Graph theory

Routing problem

Z.GUELLIL

Chapter 4: Shortest Path Problem

- Introduction to the shortest path problem
- Dijkstra
- Bellman-Ford Algorithm
- Floyd- Warshall algorithm

Problem Statement

- The routing problem is a central challenge in many graph theory applications.
- Representing networks such as routes, Internet connections, or social interactions.
- The goals is to minimize some metric, such as distance, time, or cost.
- Routing problems apply to both directed and undirected graphs.
- This is crucial in a variety of areas:
  - Transportation networks: Minimize the distance or travel time between two points.
  - Telecommunications and computer networks: Optimizing bandwidth usage and reducing transmission delays.
  - Logistics and distribution: Finding optimal routes for transporting goods while minimizing costs.

Problem Statement

- Let G=(V,E) be a graph where:
- Each edge $\boldsymbol{e} \in \boldsymbol{E}$ may have an associated *weight* ***w(e)***, representing a cost metric such as distance, time, or expense.
- The weights may be non-negative or, in some cases, include negative values, depending on the application.
- The routing problem involves finding a path or a set of paths $P = (v_1, v_2, \dots, v_k) \in G$ that connects a *source node* $s \in V$ to a *destination node* $t \in V$, such that a specified *objective function* is minimized.
- The objective function typically minimizes the **path cost**: The sum of edge weights along the path, $\sum_{\{e \in P\}} w(e)$.

Problem Statement

- there are indeed three main types of shortest path problems :

1. **Shortest Path Between Two Specific Vertices**: Finding the shortest path from vertex **i** to vertex j.
2. **Shortest Path from a Single Vertex to All Other Vertices**: Finding the shortest paths from a single *source* vertex i to *all other vertices* in the graph.
   - Common algorithms for the first and second problem include:
     - **Dijkstra's Algorithm** (for non-negative edge weights)
     - **Bellman-Ford Algorithm** (handles negative weights and detects negative cycles)
3. **Shortest Paths Between All Pairs of Vertices**: Finding the shortest paths bet ween every pair of vertices in the graph.

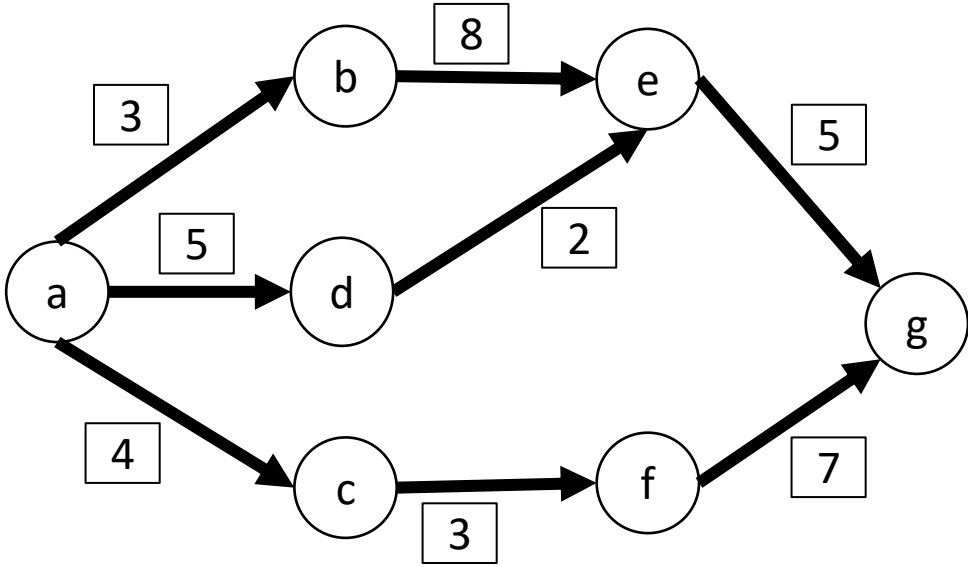
- The Floyd-Warshall algorithm is specifically designed for this problem.

Dijkstra's algorithm

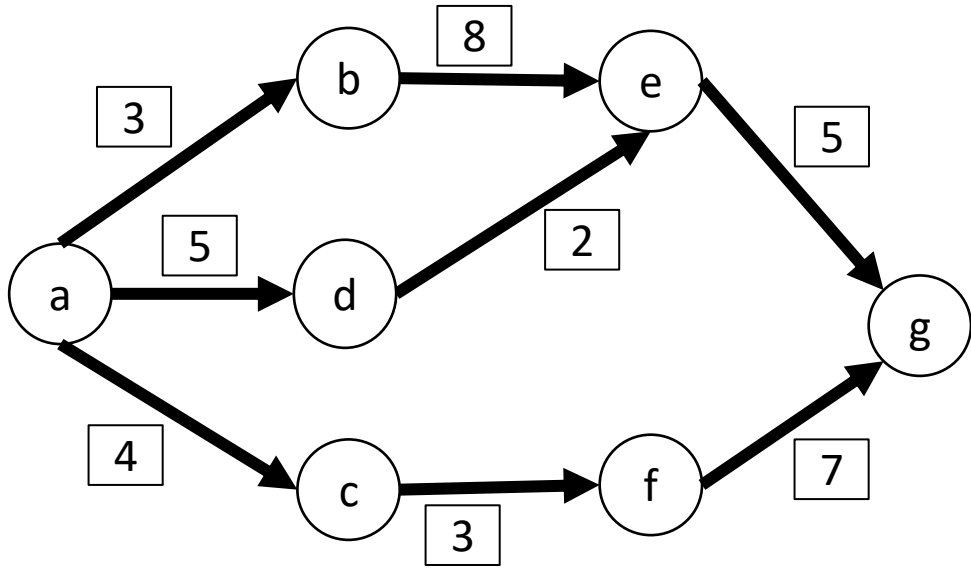
- The algorithm due to Dijkstra is based on the following principle: If the shortest path connecting E to S passes through the vertices S_1, S_2, …, S_k then, the different steps are also the shortest paths connecting E to the different vertices S_1, S_2, …, S_k.
- We construct the desired path step by step by choosing at each iteration of the algorithm, a vertex S_i of the graph among those which have not yet been processed, such that the provisionally known length of the shortest path going from E to S_i is the shortest possible.

Dijkstra’s Algorithm

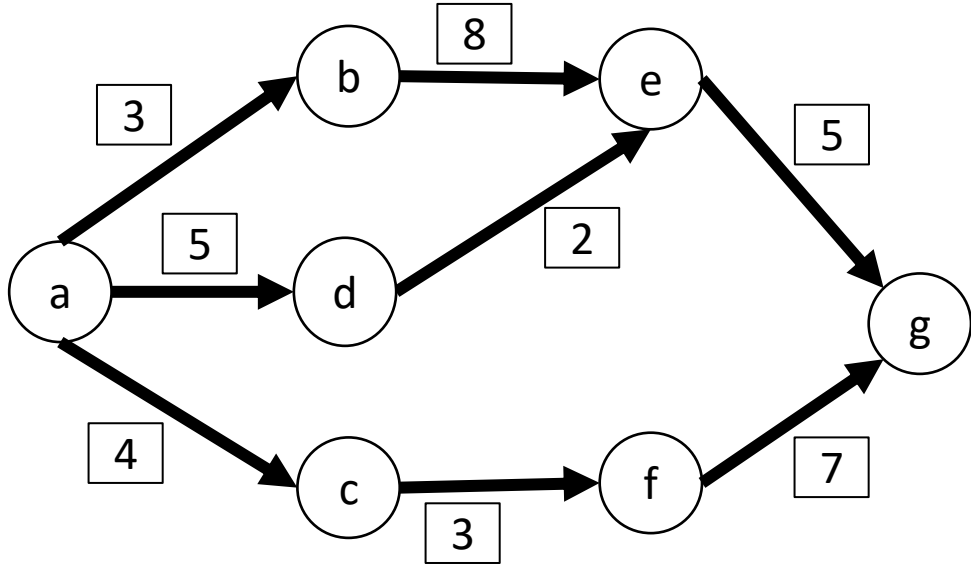
- **Initialization of the algorithm:**
  - Set the weight (distance) of the source vertex to 0.
  - Assign a temporary weight of ∞ to all other vertices.
- **Repeat** the following operations **until** all vertices have definitive weights (shortest distances) from the source.
- **Select the Vertex with Minimum Weight:**
  - Among vertices with temporary weights, choose the vertex $\underline{X}$ that has the minimum weight w($\underline{X}$).
  - Fix $\underline{X}$ by marking it as permanently assigned the weight w($\underline{X}$).
- **Update Adjacent Vertices:**
  - For each unmarked vertex **Y** adjacent to the recently fixed vertex **X** :
  - Calculate the tentative weight **s**, which is the sum of the weight of **x** (i.e., w(**X**)) and the weight of the edge connecting **X to Y**.
  - If **s** is less than **Y** 's current temporary weight, update **Y**'s weight to **s** and note **X** as the predecessor of **Y** (to trace the path back to the source).
- **When the vertex is finally marked**
- The shortest path from **Source** to **Destination** is obtained by writing the path from left to right starting from the end.


a
b
c
d
e
f
g
3
8
5
5
2
4
3
7

a	b	c	d	e	f	g

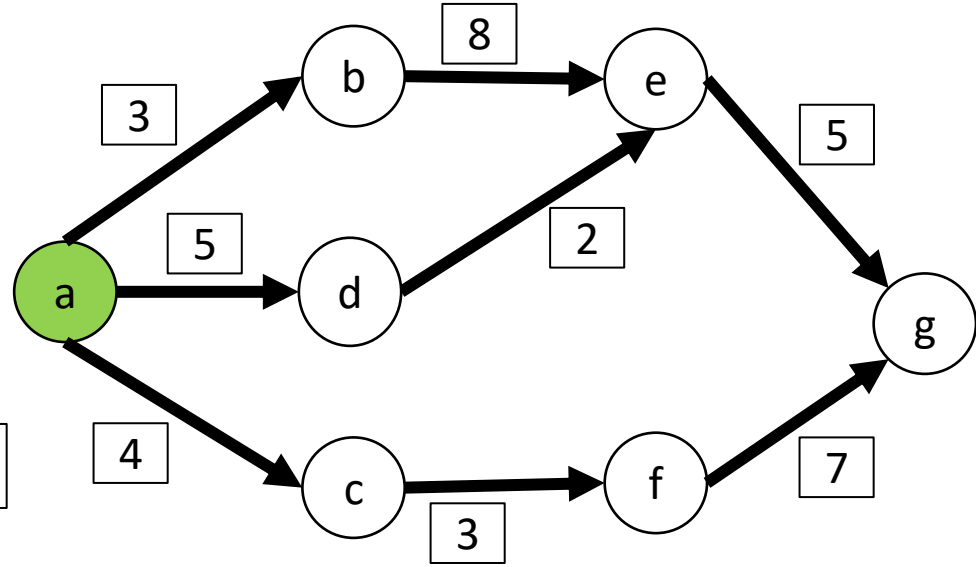


a	b	c	d	e	f	g
0	∞	∞	∞	∞	∞	∞

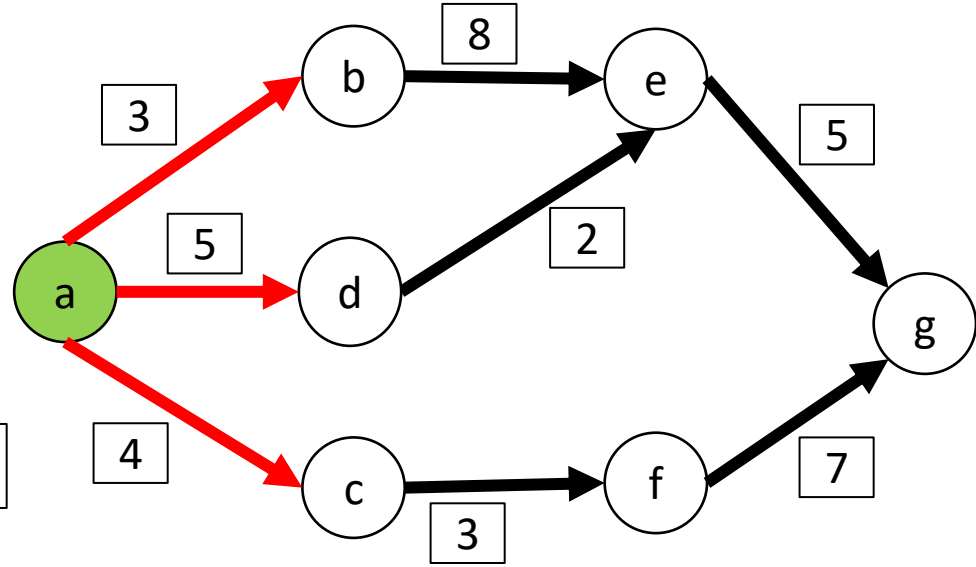


a	b	c	d	e	f	g
0	∞	∞	∞	∞	∞	∞

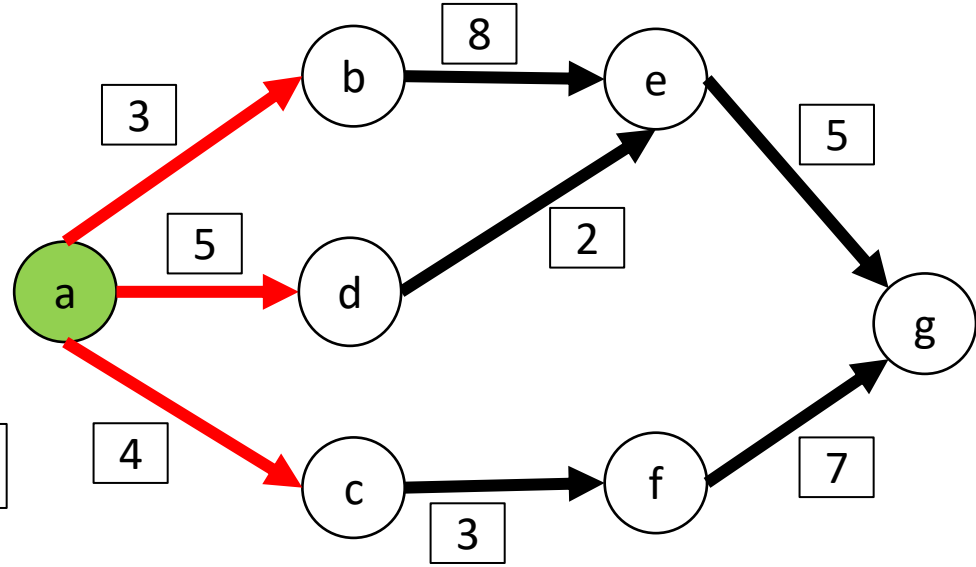
a(0)



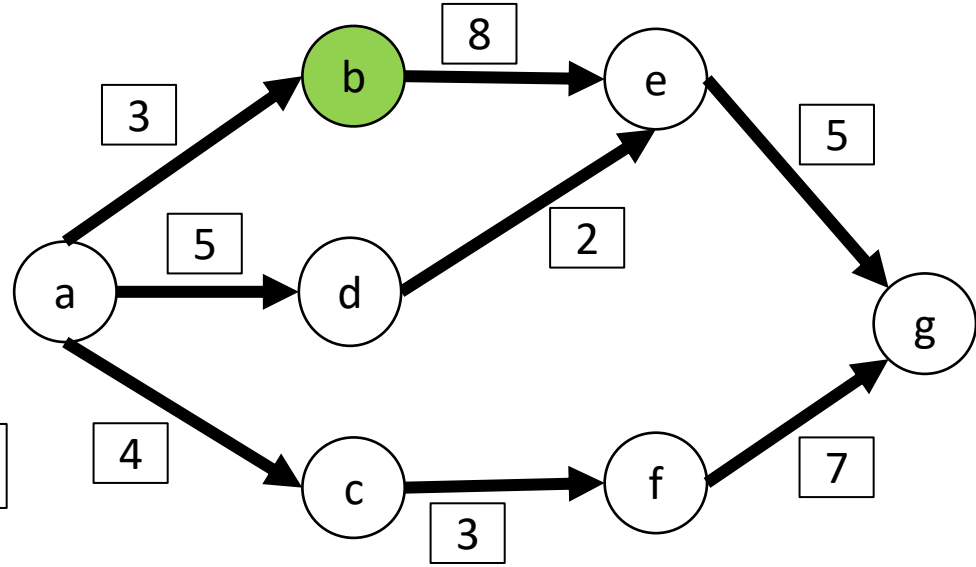
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	



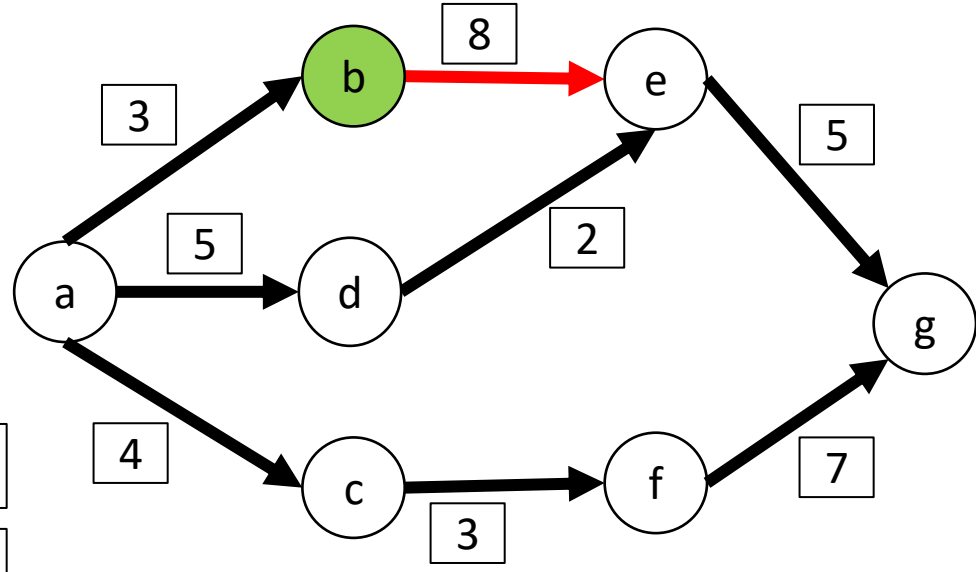
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	



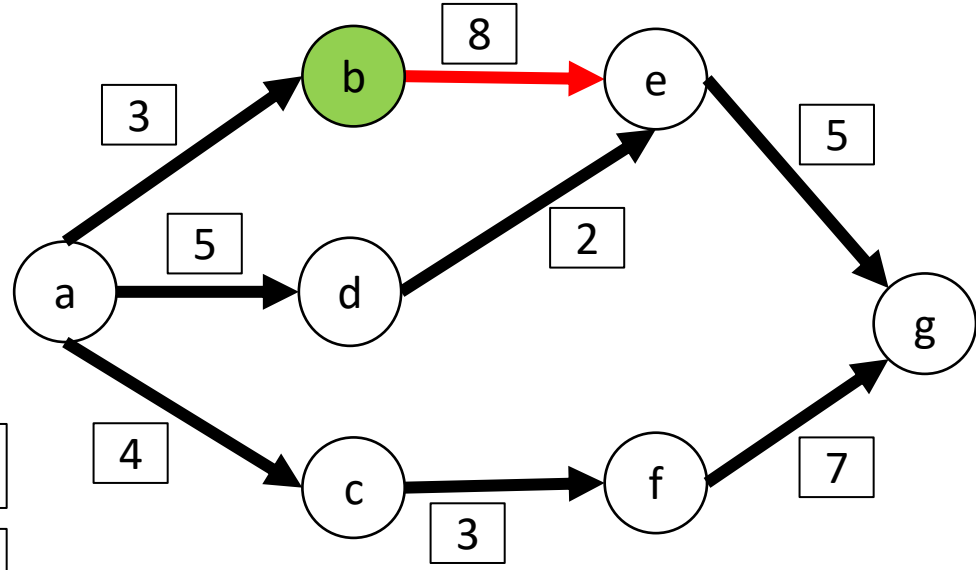
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	



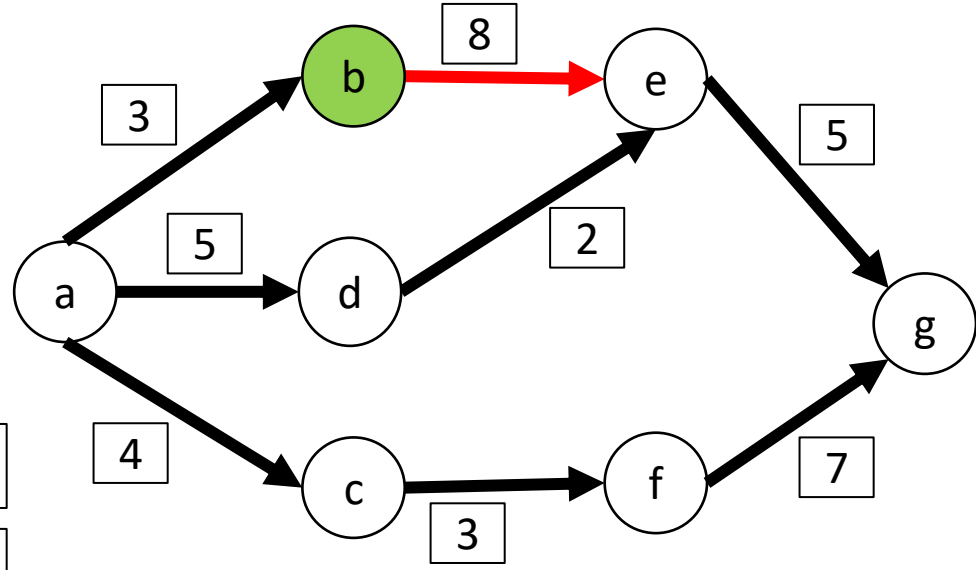
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)



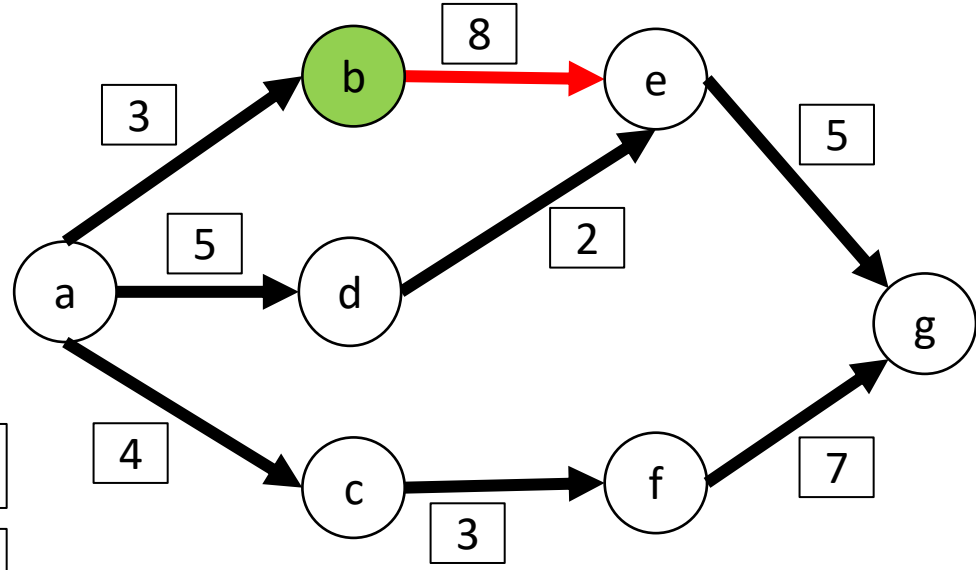
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	∞	∞	∞	



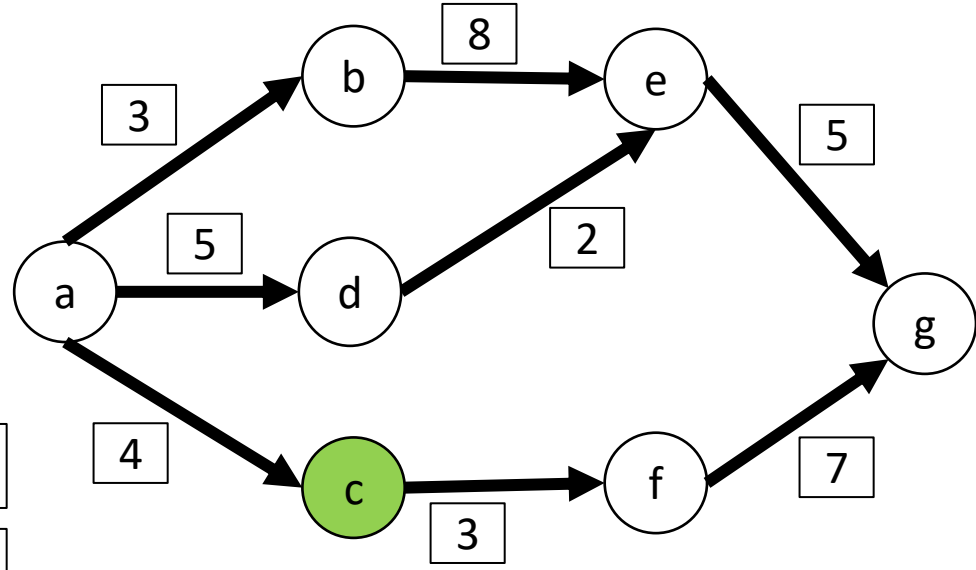
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	B(3+8)	∞	∞	



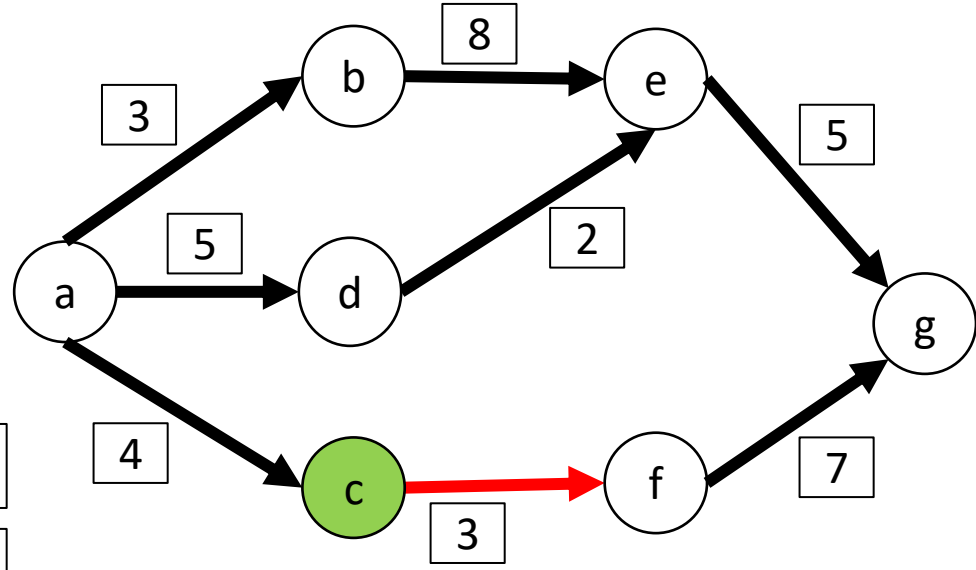
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	



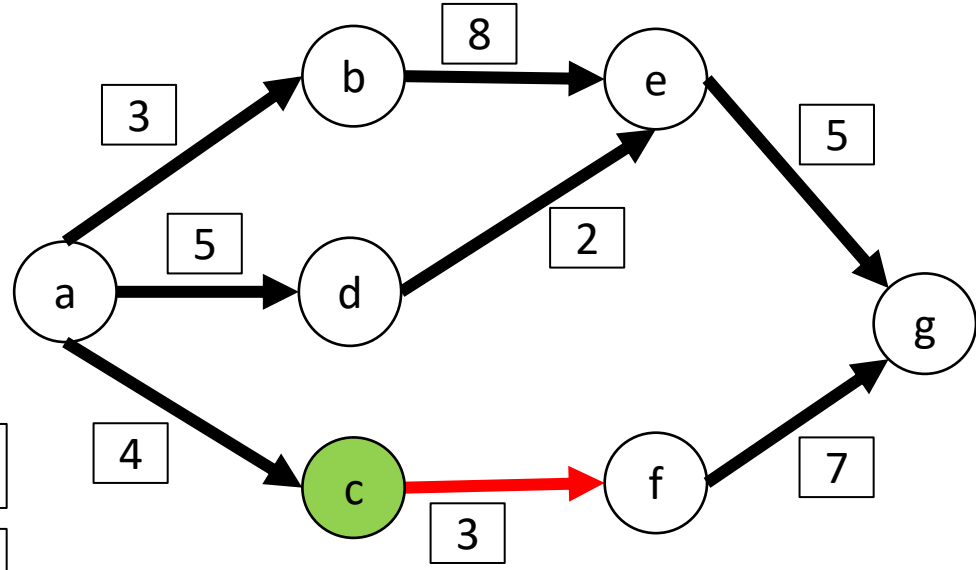
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)



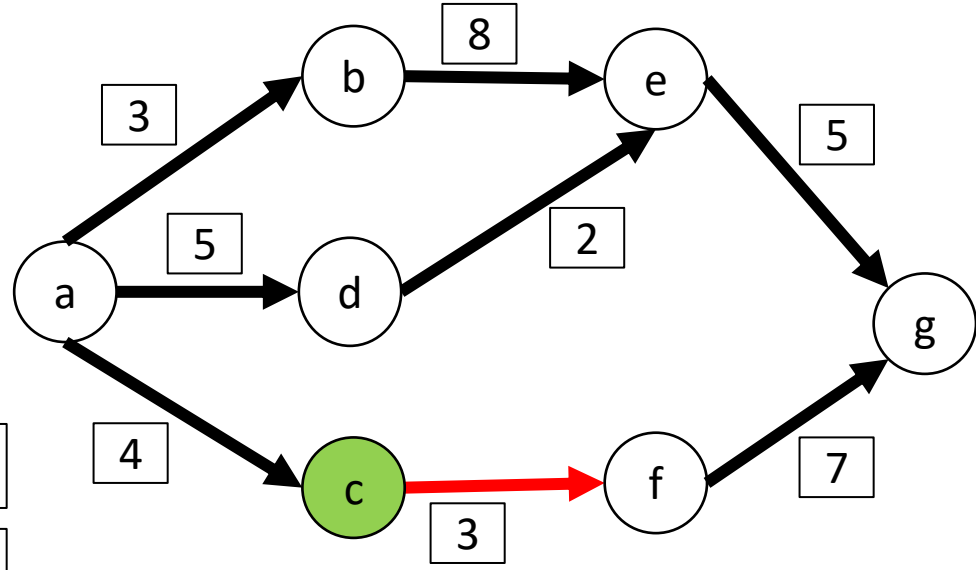
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	∞	∞	



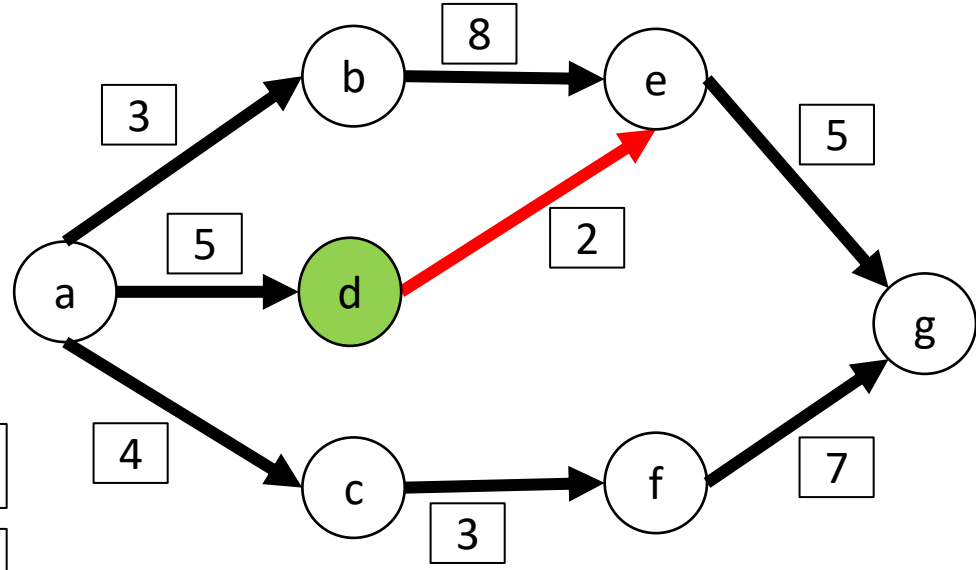
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(3+4)	∞	



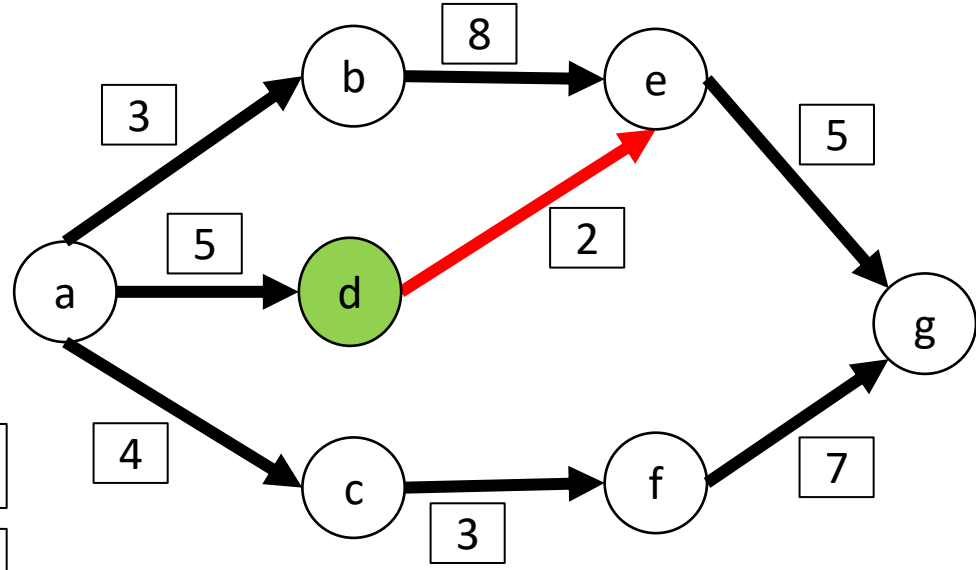
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	

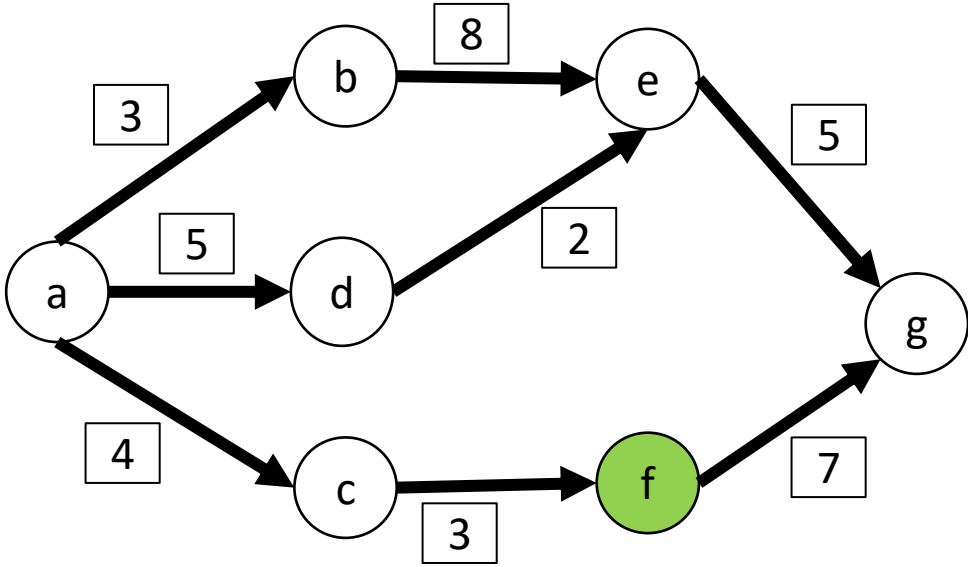


a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	d(5)
				b(11)	C(7)	∞	



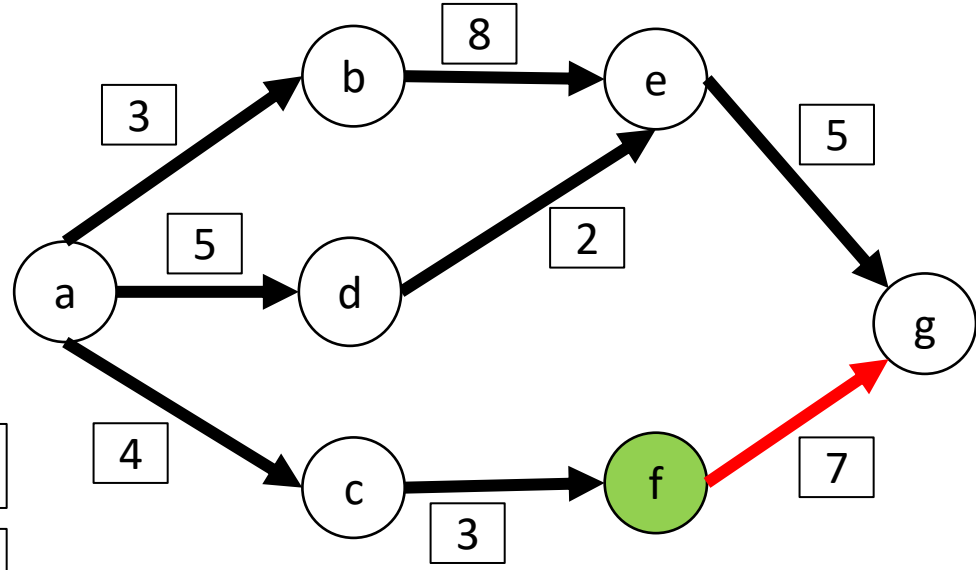
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	d(5)
				d(7)	C(7)	∞	



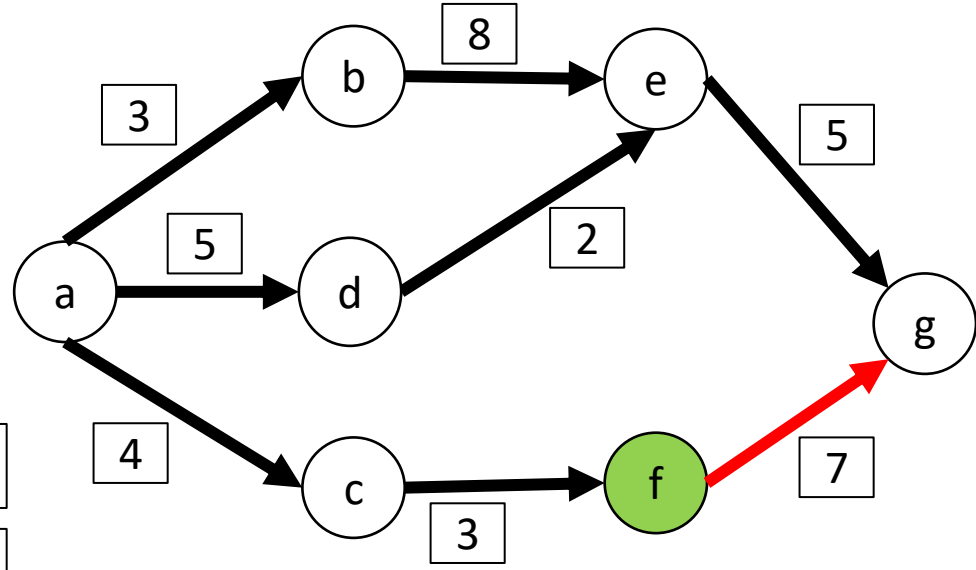


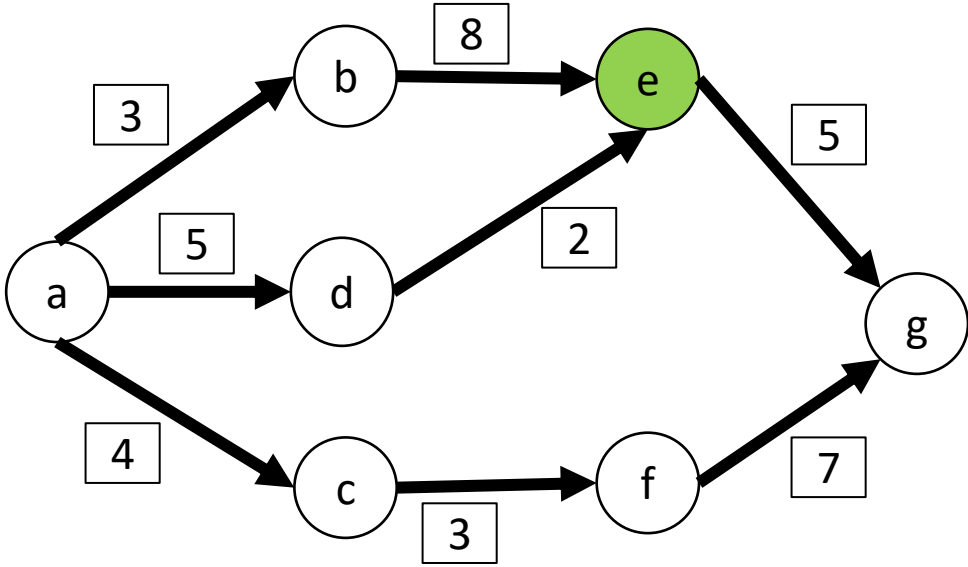
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	d(5)
				d(7)	C(7)	∞	f(7)

a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	d(5)
				d(7)	C(7)	∞	f(7)
				d(7)		∞	

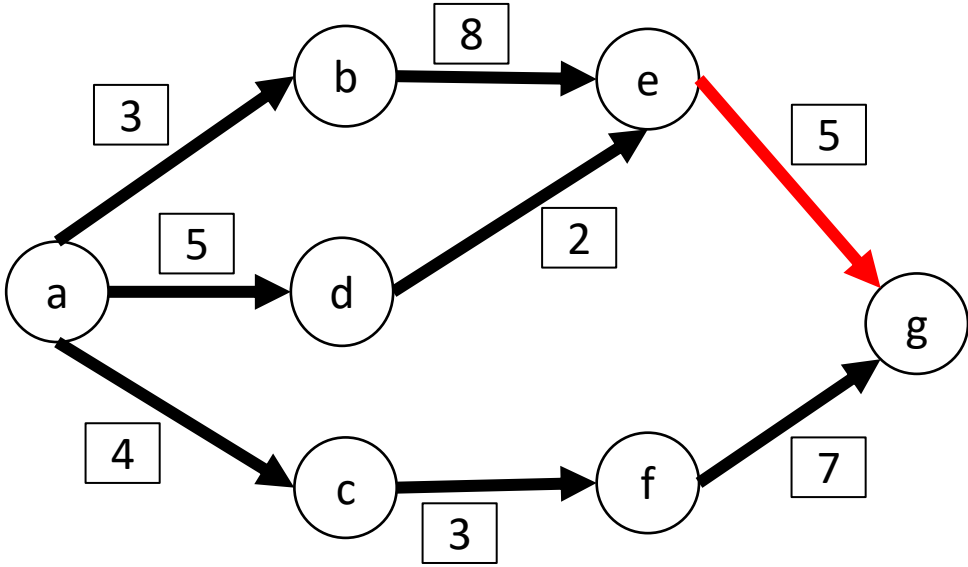


a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	d(5)
				d(7)	C(7)	∞	f(7)
				d(7)		f(14)	

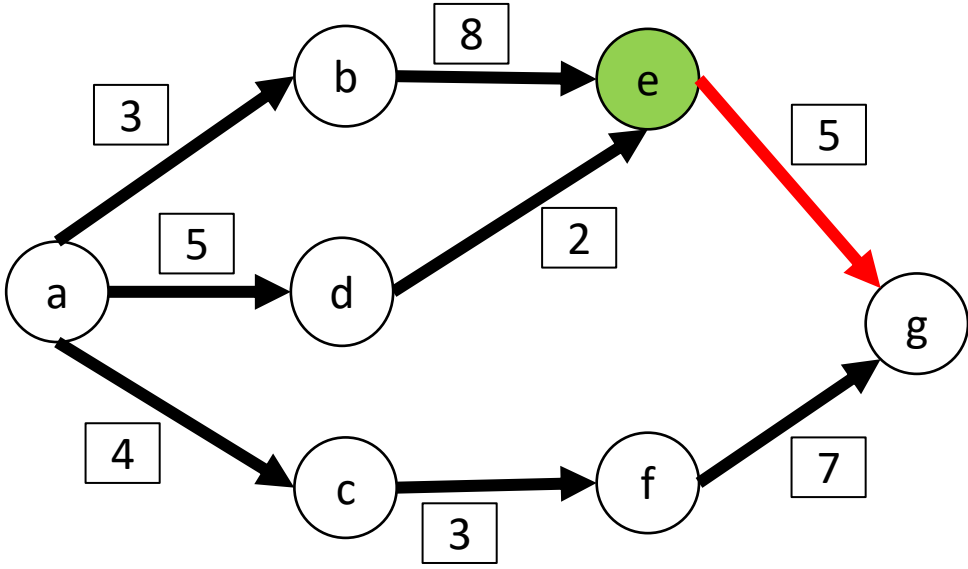




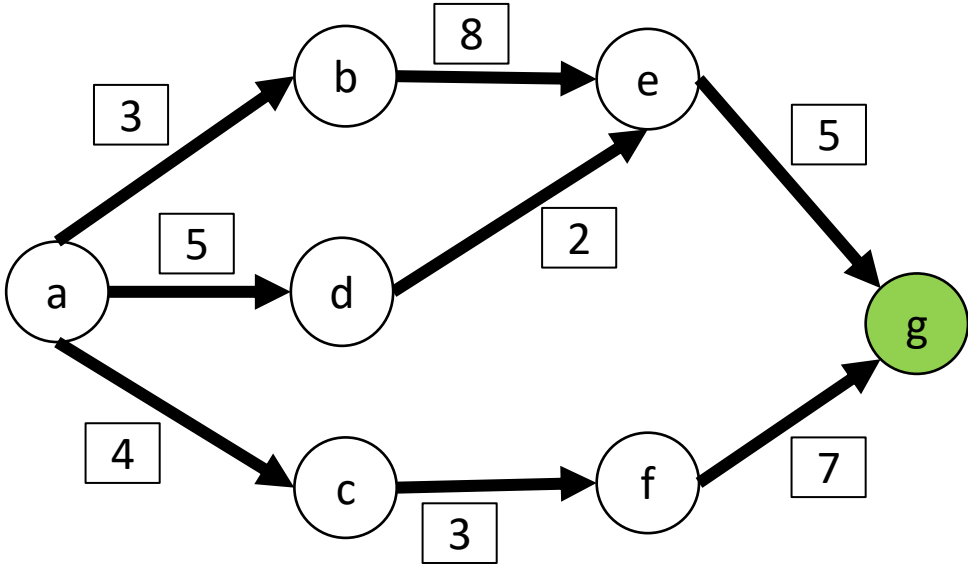
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	d(5)
				d(7)	C(7)	∞	f(7)
				d(7)		f(14)	e(7)



a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	d(5)
				d(7)	C(7)	∞	f(7)
				d(7)		f(14)	e(7)
						f(14)	

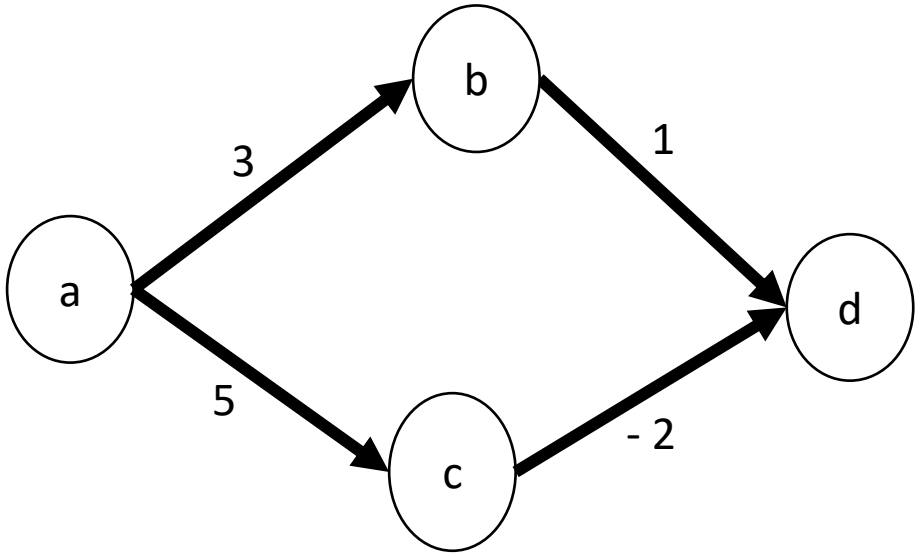


a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	d(5)
				d(7)	C(7)	∞	f(7)
				d(7)		f(14)	e(7)
						e(12)	



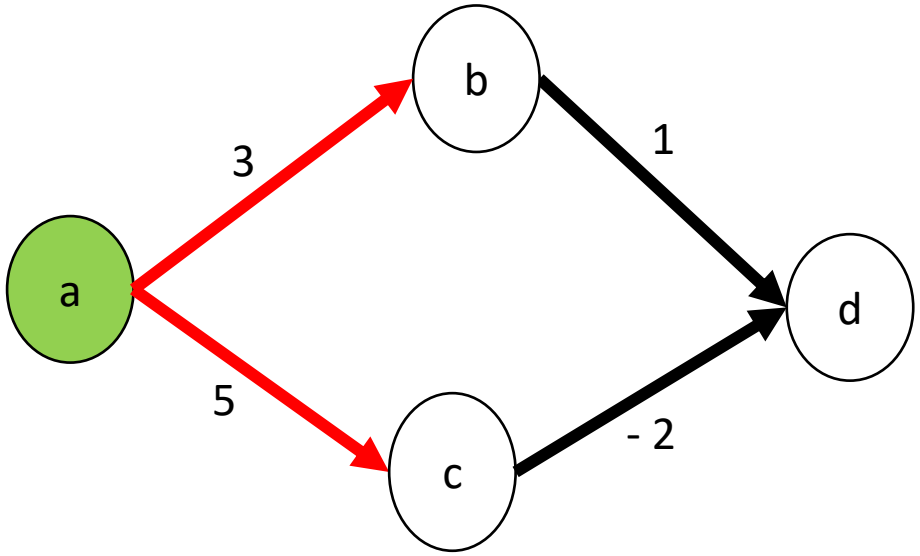
a	b	c	d	e	f	g	
0	∞	∞	∞	∞	∞	∞	a(0)
	a(3)	a(4)	a(5)	∞	∞	∞	b(3)
		a(4)	a(5)	b(11)	∞	∞	c(4)
			a(5)	b(11)	C(7)	∞	d(5)
				d(7)	C(7)	∞	f(7)
				d(7)		f(14)	e(7)
						e(12)	g(12)

Example



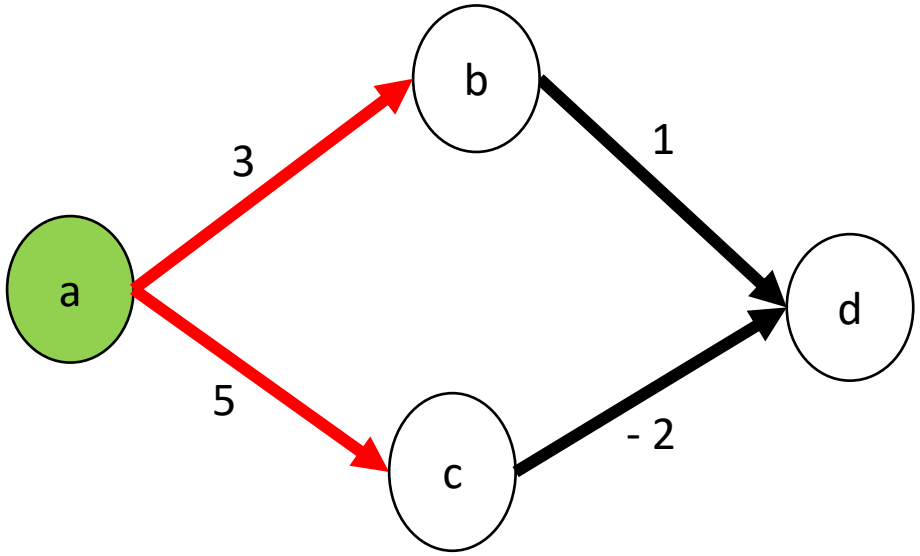
Example

a	b	c	d	
0	∞	∞	∞	a(0)



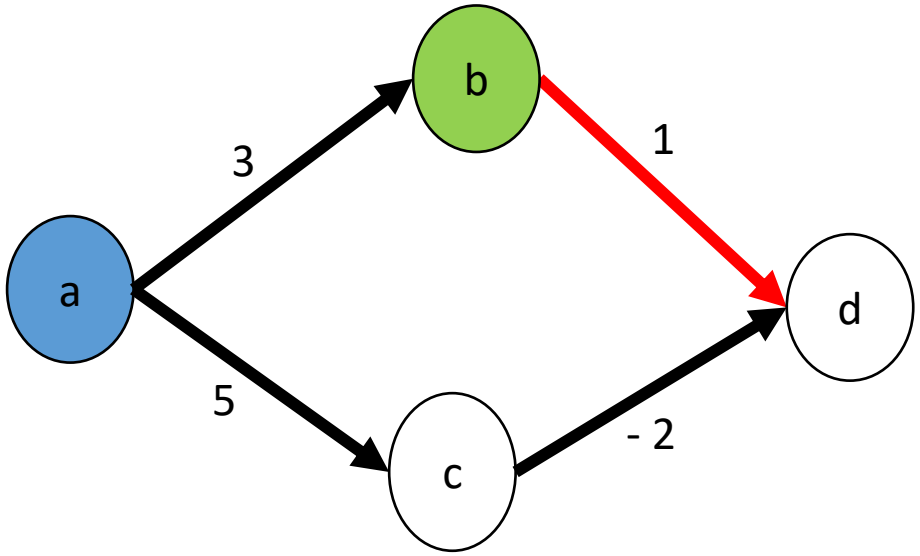
Example

a	b	c	d	
0	∞	∞	∞	a(0)
	a (3)	a(5)	∞	b (3)



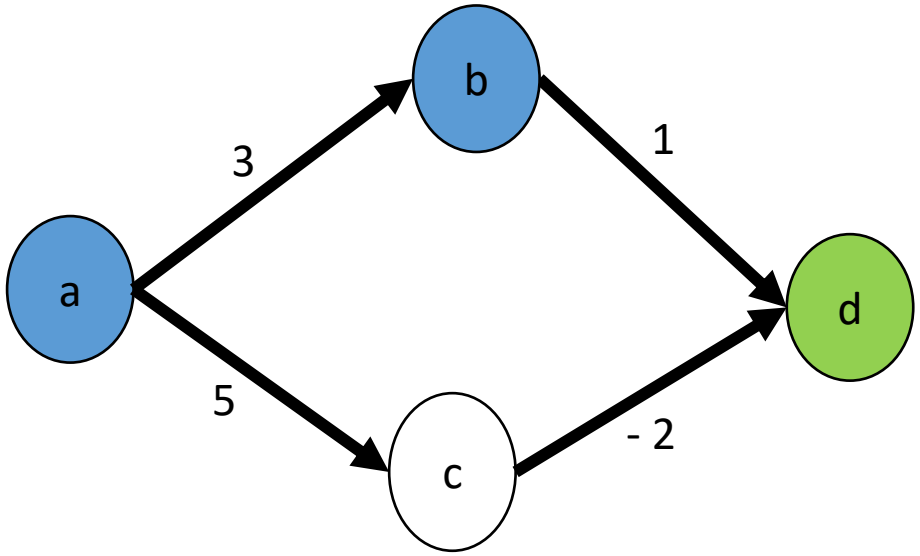
Example

a	b	c	d	
0	∞	∞	∞	a(0)
	a (3)	a(5)	∞	b (3)
		a(5)	b(4)	d (4)



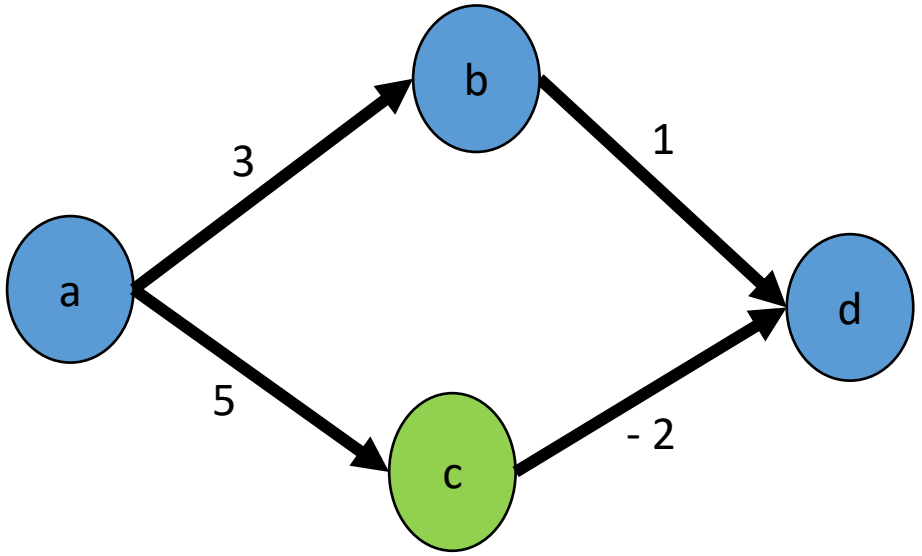
Example

a	b	c	d	
0	∞	∞	∞	a(0)
	a (3)	a(5)	∞	b (3)
		a(5)	b(4)	d (4)
		a(5)		c (5)



Example

a	b	c	d	
0	∞	∞	∞	a(0)
	a (3)	a(5)	∞	b (3)
		a(5)	b(4)	d (4)
		a(5)		c (5)



- The shortest path between a and d calculated by Dijkstra's algorithm is 4 while there is a shorter one via "c" which is 3 .

Bellman-Ford algorithm (1958-1962)

- The presence of edge weights with different signs allows the modeling of complex situations with variable costs and variable profits.
- Dijkstra's algorithm does not allow negative edges to be considered, because once a vertex is labeled, this label cannot be changed in subsequent iterations.
- Dijkstra's algorithm is therefore called label-fixing.
- The Bellman-Ford algorithm, on the other hand, allows a label that is not final until the algorithm completes (the label is changed iteratively).
- This type of algorithm is called label-correcting.

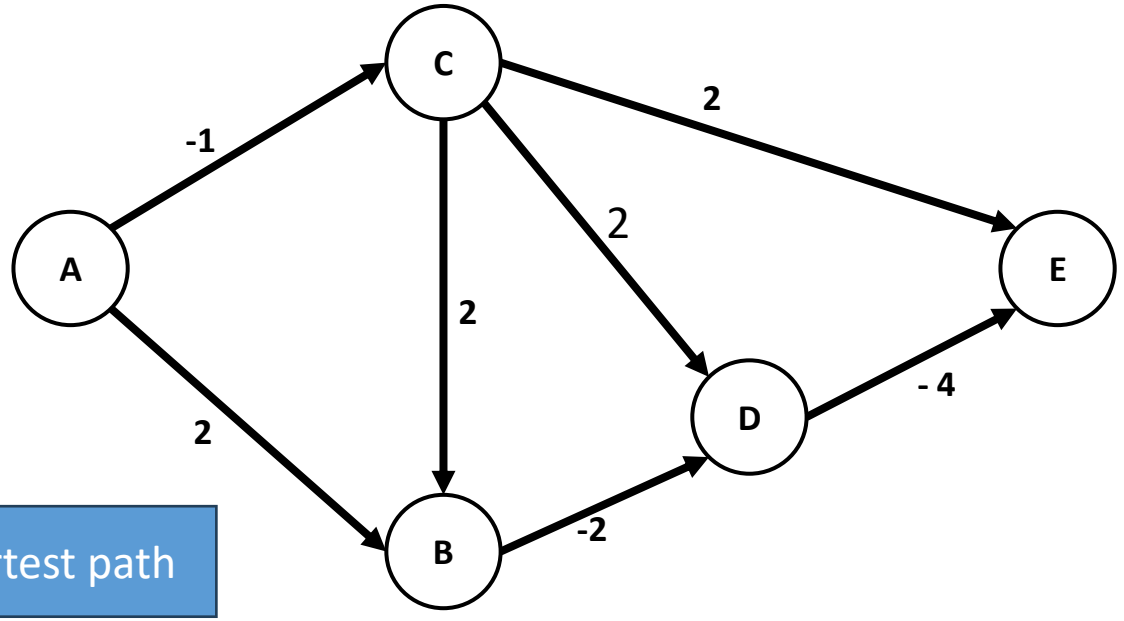
Bellman-Ford algorithm (1958-1962)

Bellman-Ford Procedure

Input: Graph **G = (V, E)** , Edge Lengths **L(u,v)** , and **S** a source vertex

Output: dist (distance) and **pred** (predecessor) two arrays containing the shortest path between the source **S** and each vertex

```
{
// Step 1: initialize graph
        for all u in V Do
        {
                dist (u) = ∞;
                prev (u) = nil ;
        }
        dist (S) = 0;
// Step 2: relax edges repeatedly
        for i = 1 to |V|-1 Do {
                for all Edge (u, v) in E Do
                        Dist (v) = min{ dist (v), dist (u) + L( u,v )
        }
// Step 3: check for negative-weight cycles
        for All edge (u, v) in E do
        if dist (u) + L( u,v ) < dist (v) then
                {
                        A negative cycle exists; find a vertex on the cycle;
                        Break;
                }
}
```



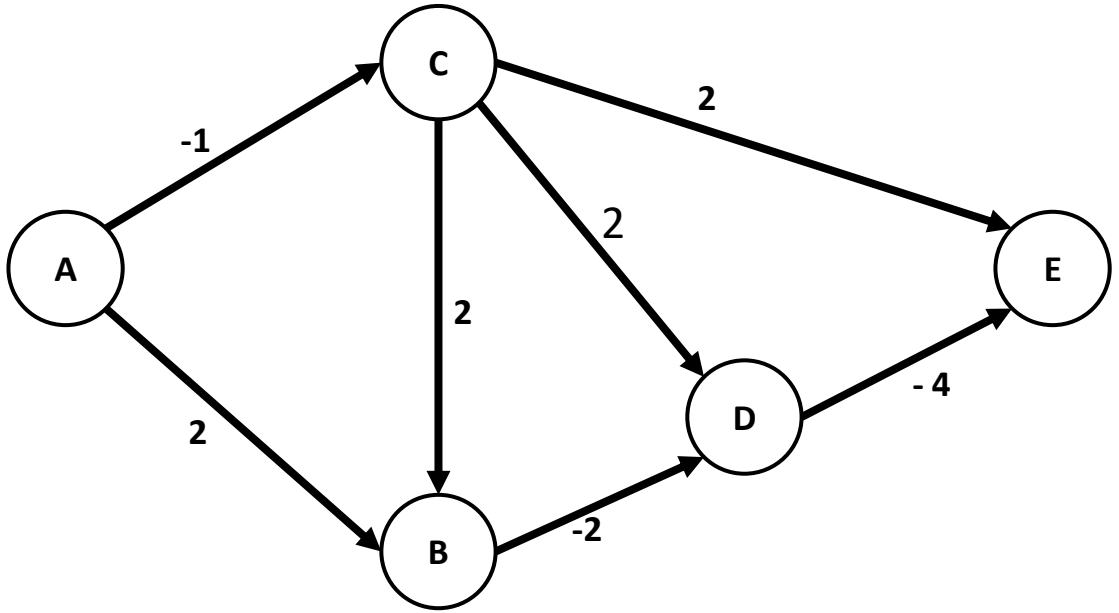
5 vertices → 4 iterations for calculating the shortest path

+

An iteration to check if there is an **improving circuit**

Initialization

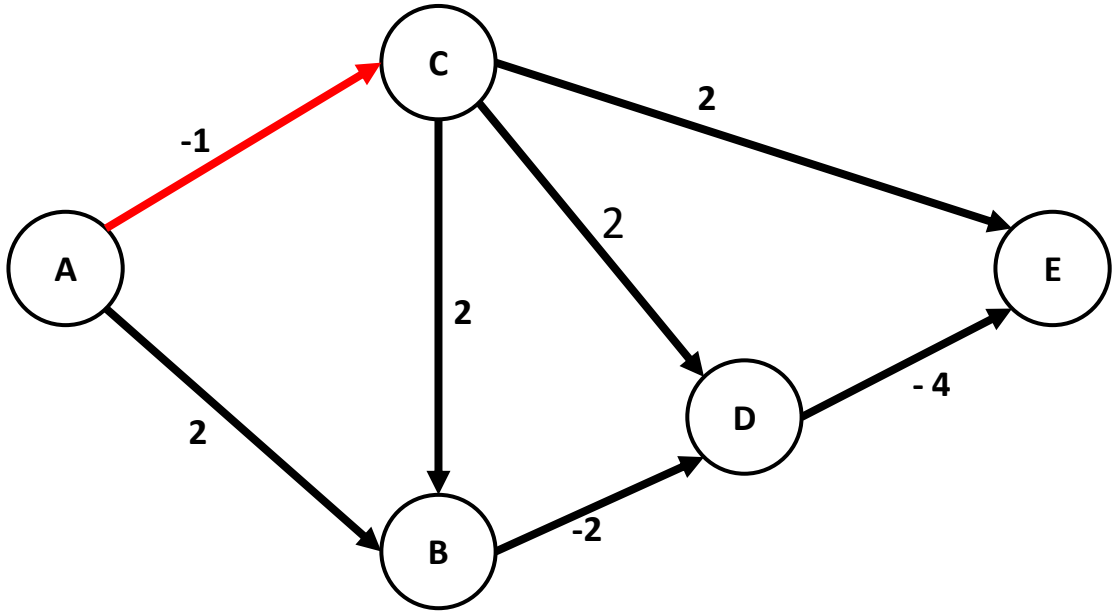
	Dist	Prev
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-



iteration 1 of 4

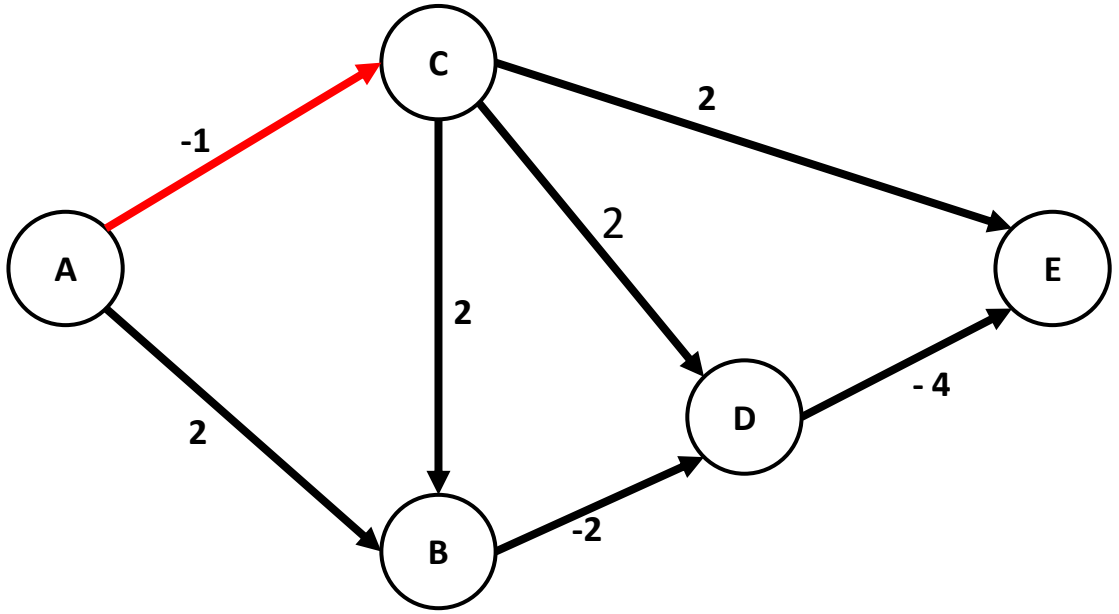
Arc(A, C)

	Dist	Prev
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-



iteration 1 of 4

Arc(A, C)



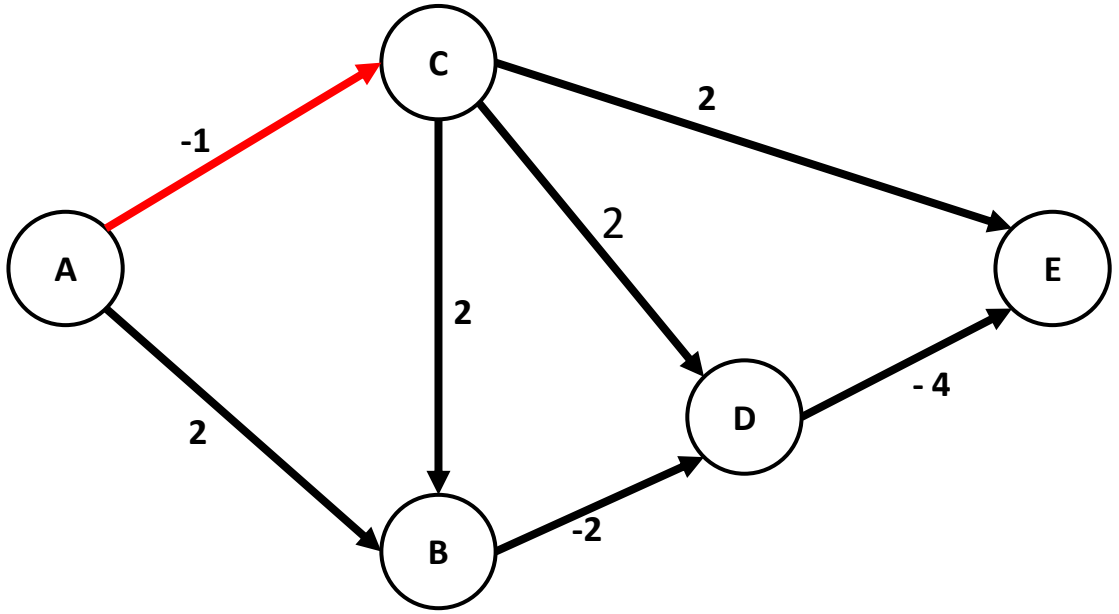
	Dist	Prev
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-

Dist (C) = min(Dist (C), Dist (A) + L(A, C))

Dist (C) = min(∞, 0+ (-1))

iteration 1 of 4

Arc(A, C)



	Dist	Prev
A	0	-
B	∞	-
C	-1	A
D	∞	-
E	∞	-

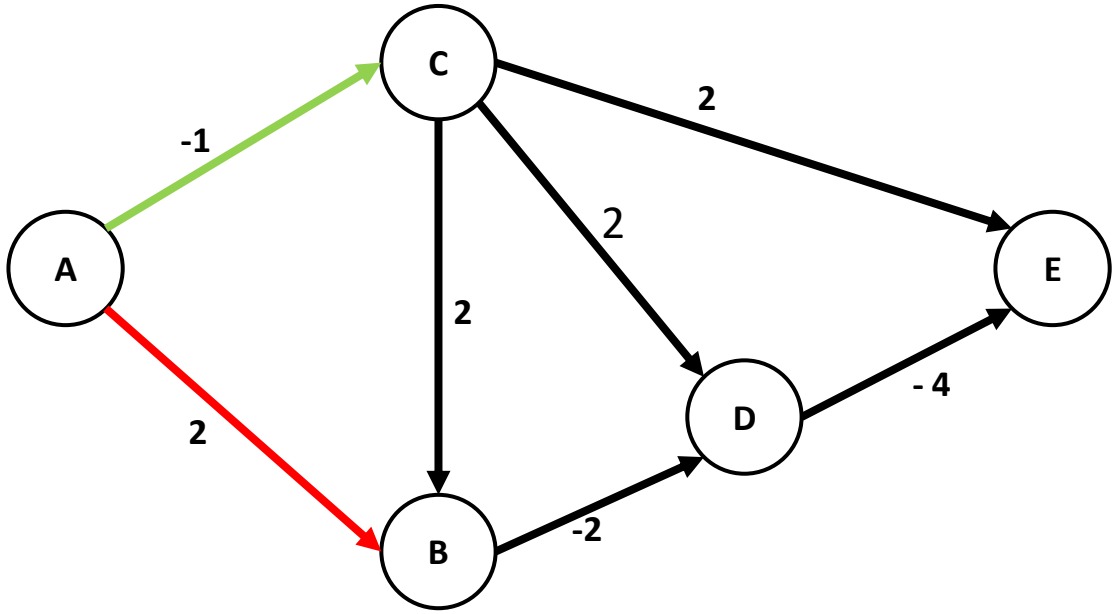
Dist (C) = min(Dist (C), Dist (A) + L(A, C))

Dist (C) = -1

iteration 1 of 4

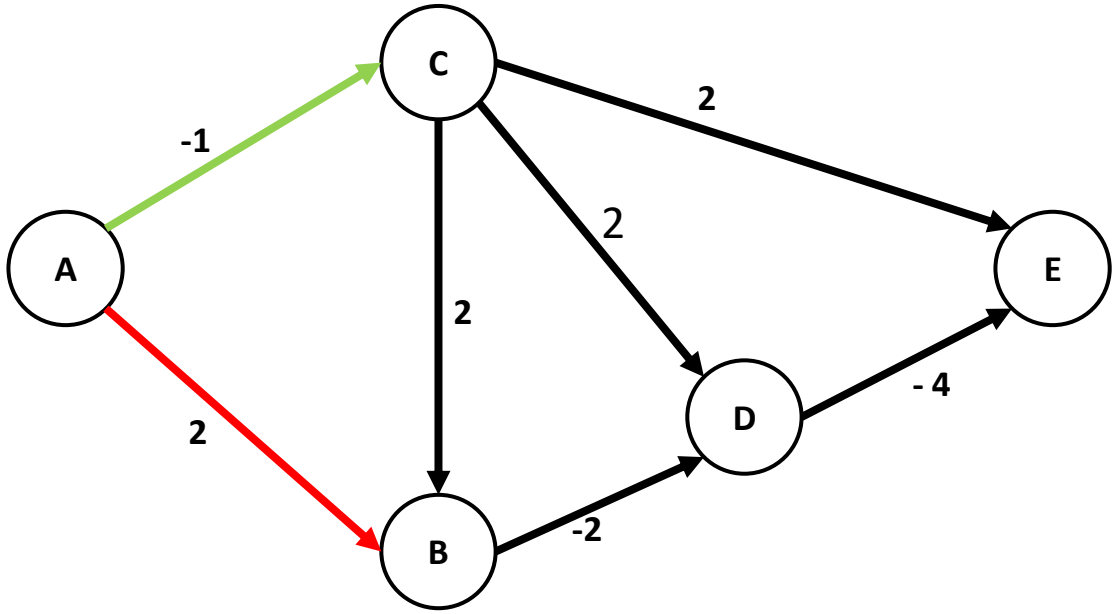
Arc(A, B)

	Dist	Prev
A	0	-
B	∞	-
C	-1	A
D	∞	-
E	∞	-



iteration 1 of 4

Arc(A, B)



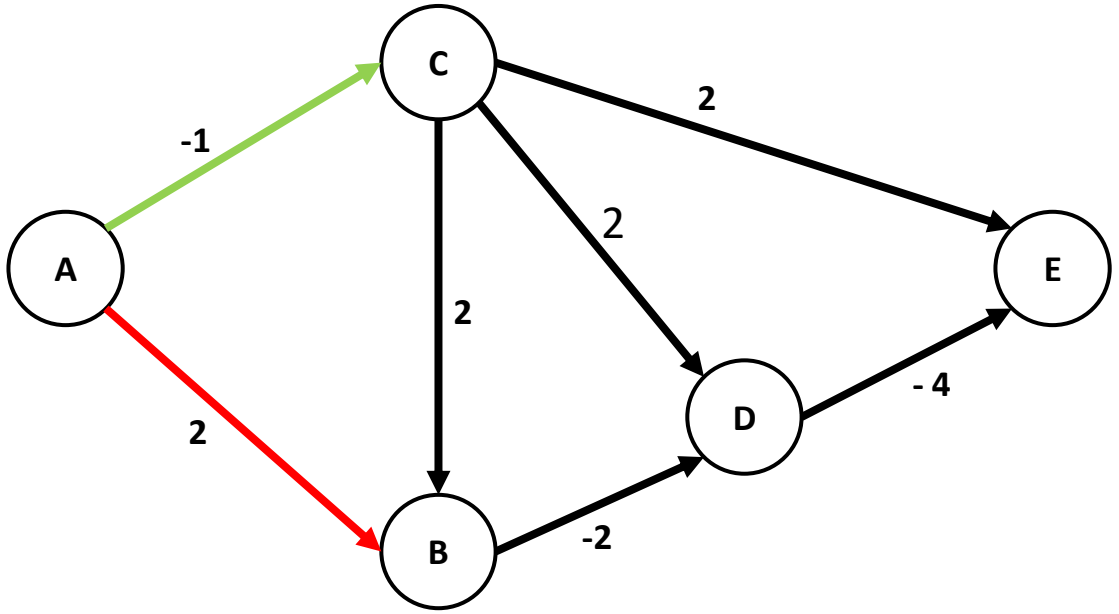
	Dist	Prev
A	0	-
B	∞	-
C	-1	A
D	∞	-
E	∞	-

Dist (B) = min(Dist (B), Dist (A) + L(A, B))

Dist (B) = min(∞, 0+ 2)

iteration 1 of 4

Arc(A, B)



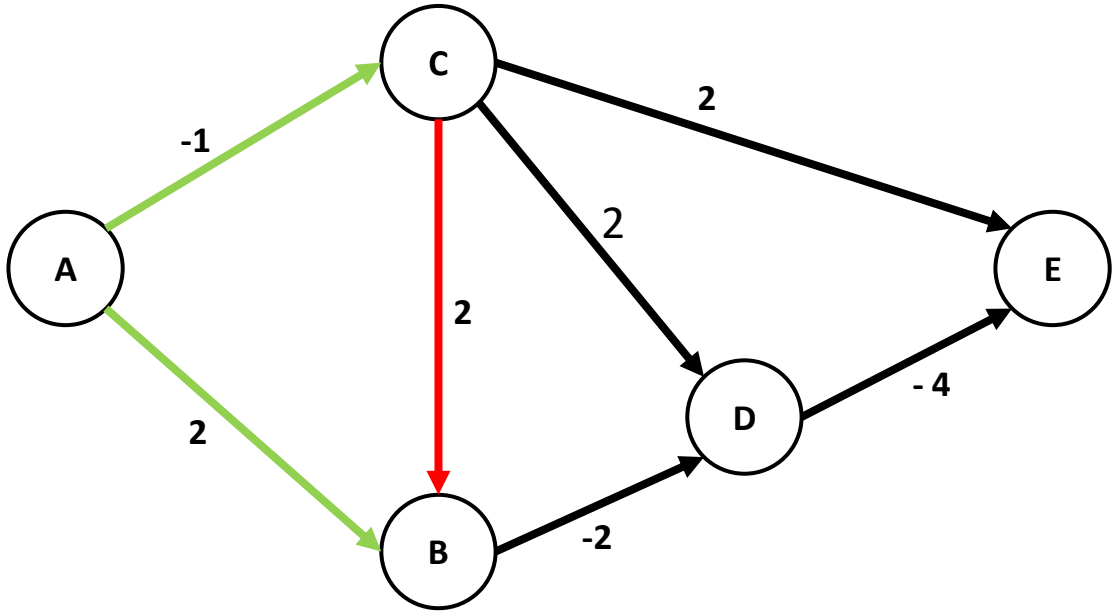
	Dist	Prev
A	0	-
B	2	A
C	-1	A
D	∞	-
E	∞	-

Dist (B) = min(Dist (B), Dist (A) + L(A, B))

Dist (B) = 2

iteration 1 of 4

Arc(C, B)



	Dist	Prev
A	0	-
B	2	A
C	-1	A
D	∞	-
E	∞	-

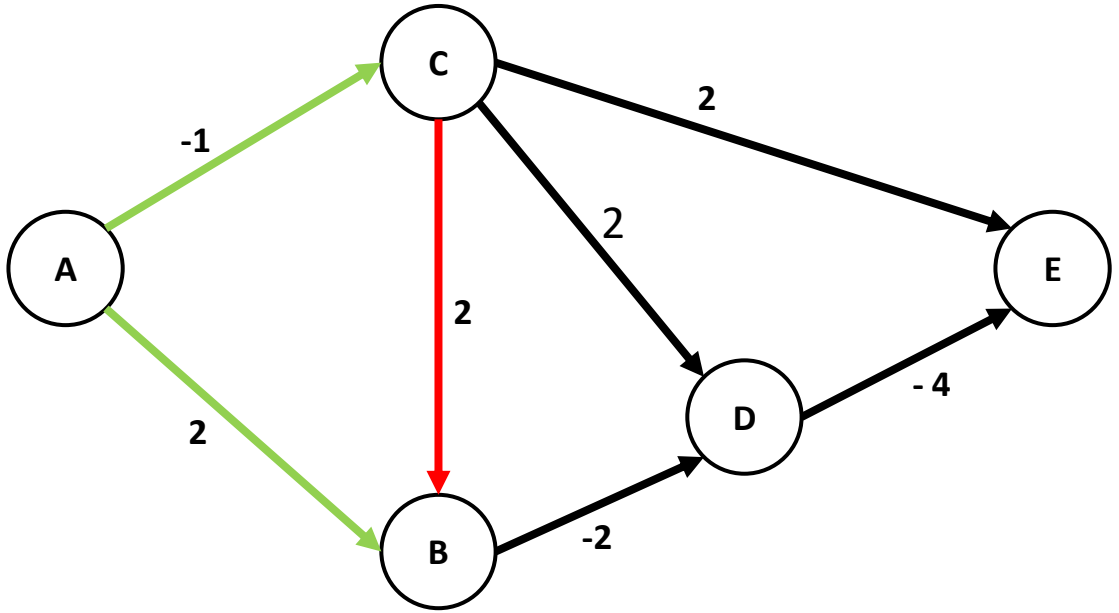
Dist (B) = min(Dist (B), Dist (C) + L(C, B))

Dist (B) = min(2, 2 + (-1)) = 1

iteration 1 of 4

Arc(C, B)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	∞	-
E	∞	-



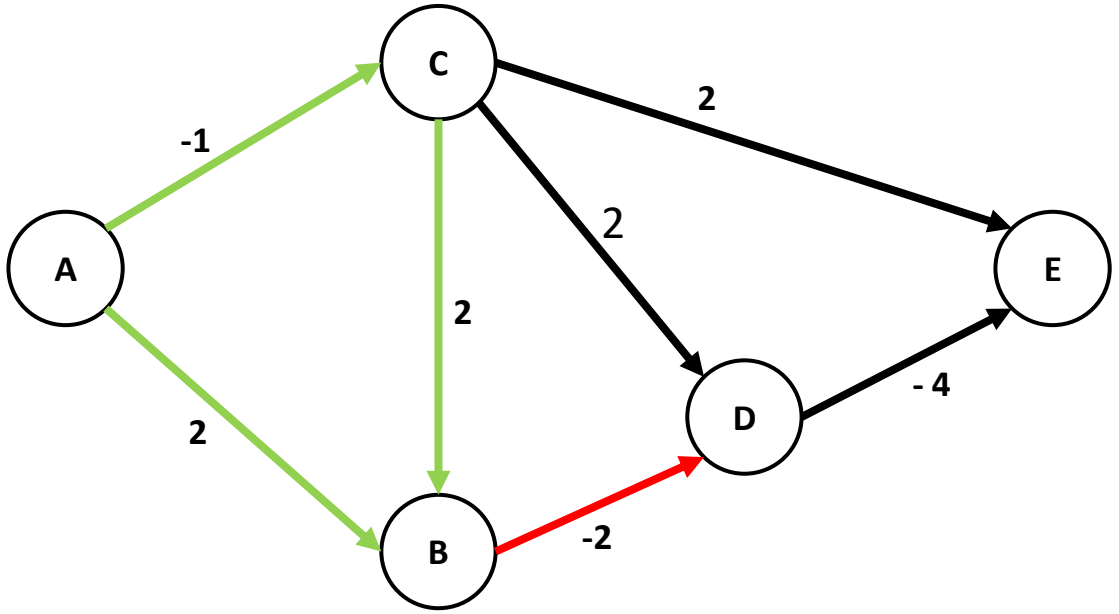
Dist (B) = min(Dist (B), Dist (C) + L(C, B))

Dist (B) = 1

iteration 1 of 4

Arc(B, D)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	∞	-
E	∞	-

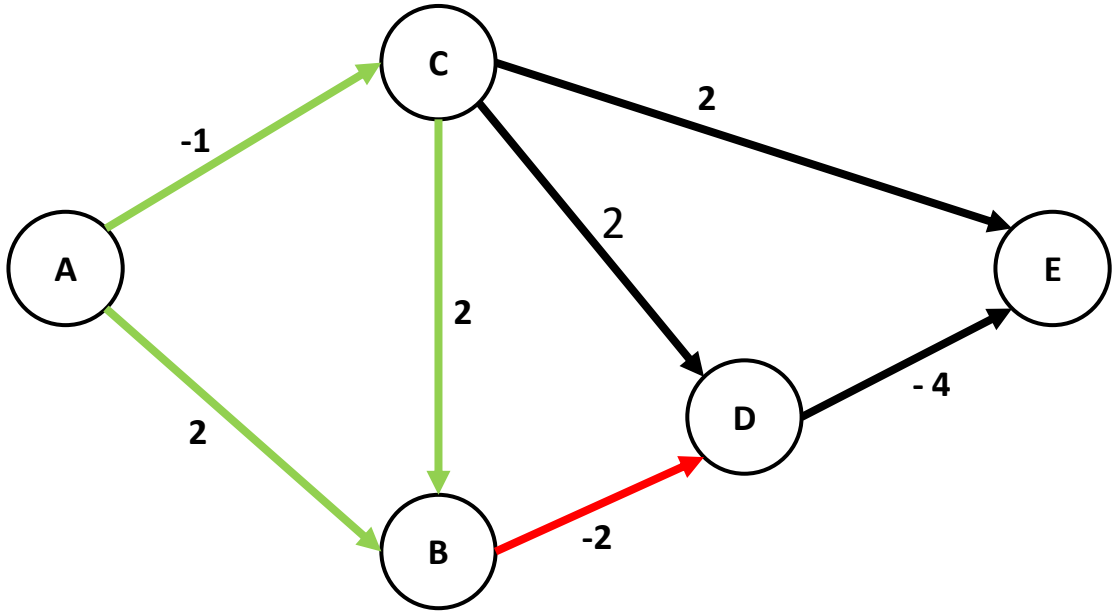


Dist (D) = min(Dist (D), Dist (B) + L(B, D))

Dist (D) = min(∞, 1+ (-2))

iteration 1 of 4

Arc(B, D)



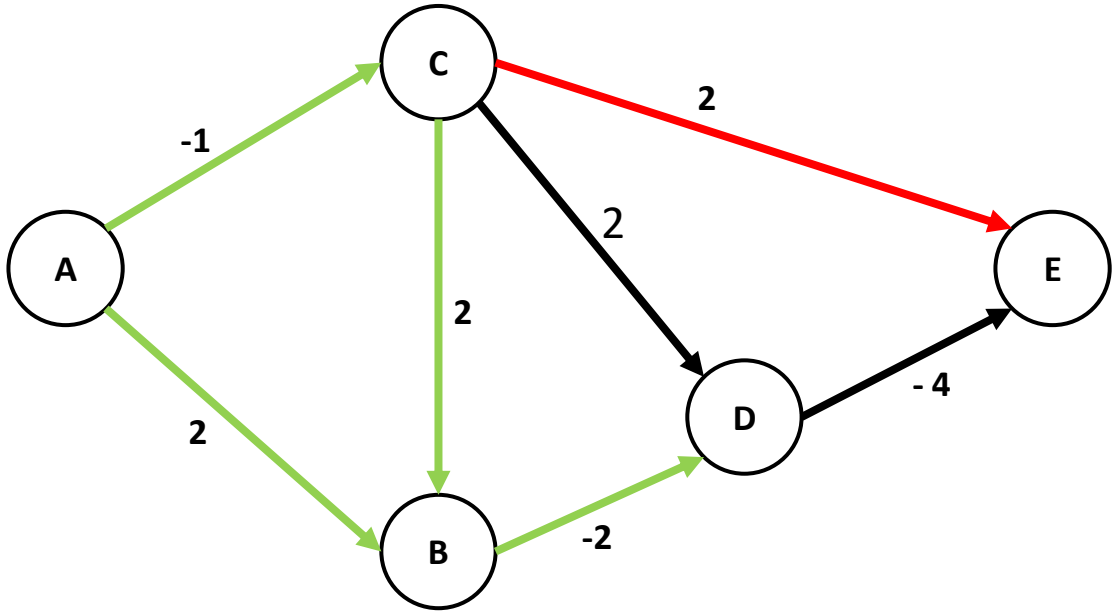
	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	∞	-

Dist (D) = min(Dist (D), Dist (B) + L(B, D))

Dist (D) = -1

iteration 1 of 4

Arc(C, E)



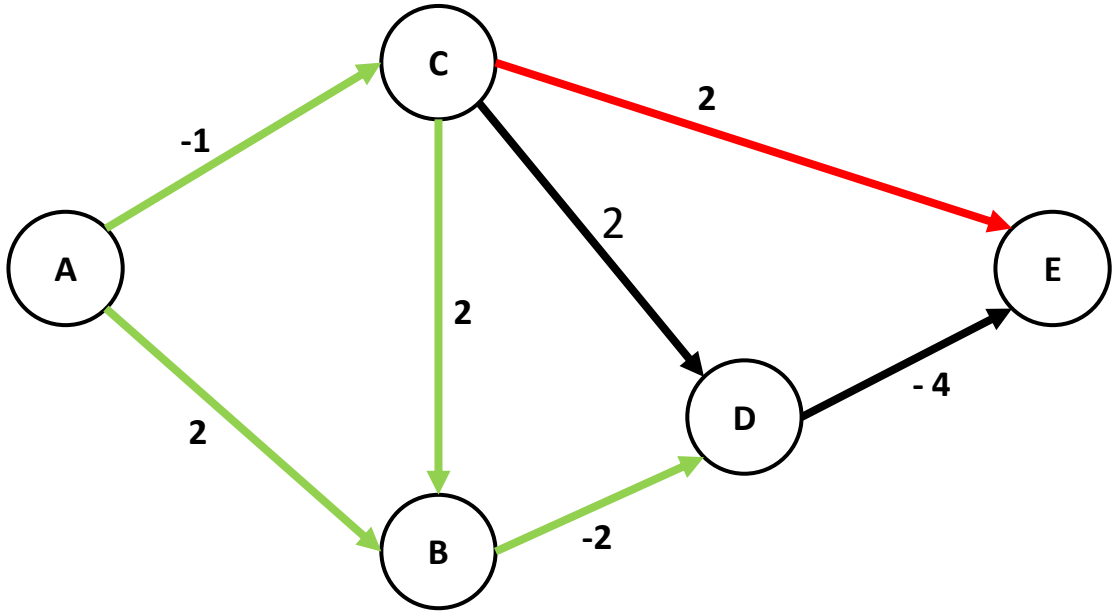
	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	∞	-

Dist (E) = min(Dist (E), Dist (C) + L(C, E))

Dist (E) = min(∞, -1+ (2)) = 1

iteration 1 of 4

Arc(C, E)



	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	∞	-

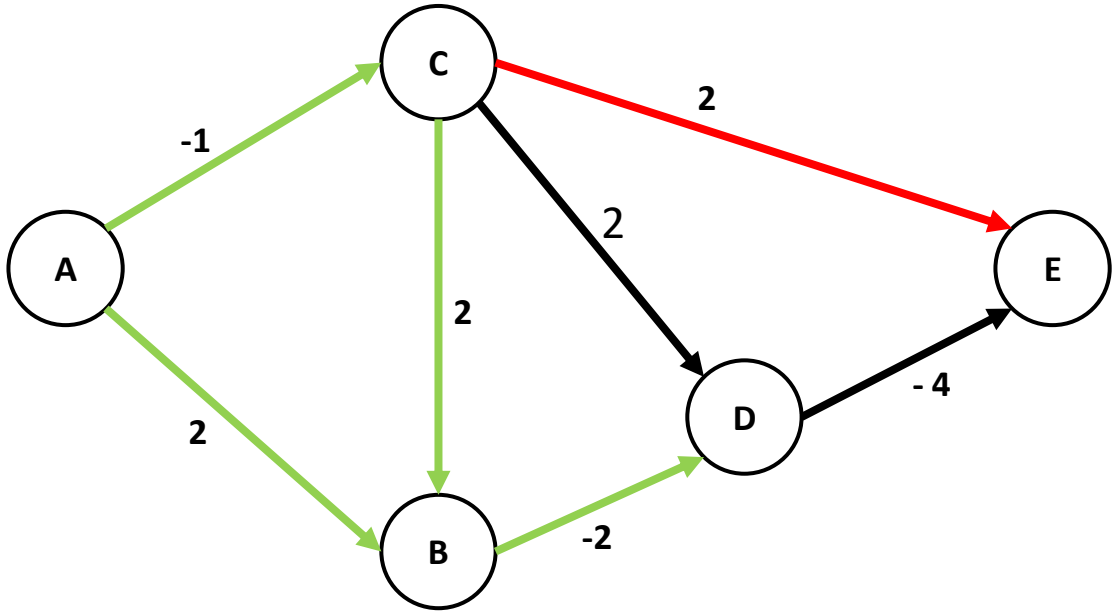
Dist (E) = min(Dist (E), Dist (C) + L(C, E))

Dist (E) = min(∞, -1+ (2)) = 1

iteration 1 of 4

Arc(C, E)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	1	C



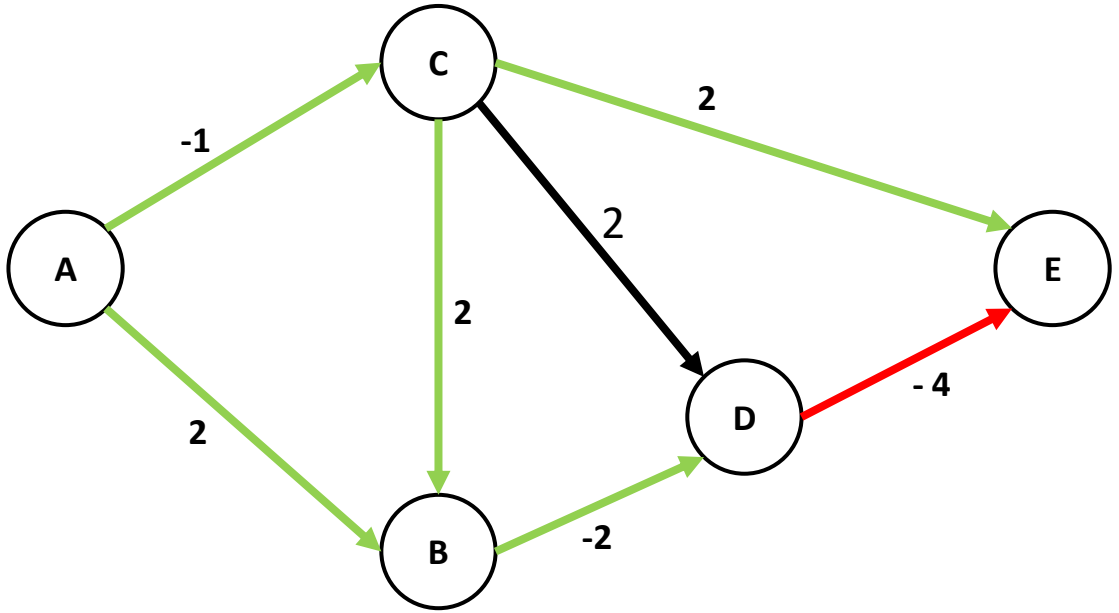
Dist (E) = min(Dist (E), Dist (C) + L(C, E))

Dist (E) = min(∞, -1+ (2)) = 1

iteration 1 of 4

Arc(D, E)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	1	C



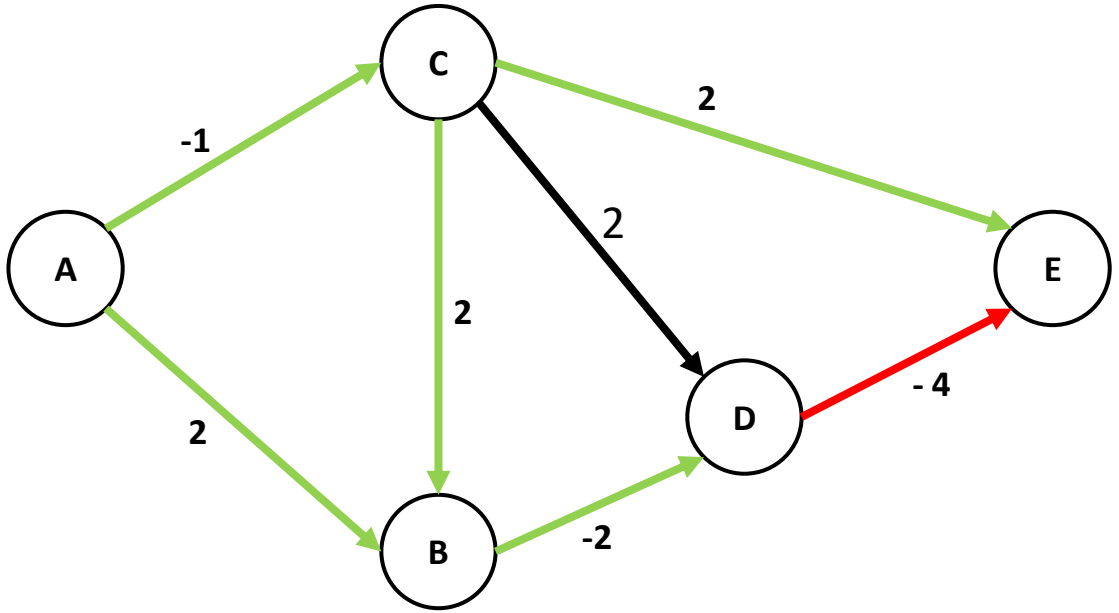
Dist (E) = min(Dist (E), Dist (D) + L(D, E))

Dist (E) = min(-1, -1+ (-4)) = -5

iteration 1 of 4

Arc(D, E)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D



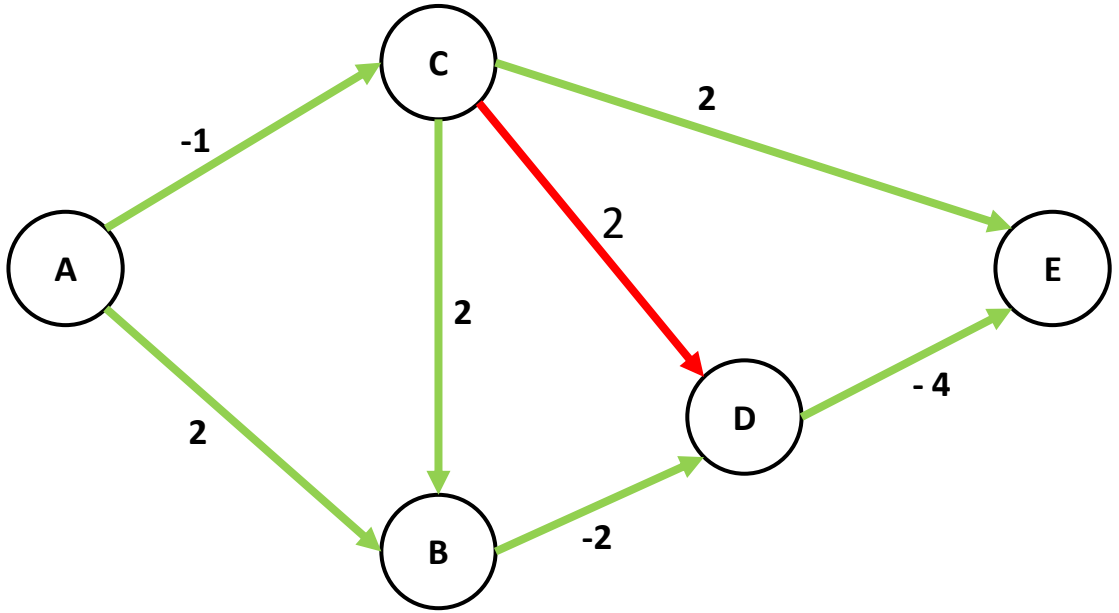
Dist (E) = min(Dist (E), Dist (D) + L(D, E))

Dist (E) = -5

iteration 1 of 4

Arc(C, D)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D

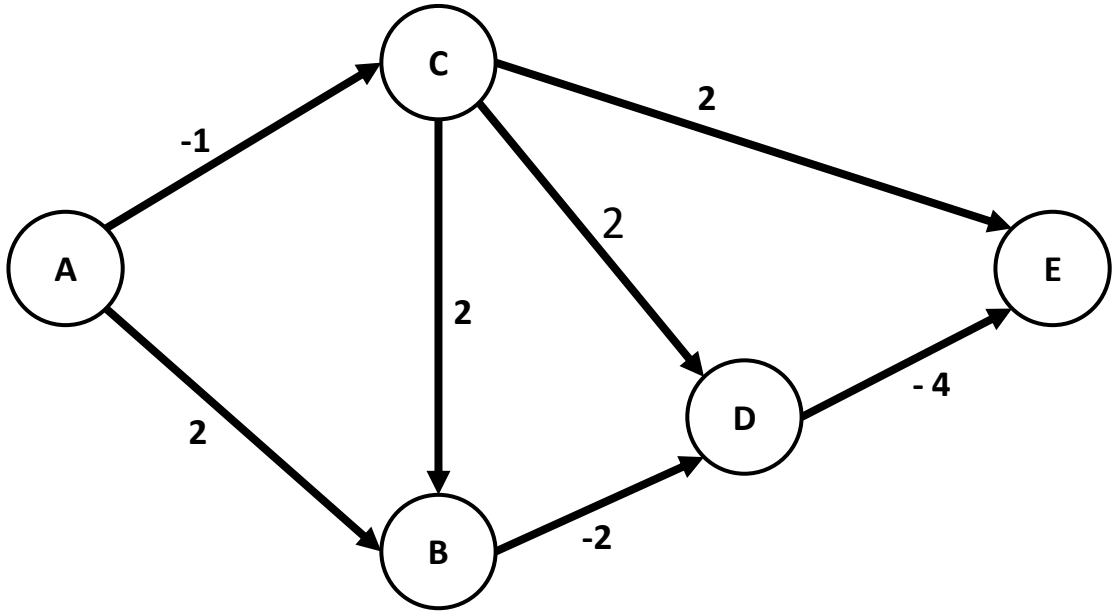


Dist (D) = min(Dist (D), Dist (C) + L(C, D))

Dist (D) = min(-1, 1 + 2) = -1

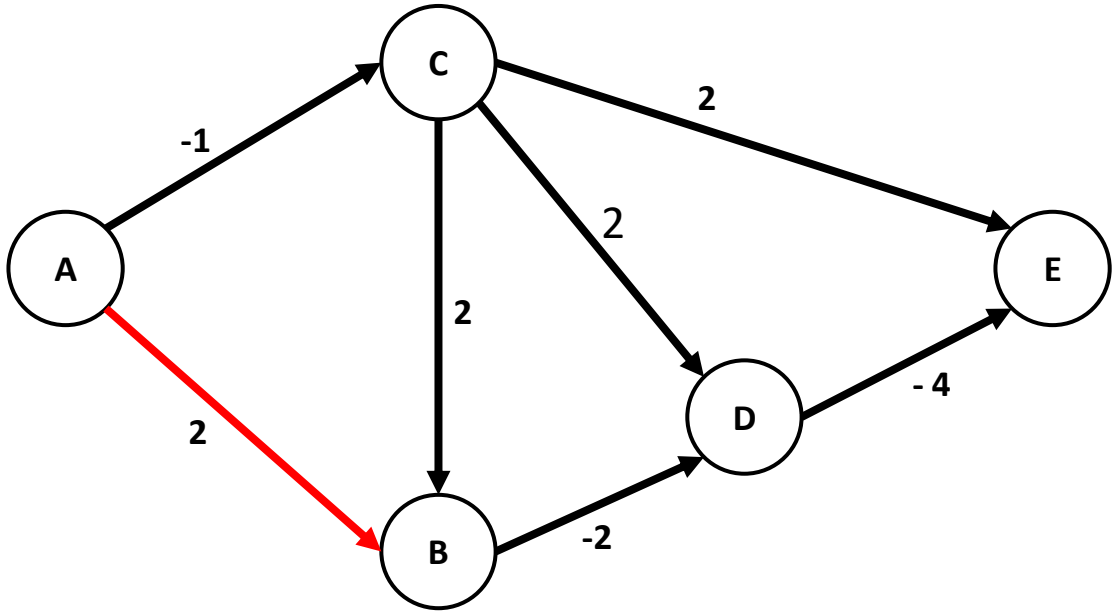
iteration 2 of 4

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D



iteration 2 of 4

Arc(A, B)



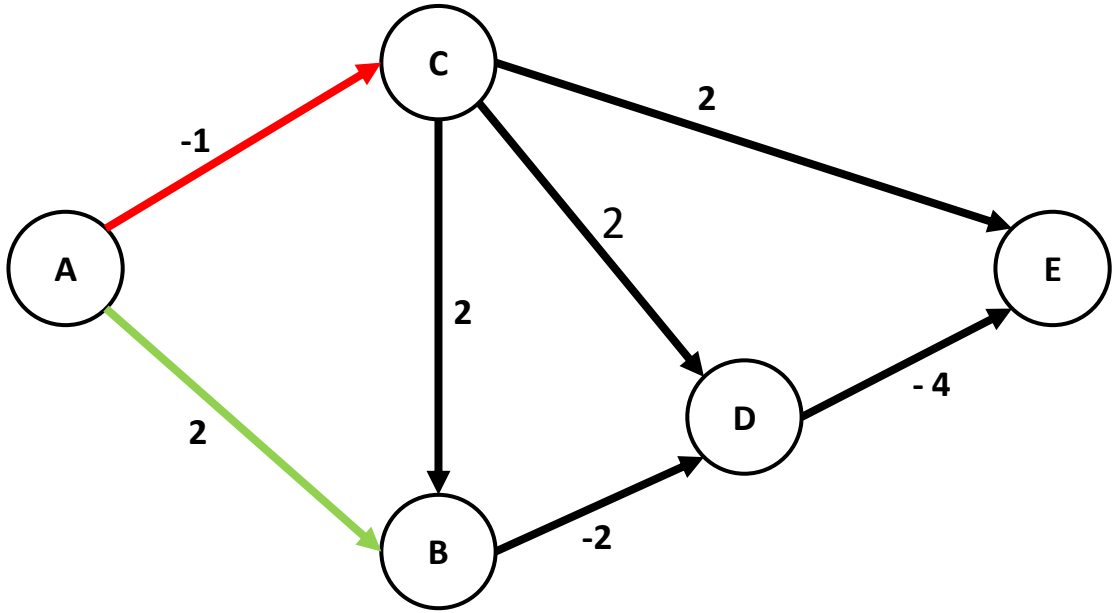
	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D

Dist (B) = min(1, 0 + 2) = 2

iteration 2 of 4

Arc(A, C)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D

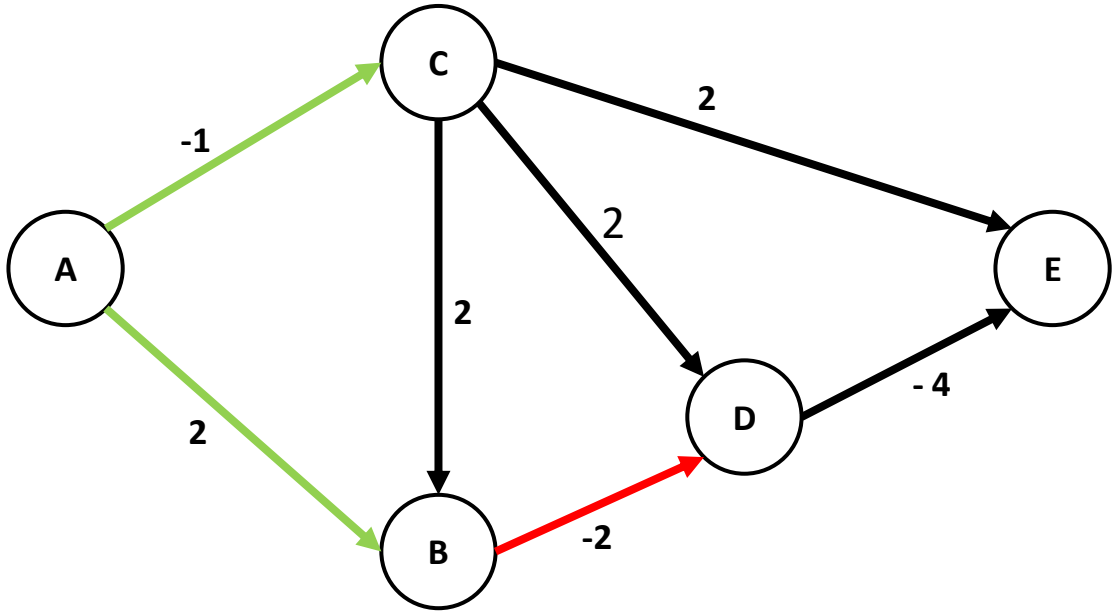


Dist (C) = min(-1, 0 + (-1)) = -1

iteration 2 of 4

Arc(B, D)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D

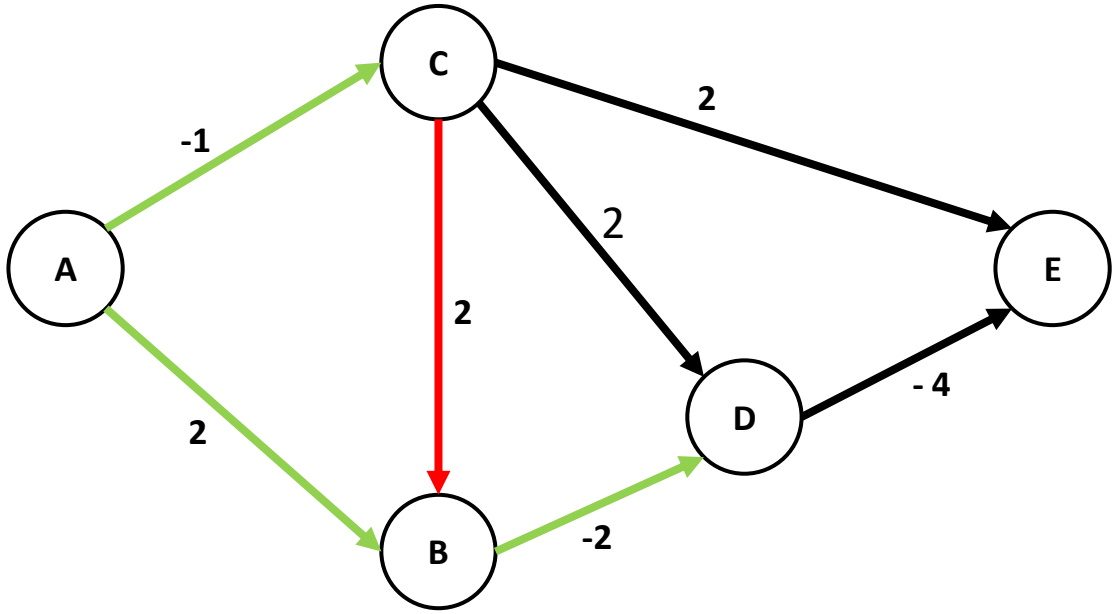


Dist (D) = min(-1, 1 + (-2)) = -1

iteration 2 of 4

Arc(C, B)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D

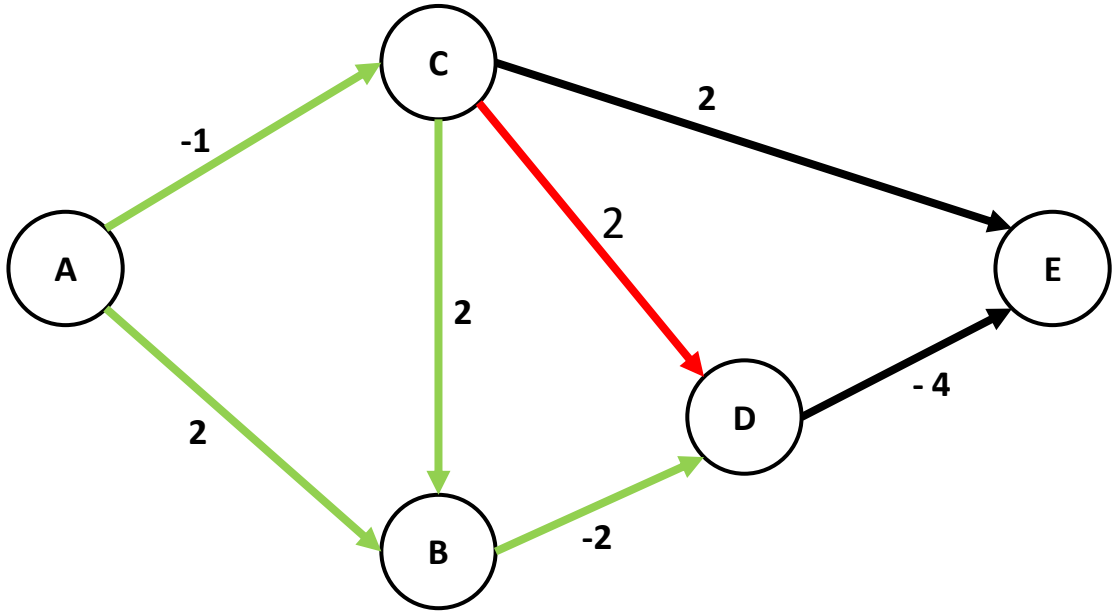


Dist (B) = min(1, -1 + (2)) = 1

iteration 2 of 4

Arc(C, D)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D

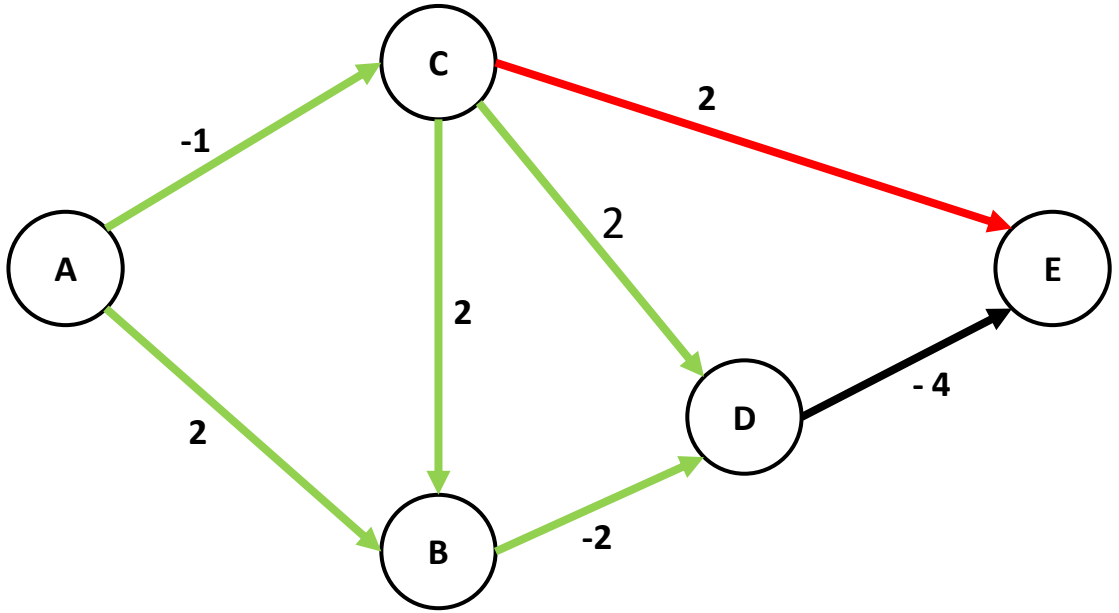


Dist (D) = min(-1, 1 + (2)) = -1

iteration 2 of 4

Arc(C, E)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D

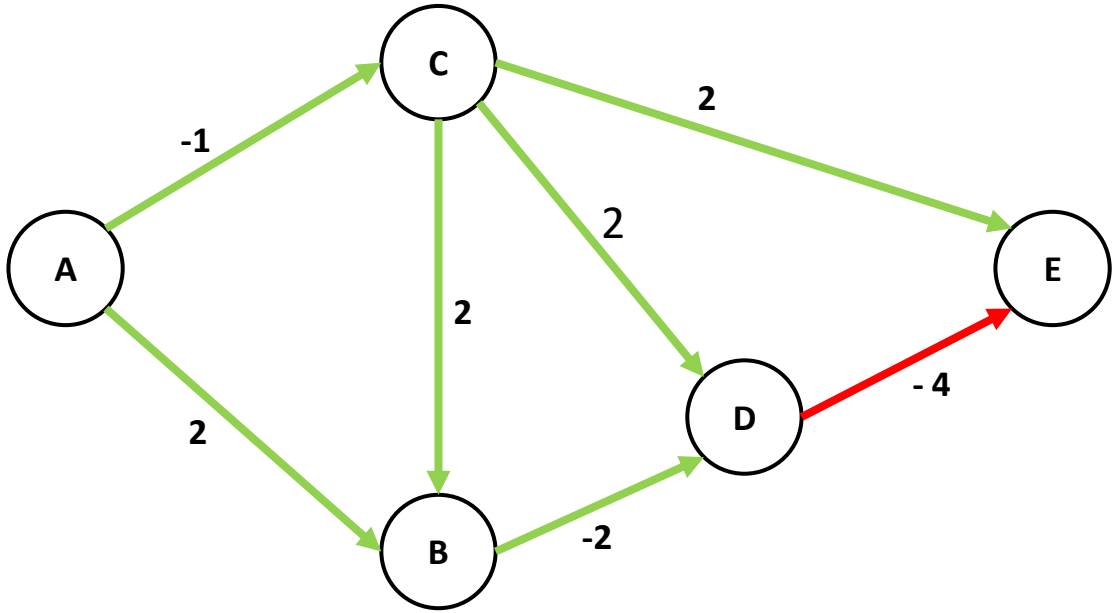


Dist (E) = min(-1, 1 + (2)) = -1

iteration 2 of 4

Arc(D, E)

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D

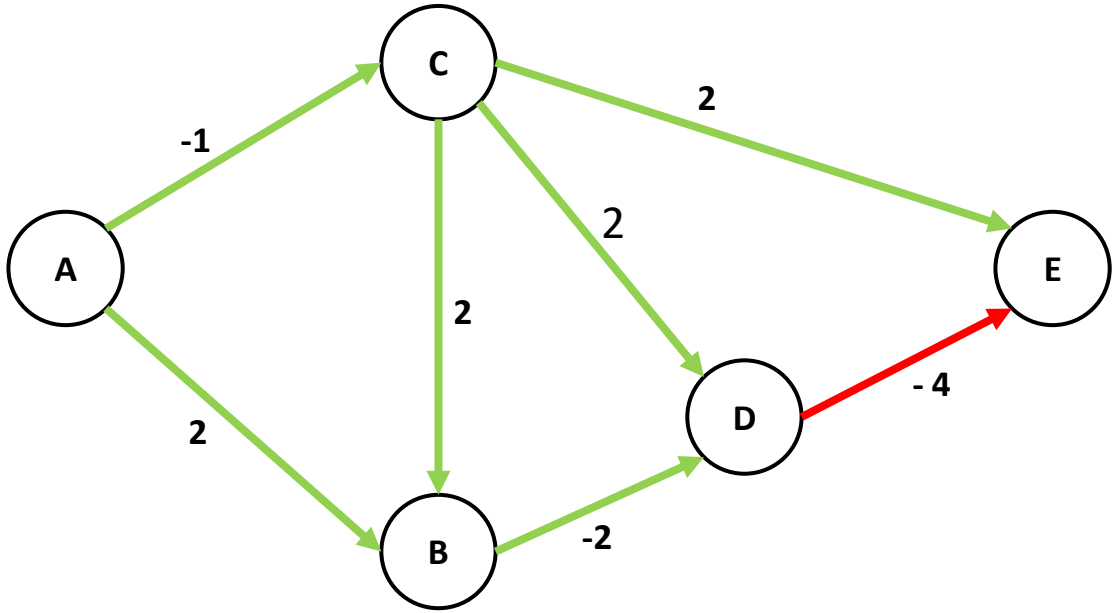


Dist (D) = min(-1, -1 + (-4)) = -5

iteration 2 of 4

Arc(D, E)

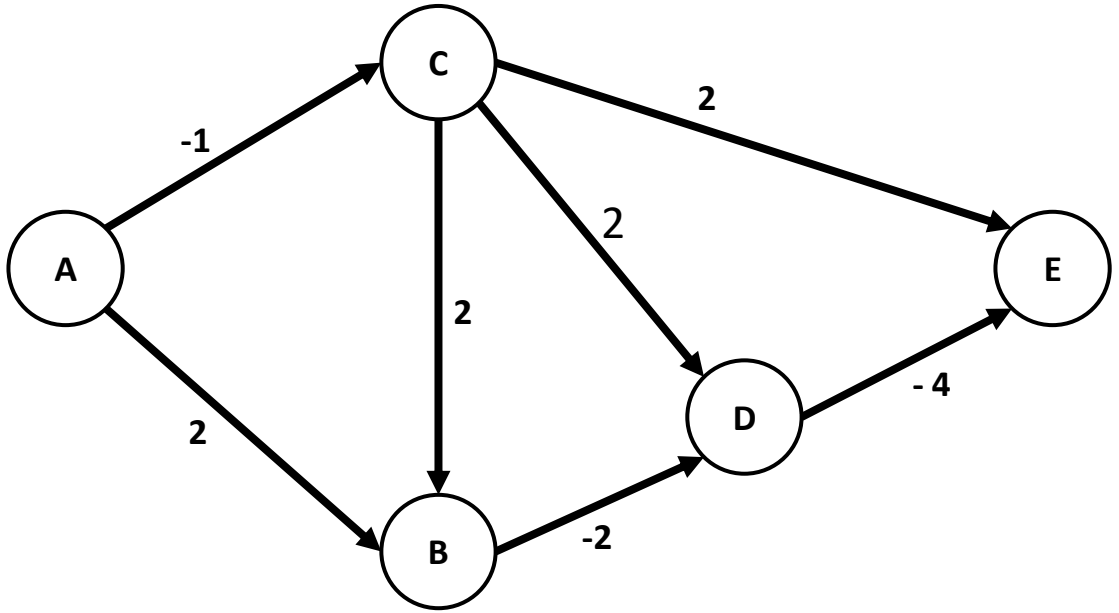
	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D



Dist (D) = min(-1, -1 + (-4)) = -5

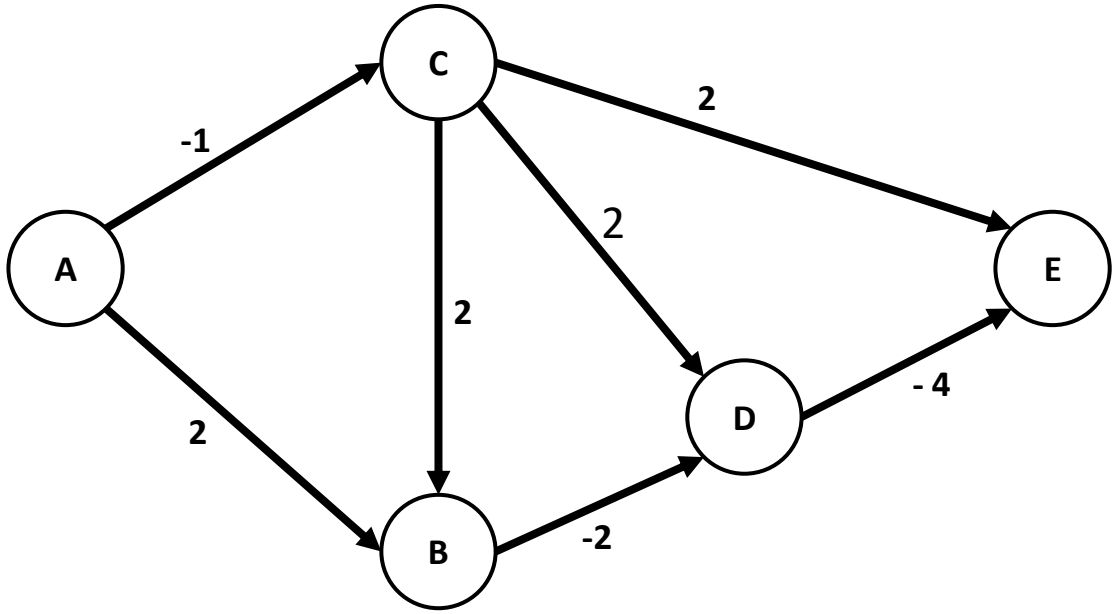
iteration 2 of 4

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D



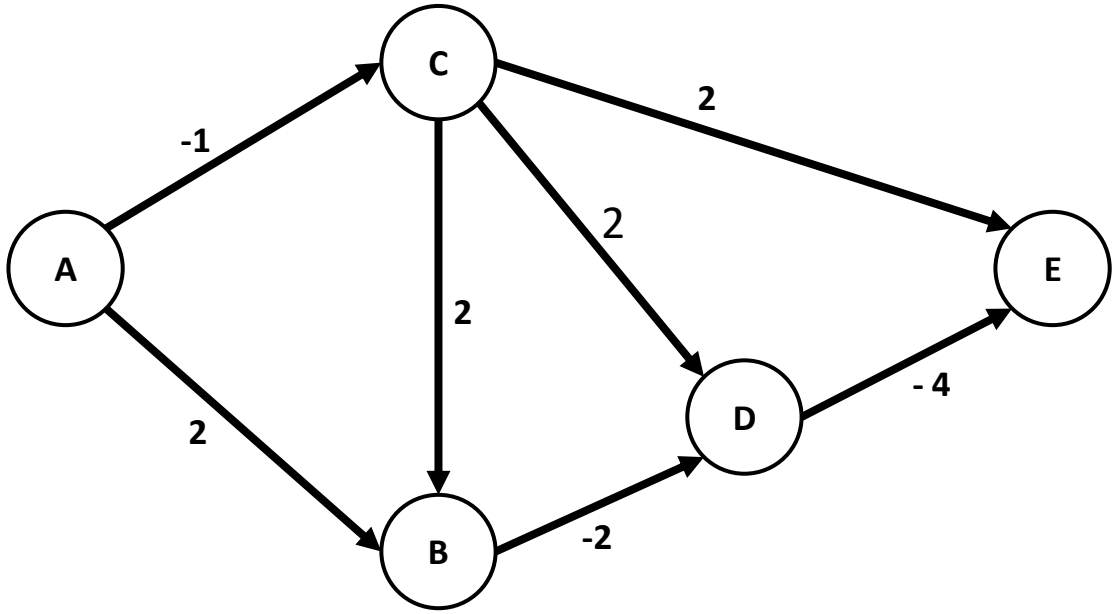
iteration 3 of 4

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D



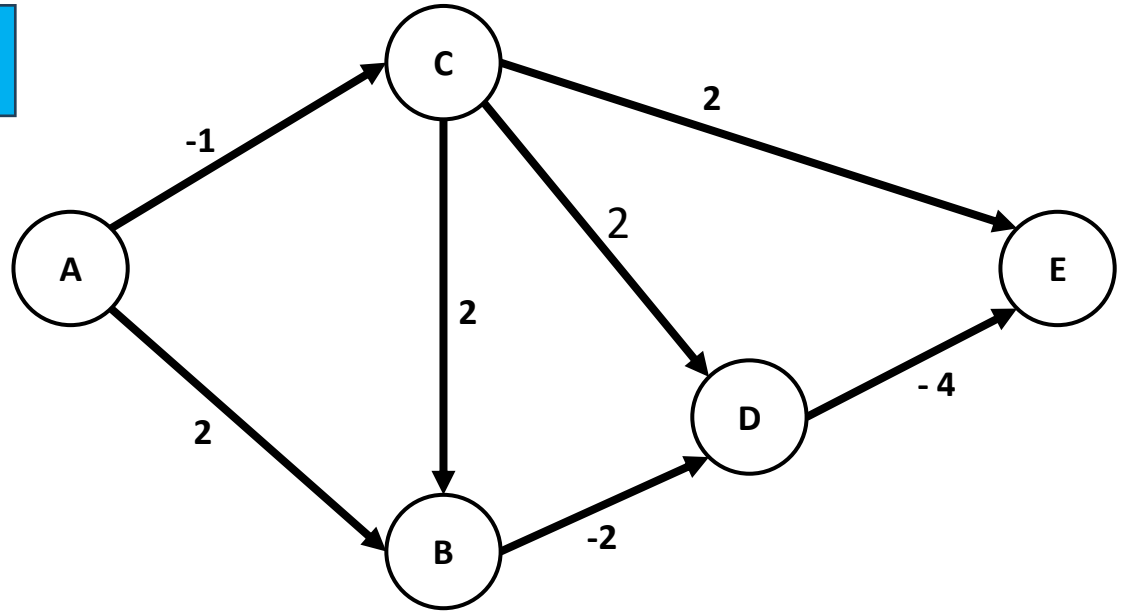
iteration 4 of 4

	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D



Detection of negative circuits

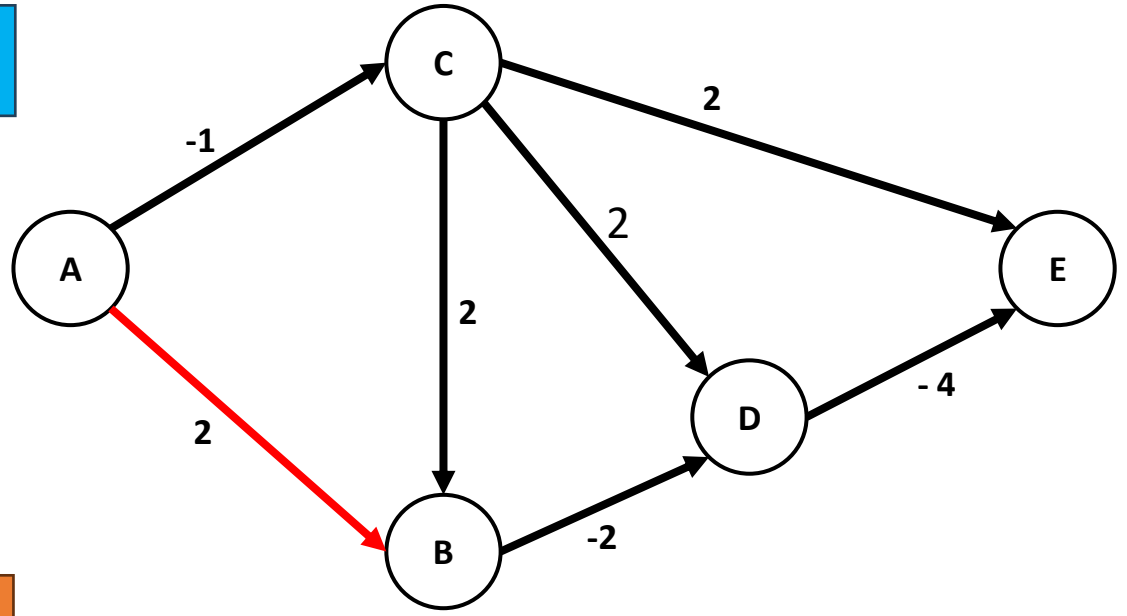
	Dist
A	0
B	1
C	-1
D	-1
E	-5



If Dist (U) > min(Dist (U), Dist (U) + L(U, V)) Then the graph contains a negative circuit

Detection of negative circuits

	Dist
A	0
B	1
C	-1
D	-1
E	-5



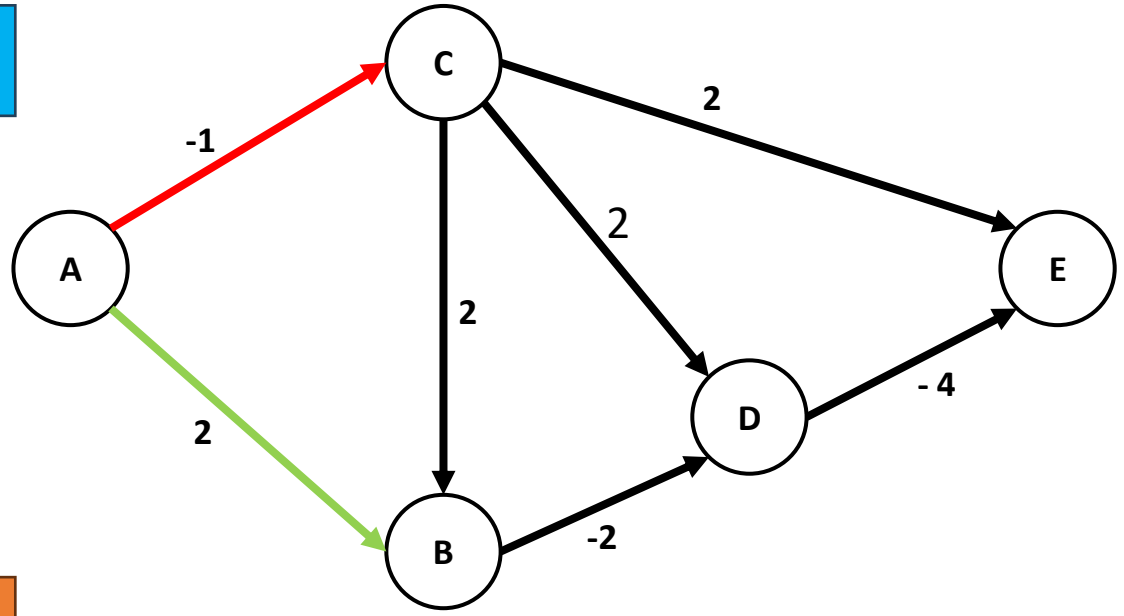
Arc(A, B)

Dist (U) > min(Dist (U), Dist (U) + L(U, V))

Dist (B) = 1 < min(1, 0 + 2) = 2

Detection of negative circuits

	Dist
A	0
B	1
C	-1
D	-1
E	-5

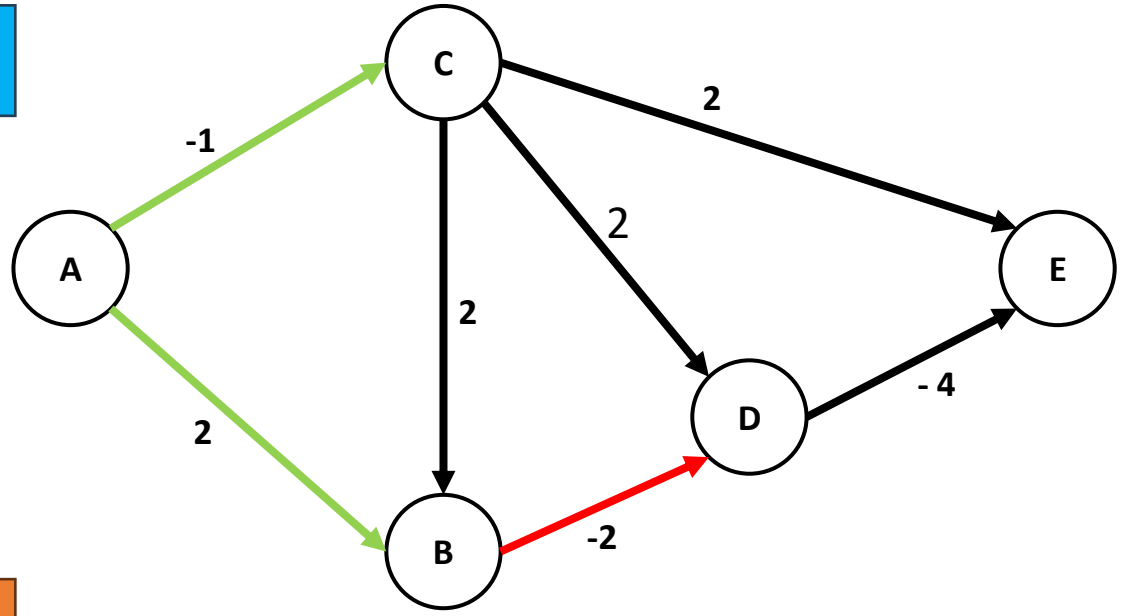


Arc(A, C)

Dist (C) = -1 <= min(-1, 0 + (-1)) = -1

Detection of negative circuits

	Dist
A	0
B	1
C	-1
D	-1
E	-5

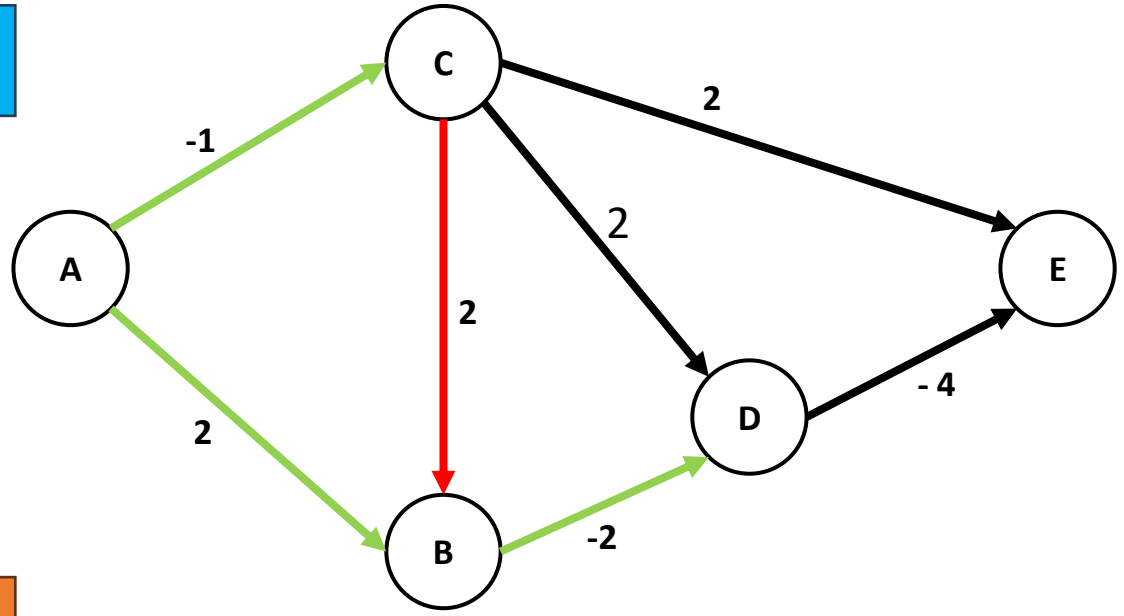


Arc(B, D)

Dist (D) =-1 <= min(-1, 1 + (-2)) = -1

Detection of negative circuits

	Dist
A	0
B	1
C	-1
D	-1
E	-5

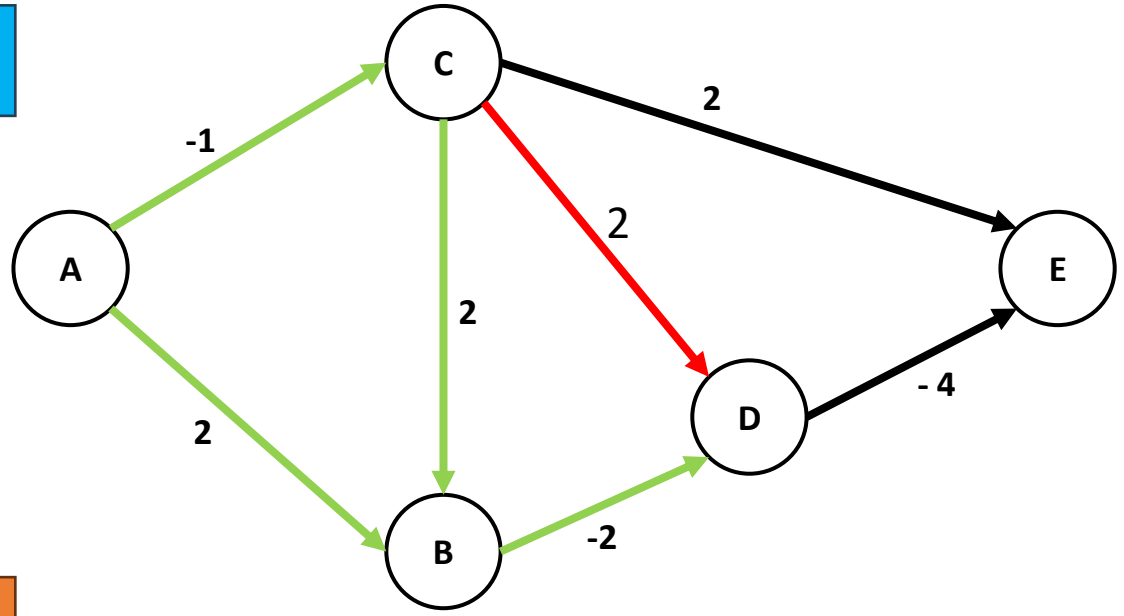


Arc(C, B)

Dist (B) = 1 <= min(1, -1 + (2)) = 1

Detection of negative circuits

	Dist
A	0
B	1
C	-1
D	-1
E	-5

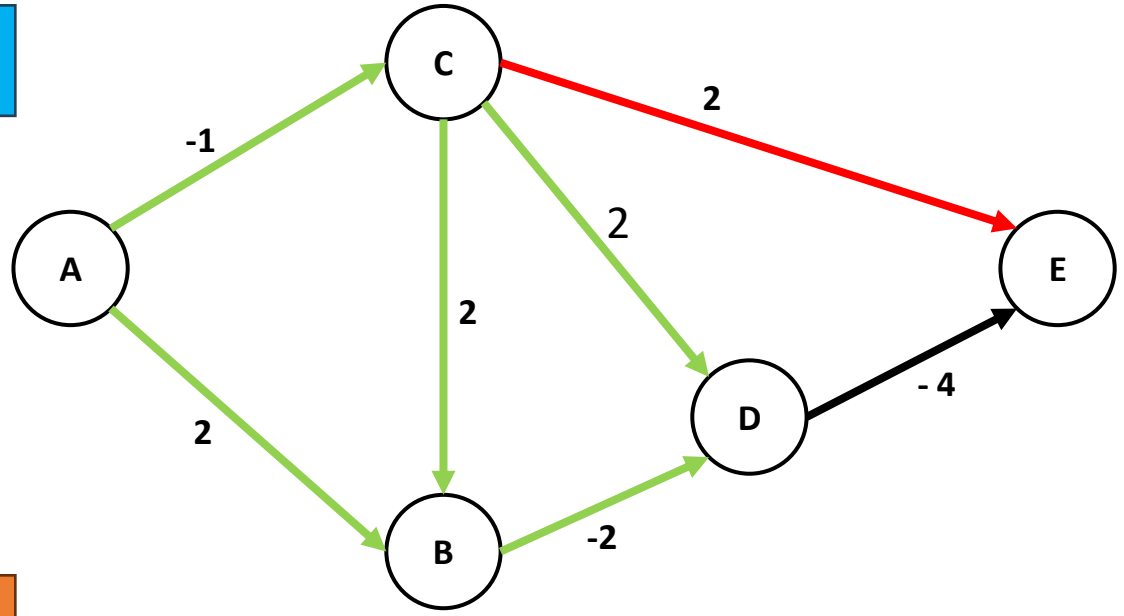


Arc(C, D)

Dist (D) = -1 <= min(-1, 1 + (2)) = -1

Detection of negative circuits

	Dist
A	0
B	1
C	-1
D	-1
E	-5

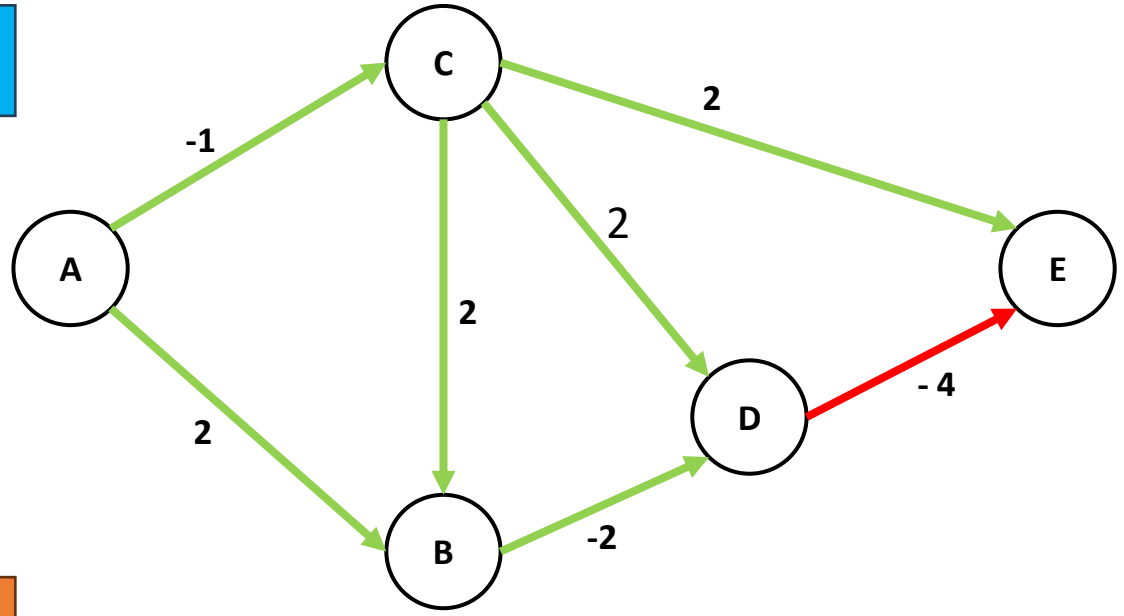


Arc(C, E)

Dist (E) = -5 <= min(-1, 1 + (2)) = -1

Detection of negative circuits

	Dist
A	0
B	1
C	-1
D	-1
E	-5

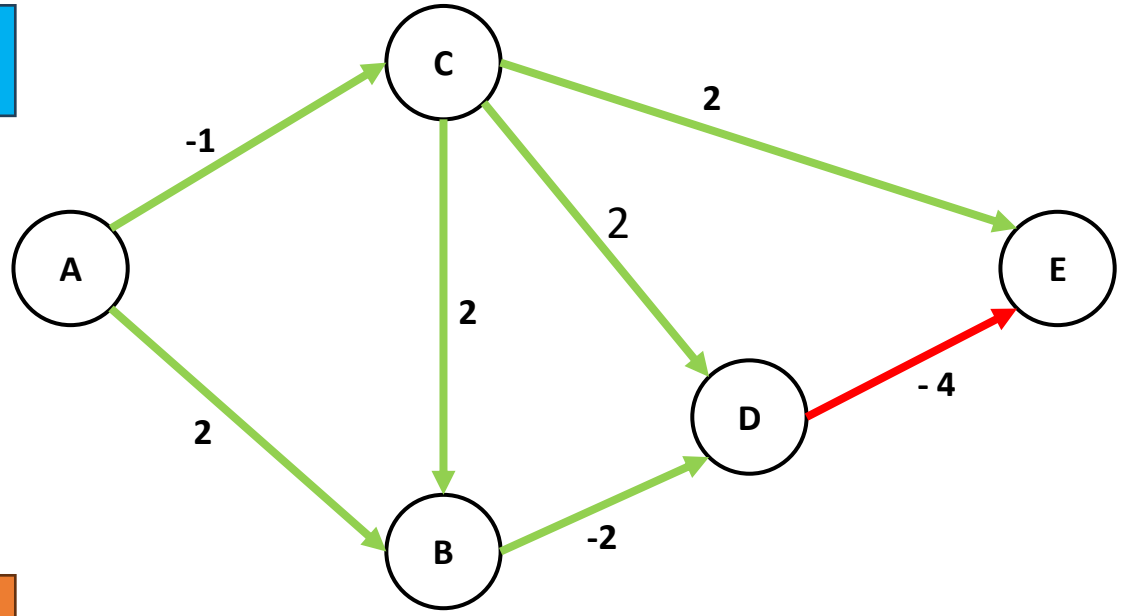


Arc(D, E)

Dist (D) = -1 <= min(-1, -1 + (-4)) = -5

Detection of negative circuits

	Dist
A	0
B	1
C	-1
D	-1
E	-5

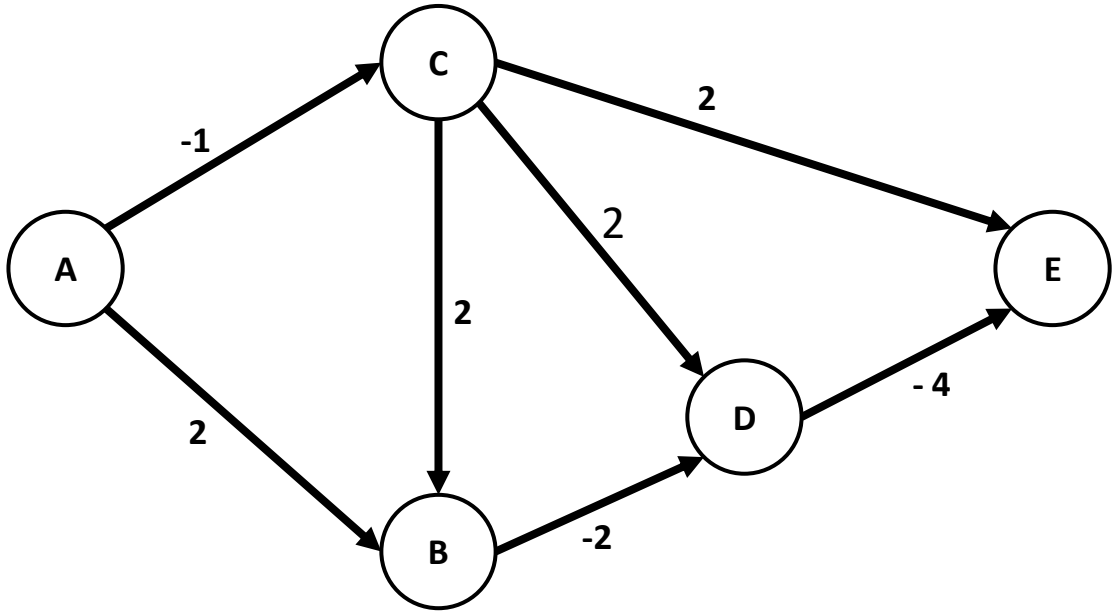


Arc(D, E)

Dist (D) = -1 <= min(-1, -1 + (-4)) = -5

END

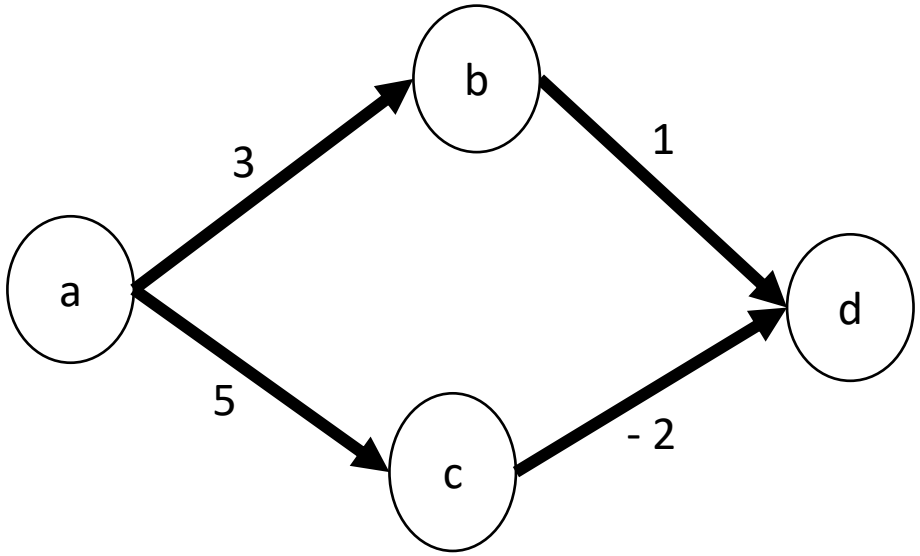
	Dist	Prev
A	0	-
B	1	C
C	-1	A
D	-1	B
E	-5	D



- Absence of negative weight circuit accessible from vertex “A” .
- The result represents the shortest paths from "A" to all other vertices of the graph as well as their weights.

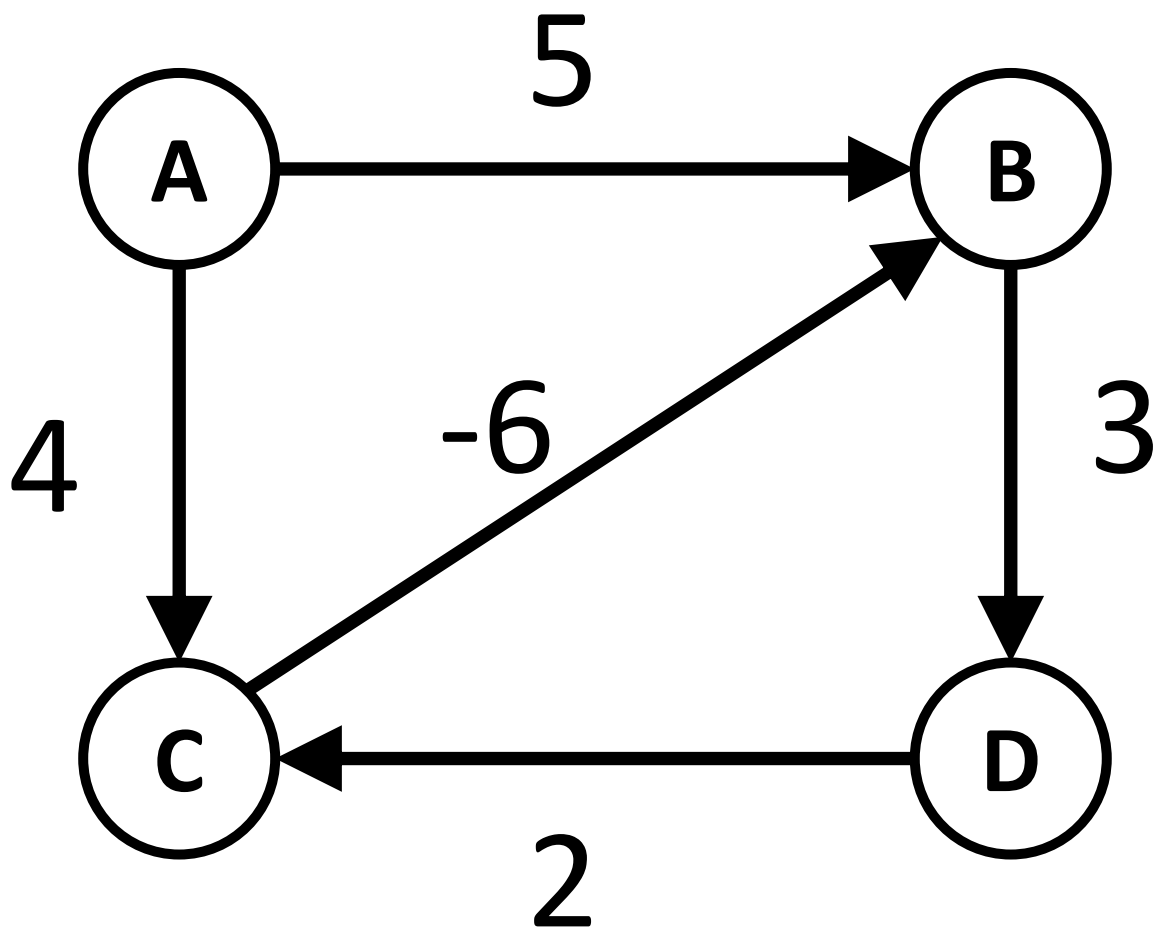
Exercise 1

- Calculate the shortest path from ***A*** to the other points of the graph using the **Bellman-Ford algorithm**.

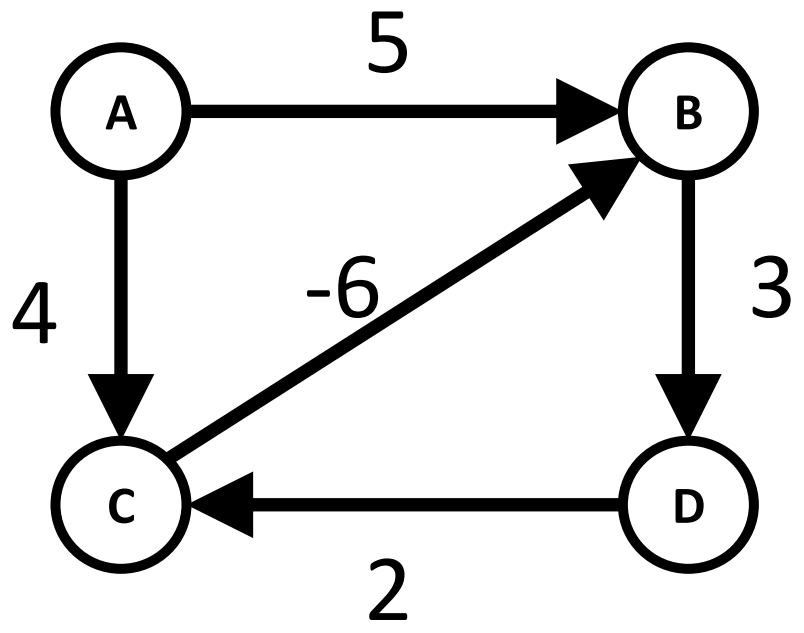


Improving cycle

- Improving cycles (or *negative cycles* in English) are infinite loops that continuously reduce the total path distance.
- They pose a real problem because each pass through the cycle can further reduce the distance, always yielding a “better” path to take.
- When a graph contains this type of cycle, there is no solution due to this infinite loop, so detecting it becomes essential.

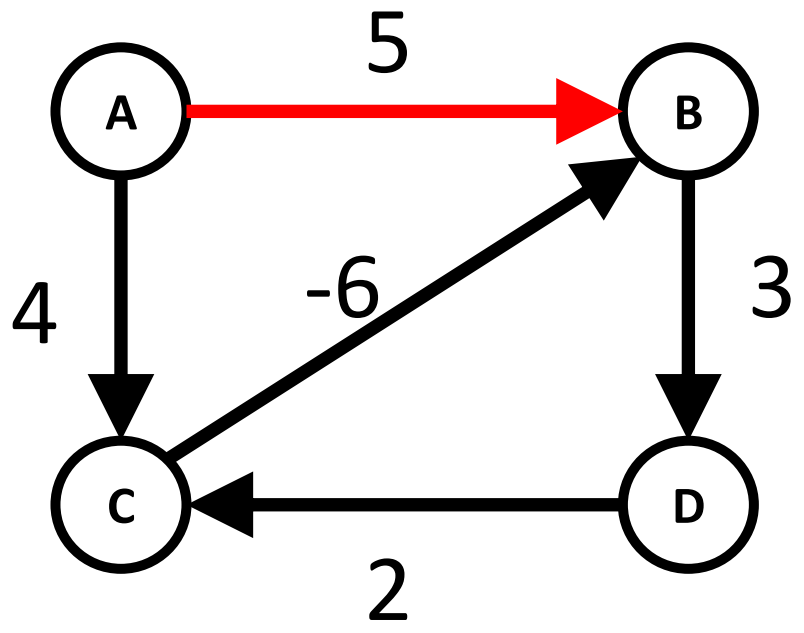

5
A
B
4
-6
3
C
D
2

	Dist	Prev
A	0	-
B	∞	-
C	∞	-
D	∞	-



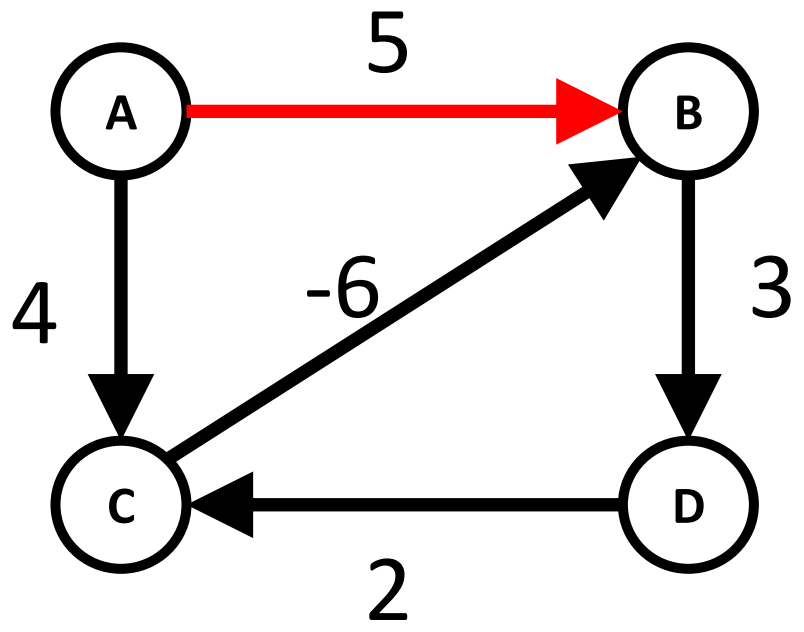
iteration 1 of 3

	Dist	Prev
A	0	-
B	∞	-
C	∞	-
D	∞	-



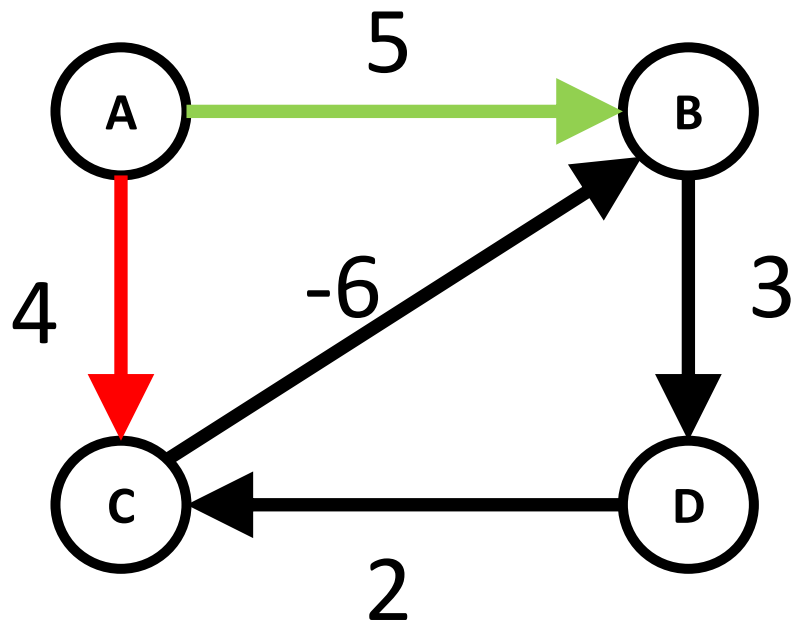
iteration 1 of 3

	Dist	Prev
A	0	-
B	5	A
C	∞	-
D	∞	-



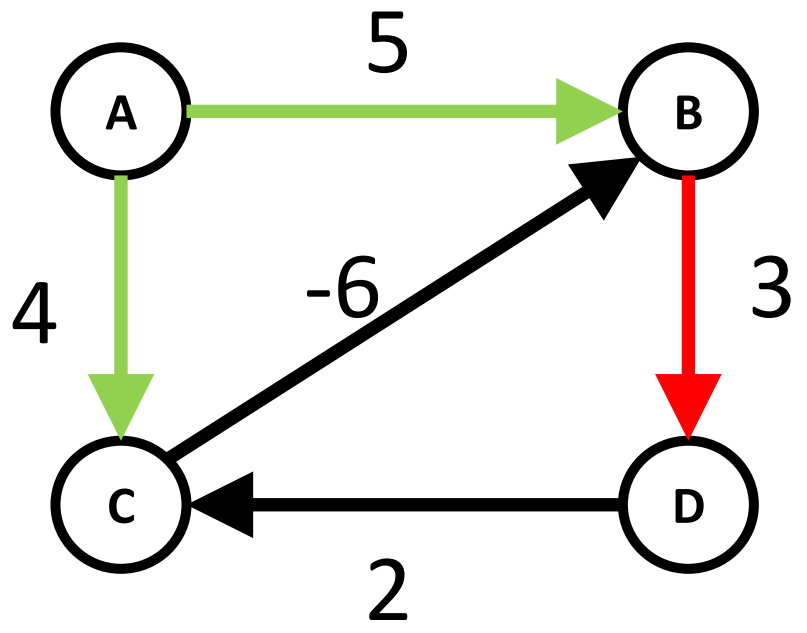
iteration 1 of 3

	Dist	Prev
A	0	-
B	5	A
C	∞	-
D	∞	-



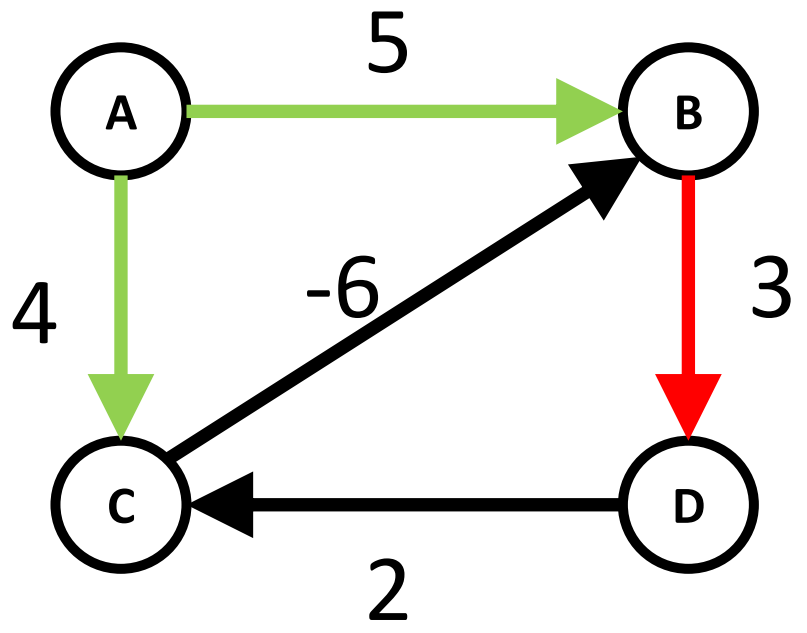
iteration 1 of 3

	Dist	Prev
A	0	-
B	5	A
C	4	A
D	∞	-



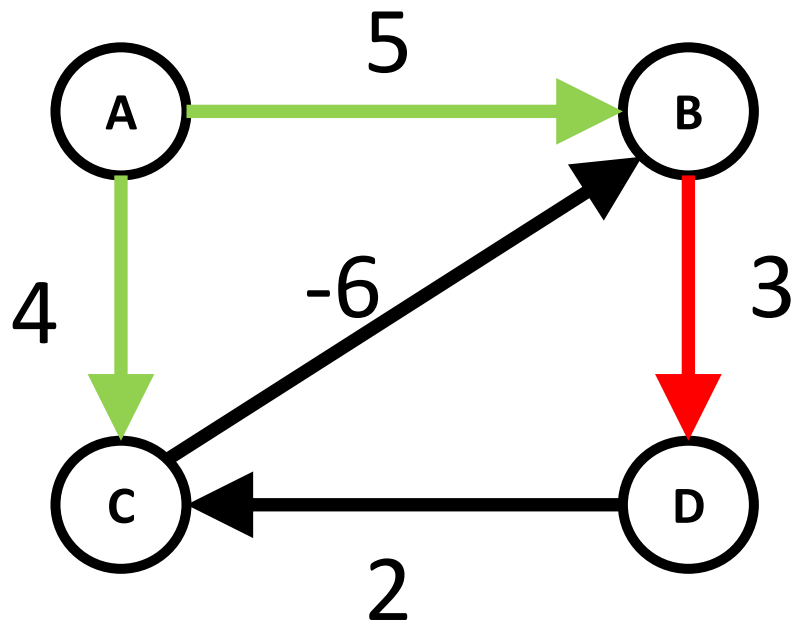
iteration 1 of 3

	Dist	Prev
A	0	-
B	5	A
C	4	A
D	∞	-



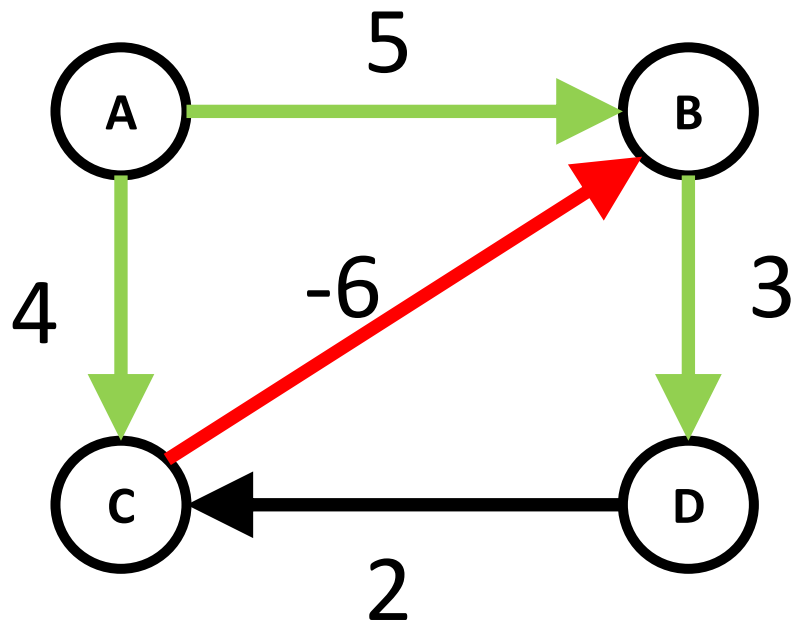
iteration 1 of 3

	Dist	Prev
A	0	-
B	5	A
C	4	A
D	8	B



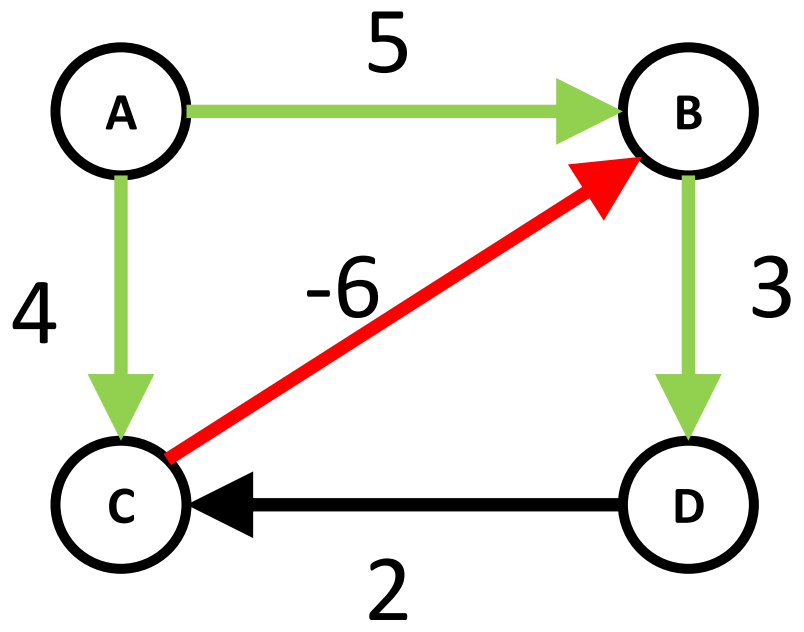
iteration 1 of 3

	Dist	Prev
A	0	-
B	5	A
C	4	A
D	8	B



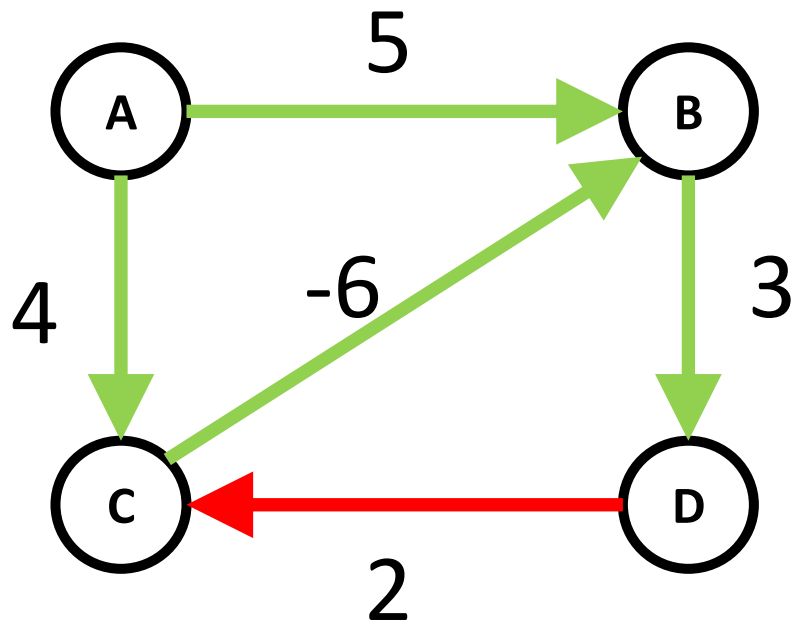
iteration 1 of 3

	Dist	Prev
A	0	-
B	-2	C
C	4	A
D	8	B



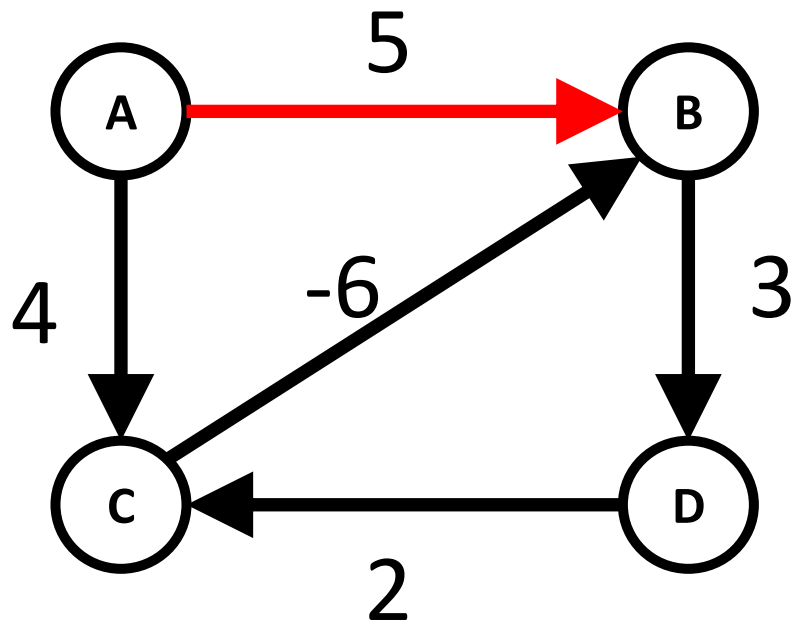
iteration 1 of 3

	Dist	Prev
A	0	-
B	-2	C
C	4	A
D	8	B



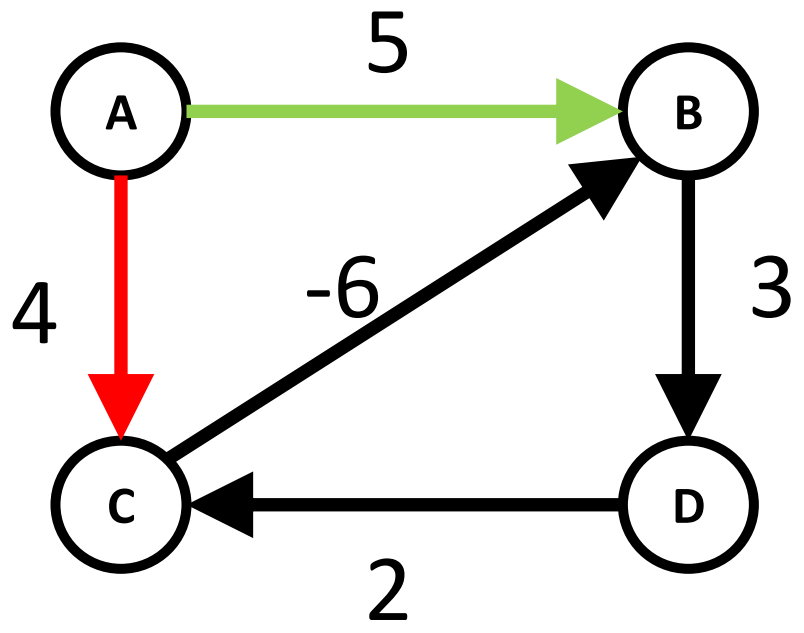
iteration 2 of 3

	Dist	Prev
A	0	-
B	-2	C
C	4	A
D	8	B



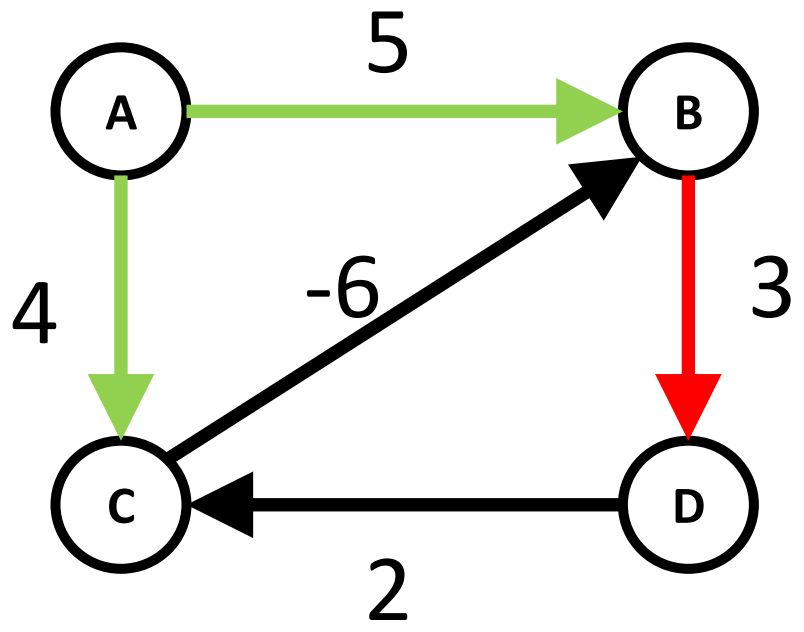
iteration 2 of 3

	Dist	Prev
A	0	-
B	-2	C
C	4	A
D	8	B



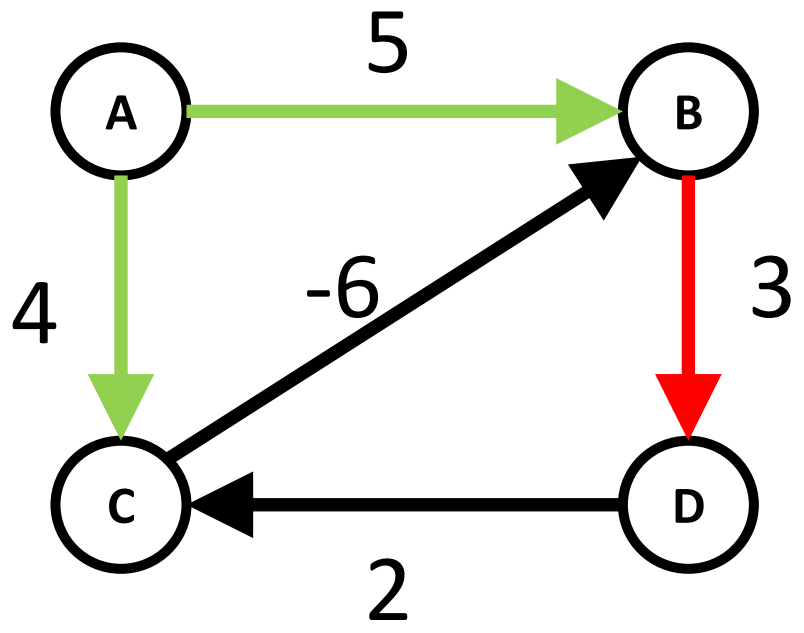
iteration 2 of 3

	Dist	Prev
A	0	-
B	-2	C
C	4	A
D	8	B



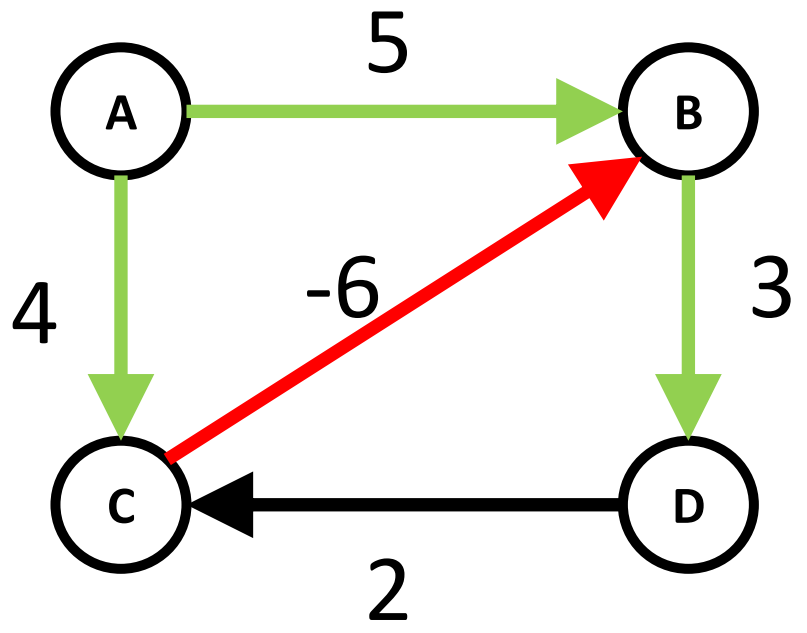
iteration 2 of 3

	Dist	Prev
A	0	-
B	-2	C
C	4	A
D	1	B



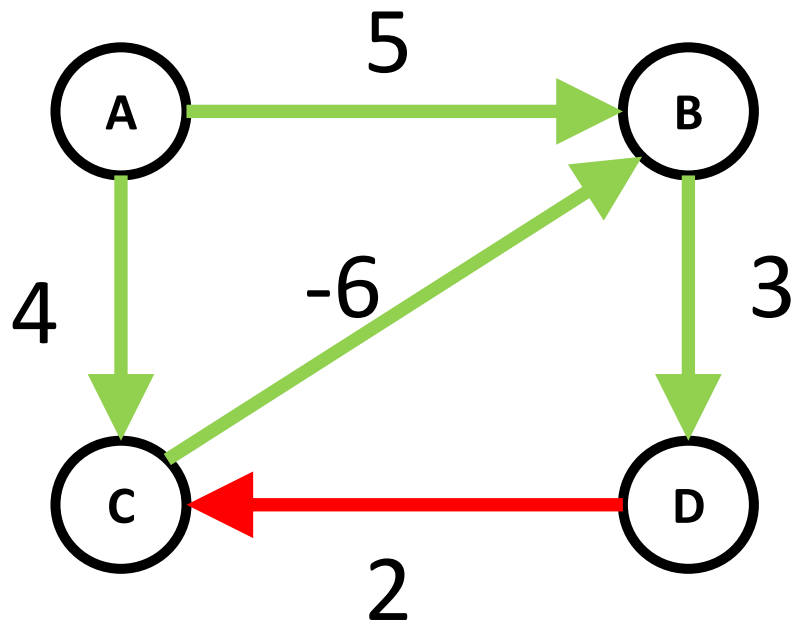
iteration 2 of 3

	Dist	Prev
A	0	-
B	-2	C
C	4	A
D	1	B



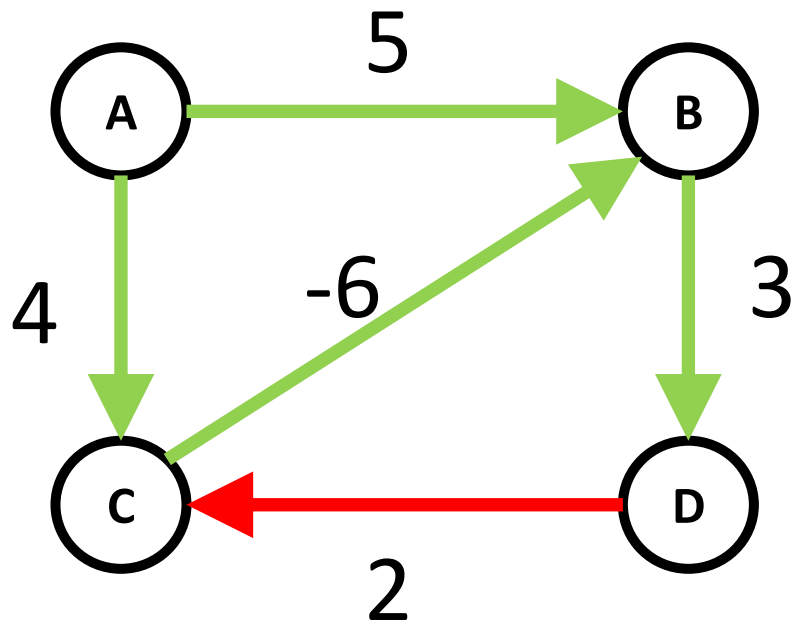
iteration 2 of 3

	Dist	Prev
A	0	-
B	-2	C
C	4	A
D	1	B



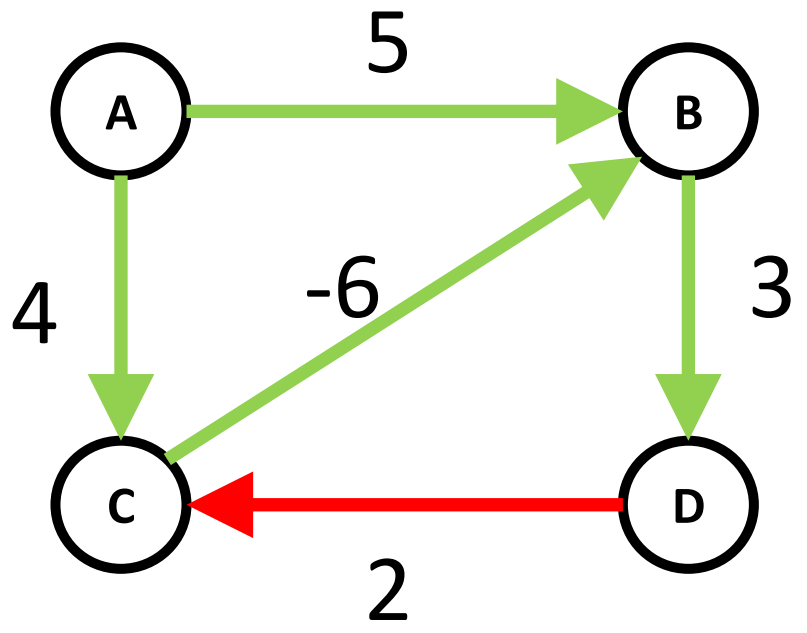
iteration 2 of 3

	Dist	Prev
A	0	-
B	-2	C
C	3	D
D	1	B



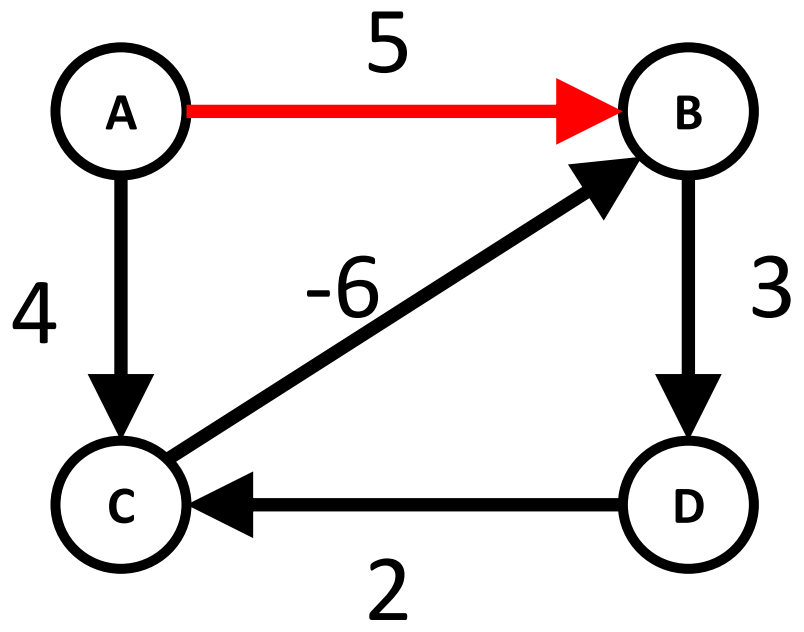
iteration 2 of 3

	Dist	Prev
A	0	-
B	-2	C
C	3	D
D	1	B



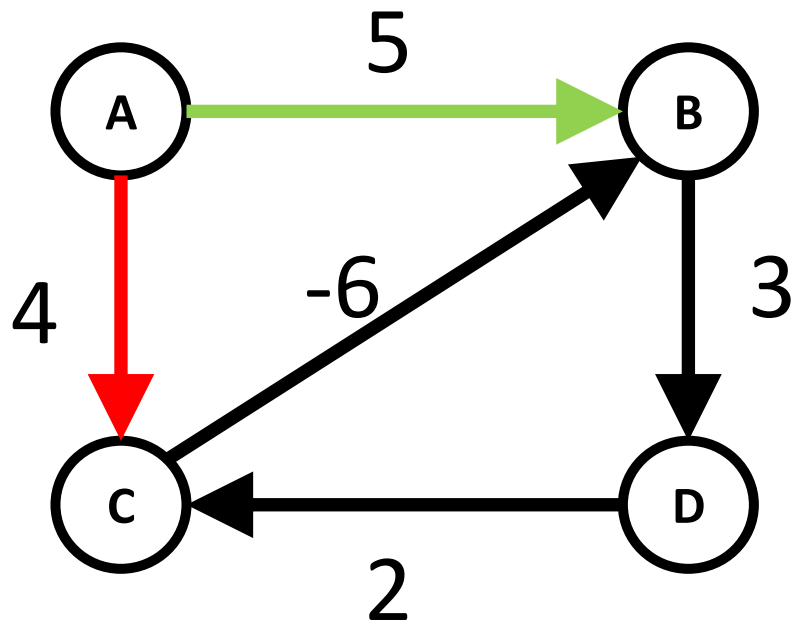
iteration 3 of 3

	Dist	Prev
A	0	-
B	-2	C
C	3	D
D	1	B



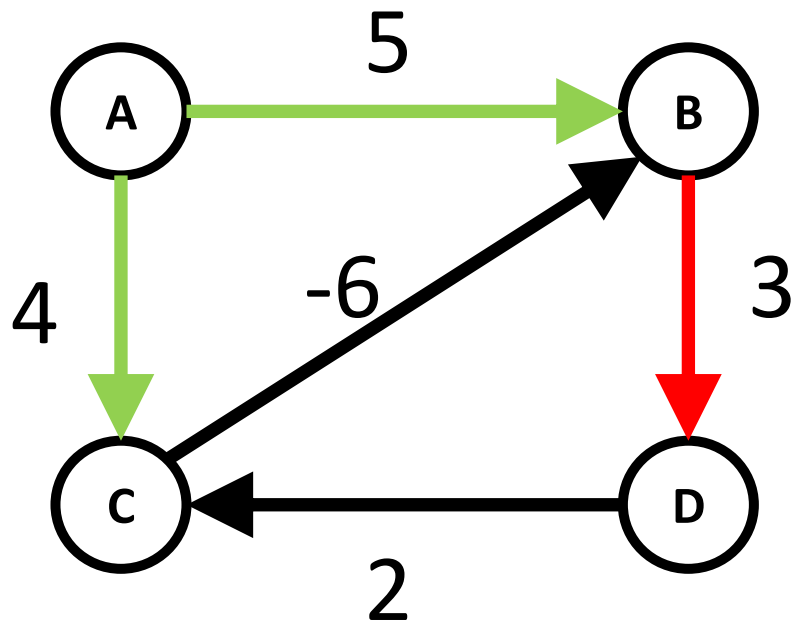
iteration 3 of 3

	Dist	Prev
A	0	-
B	-2	C
C	3	D
D	1	B



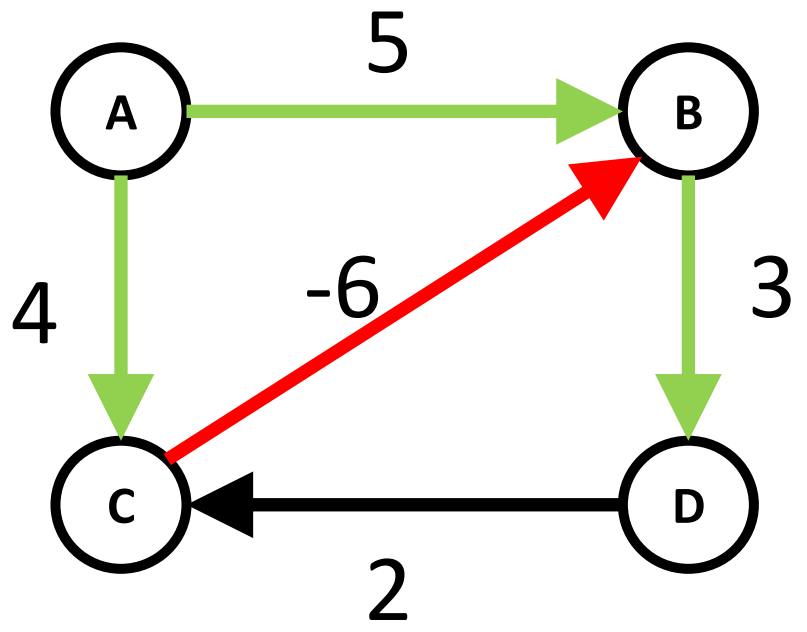
iteration 3 of 3

	Dist	Prev
A	0	-
B	-2	C
C	3	D
D	1	B



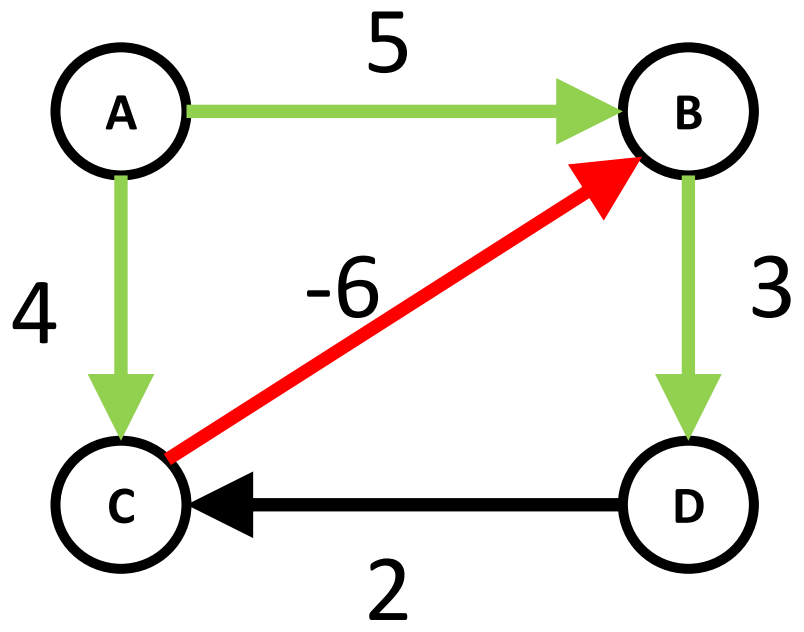
iteration 3 of 3

	Dist	Prev
A	0	-
B	-2	C
C	3	D
D	1	B



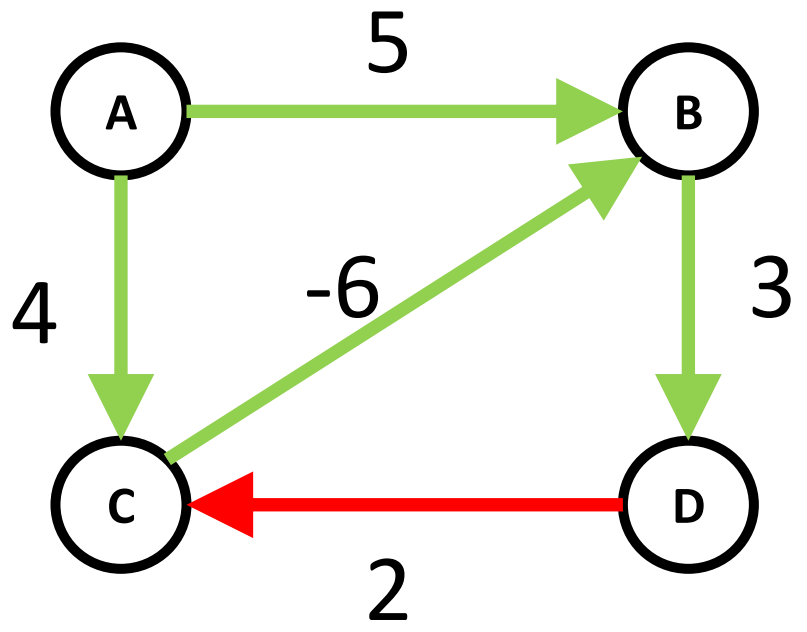
iteration 3 of 3

	Dist	Prev
A	0	-
B	-3	C
C	3	D
D	1	B



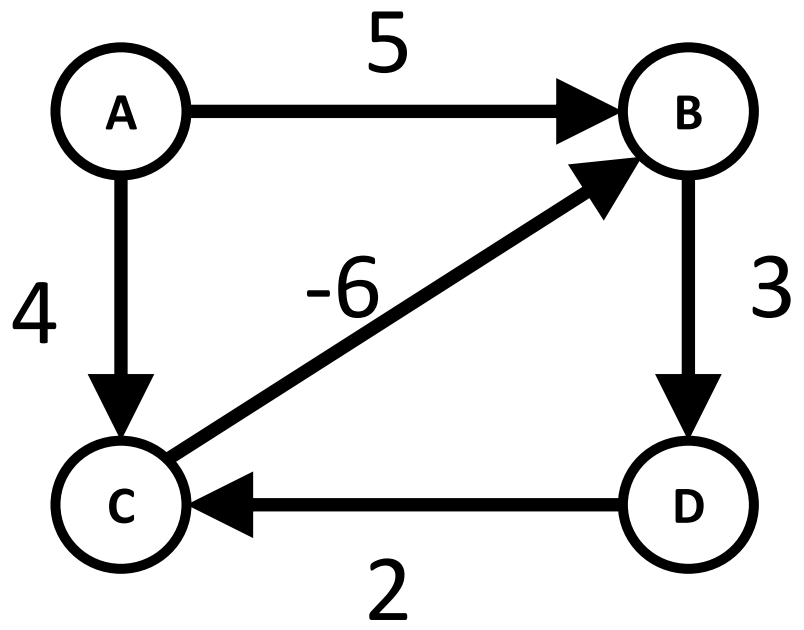
iteration 3 of 3

	Dist	Prev
A	0	-
B	-3	C
C	3	D
D	1	B



Detection of negative circuits

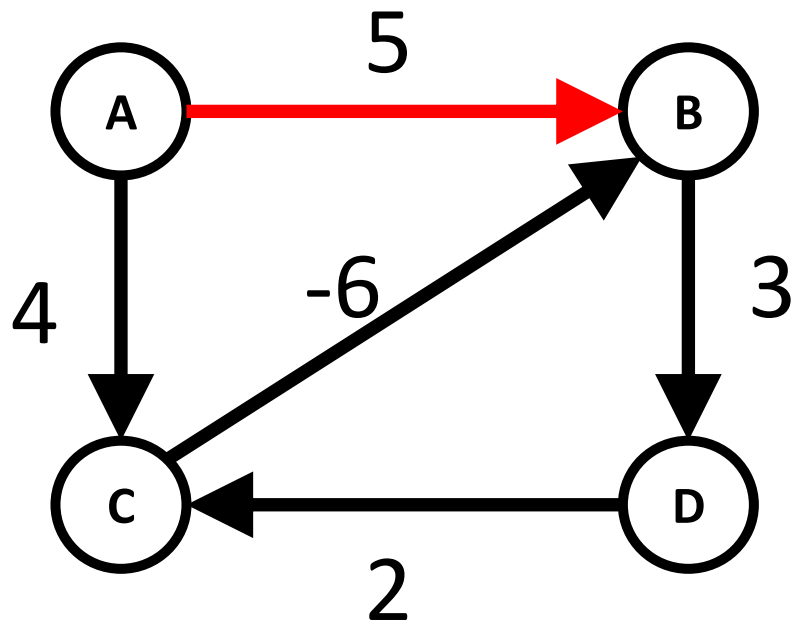
	Dist
A	0
B	-3
C	3
D	1



Dist (U) > min(Dist (U), Dist (U) + L(U, V))

Detection of negative circuits

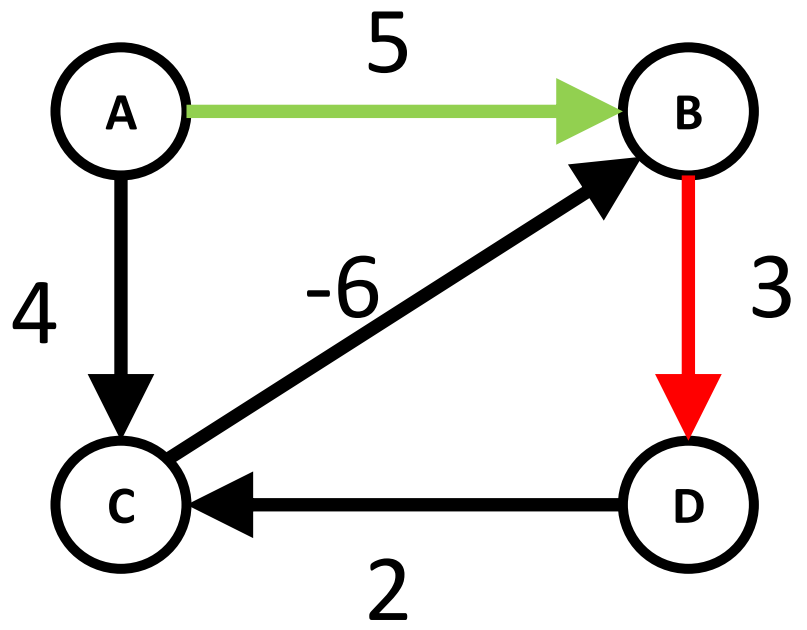
	Dist
A	0
B	-3
C	3
D	1



Dist (U) > min(Dist (U), Dist (U) + L(U, V))

Detection of negative circuits

	Dist
A	0
B	-3
C	3
D	1

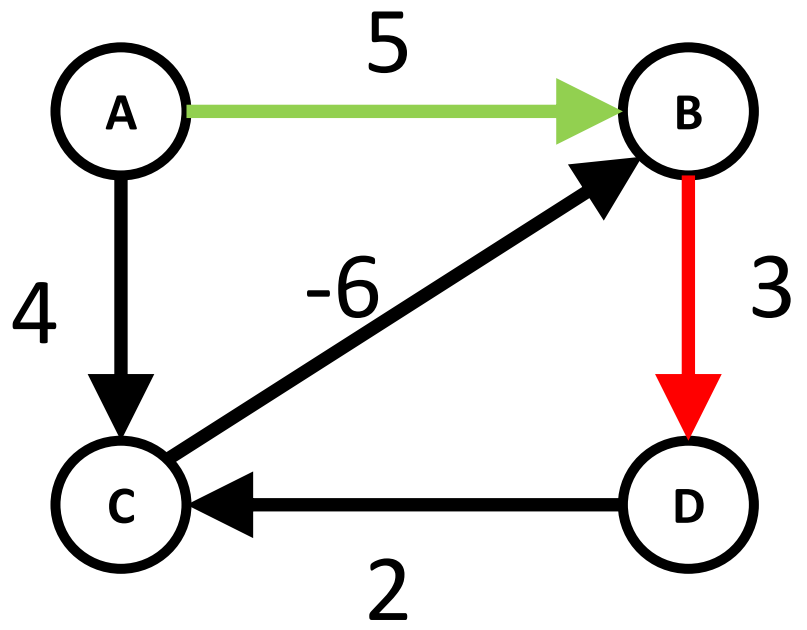


Dist (U) > min(Dist (U), Dist (U) + L(U, V))

Dist (D) = 1 > min(1, -3 + 3) = 0

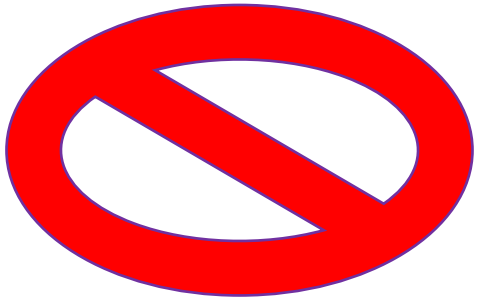
Detection of negative circuits

	Dist
A	0
B	-3
C	3
D	1



Dist (U) > min(Dist (U), Dist (U) + L(U, V))

Dist (D) = 1 > min(1, -3 + 3) = 0



Presence of negative circuits

Shortest path search algorithms

- **Dijkstra** Shortest path from one vertex to all other vertices, positive arc weight > 0
- **Bellman-Ford** Shortest path from one vertex to all other vertices, negative weight arc < 0
- **Floyd- Warshall** Shortest path between any pair of vertices

Floyd- Warshall algorithm

- Allows to calculate the shortest path for any pair (X,Y) of vertices of the graph,
- the labels are no longer an array (one label per vertex), but a matrix M of size n*n where the entry M_{ij} corresponds to the shortest path between vertices i and j .
- This algorithm is valid regardless of the weights of the arcs, including if it involves negative circuits (the algorithm allows us to prove the existence or non-existence of such circuits).
- The algorithm consists of N main iterations; for each iteration K, we compute the shortest paths between any pair of vertices with intermediate vertices belonging only to the set {1,2,…K}.
- At initialization, we calculate the shortest path between any pair of vertices that have no intermediate vertices, so we just take the length of the arcs that exist and put an infinite weight if the arc does not exist.
- Subsequently, if we note M_{ij}^{K} the value of the shortest path from i to j whose only intermediate vertices are in the set {1,2,…K}, then we have the following equality:

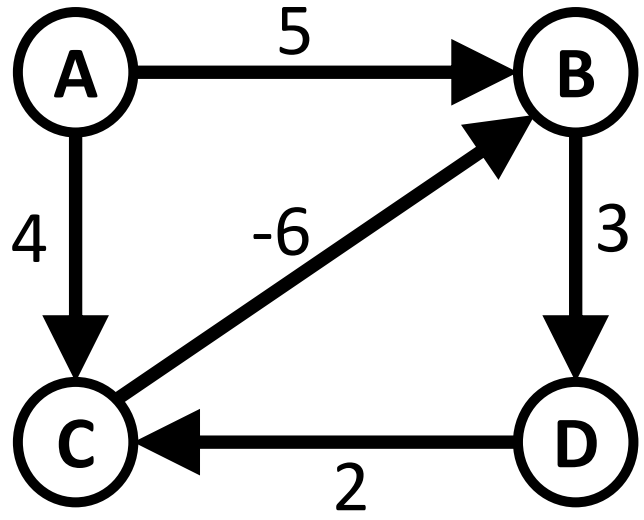
$$M_{ij}^{k} = \min(M_{ij}^{k-1}, M_{ik}^{k-1} + M_{kj}^{k-1})$$

Algorithme 11: Algorithme de Floyd

Données : Un graphe orienté pondéré $G = (X, A, W)$
Résultat : Le plus court chemin entre toute paire de sommets de G
// M : matrice des plus courts chemins
// P : matrice des prédécesseurs pour les plus courts chemins

```
1  Initialiser M à +∞
2  Initialiser P à 0
3  pour i allant de 1 à N faire
4      M_{i,i} ← 0
5      P_{i,i} ← i
6      pour tout successeur j de i faire
7          M_{i,j} ← W[i, j]
   // Calcul des matrices successives
8  pour k allant de 1 à N faire
9      pour i allant de 1 à N faire
10         pour j allant de 1 à N faire
11             si M_{i,k} + M_{k,j} < M_{i,j} alors
12                 M_{i,j} = M_{i,k} + M_{k,j}
13                 P_{i,j} = P_{k,j}
14 si ∃i|M_{i,i} < 0 alors
15     retourner Il existe un circuit de longueur négative passant par i
16 sinon
17     retourner M
```

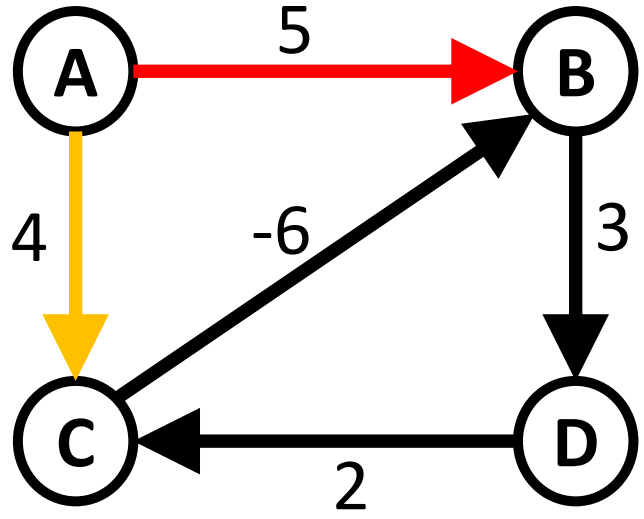
initialization



	M			
	A	B	C	D
A	0	∞	∞	∞
B	∞	0	∞	∞
C	∞	∞	0	∞
D	∞	∞	∞	0

	P			
	A	B	C	D
A	A	0	0	0
B	0	B	0	0
C	0	0	C	0
D	0	0	0	D

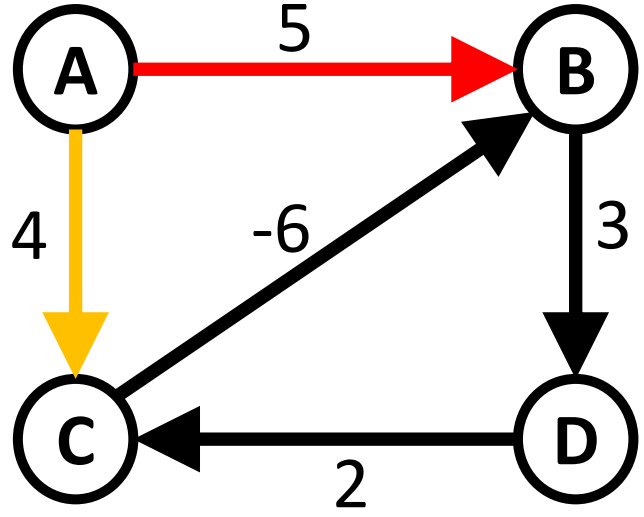
initialization



	M			
	A	B	C	D
A	0	∞	∞	∞
B	∞	0	∞	∞
C	∞	∞	0	∞
D	∞	∞	∞	0

	P			
	A	B	C	D
A	A	0	0	0
B	0	B	0	0
C	0	0	C	0
D	0	0	0	D

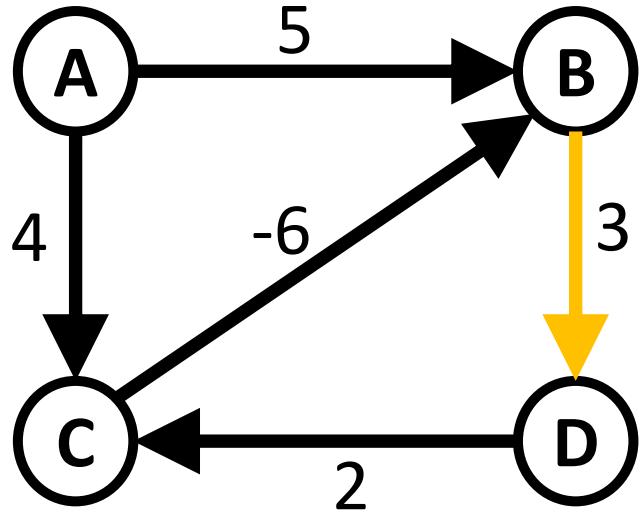
initialization



	M			
	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	∞
C	∞	∞	0	∞
D	∞	∞	∞	0

	P			
	A	B	C	D
A	A	A	A	0
B	0	B	0	0
C	0	0	C	0
D	0	0	0	D

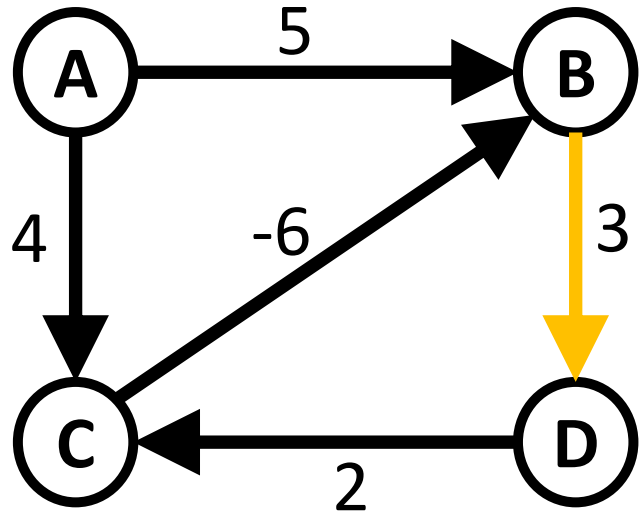
initialization



	M			
	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	∞
C	∞	∞	0	∞
D	∞	∞	∞	0

	P			
	A	B	C	D
A	A	A	A	0
B	0	B	0	0
C	0	0	C	0
D	0	0	0	D

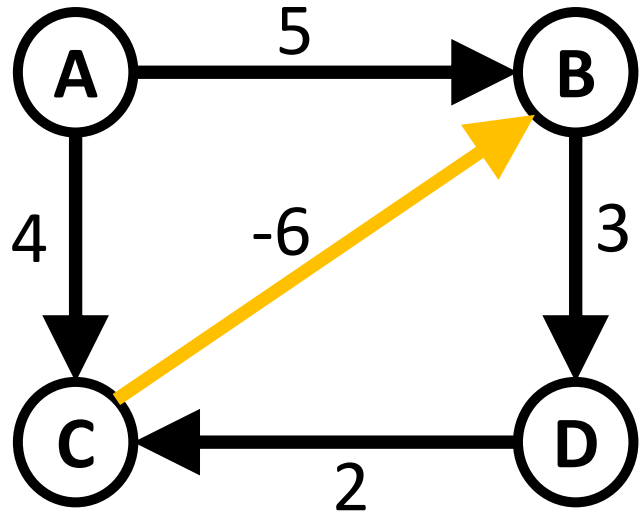
initialization



	M			
	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	3
C	∞	∞	0	∞
D	∞	∞	∞	0

	P			
	A	B	C	D
A	A	A	A	0
B	0	B	0	B
C	0	0	C	0
D	0	0	0	D

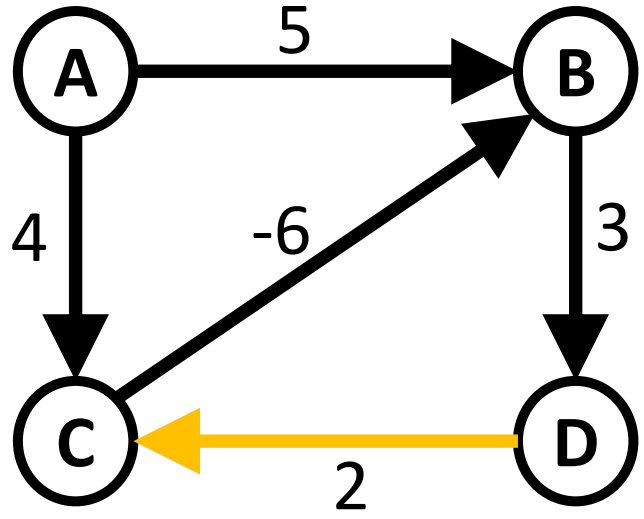
initialization



	M			
	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	∞	0

	P			
	A	B	C	D
A	A	A	A	0
B	0	B	0	B
C	0	C	C	0
D	0	0	0	D

initialization



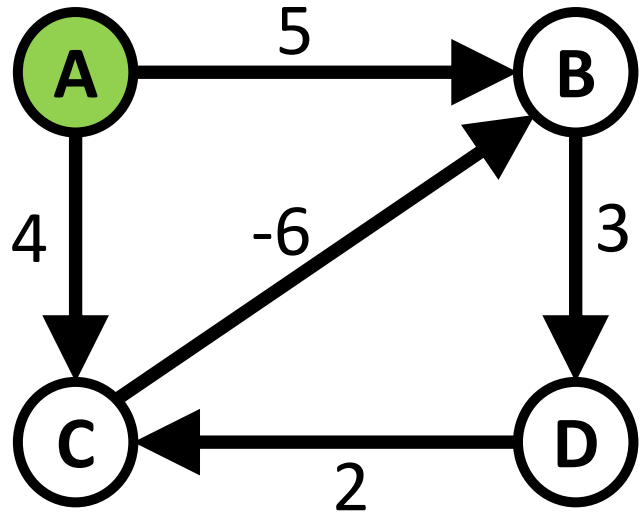
	M			
	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

	P			
	A	B	C	D
A	A	A	A	0
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 1

i = 1

j = 1



M[i , j] > M[i , k]+ M[k, j]

0 > 0 + 0

M

	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

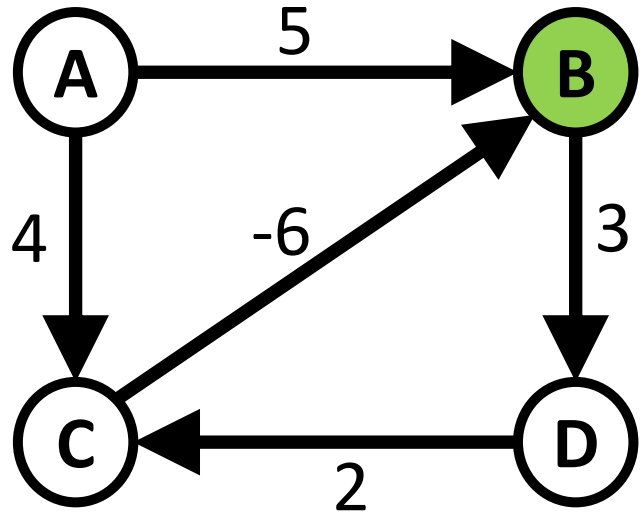
P

	A	B	C	D
A	A	A	A	0
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 1

j = 1



$M[i, j] > M[i, k] + M[k, j]$

$0 > 5 + \infty$

M

	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

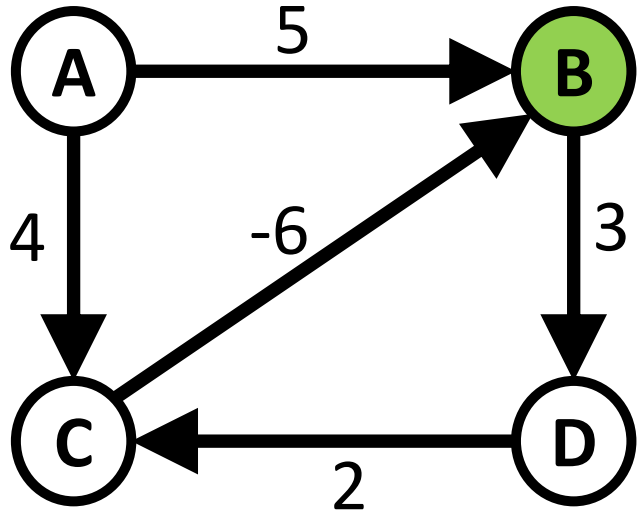
P

	A	B	C	D
A	A	A	A	0
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 1

j = 2



$M[i, j] > M[i, k] + M[k, j]$

$5 > 5 + \infty$

M

	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

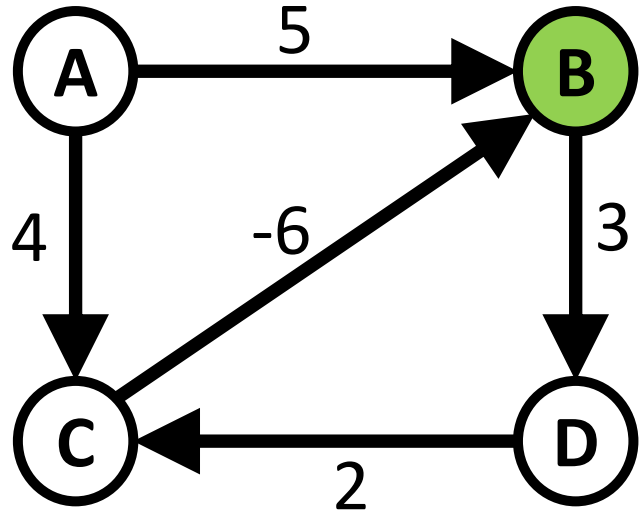
P

	A	B	C	D
A	A	A	A	0
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 1

j = 3



$M[i, j] > M[i, k] + M[k, j]$

$4 > 5 + \infty$

M

	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

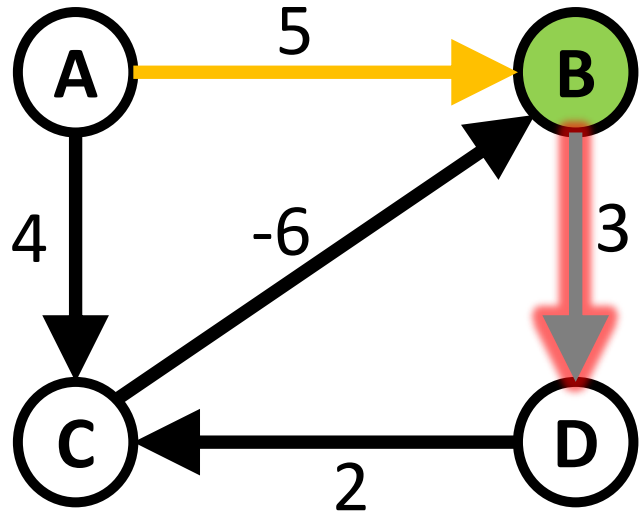
P

	A	B	C	D
A	A	A	A	0
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 1

j = 4



$M[i, j] > M[i, k] + M[k, j]$

$\infty > 5 + 3$

M

	A	B	C	D
A	0	5	4	∞
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

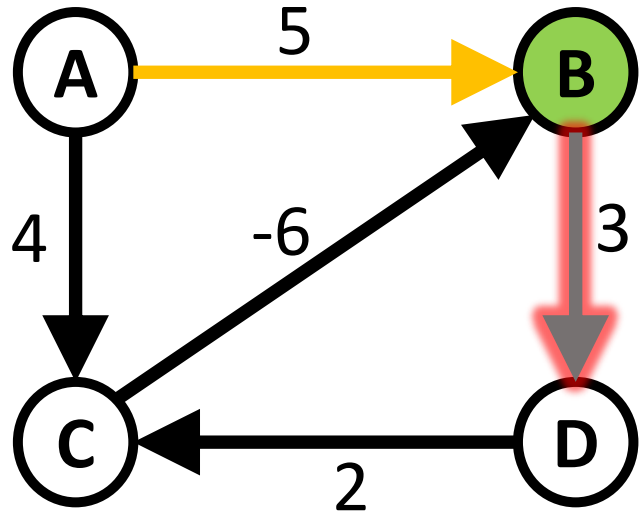
P

	A	B	C	D
A	A	A	A	0
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 1

j = 4



$M[i, j] > M[i, k] + M[k, j]$

$\infty > 8$

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

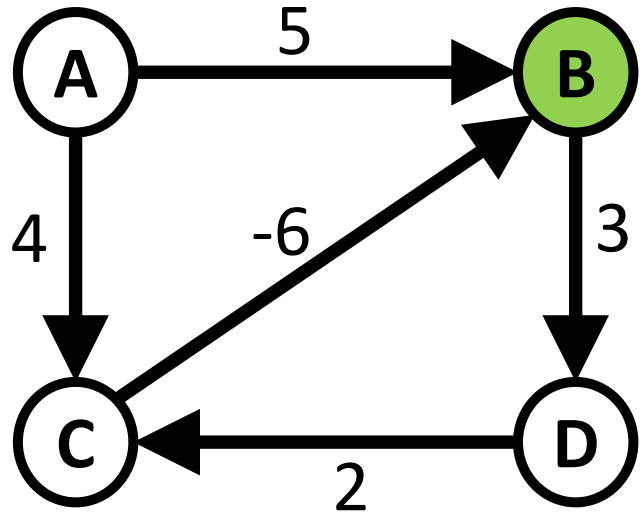
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 2

j = 1



$M[i, j] > M[i, k] + M[k, j]$

$\infty > 0 + \infty$

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

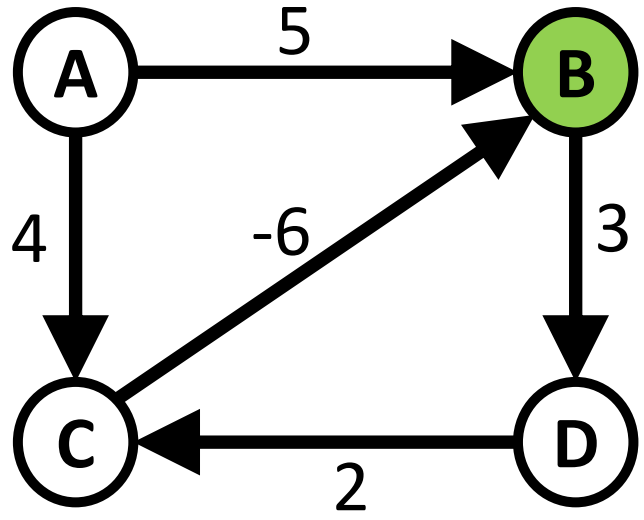
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 2

j = 2



M[i , j] > M[i , k]+ M[k, j]

0 > 0 + 0

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

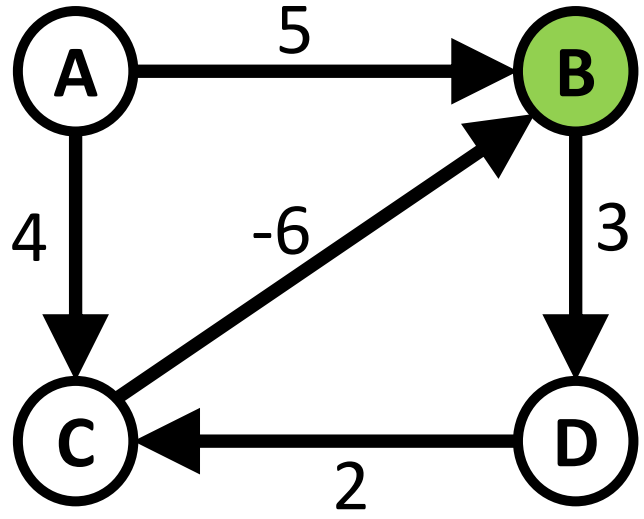
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 2

j = 3



$M[i, j] > M[i, k] + M[k, j]$

$\infty > 0 + \infty$

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

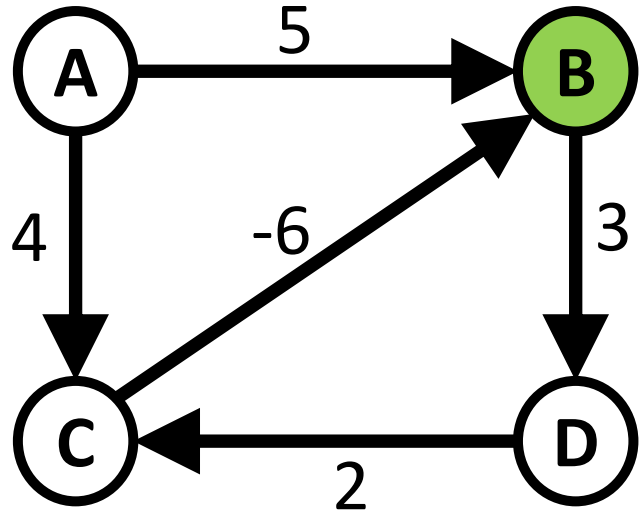
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 2

j = 4



$M[i, j] > M[i, k] + M[k, j]$

$3 > 0 + 3$

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

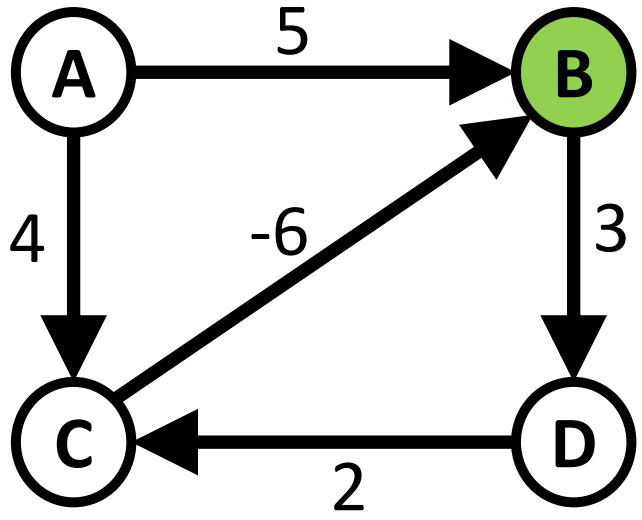
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 3

j = 1



M[i , j] > M[i , k]+ M[k, j]

∞ > -6 + ∞

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

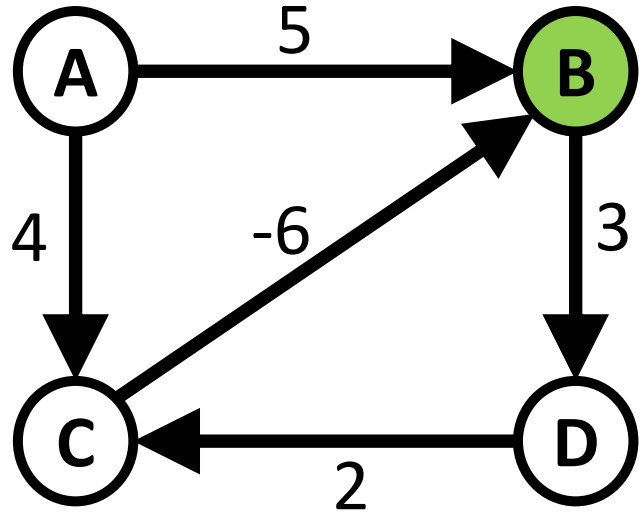
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 3

j = 2



M[i , j] > M[i , k]+ M[k, j]

-6 > -6 + 0

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

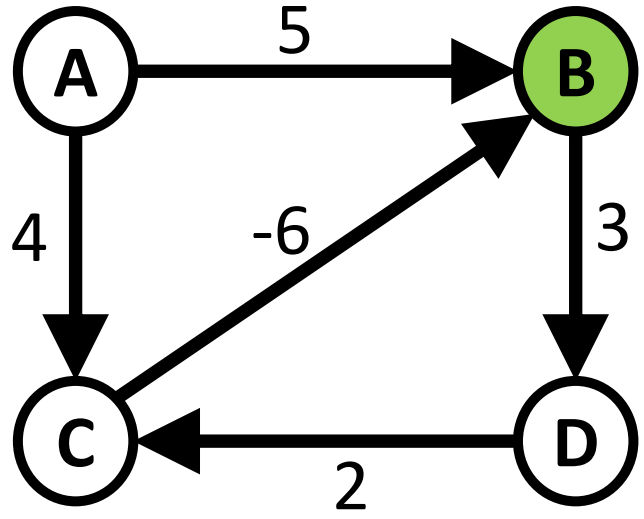
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 3

j = 3



$M[i, j] > M[i, k] + M[k, j]$

$0 > -6 + \infty$

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

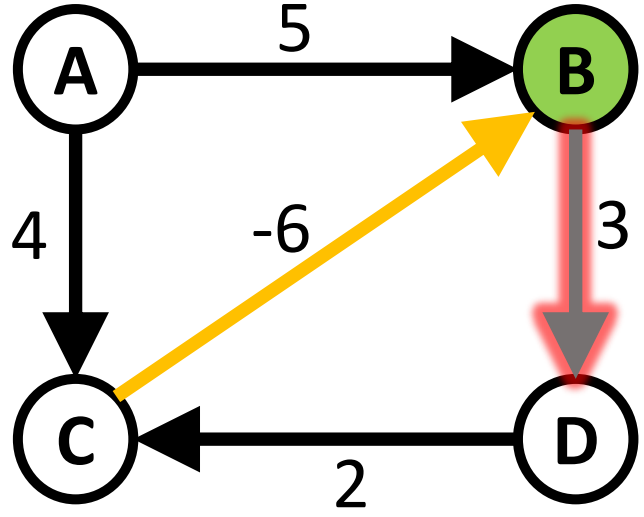
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 3

j = 4



M[i , j] > M[i , k]+ M[k, j]

∞ > -6 + 3

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	∞
D	∞	∞	2	0

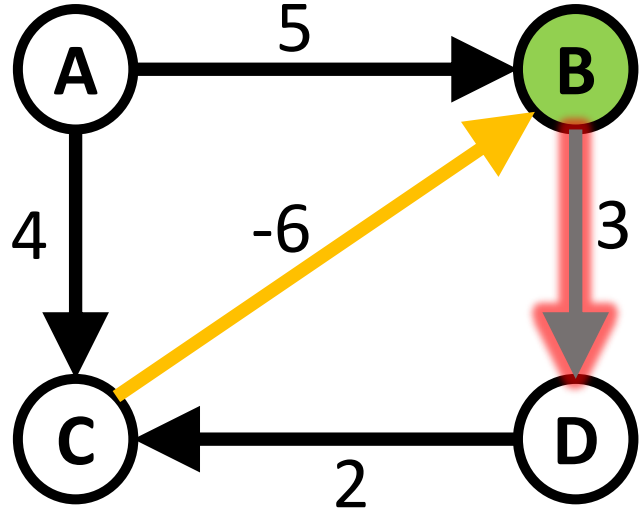
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	0
D	0	0	D	D

K = 2

i = 3

j = 4



M[i , j] > M[i , k]+ M[k, j]

∞ > - 3

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

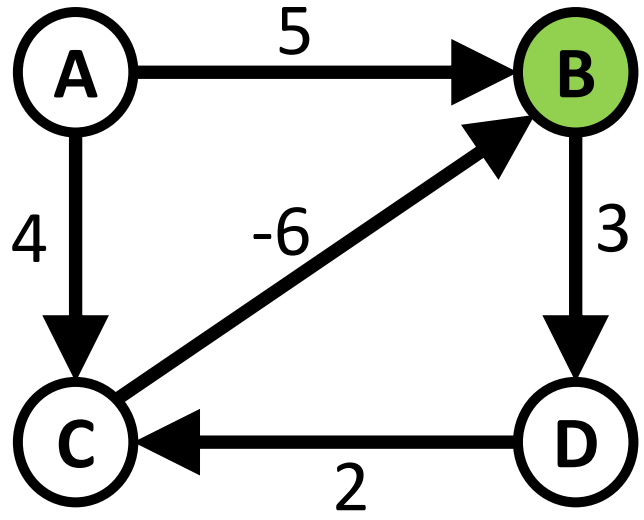
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 2

i = 4

j = 1



$M[i, j] > M[i, k] + M[k, j]$

$\infty > \infty + \infty$

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

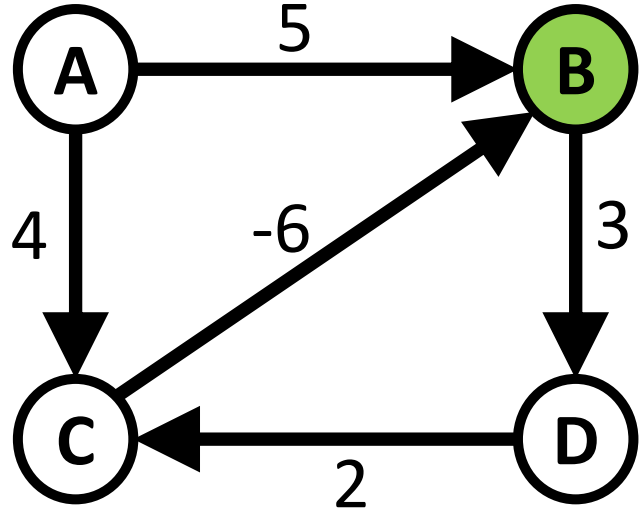
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 2

i = 4

j = 2



M[i , j] > M[i , k]+ M[k, j]

∞ > ∞ + 0

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

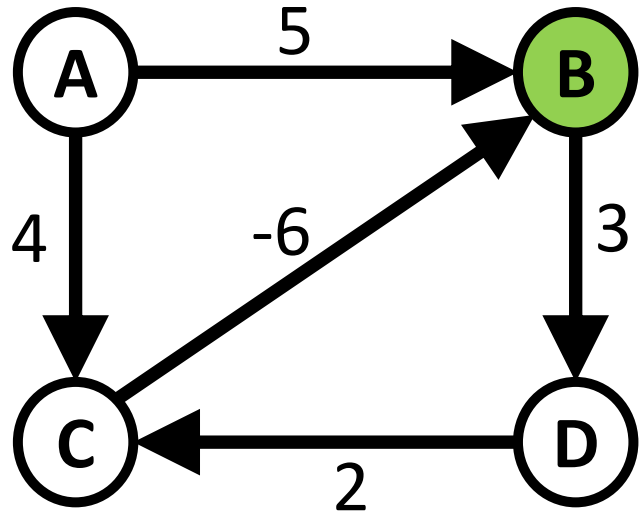
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 2

i = 4

j = 3



M[i , j] > M[i , k]+ M[k, j]

2 > ∞ + 0

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

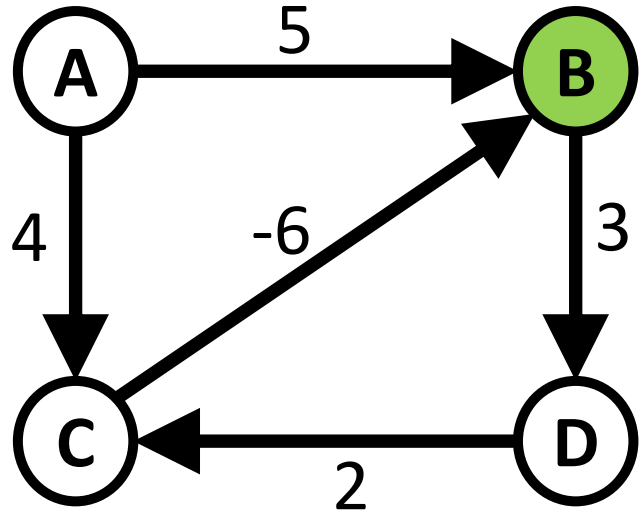
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 2

i = 4

j = 4



M[i , j] > M[i , k]+ M[k, j]

0 > ∞ + 3

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

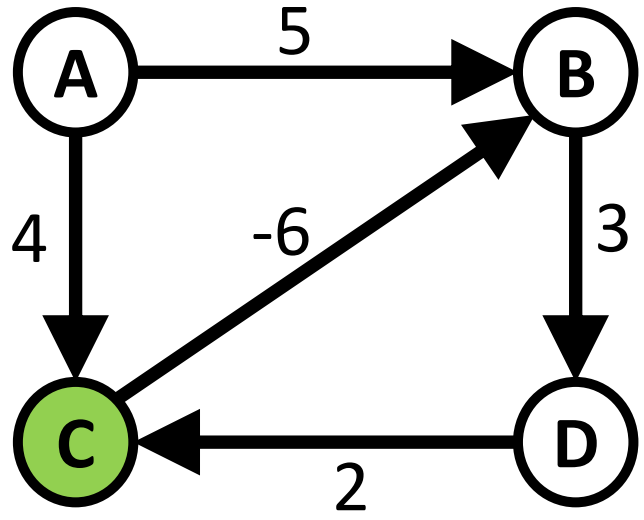
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 1

j = 1



M[i , j] > M[i , k]+ M[k, j]

0 > 4 + ∞

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

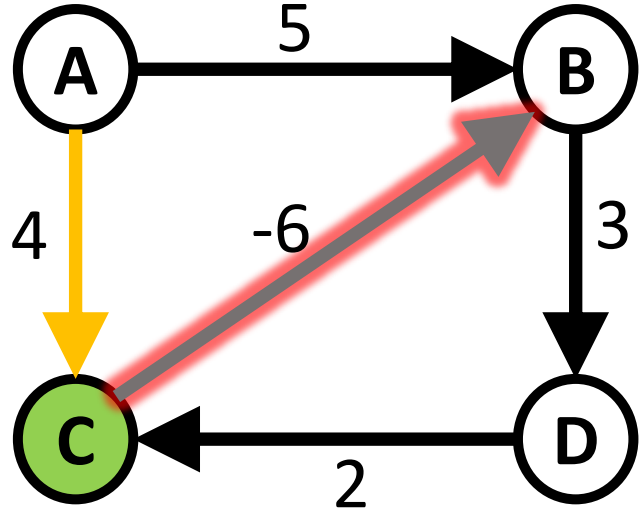
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 1

j = 2



M[i , j] > M[i , k]+ M[k, j]

5 > 4 + -6

M

	A	B	C	D
A	0	5	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

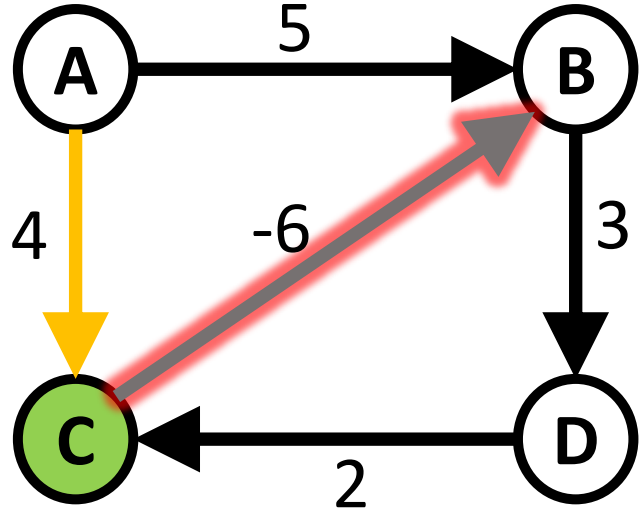
P

	A	B	C	D
A	A	A	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 1

j = 2



M[i , j] > M[i , k]+ M[k, j]

5 > -2

M

	A	B	C	D
A	0	-2	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

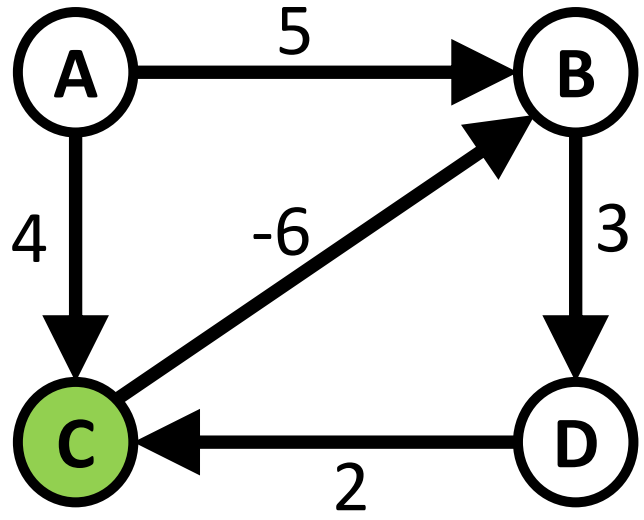
P

	A	B	C	D
A	A	C	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 1

j = 3



M[i , j] > M[i , k]+ M[k, j]

4 > 4 + ∞

M

	A	B	C	D
A	0	-2	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

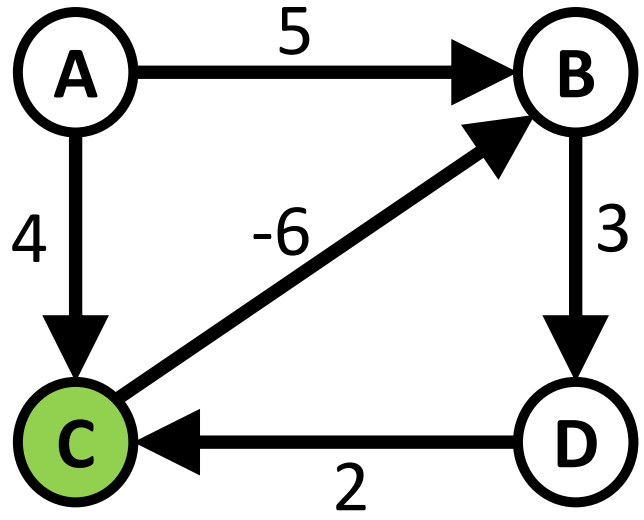
P

	A	B	C	D
A	A	C	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 1

j = 4



M[i , j] > M[i , k]+ M[k, j]

8 > 4 + -3

M

	A	B	C	D
A	0	-2	4	8
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

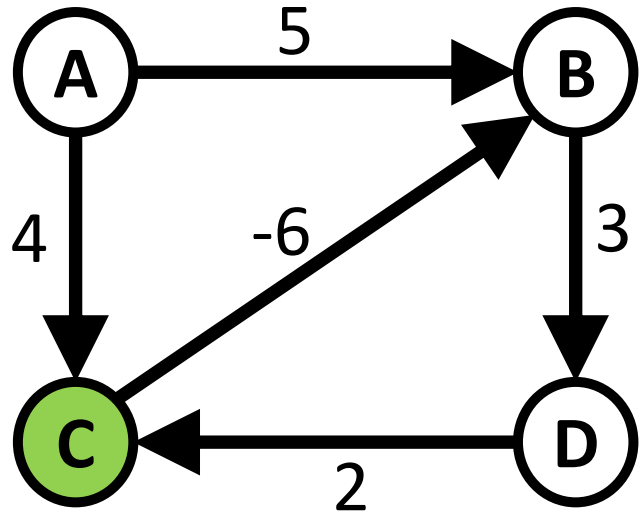
P

	A	B	C	D
A	A	C	A	B
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 1

j = 4



M[i , j] > M[i , k]+ M[k, j]

8 > 1

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

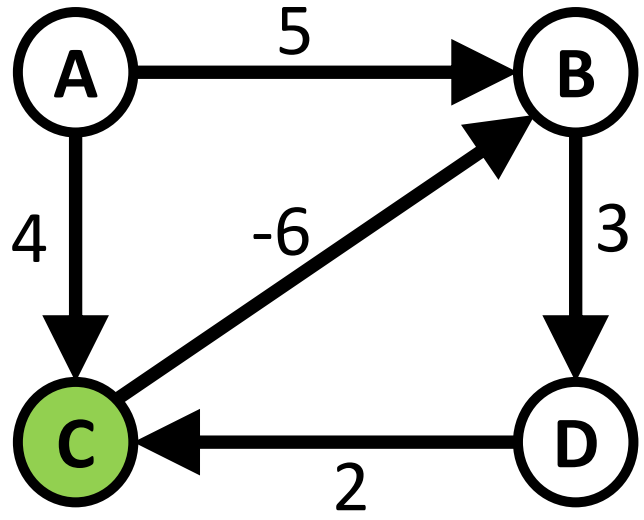
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 1

j = 4



M[i , j] > M[i , k]+ M[k, j]

8 > 1

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

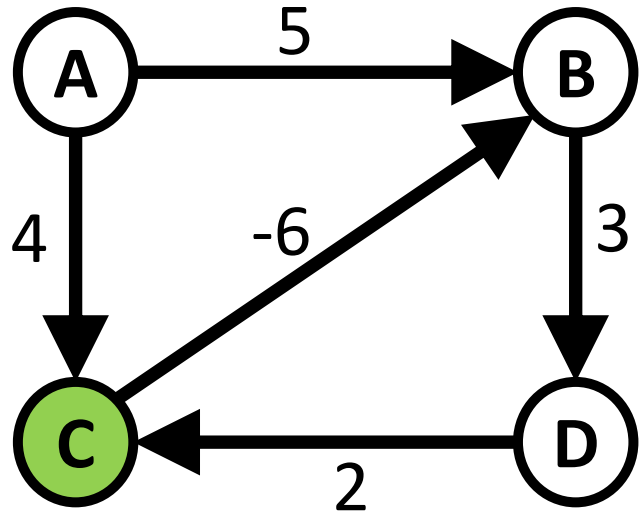
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 1

j = 4



M[i , j] > M[i , k]+ M[k, j]

8 > 1

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

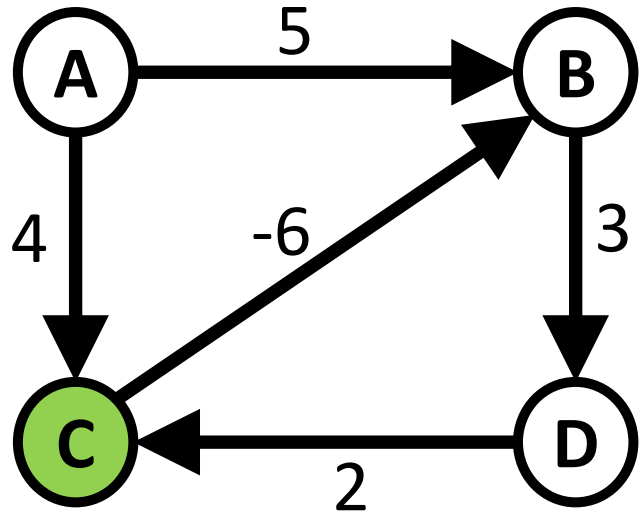
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 2

j = 1



M[i , j] > M[i , k]+ M[k, j]

∞ > ∞ + ∞

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

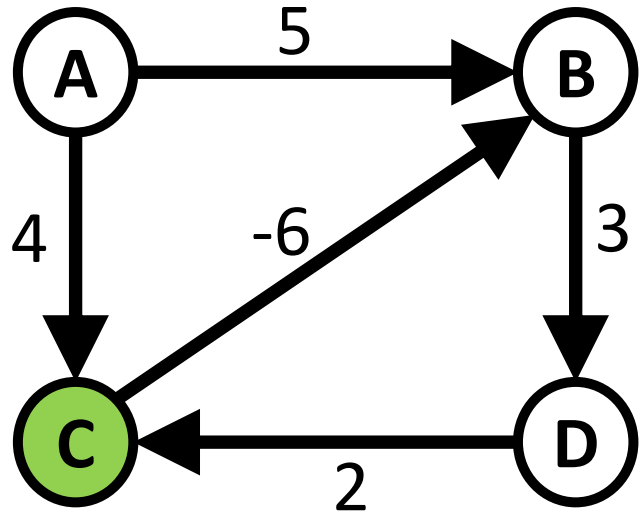
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 2

j = 2



M[i , j] > M[i , k]+ M[k, j]

0 > ∞ - 6

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

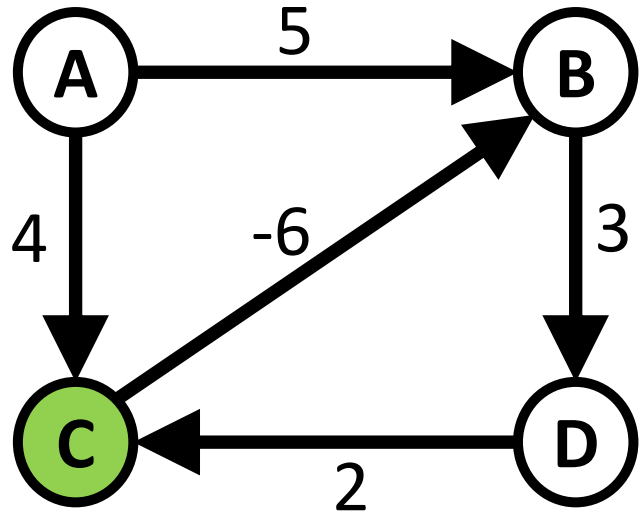
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 2

j = 3



M[i , j] > M[i , k]+ M[k, j]

∞ > ∞ - 6

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

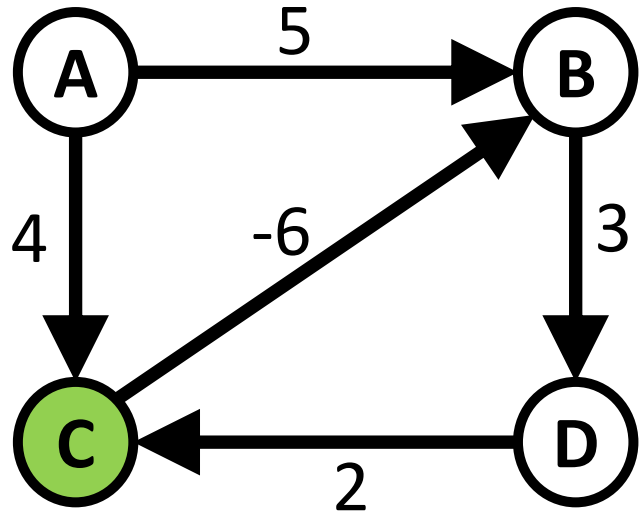
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 2

j = 4



M[i , j] > M[i , k]+ M[k, j]

3 > ∞ - 6

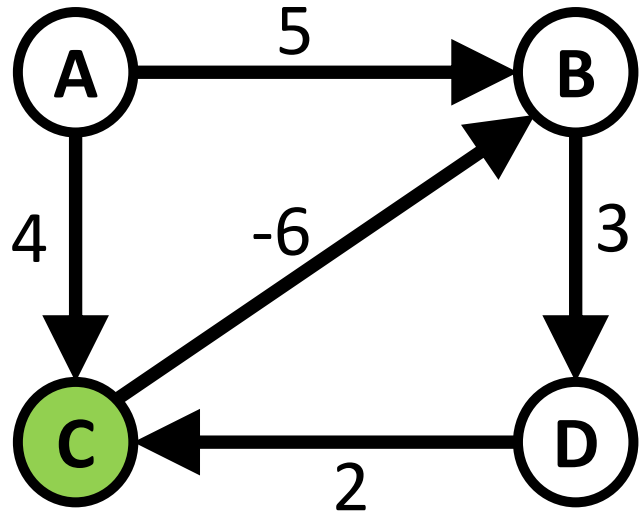
	M			
	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

	P			
	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 3

j = 1



M[i , j] > M[i , k]+ M[k, j]

∞ > ∞ + 0

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

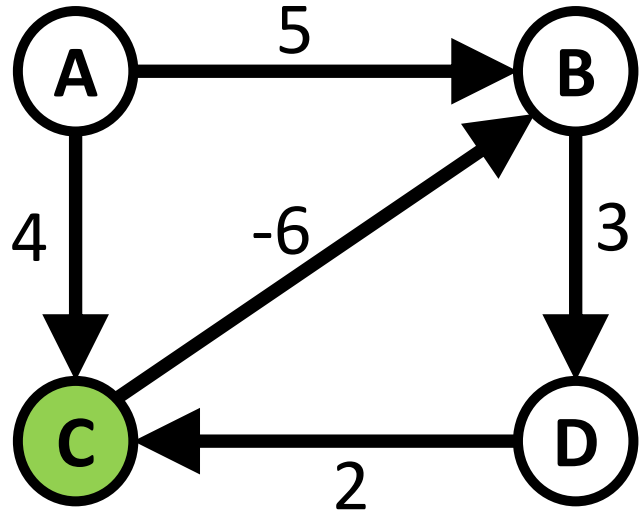
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 3

j = 2



M[i , j] > M[i , k]+ M[k, j]

-6 > -6 + 0

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

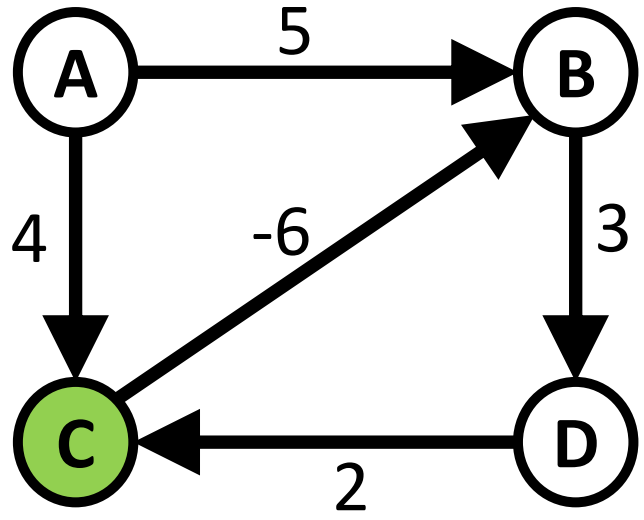
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 3

j = 3



M[i , j] > M[i , k]+ M[k, j]

0 > 0 + 0

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

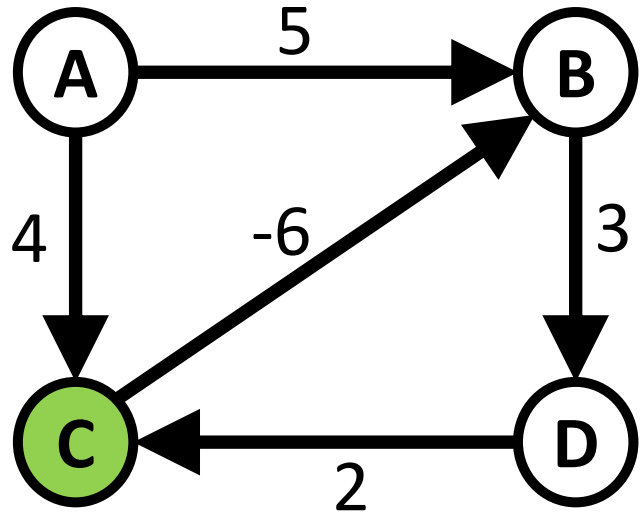
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 3

j = 4



M[i , j] > M[i , k]+ M[k, j]

-3 > 0 + -3

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

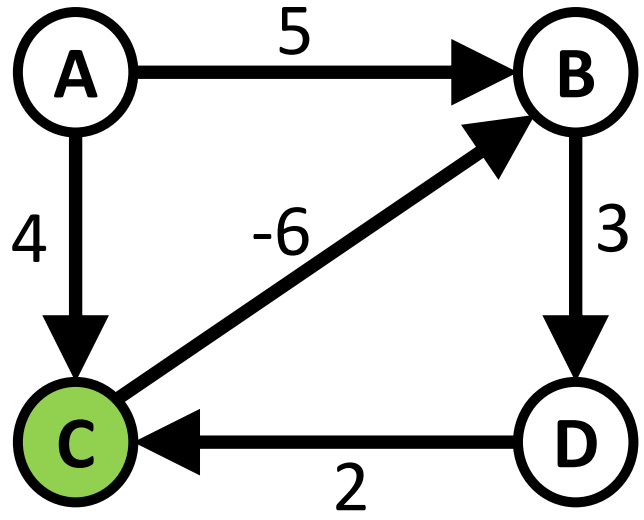
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 4

j = 1



$M[i, j] > M[i, k] + M[k, j]$

$\infty > 2 + \infty$

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

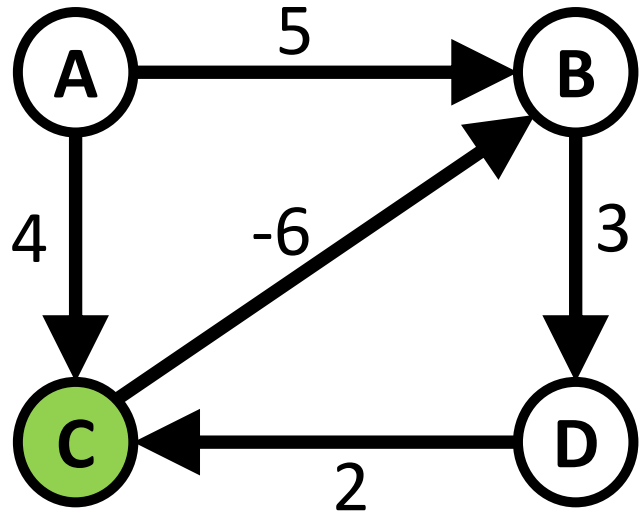
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 4

j = 2



$M[i, j] > M[i, k] + M[k, j]$

$\infty > 2 + \infty$

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

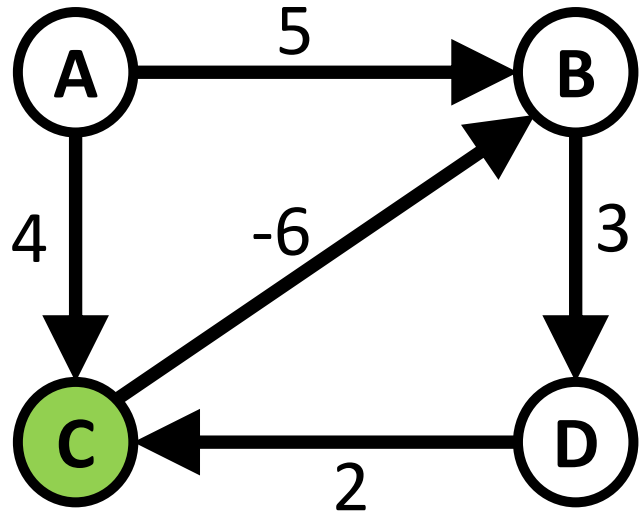
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 4

j = 3



M[i , j] > M[i , k]+ M[k, j]

2 > 2 + 0

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

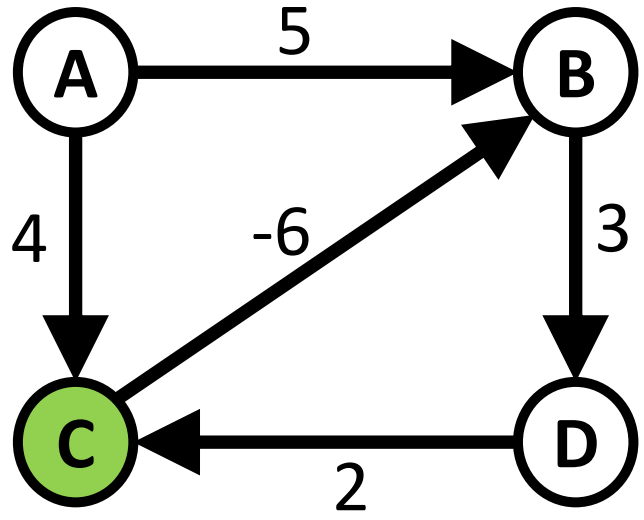
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 4

j = 3



M[i , j] > M[i , k]+ M[k, j]

0 > 2 + -3

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	0

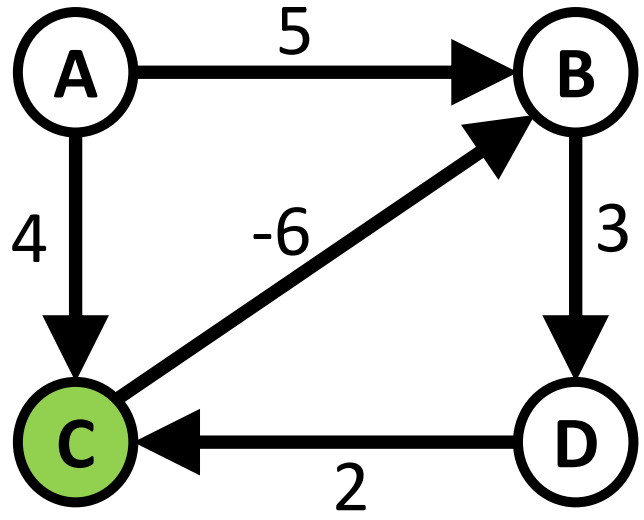
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	D

K = 3

i = 4

j = 3



M[i , j] > M[i , k]+ M[k, j]

0 > -1

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	-1

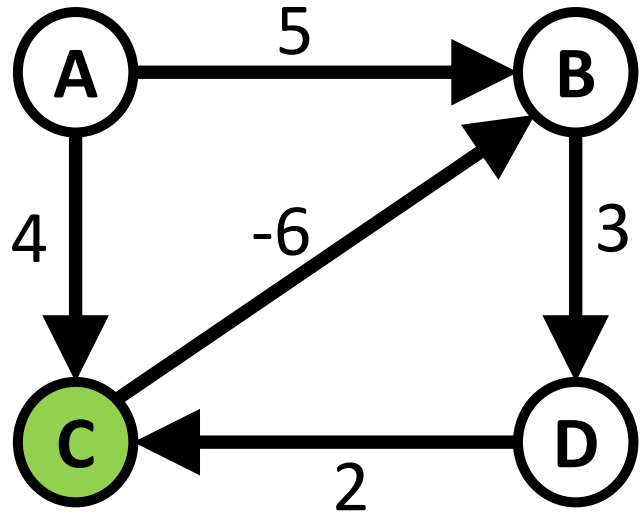
P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	C

K = 3

i = 4

j = 3



M[D, D] < 0 A negative cycle exists

M

	A	B	C	D
A	0	-2	4	1
B	∞	0	∞	3
C	∞	-6	0	-3
D	∞	∞	2	-1

P

	A	B	C	D
A	A	C	A	C
B	0	B	0	B
C	0	C	C	B
D	0	0	D	C

Exercise 2

- Compute the shortest path between each pair of Vertex,