4주차 : 최단 경로 알고리즘

다익스트라, 플로이드 와샬, 벨만 포드

신지섭

오늘 할 내용은?

최단 경로 알고리즘에 대해 알아봅시다

1. 다익스트라 (Dijkstra)

2. 플로이드 와샬 (Floyd-Warshall)

3. 벨만 포드 (Bellman-Ford)

Dijkstra 다익스트라

하나의 시작 정점에서 다른 모든 정점으로 가는 최단 경로를 구하는 알고리즘

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

   ※ priority_queue를 이용해 가장 가까운 정점을 찾자!
3. priority_queue가 빌 때까지 반복

- **greedy**로 최단 거리를 확정 → 음수 가중치의 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

```cpp
#include <iostream>
#include <queue>
using namespace std;

priority_queue<int> pq;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    int n; cin >> n;
    for (int i = 0; i < n; i++) {
        int num; cin >> num;
        if (num == 0) {
            if (pq.empty())
                cout << "0\n";
            else {
                cout << -pq.top() << "\n";
                pq.pop();
            }
        }
        else {
            pq.push(-num); // -를 붙여 특별한 정렬없이 사용
        }
    }

    return 0;
}
```

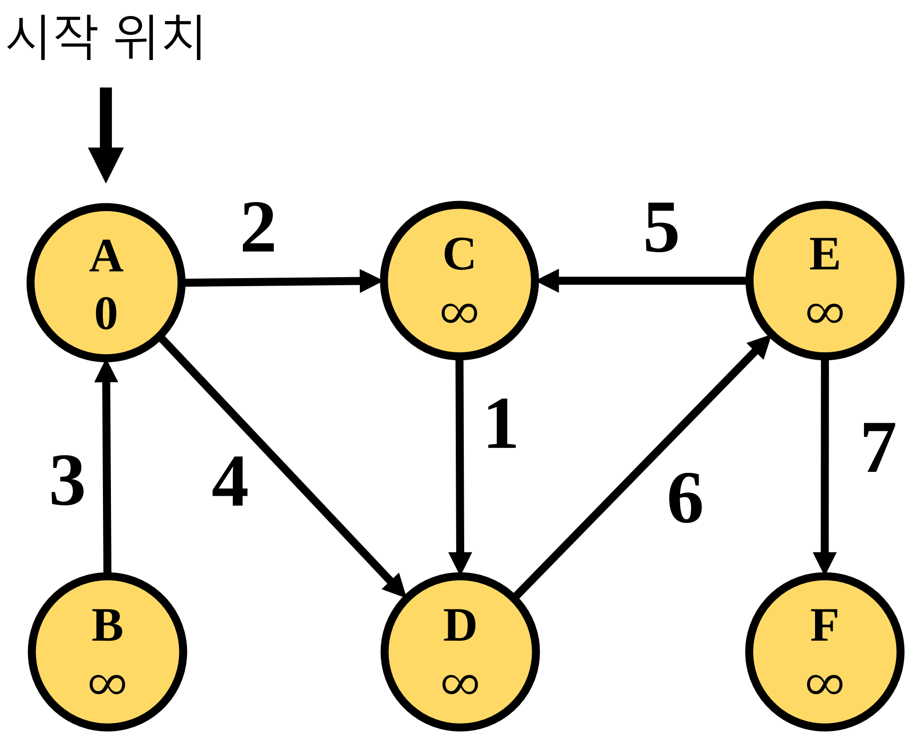
Dijkstra 다익스트라

현재 위치: **not started**

priority_queue

from	to	distance
A	A	0

top →



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

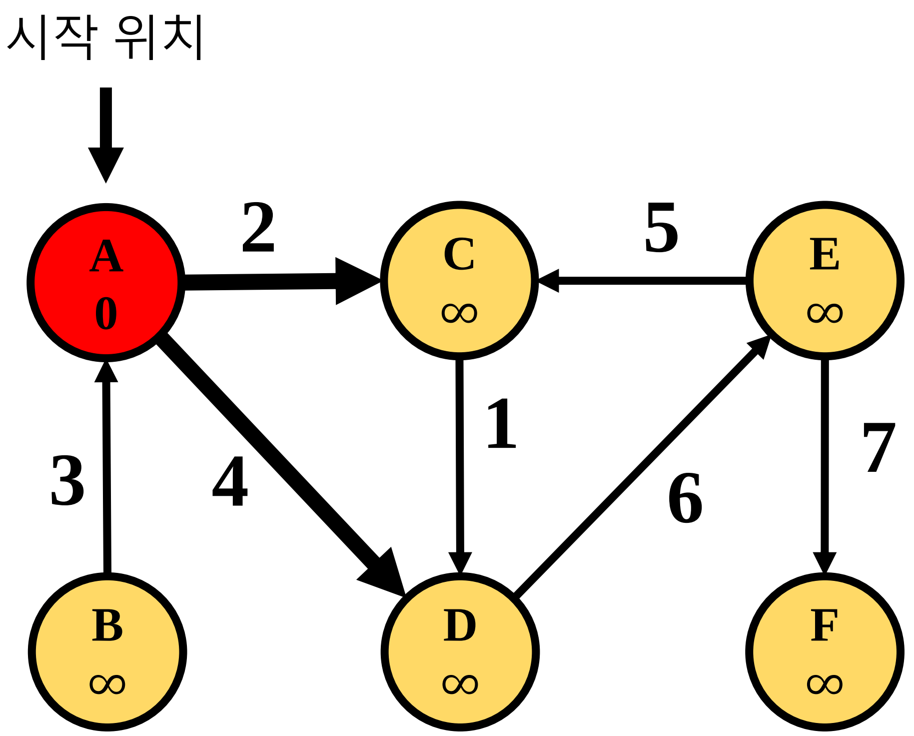
Dijkstra 다익스트라

현재 위치: A

priority_queue

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

top →



A에서 갈 수 있는 곳들의 최단 거리 후보 값을 priority_queue에 넣어줍니다

