CS 225
Data Structures

December 3 – Prim’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:
Mattox Monday
End of Semester Logistics

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

**Final Exam:** Final exams start on Reading Day (Dec. 13)
- Final is [One Theory Exam] + [One Programming Exam] together in a single exam.
- Time: 3 hours

**Grades:** There will be a “December” grade update posted today with all grades up until now.
"HEY, COME JOIN US"

Love Story - CS 225
10 views

Ritika Adhikari
Published on Dec 3, 2018

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

Instructors

Wade Fagen-Ulmschneider

Office
2215 Siebel (2nd floor, south-west corner of Siebel)

Debug Your Brain
Every Wednesday, 4pm-5:30pm, 2036 ECEB

Mattox Beckman

Office
2212 Siebel (2nd floor, south-west corner of Siebel)

Debug Your Brain
Every Wednesday, 12:15pm - 1:45pm, 3036 ECEB

Thierry Ramais
Head of Course Logistics
Prim’s Algorithm

```
PrimMST(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

```
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:

```
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
```

Dense Graph:

```
Adj. Matrix

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

• Kruskal’s Algorithm: $O(m \ lg(n))$
• Prim’s Algorithm: $O(n \ lg(n) + m)$
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?

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