SAMSUNG 삼성SDS

SW 아카데미 알고리즘 특강

Contents

오늘의 원리

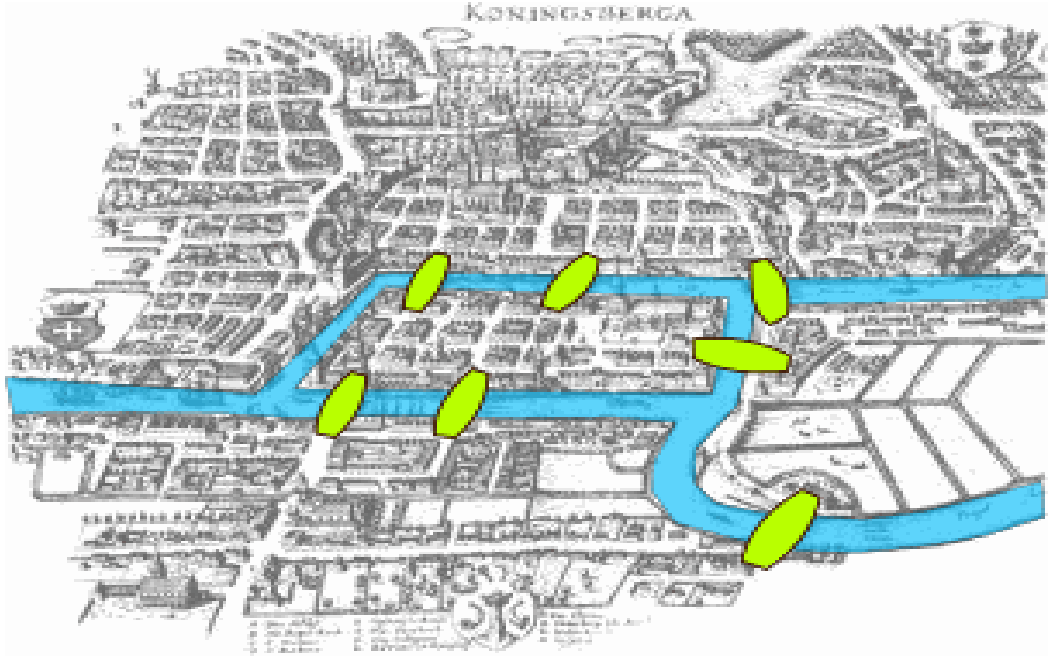
- 실제 일상생활이나 현실세계에서 풀어야 할 문제를 그래프로 추상화하여 최적의 문제해결방법을 찾아내는 연습을 한다.
- 다양한 그래프 알고리즘을 학습하고 이를 문제에 적용해 본다.

학습목표

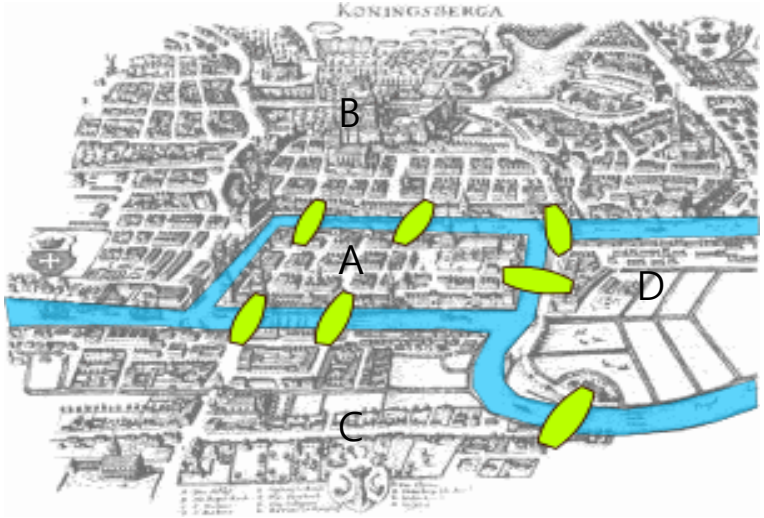
- 그래프 관련 용어와 개념을 이해하여 문제를 이해하거나 표현할 때 정확한 용어를 사용할 수 있다.
- 다양한 분야의 그래프 응용문제들의 특성을 파악하고 각 문제에 적합한 효율적인 알고리즘을 선택, 적용할 수 있다.

사전문제

1. 아래 그림과 같이 프로이센의 쾨니히스베르크에는 강이 있고 강으로 둘러싸인 섬이 하나 있다. 섬을 잇는 7개의 다리가 있는데 이때 모든 다리를 한번씩 만 건너서 출발점으로 다시 돌아 올 수 있는 방법은 무엇인가?

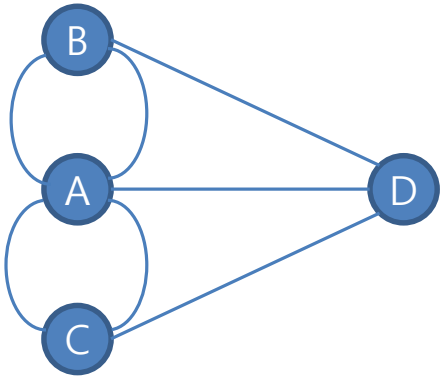


사전문제 접근법

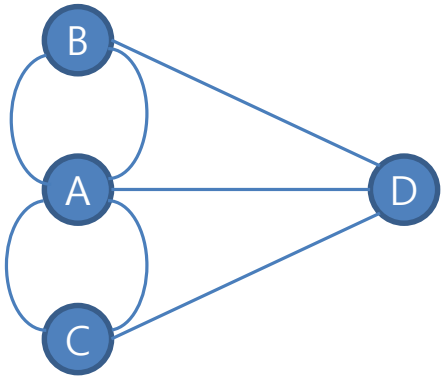


강에 의해서 구분되어지는 지역을 하나의 정점으로
각 지역을 연결하는 다리를 간선으로 생각하면
왼쪽 위의 그림은 왼쪽 아래와 같이 추상화 할 수 있다.

결국 왼쪽 아래의 그림에서 모든 정점과 모든 간선을 한 번씩만 사용하여 제자리로 돌아 올 수 있는지를 묻는 문제이고 곧 '한붓그리기' 문제라고 볼 수 있다.



사전문제 풀이법



레오나드 오일러(Leonhard Euler)는 그의 논문에서
이 문제에 대한 해법뿐 아니라 일반적인 그래프 이론을 제시하였다.

이러한 업적을 기리어 그래프 G 내의 모든 정점과 모든 에지를 포함하는 사이클을 **오일러 사이클**이라 부른다.

그래프 G가 오일러 사이클을 가진다 ⇔ 그래프 G는 연결되어 있고 각 정점은 짝수 차수이다.

결국 쾨니히스베르크의 다리건너기 문제는 홀수 차수의 정점이 2개 이상 존재하므로 오일러의 사이클이 존재 하지 않는다.

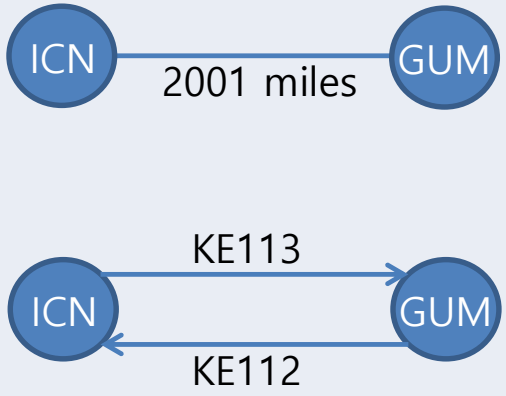
그래프(Graph)

- **그래프 정의**
  - 현실세계의 **사물**이나 **개념** 간의 **연결 관계**를 수학적 모델로 단순화 하여 표현한 것
  - 수학적 정의

정점(Vertex, Node) 집합 $V = \{ v_1, v_2, v_3, \cdots, v_n \}$ 이고, 정점간의 연결 관계들을 나타내는 간선(Edge, Link, Arc) 집합 $E = \{ (v_i, v_j) \,/\, v_i \in V, v_j \in V \} \subseteq V \times V$ 일 때,
그래프 $G = (V, E)$ 이다.

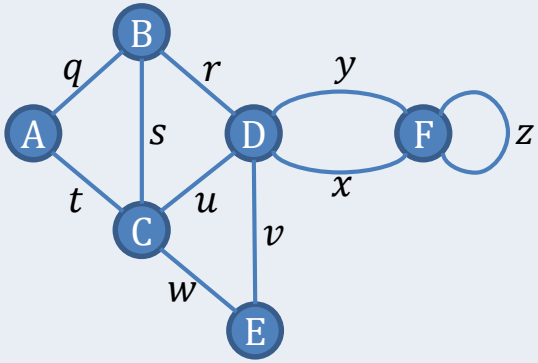
그래프 용어(1)

무향 간선 Undirected Edge	• 정점을 연결하는 간선에 방향이 존재하지 않는다. • 임의의 간선 (v_i, v_j)에 대해서 $(v_i, v_j) = (v_j, v_i)$이다.
유향 간선 Directed Edge	• 정점을 연결하는 간선에 방향이 존재한다. • 임의의 간선 (v_i, v_j)에 대해서 $(v_i, v_j) \neq (v_j, v_i)$이다.
인접 Adjacent	• 정점 v_i, v_j에 대해서 간선 $e = (v_i, v_j)$가 존재한다. ⇔ 정점 v_i와 v_j는 인접(adjacent)한다.
부속 Incident	• 정점 v_i, v_j에 대해서 간선 $e = (v_i, v_j)$가 존재한다. ⇔ 간선 e 는 정점 v_i와 v_j에 부속(incident)한다.



그래프 용어(2)

차수 **Degree**	• 정점에 부속(incident)된 간선의 수 • in-degree : 방향 그래프에서 정점에 들어오는 간선의 수 • out-degree : 방향 그래프에서 정점에서 나가는 간선의 수
다중 간선 **Multiple/ Parallel edges**	• y와 x는 다중 간선
Self-loop	• z 는 self loop
경로 **Path**	• 정점과 간선이 교대로 구성된 sequence (ex $Path(A,B) = [A,q,B]\ or\ [A,t,C,w,E,v,D,u,C,s,B]$) • 단순 경로(Simple Path) : 정점과 간선이 중복되지 않은 경로 (ex $Path(A,B) = [A,q,B]\ or\ [A,t,C,s,B]$)
회로 **Cycle**	• 시작 정점과 끝 정점이 같은 경로 (ex $Cycle(B,B) = [B,q,A,t,C,s,B]\ or\ [B,q,A,t,C,w,E,v,D,u,C,s,B]$) • 단순 회로(Simple Cycle) : 정점과 간선이 중복되지 않은 회로 (ex $Cycle(B,B) = [B,q,A,t,C,s,B]\ or\ [B,q,A,t,C,w,E,v,D,r,B]$)
연결됨 **Connected**	• 정점 v_i에서 정점 v_j로의 경로가 존재하면, 정점 v_i와 정점 v_j는 연결되어 있다고 한다.

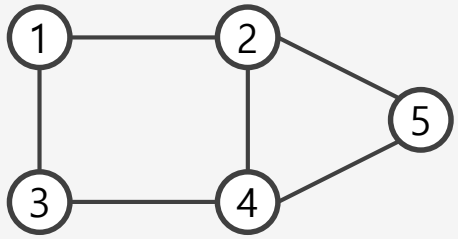


그래프 종류

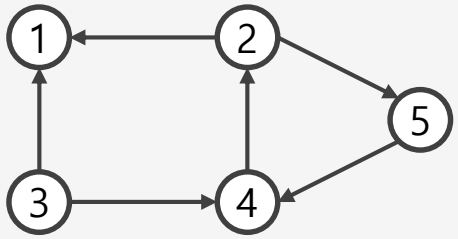
무향 그래프 **Undirected Graph**	• 무향 간선으로 이루어진 그래프
유향 그래프 **Directed Graph**	• 유향 간선으로 이루어진 그래프
가중치 그래프 **Weighted Graph**	• 가중치(or 비용)를 갖는 간선으로 이루어진 그래프
정규 그래프 **Regular Graph**	• 모든 정점이 동일한 차수를 가지는 그래프
완전 그래프 **Complete Graph**	• 임의의 두 정점 v_i, v_j에 대해서 v_i, v_j 를 잇는 간선 $Edge(v_i, v_j)$이 존재하는 그래프 • 완전그래프는 정규그래프이다. • N개의 정점을 가지는 무향 그래프의 경우 : 간선의 개수 = $\frac{1}{2} \cdot N(N-1)$ • N개의 정점을 가지는 유향 그래프의 경우 : 간선의 개수 = $N(N-1)$
연결 그래프 **Connected Graph**	• 임의의 두 정점 v_i, v_j에 대해서 경로 $Path(v_i, v_j)$가 존재하는 그래프
부분 그래프	• 어떤 그래프의 정점집합의 부분집합과 그에 속한 간선들로 이루어진 그래프 • 어떤 그래프의 간선집합의 부분집합과 그에 속한 정점들로 이루어진 그래프
트리 그래프	• 순환을 갖지 않는 연결 그래프 • 임의의 두 정점에 대해서 경로가 정확히 1개 존재한다. • 하나 이상의 정점을 가지며, 임의의 간선 e를 제거한 그래프는 연결 그래프가 아니다.

그래프 표현(1)

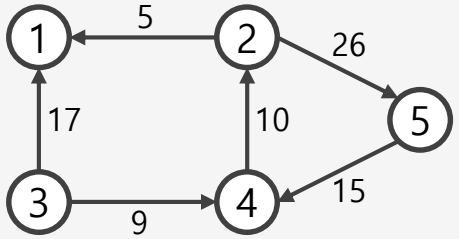
- 간선 리스트(Edge List)
  - 정점의 개수가 V개, 간선의 개수가 E개인 그래프에 대해서 $E \times 2$ (or $E \times 3$) 이차원 배열(A)에 간선 정보 저장
  - 어떤 두 정점 v_i, v_j를 연결하는 간선 $e_k = (v_i, v_j)$에 대해서 $A[k][0] = v_i$, $A[k][1] = v_j$
  - 가중치 그래프의 경우 배열의 3번째 열에 간선의 가중치를 저장



k	0	1
1	1	2
2	2	5
3	1	3
4	3	4
5	4	2
6	4	5



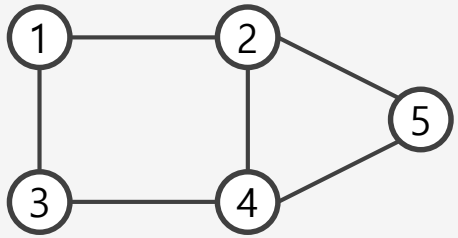
k	0	1
1	2	1
2	2	5
3	3	1
4	3	4
5	4	2
6	5	4



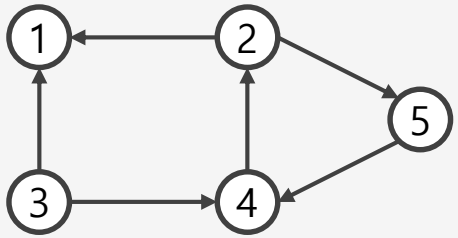
k	0	1	2
1	2	1	5
2	2	5	26
3	3	1	17
4	3	4	9
5	4	2	10
6	5	4	15

그래프 표현(2)

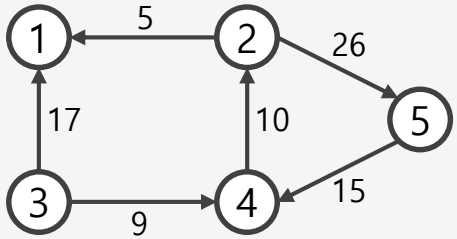
- 인접 행렬(Adjacency Matrix)
  - 정점의 개수가 V개, 간선의 개수가 E개인 그래프에 대해서 $V \times V$ 이차원 배열(A)에 그래프 정보를 저장
  - 어떤 정점 v_i, v_j를 연결하는 간선 $e = (v_i, v_j)$가 $\begin{cases} \text{존재하면} \Rightarrow A[i][j] = 1 \text{ (or 가중치)} \\ \text{존재하지 않는다면} \Rightarrow A[i][j] = 0 \end{cases}$
  - 어떤 정점 v_i, v_j가 $\begin{cases} \text{인접하면} \Rightarrow A[i][j] = 1 \text{ (or 가중치)} \\ \text{인접하지 않는다면} \Rightarrow A[i][j] = 0 \end{cases}$



i \ j	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0



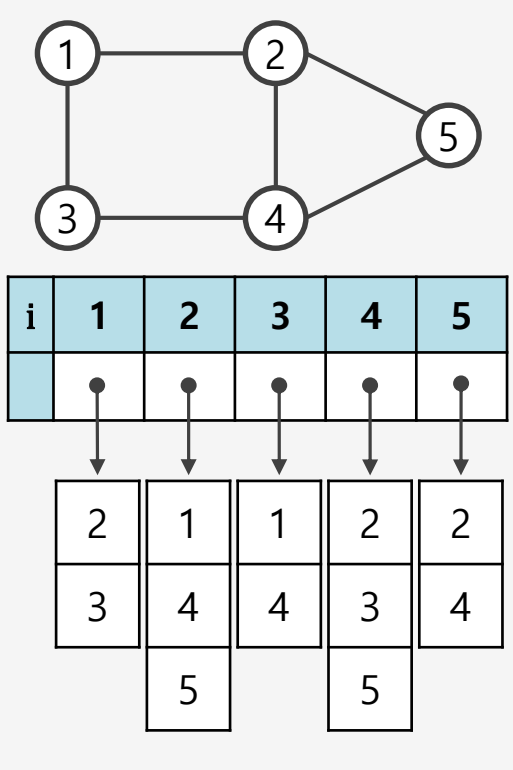
i \ j	1	2	3	4	5
1	0	0	0	0	0
2	1	0	0	0	1
3	1	0	0	1	0
4	0	1	0	0	0
5	0	0	0	1	0

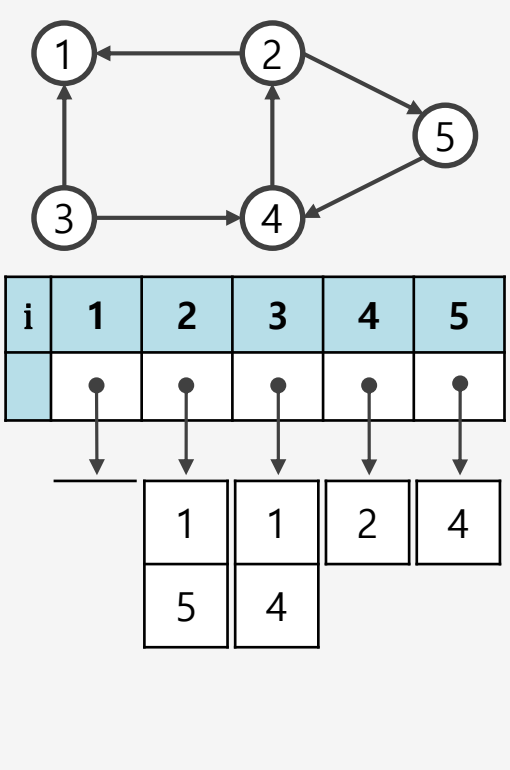


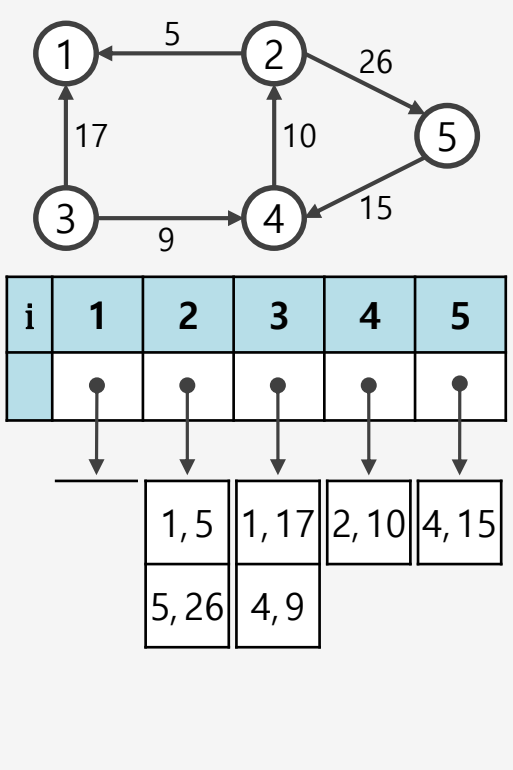
i \ j	1	2	3	4	5
1	0	0	0	0	0
2	5	0	0	0	26
3	17	0	0	9	0
4	0	10	0	0	0
5	0	0	0	15	0

그래프 표현(3)

- 인접 리스트(Adjacent List)
  - 정점의 개수가 V개, 간선의 개수가 E개인 그래프에 대해서 V개의 연결 리스트로 그래프 정보를 저장
  - 일반적으로 List(or Vector)의 배열로 구현







그래프 표현 방식 비교

V개의 정점 E개의 간선 중복간선(×) self-loops(×)	간선 리스트	인접 행렬	인접 리스트
공간	E	V^2	$V+E$
정점 v_A의 부속 간선	E	V	$deg(v_A)$
정점 v_A, v_B 인접 여부	E	1	$min(deg(v_A), deg(v_B))$
정점 삽입	1	V^2	1
간선 삽입	1	1	1

연습문제

1. 1부터 N까지의 수를 각각의 원소로 가지는 N개의 서로소 집합이 있다.
합집합 연산과, **두 원소가 같은 집합에 포함되어 있는지를 확인하는 연산**을 수행하는 프로그램을 작성하시오.

연습문제 접근법

초기의 각 집합들을 하나의 정점으로 보고,
합집합 연산은 정점들을 잇는 간선을 추가 하는 것으로 추상화 한다.
두 원소가 같은 집합인지의 확인은 두 정점이 서로 연결되어있는지를 확인한다.

서로소 집합(Disjoint Set, Union-Find)

- 서로소 집합
  - 교집합이 공집합인 **집합 (서로소 집합) 들의 정보를 확인(Find)**하고 **조작(Union)**할 수 있는 자료구조
  - **Union 연산**
    - 어떤 두 원소 a, b 에 대해서 $union(a, b)$는 각 원소가 속한 집합을 하나로 합치는 연산
    - $a \in A, b \in B$ 에 대해서, $union(a, b)$는 $A \cup B$ 이다.
  - **Find 연산**
    - 어떤 원소 a에 대해서 a가 속한 집합(집합의 대표번호)을 반환
    - $a \in A$ 에 대해서, $find(a)$는 집합 A(집합 A의 대표번호)를 반환

서로소 집합(Disjoint Set, Union-Find)

- 서로소 집합의 표현

배열 (Array)

초기화

i	1	2	3	4	5
U[i]	1	2	3	4	5

U[i] = i

Union 연산 * $O(N)$

i	1	2	3	4	5
U[i]	1	2	1	4	4

union(1, 3)
union(4, 5)

Find 연산 * $O(1)$

i	1	2	3	4	5
U[i]	1	2	1	4	4

find(3) = 1
find(5) = 4

i	1	2	3	4	5
U[i]	1	2	1	1	4

union(5, 3)

트리 (Tree)

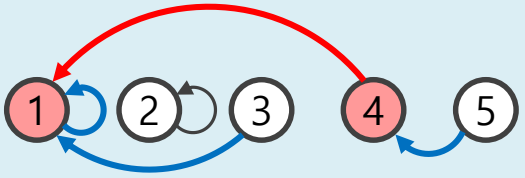
초기화

Union 연산 * $O(H)$, H는 트리의 높이

union(1, 3)
union(4, 5)

Find 연산 * $O(H)$, H는 트리의 높이

find(3) = 1
find(5) = 4



union(5, 3)

서로소 집합(Disjoint Set, Union-Find)

- 서로소 집합의 구현 및 최적화
  - 초기화 : parent 배열에 i 원소에 대한 부모 노드 번호를 저장, 루트 노드인 경우 자기 자신의 번호를 저장

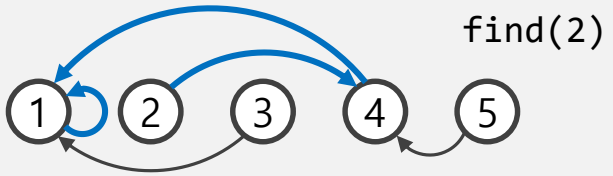
```
function initialize()
   for i : 1 ~ N
      parent[i] = i
```

  - Union 연산 : 하나의 루트 노드를 다른 하나의 루트 노드의 자식 노드로 넣어 두 트리를 합친다.

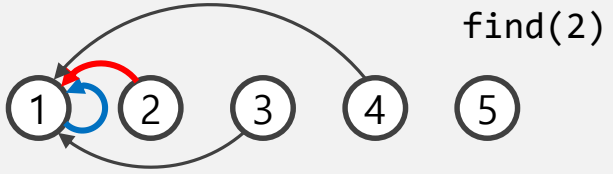
```
function union(a, b)
   aRoot = find(a)
   bRoot = find(b)
   parent[aRoot] = bRoot
```

  - Find 연산 : 주어진 원소의 루트 노드 번호를 반환한다.

```
function find(a)
   if (parent[a] == a) return a
   else return find(parent[a])
```




```
function find(a)
   if (parent[a] == a) return a
   else return parent[a] = find(parent[a])
```



연습문제 풀이법

서로소 집합(Disjoint-set, Union-Find) 자료구조를 이용하여,
주어진 원소들이 속한 집합들의 합집합 연산과,
주어진 원소들이 같은 집합에 속해 있는지 확인한다.

연습문제

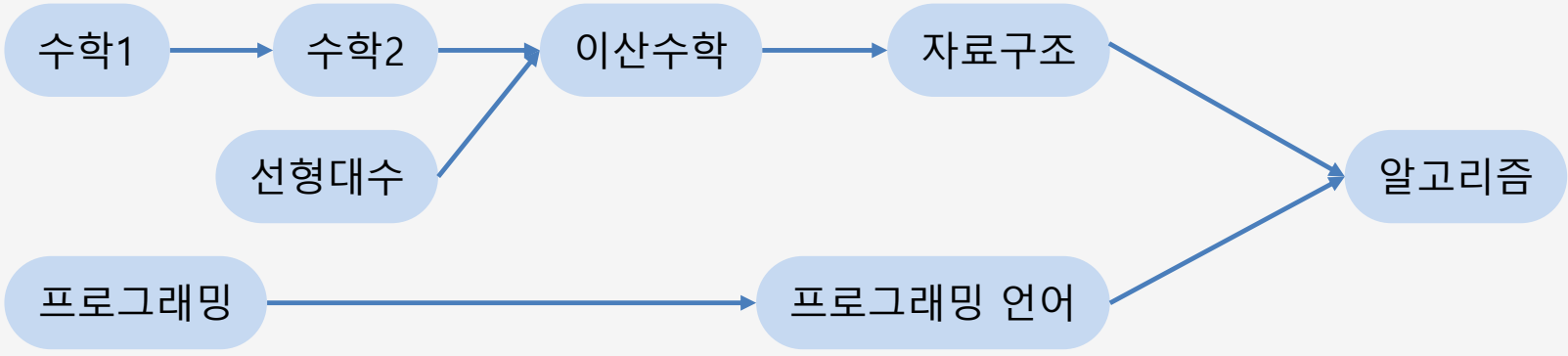
1. N명의 사람을 키 순서대로 줄을 세우려고 한다.
   줄을 세우기 위해 임의의 두 사람의 키를 비교한다.
   비교를 함으로써 둘 중 누가 앞에 서고, 누가 뒤에 서는지를 알 수 있을 것이다.
   일부 쌍에 대하여 키를 비교한 결과가 주어질 때, 줄을 세우는 프로그램을 작성하시오.

연습문제 접근법

각 사람을 하나의 정점으로
사람의 키를 비교한 각 결과를 방향 간선(키가 작은 사람 -> 키가 큰 사람)으로 하는
그래프로 추상화하여 생각해 본다.

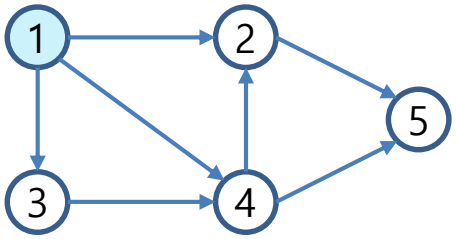
DAG(Directed Acyclic Graph)

- **DAG는 순환을 가지지 않는 방향그래프를 말한다.**
- 일반적으로 우선순위를 가진 일련의 작업들은 DAG구조를 가진다.
- DAG의 예
  - 스타크래프트에서 건물 짓는 순서
  - 대학 과정의 선수과목
- 선행자(predecessor), 후행자(successor)
  - DAG에서 어떤 정점 $v_i \in V, v_j \in V$ 에 대해서 v_i에서 v_j로의 경로가 존재하면, v_i를 v_j의 선행자(predecessor), v_j는 v_i의 후행자(successor)라 한다.
- 즉각 선행자(immediate predecessor), 즉각 후행자(immediate successor)
  - DAG에서 어떤 정점 $v_i \in V, v_j \in V$ 에 대해서 v_i에서 v_j로의 간선이 존재하면, v_i를 v_j의 즉각 선행자(immediate predecessor), v_j는 v_i의 즉각 후행자(immediate successor)라 한다.

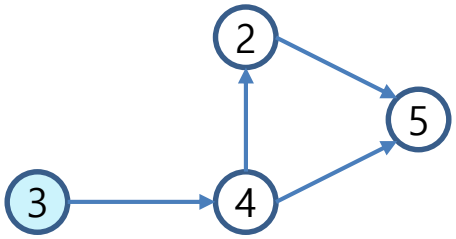


위상정렬(Topological Sort)

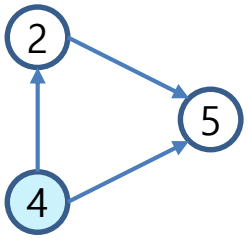
- **DAG(비순환 방향 그래프)에서 그래프의 방향성을 거스르지 않고 정점들을 나열하는 것**
- 위상정렬은 각 정점을 우선순위에 따라 배치 한 것
- 일반적으로 위상정렬의 결과는 유일하지 않다.



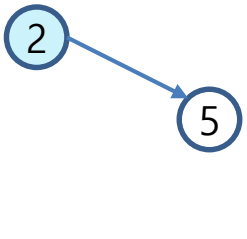
정점	1	2	3	4	5
진입 차수	0	2	1	2	2



정점	2	3	4	5
진입 차수	1	0	1	2



정점	2	4	5
진입 차수	1	0	2



정점	2	5
진입 차수	0	1

정점	5
진입 차수	0

연습문제 풀이법

사람들을 하나의 정점으로 보고,
사람 간의 키를 비교한 결과를 키가 작은 사람에서 키가 큰 사람으로 연결하는 간선으로 생각하면,
주어진 문제의 그래프는 싸이클이 없는 방향 그래프(DAG)이다.
결국 문제는 그래프의 정점들을 간선의 방향을 거스르지 않고(키 순서 대로) 정렬을 하라는 문제이고,
이는 주어진 그래프의 위상정렬을 구함으로써 해결 할 수 있다.

연습문제

1. 컴퓨터와 컴퓨터를 모두 연결하는 네트워크를 구축하려 한다.
   네트워크를 구축하기 위해서는 컴퓨터와 컴퓨터를 직접 선으로 연결하여야 한다.
   모든 컴퓨터는 서로 연결이 되어 있어야 한다. (임의의 두 컴퓨터 간 경로가 존재해야 한다.)

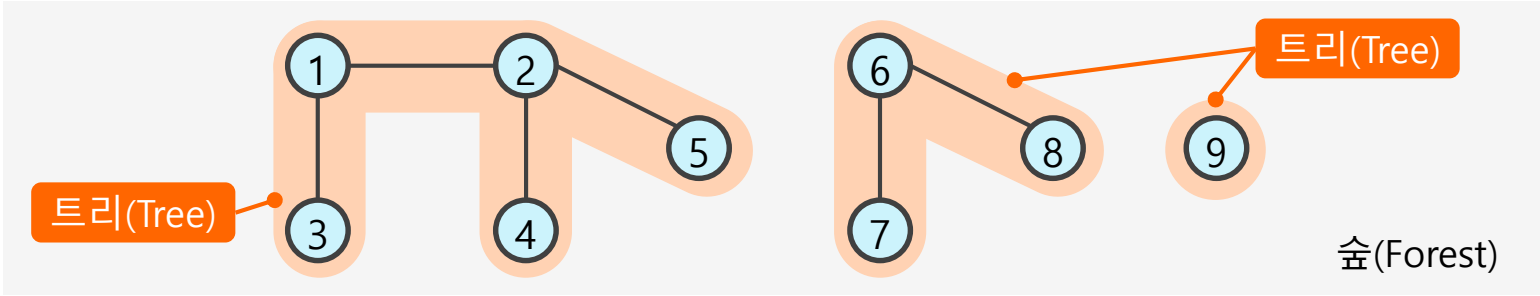
   각 컴퓨터를 연결하는데 필요한 비용이 주어졌을 때, 모든 컴퓨터를 연결하는데 필요한 최소비용을 출력하라.

연습문제 접근법

각 컴퓨터를 하나의 정점으로 컴퓨터들의 직접 연결을 간선으로 하는 그래프로 추상화하여 생각해 본다. 컴퓨터를 연결하는데 비용이 발생하고, 컴퓨터를 연결하는데 있어 간선은 방향성을 가지지 않기 때문에 무향 가중 그래프라고 볼 수 있다.

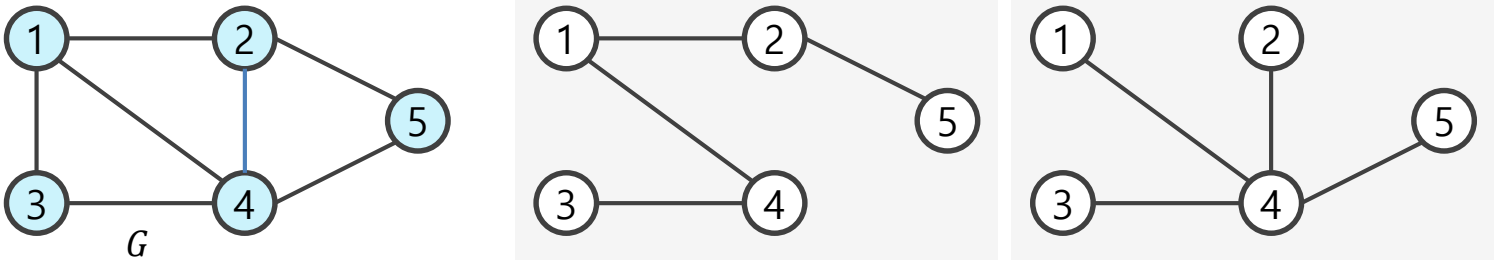
트리(Tree)

- **무향 그래프 G가 순환이 없는 그래프이면 G는 숲(Forest) 이다**



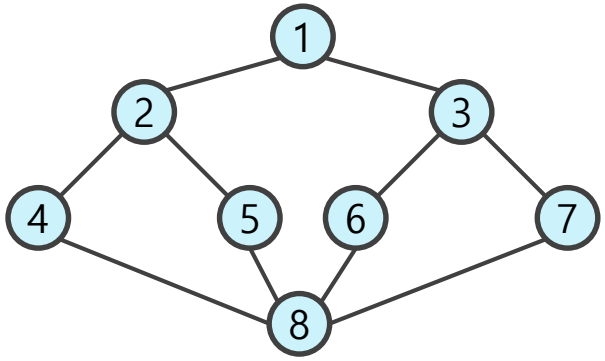
- **무향 그래프 G가 순환이 없는 연결 그래프이면 G는 트리(Tree)이다.**
  - G는 순환이 없고, 단순 그래프에서 간선을 추가할 경우 순환이 생긴다.
  - G는 연결 그래프 이고, 어떤 간선을 제거하면 연결그래프가 아니다.
  - G의 어떤 두 정점에 대해서 단순 경로가 하나 존재한다.

- 신장 트리(Spanning Tree)
  - 무향 연결 그래프 G의 부분그래프이고, G의 모든 정점을 포함하는 트리(Tree)인 그래프

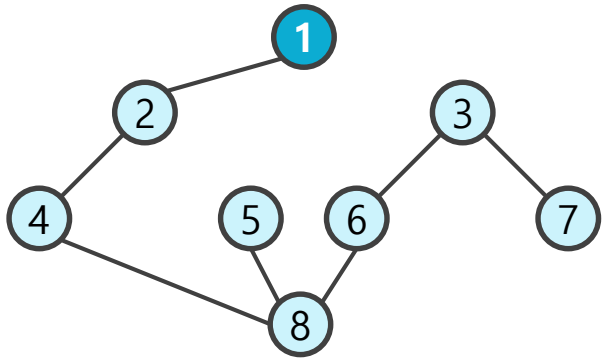


깊이 우선(DFS) 신장(Spanning) 트리(Tree)

- 무향 연결 그래프 G에서 깊이 우선 탐색(DFS)으로 탐색하면서 생성된 신장(Spanning) 트리(Tree)

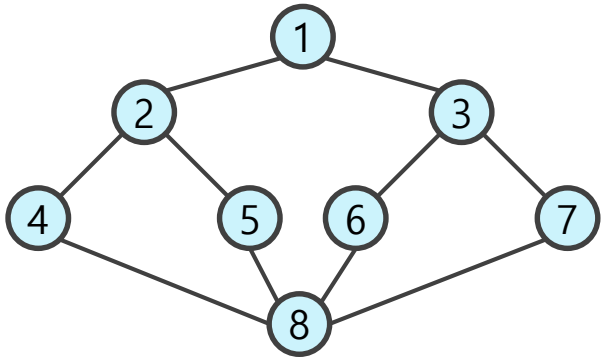


G

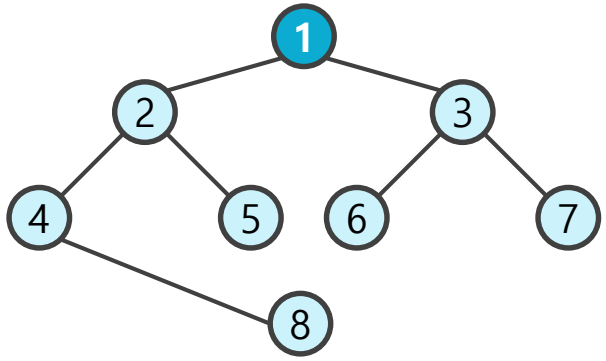


너비 우선(BFS) 신장(Spanning) 트리(Tree)

- 무향 연결 그래프 G에서 너비 우선 탐색(BFS)으로 탐색하면서 생성된 신장(Spanning) 트리(Tree)

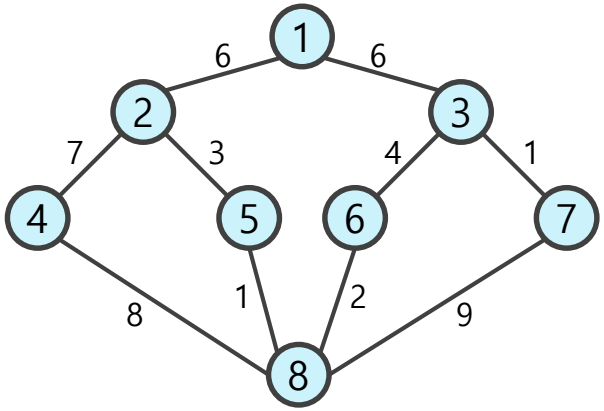


G

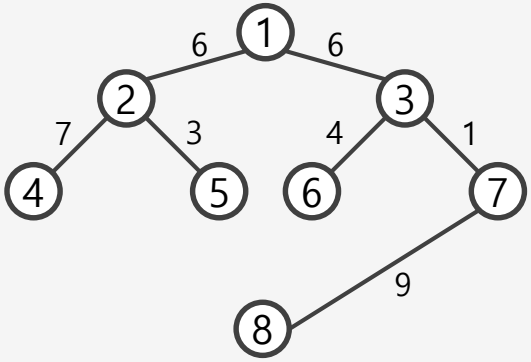


최소 신장 트리(Minimum Spanning Tree / MST)

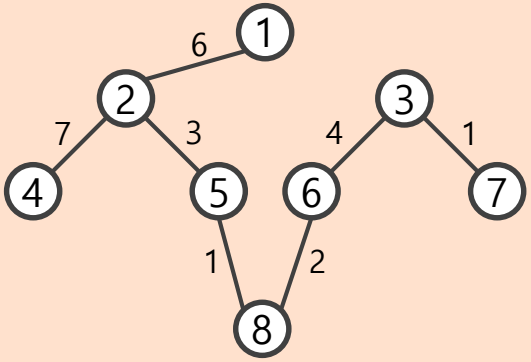
- 무향 연결 가중 그래프 G에서 간선의 가중치의 합이 최소인 신장(Spanning) 트리(Tree)



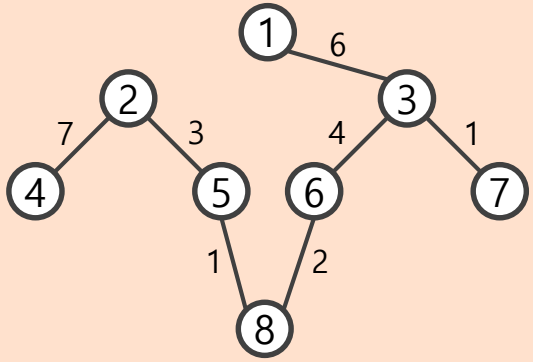
G



$sum = 36$



$sum = 24$

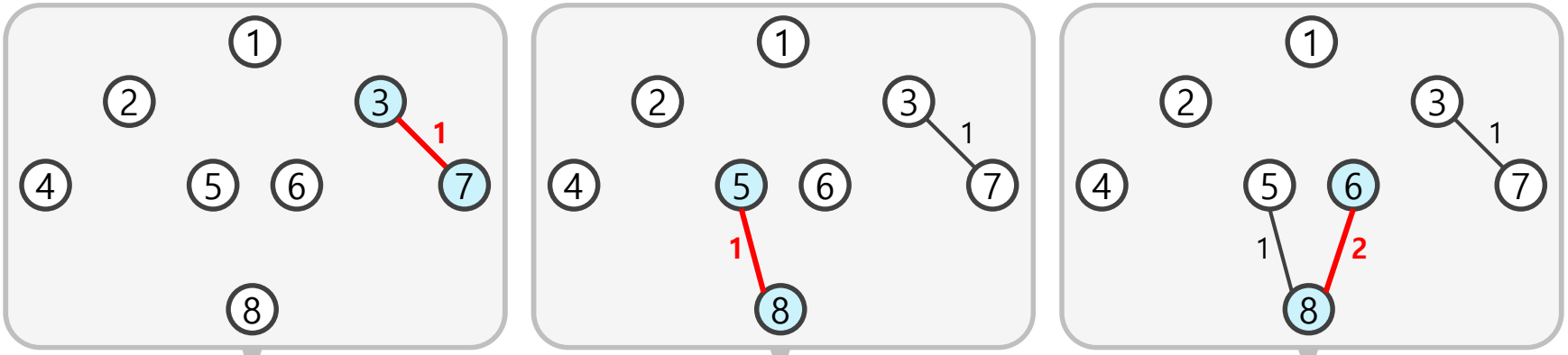


$sum = 24$

최소 신장 트리

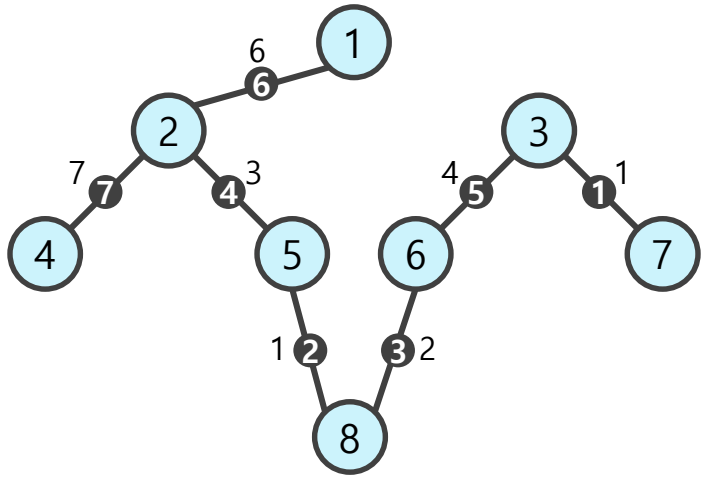
신장 트리

크루스칼 알고리즘(Kruskal's Algorithm) - 최소 신장 트리(Minimum Spanning Tree / MST)



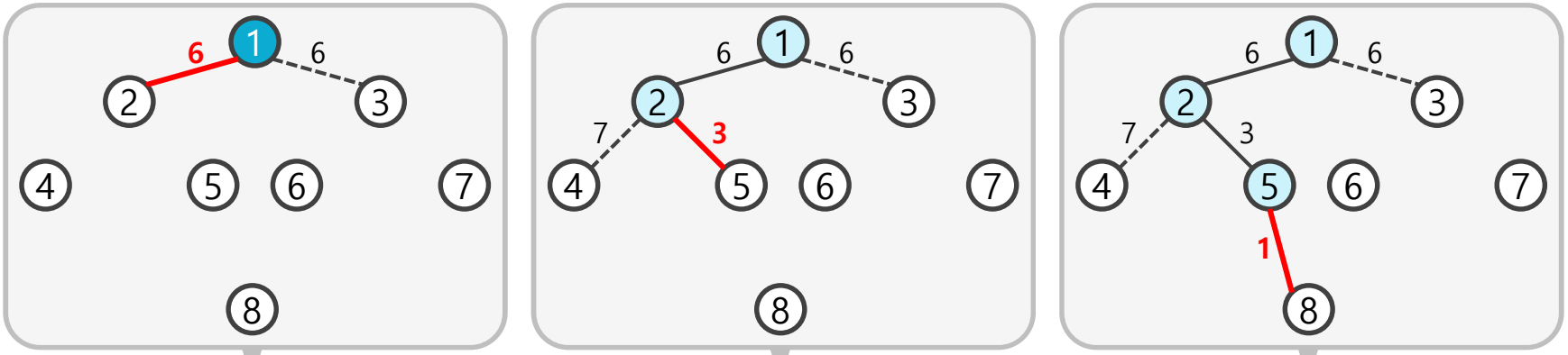
오름차순 정렬

3	1	7	❶
5	1	8	❷
8	2	6	❸
2	3	5	❹
6	4	3	❺
2	6	1	❻
1	6	3	
4	7	2	❼
4	8	8	
8	9	7	



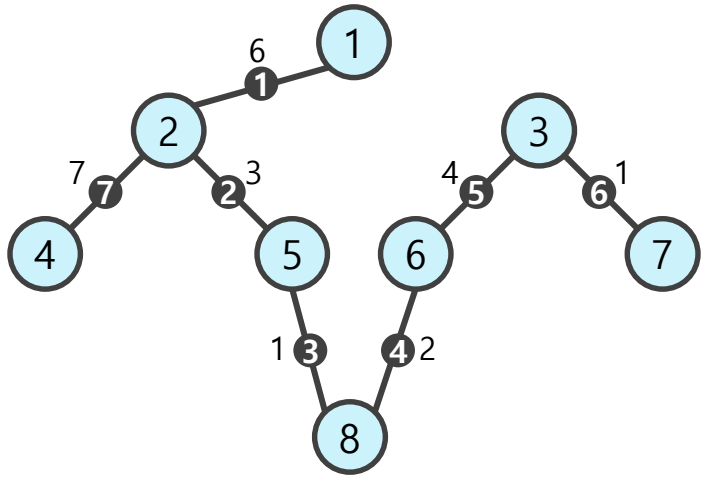
ⓝ 간선이 선택된 순서

프림 알고리즘(Prim's Algorithm) - 최소 신장 트리(Minimum Spanning Tree / MST)



최소힙

{7, 1}	7
{3, 4}	6
{7, 9}	11
{6, 2}	5
{4, 8}	10
{8, 1}	4
{5, 3}	3
{4, 7}	9
{3, 6}	8
{2, 6}	2
{1, 0}	1



ⓝ 간선이 선택된 순서

연습문제 풀이법

주어진 정점과 가중 간선을 이용하여
최소 비용으로 연결 그래프를 만드는 것으로 생각해 볼 수 있다.
결국 크루스칼 알고리즘이나 프림 알고리즘을 이용하여 최소신장트리를 완성하는 것으로 문제를 해결할 수 있다.

연습문제

1. N개의 노드로 이루어진 트리가 주어진다.
   트리의 각 노드는 1번부터 N번까지 번호가 매겨져 있으며, 루트는 1번이다.
   두 노드가 주어졌을 때, 이 노드들의 가장 가까운 공통 조상이 몇 번인지 출력한다.

2. 노드의 개수 N이 최대 10만개라면?

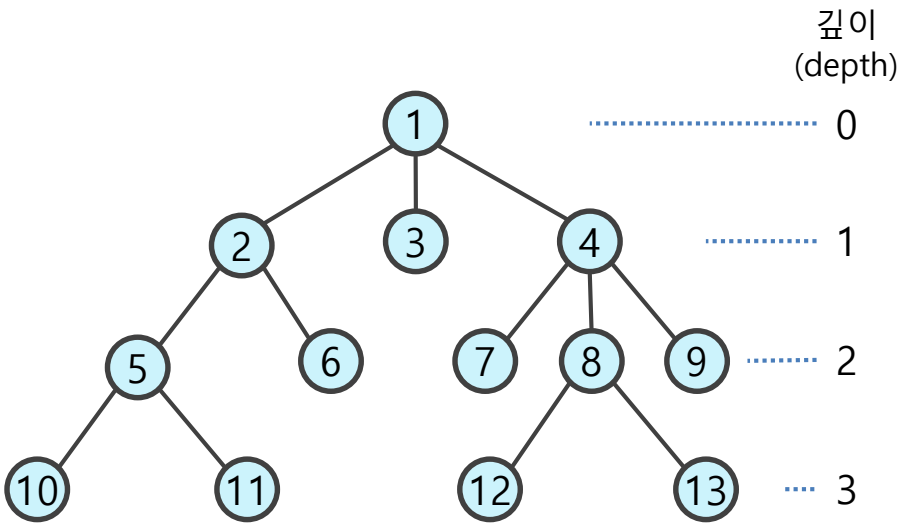
3. 가장 가까운 공통 조상을 묻는 질의가 최대 10만개라면?

연습문제 접근법

트리 그래프를 그려서 임의의 두 노드를 선택하여 최소공통 조상을 찾아보고,
경우에 따른 그 특징을 생각해 본다.

최소 공통 조상(Lowest Common Ancestor / LCA)

- 정점 A와 정점 B가 각각 자신을 포함하여 조상을 따라 거슬러 올라 갈 때 처음 공통으로 만나게 되는 정점이다.
- 정점 A 혹은 A의 조상이면서 정점 B 혹은 B의 조상인 정점들 중 가장 깊은 정점이다.



7과 9의 공통 조상들은 4 1

7과 9의 최소 공통 조상은 4

10과 6의 공통 조상들은 2 1

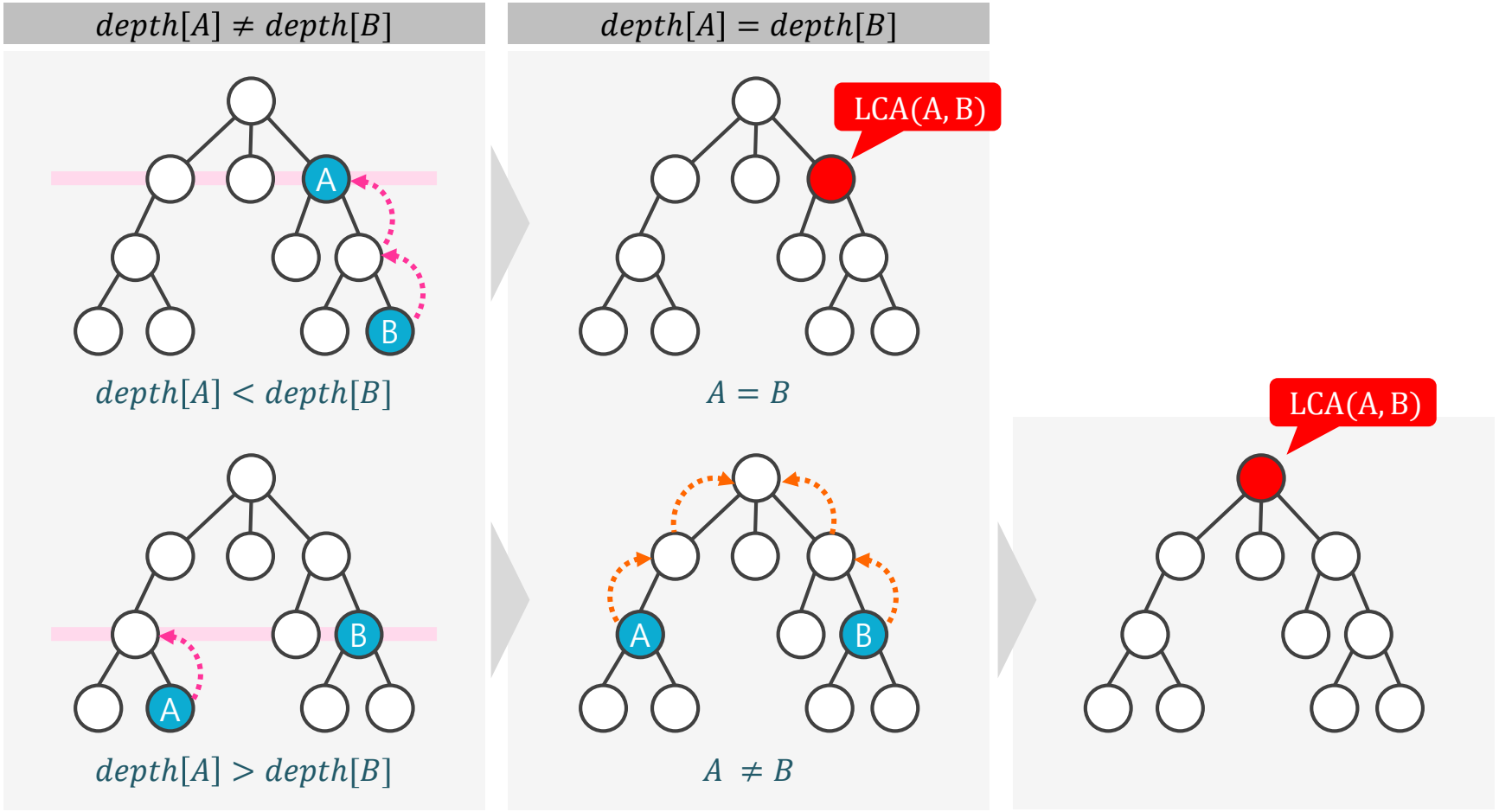
10과 6의 최소 공통 조상은 2

11과 2의 공통 조상들은 2 1

11과 2의 최소 공통 조상은 2

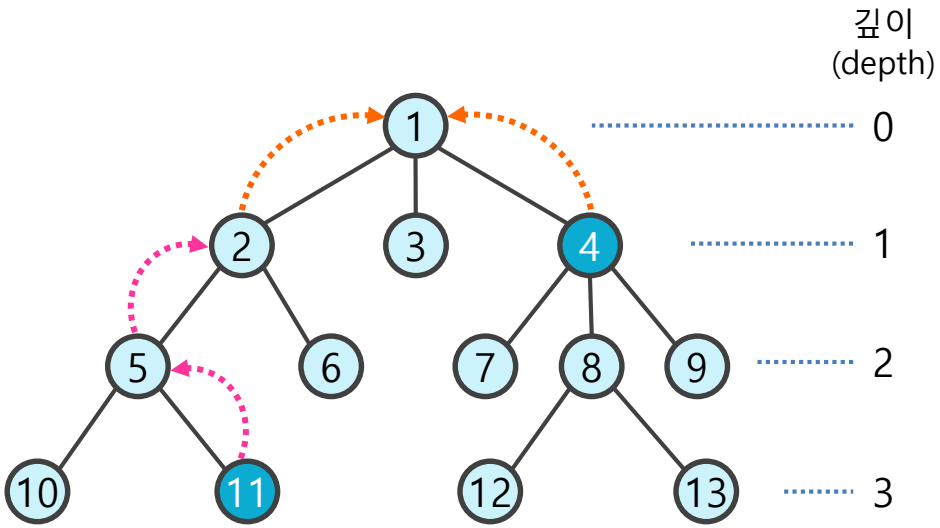
최소 공통 조상(Lowest Common Ancestor / LCA) 구하기

1. 최상위 조상 정점을 시작으로 DFS(or BFS)를 수행하여 각 정점의 **깊이와 부모 정점**을 저장한다.
2. 정점의 부모를 따라 **하나씩** 올라가 LCA를 찾는다.



최소 공통 조상(Lowest Common Ancestor / LCA) 구하기

LCA(4 , 11) ?



트리의 깊이를 H라 할 때, 시간복잡도는 $O(H)$

최악의 경우 $O(N)$

- A, B 정점의 깊이 차이가 11이라면 더 크게 움직여 서로의 깊이를 맞출 수 없을까?
- A, B 정점이 10번을 올라가도 서로가 같지 않다면 한 번에 10번을 올라갈 수는 없을까?

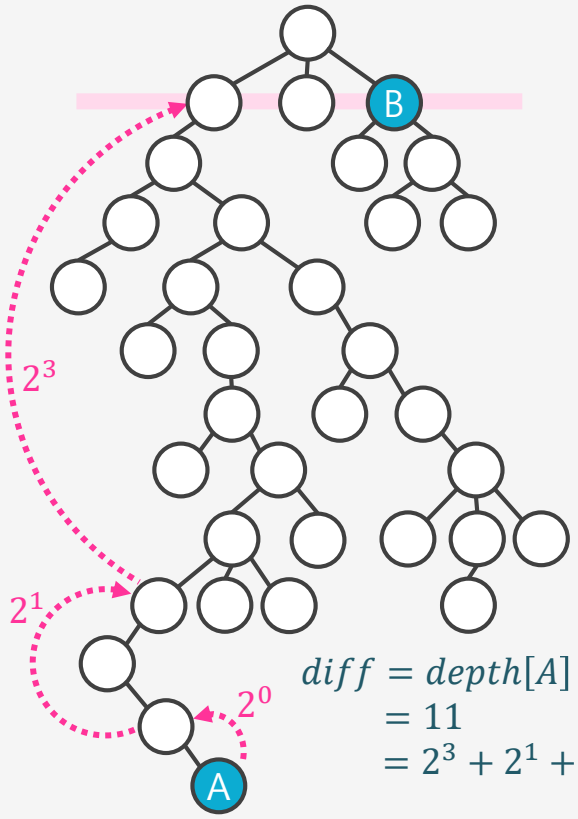
최소 공통 조상(Lowest Common Ancestor / LCA) 빠르게 구하기

1. 각 정점의 부모 뿐 아니라 2^k **번째 조상**까지 저장한다.

$$parent[k][V] = parent[k-1]\big[parent[k-1][V]\big]$$

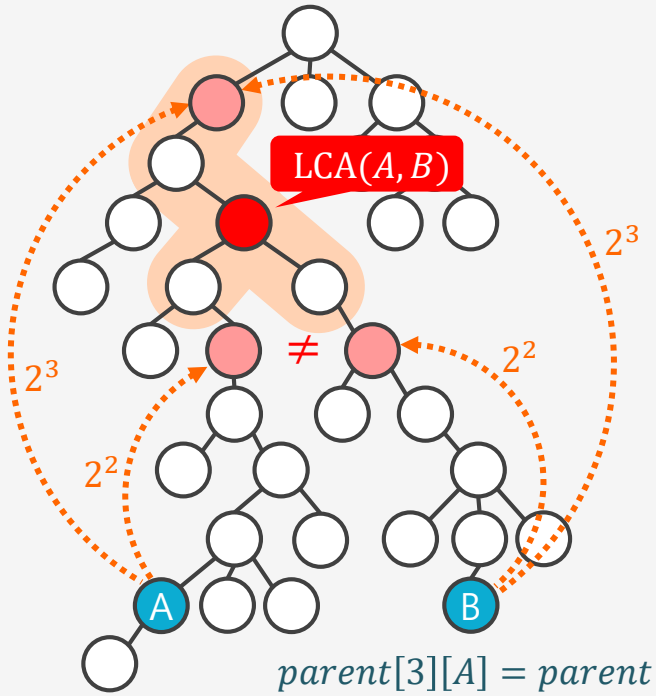
* $parent[k][V]$:
정점 V의 2^k번째 조상 정점 번호

$depth[A] \neq depth[B]$



$$diff = depth[A] - depth[B]$$
$$= 11$$
$$= 2^3 + 2^1 + 2^0$$

$depth[A] = depth[B]$



$$parent[3][A] = parent[3][B]$$
$$parent[2][A] \neq parent[2][B]$$

$LCA(A, B)$는 2^2번째 조상과 2^3번째 조상 사이에 존재

최소 공통 조상(Lowest Common Ancestor / LCA) 빠르게 구하기

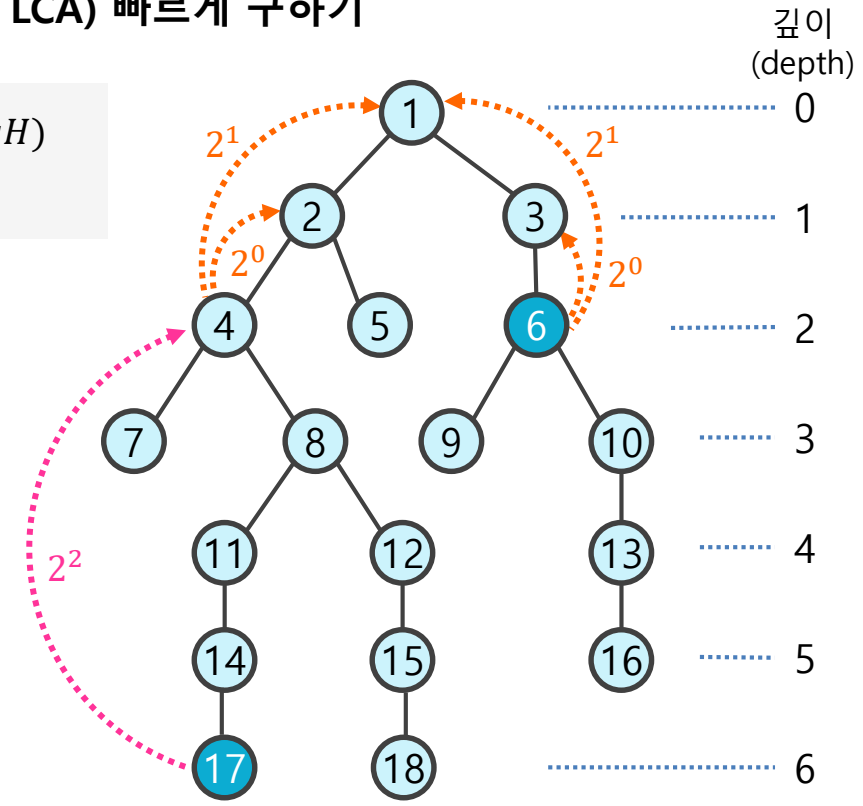
트리의 깊이를 H라 할 때, 시간복잡도는 $O(logH)$
최악의 경우 $O(logN)$

LCA(6 , 17) ?

$$diff = depth[17] - depth[6]$$
$$= 6 - 2 = 4$$
$$= 2^2$$

$$parent[2][4] = 0 = 0 = parent[2][6]$$
$$parent[1][4] = 1 = 1 = parent[1][6]$$
$$parent[0][4] = 2 \neq 3 = parent[0][6]$$

LCA(17 , 18) ?



parent =

K \ V	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0	1	1	2	2	3	4	4	6	6	8	8	10	11	12	13	14	15
1	0	0	0	1	1	1	2	2	3	3	4	4	6	8	8	10	11	12
2	0	0	0	0	0	0	0	0	0	0	1	1	1	2	2	3	4	4

연습문제 풀이법

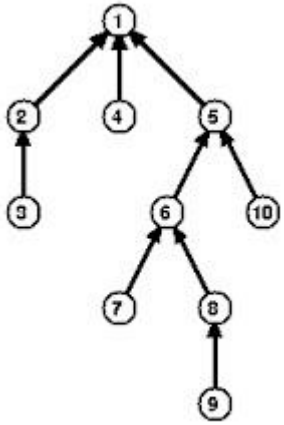
노드의 개수 N이 10만개이고 LCA 질의의 개수 Q가 10만개라고 생각해보자.

DP를 이용하지 않는다면, 최악의 경우 $O(N \times Q) \Rightarrow 10만 \times 10만 = 1조$ 로
제한 시간 내에 문제를 해결할 수 없다.

DP를 이용하면 한 번의 질의에 대해 최대 $\log(N)$ 의 연산이 필요하다.
따라서, $O(Q \times logN) \Rightarrow 10만 \times log(10만) \approx 10만 \times 20 = 2백만$ 으로
제한시간 내에 문제를 해결 할 수 있다.

연습문제

1. Rooted tree란 tree의 한 노드가 root가 되고 나머지 노드들은 다른 노드를 부모로 가지는 것이다.
어떤 노드의 조상은 자신, 자신의 부모, 부모의 부모, 부모의 부모의 부모, ..., root 까지를 일컫는다.
노드가 N개로 이루어진 트리의 정보와 루트 R이 주어졌다.
트리의 각 노드는 1번부터 N번까지 번호가 배정되어 있으며 모두 다르다.
이어서 Q개의 질의가 들어오는데, 질의는 다음과 같은 형태이다.
   - x, y : 노드 x가 노드 y의 조상이면 YES 아니면 NO



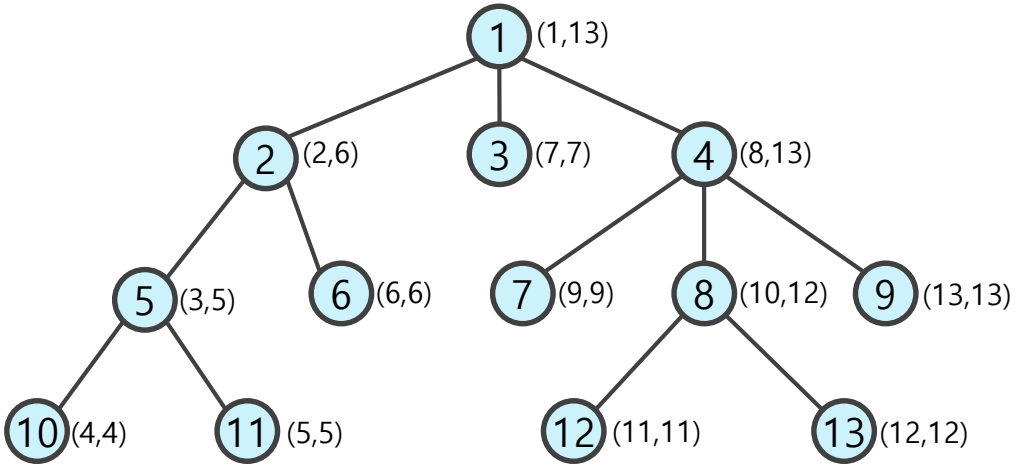
왼쪽과 같은 트리 구조라고 하자.
루트 노드는 1번노드이다

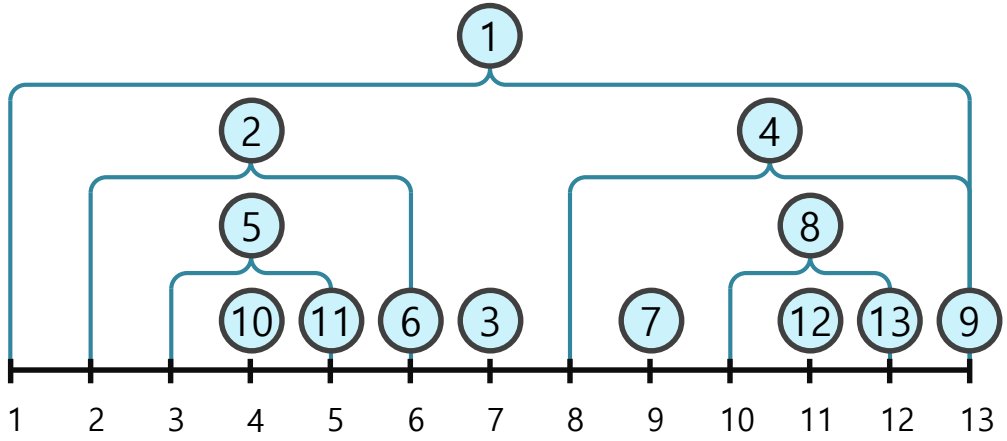
- 5번은 7번의 조상이다.
- 2번은 2번의 조상이다.
- 1번은 모든 노드의 조상이다.
- 10번은 9번의 조상이 아니다.
- 4번은 1번의 조상이 아니다.

각 질의에 올바른 대답을 하는 프로그램을 작성하라.

트리(Tree)에서 조상-자손 관계 판별

- 트리 그래프 G에서 임의의 두 정점 A, B에 대하여 A와 B가 조상-자손 관계인지 여부를 판단





- in-order : 해당 정점의 방문 순서
- out-order : 해당 정점을 벗어나는 순서

어떤 정점 A, B에 대해서

$$inout(A) = (in_A, out_A)$$
$$inout(B) = (in_B, out_B)$$

$$\begin{cases} in_A \le in_B \le out_B \le out_A \Rightarrow A\text{는 } B\text{의 조상} \\ in_B \le in_A \le out_A \le out_B \Rightarrow B\text{는 } A\text{의 조상} \\ (in_A, out_A) \cap (in_B, out_B) = \emptyset \\ \qquad\qquad \Rightarrow \text{조상 · 자손 관계가 아님} \end{cases}$$

연습문제 풀이법

주어진 그래프에 대해서 Root Node에서 부터 dfs탐색을 수행하여 in-out order 를 저장한다.
다음으로 주어지는 Q개의 질의에 대해서 아래와 같이 확인하여 답을 출력한다.
in-out(x)=(in_x , out_x), in-out(y)=(in_y , out_y) 일때,
만약 (in_x , out_x) ⊃(in_y , out_y) 이면(즉, $in_x \le in_y \le out_y \le out_x$이면) x는 y의 부모이다.

연습문제

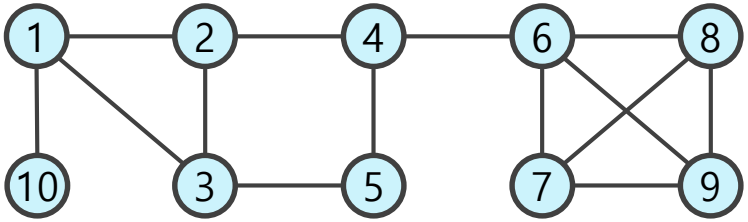
1. 그래프가 주어졌을 때, 단절점을 모두 구해 출력하는 프로그램을 작성하시오.

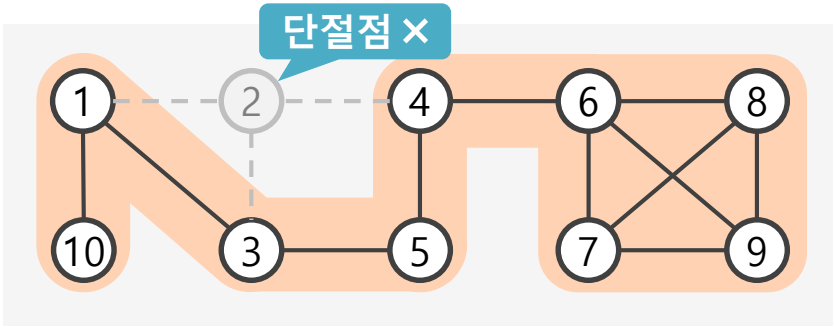
연습문제 접근법

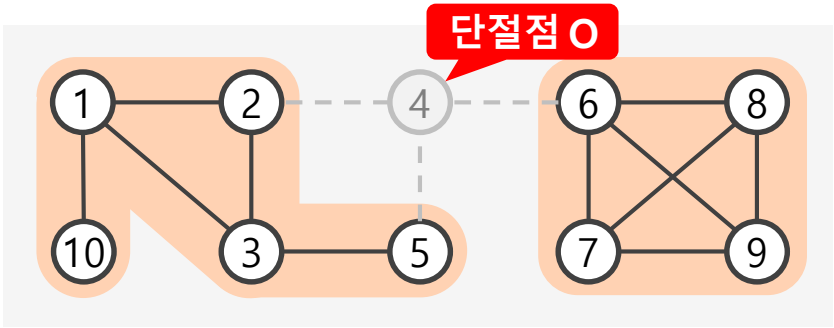
그래프에서 단절점과 단절점이 아닌 정점에서 나타나는 특징 및 차이점을 생각해 본다

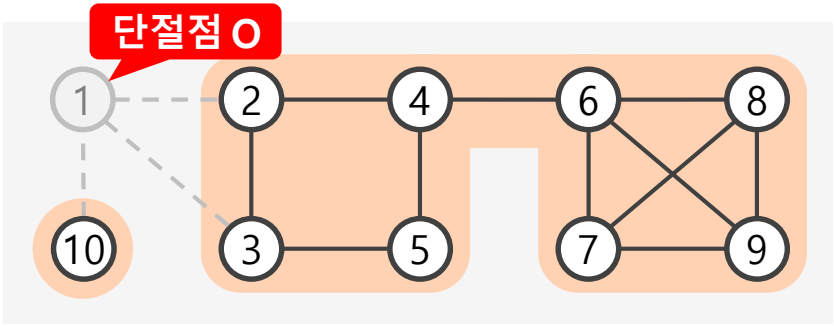
단절점(Articulation Points / Cut Vertices)

- 무향 연결 그래프 G에서 하나의 정점과 연결된 간선들을 제거했을 때 두 개의 연결 그래프로 나뉘어 진다면 해당 정점은 단절점이다.



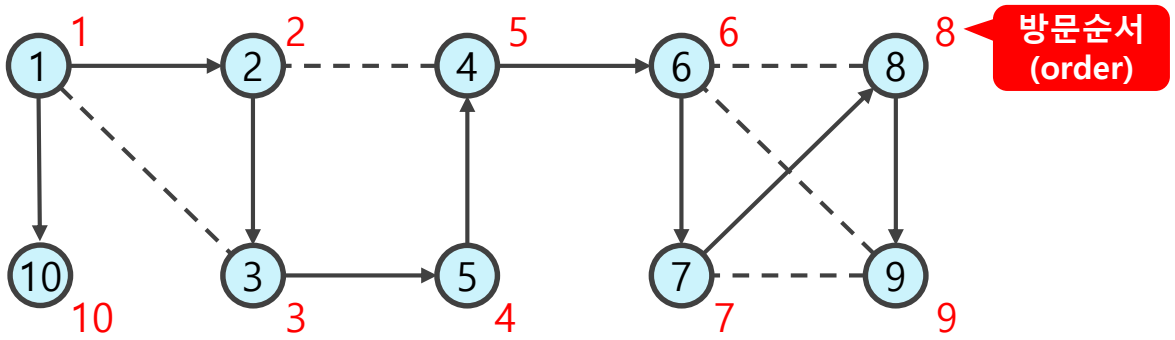






단절점 찾기

- 무향 연결 그래프 G에서 어떤 정점 A에 연결된 정점들에 대해서
  우회경로(정점 A를 거치지 않는 경로)가 없는 경우가 존재한다면 A는 단절점이다.



어떤 정점 A와 A에 연결된 정점 B에 대해서

$$\begin{cases} A\text{가 시작정점이 아닌 경우} \begin{cases} order_A \le low_B \Rightarrow A\text{는 단절점} \\ order_A > low_B \Rightarrow A\text{는 단절점이 아님} \end{cases} \\ A\text{가 시작정점인 경우} \begin{cases} 2 \le child_A \Rightarrow A\text{는 단절점} \\ 2 > child_A \Rightarrow A\text{는 단절점이 아님} \end{cases} \end{cases}$$

$order_V$: 정점 V의 방문순서
low_V : 정점 V의 low
$child_V$: 정점 V의 자식 트리 수

V 이후에 방문하는 정점들 중 V를 거치지 않고 방문 가능한 정점들의 order 중 가장 작은 값이며, 초기 값은 자신의 order이다.

연습문제 풀이법

단절점을 찾는 알고리즘을 이용하여 해결한다.
주어진 그래프에 대해서 방문하지 않은 정점을 시작으로 dfs탐색을 수행하여 order 를 저장한다.
dfs함수는 하나의 정점에 연결된 다른 정점들을 탐색하면서 low를 리턴한다.
자신과 연결된 정점에 대해서 dfs탐색을 수행 후 반환된 low값과 자신의 방문 order를 비교하여 단절점 여부를 확인한다.

- 현재 정점이 시작 정점(Root Node)이 아니면서
  자신과 연결된 정점의 dfs탐색 후 반환한 low값이 자신의 방문 order 보다 크거나 같다면
  그러한 점이 한 개라도 존재한다면 현재 정점은 단절점이다.

- 현재 정점이 시작 정점(Root Node)이면서
  자신과 연결된 자식트리의 개수가 2개 이상이라면
  현재 정점은 단절점이다.

연습문제

1. 최단경로 1

N개의 도시와 임의의 한 도시에서 출발하여 다른 도시에 도착하는 M개의 버스가 있다.
각 버스는 한 번 이용하는 데 걸리는 시간이 있다.
어떤 도시 S에서 출발하여 다른 모든 도시로의 최소 시간을 구하는 프로그램을 작성하시오.

2. 최단경로 2

N개의 도시와 임의의 한 도시에서 출발하여 다른 도시에 도착하는 M개의 버스가 있다.
각 버스는 한 번 이용하는 데 걸리는 시간이 있다. 시간 C는 양수가 아닌 경우가 있다.
C = 0인 경우는 순간이동을 하는 경우, C < 0인 경우는 타임머신으로 시간을 되돌아가는 경우이다.
어떤 도시 S에서 출발하여 다른 모든 도시로의 최소 시간을 구하는 프로그램을 작성하시오.

3. 최단경로 3

N개의 도시와 임의의 한 도시에서 출발하여 다른 도시에 도착하는 M개의 버스가 있다.
각 버스는 한 번 이용하는 데 걸리는 시간이 있다.
모든 도시의 쌍 (A, B)에 대해서 도시 A에서 도시 B로 가는데 걸리는 최소 시간을 구하는 프로그램을 작성하시오.

연습문제 접근법

각 문제의 조건과 구하고자 하는 최단거리의 특징을 생각해 보자.

최단 경로 문제

- 최단 경로 문제란 가중 그래프에서 간선의 가중치의 합이 최소가 되는 경로를 찾는 문제
- 최단 경로 문제의 종류
  - **단일 출발(single-source) 최단 경로**
    어떤 하나의 정점에서 출발하여 나머지 모든 정점 까지의 최단 경로를 찾는 문제
  - **단일 도착(single-destination) 최단 경로**
    모든 정점에서 출발하여 어떤 하나의 정점까지의 최단 경로를 찾는 문제
    그래프 내의 간선들을 뒤집으면 단일 출발 최단거리 문제로 바뀔 수 있다.
  - **단일 쌍(single-pair) 최단 경로**
    어떤 정점 v에서 v'로 가능 최단경로를 구하는 문제
  - **전체 쌍(all-pair) 최단 경로**
    모든 정점 쌍들 사이의 최단 경로를 찾는 문제
- 주요 알고리즘
  - **BFS(완전탐색 알고리즘)**
    가중치가 없거나 모든 가중치가 동일한 그래프에서 최단경로를 구하는 경우 가장 빠르다
  - **다익스트라 알고리즘(Dijkstra Algorithm)**
    음이 아닌 가중 그래프에서의 단일 쌍, 단일 출발, 단일 도착 최단 경로 문제
  - **벨만-포드 알고리즘(Bellman-Ford-Moore Algorithm)**
    가중 그래프에서의 단일 쌍, 단일 출발, 단일 도착 최단 경로 문제
  - **플로이드-워셜 알고리즘(Floyd-Warshall Algorithm)**
    전체 쌍 최단 경로 문제

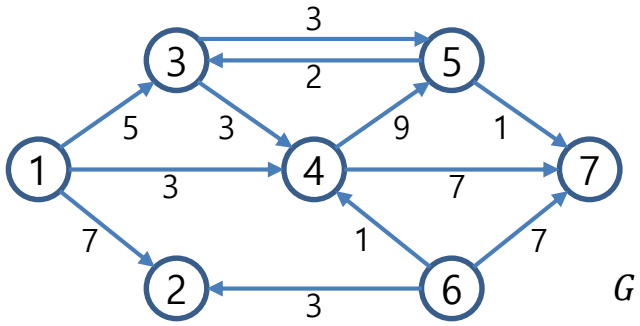
다익스트라 알고리즘(Dijkstra Algorithm) - 최단 경로 알고리즘(1)

V개의 정점과 음수가 아닌 E개의 간선을 가진 그래프 G에서
특정 출발 정점(S)에서 부터 다른 모든 정점까지의 최단경로를 구하는 알고리즘.

- **특징**
  - 각 정점을 최대 한번씩만 방문하여 최단 거리를 확정한다. (∴ 음의 가중치를 가질 수 없다.)
  - 아직 방문하지 않은 정점들 중 최단거리인 점을 찾아 방문하여 최단거리를 확정하고, 이를 반복하는 방식으로 진행된다.
  - 총 $V \times V$번 연산이 필요하다. 따라서 $O(V^2)$의 시간복잡도를 가진다.
  - 방문하지 않은 정점 중에서 최단 거리가 최소인 정점을 찾는 과정에서 우선순위 큐 혹은 힙 자료구조를 이용하면 더욱 개선된 알고리즘이 가능하다.

다익스트라 알고리즘(Dijkstra Algorithm) - 최단 경로 알고리즘(1)

7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가진 가중 그래프 G에서 정점 1에서 나머지 정점까지의 최단거리를 구하여라

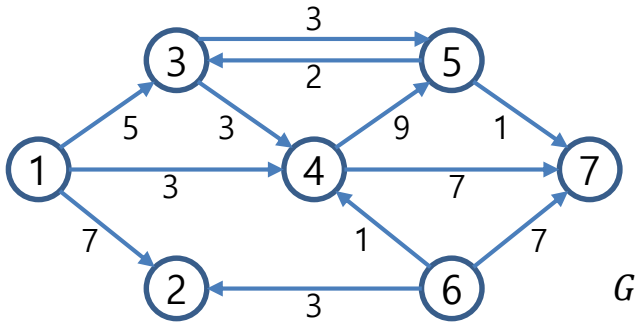


방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞

$D[K]$: 출발 정점 S에서 정점 K까지의 최단거리를 저장하는 배열

다익스트라 알고리즘(Dijkstra Algorithm) - 최단 경로 알고리즘(1)

7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가진 가중 그래프 G에서 정점 1에서 나머지 정점까지의 최단거리를 구하여라



방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
1	0	7	5	3	∞	∞	∞
4	0	7	5	3	12	∞	10
3	0	7	5	3	8	∞	10
2	0	7	5	3	8	∞	10
5	0	7	5	3	8	∞	9
7	0	7	5	3	8	∞	9

$D[K]$: 출발 정점 S에서 정점 K까지의 최단거리를 저장하는 배열

다익스트라 알고리즘(Dijkstra Algorithm) - 최단 경로 알고리즘(1)

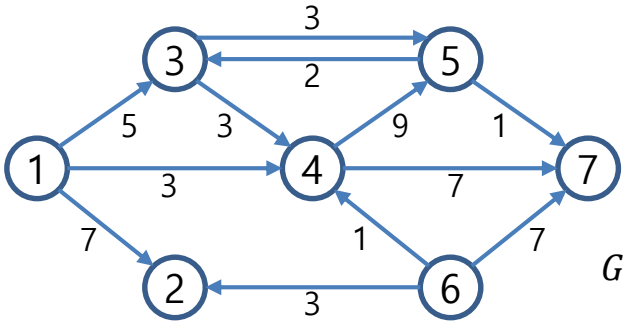
- **개선된 알고리즘 단계**

$D[S] = 0$ (출발점을 0으로 저장)	$D[S] = 0$ (출발점을 0으로 저장) 최소 힙에 출발점 S 삽입
방문하지 않은 정점 중에서 $D[K]$가 최소인 정점 I 선택	최소 힙에서 맨 위에 있는 정점 I 꺼냄
$D[J] = D[I] + W$ 로 갱신	$D[J] = D[I] + W$ 로 갱신 최소 힙에 정점 J 삽입

- **특징**
  - 최악의 경우 모든 간선을 확인할 때마다 거리 배열이 갱신이 된다고 하면,
    최소 힙에는 최대 E번 정점을 넣게 된다.(간선의 개수는 E개 이므로)
  - 최소 힙에 E개의 정점이 있을 때, 하나의 정점을 꺼낼 때 마다 $logE$의 연산이 필요하다.
    (최소 힙 삭제연산의 시간복잡도)
  - $ElogE = ElogV^2 = 2ElogV$
  - 즉, 최소 힙 혹은 우선순위 큐를 사용하는 경우, 시간 복잡도는 $O(ElogV)$이다.

다익스트라 알고리즘(Dijkstra Algorithm) - 최단 경로 알고리즘(1)

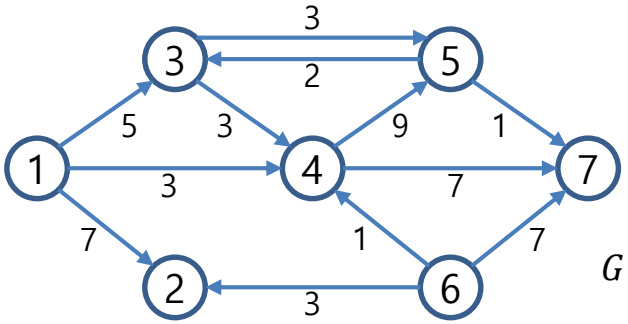
7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가진 가중 그래프 G에서 정점 1에서 나머지 정점까지의 최단거리를 구하여라



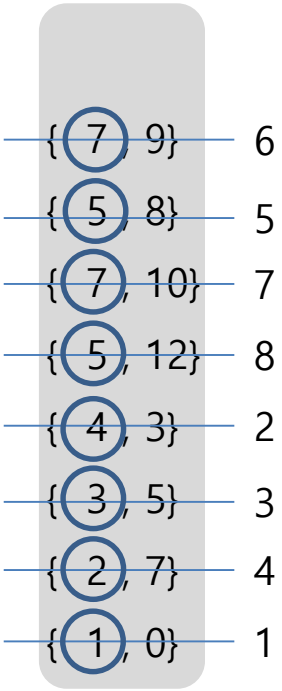
처리 정점 K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
{1, 0}							

다익스트라 알고리즘(Dijkstra Algorithm) - 최단 경로 알고리즘(1)

7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가진 가중 그래프 G에서 정점 1에서 나머지 정점까지의 최단거리를 구하여라



처리 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
{1, 0}	0	7	5	3	∞	∞	∞
{4, 3}	0	7	5	3	12	∞	10
{3, 5}	0	7	5	3	8	∞	10
{2, 7}	0	7	5	3	8	∞	10
{5, 8}	0	7	5	3	8	∞	9
{7, 9}	0	7	5	3	8	∞	9
{7, 10}	0	7	5	3	8	∞	9
{5, 12}	0	7	5	3	8	∞	9



최소 힙

벨만-포드 알고리즘(Bellman-Ford-Moore Algorithm) - 최단 경로 알고리즘(2)

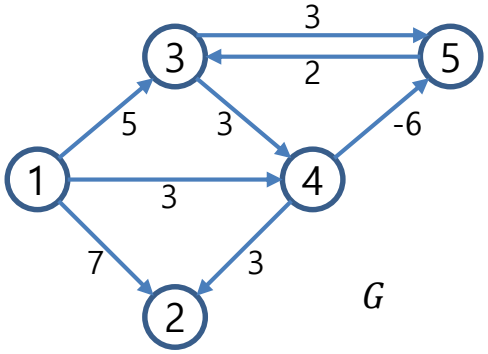
V개의 정점과 E개의 간선을 가진 가중 그래프 G에서
특정 출발 정점(S)에서 부터 다른 모든 정점까지의 최단경로를 구하는 알고리즘.

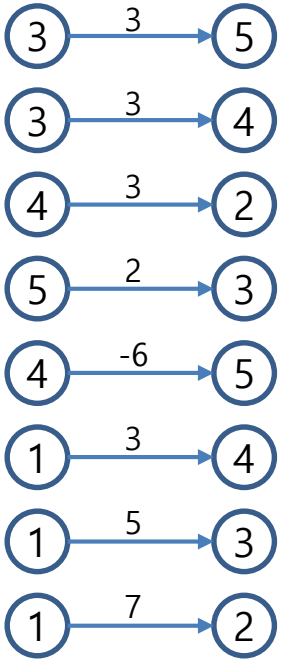
> V개의 정점과 E개의 간선을 가진 가중 그래프에서
> 어떤 정점 A에서 어떤 정점 B까지의 최단거리는 최대 $V-1$개의 간선을 사용한다.
> (=시작 정점 A를 포함하여 최대 V개의 정점을 지난다)

- **특징**
  - 음의 가중치를 가지는 간선도 가능하다.
  - 음의 사이클의 존재 여부를 확인 할 수 있다.
  - 최단거리를 구하기 위해서 $V-1$ 번 E 개의 모든 간선을 확인한다.
  - 음의 사이클 존재 여부를 확인하기 위해서 한 번 더 E개의 간선을 확인한다.
  - 총 연산횟수는 $V \times E$ 이다. 따라서 시간복잡도는 $O(VE)$이다.
  - V번째 모든 간선을 확인하였을 때 거리배열이 갱신되었다면, 그래프 G는 음의 사이클을 가지는 그래프이다.

벨만-포드 알고리즘(Bellman-Ford-Moore Algorithm) - 최단 경로 알고리즘(2)

5개의 정점과 8개의 간선으로 이루어진 가진 가중 그래프 G에서 정점 1에서 나머지 정점까지의 최단거리를 구하여라.



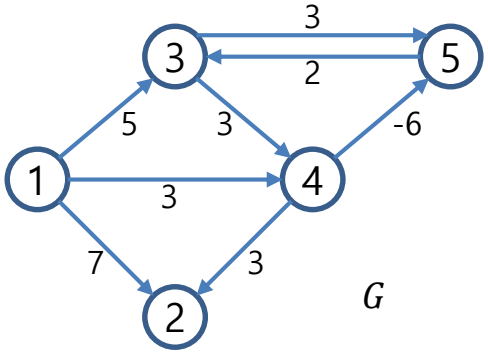


간선 사용 횟수 \ K	1	2	3	4	5
초기상태	0	∞	∞	∞	∞

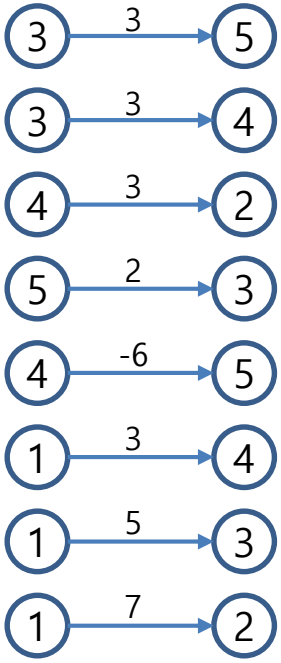
$D[K]$: 출발 정점 S에서 정점 K까지의 최단거리를 저장하는 배열

벨만-포드 알고리즘(Bellman-Ford-Moore Algorithm) - 최단 경로 알고리즘(2)

5개의 정점과 8개의 간선으로 이루어진 가진 가중 그래프 G에서 정점 1에서 나머지 정점까지의 최단거리를 구하여라.



G



간선 사용 횟수 \ K	1	2	3	4	5
초기상태	0	∞	∞	∞	∞
1	0	7	5	3	∞
2	0	6	5	3	-3
3	0	6	-1	3	-3
4	0	5	-1	2	-4
5	0	5	-2	2	-4

$D[K]$: 출발 정점 S에서 정점 K까지의 최단거리를 저장하는 배열

플로이드-워셜 알고리즘(Floyd-Warshall Algorithm) - 최단 경로 알고리즘(3)

V개의 정점과 E개의 간선을 가진 가중 그래프 G에서 모든 정점 사이의 최단경로를 구하는 알고리즘

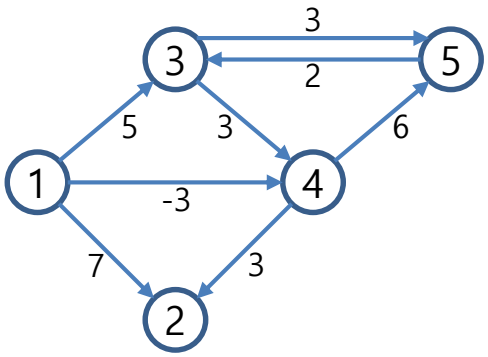
어떤 두 정점 사이의 최단 경로는 어떤 경유지(K)를 거치거나 거치지 않는 경로 중 하나이다.
즉, 정점 A와 정점 B 사이의 최단 경로는 A-B 이거나 A-K-B 이다.

만약 경유지(K)를 거친다면 최단 경로를 이루는 부분 경로 역시 최단 경로이다.
다시 말해, 만약 A-B 의 최단경로가 A-K-B 라면 A-K 와 K-B 도 각각 최단 경로이다.

- **특징**
  - 순환만 없다면 음수 가중치도 가능하다.
  - 기본적으로 동적계획법으로 접근한다.
  - 모든 가능한 경유지에 대해서 모든 정점에서 모든 정점으로 가는 최단거리를 확인하므로 연산횟수는 V^3이고, 따라서 시간복잡도는 $O(V^3)$이다.

플로이드-워셜 알고리즘(Floyd-Warshall Algorithm) - 최단 경로 알고리즘(3)

5개의 정점과 8개의 간선으로 이루어진 가진 가중 그래프 G에서 모든 정점 쌍에 대한 최단거리를 구하여라.



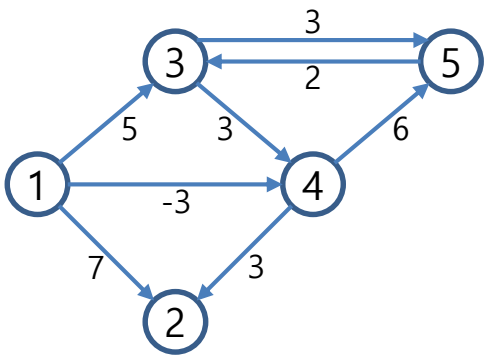
G

I \ J	1	2	3	4	5
1	0	∞	∞	∞	∞
2	∞	0	∞	∞	∞
3	∞	∞	0	∞	∞
4	∞	∞	∞	0	∞
5	∞	∞	∞	∞	0

$D[I][J]$: 정점 I에서 정점 J까지의 최단거리를 저장하는 배열

플로이드-워셜 알고리즘(Floyd-Warshall Algorithm) - 최단 경로 알고리즘(3)

5개의 정점과 8개의 간선으로 이루어진 가진 가중 그래프 G에서 모든 정점 쌍에 대한 최단거리를 구하여라.



G

I \ J	1	2	3	4	5
1	0	0	5	-3	3
2	∞	0	∞	∞	∞
3	∞	6	0	3	3
4	∞	3	8	0	6
5	∞	8	2	5	0

$D[I][J]$: 정점 I에서 정점 J까지의 최단거리를 저장하는 배열

도로 네트워크

N개의 도시와 그 도시를 연결하는 N-1개의 도로로 이루어진 도로 네트워크가 있다.
모든 도시의 쌍에는 그 도시를 연결하는 유일한 경로가 있고, 각 도로의 길이는 입력으로 주어진다.
총 K개의 도시 쌍이 주어진다.

이 때, 두 도시를 연결하는 경로 상에서 가장 짧은 도로의 길이와 가장 긴 도로의 길이를 구하는 프로그램을 작성하시오.

[입력]
첫 줄에는 테스트 케이스의 수를 나타내는 T(1 ≤ T ≤ 20)가 주어진다.
각 테스트 케이스의 첫 줄에는 N (2 ≤ N ≤ 100,000)이 주어진다.
다음 N-1줄에 도로를 나타내는 세 정수 a, b, c가 주어진다. 도시 a와 도시 b사이에 길이가 c인 도로가 있다는 뜻이다. 도로의 길이는 1,000,000보다 작거나 같은 양의 정수이다.
다음 줄에는 K가 주어진다. (1 ≤ K ≤ 100,000)
다음 K개 줄에는 도시를 나타내는 서로 다른 두 자연수 x와 y가 주어진다.

[출력]
각 테스트 케이스에 대해, K개 줄에 걸쳐 도시 x와 도시 y를 연결하는 경로에서 가장 짧은 도로의 길이와 가장 긴 도로의 길이를 출력한다.

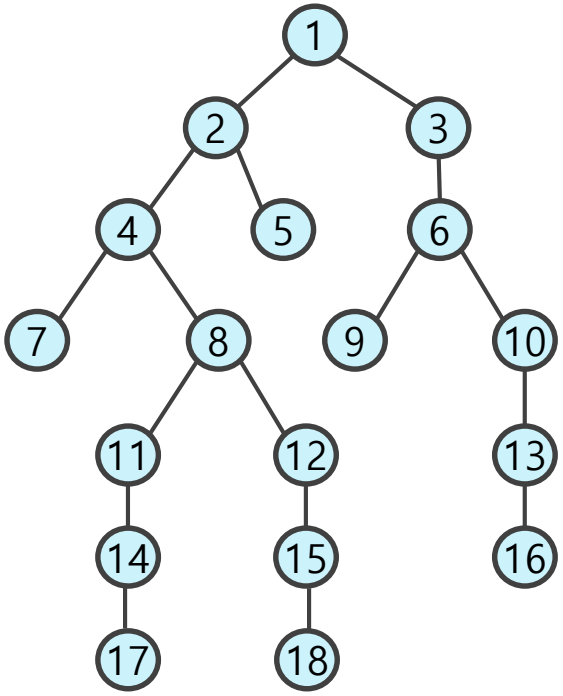
실전문제 접근법

각 도시를 하나의 정점으로
도시를 연결하는 도로들을 간선으로 하는 그래프로 추상화하여 생각해 본다.

문제에서 주어진 그래프의 특징을 파악하여 그래프의 종류를 특정해 본다.

실전문제

1. 주어진 그래프는 N개의 정점에 N-1개의 간선으로 이루어져 있다.
2. 모든 정점 쌍에는 그 도시를 연결하는 유일한 경로가 있다.
3. 연결 그래프
4. 문제의 단서로 주어진 그래프는 트리임을 알 수 있다.



오른쪽 그래프는 트리의 한 예이다.
임의의 두 정점을 선택하여 두 정점을 연결하는 단순 경로는 유일하다.

몇 가지 예를 보면 Path(7,12)={7,**4**,8,12}, Path(14,5)={14,11,8,4,**2**,5}

예를 통해서 우리는 임의의 두 정점을 잇는 단순경로는 두 정점의 조상들 중 최소 공통 조상을 반드시 지나는 경로임을 알 수 있다.
결국 LCA알고리즘을 이용해 두 정점의 경로를 파악 할 수 있고,
또한, 경로 상의 가장 긴 간선의 길이와 가장 짧은 간선의 길이도 구할 수 있다.

특히 이 문제의 경우 정점의 개수가 10만개 LCA 질의가 10만개 이므로 최악의 경우 O(10만 X 10만) = O(1조)로 DP를 이용한 LCA를 이용하지 않으면 제한 시간 내에 문제를 해결 할 수 없다.

SAMSUNG 삼성SDS

감사합니다