

Data Structures

Graph Traversals and MST

CS 225

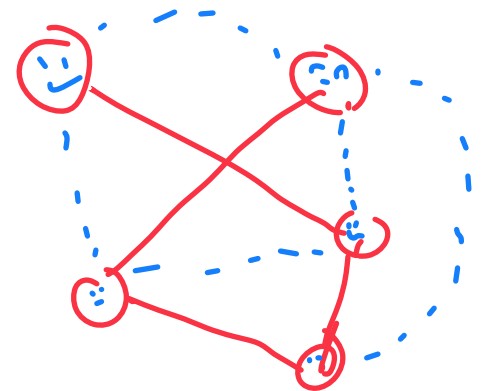
November 27, 2023

Brad Solomon & G Carl Evans



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science



Announcements

Project teams be sure to schedule your mid-project checkin soon!

This week's lab is extra credit lab

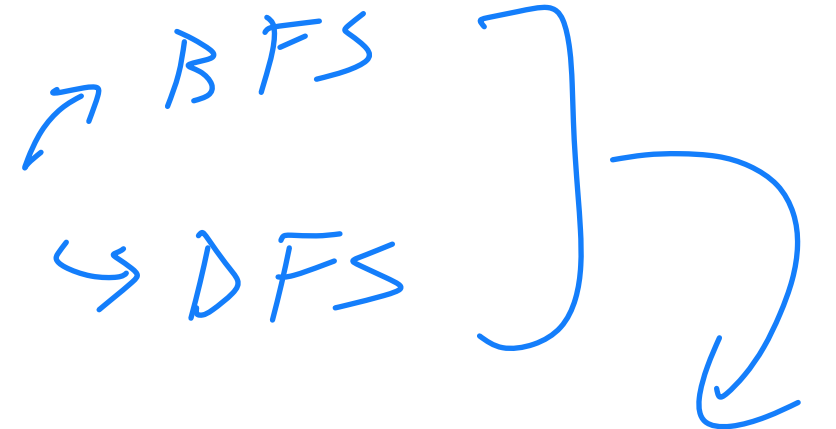
/ MP extra cred. 4

Exam 5 is happening now!

ICES Evaluations are open! If enough students submit, extra credit!

Learning Objectives

Review graph traversal algorithms



Introduce the minimum spanning tree (with weights)

Discuss Kruskal's MST Algorithm

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

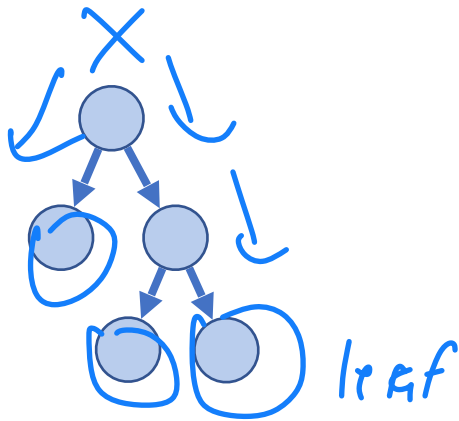
Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space	$n+m$	n^2	$n+m$
insertVertex(v)	1^*	n^*	1^*
removeVertex(v)	m^{**}	n	$\text{deg}(v)$
insertEdge(u, v)	1	1	1^*
removeEdge(u, v)	m	1	$\min(\text{deg}(u), \text{deg}(v))$
incidentEdges(v)	m	n	$\text{deg}(v)$
areAdjacent(u, v)	m	1	$\min(\text{deg}(u),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

Graph Traversals

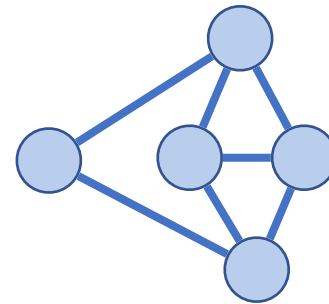
Objective: Visit every vertex and every edge in the graph.

