



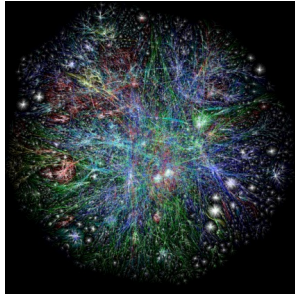
# CS 225

## Data Structures

*April 1 – Graph Implementation  
and Traversals*

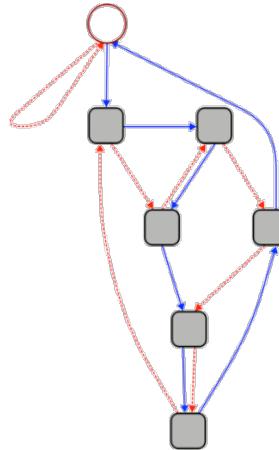
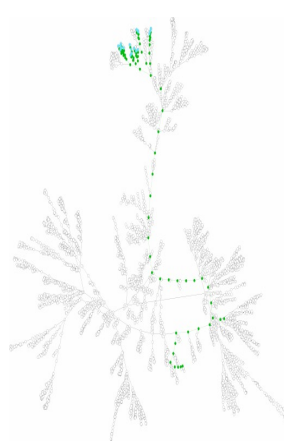
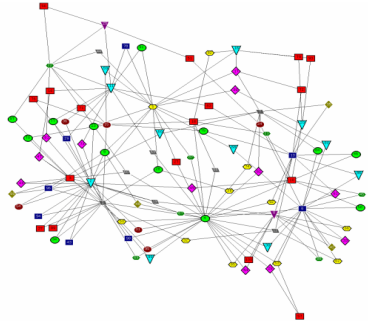
*G Carl Evans*

# Graphs



HAMLET

TROILUS AND CRESSIDA



```
heapify(int*, unsigned int);
push rfp;
mov rsi, rfp;
sub rfp, 36;
mov dword ptr [rfp + 8], rsi;
mov dword ptr [rfp + 12], esi;
mov dword ptr [rfp + 16], edi;
jmp .LBB_4
```

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

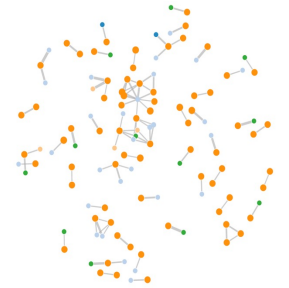
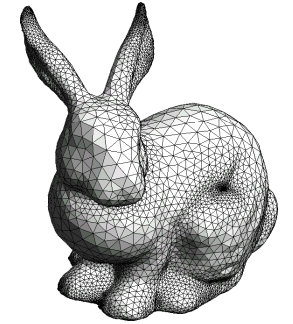
```
heapify(int*, unsigned int);
mov rax, qword ptr [rfp - 8];
mov rax, qword ptr [rfp - 8];
mov ecx, qword ptr [rfp - 12];
mov ecx, qword ptr [rax + 4*rdi];
mov ecx, qword ptr [rfp - 12];
mov ecx, qword ptr [rfp - 12];
mov ecx, qword ptr [rfp - 12];
mov ecx, 1;
mov ecx, 1;
mov ecx, qword ptr [rax + 4*rdi];
mov rsi, qword ptr [rfp - 8];
mov ecx, qword ptr [rfp - 12];
mov ecx, 1;
mov esi, ecx;
call heapify(int*, unsigned int)
```

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

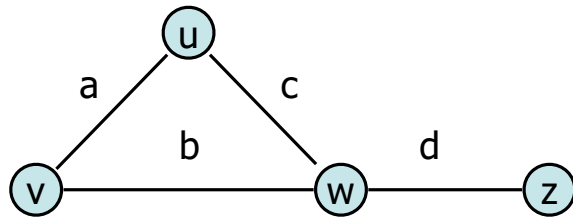
```
.LBB_4:
add rfp, 36;
pop rbp;
ret
```

To study all of these structures:

1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms



# Edge List



## Key Ideas:

- Given a vertex,  $O(1)$  lookup in vertex list
  - Implement w/ a hash table, etc
- All basic ADT operations runs in  $O(m)$  time

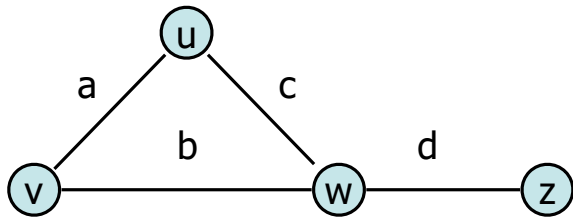
### Vertex List

u
v
w
z

### Edge List

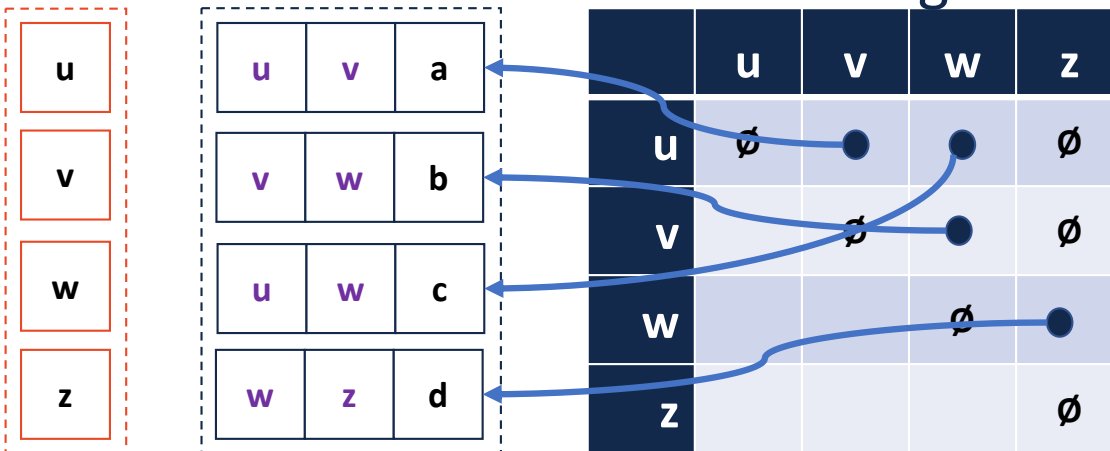
u	v	a
v	w	b
u	w	c
w	z	d

# Adjacency Matrix

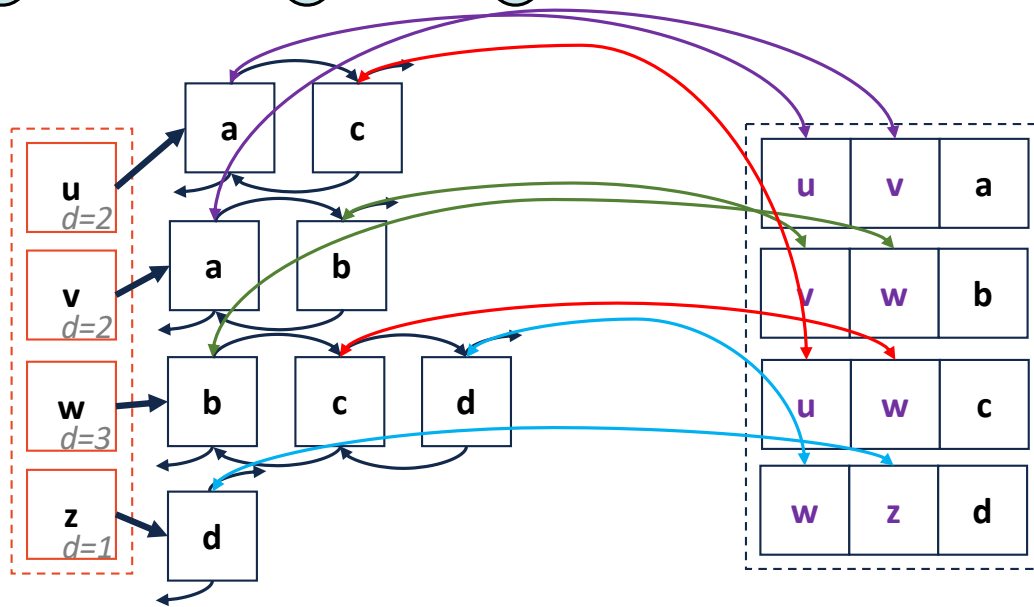
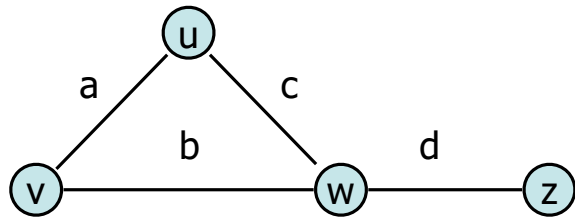