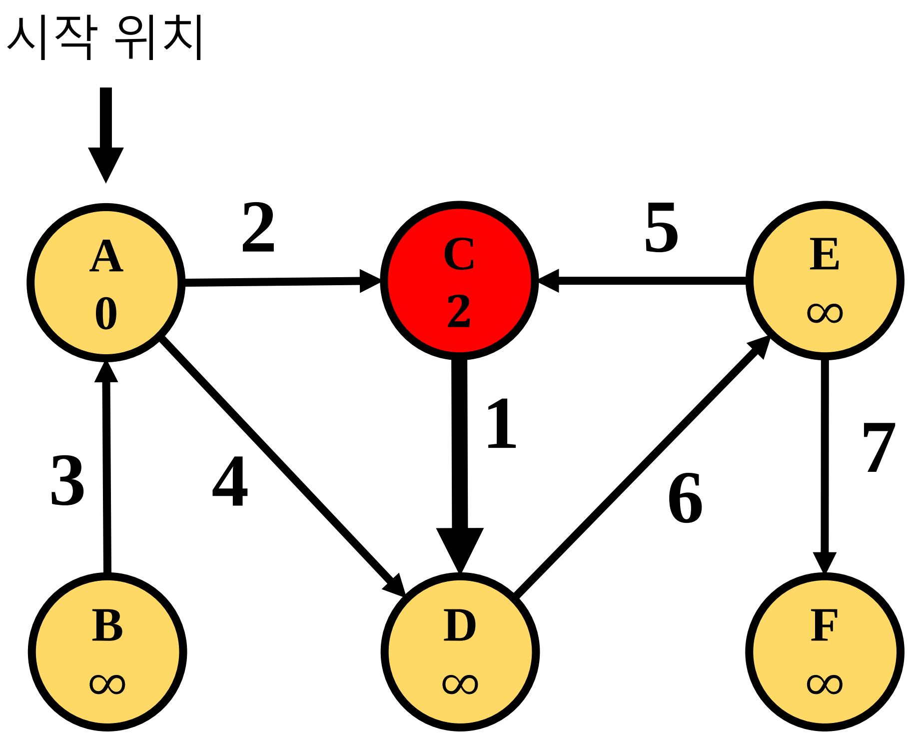
Dijkstra 다익스트라

현재 위치: C

priority_queue

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

top →



C에서 갈 수 있는 곳들의 최단 거리 후보 값을 priority_queue에 넣어줍니다

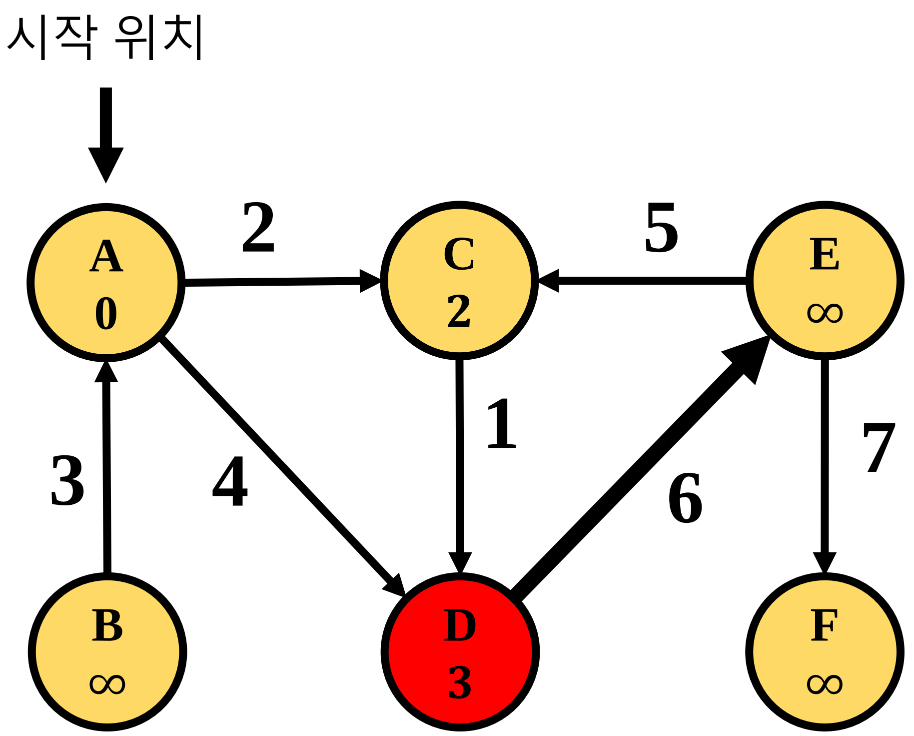
Dijkstra 다익스트라

현재 위치: **D**

priority_queue

top →

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



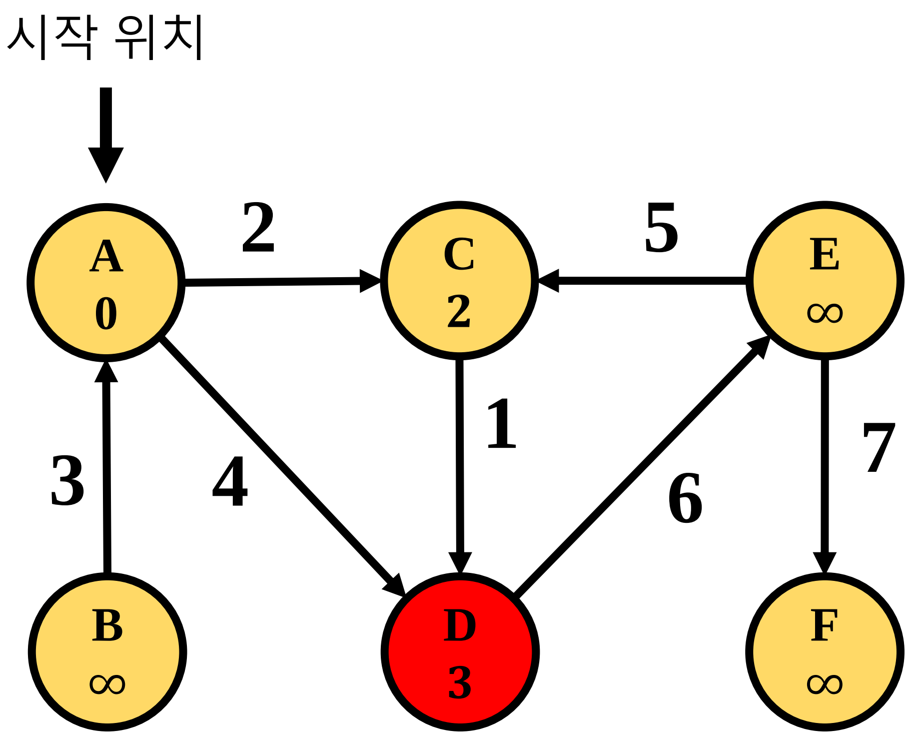
D에서 갈 수 있는 곳들의 최단 거리 후보 값을 priority_queue에 넣어줍니다

Dijkstra 다익스트라

현재 위치: **D**

priority_queue

	from	to	distance
top →	D	E	3 + 6



D까지의 최단 경로는 이미 확정되었으므로 갱신하지 않고 pass!

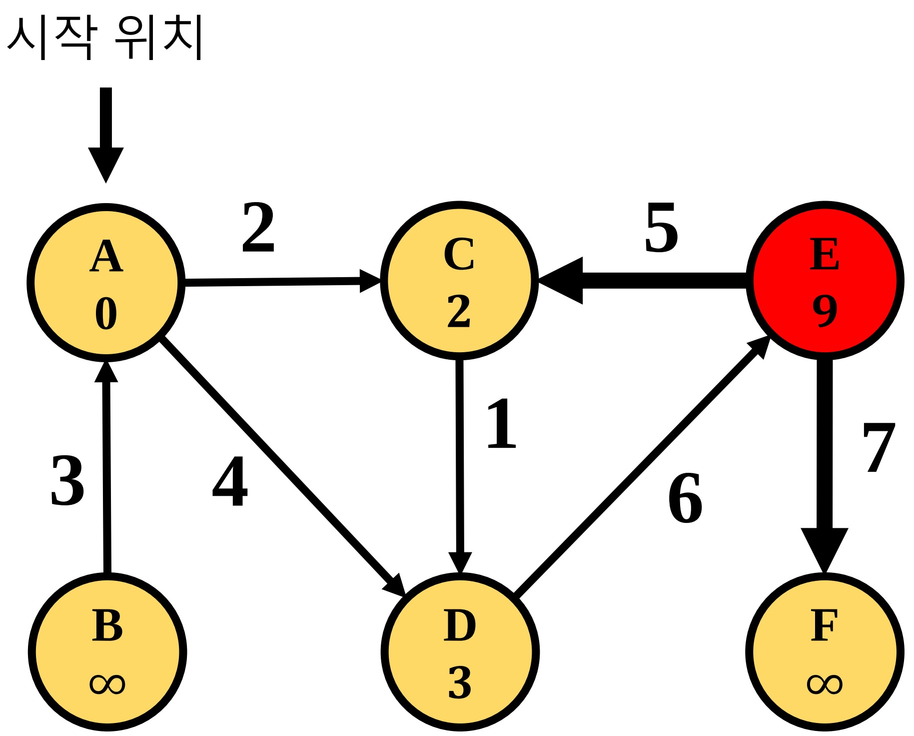
Dijkstra 다익스트라

현재 위치: E

priority_queue

top →

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



E에서 갈 수 있는 곳들의 최단 거리 후보 값을 priority_queue에 넣어줍니다

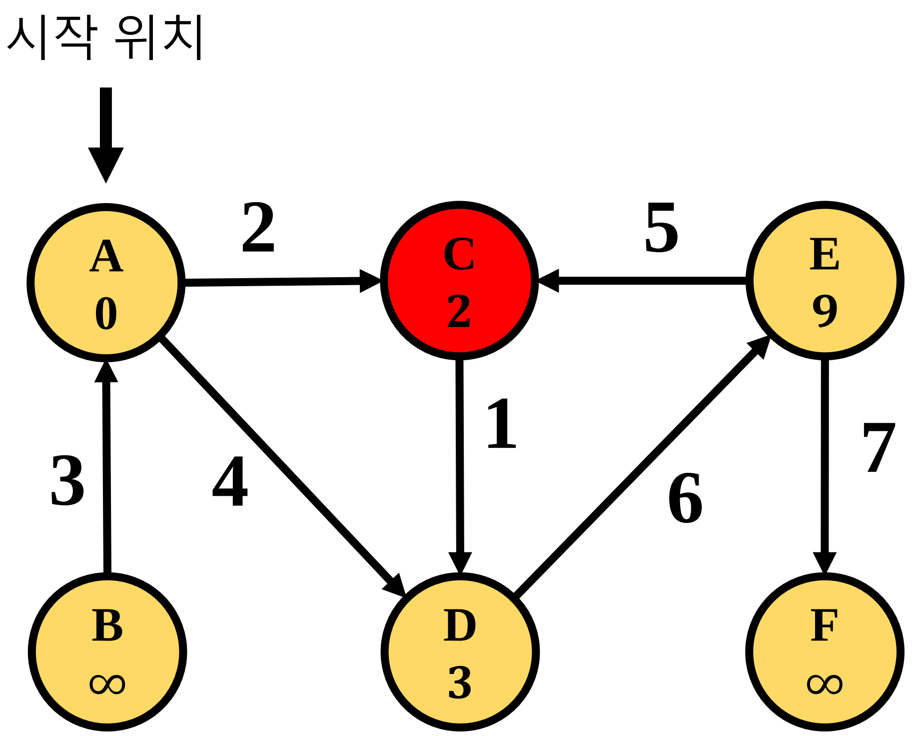
Dijkstra 다익스트라

현재 위치: C

priority_queue

top →

from	to	distance
E	F	9 + 7



C까지의 최단 경로는 이미 확정되었으므로 갱신하지 않고 pass!

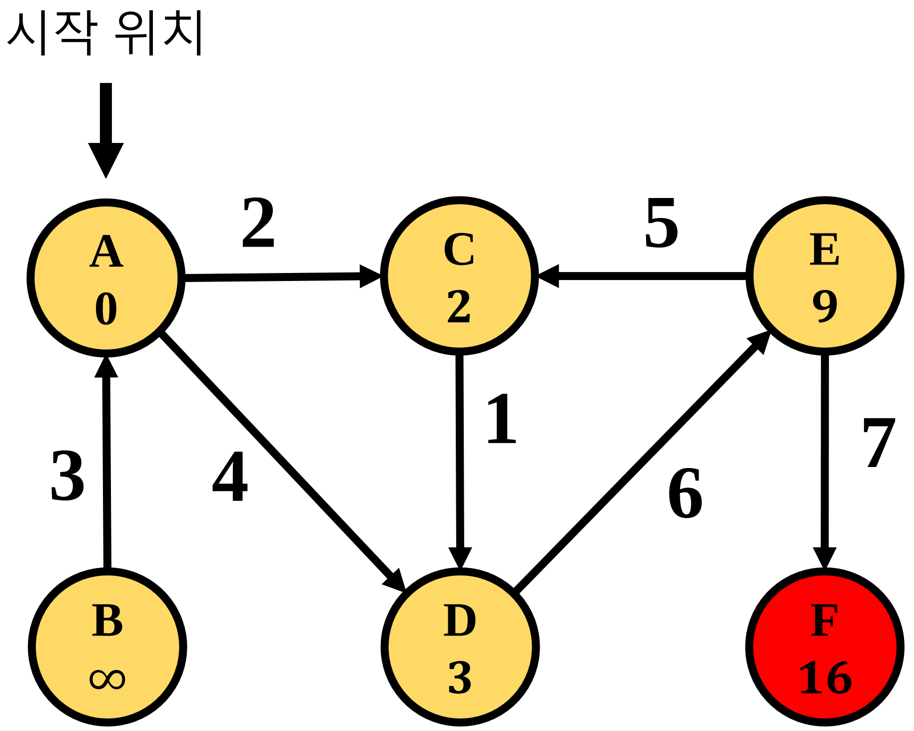
Dijkstra 다익스트라

현재 위치: **F**

priority_queue

from	to	distance

top →



priority_queue가 비었으므로 종료!

도달할 수 없으면 종료 후 최단 경로 값이 **INF**(무한)

이제 문제를 풀어봅시다!

Dijkstra 문제풀이

다익스트라 기본 문제!

백준 1753
최단 경로

1753 최단경로

문제: 주어진 방향 그래프에서 최단 경로를 찾는 문제

조건: $1 \le V \le 20{,}000$ $1 \le E \le 300{,}000$ $1 \le w \le 10$

⇒ 다익스트라를 사용하면 충분히 해결 가능!

