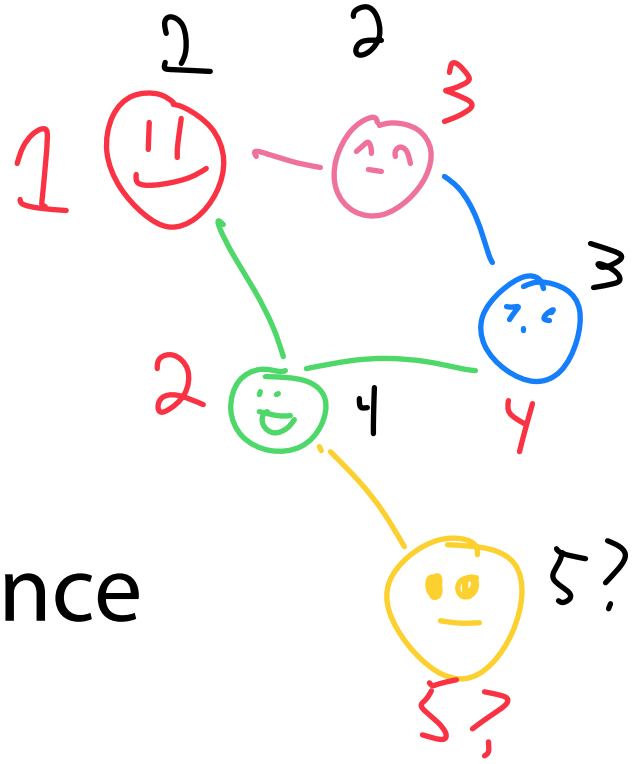
Data Structures

Graph Traversals

CS 225

Brad Solomon & G Carl Evans

November 17, 2023

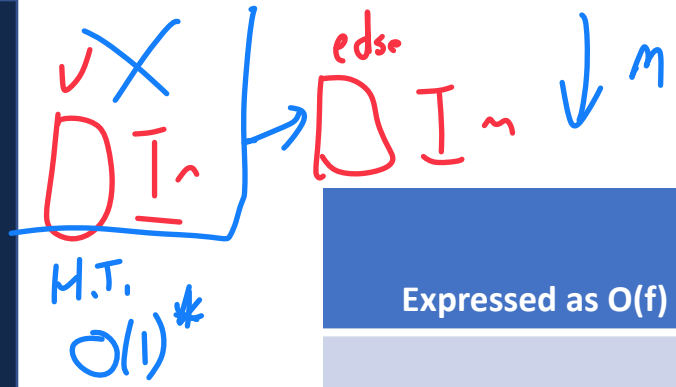


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Discuss graph traversal algorithms



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

Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space	n+m	n^2	n+m
insertVertex(v)	1*	n*	1*
removeVertex(v)	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

→ Why traversal?
↳ solving a maze (specific)
↳ shortest path

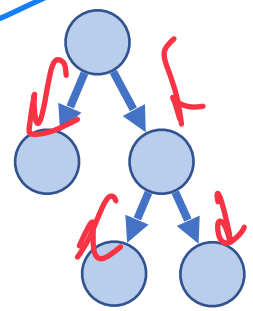
Sub-structures information about graph structure

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

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

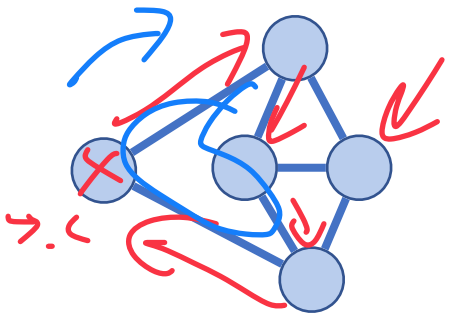
Tree traversals won't work — lets compare:

Tree



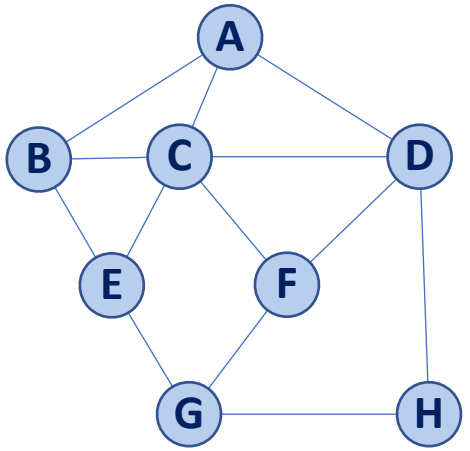
- Rooted
- Acyclic
- Notion of completeness

Graph



- No root / clear start node
- cycles
- No notion of completeness

Traversal: BFS



1) Starting node (random or user input)

2) Track current node

3) Track nodes and edges we have visited

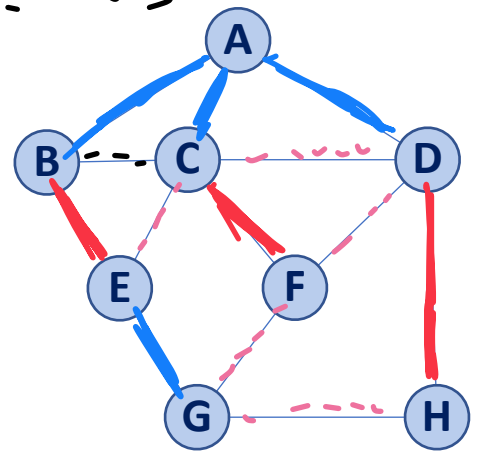
Traversal: BFS

0) Init edge labels = undiscovered, distance

1) Initialize a queue (u / v)
   - add v to queue
   - storing dist / pred

2) while queue not empty
   - Dequeue (v)
   - Add all unvisited neighbors to queue
     - Process neighbors
       - Set Dist
       - Set Pred

P: discovery
---: cross



V (current node)

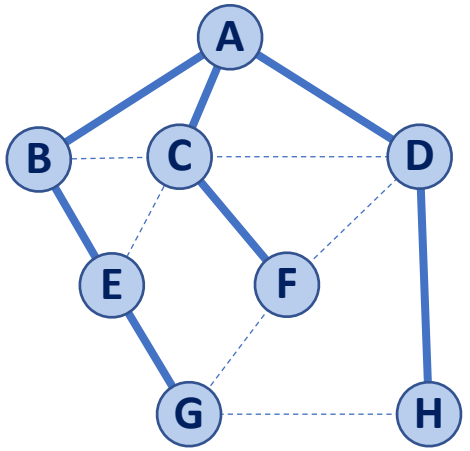
w (my neighbors)

2 ← Predecessor

v	d	P	Adjacent Edges
A	0	null	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~~

Traversal: BFS

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



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

$O(m)$

$2|E| = \sum deg(v)$

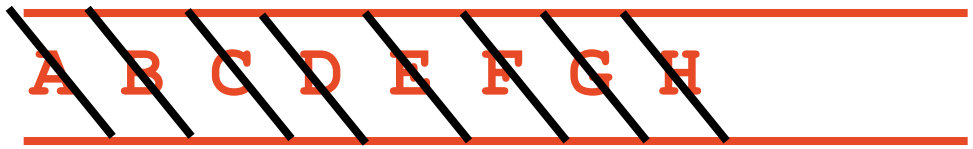
Runtime?

