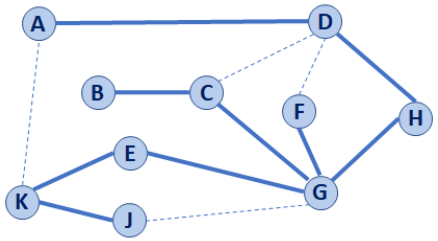
CS 225

#38: Kruskal's Algorithm

December 1, 2017 · *Wade Fagen-Ulmschneider*

Depth First Search (DFS) Traversal



DFS Traversal Starting at A, moving clockwise around edges.

```
 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         BFS(G, v)
13
14  DFS(G, v):
15    setLabel(v, VISITED)
16
17    foreach (Vertex w : G.adjacent(v)):
18      if getLabel(w) == UNEXPLORED:
19        setLabel(v, w, DISCOVERY)
20        q.enqueue(w)
21     elseif getLabel(v, w) == UNEXPLORED:
22        setLabel(v, w, BACK)
```

Running Time of DFS:

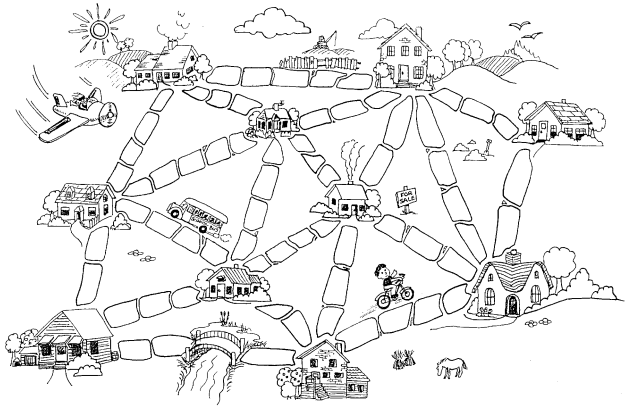
Labeling:

- Vertex:
- Edge:

Queries:

- Vertex:
- Edge:

Spanning Trees



"The Muddy City" by CS Unplugged, **Creative Commons BY-NC-SA 4.0**

Q: What road should we build first?

Q: What strategy should we use to build the next road?

A **Spanning Tree** on a connected graph **G** is a subgraph, **G'**, such that:

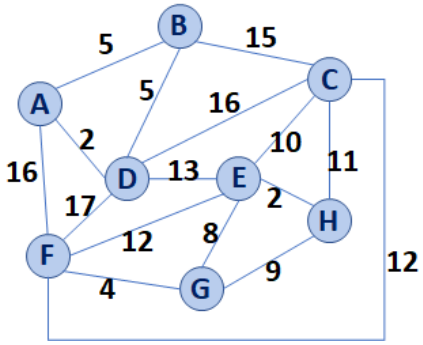
1. Every vertex is G is in G' and
2. G' is connected with the minimum number of edges

This construction will always create a new graph that is a tree (connected, acyclic graph) that spans G.

A **Minimum Spanning Tree** is a spanning tree with the minimal total edge weights among all spanning trees.

- Every edge must have a weight
  - The weights are unconstrained, except they must be additive *(eg: can be negative, can be non-integers)*
- Output of a MST algorithm produces G':
  - G' is a spanning graph of G
  - G' is a tree
  - 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)

Pseudocode for Kruskal's MST Algorithm

```
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 Running Time Analysis

We have multiple choices on which underlying data structure to use to build the Priority Queue used in Kruskal's Algorithm:

Priority Queue Implementations:	Heap	Sorted Array
Building :7-9		
Each removeMin :13		

Based on our algorithm choice:

Priority Queue Implementation:	Total Running Time
Heap	
Sorted Array	

CS 225 – Things To Be Doing:

1. Exam #12 (programming) starts Monday
2. MP7 extra credit deadline on Monday (+14 EC)
3. lab_graphs due Sunday
4. Multi-day "puzzle" POTDs available M/W/F