최단경로 코드

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> v >> e;
    cin >> k;
    for (int i = 0; i < e; i++) {
        int a, b, c; cin >> a >> b >> c;
        edges[a].push_back({ b, c });
    }
    dij(k);
    for (int i = 1; i <= v; i++) {
        if (distan[i] == INF)
            cout << "INF\n";
        else cout << distan[i] << "\n";
    }
    return 0;
}
```

```cpp
#include <iostream>
#include <queue>
#define INF 987654321 // 무한대 정의
using namespace std;

int v, e, k;
vector<pair<int, int>> edges[20005]; //도착정점, 가중치
int distan[20005];

void dij(int start) {
    for (int i = 1; i <= v; i++) {
        distan[i] = INF;
    }
    priority_queue<pair<int, int>> pq; // 경로 값, 도착정점
    pq.push({ 0,start });
    while (!pq.empty()) {
        int con = pq.top().second;
        int dis = -pq.top().first; // -를 붙여 최소값 가져오기
        pq.pop();
        if (distan[con] < INF) continue; //이미 방문했으면
        distan[con] = dis;
        for (pair<int, int> i : edges[con]) {
            int next = i.first;
            int weight = i.second + dis;
            if (distan[next] < INF) continue; //이미 방문했으면
            pq.push({ -weight, next });
        }
    }
}
```

Floyd-Warshall 플로이드 와샬

모든 정점 사이의 최단 경로를 구하는 알고리즘

distan[start][end]: start → end의 최단거리를 나타내는 **dp배열**

1. start → end의 최단거리를 간선의 길이로 초기화
2. 모든 start → end조합에 대해, 특정한 정점 mid를 경유해서 가는

   start → mid → end와 start → end의 최단 거리를 비교
3. 모든 정점 mid에 대해 **반복**

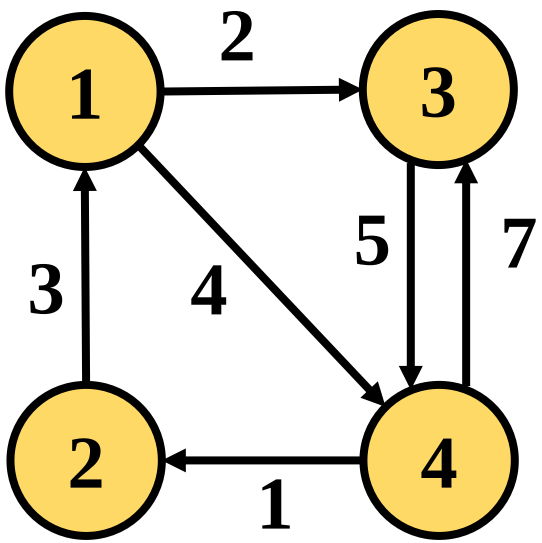
Tip. **mid**에 대한 반복문이 맨 밖에 와야 합니다!

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

Floyd-Warshall 플로이드 와샬

Step1. DP 배열 초기화

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



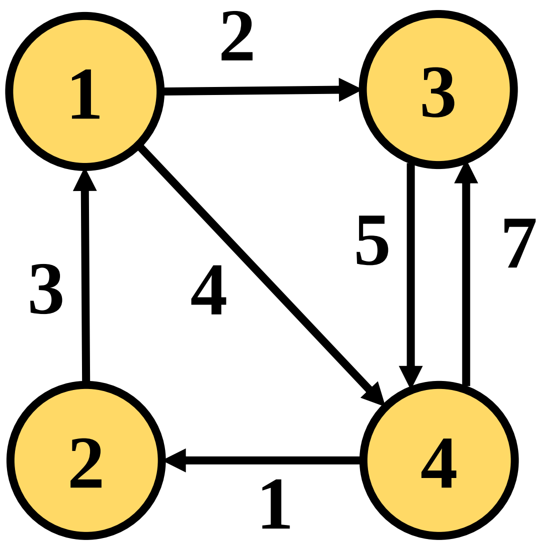
mid: **not started**
start: **not started**
end: **not started**

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

Floyd-Warshall 플로이드 와샬

Step2. start → mid → end와 start → 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

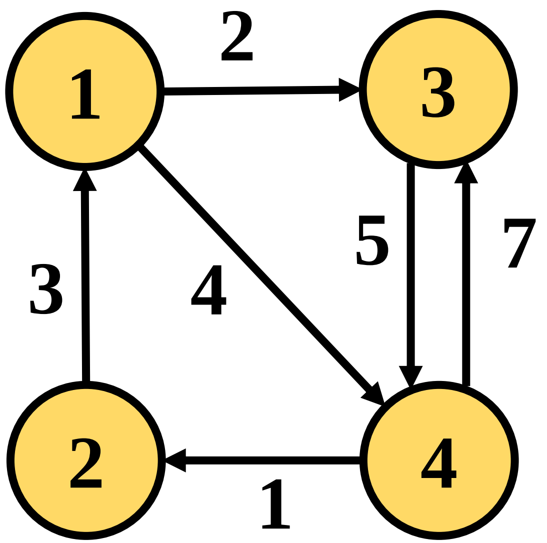
distan[start][mid]+distan[mid][end] 3 + 2

⇒ distan[start][end] 5로 갱신!

Floyd-Warshall 플로이드 와샬

Step2. start → mid → end와 start → 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

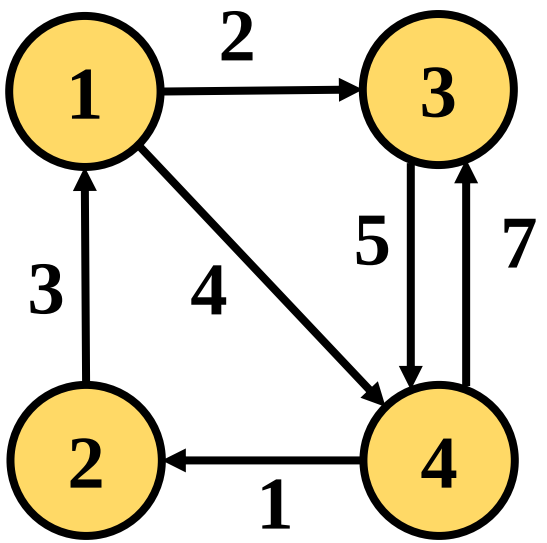
distan[start][end]	INF
distan[start][mid]+distan[mid][end]	3 + 4

⇒ distan[start][end] 7로 갱신!

Floyd-Warshall 플로이드 와샬

Step2. start → mid → end와 start → 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

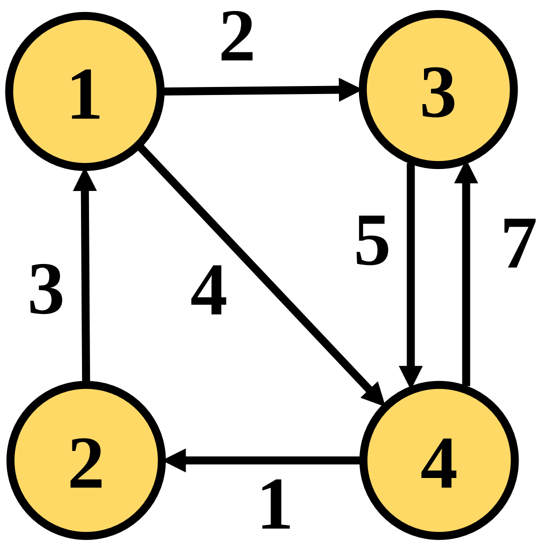
distan[start][mid]+distan[mid][end] INF + INF

⇒ Pass!

Floyd-Warshall 플로이드 와샬

Step2. start → mid → end와 start → 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

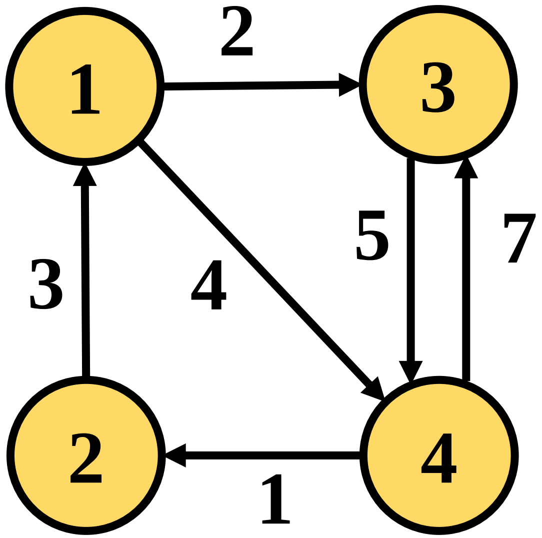
distan[start][mid]+distan[mid][end] INF + 4

⇒ Pass!

Floyd-Warshall 플로이드 와샬

Step2. start → mid → end와 start → 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

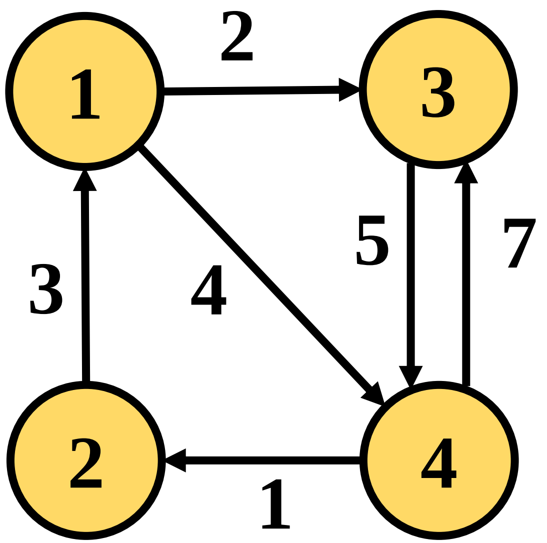
distan[start][mid]+distan[mid][end] INF + INF

⇒ Pass!

Floyd-Warshall 플로이드 와샬

Step2. start → mid → end와 start → end의 최단 거리 비교

start \ end	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

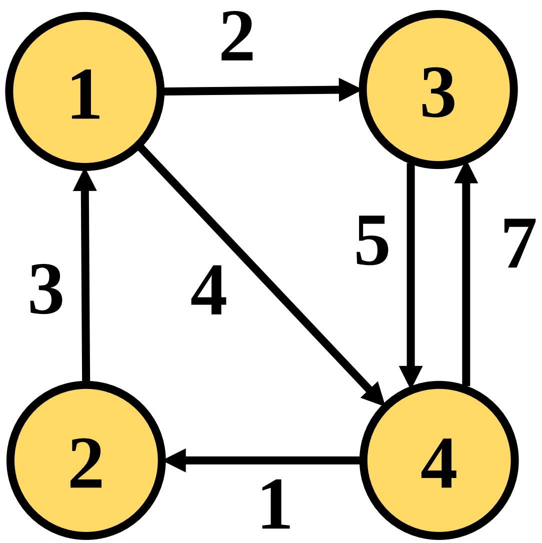
distan[start][mid]+distan[mid][end] INF +2

⇒ 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

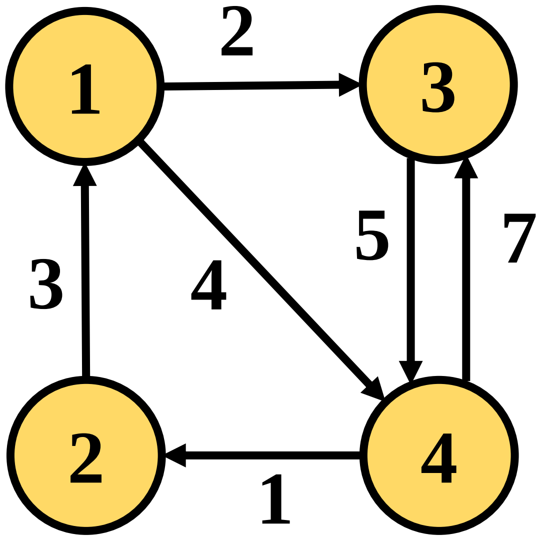
⇒ 4 → 1의 경로가 4 → 2 → 1으로 갱신!

⇒ 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



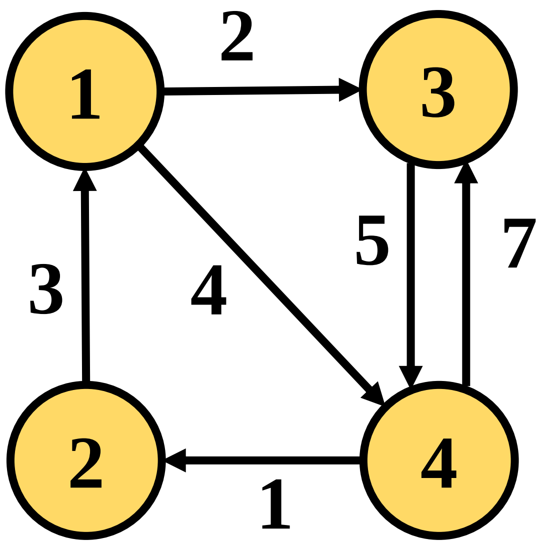
mid: 3

⇒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!

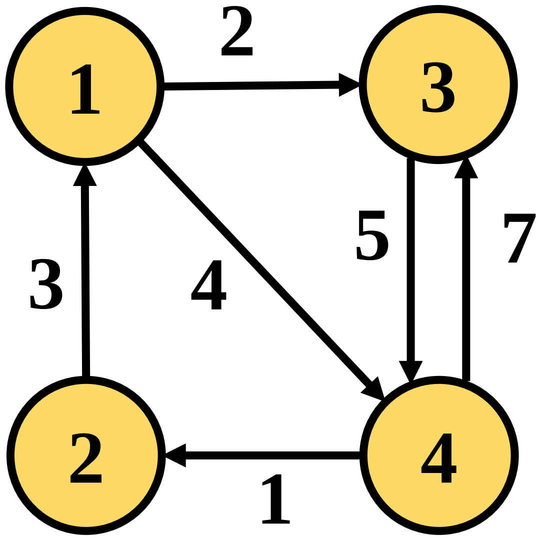
⇒ 3 → 1의 경로가 3 → 4 → 1으로 갱신!

⇒ 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 문제 풀이

플로이드 와샬 기본 문제!

백준 11404
플로이드

11404 플로이드

문제: **모든 도시의 쌍**에 대해 이동에 필요한 비용의 최소값 구하기

$1 \le n \le 100$ $1 \le m \le 100,000$ 비용은 자연수

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

플로이드 코드

초기화와 입력 받기

