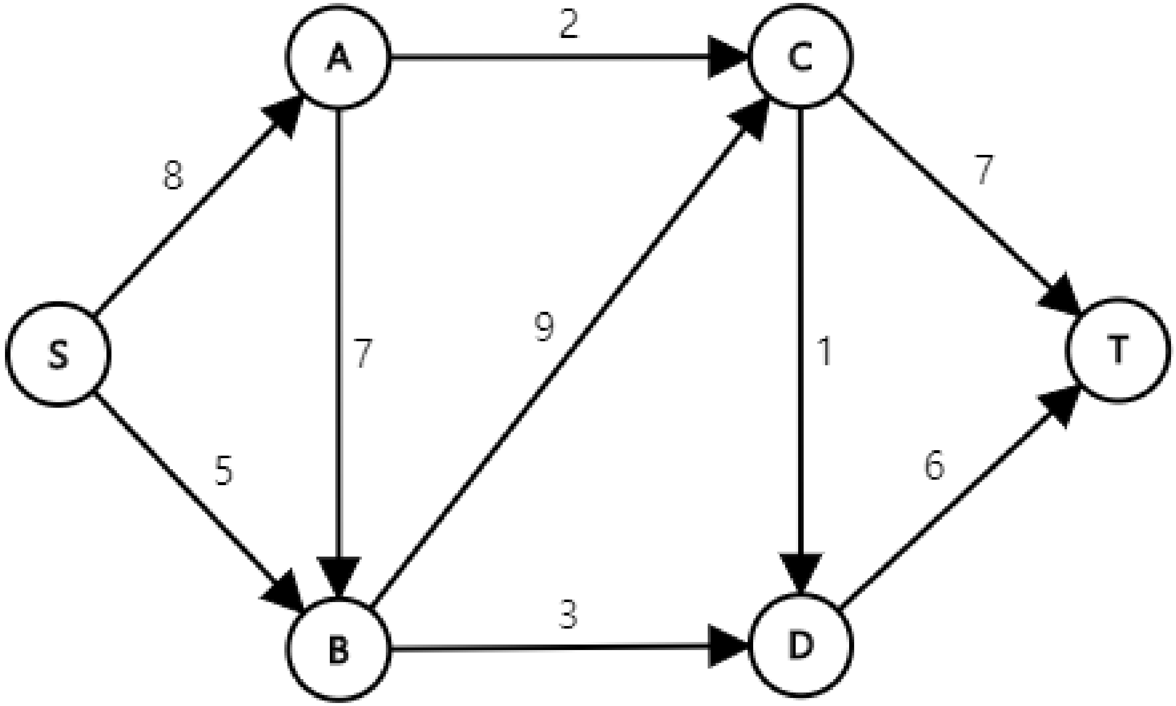
Network Flow

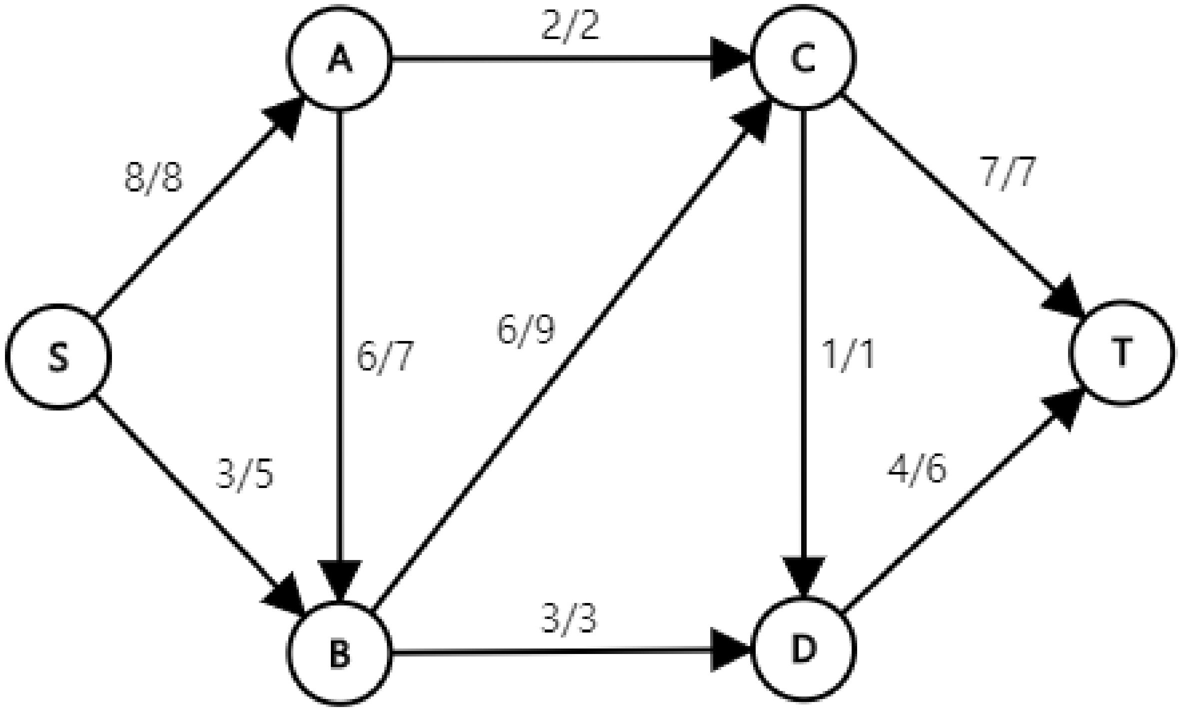
2020 Winter 중급

20141284 이기현



각 도로는 한 번에 이동가능한 차량의 수가 정해져 있다.

S에서 T로 이동할 때 한번에 도착 가능한 차의 수는?



각 도로는 한 번에 이동가능한 차량의 수가 정해져 있다.

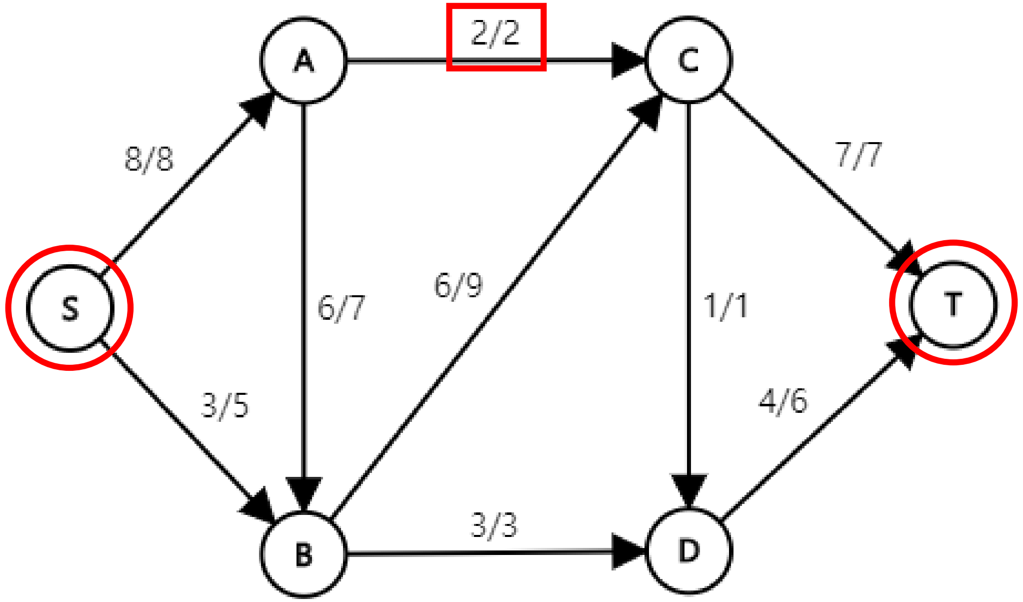
S에서 T로 이동할 때 한번에 도착 가능한 차의 수는? ⇒ 11대

용어 정리

- Flow network(유량 그래프) : 각 edge가 용량과 유량을 갖는 방향 그래프
- capacity (c, 용량) : 유량이 흐를 수 있는 수 있는 정도
- flow (f, 유량) : 흐름..?
- source (S, 소스) : 시작 위치
- sink (T, 싱크) : 끝 위치
- residual capacity (잔여용량)

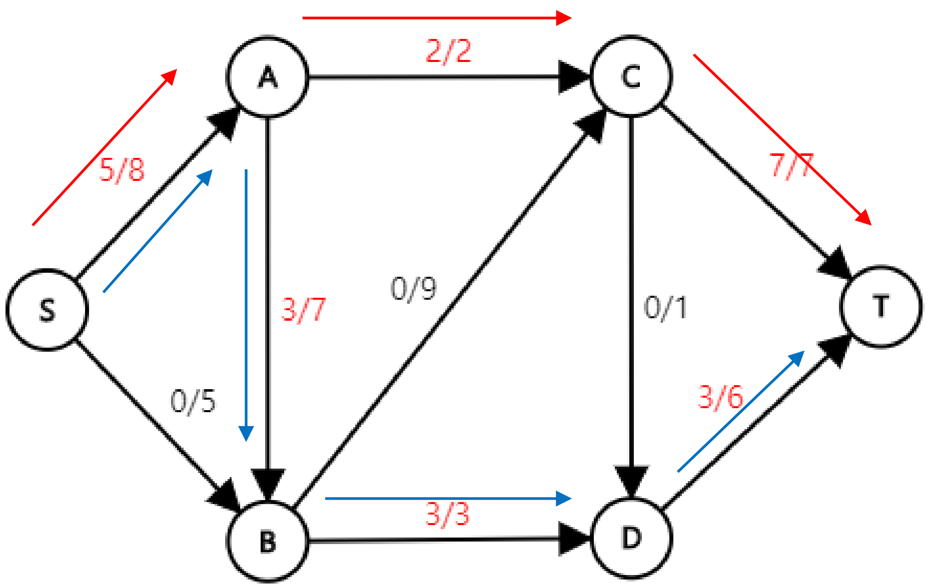
  : 용량 – 유량

(유량/용량) -> 잔여용량 = 2-2 = 0



용어 정리

- Augmenting path (증가 경로) : 잔여 용량이 남은 edge로 이루어진 S→T의 단순 경로



Flow network (유량 그래프) 의 특징

1. 용량 제한

각 간선에 흐르는 유량은 그 간선의 용량을 넘어선 안된다.

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

Flow network (유량 그래프) 의 특징

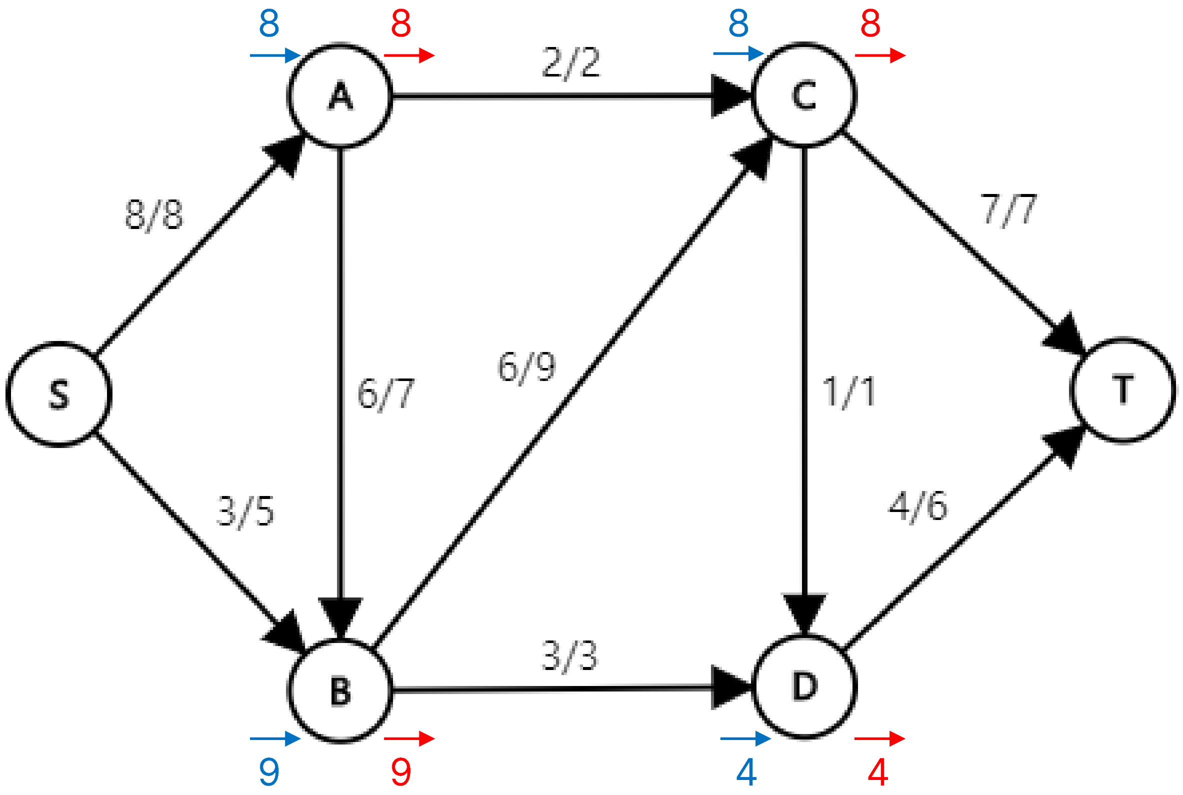
2. 유량 보존

들어오는 유량의 총합과 나가는 유량의 총합은 같다. (S, T 는 제외)

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

Flow network (유량 그래프) 의 특징

2. 유량 보존



Flow network (유량 그래프) 의 특징

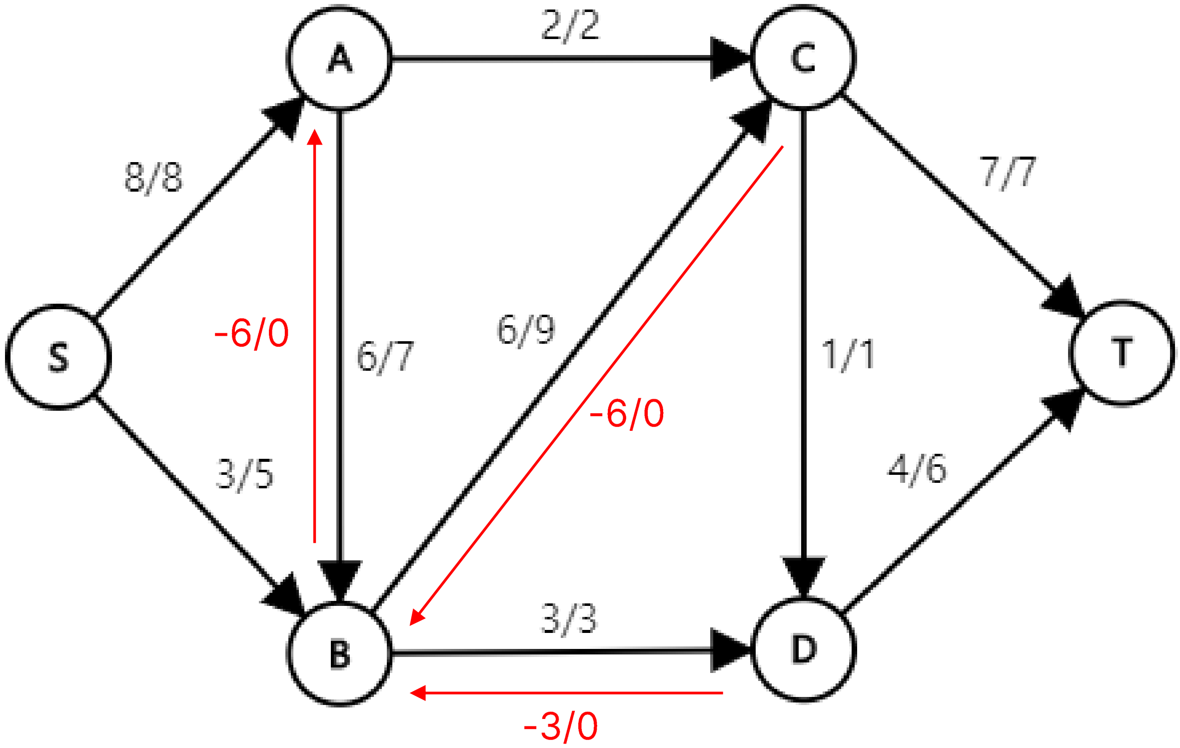
3. 유량의 대칭성

u→v로 유량이 흐를 때, 역방향으로 같은 크기의 음의 유량이 흐른다.

