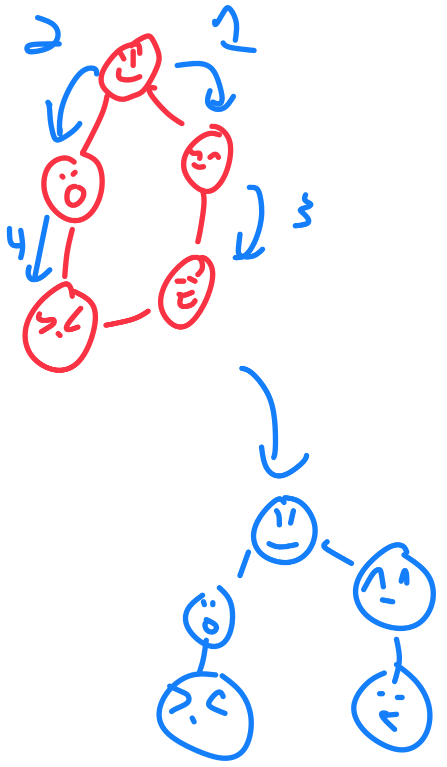
Data Structures
Graph Traversals

CS 225
Brad Solomon

October 28, 2024



Department of Computer Science

Changes to Class based on IEF

Will defer 'off-topic' questions until after class

Will use announcements channel more.

Will try out weekly 'bonus videos' summarizing content

Further reduced MP timeout to 6 hours

The return of the 'hangout' channel on Discord

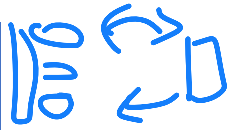
MP release dates are now on lecture's page

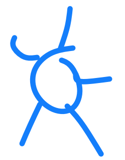
Learning Objectives

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)	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))





$deg(v) \rightarrow n-1$

Graph Traversals

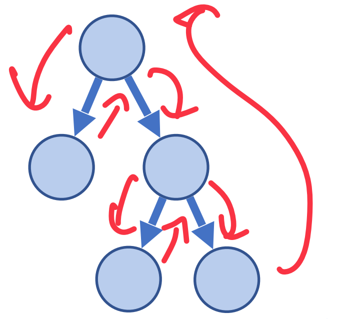
→ Solving a maze [finding path
→ Produce a spanning tree
→ Shortest path

] Find substructures

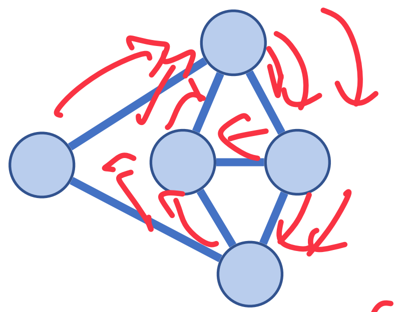
Objective: Visit every vertex and every edge in the graph.

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

Tree traversals won't work — lets compare:

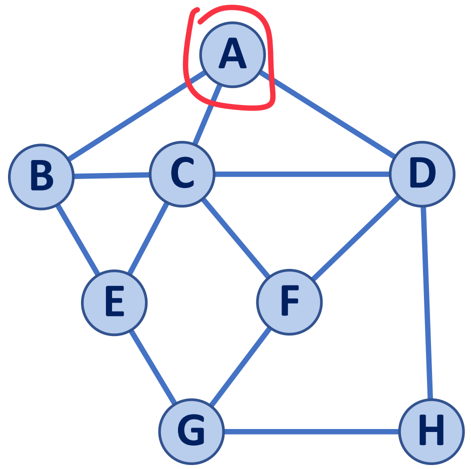


- Rooted
- Acyclic
- Notion of 'donesss'



- unrooted – choose starting pos
- Cyclic
- Hard to know when done!

Traversal: BFS



1) A starting vertex

2) A way to track visited nodes / edges

3) A graph structure (implementation)
↳ A way of getting neighbors

4) A way to track current / progress
↳ A queue!

