

# 4주차 : 최단 경로 알고리즘

다익스트라, 플로이드 와샬, 벨만 포드

# ■ 오늘 할 내용은?

---

최단 경로 알고리즘에 대해 알아보시다

1. 다익스트라 (Dijkstra)
2. 플로이드 와샬 (Floyd Warshall)
3. 벨만 포드 (Bellman Ford)

# Dijkstra 다익스트라

하나의 시작 정점에서 **다른 모든 정점**으로 가는 최단 경로를 구하는 알고리즘

1. 출발 정점에서 출발 정점까지의 최단 거리를 0으로 확정
2. 최단 거리가 확정된 정점과 **가장 가까운 정점**을 찾아 그 지점까지 최단 거리를 **확정**

※ **priority\_queue**를 이용해 가장 가까운 정점을 찾자!

3. priority\_queue가 빌 때까지 반복

- **greedy**로 최단 거리를 확정 → **음수 가중치**의 간선이 있으면 **사용 불가!**
- 시간 복잡도  $O(E \log V)$  (E: 간선 개수 / V: 정점 개수)

# priority\_queue 문제풀이

---

다익스트라에 앞서  
priority\_queue 사용법을  
익히고 갑시다!

**백준 1927**  
**최소 힙**

## 1927 최소 힙

---

문제: 가지고 있는 값 중 가장 작은 값을 출력하는 문제

- priority\_queue: 큐인데 맨 앞(front)에 가장 큰 값이 들어있는 큐
- <queue> 헤더에 정의되어 있음
- 선언 방법: `std::priority_queue<자료형> 이름;`
- priority\_queue에 - (마이너스)를 붙여 가장 작은 값이 먼저 나올 수 있도록 하자

# 최소 힙 코드

```
1 #include <iostream>
2 #include <queue>
3 using namespace std;
4
5 priority_queue<int> pq;
6
7 int main() {
8     ios_base::sync_with_stdio(false);
9     cin.tie(NULL);
10    cout.tie(NULL);
11    int n; cin >> n;
12    for (int i = 0; i < n; i++) {
13        int num; cin >> num;
14        if (num == 0) {
15            if (pq.empty())
16                cout << "0\n";
17            else {
18                cout << -pq.top() << "\n";
19                pq.pop();
20            }
21        }
22        else {
23            pq.push(-num); // -를 붙여 특별한 정렬없이 사용
24        }
25    }
26
27    return 0;
28 }
```