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

Flow network (유량 그래프) 의 특징

3. 유량의 대칭성



Network flow

- 교통망) 두 도시 사이에 이동가능한 시간당의 차량 수
- 송유관) 두 도시 사이에 보낼 수 있는 석유의 양
- 데이터 전송) 초당 전송 가능한 데이터의 양
- 등 ...

- S에서 T로 보낼 수 있는 최대 유량 (Maximum flow)를 구하자.

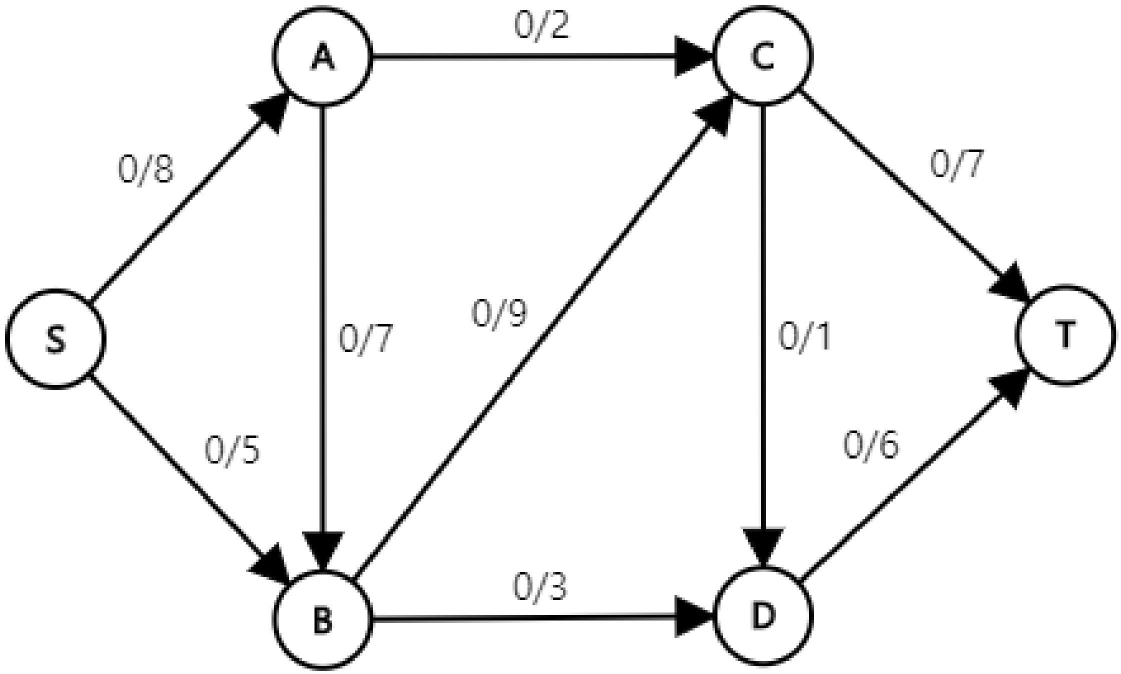
Network flow algorithm

1. S → T 의 Augmenting path 찾기

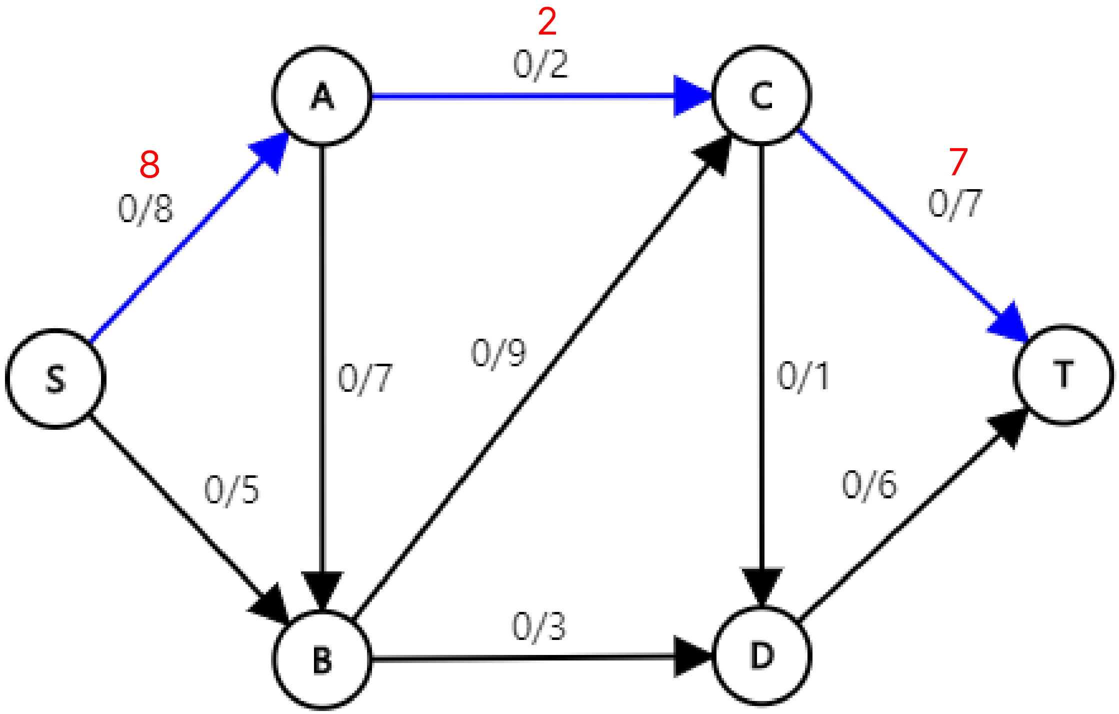
2. 찾은 Augmenting path 위 간선들의 residual capacity 중 가장 작은 값만큼 해당 path로 흘리기

