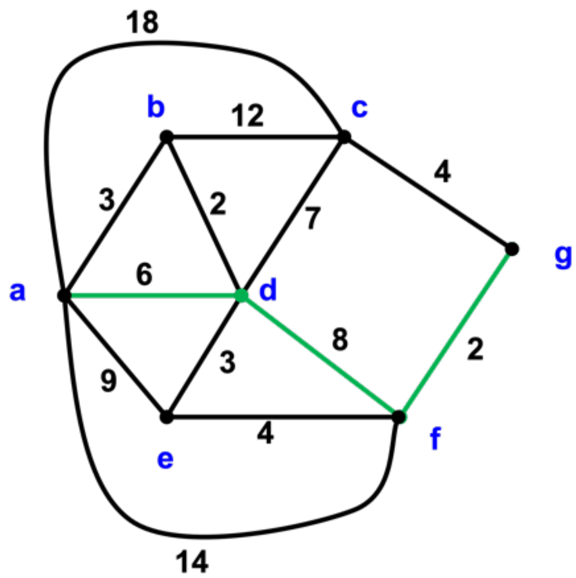
BÀI 4
TÌM KIẾM ĐƯỜNG ĐI NGẮN NHẤT

Mục tiêu

- Hiểu ý tưởng các thuật toán tìm đường đi ngắn nhất trên đồ thị: **Dijkstra**, **Bellman-Ford** và **Floyd-Warshall**;
- Cài đặt được các các thuật toán trên máy tính;
- Vận dụng giải các bài toán liên quan.

1. Một số khái niệm



Hình 1. Đồ thị có trọng số

Độ dài đường đi $\boldsymbol{P(s,t)} = \{v_1, v_2, v_3, \dots, v_k\}$ từ đỉnh s đến đỉnh t trong đồ thị có trọng số $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, kí hiệu là $\boldsymbol{W(P)}$, là tổng trọng số của các cạnh trên đường đi đó.

$$W(P) = \sum_{i=1}^{n-1} w(v_i, v_{i+1})$$

Ví dụ: Trong Hình 1, một đường đi từ đỉnh a đến đỉnh g là $P(\boldsymbol{a}, \boldsymbol{g}) = \{a, d, f, g\}$ và $W(P) = 6 + 8 + 2 = 16$ (*tổng trọng số của các cạnh ad, df và fg*).

Đường đi ngắn nhất từ đỉnh s đến đỉnh t trong đồ thị có trọng số $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, là đường đi có độ dài ngắn nhất trong tất cả các đường đi từ đỉnh s đến đỉnh t.

2. Thuật toán Dijkstra

Thuật toán **Dijkstra** được sử dụng để **tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại** (hoặc một đỉnh đến cụ thể) trong một **đồ thị có trọng số không âm**.

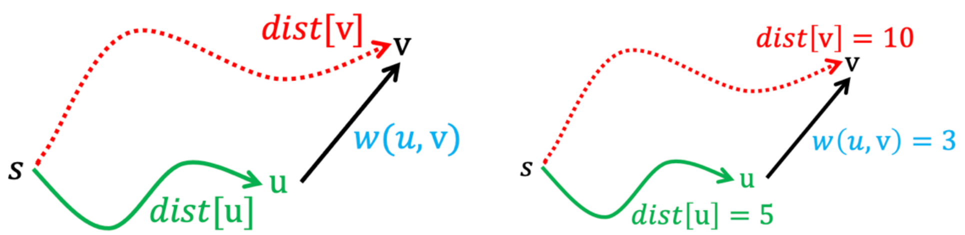
2.1. Ý tưởng của thuật toán Dijkstra

- Là một thuật toán **tham lam** (*greedy*);
- Thuật toán **lưu trữ độ dài đường đi ngắn nhất (*tạm thời*) từ đỉnh nguồn *s*** đến các đỉnh còn lại (*giả sử dùng mảng **dist***). Khi bắt đầu thuật toán:

$$dist[u] = \begin{cases} 0, nếu\ u = s \\ \infty, nếu\ u \neq s \end{cases}$$

- Lặp để tối ưu đường đi:
  - Với mỗi lần lặp, thuật toán **chọn một đỉnh *u* chưa xét và có $\boldsymbol{dist[u]}$ là nhỏ nhất** (*tham lam*) để tối ưu độ dài đường đi từ đỉnh *s* đến các đỉnh *v* kề *u* theo nguyên tắc:

Nếu **dist[v] > dist[u] + w(u,v)** *thì* **dist[v] = dist[u] + w(u,v)**



Hình 2. Cập nhật đường đi ngắn nhất đến đỉnh v thông qua đỉnh u

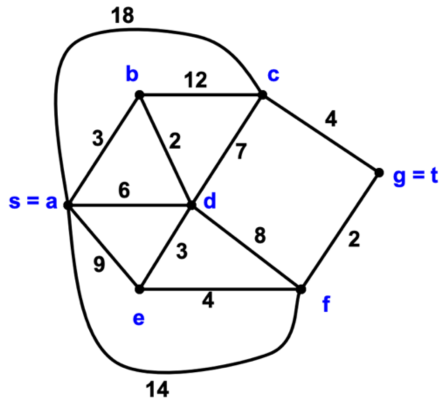
Ví dụ:

  - Khi đỉnh *u* được chọn có $dist[u] = 5$;
  - *v* là một đỉnh kề *u* với $w(u, v) = 3$ và hiện có $dist[v] = 10$

  ➔ Đường đi hiện tại từ đỉnh *s* đến đỉnh *v* dài hơn **đường đi mới đến đỉnh *v* thông qua đỉnh *u*** (*có độ dài bằng* $dist[u] + w(u, v) = 5 + 3 = 8$) → sẽ cập nhật đường đi đến đỉnh v là đường đi này.

  - Khi đỉnh ***u* được chọn thì** đường đi ngắn nhất đến đỉnh này đã tối ưu → các lần lặp tiếp theo không cần phải xử lý đỉnh này;
  - Thuật toán kết thúc khi đã chọn hết các đỉnh (*hoặc đỉnh đích đến được chọn nếu tìm đường đi ngắn nhất đến một đỉnh cụ thể*).

2.2. Minh họa thuật toán Dijkstra

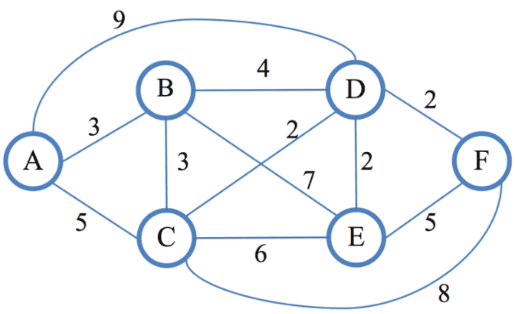


Minh họa quá trình tìm **đường đi ngắn nhất từ đỉnh a đến đỉnh g** bằng thuật toán **Dijkstra**:

Lần lặp	**a**		**b**		**c**		**d**		**e**		**f**		**g**	
	dist	pre	dist	pre	dist	pre	dist	pre	dist	pre	dist	pre	dist	pre
0	0*	-1	∞	-1	∞	-1	∞	-1	∞	-1	∞	-1	∞	-1
1			3*	a	18	a	6	a	9	a	14	a	∞	-1
2					15	b	5*	b	9	a	14	a	∞	-1
3					12	d			8*	d	13	d	∞	-1
4					12*	d					12	e	∞	-1
5											12*	e	16	c
6													14	f
Kết quả	0	-1	3	a	12	d	5	b	8	d	12	e	14	f

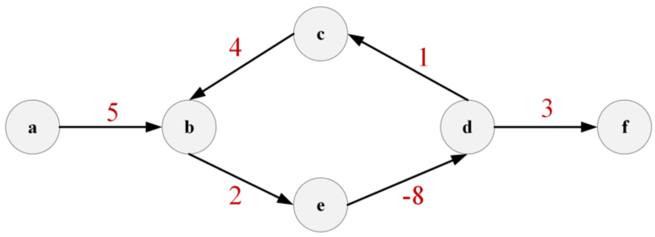
→ Đường đi ngắn nhất từ đỉnh a đến đỉnh g là {a, b, d, e, f, g} và có độ dài bằng 14.

Bài tập 1. Cho đồ thị như hình vẽ bên dưới. Minh họa quá trình tìm **đường đi ngắn nhất từ đỉnh a đến đỉnh f** bằng thuật toán **Dijkstra**.



