알고리즘 특강
그래프 알고리즘

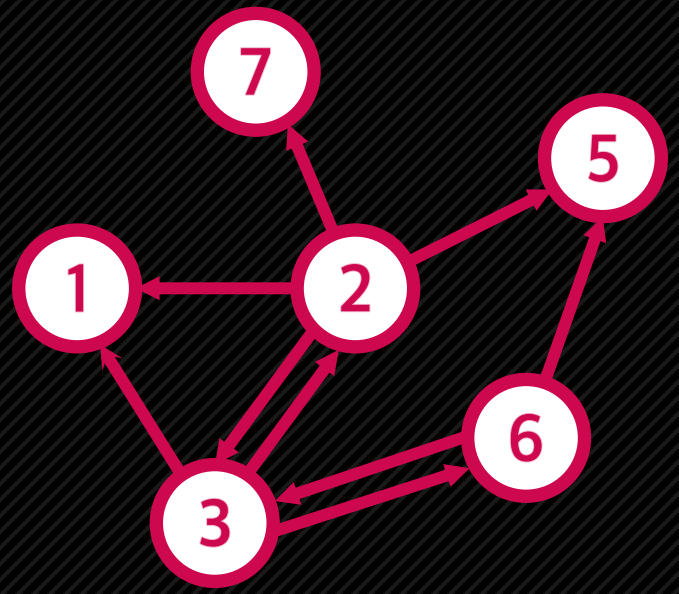
다양한 그래프 알고리즘들에 대해 알아봅시다.
최단거리 및 최소 신장 트리 알고리즘과, 활용법에 대해 배웁니다.

잠깐!

- 이번 단원부터는 코딩테스트에 잘 나오진 않지만, **가끔 출제되는 유형/어렵게 출제되는 유형을 넣었습니다.**
- 시작하시기 전에, **앞부분을 꼼꼼하게 공부하고** 오는걸 권장합니다!

사실 이제 앞 단원 위주만 마스터 하시면 코테 합격은 다 합니다...

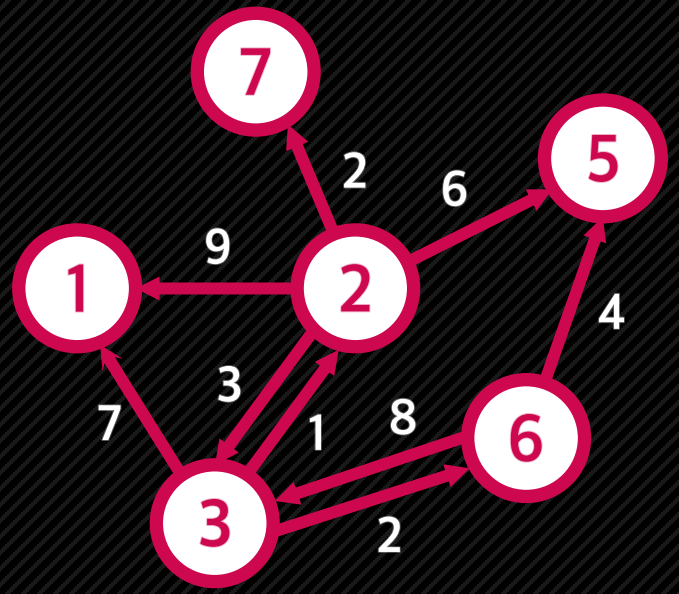
그래프가 뭐였지?



Graph

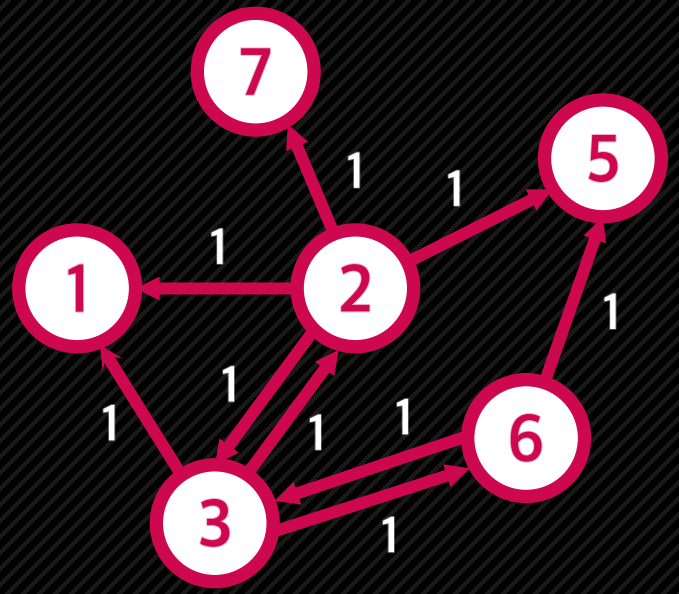
- 노드와 그 노드를 잇는 간선을 하나로 모아 놓은 자료구조.

가중치



- 각 정점과 정점 사이의 거리라고 생각하면 된다!

그동안의 그래프

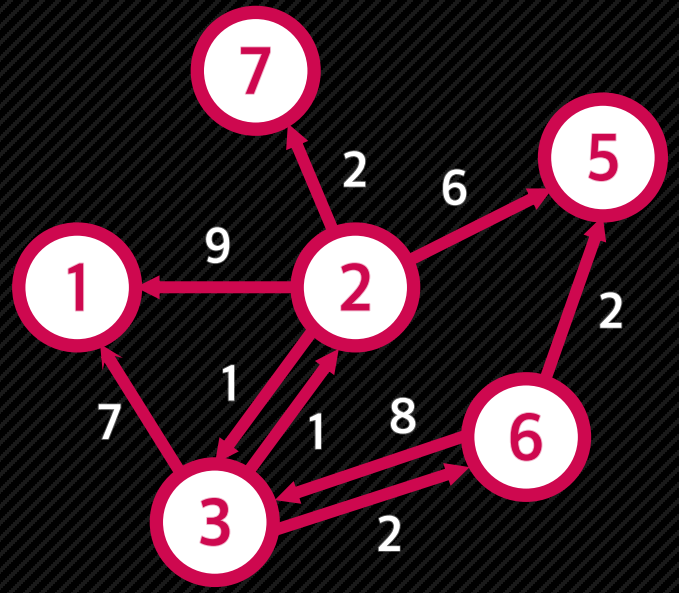


- 지금까지의 그래프는 가중치가 동일했다고 가정할 수 있음.

최단거리?

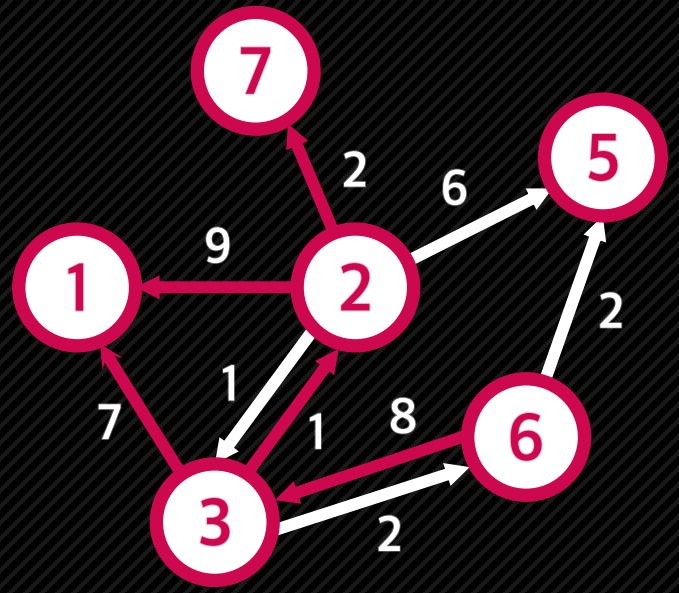
- N번 탐색 후 큐에 들어있는 내용을 모두 꺼내서 한 번 씩 탐색하면,
  큐에는 N + 1 번째로 방문한 값들이 들어있음이 보장됨!

가중치가 다르면?



2에서 5로 가는 최단거리는?

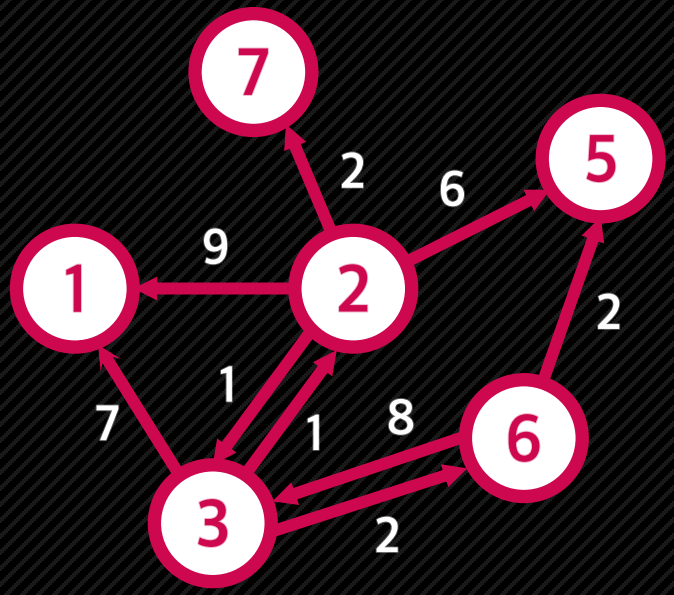
가중치가 다르면?



2에서 5로 가는 최단거리는?

가중치가 다르면?

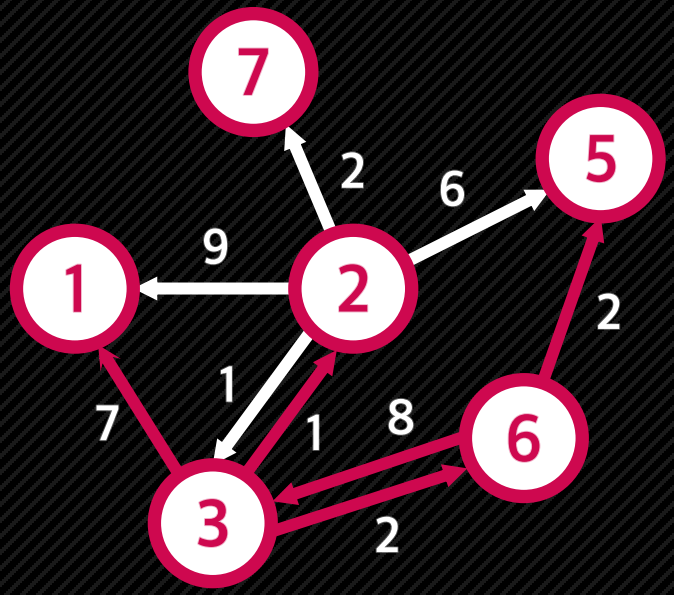
2에서 5로 가는 최단거리는?



	1	3	5	6	7
visited	INF	INF	INF	INF	INF

가중치가 다르면?

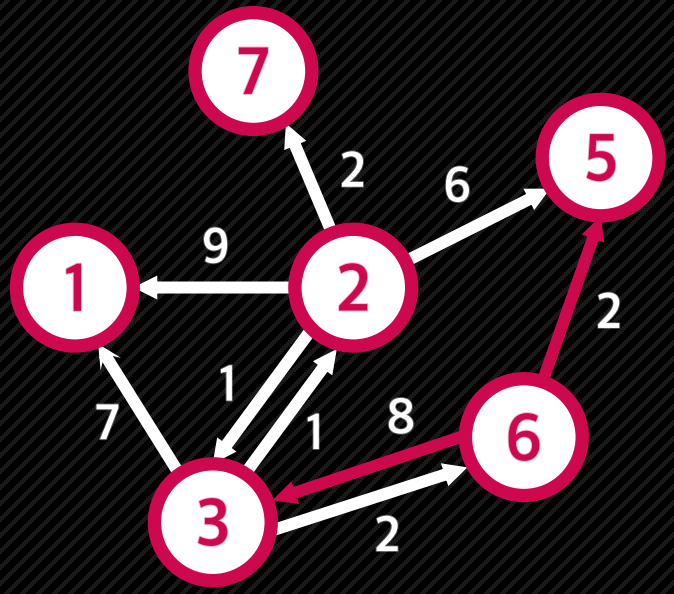
2에서 5로 가는 최단거리는?



	1	3	5	6	7
visited	9	1	6	INF	2

가중치가 다르면?

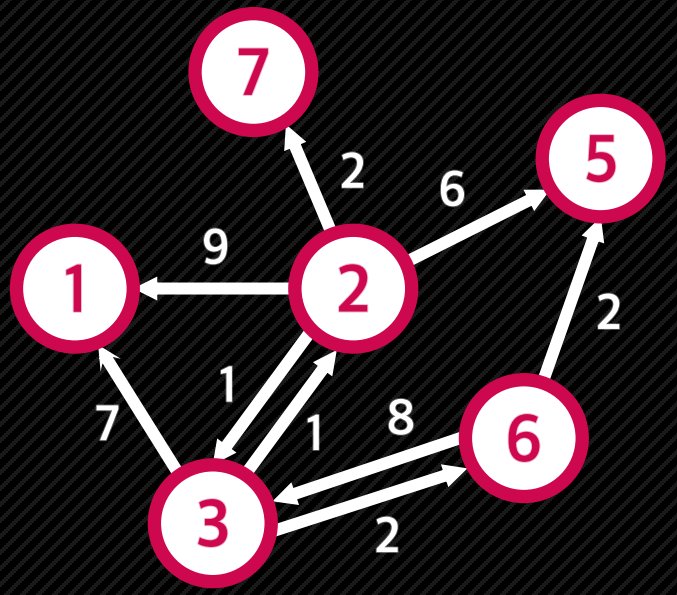
2에서 5로 가는 최단거리는?



	1	3	5	6	7
visited	9	1	6	2	2

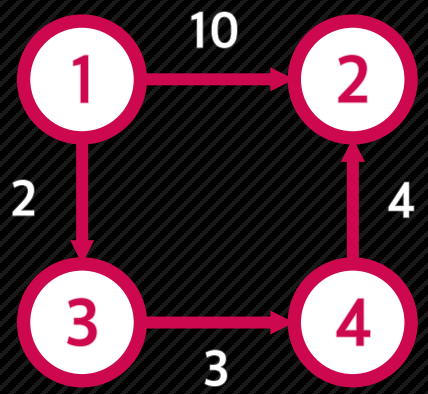
가중치가 다르면?

2에서 5로 가는 최단거리는?

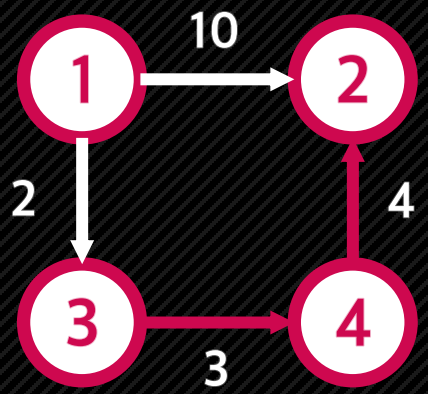


	1	3	5	6	7
visited	9	1	5	2	2

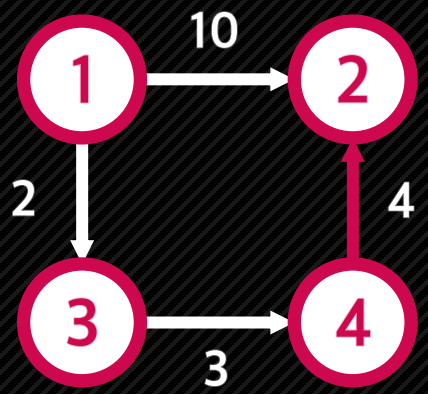
그런데…



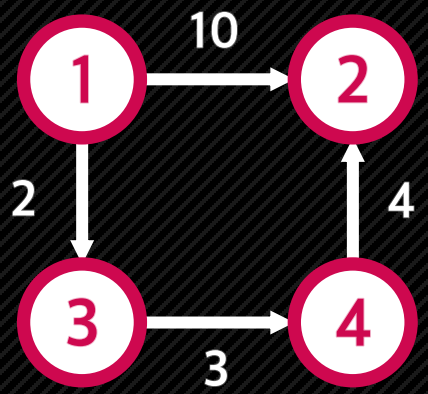
그런데…



그런데…

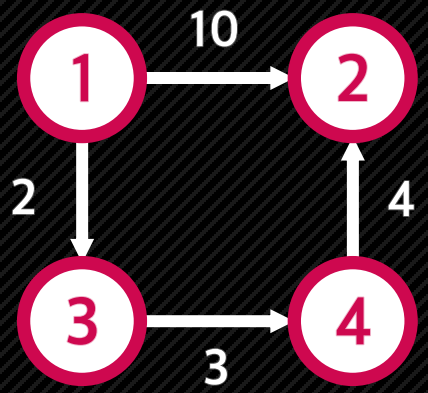


그런데…



또 들어가네?

그런데…



또 들어가네?

- 방문했던 정점을 **또 방문할 수 있음!** → 상황에 따라 시간 초과 가능.

최단거리


특정 정점에서의 거리
모든 정점과 정점의 거리
정점에 가중치 없음
BFS
Floyd-Warshell Algorithm
정점에 가중치 있음
음수 가중치 있음
Bellman-Ford Algorithm
음수 가중치 없음
Dijkstra Algorithm

다익스트라

Dijkstra Algorithm

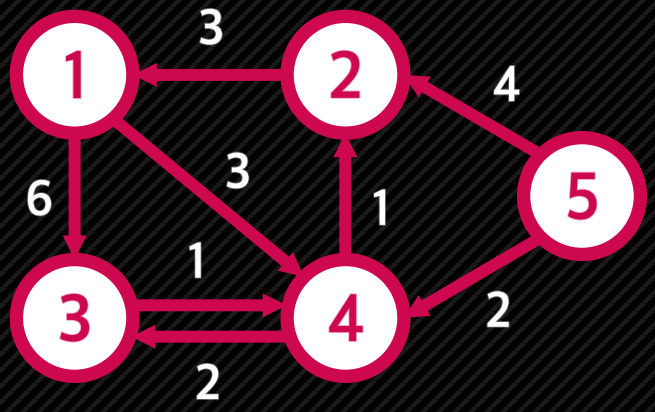
- 한 노드에서 출발하여, 다른 노드로 가는 최단거리를 구해주는 알고리즘.
- 음수 가중치가 없는 (즉, 실질적인 최단거리) 그래프만 적용이 가능함.
- 기본적인 시간 복잡도는 $O(V^2)$이나, 우선순위 큐를 활용하면 $O((V+E)\log V)$가 소요된다.

다익스트라

```
function Dijkstra(Graph, source):
    create vertex set Q

    for each vertex v in Graph:
        set dist[v] <- INF
        set prev[v] <- undef
        add v to Q

    set dist[source] <- 0

    while Q is not empty:
        set u <- vertex in Q with min dist[u]

        remove u from Q

        for each neighbor v of u:
            set alt <- dist[u] + length(u, v)
            if alt < dist[v]:
                set dist[v] <- alt
                set prev[v] <- u

    return dist[], prev[]
```

다익스트라

```
function Dijkstra(Graph, source):
    create vertex set Q

    for each vertex v in Graph:
        set dist[v] <= INF
        set prev[v] <= undef
        add v to Q

    set dist[source] <- 0
```

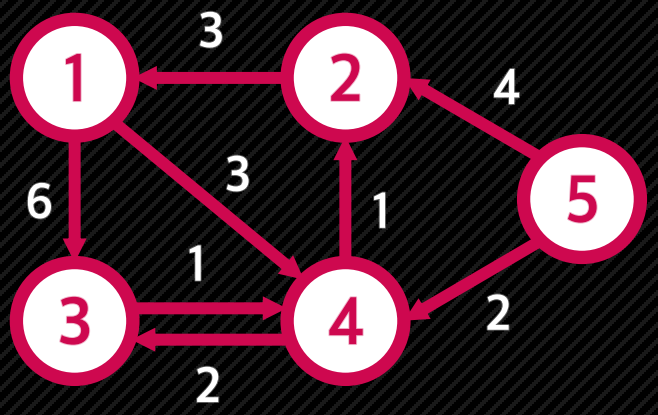


	1	2	3	4	5
dist	INF	INF	INF	INF	0
prev	-	-	-	-	-

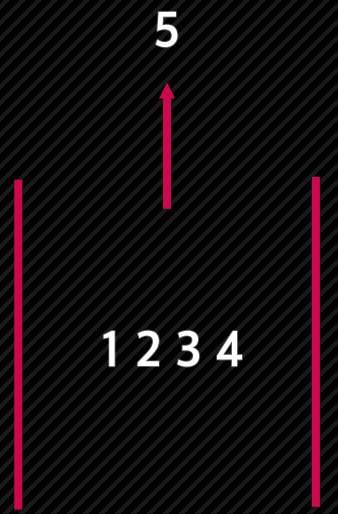
1 2 3 4 5

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

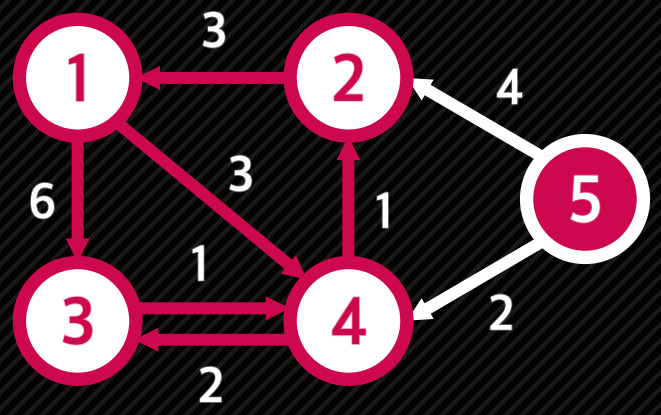


	1	2	3	4	5
dist	INF	INF	INF	INF	0
prev	-	-	-	-	-



다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

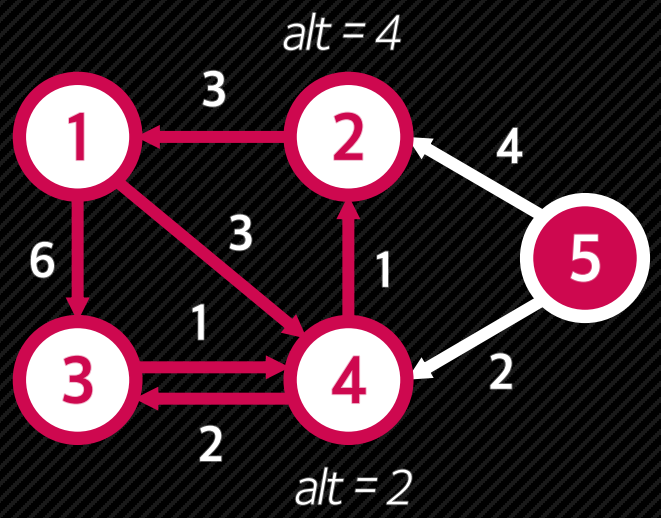


	1	2	3	4	5
dist	INF	INF	INF	INF	0
prev	-	-	-	-	-

1 2 3 4

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

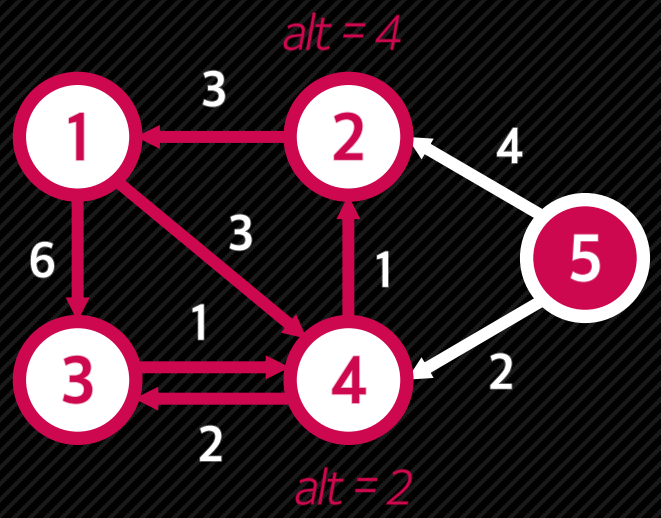


	1	2	3	4	5
dist	INF	INF	INF	INF	0
prev	-	-	-	-	-

1 2 3 4

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

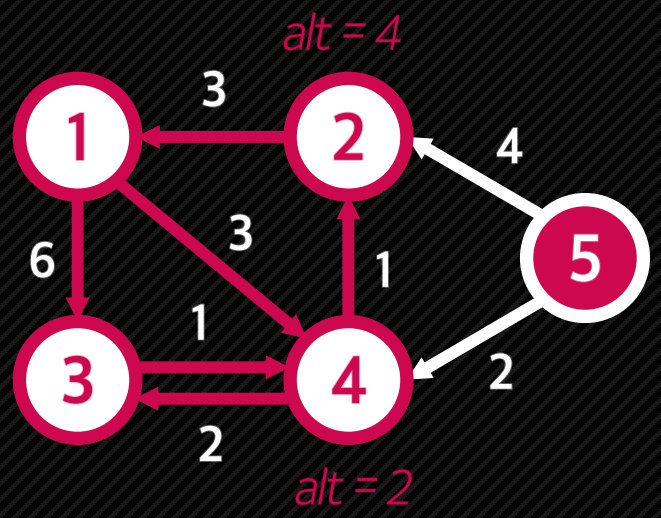


	1	2	3	4	5
dist	INF	INF	INF	INF	0
prev	-	-	-	-	-

1 2 3 4

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

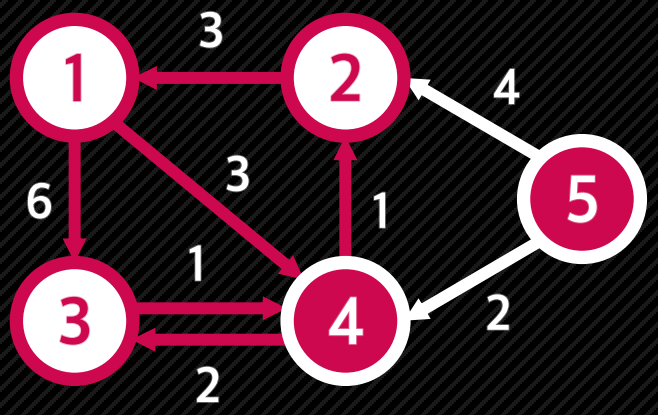


	1	2	3	4	5
dist	INF	4	INF	2	0
prev	-	5	-	5	-

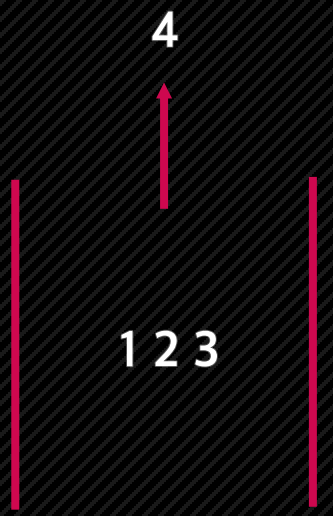
1 2 3 4

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

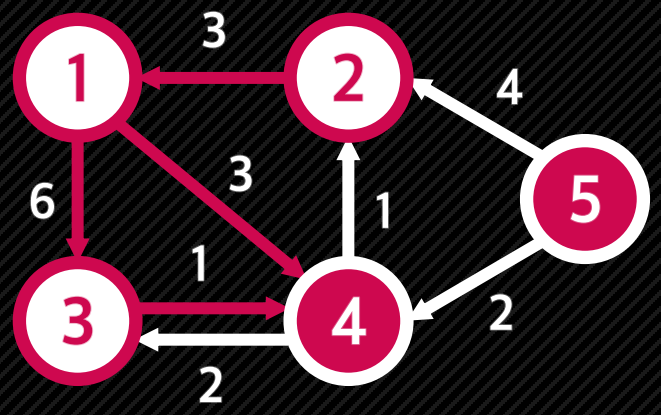


	1	2	3	4	5
dist	INF	4	INF	2	0
prev	-	5	-	5	-



다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

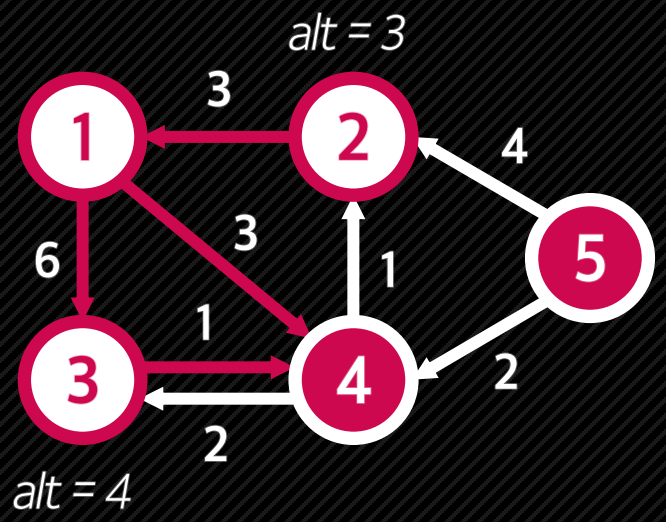


	1	2	3	4	5
dist	INF	4	INF	2	0
prev	-	5	-	5	-

1 2 3

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

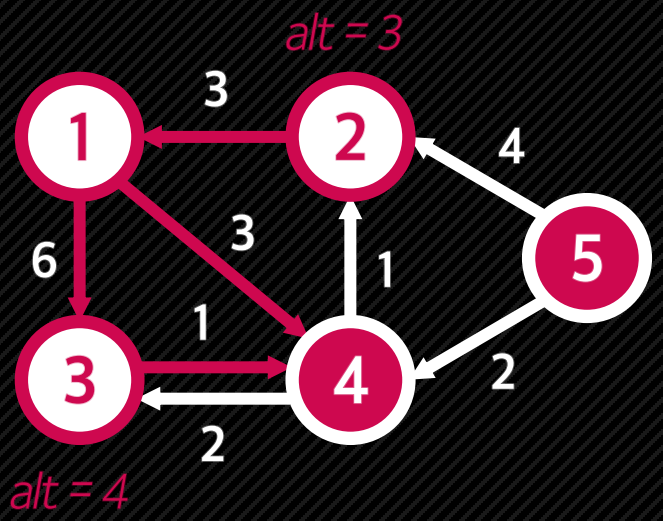


	1	2	3	4	5
dist	INF	4	INF	2	0
prev	-	5	-	5	-

1 2 3

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

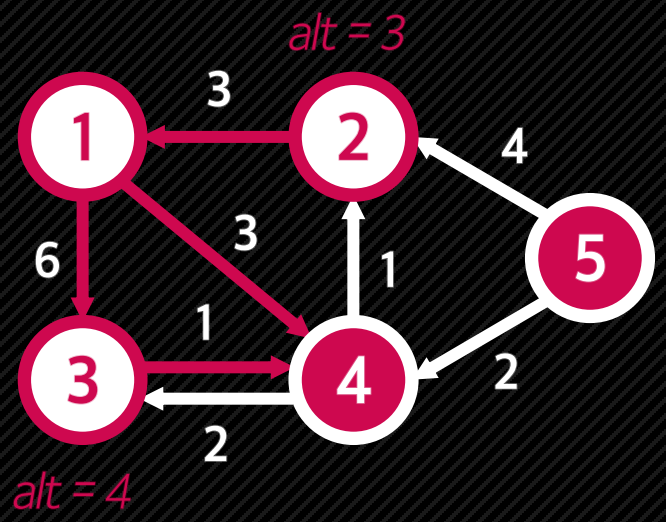


	1	2	3	4	5
dist	INF	4	INF	2	0
prev	-	5	-	5	-

1 2 3

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

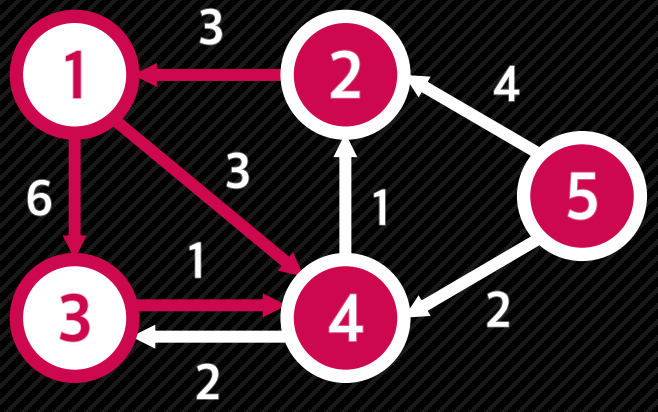


	1	2	3	4	5
dist	INF	3	4	2	0
prev	-	4	4	5	-

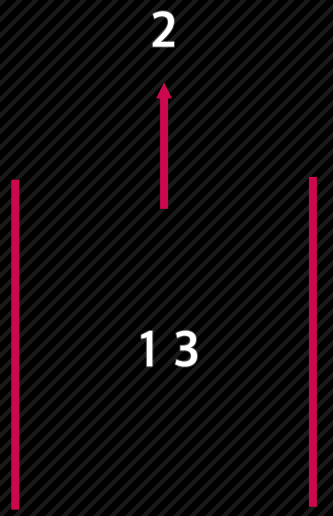
1 2 3

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

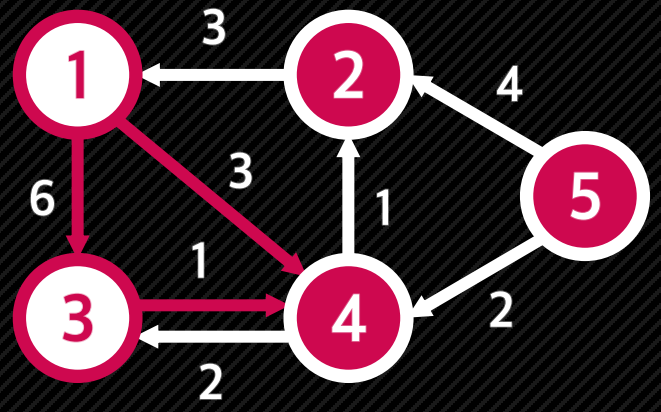


	1	2	3	4	5
dist	INF	3	4	2	0
prev	-	4	4	5	-



다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

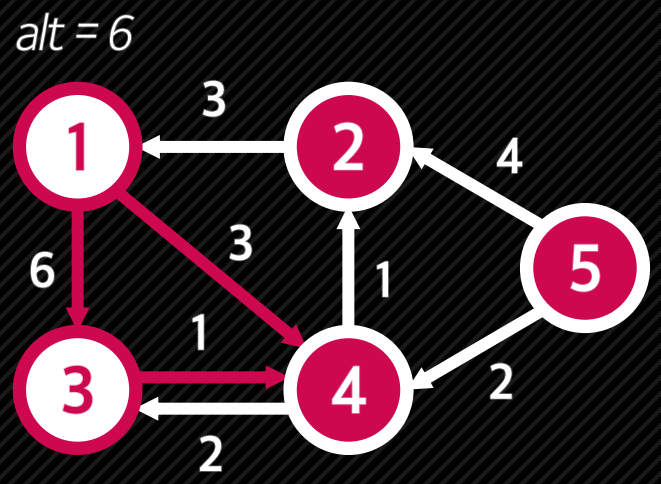


	1	2	3	4	5
dist	INF	3	4	2	0
prev	-	4	4	5	-

1 3

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

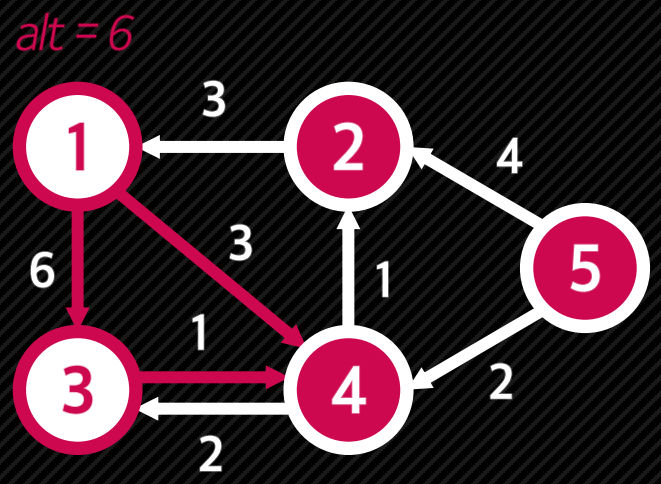


	1	2	3	4	5
dist	INF	3	4	2	0
prev	-	4	4	5	-

1 3

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

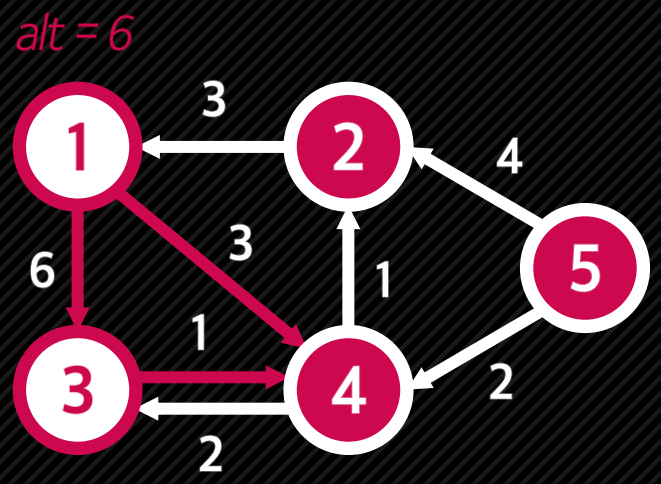


	1	2	3	4	5
dist	INF	3	4	2	0
prev	-	4	4	5	-

1 3

다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

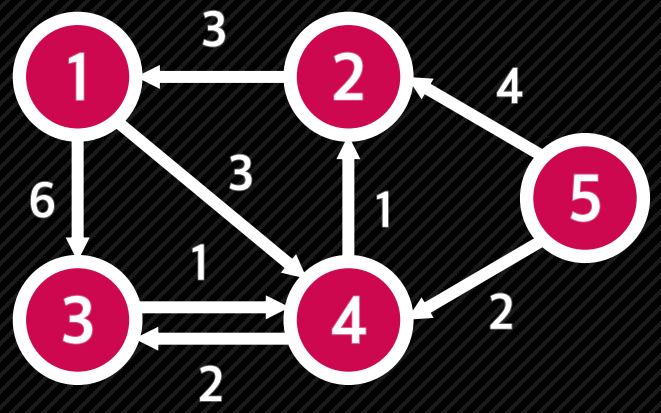


	1	2	3	4	5
dist	6	3	4	2	0
prev	2	4	4	5	-

1 3

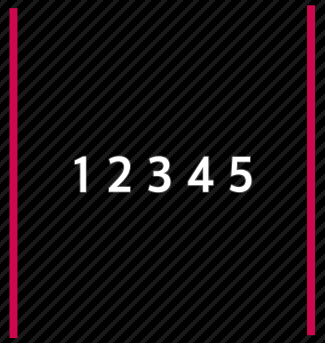
다익스트라

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```



	1	2	3	4	5
dist	6	3	4	2	0
prev	2	4	4	5	-

현실적으로...

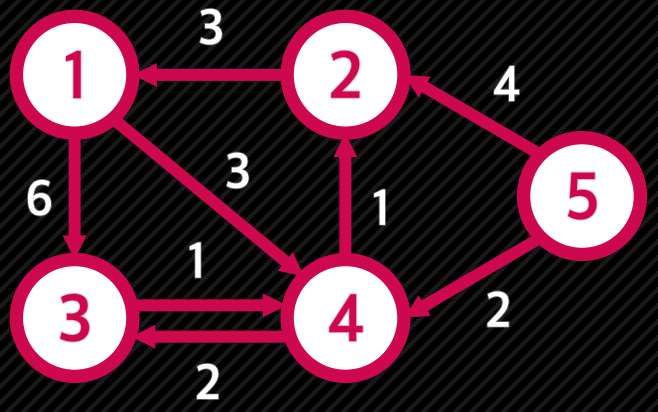



```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

무슨 기준으로 우선순위 큐에서 뺄 수 있을까?

현실적으로...

```
function Dijkstra(Graph, source):
    create vertex set Q

    for each vertex v in Graph:
        set dist[v] <= INF
        set prev[v] <= undef
        add v to Q

    set dist[source] <- 0
```

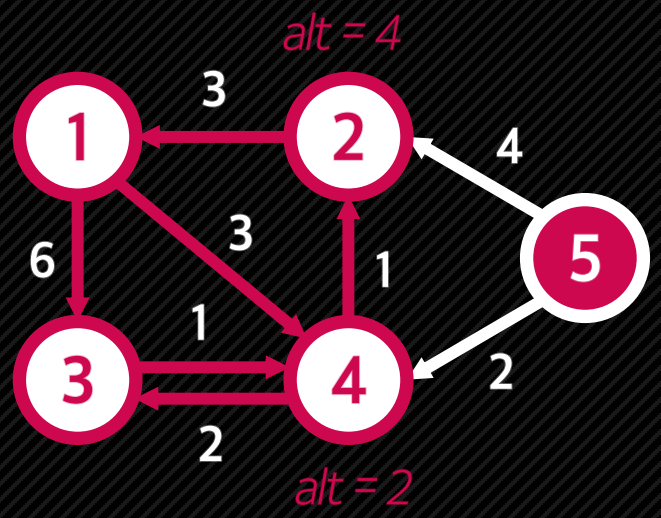


	1	2	3	4	5
dist	INF	INF	INF	INF	0
prev	-	-	-	-	-

(0, 5)
(INF, 1)
(INF, 1)
(INF, 1)
(INF, 1)

현실적으로...

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```

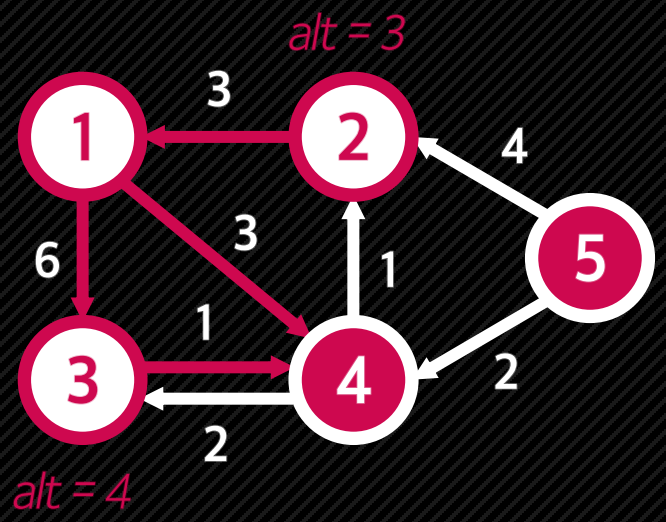


	1	2	3	4	5
dist	INF	4	INF	2	0
prev	-	5	-	5	-

(2, 4)
(4, 2)
(INF, 1)
(INF, 2)
(INF, 3)
(INF, 4)

현실적으로...

```
while Q is not empty:
    set u <- vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:
        set alt <- dist[u] + length(u, v)
        if alt < dist[v]:
            set dist[v] <- alt
            set prev[v] <- u
```



	1	2	3	4	5
dist	INF	3	4	2	0
prev	-	4	4	5	-

(3, 2)
(4, 2)
(4, 3)
(INF, 1)
(INF, 2)
(INF, 3)
(INF, 4)

연습문제

Gold 5 - 최단경로 (#1753)

요약

- 방향그래프가 주어지면 주어진 시작점에서 다른 모든 정점으로의 최단 거리를 구하는 프로그램을 작성하시오.
- 단, 모든 간선의 가중치는 10 이하이다.

제약조건

- 정점의 수의 범위는 1 <= V <= 20,000 이다.
- 간선의 수의 범위는 1 <= E <= 300,000 이다.

실습!

Gold 4 - 특정한 최단경로 (#1504)

요약

- 1번 부터 N번까지 최단거리로 이동하려고 하는데, 주어진 두 정점을 반드시 거쳐가야 한다.
- 해당 조건을 만족하는 최단거리를 출력하는 프로그램을 작성하시오.

제약조건

- 정점의 수의 범위는 1 <= V <= 800 이다.
- 간선의 수의 범위는 1 <= E <= 200,000 이다.

플로이드-워셜

Floyd-Warshell Algorithm

- 가능한 **모든 노드의 최단거리**를 구해주는 알고리즘.
- 시간 복잡도는 $O(V^3)$ 이기 때문에, 일반적으로 코테에선 **1,000개 이하의 간선**인 경우에만 사용 가능함.

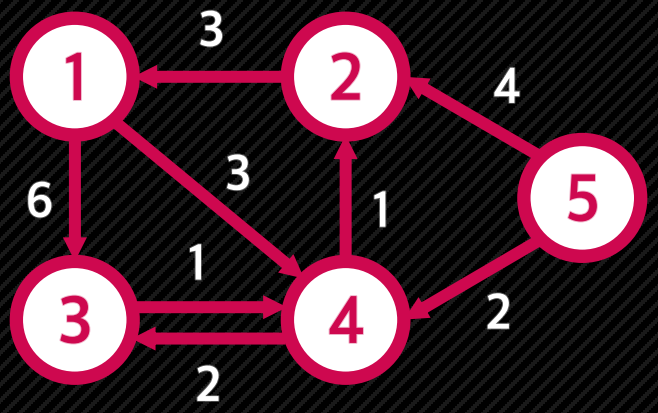
플로이드-워셜

```
function floyd():
    set dist <- |V| * |V| array of minimum distances initialized to INF
    for each edge(u, v)
        set dist[u][v] <- length(u, v)
    for k from 1 to |V|
        for i from 1 to |V|
            for j from 1 to |V|
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    set dist[i][j] <- dist[i][k] + dist[k][j]
                end if
```

- I에서 J로 가는 거리보다 K를 거쳐 가는 것이 더 빠르다면, 갱신하자!

플로이드-워셜

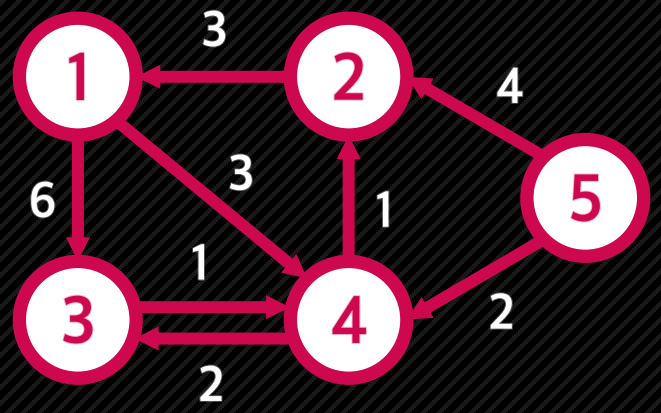
```
function floyd():
    set dist <- |V| * |V| array of minimum distances initialized to INF
    for each edge(u, v)
        set dist[u][v] <- length(u, v)
```



	1	2	3	4	5
1	INF	INF	INF	INF	INF
2	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF
5	INF	INF	INF	INF	INF

플로이드-워셜

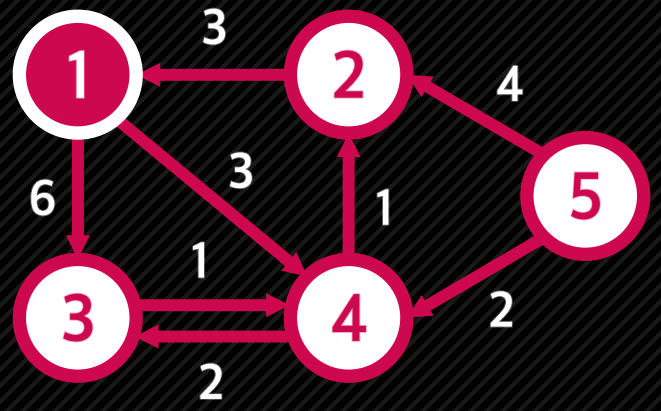
```
function floyd():
    set dist <- |V| * |V| array of minimum distances initialized to INF
    for each edge(u, v)
        set dist[u][v] <- length(u, v)
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	INF	INF	INF
3	INF	INF	0	1	INF
4	INF	1	2	0	INF
5	INF	4	INF	2	0

플로이드-워셜

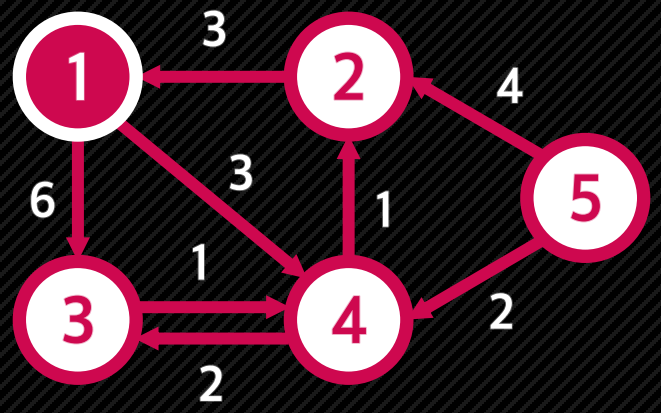
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	INF	INF	INF
3	INF	INF	0	1	INF
4	INF	1	2	0	INF
5	INF	4	INF	2	0

플로이드-워셜

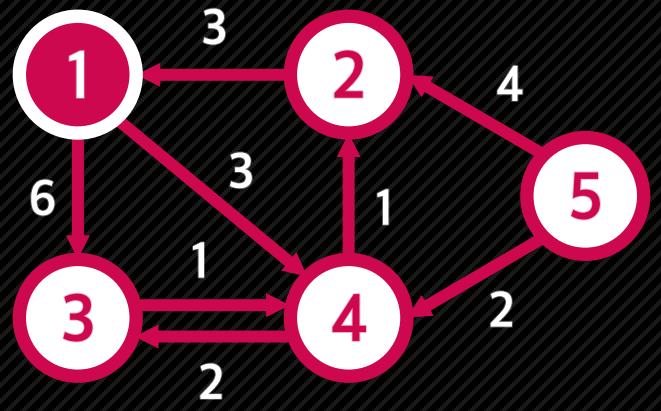
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	INF	INF	INF
3	INF	INF	0	1	INF
4	INF	1	2	0	INF
5	INF	4	INF	2	0

플로이드-워셜

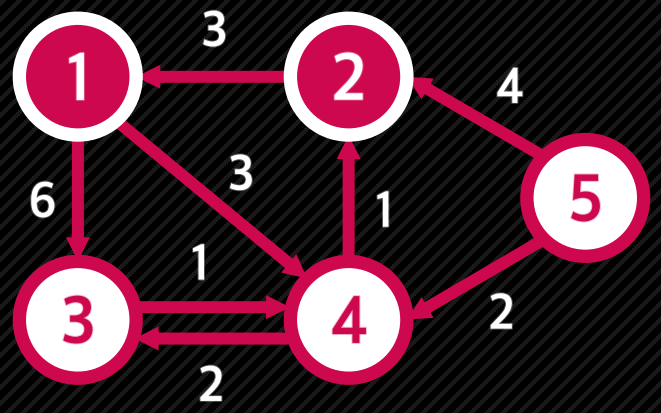
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	9	6	INF
3	INF	INF	0	1	INF
4	INF	1	2	0	INF
5	INF	4	INF	2	0

플로이드-워셜

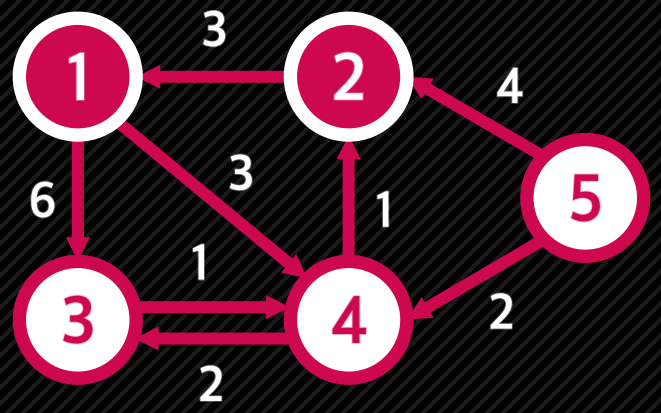
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	9	6	INF
3	INF	INF	0	1	INF
4	INF	1	2	0	INF
5	INF	4	INF	2	0

플로이드-워셜

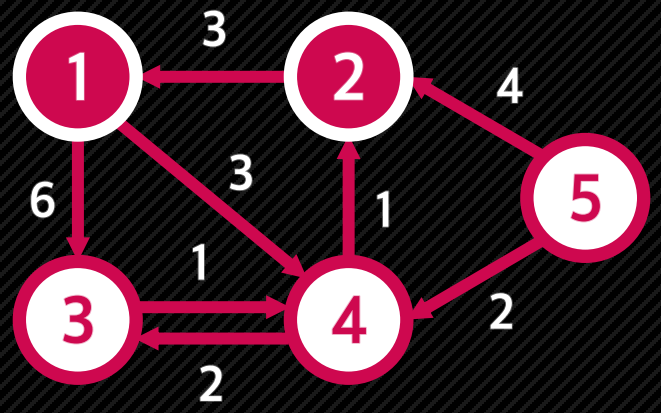
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	9	6	INF
3	INF	INF	0	1	INF
4	INF	1	2	0	INF
5	INF	4	INF	2	0

플로이드-워셜

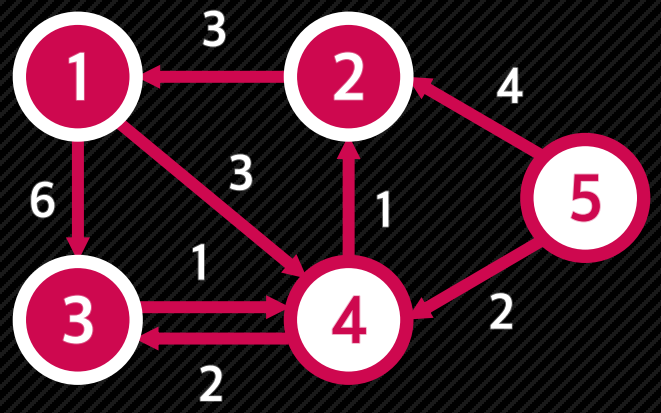
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	9	6	INF
3	INF	INF	0	1	INF
4	4	1	2	0	INF
5	7	4	13	2	0

플로이드-워셜

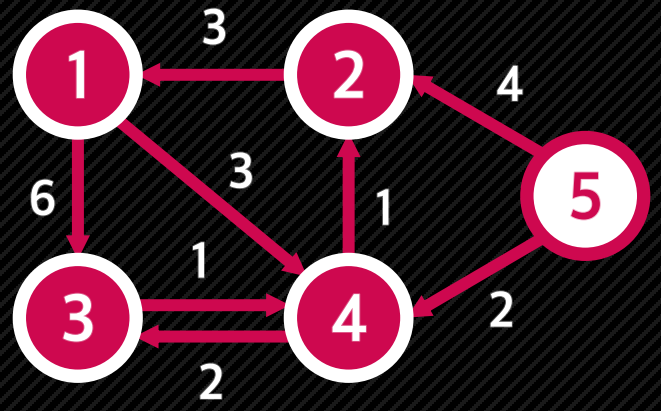
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	9	6	INF
3	INF	INF	0	1	INF
4	4	1	2	0	INF
5	7	4	13	2	0

플로이드-워셜

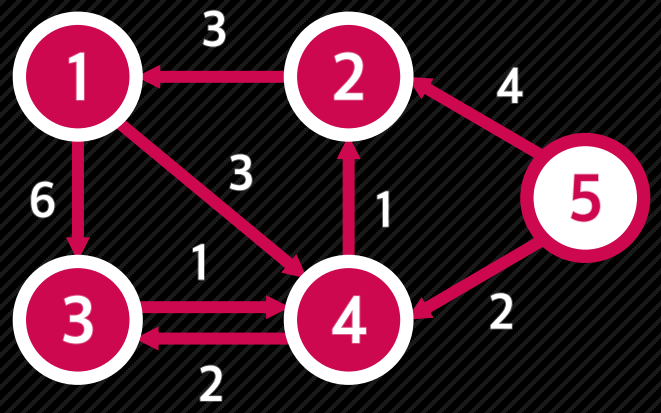
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	9	6	INF
3	INF	INF	0	1	INF
4	4	1	2	0	INF
5	7	4	13	2	0

플로이드-워셜

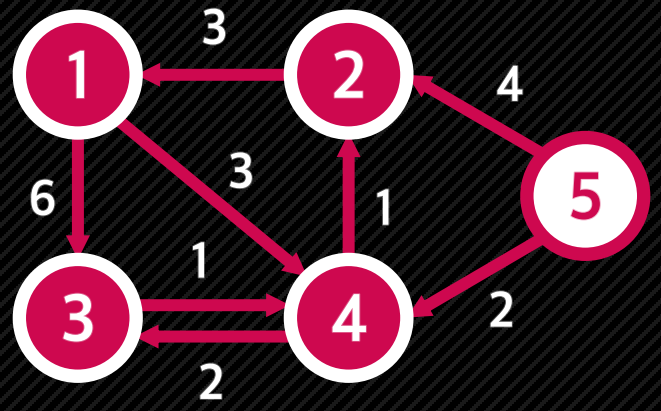
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	INF	6	3	INF
2	3	0	9	6	INF
3	INF	INF	0	1	INF
4	4	1	2	0	INF
5	7	4	13	2	0

플로이드-워셜

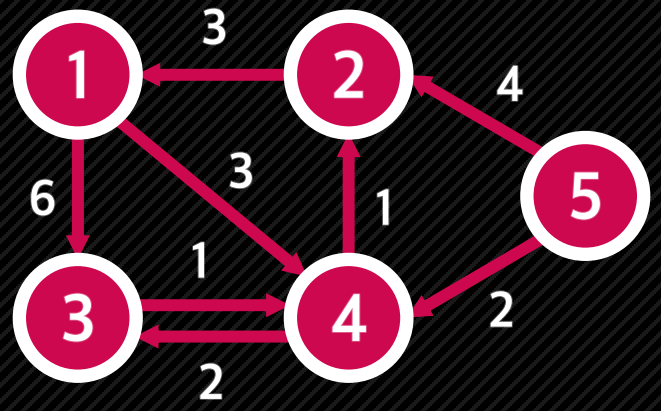
```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	4	5	3	INF
2	3	0	9	6	INF
3	5	2	0	1	INF
4	4	1	2	0	INF
5	6	4	4	2	0

플로이드-워셜

```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```



	1	2	3	4	5
1	0	4	5	3	INF
2	3	0	9	6	INF
3	5	2	0	1	INF
4	4	1	2	0	INF
5	6	4	4	2	0

유의할 점

```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j] then
                set dist[i][j] <- dist[i][k] + dist[k][j]
            end if
```

- K가 반드시 for문의 첫번째로 가야함!!

풀은 문제도 다시보자!

Gold 4 - 특정한 최단경로 (#1504)

요약

- 1번 부터 N번까지 최단거리로 이동하려고 하는데, 주어진 두 정점을 반드시 거쳐가야 한다.
- 해당 조건을 만족하는 최단거리를 출력하는 프로그램을 작성하시오.

제약조건

- 정점의 수의 범위는 1 <= V <= 800 이다.
- 간선의 수의 범위는 1 <= E <= 200,000 이다.

연습, 또 연습!

Gold 4 - 플로이드 (#11404)

요약

- N개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시로 도착하는 M개의 버스가 있다. 각 버스는 요금이 있다.
- 모든 쌍 (A, B)에 대해 A에서 B로 갈 수 있는 비용의 최솟값을 구하는 프로그램을 작성하시오.

제약조건

- 도시의 수의 범위는 2 <= N <= 100 이다.
- 노선의 수의 범위는 1 <= M <= 100,000 이다.

“Any Question?”