↳ # of times looked at vertex $O(n)$

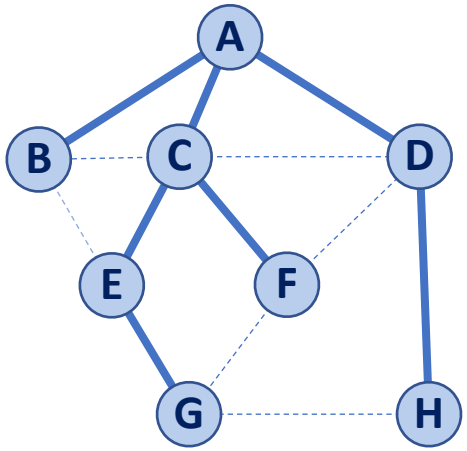
↳ # of times processed vertex $\longrightarrow O(n+m)$

↳ # of times labeled edge $O(m)$

looked $2m$

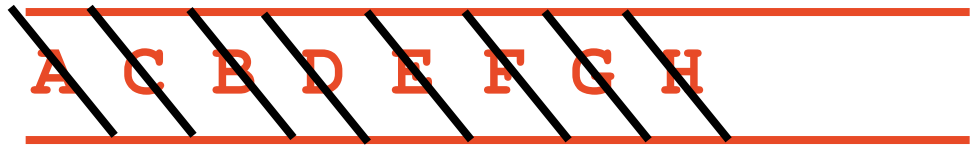


Traversal: BFS



v	d	P	Adjacent Edges
A	0	-	C B 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

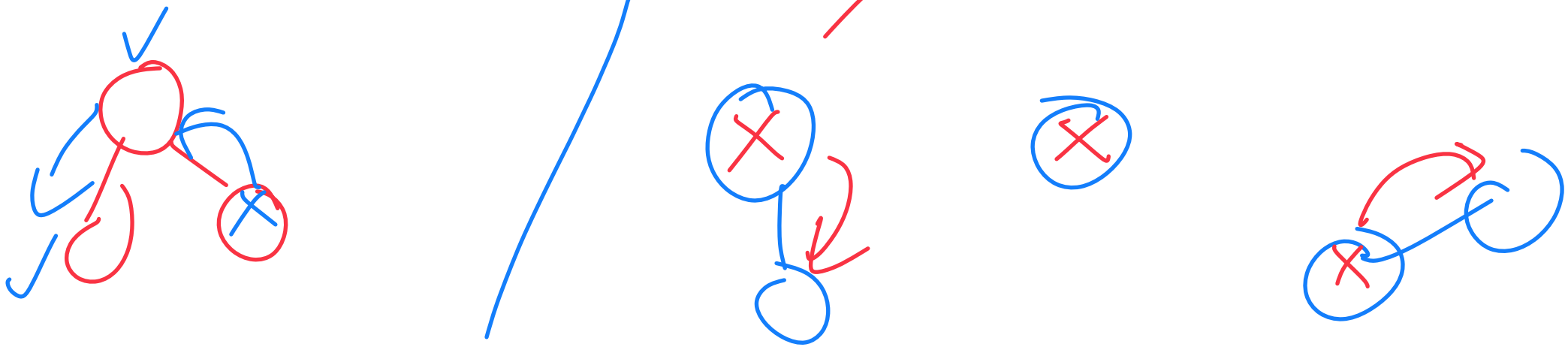
Different traversals for different start node & different neighbor order



