Chương 4: BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT

- Hai giải thuật duyệt đồ thị cơ bản: Depth-First Search (DFS) , Breadth-First Search (BFS).

- Về mặt ý nghĩa, cả 2 thuật toán đều thực hiện trên đồ thị không có trọng số.

- Thuật toán DFS cho ta tìm đường đi đến đỉnh ta muốn (có thể chưa ngắn nhất).
- Còn thuật toán BFS cũng tìm đường đi đến đỉnh ta muốn nhưng là đường ngắn nhất (có thể).
- Trong đồ thị có trọng số, vấn đề sẽ khó hơn, nó nảy sinh ra 1 bài toán, đó là bài toán tìm đường đi ngắn nhất giữa 2 đỉnh
- Để gải quyết khó khăn này, ta có một số các thuật toán khác:

Moore – Dijkstra; Bellman-Ford; Floyd-Warshall.....

Nội dung

Để giải quyết bài toán này, đến giờ có nhiều thuật toán và các biến thể: Moore – Dijkstra; Bellman-Ford; Floyd-Warshall.....

Thuật toán Dijkstra cho phép tìm đường đi ngắn nhất từ một đỉnh s đến các đỉnh còn lại của đồ thị và chiều dài (trọng số) tương ứng.

Phương pháp của thuật toán là xác định tuần tự đỉnh có chiều dài đến s theo thứ tự tăng dần.

Thuật toán được xây dựng trên cơ sở gán cho mỗi đỉnh các nhãn tạm thời.

Nhãn tạm thời của các đỉnh cho biết cận trên của chiều dài đường đi ngắn nhất từ s đến đỉnh đó.

Nhãn của các đỉnh sẽ biến đổi trong các bước lặp, mà ở mỗi bước lặp sẽ có một nhãn tạm thời trở thành chính thức.

Nếu nhãn của một đỉnh nào đó trở thành chính thức thì đó cũng chính là chiều dài ngắn nhất của đường đi từ s đến đỉnh đó.

➢ Nội dung

- Phát biểu bài toán tìm đường đi ngắn nhất
- Thuật toán Dijkstra
- Thuật toán Bellman-Ford
- Thuật toán Floyd

➢ Phát biểu bài toán

- Bài toán dạng tổng quát:
  - Tìm đường đi ngắn nhất từ một đỉnh xuất phát $s \in X$ (đỉnh nguồn) đến đỉnh cuối $t \in X$ (đỉnh đích).
  - Đường đi như vậy được gọi là đường đi ngắn nhất từ ***s*** đến ***t***.
  - Độ dài của đường đi ***d(s,t)*** được gọi là khoảng cách ngắn nhất từ ***s*** đến ***t*** (trong trường hợp tổng quát ***d(s,t)*** có thể âm).
  - Nếu như không tồn tại đường đi từ ***s*** đến ***t*** thì độ dài đường đi $d(s,t)=\infty$.

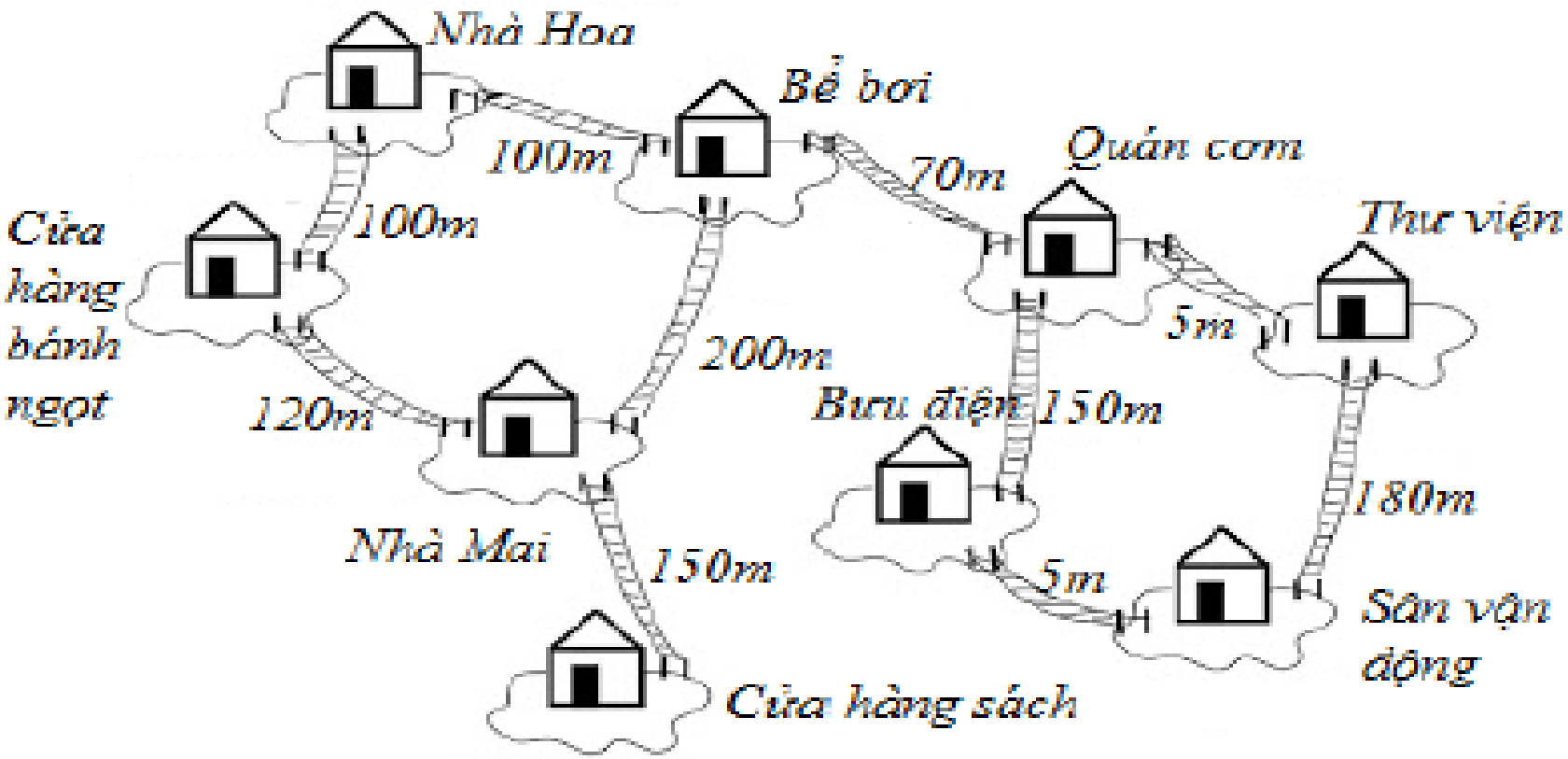
➢ Một số thể hiện cụ thể của bài toán

- **Trường hợp 1.** Nếu ***s*** cố định và ***t*** thay đổi:
  - Tìm đường đi ngắn nhất từ ***s*** đến tất cả các **đỉnh còn lại** trên đồ thị.
  - Với đồ thị có trọng số không âm, bài toán luôn có lời giải bằng thuật toán **Dijkstra**.
  - Với đồ thị có trọng số âm nhưng không tồn tại chu trình âm, bài toán có lời giải bằng thuật toán **Bellman-Ford**.
  - Trường hợp đồ thị có **chu trình âm**, bài toán không có lời giải.
- **Trường hợp 2.** Nếu ***s*** thay đổi và ***t*** cũng thay đổi:
  - Tìm đường đi ngắn nhất giữa tất cả các **cặp đỉnh** của đồ thị.
  - Bài toán luôn có lời giải trên đồ thị không có chu trình âm.
  - Với đồ thị có trọng số không âm, bài toán được giải quyết bằng cách thực hiện lặp lại ***n*** lần thuật toán **Dijkstra**.
  - Với đồ thị không có chu trình âm, bài toán có thể giải quyết bằng thuật toán **Floyd**.

➢ Một số thể hiện cụ thể của bài toán

N- số đỉnh M: Số cạnh	**Bellman-Ford**	**Dijkstra (cơ bản)**	**Dijkstra (trên đồ thị thưa)**	**Floyd-Warshall**
Bài toán giải quyết	Đường đi ngắn nhất **một nguồn**	Đường đi ngắn nhất **một nguồn**	Đường đi ngắn nhất **một nguồn**	Đường đi ngắn nhất **mọi cặp đỉnh**
Độ phức tạp	O(N∗M)	$O(N^2+M)$	O(MlogN)	$O(N^3)$
Sử dụng được cho trọng số âm	Có	Không	Không	Có
Tìm được chu trình âm	Có	Không	Không	Không

➢ Phát biểu bài toán tìm đường đi ngắn nhất



Ví dụ sơ đồ khu dân cư

➢ Phát biểu bài toán

Bài toán: Cho đồ thị có trọng số $G = (X, U)$, mỗi cạnh (cung) $u \in U$ có trọng số $l(u) \geq 0$. Hãy tìm đường đi ngắn nhất từ đỉnh x cho trước đến mọi đỉnh khác của G.

Lưu ý: Đường đi từ đỉnh x đến y gọi là đường đi ngắn nhất nếu tổng trọng số của các cạnh (cung) trên đường đi này là nhỏ nhất.

Nếu trọng số của các cạnh (cung) trên đồ thị đều bằng nhau thì ta coi đồ thị không có trọng số. Lúc đó, đường đi từ x đến y là ngắn nhất nếu tổng số các cạnh (cung) trên đường đi này là ít nhất.

Thuật toán Floyd

➢ Thuật toán Floyd: Mô tả thuật toán

- Mục đích
  - Sử dụng để tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị.
  - Áp dụng cho đồ thị có hướng và không có chu trình âm (có thể có cạnh âm).
- Tư tưởng
  - Thực hiện quá trình lặp
    - Xét từng đỉnh, với tất cả các đường đi (giữa 2 đỉnh bất kỳ)
    - Nếu đường đi hiện tại lớn hơn đường đi qua đỉnh đang xét, thay lại thành đường đi qua đỉnh này.

❖ Thuật toánFloyd: Mô tả thuật toán

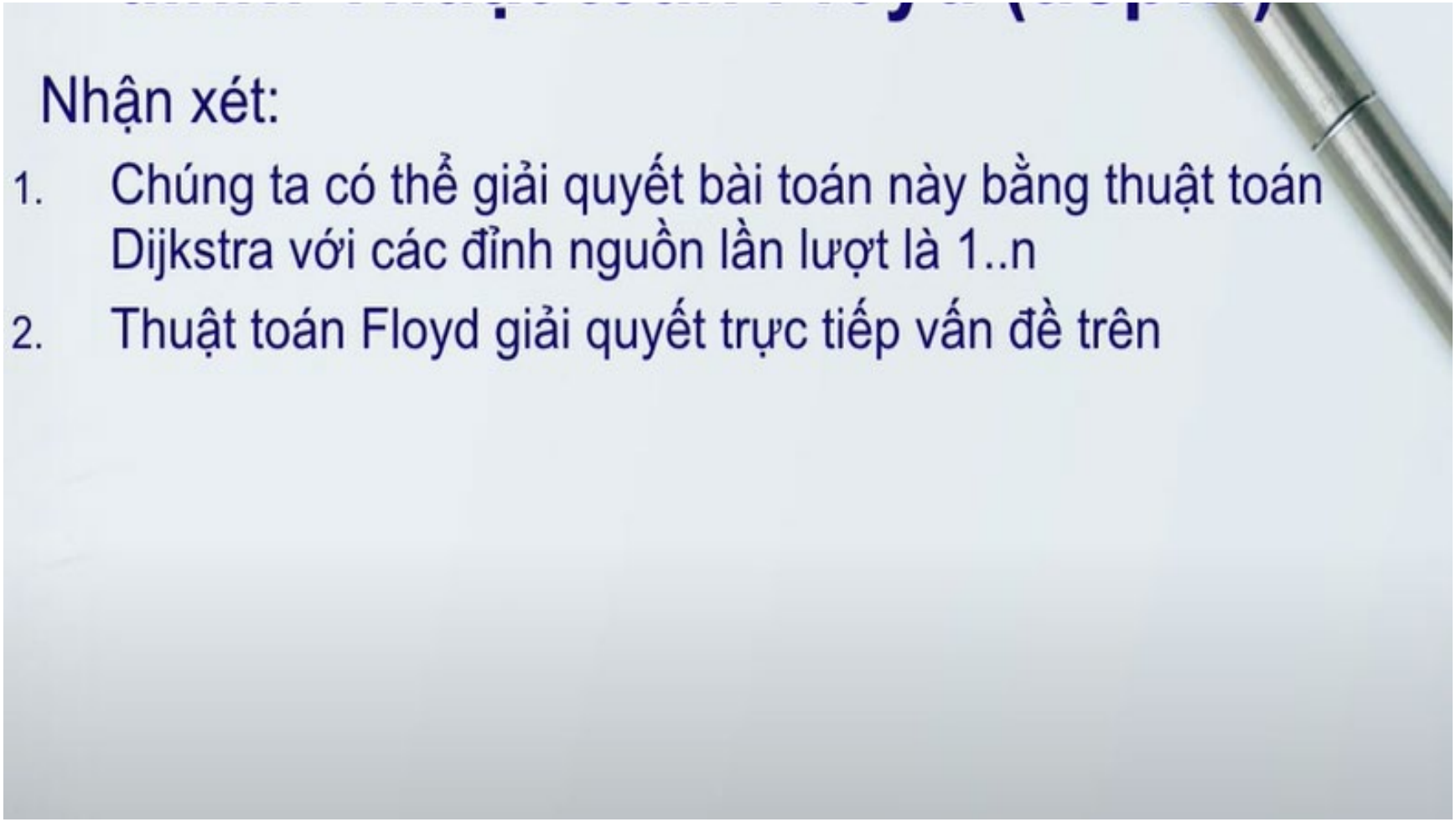
Bài toán:

1. Cho đồ thị G =(X,U) đơn liên thông có trọng số
2. Tìm đường đi ngắn nhất giữa mọi cặp đỉnh x,y trong đồ thị
3. Các đỉnh x được đánh số 1 tới n
4. Đồ thị được biểu diễn bởi ma trận kề C[1..n,1..n], trong đó C[i,j] là độ dài cung (i,j) nếu không có cung (i,j) thì C[i,j] = ∞

Dikstra: Nguồn đơn tới đa đích

Floyd: Đa Nguồn tới đa đích

❖ Thuật toán Floyd: Mô tả thuật toán



➢ Thuật toán Floyd: Mô tả thuật toán

Nhận xét:

1. Nếu đỉnh k nằm trên đường đi ngắn nhất từ đỉnh i tới đỉnh j thì các đường đi từ i tới k và từ k tới j phải là đường đi ngắn nhất từ i tới k và từ k tới j tương ứng

❖ Thuật toán Floyd: Mô tả thuật toán

Giải thuật:

1. Sử dụng ma trận A để lưu độ dài đường đi ngắn nhất giữa mọi cặp đỉnh.
2. Ban đầu ta đặt A[i,j] = C[i,j], có nghĩa là ban đầu A chứa độ dài đường đi trực tiếp các đỉnh x đến y thuộc đồ thị mà không đi qua đỉnh nào cả.

