Design and Analysis of Algorithms

(CS345/CS345A)

Lecture 19

Dynamic Programming – (Final lecture)

- **Bellman-Ford Algorithm (A new perspective)**
- **All-Pairs Shortest Paths**

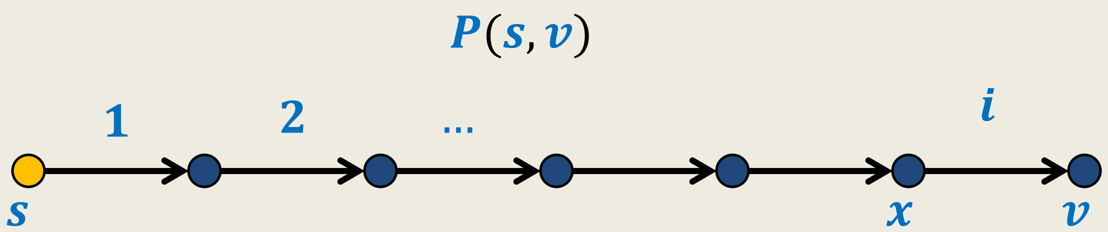
BELLMAN-FORD ALGORITHM

For shortest paths in a graph with

Negative weights

BUT No negative cycle

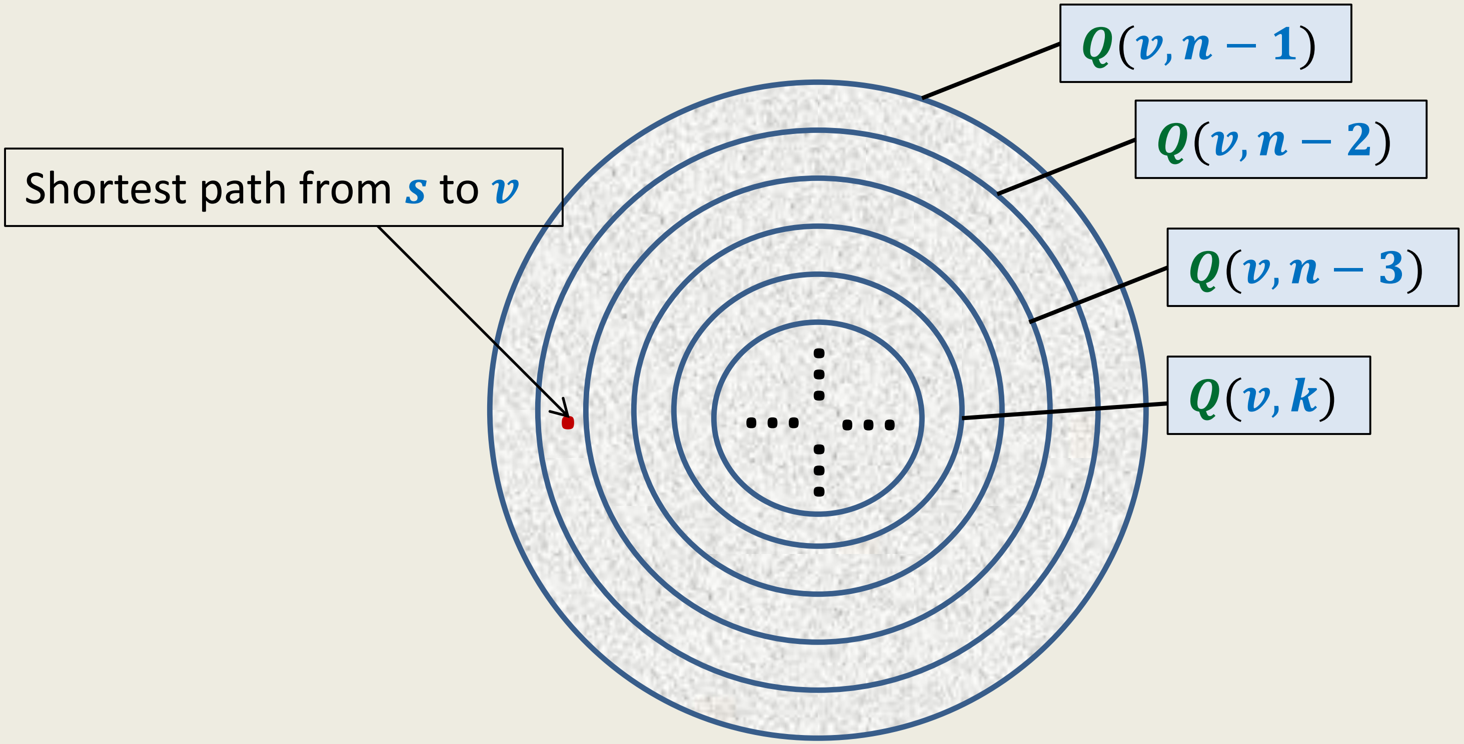
Exploiting the Optimal subpath property



Observation:

Aim: To compute $P(v, n-1)$

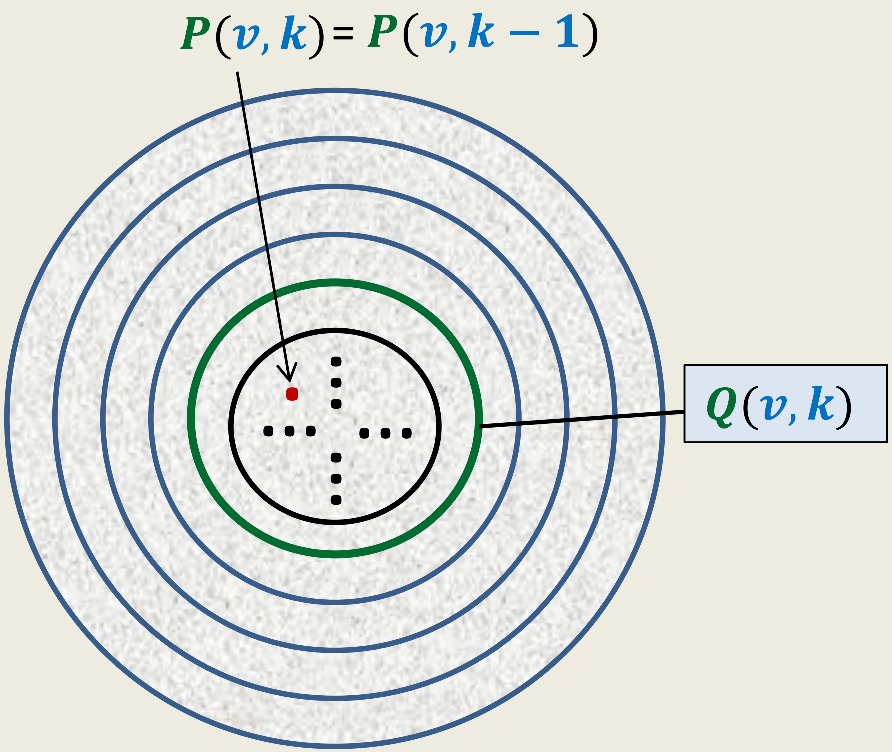
All paths from s to v.



