March 29 – Minimum Spanning Tree (Prim)
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
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
```
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

```
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
```
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(n lg(n) + n² 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:

We know that MSTs are always run on a minimally connected graph:

\[ n - 1 \leq m \leq \frac{n(n-1)}{2} \]

\[ O(n) \leq O(m) \leq O(n^2) \]
MST Algorithm Runtime:

- Kruskal’s Algorithm: \( O(n + m \log(n)) \)
  
  **Sparse Graph:** \( m \sim n \)
  \( O(n + n \log(n)) \)
  
  **Dense Graph:** \( m \sim n^2 \)
  \( O(n + n^2 \log(n)) \)

- Prim’s Algorithm: \( O(n \log(n) + m \log(n)) \)
  
  **Sparse Graph:**
  \( O(n \log(n) + n \log(n)) \)
  \( \equiv O(n \log(n)) \)
  
  **Dense Graph:**
  \( O(n \log(n) + n^2 \log(n)) \)
  \( \equiv O(n^2 \log(n)) \)

So in both edges same
Suppose I have a new heap:

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

What’s the updated running time?

<table>
<thead>
<tr>
<th></th>
<th>Binary Heap</th>
<th>Fibonacci Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove Min</td>
<td>O(lg(n))</td>
<td>O(lg(n))</td>
</tr>
<tr>
<td>Decrease Key</td>
<td>O(lg(n))</td>
<td>O(1)*</td>
</tr>
</tbody>
</table>

PrimMST(G, s):
6   foreach (Vertex v : G):
7     d[v] = +inf
8     p[v] = NULL
9     d[s] = 0
10
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
MST Algorithm Runtimes:

- Kruskal’s Algorithm: $O(m \lg(n))$
- Prim’s Algorithm: $O(n \lg(n) + m \lg(n))$
Shortest Path