```cpp
#include <iostream>
#include <algorithm>
#define INF 987654321
using namespace std;

int dp[105][105];
int n, m;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n;
    cin >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == j)
                dp[i][j] = 0;
            else
                dp[i][j] = INF;
        }
    } // 초기화
    for (int i = 0; i < m; i++) {
        int a, b, c; cin >> a >> b >> c;
        dp[a][b] = min(dp[a][b], c); // 간선 받기
    }
```

플로이드 코드

dp 배열 갱신, 출력

```
    for (int mid = 1; mid <= n; mid++) {
        for (int start = 1; start <= n; start++) {
            for (int end = 1; end <= n; end++) {
                dp[start][end] = min(dp[start][end], dp[start][mid] + dp[mid][end]);
            } // dp 갱신
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (dp[i][j] == INF) // 출력
                cout << "0 ";
            else cout << dp[i][j] << " ";
        }
        cout << "\n";
    }

    return 0;
}
```

Bellman-Ford 벨만 포드

하나의 시작 정점에서 다른 모든 정점으로 가는 최단 경로를 구하는 알고리즘

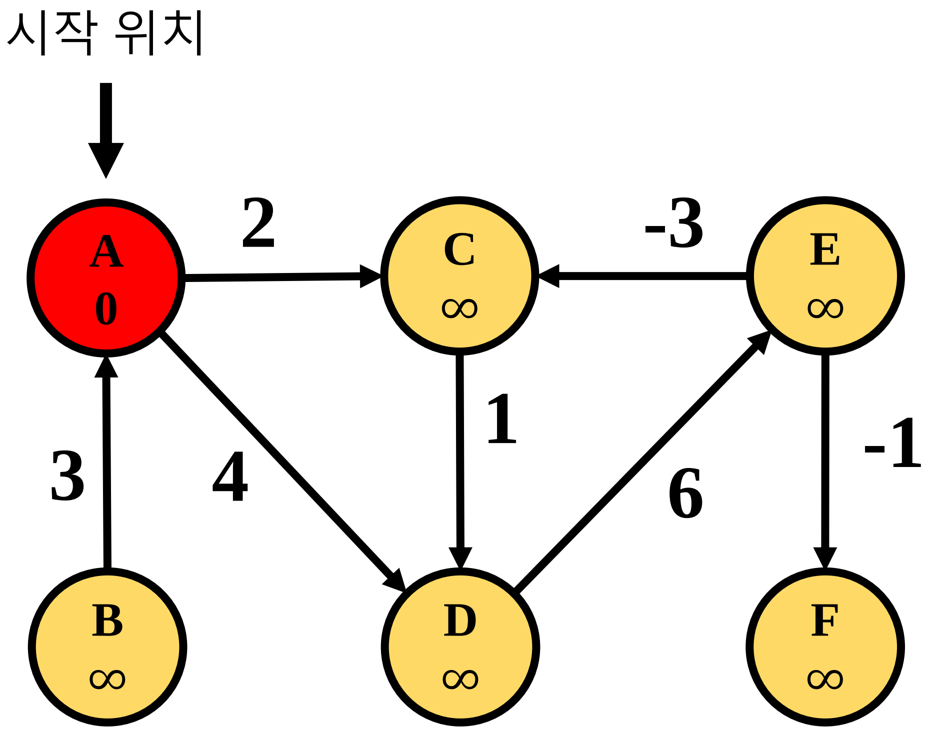
