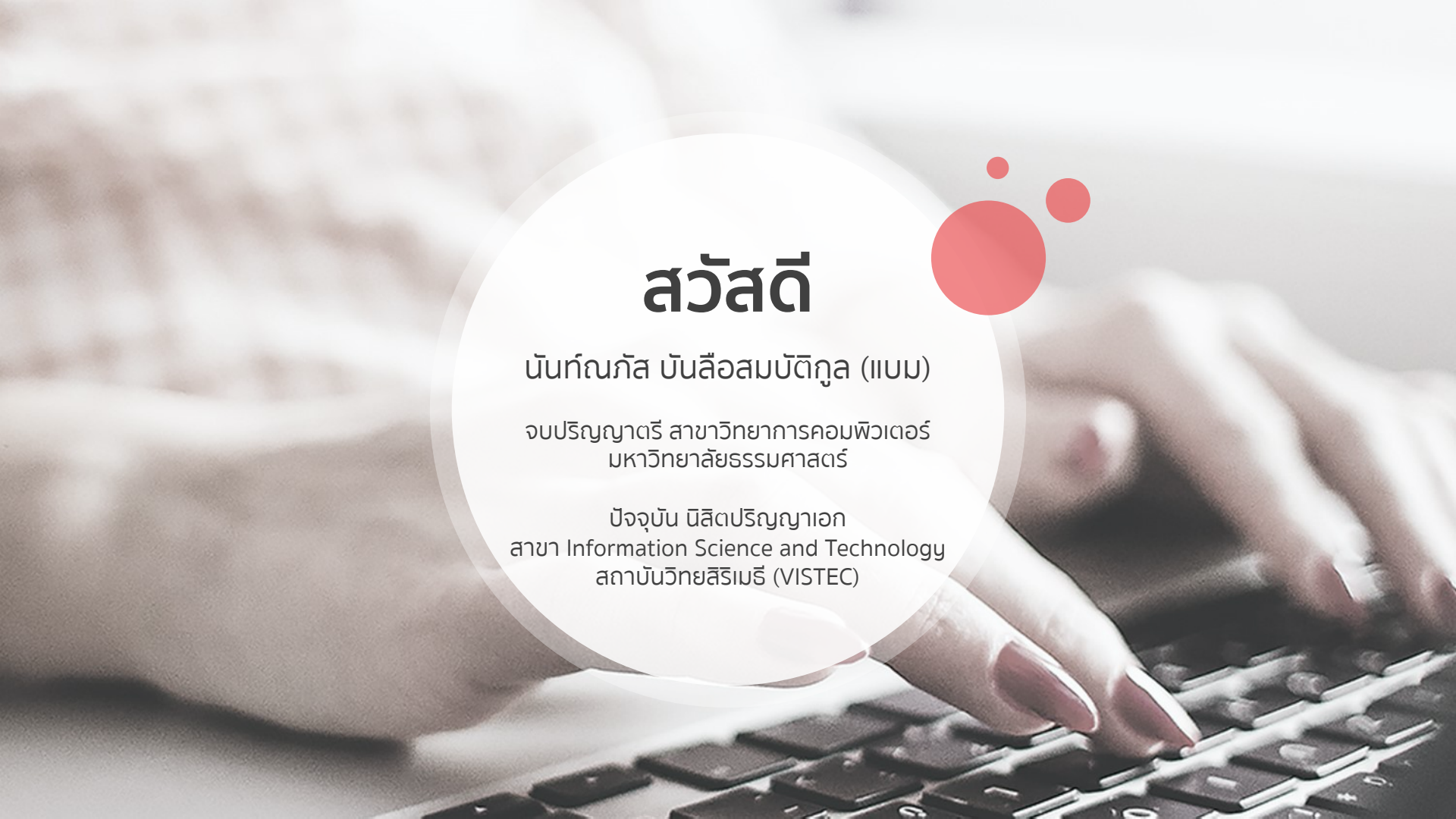
Graph Algorithm

สื่อการสอนสำหรับค่ายคอมพิวเตอร์โอลิมปิก สอวน. ค่าย 2/2562

โดย นางสาว นันท์ณภัส บันลือสมบัติกูล, สถาบันวิทยสิริเมธี (VISTEC)



สวัสดี

นันท์ณภัส บันลือสมบัติกูล (แบม)

จบปริญญาตรี สาขาวิทยาการคอมพิวเตอร์
มหาวิทยาลัยธรรมศาสตร์

ปัจจุบัน นิสิตปริญญาเอก
สาขา Information Science and Technology
สถาบันวิทยสิริเมธี (VISTEC)

เช็คชื่อกันเถอะ

https://forms.gle/VXHXUgvCCom8Rf1c9

Agenda

01 **Shortest path**
คืออะไร, Algorithm

02 **All pair shortest path**
คืออะไร, Algorithm

03 **Topological sort**
คืออะไร, Algorithm

04 **โจทย์**
ตัวอย่างข้อสอบกราฟ

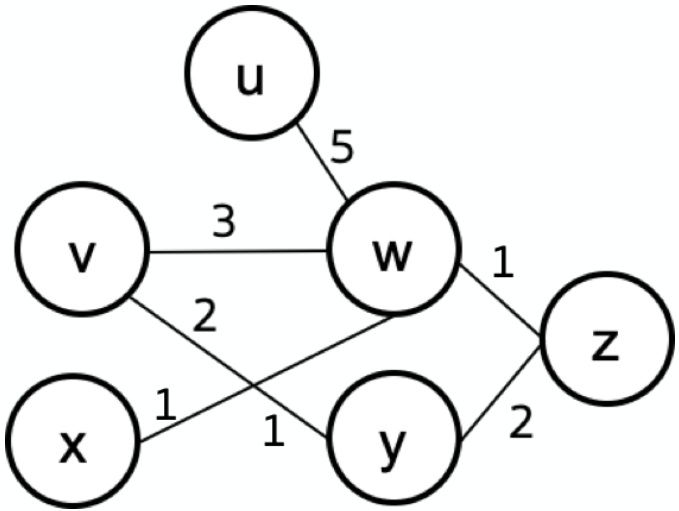
Shortest path

คืออะไร, Algorithm

Shortest path of Graph

คืออะไร, Algorithm

- **มีจุดเริ่มต้น (source) 1 จุด**
- ถามว่า **เส้นทางใด** ใกล้ที่สุด / ใช้ระยะทางน้อยสุด / weight ต่ำที่สุด ในการ**ไปถึง destination**

