Max Flow and Min Cut

Max Flow와 Min Cut 문제의 관계 이해하고 Baseball Elimination 문제 해결에 적용해 보기

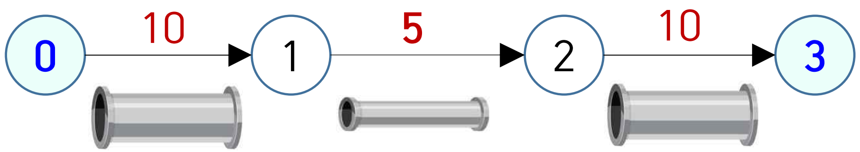
01. Max Flow 문제 정의 및 예습자료 주요내용 복습
02. Ford-Fulkerson 알고리즘
03. Min Cut 문제 정의 및 Max Flow 문제와의 연관성
04. Maxflow-mincut 활용 예: Baseball Elimination 문제
05. 실습: Baseball Elimination 구현

이론 수업 중 실습을 병행하며 진행하므로
첨부 코드를 미리 다운 받아 실행 가능하게 준비해 두세요.

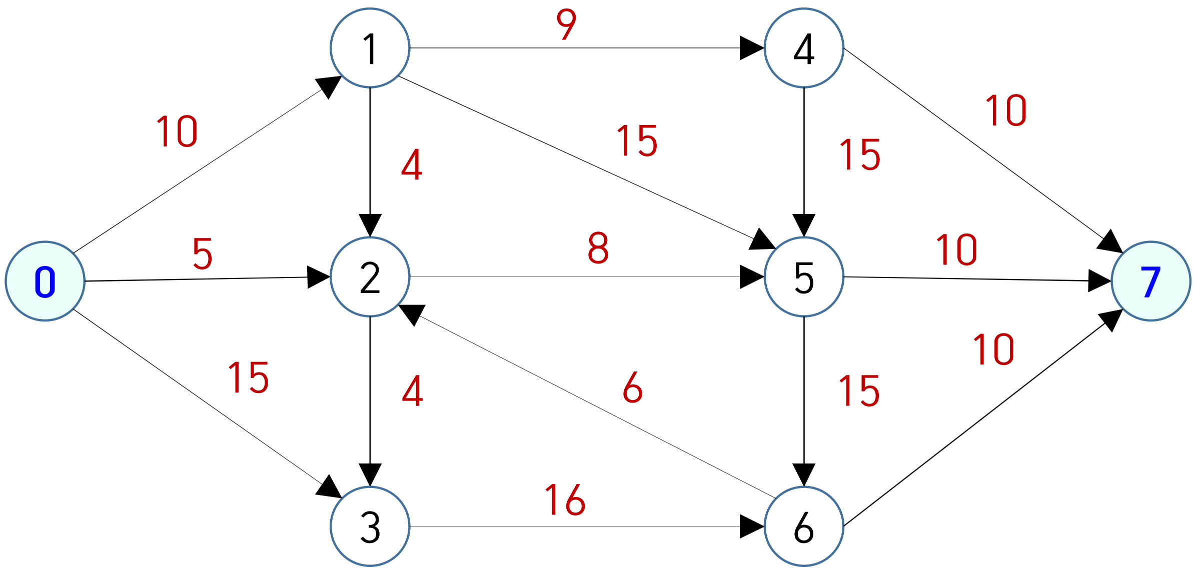
Max Flow 문제의 입력

- Edge-**weight**ed Digraph
- 간선의 weight: (거리 아닌) flow 흐를 수 있는 최대 량 **capacity** 나타냄 ≥0

음수가 아닌 간선

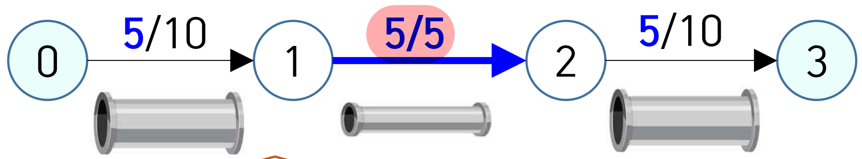


- Flow의
- **출발지 s**
- **도착지 t**



Max Flow: 출발지 s → 도착지 s로 흐를 수 있는 flow의 최대량

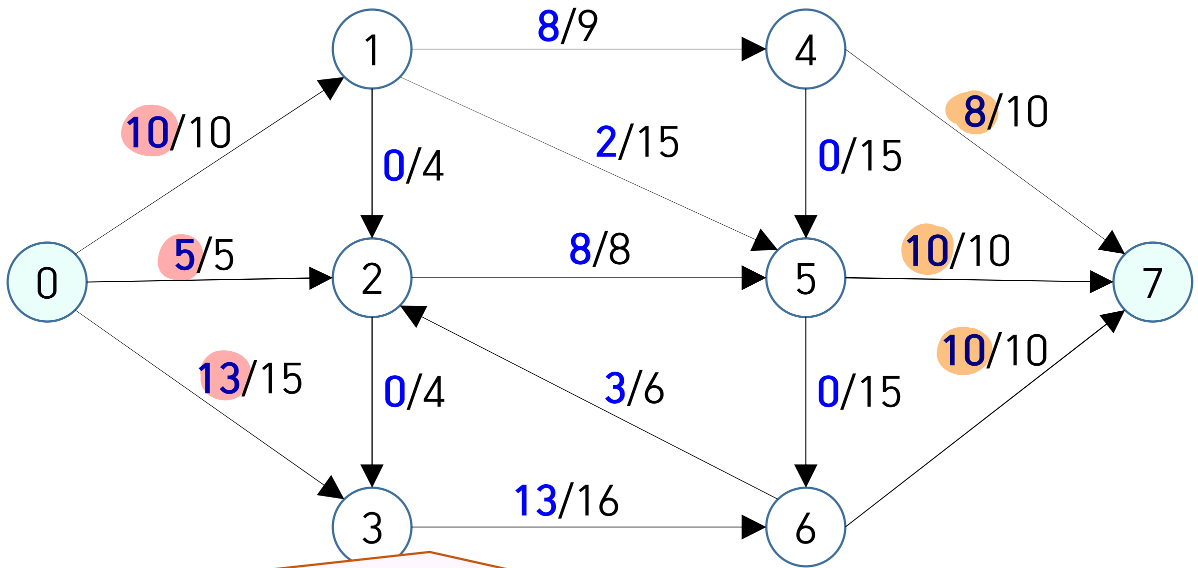
max flow = 5



: Over

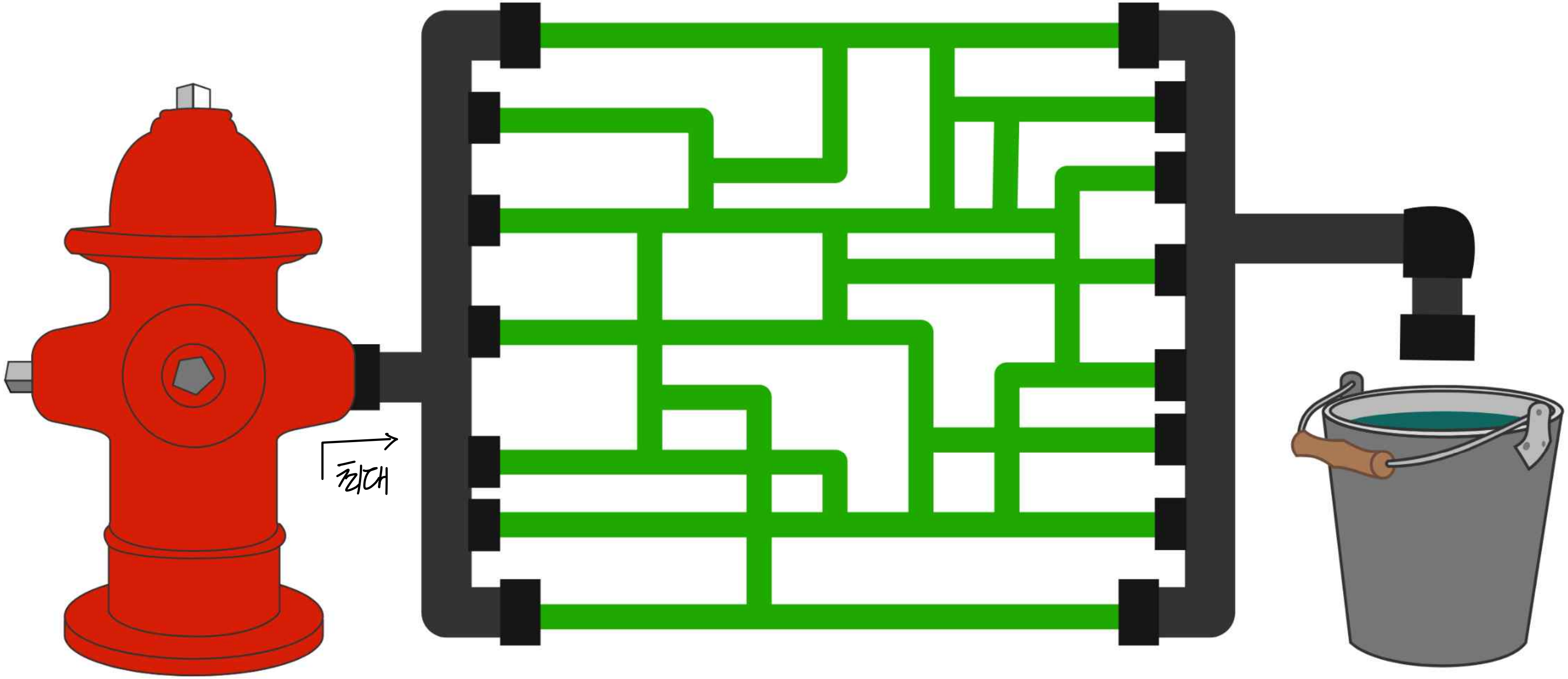
간선을 흐르는 flow / capacity

max flow = 28
= 10+5+13



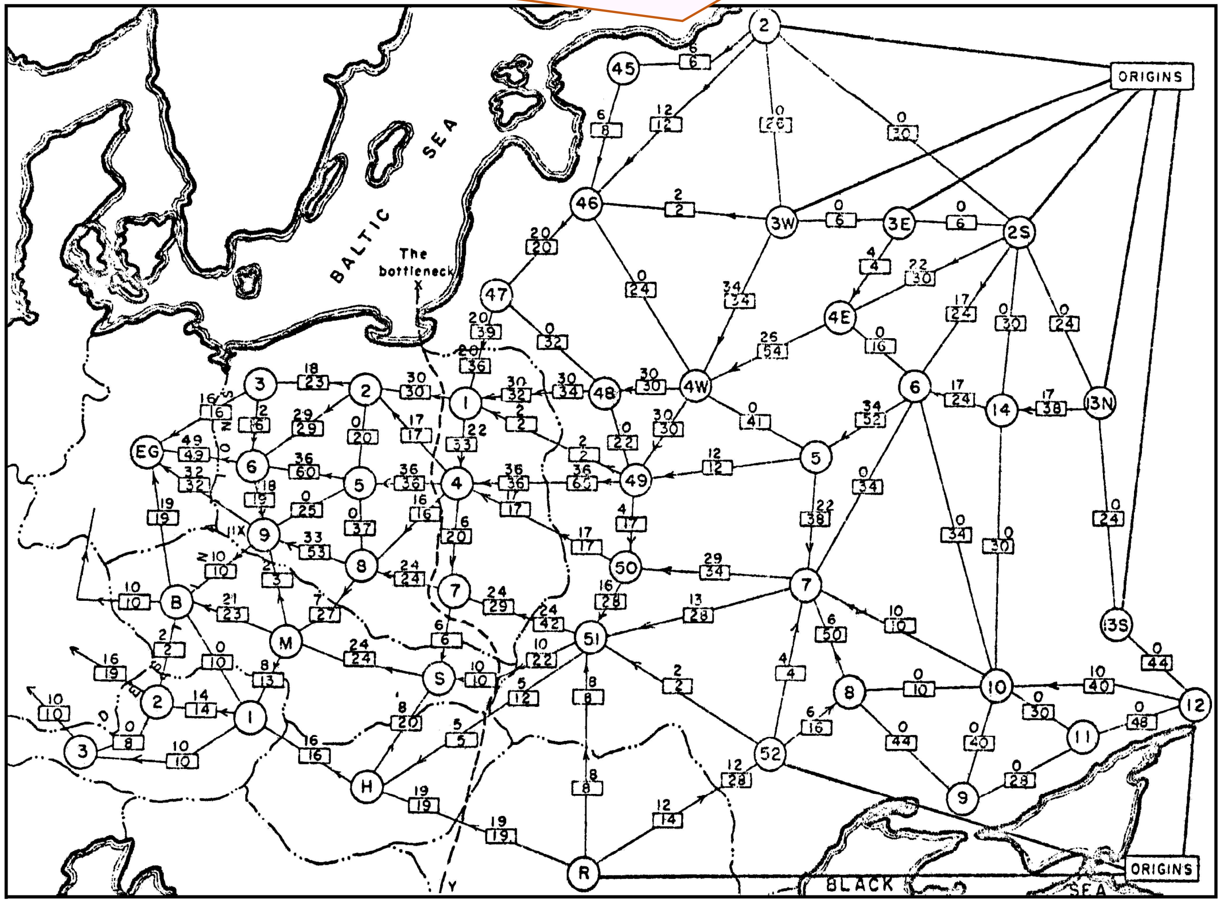
max flow = 28
= 8+10+10

'(1) 최대량'에 더해 '(2) 그러한 최대량으로 보내려면 어느 간선 통해 어느 정도 flow 흘려야 하는가'도 함께 필요한 경우도 많음



https://brilliant.org/wiki/max-flow-min-cut-algorithm/

전쟁 중 후방 → 전방으로 지속적으로 물자 전송 필요
최대로 보내려면 어느 철도선 따라 얼마만큼 보내면 될까?



<러시아와 동유럽 국가 간 철도 연결도 및 운송량>

- 비교적 명백히 max flow인 활용 예

- 전쟁 중 물자 수송: 두 지점 간 수송 가능한 최대 수송량과 경로는?

- 컴퓨터망 연결: 인터넷에 연결된 두 지점 간 전송 가능한 최대 데이터 량은?

- 연결선 추가: 컴퓨터망에서 두 지점 간 데이터 전송량을 증가시키려면 어느 부분에 링크를 추가 개설하는 것이 가장 적은 비용으로 큰 효과를 얻을 수 있을까?

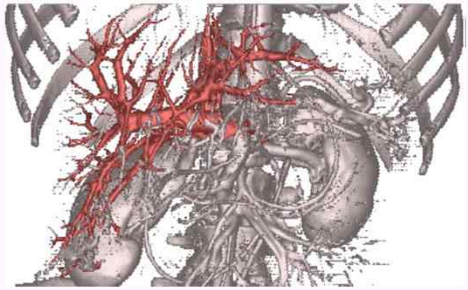
- …

- **명백히 max flow이지 않은** 활용 예

- 짝짓기 문제(Bipartite Matching): 집합 A, B가 있다. A에 속한 사람 각각이 B에 속한 사람 일부를 선택하고, B에 속한 각각도 A에 속한 사람 일부를 선택했다. 서로 선택한 사람끼리 1 대 1로 짝지어야 한다면, 모두 만족할 수 있는 짝짓는 방법은?

- Baseball Elimination: 팀간 리그전 하는 스포츠에서 승/패/남은 경기수 주어졌을 때 100% 1위 될 수 없는 팀을 찾고, 이유를 설명 하시오.

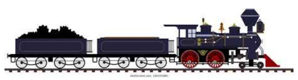
- Medical Image Segmentation: 이미지에서 특정 속성 만족하는 부분 찾기



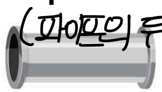
- …

Maxflow 문제에서 꼭 지켜야 하는 물리 법칙

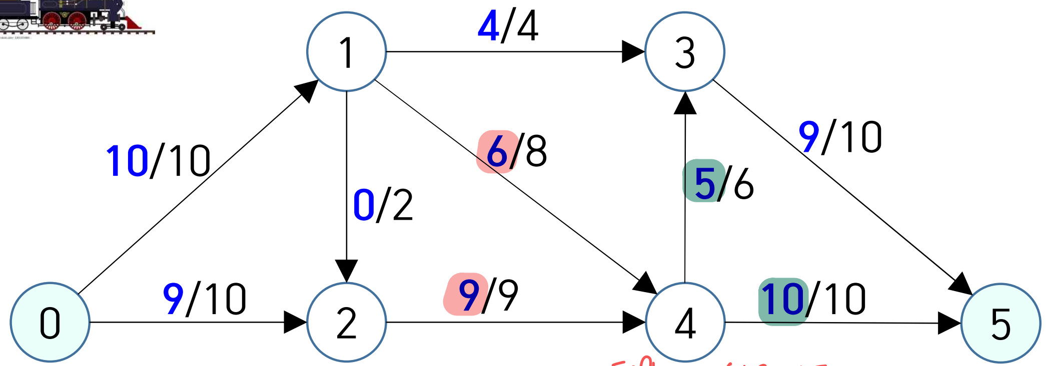
- flow는 간선 방향 따라 흐름



- 0 ≤ flow ≤ capacity
  (파이프의 두께를 넘어서 흐를 수는 X)



- Local equilibrium 조건:
- 출발지 s와 도착지 t 제외한 모든 정점에서
- inflow 합 = outflow 합 (총량이 유지되어야 함)



Inflow = 6+9 = 15
outflow = 5+10 = 15

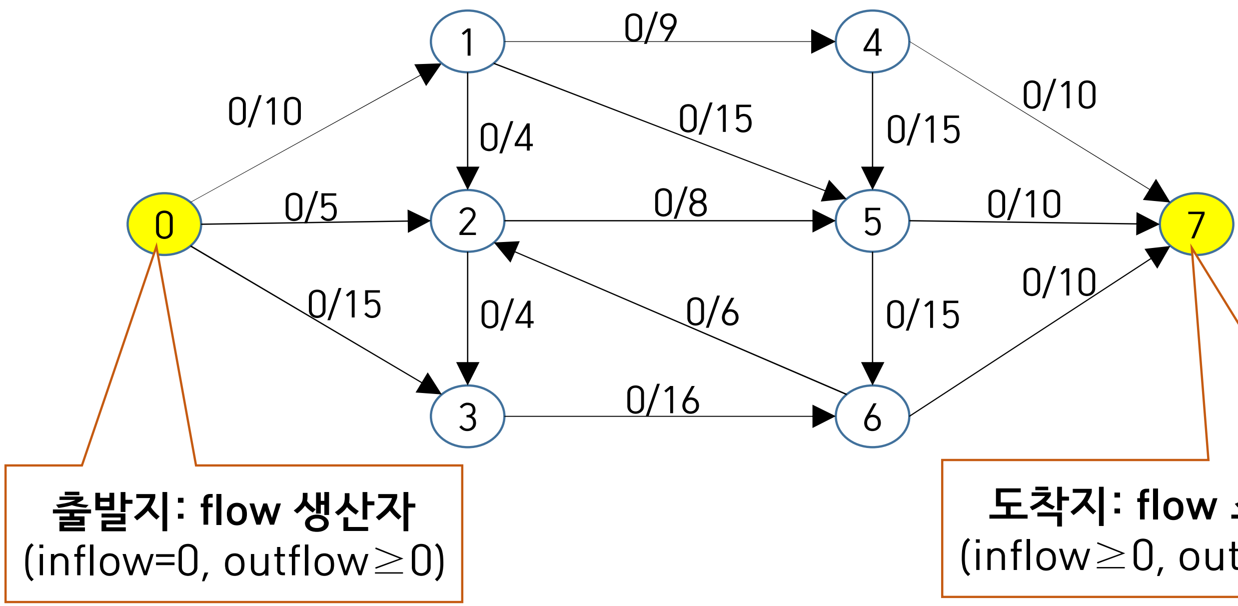
[Q] inflow < outflow인 유일한 정점은? 출발지(생산자)
출발지는 나가는 flow만 있음

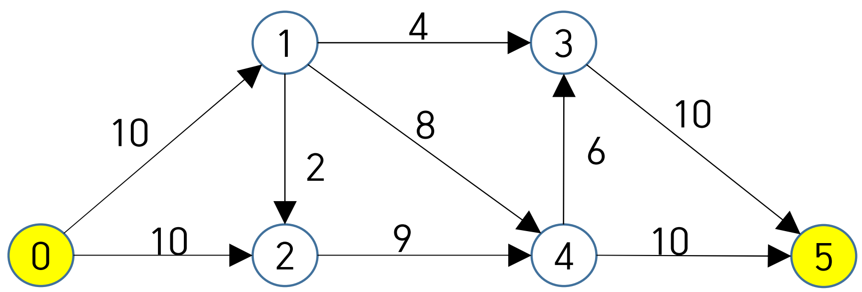
[Q] inflow > outflow인 유일한 정점은? 도착지(소비자)
도착지는 들어오는 flow만 있음

