



香港中文大學 (深圳)
The Chinese University of Hong Kong

CSC3100 Data Structures

Lecture 23: Graph shortest path

Yixiang Fang
School of Data Science (SDS)
The Chinese University of Hong Kong, Shenzhen



Outline

- ▶ We focus on weighted graphs
- ▶ Graphs with non-negative weights
 - Single-Source Shortest Path: Dijkstra's algorithm
- ▶ All-Pair Shortest Path: Floyd's algorithm
- ▶ Graphs with negative weights
 - Bellman-Ford algorithm



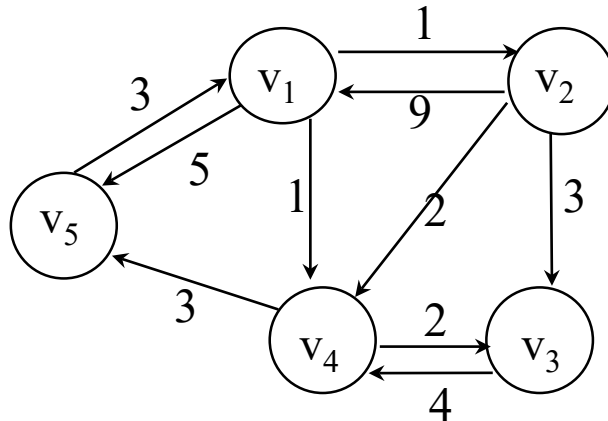
All pairs shortest path

- **A representation:** a weight matrix where

$W(i, j) = 0$ if $i = j$

$W(i, j) = \infty$ if there is no edge between i and j

$W(i, j)$ = "weight of edge"



	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

- **Problem:** find the shortest distance/path between every pair of vertices of a graph



A straightforward method

- ▶ A naïve method is to run a single-source shortest path algorithm for each vertex
 - Run Dijkstra's algorithm $|V|$ times
 - Dijkstra's algorithm's time complexity: $O(|E| \times \lg|V|)$
 - Total time cost: $O(|V| \times |E| \times \lg|V|)$
- ▶ Floyd's algorithm
 - Total time cost: $O(|V|^3)$
 - For dense subgraphs, Floyd's algorithm is faster
 - It is easier to implement



Floyd's algorithm

- ▶ We have shown principle of optimality applies to shortest path problems
- ▶ How can we define the shortest distance d_{ij} in terms of "smaller" problems?
- ▶ Main idea of Floyd's algorithm
 - One way is to restrict the paths to only include vertices from a restricted subset
 - Initially, the subset is empty
 - Then, it is incrementally increased until it includes all the vertices



The subproblems

- ▶ Let $D^{(k)}[i,j]$ = weight of a shortest path from v_i to v_j using only vertices from $\{v_1, v_2, \dots, v_k\}$ as intermediate vertices in the path
 - $D^{(0)} = W$
 - $D^{(n)} = D$ which is the goal matrix
- ▶ How do we compute $D^{(k)}$ from $D^{(k-1)}$?

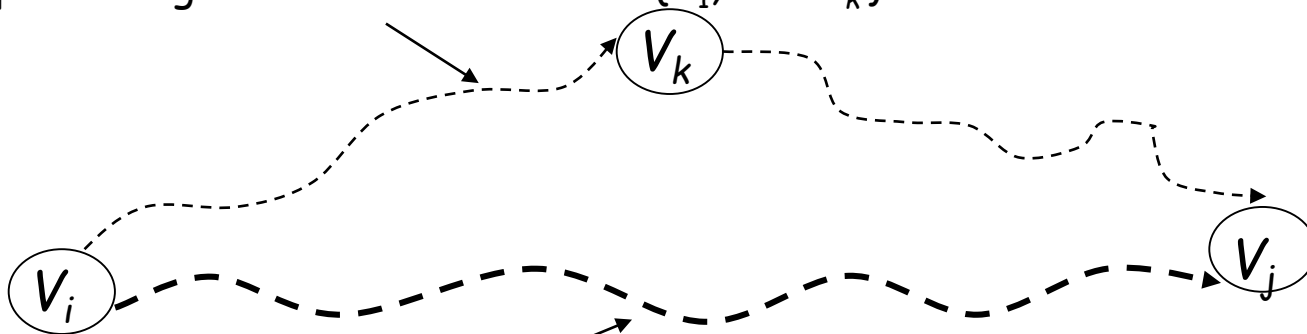


The recursive definition:

Case 1: A shortest path from v_i to v_j restricted to using only vertices from $\{v_1, v_2, \dots, v_k\}$ as intermediate vertices does not use v_k Then $D^{(k)}[i,j] = D^{(k-1)}[i,j]$

Case 2: A shortest path from v_i to v_j restricted to using only vertices from $\{v_1, v_2, \dots, v_k\}$ as intermediate vertices does use v_k Then $D^{(k)}[i,j] = D^{(k-1)}[i,k] + D^{(k-1)}[k,j]$

Shortest path using intermediate vertices $\{V_1, \dots, V_k\}$



Shortest path using intermediate vertices $\{V_1, \dots, V_{k-1}\}$



The recursive definition

► Since

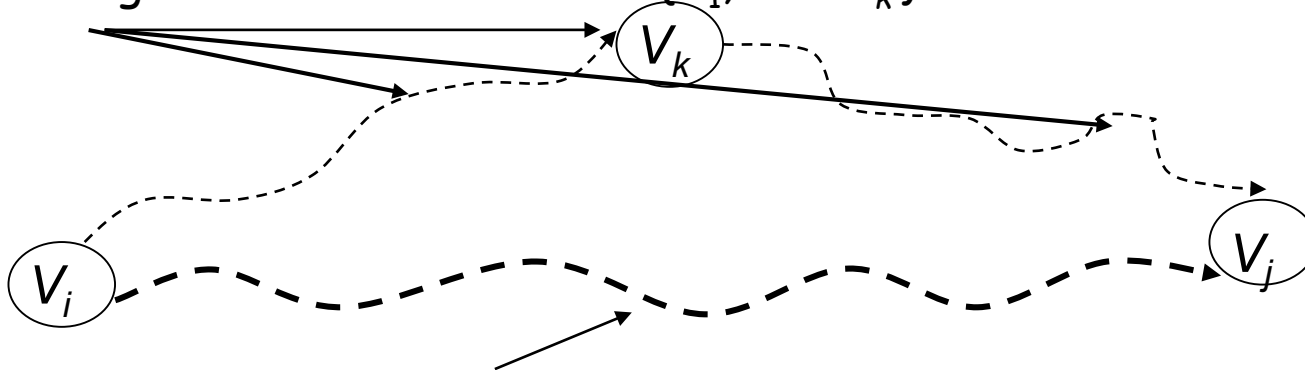
$$D^{(k)}[i,j] = D^{(k-1)}[i,j] \text{ or}$$

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

We conclude:

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

Shortest path using intermediate vertices $\{V_1, \dots, V_k\}$



Shortest Path using intermediate vertices $\{V_1, \dots, V_{k-1}\}$



The pointer array P

- ▶ How to recover the shortest paths?
 - We can use a pointer array P , which initially contains 0
 - Each time that a shorter path from i to j is found, the k that provided the minimum distance is saved
 - To print the intermediate nodes on the shortest path by a recursive procedure, which print the shortest paths from i and k , and from k to j



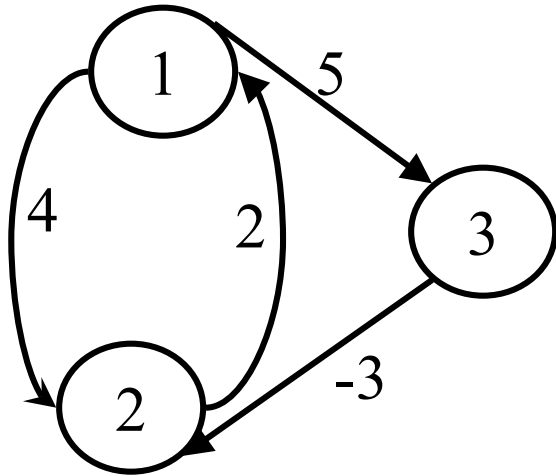
Floyd's algorithm using $n+1$ D matrices

Floyd//Computes shortest distance between all pairs of
//nodes, and saves P to enable finding shortest paths

1. $D^0 \leftarrow W$ // initialize D array to $W[]$
2. $P \leftarrow 0$ // initialize P array to $[0]$
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. if ($D^{k-1}[i, j] > D^{k-1}[i, k] + D^{k-1}[k, j]$)
7. then $D^k[i, j] \leftarrow D^{k-1}[i, k] + D^{k-1}[k, j]$
8. $P[i, j] \leftarrow k$;
9. else $D^k[i, j] \leftarrow D^{k-1}[i, j]$



Example



$$W = D^0 =$$

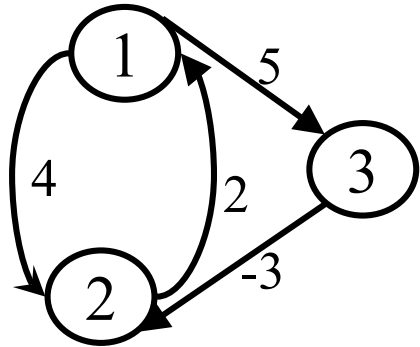
	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

$$P =$$

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0



Example


$$D^0 =$$

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

$k = 1$

Vertex 1 can be
intermediate node

$$D^1 =$$

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

$$\begin{aligned} D^1[2,3] &= \min(D^0[2,3], D^0[2,1]+D^0[1,3]) \\ &= \min(\infty, 7) \\ &= 7 \end{aligned}$$

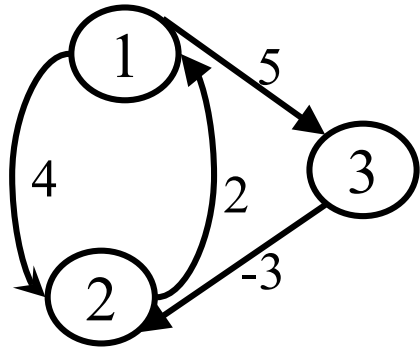
$$P =$$

	1	2	3
1	0	0	0
2	0	0	1
3	0	0	0

$$\begin{aligned} D^1[3,2] &= \min(D^0[3,2], D^0[3,1]+D^0[1,2]) \\ &= \min(-3, \infty) \\ &= -3 \end{aligned}$$



Example



$D^1 =$

1	2	3
0	4	5
2	0	7
3	∞	-3

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

$k = 2$

Vertices 1, 2 can be intermediate

$D^2 =$

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

$$\begin{aligned}
 D^2[1,3] &= \min(D^1[1,3], D^1[1,2]+D^1[2,3]) \\
 &= \min(5, 4+7) \\
 &= 5
 \end{aligned}$$

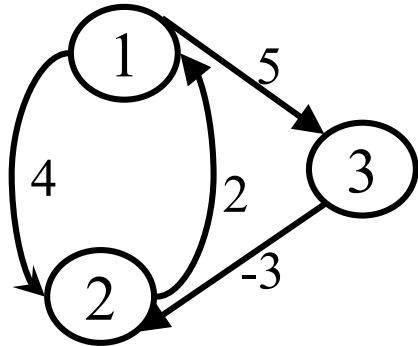
$P =$

	1	2	3
1	0	0	0
2	0	0	1
3	2	0	0

$$\begin{aligned}
 D^2[3,1] &= \min(D^1[3,1], D^1[3,2]+D^1[2,1]) \\
 &= \min(\infty, -3+2) \\
 &= -1
 \end{aligned}$$



Example



$D^2 =$

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

$$k = 3$$

Vertices 1, 2, 3 can be intermediate

$D^3 =$

	1	2	3
1	0	2	5
2	2	0	7
3	-1	-3	0

$$\begin{aligned} D^3[1,2] &= \min(D^2[1,2], D^2[1,3] + D^2[3,2]) \\ &= \min(4, 5 + (-3)) \\ &= 2 \end{aligned}$$

$P =$

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

$$\begin{aligned} D^3[2,1] &= \min(D^2[2,1], D^2[2,3] + D^2[3,1]) \\ &= \min(2, 7 + (-1)) \\ &= 2 \end{aligned}$$



Floyd's algorithm using 2 D matrices

Floyd

1. $D \leftarrow W$ // initialize D array to $W[]$
2. $P \leftarrow 0$ // initialize P array to $[0]$
3. for $k \leftarrow 1$ to n
 // Computing D' from D
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. if $(D[i, j] > D[i, k] + D[k, j])$
7. then $D'[i, j] \leftarrow D[i, k] + D[k, j]$
8. $P[i, j] \leftarrow k;$
9. else $D'[i, j] \leftarrow D[i, j]$
10. Move D' to D



Can we use only one D matrix?

- ▶ $D[i,j]$ depends only on elements in the k th column and row of the distance matrix
- ▶ We will show that the k th row and the k th column of the distance matrix are unchanged when D^k is computed
- ▶ This means D can be calculated *in-place*



The main diagonal values

- ▶ Before we show that k th row and column of D remain unchanged we show that the main diagonal remains 0
- ▶
$$\begin{aligned} D^{(k)}[j,j] &= \min\{ D^{(k-1)}[j,j], D^{(k-1)}[j,k] + D^{(k-1)}[k,j] \} \\ &= \min\{ 0, D^{(k-1)}[j,k] + D^{(k-1)}[k,j] \} \\ &= 0 \end{aligned}$$
- ▶ Based on which assumption?



The k th column

- ▶ k th column of D^k is equal to the k th column of D^{k-1}
- ▶ *Intuitively true* - a path from i to k will not become shorter by adding k to the allowed subset of intermediate vertices
- ▶ For all i , $D^{(k)}[i,k] =$
 - $= \min\{ D^{(k-1)}[i,k], D^{(k-1)}[i,k] + D^{(k-1)}[k,k] \}$
 - $= \min\{ D^{(k-1)}[i,k], D^{(k-1)}[i,k] + 0 \}$
 - $= D^{(k-1)}[i,k]$



The k th row

- ▶ k th row of D^k is equal to the k th row of D^{k-1}

$$\begin{aligned}\text{For all } j, D^{(k)}[k,j] &= \\ &= \min\{ D^{(k-1)}[k,j], D^{(k-1)}[k,k] + D^{(k-1)}[k,j] \} \\ &= \min\{ D^{(k-1)}[k,j], 0 + D^{(k-1)}[k,j] \} \\ &= D^{(k-1)}[k,j]\end{aligned}$$



Question

- ▶ Can we claim that D^k equals to D^{k-1} , D^{k-2} ?
 - No, we can only claim that because
 - The 1-st row and 1-st column of D^1 equal to the 1-st row and 1-st column of D^0 , respectively
 - The 2-nd row and 2-nd column of D^2 equal to the 2-nd row and 2-nd column of D^1 , respectively
 -



Floyd's algorithm using a single D

Floyd

```
1.  $D \leftarrow W$  // initialize  $D$  array to  $W[ ]$ 
2.  $P \leftarrow 0$  // initialize  $P$  array to  $[0]$ 
3. for  $k \leftarrow 1$  to  $n$ 
4.   do for  $i \leftarrow 1$  to  $n$ 
5.     do for  $j \leftarrow 1$  to  $n$ 
6.       if ( $D[i, j] > D[i, k] + D[k, j]$ )
7.         then  $D[i, j] \leftarrow D[i, k] + D[k, j]$ 
8.            $P[i, j] \leftarrow k;$ 
```

} $O(|V|^3)$

Total time cost: $O(|V|^3)$



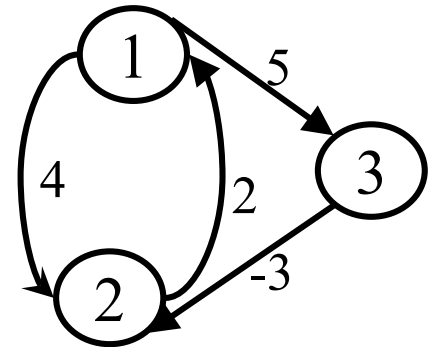
Printing intermediate nodes on shortest path from q to r

```
path(index  $q$ ,  $r$ )  
  if ( $P[q, r] \neq 0$ )  
    path( $q$ ,  $P[q, r]$ )  
    println( "v" +  $P[q, r]$ )  
    path( $P[q, r]$ ,  $r$ )  
    return;  
  //no intermediate nodes  
  else return
```

Before calling path check $D[q, r] < \infty$,
and print node q , after the call to
path print node r

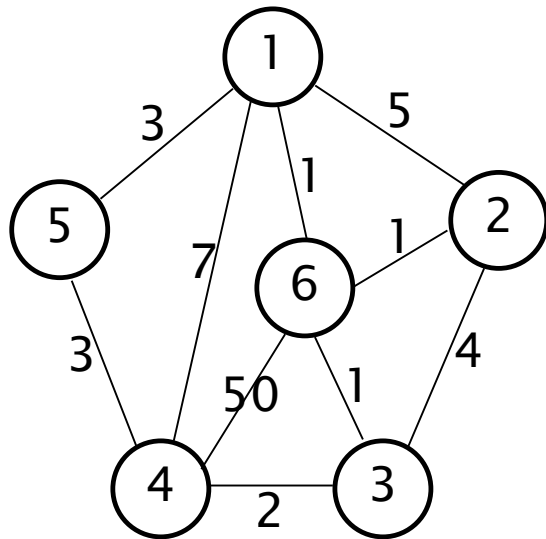
$P =$

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0





Example

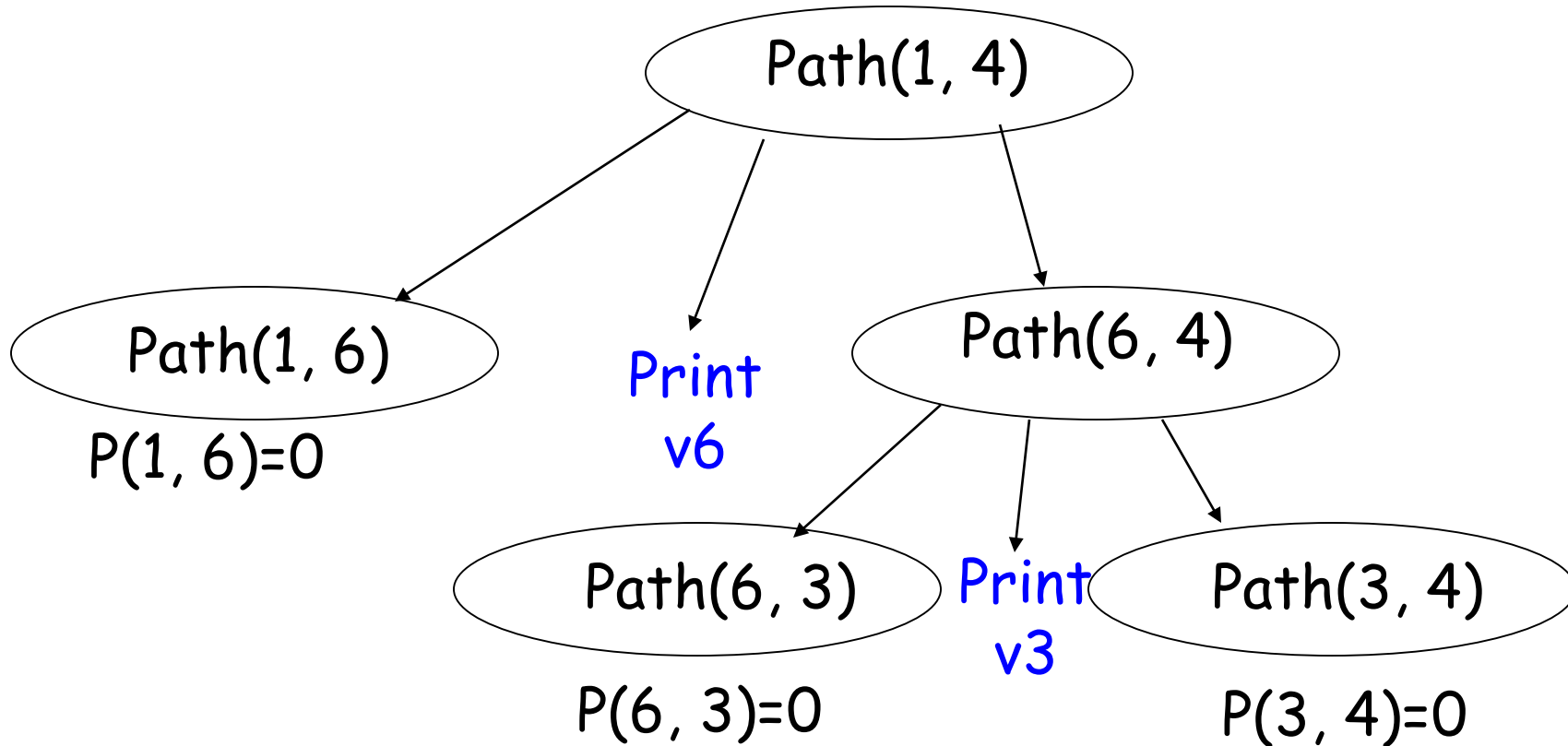


	1	2	3	4	5	6
1	0	2(6)	2(6)	4(6)	3	1
2	2(6)	0	2(6)	4(6)	5(6)	1
$D^6 = 3$	2(6)	2(6)	0	2	5(4)	1
4	4(6)	4(6)	2	0	3	3(3)
5	3	5(6)	5(4)	3	0	4(1)
6	1	1	1	3(3)	4(1)	0

The values in parenthesis are the non-zero P values



The call tree for Path(1, 4)



The intermediate nodes on the shortest path from v_1 to v_4 are v_6 , v_3 , so the shortest path is v_1, v_6, v_3, v_4 .



Recommended reading

- ▶ Reading this week
 - Textbook Chapters 24-25
- ▶ Next lecture
 - Some java data structures