Prim’s Algorithm

- Find an MST on edge-weighted, connected, *undirected* graphs
- Greedily select edges one by one and add to a growing sub-graph
- Grows a tree from a single vertex

Prim’s Algorithm

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and add that vertex to the tree
3. Repeat step 2 (until all vertices are in the tree)

Prim's Algorithm

- Given graph $G = (V, E)$
- Start with 2 sets of vertices: 'innies' & 'outies'
  - 'Innies' are visited nodes (initially empty)
  - 'Outies' are not yet visited (initially V)
- Select first innie arbitrarily (root of MST)
- Repeat until no more outies
  - Choose outie (v') with smallest distance from *any* innie
  - Move v' from outies to innies
- Implementation issue: use linear search or PQ*?*

Prim: Data structures

- A vector of classes or structures
- For each vertex v, record:
  - k_v: Has v been visited?
    (initially **false** for all $v \in V$)
  - d_v: What is the minimal edge weight to v?
    (initially ∞ for all $v \in V$, except $v_r = 0$)
  - p_v: What vertex precedes (is parent of) v?
    (initially **unknown** for all $v \in V$)

Prim's Algorithm

Set starting point d_v to 0.

Loop v times (until every k_v is true):

1. **From the set of vertices for which k_v is false, select the vertex v having the smallest tentative distance d_v.**
2. **Set k_v to true.**
3. **For each vertex w adjacent to v for which k_w is false, test whether d_w is greater than distance(v,w). If it is, set d_w to distance(v,w) and set p_w to v.**

Implementing Prim's

- Implement in the order listed:
  - 1: Loop over all vertices: find smallest false k_v
  - 2: Mark k_v as true
  - 3: Loop over all vertices: update false neighbors of k_v
- Common Mistake: Set the first vertex to true outside the loop
- Reordering this can result in a simple algorithm that simply doesn't work

Complexity – Linear Search

Loop v times:

$|V|$ times

1. **From the set of vertices for which k_v is false, select the vertex v having the smallest tentative distance d_v.**

$O(|V|)$

2. **Set k_v to true.**

$O(1)$

3. **For each vertex w adjacent to v for which k_w is false, test whether d_w is greater than distance(v,w). If it is, set d_w to distance(v,w) and set p_w to v.**

Most at this vertex: $O(|V|)$. Cost of each: $O(1)$.

Prim's (Heap) Algorithm

Algorithm Prims_Heaps(G, s_0)

//Initialize	
n = |V|	***O*(1)**
create_table(n) //stores k,d,p	***O*(*V*)**
create_pq() //empty heap	***O*(1)**
table[s_0].d = 0	***O*(1)**
insert_pq(0, s_0)	***O*(1)**

Prim's (Heap) Algorithm

while (!pq.isempty)	$O(E)$
v_0 = getMin() //heap top() & pop()	$O(\log E)$
if (!table[v_0].k) //not known	$O(1)$
table[v_0].k = true	$O(1)$
for each $v_i \in$ Adj[v_0]	$O(1 + E/V)$
if (!table[v_i].k)	$O(1)$
distance = weight(v_0, v_i)	$O(1)$
if (distance < table[v_i].d)	$O(1)$
table[v_i].d = distance	$O(1)$
table[v_i].p = v_0	$O(1)$
insert_pq(distance, v_i)	$O(\log E)$

Complexity – Heaps

Repeat until the PQ is empty:

$|E|$ times

1. **From the set of vertices for which k_v is false, select the vertex v having the smallest tentative distance d_v.**

$O(\log |E|)$

2. **Set k_v to true.**

$O(1)$

3. **For each vertex w adjacent to v for which k_w is false, test whether d_w is greater than distance(v,w). If it is, set d_w to distance(v,w) and set p_w to v.**

Most at this vertex: $O(|V|)$. Cost of each: $O(\log |E|)$.
Note: Visits every edge once (over all iterations) = $O(|E|)$.

Prim's: Complexity Summary

- ***O(V²)*** for the simplest nested-loop implementation

- ***O(E* log *E)*** with heaps
  - Is this always faster?
  - Think about the complexity of the PQ version for dense versus sparse graphs