Lecture 9

Dynamic Programming II: Knapsack, Interval Scheduling, Bellman-Ford

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

Dynamic Programming

Technique for solving optimization problems.

Solve problem by solving **sub**-problems and combine:

This is called Optimal substructure property.

- Similar to divide-and-conquer: recursion (for solving sub-problems)
- Sub-problems overlap: solve them only once!

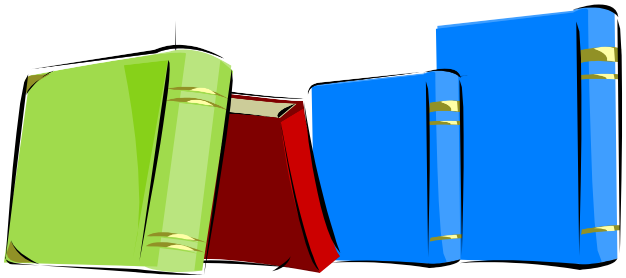
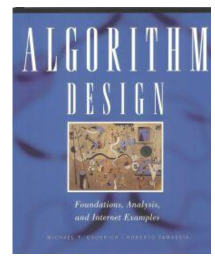
DP = recursion + re-use (Memoization)

Case study II: 0/1 Knapsack

Problem: A set of n items, with each item i having positive weight w_i and positive benefit v_i. You are asked to choose items with maximum total benefit so that the total weight is at most W

Example:

"knapsack" with 9 lbs capacity

Items:  

Weight:	4 lbs	2 lbs	2 lbs	6 lbs	2 lbs
Benefit:	$20	$3	$6	$25	$80

Solution:

- item 5 ($80, 2 lbs)
- item 3 ($6, 2lbs)
- item 1 ($20, 4lbs)

Case study II: 0/1 Knapsack

Idea: Dynamic Programming.

Step 1: Define the problem and subproblems.

Answer: Let $DP[k, j]$ be the maximum value I can get from items $\{1, \ldots, k\}$ without exceeding j.

Case study II: 0/1 Knapsack

Idea: Dynamic Programming.

Step 1: Define the problem and subproblems.

Answer: Let $DP[k, j]$ be the maximum value I can get from items $\{1, \ldots, k\}$ without exceeding j.

Step 2: Define the goal/output given Step 1.

It is $\boldsymbol{DP[n, W]}$.

Case study II: 0/1 Knapsack

Idea: Dynamic Programming.

Step 1: Define the problem and subproblems.

Answer: Let $DP[k, j]$ be the maximum value I can get from items $\{1, \ldots, k\}$ without exceeding j.

Step 2: Define the goal/output given Step 1.

It is $\boldsymbol{DP[n, W]}$.

Step 3: Define the base cases

It is $DP[0, j] = 0$ for all j and $DP[i, 0] = 0$ for all i.

Step 4: Define the recurrence

Case study II: 0/1 Knapsack

Idea: Dynamic Programming.

Step 4: Define the recurrence

Item k will be **used** or **not**.

$$DP[k,j] \; = \; \max(\mathbf{DP[k-1,j-w_k]+v_k}, \mathbf{DP[k-1,j]})$$

Case study II: 0/1 Knapsack

Idea: Dynamic Programming.

Step 4: Define the recurrence

Item k will be **used** or **not**.

$$DP[k,j] \;=\; \max(\mathbf{DP[k-1, j-w_k] + v_k}, \mathbf{DP[k-1, j]})$$

Question: How do we know that item k does not have weight more than j?

Case study II: 0/1 Knapsack

Idea: Dynamic Programming.

Step 4: Define the recurrence

Item k will be **used** or **not**.

$$DP[k,j] = \begin{cases} \text{if } w_k \leq j & \max(\mathbf{DP[k-1, j-w_k] + v_k}, \mathbf{DP[k-1, j]}) \\ \text{If } w_k > j & \boldsymbol{DP[k-1, j]} \end{cases}$$

Answer: Add an if statement in the recurrence.

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

Initialization:

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0				
2	0				
3	0				

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0 $(j < w_1)$			
2	0	0 $(j < w_2)$			
3	0	0 $(j < w_3)$			

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	max(0,v_1+0)		
2	0	0			
3	0	0			

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	1		
2	0	0	max(1,v_2+0)		
3	0	0			

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	1		
2	0	0	1		
3	0	0	1 ($j < w_3$)		

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	1	max(0,v_1+0)	
2	0	0	1		
3	0	0	1		

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	1	1	
2	0	0	1	max(1,v_2+0)	
3	0	0	1		

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	1	1	
2	0	0	1	1	
3	0	0	1	max(1,v_3+0)	

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	1	1	max(0,v_1+0)
2	0	0	1	1	
3	0	0	1	5	

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	1	1	1
2	0	0	1	1	max(1,v_2+1)
3	0	0	1	5	

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	1	1	1
2	0	0	1	1	2
3	0	0	1	5	max(2,0+v_3)

Case study II: 0/1 Knapsack

Example: 3 items, $W = 4$
$w_1 = 2, v_1 = 1, w_2 = 2, v_2 = 1, w_3 = 3, v_3 = 5$

	j=0	1	2	3	4
i=0	0	0	0	0	0
1	0	0	1	1	1
2	0	0	1	1	2
3	0	0	1	5	**5**

Case study II: 0/1 Knapsack

Pseudocode:

Array DP[][]
For $i = 0$ to n **do**
  $\text{DP}[i, 0] \leftarrow 0$
For $j = 1$ to W **do**
  $\text{DP}[0, j] \leftarrow 0$
For $i = 1$ to n **do**
  For $j = 1$ to W **do**
    If $j < w_i$ **then**
      $\text{DP}[i, j] \leftarrow \text{DP}[i-1, j]$
    else $\text{DP}[i, j] \leftarrow \max(\text{DP}[i-1, j], \text{DP}[i-1, j-w_i] + v_i)$
return $\text{DP}[n, W]$

Initialization

Bottom up filliing DP

Goal

Case study II: 0/1 Knapsack

Pseudocode:

Array DP[][]
For $i = 0$ to n **do**
  $\text{DP}[i, 0] \leftarrow 0$
For $j = 1$ to W **do**
  $\text{DP}[0, j] \leftarrow 0$
For $i = 1$ to n **do**
  For $j = 1$ to W **do**
    If $j < w_i$ **then**
      $\text{DP}[i, j] \leftarrow \text{DP}[i-1, j]$
    else $\text{DP}[i, j] \leftarrow \max(\text{DP}[i-1, j], \text{DP}[i-1, j-w_i] + v_i)$
return $\text{DP}[n, W]$

