April 24 – Dijkstra’s Algorithm
Wade Fagen-Ulmschneider, Craig Zilles
Prim's Algorithm

```
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 + n \lg(n) + n^2 + m \lg(n))</td>
<td>O(n + n \lg(n) + m \lg(n) + m)</td>
</tr>
<tr>
<td>Unsorted Array</td>
<td>O(n + n^2 + m)</td>
<td>O(n + n^2 + m)</td>
</tr>
</tbody>
</table>
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>Adj. Matrix</th>
<th>Adj. List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>(O(n^2 + m \lg(n)))</td>
</tr>
<tr>
<td>Unsorted Array</td>
<td>(O(n^2 + m))</td>
</tr>
</tbody>
</table>
MST Algorithm Runtime:

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

- What must be true about the connectivity of a graph when running an MST algorithm?

- How does \( n \) and \( m \) relate?
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
```

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>
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: $O(n \lg(n) + m)$
End of Semester Logistics

**Lab:** Your final CS 225 lab is this week.

**Final Exam:** Final exams start on Reading Day (May. 2)
• Final is [One Theory Exam] + [One Programming Exam] together in a single exam.
• Time: 3 hours

**Grades:** There will be an April grade update posted this week with all grades up until now.
"HEY, COME JOIN US"

https://www.youtube.com/watch?v=7Ug1fr_ID_s
CAs

Instructors

Wade Fagen-Ulmschneider
waf

Craig Zilles
zilles

Thierry Ramais
ramais
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  PriorityQueue Q // min distance, defined by d[v]
11  Q.buildHeap(G.vertices())
12  Graph T // "labeled set"
13
14  repeat n times:
15     Vertex u = Q.removeMin()
16     T.add(u)
17     foreach (Vertex v : neighbors of u not in T):
18         if _______________ < d[v]:
19             d[v] = _______________
20             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
```