Traversal: BFS

0) Initialize edge labels
vertex

distance
predecessor

↳ dist is distance from 'root'

↳ pred is path to get there

1) Initialize queue

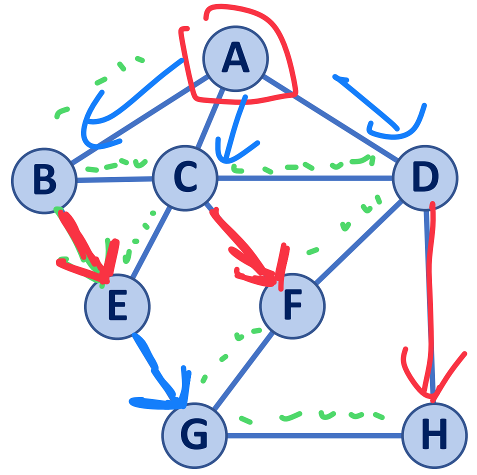
↳ Add root (and set label)

2) While queue not empty

↳ Dequeue front ('v')

↳ Process 'v'

↳ Add unvisited neighbors to queue (and label)



→ Discovery

.... Cross

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 Look at C D G
G	3	E	E F H
H	2	D	D G

Does C have dist?

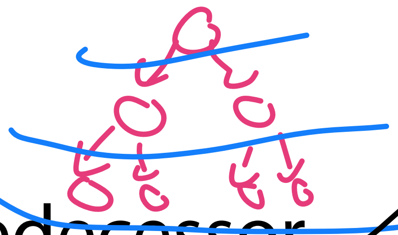
— labels tell me visited

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

↳ Queue tells me "current"

Know if visited ie dist | pred has value

Traversal: BFS



Graph implementation stores table

Vertex Node has member variable (depth, pred)

Initialize queue / depth / predecessor

or array!

or map!

While queue not empty:

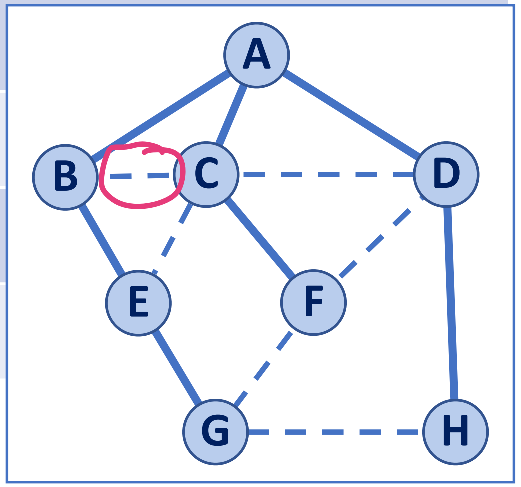
  Remove front vertex of queue

  Check if edge connects to new vertex

    Set dist / pred if new vertex

  Add unvisited edges to queue

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~~ ~~G~~ ~~H~~

Cross edges have meaning

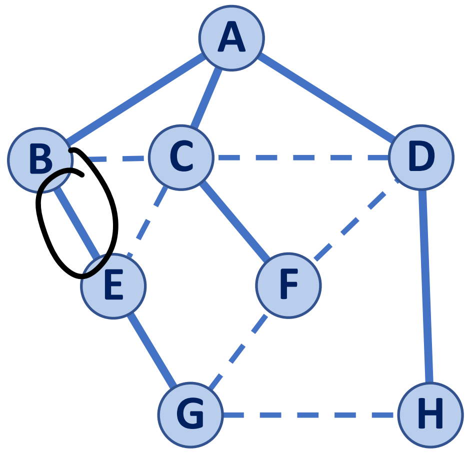
↳ we already saw that vertex through a shorter path

↳ Dist between vertices linked by cross is ≤ 1

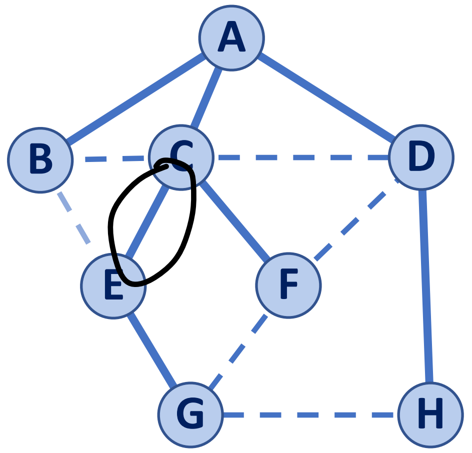
Traversal: BFS

implementation

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

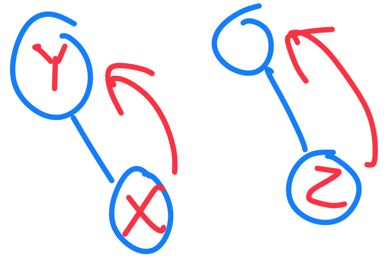
```
BFS(G):
  foreach (Vertex v : G.vertices()):
   setPred(v, NULL)
   setDist(v, -1)

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

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

Initialization

Initialization

If unvisited BFS(Graph, 'cot')

↳ Do BFS for each connected component



```
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)
```

Initialization

Look at all edges

If unvisited

If visited

```
 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?

