Reference textbook: R. Napolitan and K. Naimipour, Foundations of Algorithms (4th ed.), Jones and Bartlett, 2011

ACM Tutorial: Dynamic Programming (Part II)

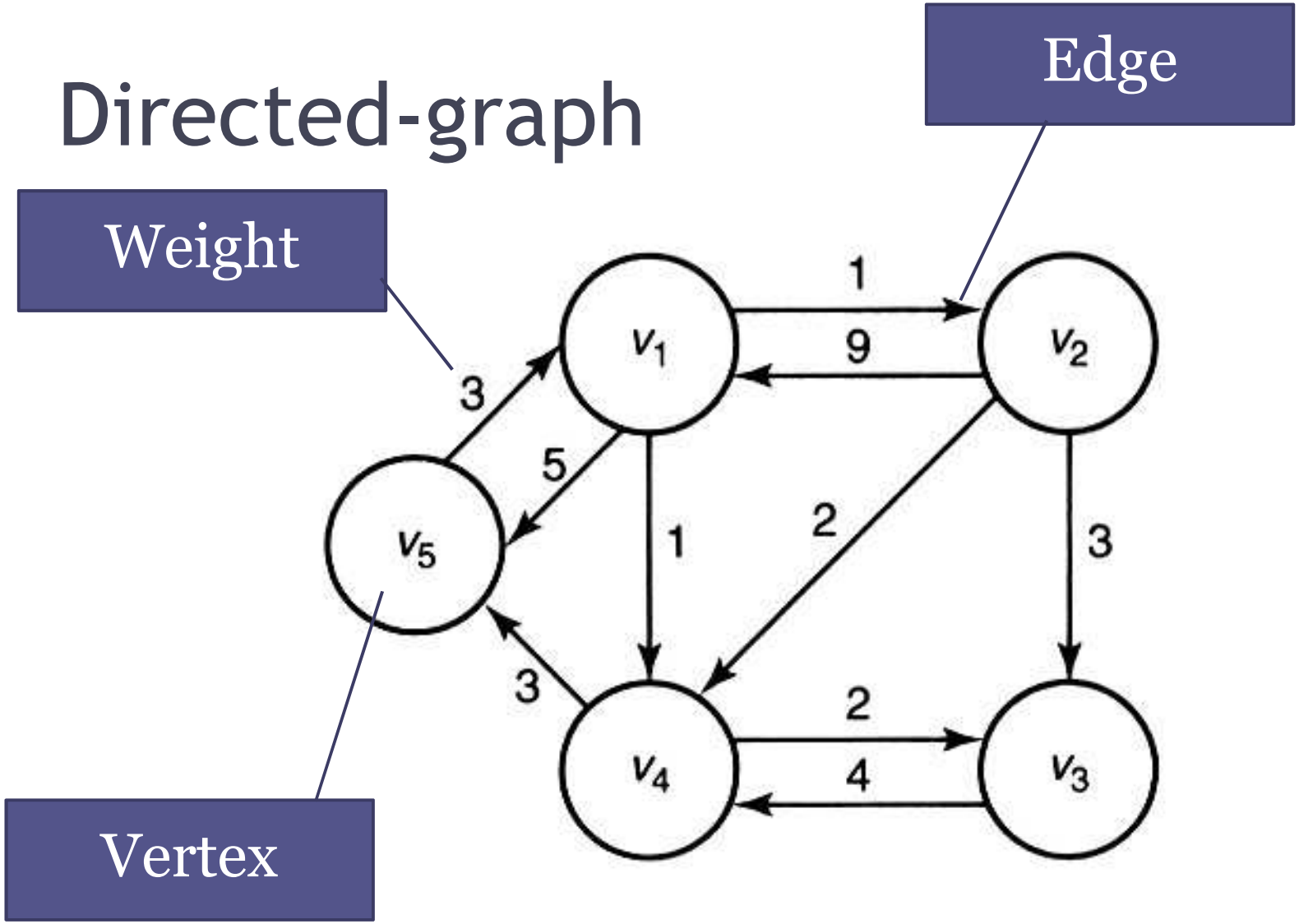
Dr. Ukrit Watchareeruetai
Department of Engineering and Technology,
International College, KMITL