Initialization

Bottom up filliing DP

Goal

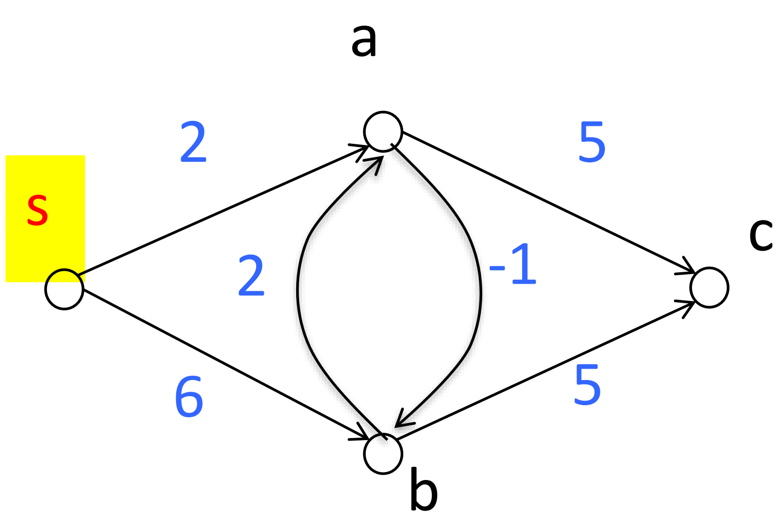
Running time: $\Theta(nW)$

Case study III: Bellman-Ford

Problem: Given a directed graph $G(V,E)$, with edge-weights w_e for every edge e and a source node s, find all shortest-path weights from s to all other vertices.

Remark: A path $p = \langle v_0, v_1, \ldots, v_k \rangle$ has weight $\sum_{i=1}^{k} w(v_{i-1}, v_i)$.

Example:

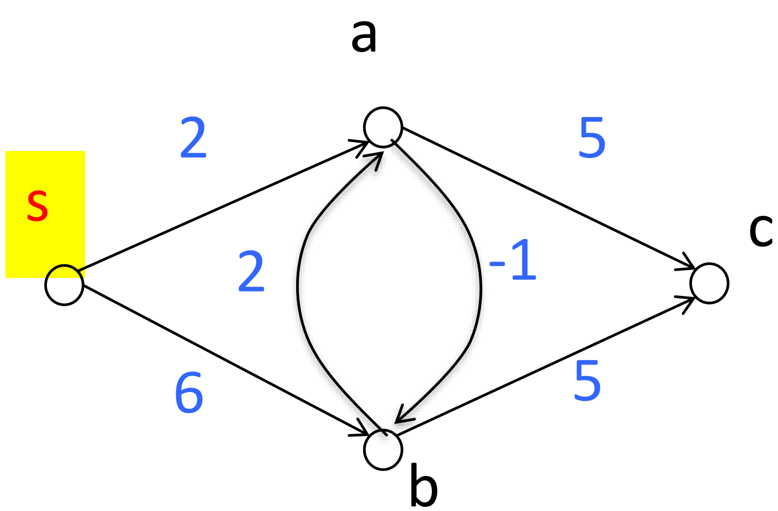


Case study III: Bellman-Ford

Problem: Given a directed graph $G(V, E)$, with edge-weights w_e for every edge e and a source node s, find all shortest-path weights from s to all other vertices.

Remark: A path $p = \langle v_0, v_1, \dots, v_k \rangle$ has weight $\sum_{i=1}^{k} w(v_{i-1}, v_i)$.

Example:



Solution:

$$d[s] = 0$$

$$d[a] = 2$$

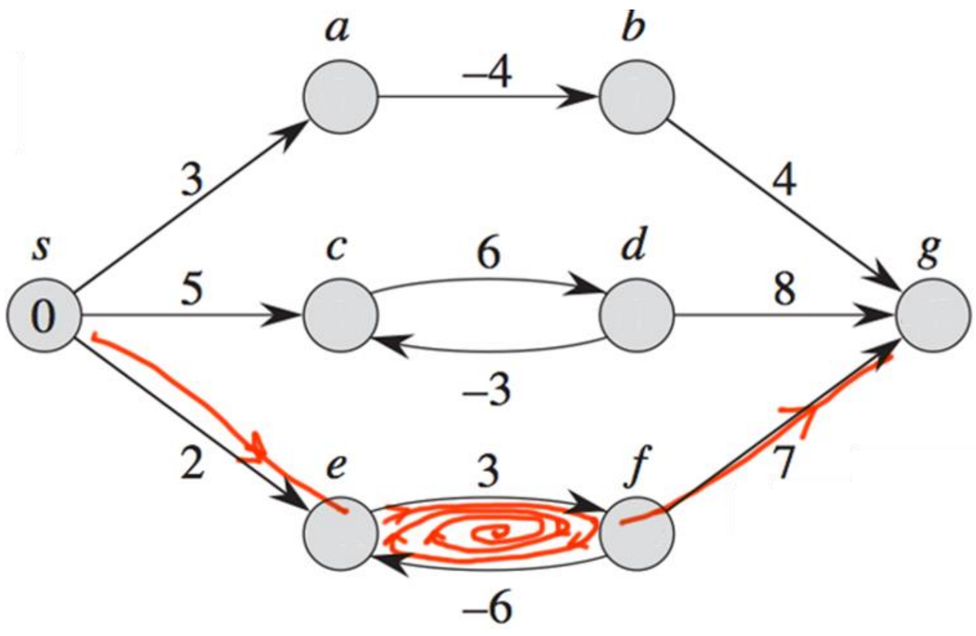
$$d[b] = 1$$

$$d[c] = 6$$

Case study III: Bellman-Ford

Assumption: There are no negative cycles. Otherwise, the question of shortest-path is ill-posed. Why?

