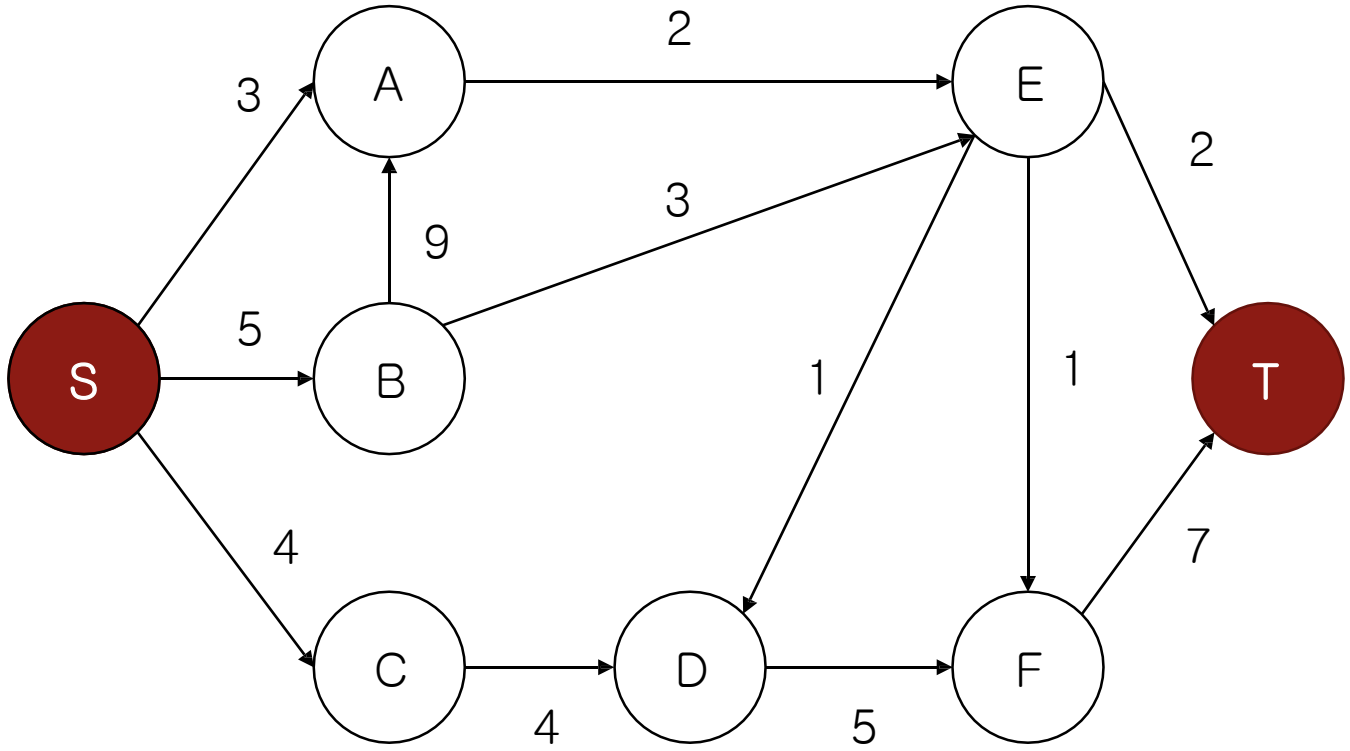
04. Network Flow

2019 Summer / 20141574 임지환

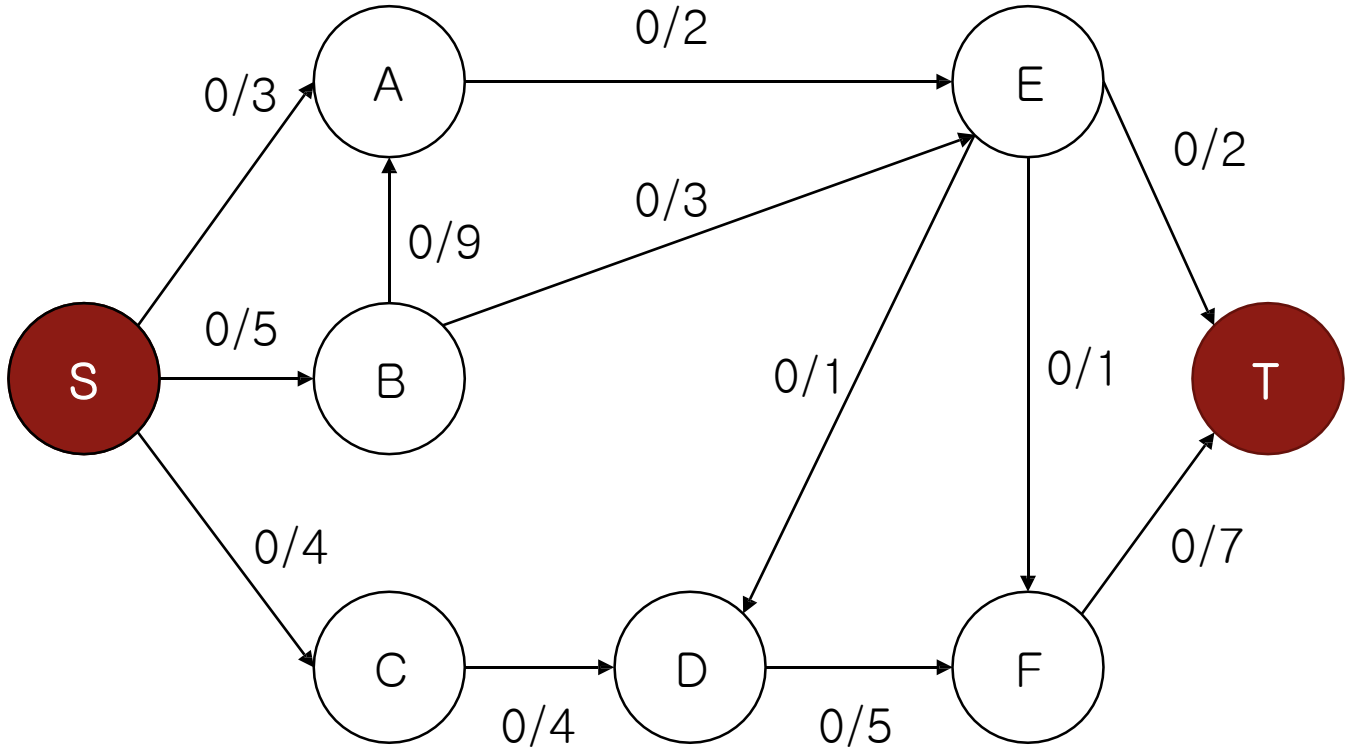
Network Flow

Ex) S->T로 이동할 때 한번에 최대 얼마나 많은 차량이 이동할 수 있는가



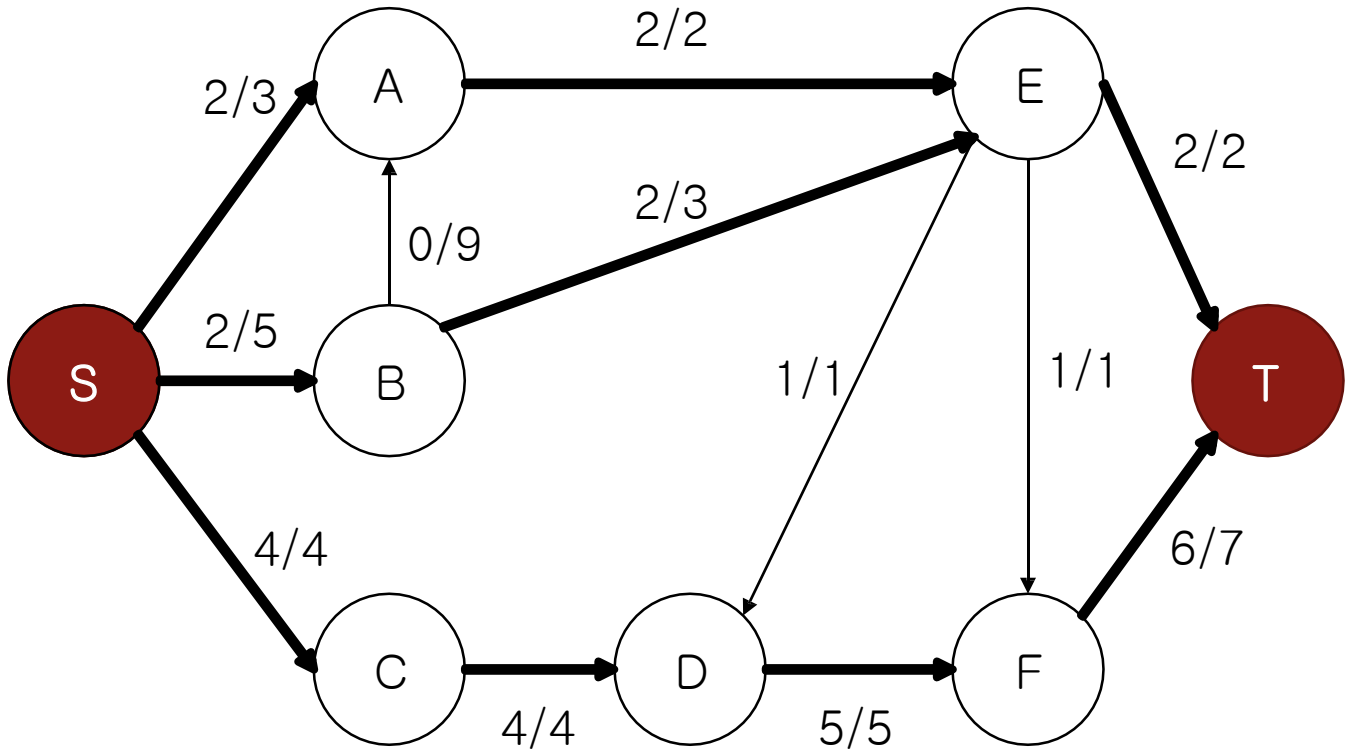
Network Flow

Ex) S->T로 이동할 때 한번에 최대 얼마나 많은 차량이 이동할 수 있는가

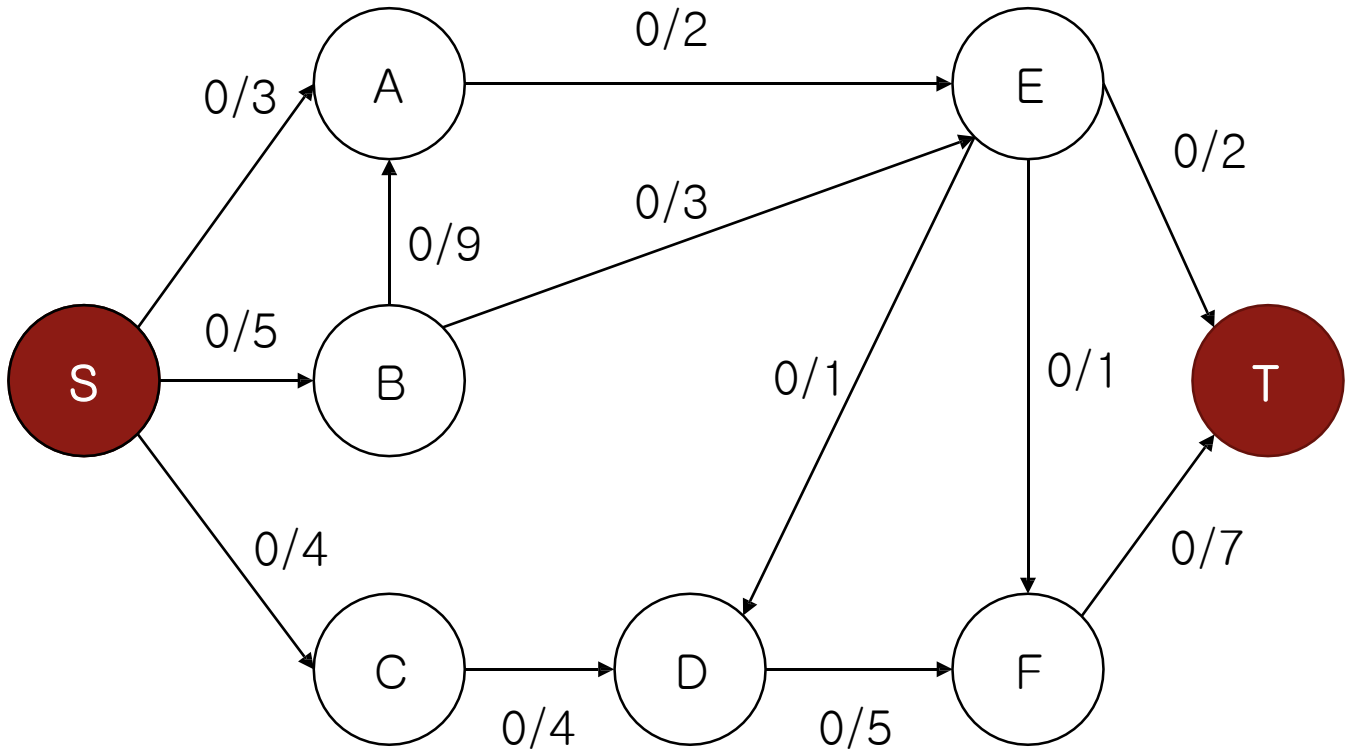


Network Flow

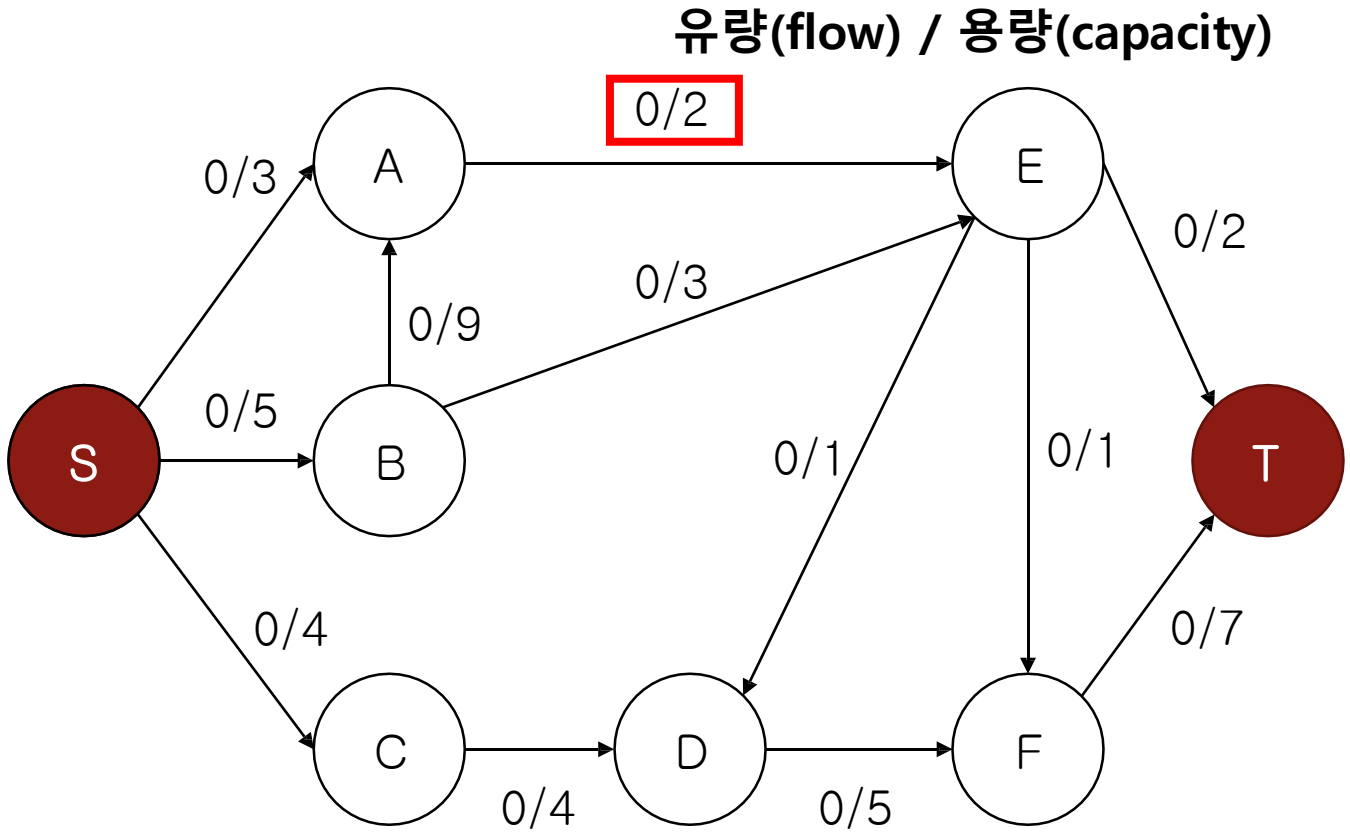
Ex) S->T로 이동할 때 한번에 최대 얼마나 많은 차량이 이동할 수 있는가