3. Augmenting path가 없을 때까지 1, 2를 반복

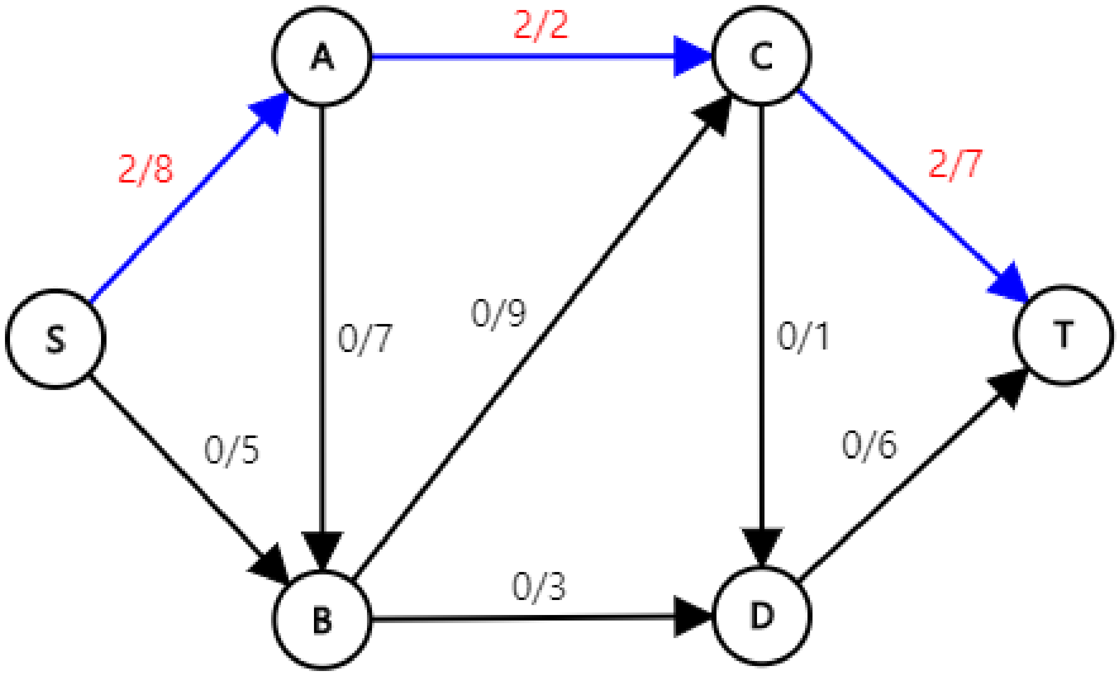
Network flow algorithm



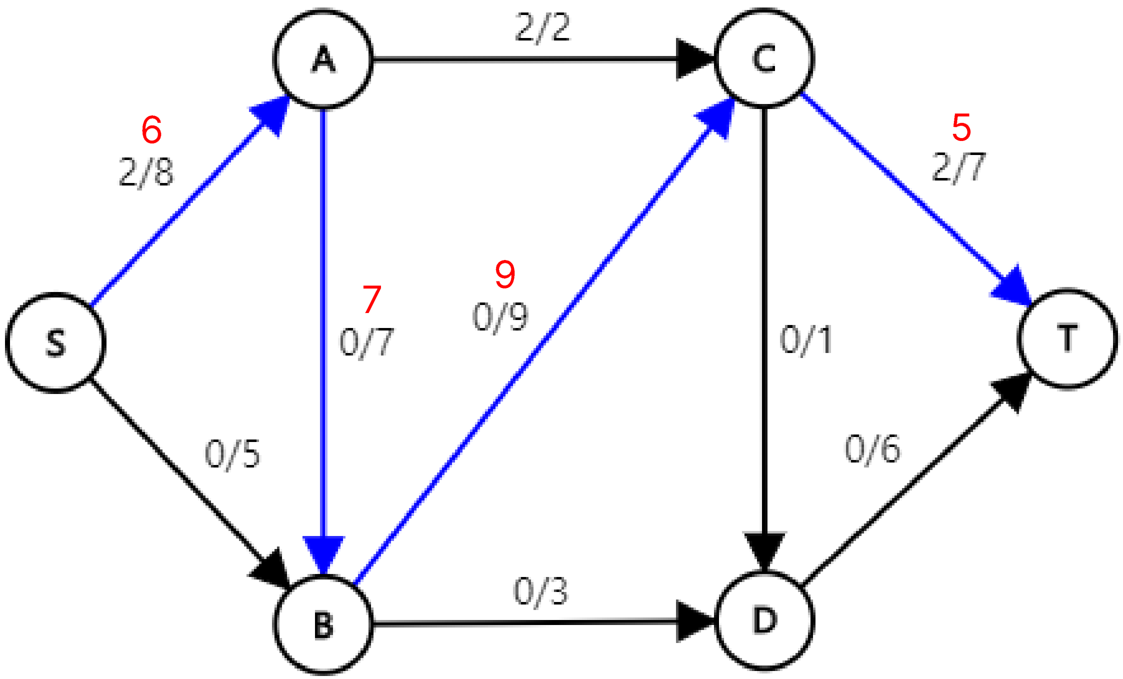
Network flow algorithm



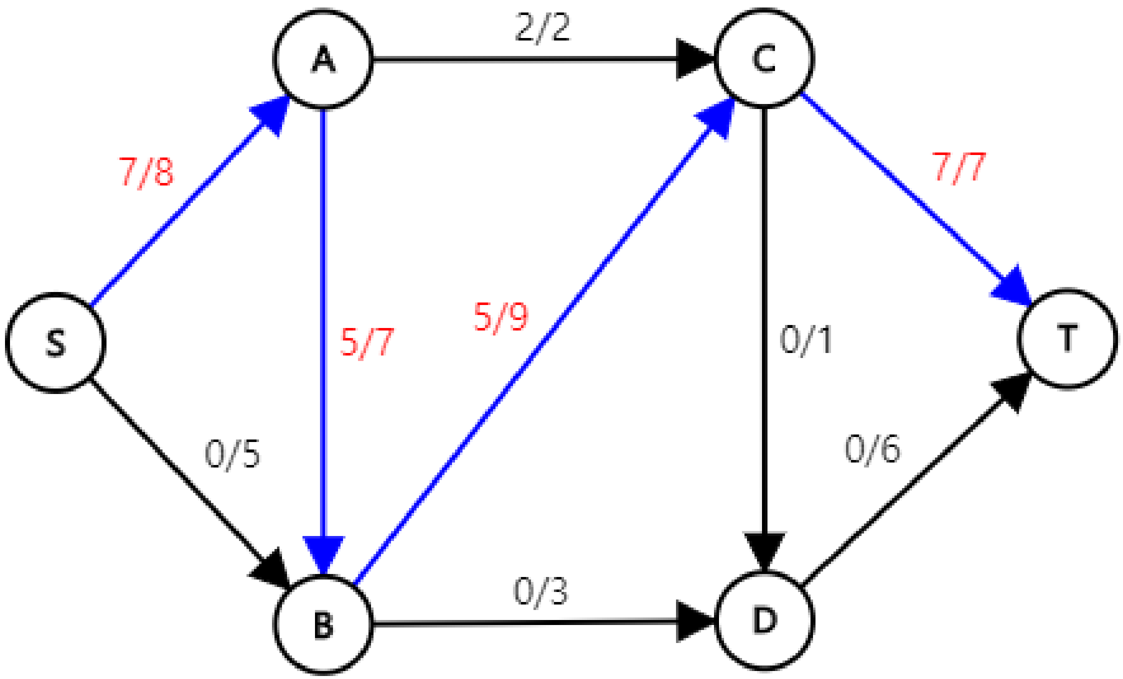
Network flow algorithm



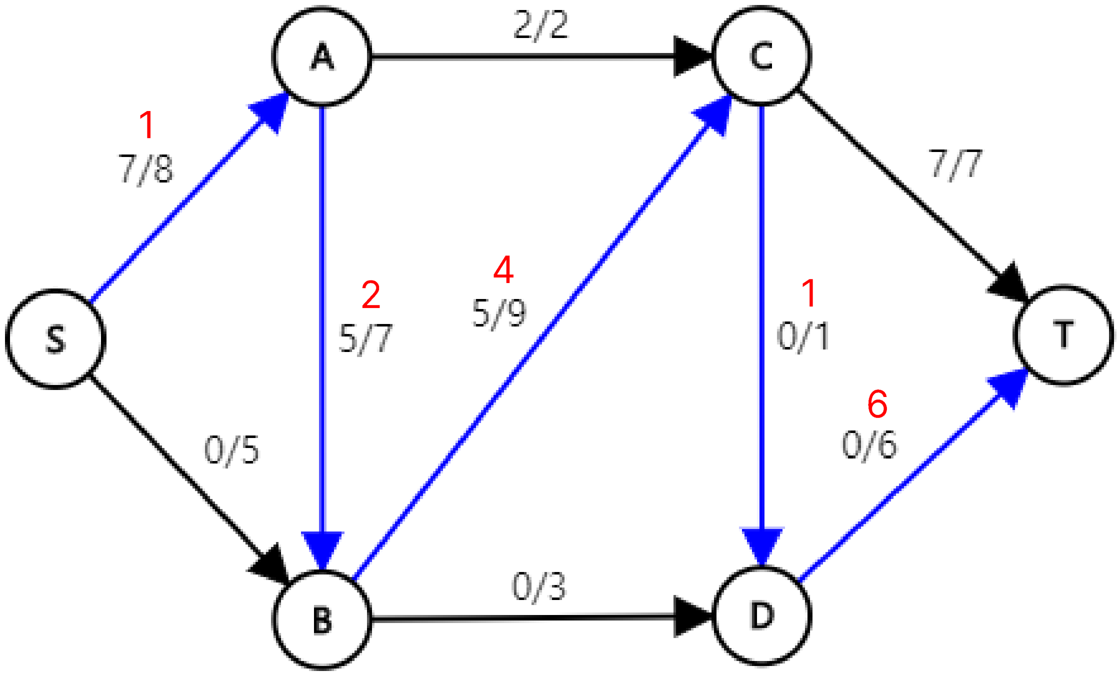
Network flow algorithm



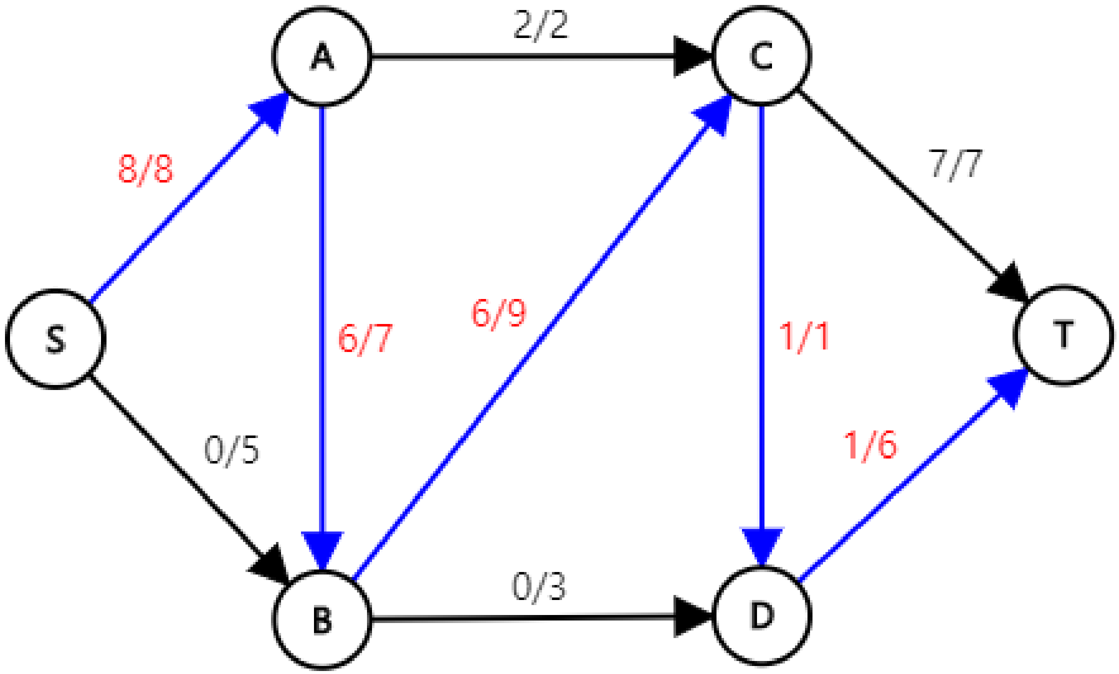
Network flow algorithm



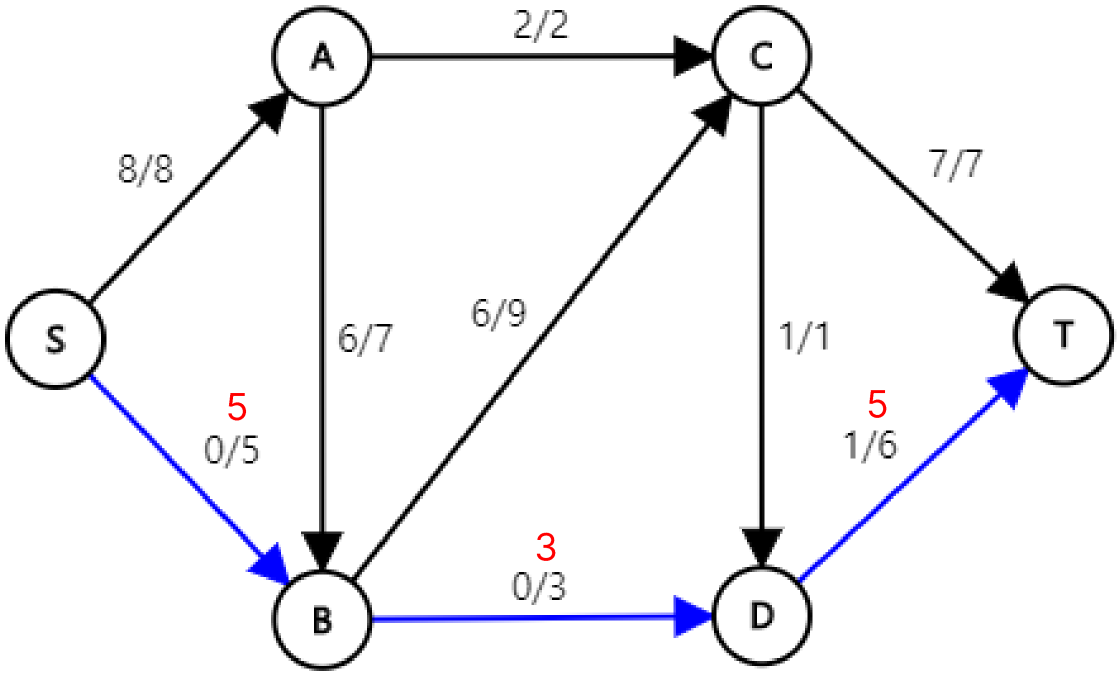
Network flow algorithm



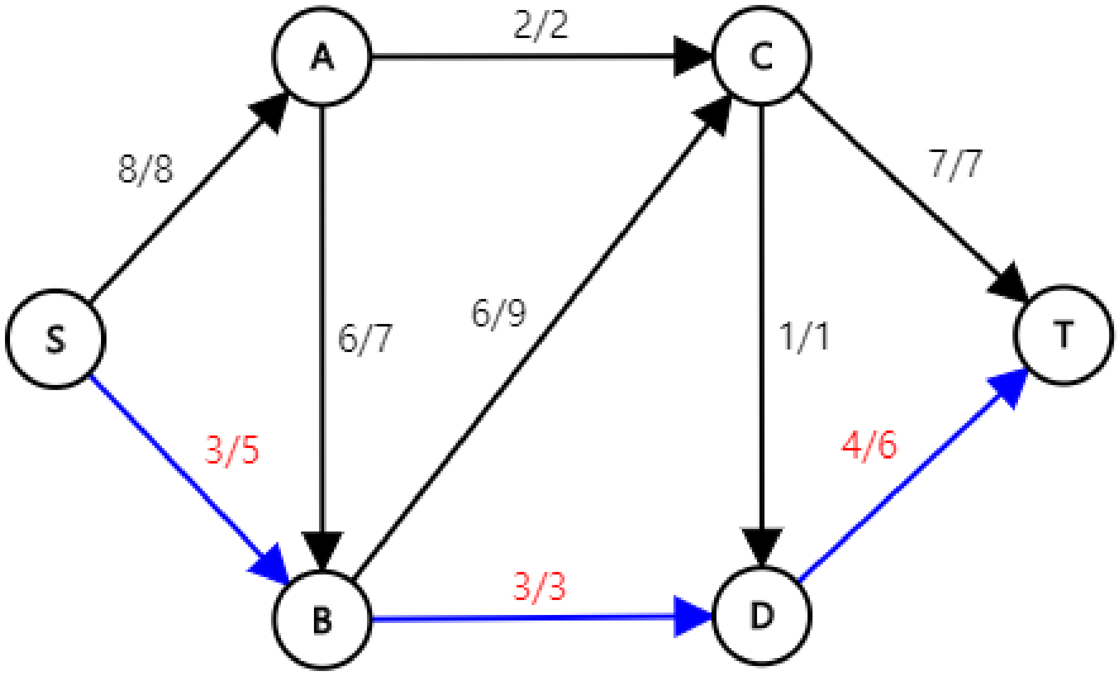
Network flow algorithm



Network flow algorithm

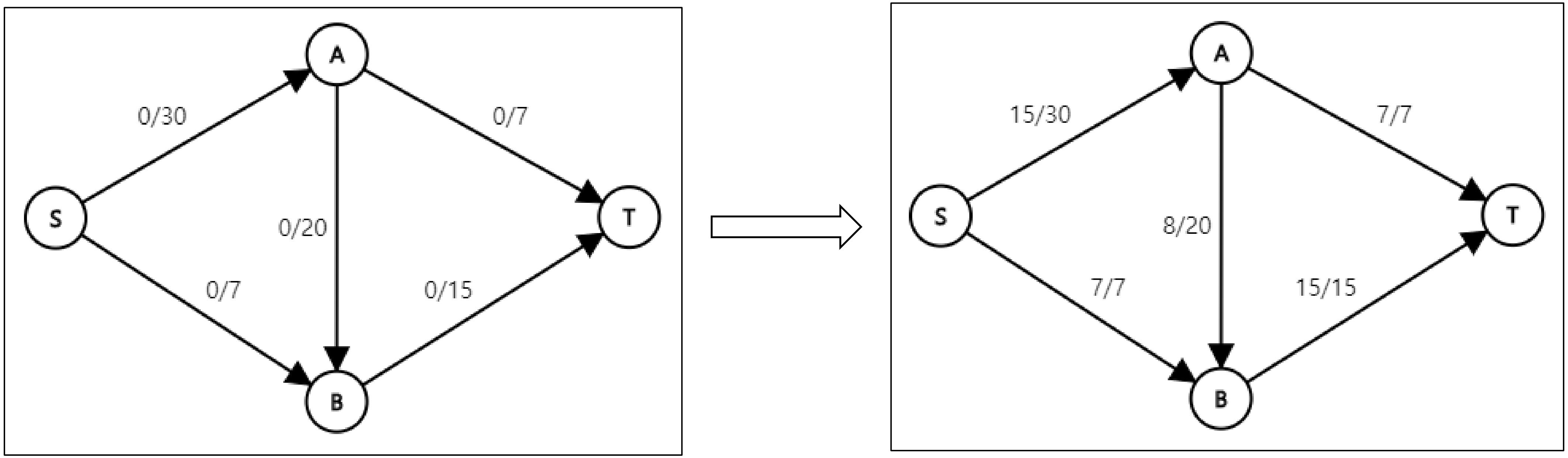


Network flow algorithm



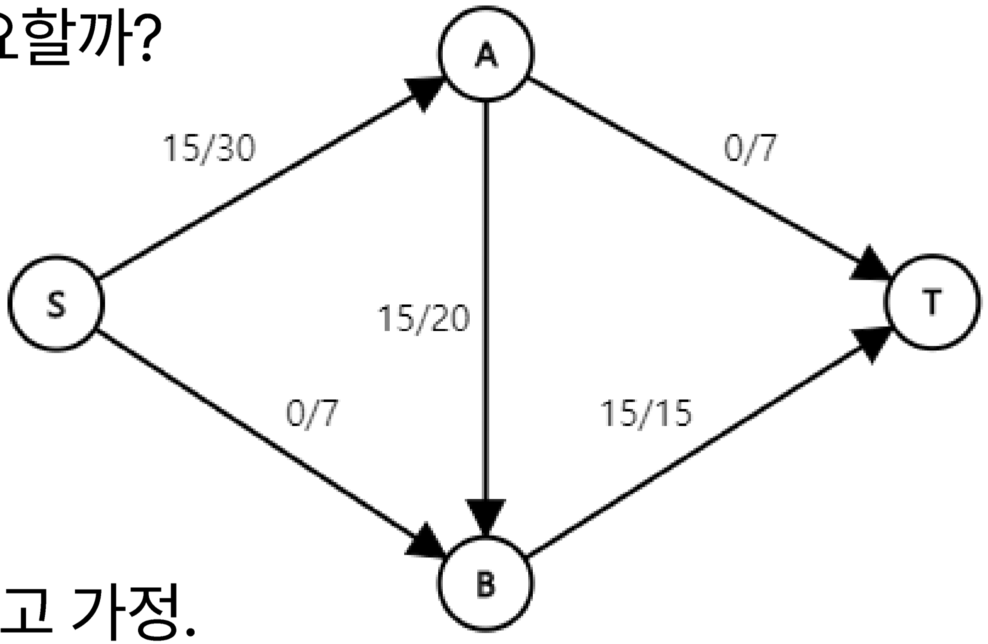
Flow network (유량 그래프) 의 특징 revisit

3. 유량의 대칭성 – 왜 필요할까?



Flow network (유량 그래프) 의 특징 revisit

3. 유량의 대칭성 – 왜 필요할까?

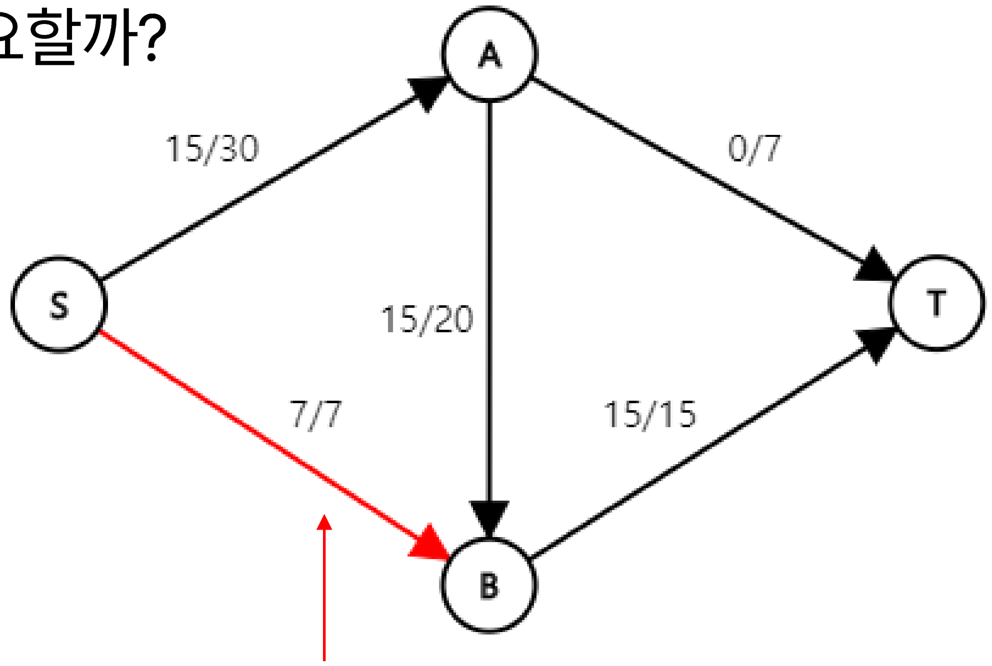


현재 flow가

이와 같이 흘러있다고 가정.

