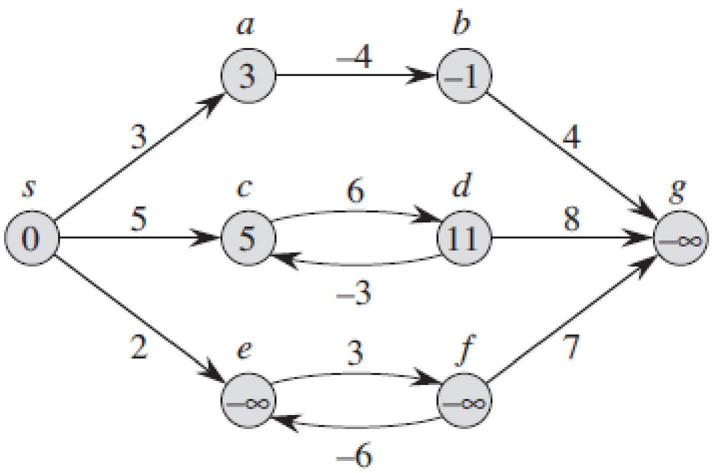
Shortest-Path Problem (최단경로 문제)

벨만-포드 알고리즘은 Directed Weighted Graph (유향 가중 그래프) 에서 Shortest path (최단 경로) 를 구하는 알고리즘이다. 가중 그래프에서 **최단 경로란 어떤 vertex까지의 경로 중 edge의 weight 합을 최소로 하는 경로** 를 말하며 다음과 같은 종류의 문제들이 있다.

- Single-source shortest path : 정점 s에서 출발하여 임의의 정점 v까지의 shortest path를 구하는 문제
- Single-destination shortest path : 임의의 정점 s에서 출발하여 정점 v까지의 shortest path를 구하는 문제로 single-source shortest path로 바꿔서 풀면 된다.
- Single-pair shortest path : 정점 s에서 출발하여 정점 v까지의 shortest path를 구하는 문제
- All-pairs shortest path : 임의의 정점 s에서 출발하여 임의의 정점 v까지의 shortest path를 구하는 문제

위의 최단 경로 문제 중 벨만-포드 알고리즘이 적용되는 문제는 1번째인 **single-source shortest path** 문제이다. 벨만-포드 알고리즘은 negative weight(음의 가중치)이 적용된 간선을 허용한다. 하지만, **negative-weight-cycle (음의 가중치 순환) 은 허용하지 않는다.** 이유가 무엇일까?



다음과 같은 그래프에서 s→g의 경로 중 c,d와 e,f는 cycle을 생성하고 둘 다 negative-weight-edge를 가지고 있다.

- c,d : c→d가 6이고 d→c가 -3으로 양수의 weight이 더 크기 때문에 아무리 cycle을 돈다 하더라도 최단경로는 1가지 밖에 안된다.
- e,f : e→f가 3이고 f→e가 -6으로 음수의 weight이 더 크기 때문에 cycle을 돌면 돌수록 최단경로가 계속해서 갱신된다. 따라서 음의 무한대까지 계속될 수 있기 때문에 이런 negative-weight-cycle은 허용되지 않는다.

이제 실제 벨만-포드 알고리즘을 알아보기 위해 2가지 개념에 대해 알아보기로 하자.

Initialization

Initialization은 초기화인데 최단경로 알고리즘을 적용하기 위해서 필요한 준비라고 생각하면 된다. 임의의 vertex v에 대해서 항상 다음 값들을 계산해 주어야 한다.

- $d[v]$: shortest path estimate로 계속해서 갱신되는 최단경로의 후보값들을 말한다. 최종적으로는 최단경로의 값이 되기 때문에 $d[v] \geq \delta(s,v)$ 가 성립한다. $\delta(s,v)$ 는 s→v 경로 중 최단 경로 값을 의미한다.
- $\pi[v]$: predecessor of v on a shortest path from s로 s→v의 최단 경로 중 v의 직전 vertex를 의미한다.

그래프의 가장 초기 상태에서 시작 vertex는 당연히 시작이니까 $d[v]$가 0이고 나머지는 모르기 때문에 ∞로 초기화 해주고, 시작 vertex를 포함한 모든 vertex는 아직 predecessor를 모르기 때문에 $\pi[v]$는 NIL이다.

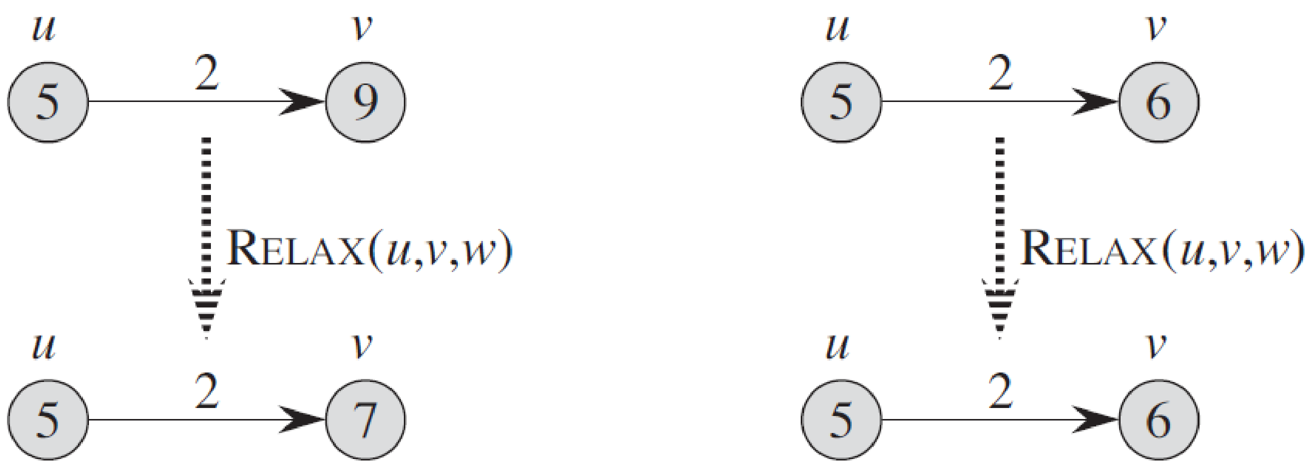
INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in G.V$
  $v.d = \infty$
  $v.\pi = \text{NIL}$
$s.d = 0$

따라서, 다음과 같은 INITIALIZE-SINGLE-SOURCE(G,s) 함수로 초기 그래프의 d, π 값들을 초기화 해주어야 한다.

Relaxation

Relaxation은 새로운 경로를 발견해서 최단 경로를 갱신하는 것을 말한다.



(a) (b)

- 왼쪽 그림에선 u→v로 가는 경로 중 최단경로가 9였는데 새로운 경로가 발견되어서 7이 될 수 있다면 9를 7로 업데이트 시켜주는 것이다.
- 오른쪽 그림에선 u→v로 가는 경로 중 최단경로가 6이었는데 새로운 경로도 6이라면 둘 중 어느 것을 택할지는 이제 구현의 문제가 된다.

RELAX(u, v, w)

if $v.d > u.d + w(u, v)$
  $v.d = u.d + w(u, v)$
  $v.\pi = u$

RELAX(u,v,w) 함수는 u→v의 경로에서 v의 최단경로보다 u의 최단경로와 u→v의 가중치를 합한 값이 더 작을 경우 그 경로로 업데이트를 시켜준다. u의 최단경로 + u→v의 가중치가 새로운 경로를 의미한다.

