자료구조 (Data Structure
9주차: 그래프 최단경로

그래프

- 그래프는 노드와 연결선을 모은 것이다

  - vertexes: 전체 노드 리스트

  - edges: 전체 연결선 리스트

  - adj_matrix: 연결선 이차원 배열

그래프 노드

- 그래프 노드는 데이터와 이웃 리스트를 갖고 있다

  - distance: 특정 노드로부터의 최단 거리

  - parent: 최단 경로 상에서 이전 노드

  - status: 방문 상태

  - adj_list: 연결선 이차원 리스트

그래프 연결선

- 그래프 연결선에는 연결되는 노드와 데이터가 있다

  - length: 연결선의 비용

  - node1: 연결선의 노드

  - node2: 연결선의 노드

그래프 최단거리

- 그래프에서 한 노드에서 다른 노드로의 최단거리를 구해야할 경우는?

- 예시) 네비게이션, 인터넷 통신, 공격 경로 분석

이번 시간 목차

- 그래프의 한 노드에서 다른 모든 노드까지 최단거리

  - 연결선 비용이 모두 1일 때

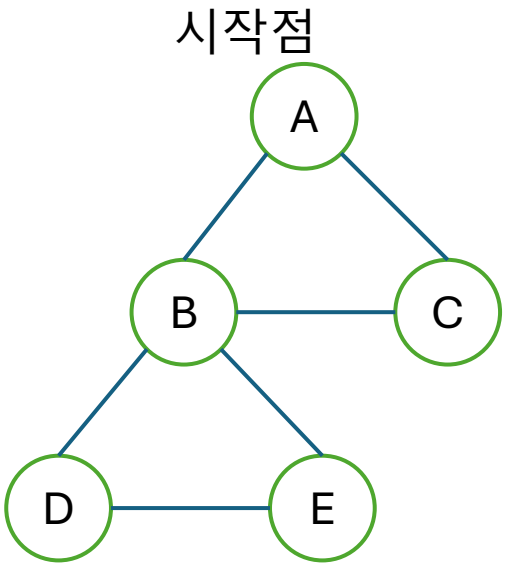
  - 연결선 비용이 0 또는 1일 때

  - 연결선 비용이 0 또는 양수일 때

  - 연결선 비용에 음수가 있을 때

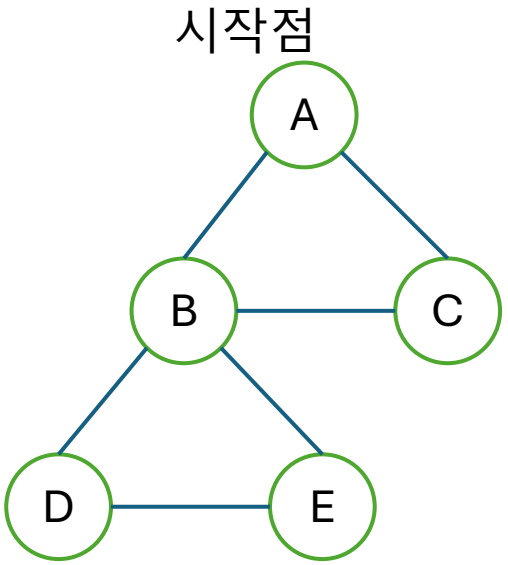
연결선 비용이 모두 1일 때

- 최단거리가 0인 노드는?
- 최단거리가 1인 노드는?
- 최단거리가 2인 노드는?



연결선 비용이 모두 1일 때

- 최단거리가 n인 노드들을 전부 알고 있을 때, 최단거리가 n + 1인 노드들을 찾는 방법은?



