Shortest Path Algorithms

Josh Wilcox (jw14g24@soton.ac.uk)

March 11, 2025

Contents

1 Single Source Shortest Path

- Looks for the **minimal cost path** from a *single source* to all other vertices of the graph
- Let $G = (V, E)$ with the weight function:

$$w : E \to \mathbb{R}$$

- The total wieght of a path $p = (v_0, \dots, v_k)$:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Shortest path cost between u and v:

$$\delta(u, v) = \begin{cases} \min\ w(p) & \text{if there is a path from} u \to v \\ \infty otherwise \end{cases}$$

1.1 Negative Path Cycles

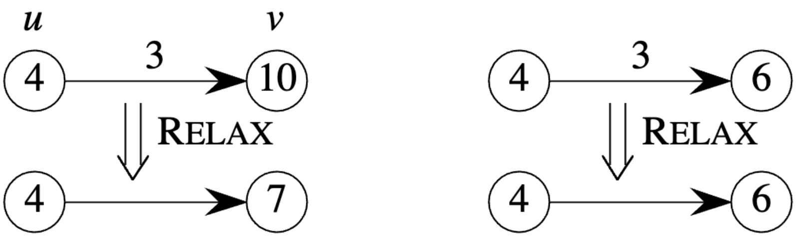
- If a cycle exists that have a negative total weight, it is undefined to get a shortest path as you could just keep going round and round to get a more and more negative total weight

1.2 Representation of Shortest Paths

- Maintain for every vertex v its predecessor $v.\pi$
- At termination $v.\pi$ will be the predecessor of v on a shortest path from source $s \to v$
- Maintain a value $v.\delta$ which will be the value of the shortest path cost from source $s \to v$
- During the execution of the algorithm $v.\delta$ will be an **upper bound** on the value of the shortest path cost

2 Relaxation

- Relaxing an edge (u, v) means testing if we can improve the shortest path cost of a graph by using the edge (u, v)
- If using this new edge causes a good change, we can update $v.\delta$ and $v.\pi$



Algorithm 1 RELAX(u, v)

if $v.\delta > u.\delta + w(u, v)$ **then**
  $v.\delta \leftarrow u.\delta + w(u, v)$
  $v.\pi \leftarrow u$
end if

3 Initialising a graph for shortest path

Algorithm 2 INITIALIZE(G, s)

for $v \in V$ **do**
    $v.\delta \leftarrow \infty$
    $v.\pi \leftarrow$ NULL
end for
$s.\delta \leftarrow 0$

4 Bellman-Ford Algorithm

- Computes the shortest path from a given source to all other nodes in the graph
- Uses `RELAX` on all the edges in the graph $|V| - 1$ times
- That's literally it
  - Set the source node to have distance 0 and the rest ∞
  - `RELAX` on the set of edges in order $|V| - 1$ times

Algorithm 3 Bellman-Ford Algorithm

Algorithm 1: Bellman-Ford algorithm
INITIALIZE(G, s)
for $i \leftarrow 1$ to $|V| - 1$ **do**
    for each $(u, v) \in E$ **do**
        RELAX(u, v)
    end for
end for

4.1 Complexity

- Initialisation is $\mathcal{O}(|V|)$
- Double Loop is $\mathcal{O}(|V| \cdot |E|)$
- Total Cost is $\mathcal{O}(|V| \cdot |E|)$
- Slower than Dijkstra's but can handle negative weights

5 Dijkstra's Algorithm

- Works only when **all weights are positive**
- Faster than Bellman-Ford
- Maintains a set S of nodes whose shortest paths have been determined
- All other nodes are kept in a min-priority queue to keep track of the next node to process

5.1 Pseudocode

Algorithm 4 Dijkstra's Algorithm

1: **DIJKSTRA**(G, s)
2: **INITIALIZE**(G, s) $\triangleright$ $O(n)$
3: $S \leftarrow \emptyset$ $\triangleright$ $O(1)$
4: $Q \leftarrow V$ $\triangleright$ $O(n)$
5: **while** $Q \neq \emptyset$ **do** $\triangleright$ $O(n)$
6:     $u \leftarrow$ EXTRACT-MIN(Q) $\triangleright$ $O(\log n)$
7:     $S \leftarrow S \cup \{u\}$ $\triangleright$ $O(1)$
8:     **for** each $v \in \text{adj}[u]$ **do** $\triangleright$ $O(|\text{adj}[u]|)$
9:         **RELAX**(u, v) $\triangleright$ $O(\log n)$
10:     **end for**
11: **end while**

5.2 Time Complexity

- **INITIALIZE**(G, s) takes $O(n)$
- Line 1: $S \leftarrow \emptyset$ takes $O(1)$
- Line 2: $Q \leftarrow V$ takes $O(n)$
- Line 3: The while loop runs $O(n)$ times
- Line 4: $u \leftarrow$ EXTRACT-MIN(Q) takes $O(\log n)$
- Line 5: $S \leftarrow S \cup \{u\}$ takes $O(1)$
- Line 6: The for loop runs $O(|\text{adj}[u]|)$ times
- Line 7: **RELAX**(u, v) takes $O(\log n)$

5.3 Explanation

The **RELAX**(u, v) operation involves updating the priority queue, which takes $O(\log n)$ time because it requires adjusting the position of the vertex v in the min-heap (priority queue). This adjustment is necessary to maintain the heap property after the potential decrease in the key value of v.

5.4 Why negative weights don't work

- Dijkstra's algorithm uses a greedy approach that assumes once a vertex is included in set S, its shortest path has been found
- This assumption only holds when all edge weights are non-negative
- With negative weights, a shorter path to an already processed vertex might be discovered later
- Example: Consider vertices $s \to a \to b$ with weights $w(s, a) = 2$ and $w(a, b) = 1$
  - Dijkstra would process a with distance 2, then b with distance 3
  - If there was also an edge $s \to c \to b$ with $w(s, c) = 3$ and $w(c, b) = -2$
  - This path would have total weight 1, which is better than 3
  - But b was already processed and won't be reconsidered
- The algorithm terminates prematurely, missing potentially shorter paths
- This is why Bellman-Ford is used for graphs with negative weights, as it repeatedly relaxes all edges