Input: Graph, G

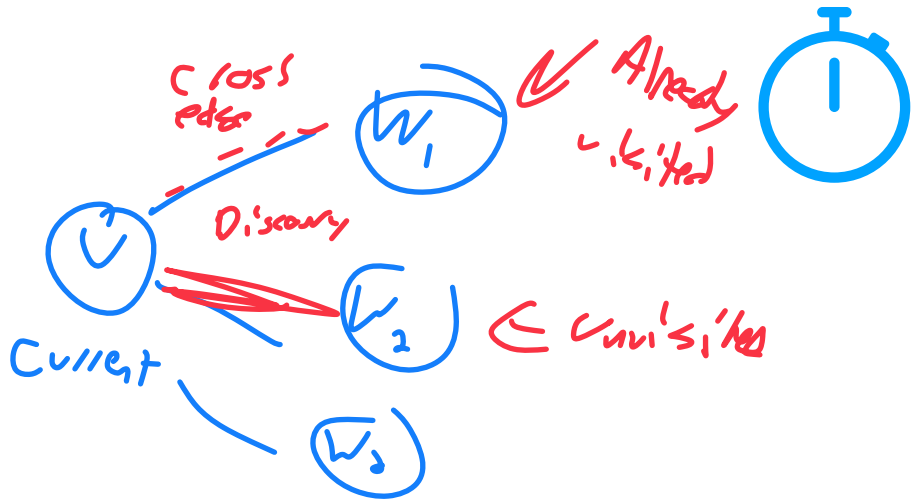
Output: A labeling of the edges in G as discovery or cross

```
 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)
```



```
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)
```

```
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 if edge unlabeled
27         setLabel((v, w), CROSS)
```



initialize

Look at edge

unvisited vertex/edge gets labeled

```
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

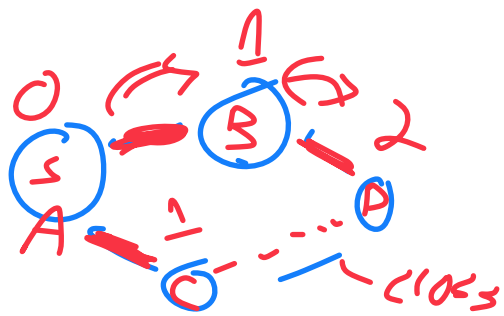
# of 0s in dist is # components

count it

```
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: If edge unlabeled (edge == unexplored)
27         setLabel((v, w), CROSS)
```

Cycle Detection?

↳ cross edge is at least one cycle



BFS Observations

source

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 **A** to **H**?

