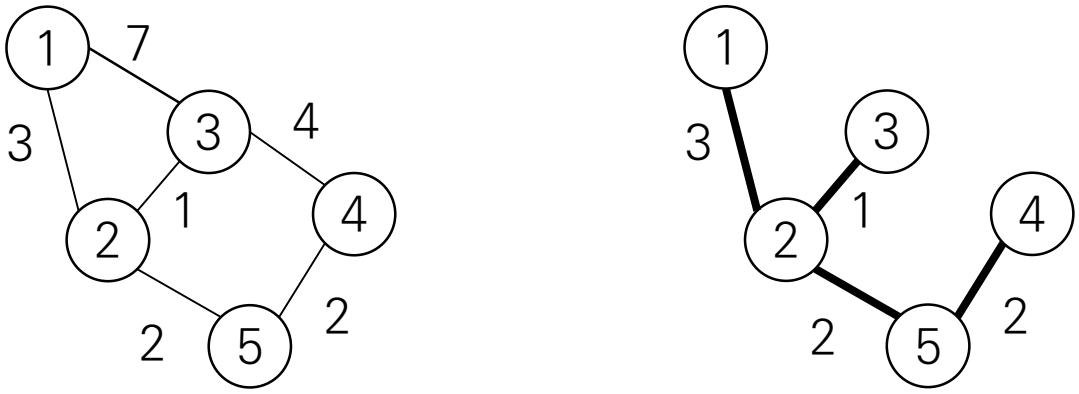
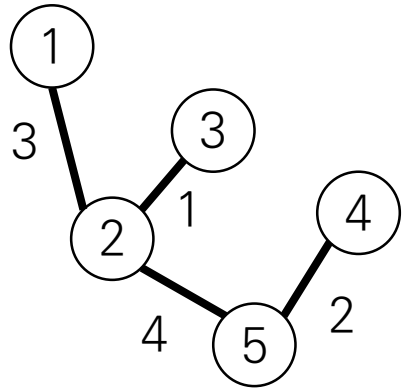
최소신장트리(MST)

- 가중치를 가진 무향그래프에서, 간선의 가중치의 합이 최소가 되도록 모든 노드를 연결한 트리.



- 한 그래프에 여러 개의 MST가 존재할 수 있다.

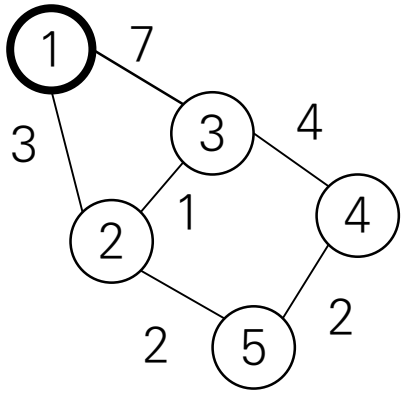
- MST에 속한 간선의 저장.
  - 자식을 인덱스로 부모를 저장하는 방식으로 저장하거나 간선의 배열 형태로 저장.