❖ Thuật toán Floyd: Mô tả thuật toán

3. Sau đó ta thực hiện n lần lặp. Sau lần lặp thứ k, ma trận A sẽ chứa độ dài các đường đi ngắn nhất chỉ đi qua các đỉnh thuộc {1,2,…,k}
4. Do đó sau n bước lặp ta nhận được ma trận A chứa độ dài các đường đi ngắn nhất

➢ Thuật toán Floyd: Mô tả thuật toán

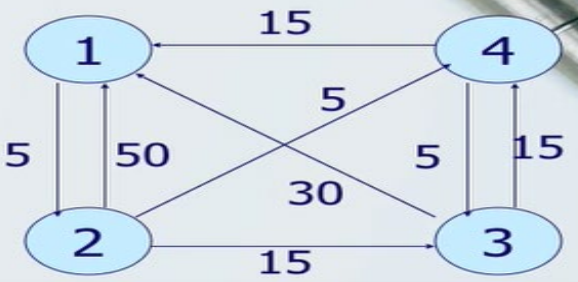
5. Kí hiệu A_k là ma trận A sau lần lặp thứ k, khi đó $A_k[i,j]$ được tính theo công thức sau:

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

❖ Thuật toán Floyd: Mô tả thuật toán

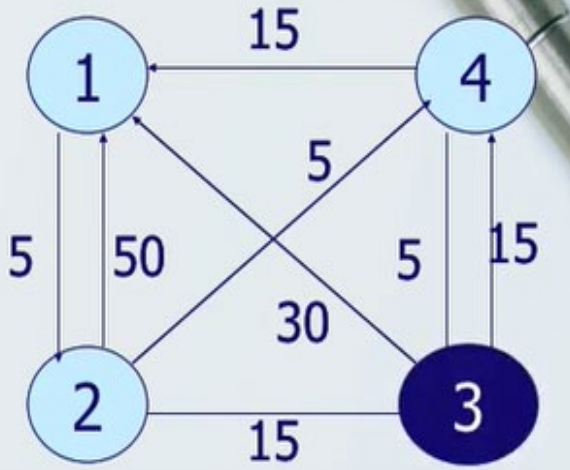
Xét ví dụ:

$$C = \begin{vmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{vmatrix}$$



Xét đỉnh 1: Gán $A_1 = C$, Cập nhật trọng số các cặp đỉnh đi qua đỉnh 1: đỉnh (4,2) = 15+5 = 20; đỉnh (3,2) = 30+5 = 35

$$A_1 = \begin{vmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \overset{35}{\infty} & 0 & 15 \\ 15 & \overset{20}{\infty} & 5 & 0 \end{vmatrix}$$



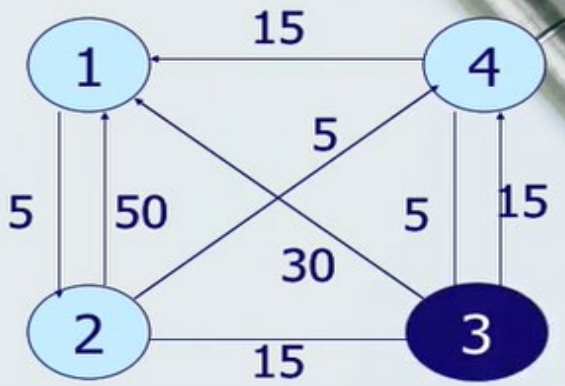
Cập nhật trọng số các cặp đỉnh đi qua đỉnh 1.

❖ Thuật toán Floyd: Mô tả thuật toán

Xét đỉnh 2: Cập nhật trọng số các cặp đỉnh đi qua đỉnh 2.

Sau khi gán $A_2 = A_1$

$$A_1 = \begin{vmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{vmatrix} = A_2$$



Cập nhật trọng số các cặp đỉnh đi qua đỉnh 2.

$$A_2 = \begin{vmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{vmatrix}$$

Cập nhật trọng số các cặp đỉnh đi qua đỉnh 2

$= A_2$

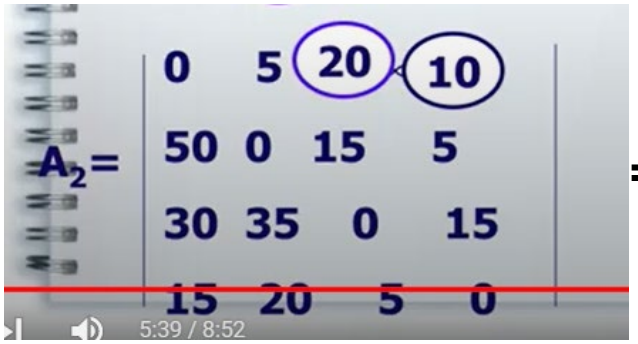
$$A_2 = \begin{vmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{vmatrix}$$

5:39 / 8:52

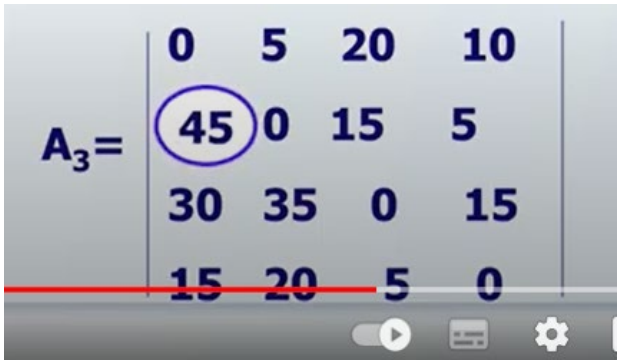
❖ Thuật toán Floyd: Mô tả thuật toán

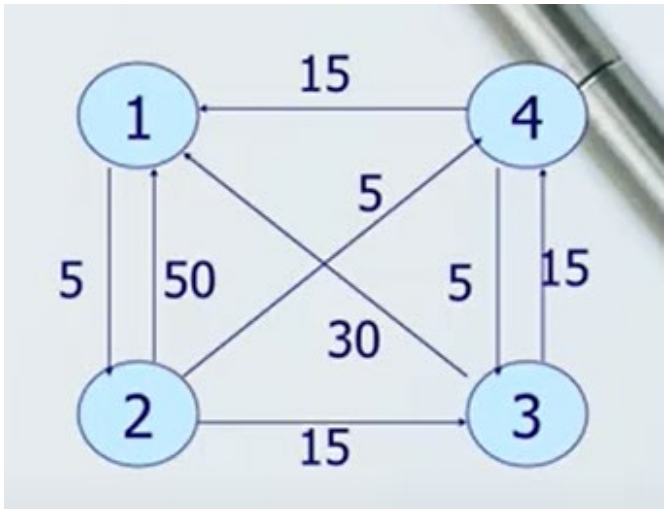
Xét đỉnh 3: Cập nhật trọng số các cặp đỉnh đi qua đỉnh 3.

Sau khi gán $A_3 = A_2$



$=A_3$ Cập nhật trọng số các cặp đỉnh đi qua đỉnh 3





❖ Thuật toán Floyd: Mô tả thuật toán

Xét đỉnh 4: Cập nhật trọng số các cặp đỉnh đi qua đỉnh 4.

Sau khi gán $A_4 = A_3$

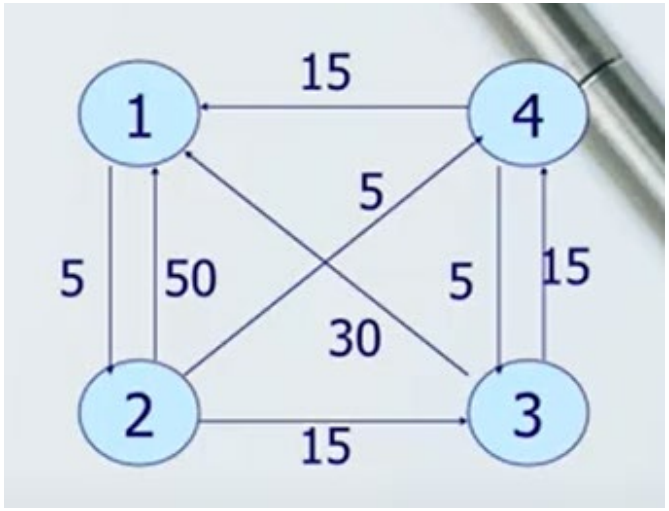
$A_3 =$

0	5	20	10
45	0	15	5
30	35	0	15
15	20	5	0

$=A_4$ Cập nhật trọng số các cặp đỉnh đi qua đỉnh 4

$A_4 =$

0	5	20	10
20	0	10	5
30	35	0	15
15	20	5	0



Kết Luận: Ma trận đỉnh A4 chỉ ra khoảng cách ngắn nhất giữa các cặp đỉnh.

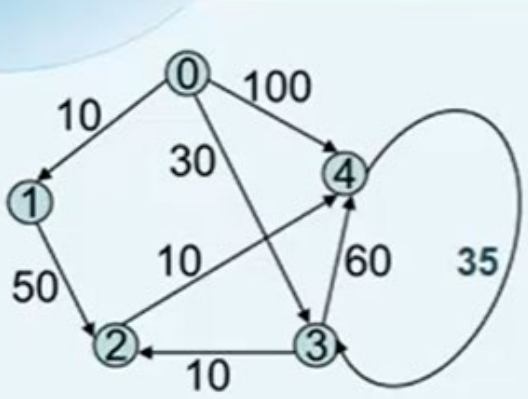
❖ Thuật toán Floyd: Mô tả thuật toán

Floyd

1. For i =1 to n do
    1. For j =1 to n do
        1. A[i,j] = C[i,j]
2. For k=1 to n do
    1. For i =1 to n do
        1. For j =1 to n do
            1. If A[i,k] +A[k,j] <A[i,j] then A[i,j] = A[i,k] +A[k,j]

❖ Thuật toán Floyd: Mô tả thuật toán

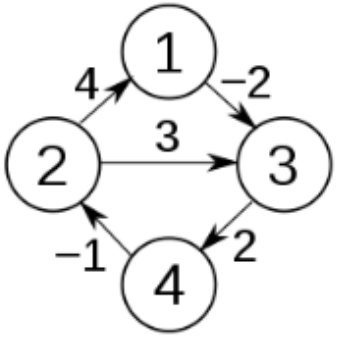
Cho đồ thị sau: Tìm kết quả cho thuật toán Floyd (xác định A_0 đến A_4. Ban đầu gán A_0 = C[i,j].