# Dijkstra 다익스트라

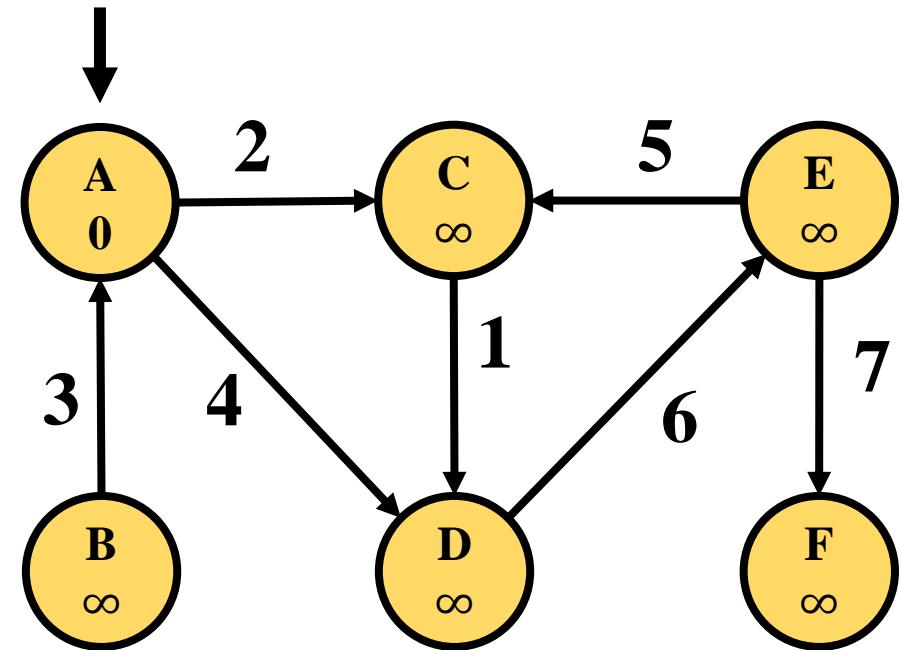
현재 위치: **not started**

priority\_queue

top →

from	to	distance
A	A	0

시작 위치



A의 최단 경로를 0, 나머지 정점의 최단 경로는 INF(무한)으로 설정합니다

# Dijkstra 다익스트라

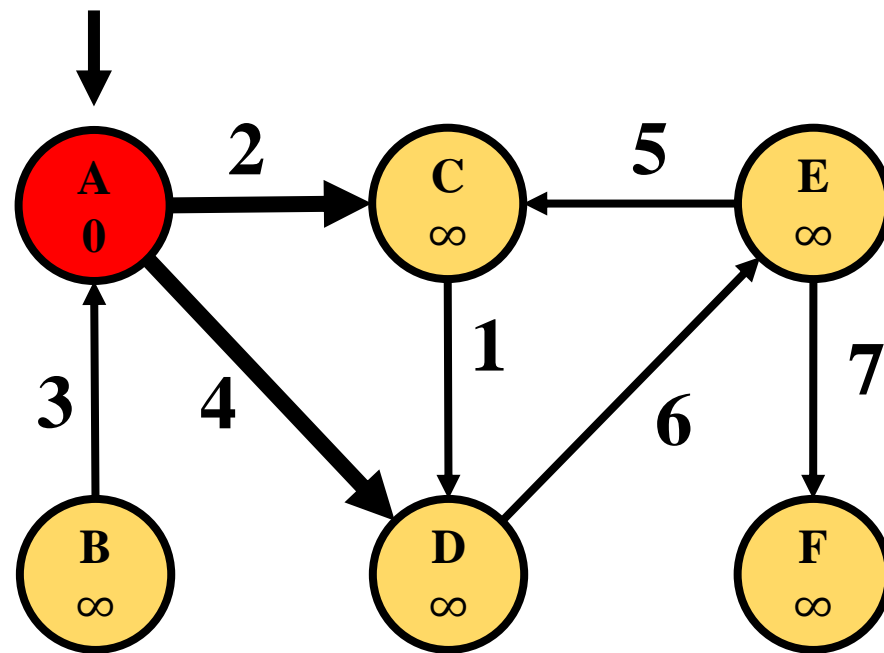
현재 위치: **A**

priority\_queue

from	to	distance
A	C	0 + 2
A	D	0 + 4

top →

시작 위치



A에서 갈 수 있는 곳들의 최단 거리 후보 값을 priority\_queue에 넣어줍니다



# Dijkstra 다익스트라

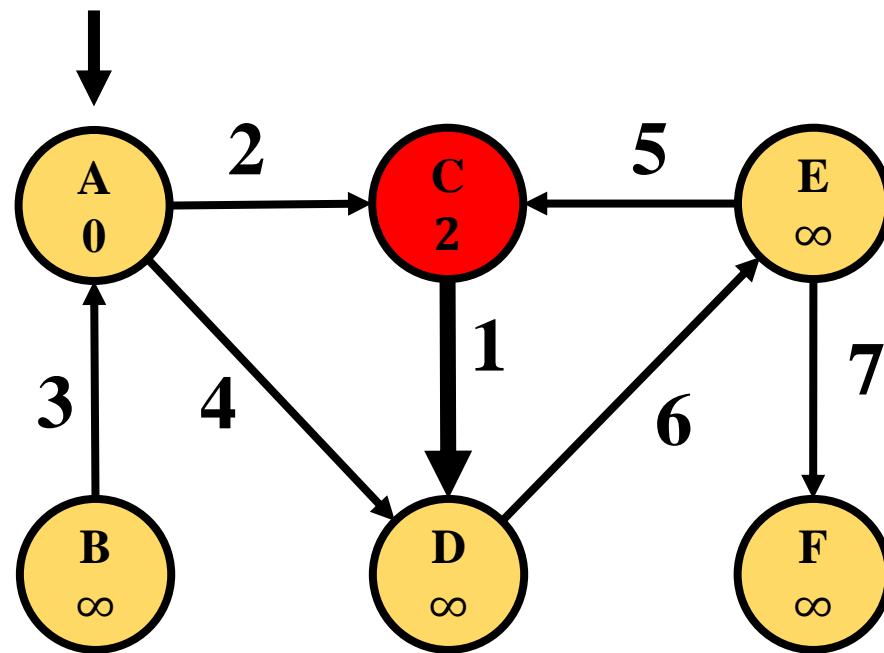
현재 위치: **C**

priority\_queue

from	to	distance
C	D	2 + 1
A	D	0 + 4

top →

시작 위치



C에서 갈 수 있는 곳들의 최단 거리 후보 값을 priority\_queue에 넣어줍니다

# Dijkstra 다익스트라

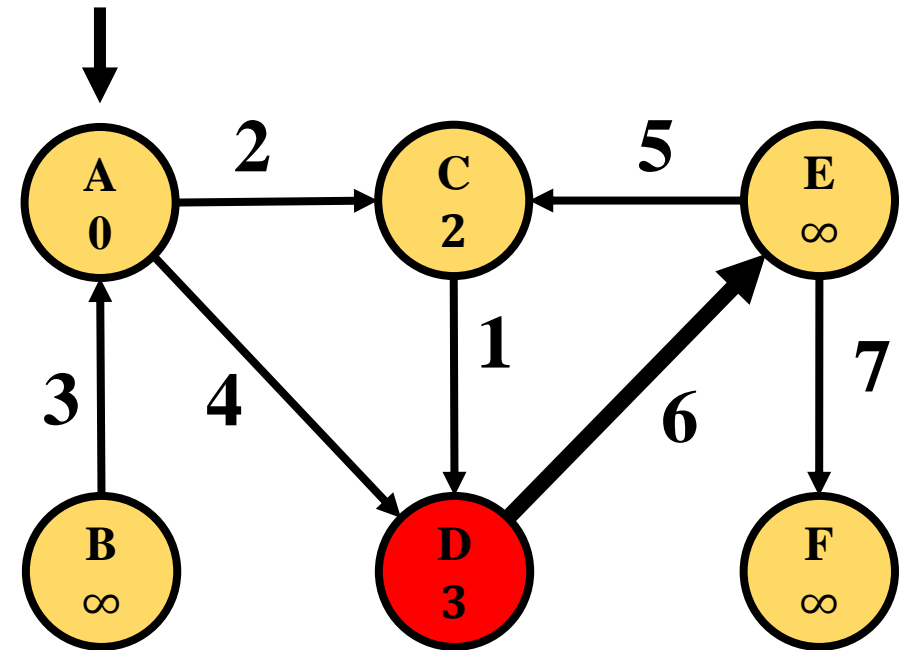
현재 위치: **D**

priority\_queue

from	to	distance
A	D	0 + 4
D	E	3 + 6

top →

시작 위치



D에서 갈 수 있는 곳들의 최단 거리 후보 값을 priority\_queue에 넣어줍니다

# Dijkstra 다익스트라

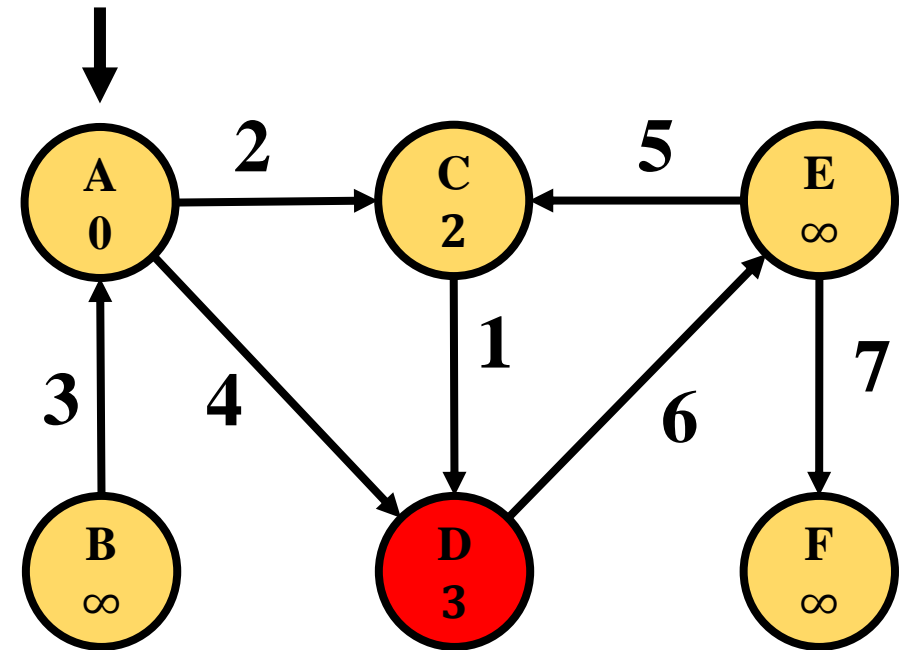
현재 위치: **D**

priority\_queue

from	to	distance
D	E	3 + 6

top →

시작 위치



D까지의 최단 경로는 이미 확정되었으므로 갱신하지 않고 pass!

# Dijkstra 다익스트라

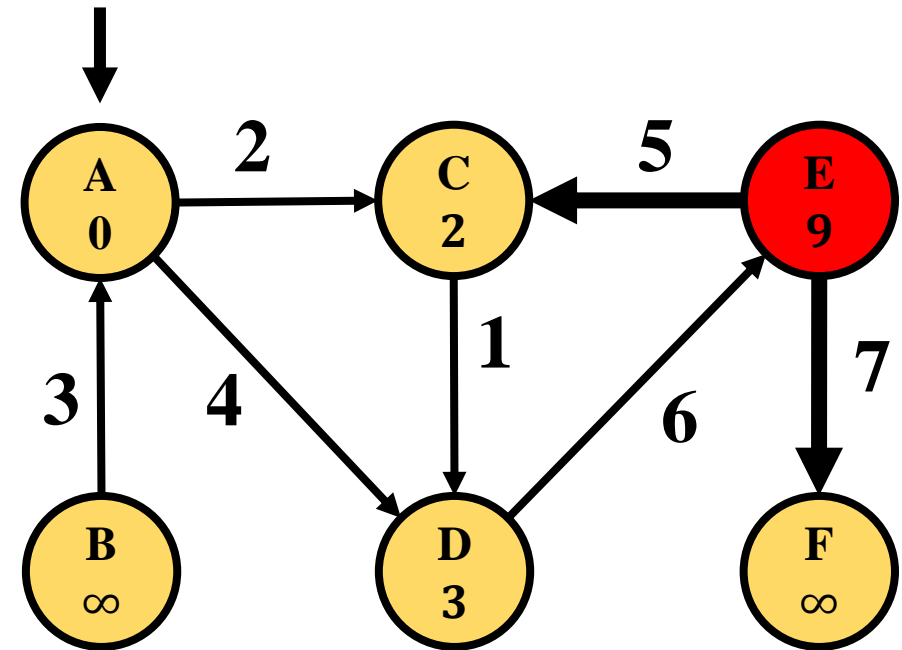
현재 위치: **E**

priority\_queue

from	to	distance
E	C	9 + 5
E	F	9 + 7

top →

시작 위치



E에서 갈 수 있는 곳들의 최단 거리 후보 값을 priority\_queue에 넣어줍니다

# Dijkstra 다익스트라

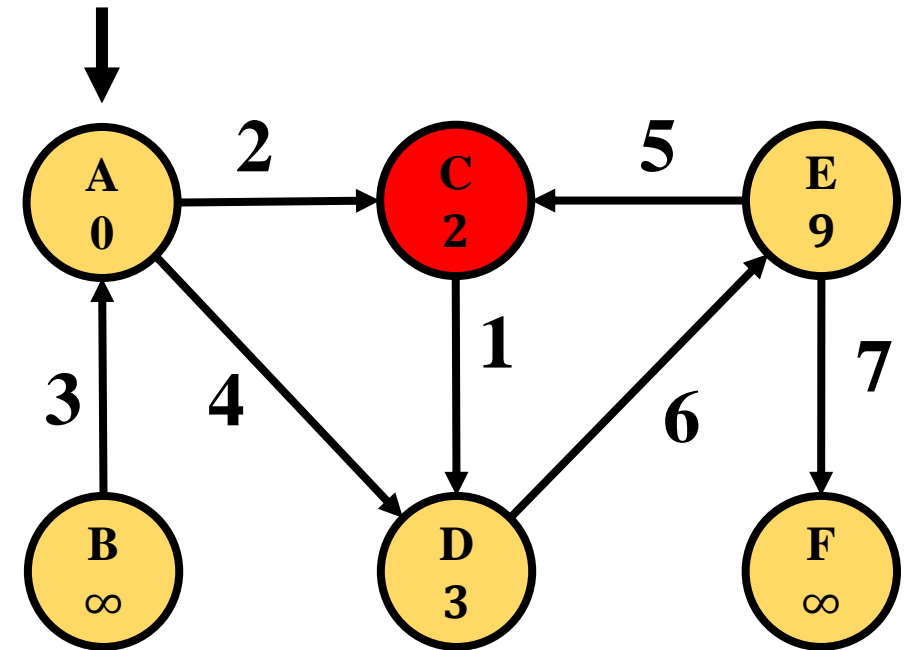
현재 위치: **C**

priority\_queue

from	to	distance
E	F	9 + 7

top →

시작 위치



C까지의 최단 경로는 이미 확정되었으므로 갱신하지 않고 pass!

# Dijkstra 다익스트라

현재 위치: **F**

priority\_queue

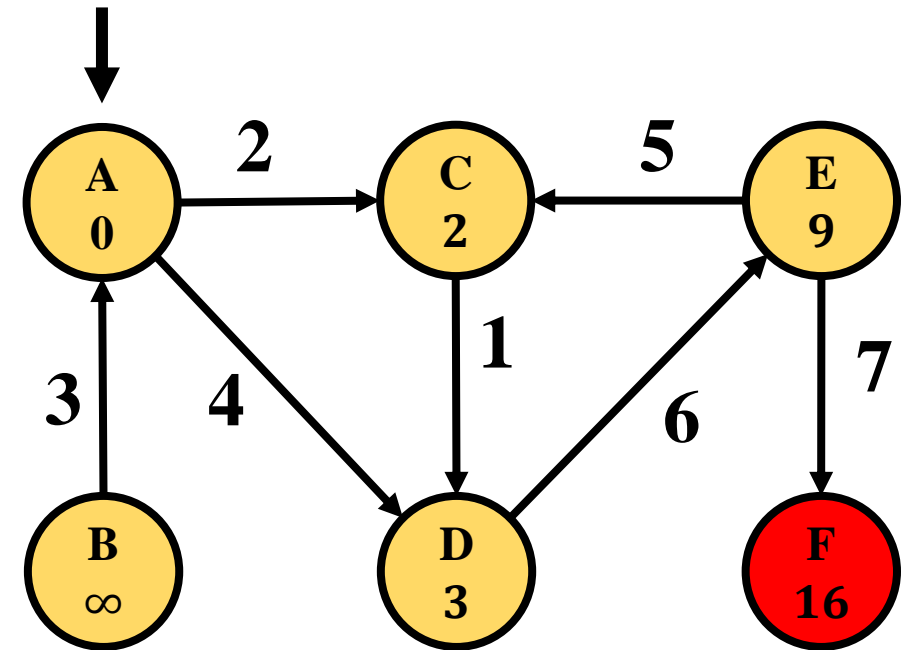
from	to	distance

top →

priority\_queue가 비었으므로 종료!

