Lecture 13: Chapter 3 Part 1

Dynamic Programming
CS3310

Divide-and-Conquer

Recall that *Divide-and-Conquer* solves a problem with a *top-down* approach.

- With `Mergesort`, we start at the <u>top level</u> with an entire list of values.
- We recursively break the list into smaller instances until we arrive at a **base case**.
- *Divide-and-Conquer* works great when the smaller instances are unrelated to each other i.e. with `Mergesort` we sort two subarrays which are completely independent of each other.
- However, when the smaller instances <u>are</u> related to each other, this technique quickly becomes unviable.

Dynamic Programming

Dynamic Programming takes the opposite approach.

- With Dynamic Programming, we solve the smaller instances first, store the results, and look these results up later instead of recomputing them.
- For example, if we are computing a Fibonacci number, we don't want to compute F_5 over and over again. If we store this value, we can retrieve it whenever we need without recursively computing F_4, F_3, F_2, F_1 each time
- This is called a *bottom-up* approach.

Binomial Coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- The **binomial coefficient** is also known as a combination, or C(n, k)
- We can establish the following recursive property for the above equation:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0 \text{ or } k = n \end{cases}$$

- Based on this recursive property, what is a divide-and-conquer algorithm to determine the binomial coefficient of any n and k?

Binomial Coefficient Divide-and-Conquer

Problem: Compute the binomial coefficient

Inputs: nonnegative integers n and k, where $k \leq n$

Outputs: The binary coefficient of n and k i.e. $C(n, k)$

```
int bin (int n, int k)
{
        if (k == 0 || n == k)
                return 1;
        else
                return bin(n - 1, k - 1) + bin(n - 1, k);
}
```

Binomial Coefficient Divide-and-Conquer

This algorithm is very inefficient!

- The same instances are solved in each recursive call.
    - $bin(n - 1, k - 1)$ and $bin(n - 1, k)$ both require the result of $bin(n - 2, k - 1)$
        - $bin(n - 2, k - 1)$ is recomputed in both recursive calls.
    - Divide-and-Conquer is a poor choice when an instance is divided into two smaller instances that are almost as large as the original instance.

Binomial Coefficient Dynamic

We can use **dynamic programming** to solve this much more efficiently.
We construct a solution from the bottom-up in a two-dimensional array B, where B[i][j] contains C(i, j).

For every dynamic programming algorithm, we follow two basic steps:

1. Establish a recursive property for the problem. (Which we already did)
2. Solve an instance of the problem in a *bottom-up* fashion. In other words, *start* at the base case and work our way up to the top.

For the Binomial Coefficient problem, we compute the first row in B, then the second, etc.

Binomial Coefficient Dynamic

To calculate C(4, 2), we fill B with the following values:

- Why do we only need the values in columns 0, 1, and 2 for each row?

	0	1	2	3	4	j	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3				
4	1	4	6				
i							
n							

Binomial Coefficient Dynamic

To calculate C(4, 2), we fill B with the following values:

- Why do we only need the values in columns 0, 1, and 2 for each row?
- $k = 2$, so we will never pass a number larger than 2 for k to `bin`.
- Suppose we were to solve this problem with divide-and-conquer:

	0	1	2	3	4	j	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3				
4	1	4	6				
i							
n							

bin(n, k) = bin(n - 1, k - 1) + bin(n - 1, k);

At the top level, we have bin(4, 2) = bin(3, 1) + bin(3, 2)

- i.e. We will never need bin(3, 3)

Binomial Coefficient Dynamic

Let’s say we want to calculate C(4, 2)

- Compute row 0:
  - B[0][0] (when $k = n$ we return 1)
- Compute row 1:
  - B[1][0] = 1 (when $k = 0$ we return 1)
  - B[1][1] = 1 (when $k = n$ we return 1)
- Compute row 2:
  - B[2][0] = 1
  - B[2][1] = B[1][0] + B[1][1] = 2
  - B[2][2] = 1

	0	1	2	3	4	j	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3				
4	1	4	6				
i							
n							

Binomial Coefficient Dynamic

- Compute row 2:
  - B[2][0] = 1
  - B[2][1] = B[1][0] + B[1][1] = 2
  - B[2][2] = 1
- Compute row 3:
  - B[3][0] = 1
  - B[3][1] = B[2][0] + B[2][1] = 3
  - B[3][2] = B[2][1] + B[2][2] = 3
- Compute row 4:
  - B[4][0] = 1
  - B[4][1] = B[3][0] + B[3][1] = 4
  - B[4][2] = B[3][1] + B[3][2] = 6

	0	1	2	3	4	j	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3				
4	1	4	6				
i							
n							

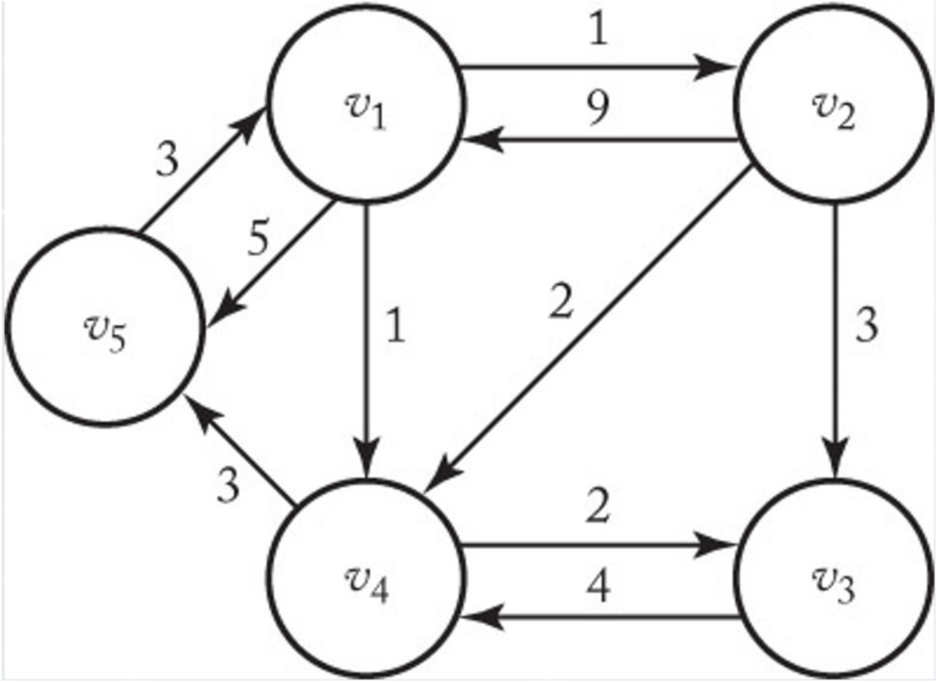
Binomial Coefficient Dynamic

```
int bin2 (int n, int k)
          index i, j;

     // two dimensional n × k array to hold calculated results
          int B[0..n][0..k];

          for (i = 0; i <= n; i++)
                    for (j = 0; j <= minimum(i, k); j++)
                              if (j == 0 || j == i)
                                        B[i][j] = 1;
                              else
                                        // retrieve previously calculated binomial
coefficients
                                        B[i][j] = B[i-1][j-1] + B[i-1][j];
          return B[n][k]
```

Floyd's Algorithm

- With Dijkstra's Algorithm, we found the shortest path from a <u>single</u> vertex to every other vertex.

- Floyd's algorithm uses dynamic programming to find the shortest path from <u>every</u> vertex to every other vertex.



Floyd’s Algorithm

- The goal is to calculate D from W
- W is the adjacency matrix for the graph on the previous slide.
- D is a two-dimensional array containing the lengths of the shortest paths from any vertex to any other vertex.
- One option: determine for each vertex the lengths of all the paths from it to every other vertex.
  - This algorithm is worse than exponential!

	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

Floyd's Algorithm

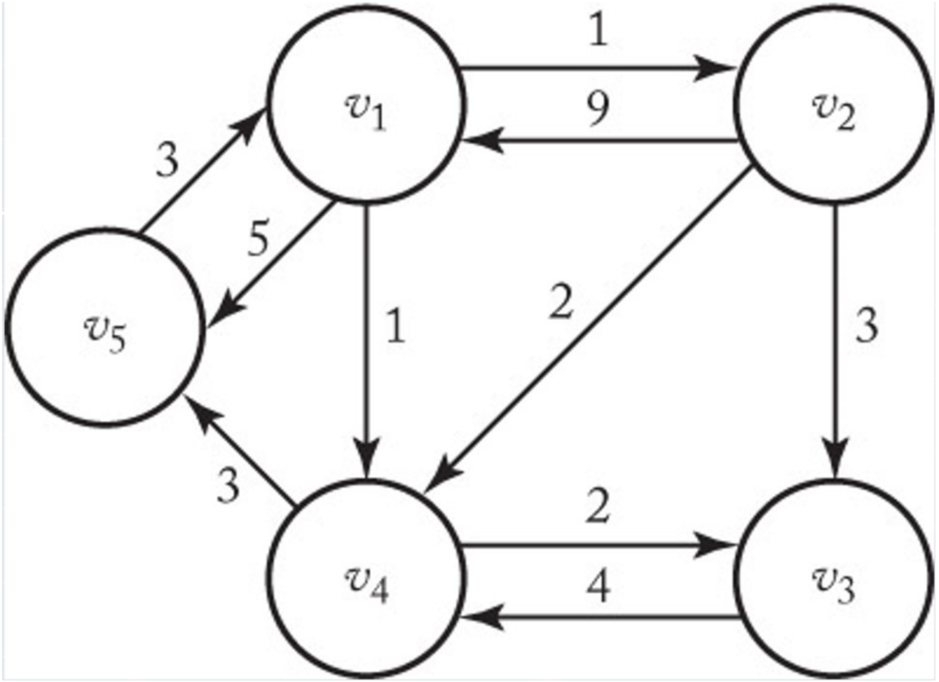
Floyd's algorithm solves this problem in cubic time.

- We create $n + 1$ arrays, where n is the # of vertices in the graph. Each array represents an intermediate step between W and D.
  - The first array is named $D^{(0)}$
  - The second array is named $D^{(1)}$
  - The kth array is named $D^{(k)}$
- $D^{(k)}[i][j]$ = the length of the shortest path from v_i to v_j, using only vertices in the set $\{ v_1, v_2, \ldots, v_k \}$ as possible intermediates.
  - i.e. $D^{(2)}$ is a two dimensional array containing the distance from every vertex in the graph to every other vertex, using only v_1 and v_2 as possible intermediates.

Floyd's Algorithm

Let's calculate the shortest path from vertex 2 to 5 using Floyd's Algorithm.

- ➢ **Note**: Floyd's algorithm calculates the path from each vertex to each vertex, but we'll trace just one for the sake of time/space.

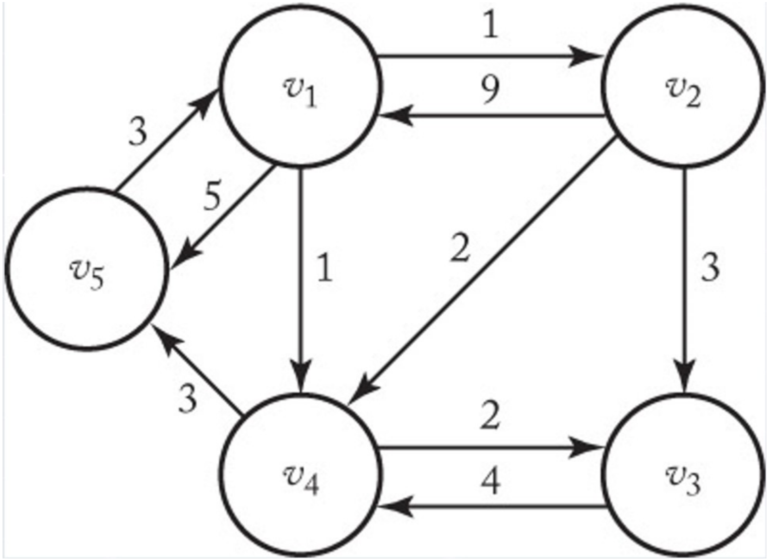


Step one: Determine the length from v_2 to v_5 using no intermediaries:

$D^{(0)}[2][5] = \text{length}[v_2, v_5] = \infty$

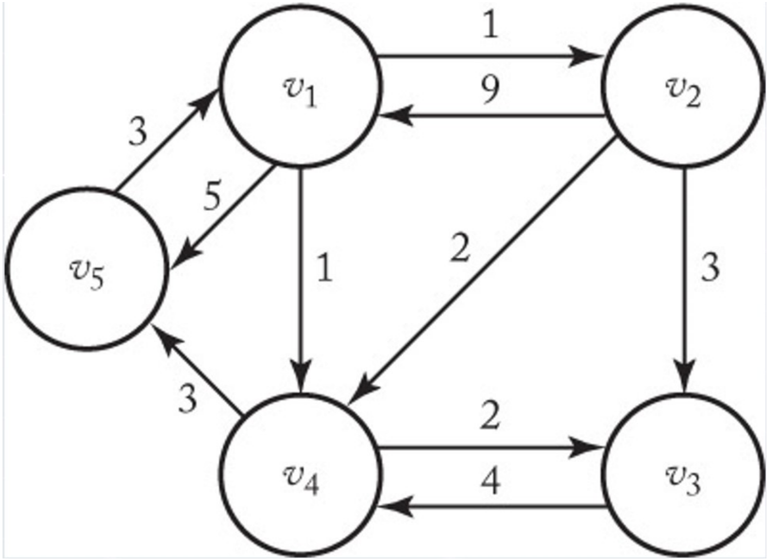
Floyd's Algorithm

- $D^{(0)}[2][5] = \text{length}[v_2, v_5] = \infty$
- $D^{(1)}[2][5] =$



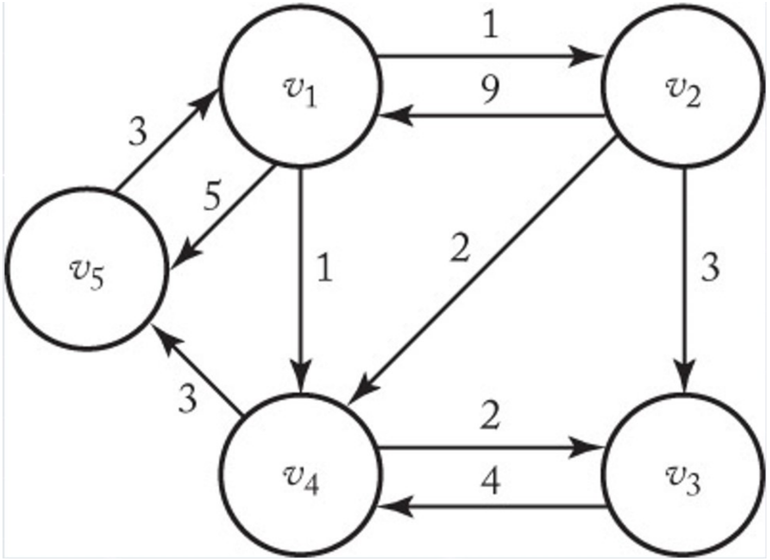
Floyd's Algorithm

- $D^{(0)}[2][5] = \text{length}[v_2, v_5] = \infty$
- $D^{(1)}[2][5] = \min(\text{length}[v_2, v_5], \text{length}[v_2, v_1, v_5]) = \min(\infty, 14) = 14$
- $D^{(2)}[2][5] =$



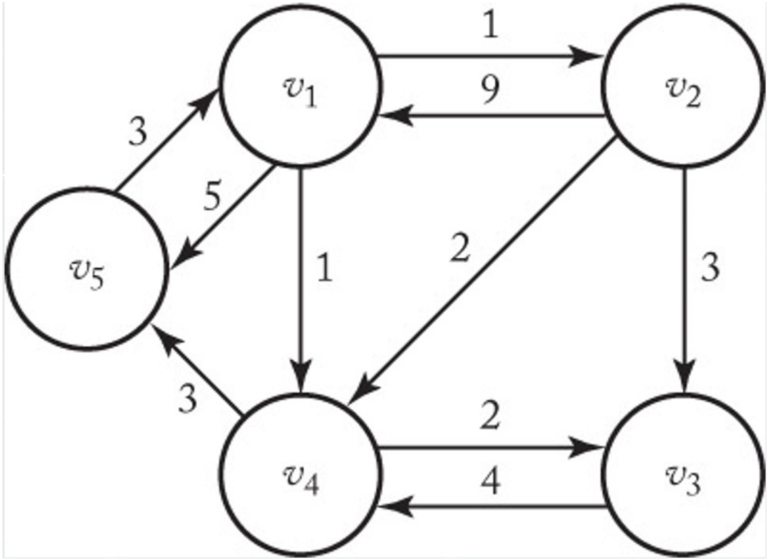
Floyd's Algorithm

- $D^{(0)}[2][5] = \text{length}[v_2, v_5] = \infty$
- $D^{(1)}[2][5] = \min(\text{length}[v_2, v_5], \text{length}[v_2, v_1, v_5]) = \min(\infty, 14) = 14$
- $D^{(2)}[2][5] = 14$ (a shortest path cannot pass through the starting vertex)
- $D^{(3)}[2][5] =$



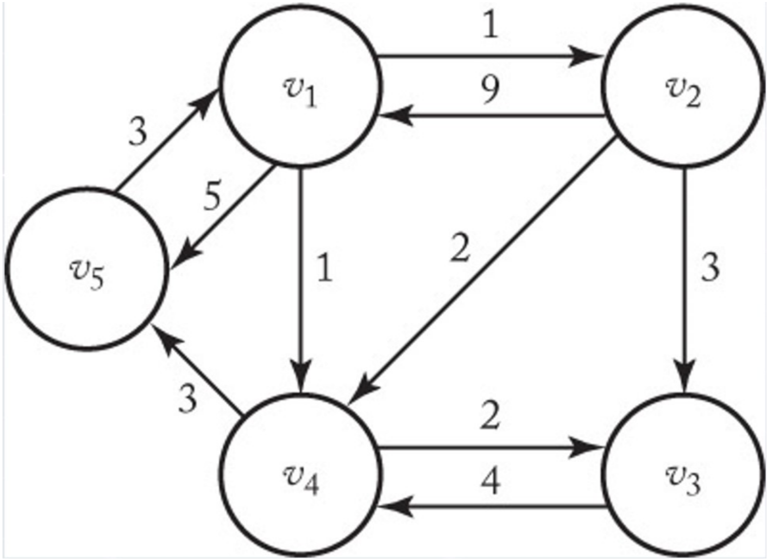
Floyd's Algorithm

- $D^{(0)}[2][5] = \text{length}[v_2, v_5] = \infty$
- $D^{(1)}[2][5] = \min(\text{length}[v_2, v_5], \text{length}[v_2, v_1, v_5]) = \min(\infty, 14) = 14$
- $D^{(2)}[2][5] = 14$ (a shortest path cannot pass through the starting vertex)
- $D^{(3)}[2][5] = 14$ (no connection to v_5 through v_3 without using v_4, which we can't use this step)
- $D^{(4)}[2][5] =$



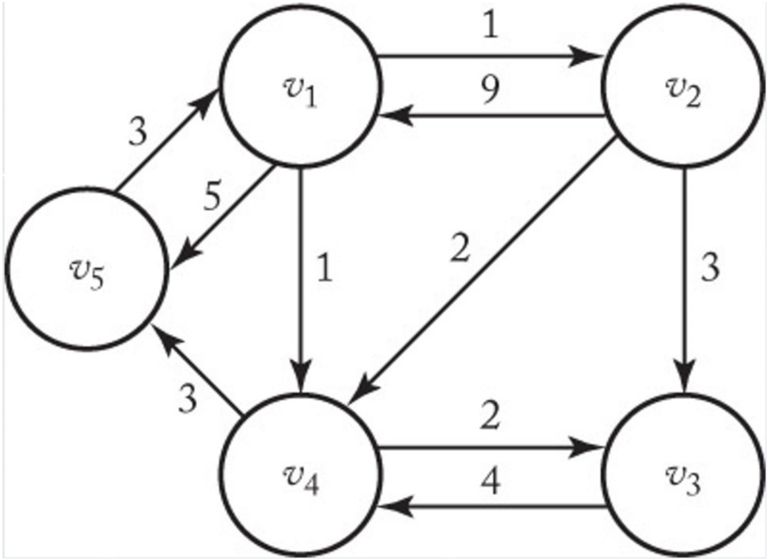
Floyd's Algorithm

- $D^{(0)}[2][5]$ = length[v_2, v_5] = ∞
- $D^{(1)}[2][5]$ = min(length[v_2, v_5], length[v_2, v_1, v_5]) = min(∞, 14) = 14
- $D^{(2)}[2][5]$ = 14 (a shortest path cannot pass through the starting vertex)
- $D^{(3)}[2][5]$ = 14 (no connection to v_5 through v_3 without using v_4, which we can't use this step)
- $D^{(4)}[2][5]$ = 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**
- $D^{(5)}[2][5]$ =



