ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

# Discrete Mathematics

Course preparation group:
Nguyễn Khánh Phương
Đỗ Phan Thuận
Phạm Quang Dũng
Huỳnh Thanh Bình
Trần Vĩnh Đức
Bùi Quốc Trung
Đinh Viết Sang
Ban Hà Bằng

## Content of Part 2





PART 1
COMBINATORIAL THEORY
(Lý thuyết tổ hợp)

PART 2
**GRAPH THEORY**
(Lý thuyết đồ thị)

## Content

# 1. Shortest Path

- Generalize distance to weighted setting
- Digraph $G = (V,E)$ with weight function $W$: $E \rightarrow R$ (assigning real values to edges)
- Weight of path $p = v_1 \xrightarrow{w(v_1,v_2)} v_2 \xrightarrow{w(v_2,v_3)} \dots \xrightarrow{w(v_{k-1},v_k)} v_k$ is $$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$
- Shortest path = a path of the minimum weight

$$\delta(u,v) = \begin{cases} \min\{\omega(p) : u \overset{p}{\rightsquigarrow} v\}; & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise.} \end{cases}$$

- Applications
  - static/dynamic network routing
  - robot motion planning
  - map/route generation in traffic
  - speech interpretation (best interpretation of a spoken sentence)
  - medical imaging



# Shortest-Path Variants

- **Single-source shortest-paths problem**
  - Find a shortest path from a given source (vertex $s$) to each of the vertices.
- **Single-destination shortest-paths problem**
  - Find a shortest path to a given *destination* vertex $t$ from each vertex $v$.
- **Single-pair shortest-path problem**
  - Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.
- **All-pairs shortest-paths problem**
  - Find a shortest path from $u$ to $v$ for every pair of vertices $u$ and $v$

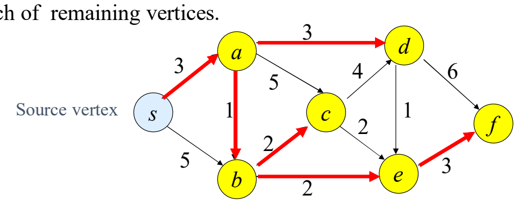
Comment:

- The problems are arranged in order from simple to complex
- Whenever there is an efficient algorithm for solving one of the three problems, the algorithm can also be used to solve the remaining two problems.



# Example

Graph $G = (V, E)$, source vertex $s \in V$, find the shortest path from $s$ to each of remaining vertices.

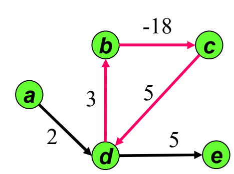



| | $s$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ |
|---|---|---|---|---|---|---|---|
| path | $s$ | $s,a$ | $s,a,b$ | $s,a,b,c$ | $s,a,d$ | $s,a,b,e$ | $s,a,b,e,f$ |
| weight | 0 | 3 | 4 | 6 | 6 | 6 | 9 |



# Negative Weights and Cycles?

- Negative edges are OK.
- *Negative weight cycles:* NO (otherwise paths with arbitrary small "lengths" would be possible)




Cycle: $(d \rightarrow b \rightarrow c \rightarrow d)$

Length = -10

Path from $a$ to $e$:

$P$: $a \rightarrow \sigma(d \rightarrow b \rightarrow c \rightarrow d) \rightarrow e$

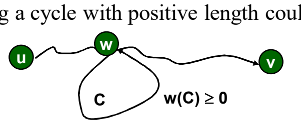
$w(P) = 7\text{-}10\sigma \rightarrow -\infty$, khi $\sigma \rightarrow +\infty$

**Assumption:**
Graph does not contain negative weight cycles

## Properties of shortest paths

- ***Property* 1.** Shortest-paths can have no cycles (= The shortest path can always be found among single paths). Path where vertices are distinct.

Proof: Removing a cycle with positive length could reduce the length of the path.




