Sogang ICPC Team

2021-1 중급 스터디 7회차

Graphs

임지환
raararaara@gmail.com
2021.5.23

- State Space
  - 입력 크기와 상태 공간
  - Meet in the middle
  - Bidirectional Search
  - Bitwise state
- Shortest Path
  - 0-1 BFS
  - SPFA (Shortest Path Faster Algorithm)
  - Floyd-warshall's Algorithm 응용
- Maximum Flow
  - 유량 그래프
  - Dinic's Algorithm

State Space

- 입력 크기와 상태 공간
- MITM (Meet In the Middle)
- Bidirectional Search
- Bitwise state

- 그래프 문제를 만났을 때

1) 문제 유형?
2) 방법론?

- 그래프 문제를 만났을 때

1) 문제 유형?
2) 방법론?
3) 입력 크기

- 입력 크기에 따라 고려해야 할 요소들

1. 왜 작을까?(혹은 클까?)
2. (내가 생각했던 방법이) 이러한 입력 크기에서도 동작할까?
3. 특정 성질이 있어 경우의 수가 줄어들지는 않을까?

Problem 1

- N($1 \leq N \leq 30$)개의 물건($w_i \leq 10^9$)
- 가방의 허용 무게 C($1 \leq C \leq 10^9$)

- 가방에 물건을 넣는 방법의 수?

- 문제 해결 과정

1. dp인가? $d[n][C]$: n개의 물건을 고려하여 무게 C를 만드는 경우의 수?

   어림도 없다. C가 문제다

2. C 자체를 요인에서 배제
3. 완전 탐색? $2^{\mathrm{N}} \leq 2^{30}\ldots$

- 중간 만남 기법

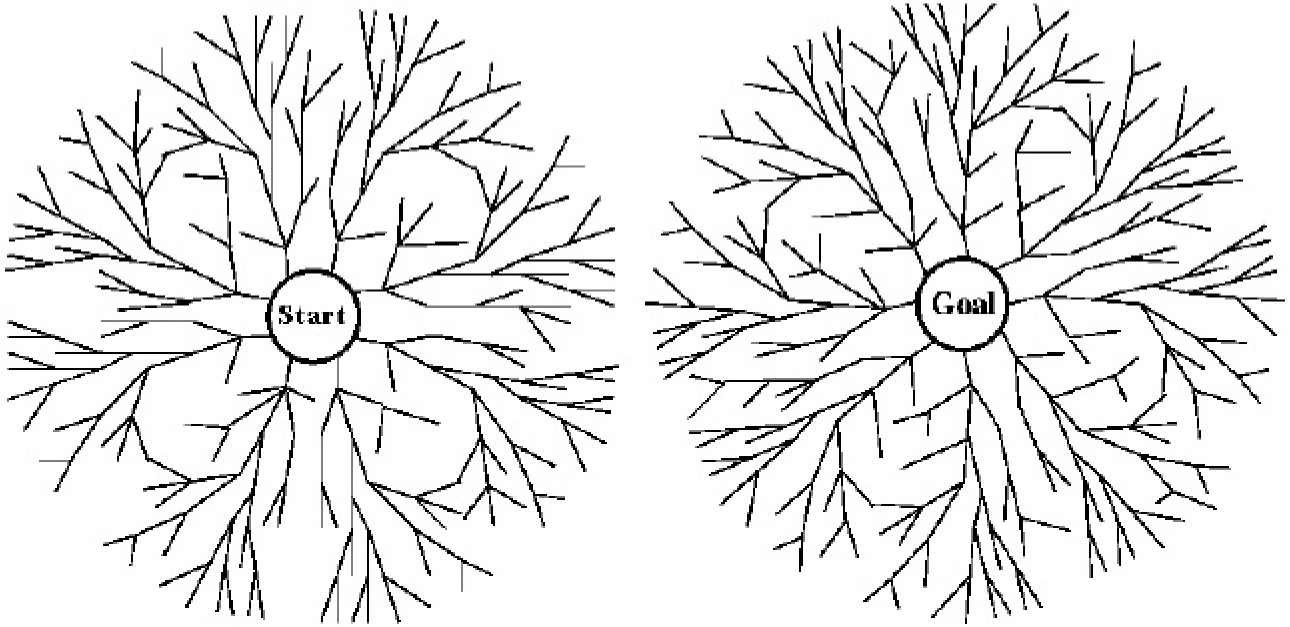
탐색 공간을 유사한 크기의 두 부분으로 나누고, 각 부분에 대해 독립적으로 탐색을 수행한 후 결과를 조합하여 답을 내는 기법

- 문제 해결 과정

1. N개의 물건을 두 그룹으로 나눈다.
2. 그룹마다 나올 수 있는 무게의 경우의 수를 구한다. (각각 2^{15}개의 경우의 수)
3. 한 그룹의 각각의 경우(w_L)에 대하여, $C - w_L$이하인 경우를 반대쪽에서 count

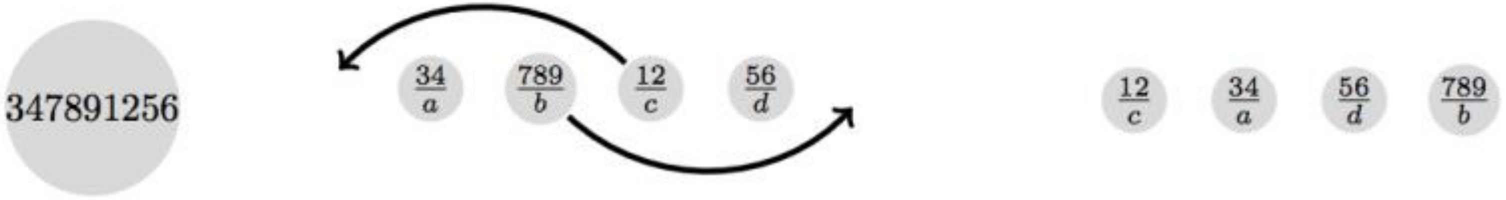
- 시작 노드와 목표노드에서 동시에 탐색 시작

- b: branching factor
- d: depth of solution
- Time & Space Complexity: $O(b^{d/2})$ each



Problem 2

- 1~n $(1 \le n \le 10)$으로 구성된 permutation
- swap 방식:



- permutation을 오름차순으로 만드는데 드는 최소 swap 횟수

- 문제 해결 전략

1. 상태 공간의 개수: 10! (= 3,628,800 = $|V|$)
2. 하나의 state에서 다른 state로의 전이: ${}_{10}H_3$ (= branching factor = 220)
3. $|E| = |V| \times |b| = 798,336,000$

- 문제 해결 전략

4. Official solution:

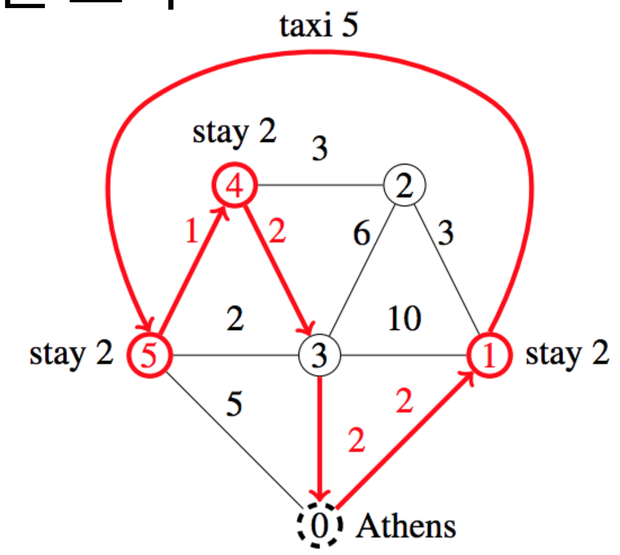
   The maximum number of steps required is at most 6. (Can be confirmed by naïve exploration in a few minutes.)

5. set d(depth of solution) by 6,

   Time & Space Complexity: 220^3 = 10,648,000

Problem 3

- N ($1 \le N \le 20{,}000$)개의 도시, P ($1 \le P \le 15$)개의 방문하고자 하는 도시
- M ($1 \le M \le 10^5$)개의 간선
- 여유 시간 G ($1 \le G \le 10^5$)
- 방문하고자 하는 도시마다 체류 시간 $t_i (1 \le t_i \le 500)$ exists
- 단 한번의 택시 탑승 기회, 택시 비용 T($1 \le T \le 500$)
- 간선 가중치(이동 시간) $t_i (1 \le t_i \le 500)$



- 모든 방문하고자 하는 도시를 방문하고 원래 자리로 돌아온다 할 때:

➢ 택시를 타지 않고 도착할 수 있을까?