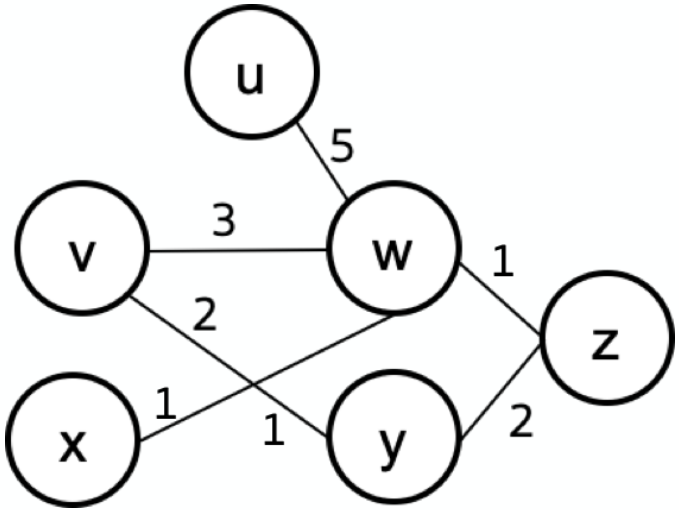


จงหาเส้นทางที่ใกล้ที่สุด จาก u ไป y

Shortest path of Graph

คืออะไร, Algorithm

- หรือ ถามว่า **ระยะทางที่ใกล้ที่สุด** จากจุดเริ่มต้น **ไปยังทุกจุด** เป็นเท่าใด **(สร้าง Shortest Path Tree)**

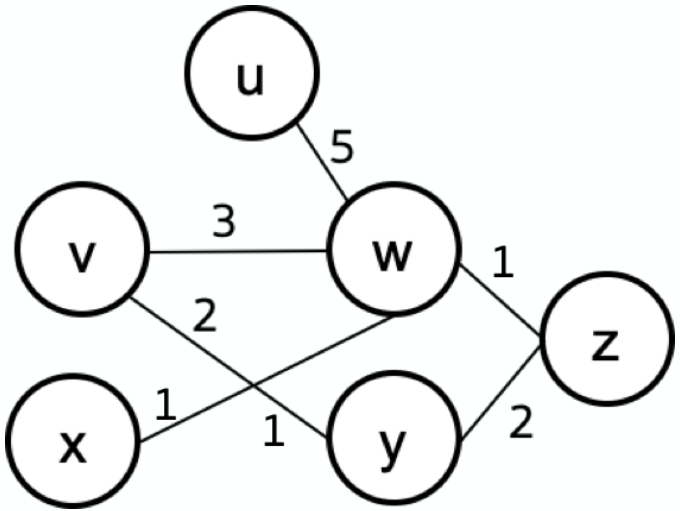


จงหาระยะทางที่สั้นที่สุด
ในการเดินไปให้ครบทุก node เมื่อเริ่มจาก u

Shortest path of Graph

คืออะไร, Algorithm

- หรือ ถามว่า **ระยะทางที่ใกล้ที่สุด** จากจุดเริ่มต้น **ไปยังทุกจุด** เป็นเท่าใด **(สร้าง Shortest Path Tree)**



จงหาระยะทางที่สั้นที่สุด
ในการเดินไปให้ครบทุก node เมื่อเริ่มจาก u

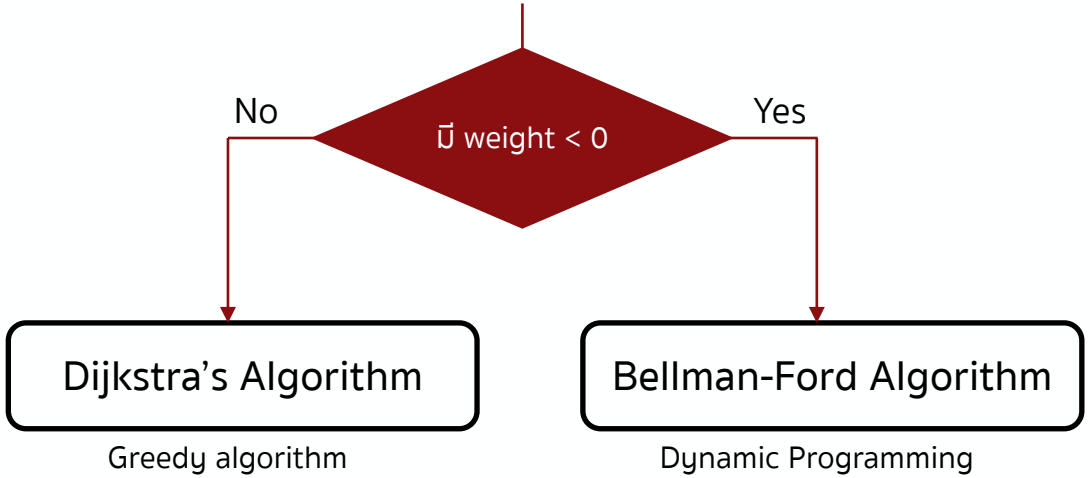
u -> v ::
u -> w ::
u -> x ::
u -> y ::
u -> z ::

(ผลลัพธ์จากการสร้าง **Shortest Path Tree**)

Shortest path of Graph

คืออะไร, Algorithm

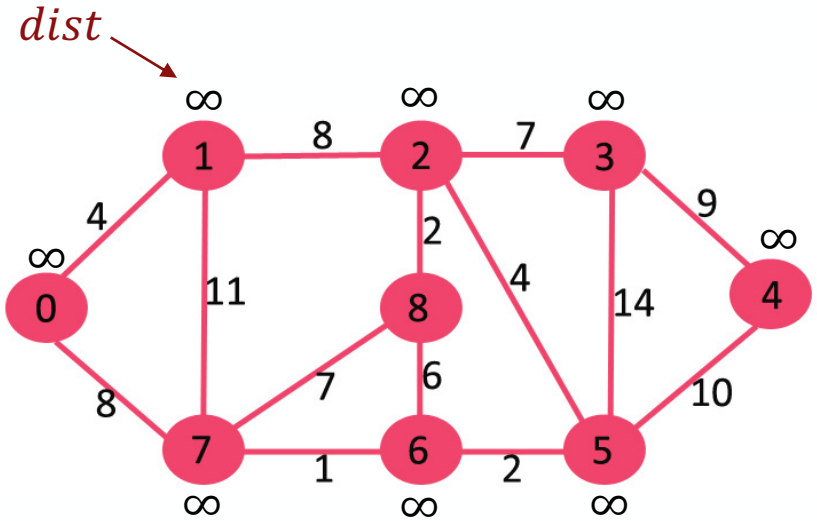
- **วิธีการหา Shortest Path / Shortest Path tree**



Shortest path of Graph

Dijkstra's Algorithm

- วิธีการสร้าง Shortest Path Tree **(Dijkstra's Algorithm)**
  - Distance ที่สั้นที่สุด จาก src ถึงแต่ละ node
  - ใช้ **BFS** with **Priority queue**