C[I,J]	0	1	2	3	4
0	0	10	∞	30	100
1	∞	0	50	∞	∞
2	∞	∞	0	∞	10
3	∞	∞	10	0	60
4	∞	∞	∞	35	0

❖ Thuật toán Floyd: Mô tả thuật toán

Ví dụ: Cho đồ thị có hướng sau. Tìm ma trận đường đi ngắn nhất theo Floyd



$k=0$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	−1	∞	0

$k=1$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	∞
	2	4	0	**2**	∞
	3	∞	∞	0	2
	4	∞	−1	∞	0

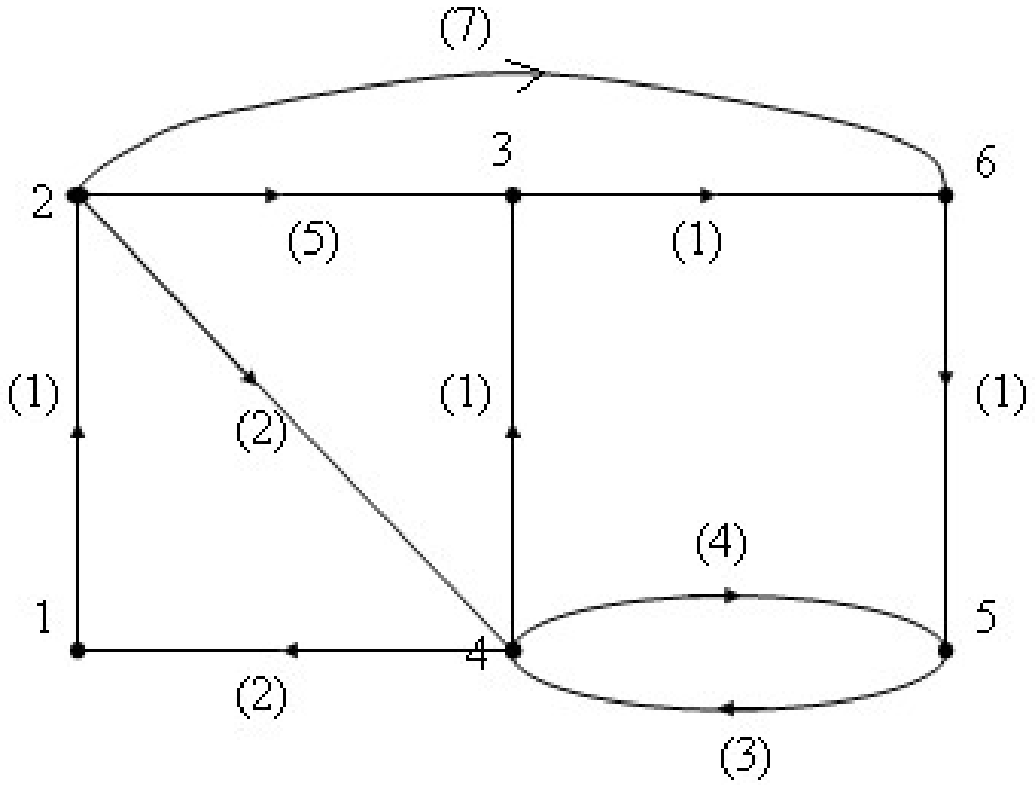
$k=2$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	∞
	2	4	0	2	∞
	3	∞	∞	0	2
	4	**3**	−1	**1**	0

$k=3$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	**0**
	2	4	0	2	**4**
	3	∞	∞	0	2
	4	3	−1	1	0

$k=4$		j			
		1	**2**	**3**	**4**
i	**1**	0	**−1**	−2	0
	2	4	0	2	4
	3	**5**	**1**	0	2
	4	3	−1	1	0

❖ Kiểm nghiệm thuật toán (1/3)

- Áp dụng thuật toán Floyd tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị.

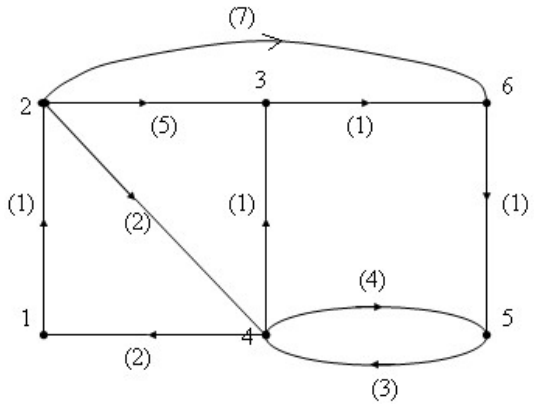


d[i][j] p[i][j]

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

Khởi tạo

Kiểm nghiệm thuật toán (2/3)



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

Khởi tạo

	1	2	3	4	5	6
1	0,1	1,1	∞,1	∞,1	∞,1	∞,1
2	∞,2	0,2	5,2	2,2	∞,2	7,2
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	**3,1**	1,4	0,4	4,4	∞,4
5	∞,5	∞,5	∞,5	3,5	0,5	∞,5
6	∞,6	∞,6	∞,6	∞,6	1,6	0,6

k=1

	1	2	3	4	5	6
1	0,1	1,1	**6,2**	**3,2**	∞,1	**8,2**
2	∞,2	0,2	5,2	2,2	∞,2	7,2
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	3,1	1,4	0,4	4,4	**10,2**
5	∞,5	∞,5	∞,5	3,5	0,5	∞,5
6	∞,6	∞,6	∞,6	∞,6	1,6	0,6

k=2

	1	2	3	4	5	6
1	0,1	1,1	6,2	3,2	∞,1	**7,3**
2	∞,2	0,2	5,2	2,2	∞,2	**6,3**
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	3,1	1,4	0,4	4,4	**2,3**
5	∞,5	∞,5	∞,5	3,5	0,5	∞,5
6	∞,6	∞,6	∞,6	∞,6	1,6	0,6

k=3

	1	2	3	4	5	6
1	0,1	1,1	**4,4**	3,2	**7,4**	**5,3**
2	**4,4**	0,2	**3,4**	2,2	**6,4**	**4,3**
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	3,1	1,4	0,4	4,4	2,3
5	**5,4**	**6,1**	**4,4**	3,5	0,5	**5,3**
6	∞,6	∞,6	∞,6	∞,6	1,6	0,6

k=4

	1	2	3	4	5	6
1	0,1	1,1	4,4	3,2	7,4	5,3
2	4,4	0,2	3,4	2,2	6,4	4,3
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	3,1	1,4	0,4	4,4	2,3
5	5,4	6,1	4,4	3,5	0,5	5,3
6	**6,4**	**7,1**	**5,4**	**4,5**	1,6	0,6

k=5

	1	2	3	4	5	6
1	0,1	1,1	4,4	3,2	**6,6**	5,3
2	4,4	0,2	3,4	2,2	**5,6**	4,3
3	**7,4**	**8,1**	0,3	**5,5**	**2,6**	1,3
4	2,4	3,1	1,4	0,4	**3,6**	2,3
5	5,4	6,1	4,4	3,5	0,5	5,3
6	6,4	7,1	5,4	4,5	1,6	0,6

k=6

Kiểm nghiệm thuật toán (3/3)

	1	2	3	4	5	6
1	0,1	1,1	4,4	3,2	**6,6**	5,3
2	4,4	0,2	3,4	2,2	**5,6**	4,3
3	**7,4**	**8,1**	0,3	**5,5**	**2,6**	1,3
4	2,4	3,1	1,4	0,4	**3,6**	2,3
5	5,4	6,1	4,4	3,5	0,5	5,3
6	6,4	7,1	5,4	4,5	1,6	0,6

k=6

KQ:

1-> 2 (CD= 1): 2 <- 1
1-> 3 (CD= 4): 3 <- 4 <- 2 <- 1
1-> 4 (CD= 3): 4 <- 2 <- 1
1-> 5 (CD= 6): 5 <- 6 <- 3 <- 4 <- 2 <- 1
1-> 6 (CD= 5): 6 <- 3 <- 4 <- 2 <- 1

2-> 1 (CD= 4): 1 <- 4 <- 2
2-> 3 (CD= 3): 3 <- 4 <- 2
2-> 4 (CD= 2): 4 <- 2
2-> 5 (CD= 5): 5 <- 6 <- 3 <- 4 <- 2
2-> 6 (CD= 4): 6 <- 3 <- 4 <- 2

3-> 1 (CD= 7): 1 <- 4 <- 5 <- 6 <- 3
3-> 2 (CD= 8): 2 <- 1 <- 4 <- 5 <- 6 <- 3
3-> 4 (CD= 5): 4 <- 5 <- 6 <- 3
3-> 5 (CD= 2): 5 <- 6 <- 3
3-> 6 (CD= 1): 6 <- 3

4-> 1 (CD= 2): 1 <- 4
4-> 2 (CD= 3): 2 <- 1 <- 4
4-> 3 (CD= 1): 3 <- 4
4-> 5 (CD= 3): 5 <- 6 <- 3 <- 4
4-> 6 (CD= 2): 6 <- 3 <- 4

5-> 1 (CD= 5): 1 <- 4 <- 5
5-> 2 (CD= 6): 2 <- 1 <- 4 <- 5
5-> 3 (CD= 4): 3 <- 4 <- 5
5-> 4 (CD= 3): 4 <- 5
5-> 6 (CD= 5): 6 <- 3 <- 4 <- 5

6-> 1 (CD= 6): 1 <- 4 <- 5 <- 6
6-> 2 (CD= 7): 2 <- 1 <- 4 <- 5 <- 6
6-> 3 (CD= 5): 3 <- 4 <- 5 <- 6
6-> 4 (CD= 4): 4 <- 5 <- 6
6-> 5 (CD= 1): 5 <- 6

➢ Thuật toán Dijkstra

- Thuật toán Dijkstra dùng để tìm đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại trong đồ thị.
- Được sử dụng cho đồ thị không có cung **trọng số** âm.
- Thuật toán
  - Đầu vào
    - Đồ thị có hướng G=(V,E) với n đỉnh.
    - s $\in$ V là đỉnh xuất phát.
    - a[u,v], u,v $\in$ V là ma trận trọng số
  - Đầu ra
    - Khoảng cách từ s đến tất cả các đỉnh còn lại d[v], v $\in$ V .
    - Truoc[v], v $\in$ V là đỉnh đi trước v trong đường đi ngắn nhất từ s đến v.

➢ Thuật toán Dijkstra

1. Ý tưởng

Thuật toán Dijkstra có thể giải quyết bài toán tìm đường đi ngắn nhất trên đồ thị vô hướng lẫn có hướng miễn là trọng số **không âm**.

Ý tưởng cơ bản của thuật toán như sau:

- *Bước 1*: Từ đỉnh gốc, khởi tạo khoảng cách tới chính nó là 0, khởi tạo khoảng cách nhỏ nhất ban đầu tới các đỉnh khác là $+\infty$. Ta được danh sách các khoảng cách tới các đỉnh.
- *Bước 2*: Chọn đỉnh `a` có khoảng cách nhỏ nhất trong danh sách này và ghi nhận. Các lần sau sẽ không xét tới đỉnh này nữa.
- *Bước 3*: Lần lượt xét các đỉnh kề `b` của đỉnh `a`. Nếu *khoảng cách từ đỉnh gốc* tới đỉnh `b` nhỏ hơn khoảng cách hiện tại đang được ghi nhận thì cập nhật giá trị và đỉnh kề `a` vào khoảng cách hiện tại của `b`.
- *Bước 4*: Sau khi xét tất cả đỉnh kề `b` của đỉnh `a`. Lúc này ta được danh sách khoảng cách tới các điểm đã được cập nhật. Quay lại *Bước 2* với danh sách này. Thuật toán kết thúc khi chọn được khoảng cách nhỏ nhất từ tất cả các điểm.

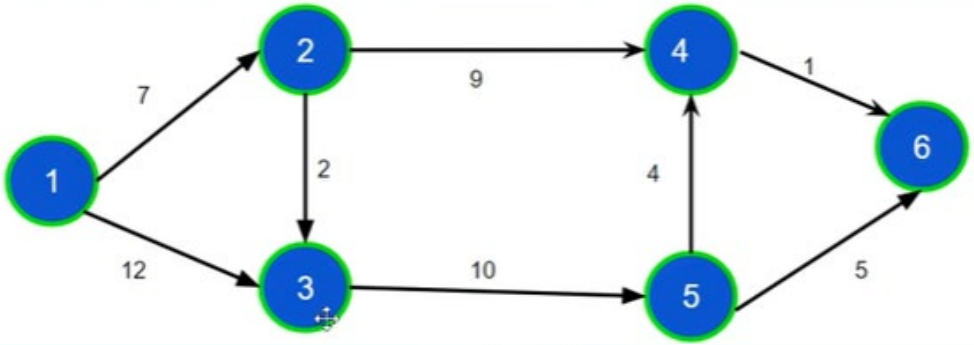
❖ Thuật toán Dijkstra

Áp dụng : Tìm đường đi ngắn nhất từ 1 đỉnh S tới mọi đỉnh còn lại trên đồ thị, Dijkstra có thể áp dụng cho cả đồ thị có hướng hoặc vô hướng không có trọng số âm.

Độ phức tạp : O((E+V) logV)

Tham khảo : Hackerearth

Ví dụ:



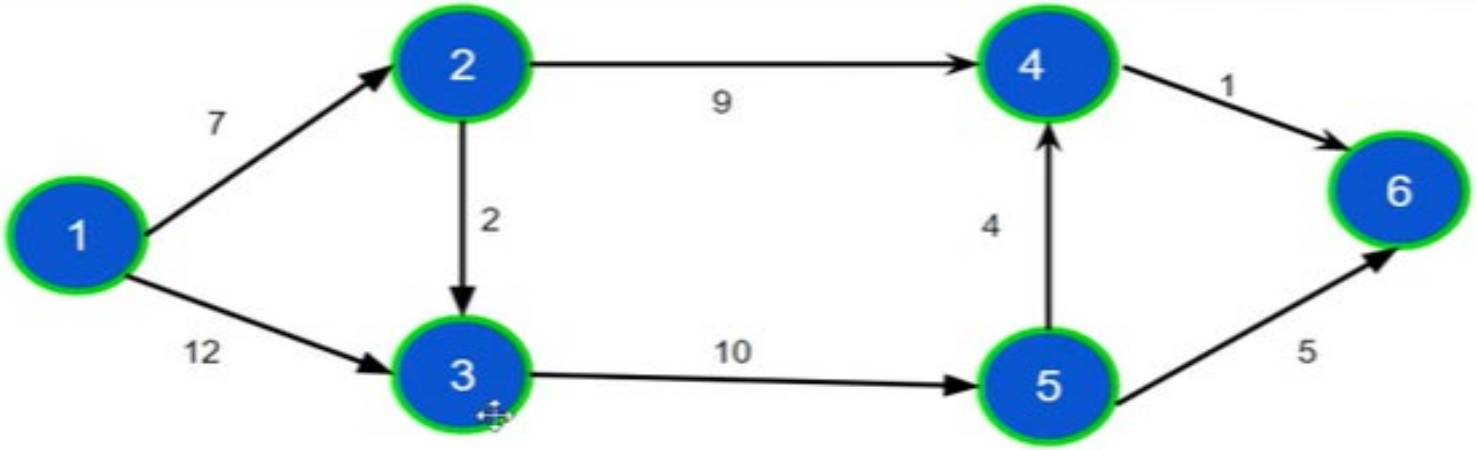
Iteration	unmarked	marked	Current	1	2	3	4	5	6
	{1, 2, 3, 4, 5, 6}			0	Inf	Inf	Inf	Inf	Inf
1									
2									
3									
4									
5									
6									

