



# CS 225

## Data Structures

*November 17 – Graph Traversals and MST*

*G Carl Evans*





## BFS Observations

**Obs. 1:** Traversals can be used to count components.

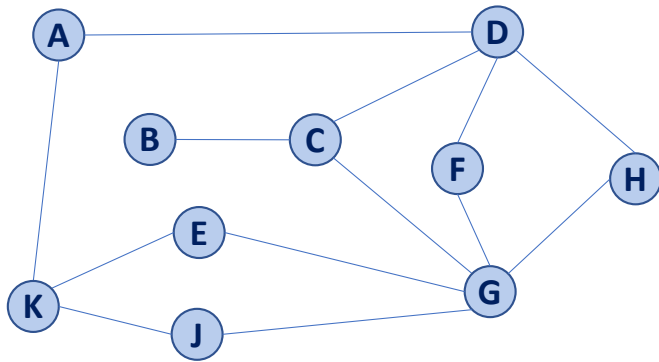
**Obs. 2:** Traversals can be used to detect cycles.

**Obs. 3:** In BFS,  $d$  provides the shortest distance to every vertex.

**Obs. 4:** In BFS, the endpoints of a cross edge never differ in distance,  $d$ , by more than 1:

$$|d(u) - d(v)| = 1$$

# Traversal: DFS

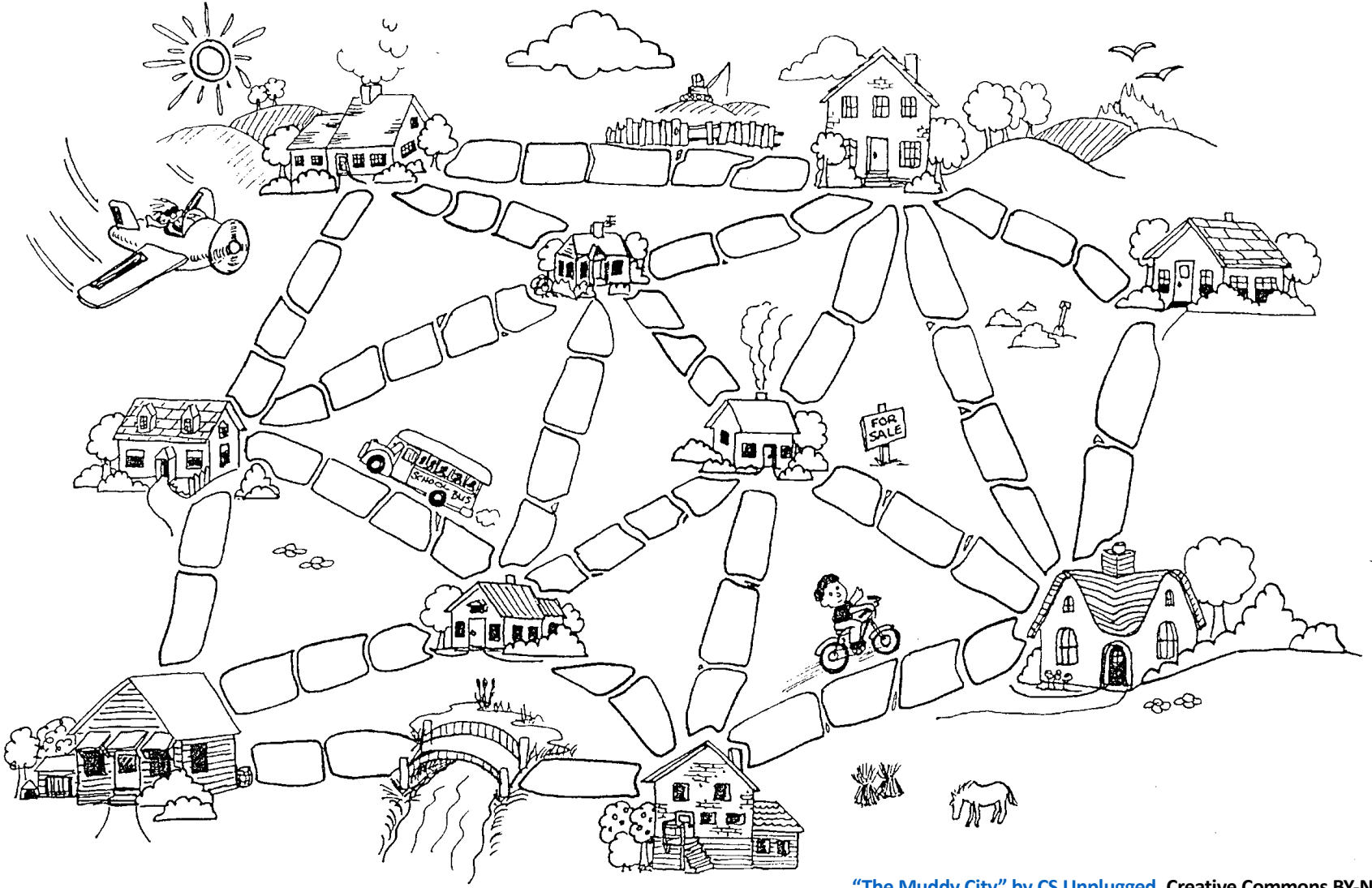


```
1 BFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and cross edges
5
6   foreach (Vertex v : G.vertices()):
7     setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9     setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11    if getLabel(v) == UNEXPLORED:
12      BFS(G, v)
```

```
14 BFS(G, v) :
15   Queue q
16   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20     v = q.dequeue()
21     foreach (Vertex w : G.adjacent(v)):
22       if getLabel(w) == UNEXPLORED:
23         setLabel(v, w, DISCOVERY)
24         setLabel(w, VISITED)
25         q.enqueue(w)
26       elseif getLabel(v, w) == UNEXPLORED:
27         setLabel(v, w, CROSS)
```

```
1 DFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and back edges
5
6   foreach (Vertex v : G.vertices()):
7     setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9     setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11    if getLabel(v) == UNEXPLORED:
12      DFS(G, v)
```

```
14 DFS(G, v) :
15 Queue q
16   setLabel(v, VISITED)
17 q.enqueue(v)
18
19 while !q.empty():
20 v = q.dequeue()
21   foreach (Vertex w : G.adjacent(v)):
22     if getLabel(w) == UNEXPLORED:
23       setLabel(v, w, DISCOVERY)
24       setLabel(w, VISITED)
25       DFS(G, w)
26     elseif getLabel(v, w) == UNEXPLORED:
27       setLabel(v, w, BACK)
```