➢ 택시를 타고 도착할 수 있을까?

➢ 택시를 타도 도착할 수 없을까?

Problem 3

- 문제 해결 전략

1. 방문하고자 하는 도시의 수가 적음
2. 다 돌고 제자리로 돌아와야 함
3. by 1&2 => TSP(Traveling Salesman Problem)
4. 방문 희망 지점간 거리 handling (by Dijkstra's)
5. 택시 탑승 여부

Shortest Path

- 0-1 BFS
- SPFA (Shortest Path Faster Algorithm)
- Floyd-Warshall's Algorithm 응용

- Single Source Shortest Path at:

$\Rightarrow$ unweighted graph: BFS, $O(|V| + |E|)$
$\Rightarrow$ weighted graph: Dijkstra's, $O(|E| \log|V|)$ (non-negative edges)
Bellman-Ford's, $O(|V||E|)$

- 가중치가 0 또는 1(또는 균일)한 경우 현재 state로부터 다음 state까지의 거리:

1) 유지
2) 1 증가

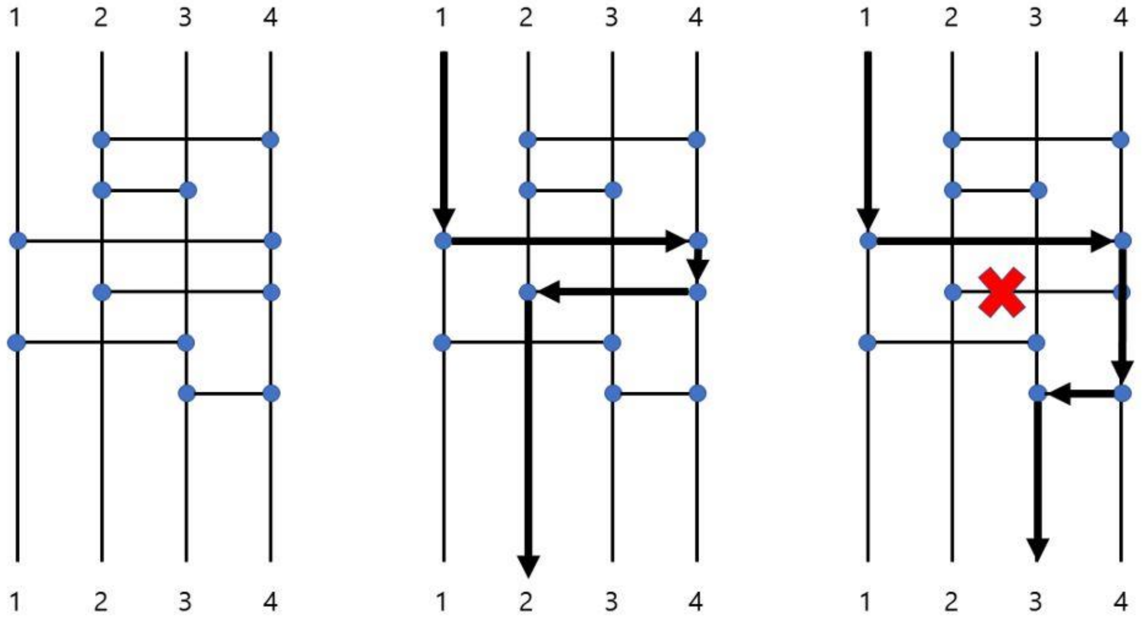
- BFS에서 queue 내부의 노드들에 대한 거리: increasing order

- use deque instead of queue-> $O(|V| + |E|)$ at weighted graph

Problem 4

- $N(2 \leq N \leq 1500)$개의 세로선, $k(1 \leq k \leq 2000)$개의 가로이음선
- $m(1 \leq m \leq 10^5)$개의 출발점과 도착점 쌍의 개수

- 출발점에서 도착점을 가기 위해 무시해야 하는 가로선의 최소 개수들의 합
- 테스트케이스가 70개 이하



- 문제 해결 전략

1. 사다리에서 각 point를 노드로,
2. 가로선을 타고 이동하는 경로의 가중치를 0, 무시하고 내려가는 경로의 가중치를 1
3. 시작점마다의 쿼리를 묶어서 처리 -> N번의 0-1 BFS, $O(N \cdot 2k)$

$$TC \cdot (N \cdot 2k + m) = 427{,}000{,}000$$

Recall: Bellman-Ford's Algorithm

✓ 음수 가중치에서도 동작 가능
✓ $|V| - 1$개 이하의 간선에 의해 최단 경로가 보장이 된다면
✓ $|V| - 1$번의 relaxation으로 최단경로 갱신

```
BELLMAN-FORD(G, w, s)
INIT-SINGLE-SOURCE(G,s)
for i = 1 to |G.V| - 1
  for each edge (u,v) in G.E
    RELAX(u,v,w)
for each edge (u,v) in G.E
  if v.d > u.d + w(u,v)
    return FALSE
return TRUE
```

- Shortest Path Faster Algorithm

✓ '모든 간선에 대해'가 아닌, 바뀐 정점과 연결된 간선에 대해서만 업데이트

✓ more sparse, faster algorithm, $O(|V|+|E|) \le T(V,E) \le O(|V||E|)$

```
SPFA(G, s)
for each vertex v != s in G.V
  v.d := inf
s.d := 0
offer s into Q
while Q is not empty
  u := poll Q
  for each edge (u,v) in G.E
    if u.d + w(u,v) < v.d then
      v.d := u.d + w(u,v)
      if v is not in Q then
        offer v into Q
```

Problem 5

- $N(1 \leq N \leq 200)$종류의 코인
- 코인마다의 판매가가 달라 환전 비율이 다른 상황
- $M(1 \leq M \leq N(N-1)/2)$개의 환전 비율 (ETH BTC 0.06: 1ETH로 0.06 BTC 구매 가능)

- 거래소 내에서 코인을 잘 굴렸을 때 수익을 발생시킬 수 있는가?