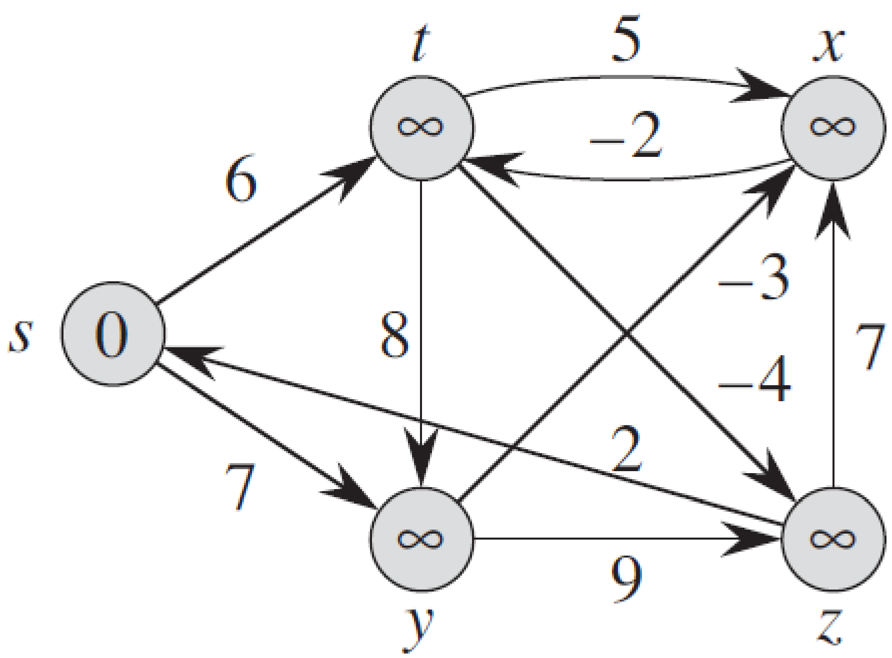
Bellman-Ford Algorithm (벨만-포드 알고리즘)

이제 진짜 벨만-포드 알고리즘을 살펴보자. 먼저 수도코드 부터 보면,

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| - 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

- 1번째 줄에서 INITIALIZE-SINGLE-SOURCE 함수로 그래프를 초기화 시킨다.
- 2~4번째 줄에서 $|V|-1$ 번 반복하는데 그 이유는 Cycle이 없는 경우 edge의 최대 개수가 $|V|-1$ 개 이기 때문이다.
- $|V|-1$ 번 반복할 때, 각각의 반복에서 그래프의 모든 edge에 대하여 RELAX 함수 relaxation을 해줘야 한다. 그 이유는 최단경로가 새로운 경로로 갱신될 수 있기 때문이다.
- 5~8번째 줄에선, 새로 업데이트 되는 최단경로가 있는지 모든 edge에 대해서 검사한 뒤 있으면 negative-weight-cycle이 존재하는 것이므로 FALSE를 리턴하고 없으면 TRUE를 리턴한다.

이제 그래프에 알고리즘이 어떻게 적용되는지 살펴보자.



첫번째로 그래프에서 s로부터 각 vertex까지의 distance를 ∞로 초기화해준다. 또한 여기선 보이지 않지만 π의 값 또한 NIL로 초기화 되어있는 상태이다. s는 시작 vertex이니 distance는 0이다.

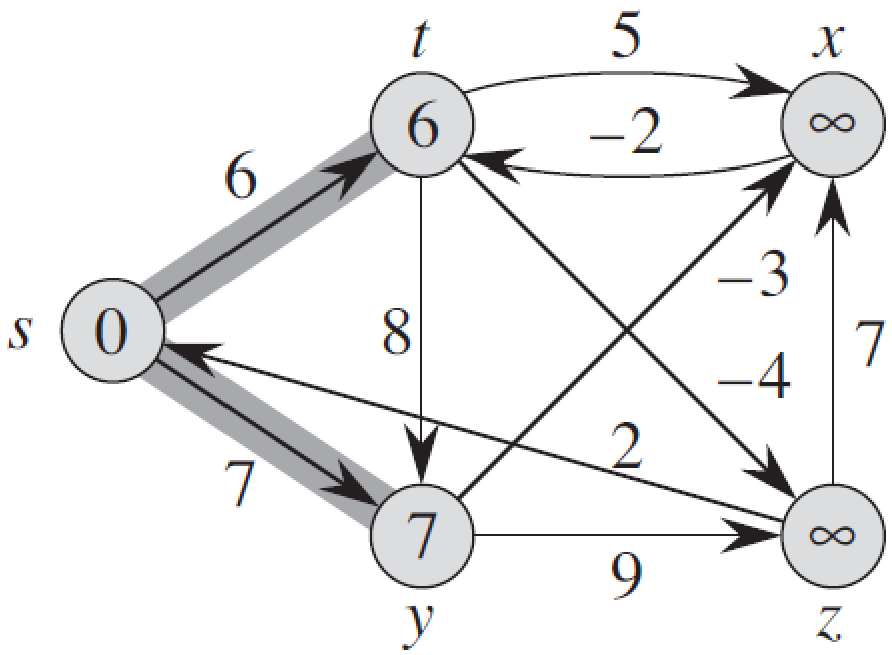
이제 $|V|-1$ 번인 4번동안 모든 edge에 대하여 relaxation을 해야 한다. 어떤 순서대로 해도 상관 없지만, $|V|-1$ 번 동안 할 때의 순서는 항상 같아야 한다. 순서는 다음과 같다.