Flow network (유량 그래프) 의 특징 revisit

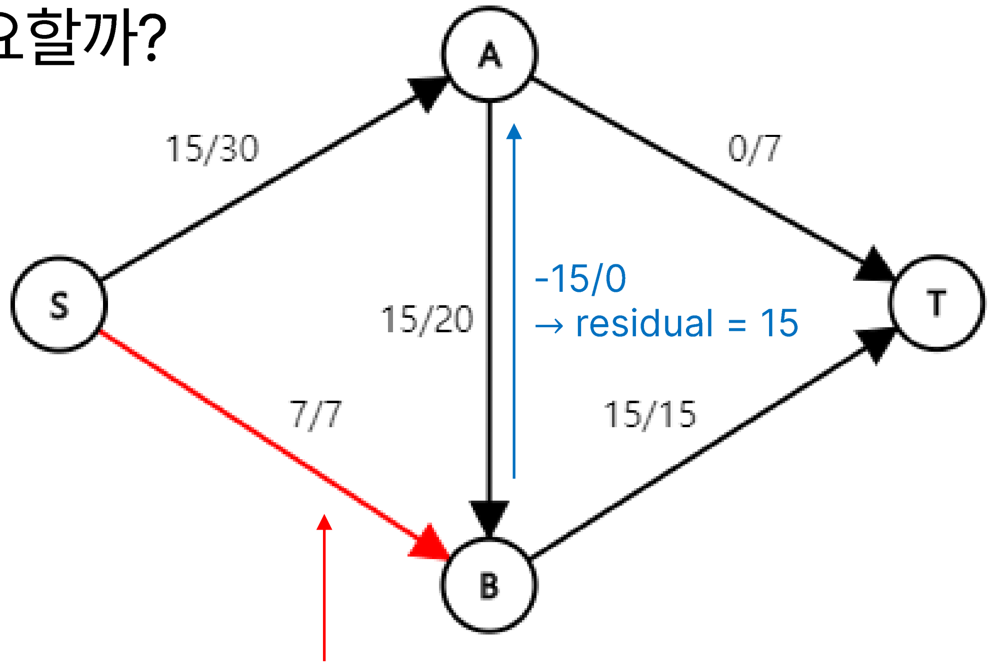
3. 유량의 대칭성 – 왜 필요할까?



이 쪽으로 flow를 흘리면? 아무것도 못한다...

Flow network (유량 그래프) 의 특징 revisit

3. 유량의 대칭성 – 왜 필요할까?



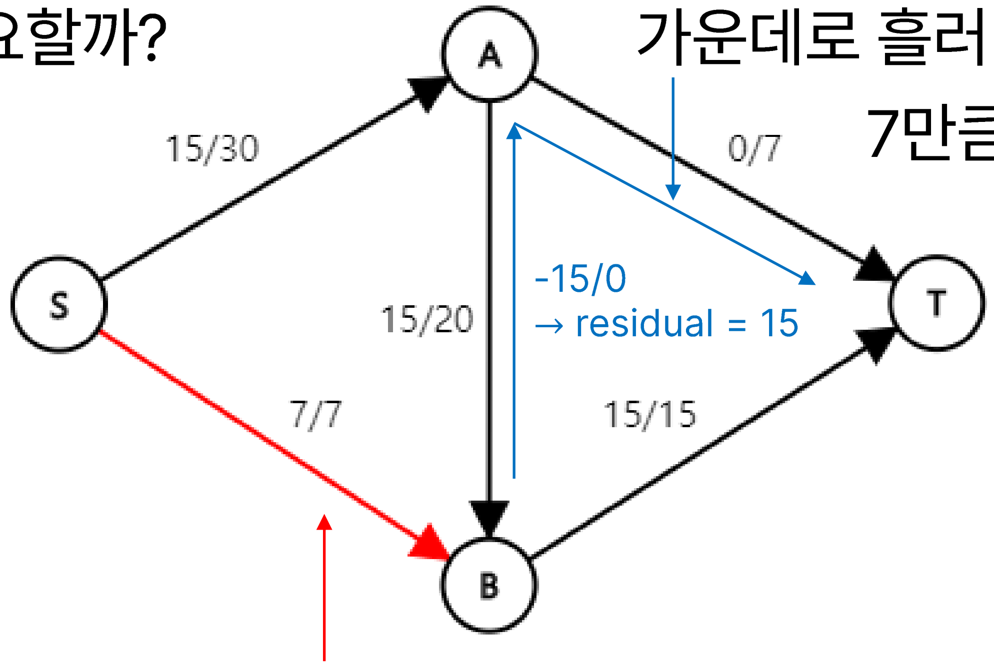
이 쪽으로 flow를 흘리면?

아무것도 못한다…

Flow network (유량 그래프) 의 특징 revisit

3. 유량의 대칭성 – 왜 필요할까?

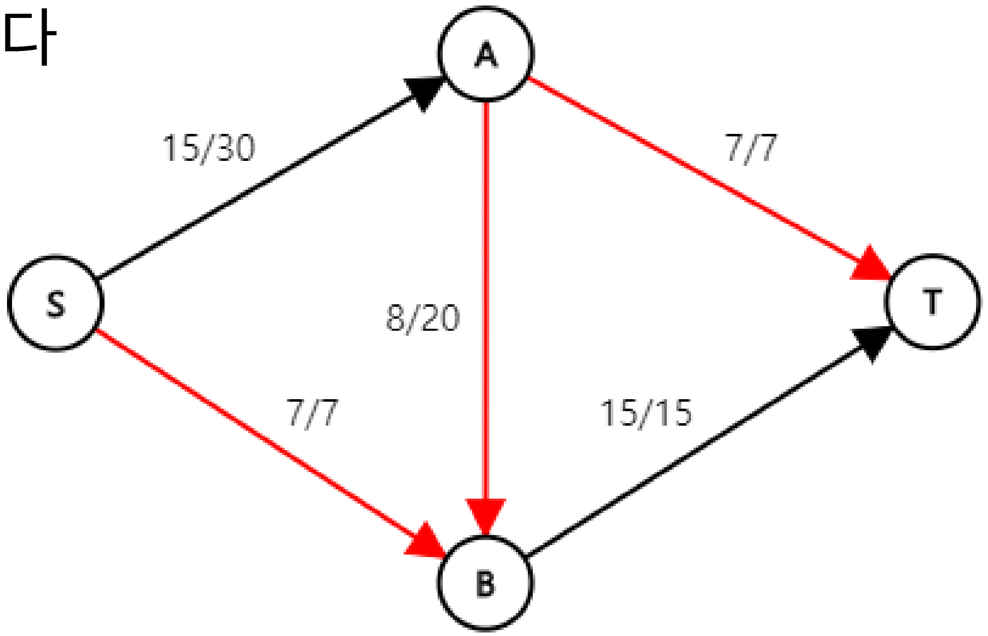
가운데로 흘러 들어갔던 유량을 이쪽으로 7만큼만 넘길 수 있다면?



이 쪽으로 flow를 흘리면? 아무것도 못한다...

Flow network (유량 그래프) 의 특징 revisit

3. 유량의 대칭성 – 필요하다



Network flow algorithm

1. S → T 의 Augmenting path 찾기 ⇒ 어떻게 찾을까??

2. 찾은 Augmenting path 위 간선들의 residual capacity 중
가장 작은 값만큼 해당 path로 흘리기

3. Augmenting path가 없을 때까지 1, 2를 반복

Ford-Fulkerson algorithm

1. “DFS”로 S → T 의 Augmenting path 찾기

2. 찾은 Augmenting path 위 간선들의 residual capacity 중 가장 작은 값만큼 해당 path로 흘리기

3. Augmenting path가 없을 때까지 1, 2를 반복

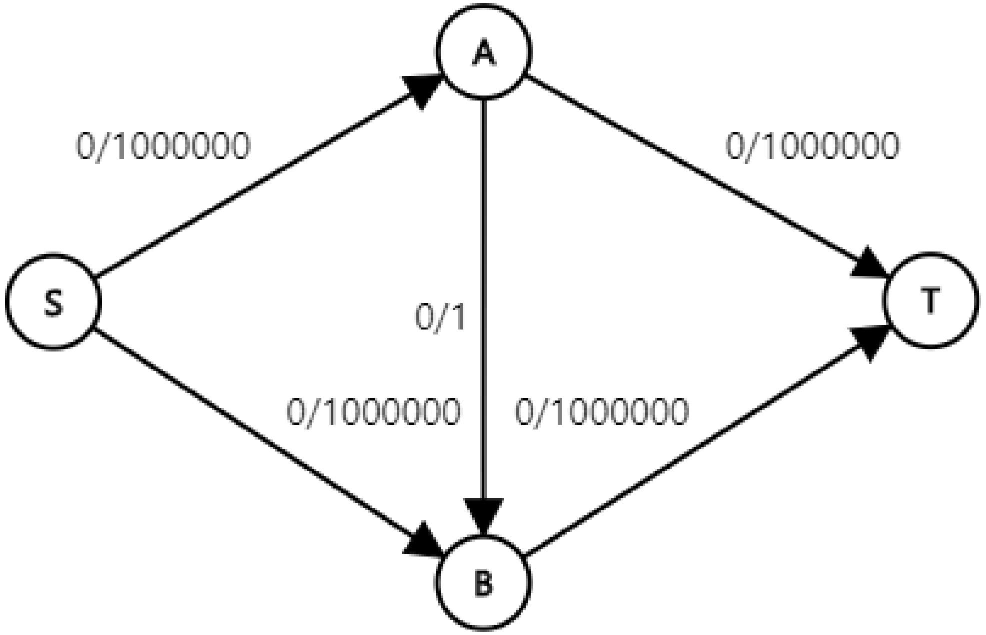
Ford-Fulkerson algorithm

1. “DFS”로 S → T 의 Augmenting path 찾기 ⇒ $O(|V| + |E|)$

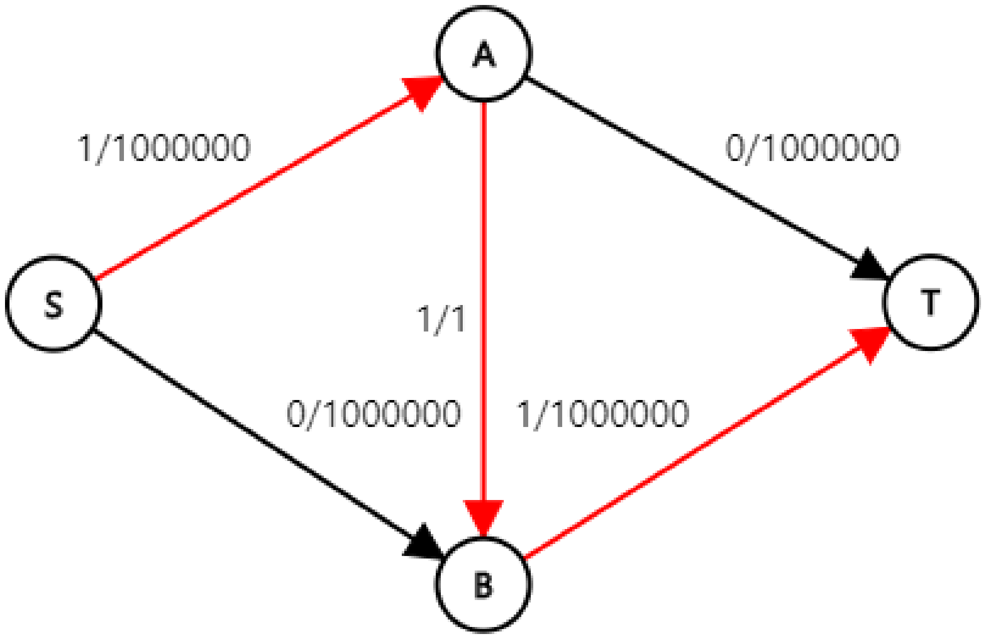
2. 찾은 Augmenting path 위 간선들의 residual capacity 중
   가장 작은 값만큼 해당 path로 흘리기 ⇒ $O(|V|)$

3. Augmenting path가 없을 때까지 1, 2를 반복

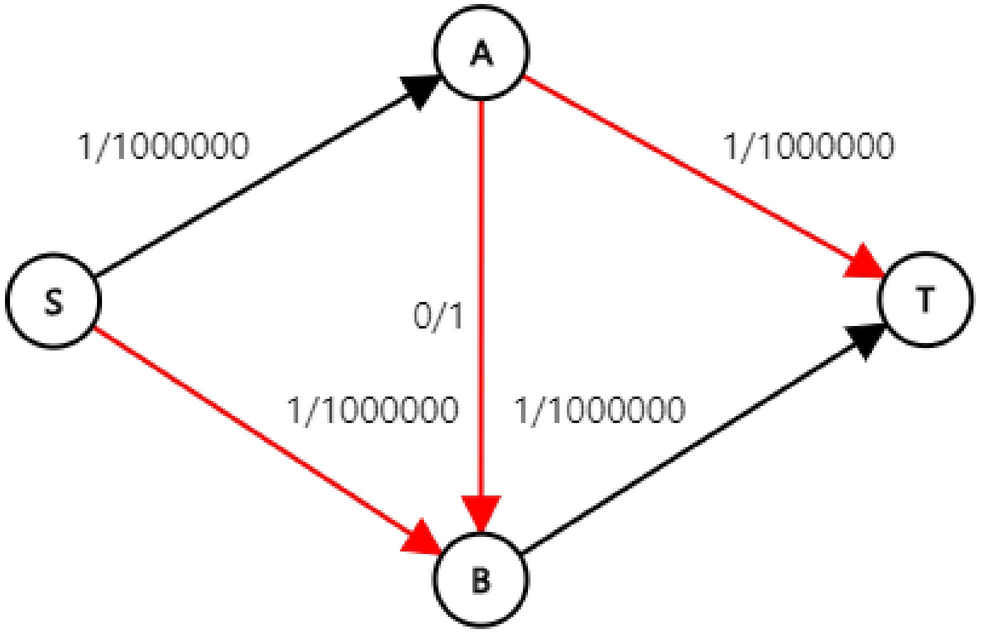
Ford-Fulkerson algorithm



Ford-Fulkerson algorithm



Ford-Fulkerson algorithm



반복 된다면?

Ford-Fulkerson algorithm

1. “DFS”로 S → T 의 Augmenting path 찾기 ⇒ $O(|V| + |E|)$

2. 찾은 Augmenting path 위 간선들의 residual capacity 중
   가장 작은 값만큼 해당 path로 흘리기 ⇒ $O(|V|)$

3. Augmenting path가 없을 때까지 1, 2를 반복 ⇒ $O(F)$, F : 흐를 수 있는 최대 유량

Ford-Fulkerson algorithm

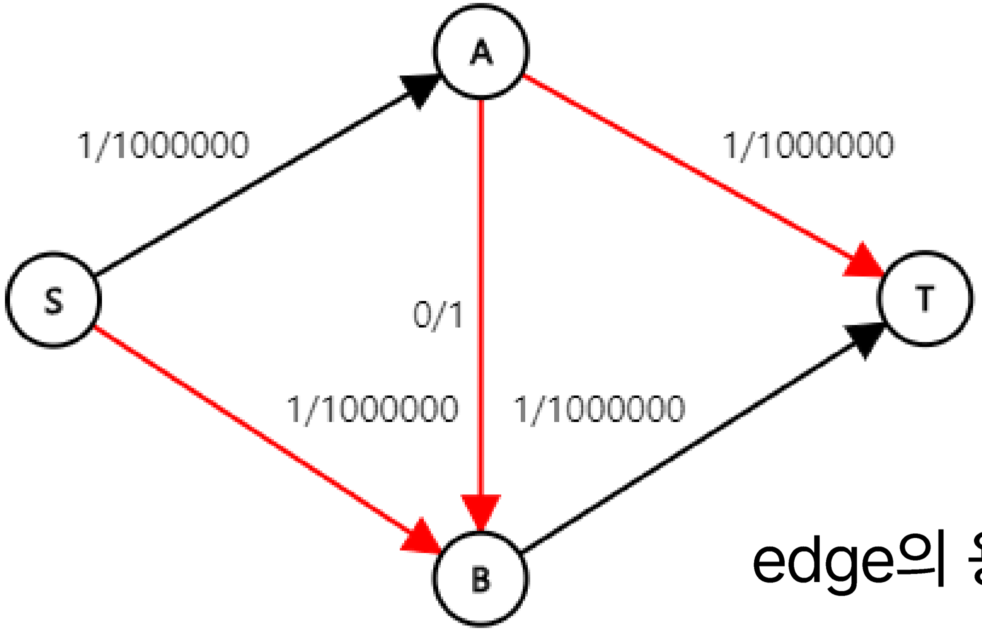
1. "DFS"로 S → T 의 Augmenting path 찾기 ⇒ $O(|V| + |E|)$

2. 찾은 Augmenting path 위 간선들의 residual capacity 중
   가장 작은 값만큼 해당 path로 흘리기 ⇒ $O(|V|)$

3. Augmenting path가 없을 때까지 1, 2를 반복 ⇒ $O(F)$, F : 흐를 수 있는 최대 유량

$$\Rightarrow O\big((|V| + |E|)F\big) = O(|E|F)$$

Ford-Fulkerson algorithm



edge의 용량에 0이 3개 더 붙으면?

