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

CSC3100 Data Structures
Lecture 18: Graph shortest path

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

Outline

- Single-Source Shortest Path Algorithm
  - Non-negative weights: Dijkstra's algorithm
  - Non-negative and negative weights: Bellman-Ford algorithm

- All-Pair Shortest Path Algorithm
  - Floyd's algorithm

Bellman-Ford Algorithm

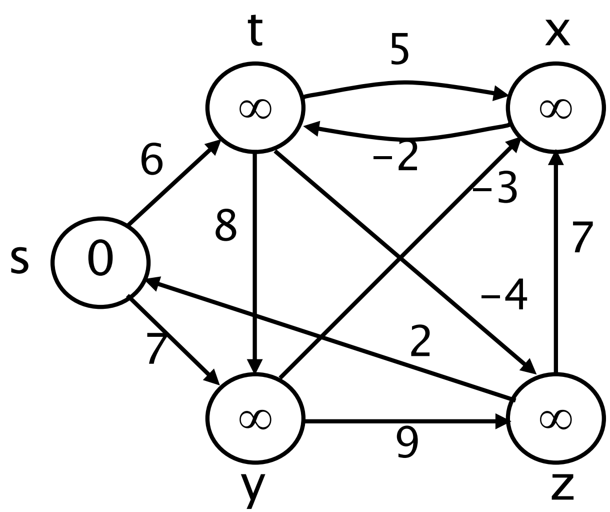
- Single-source shortest path problem
  - Computes $\delta(s, v)$ and $p[v]$ for all $v \in V$

- Allows negative edge weights - can detect negative cycles
  - Returns TRUE if no negative-weight cycles are reachable from the source s
  - Returns FALSE otherwise $\Rightarrow$ no solution exists

Bellman-Ford Algorithm (cont'd)

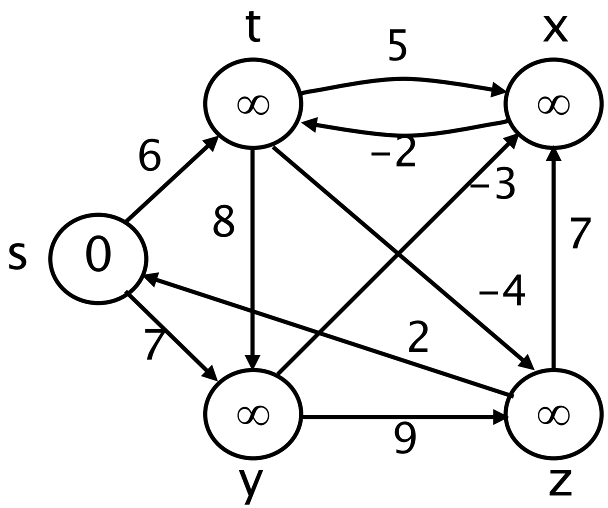
- Idea:
  - Each edge is relaxed |V|-1 times by making |V|-1 passes over the whole edge set
  - Any path will contain at most |V|-1 edges

Edge order: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

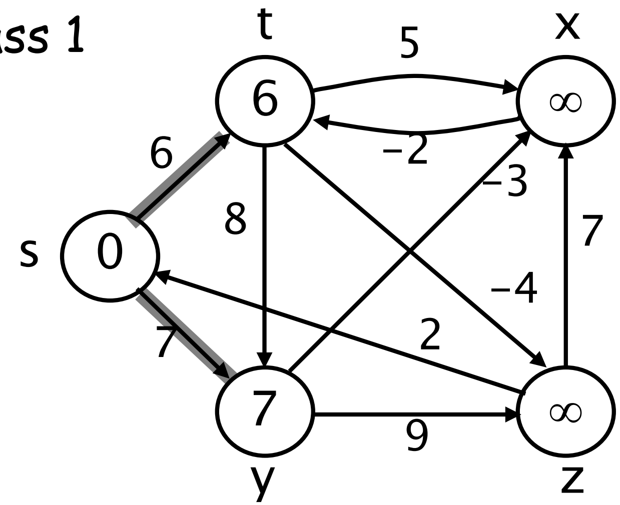


Relaxation:
If d[v] > d[u] + w(u, v)
⇒ d[v] = d[u] + w(u,v)

BELLMAN-FORD(V, E, w, s)