연결선 비용이 모두 1일 때 - BFS 1/2

함수 BFS(그래프 G, 시작 노드 s):

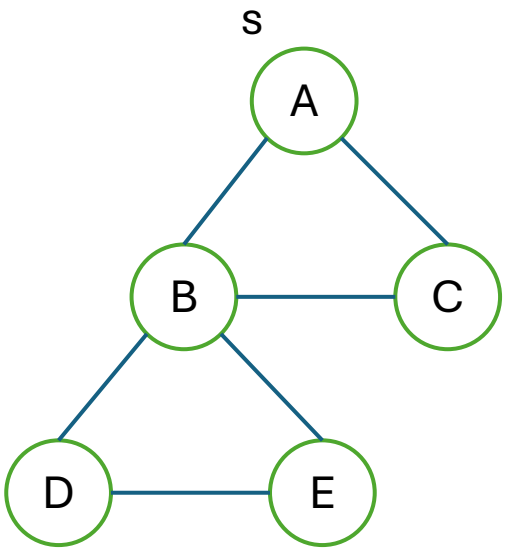
- to_visit = 새로운 큐
- 반복: G의 모든 노드 u에 대해,
  - u->distance = 아주 큰 수
  - u->parent = 없음
  - u->status = 추가된적없음
- s->distance = 0
- s->status = 추가된적있음
- to_visit에 s 추가

연결선 비용이 모두 1일 때 - BFS 2/2

- 반복: to_visit에 데이터가 있는 동안
  u = DEQUEUE(to_visit)
  반복: u의 모든 이웃 노드 v에 대해,
    만약: v->status가 방문전이면,
      v->parent = u
      v->distance = u->distance + 1
      v->status = 추가된적있음
      to_visit에 v 추가

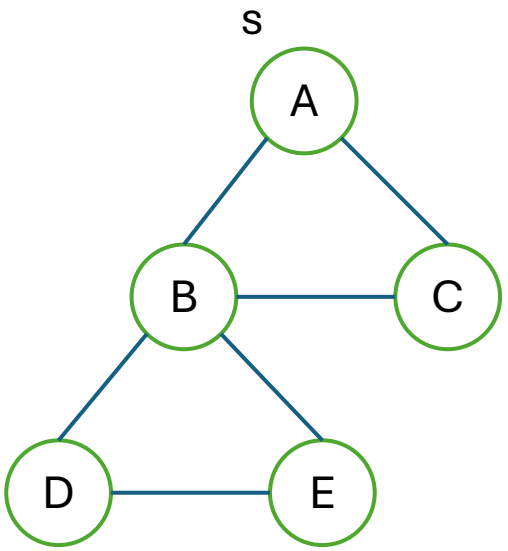
연결선 비용이 모두 1일 때

- BFS(G, s)에서 최단거리가 보장되는 이유는?



연결선 비용이 모두 1일 때

- 다음 그래프에서 BFS(G, s)를 실행하면?



연결선 비용이 모두 1일 때 - 시간복잡도

- 초기화
  - 모든 노드 V에 값 설정. O(V)

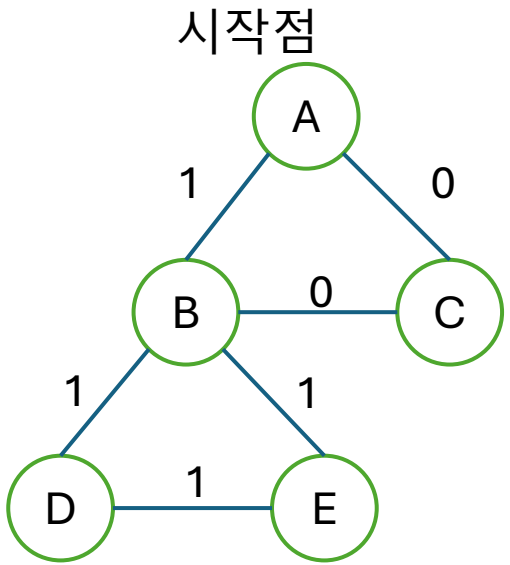
- 큐 연산
  - 모든 노드가 한번 큐에 들어가고 나옴. O(V)

- 이웃 탐색
  - 모든 노드의 이웃을 방문. 연결선 전체. O(E)

총 O(V + E)

연결선 비용이 0 또는 1일 때

- 최단거리가 0인 노드는?
- 최단거리가 1인 노드는?
- 최단거리가 2인 노드는?



연결선 비용이 0 또는 1일 때

- 최단거리를 구한 노드들로부터 다른 노드들의 최단거리를 구하는 방법은?

