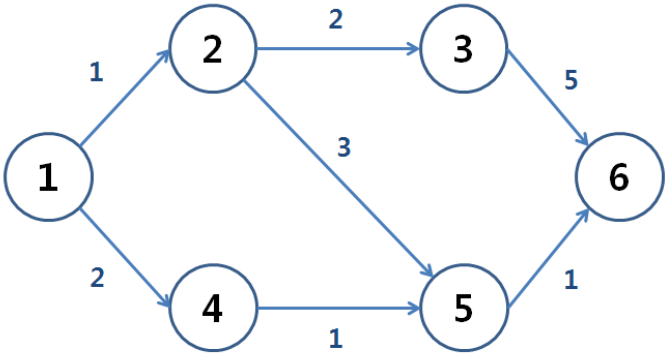
Shortest Path Faster Algorithm (SPFA)

가중치 그래프에서의 단일 시작점 최단경로 알고리즘입니다.

다음과 같은 문제를 생각해 보겠습니다.



총 6개의 도시가 있고, 각 도시 사이에 도로가 있습니다.

도로에서는 화살표가 있는 방향으로만 이동이 가능하며, 이동하는 데에 걸리는 시간은 간선 위에 적혀있는 숫자와 같습니다. 이 때에 1번 도시에서 1번이 아닌 임의의 한 도시에 도달하는 데에 필요한 최단시간은 얼마나 필요할까요?

최단경로 알고리즘 중 널리 알려진 방법으로, Floyd-Warshall 알고리즘과 Dijkstra 알고리즘이 있습니다. 이번에는 구현이 간단하면서, 평균적으로 매우 빠른 수행시간을 보여주는 Shortest Path Faster (SPFA) 알고리즘에 대해서 알아보겠습니다.

SPFA 알고리즘에는 다음의 변수들이 필요합니다.

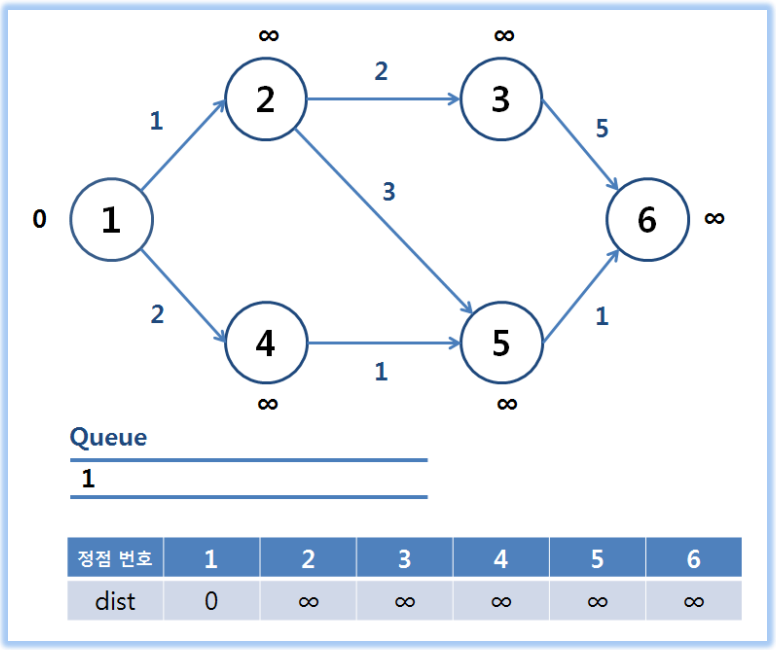
Queue : 최단경로를 계산하기 위해 탐색해야 하는 정점의 리스트

inQ(x) : x번 정점이 Queue 안에 존재하는지의 여부 (0 : 없음, 1 : 있음)

dist(x) : 시작점으로부터 x번 정점에 도달하는 최단경로의 길이

알고리즘 수행 시작 전에는 다음과 같은 상황입니다. dist(1) = 0이고, 나머지 정점들에 대해서는 dist 배열의 값이 무한대입니다. Queue에는 시작점인 1번 정

점이 들어가 있고, inQ(1) = 1이 됩니다.

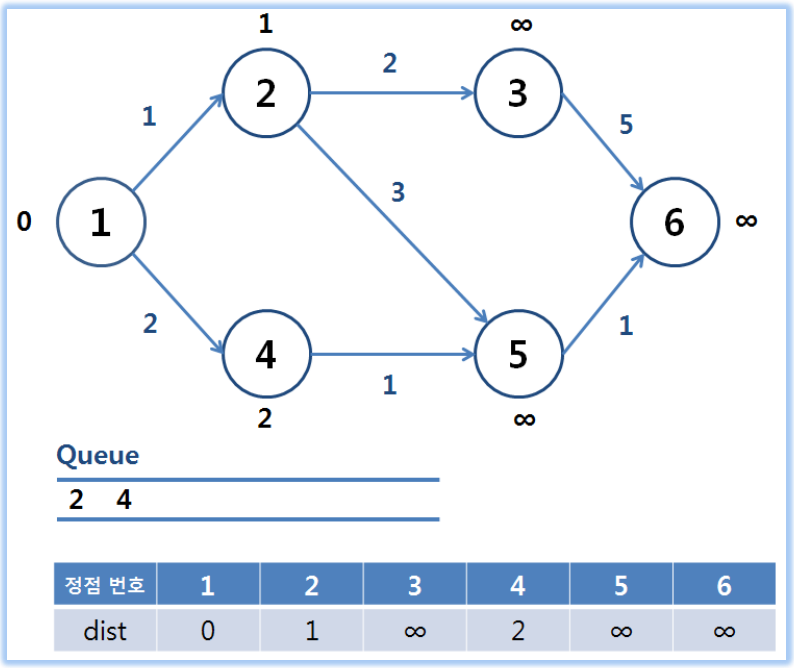


1번 정점을 Queue에서 pop하고 inQ(1) = 0으로 수정합니다.

1번 정점에서 나가는 방향의 간선들을 살펴봅니다. 2번 정점에 도달하는 최단거리를 dist(1) + 1 = 1로 갱신할 수 있으므로 갱신합니다. 2번 정점에 도달하는 최단거리를 갱신하였으므로 2번 정점에 대해 추가적으로 탐색을 수행해야 합니다. 2번 정점이 Queue에 들어있는지를 inQ(2) 값으로 확인합니다. 위의 예에서는 Queue에 2번 정점이 없으므로 Queue에 삽입하고, inQ(2) = 1로 값을 수정합니다.

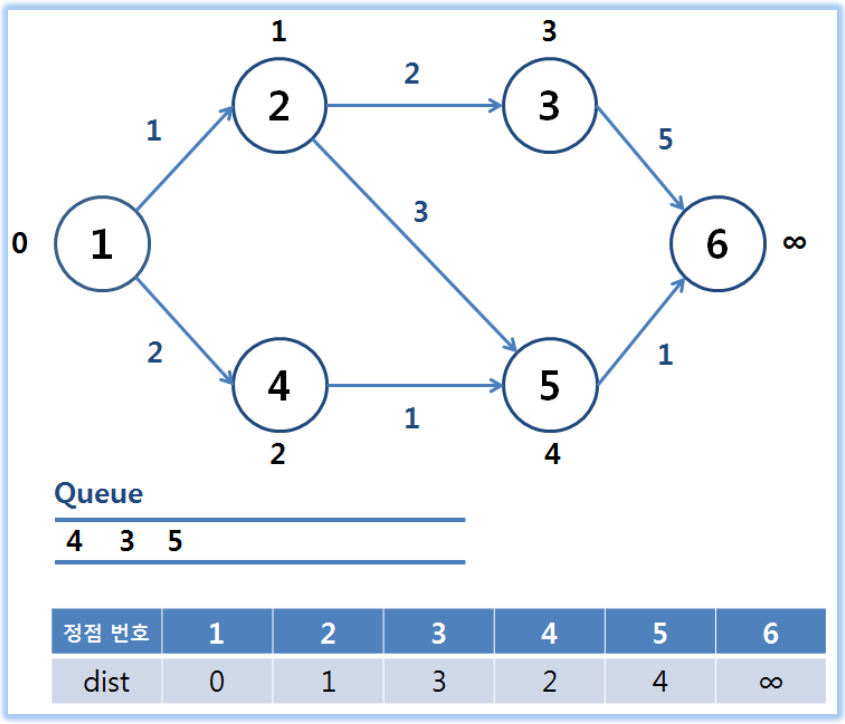
4번 정점에 대해서도 마찬가지로 최단거리를 갱신하고, Queue에 4번 정점이 현재 없으므로 삽입합니다. inQ(4)도 1이 됩니다.

1번 정점에 대해서 모든 간선을 처리한 후의 상황은 다음과 같습니다.



정점 번호	1	2	3	4	5	6
dist	0	1	∞	2	∞	∞

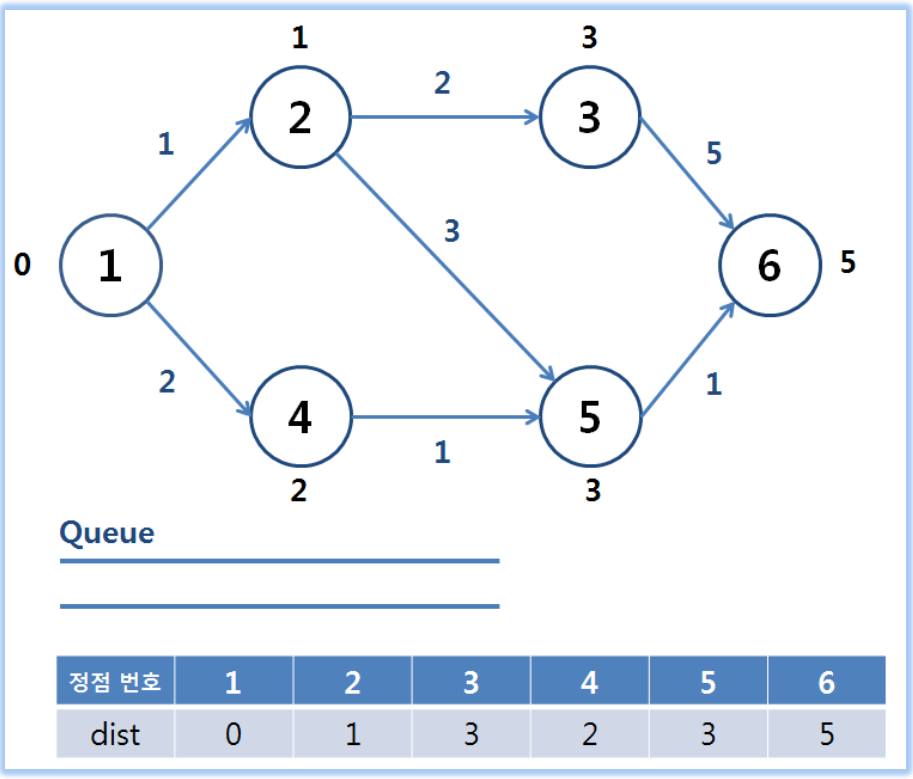
이제 2번 정점을 Queue에서 pop하고 inQ(2)=0이 됩니다. 1번 정점에서 처리하였던 것과 같은 방법으로 간선들을 훑어보며 알고리즘을 수행합니다. 2번 정점에서 나가는 방향의 간선은 3번 정점과 5번 정점을 가리킵니다. 최단거리를 갱신하고 Queue에 삽입하며 inQ 배열을 갱신합니다. 2번 정점을 모두 처리한 이후의 상황은 다음과 같습니다.



정점 번호	1	2	3	4	5	6
dist	0	1	3	2	4	∞

다음으로 4번 정점을 Queue에서 pop하고 inQ(4)=0이 됩니다. 4번 정점에서 5번 정점을 가리키는 간선을 봅니다. dist(5)=4 > dist(4) + 1 = 3이므로 최단경로를 갱신할 수 있습니다. 5번 정점을 Queue에 삽입하려고 했지만, inQ(5)=1이므로 Queue에 5번 정점이 이미 있다는 것을 알 수 있고, Queue에 삽입하지 않습니다.

마지막으로 3번 정점과 5번 정점을 Queue에서 pop하여 위와 마찬가지로 처리하면 최종적으로 1번 정점에서 출발하여 다른 정점에 이르는 모든 최단경로의 길이를 알 수가 있습니다.



정점 번호	1	2	3	4	5	6
dist	0	1	3	2	3	5

SPFA 알고리즘을 Pseudo Code로 정리하면 다음과 같습니다.

```
procedure Shortest-Path-Faster-Algorithm(G, s)
 1    for each vertex v ≠ s in V(G)
 2        d(v) := ∞
 3    d(s) := 0
 4    offer s into Q
 5    while Q is not empty
 6        u := poll Q
 7        for each edge (u, v) in E(G)
 8            if d(u) + w(u, v) < d(v) then
 9                d(v) := d(u) + w(u, v)
10                if v is not in Q then
11                    offer v into Q
```

최악의 경우, O(VE)이지만 평균적으로 O(E)의 수행 복잡도를 보여주는 매우 효과적인 알고리즘입니다.

주의할 점으로 어떤 정점에 이르는 최단경로의 길이가 여러 번 갱신되면, Queue에 그 정점이 여러 번 삽입될 수 있으므로, Queue를 환형 큐로 구현하거나 Queue의 크기를 적당한 상수 (50 ~ 100) * |정점의 개수| 만큼 설정해주어야 합니다.