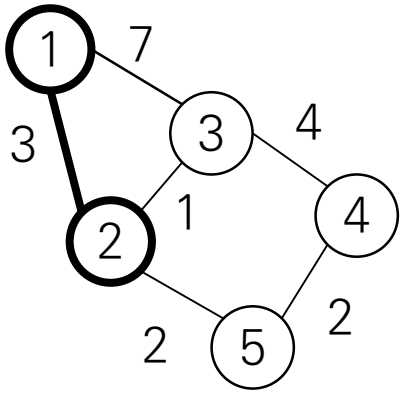
1	2	3	4	5
1	1	2	5	2

1	2
2	3
5	4
2	5

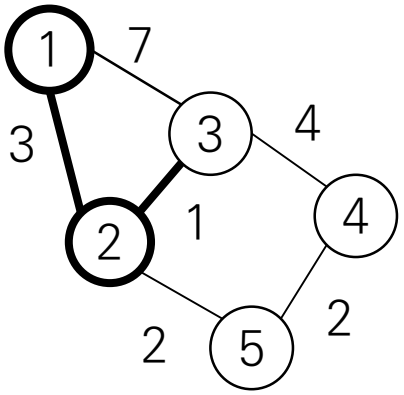
Prim 알고리즘

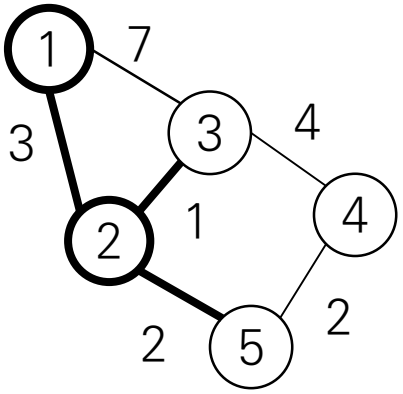


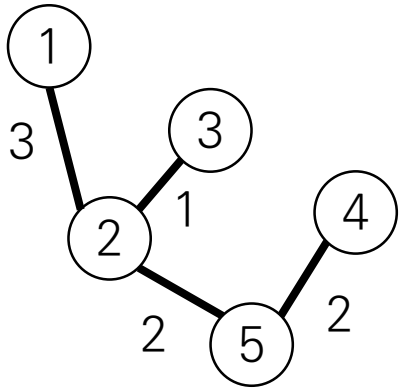
임의의 노드를 선택해
MST에 포함시킨다.



MST에 속한 노드와 인접하고, 아직
MST에 속하지 않은 노드 중 비용이
최소인 노드를 추가한다.







모든 노드가 MST에 포함되면 종료한다.

- MST에 속한 노드에 인접한 다른 노드를 찾는 과정이 중복되어 시간이 오래걸리는 단점이 있다.
  - 예) 1의 인접, 1-2의 인접, 1-2-3의 모든 인접
- 비용이 최소인 인접 노드를 찾아 노드 번호를 우선순위큐(또는 최소힙)에 넣어서 처리하면 시간이 단축된다. (Kruskal 알고리즘과 처리 시간이 비슷함.)

▪ Prim 알고리즘

비용을 표시한 인접 행렬 생성.

노드 한 개를 MST에 추가.

모든 노드가 MST에 포함 될 때까지 다음을 반복.

MST에 포함된 모든 노드에 대해, MST에 미포함된 인접 노드와의 비용을 비교.

최소 비용인 인접 노드를 MST에 추가.

MST에 포함된 모든 노드에 대해 조사하는 부분이 반복되어 노드가 많은 경우 시간이 오래 걸림.

▪ 우선순위 큐를 사용한 Prim 알고리즘

비용을 표시한 인접 행렬 생성.

노드 n1을 MST에 추가.

n1과 인접이면서 MST가 아닌 모든 노드 n2와의 에지 e(n1,n2)를 큐에 추가.

모든 노드가 MST에 포함 될 때까지 다음을 반복.

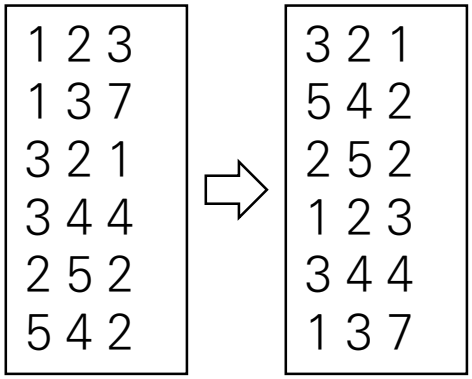
    t(n1, n2) <- 디큐 (비용이 최소인 에지)

    n2가 MST에 없으면

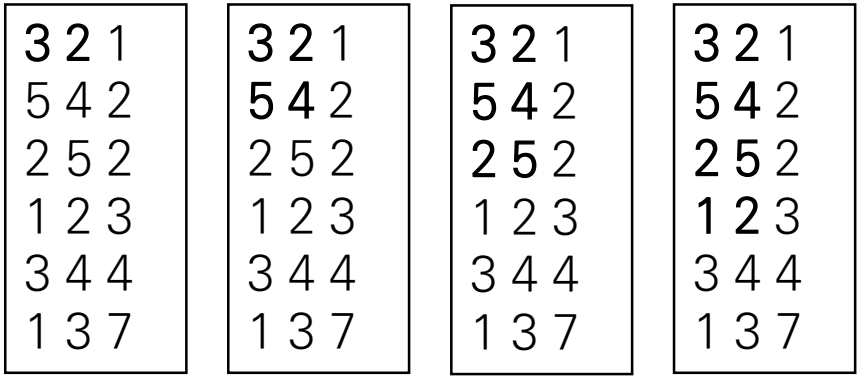
        n2를 MST에 추가.

        n2와 인접이면서 MST가 아닌 모드 노드 n3와의 에지 e(n2, n3)를 큐에 추가.