연결선 비용이 0 또는 1일 때

- 양방향 큐

  - PUSH_FRONT(): 제일 앞에 데이터 추가
  - PUSH_REAR(): 제일 뒤에 데이터 추가

  - POP_FRONT: 제일 앞 데이터 삭제
  - POP_REAR: 제일 뒤 데이터 삭제

연결선 비용이 0 또는 1일 때 - BFS_0_1 1/3

함수 BFS_0_1(그래프 G, 시작 노드 s):

- to_visit = 새로운 양방향 큐
- 반복: G의 모든 노드 u에 대해,
  - u->distance = 아주 큰 수
  - u->parent = 없음
  - u->status = 최단거리발견전

- s->distance = 0
- PUSH_FRONT(to_visit, s)

연결선 비용이 0 또는 1일 때 - BFS_0_1 2/3

- 반복: to_visit에 데이터가 있는 동안

  u = POP_FRONT(to_visit)

  만약: u->status != 최단거리발견전이면,

  다음 반복으로

  u->status = 최단거리발견됨

  반복: u의 모든 이웃 v와 (u, v)를 연결선 e에 대해,

  만약: v->distance

  ≤ u->distance + e->length,

  다음 반복으로

연결선 비용이 0 또는 1일 때 - BFS_0_1 3/3

v->parent = u

v->distance = u->distance + e->length

만약: e->length == 0이면,

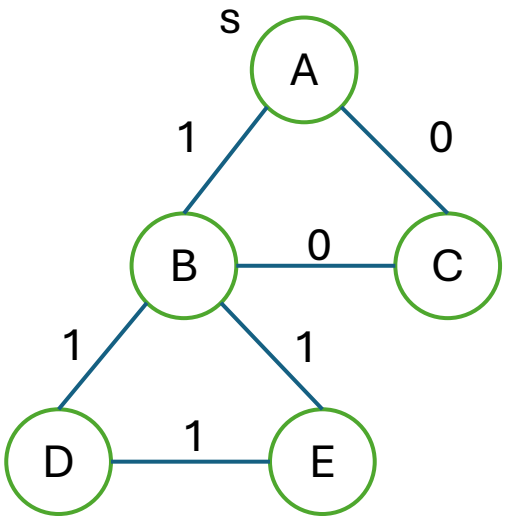
PUSH_FRONT(to_visit, v)

그 외에:

PUSH_REAR(to_visit, v)

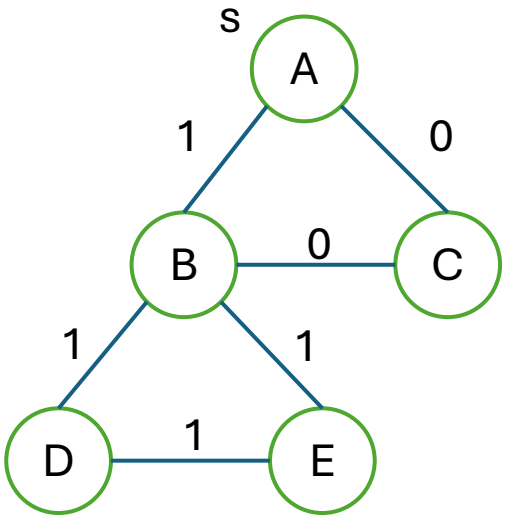
연결선 비용이 0 또는 1일 때

- BFS_0_1(G, s)와 BFS(G, s)의 차이점은?



연결선 비용이 0 또는 1일 때

- BFS_0_1(G, s)에서 최단거리가 보장되는 이유는?



연결선 비용이 0 또는 1일 때 - 시간복잡도

- 초기화
  - 모든 노드 V에 값 설정. O(V)

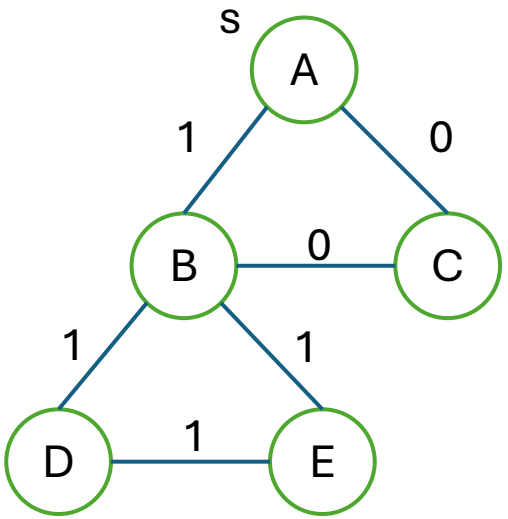
- 이웃 탐색
  - 모든 노드의 이웃을 방문.
  - 최악 방문할 때마다 큐에 추가. O(E)

