최단 경로 – 다익스트라 알고리즘

Shortest path

- 벨만 포드 : 각 간선을 V – 1 번 계산 / 음수 간선 있어도 사용 가능
- 다익스트라 : 각 간선을 1번 계산 / 음수 간선 있으면 사용 불가능

☞ 음수 간선 사이클 없으면 사용 가능함

- 기본 구조는 벨만 포드 알고리즘과 동일

```
if(dist[to] > dist[from] + cost){
        dist[to] = dist[from] + cost
}
```

- dist[i] = 시작점 → i로 가는 최단 거리
  visited[i] = i가 체크 되어있으면 True, 아닌 경우 False

① 체크되어 있지 않은 정점 중에서 dist의 값이 가장 작은 정점 x를 선택한다.
   minheap 우선순위 큐 사용

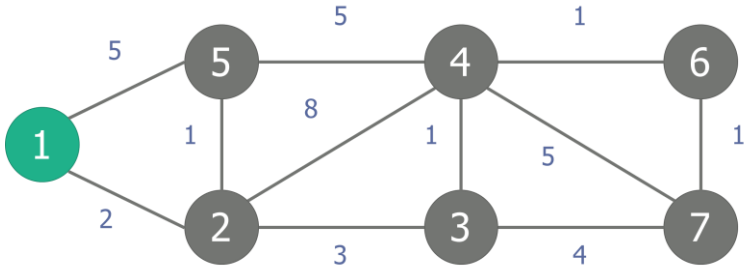
② x를 체크한다.

③ x와 연결된 모든 정점을 검사한다.
   간선의 정보를 (from, to, weight) = (u, v, w) 라고 하였을 때
   dist[v] > dist[u] + w 이면 갱신한다.

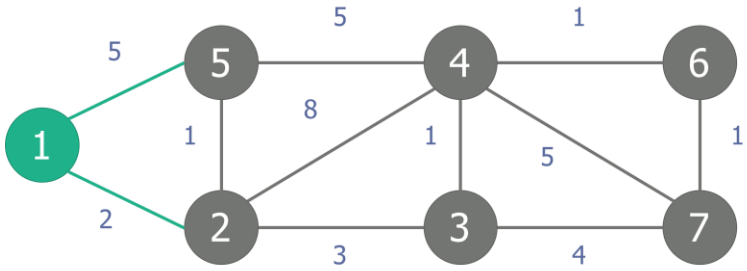
→ ①, ②, ③ 단계를 모든 정점에 대하여 실시한다.

최단 경로 – 다익스트라 알고리즘

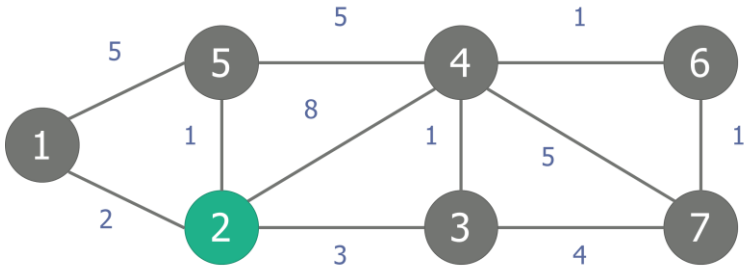
[Shortest path]



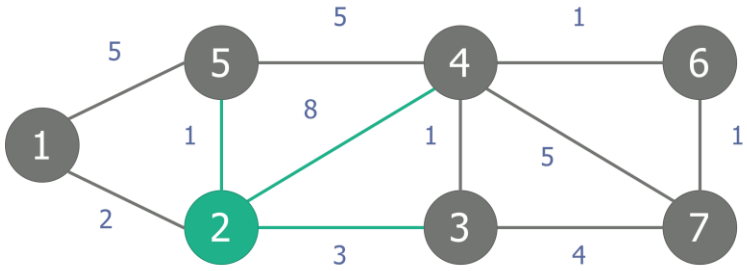
i	1	2	3	4	5	6	7
D[i]	0	∞	∞	∞	∞	∞	∞
C[i]	1						



i	1	2	3	4	5	6	7
D[i]	0	2	∞	∞	5	∞	∞
C[i]	1						



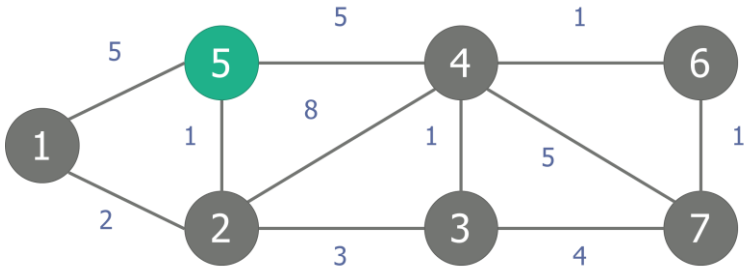
i	1	2	3	4	5	6	7
D[i]	0	2	∞	∞	5	∞	∞
C[i]	1	1					



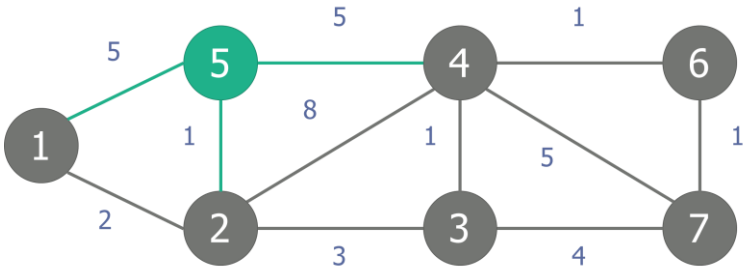
i	1	2	3	4	5	6	7
D[i]	0	2	5	10	3	∞	∞
C[i]	1	1					

최단 경로 – 다익스트라 알고리즘

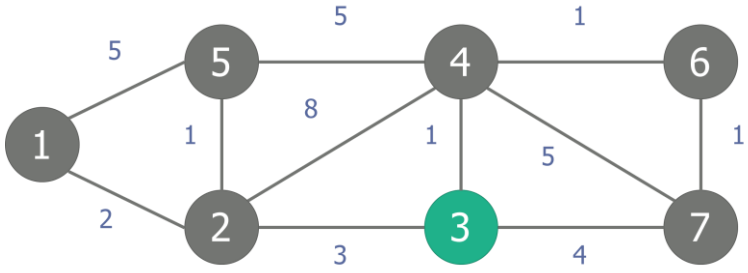
[Shortest path]



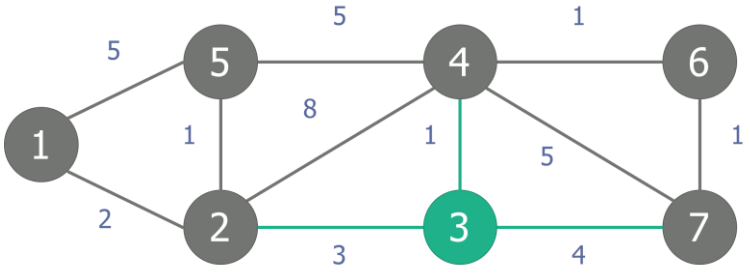
i	1	2	3	4	5	6	7
D[i]	0	2	5	10	3	∞	∞
C[i]	1	1			1		



i	1	2	3	4	5	6	7
D[i]	0	2	5	8	3	∞	∞
C[i]	1	1			1		



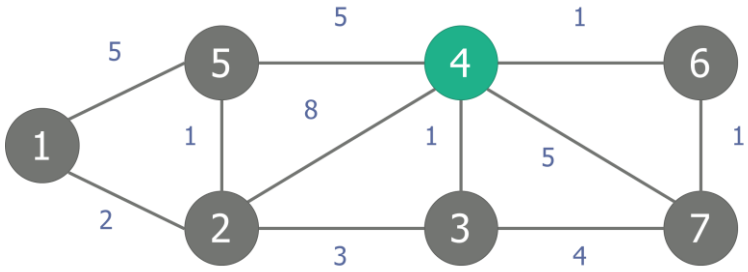
i	1	2	3	4	5	6	7
D[i]	0	2	5	8	3	∞	∞
C[i]	1	1	1		1		



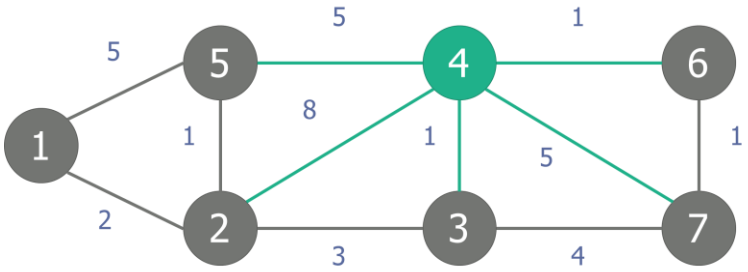
i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	∞	9
C[i]	1	1	1		1		

최단 경로 – 다익스트라 알고리즘

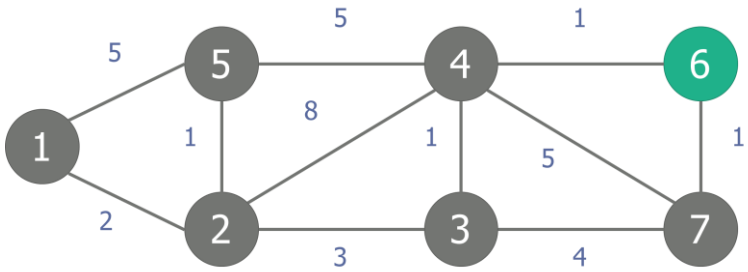
[Shortest path]



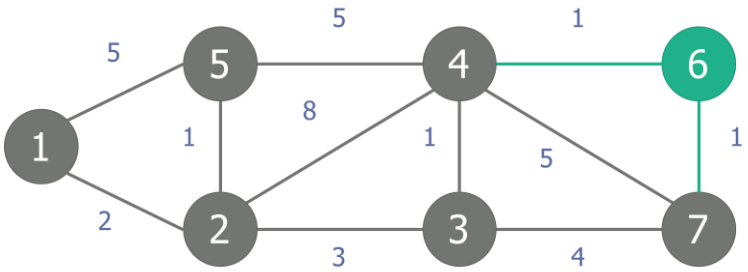
i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	∞	9
C[i]	1	1	1	1	1		



i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	9
C[i]	1	1	1	1	1		



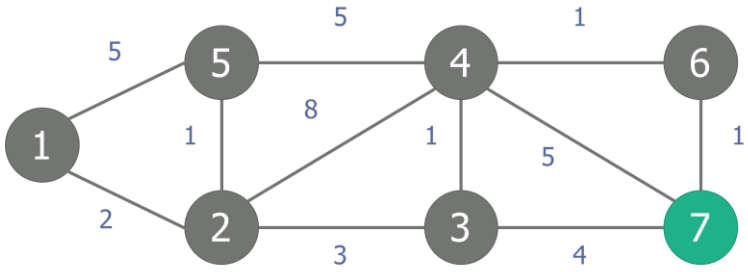
i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	9
C[i]	1	1	1	1	1	1	



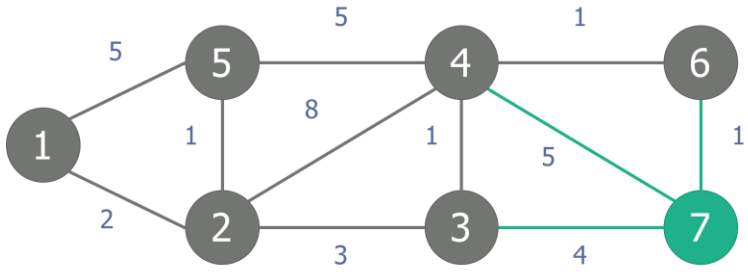
i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	8
C[i]	1	1	1	1	1	1	

최단 경로 – 다익스트라 알고리즘

Shortest path



i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	8
C[i]	1	1	1	1	1	1	1



i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	8
C[i]	1	1	1	1	1	1	1

- 우선순위 큐에 추가되는 원소의 수는 **최대 O(E)**가 된다.
  우선순위 큐에 삽입 삭제 하는데 O(lgE)가 걸리므로 시간 복잡도는 O(ElgE)가 된다.

최단 경로 – 다익스트라 알고리즘

Shortest path

- 우선순위 큐를 이용한 minheap 을 써서 다익스트라 알고리즘을 구현

```
dist[start] = 0;
priority_queue<pi, vector<pi>, greater<pi>> pq;
pq.push(pi(0, start));

while (!pq.empty()) {
        pi here = pq.top();
        pq.pop();
        int u = here.second;

        if (visited[u]) {
                continue;
        }
        visited[u] = true;

        for (int i = 0; i < AdjList[u].size(); ++i) {
                pi next = AdjList[u][i];
                int v = next.first;
                int w = next.second;

                if (dist[v] > dist[u] + w) {
                        dist[v] = dist[u] + w;
                        pq.push(pi(dist[v], v));
                }
        }
}
```