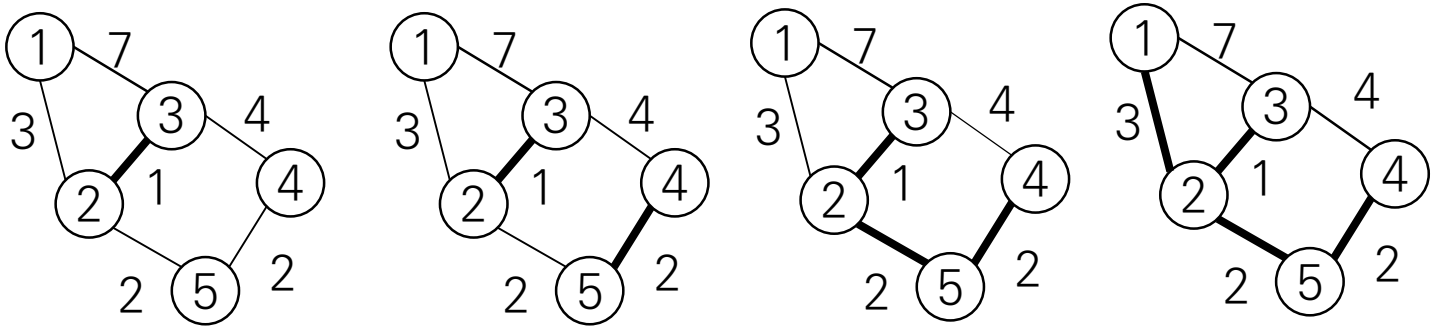
Kruskal 알고리즘



가중치에 대해 오름차순 정렬

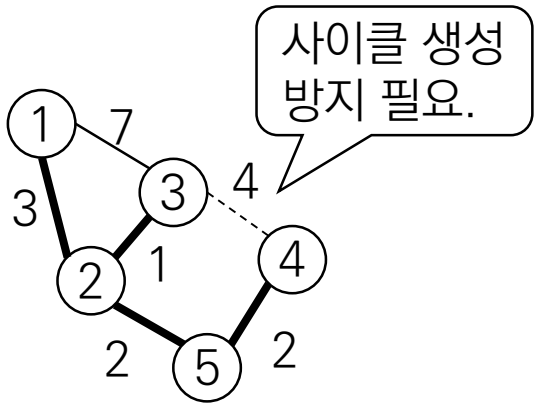


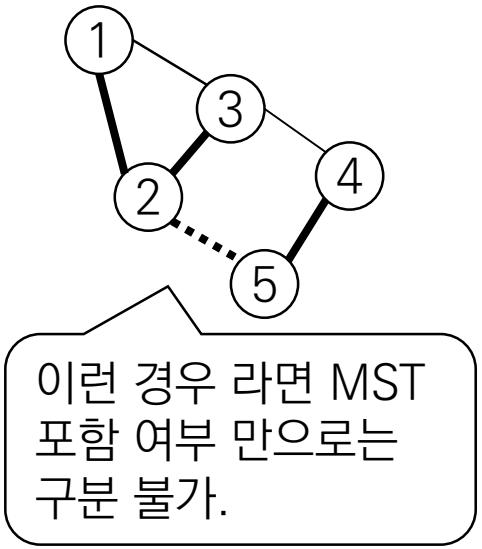
차례대로 선택



- 추가로 고려할 부분.

```
3 2 1
5 4 2
2 5 2
1 2 3
3 4 4
1 3 7
```





Kruskal의 경우 대표 원소 관리가 필요.

노드	1	2	3	4	5	6	7
대표	**1**	1	1	**4**	4	6	7

2의 대표 원소와 5의 대표원소가 다르므로 연결 가능.

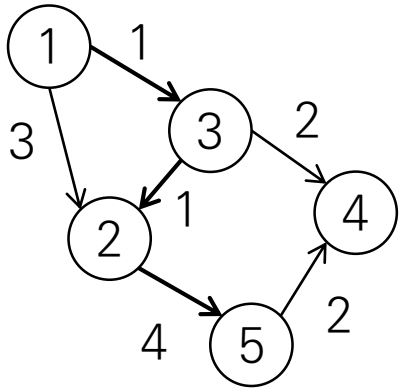
연습

- 1번부터 N번까지의 노드로 구성된 그래프에서 MST의 가중치의 합을 출력하시오. N과 간선의 수, 간선의 양쪽 노드 번호와 가중치가 주어진다.

```
5 6
1 2 3
1 3 7
3 2 1
3 4 4
2 5 2
5 4 2
```

최소 비용

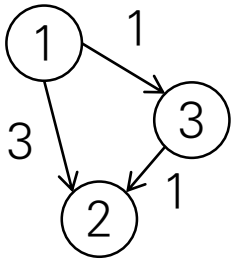
- 한 노드에서 다른 노드에 도착하는 비용 계산.
  - 모든 노드를 방문할 필요는 없는 경우.
  - 이미 비용이 계산된 노드도 다른 경로로 접근하면 비용이 감소할 수 있음.
  - 비용이 계산된 노드에 대한 중복 연산을 줄여야 함.



1에서 5로 가는 비용
3, 2를 거쳐가면 최소.
2만 거치면 오히려 비용이 증가.

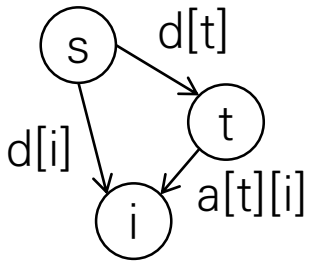
▪ 다익스트라 (Dijkstra) 알고리즘

- 조건
  - 모든 노드를 거쳐갈 필요는 없음.
  - 모든 비용이 양수.
- 출발지에서 가까운 노드부터 경유지로 선택하고, 경유지를 거쳐 갈 수 있는 노드에 대해 기존의 비용과 경유하는 비용을 비교해 새 비용을 정한다.
- 모든 노드가 경유지로 고려되면 종료한다.
- 계산이 끝나면 모든 노드에 대한 최소 비용이 계산되어 있다.
- 원하는 노드까지의 비용을 출력한다.



경유지를 고려 하기 전 1에서 2로 가는 비용은 3
경유지를 거쳐 1에서 2로 가는 비용은 2

- 다익스트라 (Dijkstra) 알고리즘



출발 s.
인접행렬 a[][].
경유지로 고려되었음을 표시하는 u[].
출발에서 노드까지의 비용 d[].
경유지로 고려 안된 노드가 있으면 다음을 반복.
    고려안된 노드 중 d[t]가 최소인 노드 t를 찾아 고려됨으로 표시.
    t와 인접인 노드 i에 대해,
    출발지에서 t를 거쳐 i로 가는 비용 d[t]+a[t][i]과
    현재 i까지 가는 비용 중 작은 쪽을 작은 쪽을 새로운 d[i]로 선택.

    d[i] = min(d[i], d[t] + a[t][i])

▪ 다익스트라 (Dijkstra) 알고리즘

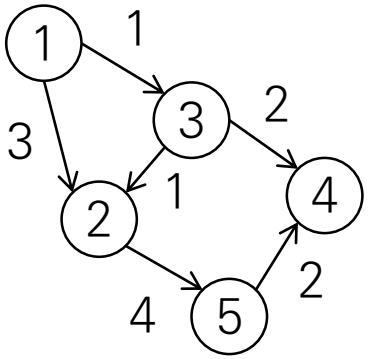
- 출발 s = 1, 도착 g = 5

경유지로 고려 여부

초기화	1	2	3	4	5
u	1	0	0	0	0

최소 거리

	1	2	3	4	5
d	0	3	1	∞	∞



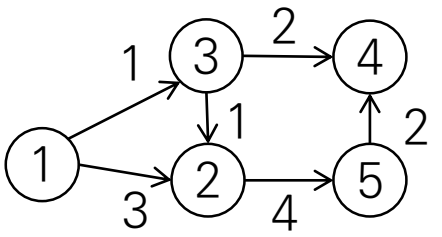

```
u[s] = 1
d[] 초기화
while( u[]에 0이 남아 있으면 )
      u[t] == 0 이고 d[t]가 최소인 노드 t를 찾는다.
      u[t] = 1
      t의 모든 인접 i에 대해
            d[i] = min(d[i], d[t] + a[t][i])
return d[g]
```

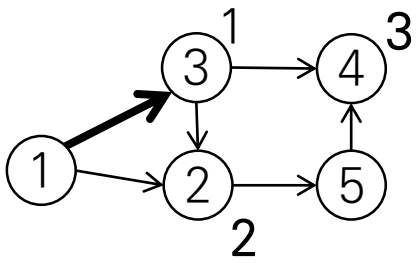
s는 경유지로 고려된 것으로 표시

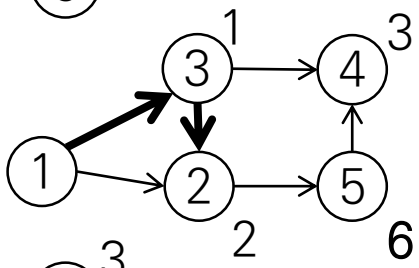
i는 경유지로 고려됨

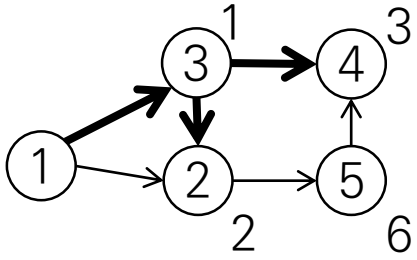
출발 s = 1, 도착 g = 5

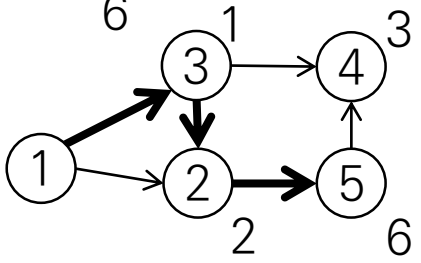
	경유지로 고려 여부					최소 거리					
초기화	1	2	3	4	5		1	2	3	4	5
u	1	0	0	0	0	d	0	3	1	∞	∞
t=3	1	2	3	4	5		1	2	3	4	5
u	1	0	1	0	0	d	0	**2**	1	**3**	∞
t=2	1	2	3	4	5		1	2	3	4	5
u	1	1	1	0	0	d	0	2	1	3	**6**
t=4	1	2	3	4	5		1	2	3	4	5
u	1	1	1	1	0	d	0	2	1	3	6
t=5	1	2	3	4	5		1	2	3	4	5
u	1	1	1	1	1	d	0	2	1	3	6











- u[t] == 0인 모든 노드를 반복해서 검색하면 시간이 오래 걸림.
- 우선순위 큐를 사용하면 속도를 높일 수 있음.

```
u[s] = 1
s에 인접한 모든 노드 인큐.
while( 큐가 비어있지 않으면 )
    t = 디큐()
    u[t] = 1
    t의 모든 인접 i에 대해
        d[i]보다 (d[t] + a[t][i])가 작으면
            d[i] = d[t]+a[t][i]
            enqueue(i)
d[g]를 리턴
```

- 앞의 문제를 BFS를 변형해서 계산할 수 있다.
  - 다른 경로에 의해 비용이 갱신된 노드를 큐에 넣는다.
  - 더 이상 비용이 갱신되는 노드가 없으면 완료.
  - 중복이 많이 발생하지만 노드의 수가 많지 않으면 속도가 빠르다.

```
BFS(s)
    enQ(s)                  // 시작점 enqueue
    V[s] = 1                // 방문 표시
    while( is_not_emptyQ())
        t = deQ()           // 비용이 갱신된 노드 dequeue
        visit(t)        // u[t]==0 && d[t]가 최소인 t 찾기
        for i : 1 -> N
            // i가 인접이면서 d[i]가 갱신이 되면
            if( A[t][i] == 1 && d[i] < d[t] + a[t][i] )
                enQ(i)
                d[i] = d[t] + a[t][i]
```