- 큐에서 삭제: 한번 삭제. O(E)

총 O(V + E)

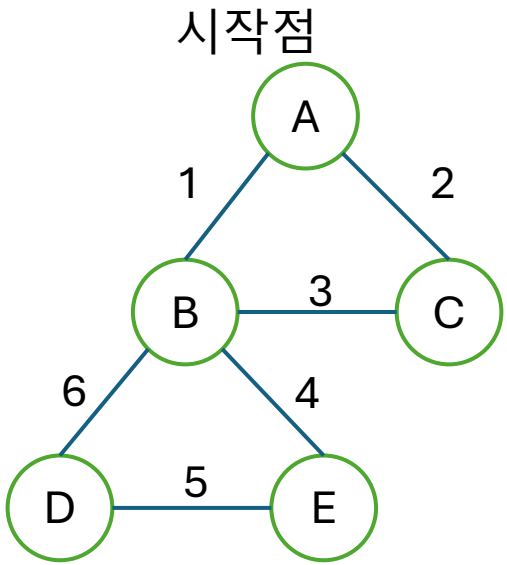
연결선 비용이 0 또는 1일 때

- 다음 그래프에서 BFS_0_1(G, s)를 실행하면?



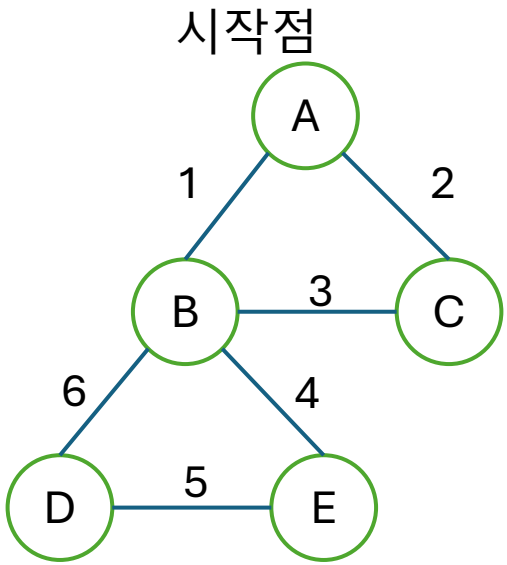
연결선 비용 ≥ 0일 때

- 최단거리가 가장 작은 노드는?
- 최단거리가 두번째로 작은 노드는?
- 최단거리가 세번째로 작은 노드는?



연결선 비용 ≥ 0일 때

- 최단거리를 구한 노드들로부터 다른 노드들의 최단거리를 구하는 방법은?



연결선 비용 ≥ 0일 때 - DIJKSTRA 1/3

함수 DIJKSTRA(그래프 G, 시작 노드 s):

- to_visit = 우선순위 큐 (최소 length)
- 반복: G의 모든 노드 u에 대해,

  u->distance = 아주 큰 수

  u->parent = 없음

  u->status = 최단거리발견전

- s->distance = 0
- to_visit에 s 추가

연결선 비용 ≥ 0일 때 - DIJKSTRA 2/3

- 반복: to_visit에 데이터가 있는 동안

  u = DEQUEUE(to_visit)

  만약: u->status == 최단거리발견됨이면,

  다음 반복으로

  u->status = 최단거리발견됨

연결선 비용 ≥ 0일 때 - DIJKSTRA 3/3

반복: u의 모든 이웃 노드 v와 (u, v)를 연결선 e,

만약: v->distance

≤ u->distance + e->length이면,

다음 반복으로

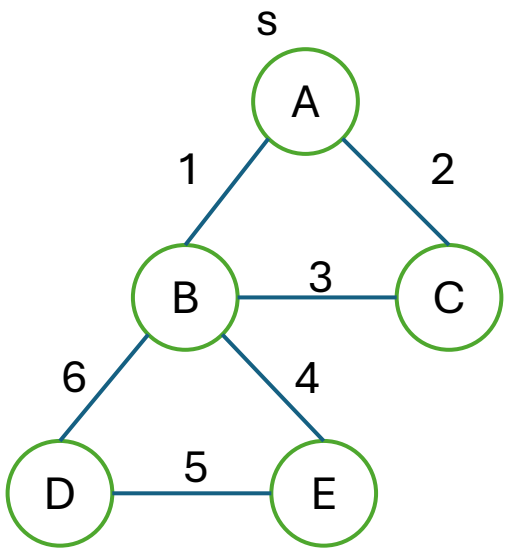
v->parent = u

v->distance = u->distance + e->length

ENQUEUE(to_visit, v)