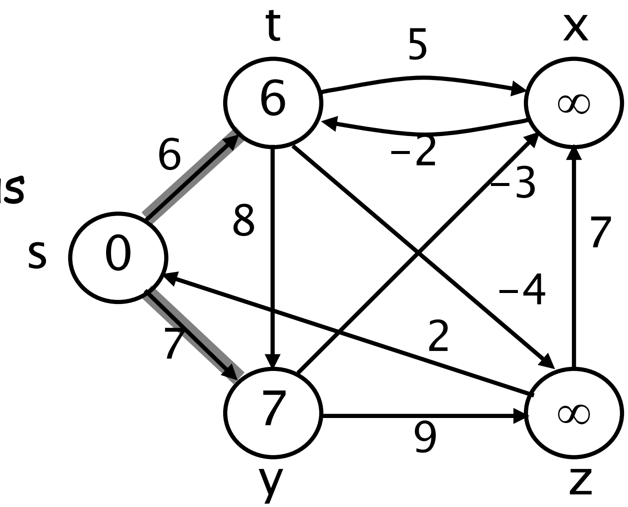
Pass 1



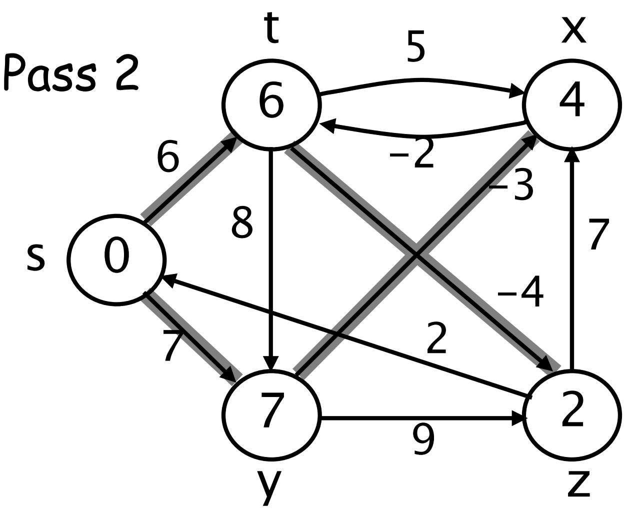
Edge order: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

Example

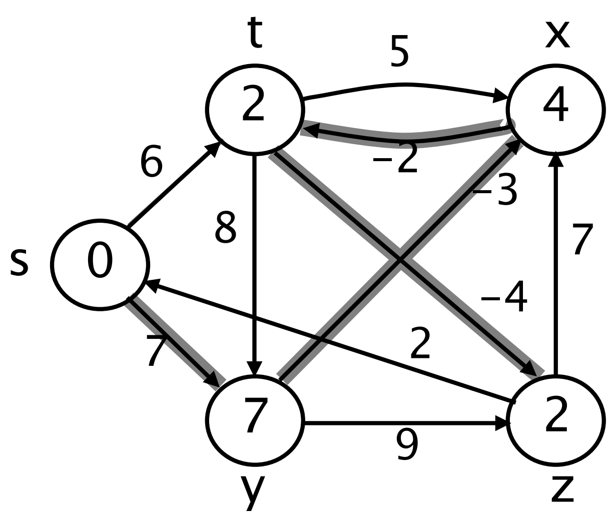
Pass 1 (from previous slide)



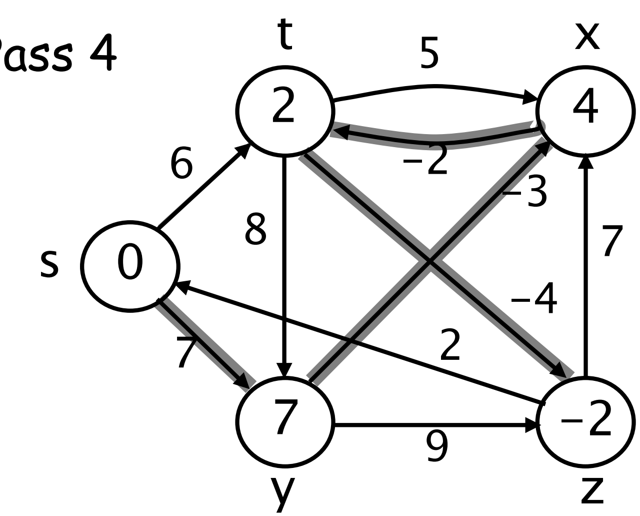
Pass 2



Pass 3



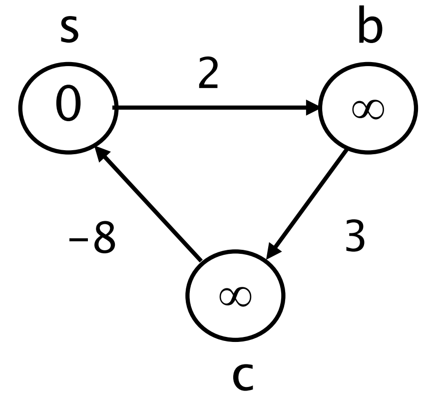
Pass 4



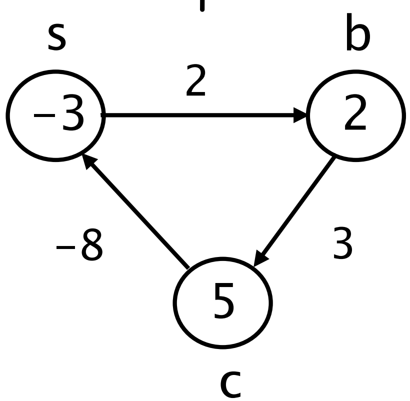
Edge order: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

Detecting Negative Cycles (perform extra test after V-1 iterations)

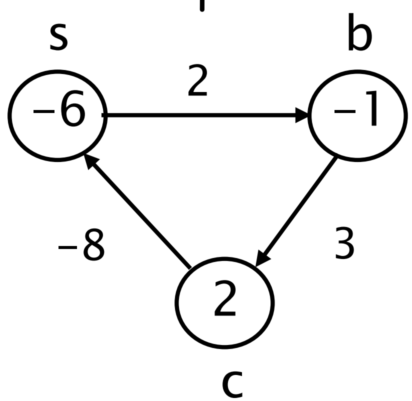
- **for** each edge $(u, v) \in E$
- **do if** $d[v] > d[u] + w(u, v)$
- **then return** FALSE
- **return** TRUE



1st pass



2nd pass



(s,b) (b,c) (c,s)

Look at edge (s, b):

$d[b] = -1$
$d[s] + w(s, b) = -4$

$\Rightarrow d[b] > d[s] + w(s, b)$

BELLMAN-FORD(V, E, w, s)

INITIALIZE-SINGLE-SOURCE(V, s) $\longleftarrow \Theta(|V|)$

for $i \leftarrow 1$ to $|V| - 1$ $\longleftarrow O(|V|)$

  do for each edge $(u, v) \in E$ $\longleftarrow O(|E|)$

    do RELAX(u, v, w)

$O(|V||E|)$

for each edge $(u, v) \in E$ $\longleftarrow O(|E|)$

  do if $d[v] > d[u] + w(u, v)$

    then return FALSE

return TRUE

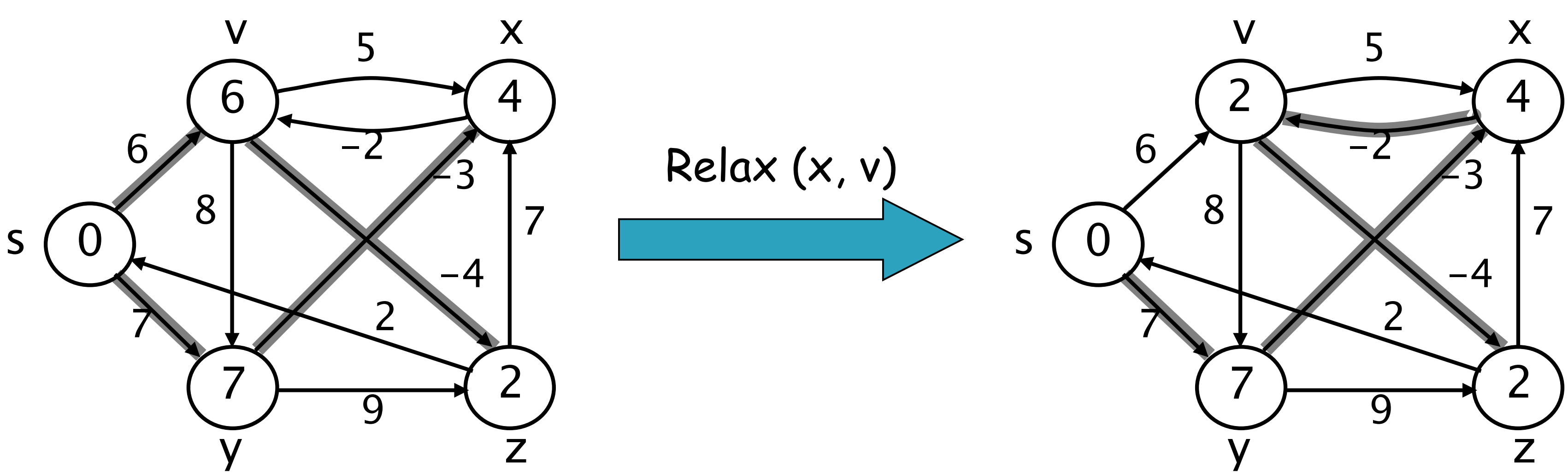
Running time: $O(|V|+|V||E|+|E|)=O(|V||E|)$

Key points of BELLMAN-FORD

- After |V|-1 iterations, d values will not be updated or can't be lower any more, and d values store the measure of the shortest path. Why?
  - Using a counter example to help you do the analysis
  - How to prove its correctness?

Shortest Path Properties

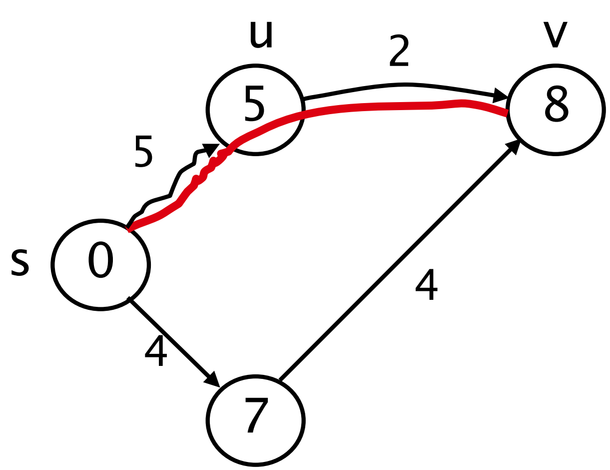
- **Upper-bound property**
  - We always have $d[v] \geq \delta(s, v)$ for all v
  - The estimate never goes up – relaxation only lowers the estimate



Shortest Path Properties

- **Convergence property**

  If $s \rightsquigarrow u \rightarrow v$ is a shortest path, and if $d[u] = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $d[v] = \delta(s, v)$ at all times after relaxing (u, v)



• If $d[v] > \delta(s, v) \Rightarrow$ after relaxation:

$$d[v] = d[u] + w(u, v)$$
$$d[v] = 5 + 2 = 7$$

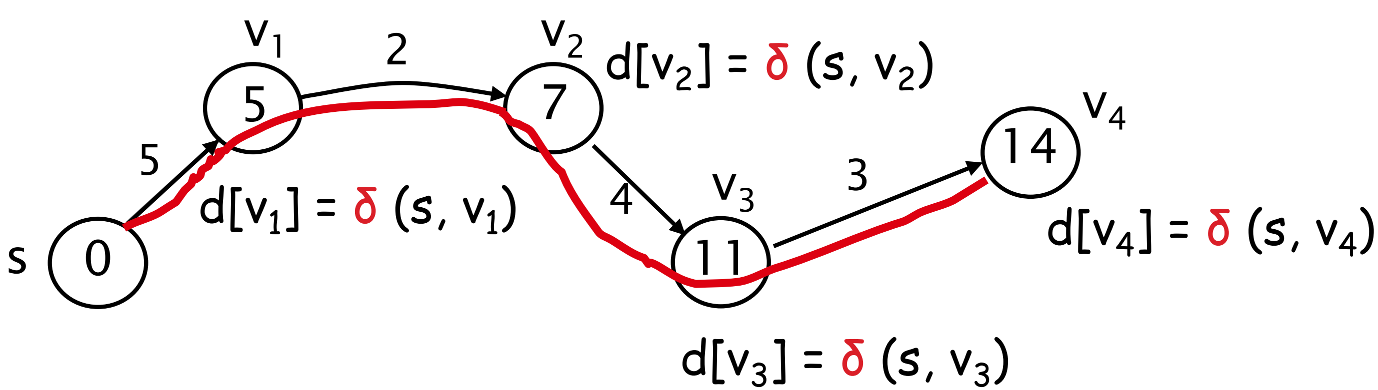
• Otherwise, the value remains unchanged, because it must have been the shortest path value

Shortest Path Properties

- **Path relaxation property**

  Let $p=\langle v_0, v_1, \ldots, v_k\rangle$ be a shortest path from $s = v_0$ to v_k

  If we relax, in order, $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, even intermixed with other relaxations, then $d[v_k] = \delta(s, v_k)$



Correctness of Belman-Ford Algorithm

- **Theorem:** Show that $d[v] = \delta(s, v)$, for every v, after $|V| - 1$ passes

  Case 1: G does not contain negative cycles which are reachable from s

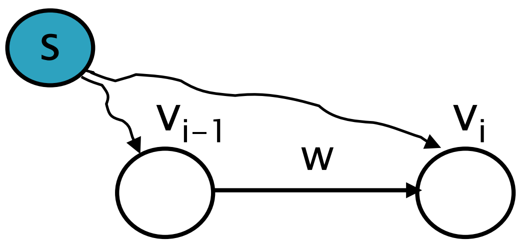
  - Assume that the shortest path from s to v is
    $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $s = v_0$ and $v = v_k$, $k \le |V|-1$
  - Use mathematical induction on the number of passes i to show that:
    $$d[v_i] = \delta(s, v_i) \text{ , } i=0,1,\ldots,k$$

Correctness of Belman-Ford Algorithm (cont.)

Base Case: $i=0$, $d[v_0] = \delta(s, v_0) = \delta(s, s) = 0$

Inductive Hypothesis: $d[v_{i-1}] = \delta(s, v_{i-1})$

Inductive Step: $d[v_i] = \delta(s, v_i)$



$d[v_{i-1}] = \delta(s, v_{i-1})$

After relaxing (v_{i-1}, v_i) (convergence property) :

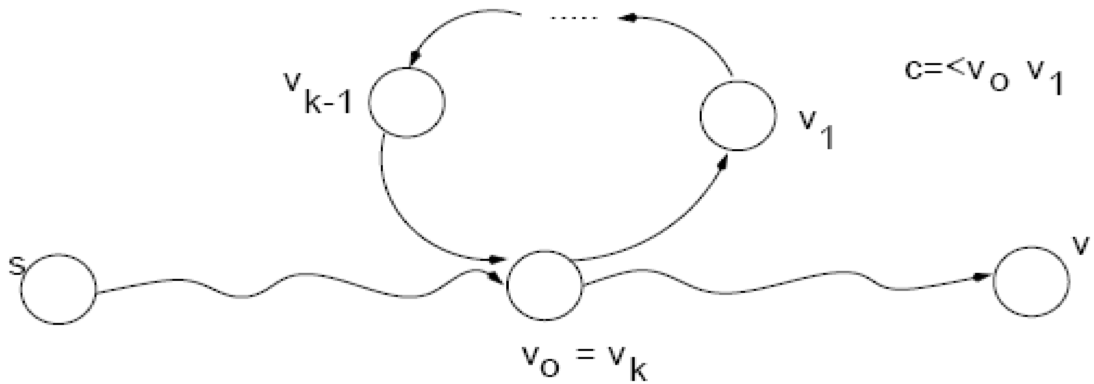
$d[v_i] \le d[v_{i-1}] + w = \delta(s, v_{i-1}) + w = \delta(s, v_i)$

From the upper bound property: $d[v_i] \ge \delta(s, v_i)$

Therefore, $d[v_i] = \delta(s, v_i)$

Correctness of Belman-Ford Algorithm (cont.)

- <u>Case 2:</u> G contains a negative cycle which is reachable from s



$c = \langle v_o\ v_1\ \ v_k \rangle$ is a negative cycle

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

Proof by Contradiction: suppose the algorithm returns a solution

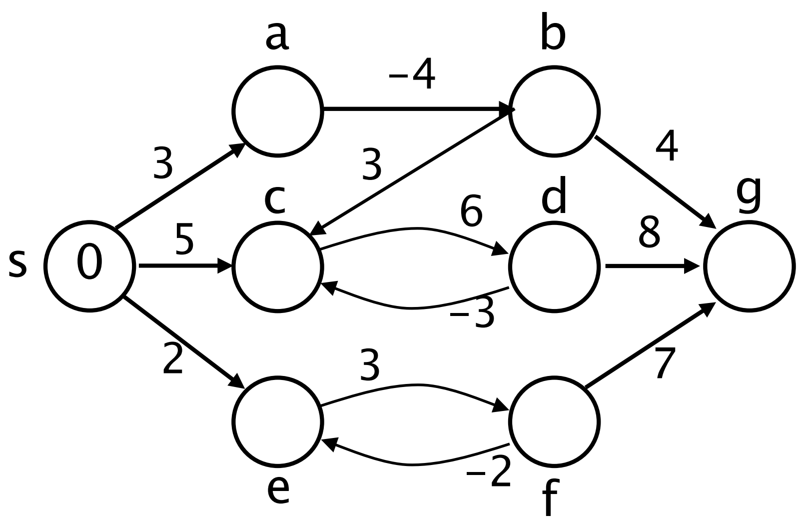
After relaxing (v_{i-1}, v_i): $dist[v_i] \le dist[v_{i-1}] + w(v_{i-1}, v_i)$

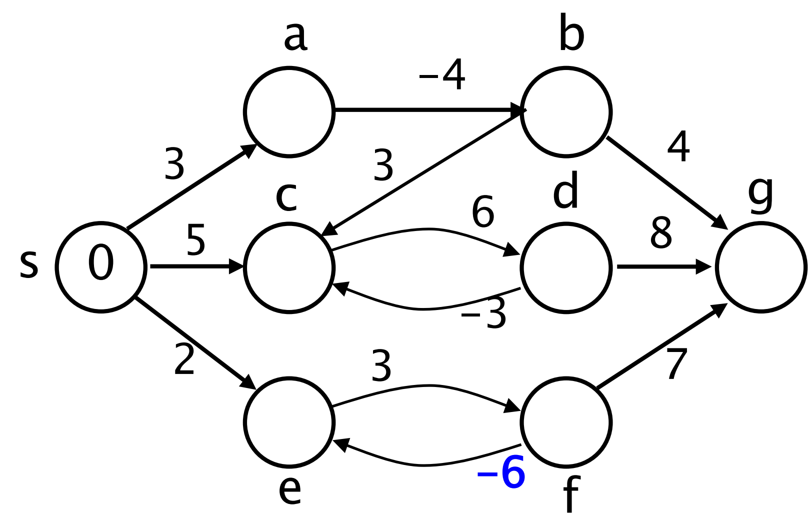
$$\Rightarrow \sum_{i=1}^{k} dist[v_i] \le \sum_{i=1}^{k} dist[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$$\Rightarrow \sum_{i=1}^{k} w(v_{i-1}, v_i) \ge 0 \ (\sum_{i=1}^{k} dist[v_i] = \sum_{i=1}^{k} dist[v_{i-1}])$$

Contradiction!

Exercise 1





Floyd's Algorithm

(all pairs shortest paths)

All pairs shortest path

- The graph: may contain negative edges but no negative cycles

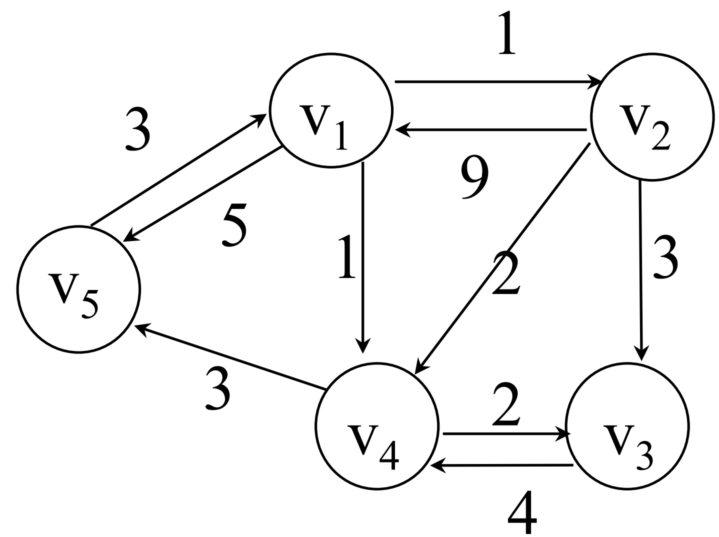
- A representation: a weight matrix where
  W(i,j)=0 if i=j
  W(i,j)=∞ if there is no edge between i and j
  W(i,j)="weight of edge"

- The problem: find the shortest path between every pair of vertices of a graph

- Note: we have shown principle of optimality applies to shortest path problems

The weight matrix and the graph

	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



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

The subproblems

- How can we define the shortest distance $d_{i,j}$ in terms of "smaller" problems?
- 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

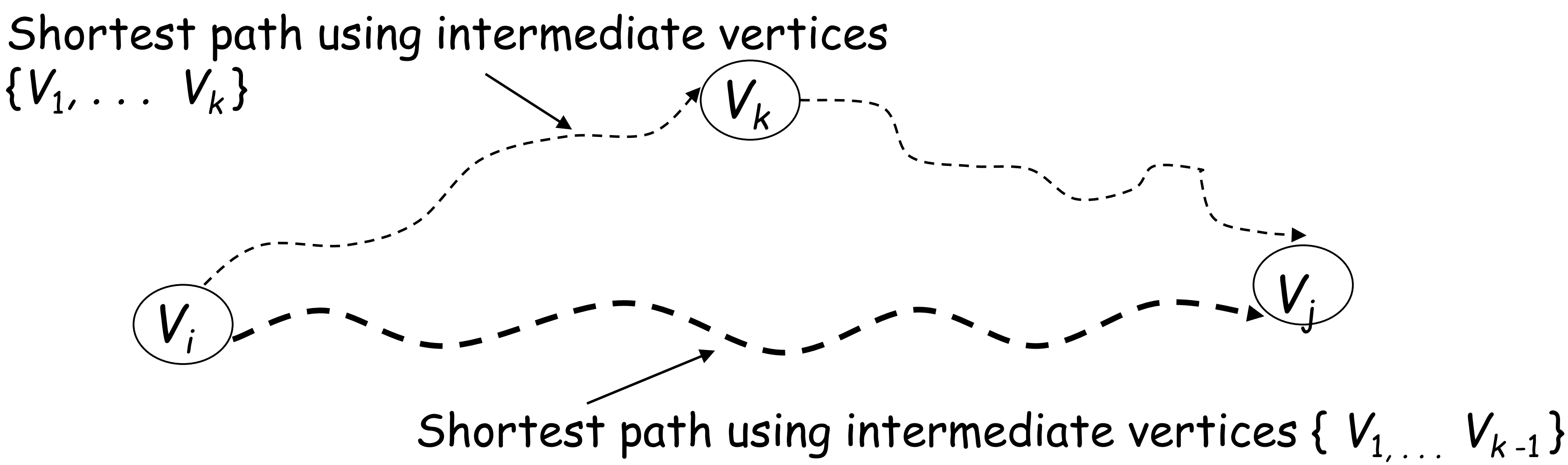
The subproblems

- Let $D^{(k)}[i,j]$=weight of a shortest path from v_i to v_j using only vertices from $\{v_1,v_2,\ldots,v_k\}$ 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 $\{v_1, v_2, \ldots, v_k\}$ 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 $\{v_1, v_2, \ldots, v_k\}$ as intermediate vertices does use v_k Then $D^{(k)}[i,j] = D^{(k-1)}[i,k] + D^{(k-1)}[k,j]$



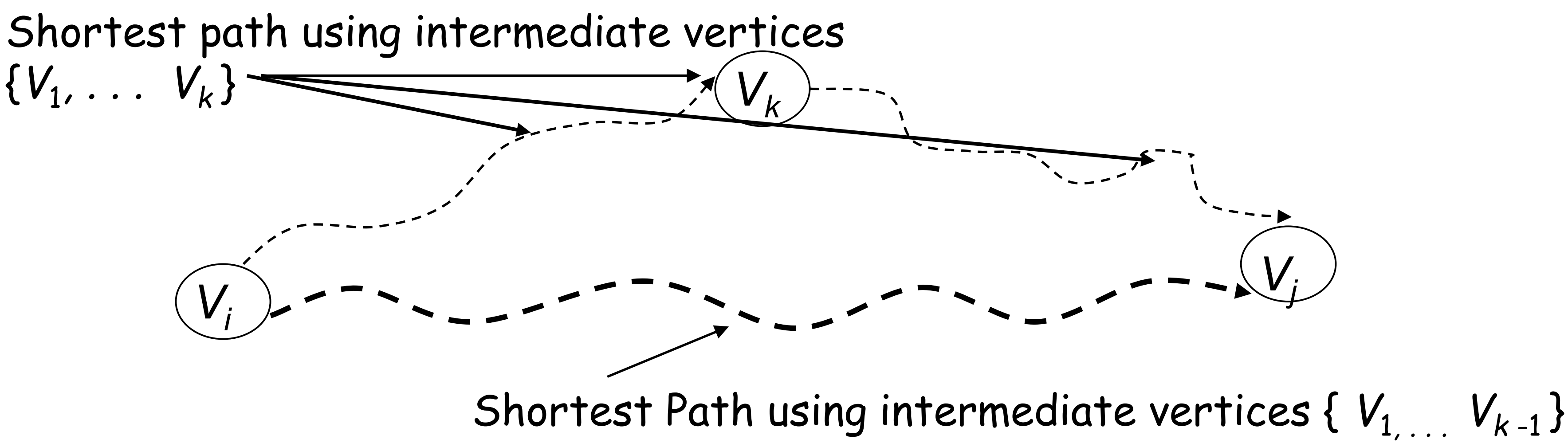
The recursive definition

- 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:

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



The pointer array P

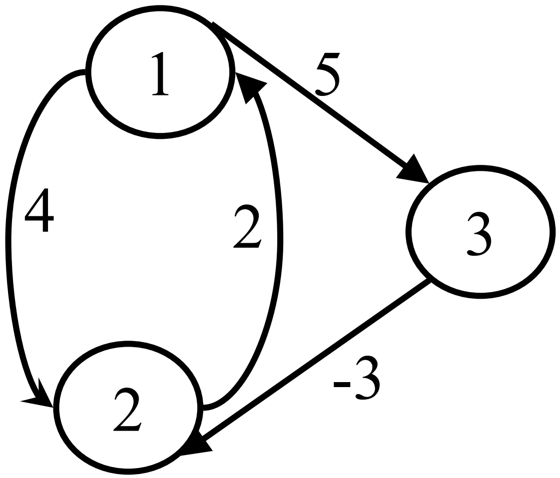
- Used to enable finding a shortest path
- Initially the array contains 0
- Each time that a shorter path from *i* to *j* is found the *k* that provided the minimum is saved (highest index node on the path from *i* to *j*)
- To print the intermediate nodes on the shortest path a recursive procedure that print the shortest paths from *i* and *k*, and from *k* to *j* can be used

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



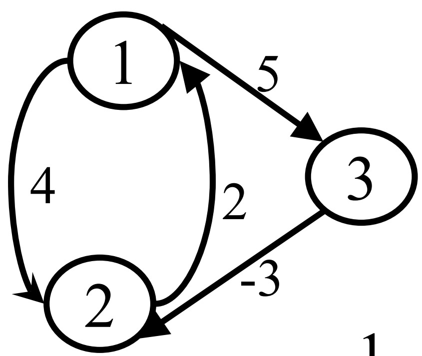
$W = D^0 =$

	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

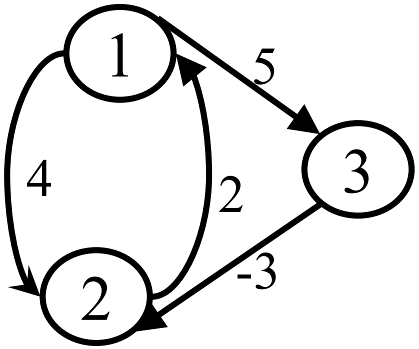
$$D^1[2,3] = \min(D^0[2,3], D^0[2,1]+D^0[1,3])$$
$$= \min (\propto, 7)$$
$$= 7$$

$P =$

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

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

Example



$D^1 =$

	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

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

$P =$

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

$D^2[3,1] = \min(D^1[3,1], D^1[3,2]+D^1[2,1])$
$= \min (\propto, -3+2)$
$= -1$

Floyd's Algorithm: Using 2 D matrices

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

Can we use only one D matrix?

- $D[i,j]$ depends only on elements in the kth column and row of the distance matrix
- We will show that the kth row and the kth 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 kth row and column of D remain unchanged we show that the main diagonal remains 0

- $D^{(k)}[\, j,j \,] = \min\{ D^{(k-1)}[\, j,j \,], D^{(k-1)}[\, j,k \,] + D^{(k-1)}[\, k,j \,] \}$
  $= \min\{ 0,\ D^{(k-1)}[\, j,k \,] + D^{(k-1)}[\, k,j \,] \}$
  $= 0$

- 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}

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

Question

- Can we claim that D^k equals to D^{k-1}, D^{k-2} ?
  - No, we can only claim that
    - 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. $P[\, i, j\,] \leftarrow k$;

$O(|V|^3)$

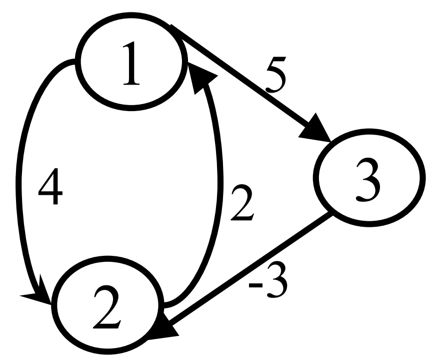
Total time cost: $O(|V|^3)$

Printing intermediate nodes on shortest path from q to r

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

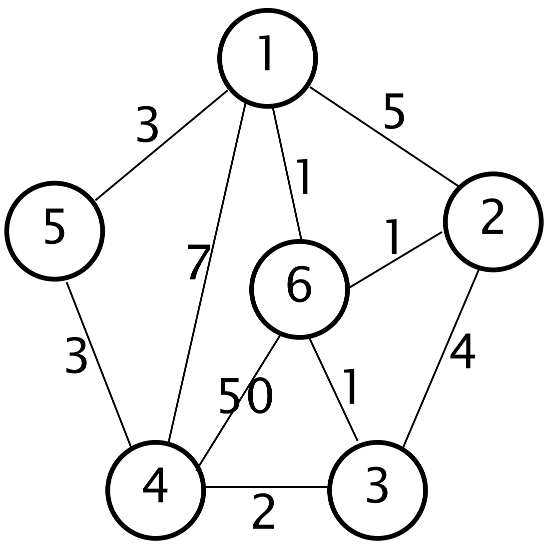
P =

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



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

Example

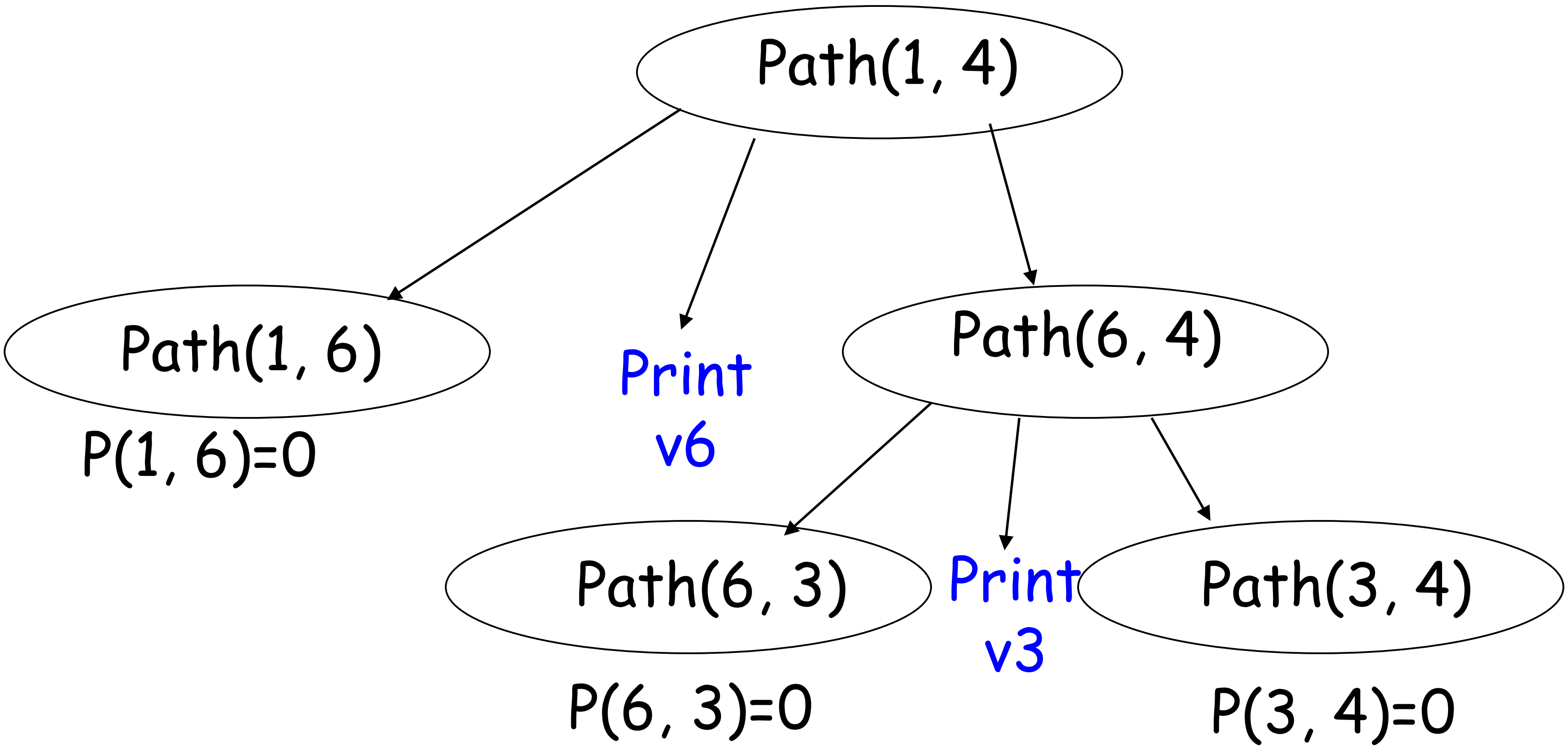


$D^6 =$

	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
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 1 to 4 are v6, v3.
The shortest path is v1, v6, v3, v4.

Recommended Reading

- Reading this week
  - Textbook Chapters 24-25

- Next Week
  - DAG checking and topological sort, Chapter 22