March 26 – 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
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

Diagram:

- A
- B
- C
- D
- E
- F
- G
- H

Weights:
- (A, D) = 5
- (E, H) = 16
- (F, G) = 11
- (A, B) = 2
- (B, D) = 10
- (G, E) = 17
- (G, H) = 12
- (E, C) = 4
- (E, F) = 13
- (F, C) = 8
- (D, E) = 2
- (B, C) = 9
- (C, D) = 15
- (A, F) = 11
- (D, F) = 10
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)

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></td>
<td></td>
</tr>
<tr>
<td>:7-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Each removeMin</td>
<td>:13</td>
<td></td>
</tr>
</tbody>
</table>

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. 
11. while |T.edges()| < n-1:
   12. Vertex (u, v) = Q.removeMin()
   13. if forest.find(u) != forest.find(v):
       14. T.addEdge(u, v)
       15. forest.union( forest.find(u),
           16. forest.find(v) )
   17. 
18. return T
Kruskal’s Algorithm

Priority Queue: | Total Running Time
--- | ---
Heap | 
Sorted Array | 

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

1. **PrimMST(G, s):**
   2. Input: G, Graph;
   3. s, vertex in G, starting vertex
   4. Output: T, a minimum spanning tree (MST) of G
   5. 
   6. foreach (Vertex v : G):
      7. d[v] = +inf
      8. p[v] = NULL
      9. d[s] = 0
   10. PriorityQueue Q   // min distance, defined by d[v]
   11. Q.buildHeap(G.vertices())
   12. Graph T           // "labeled set"
   13. 
   14. repeat n times:
      15. Vertex m = Q.removeMin()
      16. T.add(m)
      17. 
      18. foreach (Vertex v : neighbors of m not in T):
      19. if cost(v, m) < d[v]:
          20. d[v] = cost(v, m)
          21. p[v] = m
   22. 
   23. return T
Prim’s Algorithm

```
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
15        repeat n times:
16            Vertex m = Q.removeMin()
17            T.add(m)
18            foreach (Vertex v : neighbors of m not in T):
19                if cost(v, m) < d[v]:
20                    d[v] = cost(v, m)
21                    p[v] = m
```
Prim’s Algorithm

Sparse Graph:

```plaintext
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
```

Dense Graph:

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

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

• 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:
  $O(n + m \log(n))$

- Prim’s Algorithm:
  $O(n \log(n) + m \log(n))$