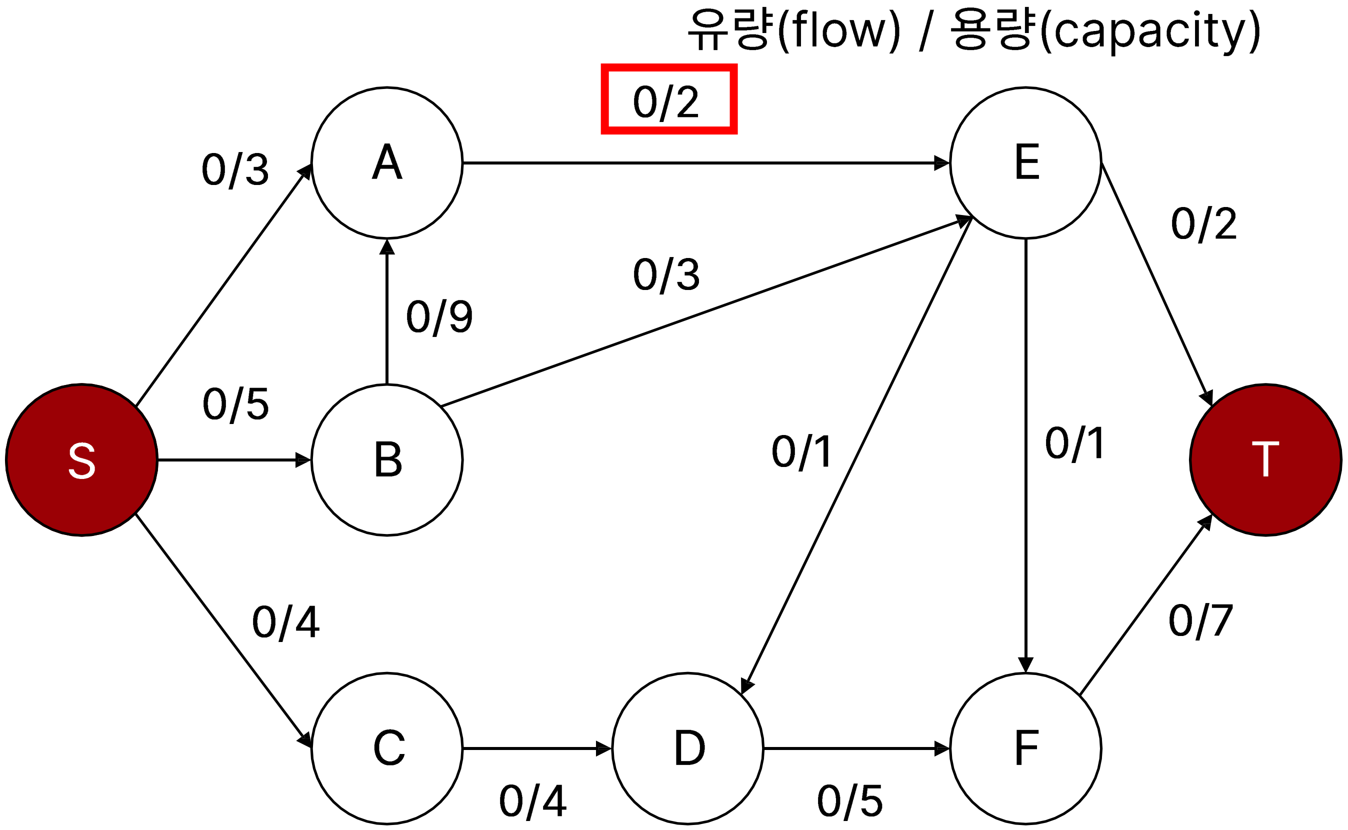
- 문제 해결 전략

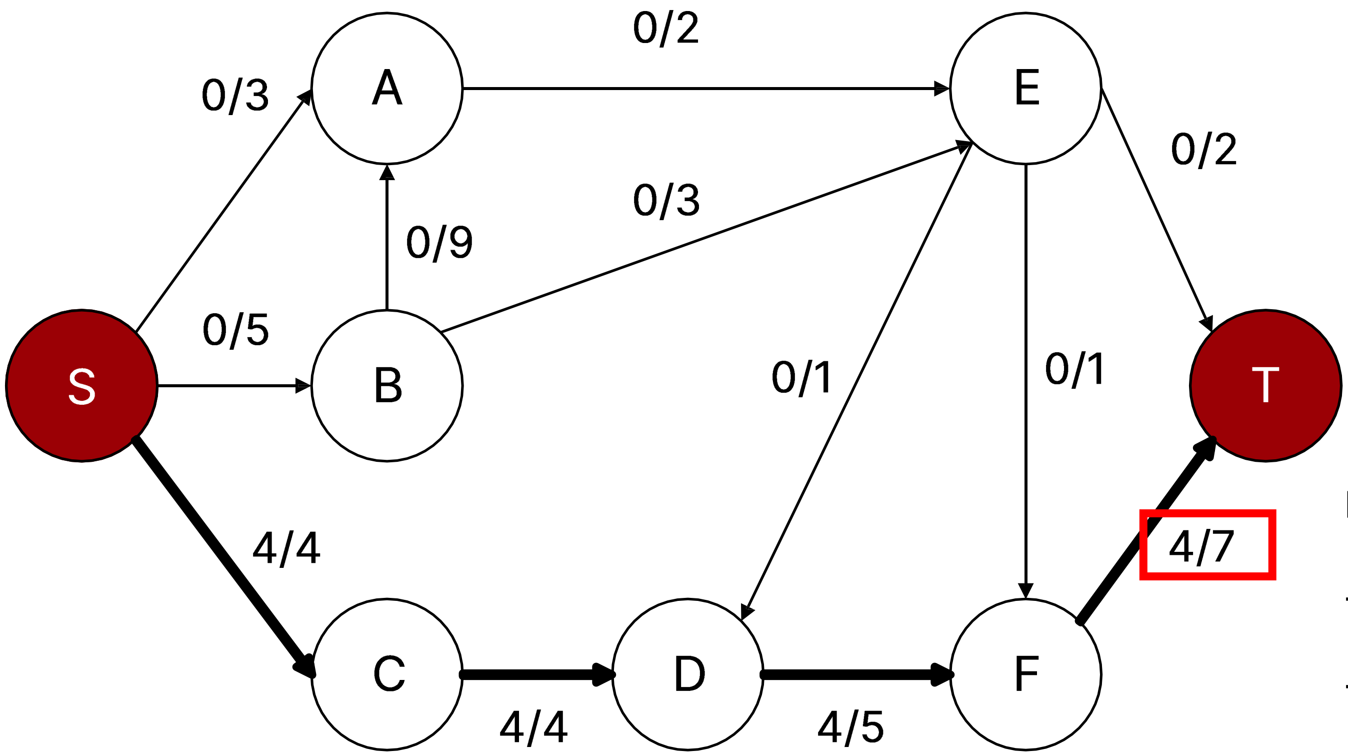
1. 노드 수가 많지 않으므로 All-pair Shortest Path 고려
2. 덧셈 형태가 아닌 곱셈의 경우 -> 가중치에 log를 씌워 덧셈으로 변환
3. 최솟값이 아닌 최댓값을 구해야 하기 때문에 log + 음수화

Maximum Flow

- 유량 그래프
- Dinic’s Algorithm

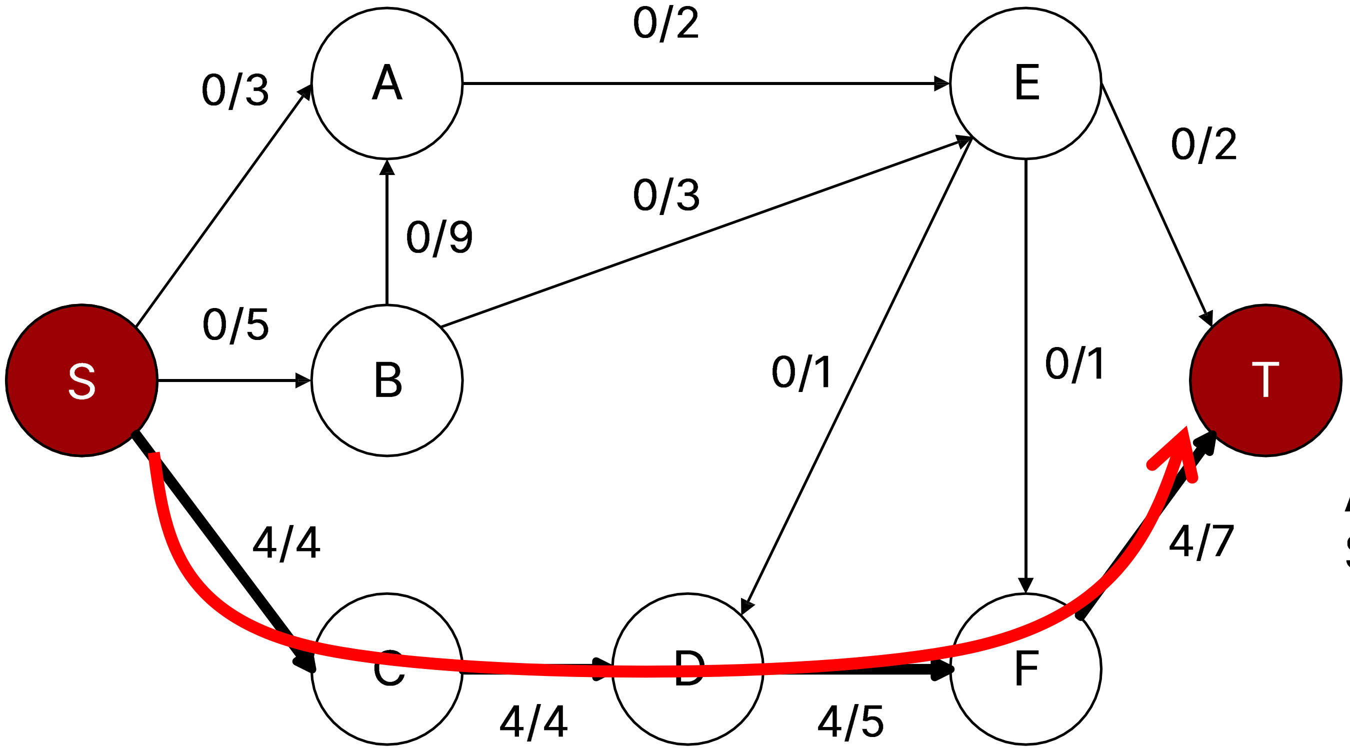
- 유량 그래프: 그래프의 각 간선(edge)에 용량(capacity)과 현재 흐르고 있는 유량(flow)이 주어진 방향 그래프