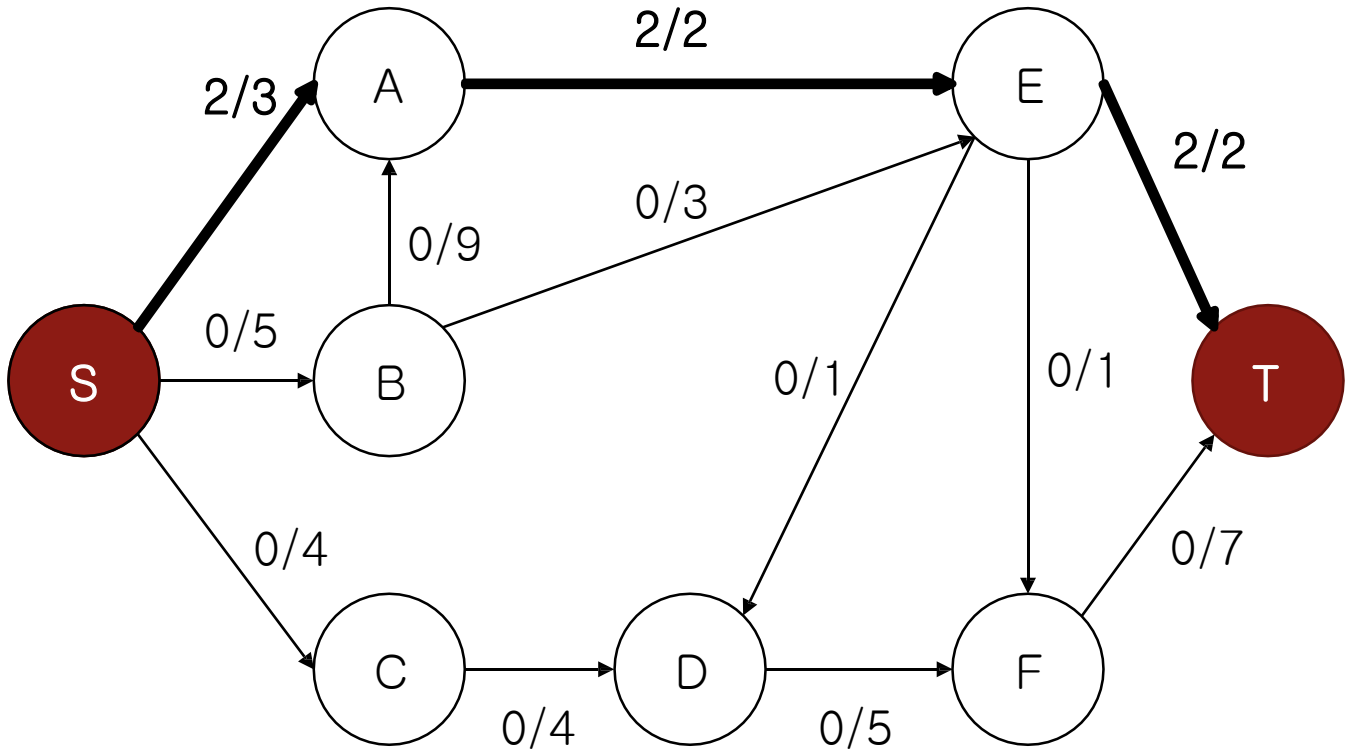
Network Flow



Network Flow



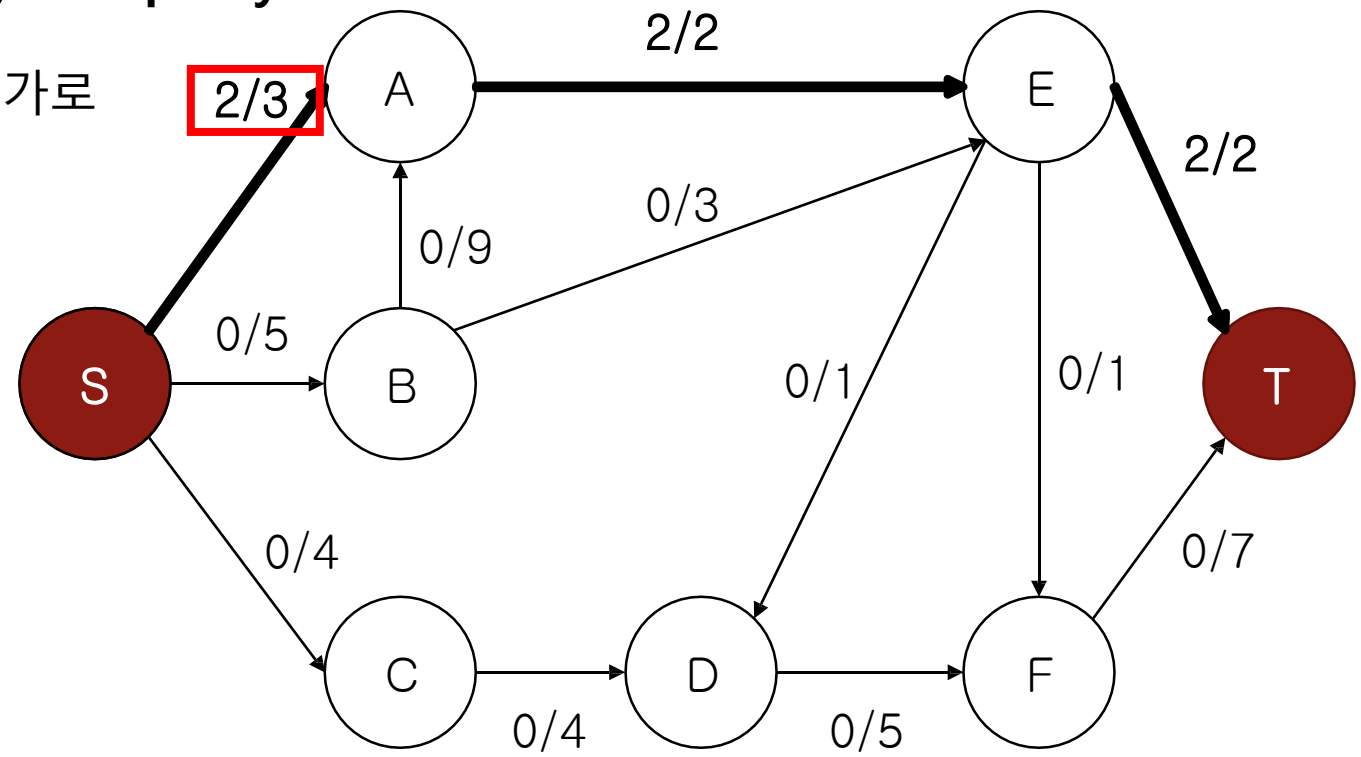
Network Flow



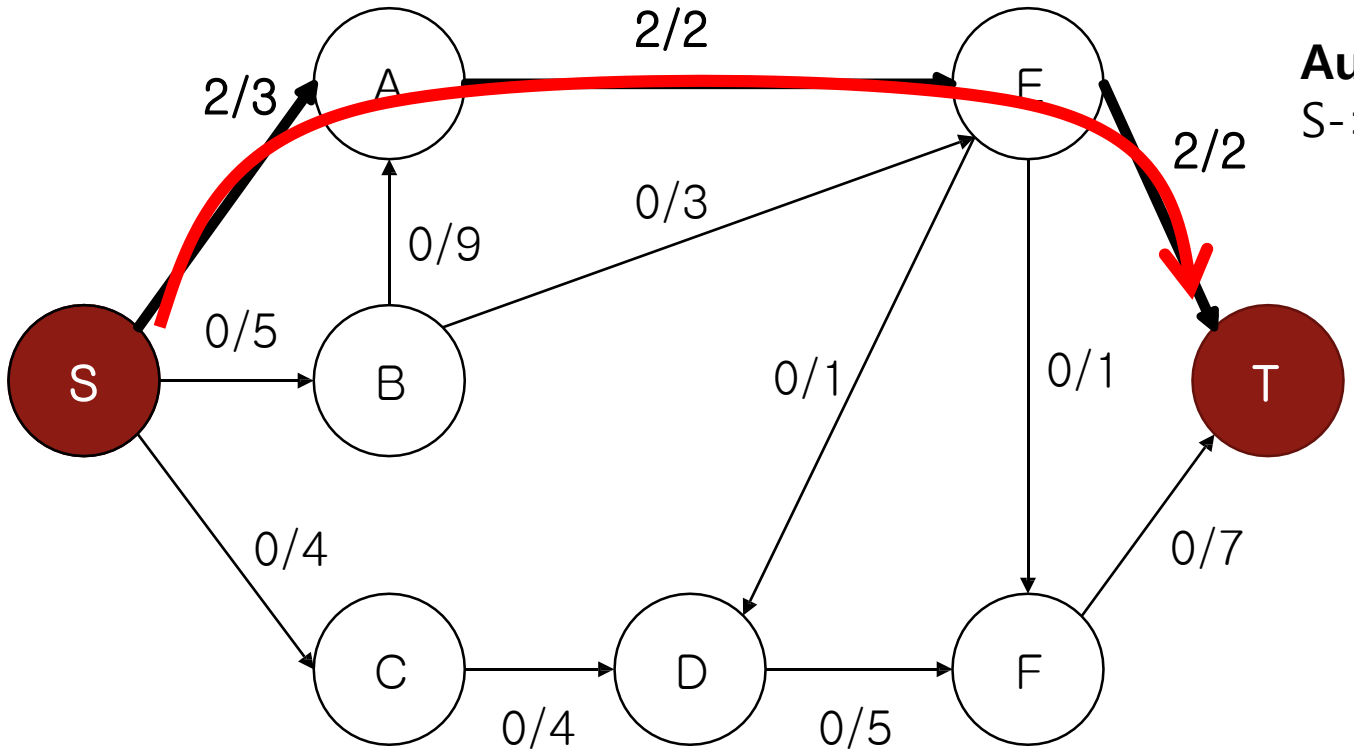
Network Flow

residual capacity := capacity – flow

현재 상태에서 추가로
흐를 수 있는 양



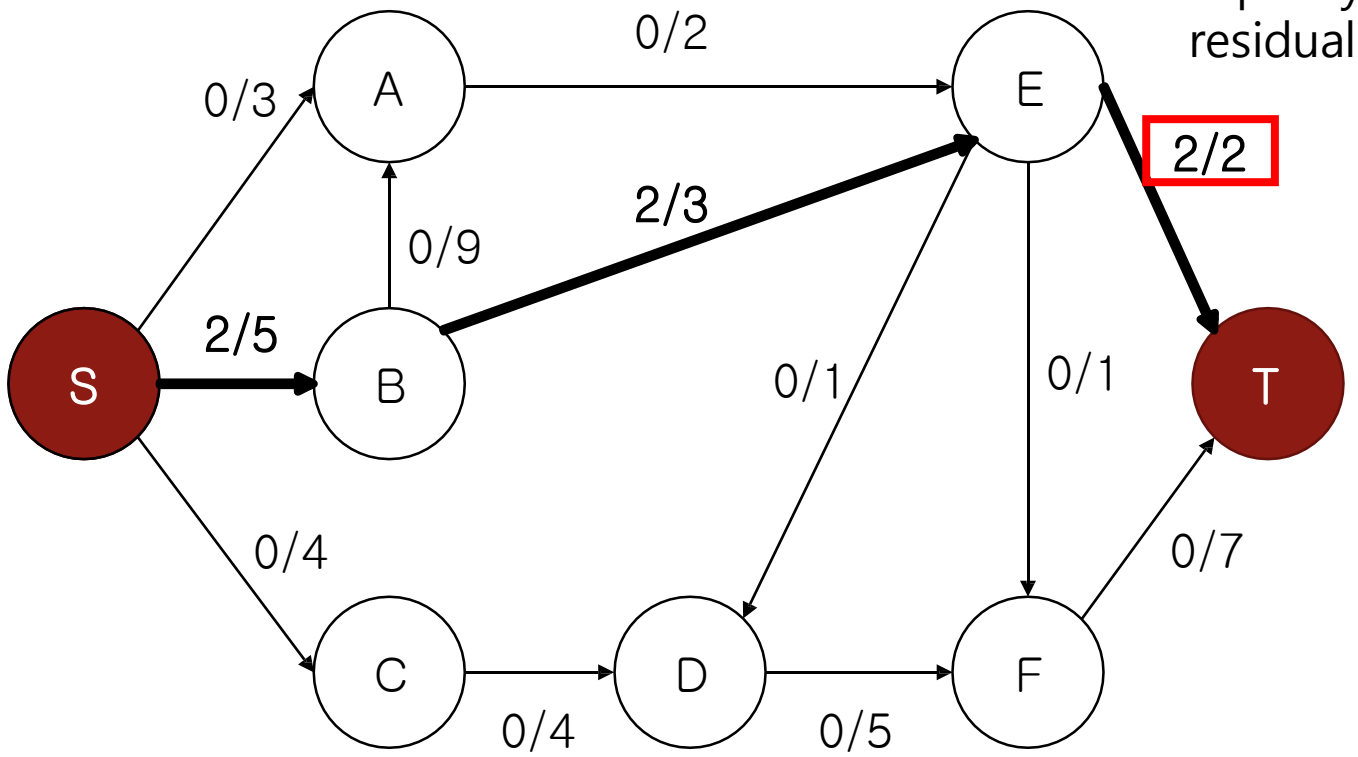
Network Flow



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

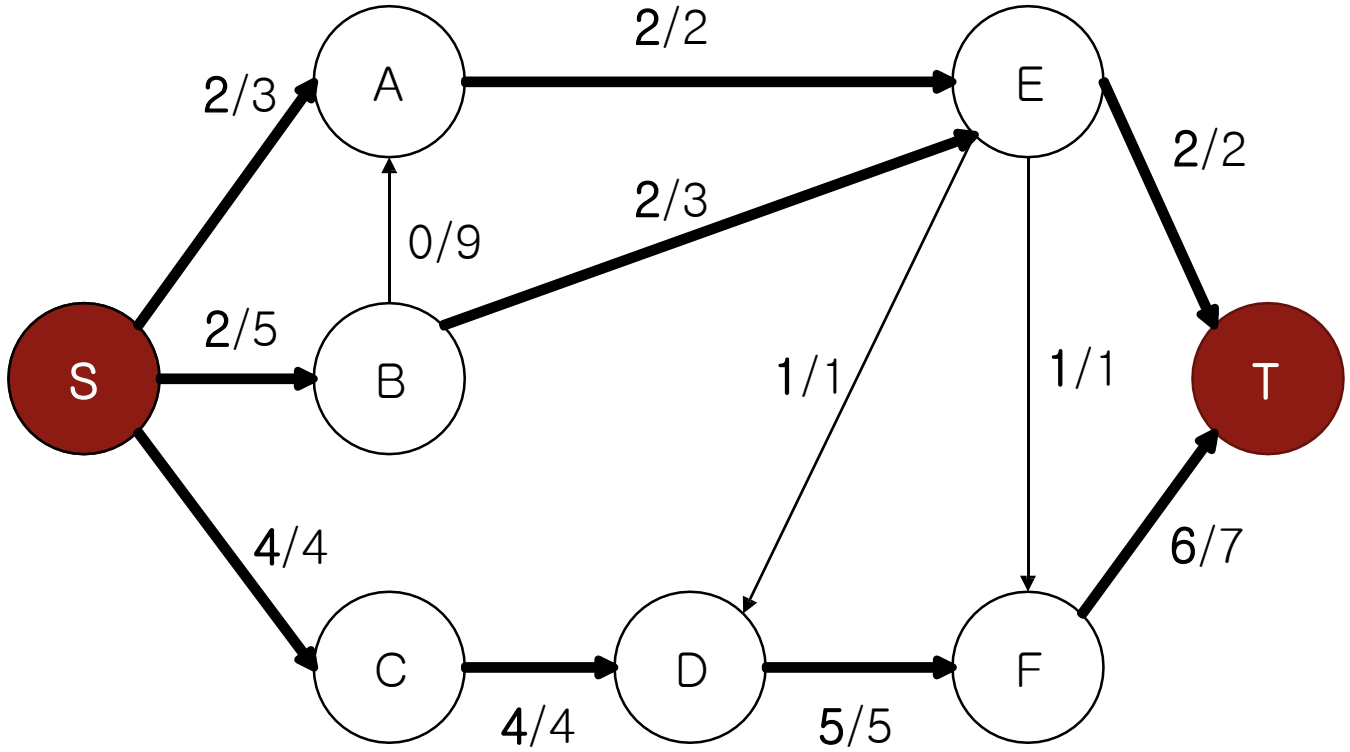
Network Flow

Blocking Flow
증가경로에서 residual capacity가 가장 낮은 edge의 residual capacity



Network Flow

Total Flow : 8 => Maximum flow



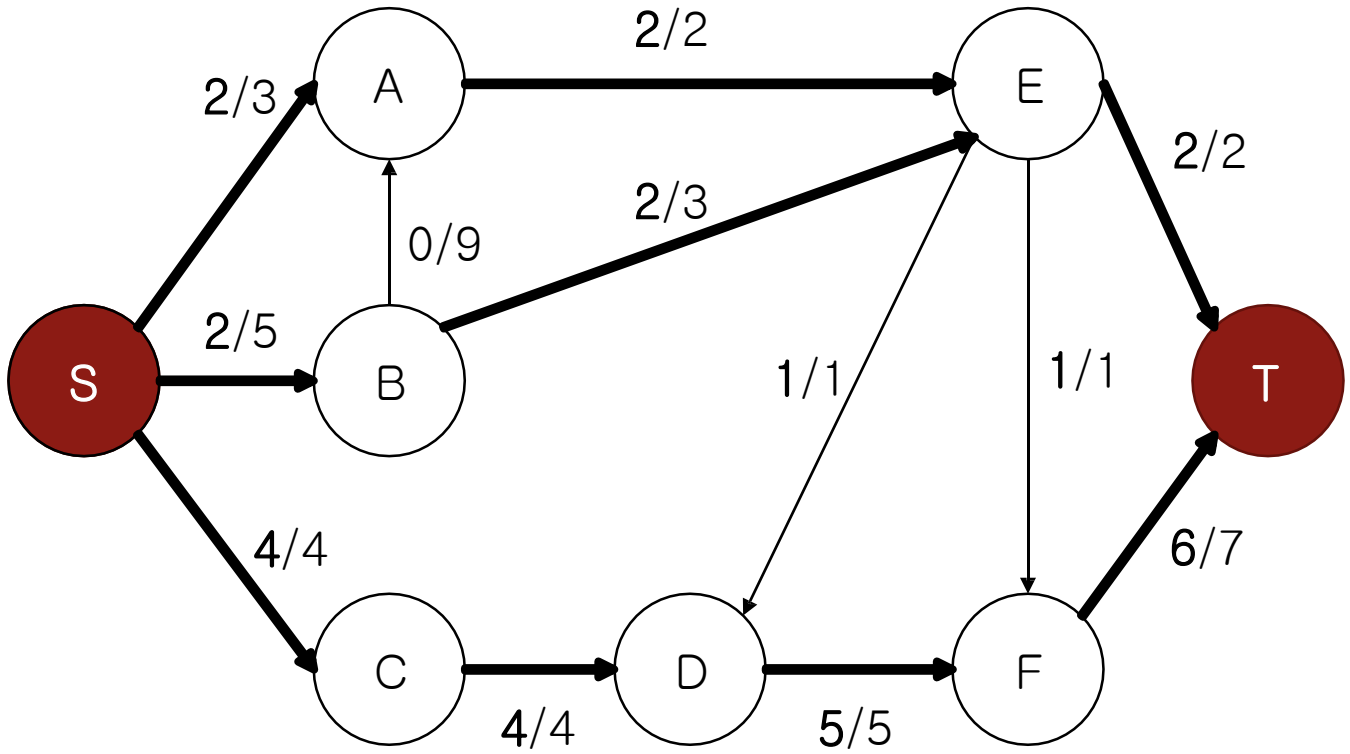
Properties of Network Flow

1. 유량 보존

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

들어오는 유량의 총합과 나가는 유량의 총 합이 같다.

Network Flow

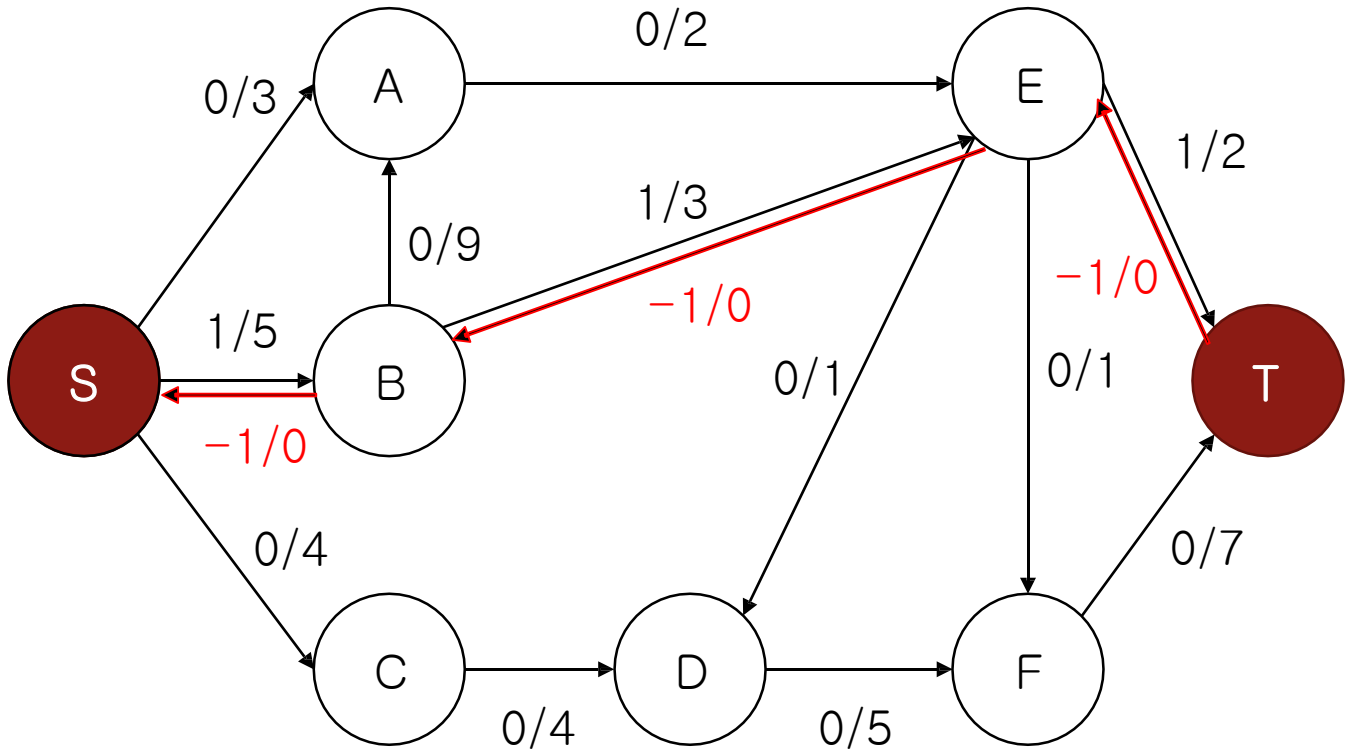


Properties of Network Flow

2. 유량의 대칭성

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

Network Flow



Properties of Network Flow

3. 용량 제한 속성

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

Network Flow Algorithm의 목적?

