Shortest Paths Revisited 3/4
All-Pairs Shortest Paths

Lecture 07.08 by *Marina Barsky*

The Floyd-Warshall algorithm

All-Pairs Shortest Paths Problem

Input: directed graph G=(V,E) with edge costs C [no special source vertex].

Output: if G has no negative cycles, the length of a shortest path **between each pair of vertices** u,v ∈ V.

All-pairs shortest paths: possible solutions

Use single-source shortest path algorithm:

Repeat n times (once for each vertex as a source)

1. If the costs are non-negative

n*Dijkstra (m log n) = O(nm log n) = $\begin{cases} O(n^2 \log n) & \text{if } m=O(n) \text{ [sparse]} \\ O(n^3 \log n) & \text{if } m=O(n^2) \text{ [dense]} \end{cases}$

2. If allowing negative costs:

n*Bellman-Ford (nm) = O(n^2m) = $\begin{cases} O(n^3) & \text{if } m=O(n) \text{ [sparse]} \\ O(n^4) & \text{if } m=O(n^2) \text{ [dense]} \end{cases}$

We will develop a special Dynamic Programming algorithm:

Floyd-Warshall: always $O(n^3)$

All-Pairs Shortest Paths

Floyd-Warshall Algorithm

Dynamic Programming!

Order of subproblems

Again – there is no “natural” ordering of subproblems: which subproblem is smaller than the other?

Idea: we invent our own order of subproblems:

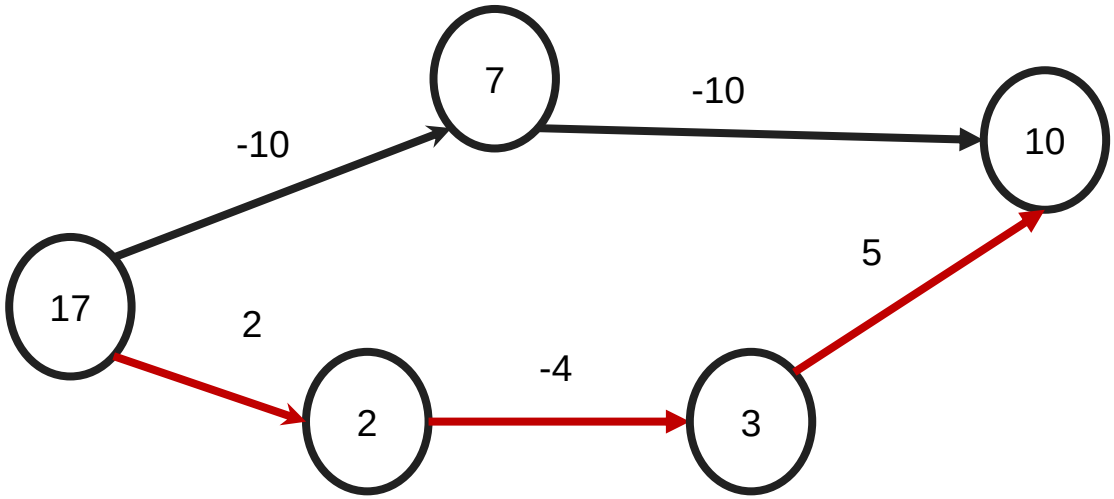
- We impose arbitrary ordering on vertices v_1, v_2, … v_n
- Each vertex gets a numeric id: V = {1,2,…,n}
- Now we have a sequence {1,2,…,n} of vertices

- **Similar to knapsack problem**, in each iteration k we will compute all shortest paths using only a subset of vertices {1,2,…k} as intermediate nodes on each shortest path

Subproblem

- V={1,2,…,n}
- We are allowed to use only {1,…,k}
- Each subproblem P(i, j, k) represents the cost of the shortest path from i to j using only the first 1…k vertices in the sequence

