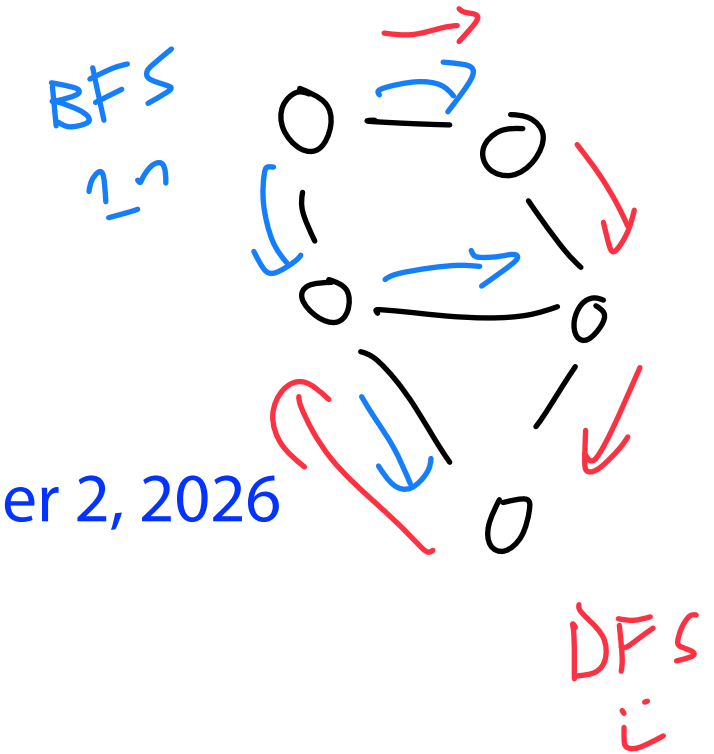
Data Structures
Graph Traversals



CS 225
Brad Solomon

November 2, 2026



Department of Computer Science

Learning Objectives

Introduce graph traversal algorithms

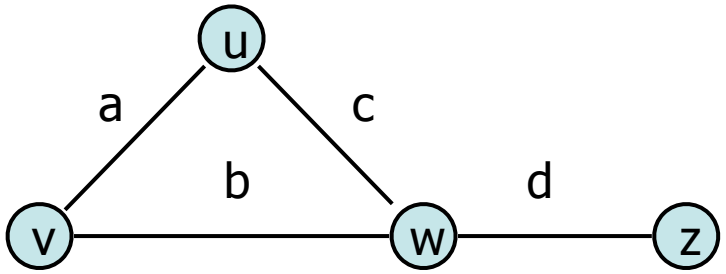
Announcement

↳ Exam 4 is next week!

↳ Lab for this week is review session

Bonus lab releasing this week

Adjacency List

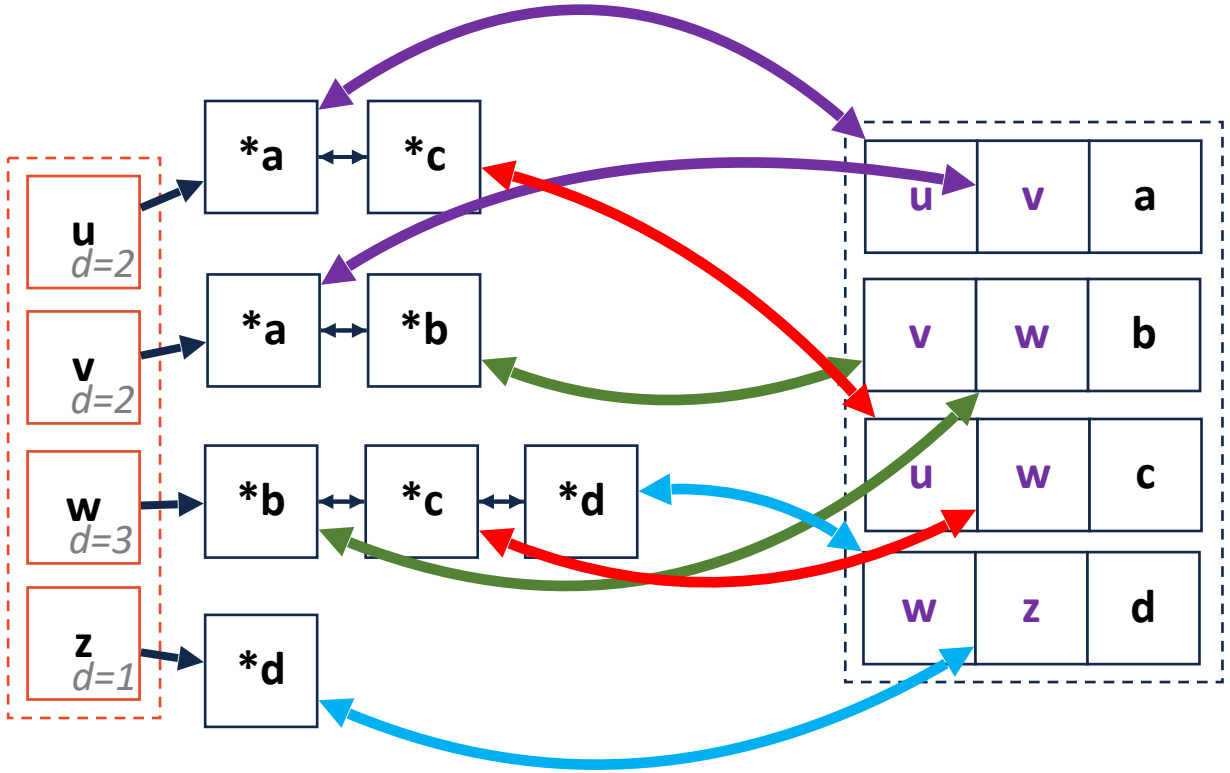


Vertex Storage:

A map with start vertex key and LL value

Each linked list is bidirectional with size variable

Each node is a pointer to edge in edge list



Edge Storage:

A list of (v1, v2, weight) edges

Also store pointers back to nodes

$|V| = n, |E| = m$



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

If dense ↔ The same

If sparse deg(v) better

Graph Traversals

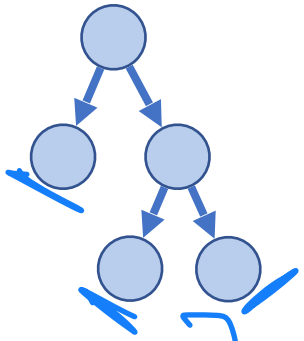
→ Navigate a maze
→ Solves shortest path*
→ Produces a spanning tree*
→ Topological sort
→ ....

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

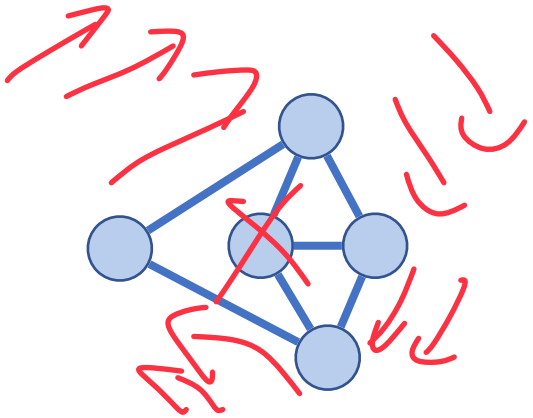
- All nodes connected
- No cycles
- Putting all but not edges

How can we systematically go through a complex graph in the fewest steps?

Tree traversals won't work — lets compare:

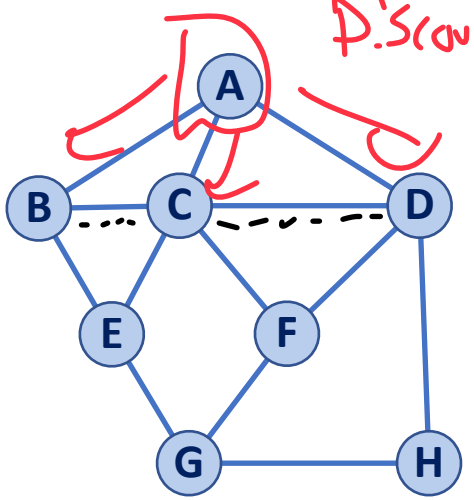


- Rooted
- Acyclic
- **A clear 'endpoint'**



- No root (any start position valid)
- Cycles
- **No obvious 'endpoint'**

Traversal: BFS



To complete a traversal what do we need?

As input:

A starting node

(a graph)

To track visited list AND track label information AND solve multiple graph problems:

- Track distance from start
- Predecessor vertex

To keep track of:

Our current location in the traversal (queue)

Our current state in the traversal (visited list)

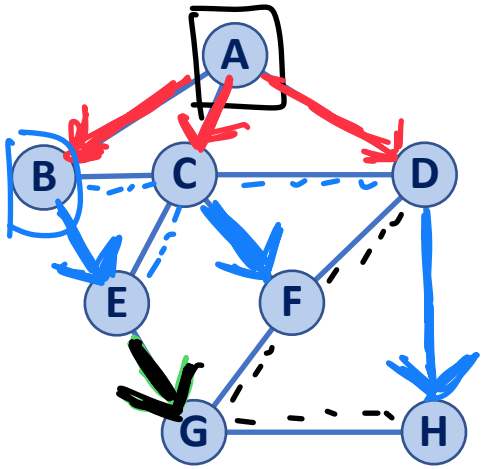
Label information for all edges

→ discovery edge] produces spanning tree

--- cross edge] clear sign of cycle

Traversal: BFS

0) Initializing
- Queue (add start V)
- dist (v is 0)
- pred (pred is null)



Discovery edges →
Cross ----

While Queue is not empty
- tmp = Q.dequeue()
- "Process" tmp
  - Look up all children
  - Add non-processed children to Queue
  - Update dist & prev
    - dist = $dist_{source}$ + 1
    - prev = source

label / process using dist / pred is implementation choice

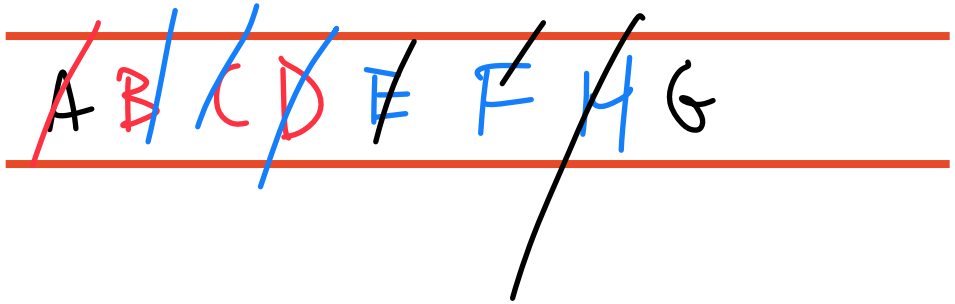
dist pred — what imp choice?