- Set up dist ทุกตัวเป็น INF
- เริ่มต้นจาก src -> s = src
- dist[s] = 0

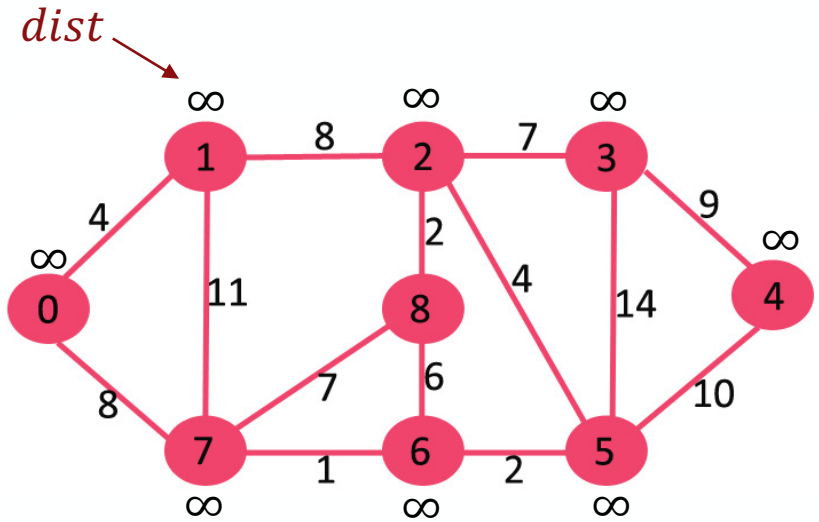
- ถ้ายัง visited ไม่ครบทุกตัว
  เลือก **s** ที่มี **dist** น้อยที่สุด
  mark ว่า visit **s** แล้ว
  For each **node** in Adj(s)
    ถ้า **dist[s] + w(s –> node)** < **dist[node]:**
      update **dist[node]**

- ค่า **dist** ที่ได้ของแต่ละ node คือ shortest distance from src

Shortest path of Graph

Dijkstra's Algorithm

- วิธีการสร้าง Shortest Path Tree **(Dijkstra's Algorithm)**
  - Distance ที่สั้นที่สุด จาก src ถึงแต่ละ node
  - ใช้ **BFS** with **Priority queue**

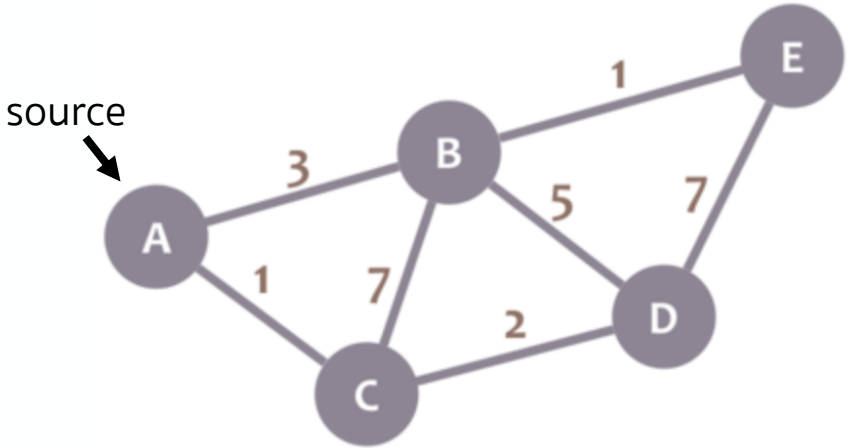


- **เป็น BFS ยังไง?**
  - ค่อยๆ อัพเดท dist ทีละขั้นตามแนวกว้าง
  - ใช้ Priority queue เพื่อเก็บ node ที่ต้องไปต่อแบบเรียงลำดับ dist จากน้อยไปมาก

Shortest path of Graph

Dijkstra's Algorithm

- แบบฝึกหัด
  - หา Shortest Path Tree จากกราฟด้านล่าง ด้วย **Dijkstra's Algorithm**

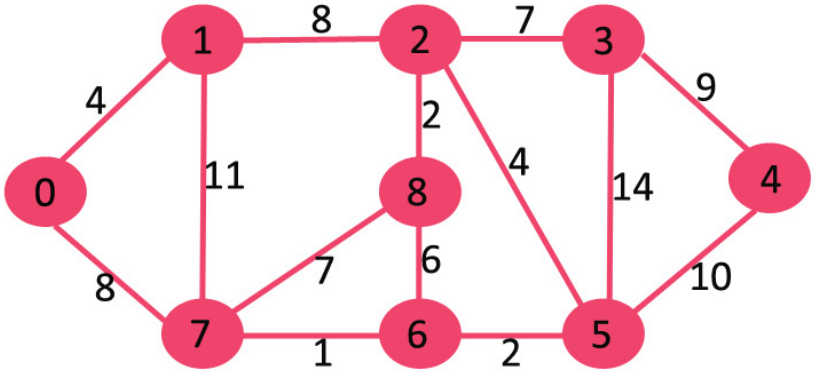


	A	B	C	D	E
dist =					

https://forms.gle/QYysNLWsLMkF9qKH6

Shortest path of Graph

Dijkstra's Algorithm



Let's code !!

[05-Dijkstra.cpp]

Shortest path of Graph

Dijkstra's Algorithm

- **แบบฝึกหัด**
  - จงหาเส้นทางที่ใกล้ที่สุดในการเดินทางจากสถานที่เริ่มต้น ไปยังปลายทางที่กำหนด โดยมีกราฟเป็น input

- **ข้อมูลนำเข้า (Input)**
  - จำนวนสถานที่ในแผนที่ (N)
  - จำนวนเส้นทางที่เชื่อมต่อแต่ละสถานที่ (E)
  - สถานที่ต้นทาง สถานที่ปลายทาง
  - ต้นทาง ปลายทาง ระยะทาง ของ เส้นทางที่ 1
  - ต้นทาง ปลายทาง ระยะทาง ของ เส้นทางที่ 2
  - ...
  - ต้นทาง ปลายทาง ระยะทาง ของ เส้นทางที่ E

- **ผลลัพธ์ (Output)**
  - v_1 v_2 v_3 ... v_k

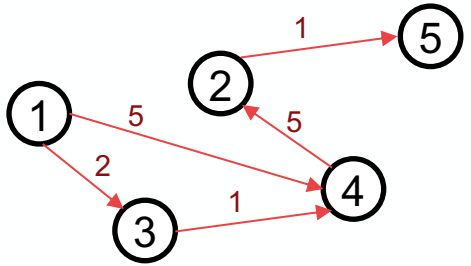
  (vi แทนหมายเลขสถานที่ โดยคั่นด้วยเว้นวรรค)
  - ระยะทางที่ใช้

- **ตัวอย่าง input**

```
4
4
1 2
1 4 5
1 3 2
3 4 6
4 2 5
```

- **ตัวอย่าง output**

```
1 3 4 2
8
```



https://forms.gle/nGaiSyxq4SxwzjzMA

Shortest path of Graph

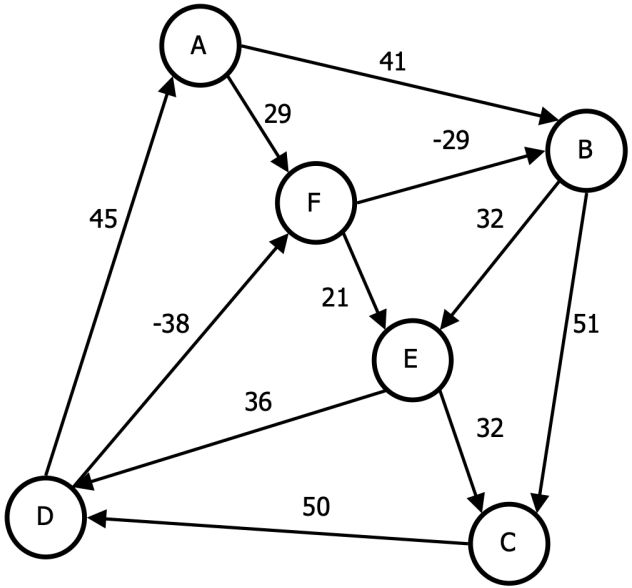
Bellman-Ford Algorithm

- ในกรณีที่กราฟมี weight เป็นค่าติดลบ ไม่สามารถรับประกันได้ว่า Dijkstra's จะถูก
- ให้ใช้ **Bellman–Ford Algorithm**
- (ต้องยอม ถึงแม้จะใช้เวลาเยอะกว่า)

