April 25 – Dijkstra’s Algorithm
G Carl Evans
Graphs

To study all of these structures:
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms
MST Algorithm Runtime:

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

\[ n-1 \leq m \leq n(n-1) / 2 \]

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

- Kruskal’s Algorithm:
  \[ O(n + m \ lg(n)) \]
  - Sparse Graph:
  - Dense Graph:

- Prim’s Algorithm:
  \[ O(n \ lg(n) + m \ lg(n)) \]
  - Sparse Graph:
  - Dense Graph:
Suppose I have a new heap:

```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
```

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>

<table>
<thead>
<tr>
<th>PrimMST(G, s):</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
</tbody>
</table>
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
Dijkstra’s Algorithm (SSSP)

DijkstraSSSP(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 u = Q.removeMin()
17        T.add(u)
18    foreach (Vertex v : neighbors of u not in T):
19        if _______________ < d[v]:
20            d[v] = _______________
21            p[v] = m
Dijkstra’s Algorithm (SSSP)

What about negative weight cycles?
Dijkstra’s Algorithm (SSSP)

What about negative weight edges, without negative weight cycles?
Dijkstra’s Algorithm (SSSP)

What is the running time?

```plaintext
DijkstraSSSP(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 u = Q.removeMin()
17        T.add(u)
18        foreach (Vertex v : neighbors of u not in T):
19            if ______________ < d[v]:
20                d[v] = ______________
21                p[v] = m
```