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

Adj matrix 25%
dist 70%
B: end, start

This is my visited list

Front



Traversal: BFS

|V| = n, |E| = m



Initialize queue / dist / predecessor

While queue not empty:

Remove front vertex of queue

Check if edge connects to new vertex

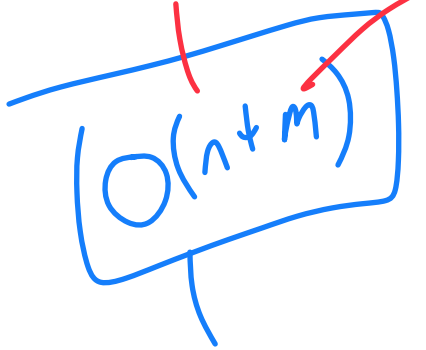
Set dist / pred if new vertex

Add unvisited edges to queue

process non processed children

visit n nodes

label m edges



O(n) 6 6/6

O(m) 20 5/6

O(n+m) 20/20

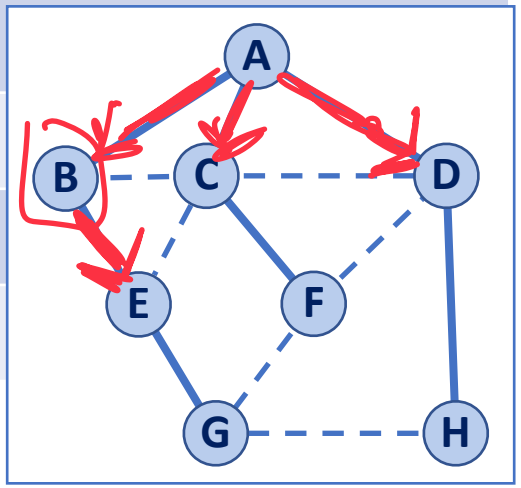
$O(n^2)$

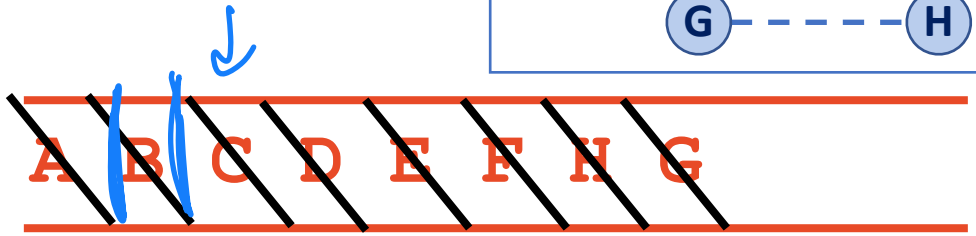
$O(m^2)$ X

Running time?

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

Join Code: 225



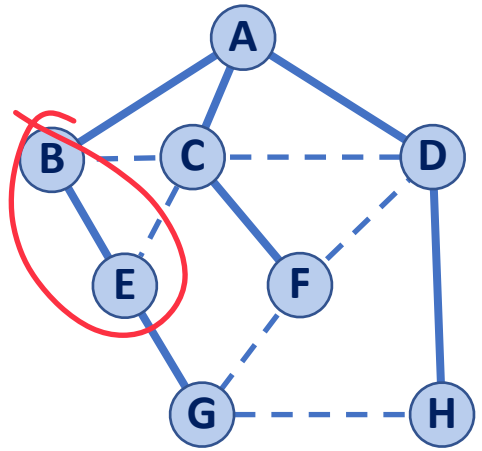


Traversal: BFS

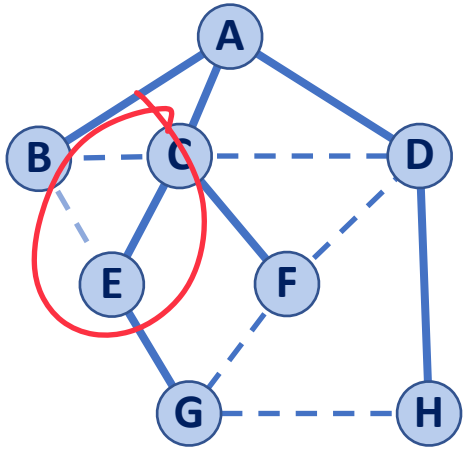
Implementation choice of graph matters

Traversal depends on start position as well as order of edges at each node

output



v	d	P	Adjacent Edges
A	0	-	B C D
B	1	A	A C E
C	1	A	A B D E F



v	d	P	Adjacent Edges
A	0	-	C B D
B	1	A	A C E
C	1	A	A B D E F

Input: Graph, G

Output: A labeling of the edges in G as discovery or cross

```
 1 BFS(G):
 2   foreach (Vertex v : G.vertices()):
 3     setPred(v, NULL)
 4     setDist(v, -1)
 5
 6   foreach (Edge e : G.edges()):
 7     setLabel(e, UNEXPLORED)
 8
 9   foreach (Vertex v : G.vertices()):
10     if getDist(v) == -1:
11        BFS(G, v)
```

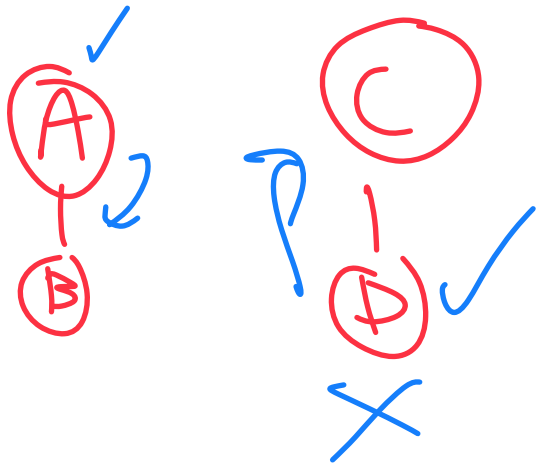
unlabeled vertices

Initialization

$O(n+m)$

unlabeled edges

???



