

Data Structures

Traversals 2 and Minimum Spanning Tree

CS 225

April 6, 2026

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Poetry Club

Tired of segfaults and memory errors? Need a break from studying, or interested in the humanities?



- *Tuesday, April 7
6-7PM*
- *English Building
149*
- *Snacks Provided*
- *Fill out QR for
snack
headcount*



Learning Objectives

Continue graph traversal algorithms (mostly DFS)

Introduce the minimum spanning tree (with weights)

Introduce Kruskal's / Prim's MST Algorithms

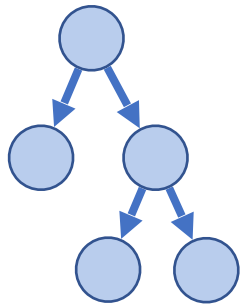
Begin discussing implementation of Kruskal's

Graph Traversals

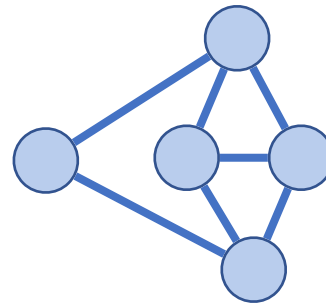
Objective: Visit every vertex and every edge in the graph.

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'



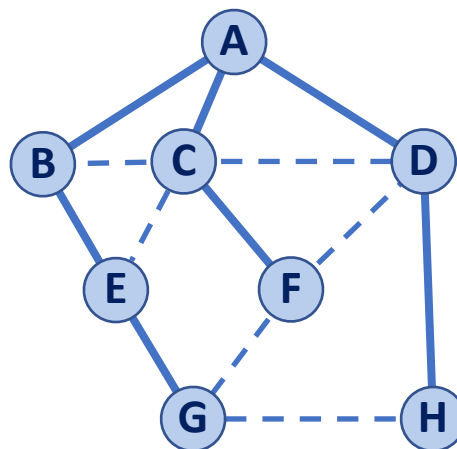
- No root (any start position valid)
- Cycles
- No obvious 'endpoint'

```

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)

```

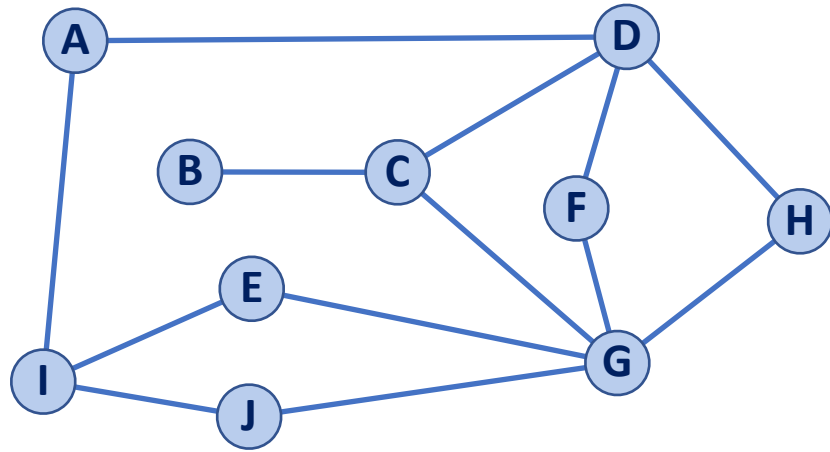
v	d	P	Adjacent Edges
A	0	-	B C D
B	1	A	A C E
C	1	A	A B D E F
D	1	A	A C F H
E	2	B	B C G
F	2	C	C D G
G	3	E	E F H
H	2	D	D G



