Dynamic Programming Part 4

Arnav Gupta

April 9, 2024

Contents

1 Bellman-Ford Algorithm

1.1 Outlook

Source is fixed.

If no negative cycle, compute all distances $\delta(s, v)$.

Can detect negative cycles.

Very simple pseudo-code, but slower than Dijkstra's.

1.2 Algorithm

1.2.1 Definition

For i from 0 to $n-1$ set $\delta_i(s,v)$ to be the length of the shortest path from s to v with at most i edges.

If no such path exists $\delta_i(s,v) = \infty$.

1.2.2 Observations

This gives $\delta_0(s,s) = 0$ and $\delta_0(s,v) = \infty$ for any other v.

If there is no negative cycle, $\delta_{n-1}(s,v) = \delta(s,v)$ (shortest paths are simple).

In any case, $\delta(s,v) \leq \delta_i(s,v)$ for all i and for all v.

1.2.3 Recurrence

$$\delta_i(s,v) = \min(\delta_{i-1}(s,v), \min_{(u,v)\in E} \delta_{i-1}(s,u) + w(u,v))$$

1.3 Pseudocode 1

```
d0 = [0, inf, ..., inf]
parent = [s, 0, ..., 0]
for i in range(1, n-1):
    for all v in V:
        di[v] = d(i-1)[v]
        for all (u,v) in E:
            if d(i-1)[u] + w(u,v) < di[v]:
                di[v] = d(i-1)[u] + w(u,v)
                parent[v] = v
```

1.3.1 Correctness

$d_i[v] = \delta_i(s,v)$ so $d_{n-1}[v] = \delta(s,v)$ if no negative cycles

1.4 Pseudocode 2

Idea: use a single array d

```
d = [0, inf, ..., inf]
parent = [s, 0, ..., 0]
for i in range(1, n-1):
    for all (u,v) in E:
        if d[u] + w(u,v) < d[v]:
            d[v] = d[u] + w(u,v)
            parent[v] = u
```

Runtime: $O(mn)$

1.4.1 Correctness

Claim 1: For all i, after iteration i, $d[v] \leq d_i[v]$ for all v

Proof: by induction:

- true for $i = 0$ so suppose true at index $i - 1$ and prove true at i
- at the beginning of the loop, for all v, $d[v] \leq d_{i-1}[v]$
- $d[v]$ can only decrease, so this stays true throughout the loop
- $d[v]$ is replaced by $\min(d[v], \min_{(u,v)\in E}(x + w(u, v)))$ where $x \leq d_{i-1}[u]$
- so at the end of iteration i, $d[v] \leq d_i[v]$

Relaxation: the operation $d[v] = \min(d[v], d[u] + w(u, v))$

Claim 2: $d[v]$ can only decrease through a relaxation, and if $\delta(s, u) \leq d[u]$ and $\delta(s, v) \leq d[v]$ before a relaxation, then $\delta(s, v) \leq d[v]$ post-relaxation

Proof:

- obvious that $d[v]$ can only decrease through a relaxation
- second item: $\delta(s, v) \leq \delta(s, u) + w(u, v)$ by the triangle equality, so $\delta(s, v) \leq d[u] + w(u.v)$, but also $\delta(s, v) \leq d[v]$

Consequence: if all $d[v]$ satisfy $\delta(s, v) \leq d[v]$ and any number of relaxations are applied, all inequalities stay true

Claim 3: for $i = 0, \ldots, n - 1$, after iteration i, $\delta(s, v) \leq d[v] \leq \delta_i(s, v)$ for all v

1.5 Summary

1.5.1 No Negative Cycle

At the end $d[v] = \delta(s, v)$ for all v. In particular, for any edge (u, v), $d[v] \leq d[u] + w(u, v)$ by the triangle inequality

1.5.2 Negative Cycle

Let this negative cycle be $v_1, \ldots, v_k$ where $v_k = v_1$

Claim: There must be an edge (v_i, v_{i+1}) with $d[v_{i+1}] > d[v_i] + w(v_i, v_{i+1})$, otherwise $d[v_{i+1}] \leq d[v_i] + w(v_i, v_{i+1})$ for all i. Then, sum and derive a contradiction.

For an extra $O(m)$, can check the presence of a negative cycle.

2 Floyd-Warshall Algorithm

No fixed source: computes all distances $\delta(u, v)$

Negative weights are fine, but no negative cycle.

Simple pseudo-code, but slower than other algorithms.

Doing Bellman-Ford from all u takes $O(mn^2)$.

2.1 Subproblems for DP

Bellman-Ford uses paths with fixed numbers of steps.

Floyd-Warshall restricts which vertices can be used.

2.2 Definition

For i from 0 to n, set $D_i(v_j, v_k)$ to the length of the shortest path from v_j to v_k with all intermediate vertices in $v_1, \ldots, v_i$.

For $i = 0$:

- $D_0(v_j, v_j) = 0$
- $D_0(v_j, v_k) = w(v_j, v_k)$ if there is an edge (v_j, v_k)
- $D_0(v_j, v_k) = \infty$ otherwise

$D_n(v_j, v_k) = \delta(v_j, v_k)$

2.3 Correctness

$D_i(v_j, v_k) = \min(D_{i-1}(v_j, v_k), D_{i-1}(v_j, v_i) + D_{i-1}(v_i, v_k))$

Proof: either the shortest path does not go through v_i or it does (if so, only once)

2.4 Pseudocode

```
FloydWarshall(G):
    setup D0 above
        for i in range(1, n):
            for j in range(1, n):
                for k in range(1, n):
                    Di[vj, vk] = min(D(i-1)[vj, vk], D(i-1)[vj, vi] + D(i-1)[vi, vk])
```

Runtime and memory: $\Theta(n^3)$