1 그래프의 활용

1 그래프의 활용

1 가중치 그래프(Weighted Graph)

- 그래프의 정점을 연결하는 간선마다 일정한 값을 할당한 그래프
- 간선에 할당하는 값은 각 정점 간의 거리나 비용과 같은 속성이 될 수 있음
  - 예) 정점이 도시를, 간선이 건설할 도로를 나타낸다면 도로의 건설비용을 간선의 가중치에 나타낼 수도 있음

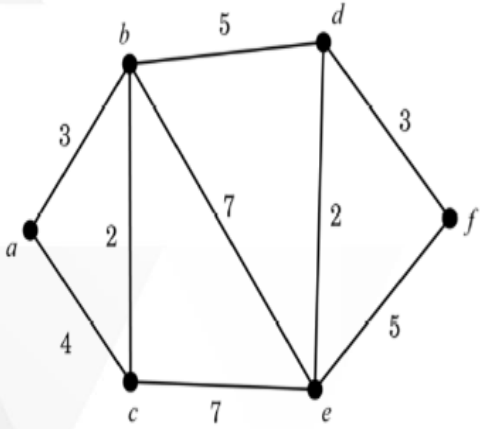
※ 가중치 그래프의 정의

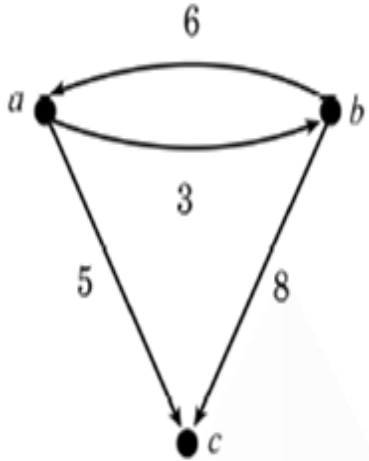
- 그래프 G=(V, E)에서 각 간선에 가중치(Weight)를 부여한 그래프를 의미

1 그래프의 활용

1 가중치 그래프(Weighted Graph)

가중치 그래프의 예





2 최단 경로 문제

2 최단 경로 문제

1 최단 경로(Shortest Path)

- 두 정점 사이에 존재하는 여러 경로 중 가중치의 합이 가장 짧은 경로
  - 예) 한 지점에서 다른 지점으로 갈 때 가장 빠른 길을 찾는 것과 비슷한 문제이며 각 구간에서 걸리는 시간이 가중치가 될 수 있음

2 최단 경로 문제

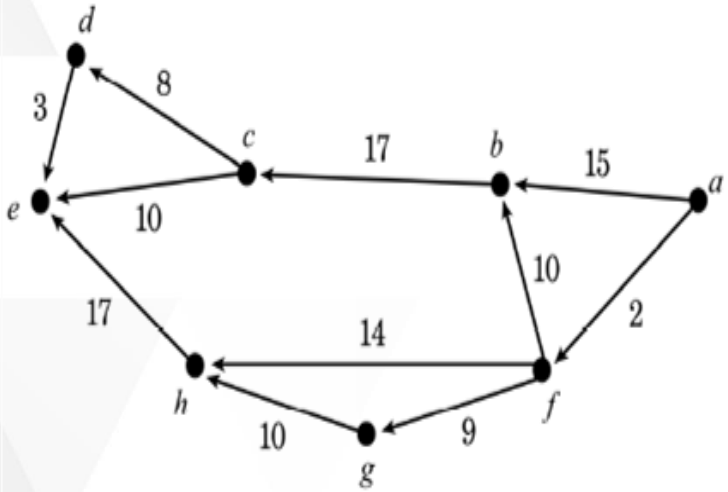
- 다양한 경로들 중 가장 효과적이고 경제적인 경로를 찾는 문제
- 도로망이나 네트워크 경로 등을 선택하는 데 응용할 수 있음
- 출발점부터 최종 도착점까지의 경로에 포함되는 정점을 나열하고 각 간선에 부여된 가중치를 더하여 가장 작은 가중치의 합을 갖는 경로를 선택해 가는 것

※ 최단 경로 문제의 정의

- |E| > 0인 그래프 G=(V, E)에서 정점 v_1,$[\![v]\!]\_2 \in V$ 사이의 가장 짧은 거리의 경로를 찾는 문제
  - 출발점(Source) : 경로의 시작점
  - 도착점(Destination) : 경로의 목적지

2 최단 경로 문제

- 가중치 그래프에서 최단 경로 구하기



(풀이)

① a-b-c-d-e : 15+17+8+3=43
② a-b-c-e : 15+17+10=42
③ a-f-b-c-d-e : 2+10+17+8+3=40
④ a-f-b-c-e : 2+10+17+10=39
⑤ a-f-h-e : 2+14+17=33
⑥ a-f-g-h-e : 2+9+10+17=38

2 최단 경로 문제

3 다익스트라 알고리즘(Dijkstra Algorithm)

- 시작점으로부터 최단 경로를 갖는 정점들을 차례로 탐색해 가는 알고리즘
- 한 정점에서 다른 모든 정점 간의 최단경로를 구함
- 시작 정점은 거리를 0으로, 다른 모든 정점은 거리를 무한대(∞)로 하여 시작
- 거리가 최소인 정점을 선택하고 이것에 인접한 정점을 거리를 최단 거리로 변경하는 과정을 반복함
- 간선의 가중치가 모두 '양수'일 때 유효함

2 최단 경로 문제

3 다익스트라 알고리즘(Dijkstra Algorithm)

적용 방법

1. 시작 정점에서 가장 인접한 정점을 찾는다. 그 정점까지의 거리가 최단거리이다. 지금까지 최단거리가 알려진 정점은 2개(자기 자신과 지금 찾은 정점)가 된다.
   최단거리가 알려진 정점들의 집합을 S라 하자.

2. 집합 S에 포함되지 않은 정점들 중에서 시작 정점으로부터 가장 가까운 정점을 찾는다. 이 새로운 정점은 집합 S에 바로 이웃한 정점들 중 하나일 것이다.
   그 정점까지의 거리는 최단거리이고 그 정점을 집합 S에 포함시킨다.

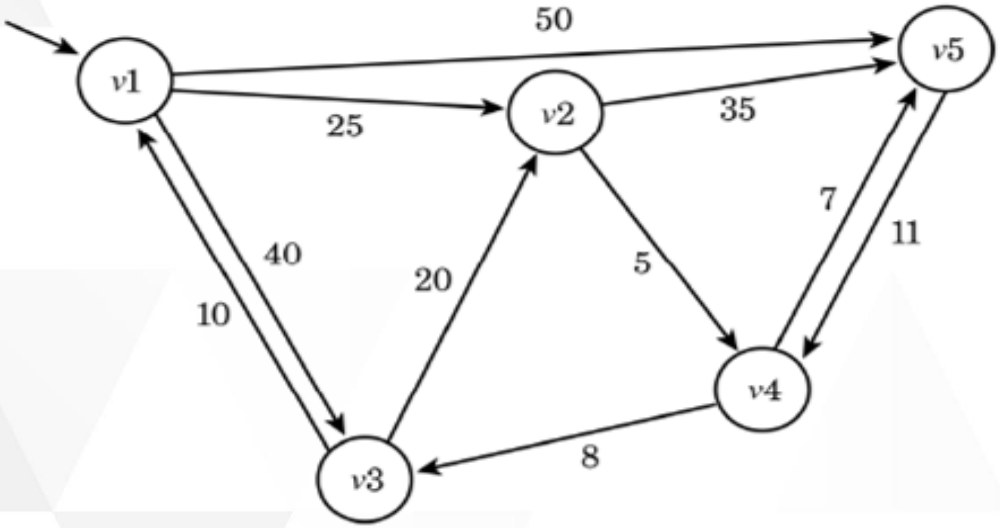
3 다익스트라 알고리즘(Dijkstra Algorithm)

적용 방법

3. 새로운 정점이 없을 때까지, 즉 모든 정점이 집합 S에 포함될 때까지 ②의 과정 반복한다.

3 다익스트라 알고리즘(Dijkstra Algorithm)

가중치 그래프에 다익스트라 알고리즘 적용



2 최단 경로 문제

3 다익스트라 알고리즘(Dijkstra Algorithm)

다이스트라 알고리즘 적용표

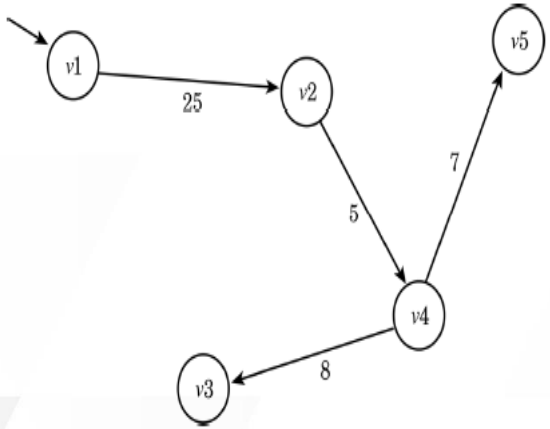
단계	정점의 집합	선택 정점	선택 정점에서 해당 정점 사이의 가중치				
			$v1$	$v2$	$v3$	$v4$	$v5$
초기	$v1$	$v1$	0	25	40	∞	50
1	$v1, v2$	$v2$	0	25	40	30	50
2	$v1, v2, v4$	$v4$	0	25	38	30	37
3	$v1, v2, v4, v5$	$v5$	0	25	38	30	37
4	$v1, v2, v3, v4, v5$	$v3$	0	25	38	30	37

2 최단 경로 문제

3 다익스트라 알고리즘(Dijkstra Algorithm)

최단경로

- 다익스트라 알고리즘이 모두 완료되면 다음과 같은 최단 경로를 얻을 수 있음

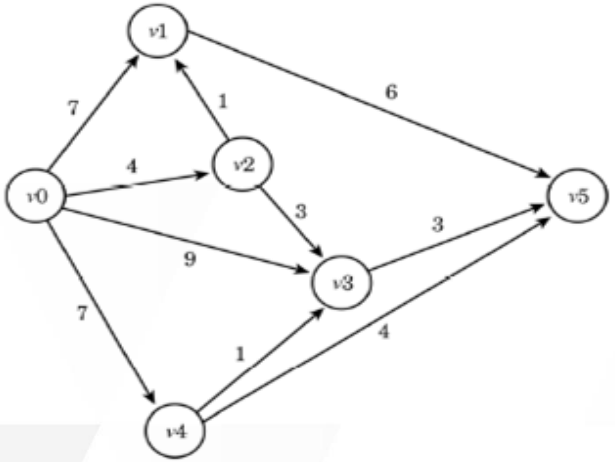


2 최단 경로 문제

3 다익스트라 알고리즘(Dijkstra Algorithm)

◆ 예시 1

다익스트라 알고리즘을 이용하여 다음 그래프의 최단 경로를 구하시오.(단, 시작 정점은 v0)



3 다익스트라 알고리즘(Dijkstra Algorithm)

◆ 예시 1

(풀이)
초기 단계에서는 정점 v0 에
연결된 정점들의 가중치를 계산함,
이때 v5는 직접적으로 연결되지
않았으므로 ∞가 됨

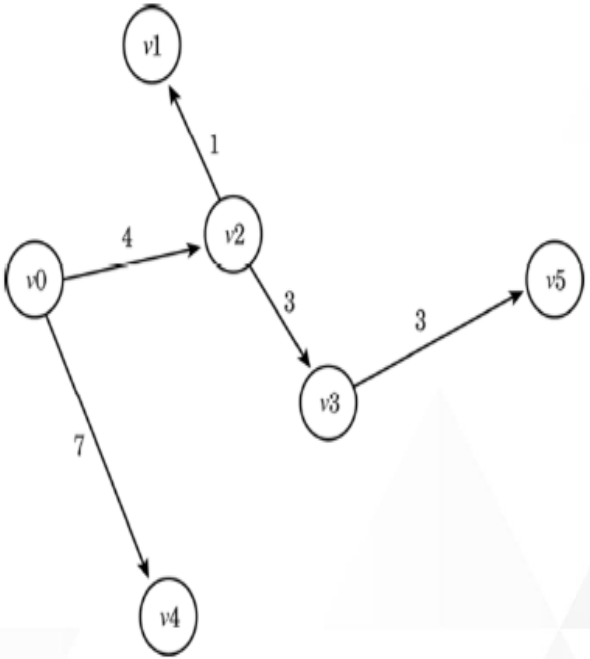
초기 단계가 완료되면 v0에서 v2로
연결된 경로의 가중치가 4로 가장
작으므로 정점 v2가 선택된 후
1단계가 시작됨
이러한 과정을 단계별로 나타내면 다음과 같음

단계	정점의 집합	선택 정점	선택 정점에서 해당 정점 사이의 가중치					
			v0	v1	v2	v3	v4	v5
초기	v0	v0	0	7	4	9	7	∞
1	v0, v2	v2	0	5	4	7	7	∞
2	v0, v2, v1	v1	0	5	4	7	7	11
3	v0, v2, v1, v3	v3	0	5	4	7	7	10
4	v0, v2, v1, v3, v4	v4	0	5	4	7	7	10
5	v0, v2, v1, v3, v4, v5	v5	0	5	4	7	7	10

3 다익스트라 알고리즘(Dijkstra Algorithm)

- 예시 1

(풀이)
따라서 최종적으로
얻어진 최단경로는
다음과 같음



3 경로의 존재

3 경로의 존재

1 경로의 존재

- 정점 v_1과 v_2를 연결하는 간선 $\langle v_1, v_2 \rangle$가 존재한다면 v_1에서 v_2로 가는 길이가 1인 경로가 있다는 의미

  - E^1 : 길이가 1인 경로
  - E^2 : 길이가 2인 경로
  - E^k : 길이가 k인 경로

$$E^1 = E$$
$$E^k = E^{k-1} o\ E \ (k \geq 2)$$

- 두 정점 사이에 경로가 존재하는지 여부는 $E^* = E \cup E^2 \cup E^3 \cup \cdots$ 를 구하면 됨

3 경로의 존재

2 도달 행렬(Reachability Matrix)

- 방향 그래프에서 정점 간 경로의 존재 여부를 행렬로 표현한 것
- A^*를 도달행렬이라고 하고 행렬 A에 대한 거듭제곱은 부울 곱 연산, 합은 부울 합 연산

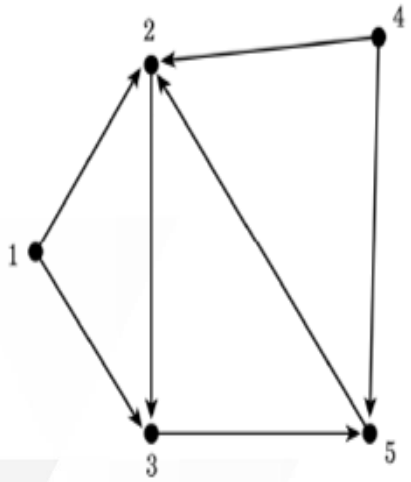
> E에 대한 인접행렬이 A 이고 E^*에 대한 인접행렬이 A^* 이면
>
> $$A^* = A + A^2 + A^3 + \cdots + A^n$$

3 경로의 존재

2 도달 행렬(Reachability Matrix)

◆ 예시 1

다음 방향 그래프에 대한 도달행렬을 구하시오.



2 도달 행렬(Reachability Matrix)

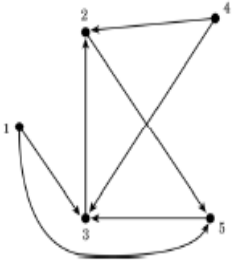
◆ 예시 1

(풀이)

$$A=\begin{array}{c} \\ 1\\2\\3\\4\\5\end{array}\begin{array}{c}\begin{array}{ccccc}1&2&3&4&5\end{array}\\\begin{bmatrix}0&1&1&0&0\\0&0&1&0&0\\0&0&0&0&1\\0&1&0&0&1\\0&1&0&0&0\end{bmatrix}\end{array}$$

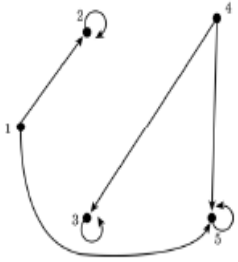
이제 A^2부터 A^5까지 구하면 다음과 같다.

$$A^2=\begin{array}{c} \\ 1\\2\\3\\4\\5\end{array}\begin{array}{c}\begin{array}{ccccc}1&2&3&4&5\end{array}\\\begin{bmatrix}0&0&1&0&1\\0&0&0&0&1\\0&1&0&0&0\\0&1&1&0&0\\0&0&1&0&0\end{bmatrix}\end{array}$$



〈A^2와 방향 그래프〉

$$A^3=\begin{array}{c} \\ 1\\2\\3\\4\\5\end{array}\begin{array}{c}\begin{array}{ccccc}1&2&3&4&5\end{array}\\\begin{bmatrix}0&1&0&0&1\\0&1&0&0&0\\0&0&1&0&0\\0&0&1&0&1\\0&0&0&0&1\end{bmatrix}\end{array}$$



〈A^3와 방향 그래프〉

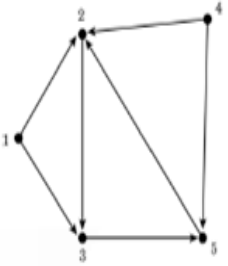
3 경로의 존재

2 도달 행렬(Reachability Matrix)

- 예시 1

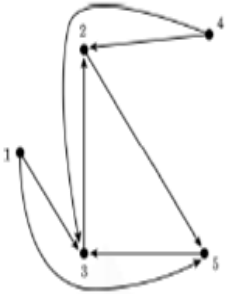
(풀이)

$$A^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



〈A^4와 방향 그래프〉

$$A^5 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$



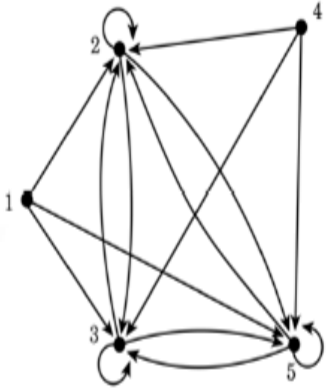
〈A^5와 방향 그래프〉

2 도달 행렬(Reachability Matrix)

◆ 예시 1

(풀이) 따라서 $A^* = A + A^2 + \cdots + A^5$이므로 A^*는 A^1부터 A^5까지의 불 합을 계산하여 구할 수 있다.

$$A^* = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix} \end{array}$$



〈A^*와 방향 그래프〉

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

- 그래프에서 모든 정점 사이의 최단 거리를 구하기 위한 알고리즘
- 모든 정점에서 모든 정점으로의 최단거리를 구하는 것
  - 다익스트라 알고리즘은 어느 한 지점으로부터 모든 정점으로의 최단거리를 구함
- 정점 A에서 B로 가는 경로와 정점 B에서 C로 가는 경로가 존재하면 A에서 C로 가는 경로도 존재한다고 할 수 있음

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

- 그래프 없이 인접행렬만을 보고 이행 폐쇄를 결정하는 방법
- 가중치가 음수일 때도 경로 구할 수 있음
  - 다익스트라는 가중치가 양수일 때만 구할 수 있음

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

- 한 정점에서 목적지 정점까지의 거리가 다른 정점들을 거쳐서 가는 가중치보다 크다면 거쳐서 가는 거리값을 저장해 나가는 방법
  - 예) 두 정점 i, j가 존재한다고 가정할 때, i에서 j로 가는 것이 빠르냐? 아니면 i에서 k를 거쳐서 j로 가는게 빠르냐? 이 두 가지 조건을 비교함

- 반복문 3개를 정점 수만큼 수행하기 때문에 시간 복잡도는 $O(n^3)$
  - 3중 for문의 형태 사용

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

- 최단 경로 값을 저장하는
  이중 배열 dist[i][j]가 있다고 하면

  Dist[4][2]=10 이 저장되어 있다면
  현재까지 계산된 정점4에서 정점2까지의
  최단 경로는 10이라는 의미임

  그런데 **dist[4][1]=1, dist[1][2]=2** 라고 하면
  정점4에서 정점1까지의 경로값은 1이고
  다시 정점1에서 정점2까지의 경로값은 2이므로
  정점4가 정점1을 거쳐서 정점2에 가는 경로인
  '정점4→ 정점1 → 정점2 ' 의 경로값은 총 3이 됨

3 경로의 존재

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

- 최단 경로 값을 저장하는
  이중 배열 dist[i][j]가 있다고 하면

  Dist[4][2]=10 이 저장되어 있다면
  현재까지 계산된 정점4에서 정점2까지의
  최단 경로는 10이라는 의미임

  그런데 **dist[4][1]=1, dist[1][2]=2** 라고 하면
  정점4에서 정점1까지의 경로값은 1이고
  다시 정점1에서 정점2까지의 경로값은 2이므로
  정점4가 정점1을 거쳐서 정점2에 가는 경로인
  '정점4→ 정점1 → 정점2 ' 의 경로값은 총 3이 됨

따라서 기존에 저장된 dist[4][2]=10 값보다 작으므로

3 경로의 존재

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

- 최단 경로 값을 저장하는
  이중 배열 dist[i][j]가 있다고 하면

  Dist[4][2]=10 이 저장되어 있다면
  현재까지 계산된 정점4에서 정점2까지의
  최단 경로는 10이라는 의미임

  그런데 **dist[4][1]=1, dist[1][2]=2** 라고 하면
  정점4에서 정점1까지의 경로값은 1이고
  다시 정점1에서 정점2까지의 경로값은 2이므로
  정점4가 정점1을 거쳐서 정점2에 가는 경로인
  '정점4→ 정점1 → 정점2 ' 의 경로값은 총 3이 됨

dist[4][2]
=dist[4][1] + dist[1][2]

3 경로의 존재

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

- 최단 경로 값을 저장하는
  이중 배열 dist[i][j]가 있다고 하면

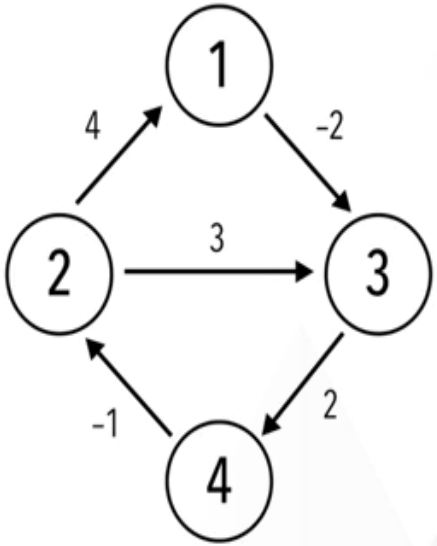
  Dist[4][2]=10 이 저장되어 있다면
  현재까지 계산된 정점4에서 정점2까지의
  최단 경로는 10이라는 의미임

  그런데 **dist[4][1]=1, dist[1][2]=2** 라고 하면
  정점4에서 정점1까지의 경로값은 1이고
  다시 정점1에서 정점2까지의 경로값은 2이므로
  정점4가 정점1을 거쳐서 정점2에 가는 경로인
  '정점4→ 정점1 → 정점2 ' 의 경로값은 총 3이 됨

즉, dist[4][2]= 1+2 = 3
으로 되어 새로운 최단 거리
값이 저장됨

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

```
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]
        dist [i][j] ← dist [i][k] + dist [k][j]
      end if
```



	1	2	3	4
1	0	∞	-2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	∞	-1	∞	0

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

```
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]
        dist [i][j] ← dist [i][k] + dist [k][j]
      end if
```

k = 1 2 3 4
i = 1 2 3 4
j = 1 2 3 4

dist [i][j] > dist [i][k] + dist [k][j]
dist [1][1] > dist [1][1] + dist [1][1]
0 > 0 + 0
X 0 > 0

k = 1 2 3 4
i = 1 2 3 4
j = 1 2 3 4

dist [i][j] > dist [i][k] + dist [k][j]
dist [1][2] > dist [1][1] + dist [1][2]
∞ > 0 + ∞
X ∞ > ∞

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

k = 1 2 3 4
i = 1 2 3 4
j = 1 2 3 4

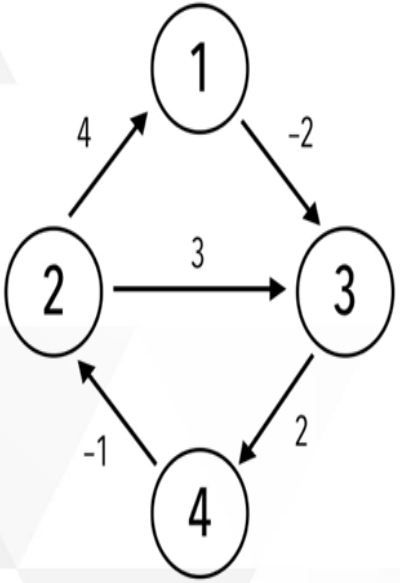
dist [i][j] > dist [i][k] + dist [k][j]
dist [2][3] > dist [2][1] + dist [1][3]
3 > 4 + -2
3 > 2

가중치 값이
새로운 값으로 갱신됨!!

	1	2	3	4
1	0	∞	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	∞	-1	∞	0

3 플로이드 와샬의 알고리즘(Floyd-Warshall Algorithm)

- 플로이드의 알고리즘 적용한 후 얻어진 최종 결과

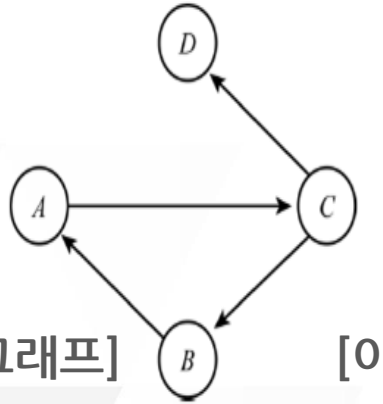


	1	2	3	4
1	0	-1	-2	0
2	4	0	2	4
3	5	1	0	2
4	3	-1	1	0

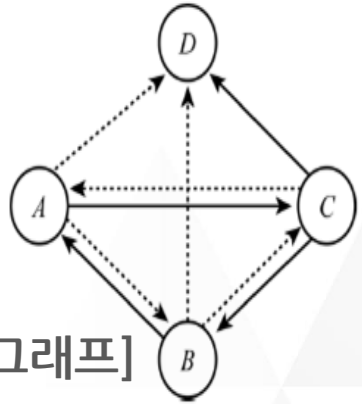
3 경로의 존재

4 이행폐쇄(Transitive Closure)

- 거쳐서 갈 수 있는 모든 곳을 직접 가는 간선으로 연결한 그래프
- 간선의 추가를 통해서 대부분의 노드들이 인접 노드로 바뀌게 되며 인접행렬로 표현하는 것이 편리



[방향 그래프]



[이행폐쇄 그래프]

3 경로의 존재

5 그래프 탐색

- 그래프에 연결된 모든 정점을 탐색하기 위해서는 임의의 정점을 선택하여 탐색하는 방법보다는 시작점을 기준으로 일정한 방향으로 탐색하는 것이 효율적

- 대표적으로 **깊이우선탐색**과 **너비우선탐색**이 있음

5 그래프 탐색

깊이우선탐색(DFS: Depth First Search)

- 그래프 탐색 방법 중 하나
- 어떠한 시작점 v_1에서 인접해 있는 정점 중 아직 탐색하지 않은 정점 v_2를 방문하고 정점 v_2에 인접해 있는 정점 중 아직 탐색하지 않은 정점 v_3을 방문하는 것을 반복
- v_n에 인접한 모든 정점을 방문한 경우 이전에 마지막으로 방문한 정점 v_{n-1}로 돌아가서 인접해 있는 정점 중 방문하지 않은 정점을 다시 방문하는 과정을 반복

5 그래프 탐색

깊이우선탐색(DFS: Depth First Search) 과정

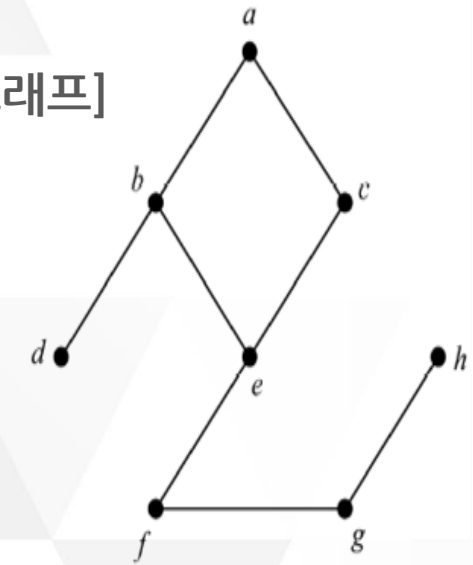
① 시작점 v 를 탐색
② 정점 v 에 인접한 정점들 중
탐색되지 않은 정점 v_i를 탐색
③ 정점 v_i를 v로 하여 ②를 반복
④ 더 이상 탐색되지 않은 정점이 없으면
이전 탐색한 정점을 v 로 하여 ②와 ③을 반복
⑤ 그래프의 모든 정점을 탐색할 때까지 반복

3 경로의 존재

5 그래프 탐색

깊이우선탐색(DFS: Depth First Search) 방법 예

[그래프]



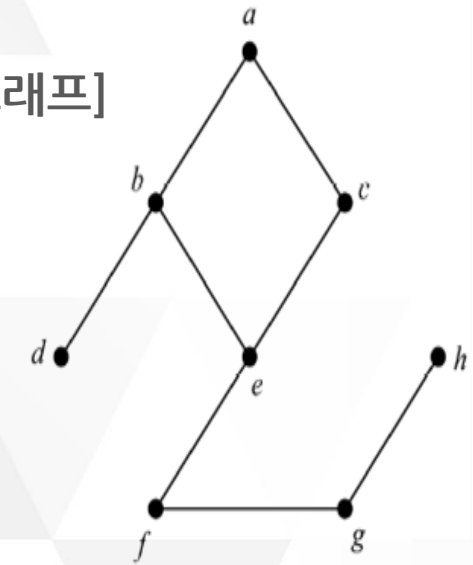
1) 정점 a를 시작점으로 하면 a와 인접된 정점 중 탐색되지 않은 정점은 b와 c로, 만약b를 먼저 탐색한다면 경로는 a-b
2) 이제 정점 b를 기준으로 b와 연결된 정점은 d와 e가 있는데 d를 먼저 탐색하면 경로는 a-b-d
3) 정점 d는 인접하고 탐색되지 않은 정점은 없으므로 다시 b로 돌아감
4) 정점 b와 인접하면서 탐색되지 않은 정점은 e이므로 e를 탐색하면 경로는 a-b-d-e

3 경로의 존재

5 그래프 탐색

깊이우선탐색(DFS: Depth First Search) 방법 예

[그래프]



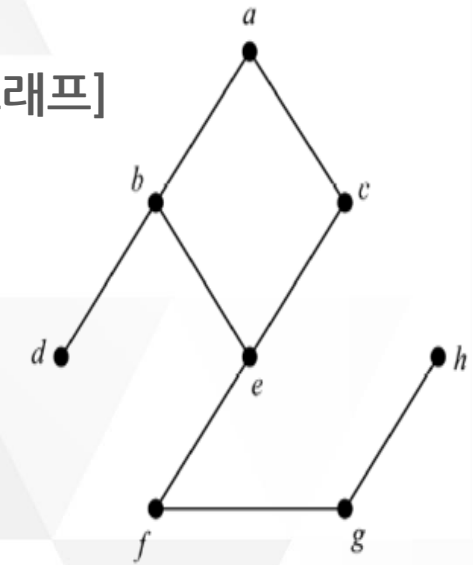
5) 정점 e와 인접하면서 탐색되지 않은 정점은 f와 c가 있음, f를 먼저 탐색하면 경로는 a-b-d-e-f
6) 정점 f와 인접하고 탐색되지 않은 정점 g가 있으므로 이를 탐색하면 경로는 a-b-d-e-f-g
7) g와 인접하고 탐색되지 않은 정점은 h가 있으므로 이를 탐색하면 경로는 a-b-d-e-f-g-h
8) 정점 h와 인접하고 탐색되지 않은 정점은 더 이상 없으므로 다시 g로 돌아감

3 경로의 존재

5 그래프 탐색

깊이우선탐색(DFS: Depth First Search) 방법 예

[그래프]

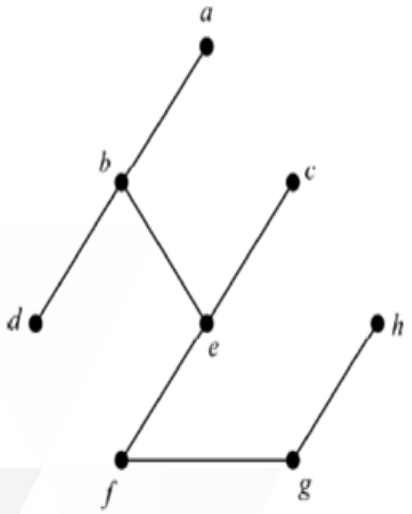


9) g와 인접하고 탐색되지 않은 정점은 더 이상 없으므로 다시 f로 돌아감
10) f와 인접하고 탐색도지 않은 정점은 더 이상 없으므로 e로 돌아감
11) e와 인접하고 탐색되지 않은 정점은 c가 있으므로 c를 탐색하면 경로는 다음과 같음, a-b-d-e-f-g-h-c

5 그래프 탐색

깊이우선탐색(DFS: Depth First Search) 결과

- 깊이우선탐색을 통해 얻어진 경로는 다음과 같음



3 경로의 존재

5 그래프 탐색

너비우선탐색(BFS: Breath First Search)

- 시작 정점을 방문한 후에 시작 정점에 인접한 모든 정점들을 우선 방문하는 방법
- 더 이상 방문하지 않은 정점이 없을 때까지 방문하지 않은 모든 정점들에 대해서도 넓이 우선 검색을 적용
- 시작점에 가까운 정점들을 먼저 탐색하고 시작점과 멀리 있는 정점들을 나중에 탐색

5 그래프 탐색

너비우선탐색(BFS: Breath First Search) 과정

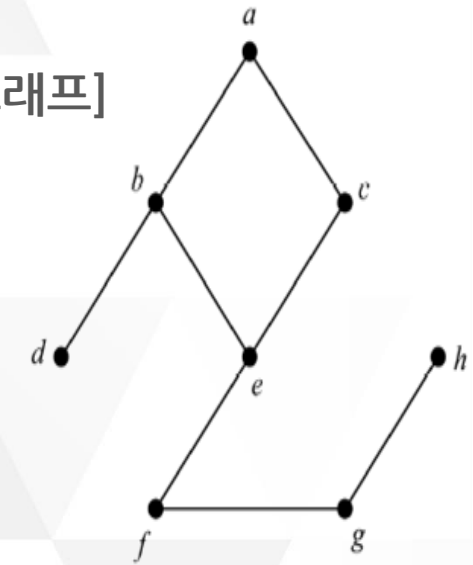
- 너비우선탐색은 시작점 v_1로 부터 인접한 정점 $v_{2_1}, \cdots, v_{2_n}$을 모두 탐색하고 다시 정점 v_{2_1}을 시작으로 인접한 정점 $v_{3_1}, \cdots, v_{3_n}$ 을 차례로 탐색하는 방법을 반복

3 경로의 존재

5 그래프 탐색

너비우선탐색(BFS: Breath First Search) 방법 예

[그래프]



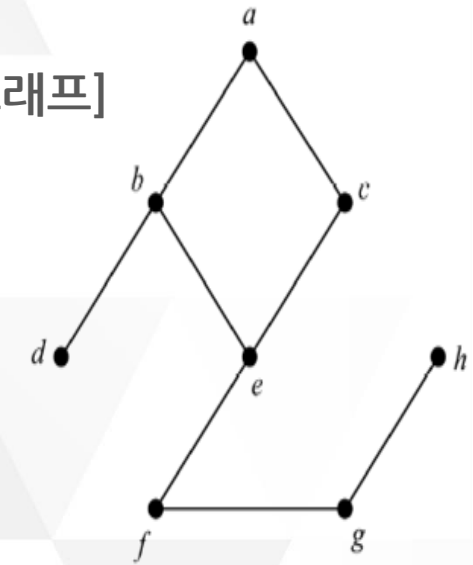
1) 정점 a와 인접하고 탐색되지 않은 정점은 b, c가 있음
   b부터 탐색하고 c를 탐색하면 경로는 a-b-c가 됨
2) 정점 b와 인접하고 탐색되지 않은 정점 d, e가 있음
   d부터 탐색하고 e를 탐색하면 경로는 a-b-c-d-e 가 됨
3) c와 인접하고 탐색되지 않은 정점은 없음
4) d와 인접하고 탐색되지 않은 정점은 없음

3 경로의 존재

5 그래프 탐색

너비우선탐색(BFS: Breath First Search) 방법 예

[그래프]



5) e와 인접하고 탐색되지 않은 정점은 f가 있음
f를 탐색하면 경로는 a-b-c-d-e-f가 됨
6) f와 인접하고 탐색되지 않은 정점은 g가 있으므로 g를 탐색하면 경로는 a-b-c-d-e-f-g가 됨
7) g에 인접하고 탐색되지 않은 정점은 h가 있으므로 h를 탐색하면 a-b-c-d-e-f-g-h 경로가 됨
8) h에 인접하고 탐색되지 않은 정점은 없음

5 그래프 탐색

너비우선탐색(BFS: Breath First Search) 결과

- 너비우선탐색을 통해 얻어진 경로는 다음과 같음