Contents

- Shortest path problem
- Floyd's algorithm for shortest paths

Shortest path problem

Directed-graph



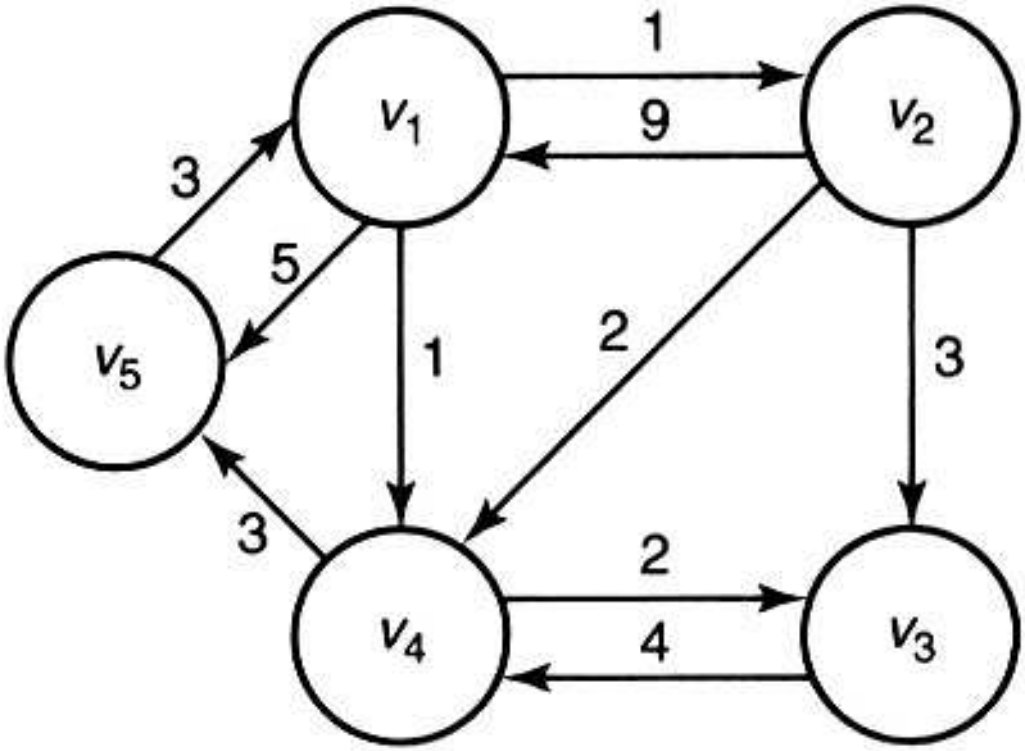
Graph representation

- We represent a weighted graph containing n vertices by an array W where

$$W[i][j] = \begin{cases} wieght\ on\ edge & if\ there\ is\ an\ edge\ from\ v_i\ to\ v_j \\ \infty & if\ there\ is\ no\ edge\ from\ v_i\ to\ v_j \\ 0 & if\ i = j \end{cases}$$

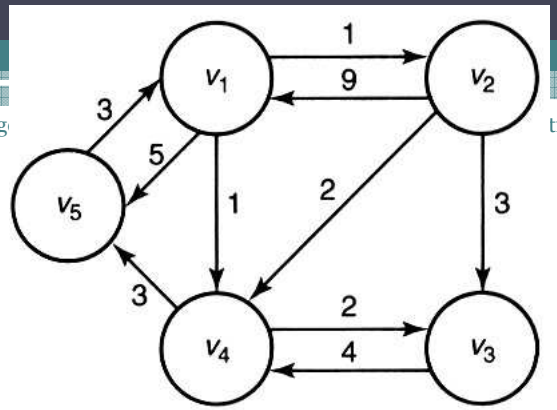
- This array is called the *adjacency matrix*.

Graph representation



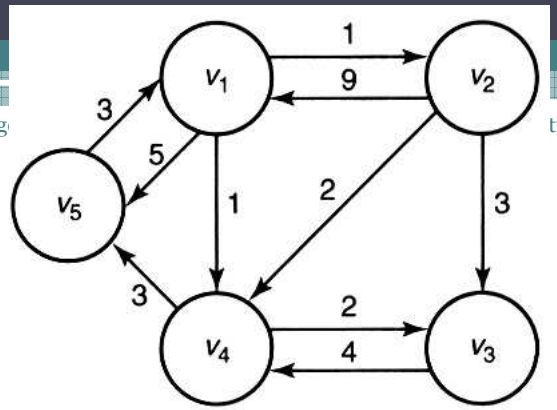
	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

W



Path

- Path is a sequence of vertices such that there is an edge from each vertex to its successor.
  - E.g., the sequence $[v_1, v_4, v_3]$ is a path because there is an edge from v_1 to v_4 and an edge from v_4 to v_3.
  - The sequence $[v_3, v_4, v_1]$ is not a path because there is no edge from v_4 to v_1.