도달할 수 없으면 종료 후 최단 경로 값이 **INF**(무한)

시작 위치



이제 문제를 풀어봅시다!

# Dijkstra 문제풀이

---

다익스트라 기본 문제!

**백준 1753**  
**최단 경로**

## 1753 최단경로

---

문제: 주어진 방향 그래프에서 최단 경로를 찾는 문제

조건:  $1 \leq V \leq 20,000$        $1 \leq E \leq 300,000$      $1 \leq w \leq 10$

⇒ 다익스트라를 사용하면 충분히 해결할 수 있다



# 최단경로 코드

```
31 int main() {
32     ios_base::sync_with_stdio(false);
33     cin.tie(NULL);
34     cout.tie(NULL);
35     cin >> v >> e;
36     cin >> k;
37     for (int i = 0; i < e; i++) {
38         int a, b, c; cin >> a >> b >> c;
39         edges[a].push_back({ b, c });
40     }
41     dij(k);
42     for (int i = 1; i ≤ v; i++) {
43         if (distan[i] == INF)
44             cout << "INF\n";
45         else cout << distan[i] << "\n";
46     }
47     return 0;
48 }
```

```
1 #include <iostream>
2 #include <queue>
3 #define INF 987654321 // 무한대 정의
4 using namespace std;
5
6 int v, e, k;
7 vector<pair<int, int>> edges[20005]; // 도착정점, 가중치
8 int distan[20005];
9
10 void dij(int start) {
11     for (int i = 1; i ≤ v; i++) {
12         distan[i] = INF;
13     }
14     priority_queue<pair<int, int>> pq; // 경로 값, 도착정점
15     pq.push({ 0, start });
16     while (!pq.empty()) {
17         int con = pq.top().second;
18         int dis = -pq.top().first; // -를 붙여 최소값 가져오기
19         pq.pop();
20         if (distan[con] < INF) continue; // 이미 방문했으면
21         distan[con] = dis;
22         for (pair<int, int> i : edges[con]) {
23             int next = i.first;
24             int weight = i.second + dis;
25             if (distan[next] < INF) continue; // 이미 방문했으면
26             pq.push({ -weight, next });
27         }
28     }
29 }
```

# Floyd Warshall 플로이드 와샬

모든 정점 사이의 최단 경로를 구하는 알고리즘

**distan[start][end]:** start  $\rightarrow$  end의 최단거리를 나타내는 dp배열

1. start  $\rightarrow$  end의 최단거리를 간선의 길이로 초기화
2. 모든 start  $\rightarrow$  end조합에 대해, 특정한 정점 mid를 경유해서 가는  
start  $\rightarrow$  mid  $\rightarrow$  end와 start  $\rightarrow$  end의 최단 거리를 비교
3. 모든 정점 mid에 대해 반복

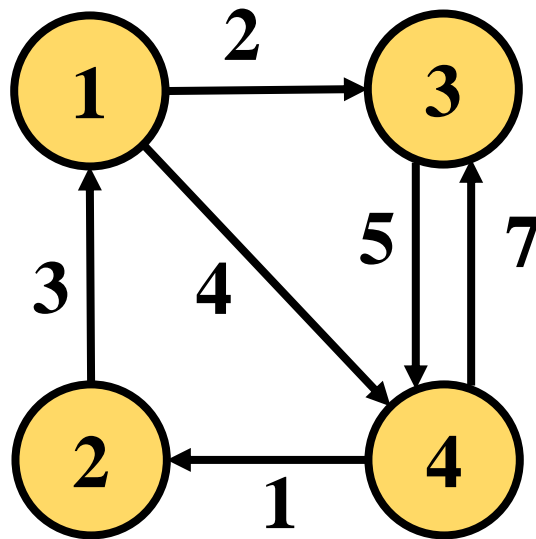
Tip. **mid**에 대한 반복문이 맨 밖에 와야 합니다!

- 음수 가중치의 간선이 있어도 사용가능!
- 시간 복잡도  $O(V^3)$  (V: 정점 개수)

# Floyd Warshall 플로이드 와샬

## Step1. DP 배열 초기화

end start	1	2	3	4
1	0	INF	2	4
2	3	0	INF	INF
3	INF	INF	0	5
4	INF	1	7	0



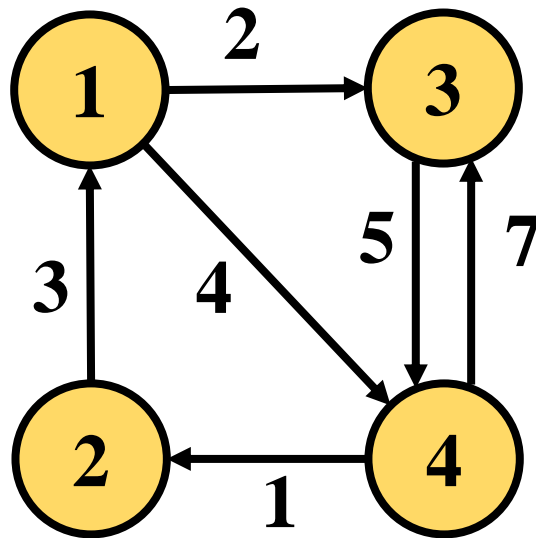
mid: **not started**  
start: **not started**  
end: **not started**

- Start == end이면 0으로 초기화
- 바로 갈 수 있으면 간선의 길이로 초기화
- 바로 갈 수 없으면 INF로 초기화

# Floyd Warshall 플로이드 와샬

Step2. start  $\rightarrow$  mid  $\rightarrow$  end와 start  $\rightarrow$  end의 최단 거리 비교

end start	1	2	3	4
1	0	INF	2	4
2	3	0	INF	INF
3	INF	INF	0	5
4	INF	1	7	0



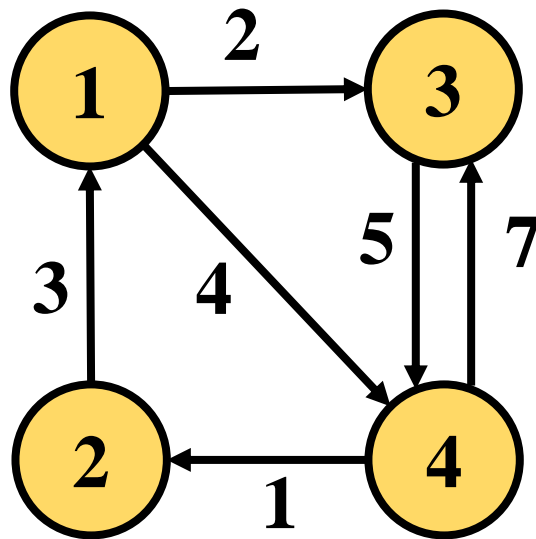
mid: **1**  
start: **2**  
end: **3**

distan[start][end] **INF**  
distan[start][mid]+distan[mid][end] **3 + 2**  
 $\Rightarrow$  distan[start][end] **5**로 갱신!

# Floyd Warshall 플로이드 와샬

Step2. start  $\rightarrow$  mid  $\rightarrow$  end와 start  $\rightarrow$  end의 최단 거리 비교

end start	1	2	3	4
1	0	INF	2	4
2	3	0	5	INF
3	INF	INF	0	5
4	INF	1	7	0



mid: **1**  
start: **2**  
end: **4**

distan[start][end] **INF**

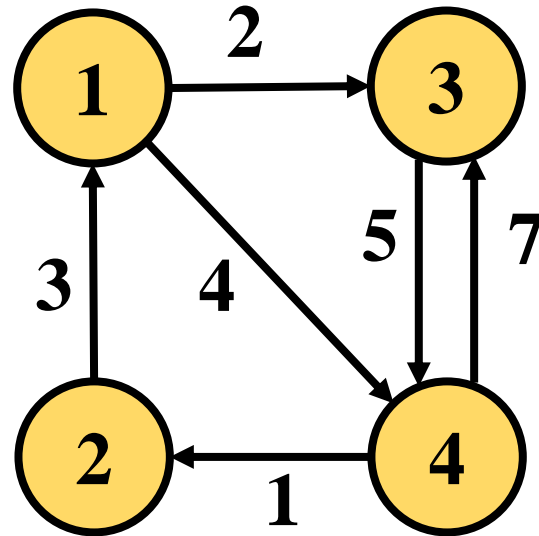
distan[start][mid]+distan[mid][end] **3 + 4**

$\Rightarrow$  distan[start][end] **7**로 갱신!

# Floyd Warshall 플로이드 와샬

Step2. start  $\rightarrow$  mid  $\rightarrow$  end와 start  $\rightarrow$  end의 최단 거리 비교

end start	1	2	3	4
1	0	INF	2	4
2	3	0	5	7
3	INF	INF	0	5
4	INF	1	7	0



mid: **1**  
start: **3**  
end: **2**

distan[start][end]

INF

distan[start][mid]+distan[mid][end]

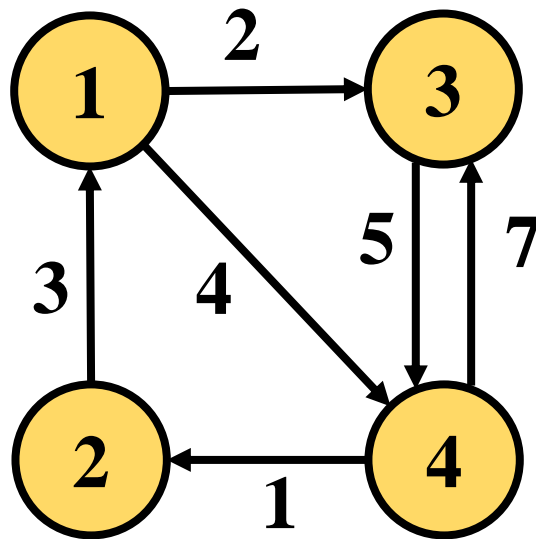
INF + INF

$\Rightarrow$  Pass!

