Shortest Path Algorithm
(최단경로 알고리즘)

㈜한컴에듀케이션 이주현

1. Shortest Path Algorithm

- ❖ 전제 조건
  - ➢ 가중치가 있는 방향 그래프(weighted directed graph)
  - ➢ 무향그래프는 양쪽 방향 간선이 있는 그래프로 생각한다.
- ❖ 두 정점 사이의 최단 경로
  - ➢ 두 정점 사이의 경로들 중에 간선의 가중치 합이 최소인 경로
  - ➢ 간선 가중치의 합이 음수인 싸이클이 있다면 문제가 정의 될 수 없다.
- ❖ 참조
  - ✓ 쉽게 배우는 알고리즘 – 문병로
  - ✓ wiki

❖ 모든 쌍 최단 경로

- ➢ 모든 정점 쌍 사이의 최단 경로를 구한다.
- ➢ Floyd-Warshall Algorithm (플로이드 와샬 알고리즘)

❖ 단일 시작점 최단 경로

- ➢ 하나의 시작점으로부터 각 정점에 이르는 최단 경로를 구한다.
- ➢ Dijkstra Algorithm(다익스트라 알고리즘)
  - ✓ 음의 가중치를 포함하지 않는 최단 경로 알고리즘
- ➢ Bellman-Ford Algorithm(벨만-포드 알고리즘)
  - ✓ 음의 가중치를 허용하는 단일 시작점 최단경로 알고리즘이다.
  - ✓ 음의 가중치를 허용하지만 음의 싸이클은 허용되지 않는다.

2. Floyd-Warshall Algorithm

- All pairs Shortest Path
  - 모든 정점 간에 상호 최단거리 구하기.
  - 동적 프로그래밍을 이용한 방법
- 응용 예
  - Road Atlas
  - Navigation System
  - Network Communication

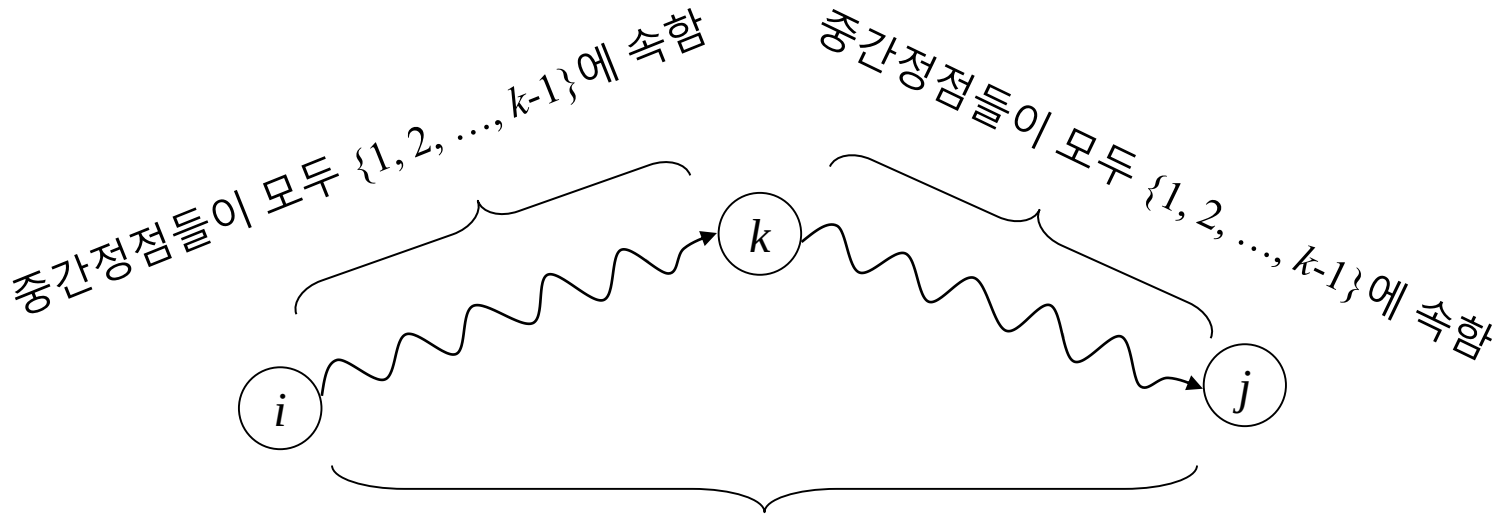
Floyd-Warshall Algorithm – cont.

```
FloydWarshall(G, V){
    for i = 1 to |V|:
        for j = 1 to |V|:
            d_ij^0 = w_ij
    for k = 1 to |V|:                    // 경유지 정점
        for i = 1 to |V|:                // 출발지 정점
            for j =1 to |V|:             // 도착지 정점
                d_ij^k = min(d_ik^(k-1) + d_kj^(k-1))
}
```

- d_{ij}^{k} : 경유지 정점으로 1, 2, …, k 들 만을 사용하여 정점 i 로부터 정점 j 까지에 이르는 최단경로
- 시간복잡도 : $O(|V|^3)$

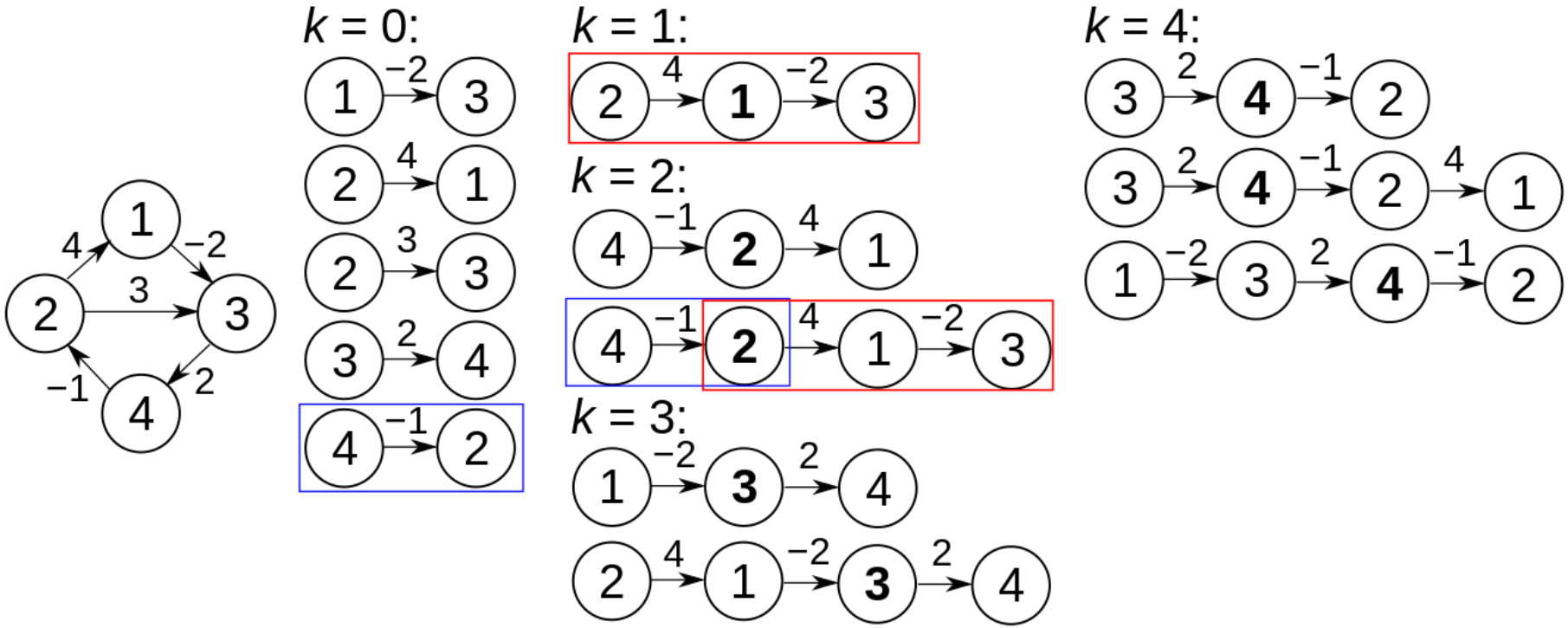
Floyd-Warshall Algorithm – cont.

d^k_{ij} 를 그림으로…



Floyd-Warshall Algorithm – cont.

Example



Floyd-Warshall Algorithm – sample code

```
int adjMat [LM][LM];                        // 인접 행렬(비용, 거리 등등)
int path[LM][LM];                           // 경로 저장 배열
void FloydWarshall(){
    for (int k=1; k<=N;k++){                // k : 중간 정점
        for (int i=1; i<=N; i++){           // i : 출발 정점
            for (int j=1; j<=N; j++){       // j : 도착 정점
                if(adjMat[i][j] > adjMat[i][k] + adjMat[k][j]){
                    adjMat[i][j] = adjMat[i][k] + adjMat[k][j];
                    path[i][j] = k;
                }
            }
        }
    }
}
```

3. Dijkstra Algorithm

- 단일 시작점 최단 경로
  - 하나의 시작점으로부터 각 정점에 이르는 최단 경로를 구한다.
- 간선에 음의 가중치를 사용할 수 없다.
- 데이크스트라 알고리즘은 원래 두 정점 사이의 최단 경로를 구하는 알고리즘이지만 일반화된 버전은 한 점을 시작점으로 다른 모든 정점까지의 최단 경로를 구하여 최단 경로 트리를 만들 수 있다.
- 참조
  - Wikipedia : 데이크스트라 알고리즘
  - A note on two problems in connexion with graphs

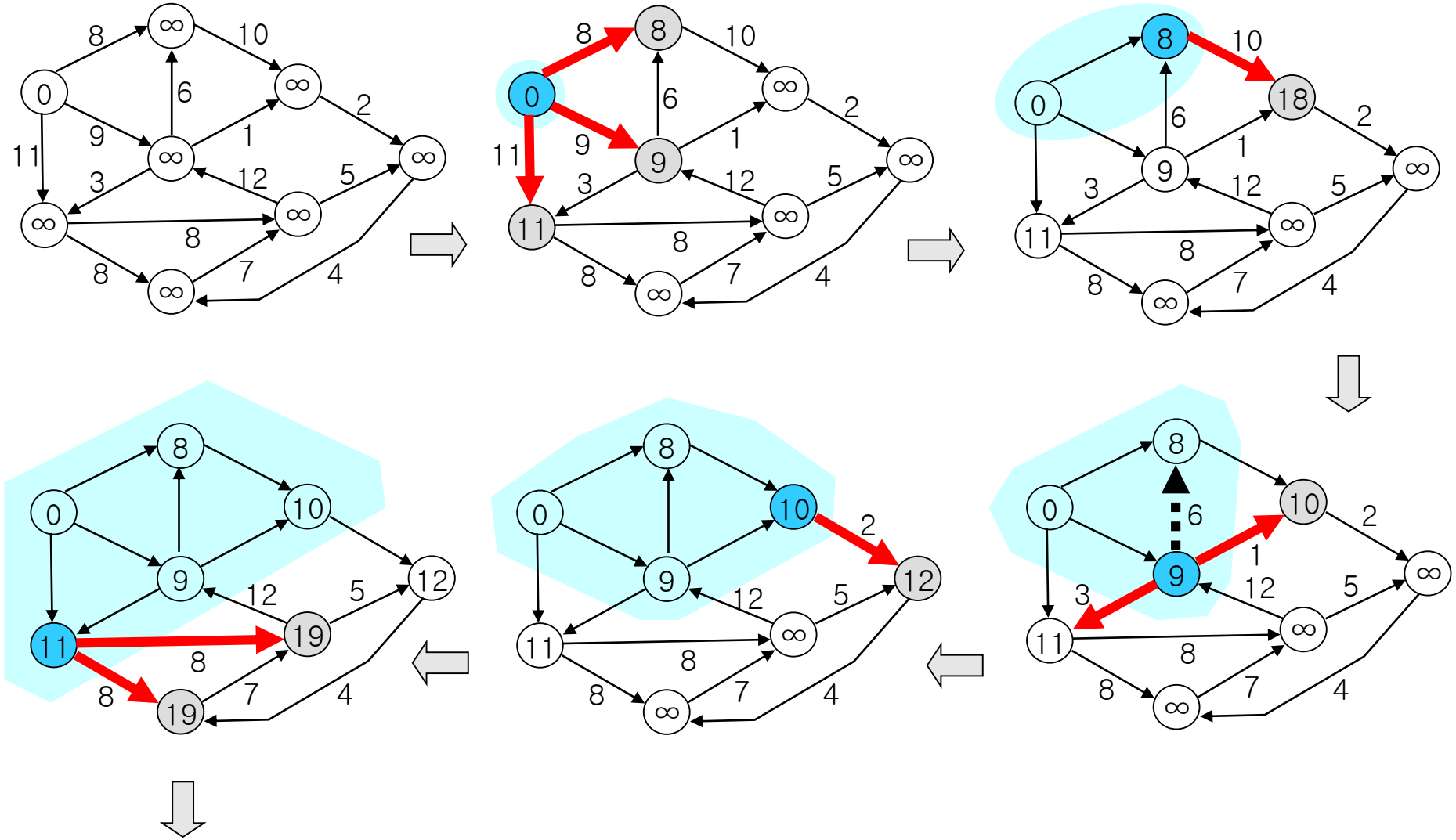
Dijkstra Algorithm pseudocode

```
Dijkstra(G, V, source){
    S = ∅                                   // 최단 경로 정점 집합 : 초기에는 공집합
    for i = 1 to |V|:                       // 시작점으로부터의 거리 초기값을 무한대로
        d_i = ∞
    d_source = 0                            // 시작점의 거리 초기 값은 0
    while S ≠ V:
        u = extract_min(V − S, d_u)         // 남아있는 노드 중 최소 거리를 갖는 정점 찾기
        S = S ∪ {u}                         // 찾은 정점을 해집합에 포함시키기
        for each adjacent vertext v of u:
            if  d_v > d_u + w_uv:           // 찾은 정점으로부터 인접한 정점의 거리 업데이트
                d_v = d_u + w_uv            // relaxation(이완)
}
```

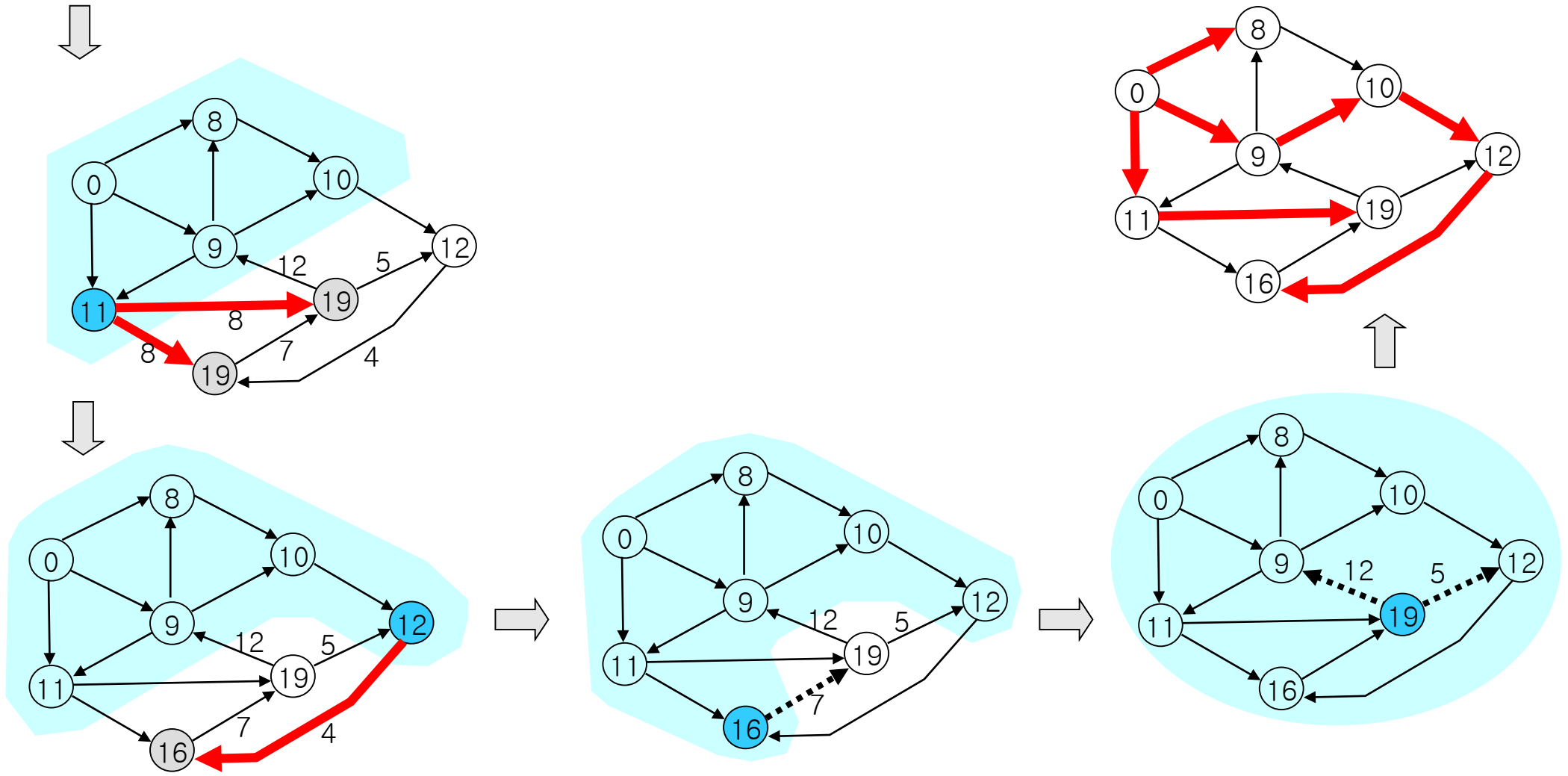
❖ 시간복잡도 : $O(|V|^2)$, 우선순위큐(힙)를 이용하면 $O(|E|log|V|)$

Dijkstra Algorithm Example

Example



Dijkstra Algorithm Example – cont.



Dijkstra Algorithm – sample code : $O(|V|^2)$

```
int adjMat[LM][LM];                              // 인접 행렬(비용, 거리 등등)
int dist[LM];                                    // 출발지로부터 최단거리를 저장할 배열
int path[LM], visited[LM];                         // 경로, 해집합(방문 체크)

void  Dijkstra(int source){
        for (int i=1; i<=n; i++) dist[i] = INF; // initialize dist[]
        dist[source] = 0;                        // initialize dist[source]
        for (int i=1; i<=n; i++){
                int minNode, minVal =INF;
                for (int j=1; j<=n; j++){        // extract min node & value
                        if(visited[j]==0 && minVal > dist[j]){
                                minNode = j, minVal = dist[j];
                        }
                }
                visited [minNode] = 1;           // check min node
                for (int j=1; j<=n; j++){        // relaxation
                        if( dist[j] > minVal + adjMat[minNode][j]){
                                dist[j] = minVal + adjMat[minNode][j];
                                path[j] = minNode;
                        }
                }
        }
}
```

Dijkstra Algorithm – sample code : $O(|E|log|V|)$

```cpp
const int SIZE = (int)1e5 + 5;
const int INF = (int)21e8;
struct Data{
    int u, d; // u: 정점, d: u까지 거리
    bool operator<(const Data&r)const{
        return d > r.d; // 부등호에 주의
    }
};

int N;
int dist[SIZE];
int path[SIZE];
int visited[SIZE];
vector<Data> adj[SIZE];  // 인접배열
priority_queue<Data> pq;
```

```cpp
void  Dijkstra(int source){
    // init dist[]
    for (int i=1; i<=N; i++) dist[i] = INF;
    dist[source] = 0; // init dist[source]
    pq.push({source, 0});
    while(!pq.empty()){
        Data t = pq.top();    // extract min
        pq.pop();
        if(visited[t.u]) continue;
        visited[t.u] = true;  // check min node
        for(auto&a:adj[t.u]){ // relaxation
            if(dist[a.u] > t.d + a.d){
                dist[a.u] = t.d + a.d;
                path[a.u] = t.u;
                pq.push({a.u, dist[a.u]});
            }
        }
    }
}
```

4. Bellman-Ford Algorithm

- 단일 시작점 최단 경로
  - 하나의 시작점으로부터 각 정점에 이르는 최단 경로를 구한다.
- 간선에 음의 가중치를 허용하지만 음의 싸이클을 허용하지 않는다.
- 벨만-포드 알고리즘은 다음 작업을 $|V| - 1$ 회 반복한다.
  - 각 정점에 대한 시작점에서의 거리를 모든 간선을 이용하여 이완(relaxation)시킨다.
  - $|V|$ 번째에도 이완이 일어난다면 음의 싸이클이 존재하는 것이다.
- 참조
  - wiki
  - geeksforgeeks

Bellman-Ford Algorithm pseudocode

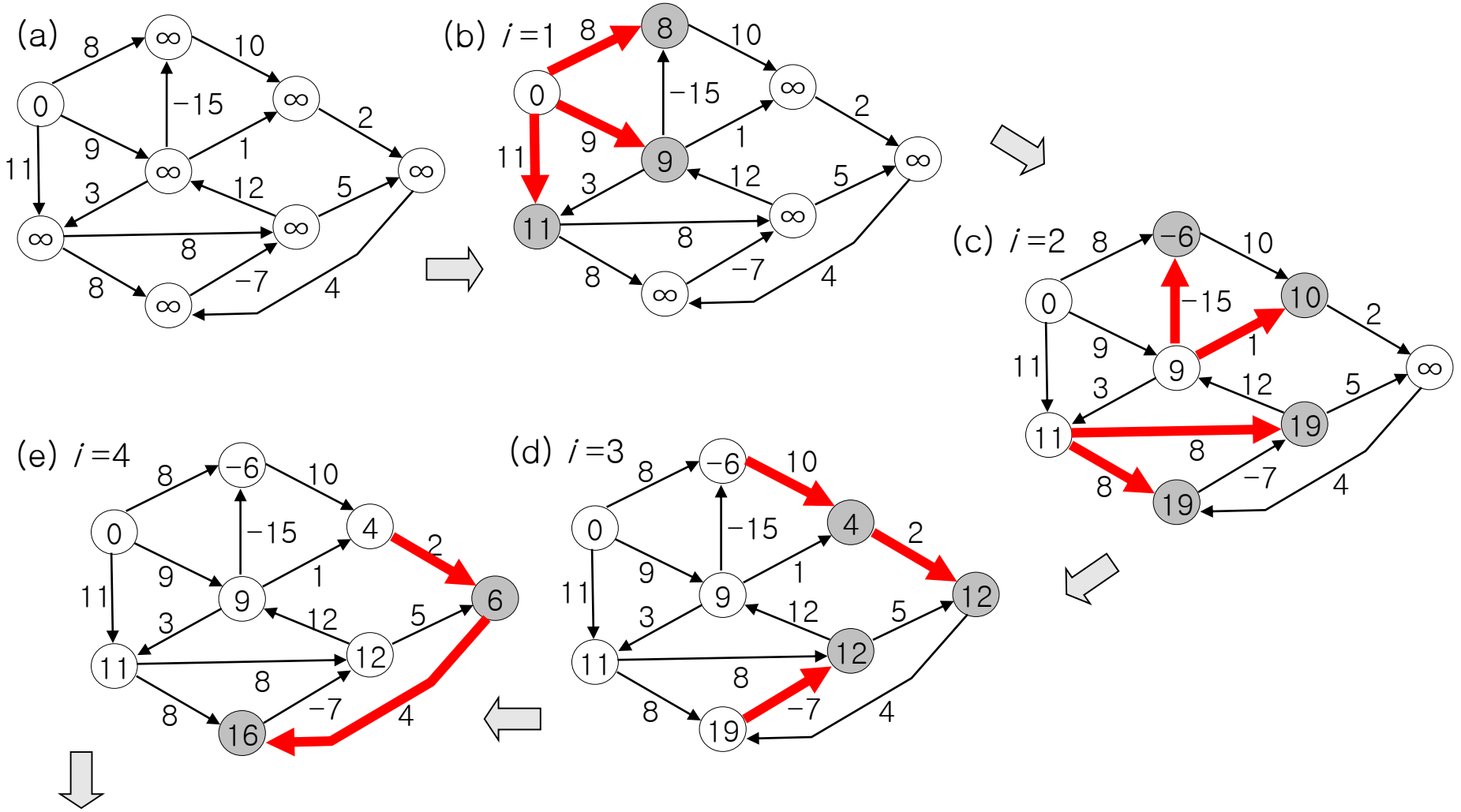
```
BellmanFord(G, V, source){
    for i = 1 to |V|:                    // 시작점으로부터의 거리 초기값을 무한대로
        d_i = ∞
    d_source = 0                         // 시작점의 거리 초기 값은 0

    flag = false                         // 싸이클 유무를 판단할 변수
    for i = 1 to |V|:
        flag = false
        for each (u, v):
            if d_v^i > d_u^(i-1) + w_uv:     // 찾은 정점으로부터 인접한 정점의 거리 업데이트
                d_v^i = d_u^(i-1) + w_uv     // relaxation(이완)
                flag = true                  // i == |V| 에서 이완이 일어나면 음수 싸이클이 존재함
    return flag
}
```

❖ d_v^i : 최대 i 개의 간선을 거쳐 시작점 $source$ 로부터 v 까지의 최단 경로(거리)

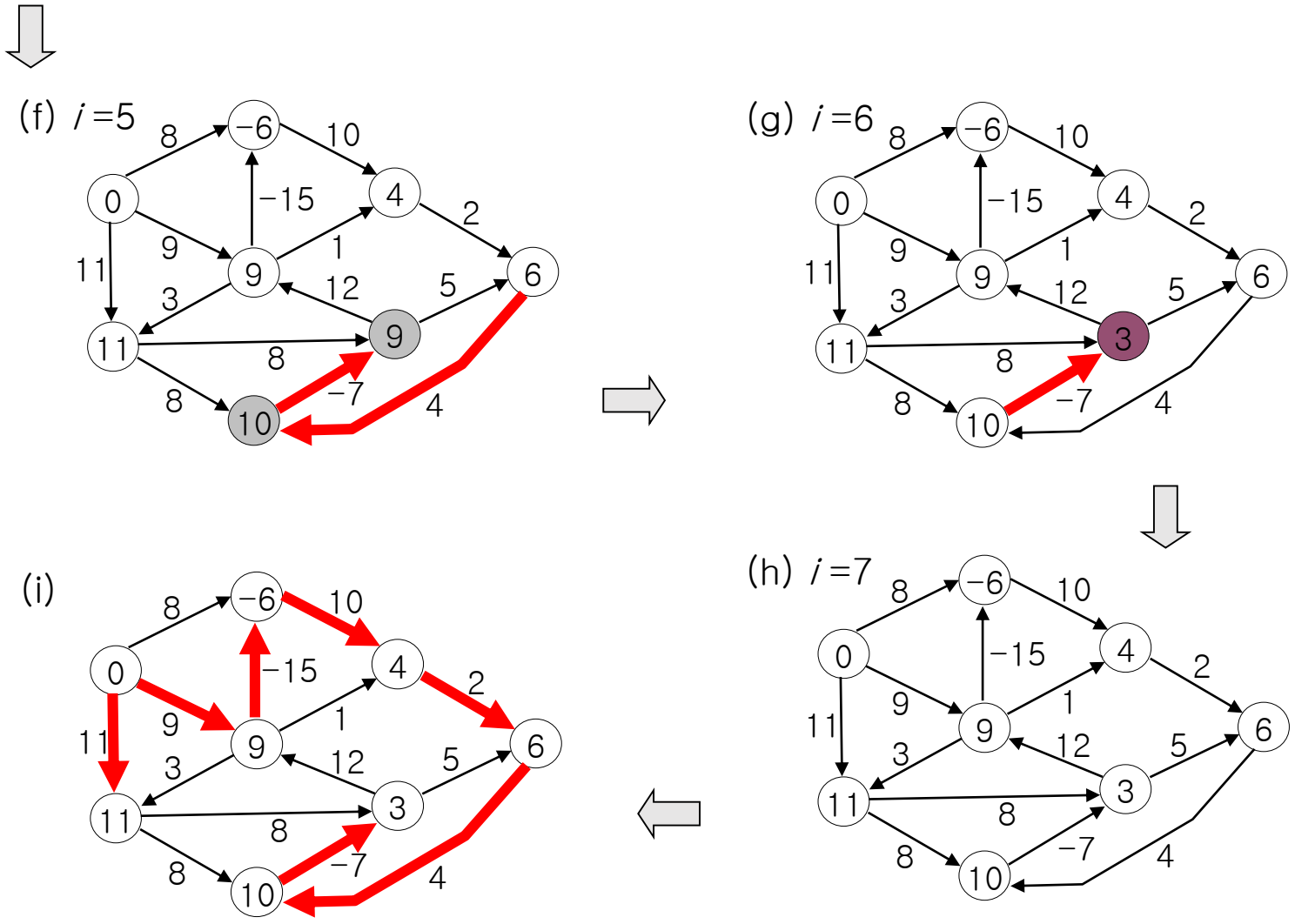
❖ 시간복잡도 : $O(|E| * |V|)$

Bellman-Ford Algorithm Example

Example



Bellman-Ford Algorithm Example – cont.



Bellman-Ford Algorithm – sample code : $O(|E| * |V|)$

```
struct Data{
    int s, e, w;
}edges[LM];                                     // 간선정보 목록
int dist[LM];                                   // 출발지로부터 최단거리를 저장할 배열
int N, en;                                      // N: 정점수, E: 간선수

bool BellmanFord(int source){
    int i, j;
    bool negCycle = false;
    for(i=1;i<=N;++i) dist[i] = INF;            // 거리배열 초기화
    dist[source] = 0;
    for(i=0;i<N;++i){
        negCycle = false;                       // 음의 싸이클을 판단할 변수
        for(j=0;j<en;++j){
            int s = edges[j].s, e = edges[j].e;
            int w = edges[j].w;
            if(dist[e] > dist[s] + w){
                dist[e] = dist[s] + w;          // 이완(relaxation)
                negCycle = true;                // i==N 에서 이완이 일어나면 음의 싸이클이 존재
            }
        }
    }
    return negCycle;
}
```

5. SPFA(Shortest Path Faster Algorithm)

- 단일 시작점 최단 경로
  - 하나의 시작점으로부터 각 정점에 이르는 최단 경로를 구한다.
- 벨만-포드 알고리즘을 개선한 알고리즘이다.
  - 모든 간선 정보를 검토하는 것이 아닌
    거리를 이완 시킬 가능성이 있는 정점들을 큐에 담고
  - 큐에 담긴 정점과 연결된 간선 정보만을 검토한다.
- 시간복잡도
  - 랜덤데이터에 대하여 $O(|E|)$의 시간복잡도를 갖지만
  - 최악의 경우 $O(|E| * |V|)$의 시간복잡도를 갖는다.
    때문에 음의 가중치를 포함하지 않는 경우 다익스트라 알고리즘이 더 선호된다.

SPFA Algorithm pseudocode

```
spfa(G, source){
    for i = 1 to |V|:                    // 시작점으로부터의 거리 초기값을 무한대로
        d_i = ∞
    d_source = 0                         // 시작점의 거리 초기 값은 0

    offer source into Q                  // 시작노드를 큐에 담는다.
    while Q is not empty:
        u := poll Q                      // 큐에서 점점을 꺼내어 u에 저장한다.
        for each (u, v):
            if d_v^i > d_u^(i-1) + w_uv:     // 찾은 정점으로부터 인접한 정점의 거리 업데이트
                d_v^i = d_u^(i-1) + w_uv     // relaxation(이완)
                if v is not Q:               // 큐에 v가 들어있지 않다면
                    offer v into Q           // 큐에 v를 담는다.
}
```

❖ d_v^i : 최대 i 개의 간선을 거쳐 시작점 $source$ 로부터 v 까지의 최단 경로(거리)

❖ 시간복잡도 : $O(|E| * |V|)$

SPFA Algorithm – sample code : $O(|E| * |V|)$

```
struct Edge{ int v, w;};
vector<Edge> adj[LM];                                   // 간선정보 목록 인접리스트
bool isinQ[LM];                                         // 정점이 큐에 포함되어 있는가를 저장할 배열
int usedEdgeCnt[LM];                                    // 각 점점별 dist[]를 이완하는데 사용된 간선 수 저장
int dist[LM];                                           // 출발지로부터 최단거리를 저장할 배열
int N;                                                  // N: 정점수
queue <int> que;                                        // dist[]를 이완시킬 가능성이 있는 정점들의 목록

bool SPFA (int source){
    for (int i=1; i<=N; i++) dist[i] = INF;             // 거리배열 초기화
    dist[source] = 0;
    que.push(source);
    while (!que.empty()){
        int u = que.front();
        que.pop();
        isinQ[u] = false;
        for(auto&d:adj[u]){
            if(dist[d.v] > dist[u] + d.w){
                dist[d.v] = dist[u] + d.w;
                usedEdgeCnt[d.v] = usedEdgeCnt[u] + 1;  // 정점 dist[d.v]에 사용된 간선 수 저장
                if(usedEdgeCnt[d.v] >= N) return true;  // 정점 d.v의 이완횟수가 N번 이상 발생한 경우 싸이클이 존재함.
                if(!isinQ[d.v]){                        // 큐에 d.v가 포함되어 있지 않다면
                    que.push(d.v);
                    isinQ[d.v] = true;
                }
            }
        }
    }
    return false;
}
```

감사합니다.^^