1. 시작 정점은 0, 다른 정점은 INF로 초기화
2. **모든 간선**을 확인하면서 각 간선의 도착 정점까지의 최단 거리 갱신
3. 2번을 **V – 1**번 반복 → 두 정점 사이 최대 간선 개수는 V - 1이기 때문
4. 2번을 **한 번 더** 했을 때 갱신이 된다면 음수 사이클 존재!

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

Bellman-Ford 벨만 포드

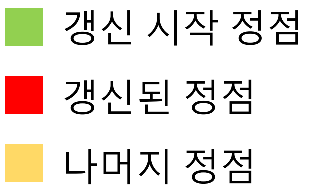
갱신 반복 횟수: 0

정점 번호	A	B	C	D	E	F
최단거리	0	∞	∞	∞	∞	∞



A의 최단 경로를 0,

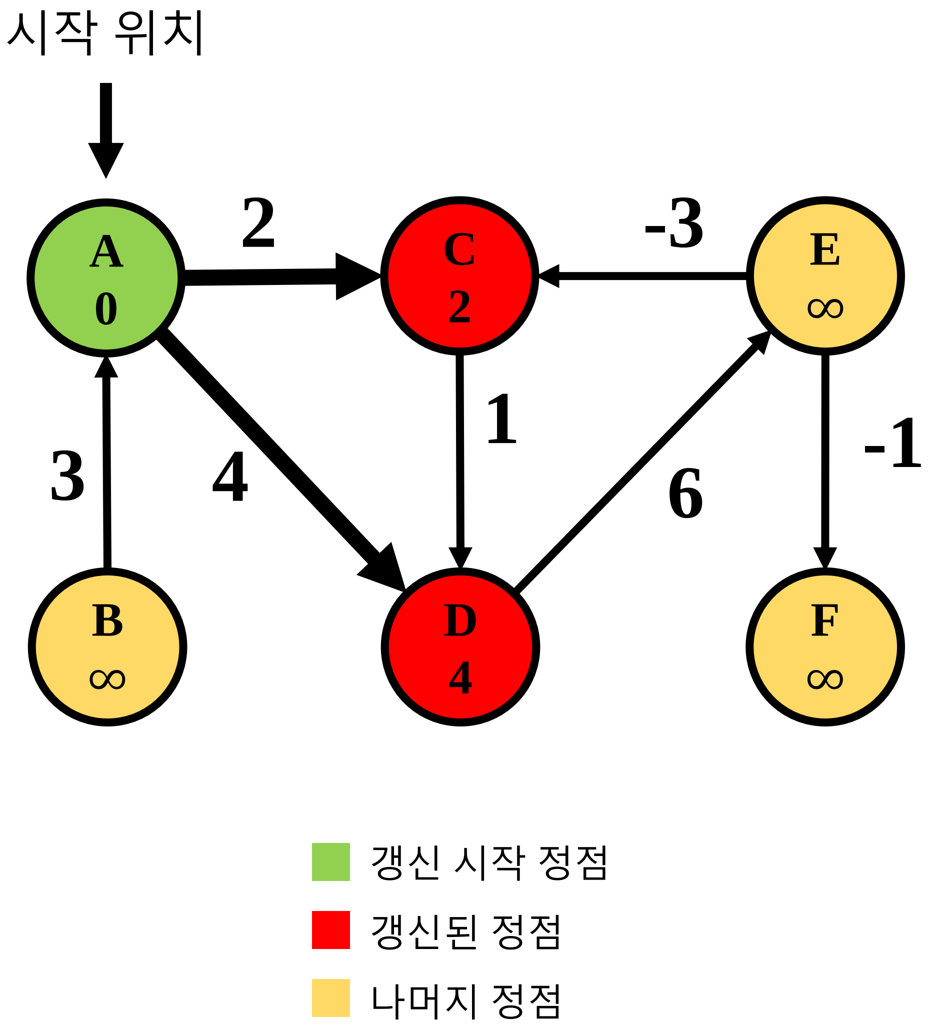
나머지 정점의 최단 경로는 INF(무한)으로 초기화



Bellman-Ford 벨만 포드

갱신 반복 횟수: 1

정점 번호	A	B	C	D	E	F
최단거리	0	∞	2	4	∞	∞



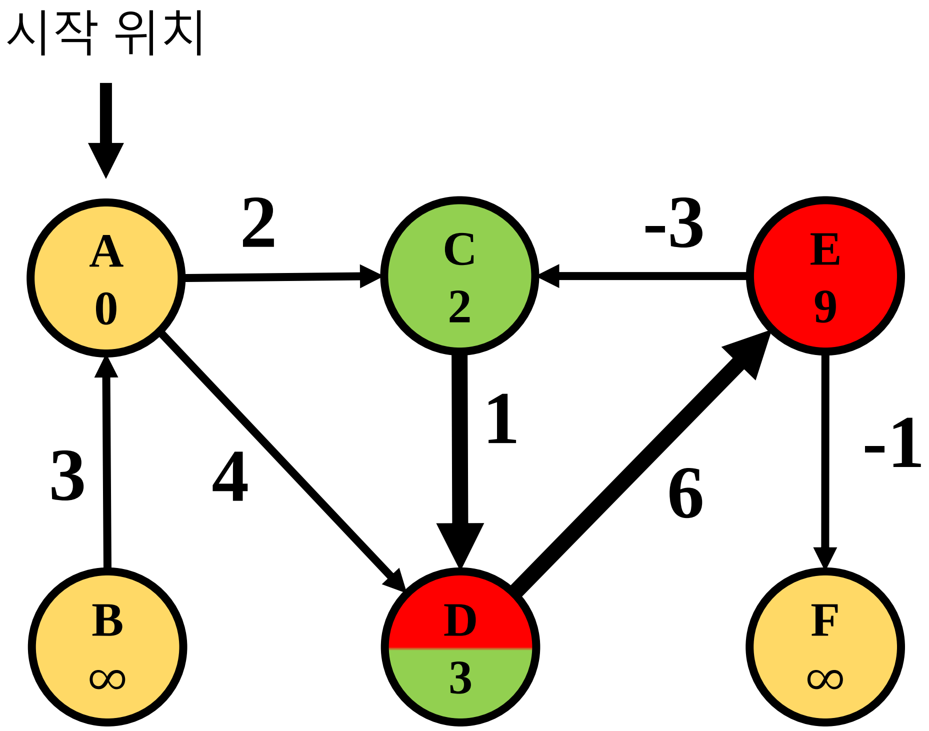
갱신된 정점에서 출발하는 간선에 대해

도착지점의 최단거리 갱신

Bellman-Ford 벨만 포드

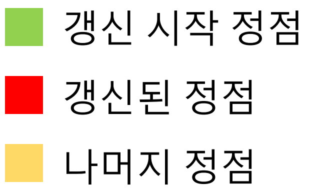
갱신 반복 횟수: 2

정점 번호	A	B	C	D	E	F
최단거리	0	∞	2	3	9	∞



갱신된 정점에서 출발하는 간선에 대해

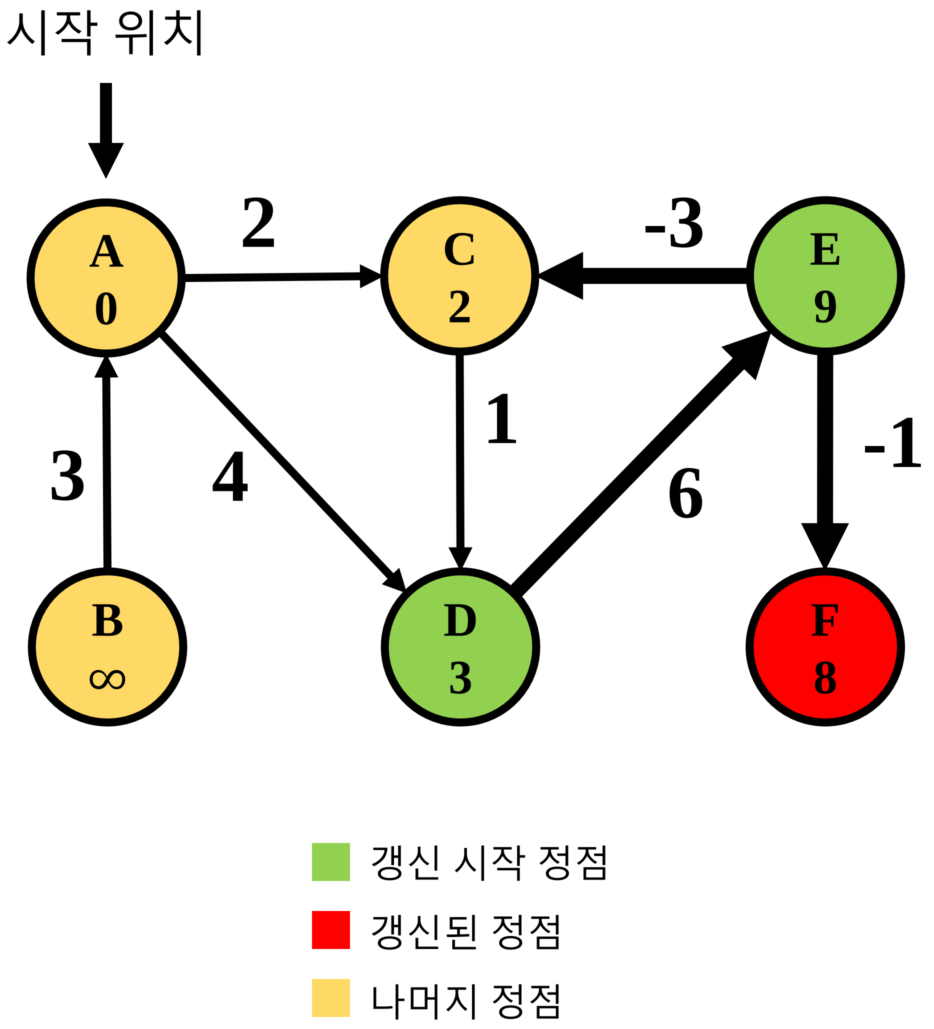
도착지점의 최단거리 갱신



Bellman-Ford 벨만 포드

갱신 반복 횟수: 3

정점 번호	A	B	C	D	E	F
최단거리	0	∞	2	3	9	8



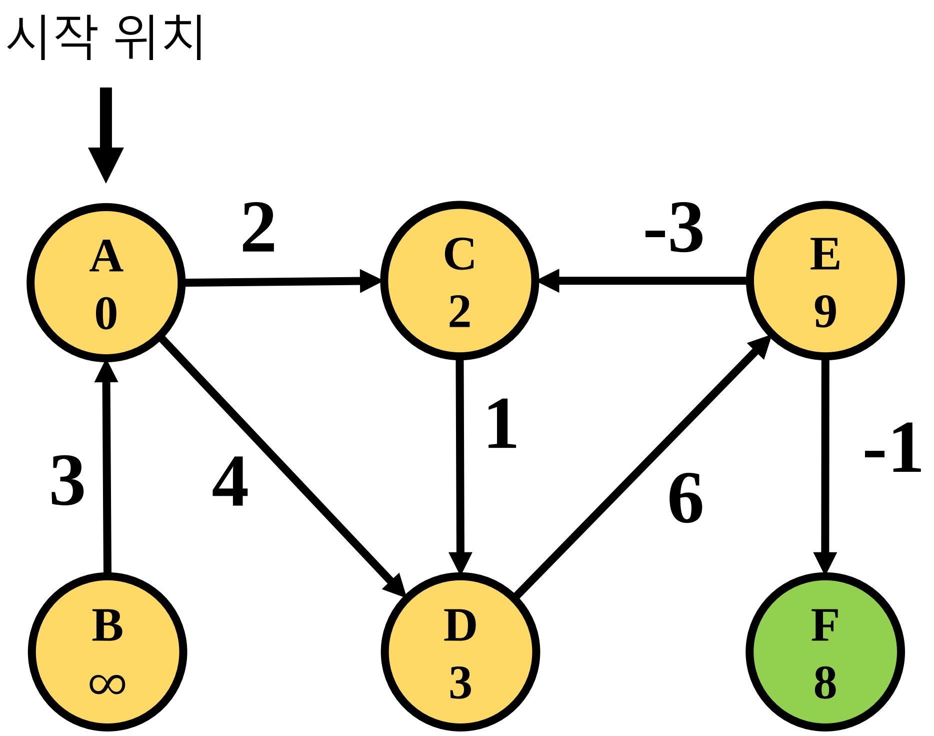
갱신된 정점에서 출발하는 간선에 대해

도착지점의 최단거리 갱신

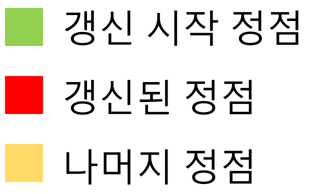
Bellman-Ford 벨만 포드

갱신 반복 횟수: 4

정점 번호	A	B	C	D	E	F
최단거리	0	∞	2	3	9	8



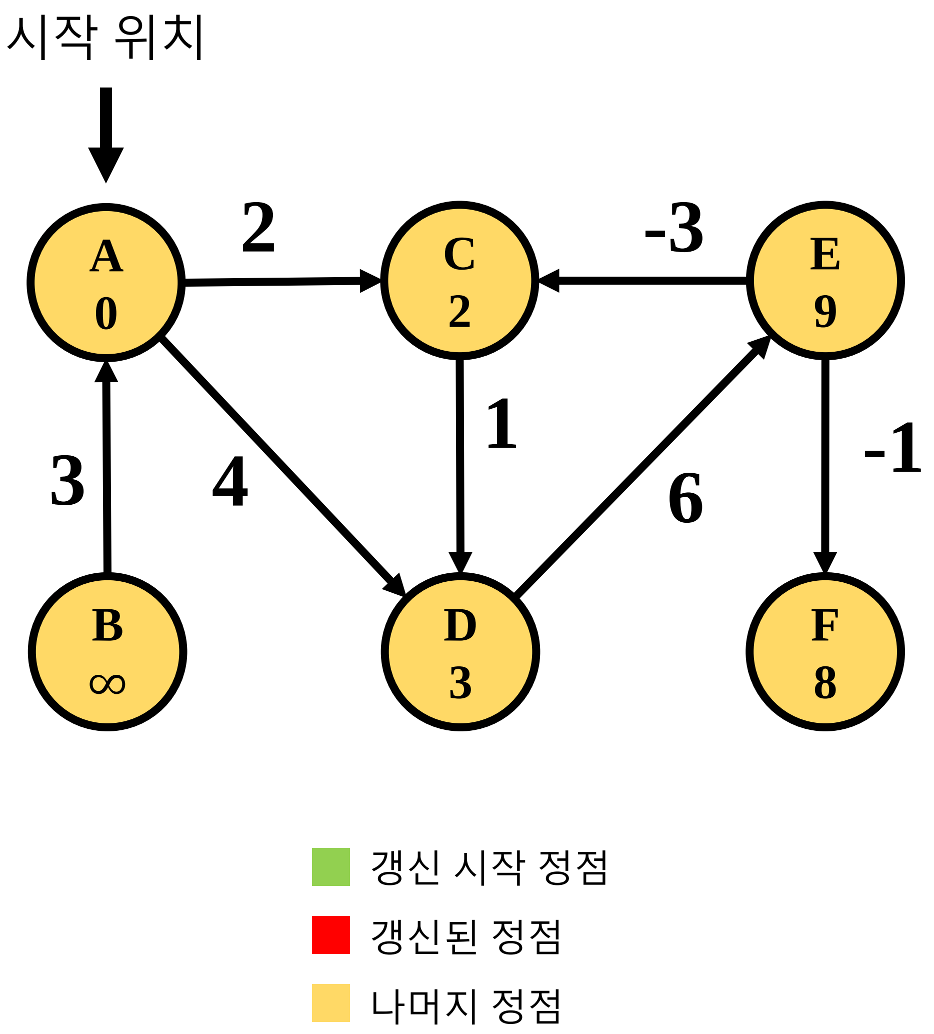
갱신된 정점에서 출발하는 간선에 대해
