November 20 – MSTs: Kruskal + Prim’s Algorithm

G Carl Evans
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

1. KruskalMST(G):
   2. DisjointSets forest
   3. foreach (Vertex v : G):
      4.     forest.makeSet(v)
   5. PriorityQueue Q    // min edge weight
   6. foreach (Edge e : G):
      7.     Q.insert(e)
   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),
                    forest.find(v) )
      14. return T
Kruskal’s Algorithm

```
Priority Queue: | Heap | Sorted Array |
----------------|------|-------------|
Building       | :6-8 |             |
Each removeMin | :13  |             |

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

#### 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>Total Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td></td>
</tr>
<tr>
<td>Sorted Array</td>
<td></td>
</tr>
</tbody>
</table>
Kruskal’s Algorithm

Which Priority Queue Implementation is better for running Kruskal’s Algorithm?

• 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())

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

```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>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:

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(n² + m lg(n))</td>
<td>O(n lg(n) + m lg(n))</td>
</tr>
<tr>
<td>Unsorted Array</td>
<td>O(n²)</td>
<td>O(n²)</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:
  \( O(n \lg(n) + m \lg(n)) \)