Example

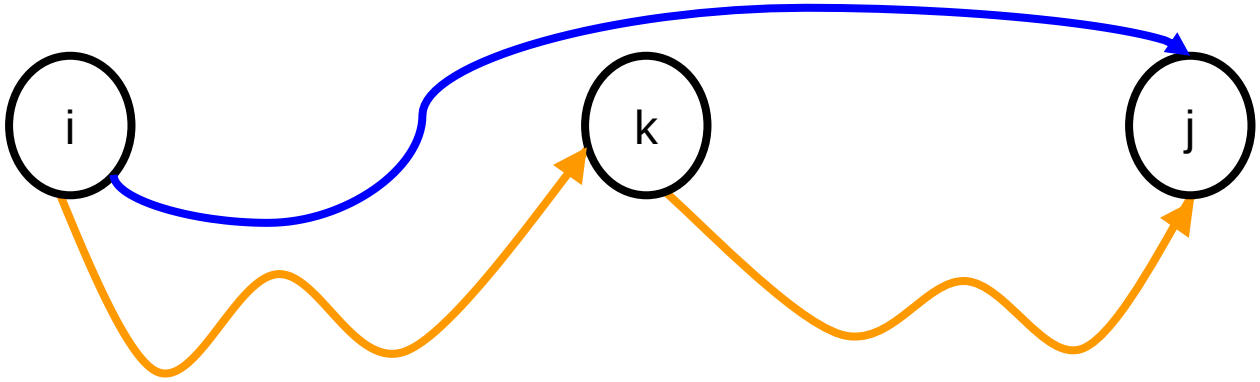


i=17, j= 10, k= 5
P(i, j, k) = 3

Optimal subproblems: intuition

When we allow the next vertex k to be included as intermediate vertex on the path i~>j, we have the following choices:

- Do not include new vertex k as part of the shortest path from i to j.
  The cost of the shortest path i~>j remains P(i, j, k-1)
- If vertex k can be used to improve P(i, j, k-1), then k is internal to path P(i,j,k).
  In this case both P(i, k, k-1) and P(k, j, k-1) are shortest paths which use first k-1 vertices [which we already computed as subproblems for k-1]



We choose min between P(i, j, k-1) and [P(i, k, k-1) + P(k, j, k-1)]
All these min-cost paths are already computed in iteration k-1

Recurrence relation

- Input: directed graph G={V,E} – where vertices are numbered: V={1, …n}, and the cost matrix C with all edge costs.
- For each pair (i, j) ∈ V, let P(i, j, k) be the cost of the shortest path i~>j which uses only k first vertices from V as intermediate nodes on the path.

- Base case: no intermediate vertices are allowed (a single-edge path)

$$P(i,j,0) = \begin{cases} 0 \text{ if } i=j \\ C_{ij} \text{ if edge}(i,j) \in E \\ \infty \text{ otherwise} \end{cases}$$

Recurrence relation

- Input: directed graph G={V,E} – where vertices are numbered: V={1, …n}, and the cost matrix C with all edge costs.
- For each pair (i,j) ∈ V, let P(i, j, k) be the cost of the shortest path i~>j which uses only k first vertices from V as intermediate nodes on the path.