도착지점의 최단거리 갱신



Bellman-Ford 벨만 포드

갱신 반복 횟수: 5

정점 번호	A	B	C	D	E	F
최단거리	0	∞	2	3	9	8

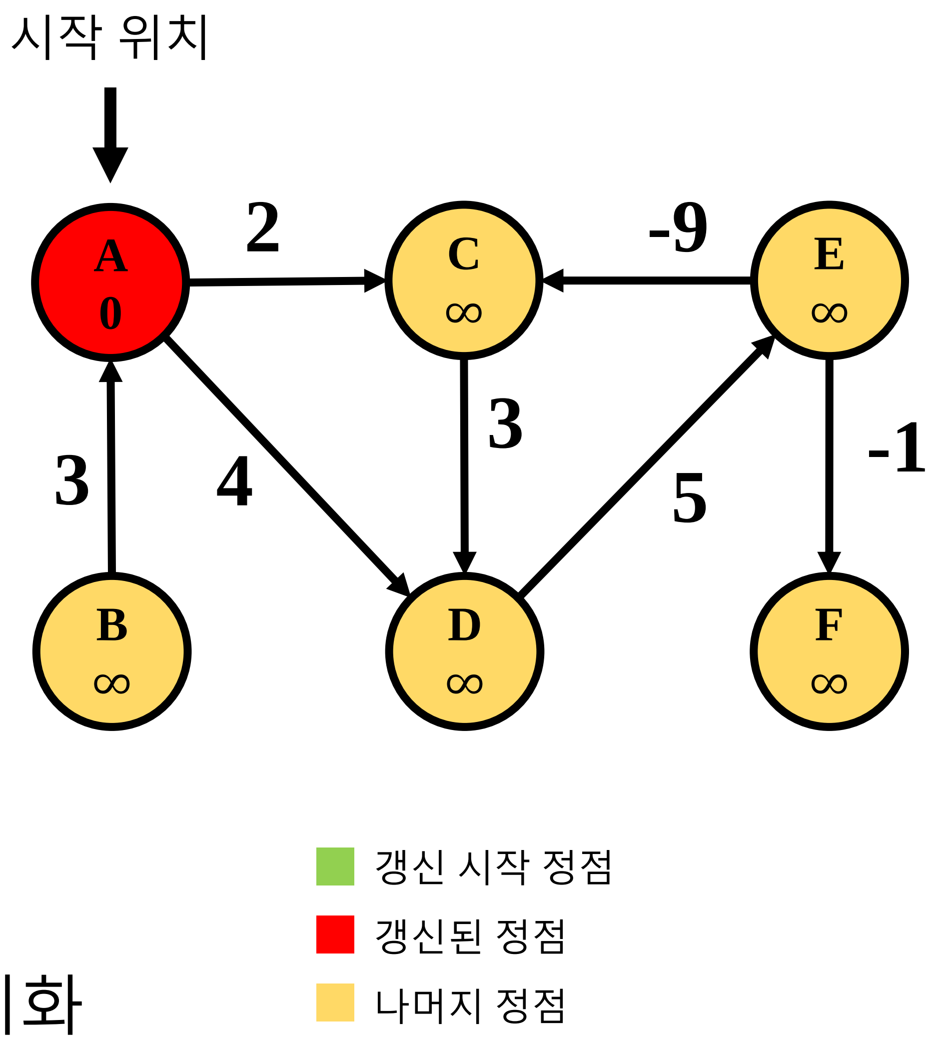


V – 1번째 이후에 갱신되는 정점이 없다
→ 음수 사이클이 존재하지 않는다!

Bellman-Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 0

정점 번호	A	B	C	D	E	F
최단거리	0	∞	∞	∞	∞	∞



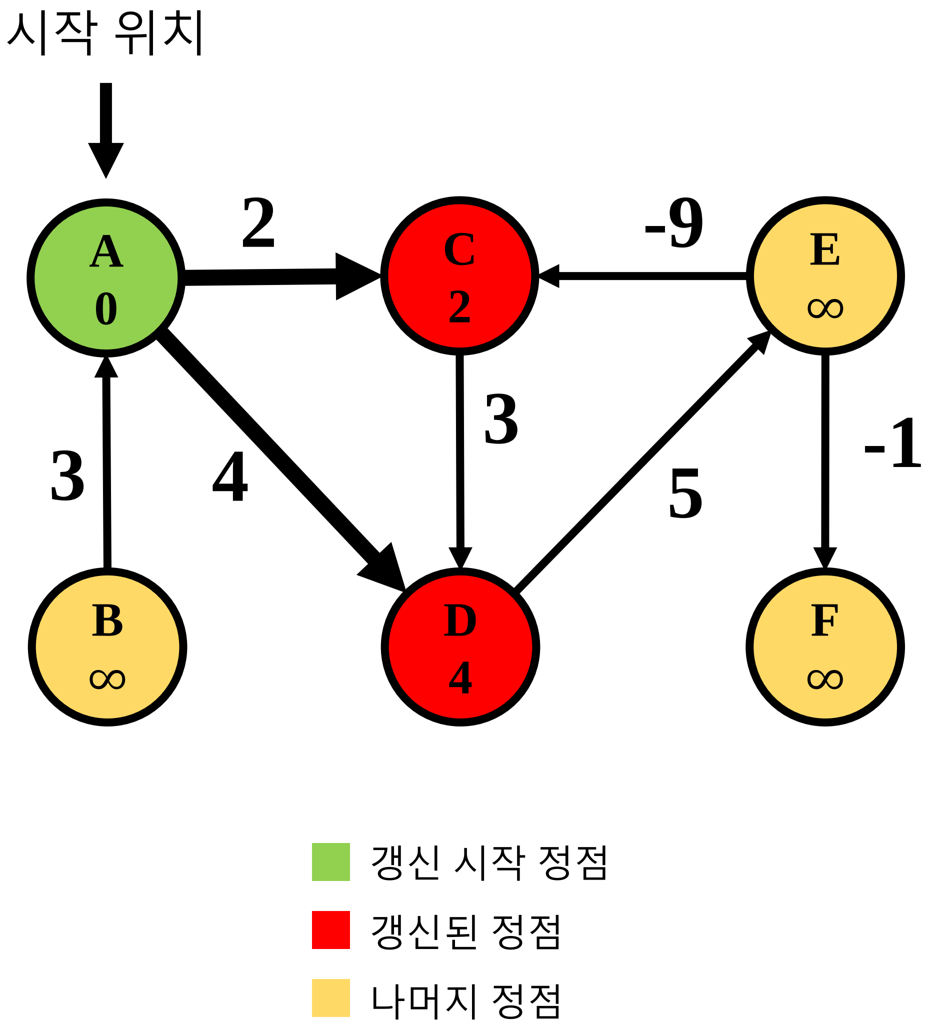
A의 최단 경로를 0,

나머지 정점의 최단 경로는 INF(무한)으로 초기화

Bellman-Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 1

정점 번호	A	B	C	D	E	F
최단거리	0	∞	2	4	∞	∞



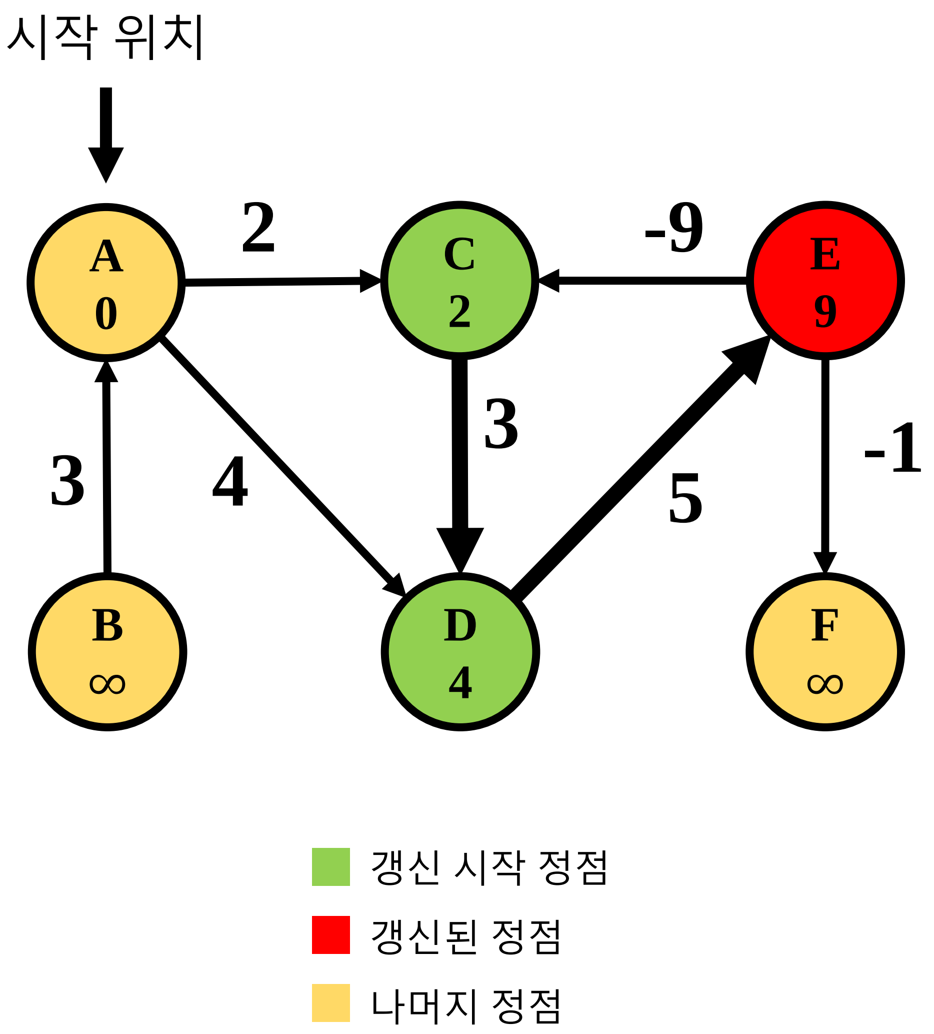
갱신된 정점에서 출발하는 간선에 대해

도착지점의 최단거리 갱신

Bellman-Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 2

정점 번호	A	B	C	D	E	F
최단거리	0	∞	2	4	9	∞



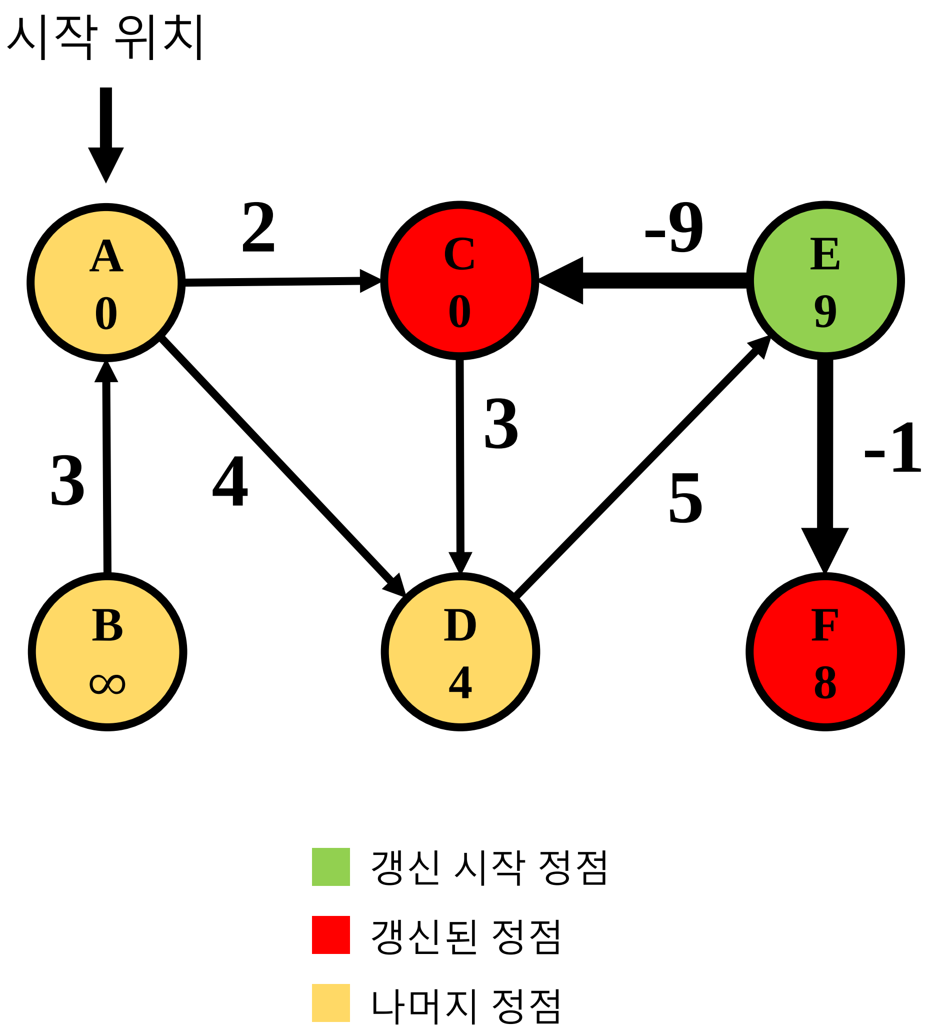
갱신된 정점에서 출발하는 간선에 대해

도착지점의 최단거리 갱신

Bellman-Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 3

정점 번호	A	B	C	D	E	F
최단거리	0	∞	0	4	9	8



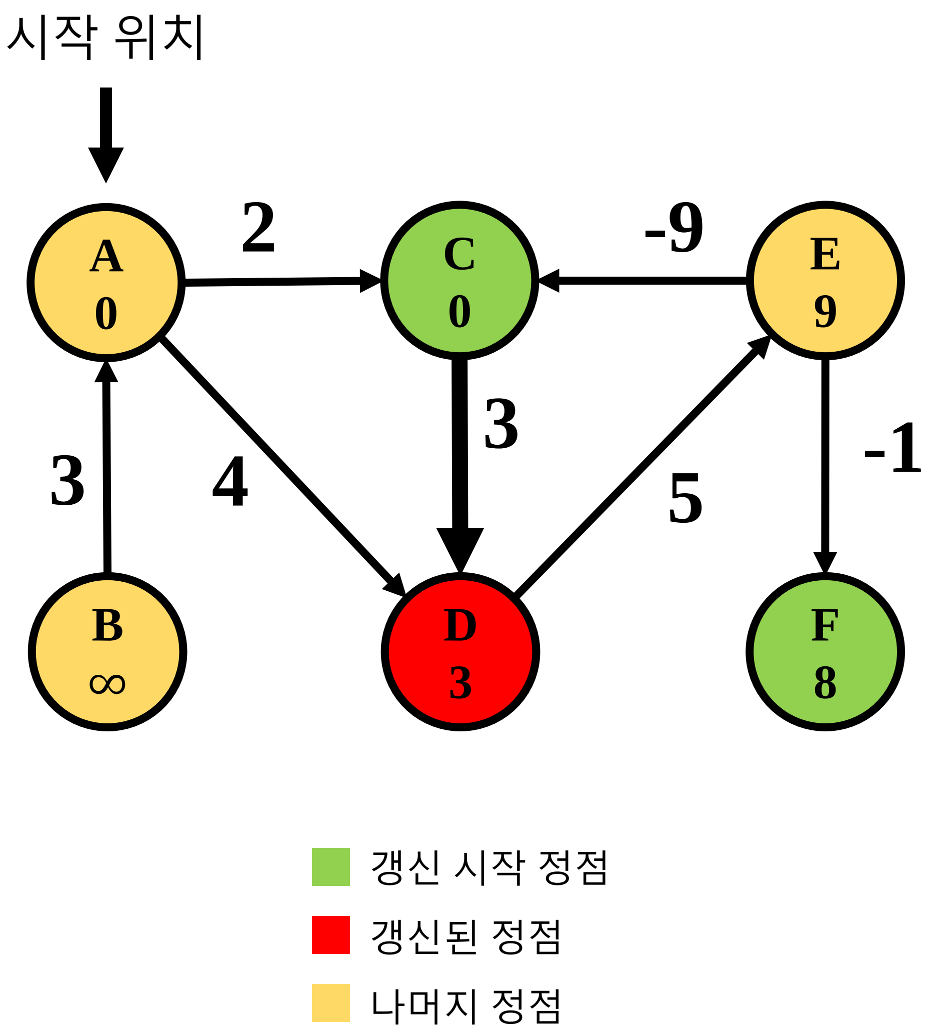
갱신된 정점에서 출발하는 간선에 대해

도착지점의 최단거리 갱신

Bellman-Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 4

정점 번호	A	B	C	D	E	F
최단거리	0	∞	0	3	9	8



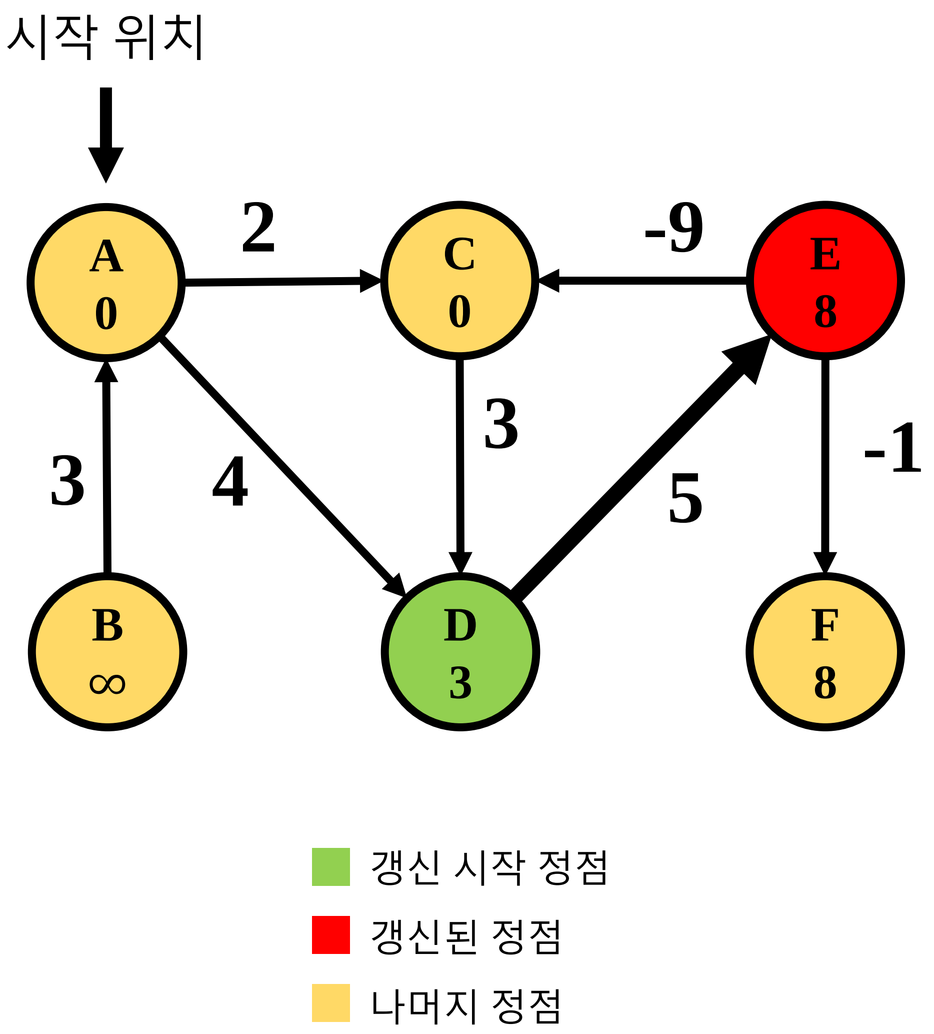
갱신된 정점에서 출발하는 간선에 대해

도착지점의 최단거리 갱신