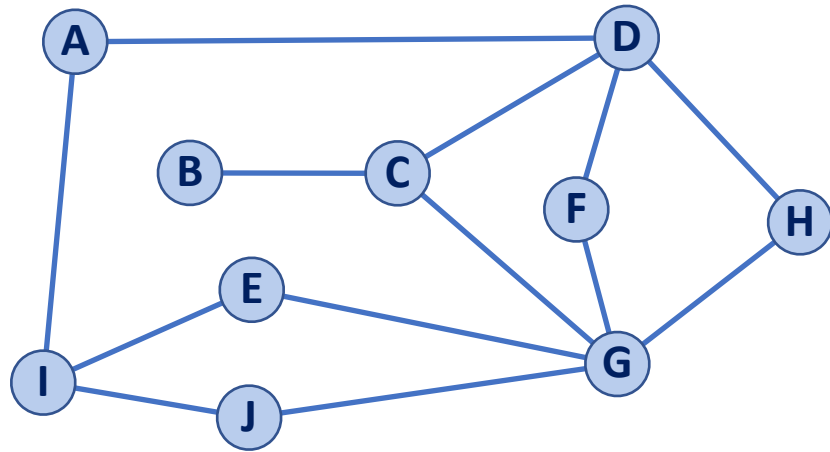
BFS Observations

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 ($|\mathbf{d(u)} - \mathbf{d(v)}| = 1$)

Traversal: DFS



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 child of tmp

Add to stack

dist = tmp.dist+1

pred = tmp

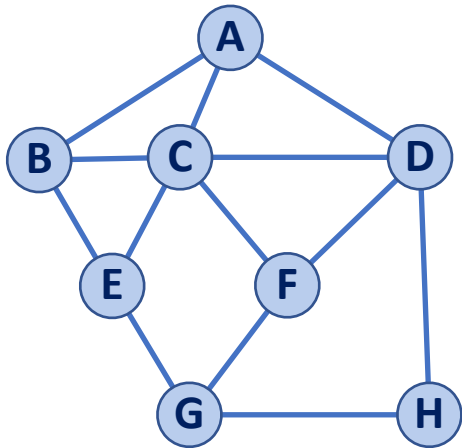
If no unvisited children

stack.pop()

A

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)
```

```

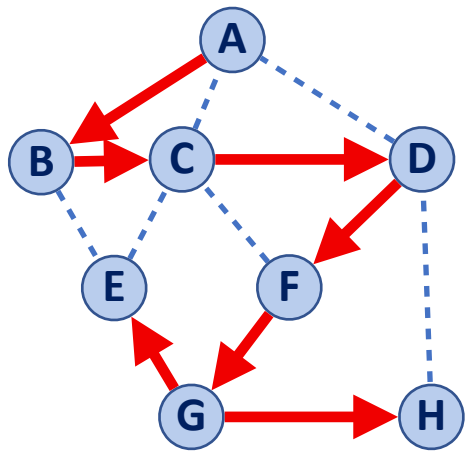
12 DFS (G, v) :
13
14   foreach (Vertex w : G.adjacent(v)) :
15     if( getDist(w) == -1):
16       setLabel((v, w), DISCOVERY)
17       setPred(w, v)
18       setDist(w, v + 1)
19       DFS (G, w)
20     else:
21       setLabel((v, w), BACK)

```

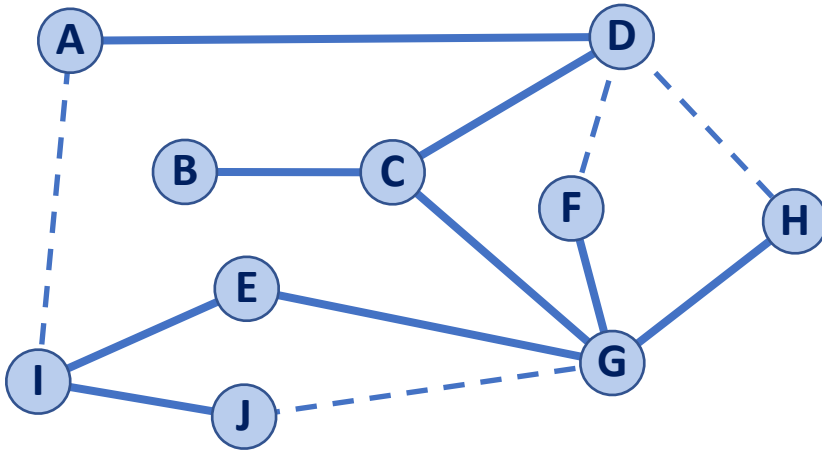
v	d	P	Adjacent Edges
A	0	-	B C D
B	1	A	A C E
C	2	B	A B D E F
D	3	C	A C F H
E	6	G	B C G
F	4	D	C D G
G	5	F	E F H
H	6	G	D G

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

State of stack when H is added:



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?

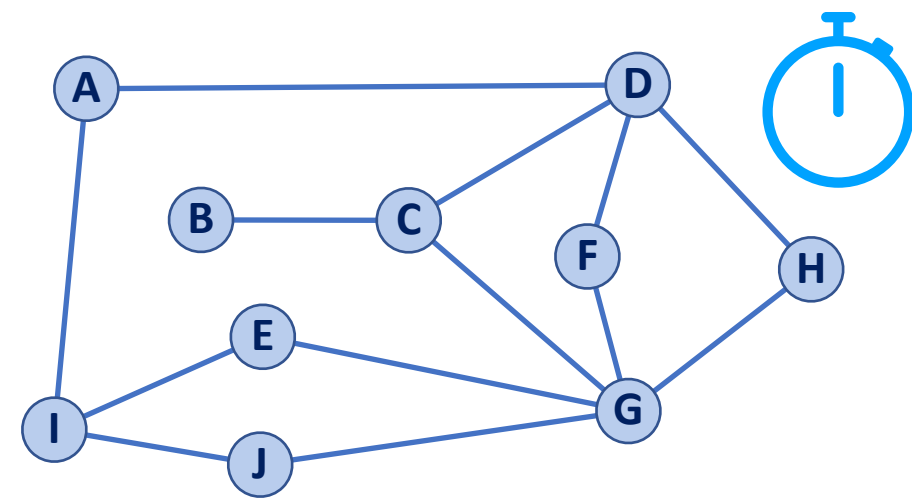
Running time of DFS

Labeling:

- Vertex:
- Edge:

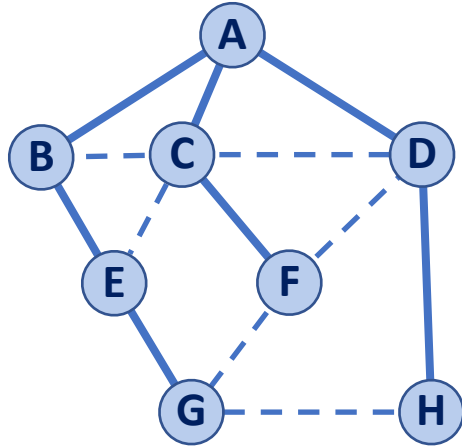
Traversal:

- Vertex:
- Edge:



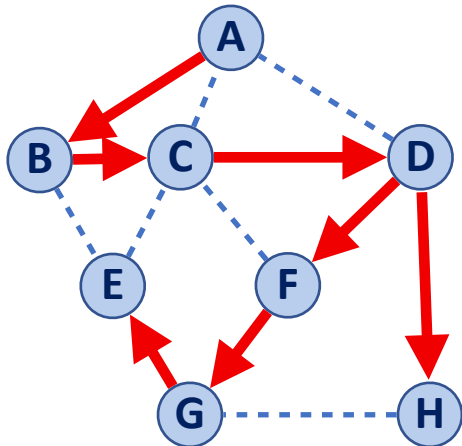
Efficiency: DFS vs BFS

BFS:



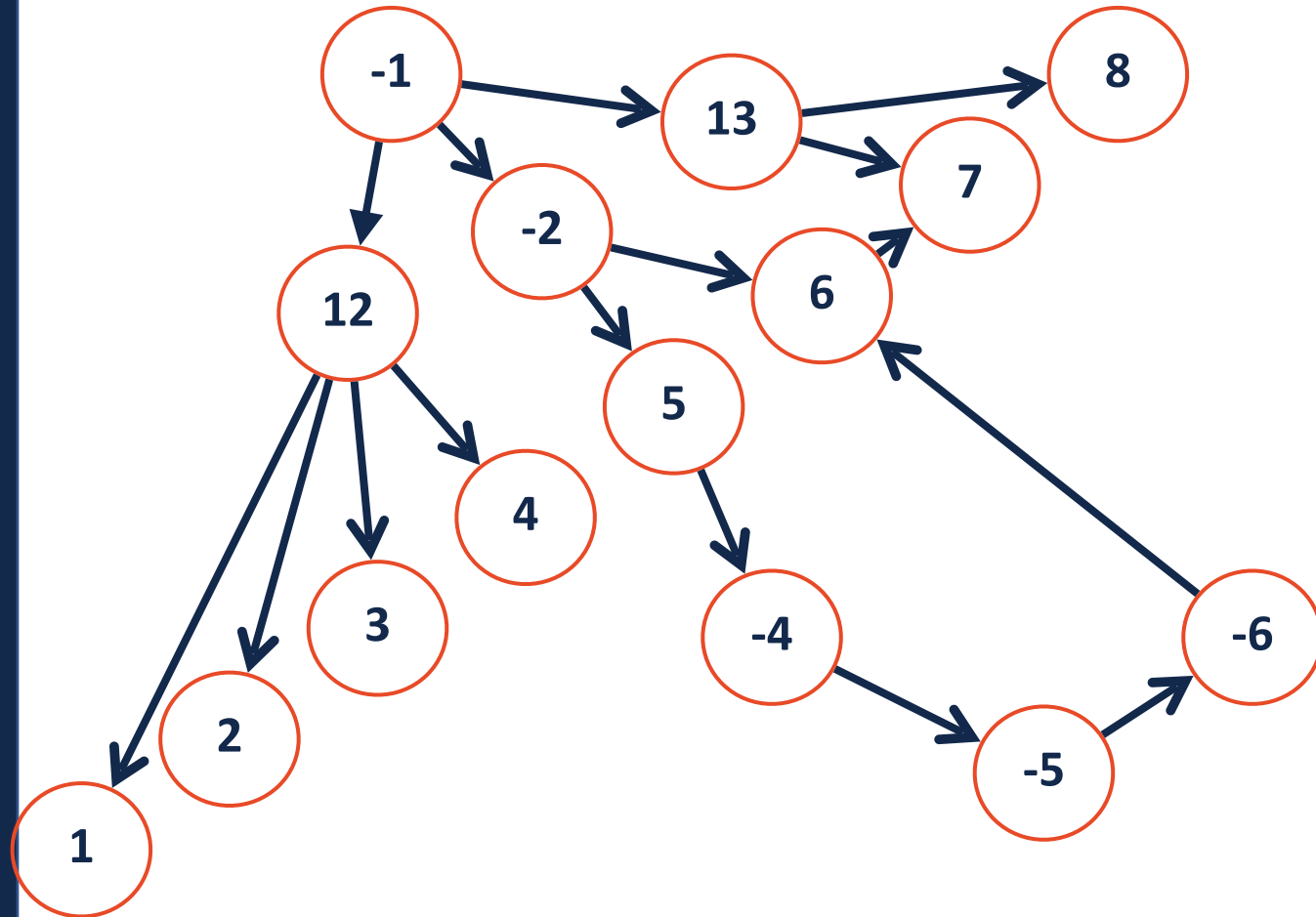
A B C D E F H G

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

Solves unweighted MST

Solves shortest path

Solves cycle detection

Memory bounded by width

DFS

Solves unweighted MST

Solves cycle detection

Memory bounded by longest path

Graph Algorithms: Minimum Spanning Trees



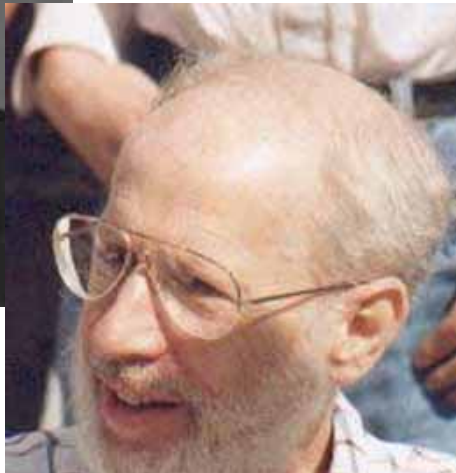
Vojtěch Jarník (1897-1970)

First wrote algorithm for MST

Robert Prim (1921 — 2021)

Rediscovered Jarník's algorithm

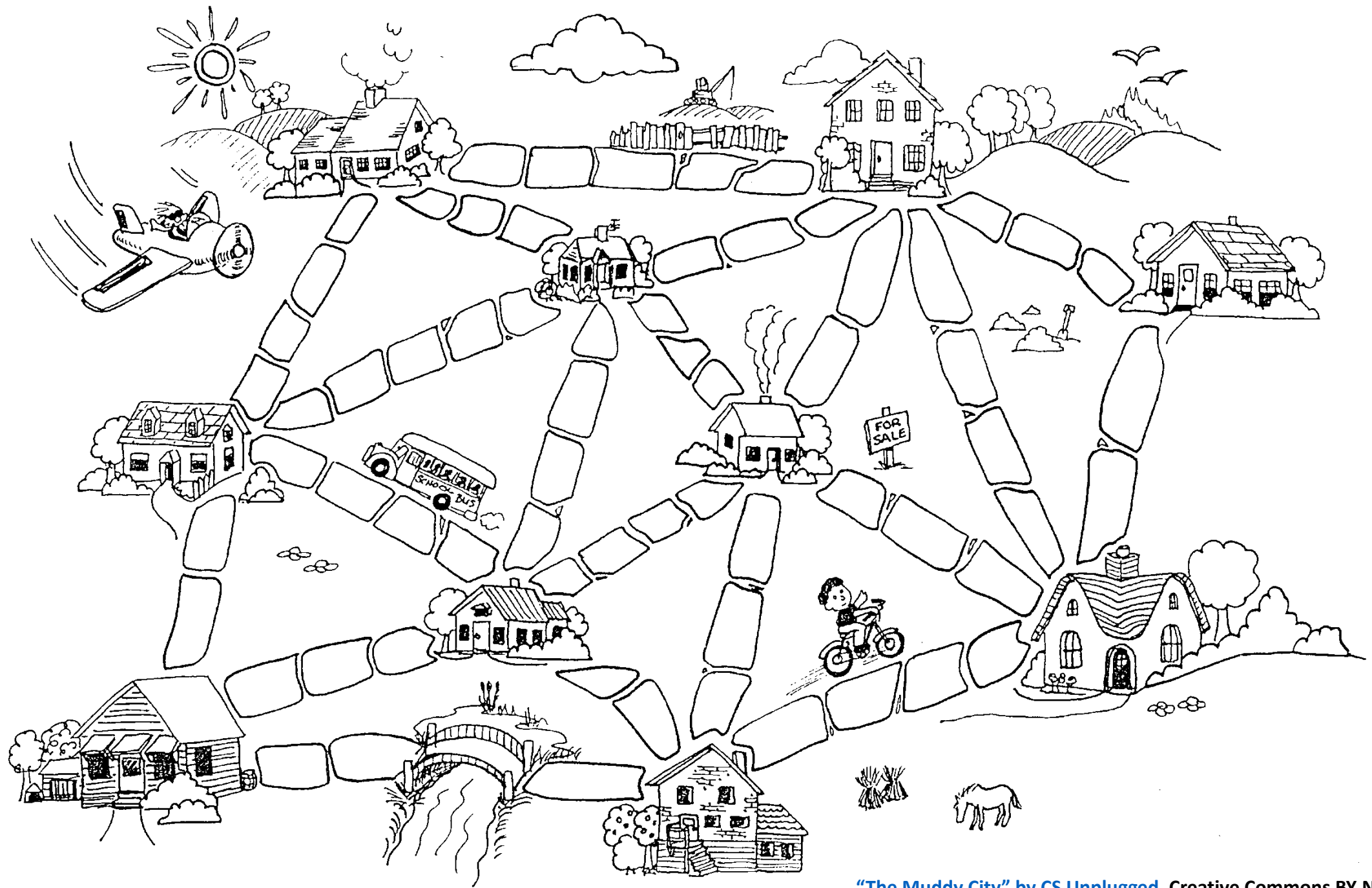
(It's named after Prim now though)



Joseph Kruskal (1928 — 2010)

Worked in Bell Labs with Prim

Came up with his own MST algorithm

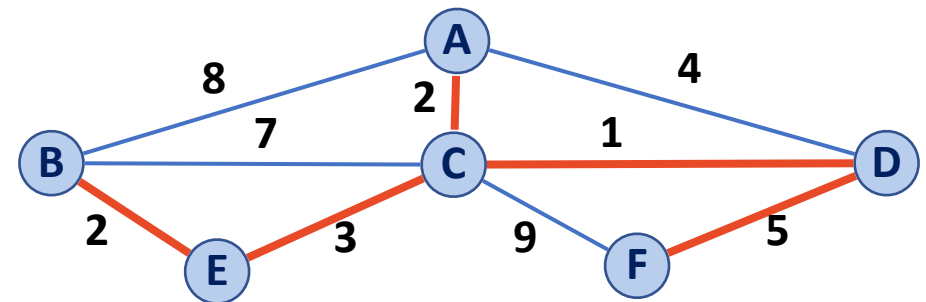


Minimum Spanning Tree Algorithms

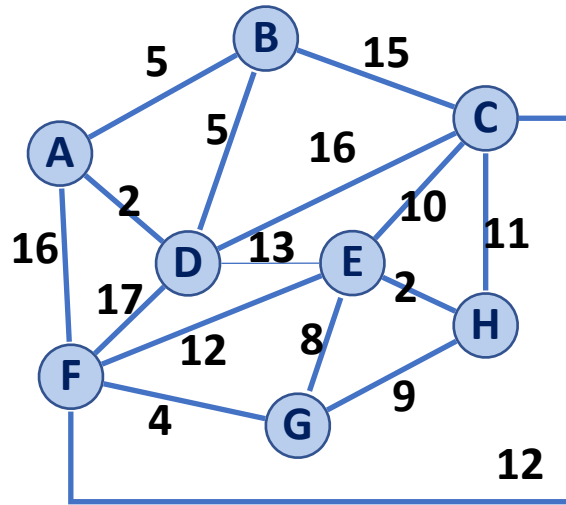
Input: Connected, undirected graph G with edge weights (unconstrained, but must be additive)

