Dijkstra's Shortest Path and Prim's Minimum Spanning Tree

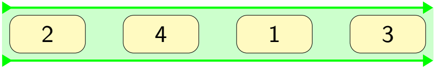
Author

University of Illinois Urbana-Champaign

Date

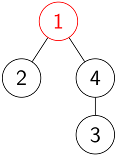
Queues vs Priority Queues

Standard Queue: Stores things in the offer they were enqueued.



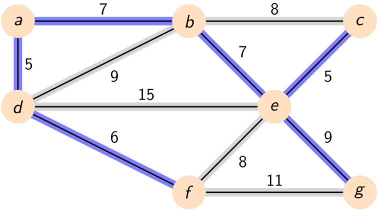

```
Queue<Integer> q = new
    ArrayDequeue<>();
q.offer(3);
q.offer(1);
q.offer(4);
q.offer(2);
```

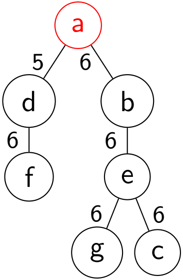
Priority Queue: Reorders on insertion so the min or max value is always on top.




```
Queue<Integer> pq = new
    PriorityQueue<>();
pq.offer(3);
pq.offer(1);
pq.offer(4);
pq.offer(2);
```

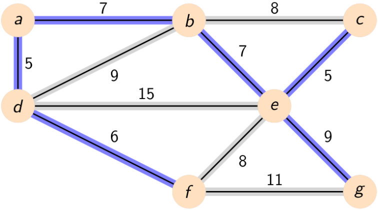
Prim's Key Points

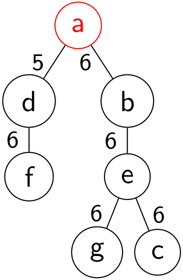




- **Purpose:** For an weighted undirected graph, Prim's MST finds a subgraph (tree) that meets the following condition:
  1. All vertices are reachable.
  2. The sum of the edges in the graph are minimized.

Prim's Key Points





- **Greedy Algorithm:** A algorithm that uses of making the optimal decision at each individual step such that the sum of these locale ally optimal steps leads to a globally optimal solution.
- **Prim's Greedy Heuristic:** Visit a reachable vertex with the smallest distance between it and it's parent at each step.

The Algorithm - Setup (Steps 1 & 2)

Algorithm 1: Prim's Minimum Spanning Tree

```
Function Prim(G, Source)
    for u ∈ G.Vertex do                                   (1)
        u.dist ← ∞
        u.parent ← null
    end
    S.dist ← 0                                            (2)
    PQ ← ∅
    PQ.Enqueue(Source)
    while PQ ≠ ∅ do
        u ← PQ.RemoveMin()
        for e ∈ G.Adj[u] do
            if e.dest ∈ PQ AND e.weight < v.dist
             then
                v.parent ← u
                v.dist ← e.weight
                PQ.Reprioritize()
            else
        end
    end
return
```

- **Step 1:** For each vertex, initialize the parents and the distance to that parent to be null and infinity (`Integer.MAX_VALUE`), respectively .
- **Step 2:** The source distance to parent to be 0, make a priority queue, and place that vertex in it.
    - After doing this, what will be the first entry in the PQ? Why?

The Algorithm - Finding the MST (Steps 3-5)

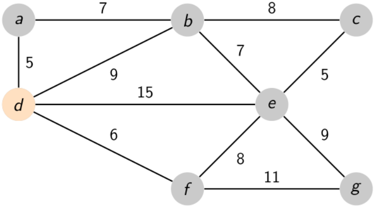
Algorithm 1: Prim's Minimum Spanning Tree

Function Prim(G, $Source$)
  for $u \in G.Vertex$ **do**
    u.dist $\leftarrow \infty$
    u.parent $\leftarrow$ null
  end
  S.dist $\leftarrow 0$
  PQ $\leftarrow \emptyset$
  PQ.Enqueue(Source)
  while $PQ \neq \emptyset$ **do**
    u $\leftarrow$ PQ.RemoveMin()
    for $e \in G.Adj[u]$ **do**
      if $e.dest \in PQ$ AND $e.weight < v.dist$ **then**
        v.parent $\leftarrow$ u
        v.dist $\leftarrow$ e.weight
        PQ.Reprioritize()
      else
    end
  end
return

- **Step 3:** While there are still nodes unvisited, get the node with the minimum distance to it's parent (i.e., a node that's already been visited)
- **Step 4:** For each vertex adjacent to U, check if hasn't been visited (still in PQ) and if the weight from it to the vertex we're visiting is less than it's previous weight.
- **Step 5:** If it is, update the parent and the weight to that parent.