Bellman-Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 5

정점 번호	A	B	C	D	E	F
최단거리	0	∞	0	3	8	8

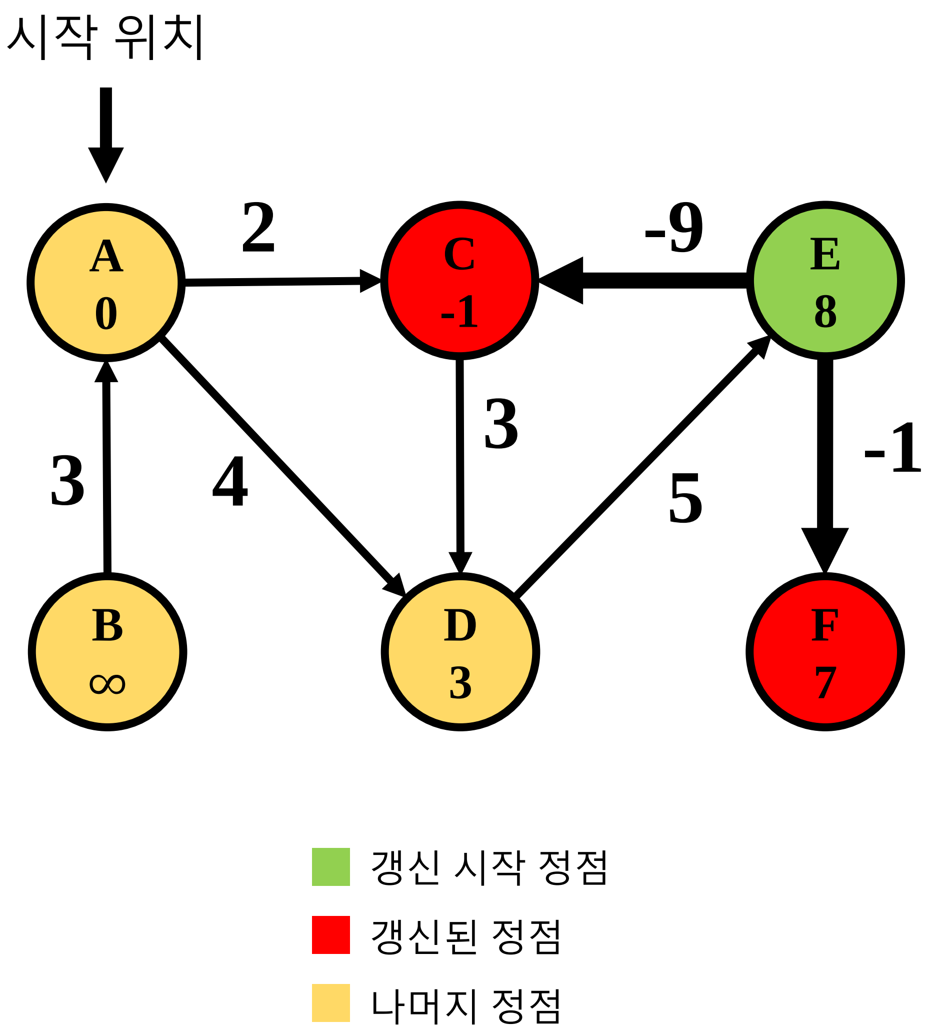


갱신된 정점에서 출발하는 간선에 대해
도착지점의 최단거리 갱신

Bellman-Ford 벨만 포드 – with 음수 사이클

갱신 반복 횟수: 6

정점 번호	A	B	C	D	E	F
최단거리	0	∞	-1	3	8	7



V번째에 갱신된다 (C, F 갱신)

→ 음수 사이클이 **존재**한다!

Bellman-Ford 문제풀이

벨만 포드 기본 문제!

백준 11657
타임머신

11657 타임머신

문제: 음수 가중치가 있는 그래프에서 최단 경로 구하기

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

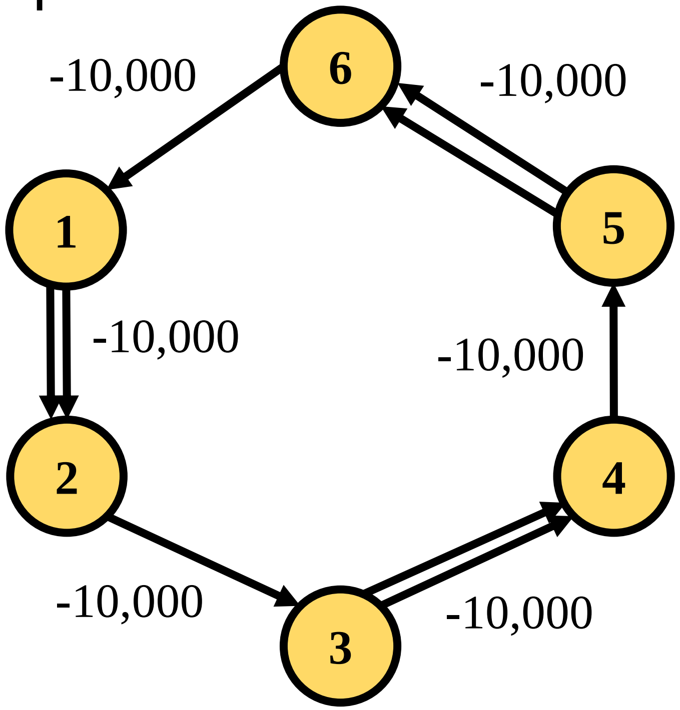
※ 주의할 점

⇒ 가중치가 전부 -10,000이고 고리형태이면 최단 거리는

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

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

⇒ 최단 거리 배열의 자료형이 **long long**이어야 한다



타임머신 코드

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        ll a, b, c; cin >> a >> b >> c;
        edges.push_back({ { a, b }, c });
    }
    if (bf()) { // 음수 사이클이 있으면
        cout << "-1\n";
        return 0;
    }
    for (int i = 2; i <= n; i++) {
        if (distan[i] == INF)
            cout << "-1\n";
        else cout << distan[i] << "\n";
    }
    return 0;
}
```

