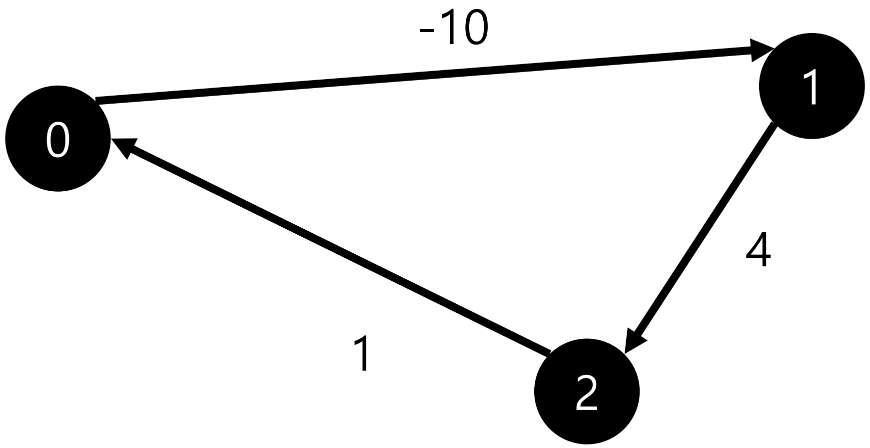
Shortest path

최단 경로(shortest path)
1. 음수 사이클이 없다.
2. 방향 그래프(directed graph)
3. 가중치 그래프(weighted graph)
4. 경로의 길이
: 에지 가중치의 합

음수 사이클(negative cycle)



최단 경로 알고리즘의 종류

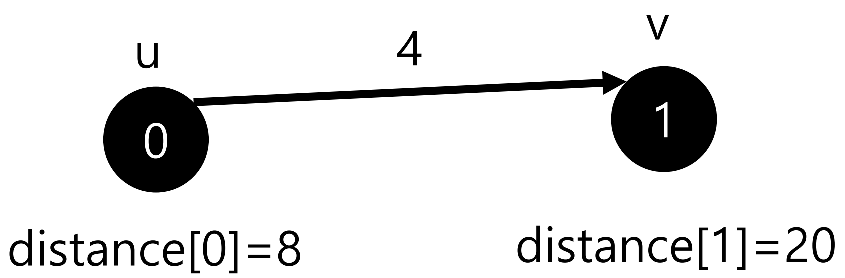
1. 하나의 출발점과 나머지 모든 목적지
   1) Dijkstra 알고리즘(음수 가중치가 없다)
   2) Bellman-Ford 알고리즘(일반적인 경우)
2. 모든 (출발점,목적지) 쌍
   1) Floyd-Warshall 알고리즘

Dijkstra 알고리즘

Dijkstra 알고리즘

1. 탐욕 알고리즘
2. 음수 가중치가 없다.
3. 최단 경로가 발견된 정점의 집합 S
4. 정점 v(v∈ $V - S$)의 distance[v]
   : 출발 정점에서 S에 있는 정점만 거쳐
   v에 도달하는 경로의 길이

Relaxation of an edge

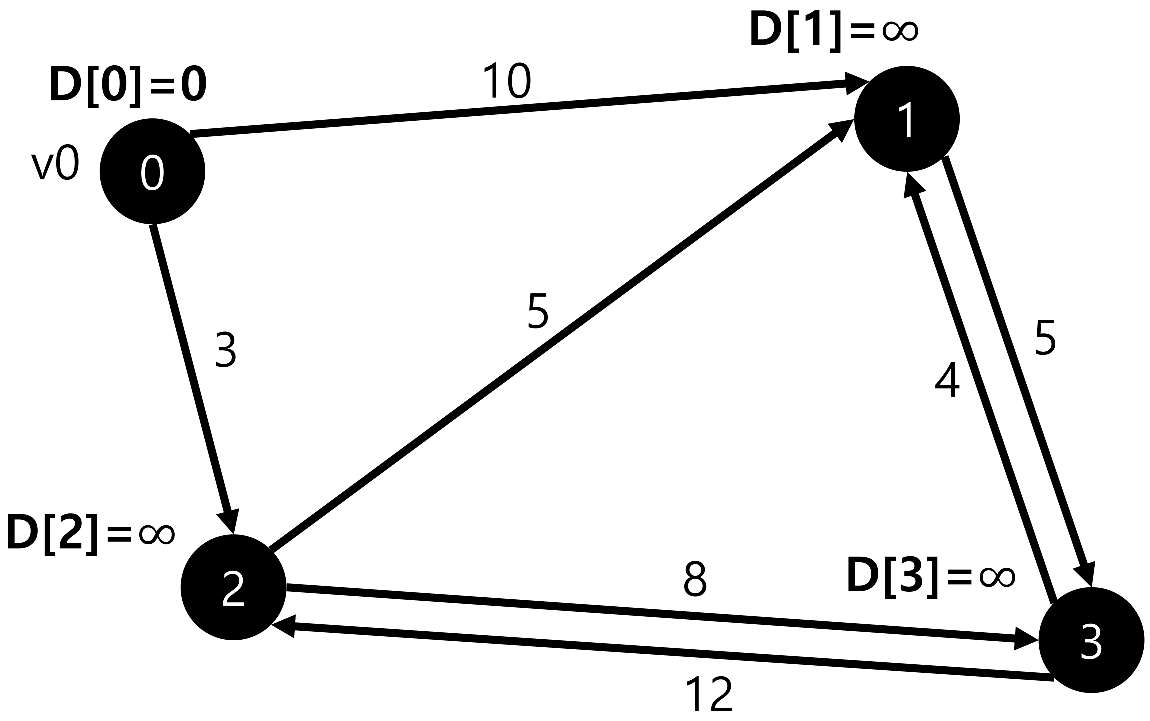


출발 정점에서 정점 v까지 정점 u를 거치지 않고
오는 경로의 길이 distance[1]보다
출발 정점에서 정점 u까지 먼저 온 후 정점 v로 가는
경로의 길이 distance[0] + w(u, v)가 더 작으면
distance[1]를 업데이트한다

```
if distance[v] > distance[u]+w[u, v]
    distance[v]=distance[u]+w[u, v]
```

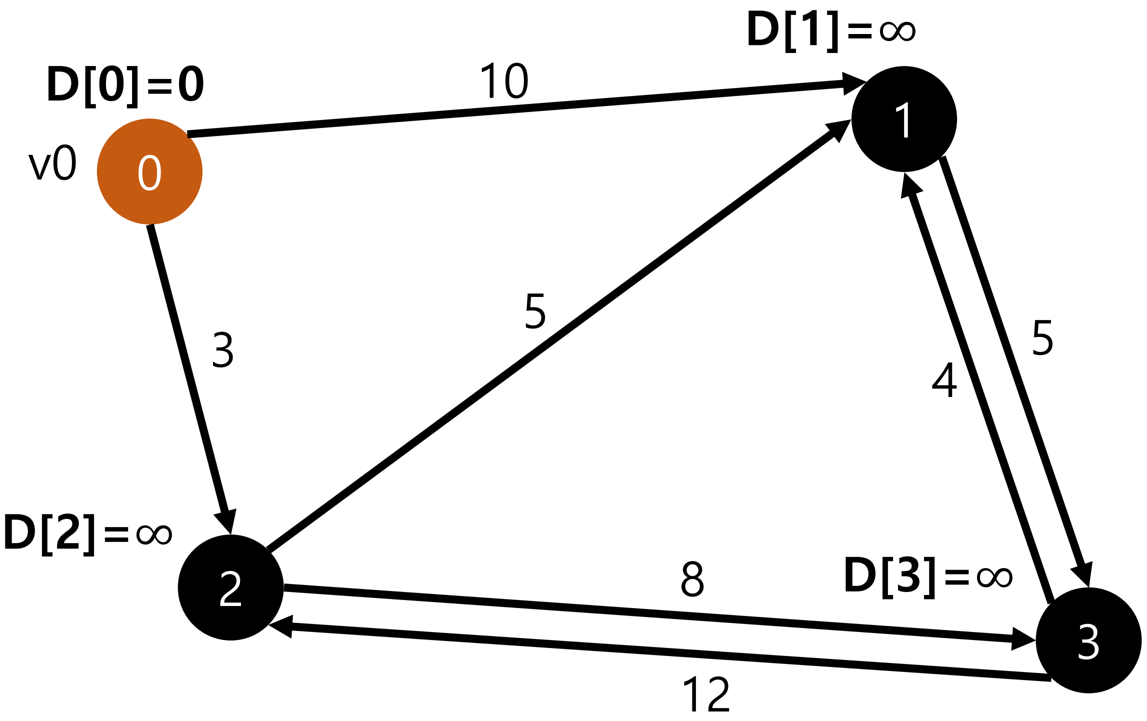
Dijkstra 알고리즘

S={}



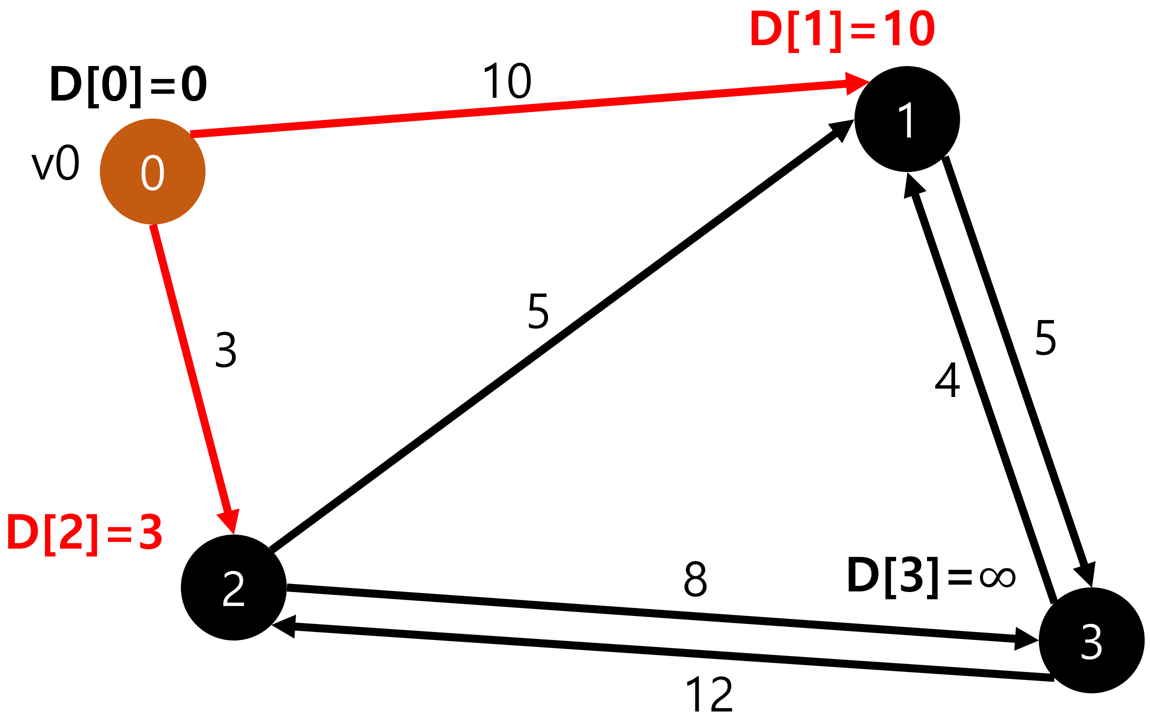
Dijkstra 알고리즘

S={0}



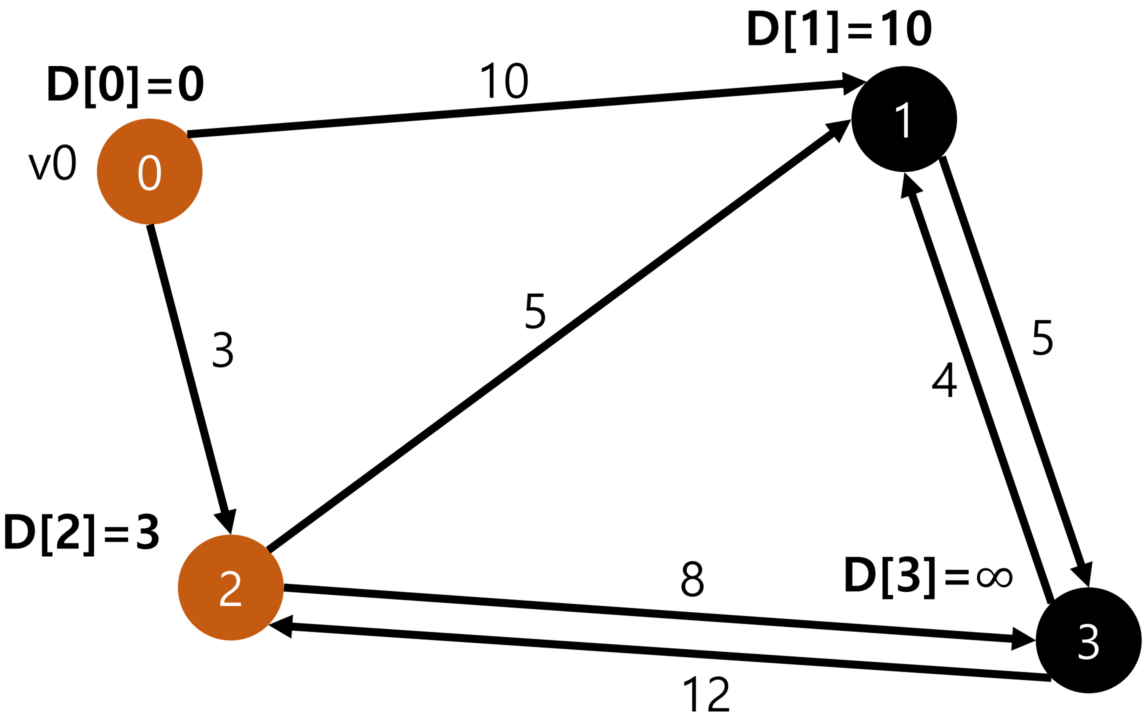
Dijkstra 알고리즘

S={0}



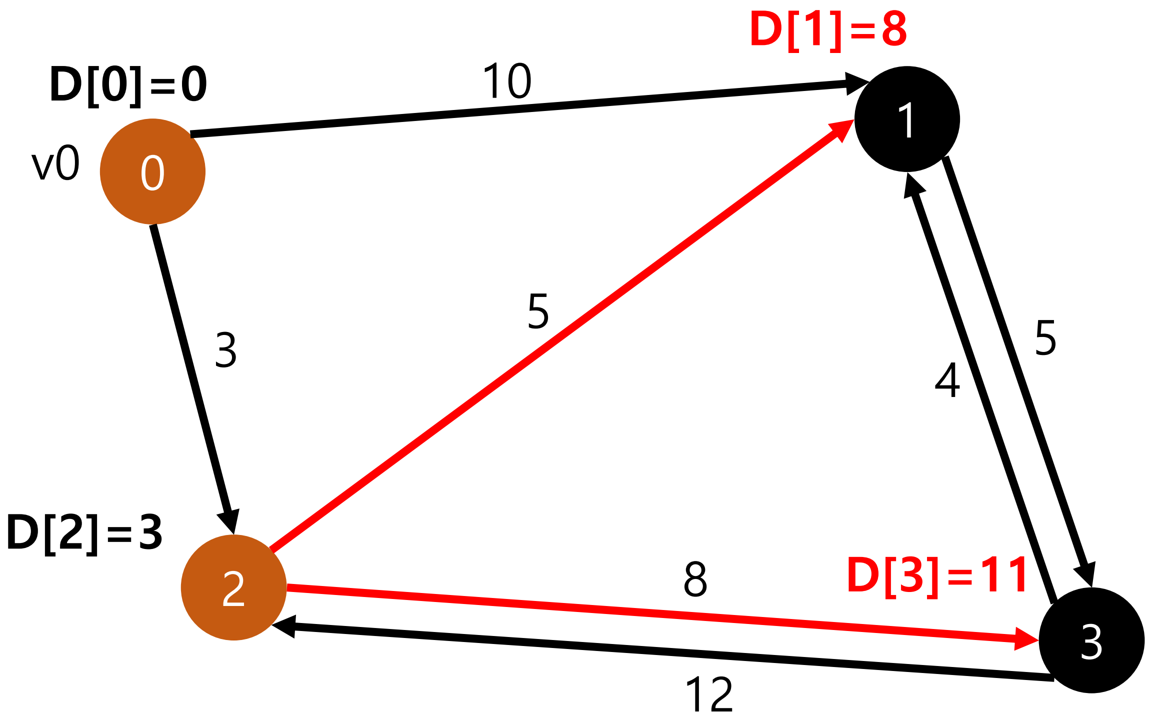
Dijkstra 알고리즘

$S=\{0, 2\}$



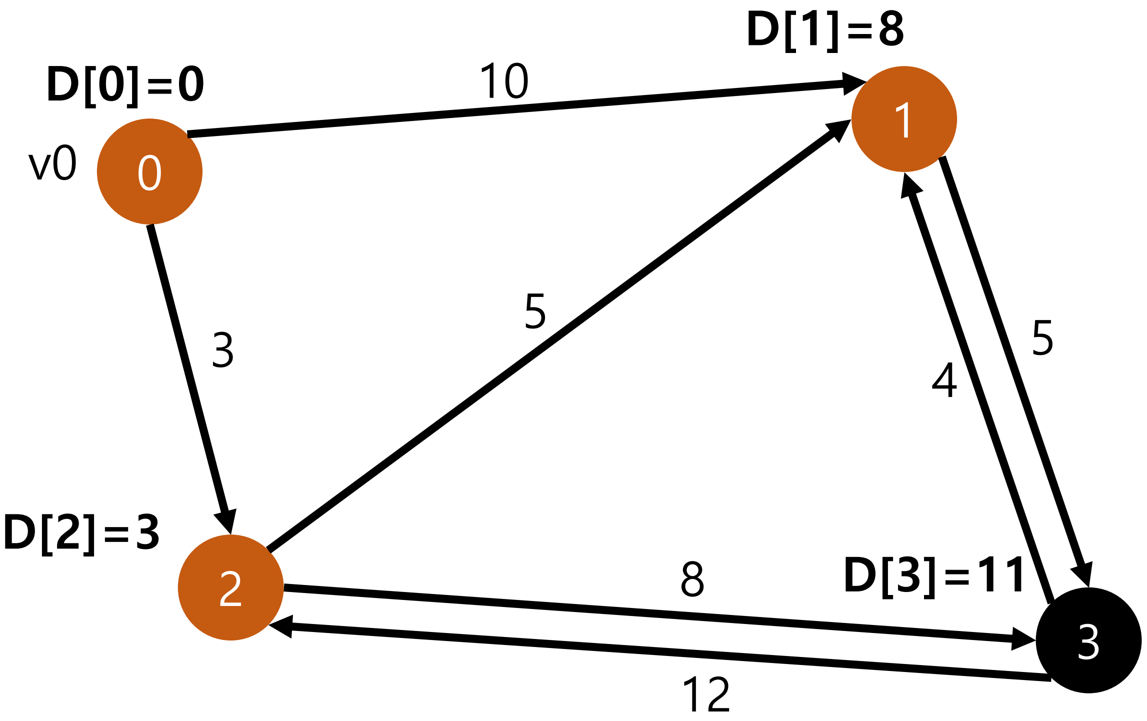
Dijkstra 알고리즘

S={0, 2}



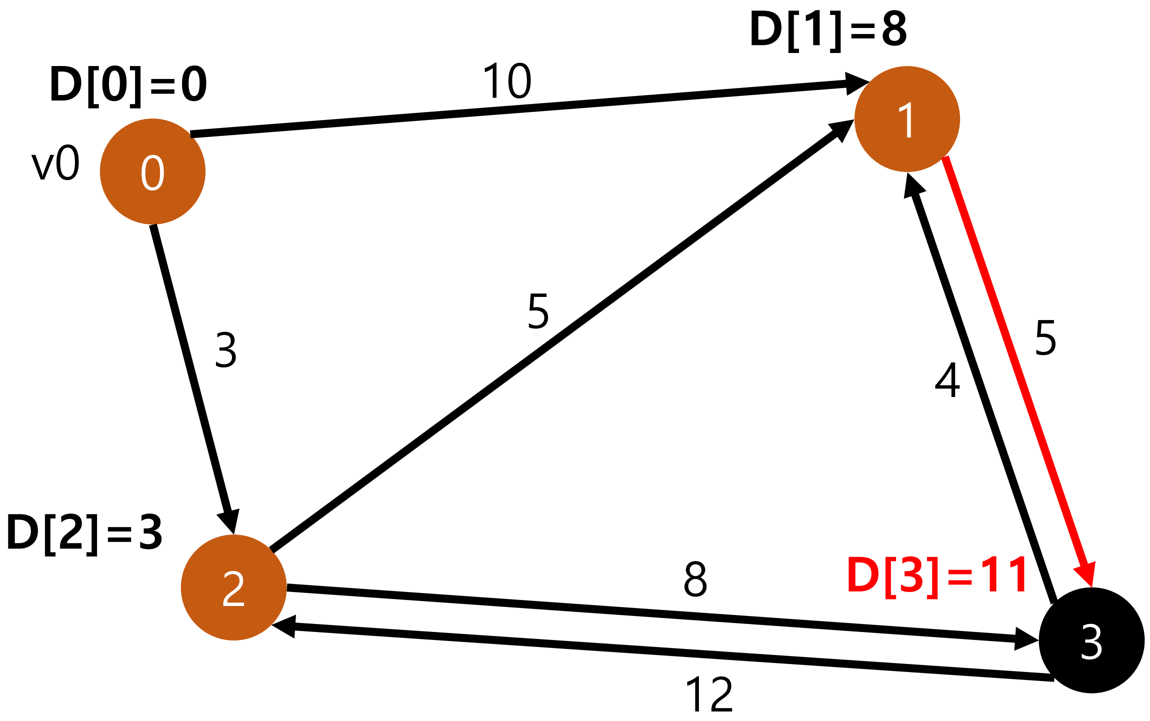
Dijkstra 알고리즘

S={0, 2, 1}



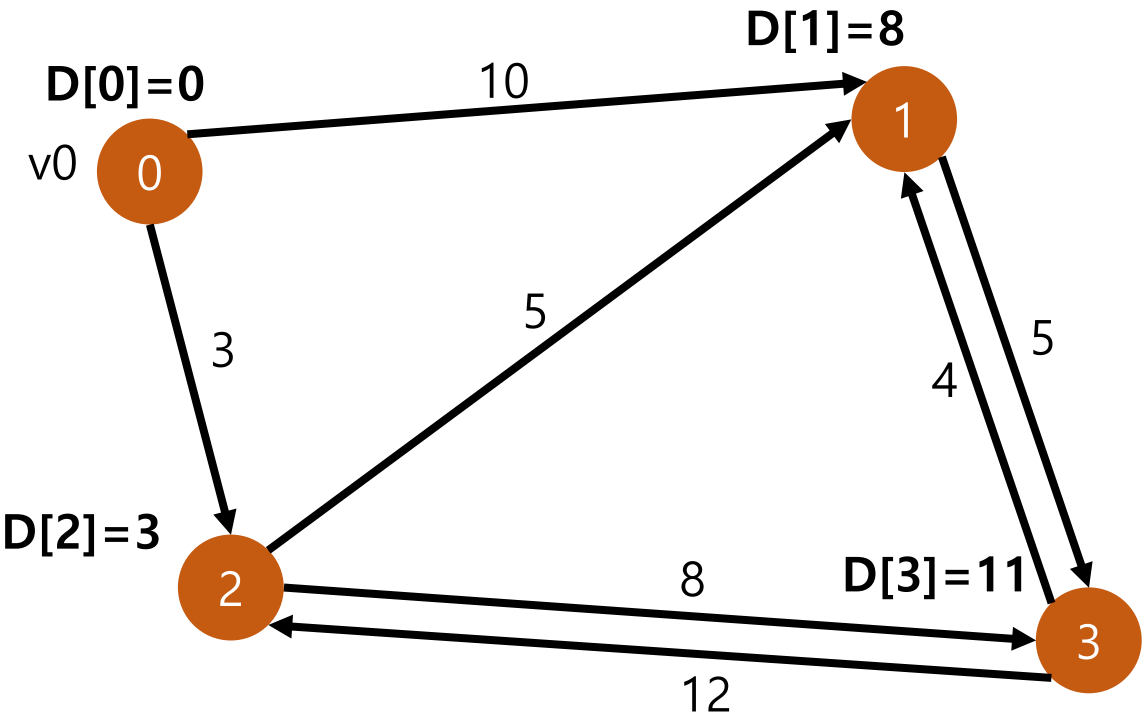
Dijkstra 알고리즘

S={0, 2, 1}



Dijkstra 알고리즘

S={0, 2, 1, 4}



Floyd-Warshall 알고리즘

Floyd-Warshall 알고리즘

1. Dynamic Programming
2. 모든 (출발점, 목적지) 쌍에 대한 최단 경로

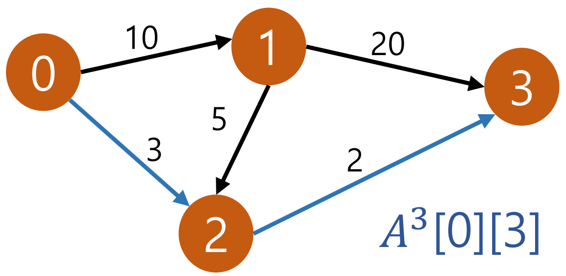
Floyd-Warshall 알고리즘

0~k 정점만 거치면서 i에서 j까지 가는 최단 경로의 길이

1. Recursion
   $\boxed{A^{k}[\mathrm{i}][\mathrm{j}]}$=min{$A^{k-1}[i][j]$, $A^{k-1}[\mathrm{i}][\mathrm{k}]+A^{k-1}[\mathrm{k}][\mathrm{j}]$}
2. Base case
   $A^{-1}[\mathrm{i}][\mathrm{j}]$=w(i, j)

$\boldsymbol{A^{1}[0][3]}$ **:** 0, 1 정점만 거쳐서 0에서 3까지 가는 최단 경로의 길이

→ **min{$\underset{\infty}{A^{0}[0][3]}$, $\underset{10}{A^{0}[0][1]}+\underset{20}{A^{0}[1][3]}$}**

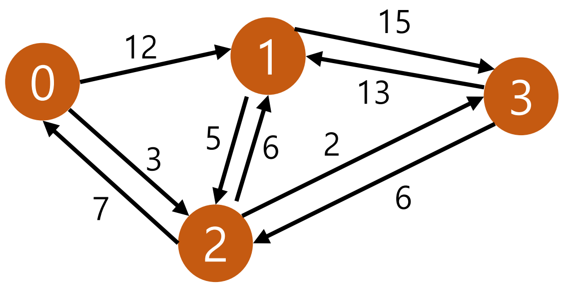


$A^{3}[0][3]$

정점 i에서 정점 j의 최단 경로 길이

$\boldsymbol{A^{n-1}[\mathrm{i}][\mathrm{j}]}$

Floyd-Warshall 알고리즘

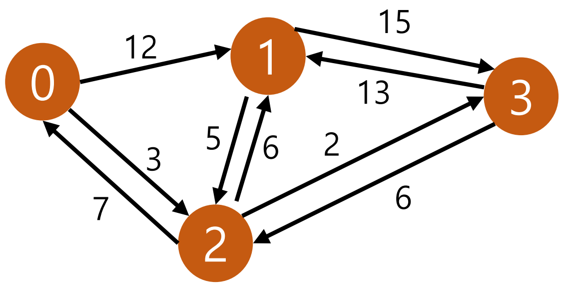


$\boldsymbol{A^{-1}}$

노드 i에서 j까지
아무 노드도 거치지 않는 경로
즉, 노드 i와 j가 인접하면 A^{-1}[i][j]>0
그렇지 않으면 A^{-1}[i][j]=∞

A^{-1}	0	1	2	3
0	0	12	3	∞
1	∞	0	5	15
2	7	6	0	2
3	∞	13	6	0

Floyd-Warshall 알고리즘

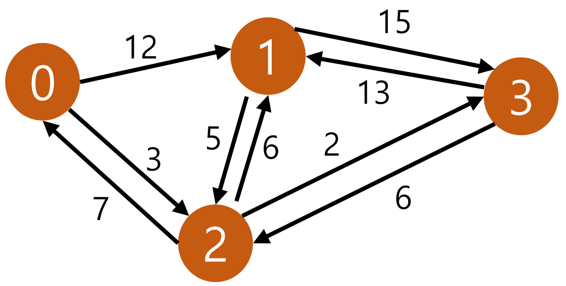


$\boldsymbol{A^0}$

노드 i에서 j까지
정점 0을 거치거나 안 거칠 때 경로
A^0[i][j]= min{A^{-1}[i][j], A^{-1}[i][0]+A^{-1}[0][j]}

A^0	0	1	2	3
0	0	12	3	∞
1	∞	0	5	15
2	7	6	0	2
3	∞	13	6	0

Floyd-Warshall 알고리즘

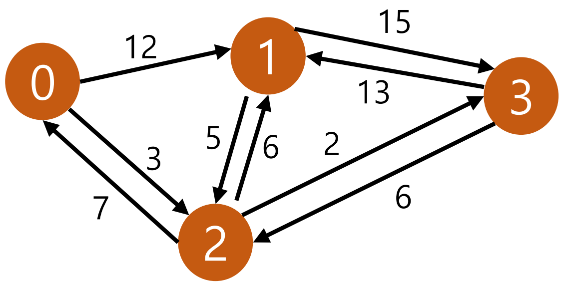


A^1

A^1	0	1	2	3
0	0	12	3	27
1	∞	0	5	15
2	7	6	0	2
3	∞	13	6	0

노드 i에서 j까지
정점 0, 1을 거치거나 안 거칠 때 경로
A^1[i][j]= min{A^0[i][j], A^0[i][1]+A^0[1][j]}

Floyd-Warshall 알고리즘

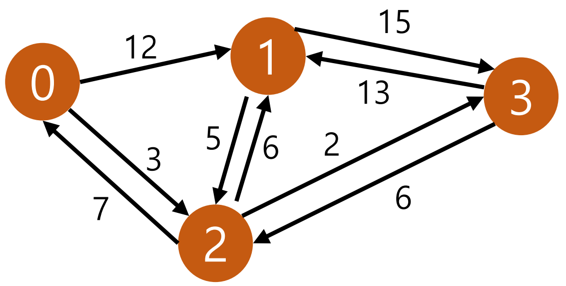


A^2

노드 i에서 j까지
정점 0, 1, 2을 거치거나 안 거칠 때 경로
A^2[i][j]= min{A^1[i][j], A^1[i][2]+A^1[2][j]}

A^2	0	1	2	3
0	0	9	3	5
1	12	0	5	7
2	7	6	0	2
3	13	12	6	0

Floyd-Warshall 알고리즘

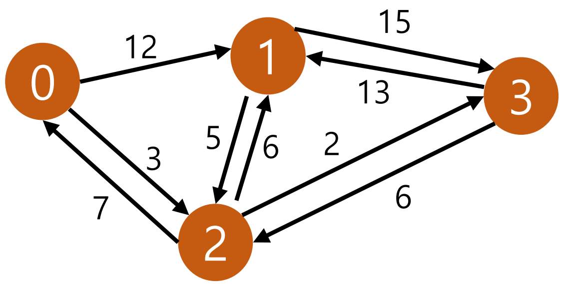


A^3	0	1	2	3
0	0	9	3	5
1	12	0	5	7
2	7	6	0	2
3	13	12	6	0

A^3

노드 i에서 j까지
정점 0, 1, 2, 3을 거치거나 안 거칠 때 경로
A^3[i][j]= min{A^2[i][j], A^2[i][3]+A^2[3][j]}

Floyd-Warshall 알고리즘



$\boldsymbol{A^3[\mathrm{i}][\mathrm{j}]}$

정점 i에서 정점[j]까지의
최단 경로 길이

A^3	0	1	2	3
0	0	9	3	5
1	12	0	5	7
2	7	6	0	2
3	13	12	6	0

2차원 배열 하나만 쓰면 되는 이유

A^2[i][j]= min{A^1[i][j], A^1[i][2]+A^1[2][j]}
→ A^2 행렬을 구하려면 A^1 배열도 있어야
할 것 같지만 그렇게 하지 않고
구한 값을 그냥 덮어쓰면 된다.

A^k[i][k]= A^{k-1}[i][k]이고 A^k[k][j]= A^{k-1}[k][j]

2차원 배열 하나만 쓰면 되는 이유

A^2	0	1	2	3
0	0	9	3	5
1	12	0	5	7
2	7	6	0	2
3	13	12	6	0

$A^2[3][1]$을 계산할 때 $A^2[2][1]$은 이미 계산이 되어 $A^1[2][1]$이 아니지만 정의에 의해 $A^2[2][1]$와 $A^1[2][1]$가 같으므로 배열 하나에서 계산하여 덮어쓰면 된다.