Floyd Warshall Algorithm (All Pairs)

Frank

November 7, 2021

1 Floyd Warshall

1.1 Problem

- **Input:**
  - $G = (V, E)$ directed graph
  - $wt : E \to \mathbb{R}$ [assume neg-weight cycles]
- **Output:**
  - $\forall u, v \in V$, give $\delta(u, v) =$ min weight of $u \to v$ path.

1.2 Modifying Bellman-Ford (Attempt at all pairs problem)

- Let $L[u, v, k] =$ min weight of $u \to v$ path with at most k edges.

  Then

$$L[u, v, k] = min_{w:(w,v)\in E} L[u, w, k-1] + wt(w, v)$$

---

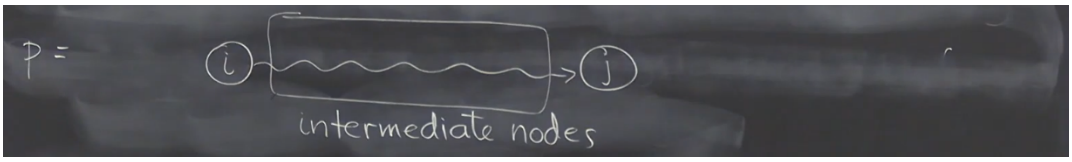
procedure MODIFIED-BF
    Init $L[u, v, 0] \forall u, v$
    for $(k = 1..n)$ **do**
        for (each $u \in V$) **do**
            for (each $v \in V$) **do**
                for (each pred w of v) **do**
                    update $L[u, v, k]$

This is $O(n^4)$ or $O(n^2m)$, similar to doing BF n times

1.3 Subproblems

WLOG (without loss of generality), assume $V = \{1, 2, \dots n\}$

Define intermediate nodes as follows:

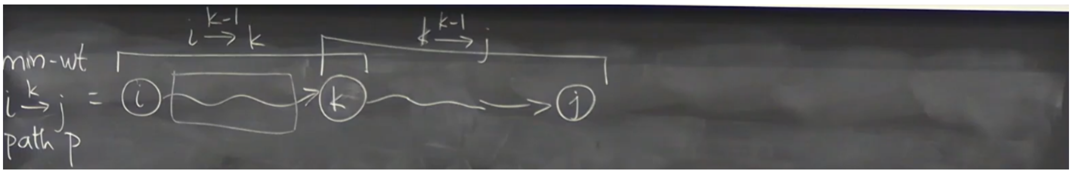


Define $i \xrightarrow{k} j$ path: only nodes in $\{1, 2, \dots, k\}$ are intermediate nodes

- Then $C[i, j, k] =$ min weight of $i \xrightarrow{k} j$ path (∞ if no $i \xrightarrow{k} j$ path). $(\star)$

  Then $C[i, j, k]$ is our constraint. Note that here, k limits which nodes we can visit (from $1 \to k$), but in BF, k limits the number of edges we can use. Thus compute the original problem using $C[i, j, n]$.

- Consider min weight $i \xrightarrow{k} j$ path P, consider cases:
  1. k not an intermediate node in $P \implies C[i, j, k] = C[i, j, k-1]$
  2. k is an intermediate node in P. WLOG, only one k (assumed no neg-weighted cycles). $\implies C[i, j, k] = C[i, k, k-1] + C[k, j, k-1]$



Then we have recurrence:

$$C[i,j,k] = \begin{cases} 0 & k = 0, i = j \\ wt(i,j) & k = 0, (i,j) \in E \\ \infty & k = 0, (i,j) \notin E \\ min\{\overset{\text{case 1}}{C[i,j,k-1]}, \overset{\text{case 2}}{C[i,k,k-1] + C[k,j,k-1]}\} & k > 0 \end{cases} \qquad (\dagger)$$

1.4 Algorithm

procedure FW(G, wt)
    for $(i = 1..n)$ **do**
        for $(j = 1..n)$ **do**
            if $(i = j)$ **then**
                $C[i, j, 0] \leftarrow 0$
            else if $((i, j) \in E)$ **then**
                $C[i, j, 0] \leftarrow wt(i, j)$
            else
                $C[i, j, 0] \leftarrow \infty$
    for $(k = 1..n)$ **do**
        for $(i = 1..n)$ **do**
            for $(j = 1..n)$ **do**
                $C[i, j, k] = min\{\overset{\text{case 1}}{C[i, j, k-1]}, \overset{\text{case 2}}{C[i, k, k-1] + C[k, j, k-1]}\}$
    return $C[-, -, n]$ ▷ return all pairs

1.5 Space/Time Complexity

- Running time is trivially $O(n^3)$
- Space complexity is $O(n^3)$ (3-d array)

  Better is $O(n^2)$ (2 $n \times n$ array)

  Best is $O(n^2)$ (1 $n \times n$ array). Left as an exercise to show why overwriting k is still correct.

1.6 Computing actual path

Keep track of predecessor of j on shortest $i \xrightarrow{k} j$ path for every i, j, k.

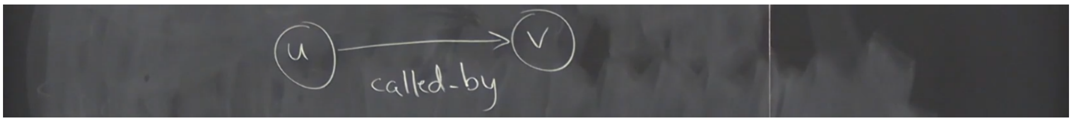
Predecessor does not change in case 1, predecessor of j changes to the predecessor of $k \xrightarrow{k-1} j$ path in case 2

- Or we can simply use $C[-, -, n]$ to backtrace.

1.7 Transitive closure of digraph $G = (V, E)$

Denote $G^\star = (V, E^\star)$ with same nodes as G, then $(u, v) \in E^\star \iff \exists u \to v$ path in G

- Suppose we have a graph where nodes are represented by procedures, and edges represent "called-by"



Suppose u is changed, we want to know what effect of this change has on the entire graph. In other words, we want to know the **transitive closure** of this graph (replace path by edge). $G^\star$ does this.

We can actually use FW algorithm to compute $G^\star$ in $O(n^3)$.

> Compute all $C[i, j, k]$, and if $C[i, j, n]$ is not 0, then there is a path from i to j, and all non-zero entries correspond to edges that are in the transitive closure.

- Another way is to let

$$C[i, j, k] = \begin{cases} 1 & \exists i \xrightarrow{k} j \text{ path} \\ 0 & \text{otherwise} \end{cases}$$

Then take $k = n$, and compute a matrix that represents the transitive closure of G

Then the recurrence is

$$C[i, j, k] = C[i, j, k-1] \vee (C[i, k, k-1] \wedge C[k, j, k-1])$$

1.8 Detecting Negative Weighted Cycles With FW

Claim 4: G has a neg-weight cycle $\iff \exists u : C[u, u, n] < 0$

> Left as an exercise to prove that it is true.

2 Johnson's Algorithm (all pairs non-negative)

2.1 All Pairs Using Dijkstra's

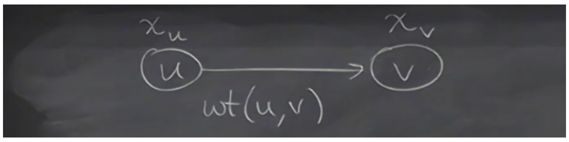
Run Dijkstra's algorithm n times, running time is $O(n \times m \log n)$, which is better than $O(n^3)$, which is the running time for FW, but requires all edges to have non-neg weight.

2.2 Introduction

The idea then is to reweight the edges so that

1. $\forall e \in E$, e has $wt \geq 0$
2. reweighing preserves shortest paths

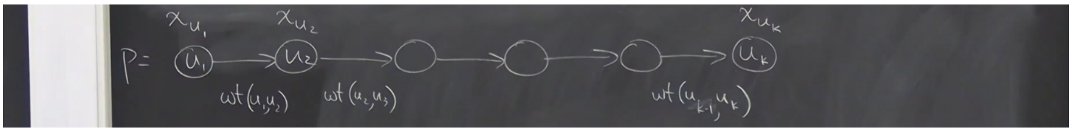
- Suppose we have u, v and $wt(u, v)$, lets assign a weight to every node as well.



Define new weight and think of x_u and x_v as **potential**

$$wt'(u,v) = wt(u,v) + \overbrace{x_u - x_v}^{\Delta \text{ potential}}$$

- Now consider a path P as follows:



Then

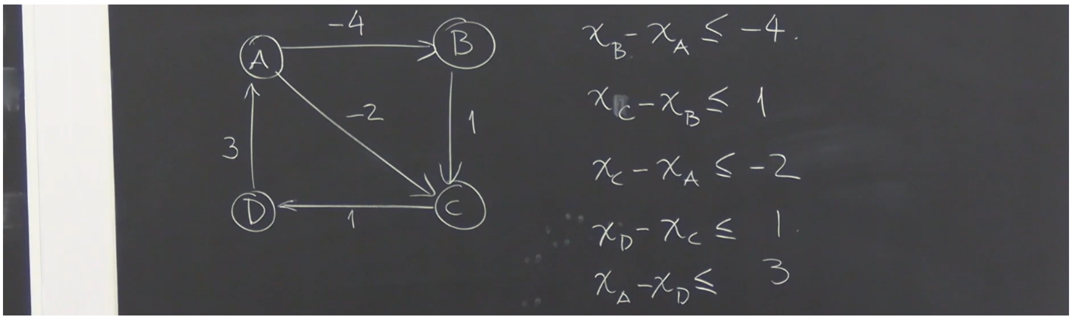
$$\begin{aligned} wt'(P) &= wt(u_1, u_2) + x_{u_1} - \cancel{x_{u_2}} \\ &+ wt(u_2, u_3) + \cancel{x_{u_2}} - \cancel{x_{u_3}} \\ &+ \ldots + \\ &+ wt(u_{k-1}, u_k) + \cancel{x_{u_{k-1}}} - x_{u_k} \\ &= wt(P) + x_{u_1} - x_{u_k} \end{aligned}$$

So our reweighing preserves shortest path (ensures 2)

- For (1), we need

$$\forall(u,v) \in E, wt'(u,v) = wt(u,v) + x_u - x_v \geq 0$$
$$\forall(u,v) \in E,\ x_v - x_u \leq wt(u,v) \quad (\star\star)$$

Consider the following graph, we are looking for:



Claim 5: Inequalities $(\star\star)$ are satisfiable $\iff$ G has no neg-weight cycle

Proof ($\Longrightarrow$):

$\forall u$, let $\hat{x}_u$ be value for x_u satisfying $(\star\star)$
Let $C = u_1, u_2, \ldots, u_k, u_1$ be any cycle in G
Then by $(\star\star)$

$$\begin{aligned}
\hat{x}_{u_2} - \hat{x}_{u_1} &\leq wt(u_1, u_2) \\
\hat{x}_{u_3} - \hat{x}_{u_2} &\leq wt(u_2, u_3) \\
&\vdots \\
\hat{x}_{u_k} - \hat{x}_{u_{k-1}} &\leq wt(u_{k-1}, u_k) \\
\hat{x}_{u_1} - \hat{x}_{u_k} &\leq wt(u_k, u_1)
\end{aligned}$$

Adding up all these rows yields

$$0 \leq wt(C)$$

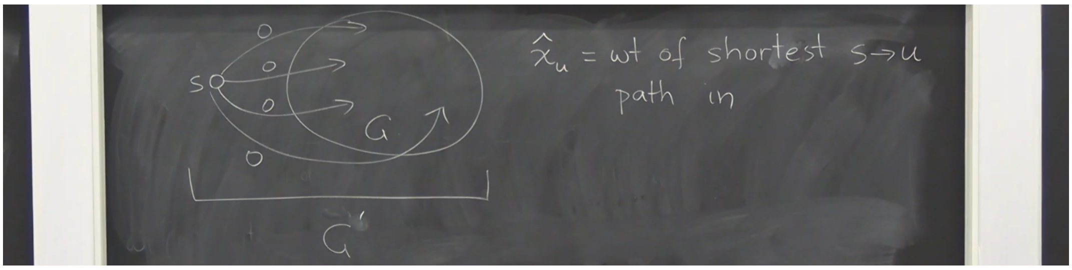
Proof ($\Longleftarrow$):

Note that

$$x_v \leq x_u + wt(u,v)$$

Which holds if x_v is the shortest path from $s \to v$ and x_u is the shortest path from $s \to u$
Then use our trick from BF algorithm and create a fake node s, and put 0 weight edges from s to every node.

Thus, we can assign $\hat{x}_u$ to the following



Note we can do this because we assumed G has no neg-weighted cycles.

This satisfies ($\star\star$)

2.3 Johnson's Algorithm

1. Define $G' = (V', E')$ from $G = (V, E)$

   where $V' = V \cup \{s\}$ and $E' = E \cup \{(s, u) : u \in V\}$

   Let $wt(s, u) = 0 \ \forall u$

2. Run BF on (G', s, wt) to compute $x_u = wt$ of shortest $s \to u$ path under wt or to detect a neg wt cycle.

3. If G' under wt has a neg wt cycle, then return "undefined"

4. $\forall (u, v) \in E.\ wt'(u, v) = wt(u, v) + x_u - x_v$

5. For each $u \in V$ do

   Run Dijkstra(G, u, wt')

   $D'[u, v] = wt$ of shortest $u \to v$ path under wt'

6. For each $u, v \in V$ do

   $D[u, v] = D'[u, v] + x_v - x_u$ (Reversing the changes)

2.4 Running Time

Dominated by step 5: Thus is $O(n \times m \log n)$

Given sparse graph, this algorithm a bit worse than $O(n^2)$, which is a lot better than FW