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#define INF 987654321
using namespace std;
using ll = long long;

int n, m;
vector<pair<pair<ll, ll>, ll>> edges;
ll distan[505]; // underflow 조심!!

int bf() {
    for (int i = 1; i <= n; i++) {
        distan[i] = INF;
    } // 초기화
    distan[1] = 0;
    for (int i = 1; i < n; i++) {
        for (pair<pair<ll, ll>, ll> edge : edges) {
            ll start = edge.first.first; // 시작 정점
            ll end = edge.first.second; // 도착 정점
            ll weight = edge.second; // 가중치
            if(distan[start] != INF) // 갱신
                distan[end] = min(distan[end], distan[start] + weight);
        }
    }
    for (pair<pair<ll, ll>, ll> edge : edges) {
        ll start = edge.first.first;
        ll end = edge.first.second;
        ll weight = edge.second;
        if (distan[start] != INF && distan[end] > distan[start] + weight) {
            return 1; // n - 1번 이후 갱신이 있으면 음수 사이클 존재
        }
    }
    return 0; // 음수 사이클 없음
}
```

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

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <cstring>
#include <queue>
#define INF 987654321
using namespace std;
using ll = long long;

int n, m;
int inQueue[505]; // 큐에 들어있는지 확인
int cycle[505]; // 정점 방문 횟수 체크
ll distan[505];
vector<pair<ll, ll>> edges[505]; // 도착정점, 가중치
```

타임머신 코드 using SPFA

```cpp
int bf() {
    memset(cycle, 0, sizeof(cycle));
    for (int i = 1; i <= n; i++) {
        distan[i] = INF;
    }
    queue<ll> q;
    distan[1] = 0;
    inQueue[1] = 1;
    cycle[1]++;
    q.push(1); // 큐에 넣기
    while (!q.empty()) {
        int con = q.front();
        q.pop();
        inQueue[con] = 0;
        for (pair<ll, ll> i : edges[con]) {
            if (distan[i.first] > distan[con] + i.second) { // 갱신되면
                distan[i.first] = distan[con] + i.second;
                if (inQueue[i.first] == 0) { // 큐 안에 없으면
                    inQueue[i.first] = 1;
                    cycle[i.first]++;
                    q.push(i.first); // 넣기
                }
                if (cycle[i.first] == n) // 정점을 충분히 많이 방문했을때
                    return 1;
            }
        }
    }
    return 0;
}
```

타임머신 코드 using SPFA

```
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        ll a, b, c; cin >> a >> b >> c;
        edges[a].push_back({ b, c });
    }
    if (bf()) {
        cout << "-1\n";
        return 0;
    }
    for (int i = 2; i <= n; i++) {
        if (distan[i] == INF)
            cout << "-1\n";
        else cout << distan[i] << "\n";
    }
    return 0;
}
```

최단 경로 알고리즘 정리

	다익스트라	플로이드 와샬	벨만 포드
알 수 있는 최단 경로	한 정점에서의 다른 모든 정점까지의 최단 경로	모든 정점 사이의 최단경로	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

5 17396 백도어

4 1504 특정한 최단 경로

3 1238 파티

Problem Set – 플로이드 와샬

5 2660 회장뽑기

3 11780 플로이드2

2 2610 회의준비

Problem Set – 벨만 포드

4 1865 웜홀

2 1219 오민식의 고민

Problem Set – 도전!

5 5719 거의 최단 경로

5 3860 할로윈 묘지