Shortest Path Algorithm

- Find shortest path between two nodes
  - “distance” = “cost”
- Typical variant
  - Find shortest path between one node and all other nodes
  - Shortest path tree from staring node

Shortest Path Example

- Example: least-cost path from BDL to any other airport

<table>
<thead>
<tr>
<th></th>
<th>ALB</th>
<th>BDL</th>
<th>LGA</th>
<th>JFK</th>
<th>EWR</th>
<th>HPN</th>
<th>PVD</th>
<th>BOS</th>
<th>MHT</th>
<th>BTV</th>
<th>SWF</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALB</td>
<td>75</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>257</td>
</tr>
<tr>
<td>BDL</td>
<td>167</td>
<td>82</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>147</td>
</tr>
<tr>
<td>LGA</td>
<td>75</td>
<td>63</td>
<td></td>
<td></td>
<td>180</td>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>180</td>
</tr>
<tr>
<td>JFK</td>
<td>123</td>
<td>55</td>
<td>95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EWR</td>
<td>167</td>
<td>63</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>288</td>
</tr>
<tr>
<td>HPN</td>
<td>55</td>
<td>225</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PVD</td>
<td>95</td>
<td></td>
<td>225</td>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOS</td>
<td>82</td>
<td>225</td>
<td>99</td>
<td>110</td>
<td>202</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MHT</td>
<td>288</td>
<td>110</td>
<td>99</td>
<td>99</td>
<td>139</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTV</td>
<td>257</td>
<td>147</td>
<td>180</td>
<td>202</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWF</td>
<td>59</td>
<td>49</td>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ideas

- Problem and methodology
  - Find the shortest path from a starting vertex to all other vertices.
  - Add an edge one by one: find the path to each vertex one by one
- Easier to solve for vertices close to starting vertex
  - Neighbors are easy to determine.
- Avoid adding edges where “future path” may be better
  - Cost is positive and additive.
  - Path cost can not go down later.
- Iteratively expand the set of nodes where the shortest path is known

Dijkstra’s Algorithm

- Assign starting vertex to set and set distance to zero
- Repeat until all nodes are in set
  - For all nodes \( j \) not in set, calculate lowest distance to start vertex
    - \( \text{(Distance of node } i \text{ in set)} + \text{(cost of edge from } i \text{ to } j) \)
    - Remember predecessor node for \( j \) (i.e., \( i \))
  - Pick node \( j \) not in set with lowest total distance
    - Add edge from predecessor \( j \) to tree
    - Add \( j \) to set
- Variables needed to maintain information
  - Set of nodes for which shortest path is known
  - Distance to start node (as far as known)
  - Predecessor node for current distance
  - Needed to reconstruct shortest path tree

Example

- Initial setup: start node in set, distance 0

Example

- Update distance to neighboring non-set nodes
Example

- Pick non-set node with min distance

Example

- Update distance to neighboring non-set nodes
Example

- Pick non-set node with min distance

Example

- Update distance to neighboring non-set nodes
Example

• Pick non-set node with min distance

• Update distance to neighboring non-set nodes
Example

- Pick non-set node with min distance

Example

- Update distance to neighboring non-set nodes

Example

- Pick non-set node with min distance

Example

- Update distance to neighboring non-set nodes
Example

• Pick non-set node with min distance

Example

• Update distance to neighboring non-set nodes

Example

• Pick non-set node with min distance

Example

• Final result
Implementation of Dijkstra’s Algorithm

- Data structure setup
  ```java
  public Graph shortestPath(Vertex start)
  {
      Graph sp = new Graph(maxVertices);
      int[] dist = new int[activeVertices];
      final int infinity = Integer.MAX_VALUE; // define "infinity"
      boolean[] inSet = new boolean[activeVertices];
      Vertex[] predecessor = new Vertex[activeVertices];

      for (int i=0; i<activeVertices; i++)
      { // copy vertices to graph
        sp.addVertex(vertices[i]);
        dist[i] = infinity;
        inSet[i] = false;
        predecessor[i] = null;
      }
      ...  
  }
  ```

- Finding of min and adding of edge
  ```java
  min=null;
  
  for (int j=0; j<activeVertices; j++)
  { // find non-set vertex with least distance
    if (!inSet[j] && (min==null ||
        (min!=null && dist[j]<dist[min.graphIndex]))))
      min=vertices[j];
  }
  // add edge with same weight as in original graph
  sp.addEdge(min, predecessor[min.graphIndex], edges[predecessor[min.graphIndex].graphIndex][min.graphIndex]);
  inSet[min.graphIndex]=true; // mark vertex as part of set

  return sp;
  ```

- Adding of start node and updating of neighbor costs
  ```java
  dist[start.graphIndex]=0; // initialize starting vertex
  inSet[start.graphIndex]=true; // make part of set
  Vertex min=start; // min tracks last added node

  for (int i=1; i<activeVertices; i++)
  { // run once for every non-start vertex
    for (int j=0; j<activeVertices; j++)
    { // update costs of min's neighbors
      if (!inSet[j] && edges[min.graphIndex][j]!=0)
      { // if edge to non-set vertex exists
        if (dist[min.graphIndex]+edges[min.graphIndex][j]<dist[j])
        { // check if path via min is shorter
          dist[j]=dist[min.graphIndex]+edges[min.graphIndex][j];
          predecessor[j]=min; // update distance and predecessor
        }
      }
    }
  }
  ```