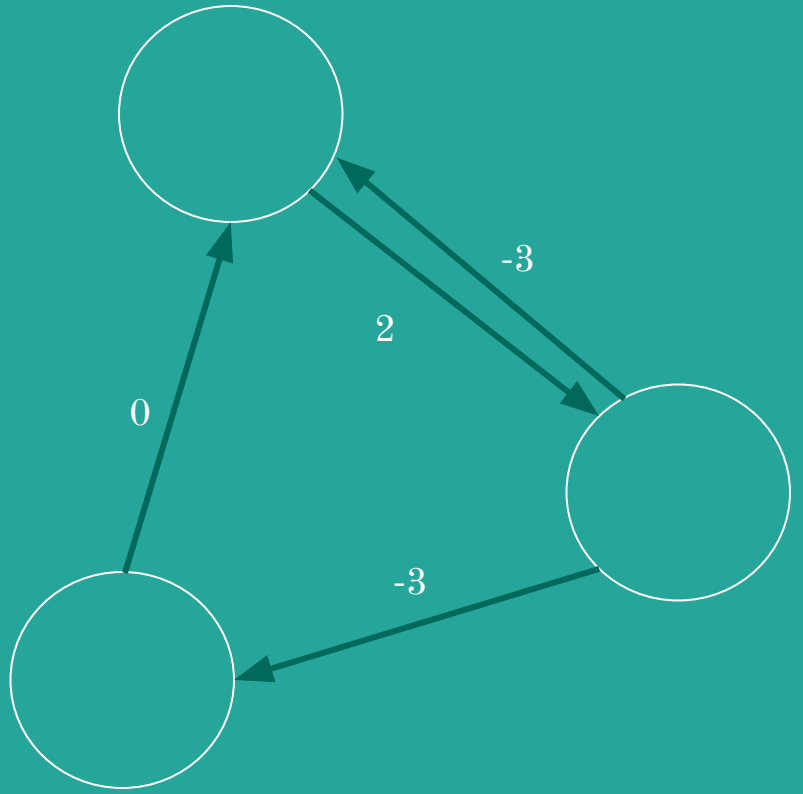
Floyd-Warshall Algorithm

플로이드-워셜 알고리즘

플로이드-워셜 알고리즘은 **음수 사이클이 없는 그래프내의 각 모든 정점에서 각 모든 정점에 까지의 최단거리를 모두 구할 수 있는 알고리즘**이다.

다익스트라 알고리즘과는 다르게 **그래프에 음수 사이클만 존재하지 않으면, 음의 가중치를 갖는 간선이 존재해도 상관이 없다는 것**이다.

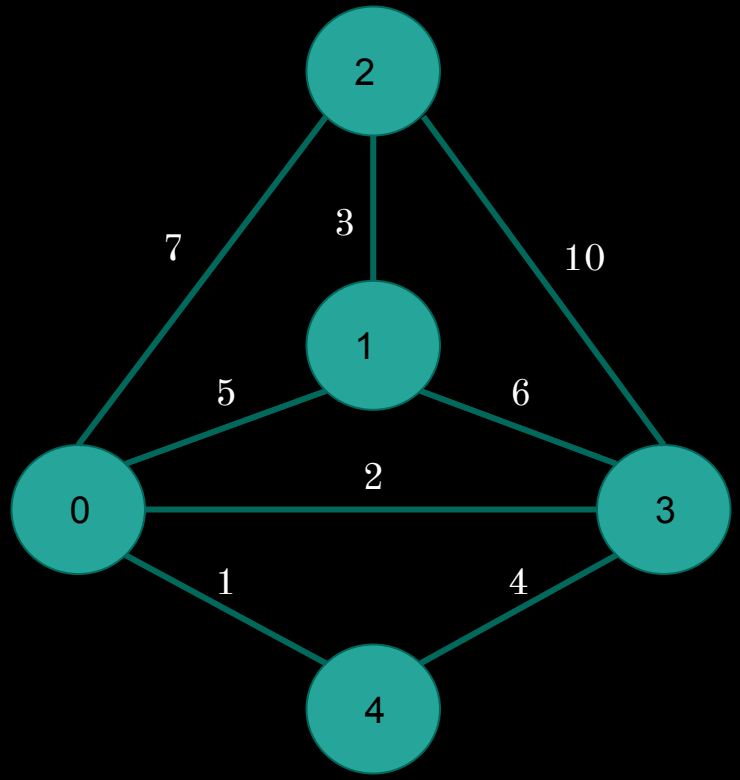
음수 사이클 : 사이클의 모든 경로를 지나 원래 지점으로 돌아 왔을때, 최종적인 비용이 음수가 되는 경우.



본론

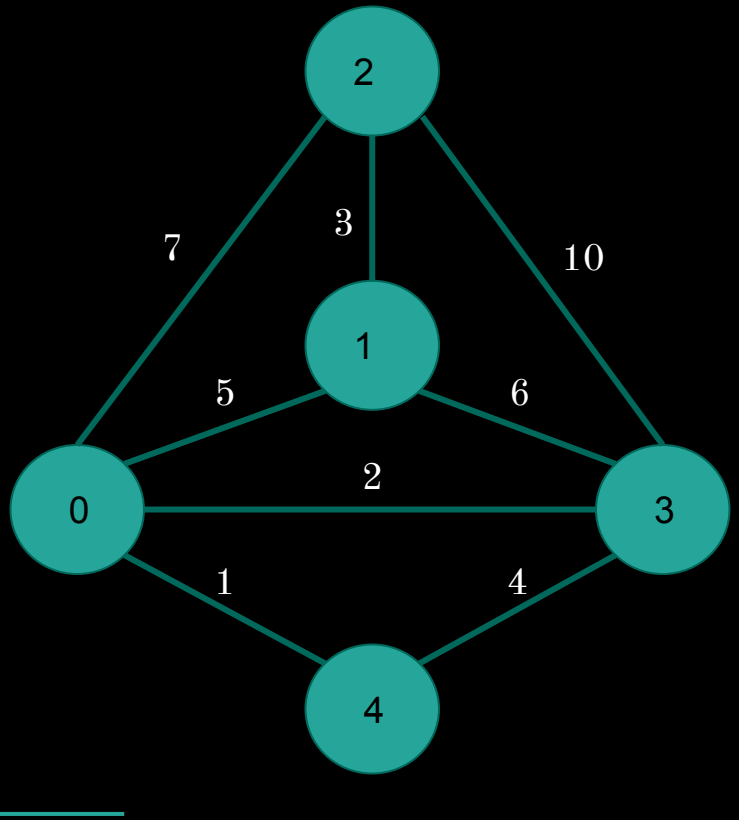
플로이드-워셜 알고리즘은 **다이나믹 프로그래밍 기법을 사용한 알고리즘**이고, **인접 행렬을 이용**하여 각 노드간 최소 비용을 계산한다.

플로이드-워셜 알고리즘은 **모든 노드에서, 모든 노드로 가는 최소 비용을 단계적으로 갱신하면서 진행되는 알고리즘**이다. 여기서 '단계적으로 갱신한다'란, **노드에서 노드로 가는 간선의 개수가 0개에서, N개(총 노드의 개수 만큼 간선을 선택)까지 몇 개의 간선을 거쳐서 해당 노드로 가는지를 '모두' 고려**한다는 이야기이다.



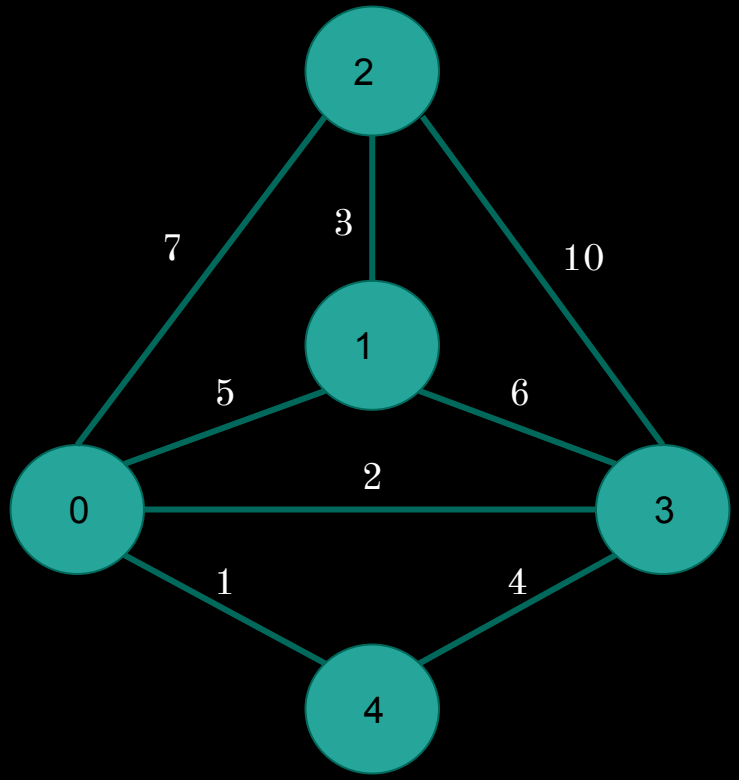
우선, **0개의 간선을 거쳐서 모든 노드에서 모든 노드로 가는 경우**를 생각해보자. **0개의 간선을 거쳐서 모든 노드에서 모든 노드로 가는 방법은 자기 자신에서 자기 자신의 노드로 가는 경우** 밖에 없다. 따라서, **초기 인접 행렬의 값은 자기 자신에서 자기 자신으로 가는 경우(=0)를 제외하고는 다른 노드로 갈 수 있는 방법이 없으므로 모두 매우 큰값으로 초기화**해준다. 그렇다면 인접행렬은 아래와 같은 값을 가지게 될 것이다.

0 INF INF INF INF
INF 0 INF INF INF
INF INF 0 INF INF
INF INF INF 0 INF
INF INF INF INF 0



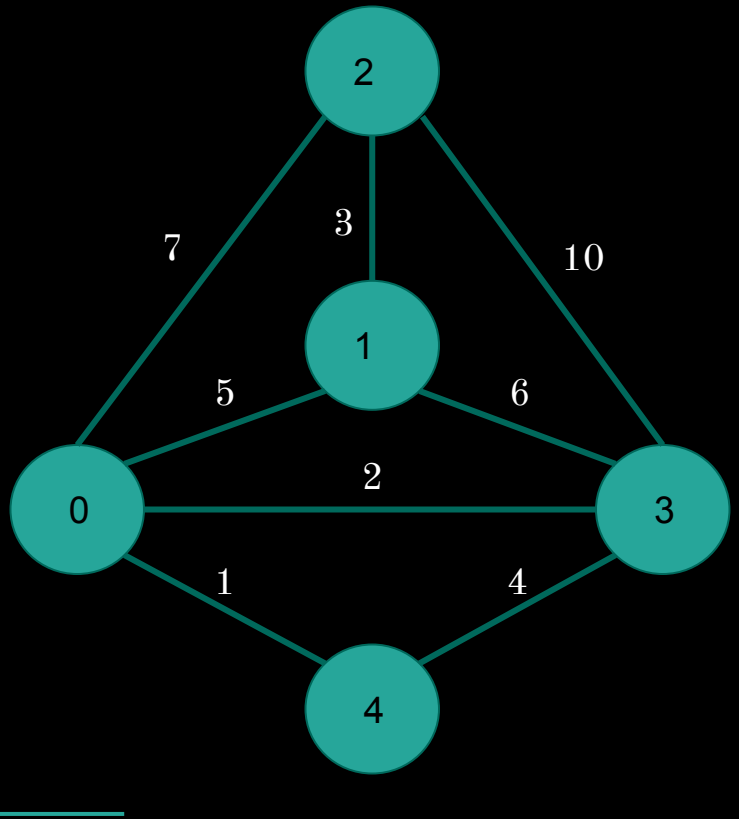
그 다음은 **1개의 간선을 거쳐서 모든 노드에서 노드로 가는 경우**를 생각해보자. **1개의 간선을 거쳐서 모든 노드로 가는 방법은, 말 그대로 주어진 그래프의 간선의 연결상태를 고려하라는 것**이다. 물론 **노드에서 노드로 가는 간선이 여러가지인 경우**가 존재한다면, **그 간선 중 최소 비용인 것을 선택**하면 된다. **보통 여기까지의 단계가 플로이드-워셜 알고리즘에서 초기화 단계에 속한다.** 즉, 위 그림 그대로 간선의 연결 상태를 고려하여 인접행렬을 초기화하면 다음과 같을 것이다.

```
0   5   7   2   1
5   0   3   6 INF
7   3   0  10 INF
2   6  10   0   4
1 INF INF   4   0
```



이렇게 간선 0개를 거친 경우, 1개를 거친 경우, …, N개를 거친 경우의 최소 거리 비용을 비교하면서 인접행렬(각 노드에서 노드까지의 거리 비용을 나타냄)을 갱신하는 것이 플로이드 알고리즘이다. 간선의 개수를 차례로 고려하면서 최소값을 갱신해 나가는 것이다. 간선 2개를 거쳐서 가는 경우를 고려하고 난 뒤의 인접행렬은? 아래와 같을 것이다.

0 5 7 2 1
5 0 3 6 6
7 3 0 9 8
2 6 9 0 3
1 6 8 3 0



구현

```java
// 노드 개수
N = Integer.parseInt(br.readLine());
// 간선 개수
M = Integer.parseInt(br.readLine());


// 플로이드 초기 거리 테이블 초기화
dist = new int[N][N];
for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
        // 자기 자신으로 가는 경우를 제외하고는 매우 큰 값(N개의 노드를 모두 거쳐서 가더라도 더 큰 값)
        if (i != j) dist[i][j] = INF;
for (int i = 0; i < M; ++i) {
    st = new StringTokenizer(br.readLine());
    int a = Integer.parseInt(st.nextToken());
    int b = Integer.parseInt(st.nextToken());
    int cost = Integer.parseInt(st.nextToken());
    // 가는 경로가 하나가 아닐 수 있으므로 그 중 최소 비용을 저장
    dist[a][b] = Math.min(dist[a][b], cost);
    dist[b][a] = Math.min(dist[b][a], cost);
}
```

```java
// 플로이드 워셜 알고리즘
// 노드를 1개부터 N개까지 거쳐가는 경우를 모두 고려한다.
// 임의의 노드 s에서 e까지 가는 데 걸리는 최단거리를 구하기 위해서
// s와 e 사이의 노드인 m에 대해 s에서 m까지 가는 데 걸리는 최단거리와
// m에서 e까지 가는 데 걸리는 최단거리를 이용
for (int m = 0; m < N; m++)
    // 노드 s에서 e로 가는 경우.
    for (int s = 0; s < N; s++)
        for (int e = 0; e < N; e++)
            // m번째 노드를 거쳐가는 비용이 기존 비용보다 더 작은 경우 갱신
            // 또는 연결이 안되어있던 경우(INF) 연결 비용 갱신
            dist[s][e] = Math.min(dist[s][e], dist[s][m] + dist[m][e]);
```

과제

- 1389. 케빈 베이컨의 6단계 법칙(가중치 없는 플로이드)
- 15723. n단 논법 (플로이드로 풀어라)
- 11404. 플로이드 (근본 그 자체)