A Slight Searching Wrinkle: Weighted Graphs

Each edge has an associated weight or cost.

![Diagram of a network with cities and distances]

Diffentiating Between Path Length and Path Cost

Path length: the number of edges in the path
Path cost: the sum of the costs of each edge

![Diagram of a network with cities and distances]

Formally speaking …

Given a graph \( G = (V, E) \) and a vertex \( s \in V \), find the shortest path from \( s \) to every vertex in \( V \)

Many variations:
- Weighted vs. unweighted
- Cyclic vs. acyclic
- Positive weights only vs. negative weights allowed
- Multiple weight types to optimize
- Directed vs undirected graph

The Quest For Food

Can we calculate shortest distance to all nodes from Sieg 226?

![Diagram of a network with cities and distances]

The Trouble with Negative Weighted Cycles

What’s the shortest path from A to E? (or to B, C, or D, for that matter)

![Diagram of a network with cities and distances]
**Dijkstra, Edsger Wybe**

Legendary figure in computer science; now a professor at University of Texas.

Supports teaching introductory computer courses without computers (pencil and paper programming)

Supposedly wouldn’t (until recently) read his e-mail; so, his staff had to print out messages and put them in his box.

**Dijkstra’s Idea**

Adapt BFS to handle weighted graphs

Two kinds of vertices:
- **Finished vertices**
  - Shortest distance is computed
- **Unknown vertices**
  - Have tentative distance

**Dijkstra’s Idea**

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

**Dijkstra’s vs BFS**

At each step:
1) Pick vertex from queue
2) Add it to visited vertices
3) Update queue with neighbours

**Dijkstra’s Algorithm**

At each step:
1) Select the unknown node with the lowest cost: n
2) Mark n as known
3) For each node a which is adjacent to n
   a’s cost = min(a’s old cost, n’s cost + cost of (n, a))

**Dijkstra’s 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 the unknown node with the lowest cost: n
   Mark n as known
   For each node a which is adjacent to n
      a’s cost = min(a’s old cost, n’s cost + cost of (n, a))

**Dijkstra’s Algorithm in Action**

<table>
<thead>
<tr>
<th>vertex</th>
<th>known</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm for Single Source, Shortest Path
• Classic algorithm for solving shortest path in weighted graphs without negative weights
• A greedy algorithm (irrevocably makes decisions without considering future consequences)
• Intuition:
  – shortest path from source vertex to itself is 0
  – cost of going to adjacent nodes is at most edge weights
  – cheapest of these must be shortest path to that node
  – update paths for new node and continue picking cheapest path

The Known Cloud
Better path to V? Not!

Inside the Cloud (Proof)
Prove by induction on # of nodes in the cloud:
Initial cloud is just the source with shortest path 0
Assume: Everything inside the cloud has the correct shortest path
Inductive step: Once we prove the shortest path to some node V (which is not in the cloud) is correct, we add it to the cloud

Data Structures for Dijkstra’s Algorithm
$|V|$ times: Select the unknown node with the lowest cost
$|E|$ times:
- $\text{findMin/deleteMin}$
- $\text{decreaseKey}$
- $\text{find by name}$

When does Dijkstra’s algorithm not work?

Graphs are Really Important!