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'



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

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)
```

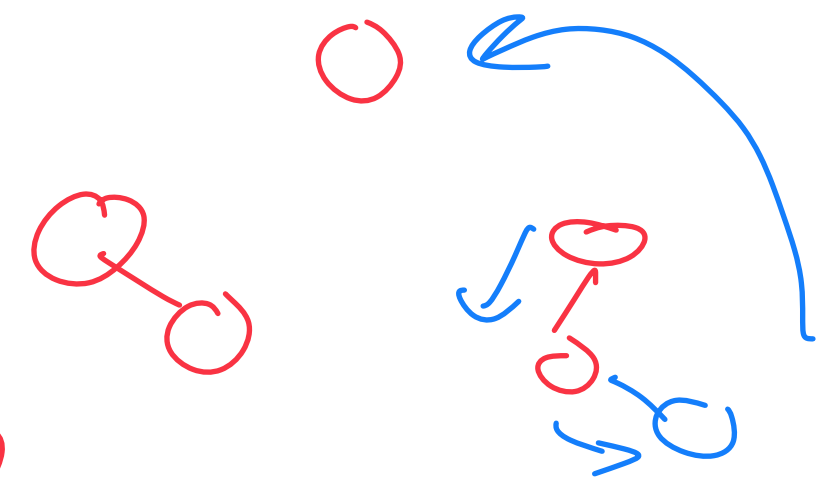
predecessor
distance

Count it to count components

Traversals in general

↳ Label on edges

↳ visit every vertex (and edge)



```

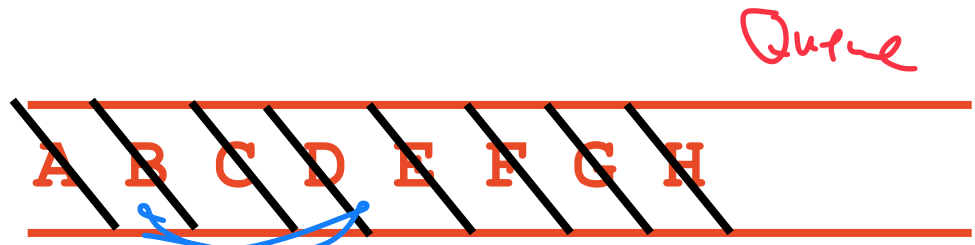
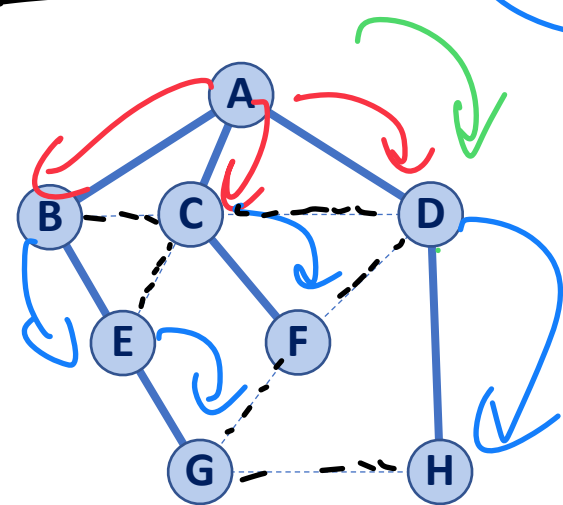
12 BFS (G, v):
13   Queue q
14   setDist(v, 0)
15   q.enqueue(v)
16
17   while !q.empty():
18     v = q.dequeue()
19
20     foreach (Vertex w : G.adjacent(v)):
21       if( getDist(w) == -1):
22         setLabel((v, w), DISCOVERY)
23         setPred(w, v)
24         setDist(w, v + 1)
25         q.enqueue(w)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

```

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

Discovery
 --- cross

MST is discovery edges





BFS Observations

1. BFS can be used to count components

↳ We do BFS once for each component
↳ count # of vertices w/ dist 0
↳ counter in loop

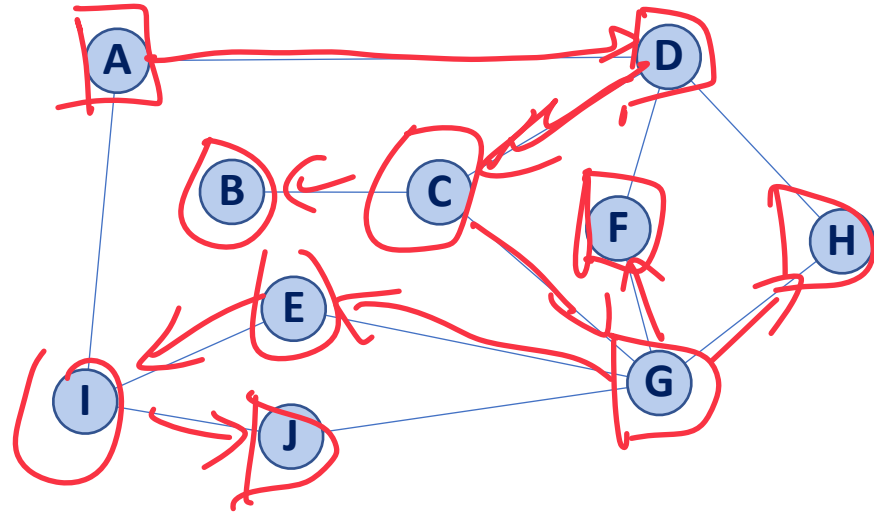
2. BFS can be used to detect cycles

↳ cross edges are cycles (at least one)

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

↳ is shortest path in unweighted graph

Traversal: DFS



Pop
then
push

vs
Depth?

A D C B G E I J F H

1) Init stack

↳ Insert some item

2) While stack not empty

Pop top item

Get next child

if have more children (unvisited)

↳ push back to stack

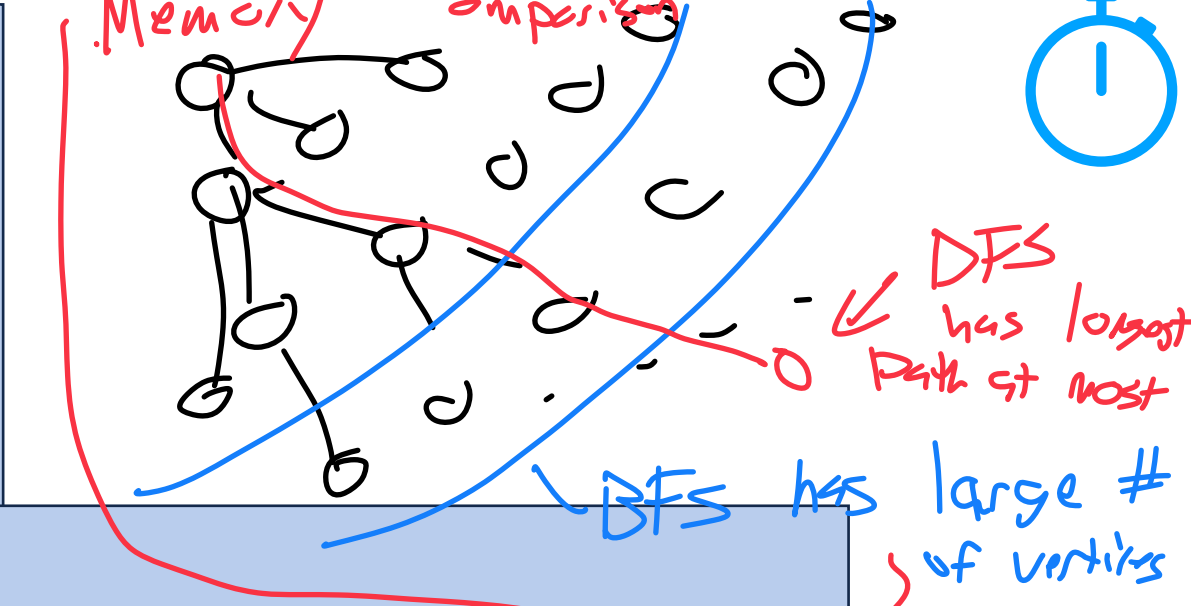
DFS (child)

~~K~~
~~H~~
~~J~~
~~I~~
~~F~~
~~E~~
~~D~~
~~C~~
~~B~~
Stack: A

```

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)

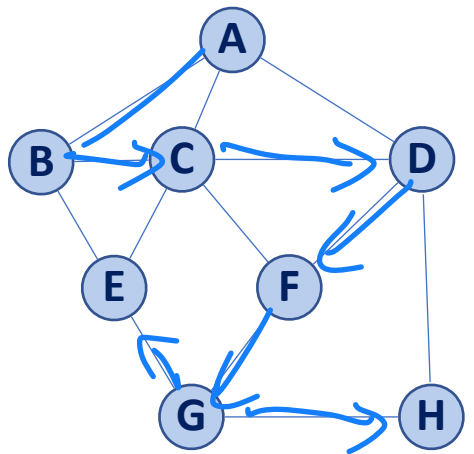
```



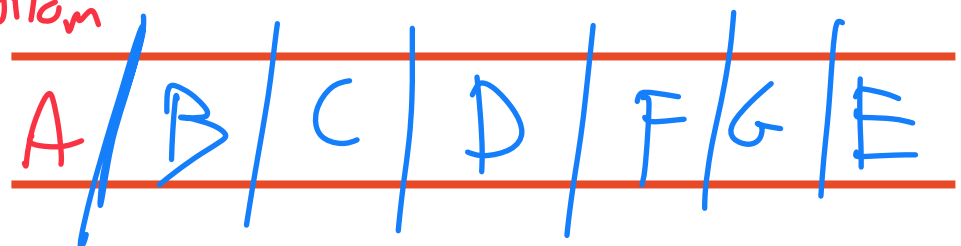
```