Edmond-Karp algorithm

1. “BFS”로 S → T 의 “최단 길이의” Augmenting path 찾기

2. 찾은 Augmenting path 위 간선들의 residual capacity 중
   가장 작은 값만큼 해당 path로 흘리기

3. Augmenting path가 없을 때까지 1, 2를 반복

Edmond-Karp algorithm

1. “BFS”로 S → T 의 “최단 길이의” Augmenting path 찾기 ⇒ $O(|V| + |E|)$

2. 찾은 Augmenting path 위 간선들의 residual capacity 중
   가장 작은 값만큼 해당 path로 흘리기 ⇒ $O(|V|)$

3. Augmenting path가 없을 때까지 1, 2를 반복

Edmond-Karp algorithm

1. "BFS"로 S → T 의 "최단 길이의" Augmenting path 찾기 ⇒ $O(|V| + |E|)$

2. 찾은 Augmenting path 위 간선들의 residual capacity 중
   가장 작은 값만큼 해당 path로 흘리기 ⇒ $O(|V|)$

3. Augmenting path가 없을 때까지 1, 2를 반복 ⇒ $O(|V||E|)$ 증명은 구사과님이 해주십니다..(링크)

Edmond-Karp algorithm

1. “BFS”로 S → T 의 “최단 길이의” Augmenting path 찾기 ⇒ $O(|V| + |E|)$

2. 찾은 Augmenting path 위 간선들의 residual capacity 중
   가장 작은 값만큼 해당 path로 흘리기 ⇒ $O(|V|)$

3. Augmenting path가 없을 때까지 1, 2를 반복 ⇒ $O(|V||E|)$ 증명은 구사과님이 해주십니다..(링크)

$$\Rightarrow O\big((|V| + |E|)|V||E|\big) = O(|V||E|^2)$$

#17412 도시 왕복하기 5

- 1~N 번의 도시와 P개의 길이 있다.
- 1번과 2번 사이의 길은 없으며
- 1번에서 2번으로 가는 서로 다른 경로를 최대한 많이 찾으려 한다.
- 한 경로에 포함된 길은 다른 경로에 포함되면 안된다.
- 이 때, 최대 경로의 수는?
- $0 \le N \le 400, 0 \le P \le 10{,}000$

#17412 도시 왕복하기 5

- 한 경로에 포함된 길은 다른 경로에 포함되면 안된다.
  ⇒ 한 길은 한 번만 지나갈 수 있다.
  ⇒ 각 길의 최대 용량이 1이다.

#17412 도시 왕복하기 5

- 한 경로에 포함된 길은 다른 경로에 포함되면 안된다.
  ⇒ 한 길은 한 번만 지나갈 수 있다.
  ⇒ 각 길의 최대 용량이 1이다.

- S = 1, T = 2, 각 edge의 capacity = 1 일 때,
  흐를 수 있는 최대 유량을 구하라.

#17412 도시 왕복하기

- 소스 : 1 / 싱크 : 2
- 각 edge 의 capacity : 1
- 역방향 edge의 capacity : 0

```
    int n, p, S, T;
    int c[mxn][mxn], f[mxn][mxn];
    vector<vector<int> > adj(mxn);

void addedge(int u, int v) {
    c[u][v] = 1;
    adj[u].push_back(v);
    adj[v].push_back(u);
}
inline int residual(int cur, int next) {
    return c[cur][next] - f[cur][next];
}
int main() {
    cin >> n >> p;

    for (int i = 0; i < p; ++i) {
        int u, v; cin >> u >> v;
        addedge(u, v);
    }
```

#17412 도시 왕복하기 5

- BFS를 통해 “최단 경로의”
  Augmenting path 구하기

- prev : 경로 탐색을 위한 이전 노드 저장
- 35 ~ 36 line
  방문하지 않고 잔여용량이 남은 노드만
- 43 line
  T로 가는 경로가 존재 X

```
28 while (1) {
29     vector<int> prev(mxn, -1);
30     queue<int> q;
31     q.push(S);
32     while (!q.empty() && prev[T] == -1) {
33         int cur = q.front(); q.pop();
34         for (int next : adj[cur]) {
35             if (prev[next] != -1) continue;
36             if (residual(cur, next) <= 0) continue;
37             prev[next] = cur;
38             q.push(next);
39             if (next == T) break;
40         }
41     }
42 
43     if (prev[T] == -1) break;
```

#17412 도시 왕복하기

- “최단 경로” 가 존재할 때, 유량 흘리기

- prev 배열을 이용해 경로를 따라가며
  흘릴 수 있는 최소 잔여용량 구하기
- 49 ~ 50 line
  정방향 간선엔 flow를 흘리고
  역방향 간선엔 flow를 빼주기

```
45        int flow = INF;
46        for (int x = T; x != S; x = prev[x])
47            flow = min(flow, residual(prev[x], x));
48        for (int x = T; x != S; x = prev[x]) {
49            f[prev[x]][x] += flow;
50            f[x][prev[x]] -= flow;
51        }
52        totflow += flow;
53    }
54    return totflow;
```

Network flow algorithm

1. 꼭 용량이 0인 역방향 간선을 넣어주어야 한다 → 유량의 대칭성

2. 양방향성을 띄는(=방향성이 없는) 그래프라면 ?
   → 역방향 간선의 용량을 같은 크기로 하면 된다.

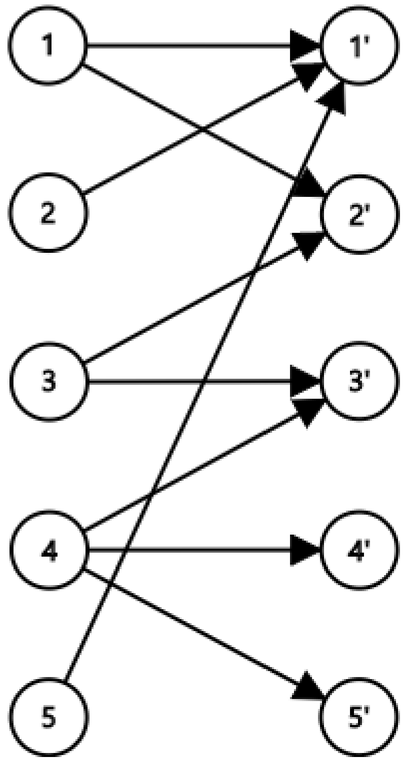
3. 간선의 "용량"에 주의 하기 & 정점 방문을 제한하기도 함
   → "모델링" 이 제일 중요하다.

4. Edmond-Karp는 Ford-Fulkerson의 개선 알고리즘으로 $\min\{O(|E|F), O(|V||E|^2)\}$ 이다.

#2188 축사 배정 4

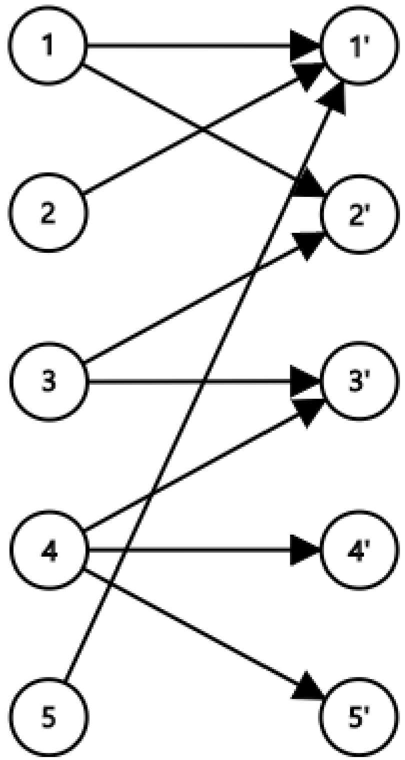
- N마리의 소와 M개의 축사가 있다.
- 각 축사엔 1마리의 소밖에 못 들어가며
- 각 소는 들어가기를 희망하는 축사가 있다.
- 각 소를 희망 축사에 넣을 때, 최대로 많이 넣을 수 있는 소는?
- $1 \le N \le 200, 0 \le P \le 200$

#2188 축사 배정 4



#2188 축사 배정 4

- 각 소는 하나의 축사에 배정
  ⇒ 각 소로부터 나가는 edge 중 1개만 선택됨

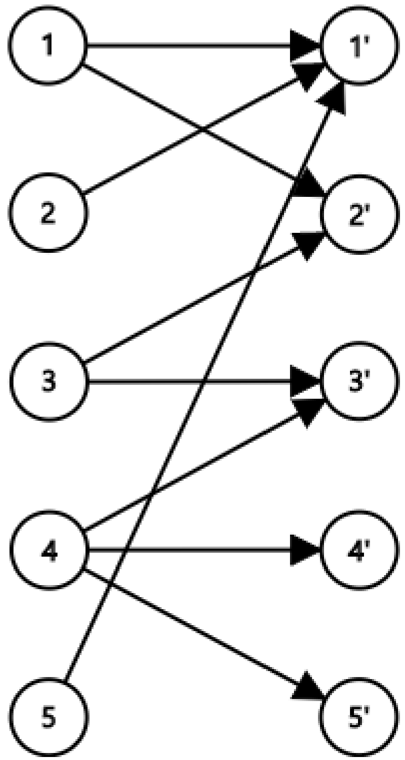


#2188 축사 배정 4

- 각 소는 하나의 축사에 배정

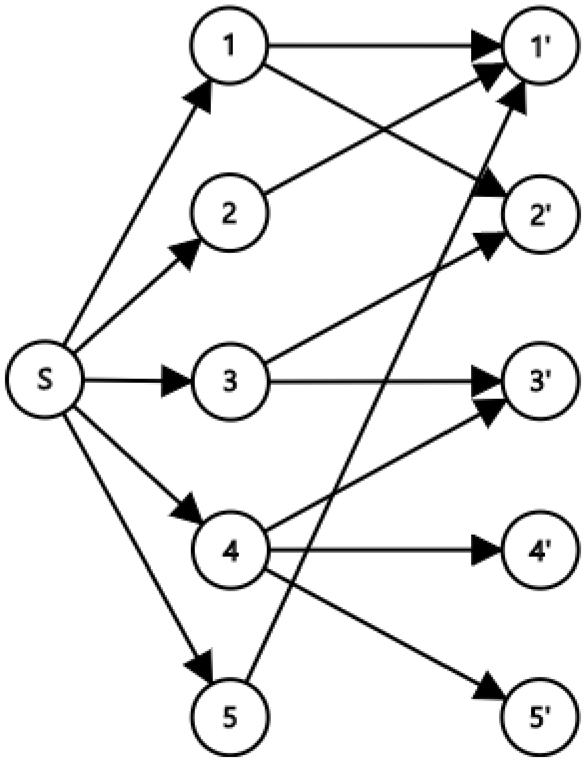
  ⇒ 각 소로부터 나가는 edge 중 1개만 선택됨

  ⇒ 각 소로부터 유량이 1만 흐를 수 있음



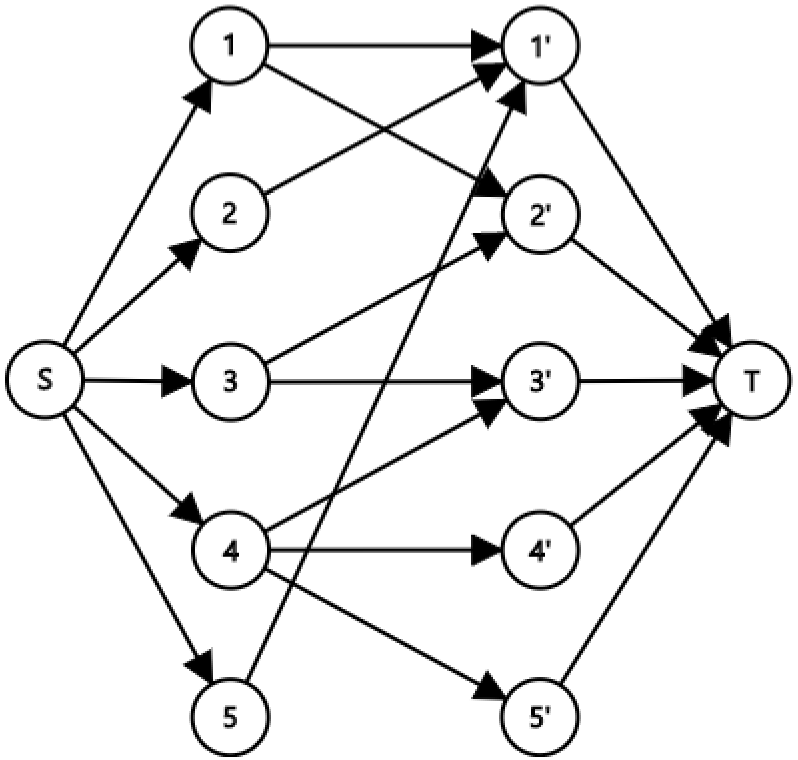
#2188 축사 배정 4

- 각 소는 하나의 축사에 배정
  ⇒ 각 소로부터 나가는 edge 중 1개만 선택됨
  ⇒ 각 소로부터 유량이 1만 흐를 수 있음
  ⇒ 각 소로 들어오는 유량도 1이어야 함



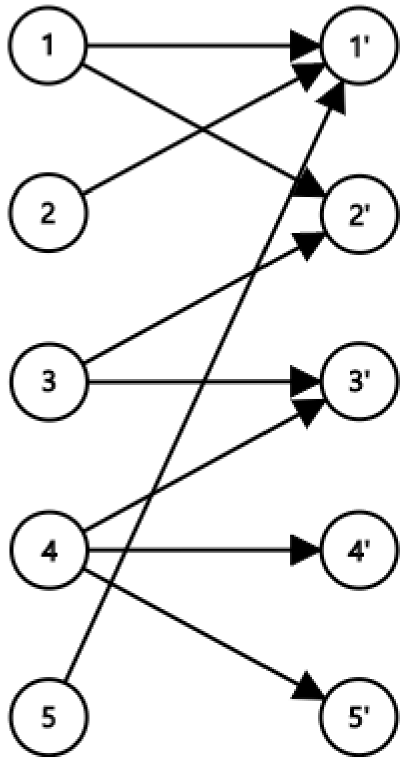
#2188 축사 배정 4

- 각 소는 하나의 축사에 배정
  ⇒ 각 소로부터 나가는 edge 중 1개만 선택됨
  ⇒ 각 소로부터 유량이 1만 흐를 수 있음
  ⇒ 각 소로 들어오는 유량도 1이어야 함
  + 각 축사에서 나가는 유량도 1



이분 그래프 (Bipartite graph)

- 모든 정점을 두 개의 그룹 A, B 로 나눌 수 있고,
- 모든 간선은 두 그룹의 정점을 연결한다.