Add int counter to loop @9

Loop runs equal to # connected components

↓

if case

```
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)
```

Cycle Detection?

Any cross edge is a loop! (undirected graph)

BFS Observations

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**?

$A \to D - H$, 2

Backtrace from H

BFS gives shortest path

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

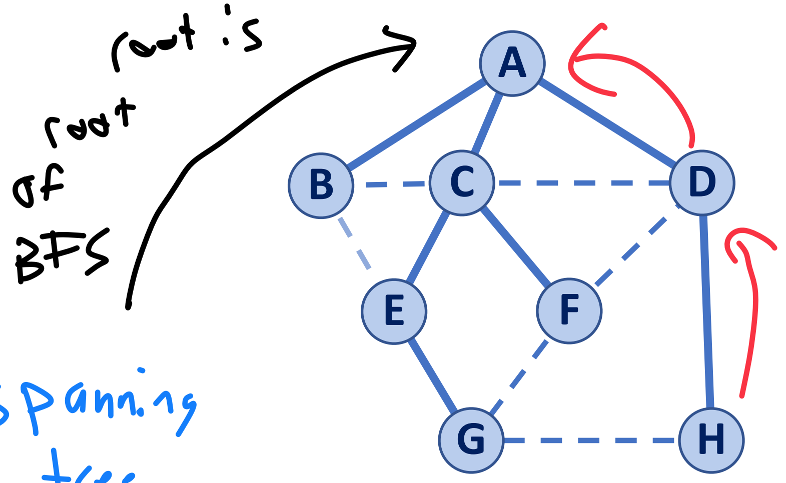
$E - G - H$, 2

... but only for paths from root

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

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

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



BFS Observations

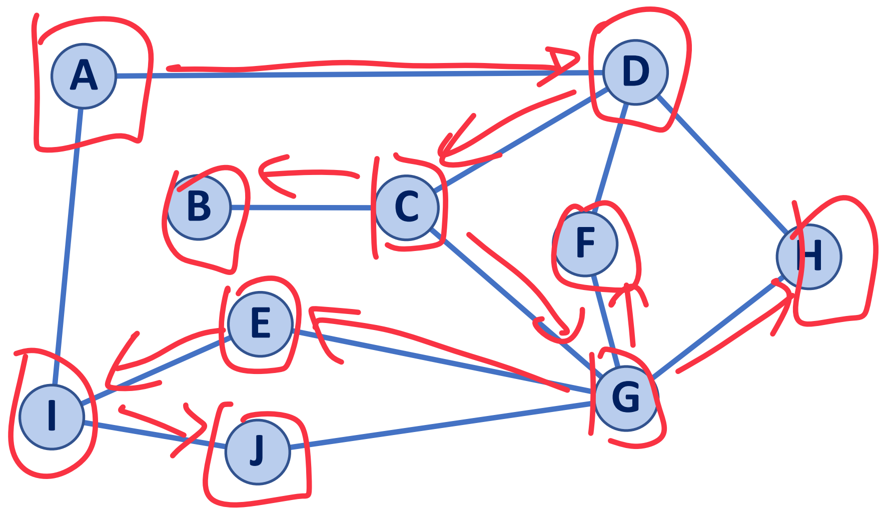
1. BFS can be used to count components

2. BFS can be used to detect cycles (cross edge)

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)| = 1**)

Traversal: DFS



0) Initialize dist/pred

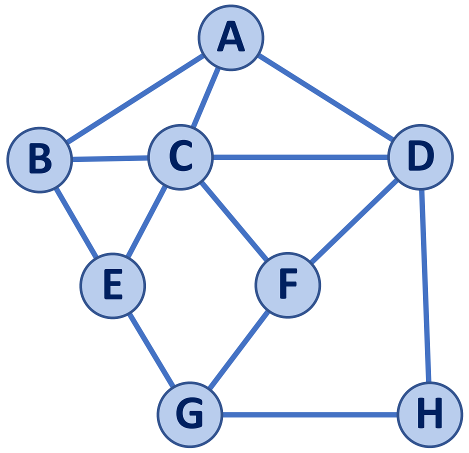
1) Init stack
↳ init w/ root

2) While stack not empty
↳ Peek A & get 1 unvisited child
↳ Add child to stack
↳ If no children unvisited
Pop from stack

