Dijkstra's Single Source Shortest Path Algorithm

Andreas Klappenecker
Single Source Shortest Path

Given:
- a directed or undirected graph $G = (V,E)$
- a source node $s$ in $V$
- a weight function $w: E \rightarrow \mathbb{R}$.

Goal: For each $v$ in $V$, find a path of minimum total weight from the source node $s$ to $v$. 
Single Source Shortest Path

Given:
- a directed or undirected graph $G = (V, E)$
- a source node $s$ in $V$
- a weight function $w: E \rightarrow \mathbb{R}$.

Goal: For each $v$ in $V$, find a path of minimum total weight from the source node $s$ to $v$. 
Special Case

Suppose that the weights of all edges are the same. Then breadth-first search can be used to solve the single-source shortest path problem.

Indeed, the tree rooted at s in the BFS forest is the solution.

**Goal:** Solve the more general problem of single-source shortest path problems with arbitrary (non-negative) edge weights.
Intermezzo: Priority Queues
Priority Queues

A min-priority queue is a data structure for maintaining a set $S$ of elements, each with an associated value called key. It supports the operations:

- $\text{insert}(S,x)$ which realizes $S := S \cup \{x\}$
- $\text{minimum}(S)$ which returns the element with the smallest key.
- $\text{extract-min}(S)$ which removes and returns the element with the smallest key from $S$.
- $\text{decrease-key}(S,x,k)$ which decreases the value of $x$'s key to the lower value $k$, where $k < \text{key}[x]$. 
Simple Array Implementation

Suppose that the elements are numbered from 1 to n, and that the keys are stored in an array key[1..n].

- insert and decrease-key take $O(1)$ time.
- extract-min takes $O(n)$ time, as the whole array must be searched for the minimum.
Binary min-heap Implementation

Suppose that we realize the priority queue of a set with \( n \) elements with a binary min-heap.

- **extract-min** takes \( O(\log n) \) time.
- **decrease-key** takes \( O(\log n) \) time.
- **insert** takes \( O(\log n) \) time.

Building the heap takes \( O(n) \) time.
Fibonacci-Heap Implementation

Suppose that we realize the priority queue of a set with n elements with a Fibonacci heap. Then

• extract-min takes $O(\log n)$ amortized time.
• decrease-key takes $O(1)$ amortized time.
• insert takes $O(1)$ time.

[One can even realize priority queues with worst case times as above]
Dijkstra’s Single Source Shortest Path Algorithm
Dijkstra's SSSP Algorithm

We assume all edge weights are nonnegative.

Start with source node $s$ and iteratively construct a tree rooted at $s$.

Each node keeps track of the tree node that provides cheapest path from $s$.

At each iteration, we include the node into the tree whose cheapest path from $s$ is the overall cheapest.
Implementation Questions

- How can each node keep track of its best path to s?
- How can we know which node that is not in the tree yet has the overall cheapest path?
- How can we maintain the shortest path information of each node after adding a node to the shortest path tree?
Dijkstra's Algorithm

Input: $G = (V,E,w)$ and source node $s$

for all nodes $v$ in $V$ {
    $d[v] := \infty$
}

$d[s] := 0$

Enqueue all nodes in priority queue $Q$ using $d$ values as keys.

while ($Q$ is not empty) {
    $u := \text{extract-min}(Q)$
    for each neighbor $v$ of $u$ {
        if ($d[u] + w(u,v) < d[v]$) { // relax
            $d[v] := d[u] + w(u,v)$;
            $\text{decrease-key}(Q,v,d[v])$
            $\text{parent}(v) := u$
        }
    }
}

while (Q is not empty) {
    $u := \text{extract-min}(Q)$
    for each neighbor $v$ of $u$ {
        if ($d[u] + w(u,v) < d[v]$) { // relax
            $d[v] := d[u] + w(u,v)$;
            $\text{decrease-key}(Q,v,d[v])$
            $\text{parent}(v) := u$
        }
    }
}

}
Dijkstra's Algorithm Example

Dijkstra's algorithm is an algorithm for finding the shortest path between nodes in a graph. It is based on a greedy approach, where the algorithm repeatedly selects the unvisited node with the lowest distance value, updates the distances of its neighbors, and marks it as visited.

In the example shown, node 'a' is the source node.

The algorithm proceeds as follows:

1. Initialize the distance of the source node 'a' to 0 and the distance of all other nodes to infinity.
2. Create a set Q containing all nodes except the source node 'a'.
3. While Q is not empty, do the following:
   a. Select the node u with the minimum distance value from Q.
   b. Mark u as visited.
   c. Update the distances of all unvisited neighbors of u.
4. The algorithm terminates when all nodes have been visited.

