeue()
20     if( getLabel(v) == UNEXPLORED)
21       setLabel(v, VISITED)
22       setLabel(p, DISCOVERY)
23       foreach (Vertex w : G.adjacent(v)):
24         q.enqueue(w,v)
25     else
26       setLabel(p, BACK)
27
```

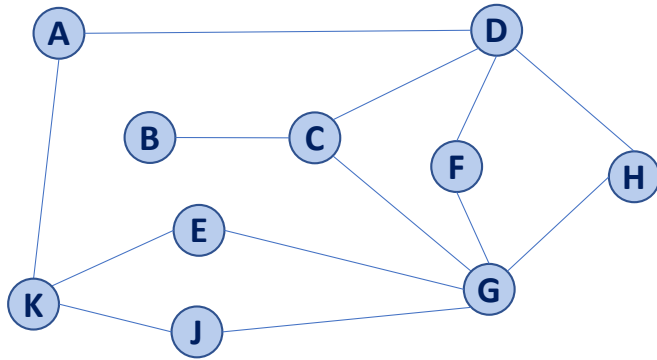
```
1 BFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and cross edges
5
6   foreach (Vertex v : G.vertices()):
7     setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9     setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11    if getLabel(v)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

```
14 BFS(G, v) :
15   Queue q
16   q.enqueue(v)
17
18   while !q.empty()
19     [v,p] = q.dequeue()
20     if( getLabel(v) == UNEXPLORED)
21       setLabel(v, VISITED)
22       setLabel(p, DISCOVERY)
23       foreach (Vertex w : G.adjacent(v)):
24         q.enqueue(w,v)
25     else
26       setLabel(p, CROSS)
27
```

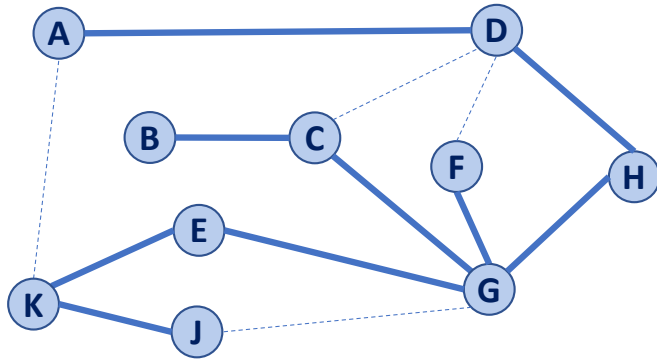
```
1 DFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and back edges
5
6   foreach (Vertex v : G.vertices()):
7     setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9     setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11    if getLabel(v) == UNEXPLORED:
12      DFS(G, v, null)
```

```
14 DFS(G, v, p) :
15 Stack q
16 q.enqueue(v)
17
18 while !q.empty()
19 [v,p] = q.dequeue()
20   if( getLabel(v) == UNEXPLORED)
21     setLabel(v, VISITED)
22     setLabel(p, DISCOVERY)
23     foreach (Vertex w : G.adjacent(v)):
24       q.enqueue(w,v) DFS(G, w, v)
25   else
26     setLabel(p, BACK)
27
```

Traversal: DFS



Traversal: DFS



————— Discovery Edge

----- Back Edge

Running time of DFS

Labeling:

- Vertex:
- Edge:

Queries:

- Vertex:
- Edge:

