CS 225

Data Structures

November 29 – Dijkstra’s Algorithm Analysis
G Carl Evans
Dijkstra’s Algorithm (SSSP)

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

\[
\text{DijkstraSSSP}(G, s):
\]

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

Dijkstra gives us the shortest path from our path (single source) to **every** connected vertex!

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>p:</td>
<td>--</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>F</td>
<td>A</td>
<td>F</td>
<td>C</td>
</tr>
<tr>
<td>d:</td>
<td>0</td>
<td>10</td>
<td>17</td>
<td>15</td>
<td>12</td>
<td>7</td>
<td>11</td>
<td>21</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm (SSSP)

Q: How does Dijkstra handle a single heavy-weight path vs. many light-weight paths?
Dijkstra’s Algorithm (SSSP)

Q: How does Dijkstra handle a single heavy-weight path vs. many light-weight paths?
Dijkstra’s Algorithm (SSSP)

Q: How does Dijkstra handle a single heavy-weight path vs. many light-weight paths?
Dijkstra’s Algorithm (SSSP)

**Q:** How does Dijkstra handle undirected graphs?
Q: How does Dijkstra handle negative weight cycles?
Q: How does Dijkstra handle negative weight edges, without a negative weight cycle?
Dijkstra’s Algorithm (SSSP)

Q: How does Dijkstra handle negative weight cycles?
Dijkstra’s Algorithm (SSSP)

Q: How does Dijkstra handle negative weight cycles?
Dijkstra’s Algorithm (SSSP)

Q: How does Dijkstra handle negative weight cycles?

Shortest Path (A \(\rightarrow\) E):

- Path: A \(\rightarrow\) F \(\rightarrow\) E \(\rightarrow\) (C \(\rightarrow\) H \(\rightarrow\) G \(\rightarrow\) E)*
- Length: 12
- Length: -5 (repeatable)
Q: How does Dijkstra handle negative weight edges, without a negative weight cycle?
Dijkstra’s Algorithm (SSSP)

What is Dijkstra’s running time?

DijkstraSSSP(G, s):
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      repeat n times:
14         Vertex u = Q.removeMin()
15         T.add(u)
16         foreach (Vertex v : neighbors of u not in T):
17             if cost(u, v) + d[u] < d[v]:
18                 d[v] = cost(u, v) + d[u]
19                 p[v] = m
20         return T
Suppose I have a new heap:

<table>
<thead>
<tr>
<th>DijkstraSSSP(G, s):</th>
<th>Binary Heap</th>
<th>Fibonacci Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>foreach (Vertex v : G):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d[v] = +inf</td>
<td>O(\lg(n))</td>
<td>O(\lg(n))</td>
</tr>
<tr>
<td>p[v] = NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d[s] = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PriorityQueue Q // min distance, defined by d[v]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q.buildHeap(G.vertices())</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph T // &quot;labeled set&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>repeat n times:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertex u = Q.removeMin()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T.add(u)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>foreach (Vertex v : neighbors of u not in T):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if cost(u, v) + d[u] &lt; d[v]:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d[v] = cost(u, v) + d[u]</td>
<td>O(\lg(n))</td>
<td>O(1)*</td>
</tr>
<tr>
<td>p[v] = m</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What’s the updated running time?
Landmark Path Problem

Suppose you want to travel from A to G.

Q1: What is the shortest path from A to G?
Landmark Path Problem

Suppose you want to travel from A to G.

Q2: What is the fastest algorithm to use to find the shortest path?
Landmark Path Problem

In your journey between A and G, you also want to visit the landmark L.

Q3: What is the shortest path from A to G that visits L?
Landmark Path Problem

In your journey between A and G, you also want to visit the landmark L.

**Q4:** What is the fastest algorithm to find this path?

**Q5:** What are the specific call(s) to this algorithm?