Multiple connected components

```
1  BFS(G):
2    foreach (Vertex v : G.vertices()):
3      setPred(v, NULL)
4      setDist(v, -1)
5
6    foreach (Edge e : G.edges()):
7      setLabel(e, UNEXPLORED)
8
9    foreach (Vertex v : G.vertices()):
10     if getDist(v) == -1:
11       BFS(G, v)
```

$\sum_{v} \deg(v) = 2|E|$ (undirected)

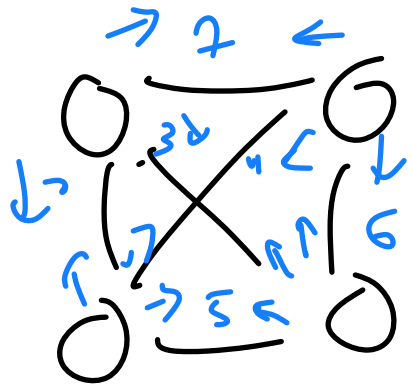
v is start

```
12 BFS(G, v):
13   Queue q
14   setDist(v, 0)
15   q.enqueue(v)
16
17   while !q.empty():
18     v = q.dequeue()
19
20   foreach (Vertex w : G.adjacent(v)):
21     if( getDist(w) == -1):
22       setLabel((v, w), DISCOVERY)
23       setPred(w, v)
24       setDist(w, v + 1)
25       q.enqueue(w)
26     else:
27       setLabel((v, w), CROSS)
```

Init start to 0

$|V|\times$ deg(w)

Process & label children



3 + 3 + 3 + 3 = 12

2·6

```
1  BFS(G):
2    foreach (Vertex v : G.vertices()):
3      setPred(v, NULL)
4      setDist(v, -1)
5
6    foreach (Edge e : G.edges()):
7      setLabel(e, UNEXPLORED)
8
9    foreach (Vertex v : G.vertices()):
10     if getDist(v) == -1:
11       BFS(G, v)
```

Count connected components?

↳ Yes! Lines 9-11

Add a count that gives up # of connected components

```
12 BFS(G, v):
13   Queue q
14   setDist(v, 0)
15   q.enqueue(v)
16
17   while !q.empty():
18     v = q.dequeue()
19
20     foreach (Vertex w : G.adjacent(v)):
21       if( getDist(w) == -1):
22         setLabel((v, w), DISCOVERY)
23         setPred(w, v)
24         setDist(w, v + 1)
25         q.enqueue(w)
26       else:
27         setLabel((v, w), CROSS)
```

Labeled directly

Cycle Detection?

↳ Yes

Edge labels tell us if a cycle exist



BFS Observations

What is the shortest path from **A** to **H**?

A — D — H

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**

What is the shortest path from **E** to **H**?

↳ We don't know only shortest path from start

E — H

BFS Observations

What is the shortest path from **A** to **H**?

A-D-H (see by walking backwards!)

What is the shortest path from **E** to **H**?

Can't tell by BFS!

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**

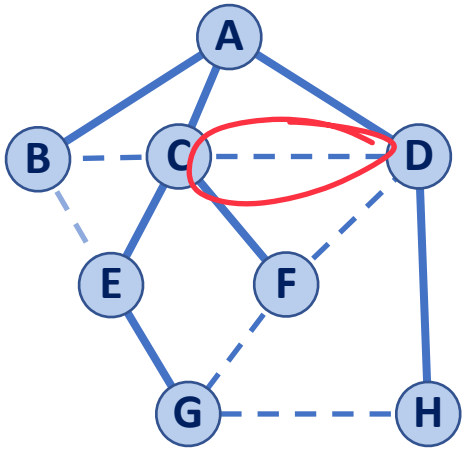
BFS Observations

If my node has distance **d**, do I know anything about the nodes connected by a **cross edge**?

A cycle exists!

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

If my node has distance **d**, do I know anything about the nodes connected by a **cross edge**?

A cross edge tells us we have at least one cycle

What structure is made from **discovery edges**?

A spanning tree

root is start
No cycles

↳ minimum spanning tree for unweighted graphs

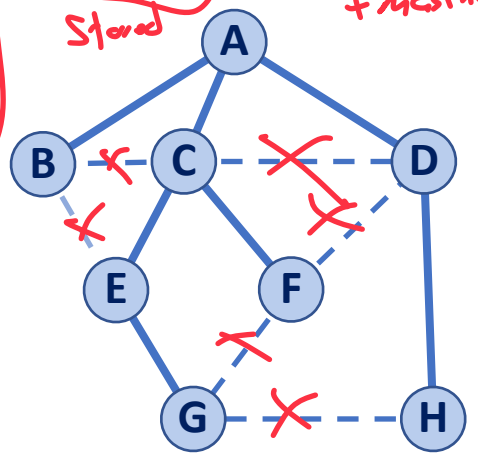
v	d	P	Adjacent Edges
A	0	-	C B D
B	1	A	A C E
C	1	A	B A D E F
D	1	A	A C F H
E	2	C	B C G
F	2	C	C D G
G	3	E	E F H
H	2	D	D G

