Kruskal’s Algorithm

Pseudocode for Kruskal’s MST Algorithm

```
KruskalMST(G):
    DisjointSets forest
    foreach (Vertex v : G):
        forest.makeSet(v)
    PriorityQueue Q    // min edge weight
    foreach (Edge e : G):
        Q.insert(e)
    Graph T = (V, {})
    while |T.edges()| < n-1:
        Vertex (u, v) = Q.removeMin()
        if forest.find(u) != forest.find(v):
            T.addEdge(u, v)
            forest.union( forest.find(u),
                          forest.find(v) )
    return T
```

Kruskal’s Running Time Analysis

We have multiple choices on which underlying data structure to use to build the Priority Queue used in Kruskal’s Algorithm:

<table>
<thead>
<tr>
<th>Priority Queue Implementations</th>
<th>Heap</th>
<th>Sorted Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building</td>
<td>6-8</td>
<td></td>
</tr>
<tr>
<td>Each removeMin</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Based on our algorithm choice:

<table>
<thead>
<tr>
<th>Priority Queue Implementation</th>
<th>Total Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td></td>
</tr>
<tr>
<td>Sorted Array</td>
<td></td>
</tr>
</tbody>
</table>

Reflections

Why would we prefer a Heap?

Why would be prefer a Sorted Array?

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.

Proof in CS 374!
Prim’s Minimum Spanning Tree Algorithm

Pseudocode for Prim’s MST Algorithm

```
PrimMST(G, s):
    Input: G, Graph; s, vertex in G, starting vertex of algorithm
    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
```

Running Time of MST Algorithms

Kruskal’s Algorithm:

Prim’s Algorithm:

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

...what does this imply about the relationship between \( n \) and \( m \)?

<table>
<thead>
<tr>
<th>Kruskal’s MST</th>
<th>Prim’s MST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Q:** Suppose we built a new heap that optimized the decrease-key operation, where decreasing the value of a key in a heap updates the heap in amortized constant time, or \( O(1) \)*. How does that change Prim’s Algorithm runtime?

Final big-\( O \) Running Times of classical MST algorithms:

<table>
<thead>
<tr>
<th>Kruskal’s MST</th>
<th>Prim’s MST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CS 225 – Things To Be Doing:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. mp_mazes due today!</td>
</tr>
<tr>
<td>2. If your final project has not been approved get it revised.</td>
</tr>
<tr>
<td>3. Daily POTDs are ongoing for +1 point /problem but pausing over break</td>
</tr>
</tbody>
</table>