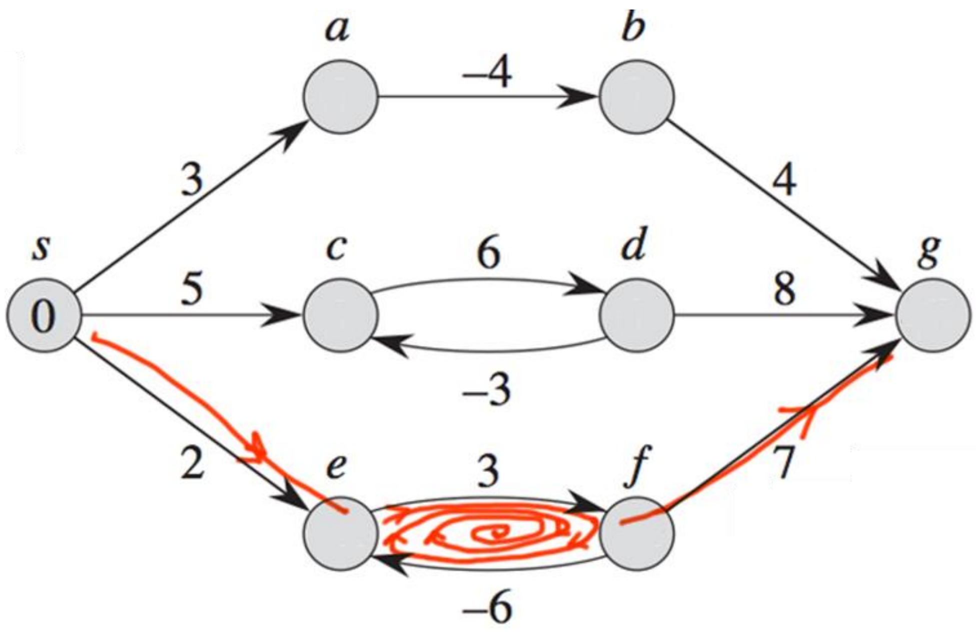
Example:



Case study III: Bellman-Ford

Assumption: There are no negative cycles. Otherwise, the question of shortest-path is ill-posed. Why?

Example:



Shortest path $-\infty$!

Case study III: Bellman-Ford

Idea: Dynamic Programming.

Step 1: Define the problem and subproblems.

Answer: Let $d[\boldsymbol{v}, \boldsymbol{k}]$ be the shortest weight from s to $\boldsymbol{v}$ using at most $\boldsymbol{k}$ edges.

Case study III: Bellman-Ford

Idea: Dynamic Programming.

Step 1: Define the problem and subproblems.

Answer: Let $d[\boldsymbol{v}, \boldsymbol{k}]$ be the shortest weight from s to $\boldsymbol{v}$ using at most $\boldsymbol{k}$ edges.

Step 2: Define the goal/output given Step 1.

It is $\boldsymbol{d}[\boldsymbol{w}, \boldsymbol{n}-\mathbf{1}]$ for shortest weight from s to w.

Case study III: Bellman-Ford

Idea: Dynamic Programming.

Step 1: Define the problem and subproblems.

Answer: Let $d[\boldsymbol{v}, \boldsymbol{k}]$ be the shortest weight from s to $\boldsymbol{v}$ using at most $\boldsymbol{k}$ edges.

Step 2: Define the goal/output given Step 1.

It is $\boldsymbol{d[w, n-1]}$ for shortest weight from s to w.

Step 3: Define the base cases

It is $d[\boldsymbol{s}, k] = 0$ for all k, $d[v, 0] = \infty$ for all $v \neq s$.

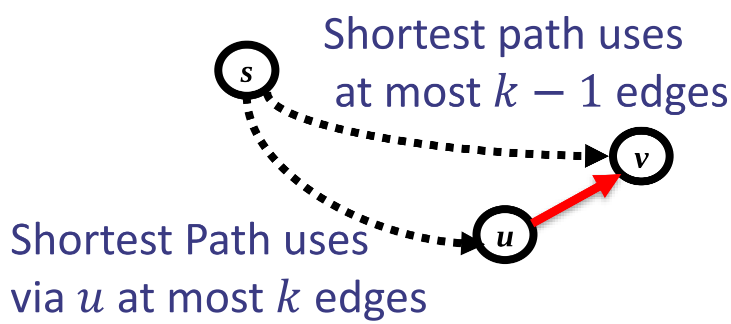
Step 4: Define the recurrence

Case study III: Bellman-Ford

Idea: Dynamic Programming.

Step 4: Define the recurrence

Shortest path from s to v uses k edges via an intermediate edge $(\boldsymbol{u}, \boldsymbol{v})$ or at most $k-1$ edges.

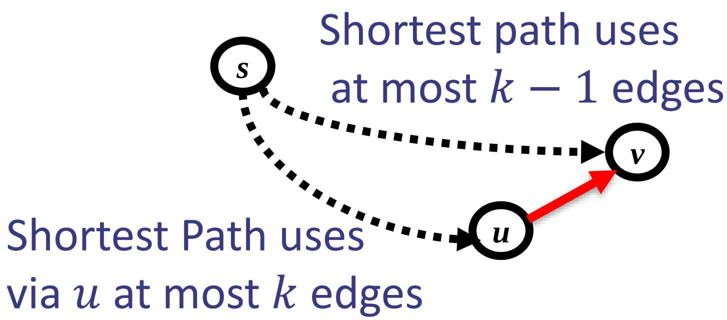


Case study III: Bellman-Ford

Idea: Dynamic Programming.

Step 4: Define the recurrence

Shortest path from s to v uses k edges via an intermediate edge $(\boldsymbol{u}, \boldsymbol{v})$ or at most $k-1$ edges.



$$d[v,k] = min(\min_{u}\{d[u,k-1] + w(u,v)\}, d[v,k-1])$$

Case study III: Bellman-Ford

Pseudocode:

Array d[][]
For $k = 0$ to $n-1$ **do**
  $\mathrm{d}[s,k] \leftarrow 0$
For each vertex $u \neq s$ **do**
  $\mathrm{d}[u,0] \leftarrow +\infty$

Initialization

For $k = 1$ to $n-1$ **do**
  For each edge (u,v) **do**
    If $\mathrm{d}[v,k] > \mathrm{d}[u,k-1] + w(u,v)$ **then**
      $\mathrm{d}[v,k] \leftarrow \mathrm{d}[u,k-1] + w(u,v)$

Bottom up filliing DP

return $\mathrm{DP}[w][n-1]$

Goal (shortest path from s to w)

Case study III: Bellman-Ford

Pseudocode: Why 2D? We can use less memory.

Array d[][]
For $k = 0$ to $n-1$ **do**
  $\mathrm{d}[s,k] \leftarrow 0$
For each vertex $u \neq s$ **do**
  $\mathrm{d}[u,0] \leftarrow +\infty$

Initialization

For $k = 1$ to $n-1$ **do**
  For each edge (u,v) **do**
    If $\mathrm{d}[v,k] > \mathrm{d}[u,k-1] + w(u,v)$ **then**
      $\mathrm{d}[v,k] \leftarrow \mathrm{d}[u,k-1] + w(u,v)$

Bottom up filliing DP

return $\mathrm{DP}[w][n-1]$

Goal (shortest path from s to w)

Case study III: Bellman-Ford

Pseudocode: Algorithm to know.

Array d[]
  $\mathrm{d}[s] \leftarrow 0$
For each vertex $u \neq s$ **do**
  $\mathrm{d}[u] \leftarrow +\infty$
For $k = 1$ to $n-1$ **do**
  For each edge (u, v) **do**
    If $\mathrm{d}[v] > \mathrm{d}[u] + w(u, v)$ **then**
      $\mathrm{d}[v] \leftarrow \mathrm{d}[u] + w(u, v)$
return $\mathrm{DP}[w]$

Initialization

Bottom up filliing DP

Goal (shortest path from s to w)

Case study III: Bellman-Ford

Pseudocode: Algorithm to know.

Array d[]
  d[s] $\leftarrow 0$
For each vertex $u \neq s$ **do**
  d[u] $\leftarrow +\infty$
For $k = 1$ to $n - 1$ **do**
  For each edge (u, v) **do**
    If d[v] $>$ d[u] $+ w(u, v)$ **then**
      d[v] $\leftarrow$ d[u] $+ w(u, v)$
return DP[w]

Initialization

Bottom up filliing DP

Relaxation of (u, v)

Goal (shortest path from s to w)

Case study III: Bellman-Ford

In words: $d[s] = 0, d[u] = +\infty$ for $u \neq s$.

For $n - 1$ times, relax all the edges (u, v).

Relaxation of (u, v)

if $\mathrm{d}[v] > \mathrm{d}[u] + w(u, v)$ **then**
$\mathrm{d}[v] \leftarrow \mathrm{d}[u] + w(u, v)$

Running time $\Theta(|V| \cdot |E|)$

Case study III: Bellman-Ford

In words: $d[s] = 0, d[u] = +\infty$ for $u \neq s$.

For $n - 1$ times, relax all the edges (u, v).

Relaxation of (u, v)

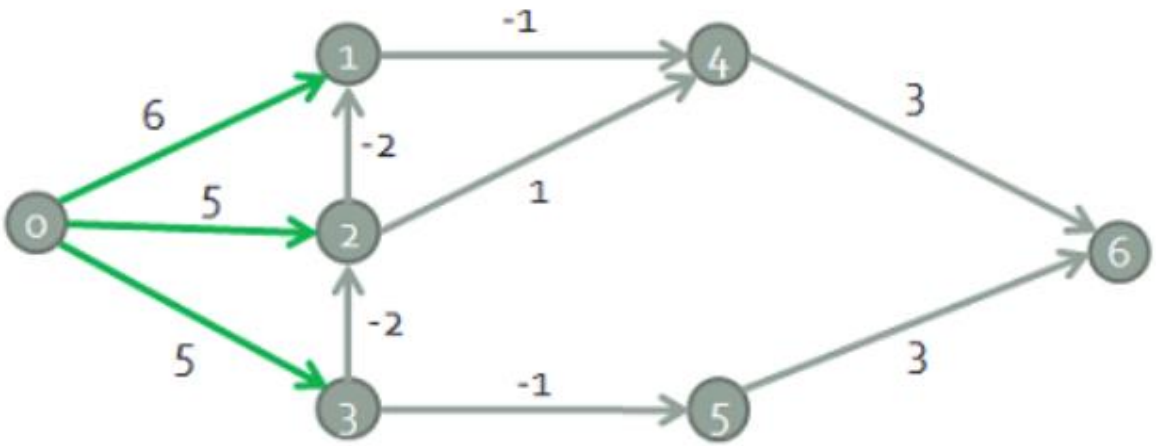
If $\mathrm{d}[v] > \mathrm{d}[u] + w(u, v)$ **then**
$\mathrm{d}[v] \leftarrow \mathrm{d}[u] + w(u, v)$

Running time $\Theta(|V| \cdot |E|)$

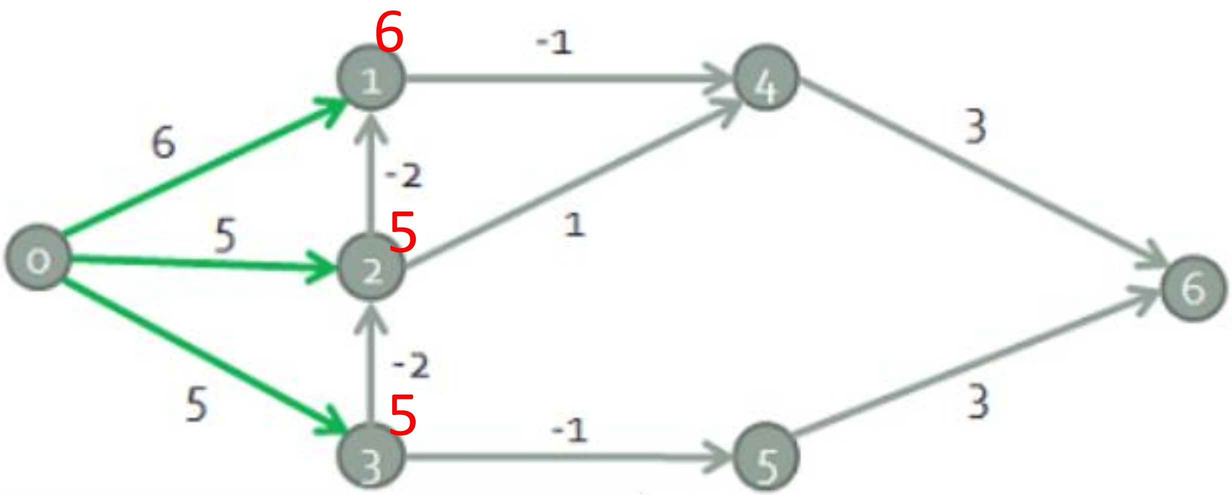
Property: Suppose we relax all edges one more time. If $d[]$ decreases for a vertex then there is a negative cycle. If $d[]$ remains the same, no negative cycle.

Case study III: Bellman-Ford

Find the shortest weight path from node 0.