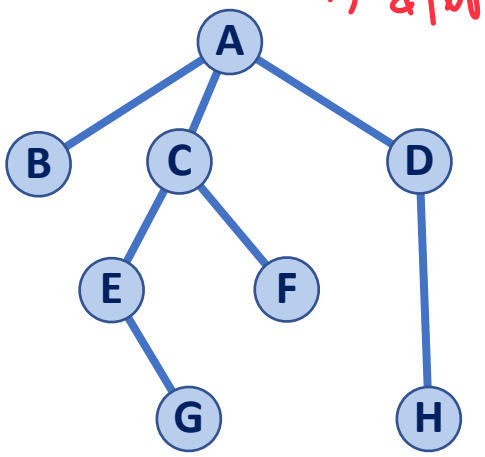
Stored

Traversing

G.adjacent(v)

repeatedly





BFS Observations

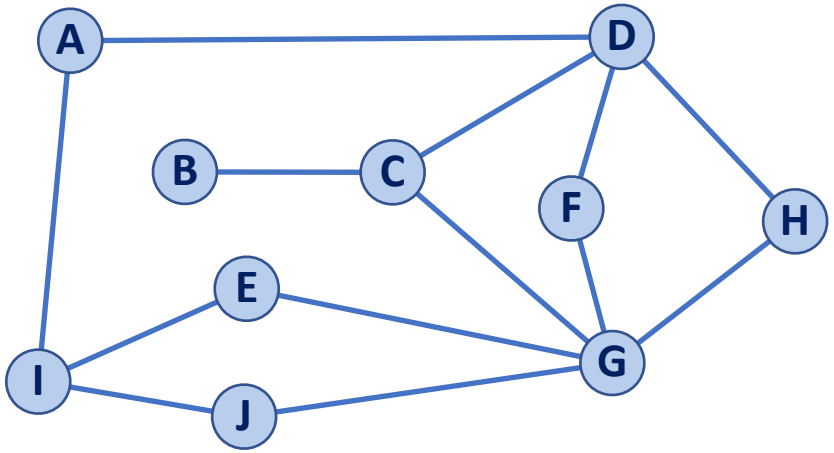


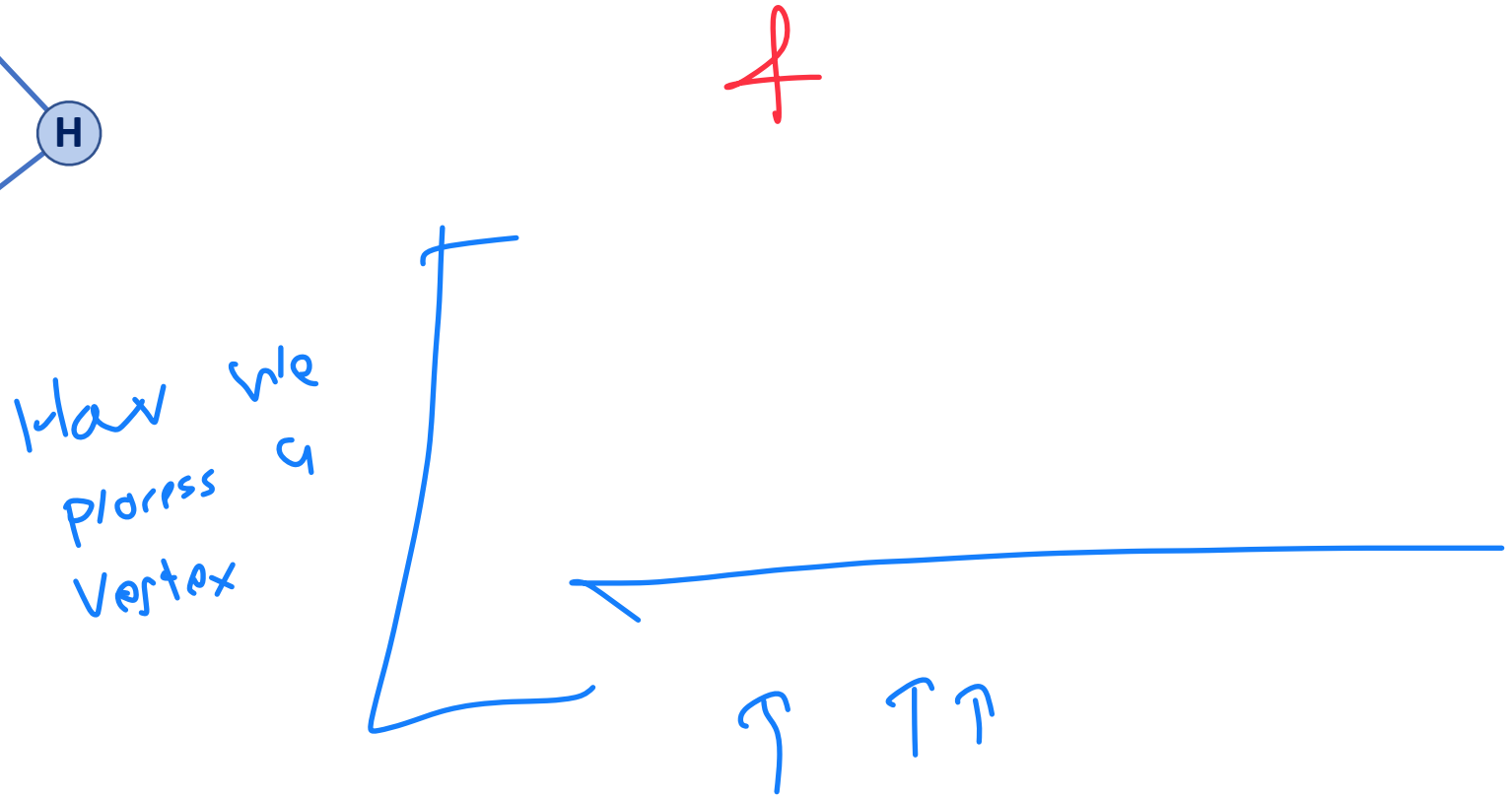
1. BFS can be used to count components
2. BFS can be used to detect cycles
3. The BFS 'distance' value is always the shortest distance from source to any vertex (and the discovery edges form a spanning tree)
4. The endpoints of a cross edge never differ in distance by more than 1 ($|d(u) - d(v)| = 1$)

Traversal: DFS

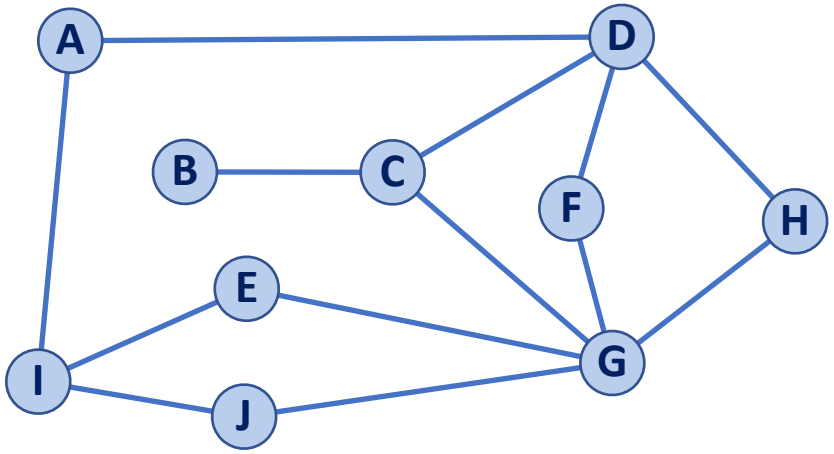
Exactly the same as BFS but...

replace queue w/ stack





Traversal: DFS



Initialize dist / pred / stack

All dist null (start node dist 0)

All pred -1 (start node pred -1)

Stack loaded with start node

While stack not empty

tmp=stack.peek()

Process one unvisited child

Add to stack

dist = tmp.dist+1

pred = tmp

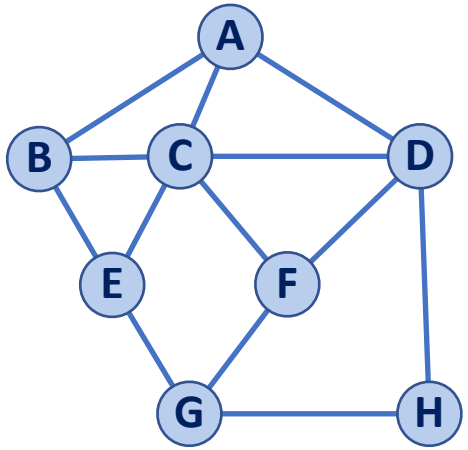
If no unvisited children

stack.pop()

Stack

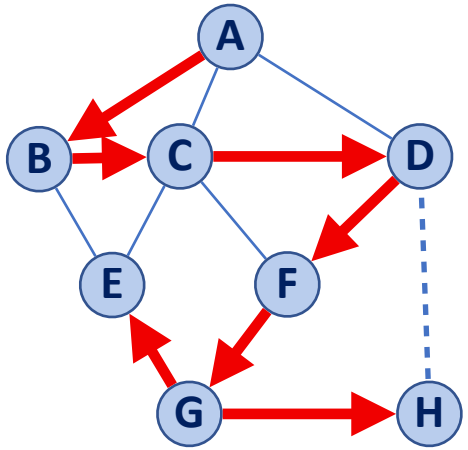


```
1  DFS(G):
2    foreach (Vertex v : G.vertices()):
3      setPred(v, NULL)
4      setDist(v, -1)
5
6    foreach (Edge e : G.edges()):
7      setLabel(e, UNEXPLORED)
8
9    foreach (Vertex v : G.vertices()):
10     if getDist(v) == -1:
11       DFS(G, v)
```




```
12 DFS(G, v):
13
14     foreach (Vertex w : G.adjacent(v)):
15         if( getDist(w) == -1):
16             setLabel((v, w), DISCOVERY)
17             setPred(w, v)
18             setDist(w, v + 1)
19             DFS(G, w)
20         else:
21             setLabel((v, w), BACK)
```

```
12 DFS(G, v):
13
14     foreach (Vertex w : G.adjacent(v)):
15         if( getDist(w) == -1):
16             setLabel((v, w), DISCOVERY)
17             setPred(w, v)
18             setDist(w, v + 1)
19             DFS(G, w)
20         else:
21             setLabel((v, w), BACK)
```



$|V| = n, |E| = m$

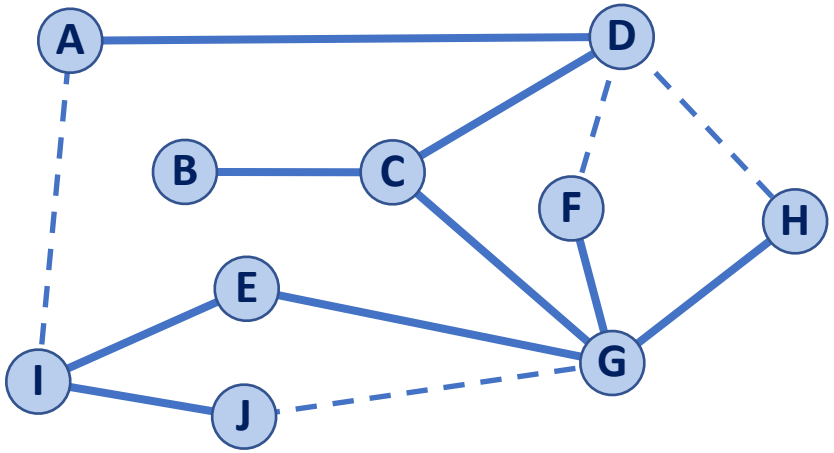
Running time?

