Data Structures
Graph Traversals

CS 225
Brad Solomon

October 29, 2025

Mattox sick
Guest Lecture :)

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Announcements

EC points have been awarded for all IEF and surveys

MP_traversal survey will go out today

FAIR cases are in the works — remember course policies!

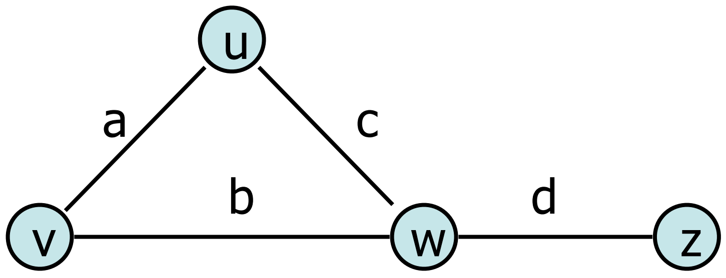
Learning Objectives

Discuss graph traversal algorithms

Graph Implementation: Edge List $|V| = n, |E| = m$

The equivalent of an 'unordered' data structure

Is the smallest storage structure



u
v
w
z

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

Vertex Storage:

An optional list of vertices

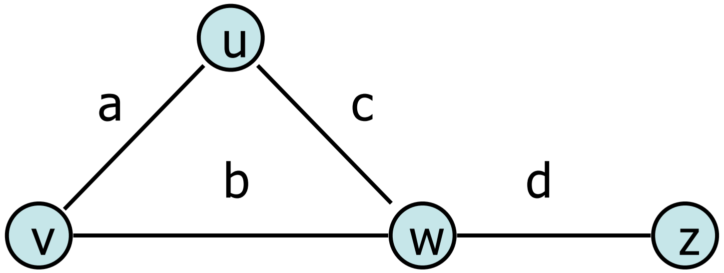
Edge Storage:

A list storing edges as (V1, V2, Weight)

Most graphs are stored as just an edge list!

Graph Implementation: Adjacency Matrix

|V|= n, |E|= m



Vertex Storage:

A hash table of vertices

Implicitly or explicitly store index

u	0
v	1
w	2
z	3

	0	1	2	3
0	-	a	c	0
1		-	b	0
2			-	d
3				-

Edge Storage:

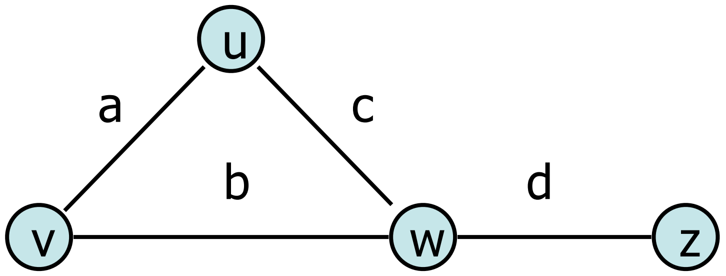
A |V| x |V| matrix of edges

Big

Weight is stored at position (u, v)