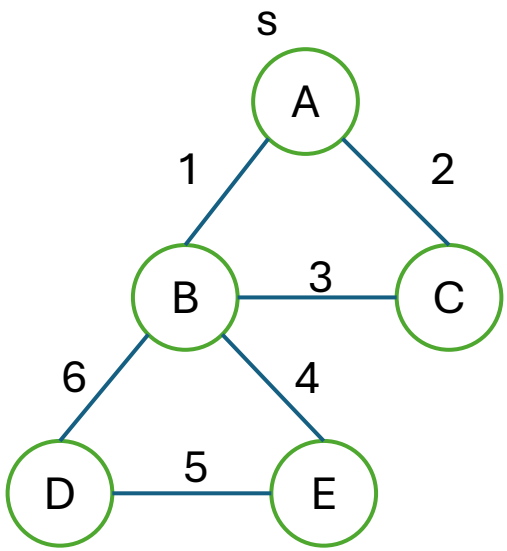
연결선 비용 ≥ 0일 때

- DIJKSTRA(G, s)와 BFS_0_1(G, s)의 차이점은?



연결선 비용 ≥ 0일 때

- DIJKSTRA(G, s)에서 최단 거리가 보장되는 이유는?



연결선 비용 ≥ 0일 때 - 시간복잡도

- 초기화
  - 모든 노드 V에 값 설정. O(V)

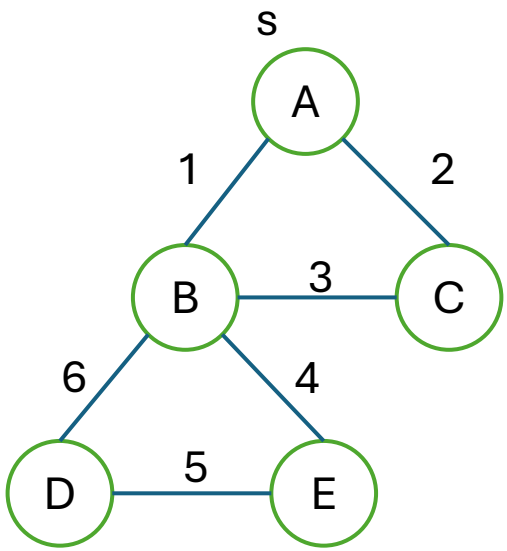
- 이웃 탐색
  - 모든 노드의 이웃을 방문.
  - 최악 방문할 때마다 큐에 추가. O(E log E)

- 큐에서 삭제: 한번 삭제. O(E log E)

총 O(E log E) = O(E log V)

연결선 비용 ≥ 0일 때

- DIJKSTRA(G, s)가 실행되는 순서는?



연결선 비용 ≥ 0일 때 DIJKSTRA_SIMPLE 1/2

함수 DIJKSTRA_NOHEAP(그래프 G, 시작 노드 s):

- 반복: G의 모든 노드 u에 대해,

  u->distance = 아주 큰 수

  u->parent = 없음

  u->status = 최단거리발견전

- s->distance = 0

연결선 비용 ≥ 0일 때 DIJKSTRA_SIMPLE 2/2

- 반복: G에 최단거리발견전인 노드가 있는 동안
  u = G의 최단거리발견전 노드 중 최소 distance
  u->status = 최단거리발견됨
  반복: u의 모든 이웃 v와 (u, v) 연결선 e,
    만약: v->distance
      ≤ u->distance + e->length이면,
      다음 반복으로
    v->parent = u
    v->distance = u->distance + e->length

연결선 비용 ≥ 0일 때 - 힙없이 시간복잡도

- 초기화
  - 모든 노드 V에 값 설정. O(V)