2

A — D — H

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

↳ No idea!

BFS gives shortest path from source to every other node

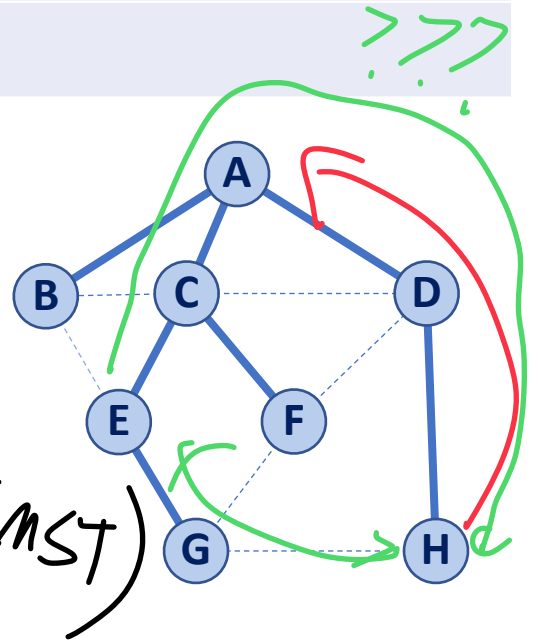
If my node has distance **d**, do I know anything about the nodes connected by a **cross edge**?

vertex u, v : $|d(u) - d(v)| \leq 1$

↳ cross edge

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

Minimum Spanning tree (MST)



BFS Observations

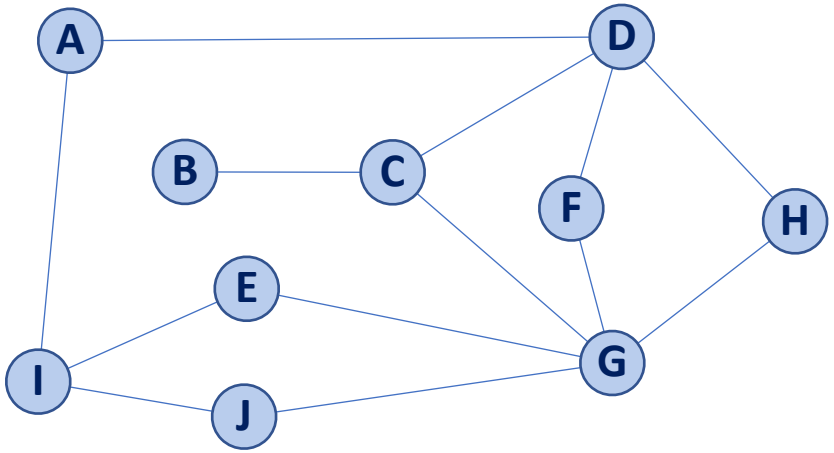
1. BFS can be used to count components
2. BFS can be used to detect cycles

3. The BFS 'distance' value is always the shortest distance from source to any vertex (and the discovery edges form a MST)

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

Traversal: DFS

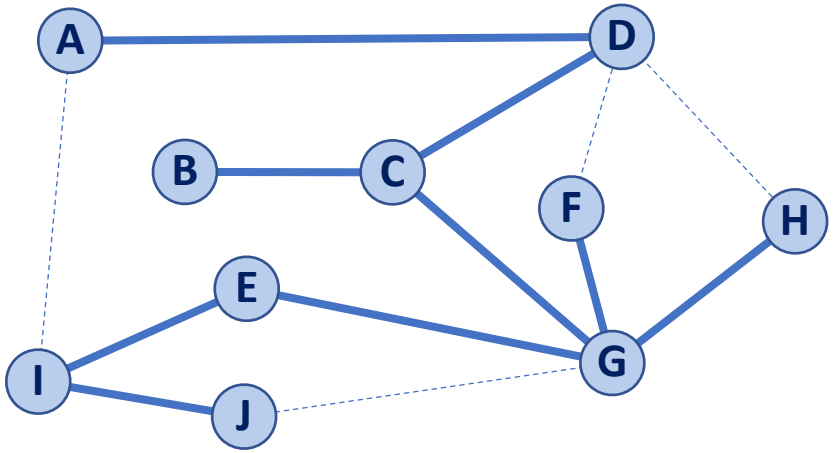
↪ DFS is BFS but a stack and not a queue