중간(전달자)

출발지에서는 나가는 간선만, 도착지에서는 들어오는 간선만 고려

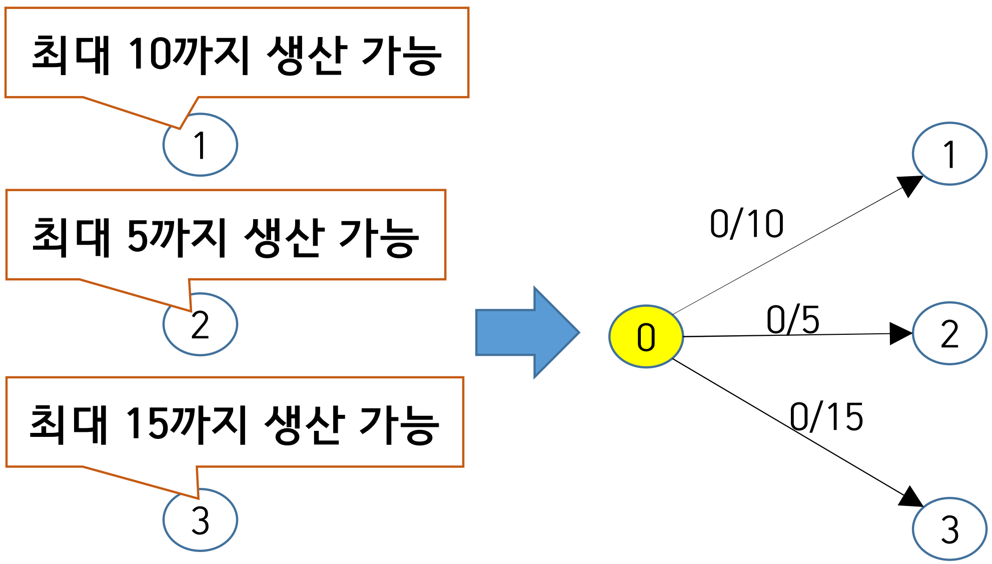
- 출발지에서 목적지 방향으로 보낼 수 있는 flow를 구하는 것이 목표이므로
- 출발지로 들어오는 간선, 목적지에서 나가는 간선 있더라도 제외하고 문제 풀이
- 따라서 앞으로 볼 flow network은 모두 출발지에서는 나가는 간선만, 도착지에는 들어오는 간선만 있는 경우 봄



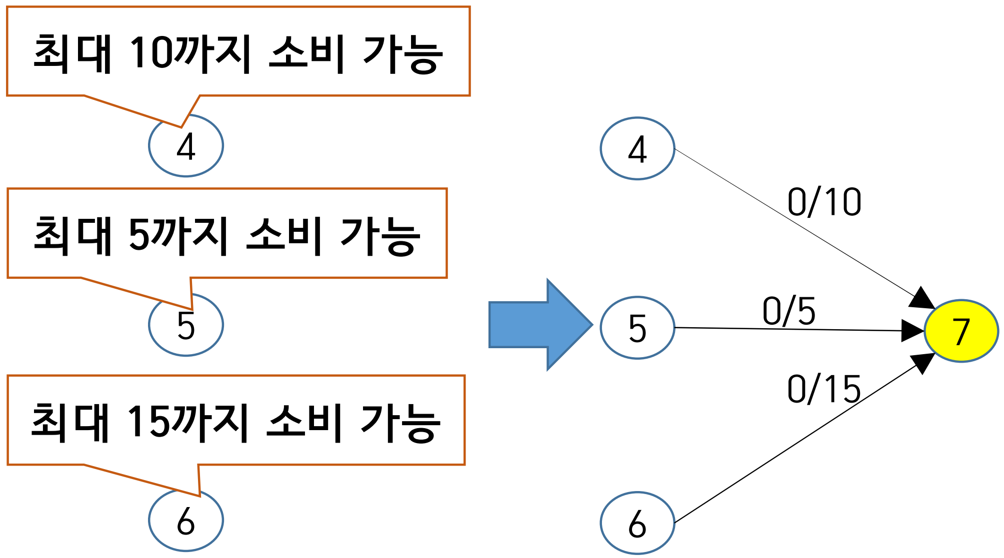


Multi-source, multi-sink도 single-source, single-sink로 대응 가능

flow 생산자 여럿인 경우



flow 소비자 여럿인 경우



Max Flow and Min Cut

Max Flow와 Min Cut 문제의 관계 이해하고 Baseball Elimination 문제 해결에 적용해 보기

01. Max Flow 문제 정의 및 예습자료 주요내용 복습

02. Ford-Fulkerson 알고리즘

Augmenting path 찾아 max flow 문제 해를 찾는 과정

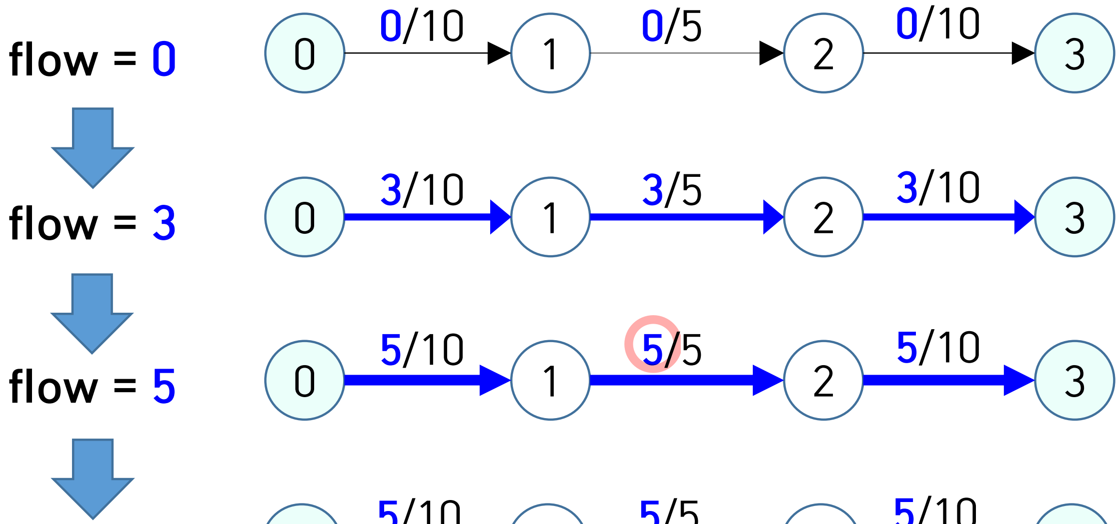
03. Min Cut 문제 정의 및 Max Flow 문제와의 연관성

04. Maxflow-mincut 활용 예: Baseball Elimination 문제

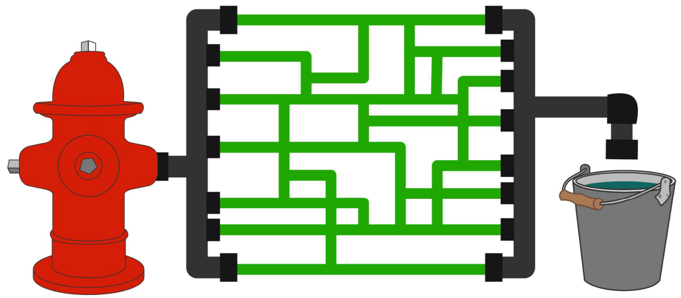
05. 실습: Baseball Elimination 구현

- 복잡한 그래프에 대해 일반적으로 max flow 쉽게 구하는 방법은
- 아직 존재하지 않으며 (예: 간단한 수학식)
- 더는 더할 수 없을 때까지 **flow를 조금씩 더해가는 방법** 사용
- 이를 위해 출발지 s에서 도착지 t까지 flow를 더할 수 있는 경로 찾아 flow 더해감
- 이러한 경로를 **augmenting path**라 함

조금씩 흘려보내서, 여유가 있는 경로를 찾아야 함.



더는 flow 증가시킬 수 없음
따라서 **max flow = 5**

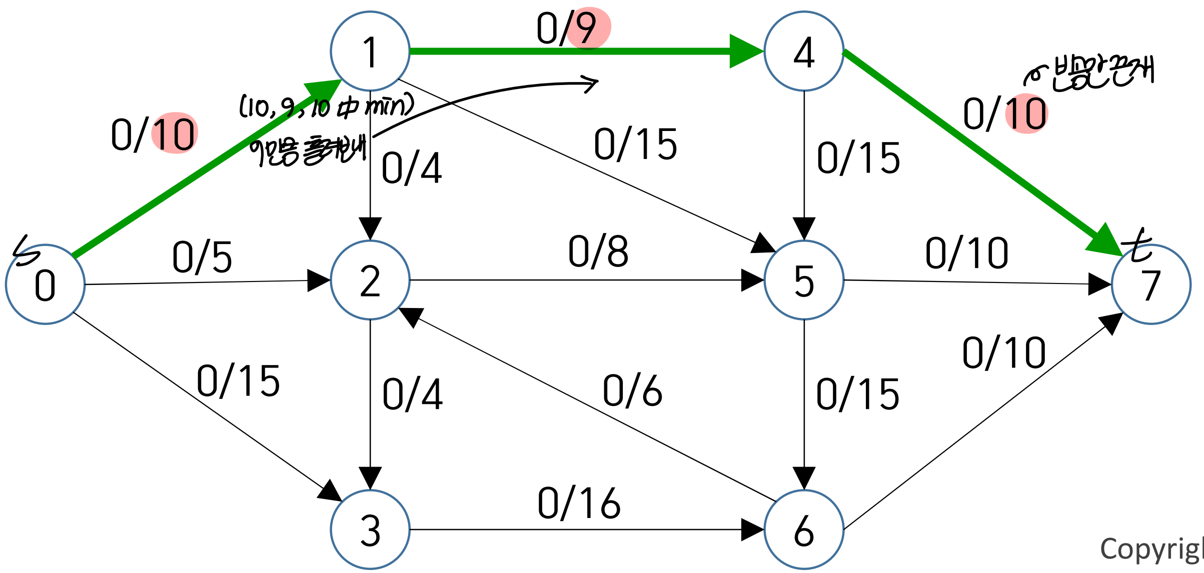


- 복잡한 그래프에 대해 일반적으로 max flow 쉽게 구하는 방법은
- 아직 존재하지 않으며 (예: 간단한 수학식)
- 더는 더할 수 없을 때까지 **flow를 조금씩 더해가는 방법** 사용

빈 공간이 있는 간선

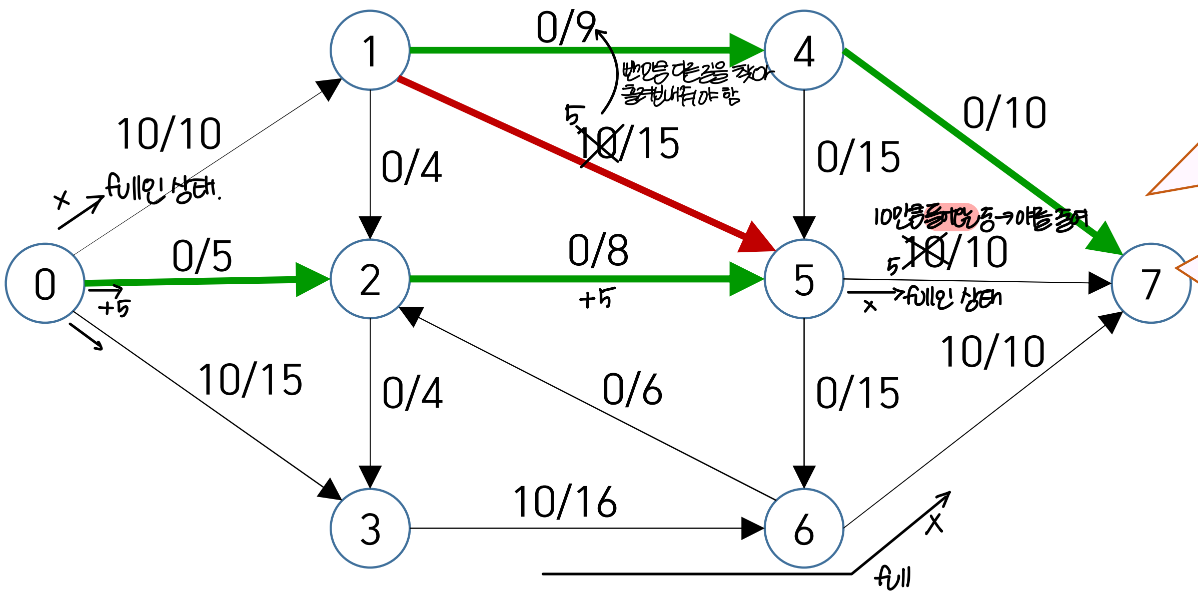
- 다음 조건 만족하는 경로 (**augmenting path**) 찾아 flow 더해감
  - 출발지 s에서 도착지 t까지 연속된 간선의 집합
  - **Forward 간선 (s→t 방향): flow 더할 수 있으면 선정 (현재 배정된 flow < capacity)**
  - **Backward 간선(s←t 방향): flow 뺄 수 있으면 선정 (현재 배정된 flow > 0)**

이미 flow가 배정되어있는 간선



- 다음 조건 만족하는 경로 (**augmenting path**) 찾아 flow 더해감
  - 출발지 s에서 도착지 t까지 연속된 간선의 집합
  - **Forward 간선 (s→t 방향): flow 더할 수 있으면 선정 (현재 배정된 flow < capacity)**
  - **Backward 간선(s←t 방향): flow 뺄 수 있으면 선정 (현재 배정된 flow > 0)**
  - **이번에 추가할 flow 막힘 없이 흐를 수 있도록 기존 배정한 flow redirect하는 역할**

backward 간선 이후, forward 따라감.

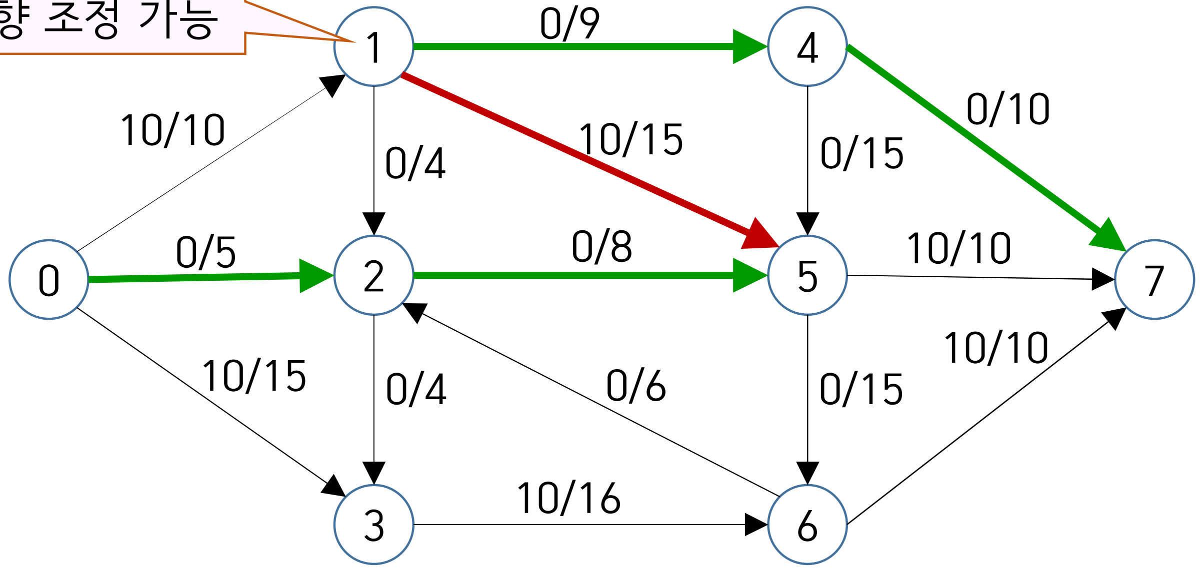


왜 forward 간선 아닌 backward 간선 선택되었나? 5에서 7까지 갈 수 있는 forward 경로 없음 (5→7, 6→7 모두 가득 참)

하지만 backward 간선 1→5 의 flow를 1→4쪽으로 방향 조정하면 7까지 갈 수 있음

Augmenting path에서
backward 간선들 후에 forward 간선 선택해야 redirection 가능

Backward 간선에서 빠지는 flow 만큼
이어지는 forward 간선으로 더해야 방향 조정 가능

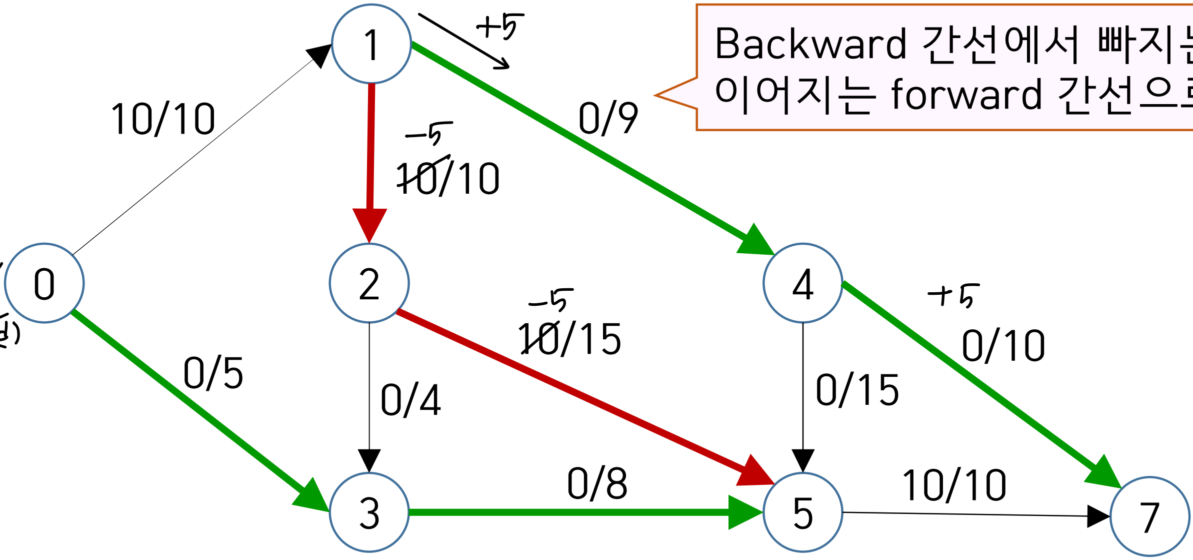


Augmenting path에서 backward 간선들 후에 forward 간선 선택해야 redirection 가능

⊖ ⊕

Backward 간선에서 빠지는 flow 만큼
이어지는 forward 간선으로 더해야 방향 조정 가능

맨처음엔 forward간선 (아무것도 양보하지않아서 감소할 여지X → ~~backward~~)