The table below shows the state of the algorithm at each iteration:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Q)</td>
<td>abcde</td>
<td>bcde</td>
<td>cde</td>
<td>de</td>
<td>d</td>
<td>Ø</td>
</tr>
<tr>
<td>(d[a])</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(d[b])</td>
<td>(\infty)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>(d[c])</td>
<td>(\infty)</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>(d[d])</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>16</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>(d[e])</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
Dijkstra's Algorithm Example

![Diagram of a graph with labeled edges and nodes, showing the steps of Dijkstra's algorithm.]

<table>
<thead>
<tr>
<th>Q</th>
<th>iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>abcde</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>∞</td>
</tr>
<tr>
<td>d</td>
<td>∞</td>
</tr>
<tr>
<td>e</td>
<td>∞</td>
</tr>
</tbody>
</table>

a is source node
Dijkstra's Algorithm Example

![Graph and Table]

- a is source node

---

**Graph**

- Nodes: a, b, c, d, e
- Edges and Weights:
  - a to b: 2
  - a to d: 10
  - a to c: 12
  - b to c: 8
  - b to d: 9
  - c to b: 4
  - c to d: 6
  - c to e: 3
  - d to e: 2

**Table**

<table>
<thead>
<tr>
<th>iteration</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>abcde</td>
<td>bcde</td>
<td>cde</td>
<td>de</td>
<td>d</td>
<td>ø</td>
</tr>
<tr>
<td>d[a]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d[b]</td>
<td>∞</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>d[c]</td>
<td>∞</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>d[d]</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>16</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>d[e]</td>
<td>∞</td>
<td>∞</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
Dijkstra's Algorithm Example

**Diagram:**
- Node a is the source node.
- Edges and distances:
  - a to b: 2
  - a to c: 12
  - a to d: 10
  - a to e: 8
  - b to c: 3
  - b to d: 9
  - c to d: 4
  - c to e: 6
  - d to c: 2

**Table:**

<table>
<thead>
<tr>
<th>Q</th>
<th>iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>abcde</td>
<td>0</td>
</tr>
<tr>
<td>bcde</td>
<td></td>
</tr>
<tr>
<td>cde</td>
<td></td>
</tr>
<tr>
<td>de</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Ø</td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra's Algorithm Example

- **Diagram:**
  - Nodes: a, b, c, d, e
  - Edges with weights:
    - a to b: 2
    - a to c: 12
    - a to d: 10
    - b to c: 8
    - b to d: 4
    - c to b: 3
    - c to d: 2
    - d to e: 9

- **Table: Iteration Recap**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>abcde</td>
<td>bcde</td>
<td>cde</td>
<td>de</td>
<td>d</td>
<td>∅</td>
</tr>
<tr>
<td>d[a]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d[b]</td>
<td>∞</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>d[c]</td>
<td>∞</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>d[d]</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>16</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>d[e]</td>
<td>∞</td>
<td>∞</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

- **Notes:**
  - a is source node
  - ∞ represents infinity
Dijkstra's Algorithm Example

a is source node

<table>
<thead>
<tr>
<th>iteration</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>abcde</td>
<td>bcde</td>
<td>cde</td>
<td>de</td>
<td>d</td>
<td>∅</td>
</tr>
<tr>
<td>d[a]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d[b]</td>
<td>∞</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>d[c]</td>
<td>∞</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>d[d]</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>16</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>d[e]</td>
<td>∞</td>
<td>∞</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
Dijkstra's Algorithm Example

**Graph:**
- **Source Node:** a
- Vertices: a, b, c, d, e
- Edges and Weights:
  - (a, b): 2
  - (a, c): 12
  - (b, c): 8
  - (b, d): 4
  - (b, e): 9
  - (c, d): 6
  - (c, e): 3
  - (d, e): 2

**Table:**
- **Iteration:**
  - 0: Q = abcde
  - 1: Q = bcde
  - 2: Q = cde
  - 3: Q = de
  - 4: Q = d
  - 5: Q = Ø
- **Distance from a:**
  - d[a]: 0 → 0 → 0 → 0 → 0 → 0 → 0
  - d[b]: ∞ → 2 → 2 → 2 → 2 → 2 → 2
  - d[c]: ∞ → 12 → 10 → 10 → 10 → 10 → 10
  - d[d]: ∞ → ∞ → ∞ → 16 → 13 → 13 → 13
  - d[e]: ∞ → ∞ → 11 → 11 → 11 → 11 → 11
Dijkstra's Algorithm Example

Dijkstra's Algorithm Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>abcde</td>
<td>bcde</td>
<td>cde</td>
<td>de</td>
<td>d</td>
<td>Ø</td>
</tr>
<tr>
<td>d[a]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d[b]</td>
<td>∞</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>d[c]</td>
<td>∞</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>d[d]</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>16</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>d[e]</td>
<td>∞</td>
<td>∞</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
Dijkstra's Algorithm Example

