CS 225

Data Structures

November 17 – MSTs: Prim’s Algorithm

G Carl Evans
Partition Property

Consider an arbitrary partition of the vertices on $G$ into two subsets $U$ and $V$. 
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

```
PrimMST(G, s):
1   Input: G, Graph;
2       s, vertex in G, starting vertex
3   Output: T, a minimum spanning tree (MST) of G
4
5   foreach (Vertex v : G):
6       d[v] = +inf
7       p[v] = NULL
8       d[s] = 0
9
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 m = Q.removeMin()
16      T.add(m)
17      foreach (Vertex v : neighbors of m not in T):
18          if cost(v, m) < d[v]:
19              d[v] = cost(v, m)
20              p[v] = m
21
22  return T
```
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>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

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></th>
<th>Adj. Matrix</th>
<th>Adj. List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>$O(n^2 + m \log(n))$</td>
<td>$O(n \log(n) + m \log(n))$</td>
</tr>
<tr>
<td>Unsorted Array</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</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:
  \( O(n + m \lg(n)) \)

• Prim’s Algorithm:
  \( O(n \lg(n) + m \lg(n)) \)
Shortest Path