- Base case: no intermediate vertices are allowed

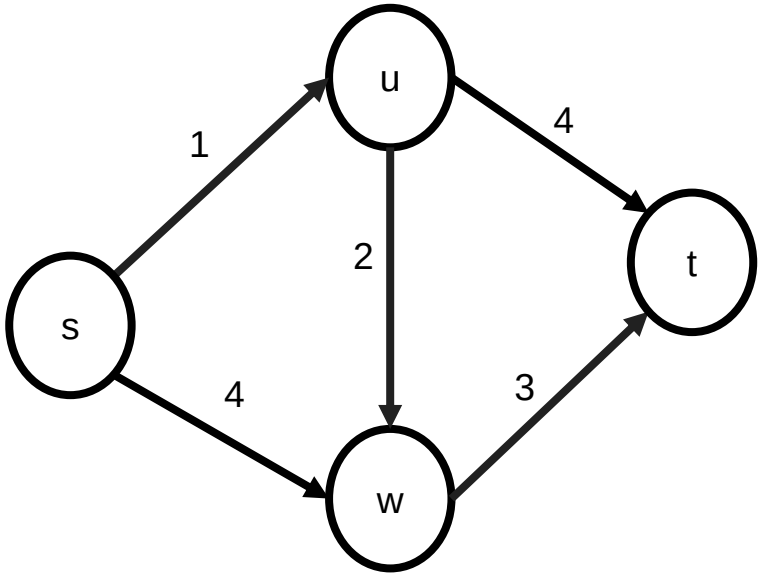
$$P(i,j,0) = \begin{cases} 0 \text{ if } i=j \\ C_{ij} \text{ if edge}(i,j) \in E \\ \infty \text{ otherwise} \end{cases}$$

- Recurrence: for any k, $0 < k \leq n$

$$P(i, j, k) = \min \begin{cases} P(i, j, k-1) \\ P(i, k, k-1) + P(k, j, k-1) \end{cases}$$

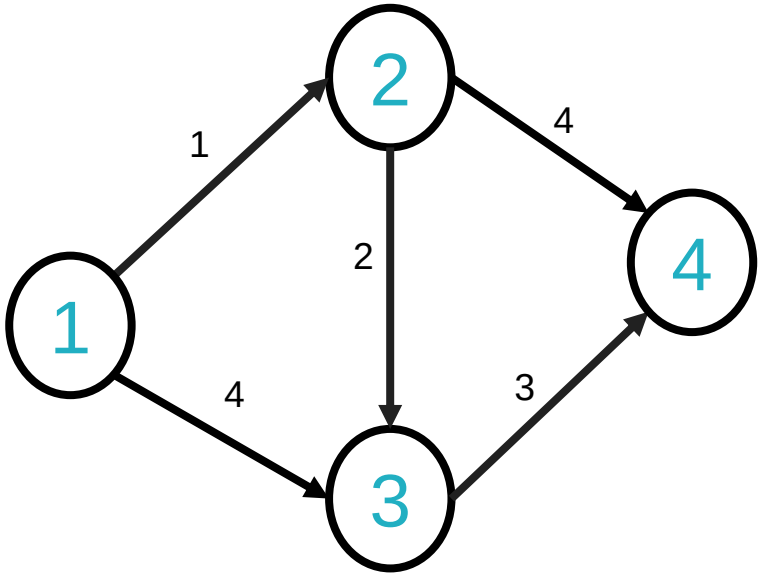
Floyd-Warshall algorithm: very tiny illustration

Because the DP table is 3D



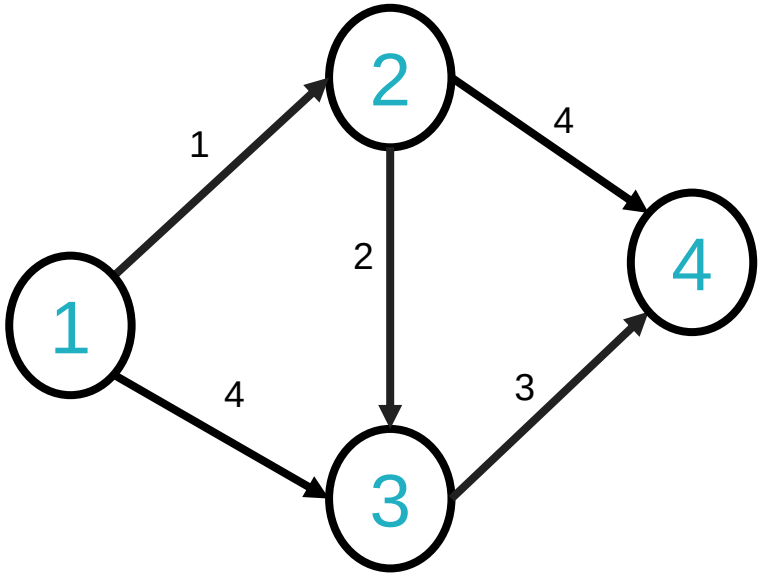
- First step: relabel vertices with numeric IDs

Floyd-Warshall algorithm: illustration



- Possible intermediate vertices: [1, 2, 3, 4]

Floyd-Warshall algorithm: illustration



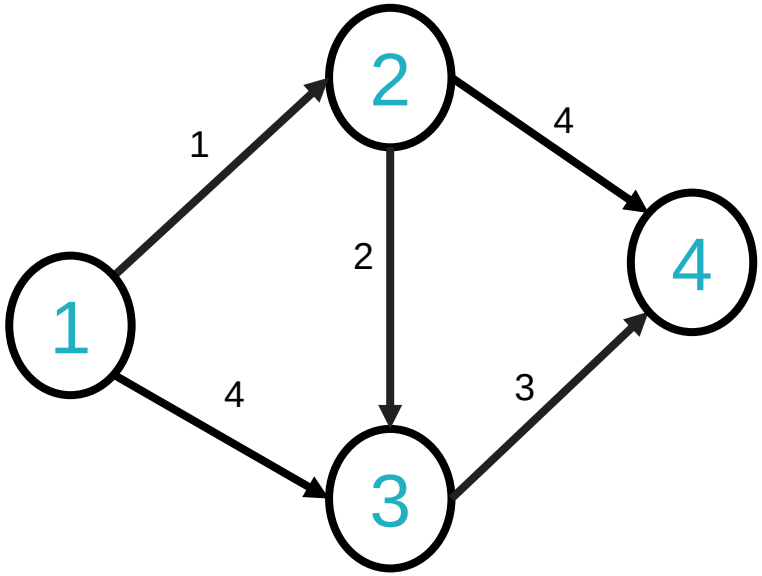
to

		j			
		1	2	3	4
from i	1	0	1	4	∞
	2	∞	0	2	4
	3	∞	∞	0	3
	4	∞	∞	∞	0

k = 0

- Possible intermediate vertices: [1, 2, 3, 4]
- Base case: none of these are allowed as intermediate vertices = no intermediate vertices = single-edge paths

Floyd-Warshall algorithm: illustration



- Possible intermediate vertices: [1, 2, 3, 4]
- Only vertex 1 is allowed as intermediate vertex

to

from i \ j	1	2	3	4
1	0	1	4	∞
2	∞	0	2	4
3	∞	∞	0	3
4	∞	∞	∞	0

k = 0

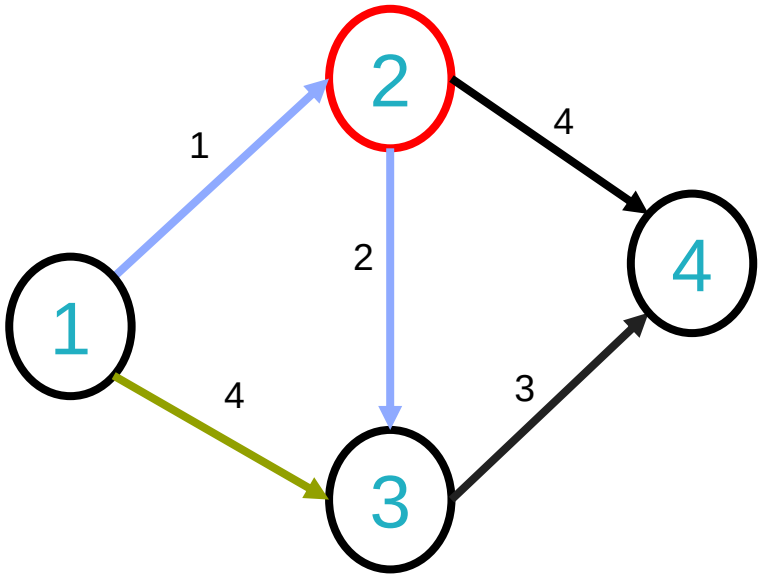
to

from i \ j	1	2	3	4
1	0	1	4	∞
2	∞	0	2	4
3	∞	∞	0	3
4	∞	∞	∞	0

k = 1

P(1,2,1) = min[P(1,2,0), P(1,1,0) + P(2,2,0)]

Floyd-Warshall algorithm: illustration



to

from i	j	1	2	3	4
	1	0	1	4	∞
	2	∞	0	2	4
	3	∞	∞	0	3
	4	∞	∞	∞	0

k = 1

- Possible intermediate vertices: [1, 2, 3, 4]
- Now both 1 and 2 are allowed as intermediate vertices
- Additional intermediate vertex is 2

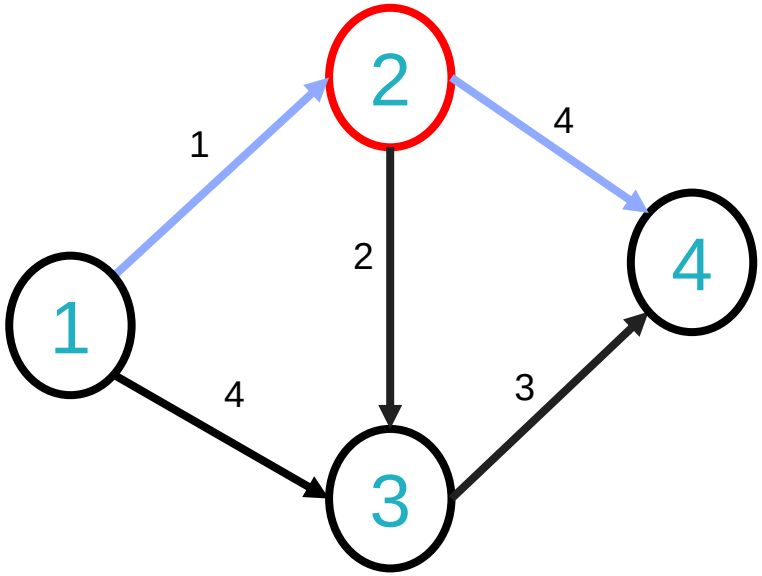
to

from i	j	1	2	3	4
	1	0	1	3	5
	2	∞	0	2	4
	3	∞	∞	0	3
	4	∞	∞	∞	0

k = 2

P(1,3,2) = min[P(1,3,1), P(1,2,1) + P(2,3,1)]

Floyd-Warshall algorithm: illustration



to

from i	j	1	2	3	4
1	0	1	4	∞
2	∞	0	2	4
3	∞	∞	0	3
4	∞	∞	∞	0

k = 1

- Possible intermediate vertices: [1, 2, 3, 4]
- Now both 1 and 2 are allowed as intermediate vertices
- Additional intermediate vertex is 2