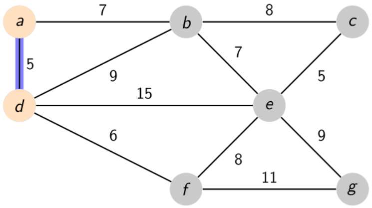
Prim's - Iteration 1

Vertex	D	A	B	C	E	F	G
Parent	-	-	-	-	-	-	-
Distance	0	∞	∞	∞	∞	∞	∞



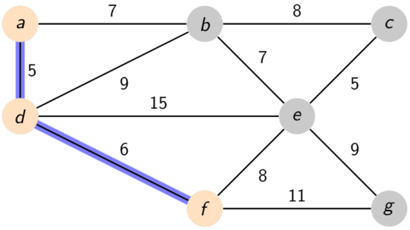
Vertex	A	B	C	E	F	G
Parent						
Distance						

Vertex	A	F	B	E	C	G
Parent	D	D	D	D	-	-
Distance	5	6	9	15	∞	∞



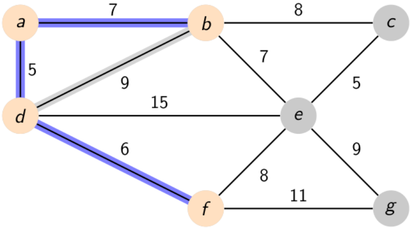
Vertex	F	B	E	C	G
Parent					
Distance					

Vertex	F	B	E	C	G
Parent	D	A	D	-	-
Distance	6	7	15	∞	∞



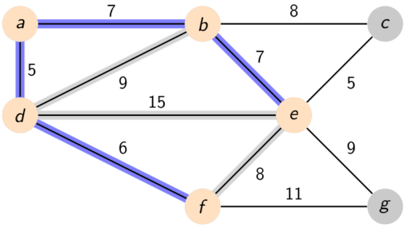
Vertex	B	E	G	C
Parent				
Distance				

Vertex	B	E	G	C
Parent	A	F	F	-
Distance	7	8	11	∞



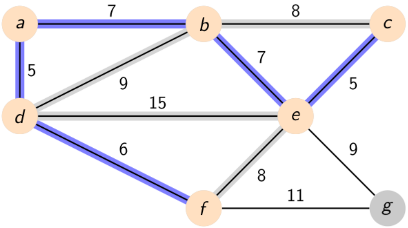
Vertex	E	C	G
Parent			
Distance			

Vertex	E	C	G
Parent	B	B	F
Distance	7	8	11



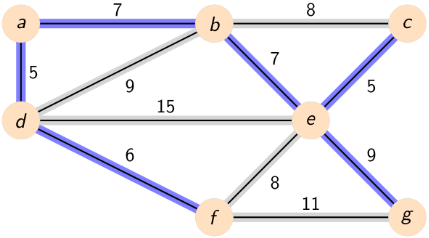
Vertex	C	G
Parent		
Distance		

Vertex	C	G
Parent	E	E
Distance	5	9



Vertex	G
Parent	
Distance	

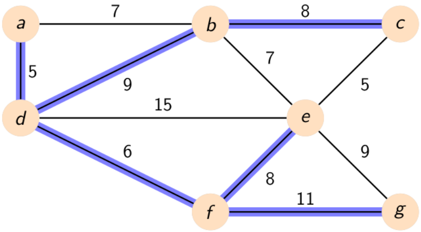
Vertex	G
Parent	E
Distance	9



Outputting the MST

- After the algorithm has run, the edges that make up the MST will be stored in the vertex-parent pairs. To output the MST, iterate over the vertices and print out those pairs.
- We could also keep track as we traverse (like BST).

Dijkstra's Key Points



- **Goal:** Find the shortest path from a source vertex to every vertex in the graph.
- **Dijkstra's Greedy Heuristic:** At each step, visit the vertex that has the minimum distance *from our source*.

Prim's vs Dijkstra's

Prim's MST:

```
while PQ ≠ ∅ do
    u ← PQ.getMin()
    for e ∈ G.Adj[u] do
        if e.dest ∈ PQ AND e.weight < v.dist
          then
            v.parent ← u
            v.dist ← e.weight
            PQ.reprioritize(v);
        end
    end
end
```

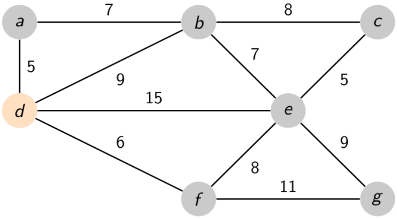
- Selecting the vertex that has and edge accessing it which has the minimum distance overall.
- Updating the distance associated with that vertex if we find another edge that accesses it that has a lower weight.