2.3. Điều kiện hoạt động của thuật toán Dijkstra

- Đồ thị không chứa cạnh có trọng số âm:
    - Thuật toán Dijkstra hoạt động theo giả định khi **chọn một đỉnh thì đường đi ngắn nhất đến đỉnh đó đã tối ưu** và không thay đổi nữa;
    - Độ dài của một **đường đi đi qua chu trình âm có thể giảm vô hạn** nếu liên tục đi qua chu trình âm đó.



Hình 3. Đồ thị có trọng số âm và chu trình âm

2.4. Cài đặt thuật toán Dijkstra

Input: đồ thị G(V,E), đỉnh nguồn *s*

Output: đường đi ngắn nhất từ đỉnh *s* đến tất cả các đỉnh của đồ thị

```
Dijkstra(s)
   //Khởi tạo
   dist[v] ← INF (với mọi v ∈ V)
   pre[v] ← -1
   dist[s] ← 0
   Lặp cho đến khi không có đỉnh nào được chọn(1)
      Chọn u ← đỉnh có dist nhỏ nhất trong các đỉnh chưa xét(2)
      Đánh dấu đã xét đỉnh u (dist[u] đã tối ưu)
      Xét các đỉnh v kề u mà chưa đánh dấu
         Nếu dist[v] chưa tối ưu thì tối ưu dist[v](3)
```

(1) Trường hợp tìm đường đi ngắn nhất đến một đỉnh cụ thể thì dừng thuật toán khi đỉnh đó được chọn;

(2) Các chọn đỉnh u chưa xét và có dist nhỏ nhất

- **Cách 1:** sử dụng vòng lặp duyệt tất cả các đỉnh chưa viếng thăm để tìm đỉnh u có dist nhỏ nhất.

```
minDist ← INF
minVertex ← -1
Xét mọi đỉnh v ∈ V
    Nếu v chưa xét và có dist < minDist
        minDist ← dist[v]
        minVertex ← v
```

- **Cách 2:** Sử dụng *"hàng đợi ưu tiên"* (*priority_queue*) để lưu các đỉnh chưa viếng thăm theo thứ tự dựa trên dist của đỉnh đó:
  - CTDL: **PriorityQueue<TElement,TPriority>** *(.NET 6.0 trở lên)*
    Ví dụ: Khai báo **PriorityQueue** để chứa các đỉnh u (**TElement** kiểu **char**), ưu tiên đỉnh có dist[u] nhỏ nhất (**Tpriority** kiểu **int**)

```
PriorityQueue<char, int> pq = new PriorityQueue<char, int>();
```

Thêm (v, dist[v])	pq
`pq.Enqueue('a', 5)`	`('a', 5)`
`pq.Enqueue('c', 2)`	`('c', 2), ('a', 5)`
`pq.Enqueue('a', 3)`	`('c', 2), ('a', 3), ('a', 5)`

    Lưu ý: Một phần tử có thể được thêm **PriorityQueue** nhiều lần nên mỗi khi lấy một đỉnh từ **PriorityQueue** nên kiểm tra đỉnh đó đã xử lý chưa.
  - CTDL: **SortedSet<T>**
    Ví dụ: Khai báo **SortedSet** để chứa các **Tuple<int dist, char u>** để lưu các cặp giá trị ưu tiên dựa vào có dist

```
SortedSet<(int dist, char u)> pq = new SortedSet<(int, char)>();
```

Thêm (dist, v)	pq
`pq.Add((5, 'a'))`	`(5, 'a')`
`pq.Add((2, 'c'))`	`(2, 'c'), (5, 'a')`
`pq.Add((3, 'a'))`	`(2, 'c'), (3, 'a'),(5, 'a')`

    Lưu ý: **SortedSet** cho phép thêm và xóa phần tử nên có thể cập nhật lại dist của đỉnh trong **SortedSet** để đảm bảo mỗi đỉnh chỉ xuất hiện một lần.

(3) Điều kiện cần cập nhật dist[v]

```
    Nếu dist[v] > dist[u] + w(u,v)
       dist[v] = dist[u] + w(u,v)
       pre[v] = u   //Lưu vết đường đi
```

2.5. Độ phức tạp của thuật toán Dijkstra:

- Sử dụng vòng lặp để tìm đỉnh min: **O(**$|V|^2 + |E|$**)**.
- Sử dụng cấu trúc hàng đợi ưu tiên: **O(**$(|V|+|E|).\log(|V|)$**)**.