Shortest path of Graph

Bellman-Ford Algorithm

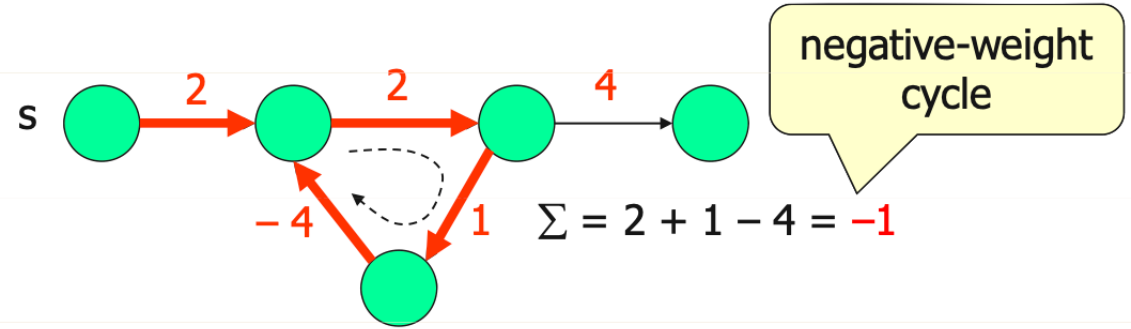
- **ตัวอย่างในชีวิตประจำวัน**
  - แต่ละ **Node** แทน**สินค้า**ทั้งหมดในรายการ
  - **Edge** (u, v) แทน**ราคา**เมื่อซื้อสินค้า u แล้วซื้อสินค้า v
    - เช่น เมื่อซื้อ A -> ซื้อ B จะเสียค่าใช้จ่าย 41 บาท
    - เมื่อซื้อ F -> ซื้อ B จะได้รับส่วนลด 29 บาท
  - การแก้ปัญหาว่า “ควรซื้อสินค้าแต่ละชนิดอย่างไร เพื่อให้ได้ราคาถูกที่สุด เมื่อซื้อสินค้า A เป็นชิ้นแรก”



Shortest path of Graph

Bellman-Ford Algorithm

- **Bellman–Ford Algorithm**
  - **Input:** source vertex
  - **Output:**
    - If no negative cycle detected -> returns **Shortest Path Tree** (Shortest path to all vertices)
    - If negative cycle detected -> the shortest distance will not be calculated, the **cycle** is reported.



คือ **cycle ที่ weight รวมมีค่า < 0**,
ซึ่งทำให้หาจุดสิ้นสุดไม่ได้ เพราะยิ่งเดินในวง ค่ายิ่งลดลงเรื่อย ๆ

Shortest path of Graph

Bellman-Ford Algorithm

- ขั้นตอนของ **Bellman–Ford Algorithm**

กำหนดให้ V = set of vertices, src = Vertex เริ่มต้น

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance โดย ทำด้านล่างนี้ทั้งหมด |V| - 1 ครั้ง
   - แต่ละ **edge (u, v)**

     if dist[v] > dist[u] + weight of edge(u,v):

     dist[v] = dist[u] + weight of edge(u,v)

   ** สรุปคือ วนทุก edge -> แต่ละ edge เก็บค่าน้อยที่สุดไว้ที่ node ปลายทาง และวนต่อจนครบ -> ทำไป |V| -1 รอบ

3. Report if there's negative cycle
   - แต่ละ **edge (u, v)**

     if dist[v] > dist[u] + weight of edge(u,v):

     Report "Graph contains negative weight cycle"

   ** ข้อ 2. เป็นการการันตีแล้วว่านั้นคือเส้นทางที่สั้นที่สุด ถ้าตรงนี้ยังเจออีก แสดงว่าเจอ negative weight cycle

4. If there's **no** negative weight cycle:

   dist[v] = shortest path จาก src มายัง v เมื่อ v ∈ V

Shortest path of Graph

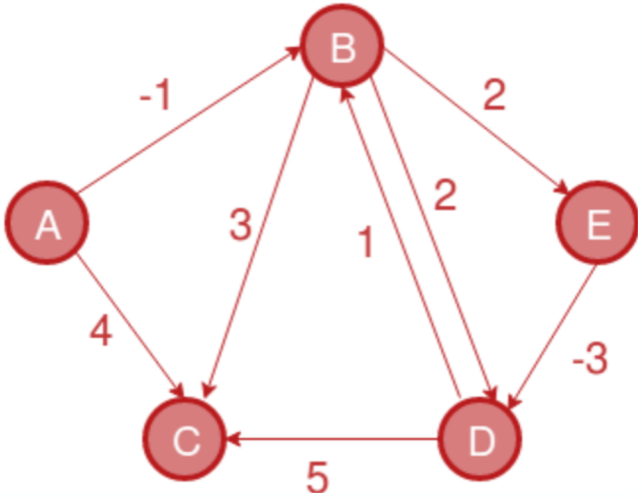
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))

3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"

4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = { }

	A	B	C	D	E
dist =					

Shortest path of Graph

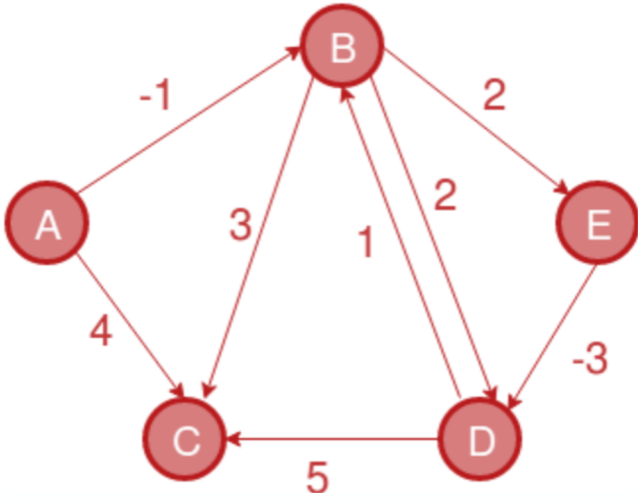
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))

3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"

4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

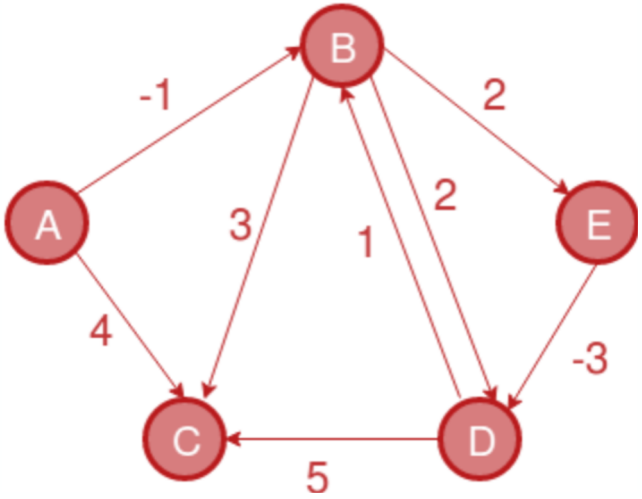
	A	B	C	D	E
dist =					

Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))
3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**INF**	**INF**	**INF**	**INF**

