Dijkstra’s Algorithm Continued

E.W. Dijkstra (1930-2002)

Dijkstra’s Algorithm: Pseudocode

Initialize the cost of each node to $\infty$
Initialize the cost of the source to 0
While there are unknown nodes left in the graph
   Select an unknown node $b$ with the lowest cost
   Mark $b$ as known
   For each node $a$ adjacent to $b$
      $a$’s cost = min($a$’s old cost, $b$’s cost + cost of $(b, a)$)

Dijkstra’s Algorithm: Implementation

void Graph::dijkstra(Vertex s)
{
    Vertex v,w;
    Initialize s.dist = 0 and set dist of all other vertices to infinity
    while (there exist unknown vertices, find the one $b$ with the smallest distance)
    
    $b$.known = true;

    for each a adjacent to b
    
    if (!a.known)
    
    if (b.dist + Cost_ba < a.dist){
    
    decrease(a.dist to= b.dist + Cost_ba);
    
    a.path = b;
    
    }
    
    }

Dijkstra’s vs BFS

At each step:
1) Pick closest unknown vertex
2) Add it to finished vertices
3) Update distances

Dijkstra’s Algorithm
$O(|V|^2 + |E|)$ directly

Breadth-first Search
$O(|V| + |E|)$

Operations to be performed:

deleteMin()

decreaseKey()
Single-Source Shortest Path

• Given a graph $G = (V, E)$ and a single distinguished vertex $s$, find the shortest weighted path from $s$ to every other vertex in $G$.

All-Pairs Shortest Path:

• Find the shortest paths between all pairs of vertices in the graph.
• How?

Analysis

• Total running time for Dijkstra’s:
  $O(|V|^2 + |E|)$ (linear scan)
  $O(|V| \log |V| + |E| \log |V|)$ (heaps)

What if we want to find the shortest path from each point to ALL other points?

Dynamic Programming

Algorithmic technique that systematically records the answers to sub-problems in a table and re-uses those recorded results (rather than re-computing them).

Simple Example: Calculating the Nth Fibonacci number.

$Fib(N) = Fib(N-1) + Fib(N-2)$

Floyd-Warshall

```
for (int k = 1; k <= V; k++)
    for (int i = 1; i <= V; i++)
        for (int j = 1; j <= V; j++)
            if ( (M[i][k] + M[k][j] ) < M[i][j] )
                M[i][j] = M[i][k] + M[k][j]
```

Invariant: After the $k$th iteration, the matrix includes the shortest paths for all pairs of vertices (i,j) containing only vertices 1..$k$ as intermediate vertices.

Initial state of the matrix:

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>2</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
<td>0</td>
<td>-2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>d</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
```

$M[i][j] = \min(M[i][j], M[i][k]+M[k][j])$

Floyd-Warshall - for All-pairs shortest path:

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>2</td>
<td>-4</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
<td>0</td>
<td>-4</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>d</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>e</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>
```

Final Matrix Contents
Floyd-Warshall

Performance
• Time = $O(|V|^3)$
• Space = $O(|V|^2)$