3. Thuật toán Bellman-Ford

Thuật toán **Bellman-Ford** được sử dụng để **tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại** (hoặc một đỉnh đến cụ thể) trong một **đồ thị có trọng số** (có thể là trọng số âm). Thuật toán **Bellman-Ford** có thời gian chạy chậm hơn Dijkstra nhưng lại có thể xử lý đồ thị có cạnh trọng số âm và phát hiện chu trình âm.

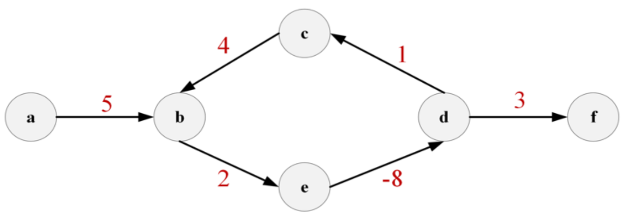
3.1. Ý tưởng của thuật toán Bellman-Ford

- **Nếu đồ thị không có chu trình âm**, đường đi ngắn nhất từ một đỉnh s đến một đỉnh t chỉ đi qua mỗi đỉnh tối đa một lần → **Đồ thị có n đỉnh thì đường đi ngắn nhất có tối đa $(n-1)$ cạnh**.
- Đầu tiên, **thực hiện $(n-1)$ lần lặp**. **Mỗi lần lặp duyệt tất cả các cạnh** (u, v) có trọng số $w(u, v)$ để **kiểm tra có thể thêm cạnh (u, v) vào đường đi ngắn nhất** từ đỉnh s đến đỉnh v hay không với điều kiện:

Nếu **dist[v] > dist[u] + w(u,v)** *thì* **dist[v] = dist[u] + w(u,v)**

- Nếu sau $(n-1)$ lần lặp, dist của đỉnh nào vẫn tiếp tục giảm ở các lần lặp tiếp theo thì đường đi đến đỉnh đó đi qua chu trình âm:
  - Nếu cần xác định một đỉnh có đường đi ngắn nhất đi qua chu trình âm thì chỉ cần thêm 1 lần lặp;
  - Nếu cần xác định tất cả các đỉnh có đường đi ngắn nhất đi qua chu trình âm thì chỉ lặp thêm n lần nữa.

3.2. Minh họa thuật toán Bellman-Ford



Minh họa quá trình tìm **đường đi ngắn nhất từ đỉnh a đến đỉnh f** bằng thuật toán **Bellman-Ford**:

- **Bước 1:** Xét tất cả các cạnh (n-1) lần để thêm cạnh vào đường đi ngắn nhất:

Lần lặp	**Xét cạnh**	**a**		**b**		**c**		**d**		**e**		**f**	
		dist	pre	dist	pre	dist	pre	dist	pre	dist	pre	dist	pre
Khởi tạo		0	-1	∞	-1	∞	-1	∞	-1	∞	-1	∞	-1
1	(a, b, 5)			5	a								
	(b, e, 2)									7	b		
	(c, b, 4)			-	-								
	(d, c, 1)					-	-						
	(d, f, 3)											-	-
	(e, d, -8)							-1	e				
2	(a, b, 5)			-	-								
	(b, e, 2)									-	-		
	(c, b, 4)			-	-								
	(d, c, 1)					0	d						
	(d, f, 3)											2	d

Lần lặp	**Xét cạnh**	**a**		**b**		**c**		**d**		**e**		**f**	
		dist	pre	dist	pre	dist	pre	dist	pre	dist	pre	dist	pre
	(e, d, -8)							-	-				
3	(a, b, 5)			-	-								
	(b, e, 2)									-	-		
	(c, b, 4)			4	c								
	(d, c, 1)					-	-						
	(d, f, 3)											-	-
	(e, d, -8)							-	-				
4	(a, b, 5)			-	-								
	(b, e, 2)									6	b		
	(c, b, 4)			-	-								
	(d, c, 1)					-	-						
	(d, f, 3)											-	-
	(e, d, -8)							-2	e				
5	(a, b, 5)			-	-								
	(b, e, 2)									-	-		
	(c, b, 4)			-	-								
	(d, c, 1)					-1	d						
	(d, f, 3)											-1	d
	(e, d, -8)							-	-				
Kết quả		0	-1	4	c	-1	d	-2	e	6	b	-1	d

