CS 225
Data Structures

December 5 – Dijkstra’s Algorithm
Wade Fagen-Ulmschneider
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):

1. foreach (Vertex v : G):
2.     d[v] = +inf
3.     p[v] = NULL
4.     d[s] = 0
5.     PriorityQueue Q // min distance, defined by d[v]
6.     Q.buildHeap(G.vertices())

7. Graph T // "labeled set"

8. repeat n times:
9.     Vertex m = Q.removeMin()
10.    T.add(m)
11.    foreach (Vertex v : neighbors of m not in T):
12.       if cost(v, m) < d[v]:
13.           d[v] = cost(v, m)
14.           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>

*Note: The running time for Decrease Key in Fibonacci Heap is O(1) because Fibonacci Heap allows efficient decrease key operations.
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: $O(n \lg(n) + m)$
MP7

Extra Credit: Due TONIGHT! (+7 points!)

Part 3: Released Tomorrow
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?

<table>
<thead>
<tr>
<th></th>
<th>DijkstraSSSP(G, s):</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>foreach (Vertex v : G):</td>
</tr>
<tr>
<td>7</td>
<td>d[v] = +inf</td>
</tr>
<tr>
<td>8</td>
<td>p[v] = NULL</td>
</tr>
<tr>
<td>9</td>
<td>d[s] = 0</td>
</tr>
<tr>
<td>10</td>
<td>PriorityQueue Q // min distance, defined by d[v]</td>
</tr>
<tr>
<td>11</td>
<td>Q.buildHeap(G.vertices())</td>
</tr>
<tr>
<td>12</td>
<td>Graph T // &quot;labeled set&quot;</td>
</tr>
<tr>
<td>13</td>
<td>repeat n times:</td>
</tr>
<tr>
<td>14</td>
<td>Vertex u = Q.removeMin()</td>
</tr>
<tr>
<td>15</td>
<td>T.add(u)</td>
</tr>
<tr>
<td>16</td>
<td>foreach (Vertex v : neighbors of u not in T):</td>
</tr>
<tr>
<td>17</td>
<td>if __________________ &lt; d[v]:</td>
</tr>
<tr>
<td>18</td>
<td>d[v] = __________________</td>
</tr>
<tr>
<td>19</td>
<td>p[v] = m</td>
</tr>
</tbody>
</table>

What is the running time?