네트워크 플로우

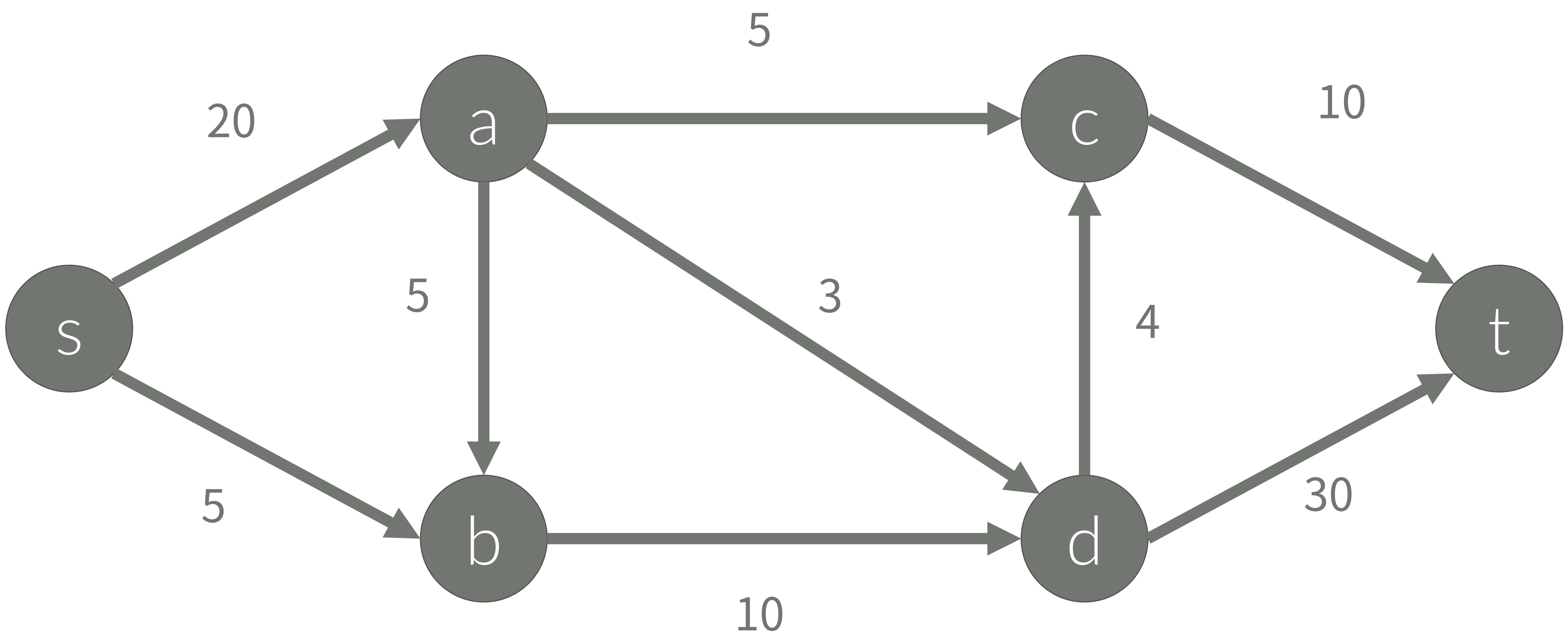
최백준 choi@startlink.io

최대 유량

최대 유량

Maximum Flow

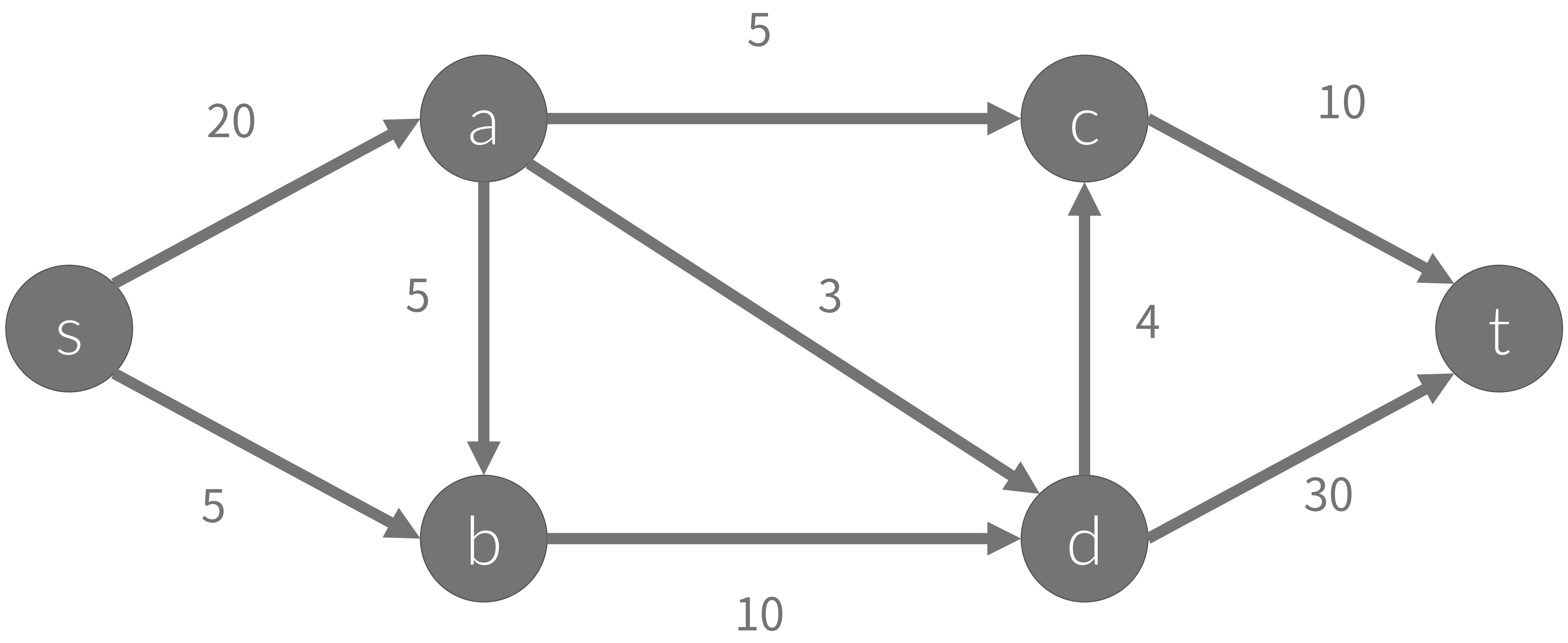
- 각 간선이 나타내는 것은 흐를 수 있는 양
- s에서 t로 최대 얼마나 흐를 수 있는가?



최대 유량

Maximum Flow

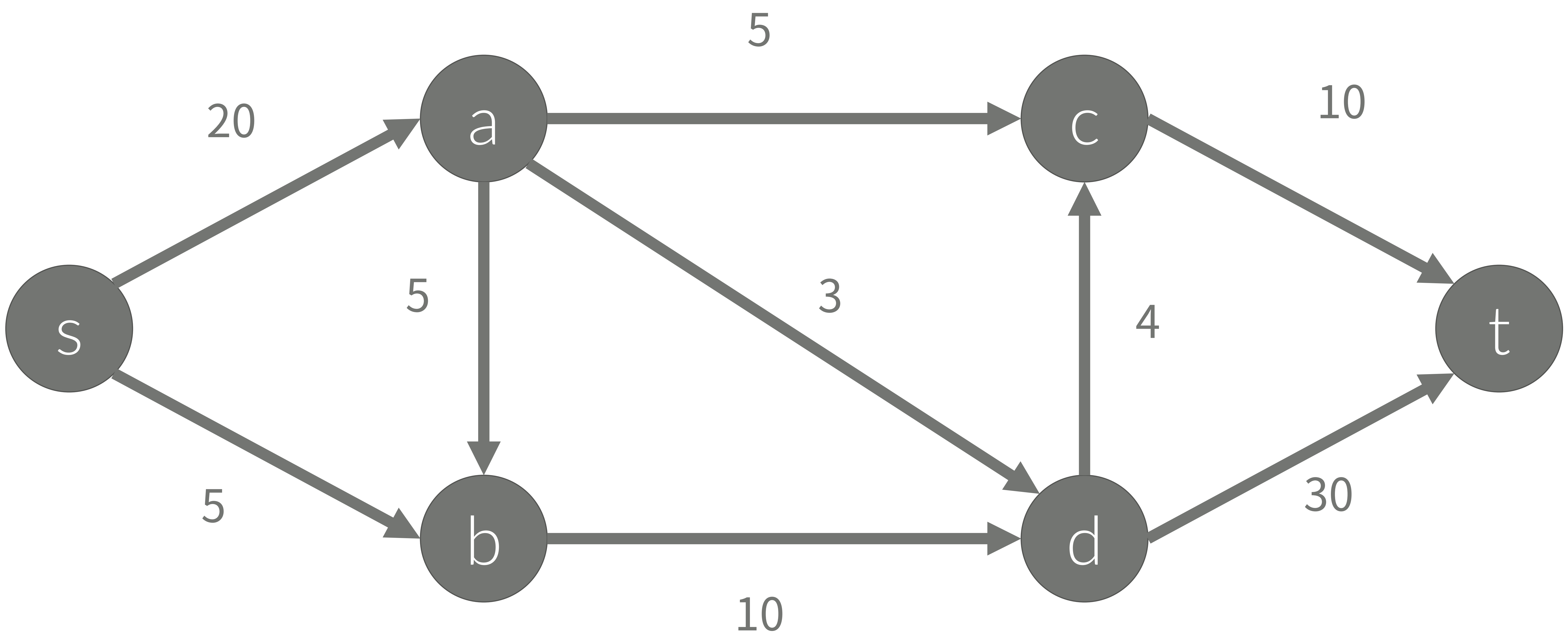
- s → a로 최대 20만큼만 흐를 수 있다.
- a → c로 최대 5만큼 흐를 수 있다.



최대 유량

Maximum Flow

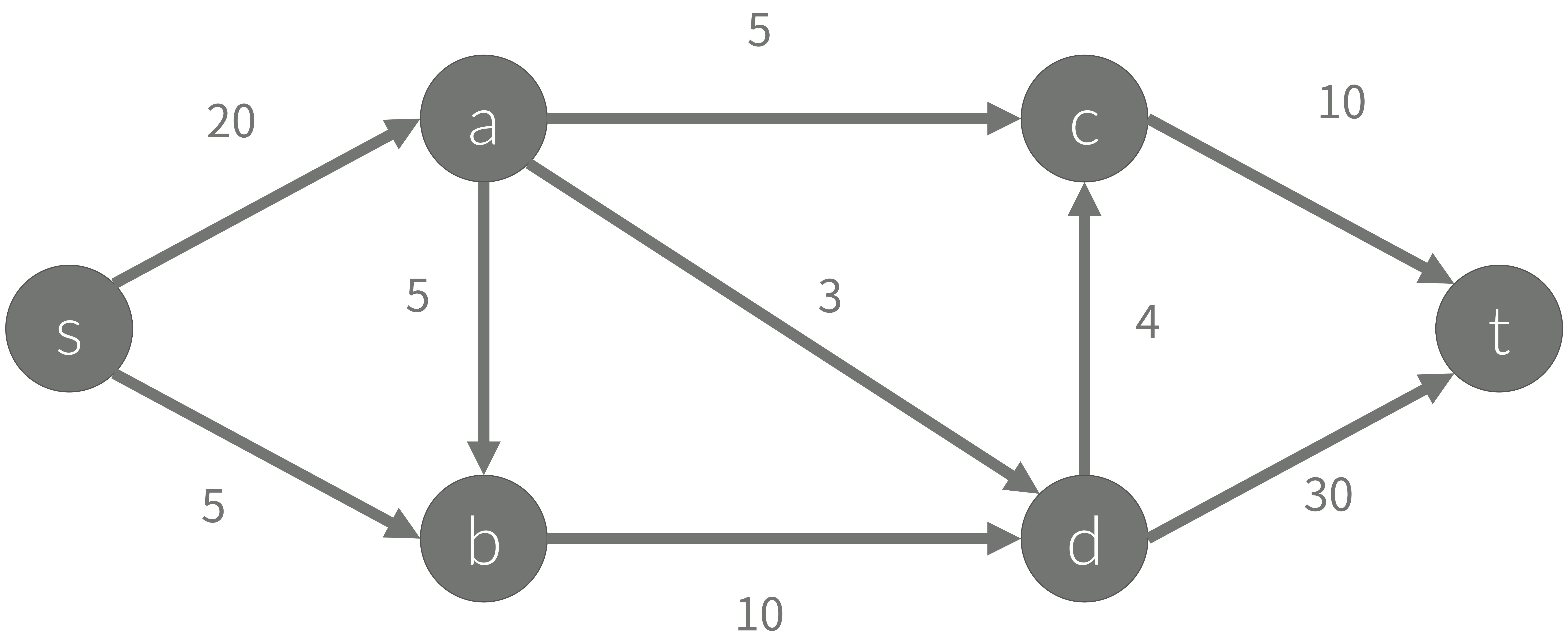
- s → a로 20을 보냈다고 하더라도
- a → c, a → d, a → b는 5+3+5 = 13이기 때문에, 13 이상을 보낼 수 없다



최대 유량

Maximum Flow

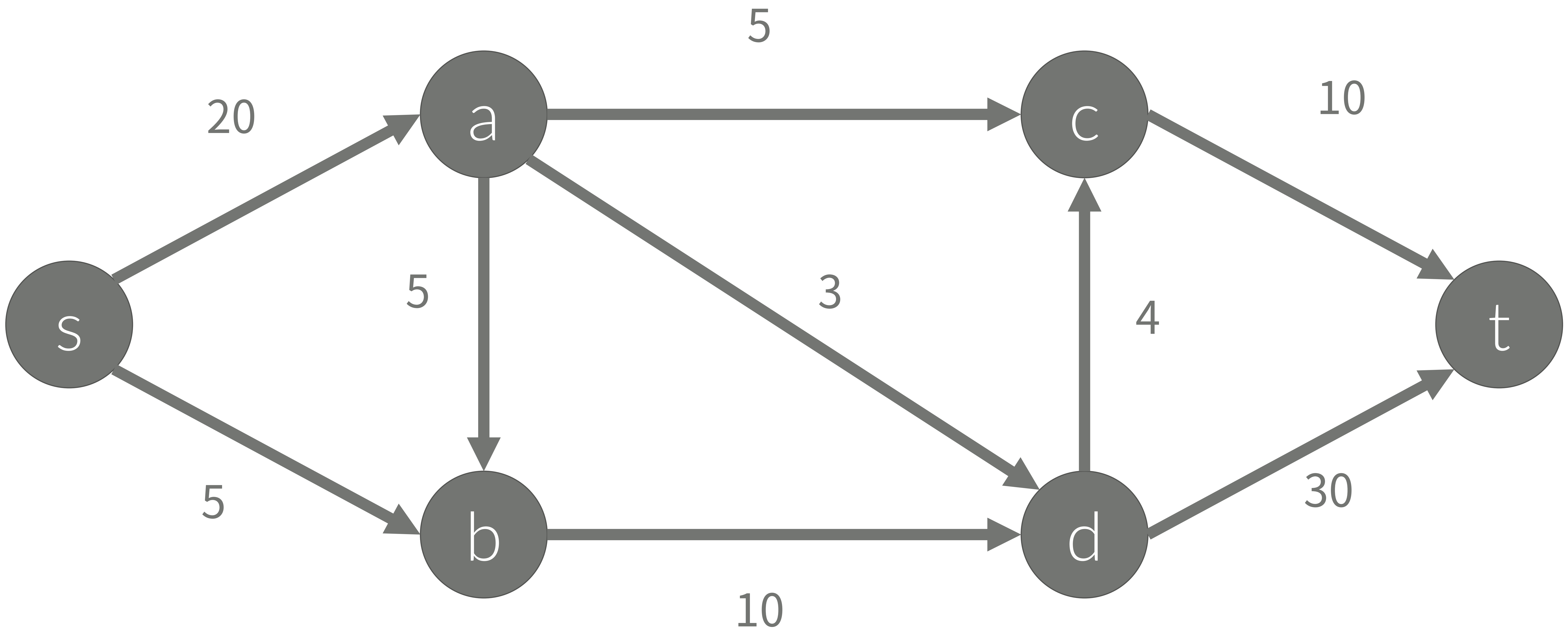
- s에서 t로 최대 얼마나 보낼 수 있는지를 구하는 문제



최대 유량

Maximum Flow

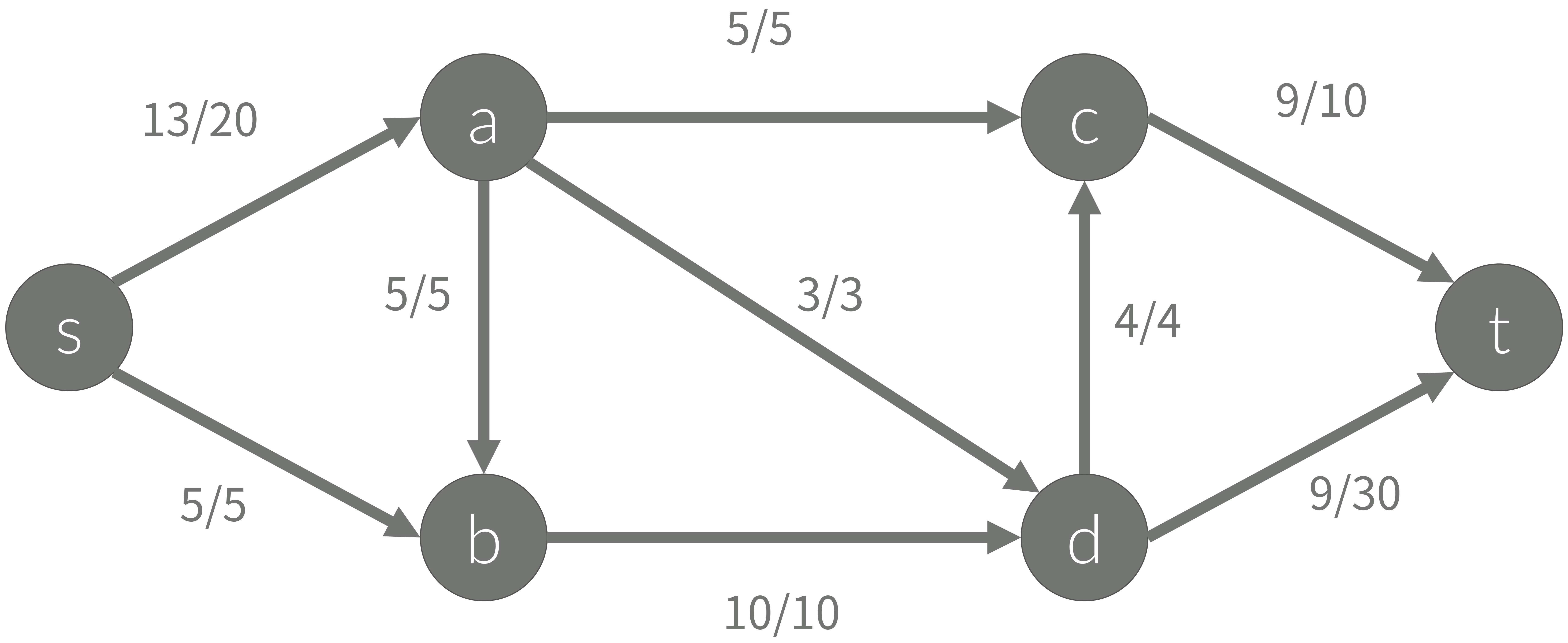
- 간선에 나타나있는 것: capacity
- 실제 그 간선을 따라서 흐른 양: flow



최대 유량

Maximum Flow

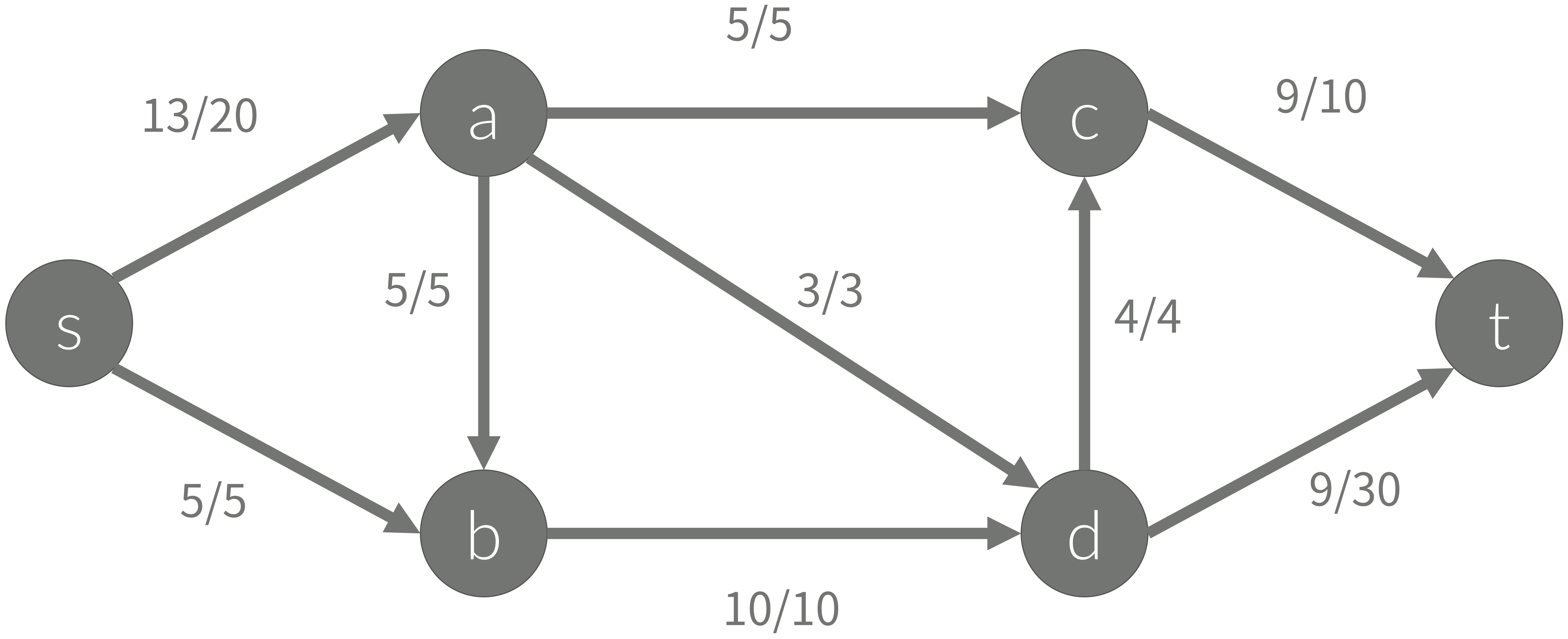
- flow/capacity



최대 유량

Maximum Flow

- 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 → v 간선의 용량: c(u, v)
- u → v 간선의 흐름 양: f(u, v)

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 x를 보냈으면, $v \rightarrow u$로는 $-x$를 보낸다고 한다
- Flow conservation
  - $\Sigma_{v \in V} f(u,v) = 0$
  - u와 연결된 모든 간선의 흐름양의 합은 0이다
  - Skew symmetry에 의해서 음수를 저장했기 때문

최대 유량

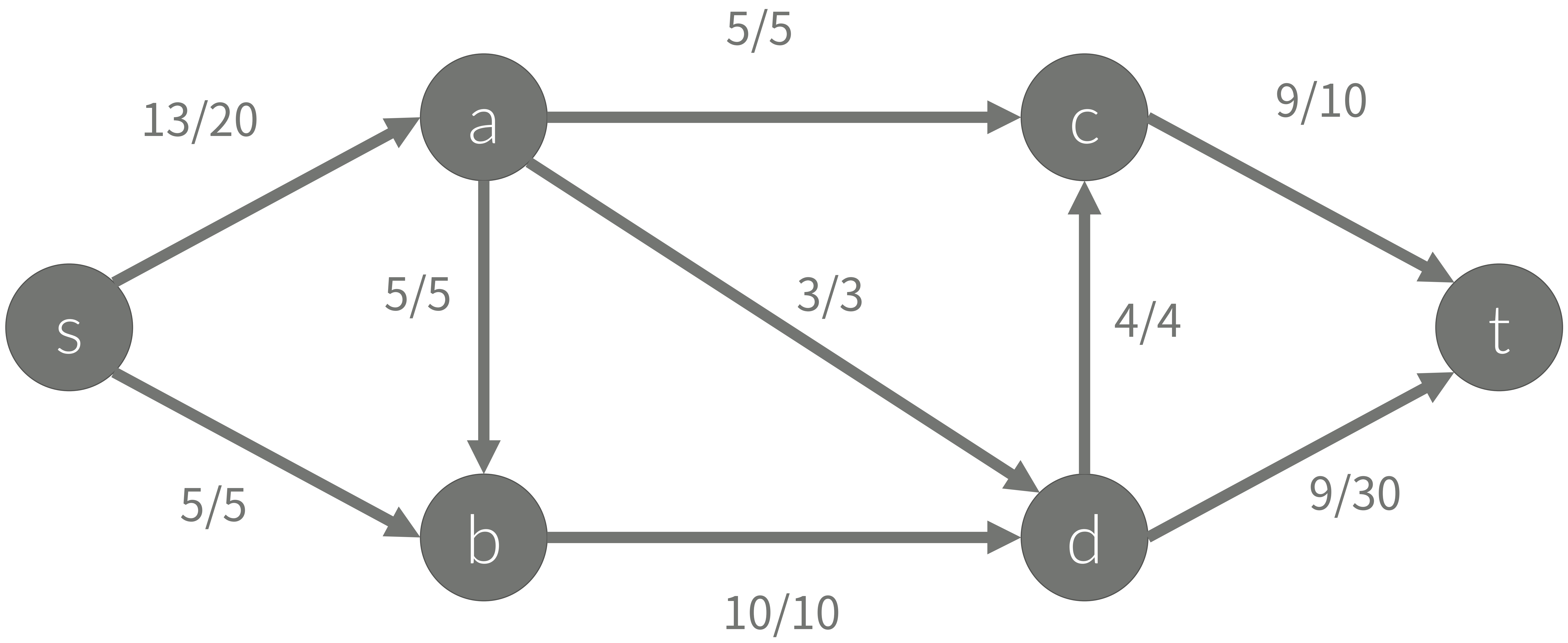
Maximum Flow

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

Residual Capacity

Maximum Flow

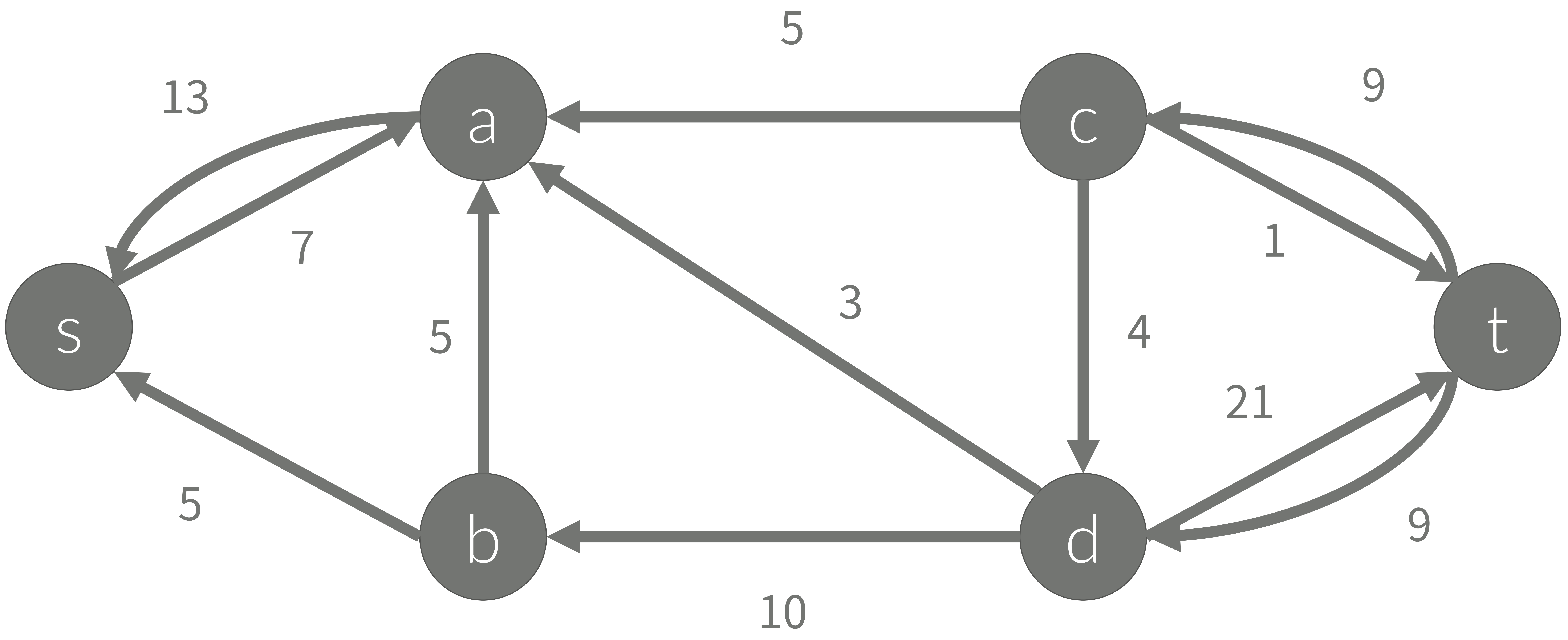
- cf(u, v) = c(u, v) – f(u v)
- Available Capacity



Residual Capacity

Maximum Flow

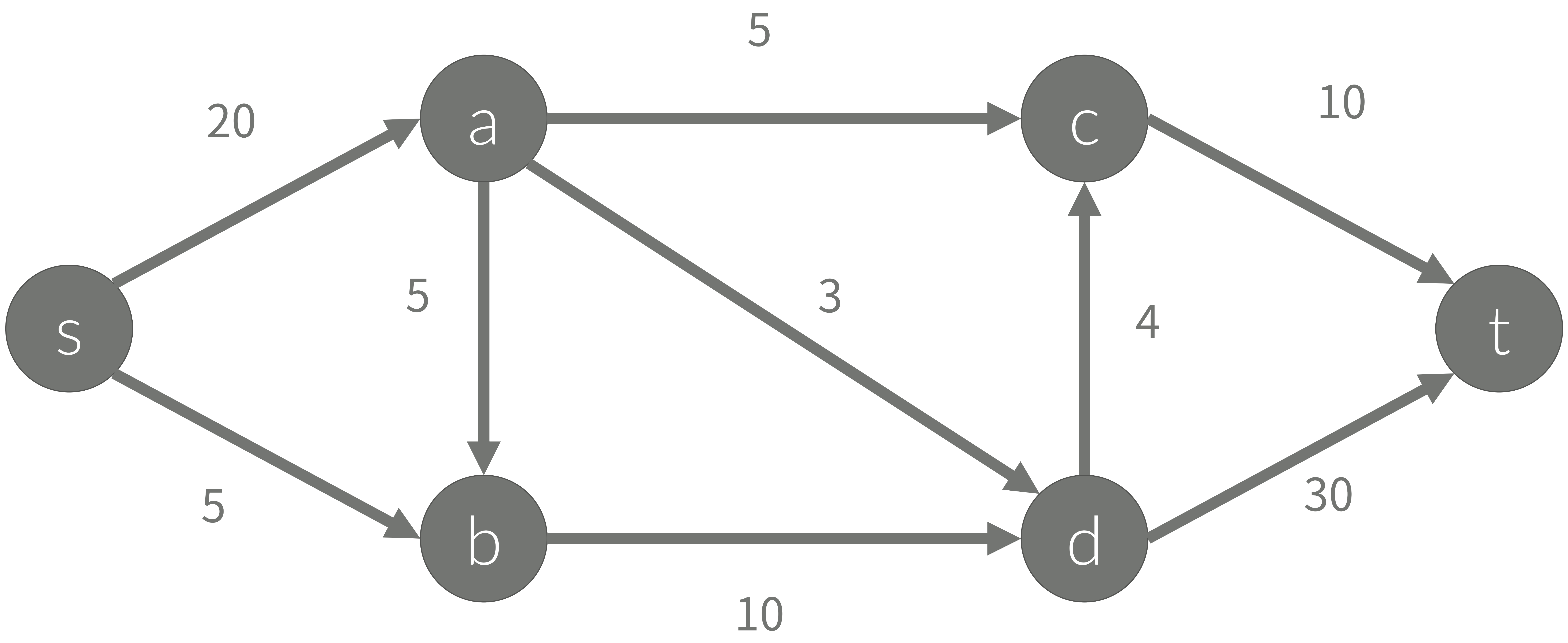
- cf(u, v) = c(u, v) – f(u v)



Augmenting Path

Maximum Flow

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



Ford-Fulkerson

Ford-Fulkerson

Maximum Flow

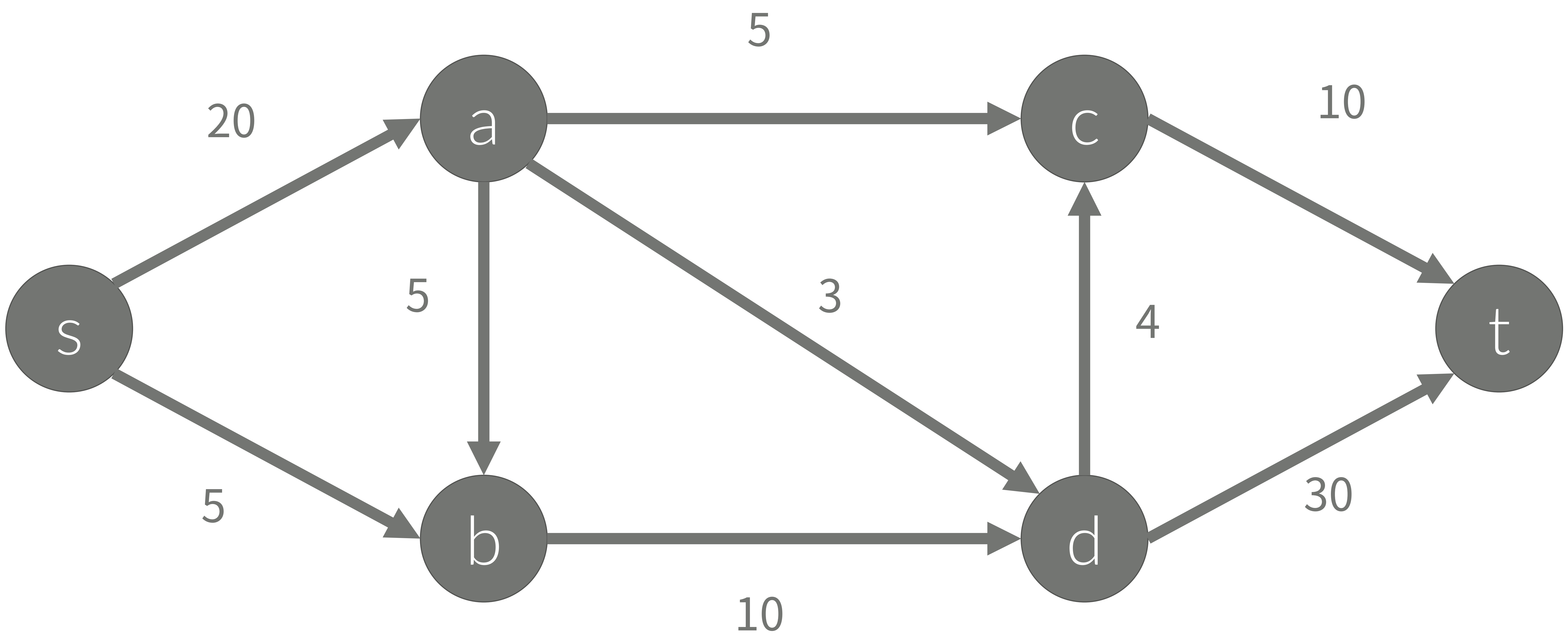
- 최대 유량을 구하는 알고리즘

1. Augmenting Path를 DFS를 이용해서 구한다.
2. m = Augmenting Paht 상에서의 최소값을 구한다
3. (u_i, u_{i+1}) 방향의 Residual Capacity에서 m을 뺀다
4. (u_{i+1}, u_i) 방향의 Residual Capacity에 m을 더한다.
5. 위의 과정을 Augmenting Path를 못 구할때 까지 계속 한다

Ford-Fulkerson

Maximum Flow

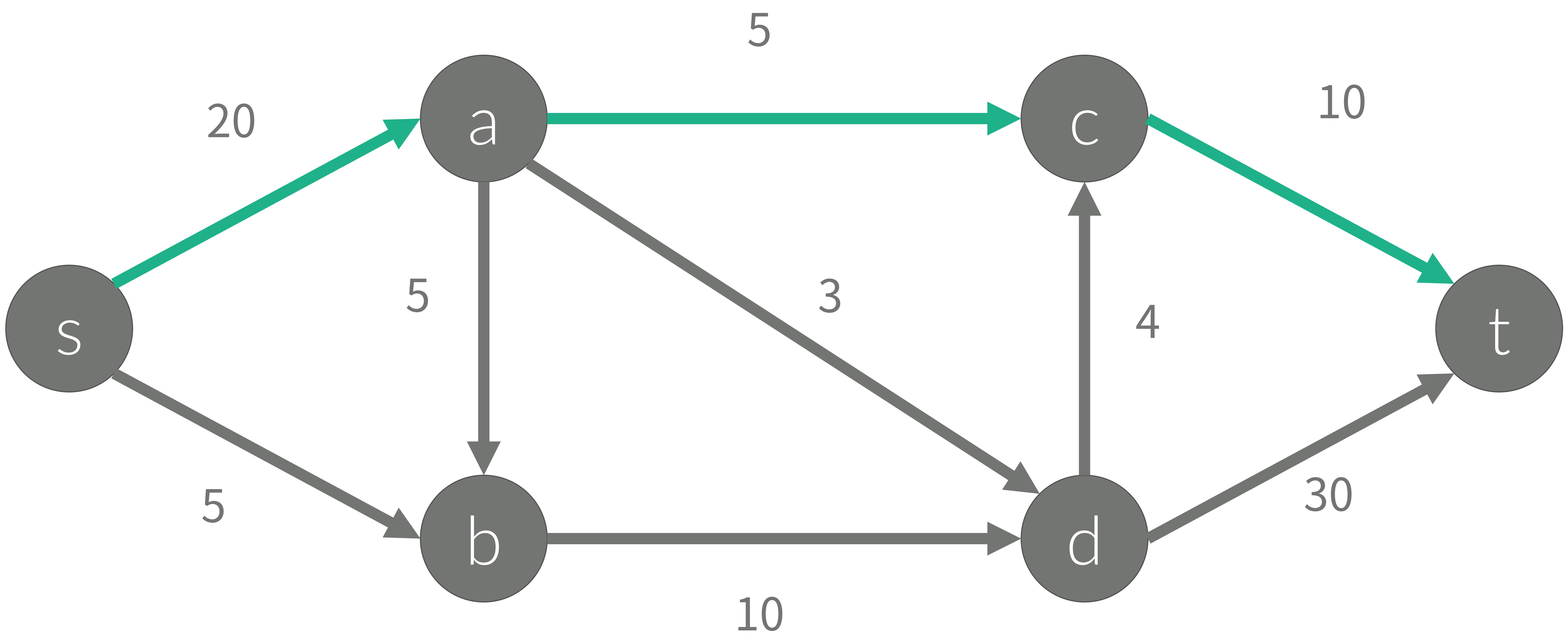
- 그래프



Ford-Fulkerson

Maximum Flow

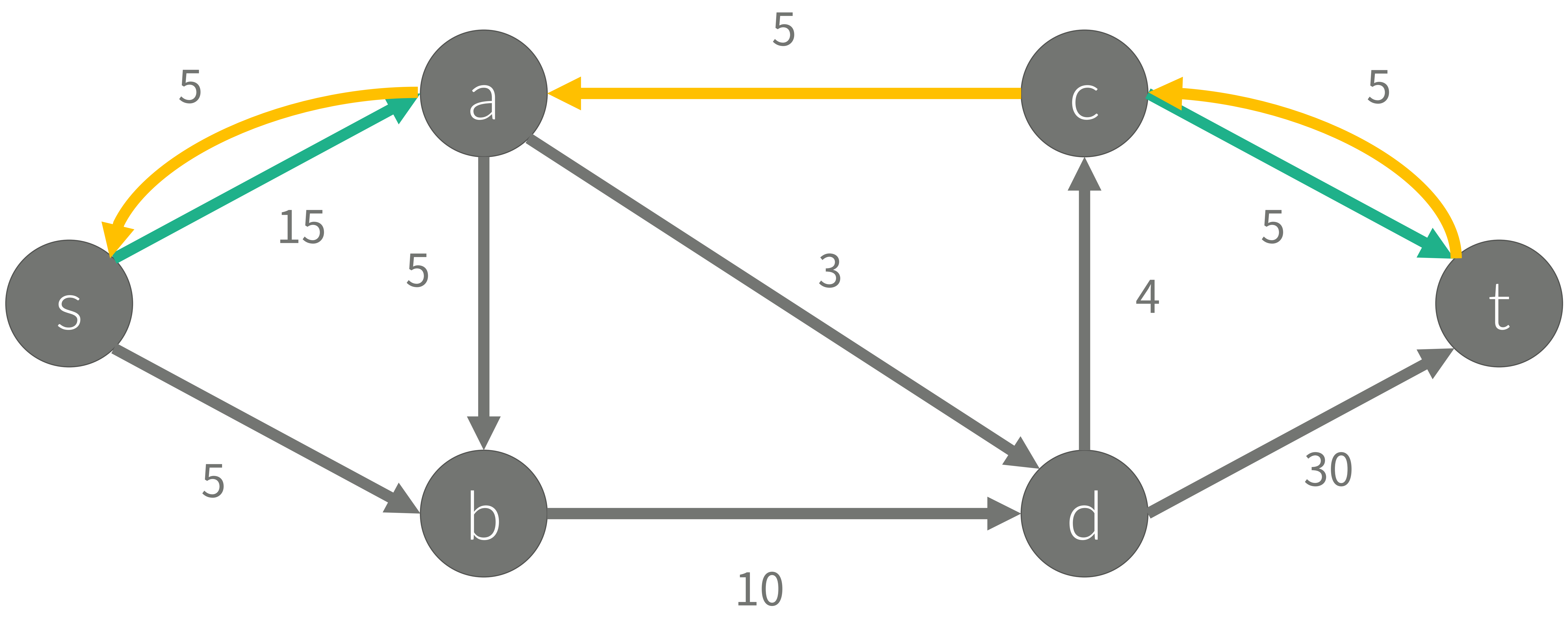
- Augmenting Path: s → a → c → t
- m = 5



Ford-Fulkerson

Maximum Flow

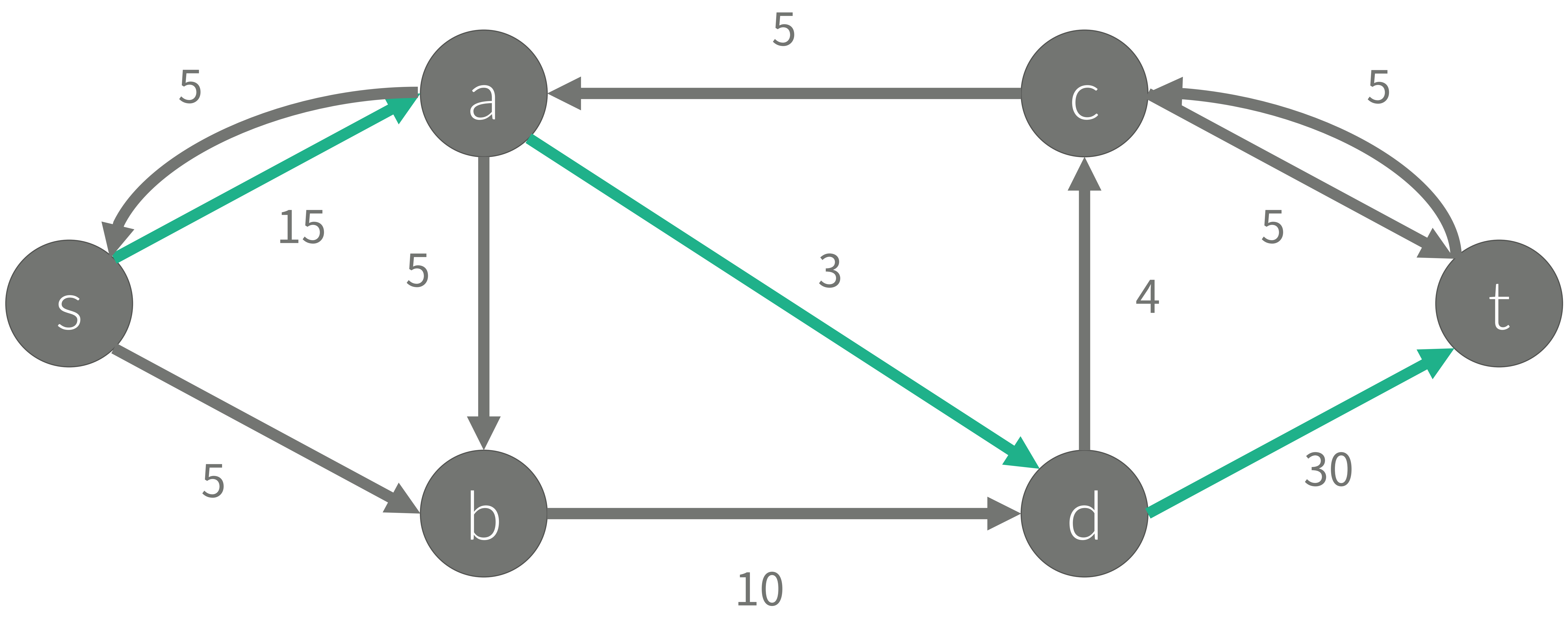
- Augmenting Path: s → a → c → t
- m = 5



Ford-Fulkerson

Maximum Flow

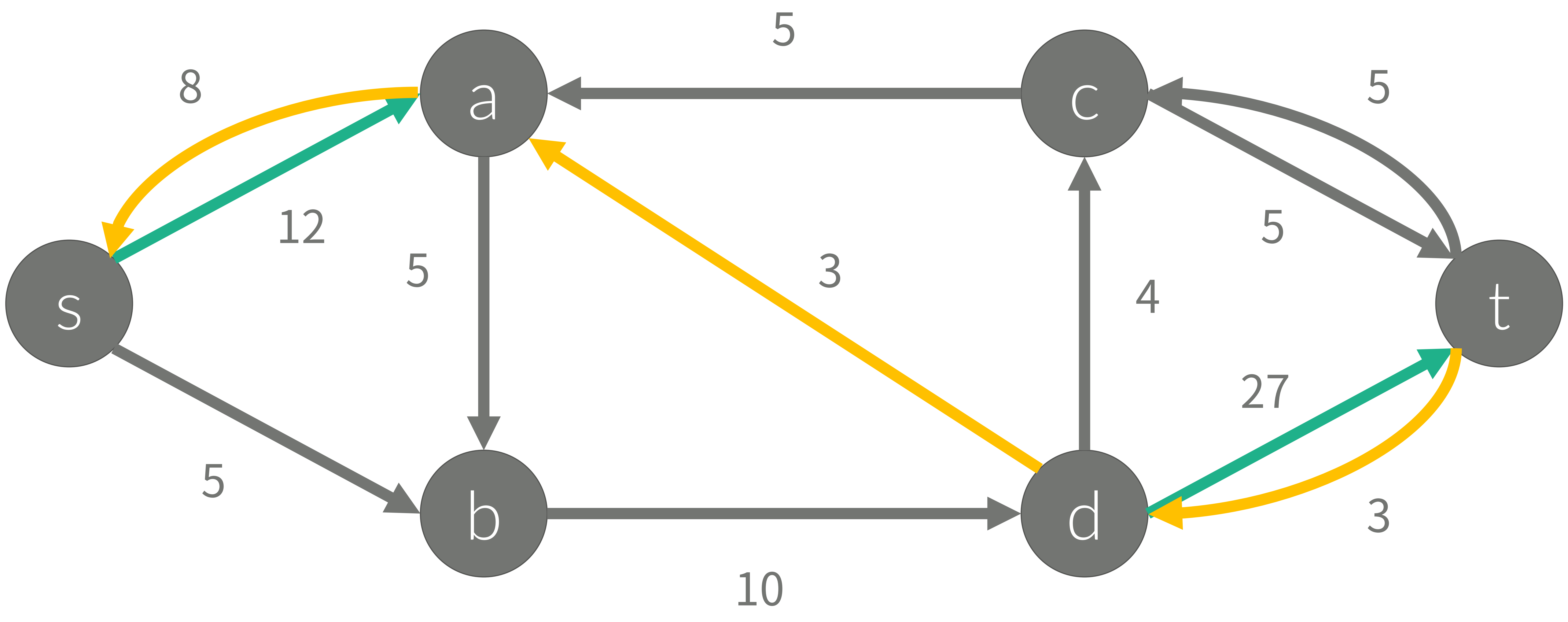
- Augmenting Path: s → a → d → t
- m = 3



Ford-Fulkerson

Maximum Flow

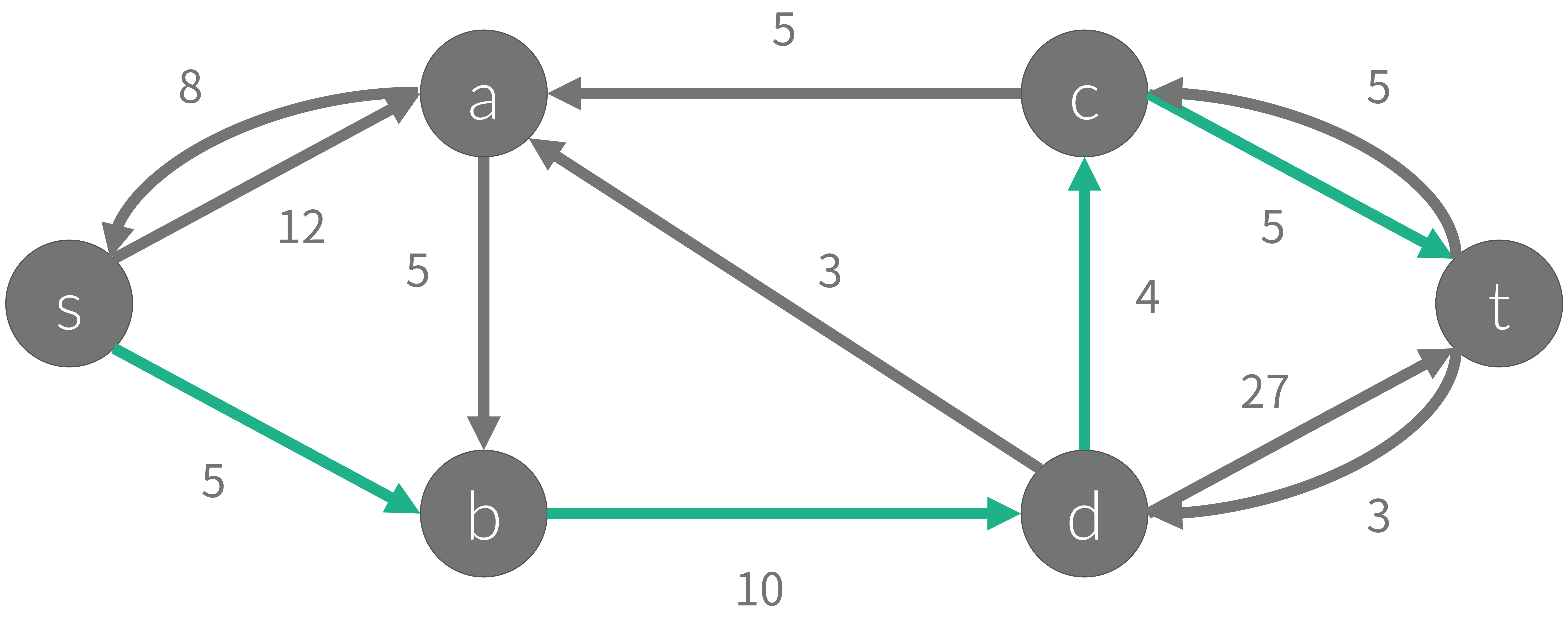
- Augmenting Path: s → a → d → t
- m = 3



Ford-Fulkerson

Maximum Flow

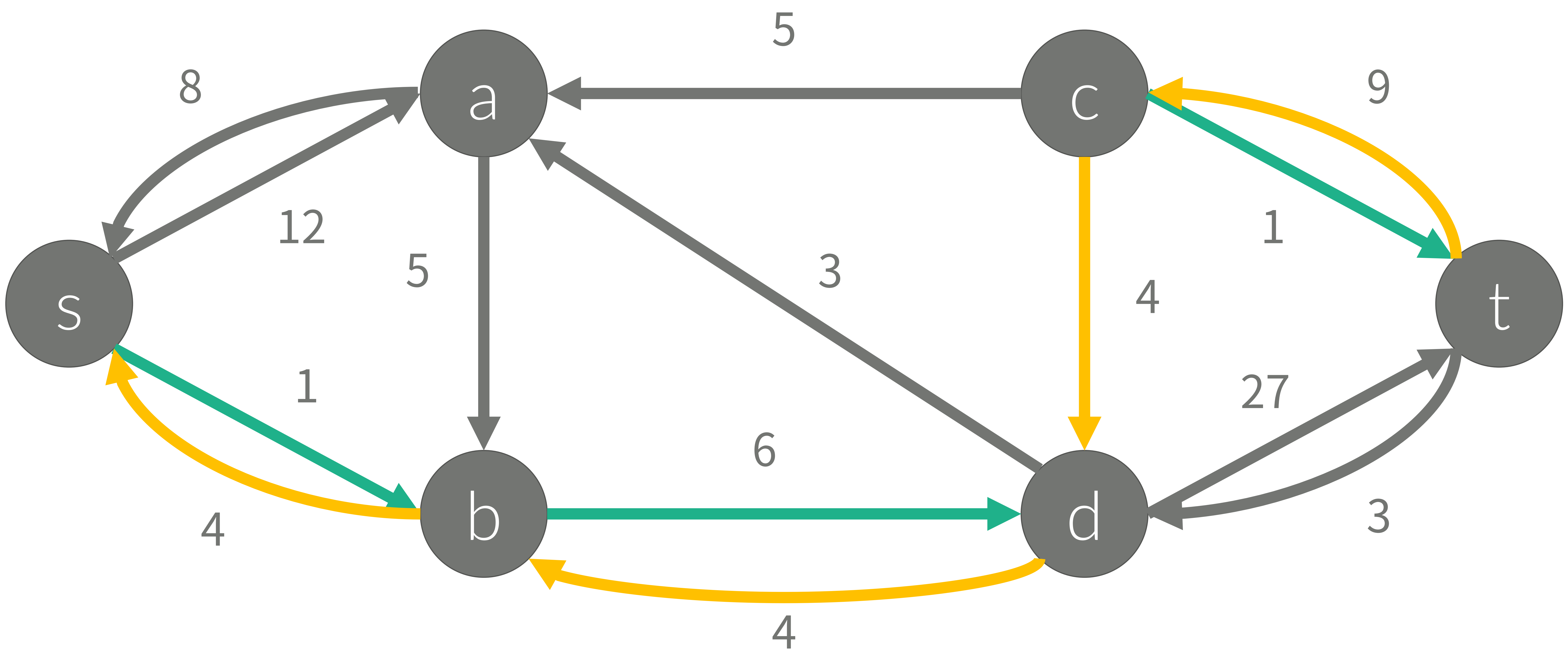
- Augmenting Path: s → b → d → c → t
- m = 4



Ford-Fulkerson

Maximum Flow

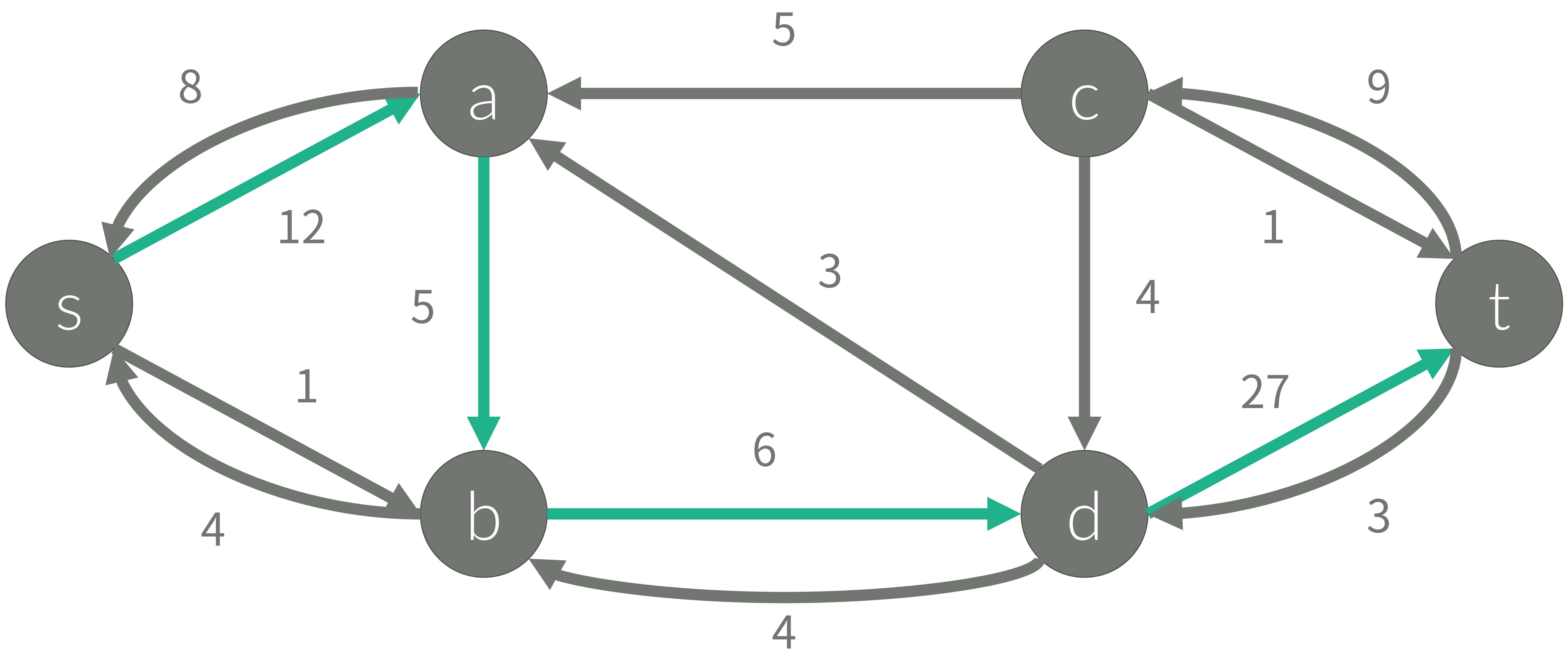
- Augmenting Path: s → b → d → c → t
- m = 4



Ford-Fulkerson

Maximum Flow

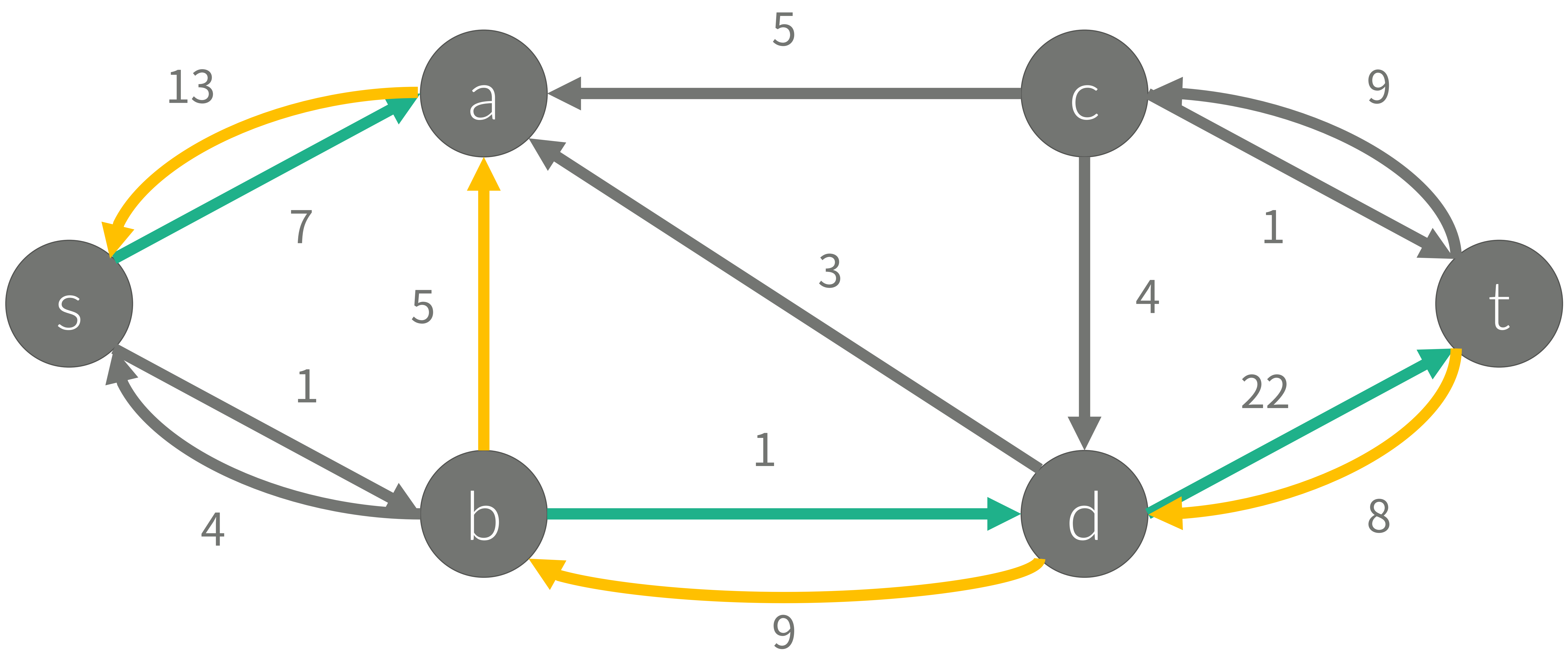
- Augmenting Path: s → a → b → d → t
- m = 5



Ford-Fulkerson

Maximum Flow

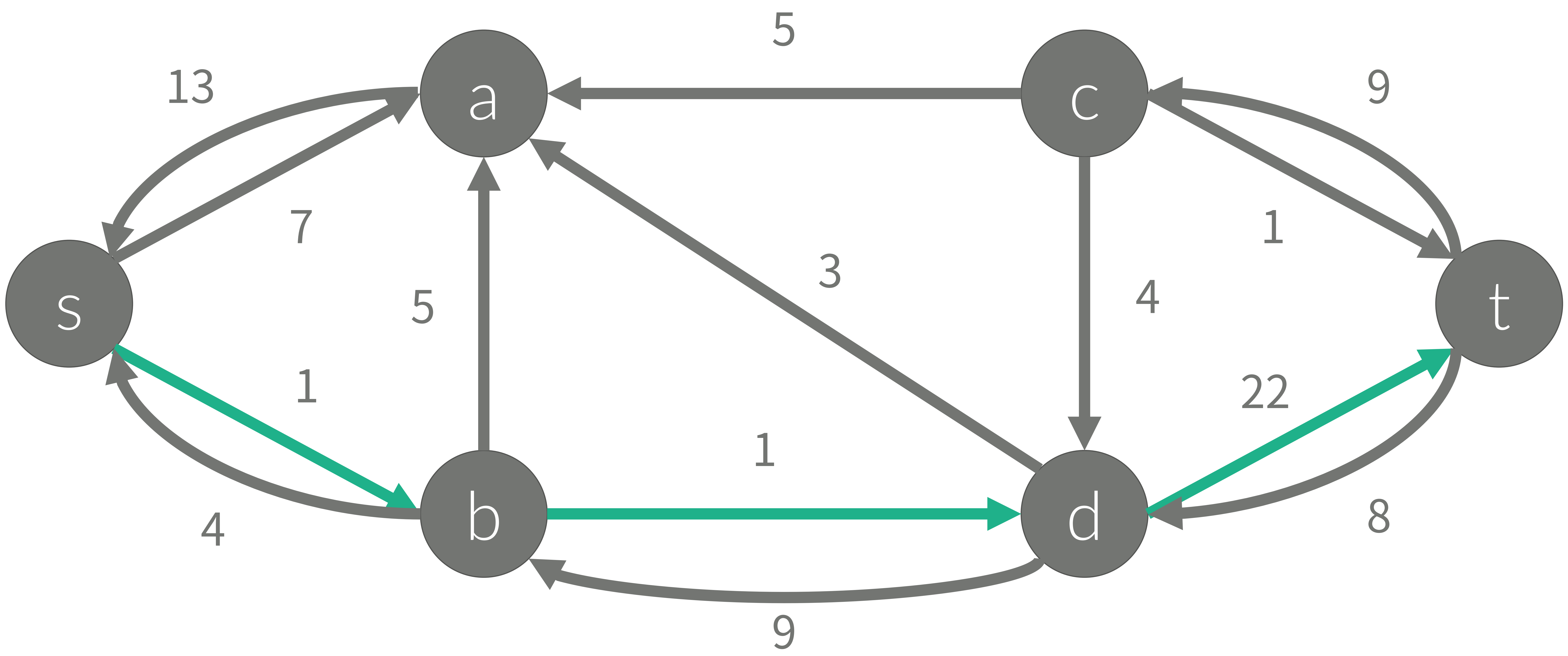
- Augmenting Path: s → a → b → d → t
- m = 5



Ford-Fulkerson

Maximum Flow

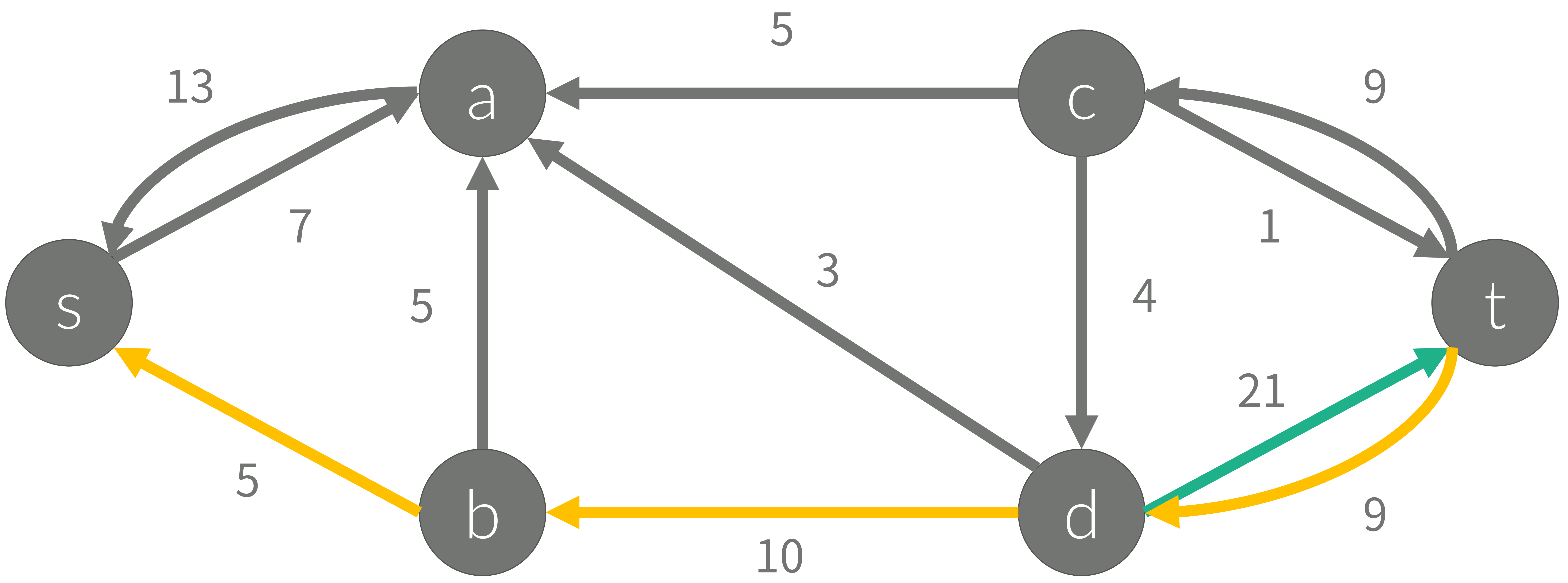
- Augmenting Path: s → b → d → t
- m = 1



Ford-Fulkerson

Maximum Flow

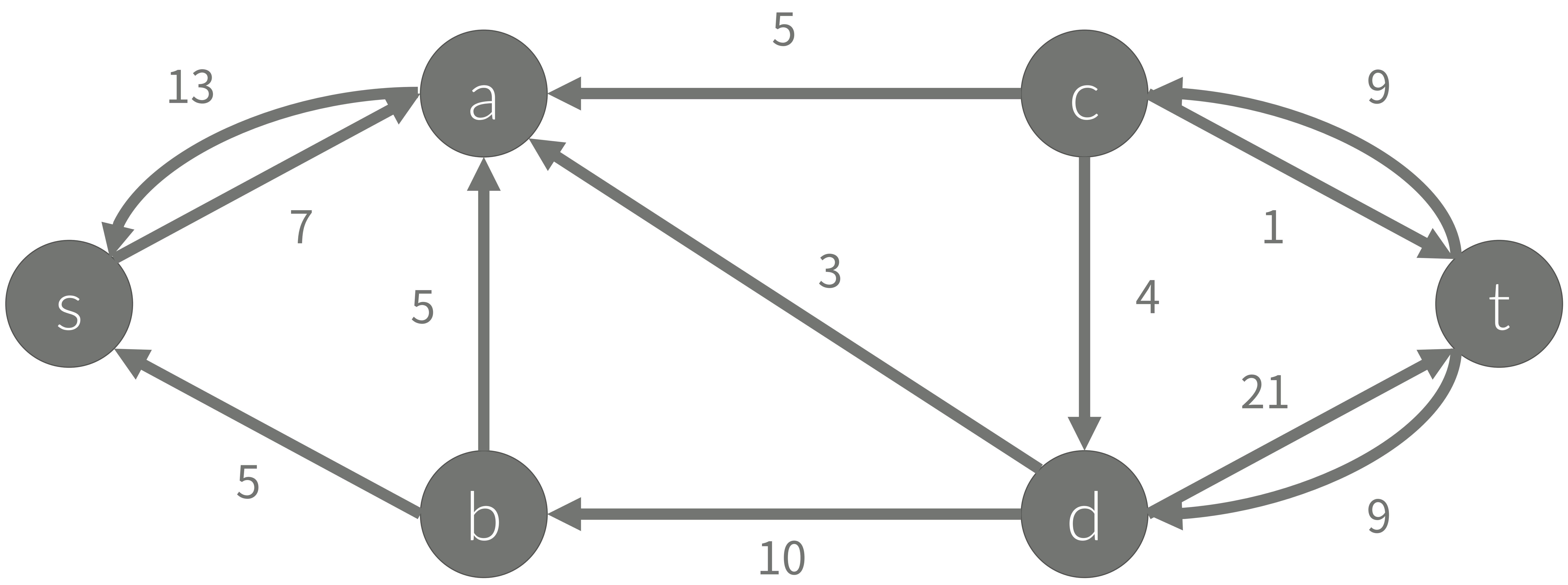
- Augmenting Path: s → b → d → t
- m = 1



Ford-Fulkerson

Maximum Flow

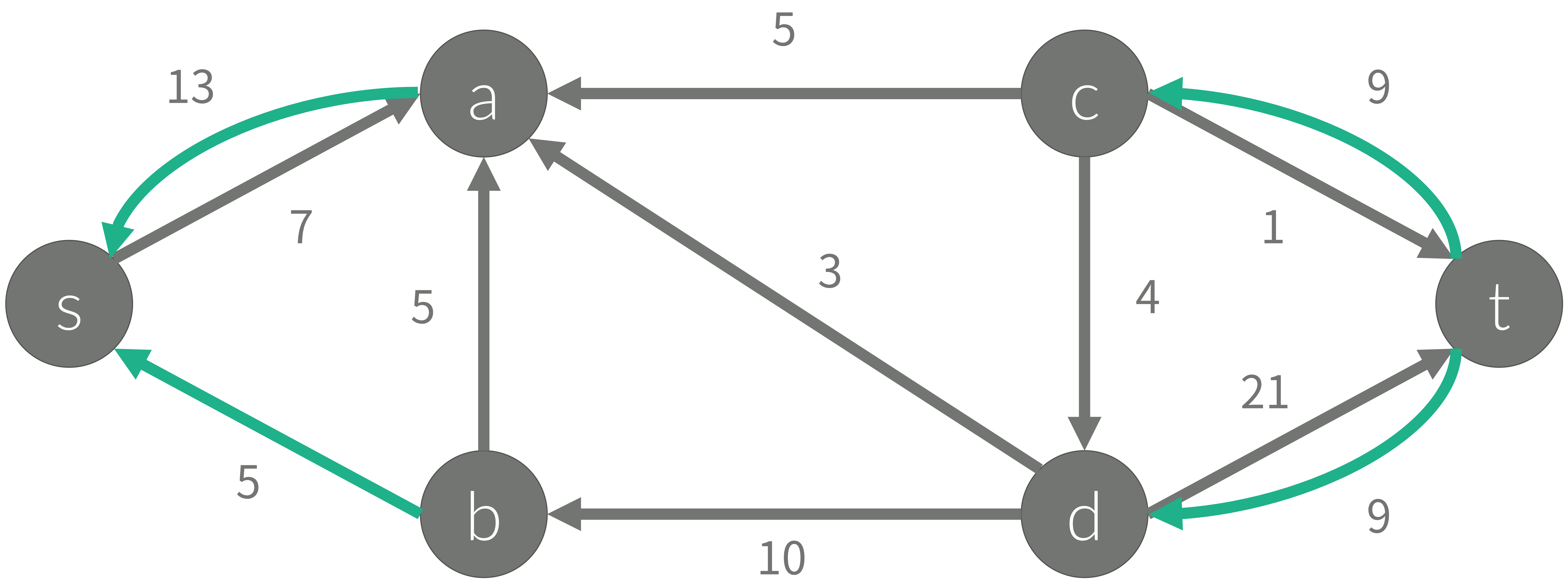
- 더 이상 Augmenting path가 없다



Ford-Fulkerson

Maximum Flow

- Maximum Flow: 18



Ford-Fulkerson

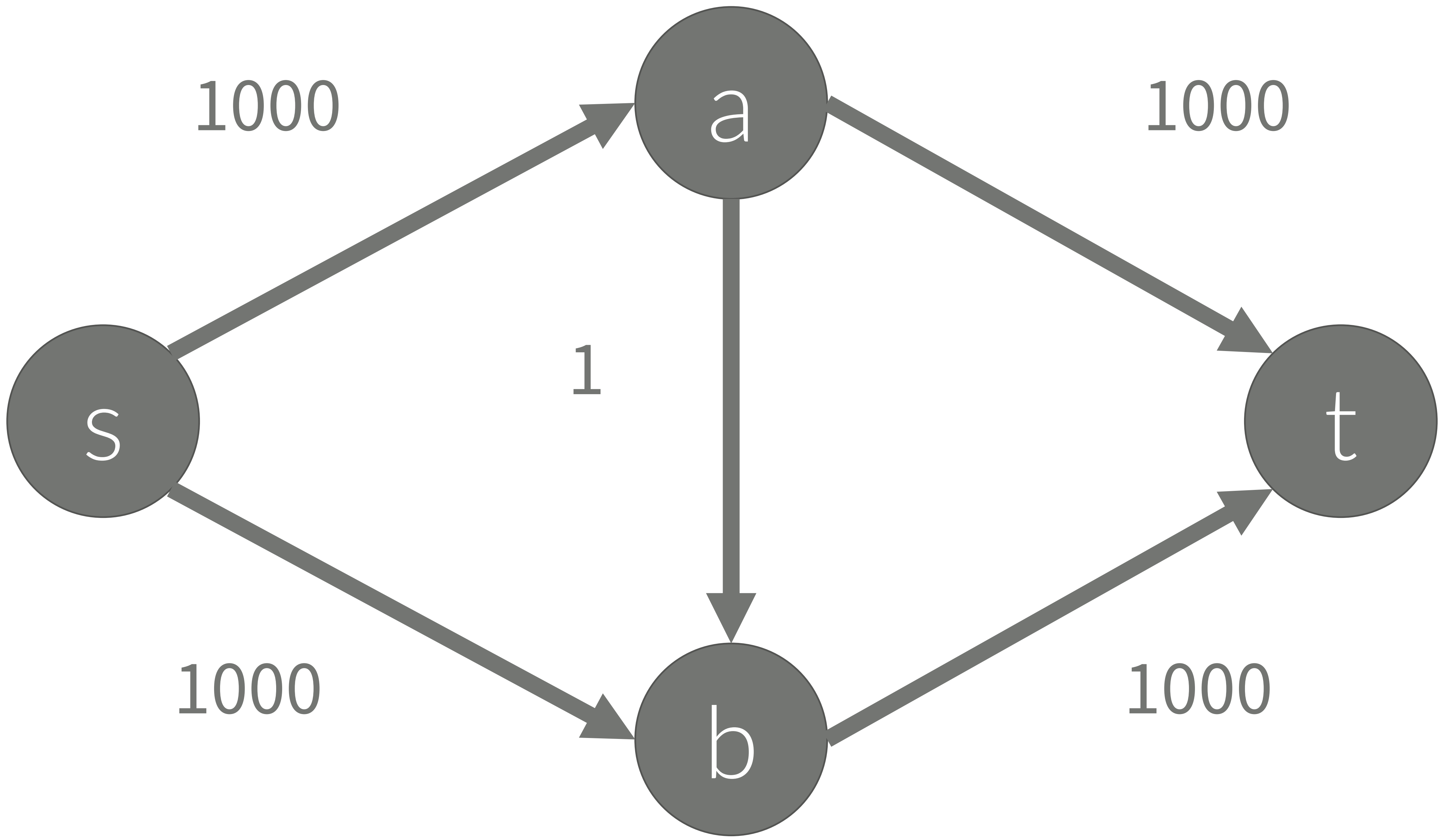
Maximum Flow

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

Ford-Fulkerson

Maximum Flow

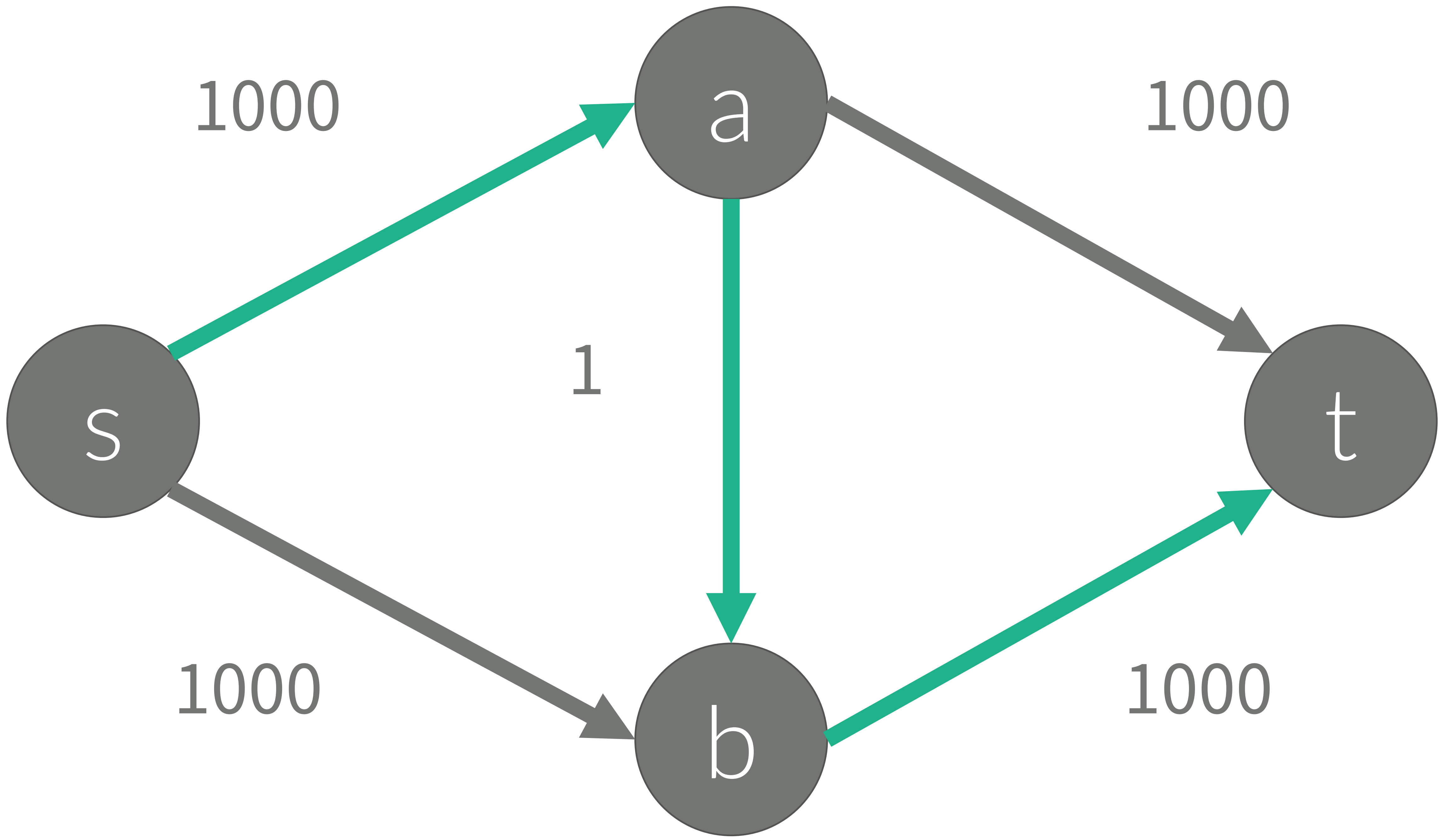
- 이런 경우



Ford-Fulkerson

Maximum Flow

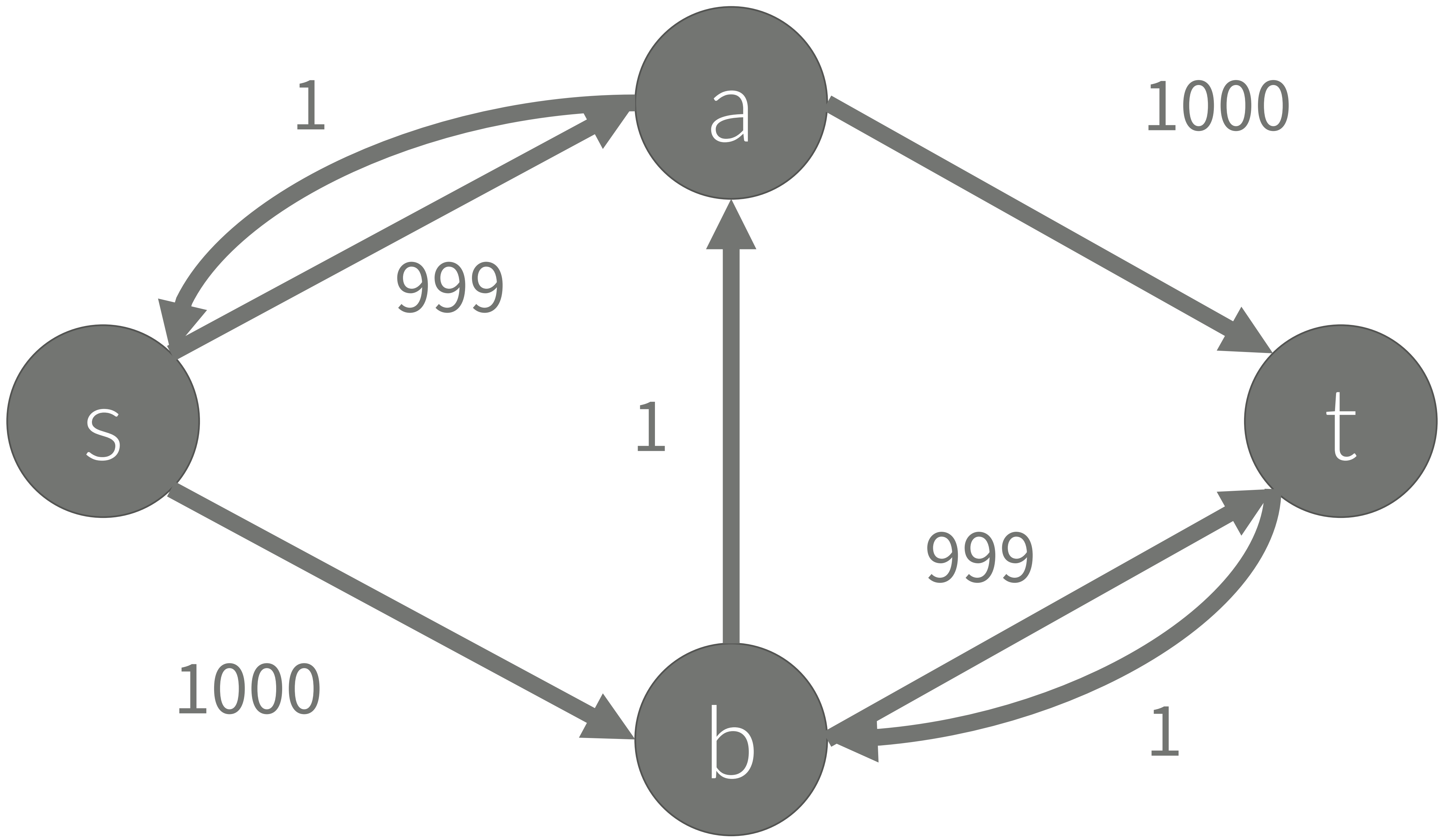
- 이런 경우



Ford-Fulkerson

Maximum Flow

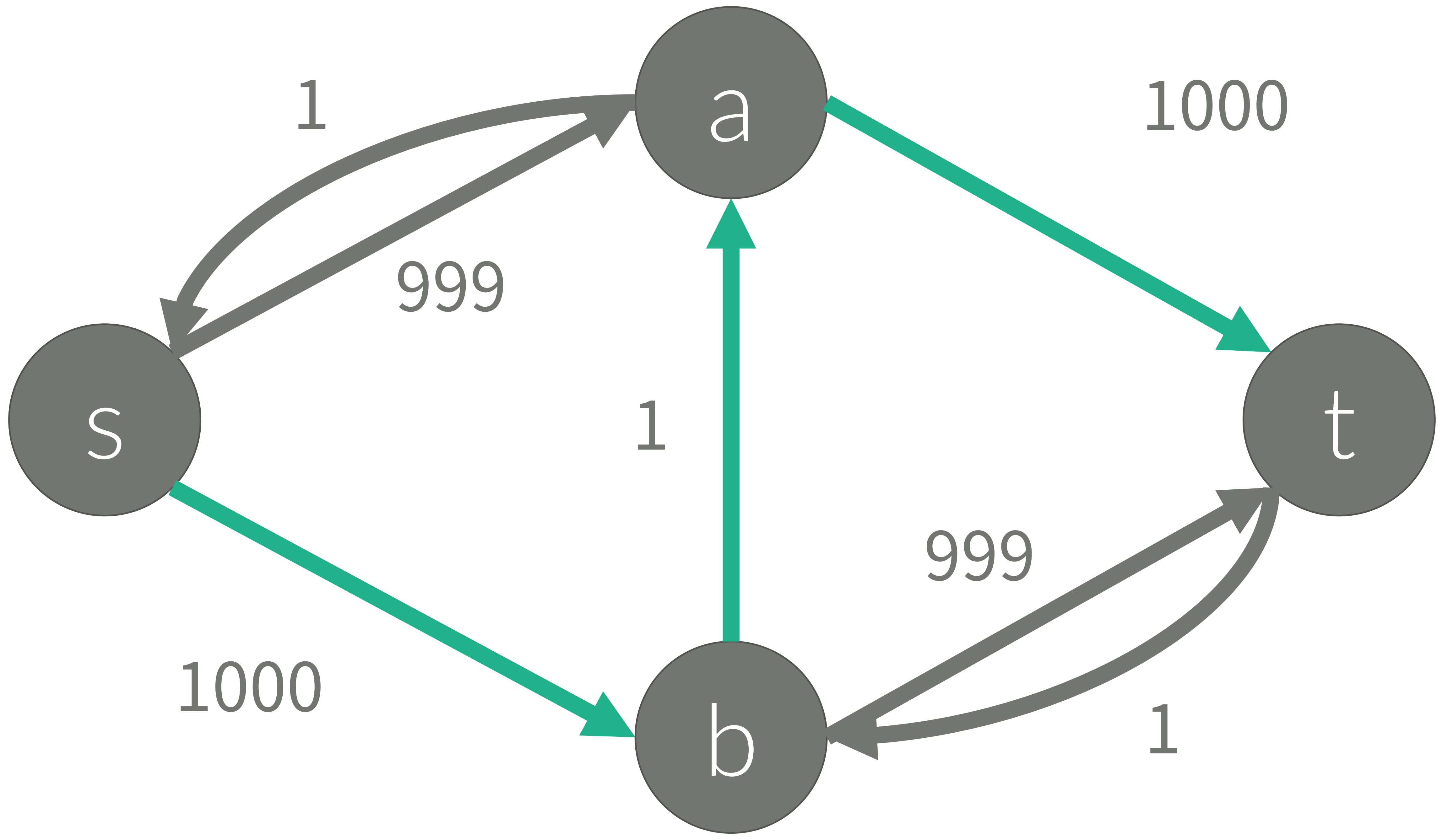
- 이런 경우



Ford-Fulkerson

Maximum Flow

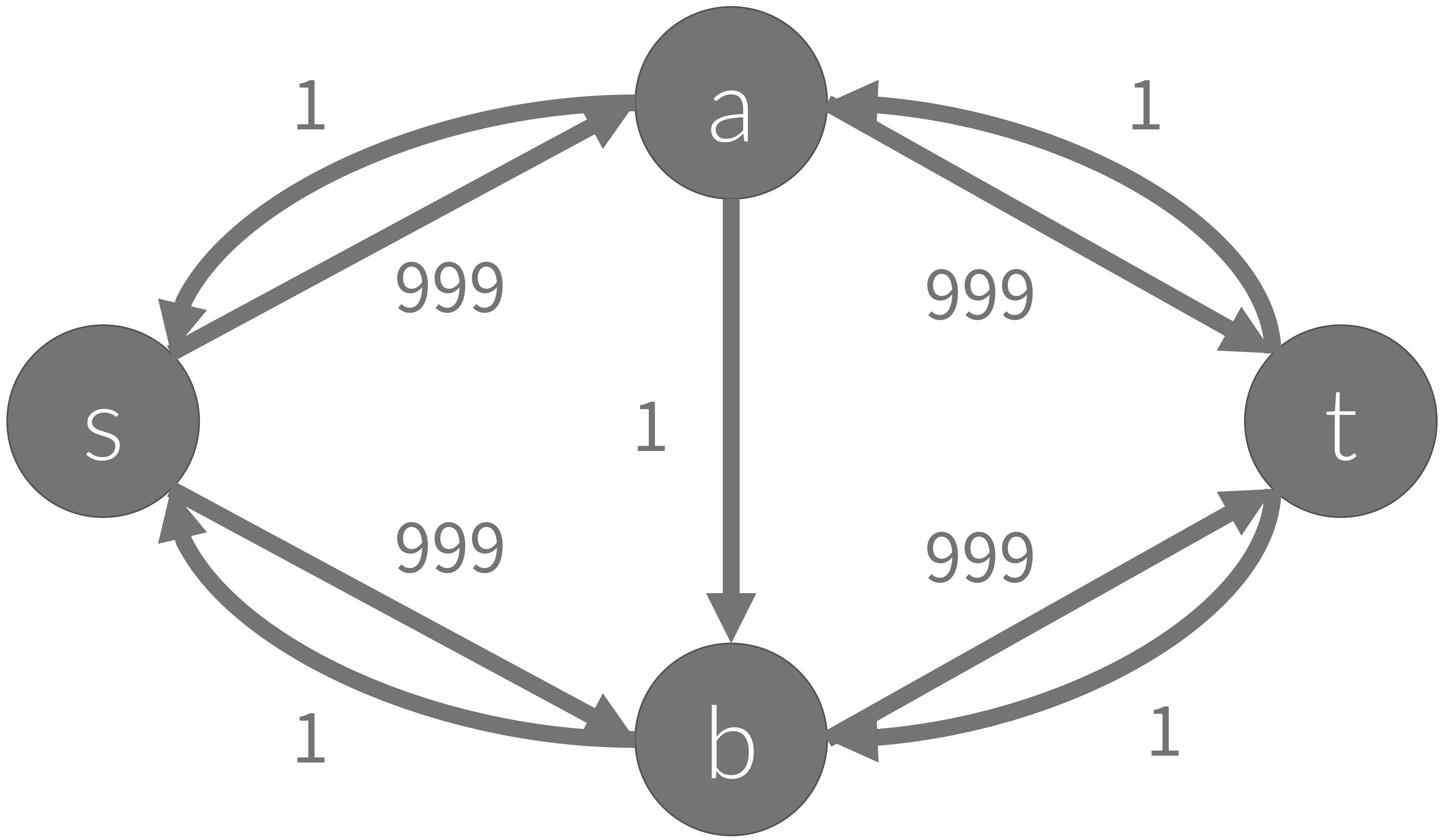
- 이런 경우



Ford-Fulkerson

Maximum Flow

- 이런 경우



Ford-Fulkerson

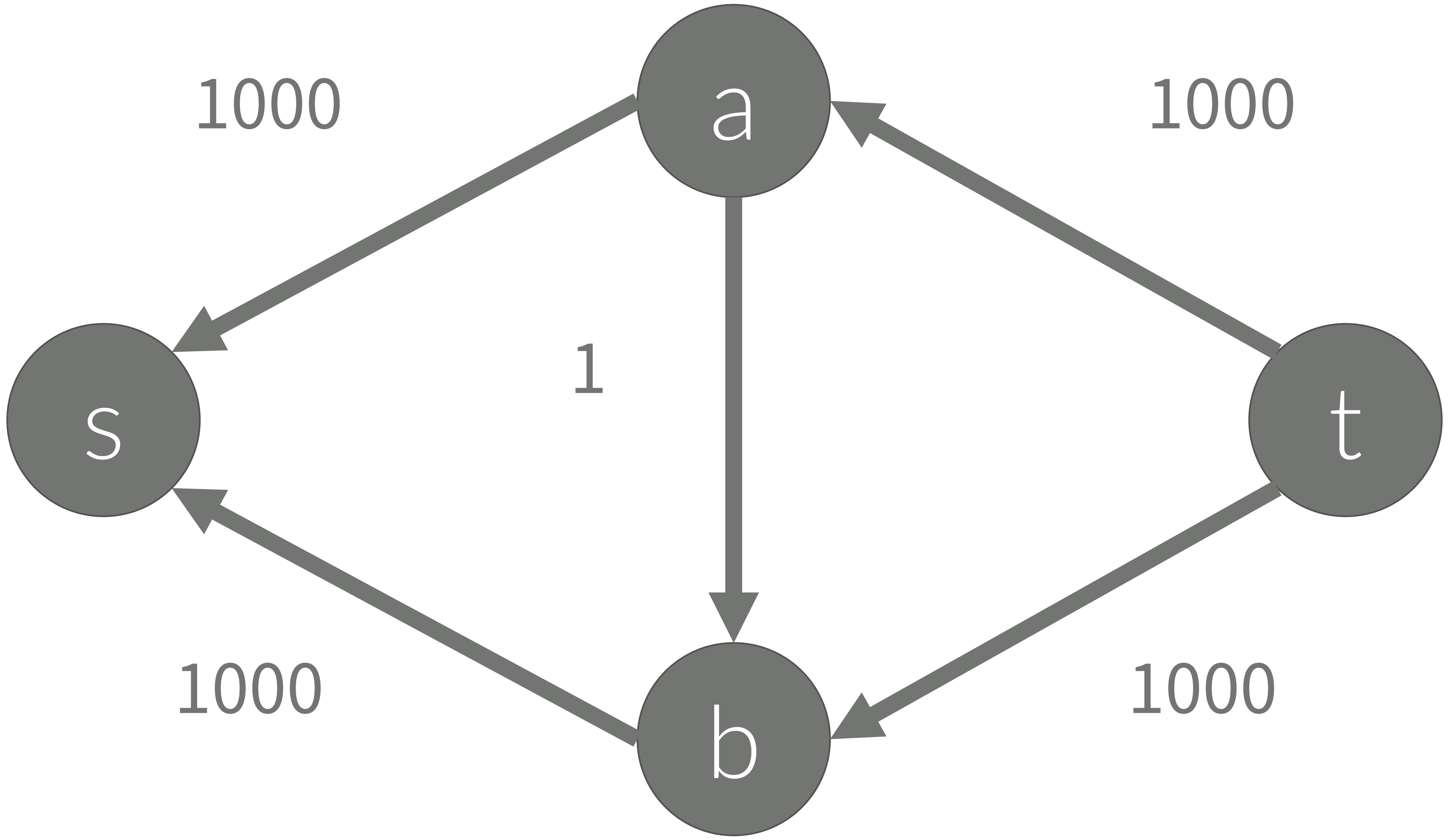
Maximum Flow

- 이 과정을 1998번 더 하면

Ford-Fulkerson

Maximum Flow

- Flow network



Edmond-Karp

Edmond-Karp

Maximum Flow

- 최대 유량을 구하는 알고리즘

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

Edmond-Karp

Maximum Flow

- Edmond-Karp와 Ford-Fulkerson의 차이는 DFS와 BFS밖에 없다.
- Edmond-Karp의 시간 복잡도는 $O(VE^2)$이다.
- BFS의 시간 복잡도: $O(E)$
- 각 간선이 최소값이 되는 횟수는 최대 V번이다.
- 따라서, $O(VE^2)$

최대 유량

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

- 최대 유량을 구하는 문제

최대 유량

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

- Ford Fulkerson 소스: http://codeplus.codes/363499b0f90f47ec939c8f760e19ce3c
- Edmond Karp 소스: http://codeplus.codes/aced5f770aa147cdb318d8e72fef243f

이분 매칭

이분 매칭

Bipartite Matching

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

이분 매칭

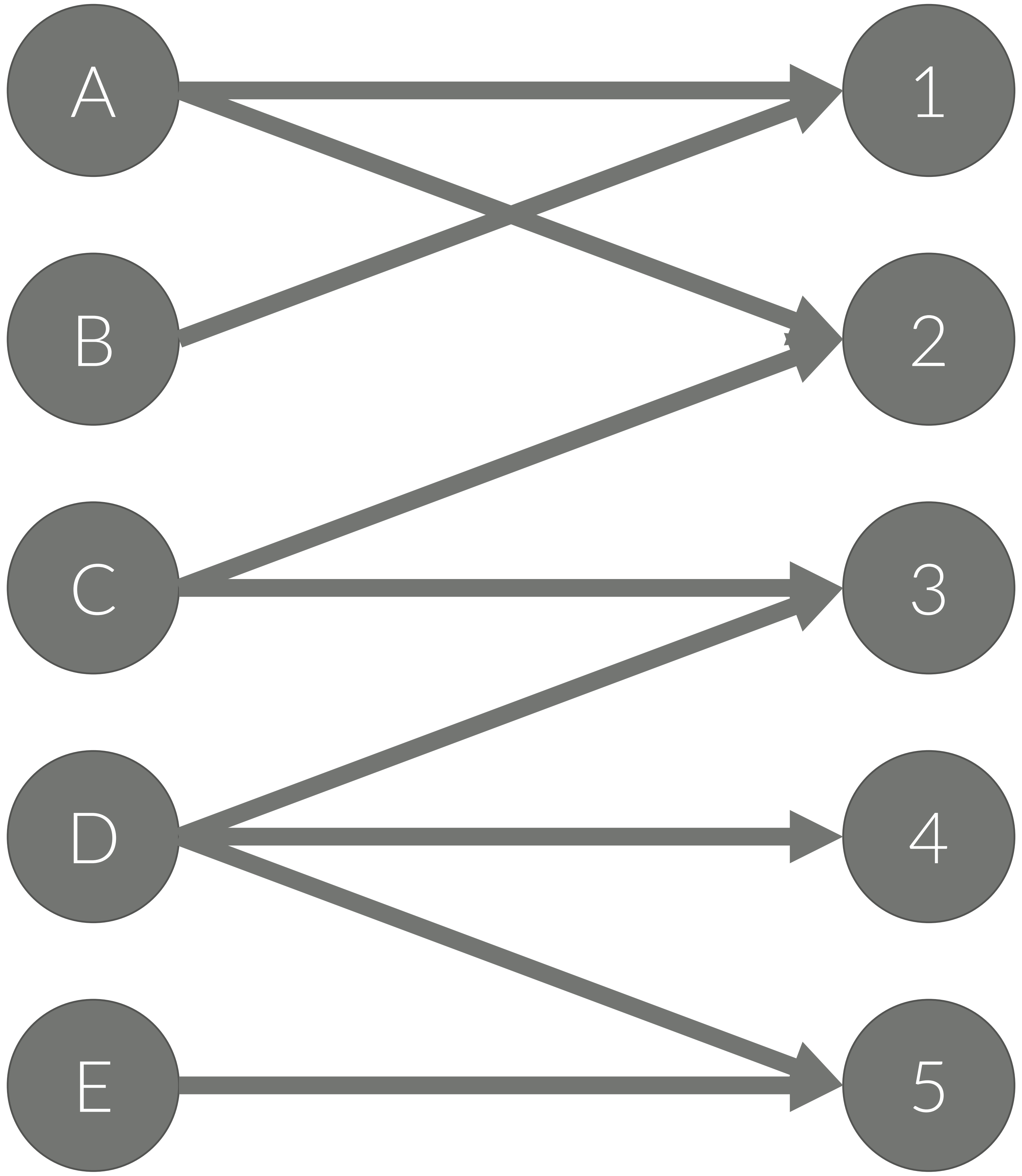
Bipartite Matching

- 이분 매칭: 이분 그래프에서 매칭의 최대값

이분 매칭

Bipartite Matching

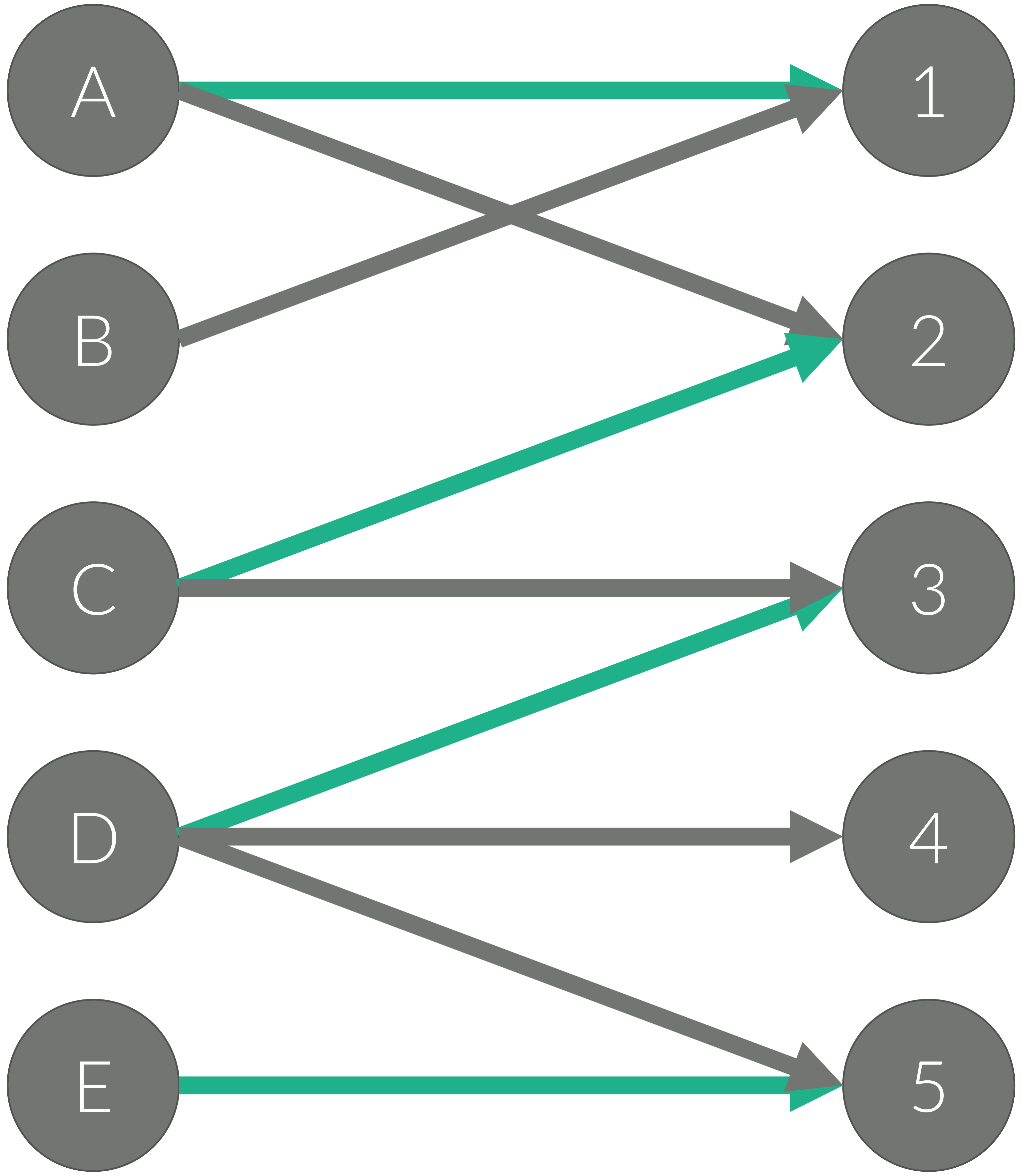
- 사람: A~E, 일: 1~5, 각 사람이 할 수 있는 일을 Edge로 연결



이분 매칭

Bipartite Matching

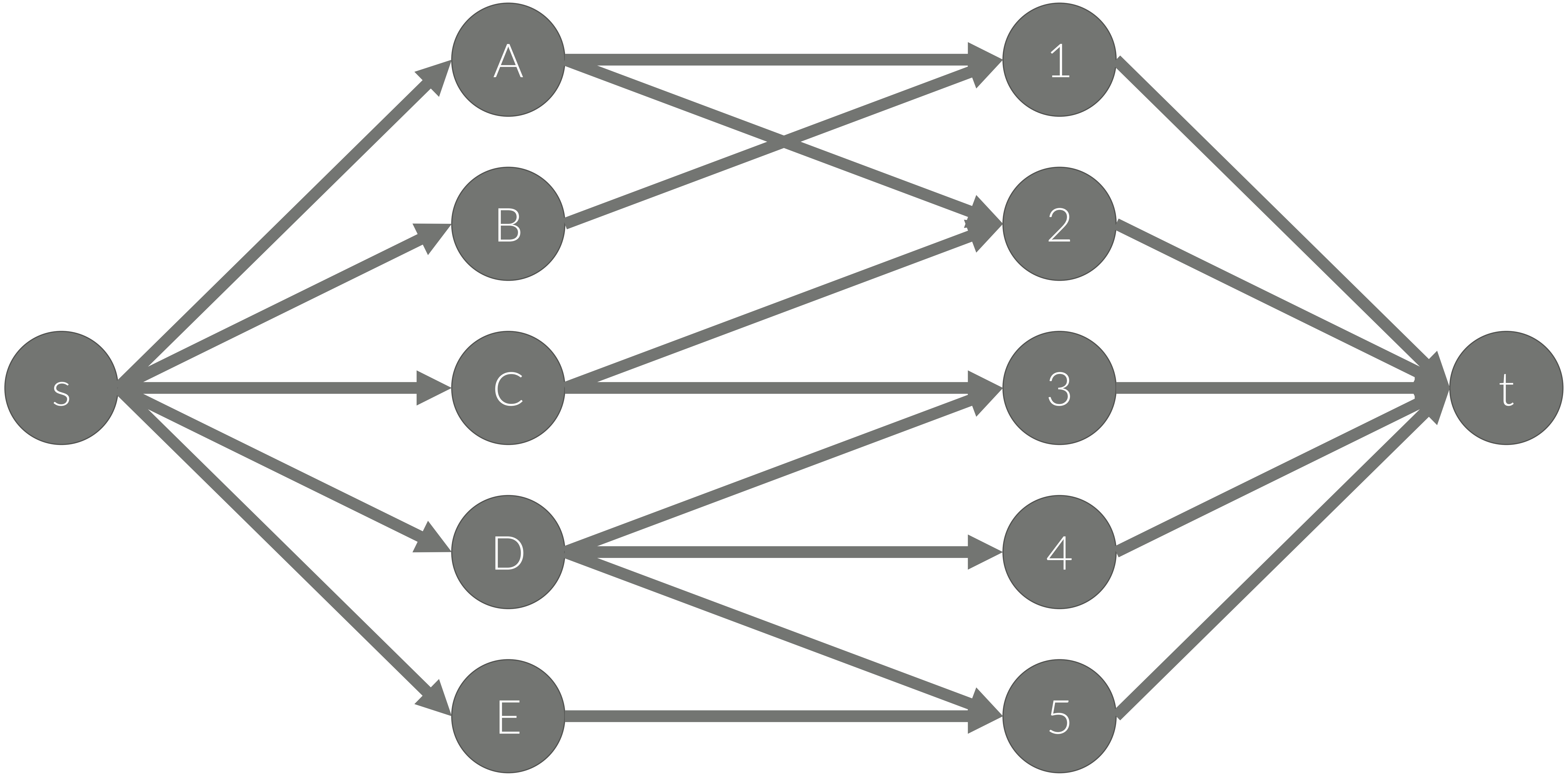
- 사람: A~E, 일: 1~5, 각 사람이 할 수 있는 일을 Edge로 연결



이분 매칭

Bipartite Matching

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



이분 매칭

Bipartite Matching

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

이분 매칭

Bipartite Matching

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

열혈강호

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

- 사람 N명 일 M개
- 각 직원은 한 개의 일만 할 수 있고, 각각의 일을 담당하는 사람은 1명이어야 한다.
- 할 수 있는 일의 최대 개수

열혈강호

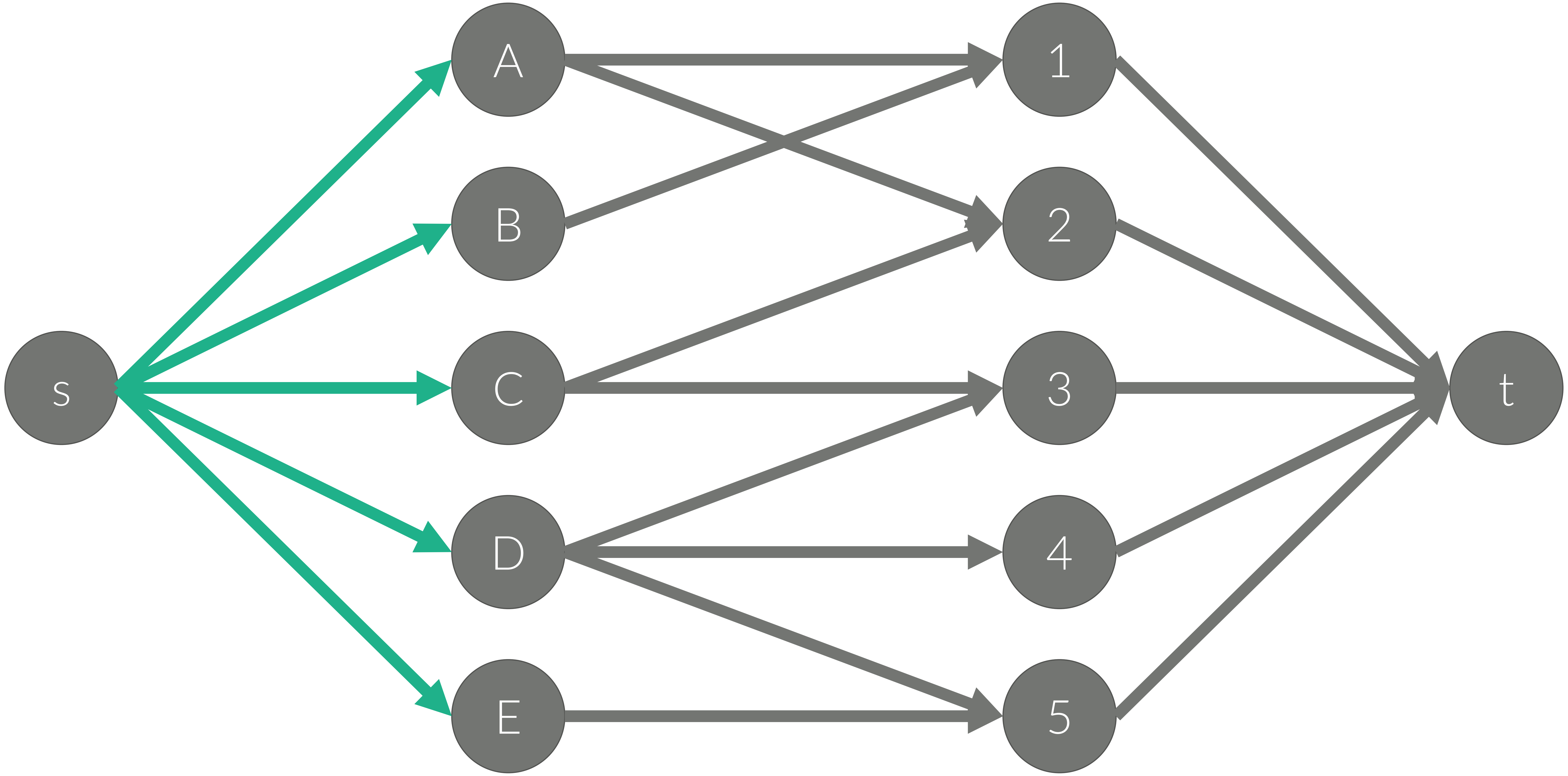
https://www.acmicpc.net/problem/11375

- 소스: http://codeplus.codes/46d33890f7474934b4a69c63ad25e615
- 소스: http://codeplus.codes/77d8b17d8c8348e786c81581cce23440

이분 매칭

Bipartite Matching

- 항상 1 또는 0이기 때문에, edge를 만들지 않아도 된다



이분 매칭

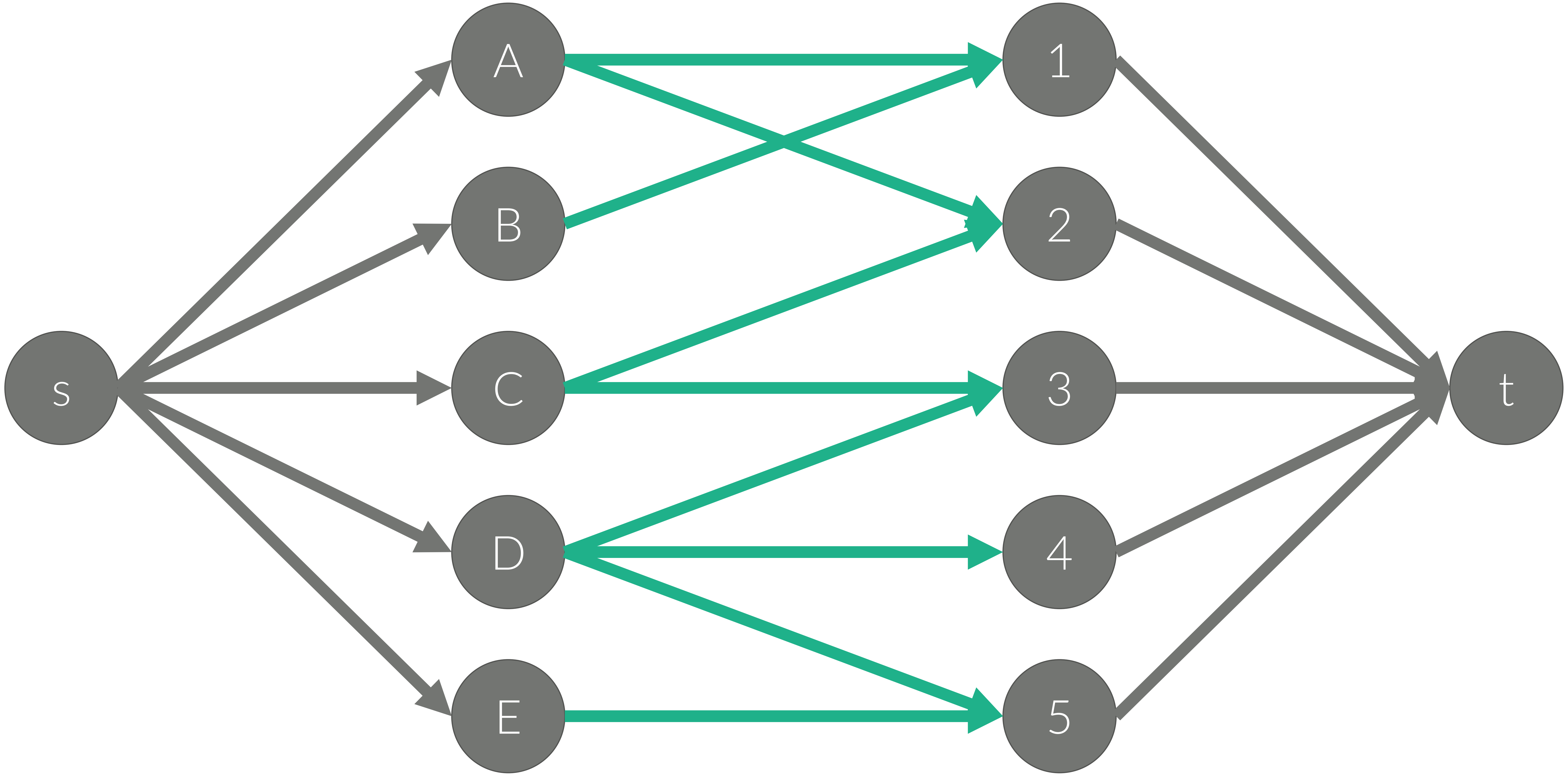
Bipartite Matching

```
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

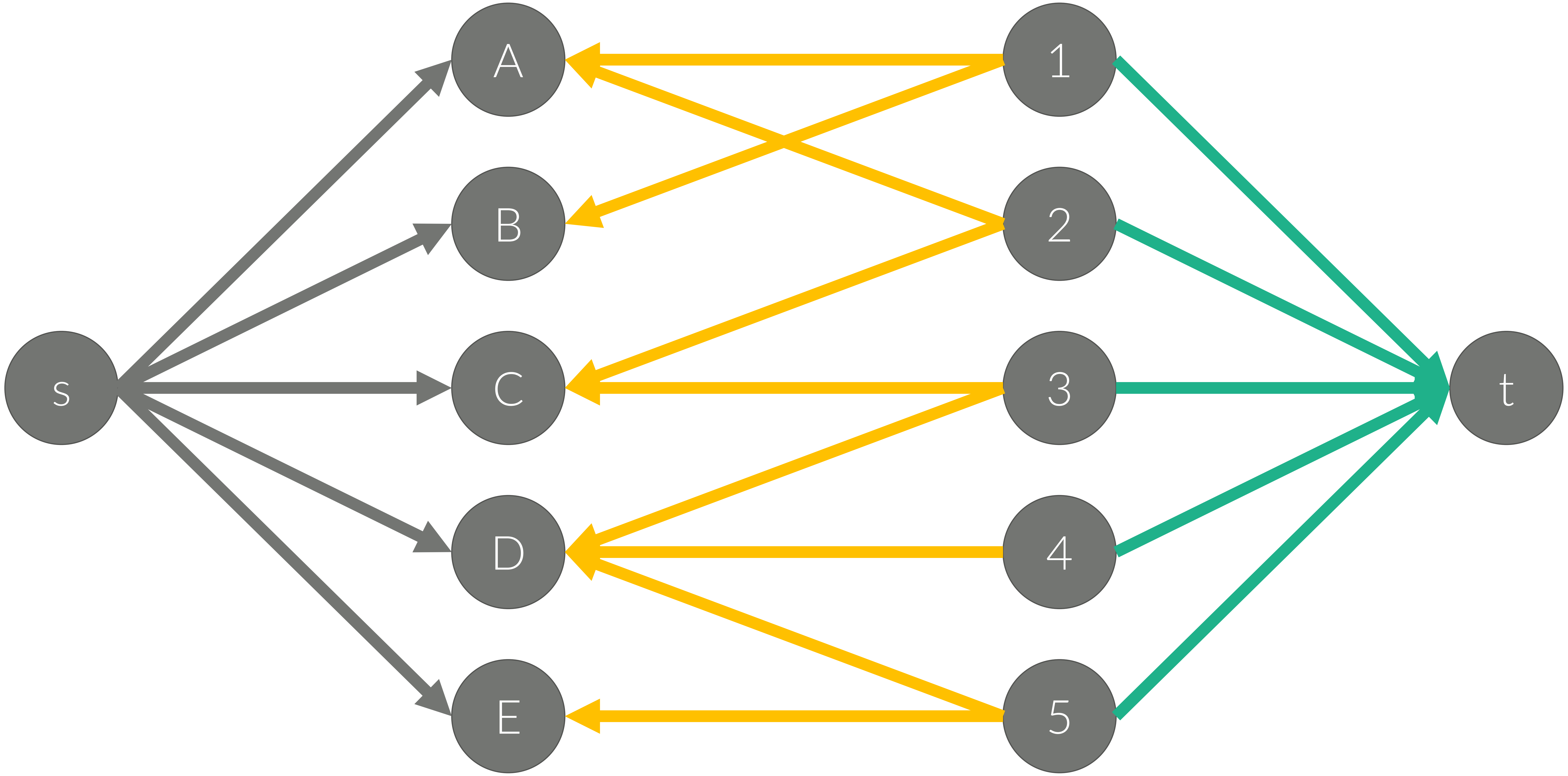
- 그대로 유지해야 한다



이분 매칭

Bipartite Matching

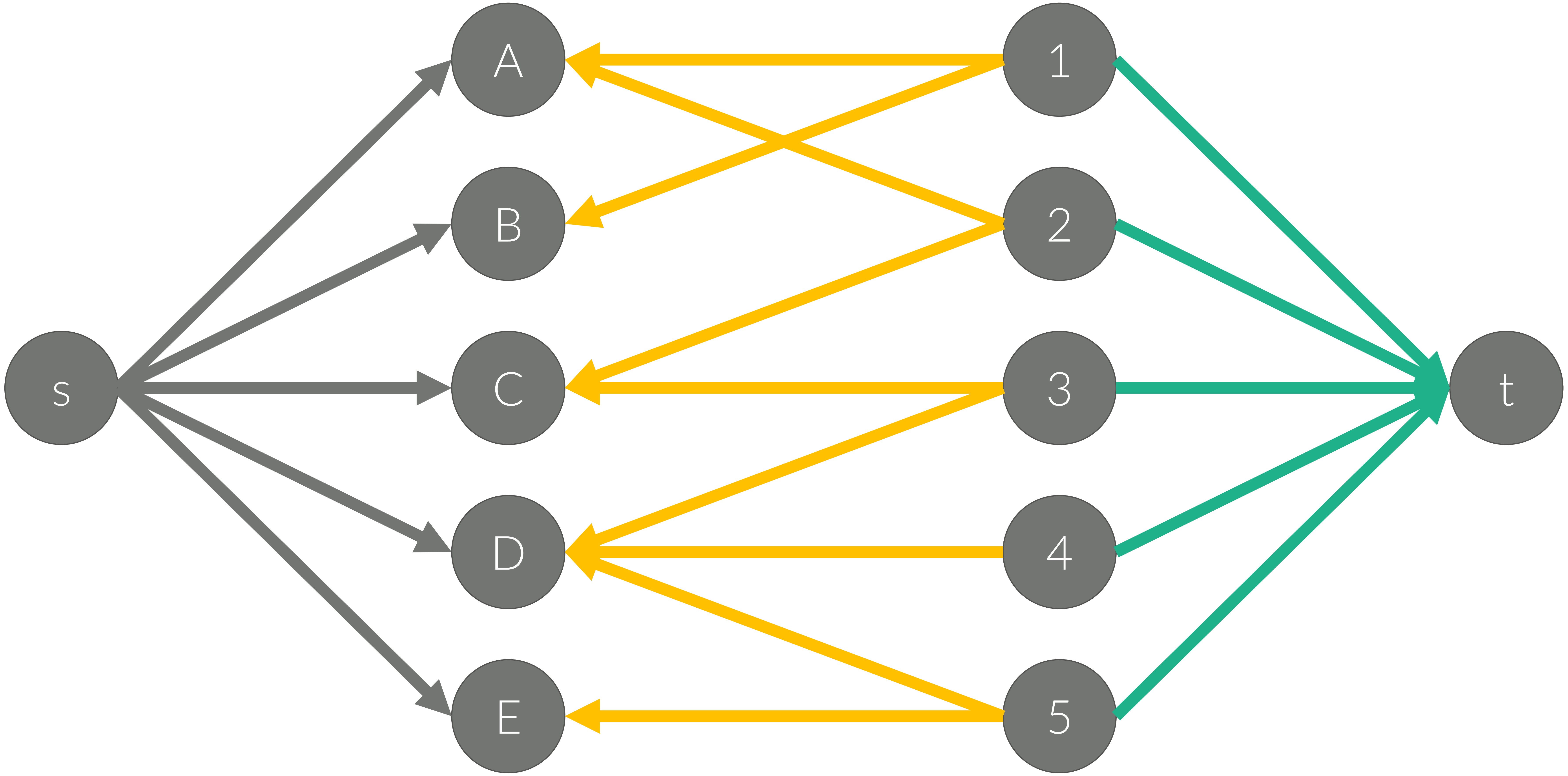
- 오른쪽은 항상 Sink로 가거나 아니면 다시 왼쪽으로 가게된다



이분 매칭

Bipartite Matching

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



이분 매칭

Bipartite Matching

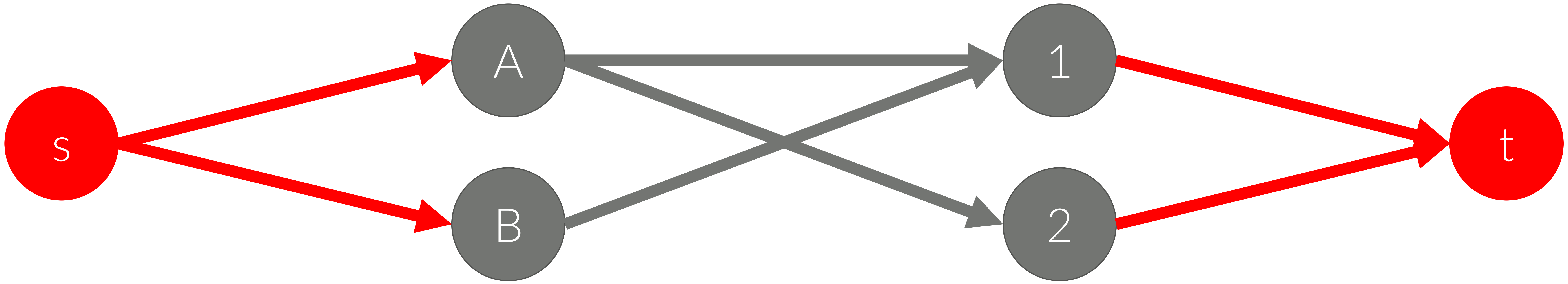
```
bool dfs(int x) {
    if (x == -1) return true;
    for (int next : graph[x]) {
        if (check[next]) continue;
        check[next] = true;
        if (dfs(pred[next])) {
            pred[next] = x;
            return true;
        }
    }
    return false;
}
```

이분 매칭

Bipartite Matching

i	1	2
pred[i]	-1	-1

- 예시
- 빨간 정점, 간선은 실제로는 없다.

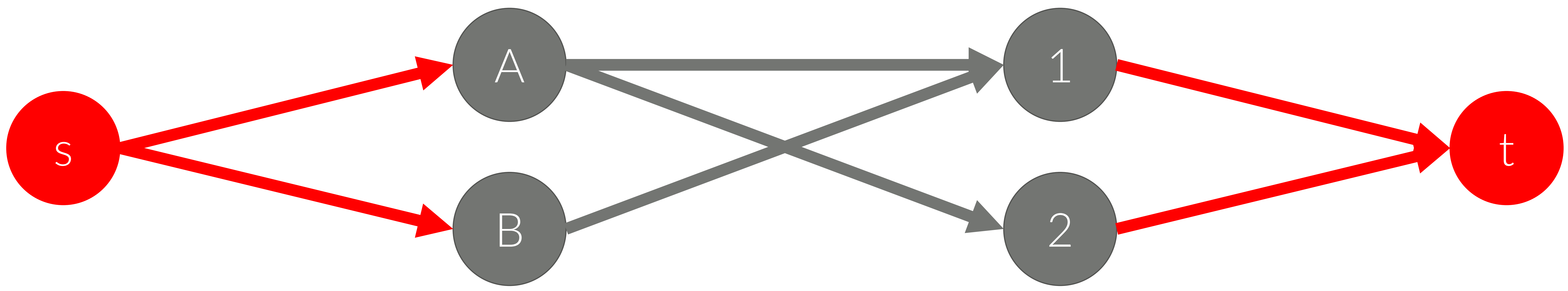


이분 매칭

Bipartite Matching

- A에서 탐색을 시작
- 그림: s → A → 1 → t
- 실제: A → (pred[1] = t)

i	1	2
pred[i]	-1	-1

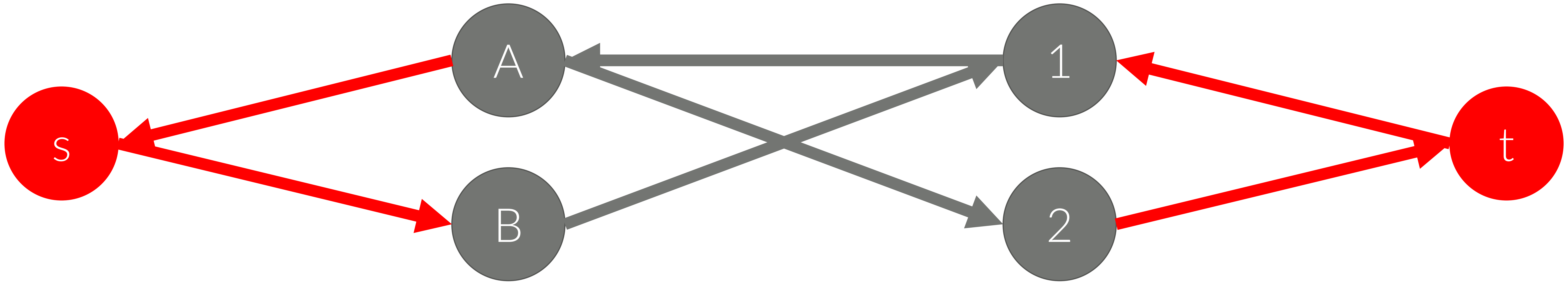


이분 매칭

Bipartite Matching

i	1	2
pred[i]	A	-1

- A에서 탐색을 시작
- 그림: s → A → 1 → t
- 실제: A → (pred[1] = t)

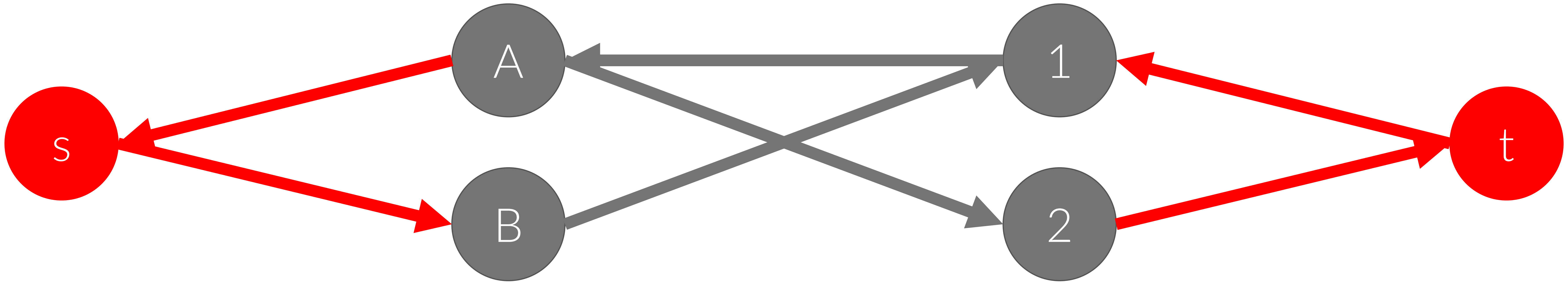


이분 매칭

Bipartite Matching

i	1	2
pred[i]	A	-1

- B에서 탐색을 시작
- 그림: s → B → 1 → A → 2 → t
- 실제: B → (pred[1] = A) → (pred[2] = t)

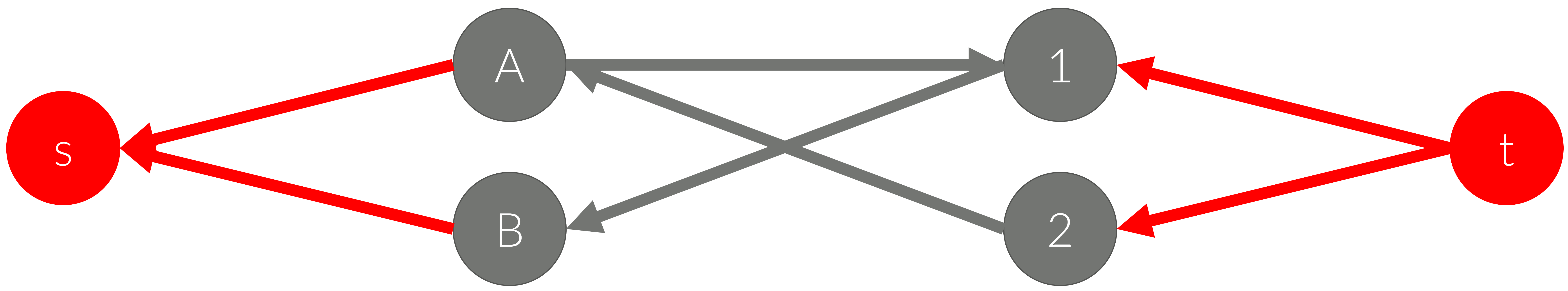


이분 매칭

Bipartite Matching

i	1	2
pred[i]	B	A

- B에서 탐색을 시작
- 그림: s → B → 1 → A → 2 → t
- 실제: B → (pred[1] = A) → (pred[2] = t)



열혈강호

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

- 소스: http://codeplus.codes/0d237698e32243938eba56888c8d5290

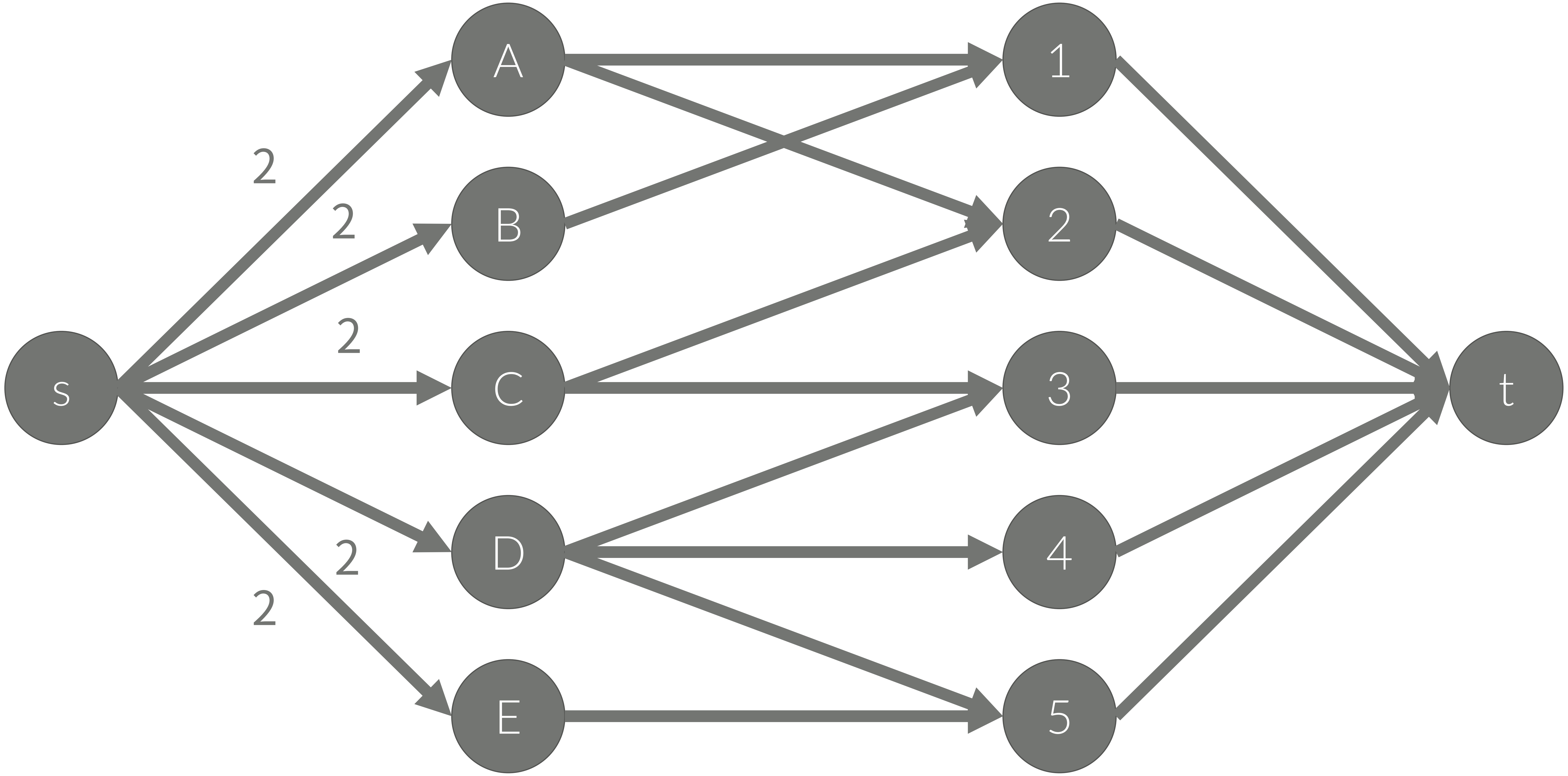
열혈강호 2

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

- 사람 N명 일 M개
- 각 직원은 최대 두 개의 일을 할 수 있고, 각각의 일을 담당하는 사람은 1명이어야 한다.
- 할 수 있는 일의 최대 개수

열혈강호 2

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



열혈강호 2

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

- 소스: http://codeplus.codes/45b6890ec05b4e78b4b5349f9073f5fd

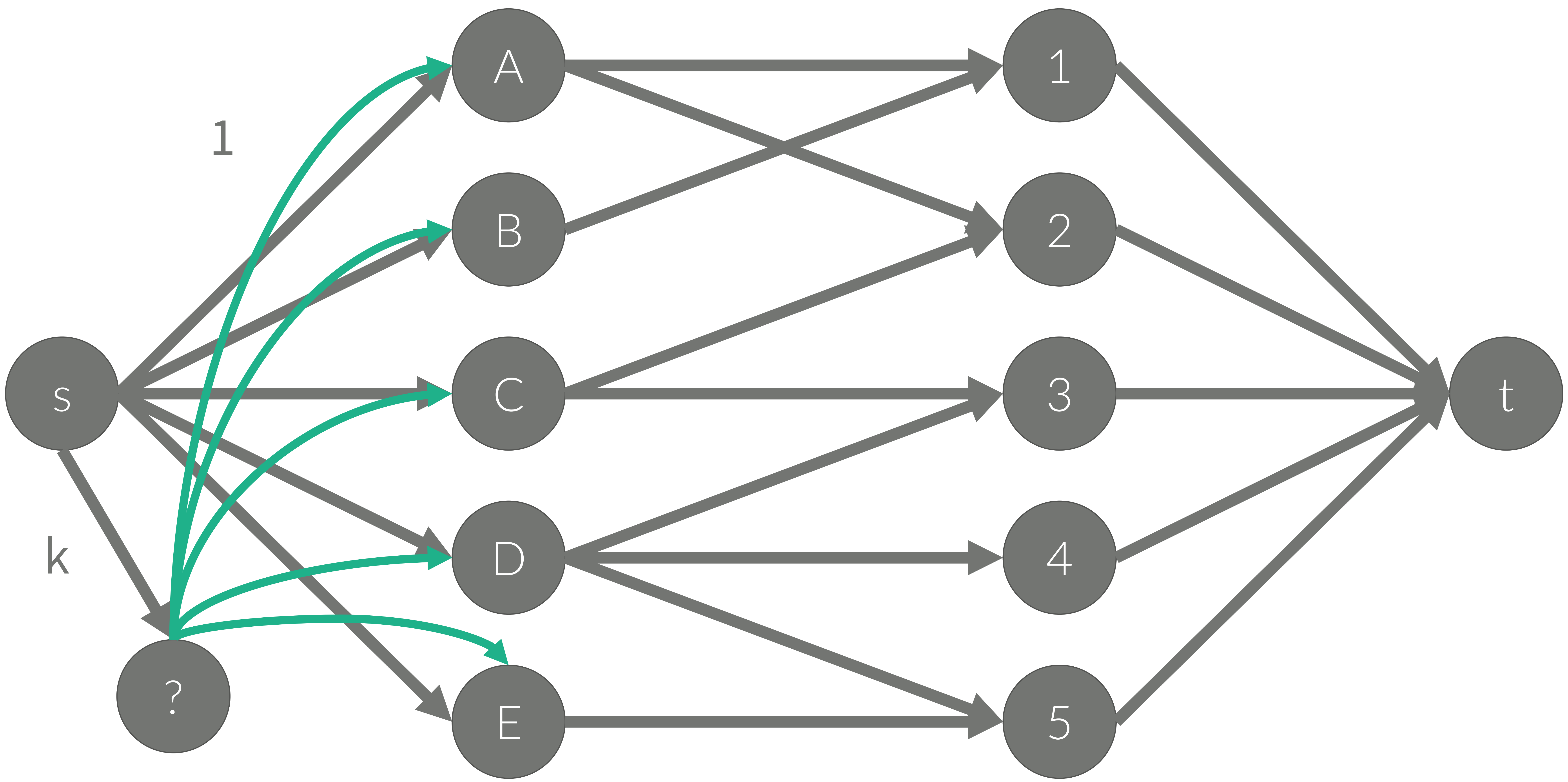
열혈강호 3

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

- 사람 N명 일 M개
- 각 직원은 한 개의 일만 할 수 있고, 각각의 일을 담당하는 사람은 1명이어야 한다.
- 단, N명 중에서 K명은 일을 최대 두 개 할 수 있다.
- 할 수 있는 일의 최대 개수

열혈강호 3

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



열혈강호 3

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

- 소스: http://codeplus.codes/66e63fdd7ece446db65a3a3d43d9ecb6

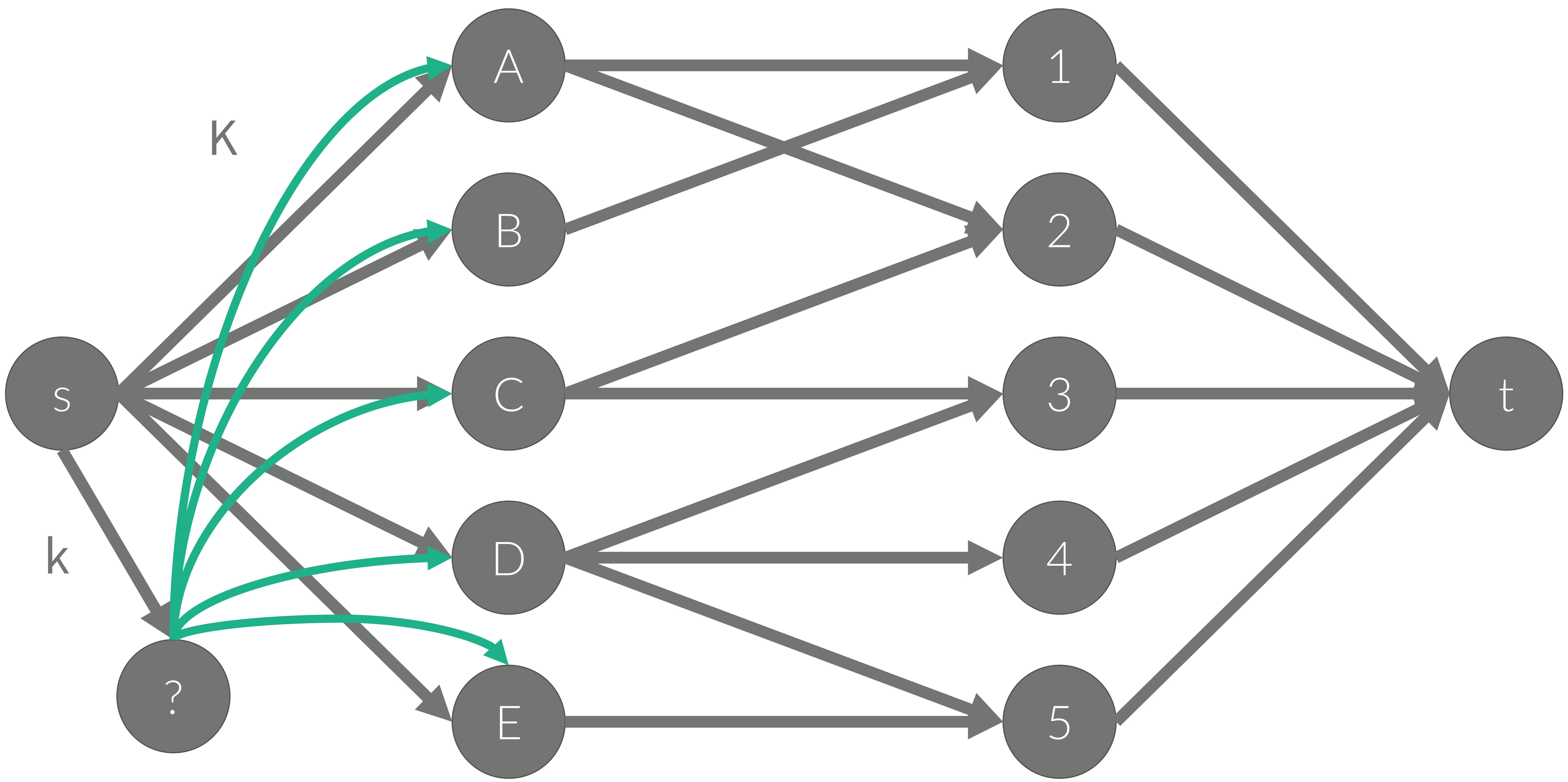
열혈강호 4

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

- 사람 N명 일 M개
- 각 직원은 한 개의 일만 할 수 있고, 각각의 일을 담당하는 사람은 1명이어야 한다.
- 지난달에 벌점을 X점 받은 사람은 일을 최대 X+1개까지 할 수 있다.
- 벌점의 합 K를 알고 있을 때, 벌점을 직원에게 적절히 분배해서
- 할 수 있는 일의 최대 개수

열혈강호 4

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



열혈강호 4

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

- 소스: http://codeplus.codes/a45e14b9568c4ef38a57d57bf479b6fb

축사 배정

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

- 소 N마리, 축사 M개가 있다. 축사에는 최대 한 개의 소만 들어갈 수 있다.
- 각 소가 들어가기 원하는 축사 번호가 주어졌을 때
- 최대한 많은 소를 축사에 배정하는 문제

축사 배정

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

- 열혈강호와 똑같은 문제
- 왼쪽: 소, 오른쪽: 축사

축사 배정

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

- 소스: http://codeplus.codes/b122b9992202464697a0bc2062acc2f6

노트북의 주인을 찾아서

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

- N명의 학생과 N개의 노트북이 있다.
- 각 학생이 자신의 노트북이라고 주장하는 정보가 M개 주어진다.
- a번 사람이 b번 노트북을 자신의 것이라고 생각한다는 의미
- 최대 만족할 수 있는 사람의 수를 구하는 문제 (사람 1명 – 노트북 1개 쌍)

노트북의 주인을 찾아서

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

- 열혈강호와 똑같은 문제
- 왼쪽: 사람, 오른쪽: 노트북

노트북의 주인을 찾아서

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

- 소스: http://codeplus.codes/277ac39bded84a36a46b0982d502ac3f

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

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

상어의 저녁식사

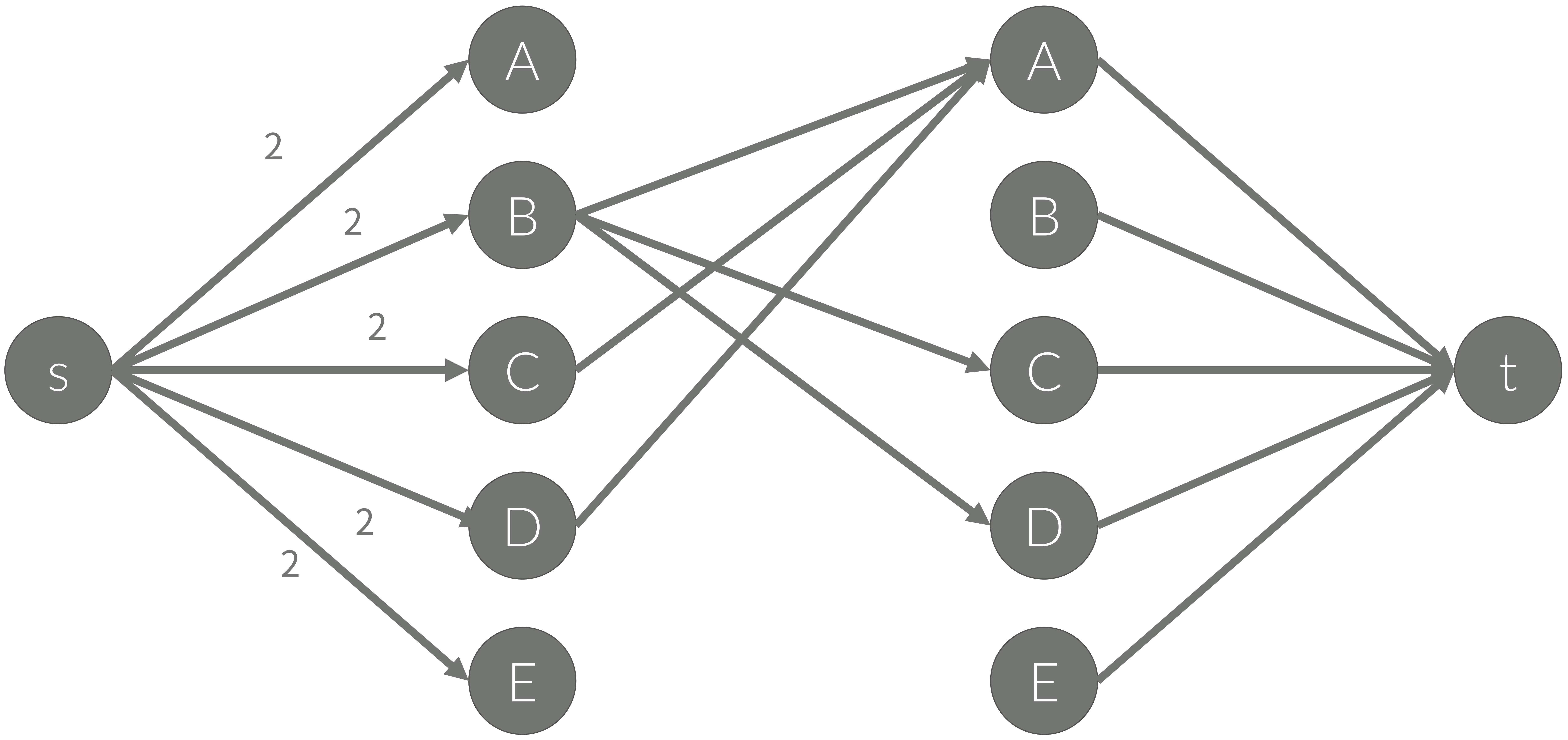
https://www.acmicpc.net/problem/1671

- 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 < j이면 잡아먹을 수 있다고 가정

상어의 저녁식사

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



상어의 저녁식사

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

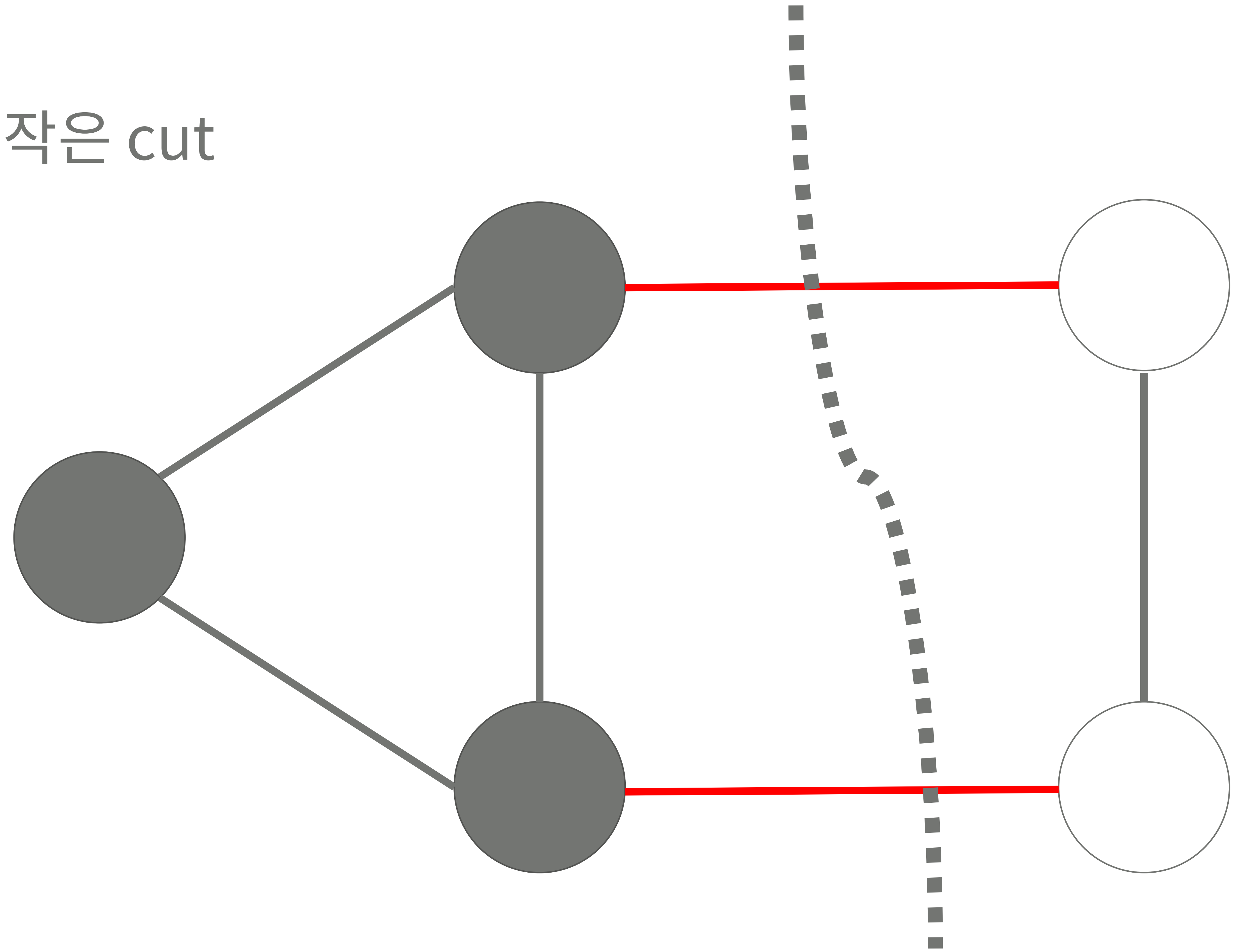
- 소스: http://codeplus.codes/cf235f2c042e443295b015b39200837a

Min-cut

Minimum cut

Minimum Cut

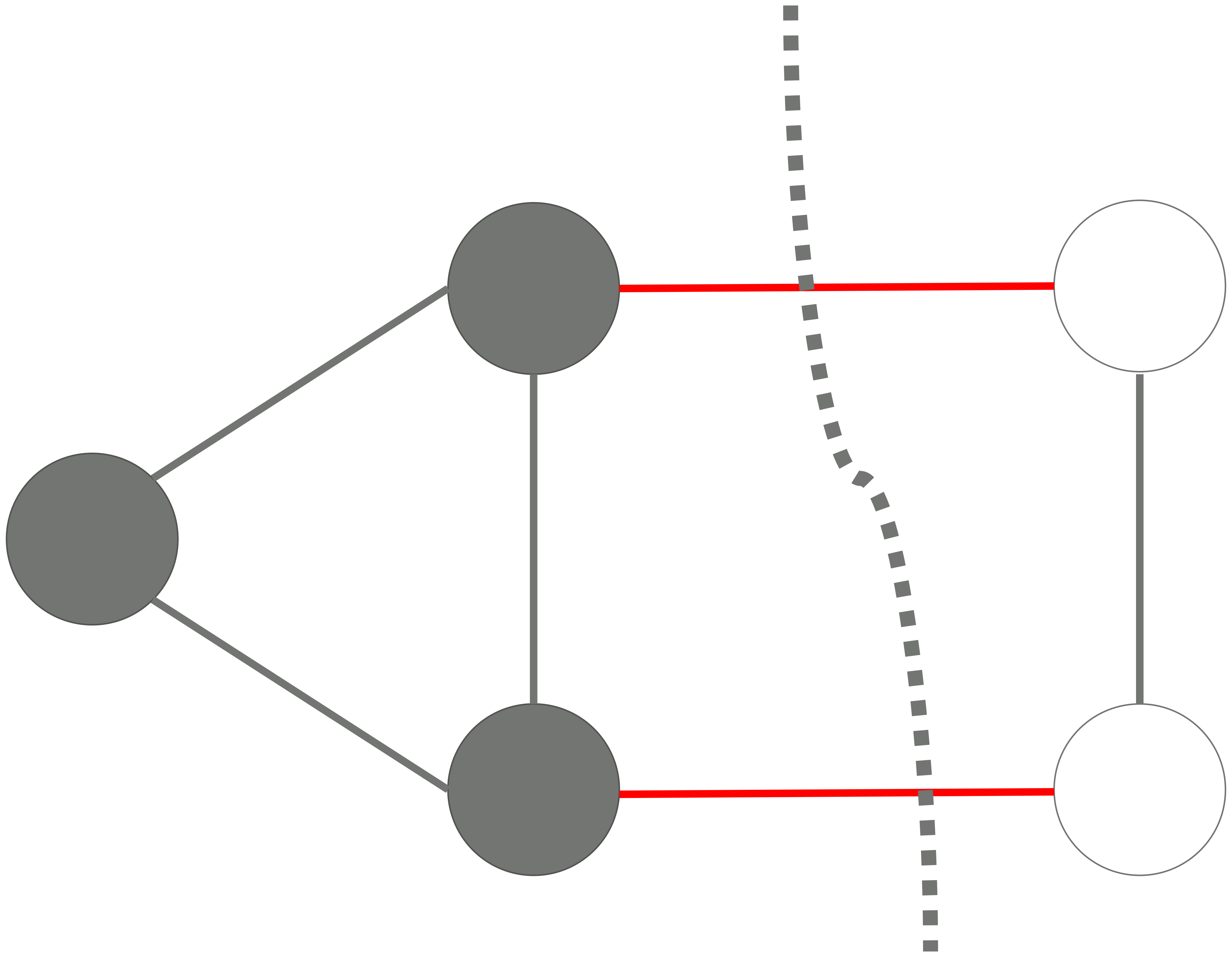
- Cut: 그래프를 2개의 서로다른 집합으로 나누는 것
- 두 집합을 A와 B라고 했을 때, 한 쪽 끝은 A에 다른 한 쪽 끝은 B에 있는 간선을 cut-set이라고 한다
- Minimum cut: 가장 작은 cut



Max-flow Min-cut Theorem

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 합의 최소값



학교 가지마!

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

- N*M 크기의 도시
- 빈 칸: .
- 벽: #
- 도현: K
- 학교: H

K	#	
#		H

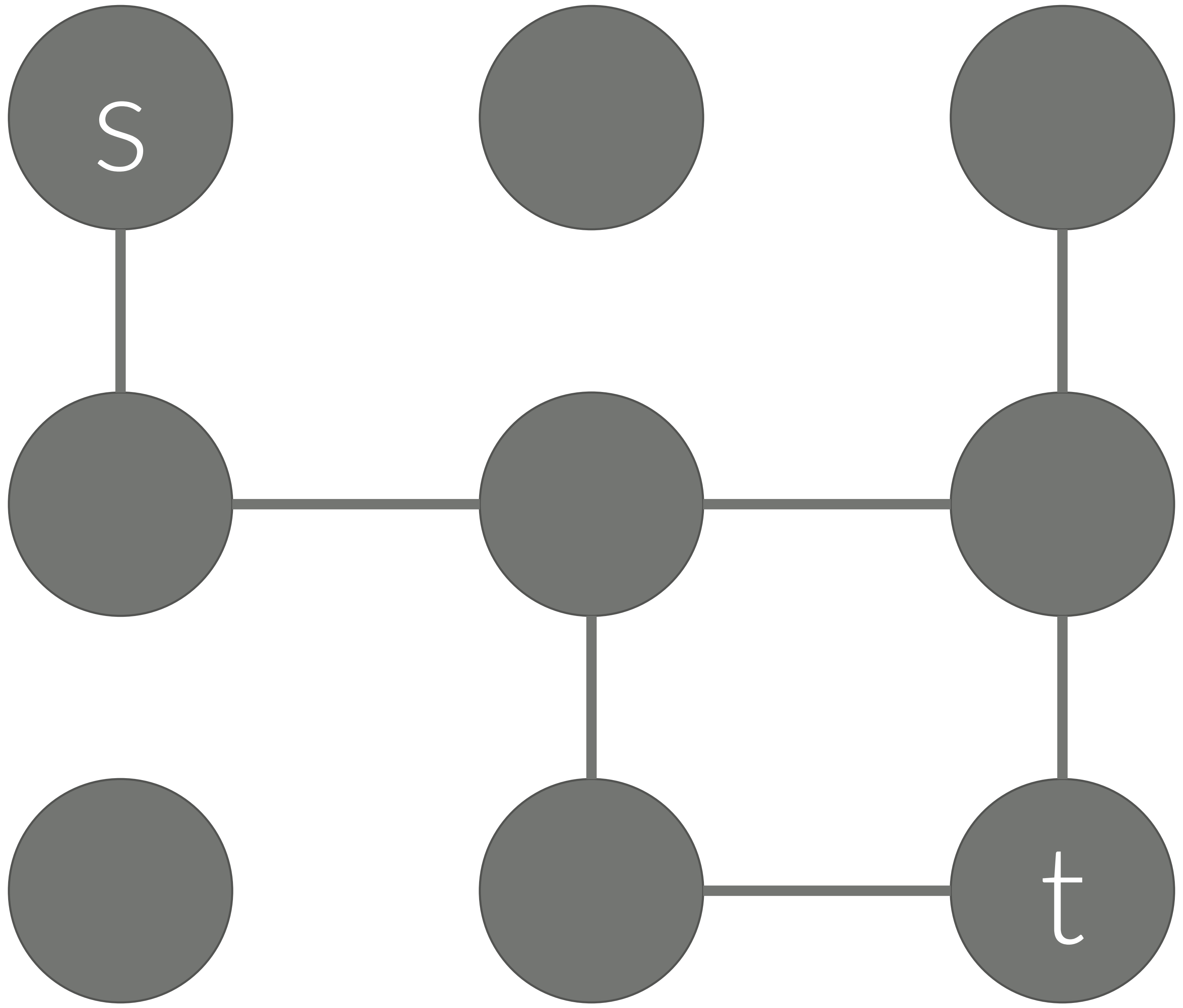
- 빈 칸을 적절히 벽으로 바꿔서 학교로 가지 못하게 하는 문제
- 최소 개수를 구해야 한다

학교 가지마!

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

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

K	#	
#		H

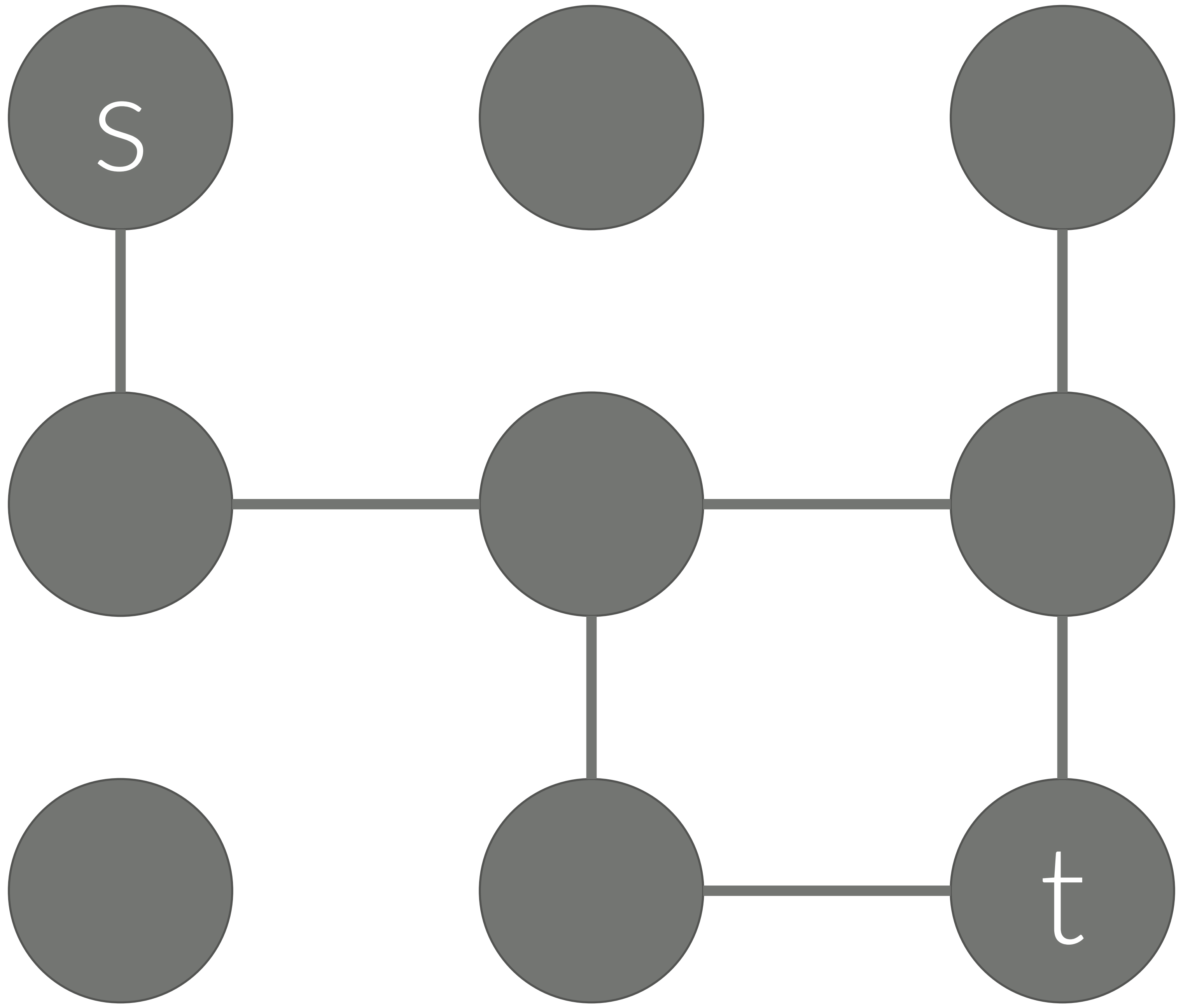


학교 가지마!

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

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

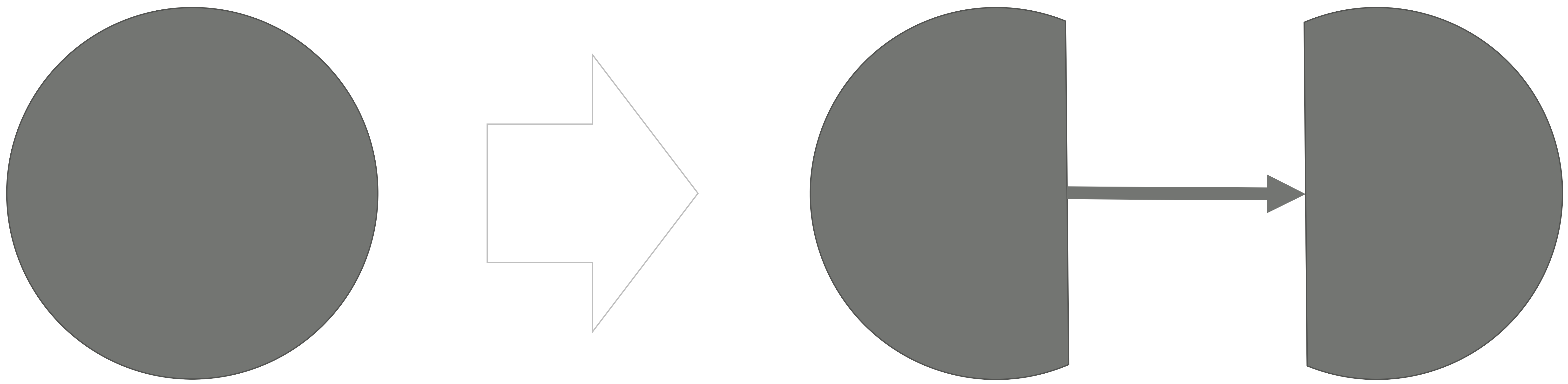
K	#	
#		H



학교 가지마!

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

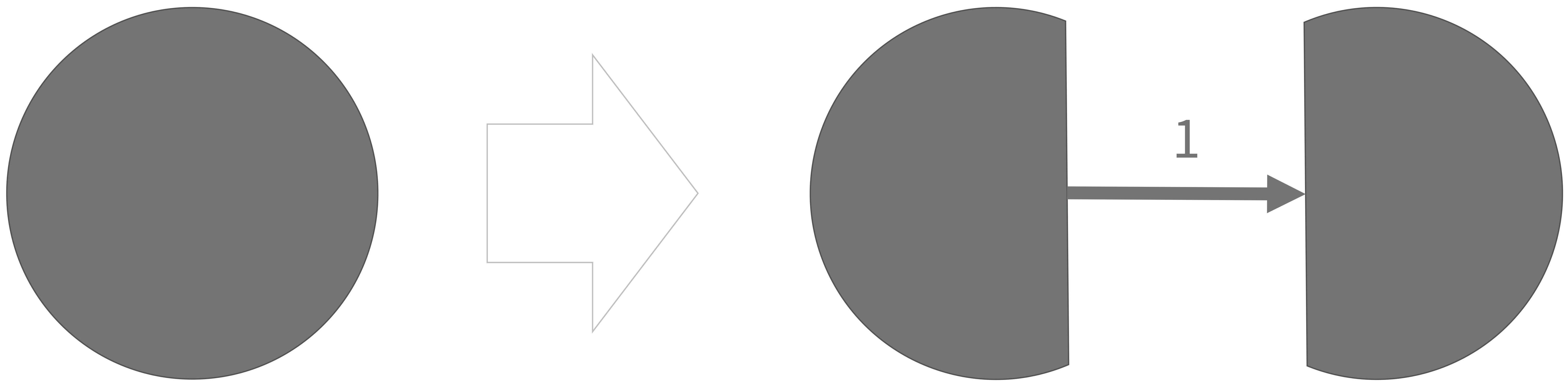
- 각 칸을 둘로 나눈다
- capacity는?



학교 가지마!

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

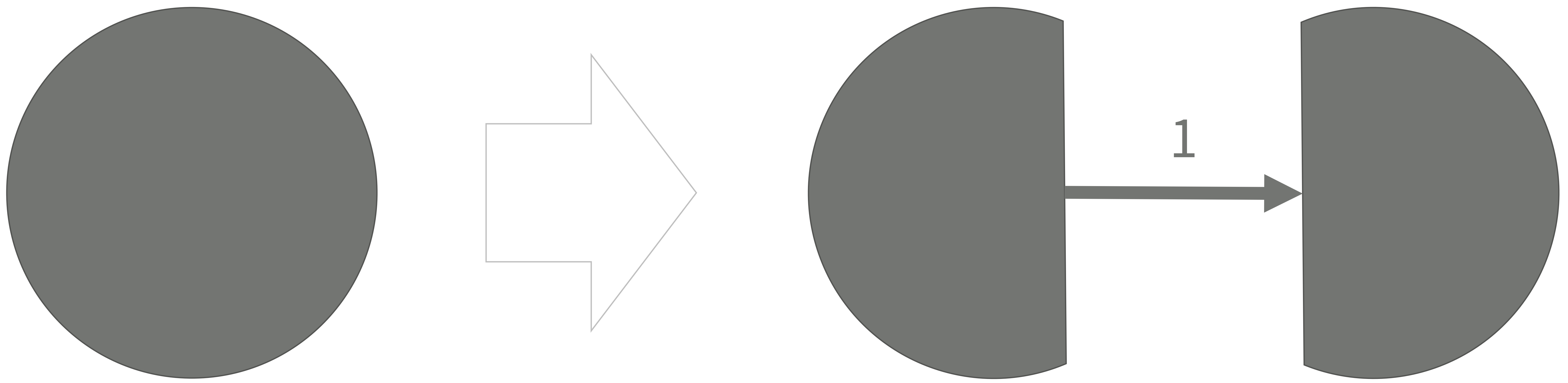
- 각 칸을 둘로 나눈다
- capacity는? 1



학교 가지마!

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

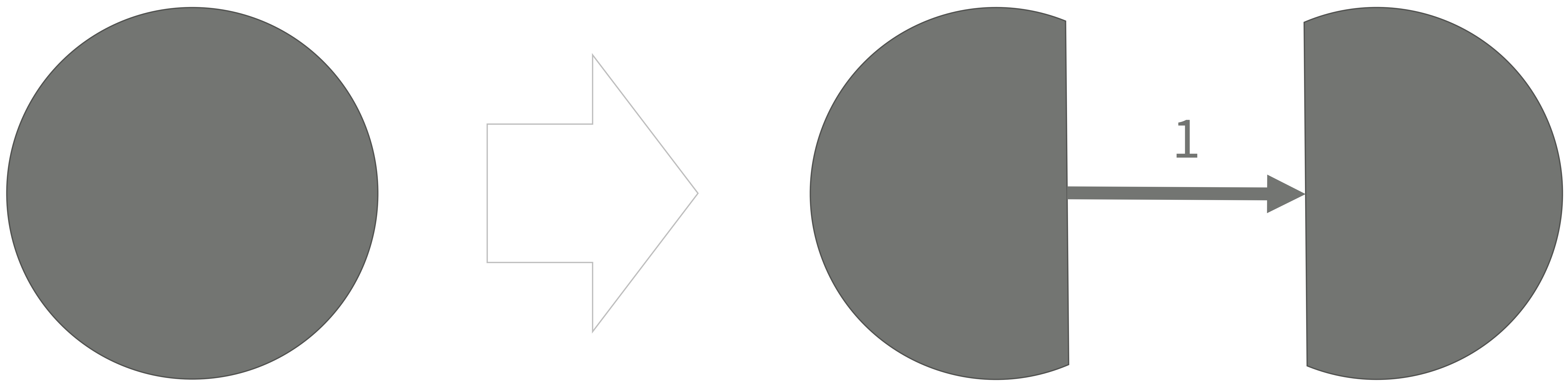
- 정점 X를 Xin과 Xout 으로 나누고, Xin -> Xout은 1로 연결
- X와 Y를 이동할 수 있으면, Xout -> Yin, Yout -> Xin 을 연결



학교 가지마!

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

- 소스: http://codeplus.codes/4982a11c559b44239d0add3a5f87046c



최소 버텍스 커버

최소 버텍스 커버

Minium Vertex Cover

- Vertex Cover: 정점 집합 S가 있을 때, 모든 간선은 양 끝점중 하나가 S에 포함되어야 함
- Minimum Vertex Cover: 최소값

König's theorem

Minium 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

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

돌		돌
	돌	
	돌	

돌멩이 제거

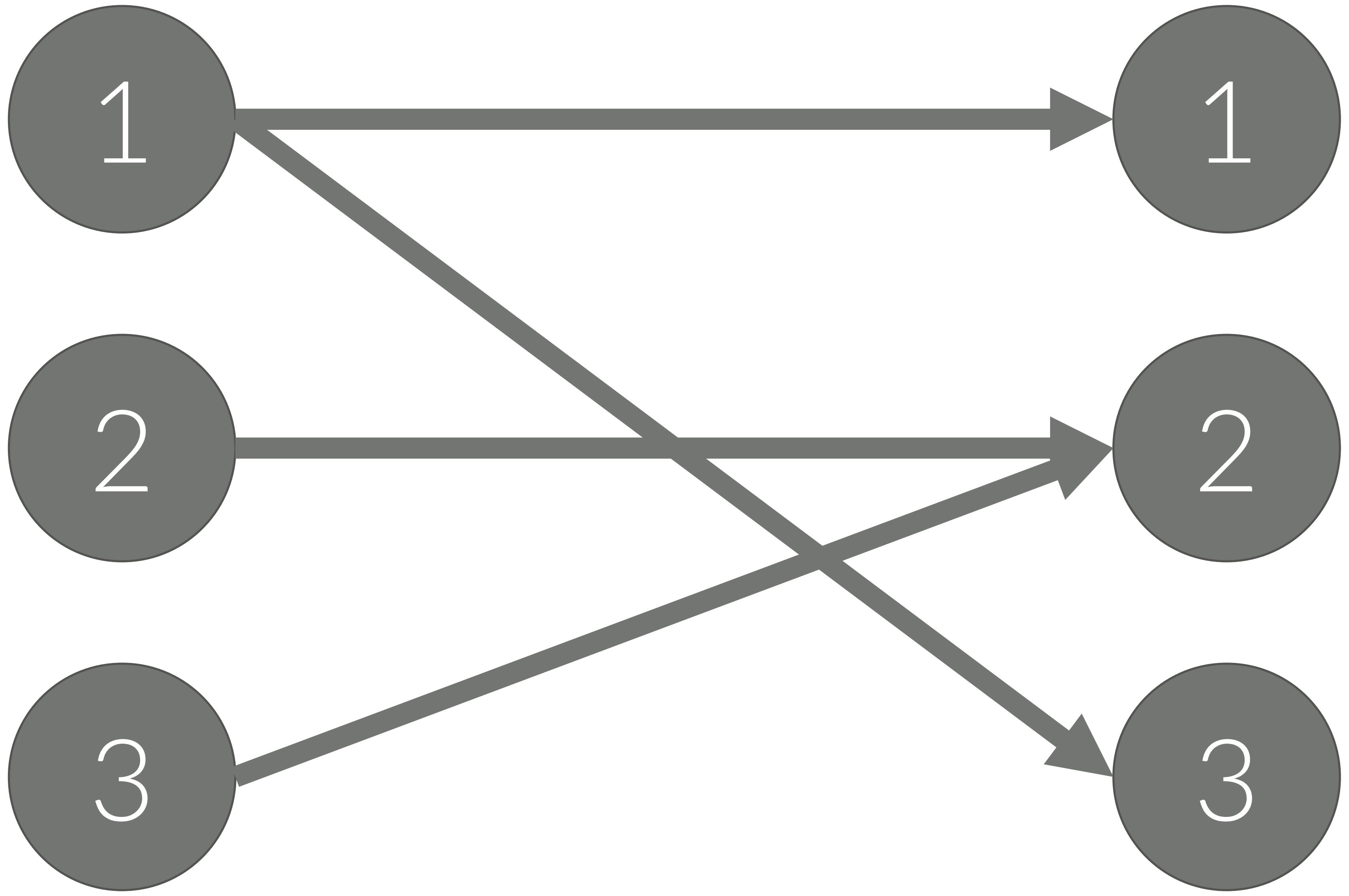
https://www.acmicpc.net/problem/1867

- 이분 그래프를 만든다

- 왼쪽: 행
- 오른쪽: 열
- i행 j열에 돌멩이가 있으면
- 왼쪽 i → 오른쪽 j를 연결

돌멩이 제거

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



돌		돌
	돌	
	돌	

돌멩이 제거

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

- 소스: http://codeplus.codes/427feb209183480e8f2cece022969418

최대 독립 집합

최대 독립 집합

Maximum Independent Set

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

최대 독립 집합

Maximum Independent Set

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

컨닝 2

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

- N행 M열 직사각형 교실에서 시험을 보려고 한다.
- 최대 몇 명의 학생이 시험을 볼 수 있는가?
- $1 \leq N, M \leq 80$

X		X
X	사람	X

컨닝 2

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

- 각 칸을 정점으로 생각하고, 앉을 수 없는 칸을 간선으로 연결하자
- 문제는 이 그래프에서 Maximum Independent Set을 찾는 문제

컨닝 2

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

- 열의 홀/짝을 기준으로 왼쪽과 오른쪽을 나눌 수 있다.

컨닝 2

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

- 소스: http://codeplus.codes/bea6b28db96b412e911717658f52848d