Shortest path problem

- The *length* of a path in a weighted graph is the *sum of the weights* on the path.

- A problem in many applications is finding the *shortest paths* from each vertex to all other vertices.

Floyd's algorithm for shortest paths

Adjacency matrix

D contains the length of the shortest paths

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

W

	1	2	3	4	5
1	0	1	3	1	4
2	8	0	3	2	5
3	10	11	0	4	7
4	6	7	2	0	3
5	3	4	6	4	0

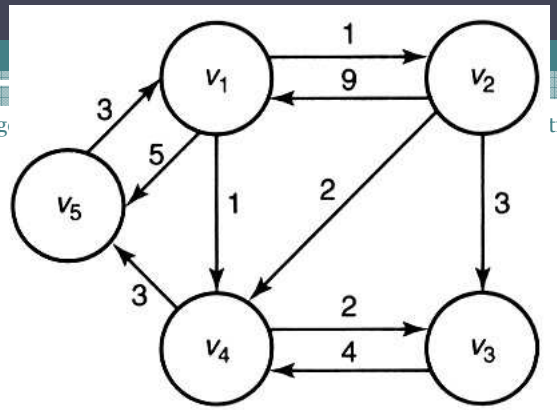
D

- If we can develop a way to calculate the values in D from those in W, we will have an algorithm for the shortest path problem.

Solving shortest path problem using DP

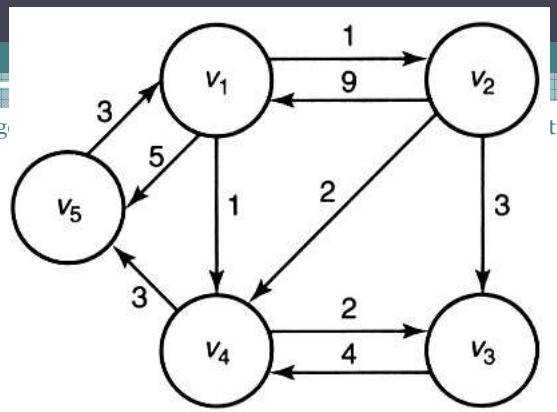
- We accomplish this by creating a sequence of $n + 1$ arrays $D^{(k)}$, where $0 \leq k \leq n$ and where

$D^{(k)}[i][j]$ = length of a shortest path from v_i to v_j using only vertices in the set $\{v_1, v_2, \ldots, v_k\}$ as intermediate vertices.



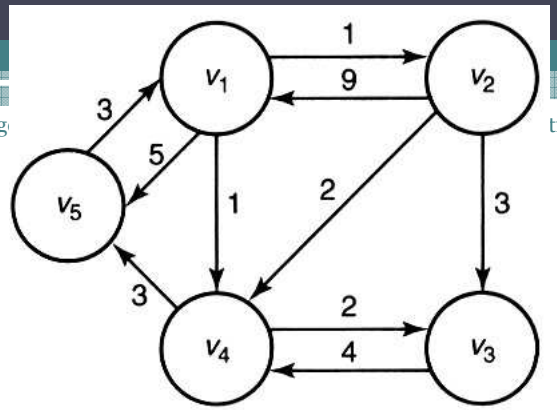
Example

- Calculate $D^{(k)}[2][5]$ for the above graph.
  - $D^{(0)}[2][5] = length[v_2, v_5] = \infty$
  - $D^{(1)}[2][5] = min(length[v_2, v_5], length[v_2, v_1, v_5])$
    $= min(\infty, 14) = 14$
  - $D^{(2)}[2][5] = D^{(1)}[2][5] = 14$
    - *They are equal because a shortest path starting from v_2 cannot pass through v_2.*