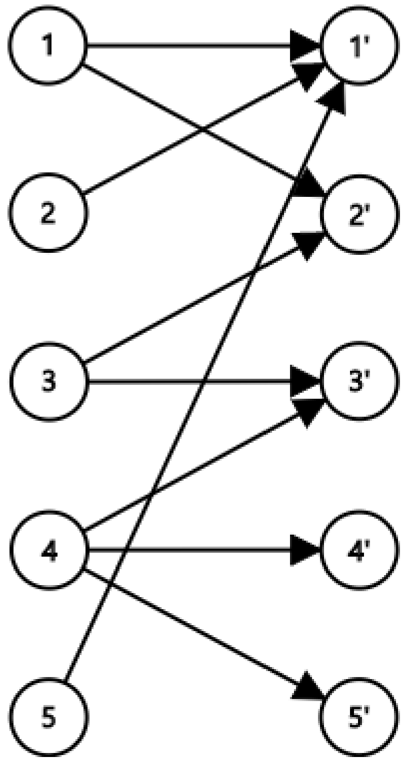


이분 매칭 (Bipartite matching)

- 모든 간선이 A → B 로 향하고
- A의 각 정점은 B의 점점 1개만 가질 수 있을 때
- A 그룹에서 B 그룹으로 매칭될 수 있는 최대를 구하라.

  ⇒ maximum matching (최대 매칭) 을 구하라.

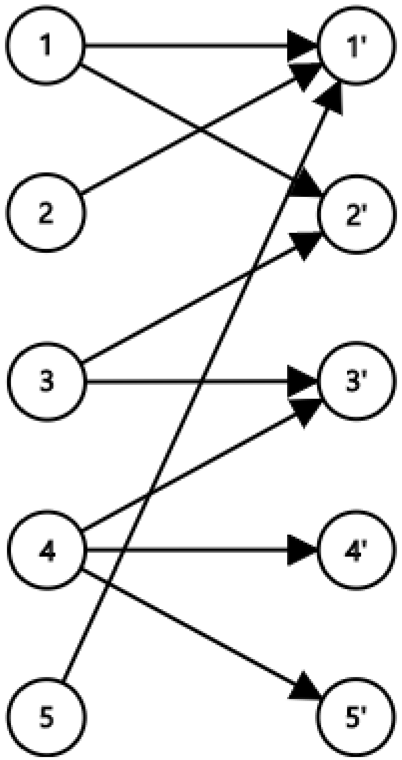
- S → A , B → T 로 향하는 용량 1인 간선을 연결할 때
- S → T 로 가는 최대 유량을 구하는 것과 같다.



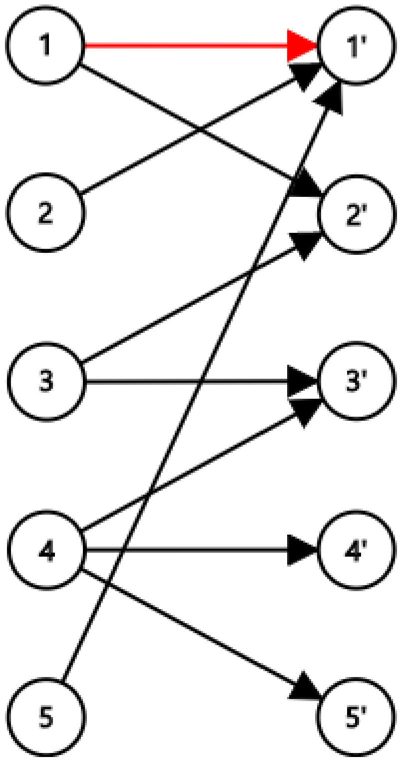
이분 매칭 (Bipartite matching)

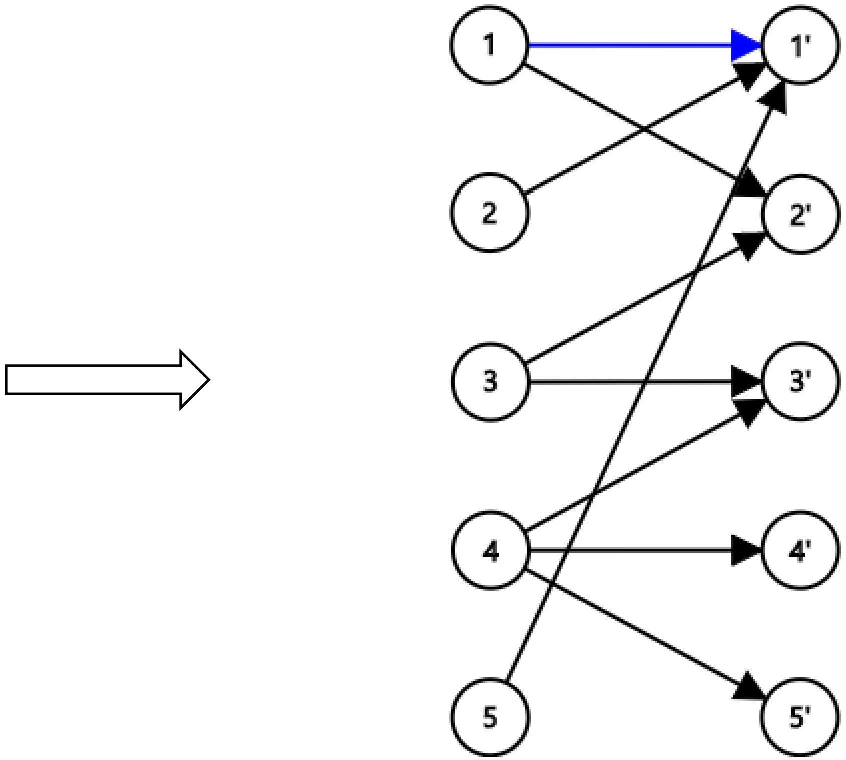
- Edmond-Karp algorithm의 시간 복잡도 = $\min\{O(|E|F), O(|V||E|^2)\}$
- 이분 매칭의 경우, 모든 용량이 1이므로 F가 최대 $O(|V|)$로 결정
- 그렇기 때문에 $O(|V||E|)$로 해결가능.
  (그냥 dfs를 이용할 생각)

이분 매칭 (Bipartite matching)

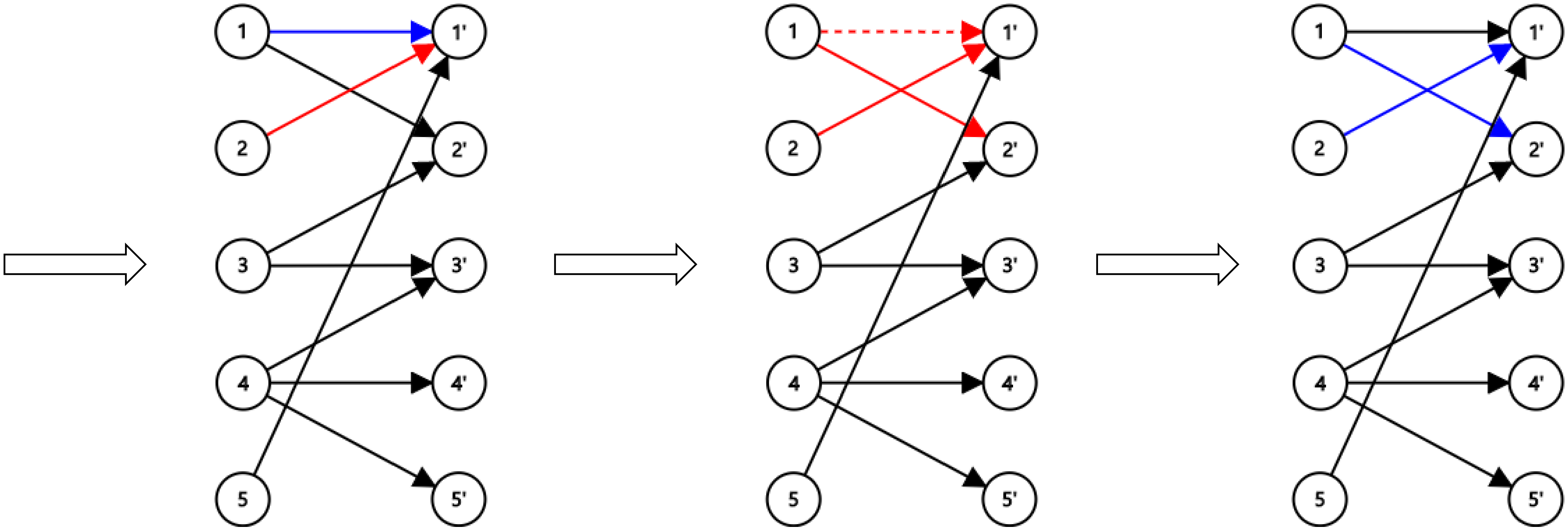


이분 매칭 (Bipartite matching)

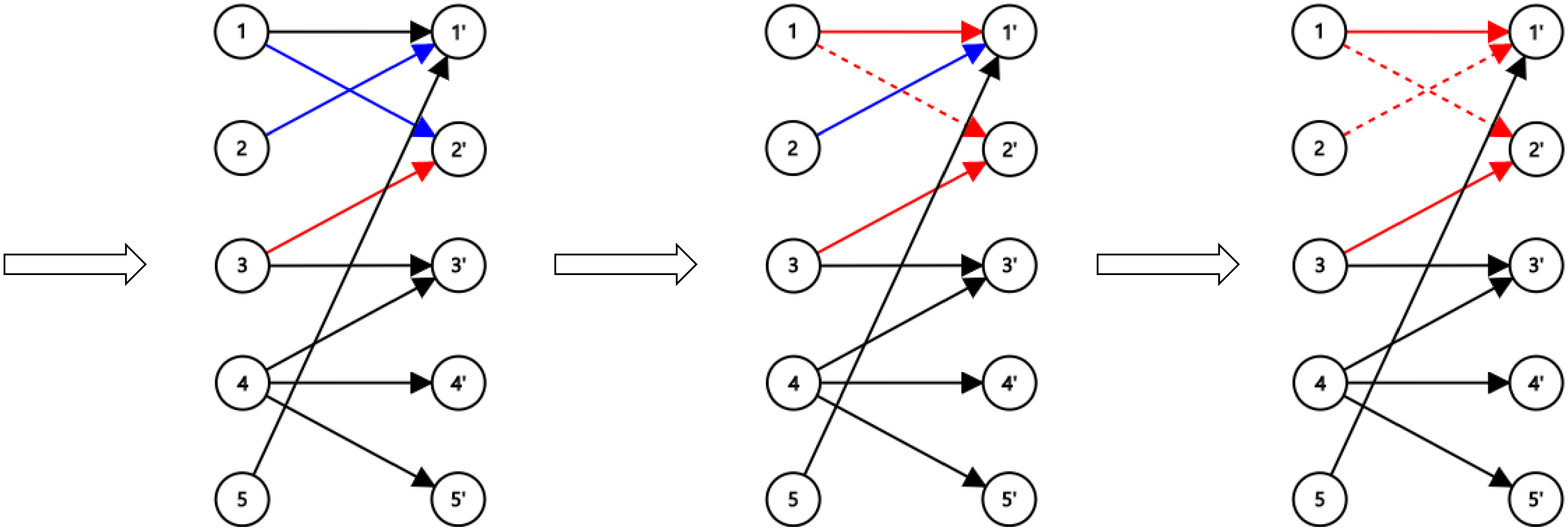




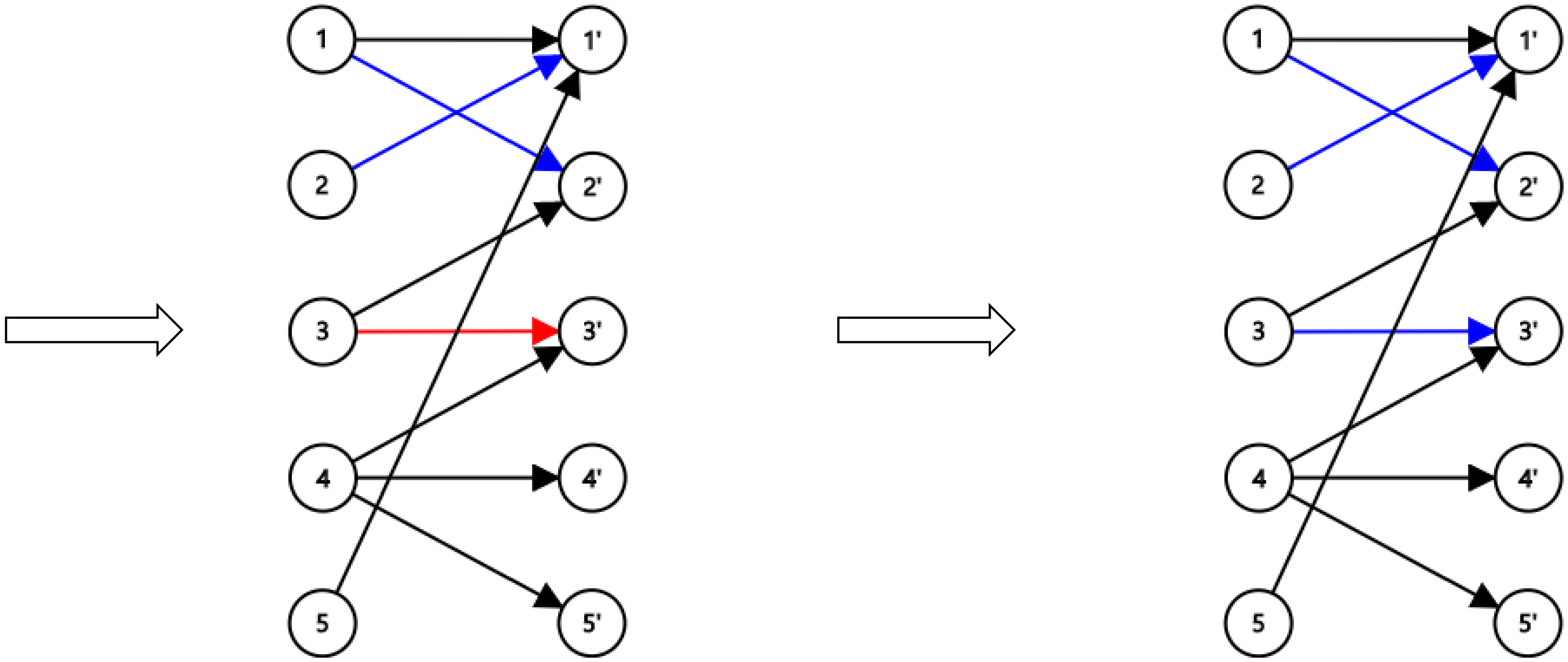
이분 매칭 (Bipartite matching)