- **Bước 2:** Xét tất cả các cạnh thêm 1 lần để kiểm tra chu trình âm:

Lần lặp	**Xét cạnh**	**a**		**b**		**c**		**d**		**e**		**f**	
		dist	pre	dist	pre	dist	pre	dist	pre	dist	pre	dist	pre
Kết quả (n-1) lần lặp		0	-1	4	c	-1	d	-2	e	6	b	-1	d
6	(a, b, 5)			-	-								
	(b, e, 2)									-	-		
	(c, b, 4)			-∞	c								
	(d, c, 1)					-	-						
	(d, f, 3)											-	-
	(e, d, -8)							-	-				
Kết quả		0	-1	-∞	c	-1	d	-2	e	6	b	-1	d

  - Trong lần lặp này, **nếu đỉnh u nào mà $dist[u]$ vẫn có thể tối ưu** được thì **đường đi ngắn nhất đến đỉnh đó đi qua chu trình âm**.
  - Nếu muốn **xác định tất cả các đỉnh** mà đường đi ngắn nhất đến đỉnh đó đi qua chu trình âm thì ở bước 2 phải **xét tất cả các cạnh đủ n lần**.

Lần lặp	Xét cạnh	a		b		c		d		e		f	
		dist	pre	dist	pre	dist	pre	dist	pre	dist	pre	dist	pre
Kết quả lần lặp 6		0	-1	$-\infty$	c	-1	d	-2	e	6	b	-1	d
7	(a, b, 5)			-	-								
	(b, e, 2)									$-\infty$	b		
	(c, b, 4)			-	-								
	(d, c, 1)					-	-						
	(d, f, 3)											-	-
	(e, d, -8)							$-\infty$	e				
8	(a, b, 5)			-	-								
	(b, e, 2)									-	-		
	(c, b, 4)			-	-								
	(d, c, 1)					$-\infty$	d						
	(d, f, 3)											$-\infty$	d
	(e, d, -8)							-	-				
9, 10, 11	(a, b, 5)			-	-								
	(b, e, 2)									-	-		
	(c, b, 4)			-	-								
	(d, c, 1)					-	-						
	(d, f, 3)											-	-
	(e, d, -8)							-	-				
Kết quả		0	-1	$-\infty$	c	$-\infty$	d	$-\infty$	e	$-\infty$	b	$-\infty$	d

- **Bước 3:** Xác định chu trình âm:
    - Bắt đầu từ đỉnh u có $dist[u] = -\infty$, truy vết theo mảng pre đủ n lần. Mục đích của bước này là để **đảm bảo u chắc chắn thuộc chu trình âm** vì nếu $dist[u] = -\infty$ thì đường đi ngắn nhất đến đỉnh u đi qua chu trình âm nhưng đỉnh u có thể không thuộc chu trình âm;
    - Sau khi xác định được đỉnh u chắc chắn thuộc chu trình âm thì tiếp tục truy vết theo mảng pre cho đến khi gặp lại chính nó. Đảo ngược kết quả sẽ được một chu trình âm.

  Ví dụ: Đỉnh f có $dist[f] = -\infty$ (*nhưng f không thuộc chu trình âm*)
    - Xuất phát từ đỉnh f, truy vết theo mảng pre đủ n lần

      Mảng pre:

u	a	b	c	d	e	f
pre[u]	-1	c	d	e	b	d

Truy vết theo mảng pre đủ n lần

Lần lặp	0	1	2	3	4	5	6
u	f	d	e	b	c	d	e

➔ Đỉnh e chắc chắn thuộc chu trình âm

- Từ đỉnh *e*, truy vết theo mảng *pre* cho đến khi gặp lại chính nó: e ← b ← c ← d ← (e). Đảo ngược kết quả ta được một chu trình âm: e → d → c → b → e.

3.3. Cài đặt thuật toán Bellman-Ford

Input: đồ thị G(V,E), đỉnh nguồn *s*

Output: đường đi ngắn nhất từ đỉnh *s* đến tất cả các đỉnh của đồ thị

```
BellmanFord(s)
   //Khởi tạo
   dist[v] ← INF (với mọi v ∈ V)
   pre[v] ← -1
   dist[s] ← 0
   //Thêm cạnh vào các đường đi ngắn nhất
   Lặp (n-1)
      Xét các cạnh (u,v)
         Nếu dist[v] > dist[v] + w(u,v) thì
            dist[v] = dist[v] + w(u,v)
            pre[v] = u
   //Kiểm tra chu trình âm
   Xét các cạnh (u,v)
      Nếu dist[v] > dist[v] + w(u,v) thì dist[v] = -INF
```

3.4. Độ phức tạp của thuật toán Bellman-Ford: **O(|V|.|E|)**

4. Thuật toán Floyd-Warshall

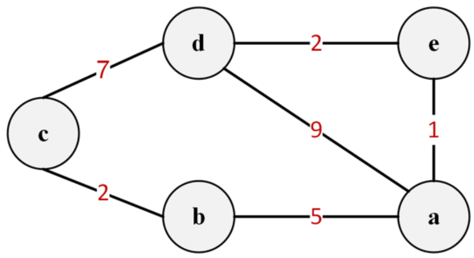
Thuật toán **Floyd-Warshall** được sử dụng để **tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh** trong một đồ thị có trọng số (*có thể có cạnh âm nhưng không có chu trình âm*)

4.1. Ý tưởng của thuật toán Floyd-Warshall

- Là một thuật toán **quy hoạch động** (*dynamic programming*);
- Ban đầu, khởi tạo một ma trận **dist** kích thước *n x n* (*n* là số đỉnh) để lưu khoảng cách ngắn nhất giữa các cặp đỉnh.
- Lần lượt chọn từng đỉnh của đồ thị làm đỉnh trung gian (*quy ước là đỉnh k*). Với mỗi cặp đỉnh phân biệt và không trùng với đỉnh trung gian (*quy ước là đỉnh u và v*), tối ưu đường đi ngắn nhất từ đỉnh *u* đến *v* theo công thức:

Nếu **dist[u,v] > dist[u,k] + dist[k,v]** *thì* **dist[u,v] = dist[u,k] + dist[k,v]**

4.2. Minh họa thuật toán Floyd-Warshall



- **Bước 1**: Khởi tạo ma trận **dist**

dist	a	b	c	d	e
a	0	5	∞	9	1
b	5	0	2	∞	∞
c	∞	2	0	7	∞
d	9	∞	7	0	2
e	1	∞	∞	2	0

pre	a	b	c	d	e
a	a	a	a	a	a
b	b	b	b	b	b
c	c	c	c	c	c
d	d	d	d	d	d
e	e	e	e	e	e

- **Bước 2:** Xét từng đỉnh trung gian k
  - **k = 'a'**

u	k	v	dist[u,v]	dist[u,k] + dist[k,v]	Cập nhật dist[u,v]	Cập nhật pre[u,v] = pre[k,v]	Ghi chú
b	a	c	2	5 + ∞	-	-	(1)
b	a	d	∞	5 + 9	14	pre[b, d] = a	(2)
b	a	e	∞	5 + 1	6	pre[b, e] = a	(3)
c	a	b	2	∞ + 5	-	-	(1)
c	a	d	7	∞ + 9	-	-	(4)
c	a	e	∞	∞ + 1	-	-	(5)
d	a	b	∞	9 + 5	14	pre[d, b] = a	(2)
d	a	c	7	9 + ∞	-	-	(4)
d	a	e	2	9 + 1	-	-	(6)
e	a	b	∞	1 + 5	6	pre[e, b] = a	(3)
e	a	c	∞	1 + ∞	-	-	(5)
e	a	d	2	1 + 9	-	-	(6)

