#30: DFS and Minimum Spanning Trees (MST)

April 21, 2021 · *Brad Solomon*

Graph Traversal – BFS

Big Ideas: Utility of a BFS Traversal

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**

Modifying BFS to create DFS

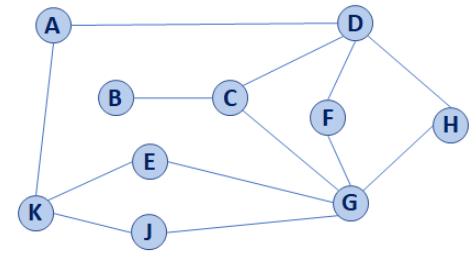
```
BFS(G):
  Input: Graph, G
  Output: A labeling of the edges on
      G as discovery and cross edges

  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)
```

DFS Graph Traversal



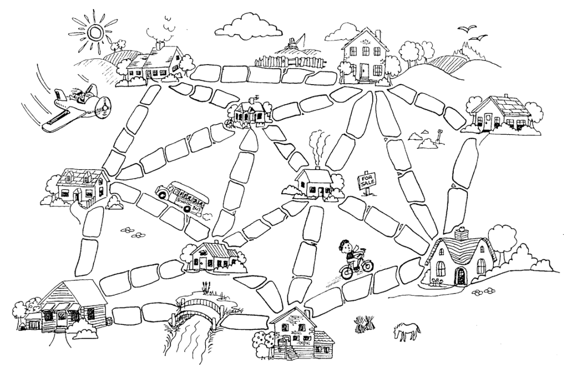
Idea: Traverse deep into the graph quickly, visiting more distant nodes before neighbors.

Two types of edges:

-

-

Minimum Spanning Tree



[“The Muddy City” by CS Unplugged](), **Creative Commons BY-NC-SA 4.0**

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 ___________ (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. There may be multiple minimum spanning trees, but they will have the same total weight.

Pseudocode for Kruskal's MST Algorithm

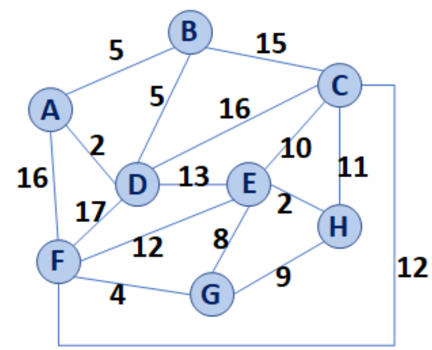
```
1   KruskalMST(G):
2     DisjointSets forest
3     foreach (Vertex v : G):
4       forest.makeSet(v)
5
6     PriorityQueue Q    // min edge weight
7     foreach (Edge e : G):
8       Q.insert(e)
9
10    Graph T = (V, {})
11
12    while |T.edges()| < n-1:
13      Vertex (u, v) = Q.removeMin()
14 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

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)

Based on our algorithm choice:

Priority Queue Implementation:	Total Running Time
Heap	
Sorted Array	

Reflections

Why would we prefer a Heap?

Why would be prefer a Sorted Array?

CS 225 – Things To Be Doing:
1. Keep working on mp_mazes! 2. Mid-Project Check-ins this week! (Keep working on project) 3. Lab_hash released today! 4. POTD Ongoing