```
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)
22
23
24
25
26
27
```

Traversal: DFS



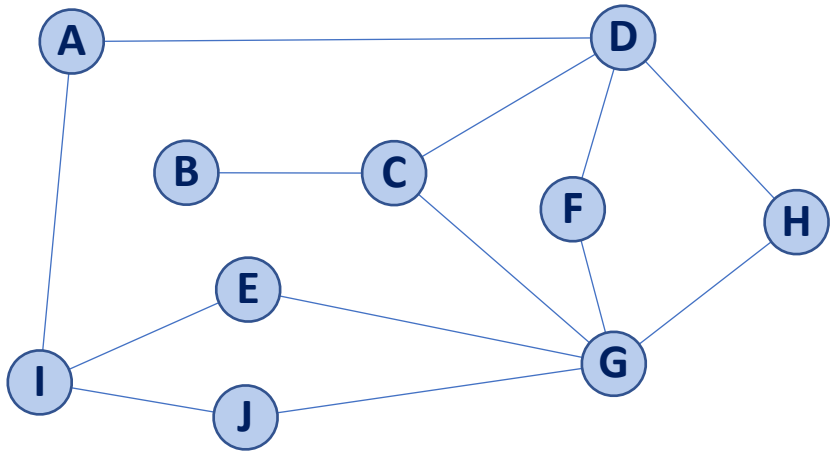
Does distance have meaning here?

Do our edge labels have meaning here?

Discovery Edge

Back Edge

Running time of DFS



Labeling:

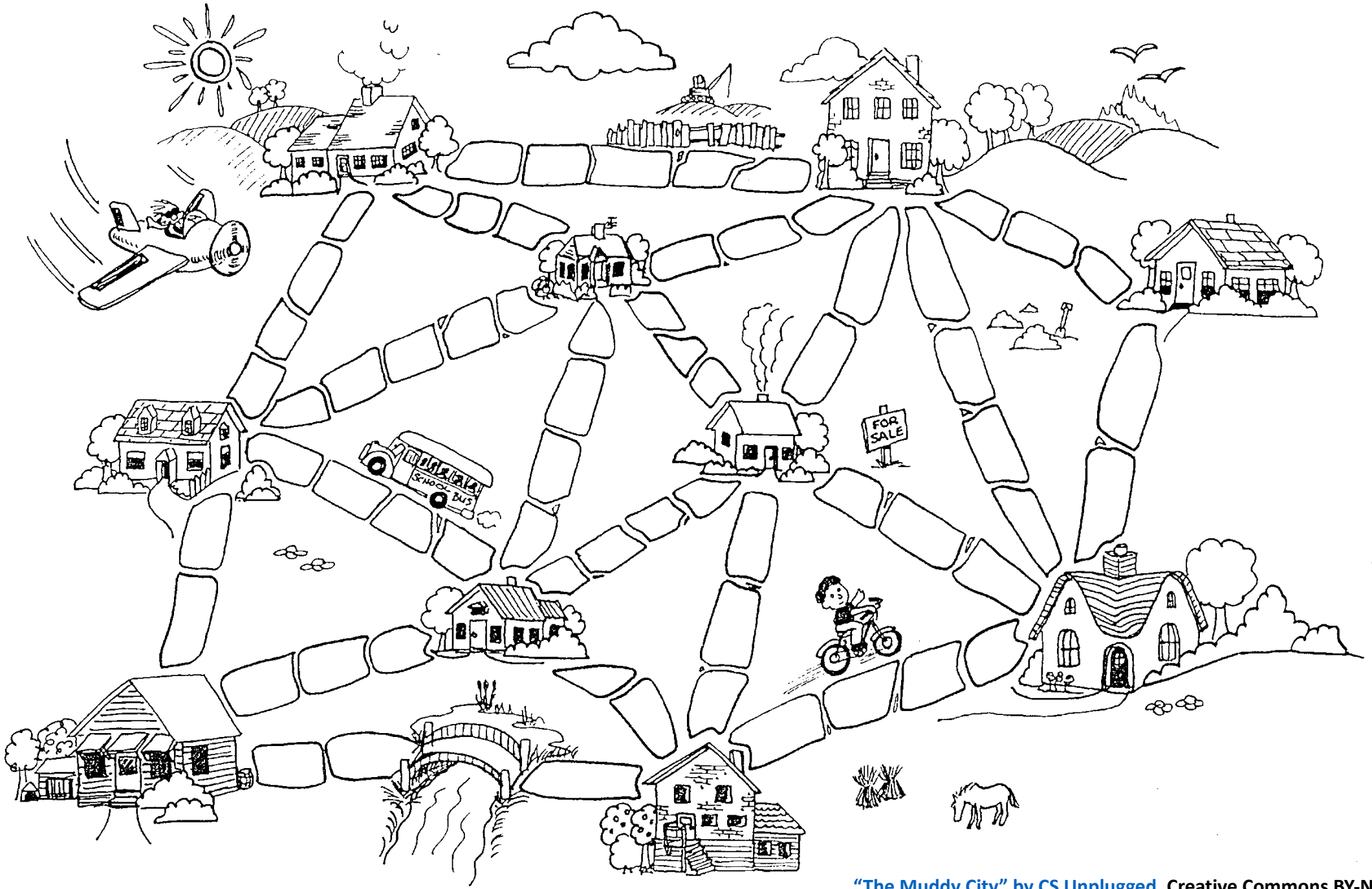
- Vertex:

- Edge:

Queries:

- Vertex:

- Edge:



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