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)

```



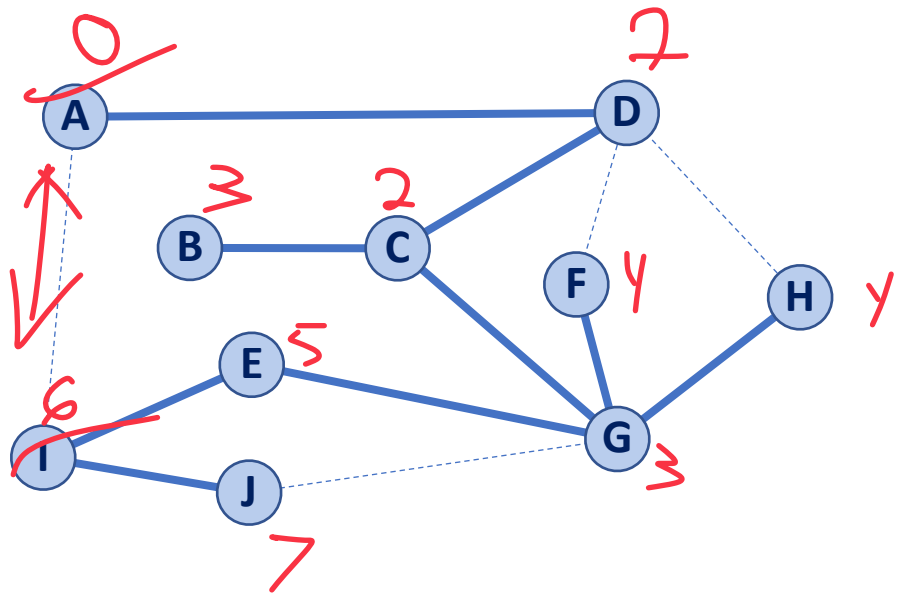
Bottom



A → [B, C, D] stop!

↳ DFS(B) → DFS(C) → DFS(D)

Traversal: DFS



 Discovery Edge

 Back Edge

Not cross b/c cross implies $\text{dist}(u,v) \leq 1$

Do we still make a spanning tree?

↳ Yes it makes an MST*

* on unweighted graphs

Does distance have meaning here?

↳ Has no real meaning!

~~↳ I have a path at least as long as largest distance~~

Do our edge labels have meaning here?

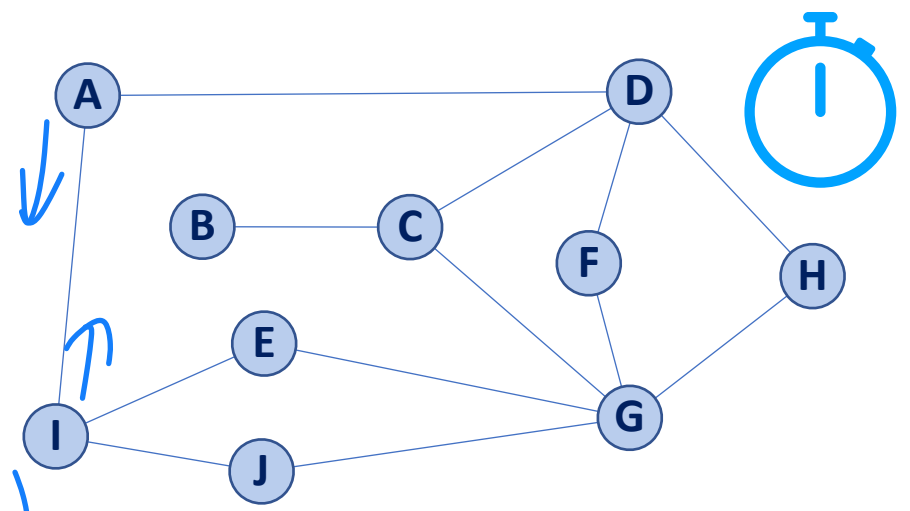
↳ Back edge still implies cycle

Running time of DFS $|V| = n$ $|E| = m$

Labeling:

- Vertex: $O(n)$
- Edge: $O(m)$

BFS: $O(n + m)$
DFS: $O(n + m)$



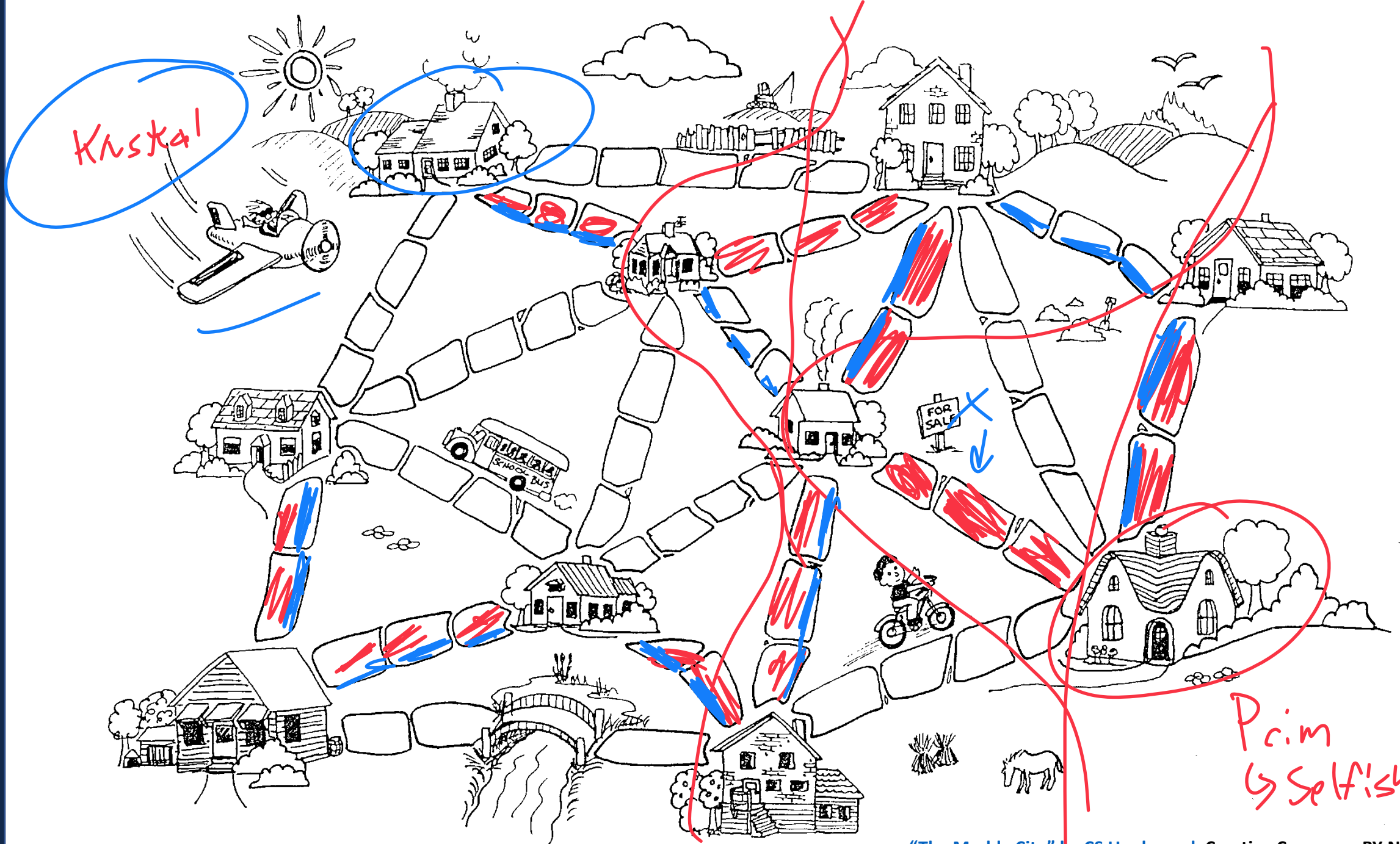
Traversal:

- Vertex: Each vertex looks at all neighbors
 look at all vertices

$$\sum_v \deg(v) = 2E = 2n \quad O(m)$$

- Edge: $O(m)$