~~H~~
~~F~~
~~J~~
~~I~~
~~E~~
G
~~B~~
C
D
Bottom A

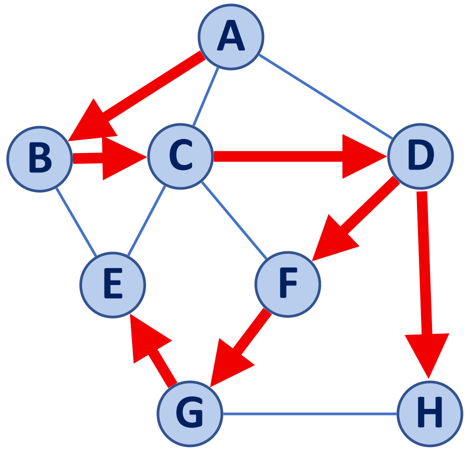
```
DFS(G):
  foreach (Vertex v : G.vertices()):
   setPred(v, NULL)
   setDist(v, -1)

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

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

```
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)
```



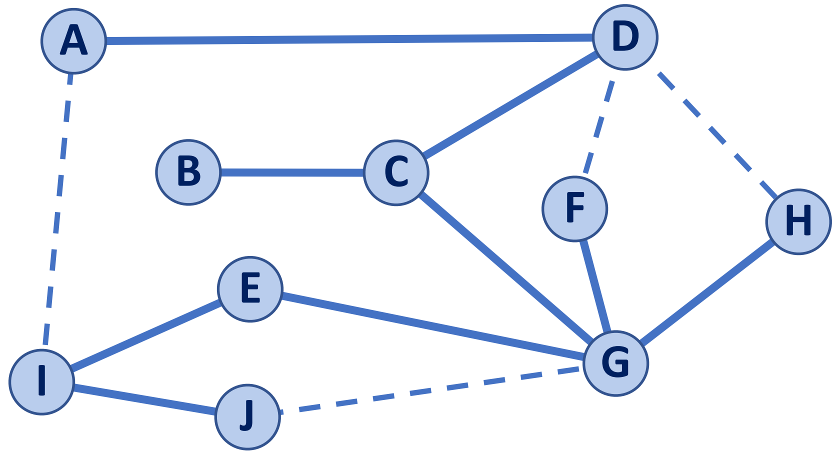
```
DFS(G, v):

    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



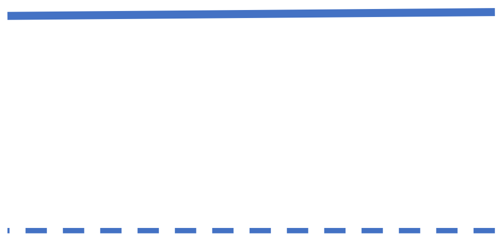
A B C D F G E H

Traversal: DFS

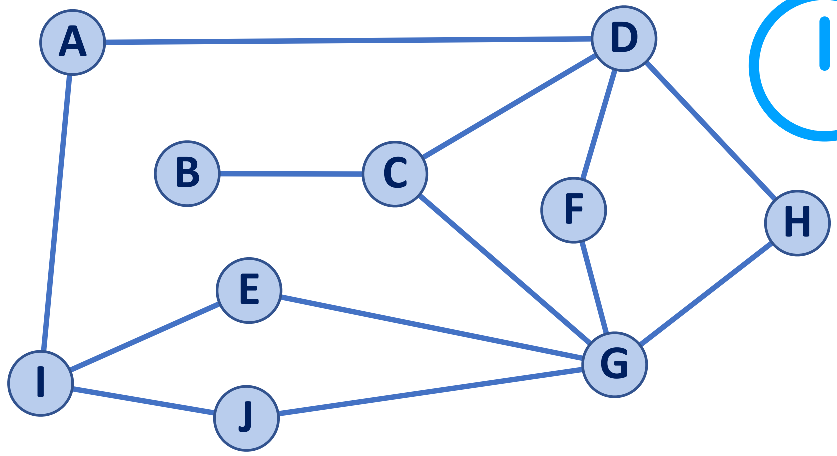


Does distance have meaning here?

Do our edge labels have meaning here?



