November 20 – Graph Traversals and MST
G Carl Evans
Graphs

To study all of these structures:
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms
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
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(G):
    Input: Graph, G
    Output: A labeling of the edges on G as discovery and back 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:
            DFS(G, v)

DFS(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)
                DFS(G, w)
            elseif getLabel(v, w) == UNEXPLORED:
                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

```
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

<table>
<thead>
<tr>
<th>Priority Queue:</th>
<th>Heap</th>
<th>Sorted Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building</td>
<td>:7-9</td>
<td></td>
</tr>
<tr>
<td>Each removeMin</td>
<td>:13</td>
<td></td>
</tr>
</tbody>
</table>

```
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

**KruskalMST(G):**

1. DisjointSets forest
2. foreach (Vertex v : G):
   3.    forest.makeSet(v)
4. 
5. PriorityQueue Q    // min edge weight
6. foreach (Edge e : G):
   7.    Q.insert(e)
8. 
9. Graph T = (V, {})

10. while |T.edges()| < n-1:
    11.   Vertex (u, v) = Q.removeMin()
    12.   if forest.find(u) != forest.find(v):
          13.      T.addEdge(u, v)
          14.      forest.union( forest.find(u),
                          15.                  forest.find(v) )
    16. 
17. return T

<table>
<thead>
<tr>
<th>Priority Queue:</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>