Bellman-Ford algorithm for Single-Source Shortest Path

Given $G = (V, E)$ and source $s \in V$. Find shortest distance from s to every reachable node $v \in V$. Compute $dist[v]$ for every $v \in V$. Also compute $parent[v]$ for all $v \in V$, where $parent[v]$ is the predecessor of v on a shortest path from s to v.

```
Function Initialize-Single-Source (G, s)
    for v ∈ V do
        dist[v] ← ∞
        parent[v] ← NIL
    dist[v] ← 0
Function Relax (u, v, w)
    if dist[v] > dist[u] + w(u, v) then
        dist[v] ← dist[u] + w(u, v)
        parent[v] ← u
Function Bellman-Ford (G, s, w)
    Initialize-Single-Source (G, s)
    for i = 1 to |V| − 1 do
        for (u, v) ∈ E do
            Relax (u, v, w)
    for (u, v) ∈ E do
        if dist[v] > dist[u] + w(u, v) then
            return False
    return True
```

Proposition. *Properties of shrotest paths of relaxation*

1. ***Triangular Inequality***

$$\delta(s, v) \leq \delta(s, u) + w(u, v) \qquad \textit{for all } (u, v) \in E$$

2. ***Upper bound property***

$$dist[v] \geq \delta(s, v) \qquad \textit{for all } v \in V$$

Implies if $dist[v] = \delta(s, v)$ then $dist[v]$ stays the same forever

3. ***No-path property*** *If there is no path from s to v then $dist[v] = \delta(s, v) = \infty$*

4. ***Convergence property*** *If $s \xrightarrow{p} u \to v$ is a shortest path from s to v and $dist[v] = \delta(s, u)$, then if you relax the edge (u, v), you obtain $dist[v] = \delta(s, v)$*

5. ***Path-Relaxation property*** *If $p = \langle v_0, v_1, \cdots, v_k \rangle$ is a shortest path from $v_0 = s$ to v_k, and we relax the edges (v_0, v_1), (v_1, v_2), $\cdots$, (v_{k-1}, v_k) in that order, then*

$$dist[v_k] = \delta(s, v_k)$$

1. **Complexity** $\Theta(|V||E|) = \Theta(|V|^3)$

2. **Discussion** works well on negative weights as well. Only restriction is there cannot be any negative cycle.

Dijkstra's Algorithm for Shortest Path

Given $G = (V, E)$ and source $s \in V$ and weight $w : E \to \mathbb{R}^+$, nonnegative. Find shortest distance from s to every reachable node $v \in V$. Compute $dist[v]$ for every $v \in V$. Also compute $parent[v]$ for all $v \in V$, where $parent[v]$ is the predecessor of v on a shortest path from s to v. Let $S \subseteq V$ be set of all vertices where shortest distance from s has already been found. At any iteration, we have a set S whose shortest distances from s we know, and we have $V \setminus S$

```
Function Initialize-Single-Source (G, s)
    for v ∈ V do
        dist[v] ← ∞
        parent[v] ← NIL
    dist[v] ← 0
Function Relax (u, v, w)
    if dist[v] > dist[u] + w(u, v) then
        dist[v] ← dist[u] + w(u, v)
        parent[v] ← u
Function Dijkstra-Shortest-Path (G, s, w)
    Initialize-Single-Source (G, s)
    S ← Ø
    Q ← V
    while Q ≠ Ø do
        u ← Extract-Min(Q)
        S ← S ∪ {u}
        for v ∈ Adj[u] do
            Relax(u, v, w)
```

Complexity

1. $|V|$ iterations for while loop, Extract-Min takes $O(\lg |V|)$ each time

2. Inner for loop runs for $|E|$ times in total

3. In summary, if Q implemented with, then

    (a) **array** $\Theta(|V|^2)$

(b) **binary heap** $\Theta(|E|\lg|V| + |V|\lg|V|) = \Theta(|E|\lg|V|)$ because in relax step in worst case every vertices has *dist* updated which takes $O(\lg|V|)$ each time for a total of $|E|$ vertices