Running time of DFS

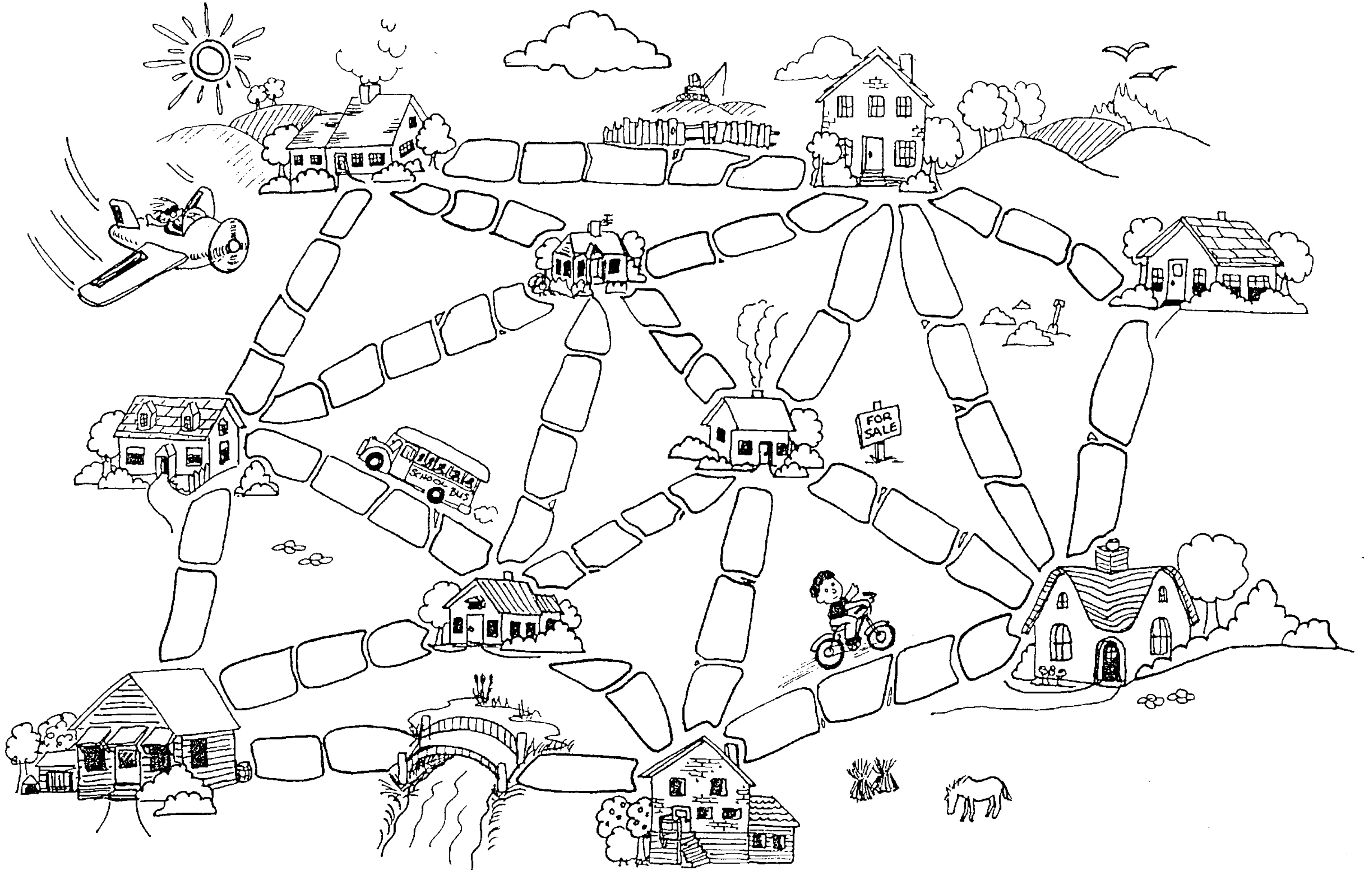


Labeling:

- Vertex:
- Edge:

Queries:

- Vertex:
- Edge:

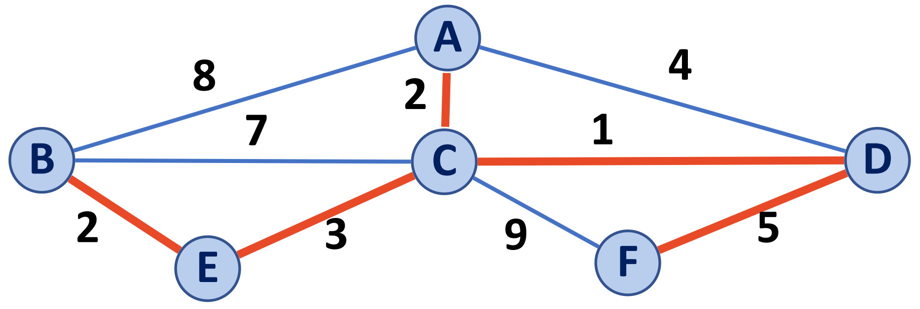

FOR SALE
SCHOOL BUS

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


A
B
C
D
E
F
8
2
4
7
1
2
3
9
5