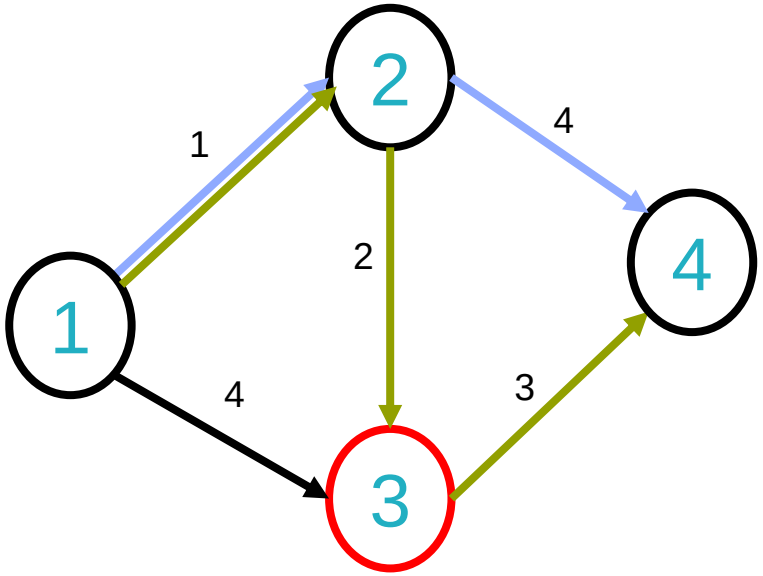
to

from i	j	1	2	3	4
1	0	1	3	5
2	∞	0	2	4
3	∞	∞	0	3
4	∞	∞	∞	0

k = 2

P(1,4,2) = min[P(1,4,1), P(1,2,1) + P(2,4,1)]

Floyd-Warshall algorithm: illustration



to

from i \ j	1	2	3	4
1	0	1	3	5
2	∞	0	2	4
3	∞	∞	0	3
4	∞	∞	∞	0

k = 2

- Possible intermediate vertices: [1, 2, 3, 4]
- Now both 1, 2, 3 are allowed as intermediate vertices
- Additional intermediate vertex is 3

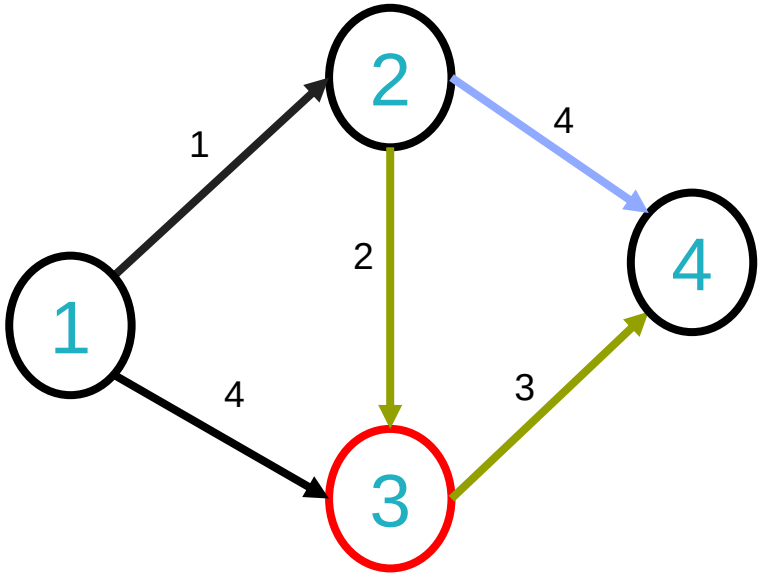
to

from i \ j	1	2	3	4
1	0	1	3	5
2	∞	0	2	4
3	∞	∞	0	3
4	∞	∞	∞	0

k = 3

P(1,4,3) = min[P(1,4,2), P(1,3,2) + P(3,4,2)]

Floyd-Warshall algorithm: illustration



to

from i \ j	1	2	3	4
1	0	1	3	5
2	∞	0	2	4
3	∞	∞	0	3
4	∞	∞	∞	0

k = 2

- Possible intermediate vertices: [1, 2, 3, 4]
- Now 1, 2, 3 are allowed as intermediate vertices
- Additional intermediate vertex is 3

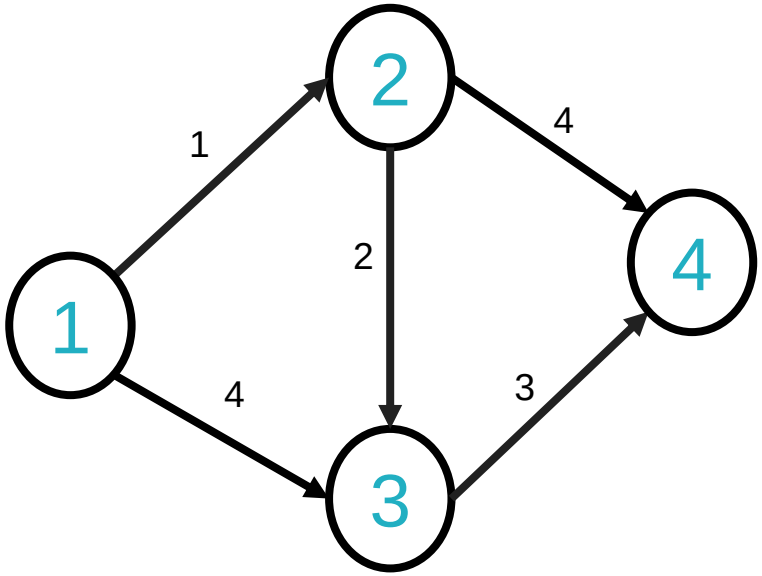
to

from i \ j	1	2	3	4
1	0	1	3	5
2	∞	0	2	4
3	∞	∞	0	3
4	∞	∞	∞	0

k = 3

P(2,4,3) = min[P(2,4,2), P(2,3,2) + P(3,4,2)]

Floyd-Warshall algorithm: illustration



to

		j			
		1	2	3	4
from i	1	0	1	3	5
	2	∞	0	2	4
	3	∞	∞	0	3
	4	∞	∞	∞	0

- Possible intermediate vertices: [1, 2, 3, 4]
- Now all are allowed as intermediate vertices
- Additional intermediate vertex is 4
- In this example 4 is not an intermediate vertex on any path
- We can stop

These are final shortest paths from any node to any other node in the graph

This 2D table is returned by an algorithm after the last iteration

Pseudocode

Algorithm FloydWarshall (digraph G=(V, E), edge costs C)

```
A: = nxnxn 3D array indexed by k, i, and j
# base case
for each i ∈ V:
        for each j ∈ V:
                if i=j                  A[0, i, i] := 0
                else if (i, j) ∈ E      A[0, i, j] := C_ij
                else                    A[0, i, j] := ∞

# DP table
for k from 1 to n:
        for i from 1 to n:
                for j from 1 to n:
                        A[k, i, j] = min (A[k-1, i, j], A[k-1, i, k] + A[k-1, k, j])
return A[n]             # last 2D matrix contains all-pair shortest path costs
```

Total n^3 subproblems with O(1) work per subproblem

Running time $O(n^3)$

Floyd-Warshall algorithm: notes

- **Negative cycles:**
  - To trust the results – we need to check that graph does not have negative cycles
  - If we scan the diagonal of the final matrix A[n], then all values A[n, i, i] must be 0.
  - If any of distances from node i to itself is < 0 – graph contains negative cycles

- **Space improvement:**
  - We do not have to store the entire 3D array to recover actual shortest path between a pair of vertices
  - It is enough for each pair of vertices (i, j) to store the max index of an internal node on the path from i to j: the last value of k which was used to improve the cost of i~>j
  - Knowing this vertex, we can recursively obtain shortest paths i~>k and k~>j and recover the entire path

- **Undirected graphs:**
  - The Floyd-Warshall algorithm also works for undirected graphs, but only when there are no negative-weight edges

Results: All-Pairs Shortest Paths

For sparse graphs **with non-negative edges**: use n*Dijkstra

The best!

1. **Graphs with non-negative edge costs**:

n*Dijkstra (m log n) = O(nm log n) =

- $O(n^2 \log n)$ if m=O(n) [sparse]
- $O(n^3 \log n)$ if m =$O(n^2)$ [dense]

2. **General graphs:**

n*Bellman-Ford (nm) = $O(n^2m)$ =

- $O(n^3)$ if m=O(n) [sparse]
- $O(n^4)$ if m=$O(n^2)$ [dense]

1*Floyd-Warshall: $O(n^3)$

Can we do better for generic graphs?