(c) **Fibonacci heap** $\Theta(|V|\lg|V| + |E|)$

Proof of correctness

Proposition. *At the start of each iteration in the while loop, we have*

$$dist[v] = \delta(s, v) \qquad \textit{for all } v \in S$$

Proof. Proof by contradiction. Assume there is a first vertex u such that $u \in S$ and $dist[u] > \delta(s, u)$ Note $u \neq s$ (since it is $dist[s] = 0$ is shortest) and $\delta[u] \neq \infty$ (i.e. reachable from s) In particular there is a shortest path $s \xrightarrow{p} u$ in S. Consider $s \xrightarrow{p_1} x \to y \xrightarrow{p_2} u$. Note here $x \in S$, $y \in Q$, and all vertices in p_1 are in S. Since $x \in S$ and u is the first vertex to defy the condition we have $dist[x] = \delta(s, x)$ When we added x to S we relaxed the edge (x, y). By convergence property we have $dist[y] = \delta(s, y)$. Since y occurs before u in that path $\delta(s, y) \leq \delta(s, u)$. By upper-bound property, we have $\delta(s, u) \leq dist[u]$.

$$dist[y] = \delta(s, y) \leq \delta(s, u) \leq dist[u]$$

Since $y, u \in Q$ and we chose u to add to S, so

$$dist[y] \geq dist[u]$$

Then in particular

$$dist[y] = \delta(s, y) = \delta(s, u) = dist[u]$$

The latte equality contradicts with assumption. □

All-pairs Shortest Path

Given $G = (V, E)$ and weight w. Want to find $dist[i, j]$ which is the shortest distance between vertices $v_i, v_j \in V$.

```
Function All-Pairs (G, w)
    for s ∈ V do
        Single-Source(G, w, s)
```

Complexity

1. Non-negative weights: Use *Dijkstra* for single source

$$\Theta(|V|^2 \lg|V| + |E||V|) \leq \Theta(|V|^3)$$

2. Potentially negative weights: Use Bellman-Ford for single source

$$\Theta(|V|^2|E|) \leq \Theta(|V|^4)$$

Dynamic programming approach

1. **Optimal substructure** Let l_{ij}^{m} be shortest distance from i to j using at most m edges
$$l_{ij}^{(m)} = Min\{l_{ij}^{(m)}, \underset{1\leq k\leq |V|}{Min}\{l_{ik}^{(m+1)} + w_k j\}\}$$
where
$$l_{ij}^{(0)} = \begin{cases} 0 & \text{i = j} \\ \infty & \text{otherwise} \end{cases}$$

2. **bottom up approach**

```
Function Extend-Shortest-Path (L, w)
    n ← number of rows of L
    L′ = (l′_ij) ← n × n matrix
    for i = 1 to |V| do
        for j = 1 to |V| do
            l′_ij ← ∞
            for k = 1 to |V| do
                l′_ij ← {l′_ij, l_jk + w_kj}
    return L′
```

Complexity is $\Theta(|V|^3)$

another approach

1. **Optimal substructre** Let $d_{ij}^{(k)}$ is length of a shortest path $i \xrightarrow{p} j$ where for all $v \in p$ we have $v \subseteq \{1, \cdots, k\}$
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & k = 0 \\ Min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & k \geq 1 \end{cases}$$
So if k is on the shortest path then $d_{ij}^{(k)} = d_{ij}^{(k-1)}$, otherwise it is summation of shortest distance with vertices up to but no including k where
$$d_{ij}^{(0)} = \begin{cases} w_{ij} & (i,j) \in E \\ \infty & otherwise \end{cases}$$

2. **Algorithm**

Function `Floyd-Warshall` (W)
    $n \leftarrow$ number of rows of W
    $D^{(0)} \leftarrow W$
    for $k = 1$ **to** $|V|$ **do**
        $D^{(k)} = (d_{ij}^{(k)}) \leftarrow$ be $n \times n$ matrix
        for $i = 1$ **to** $|V|$ **do**
            for $i = 1$ **to** $|V|$ **do**
                $d_{ij}^{(k)} \leftarrow Min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$
    return D

Transitive closure of G

Given $W = (w_{ij})_{n \times n}$ where w_{ij} is weight of edge (i, j). Transitive closure is a matrix consisting of 1 and 0, where 1 represent if there is path from i to j, whereas 0 represent if there is i is not reachable from j. One approach, compute W^n with $\Theta(n^3 \lg n)$

1. **optimal substructure** let

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

where

$$t_{ij}^{(0)} = \begin{cases} 1 & i = j \text{ or } (i, j) \in E \\ 0 & otherwise \end{cases}$$