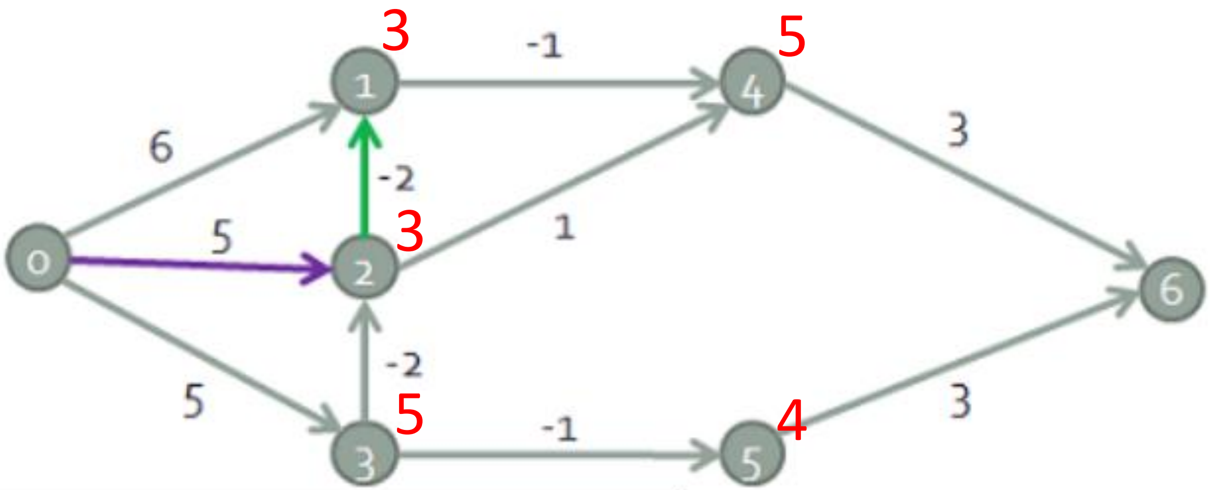


Case study III: Bellman-Ford



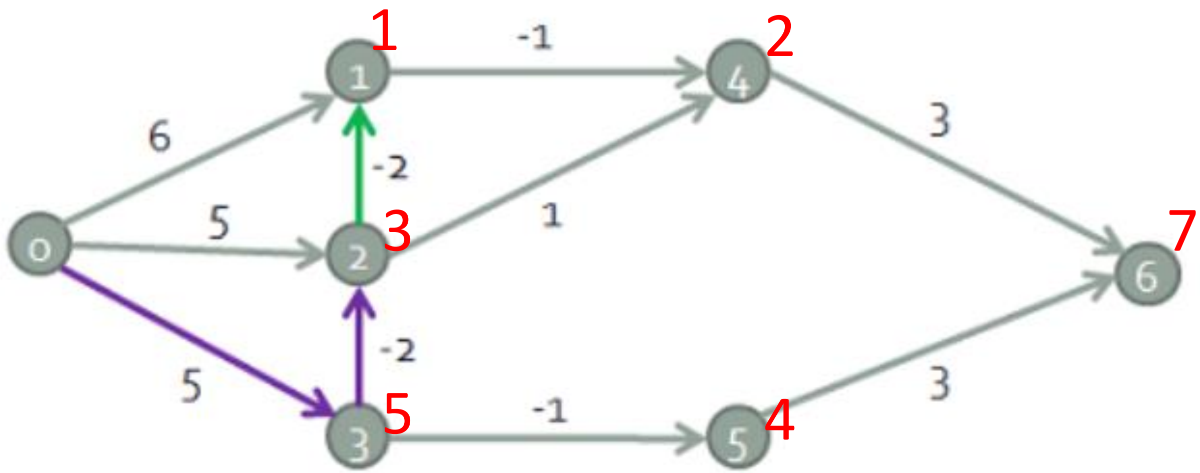
k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2						
3						
4						
5						
6						

Case study III: Bellman-Ford



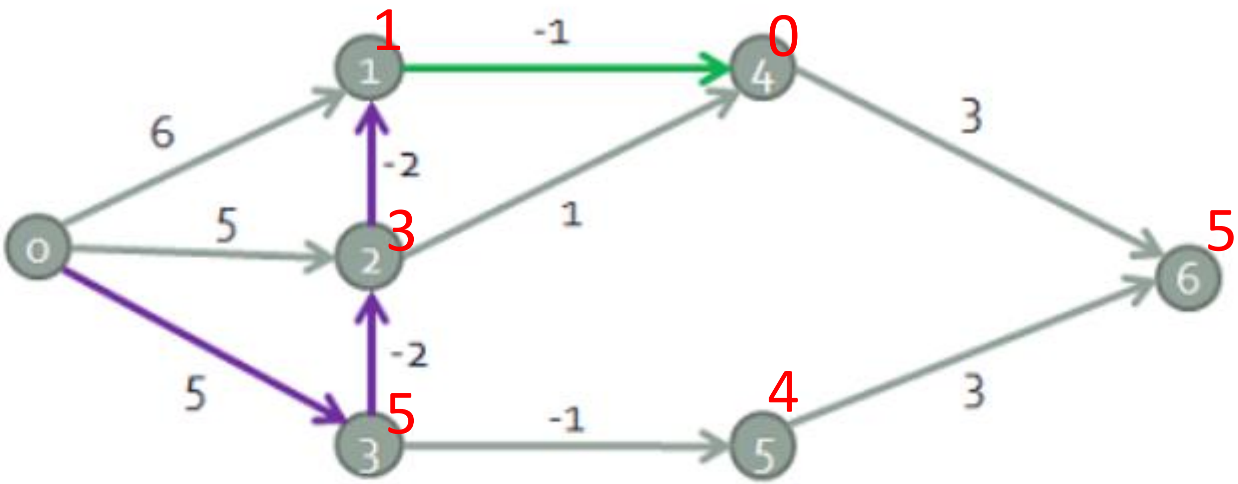
k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2	3	3	5	5	4	∞
3						
4						
5						
6						

Case study III: Bellman-Ford



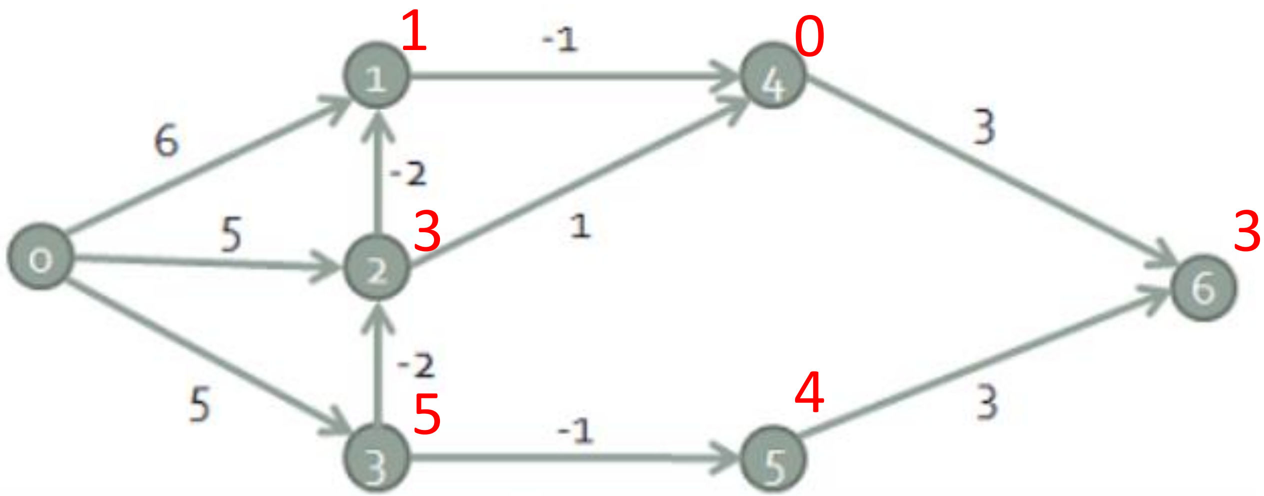
k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2	3	3	5	5	4	∞
3	1	3	5	2	4	7
4						
5						
6						