v	d	P	Adjacent Edges
A	0	-	B C D
B	1	A	A C E
C	2	B	A B D E F
D	3	C	A C F H
E	6	G	B C G
F	4	D	C D G
G	5	F	E F H
H	6	G	D G

State of stack when H is added:

H
~~E~~
G
F
D
C
B
A

Traversal: DFS



Discovery Edge

Back Edge

Do we still make a spanning tree?

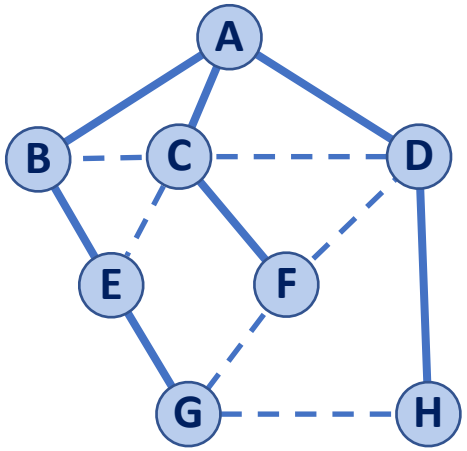
Does distance have meaning here?

Do our edge labels have meaning here?

Efficiency: DFS vs BFS

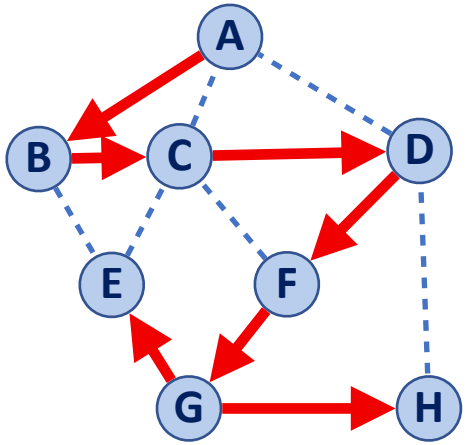
$|V| = n, |E| = m$

BFS:



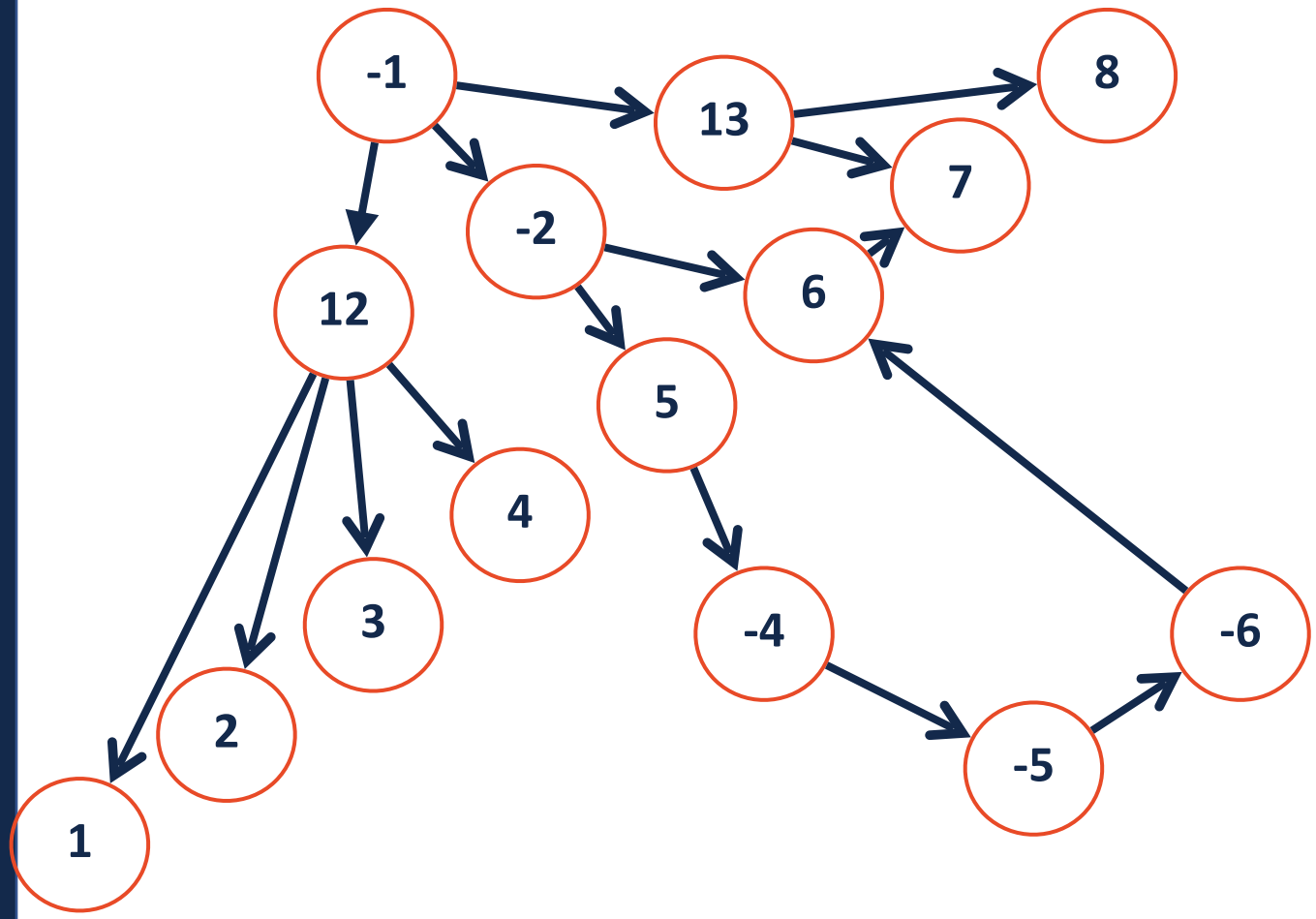
A B C D E F H G

DFS:



A B C D F G E H

Space Efficiency: DFS vs BFS



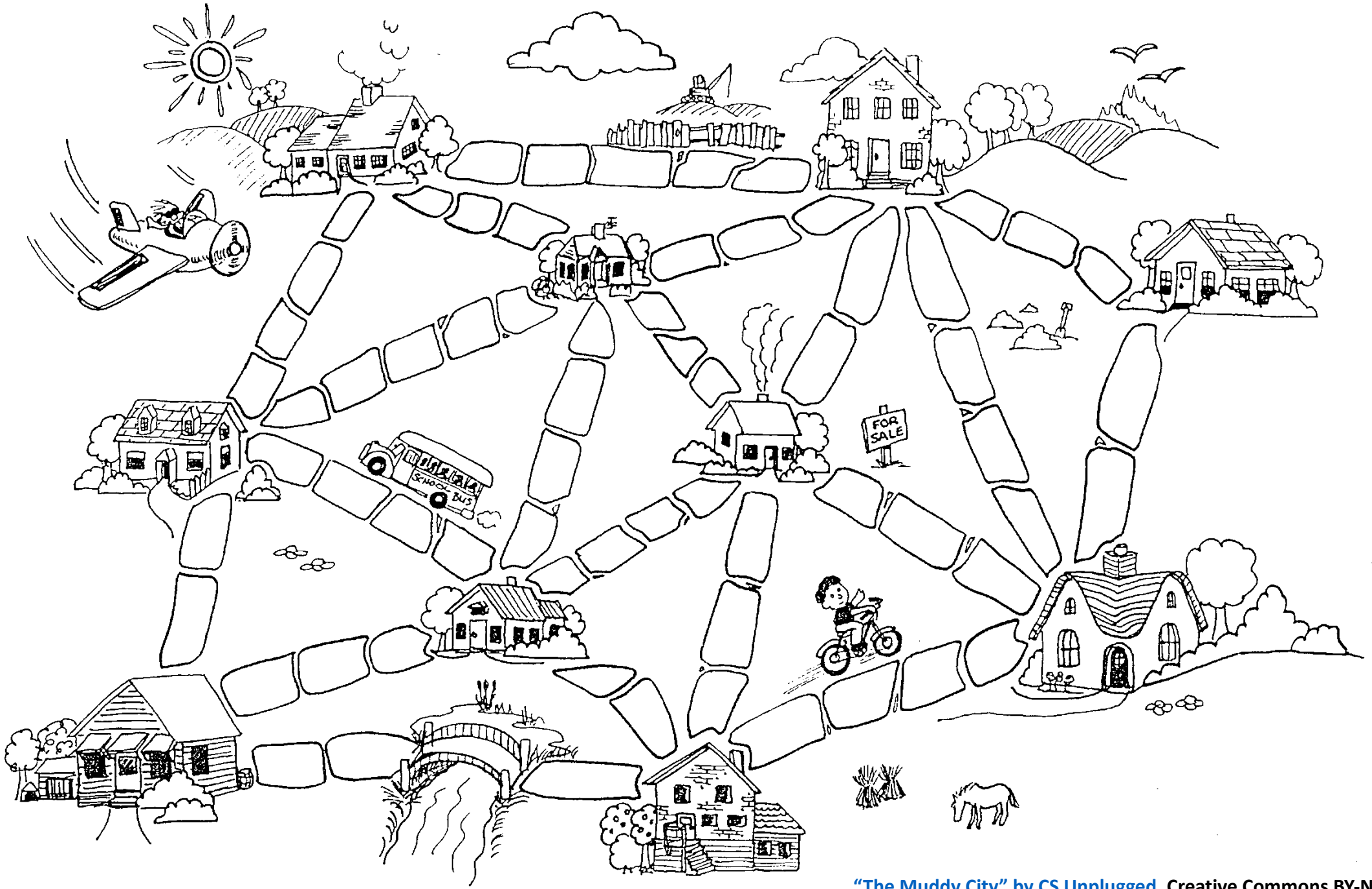
Summary: DFS and BFS

$|V| = n, |E| = m$



Both are **O(n+m)** traversals! They label every edge and every node

BFS	**DFS**
Solves unweighted MST	Solves unweighted MST
Solves shortest path	
Solves cycle detection	Solves cycle detection
Memory bounded by width	Memory bounded by longest path



"The Muddy City" by CS Unplugged, Creative Commons BY-NC-SA 4.0

Minimum Spanning Tree Algorithms

Input: Connected, undirected graph **G** with edge weights (unconstrained, but must be additive)

Output: A graph G' with the following properties:

- G' is a spanning graph of G
- G' is a tree (connected, acyclic)
- G' has a minimal total weight among all spanning trees

