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**

```plaintext
1 BFS(G):
2     Input: Graph, G
3     Output: A labeling of the edges on G as discovery and cross edges
4     foreach (Vertex v : G.vertices()):
5         setLabel(v, UNEXPLORED)
6     foreach (Edge e : G.edges()):
7         setLabel(e, UNEXPLORED)
8     foreach (Vertex v : G.vertices()):
9         if getLabel(v) == UNEXPLORED:
10             BFS(G, v)
11     BFS(G, v):
12         Queue q
13         setLabel(v, VISITED)
14         q.enQueue(v)
15         while !q.empty():
16             v = q.dequeue()
17             foreach (Vertex w : G.adjacent(v)):
18                 if getLabel(w) == UNEXPLORED:
19                     setLabel(v, w, DISCOVERY)
20                     setLabel(w, VISITED)
21                     q.enQueue(w)
22                 elseif getLabel(v, w) == UNEXPLORED:
23                     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**

---

**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 $G$. 

---

**“The Muddy City” by CS Unplugged,** Creative Commons BY-NC-SA 4.0
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 (e.g., 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

```plaintext
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
```

### Kruskal’s Algorithm

#### Based on our algorithm choice:

<table>
<thead>
<tr>
<th>Priority Queue Implementation</th>
<th>Total Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td></td>
</tr>
<tr>
<td>Sorted Array</td>
<td></td>
</tr>
</tbody>
</table>

### 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