그래프 최단경로

김영균

최단경로의 종류

- single source shortest path

  한 개의 시작점(source)에서 다른 모든 점까지의 최단경로

  - unweighted graph : bfs
  - weighted graph
    1) 음의 가중치가 없을 때: dijkstra
    2) 음의 가중치가 있을 때: bellman-ford, spfa
  - DAG: topological sort

- all pairs shortest path

  모든 정점 쌍끼리 최단경로

  - floyd-warshall

정의

- **경로의 길이**

  경로가 지나는 간선의 가중치의 합

- **정점 u에서 정점 v로 가는 최단 경로**

  u에서 v로 가는 경로의 길이가 최소인 경로

- **간선의 완화 (Edge Relaxation)**

  정점 u에서 v로의 간선에 대해 $dist[v] > dist[u] + weight(u,v)$ 만족하면 $dist[v]$를 $dist[u] + weight(u,v)$로 업데이트 시켜주는 과정

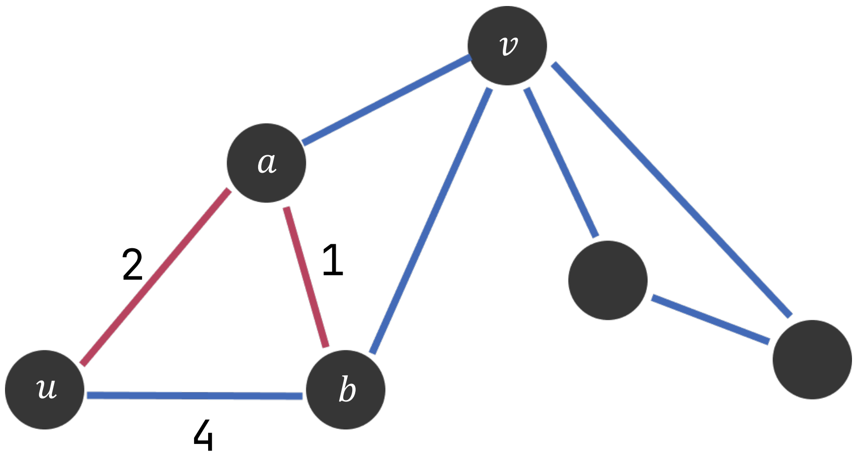


그림1. 최단경로

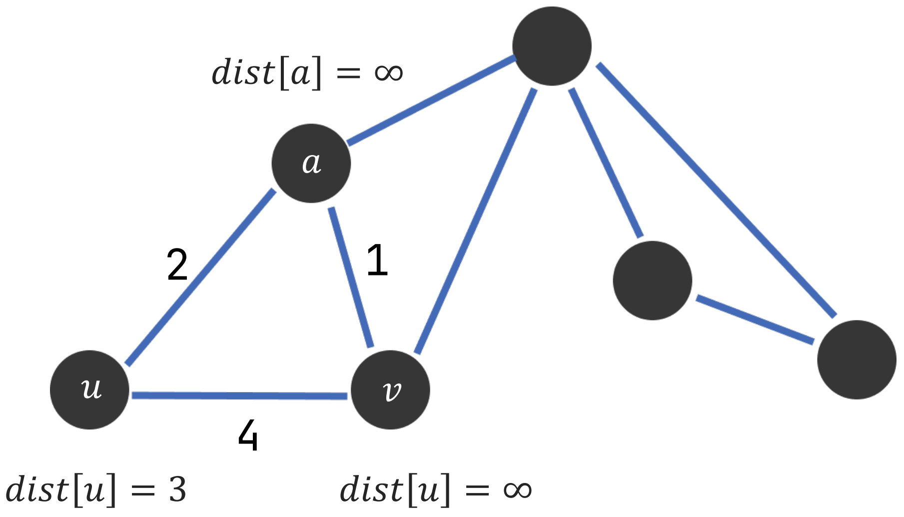


그림2. 간선의 완화

dijkstra

- bfs의 특징

정점 u에서 v까지 최단경로가 존재할 때 정점 u에서 최단경로에 속한 정점들까지 거리도 최단경로이다.

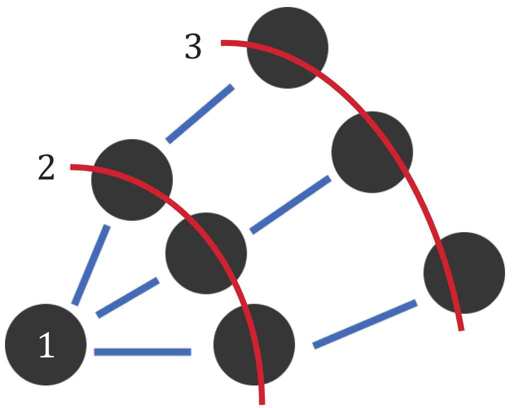


그림1. bfs 전파

- bfs

1. 큐에 시작점 start를 넣는다.
2. 큐에서 정점 한 개(here)을 뽑는다.
3. here과 인접한 모든 정점(there)에 대해
    3-1. 큐에 there이 들어간적이 없다면
    3-2. dist[there] = dist[here] + 1
    3-3. 큐에 there을 넣는다.
4. 큐에 원소가 있다면 2번 과정을 반복한다.

```
memset(dist, -1, sizeof(dist));
dist[start] = 0;
queue<int> q;
q.push(start);
while(!q.empty()) {
    int here = q.front();
    q.pop();
    for(int there: adj[here]) {
        if(dist[there] != -1) {
            dist[there] = dist[here] + 1;
            q.push(there);
        }
    }
}
```

dijkstra

- disjkstra

1. 큐에 시작점 start를 넣는다.
2. 큐에서 정점 한 개(here)을 뽑는다.
3. here과 인접한 모든 정점(there)에 대해
   3-1. 큐에 there이 들어간적이 없다면
   3-2. dist[there] = dist[here] + **(here과 there의 가중치)**;
   3-3. 큐에 there을 넣는다.
4. 큐에 원소가 있다면 2번 과정을 반복한다.

```
memset(dist, -1, sizeof(dist));
dist[start] = 0;
queue<int> q;
q.push(start);
while(!q.empty()) {
    int here = q.front();
    q.pop();
    for(auto [there, weight] : adj[here]) {
        if(dist[there] != -1) {
            dist[there] = dist[here] + weight;
            q.push(there);
        }
    }
}
```

dijkstra

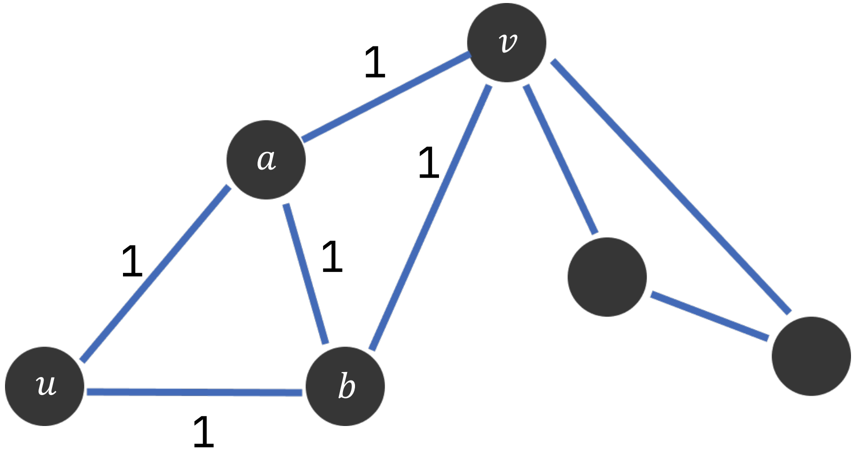
- disjkstra

  1. 큐에 시작점 start를 넣는다.
  2. 큐에서 정점 한 개(here)을 뽑는다.
  3. here과 인접한 모든 정점(there)에 대해

     3-1. 큐에 there이 들어간적이 없다면

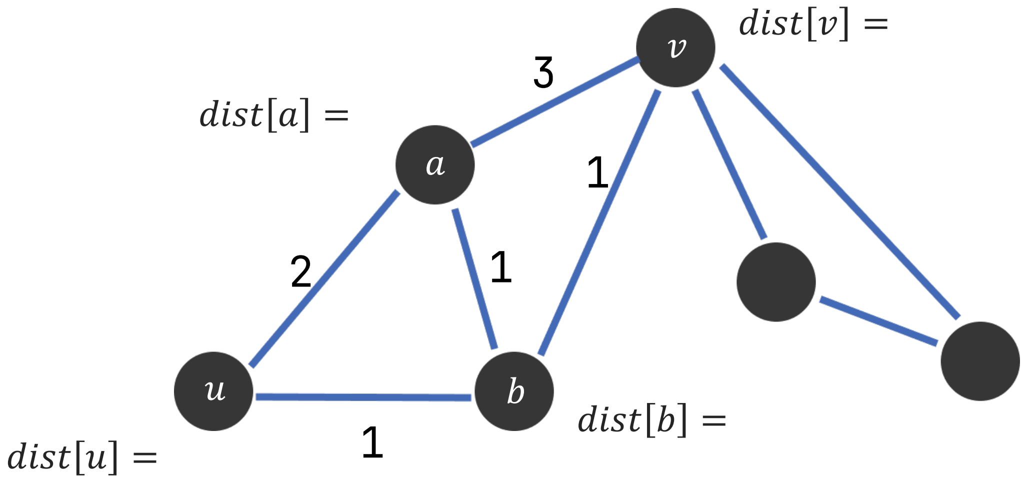
     3-2. dist[there] = dist[here] + **(here과 there의 가중치);**

     3-3. 큐에 there을 넣는다.
  4. 큐에 원소가 있다면 2번 과정을 반복한다.



큐: $u \to a \to b \to v$

$dist[u] = 0, dist[a] = 1, dist[b] = 1, dist[v] = 2$



큐: $u \to a \to b \to v$

dijkstra

- disjkstra

1. **우선순위큐**에 시작점 start를 넣는다.
2. **우선순위큐**에서 정점 한 개(here)을 뽑는다.
3. here과 인접한 모든 정점(there)에 대해
   3-1. **우선순위큐**에 there이 들어간적이 없다면
   3-2. dist[there] = dist[here] + (here과 there의 가중치);
   3-3. **우선순위큐**에 there을 넣는다.
4. **우선순위큐**에 원소가 있다면 2번 과정을 반복한다.

```
memset(dist, -1, sizeof(dist));
dist[start] = 0;
priority_queue<int> q;
q.push(start);
while(!q.empty()) {
    int here = q.top();
    q.pop();
    for(auto [there, weight] : adj[here]) {
        if(dist[there] != -1) {
            dist[there] = dist[here] + weight;
            q.push(there);
        }
    }
}
```

dijkstra

- disjkstra

  1. 우선순위큐에 **(0, start)**를 넣는다.
  2. 우선순위큐에서 **(d, here)**을 뽑는다.
  3. here과 인접한 모든 정점(there)에 대해

     3-1. 우선순위큐에 there이 들어간적이 없다면

     3-2. dist[there] = dist[here] + (here과 there의 가중치);

     3-3. 우선순위큐에 **(dist[there], there)**을 넣는다.
  4. 우선순위큐에 원소가 있다면 2번 과정을 반복한다.

```
memset(dist, -1, sizeof(dist));
dist[start] = 0;
priority_queue<pair<int, int>> q;
q.push({0, start});
while(!q.empty()) {
    int d = -q.top().first;
    int here = q.top().second;
    q.pop();
    for(auto [there, weight] : adj[here]) {
        if(dist[there] != -1) {
            dist[there] = dist[here] + weight;
            q.push({-dist[there], there});
        }
    }
}
```

dijkstra

- disjkstra

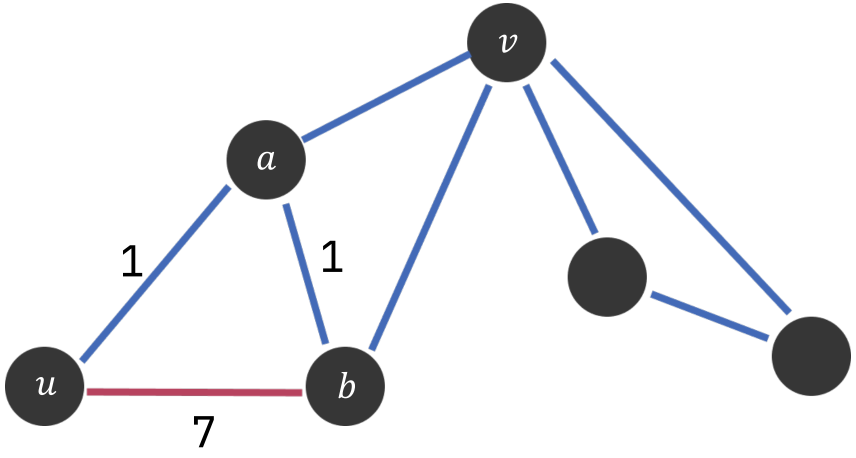
  1. 우선순위큐에 **(0, start)**를 넣는다.
  2. 우선순위큐에서 **(d, here)**을 뽑는다.
  3. here과 인접한 모든 정점(there)에 대해

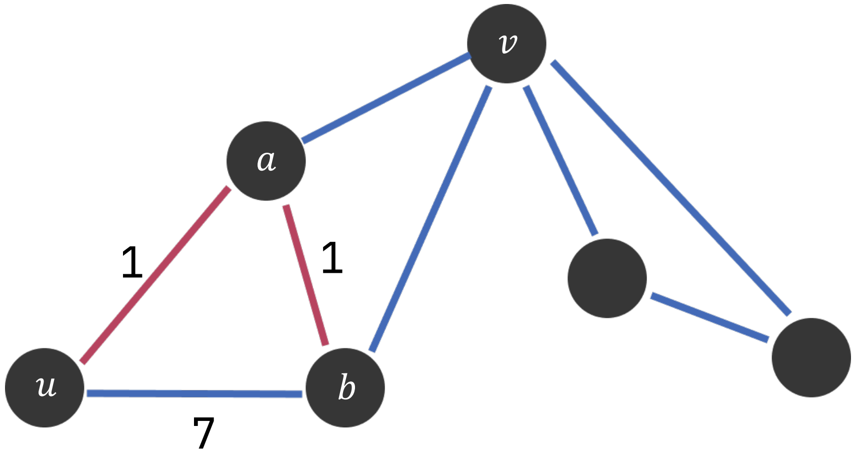
     3-1. 우선순위큐에 there이 들어간적이 없다면

     3-2. dist[there] = dist[here] + (here과 there의 가중치);

     3-3. 우선순위큐에 **(dist[there], there)**을 넣는다.
  4. 우선순위큐에 원소가 있다면 2번 과정을 반복한다.

```
memset(dist, -1, sizeof(dist));
dist[start] = 0;
priority_queue<pair<int, int>> q;
q.push({0, start});
while(!q.empty()) {
    int d = -q.top().first;
    int here = q.top().second;
    q.pop();
    for(auto [there, weight] : adj[here]) {
        if(dist[there] != -1) {
            dist[there] = dist[here] + weight;
            q.push({-dist[there], there});
        }
    }
}
```





dijkstra

- disjkstra

  1. 우선순위큐에 (0, start)를 넣는다.
  2. 우선순위큐에서 (d, here)을 뽑는다.
  3. here과 인접한 모든 정점(there)에 대해

     3-1. **dist[there] > dist[here] + (here과 there의 가중치) 라면**

     3-2. dist[there] = dist[here] + (here과 there의 가중치);

     3-3. 우선순위큐에 (dist[there], there)을 넣는다.
  4. 우선순위큐에 원소가 있다면 2번 과정을 반복한다.

```
memset(dist, -1, sizeof(dist));
dist[start] = 0;
priority_queue<pair<int, int>> q;
q.push({0, start});
while(!q.empty()) {
    int d = -q.top().first;
    int here = q.top().second;
    q.pop();
    for(auto [there, weight] : adj[here]) {
        if(dist[there] > dist[here] + weight) {
            dist[there] = dist[here] + weight;
            q.push({-dist[there], there});
        }
    }
}
```

dijkstra

- disjkstra

초기화: 시작점을 제외한 모든 정점에 대해 dist의 값은 무한히 큰 값

1. 우선순위큐에 (0, start)를 넣는다.
2. 우선순위큐에서 (d, here)을 뽑는다.
3. here과 인접한 모든 정점(there)에 대해
   3-1. dist[there] > dist[here] + (here과 there의 가중치) 라면
   3-2. dist[there] = dist[here] + (here과 there의 가중치);
   3-3. 우선순위큐에 (dist[there], there)을 넣는다.
4. 우선순위큐에 원소가 있다면 2번 과정을 반복한다.

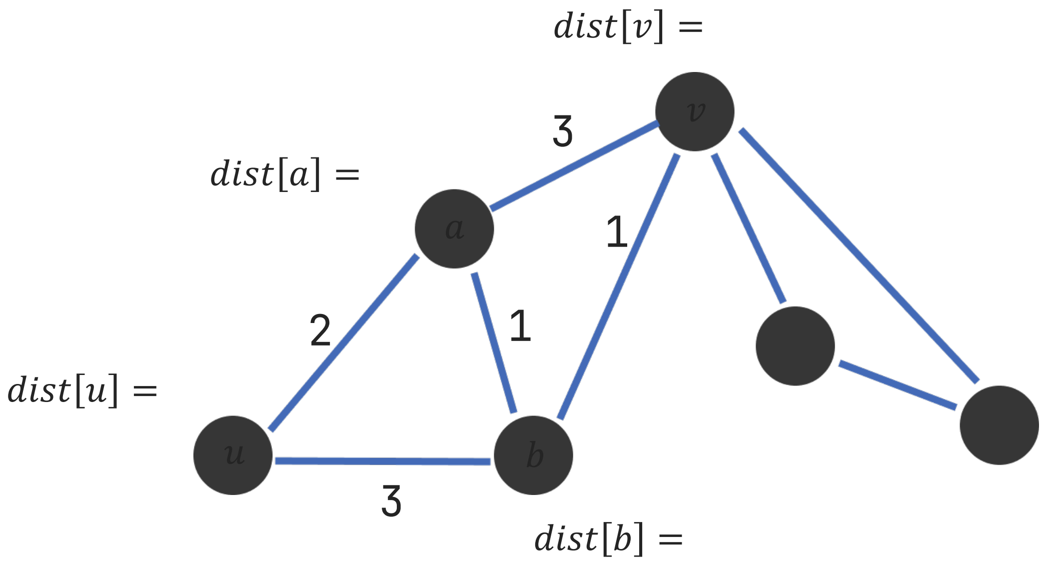
```
memset(dist, 0x3f, sizeof(dist));
dist[start] = 0;
priority_queue<pair<int, int>> q;
q.push({0, start});
while(!q.empty()) {
    int d = -q.top().first;
    int here = q.top().second;
    q.pop();
    for(auto [there, weight] : adj[here]) {
        if(dist[there] > dist[here] + weight) {
            dist[there] = dist[here] + weight;
            q.push({-dist[there], there});
        }
    }
}
```

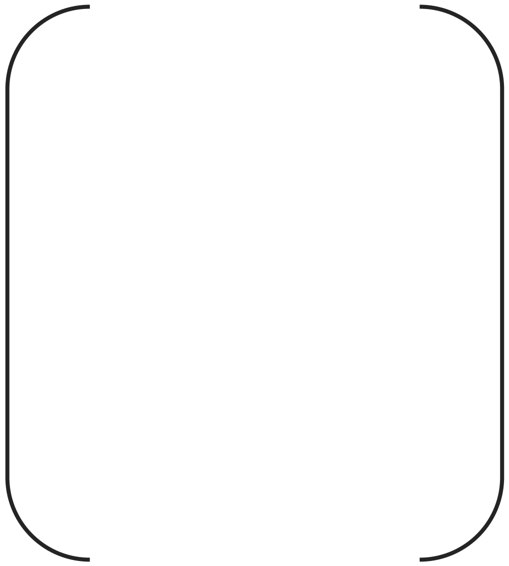
dijkstra

- disjkstra

초기화: 시작점을 제외한 모든 정점에 대해 dist의 값은 무한히 큰 값

1. 우선순위큐에 (0, start)를 넣는다.
2. 우선순위큐에서 (d, here)을 뽑는다.
3. here과 인접한 모든 정점(there)에 대해
   - 3-1. dist[there] > dist[here] + (here과 there의 가중치) 라면
   - 3-2. dist[there] = dist[here] + (here과 there의 가중치);
   - 3-3. 우선순위큐에 (dist[there], there)을 넣는다.
4. 우선순위큐에 원소가 있다면 2번 과정을 반복한다.





우선순위 큐(min-heap)

dijkstra

- disjkstra

초기화: 시작점을 제외한 모든 정점에 대해 dist의 값은 무한히 큰 값

1. 우선순위큐에 (0, start)를 넣는다.

2. 우선순위큐에서 (d, here)을 뽑는다.

3. d > dist[here] 이라면 4번 과정을 넘어간다.

4. here과 인접한 모든 정점(there)에 대해

    4-1. dist[there] > dist[here] + (here과 there의 가중치) 라면

    4-2. dist[there] = dist[here] + (here과 there의 가중치);

    4-3. 우선순위큐에 (dist[there], there)을 넣는다.

5. 우선순위큐에 원소가 있다면 2번 과정을 반복한다.

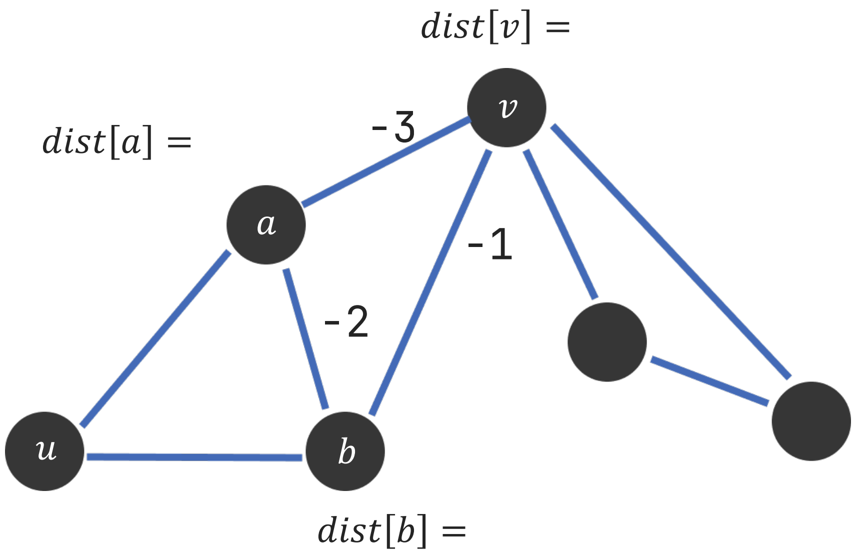
```
memset(dist, 0x3f, sizeof(dist));
dist[start] = 0;
priority_queue<pair<int, int>> q;
q.push({0, start});
while(!q.empty()) {
    int d = -q.top().first;
    int here = q.top().second;
    q.pop();
    if(dist[here] < d) continue;
    for(auto [there, weight] : adj[here]) {
        if(dist[there] > dist[here] + weight) {
            dist[there] = dist[here] + weight;
            q.push({-dist[there], there});
        }
    }
}
```

dijkstra

- 시간복잡도 $O((V+E)logV)$

```
memset(dist, 0x3f, sizeof(dist));
dist[start] = 0;
priority_queue<pair<int, int>> q;
q.push({0, start});
while(!q.empty()) {
    int d = -q.top().first;
    int here = q.top().second;
    q.pop();
    if(dist[here] < d) continue;
    for(auto [there, weight] : adj[here]) {
        if(dist[there] > dist[here] + weight) {
            dist[there] = dist[here] + weight;
            q.push({-dist[there], there});
        }
    }
}
```

- 음수 가중치가 있을 땐 시간복잡도가 보장되지 않으며 음의 사이클이 있으면 무한루프를 돌게된다.



bellman-ford

- bellman-ford

초기화: 시작점을 제외한 모든 정점에 대해 dist의 값은 무한히 큰 값

1. 주어진 모든 간선에 대해 간선 완화를 진행한다.

2. 1번을 $V-1$(정점의 개수 - 1)번 반복한다.

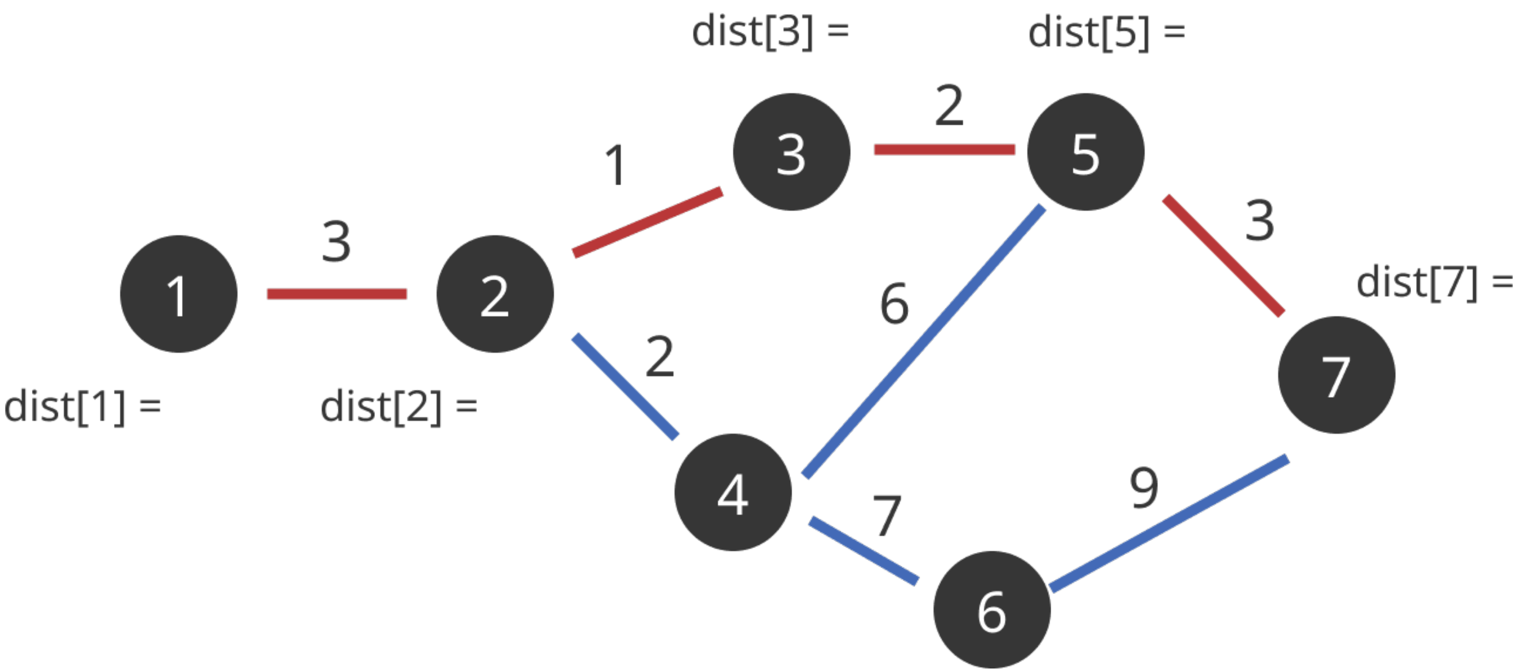
```
#define INF 0x3f3f3f3f
int from[edge_size];
int to[edge_size];
int weight[edge_size];
int dist[vertex_size];

memset(dist, 0x3f, sizeof(dist));
dist[start] = 0;

for(int i = 1; i < vertex_size; i++) {
    for(int j = 0; j < edge_size; j++) {
        if(dist[from[j]] == INF) continue;
        if(dist[to[j]] > dist[from[j]] + weight[j]) {
            dist[to[j]] > dist[from[j]] + weight[j];
        }
    }
}
```

bellman-ford

- **간선 완화를 $V-1$만큼 반복하는 이유**

정점의 개수가 V개인 그래프에서 최단 경로에 포함된 정점의 개수는 많아야 $V-1$개이다.

따라서 $V-1$만큼 간선의 완화를 반복하면 최단 경로를 구할 수 있다.



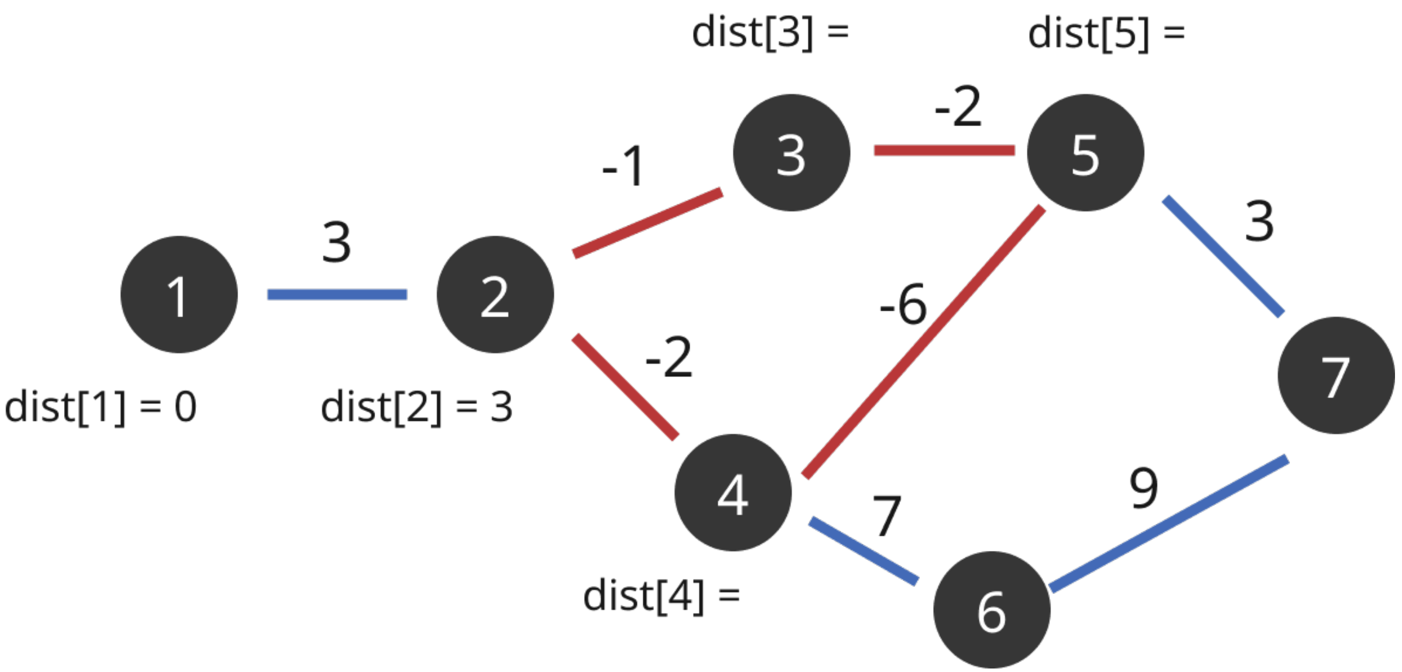
bellman-ford

- **V번 반복하면 음의 사이클을 확인할 수 있다.**

정점의 개수가 V개인 그래프에서 최단 경로에 포함된 정점의 개수는 많아야 $V-1$개이다.

따라서 $V-1$만큼 간선의 완화를 반복하면 최단 경로를 구할 수 있다.

근데 V번 반복해도 간선이 완화가 된다? → **"음의 사이클이 있다"** 라고 판단할 수 있다.



bellman-ford

- **시간복잡도 분석**

벨만포드 알고리즘은 모든 간선(E)에 대해 간선의 완화를 $V-1$번 진행하는 알고리즘 → $O(VE)$

```
#define INF 0x3f3f3f3f
int from[edge_size];
int to[edge_size];
int weight[edge_size];
int dist[vertex_size];

memset(dist, 0x3f, sizeof(dist));
dist[start] = 0;

bool cycle = false;
for(int i = 1; i <= vertex_size; i++) {
    for(int j = 0; j < edge_size; j++) {
        if(dist[from[j]] == INF) continue;
        if(dist[to[j]] > dist[from[j]] + weight[j]) {
            dist[to[j]] > dist[from[j]] + weight[j];
            if(i == vertex_size) cycle = true;
        }
    }
}
```