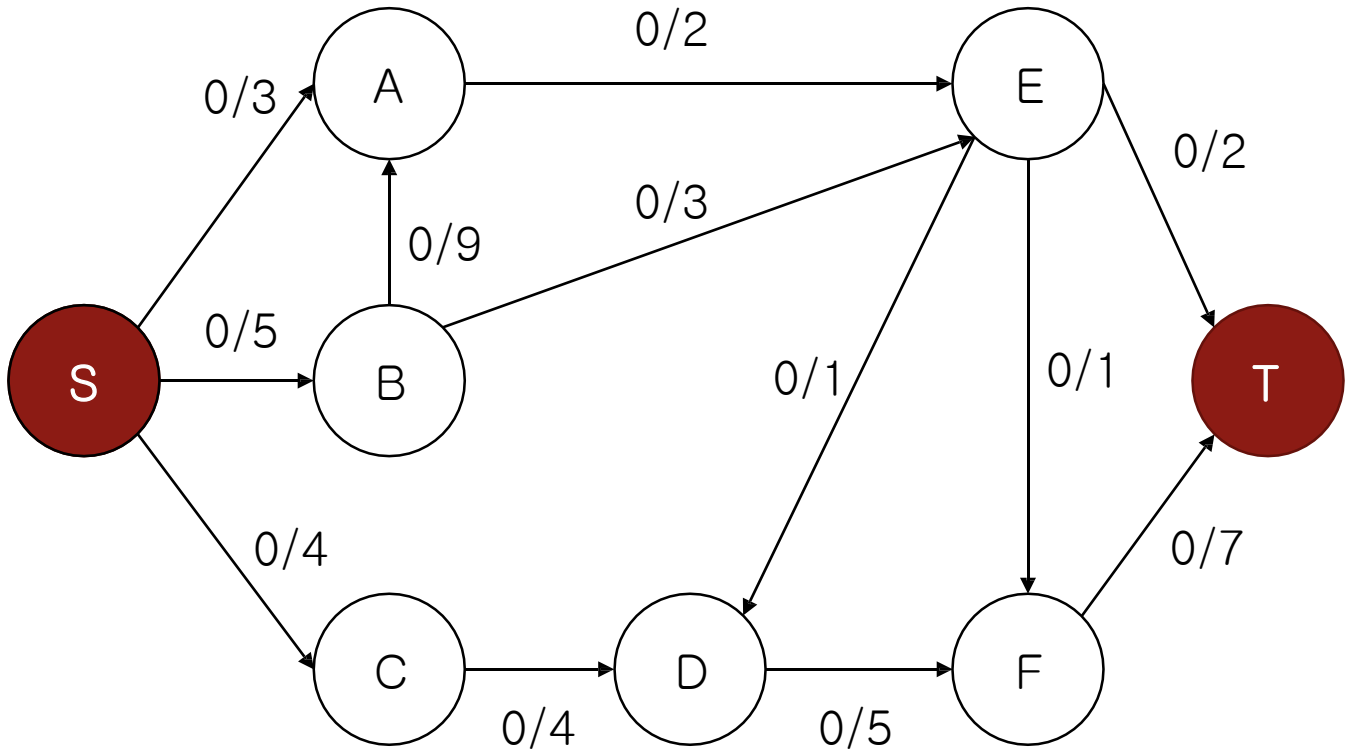
주어진 시작지점(Source)과 끝지점(Sink)가 있을 때

흘려보낼 수 있는 최대 유량

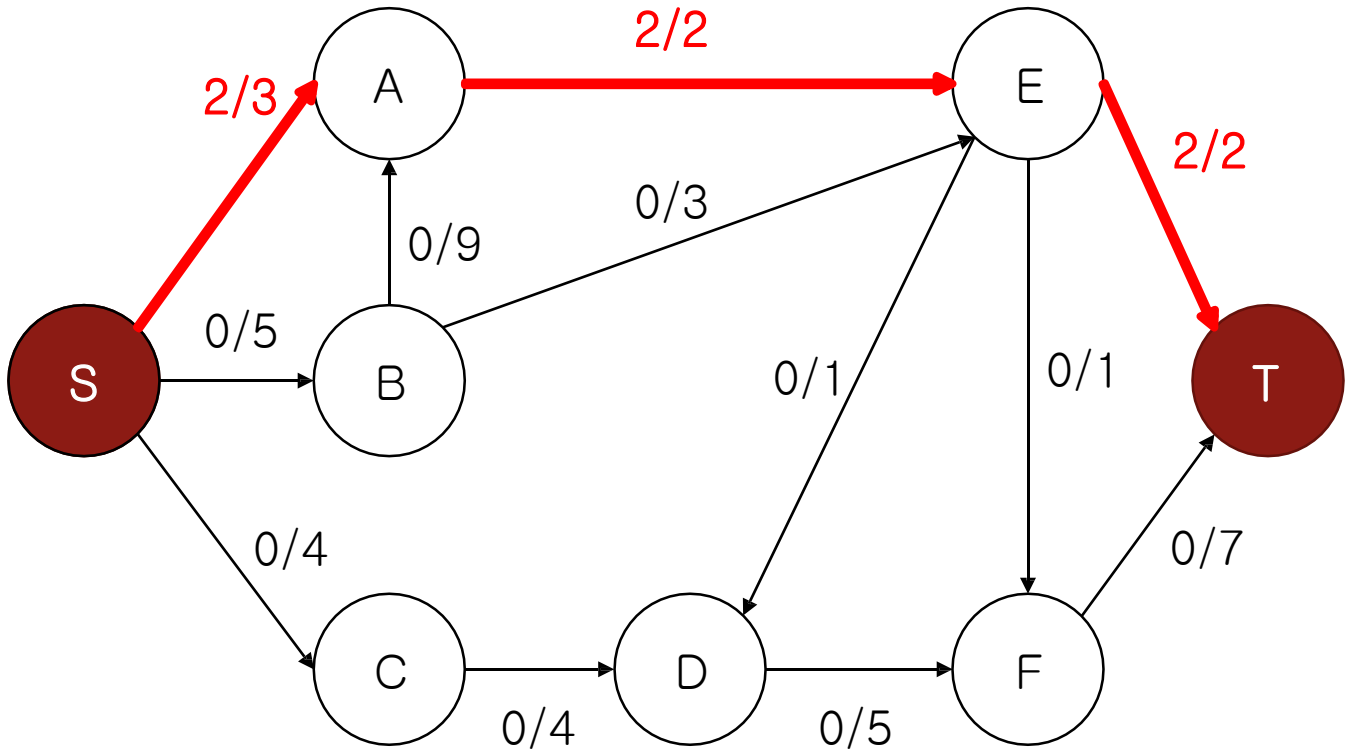
Ford-Fulkerson algorithm

1. DFS를 통해 Augmenting Path 찾기
2. 찾으면 해당 경로로 blocking flow만큼의 유량 흘리기
3. Augmenting Path가 없을 때까지 1,2를 반복.

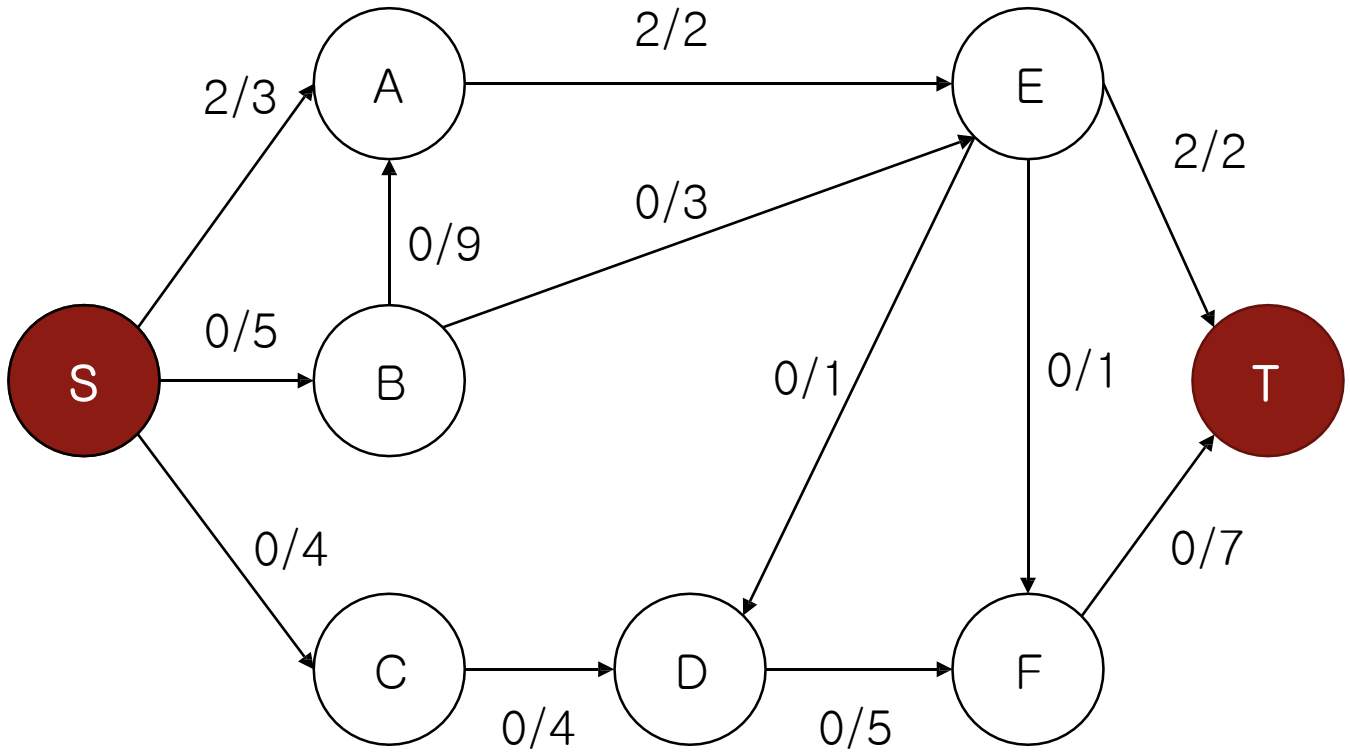
Ford-Fulkerson algorithm



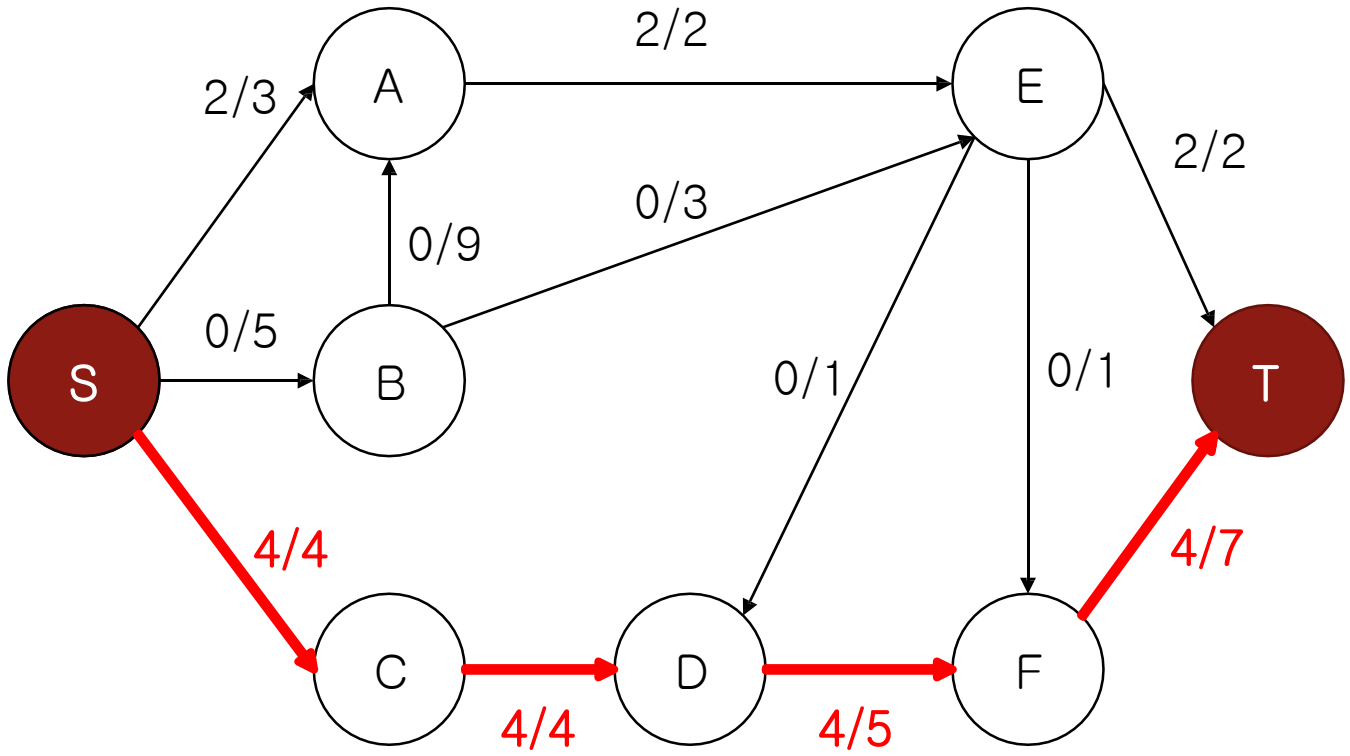
Ford-Fulkerson algorithm



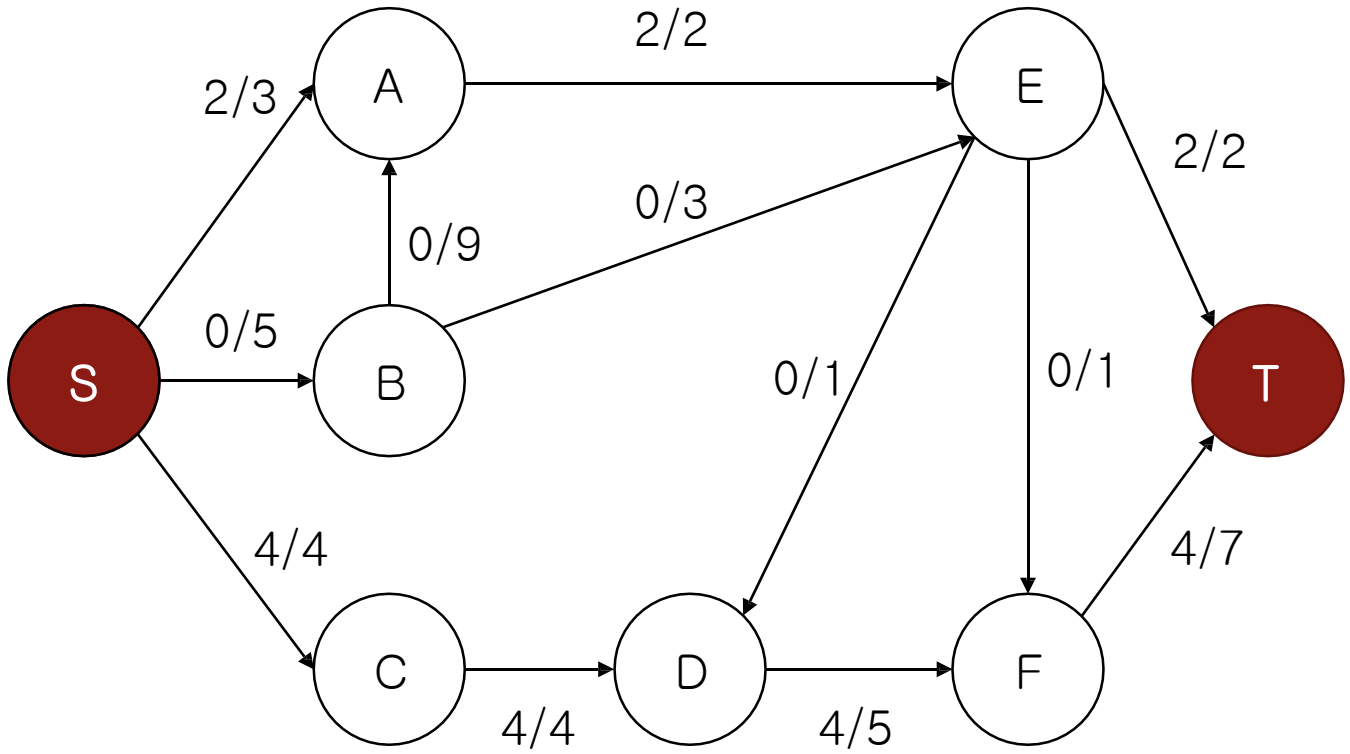
Ford-Fulkerson algorithm



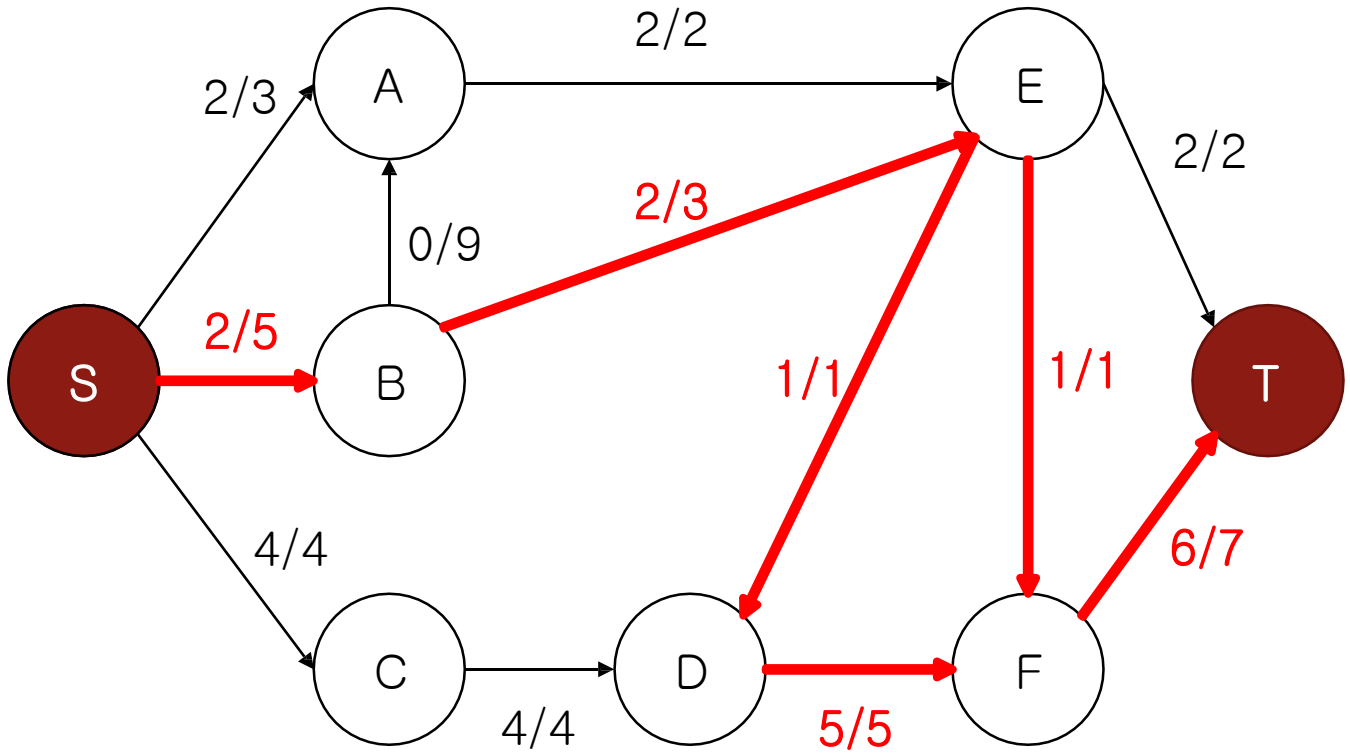
Ford-Fulkerson algorithm



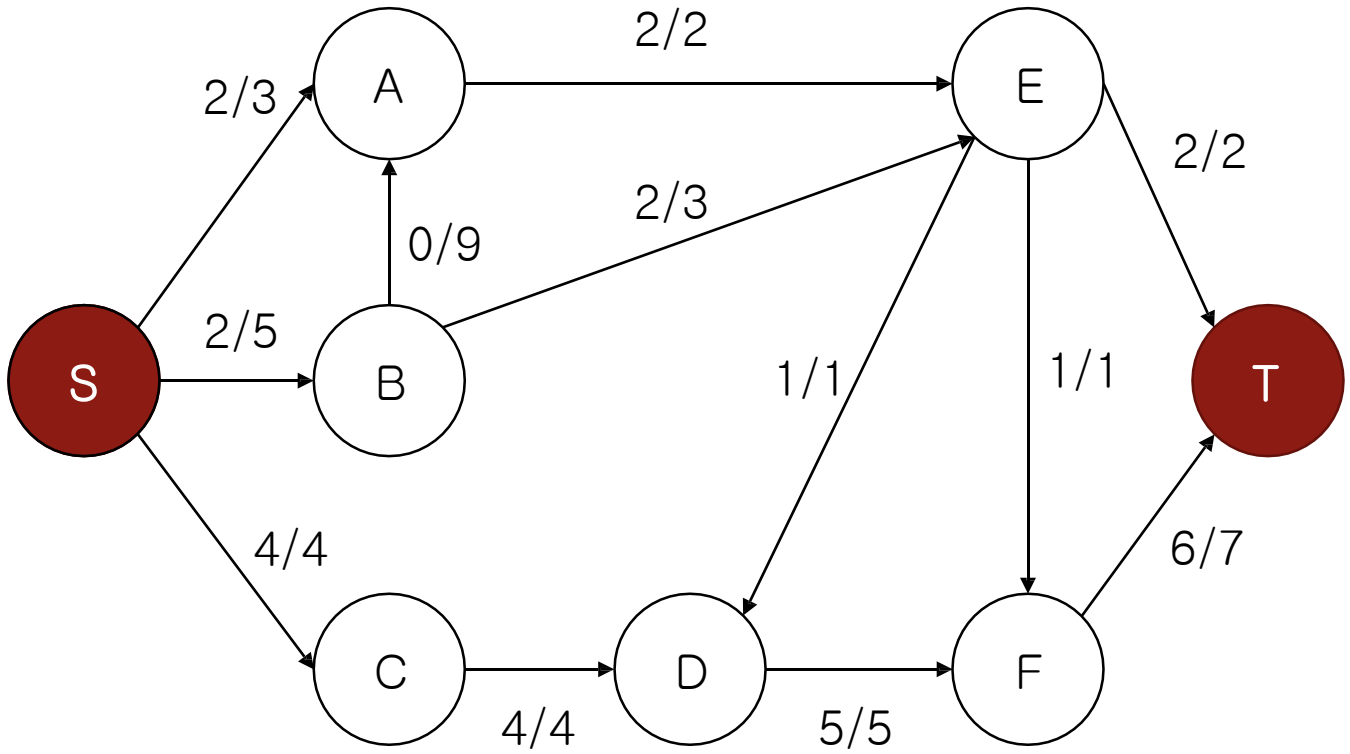
Ford-Fulkerson algorithm



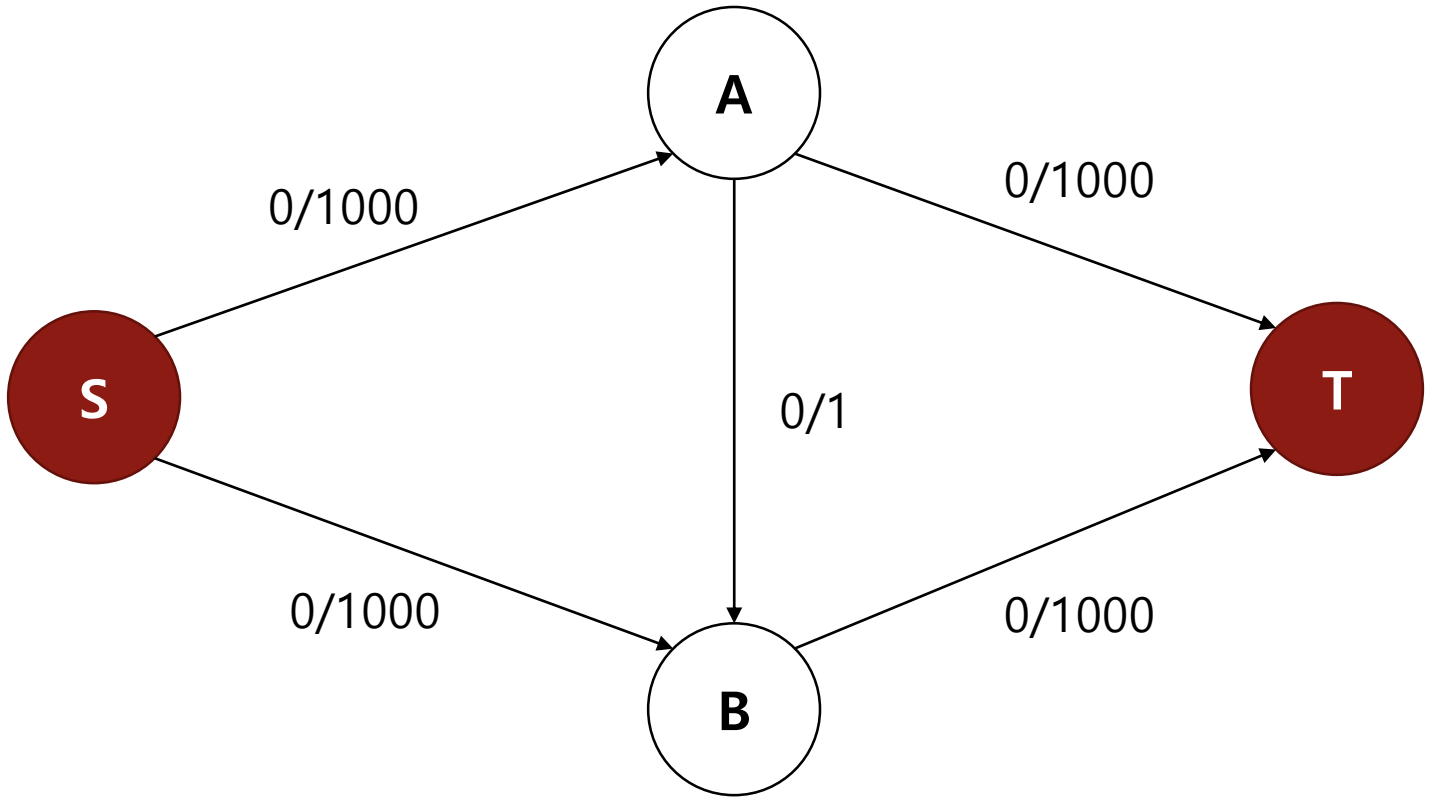
Ford-Fulkerson algorithm



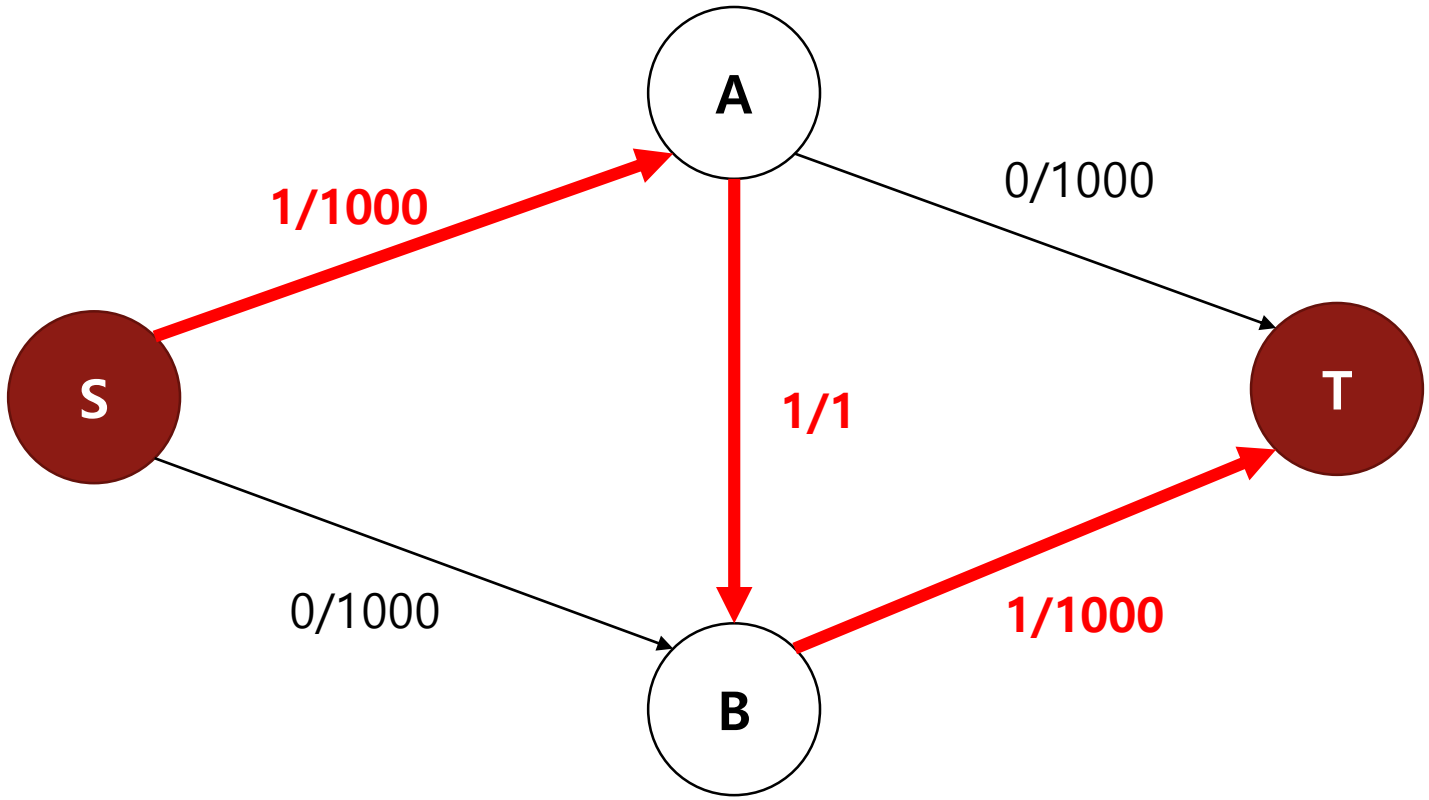
Ford-Fulkerson algorithm



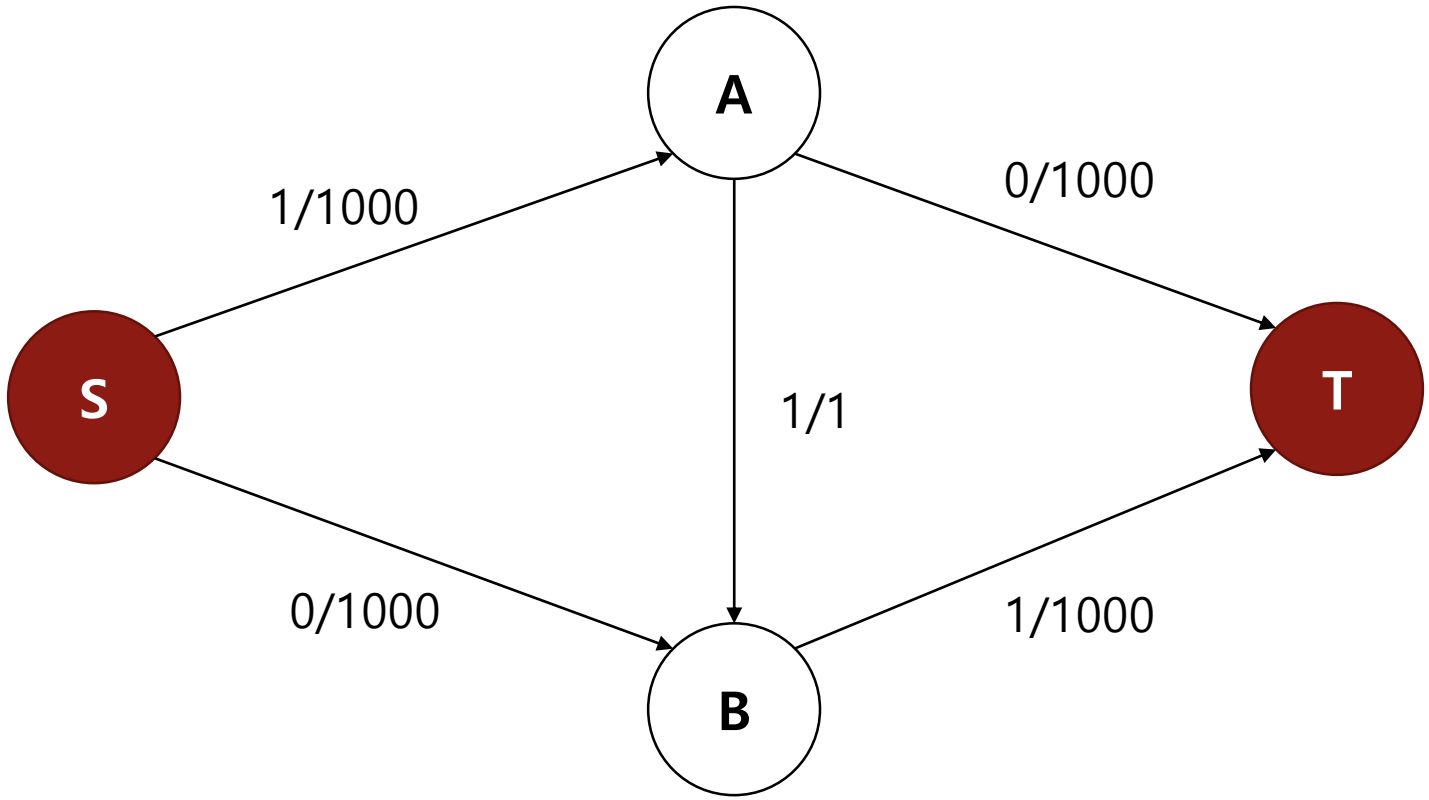
Ford-Fulkerson algorithm



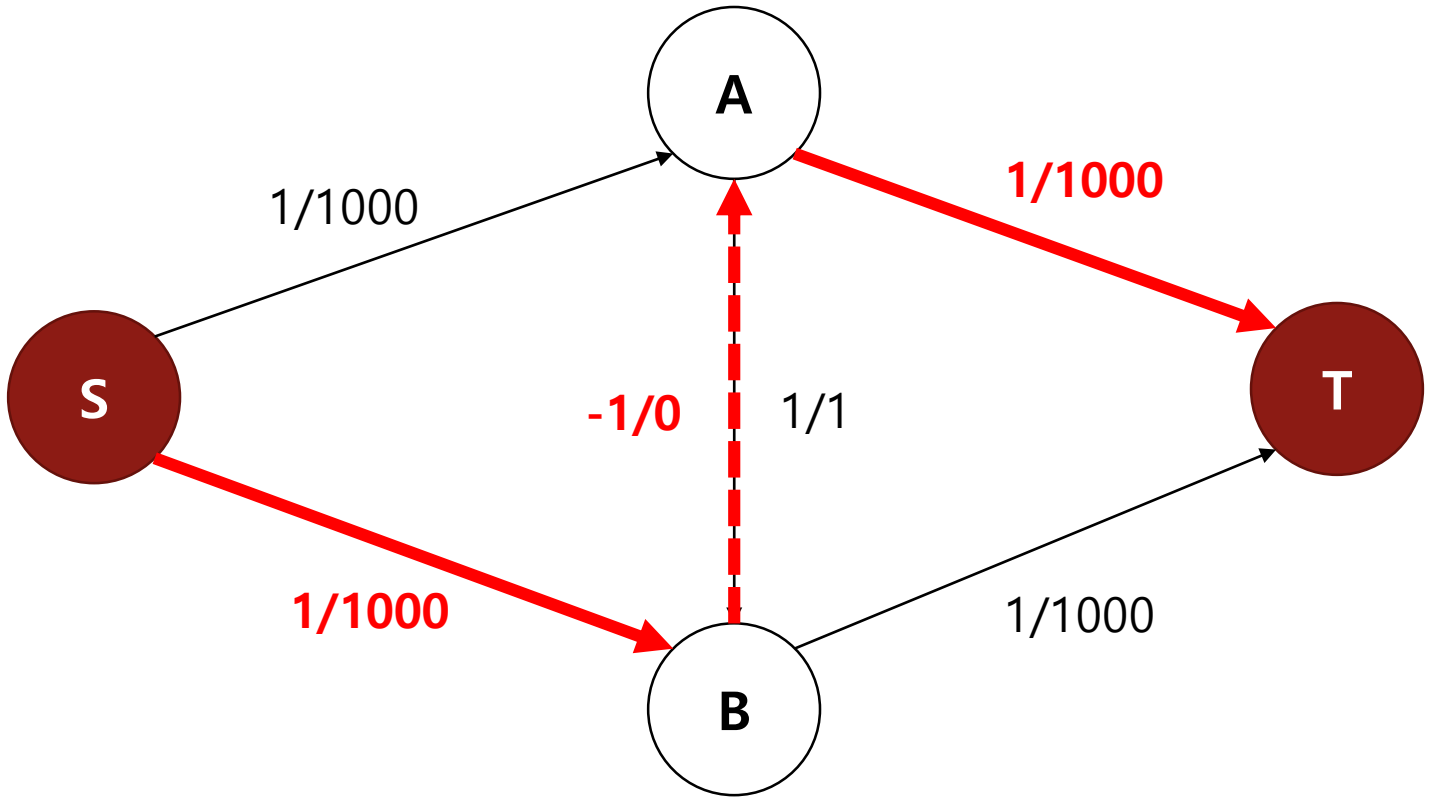
Ford-Fulkerson algorithm



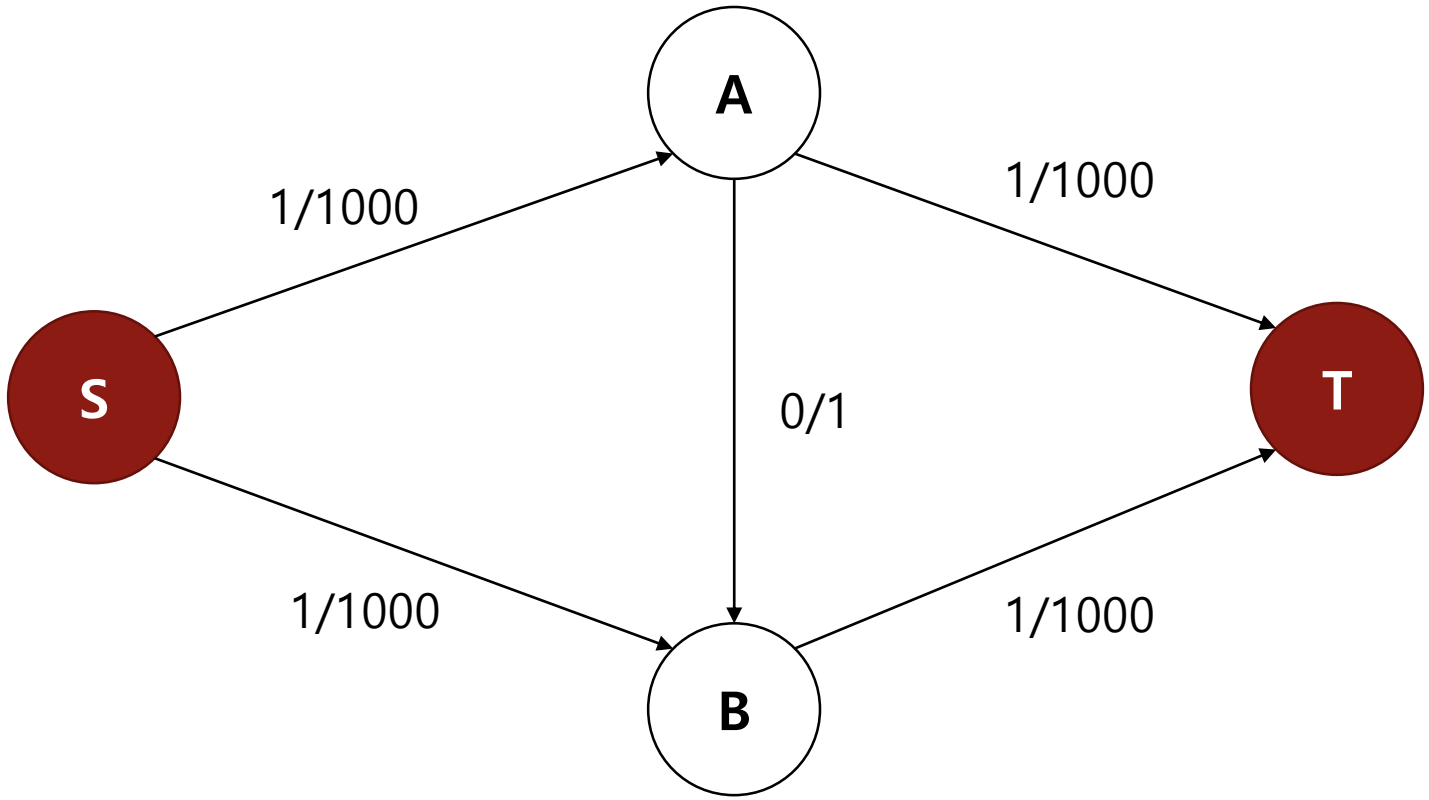
Ford-Fulkerson algorithm