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

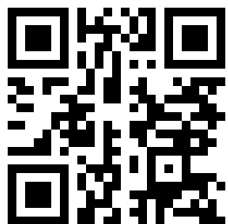


Kruskal's Algorithm *(A graph algorithm for the MST problem)*

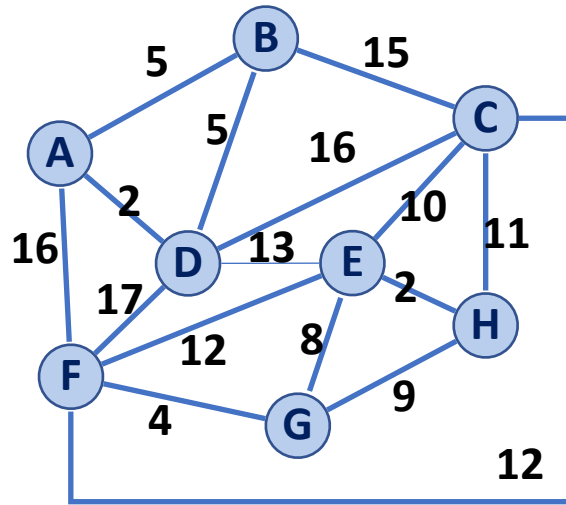


What information do I need to get efficiently?

- 1) The global minimum edge weight
- 2) Edge connections given a vertex
- 3) If two vertices are already connected
- 4) A visited list of vertices



Kruskal's Algorithm *(A graph algorithm for the MST problem)*

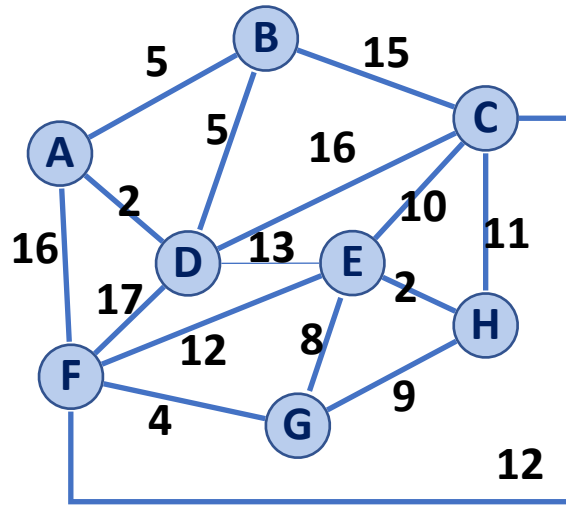


What information do I need to get efficiently?

- 1) The global minimum edge weight
- 3) If two vertices are already connected

What does this tell me about my implementation choices?

Kruskal's Algorithm *(A graph algorithm for the MST problem)*



What information do I need to get efficiently?

- 1) The global minimum edge weight
- 3) If two vertices are already connected