## Key Ideas:

- 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

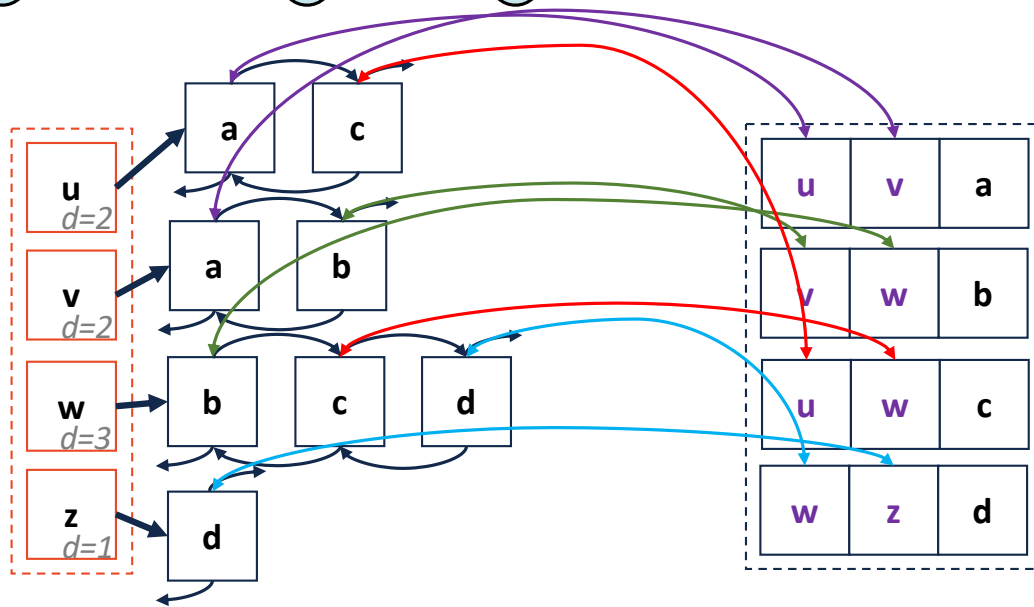
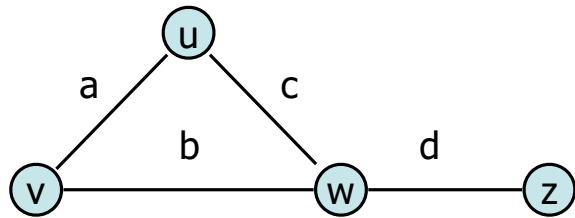


