All-Pairs Shortest Path

Data Structures and Algorithms

Nanjing University, Fall 2021
郑朝栋

SSSP and APSP

- **Single-Source Shortest Paths (SSSP) Problem**:
  Given a graph $G = (V, E)$ and a weight function w, given a source node s, find a shortest path from s to every $u \in V$.
- **All-Pairs Shortest Paths (APSP) Problem**:
  Given a graph $G = (V, E)$ and a weight function w, for every pair $(u, v) \in V \times V$, find a shortest path from u to v.

APSP from multiple SSSP

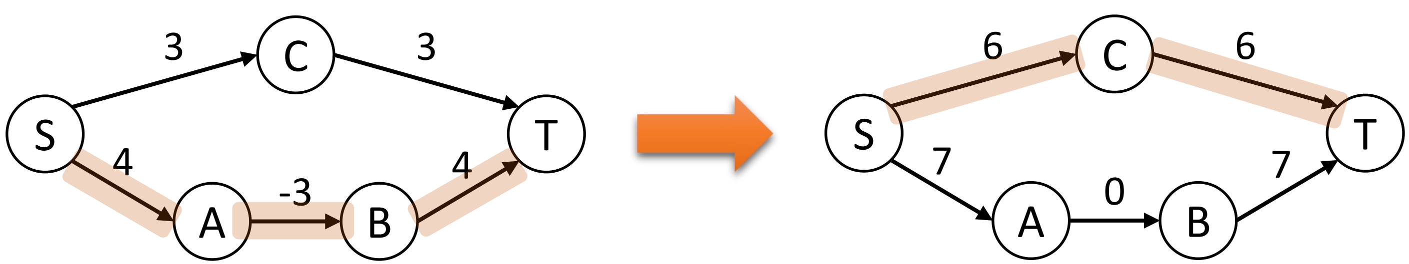
Straightforward solution for APSP:

For each $v \in V$, execute SSSP algorithm once!

	SSSP	APSP
BFS (Unit-weight Graphs)	$O(n+m) = O(n^2)$	$O(n^3)$
Dijkstra (Positive-weight Graphs)	$O((n+m)\lg n) = O(n^2 \lg n)$ (using binary heap for priority queue)	$O(n^3 \lg n)$
Bellman-Ford (Arbitrary-weight Directed)	$O(nm) = O(n^3)$	$O(n^4)$
Topological Sort Variant (Arbitrary-weight DAG)	$O(n+m) = O(n^2)$	$O(n^3)$

APSP from multiple SSSP

- **Positive-weight Graphs:** Repeating Dijkstra gives $O(n^3 \lg n)$.
  Arbitrary-weight Graphs: Repeating Bellman-Ford gives $O(n^4)$.
- Faster algorithms for arbitrary-weight graphs?
- **Intuition**: modify edge weights *without* changing shortest path, so that Dijkstra's algorithm can work.
- Add $\max\{-1 \cdot w(u,v)\}$ to each edge?
- **NO!** Shortest paths may change!
  - Given (u,v), different paths may change by different amount!



APSP from multiple SSSP

- Faster algorithms for arbitrary-weight graphs?
- **Intuition**: modify edge weights *without* changing shortest path, so that Dijkstra's algorithm can work.
- **Requirement:** $\widehat{w}(u \rightarrow^{p_1} v) > \widehat{w}(u \rightarrow^{p_2} v)$ iff $w(u \rightarrow^{p_1} v) > w(u \rightarrow^{p_2} v)$
- **Alternatively:** for every path from u to v, $\widehat{w}$ change it by **same** amount!
- $\widehat{w}(u, v) = h(u) + w(u, v) - h(v)$
- Imagine $h(u)$ is entry gift and $h(v)$ is exit tax for traveling through (u, v)
- $\widehat{w}(u \rightarrow^{p_1} v) = \widehat{w}(u \rightarrow x_1 \rightarrow \cdots \rightarrow x_k \rightarrow v) = \widehat{w}(u \rightarrow x_1) + \cdots + \widehat{w}(x_k \rightarrow v)$

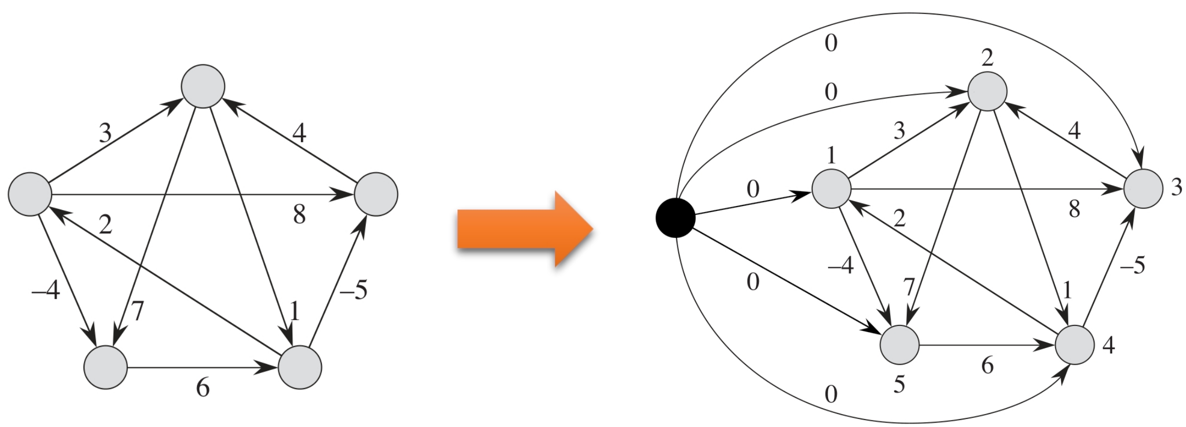
$$= (h(u) + w(u \rightarrow x_1) - h(x_1)) + (h(x_1) + w(x_1 \rightarrow x_2) - h(x_2))$$
$$+ \cdots + (h(x_{k-1}) + w(x_{k-1} \rightarrow x_k) - h(x_k)) + (h(x_k) + w(x_k \rightarrow v) - h(v))$$
$$= h(u) + w(u \rightarrow x_1) + \cdots + w(x_k \rightarrow v) - h(v)$$

APSP from multiple SSSP

- Faster algorithms for arbitrary-weight graphs?
- **Intuition**: modify edge weights *without* changing shortest path, so that Dijkstra's algorithm can work.
- **Requirement:** $\widehat{w}(u \to^{p_1} v) > \widehat{w}(u \to^{p_2} v)$ iff $w(u \to^{p_1} v) > w(u \to^{p_2} v)$
- **Alternatively:** for every path from u to v, $\widehat{w}$ change it by **same** amount!
- $\widehat{w}(u, v) = h(u) + w(u, v) - h(v) \geq 0$
- Let $h(u) = dist(z, u)$ for some *fixed* $z \in V$, then $\widehat{w}(u, v) = dist(z, u) + w(u, v) - dist(z, v) \geq 0$
- It is possible that we cannot find such z that reaches every node!

APSP from multiple SSSP

- Faster algorithms for arbitrary-weight graphs?
- **Intuition**: modify edge weights *without* changing shortest path, so that Dijkstra's algorithm can work.
- $\widehat{w}(u,v) = h(u) + w(u,v) - h(v) \geq 0$
- Add node z that goes to every node in G with a weight 0 edge.
  - $H = (V \cup \{z\}, E \cup \{(z,x) \mid x \in V\})$ with $w(z,x) = 0$.



APSP from multiple SSSP

- Faster algorithms for arbitrary-weight graphs?
- **Strategy**: modify edge weights *without* changing shortest path, so that Dijkstra's algorithm can work.
- Add node z that goes to every node in G with a weight 0 edge.
- **Reweight edges**: $\widehat{w}(u,v) = dist(z,u) + w(u,v) - dist(z,v) \geq 0$
  - For node pairs in G, addition of z does not create new shortest path.
  - For node pairs in G, a path is shortest under w iff this path is shortest under $\widehat{w}$.

JohnsonAPSP(G):

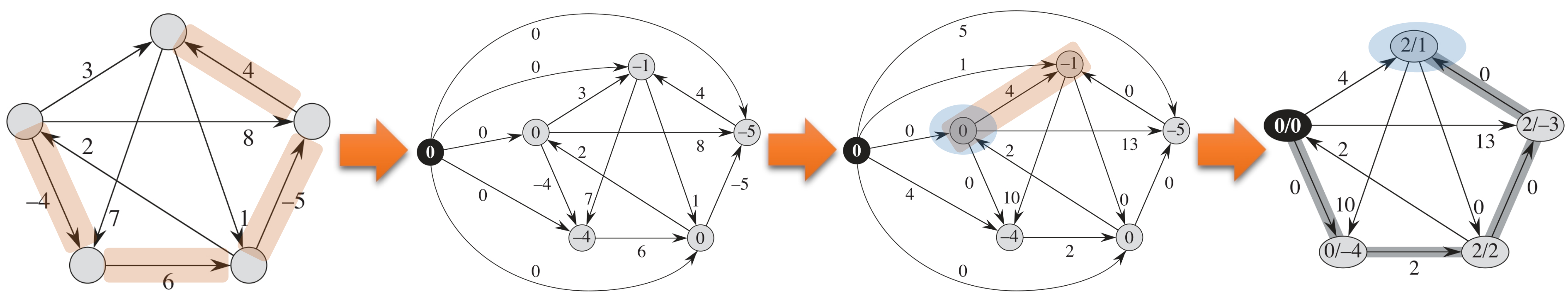
```
Create H=(V+{z},E+{(z,v)|v∈V}) with w(z,v)=0
Bellman-FordSSSP(H,z) to obtain dist_H
for (each edge (u,v) in H.E)
  w'(u,v) = dist_H(z,u)+w(u,v)-dist_H(z,v)
for (each node u in G.V)
  DijkstraSSSP(G,u) with w' to obtain dist_G,w'
  for (each node v in G.V)
    dist_G(u,v) = dist_G,w'(u,v)+dist_H(z,v)-dist_H(z,u)
```

APSP from multiple SSSP

- Faster algorithms for arbitrary-weight graphs?
- **Strategy**: *reweight* edges *without* changing shortest path, so that Dijkstra's algorithm can work.

JohnsonAPSP(G):

```
Create H=(V+{z},E+{(z,v)|v∈V}) with w(z,v)=0
Bellman-FordSSSP(H,z) to obtain dist_H
for (each edge (u,v) in H.E)
  w'(u,v) = dist_H(z,u)+w(u,v)-dist_H(z,v)
for (each node u in G.V)
  DijkstraSSSP(G,u) with w' to obtain dist_{G,w'}
  for (each node v in G.V)
    dist_G(u,v) = dist_{G,w'}(u,v)+dist_H(z,v)-dist_H(z,u)
```



APSP from multiple SSSP

Johnson's algorithm

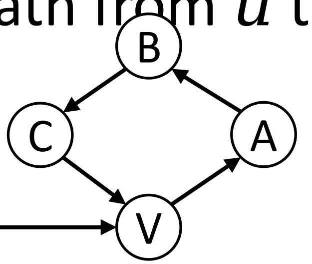
- **Positive-weight Graphs:** Repeating Dijkstra gives $O(n^3 \lg n)$.
  Arbitrary-weight Graphs: Repeating Bellman-Ford gives $O(n^4)$.
- Faster algorithms for arbitrary-weight graphs?
- **Johnson's Alg.**: *reweight* edges *without* changing shortest path, so that Dijkstra's algorithm can work.
- Johnson's algorithm combines Dijkstra and Bellman-Ford, resulting a runtime of $O(n^3 \lg n)$, for arbitrary weight graphs.

JohnsonAPSP(G):

```
Create H=(V+{z},E+{(z,v)|v∈V}) with w(z,v)=0
Bellman-FordSSSP(H,z) to obtain dist_H
for (each edge (u,v) in H.E)
  w'(u,v) = dist_H(z,u)+w(u,v)-dist_H(z,v)
for (each node u in G.V)
  DijkstraSSSP(G,u) with w' to obtain dist_G,w'
  for (each node v in G.V)
    dist_G(u,v) = dist_G,w'(u,v)+dist_H(z,v)-dist_H(z,u)
```

APSP via Recursion

- $dist(u, v) = \begin{cases} 0 & \text{if } u = v \\ min_{(x,v)\in E}\{dist(u, x) + w(x, v)\} & \text{otherwise} \end{cases}$
- This recurrence is correct,
  but it does not lead to a recursive algorithm directly!
- Cycle in the graph can make the recursion never ends!
- Introduce an **additional parameter** in the recurrence:
  $dist(u, v, l)$: shortest path from u to v that uses at most l edges.



- $dist(u, v, l) =$
  $$\begin{cases} 0 & \text{if } l = 0 \text{ and } u = v \\ \infty & \text{if } l = 0 \text{ and } u \neq v \\ \min\left\{\begin{matrix} dist(u, v, l-1) \\ \min_{(x,v)\in E}\{dist(u, x, l-1) + w(x, v)\} \end{matrix}\right\} & \text{otherwise} \end{cases}$$

APSP via Recursion

- $dist(u, v, l) =$
$$\begin{cases} 0 & \text{if } l = 0 \text{ and } u = v \\ \infty & \text{if } l = 0 \text{ and } u \neq v \\ \min \left\{ \begin{matrix} dist(u, v, l-1) \\ \min\limits_{(x,v) \in E} \{dist(u, x, l-1) + w(x, v)\} \end{matrix} \right\} & \text{otherwise} \end{cases}$$
- Evaluate this recurrence easily in a "**bottom-up**" fashion!
  - $dist(\cdot, \cdot, 0)$ are easy to compute, given input graph.
  - $dist(\cdot, \cdot, 1)$ are easy to compute, if $dist(\cdot, \cdot, 0)$ are known.
  - $dist(\cdot, \cdot, l+1)$ are easy to compute, if $dist(\cdot, \cdot, l)$ are known.
  - $dist(\cdot, \cdot, n-1)$ are what we want!
- Don't always need a recursive algorithm to evaluate recurrence, often an iterative alternative exists.

APSP via Recursion

- $dist(u,v,l) =$
$$\begin{cases} 0 & \text{if } l = 0 \text{ and } u = v \\ \infty & \text{if } l = 0 \text{ and } u \neq v \\ \min\left\{ \begin{array}{c} dist(u,v,l-1) \\ \min_{(x,v)\in E} \{dist(u,x,l-1) + w(x,v)\} \end{array} \right\} & \text{otherwise} \end{cases}$$
- Evaluate this recurrence easily in a "**bottom-up**" fashion!

RecursiveAPSP(G):

```
for (every pair (u,v) in V*V)
  if (u=v) then dist[u,v,0]=0
  else dist[u,v,0]=INF
for (l=1 to n-1)
  for (each node u)
    for (each node v)
      dist[u,v,l] = dist[u,v,l-1]
      for (each edge (x,v) going to v)
        if (dist[u,v,l] > dist[u,x,l-1] + w(x,v))
          dist[u,v,l] = dist[u,x,l-1]+w(x,v)
```

Runtime is $O(n^4)$.

Can this approach do better?

APSP via Recursion

- $dist(u, v, l) =$
$$\begin{cases} 0 & \text{if } l = 0 \text{ and } u = v \\ \infty & \text{if } l = 0 \text{ and } u \neq v \\ \min \left\{ \begin{array}{c} dist(u, v, l-1) \\ \min\limits_{(x,v) \in E} \{dist(u, x, l-1) + w(x, v)\} \end{array} \right\} & \text{otherwise} \end{cases}$$
- This recursion is like "1 and $l - 1$ split" in divide-and-conquer.
- How about "$l/2$ and $l/2$ split"?
  (You might have noticed "$l/2$ and $l/2$ split" is usually better than "1 and $l - 1$ split")
- $dist(u, v, l) =$
$$\begin{cases} w(u, v) & \text{if } l = 1 \text{ and } (u, v) \in E \\ \infty & \text{if } l = 1 \text{ and } (u, v) \notin E \\ \min\limits_{x \in V} \{dist(u, x, l/2) + dist(x, v, l/2)\} & \text{otherwise} \end{cases}$$
- Start with $dist(\cdot\,, \cdot\,, 1)$, then double l each time, until $2^{\lceil \lg n \rceil}$

APSP via Recursion

- $dist(u,v,l) = \begin{cases} w(u,v) & \text{if } l = 1 \text{ and } (u,v) \in E \\ \infty & \text{if } l = 1 \text{ and } (u,v) \notin E \\ \min_{x \in V} \{dist(u,x,l/2) + dist(x,v,l/2)\} & \text{otherwise} \end{cases}$
- Start with $dist(\cdot\,,\cdot\,,1)$, then double l each time, until $2^{\lceil \lg n \rceil}$

FasterRecursiveAPSP(G):

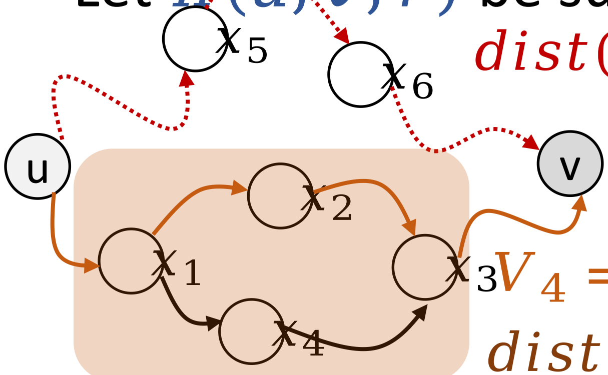
```
for (every pair (u,v) in V*V)
  if ((u,v) in E) then dist[u,v,1]=w(u,v)
  else dist[u,v,1]=INF
for (i=1 to Ceil(lg(n)))
  for (each node u)
    for (each node v)
      dist[u,v,i] = INF
      for (each node x)
        if (dist[u,v,i] > dist[u,x,i-1] + dist[x,v,i-1])
          dist[u,v,i] = dist[u,x,i-1] + dist[x,v,i-1]
```

Runtime is $O(n^3 \lg n)$.

Can this approach do better?

APSP via Recursion

- **Strategy:** recuse on the *set of node* the shortest paths use.
  (Previous algorithms recuse on # of edges the shortest paths use.)
- Number nodes arbitrarily: $x_1, x_2, \cdots, x_n$;
  Define $V_r = \{x_1, x_2, \cdots, x_r\}$.
- Define $dist(u, v, r)$ be length of shortest path from u to v,
  s.t. only nodes in V_r can be intermediate nodes in paths.
  Let $\pi(u, v, r)$ be such a shortest path.



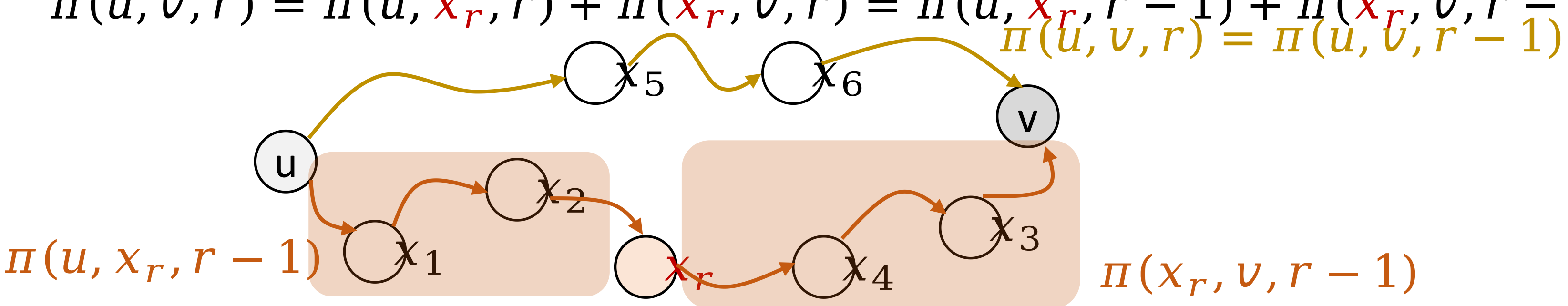
$dist(u, v) = w(u \to x_5 \to x_6 \to v)$

$V_4 = \{x_1, x_2, x_3, x_4\}$

$dist(u, v, 4) = w(u \to x_1 \to x_2 \to x_3 \to v)$

APSP via Recursion

- **Strategy:** recuse on the *set of node* the shortest paths use.
- Number nodes arbitrarily: $x_1, x_2, \cdots, x_n$; let $V_r = \{x_1, x_2, \cdots, x_r\}$.
- Define $dist(u, v, r)$ be length of shortest path from u to v,
  s.t. only nodes in V_r can be intermediate nodes in paths.
  Let $\pi(u, v, r)$ be such a shortest path.
- **Observation:** either $\pi(u, v, r)$ go through x_r or not.
- **Latter case**: $\pi(u, v, r) = \pi(u, v, r-1)$
- **Former case**:
  $\pi(u, v, r) = \pi(u, x_r, r) + \pi(x_r, v, r) = \pi(u, x_r, r-1) + \pi(x_r, v, r-1)$



APSP via Recursion

- **Strategy:** recuse on the *set of node* the shortest paths use.
- Number nodes arbitrarily: $x_1, x_2, \cdots, x_n$; let $V_r = \{x_1, x_2, \cdots, x_r\}$.
- Define $dist(u, v, r)$ be length of shortest path from u to v,
  s.t. only nodes in V_r can be intermediate nodes in paths.
  Let $\pi(u, v, r)$ be such a shortest path.
- **Observation:** either $\pi(u, v, r)$ go through x_r or not.
- **Latter case**: $\pi(u, v, r) = \pi(u, v, r-1)$
- **Former case**:
  $\pi(u, v, r) = \pi(u, x_r, r) + \pi(x_r, v, r) = \pi(u, x_r, r-1) + \pi(x_r, v, r-1)$
- $dist(u, v, r) =$

$$\begin{cases} w(u, v) & \text{if } r = 0 \text{ and } (u, v) \in E \\ \infty & \text{if } r = 0 \text{ and } (u, v) \notin E \\ \min \begin{Bmatrix} dist(u, v, r-1) \\ dist(u, x_r, r-1) + dist(x_r, v, r-1) \end{Bmatrix} & \text{otherwise} \end{cases}$$

No longer needs to enumerate multiple potential choices!

APSP via Recursion

The Floyd-Warshall Algorithm

- **Strategy:** recuse on the *set of node* the shortest paths use.
- Define $dist(u,v,r)$ be length of shortest path from u to v, s.t. only nodes in $V_r = \{x_1, x_2, \cdots, x_r\}$ can be intermediate nodes in paths.
- $dist(u,v,r) =$

$$\begin{cases} w(u,v) & \text{if } r = 0 \text{ and } (u,v) \in E \\ \infty & \text{if } r = 0 \text{ and } (u,v) \notin E \\ \min\left\{ \begin{matrix} dist(u,v,r-1) \\ dist(u,x_r,r-1) + dist(x_r,v,r-1) \end{matrix} \right\} & \text{otherwise} \end{cases}$$

FloydWarshallAPSP(G):

```
for (every pair (u,v) in V*V)
  if ((u,v) in E) then dist[u,v,0]=w(u,v)
  else dist[u,v,0]=INF
for (r=1 to n)
  for (each node u)
    for (each node v)
      dist[u,v,r] = dist[u,v,r-1]
      if (dist[u,v,r] > dist[u,x_r,r-1] + dist[x_r,v,r-1])
        dist[u,v,r] = dist[u,x_r,r-1] + dist[x_r,v,r-1]
```

Runtime is $O(n^3)$.

Application of APSP

Compute Transitive Closure

FloydWarshallAPSP(G):

```
for (every pair (u,v) in V*V)
  if ((u,v) in E) then dist[u,v,0]=w(u,v)
  else dist[u,v,0]=INF
for (r=1 to n)
  for (each node u)
    for (each node v)
      dist[u,v,r] = dist[u,v,r-1]
      if (dist[u,v,r] > dist[u,x_r,r-1] + dist[x_r,v,r-1])
        dist[u,v,r] = dist[u,x_r,r-1] + dist[x_r,v,r-1]
```

FloydWarshallTransitiveClosure(G):

$$t_{u,\ v}^{(r)} = t_{u,\ v}^{(r-1)} \vee \left(t_{u,\ x_r}^{(r-1)} \wedge t_{x_r,\ v}^{(r-1)}\right)$$

```
for (every pair (u,v) in V*V)
  if ((u,v) in E) then t[u,v,0] = TRUE
  else t[u,v,0] = FALSE
for (r=1 to n)
  for (each node u)
    for (each node v)
      t[u,v,r] = t[u,v,r-1]
      if (t[u,x_r,r-1] AND t[x_r,v,r-1])
        t[u,v,r] = TRUE
```

Reading

- [CLRS] Ch.25
- [Erickson v1] Ch.9