Case study III: Bellman-Ford



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

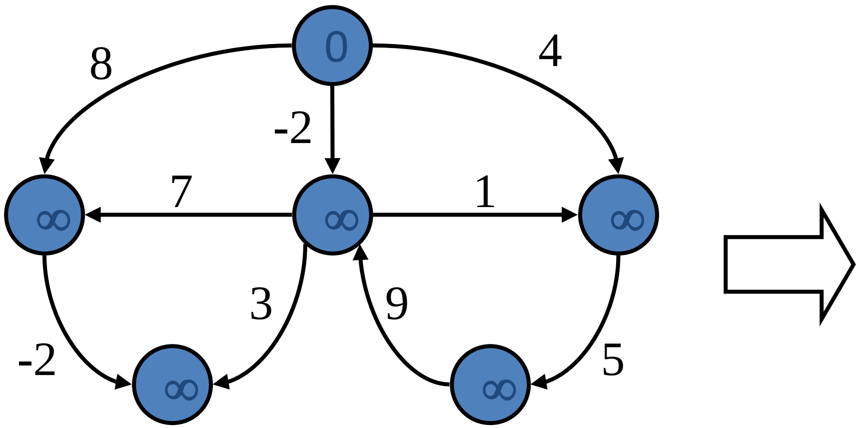
Case study III: Bellman-Ford



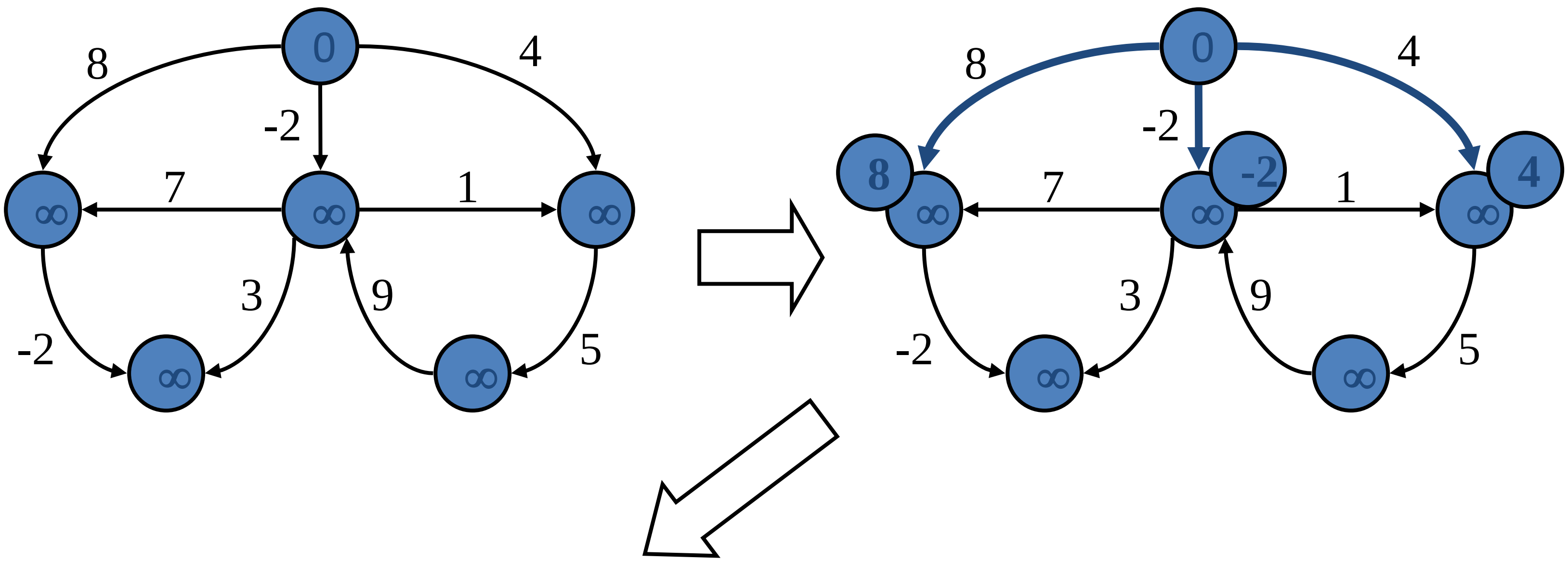
k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2	3	3	5	5	4	∞
3	1	3	5	2	4	7
4	1	3	5	0	4	5
5	1	3	5	0	4	3
6	1	3	5	0	4	3