$Q(v, k)$:

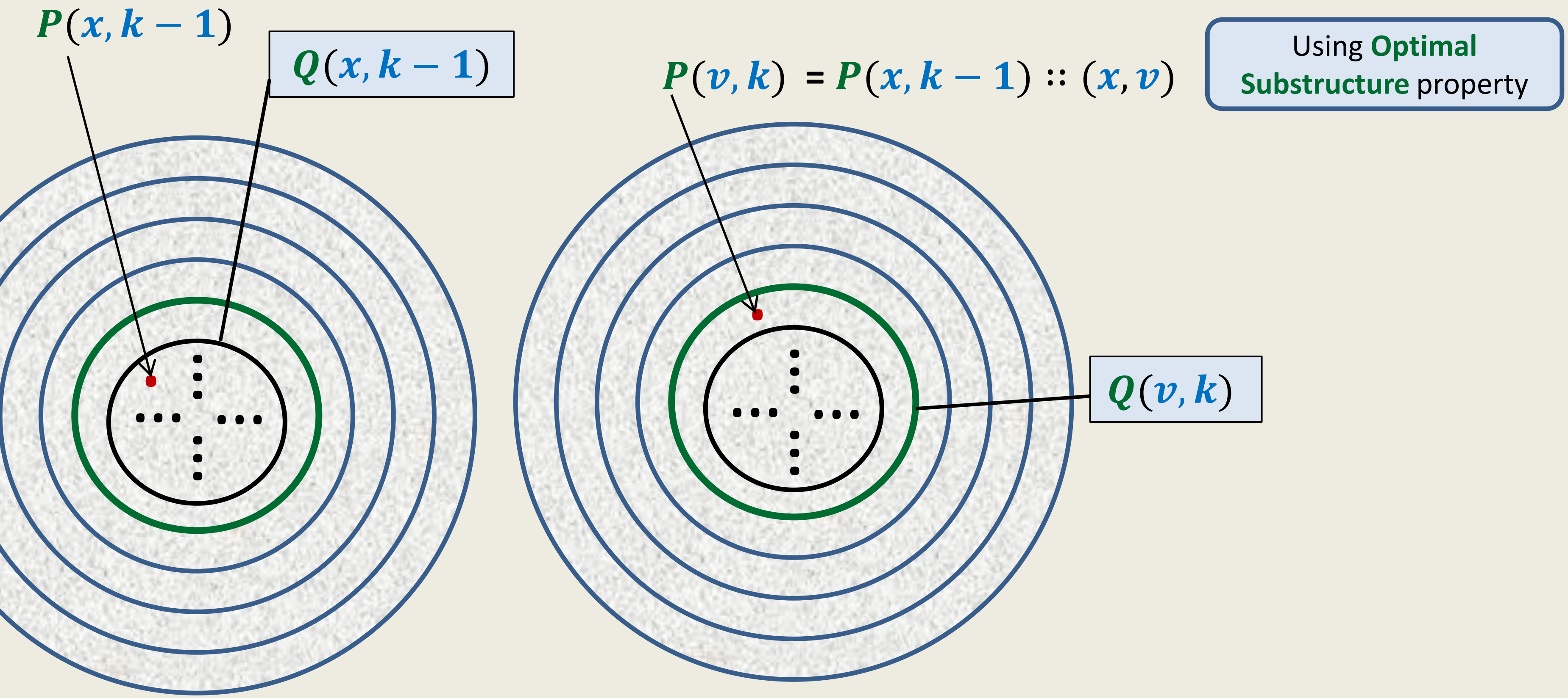
$P(v, k)$:

To compute $P(v, k)$



$Q(v, k)$: all paths from s to v consisting of **at most** k edges.

$P(v, k)$: the shortest among all paths from s to v consisting of **at most** k edges.



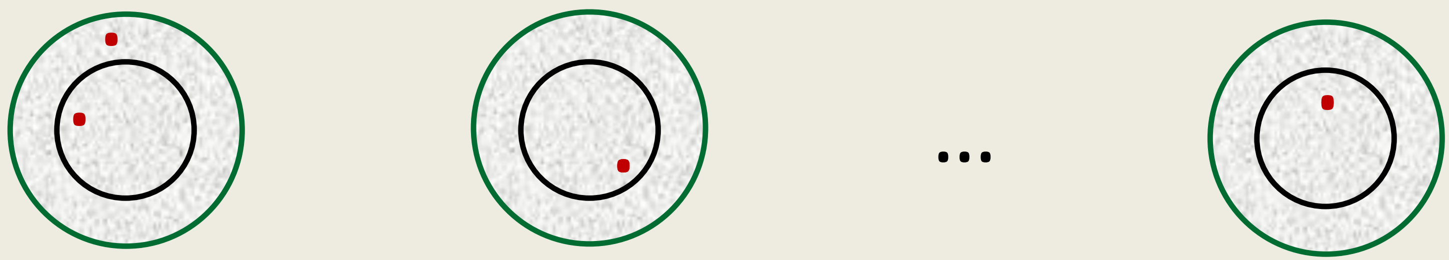
$Q(v,k)$: all paths from s to v consisting of **at most** k edges.

$P(v,k)$: the shortest among all paths from s to v consisting of **at most** k edges.

Collaboration of vertices

Given $\boldsymbol{P}(\boldsymbol{v_j}, \boldsymbol{k-1})$ for all $\boldsymbol{v_1, \dots, v_n}$.

Computing $\boldsymbol{P}(\boldsymbol{v_j}, \boldsymbol{k})$ for all $\boldsymbol{v_1, \dots, v_n}$.

