Shortest Path

Dijkstras:

Implementation with PQ

- O(|E| + |V| log |V|)
- Make source current node with its distance 0
- Repeat until no elements in PQ
  - If current node is not in distance map or has a greater value than computed value place the current node in the distance map (mapped to distance)
  - Place its neighbors in the PQ with their distances to current node + current node distance
  - Dequeue min element from PQ and make current node

Bellman Ford:

psuedo java code

```
for i=1 to size(vertices)-1
   for each edge (u,v)
      if distance[u]+w < distance[v]
         distance[v]=distance[u]+w
         predecessor[v]=u
```

c++ code

```
function BellmanFord(list vertices, list edges, vertex source)
   ::distance[],predecessor[]

   // This implementation takes in a graph, represented as
   // lists of vertices and edges, and fills two arrays
   // (distance and predecessor) with shortest-path
   // (less cost/distance/metric) information

   // Step 1: initialize graph
   for each vertex v in vertices:
       if v is source then distance[v] := 0
       else distance[v] := inf
       predecessor[v] := null

   // Step 2: relax edges repeatedly
   for i from 1 to size(vertices)-1:
       for each edge (u, v) in Graph with weight w in edges:
           if distance[u] + w < distance[v]:
               distance[v] := distance[u] + w
               predecessor[v] := u
```

```
// Step 3: check for negative-weight cycles
for each edge (u, v) in Graph with weight w in edges:
    if distance[u] + w < distance[v]:
        error "Graph contains a negative-weight cycle"
return distance[], predecessor[]
```

Notes

- O(|V| |E|)
- Allows negative weights