What does this tell me about my implementation choices?

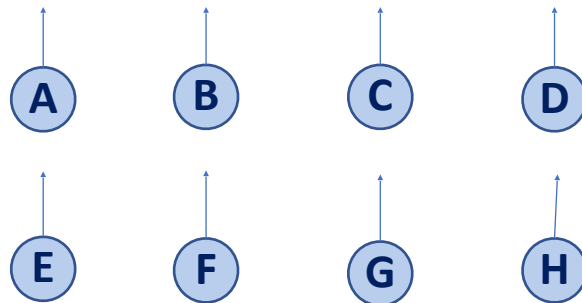
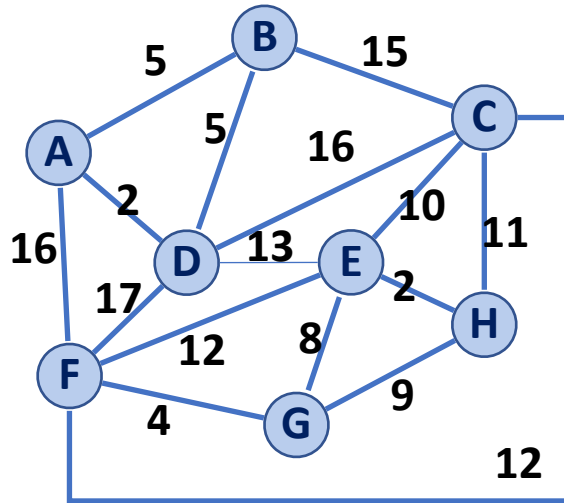
- 1) We need a **priority queue** of edges (sorted by weight)
- 2) We need a **disjoint set** of vertices

Kruskal's Algorithm

1) Build a **priority queue** on edges

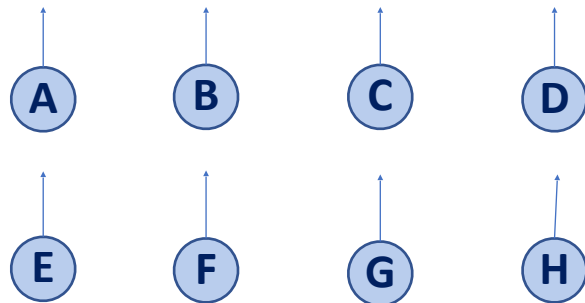
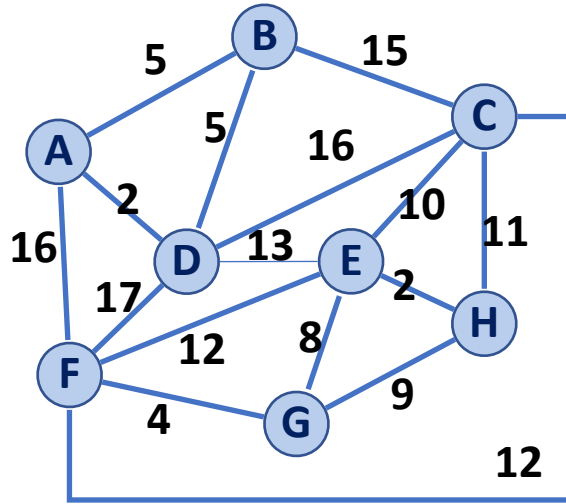
2) Build a **disjoint set** on vertices

(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

(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)



- 1) Build a **priority queue** on edges
A minheap or *A sorted array*
- 2) Build a **disjoint set** on vertices
All vertices start as their own set
- 3) Loop through min edges
If edge connects two disjoint sets
Union sets and record edge in MST
- 4) Stop when:
N-1 edges recorded
Only a single disjoint set remains

Kruskal's Algorithm

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

What is the Big O?

```
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

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

What is the Big O?

```
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
```

2 — 4: $O(n)$

6 — 7: Heap: $O(m)$
Sorted List: $O(m \log m)$

11: $m \times \langle 12-17 \rangle$

12—17: Heap: $O(\log m)$
Sorted List: $O(1)$

Disjoint set we treat as $O(1)$ b/c path compression w/ smart union

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:	Heap	Sorted Array
Building :7	$O(m)$	$O(m \log m)$
Each removeMin :12	$O(m \log m)$	$O(m)$

Both result in $m + m \log m$

Why is heap good?

If edge weights can change!

Why is sorted array good?

Sorted array not destroyed and can be useful in other algorithms!

```
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
```