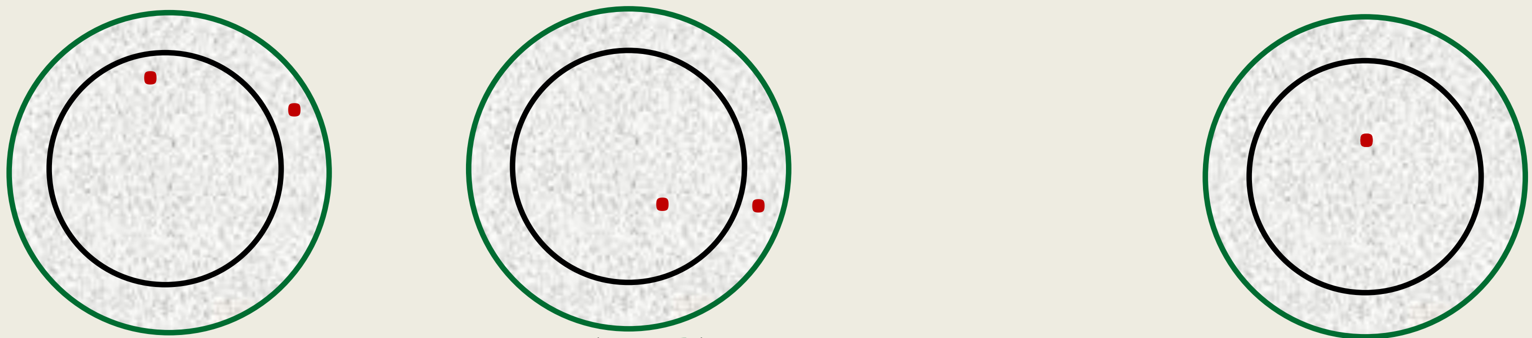


$\boldsymbol{Q}(\boldsymbol{v_1}, \boldsymbol{k-1})$ $\boldsymbol{Q}(\boldsymbol{v_2}, \boldsymbol{k-1})$ $\boldsymbol{Q}(\boldsymbol{v_n}, \boldsymbol{k-1})$

Collaboration of vertices

Given $\boldsymbol{P}(\boldsymbol{v_j}, \boldsymbol{k})$ for all $\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}$.

Computing $\boldsymbol{P}(\boldsymbol{v_j}, \boldsymbol{k+1})$ for all $\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}$.

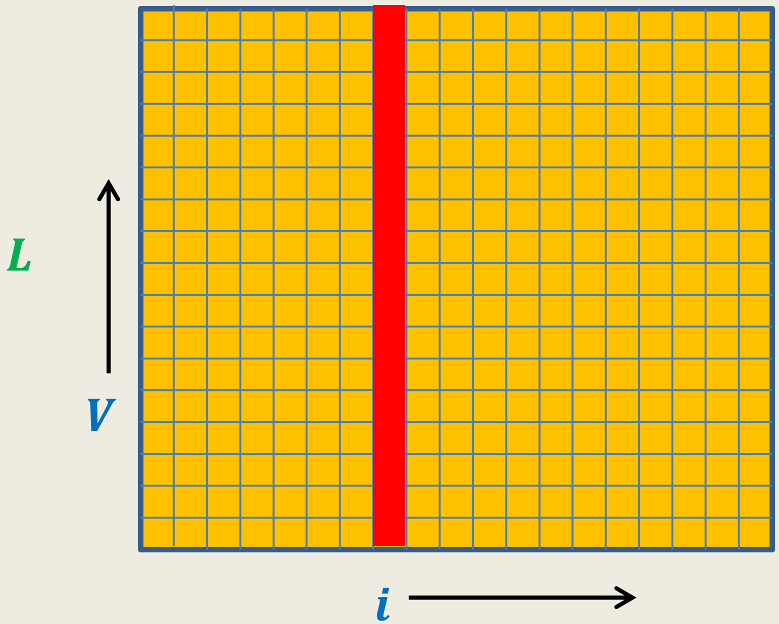


$\boldsymbol{Q}(\boldsymbol{v_1}, \boldsymbol{k})$ $\boldsymbol{Q}(\boldsymbol{v_2}, \boldsymbol{k})$ $\boldsymbol{Q}(\boldsymbol{v_n}, \boldsymbol{k})$

And so on …

$L(v, i)$: length of the shortest $s \rightsquigarrow v$ having **at most** i edges

Aim: To compute $L(v,\ n-1\ )$ for each v.



Bellman-Ford's algorithm

Bellman-Ford-algo(s,G)

{

For each $v \in V \backslash \{s\}$ **do**

  If $(s, v) \in E$ **then** $L[v, 1] \leftarrow \omega(s, v)$

   else $L[v, 1] \leftarrow \infty$;

$L[s, 1] \leftarrow 0$;

Initializing $L[*, 1]$

For $i = 2$ to $n - 1$ do

{ **For each** $v \in V$ do

  { $L[v, i] \leftarrow L[v, i-1]$;

   For each $(x, v) \in E$ **do**

    $L[v, i] \leftarrow$ **min**($L[v, i]$, $L[x, i-1] + \omega(x, v)$)

  }

}

}

Computing $L[v, i]$

Lemma: $L[v, i]$ stores the shortest path from s to v having **at most** i edges.

Single source shortest paths in a graph

Problem: Given a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ on $\boldsymbol{n} = |\boldsymbol{V}|$ vertices and $\boldsymbol{m} = |\boldsymbol{E}|$ edges, and a source vertex $\boldsymbol{s}$, compute shortest path from $\boldsymbol{s}$ to $\boldsymbol{v}$ for each $\boldsymbol{v} \in \boldsymbol{V}$.

Solutions:

- Edge weights are **non-negative**

  Dijkstra's algorithm

  Time complexity = **O**($\boldsymbol{m} + \boldsymbol{n}\log \boldsymbol{n}$)

- Edge weights are **negative** but **no-negative cycle**

  Bellman-Ford algorithm.

  Time complexity = **O**($\boldsymbol{mn}$)

Data structure for reporting shortest path from $\boldsymbol{s}$:

<u>Shortest paths tree rooted at $\boldsymbol{s}$</u>

Time taken to report shortest path from $\boldsymbol{s}$ to $\boldsymbol{v}$ = **O**($|\boldsymbol{P}(\boldsymbol{u}, \boldsymbol{v})|$)

All-pairs shortest paths in a graph
with positive edge weights

Problem: Given a graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges, compute distance/shortest-path from u to v for each $u, v \in V$.

Solutions:

Execute Dijkstra's algorithm from each $v \in V$.

Total time = O($mn + n^2 \log n$)

Data structure for reporting shortest path from v :

Shortest paths tree rooted at v

Space taken by the data structure = **O**(n^2)

All-pairs shortest paths in a graph
with negative edge weights but no negative cycle

Problem: Given a graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges, compute shortest path from u to v for each $u, v \in V$.

Solution:

Execute Bellman-Ford's algorithm from each $v \in V$.

Total time = O(mn^2)

How to improve it to **O(n^3)** ?

Data structure for reporting shortest path from v :

Shortest paths tree rooted at v

Space taken by the data structure = **O(n^2)**

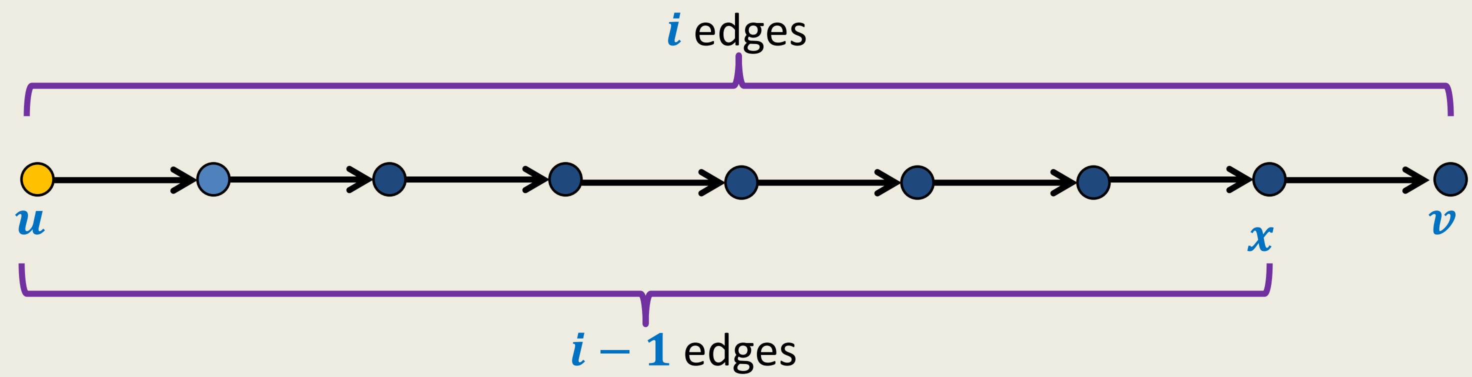
ALL-PAIRS SHORTEST PATHS IN O(n^3) TIME

In graphs with negative edge weights

but no negative cycle

The Optimal substructure property

Consider any shortest path $P(u, v)$.



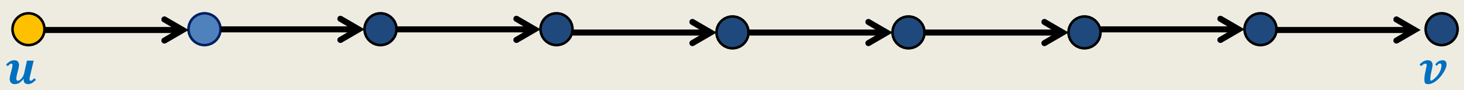
We used "no. of edges" for a recursive formulation of $\delta(s, v)$. [**Bellman Ford algo**]

$L(v, i)$: length of the shortest $s \rightsquigarrow v$ having **at most** i edges.

- $\delta(s, v) = L(v, n-1)$
- Expressed $L(v, i+1)$ recursvely in terms of $L(x, i)$ for all $(x, v) \in E$
- **Base case**: $L(v, 1) = \omega(s, v)$ if $(s, v) \in E$, and ∞ otherwise.

The Optimal substructure property

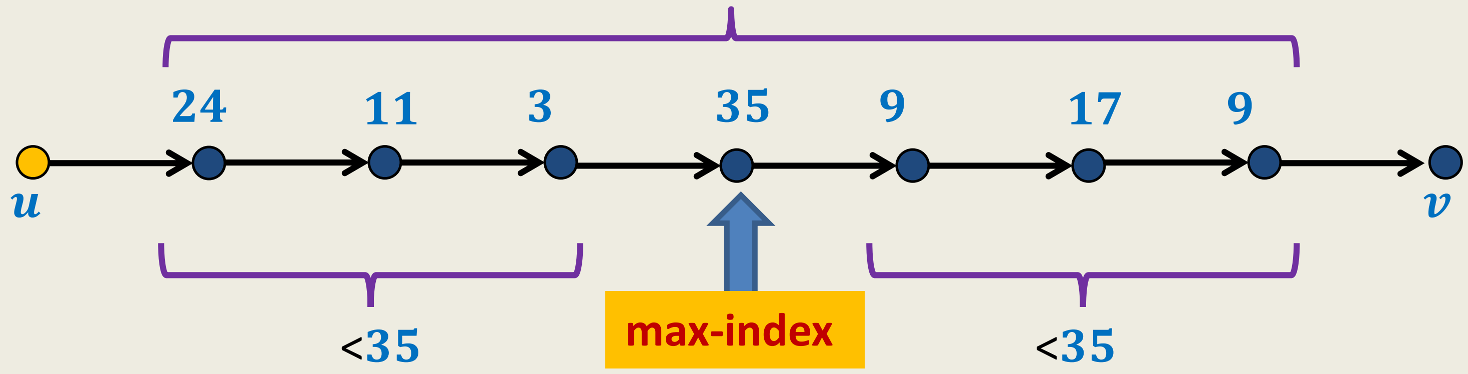
Consider any shortest path $\boldsymbol{P(u, v)}$.



Question: Can we use "**<u>vertices</u>**" for recursive formulation of $\boldsymbol{\delta(u, v)}$?

The Optimal substructure property

Consider any shortest path $P(u, v)$.



For a recursive formulation of $\delta(u, v)$,

We can use **max-index** of **intermediate vertices** on $P(u, v)$.

Term for Recursive formulation of $\delta(u, v)$?

$P_k(i, j)$: the shortest path from i to j with intermediate vertices of index $\leq k$

$D_k(i, j)$: length of $P_k(i, j)$.

Question: How can we express $\delta(i, j)$ in terms of $D_k(i, j)$?

