

# 네트워크 플로우

최백준 [choi@startlink.io](mailto:choi@startlink.io)

---

# 최대 유량

---

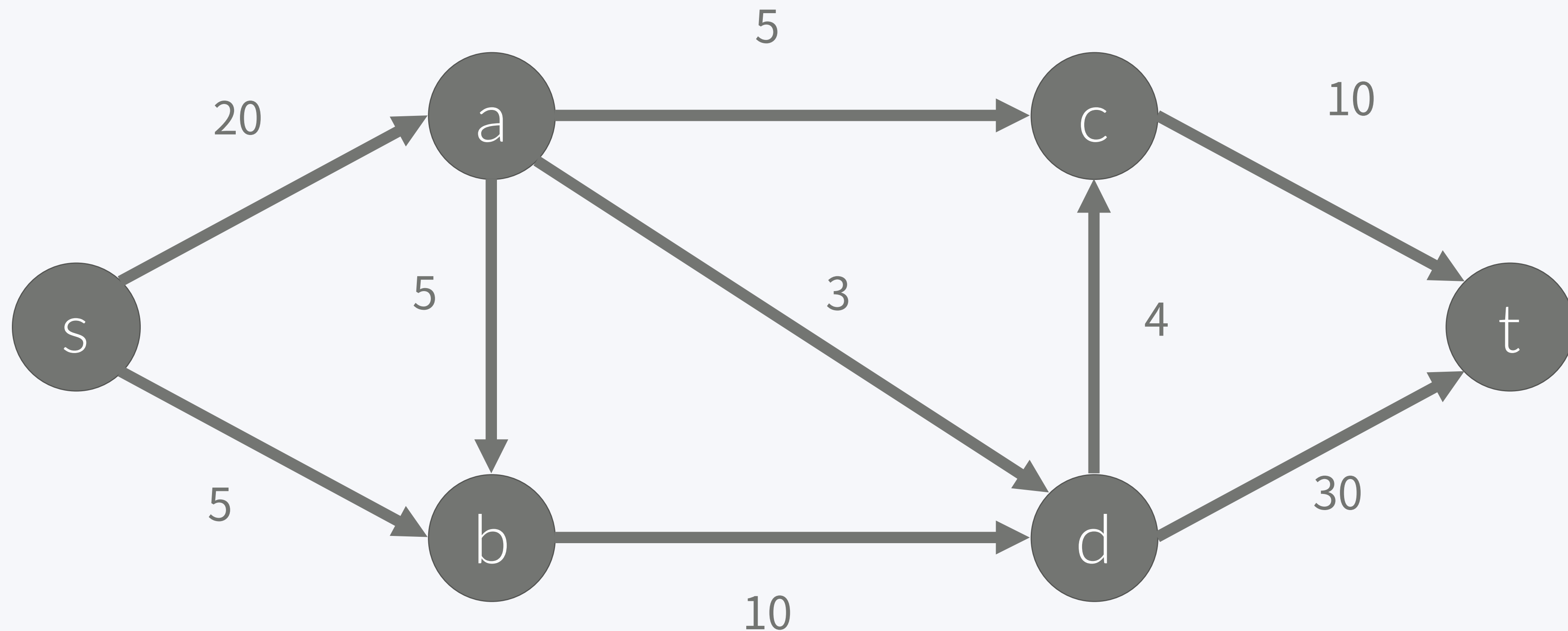
# 최대 유량

Maximum Flow

- 각 간선이 나타내는 것은 흐를 수 있는 양
- s에서 t로 최대 얼마나 흐를 수 있는가?

가중치, weight

비용 Cost 3  
용량 Capacity

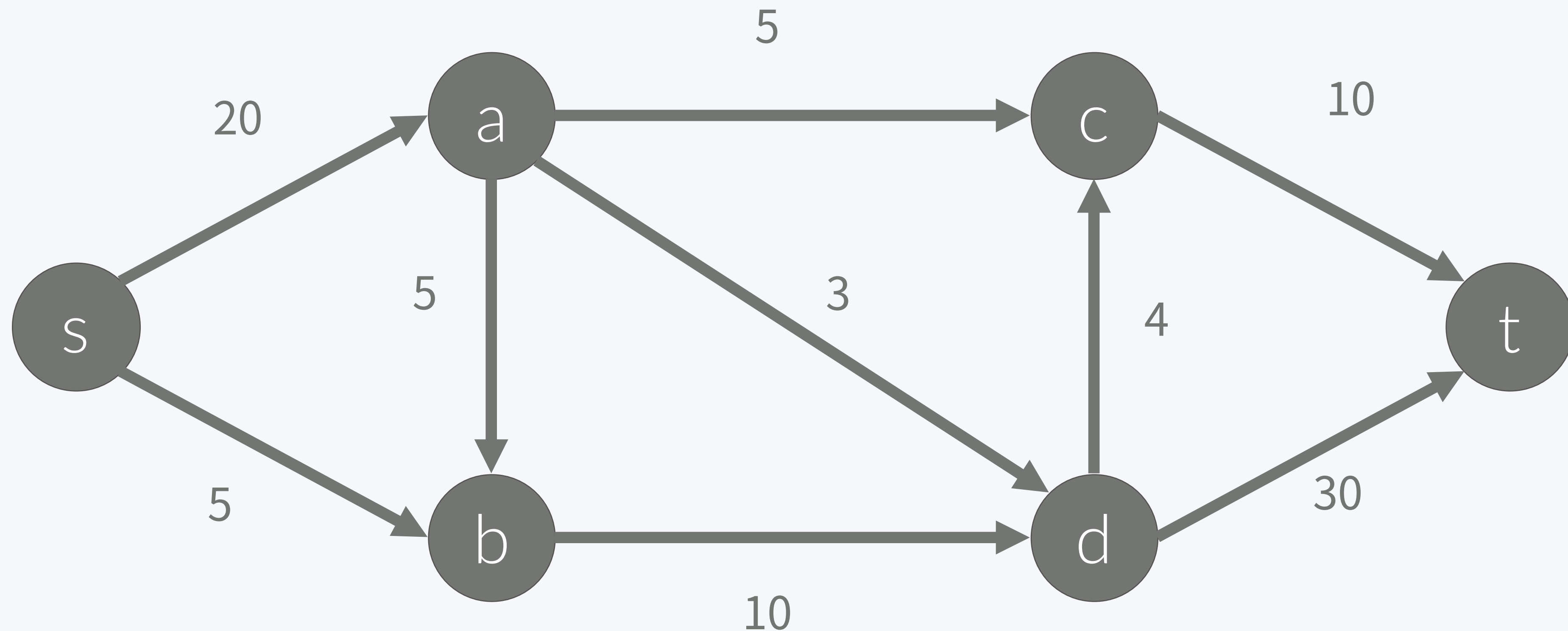


# 최대 유량

Maximum Flow

4

- $s \rightarrow a$ 로 최대 20만큼만 흐를 수 있다.
- $a \rightarrow c$ 로 최대 5만큼 흐를 수 있다.

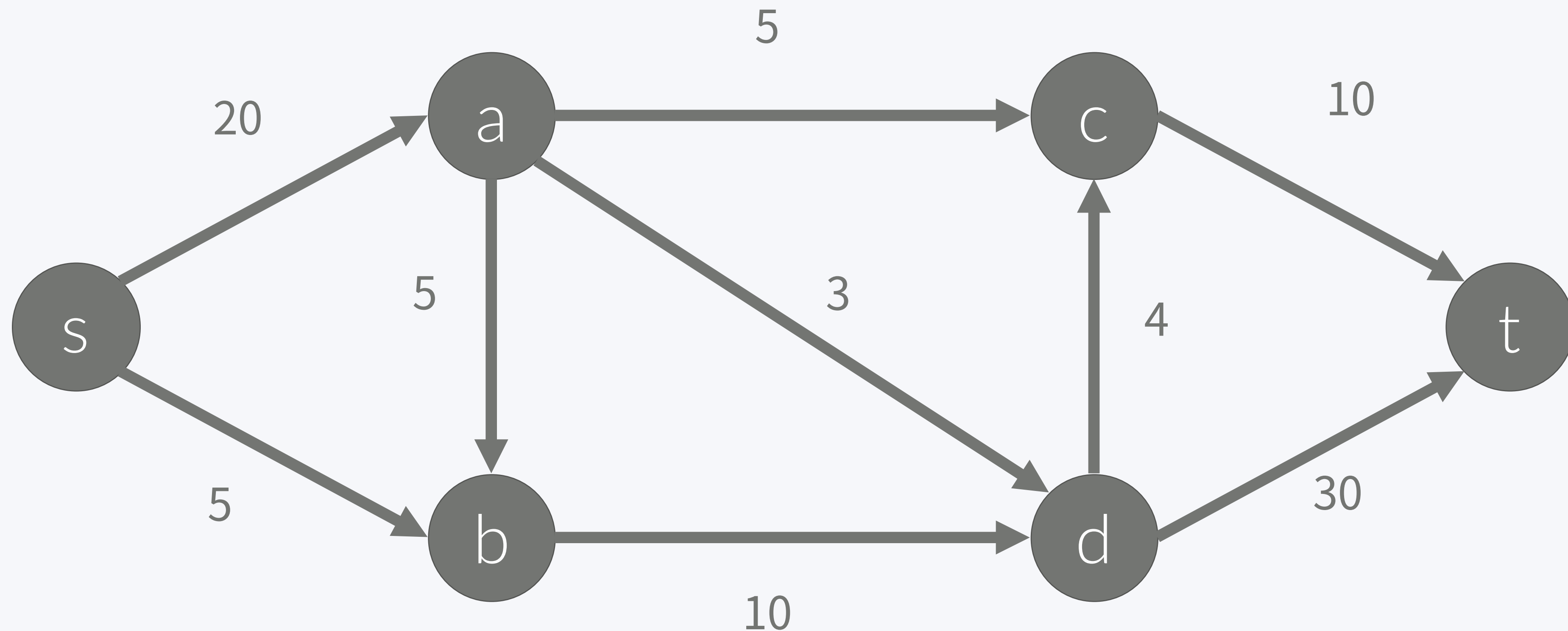


# 최대 유량

Maximum Flow

5

- $s \rightarrow a$ 로 20을 보냈다고 하더라도
- $a \rightarrow c, a \rightarrow d, a \rightarrow b$ 는  $5+3+5 = 13$ 이기 때문에, 13 이상을 보낼 수 없다

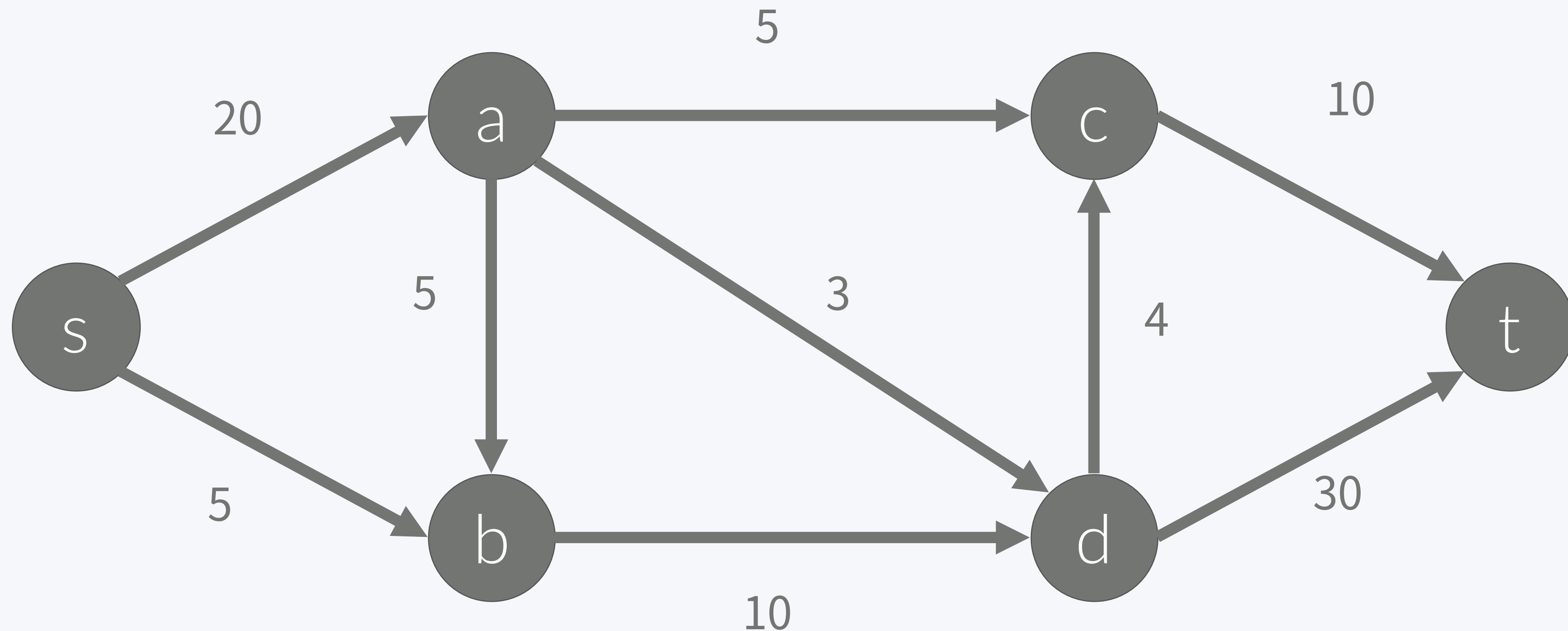


# 최대 유량

Maximum Flow

6

- s에서 t로 최대 얼마나 보낼 수 있는지를 구하는 문제

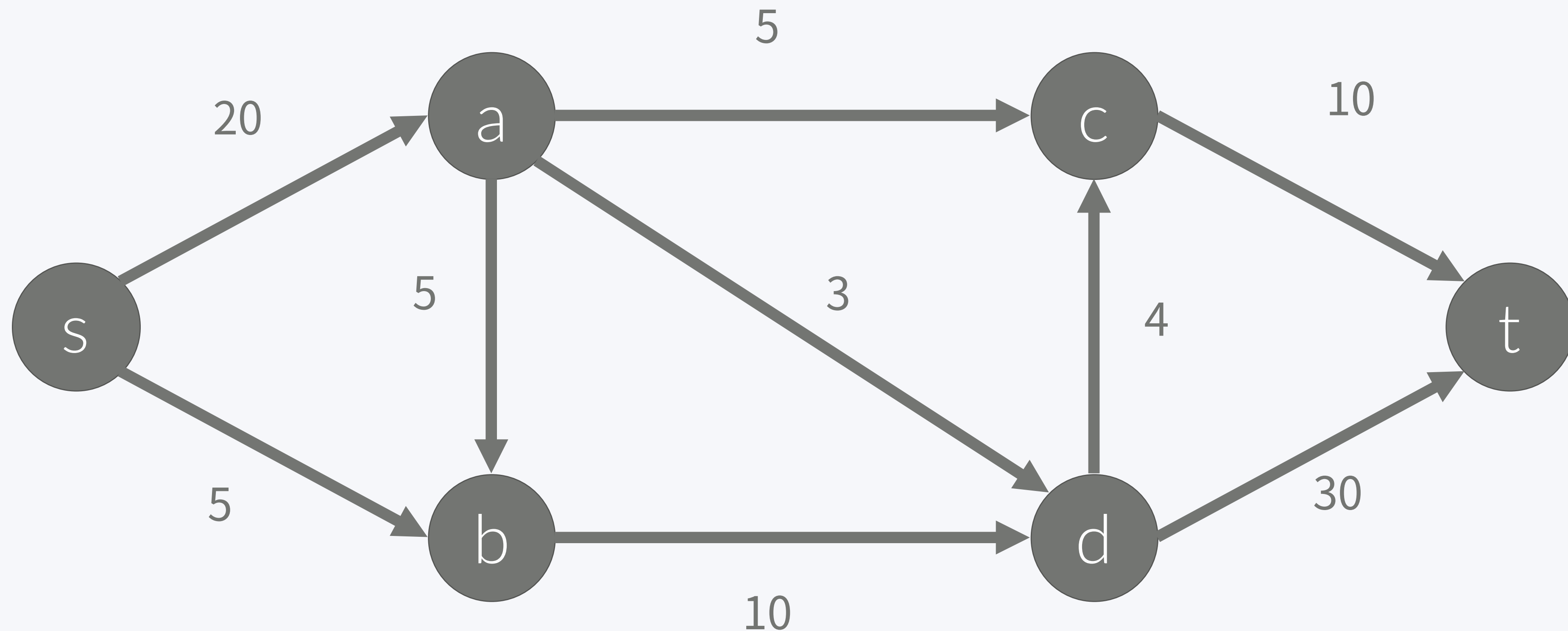


# 최대 유량

Maximum Flow

- 간선에 나타나있는 것: capacity
- 실제 그 간선을 따라서 흐른 양: flow

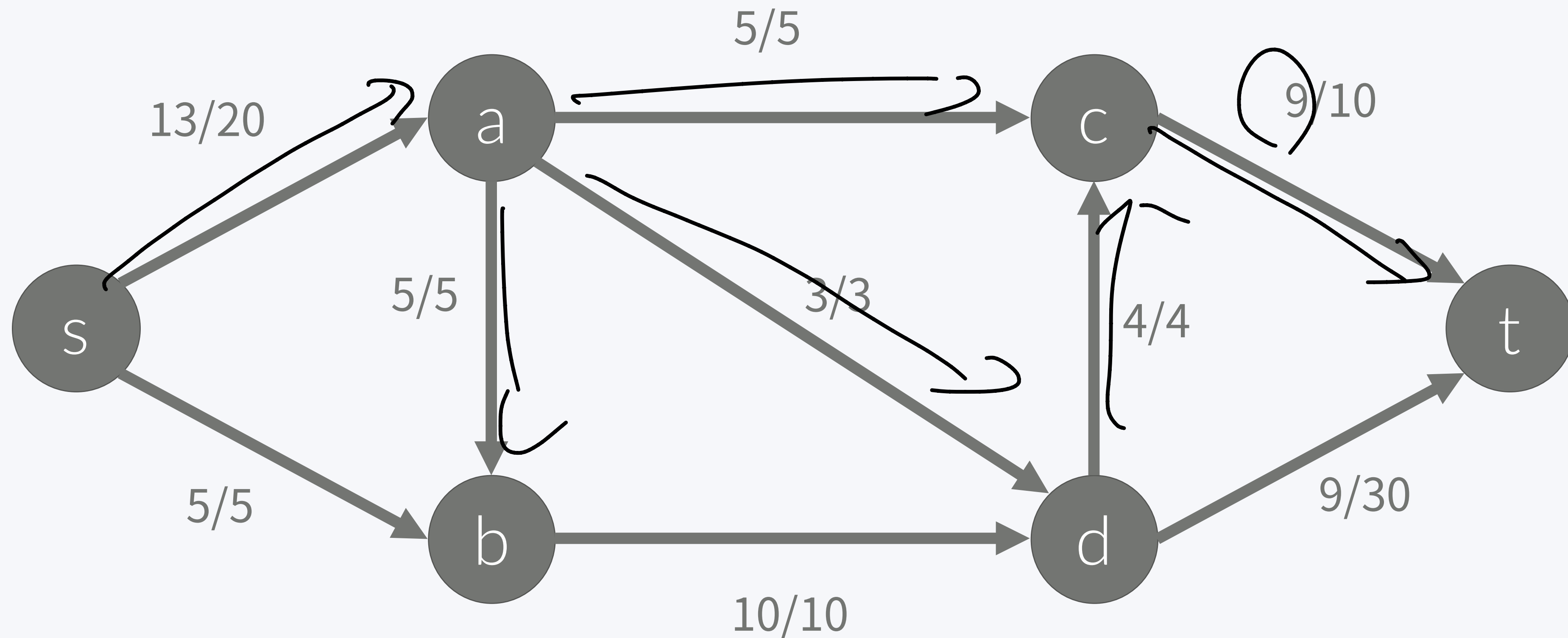
flow / Capacity  
흐른 양 / 용량



# 최대 유량

Maximum Flow

- flow/capacity



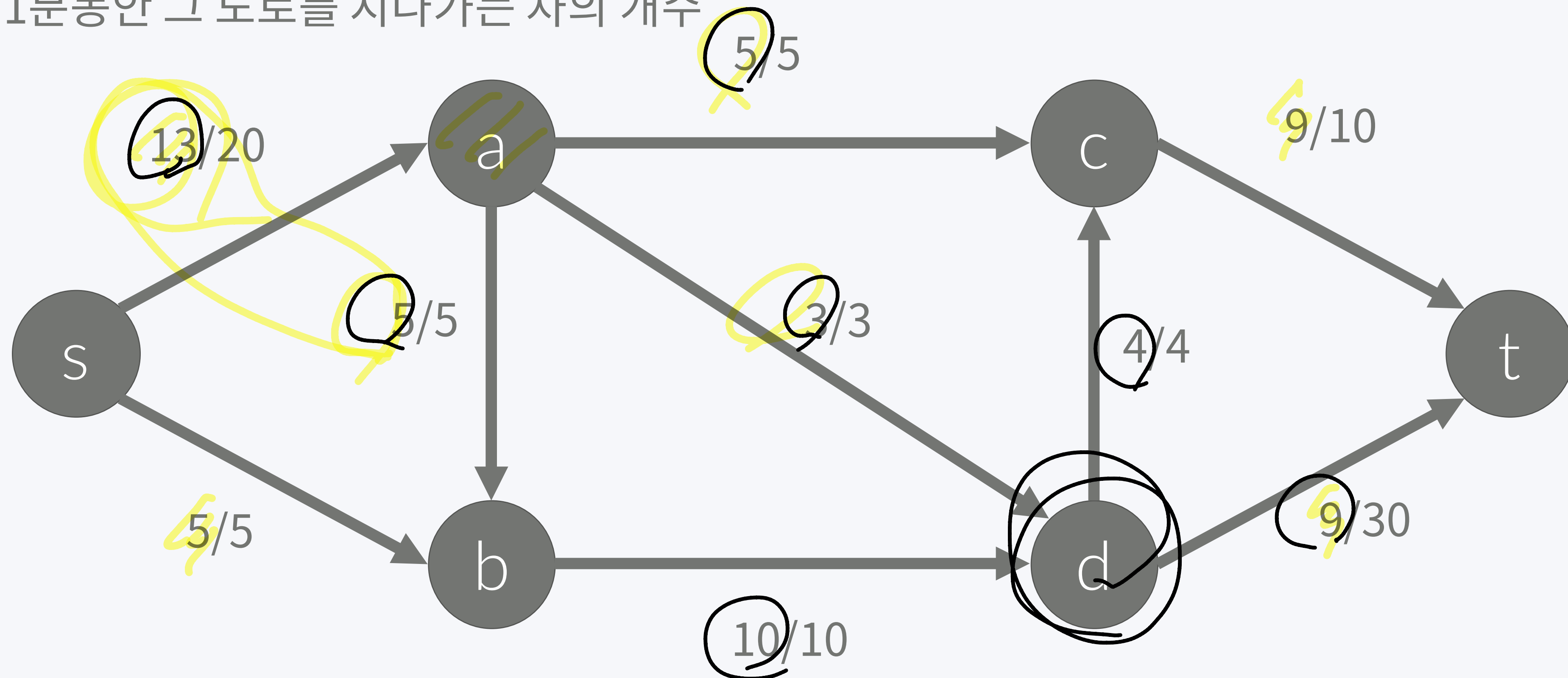


# 최대 유량

Maximum Flow

9

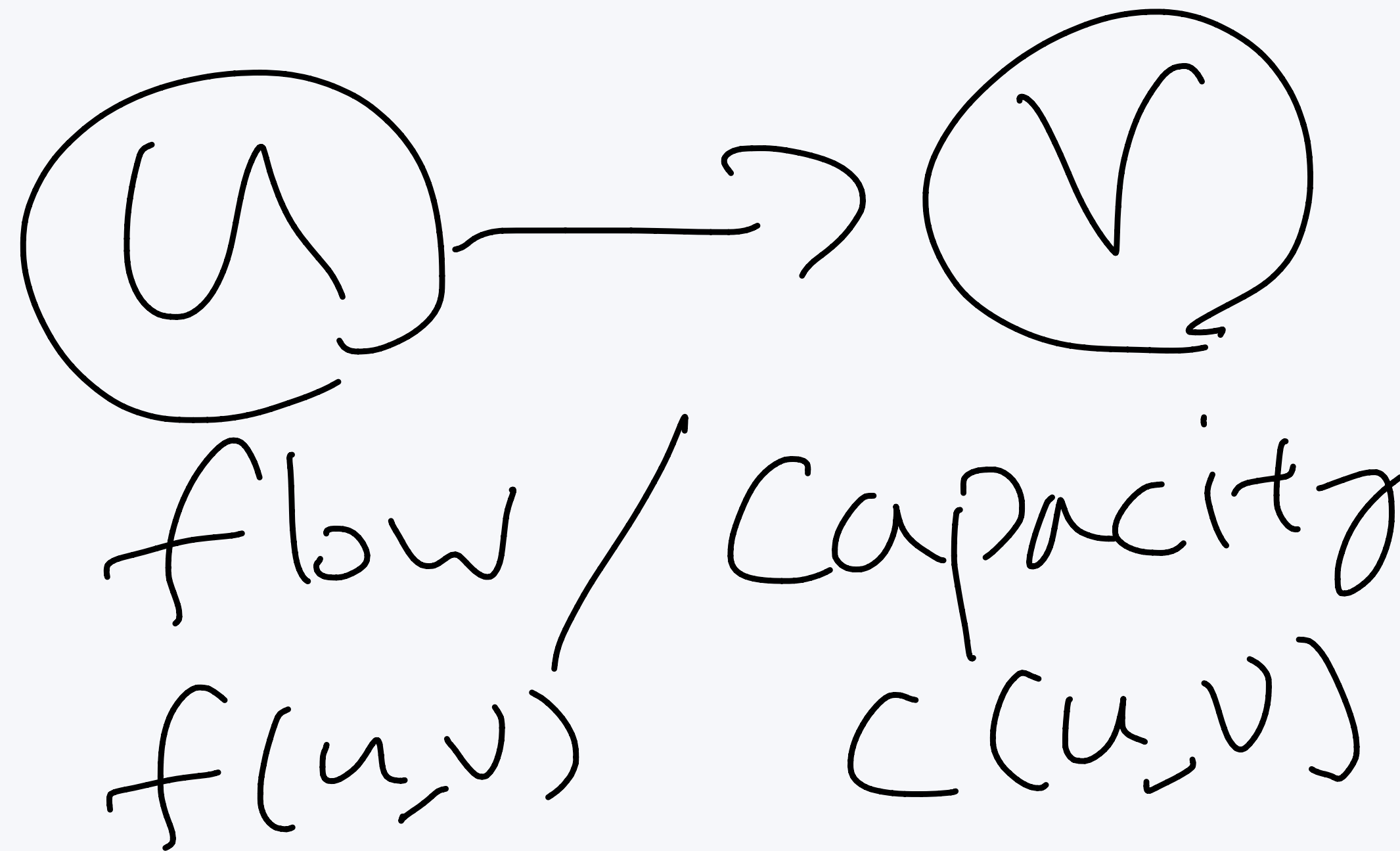
- 다른 예시 vertex: 교차로, edge: 도로
- capacity: 1분동안 그 도로를 지나갈 수 있는 차의 개수
- flow: 1분동안 그 도로를 지나가는 차의 개수



# 최대 유량

Maximum Flow

- 그래프 G에서
- $u \rightarrow v$  간선의 용량:  $c(u, v)$
- $u \rightarrow v$  간선의 흐른 양:  $f(u, v)$



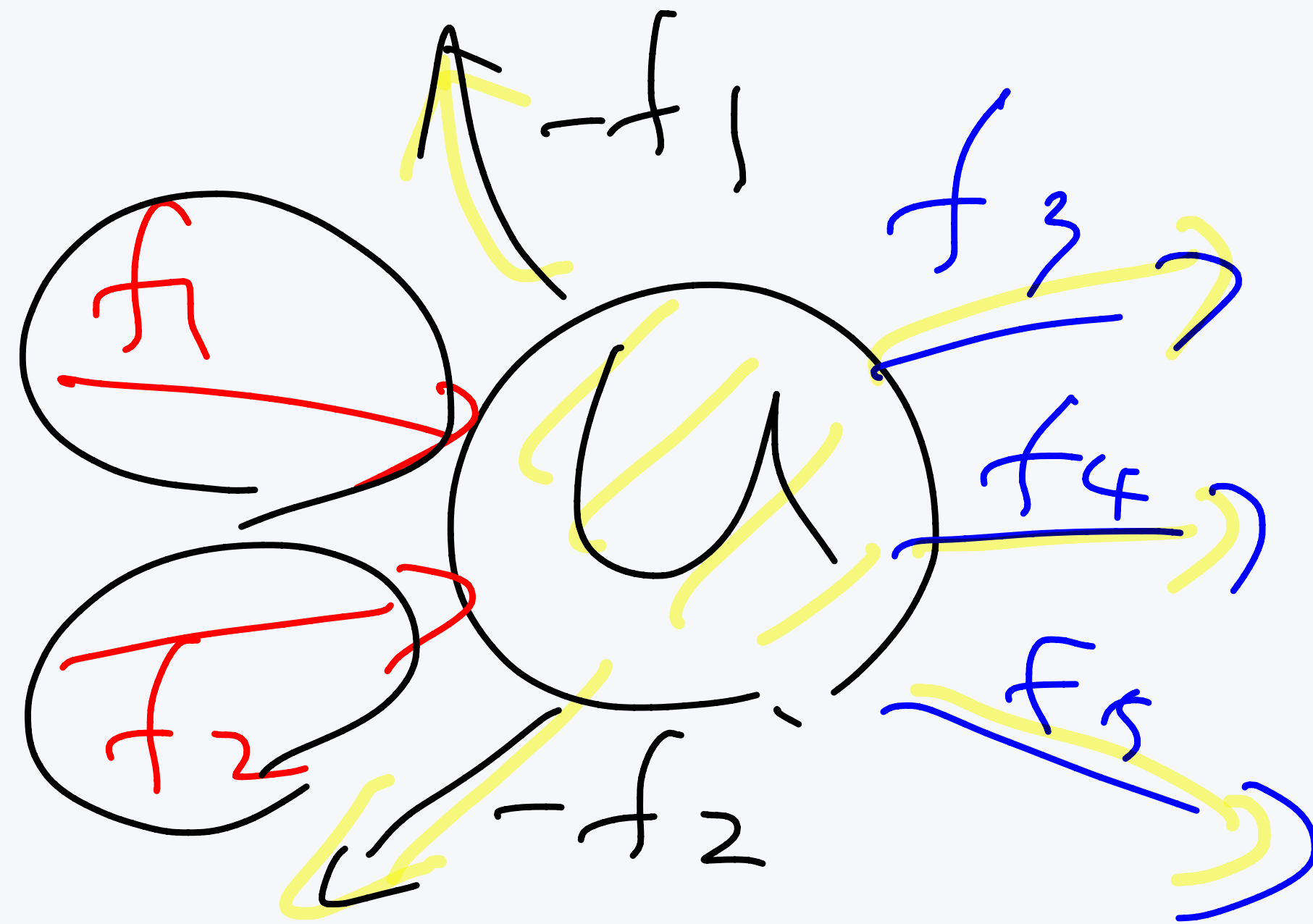
$$① f(u, v) \leq c(u, v)$$

$$② f(u, v) = -f(v, u)$$

# 최대 유량

## Maximum Flow

- 세가지 속성을 만족해야 한다
- Capacity constraint
  - $f(u, v) \leq c(u, v)$
- Skew symmetry
  - $f(u, v) = -f(v, u)$
  - $u \rightarrow v$ 로  $x$ 를 보냈으면,  $v \rightarrow u$ 로는  $-x$ 를 보낸다고 한다
- Flow conservation
  - $\sum_{v \in V} f(u, v) = 0$
  - $u$ 와 연결된 모든 간선의 흐른양의 합은 0이다
  - Skew symmetry에 의해서 음수를 저장했기 때문



$$f_1 + f_2 = f_3 + f_4 + f_5$$

# 최대 유량

Maximum Flow

12

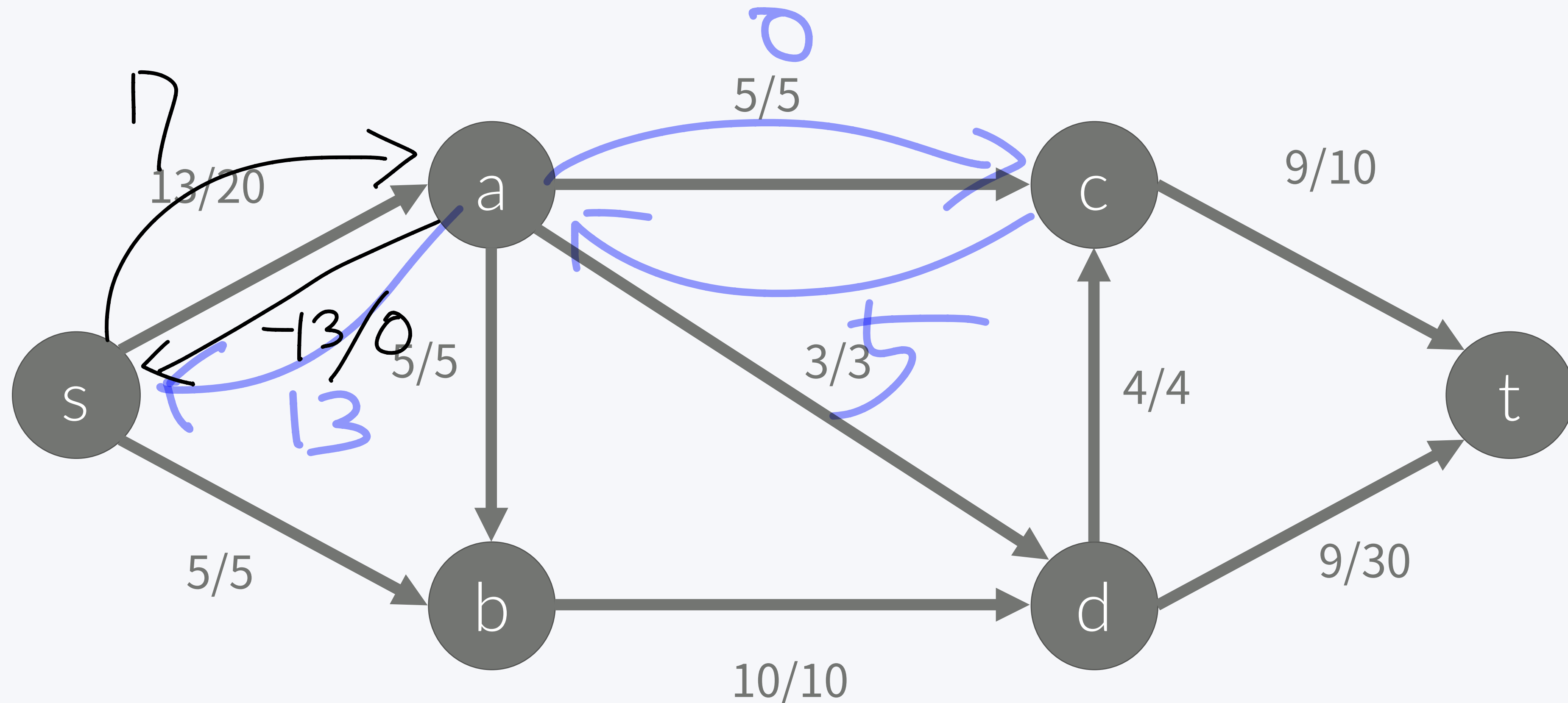
- Source (s): 시작
- Sink (t): 끝
- 총 흐른 양
  - Source에서 나간 양
  - 또는
  - Sink로 들어온 양

# Residual Capacity

13

Maximum Flow

- $cf(u, v) = c(u, v) - f(u, v)$
- Available Capacity

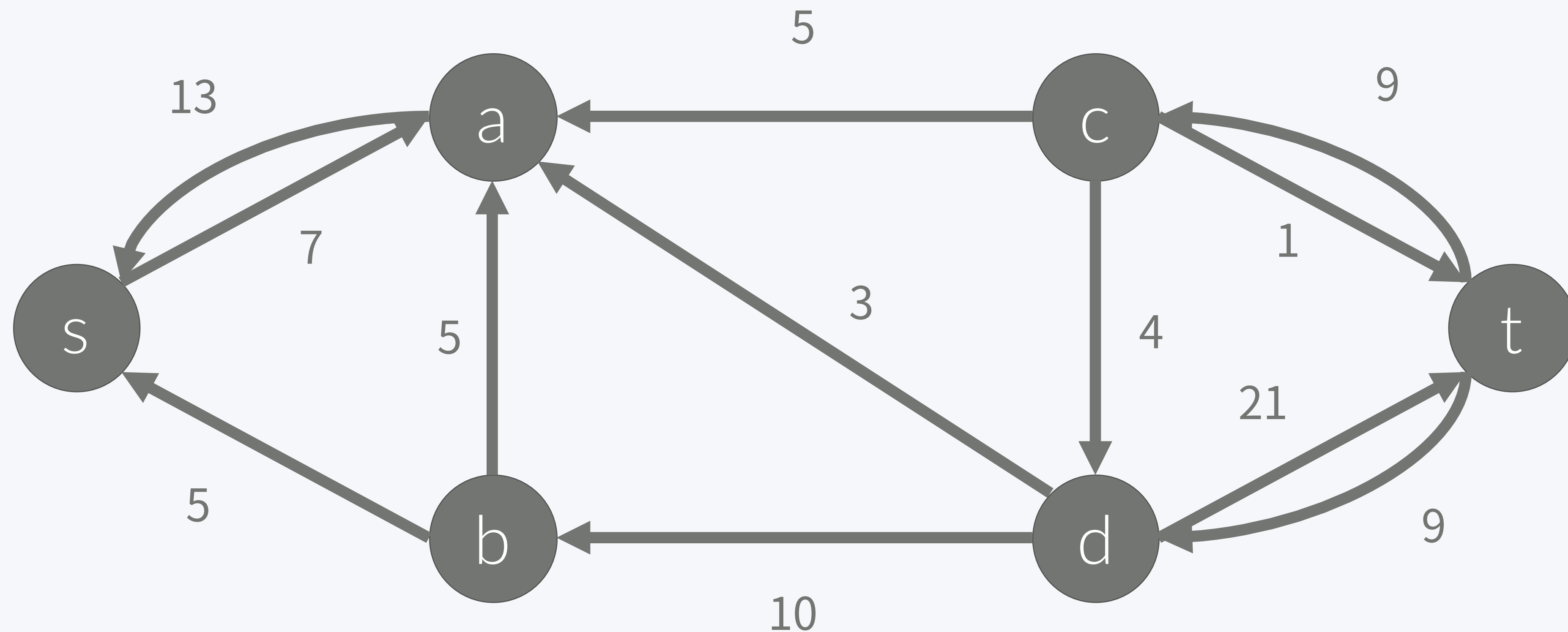


# Residual Capacity

14

Maximum Flow

- $cf(u, v) = c(u, v) - f(u, v)$

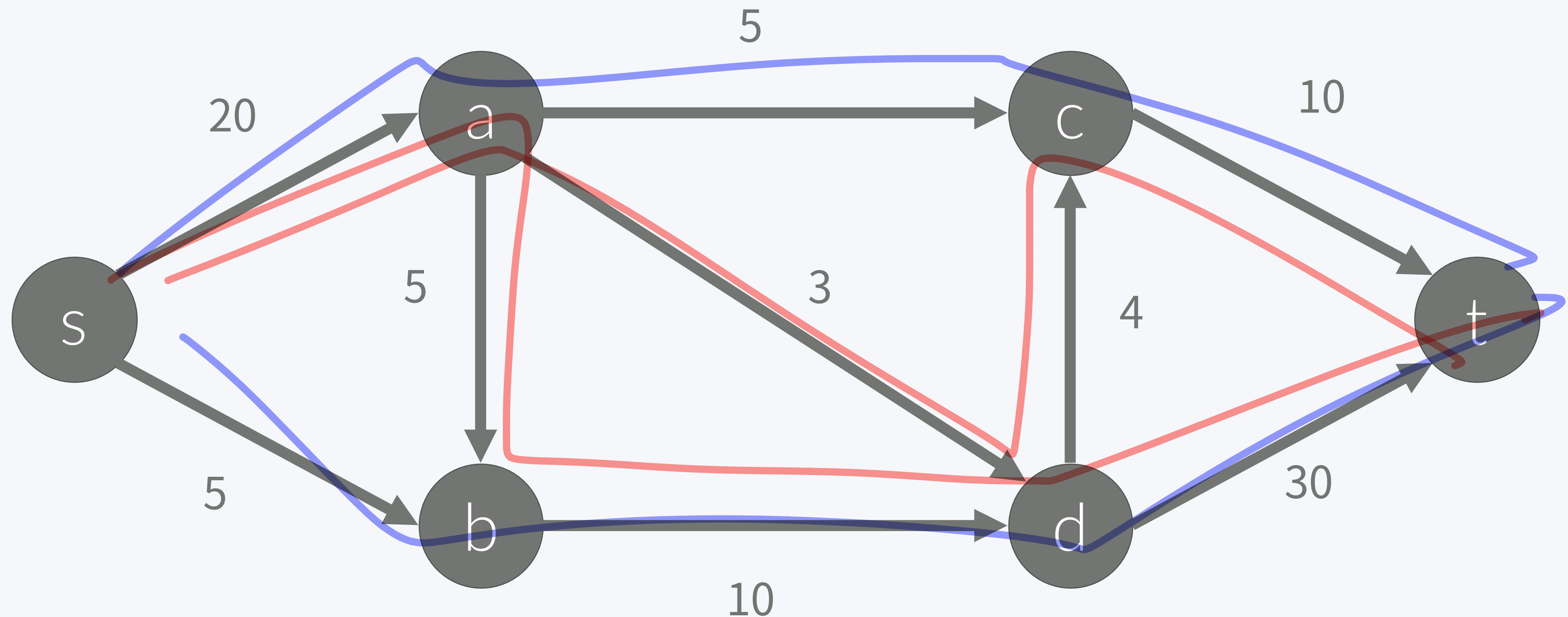


# Augmenting Path

15

Maximum Flow

- Residual network 에서 구한다
- $u_1, u_2, \dots, u_k$  ( $u_1 = \text{Source}$ ,  $u_k = \text{Sink}$ ),  $cf(u_i, u_{i+1}) > 0$



# Ford-Fulkerson

---



# Ford-Fulkerson

## Maximum Flow

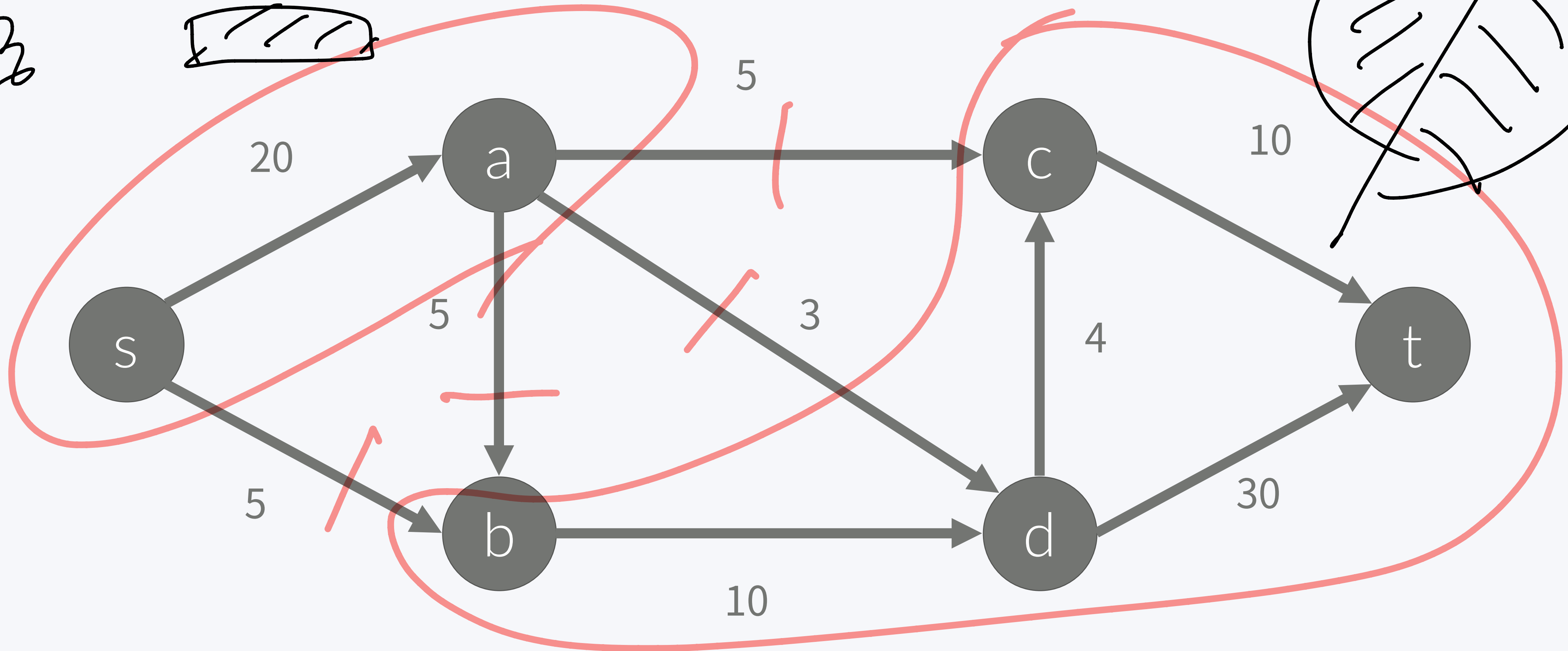
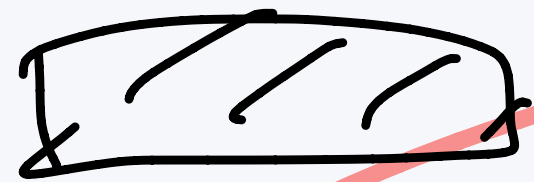
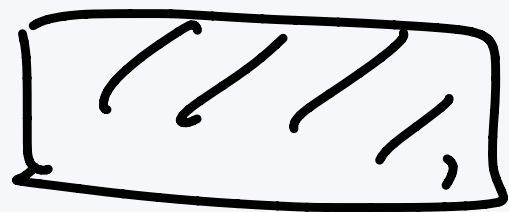
- 최대 유량을 구하는 알고리즘
1. Augmenting Path를 DFS를 이용해서 구한다.
  2.  $m = \text{Augmenting Path 상에서의 최소값}$ 을 구한다
  3.  $(u_i, u_{i+1})$  방향의 Residual Capacity에서  $m$ 을 뺀다
  4.  $(u_{i+1}, u_i)$  방향의 Residual Capacity에  $m$ 을 더한다.
  5. 위의 과정을 Augmenting Path를 못 구할때 까지 계속 한다

# Ford-Fulkerson

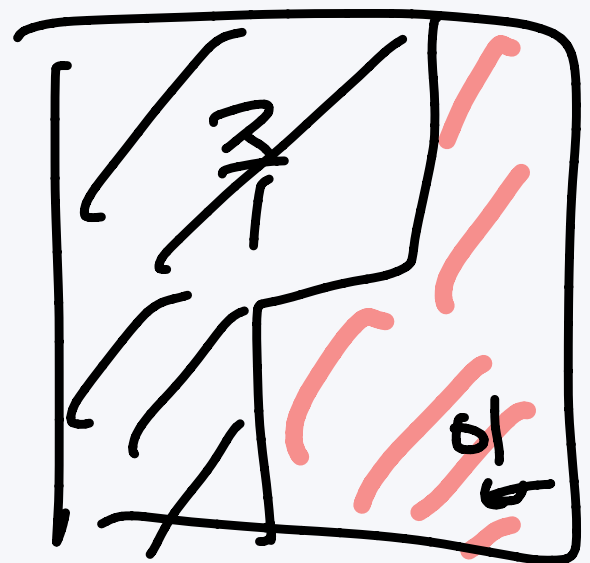
Maximum Flow

- 그래프

드라이브  
여기



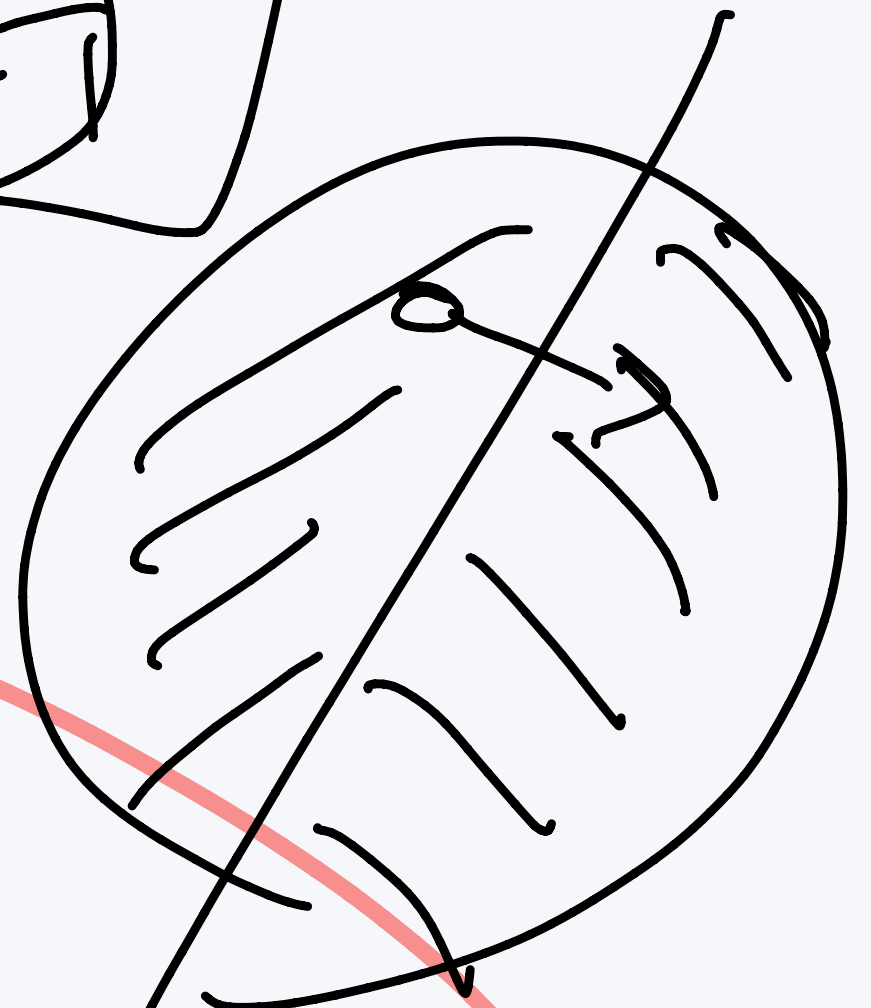
0.124  
0.72



2.54 / 6.04

2.54  
1.6

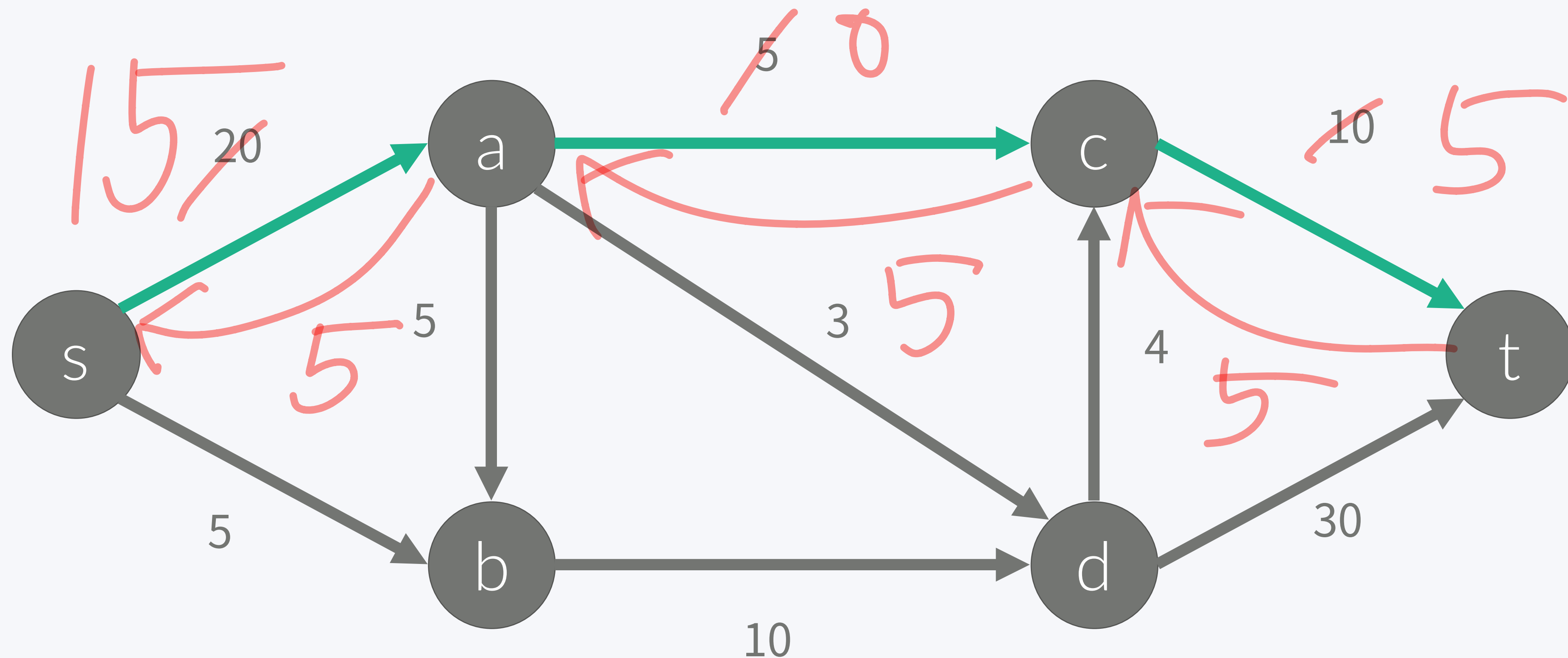
0.12  
0.72



# Ford-Fulkerson

Maximum Flow

- Augmenting Path:  $s \rightarrow a \rightarrow c \rightarrow t$
- $m = 5$

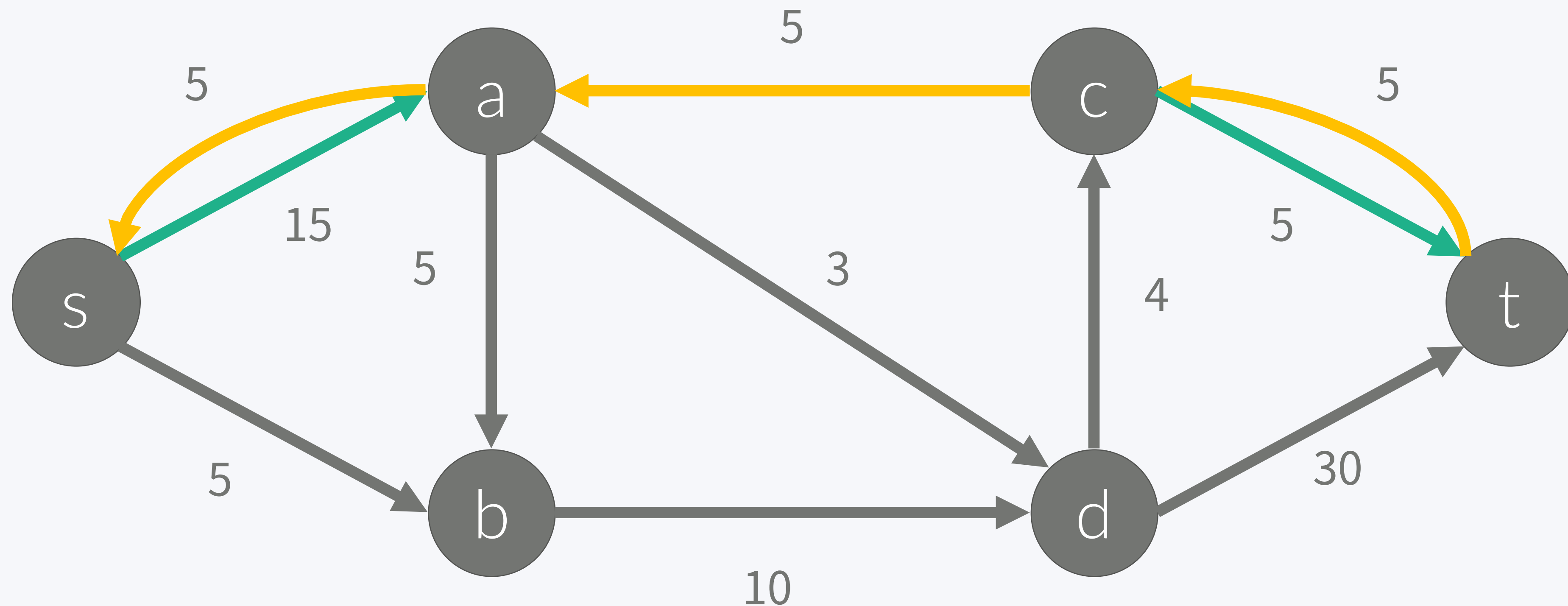


# Ford-Fulkerson

20

Maximum Flow

- Augmenting Path:  $s \rightarrow a \rightarrow c \rightarrow t$
- $m = 5$

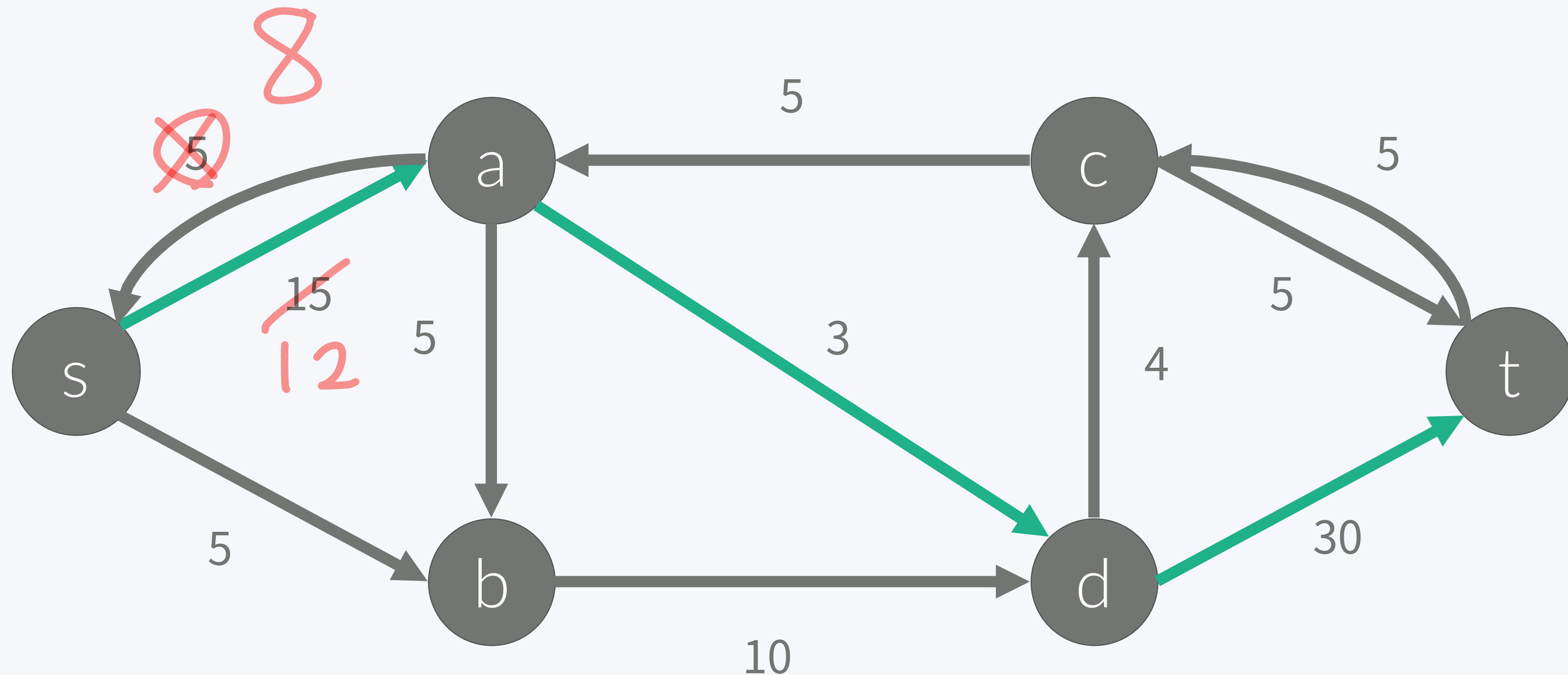


# Ford-Fulkerson

21

Maximum Flow

- Augmenting Path:  $s \rightarrow a \rightarrow d \rightarrow t$
- $m = 3$

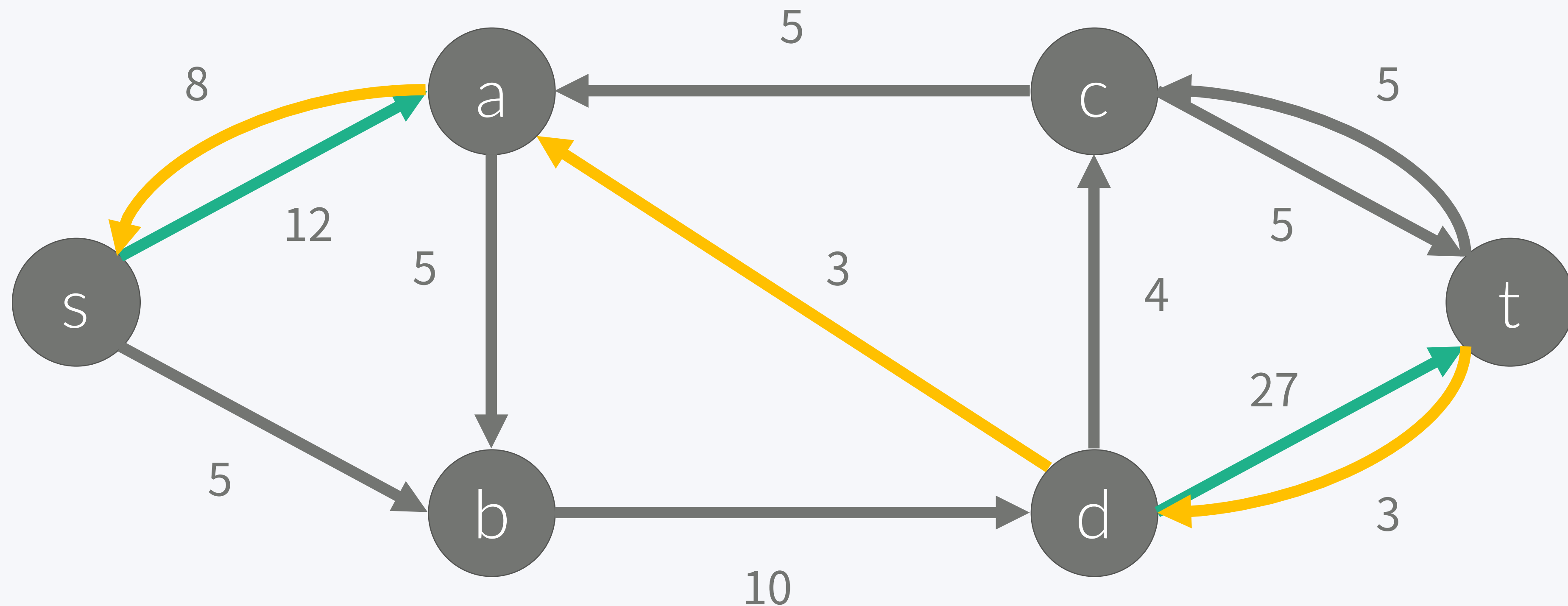


# Ford-Fulkerson

22

Maximum Flow

- Augmenting Path:  $s \rightarrow a \rightarrow d \rightarrow t$
- $m = 3$

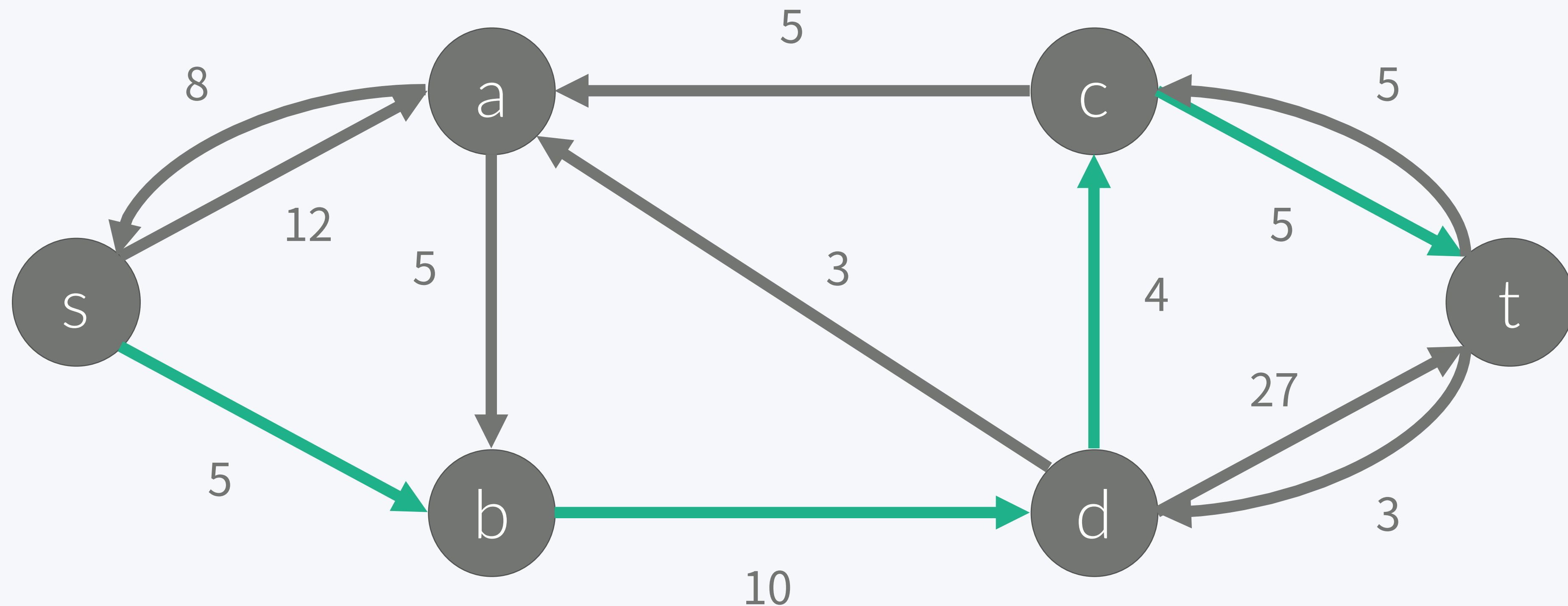


# Ford-Fulkerson

23

Maximum Flow

- Augmenting Path:  $s \rightarrow b \rightarrow d \rightarrow c \rightarrow t$
- $m = 4$

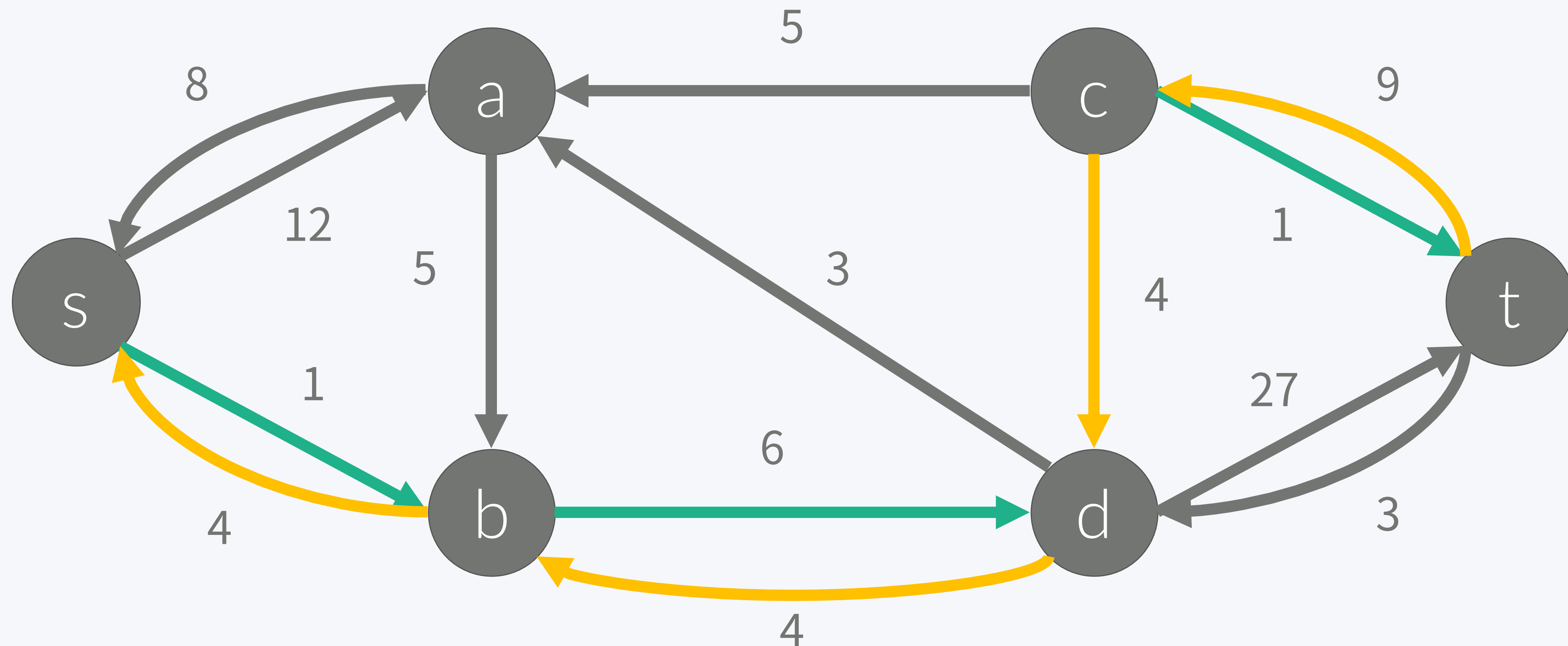


# Ford-Fulkerson

24

Maximum Flow

- Augmenting Path:  $s \rightarrow b \rightarrow d \rightarrow c \rightarrow t$
- $m = 4$



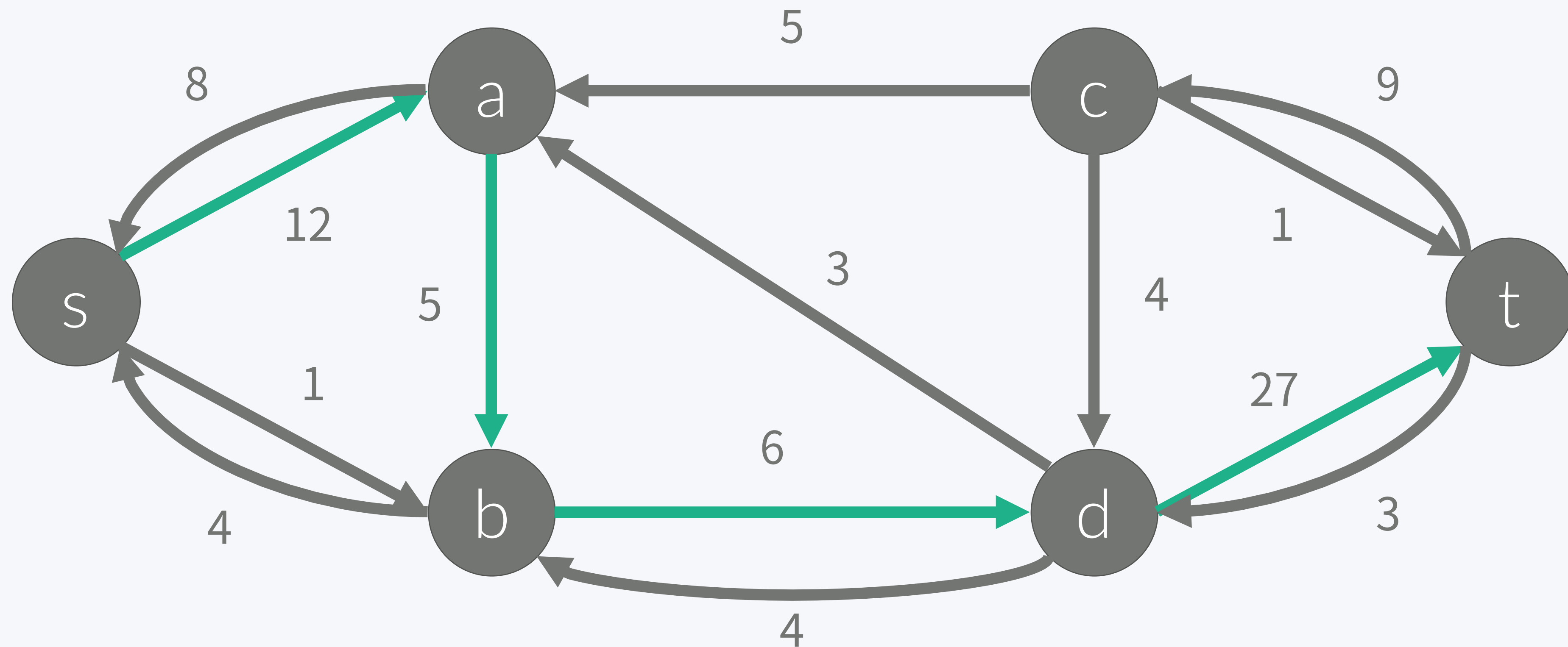


# Ford-Fulkerson

25

Maximum Flow

- Augmenting Path:  $s \rightarrow a \rightarrow b \rightarrow d \rightarrow t$
- $m = 5$

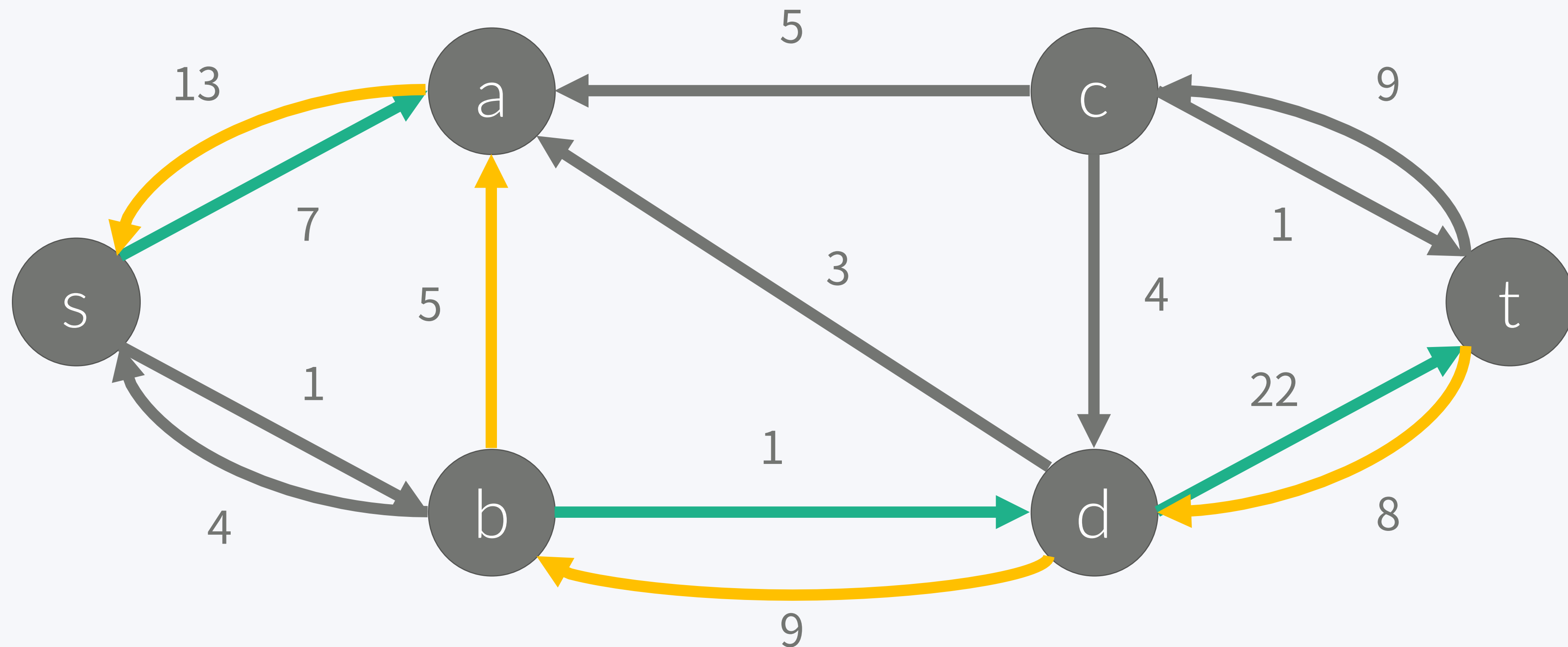


# Ford-Fulkerson

26

Maximum Flow

- Augmenting Path:  $s \rightarrow a \rightarrow b \rightarrow d \rightarrow t$
- $m = 5$

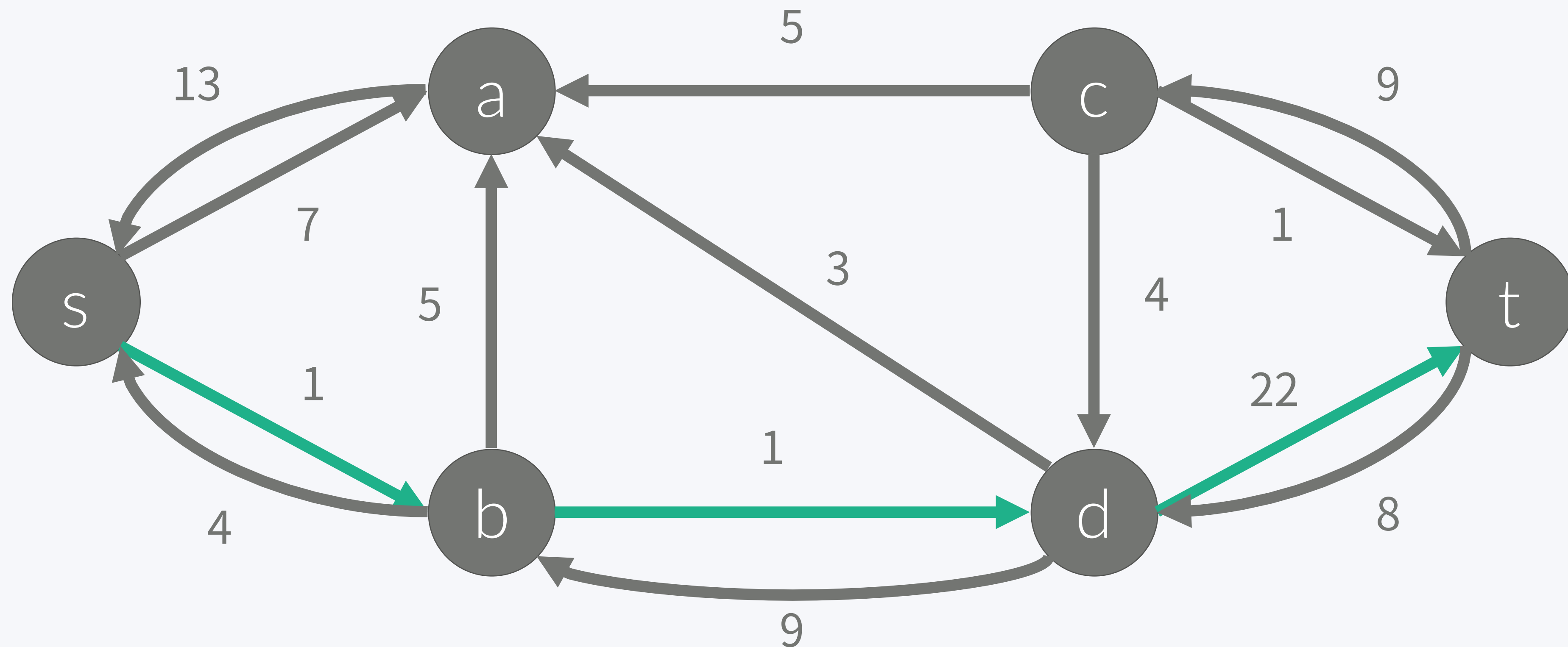


# Ford-Fulkerson

27

Maximum Flow

- Augmenting Path:  $s \rightarrow b \rightarrow d \rightarrow t$
- $m = 1$

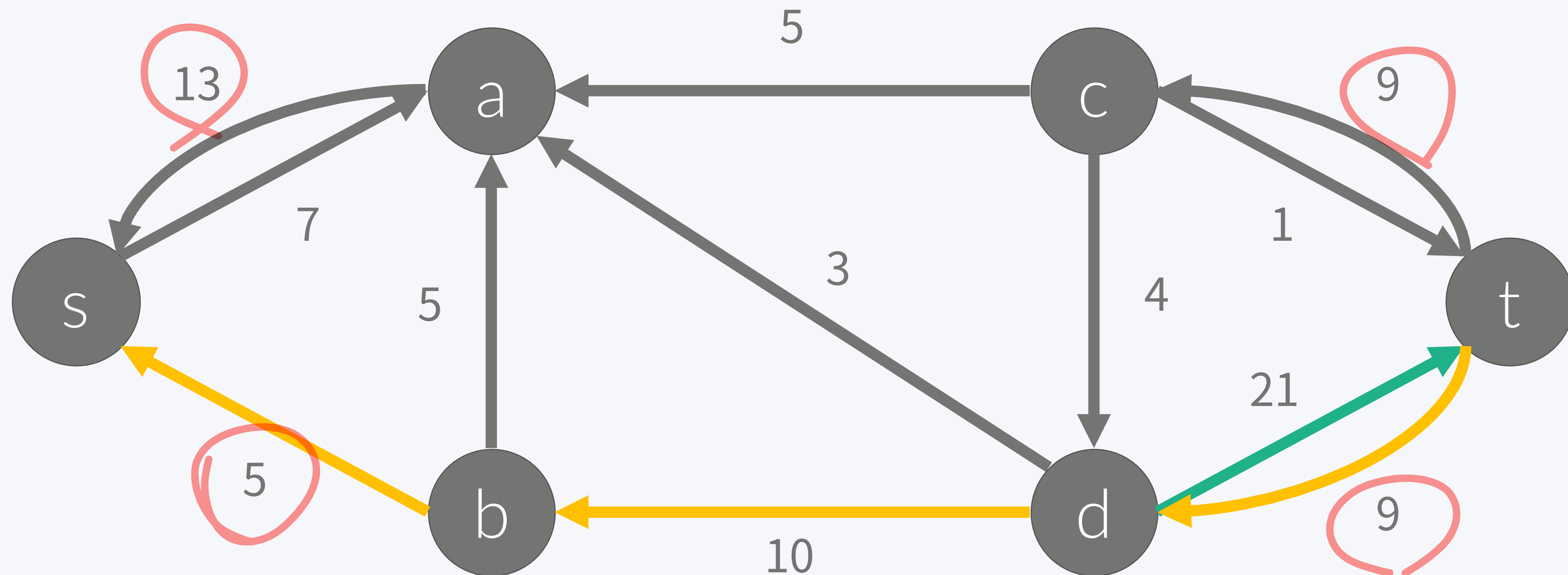


# Ford-Fulkerson

28

Maximum Flow

- Augmenting Path:  $s \rightarrow b \rightarrow d \rightarrow t$
- $m = 1$

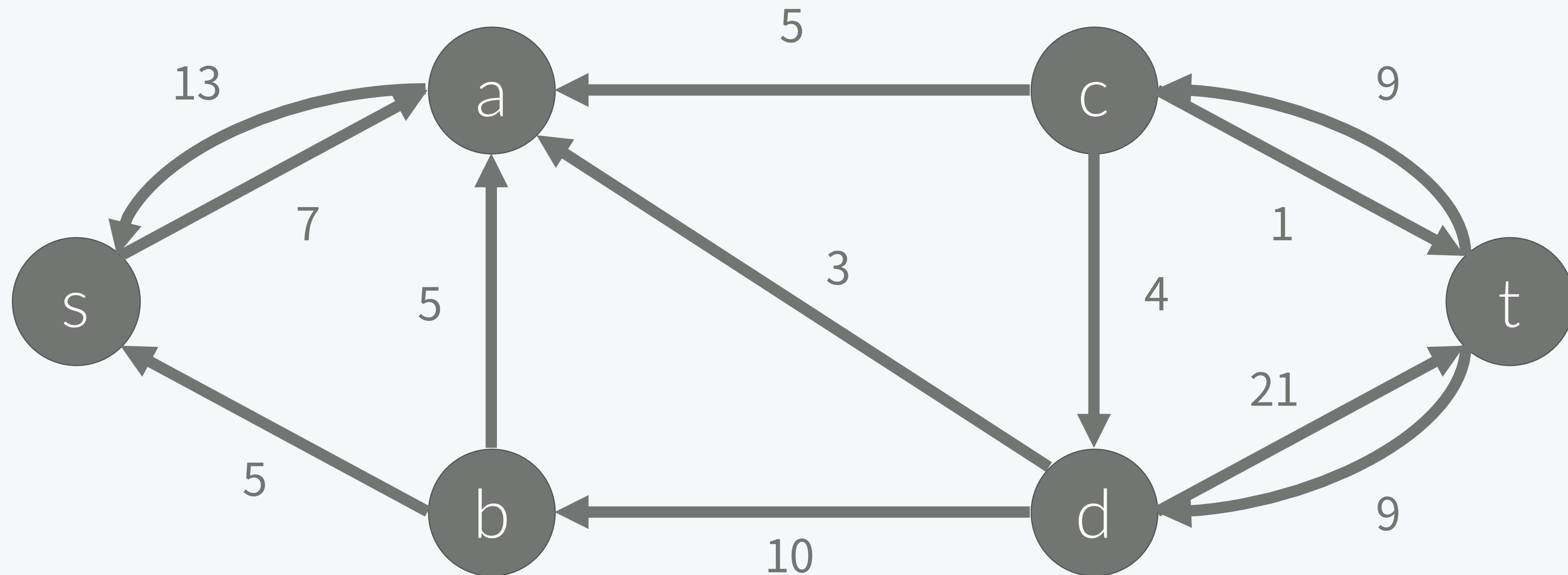


# Ford-Fulkerson

29

Maximum Flow

- 더 이상 Augmenting path가 없다



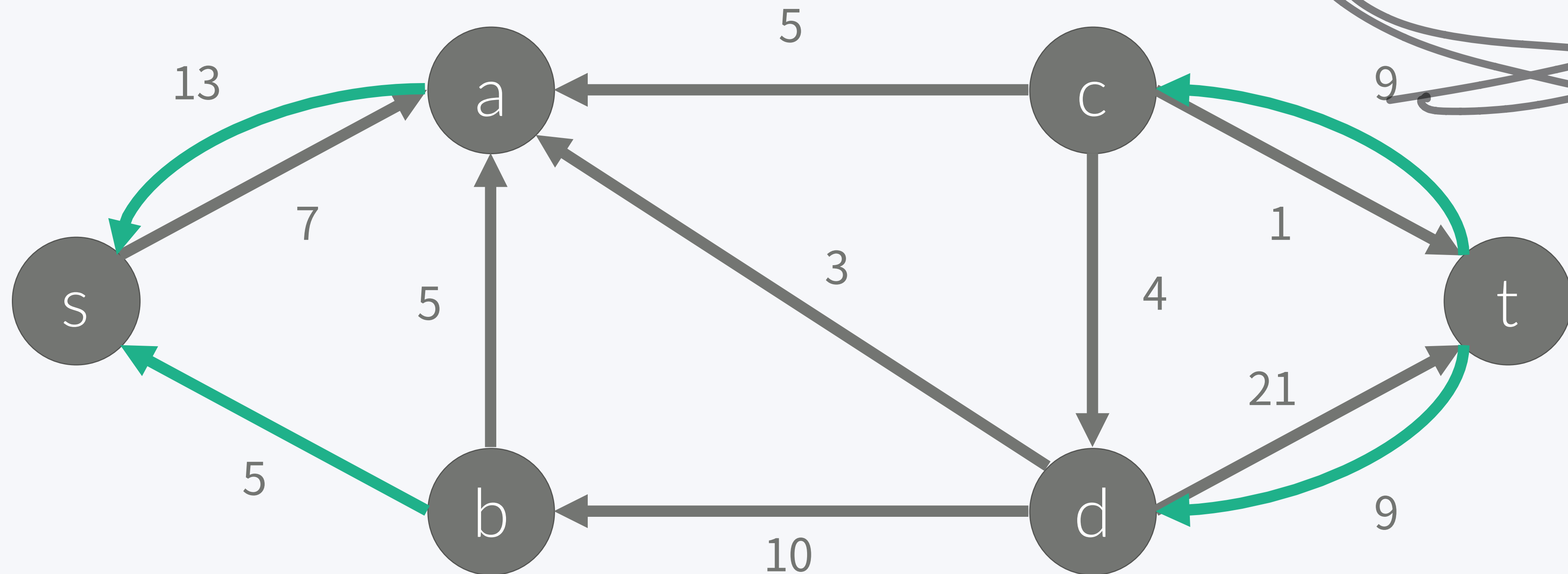
# Ford-Fulkerson

Maximum Flow

- Maximum Flow: 18

DFS

OLE



# Ford-Fulkerson

31

Maximum Flow

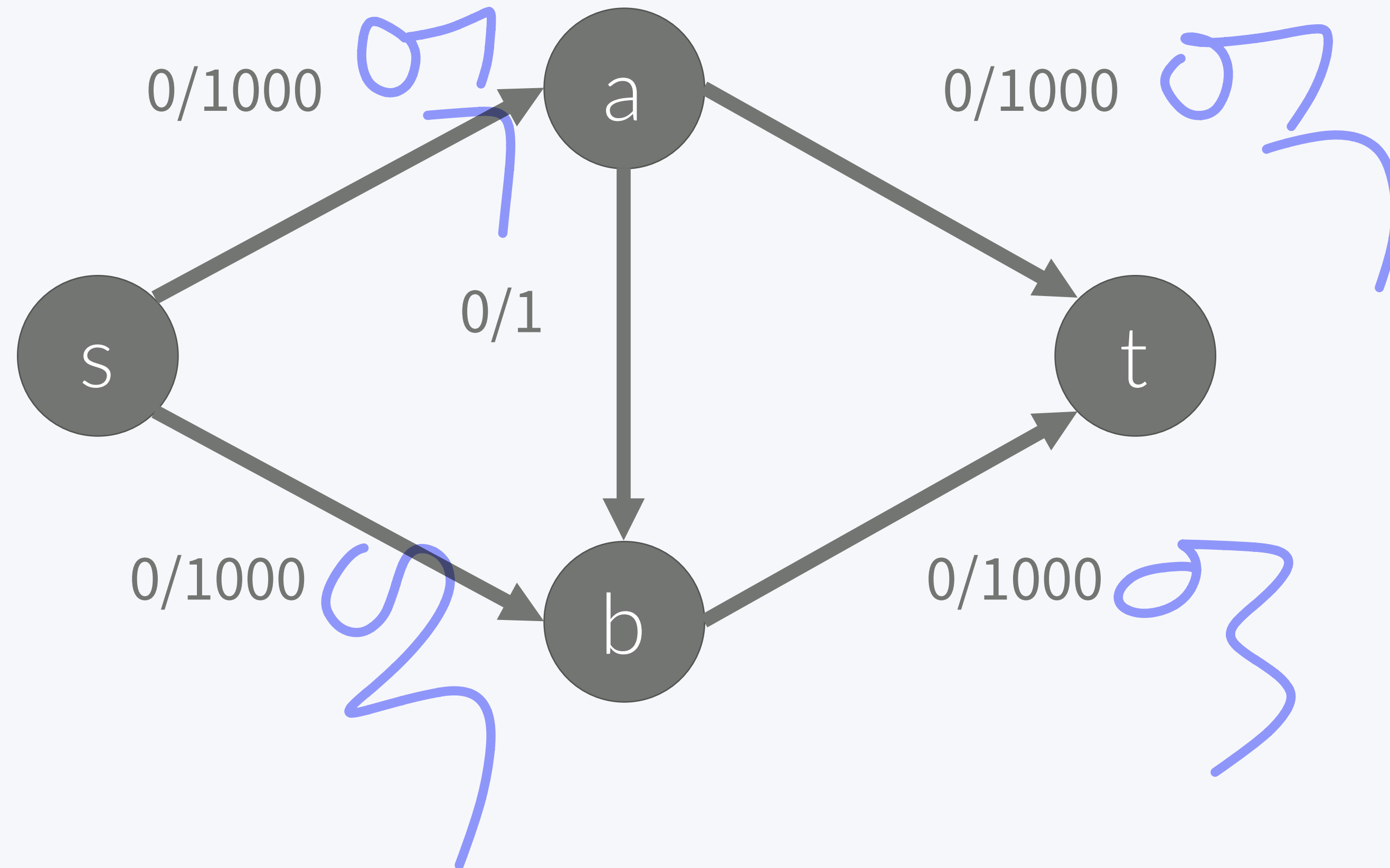
- 시간복잡도:  $O(Ef)$ 
  - $E$  : Edge 개수
  - $f$  : Maximum Flow

# Ford-Fulkerson

32

Maximum Flow

- 이런 경우



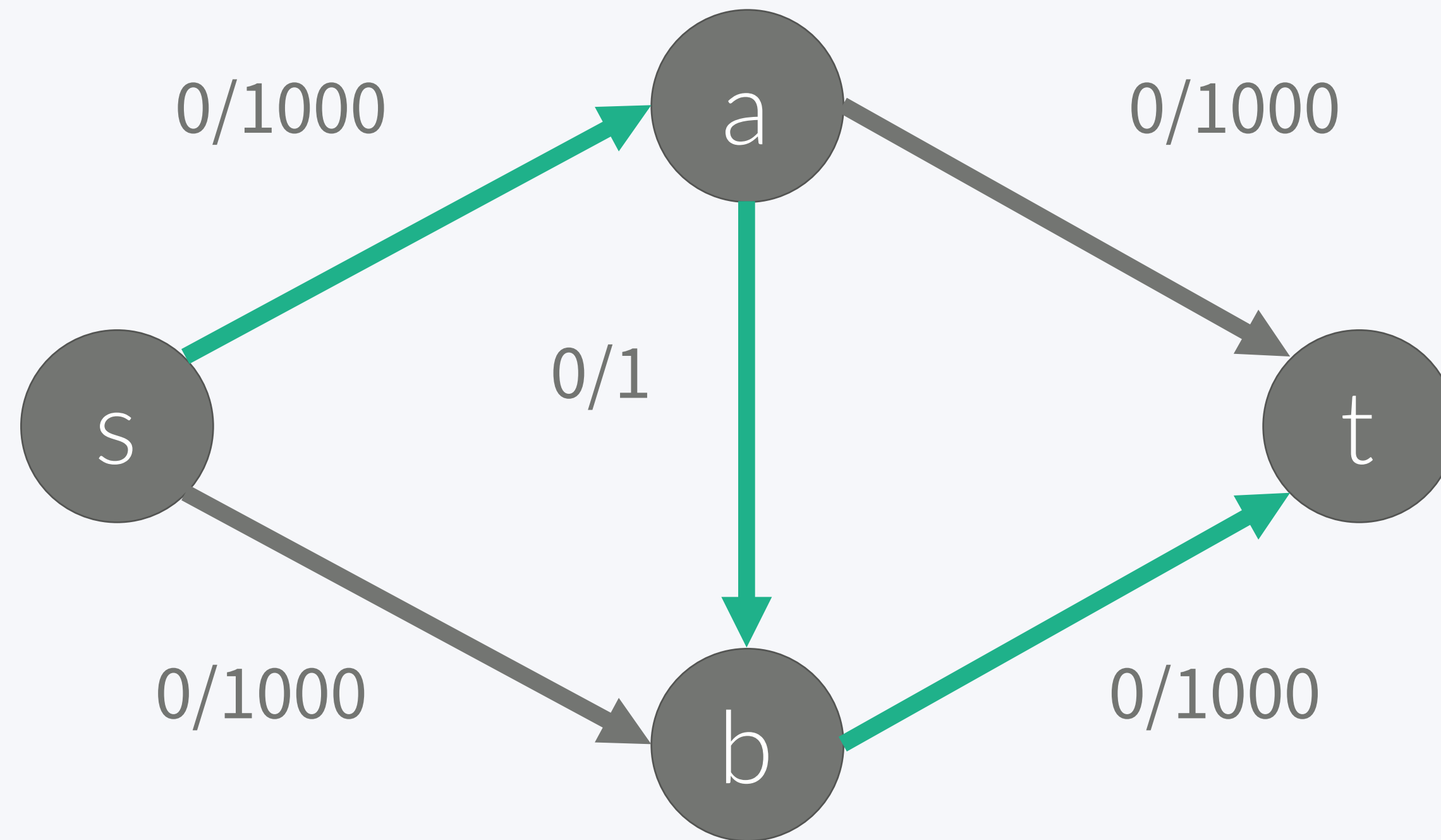


# Ford-Fulkerson

33

Maximum Flow

- 이런 경우

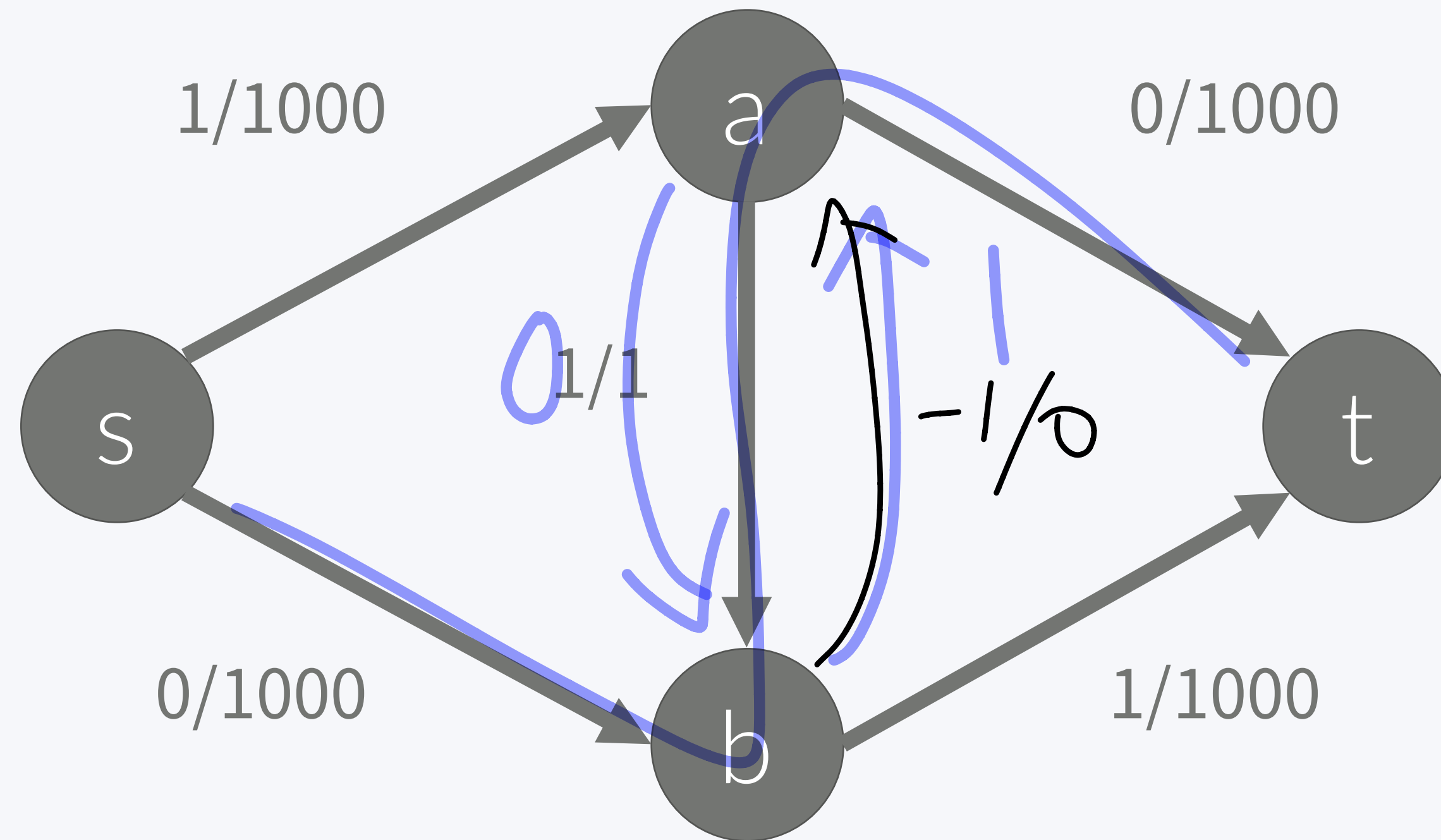


# Ford-Fulkerson

34

Maximum Flow

- 이런 경우

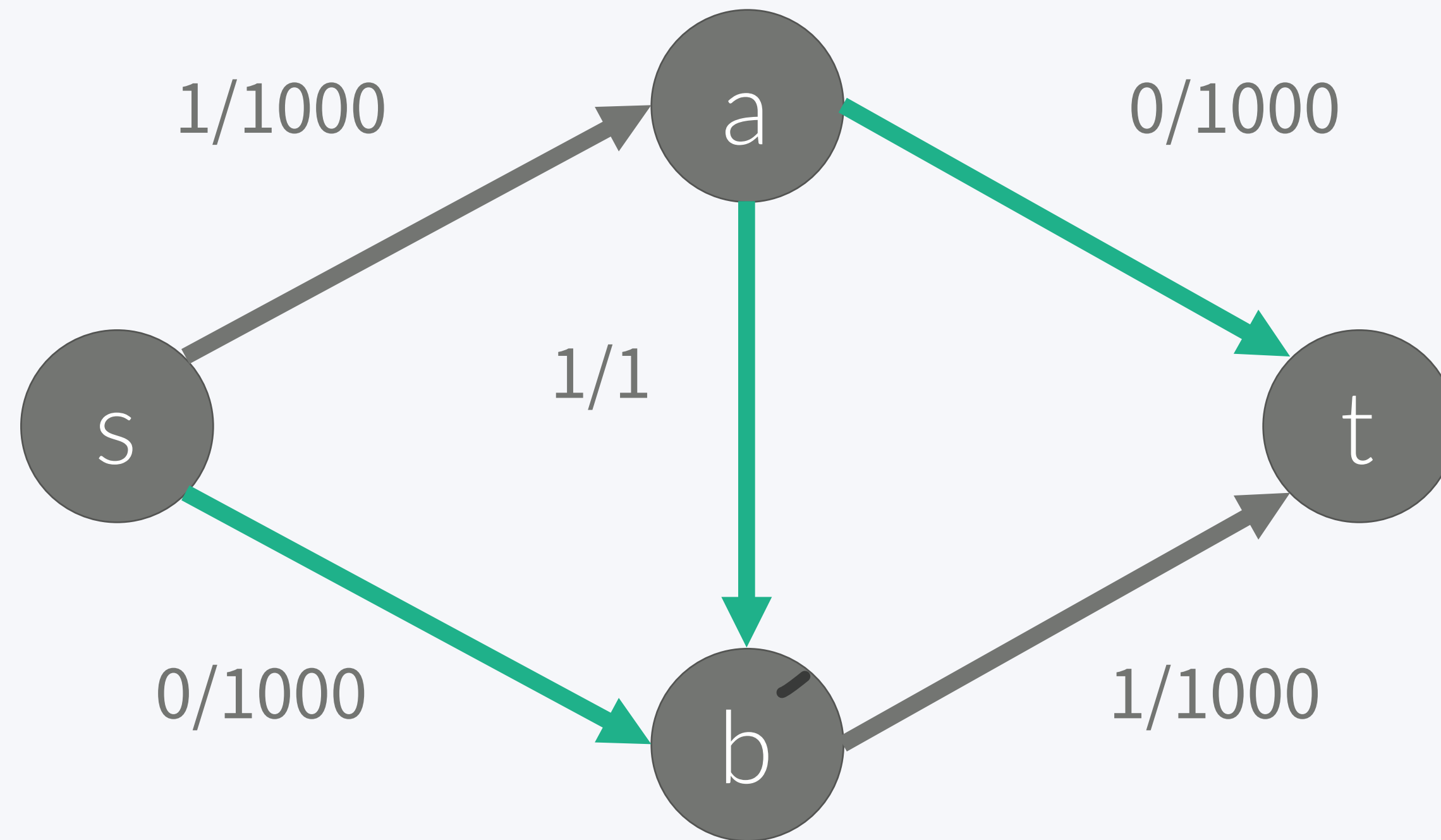


# Ford-Fulkerson

35

Maximum Flow

- 이런 경우

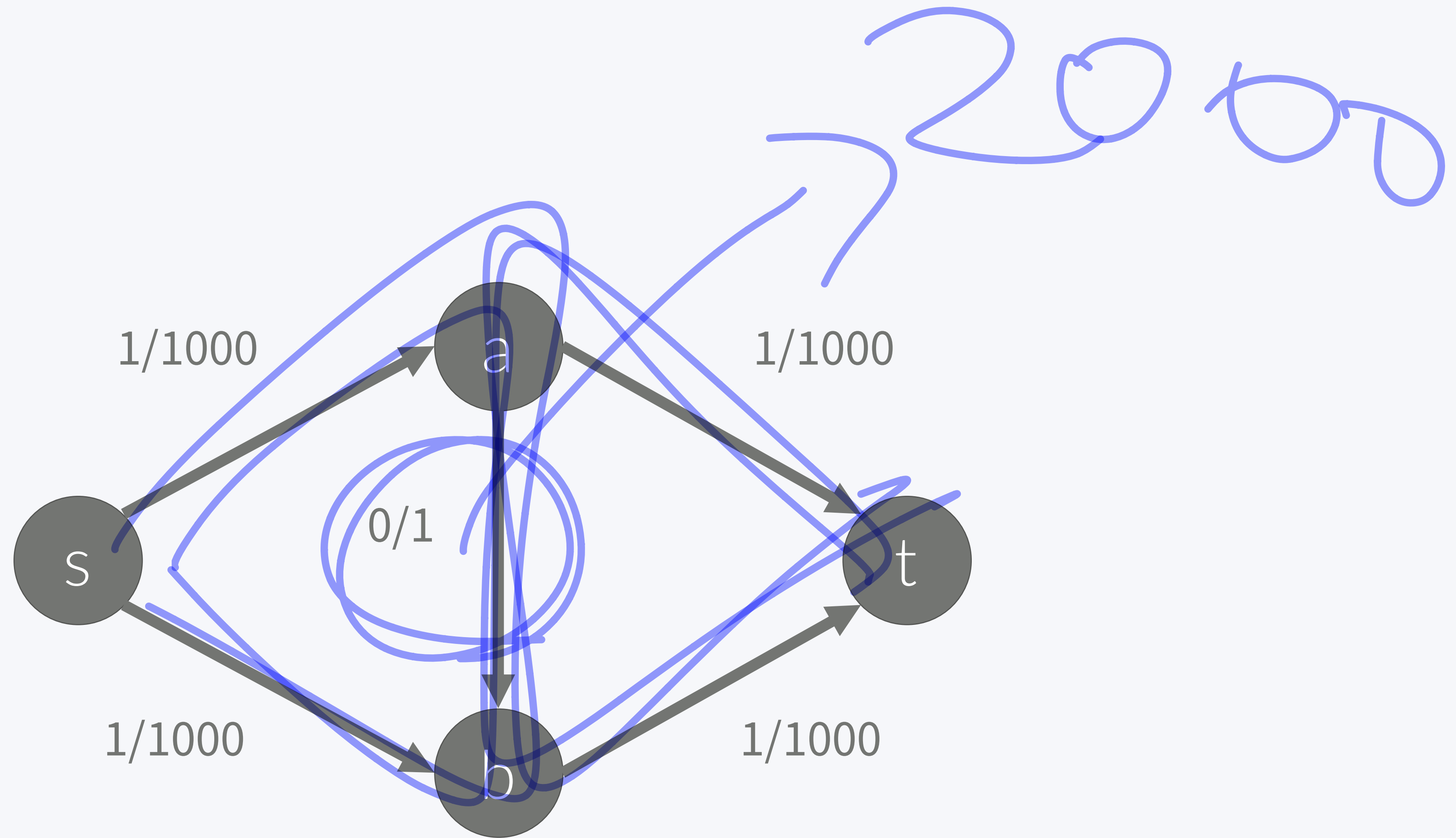


# Ford-Fulkerson

36

Maximum Flow

- 이런 경우



# Ford-Fulkerson

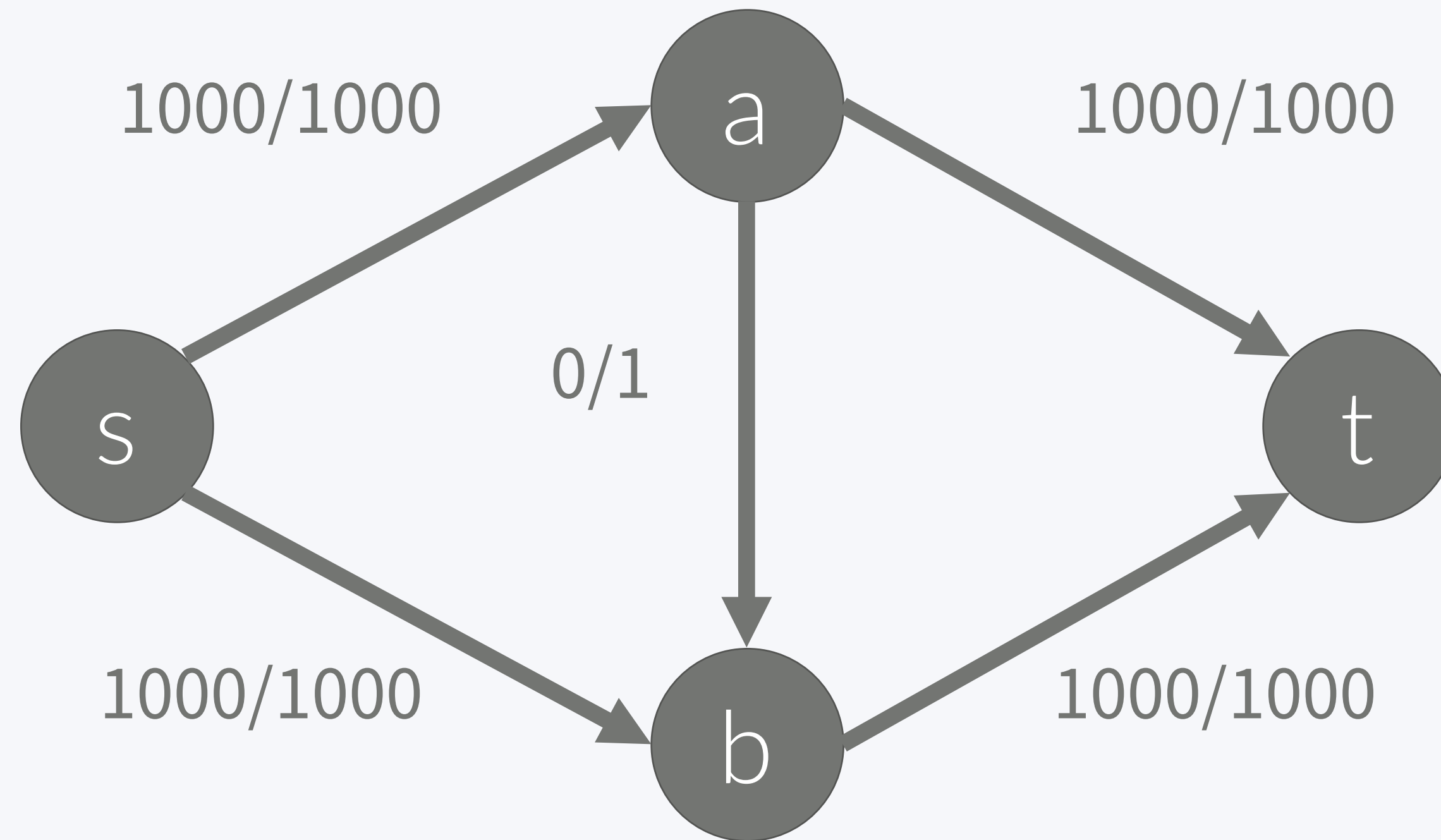
Maximum Flow

- 이 과정을 1998번 더 하면

# Ford-Fulkerson

Maximum Flow

- Flow network



OEFS

MF

Dinic:  $O(V^2E)$

Hopcroft-Karp  $O(E\sqrt{V})$

Ford Fulkerson  
 $O(Ef)$

---

Edmond Karp  
 $O(VE^2)$

---

# Edmond-Karp

# Edmond-Karp

## Maximum Flow

40

- 최대 유량을 구하는 알고리즘

1. Augmenting Path를 BFS를 이용해서 구한다.
2.  $m = \text{Augmenting Path 상에서의 최소값}$ 을 구한다
3.  $(u_i, u_{i+1})$  방향의 Residual Capacity에서  $m$ 을 뺀다
4.  $(u_{i+1}, u_i)$  방향의 Residual Capacity에  $m$ 을 더한다.
5. 위의 과정을 Augmenting Path를 못 구할때 까지 계속 한다

$$O(VE^2)$$

$$O(E) \times VE$$

모든 간선이 최소값이

최대  $\frac{V-1}{2}$  번만

선택 가능

$$O(VE^2)$$



# Total Flow

<https://www.acmicpc.net/problem/6086>

- 최대 유량을 구하는 문제

# Total Flow

<https://www.acmicpc.net/problem/6086>

- Matrix, DFS, Residual
  - <https://gist.github.com/Baekjoon/63c6f5c07cbd5251d0d6>
- Matrix, DFS, Capacity, Flow
  - <https://gist.github.com/Baekjoon/d92e03bd97277ebef159>
- Matrix, BFS, Residual
  - <https://gist.github.com/Baekjoon/44811c5633c388a16831>
- List, BFS, Residual
  - <https://gist.github.com/Baekjoon/d67c59ead5a81d1e03d4>
- List, DFS, Residual
  - <https://gist.github.com/Baekjoon/a1acd6844d9a8ef068e6>

# Total Flow

<https://www.acmicpc.net/problem/6086>

- struct NetworkFlow (DFS)
  - <https://gist.github.com/Baekjoon/4245b1376bdda0f5bc04>
- struct NetworkFlow (BFS)
  - <https://gist.github.com/Baekjoon/6aa876d32b85c709d0e3>

# 이분 매칭

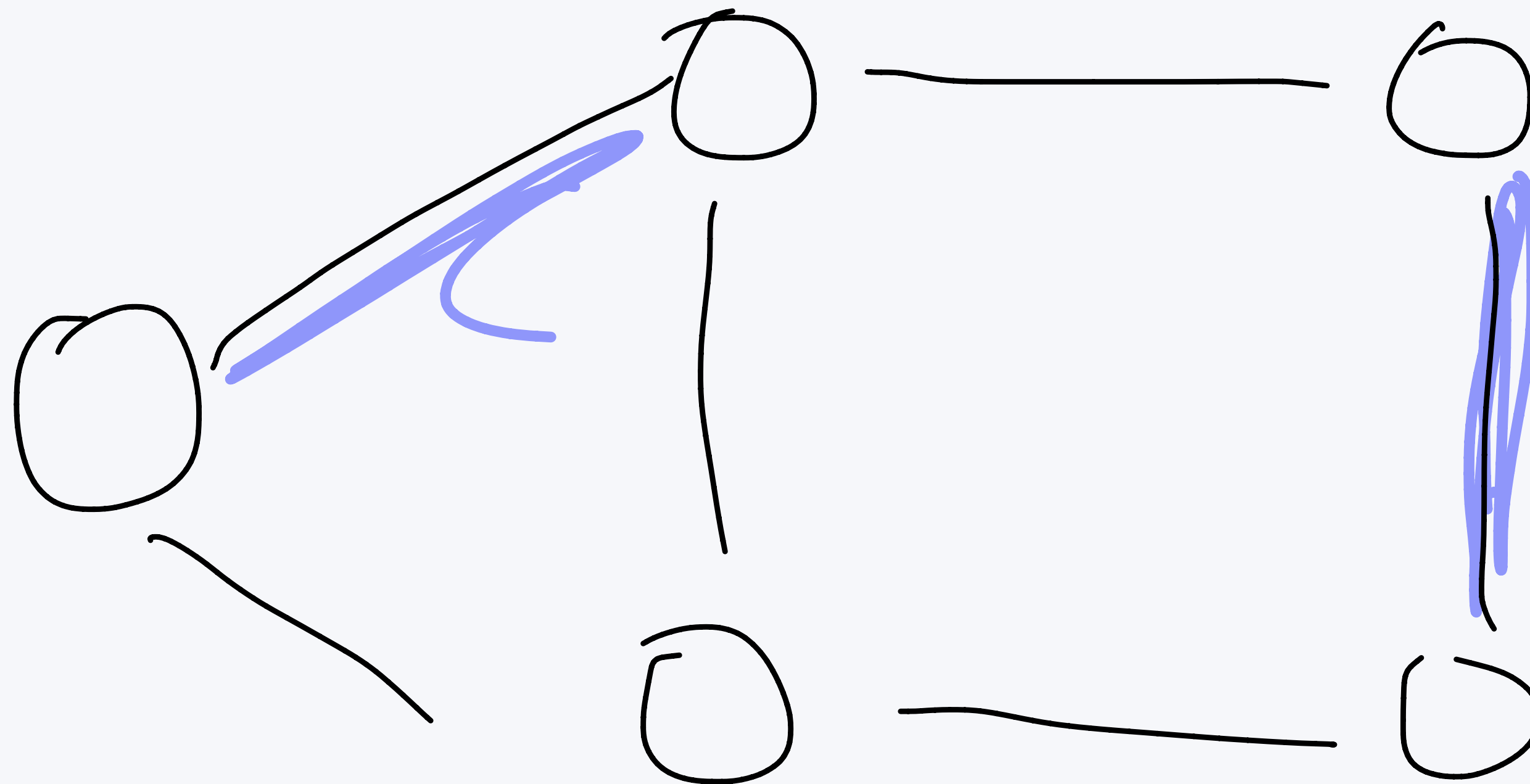


# 이분 매칭

## Bipartite Matching

45

- Matching: 그래프 G에서 적절히 간선을 선택했을 때, 각각의 간선이 공통된 vertex를 공유하지 않음
- Maximum Matching: Edge의 최대값



# 이분 매칭

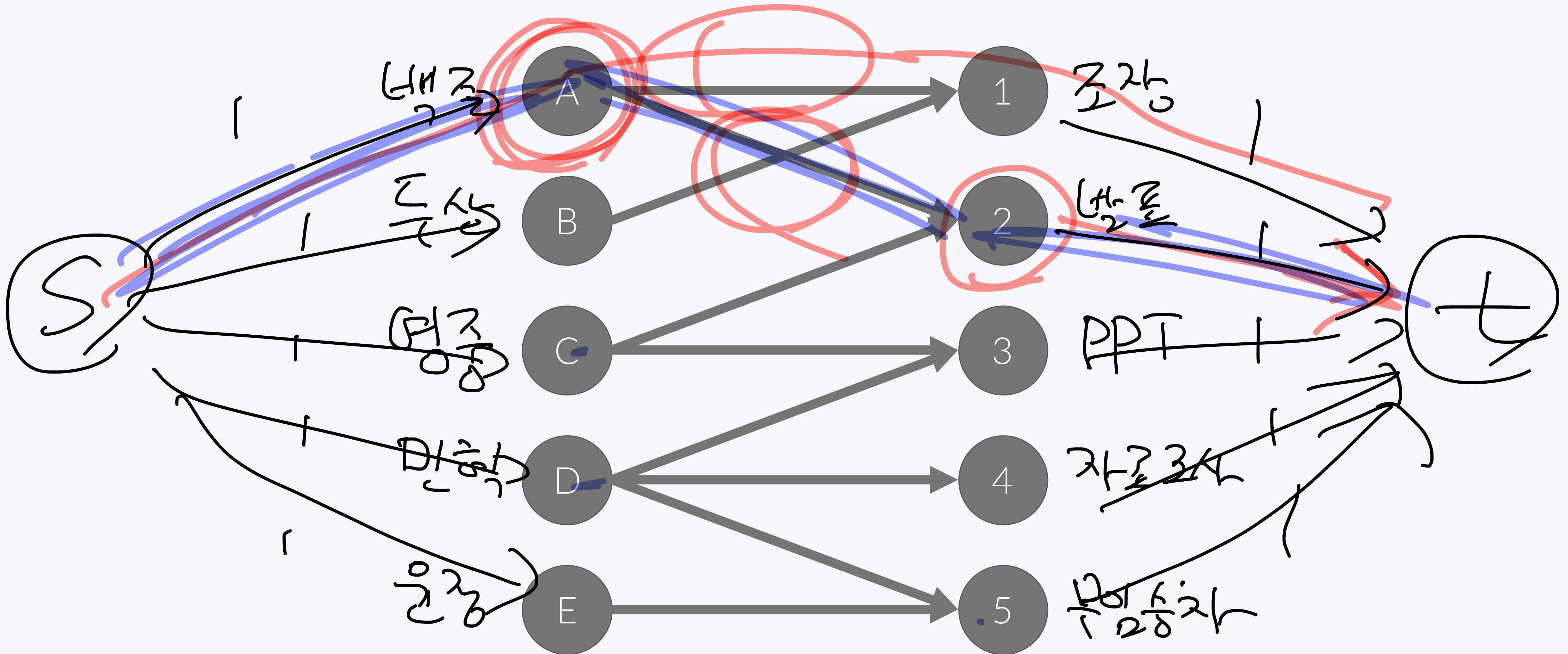
Bipartite Matching

MCMF

사람들을 1지 매칭  
모든 매칭들 (사람들)

46

- 사람: A~E, 일: 1~5, 각 사람이 할 수 있는 일을 Edge로 연결

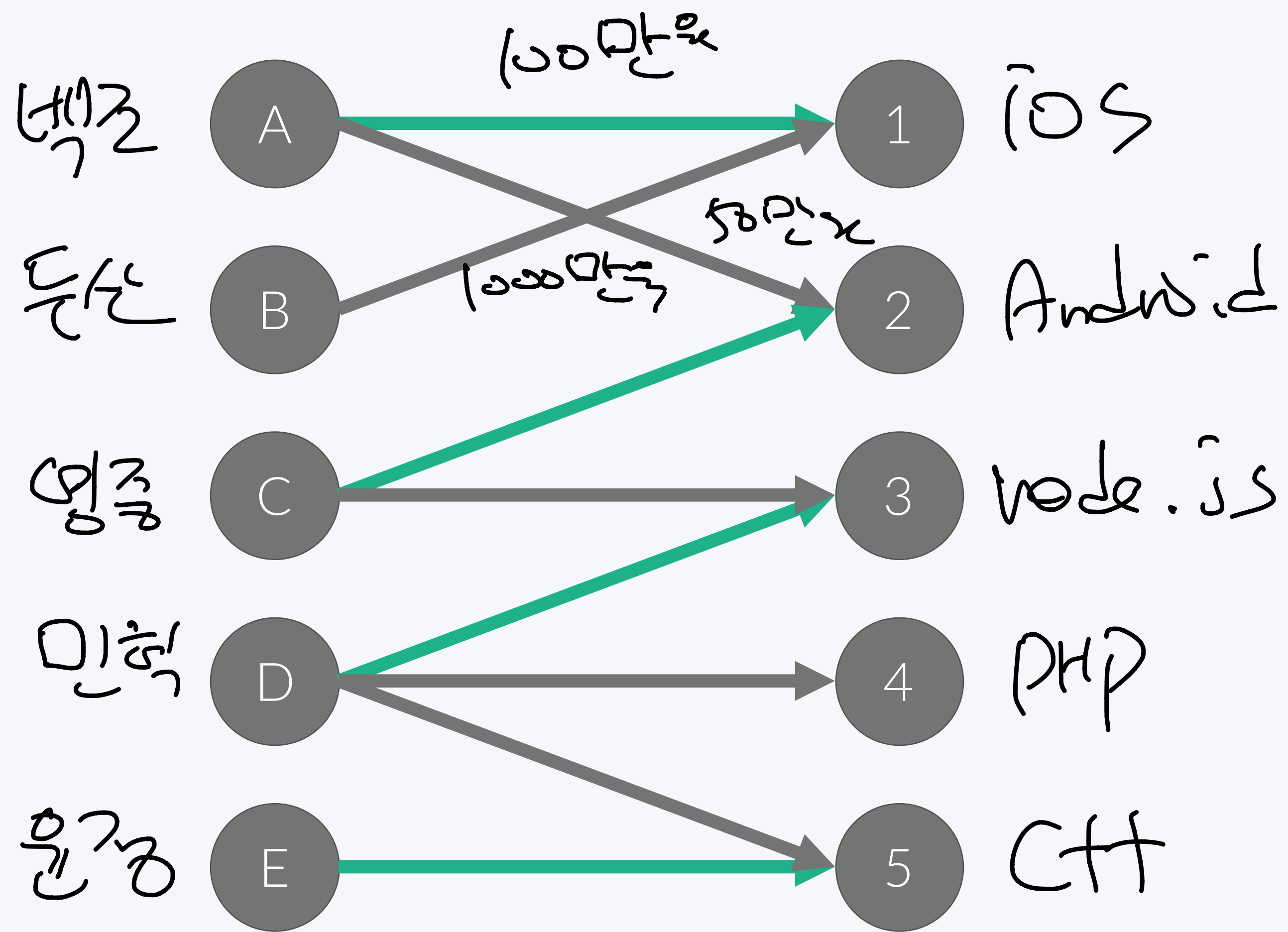
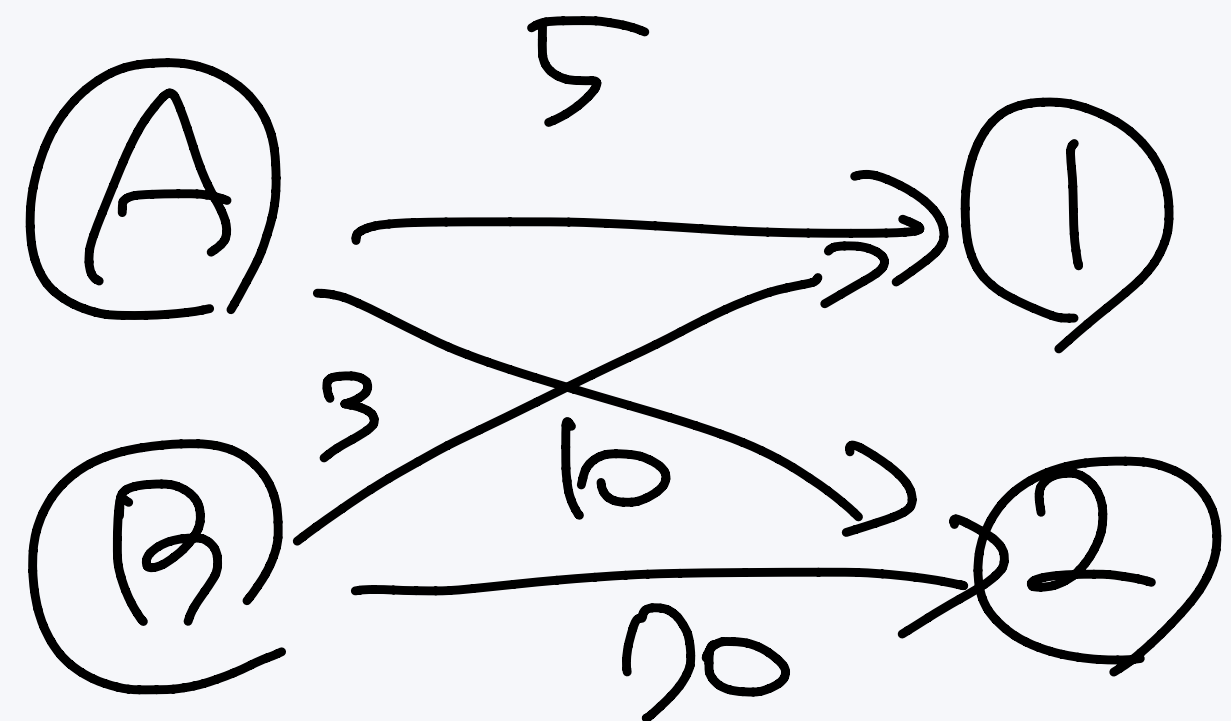


# 이분 매칭

Bipartite Matching

- 사람: A~E, 일: 1~5, 각 사람이 할 수 있는 일을 Edge로 연결

승현 사장



2.

A-1	5	125
B-2	20	125

---

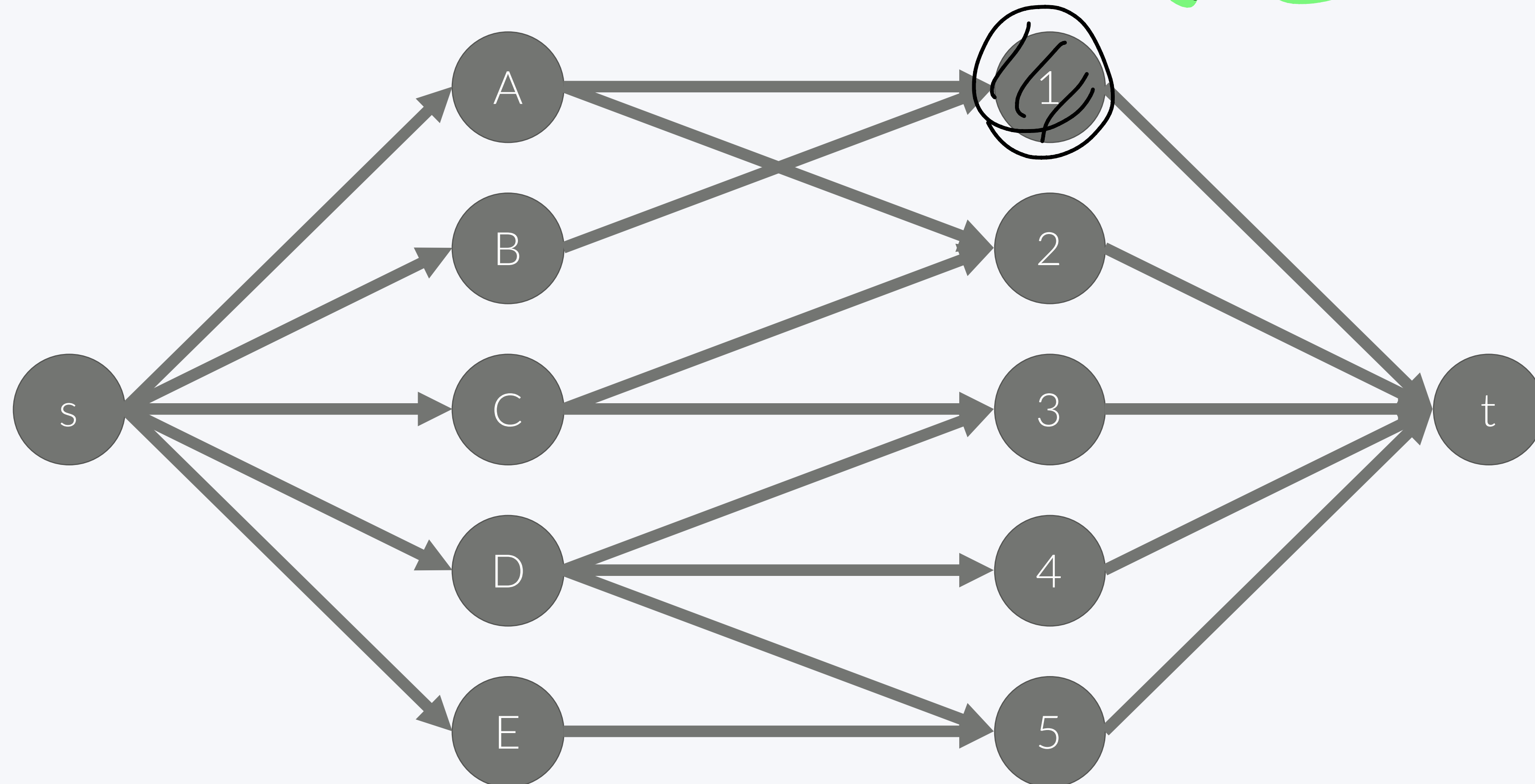
A-2    10    13  
B-1    3    13

# 이분 매칭

Bipartite Matching

- 네트워크 플로우 문제로 바뀌서 풀 수 있다. (capacity = 1)

$E \rightarrow V$   
 $\sqrt{VE^2}$



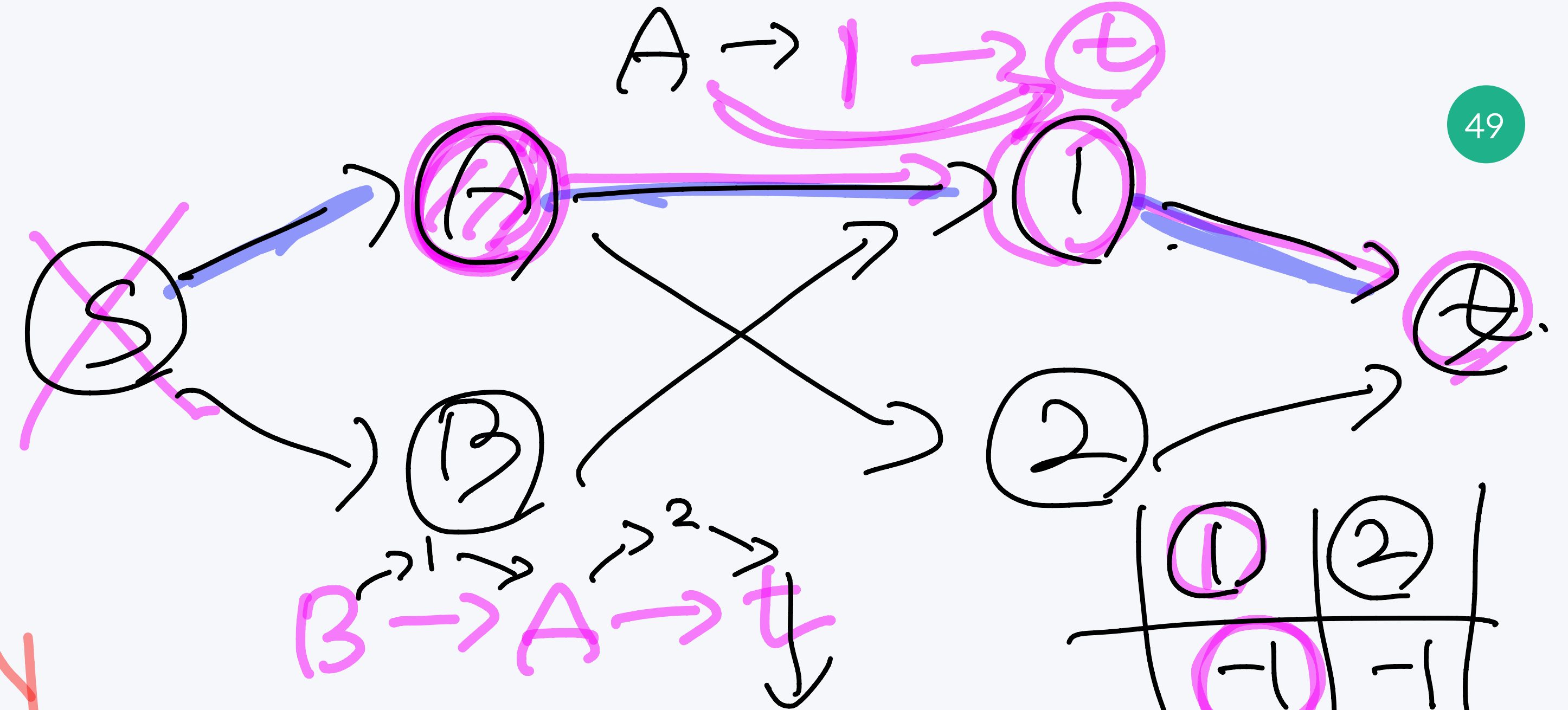


# 이분 매칭

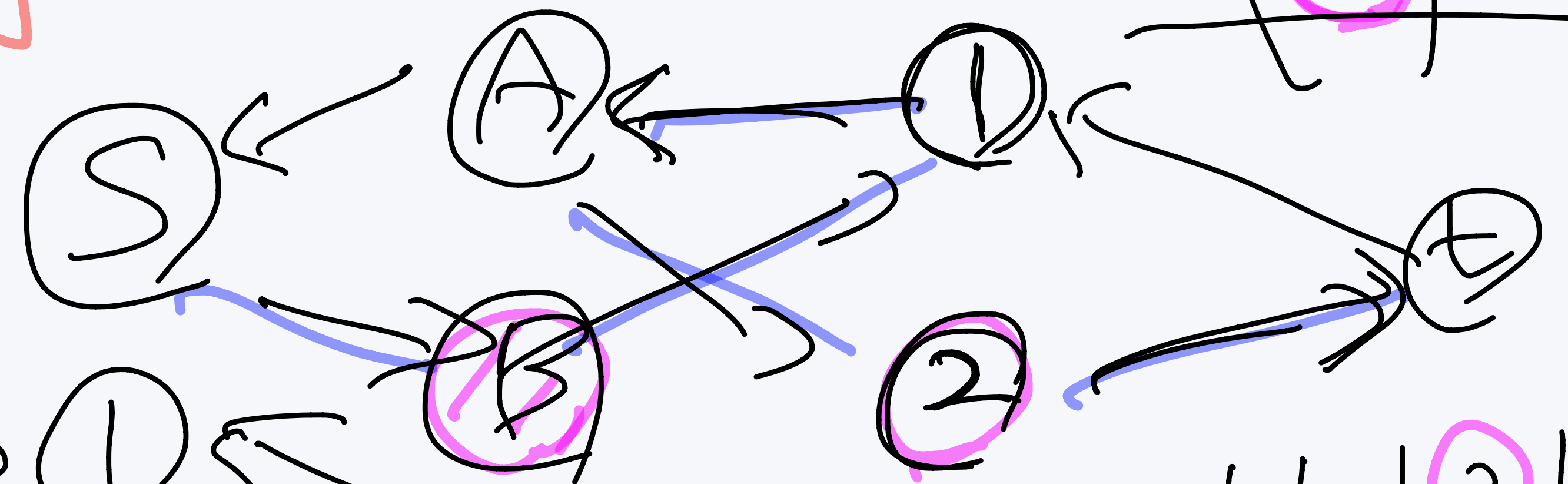
Bipartite Matching

- 이분 그래프에서 (A -> B)
- 소스(s)와 싱크(t)를 추가하고
- s -> A로 edge를 연결
- B -> t로 edge를 연결
- 모든 capacity = 1
- Maximum Flow가 답이 된다

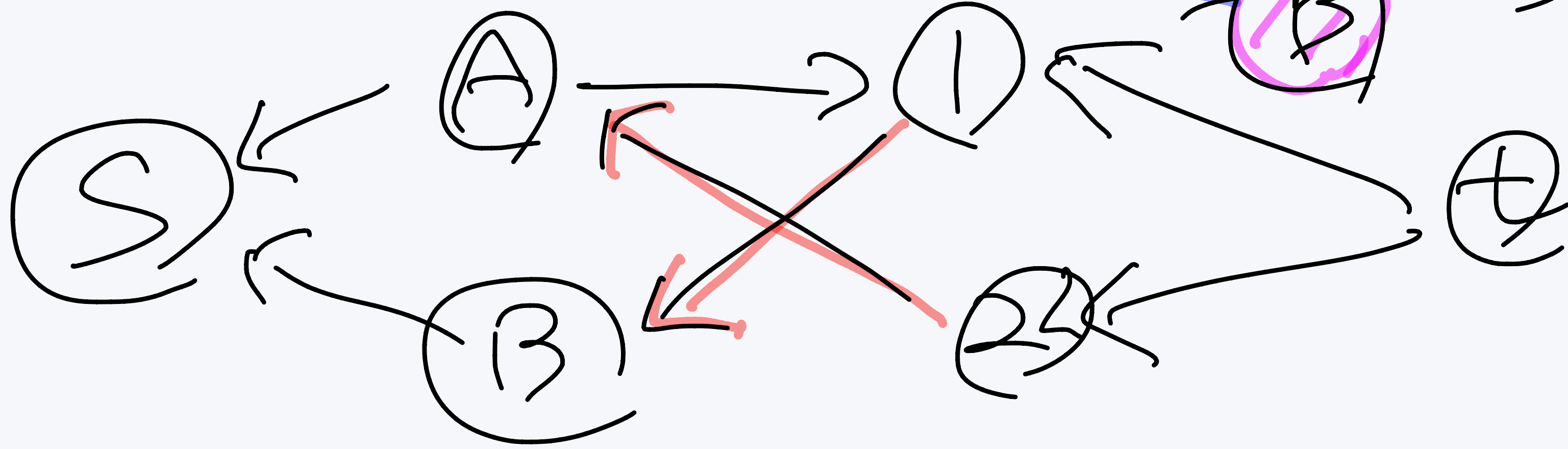
A-2  
B-1



①	②
-1	-1



1	②
①	-1



# 이분 매칭

## Bipartite Matching

- A에 속해 있는 노드는 최대 1개의 나가는 edge를 선택
- B에 속해 있는 노드는 최대 1개의 들어오는 edge를 선택



# 열혈강호

<https://www.acmicpc.net/problem/11375>

- 사람  $N$ 명 일  $M$ 개
- 각 직원은 한 개의 일만 할 수 있고, 각각의 일을 담당하는 사람은 1명이어야 한다.
- 할 수 있는 일의 최대 개수

# 열혈강호

<https://www.acmicpc.net/problem/11375>

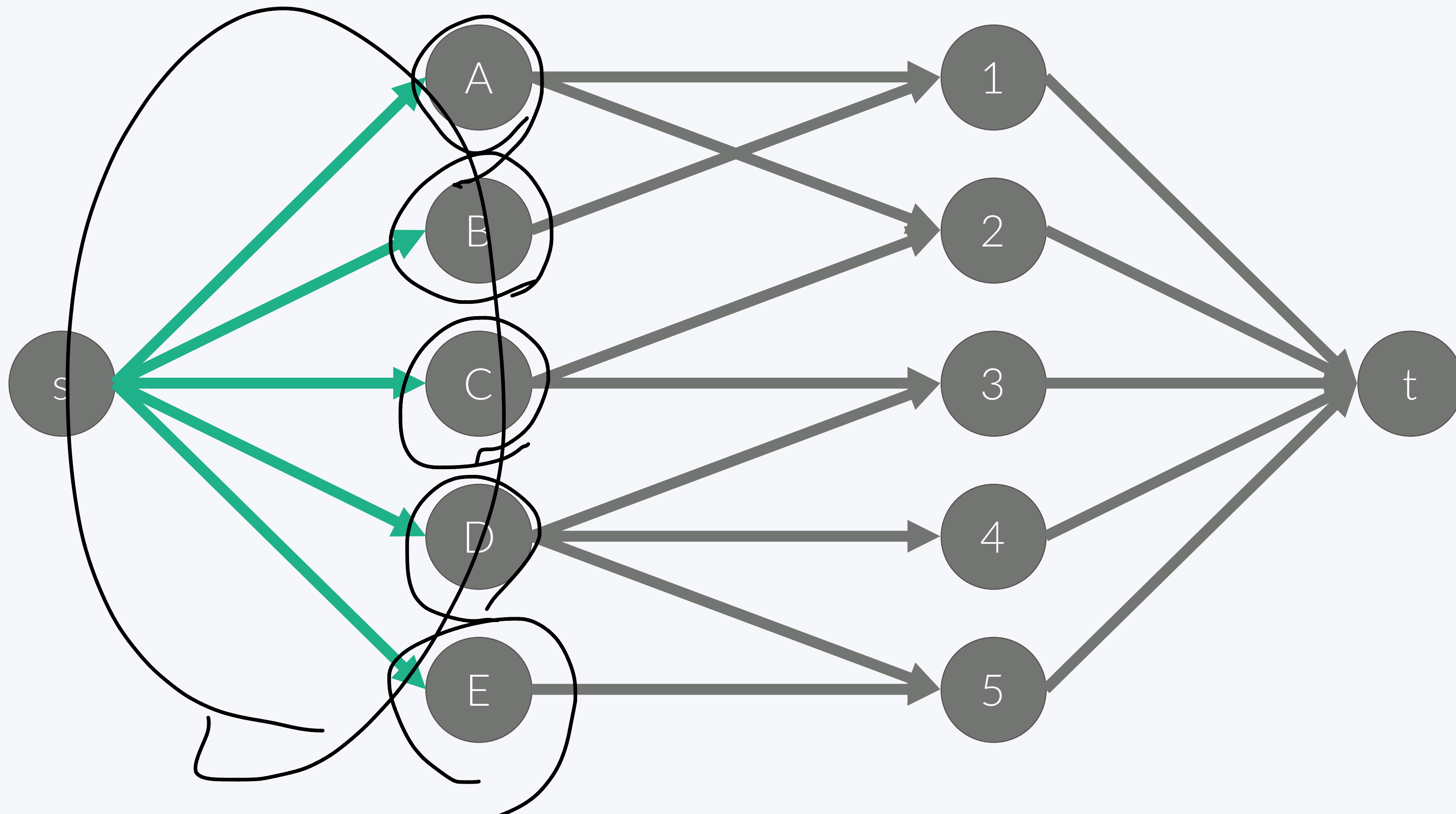
- Ford-Fulkerson: <https://gist.github.com/Baekjoon/4b6f576da76e9a035405>
- Ford-Fulkerson 다른 방식 구현: <https://gist.github.com/Baekjoon/7dc06797bddc4664f599>

# 이분 매칭

## Bipartite Matching

53

- 항상 1 또는 0이기 때문에, edge를 만들지 않아도 된다



# 이분 매칭

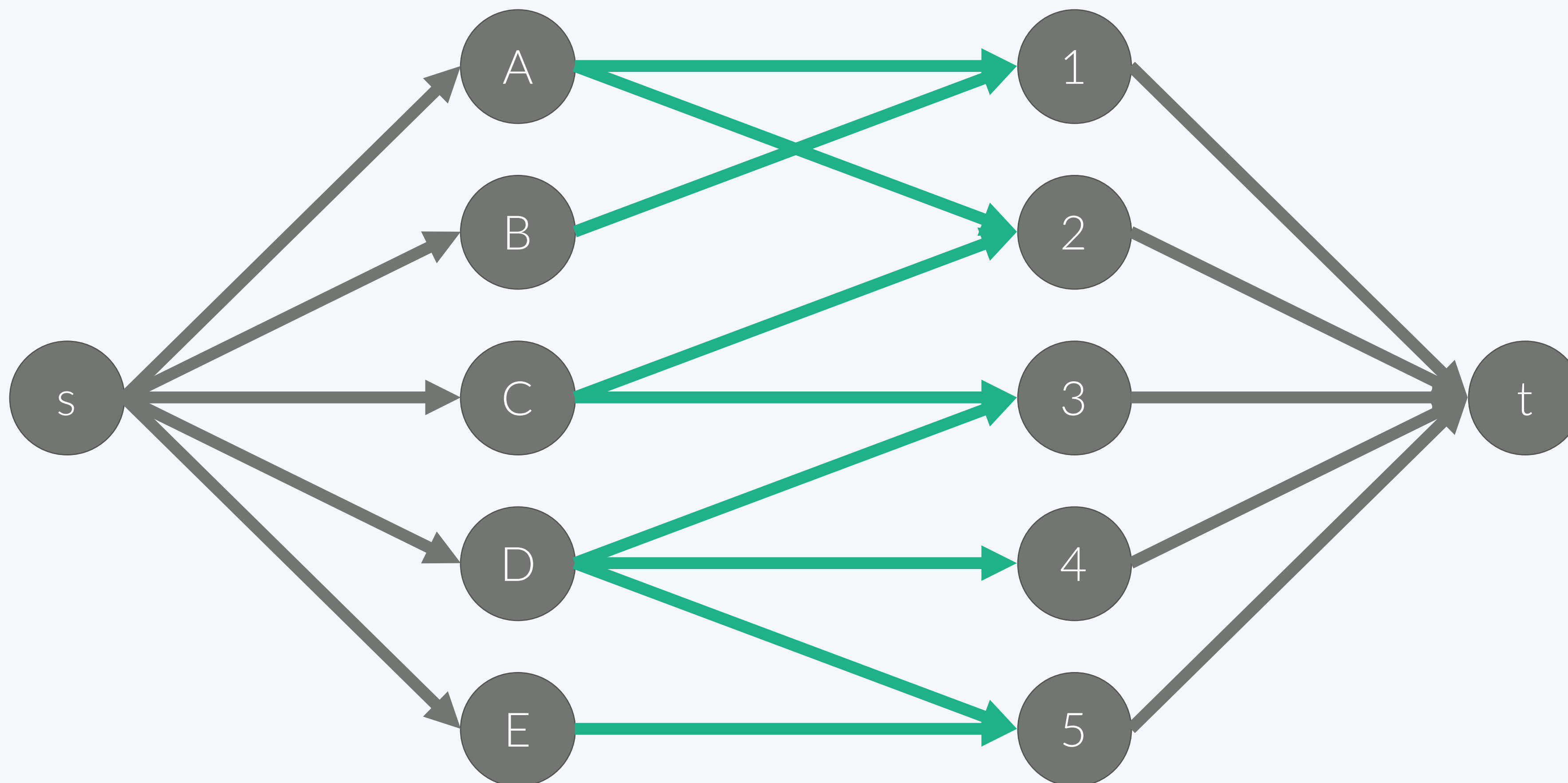
## Bipartite Matching

```
int flow() {  
    int ans = 0;  
    for (int i=0; i<n; i++) {  
        fill(check.begin(), check.end(), false);  
        if (dfs(i)) {  
            ans += 1;  
        }  
    }  
    return ans;  
}
```

# 이분 매칭

# Bipartite Matching

- 그대로 유지해야 한다

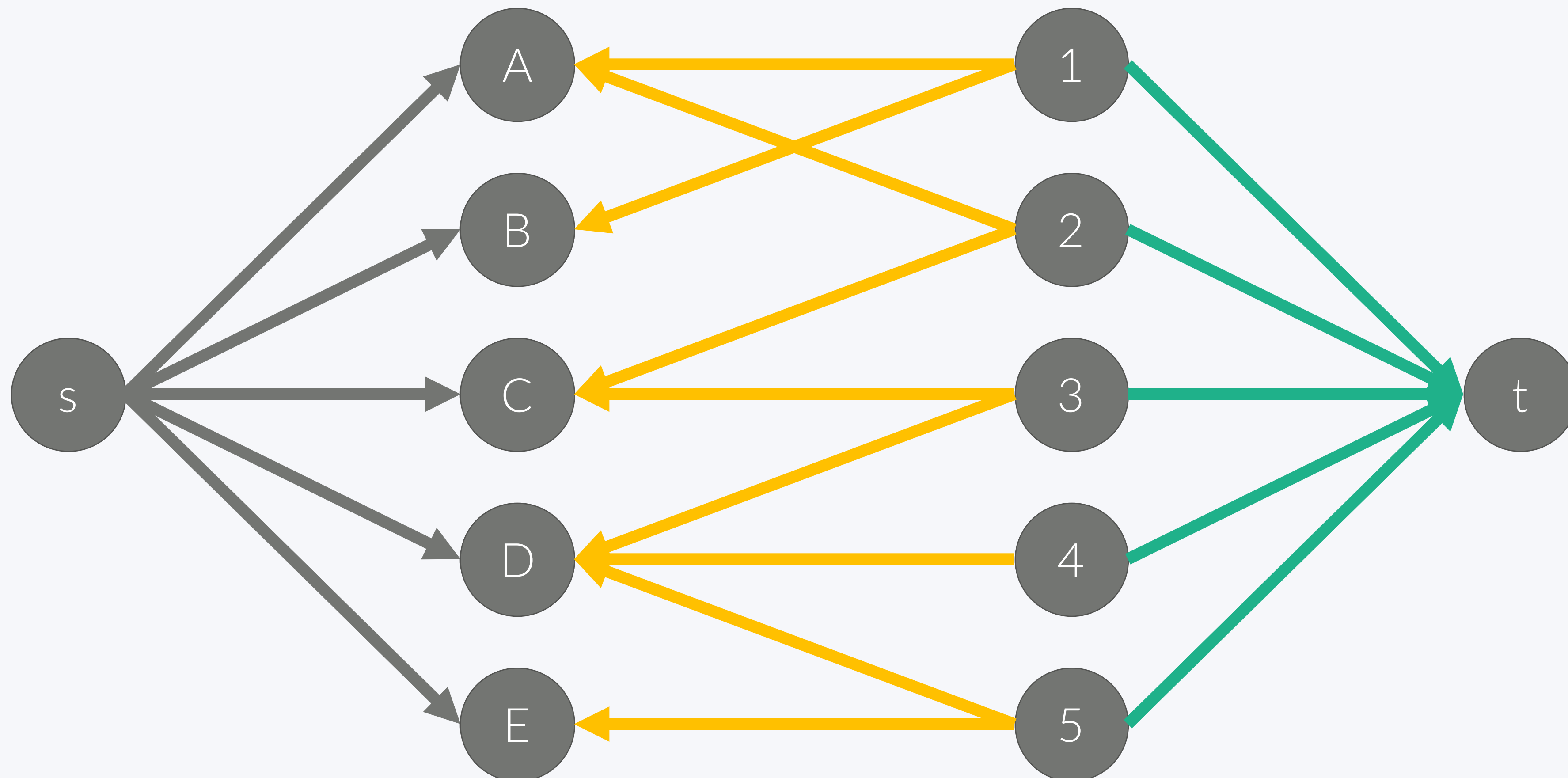


# 이분 매칭

Bipartite Matching

56

- 오른쪽은 항상 Sink로 가거나 아니면 다시 왼쪽으로 가게된다



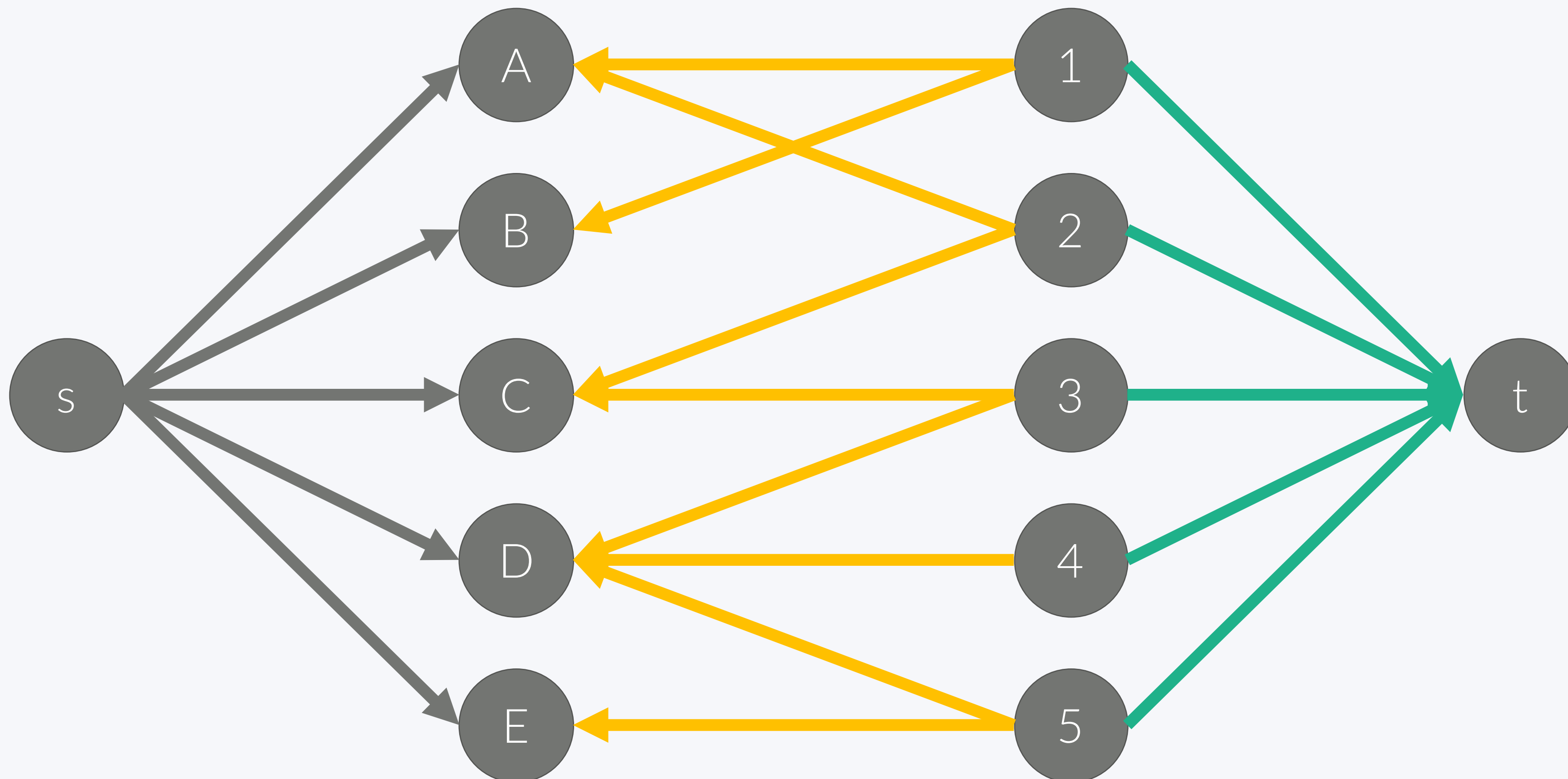


# 이분 매칭

## Bipartite Matching

57

- Sink로 가는 경우에는 -1, 왼쪽으로 가는 경우에는 그 정점 번호



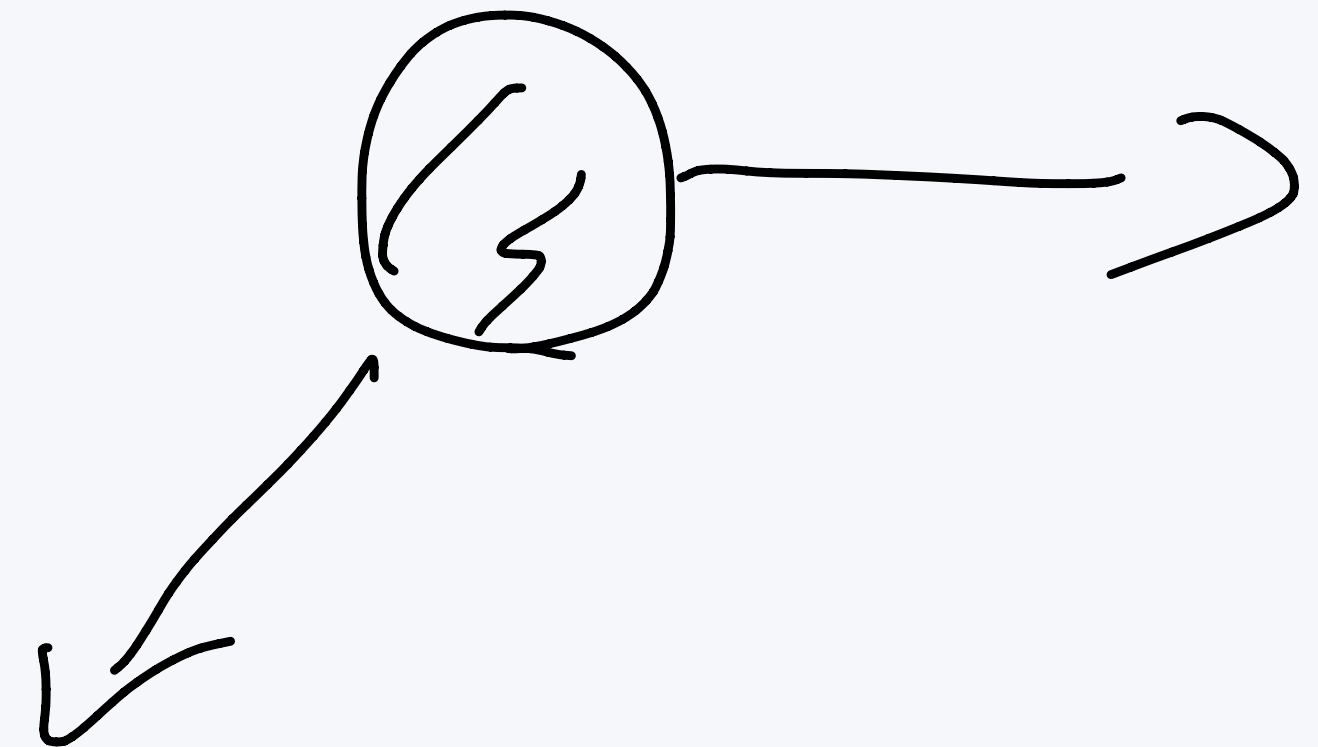
# 이분 매칭

Bipartite Matching

58

```
bool dfs(int x) {  
    if (x == -1) return true;  
    for (int next : graph[x]) {  
        if (check[next]) continue;  
        check[next] = true;  
        if (dfs(pred[next])) {  
            pred[next] = x;  
            return true;  
        }  
    }  
    return false;  
}
```

pred[i]  
S<sub>2</sub> ⊆ T     $\frac{L}{L+1}$

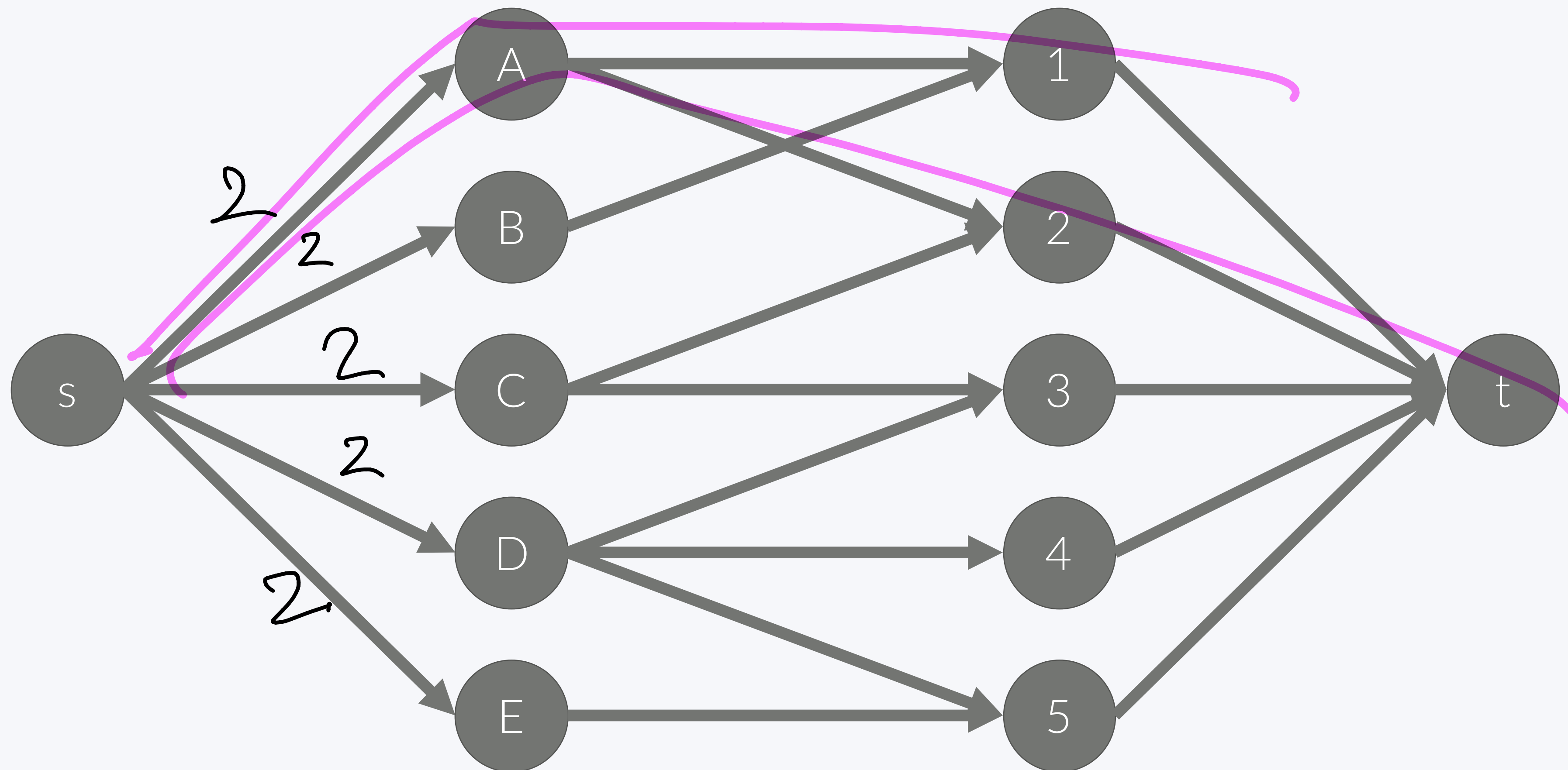


# 열혈강호

<https://www.acmicpc.net/problem/11375>

사슴 먹물 최대 2 개  
3개 먹물 (사슴)

- <https://gist.github.com/Baekjoon/0b8072965d77305b9beb>

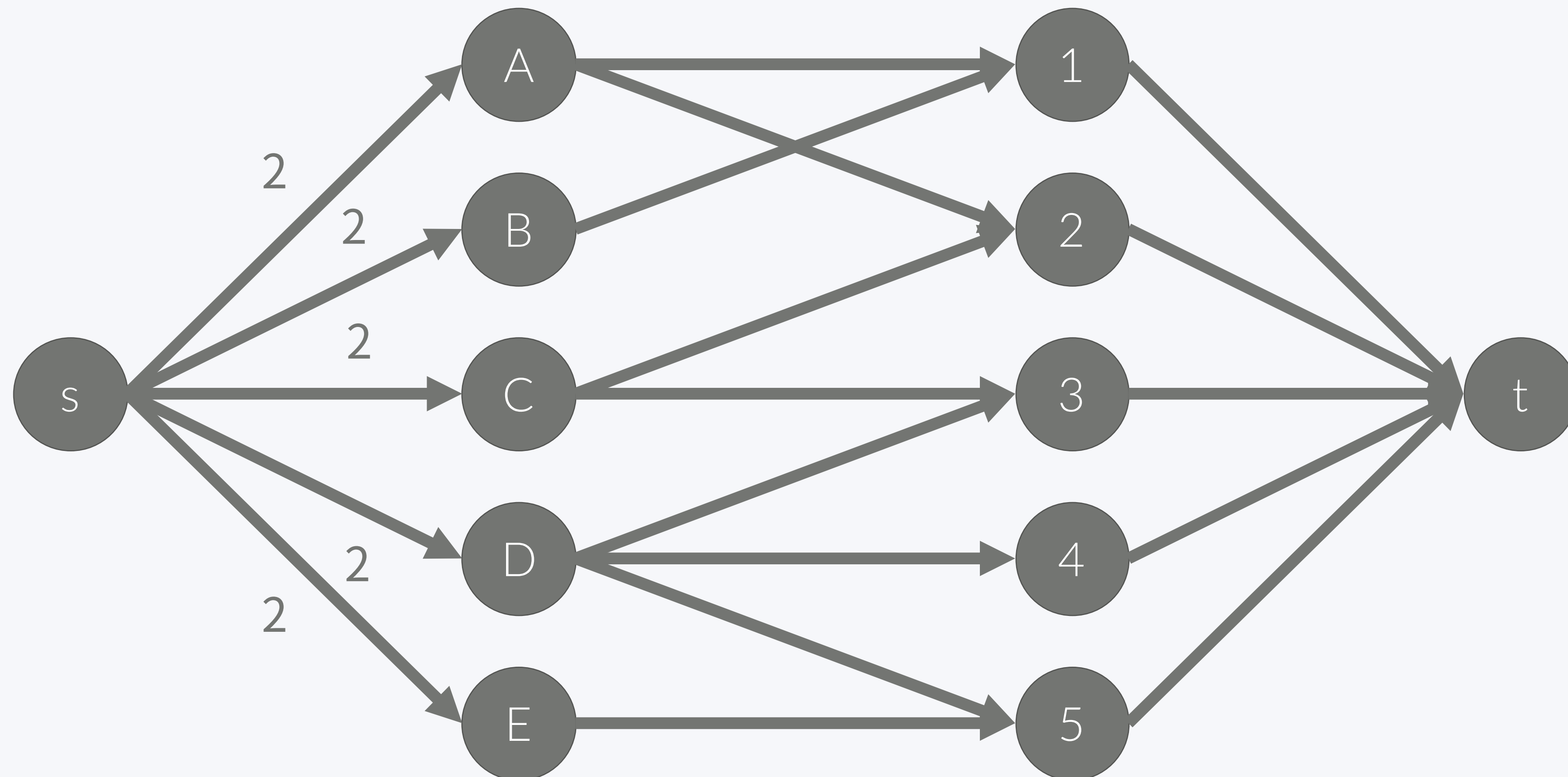


# 열혈강호 2

60

<https://www.acmicpc.net/problem/11376>

- <https://gist.github.com/Baekjoon/01163c4a0b051fb476cd>

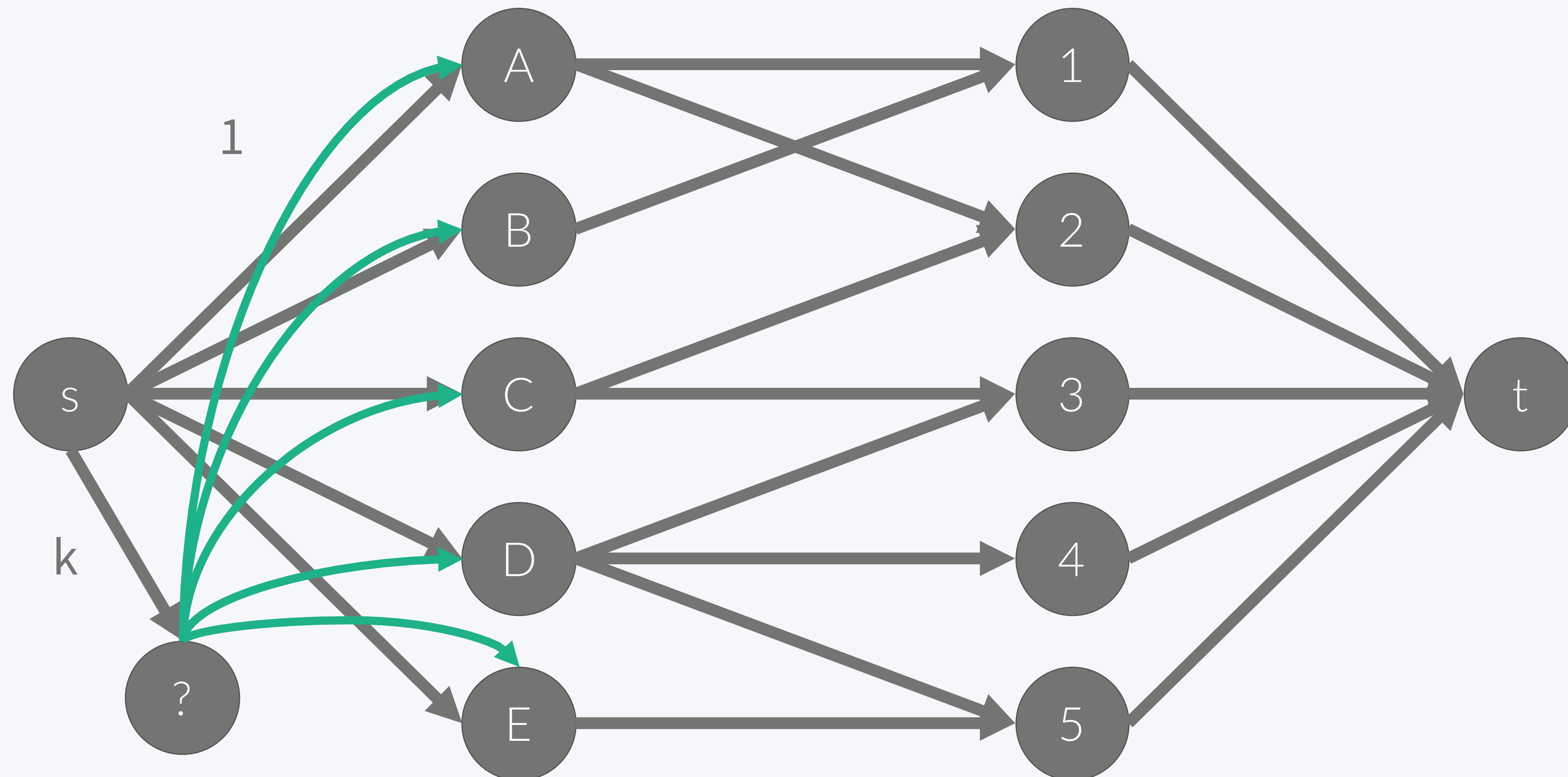


# 열혈강호 3

61

<https://www.acmicpc.net/problem/11377>

- <https://gist.github.com/Baekjoon/8961653081d03e51fd99>

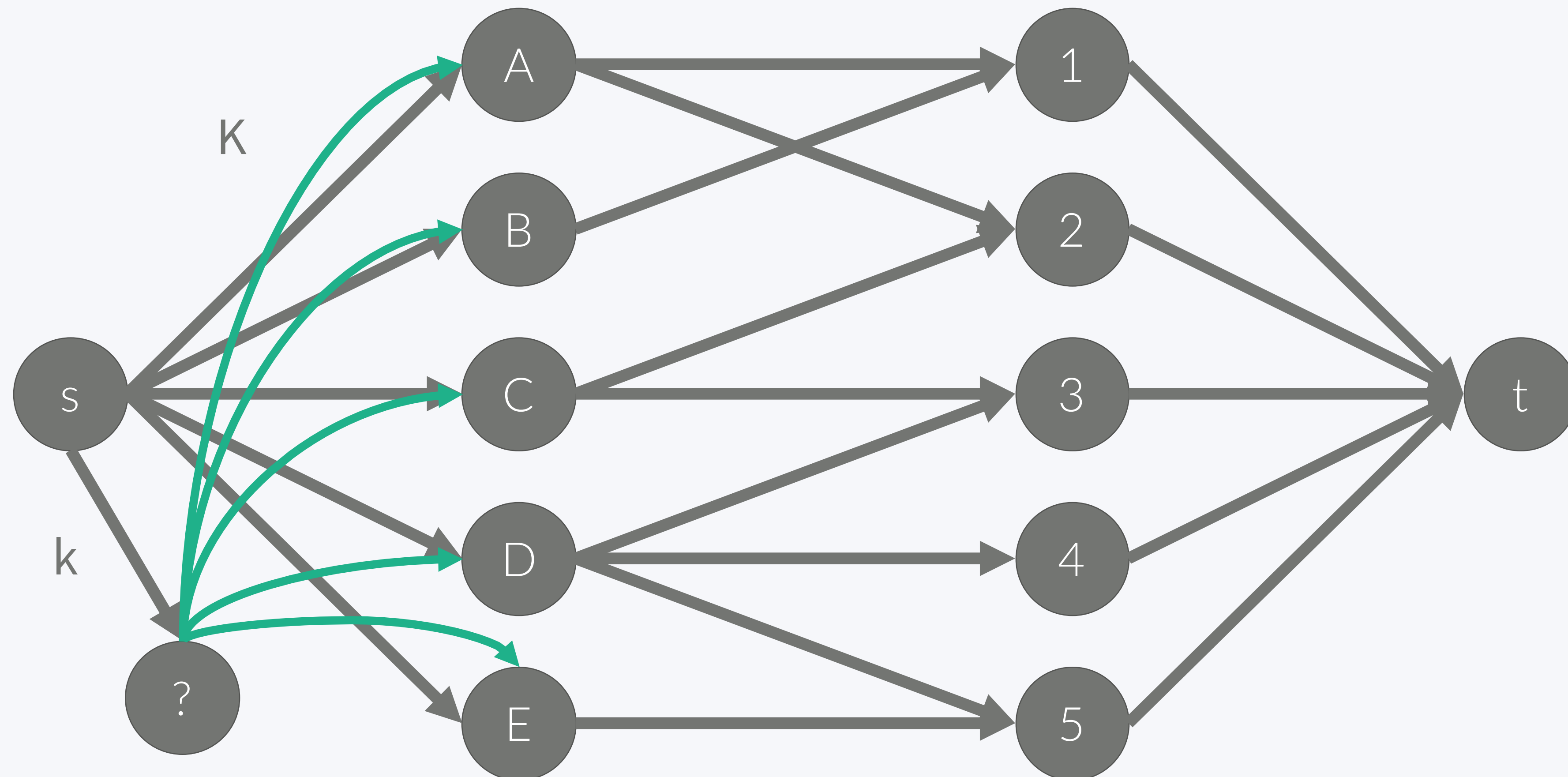


# 열혈강호 4

62

<https://www.acmicpc.net/problem/11378>

- <https://gist.github.com/Baekjoon/df740831fd0a657a01ac>



# 축사 배정

<https://www.acmicpc.net/problem/2188>

- 열혈강호와 똑같은 문제
- 왼쪽: 소, 오른쪽: 축사

# 노트북의 주인을 찾아서

<https://www.acmicpc.net/problem/1298>

- 열혈강호와 똑같은 문제
- 왼쪽: 사람, 오른쪽: 노트북
- <https://gist.github.com/Baekjoon/2742f4217e8efb72d1c1>



# 소수 쌍

<https://www.acmicpc.net/problem/1017>

- 첫 번째 수와 짝지을 수를 미리 구한 다음에
- 나머지를 매칭한다
- 왼쪽: 홀수 오른쪽: 짝수
- <https://gist.github.com/Baekjoon/4f6fefbe83302fa6807b>

# 상어의 저녁식사

<https://www.acmicpc.net/problem/1671>

- 상어는 서로를 먹는다
- A의 크기, 속도, 지능이 B의 크기, 속도, 지능보다 크거나 같으면
- A는 B를 먹을 수 있다
- 한 상어가 최대 2마리 상어만 먹을 수 있다
- 살아남을 수 있는 상어의 수

# 상어의 저녁식사

67

<https://www.acmicpc.net/problem/1671>

- 상어는 서로를 먹는다
- A의 크기, 속도, 지능이 B의 크기, 속도, 지능보다 크거나 같으면
- A는 B를 먹을 수 있다
- 한 상어가 최대 2마리 상어만 먹을 수 있다
- 살아남을 수 있는 상어의 수 =  $N - \text{최대 매칭}$

# 상어의 저녁식사

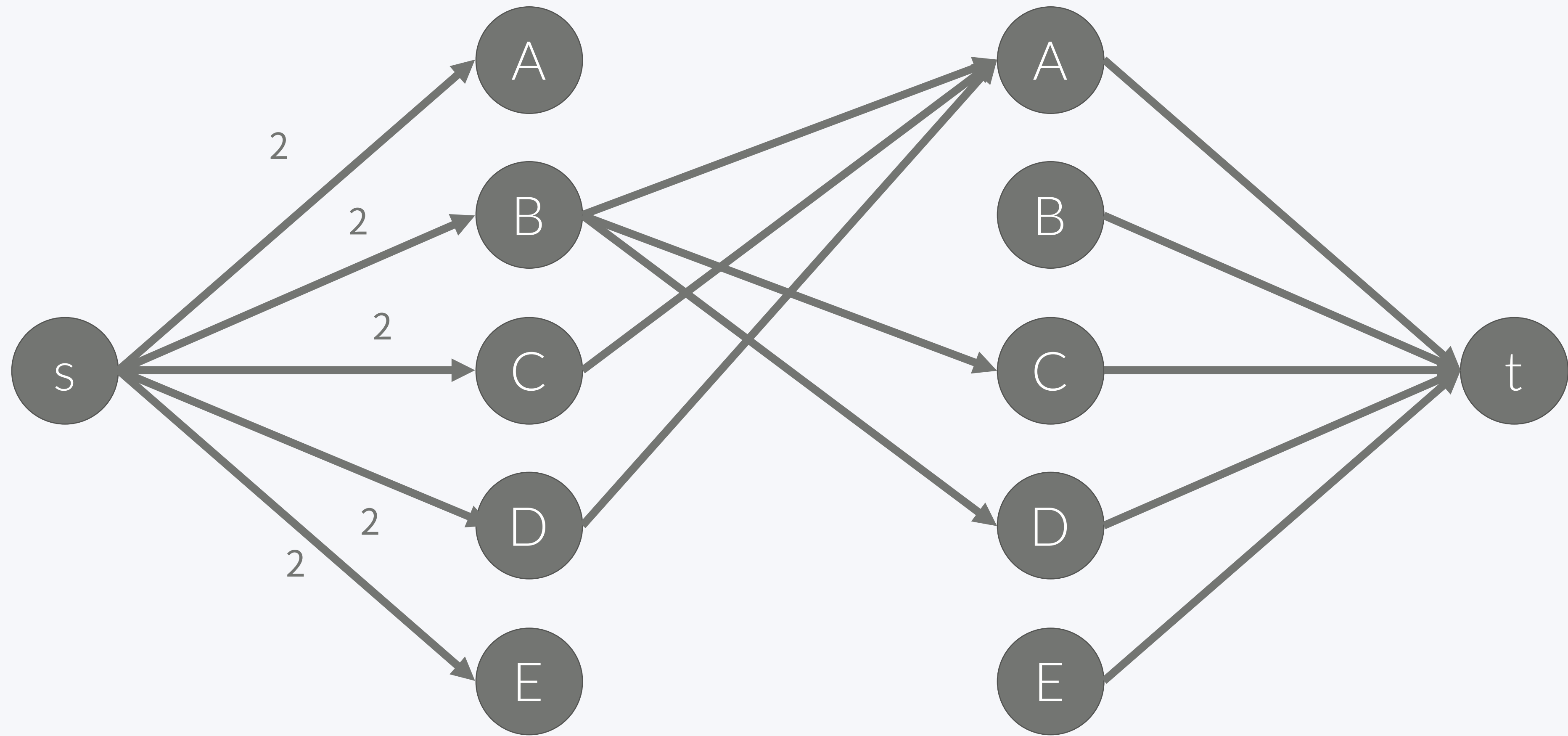
<https://www.acmicpc.net/problem/1671>

- 상어는 서로를 먹는다
- A의 크기, 속도, 지능이 B의 크기, 속도, 지능보다 크거나 같으면
- A는 B를 먹을 수 있다
- 한 상어가 최대 2마리 상어만 먹을 수 있다
- 살아남을 수 있는 상어의 수 =  $N$  - 최대 매칭
- 같은 경우에 서로를 잡아먹는 경우를 방지하기 위해서
- 같은 경우에는  $i < j$ 이면 잡아먹을 수 있다고 가정

# 상어의 저녁식사

<https://www.acmicpc.net/problem/1671>

69



# 상어의 저녁식사

70

<https://www.acmicpc.net/problem/1671>

- C/C++: <https://gist.github.com/Baekjoon/c6732fe467da1a29ac539227c06f7876>

# +이분 탐색

---

# 주차장

72

<https://www.acmicpc.net/problem/1348>

- $R \times C$  격자
  - C는 인접한 칸으로 1초에 1칸씩 이동
  - 주차하는데 필요한 최소 시간
  - 모든 차는 동시에 이동한다.
- 
- .C . . . . . P . X . . .
  - XX . . . . . . . X . . P
  - XX . . . . . C . . . . .



# 주차장

<https://www.acmicpc.net/problem/1348>

- 먼저, BFS로 모든 차와 주차장 사이의 쌍에 대해서 이동하는데 필요한 시간을 구한다.
- 그 다음, Binary Search를 이용해서 허용하는 최대 이동 시간을 결정하고, edge를 적절히 연결한 다음에 matching의 개수가 차의 개수와 같은지 확인

# 주차장

<https://www.acmicpc.net/problem/1348>

- <https://gist.github.com/Baekjoon/4d3d07aa3c255b5baa6f>

그래프  $G$

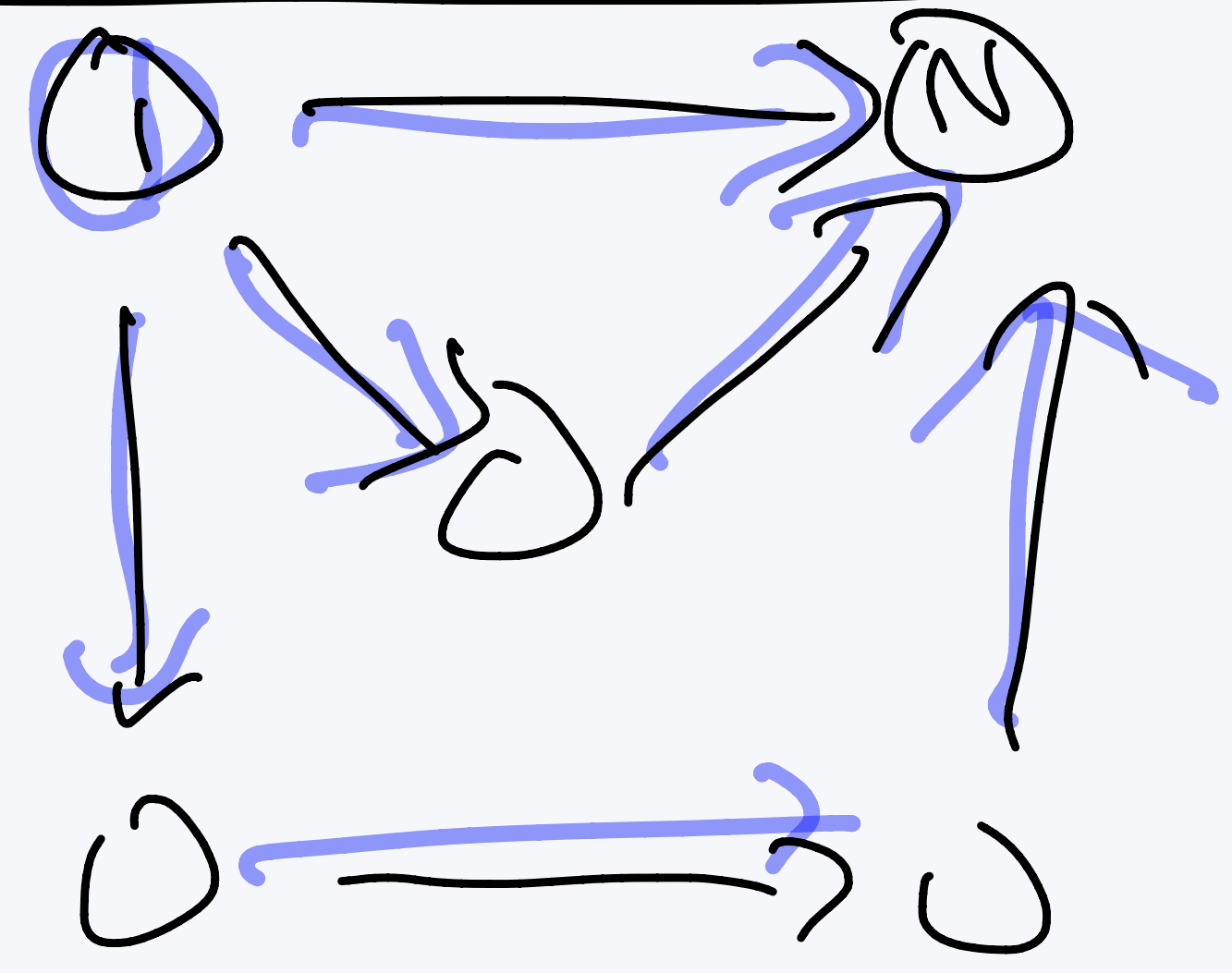
$① \rightarrow ②$

Max flow

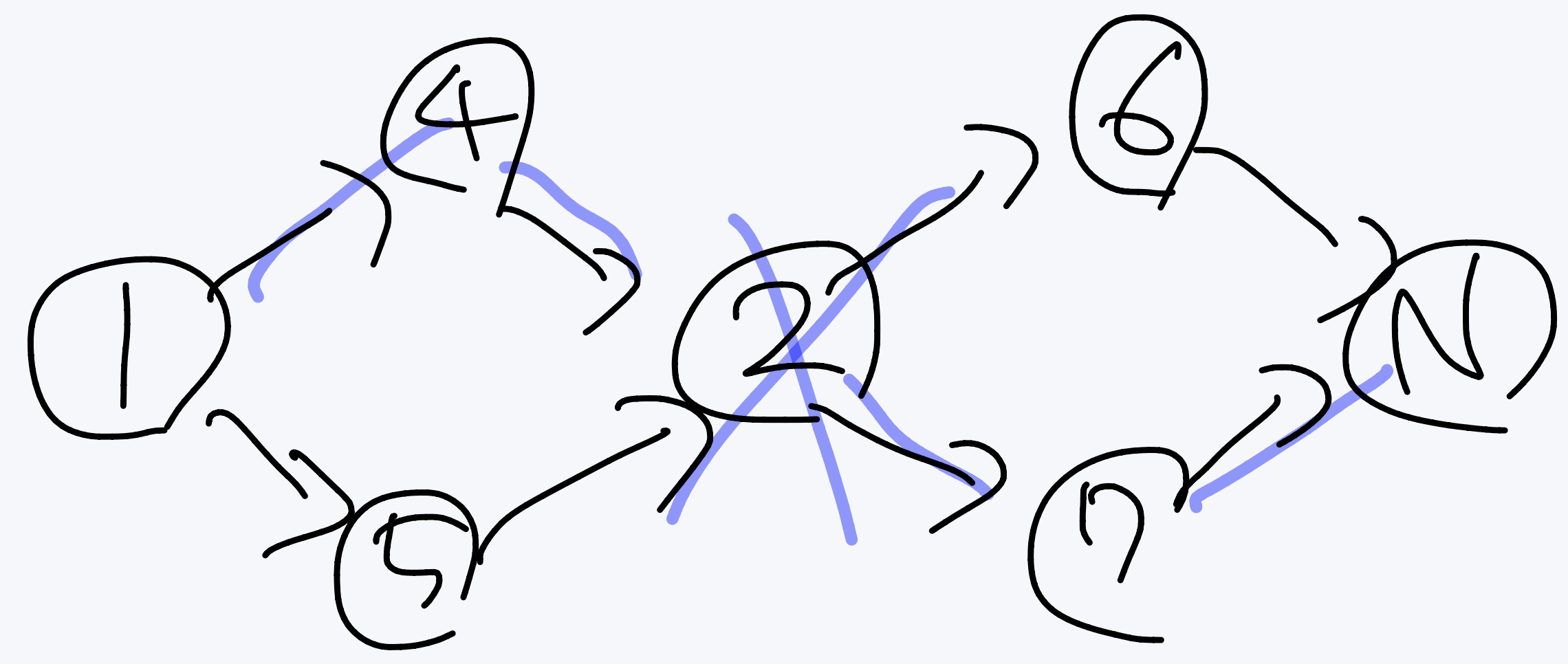
경로의 개수

- ① 한번 사용 안함 X
- ② 한번 방문 안함 X

101  
141  
N



# Min-cut

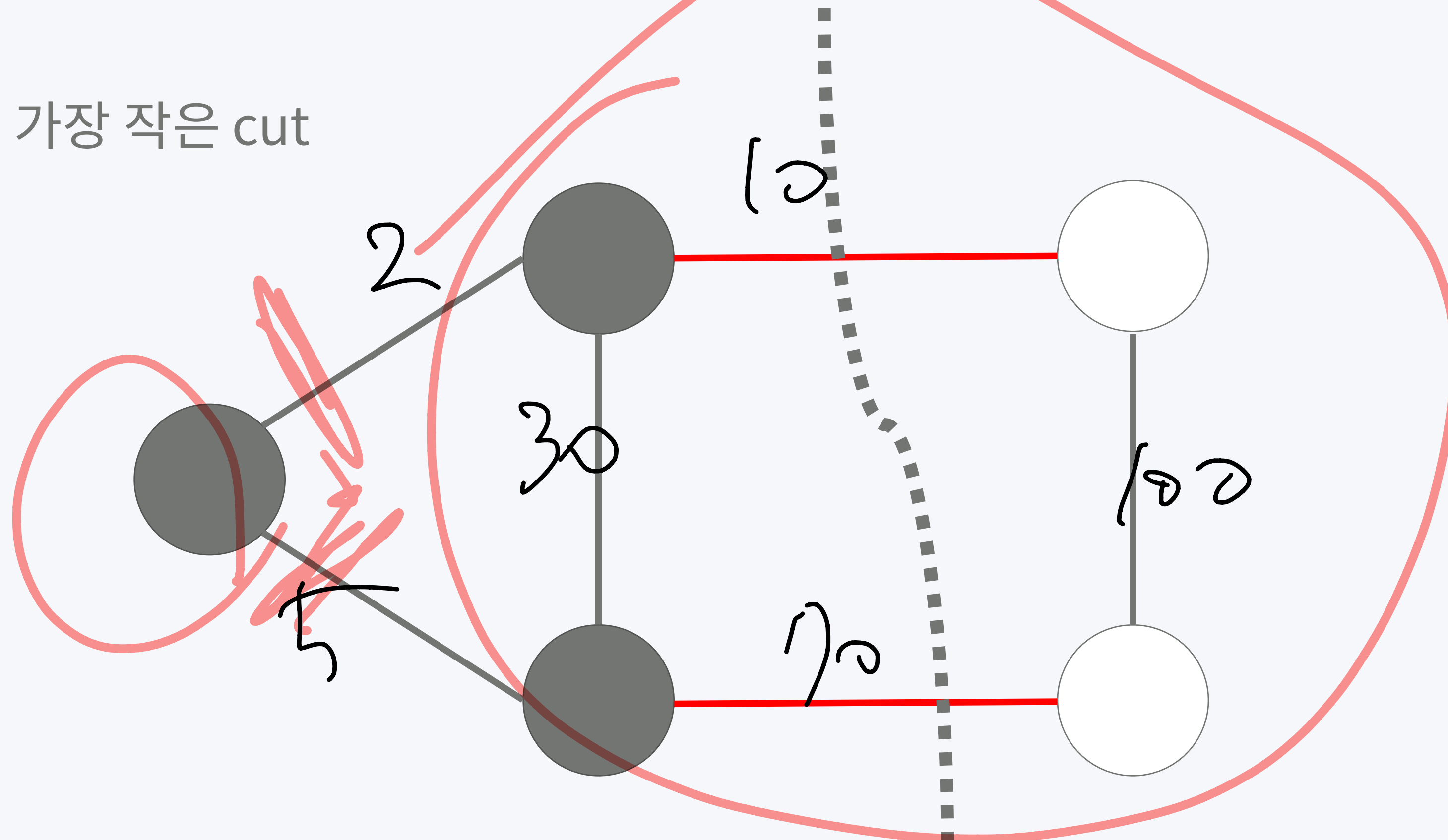


# Minimum cut

76

## Minimum Cut

- Cut: 그래프를 2개의 서로다른 집합으로 나누는 것
- 두 집합을 A와 B라고 했을 때, 한 쪽 끝은 A에 다른 한 쪽 끝은 B에 있는 간선을 cut-set이라고 한다
- Minimum cut: 가장 작은 cut

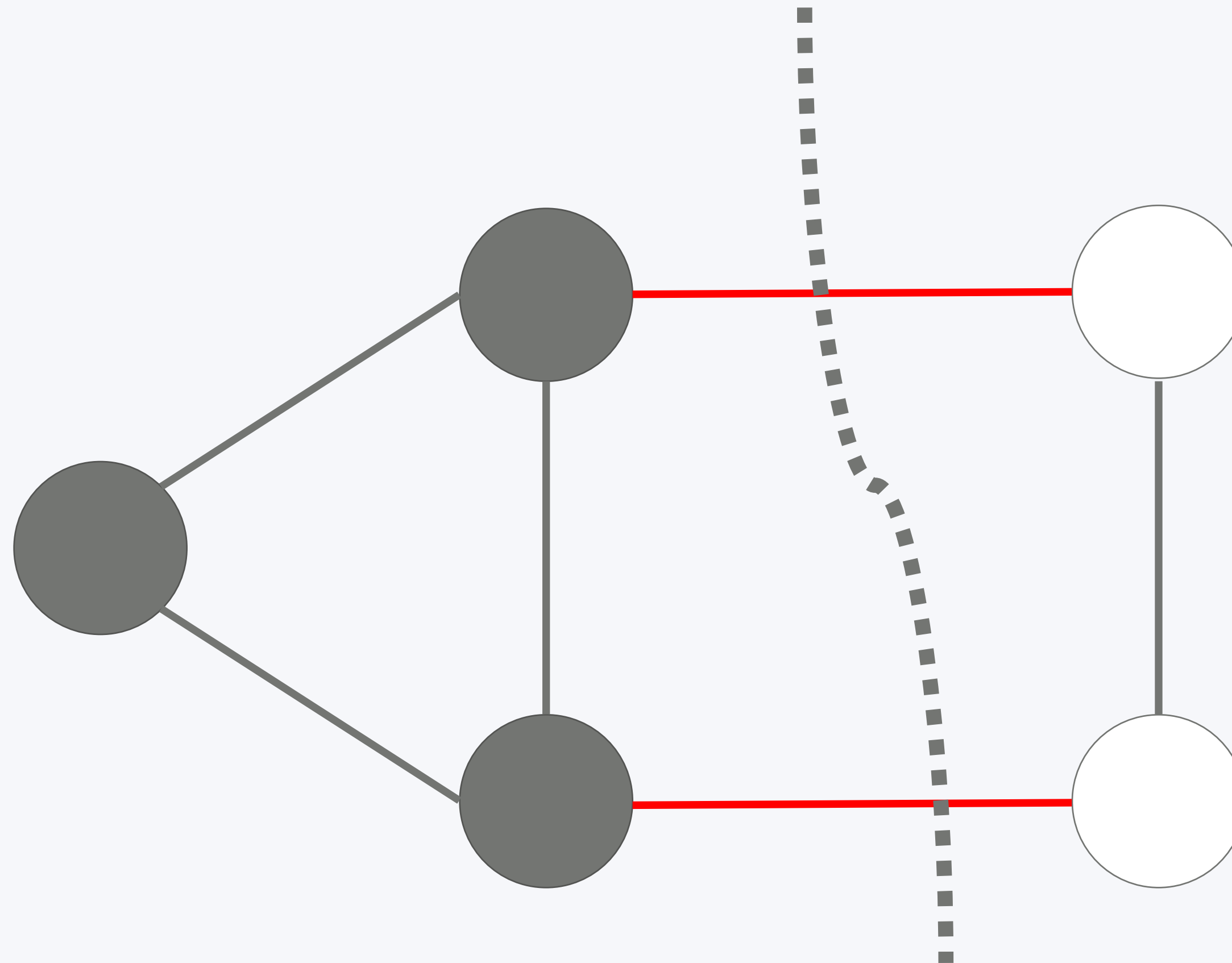


# Max-flow Min-cut Theorem

77

## Max-flow Min-cut Theorem

- Flow network에서 min-cut은 max-flow와 같다.
- 여기서 min-cut은 source->sink로 흐르지 못하게 하기 위해 제거해야 하는 edge capacity 합  
의 최소값



# 학교 가지마!

<https://www.acmicpc.net/problem/1420>

•  $N \times M$  크기의 도시

• 빈 칸: .

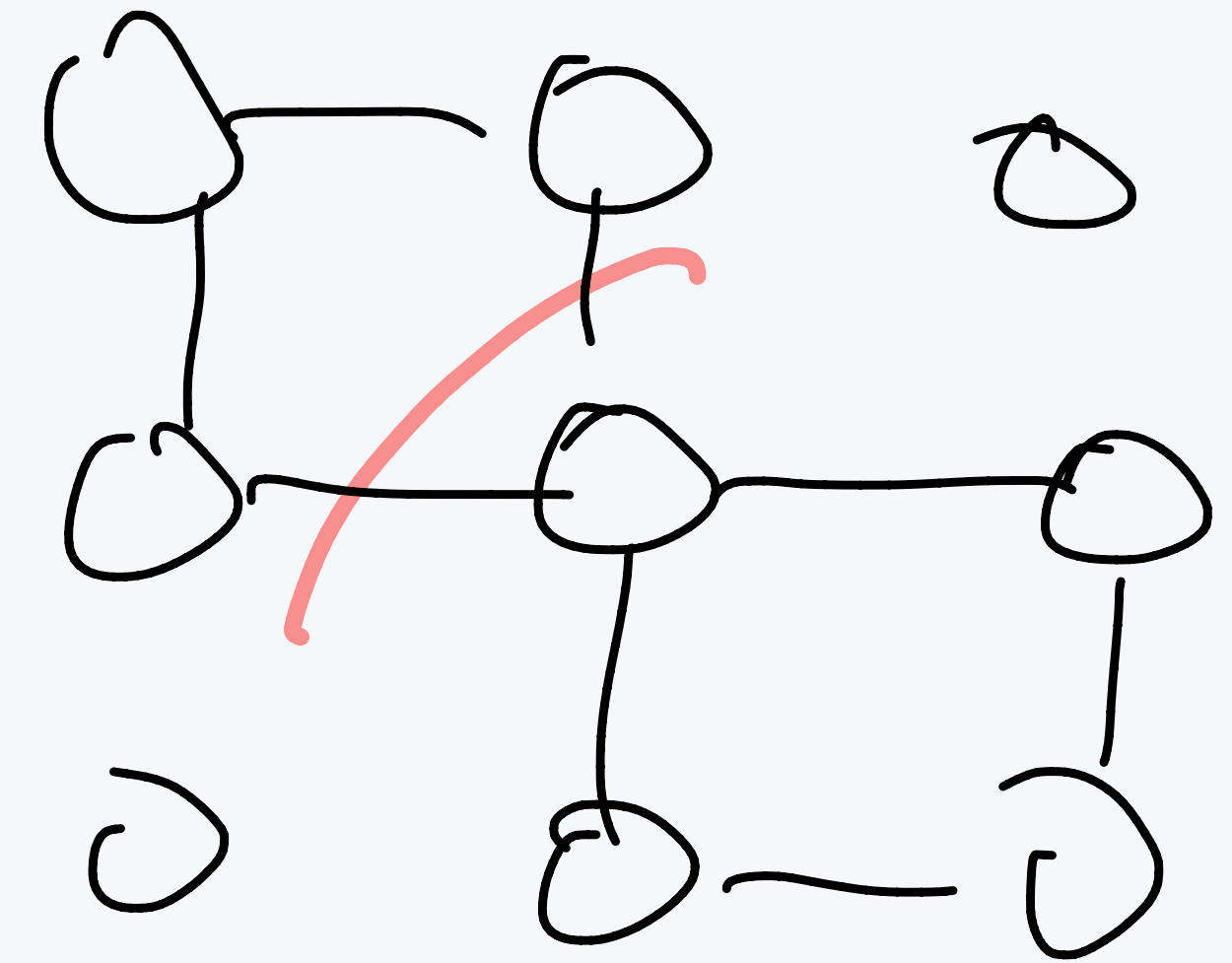
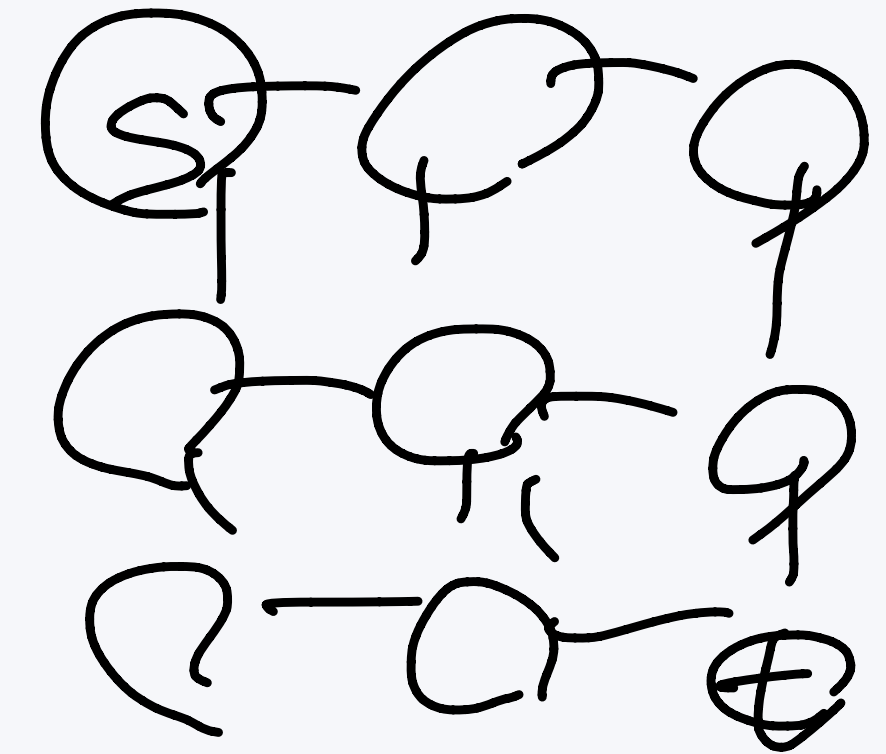
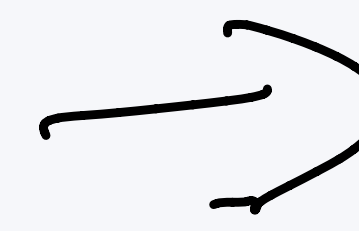
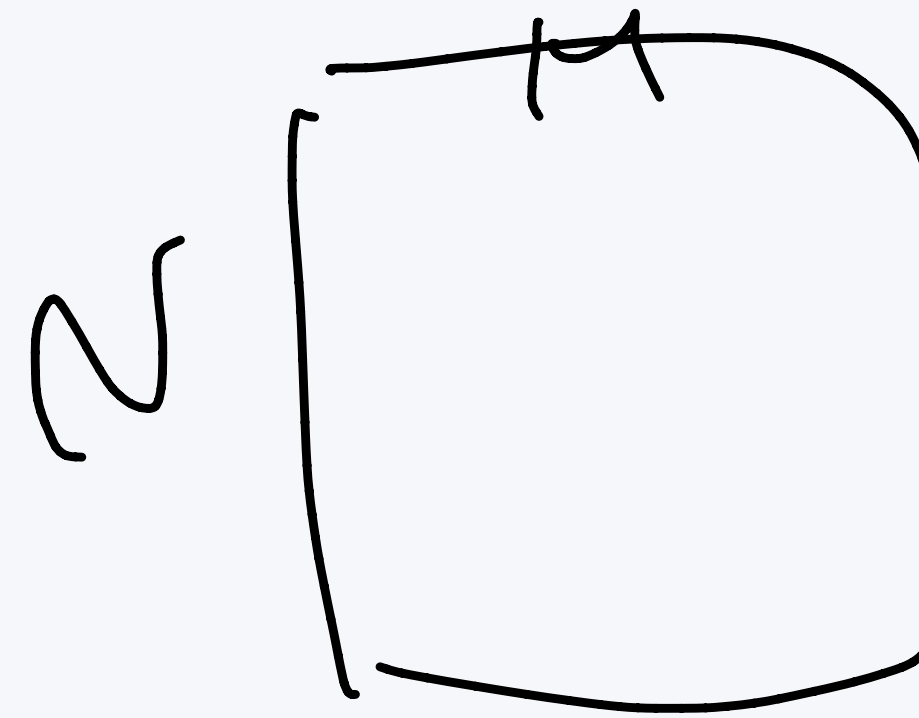
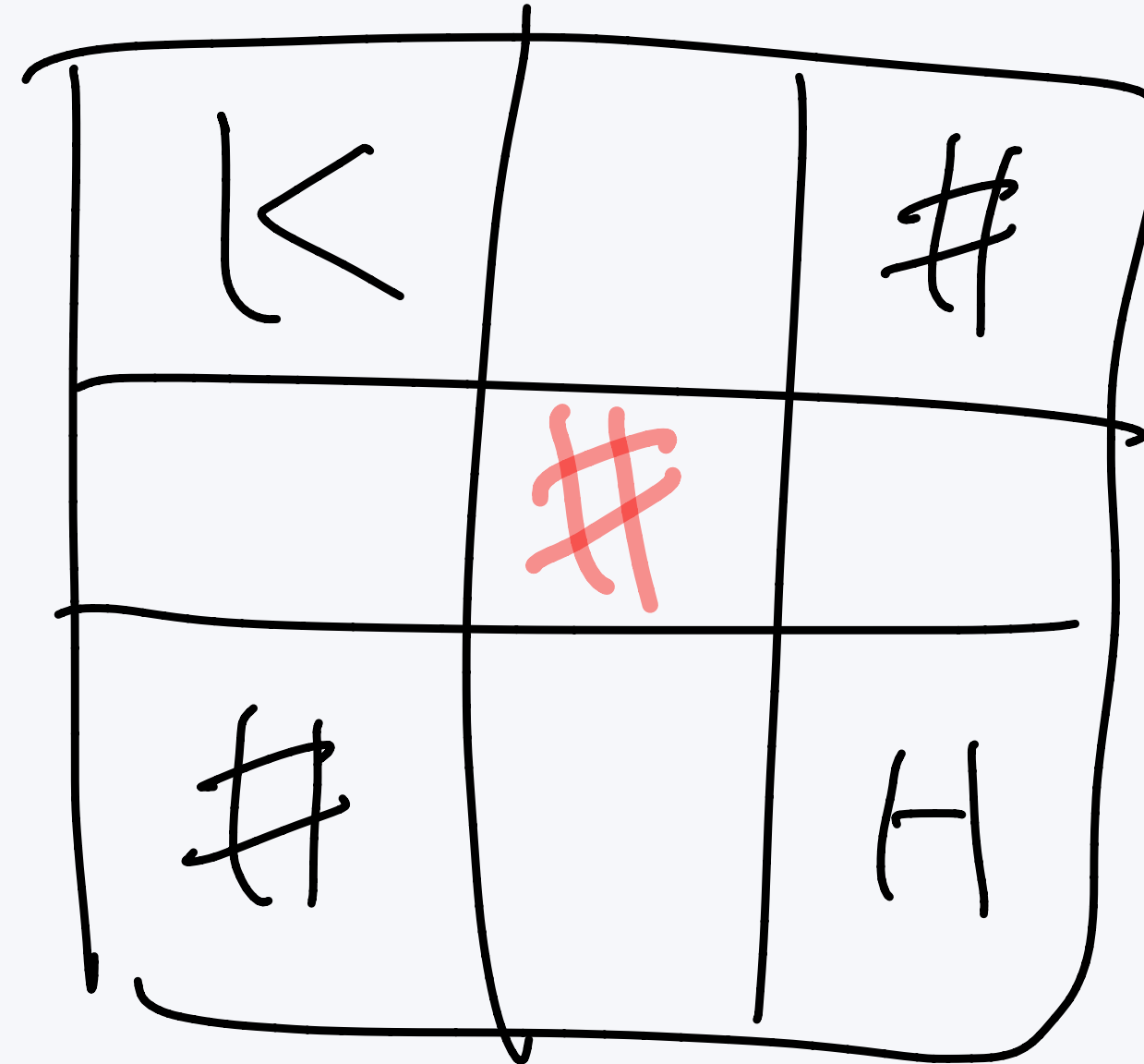
• 벽: #

• 도현: K

• 학교 H

s

t



- 빈 칸을 적절히 벽으로 바꿔서 학교로 가지 못하게 하는 문제
- 최소 개수를 구해야 한다

# 학교 가지마!

<https://www.acmicpc.net/problem/1420>

간 : 강장  
간-간 : 간선

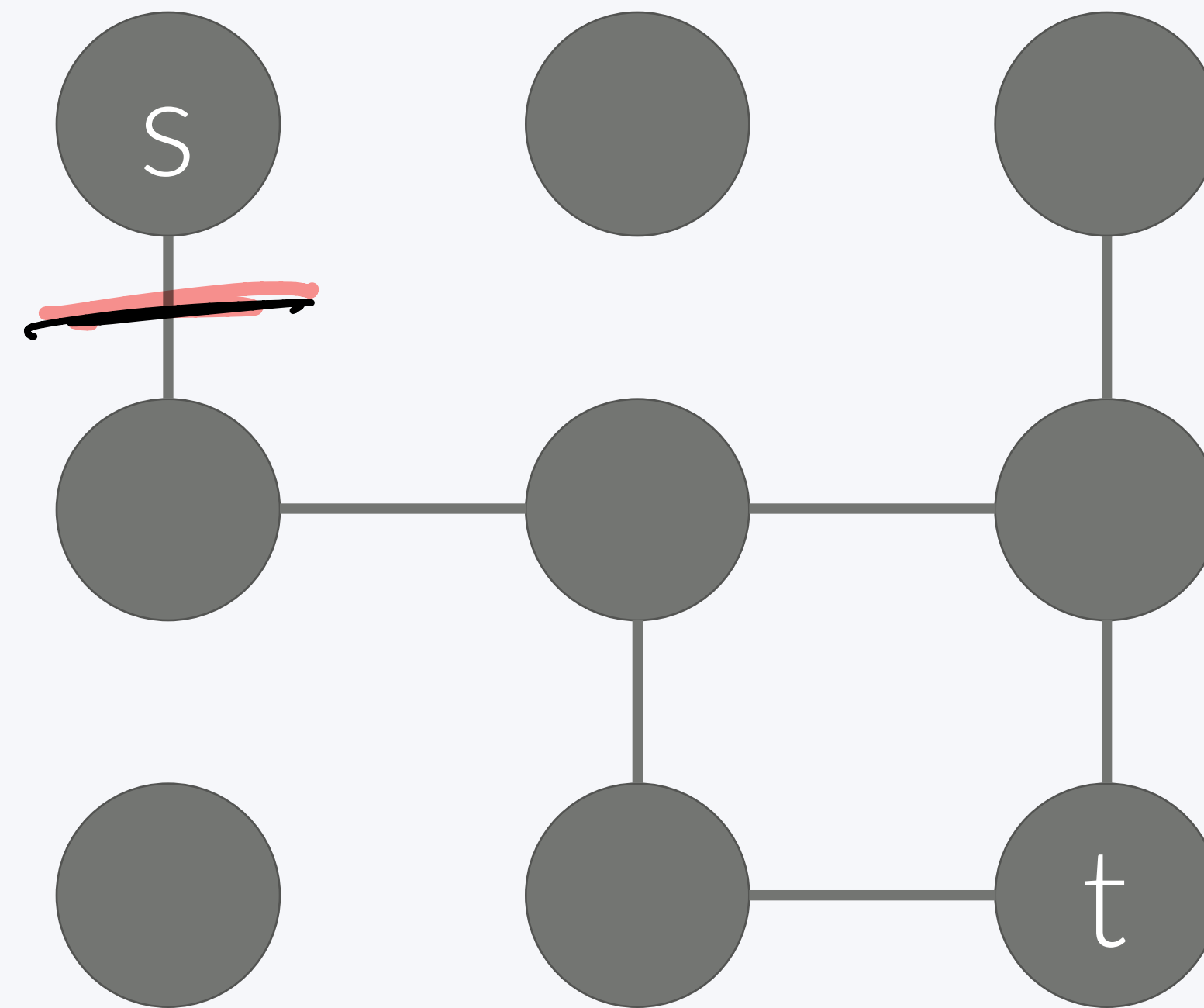
79

- 도시를 플로우 네트워크로 바꾸고, min-cut을 구하는 문제이다.

도한

K	#	
#		H

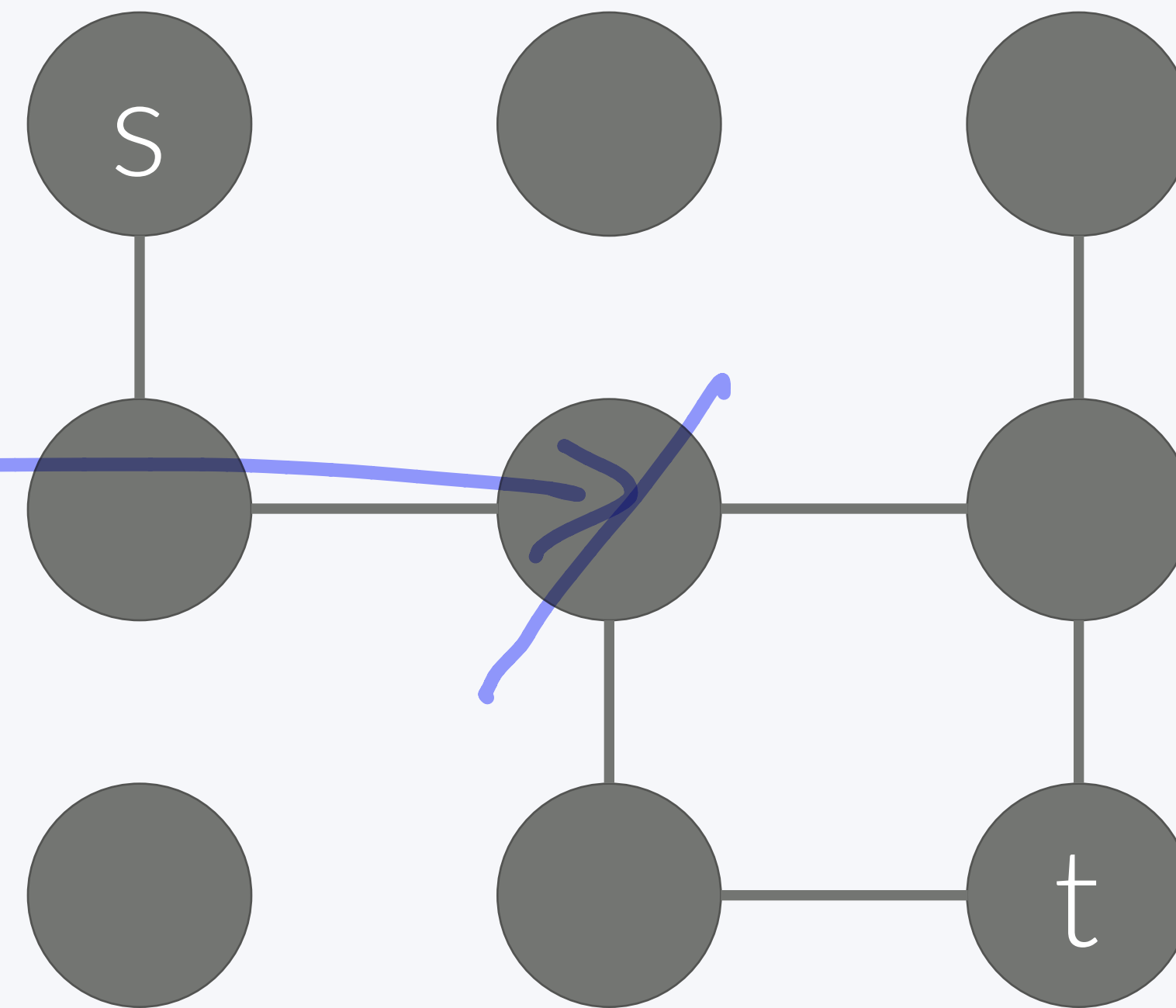
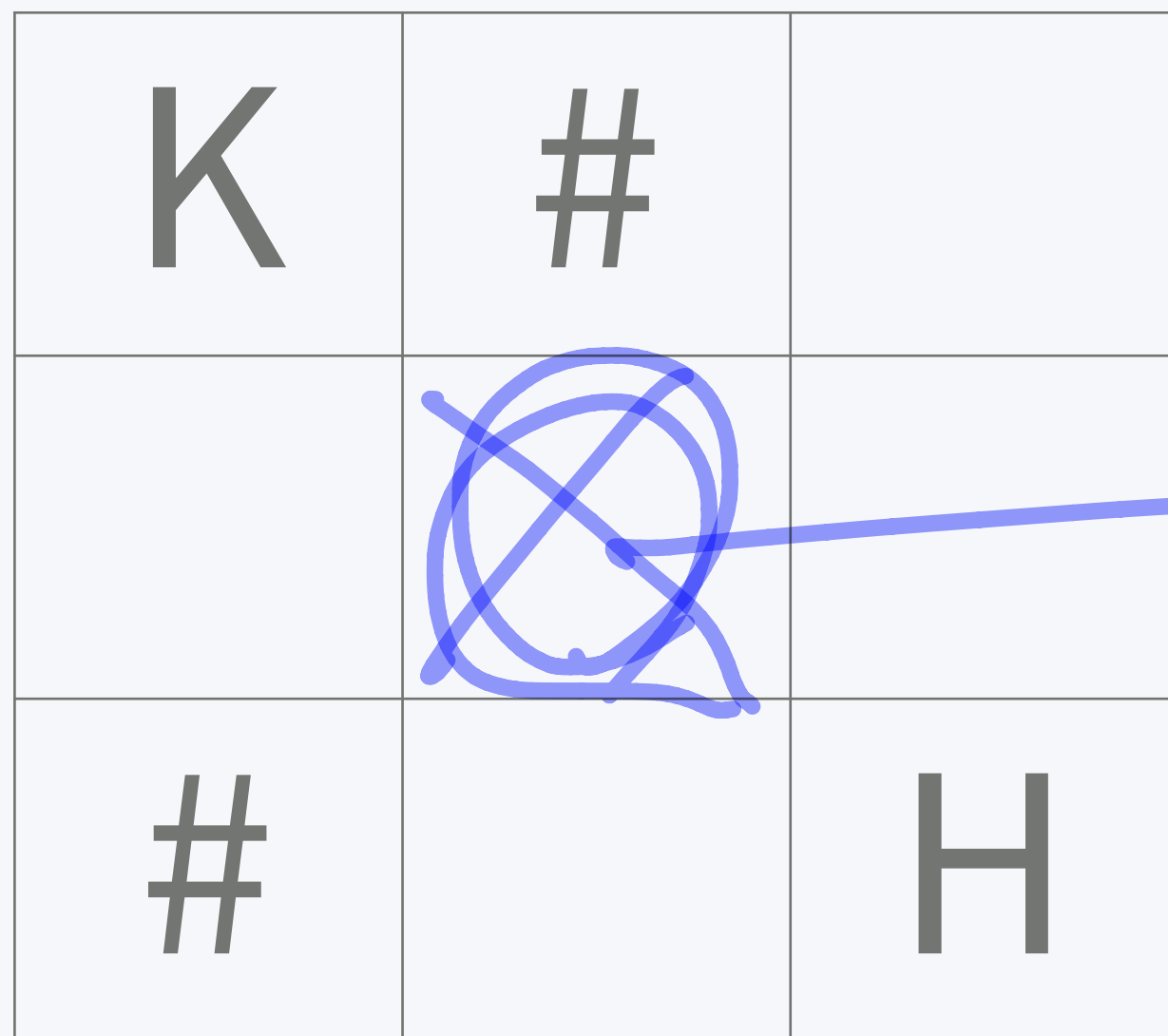
강장



# 학교 가지마!

<https://www.acmicpc.net/problem/1420>

- 그런데, 도시와 도시를 연결하는 edge를 cut하면 안된다
- 도시에 벽을 놓는 것이지, 도시와 도시 사이를 막는 것이 아니기 때문



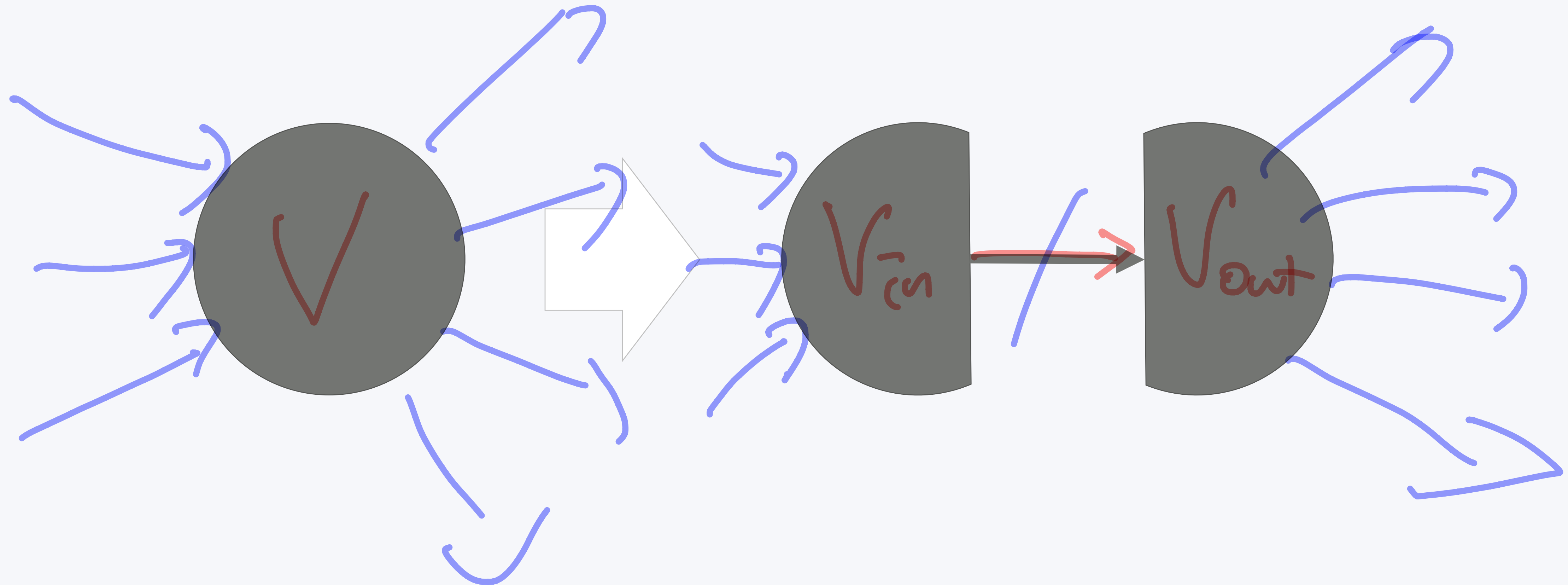


# 학교 가지마!

<https://www.acmicpc.net/problem/1420>

- 각 칸을 둘로 나눈다
- capacity는 몇?

Capacity = 1

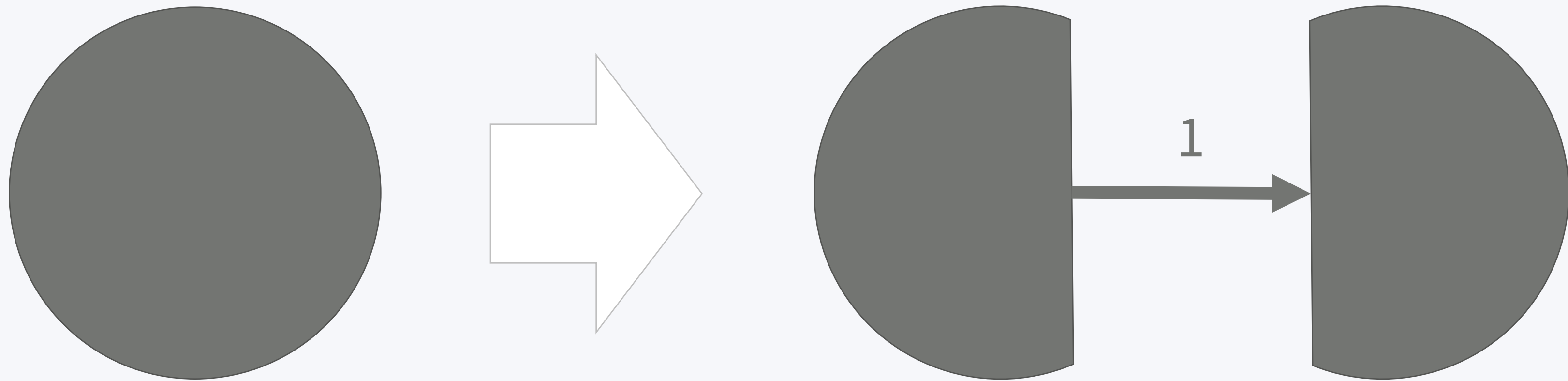


# 학교 가지마!

82

<https://www.acmicpc.net/problem/1420>

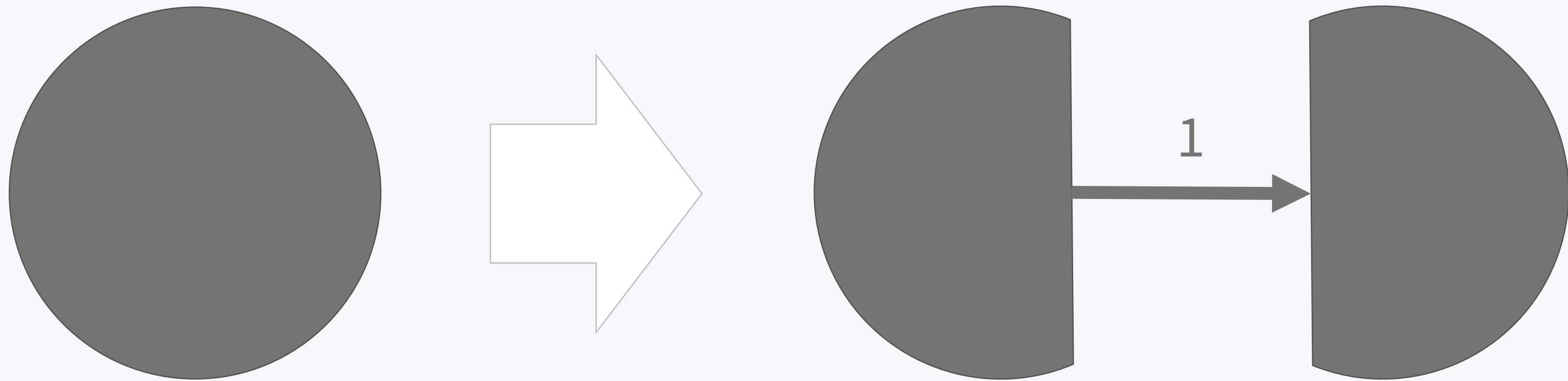
- 각 칸을 둘로 나눈다
- capacity는 몇? 1



# 학교 가지마!

<https://www.acmicpc.net/problem/1420>

- 정점  $X$ 를  $X_{in}$ 과  $X_{out}$  으로 나누고,  $X_{in} \rightarrow X_{out}$ 은 1로 연결
- $X$ 와  $Y$ 를 이동할 수 있으면,  $X_{out} \rightarrow Y_{in}$ ,  $Y_{out} \rightarrow X_{in}$  을 연결
- 이 때 capacity?

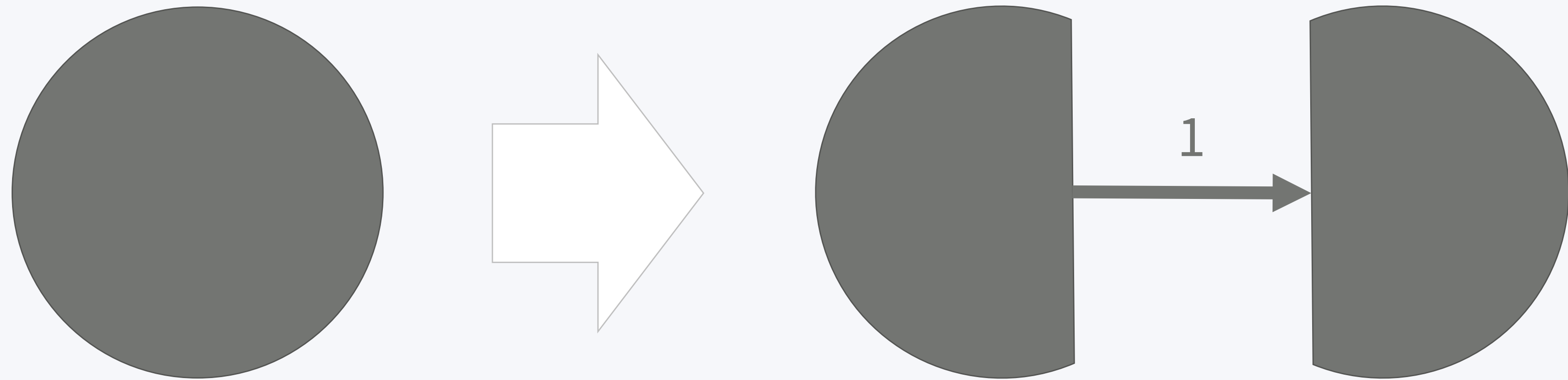


# 학교 가지마!

84

<https://www.acmicpc.net/problem/1420>

- <https://gist.github.com/Baekjoon/236b37c8d011e3d79bd8>



# 최소 버텍스 커버

---

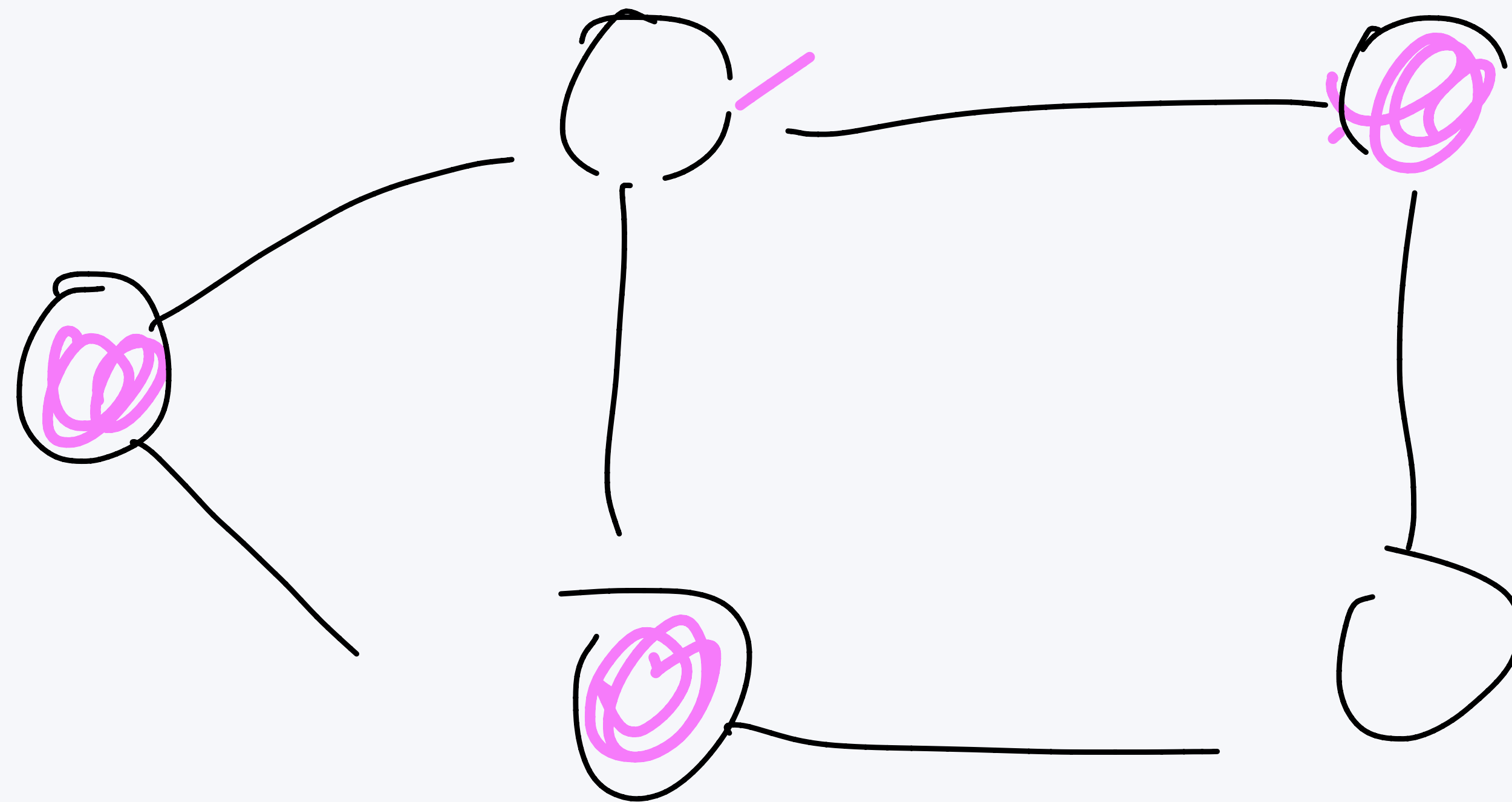
# 최소 버텍스 커버

Minium Vertex Cover

86

정제  
✓

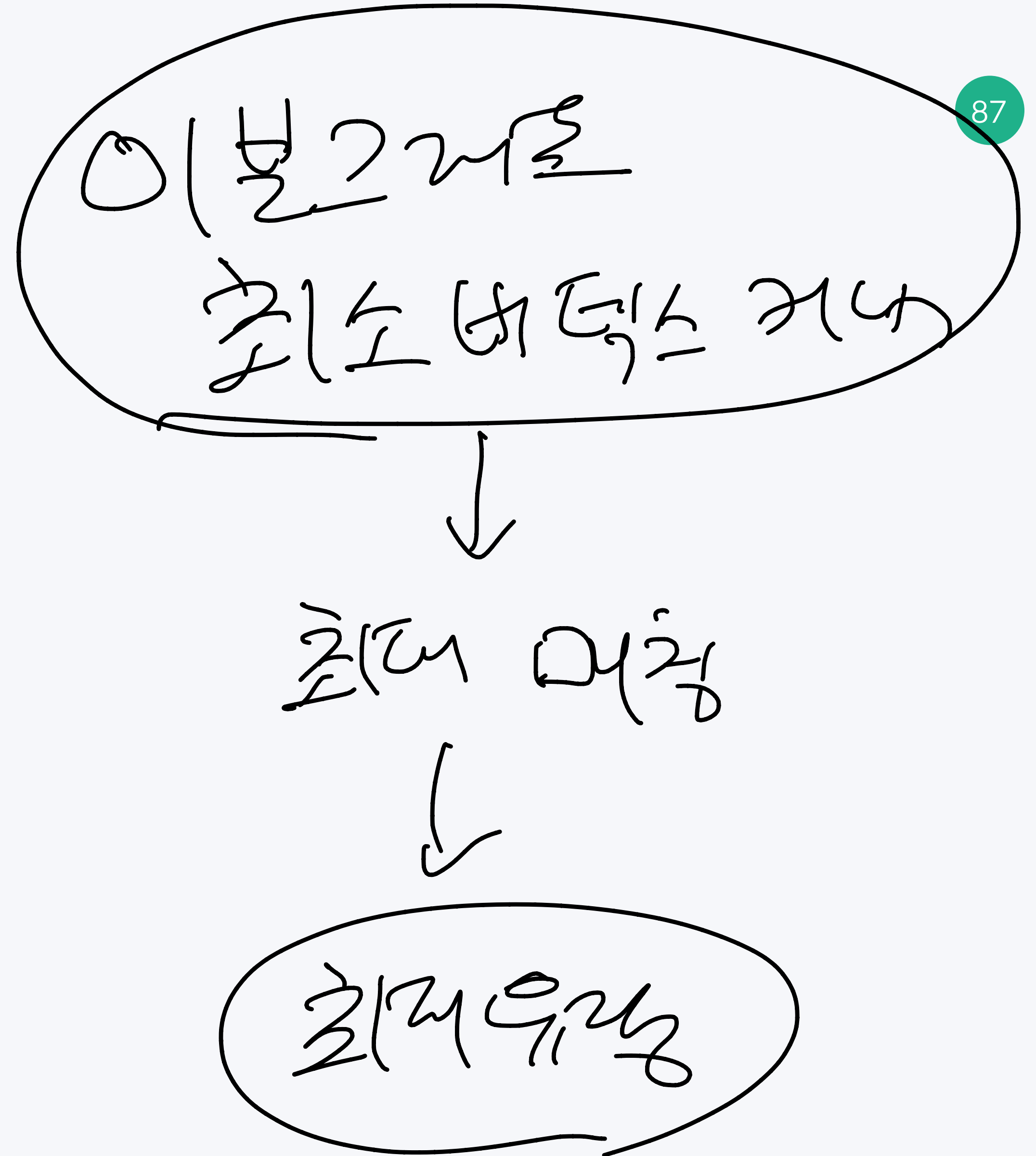
- Vertex Cover: 정점 집합  $S$ 가 있을 때, 모든 간선은 양 끝점중 하나가  $S$ 에 포함되어야 함
- Minimum Vertex Cover: 최소값



# König's theorem

Minimum Vertex Cover

- Bipartite Graph에서
- Maximum Matching은
- Minimum Vertex Cover와 같다



# 돌멩이 제거

<https://www.acmicpc.net/problem/1867>

- N행 N열에 K개의 돌멩이가 있다
- 격자 한 칸에 들어가 있고, 두 개가 한 칸에 들어간 경우는 없다
- 한 행 또는 한 열을 따라서 직선으로 움직이면서 돌멩이를 모두 줍는다
- 최소 몇 번이나 달려야 하는가?



# 돌멩이 제거

<https://www.acmicpc.net/problem/1867>

- 이분 그래프를 만든다
- 왼쪽: 행
- 오른쪽: 열
- $i$ 행  $j$ 열에 돌멩이가 있으면
- 왼쪽  $i \rightarrow$  오른쪽  $j$ 를 연결

# 돌맹이 제거

<https://www.acmicpc.net/problem/1867>

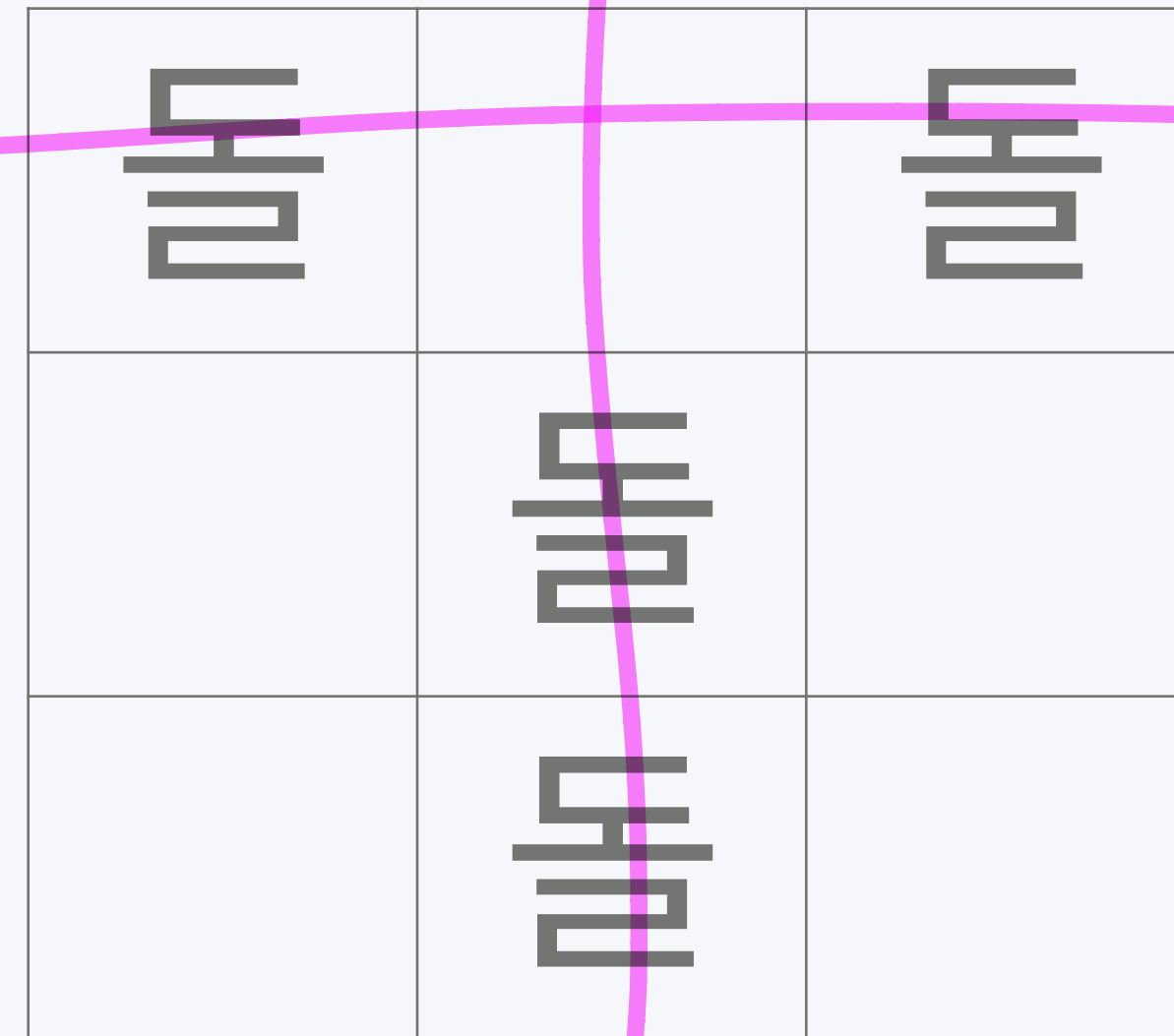
- 3 4
- 1 1
- 1 3
- 2 2
- 3 2

32칸 (행, 열)

$N \times N$

32칸은 2, 0, 1

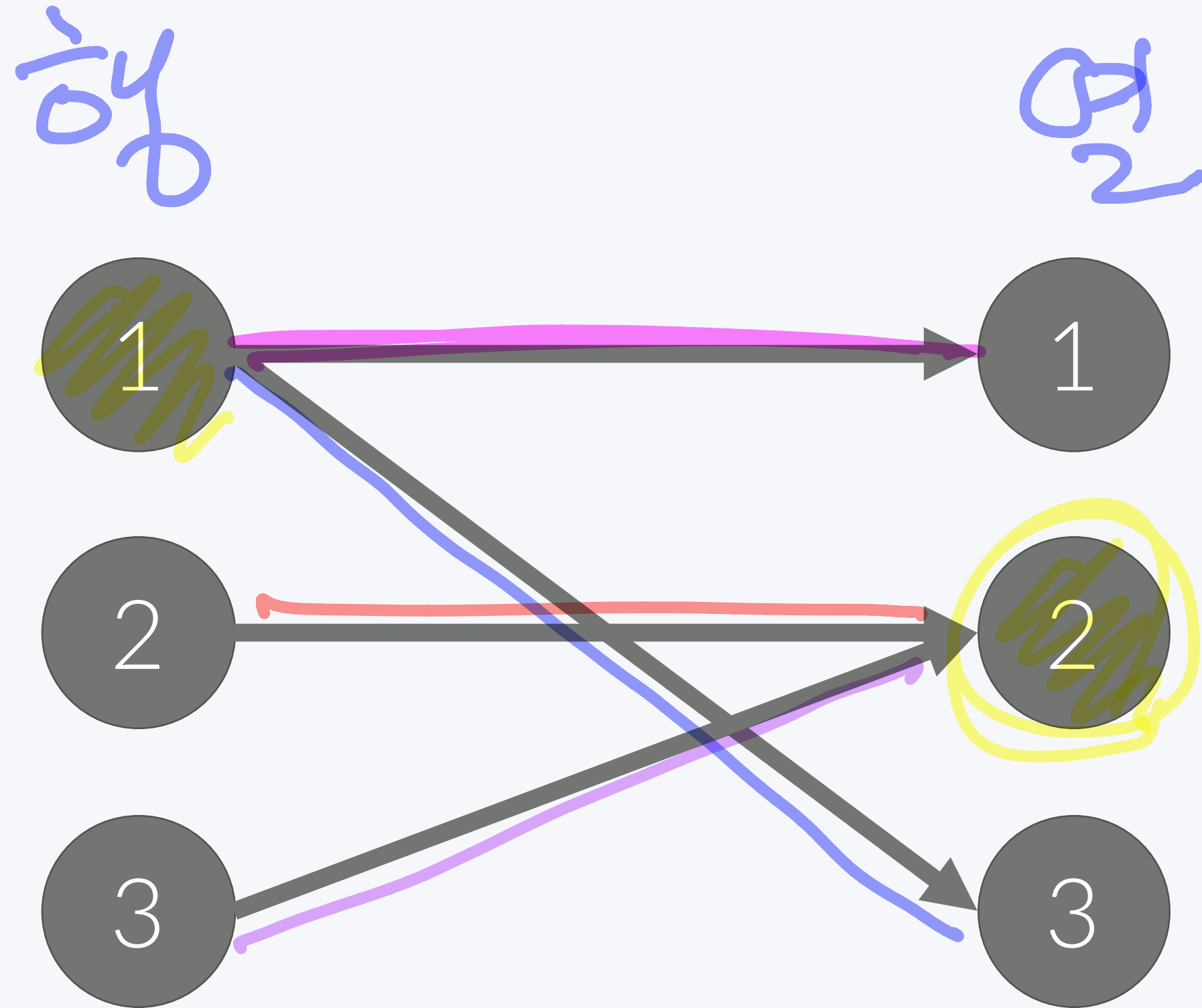
90



돌		돌
	돌	
	돌	

# 돌멩이 제거

<https://www.acmicpc.net/problem/1867>



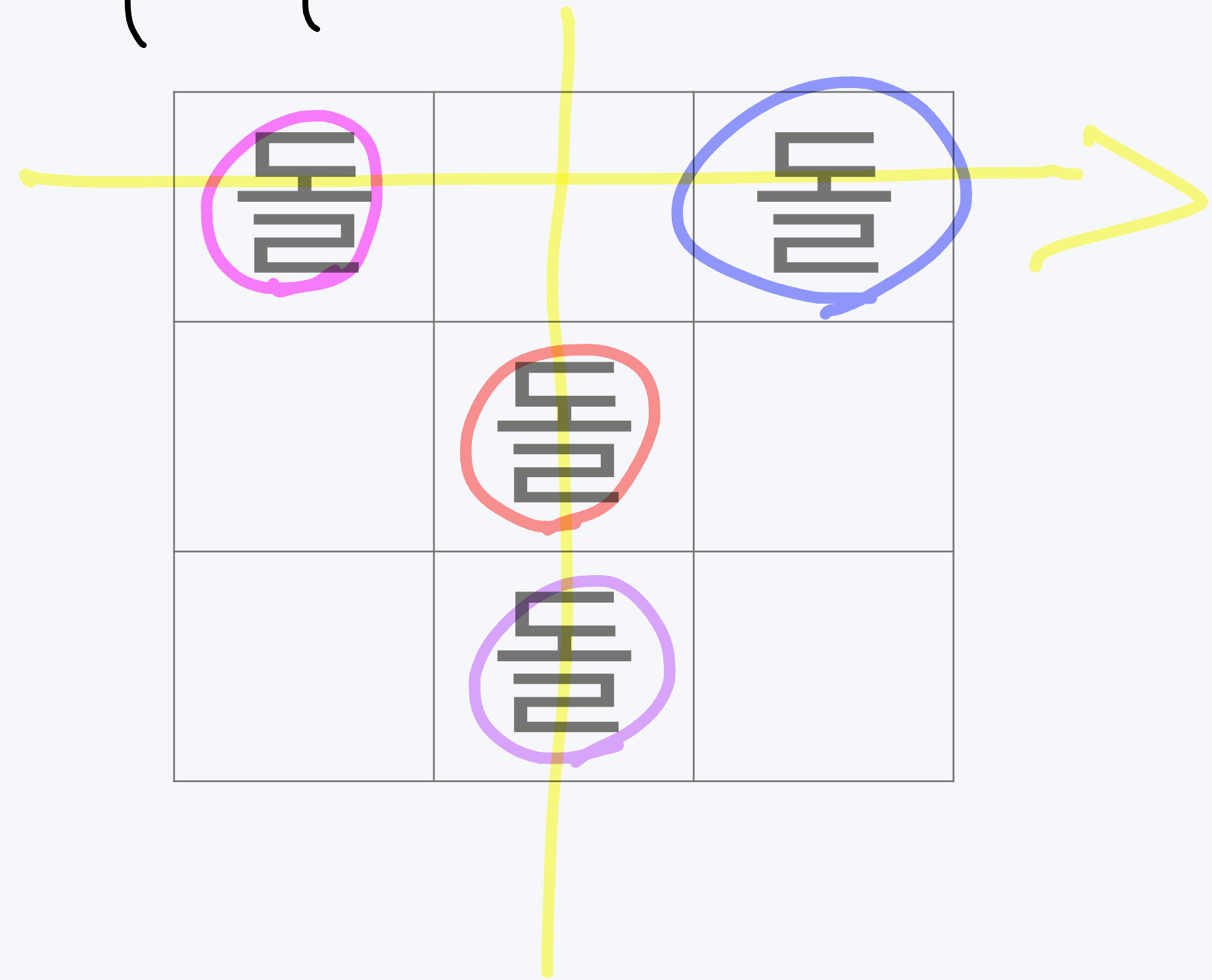
정정 김하늘

3등 간판

91

(정정 김하늘)  
↑  
↑

5등 간판 3등  
하늘 Sai도



# 돌멩이 제거

<https://www.acmicpc.net/problem/1867>

- <https://gist.github.com/Baekjoon/9481c68db12c6c9f8786>

# 게시판 구멍 막기

<https://www.acmicpc.net/problem/2414>

- $N \times M$  모양의 게시판에 구멍이 뚫려있다
- 폭이 1인 테이프로 막으려고 한다
- 길이는 무한하지만
- 끊어내는 횟수를 최소로
- 아래 예제 정답: 4

\* . \* .

. \* \* \*

\* \* \* .

. . \* .

# 게시판 구멍 막기

<https://www.acmicpc.net/problem/2414>

- 돌멩이 줍기와 똑같은 문제다
- 하지만 구멍만 막아야 한다.
- 따라서 그래프를 조금 수정해야 한다

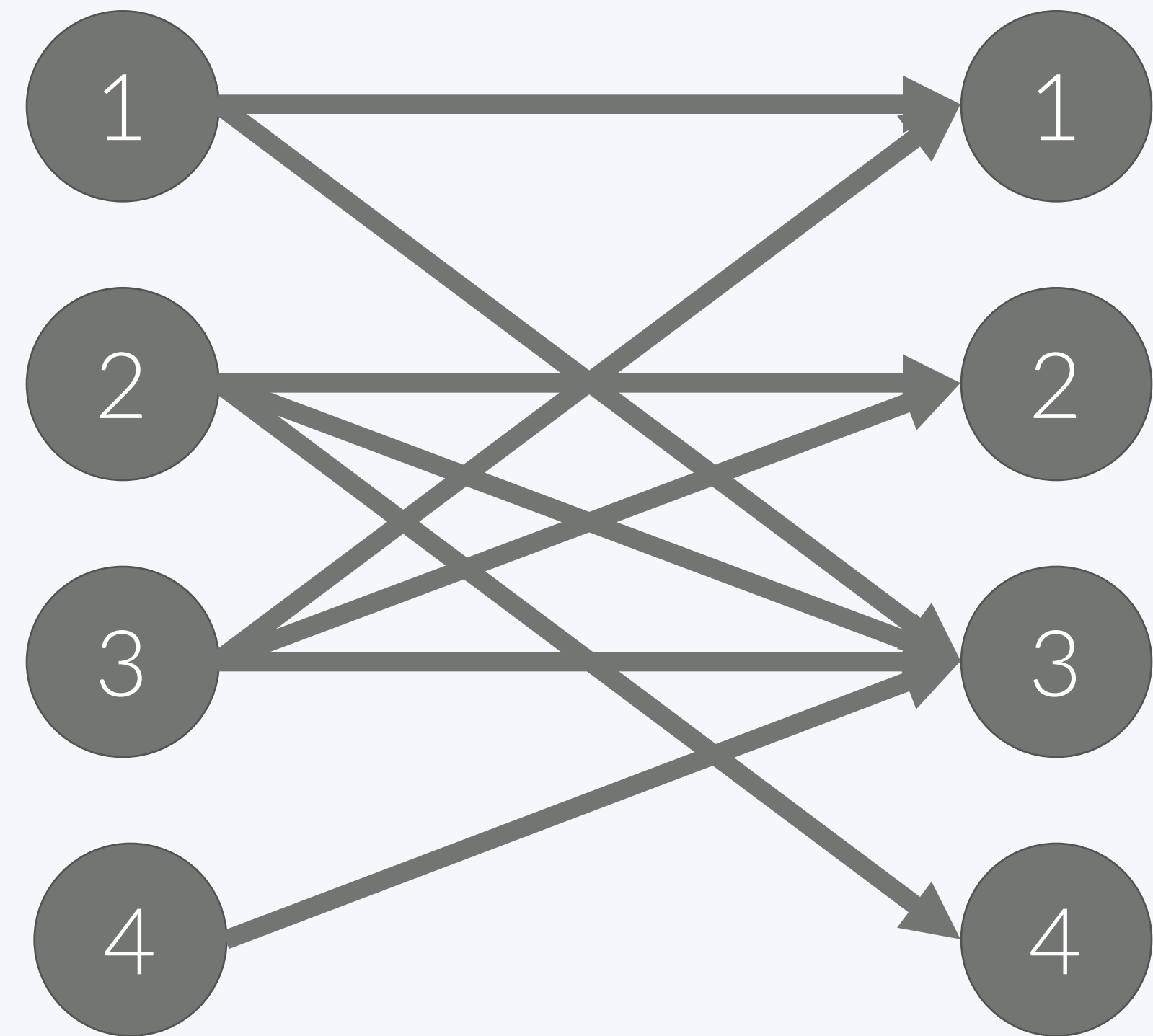
# 게시판 구멍 막기

95

<https://www.acmicpc.net/problem/2414>

- 돌멩이 줍기와 똑같은 문제다. 이 그래프가 아니다.

#		#	
	#	#	#
#	#	#	
		#	

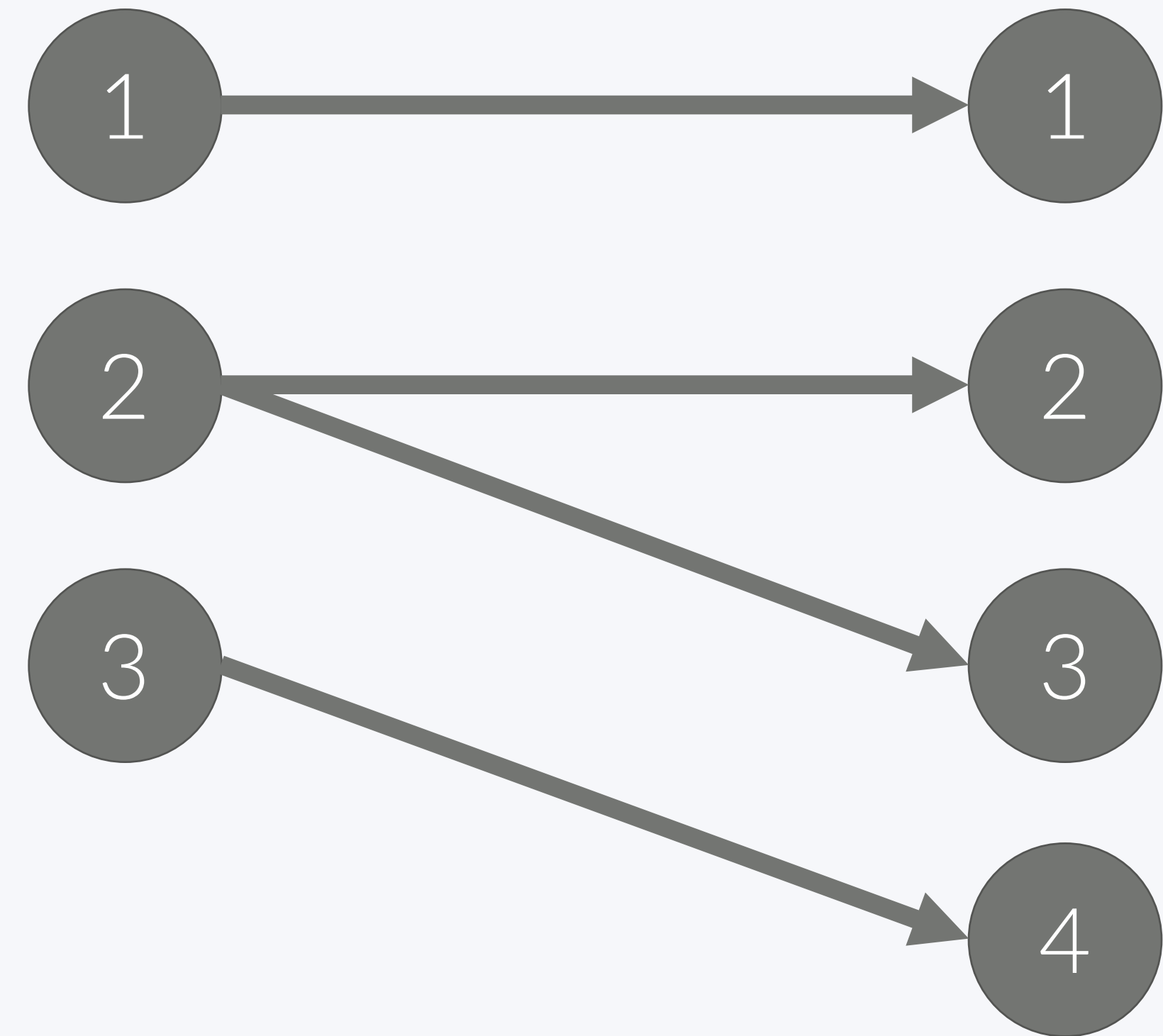


# 게시판 구멍 막기

96

<https://www.acmicpc.net/problem/2414>

- 간단하게 1x8 크기의 경우를 생각
- .#.#.#.
- 1행은 한 번에 모두 막을 수가 없음
- 인접한 # 끼리 그룹으로 나뉘야 함
- .1.22.3.
- 이렇게 한 행을 여러 그룹으로 나뉘야 함
- 열을 기준으로도 나눌 수 있음
- .1.23.4.





# 게시판 구멍 막기

97

<https://www.acmicpc.net/problem/2414>

#		#	
	#	#	#
#	#	#	
		#	

1		2	
	3	3	3
4	4	4	
		5	

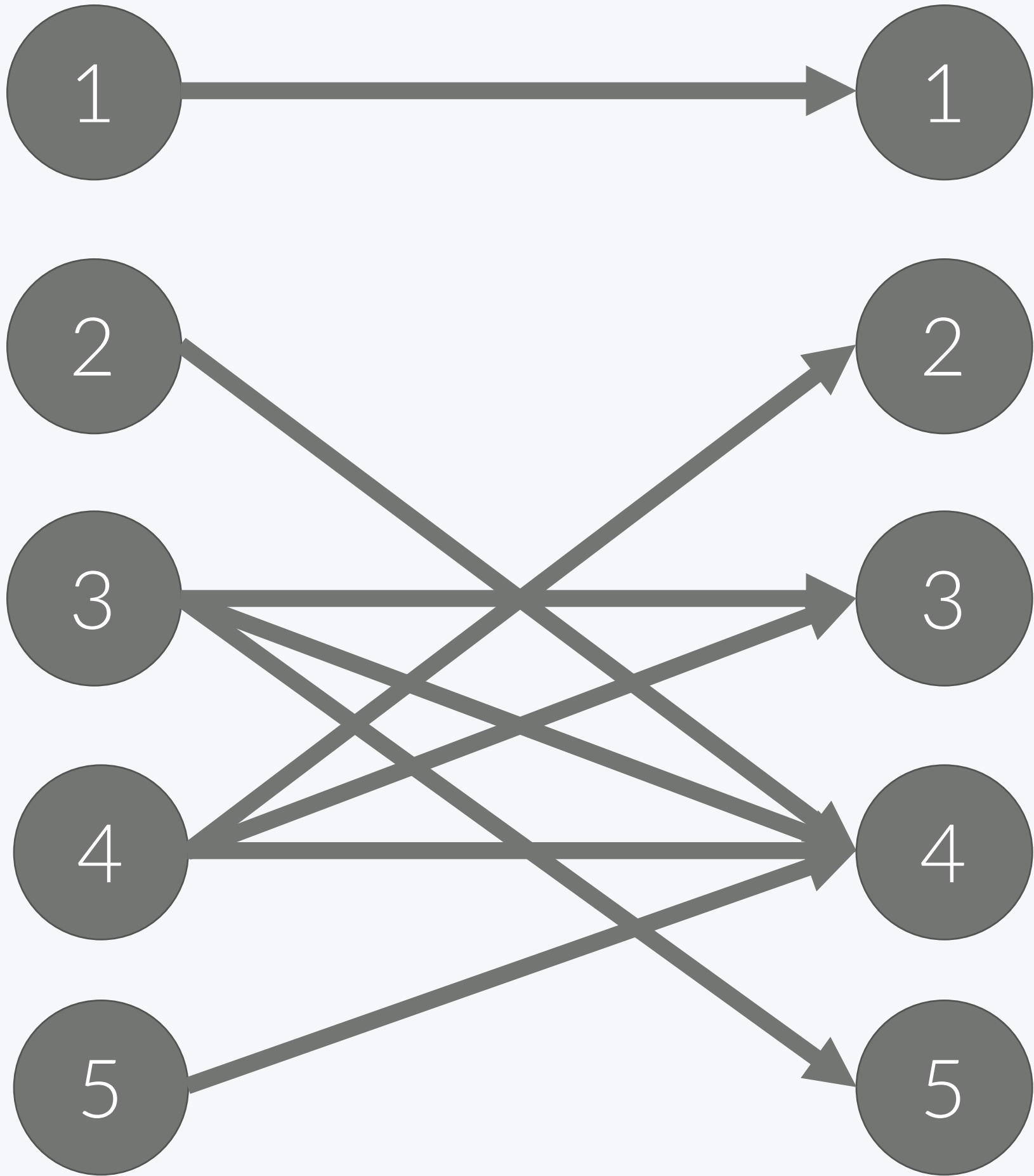
1		4	
	3	4	5
2	3	4	
		4	

# 게시판 구멍 막기

<https://www.acmicpc.net/problem/2414>

1		2	
	3	3	3
4	4	4	
		5	

1		4	
	3	4	5
2	3	4	
		4	



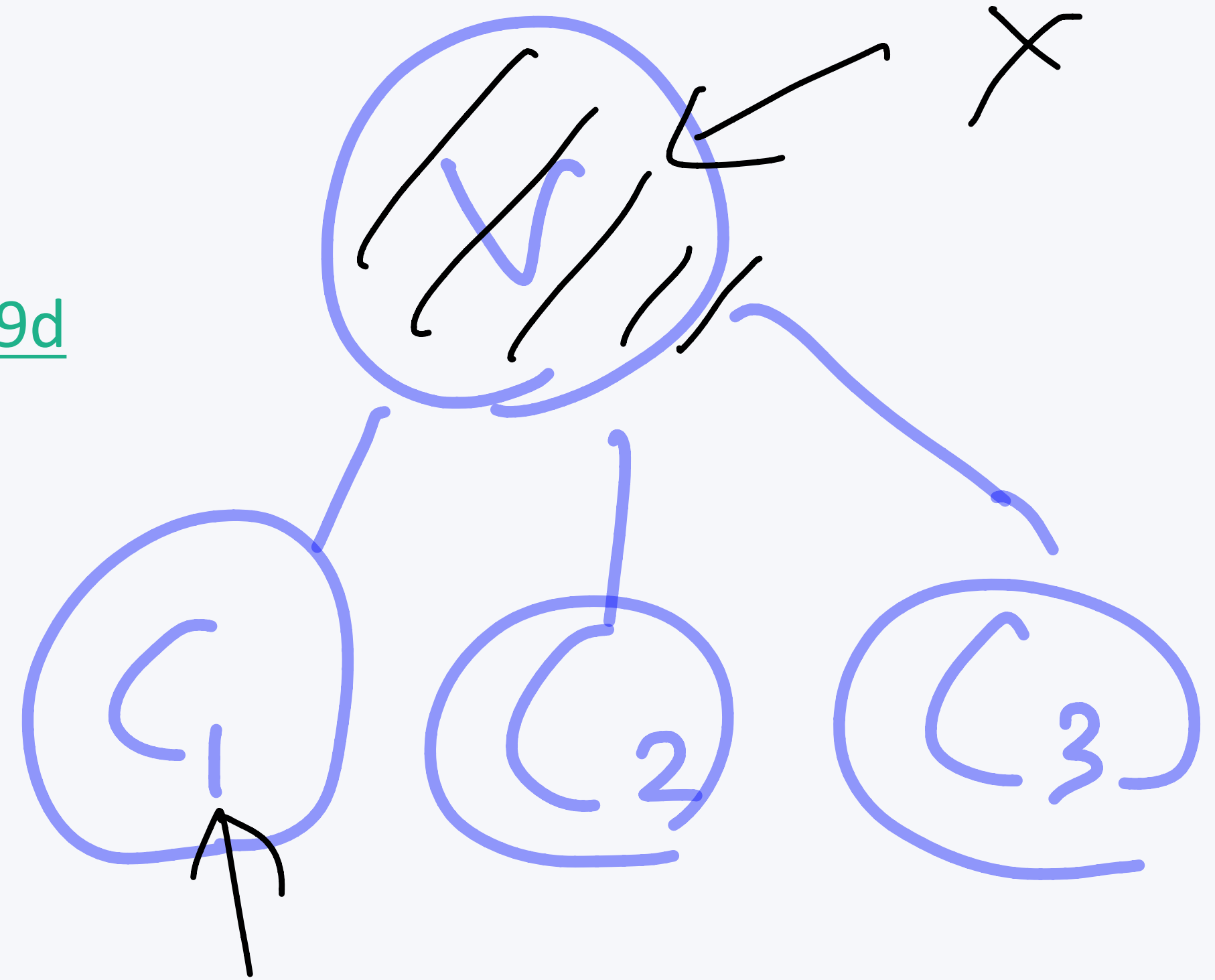
# 게시판 구멍 막기

<https://www.acmicpc.net/problem/2414>

- <https://gist.github.com/Baekjoon/d49ba57cf43ea887a79d>

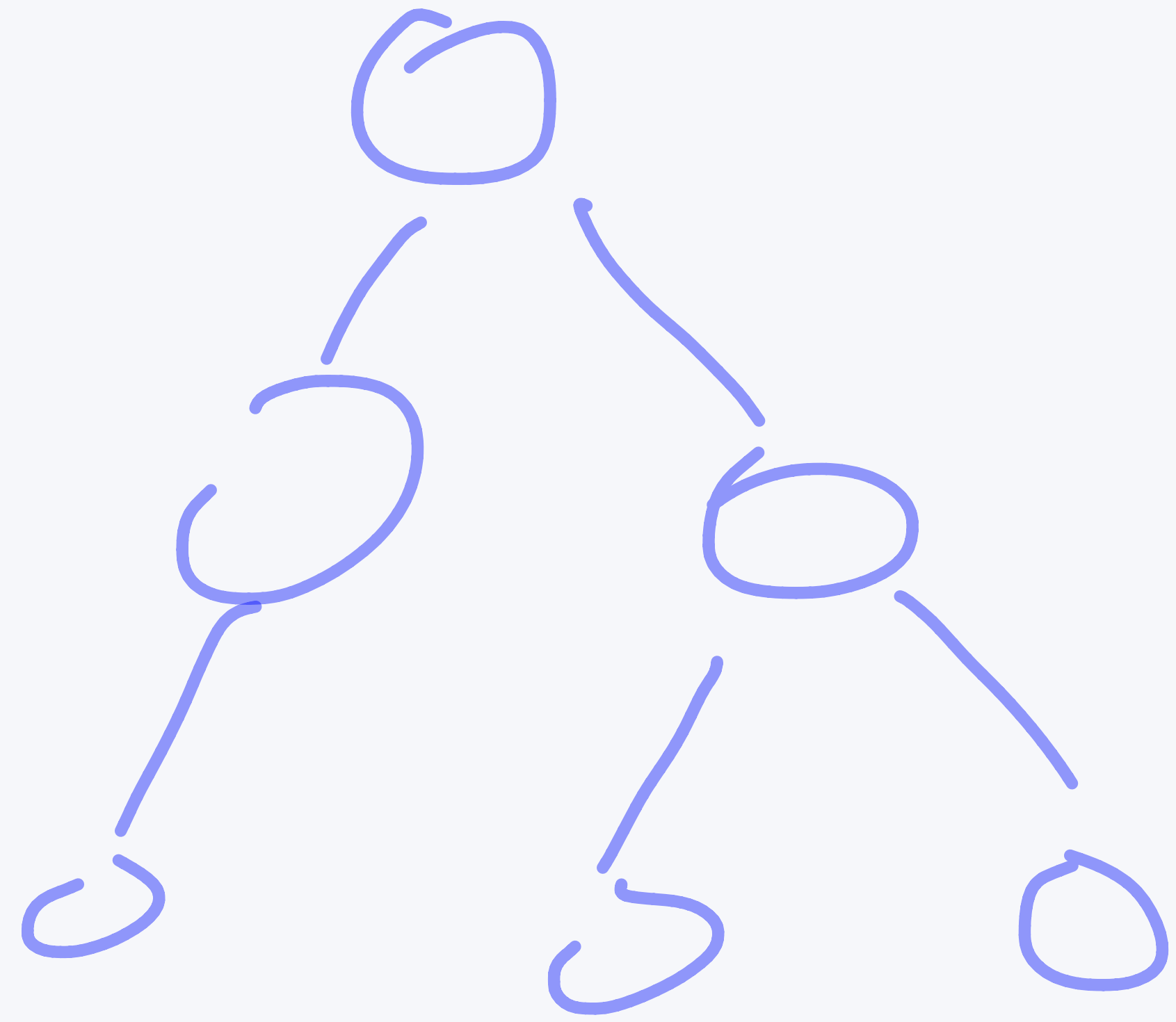
$$D[V][0] = \sum_{i=1}^n D[C_i][0]$$

$$D[V][0] = \sum \max(D[C_i][0], D[C_i][1])$$



$i-1 - \text{depth}[i] \sim i+1$

# 최대 독립 집합



Ex

DP (2213)

DP[N][C]

# 최대 독립 집합

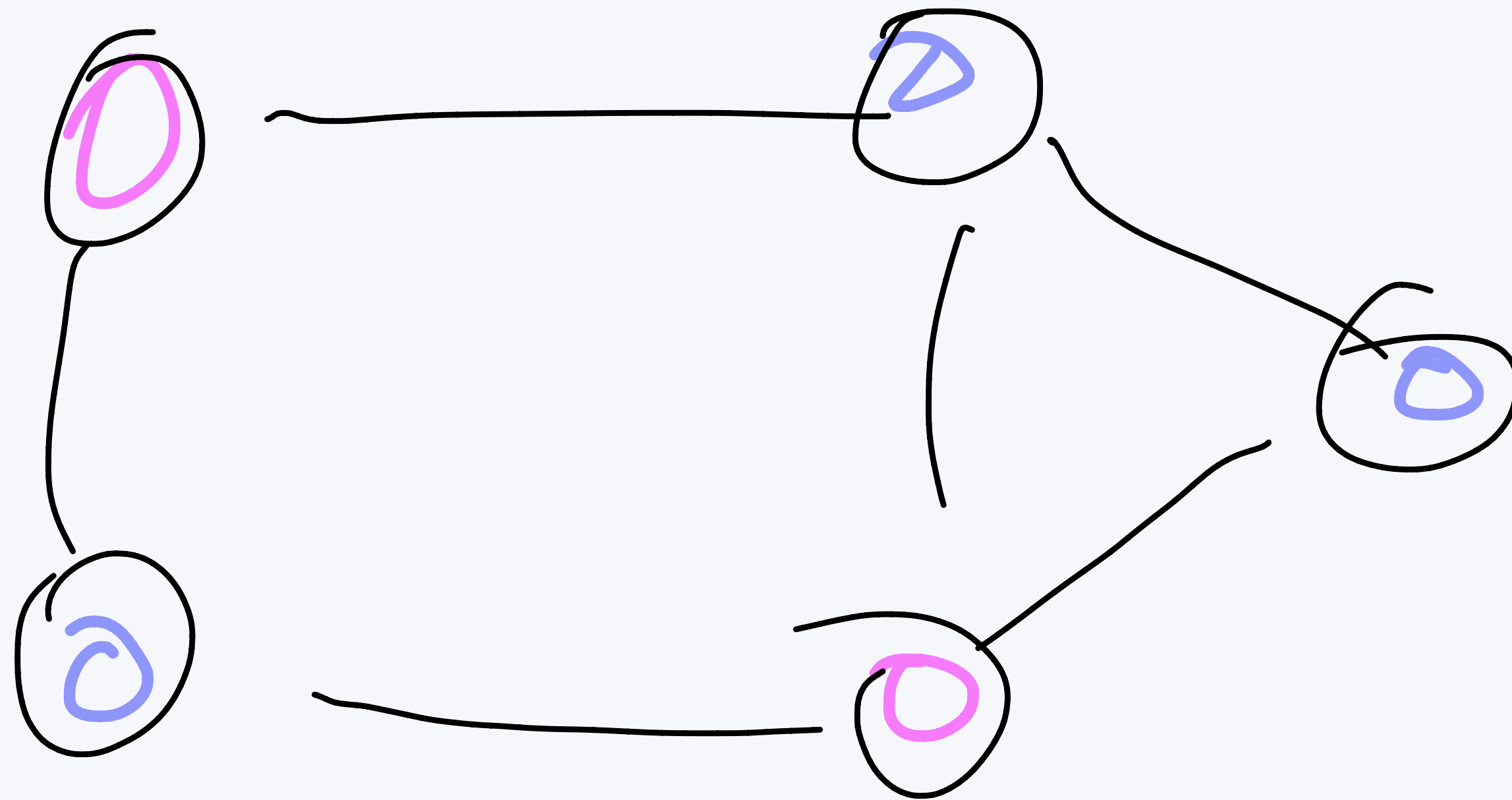
Maximum Independent Set

- 그래프 G의 정점 집합
- 집합에 포함된 모든 정점끼리를 연결하는 Edge가 없어야 함
- 이 때 최대
- 이 문제는 NP-Hard

배팅스 키바: 모든 간선의  
양 끝점 중 적어도 하나를

독립 집합: 양 끝점 모두 선택 X

(최소 배팅스 키바)<sup>c</sup> =  
최대 독립 집합



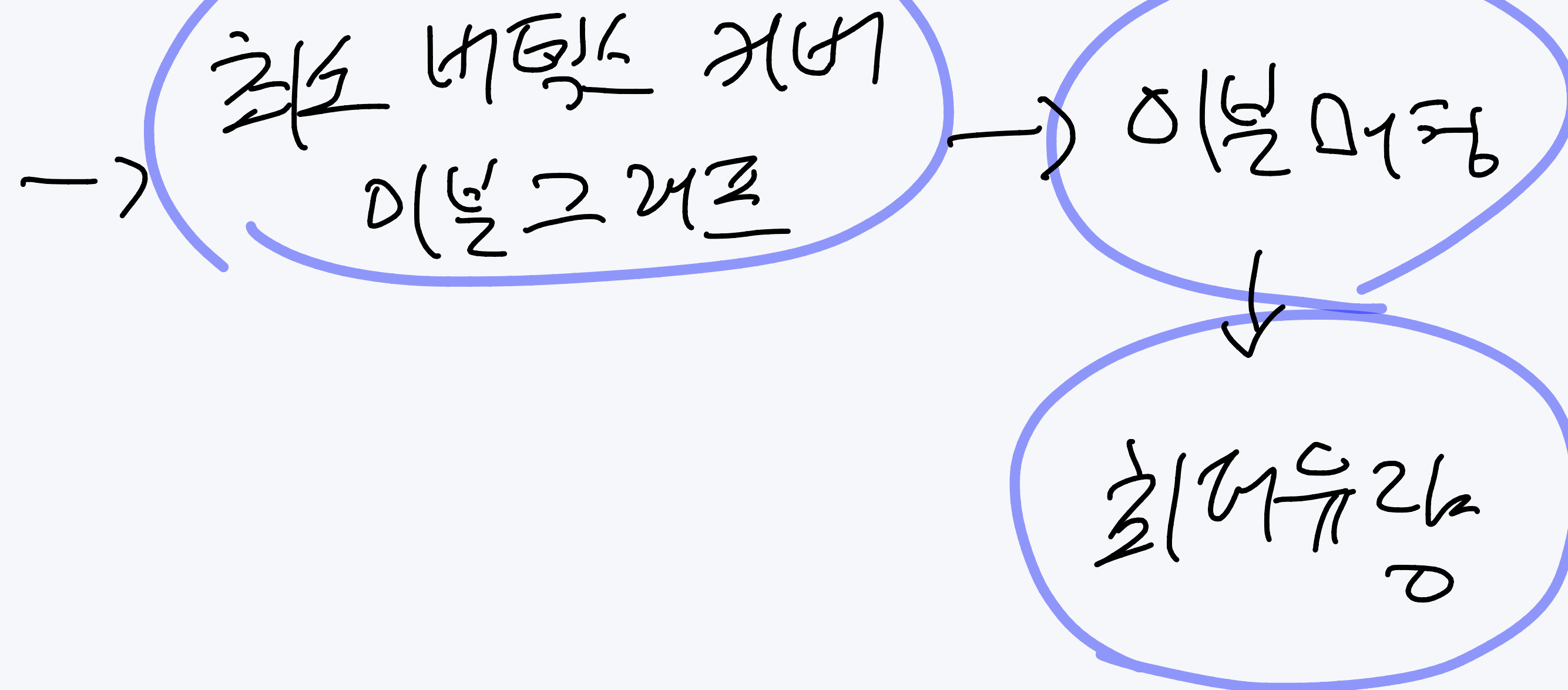
# 최대 독립 집합

102

Maximum Independent Set

- Independent Set의 Complement는 Vertex Cover다
- Maximum Independent Set의 Complement는 Minimum Vertex Cover이다
- 그래프가 이분그래프인 경우 Minimum Vertex Cover는 Maximum Flow다
- flow로 풀 수 있는 문제이다

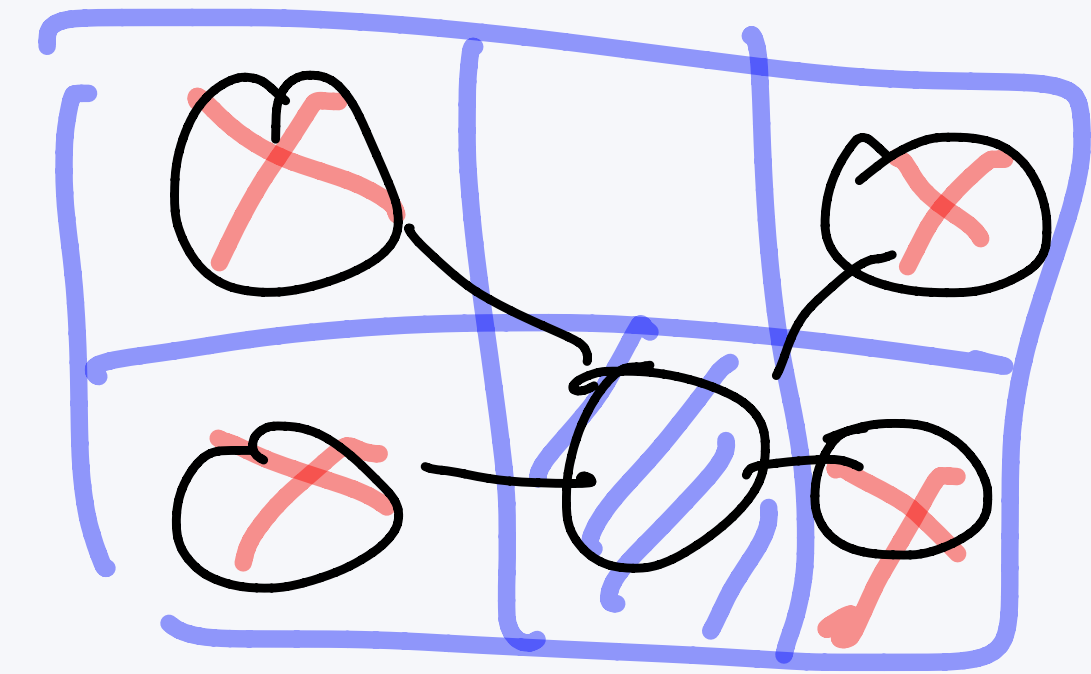
최대 독립 집합  
이분그래프



# 컨닝 2

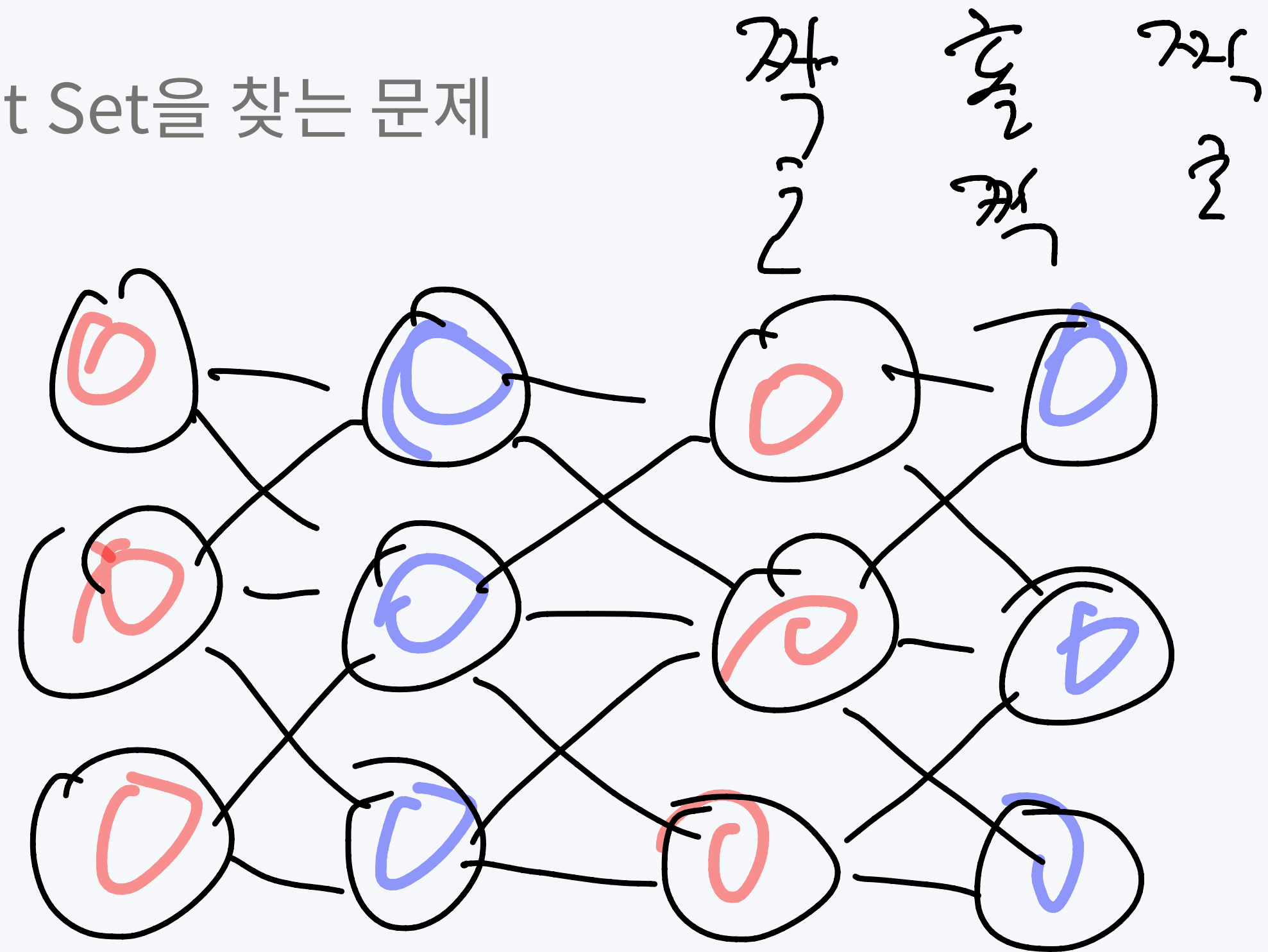
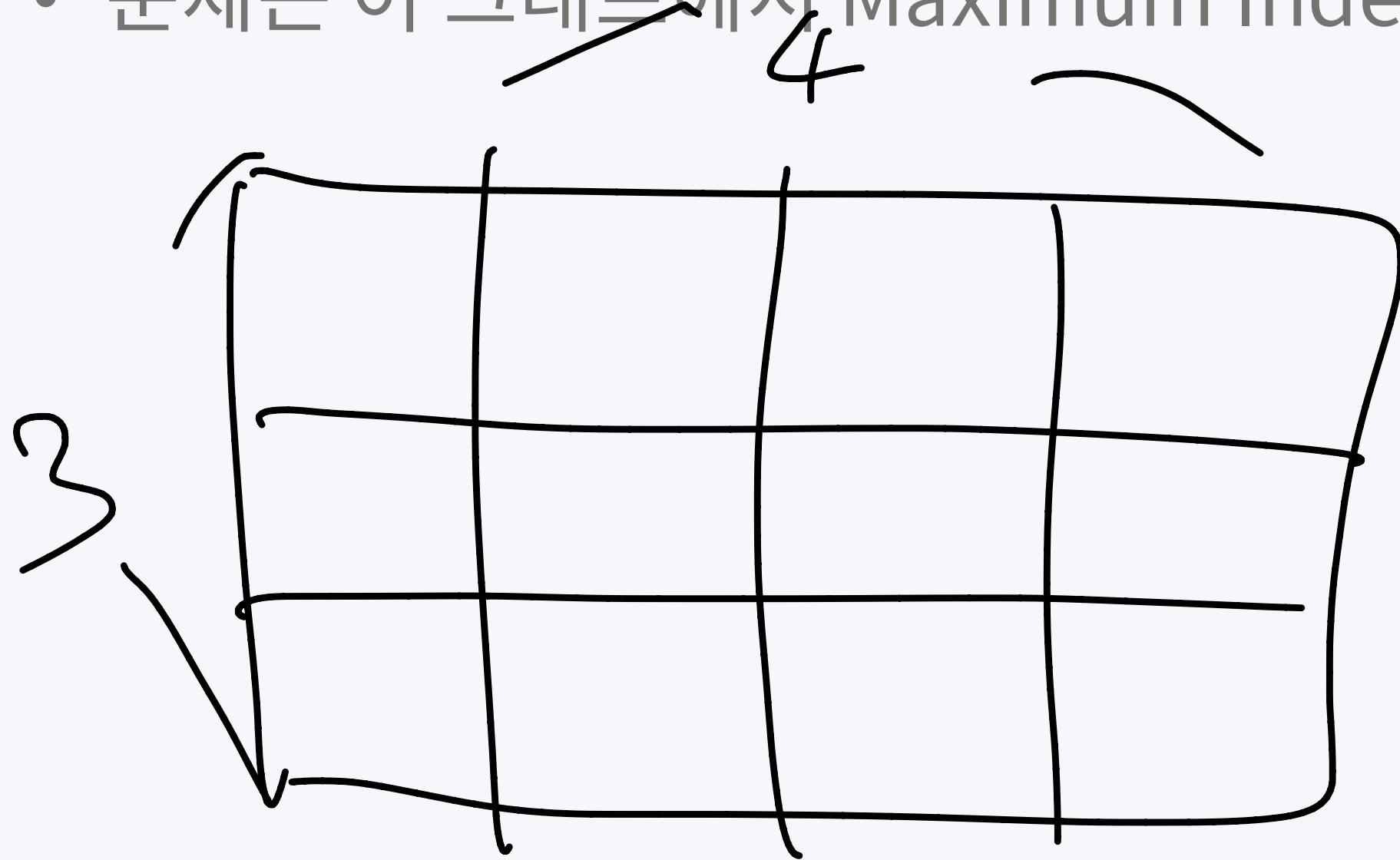
<https://www.acmicpc.net/problem/11014>

$R \times C$



103

- 각 칸을 vertex 로 생각하고, 맞을 수 없는 칸을 edge로 연결하자
- 문제는 이 그래프에서 Maximum Independent Set을 찾는 문제



이분 그래프

# 컨닝 2

<https://www.acmicpc.net/problem/11014>

104

- column 의 홀 짝을 기준으로 왼쪽과 오른쪽을 나눌 수 있다.

- <https://gist.github.com/Baekjoon/e27c28b4aac30a4ba527>

