ALOHA
#4주차

최단 경로

#CH.0

Graph (그래프)

Graph

3주차 내용을 복습해주세요!

(그래프 복습은 필수)

#CH.0

std::priority_queue (STL 우선순위 큐)

priority_queue

- 큐는 큐인데 front에 항상 가장 큰 값이 들어있다.

  (데이터를 push하면 알아서 정렬해주는 좋은 친구!)

- #include <queue> 필요

- 선언방법: std::priority_queue<자료형> 이름;

priority_queue

- O(log N)의 시간 복잡도로 연산을 수행합니다

  (2-1 자료구조 수업에서 배웁니다. -> Heap)

- 알고리즘 구현에서의 "활용"이 중요하므로 강의자료에서는 자세히 다루지 않겠습니다. (자세한 부분은 구글링을 이용해 보아요)

풀어볼까유

#1927 최소 힙

우선순위 큐를 사용해봐요

priority_queue 예제 (BOJ #1927)

- 다음 슬라이드에 해답이 있습니다.

- 오늘 배우는 중요한 내용은 아니고 pq사용법을 위한 문제이니 베끼는 것에 죄책감을 느끼지 마시고 맘껏 베껴요 ㅎㅎㅎ

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

priority_queue<int> pq; // 내림차순
priority_queue<int, vector<int>, less<int>> pq2; // 내림차순
priority_queue<int, vector<int>, greater<int>> pq3; // 오름차순

int N, x;

int main(void)
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> N;
    for (int n = 1; n <= N; n++) {
        cin >> x;
        if (x == 0) {
            if (pq.empty()) {
                cout << "0\n";
            }
            else {
                cout << -pq.top() << '\n';
                pq.pop();
            }
        }
        else {
            pq.push(-x);
        }
    }

    return 0;
}
```

출력할 때 -1 곱함

입력할 때 -1 곱함

기본적으로 priority_queue는 MaxHeap입니다.
pq안에 내림차순으로 정렬되지요.
그런데 우리는 가장 작은 값을 찾으려고 합니다.
즉 오름차순으로 정렬해야 하죠.

입력이 자연수 뿐이므로
pq에 -1을 곱한 수를 넣어주고
뺄 때 (-)부호를 없애주면
정렬방법 변경 없이 편하게 할 수 있겠죠???

코드에 있는 pq2, pq3 처럼 정렬방법을 선언 시에 설정할 수도 있어요.
(내림차순: MaxHeap / 오름차순: MinHeap)

#CH.1

Dijkstra's Algorithm (다익스트라)

Dijkstra's Algorithm

정해진 하나의 시작 정점에서부터 다른 모든 정점으로 가는
최단 경로를 구하는 알고리즘

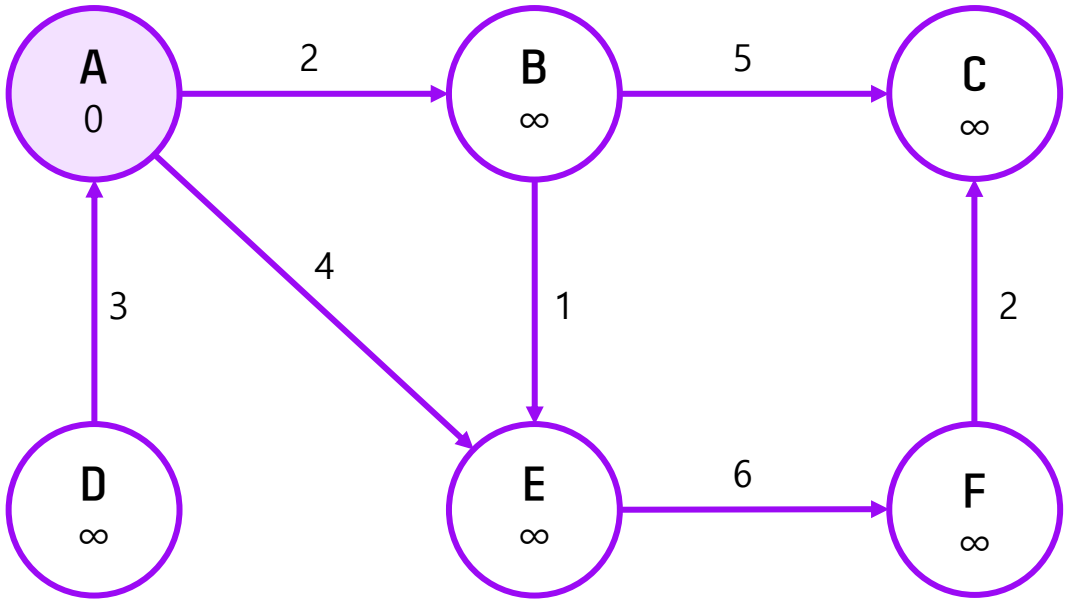
- 음수 가중치의 간선이 있다면 사용할 수 없다!
- 시간 복잡도 O(E log V) (E: 간선 개수 edge / V: 정점 개수 vertex)

Dijkstra's Algorithm

1. 출발 정점에서 출발 정점까지의 최단 거리를 0으로 확정한다.
2. 최단 거리가 확정된 정점과 가장 가까운 정점을 찾아 그 지점까지 최단 거리를 확정한다.
3. 2를 끝날 때까지 반복한다.

최단 거리가 확정된 정점에서 가장 가까운 정점을 찾는데 앞서 배운 우선순위 큐를 쓸 거예요.
BFS가 정점 간의 최단거리를 구할 때 사용된다는 것이 기억나시나요?
다익스트라 알고리즘은 쉽게 생각하면 BFS에서 큐 대신 우선순위 큐를 사용해 Greedy적으로 간선의 가중치(거리)를 처리함으로써 효율적으로 최단경로를 확정하는 알고리즘입니다.

Dijkstra's Algorithm



priority_queue

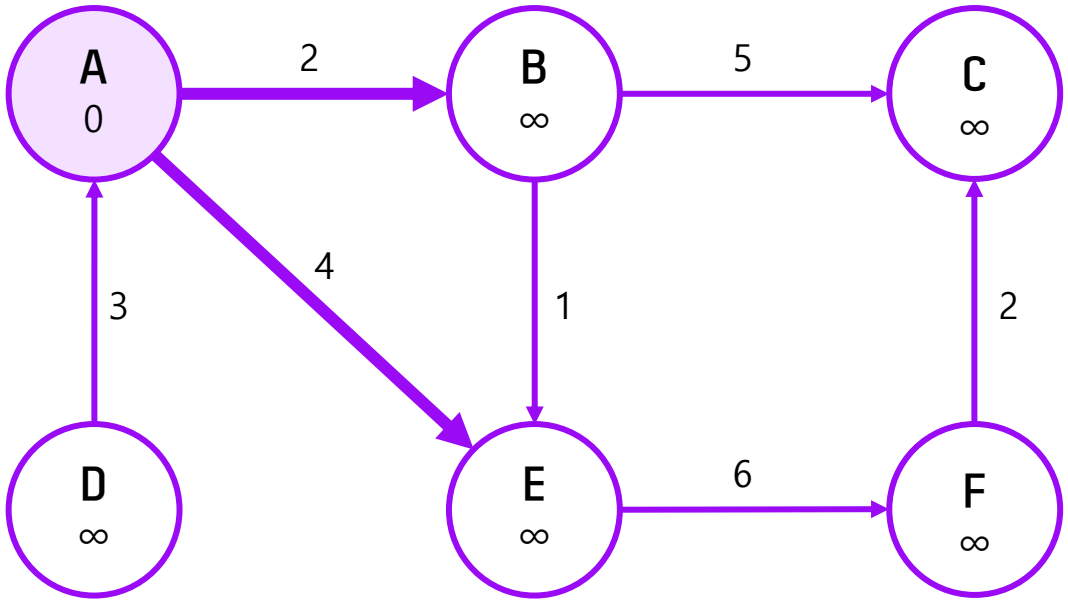
top →

from	to	distance
A	A	0

1단계
출발 정점 A에서 출발 정점 A까지의 최단 거리를 0으로 확정한다.

출발 정점: A
최단거리가 확정되지 않았거나,
도달할 수 없는 정점은 ∞로 표시

Dijkstra's Algorithm



priority_queue

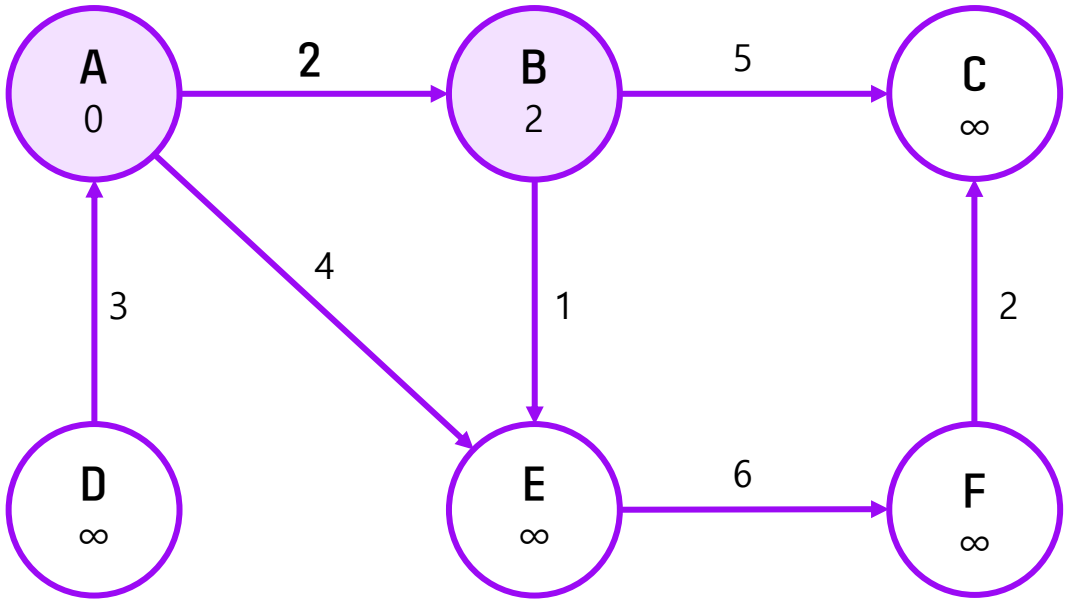
top →

from	to	distance
A	B	2
A	E	4

2단계
최단 거리가 확정된 A에서
출발하는 간선을 pq에 넣는다.

이때 pq는 간선 가중치의
오름차순으로 정렬한다.

Dijkstra's Algorithm



priority_queue

top →

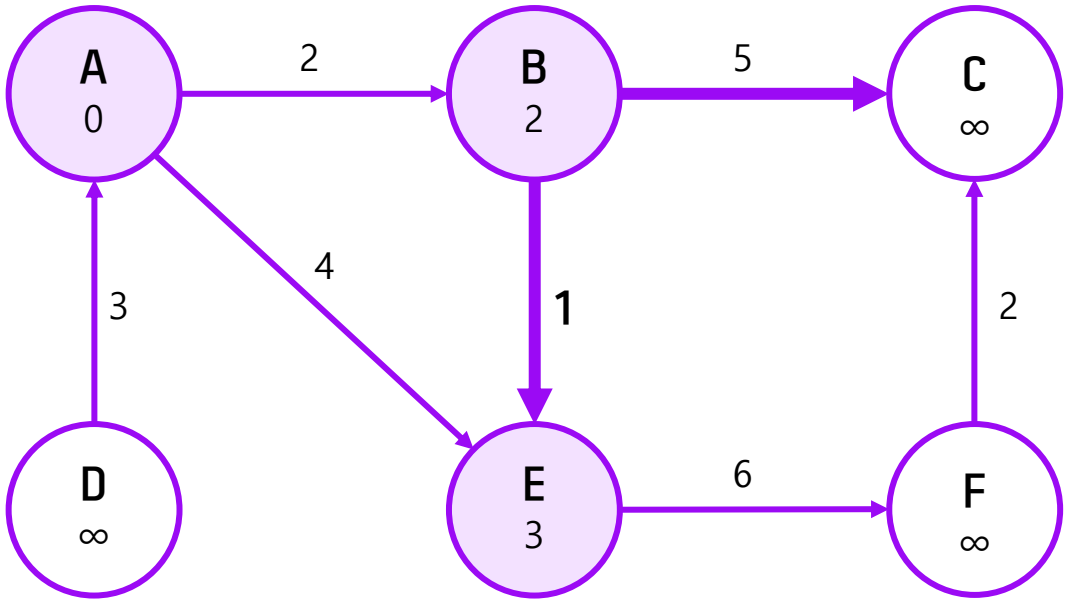
from	to	distance
A	B	2
A	E	4

3단계
최단 거리가 확정된 정점 A에서
가장 가까운 정점을 찾아 (top의 경로)
그것의 최단 거리를 확정한다.

B에 처음 도착했으므로
B까지의 최단 거리는 2가 된다.

top을 이용했다면, pop!

Dijkstra's Algorithm



priority_queue

top →

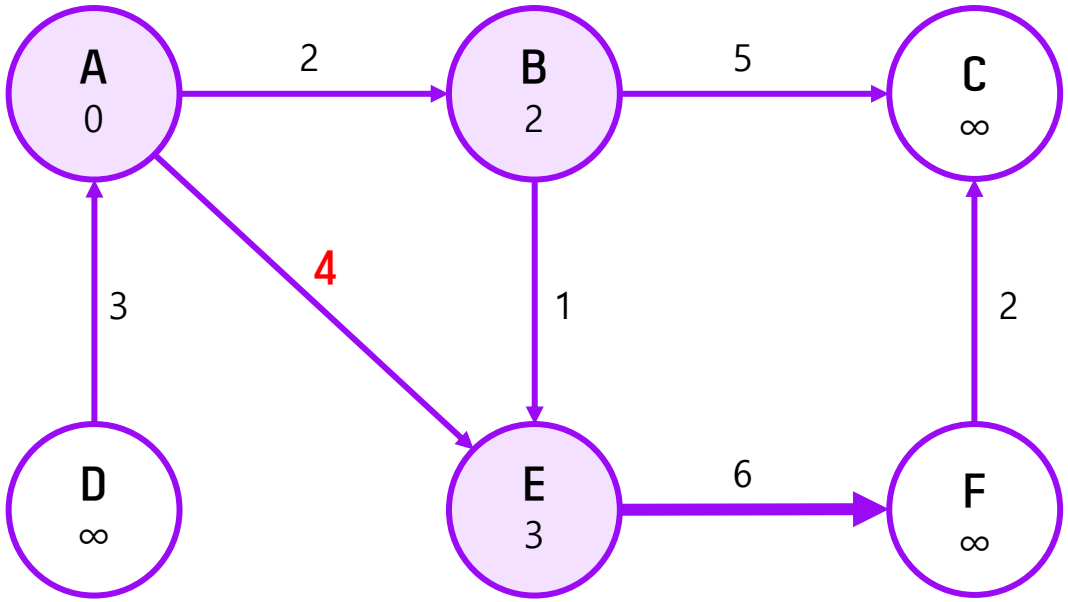
from	to	distance
B	E	2+1
A	E	4
B	C	2+5

4단계 (반복)
최단 거리가 확정된 정점에서
출발하는 간선을 pq에 push한다.

top에 있는 친구들을 하나씩 pop하면서
최단 거리를 확정한다.

E에 처음 도달했으므로
E까지의 최단거리는 3이 된다.

Dijkstra's Algorithm



priority_queue

top →

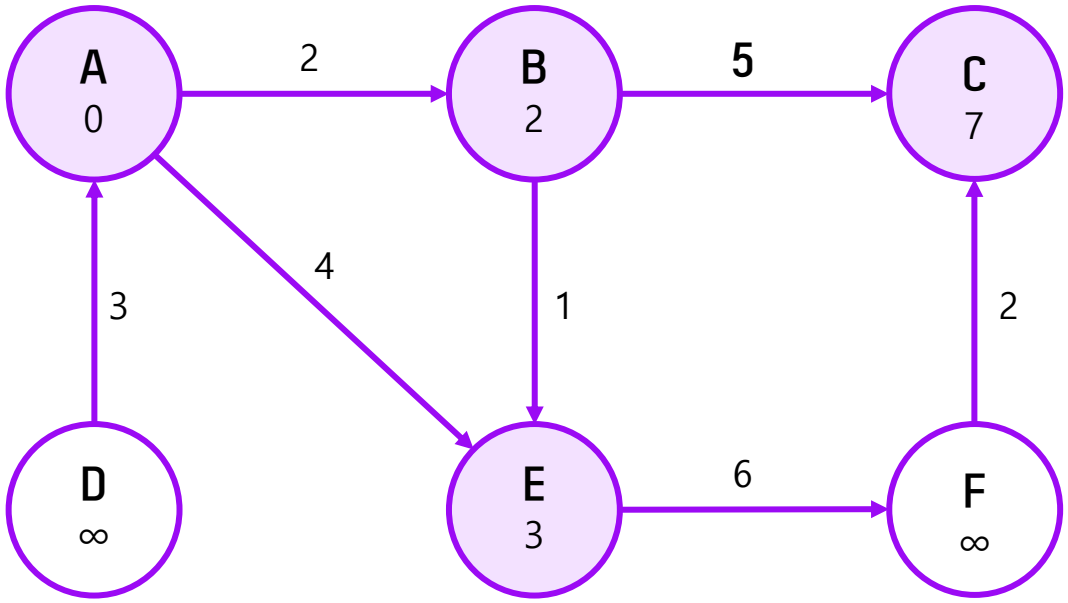
from	to	distance
A	E	4
B	C	2+5
E	F	3+6

4단계 (반복)
최단 거리가 확정된 정점에서
출발하는 간선을 pq에 push한다.

top에 있는 친구들을 하나씩 pop하면서
최단 거리를 확정한다.

E까지의 최단 거리는
이미 3으로 확정되었으므로
A -> E 간선을 통해 도착하면 더 멀다.
따라서 **갱신하지 않고 넘어간다.**

Dijkstra's Algorithm



priority_queue

top →

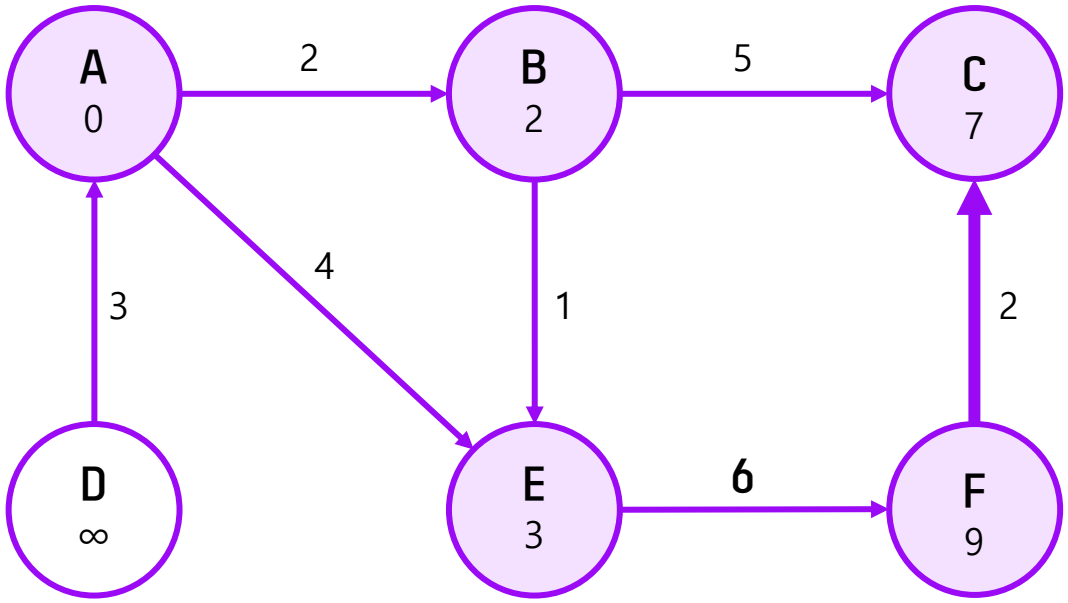
from	to	distance
B	C	2+5
E	F	3+6

4단계 (반복)
최단 거리가 확정된 정점에서
출발하는 간선을 pq에 push한다.

top에 있는 친구들을 하나씩 pop하면서
최단 거리를 확정한다.

C에 처음 도달했으므로
C까지의 최단거리는 7이 된다.

Dijkstra's Algorithm



priority_queue

top →

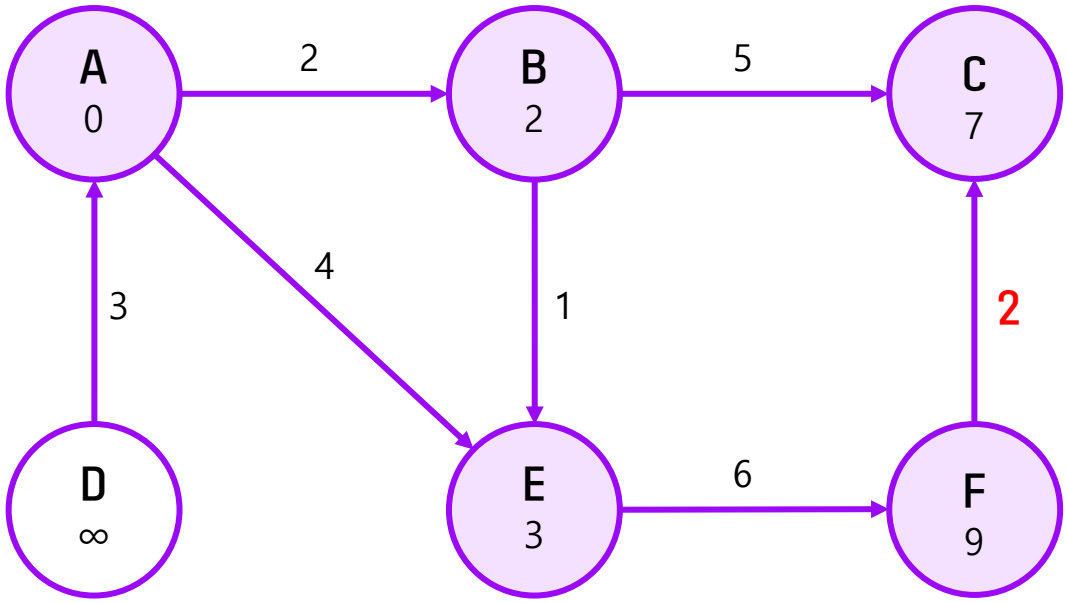
from	to	distance
E	F	3+6
F	C	9+2

4단계 (반복)
최단 거리가 확정된 정점에서
출발하는 간선을 pq에 push한다.

top에 있는 친구들을 하나씩 pop하면서
최단 거리를 확정한다.

F에 처음 도달했으므로
F까지의 최단거리는 9가 된다.

Dijkstra's Algorithm



priority_queue

top →

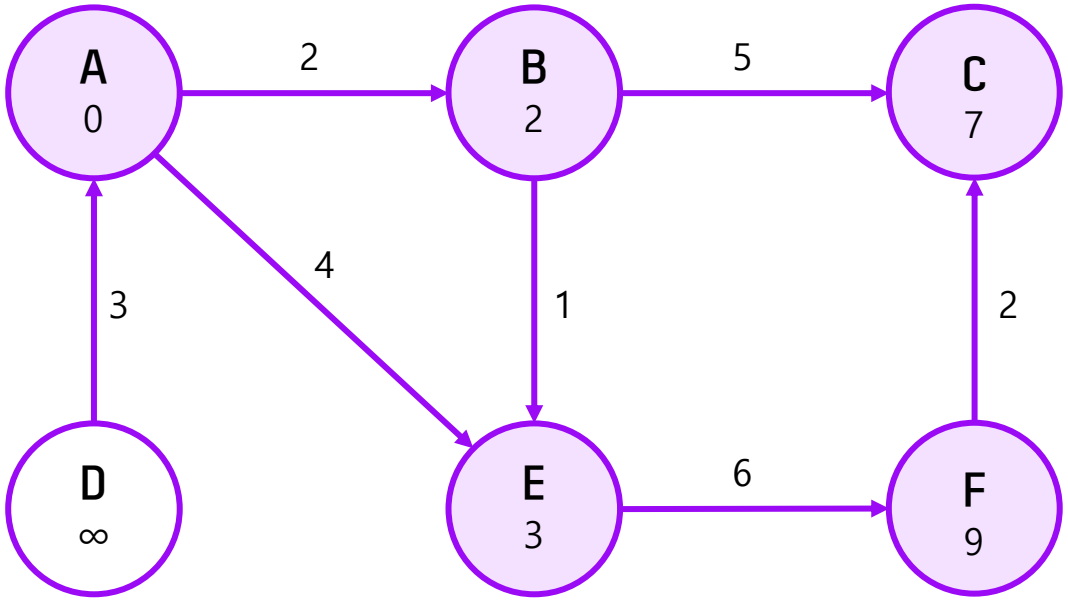
from	to	distance
F	C	9+2

4단계 (반복)
최단 거리가 확정된 정점에서
출발하는 간선을 pq에 push한다.

top에 있는 친구들을 하나씩 pop하면서
최단 거리를 확정한다.

C까지의 최단 거리는
이미 7로 확정되었으므로
F -> C 간선을 통해 도착하면 더 멀다.
따라서 **갱신하지 않고 넘어간다.**

Dijkstra's Algorithm



A to	distance
A	0
B	2
C	7
D	∞
E	3
F	9

D는 A에서 도달할 수 없다.

고로 최단 거리는 ∞로 표현할 수 있다.

```cpp
const int INF = 2'000'000'009; // 굉장히 큰 수
const int V_MAX = 1'000'006; // 정점 최대 개수

typedef pair<int,int> Edge;
vector<Edge> edges[V_MAX];
int dist[V_MAX];

void Dijkstra(int start)
{
    fill(dist,dist+V_MAX,INF); // dist[]를 INF로 초기화
    priority_queue<Edge> PQ; // #include <queue> 필요
    PQ.push({0,start}); // pair<int,int> 형태로 push

    while (!PQ.empty())
    {
        int u, v, d, w;
        tie(d, u) = PQ.top(); // #include <tuple> 필요
        PQ.pop();

        if (dist[u] < INF) continue;
        dist[u] = -d; // PQ가 내림차순 이므로 거리에 (-)붙임

        for (Edge e : edges[u]) {
            tie(v, w) = e;
            if (dist[v] < INF) continue;
            PQ.push({d - w, v});
        }
    }
}
```

거리의 부호를 주의하세요!

코드로 확인해봅시다!

dist[]: 최단 거리를 저장하는 배열

vector<Edge> edges[]: 인접 리스트 (3주차 참조)

Edge: pair<to 정점 번호, 간선 가중치>

(edges[3] == {5,4} 이면 3 -> 5 간선의 가중치(거리)가 4)

priority_queue<Edge> PQ: 우선순위 큐

Edge: pair<-(시작 정점으로부터의 거리), to 정점 번호>

(PQ.top() == {-11,5} 이면 시작 정점에서 5번 정점까지 최단 거리가 11)

INF: 무한대 (엄청 큰 수)

PQ에서 pair가 정렬될 때,

0) 기본이 내림차순 (-를 붙여서 push, pop하면 오름차순)

1) first 값 기준 정렬

2) first 값이 같다면 second 값 기준 정렬

```cpp
const int INF = 2'000'000'009; // 굉장히 큰 수
const int V_MAX = 1'000'006; // 정점 최대 개수

typedef pair<int,int> Edge;
vector<Edge> edges[V_MAX];
int dist[V_MAX];

void Dijkstra(int start)
{
    fill(dist,dist+V_MAX,INF); // dist[]를 INF로 초기화
    priority_queue<Edge> PQ; // #include <queue> 필요
    PQ.push({0,start}); // pair<int,int> 형태로 push

    while (!PQ.empty())
    {
        int u, v, d, w;
        tie(d, u) = PQ.top(); // #include <tuple> 필요
        PQ.pop();

        if (dist[u] < INF) continue;
        dist[u] = -d; // PQ가 내림차순 이므로 거리에 (-)붙임

        for (Edge e : edges[u]) {
            tie(v, w) = e;
            if (dist[v] < INF) continue;
            PQ.push({d - w, v});
        }
    }
}
```

pop 후에 방문 여부 check!

push 전에 방문 여부 check!

priority_queue를 이용하기 때문에

간선을 넣는 순서와 빼는 순서가 다를 수 있다

-> PQ에서 pop을 할 때 dist를 업데이트 하는 이유 (빼는 순서가 중요)

-> 방문 여부를 넣을 때와 뺄 때 모두 체크하는 이유 (넣을 때 체크가 뺄 때 체크를 보장하지 못함)

간편하게 Greedy를 구현할 수 있다

-> 한번 방문 & 갱신하면 그것이 최단 거리

-> 시간 복잡도가 O(E log V) 가 됨

참고)

PQ에 출발 정점을 저장하거나, 이전 정점을 저장하는 배열을 만듦으로써
최단 거리 뿐만 아니라 말그대로 **최단 경로**를 구할 수 있도록 변형할 수도 있음.

풀어볼까유

#1753 최단 경로

다익스트라를 써봅시다

Dijkstra's Algorithm (BOJ #1753)

문제 그대로 최단 경로를 구해주면 됩니다.

다음 슬라이드에 해답이 있습니다.
혼자서 풀기 힘들거나 잘 안될 때는 자료의 코드를 베껴 보아요.

```cpp
int V, E, K;

int main()
{
    int a, b, w;

    scanf("%d %d %d",&V,&E,&K);
    for (int e = 1; e <= E; e++) {
        scanf("%d %d %d",&a,&b,&w);
        edges[a].push_back({b,w});
    }

    Dijkstra(K);
    for (int v = 1; v <= V; v++) {
        if (dist[v] == INF) printf("INF\n");
        else printf("%d\n",dist[v]);
    }

    return 0;
}
```

Dijkstra 함수만 작성하면 정답!

Dijkstra 함수만 작성하면 정답입니다.

(이 함수... 앞에서 본 것 같은데...)

성공했다면 자신의 스타일로 코드를 바꿔서 다시 제출해보아요.

풀어볼까유

#1238 파티

다익스트라, but 여러 번

풀어볼까유

#1238 백도어

다익스트라,
but 0번부터 시작

풀어볼까유

#5719

거의 최단 경로

다익스트라,
but 어려움

Dijkstra's Algorithm (BOJ #5719)

- 최단 경로를 구한다.
- 최단 경로에 포함되는 모든 간선들을 지운다.
- 다시 최단 경로를 구한다.
- 최단 경로가 1개가 아닐 수도 있다.

풀어볼까유

#1854

K번째 최단경로 찾기

다익스트라,
but 어려움 2

Dijkstra's Algorithm (BOJ #1854)

- 알고리즘을 조금 변형해 봅시다.
- PQ에 더 많은 정보를 저장해야 할 것 같죠?

#CH.2

Floyd-Warshall Algorithm (플로이드-워셜)

Floyd-Warshall Algorithm

모든 정점 사이의 최단 경로를 구하는 알고리즘
모든 경우의 수를 체크함

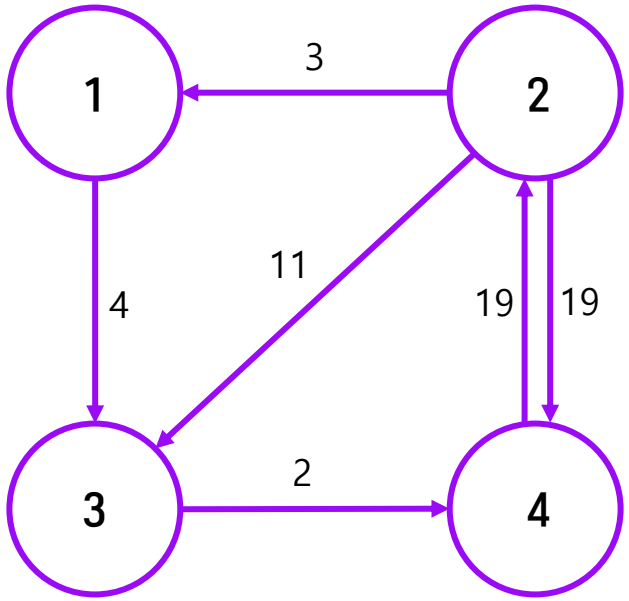
- 음수 가중치가 있어도 처리 가능!
  물론 "음수 사이클"이 있다면, 최단 거리가 -∞일 수 있으므로 제대로 처리할 수 없어요
- 시간 복잡도 O(V^3)

Floyd-Warshall Algorithm

1. i -> j의 최단 거리를 간선의 길이(정보가 없다면 무한대의 값)로 초기화한다.
2. 모든 i -> j 조합에 대해, 특정한 정점 k를 경유해서 가는 i -> k -> j 경로가 더 짧다면 해당 i -> j의 최단 거리를 i -> k -> j의 거리로 갱신한다.
3. 2를 모든 정점 k에 대해 반복한다.

알고리즘 실행 한번으로 모든 정점 간의 최단 경로를 구할 수 있어요.

Floyd-Warshall Algorithm



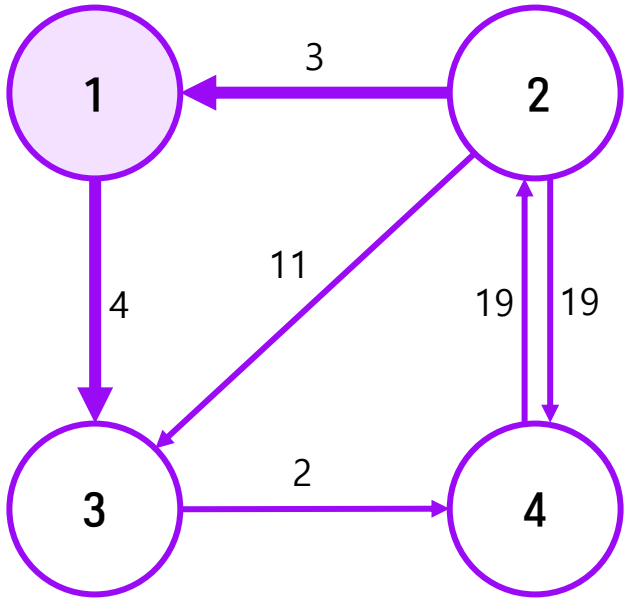
도착 / 출발	1	2	3	4
1	0	INF	4	INF
2	3	0	11	19
3	INF	INF	0	2
4	INF	19	INF	0

1단계 (인접 행렬 초기화)

1) 출발 정점과 도착 정점이 같다면, 최단 거리를 0으로 함.
2) 출발 정점과 도착 정점 사이에 간선이 존재한다면, 최단 거리를 그 간선의 길이로 함.
3) 출발 정점과 도착 정점 사이에 간선이 존재하지 않는 다면, 최단 거리를 무한대의 값으로 함.

INF는 무한대의 값 혹은 정보 없음을 의미

Floyd-Warshall Algorithm



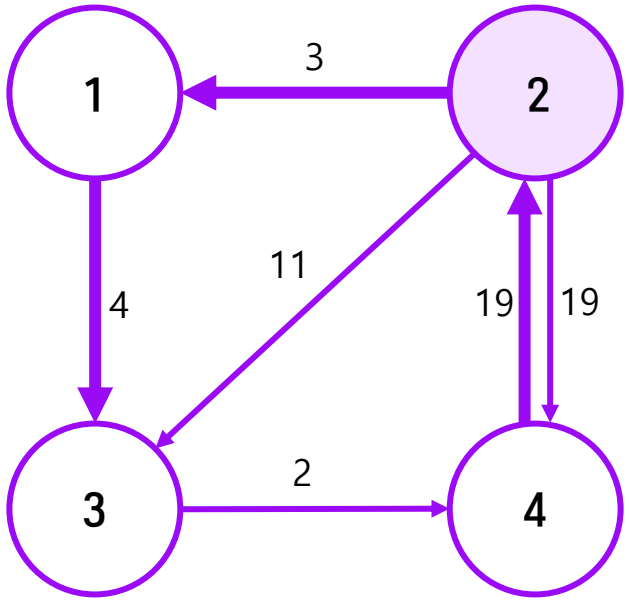
도착 / 출발	1	2	3	4
1	0	INF	**4**	INF
2	**3**	0	**7**	19
3	INF	INF	0	2
4	INF	19	INF	0

2단계 (경유 정점 k = 1)

16개의 모든 경로에 대해, 1번 정점을 경유하는 길로 돌아갔을 때의 거리가 더 짧다면 최단 거리를 갱신한다.

2-> 3의 최단 거리가 2 -> 1 -> 3의 거리로 갱신되었어요!
반면에 4 -> 3의 최단 거리는 4 -> 1의 최단 거리가 INF이므로 갱신되지 않고 여전히 INF 입니다.

Floyd-Warshall Algorithm



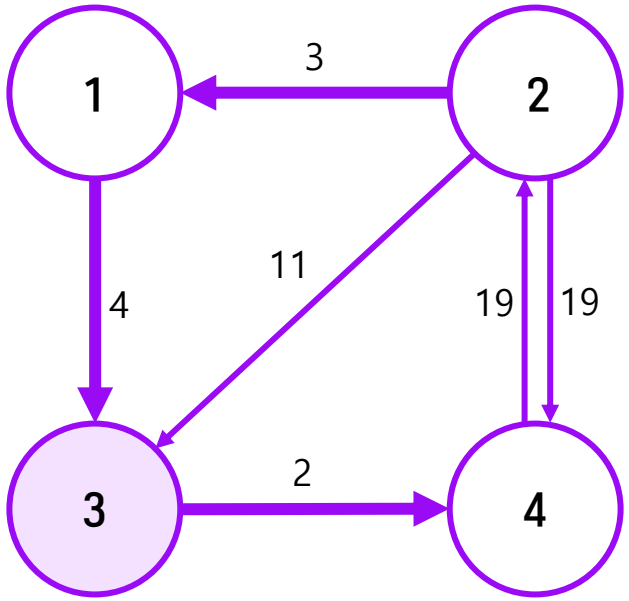
도착 / 출발	1	2	3	4
1	0	INF	4	INF
2	**3**	0	**7**	19
3	INF	INF	0	2
4	**22**	**19**	**26**	0

2단계 (경유 정점 k = 2)

16개의 모든 경로에 대해, 2번 정점을 경유하는 길로 돌아갔을 때의 거리가 더 짧다면 최단 거리를 갱신한다.

4-> 1의 최단 거리가 4 -> 2 -> 1로
4-> 3의 최단 거리가 4 -> 2 -> 3의
거리로 갱신되었어요!

Floyd-Warshall Algorithm



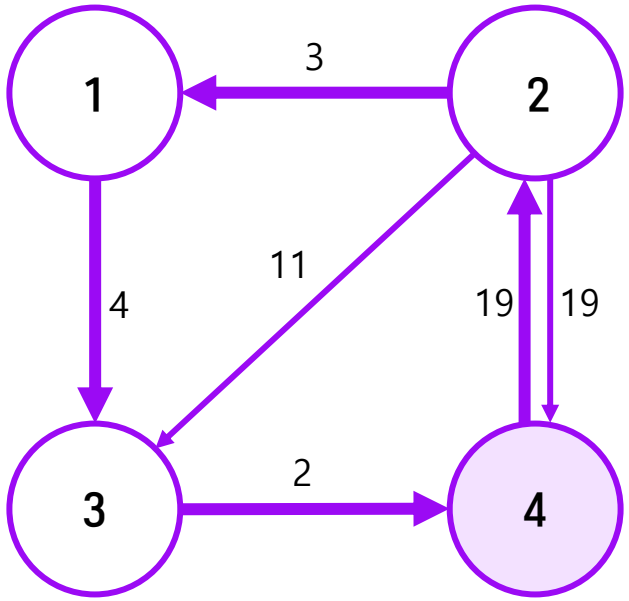
도착 / 출발	1	2	3	4
1	0	INF	4	6
2	3	0	7	9
3	INF	INF	0	2
4	22	19	26	0

2단계 (경유 정점 k = 3)

16개의 모든 경로에 대해, 3번 정점을 경유하는 길로 돌아갔을 때의 거리가 더 짧다면 최단 거리를 갱신한다.

1-> 4의 최단 거리가 1 -> 3 -> 4로
2-> 4의 최단 거리가 2 -> 3 -> 4의 거리로 갱신되었어요!

Floyd-Warshall Algorithm



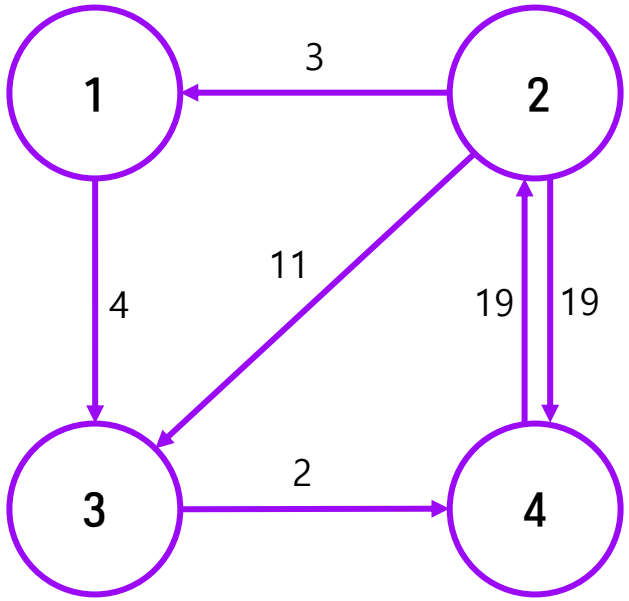
도착 / 출발	1	2	3	4
1	0	25	4	6
2	3	0	7	9
3	24	21	0	2
4	22	19	26	0

2단계 (경유 정점 k = 4)

16개의 모든 경로에 대해, 4번 정점을 경유하는 길로 돌아갔을 때의 거리가 더 짧다면 최단 거리를 갱신한다.

1-> 2의 최단 거리가 1 -> 4 -> 2로
3-> 1의 최단 거리가 3 -> 4 -> 1로
3-> 2의 최단 거리가 3 -> 4 -> 2의 거리로 갱신되었어요!

Floyd-Warshall Algorithm



도착 / 출발	1	2	3	4
1	0	25	4	6
2	3	0	7	9
3	24	21	0	2
4	22	19	26	0

모든 정점에 대한 경유 거리를 확인하면 왼쪽처럼 각 정점 간의 최단 거리를 2차원 배열로 얻을 수 있습니다.

최단 거리가 INF일 경우 해당 정점 사이에 경로가 존재하지 않음을 의미합니다.

```
9   void Floyd_Warshall(void)
10  {
11      for (int k = 1; k <= N; k++) {
12          for (int i = 1; i<= N; i++) {
13              for (int j = 1; j <= N; j++) {
14                  dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
15              }
16          }
17      }
18  }
```

k가 가장 바깥임에 주의하세요!

코드로 확인해봅시다!

dist[i][j]: i에서 j로 가는 최단 거리

(초기 상태는 3주차의 인접 행렬과 동일)

모든 i → j에 대해

k를 경유하는 경로가 더 짧다면

dist[i][j]를 dist[i][k] + dist[k][j]로 갱신

이 때 모든 k에 대해 확인을 하기 때문에

3중 for문에서 가장 바깥이

k에 대한 for문임에 주의하세요!

풀어볼까유

#11404 플로이드

풀어보면 쉬운 문제

Floyd-Warshall Algorithm (BOJ #11404)

최단 "거리" 알고리즘이라고 거리만 생각하면 안돼요!

거리, 시간, 돈 어떠한 개념이 와도 응용할 수 있어야 합니다.

다음 슬라이드에 해답이 있습니다.

혼자서 풀기 힘들거나 잘 안될 때는 자료의 코드를 베껴 보아요.

Floyd-Warshall Algorithm (BOJ #11404)

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

const int N_MAX = 102;
const int INF = N_MAX * 100'000;
int N, dist[N_MAX][N_MAX];
```

```cpp
int main(void)
{
    int a, b, c;

    for (int i = 0; i < N_MAX; i++) {
        for (int j = 0; j < N_MAX; j++) {
            if (i == j) dist[i][j] = 0;
            else dist[i][j] = INF;
        }
    }

    scanf("%d %d",&N,&M);
    for (int m = 0; m < M; m++) {
        scanf("%d %d %d",&a,&b,&c);
        dist[a][b] = min(dist[a][b], c);
    }

    Floyd_Warshall();

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            if (dist[i][j] == INF) dist[i][j] = 0;
            printf("%d ",dist[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Floyd_Warshall함수만 작성하면 정답입니다.

(이 함수 어디서 봤는데…)

성공했다면 자신의 스타일로
코드를 바꿔서 다시 제출해보아요!

풀어볼까유

#2610 회의준비

의사전달시간을 구해보자

풀어볼까유

#11780
플로이드 2

Floyd-Warshall,
but 어려움

#CH.3

Bellman-Ford Algorithm (벨만-포드)

Bellman-Ford Algorithm

하나의 시작 정점에서부터 다른 모든 정점으로 가는
최단 경로를 구하는 알고리즘

- 음수 가중치가 있어도 처리 가능!
  물론 음수 사이클이 있다면, 최단 경로의 총 가중치가 -∞이므로 처리할 수 없어요.. 하지만 이 말은 즉,
- 음수 사이클을 검출할 때 이용 가능!
- 시간 복잡도 O(VE)

Bellman-Ford Algorithm

1. 초기 최단 거리를 **시작 정점은 0**으로
   다른 정점은 무한대의 값으로 설정한다.
2. **모든 간선을 확인하면서 각 간선의 도착 정점까지의**
   최단 거리를 갱신한다. (한번도 최단 거리가 갱신되지 않은 정점에서 출발하는 간선은 넘김)
3. 2번을 V-1번 반복한다. (V: 정점의 개수)

플로이드-워셜에서는 최단 거리를 갱신할 수 있는 **경유 정점**을 찾아 갱신했다면,
벨만-포드는 최단 거리를 갱신할 수 있는 **경유 간선**을 찾아 갱신합니다.

Bellman-Ford Algorithm

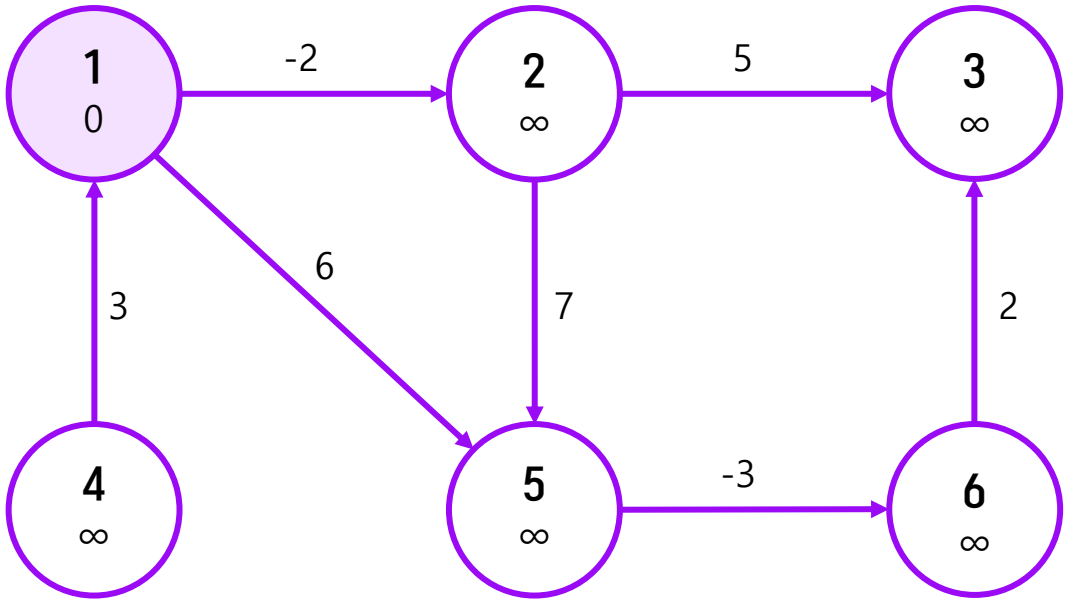
왜 V-1번 반복할까요?

V개의 정점이 있을 때, 한 정점에서 다른 정점으로 가는
경로 상에는 **최대 V-1개의 간선**이 있기 때문입니다.

ex) 1번 정점에서 5번 정점까지 정점을 최대한 거쳐가는 최단경로는
1->2->3->4->5 이렇게 4개의 간선을 거쳐가는 것이 최대이다.
1->2->5 는 최단경로가 될 수는 있지만
1->2->3->2->4->5 는 될 수가 없다.

모든 간선에 대해 V-1번만 반복하면 음수 사이클이 없는 이상
더 이상의 최단 거리 갱신이 일어나지 않습니다.

Bellman-Ford Algorithm

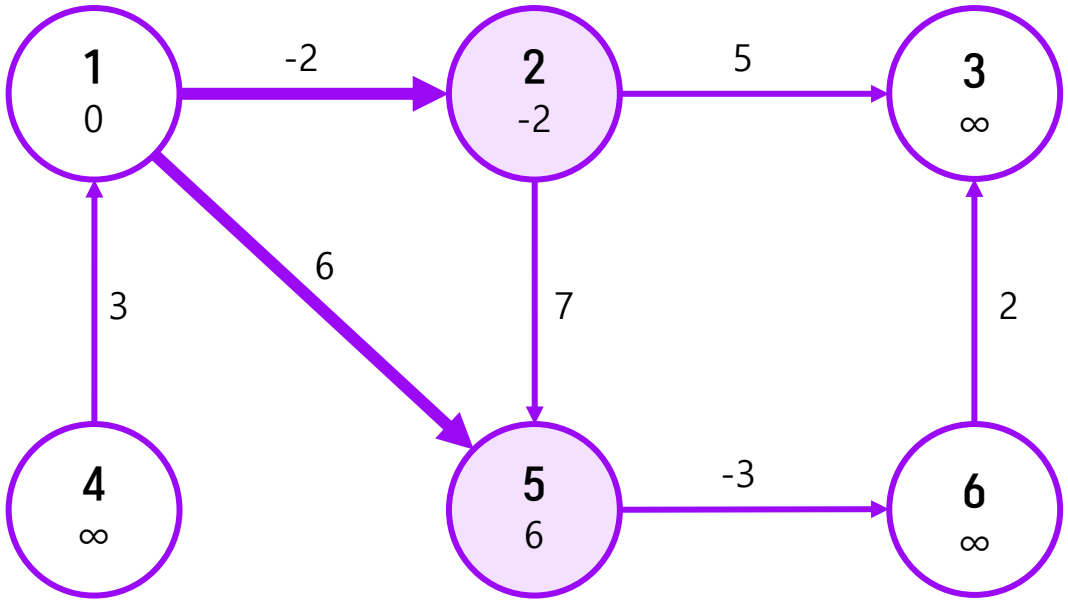


정점 번호	1	2	3	4	5	6
최단거리	0	INF	INF	INF	INF	INF

1단계
출발 정점 A에서 출발 정점 A까지의 최단 거리를 0으로 확정한다.

출발 정점: A
최단거리가 확정되지 않았거나,
도달할 수 없는 정점은 ∞로 표시

Bellman-Ford Algorithm

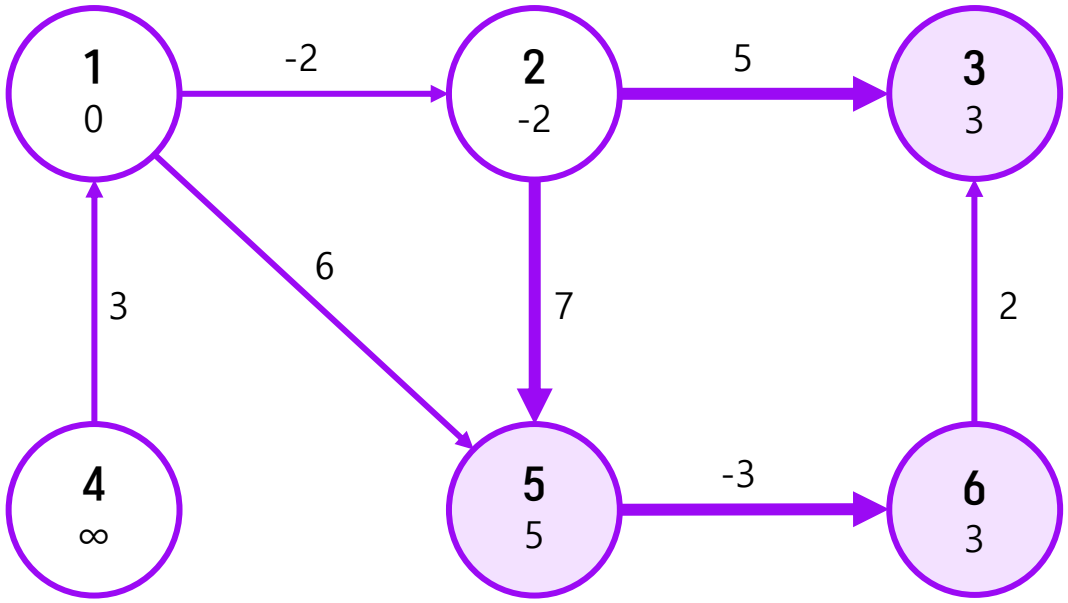


정점 번호	1	2	3	4	5	6
최단거리	0	-2	INF	INF	6	INF

2단계 (반복 1회차)
최단 거리가 갱신된 1번 정점에서 출발하는
두개의 간선에 대해
간선의 도착 정점의 최단 거리를 갱신한다.

1-> 2, 1-> 5 외의 다른 간선은
출발 정점의 최단 거리가
한번도 갱신되지 않았으므로 (INF) 넘김.

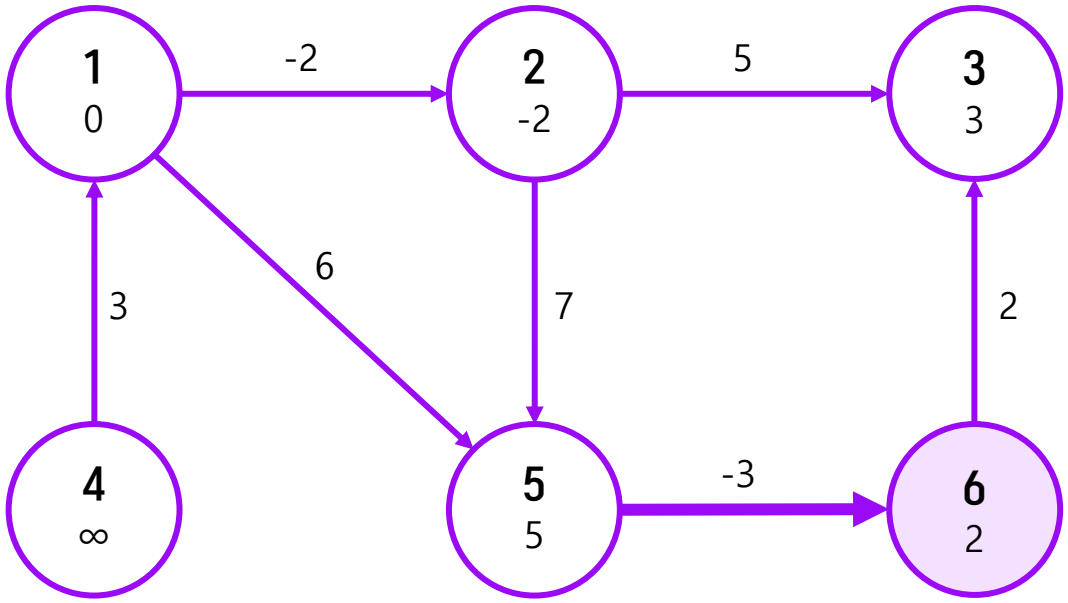
Bellman-Ford Algorithm



2단계 (반복 2회차)
최단 거리가 갱신된 정점에서 출발하는
간선의 도착 정점의 최단 거리를 갱신한다.

정점 번호	1	2	3	4	5	6
최단거리	0	-2	**3**	INF	**5**	**3**

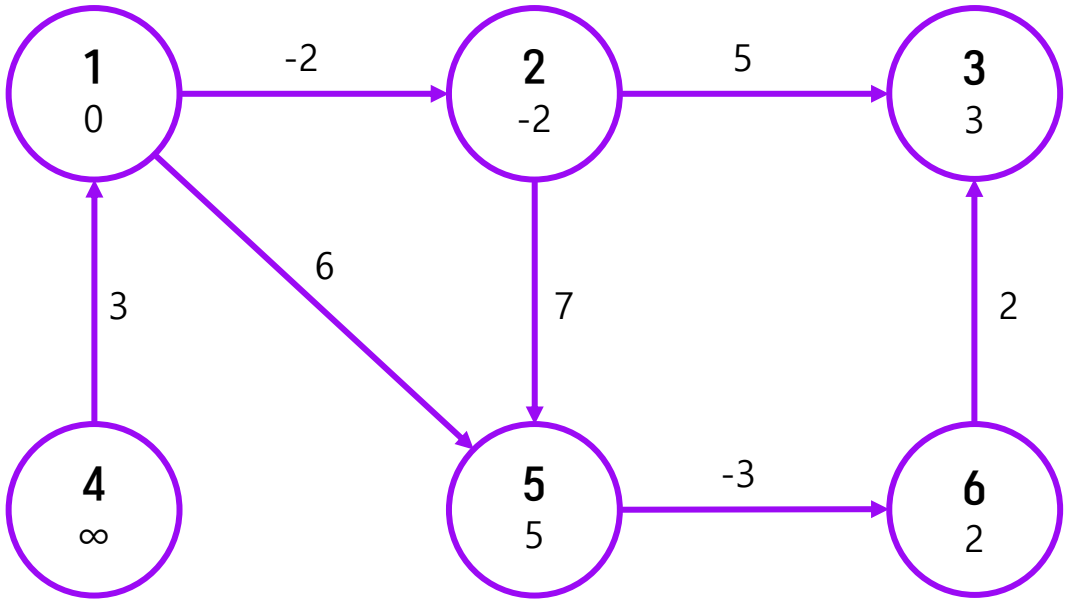
Bellman-Ford Algorithm



2단계 (반복 3회차)
최단 거리가 갱신된 정점에서 출발하는
간선의 도착 정점의 최단 거리를 갱신한다.

정점 번호	1	2	3	4	5	6
최단거리	0	-2	3	INF	5	2

Bellman-Ford Algorithm



정점 번호	1	2	3	4	5	6
최단거리	0	-2	3	INF	5	2

2단계 (반복 4, 5회차)
최단 거리가 갱신된 정점에서 출발하는
간선의 도착 정점의 최단 거리를 갱신한다.

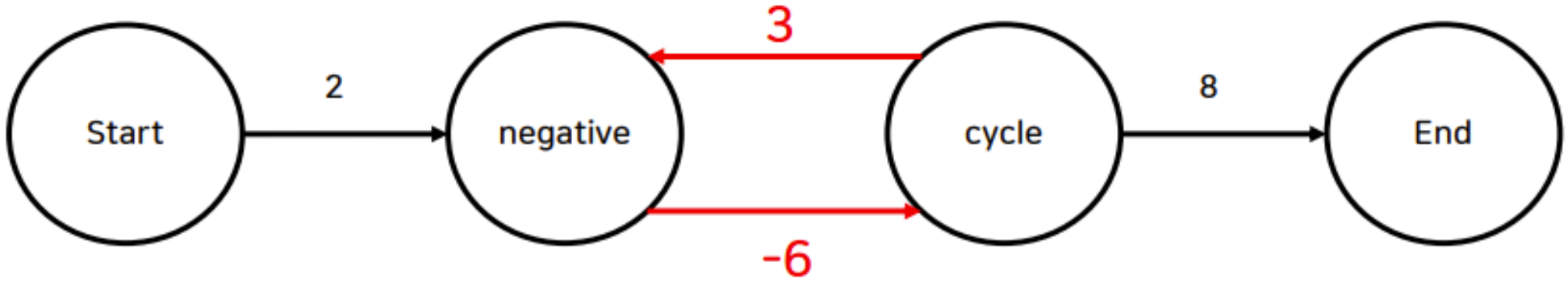
4, 5회차에서는 더 이상
최단 거리 갱신이 일어나지 않아요!

각 반복 회차에서 간선을 체크하는 순서는
갱신 과정에 차이를 줄 수 있으나
갱신 결과에는 영향을 미치지 않습니다.

#CH.3

Bellman-Ford with 음수 사이클

Bellman-Ford with 음수 사이클



Negative와 Cycle을 계속 왔다 갔다 하면?
최단 경로가 음의 무한대로 발산하게 됩니다!

이걸 어떻게 찾을까요?

Bellman-Ford Algorithm

왜 V-1번 반복할까요?

V개의 정점이 있을 때, 한 정점에서 다른 정점으로 가는
경로 상에는 최대 V-1개의 간선이 있기 때문입니다.

ex) 1번 정점에서 5번 정점까지 정점을 최대한 거쳐가는 최단경로는
1->2->3->4->5 이렇게 4개의 간선을 거쳐가는 것이 최대이다.
1->2->5 는 최단경로가 될 수는 있지만
1->2->3->2->4->5 는 될 수가 없다.

모든 간선에 대해 V-1번만 반복하면 음수 사이클이 없는 이상
더 이상의 최단 거리 갱신이 일어나지 않습니다.

Bellman-Ford with 음수 사이클

모든 간선에 대해 V-1번만 반복하면 음수 사이클이 없는 이상
더 이상의 최단 거리 갱신이 일어나지 않습니다.

음수 사이클이 있다면 몇 번을 돌리든 계속 갱신이 일어나겠죠?
해당 정점의 최단 거리가 음의 무한대로 발산하기 때문에..

V번째 반복에서도 최단 거리 갱신이 일어난다면
음수 사이클이 존재한다고 판단할 수 있어요!!

```cpp
const int INF = 2'000'000'009; // 굉장히 큰 수
const int V_MAX = 1'000'006; // 정점 최대 개수
const int E_MAX = 20'000'007; // 간선 최대 개수

typedef tuple<int,int,int> Edge;
Edge edges[E_MAX];
int V, E, dist[V_MAX];

bool Bellman_Ford(int start)
{
    int a, b, w;
    fill(dist, dist + V_MAX, INF);
    dist[start] = 0;

    // 최단 거리 갱신
    for (int v = 1; v <= V - 1; v++) {
        for (Edge e : edges) {
            tie(a, b, w) = e;
            dist[b] = min(dist[b], dist[a] + w);
        }
    }

    // 음수 사이클 존재 파악
    for (Edge e : edges) {
        tie(a, b, w) = e;
        if (dist[b] > dist[a] + w) return true;
    }

    return false;
}
```

코드로 확인해봅시다!

dist[i]: 시작 정점에서 i번 정점까지의 최단 거리

V: 정점의 개수

E: 간선의 개수

edges: 간선들을 모아 놓은 배열

Edge: tuple<출발 정점, 도착 정점, 간선 길이>

Bellan_Ford(): 음수 사이클이 있으면 true

참고) SPFA (Shortest Path Faster Algorithm)

Bellman-Ford의 비효율적인 부분을 개선한 알고리즘으로써
시간 복잡도는 똑같이 O(VE)이지만 실제로는 O(V+E), O(E)의 성능을 가짐.
(최단 거리가 갱신된 정점에서 출발하는 간선에 대해서만 체크함)

BFS와 유사하게 최단 거리가 갱신된 정점에서 출발하는 간선을 queue에 넣고
while문과 for문을 중첩하여 경로를 탐색함.
(물론 최단 거리 갱신을 위해, 방문했던 정점을 다시 방문할 수 있음)

음수 가중치가 존재하는 최단 경로 문제에서
Bellman-Ford로 시간초과가 난다면
SPFA를 찾아 공부해보는 것이 방법일 수도 있어요!
하지만 그럴 때 우선은 자신의 코드를 먼저 잘 살펴봅시다

최단 경로 총정리

Floyd-Warshall

- 모든 정점에 대하여..
- 모든 정점 간 최단 거리
- 음수 가중치 가능
- 인접 행렬 이용
- 시간 복잡도 $O(V^3)$

Bellman-Ford

- 모든 간선에 대하여..
- 시작 정점 정해져 있음
- 음수 가중치 가능
- 음수 사이클 체크 가능
- 간선 배열 이용
- 시간 복잡도 $O(VE)$

Dijkstra

- 시작 정점 정해져 있음
- 음수 가중치 불가
- 인접 리스트 이용
- 시간 복잡도 $O(E \log V)$

E: 간선의 개수

V: 정점의 개수

참고) $V - 1 \leq E \leq V^2$

풀어볼까유

#11657 타임머신

음수 가중치가 존재한다면?

풀어볼까유

#1865 웜홀

음수 사이클을 찾아보아요

풀어볼까유

#1219

오민식의 고민

어떤 자료구조?
어떤 동작 방법?

풀어볼까유

#3860 할로윈 묘지

도전해 봅시다

다음 시간에 만나요~