- ***Property 2.*** Any shortest-path in graph $G$ can not traverse through more than $n$ – 1 edges, where $n$ is the number of vertices
  - *Consequence of Property 1.*

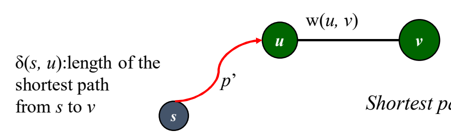
>= $n$ edges → not the simple path



## Properties of shortest paths

Denote: $\delta(u, v)$ = length of the shortest path from $u$ to $v$

Assume $P$ is the shortest path from $s$ to $v$, where $P = s \xrightarrow{p'} u \to v$.
Then $\delta(s, v) = \delta(s, u) + w(u, v)$.




*Shortest path from s to v:*

$P = s \xrightarrow{p'} u \to v$



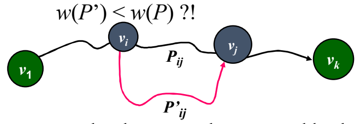
## Properties of shortest paths

**Property 3:** Assume $P = \langle v_1, v_2, \ldots, v_k \rangle$ is the shortest path from $v_1$ to $v_k$.
Then, $P_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ is the shortest path from $v_i$ to $v_j$, where $1 \le i \le j \le k$.

(In words: subpaths of shortest paths are also shortest paths)

**Proof (by contradiction).** If $P_{ij}$ is not the shortest path from $v_i$ to $v_j$, then one can find $P'_{ij}$ is the shortest path from $v_i$ to $v_j$ satisfying $w(P'_{ij}) < w(P_{ij})$. Then we get $P'$ is the path obtained from $P$ by substituting $P_{ij}$ by $P'_{ij}$, thus:

$w(P') < w(P)$ ?!




if some subpaths were not the shortest paths, one could substitute the shorter subpath and create a shorter total path

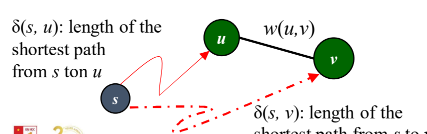


## Properties of shortest paths

Denote: $\delta(u, v)$ = length of the shortest path from $u$ to $v$

Assume $P$ is the shortest path from $s$ to $v$, where $P = s \xrightarrow{p'} u \to v$.
Then $\delta(s, v) = \delta(s, u) + w(u, v)$.

**Property 4:** Assume $s \in V$. For each edge $(u,v) \in E$, we have $\delta(s, v) \le \delta(s, u) + w(u,v)$.



## Shortest-Path Variants

- **Single-source shortest-paths problem**
  - Find a shortest path from a given source (vertex $s$) to each of the vertices.
- **Single-destination shortest-paths problem**
  - Find a shortest path to a given *destination* vertex $t$ from each vertex $v$.
- **Single-pair shortest-path problem**
  - Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.
- **All-pairs shortest-paths problem**
  - Find a shortest path from $u$ to $v$ for every pair of vertices $u$ and $v$

Comment:

- The problems are arranged in order from simple to complex
- Whenever there is an efficient algorithm for solving one of the three problems, the algorithm can also be used to solve the remaining two problems.



## Shortest path representation

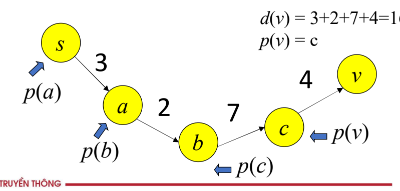
Shortest path algorithms works on 2 arrays:

- $d(v)$ = the length of shortest path from $s$ to $v$ that algorithm found so far
  (upper bound for the length of the shortest path from $s$ to $v$).
- $p(v)$ = a predecessor of $v$ in this shortest path
  (used to back trace the path from $s$ to $v$) .

$\delta(s,v) \le d(v)$

Initialization

for $v \in V(G)$
  do $d[v] \leftarrow \infty$
    $p[v] \leftarrow$ NULL
$d[s] \leftarrow 0$








## Single-source shortest paths

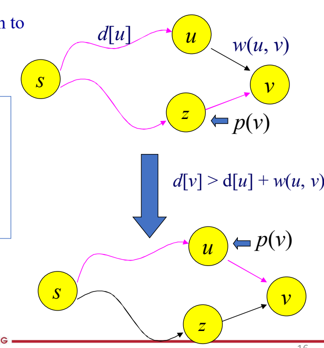
## Relaxation

Building the shortest path from $s$ to $v$:

Assume the current known shortest path from $s$ to $v$: $s \ldots z \rightarrow v$

Relaxing an edge $(u,v)$ means testing whether we can improve the shortest path to $v$ found so far by going through $u$

Relax($u$, $v$)
  if ($d[v] > d[u] + w(u, v)$)
  {   $d[v] \leftarrow d[u] + w(u, v)$
      $p[v] \leftarrow u$
  }



## Properties of Relaxation

Relax($u$, $v$)
  if ($d[v] > d[u] + w(u, v)$)
  {
    $d[v] \leftarrow d[u] + w(u, v)$
    $p[v] \leftarrow u$
  }

Shortest path algorithms differ in

- *how many times* they relax each edge, and
- *the order* in which they relax edges



## Single source shortest path

1. **Bellman-Ford algorithm**
2. Dijkstra algorithm



## Bellman-Ford algorithm

Richard Bellman
1920-1984

Lester R. Ford, Jr.
1927-2017



## Bellman-Ford algorithm

Bellman-Ford algorithm is used to find the shortest path from a vertex $s$ to each other vertex in the graph.

- **Input:** A directed graph $G=(V,E)$ and weight matrix $w[u,v] \in R$ where $u,v \in V$, source vertex $s \in V$; >= 0
  - *G does not contain* negative-weight cycle <0

- **Output:** Each $v \in V$

  $d[v] = \boxed{\delta(s, v)}$; Length of the shortest path from $s$ to $v$

  $p[v]$ - the predecessor of $v$ in this shortest path from $s$ *to* $v$.

# Bellman-Ford algorithm: Full version

```
Bellman-Ford(G, w, s)
// Step 1: Initialize shortest paths of with at most 0 edges
   Initialize-Single-Source(G, s)
/* Step 2: Calculate shortest paths with at most i edges from shortest
   paths with at most i-1 edges */
   for i in range (1, |V|)
     for each edge (u, v) ∈ E
         Relax(u, v)
   for each edge (u, v) ∈ E
     if d[v] > d[u] + w(u, v)
         return False    // there is a negative cycle
   return True
```

```
Relax(u, v)
 if d[v] > d[u] + w(u, v)
 {
    d[v] = d[u] + w[u,v] ;
    p[v] = u ;
 }
```

```
Initialize-Single-Source(G, s)
for v ∈ V\s
{
    d[v] = ∞;
    p[v] = Null;
}
p[s]=Null; d[s]=0;
```



# Bellman-Ford algorithm: Full version

```
Bellman-Ford(G, w, s)
// Step 1: Initialize shortest paths of with at most 0 edges
1.   Initialize-Single-Source(G, s)          → O(|V|)
/* Step 2: Calculate shortest paths with at most i edges from shortest
   paths with at most i-1 edges */
2.   for i in range (1, |V|)                 → O(|V||E|)
3.     for each edge (u, v) ∈ E
4.         Relax(u, v)
5.   for each edge (u, v) ∈ E                → O(|E|)
6.     if d[v] > d[u] + w(u, v)
7.         return False    // there is a negative cycle
8.   return True
```

Lines (2-4): First nested for-loop performs |V|-1 relaxation iterations; relax every edge at each iteration → Running time $O(|V||E|)$

- Running time $O(|V||E|)$
- Memmory space: $O(|V|^2)$



# Bellman-Ford algorithm

```
Bellman-Ford(G, w, s)
   Initialize-Single-Source(G, s)            → O(|V|)
   for i in range (1,|V|)                    → O(|V||E|)
     for each edge (u, v) ∈ E
        Relax(u, v)
   for each edge (u, v) ∈ E                  → O(|E|)
     if d[v] > d[u] + w(u, v)
        return False  // there is a negative cycle
   return True
```

if Ford-Bellman has not converged after |V| - 1 iterations, then there cannot be a shortest path tree, so there must be a negative weight cycle.

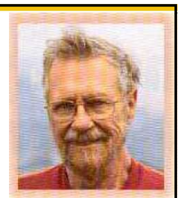


# Single source shortest path

1. Bellman-Ford algorithm
2. **Dijkstra algorithm**

## Dijkstra algorithm

- In case the weights on the edges are non-negative, the algorithm proposed by Dijkstra is more efficient than the Ford-Bellman algorithm.
- Algorithms are built by labeling vertices. The label of the vertices is initially temporary. At each iteration there is a temporary label that becomes a permanent label. If the label of a vertex $u$ becomes fixed, $d[u]$ gives us the length of the shortest path from the source $s$ to $u$. Algorithm ends when the labels of all vertices become fixed.



Edsger W.Dijkstra
(1930-2002)



## Dijkstra algorithm

- **Input:** A directed graph $G=(V,E)$ and weight matrix $w[u,v] \geq 0$ where $u,v \in V$, source vertex $s \in V$;
  - *G does not have* negative-weight cycle
- **Output:** Each $v \in V$

  $d[v] = \boxed{\delta(s, v);}$ Length of the shortest path from $s$ to $v$

  $p[v]$ - the predecessor of $v$ in this shortest path from $s$ *to* $v$.

Use greedy algorithm:

Maintain a set S of vertices for which we know the shortest path

At each iteration:

- grow S by one vertex , choosing shortest path through S to any other vertex not in S
- If the cost from S to any other vertex has decreased, update it



## Dijkstra algorithm

```
Dijkstra ( )
{
     for   v ∈ V    // Initialize
    {
         d[v] = w[s,v] ;
         p[v] = s;
    }
    d[s] = 0;   S = {s};   // S: the set of vertices with fixed label (shortest path from s to it has been found)
    T = V \ {s};          // T: the set of vertices with temporary label
    while  (T  ≠ ∅)            //Loop
     {
          Find vertex  u ∈ T satisfying  d[u] = min{ d[z] :  z  ∈  T};
          T =  T \ {u};  S= S  ∪ {u};      //Fixed label of vertex u
          for v ∈ adj[u] and v ∈ T //Assign new label to each vertex v of T if necessary (if value d[v] is decreased)
               if   (d[v] > d[u] + w[u,v])
               {
                    d[v] = d[u] + w[u,v] ;
                    p[v] = u ;
               }
     }
}
```



## Dijkstra algorithm

```
void Dijkstra ( )
{
     for   v ∈ V    // Initialize
    {
         d[v] = w[s,v] ;
         p[v]=s;
    }
    d[s] = 0;   S = {s};
    T = V \ {s};
    while  (T  ≠ ∅)                    //Loop
     {
          Find vertex  u ∈ T satisfying  d[u] = min{ d[z] :  z  ∈  T};
          T =  T \ {u};  S= S  ∪ {u};
          for   v ∈ adj[u] and v ∈ T
               if   (d[v] > d[u] + w[u,v])
               {
                    d[v] = d[u] + w[u,v] ;
                    p[v] = u ;
               }
     }
}
```

- $O(|V|^2)$ operations
  - (|V|-1) iterations: 1 for each vertex u added to the distinguished set S.
  - (|V|-1) iterations: for each adjacent vertex of the one added to the distinguished set.

## Shortest path problems

1. Bellman-Ford algorithm

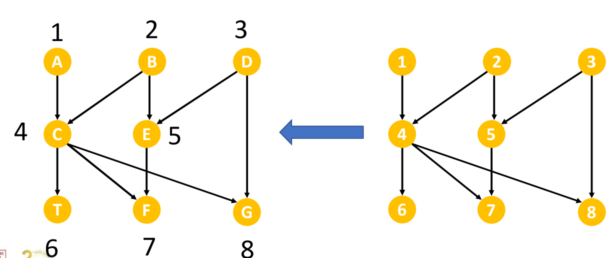
2. Dijkstra algorithm

3. Shortest path in the directed graph with no cycles
   (Directed acyclic graph (DAG))



## Single-Source Shortest Paths in DAGs

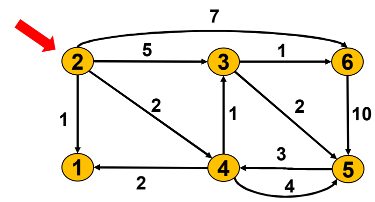
A **topological sort** or **topological ordering** of a DAG is a linear ordering of its vertices such that for every directed edge (*u, v*) from vertex *u* to vertex v, *u* comes before *v* in the ordering. (In orther words: its vertexes can be numbered so that each directed edge starting from the vertex with the smaller index to the vertex with a larger index)






## Topological sorting algorithm

- We see that: *In the DAG, there always exists a vertex with in-dgree = 0*

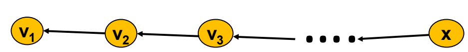





## Topological sorting algorithm

- We see that: *In the DAG, there always exists a vertex with in-dgree = 0*

Indeed, starting at vertex $v_1$ if there is an incoming edge to it from vertex $v_2$ then we move to $v_2$. If there is an edge from $v_3$ to $v_2$, then we switch to $v_3$, ... Since there is not any cycle in the graph, so after a finite number of such transfers we have to go to the vertex without incoming edge.




- Topological sorting algorithm**:**

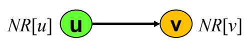
First, finding all vertices with in-dgree = 0. We index these vertices starting from 1.

Next, removing from graphs vertices that have just been indexed together with the edge going out of them, we get a new graph also without cycle, and we again starting index vertices on this new graph.

The process is repeated until all vertices of the graph has been indexed.

## Topological sorting algorithm

- **Input:** DAG $G=(V,E)$ *with the adjacent list* $Adj(v)$, $v \in V$.
- **Ouput:** *For each* $v \in V$ *the index* $NR[v]$ *satisfying*: Each directed edge $(u, v)$: $NR[u] < NR[v]$.






## Single-Source Shortest Paths in DAGs

Shortest paths are always *well-defined* in *DAGS*

- no cycles => no negative-weight cycles even if there are negative-weight edges

*In a DAG:*

- Every path is a subsequence of the topologically sorted vertex order
- If we do topological sort and process vertices in that order
- We will process each path in forward order
  - Never relax edges out of a vertex until have processed all edges into the vertex

Thus, just 1 iteration is sufficient

```
DAG-SHORTEST PATHS(G, s)
   TOPOLOGICALLY-SORT the vertices of G
   INIT(G, s)
   for each vertex u taken in topologically sorted order do
      for each v ∈ Adj[u] do
          RELAX(u, v)
```



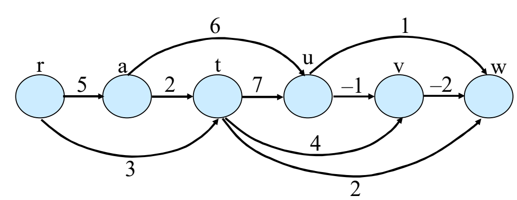
## Single-Source Shortest Paths in DAGs

```
Critical_Path ( )
{
     d[v[1]] = 0;
     for j in range (2, n+1) d[v[j]] =∞;
     for v[j] ∈ Adj[v[1]]
             d[v[j]] := w(v[1], v[j]) ;

     for  j in range (2, n+1)
       for  v ∈ Adj[v[j]]
             Relax(v[j],v)

}
```

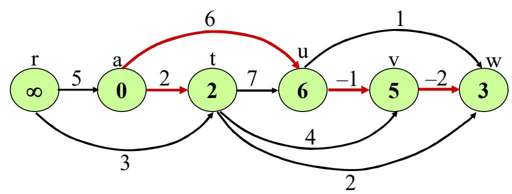


## Example

Find the shortest path from ***a*** to each other vertices of the DAG graph where vertices are already topological order



## Example




Result: The shortest path tree from *a* represented by the red edges



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

## All pairs shortest-paths

## Shortest-Path Variants

- **Single-source shortest-paths problem**
  - Find a shortest path from a given source (vertex *s*) to each of the vertices.
- **Single-destination shortest-paths problem**
  - Find a shortest path to a given *destination* vertex *t* from each vertex *v*.
- **Single-pair shortest-path problem**
  - Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.
- **All-pairs shortest-paths problem**
  - Find a shortest path from *u* to *v* for every pair of vertices *u* and *v*

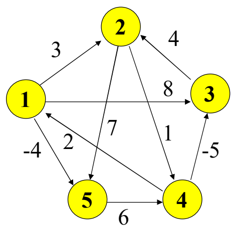


## All pairs shortest-paths

**Problem** Given directed graph $G = (V, E)$, with weight on each edge $e$ is $w(e)$,
for each pair of vertices $u$, $v$ of $V$, find the shortest path from $u$ to $v$.

- Input: *weight matrix*
- Output *matrix*: element at row $u$ column $v$ is the length of the shortest path from $u$ to $v$.
- Allow negative-weight edge
- **Assumption: None negative-length cycle**

# Example




**Input**

Weight matrix $W_{nxn} = (w)_{ij}$ where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(i,j) & \text{if } i \neq j \ \& \ (i,j) \in E \\ \infty & \text{otherwise} \end{cases}$$

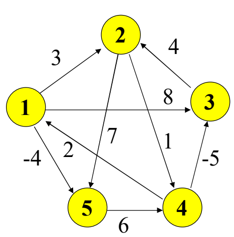
$$W_{5x5} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



# Example

**Output**

*Matrix*: element at row $u$ column $v$ is the length of the shortest path from $u$ to $v$.




Shortest path from 1 to 2 : 1- 5 - 4 - 3 - 2

$= -4 + 6 - 5 + 4$

$$\begin{pmatrix} 0 & \mathbf{1} & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ \mathbf{8} & 5 & 1 & 6 & 0 \end{pmatrix}$$

Shortest path from 5 to 1 : 5 - 4 - 1



# Floyd-Warshall algorithm

$d_{ij}^{(m)}$ = length of the shortest path from $i$ to $j$ using intermediate vertices in the vertex set $\{1, 2, \ldots, m\}$.

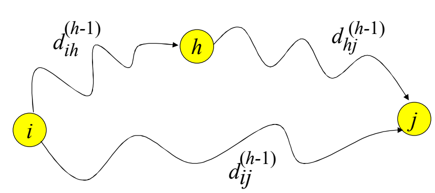
$i \rightarrow \leq m \rightarrow \leq m \rightarrow \ldots \rightarrow \leq m \rightarrow j$

Graph with $n$ vertices $\{1,2,..,n\}$ → length of the shortest path from $i$ to $j$ is $d_{ij}^{(n)}$



# Recursive formula computed $d^{(h)}$

- $d_{ij}^{(0)} = w_{ij}$ $\quad i \rightarrow j$
- $d_{ij}^{(h)} = \min\left(d_{ij}^{(h-1)}, d_{ih}^{(h-1)} + d_{hj}^{(h-1)}\right)$ if $h \geq 1$



## Floyd-Warshall

void Floyd-Warshall($n$, $W$)
{
$D^{(0)} \leftarrow W$
$\boxed{\text{for } (k=1;\ k<=n;\ k{+}{+})}$ Path going through only intermediate vertices selected from $\{1,2,..,k\}$

$\boxed{\begin{array}{l}\text{for } (i=1;i<=n;i{+}{+})\\ \quad \text{for } (j=1;j<=n;j{+}{+})\end{array}}$ All pairs $(i, j)$

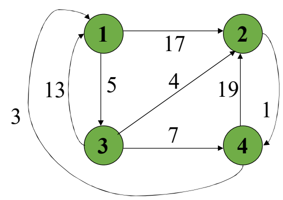
$d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

return $D^{(n)}$;
}

Running time $\Theta(n^3)$ !



## Example 1: Find shortest path between every pairs of vertices




**Input**

Weight matrix $W_{nxn} = (w)_{ij}$ where

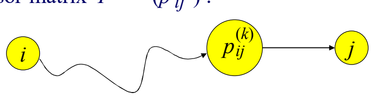
$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(i,j) & \text{if } i \neq j \ \&\ (i,j) \in E \\ \infty & \text{otherwise} \end{cases}$$

$$W_{4x4} = \begin{pmatrix} 0 & 17 & 5 & \infty \\ \infty & 0 & \infty & 1 \\ 13 & 4 & 0 & 7 \\ 3 & 19 & \infty & 0 \end{pmatrix}$$



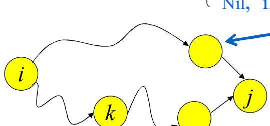
## Build the shortest path

Predecessor matrix $P^{(k)} = (p_{ij}^{(k)})$ :




Shortest path from $i$ to $j$ going through intermediate vertices only selected from $\{1, 2, \ldots, k\}$.

$$p_{ij}^{(0)} = \begin{cases} i, & \text{if } (i,j) \in E \\ \text{Nil}, & \text{if } (i,j) \notin E \end{cases}$$




$$p_{ij}^{(k)} = \begin{cases} p_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ p_{kj}^{(k-1)} & \text{otherwise} \end{cases}$$