Answer: $\delta(i, j) = D_n(i, j)$:

Base Case:

$$D_0(i, j) = \begin{cases} \omega(i, j) & \text{if } (i, j) \in E \\ \infty & \text{otherwise} \end{cases}$$

Question: What is recursive formulation of $D_k(i, j)$?

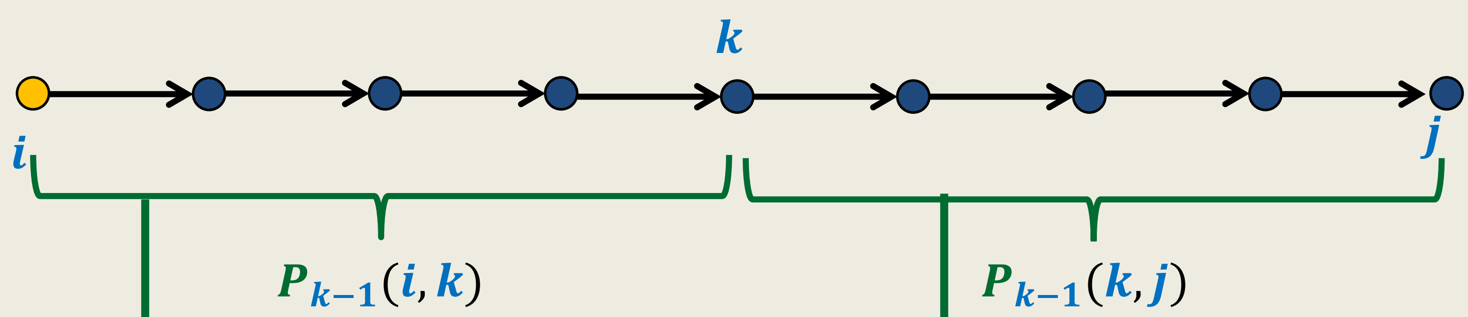
Recursive formulation of $D_k(i, j)$

Consider the path $P_k(i, j)$

There are two cases:

Case 1 : $P_k(i, j)$ does not pass through k ➔ $D_k(i, j) = D_{k-1}(i, j)$

Case 2 : $P_k(i, j)$ indeed passes through k ➔ ?



If this is longer than $P_{k-1}(i, k)$ then replacing it with $P_{k-1}(i, k)$ will **only** give a shorter $P_k(i, j)$
A contradiction!

If this is longer than $P_{k-1}(k, j)$ then replacing it with $P_{k-1}(k, j)$ will **only** give a shorter $P_k(i, j)$
A contradiction!

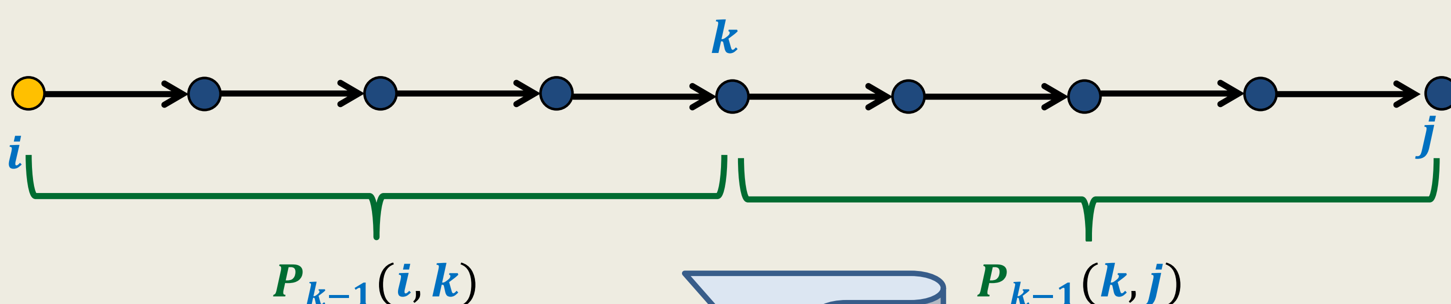
Recursive formulation of $D_k(i,j)$

Consider the path $P_k(i,j)$

There are two cases:

Case 1 : $P_k(i,j)$ does not pass through k ➔ $D_k(i,j) = D_{k-1}(i,j)$

Case 2 : $P_k(i,j)$ indeed passes through k ➔ $D_k(i,j) = D_{k-1}(i,k) + D_{k-1}(k,j)$



In other words, what is the guarantee that $P_{k-1}(i,k)$ :: $P_{k-1}(k,j)$ does not have a cycle ?

Any such cycle will surely have non-negative weight.
Removing the cycle will give a path of the same or smaller length which does not pass through k.
A contradiction !

$$D_k(i,j) = min(D_{k-1}(i,j)\ ,\ D_{k-1}(i,k) + D_{k-1}(k,j))$$

FLOYD WARSHAL ALGORITHM FOR ALL PAIRS SHORTEST PATHS

in O(n^3) time

and O(n^3) space

Floyd and Warshal's algorithm

Floyd-Warshal-algo(G)

{ **For each** i **do**

  For each j **do**

   If $(i, j) \in E$ **then** $D_0[i, j] \leftarrow \omega(i, j)$

      else $D_0[i, j] \leftarrow \infty$;

 For each i **do** $D_0[i, i] \leftarrow 0$;

Computing $D_0[*,*]$

 For k = 1 to n **do**

  For each i **do**

   For each j **do**

   { $D_k[i, j] \leftarrow D_{k-1}[i, j]$;

    $D_k[i, j] \leftarrow$ **min**($D_k[i, j]$, $D_{k-1}[i, k] + D_{k-1}[k, j]$)

   }

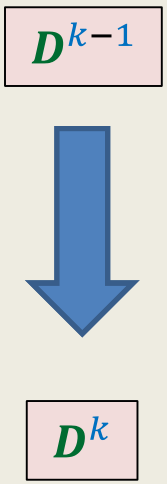
Computing $D_k[i, j]$

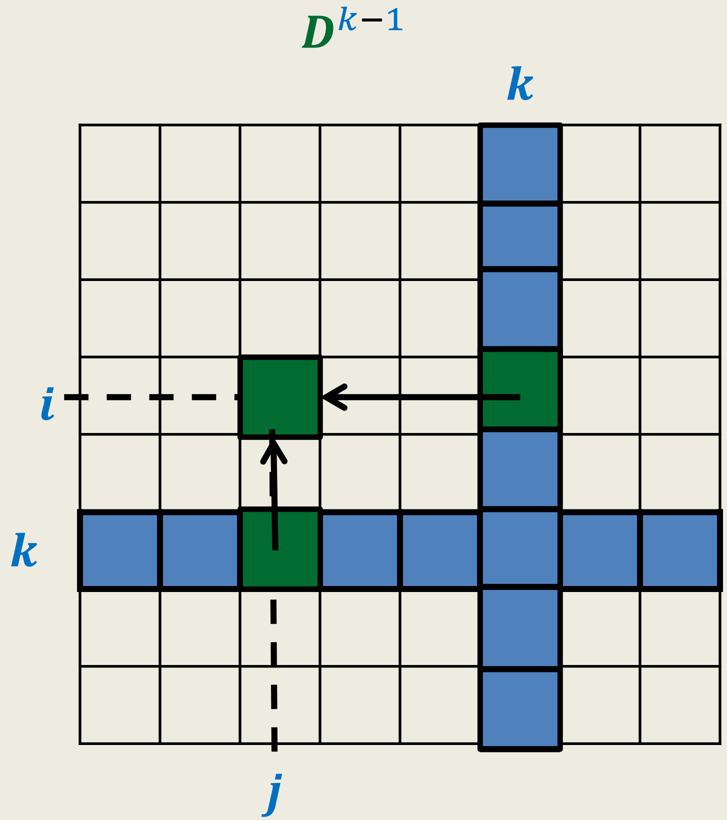
}

Lemma: $D_k[i, j]$ = length of the shortest path from i to j with all intermediate vertices of indices $\leq k$

FLOYD WARSHAL ALGORITHM FOR ALL PAIRS SHORTEST PATHS

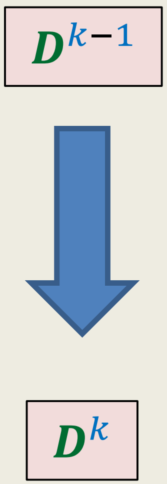
in O(n^3) time
and O(n^2) space

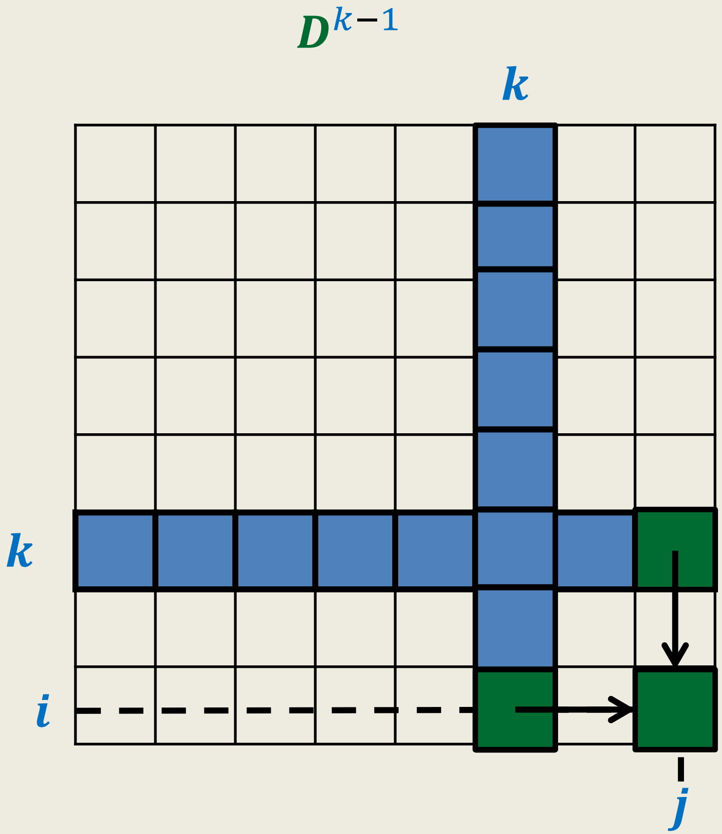




$$\boldsymbol{D_k(i,j)} = min(\boldsymbol{D_{k-1}}(i,j) \quad , \quad \boldsymbol{D_{k-1}}(i,k) + \boldsymbol{D_{k-1}}(k,j)\ )$$

Hence we can just overwrite D^{k-1} instead of creating a separate matrix for D^k ☺





$$D_k(i,j) = min(D_{k-1}(i,j)\ ,\ D_{k-1}(i,k) + D_{k-1}(k,j)\ )$$

For computing $D_k(i,j)$ for any $i \neq k, j \neq k$, we need only kth column and kth row of D_{k-1}

Moreover $D_k(k,*) = D_{k-1}(k,*)$, and $D_k(*,k) = D_{k-1}(*,k)$

Floyd and Warshal's algorithm

Floyd-Warshal-algo(G)

{ **For each** i **do**

  For each j **do**

   If $(i, j) \in E$ **then** $D[i, j] \leftarrow \omega(i, j)$;

     else $D[i, j] \leftarrow \infty$;

 For each i **do** $D[i, i] \leftarrow 0$;

 For k = 1 to n do

  For each i **do**

   For each j **do**

   { **If** $(D[i, j] > D[i, k] + D[k, j])$

     $D[i, j] \leftarrow D[i, k] + D[k, j]$;

   }

}

Lemma: At the end of kth iteration,

$D[i, j]$ = length of the shortest path from i to j with all intermediate vertices of indices $\leq k$

All-pairs shortest paths in a digraph
with negative edge weights but no negative cycle

Theorem: Given a graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges, we can compute all-pairs distances in **O**(n^3) time.
The space requirement is **O**(n^2).

Homework:

- How to retrieve shortest path ?

Hint: Augment the given algorithm with a **O**(n^2) size data structure.
(that stores all-pairs shortest paths **<u>implicitly</u>**)

This view will add to
your understanding of these two <u>algorithms</u>

In the following slides, we shall provide an alternate view of

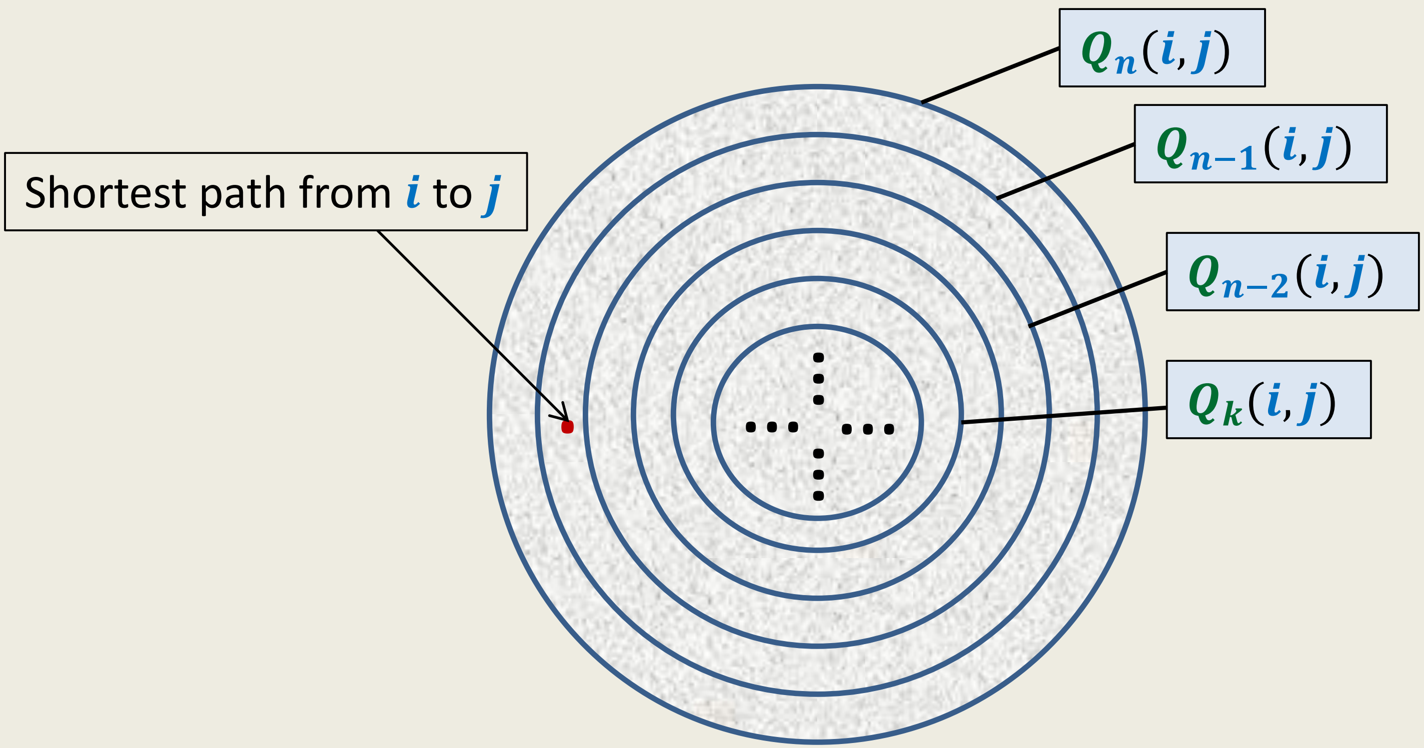
- **Floyd & Warshal** Algorithm

Both the algorithms (**Bellman-Ford** and **Floyd & Warshal**)
use **Optimal substructure property** of shortest paths.
They differ due to <u>different hierarchies</u> of sets of paths ☺.

Reviewing Floyd Warshal Algorithm

Aim: To compute $P_n(i, j)$

All paths from i to j.

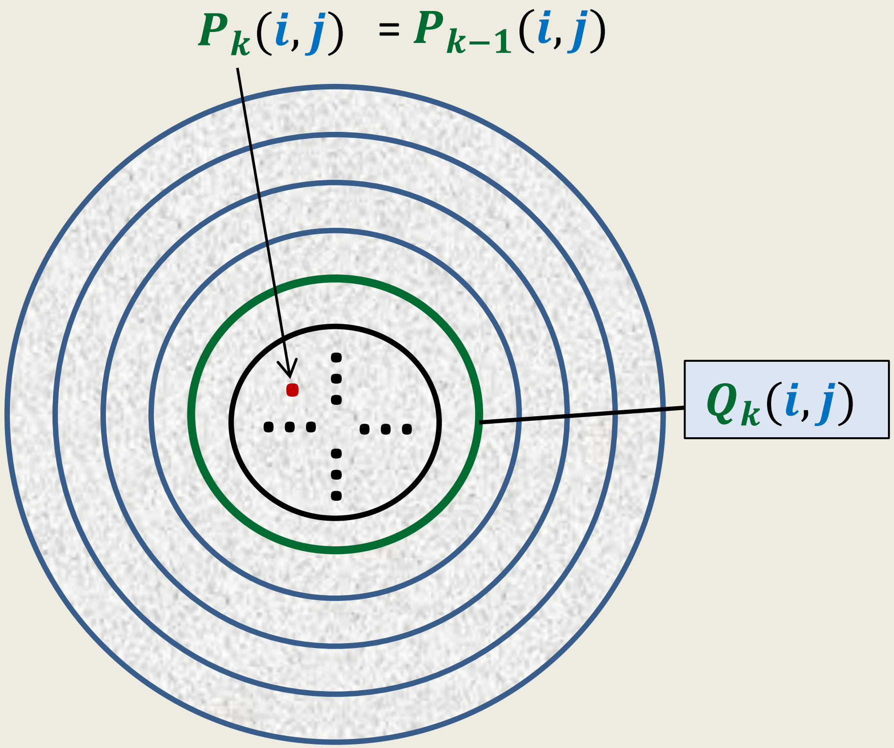


$Q_k(i,j)$: all paths from i to j with intermediate vertices having index at most k.

$P_k(i,j)$: the shortest among all paths from i to j

with each intermediate vertex having index at most k.