$$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$$

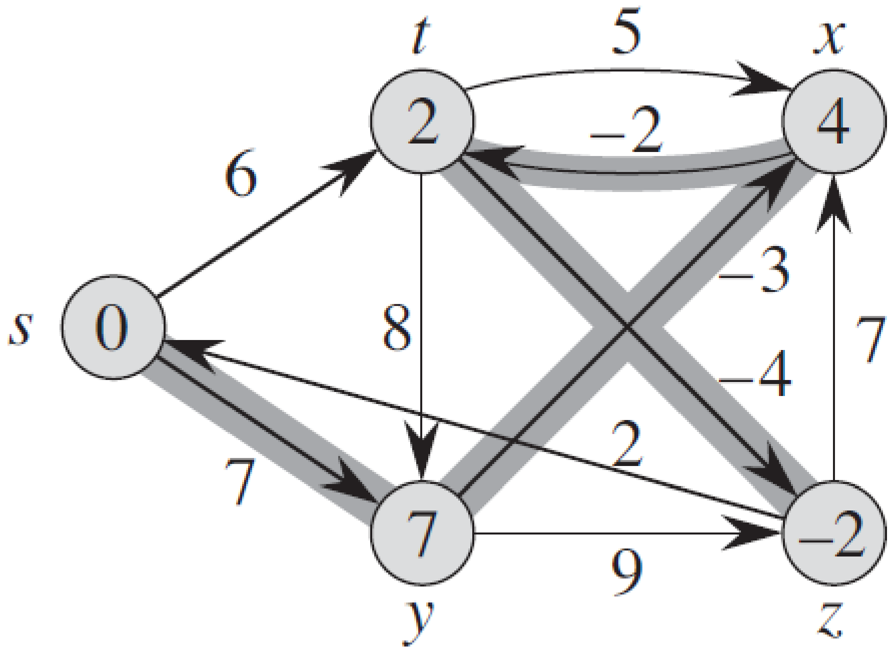
이제 이 순서로 계속 relaxation을 진행한다. 위의 그래프에서 1번째 relaxation이니 전부 해보자.

- t → x : ∞ > ∞ +5 가 성립하지 않기 때문에 relaxation X
- t → y : ∞ > ∞ +8 이 성립하지 않기 때문에 relaxation X
- t → z : ∞ > ∞ +(-4) 가 성립하지 않기 때문에 relaxation X
- x → t : ∞ > ∞ +(-2) 가 성립하지 않기 때문에 relaxation X
- y → x : ∞ > ∞ +(-3) 이 성립하지 않기 때문에 relaxation X
- y → z : ∞ > ∞ + 9 가 성립하지 않기 때문에 relaxation X
- z → x : ∞ > ∞ + 7 이 성립하지 않기 때문에 relaxation X
- z → s : ∞ > ∞ + 0 이 성립하지 않기 때문에 relaxation X
- s → t : ∞ > 0 + 6 이 성립하기 때문에 relaxation을 해서 $d[t]=6, \pi[t]=s$ 가 된다.
- s → y : ∞ > 0 + 7 이 성립하기 때문에 relaxation을 해서 $d[y]=7, \pi[y]=s$ 가 된다.

이 모든 과정을 진행한 뒤 아래 그래프가 나온다.



단 2개의 edge만 실제 relaxation이 된 것을 볼 수 있다. 이렇게 남은 3번을 모든 edge에 대하여 위의 과정과 같이 relaxation을 반복하면 모든 vertex에 대한 s로부터의 shortest path가 결정된다. 과정이 똑같으니 마지막 그림만 보고 끝내도록 하자.



시간복잡도

모든 edge E에 대하여 $|V| - 1$ 번 만큼 relaxation을 반복하는데 relaxation 연산 자체는 distance를 비교해서 더하고 predecessor를 설정해주면 되기 때문에 constant time이다. 따라서 결국 시간복잡도는 $\theta(VE)$로 느린편이라고 할 수 있다.