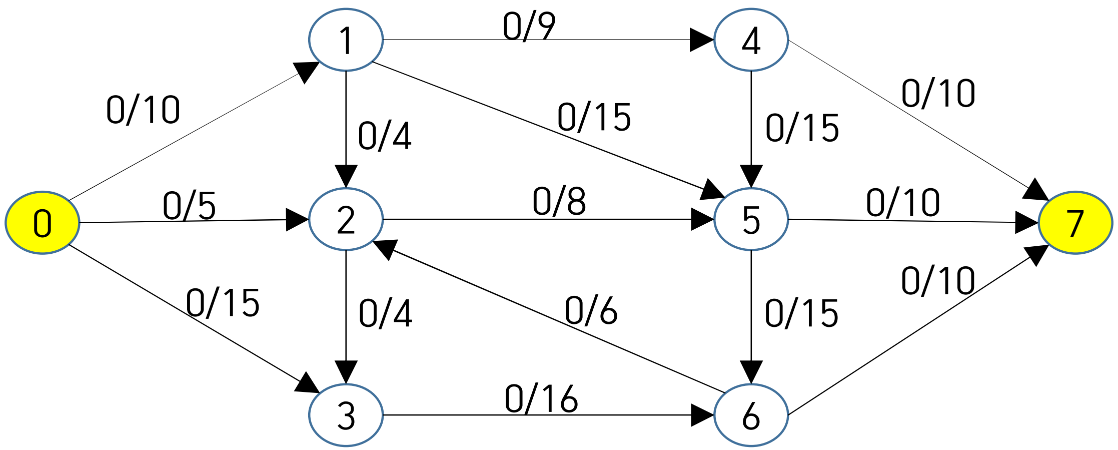


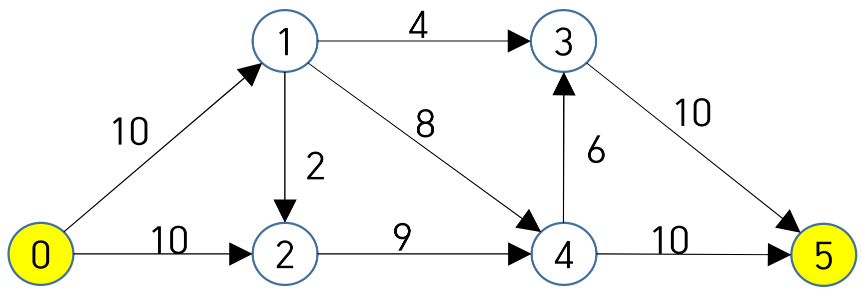
[Q] Augmenting path의 마지막 간선이 backward 간선으로 선택된다면? 문제 되지 않나? 이런경우는 불가능함. Path의 끝은 반드시 forward 간선

(∵ 도착지는 only inflow)

출발지에서는 나가는 간선만, 도착지에서는 들어오는 간선만 고려 따라서 augmenting path에서 backward 간선들 후에는 반드시 forward 간선 선택되며 도착지에 연결됨

- 출발지에서 목적지 방향으로 보낼 수 있는 flow를 구하는 것이 목표이므로
- 출발지로 들어오는 간선, 목적지에서 나가는 간선 있더라도 제외하고 문제 풀이
- 따라서 앞으로 볼 flow network은 모두 출발지에서는 나가는 간선만, 도착지에는 들어오는 간선만 있는 경우 봄





[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]

모든 간선의 flow=0으로 초기화

while augmenting path P 탐색해 존재한다면:

P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)

min(forward 간선에 더할 수 있는 flow 최대량, backward 간선에서 뺄 수 있는 flow 최대량)

P의 각 간선에 minflow 만큼 더 흘려보내기

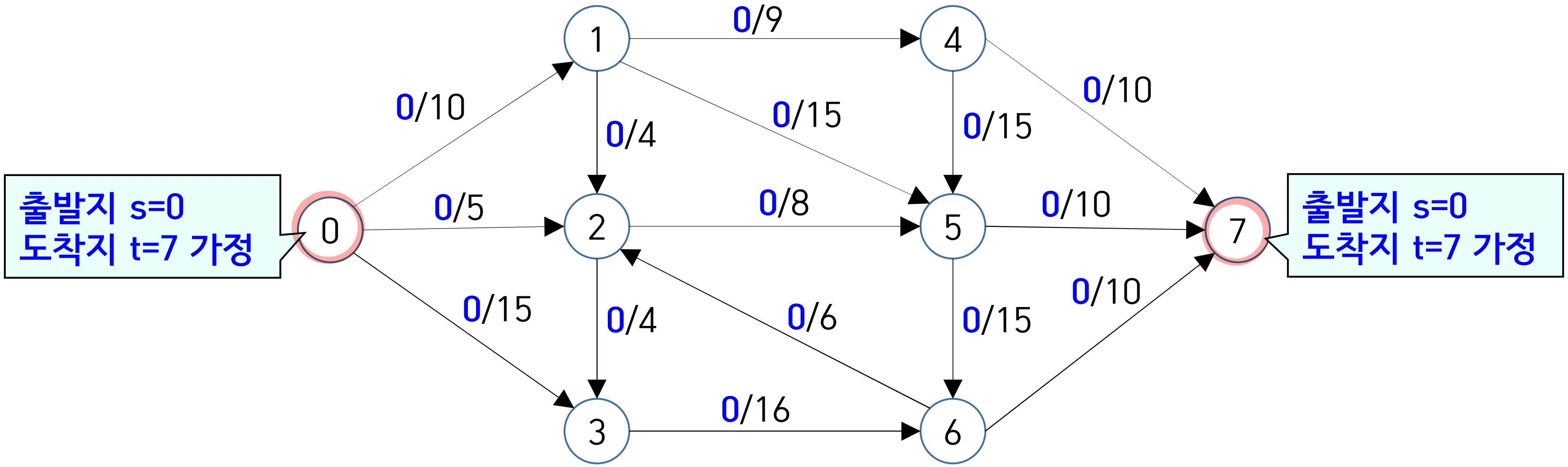
forward 간선에는 minflow만큼 flow 더하고

backward 간선에는 minflow만큼 flow 감소

augmenting path P 찾기:
s에서 t로 이어지는
flow 더할 수 있는 forward 간선과
flow 뺄 수 있는 backward 간선 집합

모든 단계에서 공통적으로
forward 간선에는 flow 더하고
backward 간선에는 뺌

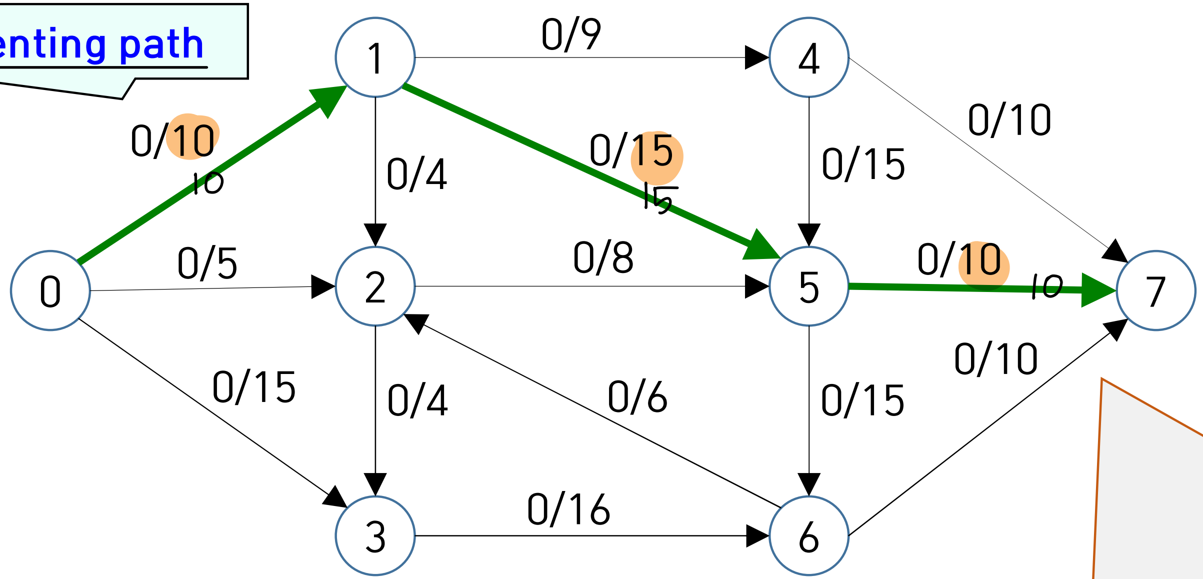
[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
    P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
    P의 각 간선에 minflow 만큼 더 흘려보내기



[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
  P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
  P의 각 간선에 minflow 만큼 더 흘려보내기

첫 번째로 찾은 augmenting path

└항상 forward간선
(∵ 처음에 감소할 필요X)

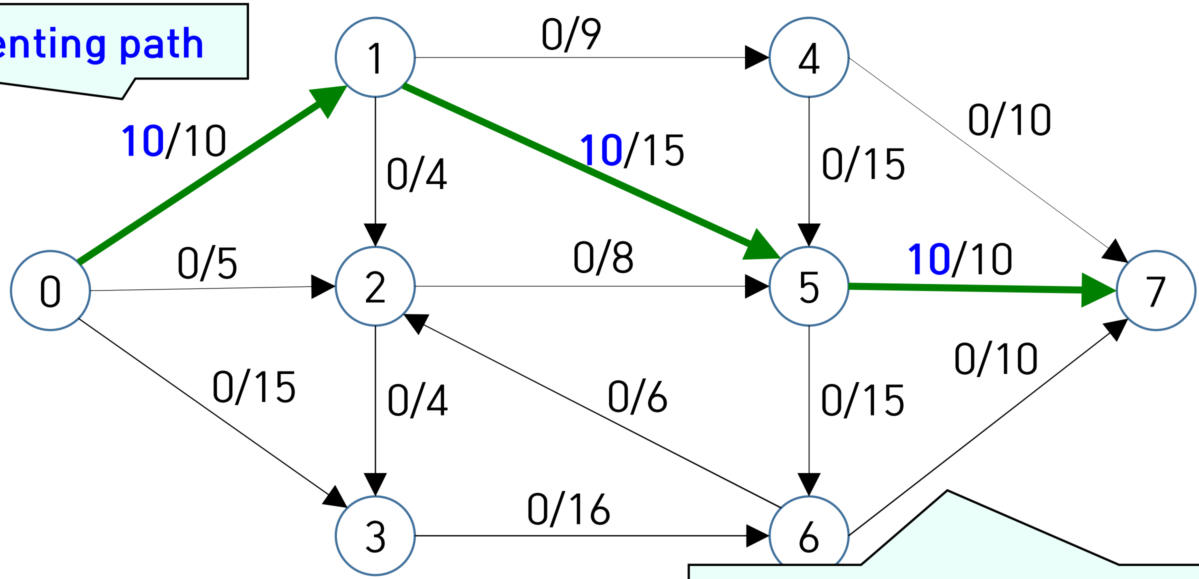


[Q] 이 augmenting path의 minflow는? 즉 추가로 흘려 보낼 수 있는 최대량은?

$min(10, 15, 10) = 10$

[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
    P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
    P의 각 간선에 minflow 만큼 더 흘려보내기

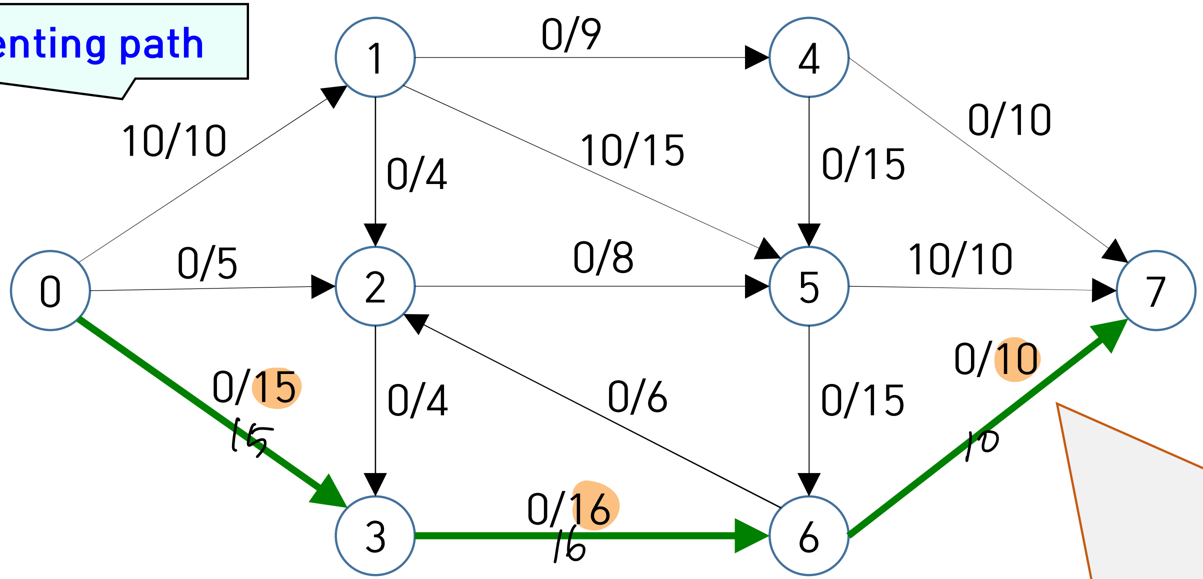
첫 번째로 찾은 augmenting path



추가로 흘려 보낼 수 있는 최대량만큼 흘려 보냄으로써
최대한 빨리 max flow를 찾고자 함

[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
    P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
    P의 각 간선에 minflow 만큼 더 흘려보내기

두 번째로 찾은 augmenting path

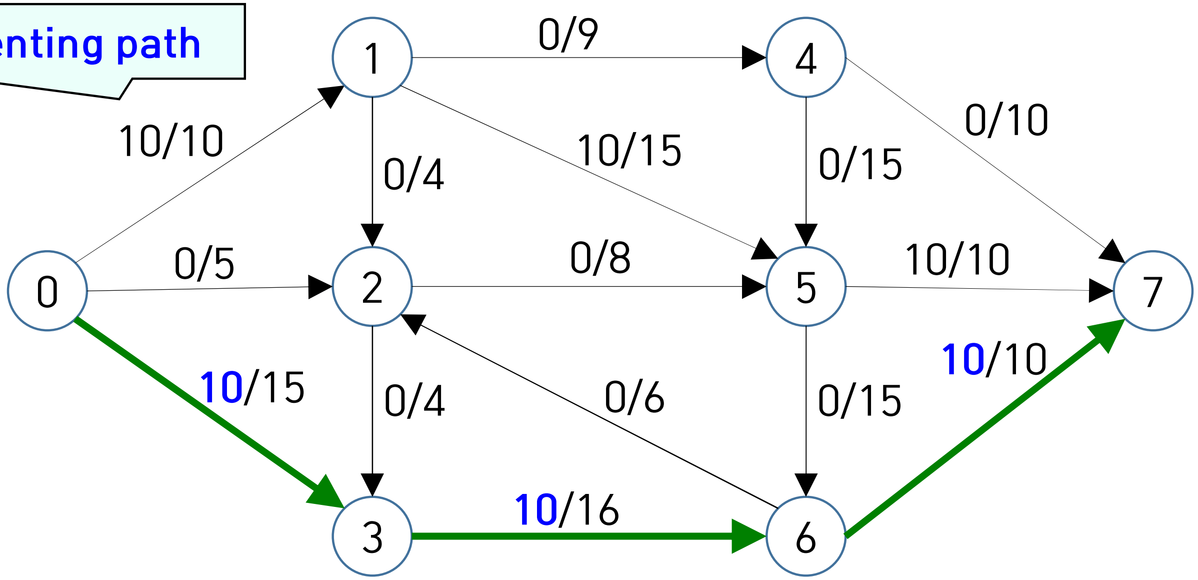


[Q] 이 augmenting path의 minflow는? 즉 추가로 흘려 보낼 수 있는 최대량은?

min(15, 16, 10) = 10

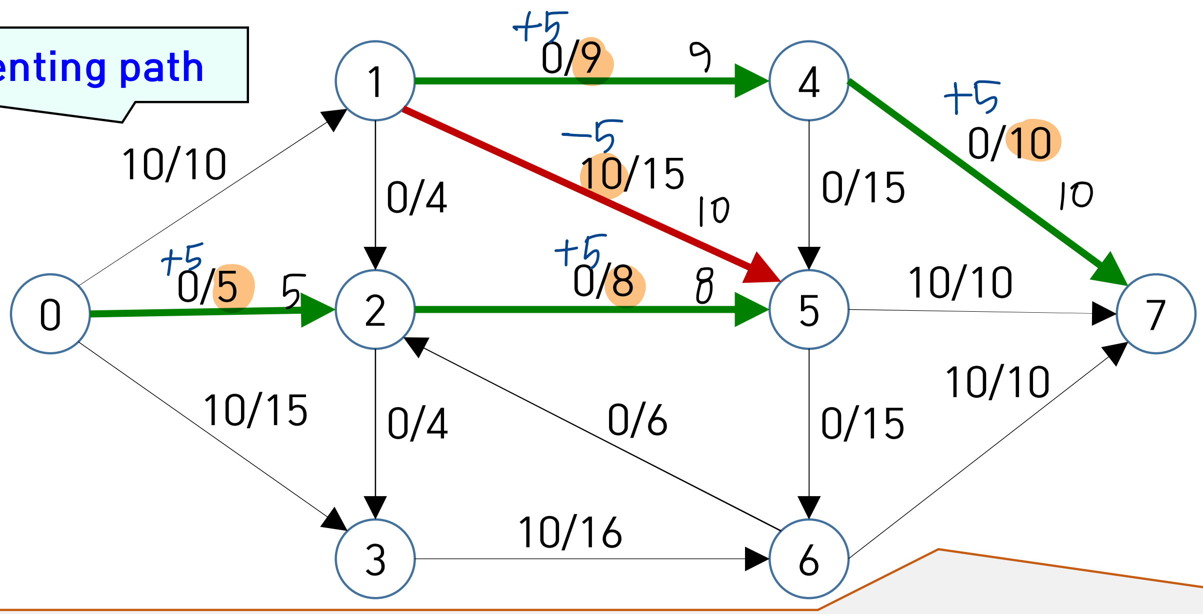
[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
    P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
    P의 각 간선에 minflow 만큼 더 흘려보내기

두 번째로 찾은 augmenting path



[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
    P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
    P의 각 간선에 minflow 만큼 더 흘려보내기

세 번째로 찾은 augmenting path

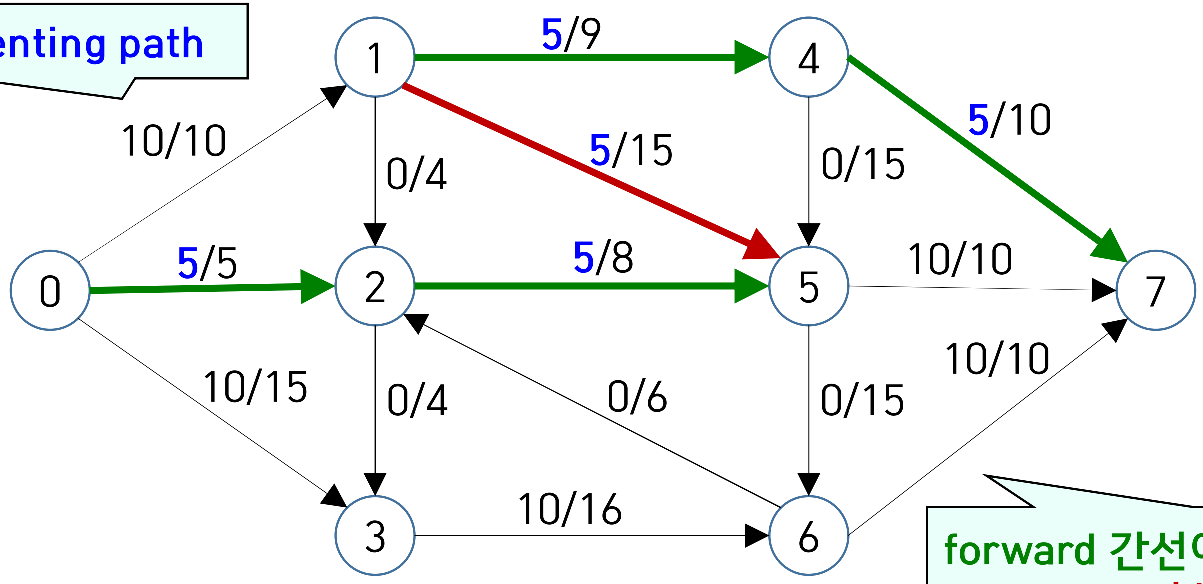


[Q] 이 augmenting path의 minflow는? 즉 추가로 흘려 보낼 수 있는 최대량은?
minflow = min(forward 간선에 더할 수 있는 flow 최대량, backward 간선에서 뺄 수 있는 flow 최대량) 임에 유의

min(5, 8, 10, 9, 10) = 5

[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
    P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
    P의 각 간선에 minflow 만큼 더 흘려보내기