# Floyd Warshall 플로이드 와샬

Step2. start  $\rightarrow$  mid  $\rightarrow$  end와 start  $\rightarrow$  end의 최단 거리 비교

end start	1	2	3	4
1	0	INF	2	4
2	3	0	5	7
3	INF	INF	0	5
4	INF	1	7	0



mid: 1  
start: 3  
end: 4

distan[start][end]

5

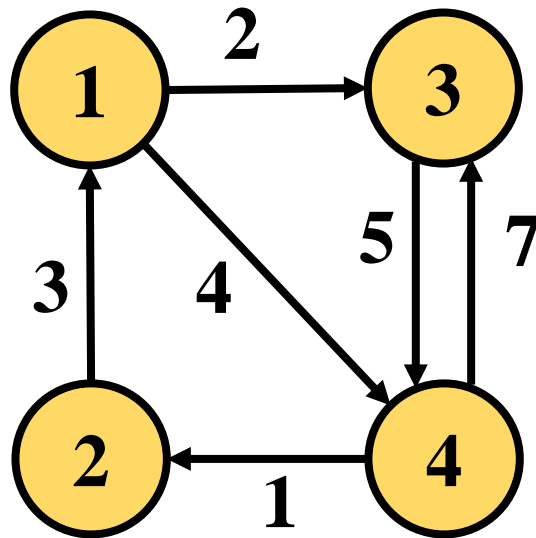
distan[start][mid]+distan[mid][end] INF + 4

$\Rightarrow$  Pass!

# Floyd Warshall 플로이드 와샬

Step2. start  $\rightarrow$  mid  $\rightarrow$  end와 start  $\rightarrow$  end의 최단 거리 비교

end start	1	2	3	4
1	0	INF	2	4
2	3	0	5	7
3	INF	INF	0	5
4	INF	1	7	0



mid: 1  
start: 4  
end: 2

distan[start][end]

1

distan[start][mid]+distan[mid][end] INF + INF

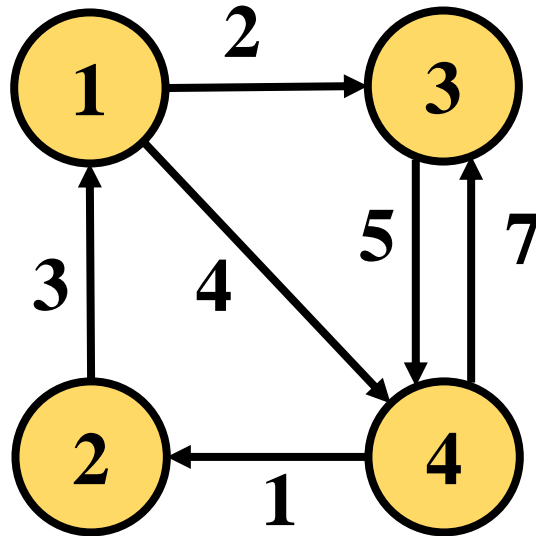
$\Rightarrow$  Pass!



# Floyd Warshall 플로이드 와샬

Step2. start  $\rightarrow$  mid  $\rightarrow$  end와 start  $\rightarrow$  end의 최단 거리 비교

end start	1	2	3	4
1	0	INF	2	4
2	3	0	5	7
3	INF	INF	0	5
4	INF	1	7	0



mid: 1  
start: 4  
end: 3

distan[start][end]

7

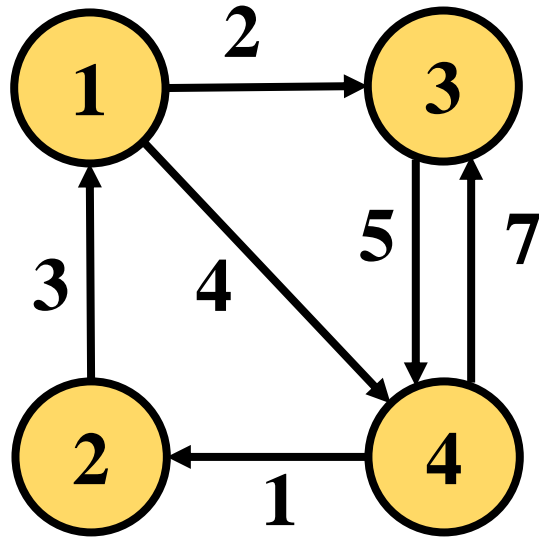
distan[start][mid]+distan[mid][end] INF +2

$\Rightarrow$  Pass!

# Floyd Warshall 플로이드 와샬

Step3. 모든 정점 mid에 대해 반복

end start	1	2	3	4
1	0	INF	2	4
2	3	0	5	7
3	INF	INF	0	5
4	4	1	6	0



mid: 2

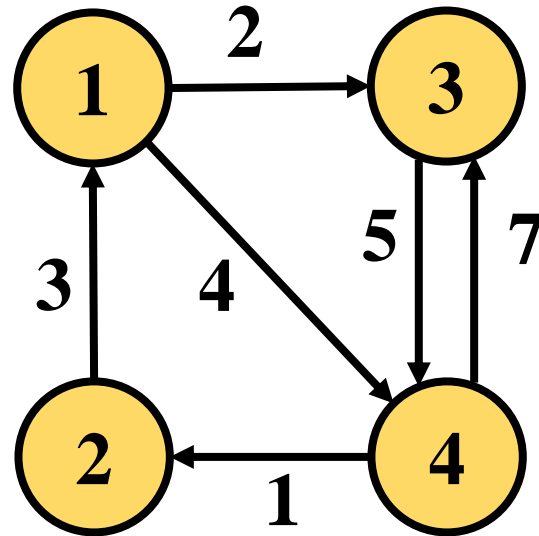
⇒ 4 → 1의 경로가 4 → 2 → 1으로 갱신!

⇒ 4 → 3의 경로가 4 → 2 → 3으로 갱신!

# Floyd Warshall 플로이드 와샬

Step3. 모든 정점 mid에 대해 반복

end start	1	2	3	4
1	0	INF	2	4
2	3	0	5	7
3	INF	INF	0	5
4	4	1	6	0



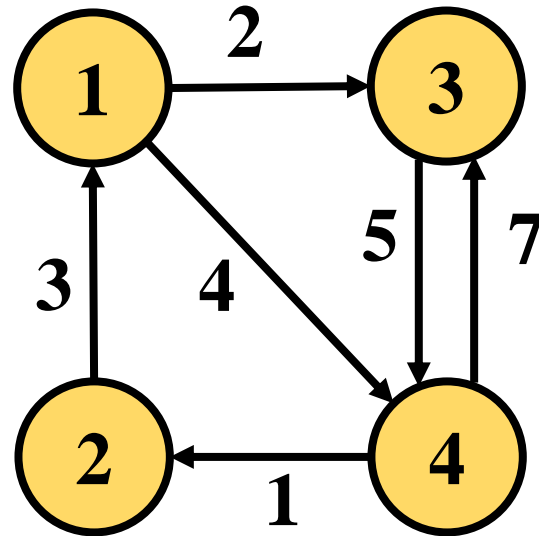
mid: 3

⇒ 갱신된 경로가 없다!

# Floyd Warshall 플로이드 와샬

Step3. 모든 정점 mid에 대해 반복

end start	1	2	3	4
1	0	5	2	4
2	3	0	5	7
3	9	6	0	5
4	4	1	6	0



mid: 4

⇒ 1 → 2의 경로가 1 → 4 → 2으로 갱신!

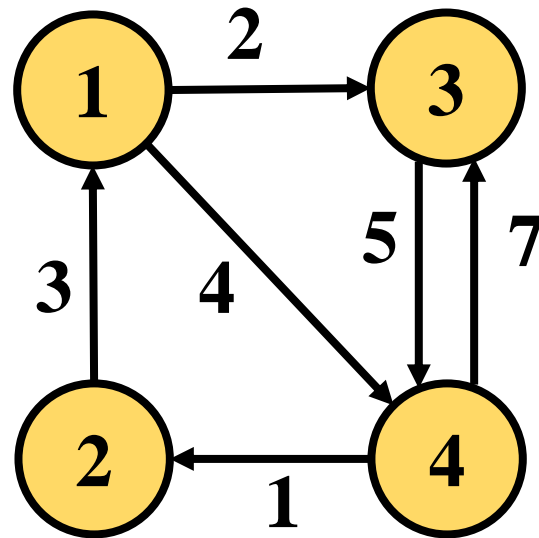
⇒ 3 → 1의 경로가 3 → 4 → 1으로 갱신!

⇒ 3 → 2의 경로가 3 → 4 → 2으로 갱신!

# Floyd Warshall 플로이드 와샬

최단 경로를 구한 결과

end start \	1	2	3	4
1	0	5	2	4
2	3	0	5	7
3	9	6	0	5
4	4	1	6	0



※ INF이면 경로가 존재하지 않는 것

# Floyd Warshall 문제풀이

---

플로이드 와샬 기본 문제!

**백준 11404**  
**플로이드**

## 11404 플로이드

---

문제: 모든 도시의 쌍에 대해 이동에 필요한 비용의 최소값 구하기

$$1 \leq n \leq 100 \qquad 1 \leq m \leq 100,000$$

$n^3 = 1,000,000$ 이므로 플로이드 와샬 알고리즘으로 충분히 가능!