Case study III: Bellman-Ford

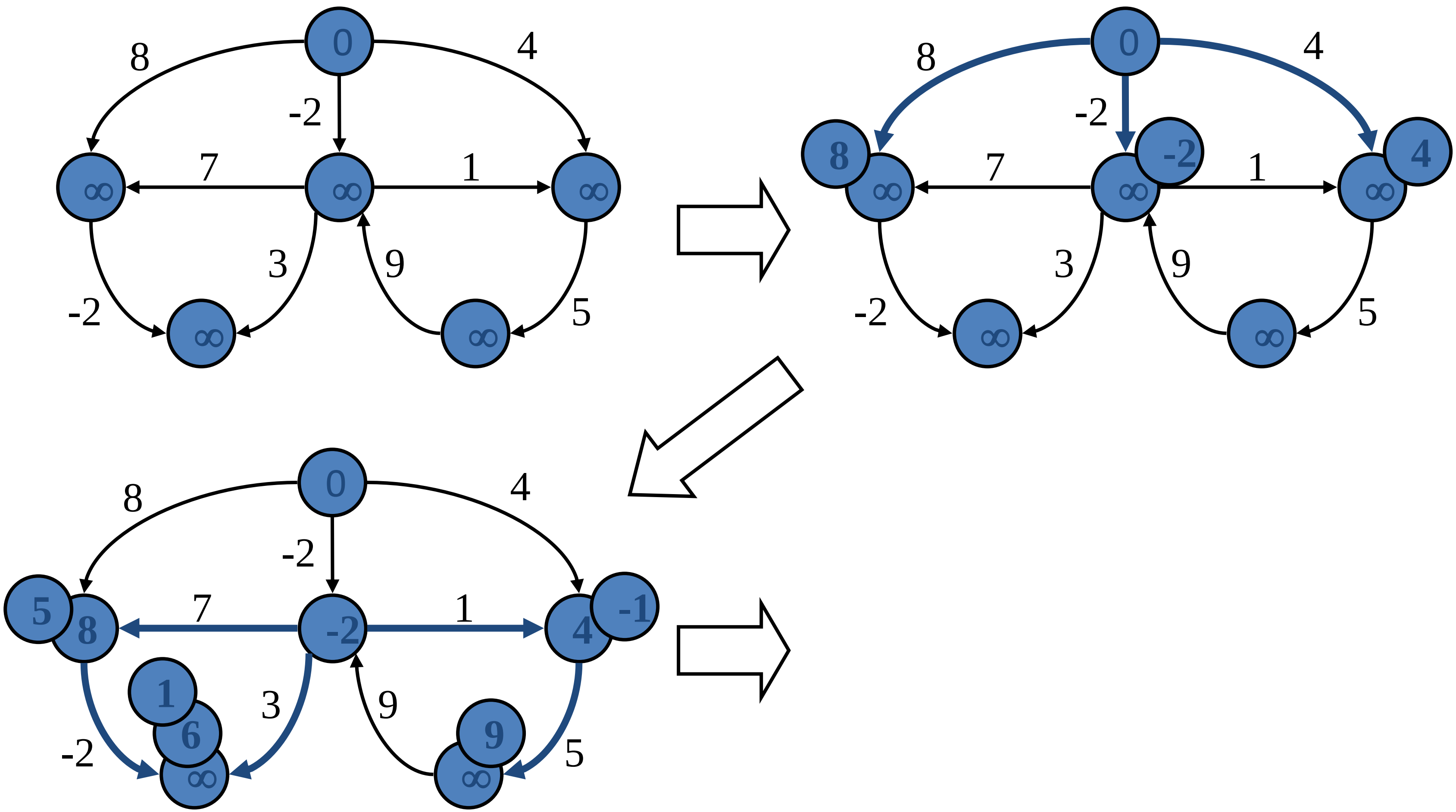
Find the shortest weight path from node 0.



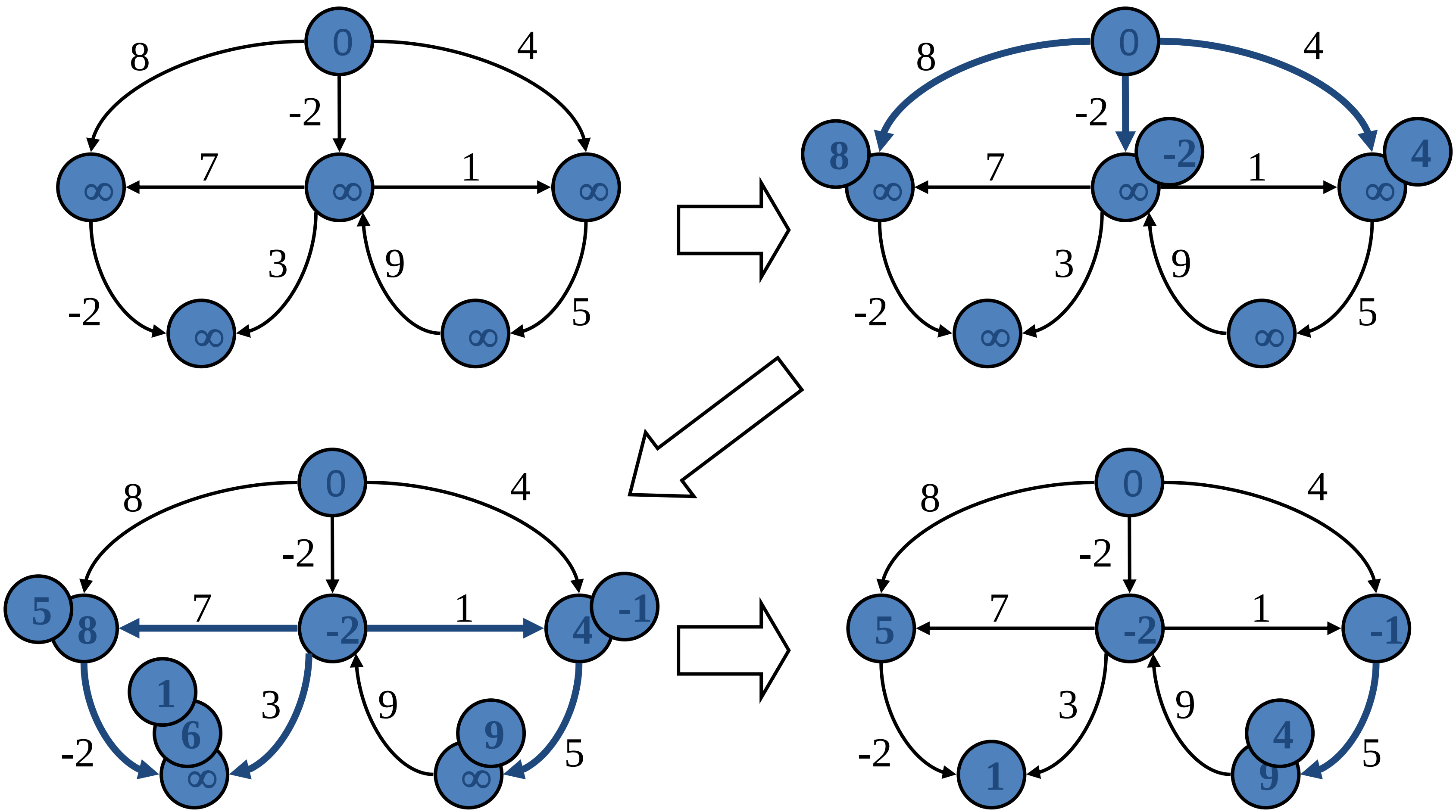
Case study III: Bellman-Ford



Case study III: Bellman-Ford



Case study III: Bellman-Ford

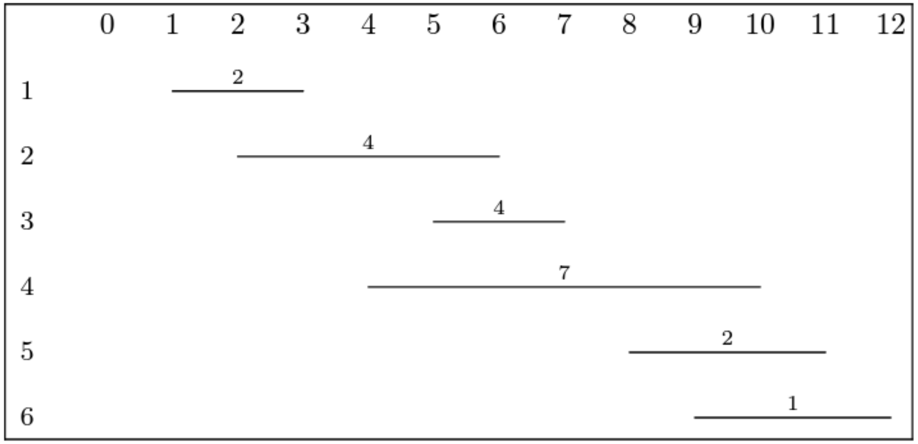


Case study IV: Interval Scheduling

Problem: You are given a collection of n intervals represented by start time, finish time, and value: (s_j, f_j, v_j). Find a non-overlapping set of intervals with maximum total value.

Example:

j	$s(j)$	$f(j)$	$v(j)$
1	1	3	2
2	2	6	4
3	5	7	4
4	4	10	7
5	8	11	2
6	9	12	1

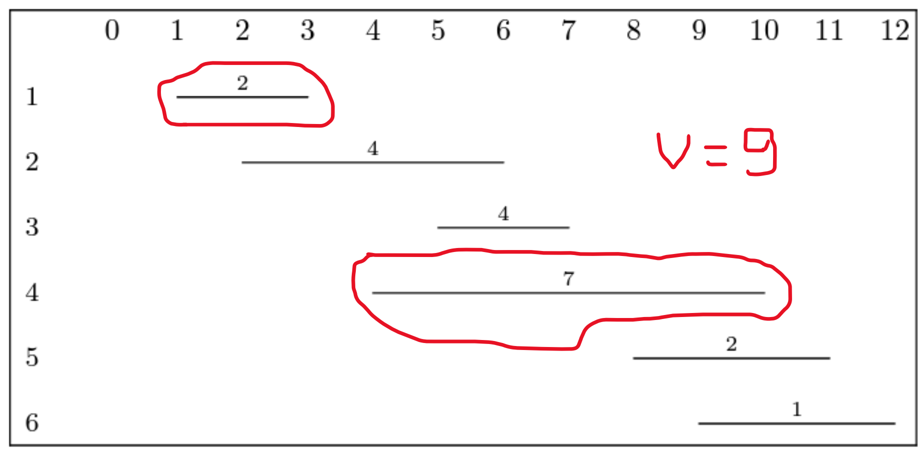


Case study IV: Interval Scheduling

Problem: You are given a collection of n intervals represented by start time, finish time, and value: (s_j, f_j, v_j). Find a non-overlapping set of intervals with maximum total value.

Example:

j	$s(j)$	$f(j)$	$v(j)$
1	1	3	2
2	2	6	4
3	5	7	4
4	4	10	7
5	8	11	2
6	9	12	1



Case study IV: Interval Scheduling

Step 1: Define the problem and subproblems.

Answer: Let $DP[j]$ be the maximum value that can be obtained from a set of non-overlapping intervals with indices in the range $\{1, \dots, j\}$

Case study IV: Interval Scheduling

Step 1: Define the problem and subproblems.

Answer: Let $DP[j]$ be the maximum value that can be obtained from a set of non-overlapping intervals with indices in the range $\{1, \dots, j\}$

Step 2: Define the goal/output given Step 1.

It is $\boldsymbol{DP[n]}$.

Case study IV: Interval Scheduling

Step 1: Define the problem and subproblems.

Answer: Let $DP[j]$ be the maximum value that can be obtained from a set of non-overlapping intervals with indices in the range $\{1, \ldots, j\}$

Step 2: Define the goal/output given Step 1.

It is $\boldsymbol{DP[n]}$.

Step 3: Define the base cases

It is $DP[0] = 0$.

Step 4: Define the recurrence

Case study IV: Interval Scheduling

Step 4: Define the recurrence

Interval j belongs to the optimal solution or **not**.

$$DP[j] = \max(\mathbf{DP[\$] + v_j}, \mathbf{DP[j-1]})$$

What is **$** ?

Case study IV: Interval Scheduling

Step 4: Define the recurrence

Interval j belongs to the optimal solution or **not**.

$$DP[j] = \max(\mathbf{DP[\$] + v_j}, \mathbf{DP[j-1]})$$

$ should be the interval with highest index in $\{1, \dots, j-1\}$ that does not intersect with j (since j is chosen).

Let $p[j]$ be the highest index in $\{1, \dots, j-1\}$ that does not intersect with j. Then the recurrence becomes

$$DP[j] = \max(\mathbf{DP[p[j]] + v_j}, \mathbf{DP[j-1]})$$

Case study IV: Interval Scheduling

Pseudocode:

$\text{Array DP}[]$

$\text{DP}[0] \leftarrow 0$ — Initialization

$\textbf{For } k = 1 \text{ to } n \textbf{ do}$

$\quad \text{DP}[k] \leftarrow \max(\text{DP}[k-1], \text{DP}[p[k]] + v[k])$ — Bottom up filliing DP

$\textbf{return } \text{DP}[n]$ — Goal

Case study IV: Interval Scheduling

Pseudocode:

Array DP[]

$\text{DP}[0] \leftarrow 0$

For $k = 1$ to n **do**

  $\text{DP}[k] \leftarrow \max(\text{DP}[k-1], \text{DP}[p[k]] + v[k])$

return $\text{DP}[n]$

Initialization

Bottom up filliing DP

Goal

Question: How can we compute $p[j]$ for $1 \leq j \leq n$ in $\Theta(n \log n)$ time?

Case study IV: Interval Scheduling

Question: How can we compute $p[j]$ for $1 \leq j \leq n$ in $\Theta(n \log n)$ time?

Answer:

- Sort first the intervals in increasing order of finishing times.

Case study IV: Interval Scheduling

Question: How can we compute $p[j]$ for $1 \leq j \leq n$ in $\Theta(n \log n)$ time?

Answer:

- Sort first the intervals in increasing order of finishing times.
- For every j, do binary search to find the interval before j with finishing time at most s_j