Ford-Fulkerson algorithm



Ford-Fulkerson algorithm



Ford-Fulkerson algorithm

Time Complexity

$$O\big((|V| + |E|)f\big) = O(|E|f)$$

Edmonds-Karp Algorithm

1. BFS를 통해 '최단' Augmenting Path 찾기
2. 해당 경로로 Blocking Flow만큼의 유량 흘리기
3. Augmenting Path가 없을 때까지 1,2를 반복

Edmonds-Karp Algorithm

Time Complexity : $O(|V||E|^2)$

proof) https://koosaga.com/133

#11375 열혈강호

강호네 회사에는 직원이 N명이 있고, 해야할 일이 M개가 있다. 직원은 1번부터 N번까지 번호가 매겨져 있고, 일은 1번부터 M번까지 번호가 매겨져 있다.

각 직원은 한 개의 일만 할 수 있고, 각각의 일을 담당하는 사람은 1명이어야 한다.

각각의 직원이 할 수 있는 일의 목록이 주어졌을 때, M개의 일 중에서 최대 몇 개를 할 수 있는지 구하는 프로그램을 작성하시오.

입력

첫째 줄에 직원의 수 N과 일의 개수 M이 주어진다. ($1 \le N, M \le 1,000$)

둘째 줄부터 N개의 줄의 i번째 줄에는 i번 직원이 할 수 있는 일의 개수와 할 수 있는 일의 번호가 주어진다.

#11375 열혈강호

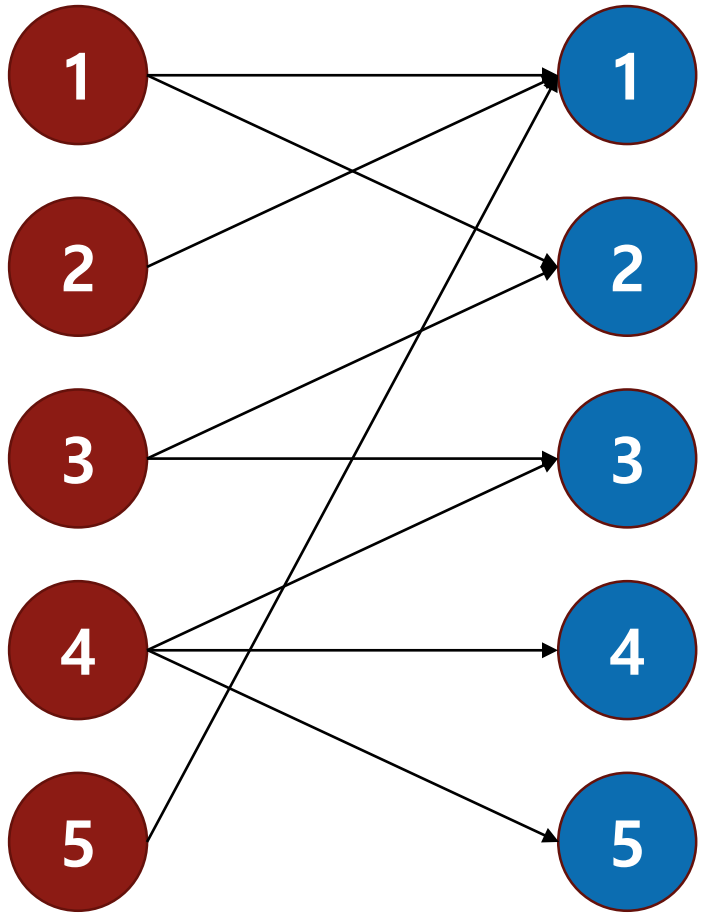
example)

5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1

#11375 열혈강호 4

example)

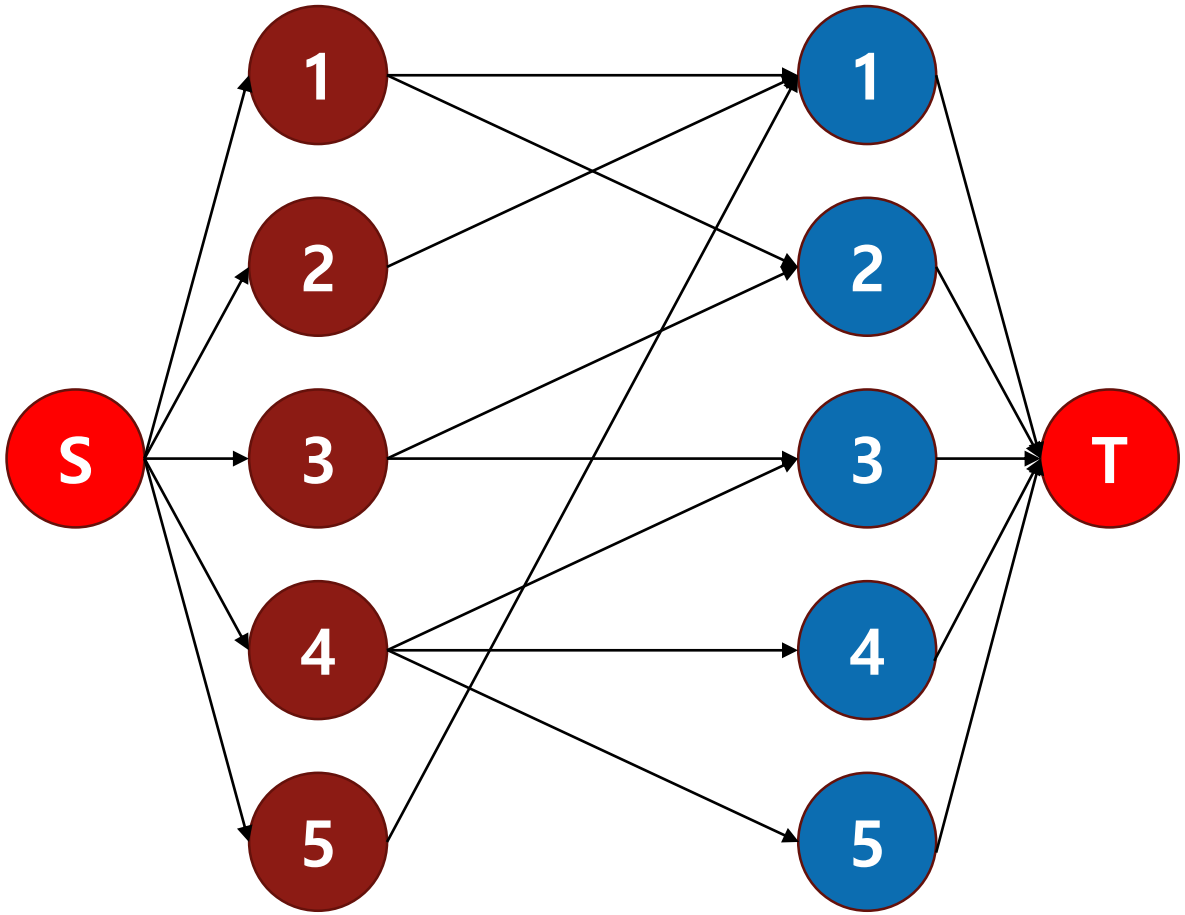
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