Example

- Calculate $D^{(k)}[2][5]$ for the above graph.
  - $D^{(3)}[2][5] = D^{(2)}[2][5] = 14$
    - *Including v_3 yields no new paths from v_2 to v_5.*
  - $D^{(4)}[2][5] = \min(\ length[v_2, v_1, v_5], length[v_2, v_4, v_5],$
    $length[v_2, v_1, v_4, v_5], length[v_2, v_3, v_4, v_5])$
    $= \min(\ 14, 5, 13, 10\ ) = 5$



Example

- Calculate $D^{(k)}[2][5]$ for the above graph.
  - $D^{(5)}[2][5] = D^{(4)}[2][5] = 5$
    - *They are equal because a shortest path ending at v_5 cannot pass through v_5.*
  - The last value computed, $D^{(5)}[2][5]$, is the length of a shortest path from v_2 to v_5 that is allowed to pass through any of the other vertices.

Solving shortest path problem using DP

- Because $D^{(n)}[i][j]$ is the length of a shortest path from v_i to v_j that is allowed to pass through any of the other vertices, it is the length of a shortest path from v_i to v_j.
- Because $D^{(o)}[i][j]$ is the length of a shortest path that is not allowed to pass through any other vertices, it is the weight on the edge from v_1 to v_j.
- We have established that

$$D^{(o)}[i][j] = W \quad and \quad D^{(n)}[i][j] = D$$

Solving shortest path problem using DP

- To determine D from W, we need only find a way to obtain $D^{(n)}$ from $D^{(0)}$.
- The steps for using dynamic programming to accomplish this are as follows:
  - *Establish* a recursive property (process) with which we can compute $D^{(k)}$ from $D^{(k-1)}$.
  - Solve an instance of the problem in a *bottom-up* fashion by repeating the process (established in Step 1) for $k = 1$ to n. This creates the sequence

$$W \quad D^0, D^1, D^2, \ldots, D^n \quad D$$

Solving shortest path problem using DP

- We accomplish the step by considering two cases:
  - Case 1: All shortest paths from v_i to v_j, using only vertices in $\{v_1, v_2..., v_k\}$ as intermediate vertices, *do not use* v_k.
  - Case 2: At least one shortest path from v_i to v_j, using only vertices in $\{v_1, v_2, ..., v_k\}$ as intermediate vertices, *does use* v_k.

Case 1

Solving shortest path problem using DP

- *Case 1:* All shortest paths from v_i to v_j, using only vertices in $\{v_1, v_2..., v_k\}$ as intermediate vertices, do not use v_k.
- Then

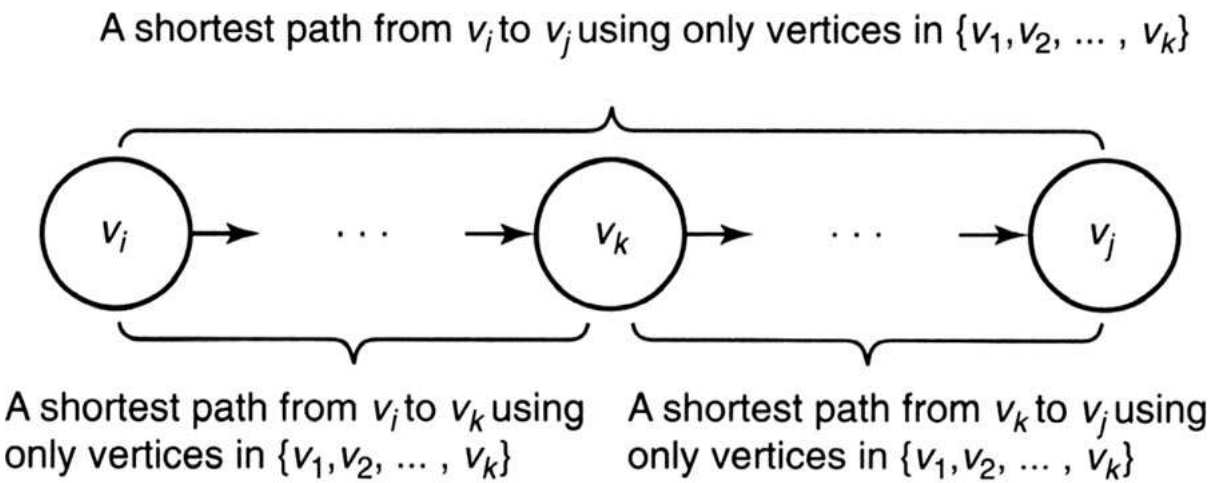
$$D^{(k)}[i][j] = D^{(k-1)}[i][j]$$

  - As in the previous example, $D^{(3)}[2][5] = D^{(2)}[2][5] = 14$
    - *Including v_3 yields no new paths from v_2 to v_5.*

