9주차 3차시 최소신장 나무와 최단경로

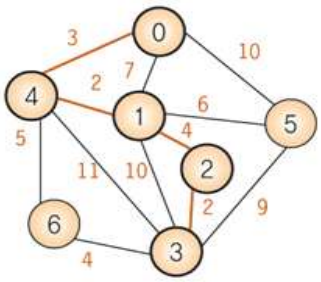
【학습목표】

1. 다이스트라 알고리즘을 이해할 수 있다.
2. 최단경로를 구할 수 있다.

학습내용1 : 다이스트라 알고리즘

1. 최단 경로(shortest path)

◎ 네트워크에서 정점 u와 정점 v를 연결하는 경로 중에서 간선들의 가중치 합이 최소가 되는 경로
◎ 간선의 가중치는 비용, 거리, 시간 등
◎ 정점 0에서 정점 3으로 가는 최단 경로 문제
- ☞ 인접행렬에서 간선이 없는 노드 쌍의 가중치는 ∞ 임
- ☞ 0,4,1,2,3이 최단 경로
- ☞ 최단경로 길이는 3+2+4+2=11



	0	1	2	3	4	5	6
0	0	7	∞	∞	3	10	∞
1	7	0	4	10	2	6	∞
2	∞	4	0	2	∞	∞	∞
3	∞	10	2	0	11	9	4
4	3	2	∞	11	0	∞	5
5	10	6	∞	9	∞	0	∞
6	∞	∞	∞	4	5	∞	0

* 유형
- ☞ 단일 출발점 최단 경로.(다이스트라 알고리즘)
- ☞ 단일 쌍 최단 경로.
- ☞ 단일 도착점 최단 경로.
- ☞ 모든 쌍 최단 경로.(플로이드 알고리즘)

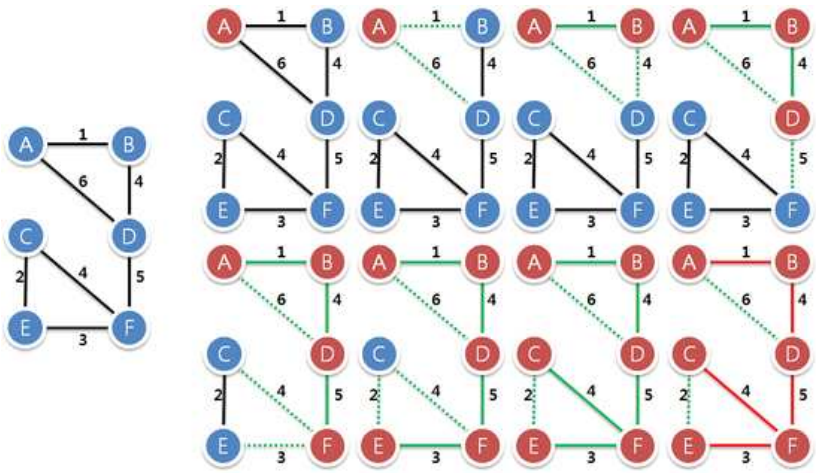
2. 단일 출발점 최단 경로

단일 시작점에서 최단 경로 구하기 문제는 임의의 시작점을 1개정하고 다른 노드들 사이의 최단 경로를 구하는 경우에 사용 된다. 여기서 노드 사이의 거리가 모두 양수인 경우로 가장 널리 사용되는 Dijkstar(다이스트라)알고리즘에 대해서 알아보자.

3. Dijkstra알고리즘

한 출발점에서 다른 모든 정점으로의 최단 경로를 구하는 문제로 가중치를 갖는 간선은 없음

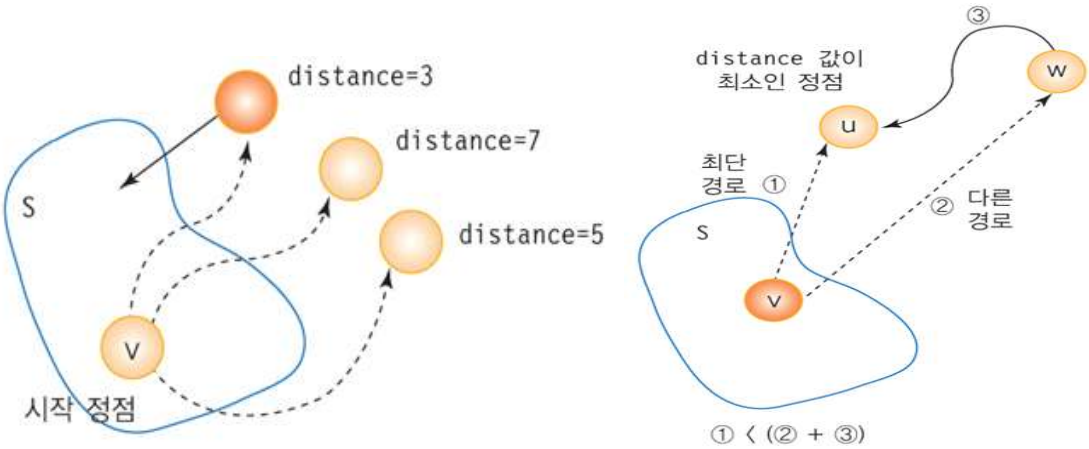
* 과정
㉠ 그림을 보면 알듯이 임의의 노드 'A'를 먼저 선택.
㉡ 임의의 노드가 정해 졌다면 임의의 노드에 연결된 간선 중 비용이 적게 드는 간선을 선택하여 인접 노드와 연결
㉢ 그림에서는 노드 'A'에 연결된 간선(A->B)는1, 간선(A->D)는6.
㉣ 여기서 간선(A->B)를 선택.
㉤ 이렇게 인접 노드와 연결이 되었다면 다시 임의의 노드와 인접 노드에 연결된 노드중 간선의 가중치가 적은 간선을 선택 하여 연결
㉥ 그림에서 보면 노드'B'는 노드 'A'에서 오는 간선의 가중치가 있으므로 포함하여 간선의 가중치 계산
㉦ 그림에서 보면 (A -> D)의 비용은6이고, (A->B->D)의 비용은5이다. 이렇듯 노드로 오기까지의 비용을 계산해야 한다.



▲ Dijkstra 방법의 적용 예

4. Dijkstra의 최단경로 알고리즘

◎ 하나의 시작 정점으로부터 모든 다른 정점까지의 최단 경로 찾음
◎ 집합 S
☞ 시작 정점 v로부터의 최단경로가 이미 발견된 정점들의 집합
◎ distance 배열
☞ 최단경로가 알려진 정점들만을 이용한 다른 정점들까지의 최단경로 길이
☞ distance 배열의 초기값(시작 정점 v)
→ distance[v] = 0
→ 다른 정점에 대한 distance 값은 시작정점과 해당 정점간의 가중치 값
◎ 매 단계에서 가장 distance 값이 작은 정점을 S에 추가

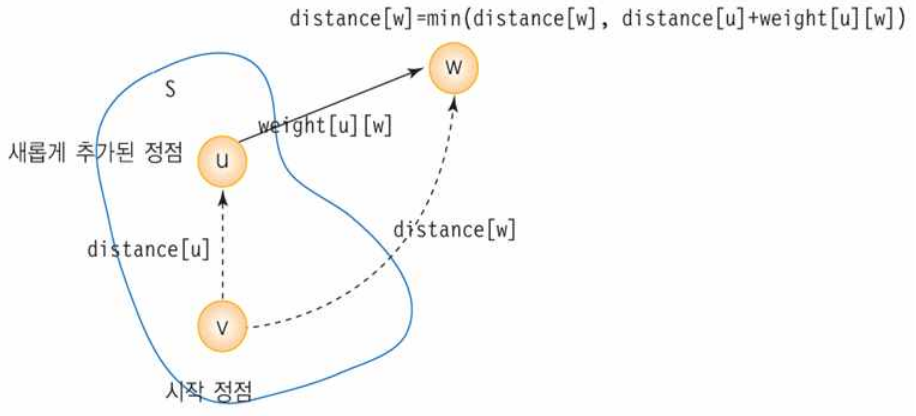


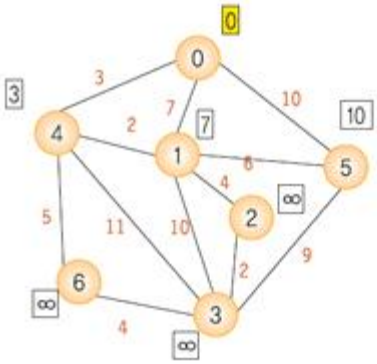
◎ distance 값이 가장 작은 정점을 u라고 하자. 그러면 시작 정점 v에서 정점 u까지의 최단거리는 경로 ①이 된다.

◎ 정점 w를 거쳐서 정점 u로 가는 가상적인 더 짧은 경로가 있다고 가정해보자, 그러면 정점 v에서 정점 u까지의 거리는 정점 v에서 정점 w까지의 거리 ②와 정점 w에서 정점 u로 가는 거리③을 합한 값이 된다.

◎ 그러나 경로 ②는 경로 ①보다 항상 길 수 밖에 없다. 왜냐하면 현재 distance 값이 가장 작은 정점은 u이기 때문이다.

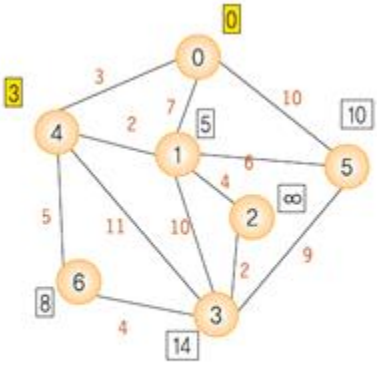
◎ 따라서 매 단계에서 distance 값이 가장 작은 정점들을 추가해나가면 시작 정점에서 모든 정점까지의 최단거리를 구할 수 있다.





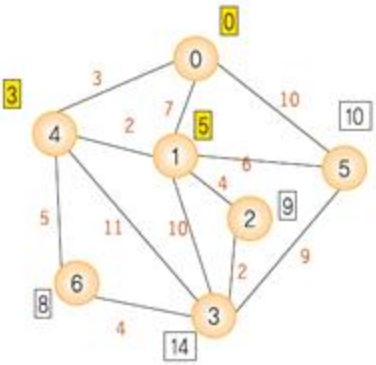
S={0}

distance[]	0	1	2	3	4	5	6
	0	7	∞	∞	3	10	∞



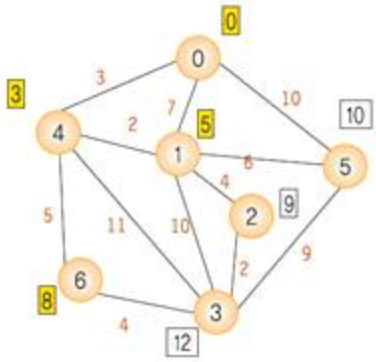
S={0, 4}

distance[]	0	1	2	3	4	5	6
	0	5	∞	14	3	10	8



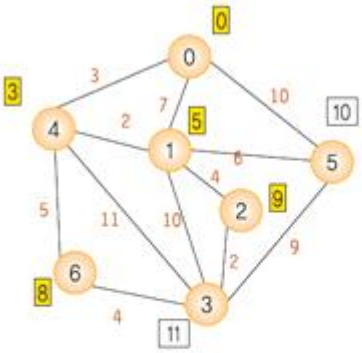
S={0, 4, 1}

distance[]	0	1	2	3	4	5	6
	0	5	9	14	3	10	8



S={0, 4, 1, 6}

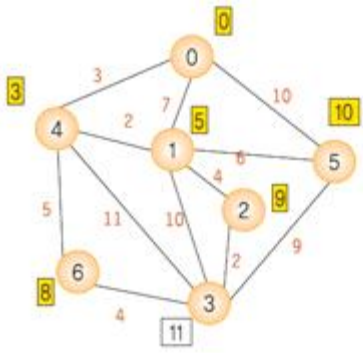
distance[]	0	1	2	3	4	5	6
	0	5	9	12	3	10	8



S={0, 4, 1, 6, 2}

distance[]=

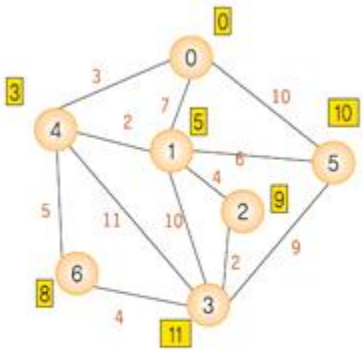
0	1	2	3	4	5	6
0	5	9	11	3	10	8



S={0, 4, 1, 6, 2, 5}

distance[]=

0	1	2	3	4	5	6
0	5	9	11	3	10	8

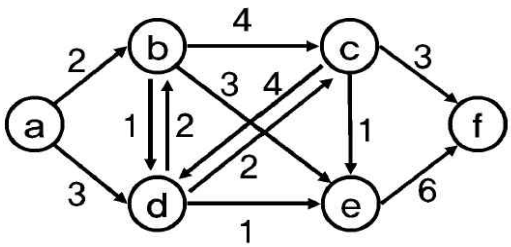


S={0, 4, 1, 6, 2, 5, 3}

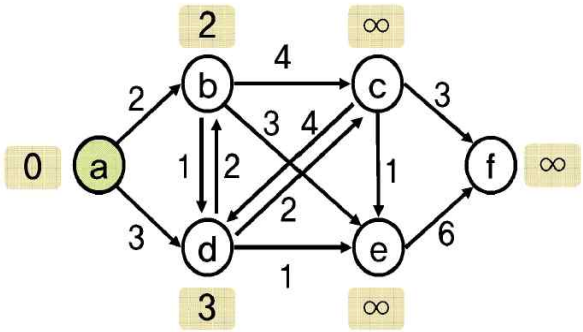
distance[]=

0	1	2	3	4	5	6
0	5	9	11	3	10	8

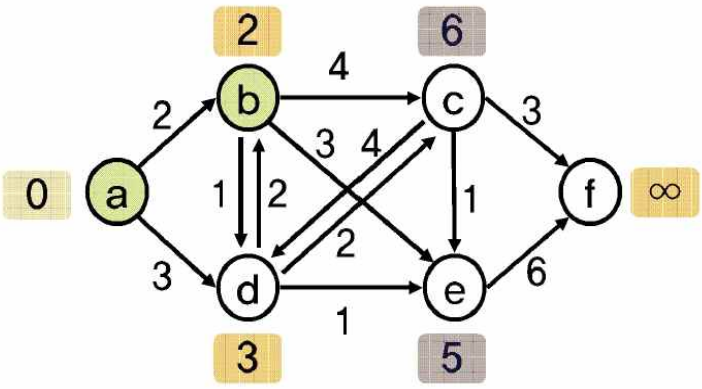
5. Dijkstra 방법의 적용 예

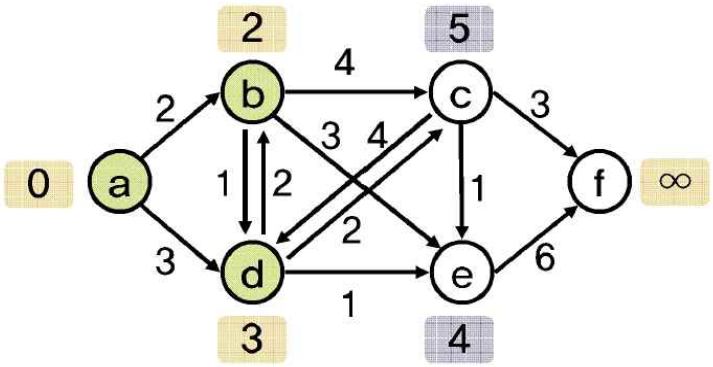


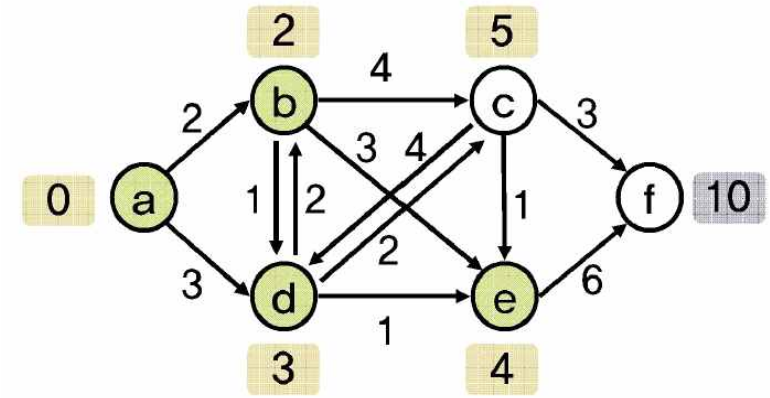
최단 경로 문제를 위한 그래프의 예

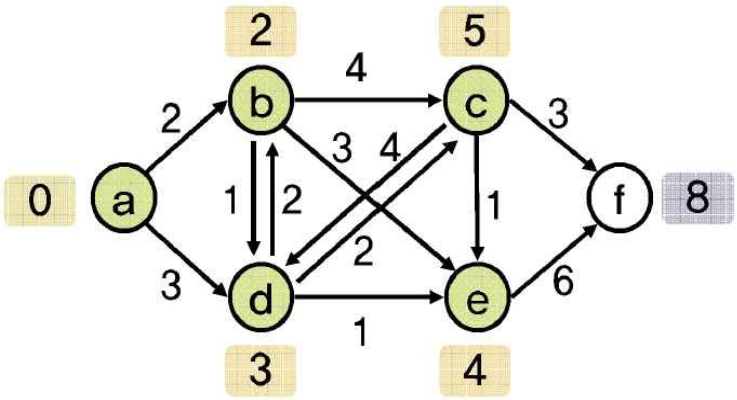


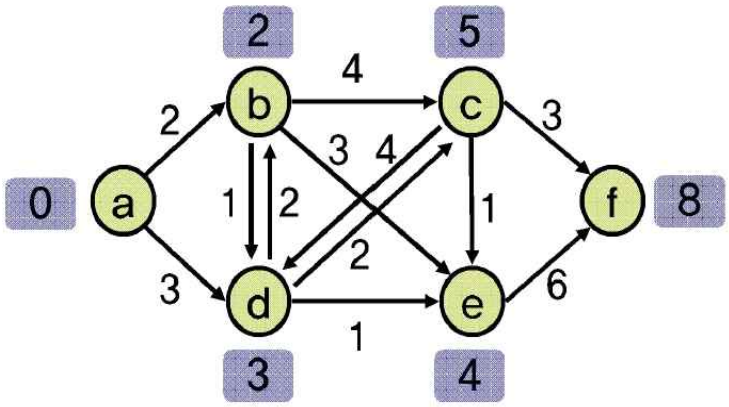
1. 미선택 정점 집합 V-U에서 거리 D가 최소인 정점 w를 선택
2. w의 인접 정점들에 대하여 w를 경유하는 거리와 기존 거리 중 작은 것을 새 거리값으로 조정




2
5
4
b
c
2
3
3
4
0
a
1
2
1
f
∞
2
3
d
e
6
1
3
4


2
5
4
b
c
2
3
3
4
0
a
1
2
1
f
10
2
3
d
e
6
1
3
4


2
5
4
b
c
2
3
3
4
0
a
1
2
1
f
8
2
3
d
e
6
1
3
4



6. 알고리즘

```
Dijkstra(G, A, s) {
/* 입력 : 그래프 G=(V,E), 인접 행렬 A, 시작정점 s, 출력 : s로부터 다른 모든 정점까지의 최단 경로의 길이 */
U = {s};
for (모든 정점 v)
D[v] = A[s][v];
while (U!= V) {
D[w]가 최소인 정점 w ∈ V-U를 선택;
U=U ∪ {w};
for (w에 인접한 모든 정점 v) {
D[v] = min(D[v], D[w] + A[w][v]);
}
}
}
```

학습내용2 : 최단 경로

1. 모든 쌍 최단 경로

모든 정점 쌍 간의 최단경로를 구하는 문제방법으로 경로의 길이가 음인 사이클이 그래프에 존재하지 않을 것을 가정하고, 단일 출발점으로 하여 반복적으로 적용해서 구할수도 있다. O(|V|s)

- 경로의 길이가 음인 사이클이 그래프에 존재하지 않는다.
- 플로이드 알고리즘을 적용.

2. 플로이드 알고리즘

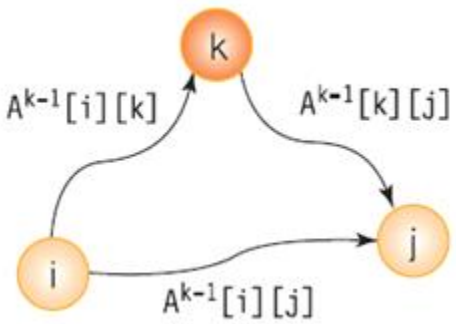
- 최단 경로 계산에 |V| X |V| 행렬 D=(dij)가 사용된다.
- 정점 i에서 j까지의 최단 경로의 길이 dij를 구함에 있어 동적 프로그래밍 방식을 적용.

◎ 모든 정점 사이의 최단경로를 찾음
◎ 2차원 배열 A를 이용하여 3중 반복을 하는 루프로 구성
◎ 인접 행렬 weight 구성
  ☞ i==j이면, weight[i][j]=0
  ☞ 두개의 정점 i,j 사이에 간선이 존재하지 않으면, weight[i][j]=∞
  ☞ 정점 i,j 사이에 간선이 존재하면, weight[i][j]는 간선 (i, j)의 가중치
  ☞ 배열 A의 초기 값은 인접 행렬 weight임

```
floyd(G)

for k ← 0 to n - 1
    for i ← 0 to  n - 1
       for j ← 0 to n - 1
           A[i][j] = min(A[i][j], A[i][k] + A[k][j])
```

◎ A^k [i][j]
  ☞ 0부터 k까지의 정점만을 이용한 정점 i에서 j까지의 최단 경로 길이
◎ A^{-1}→A^0 → A^1 → … → A^{n-1}순으로 최단 경로 구해감
◎ A^{k-1}까지 구해진 상태에서 k번째 정점이 추가로 고려되는 상황을 생각하자

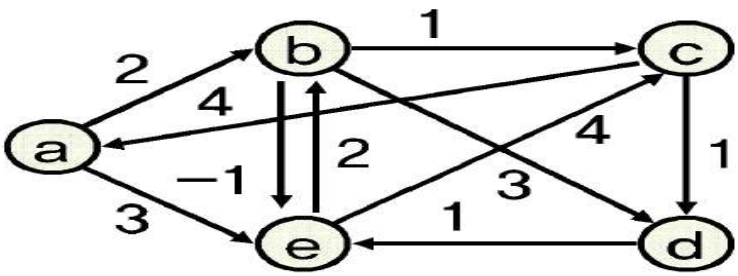


◎ 0부터 k까지의 정점만을 사용하여 정점 i에서 정점 j로 가는 최단 경로는 다음의 2가지의 경우로 나누어진다.
◎ 정점 k를 거치지 않는 경우:
  ☞ A^k[i][j] 는 k보다 큰 정점은 통과하지 않으므로 최단거리는 여전히 A^{k-1}[i][j]]임
◎ 정점 k를 거치는 경우:
  ☞ i에서 k까지의 최단거리 A^{k-1}[i][k]에 k에서 j까지의 최단거리 A^{k-1}[k][j]를 더한 값

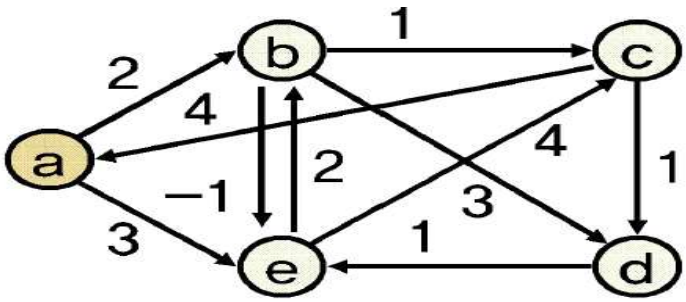
3. 알고리즘

```
Floyd(A, n) {
/* 입력: 인접행렬 A[1..n][1..n], n=|V|, 출력: 모든 정점 쌍 간의 거리 행렬 D */
int i, j, k;
for (i=1; i<=n; i++)
for (j=1; j<=n; j++)
D[i][j]=A[i][j];
for (k=1; k<=n; k++)
for(i=1; i<=n; i++)
for(j=1; j<=n; j++)
if(D[i][j] > D[i][j]+D[k][j])
D[i][j] = D[i][k]+D[k][j];
}
```

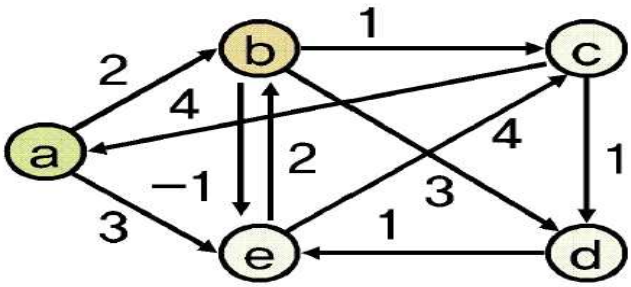
4. 플로이드 알고리즘 적용 예



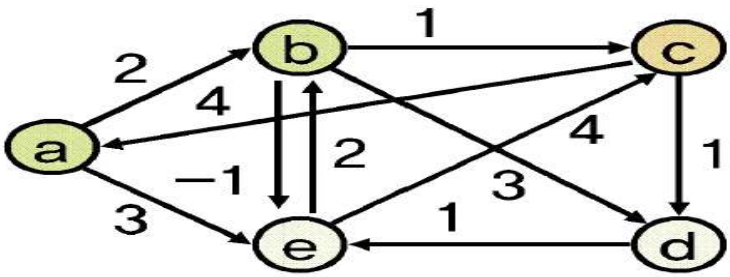
$$D^{(0)} = \begin{bmatrix} 0 & 2 & \infty & \infty & 3 \\ \infty & 0 & 1 & 3 & -1 \\ 4 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & 0 & 1 \\ \infty & 2 & 4 & \infty & 0 \end{bmatrix}$$



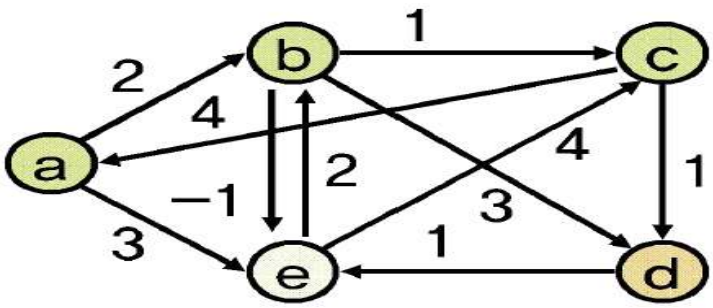
$$D^{(1)} = \begin{bmatrix} 0 & 2 & \infty & \infty & 3 \\ \infty & 0 & 1 & 3 & -1 \\ 4 & 6 & 0 & 1 & 7 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & 2 & 4 & \infty & 0 \end{bmatrix}$$



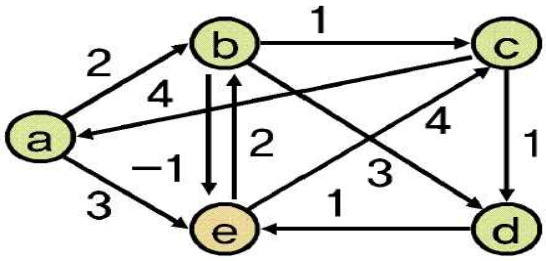
$$D^{(2)} = \begin{bmatrix} 0 & 2 & 3 & 5 & 1 \\ \infty & 0 & 1 & 3 & -1 \\ 4 & 6 & 0 & 1 & 5 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & 2 & 3 & 5 & 0 \end{bmatrix}$$



$$D^{(3)} = \begin{bmatrix} 0 & 2 & 3 & \underline{4} & 1 \\ \underline{5} & 0 & 1 & \underline{2} & -1 \\ 4 & 6 & 0 & 1 & \underline{5} \\ \infty & \infty & \infty & 0 & 1 \\ \underline{7} & 2 & 3 & \underline{4} & 0 \end{bmatrix}$$



$$D^{(4)} = \begin{bmatrix} 0 & 2 & 3 & 4 & 1 \\ 5 & 0 & 1 & 2 & -1 \\ 4 & 6 & 0 & 1 & 2 \\ \infty & \infty & \infty & 0 & 1 \\ 7 & 2 & 3 & 4 & 0 \end{bmatrix}$$



$$D^{(5)} = \begin{bmatrix} 0 & 2 & 3 & 4 & 1 \\ 5 & 0 & 1 & 2 & -1 \\ 4 & \underline{4} & 0 & 1 & 2 \\ \underline{8} & \underline{3} & \underline{4} & 0 & 1 \\ 7 & 2 & 3 & 4 & 0 \end{bmatrix}$$

최종 결과

5. Floyd의 최단경로 알고리즘 복잡도

◎ 네트워크에 n개의 정점이 있다면, Floyd의 최단경로 알고리즘은 3중 반복문을 실행되므로 시간 복잡도는 $O(n^3)$ 이 된다

◎ 모든 정점상의 최단경로를 구하려면 Dijkstra의 알고리즘 $O(n^2)$을 n번 반복해도 되며,
이 경우 전체 복잡도는 $O(n^3)$ 이 된다.

◎ 모든 정점 쌍의 최단경로를 구하는데 있어 두 알고리즘 모두 동일한 $O(n^3)$의 복잡도를 가지 지만 Floyd의 알고리즘은 매우 간결한 반복구문을 사용하므로 Dijkstra의 알고리즘 보다 효율적이다

【학습정리】

1. 단일 출발점 최단 경로
- 음의 가중치를 갖는 간선이 없는 가중 그래프에서 한 출발
- 정점x에서 다른 모든 정점까지 가중치 합이 최소인 경로를 찾는 문제

- 다이스트라 알고리즘
→ 욕심쟁이 방법 적용
→ 출발점에서 시작하여 거리가 최소인 정점을 선택해 나감으로 최단 경로를 구하는 방법
→ 정점 v의 거리 D[v]는 시작 정점 s로부터 현재까지 선택된 정점 집합 U를 경유하여 정점 v에 이르는 최소 경로의 길이를 의미
→ 적용 방법: ⓐ 미선택 정점 집합 V-U에서 거리 D가 최소인 정점 w를 선택ⓑ w의 인접 정점들에 대하여 w를 경유하는 거리와 기존 거리 중에서 작은 것을 새 거리값으로 조정
→ 시간 복잡도: 인접행렬- $O(|V|^2)$, 인접 리스트 - $O((|E|+|V|)lg|V|)$

2. 모든 쌍 최단 경로
◎ 네트워크에서 정점 u와 정점 v를 연결하는 경로 중에서 간선들의 가중치 합이 최소가 되는 경로
(경로의 길이가 음인 사이클이 그래프에 존재하지 않는 것을 가정)
→ 단일 출발점 최단 경로를 구하는 다익스트라 알고리즘을 각 정점을 출발점으로 하여 반복적으로 적용해서 구할 수도 있다 → $O(|V|^3)$

* 플로이드 알고리즘
- 최단 경로 계산에 |V| X |V| 행렬 D=(dij)가 사용된다.
- 정점 i에서 j까지의 최단 경로의 길이 dij를 구함에 있어 동적 프로그래밍 방식을 적용.

* Floyd의 최단경로 알고리즘 복잡도
- 네트워크에 n개의 정점이 있다면, Floyd의 최단경로 알고리즘은 3중 반복문을 실행되므로 시간 복잡도는 $O(n^3)$ 이 된다
- 모든 정점상의 최단경로를 구하려면 Dijkstra의 알고리즘 $O(n^2)$을 n번 반복해도 되며, 이 경우 전체 복잡도는 $O(n^3)$ 이 된다.
- 모든 정점 쌍의 최단경로를 구하는데 있어 두 알고리즘 모두 동일한 $O(n^3)$의 복잡도를 가지지만 Floyd의 알고리즘은 매우 간결한 반복구문을 사용하므로 Dijkstra의 알고리즘 보다 효율적이다