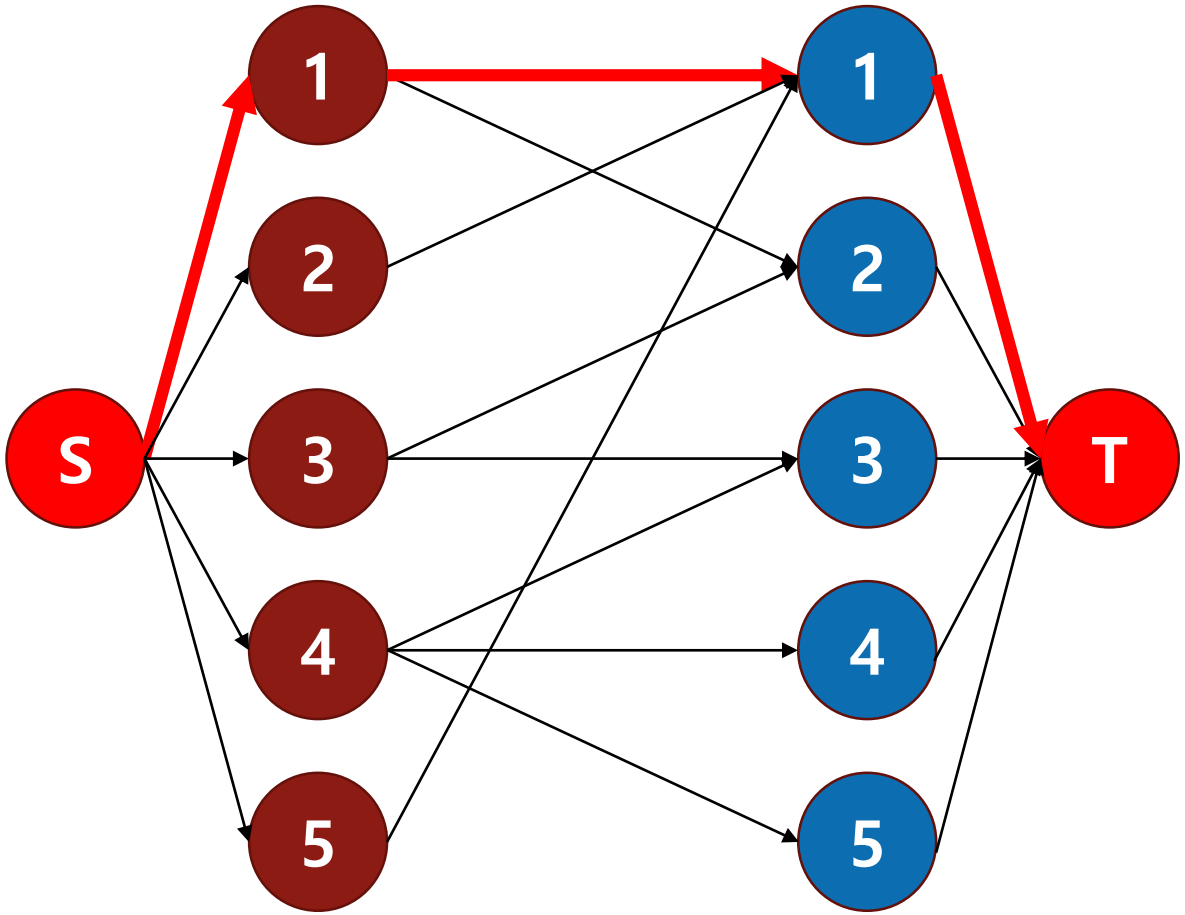
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

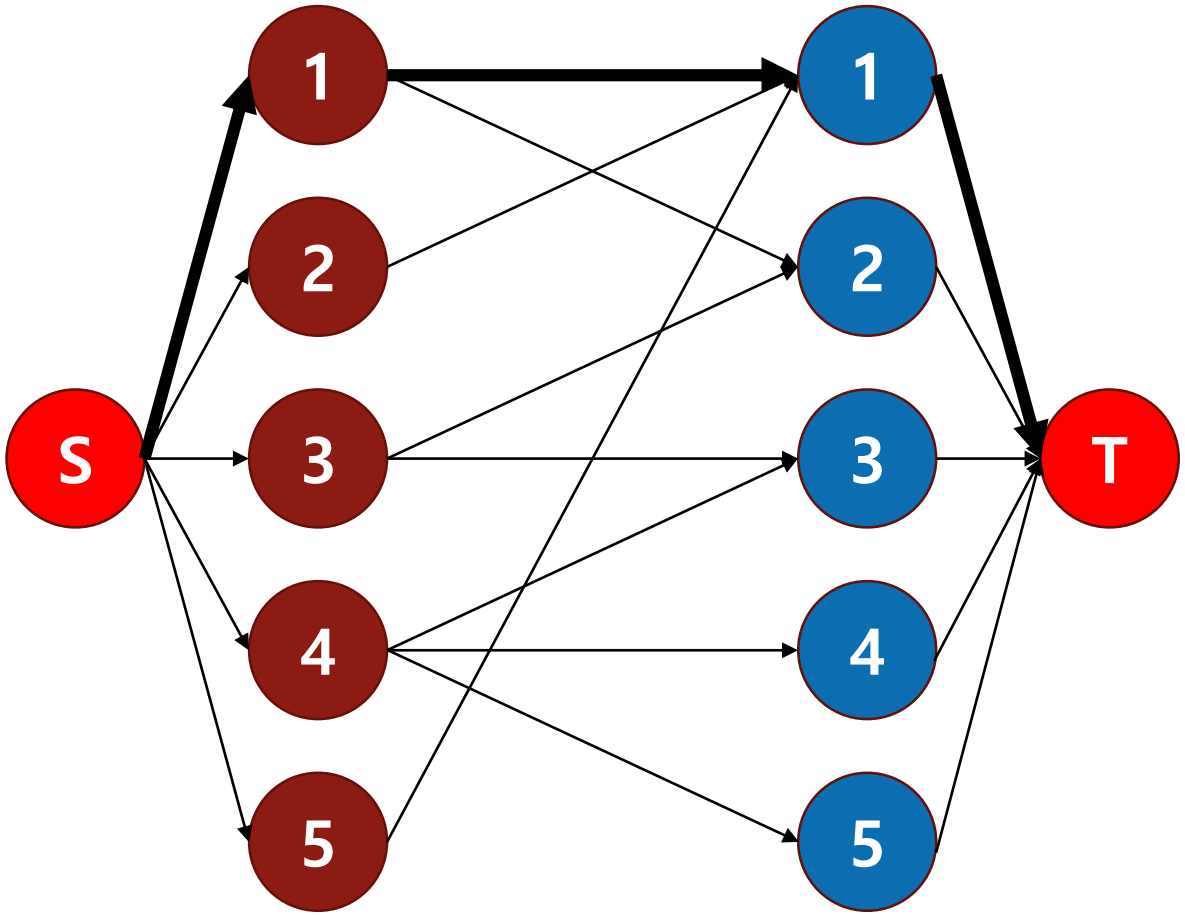
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

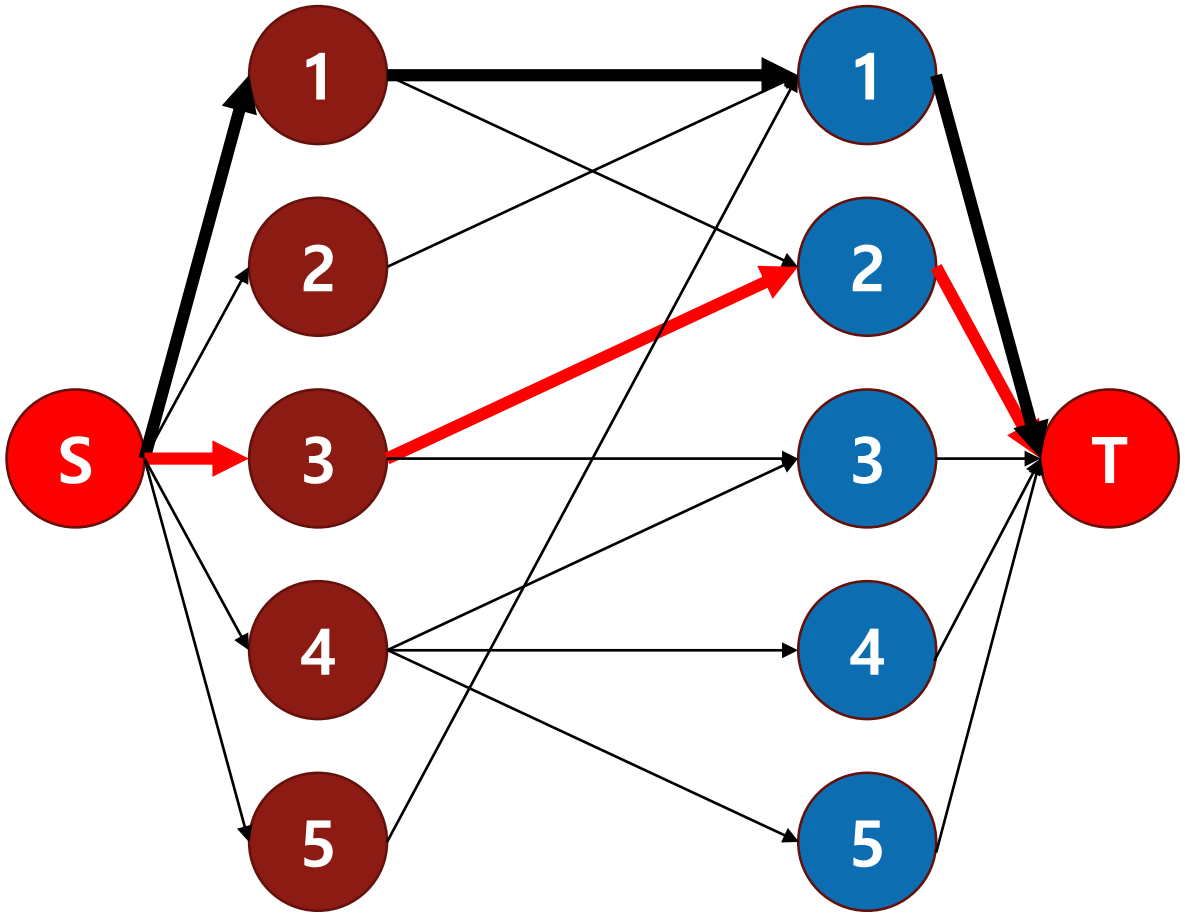
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

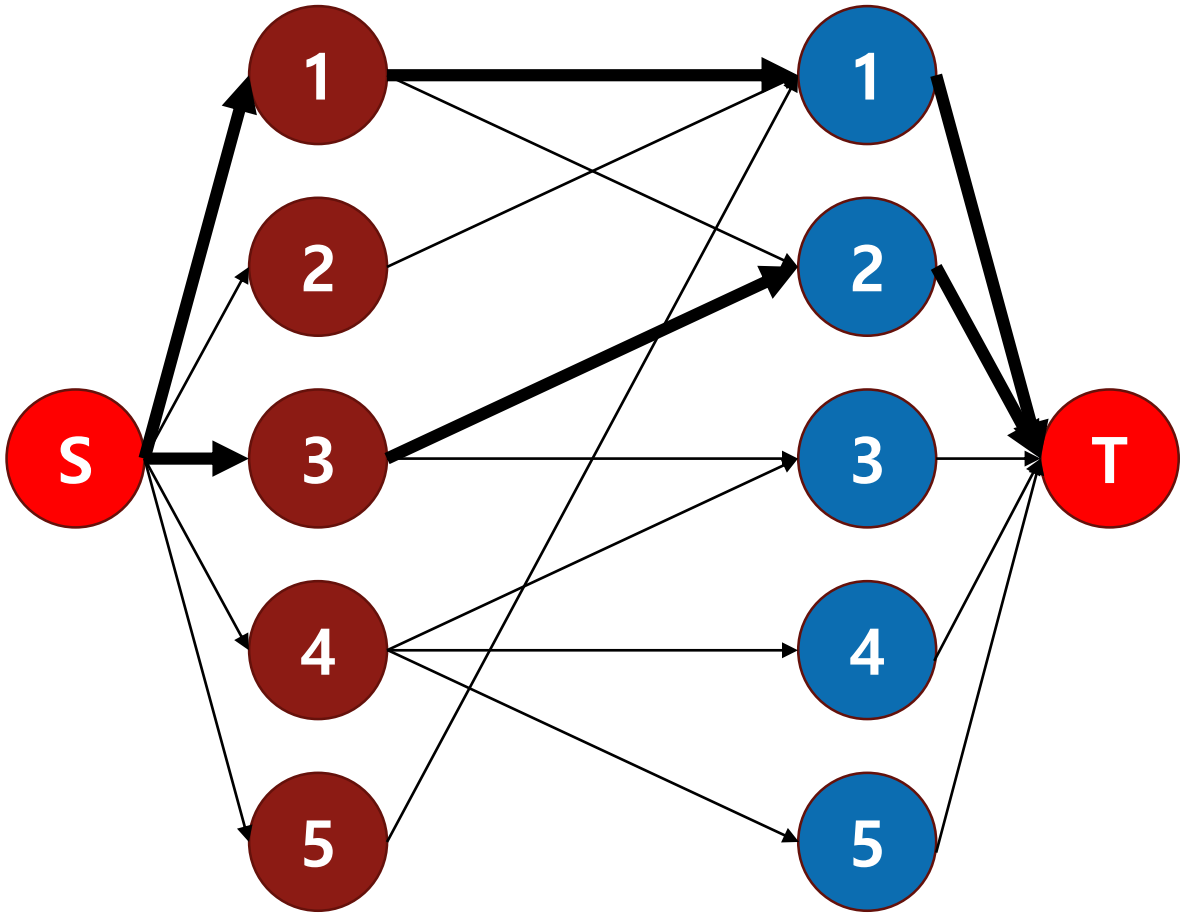
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

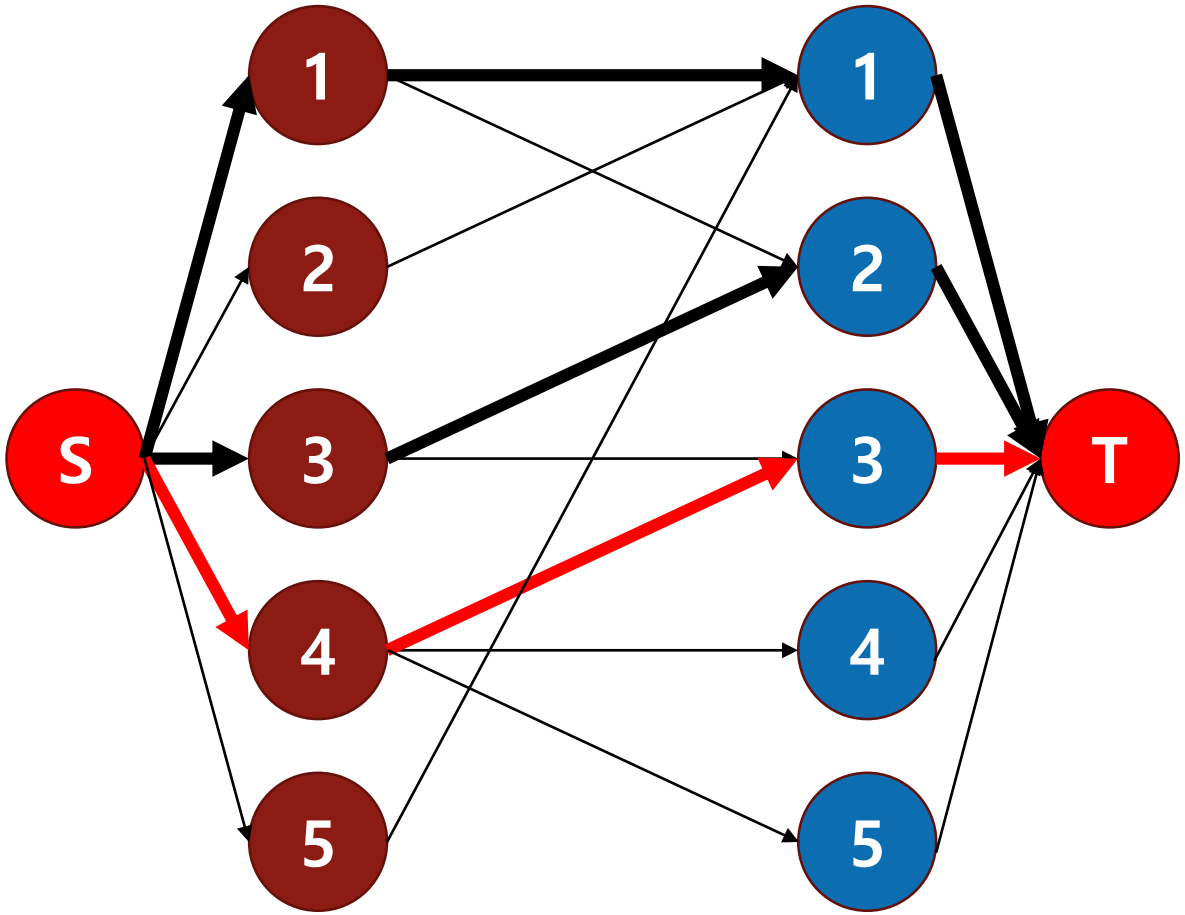
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

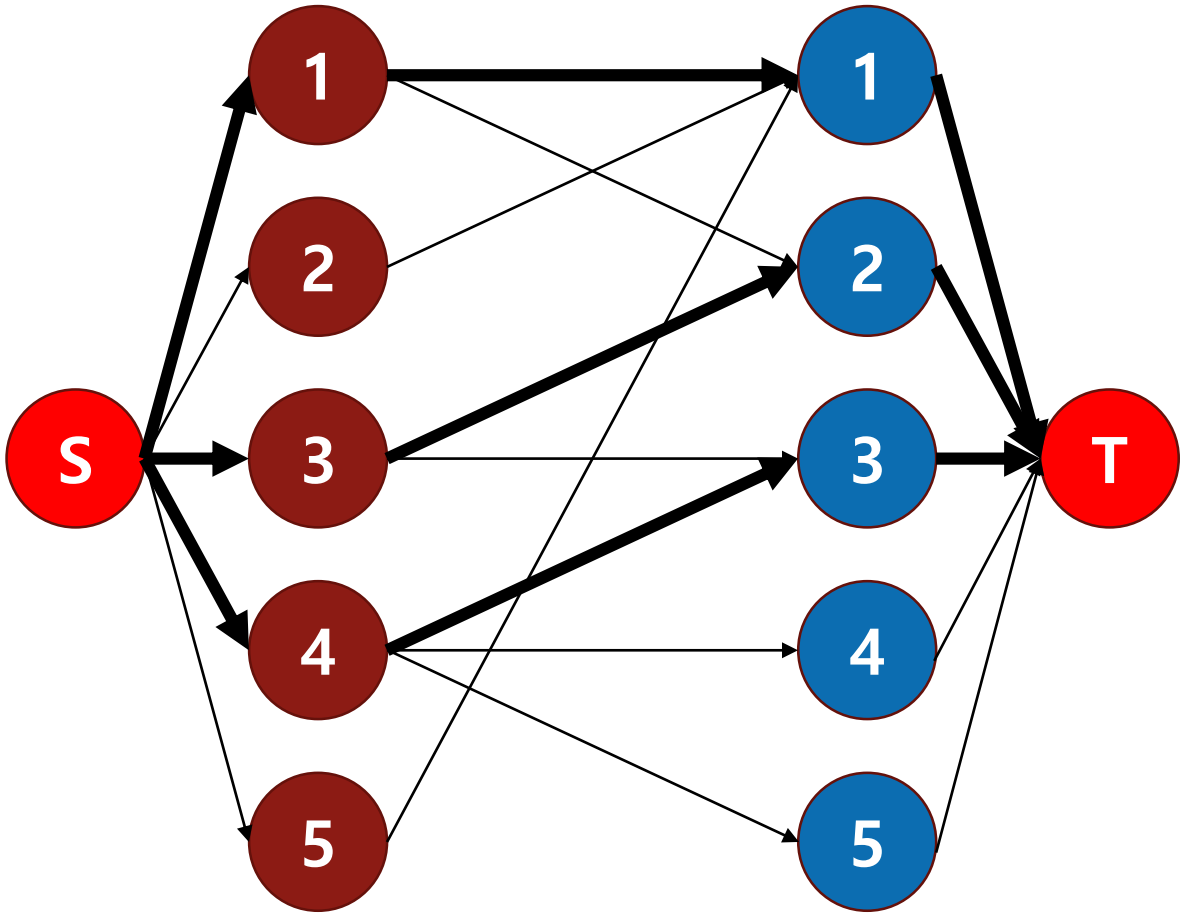
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호

example)