Floyd's Algorithm

- $D^{(0)}[2][5] = \text{length}[v_2, v_5] = \infty$
- $D^{(1)}[2][5] = \min(\text{length}[v_2, v_5], \text{length}[v_2, v_1, v_5]) = \min(\infty, 14) = 14$
- $D^{(2)}[2][5] = 14$ (a shortest path cannot pass through the starting vertex)
- $D^{(3)}[2][5] = 14$ (no connection to v_5 through v_3 without using v_4, which we can't use this step)
- $D^{(4)}[2][5] = \min(\text{length}[v_2, v_1, v_5], \text{length}[v_2, v_4, v_5], \text{length}[v_2, v_1, v_4, v_5], \text{length}[v_2, v_3, v_4, v_5]) = \min(14, 5, 13, 10) = 5$
- $D^{(5)}[2][5] = 5$ (a shortest path cannot pass through the ending vertex)



Floyd's Algorithm

- $D^{(n)}[i][j]$ is the length of a shortest path from v_i to v_j that is allowed to pass through all of the other vertices, so it is the length of a shortest path from v_i to v_j
- $D^{(0)}[i][j]$ is the length of a shortest path from v_i to v_j using no intermediate vertices, so it is the weight on the edge from v_i to v_j.
    - $\therefore D^{(0)} = W$ and $D^{(n)} = D$
- We need a way to obtain $D^{(n)}$ from $D^{(0)}$

To do this, we:

1. Establish a recursive property with which we can compute $D^{(k)}$ from $D^{(k-1)}$
2. Solve an instance of the problem in a bottom-up fashion by repeating the process (step 1) for $k = 1$ to n.

Floyd's Algorithm

There are two cases to consider when establishing the recursive property.

Case 1: At least <u>one</u> shortest path from v_i to v_j, using only vertices in $\{v_1, v_2, \ldots, v_k\}$ as intermediate vertices, does *not* use v_k.

In this case, what value can we set for $D^{(k)}[i][j]$?

Floyd's Algorithm

There are two cases to consider when establishing the recursive property.

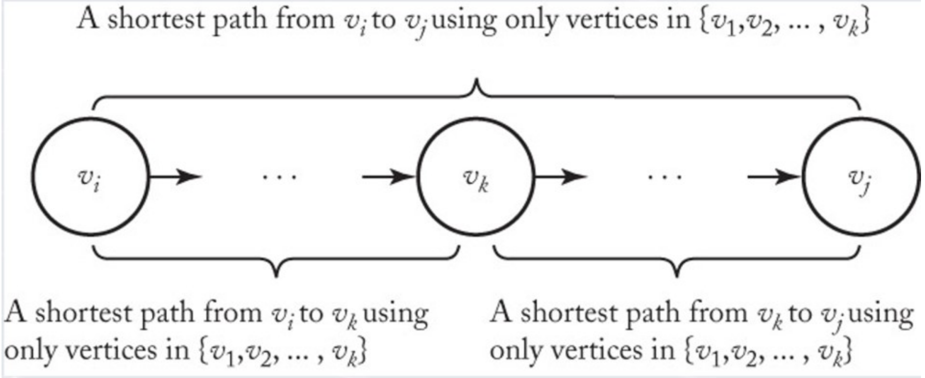
Case 1: At least <u>one</u> shortest path from v_i to v_j, using only vertices in $\{v_1, v_2, \ldots, v_k\}$ as intermediate vertices, does *not* use v_k.

- v_k is the vertex we are currently considering
- In this case, we simply choose the previously calculated path that does not use v_k.

i.e. we say $D^{(k)}[i][j] = D^{(k-1)}[i][j]$.

Floyd's Algorithm

Case 2: *All* shortest paths from v_i to v_j, using only vertices in $\{v_1, v_2, \ldots, v_k\}$ as intermediates, use v_k.
In this case, any shortest path from v_i to v_j looks like the following:



Therefore, in this case, $D^{(k)} = D^{(k-1)}[i][k] + D^{(k-1)}[k][j]$

Floyd's Algorithm

Given the following

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

Case 2: $D^{(k)} = D^{(k-1)}[i][k] + D^{(k-1)}[k][j]$

How do we define our recursive property?

Floyd's Algorithm

Given the following

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

Case 2: $D^{(k)} = D^{(k-1)}[i][k] + D^{(k-1)}[k][j]$

How do we define our recursive property?

- ➢ We take the minimum of both cases in array $k - 1$ to be the minimum in array k

- $D^{(k)}[i][j] = \mathit{minimum}(\text{Case } 1, \text{Case } 2)$:

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

Floyd’s Algorithm Step 1 Row 1

Computing row 1 of $D^{(1)}$ from $D^{(0)}$

Row 1 does not change in $D^{(1)}$

	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

$D^{(0)}$

Floyd's Algorithm Step 1 Row 2

Computing row 2 of $D^{(1)}$ from $D^{(0)}$

$D^{(1)}[2][3] = \min(D^{(0)}[2][3], D^{(0)}[2][1] + D^{(0)}[1][3])$

$= \min(3, 9 + \infty) = \mathbf{3}$

$D^{(1)}[2][4] = \min(D^{(0)}[2][4], D^{(0)}[2][1] + D^{(0)}[1][4])$

$= \min(2, 9 + 1) = \mathbf{2}$

$D^{(1)}[2][5] = \min(D^{(0)}[2][5], D^{(0)}[2][1] + D^{(0)}[1][5])$

$= \min(\infty, 9 + 5) = \mathbf{14}$

	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

$D^{(0)}$

Floyd's Algorithm Step 1 Row 3

Computing row 3 of $D^{(1)}$ from $D^{(0)}$

$$D^{(1)}[3][2] = \min(D^{(0)}[3][2], D^{(0)}[3][1] + D^{(0)}[1][2])$$
$$= \min(\infty, \infty + 1) = \infty$$

$$D^{(1)}[3][4] = \min(D^{(0)}[3][4], D^{(0)}[3][1] + D^{(0)}[1][4])$$
$$= \min(4, \infty + 1) = \mathbf{4}$$

$$D^{(1)}[3][5] = \min(D^{(0)}[3][5], D^{(0)}[3][1] + D^{(0)}[1][5])$$
$$= \min(\infty, \infty + 5) = \infty$$

Note: Since $D^{(0)}[3][1] = \infty$, no changes are made in this row.

	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

$D^{(0)}$

Floyd's Algorithm Step 1 Row 4

Computing row 4 of $D^{(1)}$ from $D^{(0)}$

Since $D^{(0)}[4][1] = \infty$, no changes are made in this row.

	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

$D^{(0)}$

Floyd's Algorithm Step 1 Row 5

Computing row 5 of $D^{(1)}$ from $D^{(0)}$

$$D^{(1)}[5][2] = \min(D^{(0)}[5][2], D^{(0)}[5][1] + D^{(0)}[1][2])$$
$$= \min(\infty, 3 + 1) = \mathbf{4}$$

$$D^{(1)}[5][3] = \min(D^{(0)}[5][3], D^{(0)}[5][1] + D^{(0)}[1][3])$$
$$= \min(\infty, 3 + \infty) = \infty$$

$$D^{(1)}[5][4] = \min(D^{(0)}[5][4], D^{(0)}[5][1] + D^{(0)}[1][4])$$
$$= \min(\infty, 3 + 1) = \mathbf{4}$$

	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

$D^{(0)}$

Floyd's Algorithm End of Step 1

	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

$D^{(0)}$

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	**14**
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	**4**	∞	**4**	0

$D^{(1)}$

After applying the changes from $D^{(0)}$ to $D^{(1)}$ we end up with the array on the right.

Floyd's Algorithm Step 2 Row 1

$D^{(2)}[1][3] = \min(D^{(1)}[1][3], D^{(1)}[1][2] + D^{(1)}[2][3])$

$= \min(\infty, 1 + 3) = \mathbf{4}$

$D^{(2)}[1][4] = \min(D^{(1)}[1][4], D^{(1)}[1][2] + D^{(1)}[2][4])$

$= \min(1, 1 + 2) = \mathbf{3}$

$D^{(2)}[1][5] = \min(D^{(1)}[1][5], D^{(1)}[1][2] + D^{(1)}[2][5])$

$= \min(5, 1 + 14) = \mathbf{5}$

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	∞	4	0

$D^{(1)}$

Floyd's Algorithm Step 2 Row 2

Row 2 and Column 2 do not change in $D^{(2)}$

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	∞	4	0

$D^{(1)}$

Floyd's Algorithm Step 2 Rows 3 and 4

Since $D^{(1)}[3][2] = \infty$, no changes are made in this row.

Since $D^{(1)}[4][2] = \infty$, no changes are made in this row.

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	∞	4	0

$D^{(1)}$

Floyd's Algorithm Step 2 Row 5

$$D^{(2)}[5][1] = \min(D^{(1)}[5][1],\ D^{(1)}[5][2] + D^{(1)}[2][1])$$
$$= \min(3,\ 4 + 9) = \mathbf{3}$$

$$D^{(2)}[5][3] = \min(D^{(1)}[5][3],\ D^{(1)}[5][2] + D^{(1)}[2][3])$$
$$= \min(\infty,\ 4 + 3) = \mathbf{7}$$

$$D^{(2)}[5][4] = \min(D^{(1)}[5][4],\ D^{(1)}[5][2] + D^{(1)}[2][4])$$
$$= \min(4,\ 4 + 2) = \mathbf{4}$$

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	∞	4	0

$D^{(1)}$

Floyd's Algorithm End of Step 2

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	∞	4	0

$D^{(1)}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(2)}$

We have finished calculating the array $D^{(1)}$

Floyd's Algorithm Step 3 Row 1

$D^{(3)}[1][2] = \min(D^{(2)}[1][2], D^{(2)}[1][3] + D^{(2)}[3][2])$
$= \min(1, 4 + \infty) = \mathbf{1}$

$D^{(3)}[1][4] = \min(D^{(2)}[1][4], D^{(2)}[1][3] + D^{(2)}[3][4])$
$= \min(1, 4 + 4) = \mathbf{1}$

$D^{(3)}[1][5] = \min(D^{(2)}[1][5], D^{(2)}[1][3] + D^{(2)}[3][5])$
$= \min(5, 4 + \infty) = \mathbf{5}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(2)}$

Floyd’s Algorithm Step 3 Row 2

$D^{(3)}[2][1] = \min(D^{(2)}[2][1], D^{(2)}[2][3] + D^{(2)}[3][1])$
$= \min(9, 3 + \infty) = \mathbf{9}$

$D^{(3)}[2][4] = \min(D^{(2)}[2][4], D^{(2)}[2][3] + D^{(2)}[3][4])$
$= \min(2, 3 + 4) = \mathbf{2}$

$D^{(3)}[2][5] = \min(D^{(2)}[2][5], D^{(2)}[2][3] + D^{(2)}[3][5])$
$= \min(14, 3 + \infty) = \mathbf{14}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(2)}$

Floyd’s Algorithm Step 3 Row 3

Row 3 and Column 3 do not change in $D^{(3)}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(2)}$

Floyd's Algorithm Step 3 Row 4

$D^{(3)}[4][1] = \min(D^{(2)}[4][1], D^{(2)}[4][3] + D^{(2)}[3][1])$
$= \min(\infty, 2 + \infty) = \infty$

$D^{(3)}[4][2] = \min(D^{(2)}[4][2], D^{(2)}[4][3] + D^{(2)}[3][2])$
$= \min(\infty, 2 + \infty) = \infty$

$D^{(3)}[4][5] = \min(D^{(2)}[4][5], D^{(2)}[4][3] + D^{(2)}[3][5])$
$= \min(3, 2 + \infty) = \mathbf{3}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(2)}$

Floyd's Algorithm Step 3 Row 5

Since $D^{(2)}[3][5] = \infty$, no changes are made in this row.

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(2)}$

Floyd's Algorithm End of Step 3

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(2)}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(3)}$

No changes were made between $D^{(2)}$ and $D^{(3)}$

Floyd's Algorithm Step 4 Row 1

$D^{(4)}[1][2] = \min(D^{(3)}[1][2], D^{(3)}[1][3] + D^{(3)}[3][2])$
$= \min(1, 4 + \infty) = \mathbf{1}$

$D^{(4)}[1][3] = \min(D^{(3)}[1][3], D^{(3)}[1][4] + D^{(3)}[4][3])$
$= \min(4, 1 + 2) = \mathbf{3}$

$D^{(4)}[1][5] = \min(D^{(3)}[1][5], D^{(3)}[1][4] + D^{(3)}[4][5])$
$= \min(5, 1 + 3) = \mathbf{4}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(3)}$

Floyd's Algorithm Step 4 Row 2

$D^{(4)}[2][1] = \min(D^{(3)}[2][1], D^{(3)}[2][4] + D^{(3)}[4][1])$
$= \min(9, 2 + \infty) = \mathbf{9}$

$D^{(4)}[2][3] = \min(D^{(3)}[2][3], D^{(3)}[2][4] + D^{(3)}[4][3])$
$= \min(3, 2 + 2) = \mathbf{3}$

$D^{(4)}[2][5] = \min(D^{(3)}[2][5], D^{(3)}[2][4] + D^{(3)}[4][5])$
$= \min(14, 2 + 3) = \mathbf{5}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(3)}$

Floyd's Algorithm Step 4 Row 3

$$D^{(4)}[3][1] = \min(D^{(3)}[3][1], D^{(3)}[3][4] + D^{(3)}[4][1])$$
$$= \min(\infty, 4 + \infty) = \infty$$

$$D^{(4)}[3][2] = \min(D^{(3)}[3][2], D^{(3)}[3][4] + D^{(3)}[4][2])$$
$$= \min(\infty, 4 + \infty) = \infty$$

$$D^{(4)}[3][5] = \min(D^{(3)}[3][5], D^{(3)}[3][4] + D^{(3)}[4][5])$$
$$= \min(\infty, 4 + 3) = \mathbf{7}$$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(3)}$

Floyd's Algorithm Step 4 Row 4

Row 4 and Column 4 do not change in $D^{(4)}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(3)}$

Floyd's Algorithm Step 4 Row 5

$D^{(4)}[5][1] = \min(D^{(3)}[5][1], D^{(3)}[5][4] + D^{(3)}[4][1])$
$= \min(3, 4 + \infty) = \mathbf{3}$

$D^{(4)}[5][2] = \min(D^{(3)}[5][2], D^{(3)}[5][4] + D^{(3)}[4][2])$
$= \min(4, 4 + \infty) = \mathbf{4}$

$D^{(4)}[5][3] = \min(D^{(3)}[5][3], D^{(3)}[5][4] + D^{(3)}[4][3])$
$= \min(\infty, 4 + 2) = \mathbf{6}$

	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(3)}$

Floyd’s Algorithm End of Step 4

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(3)}$

→

	1	2	3	4	5
1	0	1	**3**	1	**4**
2	9	0	3	2	**5**
3	∞	∞	0	4	**7**
4	∞	∞	2	0	3
5	3	4	**6**	4	0

$D^{(4)}$

Floyd's Algorithm Step 5 Row 1

$D^{(5)}[1][2] = \min(D^{(4)}[1][2], D^{(4)}[1][5] + D^{(4)}[5][2])$
$= \min(1, 4 + 4) = \mathbf{1}$

$D^{(5)}[1][3] = \min(D^{(4)}[1][3], D^{(4)}[1][5] + D^{(4)}[5][3])$
$= \min(3, 4 + 6) = \mathbf{3}$

$D^{(5)}[1][4] = \min(D^{(4)}[1][4], D^{(4)}[1][5] + D^{(4)}[5][4])$
$= \min(1, 4 + 4) = \mathbf{1}$

	1	2	3	4	5
1	0	1	3	1	4
2	9	0	3	2	5
3	∞	∞	0	4	7
4	∞	∞	2	0	3
5	3	4	6	4	0

$D^{(4)}$

Floyd's Algorithm Step 5 Row 2

$D^{(5)}[2][1] = \min(D^{(4)}[2][1], D^{(4)}[2][5] + D^{(4)}[5][1])$

$= \min(9, 5 + 3) = \mathbf{8}$

$D^{(5)}[2][3] = \min(D^{(4)}[2][3], D^{(4)}[2][5] + D^{(4)}[5][3])$

$= \min(3, 5 + 6) = \mathbf{3}$

$D^{(5)}[2][4] = \min(D^{(4)}[2][4], D^{(4)}[2][5] + D^{(4)}[5][4])$

$= \min(2, 5 + 4) = \mathbf{2}$

	1	2	3	4	5
1	0	1	3	1	4
2	9	0	3	2	5
3	∞	∞	0	4	7
4	∞	∞	2	0	3
5	3	4	6	4	0

$D^{(4)}$

Floyd's Algorithm Step 5 Row 3

$D^{(5)}[3][1] = \min(D^{(4)}[3][1], D^{(4)}[3][5] + D^{(4)}[5][1])$

$= \min(\infty, 7 + 3) = \mathbf{10}$

$D^{(5)}[3][2] = \min(D^{(4)}[3][2], D^{(4)}[3][5] + D^{(4)}[5][2])$

$= \min(\infty, 7 + 4) = \mathbf{11}$

$D^{(5)}[3][4] = \min(D^{(4)}[3][4], D^{(4)}[3][5] + D^{(4)}[5][4])$

$= \min(4, 7 + 4) = \mathbf{4}$

	1	2	3	4	5
1	0	1	3	1	4
2	9	0	3	2	5
3	∞	∞	0	4	7
4	∞	∞	2	0	3
5	3	4	6	4	0

$D^{(4)}$

Floyd's Algorithm Step 5 Row 4

$D^{(5)}[4][1] = \min(D^{(4)}[4][1], D^{(4)}[4][5] + D^{(4)}[5][1])$
$= \min(\infty, 3 + 3) = \mathbf{6}$

$D^{(5)}[4][2] = \min(D^{(4)}[4][2], D^{(4)}[4][5] + D^{(4)}[5][2])$
$= \min(\infty, 3 + 4) = \mathbf{7}$

$D^{(5)}[4][3] = \min(D^{(4)}[4][3], D^{(4)}[4][5] + D^{(4)}[5][3])$
$= \min(2, 3 + 7) = \mathbf{2}$

	1	2	3	4	5
1	0	1	3	1	4
2	9	0	3	2	5
3	∞	∞	0	4	7
4	∞	∞	2	0	3
5	3	4	6	4	0

$D^{(4)}$

Floyd's Algorithm End of Step 5

	1	2	3	4	5
1	0	1	3	1	4
2	9	0	3	2	5
3	∞	∞	0	4	7
4	∞	∞	2	0	3
5	3	4	7	4	0

$D^{(4)}$

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

$D^{(5)}$

Floyd's Algorithm

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

- Each iteration of the *k* loop calculates the next D array.
  - i.e. when $k = 2$, $D^{(2)}$ is computed.
- Why can we get away with only using one array?

Floyd's Algorithm

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

- Each iteration of the k loop calculates the next D array.
  - i.e. when $k = 2$, $D^{(2)}$ is computed.
- Why can we get away with only using one array? The values in the kth row and column are not changed during the kth iteration of the loop.
  - i.e. when $j = k$: $D[i][k] = \min(D[i][k], D[i][k] + D[k][k])$ is clearly $D[i][k]$
- In the kth iteration, $D[i][j]$ is computed from only its own value and values in the kth row and column. Since the values in k didn't change this iteration, they are still the correct values.

Floyd's Algorithm

- The version of Floyd's algorithm we have discussed simply finds the length of a shortest path from each vertex to every other vertex.

- With a slight modification, we can also compute and return what each path is.

Problem: Same as before, except shortest paths are also created.
Additional Outputs: array P. Both its rows and columns indexed from 1 to n, where:

- $P[i][j]$ = highest index of an intermediate vertex on the shortest path from v_i to v_j, if at least one intermediate vertex exists. 0 if there is no intermediate vertex.

Floyd's Algorithm

```
int floyd2(int n, const int W[][], int D[][], index P[][])
{
        for (index i = 1; i <= n; i++)
                for (index j = 1; j <= n; j++)
                        P[i][j] = 0;
        D = W
        for (k = 1; k <= n; k++)
                for (i = 1; i <=n; i++)
                        for (j = 1; j <= n; j++)
                                if (D[i][k] + D[k][j] < D[i][j])
                                        P[i][j] = k; // add intermediate vertex
k between i and j on path
                                        D[i][j] = D[i][k] + D[k][j]
}
```

Whenever we find a vertex `k` that is on the shortest path as an intermediary between `i` and `j`, we update `P[i][j] = k`

Floyd’s Algorithm

	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	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

Let’s say we calculate P on the right from W on the left.

How do we find the shortest path from 2 to 5?

Floyd’s Algorithm

	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	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

Let’s say we calculate P on the right from W on the left.

How do we find the shortest path from 2 to 5?

- P[2][5] = 4, which means $<v_2, v_4>$ is in the path.
- P[4][5] = 0, which means $<v_4, v_5>$ is in the path.
    - Final Path: $\{ <v_2, v_4>, <v_4, v_5> \}$

Floyd’s Algorithm

This procedure prints out all the intermediate vertices between *q* and *r*

```
void path(index q, r)
{
        if (P[q][r] != 0)
                path(q, P[q][r]);
                cout << "v" << P[q][r];
                path(P[q][r], r)
}
```

- Call path(2, 5). P[2][5] = 4.
  - Call path(2, 4). P[2][4] = 0, do nothing
- Print out “v4”
  - Call path(4, 5). P[4][5] = 0, do nothing

In-Class Exercise

1. Use Floyd's Algorithm to compute D from the following array W. Show each intermediate array.

	1	2	3	4
1	0	1	5	10
2	5	0	2	6
3	4	9	0	5
4	7	2	1	0