Case 2

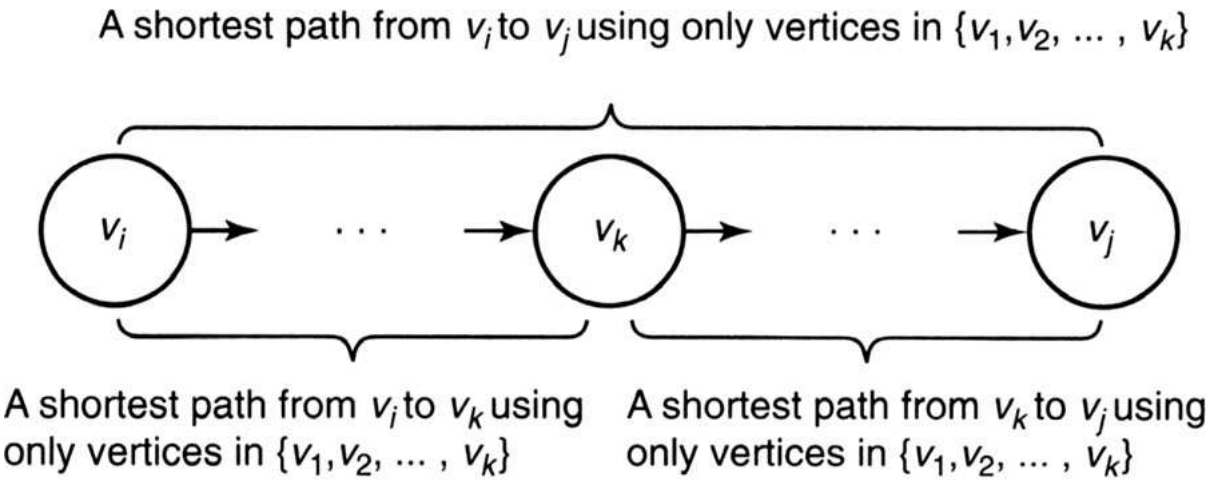
Solving shortest path problem using DP

- *Case 2:* At least one shortest path from v_1 to v_j, using only vertices in $\{v_1, v_2, \ldots, v_k\}$ as intermediate vertices, does use v_k.



