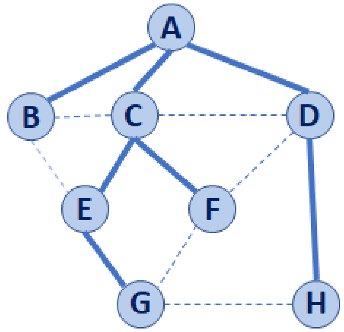
CS 225

#36: Minimum Spanning Trees

April 18, 2018 · *Wade Fagen-Ulmschneider*

BFS Graph Observations



1. Does our implementation handle disjoint graphs? How?

    a. How can we modify our code to count components?

2. Can our implementation detect a cycle? How?

    a. How can we modify our code to store update a private member variable **`cycleDetected_`**?

3. What is the running time of our algorithm?

4. What is the shortest path between **A** and **H**?

5. What is the shortest path between **E** and **H**?

    a. What does that tell us about BFS?

6. What does a cross edge tell us about its endpoints?

7. What structure is made from discovery edges in **G**?

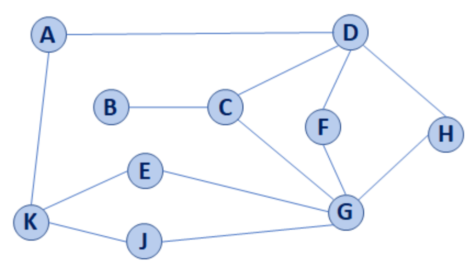
Pseudocode for DFS

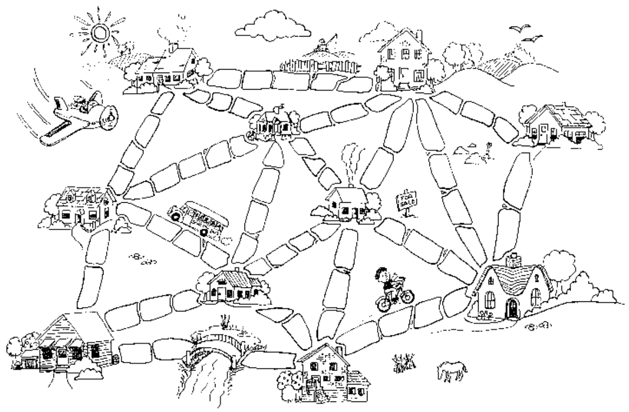
```
BFS(G):
  Input: Graph, G
  Output: A labeling of the edges on
      G as discovery and cross edges

  foreach (Vertex v : G.vertices()):
    setLabel(v, UNEXPLORED)
  foreach (Edge e : G.edges()):
    setLabel(e, UNEXPLORED)
  foreach (Vertex v : G.vertices()):
    if getLabel(v) == UNEXPLORED:
       BFS(G, v)

BFS(G, v):
  Queue q
  setLabel(v, VISITED)
  q.enqueue(v)

  while !q.empty():
    v = q.dequeue()
    foreach (Vertex w : G.adjacent(v)):
      if getLabel(w) == UNEXPLORED:
         setLabel(v, w, DISCOVERY)
         setLabel(w, VISITED)
         q.enqueue(w)
      elseif getLabel(v, w) == UNEXPLORED:
         setLabel(v, w, CROSS)
```

<u>Big Ideas: Utility of a BFS Traversal</u>

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

Depth First Search – A Modification to BFS





"The Muddy City" by CS Unplugged,

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

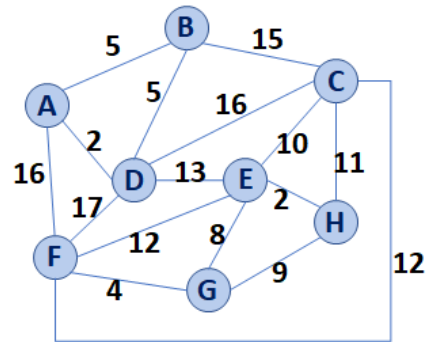
This construction will always create a new graph that is a tree (connected, acyclic graph) that spans G.

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

CS 225 – Things To Be Doing:
1. Programming Exam C ongoing **2.** MP7 is released; EC due tonight, Monday, April 23th **3.** lab_graphs available today; dues Sunday, April 22nd **4.** Daily POTDs are ongoing!

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

```
KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v : G):
    forest.makeSet(v)

  PriorityQueue Q    // min edge weight
  foreach (Edge e : G):
    Q.insert(e)

  Graph T = (V, {})

  while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) == forest.find(v):
       T.addEdge(u, v)
       forest.union( forest.find(u),
                     forest.find(v) )

  return T
```