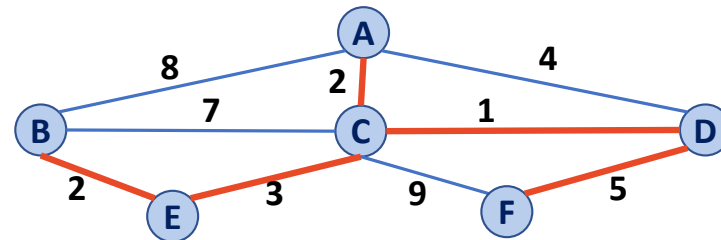
["The Muddy City"](#) by CS Unplugged, Creative Commons BY-NC-SA 4.0

# Minimum Spanning Tree Algorithms

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

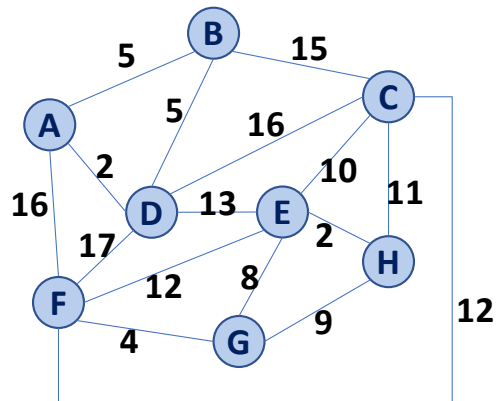
**Output:** A graph  $G'$  with the following properties:

- $G'$  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, D)

(E, H)

(F, G)

(A, B)

(B, D)

(G, E)

(G, H)

(E, C)

(C, H)

(E, F)

(F, C)

(D, E)

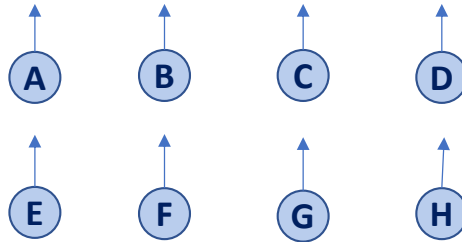
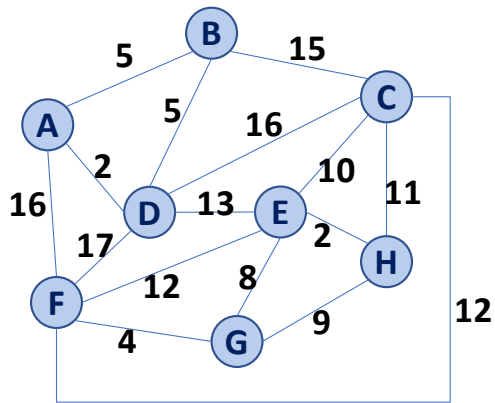
(B, C)

(C, D)

(A, F)

(D, F)

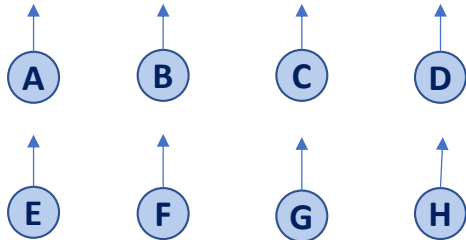
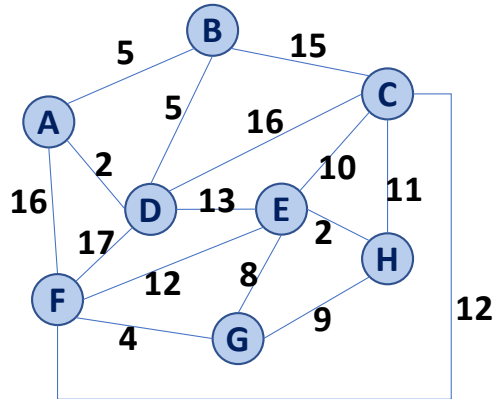
# Kruskal's Algorithm



(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)

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

```

# Kruskal's Algorithm

Priority Queue:	Heap	Sorted Array
<b>Building</b> :7-9		
<b>Each removeMin</b> :13		

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

# Kruskal's Algorithm

Priority Queue:	Total Running Time
Heap	
Sorted Array	

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