# Adjacency List



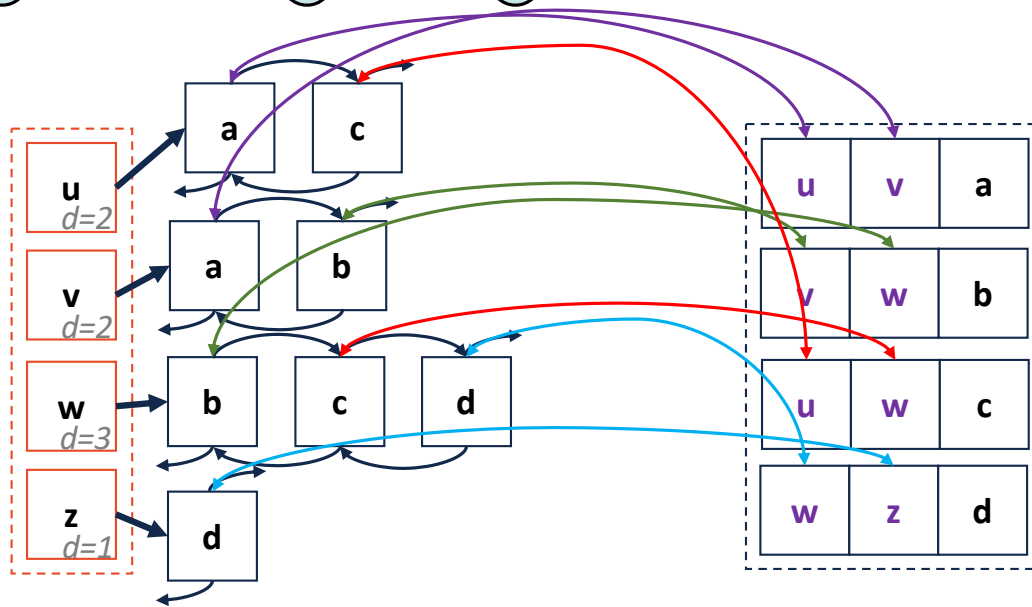
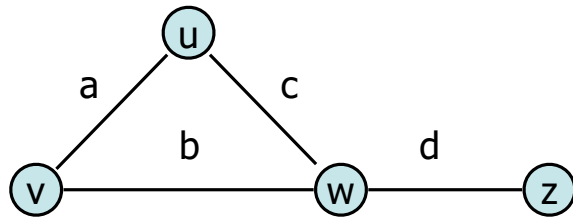
# Adjacency List

insertVertex(K key):



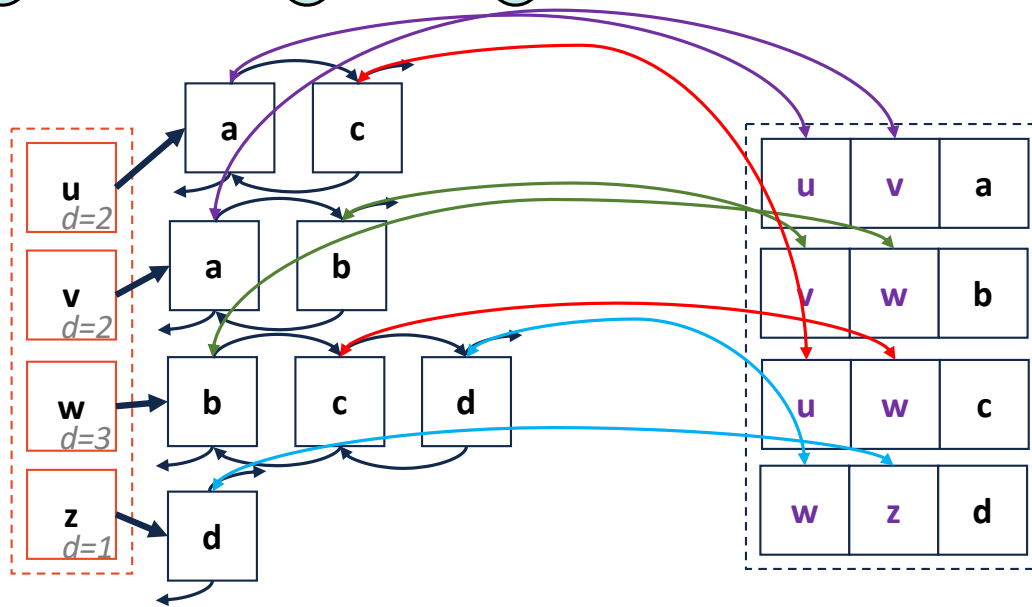
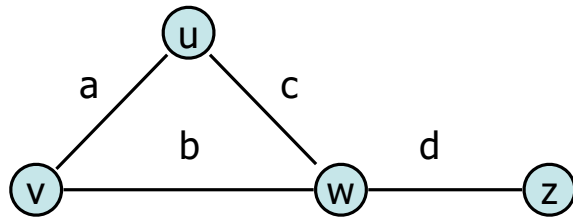
# Adjacency List

removeVertex(Vertex v):



