Lecture #19: Greedy Algorithm

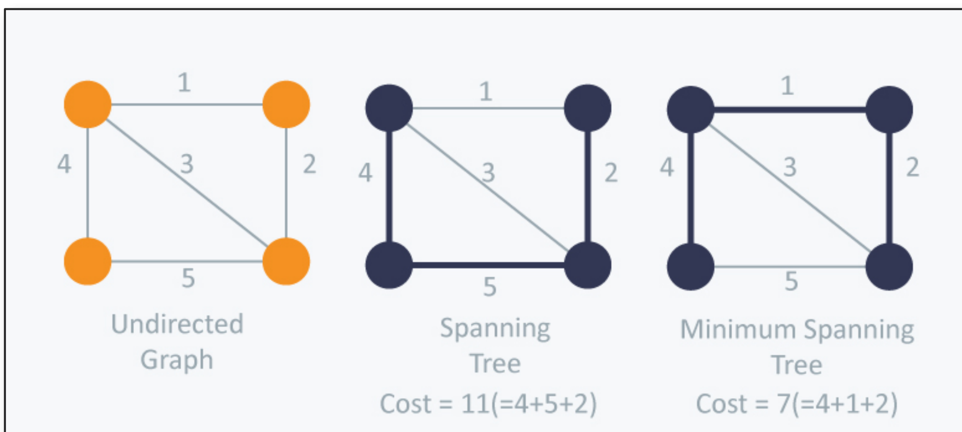
School of Computer Science and Engineering
Kyungpook National University (KNU)

Woo-Jeoung Nam

최소 신장 트리 (Minimum Spanning Tree)

- 신장 트리 **(Spanning Tree)**
  - 그래프 내에 있는 모든 정점을 연결하고 사이클이 없는 그래프를 의미 단, 신장 트리는 방향성이 없는 **(undirected)** 그래프에서만 존재
    - 트리를 만들기 위해서는 "사이클이 생기지 않아야 함"
  - **n**개의 정점이 있다면 신장 트리의 간선 수는 **n-1**개
- 최소 신장 트리 **(Minimum Spanning Tree)**
  - 최소 신장 트리**:** 간선의 가중치가 있는 그래프에서**,** 그 가중치들의 합이 가장 작은 신장 트리
  - 프림 알고리즘**(Prim's Algorithm),** 크루스칼 알고리즘**(Kruskal's Algorithm)**이 존재

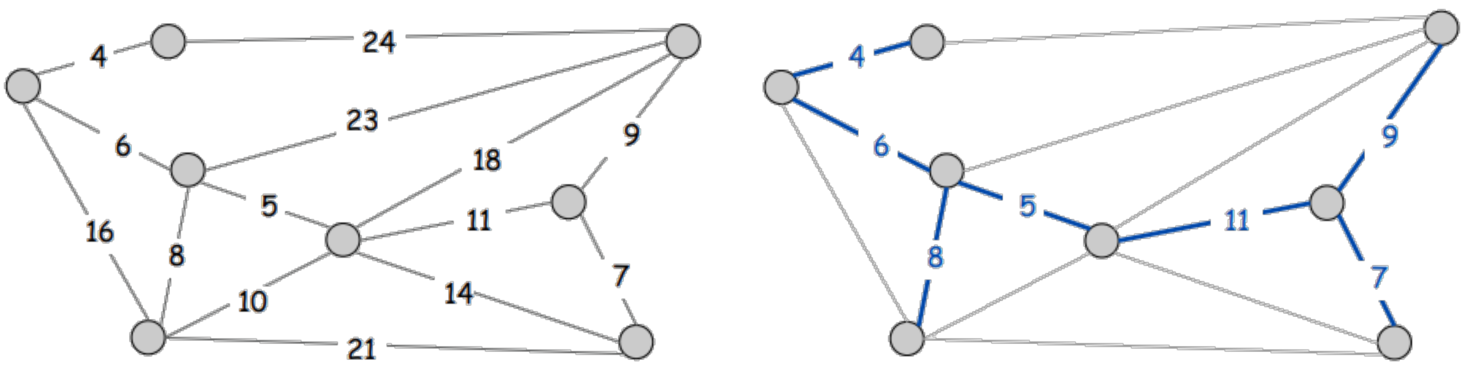
① 모든 정점 연결
② 사이클 x.



최소 신장 트리 (Minimum Spanning Tree)

- 최소 신장 트리 (Minimum spanning tree)

Minimum spanning tree. Given a connected graph $G = (V, E)$ with real-valued edge weights c_e, an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge weights is minimized.



$G = (V, E)$

$T, \ \Sigma_{e \in T} c_e = 50$

Cayley's Theorem. There are n^{n-2} spanning trees of K_n.

can't solve by brute force

최소 신장 트리 (Minimum Spanning Tree)

- 최소 신장 트리 알고리즘:

Kruskal's algorithm. Start with T = ϕ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

Prim's algorithm. Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

프림 알고리즘(Prim's Algorithm)

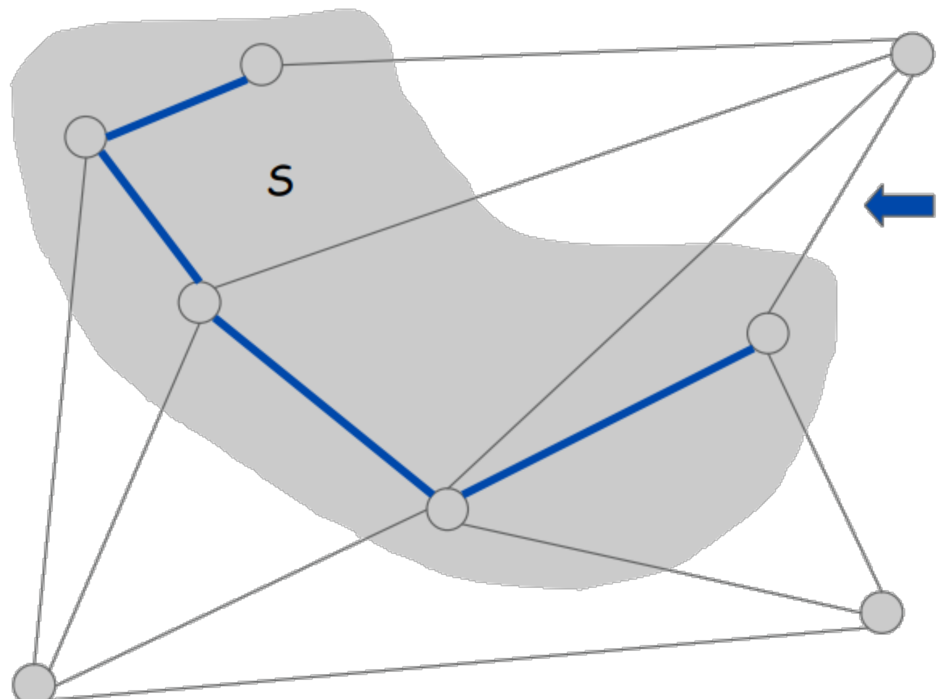
- 최소 신장 트리에 연결된 정점 주변에 있는 간선의 가중치 중 가장 작은 것을 골라 최소 신장 트리를 만드는 방법

- **1.** 그래프와 비어 있는 최소 신장 트리를 만듦
- **2.** 임의의 정점을 시작 정점으로 선택하고 최소 신장 트리의 루트 노드로 삽입
- **3.** 최소 신장 트리에 삽입된 정점과 이 정점과 인접한 정점의 가중치를 확인해서 가장 작은 가중치를 최소 신장 트리에 삽입. (최소 신장 트리에 삽입 시 사이클이 형성되지 않게 삽입)
- **4. 3**번 과정을 반복 한 후 모든 정점이 최소 신장 트리에 연결되면 종료

프림 알고리즘(Prim's Algorithm)

Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]

- Initialize S = any node.
- Apply cut property to S.
- Add min cost edge in cutset corresponding to S to T, and add one new explored node u to S.



프림 알고리즘(Prim's Algorithm)

Implementation. Use a priority queue ala Dijkstra.

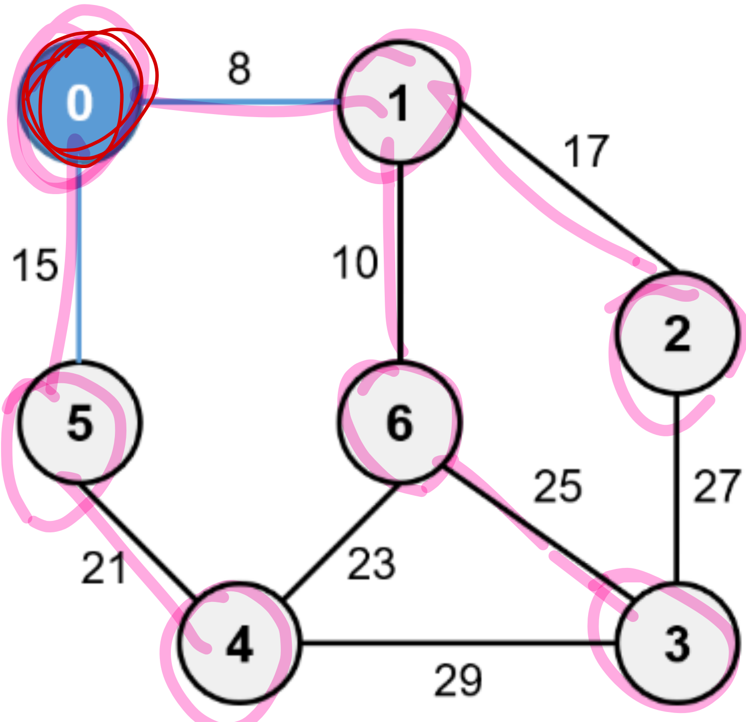
- Maintain set of explored nodes S.
- For each unexplored node v, maintain attachment cost a[v] = cost of cheapest edge v to a node in S.
- $O(n^2)$ with an array; $O(m \log n)$ with a binary heap.

```
Prim(G, c) {
   foreach (v ∈ V) a[v] ← ∞
   Initialize an empty priority queue Q
   foreach (v ∈ V) insert v onto Q
   Initialize set of explored nodes S ← φ

   while (Q is not empty) {
      u ← delete min element from Q
      S ← S ∪ { u }
      foreach (edge e = (u, v) incident to u)
          if ((v ∉ S) and (c_e < a[v]))
             decrease priority a[v] to c_e
}
```

프림 알고리즘(Prim's Algorithm)

- **0**부터 시작, 정점 **0** 을 최소 신장 트리에 추가

그래프



최소 신장 트리

21 25

25 35 36

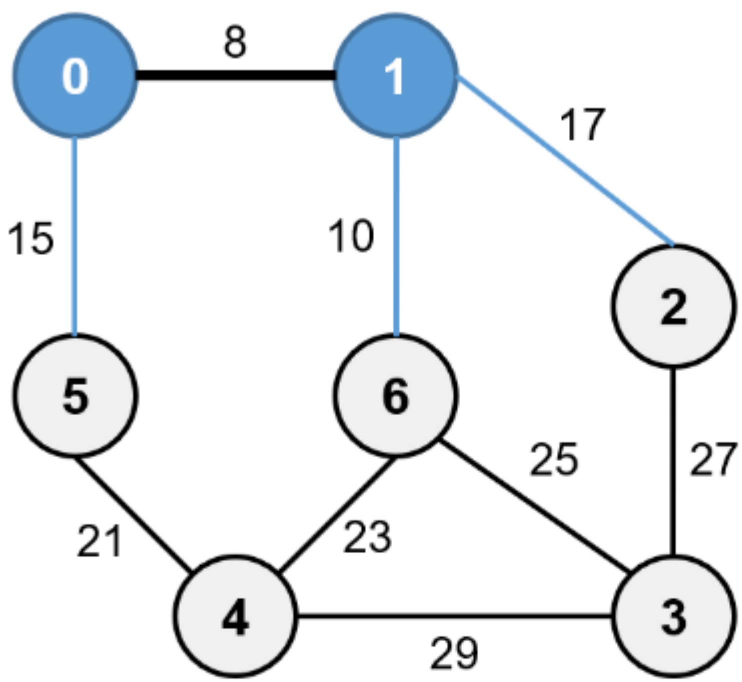
8+15+10+25

+36

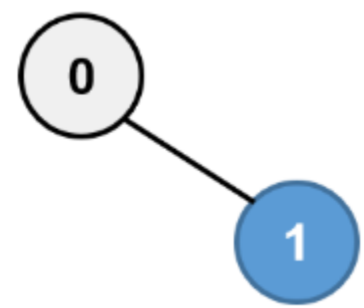
프림 알고리즘(Prim's Algorithm)

- **0**에 연결되어 있는 간선 **0-1, 0-5** 중 가중치가 가장 작은 간선은 가중치가 **8**인 **0-1** 이므로 **1**을 최소 신장 트리에 추가

그래프

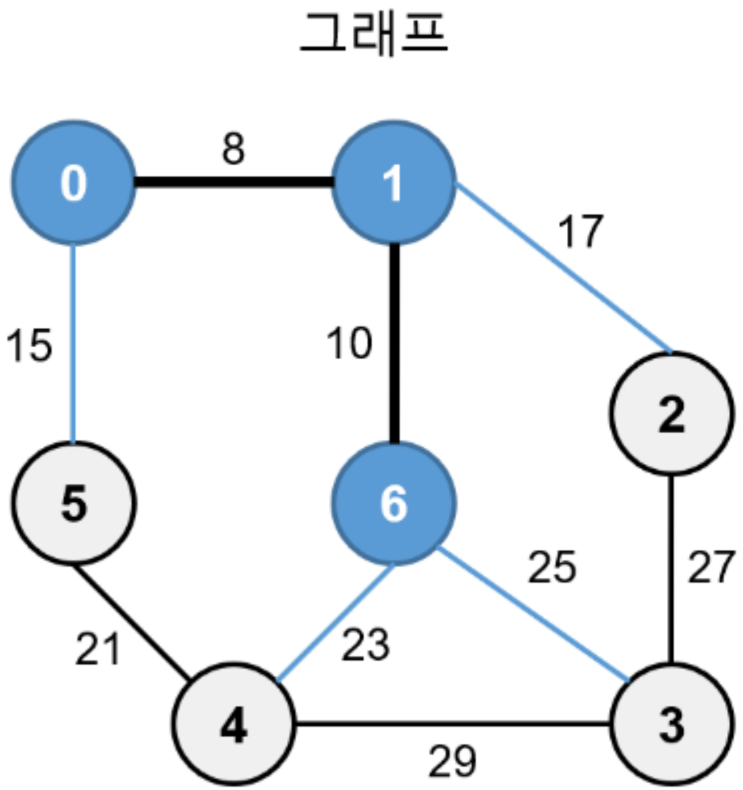


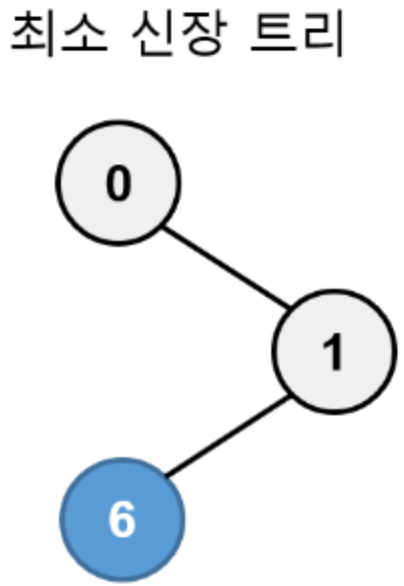
최소 신장 트리



프림 알고리즘(Prim's Algorithm)

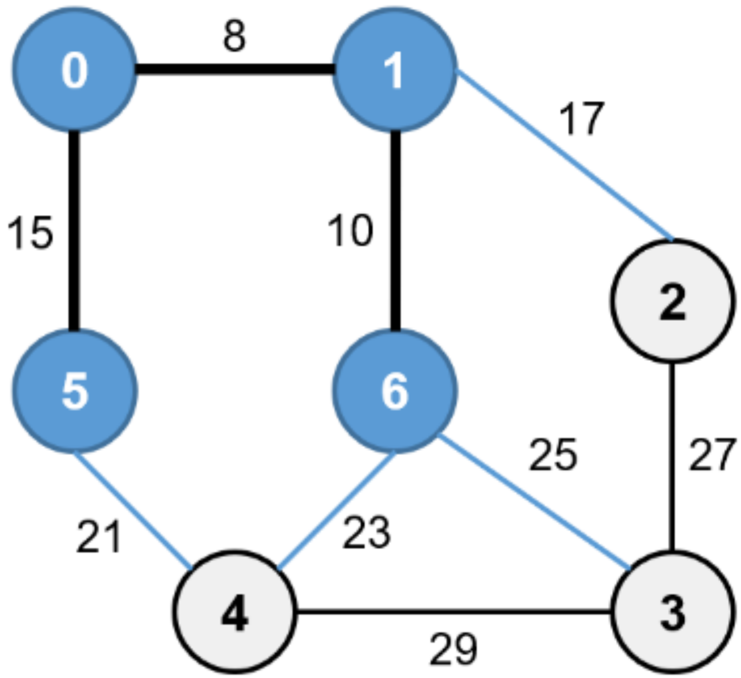
- **0, 1**에 연결되어 있는 간선 **0-5, 1-2, 1-6** 중 가중치가 가장 작은 간선은 가중치가 **10**인 **1-6** 이므로 **6**을 최소 신장 트리에 추가

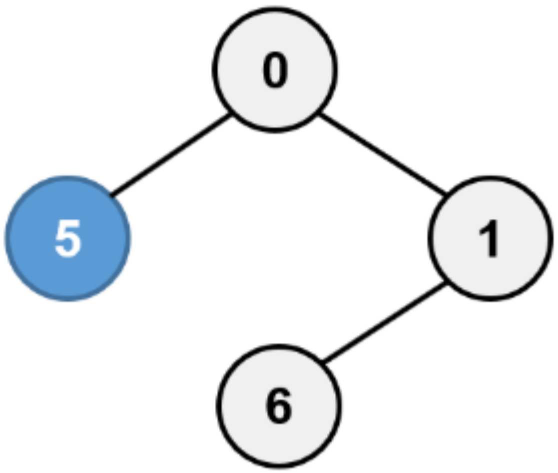




프림 알고리즘(Prim's Algorithm)

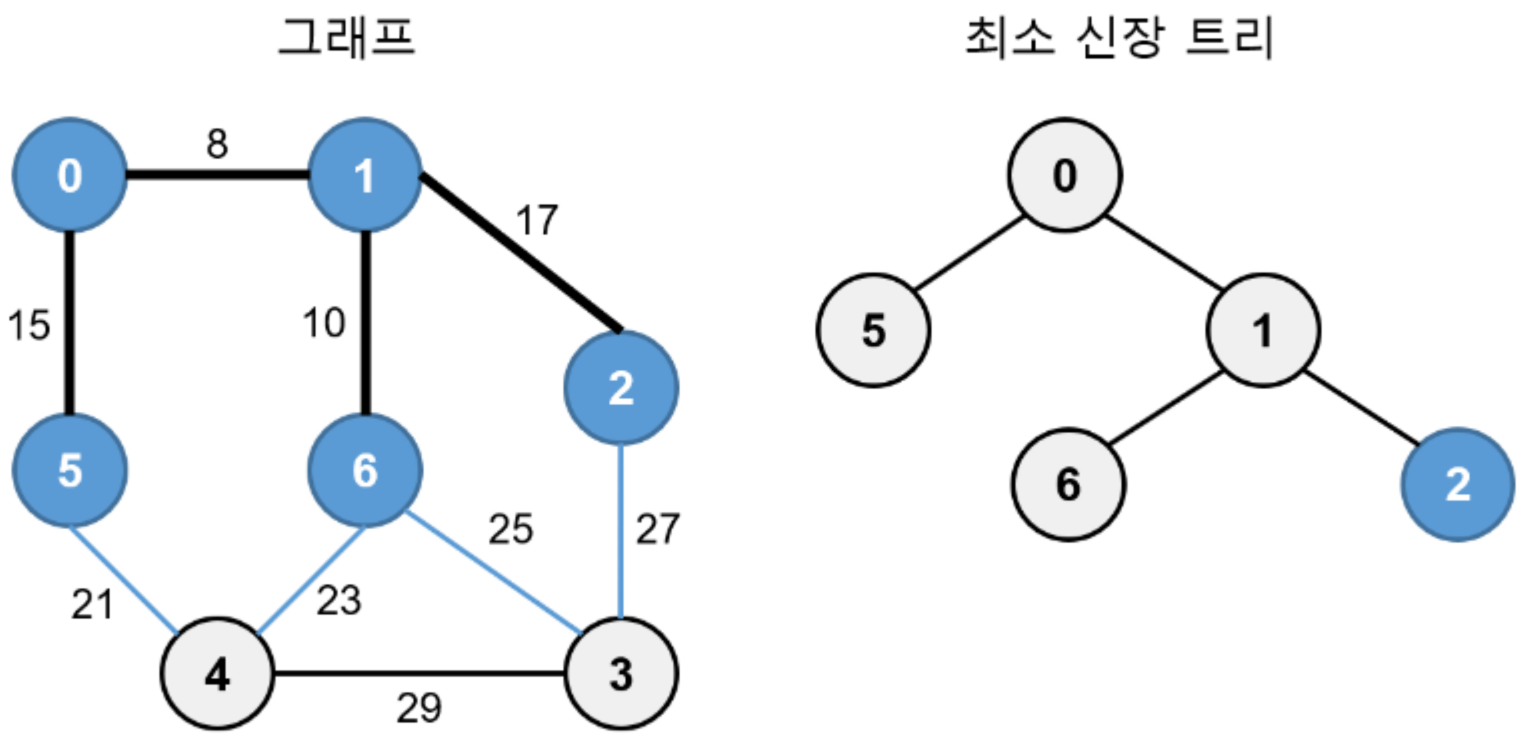
- **0, 1, 6**에 연결되어 있는 간선 **0-5, 1-2, 6-3, 6-4** 중 가중치가 가장 작은 간선은 가중치가 **15**인 **0-5** 이므로 **5**를 최소 신장 트리에 추가합니다





프림 알고리즘(Prim's Algorithm)

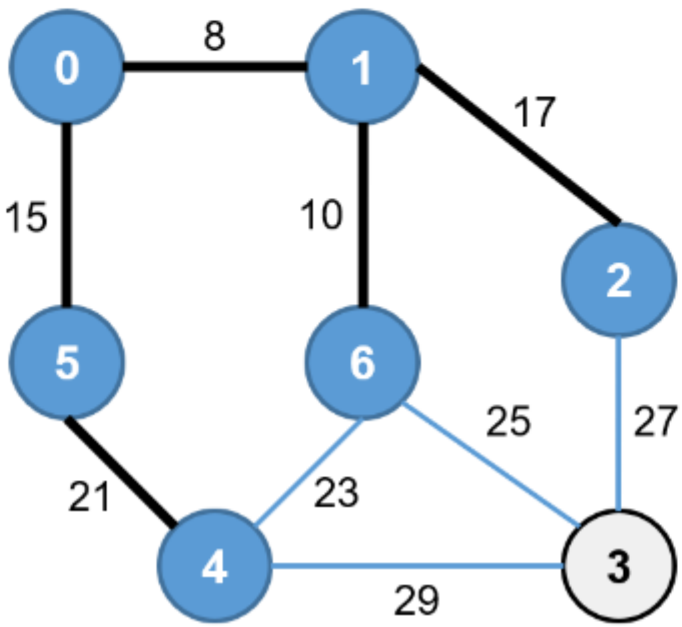
- **0, 1, 5, 6**에 연결되어 있는 간선 **1-2, 5-4, 6-3, 6-4** 중 가중치가 가장 작은 간선은 가중치가 **17**인 **1-2** 이므로 **2**를 최소 신장 트리에 추가합니다.



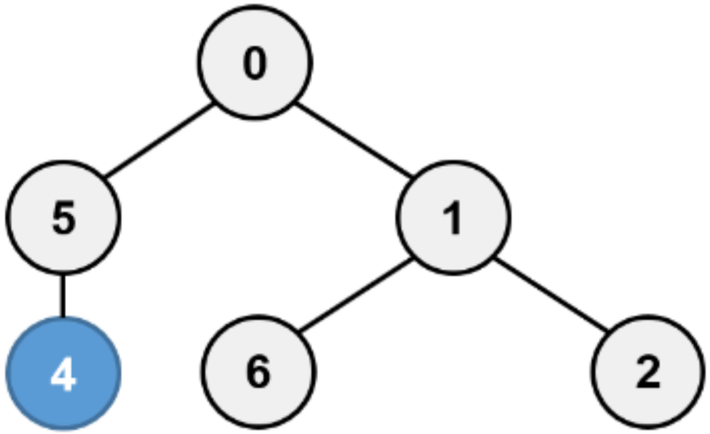
프림 알고리즘(Prim's Algorithm)

- **0, 1, 2, 5, 6**에 연결되어 있는 간선 **2-3, 5-4, 6-3, 6-4** 중 가중치가 가장 작은 간선은 가중치가 **21**인 **5-4** 이므로 **4**를 최소 신장 트리에 추가합니다.

그래프



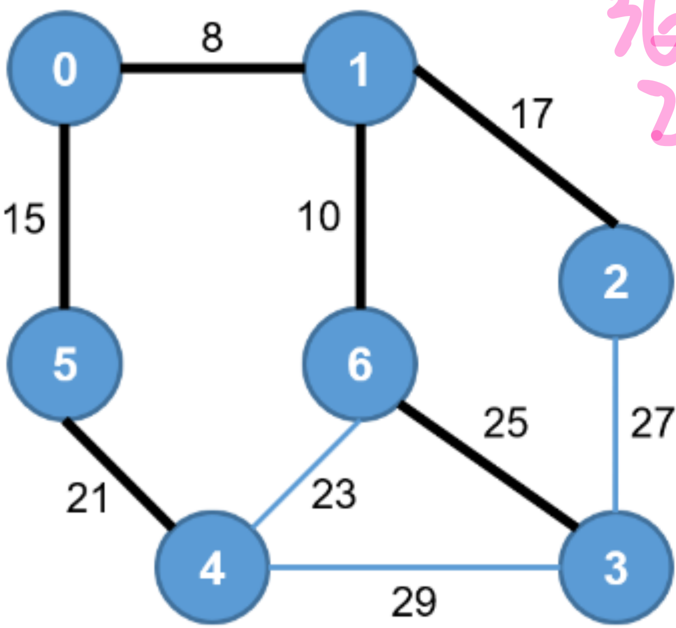
최소 신장 트리



프림 알고리즘(Prim's Algorithm)

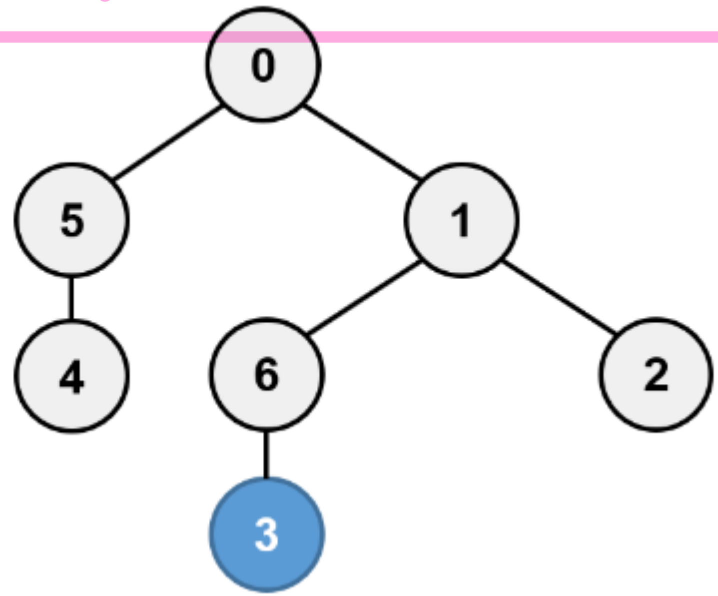
- **0, 1, 2, 4, 5, 6**에 연결되어 있는 간선 **2-3, 4-3, 6-3, 6-4** 중 가중치가 가장 작은 간선은 가중치가 **23**인 **6-4**이지만 **6-4**를 선택하면 사이클이 형성 되므로 최소 신장 트리에 추가하지 않고, **23** 다음으로 작은 가중치가 **25**인 **3**을 최소 신장 트리에 추가합니다.

그래프



35
36
25

최소 신장 트리



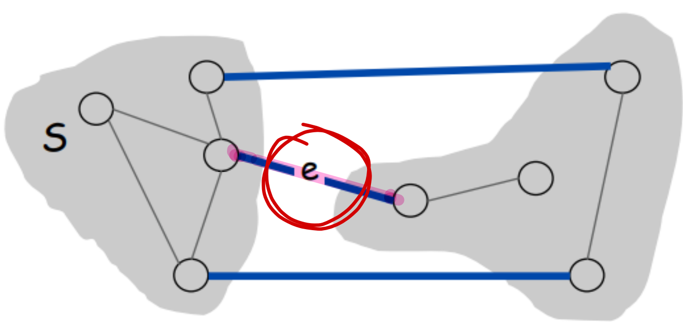
크루스칼 알고리즘

■ 크루스칼 알고리즘

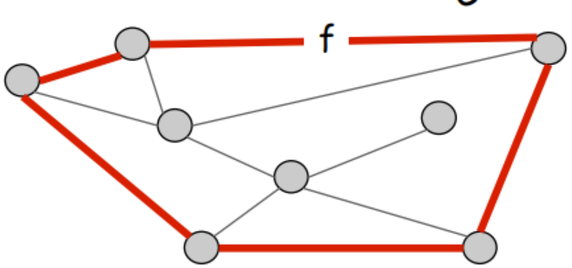
Simplifying assumption. All edge costs c_e are distinct.

Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST contains e.

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C. Then the MST does not contain f.



e is in the MST



f is not in the MST

크루스칼 알고리즘

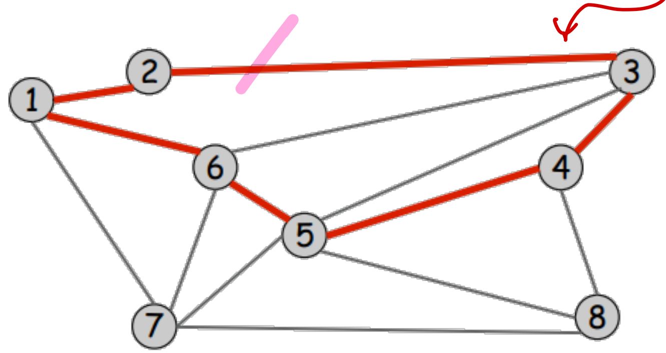
- **Cut: edge의 부분 집합**

그래프의 정점 집합을 두 개로 분할한 것을 말함
분할된 집합 → (S, V-S)

  - ➢ 단, cut에 해당하는 **edge**를 삭제하면, **2**개 혹은 **2**개 이상의 그래프로 분리 됨

사이의 모든 간선들 cut-set

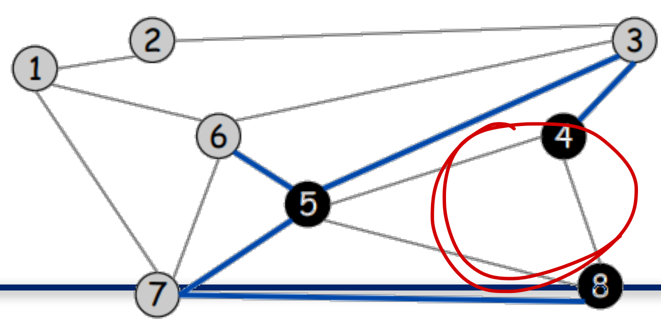
Cycle. Set of edges the form a-b, b-c, c-d, …, y-z, z-a.



만약, 2-3이 최대면 MST에 포함되지 않는다.

Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1

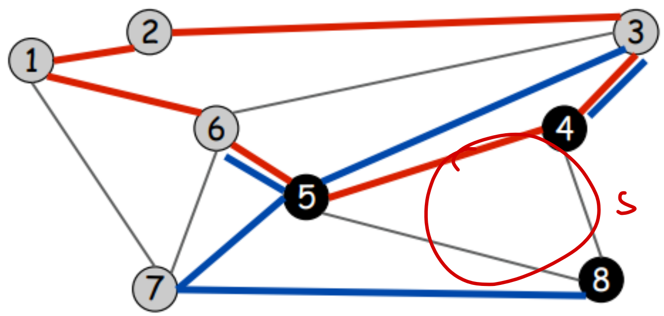
Cutset. A cut is a subset of nodes S. The corresponding cutset D is the subset of edges with exactly one endpoint in S.



Cut S = { 4, 5, 8 }
Cutset D = 5-6, 5-7, 3-4, 3-5, 7-8

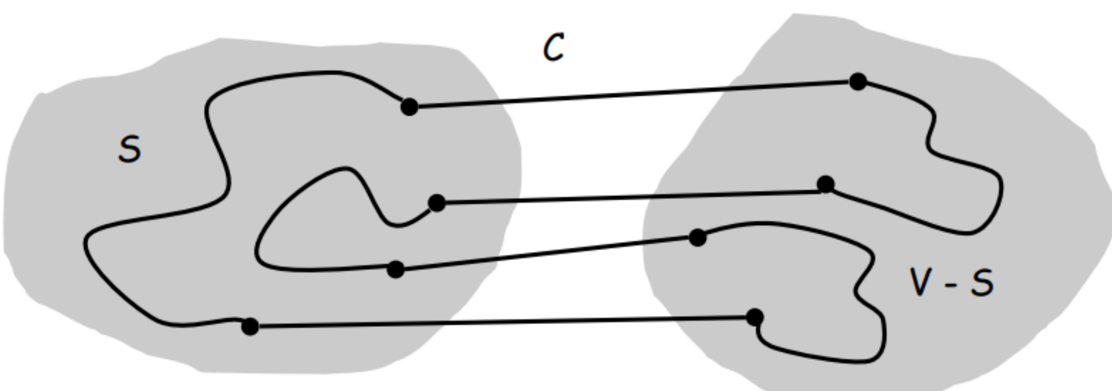
Cycle-Cut Intersection

Claim. A cycle and a cutset intersect in an even number of edges.



Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
Cutset D = 3-4, 3-5, 5-6, 5-7, 7-8
Intersection = 3-4, 5-6

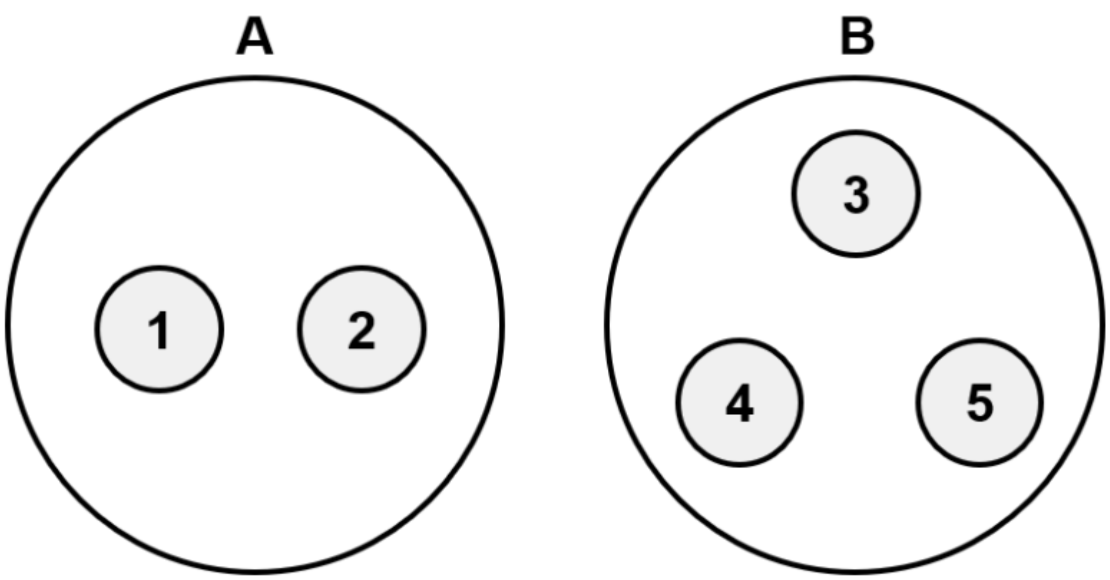
Pf. (by picture)



크루스칼 알고리즘

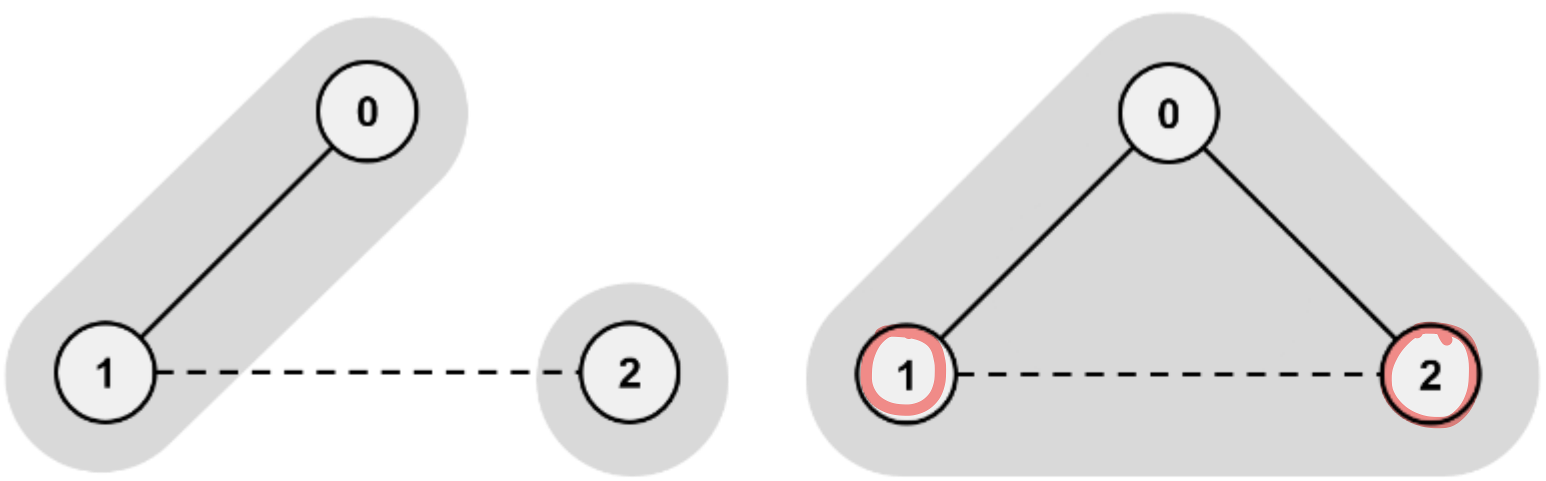
- 그래프 내의 모든 간선의 가중치를 확인하고 가장 작은 가중치부터 확인해서 최소 신장 트리를 만드는 방법
  - 그래프 내의 모든 간선의 가중치를 오름차순으로 정렬
  - 오름차순으로 정렬된 가중치를 순회하면서 최소 신장 트리에 삽입 **(**최소 신장 트리에 삽입 시 사이클이 형성되지 않게 삽입**)**

- 분리집합**(Disjoint set)** 사용



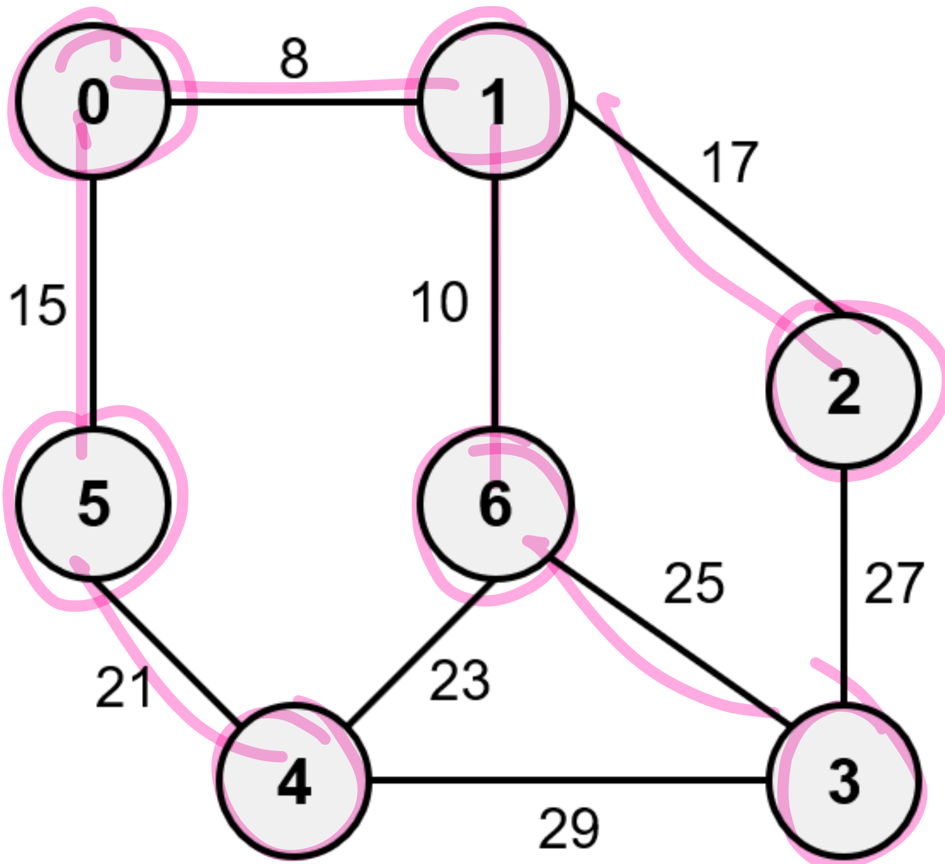
크루스칼 알고리즘

- **Cycle confirm** → 채택 하지 않음. spanning tree 조건 보장 X
  - ➢ **1-2**를 간선으로 연결할 경우
  - ➢ 왼쪽**:** 서로 다른 집합이기 때문에 **1-2**를 연결해도 사이클이 형성 되지 않음
  - ➢ 오른쪽**:** 이미 같은 집합에 속해 있기 때문에 **1-2**를 연결할 경우 사이클을 형성



크루스칼 알고리즘

■ 그래프 내의 모든 간선을 가중치를 기준으로 오름차순으로 정렬한 후, 각 정점 별로 분리 집합 생성

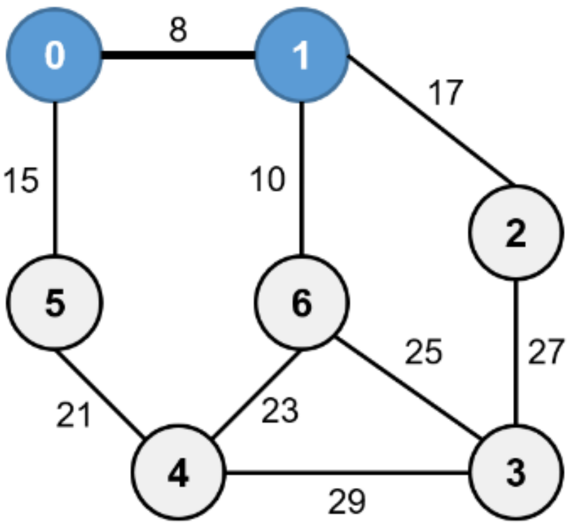


간선	가중치
0-1	8
1-6	10
0-5	15
1-2	17
4-5	21
4-6	23
3-6	25
2-3	27
3-4	29

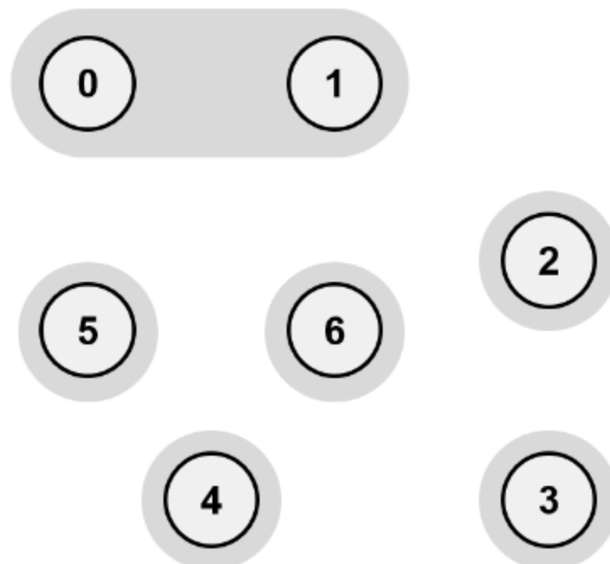
크루스칼 알고리즘

■ 첫번째로 정렬된 가중치가 **8**인 **0-1**은 서로 다른 집합이므로 최소 신장 트리에 추가하고 간선으로 연결합니다. 그리고 집합 **{0}**, **{1}**을 합집합 연산을 해서 하나의 분리 집합으로 만듭니다.

그래프



집합

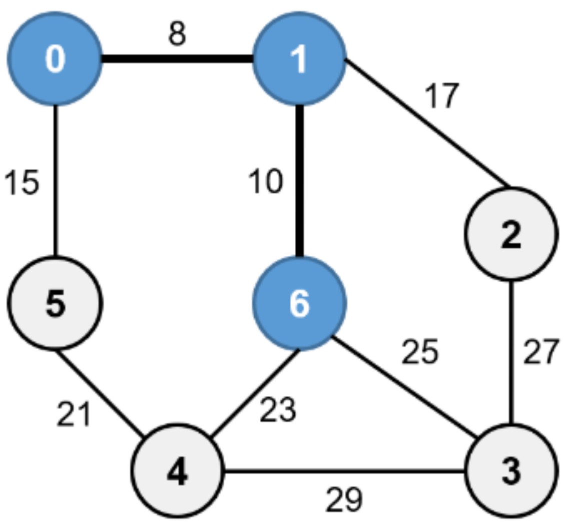


간선	가중치
0-1	8
1-6	10
0-5	15
1-2	17
4-5	21
4-6	23
3-6	25
2-3	27
3-4	29

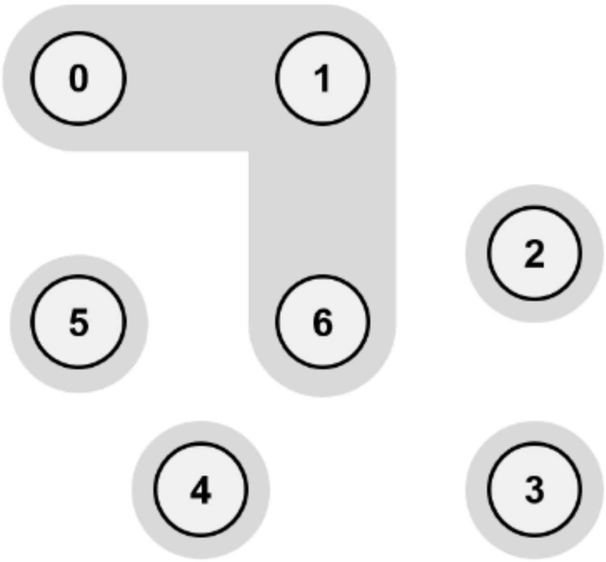
크루스칼 알고리즘

- 다음으로 가중치가 **10**인 **1-6**은 서로 다른 집합이므로 최소 신장 트리에 추가하고 간선으로 연결합니다. 그리고 집합 **{0, 1}, {6}**을 합집합 연산을 해서 하나의 분리 집합으로 만듭니다.

그래프



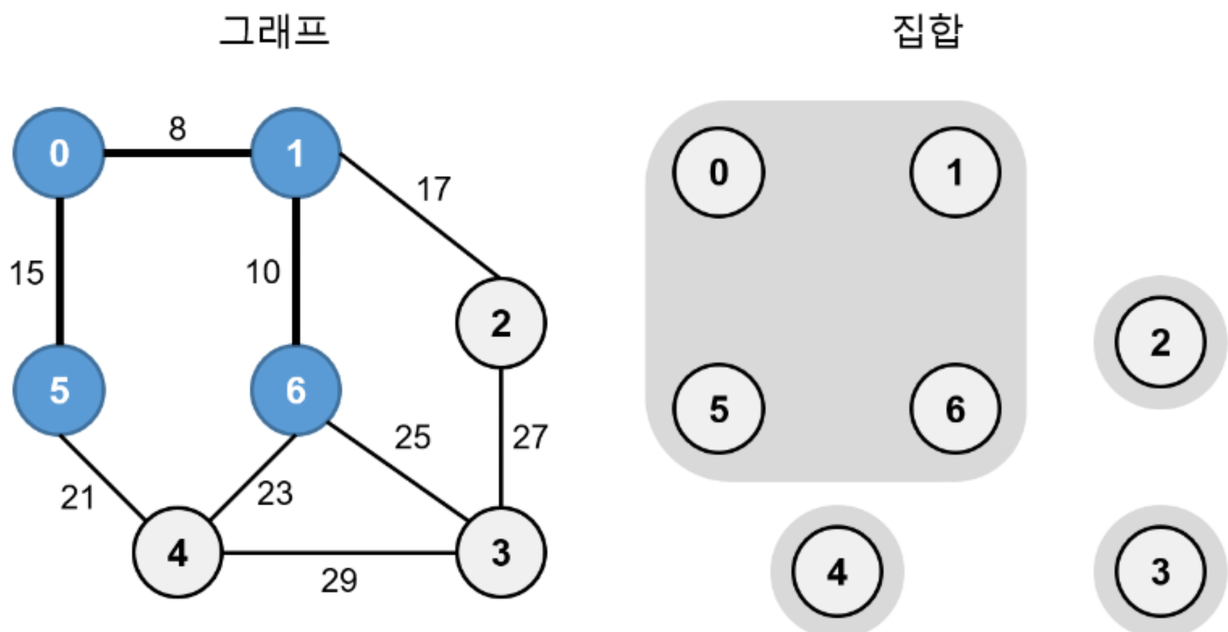
집합



간선	가중치
0-1	8
1-6	10
0-5	15
1-2	17
4-5	21
4-6	23
3-6	25
2-3	27
3-4	29

크루스칼 알고리즘

- 가중치가 **15**인 **0-5**는 서로 다른 집합이므로 최소 신장 트리에 추가하고 간선으로 연결합니다. 그리고 집합 **{0, 1, 6}, {5}**을 합집합 연산을 해서 하나의 분리 집합으로 만듭니다.

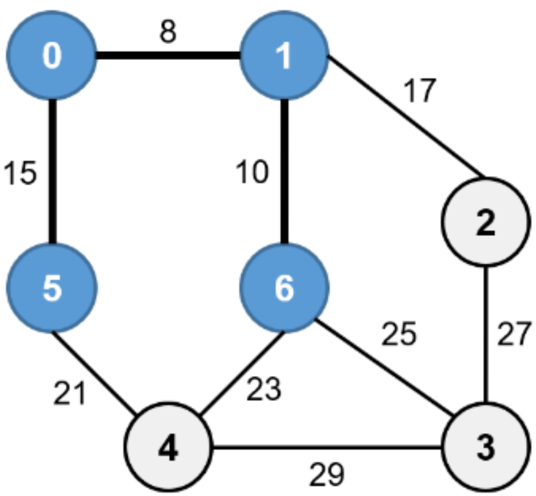


간선	가중치
0-1	8
1-6	10
0-5	15
1-2	17
4-5	21
4-6	23
3-6	25
2-3	27
3-4	29

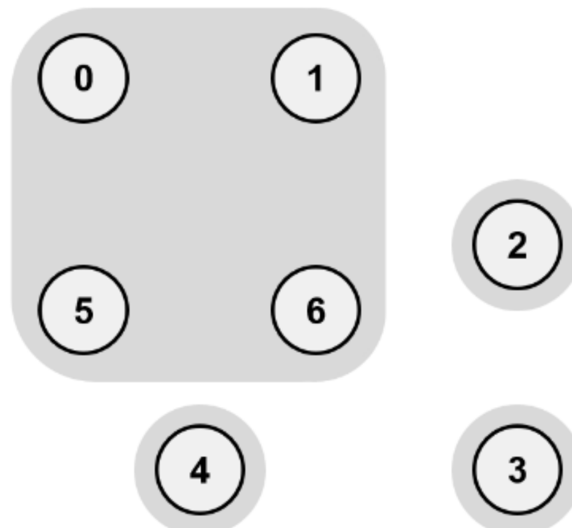
크루스칼 알고리즘

- 가중치가 **17**인 **1-2**는 서로 다른 집합이므로 최소 신장 트리에 추가하고 간선으로 연결합니다. 그리고 집합 **{0, 1, 5, 6}, {2}**을 합집합 연산을 해서 하나의 분리 집합으로 만듭니다

그래프



집합

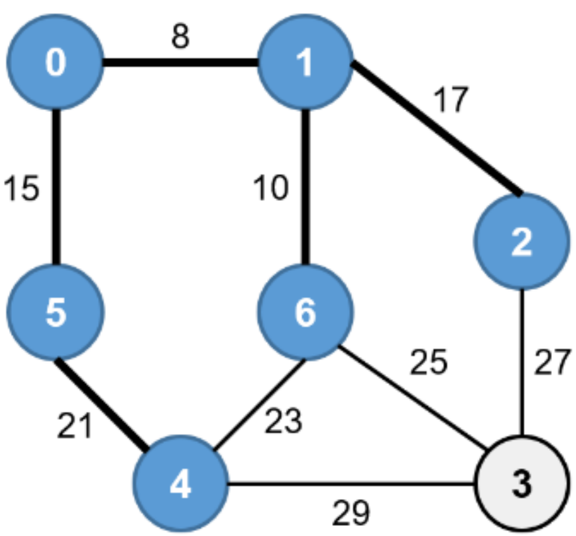


간선	가중치
0-1	8
1-6	10
0-5	15
1-2	17
4-5	21
4-6	23
3-6	25
2-3	27
3-4	29

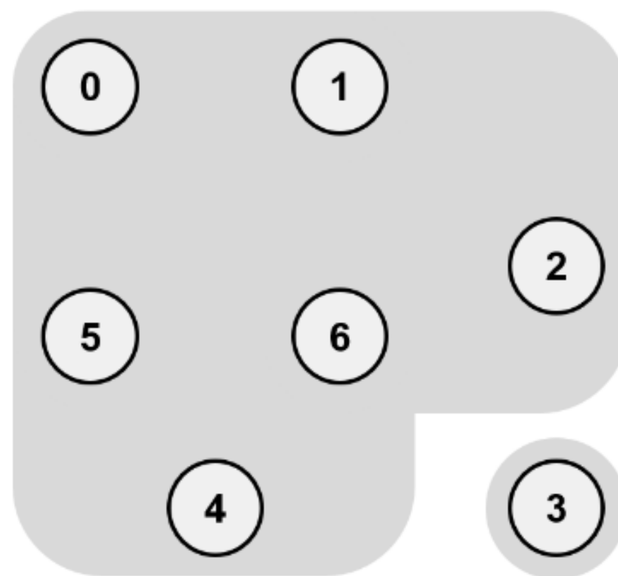
크루스칼 알고리즘

- 가중치가 **21**인 **4-5**는 서로 다른 집합이므로 최소 신장 트리에 추가하고 간선으로 연결합니다. 그리고 집합 **{0, 1, 2, 5, 6}, {4}**을 합집합 연산을 해서 하나의 분리 집합으로 만듭니다.

그래프



집합



간선	가중치
0-1	8
1-6	10
0-5	15
1-2	17
4-5	21
4-6	23
3-6	25
2-3	27
3-4	29

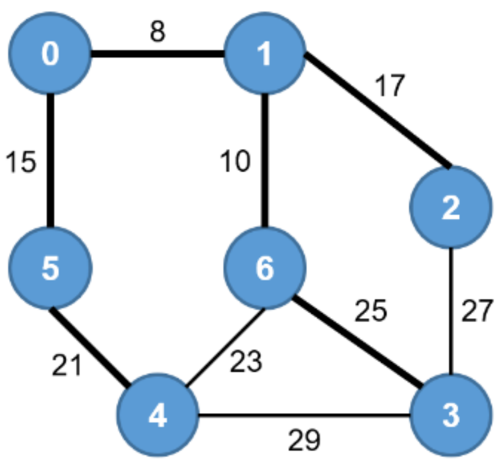
크루스칼 알고리즘

- 가중치가 **23**인 **4-6**은 서로 같은 집합에 속해 있으므로 간선을 연결하면 사이클을 형성합니다. 따라서 다음 가중치를 확인합니다.

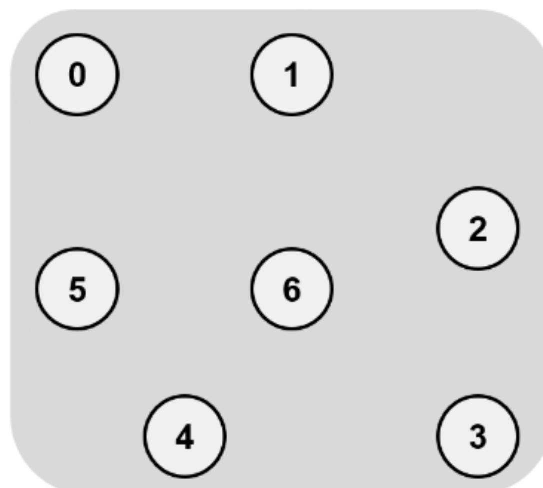
- 가중치가 **25**인 **3-6**은 서로 다른 집합이므로 최소 신장 트리에 추가하고 간선으로 연결합니다. 그리고 집합 **{0, 1, 4, 5, 6}**, **{3}**을 합집합 연산을 해서 하나의 분리 집합으로 만듭니다.

union-find.

그래프



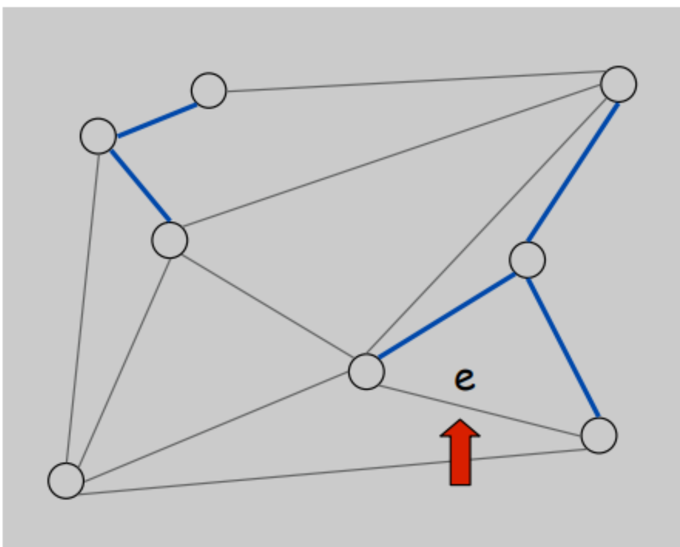
집합



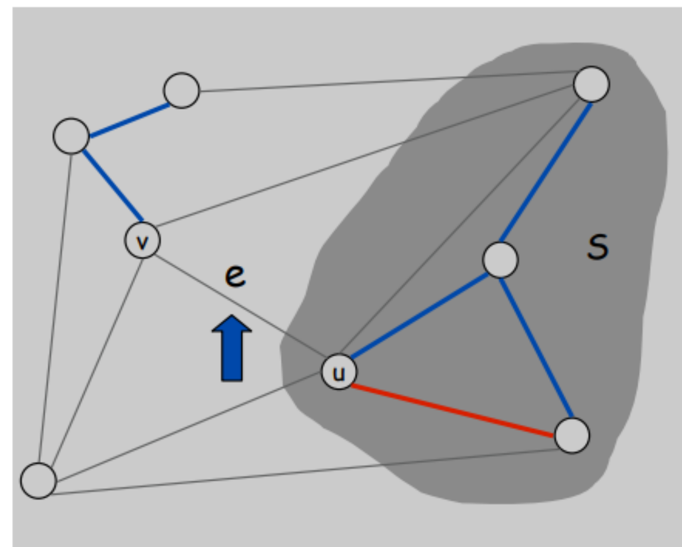
간선	가중치
0-1	8
1-6	10
0-5	15
1-2	17
4-5	21
4-6	23
3-6	25
2-3	27
3-4	29

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.
- Case 1: If adding e to T creates a cycle, discard e according to cycle property.
- Case 2: Otherwise, insert e = (u, v) into T according to cut property where S = set of nodes in u's connected component.



Case 1



Case 2

최소 신장 트리 (Minimum Spanning Tree)

Implementation. Use the union-find data structure.

- Build set T of edges in the MST.
- Maintain set for each connected component.
- $O(m \log n)$ for sorting and $O(m\ \underbrace{\alpha\ (m, n)}_{\text{essentially a constant}})$ for union-find.

$m \le n^2 \Rightarrow \log m$ is $O(\log n)$

```
Kruskal(G, c) {
   Sort edges weights so that c₁ ≤ c₂ ≤ ... ≤ cₘ.
   T ← φ

   foreach (u ∈ V) make a set containing singleton u

   for i = 1 to m          are u and v in different connected components?
      (u,v) = eᵢ
      if (u and v are in different sets) {
         T ← T ∪ {eᵢ}
         merge the sets containing u and v
      }                    merge two components
   return T
}
```

Scheduling

- **Interval Scheduling**
  - 시작시각과 종료시각이 존재하는 **job**이 여러개 있을 때, 시간이 겹치는 **job**은 **overlap**될 수 없다. 이 때, 최대로 실행할 수 있는 **job subset**을 구하는 문제 → 최대크 몇

- **Interval Partitioning**
  - 시작시각과 종료시각이 존재하는 **interval**이 여러개 있고, 실행할 수 있는 **room**이 여러개 있다. 최소의 **room**을 사용하면서 모든 **interval**을 실행하도록 **schedule**하는 문제

- **Scheduling to Minimze Lateness**
  - **length**와 **deadline**을 가지는 **job**이 여러개 있고 한 시점에 하나의 **job**밖에 실행하지 못한다. 그리고 **lateness**이란 **finish time - deadline**이다. 제일 큰 **lateness**의 크기를 최소화 하는 문제

room

length와 deadline

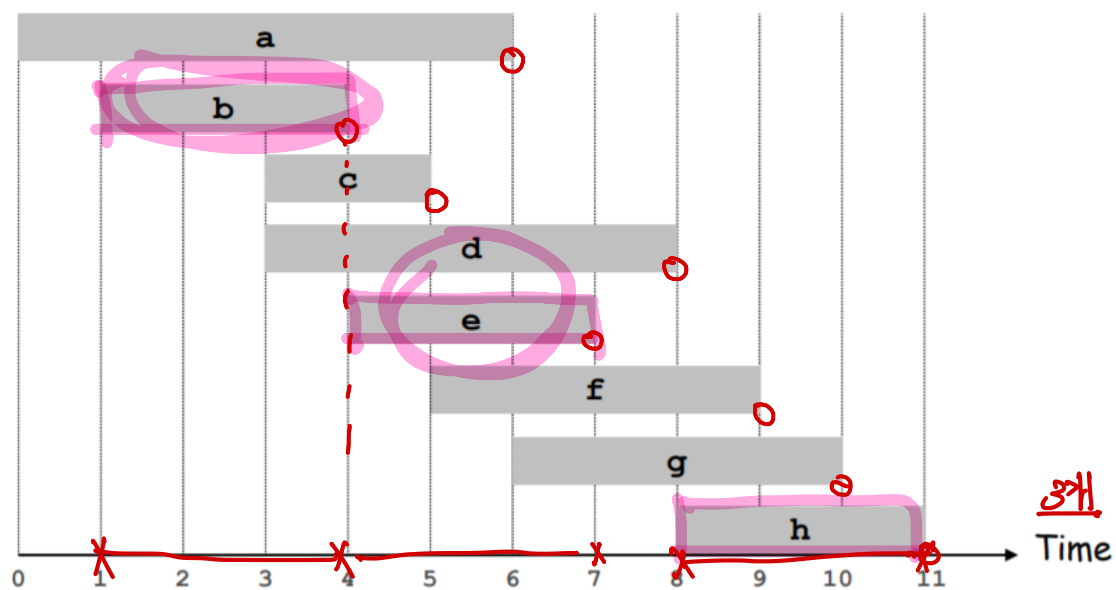
room

1개의 룸의.

https://stumash.github.io/Algorithm_Notes/greedy/intervals/intervals.html

Interval Scheduling

- **4개의 Greedy 전략**
  - ➢ **1. Start time이 빠른 job 우선**
  - ➢ **2. Finish time이 빠른 job 우선**
  - ➢ **3. Job 실행시간이 작은 job 우선**
  - ➢ **4. Conflict가 작은 job 우선**



Interval Scheduling

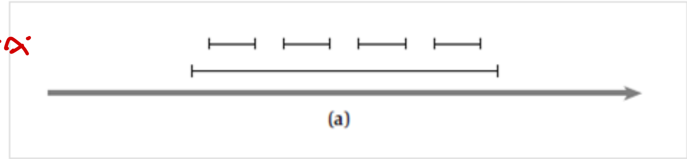
될까.

- 4개의 Greedy 전략
  - ➢ 1. Start time이 빠른 job 우선
  - ➢ 2. Finish time이 빠른 job 우선 우리의 안
  - ➢ 3. Job 실행시간이 작은 job 우선
  - ➢ 4. Conflict가 작은 job 우선
- Finish time이 빠른 순으로 실행하는 방법

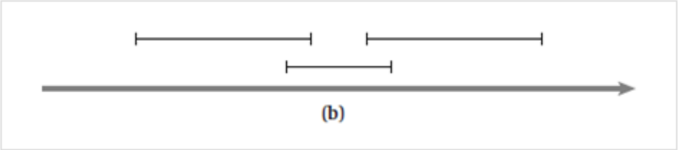
```
Sort jobs by finish times so that f₁ ≤ f₂ ≤ ... ≤ fₙ.

   set of jobs selected
A ← ϕ
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

중요 사항.

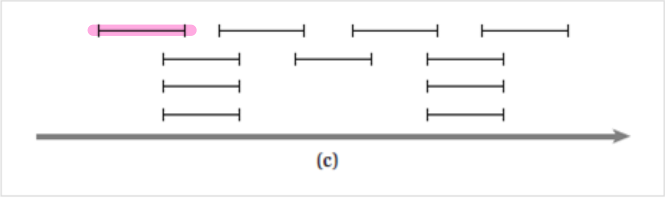
a) Earliest start time (시작시간이 빠른 순서대로)
반례는 아래와 같다.


(a)

b) Shortest interval (일의 시간이 가장 짧은 순서대로)
반례는 아래와 같다.


(b)

c) Fewest conflicts (겹치는 일이 가장 적은 순서대로)
반례는 아래와 같다.


(c)

d) Earliest finish time (끝나는시간이 빠른 순서대로)

반례를 찾을 수 없으며, 알고리즘은 다음과 같다.

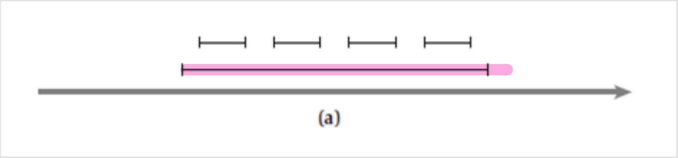
Interval Scheduling

- 4개의 Greedy 전략
  - 1. Start time이 빠른 job 우선
  - 2. Finish time이 빠른 job 우선
  - 3. Job 실행시간이 작은 job 우선
  - 4. Conflict가 작은 job 우선
- Finish time이 빠른 순으로 실행하는 방법

```
Sort jobs by finish times so that f₁ ≤ f₂ ≤ ... ≤ fₙ.

  set of jobs selected
A ← ϕ
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

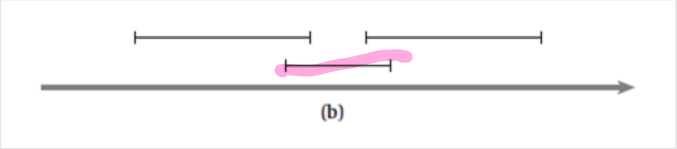
a) Earliest start time (시작시간이 빠른 순서대로)

반례는 아래와 같다.



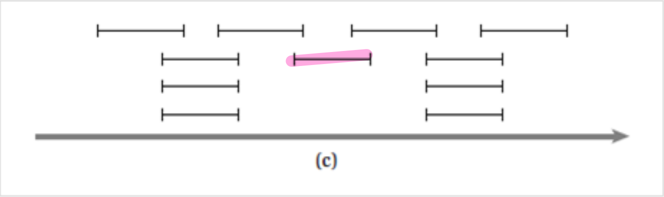
b) Shortest interval (일의 시간이 가장 짧은 순서대로)

반례는 아래와 같다.



c) Fewest conflicts (겹치는 일이 가장 적은 순서대로)

반례는 아래와 같다.



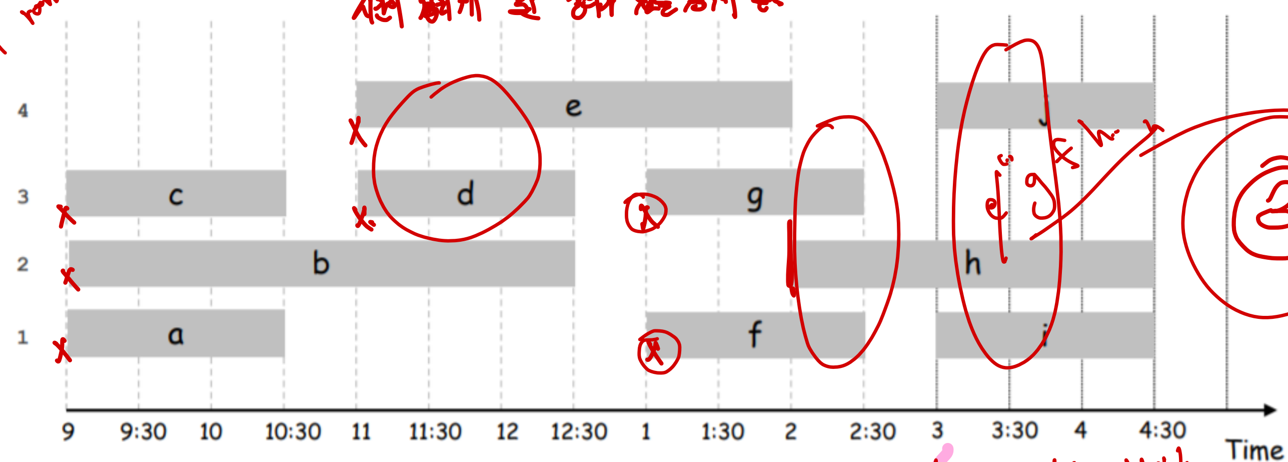
d) Earliest finish time (끝나는시간이 빠른 순서대로)

반례를 찾을 수 없으며, 알고리즘은 다음과 같다.

Interval Partitioning

- **Depth :** 어떤 시간에 겹치는 **interval**의 최대 개수
  - ➢ 아래의 예시: **depth=3**
  - ➢ **Start time**이 빠른순으로 **scheduling**

```
Sort the intervals by their starting times
Let I_1, I_2, ... I_n be the intervals in this order
for j = 1 to n {
   for each interval I_i that precedes I_j in sorted order and overlaps it {
      Exclude the classroom of I_i from consideration for I_j
   }
   if (there is any classroom in {1,2,…, d} that has net been excluded)
      Assign a non-excluded classroom to I_j
   else
      Leave I_j unlabeled
}
```



Scheduling to Minimze Lateness

- **lateness= finish time – deadline**
- **Lateness:** $l_j = \max\{0,\ f_j - d_j\}$ (단, 여기서 d_j의 경우 마감시한)
  - 주어진 모든 작업들에 대해서 스케줄링 할 때,
  - **maximum lateness** L **= max** l_j 를 최소화 할 수 있는 작업의 순서 찾기

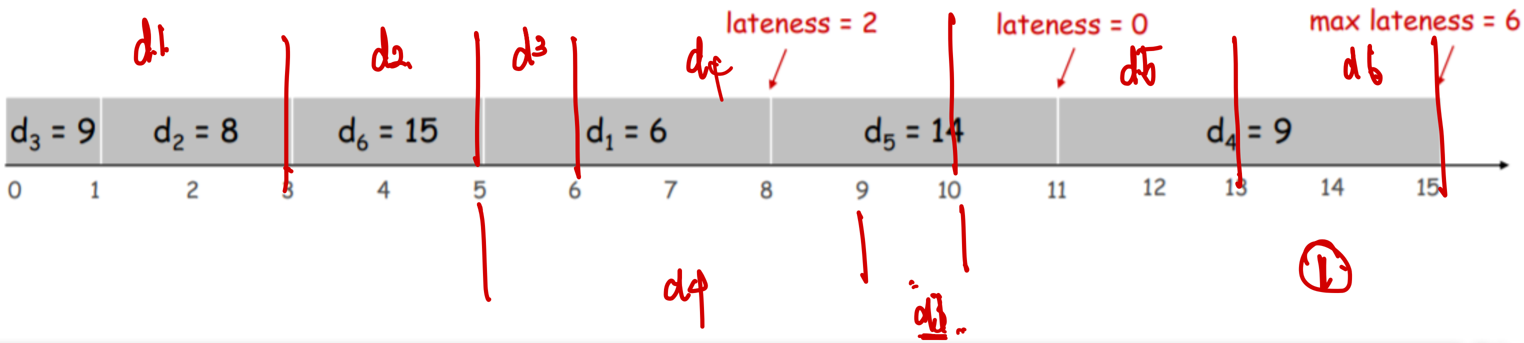
Li

$l_j = \max\{0, f_j - d_j\}$

earliest-deadline first

Ex:

	1	2	3	4	5	6	
t_j	3	2	1	4	3	2	length
d_j	6	8	9	9	14	15	deadline



Scheduling to Minimze Lateness

- **Minimizing lateness problem**
- **OPT**를 위해서 고려할 수 있는 선택지들 고민해보기:
  - **1) Shortest processing time first**
    - 처리 시간의 순서대로 (오름차순) 선택하기

처리시간

  - **2) Earliest deadline first**
    - 마감 기한이 가까운 작업부터 먼저 처리하기
  - **3) Smallest slack**
    - 여유 시간 $(dj - tj)$ 이 작은 작업부터 먼저 처리하기 (마감 – 작업에 필요한 시간)
  - 전략 선택:
    - 위의 전략들 중, 반례를 찾을 수 있는 전략들은 배제한다.

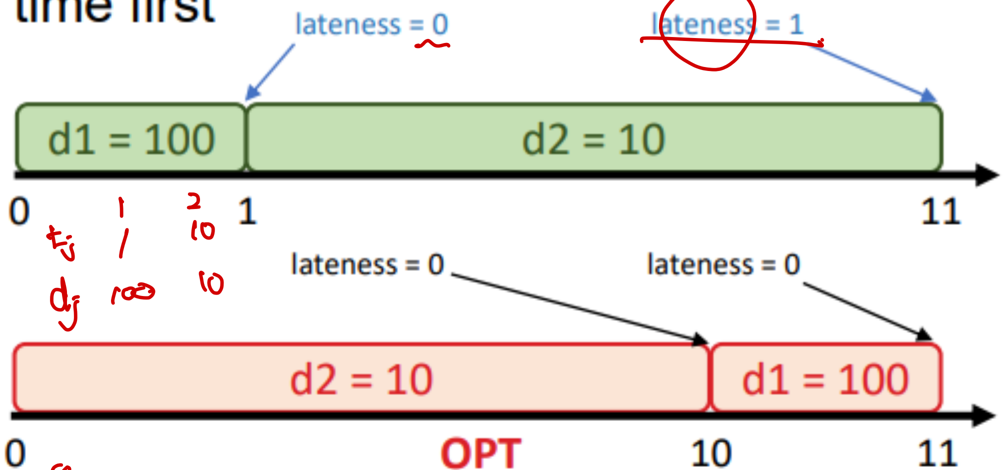
smallest slack

Scheduling to Minimze Lateness

– 1) Shortest processing time first

✓ 반례:

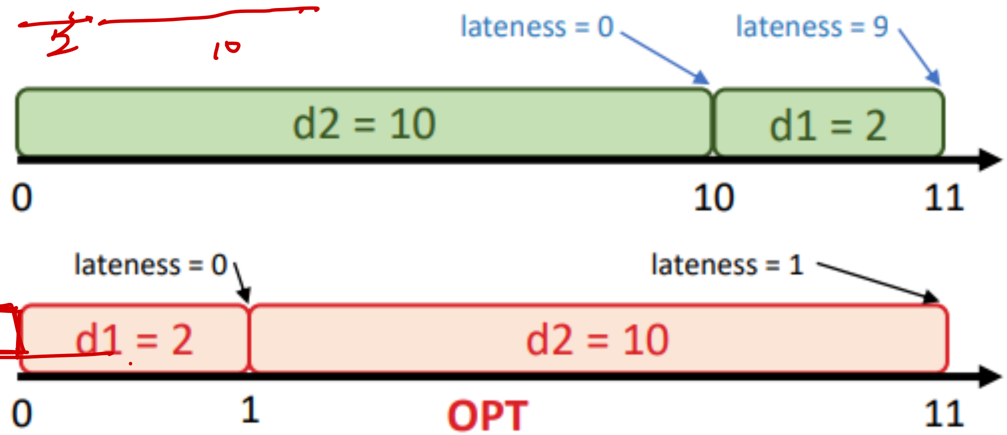
	1	2
t_j	1	10
d_j	100	10



– 3) Smallest slack

✓ 반례:

	1	2
t_j	1	10
d_j	2	10



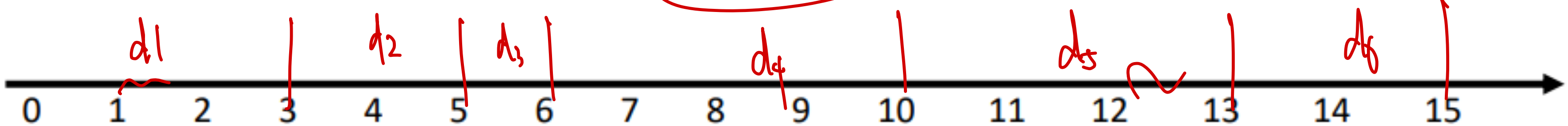
t_j	1	10
d_j	2	10

Scheduling to Minimze Lateness

- **Minimizing lateness problem** →
  - Earliest deadline first

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

lateness =




```
Sort n jobs by deadline so that d1 ≤ d2 ≤ … ≤ dn,

t ← 0
for j = 1 to n
    Assign job j to interval [t, t + tj]
    sj ← t, fj ← t + tj
    t ← t + tj
output intervals [sj, fj]
```

$[t, t+t_j]$