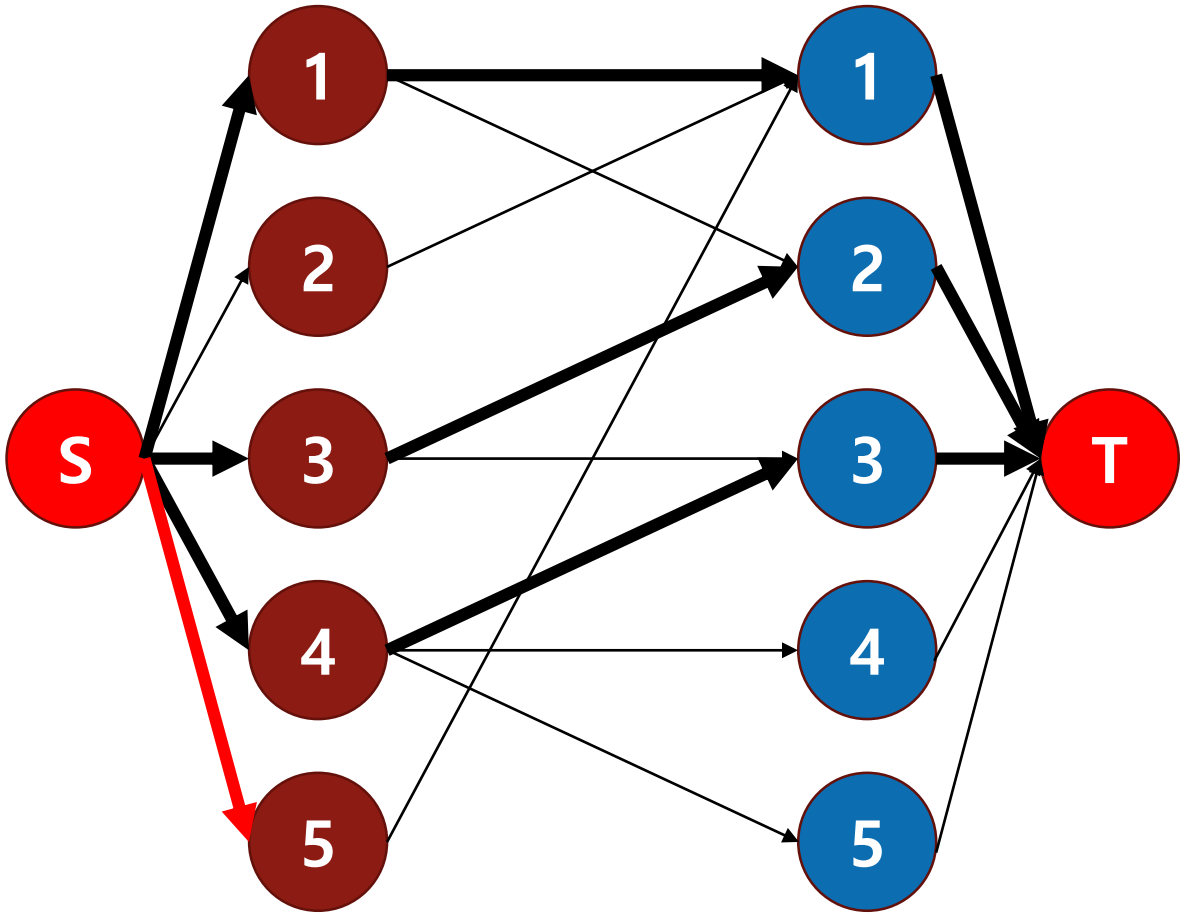
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

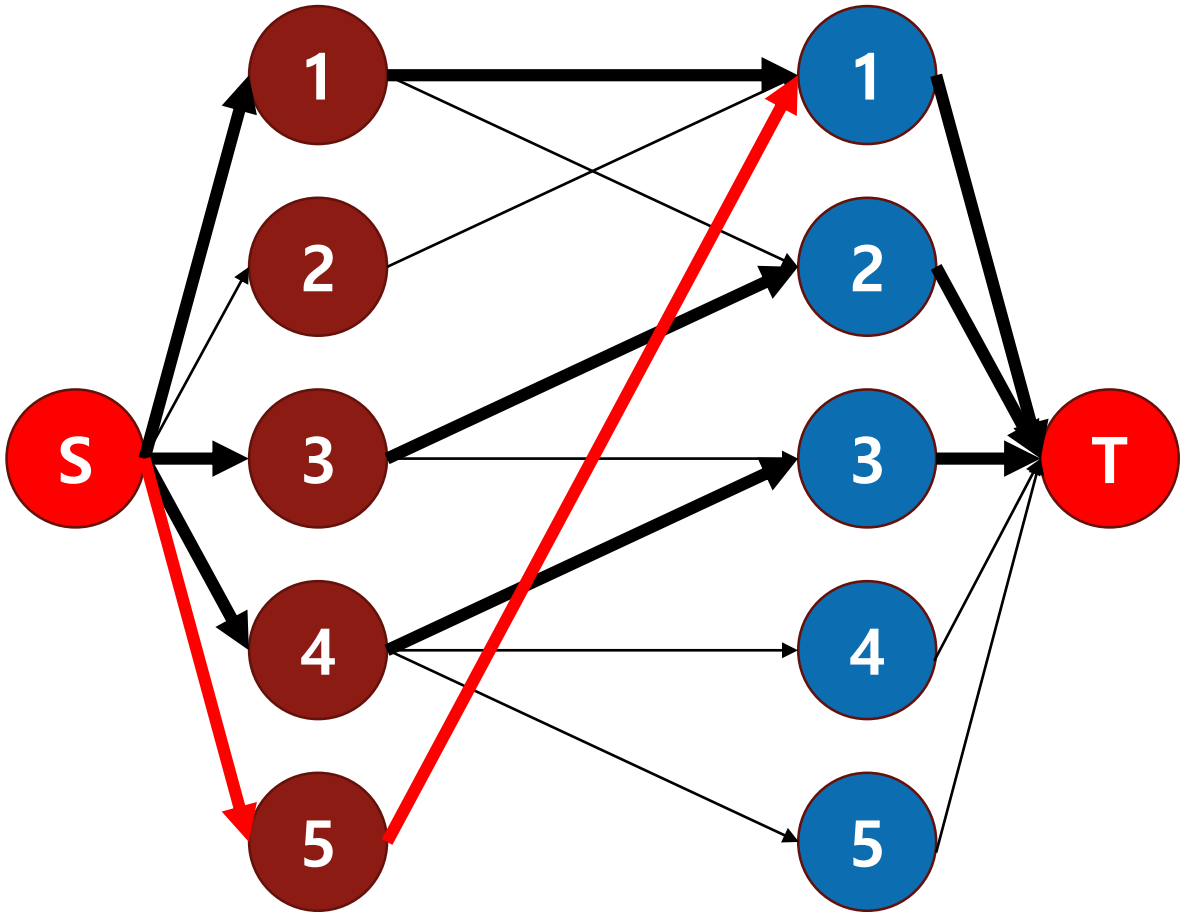
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

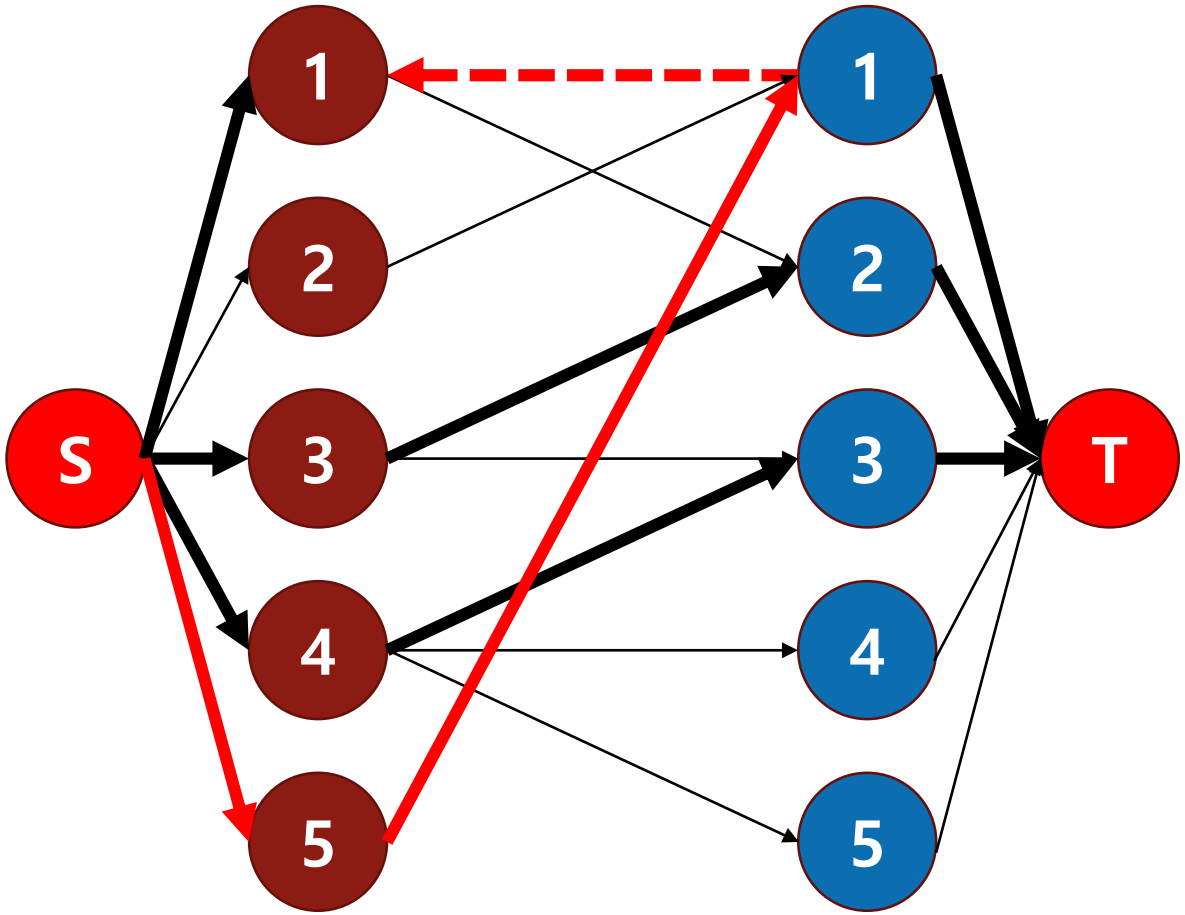
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

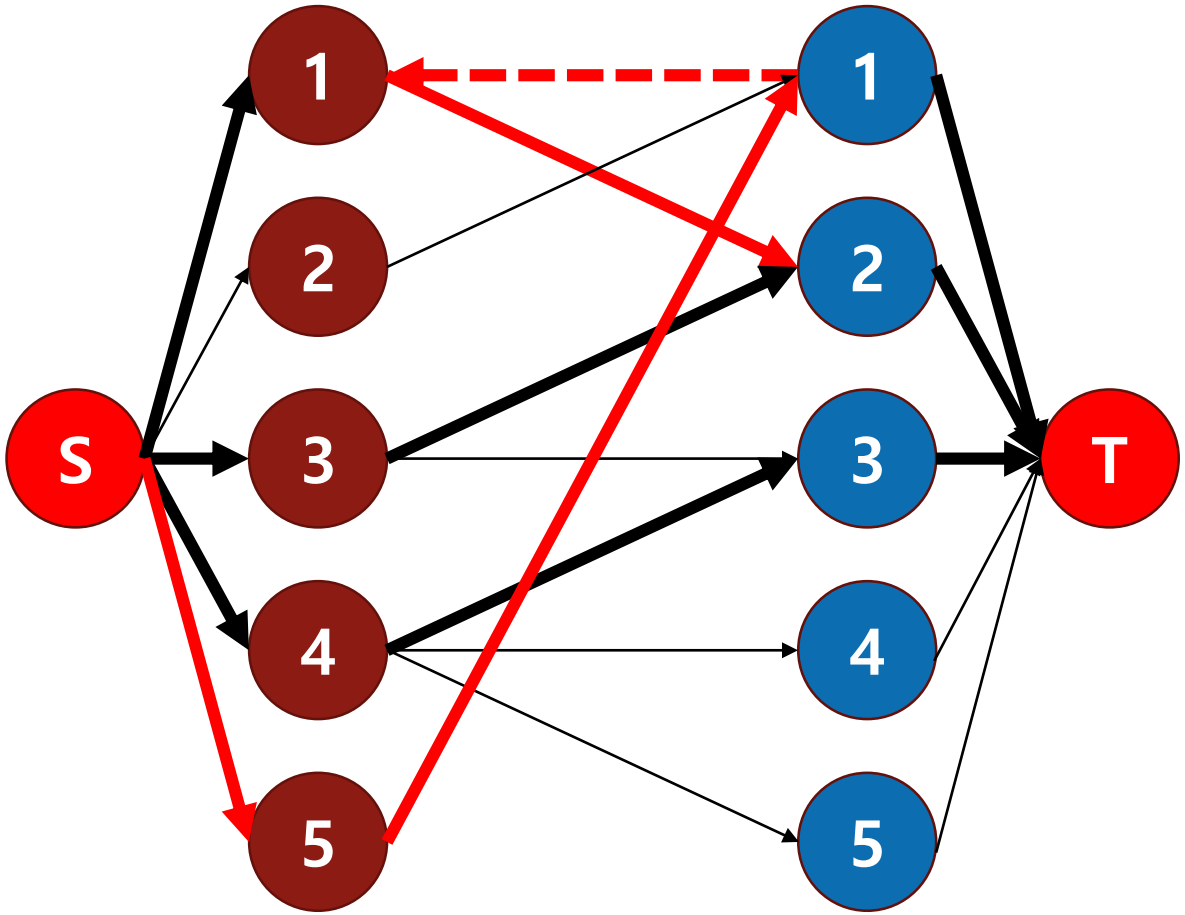
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

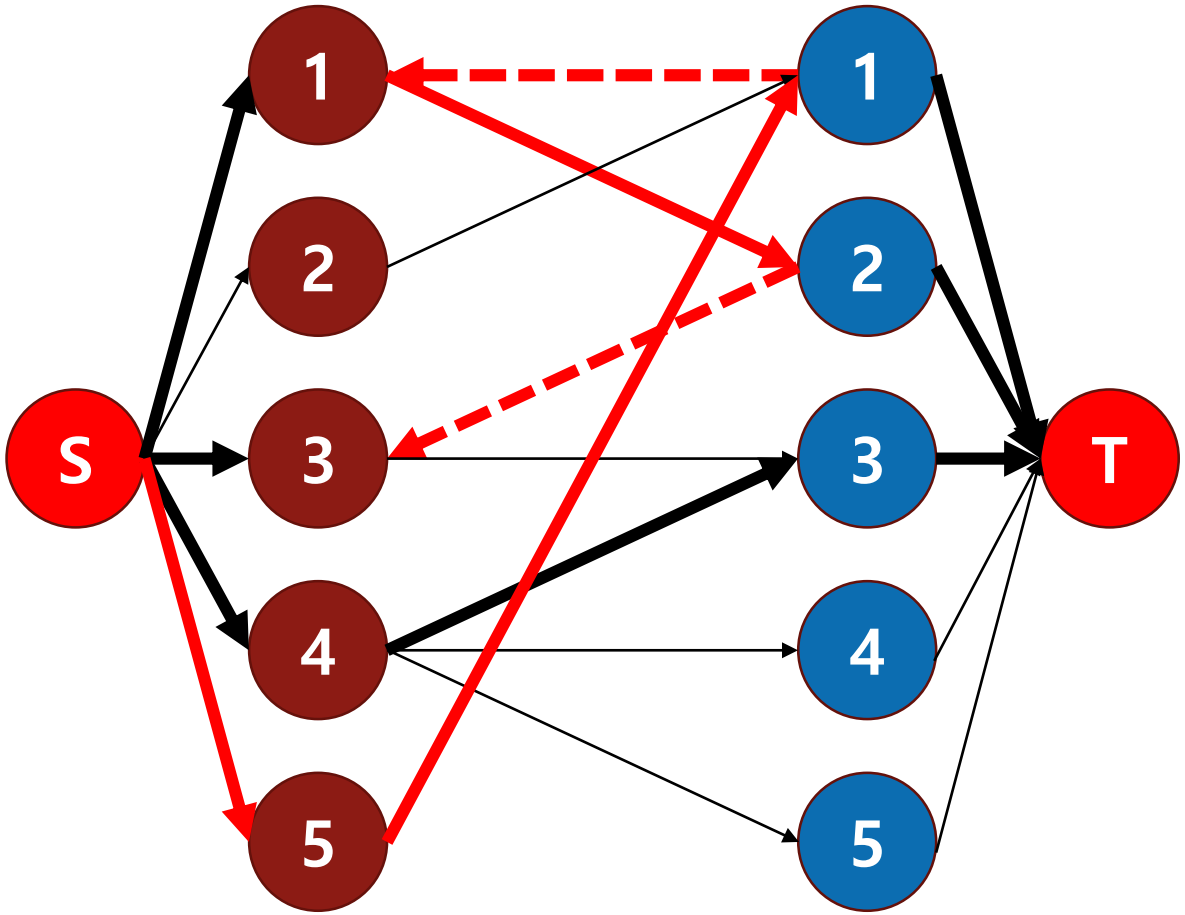
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

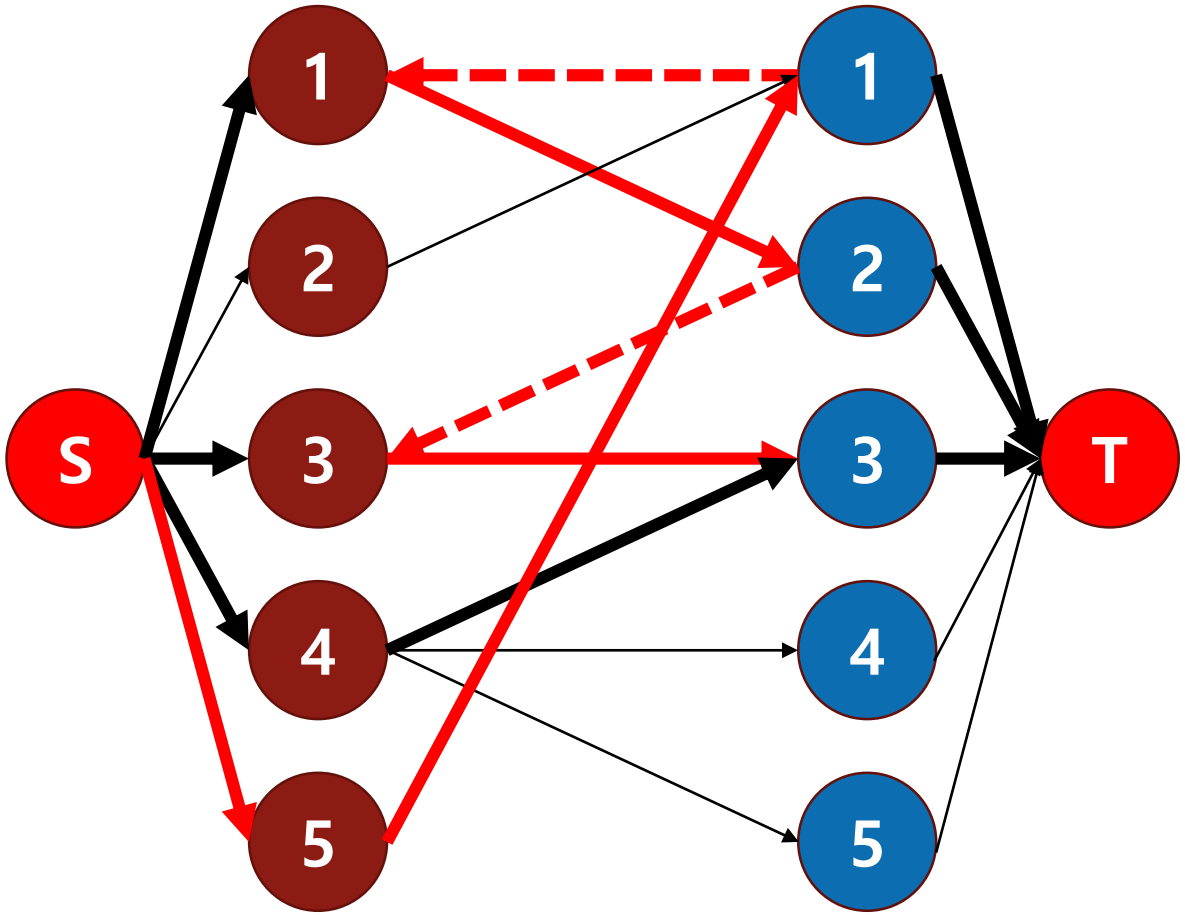
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

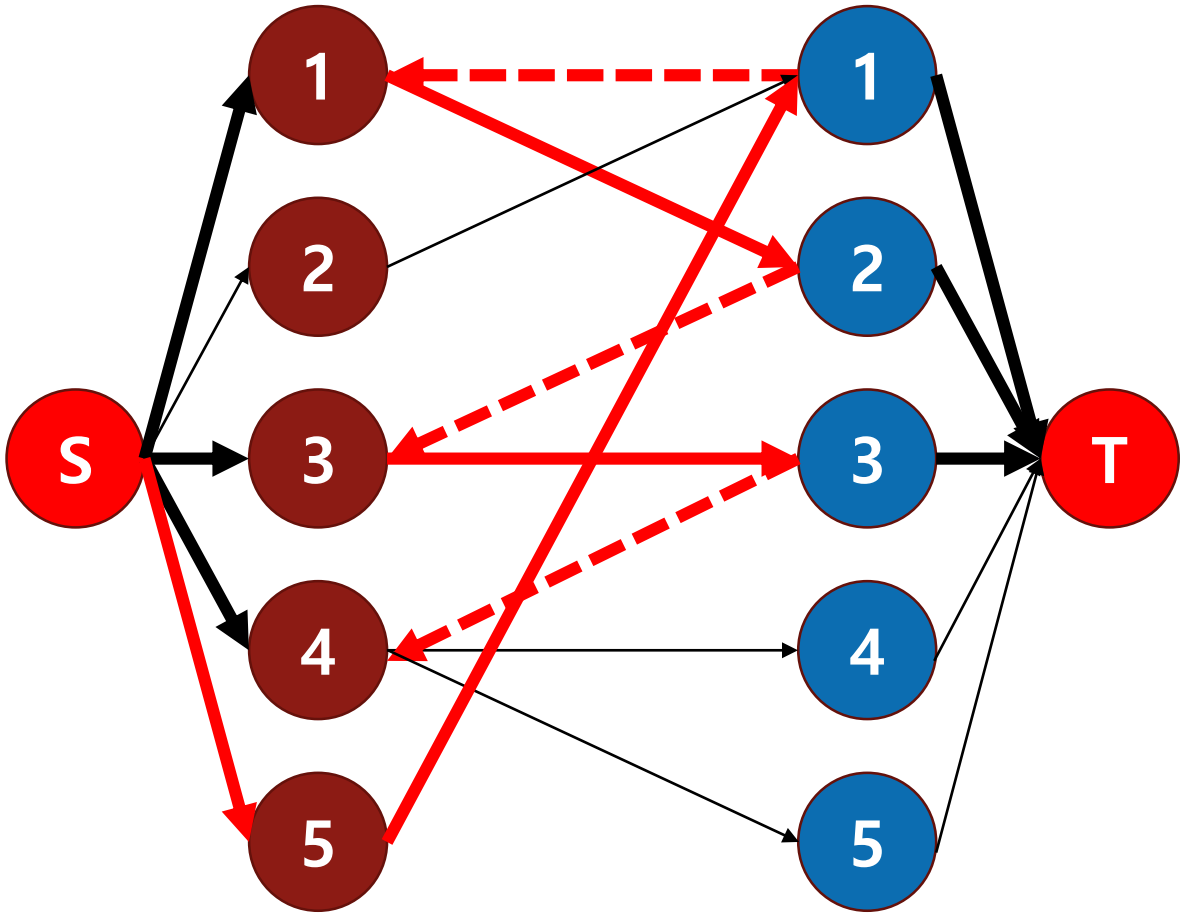
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