Dijkstra's SP:

```
while PQ ≠ ∅ do
    u ← PQ.getMin()
    for v ∈ G.Adj[u] do
        PathWeight ← u.Distance + Weight(u, v)
        if v ∈ PQ AND pathWeight < v.Distance
          then
            v.parent ← u
            v.distance ← PathWeight
            PQ.reprioritize(v);
        end
    end
end
```

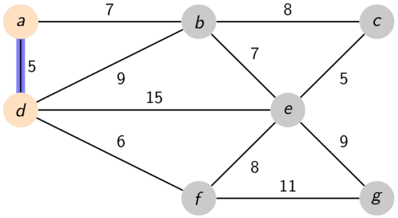
- Selecting the vertex that has the minimum distance *from our source*.
- We are checking to see if this vertex provides a shorter path to any of it's adjacent notes and updating their parent and distance if so.

Vertex	D	A	B	C	E	F	G
Parent	-	-	-	-	-	-	-
Distance	0	∞	∞	∞	∞	∞	∞



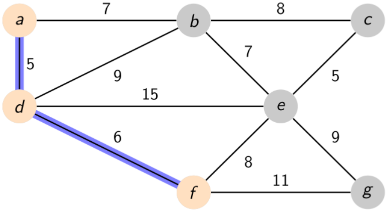
Vertex	A	F	B	E	C	G
Parent						
Distance						

Vertex	A	F	B	E	C	G
Parent	D	D	D	D	-	-
Distance	5	6	9	15	∞	∞



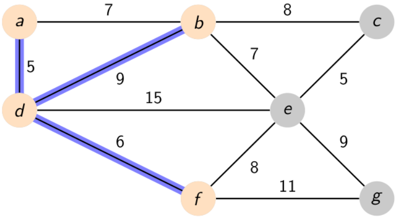
Vertex	F	B	E	C	G
Parent					
Distance					

Vertex	F	B	E	C	G
Parent	D	D	D	-	-
Distance	6	9	15	∞	∞



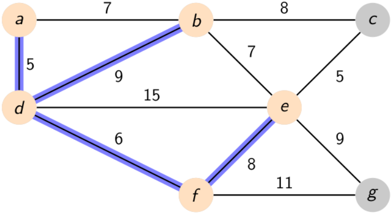
Vertex	B	E	C	G
Parent				
Distance				

Vertex	B	E	G	C
Parent	D	F	F	-
Distance	9	14	17	∞



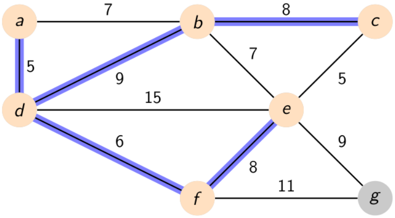
Vertex	E	G	C
Parent			
Distance			

Vertex	E	G	C
Parent	F	F	B
Distance	14	17	17



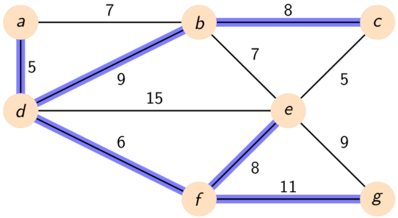
Vertex	C	G
Parent		
Distance		

Vertex	C	G
Parent	B	F
Distance	17	17

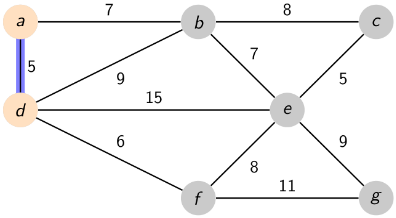


Vertex	G
Parent	
Distance	

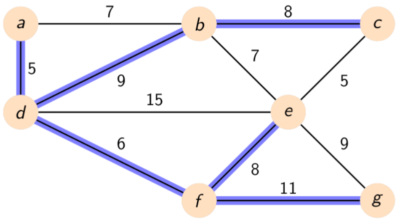
Vertex	G
Parent	F
Distance	17



Done!


a
b
c
d
e
f
g
7
8
5
9
7
5
15
6
8
9
11

Outputting the Shortest path from Source to Dest



- Start at the destination you're wanting to visit.
- Work your way backwards keeping track of each step.
- Stop once you've found the source.

How do we do this in Java? Priority Queues + HashMap

```
Map<E, E> parents = new HashMap<>();
Map<E, Integer> dists = new HashMap<>();

// We create a priority queue that orders things
// based on the value they're associated with in
// the hashmap.
Queue<E> pq = new PriorityQueue<E>(
    //tells the PQ to dists.get(vertex) to compare each edge
    Comparator.comparing(dists)
);
```

- Like BFS and DFS we will use HashMaps to keep track of values associated with vertices:
  - `parents` keeps track of each vertex's parent.
  - `dists` keeps track of the distance to that vertex.
  - `pq` keeps track of: 1) what vertexes are left and 2) orders them based on their distance in `dists`.
- To "reprioritize" the priority queue, we need to remove and readd a