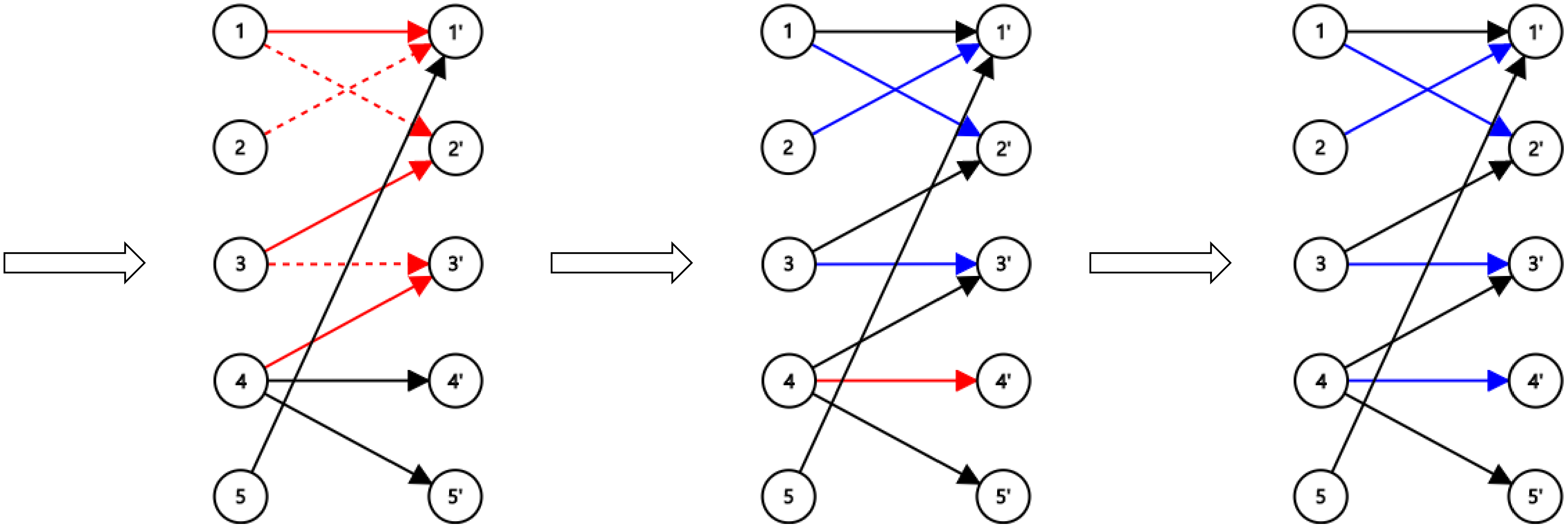
이분 매칭 (Bipartite matching)



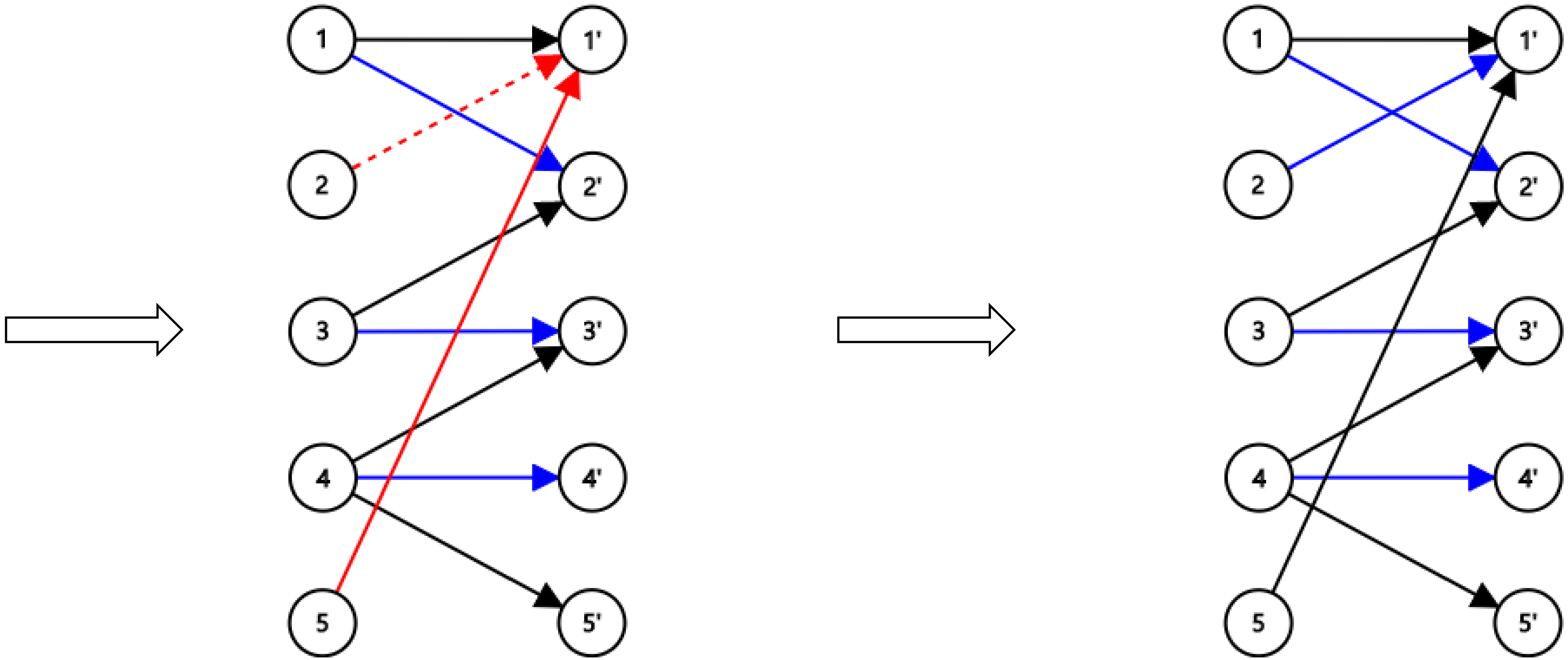
이분 매칭 (Bipartite matching)



이분 매칭 (Bipartite matching)



이분 매칭 (Bipartite matching)



이분 매칭 (Bipartite matching)

- 34 ~ 41 line
  dfs를 돌려야 하므로 인접 리스트 생성

- 46 ~ 48 line
  dfs 수행하면서 매칭 가능여부 체크

```
31	int n, m;
32	cin >> n >> m;
33
34	cow = vector<vector<int> >(n + 1);
35	for (int i = 1; i <= n; ++i) {
36		int c; cin >> c;
37		for (int j = 0; j < c; ++j) {
38			int x; cin >> x;
39			cow[i].push_back(x);
40		}
41	}
42
43	int res = 0;
44	vis = vector<bool>(n + 1);
45	from = vector<int>(m + 1, -1);
46	for (int i = 1; i <= n; ++i) {
47		fill(vis.begin(), vis.end(), false);
48		if (dfs(i, 0)) res++;
49	}
```

이분 매칭 (Bipartite matching)

vis : A그룹의 정점을 매칭했는지 여부

from : B그룹의 정점이
A그룹의 어떤 정점과 매칭됐는지

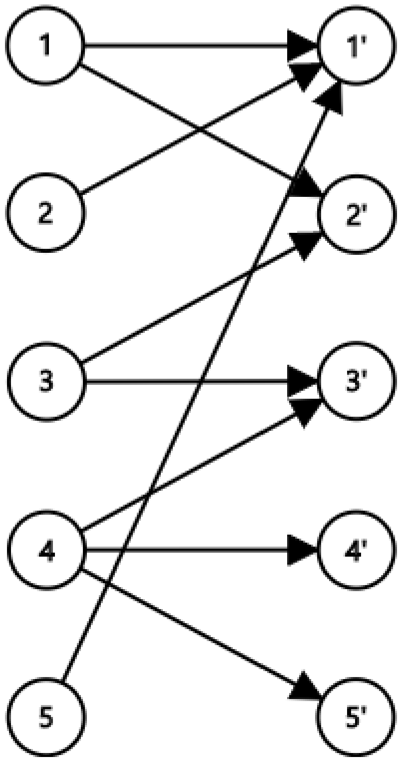
- 19 line
  from~ → next가 아직 매칭 X
  dfs~ → next에 매칭된 A그룹 정점에
  대해서 다른 점 매칭 가능여부

- 20 ~ 21 line
  새로운 점으로 매칭, true 반환

```
11      vector<bool> vis;
12      vector<int> from;
13      vector<vector<int>> cow;
14
15  bool dfs(int cur, int prev) {
16      if (vis[cur]) return false;
17      vis[cur] = true;
18      for (int next : cow[cur]) {
19          if (from[next] == -1 || dfs(from[next], next)) {
20              from[next] = cur;
21              return true;
22          }
23      }
24      return false;
25  }
```

#2188 축사 배정 4

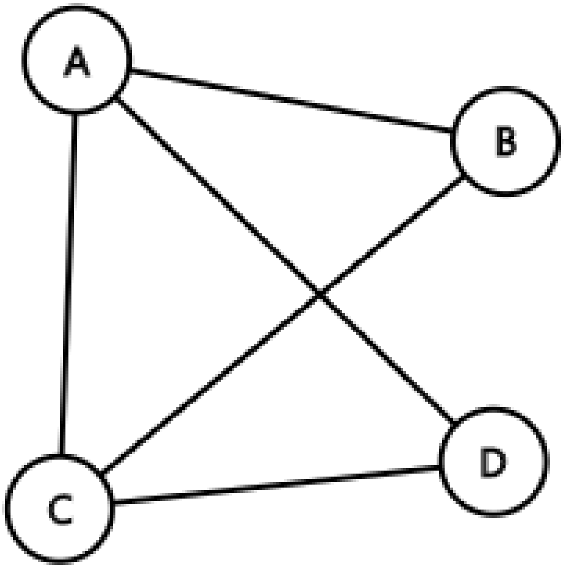
- N마리의 소와 M개의 축사가 있다.
- 각 축사엔 1마리의 소밖에 못 들어가며
- 각 소는 들어가기를 희망하는 축사가 있다.
- 각 소를 희망 축사에 넣을 때, 최대로 많이 넣을 수 있는 소는?
- $1 \leq N \leq 200, 0 \leq P \leq 200$

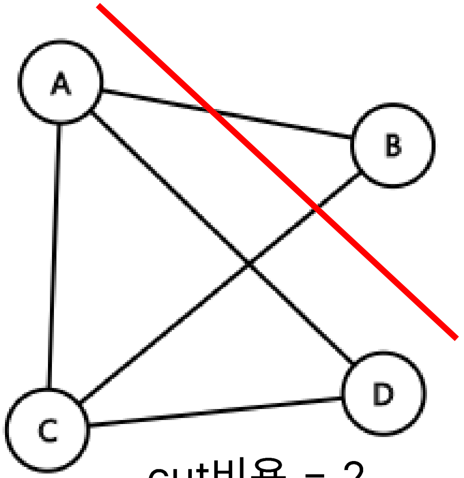


⇒ 이분 매칭 풀이 가능!

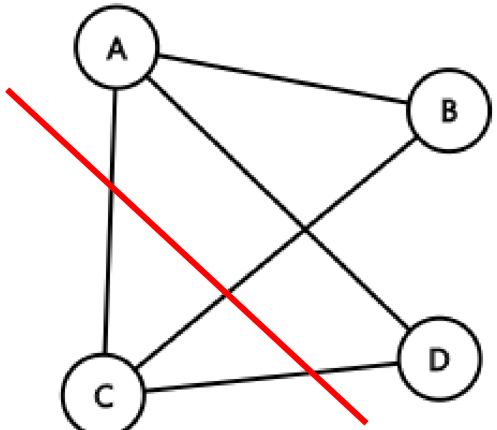
최소 컷 (Minimum cut)

- Cut : 그래프를 두 부분으로 나누는 것. 간선을 끊는다.





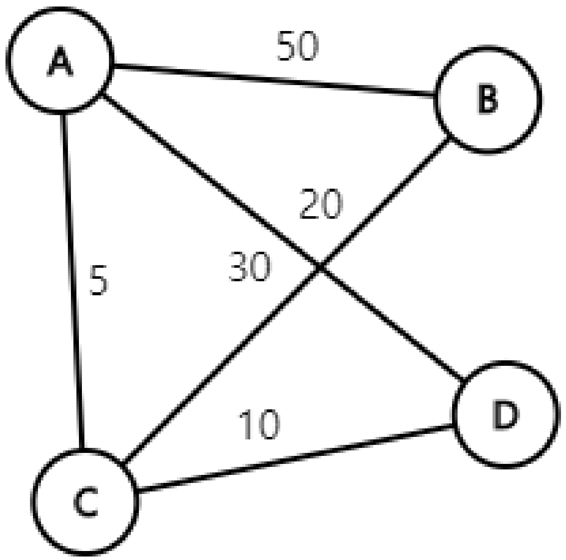
cut비용 = 2

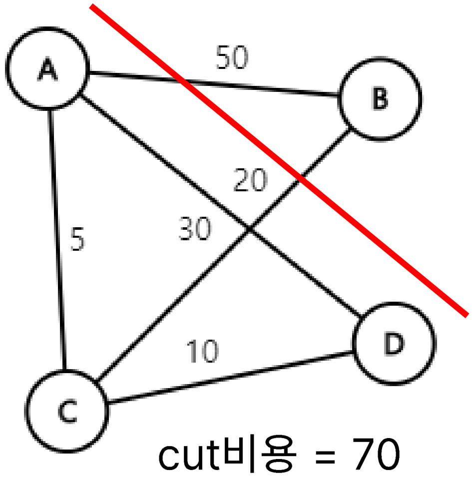


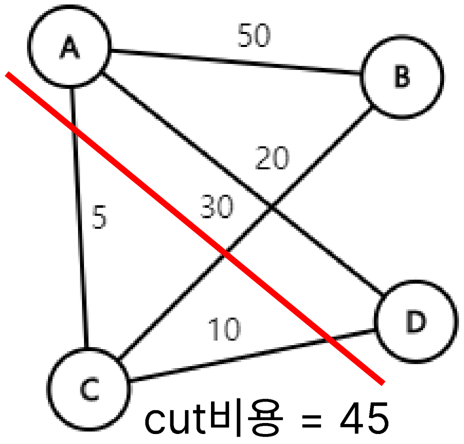
cut비용 = 3

최소 컷 (Minimum cut)

- Cut : 그래프를 두 부분으로 나누는 것. 간선을 끊는다.