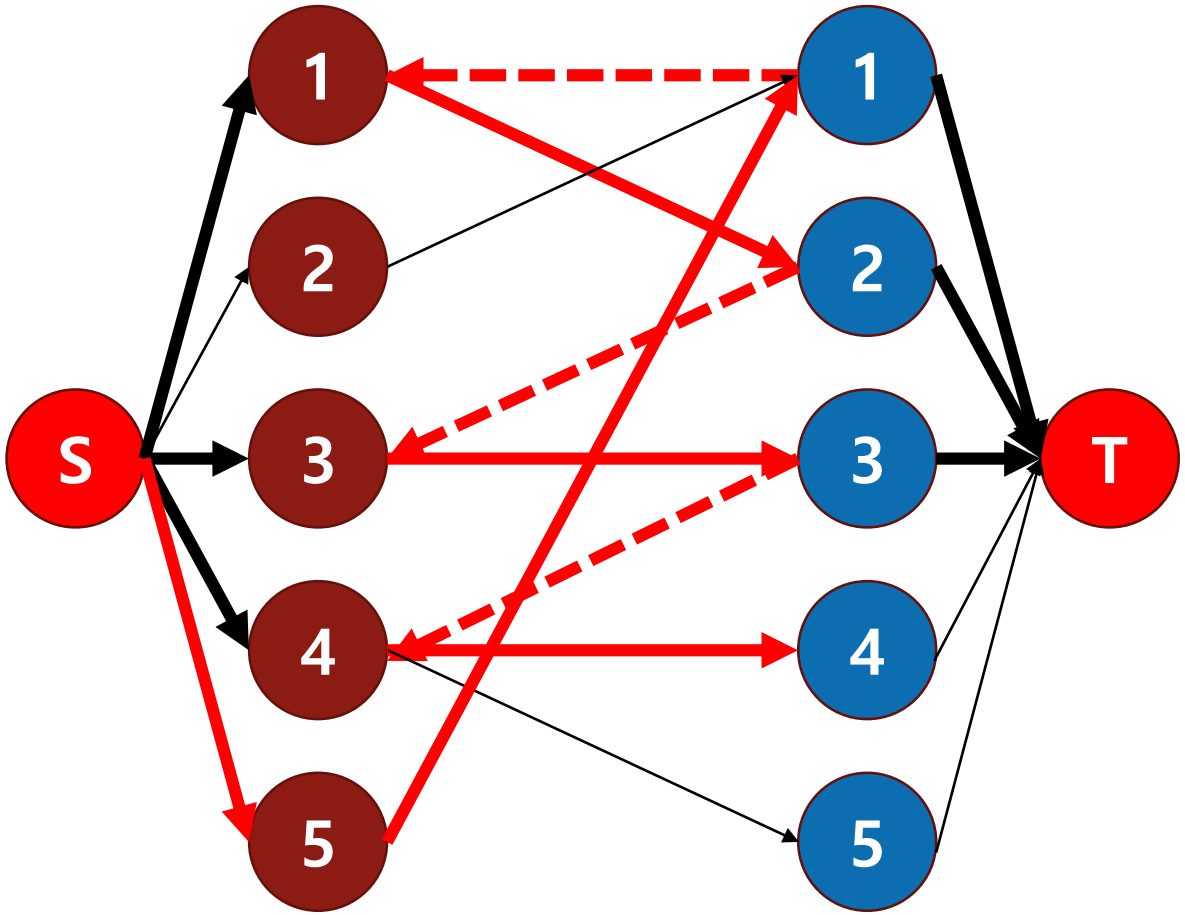
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

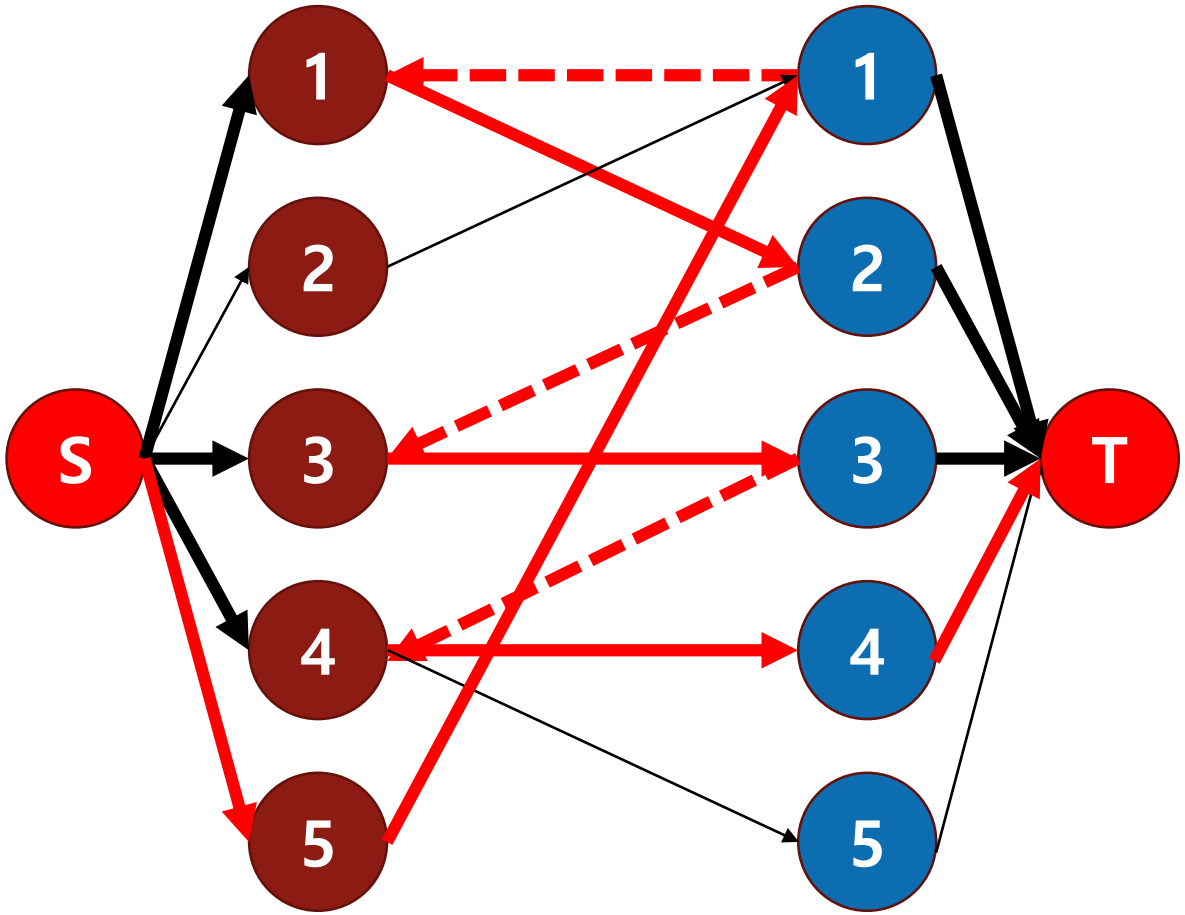
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

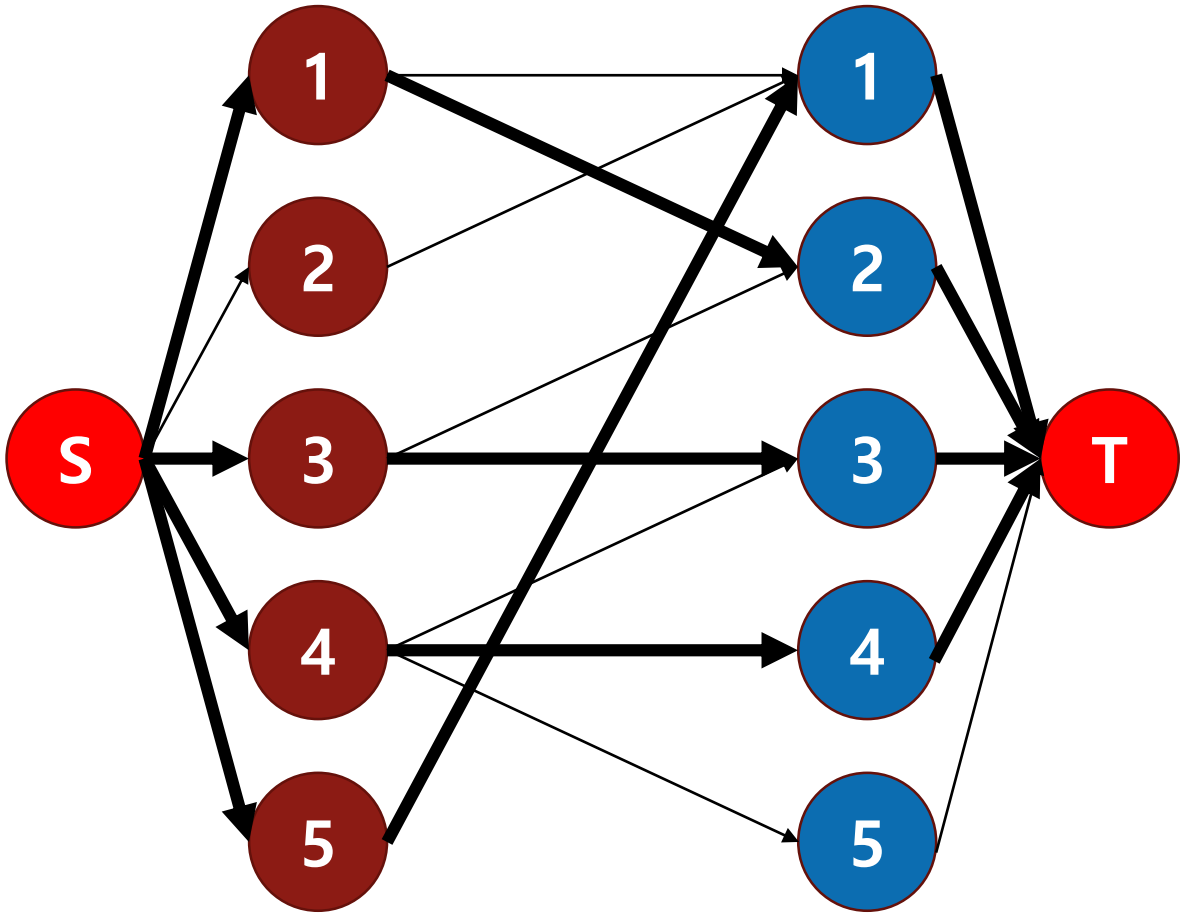
5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



#11375 열혈강호 4

example)

5 5
2 1 2
1 1
2 2 3
3 3 4 5
1 1



Edmonds-Karp Algorithm

1. BFS를 통해 '최단' Augmenting Path 찾기
2. 해당 경로로 Blocking Flow만큼의 유량 흘리기
3. Augmenting Path가 없을 때까지 1,2를 반복

Edmonds-Karp Algorithm

1. BFS를 통해 '최단' Augmenting Path 찾기
2. 해당 경로로 Blocking Flow만큼의 유량 흘리기
3. Augmenting Path가 없을 때까지 1,2를 반복

이전에!

Flow Graph

1. **Vertex**
2. **Flow Edge**
   - u, v, capacity, flow

```
 6       int S, E;
 7       const int mxn = 353;
 8       int c[mxn][mxn], f[mxn][mxn];
 9       vector<int> adj[mxn];
10
11   void addedge(int u, int v, int cap) {
12           c[u][v] = cap;
13           adj[u].push_back(v);
14           adj[v].push_back(u);
15       }
16
17   inline int residual(int u, int v)
18       { return c[u][v] - f[u][v]; }
19
```

1. BFS를 통해 '최단' Augmenting Path 찾기

```
23          int prev[mxn];
24          memset(prev, -1, sizeof(prev));
25          queue<int> q;
26          q.push(S);
27          while (!q.empty() && prev[E] == -1) {
28              int cur = q.front(); q.pop();
29              for (int next : adj[cur]) {
30                  if (residual(cur, next) > 0 && prev[next] == -1) {
31                      q.push(next);
32                      prev[next] = cur;
33                      if (next == E) break;
34                  }
35              }
36          }
```

2. 해당 경로로 Blocking Flow만큼의 유량 흘리기

```
40          int flow = INF;
41          for (int i = E; i != S; i = prev[i])
42              flow = min(flow, residual(prev[i], i));
43          for (int i = E; i != S; i = prev[i]) {
44              f[prev[i]][i]+= flow;
45              f[i][prev[i]]-= flow;
46          }
```

http://raararaara.blog.me/221616815261

#11375 열혈강호

- 유량 그래프의 구성

직원이 N명, 일이 M개

S = 0

직원 = 1 ~ N

일 = N+1 ~ N+1+M

E = N+M+2

```
19  void input() {
20      cin >> n >> m;
21      S = 0, E = n + m + 2;
22      for (int i = 1; i <= m; i++)
23          addedge(n + i, E, 1);
24      for (int i = 1; i <= n; i++) {
25          addedge(S, i, 1);
26          cin >> si;
27          while (si--) {
28              cin >> v;
29              addedge(i, n + v, 1);
30          }
31      }
32  }
```

#11375 열혈강호

- 문제의 목적?

최대로 할 수 있는 일의 개수

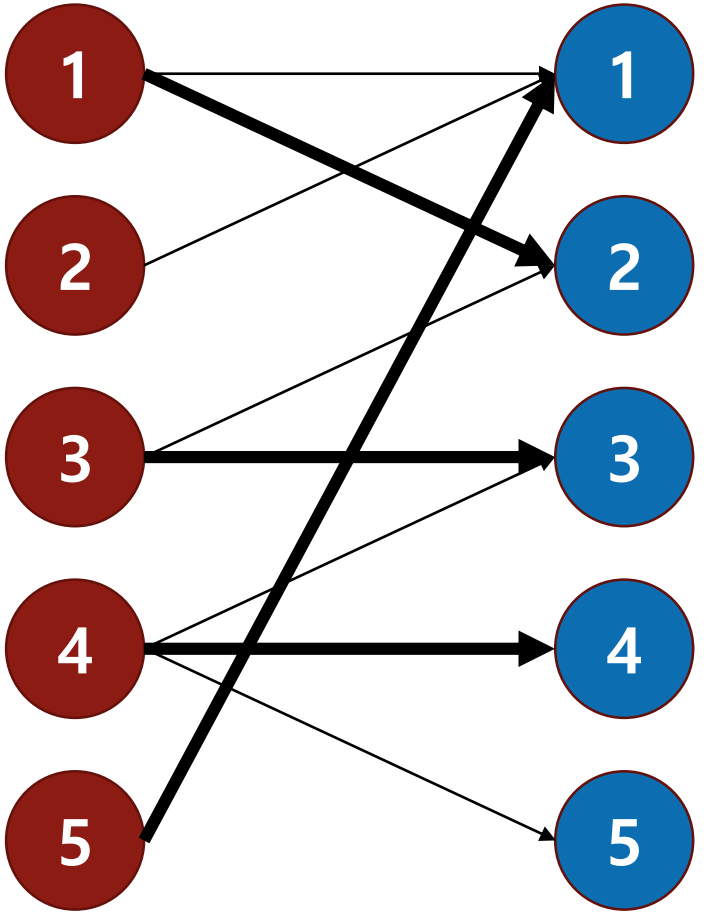
#11375 열혈강호

- 문제의 목적?

최대로 할 수 있는 일의 개수

= Augmenting Path의 개수

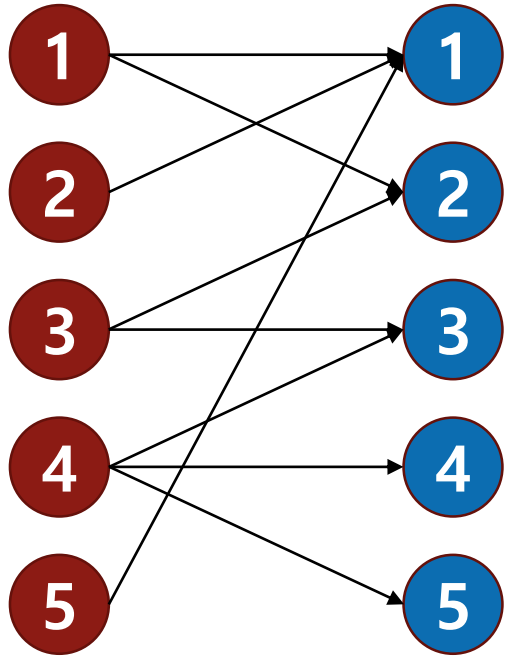
= 사람-일 간의 최대 매칭 개수



이분 그래프(Bipartite Graph)

- 정점을 두개의 그룹으로 나누었을 때

모든 간선의 양 끝 정점이 서로 다른 그룹에 속하는 형태의 그래프



이분 매칭(Bipartite Matching)

이분 그래프에서

A그룹의 정점에서 B그룹의 정점으로 간선을 연결할 때

각각이 일대일 대응으로 매칭된 형태.

#11375 열혈강호

- 문제의 목적?

최대로 할 수 있는 일의 개수

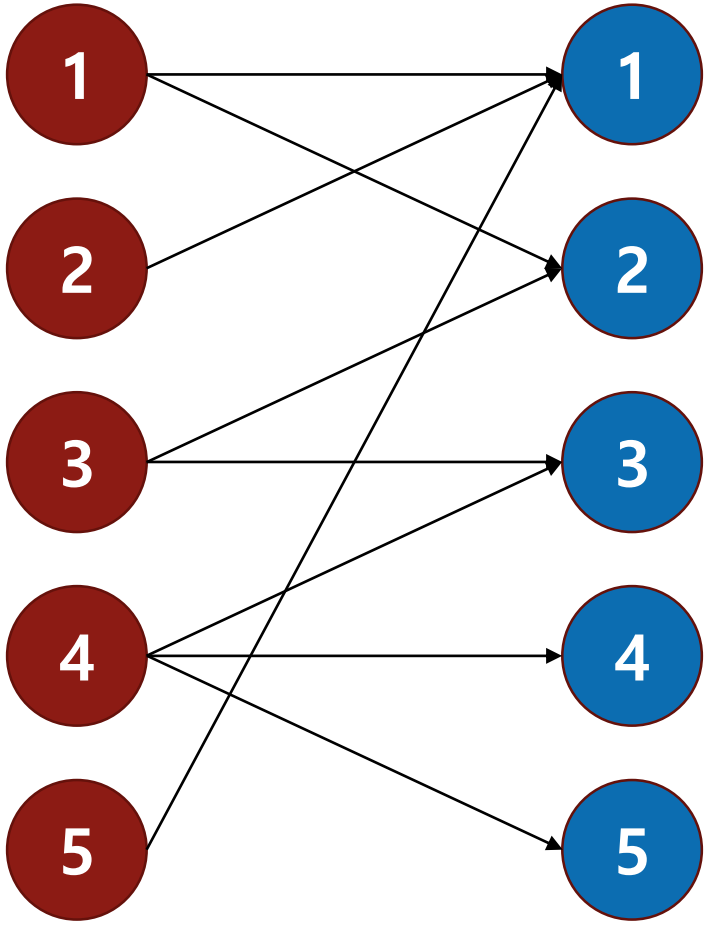
#11375 열혈강호

- 문제의 목적?