# Adjacency List

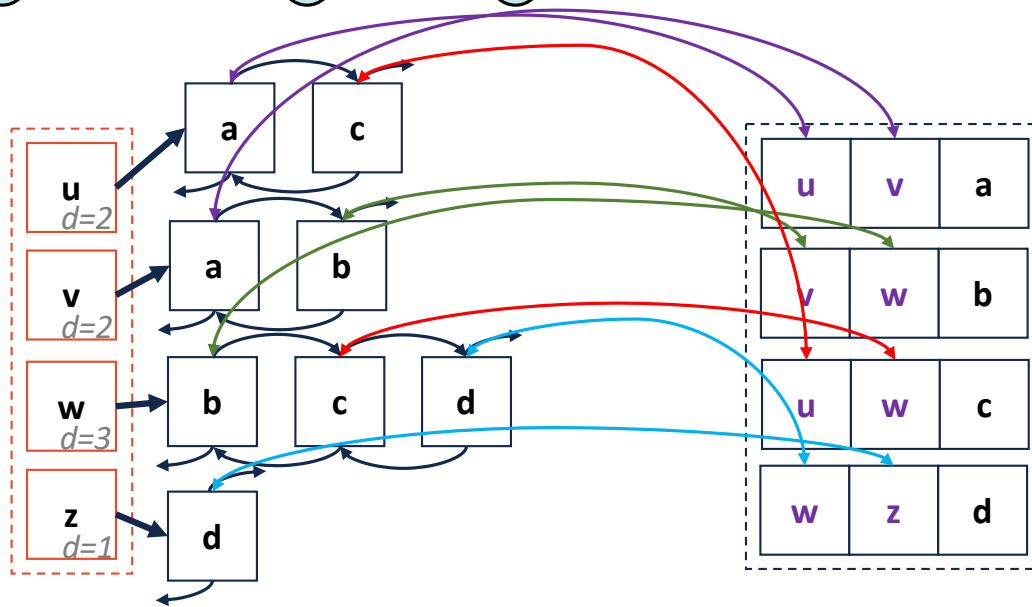
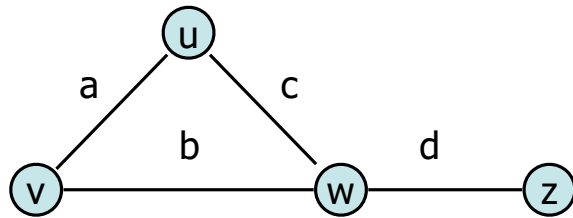
**incidentEdges(Vertex v):**





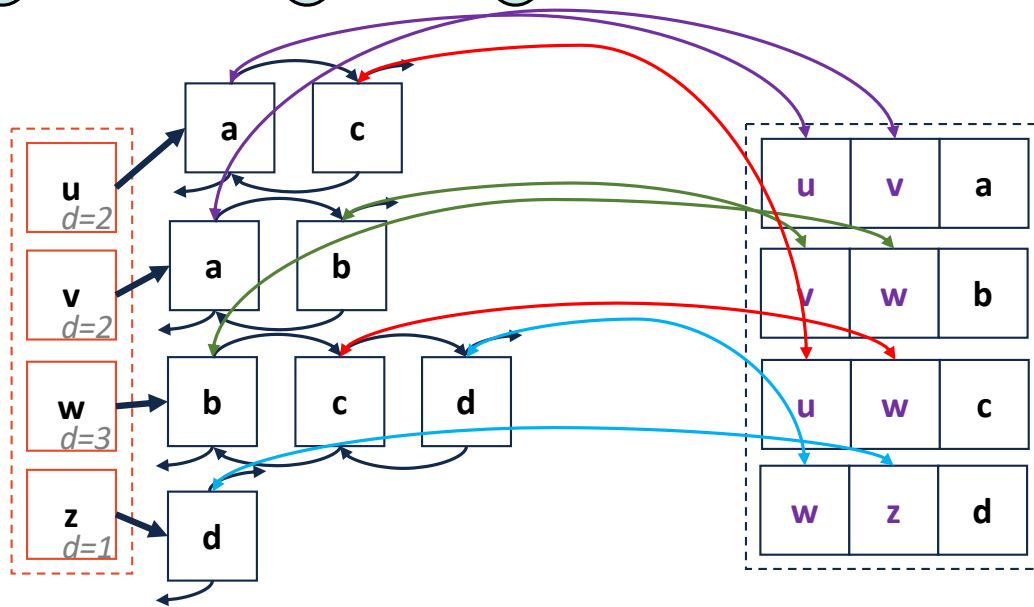
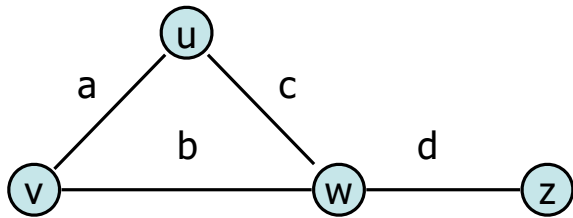
# Adjacency List

areAdjacent(Vertex v1, Vertex v2):



# Adjacency List

**insertEdge(Vertex v1, Vertex v2, K key):**



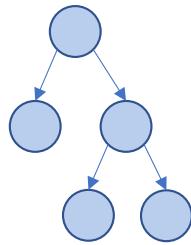
Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space	$n+m$	$n^2$	$n+m$
insertVertex(v)	1	n	1
removeVertex(v)	m	n	deg(v)
insertEdge(v, w, k)	1	1	1
removeEdge(v, w)	1	1	1
incidentEdges(v)	m	n	deg(v)
areAdjacent(v, w)	m	1	min( deg(v), deg(w) )

# Traversal:

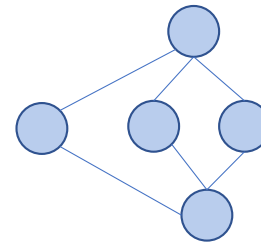
**Objective:** Visit every vertex and every edge in the graph.

**Purpose:** Search for interesting sub-structures in the graph.

We've seen traversal before ...but it's different:

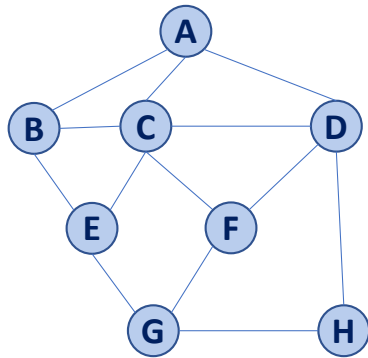


- Ordered
- Obvious Start
- 

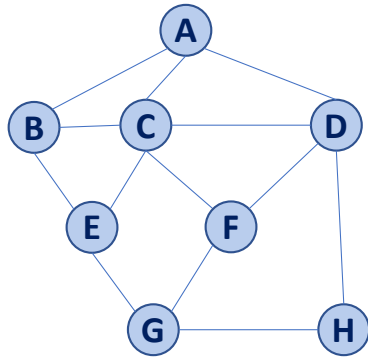


- 
- 
-

# Traversal: BFS



# Traversal: BFS

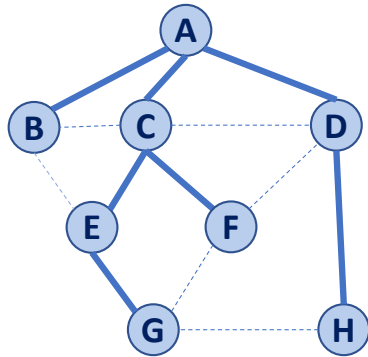


v	d	P	Adjacent Edges
A			
B			
C			
D			
E			
F			
G			
H			

---

---

# Traversal: BFS



d	p	Adjacent Edges
0	A	A CBD
1	A B	ACE
1	A C	BADEF
1	A D	ACFH
2	C E	BCG
2	C F	CDG
3	E G	EFH
2	D H	DG