Computing $P_k(i,j)$

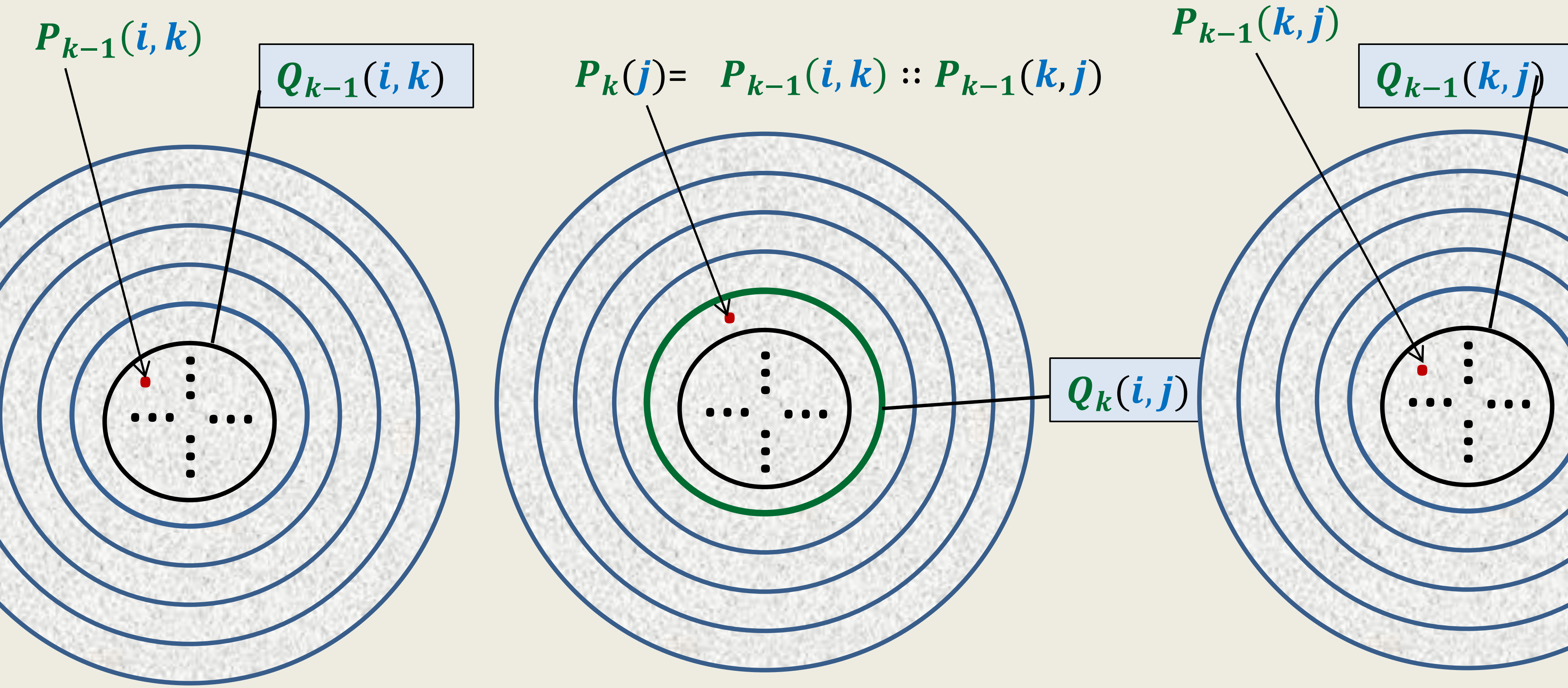


$Q_k(i,j)$: all paths from i to j with <u>intermediate vertices having index at most</u> k.

$P_k(i,j)$: the shortest among all paths from i to j

with each intermediate vertex having index at most k.

Using **Optimal Substructure** property



$Q_k(i,j)$: all paths from i to j with <u>intermediate vertices having index at most</u> k.

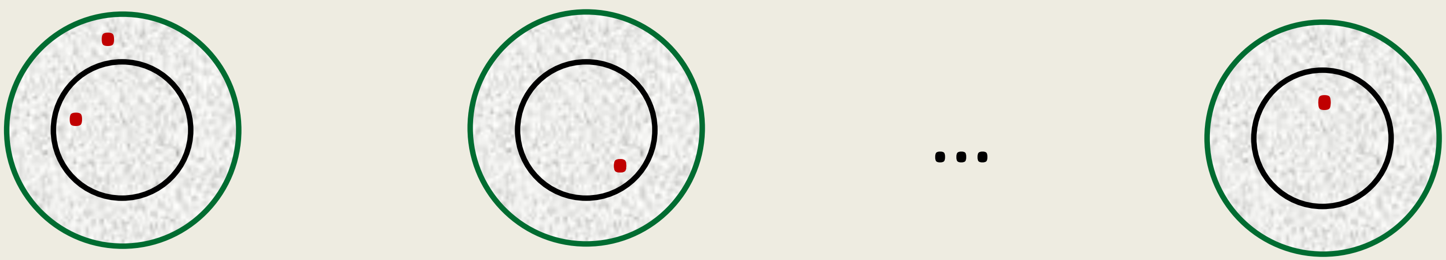
$P_k(i,j)$: the shortest among all paths from i to j

with each intermediate vertex having index at most k.

Collaboration of vertices

Given $\boldsymbol{P_{k-1}}(\boldsymbol{i},\boldsymbol{j})$ for all $(\boldsymbol{i},\boldsymbol{j})$ pairs

Computing $\boldsymbol{P_{k}}(\boldsymbol{i},\boldsymbol{j})$ for all$(\boldsymbol{i},\boldsymbol{j})$ pairs.

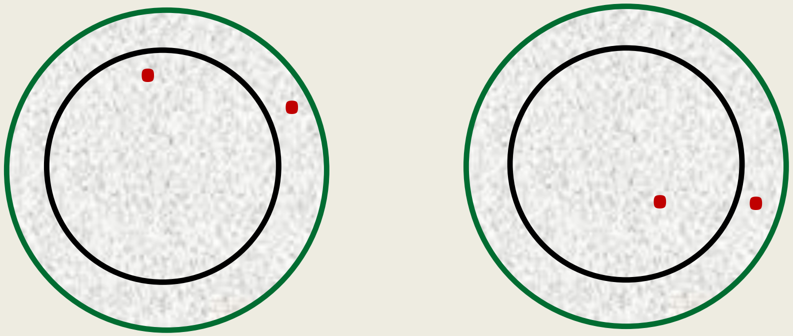


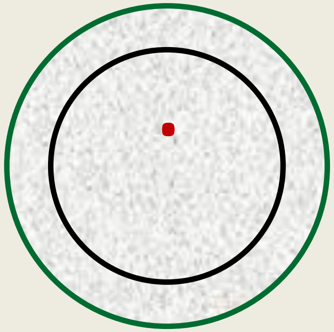
$\boldsymbol{Q_{k-1}}(\boldsymbol{i},\boldsymbol{j})$ sets

Collaboration of vertices

Given $P_k(i, j)$ for all (i, j) pairs

Computing $P_{k+1}(i, j)$ for all (i, j) pairs.





$Q_k(i, j)$ sets

And so on …