CS 225
Data Structures

April 3 – Minimum Spanning Tree
G Carl Evans
Graphs

To study all of these structures:
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms
"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):
1. DisjointSets forest
2. foreach (Vertex v : G):
   3.   forest.makeSet(v)
4. PriorityQueue Q    // min edge weight
5. foreach (Edge e : G):
   6.   Q.insert(e)
7. Graph T = (V, {})
8. Graph T = (V, {})
9. while |T.edges()| < n-1:
10.   Vertex (u, v) = Q.removeMin()
11.   if forest.find(u) != forest.find(v):
12.     T.addEdge(u, v)
13.     forest.union( forest.find(u),
14.     forest.find(v) )
15.   return T
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

<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>
### Kruskal’s Algorithm

**Priority Queue:**
- Total Running Time

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td></td>
</tr>
<tr>
<td>Sorted Array</td>
<td></td>
</tr>
</tbody>
</table>

---

```plaintext
class 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
```

---

- **Priority Queue:**
  - Heap
  - Sorted Array

- **Total Running Time:**
  - Heap
  - Sorted Array
Partition Property

Consider an arbitrary partition of the vertices on $G$ into two subsets $U$ and $V$. 
Partition Property

Consider an arbitrary partition of the vertices on $G$ into two subsets $U$ and $V$.

Let $e$ be an edge of minimum weight across the partition.

Then $e$ is part of some minimum spanning tree.
Partition Property

The partition property suggests an algorithm:
Prim’s Algorithm

PrimMST(G, s):
Input: G, Graph;
        s, vertex in G, starting vertex
Output: T, a minimum spanning tree (MST) of G

foreach (Vertex v : G):
d[v] = +inf
p[v] = NULL
d[s] = 0
PriorityQueue Q   // min distance, defined by d[v]
Q.buildHeap(G.vertices())
Graph T           // "labeled set"

repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
        if cost(v, m) < d[v]:
            d[v] = cost(v, m)
p[v] = m

return T
Prim’s Algorithm

```plaintext
6 | PrimMST(G, s):
7 |   foreach (Vertex v : G):
8 |     d[v] = +inf
9 |     p[v] = NULL
10 |     d[s] = 0
11 |     PriorityQueue Q // min distance, defined by d[v]
12 |     Q.buildHeap(G.vertices())
13 |     Graph T       // "labeled set"
14 |     repeat n times:
15 |       Vertex m = Q.removeMin()
16 |         T.add(m)
17 |         foreach (Vertex v : neighbors of m not in T):
18 |           if cost(v, m) < d[v]:
19 |             d[v] = cost(v, m)
20 |             p[v] = m
```

<table>
<thead>
<tr>
<th>Adj. Matrix</th>
<th>Adj. List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td></td>
</tr>
<tr>
<td>Unsorted Array</td>
<td></td>
</tr>
</tbody>
</table>
Prim’s Algorithm

Sparse Graph:

Dense Graph:

```java
PrimMST(G, s):
    foreach (Vertex v : G):
        d[v] = +inf
        p[v] = NULL
    d[s] = 0

    PriorityQueue Q // min distance, defined by d[v]
    Q.buildHeap(G.vertices())
    Graph T // "labeled set"

    repeat n times:
        Vertex m = Q.removeMin()
        T.add(m)
        foreach (Vertex v : neighbors of m not in T):
            if cost(v, m) < d[v]:
                d[v] = cost(v, m)
                p[v] = m
```

<table>
<thead>
<tr>
<th></th>
<th>Adj. Matrix</th>
<th>Adj. List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>O(n^2 + m lg(n))</td>
<td>O(n lg(n) + m lg(n))</td>
</tr>
<tr>
<td>Unsorted Array</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
</tbody>
</table>
MST Algorithm Runtime:

• Kruskal’s Algorithm: \( O(n + m \lg(n)) \)
• Prim’s Algorithm: \( O(n \lg(n) + m \lg(n)) \)

• What must be true about the connectivity of a graph when running an MST algorithm?

• How does \( n \) and \( m \) relate?
MST Algorithm Runtime:

• Kruskal’s Algorithm:
  \( O(n + m \lg(n)) \)

• Prim’s Algorithm:
  \( O(n \lg(n) + m \lg(n)) \)