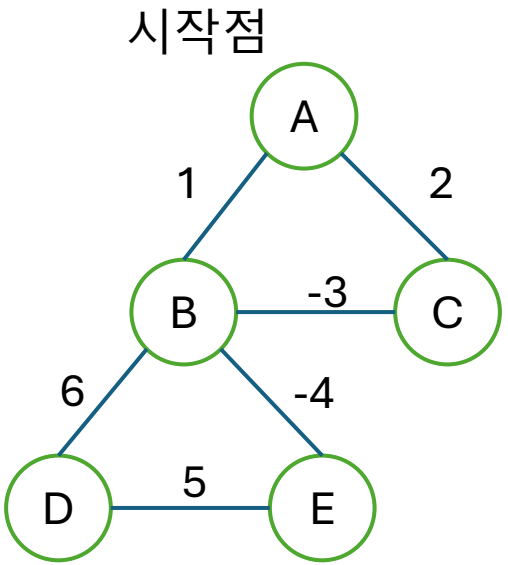
- 이웃 탐색
  - 이웃 행렬에서 모든 노드의 이웃 확인. O(E)

- 노드 선택:
  - 모든 노드 확인을 V번 반복. $O(V^2)$

총 $O(E + V^2) = O(V^2)$

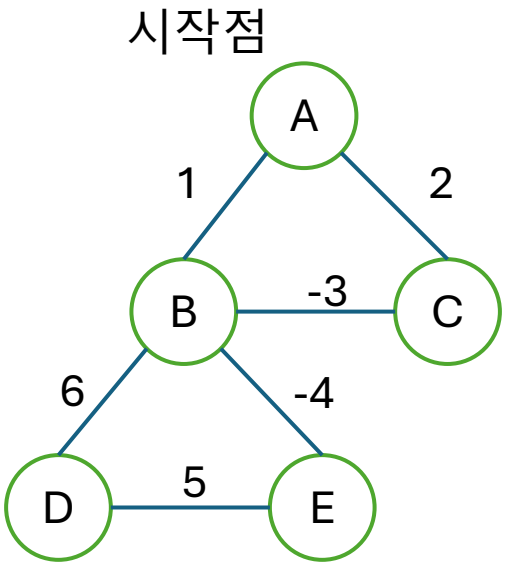
일반적인 경우

- 1회차에서 시작점부터 노드 A까지 최단거리는?
- 2회차에서 시작점부터 노드 A까지 최단거리는?
- 3회차에서 시작점부터 노드 A까지 최단거리는?



일반적인 경우

- 최대 반복해야할 횟수는?



일반적인 경우 - BELLMAN_FORD 1/3

함수 BELLMAN_FORD(그래프 G, 시작 노드 s):

- 반복: G의 모든 노드 u에 대해,
  - u->distance = 아주 큰 수
  - u->parent = 없음
- s->distance = 0
- 반복: G의 노드 개수 - 1번
  - 반복: G의 모든 연결선 e에 대해,
    - u = e->node1
    - v = e->node2