Relaxation : d[v] = min(d[v], d[u] + len(u, v))

Bước đầu tiên cho đỉnh nguồn tới chính nó bằng 0. Còn lại, từ đỉnh nguồn tới đỉnh khác là vô cùng (Inf). Bắt đầu duyệt từ đỉnh có trọng số nhỏ nhất là đỉnh 1 (trong số bằng 0)

Relaxation : d[v] = min(d[v], d[u] + len(u, v))

❖ Thuật toán Dijkstra



Iteration	unmarked	marked	Current	1	2	3	4	5	6
	{1, 2, 3, 4, 5, 6}			0	Inf	Inf	Inf	Inf	Inf
1	{2, 3, 4, 5, 6}	{1}	u = 1	0	7	12	Inf	Inf	Inf
2									
3									
4									
5									
6									

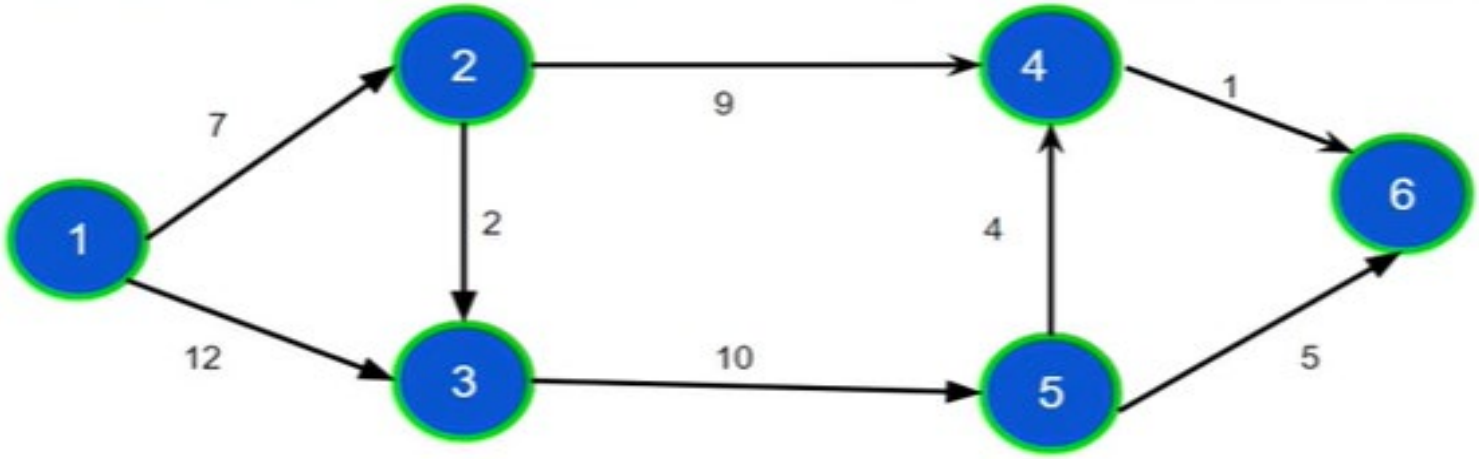
Xét đỉnh u là 1: có hai đường tới đỉnh 2 & 3: cập nhật trọng số đỉnh 2 và 3:

* Đỉnh 2: d[2] = min { d[2]; d[1] + len(1,2)} = min {Inf; (0 + 7)} = 7

* Đỉnh 3: d[3] = min { d[3]; d[1] + len(1,3)} = min { Inf; (0+12)} = 12

Relaxation : d[v] = min(d[v], d[u] + len(u, v))

❖ Thuật toán Dijkstra



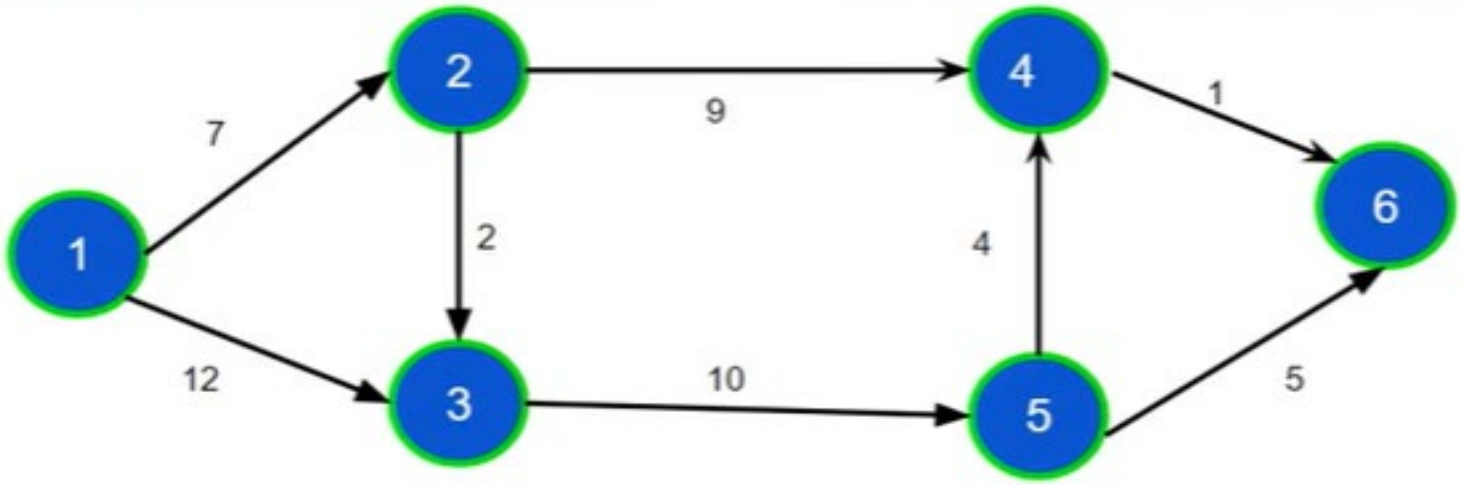
Iteration	unmarked	marked	Current	1	2	3	4	5	6
	{1, 2, 3, 4, 5, 6}			0	Inf	Inf	Inf	Inf	Inf
1	{2, 3, 4, 5, 6}	{1}	u = 1	0	7	12	Inf	Inf	Inf
2	{3, 4, 5, 6}	{1, 2}	u = 2	0	7	9	16	Inf	Inf
3									
4									
5									
6									

Relaxation : d[v] = min(d[v], d[u] + len(u, v)) d(4) = min(inf, d[2] + 9)

Ở bước 1, xét các đỉnh 2,3,4,5,6 thì đỉnh 2 có trọng số nhỏ nhất là 7. Nên xét tiếp đỉnh 2 (bước 2). Đỉnh 2 đi tới đỉnh 3 & 4 nên cập nhật trọng số tại đỉnh 3 & 4:

- Đỉnh 3: d(3) = min (d(3); d(2) + len (2,3) = min (12; 7 + 2) = 9
- Đỉnh 4: d (4) = min (d(4); d (2) +len (2,4) = min (Inf; 7 + 9) = 16

❖ Thuật toán Dijkstra

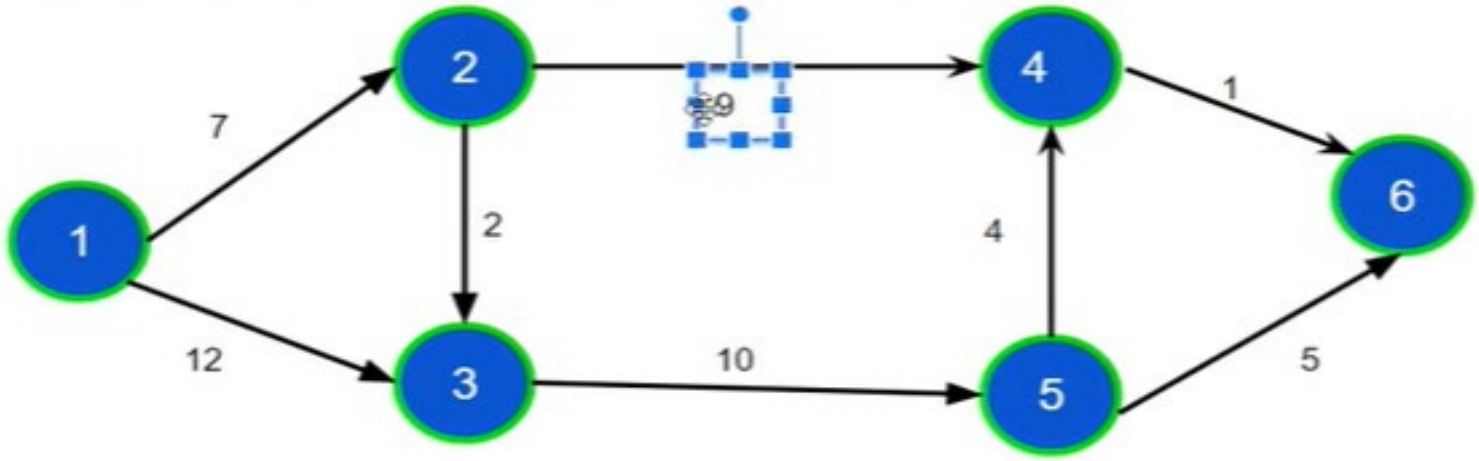


Iteration	unmarked	marked	Current	1	2	3	4	5	6
	{1, 2, 3, 4, 5, 6}			0	Inf	Inf	Inf	Inf	Inf
1	{2, 3, 4, 5, 6}	{1}	u = 1	0	7	12	Inf	Inf	Inf
2	{3, 4, 5, 6}	{1, 2}	u = 2	0	7	9	16	Inf	Inf
3	{4, 5, 6}	{1, 2, 3}	u = 3	0	7	9	16	19	Inf
4									
5									
6									

Sau khi xong bước 2, còn lại các đỉnh 3,4,5,6 thì đỉnh 3 có trọng số nhỏ nhất là 9. Tiếp theo ta marked đỉnh 3. Tại đỉnh 3 đi tới đỉnh 5. Ta cập nhật trong số đỉnh 5: d[5] = min (d [5]; d[3] + len (3,5)) = min (Inf; 9 + 10) = 19

Relaxation : d[v] = min(d[v], d[u] + len(u, v)) d(5) = min(inf, 9 + 10) = 19

❖ Thuật toán Dijkstra

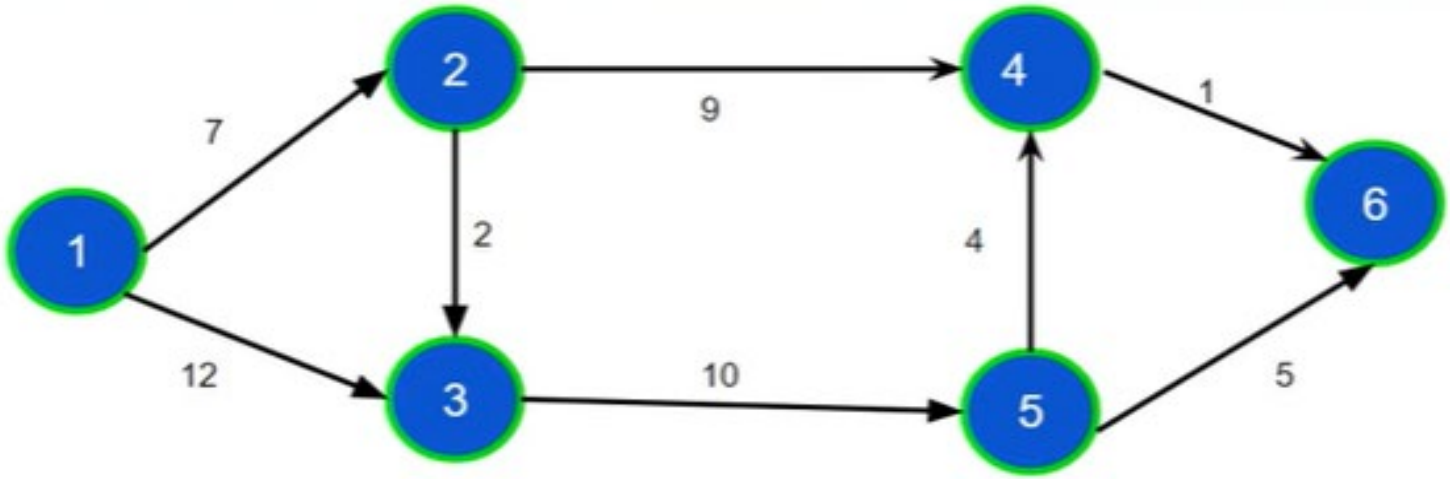


Iteration	unmarked	marked	Current	1	2	3	4	5	6
	{1, 2, 3, 4, 5, 6}			0	Inf	Inf	Inf	Inf	Inf
1	{2, 3, 4, 5, 6}	{1}	u = 1	0	7	12	Inf	Inf	Inf
2	{3, 4, 5, 6}	{1, 2}	u = 2	0	7	9	16	19	Inf
3	{4, 5, 6}	{1, 2, 3}	u = 3	0	7	9	16	Inf	Inf
4	{5, 6}	{1, 2, 3, 4}	u = 4	0	7	9	16	19	17
5									
6									

Trong bước 3: Đỉnh 4,5,6 umarked thì đỉnh 4 có trọng số nhỏ nhất, nên xét đỉnh 4. Đỉnh 4 đi tới đỉnh 6 ta cập nhật trọng số đỉnh 6 (bước 4):

