CS 225
Data Structures

April 11 – MSTs: Prim’s Algorithm
G Carl Evans
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

```c
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
12       PriorityQueue Q // min distance, defined by d[v]
13       Q.buildHeap(G.vertices())
14       Graph T        // "labeled set"
15
16       repeat n times:
17       Vertex m = Q.removeMin()
18          T.add(m)
19       foreach (Vertex v : neighbors of m not in T):
20          if cost(v, m) < d[v]:
21              d[v] = cost(v, m)
22              p[v] = m
```
PrimMST\(G, s\):

Input: \(G\), Graph;

\(s\), vertex in \(G\), starting vertex

Output: \(T\), a minimum spanning tree (MST) of \(G\)

foreach (Vertex \(v\) : \(G\)):

\(d[v] = +\infty\)

\(p[v] = \text{NULL}\)

\(d[s] = 0\)

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 \(\text{cost}(v, m) < d[v]\):

\(d[v] = \text{cost}(v, m)\)

\(p[v] = m\)

return \(T\)
Prim’s Algorithm

Sparse Graph:

Dense Graph:

```
6   PrimMST(G, s):
7     foreach (Vertex v : G):
8         d[v] = +inf
9         p[v] = NULL
10    d[s] = 0
11
12    PriorityQueue Q // min distance, defined by d[v]
13    Q.buildHeap(G.vertices())
14    Graph T       // "labeled set"
15
16    repeat n times:
17        Vertex m = Q.removeMin()
18        T.add(m)
19        foreach (Vertex v : neighbors of m not in T):
20            if cost(v, m) < d[v]:  
21                d[v] = cost(v, m)
22                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:

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

Sparse Graph:

Dense Graph:
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
```

<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>

What’s the updated running time?
MST Algorithm Runtimes:

- Kruskal’s Algorithm: \( O(m \lg(n)) \)
- Prim’s Algorithm: \( O(n \lg(n) + m \lg(n)) \)
Final Big-O MST Algorithm Runtimes:

• Kruskal’s Algorithm: $O(m \ lg(n))$

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