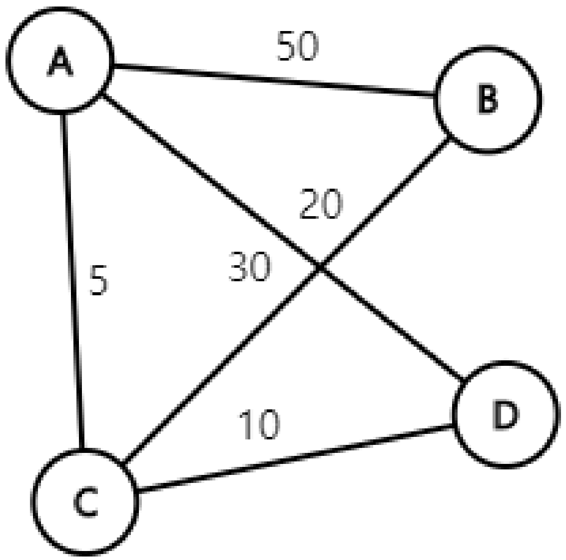


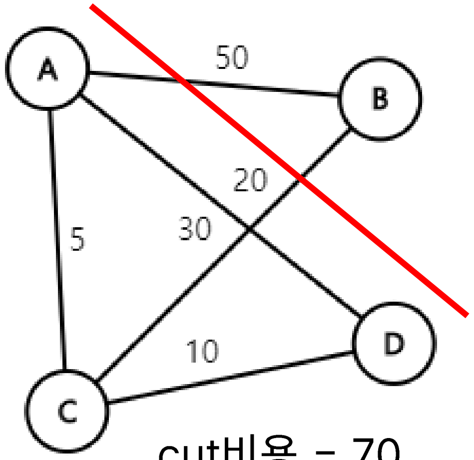




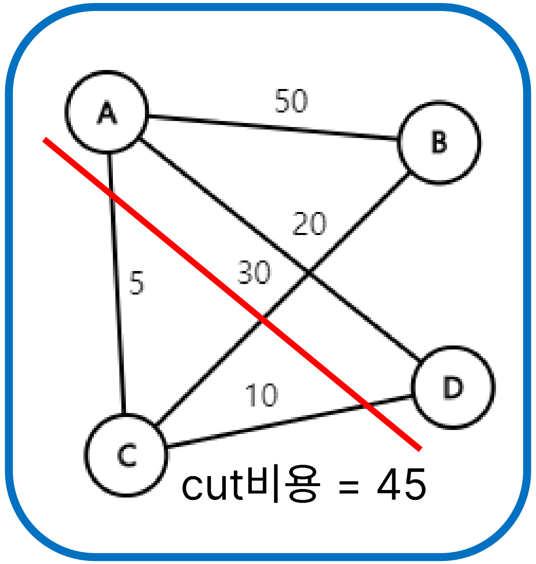
최소 컷 (Minimum cut)

- Minimum cut : 그래프를 두 부분으로 나눌 때, 가장 적은 비용이 드는 cut





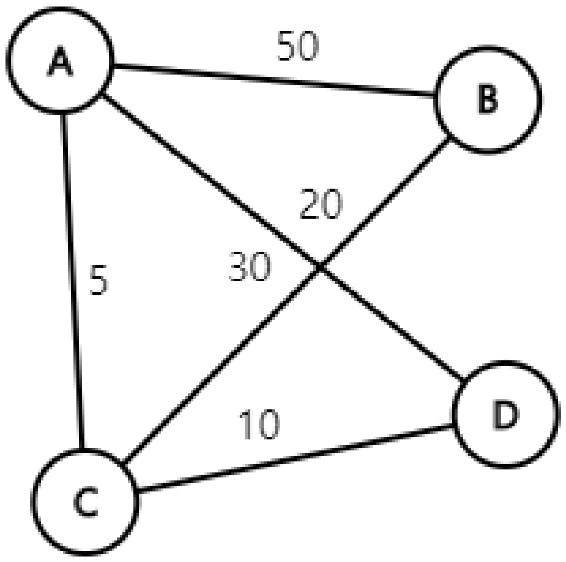
cut비용 = 70

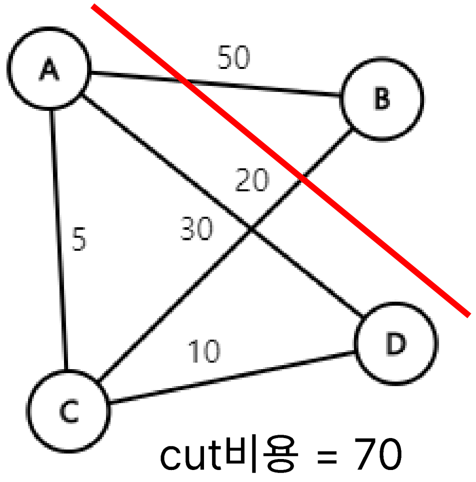


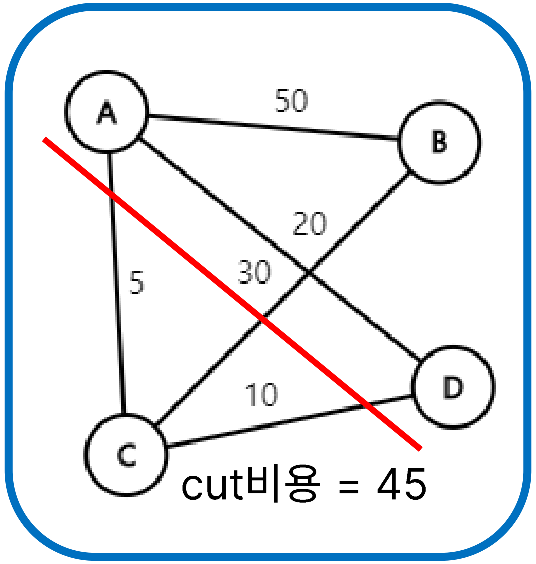
최대 유량 최소 컷 정리

- 결론만 말해서 Max flow 와 Min cut 은 “동치”이다.

  (증명은 링크 확인하자) (링크1) (링크2-증명2)

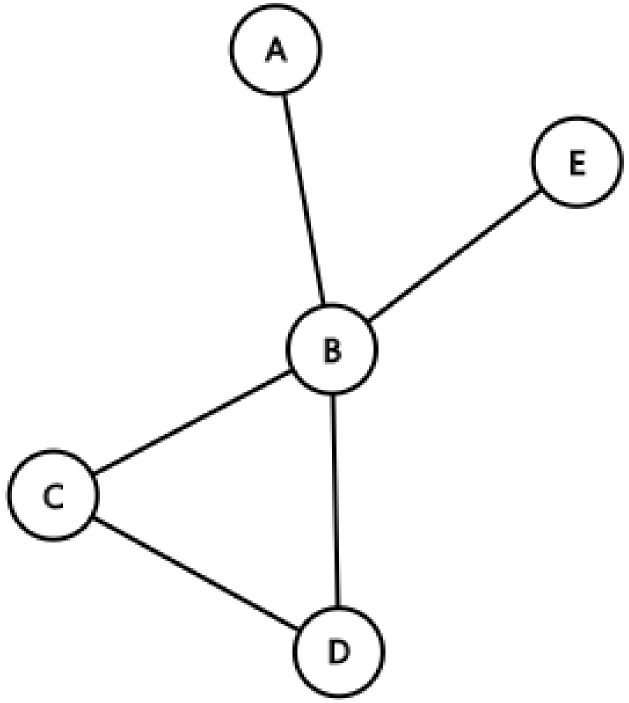


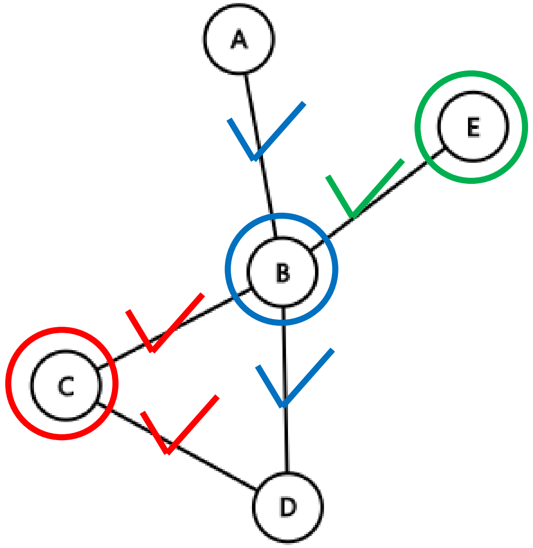


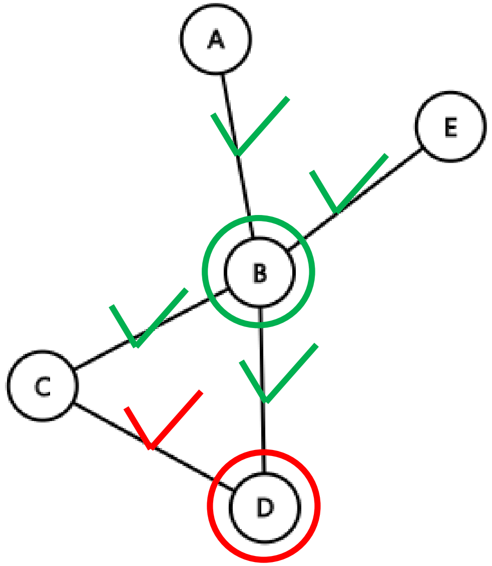


최소 버텍스 커버 (Minimum Vertex Cover)

- Vertex Cover : 정점 집합 V의 부분 집합으로 모든 간선의 양 쪽 끝점 중 하나는 Vertex cover에 속한다.

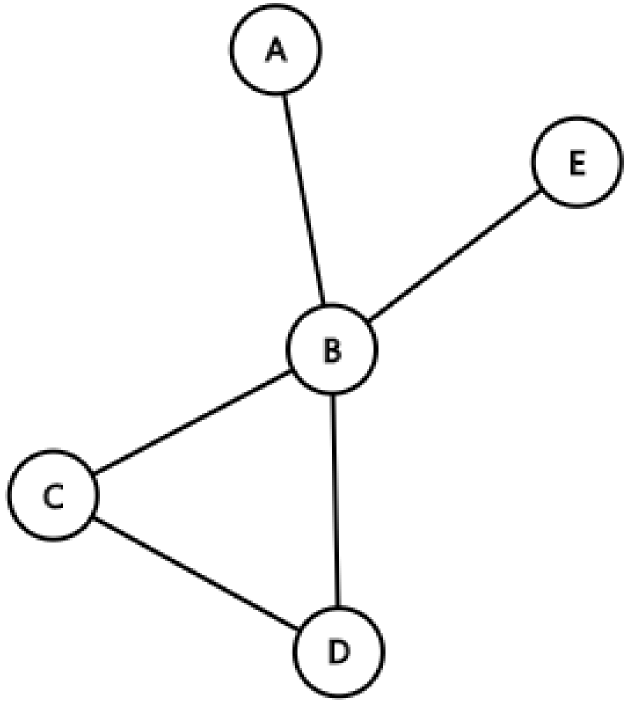


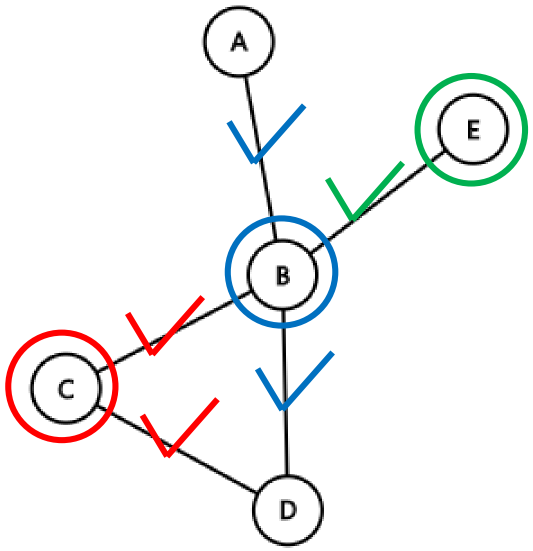


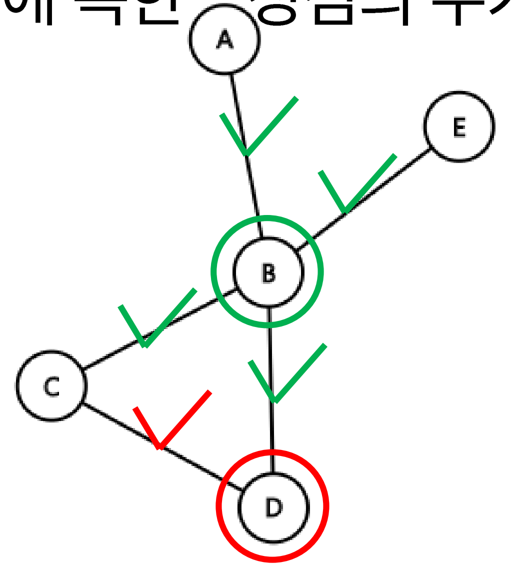


최소 버텍스 커버 (Minimum Vertex Cover)

- Minimum Vertex Cover : 정점 집합 V의 부분 집합으로 모든 간선의 최소한 한쪽 끝점은 Vertex cover에 속한다. → vertex cover에 속한 정점의 수가 최소

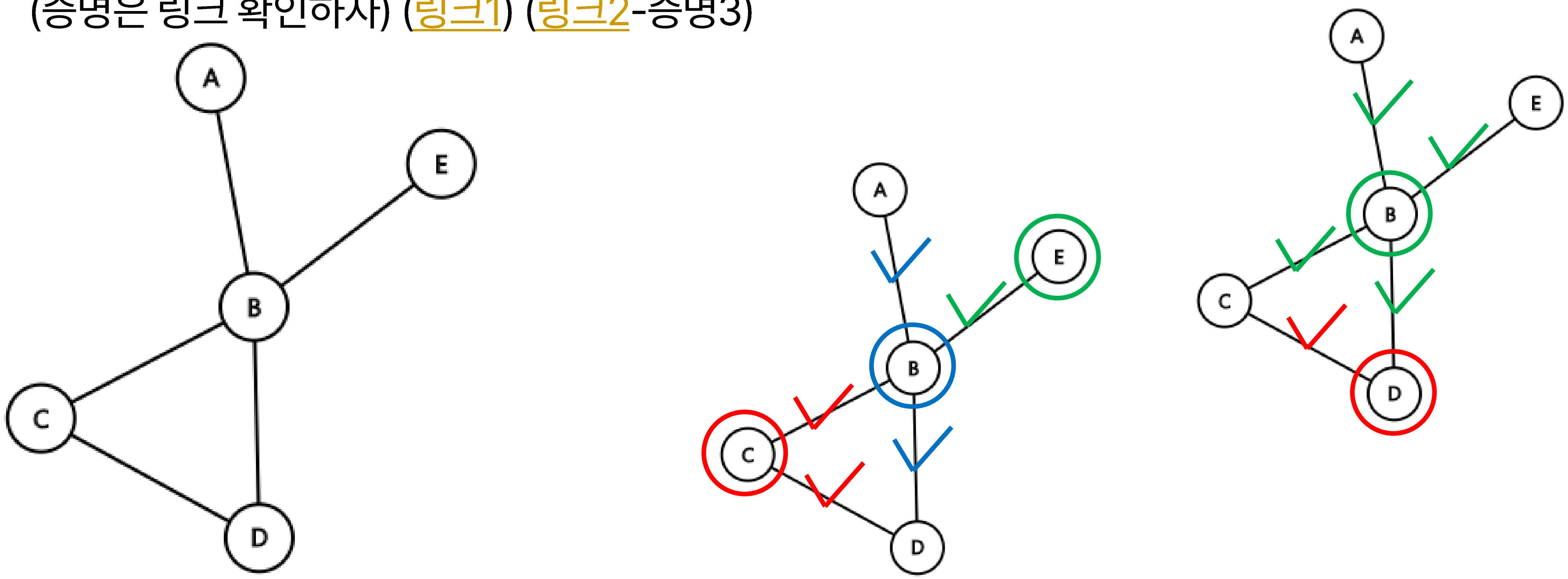






쾨닉의 정리

- 이번에도 결론만 말해서 이분그래프에서의 최소 버텍스 커버는 최대 유량과 동치이다.
  (증명은 링크 확인하자) (링크1) (링크2-증명3)



Reference

koosaga → 유량 관련 알고리즘 정리 (링크)
→ 유량 관련 알고리즘 증명 (링크)

kks227 → 네트워크 플로우, 이분탐색, 최소 컷 등등 (링크)

crocus → 네트워크 플로우 (링크) , 이분매칭(링크) , 이분매칭 시간 최적화 (링크)

더 공부하기

→ MCMF - $O(|V||E|f)$, Dinic's algorithm - $O(|V|^2|E|)$,
Hopcroft-Karp algorithm in bipartite graph - $O\left(|E|\sqrt{|V|}\right) \sim O(|V|^{2.5})$

[네트워크 플로우]

- 4 6086 최대 유량
- 5 17412 도시 왕복하기
- 3 2316 도시 왕복하기2
- 2 7616 교실로 가는 길
- 2 1658 돼지 잡기

[이분 매칭]

- 4 2188 축사 배정
- 4 11375 열혈강호
- 4 11376 열혈강호2
- 3 1017 소수 쌍
- 2 3295 단방향 링크 네트워크

[minimum cut]

- 3 11014 컨닝 2

[minimum vertex cover]

- 3 1867 돌맹이 제거