Case 2

Solving shortest path problem using DP

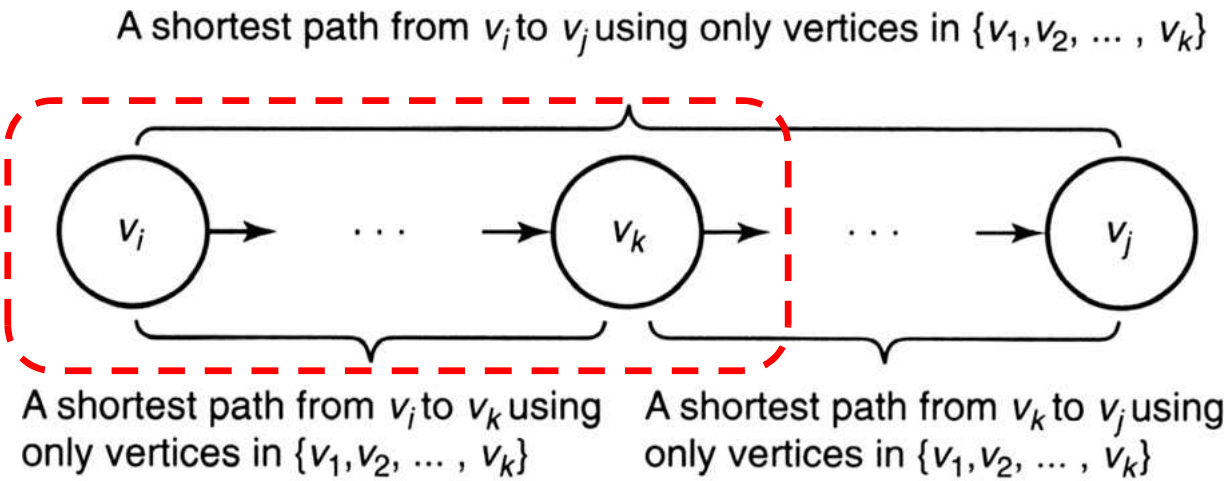


- In the second case,

$$D^{(k)}[i][j] = D^{(k-1)}[i][k] + D^{(k-1)}[k][j]$$

Case 2: explanation

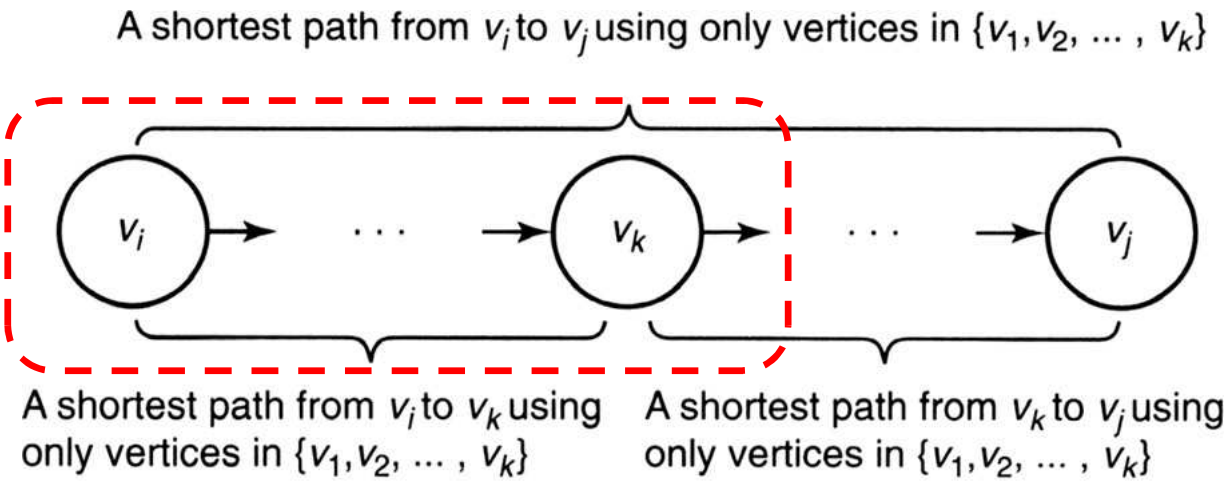
Solving shortest path problem using DP



- Because v_k cannot be an intermediate vertex on the subpath from v_i to v_k, that subpath uses only vertices in $\{v_1, v_2 \ldots, v_{k-1}\}$ as intermediates.
- This implies that the subpath's length must be equal to $D^{k-1}[i][k]$ for the following two reasons.

Case 2: explanation

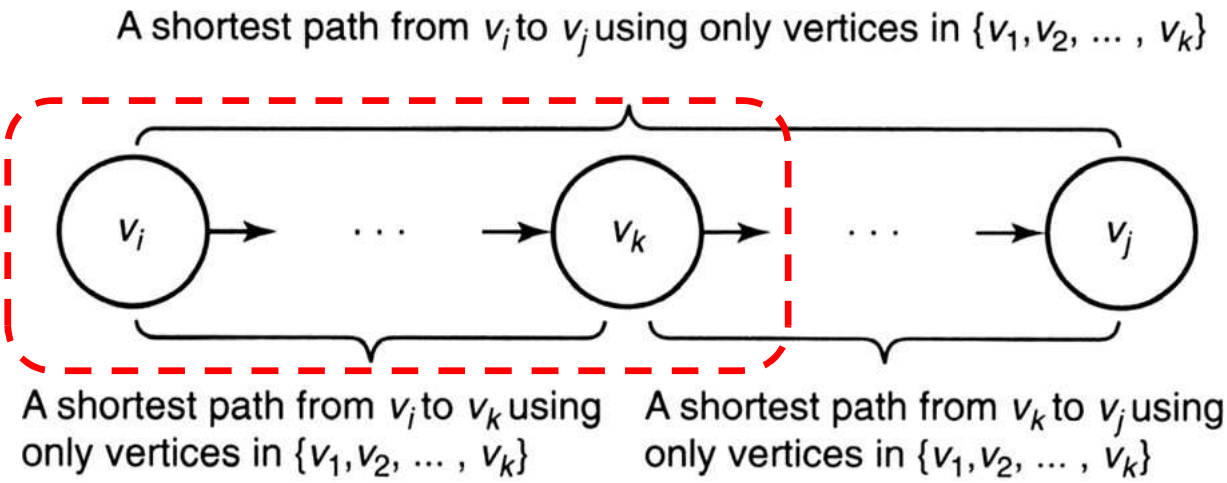
Solving shortest path problem using DP



- First, the subpath's length cannot be shorter because $D^{(k-1)}[i][k]$ is the length of a shortest path from v_1 it v_k using only vertices in $\{v_1, v_2, \ldots, v_{k-1}\}$ as intermediates.

Case 2: explanation

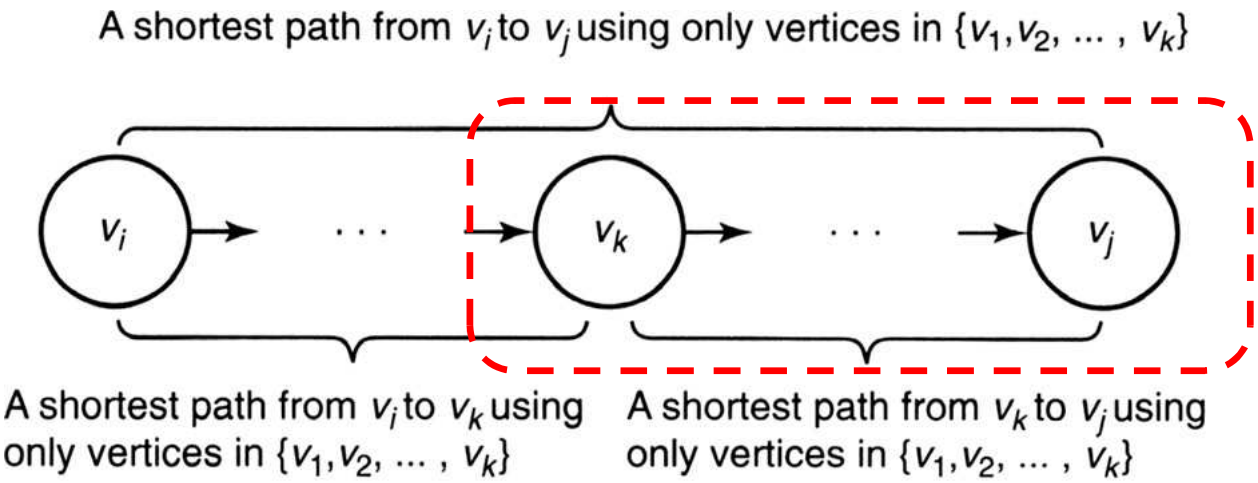
Solving shortest path problem using DP



- Second, the subpath's length cannot be longer because if it were, we could replace it in the figure by a shortest path, which contradicts that fact that the entire path in the figure is a shortest path.