# 플로이드 코드

초기화와 입력 받기

```
1  #include <iostream>
2  #include <algorithm>
3  #define INF 987654321
4  using namespace std;
5
6  int dp[105][105];
7  int n, m;
8
9  int main() {
10     ios_base::sync_with_stdio(false);
11     cin.tie(NULL);
12     cout.tie(NULL);
13     cin >> n;
14     cin >> m;
15     for (int i = 1; i ≤ n; i++) {
16         for (int j = 1; j ≤ n; j++) {
17             if (i == j)
18                 dp[i][j] = 0;
19             else
20                 dp[i][j] = INF;
21         }
22     } // 초기화
23     for (int i = 0; i < m; i++) {
24         int a, b, c; cin >> a >> b >> c;
25         dp[a][b] = min(dp[a][b], c); // 간선 받기
26     }
```



# 플로이드 코드

dp 배열 갱신, 출력

```
27     for (int mid = 1; mid ≤ n; mid++) {
28         for (int start = 1; start ≤ n; start++) {
29             for (int end = 1; end ≤ n; end++) {
30                 dp[start][end] = min(dp[start][end], dp[start][mid] + dp[mid][end]);
31             } // dp 갱신
32         }
33     }
34     for (int i = 1; i ≤ n; i++) {
35         for (int j = 1; j ≤ n; j++) {
36             if (dp[i][j] == INF) // 출력
37                 cout << "0 ";
38             else cout << dp[i][j] << " ";
39         }
40         cout << "\n";
41     }
42
43     return 0;
44 }
```

# Bellman Ford 벨만 포드

하나의 시작 정점에서 **다른 모든 정점**으로 가는 최단 경로를 구하는 알고리즘

1. 시작 정점은 0, 다른 정점은 INF로 초기화
2. **모든 간선**을 확인하면서 **각 간선의 도착 정점까지의 최단 거리 갱신**
3. 2번을  $V - 1$ 번 반복  $\rightarrow$  두 정점 사이 **최대 간선 개수는  $V - 1$** 이기 때문
4. 2번을 **한 번 더** 했을 때 갱신이 된다면 **음수 사이클 존재!**

- **음수 가중치**의 간선이 있어도 사용가능 + **음수 사이클** 검출 가능
- 시간 복잡도  **$O(VE)$**  ( $V$ : 정점 개수 /  $E$ : 간선 개수)