~~G H F E D B C 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   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20       v = q.dequeue()
21       foreach (Vertex w : G.adjacent(v)):
22           if getLabel(w) == UNEXPLORED:
23               setLabel(v, w, DISCOVERY)
24               setLabel(w, VISITED)
25               q.enqueue(w)
26           elseif getLabel(v, w) == UNEXPLORED:
27               setLabel(v, w, CROSS)
```





## BFS Analysis

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

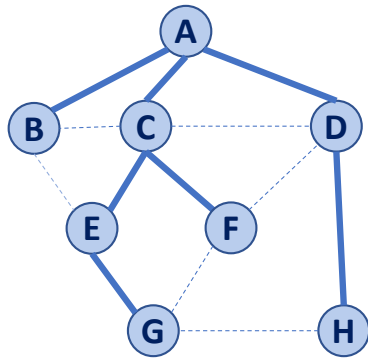
- *How do we use this to count components?*

**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?**

d	p	v	Adjacent
0	A	A	C B D
1	A	B	A C E
1	A	C	B A D E F
1	A	D	A C F H
2	C	E	B C G
2	C	F	C D G
3	E	G	E F H
2	D	H	D G



```
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   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20       v = q.dequeue()
21       foreach (Vertex w : G.adjacent(v)):
22           if getLabel(w) == UNEXPLORED:
23               setLabel(v, w, DISCOVERY)
24               setLabel(w, VISITED)
25               q.enqueue(w)
26           elseif getLabel(v, w) == UNEXPLORED:
27               setLabel(v, w, CROSS)
```

## BFS Observations

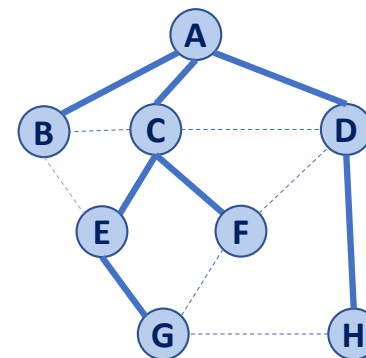
Q: What is a shortest path from **A** to **H**?

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?

d	p	v	Adjacent
0	A	A	C B D
1	A	B	A C E
1	A	C	B A D E F
1	A	D	A C F H
2	C	E	B C G
2	C	F	C D G
3	E	G	E F H
2	D	H	D G





## BFS Observations

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

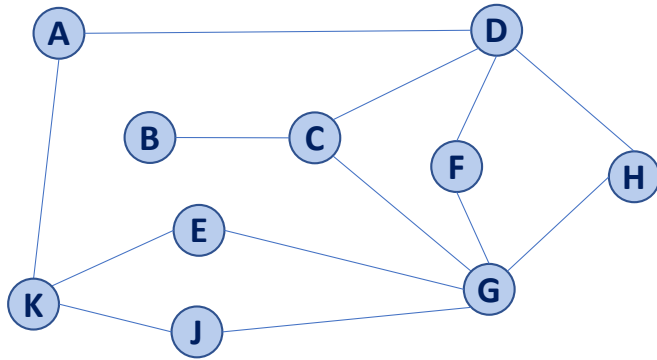
**Obs. 2:** Traversals can be used to detect cycles.

**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



```
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   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20       v = q.dequeue()
21       foreach (Vertex w : G.adjacent(v)):
22           if getLabel(w) == UNEXPLORED:
23               setLabel(v, w, DISCOVERY)
24               setLabel(w, VISITED)
25               q.enqueue(w)
26           elseif getLabel(v, w) == UNEXPLORED:
27               setLabel(v, w, CROSS)
```

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



# Running time of DFS

## Labeling:

- Vertex:
- Edge:

## Queries:

- Vertex:
- Edge:

