Problem Definition:

- Given a directed graph $G=(V, E, W)$, where each edge has a weight (length, cost),
- Find a shortest path from s to v.
  - s—source
  - v—destination.

Dijkstra's algorithm

$d[s] \leftarrow 0$

for each $v \in V \setminus \{s\}$

  do $d[v] \leftarrow \infty$, $\pi[v] \leftarrow NIL$.

$S \leftarrow \varnothing$

$Q \leftarrow V$ % $Q=V\setminus S$ is a priority queue

while $Q \neq \varnothing$

  do $u \leftarrow$ EXTRACT-MIN(Q)

   $S \leftarrow S \cup \{u\}$

relaxation step

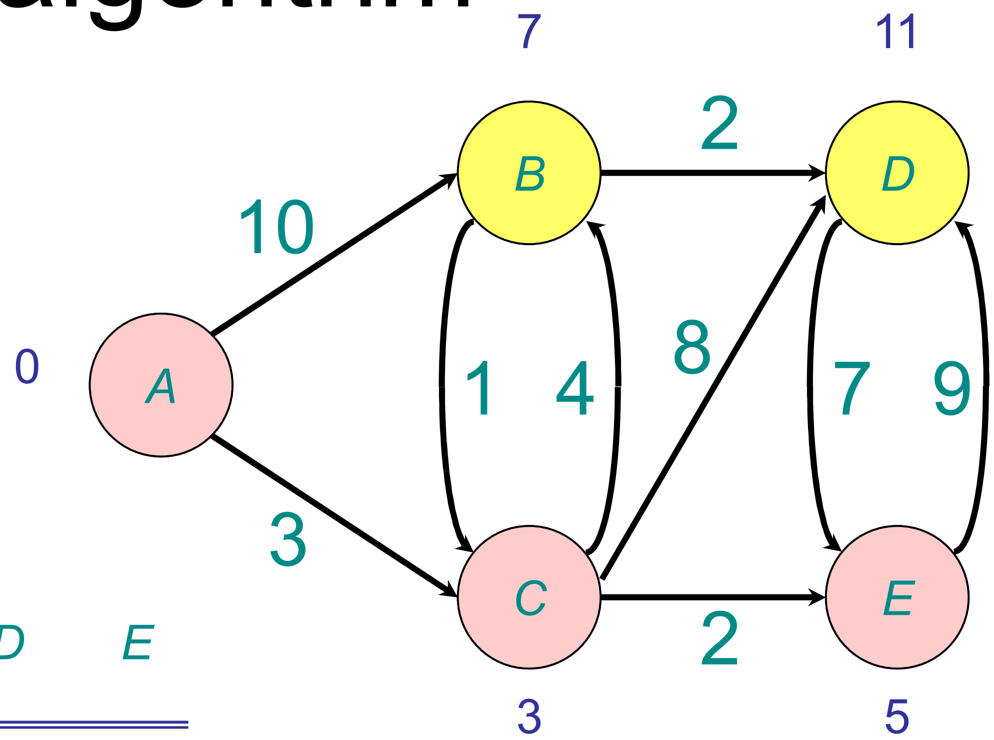
for each $v \in Adj[u]$ & $v \in V \setminus S$

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

   then $d[v] \leftarrow d[u] + w(u, v)$, $\pi[v] \leftarrow u$,

Example of Dijkstra's algorithm

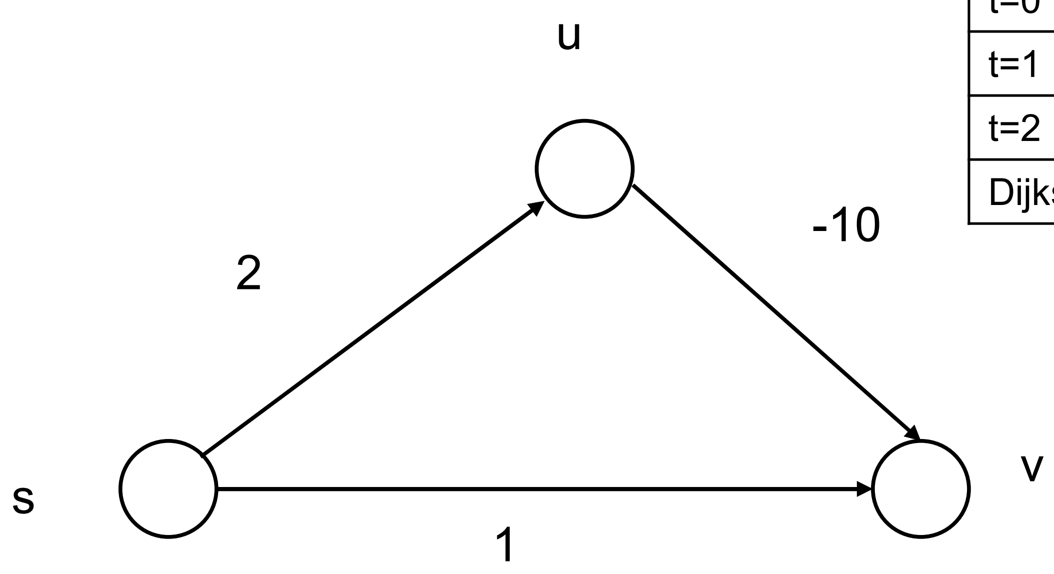
"E" ← EXTRACT-MIN(Q):



Q:	A	B	C	D	E
d	0	7	3	11	5
π	NIL	C	A	C	C

S: { A, C, E }

The algorithm does not work if there are negative weight edges in the graph



	s	v	u
t=0	0/Nil	Inf/Nil	Inf/Nil
t=1		1/s	2/s
t=2			2/s
Dijkstra doesn’t work			

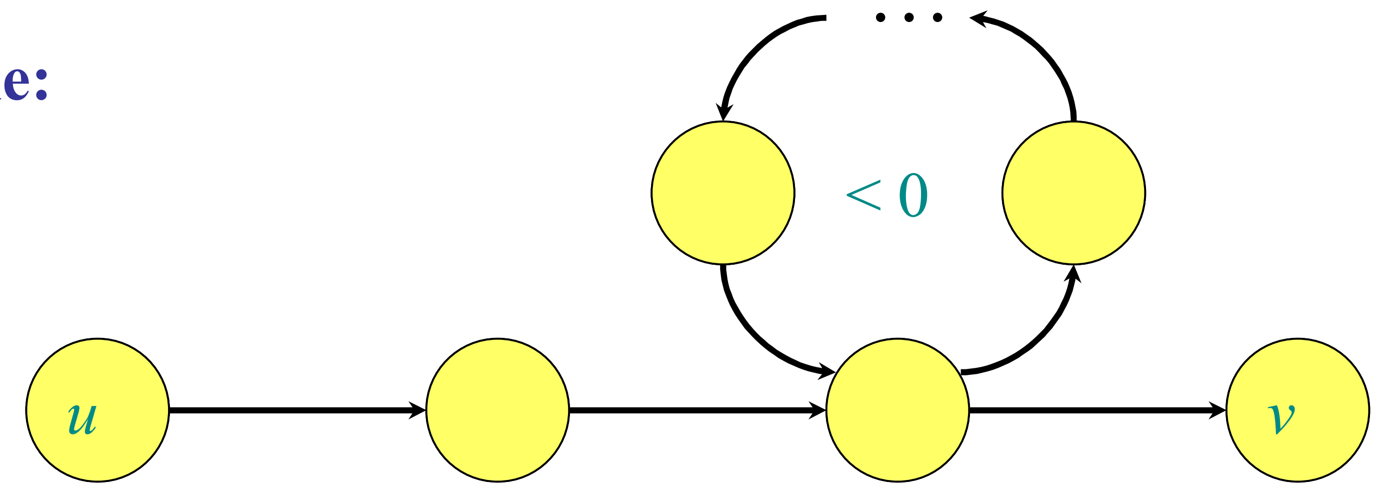
s$\rightarrow$v is shorter than s $\rightarrow$ u, but it is longer than s $\rightarrow$ u $\rightarrow$ v.

Lecture 10: Shortest Paths
with **Negative weighted** edges
Bellman-Ford algorithm

Negative-weight cycles

Recall: If a graph $G = (V, E)$ contains a negative-weight cycle, then some shortest paths may not exist.

Example:



Bellman-Ford algorithm: Finds all shortest-path lengths from a ***source*** $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.

Shortest Paths: Dynamic Programming

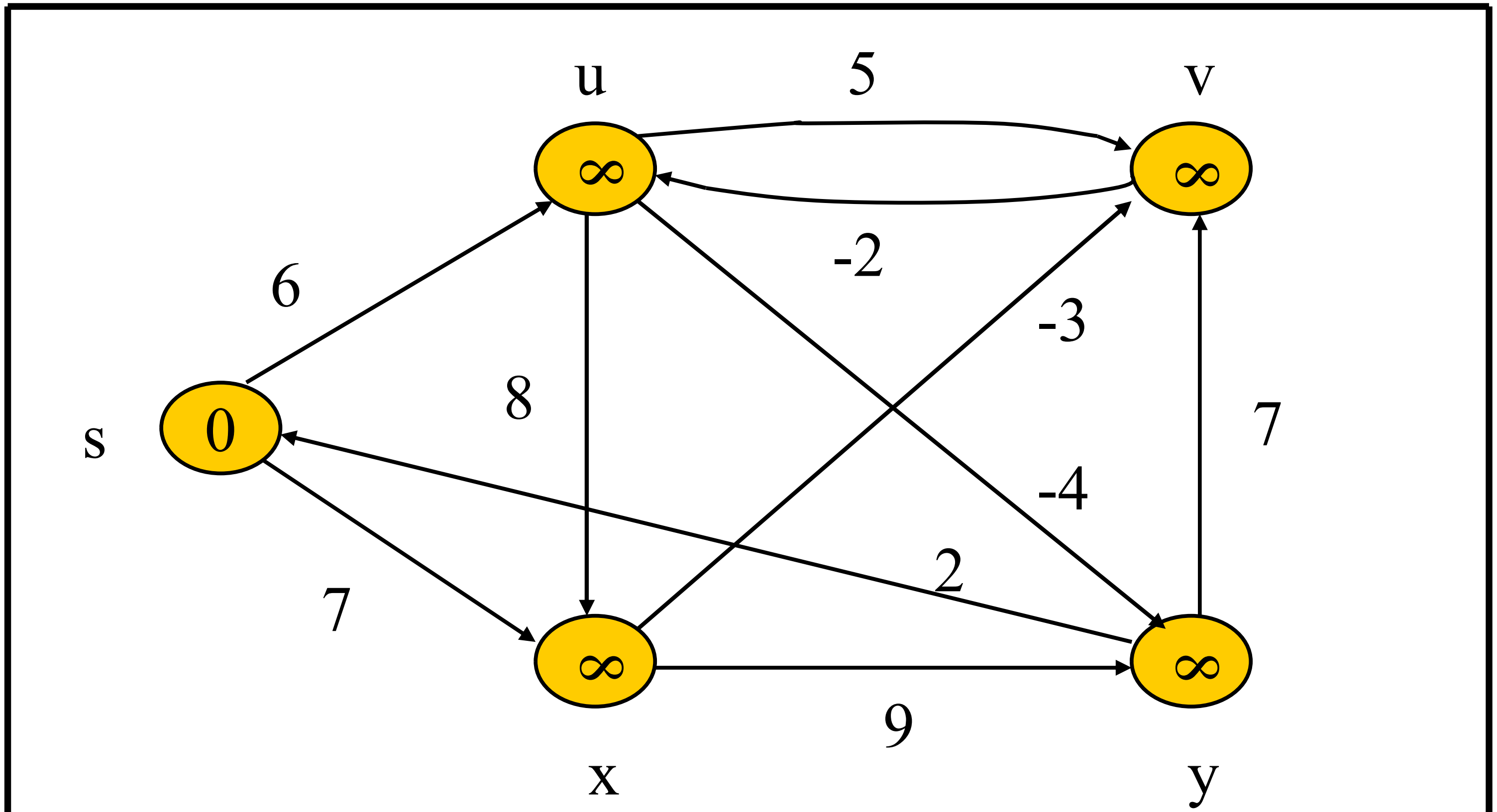
Def. OPT(i, v)=length of shortest s-v path P using at most i edges.

- Case 1: P uses at most i-1 edges.
  - OPT(i, v) = OPT(i-1, v)
- Case 2: P uses exactly i edges.
  - If (w, v) is the last edge, then OPT use the best s-w path using at most i-1 edges and edge (w, v).

$$OPT(i,v) = \begin{cases} 0 & if\ i = 0\ and\ v = s \\ \infty & if\ i = 0, and\ v \neq s \\ \min\{OPT(i-1,v), \min_{(w,v)\in E}\{OPT(i-1,w) + C_{wv}\}\} & otherwise \end{cases}$$

Remark: if no negative cycles, then OPT(n-1, v)=length of shortest s-v path.
n: the number of nodes.

- Using this recursive equation, You can design a DP algorithm. Opt(v, i) is a subproblem (exercise).



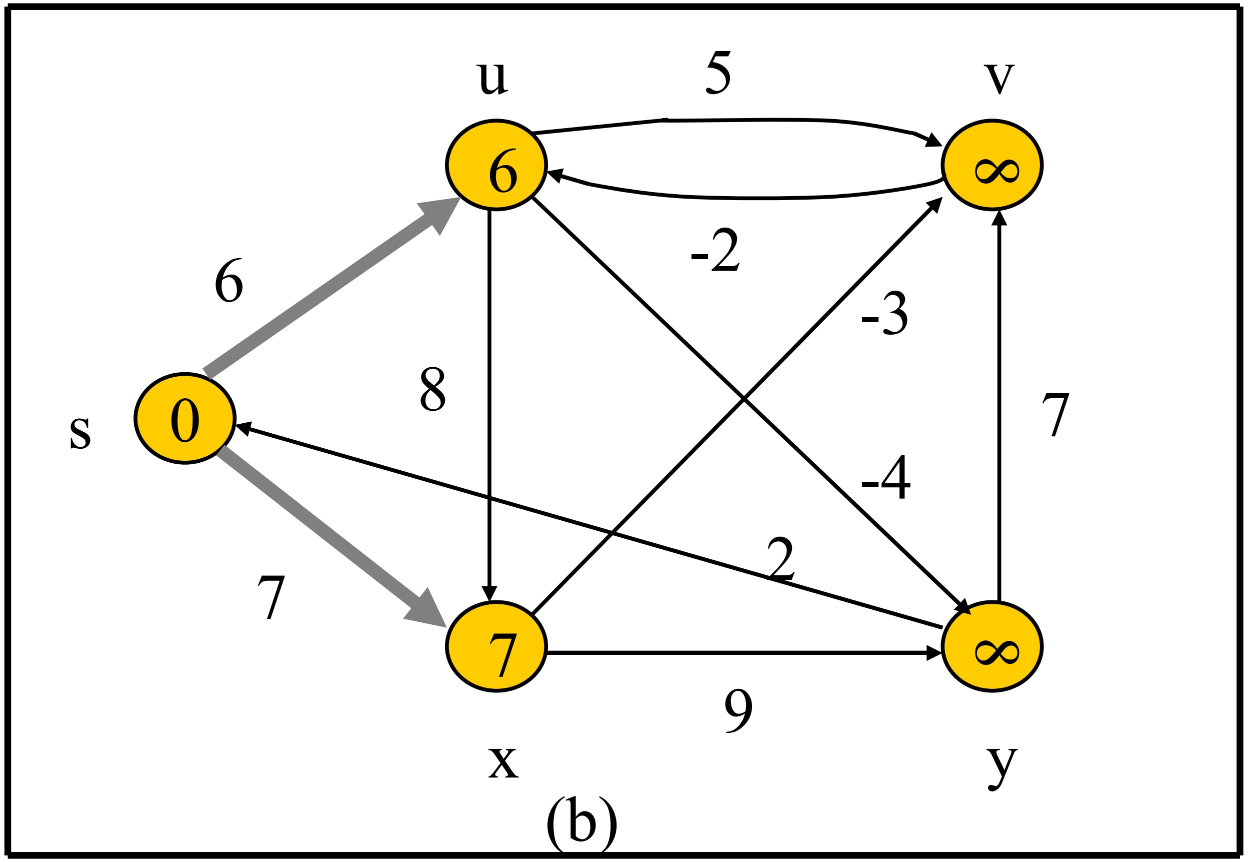
(a)

$$OPT(i,v) = \begin{cases} 0 & if\ i = 0\ and\ v = s \\ \infty & if\ i = 0, and\ v \neq s \\ \min\{OPT(i-1,v), \min_{(w,v)\in E}\{OPT(i-1,w) + C_{wv}\}\} & otherwise \end{cases}$$

vertex: s u v x y i=0

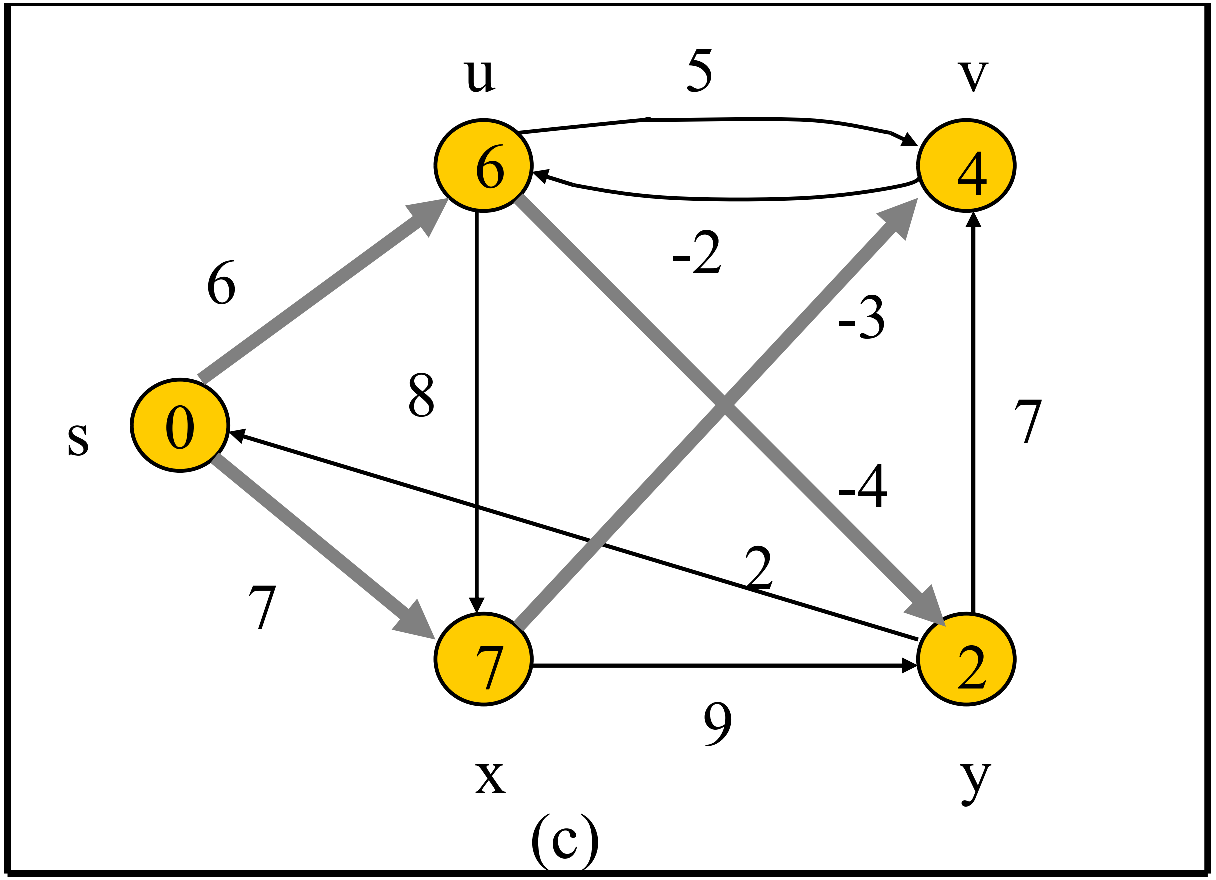
d: 0 ∞ ∞ ∞ ∞

π: s - - - -



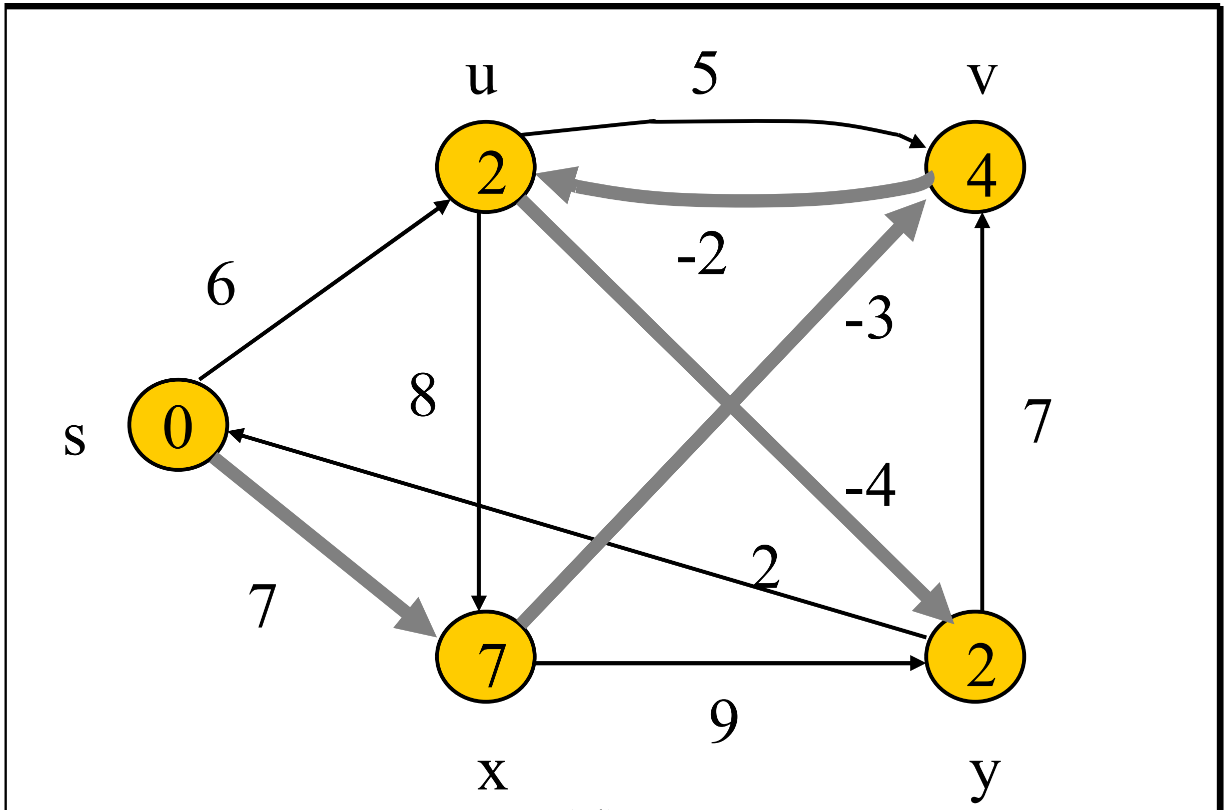
$$OPT(i,v) = \begin{cases} 0 & if\ i = 0\ and\ v = s \\ \infty & if\ i = 0,and\ v \neq s \\ \min\{OPT(i-1,v), \min_{(w,v)\in E}\{OPT(i-1,w) + C_{wv}\}\} & otherwise \end{cases}$$

vertex:	s	u	v	x	y	i=1
d:	0	6	∞	7	∞	
π:	s	s	-	s	-	



$$OPT(i,v) = \begin{cases} 0 & if\ i = 0\ and\ v = s \\ \infty & if\ i = 0, and\ v \neq s \\ \min\{OPT(i-1,v), \min_{(w,v)\in E}\{OPT(i-1,w) + C_{wv}\}\} & otherwise \end{cases}$$

vertex:	s	u	v	x	y	i=2
d:	0	6	4	7	2	
π	s	s	x	s	u	



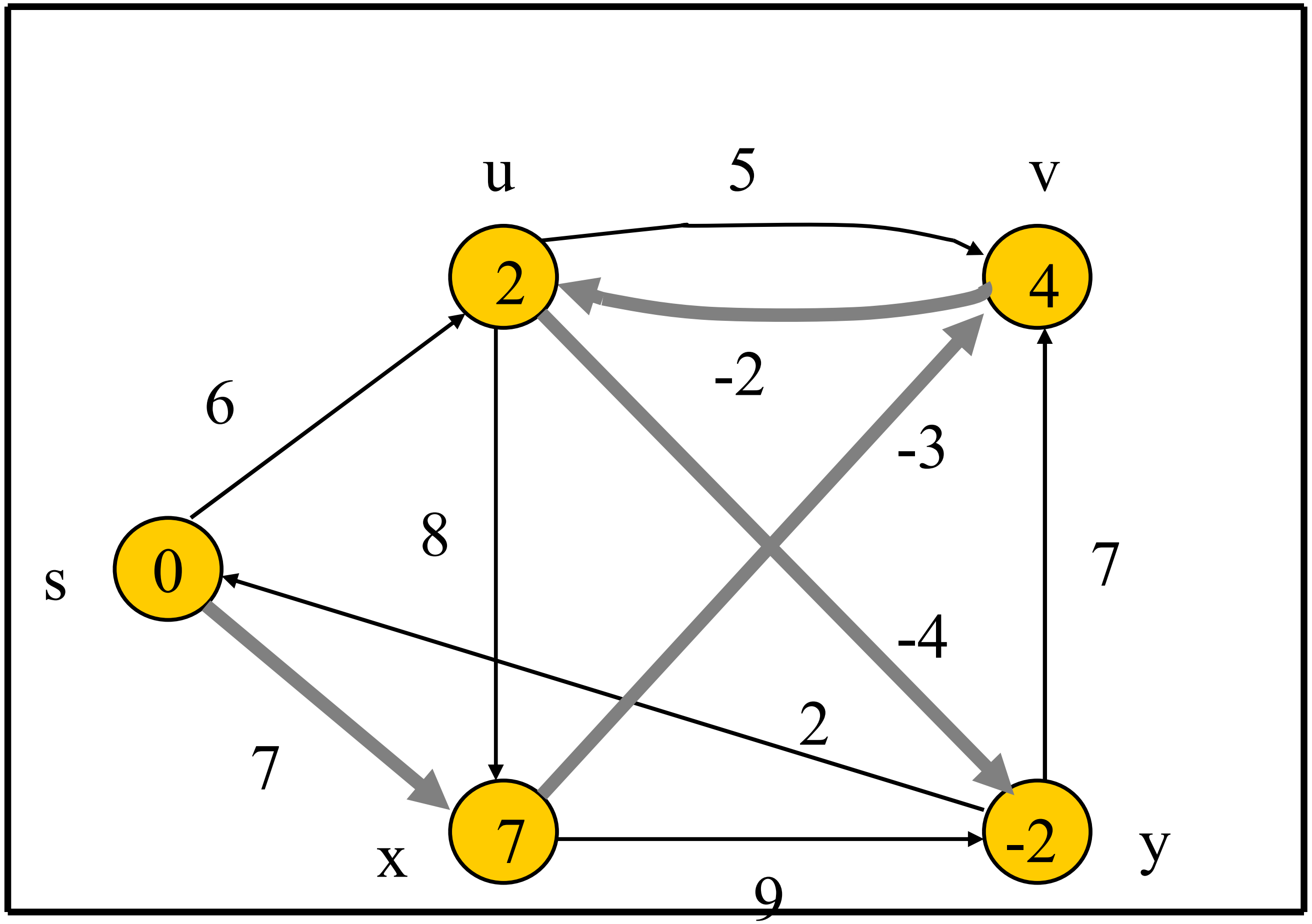
(d)

$$OPT(i,v) = \begin{cases} 0 & \textit{if } i = 0 \textit{ and } v = s \\ \infty & \textit{if } i = 0, \textit{and } v \neq s \\ \min\{OPT(i-1,v), \min_{(w,v)\in E}\{OPT(i-1,w) + C_{wv}\}\} & \textit{otherwise} \end{cases}$$

vertex: s u v x y   i=3

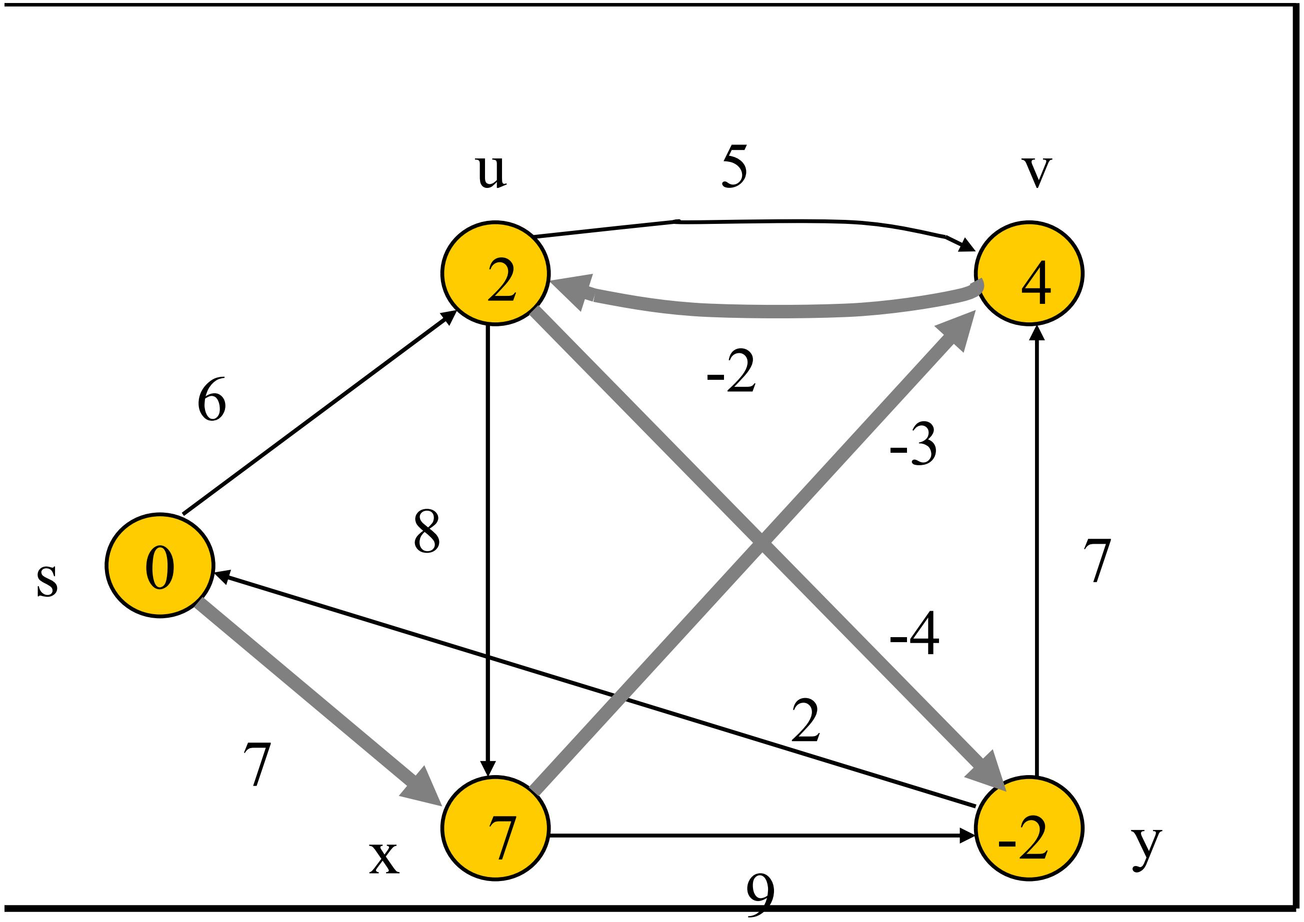
d: 0 2 4 7 2

π: s v x s u



(e)

vertex:	s	u	v	x	y	i=4
d:	0	2	4	7	-2	
π:	s	v	x	s	u	



(e)

$$OPT(i,v) = \begin{cases} 0 & if\ i = 0\ and\ v = s \\ \infty & if\ i = 0, and\ v \neq s \\ \min\{OPT(i-1,v), \min_{(w,v)\in E}\{OPT(i-1,w) + C_{wv}\}\} & otherwise \end{cases}$$

vertex: s u v x y i=5

d: 0 2 4 7 -2

π: s v x s u

$$OPT(i,v) = \begin{cases} 0 & if\ i = 0\ and\ v = s \\ \infty & if\ i = 0, and\ v \neq s \\ \min\{OPT(i-1,v), \min\limits_{(w,v)\in E} \{OPT(i-1,w) + C_{wv}\}\} & otherwise \end{cases}$$

vertex:	s	u	v	x	y	
d:	∞	∞	∞	∞	∞	i=0
π:	s	-	-	-	-	
d:	0	6	∞	7	∞	i=1
π:	s	s	-	s	-	
d:	0	6	4	7	2	i=2
π:	s	s	x	s	u	
d:	0	2	4	7	2	i=3
π:	s	v	x	s	u	
d:	0	2	4	7	-2	i=4
π:	s	v	x	s	u	
d:	0	2	4	7	-2	i=5
π:	s	v	x	s	u	

So, no negative cycle.

- Bellman-Ford algorithm is more efficient.

Bellman-Ford algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$

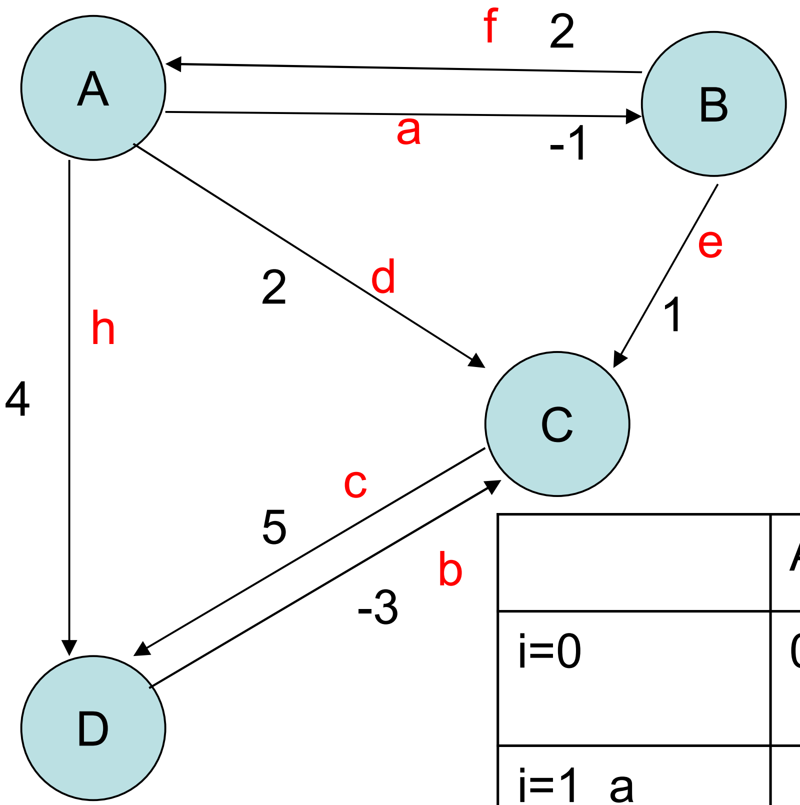
initialization

for $i \leftarrow 1$ **to** $|V| - 1$
  do for **each edge** $(u, v) \in E$
    do if $d[v] > d[u] + w(u, v)$
      then $d[v] \leftarrow d[u] + w(u, v)$

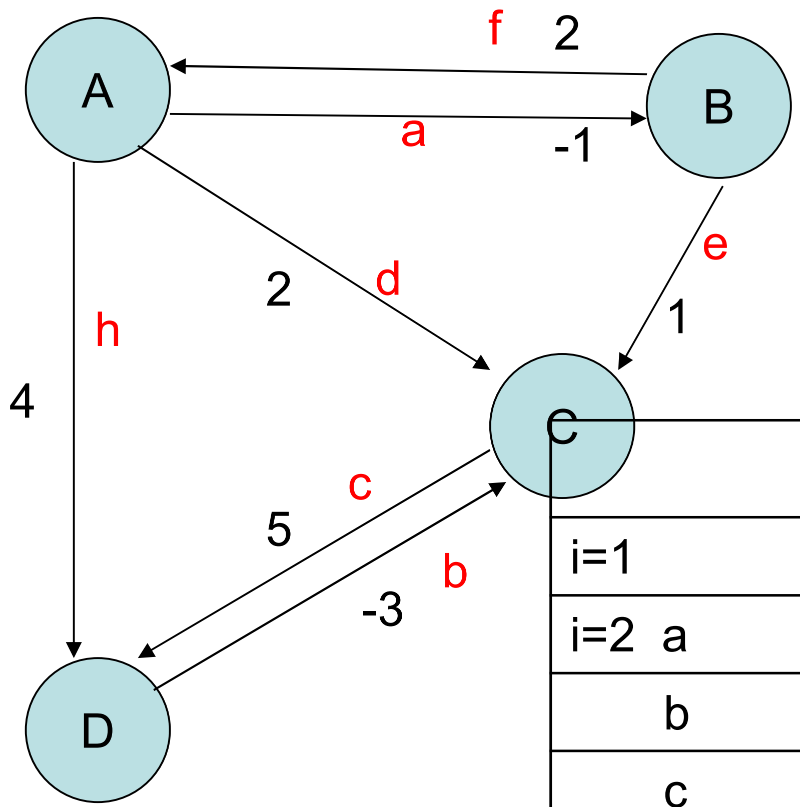
relaxation step

for each edge $(u, v) \in E$
  do if $d[v] > d[u] + w(u, v)$
    then report that a negative-weight cycle exists

At the end, $d[v] = \delta(s, v)$. Time $= O(|V|\,|E|)$.



	A	B	C	D
i=0	$0/NIL$	∞/NIL	∞/NIL	∞/NIL
i=1 a		-1/A		
b			X	
c				X
d			2/A	
e			0/B	
f	X			
h				4/A
	0/NIL	-1/A	0/B	4/A

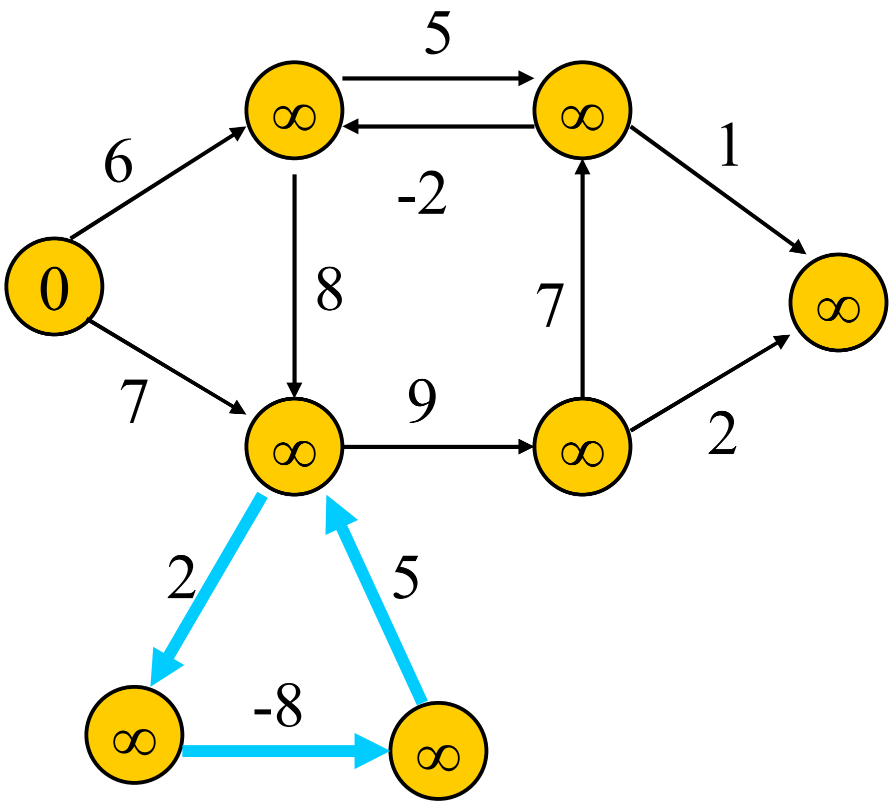


	A	B	C	D
i=1	0/NIL	-1/A	0/B	4/A
i=2 a		-1/A		
b			X	
c				X
d			X	
e			X	
f	X			
h				X
	0/NIL	-1/A	0/B	4/A

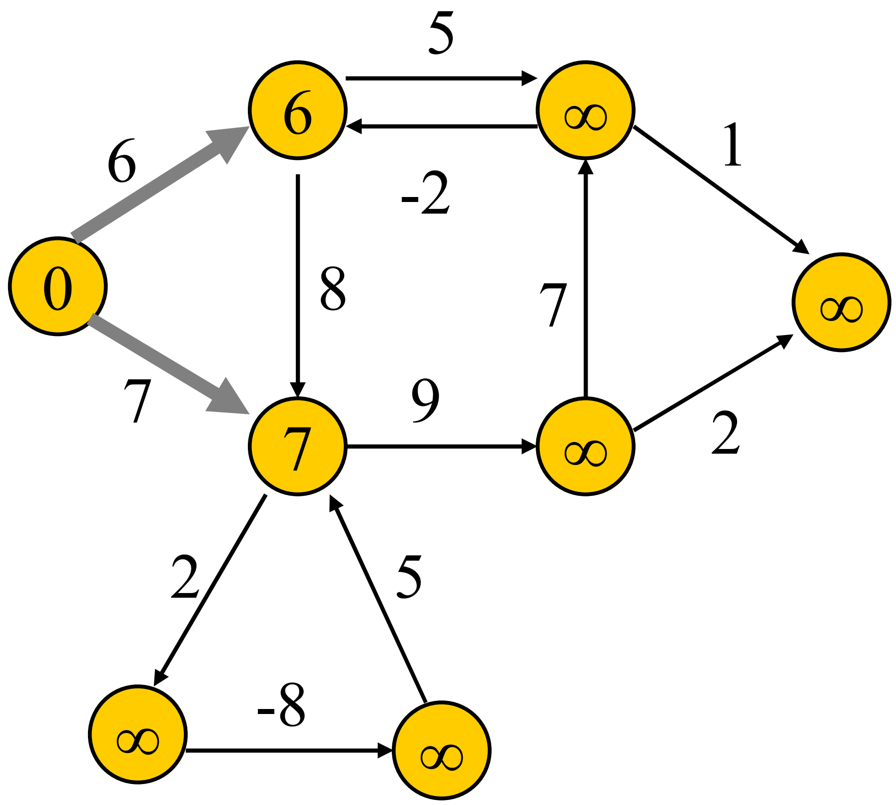
The result for i=2 is the same as i=1, so are i=3, 4.
Conclusion: no negative cycle.

Corollary: If negative-weight circuit exists in the given graph, in the n-th iteration, the cost of a shortest path from *s* to *some* node *v* will be further reduced.

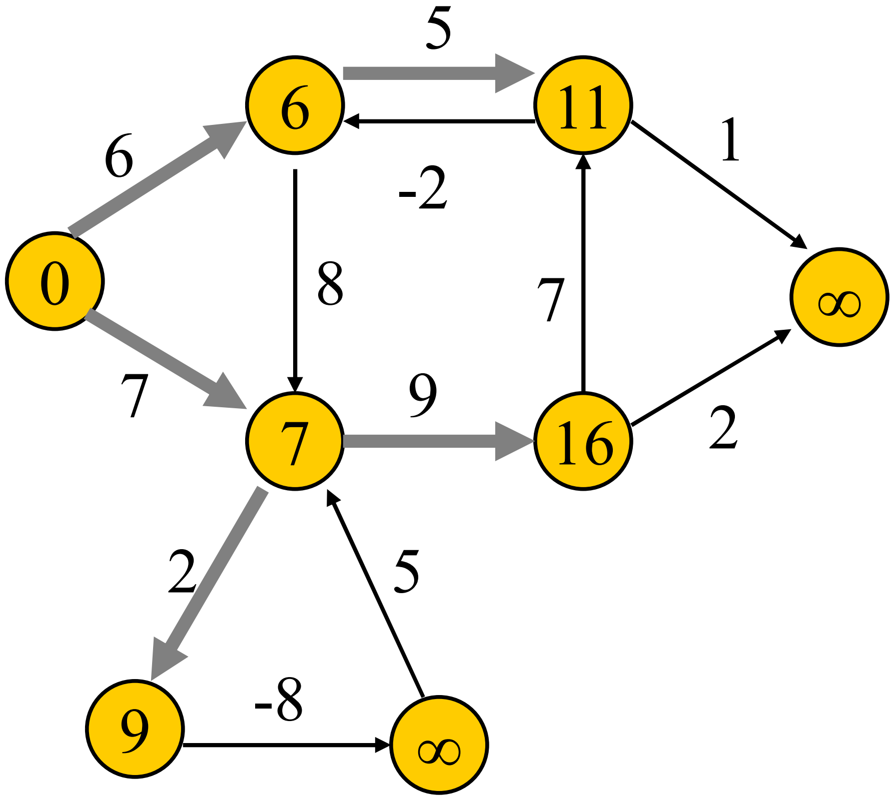
Demonstrated by the following examples.



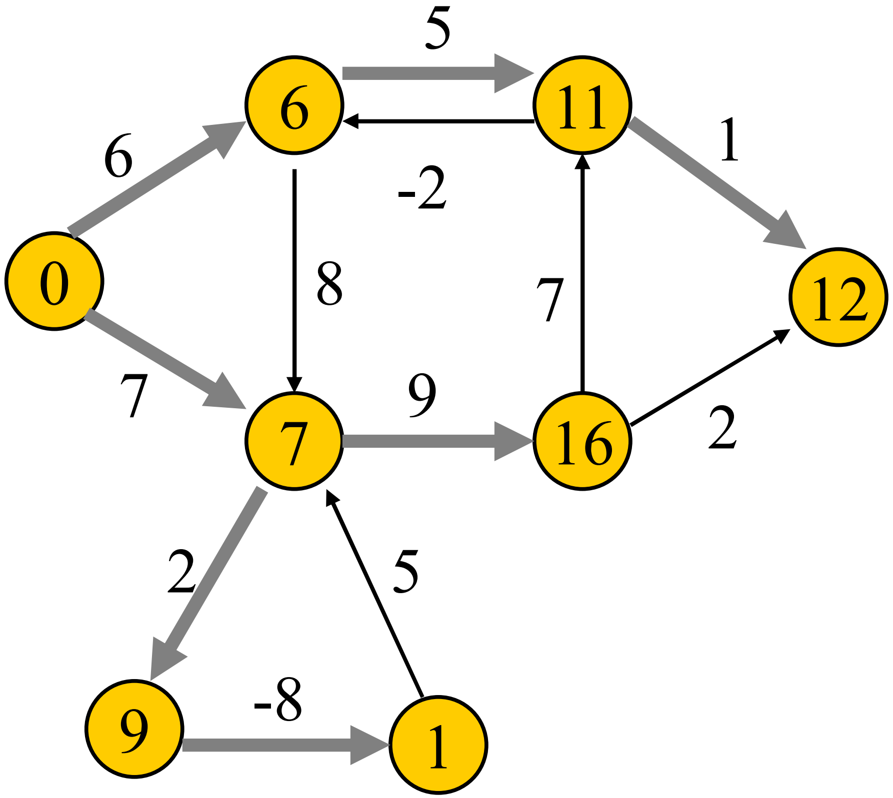
An example with negative-weight cycle



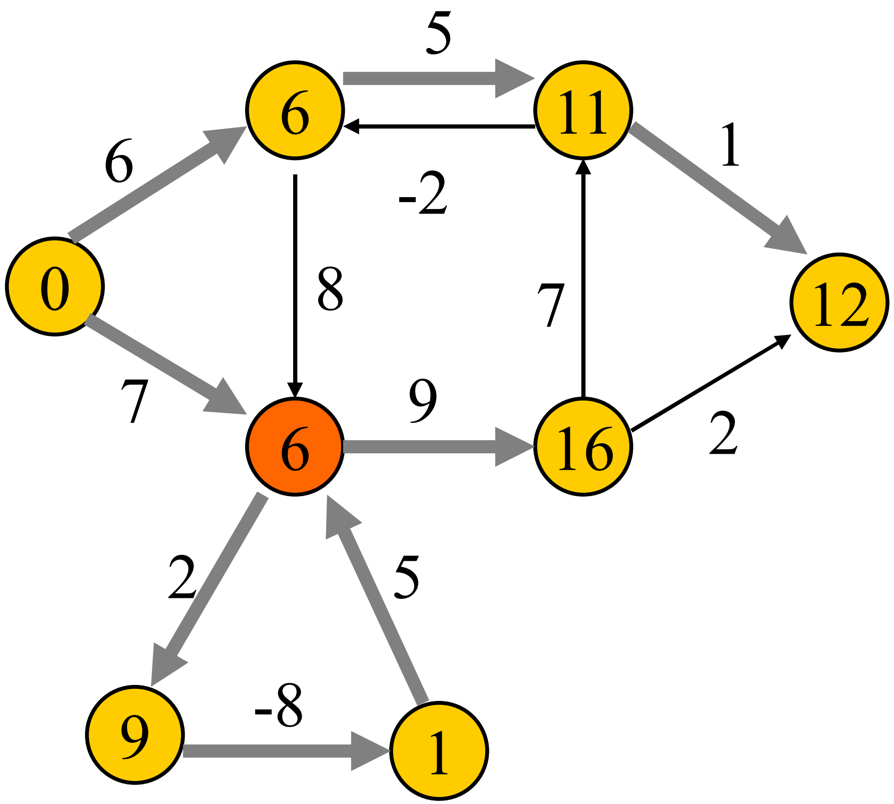
i=1



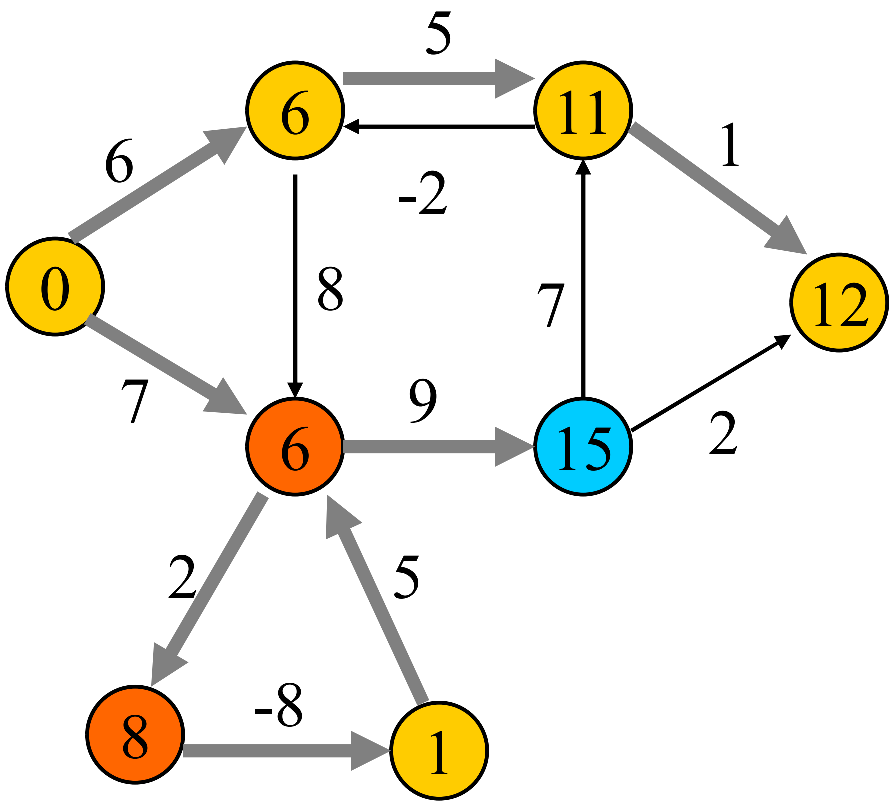
i=2



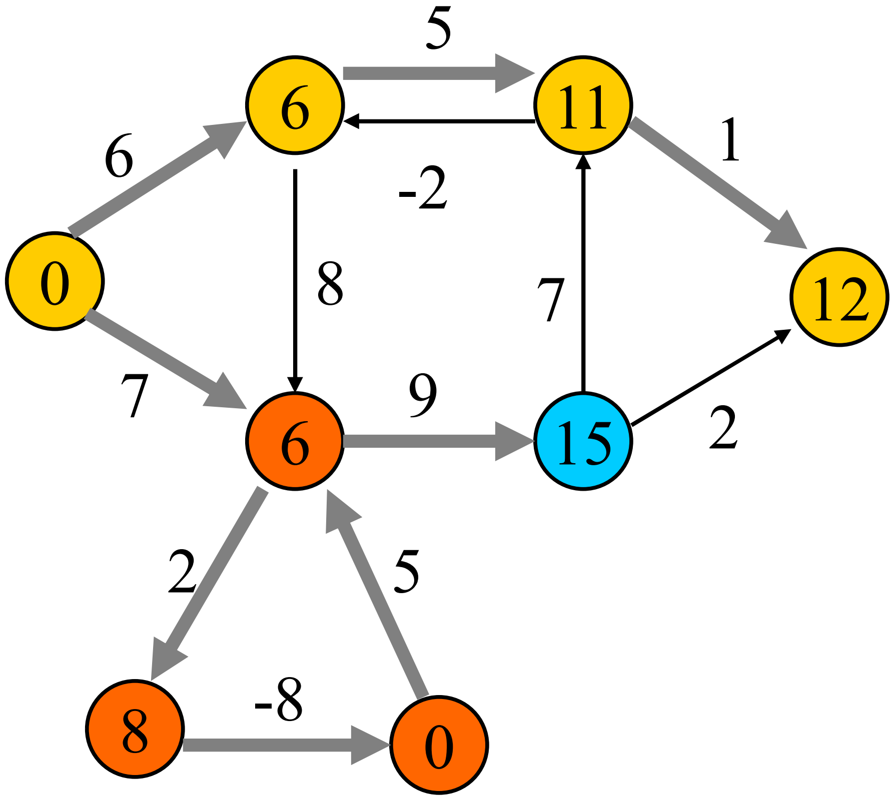
i=3



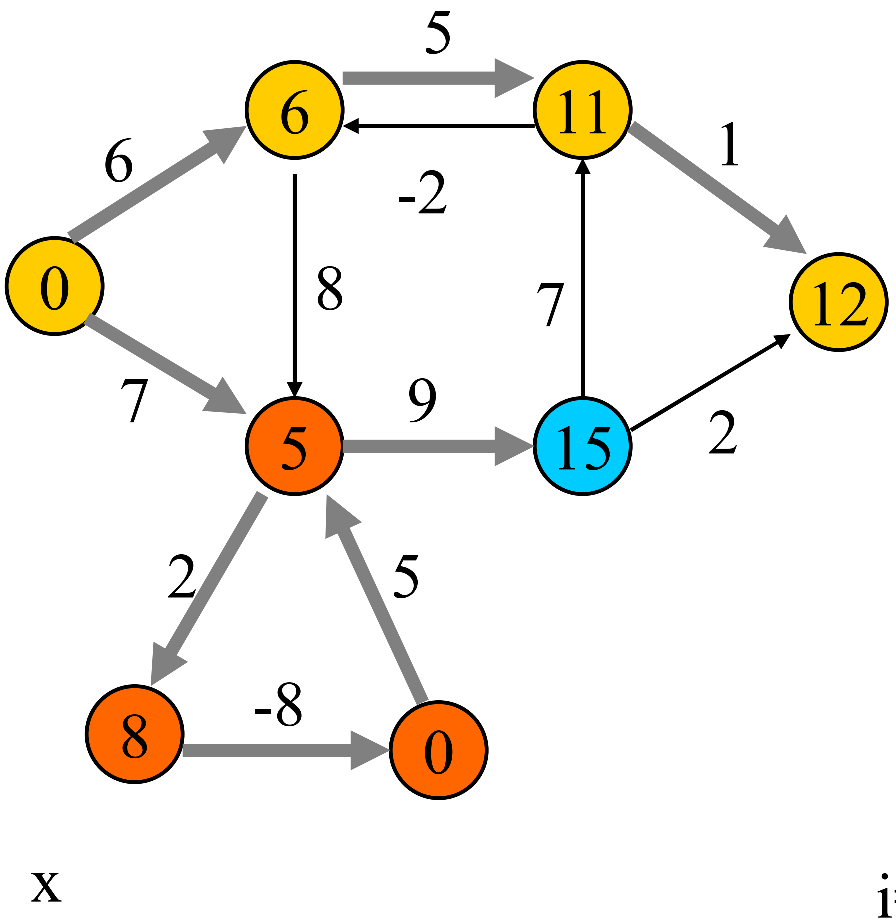
i=4



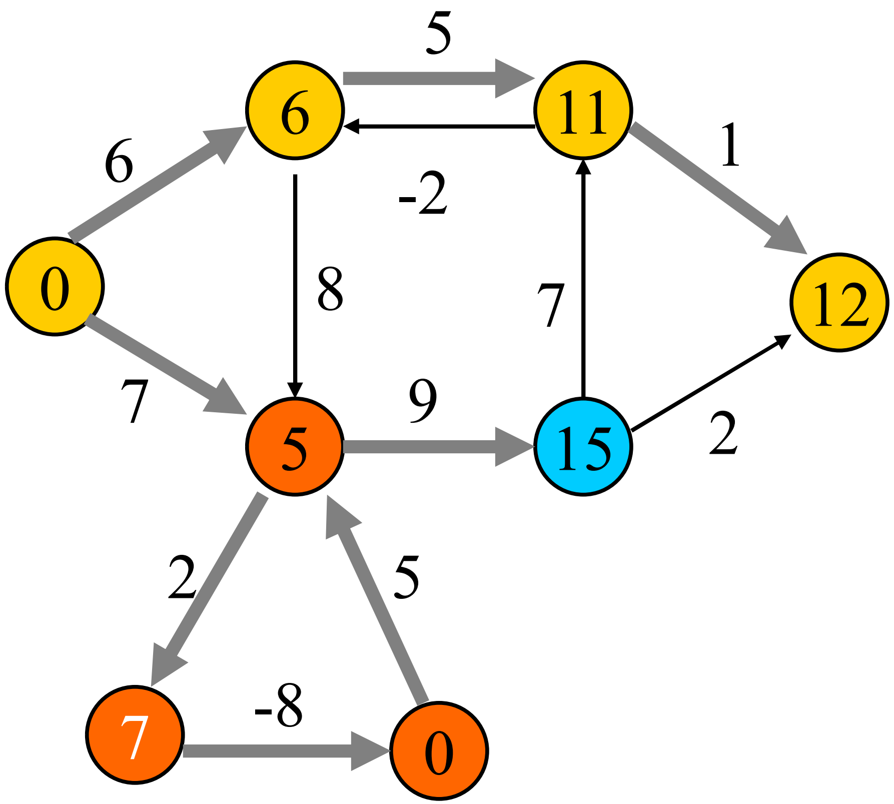
i=5



i=6



i=7



x

i=8

0-1 Knapsack Problem

Knapsack Problem ***0-1 version***

Knapsack problem.

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ kilograms and has value $v_i > 0$.
- Knapsack has capacity of W kilograms.
- Goal: fill knapsack so as to maximize total value.

Ex: { 3, 4 } has value 40.

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Greedy: repeatedly add item with maximum ratio v_i / w_i.
Ex: { 5, 2, 1 } achieves only value = 35 $\Rightarrow$ greedy not optimal.

Dynamic Programming: Adding a New Variable

Def. $OPT(i, w)$ = max profit subset of items 1, …, i with weight limit w.

- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 } using weight limit w

- Case 2: OPT selects item i.
  - new weight limit = $w - w_i$
  - OPT selects best of { 1, 2, …, i–1 } using this new weight limit

$$
OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\left\{ OPT(i-1, w),\ \ v_i + OPT(i-1, w - w_i) \right\} & \text{otherwise} \end{cases}
$$

Knapsack Problem: Bottom-Up

Knapsack. Fill up an n-by-W array.

```
Input: n, w1,…,wN, v1,…,vN

for w = 0 to W
   M[0, w] = 0

for i = 1 to n
   for w = 1 to W
      if (wi > w)
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi ]}

return M[n, W]
```

Knapsack Problem: Running Time

Running time. $\Theta(n\ W)$.

- Not polynomial in input size!
- "Pseudo-polynomial."
- Decision version of Knapsack is NP-complete. [Chapter 8]

$$OPT(i,w) = \begin{cases} 0 & if\ i = 0 \\ OPT(i-1,w) & if\ w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1, w-w_i)\} & otherwise \end{cases}$$

Knapsack Algorithm

W + 1 →

n + 1 ↓

	0	1	2	3	4	5	6	7	8	9	10	11
φ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	34	40

OPT: { 4, 3 }
value = 22 + 18 = 40

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Summary

- Bellman-ford algorithm
  - Comparison with Dijkstra Algorithm.
- Knapsack Problem.