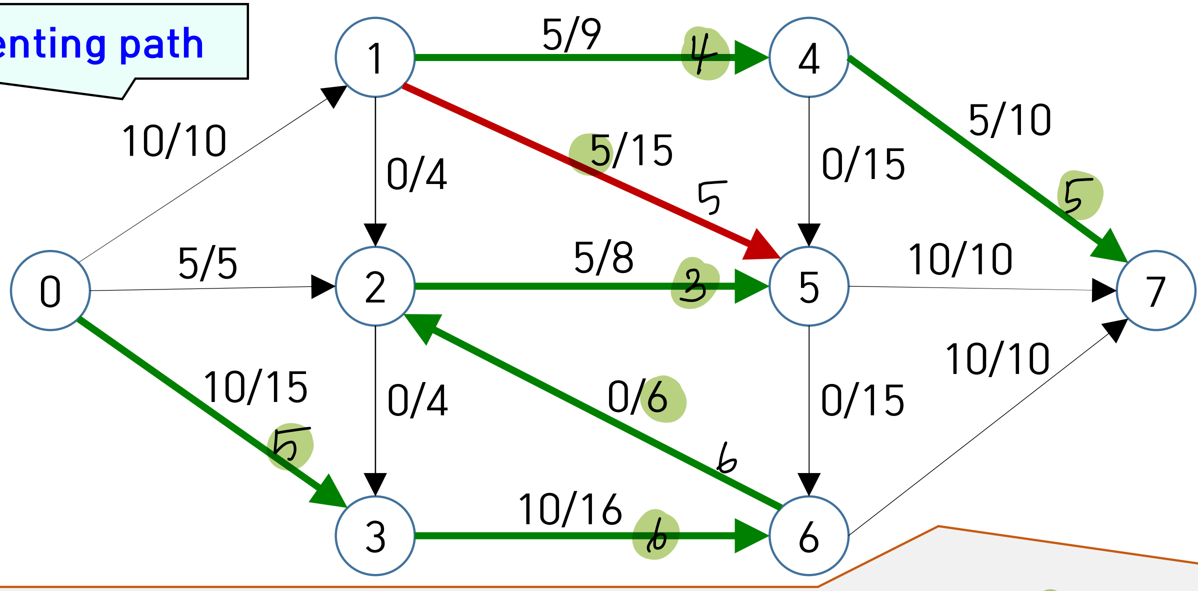
세 번째로 찾은 augmenting path



forward 간선에는 flow 더하고
backward 간선에는 뺐음에 유의

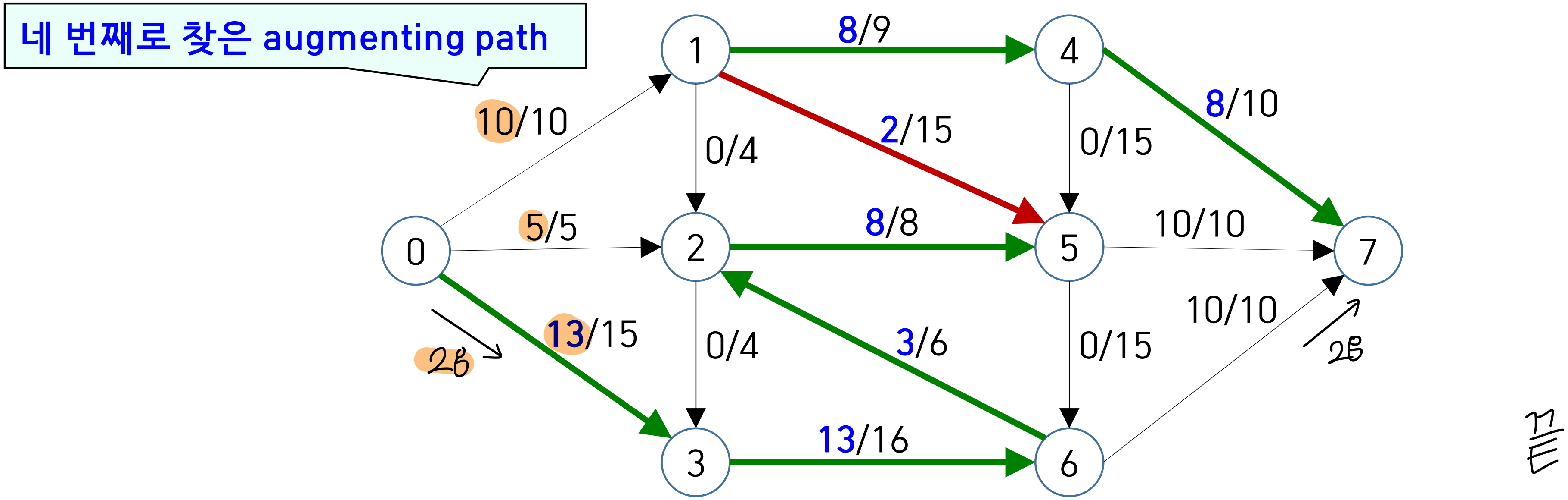
[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
    P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
    P의 각 간선에 minflow 만큼 더 흘려보내기

네 번째로 찾은 augmenting path



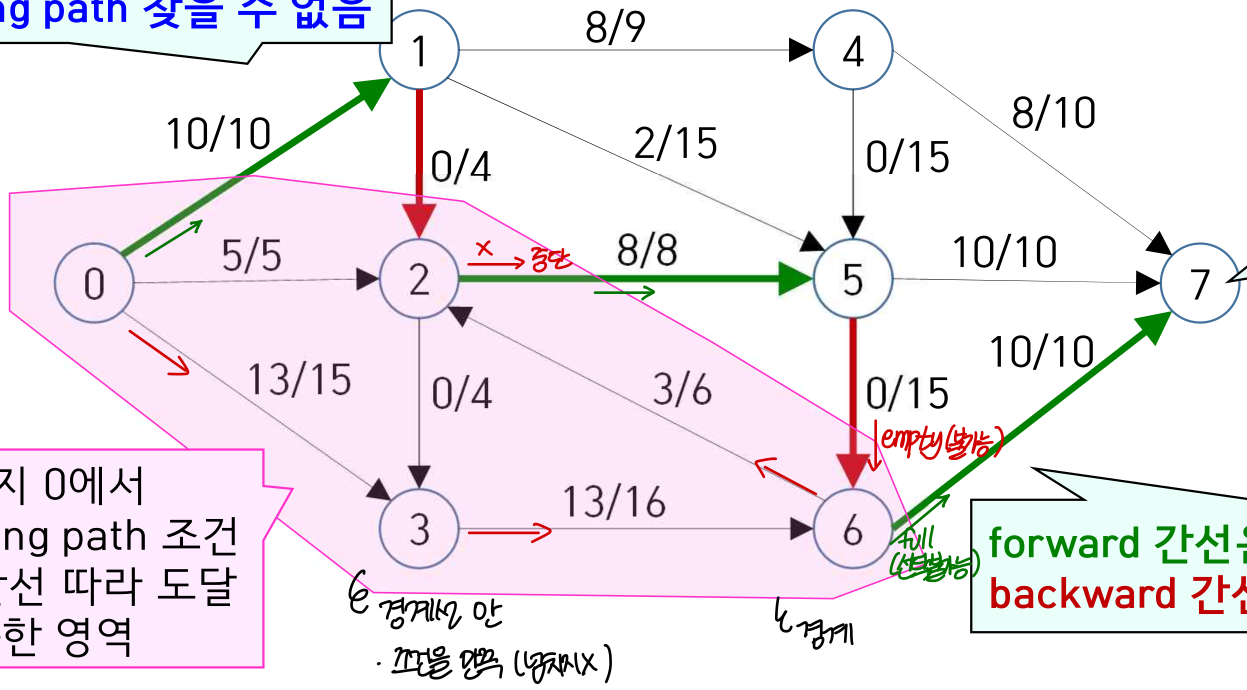
[Q] 이 augmenting path의 minflow는? 즉 추가로 흘려 보낼 수 있는 최대량은? 3
minflow = min(forward 간선에 더할 수 있는 flow 최대량, backward 간선에서 뺄 수 있는 flow 최대량) 임에 유의

[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
    P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
    P의 각 간선에 minflow 만큼 더 흘려보내기



[s-t max flow 구하기 위한 Ford-Fulkerson 알고리즘]
모든 간선의 flow=0으로 초기화
while augmenting path P 탐색해 존재한다면:
  P의 minflow 찾기 (P 통해 추가로 흘려 보낼 수 있는 최대량)
  P의 각 간선에 minflow 만큼 더 흘려보내기

더는 s-t augmenting path 찾을 수 없음



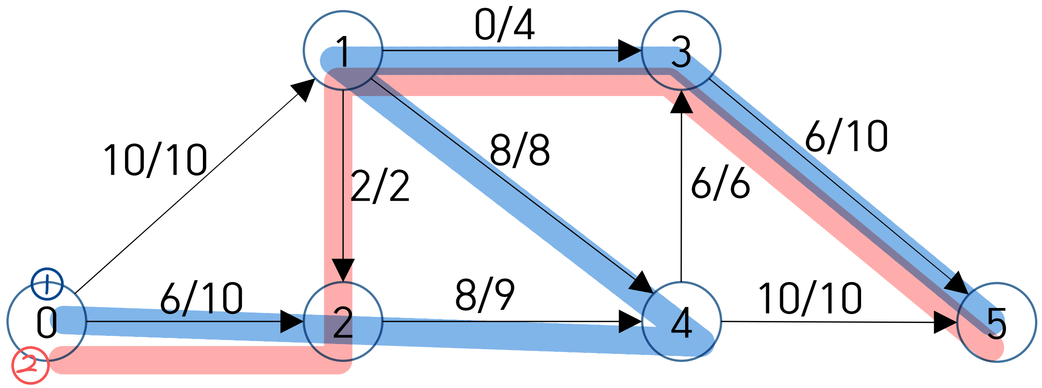
max flow = 8+10+10 = 28

출발지 0에서 augmenting path 조건에 맞는 간선 따라 도달 가능한 영역

forward 간선은 flow 더할 수 없고
backward 간선은 flow 뺄 수 없음

[Q] 아래 flow network에는 2개의 서로 다른 augmenting path가 존재한다. 이들을 모두 찾아 보시오.

s에서 시작해서
forward 간선은 flow 더할 수 있는 곳으로
backward 간선은 flow 뺄 수 있는 곳으로 따라가서
t에 도달하는 간선의 집합



Ford-Fulkerson 알고리즘의 worst-case 성능

- s-t augmenting path 찾는데
- DFS or BFS 사용한다면
- ~V+E 시간 소요
- 많은 경우 V<<E 라서 ~E

- 모든 간선의 capacity가 >=인 **정수**이고
- **max flow = U라면**
- 한 augmenting path마다 최소 +1씩 flow 추가되므로
- **최대 U회 iteration 후**에는 max flow 도달

	비용	Iteration 횟수	총 비용
Augmenting path P 찾기	$V+E$	U (최악의 경우)	$(V+E)U$ if) $V \ll E \rightarrow \sim EU$
P의 minflow 찾기	V		
P에 속한 간선에 minflow만큼 더하거나 빼기	V		

Augmenting path 최대한 잘 선정해 iteration 횟수 줄이는 것이 관건

Ford-Fulkerson

정말 비슷함.

최단경로

augmenting path(BFS)

E_N

Fattest

augmenting path(Dijkstra)

$E_{N\log N}$

(iteration 횟수↓)

```
# Perform BFS to find vertices reachable from s along with shortest paths to them
def hasAugmentingPath(self):
    self.edgeTo = [None for _ in range(self.g.V)]
    self.visited = [False for _ in range(self.g.V)]

    q = Queue()
    q.put(self.s)
    self.visited[self.s] = True
    while not q.empty():
        v = q.get()
        for e in self.g.adj[v]:
            w = e.other(v)
            if e.remainingCapacityTo(w) > 0 and not self.visited[w]:
                self.edgeTo[w] = e
                self.visited[w] = True
                q.put(w)

    return self.visited[self.t] # Is t reachable from s with current flow assignment?
```

임의의 s-t augmenting path 사용하는 방식보다 (DFS 기반)

최소간선수 augmenting path (BFS 기반)
혹은
Fattest augmenting path (minflow 가장 큰 path, Dijkstra 기반)
사용하는 방식이 일반적으로 훨씬 빠름

s에서 시작해
forward 간선은 flow 더할 수 있는 곳으로
backward 간선은 flow 뺄 수 있는 곳으로 따라가서
t에 도달하는 최소 간선수 경로 구하기

목적지 t에 도달할 수 있는 경로 있다면 True 반환

```
class FordFulkerson:
    def __init__(self, g, s, t):
        self.flow = 0.0        ∈ BFS기반
        while self.hasAugmentingPath():   # augmenting path 존재하는 경우 아래 반복
            # 찾은 augmenting path의 minflow 구하기
            minflow = float('inf')
            v = t
            while v != s:
                minflow = min(minflow, self.edgeTo[v].remainingCapacityTo(v))
                v = self.edgeTo[v].other(v)

            # 구한 minflow를 augmenting path 각 간선에 더함
            v = t
            while v != s:
                self.edgeTo[v].addRemainingFlowTo(v, minflow)
                v = self.edgeTo[v].other(v)

            # Increase the amoung of flow
            self.flow += minflow

    def hasAugmentingPath(self): # BFS 수행해 augmenting path 찾음
        # 앞 페이지 참조
```

최종적으로 maxflow 계산한 결과가 저장되는 멤버 변수

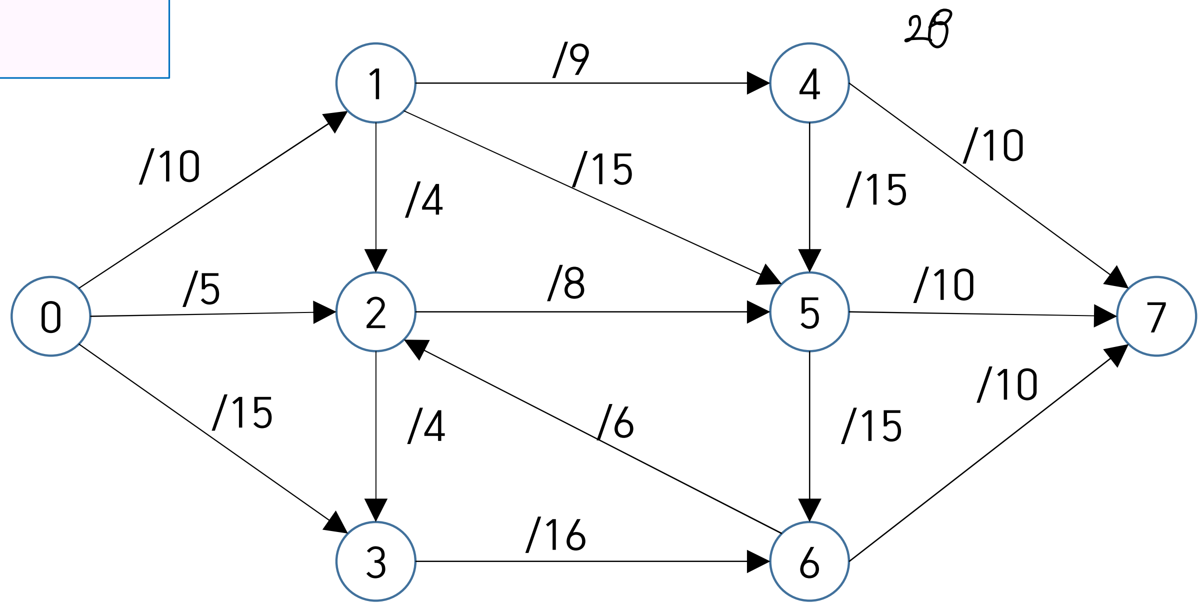
[Q] FlowGraph.py 파일의 __main__ 아래에는 FordFulkerson 클래스에 대한 unit test가 있다.
이 중 아래 코드를 주석 밖으로 꺼내 실행하면 오른쪽 그래프에 대한 max flow를 찾아준다.

(1) 출력 결과를 그래프의 간선에 적어보며 이 그래프의 max flow가 맞음을 확인하시오.

(2) 아래 코드를 잘 읽어 의미를 이해해 보시오. 오늘 실습 과제에서는 FordFulkerson 알고리즘을 활용하는 코드를 작성해야 합니다.

```
g8 = FlowNetwork.fromFile("flownet8.txt")
ff8 = FordFulkerson(g8, 0, g8.V-1)
print("ff8.flow", ff8.flow)
print("ff8.g", ff8.g)
```

0→ 마지막정점

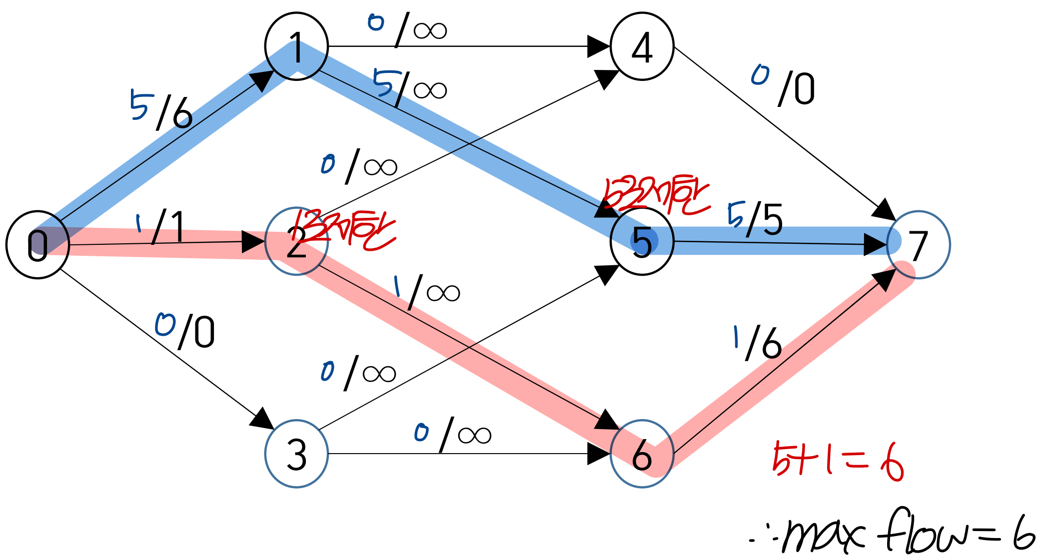


[Q] FlowGraph.py 파일의 __main__ 아래에는 FordFulkerson 클래스에 대한 unit test가 있다.
이 중 아래 코드를 주석 밖으로 꺼내 실행하면 오른쪽 그래프에 대한 max flow를 찾아준다.

(1) 출력 결과를 그래프의 간선에 적어보며 이 그래프의 max flow가 맞음을 확인하시오.

(2) 아래 코드를 잘 읽어 의미를 이해해 보시오. 오늘 실습 과제에서는 이처럼 **flow network을 만들고** 여기에 **FordFulkerson 알고리즘을 수행하는** 코드를 작성해야 합니다.

```
g8m = FlowNetwork(8)
g8m.addEdge(FlowEdge(0,1,6))
g8m.addEdge(FlowEdge(0,2,1))
g8m.addEdge(FlowEdge(0,3,0))
g8m.addEdge(FlowEdge(1,4,float('inf')))
g8m.addEdge(FlowEdge(1,5,float('inf')))
g8m.addEdge(FlowEdge(2,4,float('inf')))
g8m.addEdge(FlowEdge(2,6,float('inf')))
g8m.addEdge(FlowEdge(3,5,float('inf')))
g8m.addEdge(FlowEdge(3,6,float('inf')))
g8m.addEdge(FlowEdge(4,7,0))
g8m.addEdge(FlowEdge(5,7,5))
g8m.addEdge(FlowEdge(6,7,6))
ff8m = FordFulkerson(g8m, 0, g8m.V-1)
print("ff8m.flow", ff8m.flow)
print("ff8m.g", ff8m.g)
```



Max Flow and Min Cut

Max Flow와 Min Cut 문제의 관계 이해하고 Baseball Elimination 문제 해결에 적용해 보기

01. Max Flow 문제 정의 및 예습자료 주요

02. Ford-Fulkerson 알고리즘

03. Min Cut 문제 정의 및 Max Flow 문제와의 연관성

04. Maxflow-mincut 활용 예: Baseball Elimination 문제

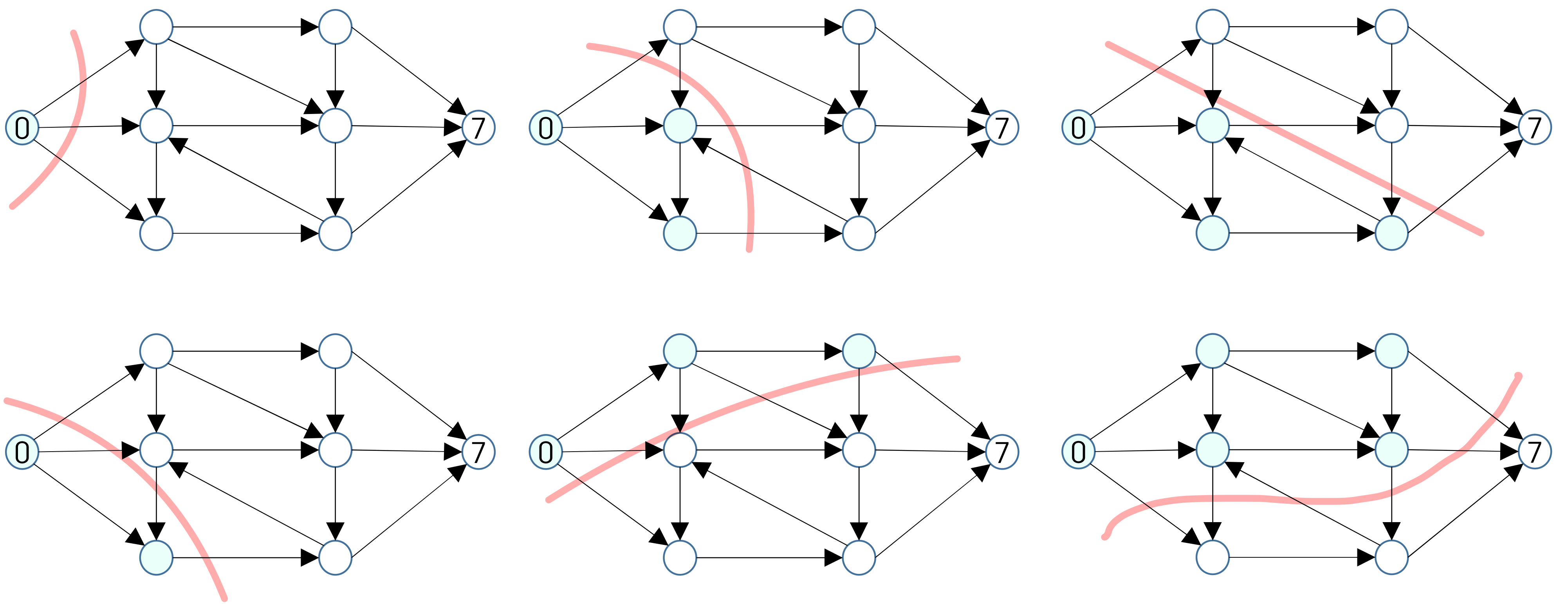
05. 실습: Baseball Elimination 구현

Maxflow와 Mincut은 함께 활용하는 경우 많음
Maxflow의 해가 나온 이유를 Mincut이 설명해 줌

이론 수업 중 실습을 병행하며 진행하므로
첨부 코드를 미리 다운 받아 실행 가능하게 준비해 두세요.

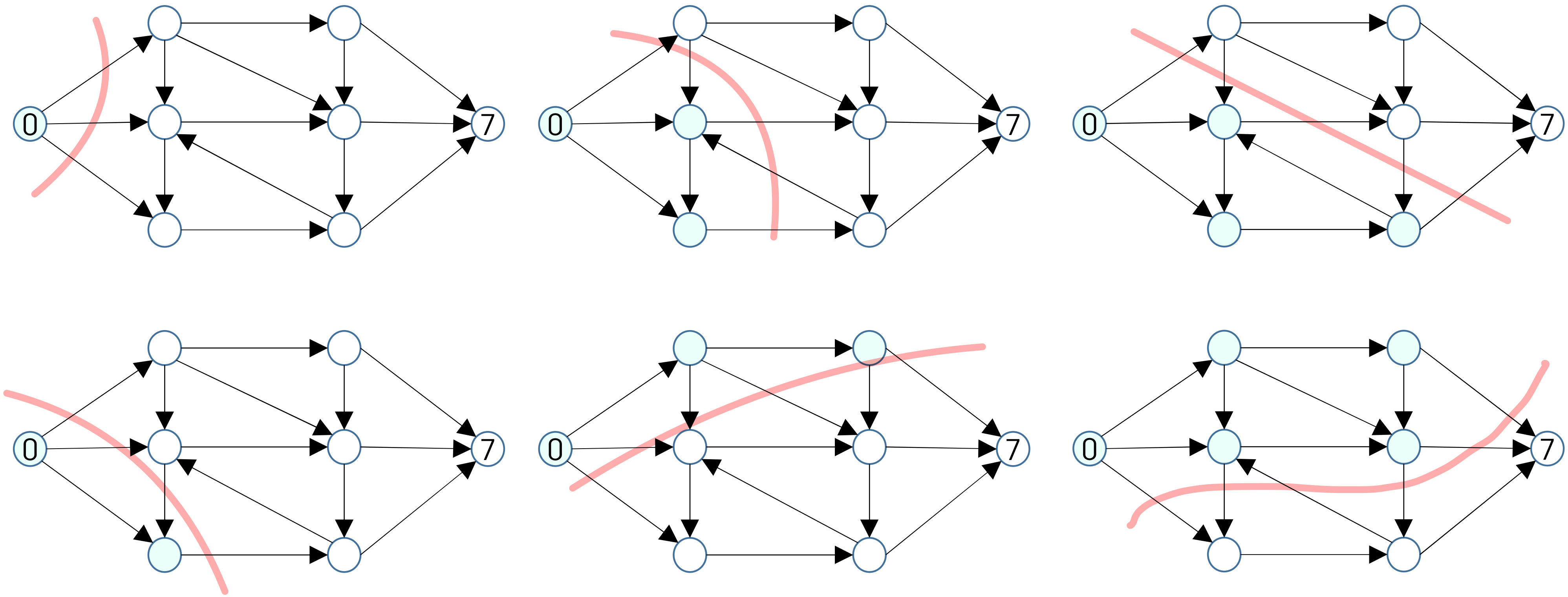
Cut (partition): G의 정점을 2개 partition으로 분할한 것

- G의 cut 예



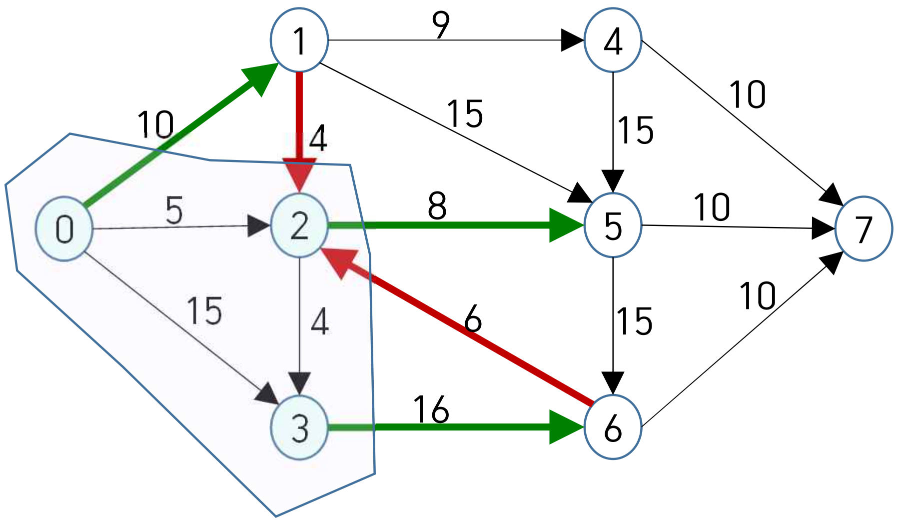
G의 **정점**을 **출발지가 속한 cut**과 (이와 배타적인) **도착지가 속한 cut**으로 분할한 경우만 고려

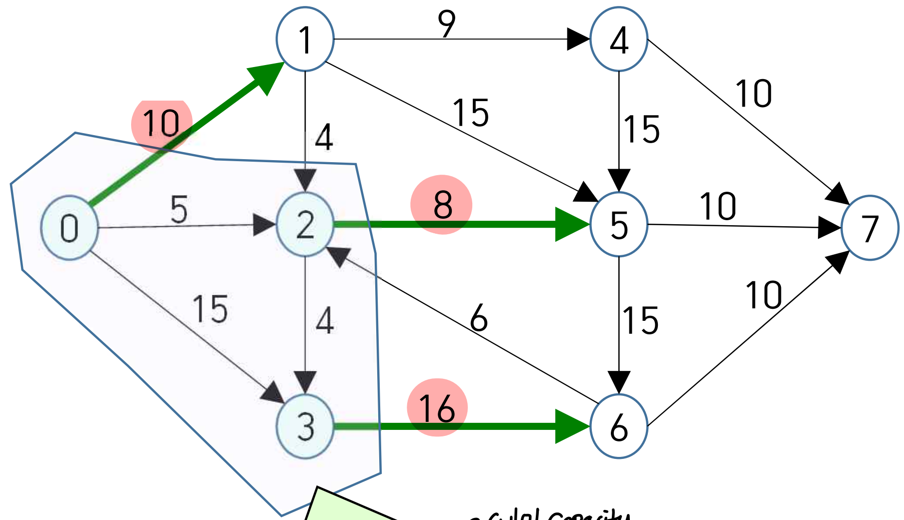
- 위 조건에 따라 분할한 예 (**출발지가 속한 cut은 푸른색**, 도착지가 속한 cut은 흰색으로 볼 수 있음)



Cut의 capacity: crossing edge 중 출발지 → 도착지 방향 간선의 capacity 합 (즉 forward 간선의 capacity 합이라고 보면 됨)

- 출발지가 속한 cut은 푸른색, 도착지가 속한 cut은 흰색





출발지 → 도착지 방향으로 이러한 **단면(cut) 통해 흘러갈 수 있는 최대 flow 량** 보여줌

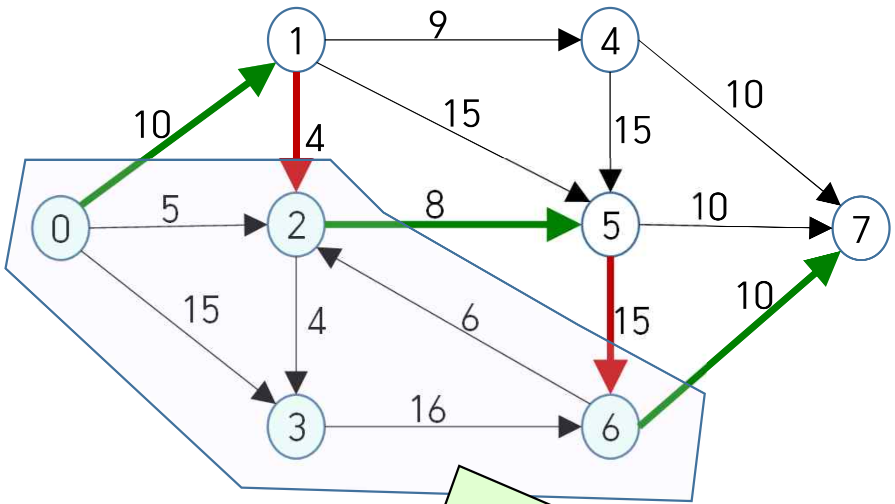
capacity = 10 + 8 + 16 = 34

← cut의 capacity

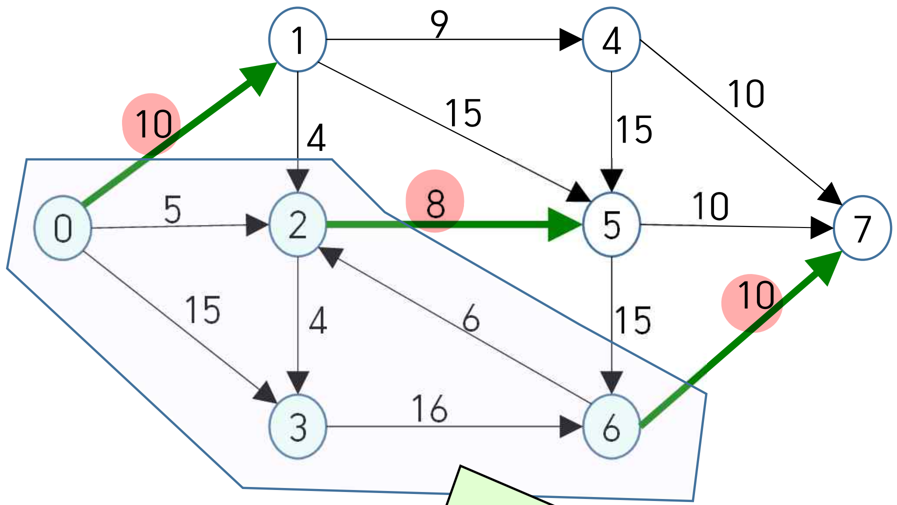
s cut ← t cut 방향 간선의 capacity는 안 따짐에 유의

Cut의 capacity: crossing edge 중 출발지 → 도착지 방향 간선의 capacity 합 (즉 forward 간선의 capacity 합이라고 보면 됨)

- 출발지가 속한 cut은 푸른색, 도착지가 속한 cut은 흰색



capacity = 10 + 8 + 10 = 28



capacity = 10 + 8 + 10 = 28

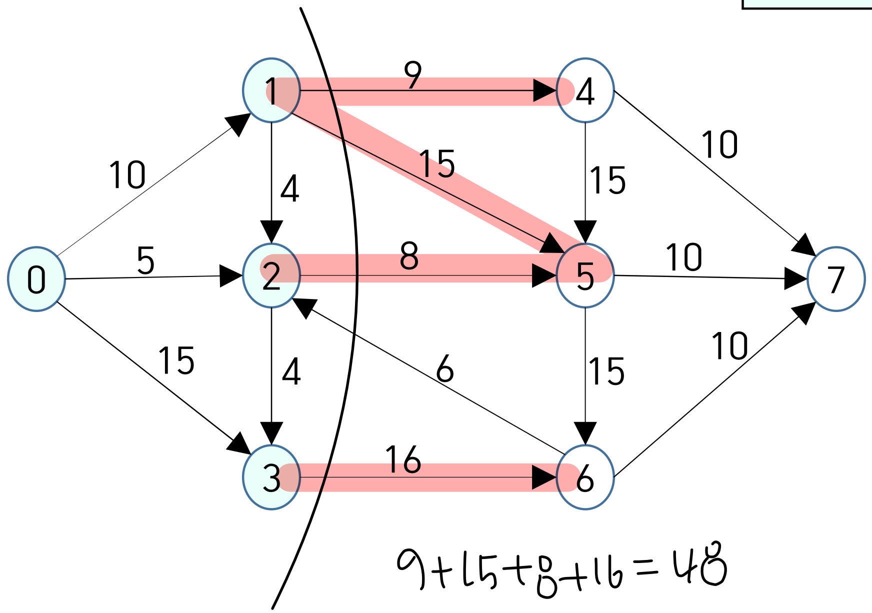
forward의 edge만 따짐
(단면의 두께)

출발지 → 도착지 방향으로
이러한 **단면(cut) 통해 흘러갈 수 있는 최대 flow 량** 보여줌

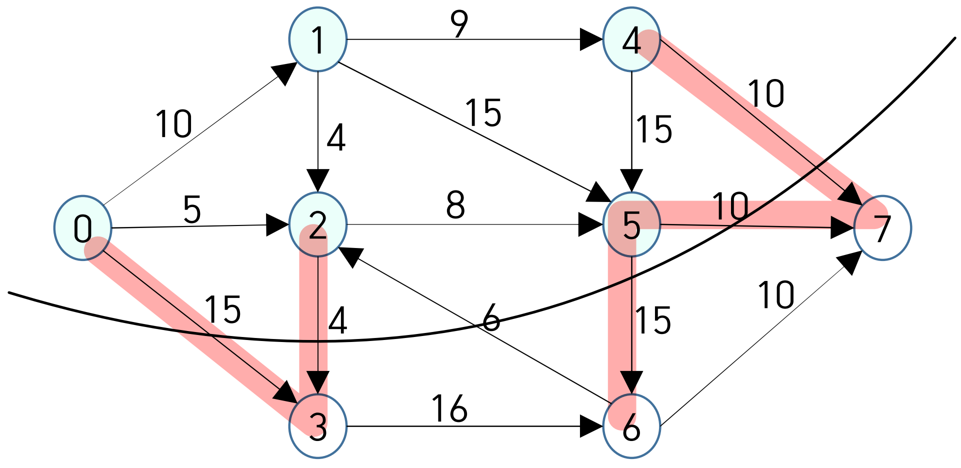
단면의 둘레)

[Q] 다음 $\boxed{\text{cut의 capacity}}$를 구하시오.

두 cut의 crossing edge (경계선 상의 간선) 중
출발지→도착지 방향 간선의 (forward 간선의) capacity합

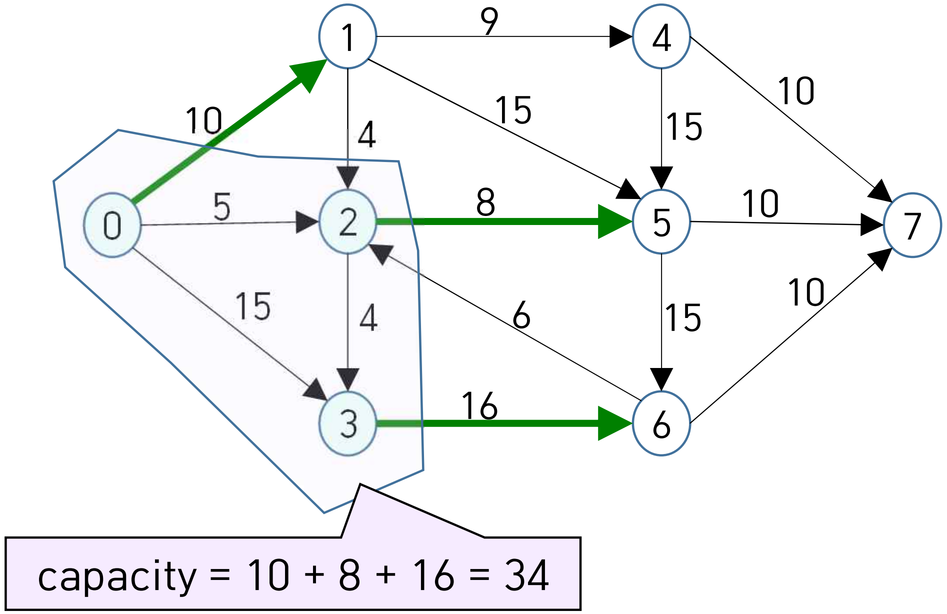


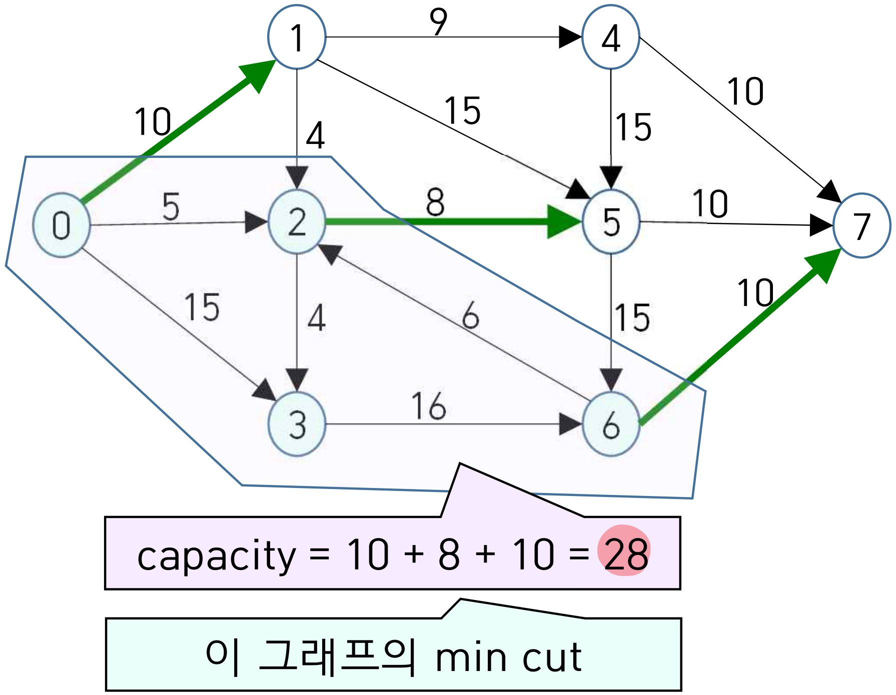
9+15+8+16=48



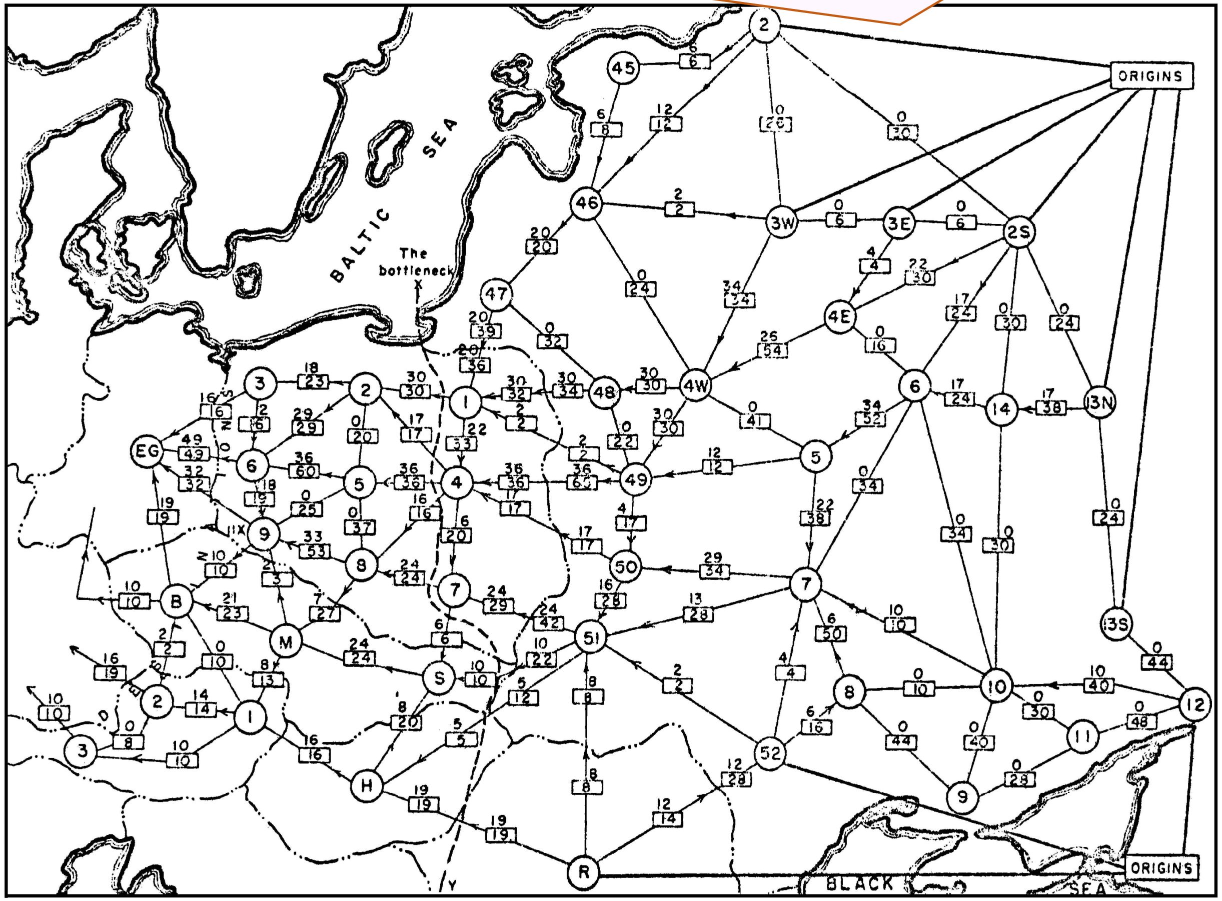
15+4+15+10+10=54

Min cut: capacity 최소인 cut



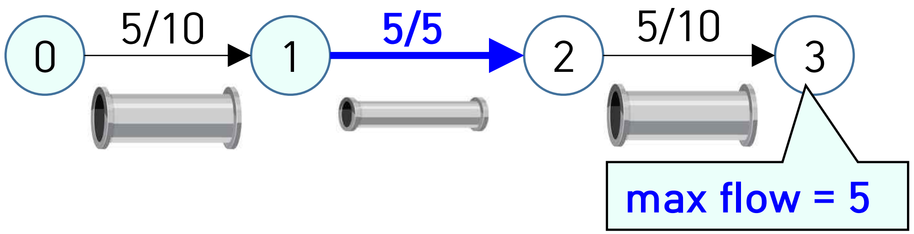


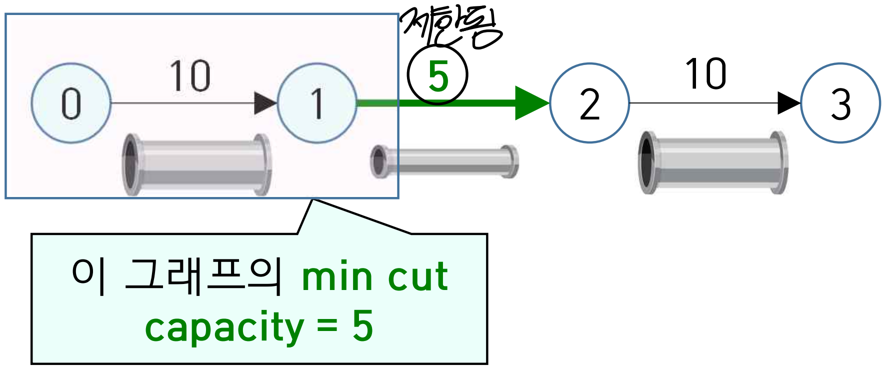
전쟁 중 적군이 후방 → 전방으로 물자 전송할 때
가장 적은 비용으로 물자 전송을 차단하려면 어느 선을 폭파해야 하나?

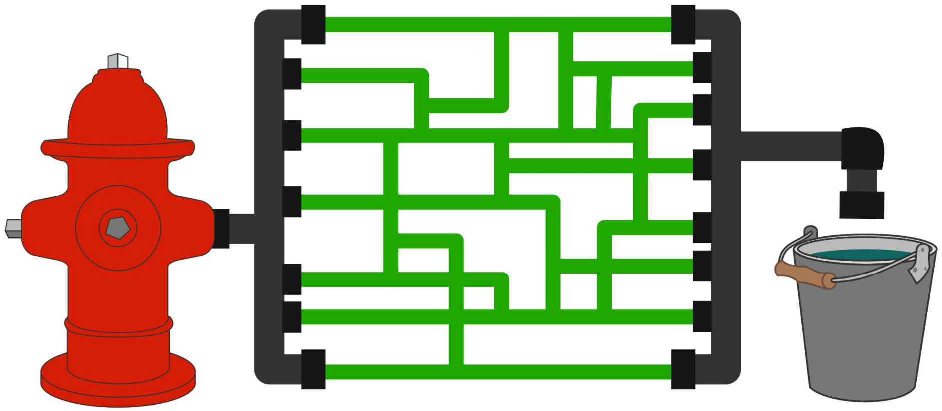


<러시아와 동유럽 국가 간 철도 연결도 및 운송량>

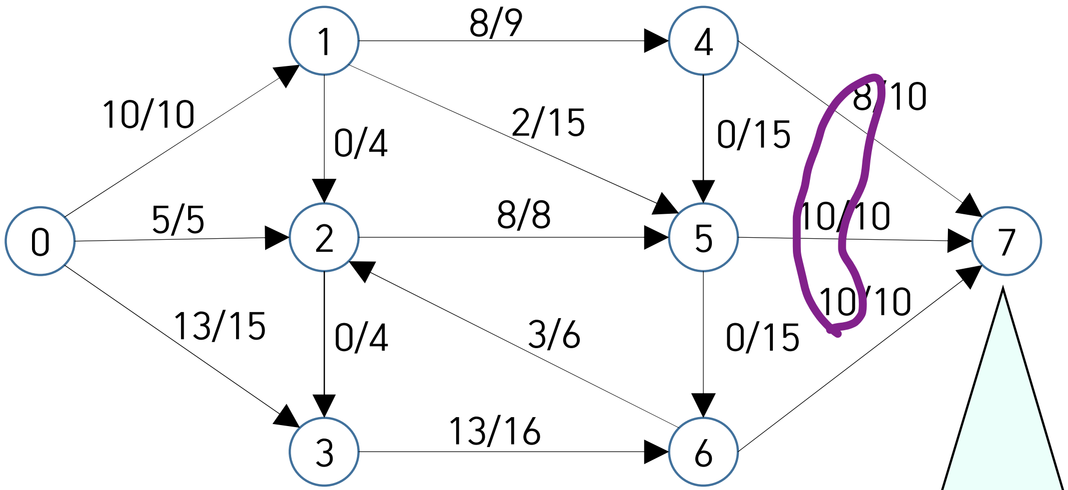
주어진 그래프 G의 max flow = min cut의 capacity



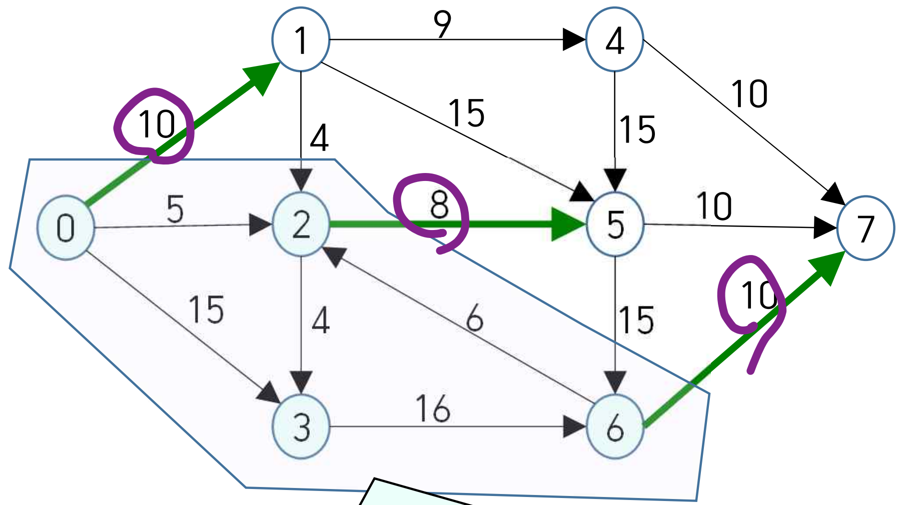




주어진 그래프 G의 max flow = min cut의 capacity

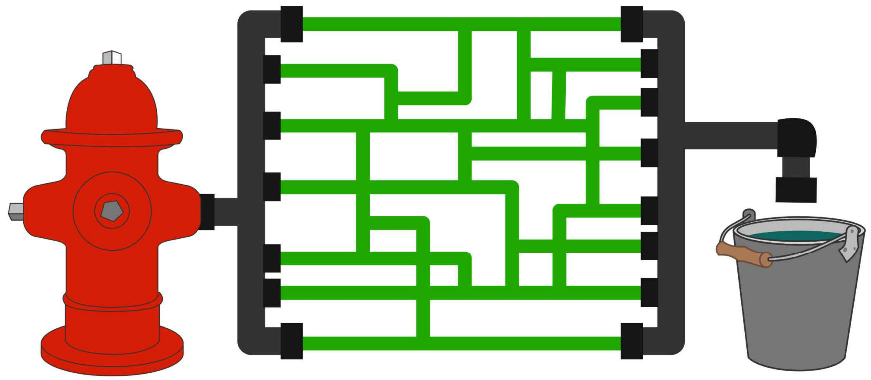


max flow = 8+10+10 = 28

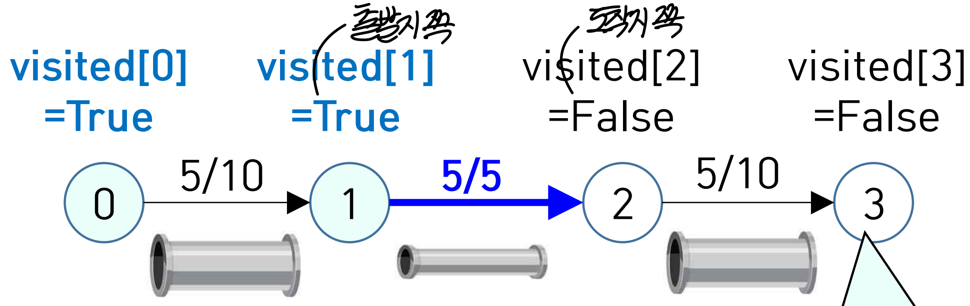


이 그래프의 min cut
capacity = 10 + 8 + 10 = 28

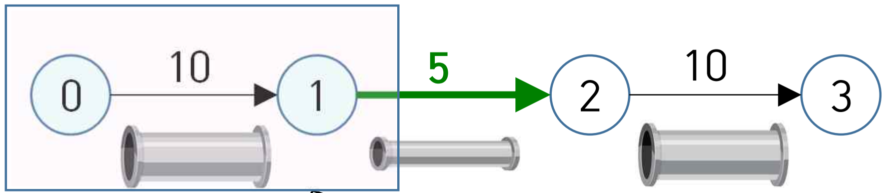
max flow



max flow의 해로부터 min cut의 capacity뿐 아니라 min cut에 속한 정점도 이끌어낼 수 있음



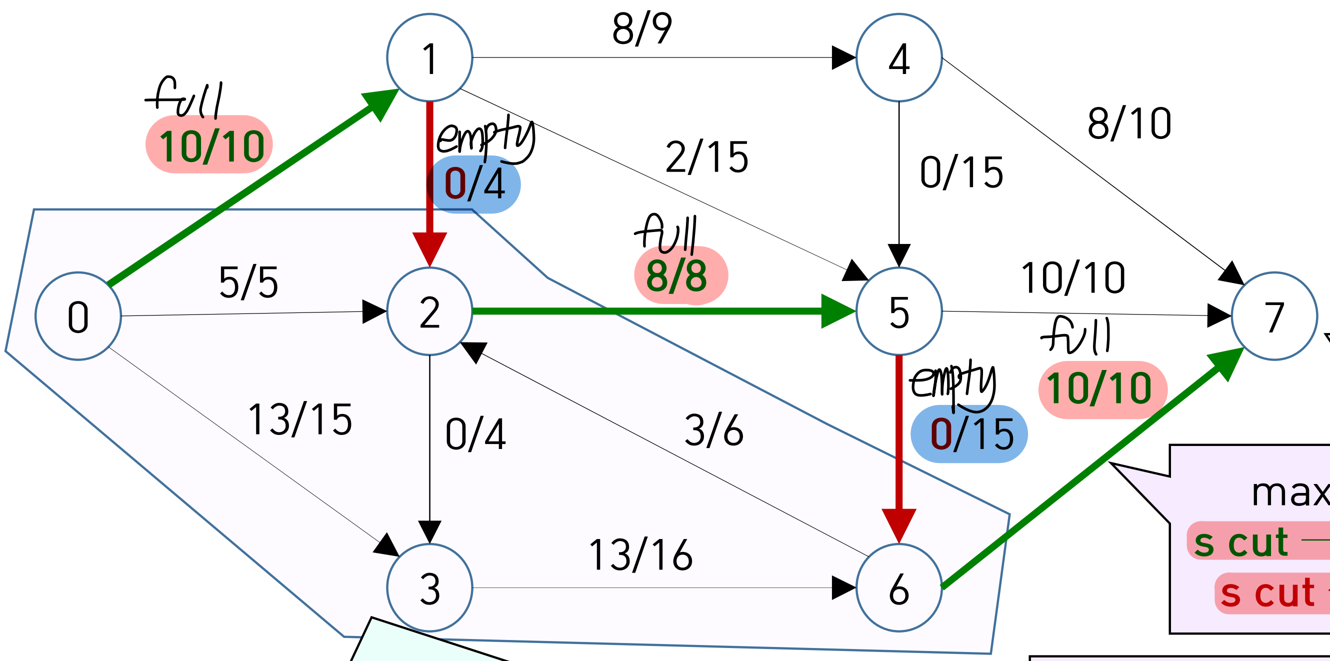
max flow = 5



이 그래프의 min cut capacity = 5

Max flow 찾았다면
더는 도착지에 도달하는 augmenting path 찾을 수 없음 의미
이때 출발지에서 augmenting path 탐색해 보면
min cut 이루는 경계는 넘어갈 수 없어 경계 내만 탐색됨
따라서 visited[v]=True면 v는 출발지 쪽 cut에 속하고
visited[v]=False면 v는 도착지 쪽 cut에 속함

max flow의 해로부터
min cut의 capacity뿐 아니라 min cut에 속한 정점도 이끌어낼 수 있음



Max flow 찾았으면
더는 t에 도달하는 augmenting
path 찾을 수 없음 의미. 이때는
s cut과 t cut 경계에서 탐색 중단됨

max flow 결과 보면 min cut 단면에서
s cut → t cut 방향 간선은 모두 **가득 차 있고**
s cut ← t cut 방향 간선은 모두 **비어 있음**

이 그래프의 min cut에서
출발지쪽 cut에 속한 정점
이들은 모두 visited[] = True

s → t 방향으로 최대 flow 흐르려면
(1) min cut에서 s → t 방향으로는 최대한 흘러야 함
또한 (2) s ← t 방향으로는 가능한 안 흘러야 함
s 쪽으로 들어오는 flow는 다시 t쪽으로 나가야 하는
데, 그러면 s → t 방향 flow를 줄여야 하므로

```
# Perform BFS to find vertices reachable from s along with shortest paths to them
def hasAugmentingPath(self):
    self.edgeTo = [None for _ in range(self.g.V)]
    self.visited = [False for _ in range(self.g.V)]

    q = Queue()
    q.put(self.s)
    self.visited[self.s] = True
    while not q.empty():
        v = q.get()
        for e in self.g.adj[v]:
            w = e.other(v)
            if e.remainingCapacityTo(w) > 0 and not self.visited[w]:
                self.edgeTo[w] = e
                self.visited[w] = True
                q.put(w)

    return self.visited[self.t] # Is t reachable from s with current flow assignment?
```

더는 augmenting path 없어
Ford Fulkerson 알고리즘 종료했을 때
s에 도달 가능 여부 표기한 visited[v] == True 라면
v는 s쪽 min Cut에 포함된 것임

s에서 시작해
forward 간선은 flow 더할 수 있는 곳으로
backward 간선은 flow 뺄 수 있는 곳으로 따라가서
t에 도달하는 최소 간선수 경로 구하기

목적지 t에 도달할 수 있는 경로 있다면 True 반환

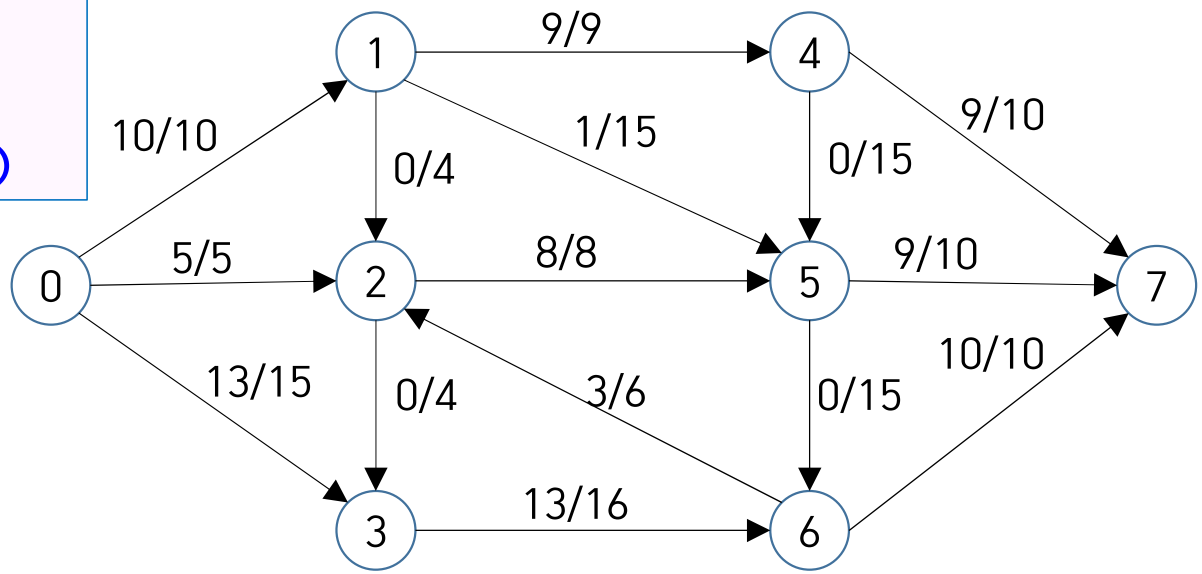
```
class FordFulkerson:
    def __init__(self, g, s, t):
        # Ford Fulkerson 알고리즘으로 s->t max flow 구해 저장
        # while loop 내부에서 hasAugmentingPath() 호출해 augmenting path 찾으며 진행
        # 더는 augmenting path 찾을 수 없다면 max flow 구한 것으로 보고 종료

    def hasAugmentingPath(self):
        # BFS 수행해 augmenting path 찾음

    def inCut(self, vertex):
        # 정점 vertex가 출발지 쪽 cut에 포함되었다면 True 반환
        return self.visited[vertex]
```

[Q] FlowGraph.py 파일의 __main__ 아래에는 FordFulkerson 클래스에 대한 unit test가 있다.
이 중 아래 코드를 주석 밖으로 꺼내 실행하면 오른쪽 그래프에 대한 max flow와 **s쪽 min cut**을 찾아준다.

(1) 출력 결과에서 보여주는 **s쪽 min cut**을 그래프의 정점에 표기해 보며 이 그래프의 min cut이 맞음을 확인하시오.

(2) 아래 코드를 잘 읽어 의미를 이해해 보시오. 오늘 실습 과제에서는 min cut을 활용하는 코드를 작성해야 합니다.

```
g8 = FlowNetwork.fromFile("flownet8.txt")
ff8 = FordFulkerson(g8, 0, g8.V-1)
print("ff8.flow", ff8.flow)
print("ff8.g", ff8.g)
print("ff8.inCut:", end=' ')
    for v in range(g8.V):
        if ff8.inCut(v): print(v, end=' ')
```



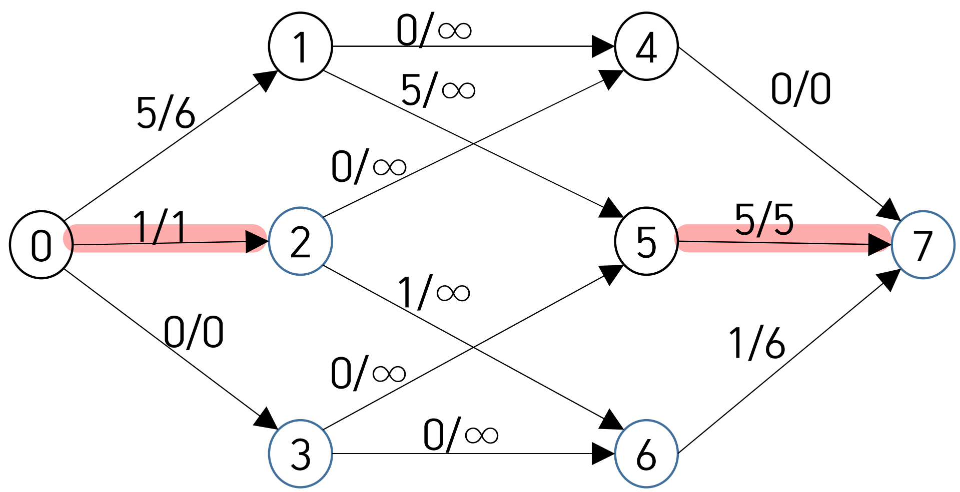
[Q] FlowGraph.py 파일의 __main__ 아래에는 FordFulkerson 클래스에 대한 unit test가 있다.
이 중 아래 코드를 주석 밖으로 꺼내 실행하면 오른쪽 그래프에 대한 max flow를 **s쪽 min cut**을 찾아준다.

(1) 출력 결과에서 보여주는 **s쪽 min cut**을 그래프의 정점에 표기해 보며 이 그래프의 min cut이 맞음을 확인하시오.

(2) 아래 코드를 잘 읽어 의미를 이해해 보시오. 오늘 실습 과제에서는 이처럼 **flow network을 만들고** 여기에 **FordFulkerson 알고리즘을 수행한 후 min cut을 확인하는** 코드를 작성해야 합니다.

```
g8m = FlowNetwork(8)
g8m.addEdge(FlowEdge(0,1,6))
g8m.addEdge(FlowEdge(0,2,1))
g8m.addEdge(FlowEdge(0,3,0))
g8m.addEdge(FlowEdge(1,4,float('inf')))
g8m.addEdge(FlowEdge(1,5,float('inf')))
g8m.addEdge(FlowEdge(2,4,float('inf')))
g8m.addEdge(FlowEdge(2,6,float('inf')))
g8m.addEdge(FlowEdge(3,5,float('inf')))
g8m.addEdge(FlowEdge(3,6,float('inf')))
g8m.addEdge(FlowEdge(4,7,0))
g8m.addEdge(FlowEdge(5,7,5))
g8m.addEdge(FlowEdge(6,7,6))
ff8m = FordFulkerson(g8m, 0, g8m.V-1)
print("ff8m.flow", ff8m.flow)
print("ff8m.g", ff8m.g)
print("ff8m.inCut:", end=' ')
    for v in range(g8m.V):
        if ff8m.inCut(v): print(v, end=' ')
```

→ 0, 1, 4, 5



Max Flow and Min Cut

Max Flow와 Min Cut 문제의 관계 이해하고 Baseball Elimination 문제 해결에 적용해 보기

01. Max Flow 문제 정의 및 예습자료 주요내용 복습
02. Ford-Fulkerson 알고리즘
03. Min Cut 문제 정의 및 Max Flow 문제와의 연관성
04. Maxflow-mincut 활용 예: Baseball Elimination 문제
05. 실습: Baseball Elimination 구현

이론 수업 중 실습을 병행하며 진행하므로
첨부 코드를 미리 다운 받아 실행 가능하게 준비해 두세요.

Baseball Elimination 문제: 운동경기의 승/패/남은 경기 주어졌을 때, 100% 1순위가 될 수 없는 팀과 이유 파악

팀별 남은경기

<입력>

	Team		승	패	남은경기	Giants	Lions	Dinos	Eagles
A		Giants	83	71	8	-	1	6	1
B		Lions	80	79	3	1	-	0	2
C		Dinos	78	78	6	6	0	-	0
D		Eagles	77	82	3	1	2	0	-

<출력> 1순위에서 100% 제거되는 팀은? 이유는?

1위는 **승수가 가장 높은** 팀 (총 경기 수는 모든 팀이 같다고 가정. 따라서 1위는 승률도 가장 높음)

1위와 승수가 같은 것도 (tie) 1위로 봄
따라서 '1순위에서 100% 제거됨'은 **1순위와 동률도 절대 될 수 없다**는 뜻

무승부는 없다고 가정

'제거될 가능성이 높음'이 아니라 수학적으로 **100% 임을 보여야 함**
(**즉 1위가 될 가능성이** 0.00000…001% 도 없이 **0%**임을 보여야 함)

Baseball Elimination 문제: 운동경기의 승/패/남은 경기 주어졌을 때, 100% 1순위가 될 수 없는 팀과 이유 파악

<입력>

	Team		승	패	남은경기	Giants	Lions	Dinos	Eagles
A		Giants	83	71	8	-	1	6	1
B		Lions	80	79	3	1	-	0	2
C		Dinos	78	78	6	6	0	-	0
D		Eagles	77	82	3	1	2	0	-

<출력> 1순위에서 100% 제거되는 팀은? 이유는?

- Eagles는 100% 1순위에서 제거됨
- Eagles는 **최대 ≤80회 (77+3) 승리**할 수 있음
- Giants는 이미 이보다 높은 83승

Baseball Elimination 문제: 운동경기의 승/패/남은 경기 주어졌을 때, 100% 1순위가 될 수 없는 팀과 이유 파악

<입력>

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

<출력> 1순위에서 100% 제거되는 팀은? 이유는?

- Lions도 100% 1순위에서 제거됨.
- Lions는 **최대 ≤83회 (80+3) 승리**할 수 있음
- Giants 혹은 Dinos가 ≥84승 할 수 있음

- 더 순위 낮은 Dinos는
- 제거 안 됨

따라서 **승, 패, 남은 경기 수** 뿐 아니라
어떤 팀 간 몇 경기가 남았는지도 고려해야 하는 간단하지 않은 문제

	Team		승	패	남은경기	Dinos	Landers	Bears	Twins	Heros
A		Dinos	75	59	28	-	3	8	7	3
B		Landers	71	63	28	3	-	2	7	4
C		Bears	69	66	27	8	2	-	0	0
D		Twins	63	72	27	7	7	0	-	0
E		Heros	49	86	27	3	4	0	0	-

1순위에서 100% 제거되는 팀은? 이유는?

	Team		승	패	남은경기	Dinos	Landers	Bears	Twins	Heros
A		Dinos	75	59	28	-	3	8	7	3
B		Landers	71	63	28	3	-	2	7	4
C		Bears	69	66	27	8	2	-	0	0
D		Twins	63	72	27	7	7	0	-	0
E		Heros	49	86	27	3	4	0	0	-

1순위에서 100% 제거되는 팀은? 이유는?

- Heros는 100% 1순위에서 제거됨. **최대 ≤76회 (49+27) 승리**할 수 있음
- 나머지 4개 팀의 승수 합 = **75 + 71 + 69 + 63 = 278**
- 나머지 4개 팀 간 남은 경기 수 = **3 + 8 + 7 + 2 + 7 = 27**
- 나머지 4개 팀 간 남은 경기 마쳤을 때 평균 승수 = (278 + 27) / 4 = 76.25

2개보다 더 많은 팀 (≥2) 간 경기 수 고려해야 제거 이유 설명 가능한 경우도 많아
고려할 경우의 수 많음

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

- 각 팀의 제거 여부 확인 위해
- 1) 팀별로 별도의 그래프 만들고
- 2) Ford-Fulkerson 사용해 제거 여부 알아냄

return (True, [Giant, Dinos])
(False, [])

- 따라서 4개 팀이 있다면
- 4개의 그래프 각각에 대해 max flow 찾아 답하게 됨

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

- 앞으로는 B팀(Lions)의 100% 제거 여부 확인한다고 가정
- 이를 위해 B팀에게 최대한 유리하게 진행해 보고
- 여전히 B팀이 1위가 될 수 없는지 확인

- B팀은 남은 경기 다 이긴다고 가정하고 : 80 → 83
- 나머지 팀 A,C,D 간의 경기를 (graph(flow network)와 max flow로) 시뮬레이션해
- B가 1위 될 수 있는지 확인

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

if E팀 있었다면,

s

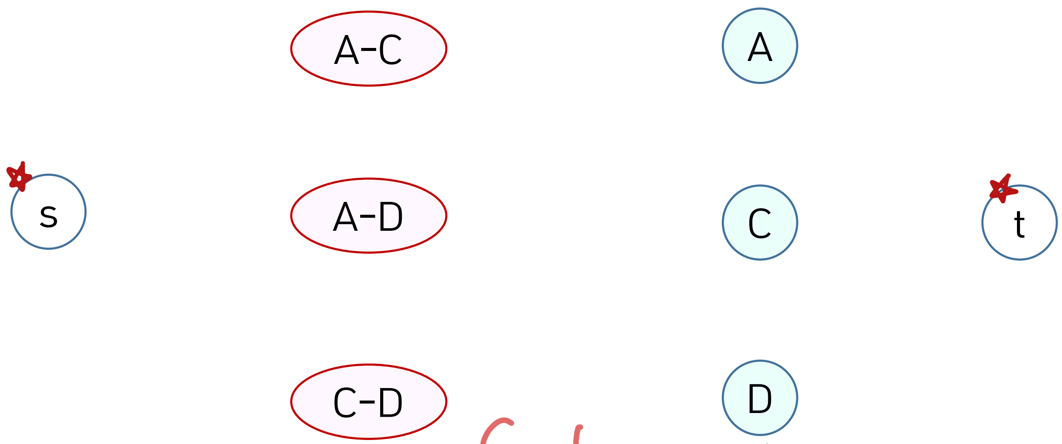
A-C
A-D
A-E
C-D
C-E
D-E

$_4C_2=6$

A
C
D
E

t

<B팀(Lions)의 제거 여부 확인하는 경우 Flow Network>



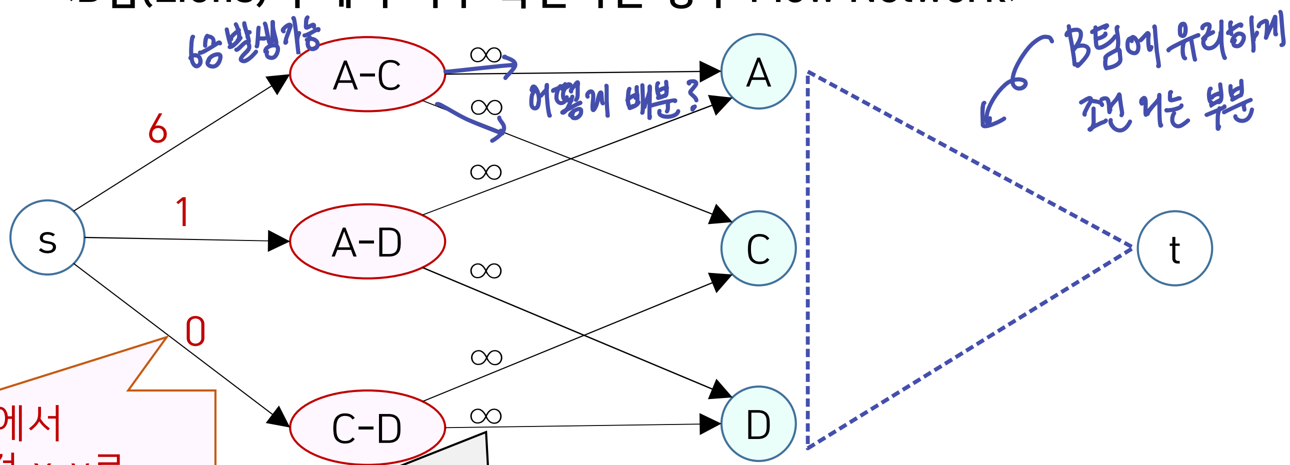
$_3C_2=6$

game 정점 만들기
(제거 대상 팀 제외한 다른 모든 2팀 간 하나씩)

team 정점 만들기
(제거 대상 팀 제외한 다른 팀마다 하나씩)

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

<B팀(Lions)의 제거 여부 확인하는 경우 Flow Network>



시작점 s에서
각 game 정점 x-y로
두 팀 x-y 간 남은 경기 수를
capacity로 가지는
간선 연결

각 game 정점 x-y에서
팀 x와 팀 y로
무한대(∞) capacity 가지는
간선 연결

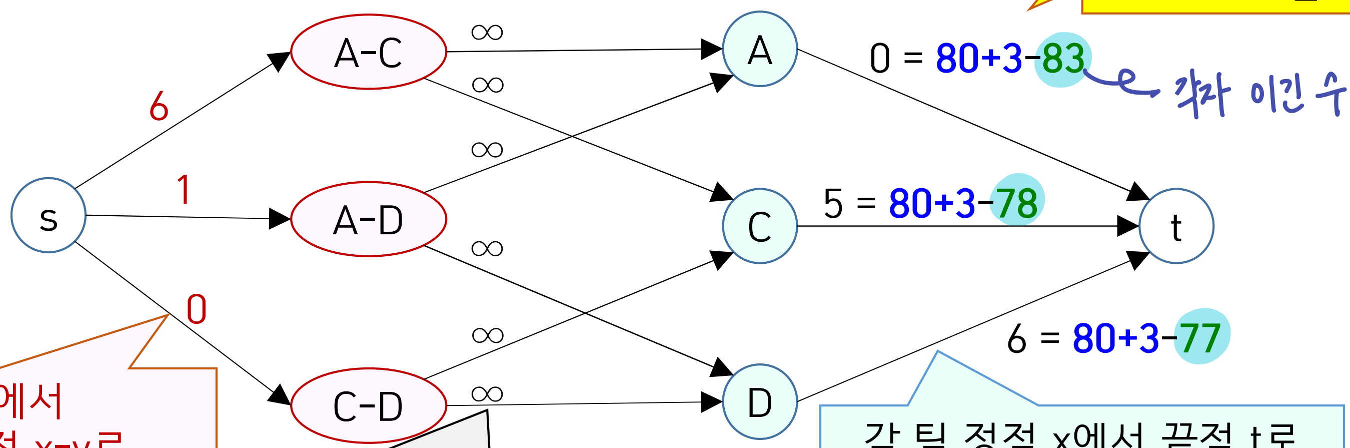
각 game x-y에 대해 **delta만큼의 승수가 배정**되며, 이것이 팀 x 혹은 팀 y로 나누어 흘러 들어갈 수 있다 는 뜻

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

제거 대상 팀(B)에 가장 유리한 상황 가정하며 (B는 모두 이기고, 다른 팀은 **B의 최대 승수** 넘지 않는), 그래도 B팀이 1위 될 수 없는지 확인하기 위함

각 팀이 **이 횟수** 이상 승리 못하도록 제한 (이 이상 승리하면 B팀은 1위가 될 수 없으므로)

<B팀(Lions)의 제거 여부 확인하는 경우 Flow Network>



시작점 s에서
각 game 정점 x-y로
두 팀 x-y 간 남은 경기 수를
capacity로 가지는
간선 연결

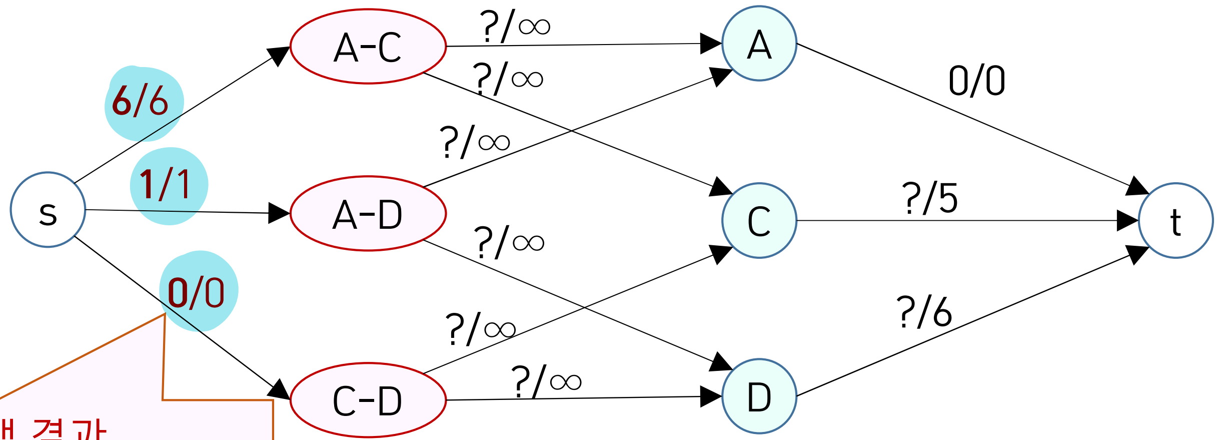
각 game 정점 x-y에서
팀 x와 팀 y로
무한대(∞) capacity 가지는
간선 연결

각 팀 정점 x에서 끝점 t로
$w_B + r_B - w_x$를
capacity로 가지는
간선 연결

w_i : 팀 i의 현재 승(win)
r_i : 팀 i의 남은 경기 수
(remaining games)

Maxflow 결과 해석 (case 1): (s에서 최대한 많이 흘려 보내본 결과)
s에서 나가는 간선이 모두 full이라면, 제거 대상 팀은 1위 될 가능성 있음

<B팀(Lions)의 제거 여부 확인하는 경우 Flow Network>



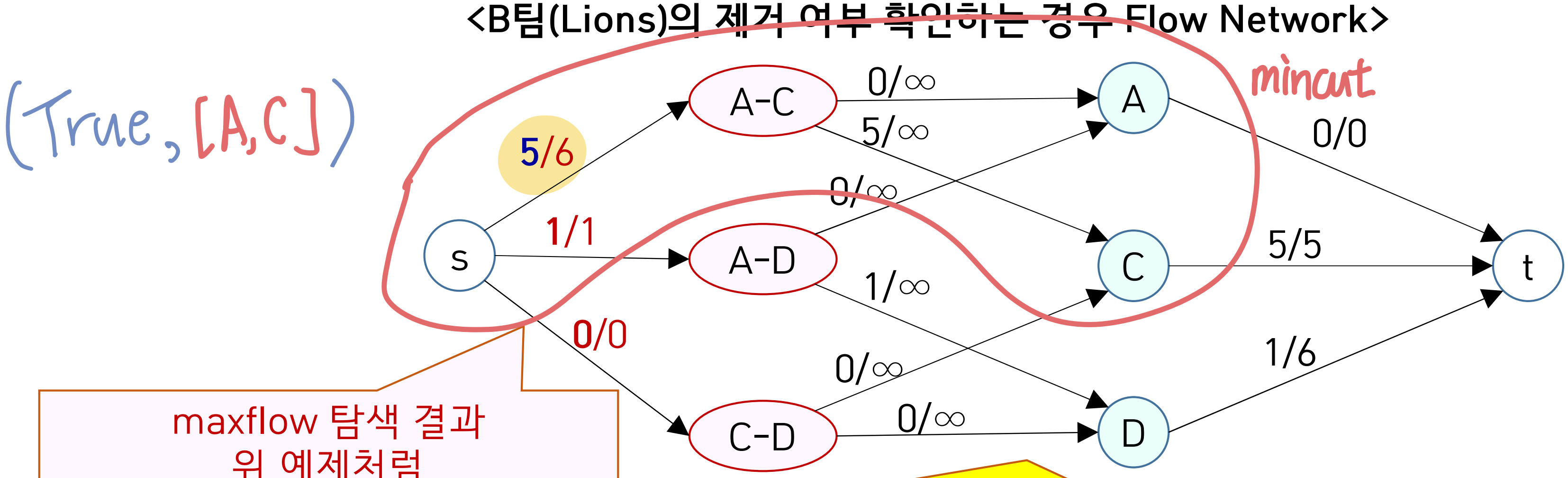
maxflow 탐색 결과
위 예제처럼
s에서 나가는 모든 간선의 capacity
를 가득 채우는 방법 있다면
B는 제거되지 않음

(제거 대상인 B팀이 1위가 될 수 있도록)
어떤 팀도 B팀의 최대 승수를 넘지 않도록
남은 경기의 승수가 배정될 가능성이
있다는 뜻이므로

(False, [])

Maxflow 결과 해석 (case 2): (s에서 최대한 많이 흘려 보내본 결과)
s에서 나가는 간선 중 full 아닌 간선 있다면, 제거 대상 팀은 1위 될 가능성 없으므로 제거

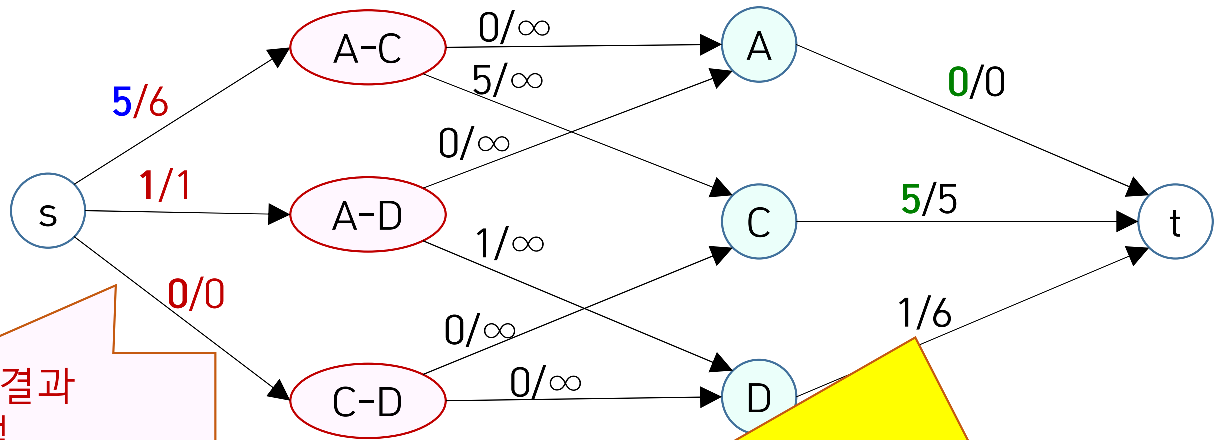
<B팀(Lions)의 제거 여부 확인하는 경우 Flow Network>



maxflow 탐색 결과
위 예제처럼
capacity가 full 아닌 간선 있다면
B는 제거됨

(제거 대상인 B팀이 1위가 될 수 있도록)
어떤 팀도 B팀의 최대 승수를 넘지 않도록
남은 경기의 승수를 배정해 보아도
배정 못하고 남는 승수 있으며
이 승수까지 배정하면 최소 한 팀은 B팀의 최대 승수 초과해
B팀은 1위에서 탈락

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

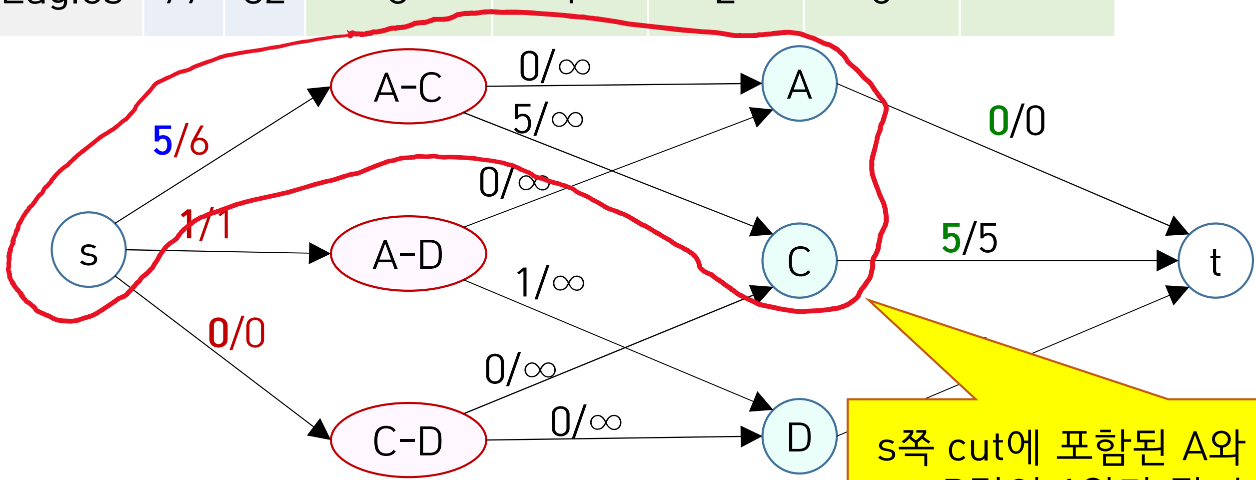


maxflow 탐색 결과
위 예제처럼
capacity가 full 아닌 간선 있다면
B는 제거됨

(제거 대상인 B팀이 1위가 될 수 있도록)
어떤 팀도 B팀의 최대 승수를 넘지 않도록
남은 경기의 승수를 배정해 보면 A팀은 **0승**, C팀은 **5승** 되고
여전히 배정 못하고 남는 승수 (A-C간 한 경기) 있음
이 승수까지 배정하면 A-C 중 최소 한 팀은
B팀의 최대 승수 초과해 B팀은 1위에서 탈락

Maxflow 결과 **제거 대상 팀은 1위 될 가능성 없다면** **mincut이 그 이유 보여줌**

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

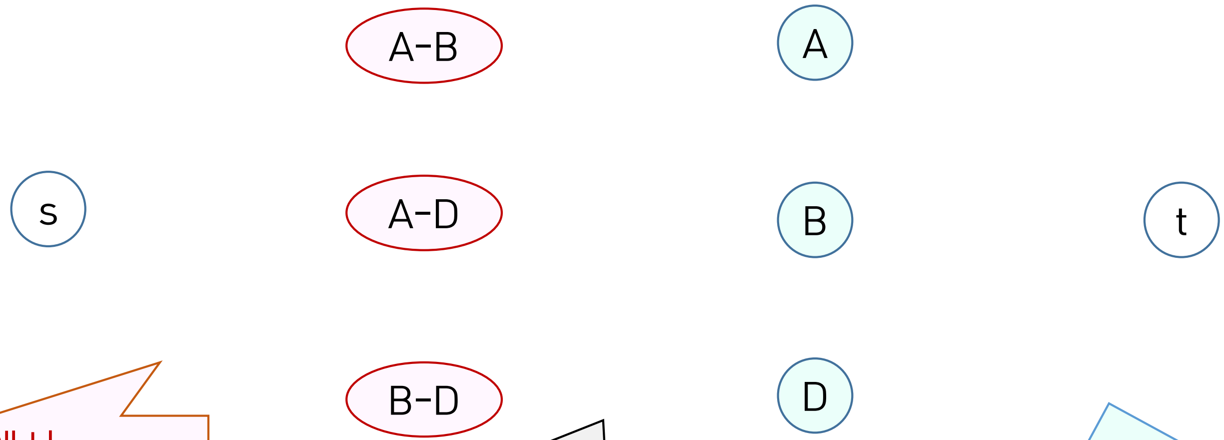


s쪽 cut에 포함된 A와 C 팀의 남은 경기가 B팀이 1위가 될 수 없는 이유 설명

[Q] C팀(Dinos)의 제거 여부를 확인하기 위한 Flow Network을 그려 보시오.

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

<C팀(Dinos)의 제거 여부 확인하는 경우 Flow Network>



w_i : 팀 i의 현재 승(win)
r_i : 팀 i의 남은 경기 수
(remaining games)

시작점 s에서
각 game 정점 x-y로
두 팀 x-y 간 남은 경기 수를
capacity로 가지는
간선 연결

각 game 정점 x-y에서
팀 x와 팀 y로
무한대(∞) capacity 가지는
간선 연결

각 팀 정점 x에서 끝점 t로
$w_C + r_C - w_x$를
capacity로 가지는
간선 연결

[Q] FlowGraph.py 파일 __main__ 아래 다음 코드를 주석 밖으로 꺼내고 실행하면
앞 페이지와 같은 flow network을 만들고 maxflow를 찾아 출력한다.
출력한 결과 각 간선에 배정된 flow를 앞 페이지에 적어 보시오.
또한 그 결과에 따라 C팀은 제거되는지, 그리고 이유는 무엇인지 설명해 보시오.

```
g8dinos = FlowNetwork.fromFile("flownet8dinos.txt") # 파일로부터 FlowNetwork 객체 생성
ff8dinos = FordFulkerson(g8dinos, 0, g8dinos.V-1) # s→t 방향으로 FF 알고리즘 실행
print("ff8dinos.flow", ff8dinos.flow) # maxflow 값 출력
print("ff8dinos.inCut:", end=' ') # s쪽 cut에 들어가는 정점 번호 출력
for v in range(g8dinos.V):
    if ff8dinos.inCut(v): print(v, end=' ')
print()
print("ff8dinos.g", ff8dinos.g) # 간선별 maxflow 배정 결과 출력
print()
```

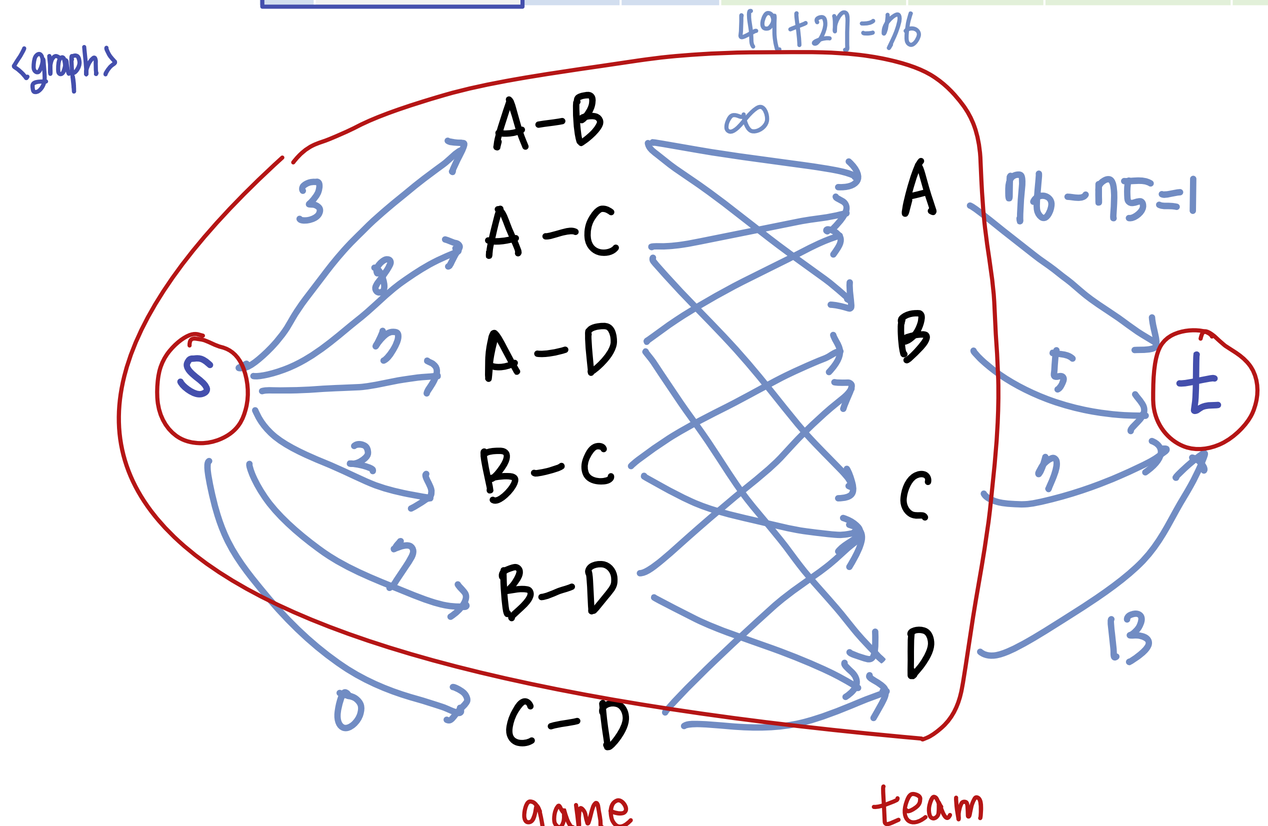
[Q] 팀이 제거되는 경우 minCut은 그 이유를 설명해 줌을 배웠다.
위 결과와 같은 경우에는 s쪽 cut에 어떤 정점이 포함되는가?

[Q] A팀(Giants)의 제거 여부를 확인하기 위한 Flow Network을 그려 보시오.
또한 Flow Network에 maxflow를 배정한 후 결과를 해석해 보시오.

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	–	1	6	1
B	Lions	80	79	3	1	–	0	2
C	Dinos	78	78	6	6	0	–	0
D	Eagles	77	82	3	1	2	0	–

[Q] E팀(Heros)의 제거 여부를 확인하기 위한 Flow Network을 그려 보시오.
(maxflow를 계산할 필요는 없음)

	Team	승	패	남은경기	Dinos	Landers	Bears	Twins	Heros
A	Dinos	75	59	28	-	3	8	7	3
B	Landers	71	63	28	3	-	2	7	4
C	Bears	69	66	27	8	2	-	0	0
D	Twins	63	72	27	7	7	0	-	0
E	Heros	49	86	27	3	4	0	0	-



(True, [A, B, C, D])

[Q] FlowGraph.py 파일 __main__ 아래 다음 코드를 주석 밖으로 꺼내고 실행하면
앞 페이지와 같은 flow network을 만들고 maxflow를 찾아 출력한다.
출력한 결과 각 간선에 배정된 flow를 앞 페이지에 적어 보시오.
또한 그 결과에 따라 E 팀은 제거되는지 답해 보시오.

```
g12heros = FlowNetwork.fromFile("flownet12heros.txt") # 파일로부터 FlowNetwork 객체 생성
ff12heros = FordFulkerson(g12heros, 0, g12heros.V-1) # s~t 대상으로 FF 알고리즘 실행
print("ff12heros.flow", ff12heros.flow) # maxflow 값 출력
print("ff12heros.inCut: ", end=' ')   # s쪽 cut에 들어가는 정점 번호 출력
for v in range(g12heros.V):
    if ff12heros.inCut(v): print(v, end=' ')
print()
print("ff12heros.g", ff12heros.g)  # 간선별 maxflow 배정 결과 출력
print()
```

[Q] 앞 페이지의 maxflow 배정 결과에 따라
E팀이 1순위에서 100% 탈락되는 이유를 설명해 보시오.

(정리) team A가 제거되는지 확인해야 한다고 가정

① maxflow-mincut 사용하지 않고도 team A가 제거되는지 확인
- team A 제외한 다른 team i에 대해
- team A가 달성할 수 있는 최대 승수 < team i의 현재 승수인 경우
- team A는 trivially 제거됨 (예: Eagles)

② ①로는 확인 불가하다면 지금까지 배운 것 처럼 flow network 만들고, maxflow 배정하고, 그 결과에 따라 team A가 제거되는지 확인

	Team	승	패	남은경기	Giants	Lions	Dinos	Eagles
A	Giants	83	71	8	-	1	6	1
B	Lions	80	79	3	1	-	0	2
C	Dinos	78	78	6	6	0	-	0
D	Eagles	77	82	3	1	2	0	-

- (정리) ② maxflow로 팀 x의 100% 1순위 탈락 여부 파악하는 방법

- 팀 x에게 가장 유리한 조건으로 진행했을 때의 결과를 flow network으로 simulation하도록 함
  - 팀 x는 남은 게임을 모두 승리하고
  - 다른 팀들은 가능하면 팀 x의 최대 승수에 도달하지 않거나 tie가 되도록 승/패 배정
  - 이렇게 배정해도 결국 팀 x의 최대 승수 넘어서는 팀이 있다면
  - 팀 x는 100% 1순위 탈락함 입증됨

Max Flow and Min Cut

Max Flow와 Min Cut 문제의 관계 이해하고 Baseball Elimination 문제 해결에 적용해 보기

01. Max Flow 문제 정의 및 예습자료 주요내용 복습

02. Ford-Fulkerson 알고리즘

03. Min Cut 문제 정의 및 Max Flow 문제와의 연관성

04. Maxflow-mincut 활용 예: Baseball Elimination 문제

05. 실습: Baseball Elimination 구현

이론 수업 중 실습을 병행하며 진행하므로
첨부 코드를 미리 다운 받아 실행 가능하게 준비해 두세요.

실습 목표: Baseball Elimination 기능 구현

- 이번 시간에 배운 Ford Fulkerson 알고리즘, Flow Graph, Flow Edge 활용해
- Baseball elimination 문제에 대한 답 구하는 기능 구현

① Ford Fulkerson 알고리즘으로 풀고자 하는 **문제를 Flow Graph로 표현**하고
② **Ford Fulkerson 알고리즘을 적용**한 후
③ **결과 해석**해 보기

프로그램 구현 조건

- 야구 경기의 중간 결과가 주어졌을 때, 1위 가능성이 없는 팀을 찾는 함수 구현

  def isEliminated (self, teamName):

- 입력 self: BaseballElimination 클래스 객체로, 위 함수는 이 클래스의 멤버 함수임
  - 이 클래스에는 주요 멤버변수로 self.teams, wins, losses, remaining, against 등이 있으며
  - 이들은 팀별 승패 및 남은 경기수를 나타냄 (이어지는 페이지에서 더 자세히 설명)
- 입력 teamName: 1순위에서 제거되었는지 확인하고자 하는 **팀의 이름**
  - 존재하지 않는 팀 이름은 입력으로 들어오지 않음

- 반환 값: 2-tuple로 (True or False, 리스트)
  - teamName이 제거되어야 하는 경우 (True, 제거 이유가 되는 팀의 리스트) 반환
  - 제거 이유가 되는 팀들은 **멤버변수 self.teams[]에 저장된 순서대로** 리스트에 담겨야 함
  - teamName이 제거되지 않는 경우 (False, []) 반환

결과를 print하지 말고 반환하세요.

- 이번 시간에 제공한 코드 FlowGraph.py에 위 함수 추가해 제출

프로그램 구현 조건

- 최종 결과물로 FlowGraph.py 파일 하나만 제출하며, 이 파일만으로 코드가 동작해야 함

- import는 원래 FlowGraph.py 파일에서 하던 패키지 외에는 추가로 할 수 없음 (Path, Queue, timeit)

- 위 파일에 포함된 FlowEdge, FlowNetwork, FordFulkerson 클래스는 반드시 사용해야 함
  - 간선은 FlowEdge 객체로 나타냄
  - 이러한 간선을 포함한 그래프는 FlowNetwork 객체로 나타냄
  - 이러한 FlowNetwork 객체에 대한 maxflow-mincut 해를 구하기 위해 FordFulkerson 클래스 활용

- FlowGraph.py 내에 이미 구현되어 있던 코드는 제거하거나 수정하지 말 것
  - 단 __main__ 아래의 코드는 테스트 위해 변경/추가해도 괜찮음
  - 각자 테스트에 사용하는 모든 코드는 반드시 if __name__ == “__main__”: 아래에 넣어
  - 제출한 파일을 import 했을 때는 실행되지 않도록 할 것