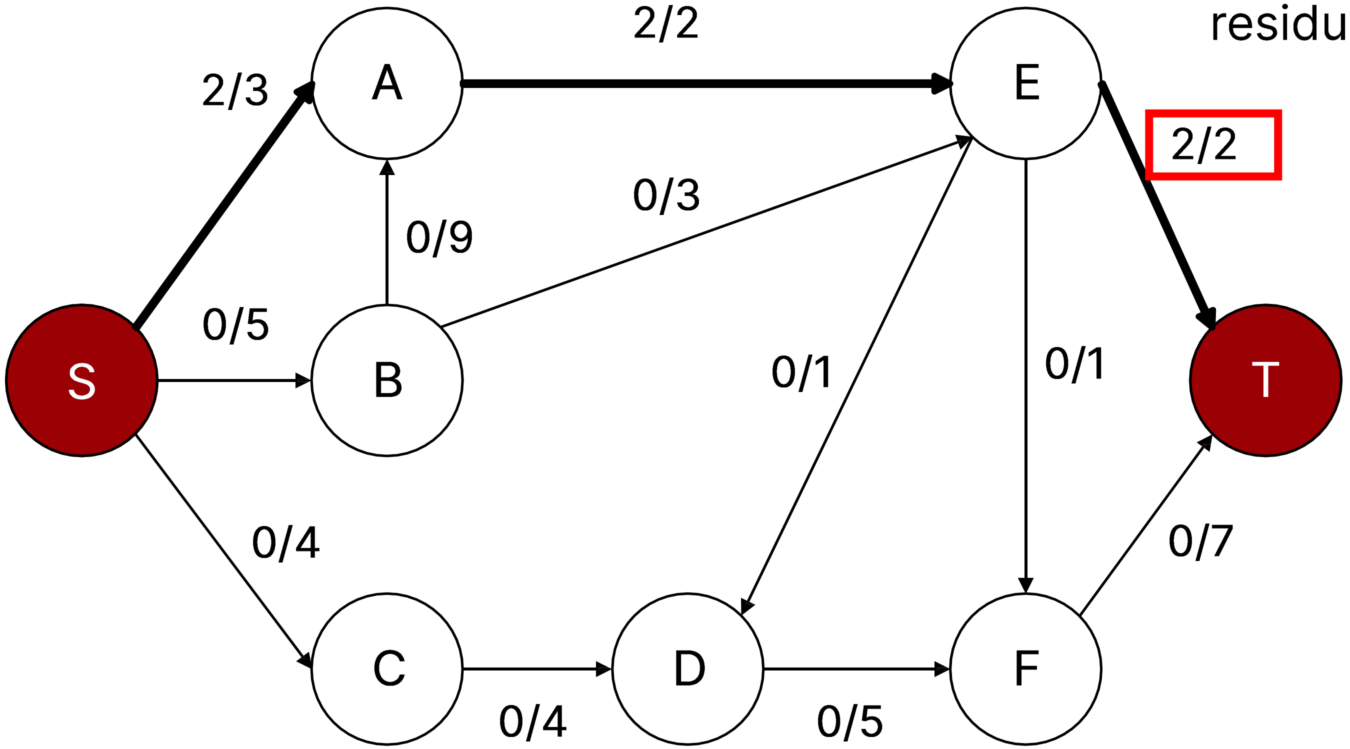


residual capacity := capacity – flow

현재 상태에서 추가로
흐를 수 있는 양

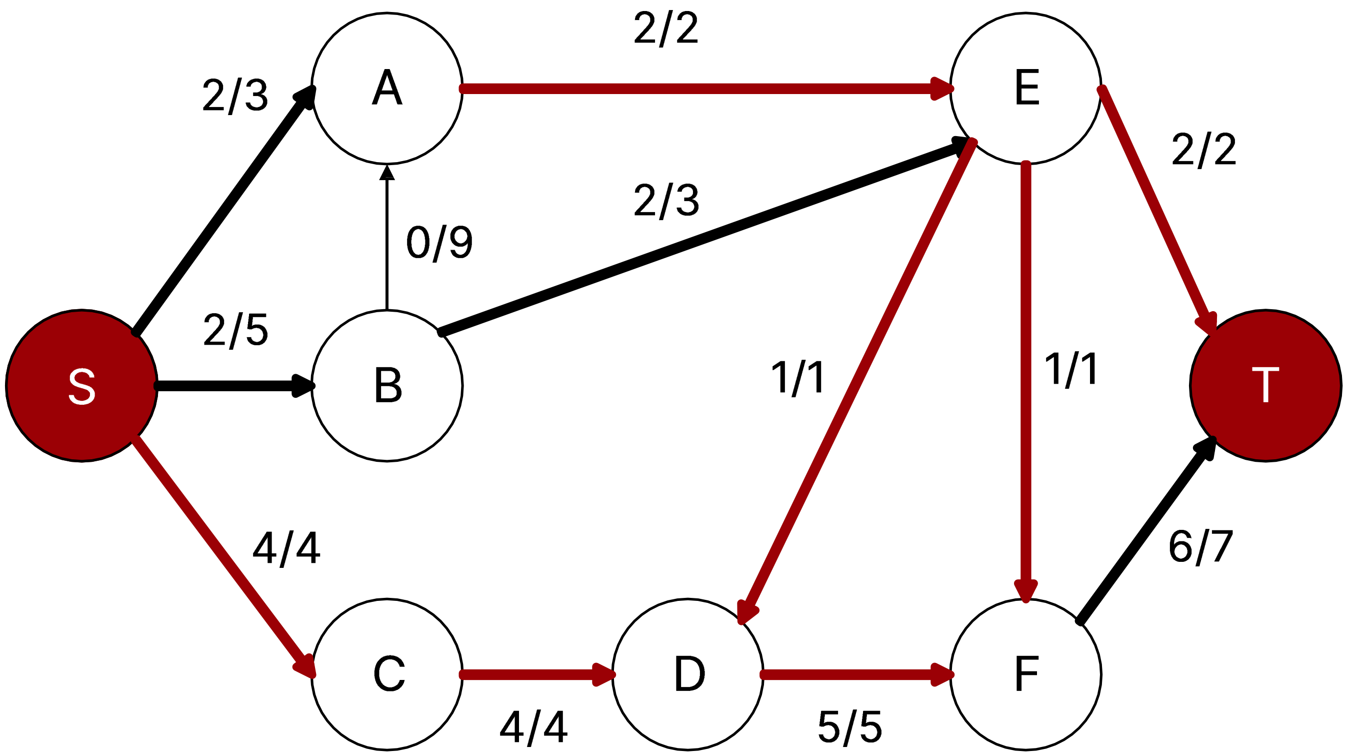


Augmenting Path(증가 경로)
S->T로 갈 수 있는 경로



Blocking Flow
증가경로에서 residual capacity가 가장 낮은 edge의 residual capacity

- Total Flow: 8 => Maximum flow



Properties of Network Flow

- 유량 보존

$$\sum_{v \in V} f(u,v) - \sum_{v \in V} f(v,u) = 0$$

- 유량의 대칭성

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

- 용량 제한 속성

$$f(u,v) \leq c(u,v)$$

- TODO: Source & Sink가 주어졌을 때 흘려보낼 수 있는 최대 유량

- Solving problem by:

1) Ford Fulkerson Algorithm
2) Edmond-Karp Algorithm
3) Dinic Algorithm

- Given Flow Graph:

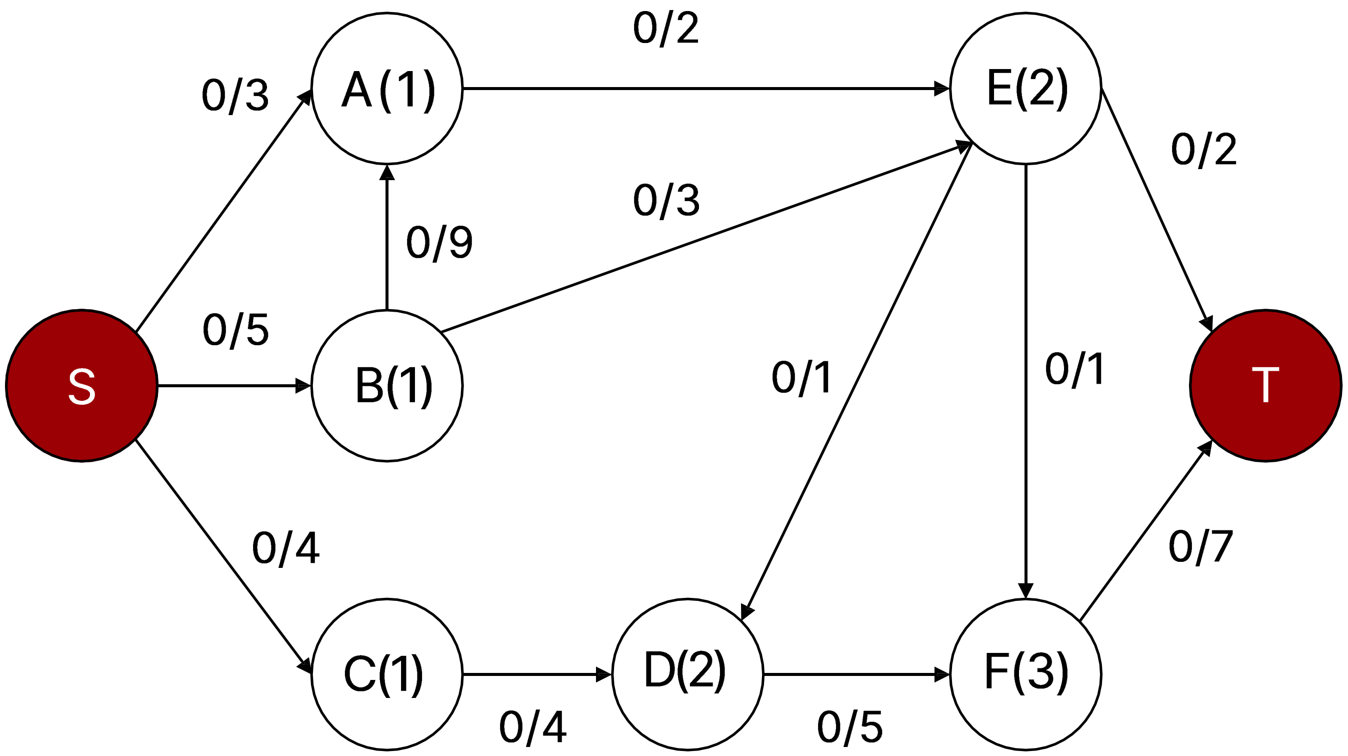
1. BFS로 Level Graph 구축
2. 찾은 Augmenting Path들에 대해 Blocking Flow만큼 모두 Flow를 흘려 보냄
3. Level Graph에서 Sink에 도달할 수 없을 때까지 1,2를 반복

- How Fast?

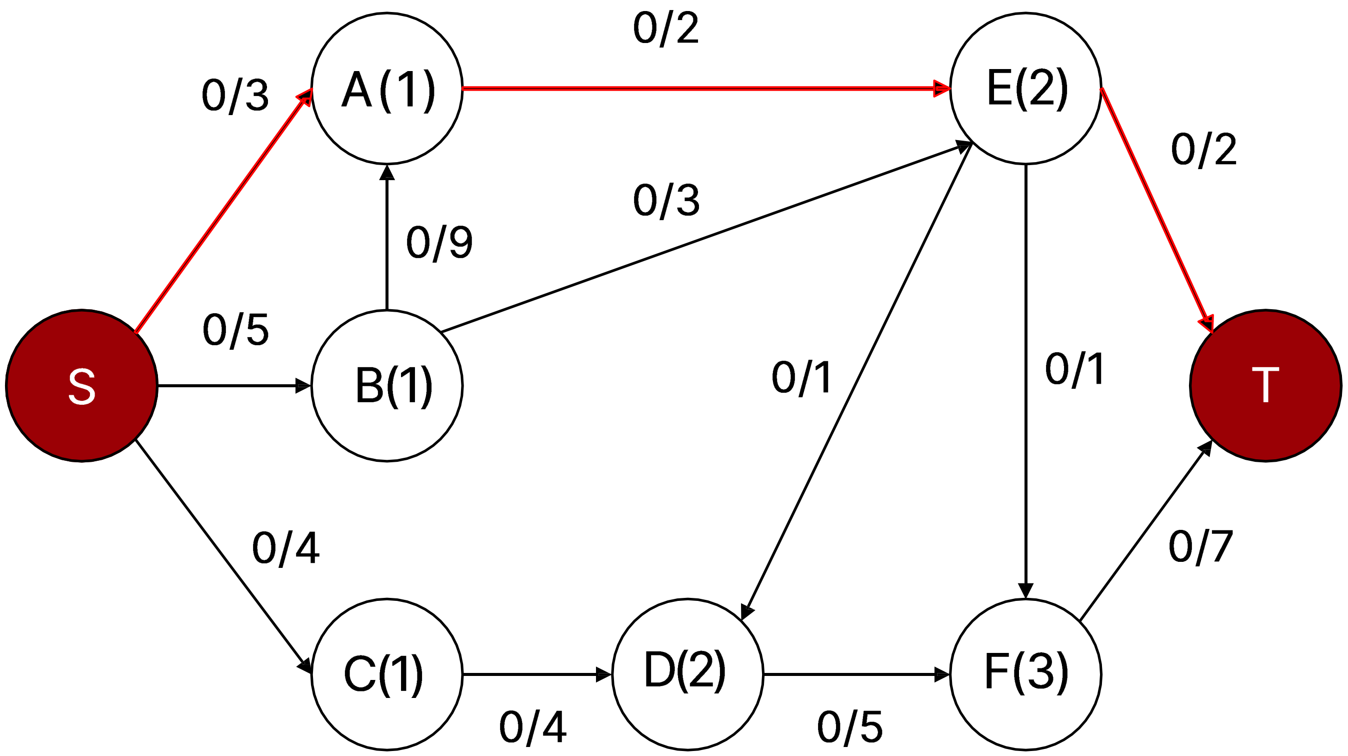
✓ $O(V^2E)$, generally $O(VE)$

code here

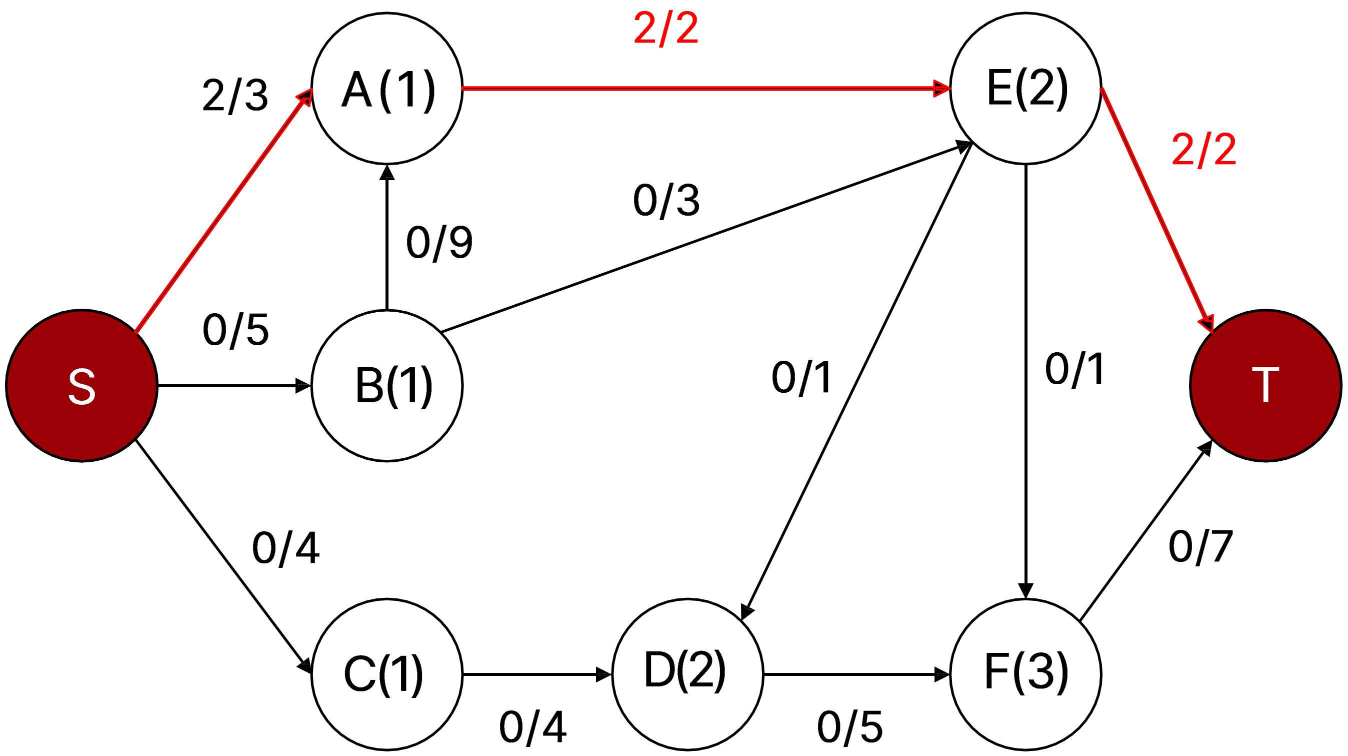
- BFS로 Level Graph 구축



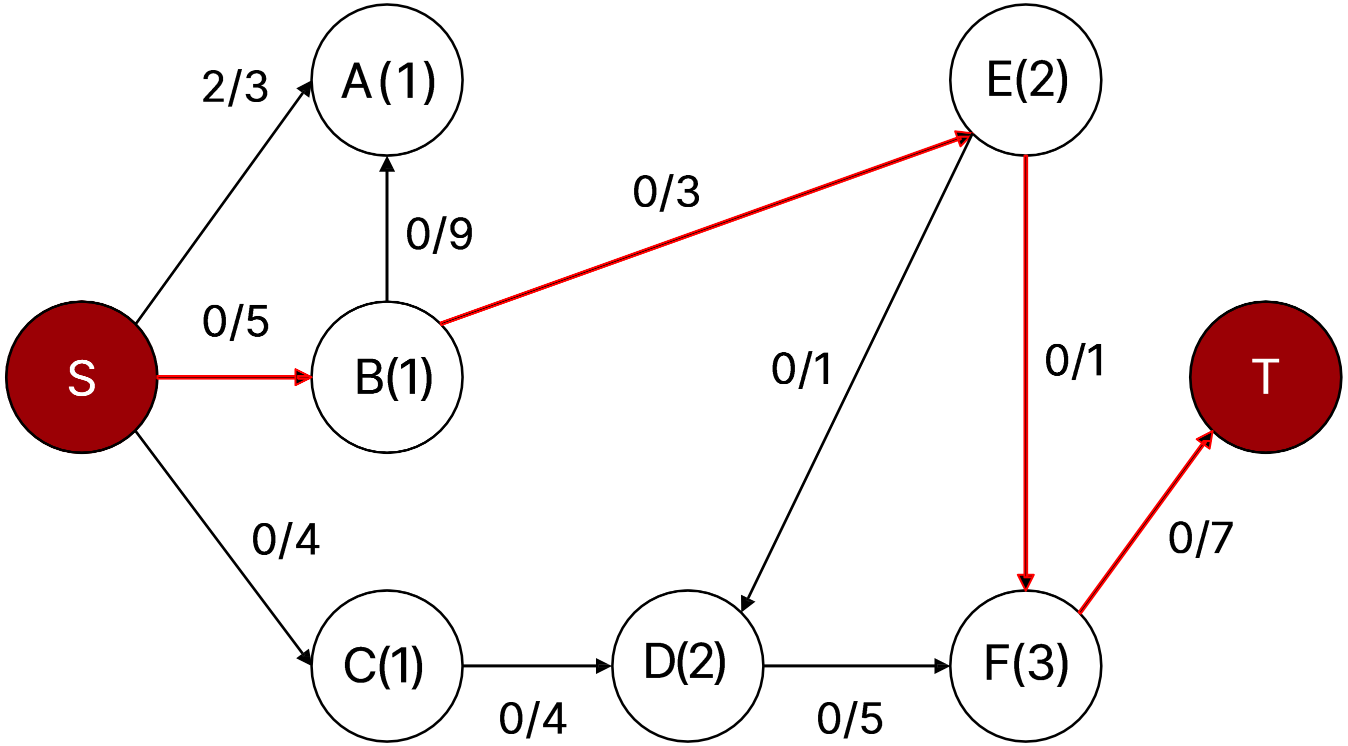
- DFS를 통해 Augmenting Path로 Blocking Flow를 찾아 흘려 보냄



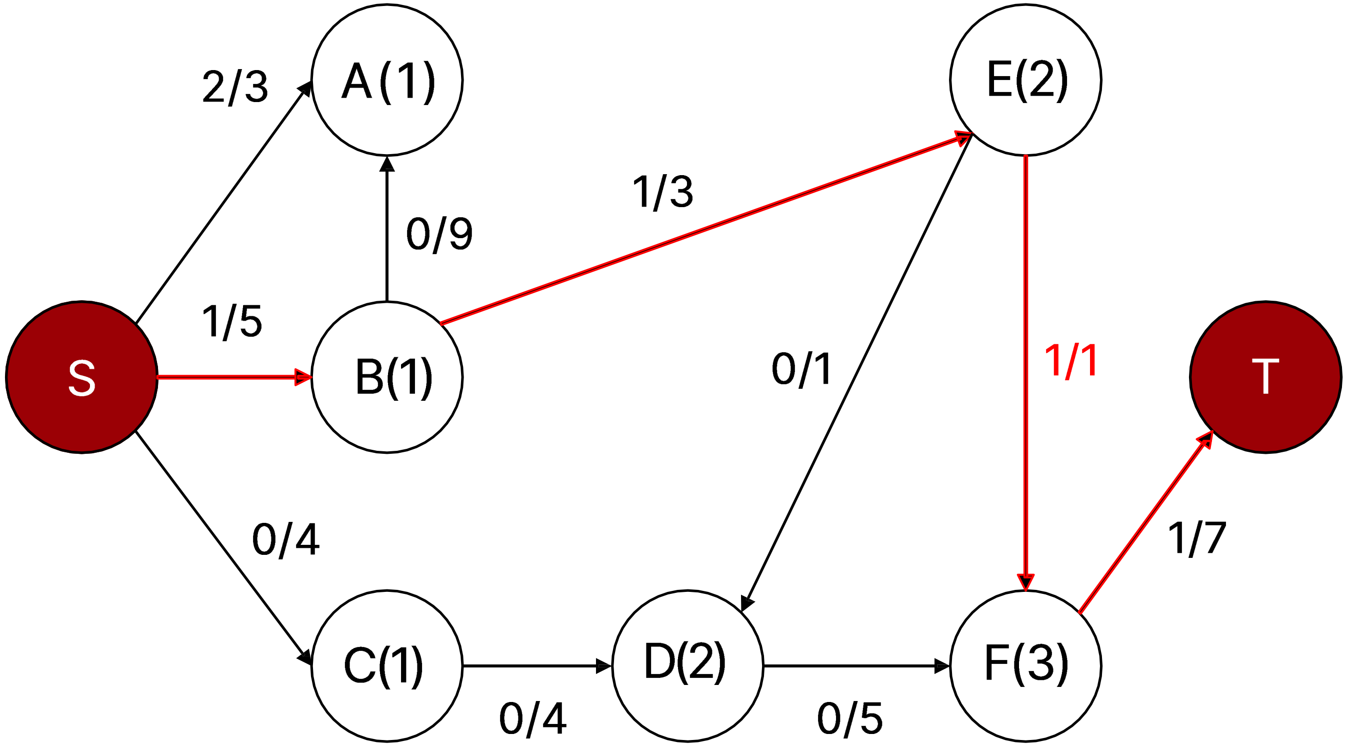
- DFS를 통해 Augmenting Path로 Blocking Flow를 찾아 흘려 보냄



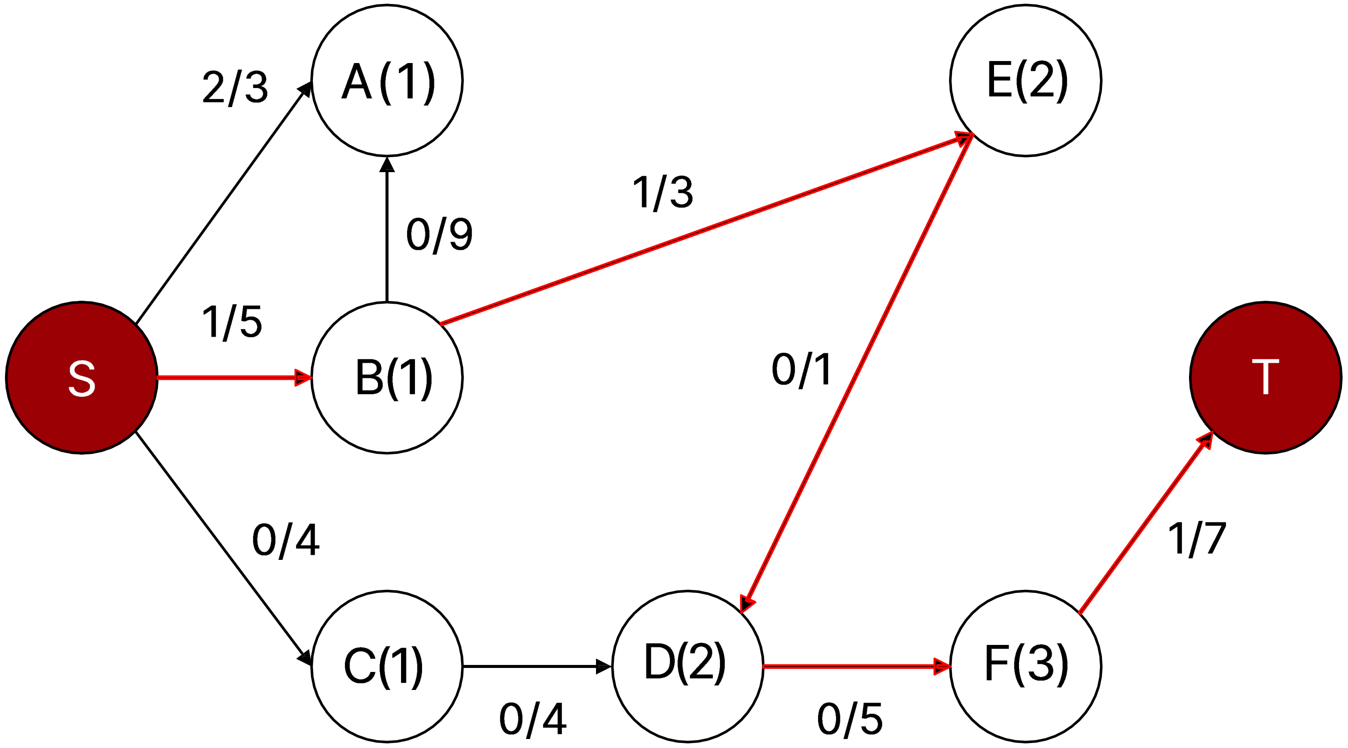
- DFS를 통해 Augmenting Path로 Blocking Flow를 찾아 흘려 보냄



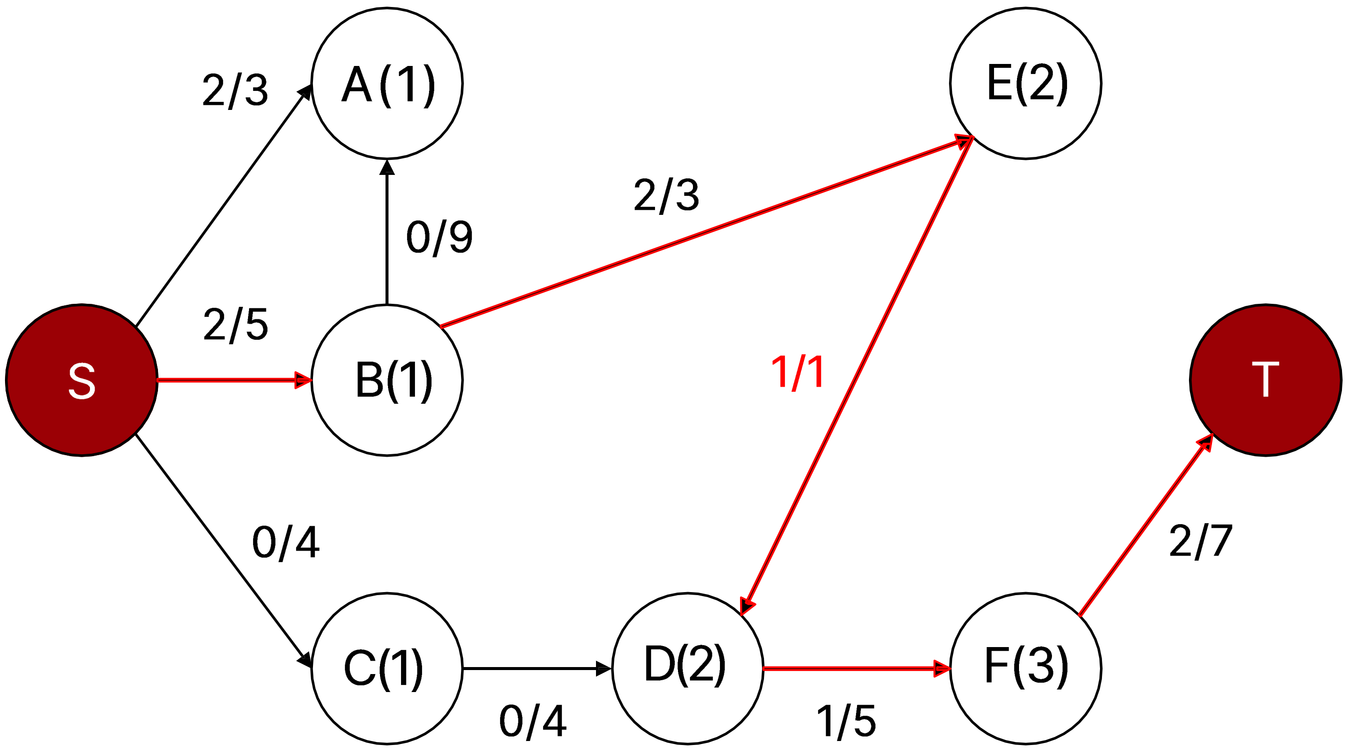
- DFS를 통해 Augmenting Path로 Blocking Flow를 찾아 흘려 보냄



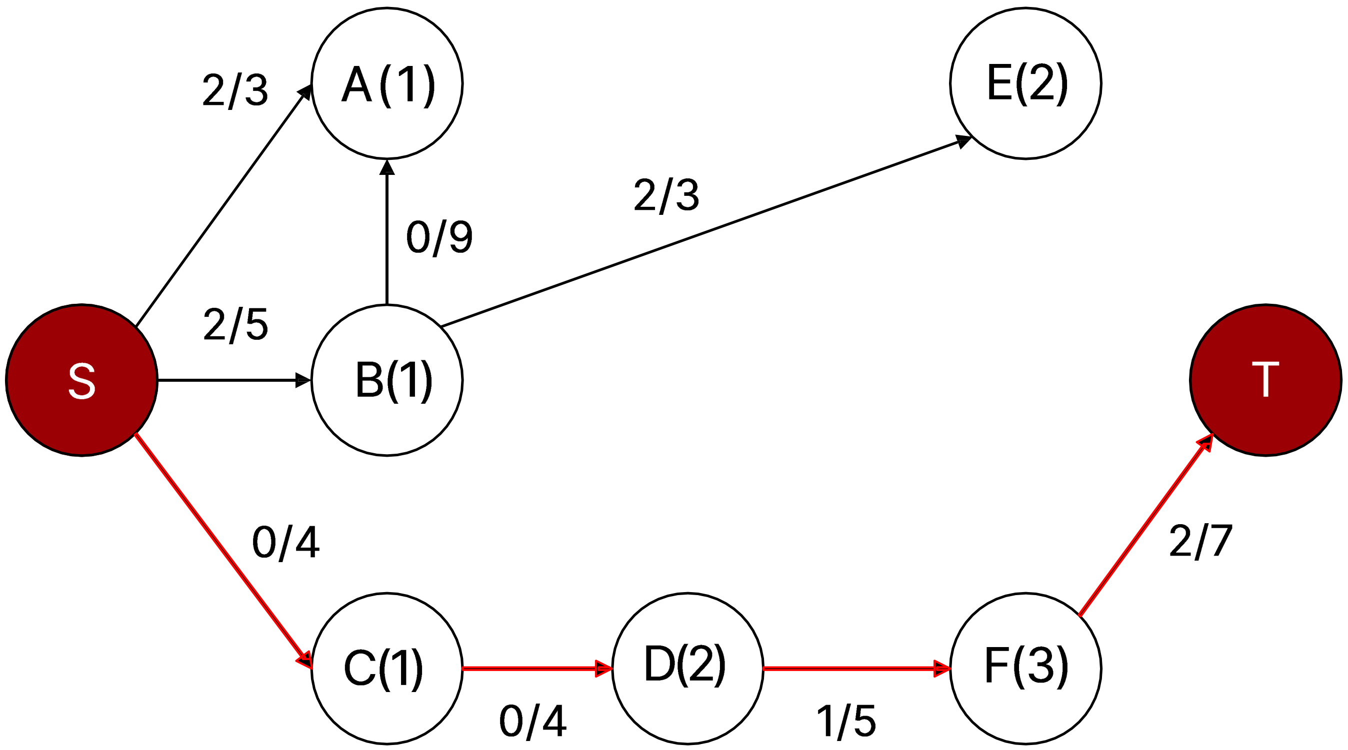
- DFS를 통해 Augmenting Path로 Blocking Flow를 찾아 흘려 보냄



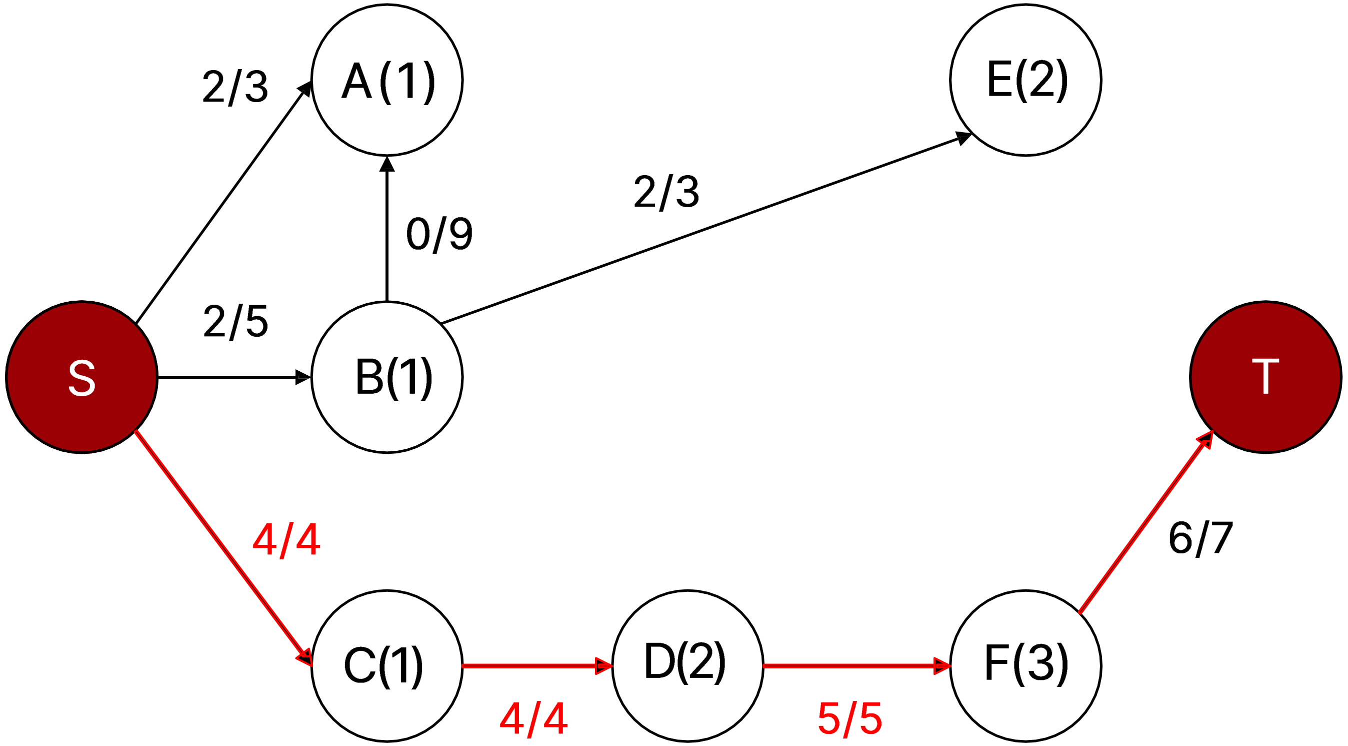
- DFS를 통해 Augmenting Path로 Blocking Flow를 찾아 흘려 보냄



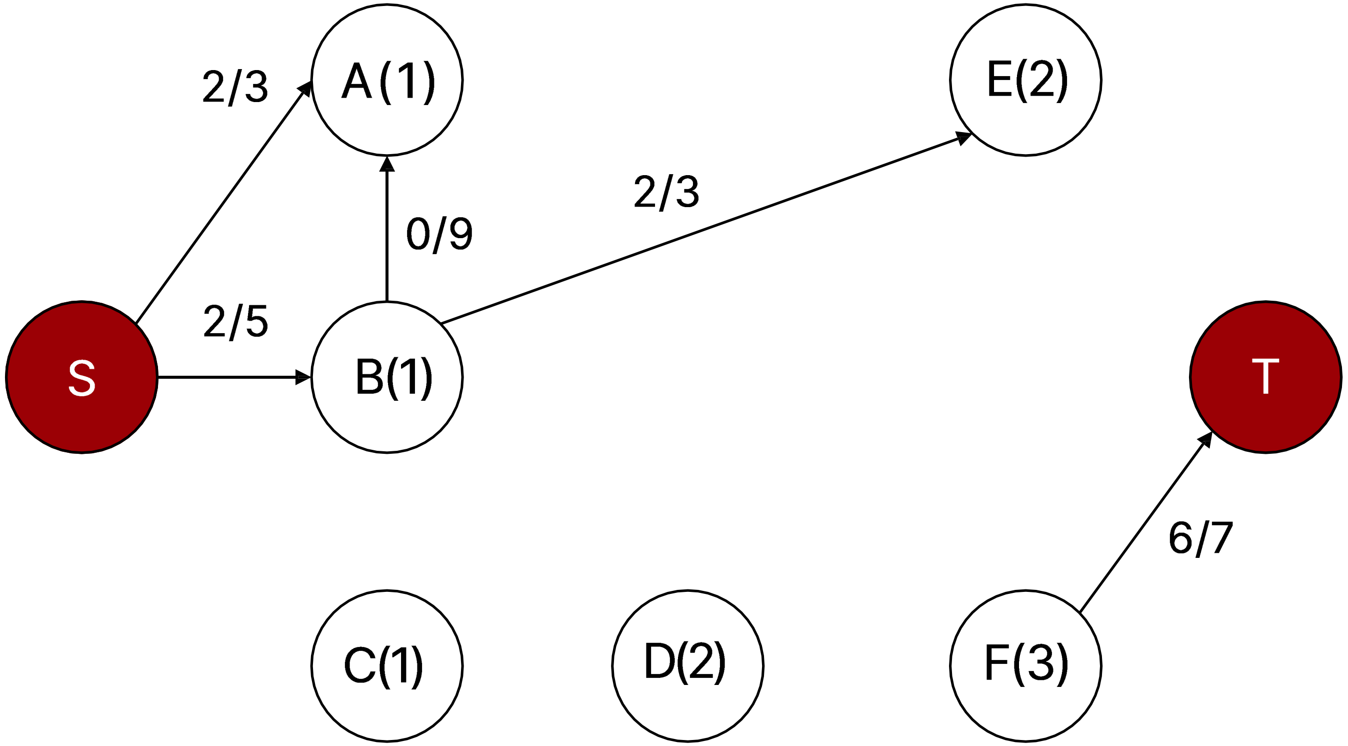
- DFS를 통해 Augmenting Path로 Blocking Flow를 찾아 흘려 보냄



- DFS를 통해 Augmenting Path로 Blocking Flow를 찾아 흘려 보냄



- DFS를 통해 Augmenting Path로 Blocking Flow를 찾아 흘려 보냄



Theorems

- https://koosaga.com/18
- https://koosaga.com/133

- State Space
  - 입력 크기와 상태 공간
  - Meet in the middle
  - Bidirectional Search
  - Bitwise state
- Shortest Path
  - 0-1 BFS
  - SPFA (Shortest Path Faster Algorithm)
  - Floyd-warshall's Algorithm 응용
- Maximum Flow
  - 유량 그래프
  - Dinic's Algorithm