Case 2: explanation

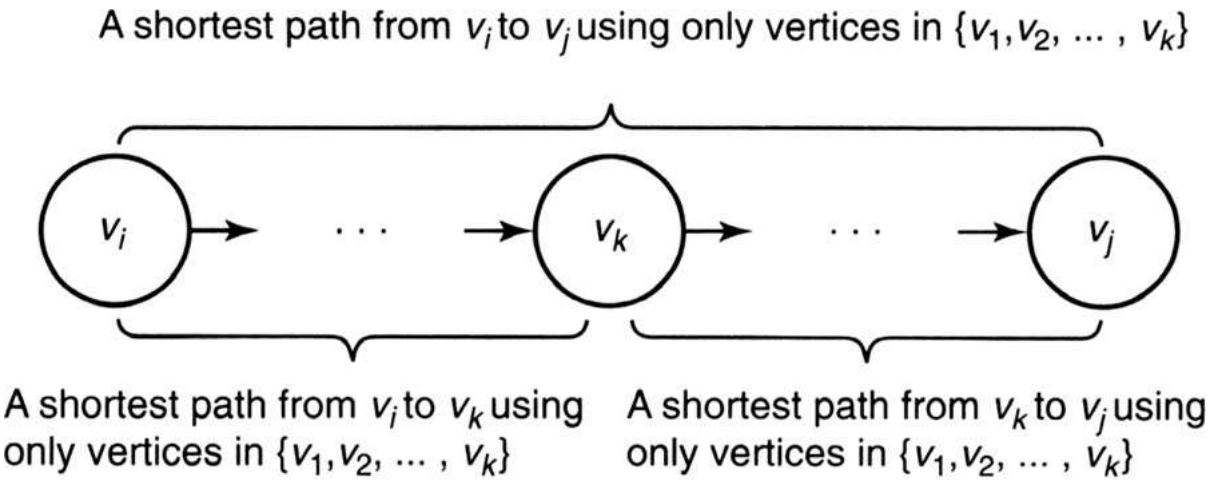
Solving shortest path problem using DP



- Similarly, the length of the subpath from v_k to v_j in the figure must be equal to $D^{(k-1)}[k][j]$.

Case 2: explanation

Solving shortest path problem using DP



- Therefore, in the second case

$$D^{(k)}[i][j] = D^{(k-1)}[i][k] + D^{(k-1)}[k][j]$$

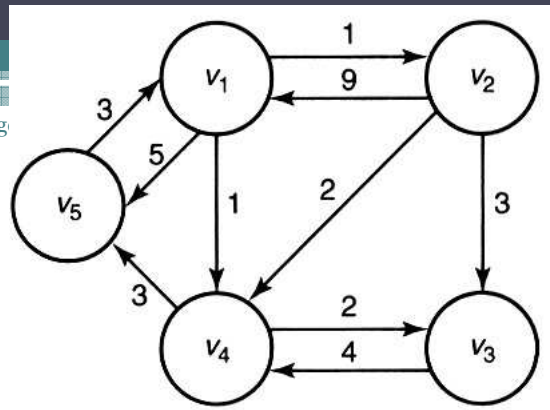
Case 1 & Case 2

Solving shortest path problem using DP

- Because we must have either case 1 or case 2, the value of $D^{(k)}[i][j]$ is the minimum of the values on the right hand side in the equalities in both cases.

- That is

$$D^{(k)}[i][j] = min(D^{(k-1)}[i][j] , D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$



Example

- To compute $D^{(2)}[5][4]$, we have to compute
  - $D^{(1)}[5][4] = min(D^{(0)}[5][4] , D^{(0)}[5][1] + D^{(0)}[1][4])$
    $= min(\infty, 3 + 1) = 4$
  - $D^{(1)}[5][2] = min(D^{(0)}[5][2] , D^{(0)}[5][1] + D^{(0)}[1][2])$
    $= min(\infty, 3 + 1) = 4$
  - $D^{(1)}[2][4] = min(D^{(0)}[2][4] , D^{(0)}[2][1] + D^{(0)}[1][4])$
    $= min(2, 9 + 1) = 2$
- Therefore,
  - $D^{(2)}[5][4] = min(D^{(1)}[5][4] , D^{(1)}[5][2] + D^{(1)}[2][4])$
    $= min(4, 4 + 2) = 4$

Floyd's algorithm for shortest path

Algorithm 3.3: Floyd's Algorithm for Shortest Paths

Problem: Compute the shortest paths from each vertex in a weighted graph to each of the other vertices. The weights are nonnegative numbers.

Inputs: A weighted, directed graph and *n*, the number of vertices in the graph. The graph is represented by a two-dimensional array *W* which has both its rows and columns indexed from 1 to *n*, where *W*[*i*][*j*] is the weight on the edge from the *i*th vertex to the *j*th vertex.

Outputs: A two-dimensional array *D*, which has both its rows and columns indexed from 1 to *n*, where *D*[*i*] [*j*] is the length of a shortest path from the *i*th vertex to the *j*th vertex.

```
void floyd (int n
                        const number W[] []
                        number D[] []
{
 index i, j, k;
 D = W;
 for (k = 1; k<=n; k++)
    for (i = 1; i <= n; i++)
       for (j = 1; j <= n; j++)
         D[i][j] = minimum(D[i][j], D[i][k] + D[k][j]);
}
```