Relaxation : d[v] = min(d[v], d[u] + len(u, v)) d(6) = min(inf, 16 + 1) = 17

❖ Thuật toán Dijkstra

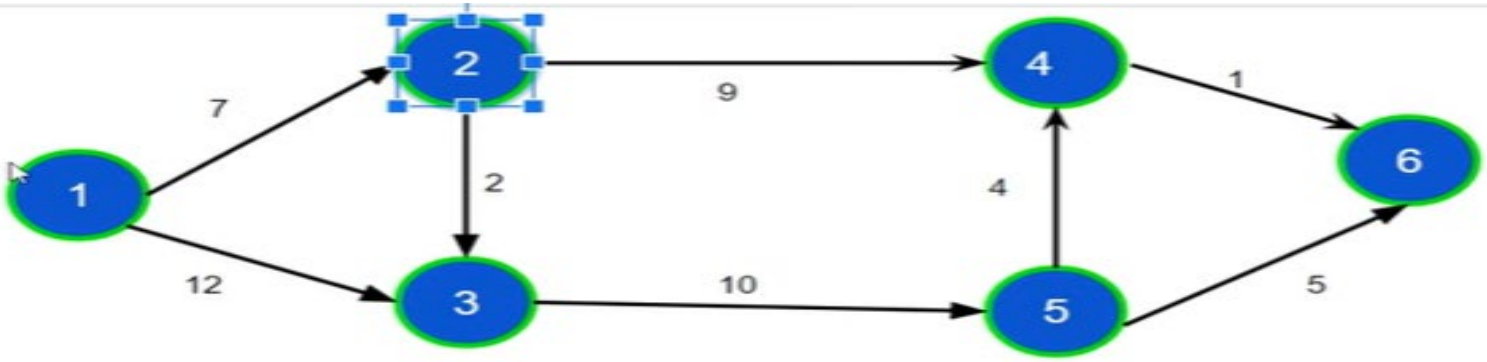


Iteration	unmarked	marked	Current	1	2	3	4	5	6
	{1, 2, 3, 4, 5, 6}			0	Inf	Inf	Inf	Inf	Inf
1	{2, 3, 4, 5, 6}	{1}	u = 1	0	7	12	Inf	Inf	Inf
2	{3, 4, 5, 6}	{1, 2}	u = 2	0	7	9	16	19	Inf
3	{4, 5, 6}	{1, 2, 3}	u = 3	0	7	9	16	19	Inf
4	{5, 6}	{1, 2, 3, 4}	u = 4	0	7	9	16	19	17
5	{5}	{1, 2, 3, 4 , 6}	u = 6	0	7	9	16	19	17
6									

Trong bước 4: xét đỉnh 5,6 thì đỉnh 6 có trong số nhỏ nhất, nên marked đỉnh 6 (còn lại đỉnh 5). Đỉnh 6 không tới đỉnh nào cả nên giữ nguyên không cập nhật trong số

Relaxation : d[v] = min(d[v], d[u] + len(u, v)) d(6) = min(inf, 16 + 1) = 17

❖ Thuật toán Dijkstra



Iteration	unmarked	marked	Current	1	2	3	4	5	6
	{1, 2, 3, 4, 5, 6}			0	Inf	Inf	Inf	Inf	Inf
1	{2, 3, 4, 5, 6}	{1}	u = 1	0	7	12	Inf	Inf	Inf
2	{3, 4, 5, 6}	{1, 2}	u = 2	0	7	9	16	19	Inf
3	{4, 5, 6}	{1, 2, 3}	u = 3	0	7	9	16	19	Inf
4	{5, 6}	{1, 2, 3, 4}	u = 4	0	7	9	16	19	17
5	{5}	{1, 2, 3, 4 , 6}	u = 6	0	7	9	16	19	17
6	{}	{1, 2, 3, 4, 6, 5}	u = 5	0	7	9	16	19	17

Relaxation : d[v] = min(d[v], d[u] + len(u, v)) d(4) = min(16, 19 + 4) = 16

Ở bước 5, còn lại đỉnh 5, tiếp tục marked đỉnh 5 (bước 6). Đỉnh 5 sẽ tới đỉnh 4 và 6. Ta cập nhật trọng số đỉnh 4 và 6:

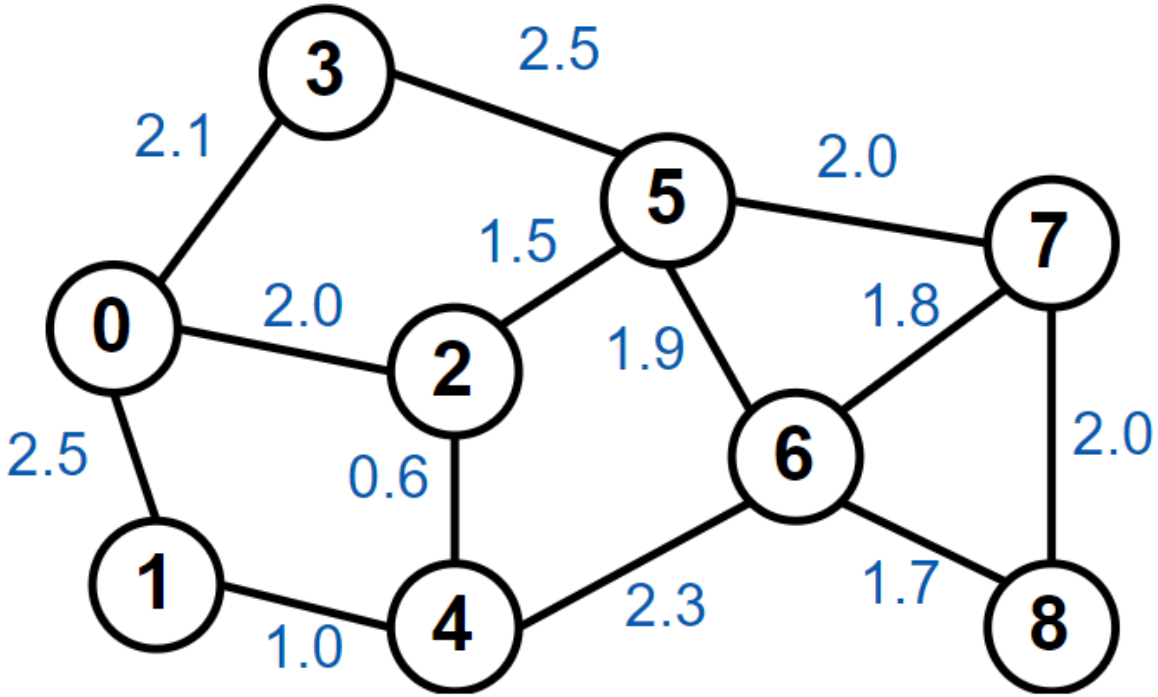
- Đỉnh 4: d(4) = min (d(4); d(5) + len (5,4) = min (16; 19+4) = 16 bằng giá trị trọng số đã có nên không cập nhật.
- Đỉnh 6: d(6) = min (d(6); d(5) + len (5,6)) = min (17; 19 +5) = 17 bằng giá trị trong số đã có nên không cập nhật.

❖ Thuật toán Dijkstra

2. Ví dụ

Để dễ dàng hiểu ý tưởng của thuật toán. Chúng ta cùng xem ví dụ với đồ thị vô hướng G.

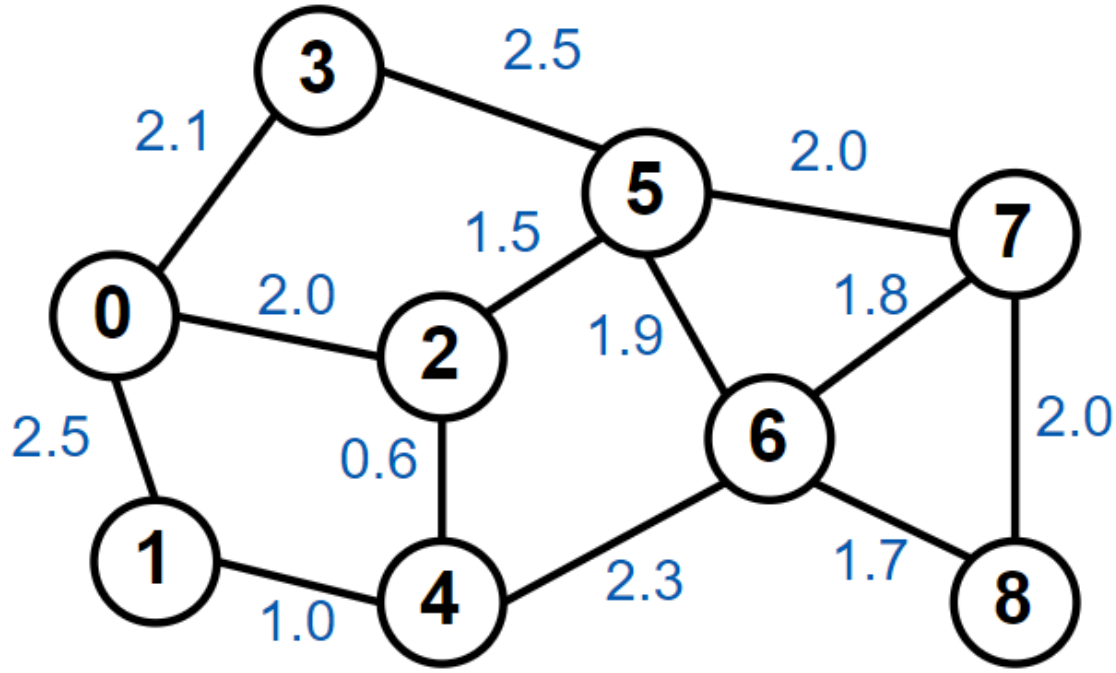
Thuật toán Dijkstra sẽ tìm khoảng cách từ đỉnh gốc 0 tới tất cả các đỉnh còn lại trong đồ thị G.



Đồ thị G

❖ Thuật toán Dijsktra

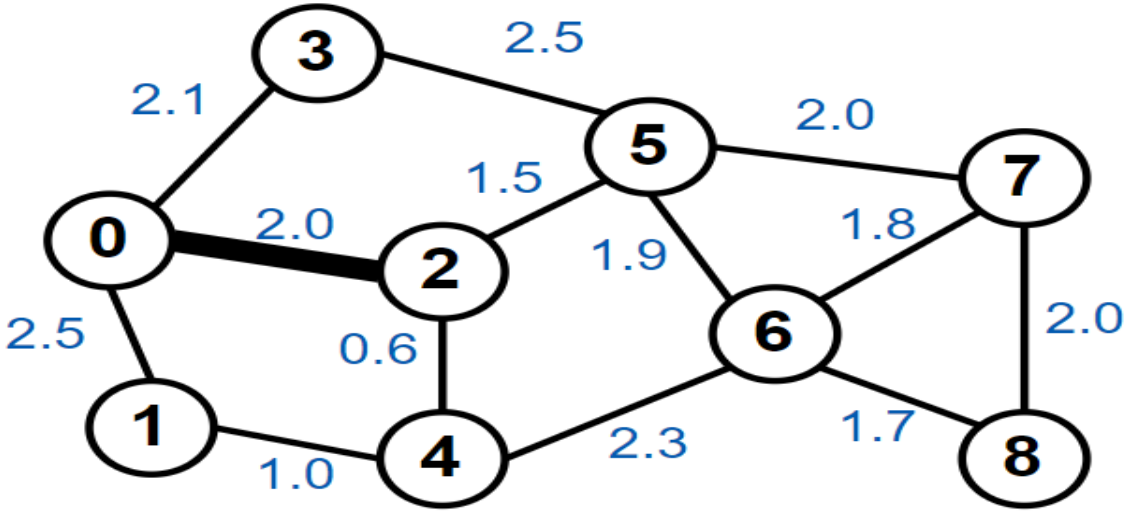
Đầu tiên, khởi tạo khoảng cách nhỏ nhất ban đầu tới các đỉnh khác là $+\infty$ và khoảng cách tới đỉnh gốc là 0. Ta được danh sách các khoảng cách tới các đỉnh.



0	1	2	3	4	5	6	7	8
0	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$

❖ Thuật toán Dijsktra

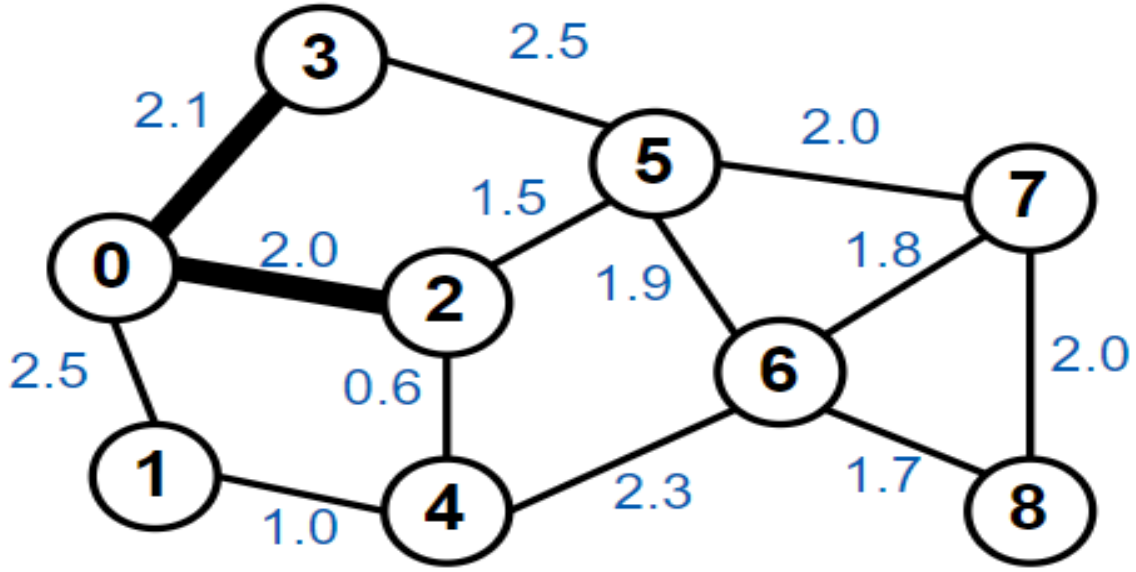
Chọn đỉnh 0 có giá trị nhỏ nhất, xét các đỉnh kề của đỉnh 0: Xét đỉnh 1, khoảng cách từ gốc đến đỉnh 1 là $2.5 < \infty$ nên ghi nhận giá trị mới là (2.5, 0) (nghĩa là khoảng cách đến đỉnh gốc hiện tại ghi nhận là 2.5, đỉnh kề liền trước là đỉnh 0). Xét tương tự cho đỉnh 2 và 3, ta được dòng thứ 2 trong bảng.



0	1	2	3	4	5	6	7	8
0	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	**(2.0, 0)**	(2.1, 0)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)

❖ Thuật toán Dijsktra

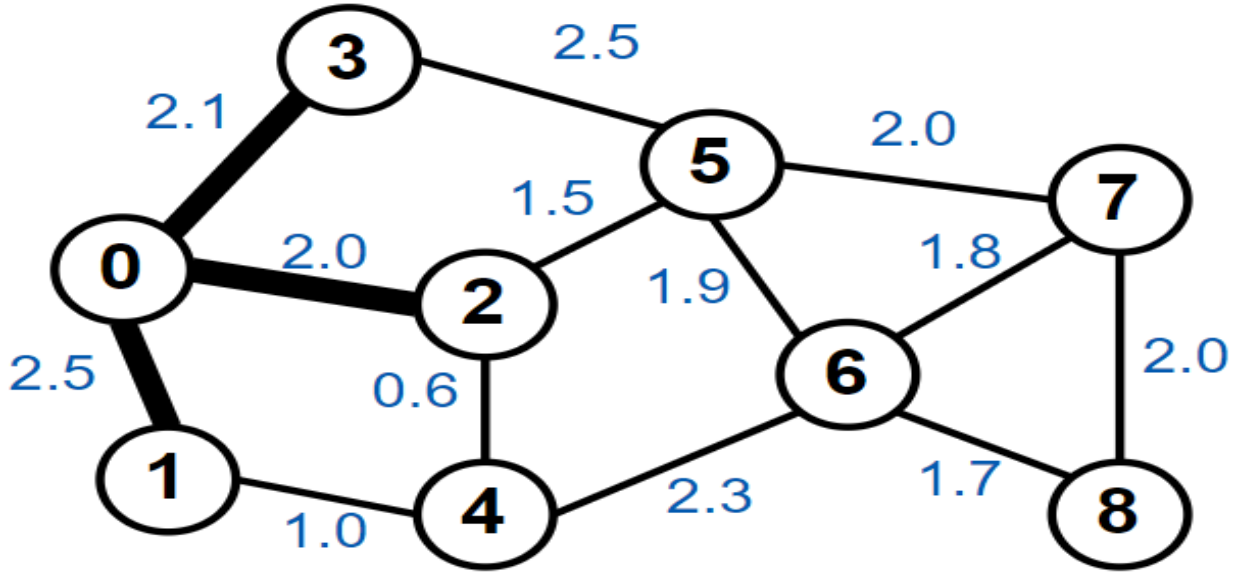
Sau khi xét tất cả các đỉnh ta chọn đỉnh 2 có khoảng cách nhỏ nhất và ghi nhận để xét tiếp. Tiếp tục xét đỉnh kề của 2 là đỉnh 4 và 5 với nguyên tắc nêu ở trên. Xét đỉnh 4, khoảng cách từ đỉnh gốc đến đỉnh 4 sẽ bằng khoảng cách từ đỉnh gốc tới đỉnh 2 cộng khoảng cách từ 2 đến 4. Nghĩa là $2.0 + 0.6 = 2.6$ nên ta ghi nhận khoảng cách tại đỉnh 4 là $(2.6, 2)$. Xét tương tự cho đỉnh 5.



0	1	2	3	4	5	6	7	8
0	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$
–	$(2.5, 0)$	**$(2.0, 0)$**	$(2.1, 0)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$
–	$(2.5, 0)$	–	**$(2.1, 0)$**	$(2.6, 2)$	$(3.5, 2)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$

❖ Thuật toán Dijsktra

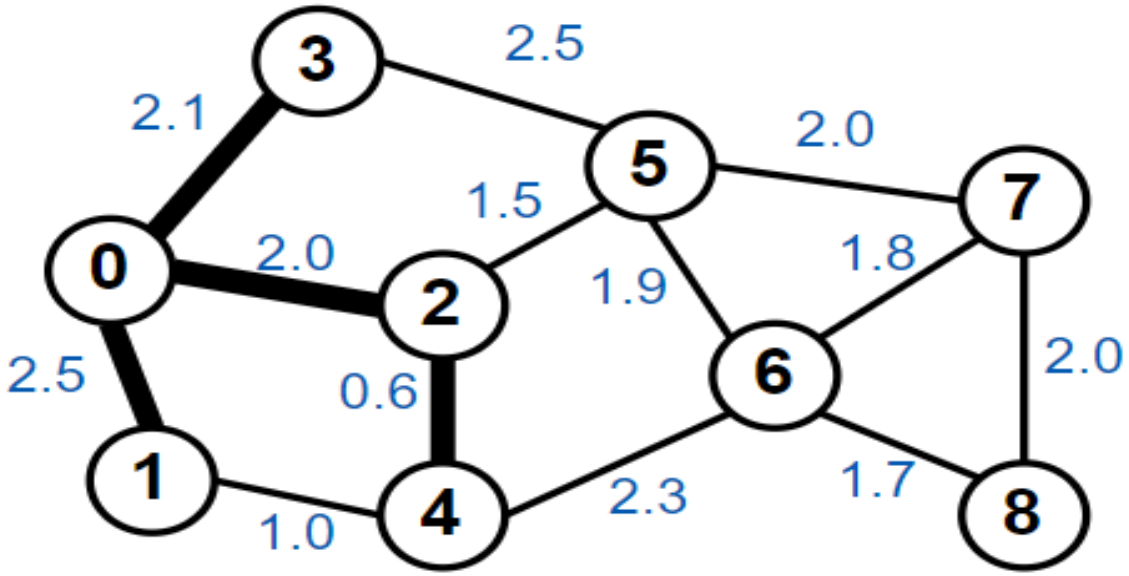
Lúc này ta chọn được đỉnh 3 có khoảng cách nhỏ nhất, xét đỉnh kề của đỉnh 3 là đỉnh 5. Khoảng cách từ gốc tới đỉnh $5 = 2.1 + 2.5 = 4.6$ lớn hơn khoảng cách hiện tại được ghi nhận, vì vậy giá trị tại đỉnh 5 không đổi.



0	1	2	3	4	5	6	7	8
0	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$
–	$(2.5, 0)$	**$(2.0, 0)$**	$(2.1, 0)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$
–	$(2.5, 0)$	–	**$(2.1, 0)$**	$(2.6, 2)$	$(3.5, 2)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$
–	**$(2.5, 0)$**	–	–	$(2.6, 2)$	$(3.5, 2)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$

❖ Thuật toán Dijsktra

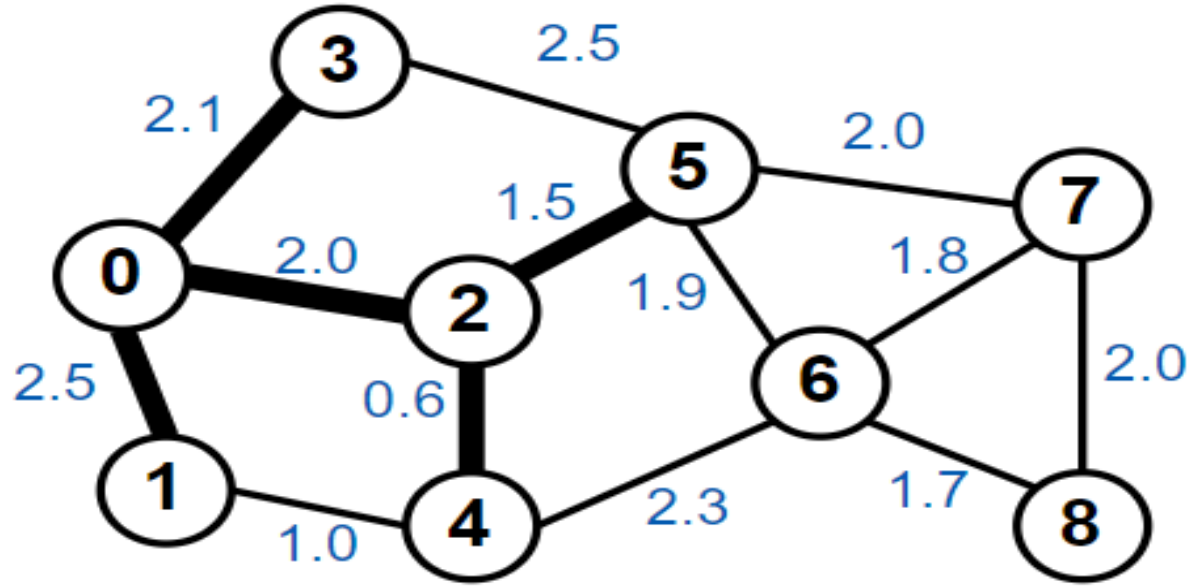
Đỉnh 1 là đỉnh được chọn tiếp theo, xét đỉnh kề của 1 là đỉnh 4. Khoảng cách từ đỉnh gốc không nhỏ hơn khoảng cách hiện tại nên ta không cập nhật gì ở đỉnh này.



0	1	2	3	4	5	6	7	8
0	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	**(2.0, 0)**	(2.1, 0)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	–	**(2.1, 0)**	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	**(2.5, 0)**	–	–	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	–	–	–	**(2.6, 2)**	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)

❖ Thuật toán Dijsktra

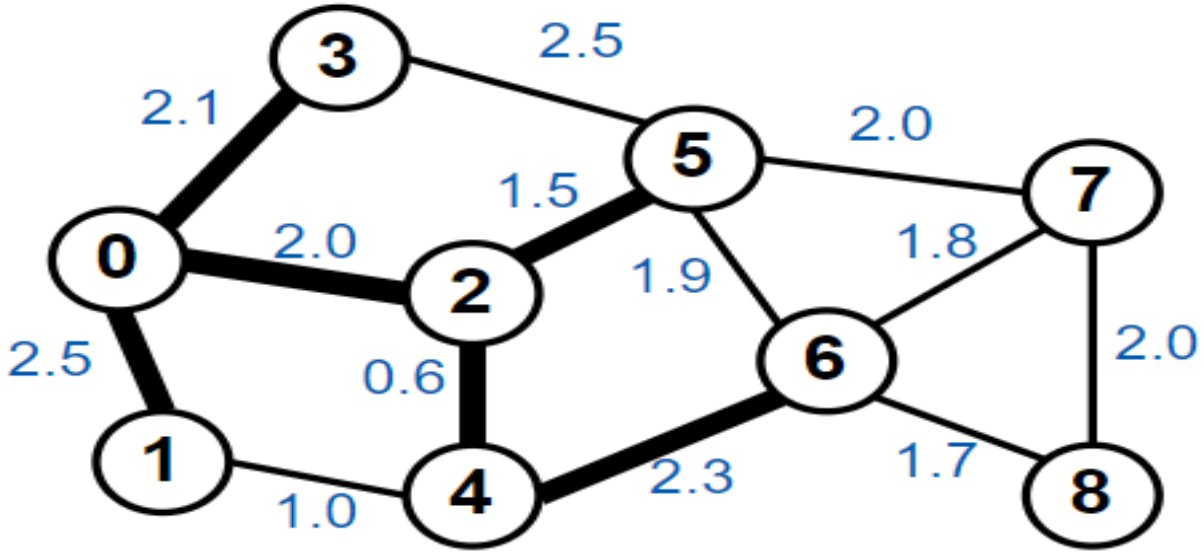
Sau khi xét xong ta chọn được đỉnh 4 là đỉnh tiếp theo. Ta cập nhật giá trị mới cho đỉnh 6.



0	1	2	3	4	5	6	7	8
0	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	**(2.0, 0)**	(2.1, 0)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	–	**(2.1, 0)**	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	**(2.5, 0)**	–	–	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	–	–	–	**(2.6, 2)**	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	–	–	–	–	**(3.5, 2)**	(4.9, 4)	(∞,–)	(∞,–)

Thuật toán Dijsktra

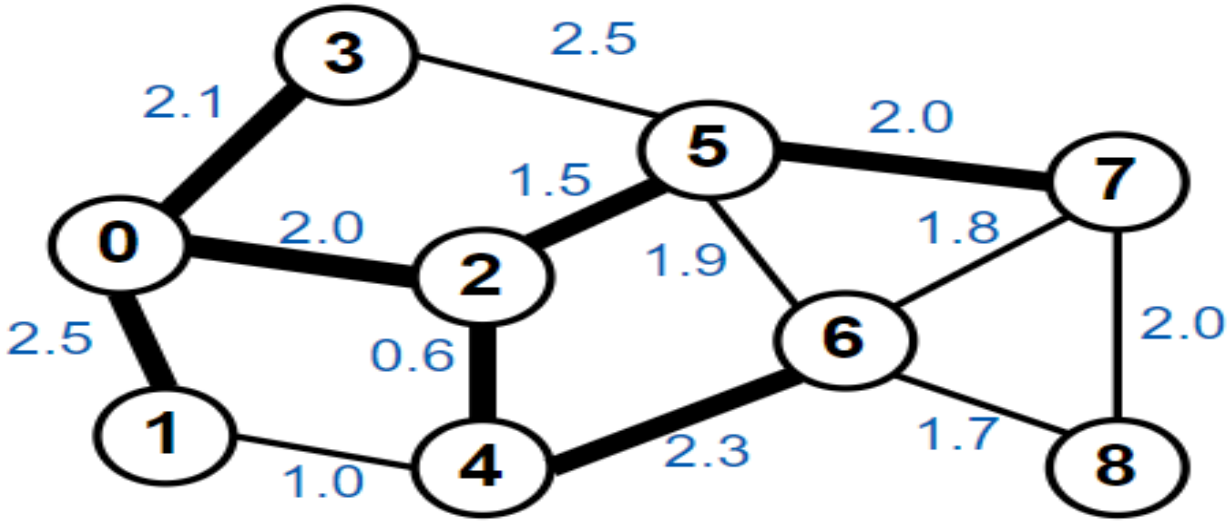
Chọn được đỉnh 5 là đỉnh nhỏ nhất, tiếp tục xét các đỉnh kề.



0	1	2	3	4	5	6	7	8
0	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	**(2.0, 0)**	(2.1, 0)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	–	**(2.1, 0)**	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	**(2.5, 0)**	–	–	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	–	–	–	**(2.6, 2)**	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	–	–	–	–	**(3.5, 2)**	(4.9, 4)	(∞,–)	(∞,–)
–	–	–	–	–	–	**(4.9, 4)**	(5.5, 5)	(∞,–)

❖ Thuật toán Dijsktra

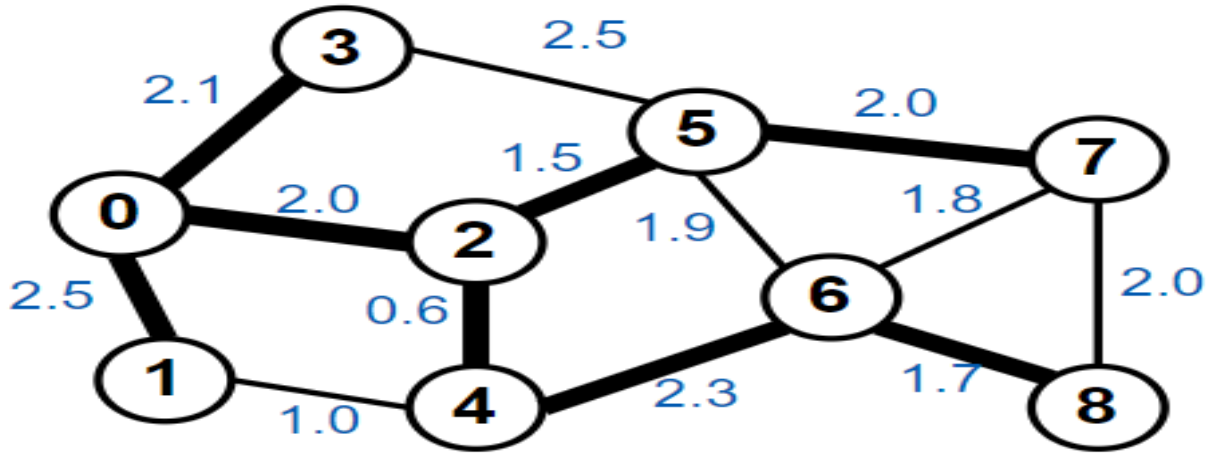
Đỉnh 6 là đỉnh tiếp theo được chọn.



0	1	2	3	4	5	6	7	8
0	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	**(2.0, 0)**	(2.1, 0)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	–	**(2.1, 0)**	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	**(2.5, 0)**	–	–	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	–	–	–	**(2.6, 2)**	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	–	–	–	–	**(3.5, 2)**	(4.9, 4)	(∞,–)	(∞,–)
–	–	–	–	–	–	**(4.9, 4)**	(5.5, 5)	(∞,–)
–	–	–	–	–	–	–	**(5.5, 5)**	(6.6, 6)

❖ Thuật toán Dijsktra

Chọn đỉnh có khoảng cách nhỏ nhất là đỉnh 7.



0	1	2	3	4	5	6	7	8
0	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	**(2.0, 0)**	(2.1, 0)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	(2.5, 0)	–	**(2.1, 0)**	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	**(2.5, 0)**	–	–	(2.6, 2)	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	–	–	–	**(2.6, 2)**	(3.5, 2)	(∞,–)	(∞,–)	(∞,–)
–	–	–	–	–	**(3.5, 2)**	(4.9, 4)	(∞,–)	(∞,–)
–	–	–	–	–	–	**(4.9, 4)**	(5.5, 5)	(∞,–)
–	–	–	–	–	–	–	**(5.5, 5)**	(6.6, 6)
–	–	–	–	–	–	–	–	**(6.6, 6)**

Thuật toán kết thúc khi chọn được khoảng cách nhỏ nhất cho tất cả các đỉnh.

❖ Thuật toán Dijsktra

```
function Dijkstra(Graph, source):
 2
 3      for each vertex v in Graph.Vertices:
 4          dist[v] ← INFINITY
 5          prev[v] ← UNDEFINED
 6          add v to Q
 7      dist[source] ← 0
 8
 9      while Q is not empty:
10          u ← vertex in Q with min dist[u]
11          remove u from Q
12
13          for each neighbor v of u still in Q:
14              alt ← dist[u] + Graph.Edges(u, v)
15              if alt < dist[v]:
16                  dist[v] ← alt
17                  prev[v] ← u
18
19       return dist[], prev[]
```

Thuật toán Ford - Bellman

➢ Thuật toán Ford-Bellman

- Thuật toán Ford-Bellman dùng để tìm đường đi ngắn nhất từ một đỉnh s đến tất cả các đỉnh còn lại của đồ thị.
- Được sử dụng cho đồ thị **không có chu trình âm**.

Cho đồ thị có hướng, có trọng số G=(V, E). Trọng số của các cạnh của G được tính như sau:

TrongSo(u, v) = ∞ nếu cung (u, v) $\notin$ E.

TrongSo(u, v) = a(u, v) nếu cung (u, v) $\in$ E.

Thuật toán tìm đường đi ngắn nhất d(v) từ đỉnh s đỉnh v, mọi v $\in$ V:

+ Xét u $\in$ V. Nếu d(u) + TrongSo(u, v) < d(v) thì ta thay d(v) = d(u) + TrongSo(u, v).

+ Quá trình này sẽ được lặp lại cho đến khi không thể có giá trị d(v) tốt hơn.

❖Thuật toán Ford-Bellman

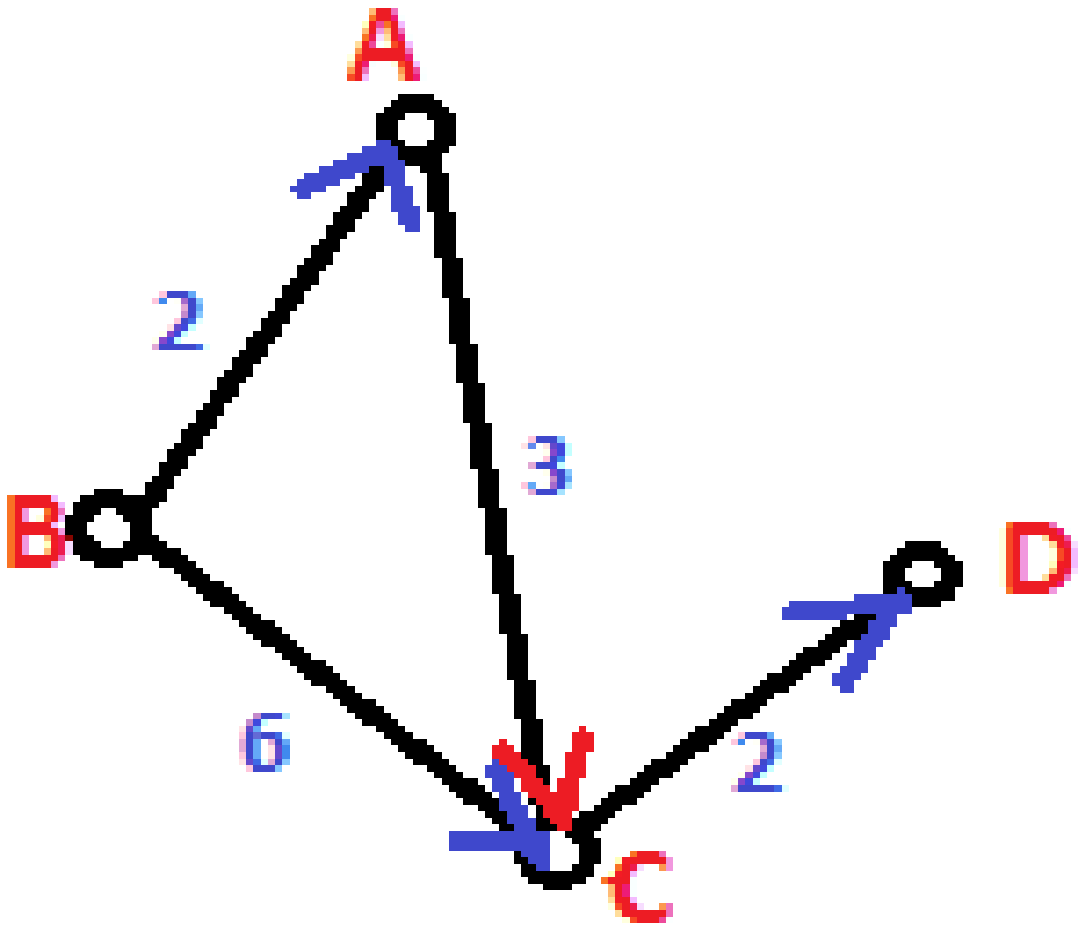
- Cài đặt thuật toán
  - Đầu vào:
    - Đồ thị có hướng G=(V,E) với n đỉnh.
    - s $\in$ V là đỉnh xuất phát.
    - a[u,v], u,v $\in$ V là ma trận trọng số
  - Đầu ra :
    - Khoảng cách từ s đến tất cả các đỉnh còn lại d[v], v $\in$ V .
    - Truoc[v], v $\in$ V là đỉnh đi trước v trong đường đi ngắn nhất từ s đến v

❖ Thuật toán Ford-Bellman

```
function BellmanFord(danh_sách_đỉnh, danh_sách_cung, nguồn)
    // hàm yêu cầu đồ thị đưa vào dưới dạng một danh sách đỉnh, một danh sách cung
    // hàm tính các giá trị khoảng_cách và đỉnh_liền_trước của các đỉnh,
    // sao cho các giá trị đỉnh_liền_trước sẽ lưu lại các đường đi ngắn nhất.

    // bước 1: khởi tạo đồ thị
    for each v in danh_sách_đỉnh:
        if v is nguồn then khoảng_cách(v):= 0
        else khoảng_cách(v):= vô cùng
        đỉnh_liền_trước(v):= null

    // bước 2: kết nạp cạnh
    for i from 1 to size(danh_sách_đỉnh)-1:
        for each (u,v) in danh_sách_cung:
            if khoảng_cách(v) > khoảng_cách(u) + trọng_số(u,v):
                khoảng_cách(v):= khoảng_cách(u) + trọng_số(u,v)
                đỉnh_liền_trước(v):= u

    // bước 3: kiểm tra chu trình âm
    for each (u,v) in danh_sách_cung:
        if khoảng_cách(v) > khoảng_cách(u) + trọng_số(u,v):
            error "Đồ thị chứa chu trình âm"
```

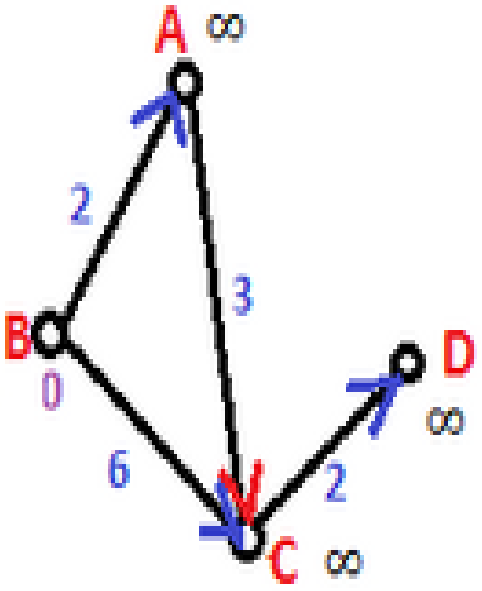
➢ Thuật toán Ford - Bellman

Ví dụ: Tìm đường đi ngắn nhất từ đỉnh B tới đỉnh D của đồ thị G



❖ Thuật toán Ford - Bellman

- Bước 0: Ta đánh dấu đỉnh xuất phát = 0, các đỉnh còn lại bằng vô cực.



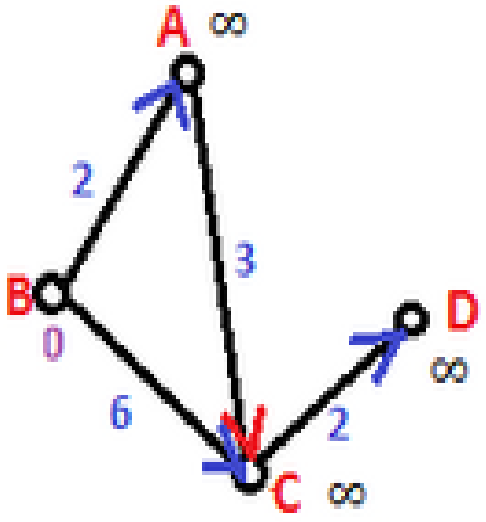
Step	0	1	2	3	4
A	∞				
B	0				
C	∞				
D	∞				

❖ Thuật toán Ford - Bellman

❑ - Bước 1:

Tại đỉnh A có đỉnh B đi vào có chi phí hiện tại (2 = 0 +2) < chi phí trước (∞) => cập nhật lại chi phí đỉnh A

Tại đỉnh C có đỉnh B đi vào có chi phí hiện tại (6 = 0+6) < chi phí trước (∞) => cập nhật lại chi phí đỉnh C



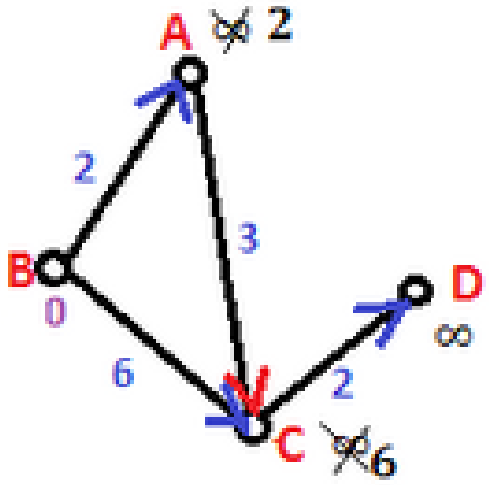
Step	0	1	2	3	4
A	∞	2			
B	0	0			
C	∞	6			
D	∞	∞			

❖ Thuật toán Ford - Bellman

❑ Bước 2:

Tại đỉnh C có đỉnh A đi vào có chi phí hiện tại (5 = 2+3) < chi phí trước (6) => cập nhật lại chi phí đỉnh C

Tại đỉnh D có đỉnh C đi vào có chi phí hiện tại (8 = 6+2) < chi phí trước (∞) => cập nhật lại chi phí đỉnh D

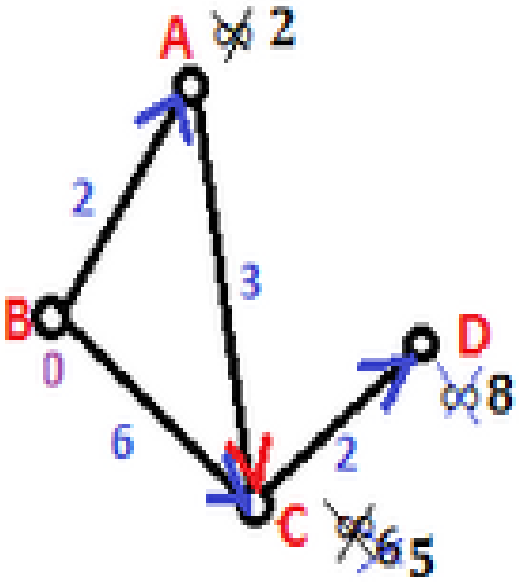


Step	0	1	2	3	4
A	∞	2	2		
B	0	0	0		
C	∞	6	5		
D	∞	∞	8		

❖ Thuật toán Ford - Bellman

❑ Bước 3:

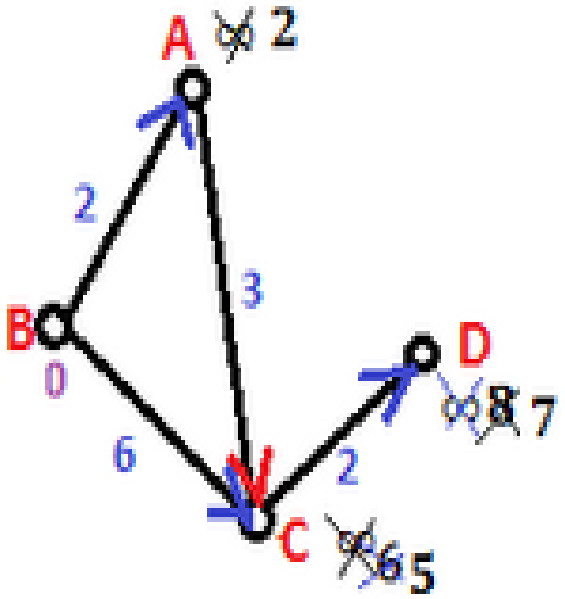
Tại đỉnh D có đỉnh A đi vào có chi phí hiện tại (7 = 5 + 2) < chi phí trước (8) => cập nhật lại chi phí đỉnh **D**



Step	0	1	2	3	4
A	∞	2	2	2	
B	0	0	0	0	
C	∞	6	5	5	
D	∞	∞	8	7	

❖ Thuật toán Ford - Bellman

❑ Bước 4: Bước 4 giống bước 3 nên thuật toán dừng.



Step	0	1	2	3	4
A	∞	2	2	2	2
B	0	0	0	0	0
C	∞	6	5	5	5
D	∞	∞	8	7	7

Kết luận:

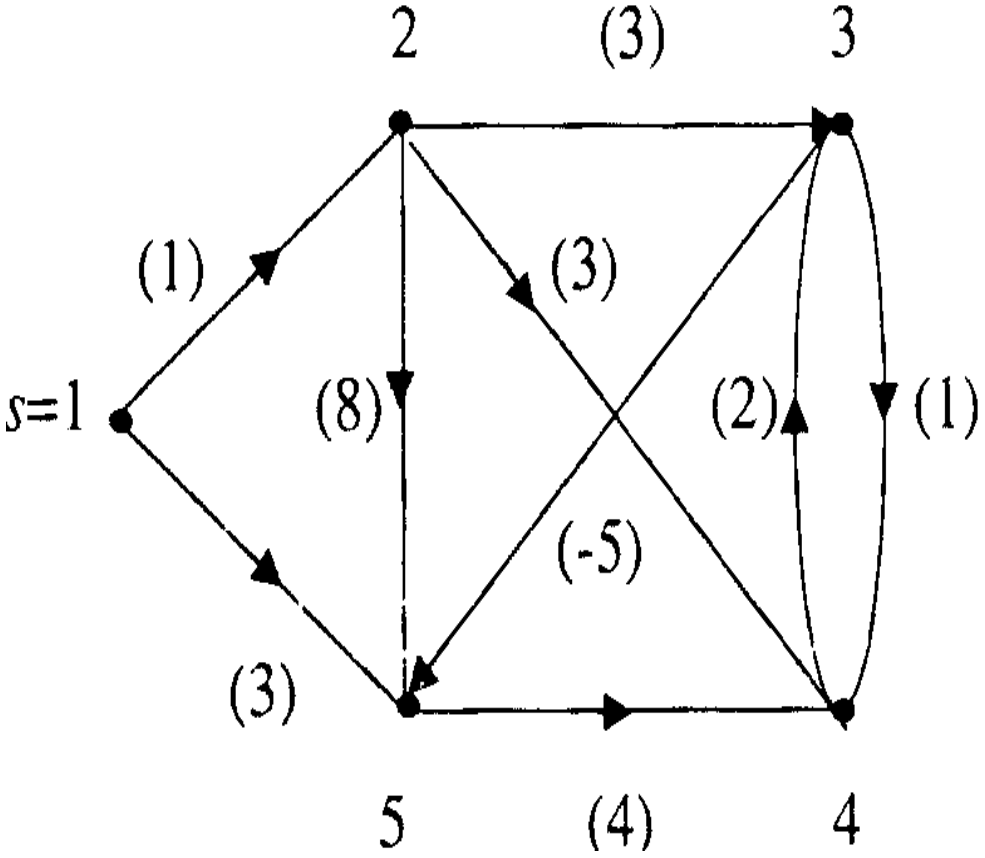
Có đường đi ngắn nhất từ **B->D: B->A->C->D**

- Lưu ý:

Nếu bước 4 không giống bước 3 => kết luận không có đường đi ngắn nhất từ B->D

❖ Thuật toán Ford - Bellman

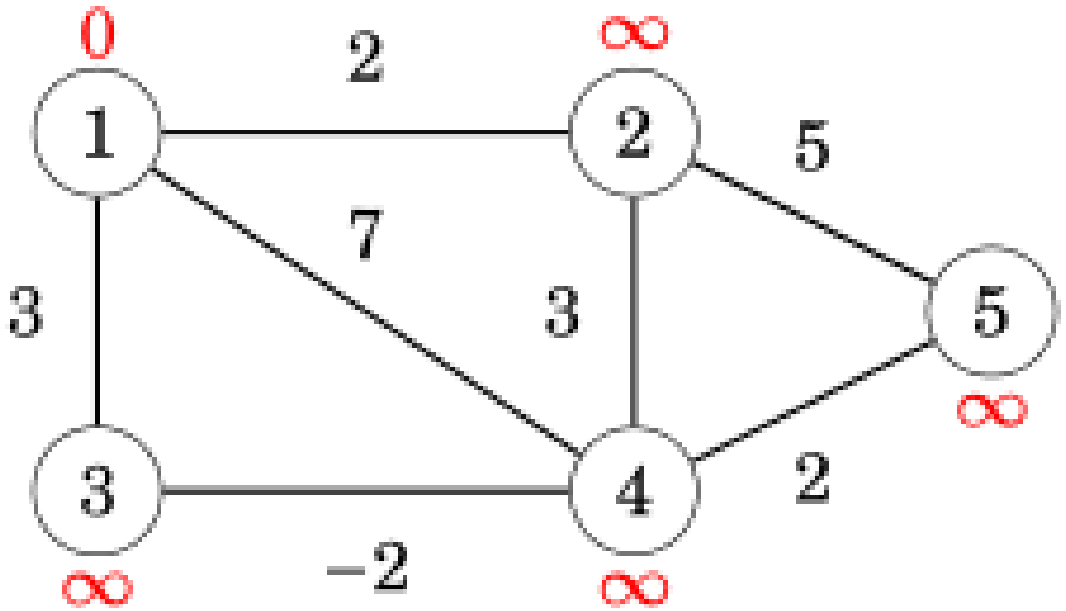
Ví dụ: Cho đồ thị sau, duyệt đồ thị theo Ford Bellman



$$A = \begin{bmatrix} \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & 3 & 3 & 8 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{bmatrix}$$

❖ Thuật toán Ford - Bellman

Ví dụ: Cho đồ thị sau, lập ma trận trọng số và duyệt đồ thị theo Ford Bellman



❖ Thuật toán Ford - Bellman

❑ Ứng dụng thuật toán:

✓ Một ứng dụng thuật toán Bellman-Ford được dùng trong các giao thức định tuyến vector khoảng cách, chẳng hạn giao thức RIP (*Routing Information Protocol*). Đây là ứng dụng phân tán vì nó liên quan đến các nút mạng (các thiết bị định tuyến) trong một hệ thống máy chủ (*autonomous system*), ví dụ một tập các mạng IP thuộc sở hữu của một nhà cung cấp dịch vụ Internet (ISP).

✓ Thuật toán gồm các bước sau:

1.Mỗi nút tính khoảng cách giữa nó và tất cả các nút khác trong hệ thống máy chủ và lưu trữ thông tin này trong một bảng.

2.Mỗi nút gửi bảng thông tin của mình cho tất cả các nút lân cận.

3.Khi một nút nhận được các bảng thông tin từ các nút lân cận, nó tính các tuyến đường ngắn nhất tới tất cả các nút khác và cập nhật bảng thông tin của chính mình.

✓ Nhược điểm chính của thuật toán Bellman-Ford trong cấu hình này là

•Không nhân rộng tốt

•Các thay đổi của tô-pô mạng không được ghi nhận nhanh do các cập nhật được lan truyền theo từng nút một.

•Đếm dần đến vô cùng (nếu liên kết hỏng hoặc nút mạng hỏng làm cho một nút bị tách khỏi một tập các nút khác, các nút này vẫn sẽ tiếp tục ước tính khoảng cách tới nút đó và tăng dần giá trị tính được, trong khi đó còn có thể xảy ra việc định tuyến thành vòng tròn)

Bài tập 1

- Cho đồ thị gồm 7 đỉnh cho bởi ma trận trọng số.
- Dùng thuật toán **Dijkstra**, tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh 7. Yêu cầu chỉ rõ những kết quả trung gian trong quá trình thực hiện thuật toán.

00	11	65	17	65	65	65
65	00	12	65	65	10	16
65	65	00	13	14	65	19
65	65	65	00	65	65	18
65	65	65	65	00	65	15
65	13	18	65	65	00	10
65	65	65	65	65	65	00

Bài tập 2

- Cho đồ thị có hướng có trọng số được biểu diễn dưới dạng ma trận trọng số như dưới đây. Hãy thực hiện:
- Dùng thuật toán **Bellman-Ford**, tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại của đồ thị? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

∞	7	∞	9	4	∞	∞	∞	∞
∞	∞	3	∞	-4	∞	∞	∞	∞
∞	∞	∞	∞	-8	∞	-3	∞	∞
∞	∞	∞	∞	∞	∞	∞	-4	∞
∞	∞	∞	5	∞	2	∞	3	∞
∞	∞	∞	∞	∞	∞	5	∞	2
∞	∞	∞	∞	∞	∞	∞	∞	-7
∞	∞	∞	∞	∞	-2	∞	∞	-3
∞	∞	∞	∞	∞	∞	∞	∞	∞

Bài tập 3

- Dùng thuật toán **Floyd**, tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị sau: