그래프 3

최백준 choi@startlink.io

오일러 회로

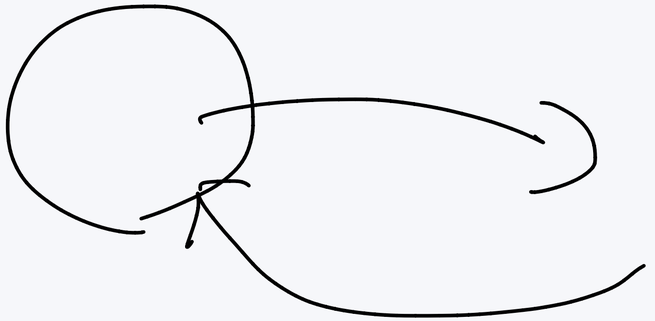
오일러 회로

Euler Circuit

- 그래프의 모든 간선을 한 번씩 통과하는 사이클
- 한붓그리기를 말한다.
- 시작점으로 돌아와야 한다

오일러 회로

Euler Circuit



- 오일러 회로가 존재하기 위한 조건
    1. 그래프는 연결되어 있어야 한다
    2. 모든 정점의 차수는 짝수이다
- 차수가 홀수인 정점이 2개인 경우에는 오일러 경로가 있다. (그 두 정점이 시작점과 끝점)

오일러 회로

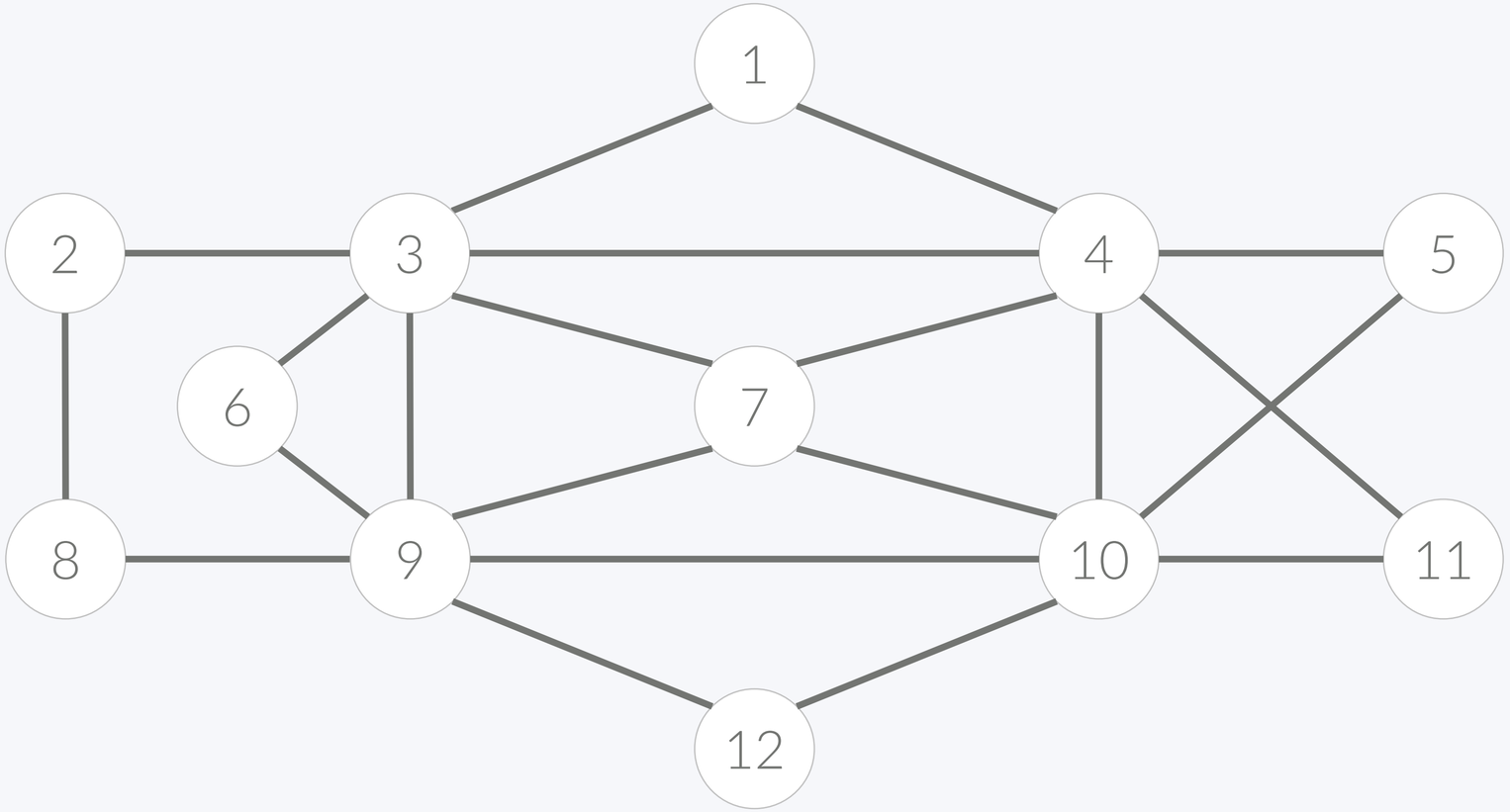
Euler Circuit

1. 임의의 정점 v에서 탐색을 시작해서 v로 돌아올 때 까지의 경로 p를 찾는다
2. 그래프를 모두 탐험하지 않았으면, p에서 탐험하지 않은 간선이 있는 정점 v'를 찾고, 거기서부터 다시 DFS 탐색을 한다. 이 때 얻은 경로를 p' 라고 한다.
3. p와 p'를 합친다.
4. 그래프를 모두 탐색할때까지 2와 3을 반복한다.

오일러 회로

Euler Circuit

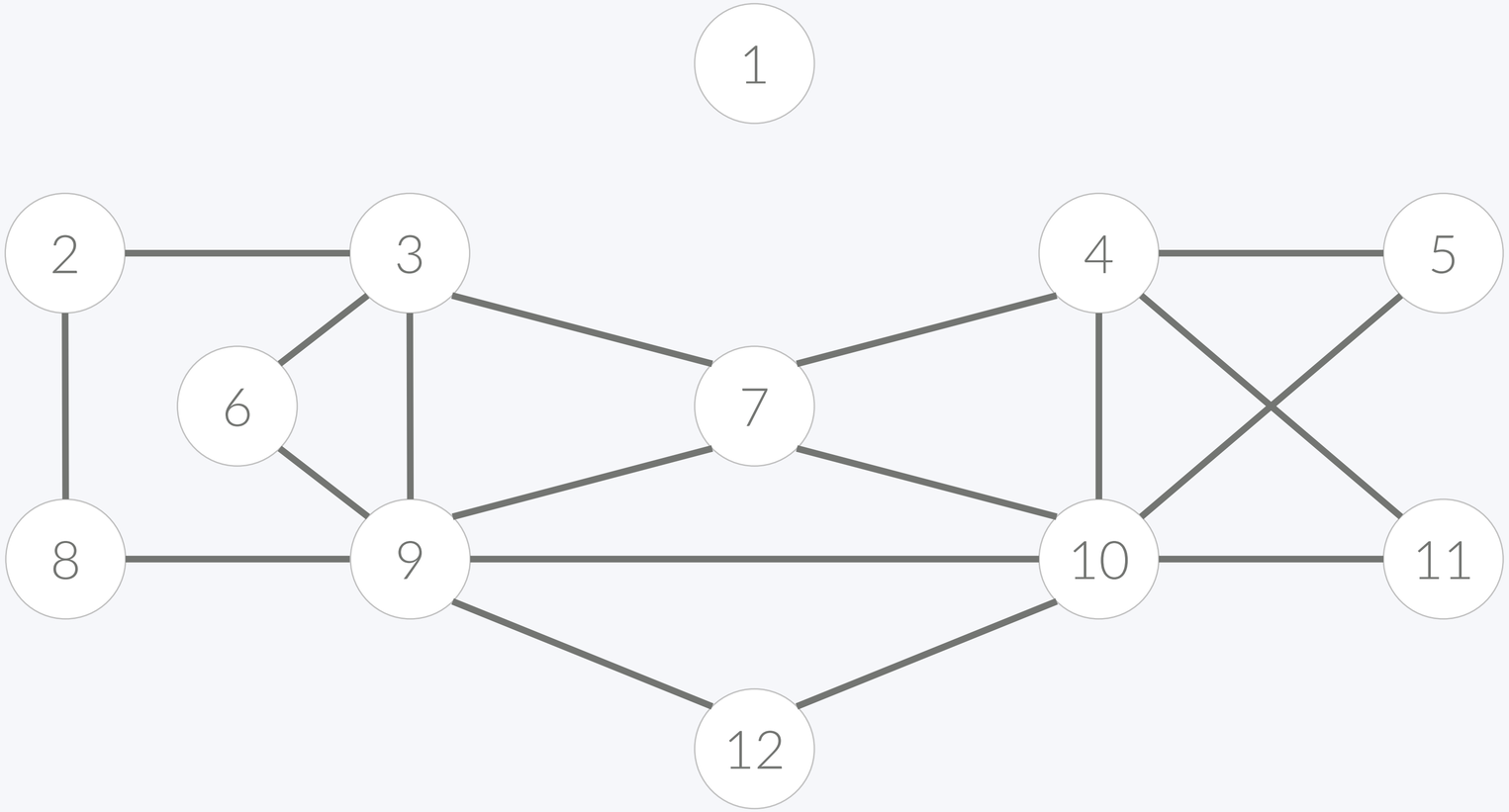
- 1에서 시작
- 1에서 시작하는 사이클을 찾음



오일러 회로

Euler Circuit

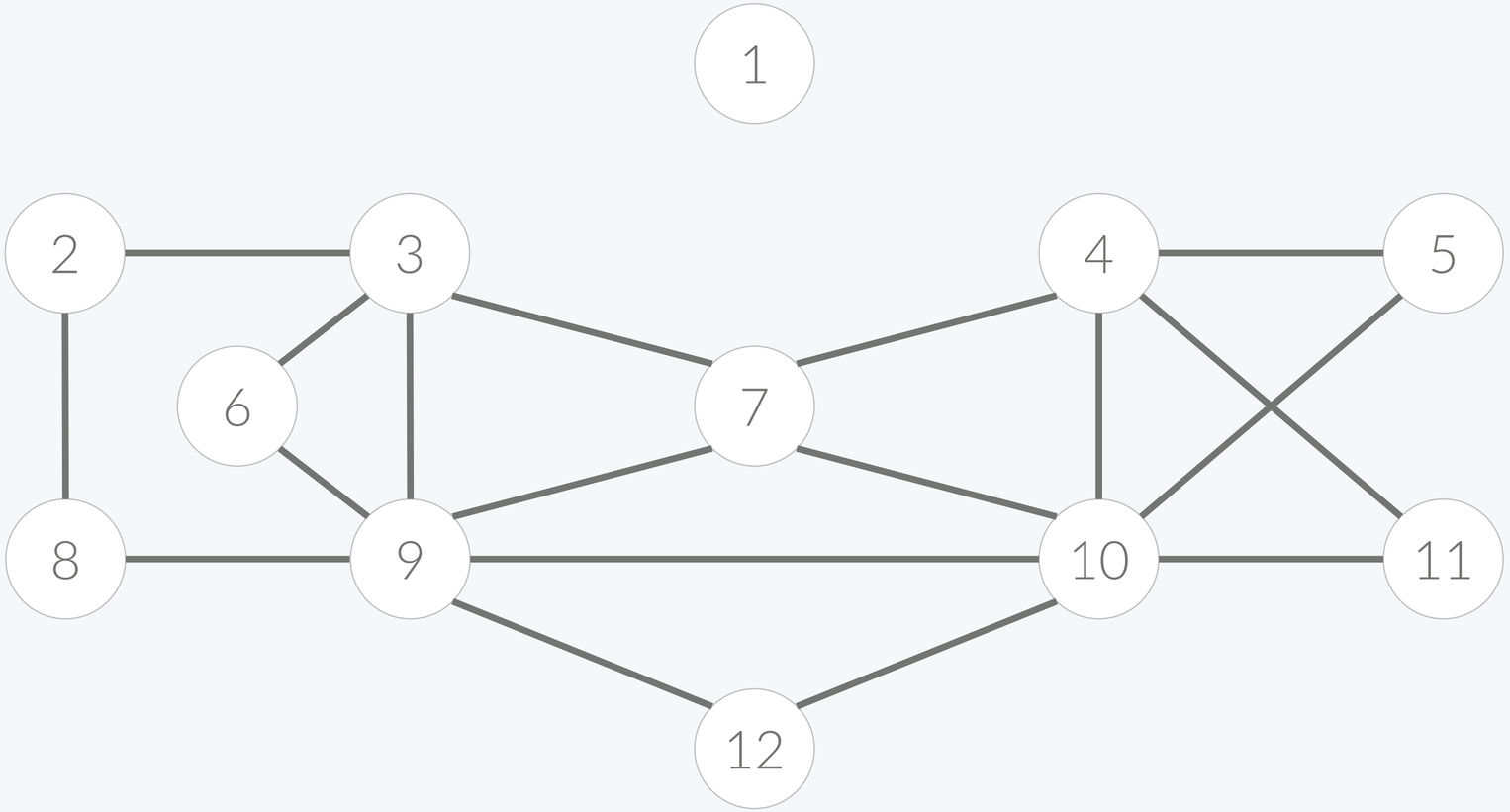
- 1에서 시작하는 사이클을 찾음
- 오일러 경로: 1 3 4 1



오일러 회로

Euler Circuit

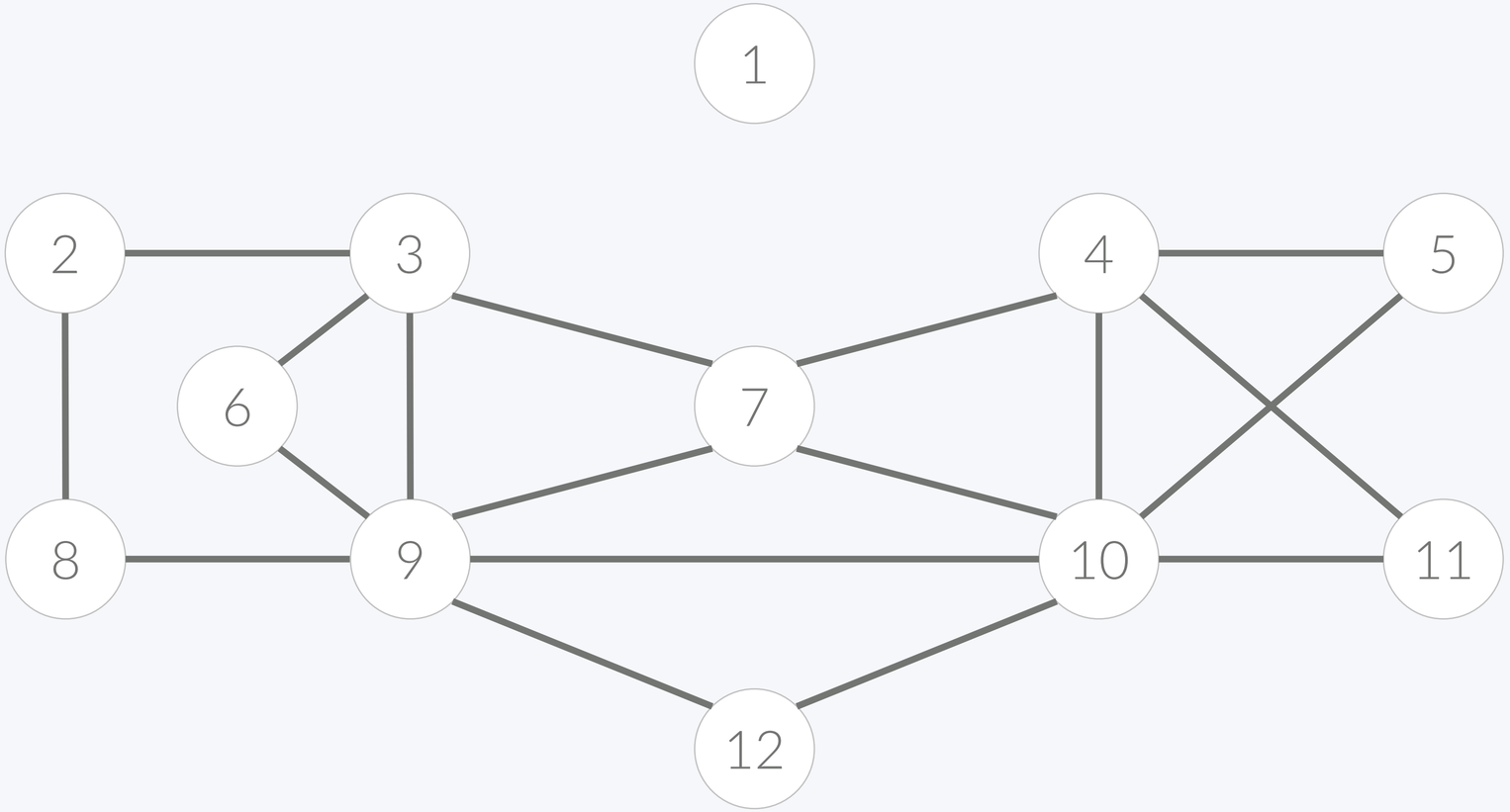
- 1에서 시작하는 사이클이 없음
- 오일러 경로: **1** 3 4 1



오일러 회로

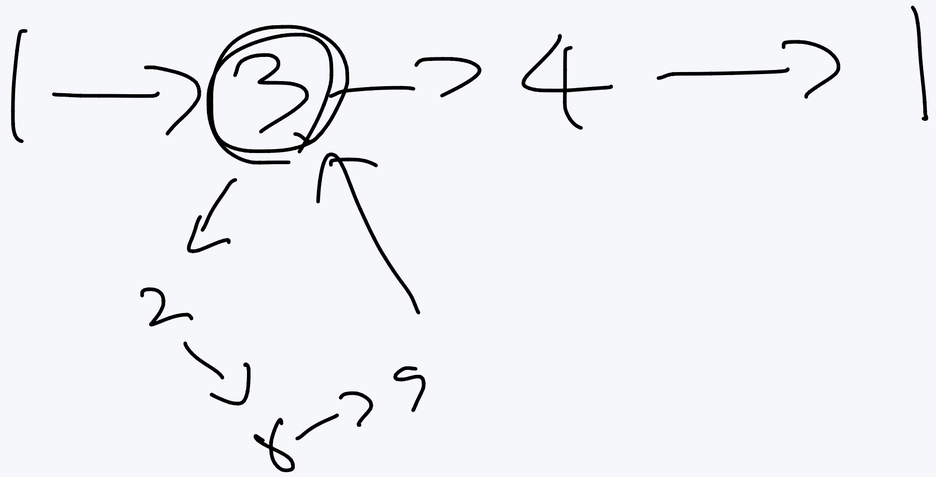
Euler Circuit

- 3에서 시작하는 사이클을 찾음
- 오일러 경로: 1 **3** 4 1

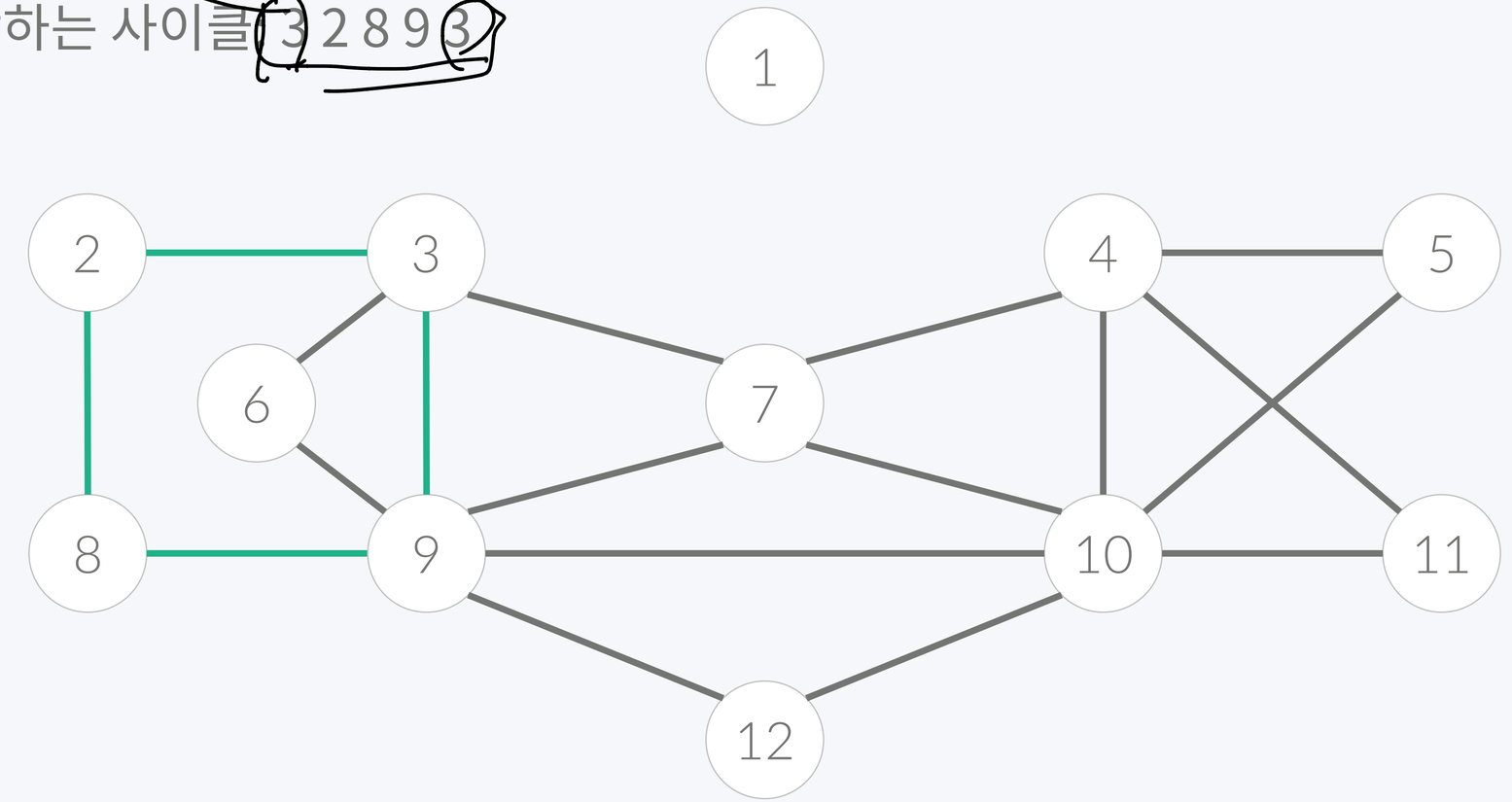


오일러 회로

Euler Circuit



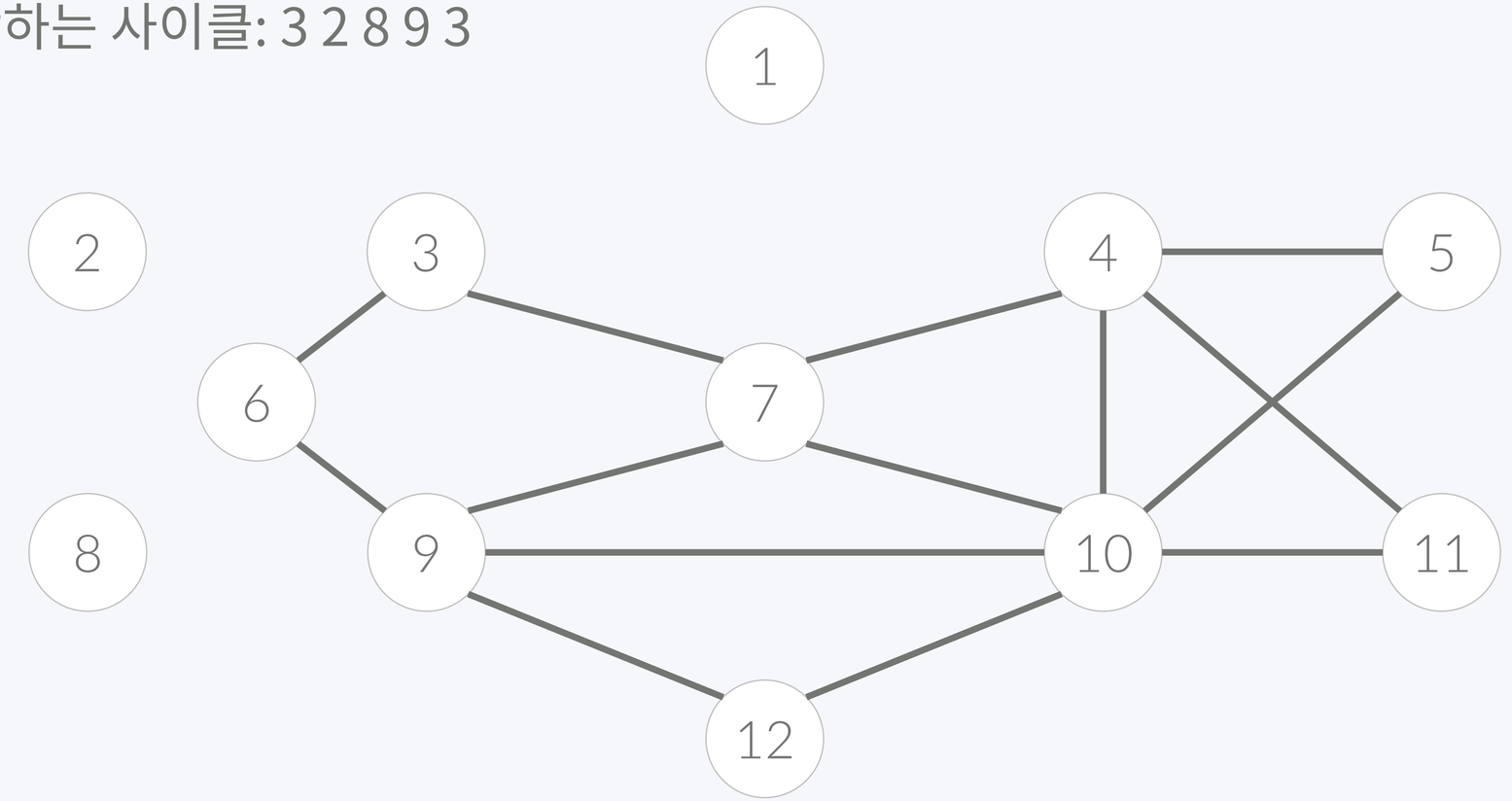
- 3에서 시작하는 사이클을 찾음
- 오일러 경로: 1 **3** 4 1
- 3에서 시작하는 사이클: 3 2 8 9 3



오일러 회로

Euler Circuit

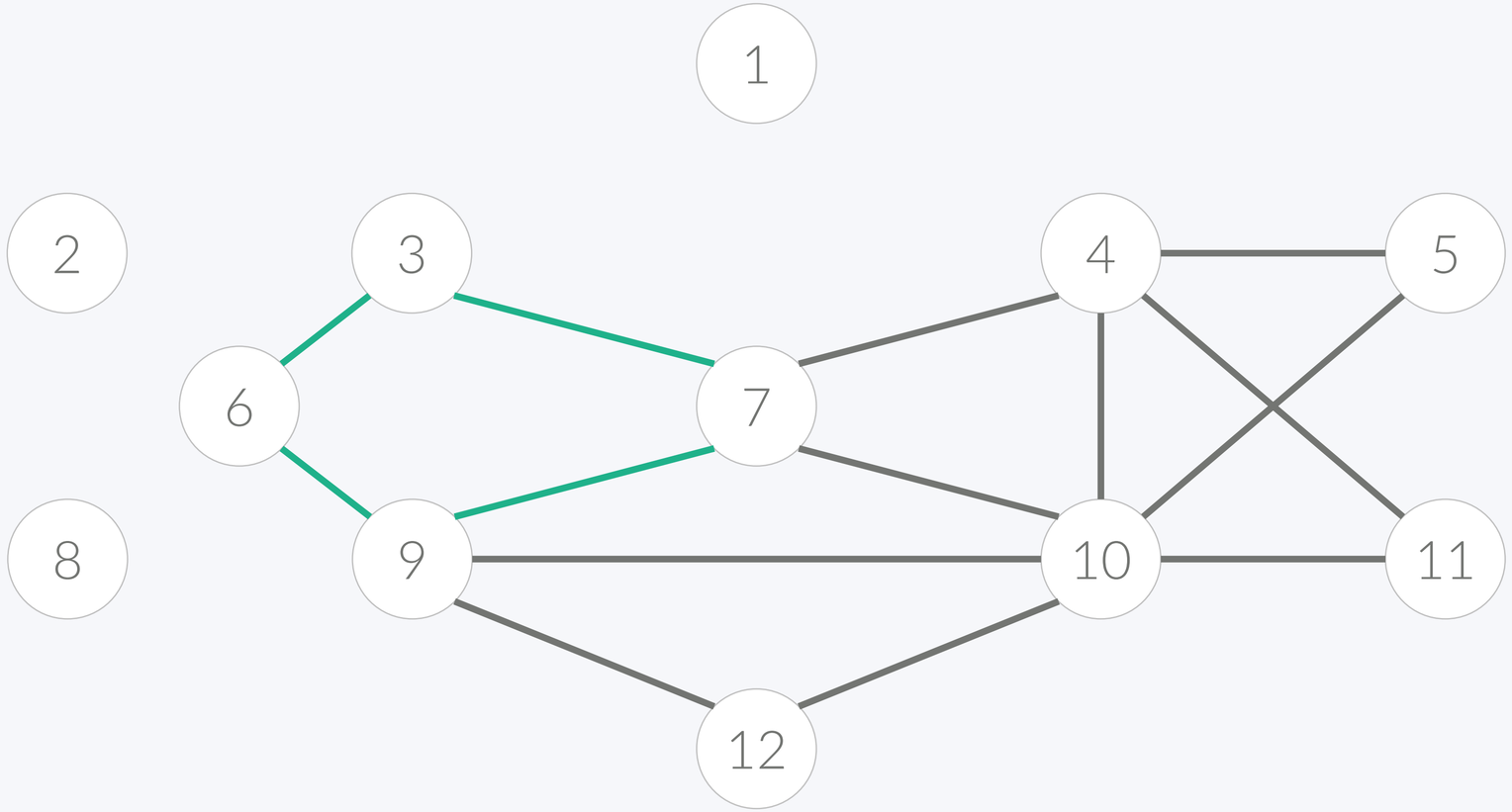
- 3에서 시작하는 사이클을 찾음
- 오일러 경로: 1 **3 2 8 9 3** 4 1
- 3에서 시작하는 사이클: 3 2 8 9 3



오일러 회로

Euler Circuit

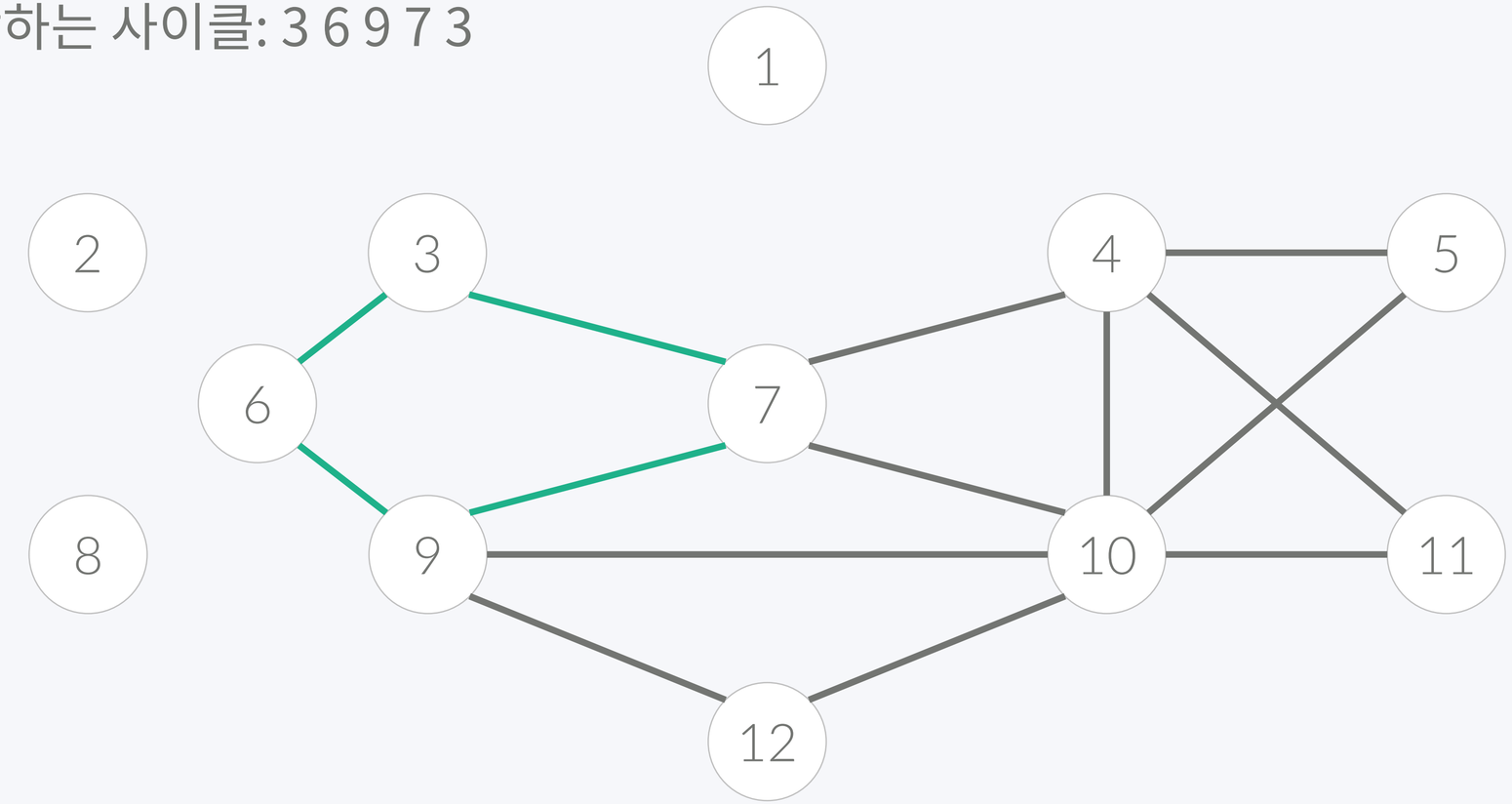
- 3에서 시작하는 사이클을 찾음
- 오일러 경로: 1 **3** 2 8 9 3 4 1



오일러 회로

Euler Circuit

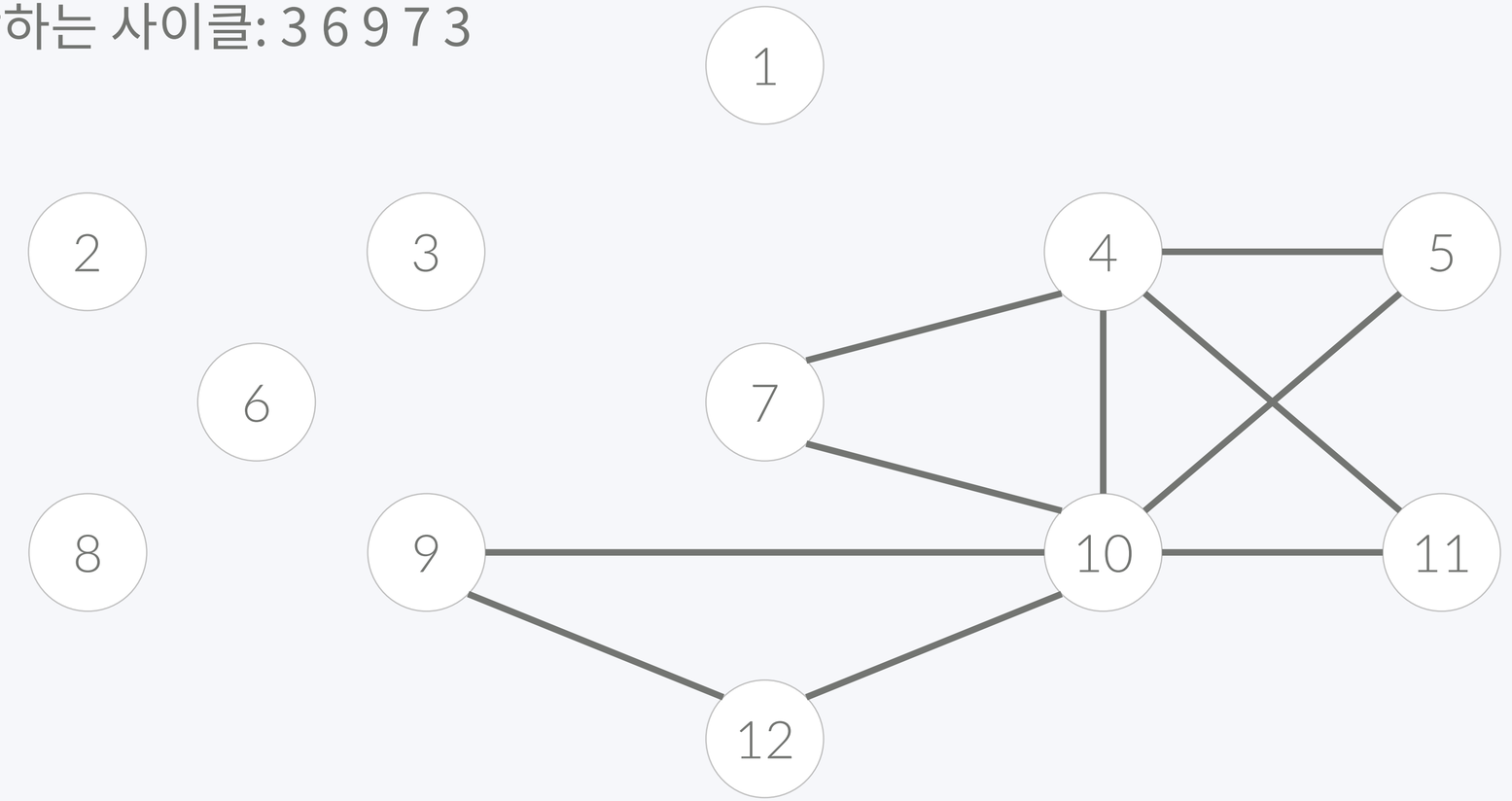
- 3에서 시작하는 사이클을 찾음
- 오일러 경로: 1 **3** 2 8 9 3 4 1
- 3에서 시작하는 사이클: 3 6 9 7 3



오일러 회로

Euler Circuit

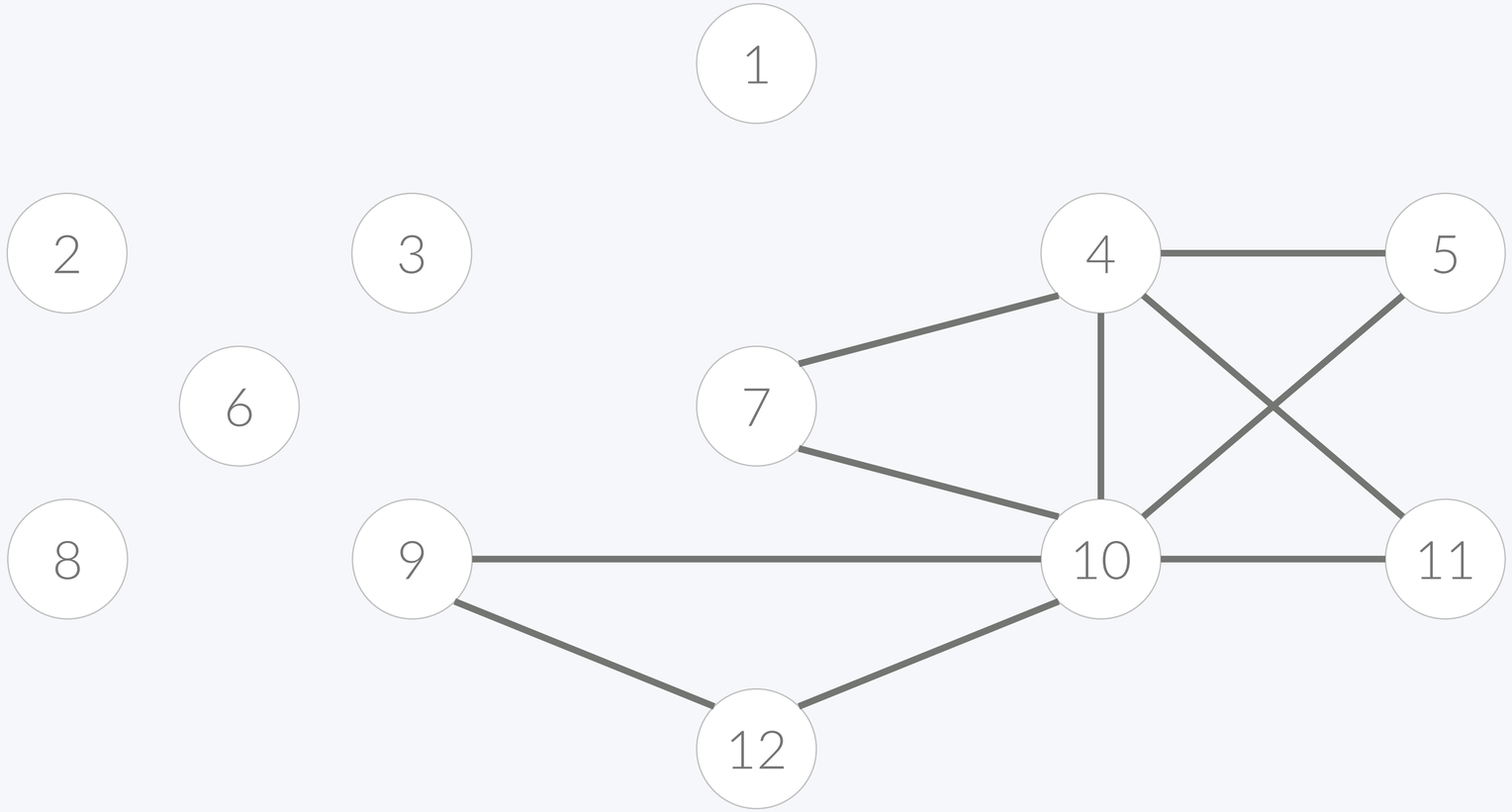
- 3에서 시작하는 사이클을 찾음
- 오일러 경로: 1 **3 6 9 7 3** 2 8 9 3 4 1
- 3에서 시작하는 사이클: 3 6 9 7 3



오일러 회로

Euler Circuit

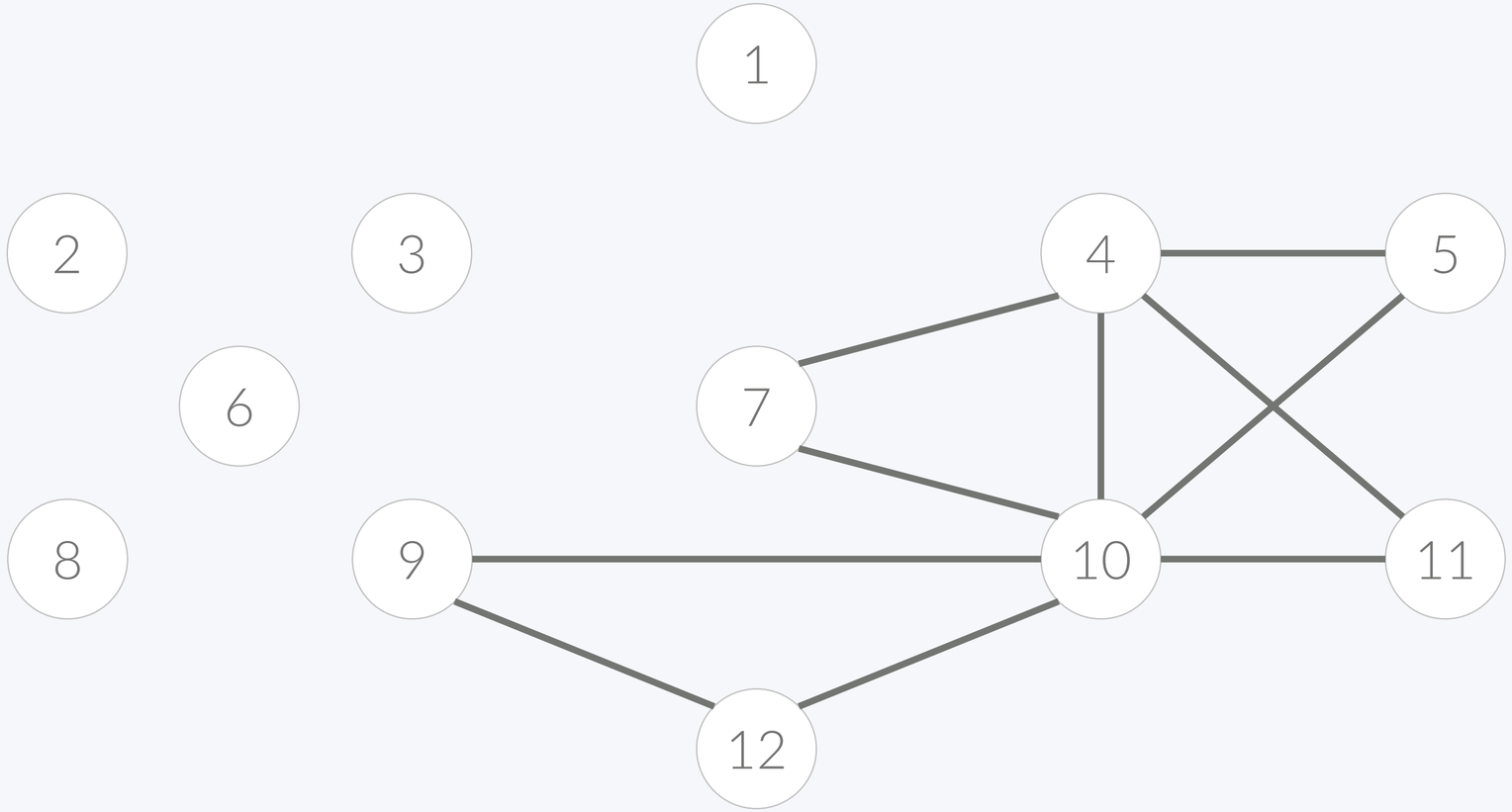
- 3에서 시작하는 사이클이 없음
- 오일러 경로: 1 **3** 6 9 7 3 2 8 9 3 4 1



오일러 회로

Euler Circuit

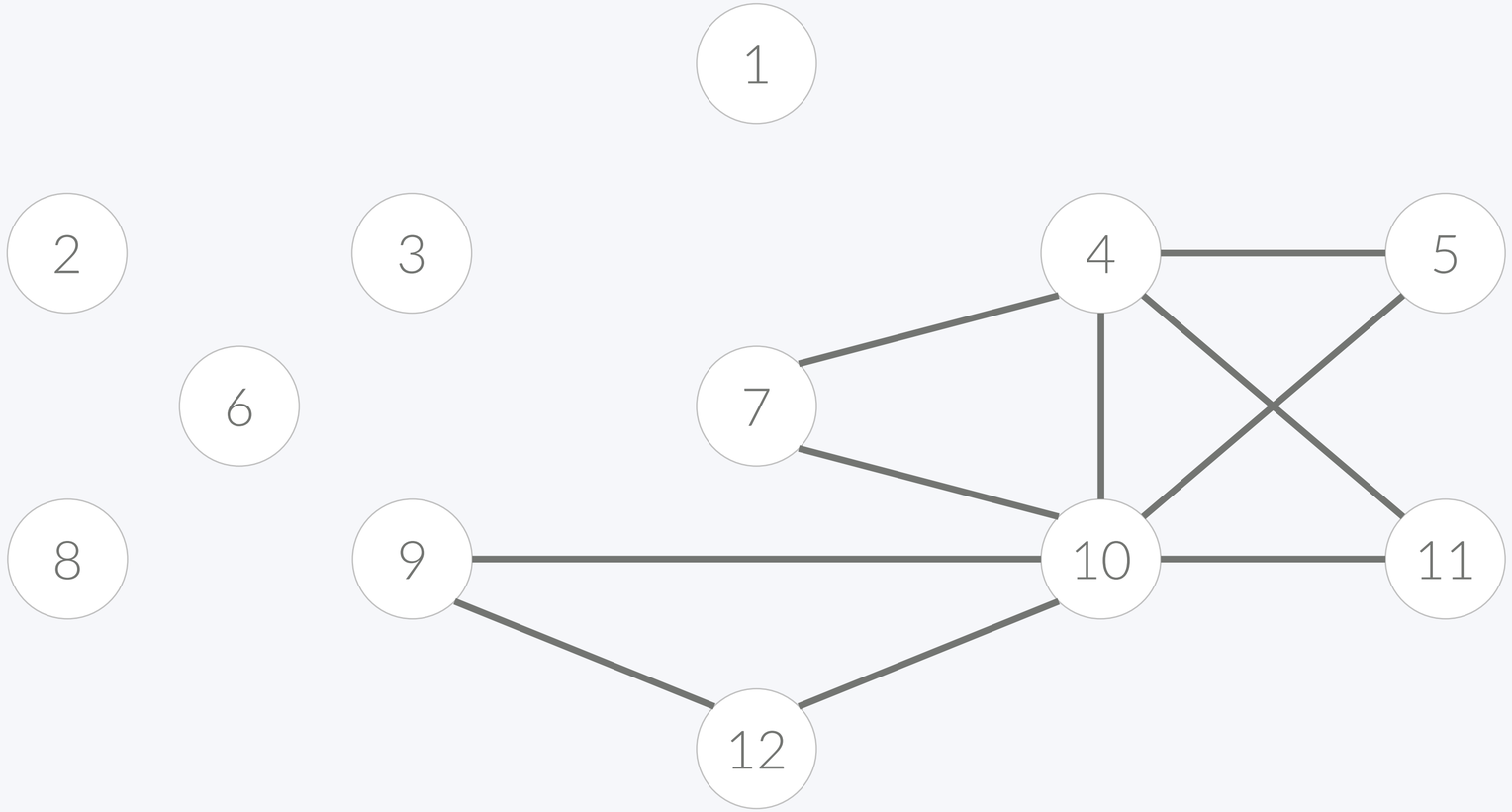
- 6에서 시작하는 사이클이 없음
- 오일러 경로: 1 3 **6** 9 7 3 2 8 9 3 4 1



오일러 회로

Euler Circuit

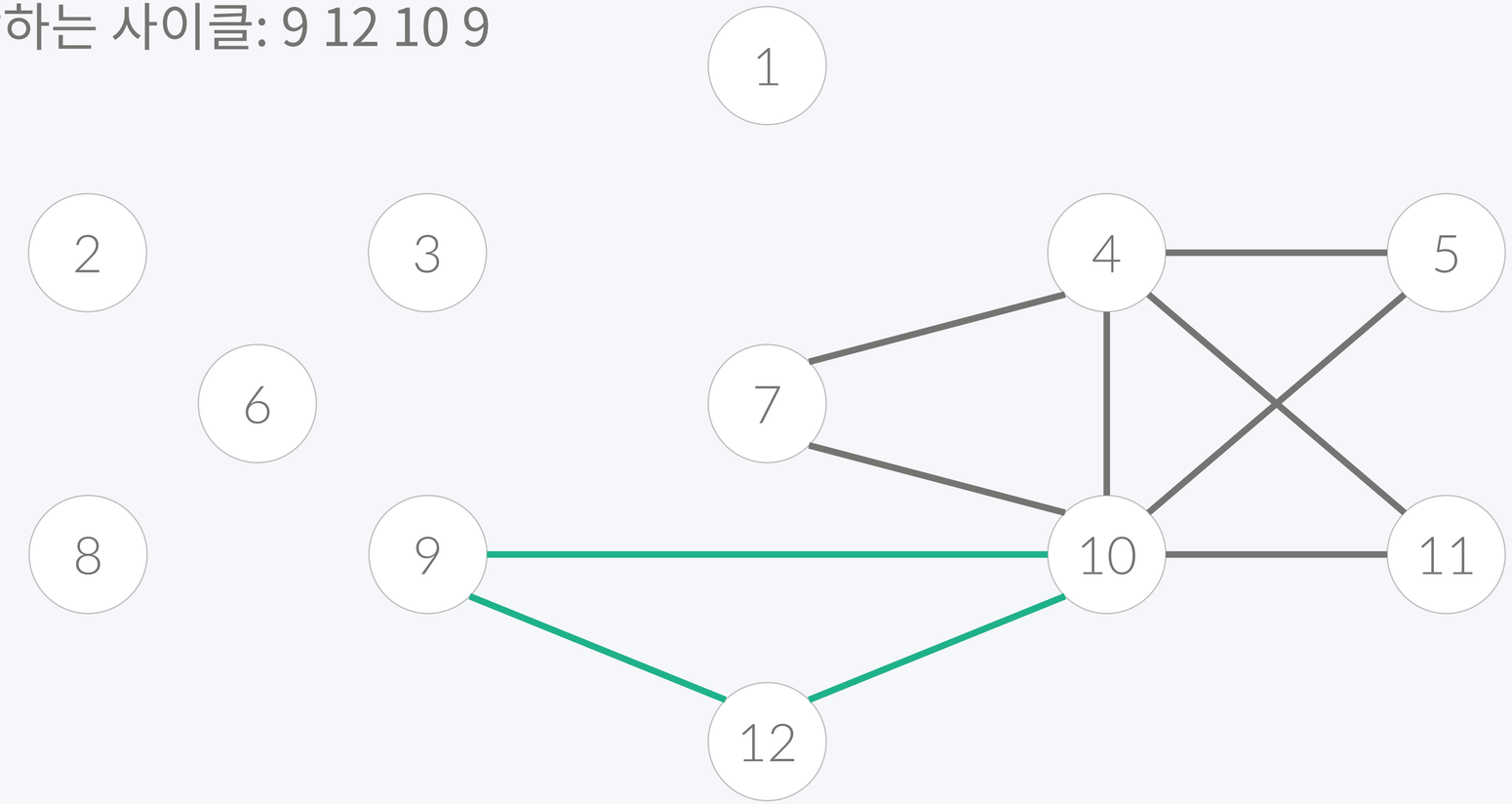
- 9에서 시작하는 사이클을 찾음
- 오일러 경로: 1 3 6 **9** 7 3 2 8 9 3 4 1



오일러 회로

Euler Circuit

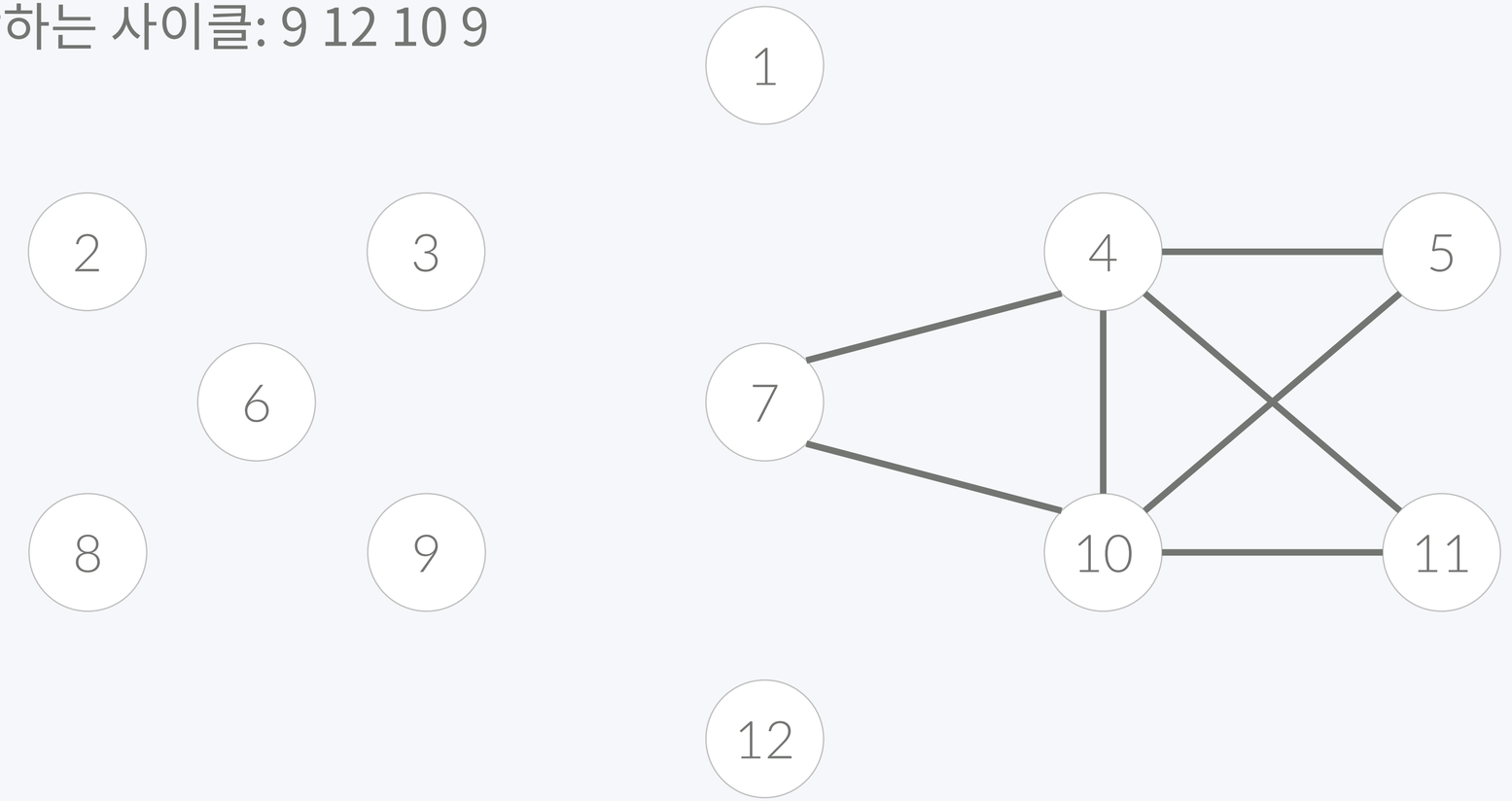
- 9에서 시작하는 사이클을 찾음
- 오일러 경로: 1 3 6 **9** 7 3 2 8 9 3 4 1
- 9에서 시작하는 사이클: 9 12 10 9



오일러 회로

Euler Circuit

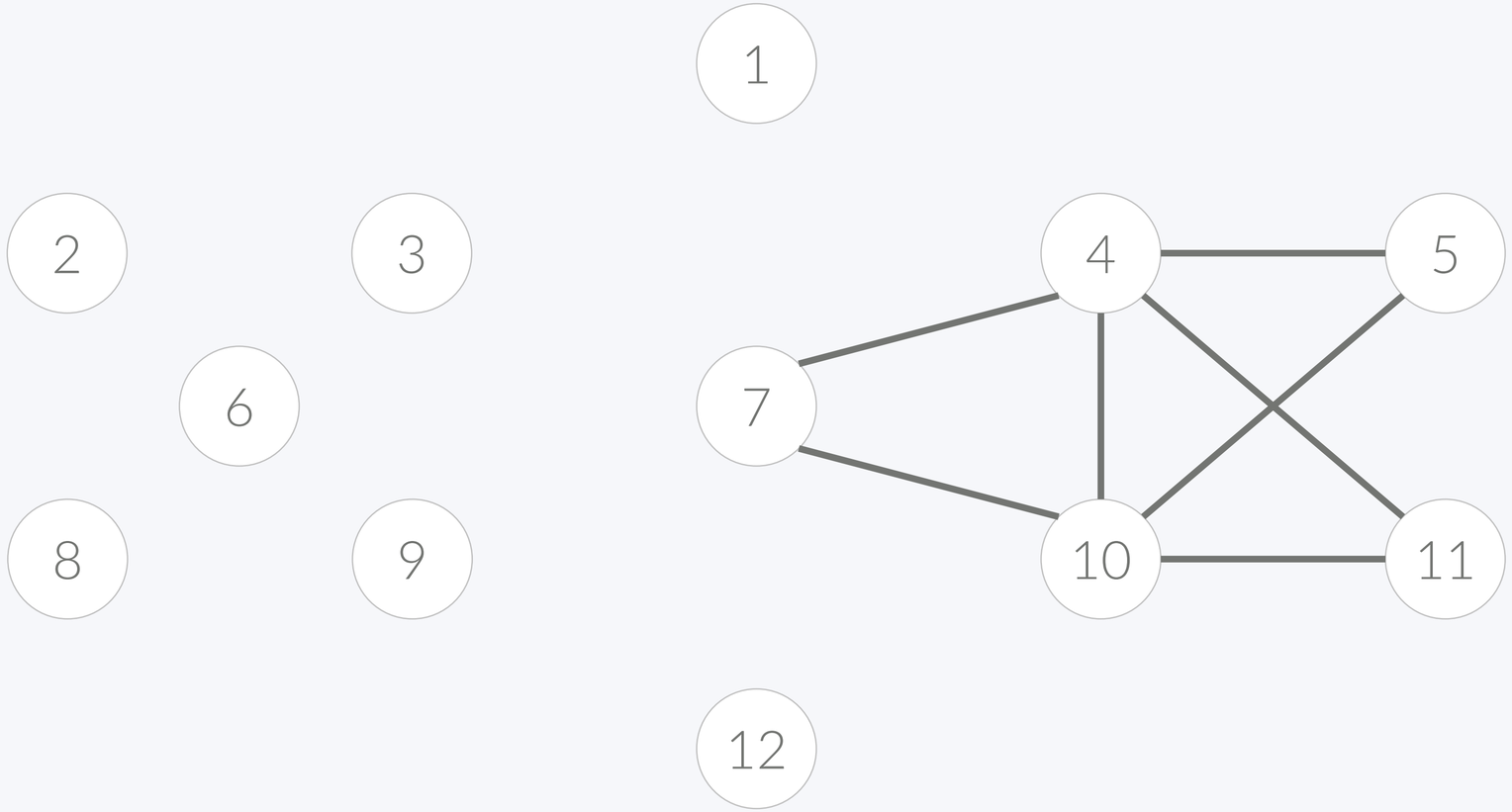
- 9에서 시작하는 사이클을 찾음
- 오일러 경로: 1 3 6 **9 12 10 9** 7 3 2 8 9 3 4 1
- 9에서 시작하는 사이클: 9 12 10 9



오일러 회로

Euler Circuit

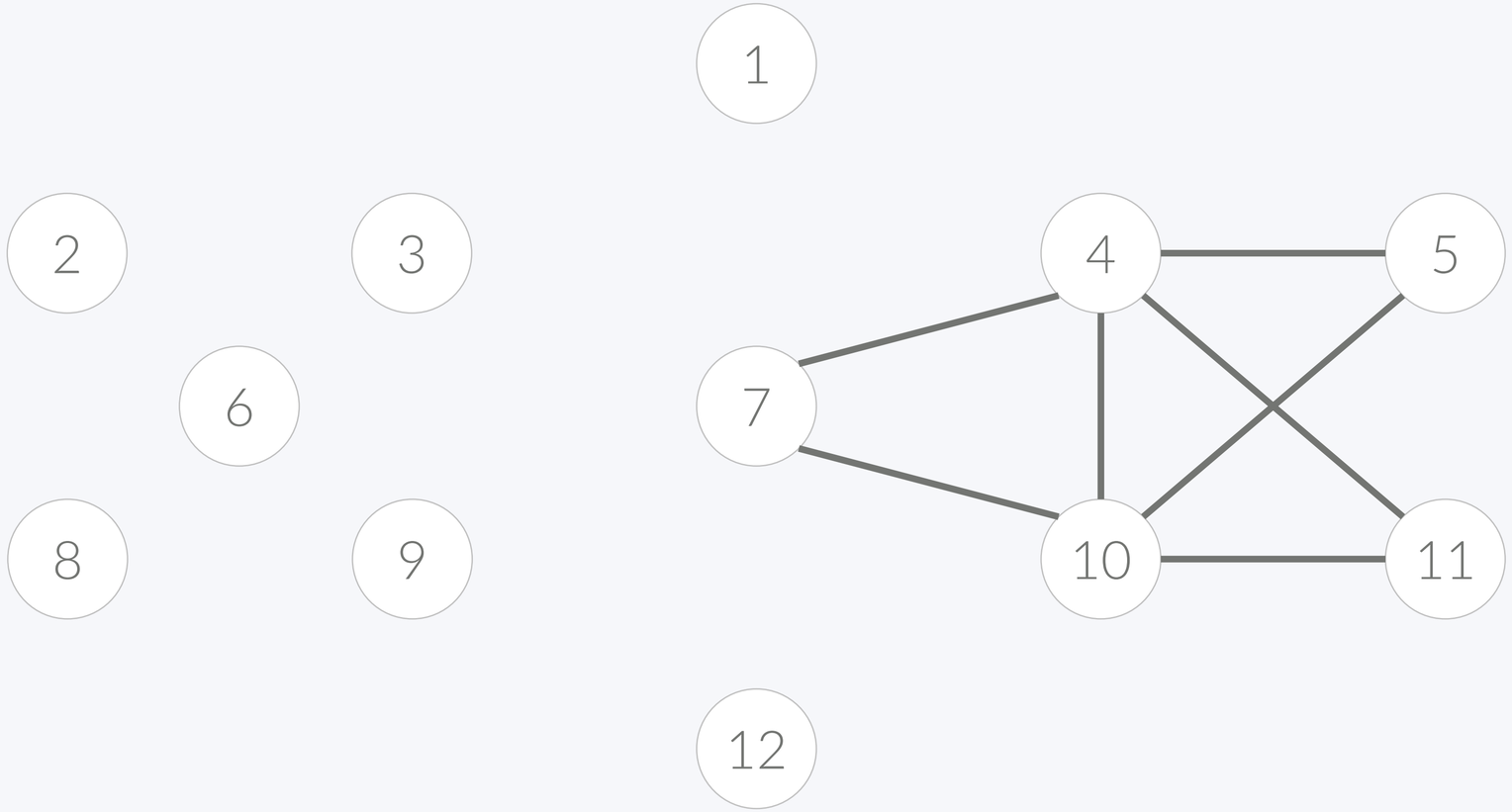
- 9에서 시작하는 사이클이 없음
- 오일러 경로: 1 3 6 **9** 12 10 9 7 3 2 8 9 3 4 1



오일러 회로

Euler Circuit

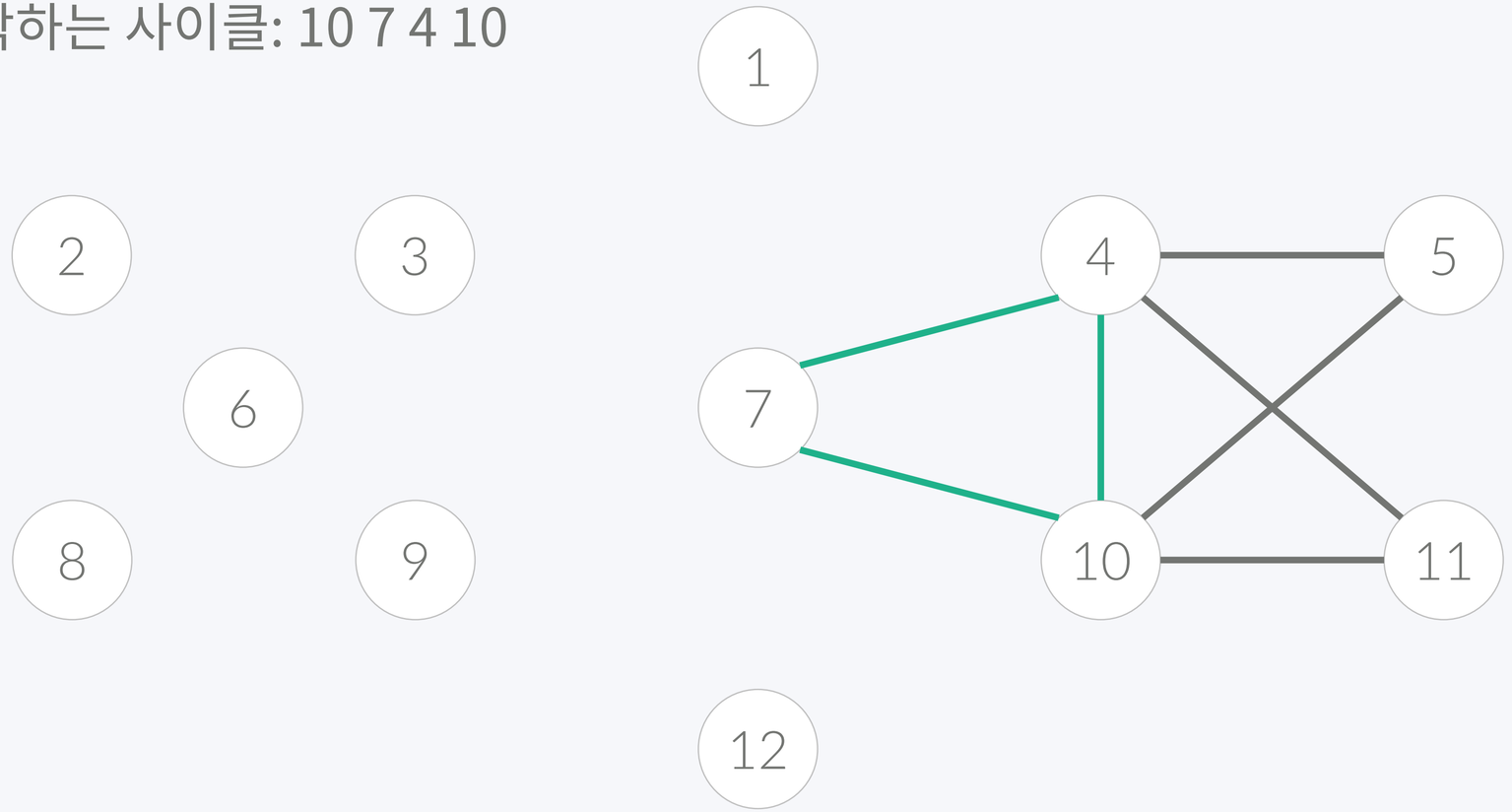
- 12에서 시작하는 사이클이 없음
- 오일러 경로: 1 3 6 9 **12** 10 9 7 3 2 8 9 3 4 1



오일러 회로

Euler Circuit

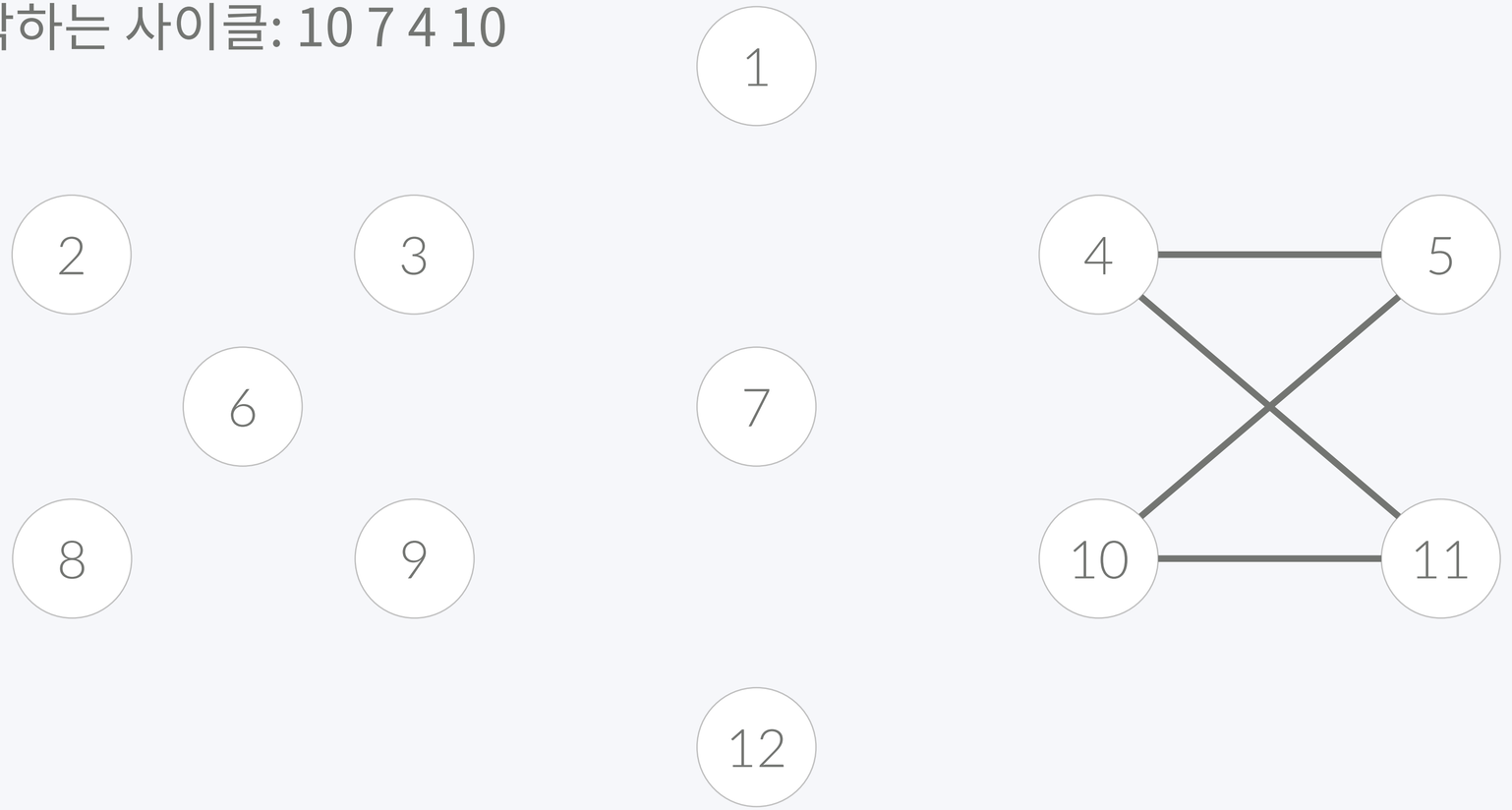
- 10에서 시작하는 사이클을 찾음
- 오일러 경로: 1 3 6 9 12 **10** 9 7 3 2 8 9 3 4 1
- 10에서 시작하는 사이클: 10 7 4 10



오일러 회로

Euler Circuit

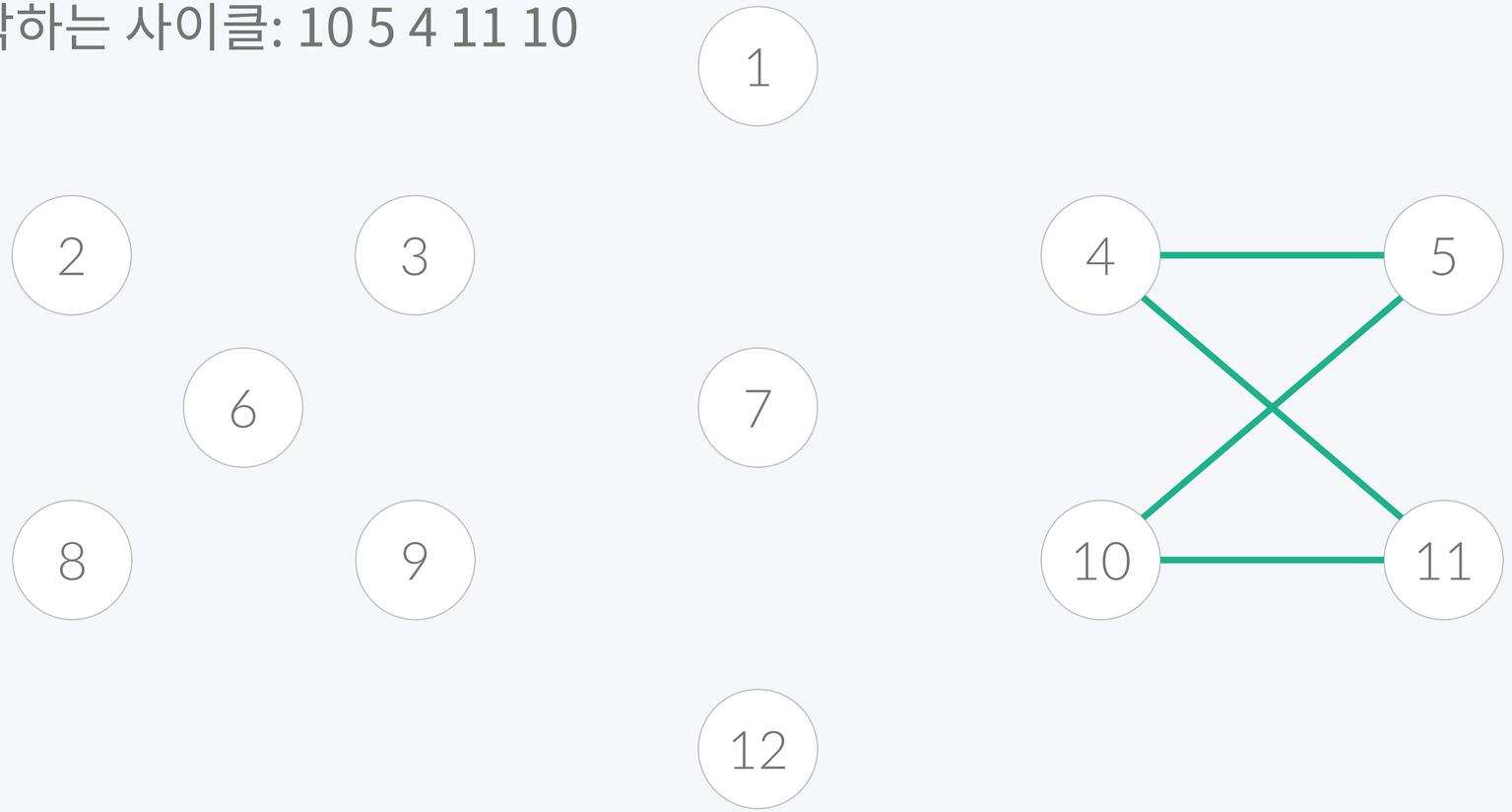
- 10에서 시작하는 사이클을 찾음
- 오일러 경로: 1 3 6 9 12 **10 7 4 10** 9 7 3 2 8 9 3 4 1
- 10에서 시작하는 사이클: 10 7 4 10



오일러 회로

Euler Circuit

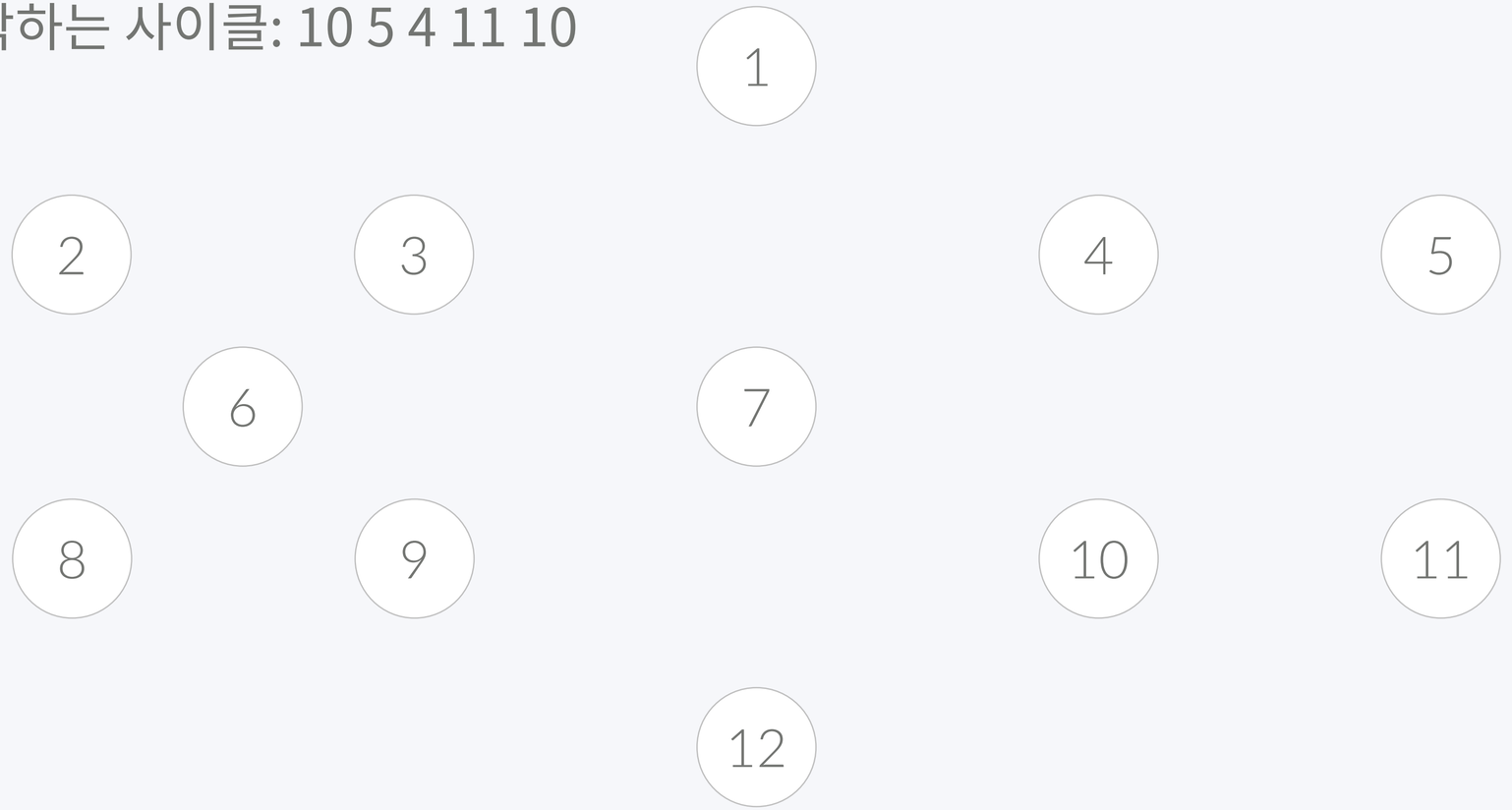
- 10에서 시작하는 사이클을 찾음
- 오일러 경로: 1 3 6 9 12 **10** 7 4 10 9 7 3 2 8 9 3 4 1
- 10에서 시작하는 사이클: 10 5 4 11 10



오일러 회로

Euler Circuit

- 10에서 시작하는 사이클을 찾음
- 오일러 경로: 1 3 6 9 12 **10 5 4 11 10** 7 4 10 9 7 3 2 8 9 3 4 1
- 10에서 시작하는 사이클: 10 5 4 11 10



오일러 회로

https://www.acmicpc.net/problem/1199

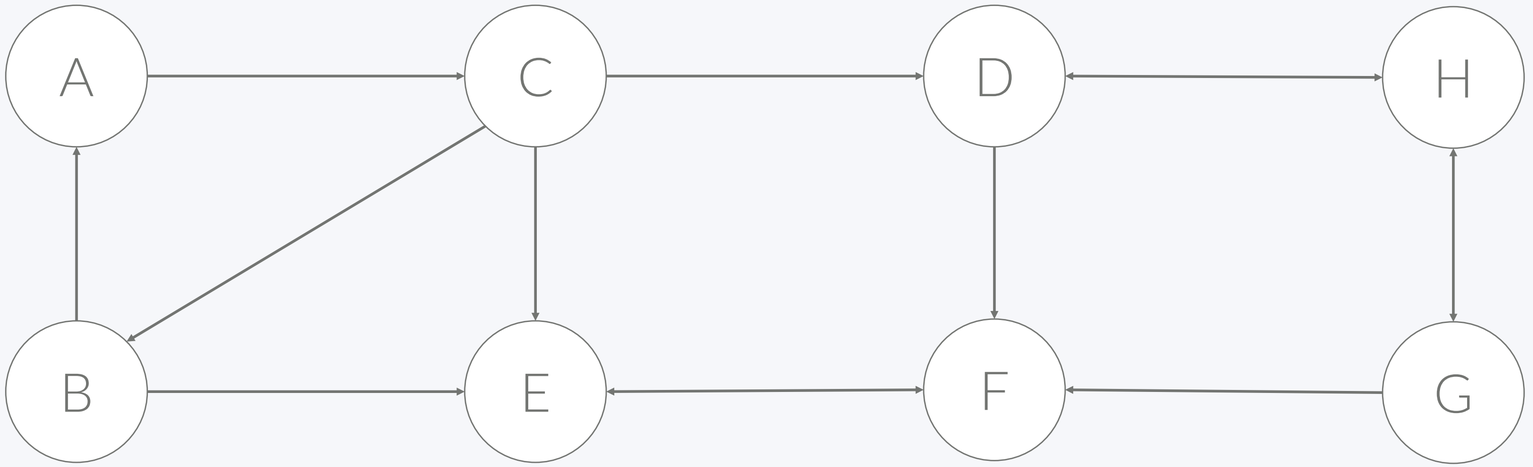
- C/C++: https://gist.github.com/Baekjoon/9d33dbae90864962ff87

SCC

강한 연결 요소

Strongly Connected Component

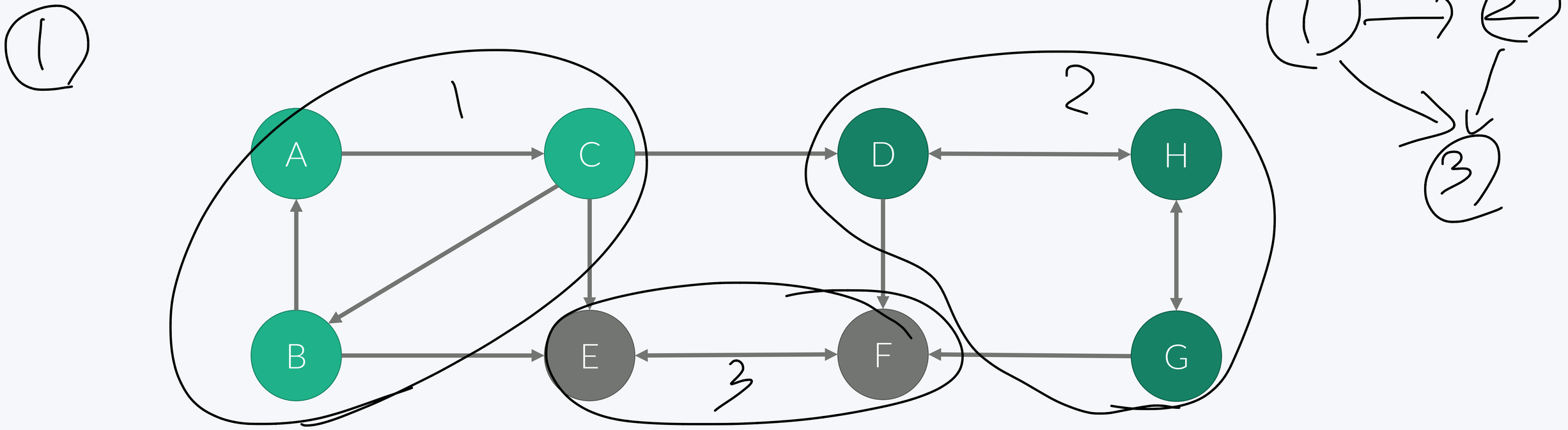
- Strongly Connected: 모든 정점에서 정점으로 이동할 수 있다
- SCC: 그래프를 Strongly Connected되게 서브 그래프로 나눔



강한 연결 요소

Strongly Connected Component

- Strongly Connected: 모든 정점에서 정점으로 이동할 수 있다
- SCC: 그래프를 Strongly Connected되게 서브 그래프로 나눔



강한 연결 요소

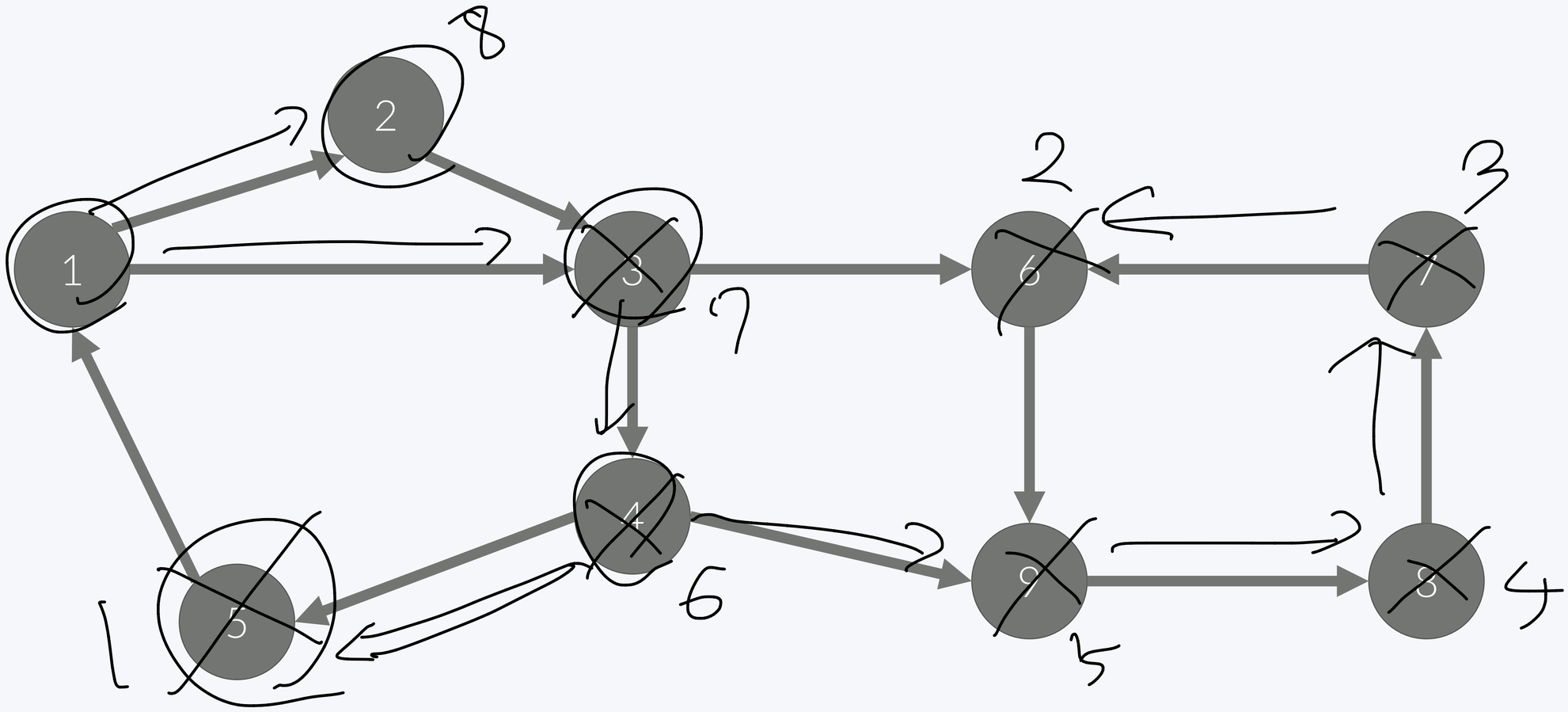
Strongly Connected Component

- 두 알고리즘이 있다
- Kosarju’s algorithm
- Tarjan’s algorithm

Kosaju's Algorithm

강한 연결 요소

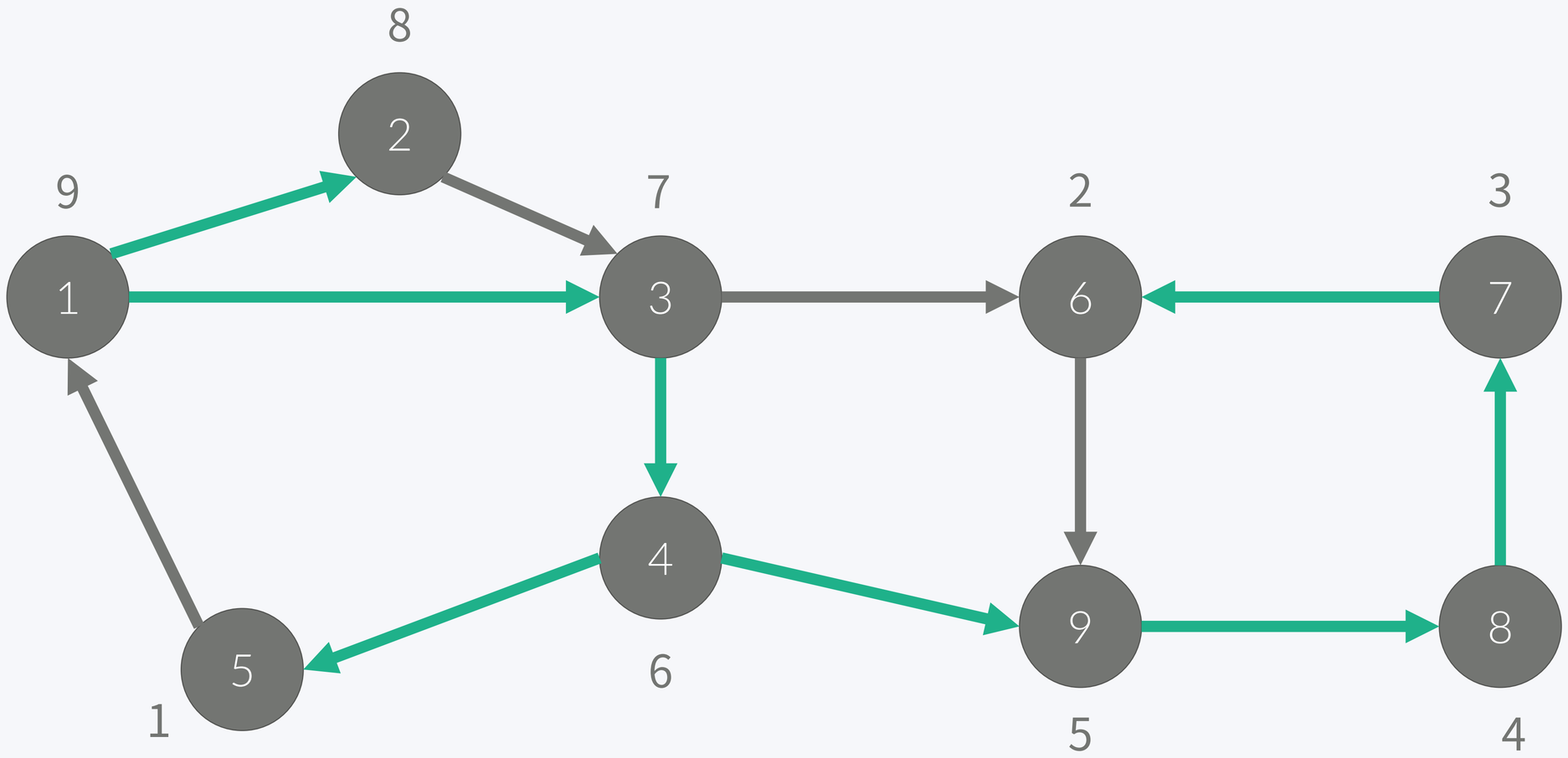
- DFS 탐색을 하면서 스택에서 나온 순서대로 번호를 매긴다



Kosaju's Algorithm

강한 연결 요소

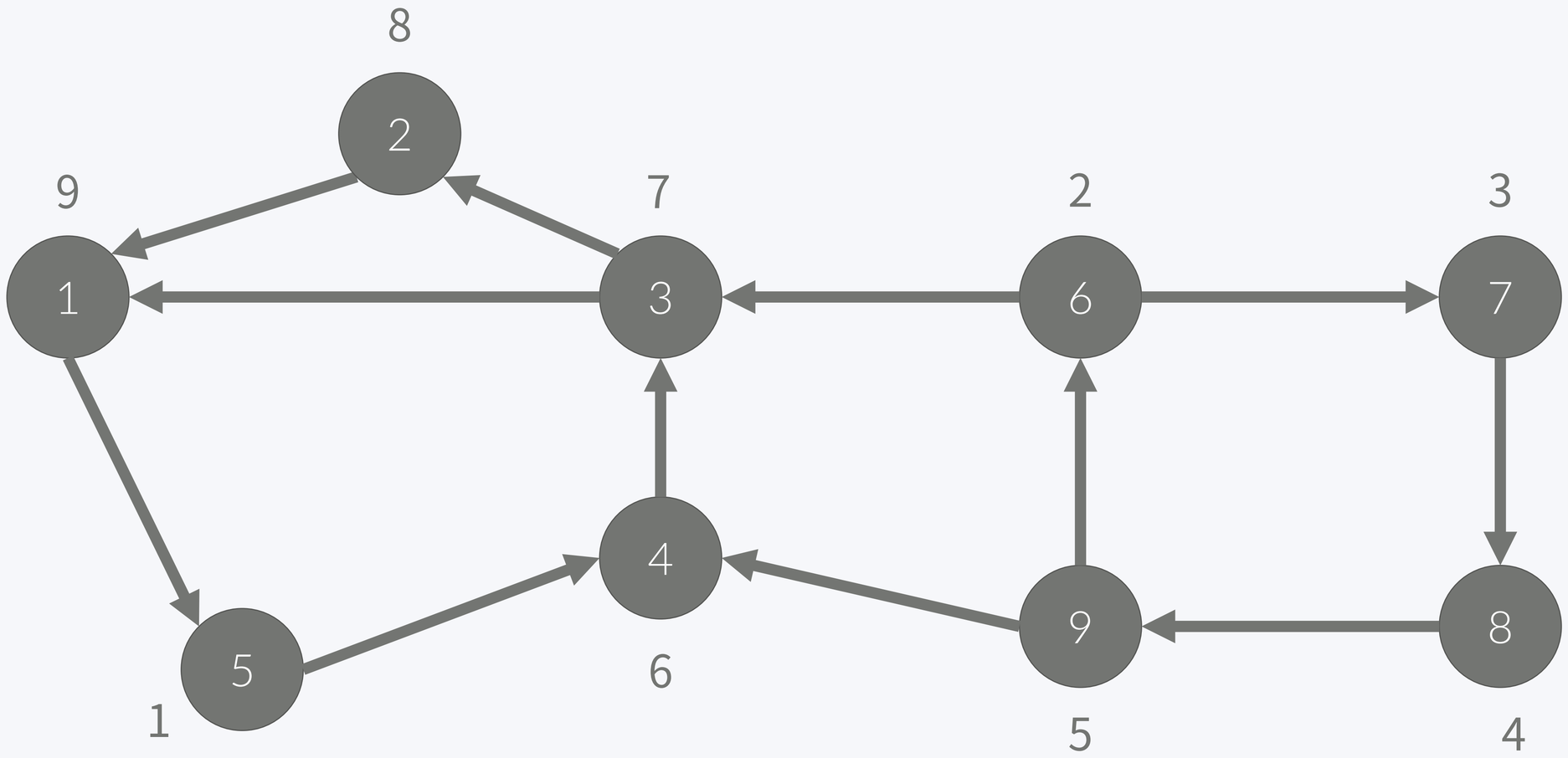
- DFS 탐색을 하면서 스택에서 나온 순서대로 번호를 매긴다



Kosaju's Algorithm

강한 연결 요소

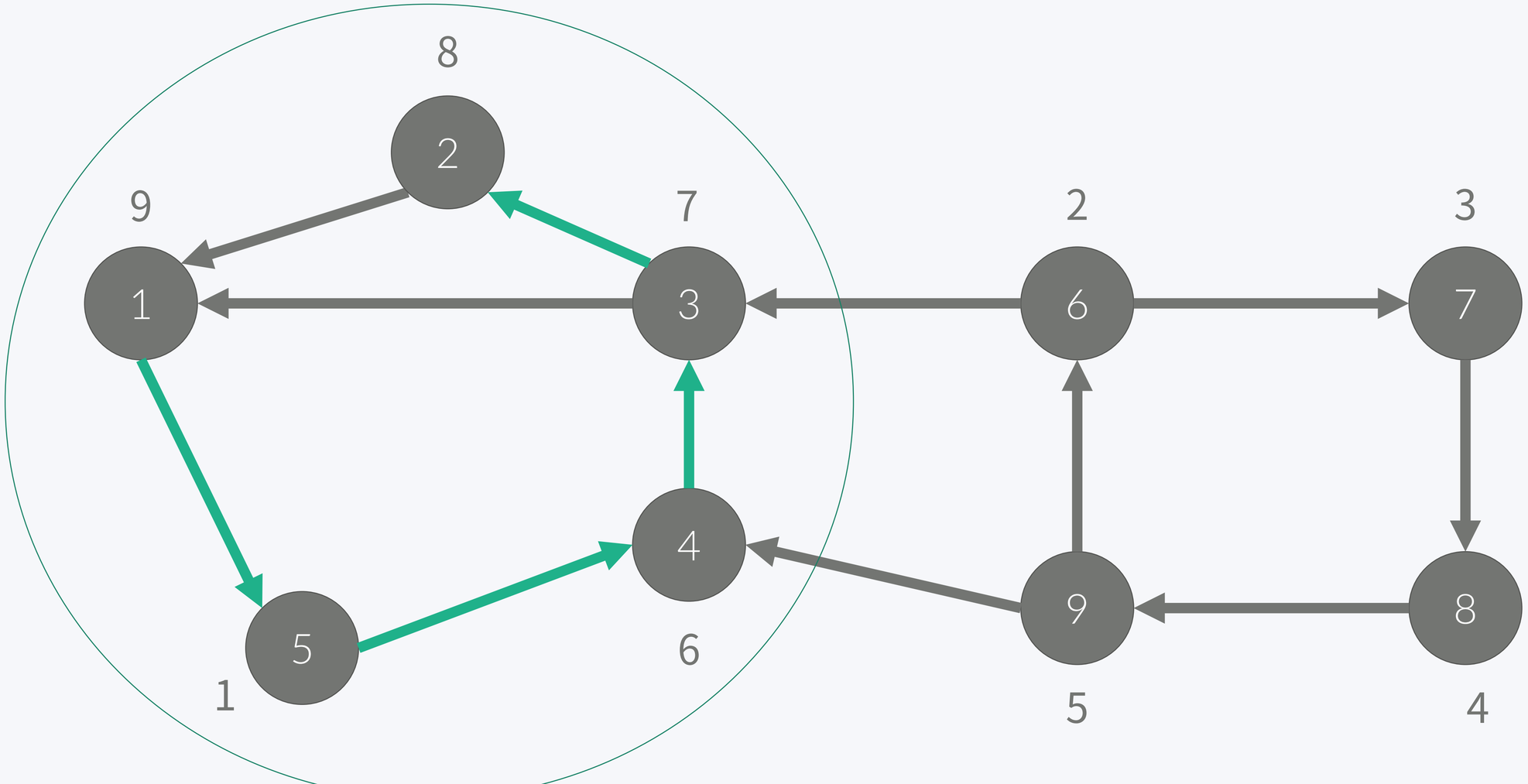
- 간선의 방향을 모두 뒤집는다



Kosaju's Algorithm

강한 연결 요소

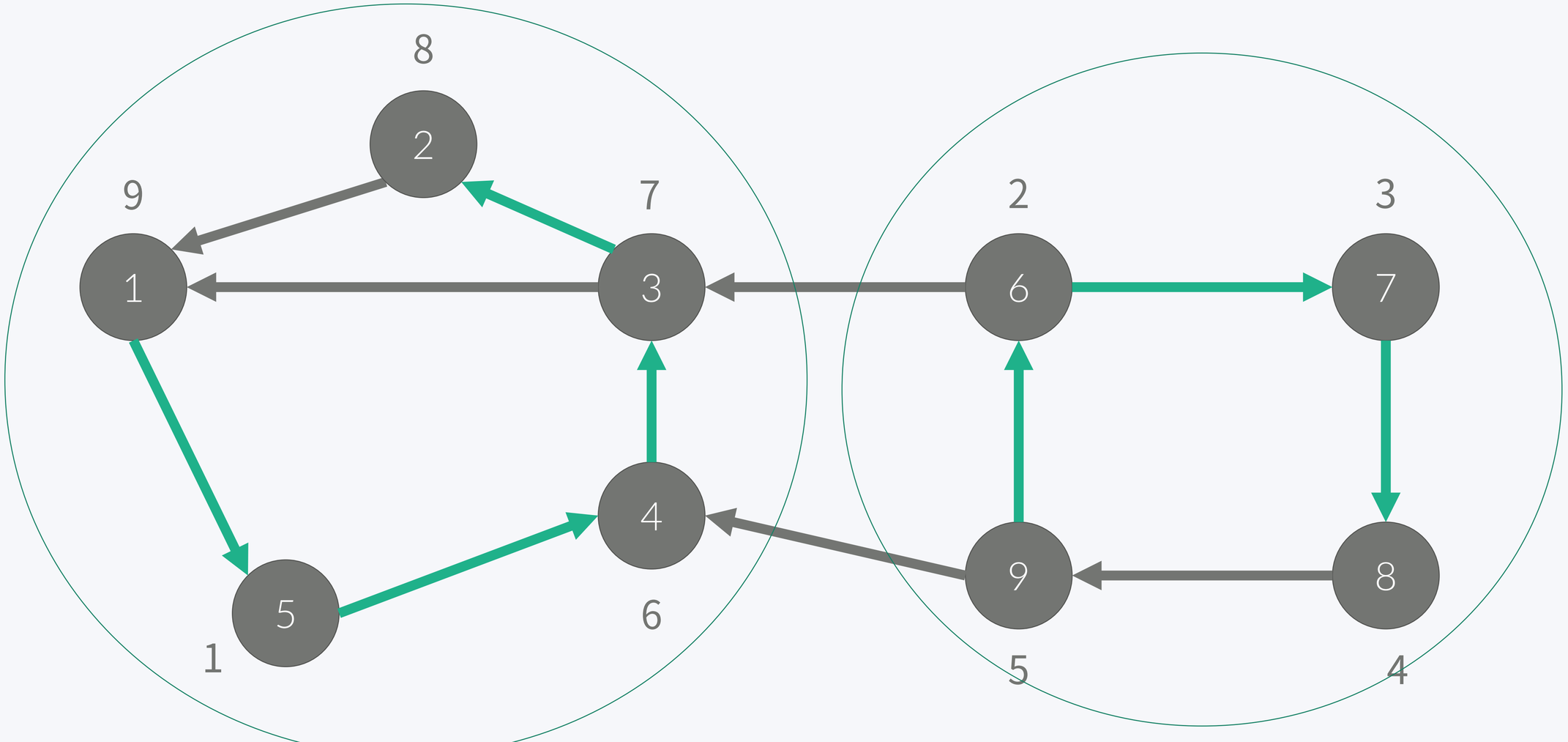
- 가장 높은 수부터 DFS



Kosaju's Algorithm

강한 연결 요소

- 가장 높은 수부터 DFS



Kosaju's Algorithm

강한 연결 요소

- C/C++: https://gist.github.com/Baekjoon/4b930e63e95b31f9af6f

DFS Tree

강한 연결 요소

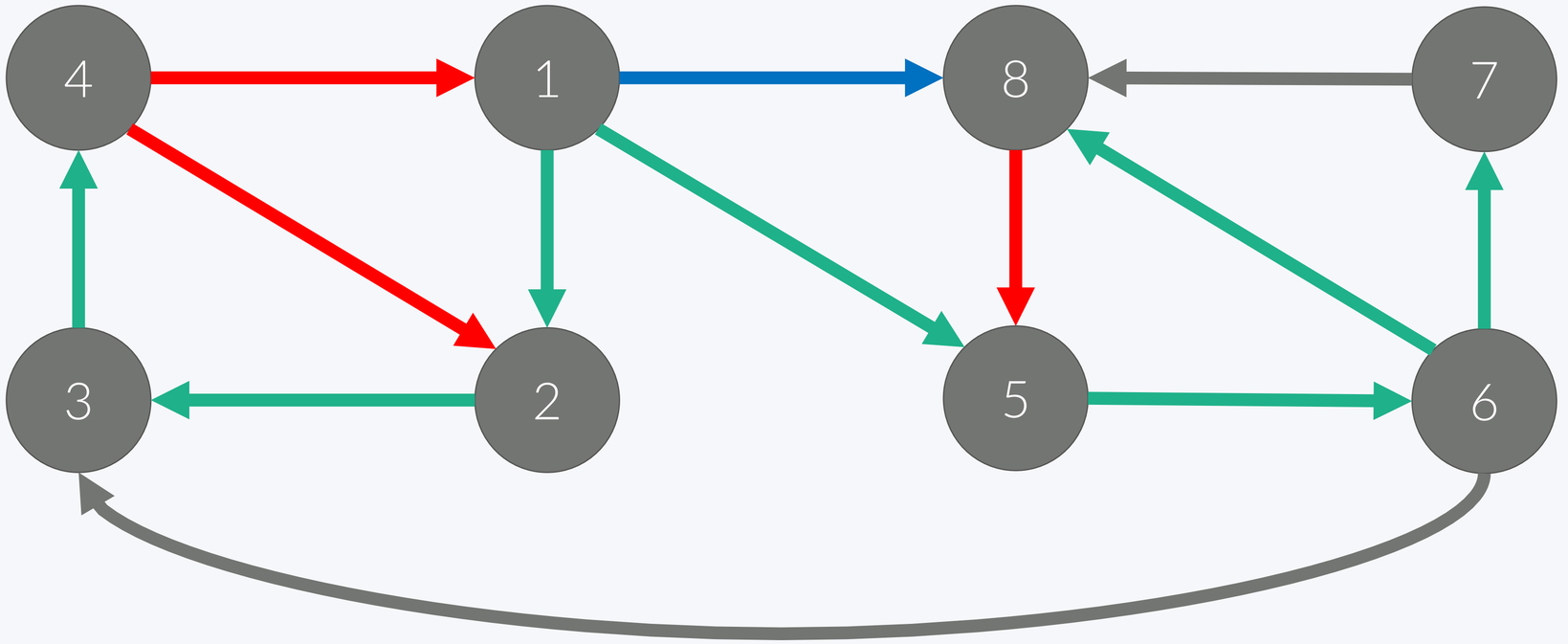
그래프 → 트리

- 그래프를 DFS로 순회했을 때, 방문한 간선은 스패닝 트리를 만든다.
- 이 트리를 DFS Tree라고 함
- DFS Tree의 Edge는 총 4가지 종류가 있음
  - Tree edge: Tree를 이루는 edge
  - Back edge: 조상으로 향하는 edge
  - Forward edge: 자식이 아닌 자손으로 향하는 edge
  - Cross edge: 그 외의 경우

DFS Tree

강한 연결 요소

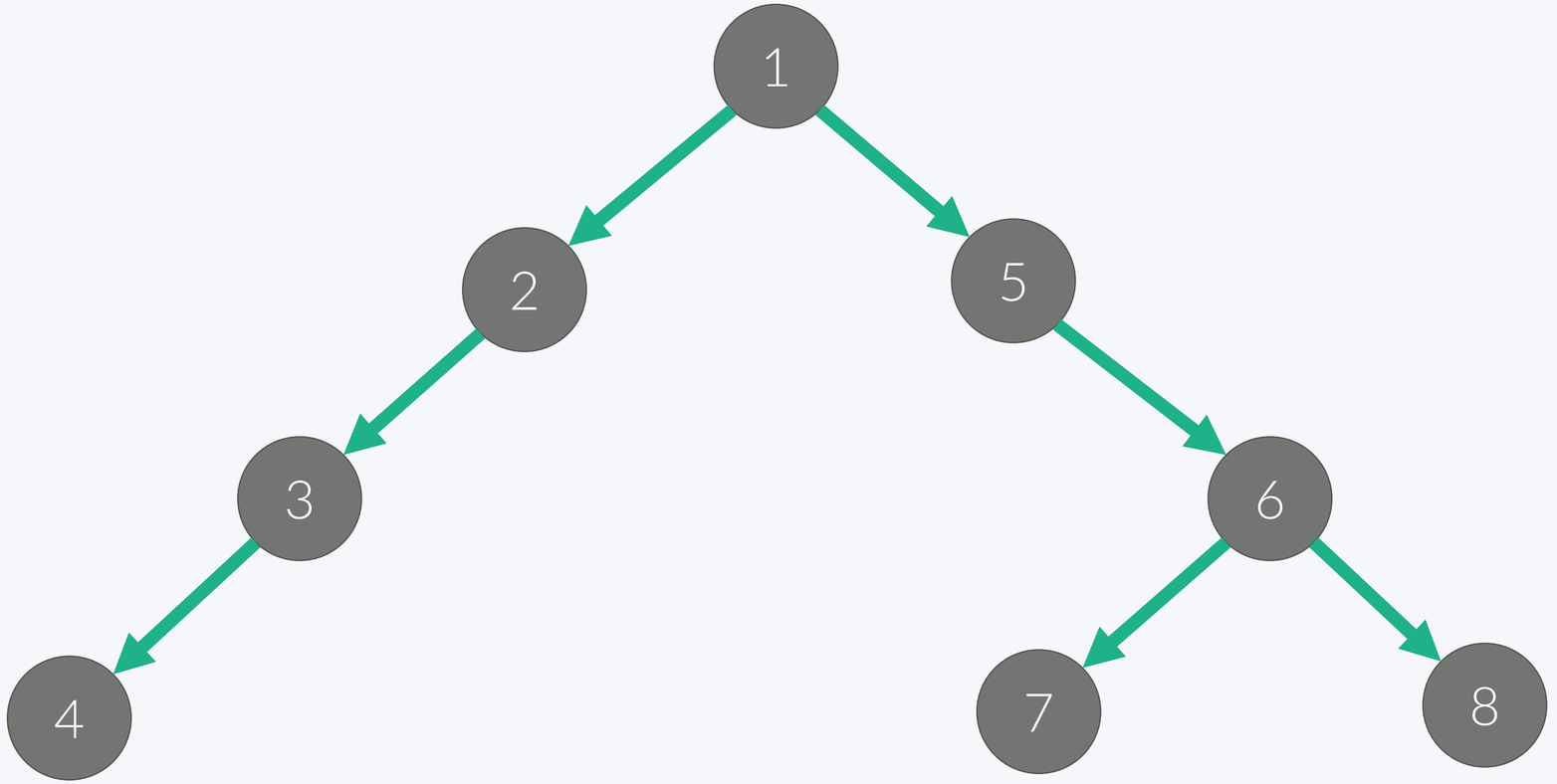
- 그래프를 DFS로 순회했을 때, 방문한 간선은 스패닝 트리를 만든다.



DFS Tree

강한 연결 요소

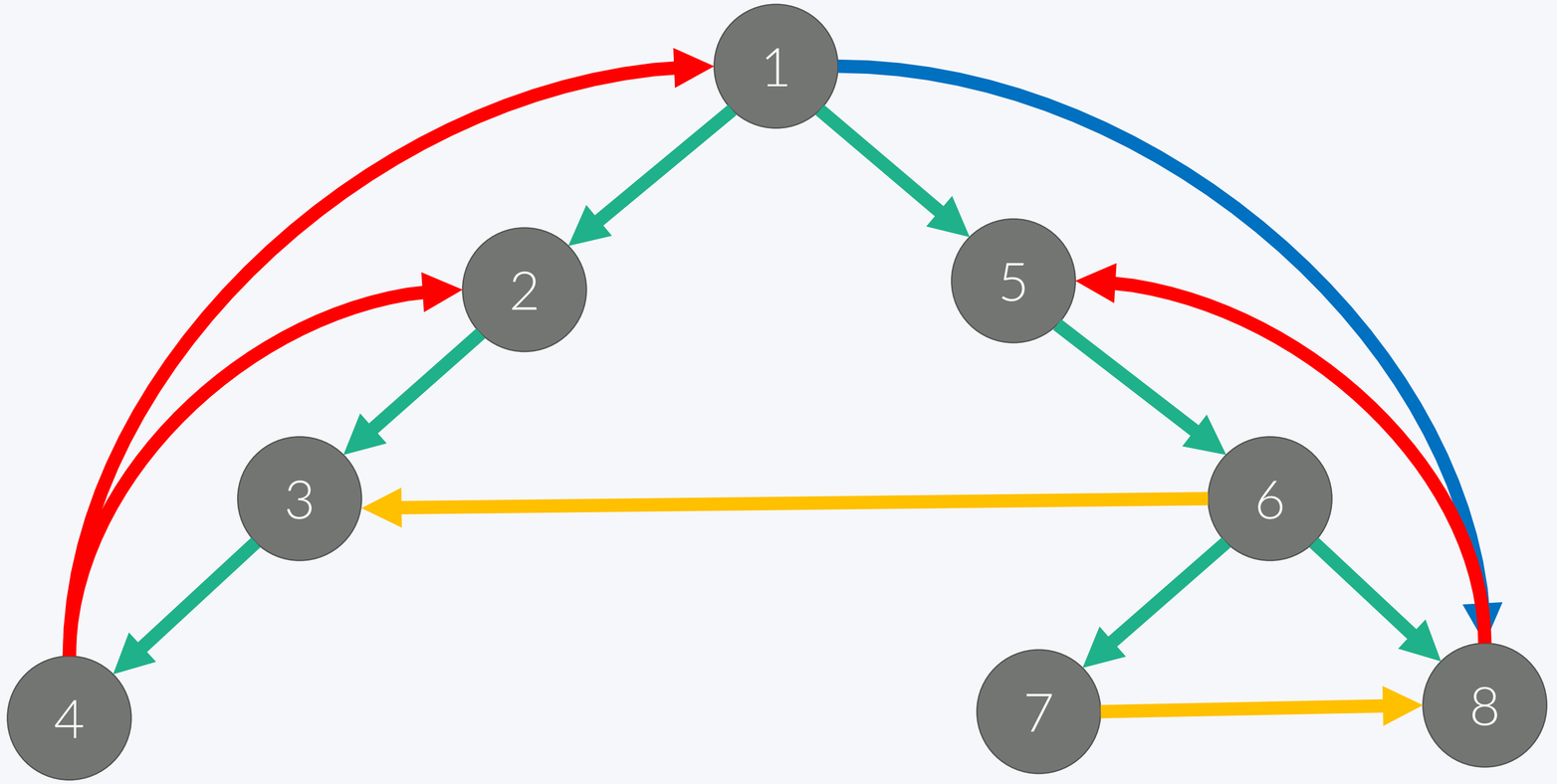
- 그래프를 DFS로 순회했을 때, 방문한 간선은 스패닝 트리를 만든다.



DFS Tree

강한 연결 요소

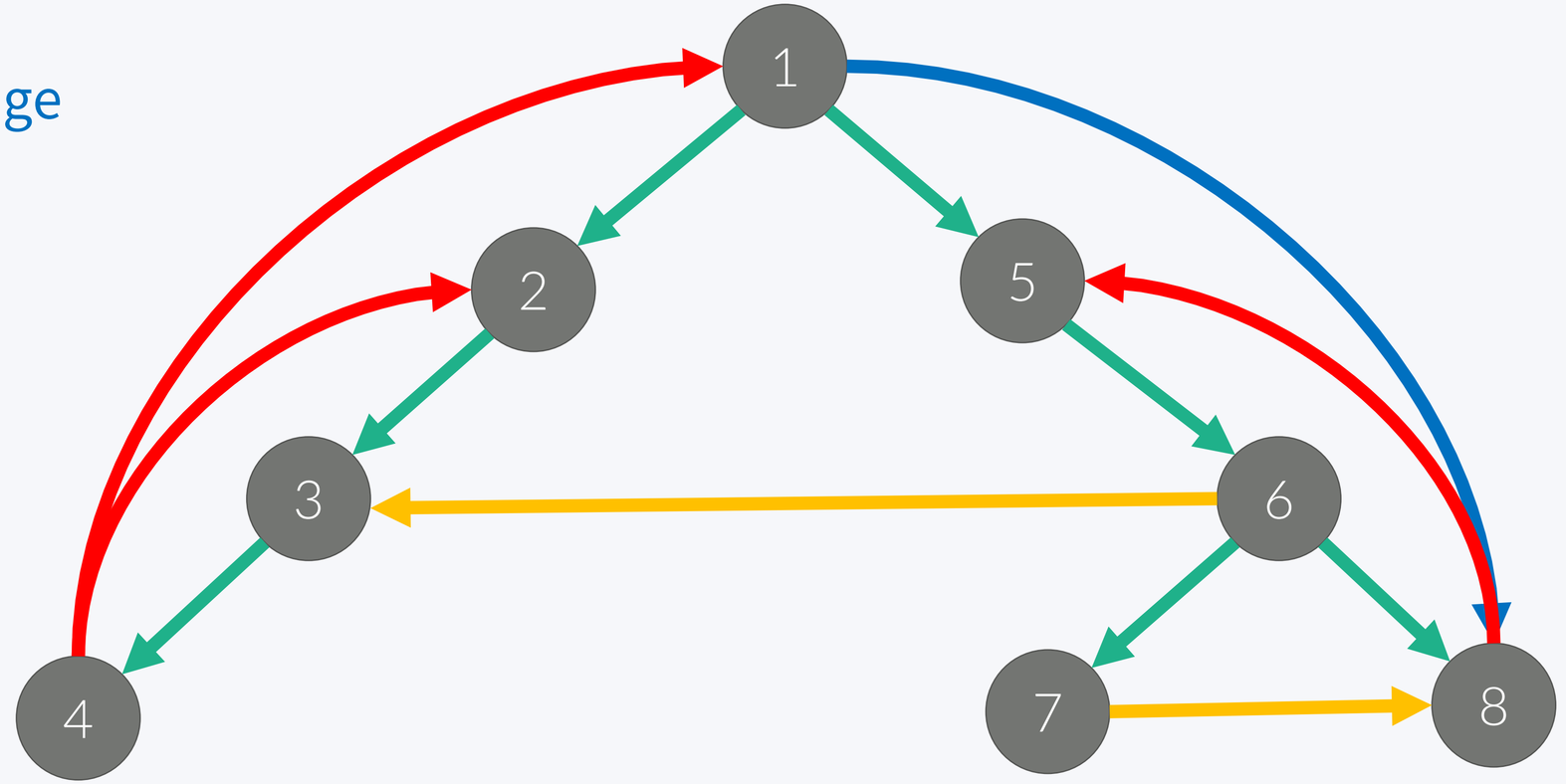
- 그래프를 DFS로 순회했을 때, 방문한 간선은 스패닝 트리를 만든다.



DFS Tree

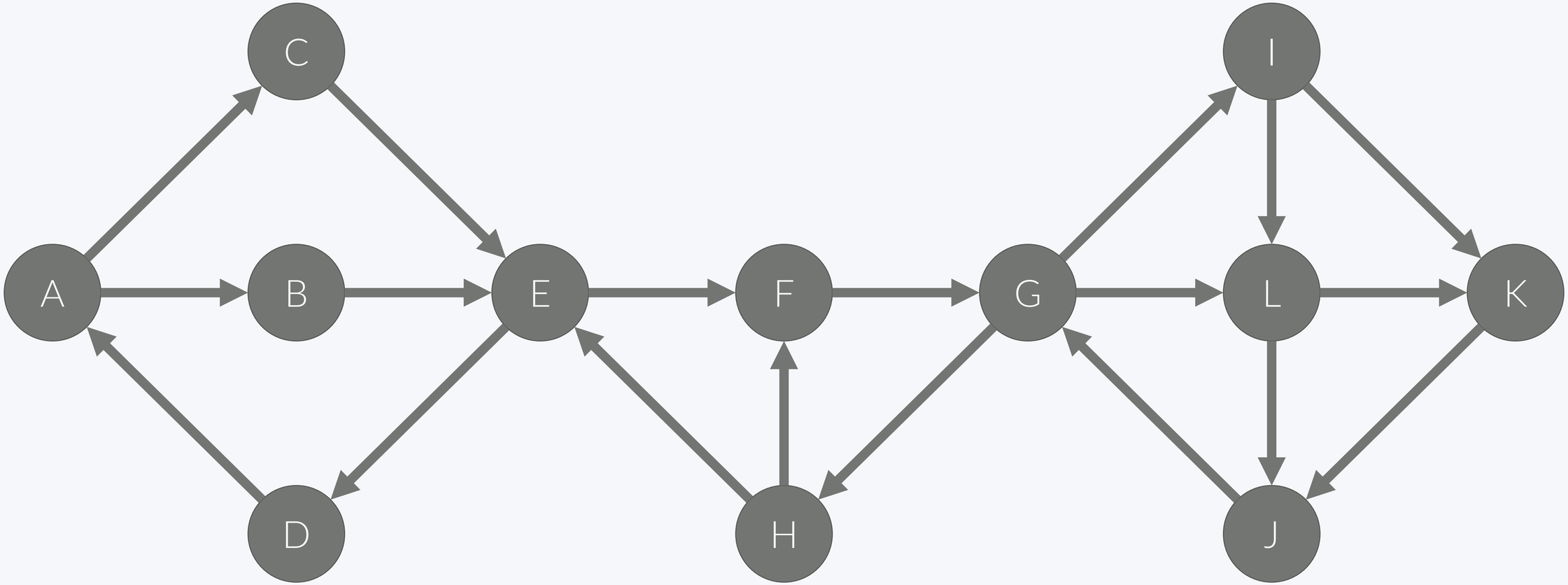
강한 연결 요소

- Tree edge
- Back edge
- Forward edge
- Cross edge



Tarjan's Algorithm

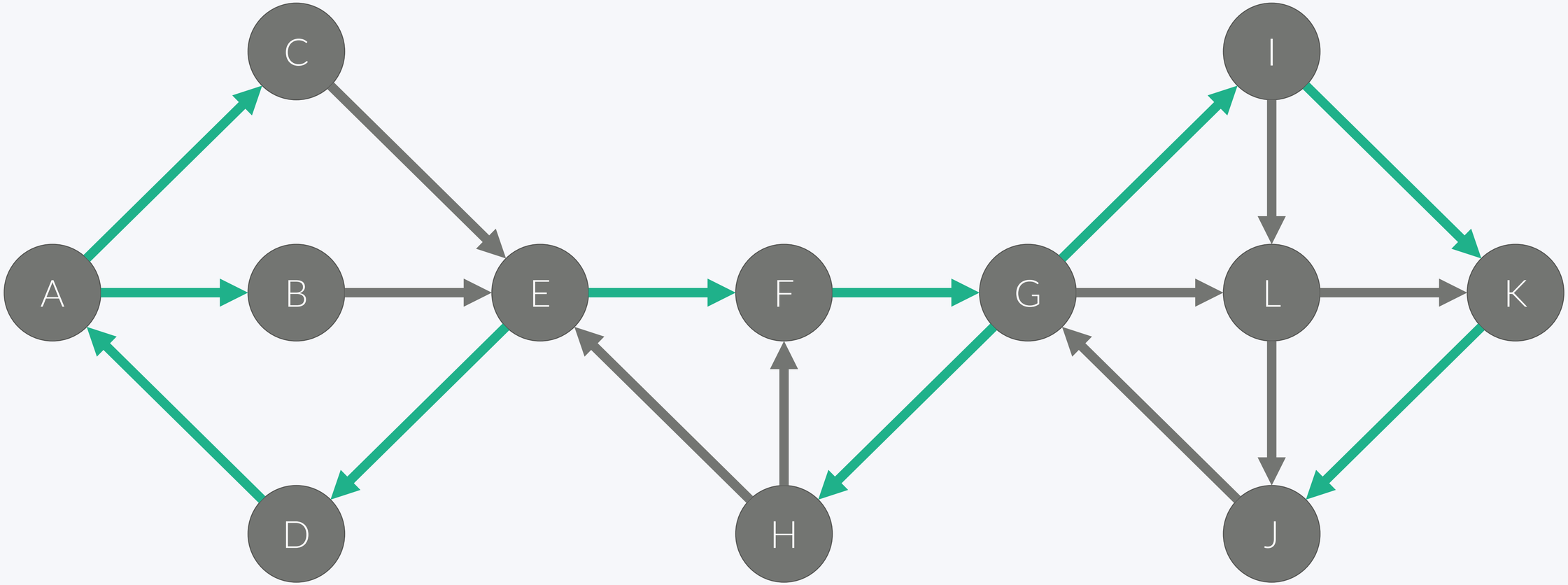
강한 연결 요소



Tarjan's Algorithm

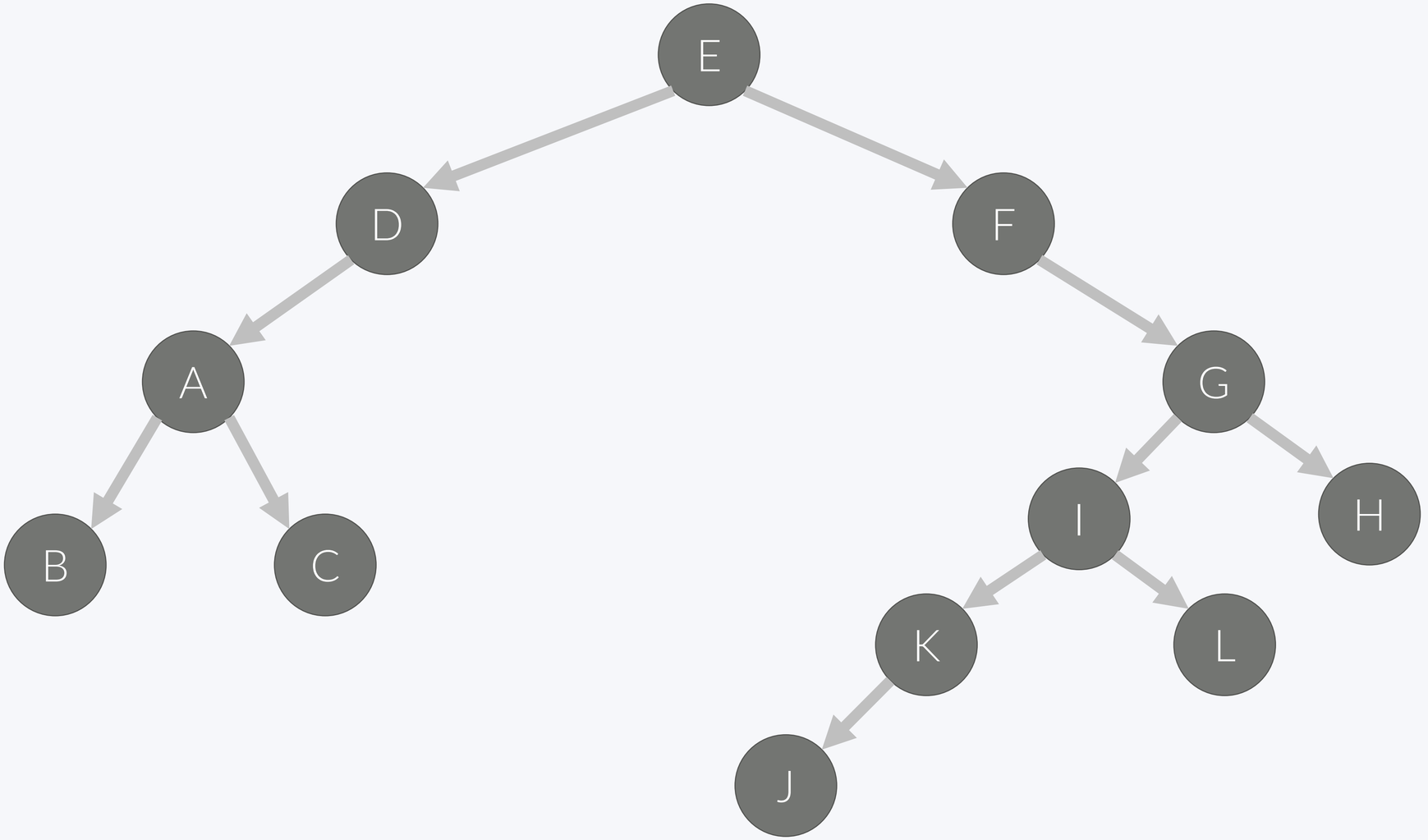
강한 연결 요소

- E에서 시작



Tarjan's Algorithm

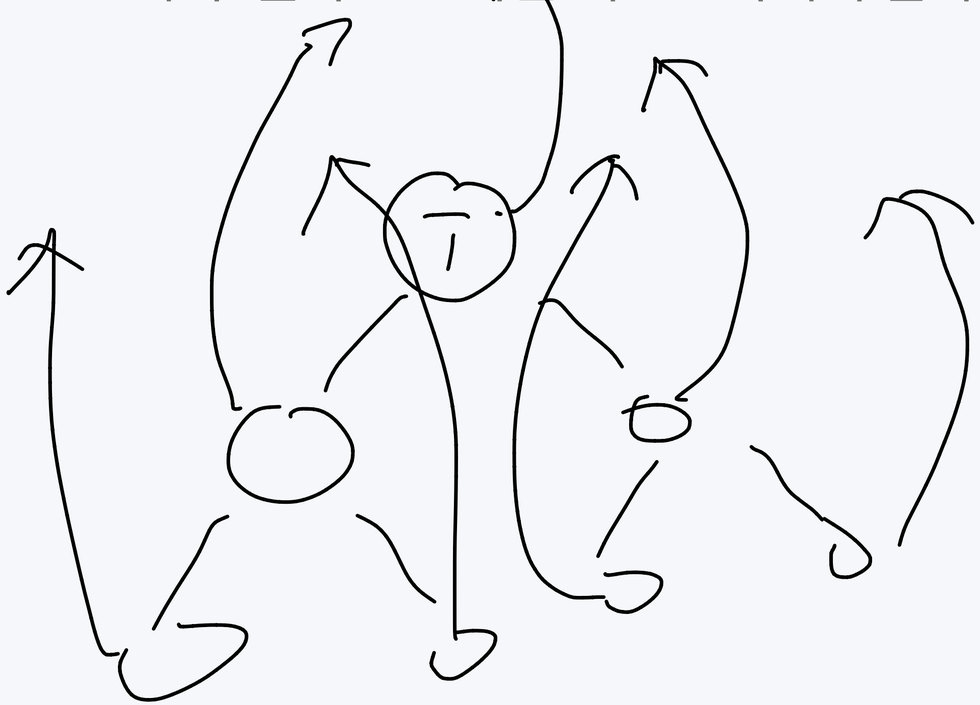
강한 연결 요소



Tarjan's Algorithm

강한 연결 요소

- num[i] = i번 정점이 dfs에서 몇 번째로 방문했는지
- low[i] = DFS Tree에서 i를 루트로하는 서브트리에서 갈 수 있는 가장 위에 있는 조상

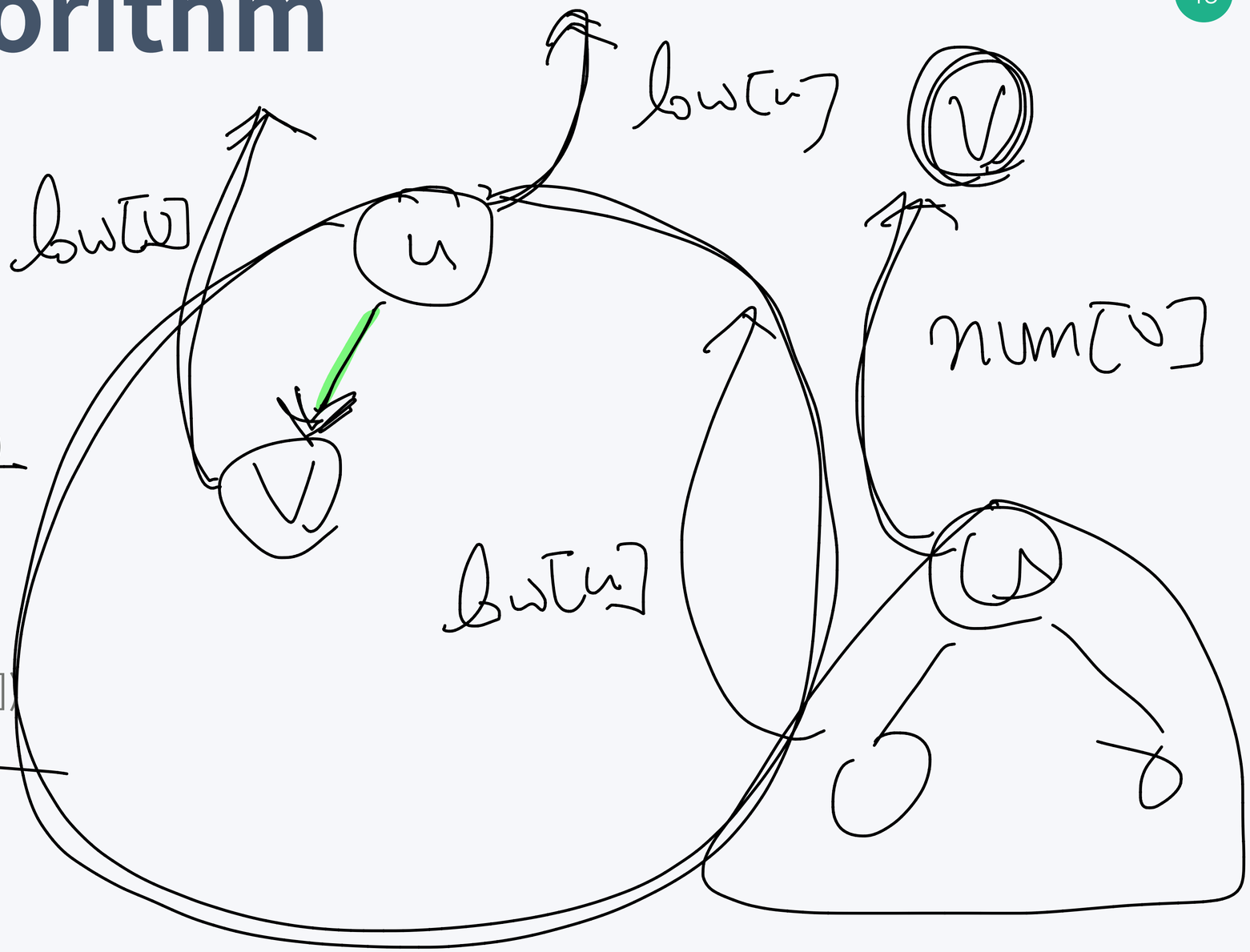


num 값이 가장 작은

Tarjan's Algorithm

강한 연결 요소

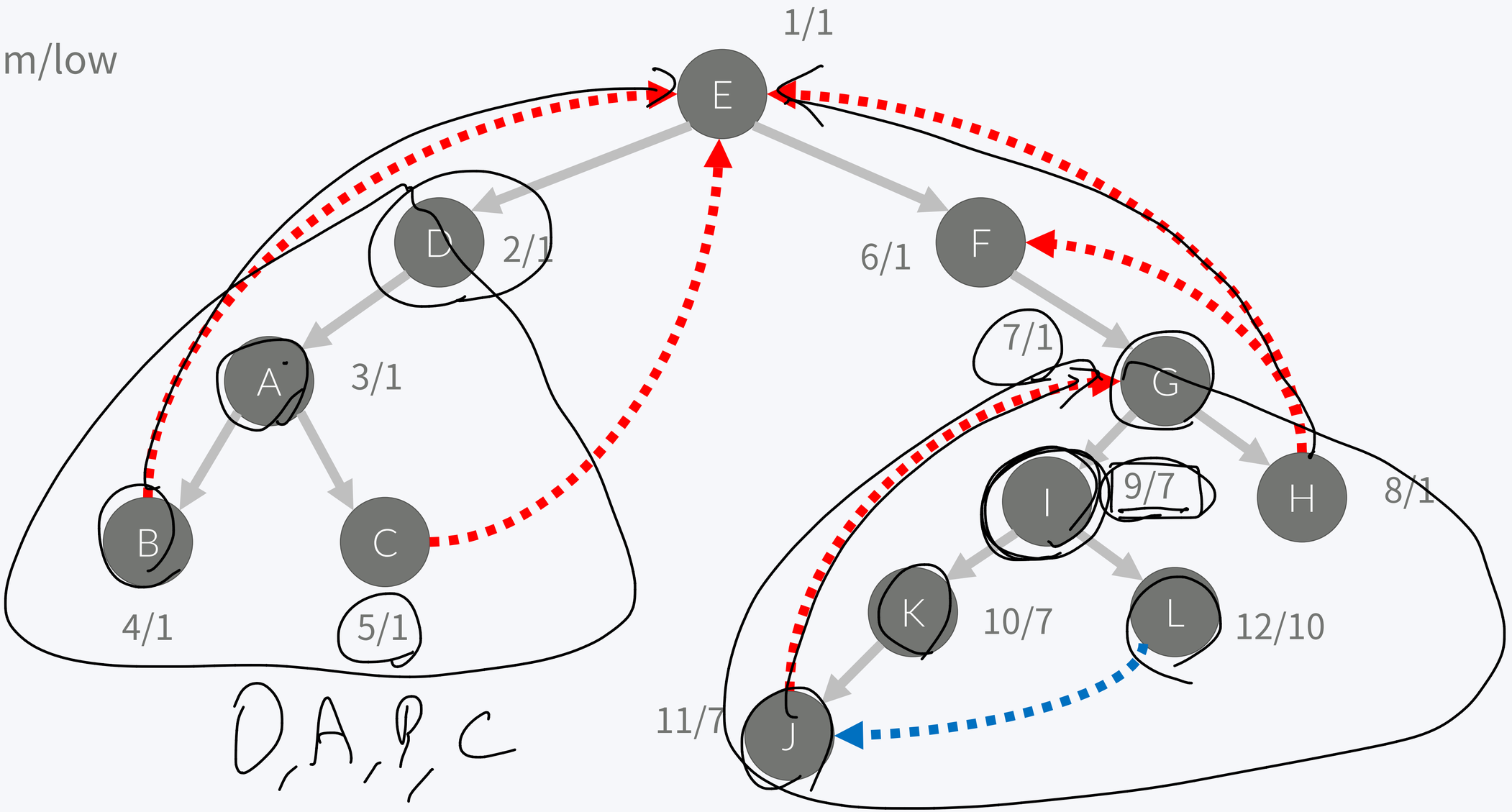
- edge (u, v)에 대해서
- Tree Edge인 경우
  - v를 아직 방문하지 않음
  - low[u] = min(low[u], low[v])
- Back Edge인 경우
  - v를 이미 방문
  - low[u] = min(low[u], num[v])



Tarjan's Algorithm

강한 연결 요소

- num/low



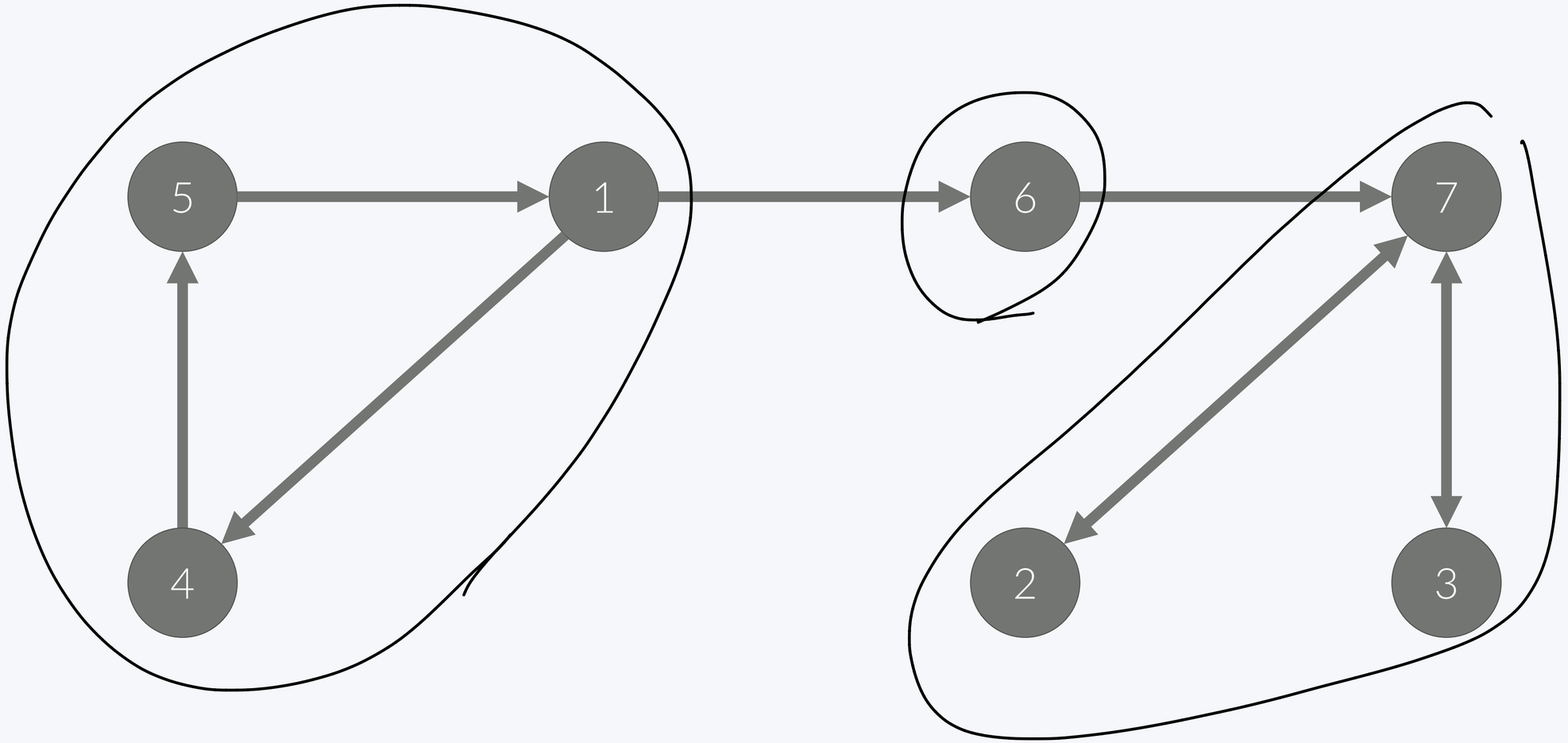
Tarjan's Algorithm

강한 연결 요소

- C/C++: https://gist.github.com/Baekjoon/df7d44b46a1360ccacab

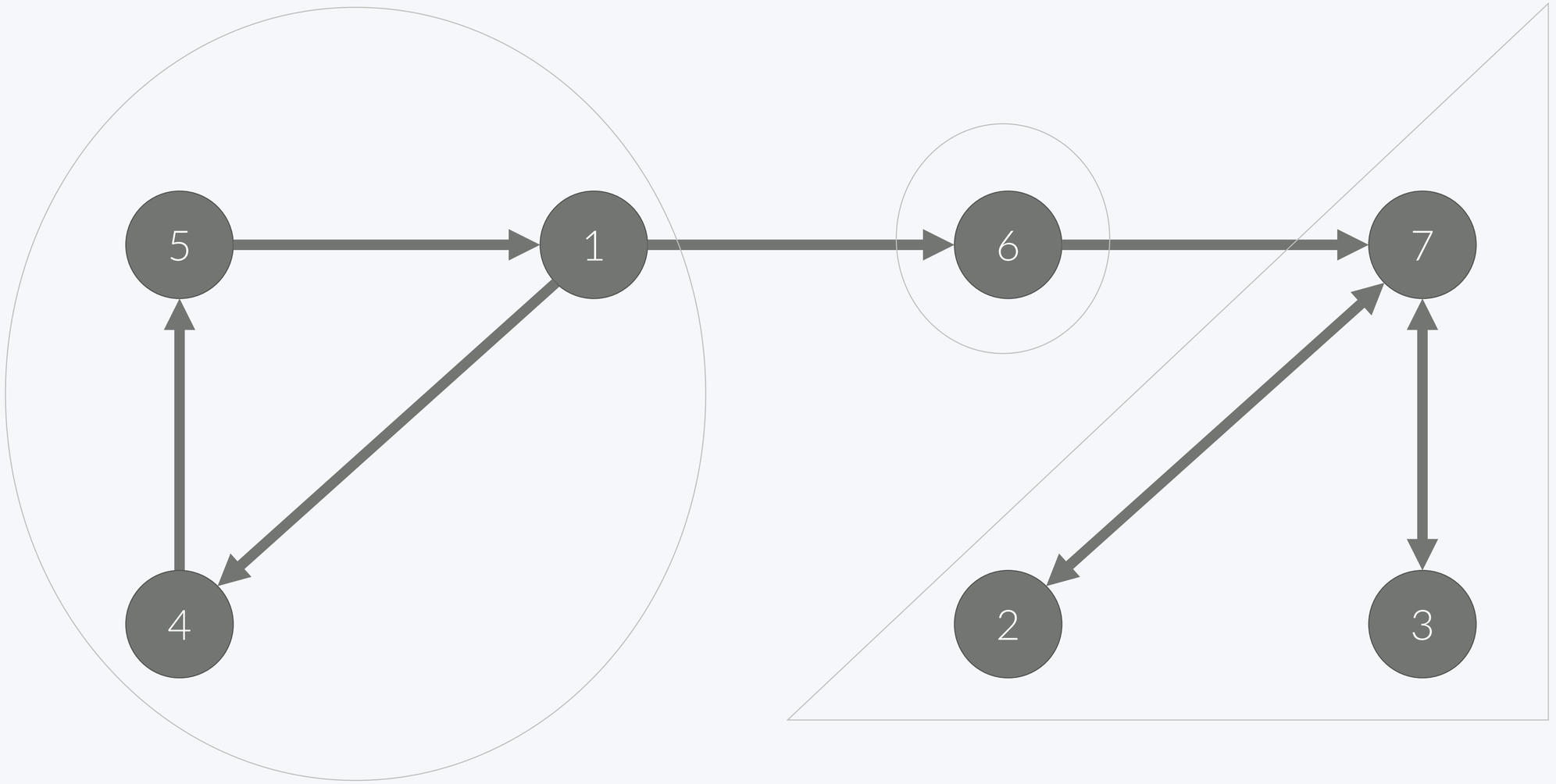
Tarjan's Algorithm

강한 연결 요소



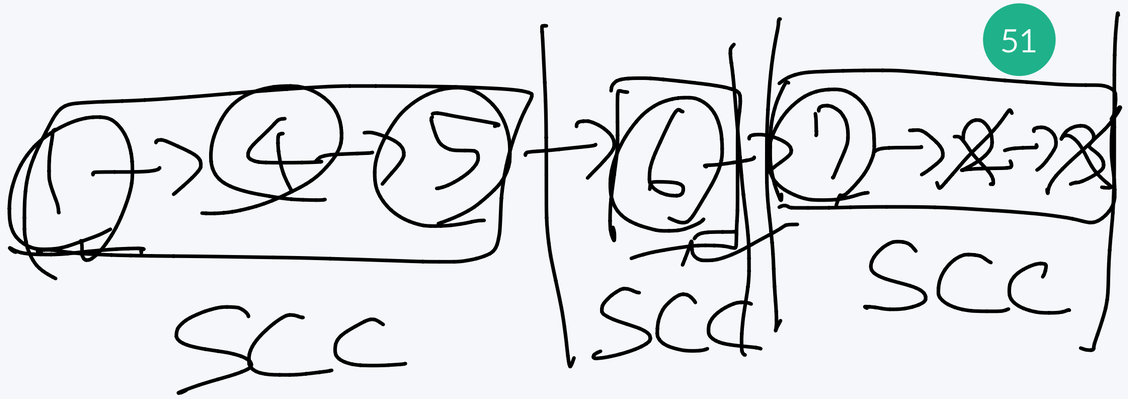
Tarjan's Algorithm

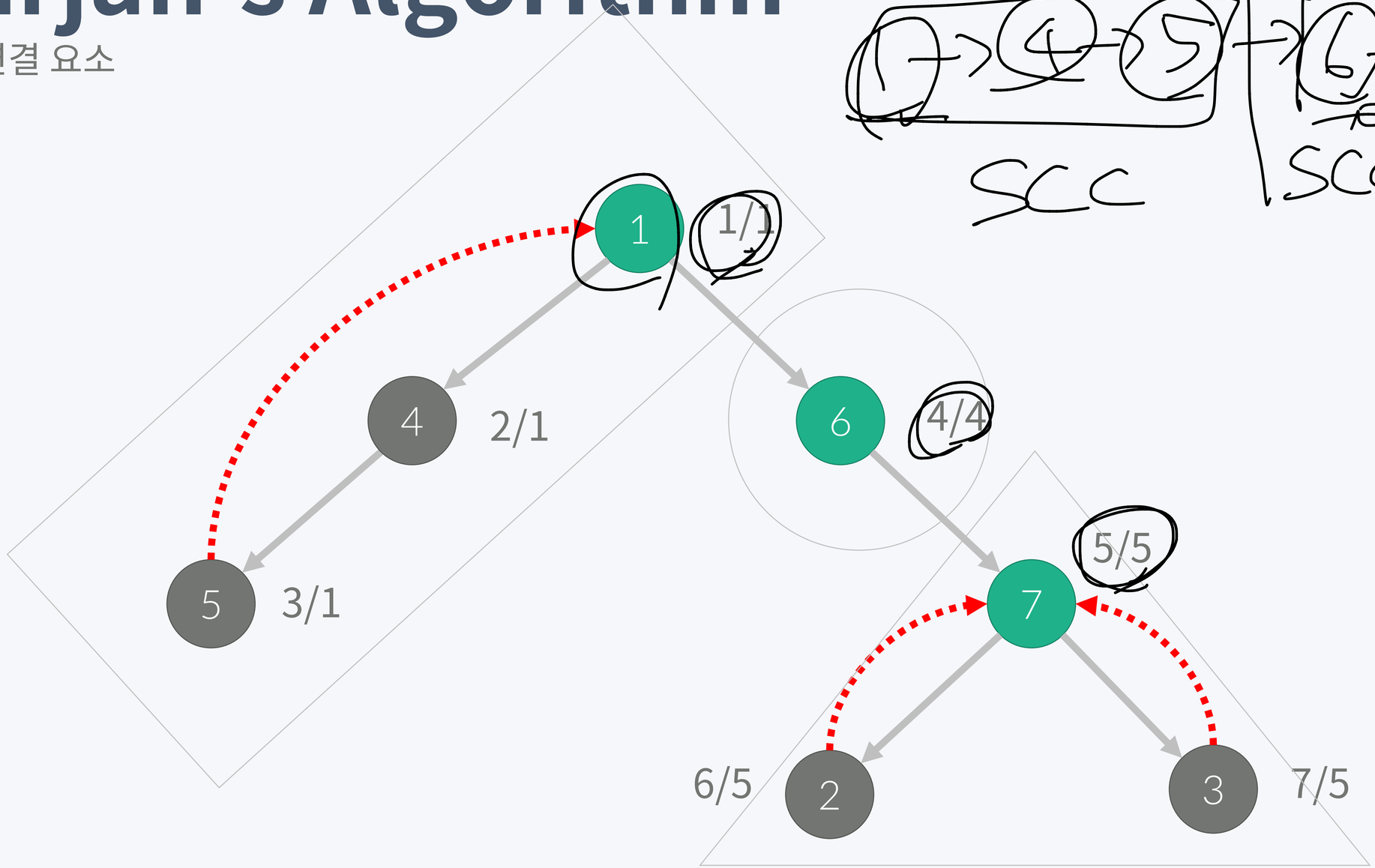
강한 연결 요소



Tarjan's Algorithm

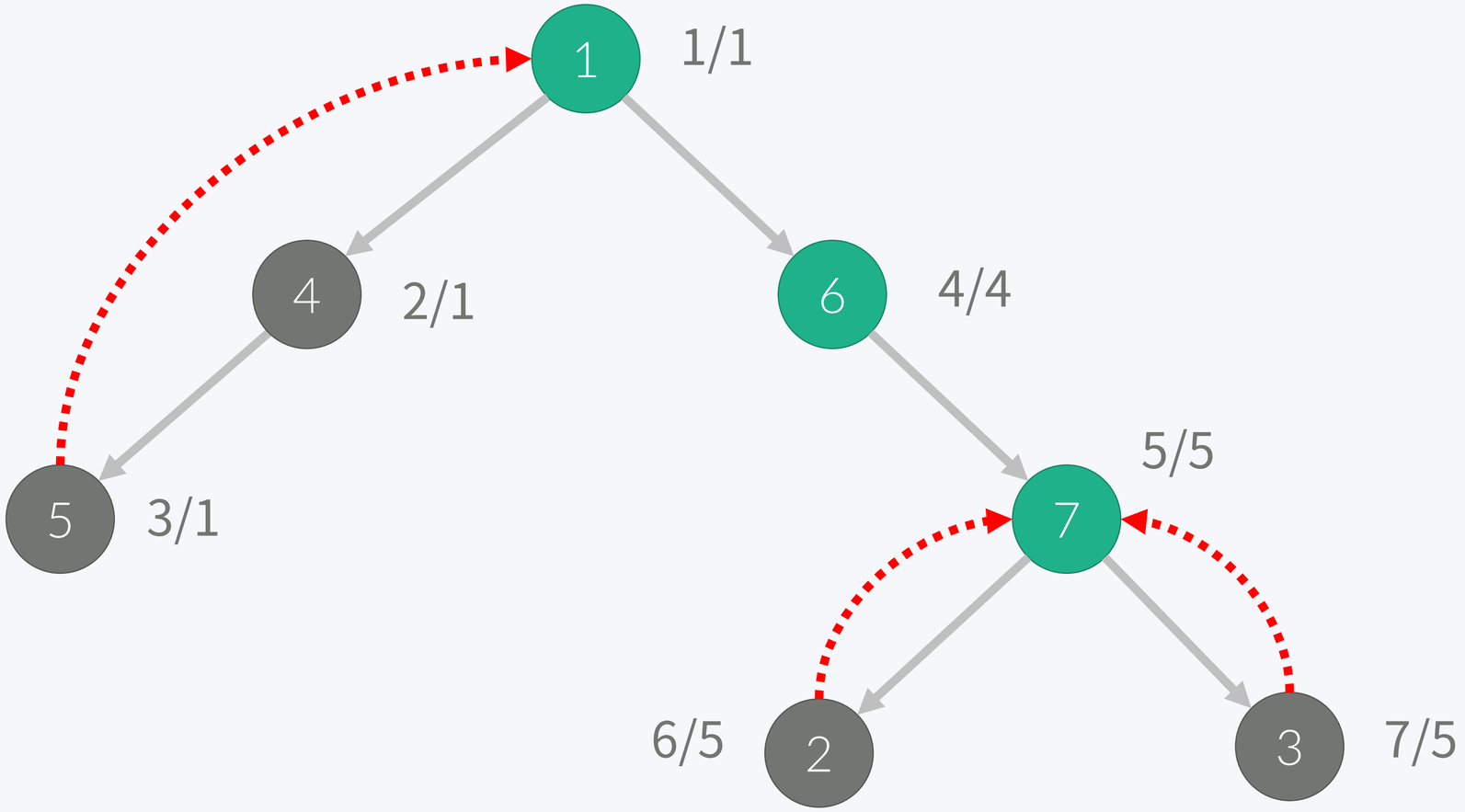
강한 연결 요소





Tarjan's Algorithm

강한 연결 요소



Tarjan's Algorithm

강한 연결 요소

- C/C++: https://gist.github.com/Baekjoon/031d3740a192ef486aba

Strongly Connected Component

https://www.acmicpc.net/problem/2150

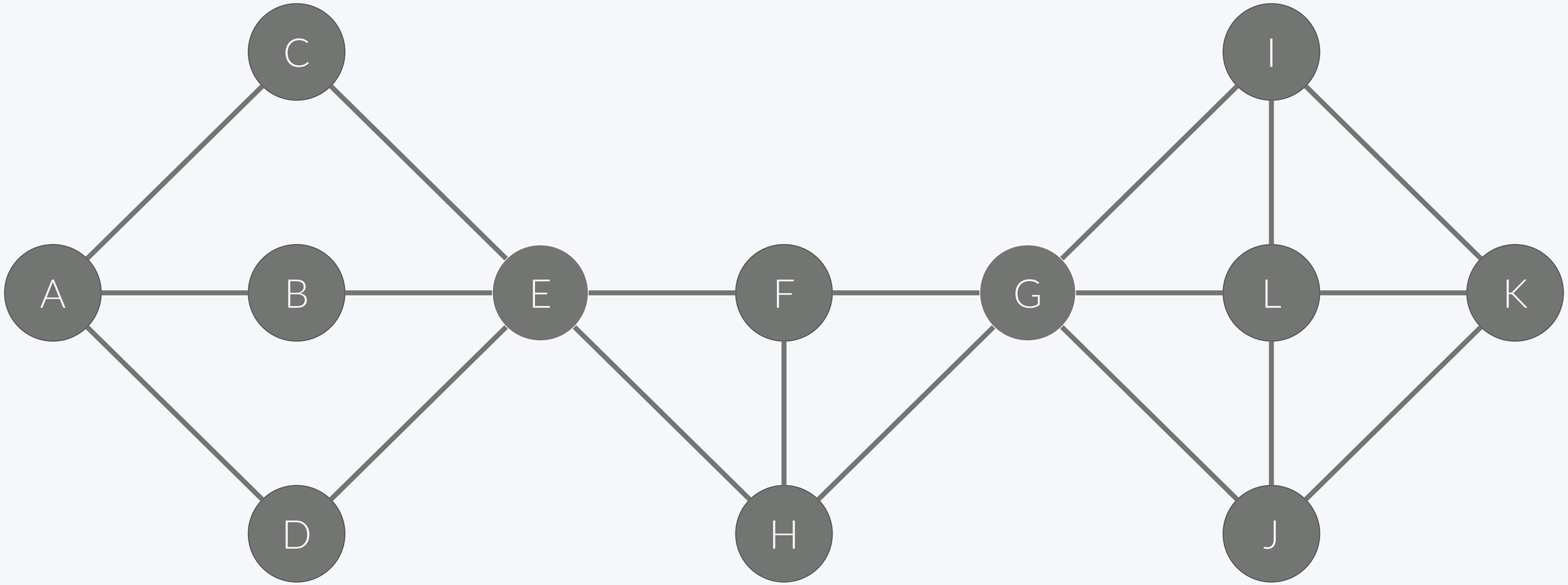
- SCC를 구하는 문제
- 소스는 앞쪽 페이지에 Tarjan, Korsaju 모두 구현되어 있다

단절점, 단절선

단절점

Articulation Point, Cut Vertex

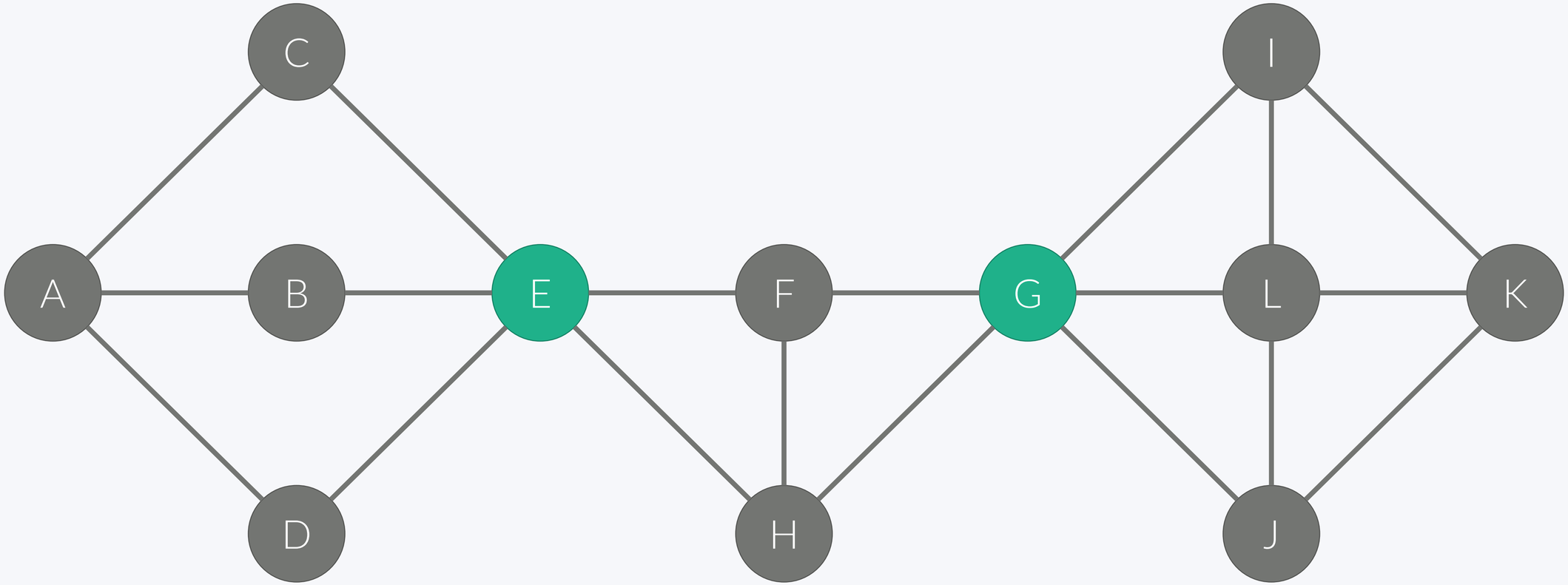
- 어떤 vertex를 그래프에서 제거했을 때, 그래프가 2개 또는 그 이상의 component로 쪼개지는 vertex



단절점

Articulation Point, Cut Vertex

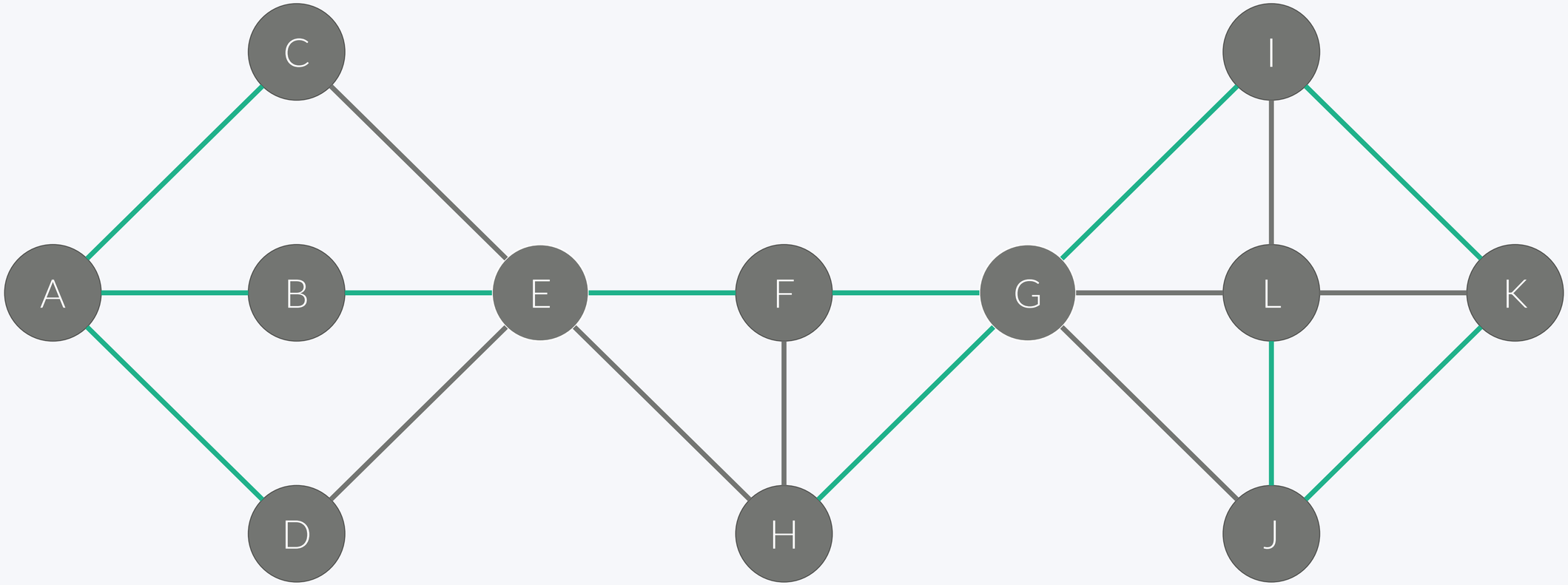
- 어떤 vertex를 그래프에서 제거했을 때, 그래프가 2개 또는 그 이상의 component로 쪼개지는 vertex



단절점

Articulation Point, Cut Vertex

- E에서 시작



단절점

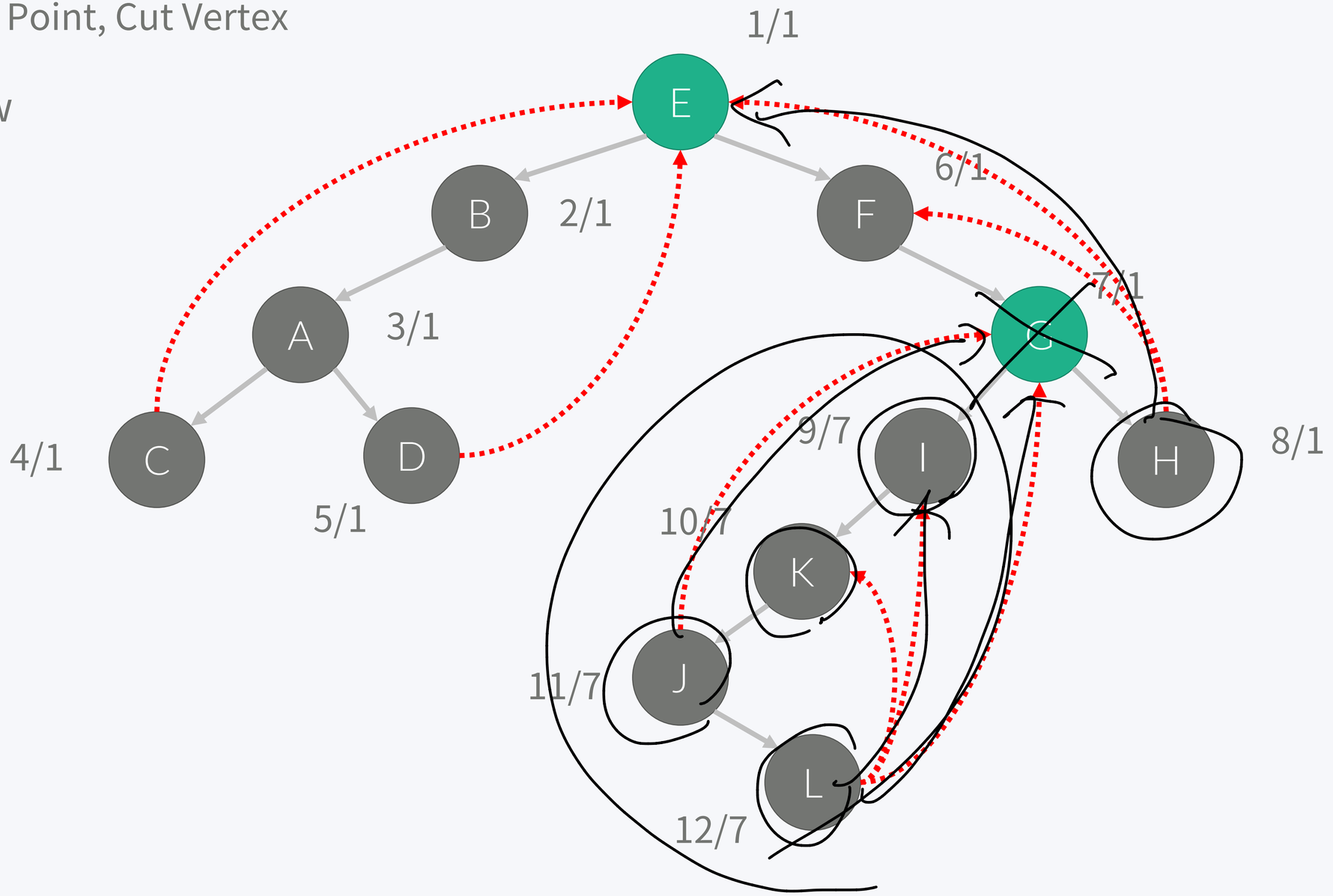
Articulation Point, Cut Vertex

- u가 루트가 아닌 경우
    - (u, v)가 Tree Edge인데
        - low[v] >= num[u] 인 경우
- u가 루트인 경우
    - children이 2개 이상

단절점

Articulation Point, Cut Vertex

- num/low



단절점

Articulation Point, Cut Vertex

- u가 루트가 아닌 경우
  - (u, v)가 Tree Edge인데
    - low[v] >= num[u] 인 경우
- u가 루트인 경우
  - children이 2개 이상

단절선

Bridge

- u가 루트가 아닌 경우
    - (u, v)가 Tree Edge인데
        - low[v] > num[u] 인 경우

단절점

https://www.acmicpc.net/problem/11266

- C/C++: https://gist.github.com/Baekjoon/bd0691b6f1f2e4d0e30d

단절선

https://www.acmicpc.net/problem/11400

- C/C++: https://gist.github.com/Baekjoon/450472009f46aab45cd0

문제 풀기

동치 증명

https://www.acmicpc.net/problem/3682

- SCC의 개수를 찾는다
- in-degree가 0이거나 out-degree가 0인 component의 개수를 각각 센다
- 최대가 답이다

- 예외: scc 개수가 1개면 답은 0

동치 증명

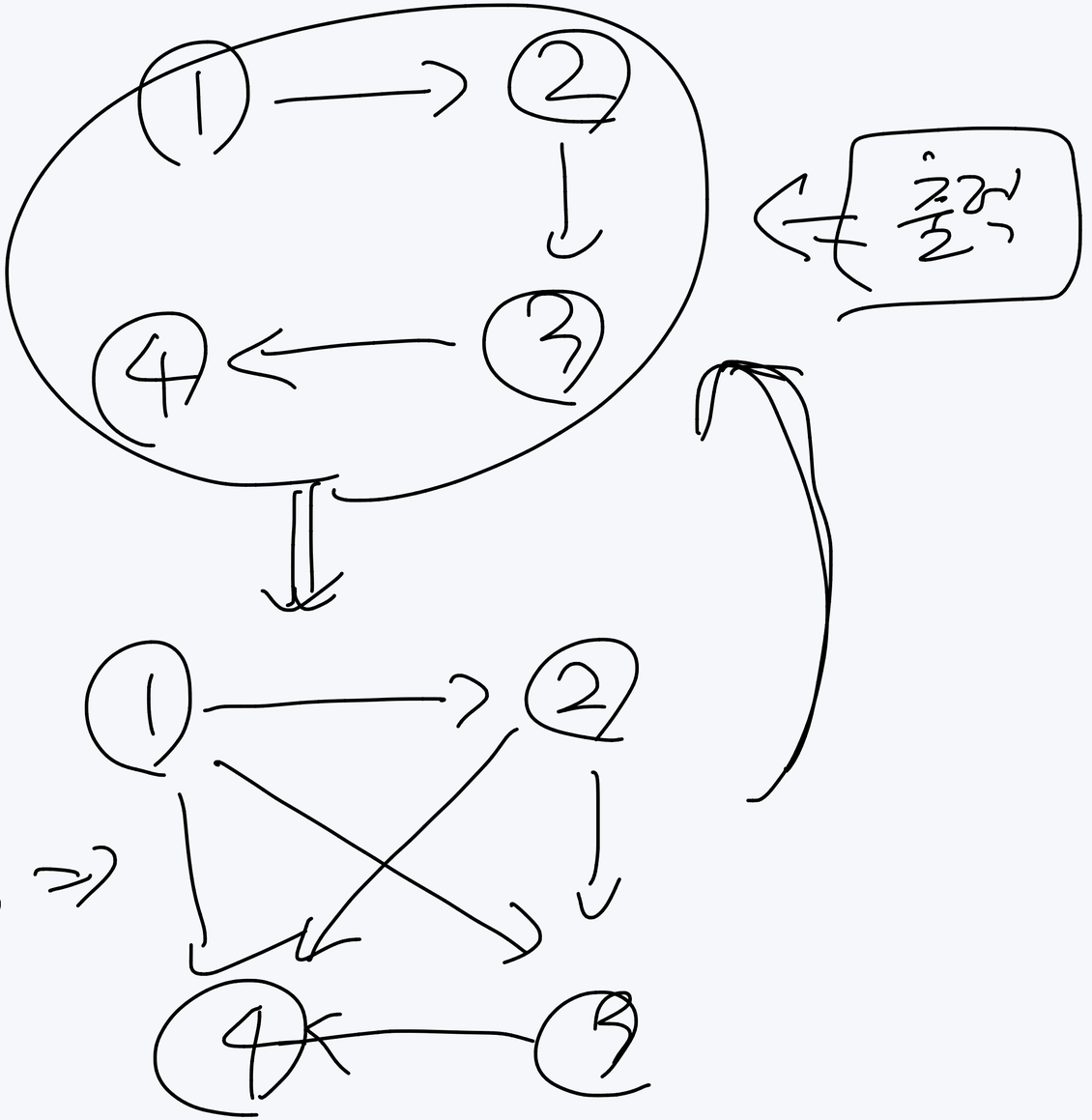
https://www.acmicpc.net/problem/3682

- C/C++: https://gist.github.com/Baekjoon/e484f0014a6879c9483b

Urban planning

https://www.acmicpc.net/problem/11097

- 단방향 도로만 존재한다
- Rechability list
- A[i][j] = i->j갈 수 있으면 1, 아니면 0



Urban planning

https://www.acmicpc.net/problem/11097

- 단방향 도로만 존재한다
- Rechability list
- A[i][j] = i->j갈 수 있으면 1, 아니면 0
- Rechability list를 이용해서 도로 계획을 구하는 문제
- 도로의 개수는 최소로 해야 한다

Urban planning

https://www.acmicpc.net/problem/11097

- Junction: Vertex
- one-way road: Edge

Urban planning

https://www.acmicpc.net/problem/11097

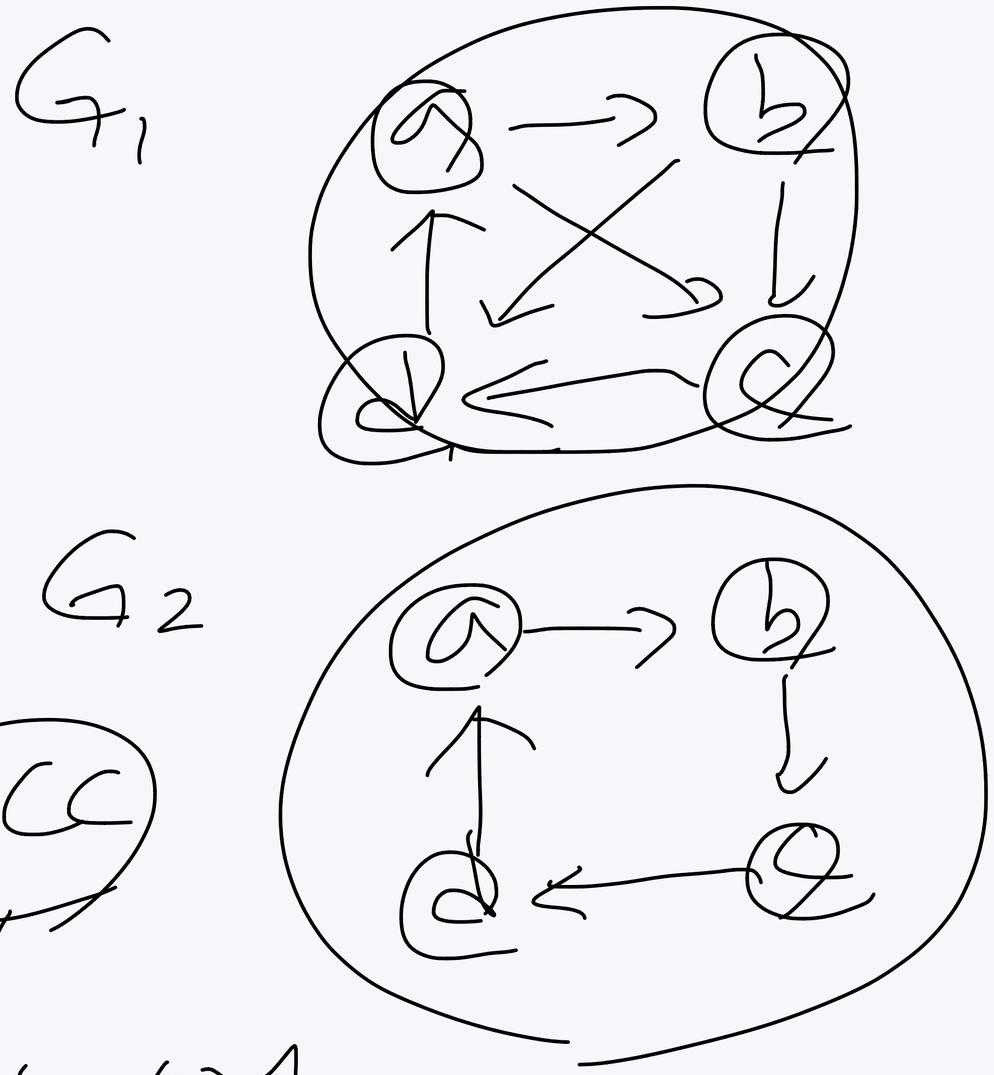
- 그래프 G0이 있을 때
- G0의 Transitive closure가 G1 (문제의 입력)
- G1의 Transitive reduction가 G2 (문제의 출력)

Urban planning

https://www.acmicpc.net/problem/11097

- G1에서 SCC는 G2에서도 SCC가 되어야 한다
- G1에서 {a, b, c, d}가 SCC였으면
- G2에서는 a->b, b->c, c->d, d->a

- SCC가 1개인 경우에는 저게 답



Urban planning

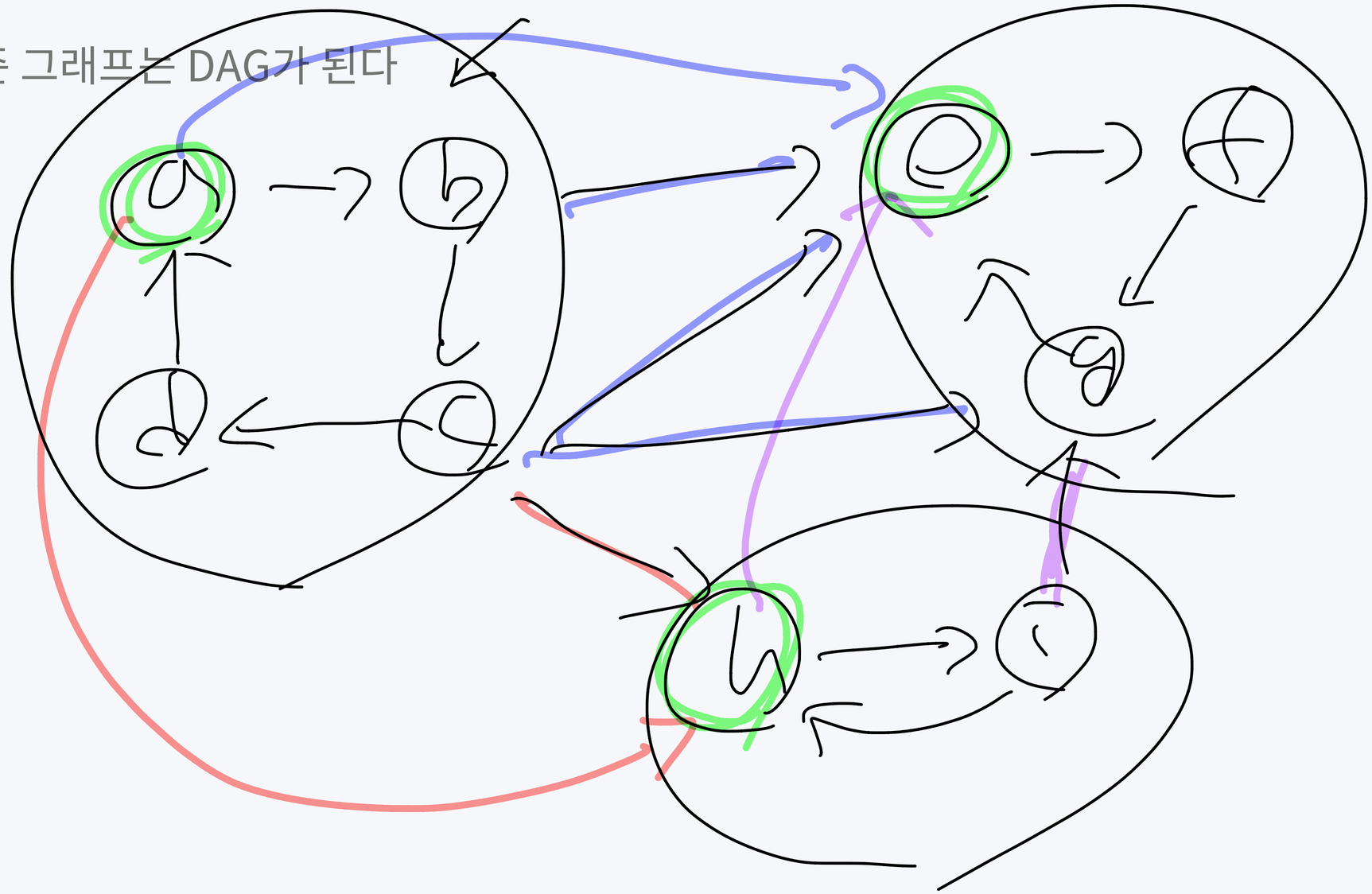
https://www.acmicpc.net/problem/11097

- SCC가 여러 개인 경우
- G0에서 x -> y가 있고
- x와 y가 G1의 SCC에서 각각 다른 component에 포함된 경우
- x는 X에, y는 Y에 포함되어 있는 경우

Urban planning

https://www.acmicpc.net/problem/11097

- SCC로 묶어준 그래프는 DAG가 된다



Urban planning

https://www.acmicpc.net/problem/11097

- a->b, b->c, a->c edge가 있을 때, a->c는 없어도 문제를 풀 수 있다

- x->y를 저장해야 하는 경우는
- x->z, z->y를 만족하는 z가 없는 경우

Urban planning

https://www.acmicpc.net/problem/11097

1. u->v, v->u가 있으면 u와 v는 SCC에서 같은 component에 속함
2. 구현을 편하게 하기 위해서 각 component에서 번호가 가장 작은 vertex를 대표 vertex로 저장
3. 그 다음, component 사이에 edge를 이어준다

Urban planning

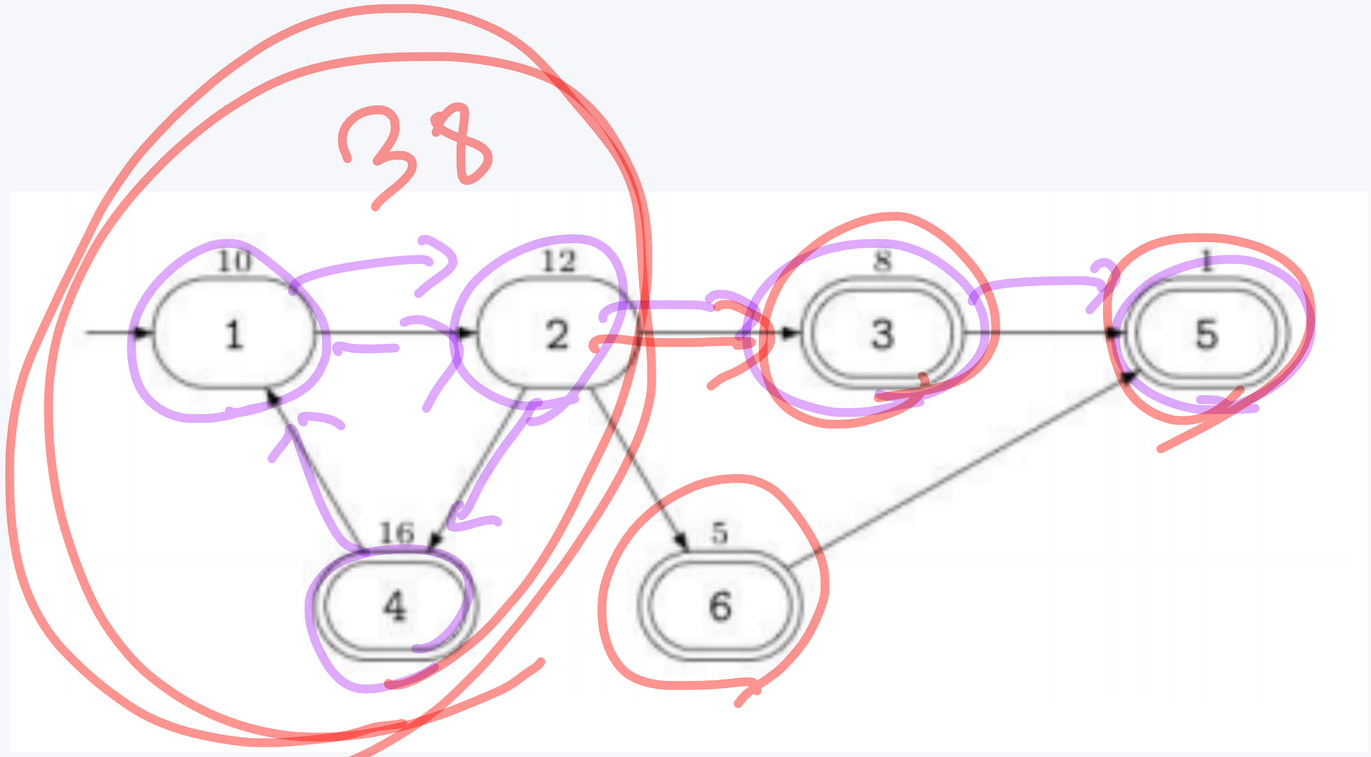
https://www.acmicpc.net/problem/11097

- C/C++: https://gist.github.com/Baekjoon/fb4ddab7392a5d2d67d1

ATM

https://www.acmicpc.net/problem/4013

- 출발 장소에서 어떤 레스토랑까지 이동하면서 인출할 수 있는 현금의 최대 액수



ATM

https://www.acmicpc.net/problem/4013

- 그래프를 SCC로 만들면 DP문제가 된다

- D[i] = A[i] + Max(D[j]) (i번 component에서 j번 component로 갈 수 있어야 함)

ATM

https://www.acmicpc.net/problem/4013

- C/C++: https://gist.github.com/Baekjoon/090e83dbe1ef91ad057c

교통 체계

https://www.acmicpc.net/problem/1734

- N개 도시, E개 도로
- 도시: 1~N
- 도로: 양방향
- $2 \leq N \leq 100,000$, $1 \leq E \leq 500,000$

- 쿼리
- 두 개의 도시 A, B가 있고, 도시 G1과 도시 G2를 잇는 도로가 있다. 도시 G1과 도시 G2 사이의 도로를 없앤 후에도 도시 A에서 도시 B로 이동할 수 있는가?
- 세 개의 도시 A, B, C가 있다. 도시 C로 들어오거나 나가는 모든 도로를 없앤 후에도 도시 A에서 도시 B로 이동할 수 있는가?

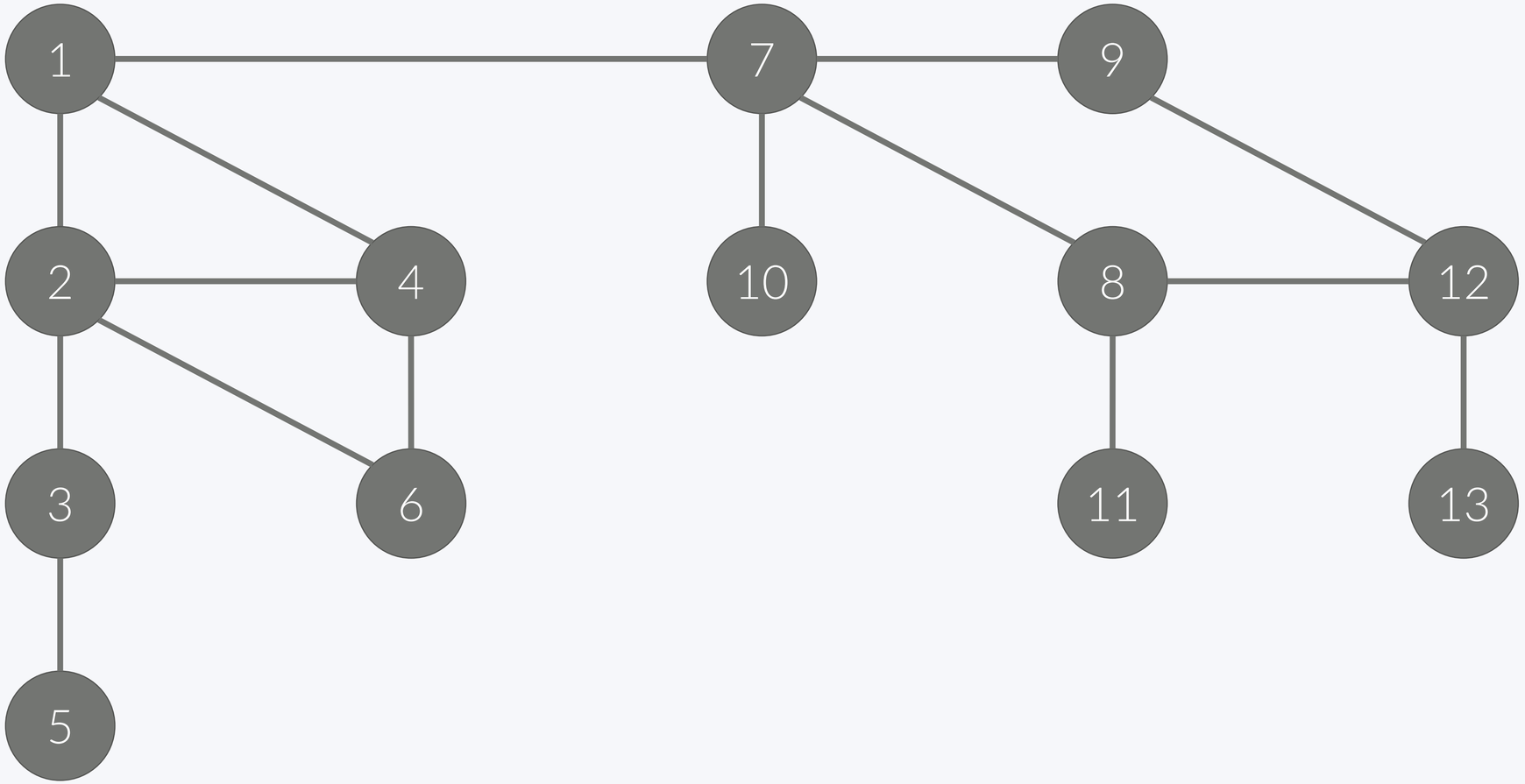
교통 체계

https://www.acmicpc.net/problem/1734

- discovery = 그 노드를 방문한 시간
- finishing = 그 노드를 빠져나간 시간
- depth = 그 노드의 깊이
- low = DFS 트리에서 그 low
  - 현재 노드 또는 그 노드를 루트로 하는 서브 트리에서 back edge 하나로 방문할 수 있는 가장 작은 discovery

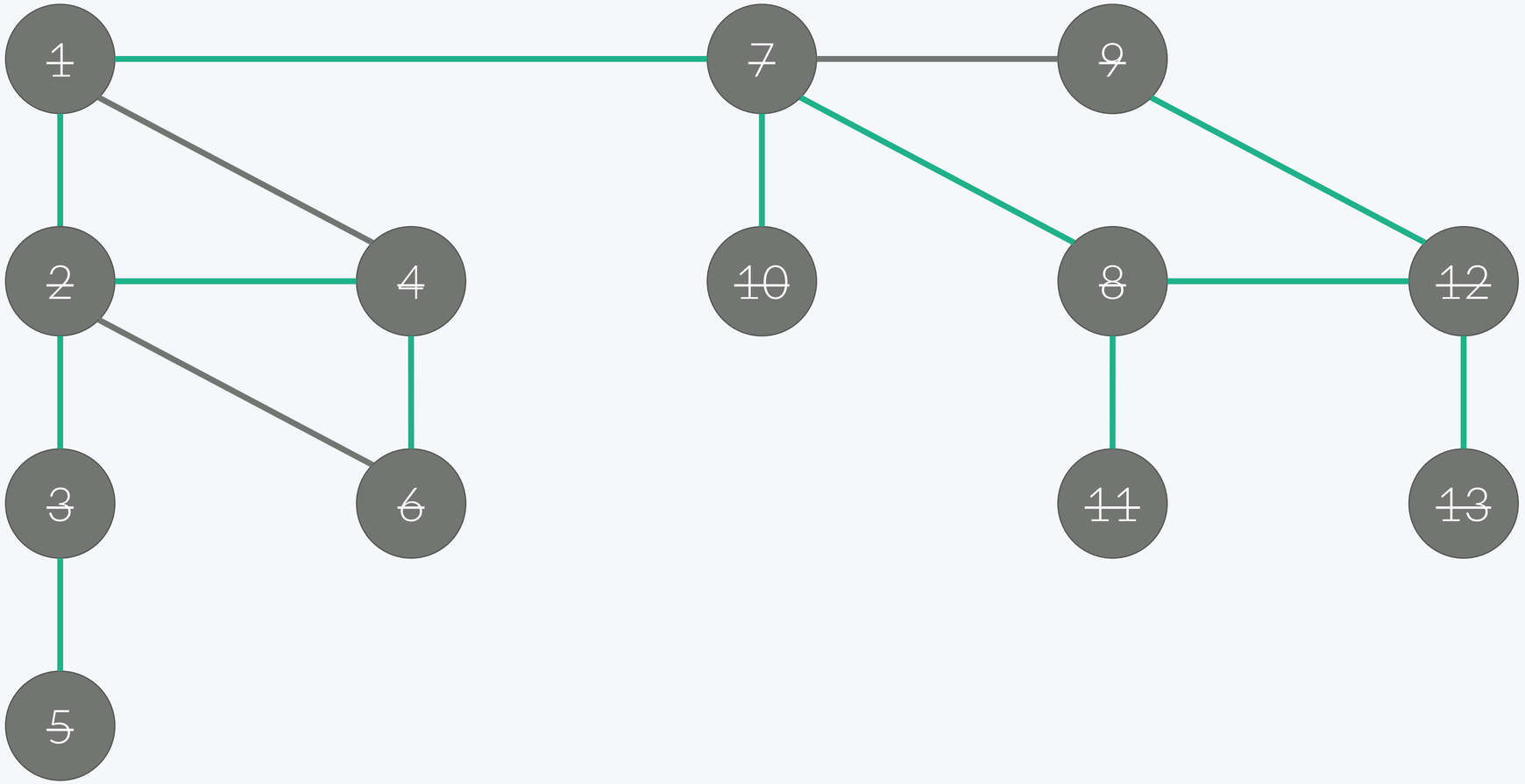
교통 체계

https://www.acmicpc.net/problem/1734



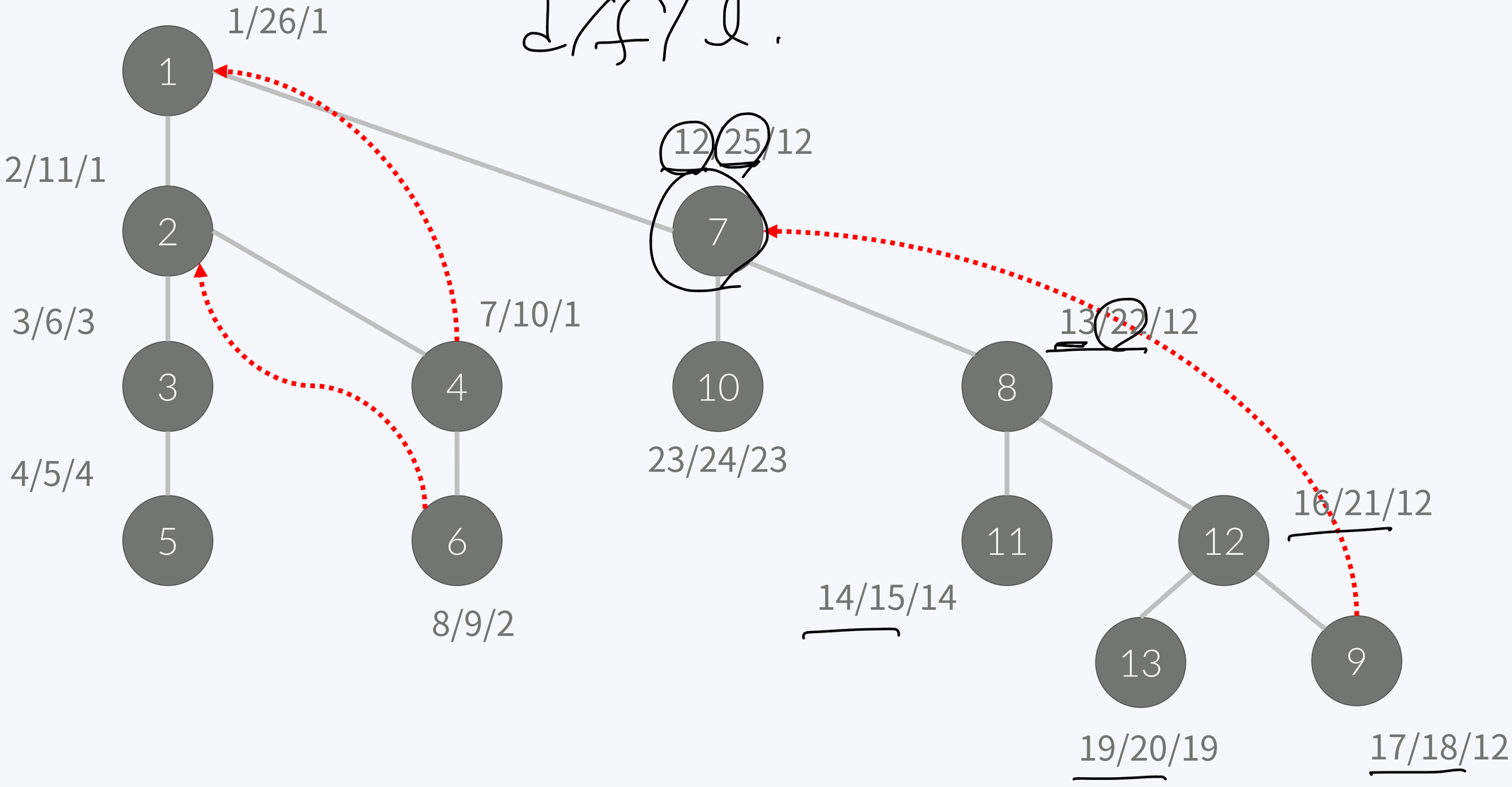
교통 체계

https://www.acmicpc.net/problem/1734



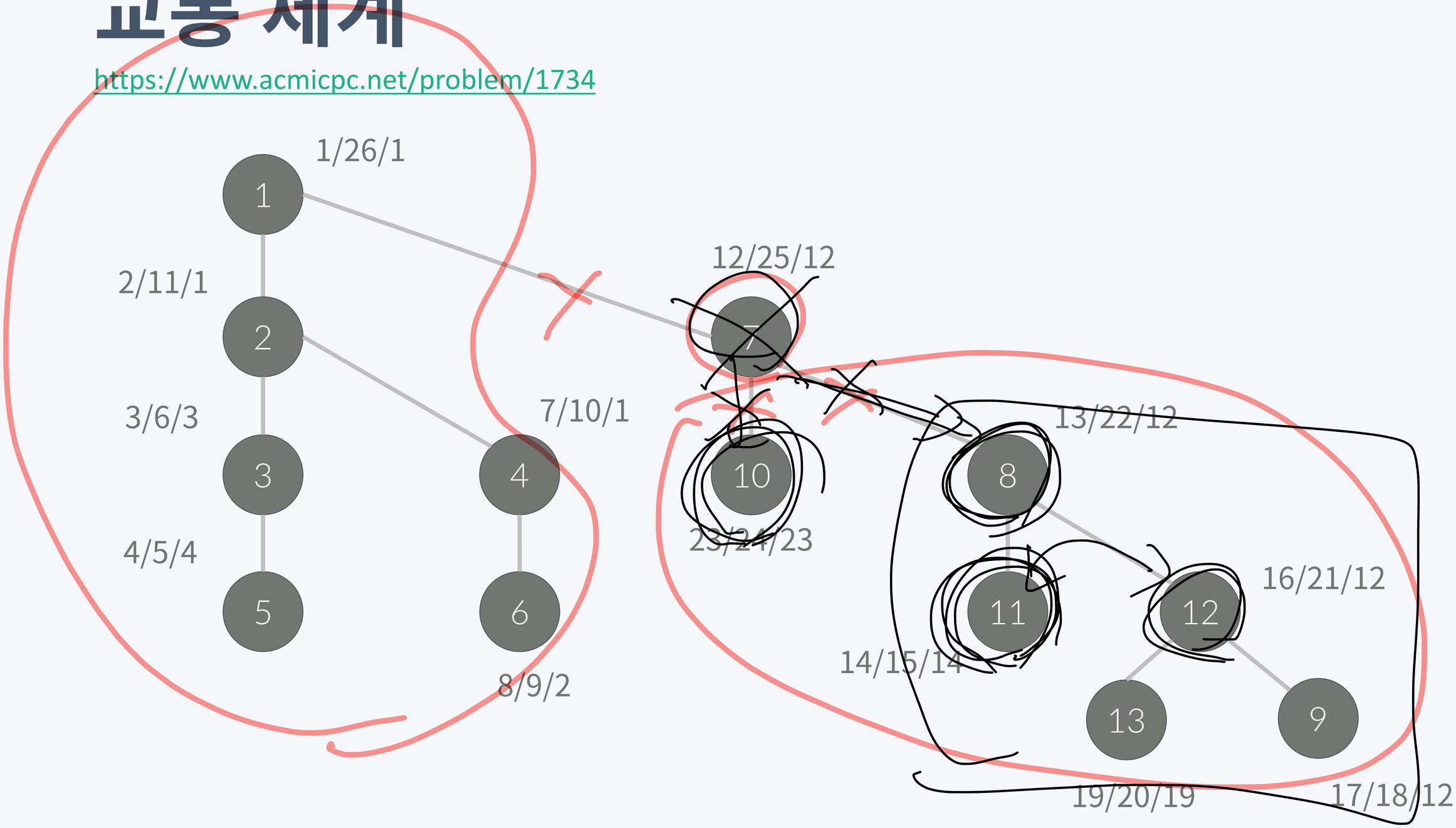
교통 체계

https://www.acmicpc.net/problem/1734



교통 체계

https://www.acmicpc.net/problem/1734



교통 체계

https://www.acmicpc.net/problem/1734

- 두 함수를 이용해서 문제를 풀 수 있다.
- is_descendant(A, B)
  - B를 루트로 하는 서브 트리 안에 A가 있나?
- find_related_child(A, B)
  - B가 A를 루트로 하는 서브 트리 안에 있을 때, 어떤 children이 B와 연결되어 있나?

교통 체계

https://www.acmicpc.net/problem/1734

- is_descendant(A, B)
  - B를 루트로 하는 서브 트리 안에 A가 있나?
  - discover[B] <= discover[A] && finish[A] <= finish[B]

교통 체계

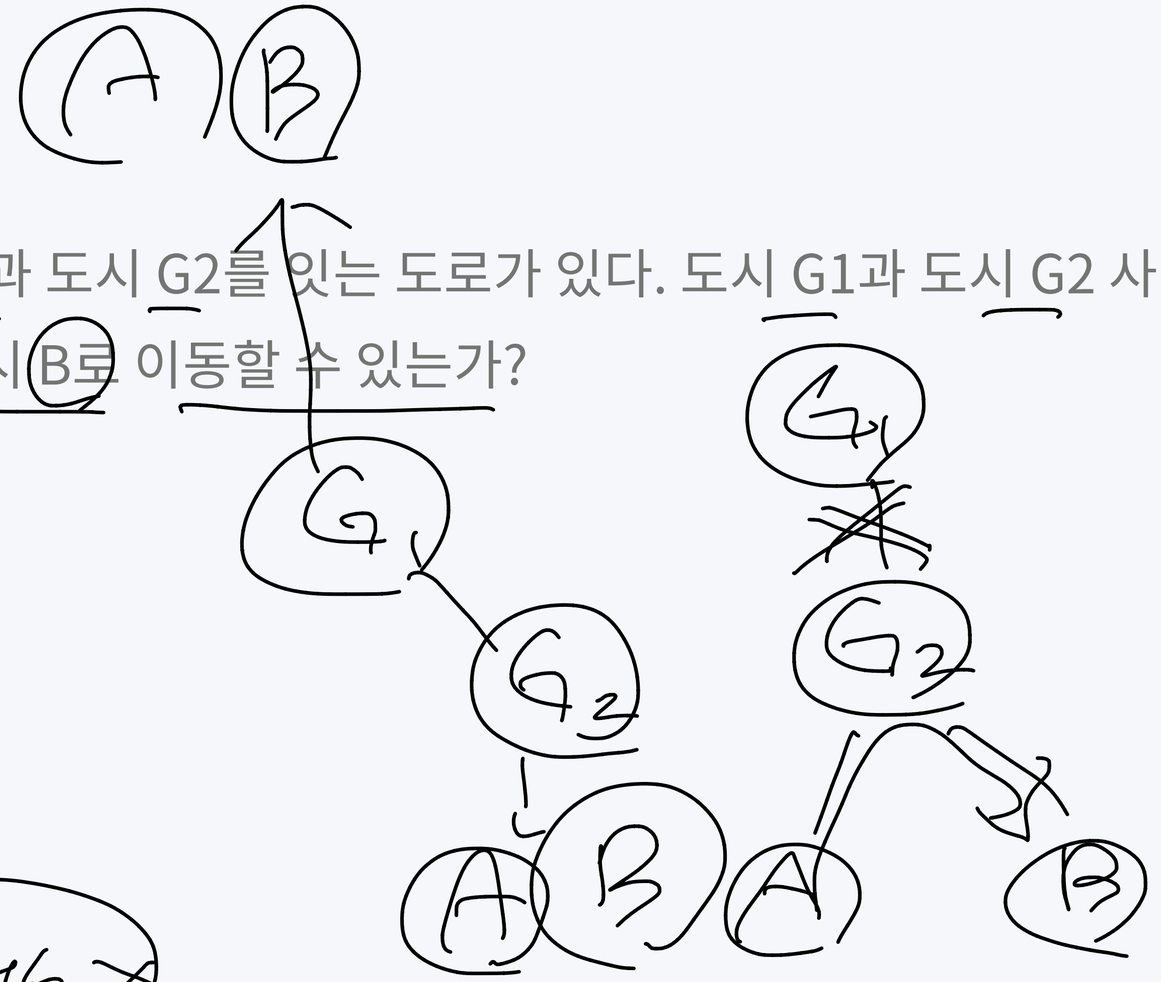
https://www.acmicpc.net/problem/1734

- find_related_child(A, B)
  - B가 A를 루트로 하는 서브 트리 안에 있을 때, 어떤 children이 B와 연결되어 있나?
- 이분 탐색을 이용해서 찾을 수 있다.
- left = 0
- right = A의 자식 개수 – 1

- discover[B] > finish[mid번째 자식] => left = mid + 1
- finish[B] < discover[mid번째 자식] => right = mid-1

교통 체계

https://www.acmicpc.net/problem/1734



- 두 개의 도시 A, B가 있고, 도시 G1과 도시 G2를 잇는 도로가 있다. 도시 G1과 도시 G2 사이의 도로를 없앤 후에도 도시 A에서 도시 B로 이동할 수 있는가?

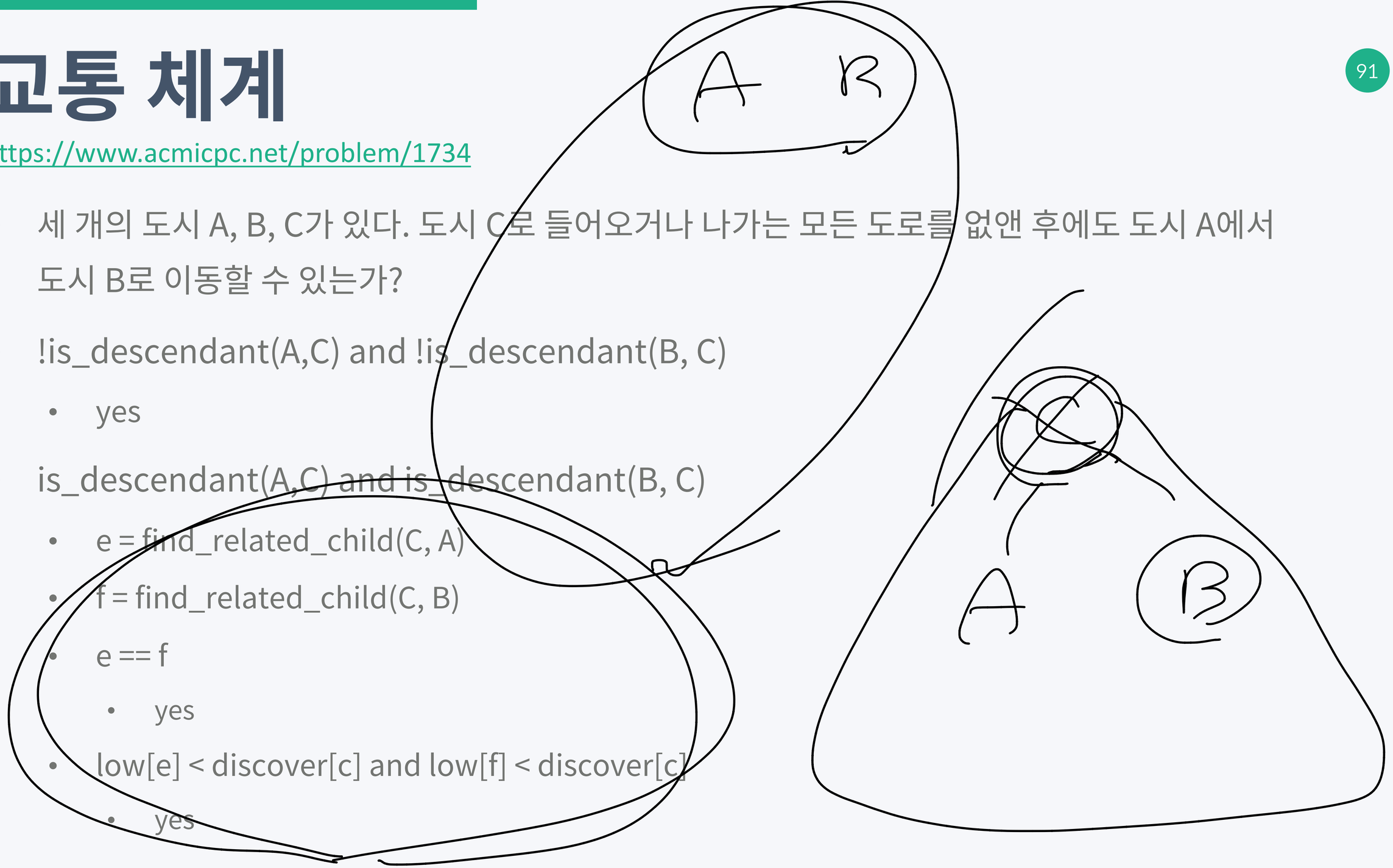
- if (is_descendant(G1,G2))
  - swap(G1,G2)

- yes인 경우
- low[G2] < discover[G2]
- is_descendant(A, G2) == is_descendant(B, G2)

교통 체계

https://www.acmicpc.net/problem/1734

- 세 개의 도시 A, B, C가 있다. 도시 C로 들어오거나 나가는 모든 도로를 없앤 후에도 도시 A에서 도시 B로 이동할 수 있는가?
- !is_descendant(A,C) and !is_descendant(B, C)
  - yes
- is_descendant(A,C) and is_descendant(B, C)
  - e = find_related_child(C, A)
  - f = find_related_child(C, B)
  - e == f
    - yes
  - low[e] < discover[c] and low[f] < discover[c]
  - yes



교통 체계

https://www.acmicpc.net/problem/1734

- 세 개의 도시 A, B, C가 있다. 도시 C로 들어오거나 나가는 모든 도로를 없앤 후에도 도시 A에서 도시 B로 이동할 수 있는가?

- else
- is_descendant(a,c)
  - swap(a,b)
- e = find_related_child(c, b)
- low[e] < discover[c]
  - yes

교통 체계

https://www.acmicpc.net/problem/1734

- C/C++: https://gist.github.com/Baekjoon/4618c135832808e8d31b

2-SAT

2-SAT

2-Satisfiability

- f = A ∧ B ∧ C ∧ D ∧ ⋯ Z
- 이런 식이 있을 때, 이 식을 true로 만들 수 있는가?
- f ≡ (x1 ∨ y1) ∧ (x2 ∨ y2) ∧ ⋯ ∧ (xn ∨ yn) 형식
- xi = true or false
- ∨ : or
- ∧ : and

f = true

x_1 = false

x_2 = false

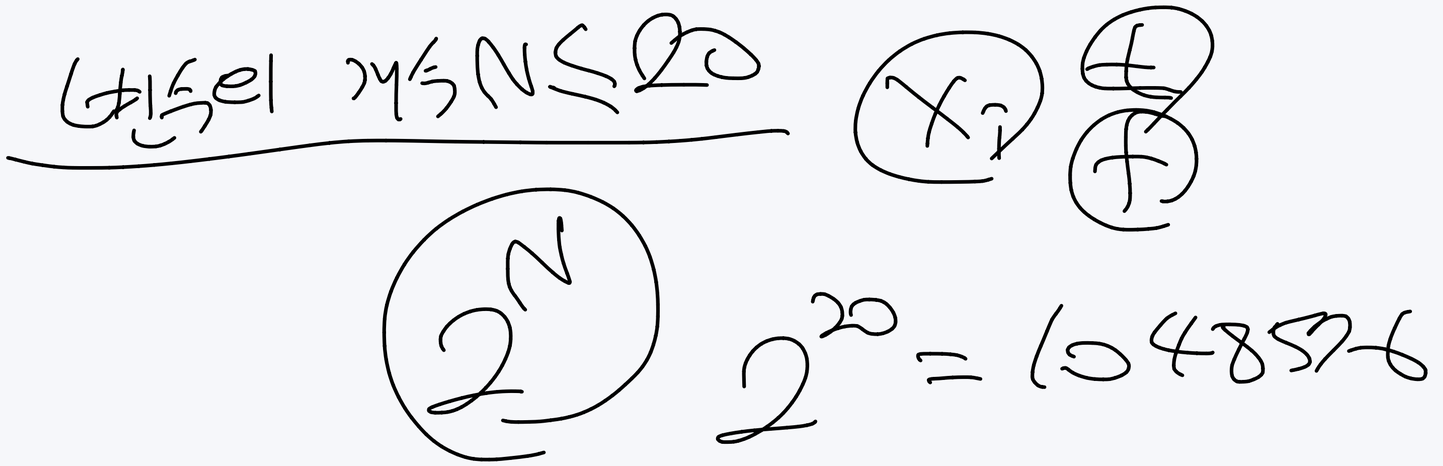
x_3 = true

$f = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee x_3) \wedge (x_3 \vee x_2)$

2-SAT - 1

https://www.acmicpc.net/problem/11277

- C/C++: https://gist.github.com/Baekjoon/075bbf88846baf424550



2-SAT - 2

https://www.acmicpc.net/problem/11278

- C/C++: https://gist.github.com/Baekjoon/0fdb2a0d5283147dd02a

2-SAT

2-Satisfiability

- xi ∨ yi는
- (¬xi => yi) ∧ (¬yi => xi)
- 와 같다

$x_i \vee y_i$

$(\neg x_i \rightarrow y_i) \wedge (\neg y_i \rightarrow x_i)$

2-SAT

2-Satisfiability

- 총 정점이 2n개인 vertex를 만들고
- xi ∨ yi 마다
- ¬xi => yi인 간선과
- ¬yi => xi인 간선을 추가

2-SAT

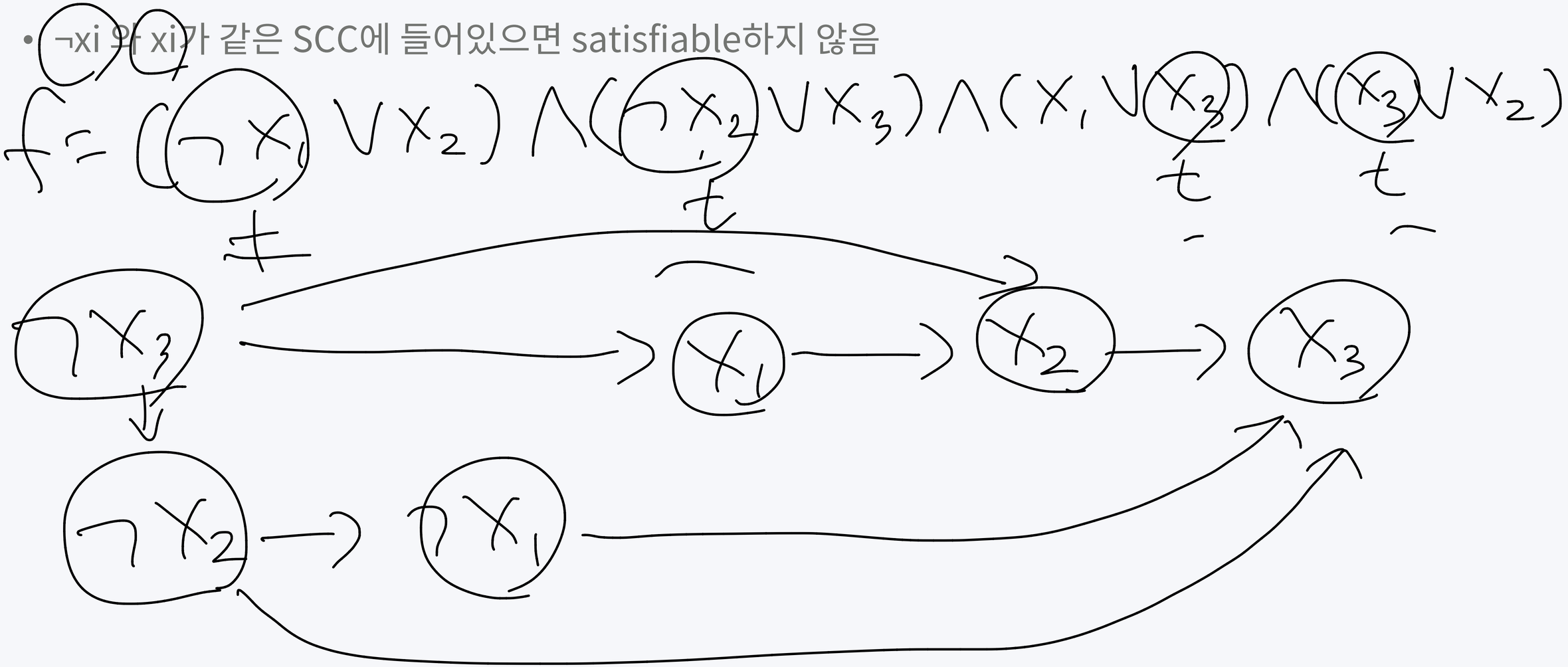
2-Satisfiability

$x \vee y$

$\neg x \rightarrow y$

$\neg y \rightarrow x$

- ¬xi 와 xi가 같은 SCC에 들어있으면 satisfiable하지 않음



2-SAT

2-Satisfiability

- SCC로 나눈 그래프는 DAG이기 때문에
- Topological Sort를 할 수 있음
- ¬xi가 xi보다 뒤에 있으면 xi는 false

2-SAT - 3

https://www.acmicpc.net/problem/11280

- C/C++: https://gist.github.com/Baekjoon/b64aae024d06aa905a0a

2-SAT - 4

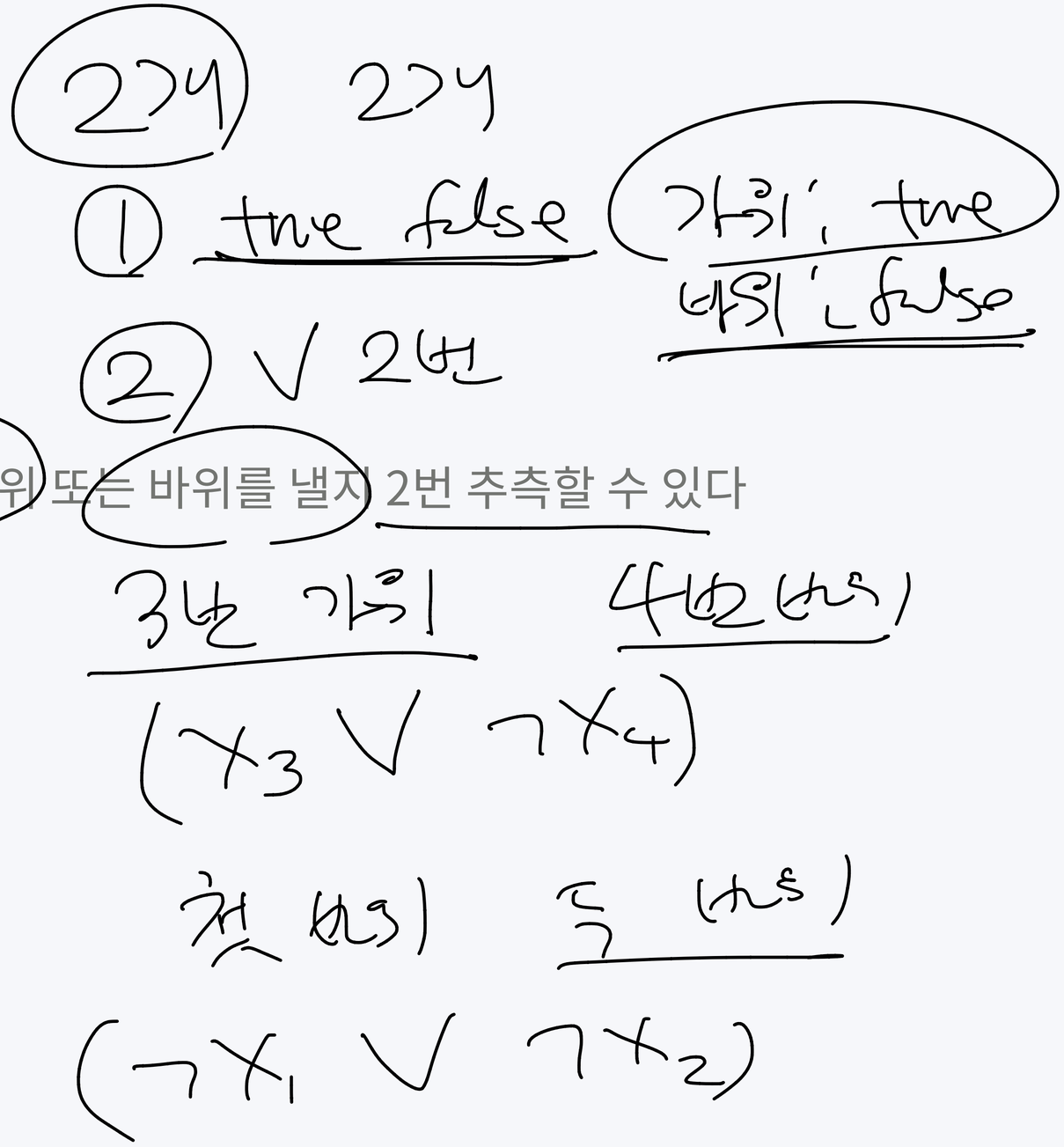
https://www.acmicpc.net/problem/11281

- C/C++: https://gist.github.com/Baekjoon/9490e4691bef79fb6dea

가위바위보

https://www.acmicpc.net/problem/2207

- 원장 선생님은 가위와 바위만 낸다
- M번 혼자 가위바위보를 하고
- 학생 N명은 원장 선생님이 몇 번째에 가위 또는 바위를 낼지 2번 추측할 수 있다



가위바위보

https://www.acmicpc.net/problem/2207

- 1~m : xi를 나타냄
- m+1~2*m: ¬xi를 나타냄

사랑과 전쟁

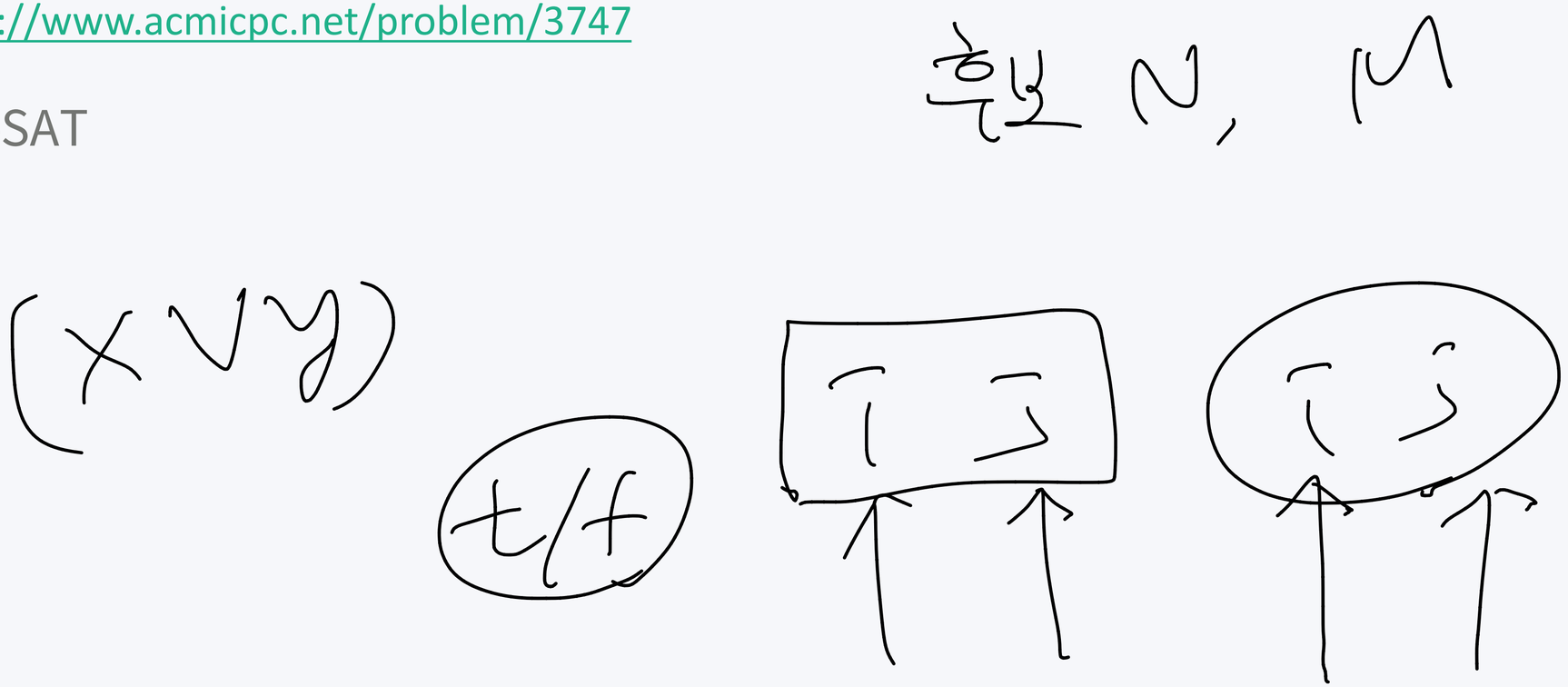
https://www.acmicpc.net/problem/4230

- 2-SAT + true false 찾기

Perfect Election

https://www.acmicpc.net/problem/3747

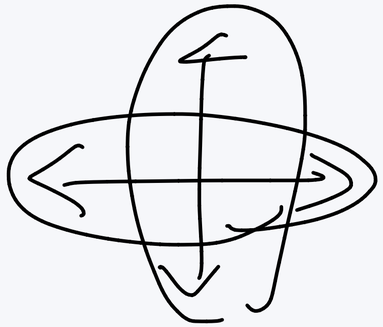
- 2-SAT



TORNJEVI

https://www.acmicpc.net/problem/3153

- 왼쪽으로 쏴야하면 Li = true
- 오른쪽으로 쏴야하면 Li = false (¬Li = true)
- 위로 쏴야 하면 Ui = true
- 아래로 쏴야 하면 Ui = false (¬Ui = true)



TORNJEVI

https://www.acmicpc.net/problem/3153

- 2-SAT을 하기 전에 미리 정할 수 있는 값도 있다
- 두 타워가 같은 row에 있고
- 그 사이에 castle이 없으면 서로를 쏠 수 없다