![Dijkstra's Algorithm Example Diagram](image)

- **a** is source node

**Table:**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Q</strong></td>
<td>abcde</td>
<td>bcde</td>
<td>cde</td>
<td>de</td>
<td>d</td>
<td>Ø</td>
</tr>
<tr>
<td>d[a]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d[b]</td>
<td>∞</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>d[c]</td>
<td>∞</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>d[d]</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>16</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>d[e]</td>
<td>∞</td>
<td>∞</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
Dijkstra's Algorithm Example

a is source node

<table>
<thead>
<tr>
<th></th>
<th>iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Q</td>
<td>abcde</td>
</tr>
<tr>
<td>d[a]</td>
<td>0</td>
</tr>
<tr>
<td>d[b]</td>
<td>∞</td>
</tr>
<tr>
<td>d[c]</td>
<td>∞</td>
</tr>
<tr>
<td>d[d]</td>
<td>∞</td>
</tr>
<tr>
<td>d[e]</td>
<td>∞</td>
</tr>
</tbody>
</table>
Correctness

Let $T_i$ be the tree constructed after $i$-th iteration of the while loop:

- The nodes in $T_i$ are not in $Q$
- The edges in $T_i$ are indicated by parent variables

Show by induction on $i$ that the path in $T_i$ from $s$ to $u$ is a shortest path and has distance $d[u]$, for all $u$ in $T_i$.

**Basis: $i = 1$.**

$s$ is the only node in $T_1$ and $d[s] = 0$. 
Induction: Assume $T_i$ is a correct shortest path tree. We need to show that $T_{i+1}$ is a correct shortest path tree as well.

Let $u$ be the node added in iteration $i$.

Let $x = \text{parent}(u)$.

Need to show that path in $T_{i+1}$ from $s$ to $u$ is a shortest path, and has distance $d[u]$.
Correctness

P, path in $T_{i+1}$ from s to u
Correctness

$P$, path in $T_{i+1}$ from $s$ to $u$

$P'$, another path from $s$ to $u$
Correctness

(a,b) is first edge in P' that leaves $T_i$
Correctness

Let $P_1$ be part of $P'$ before $(a,b)$.
Let $P_2$ be part of $P'$ after $(a,b)$.

\[
w(P') = w(P_1) + w(a,b) + w(P_2) \\
\geq w(P_1) + w(a,b) \quad (\text{since weight are nonnegative}) \\
\geq \text{wt of path in } T_i \text{ from } s \text{ to } a + w(a,b) \quad (\text{inductive hypothesis}) \\
\geq w(s\rightarrow x \text{ path in } T_i) + w(x,u) \quad (\text{alg chose } u \text{ in iteration } i \text{ and } d\text{-values are accurate, by I.H.}) \\
= w(P).
\]

So $P$ is a shortest path, and $d[u]$ is accurate after iteration $i+1$. 
Running Time

Initialization: insert each node once

- $O(V T_{\text{ins}})$

$O(V)$ iterations of while loop

- one extract-min per iteration => $O(V T_{\text{ex}})$
- for loop inside while loop has variable number of iterations...

For loop has $O(E)$ iterations total

- one decrease-key per iteration => $O(E T_{\text{dec}})$

Total is $O(V (T_{\text{ins}} + T_{\text{ex}}) + E T_{\text{dec}})$ // details depend on min-queue implementation
Running Time using Binary Heaps and Fibonacci Heaps

Recall, total running time is $O(V(T_{ins} + T_{ex}) + E\cdot T_{dec})$

If priority queue is implemented with a binary heap, then

- $T_{ins} = T_{ex} = T_{dec} = O(\log V)$
- total time is $O(E \log V)$

There are fancier implementations of the priority queue, such as Fibonacci heap:

- $T_{ins} = O(1), T_{ex} = O(\log V), T_{dec} = O(1)$ (amortized)
- total time is $O(V \log V + E)$
Running Time using Simpler Heap

In general, running time is $O(V(T_{\text{ins}} + T_{\text{ex}}) + E \cdot T_{\text{dec}})$. If graph is dense, say $|E| = \Theta(V^2)$, then $T_{\text{ins}}$ and $T_{\text{ex}}$ can be $O(V)$, but $T_{\text{dec}}$ should be $O(1)$.

Implement priority queue with an unsorted array:

- $T_{\text{ins}} = O(1)$, $T_{\text{ex}} = O(V)$, $T_{\text{dec}} = O(1)$
- Total running time is $O(V^2)$
Credits

This set of slides is based on slides prepared by Jennifer Welch.