# Bellman Ford 벨만 포드

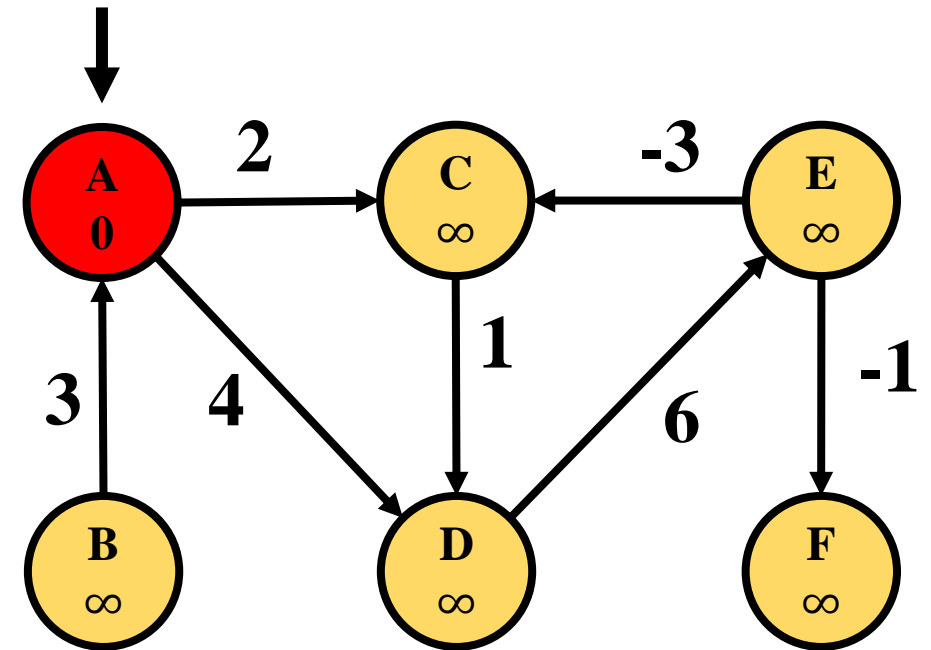
갱신 반복 횟수: 0

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

A의 최단 경로를 0,

나머지 정점의 최단 경로는 INF(무한)으로 초기화

시작 위치



■ 갱신 시작 정점

■ 갱신된 정점

■ 나머지 정점

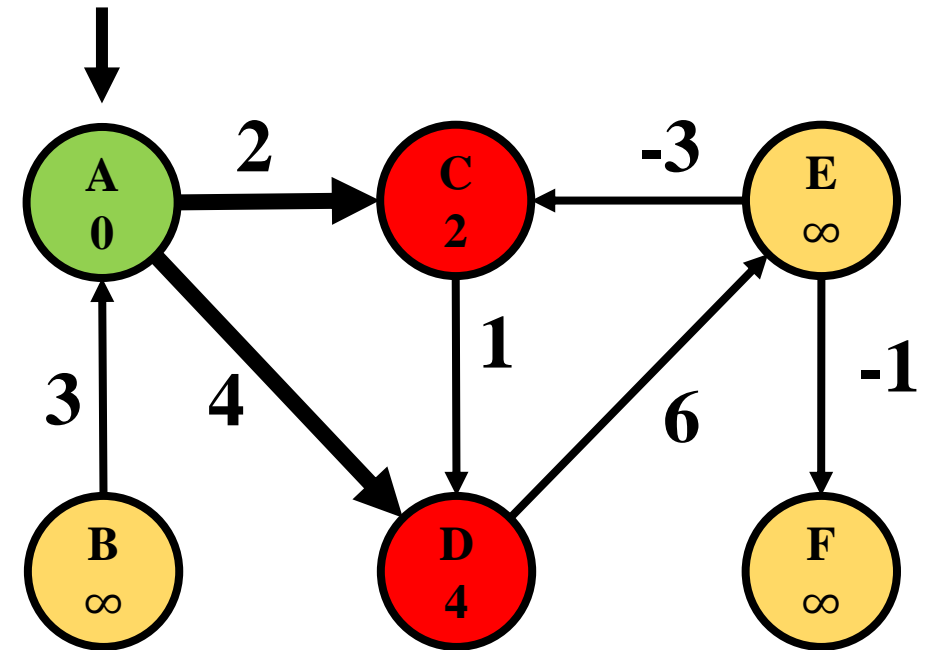
# Bellman Ford 벨만 포드

갱신 반복 횟수: 1

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	2	4	$\infty$	$\infty$

갱신된 정점에서 출발하는 간선에 대해  
도착지점의 최단거리 갱신

시작 위치



- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

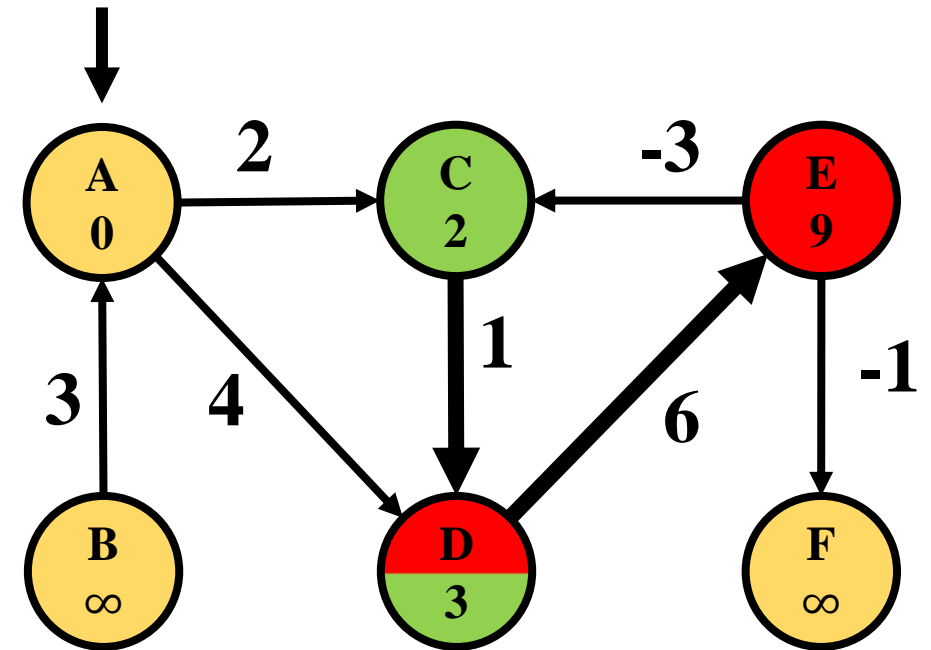
# Bellman Ford 벨만 포드

갱신 반복 횟수: 2

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	2	3	9	$\infty$

갱신된 정점에서 출발하는 간선에 대해  
도착지점의 최단거리 갱신

시작 위치



- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

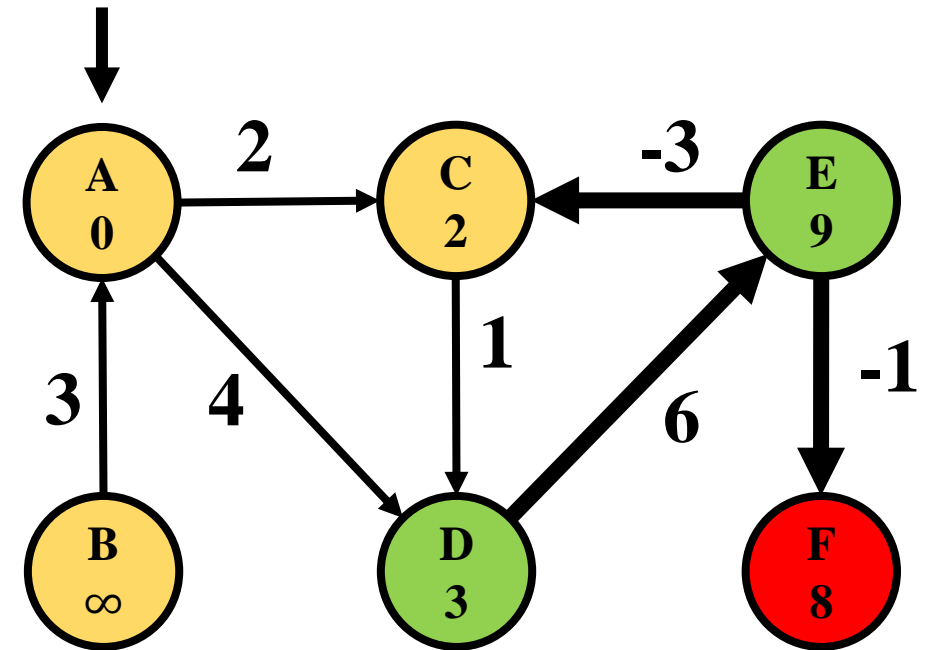
# Bellman Ford 벨만 포드

갱신 반복 횟수: 3

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	2	3	9	8

갱신된 정점에서 출발하는 간선에 대해  
도착지점의 최단거리 갱신

시작 위치



- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

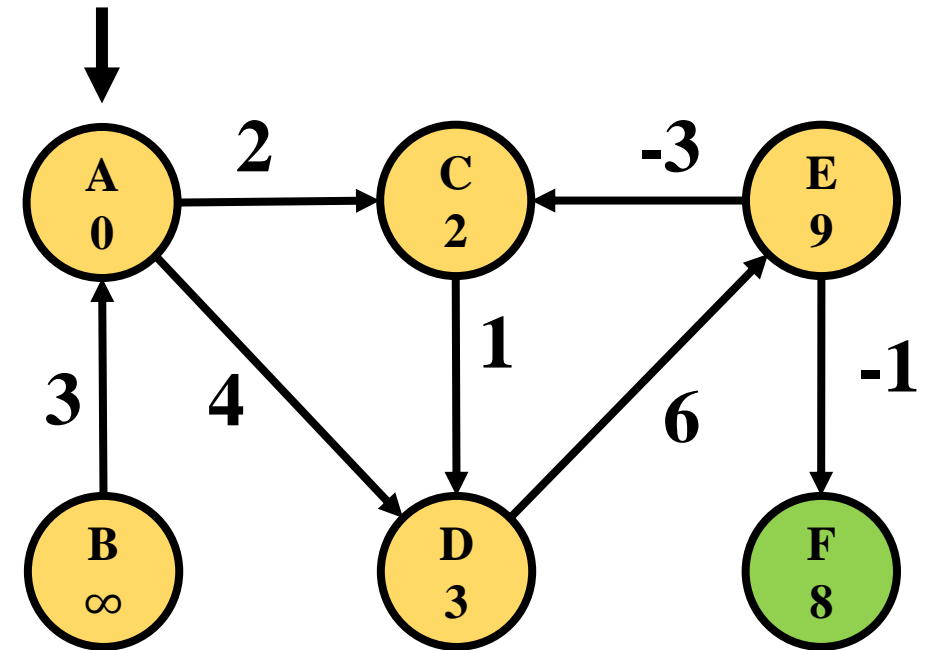
# Bellman Ford 벨만 포드

갱신 반복 횟수: 4

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	2	3	9	8

갱신된 정점에서 출발하는 간선에 대해  
도착지점의 최단거리 갱신

시작 위치



- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

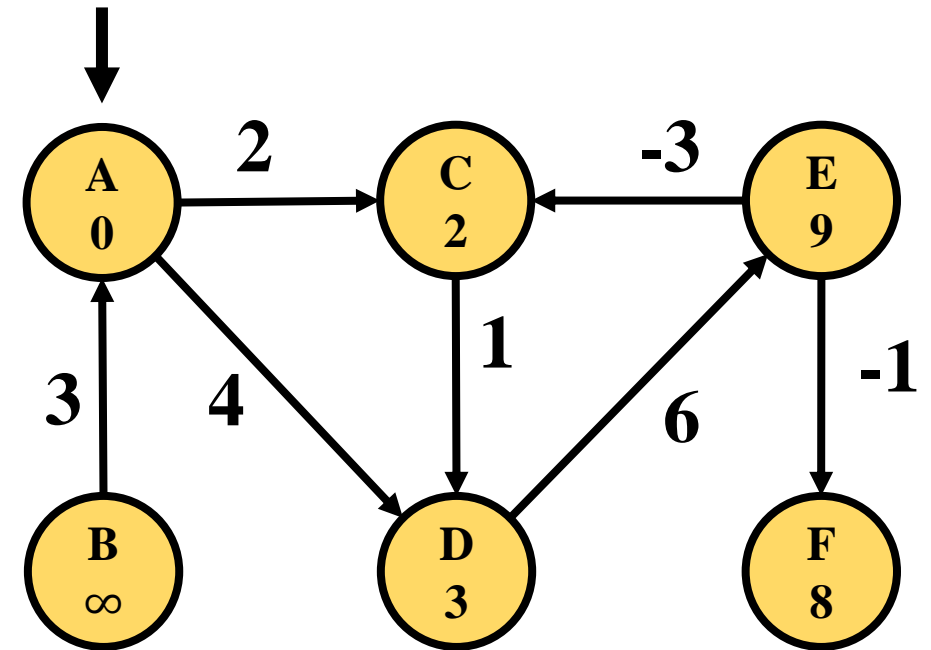
# Bellman Ford 벨만 포드

갱신 반복 횟수: 5

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	2	3	9	8

V - 1 번째 이후에 갱신되는 정점이 없다  
→ 음수 사이클이 존재하지 않는다!

