Algorithms & Data Structures I
CSC 225

Ali Mashreghi

Fall 2018

Department of Computer Science, University of Victoria

All pairs shortest path

- Now let's take a look at the **transitive closure** and **all-pairs shortest paths (APSP)** problems.

- We will describe a dynamic programming approach that solve APSP in $\boldsymbol{\Theta(n^3)}$ time and works **even on weighted graphs**.

- So, it's efficiency is the best among the four approaches that we discussed.

All pairs shortest path

- The algorithm is named **Floyd-Warshall** and was invented in 1962:

Robert Floyd

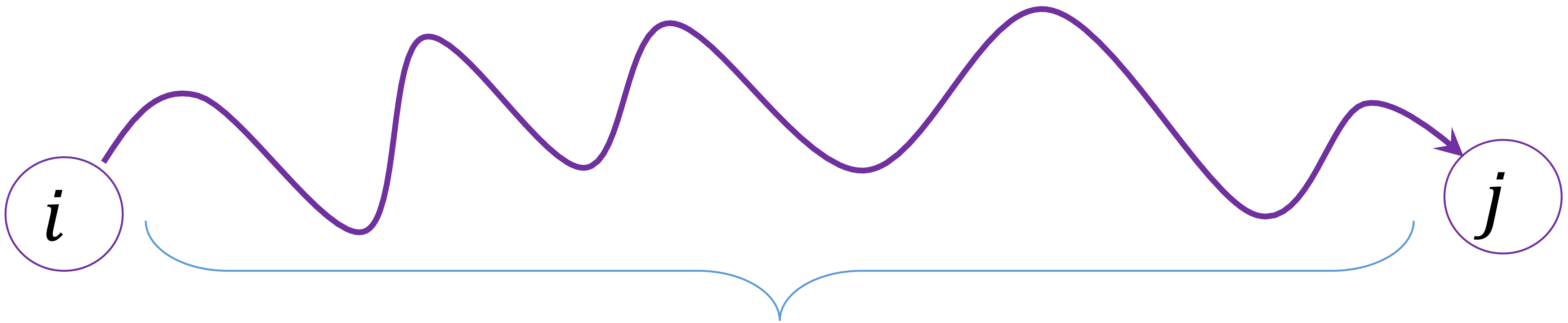
Stephen Warshall

All pairs shortest path

- Let's assume that the nodes are numbered from 1 to n.

All pairs shortest path

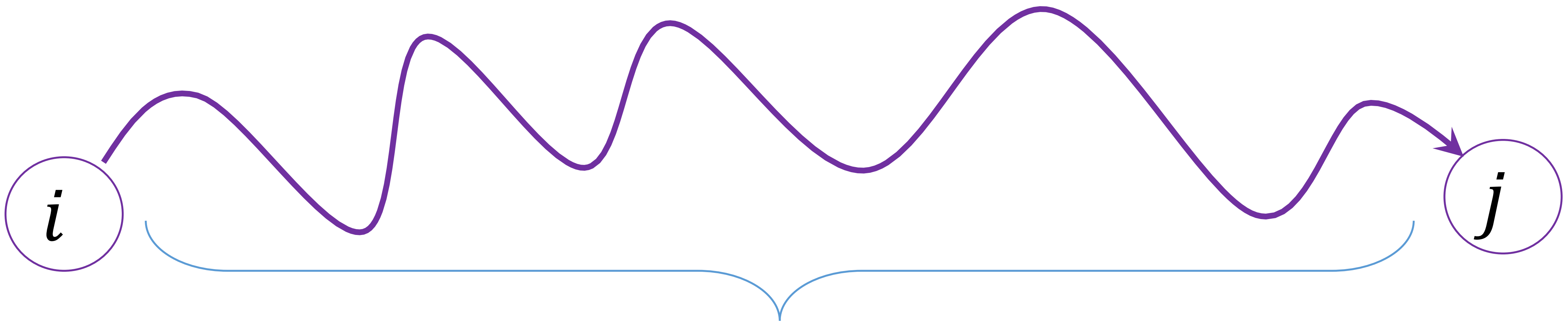
- Also, assume that p is a shortest path from i to j such that all intermediate vertices in p belong to the set $\{1,2, \dots, k\}$



p: All intermediate vertices in $\{1, \dots, k\}$

All pairs shortest path

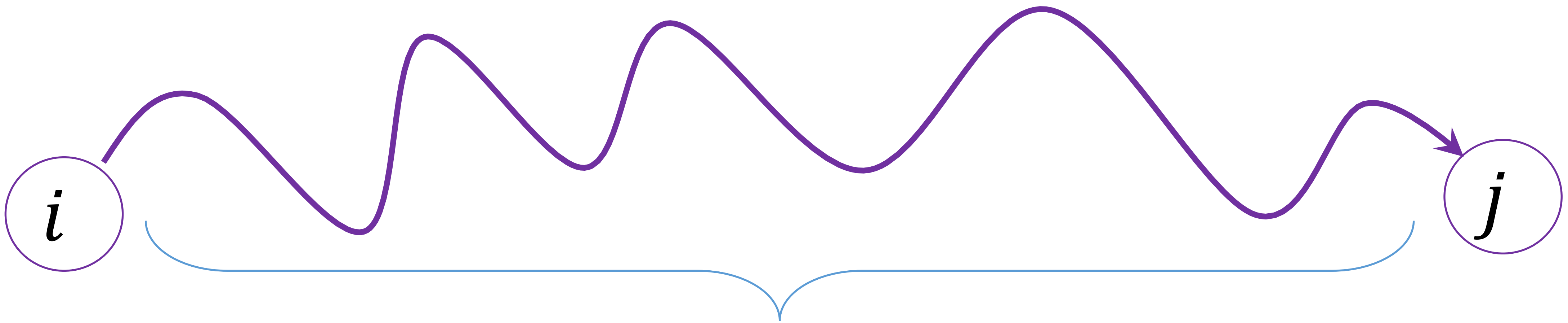
- **Note 1:** i and j are not considered intermediate vertices themselves



p: All intermediate vertices in $\{1, \dots, k\}$

All pairs shortest path

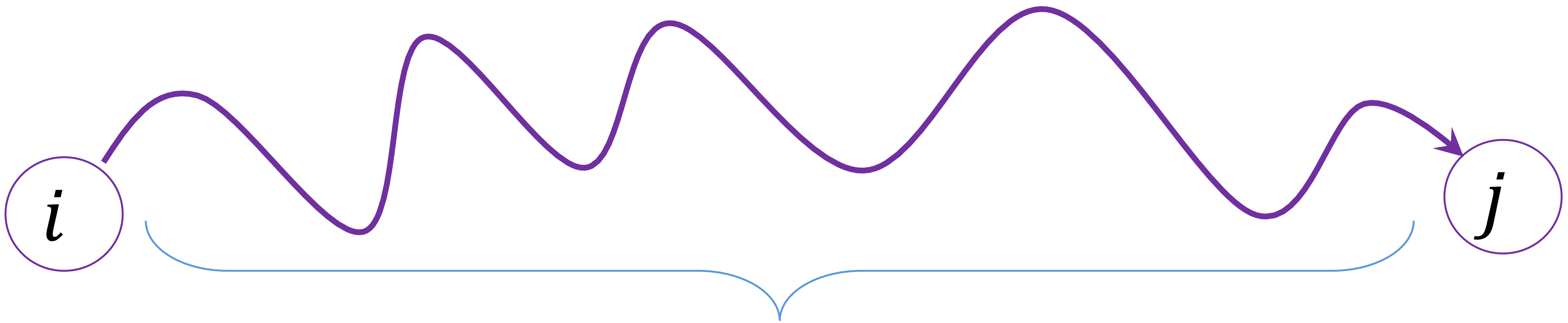
- **Note 2:** There might be no intermediate vertex on the shortest path at all. What we mean here is that if there is any, that vertex is in $\{1, \dots, k\}$



p: All intermediate vertices in $\{1, \dots, k\}$

All pairs shortest path

- Let's denote the **length of such path** with $\boldsymbol{d_{ij}^{(k)}}$ from now on.



p: All intermediate vertices in $\{1, \ldots, k\}$

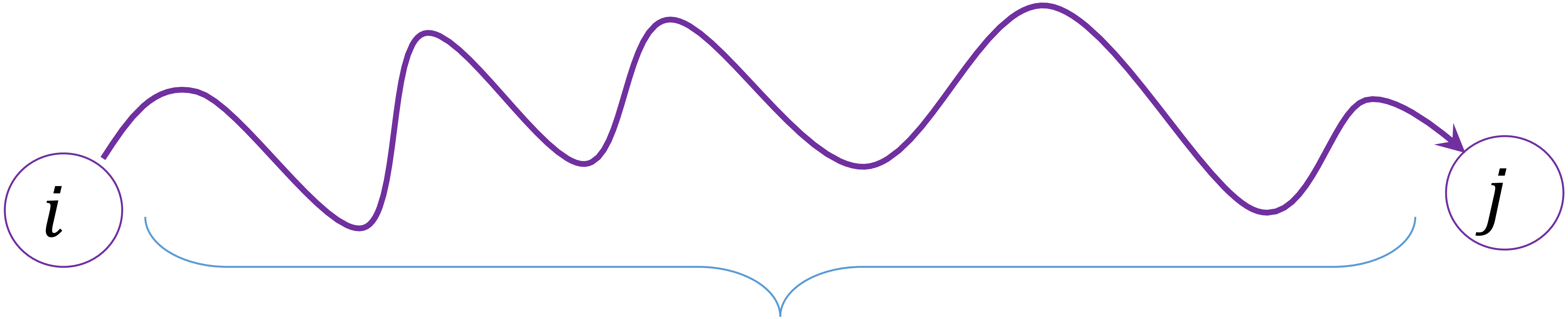
All pairs shortest path

- **Observation:** $\boldsymbol{d_{ij}^{(n)}}$ is the length of the **actual** shortest path from i to j, since we know that all intermediate vertices must be in the actual shortest path are in $\{1, 2, \dots, n\}$

- So, our goal is to compute $d_{ij}^{(n)}$ for all pairs (i, j) of vertices!

All pairs shortest path

- **Question:** How can we compute $d_{ij}^{(k)}$ recursively? Or, how can we express $d_{ij}^{(k)}$ in terms of smaller k's?



p: All intermediate vertices in $\{1, \ldots, k\}$

All pairs shortest path

- **Answer:** There are two cases:

- **Case 1:** k is not an intermediate vertex.
- In this case $d_{ij}^{(k)} = d_{ij}^{(k-1)}$ since we know that all intermediate vertices must be in $\{1,2, \ldots, k-1\}$

All pairs shortest path

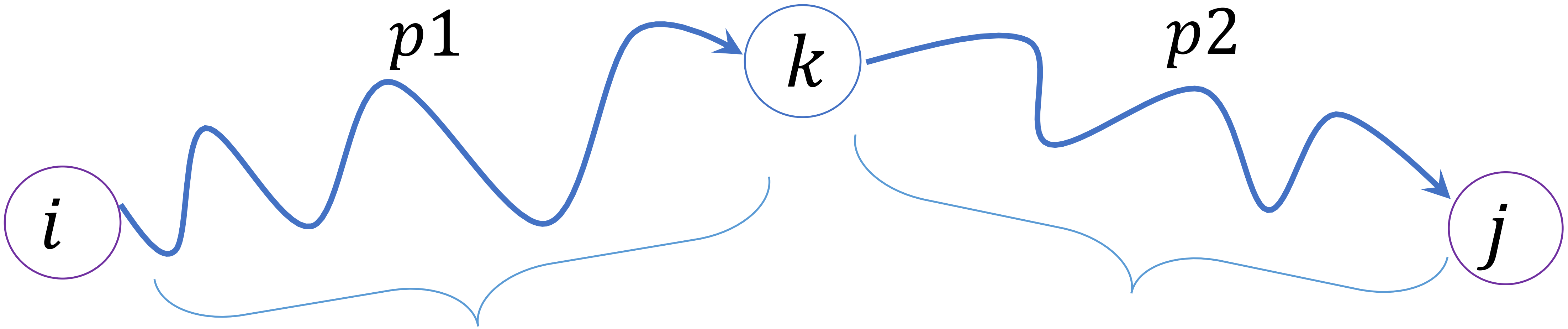
- **Case 2:** k is an intermediate vertex. In this case we can break the path i to j to two subpaths from i to k and from k to j.

- Since k is not a part of these subpaths, the intermediate vertices in both of these subpaths belong to $\{1,2, \ldots, k-1\}$.

- As a result, $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

All pairs shortest path

- $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.



all intermediate vertices are in $\{1,2,\ldots,k-1\}$

all intermediate vertices are in $\{1,2,\ldots,k-1\}$

All pairs shortest path

- Now, we can describe $d_{ij}^{(k)}$ recursively, as follows:
- $d_{ij}^{(k)} = \begin{cases} w(i,j) & k = 0 \\ \min(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & k > 0 \end{cases}$
- Initially, when $k = 0$ the shortest distance between i and j is the weight of the edge (i, j).

All pairs shortest path

FLOYD-WARSHALL(W)

1. $D = W$
2. **for** $k = 1$ **to** n
3.   **for** $i = 1$ **to** n
4.   **for** $j = 1$ **to** n
5. $D[i][j] = \min(D[i][j], \mathrm{D}[i][k] + \mathrm{D}[k][j])$
6. **return** D

All pairs shortest path

FLOYD-WARSHALL(W)

1. $D = W$
2. **for** $k = 1$ **to** n
3. **for** $i = 1$ **to** n
4. **for** $j = 1$ **to** n
5. $D[i][j] = \min(\underbrace{D[i][j]}_{d_{ij}^{(k-1)}}, \underbrace{D[i][k]}_{d_{ik}^{(k-1)}} + \underbrace{D[k][j]}_{d_{kj}^{(k-1)}})$
6. **return** D

The for loop for k has to be first, so when we want to update $D[i][j]$ (i.e. $d_{ij}^{(k)}$) all necessary entries are already computed.

All pairs shortest path

- Before the loop for k we have finished the loop for $k-1$. So, the values we use on line 5, i.e. $D[i][j], D[i][k], \text{and } D[k][j]$ correspond to $d_{ij}^{(k-1)}$, $d_{ik}^{(k-1)}$, $d_{kj}^{(k-1)}$, respectively.

- Even if these values are updated on the current for loop for k, we know that $d_{ij}^{(k)} \leq d_{ij}^{(k-1)}$; so, it won't create a problem and the algorithm is still correct.

Printing the paths

```
FLOYD-WARSHALL(W)
D = W
P // path matrix initialized with NULL
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            if D[i][j] > D[i][k] + D[k][j]
                D[i][j] = D[i][k] + D[k][j]
                P[i][j] = k
return D
```

We keep the value of k that caused the update

PRINT-PATH(i, j)

1. **print** i
2. PRINT-INTERMEDIATE-VERTICES(i, j)
3. **print** j

PRINT-INTERMEDIATE-VERTICES(i, j)

1. **if** P[i][j] == NULL
2. **return**
3. k = P[i][j]
4. PRINT-INTERMEDIATE-VERTICES(i, k)
5. **Print** k
6. PRINT-INTERMEDIATE-VERTICES(k, j)