<p><u>BFS</u></p> <ul style="list-style-type: none"> ↳ MST, <u>shortest path</u>, cycles, ↳ Memory Hog! 	<p><u>DFS</u> → mem is bounded by longest path</p> <ul style="list-style-type: none"> ↳ MST, cycles ↳ If early termination
---	--



Kuska!

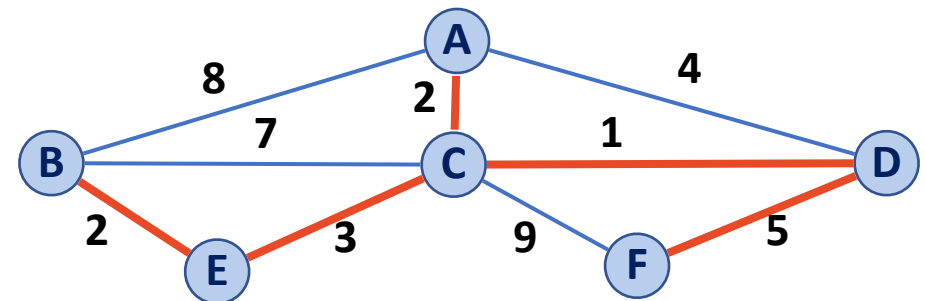
Prim
-> Selfish!

Minimum Spanning Tree Algorithms

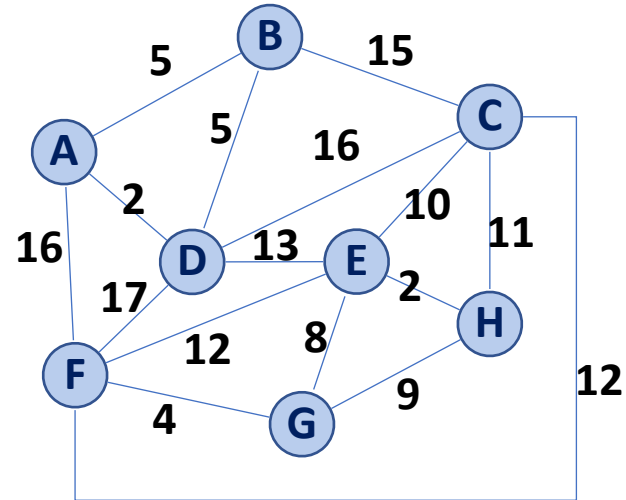
Input: Connected, undirected graph G with edge weights (unconstrained, but must be additive)

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



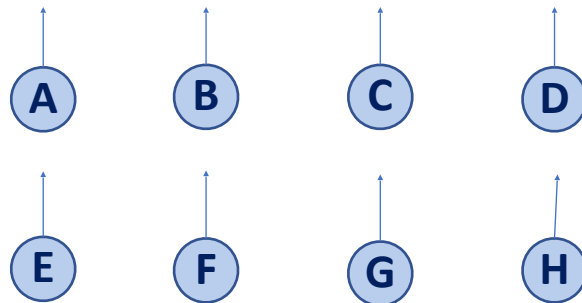
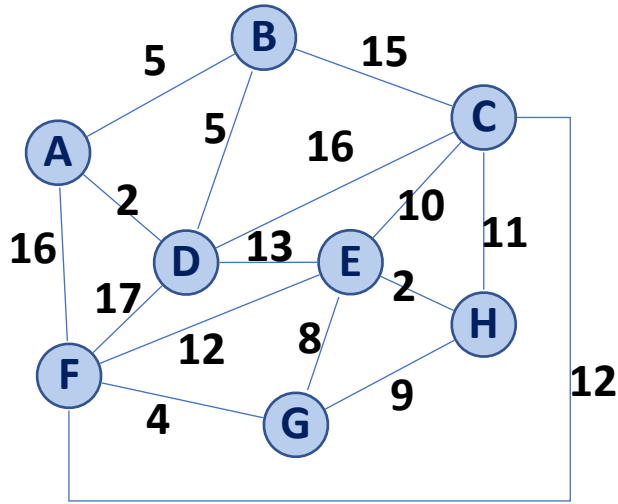
Kruskal's Algorithm



Kruskal's Algorithm

- 1) Build a **priority queue** on edges
- 2) Build a **disjoint set** on vertices

(A, D)
(E, H)
(F, G)
(A, B)
(B, D)
(G, E)
(G, H)
(E, C)
(C, H)
(E, F)
(F, C)
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)



Kruskal's Algorithm

Priority Queue:	Heap	Sorted Array
Building :7		
Each removeMin :12		

```
1 KruskalMST(G):
2   DisjointSets forest
3   foreach (Vertex v : G.vertices()):
4     forest.makeSet(v)
5
6   PriorityQueue Q // min edge weight
7   Q.buildFromGraph(G.edges())
8
9   Graph T = (V, {})
10
11  while |T.edges()| < n-1:
12    Vertex (u, v) = Q.removeMin()
13    if forest.find(u) != forest.find(v):
14      T.addEdge(u, v)
15      forest.union( forest.find(u),
16                  forest.find(v) )
17
18  return T
19
```

Kruskal's Algorithm



Priority Queue:	Total Running Time
Heap	
Sorted Array	

```
1 KruskalMST(G):
2   DisjointSets forest
3   foreach (Vertex v : G.vertices()):
4     forest.makeSet(v)
5
6   PriorityQueue Q // min edge weight
7   Q.buildFromGraph(G.edges())
8
9   Graph T = (V, {})
10
11  while |T.edges()| < n-1:
12    Vertex (u, v) = Q.removeMin()
13    if forest.find(u) != forest.find(v):
14      T.addEdge(u, v)
15      forest.union( forest.find(u),
16                  forest.find(v) )
17
18  return T
19
```