시작 위치



- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

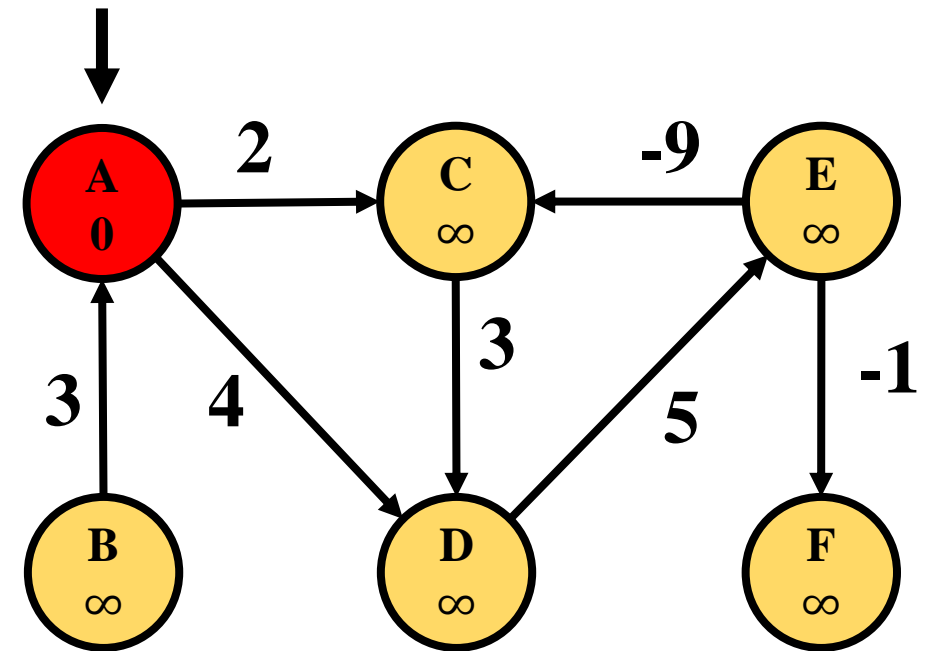


# Bellman Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 0

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

시작 위치



A의 최단 경로를 0,

나머지 정점의 최단 경로는 INF(무한)으로 초기화

■ 갱신 시작 정점

■ 갱신된 정점

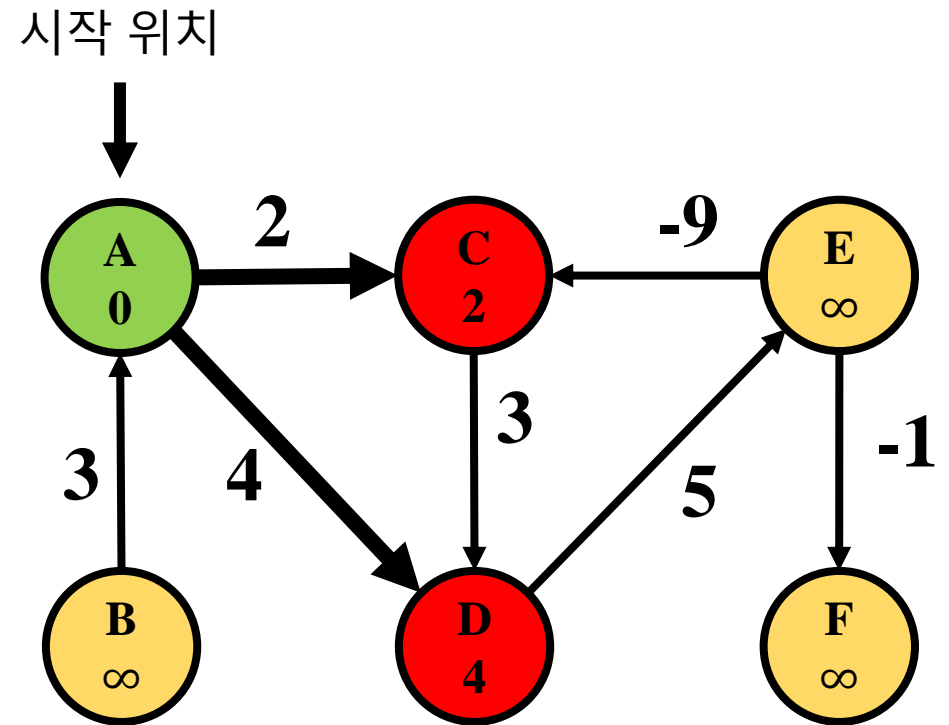
■ 나머지 정점

# Bellman Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 1

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	2	4	$\infty$	$\infty$

갱신된 정점에서 출발하는 간선에 대해  
도착지점의 최단거리 갱신



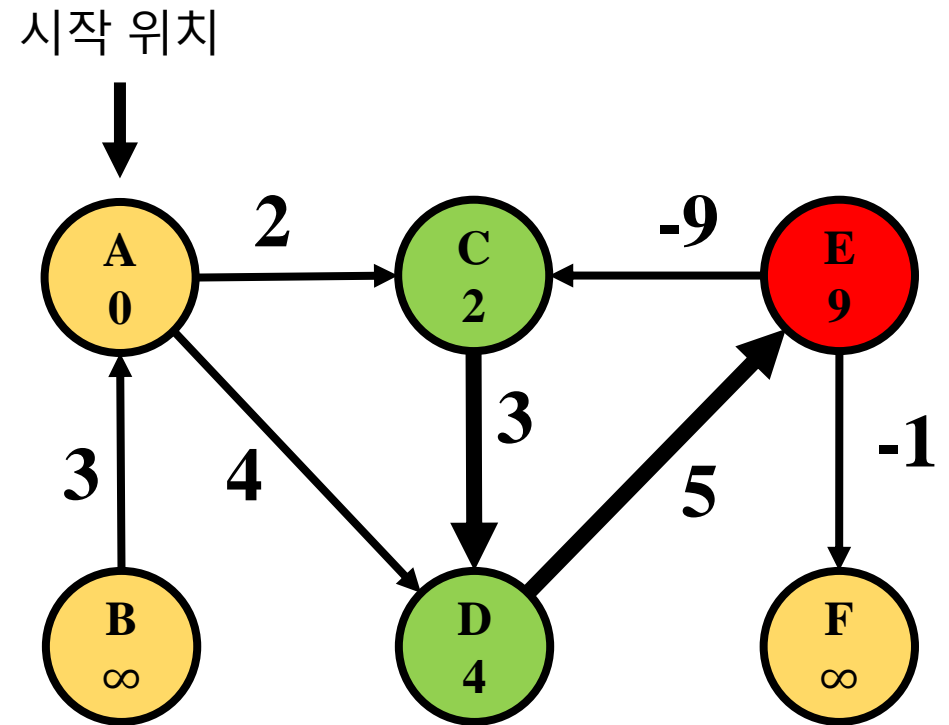
- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

# Bellman Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 2

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	2	4	9	$\infty$

갱신된 정점에서 출발하는 간선에 대해  
도착지점의 최단거리 갱신



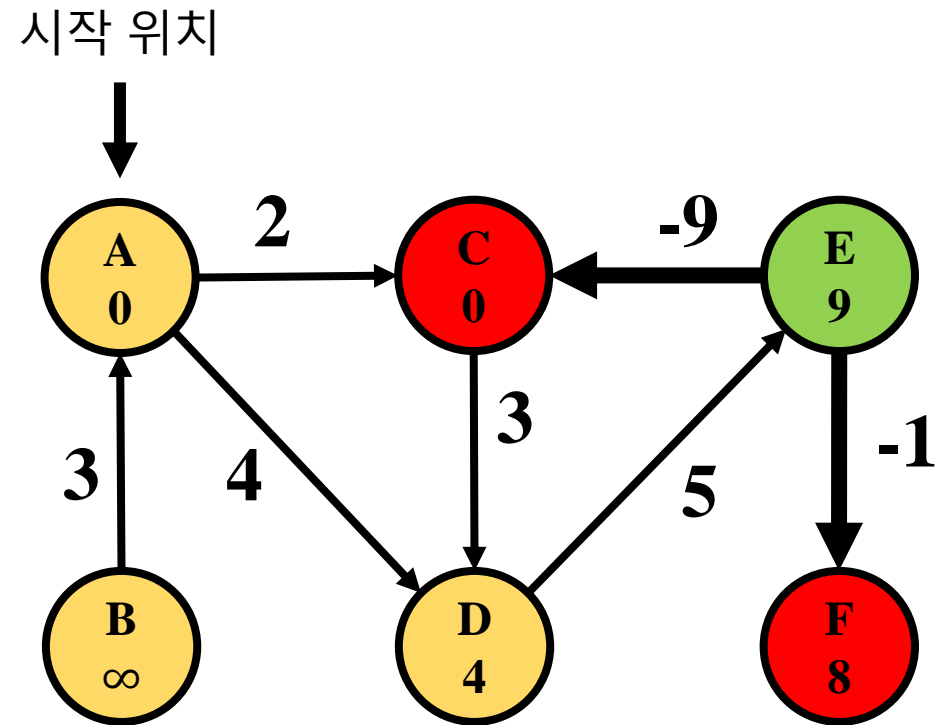
- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

# Bellman Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 3

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	0	4	9	8

갱신된 정점에서 출발하는 간선에 대해  
도착지점의 최단거리 갱신



- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

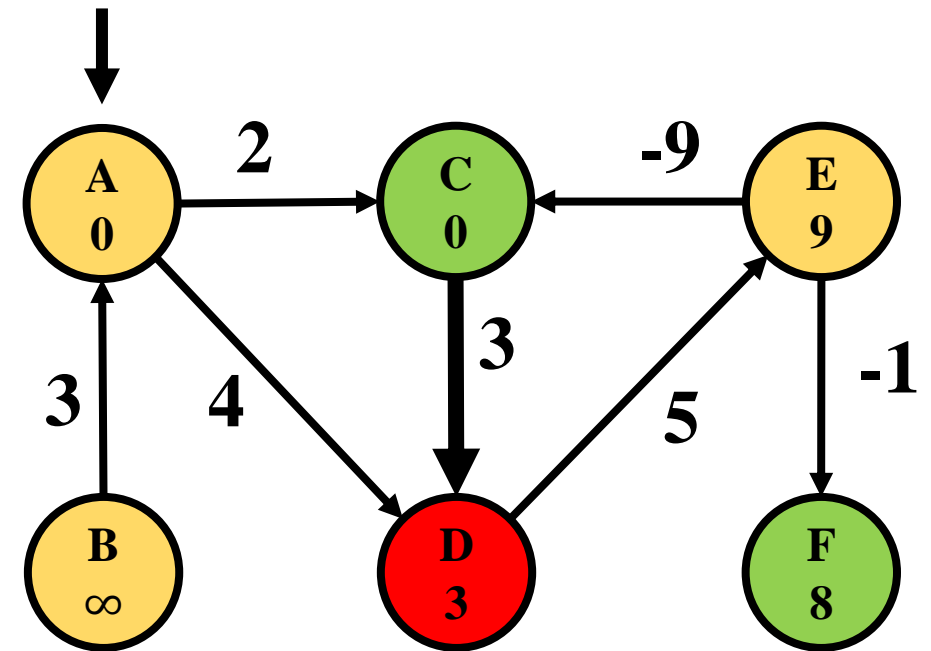
# Bellman Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 4

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	0	3	9	8

갱신된 정점에서 출발하는 간선에 대해  
도착지점의 최단거리 갱신

시작 위치



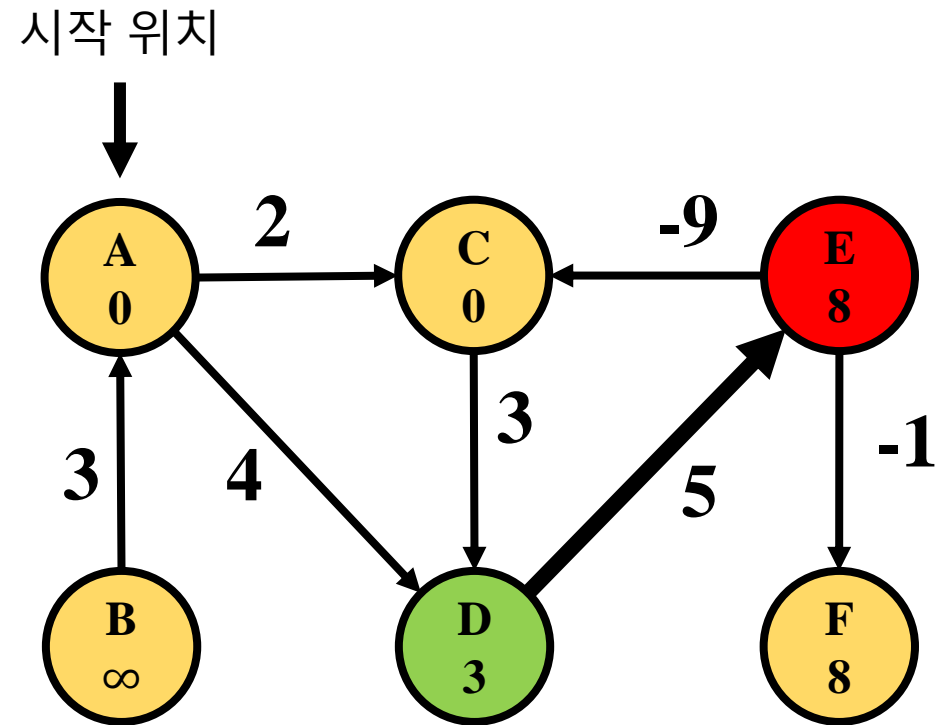
- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