일반적인 경우 - BELLMAN_FORD 2/3

만약: u->distance < 아주 큰수이고,

v->distance

> u->distance + e->length면,

v->distance

= u->distance + e->length

v->parent = u

일반적인 경우 - BELLMAN_FORD 3/3

만약: v->distance < 아주 큰수이고,

u->distance

> v->distance + e->length면,

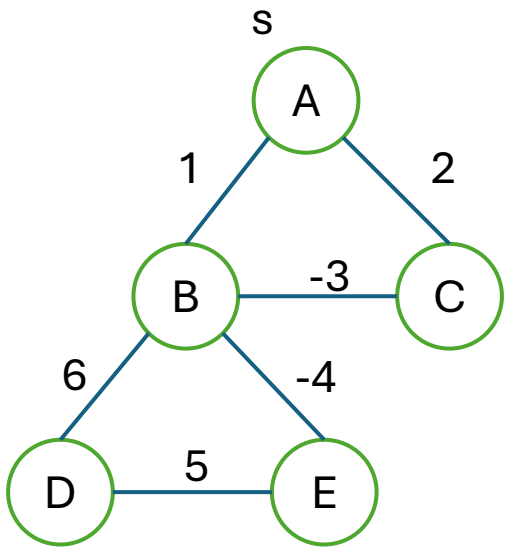
u->distance

= v->distance + e->length

u->parent = v

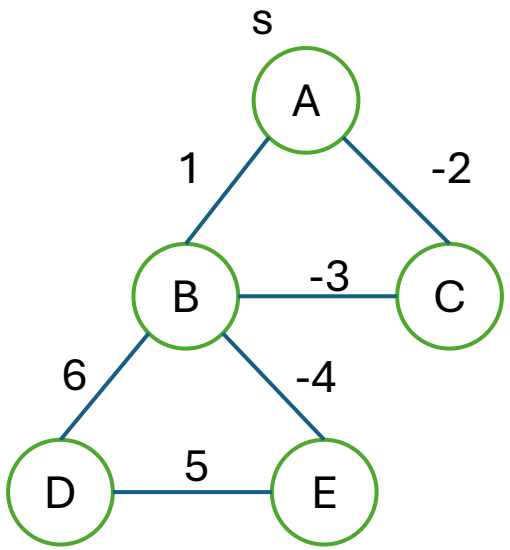
일반적인 경우

- 다음 그래프에서 BELLMAN_FORD(G, s)를 실행하면?



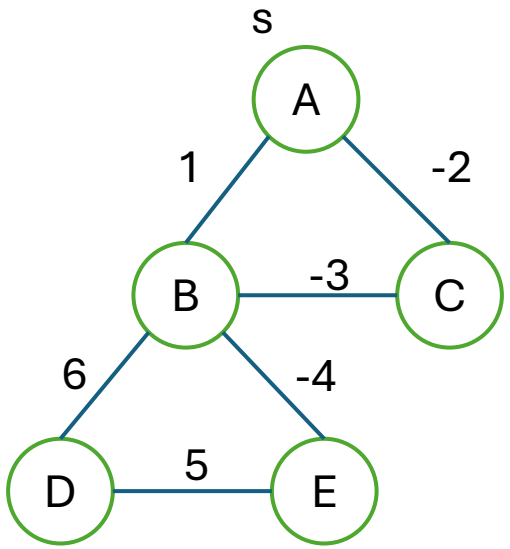
일반적인 경우

- 음수 사이클이 있다면?



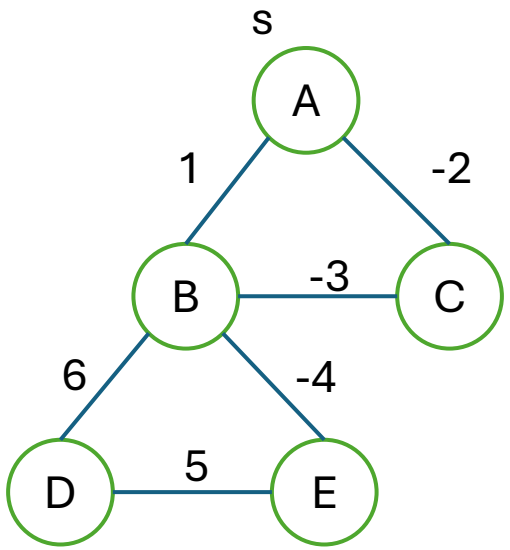
일반적인 경우

- 음수 사이클이 있는 경우와 없는 경우 |V|번째 반복에서 일어나는 일은?



일반적인 경우

- 음수 사이클을 찾는 방법은?



일반적인 경우 1/2

- 함수 BELLMAN_FORD_SAFE(그래프 G, 시작 노드 s):
- BELLMAN_FORD(G, s)
- 반복: G의 모든 연결선 e에 대해,

    u = e->node1

    v = e->node2

        만약: u->distance < 아주 큰수이고,

            v->distance

            > u->distance + e->length이면,

            반환: 음수사이클발견

일반적인 경우 2/2

만약: v->distance < 아주 큰수이고,

u->distance

> v->distance + e->length이면,

반환: 음수사이클발견

요약

- 너비 우선 탐색: 연결선의 비용이 모두 1일 때, 최단 거리 계산

- DIJKSTRA: 연결선의 비용이 0 또는 양수일 때
- BELLMAN-FORD: 연결선의 비용에 음수가 있을 때도