dist	a	b	c	d	e
a	0	5	∞	9	1
b	5	0	2	14	6
c	∞	2	0	7	∞
d	9	14	7	0	2
e	1	6	∞	2	0

pre	a	b	c	d	e
a	a	a	a	a	a
b	b	b	b	a	a
c	c	c	c	c	c
d	d	a	d	d	d
e	e	a	e	e	e

- **k = 'b'**

u	k	v	dist[u,v]	dist[u,k] + dist[k,v]	Cập nhật dist[u,v]	Cập nhật pre[u,v] = pre[v,k]	Ghi chú
a	b	c	∞	5 + 2	7	pre[a, c] = b	(1)
a	b	d	9	5 + 14	-	-	(2)
a	b	e	1	5 + 6	-	-	(3)
c	b	a	∞	2 + 5	7	pre[c, a] = b	(1)
c	b	d	7	2 + 14	-	-	(4)
c	b	e	∞	2 + 6	8	pre[c, e] = a	(5)
d	b	a	9	14 + 5	-	-	(2)
d	b	c	7	14 + 2	-	-	(4)
d	b	e	2	14 + 6	-	-	(6)
e	b	a	1	6 + 5	-	-	(3)
e	b	c	∞	6 + 2	8	pre[e, c] = b	(5)
e	b	d	2	6 + 14	-	-	(6)

dist	**a**	**b**	**c**	**d**	**e**
a	0	5	7	9	1
b	5	0	2	14	6
c	7	2	0	7	8
d	9	14	7	0	2
e	1	6	8	2	0

pre	**a**	**b**	**c**	**d**	**e**
a	a	a	b	a	a
b	b	b	b	a	a
c	b	c	c	c	a
d	d	a	d	d	d
e	e	a	b	e	e

- **k = 'c'**

dist	**a**	**b**	**c**	**d**	**e**
a	0	5	7	9	1
b	5	0	2	9	6
c	7	2	0	7	8
d	9	9	7	0	2
e	1	6	8	2	0

pre	**a**	**b**	**c**	**d**	**e**
a	a	a	b	a	a
b	b	b	b	c	a
c	b	c	c	c	a
d	d	c	d	d	d
e	e	a	b	e	e

- **k = 'd'**

dist	**a**	**b**	**c**	**d**	**e**
a	0	5	7	9	1
b	5	0	2	9	6
c	7	2	0	7	8
d	9	9	7	0	2
e	1	6	8	2	0

pre	**a**	**b**	**c**	**d**	**e**
a	a	a	b	a	a
b	b	b	b	c	a
c	b	c	c	c	a
d	d	c	d	d	d
e	e	a	b	e	e

- **k = 'e'**

dist	a	b	c	d	e
a	0	5	7	3	**1**
b	5	0	2	8	**6**
c	7	2	0	7	**8**
d	3	8	7	0	**2**
e	1	6	8	2	**0**

pre	a	b	c	d	e
a	a	a	b	d	a
b	b	b	b	d	a
c	b	c	c	c	a
d	d	a	d	d	d
e	e	a	b	e	e

- **Truy vết đường đi:** bắt đầu từ đỉnh $u = t$, cập nhật $u = pre[s,u]$ cho đến khi gặp đỉnh s.

  Ví dụ: Xác định đường đi ngắn nhất từ đỉnh ***c*** đến đỉnh ***e***

  - Truy vết đường đi: s = 'c', t = 'e'

u	**pre[s,u]** (*pre['c', u]*)
'e'	'a'
'a'	'b'
'b'	'c' (= 's' → *Dừng)*

→ Đường đi ngắn nhất từ đỉnh ***c*** đến đỉnh ***e*** là ***{c, b, a, e}*** và có độ dài là $\boldsymbol{dist[c,e] = 8}$

4.3. Cài đặt thuật toán Floyd-Warshall

Input: đồ thị G(V,E)

Output: ma trận biểu diễn đường đi ngắn nhất giữa các cặp đỉnh

```
FloydWarshall()
   //Khởi tạo
   dist[u, v] ← w(u,v)(nếu (u,v) ∈ E)
   dist[u, v] ← INF (nếu (u,v) ∉ E)
   pre[u, v] ← u

   //Lặp
   Với mỗi đỉnh k ∈ V
      Xét các cặp đỉnh (u,v) ∈ V sao cho u ≠ v ≠ k
         Nếu dist[u,v] > dist[u,k] + dist[k,v] thì
             dist[u,v] = dist[u,k] + dist[k,v]
             pre[u,v] = pre[k,v]
```

4.4. Độ phức tạp của thuật toán Floyd-Warshall: **O(|V|³)**

--- HẾT ---