# Bellman Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 5

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	0	3	8	8

갱신된 정점에서 출발하는 간선에 대해  
도착지점의 최단거리 갱신



- 갱신 시작 정점
- 갱신된 정점
- 나머지 정점

# Bellman Ford 벨만 포드 – with 음수 사이클

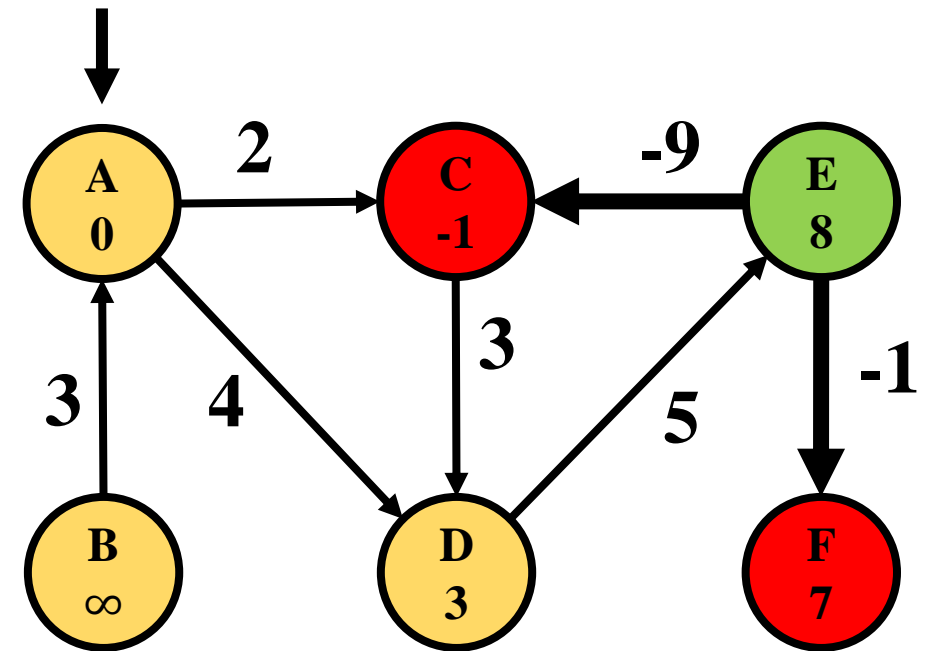
갱신 반복 횟수: 6

정점 번호	A	B	C	D	E	F
최단거리	0	$\infty$	-1	3	8	7

V번째에 갱신된다 (C, F 갱신)

→ 음수 사이클이 존재한다!

시작 위치



■ 갱신 시작 정점

■ 갱신된 정점

■ 나머지 정점

# Bellman Ford 문제풀이

---

벨만 포드 기본 문제!

**백준 11657**  
**타임머신**



# 11657 타임머신

문제: 음수 가중치가 있는 그래프에서 최단 경로 구하기

$1 \leq n \leq 500$      $1 \leq m \leq 6,000$      $-10,000 \leq C \leq 10,000$

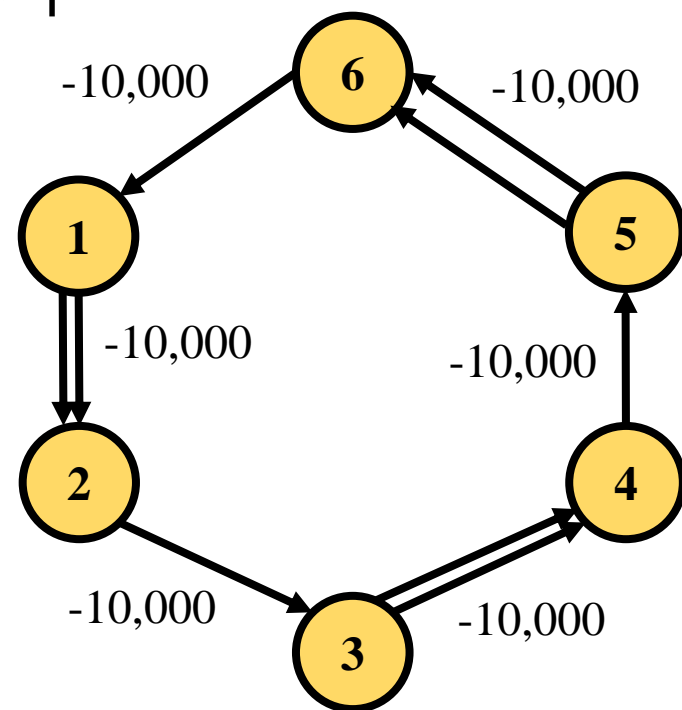
※ 주의할 점

⇒ 가중치가 전부 **-10,000**이고 **고리형태**이면 최단 거리는

$(n - 1) \times m \times (-10,000)$  이 될 수 있다

⇒ 대략  $3 \times 10^{10}$ 으로 int형이면 **underflow**가 발생할 수 있다!

⇒ 최단 거리 배열의 자료형이 **long long**이어야 한다



# 타임머신 코드

```
37 int main() {
38     ios_base::sync_with_stdio(false);
39     cin.tie(NULL);
40     cout.tie(NULL);
41     cin >> n >> m;
42     for (int i = 0; i < m; i++) {
43         ll a, b, c; cin >> a >> b >> c;
44         edges.push_back({ { a, b }, c });
45     }
46     if (bf()) { // 음수 사이클이 있으면
47         cout << "-1\n";
48         return 0;
49     }
50     for (int i = 2; i ≤ n; i++) {
51         if (distan[i] == INF)
52             cout << "-1\n";
53         else cout << distan[i] << "\n";
54     }
55     return 0;
56 }
```

```
1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4  #define INF 987654321
5  using namespace std;
6  using ll = long long;
7
8  int n, m;
9  vector<pair<pair<ll, ll>, ll>> edges;
10 ll distan[505]; // underflow 조심!!
11
12 int bf() {
13     for (int i = 1; i ≤ n; i++) {
14         distan[i] = INF;
15     } // 초기화
16     distan[1] = 0;
17     for (int i = 1; i < n; i++) {
18         for (pair<pair<ll, ll>, ll> edge : edges) {
19             ll start = edge.first.first; // 시작 정점
20             ll end = edge.first.second; // 도착 정점
21             ll weight = edge.second; // 가중치
22             if (distan[start] ≠ INF) // 갱신
23                 distan[end] = min(distan[end], distan[start] + weight);
24         }
25     }
26     for (pair<pair<ll, ll>, ll> edge : edges) {
27         ll start = edge.first.first;
28         ll end = edge.first.second;
29         ll weight = edge.second;
30         if (distan[start] ≠ INF && distan[end] > distan[start] + weight) {
31             return 1; // n - 1번 이후 갱신이 있으면 음수 사이클 존재
32         }
33     }
34     return 0; // 음수 사이클 없음
35 }
```

# SPFA (Shortest Path Faster Algorithm)

Bellman Ford의 비효율적인 부분을 개선한 알고리즘

↓  
최단 거리가 갱신된 정점에서 출발하는 간선에 대해서만 체크함

⇒ 시간 복잡도는 똑같이  $O(VE)$ 이지만

실제로는  $O(V+E)$ ,  $O(E)$ 의 성능을 가진다    최악의 경우,  $O(VE)$

구현: 출발하는 간선을 queue에 넣고 BFS와 유사하게 반복문을 사용해 탐색

# 타임머신 코드 using SPFA

```
1 #include <iostream>
2 #include <algorithm>
3 #include <vector>
4 #include <cstring>
5 #include <queue>
6 #define INF 987654321
7 using namespace std;
8 using ll = long long;
9
10 int n, m;
11 int inQueue[505]; // 큐에 들어있는지 확인
12 int cycle[505]; // 정점 방문 횟수 체크
13 ll distan[505];
14 vector<pair<ll, ll>> edges[505]; // 도착정점, 가중치
```

# 타임머신 코드 using SPFA

```
16 int bf() {
17     memset(cycle, 0, sizeof(cycle));
18     for (int i = 1; i ≤ n; i++) {
19         distan[i] = INF;
20     }
21     queue<ll> q;
22     distan[1] = 0;
23     inQueue[1] = 1;
24     cycle[1]++;
25     q.push(1); // 큐에 넣기
26     while (!q.empty()) {
27         int con = q.front();
28         q.pop();
29         inQueue[con] = 0;
30         for (pair<ll, ll> i : edges[con]) {
31             if (distan[i.first] > distan[con] + i.second) { // 갱신되면
32                 distan[i.first] = distan[con] + i.second;
33                 if (inQueue[i.first] == 0) { // 큐 안에 없으면
34                     inQueue[i.first] = 1;
35                     cycle[i.first]++;
36                     q.push(i.first); // 넣기
37                 }
38                 if (cycle[i.first] == n) // 정점을 충분히 많이 방문했을때
39                     return 1;
40             }
41         }
42     }
43     return 0;
44 }
```

# 타임머신 코드 using SPFA

```
46 int main() {
47     ios_base::sync_with_stdio(false);
48     cin.tie(NULL);
49     cout.tie(NULL);
50     cin >> n >> m;
51     for (int i = 0; i < m; i++) {
52         ll a, b, c; cin >> a >> b >> c;
53         edges[a].push_back({ b, c });
54     }
55     if (bf()) {
56         cout << "-1\n";
57         return 0;
58     }
59     for (int i = 2; i ≤ n; i++) {
60         if (distan[i] == INF)
61             cout << "-1\n";
62         else cout << distan[i] << "\n";
63     }
64     return 0;
65 }
```

# 최단 경로 알고리즘 정리

	다익스트라	플로이드 와샬	벨만 포드
알 수 있는 최단 경로	한 정점에서의 다른 모든 정점까지의 최단 경로	사이의 최모든 정점 단경로	한 정점에서의 다른 모든 정점까지의 최단 경로
시간복잡도	$O(E \log V)$	$O(V^3)$	$O(VE)$
음수 가중치	불가능	가능	가능
음수 사이클	불가능	불가능	가능

## Problem Set – 다익스트라

---



17396 백도어



1504 특정한 최단 경로



1238 파티



## Problem Set – 플로이드 와샬

---



2660 회장뽑기



11780 플로이드2



2610 회의준비

## Problem Set – 벨만 포드

---



1865 웜홀



1219 오민식의 고민

## Problem Set – 도전!

---



5719 거의 최단 경로



3860 할로윈 묘지