CS 225
Data Structures

April 23 – Dijkstra’s Algorithm
Wade Fagen-Ulmschneider
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

```
PrimsMST(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

```java
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></td>
</tr>
<tr>
<td>Unsorted Array</td>
<td></td>
</tr>
</tbody>
</table>
Prim’s Algorithm

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
Prim’s Algorithm

Sparse Graph:

Dense Graph:

```
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>Adj. Matrix</th>
<th>Adj. List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>O(n² + m lg(n))</td>
<td>O(n lg(n) + m lg(n))</td>
</tr>
<tr>
<td>Unssorted Array</td>
<td>O(n²)</td>
<td>O(n²)</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:

• Kruskal’s Algorithm:
  \[ O(n + m \log(n)) \]

• Prim’s Algorithm:
  \[ O(n \log(n) + m \log(n)) \]
MST Algorithm Runtime:

• Upper bound on MST Algorithm Runtime: $O(m \lg(n))$
Suppose I have a new heap:

```plaintext
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>Operation</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>
End of Semester Logistics

Lab: Your final CS 225 lab is this week.
  • No lab sections next week (partial week).

Final Exam: Final exams start on Reading Day (May 3)
  • Last day of office hours is Wednesday, May 2.
  • No office/lab hours once the first final exam is given.

Grades: There will be a “Pre-Final” grade update posted next week with all grades except your final.
  • MP7’s grace period extends until Tuesday, May 1
  • Goal: Have “Pre-Final” grade on Wednesday/Thursday
Shortest Path
Dijkstra’s Algorithm (SSSP)

DijkstraSSSP(G, s):
6 \hspace{1em} foreach (Vertex v : G):
7 \hspace{1em} d[v] = +inf
8 \hspace{1em} p[v] = NULL
9 \hspace{1em} d[s] = 0
10
11 \hspace{1em} PriorityQueue Q // min distance, defined by d[v]
12 \hspace{1em} Q.buildHeap(G.vertices())
13 \hspace{1em} Graph T // "labeled set"
14
15 \hspace{1em} repeat n times:
16 \hspace{2em} Vertex u = Q.removeMin()
17 \hspace{2em} T.add(u)
18 \hspace{2em} foreach (Vertex v : neighbors of u not in T):
19 \hspace{3em} if \hspace{0.5em} ________________ < d[v]:
20 \hspace{4em} d[v] = ________________
21 \hspace{4em} 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?

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