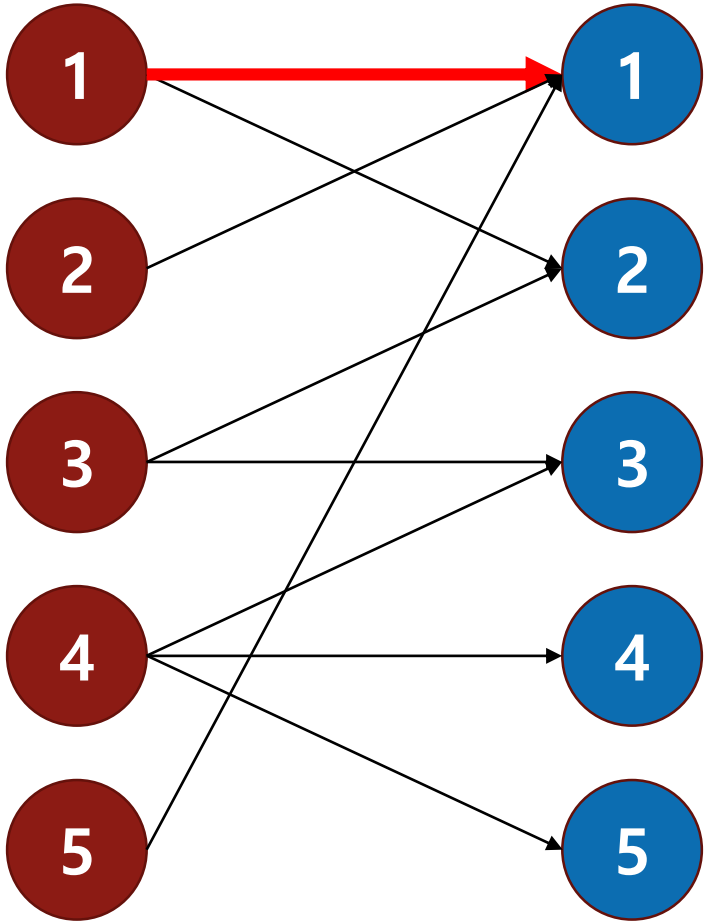
최대로 할 수 있는 일의 개수

= 이분매칭에서의 최대 매칭(Max Matching)의 수

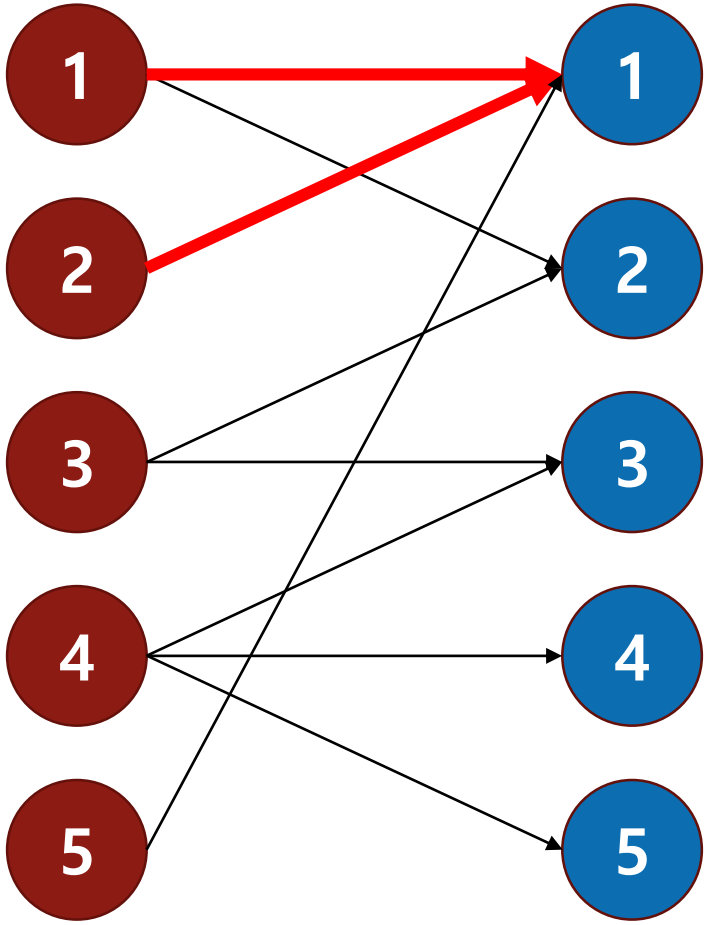
#11375 열혈강호 4



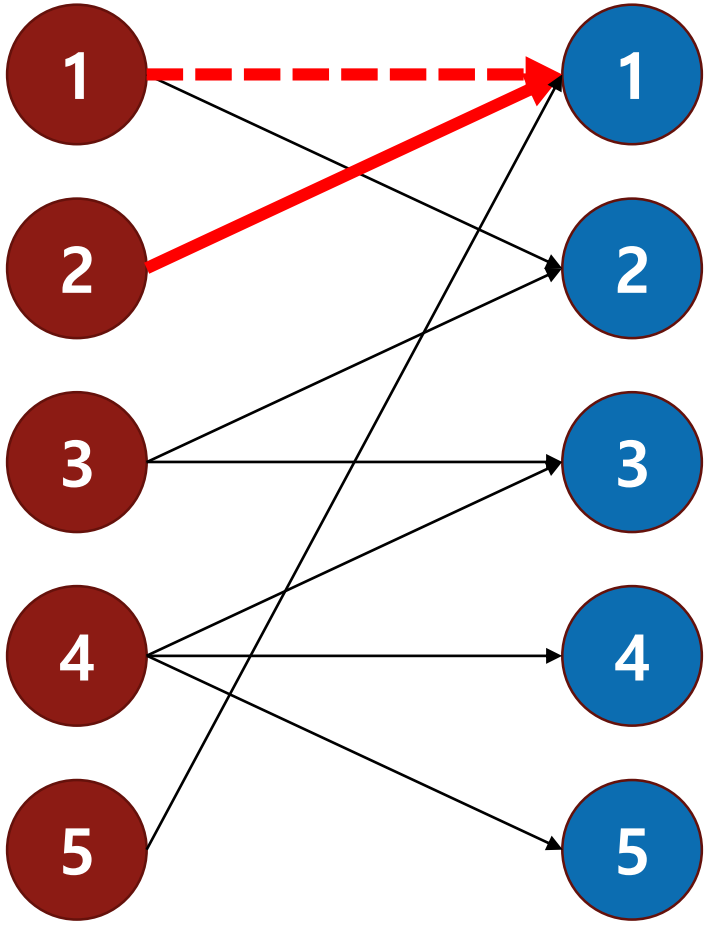
#11375 열혈강호 4



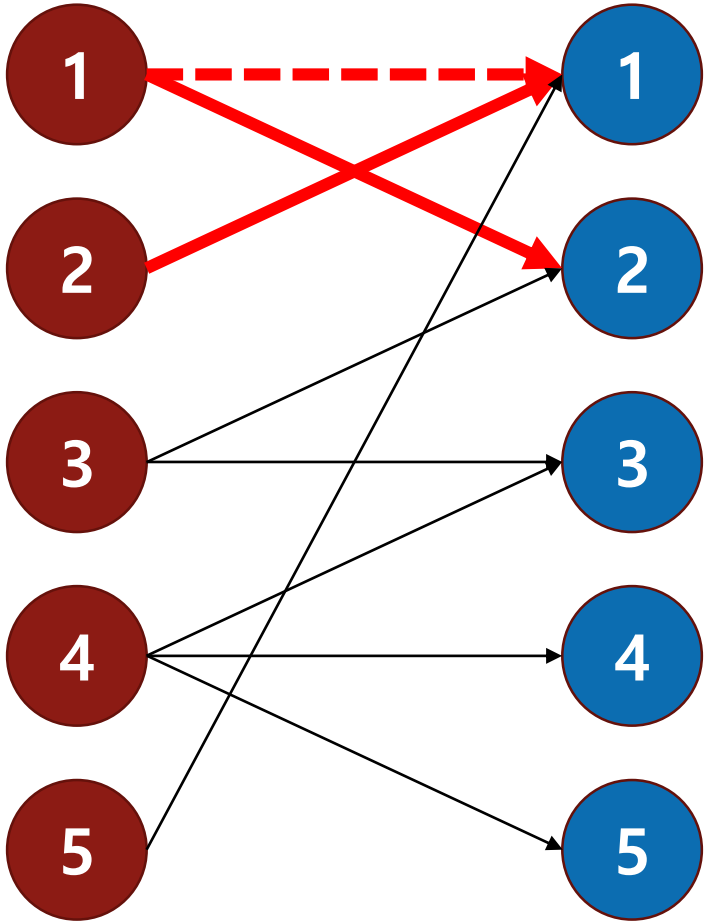
#11375 열혈강호 4



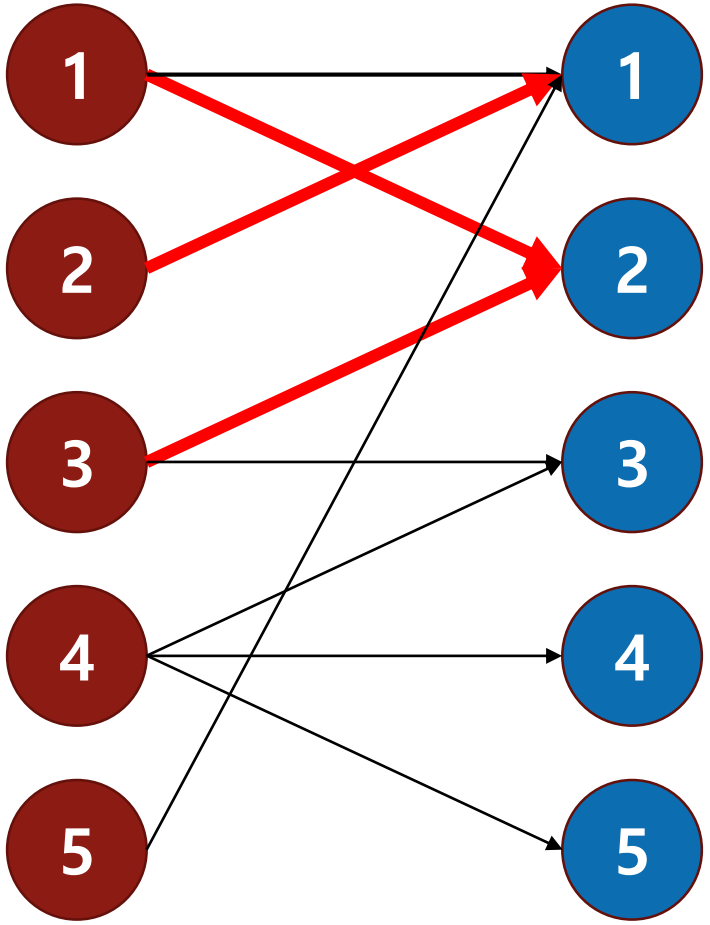
#11375 열혈강호 4



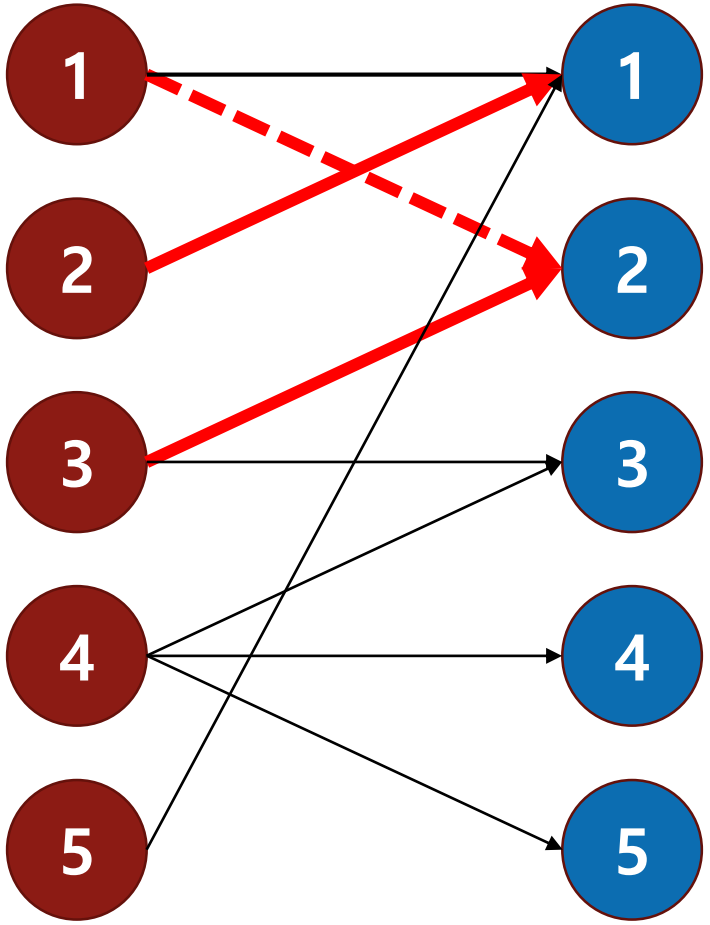
#11375 열혈강호 4



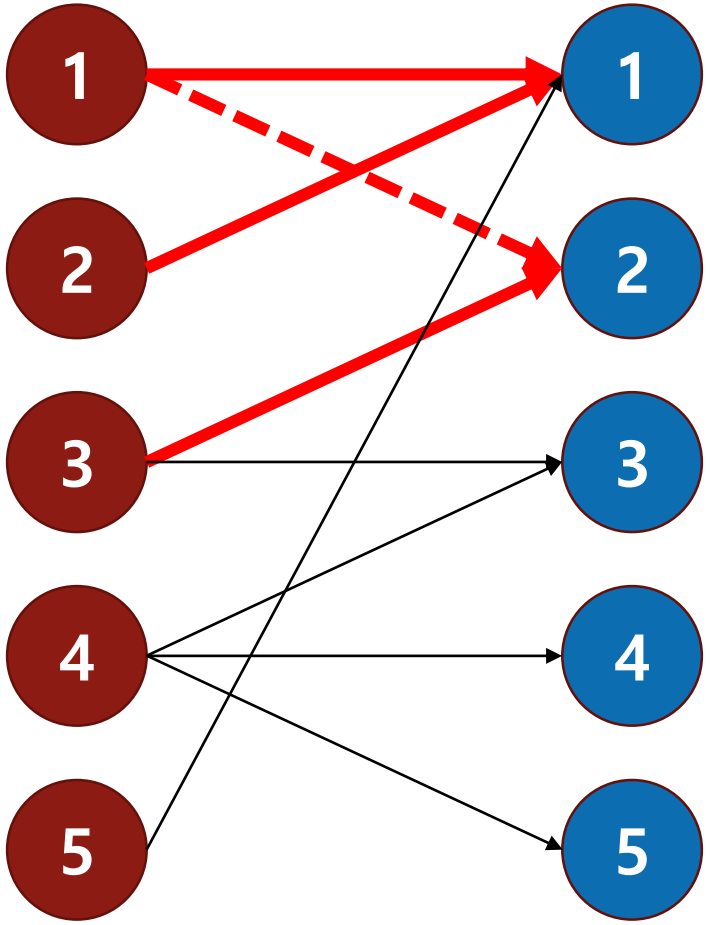
#11375 열혈강호 4



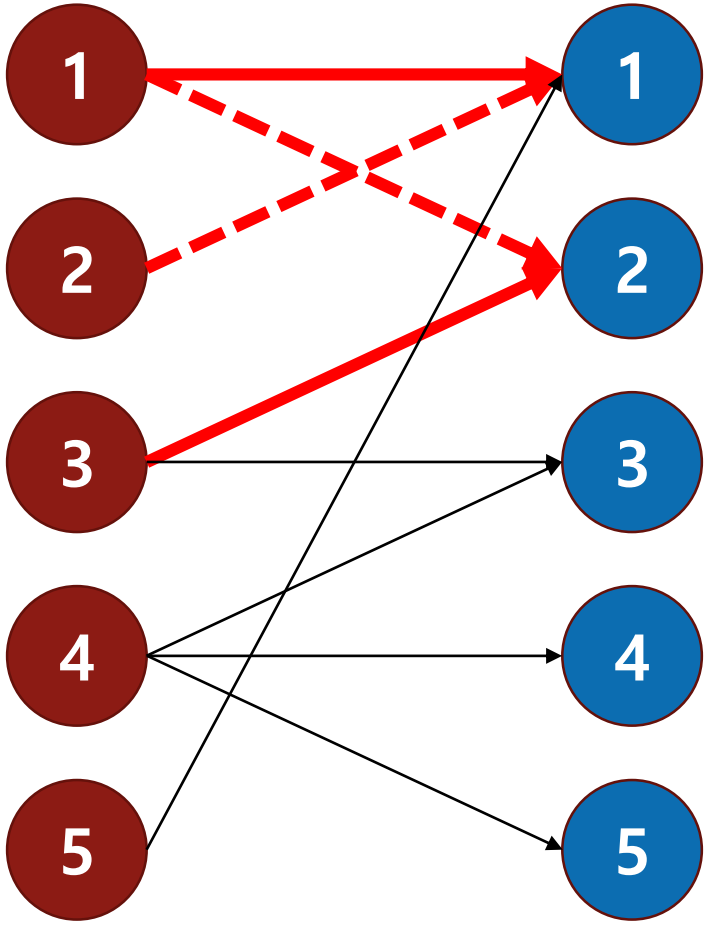
#11375 열혈강호 4



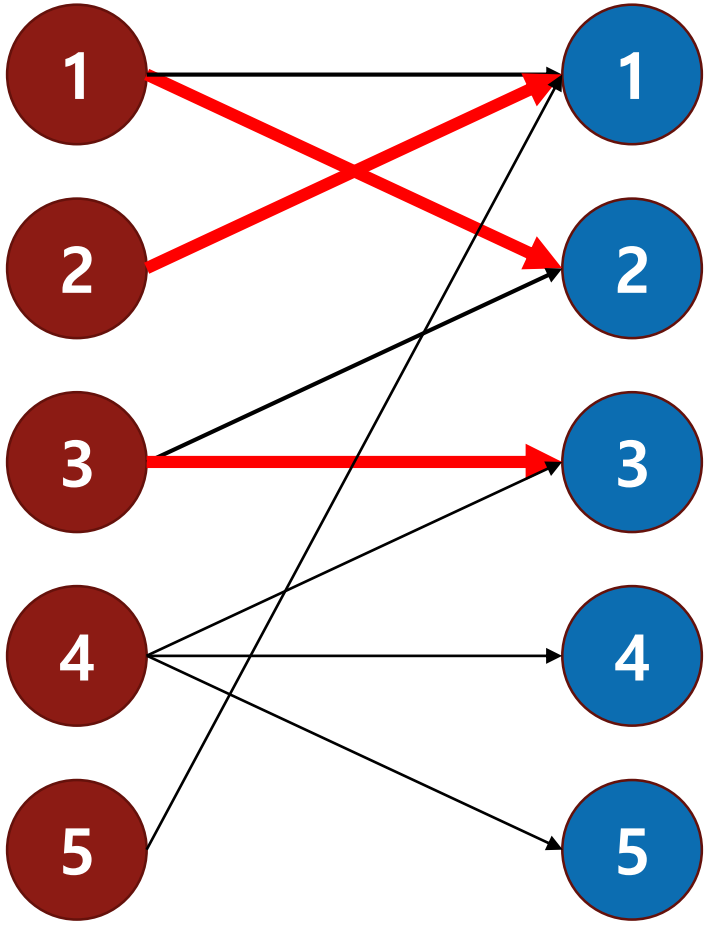
#11375 열혈강호 4



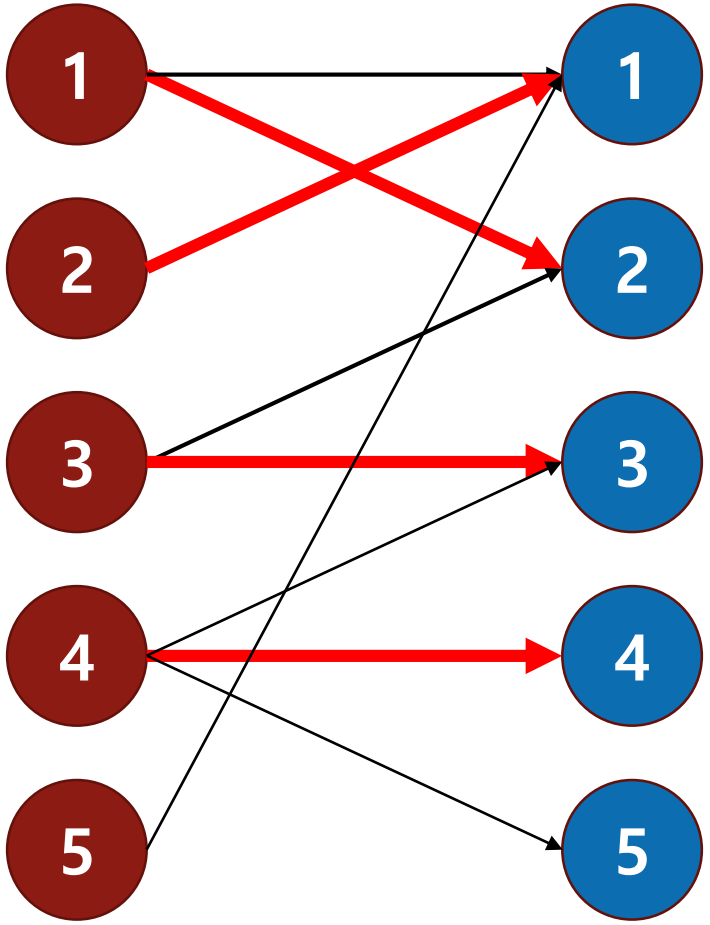
#11375 열혈강호 4



#11375 열혈강호 4



#11375 열혈강호 4



최대 매칭(Max Matching)

1. A그룹 원소에 대해 대응되는 B그룹 원소를 선택가능하면 바로 선택.
2. B그룹 원소가 이미 짝지어져 있다면, 짝지어진 A그룹의 원소가 다른 B그룹의 원소와 매칭이 될 수 있는지 확인.
3. 모든 A그룹의 원소에 대해 반복.

#11375 열혈강호 4

```
 6       int N, M, b[1001], vis[1001];
 7       vector<int> adj[1001];
 8
 9   void input() {
10       cin >> N >> M;
11       int x, v;
12       for (int i = 1; i <= N; i++) {
13           cin >> x;
14           while (x--) {
15               cin >> v;
16               adj[i].push_back(v);
17           }
18       }
19   }
```

#11375 열혈강호 4

```
31      int matching() {
32          int ret = 0;
33          for (int i = 1; i <= N; i++) {
34              memset(vis, 0, sizeof(vis));
35              if (dfs(i)) ret++;
36          }
37          return ret;
38      }
```

모든 A그룹의 원소에 대해 매칭 가능 여부 확인

#11375 열혈강호 4

```
21      bool dfs(int cur) {
22          vis[cur] = true;
23          for (int next : adj[cur])
24              if (!b[next] || !vis[b[next]] && dfs(b[next])) {
25                  b[next] = cur;
26                  return true;
27              }
28          return false;
29      }
```

대응 여부 확인, 실패시 다른 매칭 가능 여부 확인

Problem Set

- 4 11375 열혈강호
- 2 1799 비숍
- 4 6086 Total Flow
- 4 2188 축사 배정
- 4 11376 열혈강호 2
- 3 1017 소수 쌍
- 3 3295 단방향 링크 네트워크