O(1) look up in vector

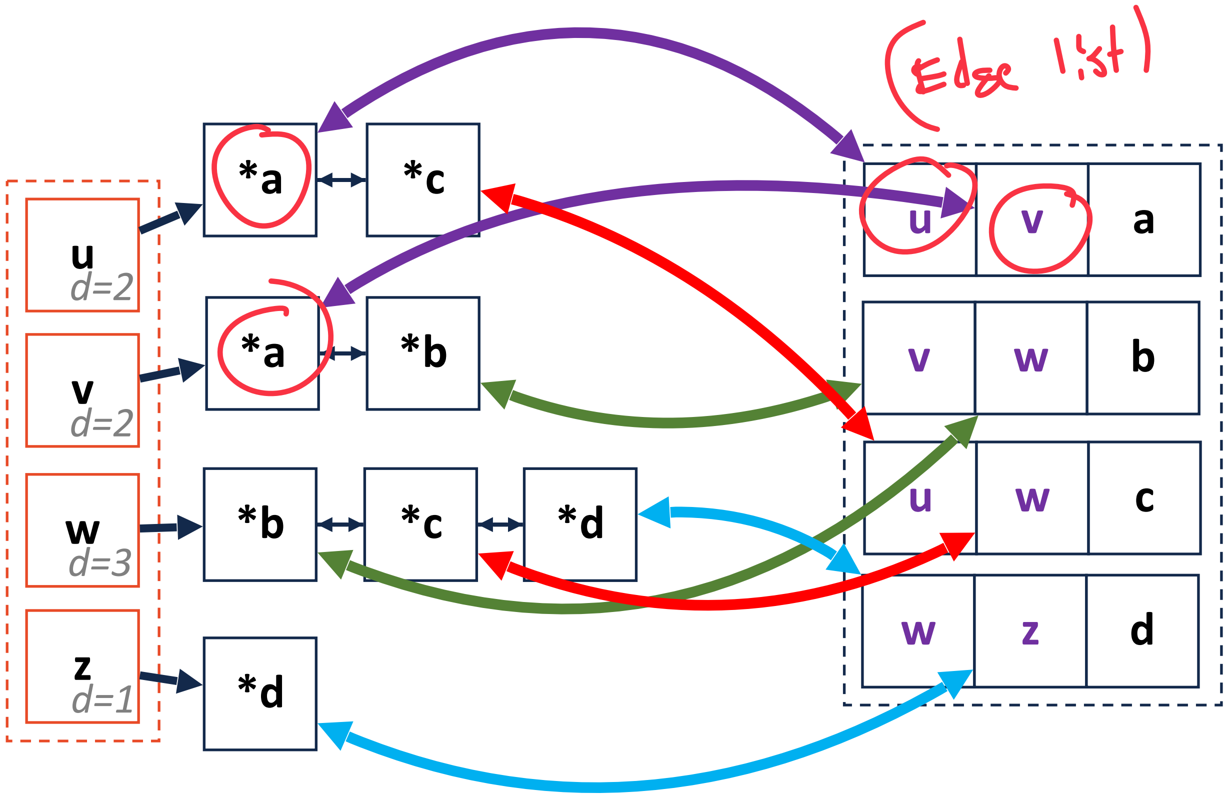
Adjacency List



Vertex Storage:

A bidirectional linked list 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	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))

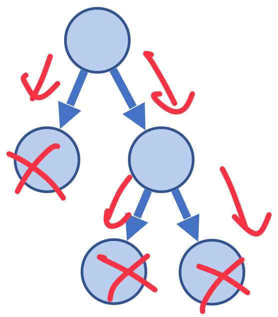
*

Graph Traversals

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

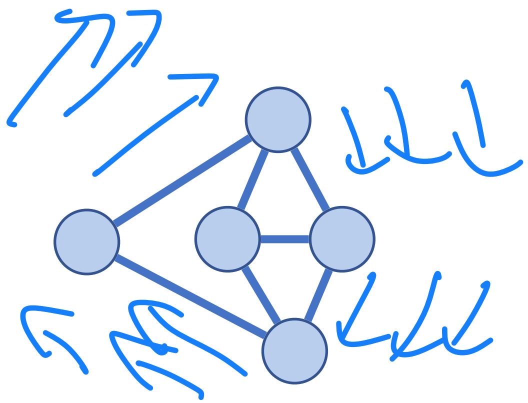
How can we systematically go through a complex graph in the fewest steps?

Tree traversals won't work — lets compare:



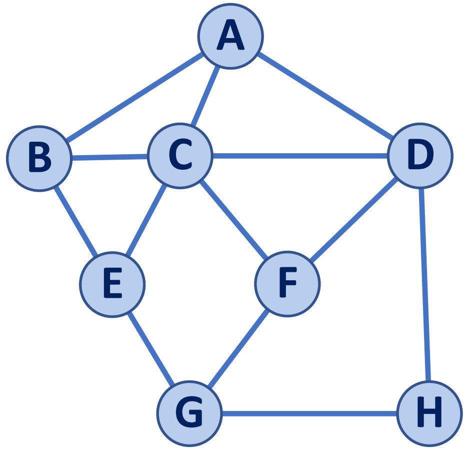
- Rooted
- Acyclic
- A clear endpoint

↳ If node has no children!



- No clear root
- cycles
- No obvious endpoint!

Traversal: BFS



To complete a traversal what do we need?

As input:

↳ I need a clear starting node

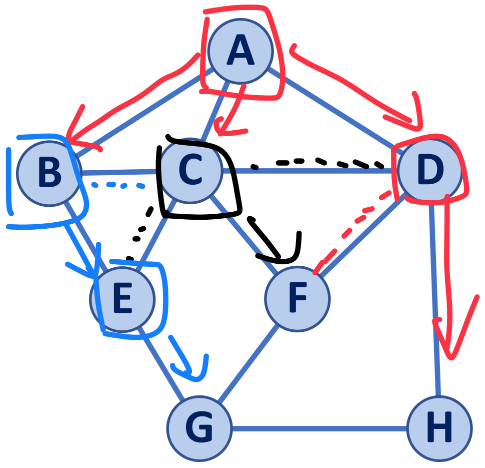
To keep track of:

↳ Nodes we visit so far

↳ and what node comes next

Traversal: BFS

0) Initialize
↳ Queue (put start in queue)
↳ Depth (start = 0)
↳ Predecessor (start = -1)



while queue not empty
↳ tmp = dequeue()
↳ Process all children
↳ Add to queue
↳ Set depth (tmp.depth +1)
↳ Set prev (to tmp)

→ Discovery (New vertex)
..... Cross (old vertex)

All discovery edges are in table
All edges not in table are cross

"visited" nodes have depth/pred

depth pred

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	2	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

Check if vertex is "new"
Does it already have depth/pred

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

Front (queue) Back

Traversal: BFS

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

Initialize queue / depth / predecessor

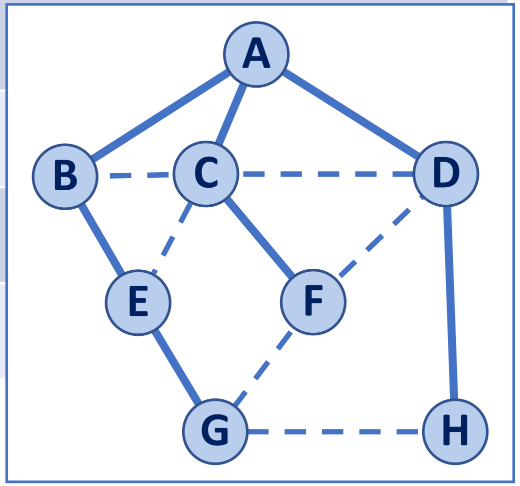
While queue not empty:

- Remove front vertex of queue
- Check if edge connects to new vertex
  - Set dist / pred if new vertex
- Add unvisited edges to queue

Every edge "visited" twice

Every vertex is processed

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

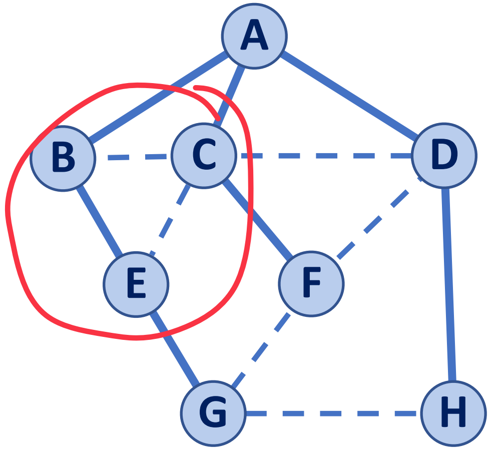


~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~ ~~F~~ ~~H~~ ~~G~~

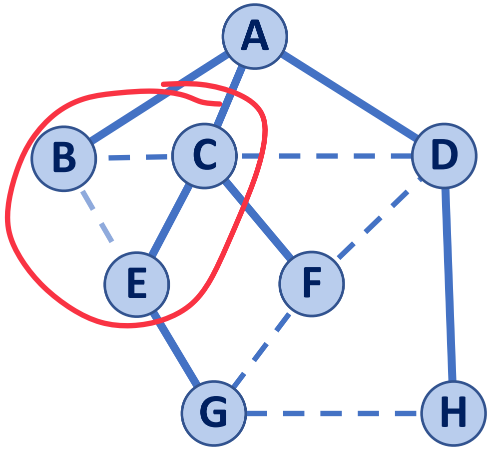
Running time? $O(m + n)$

Traversal: BFS

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



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

```
BFS(G):
  foreach (Vertex v : G.vertices()):
    setPred(v, NULL)
    setDist(v, -1)

  foreach (Edge e : G.edges()):
     setLabel(e, UNEXPLORED)

  foreach (Vertex v : G.vertices()):
     if getDist(v) == -1:
         BFS(G, v)
```

Initialization

start vertex

```
BFS(G):
  foreach (Vertex v : G.vertices()):
    setPred(v, NULL)
    setDist(v, -1)

  foreach (Edge e : G.edges()):
    setLabel(e, UNEXPLORED)

  foreach (Vertex v : G.vertices()):
    if getDist(v) == -1:
      BFS(G, v)
```

```
BFS(G, v):
  Queue q
  setDist(v, 0)
  q.enqueue(v)

  while !q.empty():
    v = q.dequeue()

    foreach (Vertex w : G.adjacent(v)):
      if( getDist(w) == -1):
        setLabel((v, w), DISCOVERY)
        setPred(w, v)
        setDist(w, v + 1)
        q.enqueue(w)
      else:
        setLabel((v, w), CROSS)
```

Initialize

label

```
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)
```

Init

call BFS

```
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)
```

Mere init

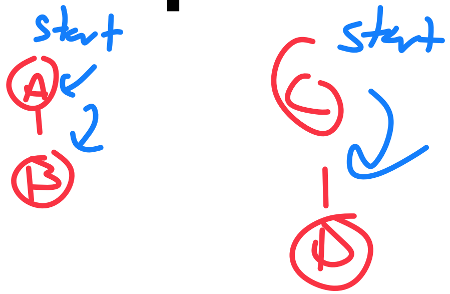
label

G2

30%

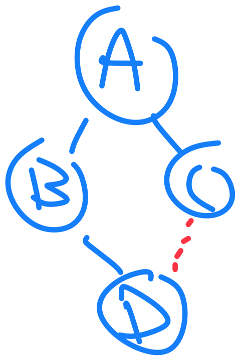
Count connected components?

9-11



Cycle Detection?

↳ cross edge label!



BFS Observations

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

A - D - H

walk backwards!

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

↳ we don't know!

BFS in graphs solves shortest path (for the start node)

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

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**?

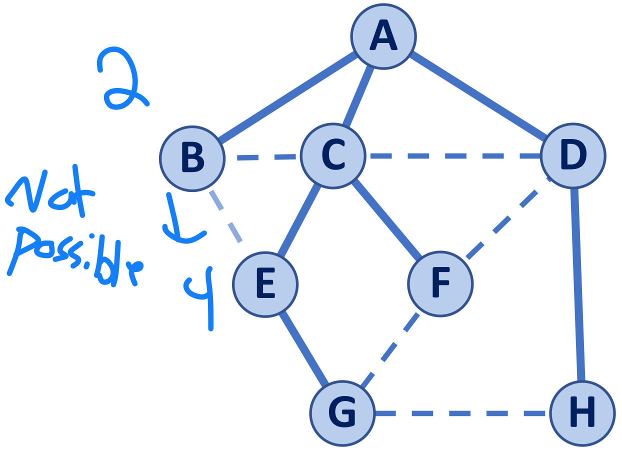
Can't tell by BFS!

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

Are ± 1 in depth/distance to me

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

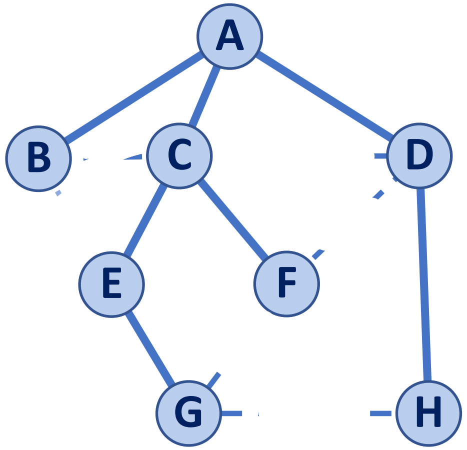
What structure is made from **discovery edges**?

↳ A tree!

(Spanning tree)

↳ connects all vertices in smallest count of 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

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)

↳ Solves shortest path

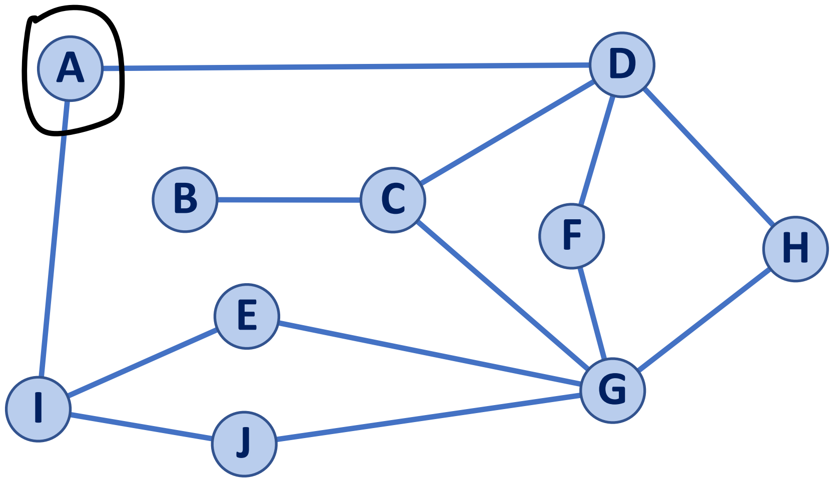
↳ Solves minimum spanning tree

4. The endpoints of a cross edge never differ in distance by more than 1 (**|d(u) - d(v)| = 1**)

Traversal: DFS

1) Initialize distance / pred / stack

↳ stack takes start



while stack not empty

tmp = pop()

process children

↳ depth = tmp.depth+1

↳ pred = tmp

A

Bottom

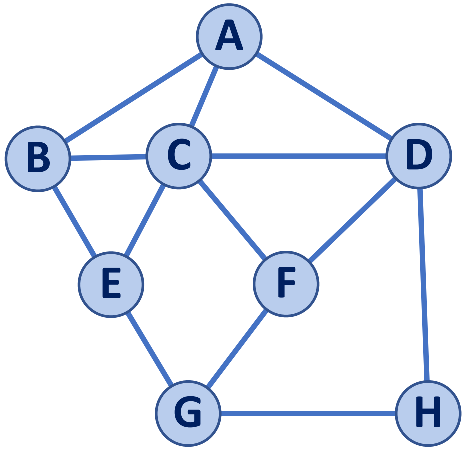
Go over example on Friday

```
DFS(G):
  foreach (Vertex v : G.vertices()):
   setPred(v, NULL)
   setDist(v, -1)

  foreach (Edge e : G.edges()):
    setLabel(e, UNEXPLORED)

  foreach (Vertex v : G.vertices()):
    if getDist(v) == -1:
        DFS(G, v)
```

init

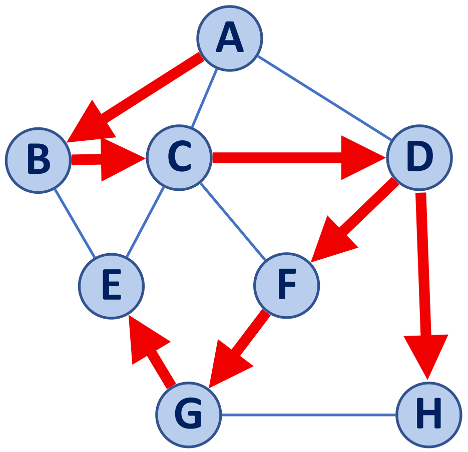
```
DFS(G, v):

    foreach (Vertex w : G.adjacent(v)):
        if( getDist(w) == -1):
            setLabel((v, w), DISCOVERY)
            setPred(w, v)
            setDist(w, v + 1)
            DFS(G, w)
        else:
            setLabel((v, w), BACK)
```

label



```
DFS(G, v):

    foreach (Vertex w : G.adjacent(v)):
        if( getDist(w) == -1):
            setLabel((v, w), DISCOVERY)
            setPred(w, v)
            setDist(w, v + 1)
            DFS(G, w)
        else:
            setLabel((v, w), BACK)
```

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	4	D	D G



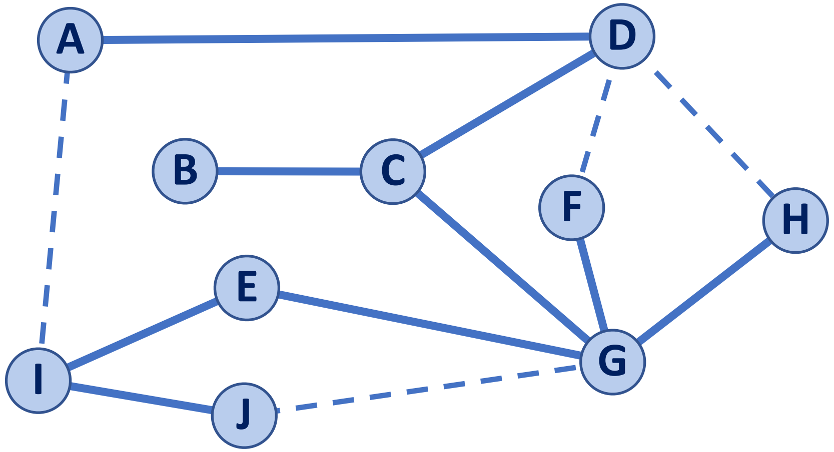
Or go over this example

A B C D F G E H

Running time?

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

Traversal: DFS



Discovery Edge

Back Edge

Do we still make a spanning tree?

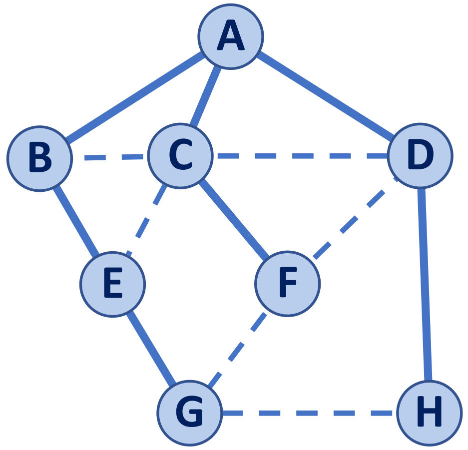
Does distance have meaning here?

Do our edge labels have meaning here?

Efficiency: DFS vs BFS

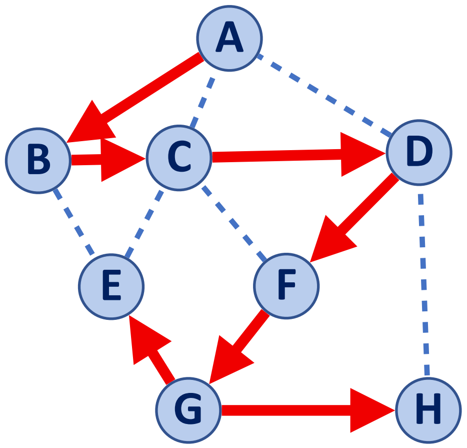
$|V| = n, |E| = m$

BFS:



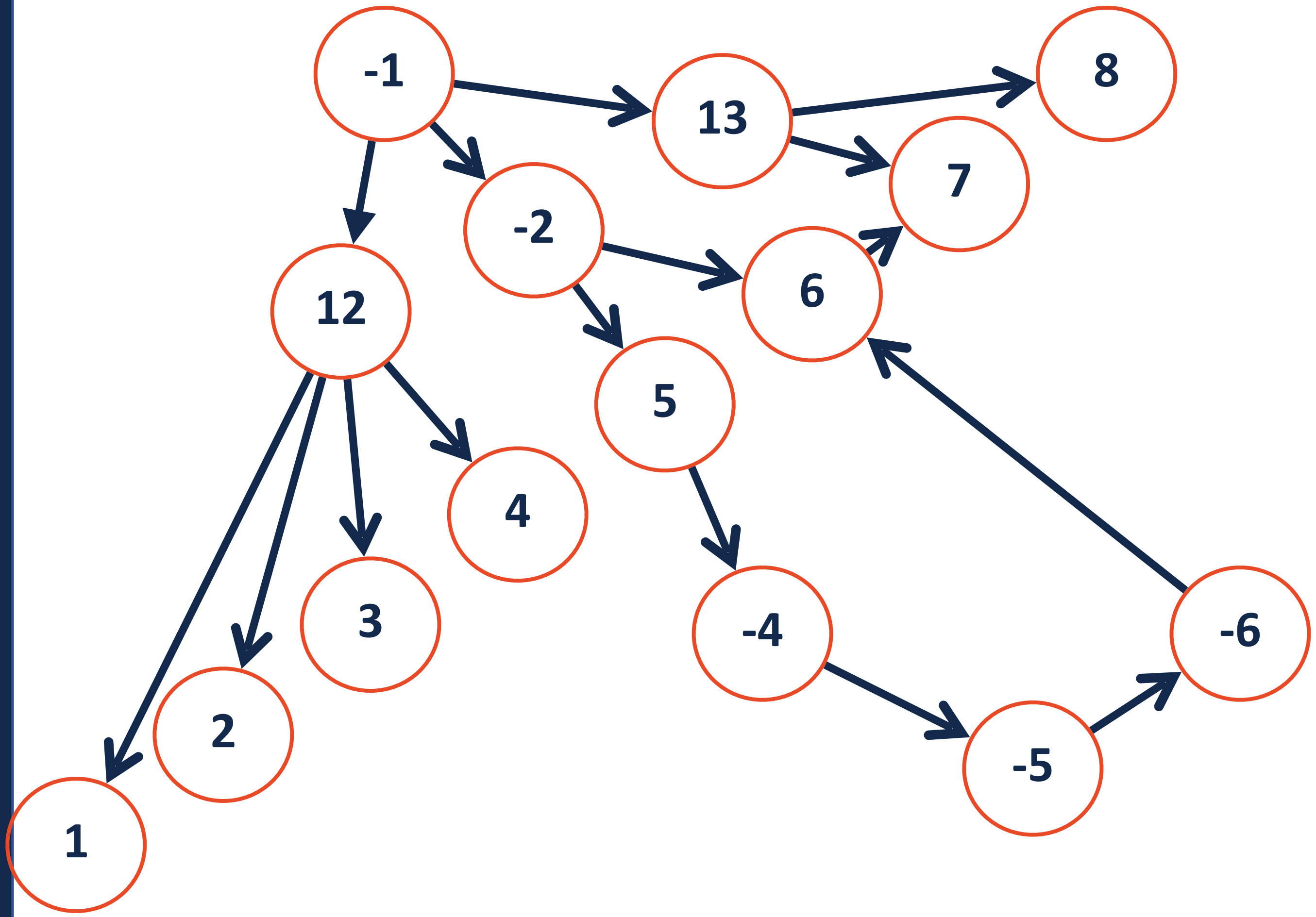
A B C D E F H G

DFS:



A B C D F G E H

Space Efficiency: DFS vs BFS

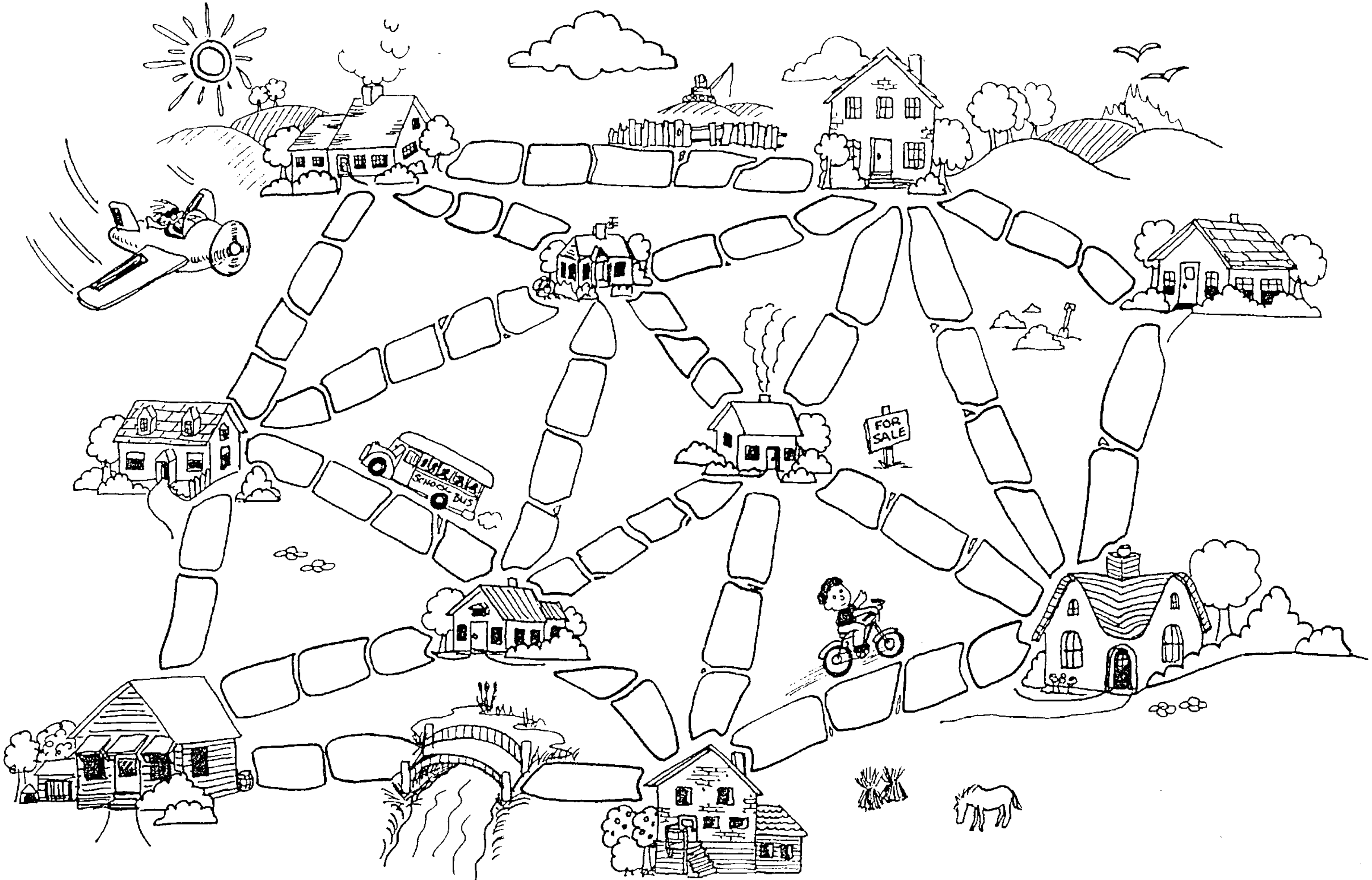


Summary: DFS and BFS

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

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


SCHOOL BUS
FOR SALE

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