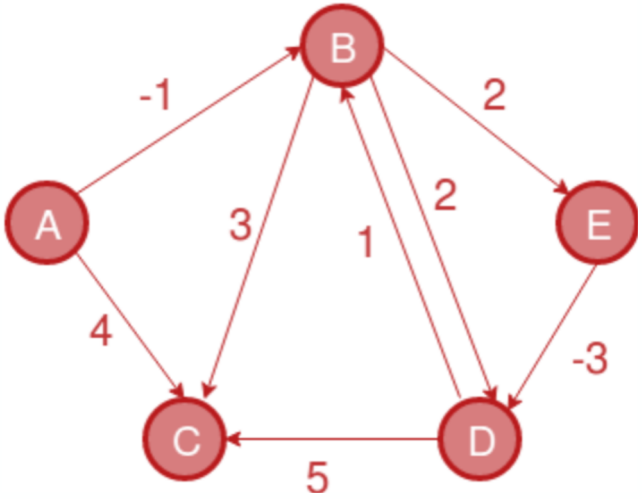
Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))

i = 1



3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree

E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

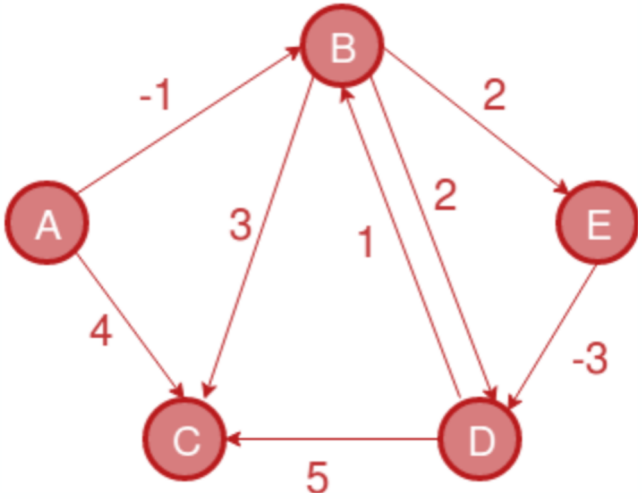
	A	B	C	D	E
dist =	**0**	**INF**	**INF**	**INF**	**INF**

Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   i = 1
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))
3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**INF**	**INF**	**INF**	**INF**

Shortest path of Graph

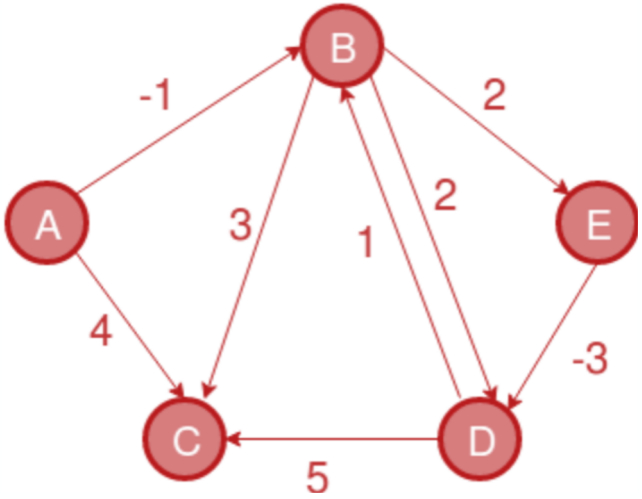
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
      For each **edge (u, v)**:
         if dist[v] > dist[u] + w(edge(u,v)):
            dist[v] = dist[u] + w(edge(u,v))

i = 1

3. Report if there's negative cycle
   For each **edge (u, v)**:
      if dist[v] > dist[u] + weight of edge(u,v):
         "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
      dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

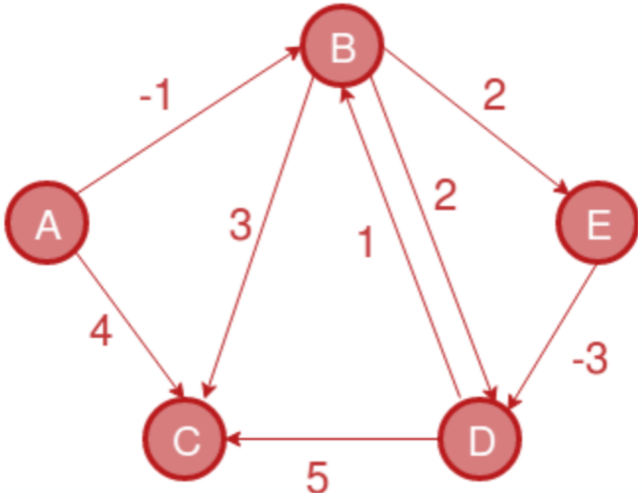
	A	B	C	D	E
dist =	0	-1	INF	INF	INF

Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   i = 1
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))



3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree

E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**INF**	**INF**	**INF**

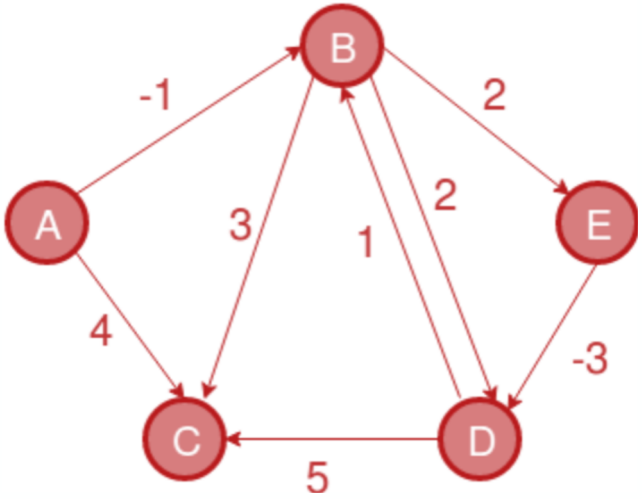
Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))

i = 1



3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree

E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

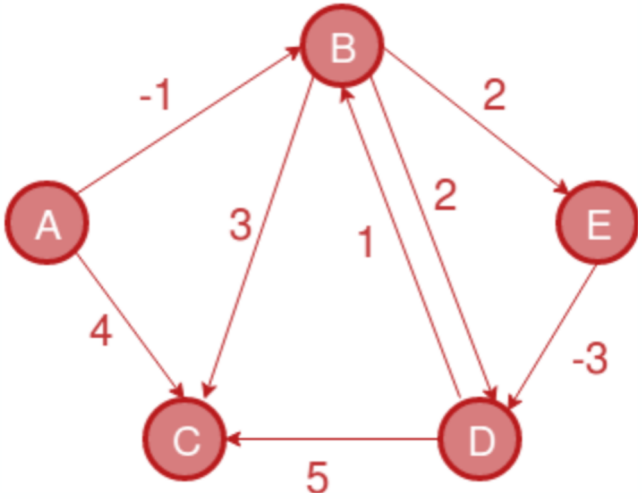
	A	B	C	D	E
dist =	**0**	**-1**	**4**	**INF**	**INF**

Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   i = 1
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))



3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree

E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

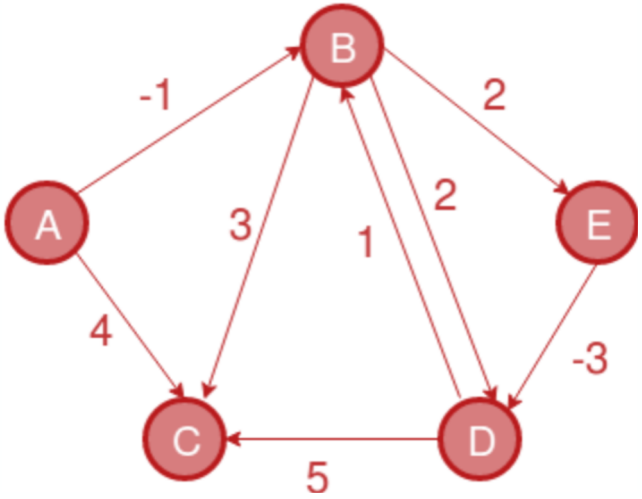
	A	B	C	D	E
dist =	**0**	**-1**	**4**	**INF**	**INF**

Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   i = 1
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))
3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**INF**	**INF**

Shortest path of Graph

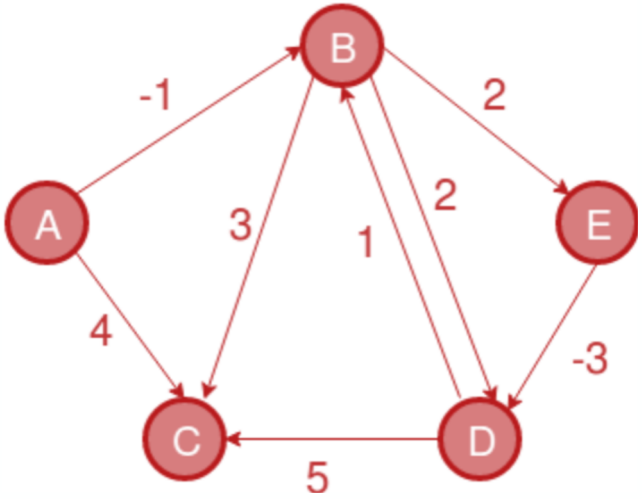
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   i = 1
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))

3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"

4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**INF**	**INF**

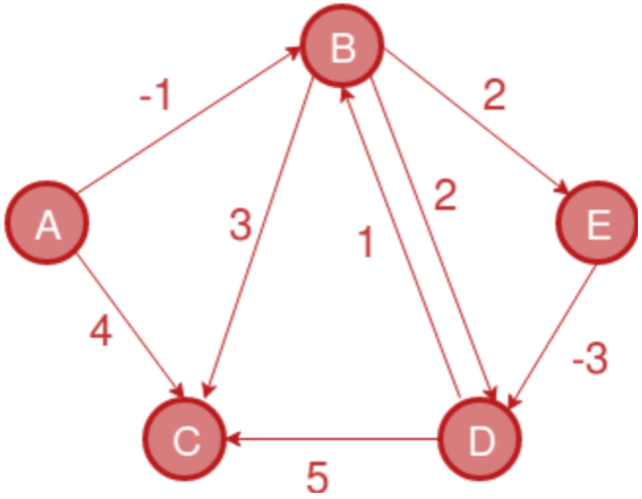
Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
      For each **edge (u, v)**:
         if dist[v] > dist[u] + w(edge(u,v)):
            dist[v] = dist[u] + w(edge(u,v))

i = 1



3. Report if there's negative cycle
   For each **edge (u, v)**:
      if dist[v] > dist[u] + weight of edge(u,v):
         "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
      dist[v] = shortest path tree

E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**1**	**INF**

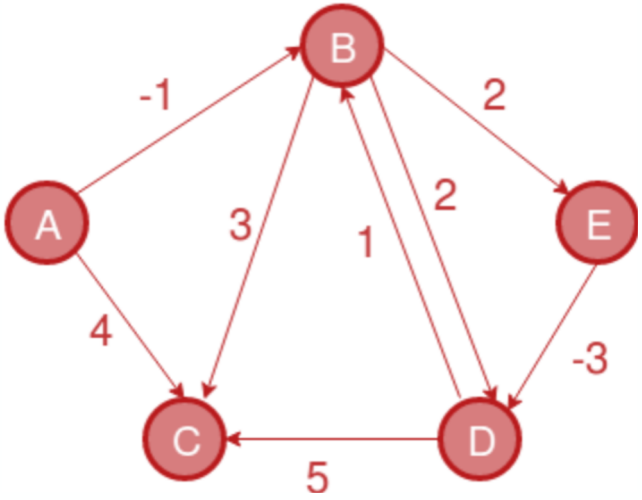
Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))

i = 1



3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree

E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**1**	**INF**

Shortest path of Graph

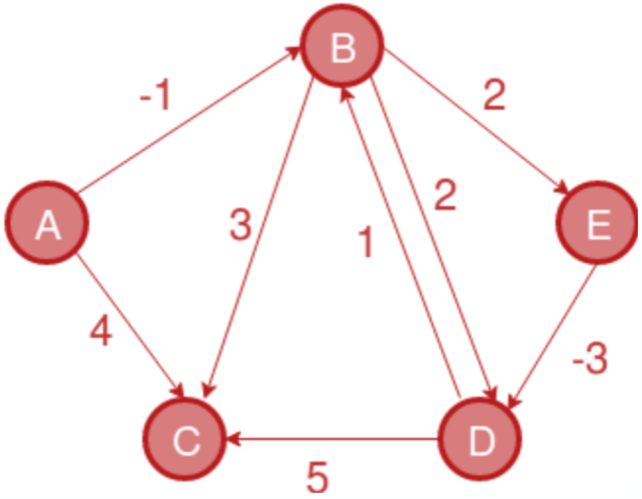
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
      For each **edge (u, v)**:
         if dist[v] > dist[u] + w(edge(u,v)):
            dist[v] = dist[u] + w(edge(u,v))

i = 1

3. Report if there's negative cycle
   For each **edge (u, v)**:
      if dist[v] > dist[u] + weight of edge(u,v):
         "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
      dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**1**	**1**

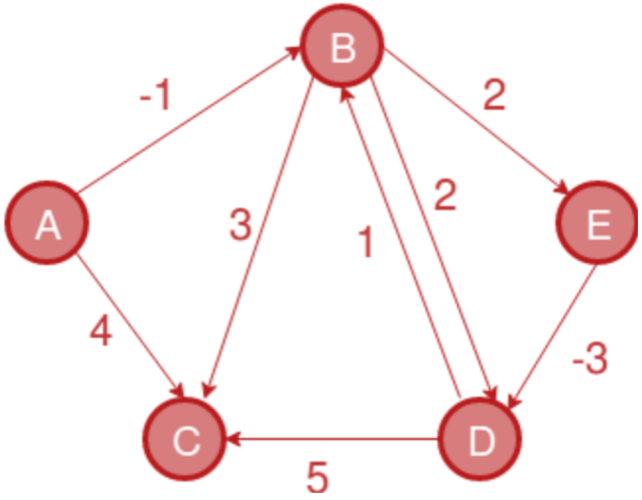
Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
      For each **edge (u, v)**:
         if dist[v] > dist[u] + w(edge(u,v)):
            dist[v] = dist[u] + w(edge(u,v))

i = 1



3. Report if there's negative cycle
   For each **edge (u, v)**:
      if dist[v] > dist[u] + weight of edge(u,v):
         "**Graph contains negative weight cycle**"

E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree

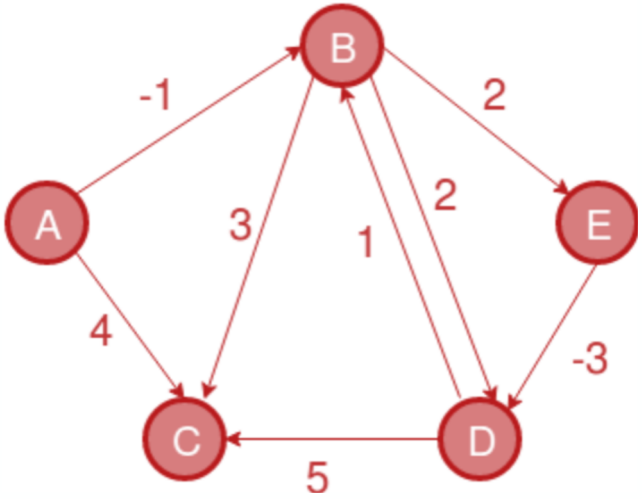
	A	B	C	D	E
dist =	**0**	**-1**	**2**	**1**	**1**

Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   i = 1
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))
3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C) (D, B) (E, D)}

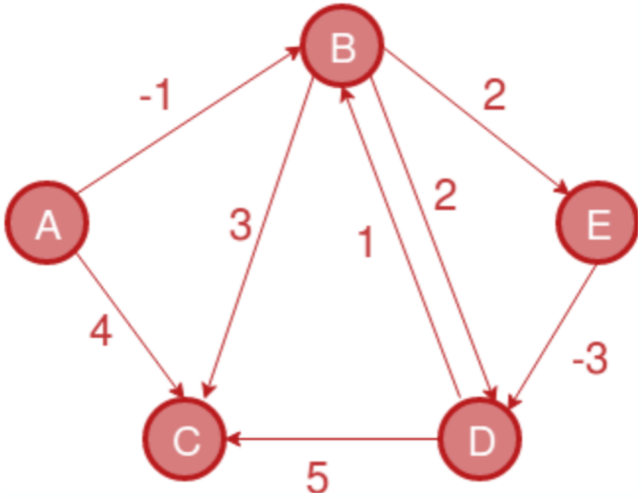
	A	B	C	D	E
dist =	**0**	**-1**	**2**	**1**	**1**

Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   i = 1
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))



3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree

E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

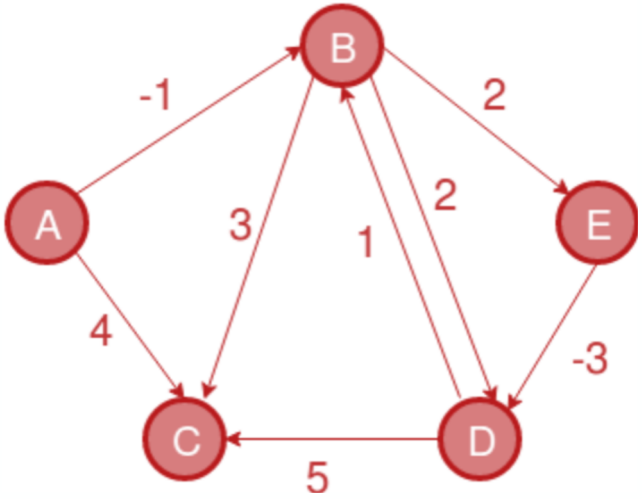
	A	B	C	D	E
dist =	**0**	**-1**	**2**	**1**	**1**

Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   i = 1
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))
3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**-2**	**1**

Shortest path of Graph

Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance

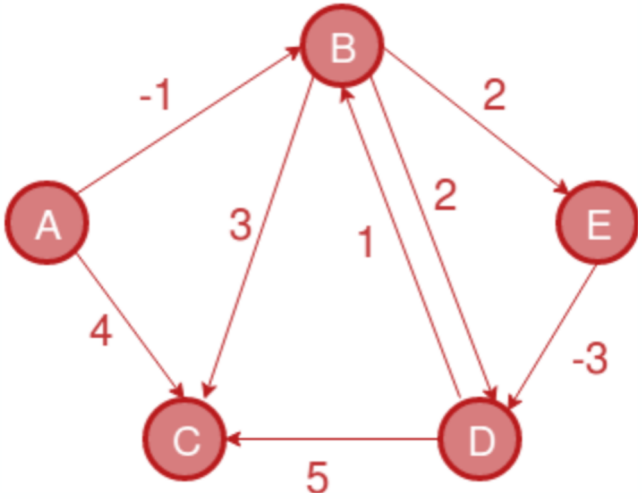
```
For i = 0 to |V| - 1:
    For each edge (u, v):
        if dist[v] > dist[u] + w(edge(u,v)):
            dist[v] = dist[u] + w(edge(u,v))
```

i = 2

3. Report if there's negative cycle

```
For each edge (u, v):
    if dist[v] > dist[u] + weight of edge(u,v):
        "Graph contains negative weight cycle"
```

4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**-2**	**1**

Shortest path of Graph

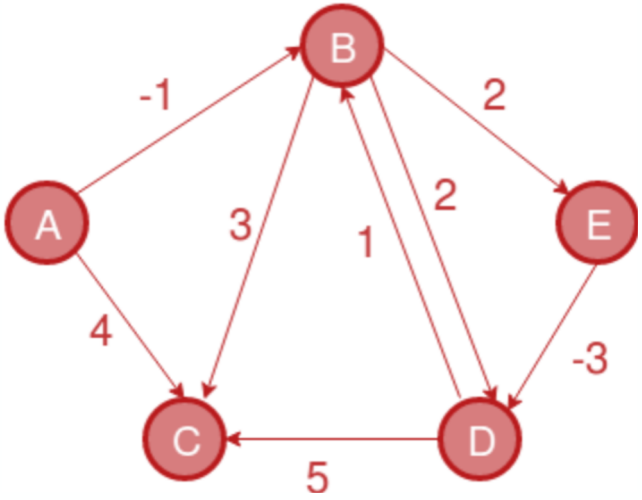
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
      For each **edge (u, v)**:
         if dist[v] > dist[u] + w(edge(u,v)):
            dist[v] = dist[u] + w(edge(u,v))

i = 2

3. Report if there's negative cycle
   For each **edge (u, v)**:
      if dist[v] > dist[u] + weight of edge(u,v):
         "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
      dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**-2**	**1**

Shortest path of Graph

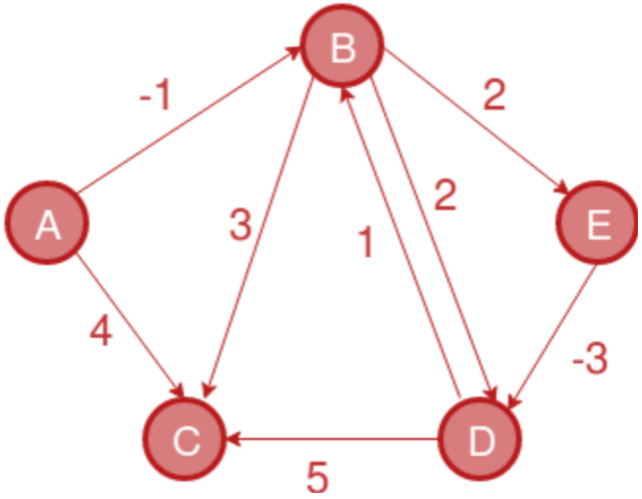
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
      For each **edge (u, v)**:
         if dist[v] > dist[u] + w(edge(u,v)):
            dist[v] = dist[u] + w(edge(u,v))

i = 3

3. Report if there's negative cycle
   For each **edge (u, v)**:
      if dist[v] > dist[u] + weight of edge(u,v):
         "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
      dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**-2**	**1**

Shortest path of Graph

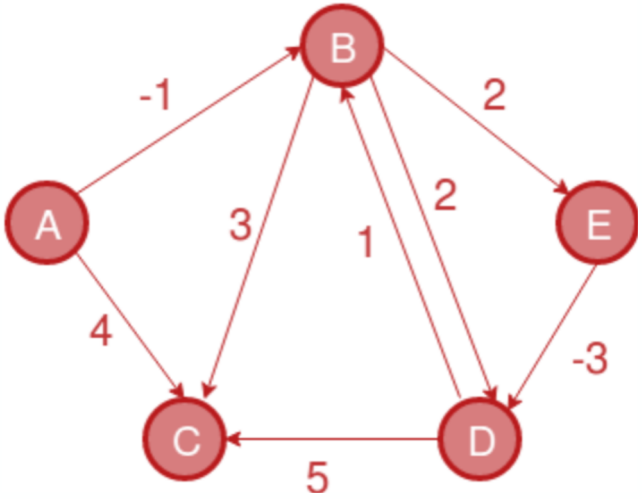
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
      For each **edge (u, v)**:
         if dist[v] > dist[u] + w(edge(u,v)):
            dist[v] = dist[u] + w(edge(u,v))

i = 4

3. Report if there's negative cycle
   For each **edge (u, v)**:
      if dist[v] > dist[u] + weight of edge(u,v):
         "**Graph contains negative weight cycle**"
4. If there's **no** negative weight cycle:
      dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**-2**	**1**

Shortest path of Graph

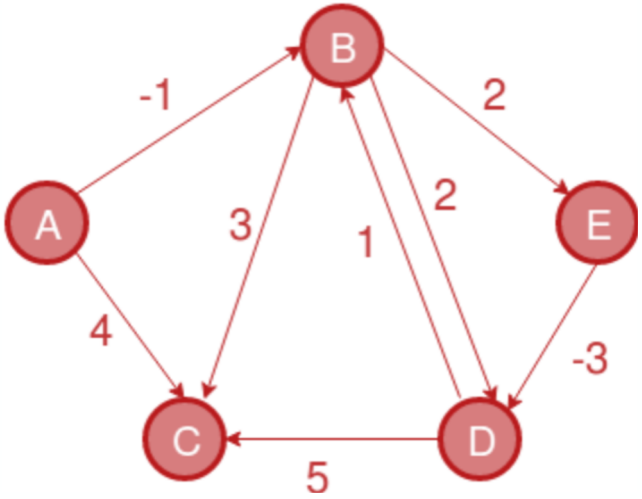
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))

3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"

4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree



E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**-2**	**1**

Shortest path of Graph

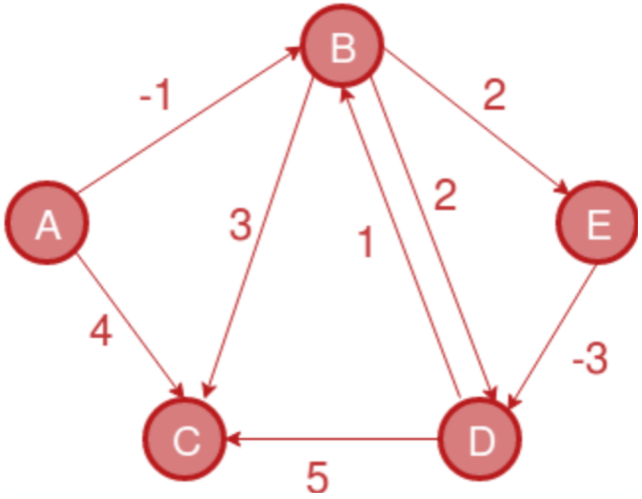
Bellman-Ford Algorithm

- **ตัวอย่าง** (src = A)

1. Initial array **dist** size |**V**| with **INF**, except for **dist[src] = 0**
2. Calculate shortest distance
   For i = 0 to |V| - 1:
   For each **edge (u, v)**:
   if dist[v] > dist[u] + w(edge(u,v)):
   dist[v] = dist[u] + w(edge(u,v))

3. Report if there's negative cycle
   For each **edge (u, v)**:
   if dist[v] > dist[u] + weight of edge(u,v):
   "**Graph contains negative weight cycle**"

4. If there's **no** negative weight cycle:
   dist[v] = shortest path tree

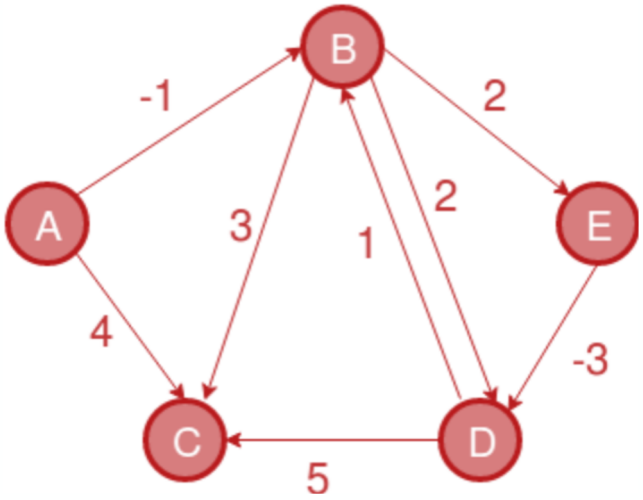


E = {(A, B), (A, C), (B, C), (B, D), (B, E), (D, C), (D, B), (E, D)}

	A	B	C	D	E
dist =	**0**	**-1**	**2**	**-2**	**1**

Shortest path of Graph

Bellman-Ford Algorithm



Let’s code !!

[06-Bellman-Ford.cpp]


To be
continued..