알고스 3주차 스터디

네트워크 플로우, MCMF

2012685 최예헌

목차

01 Network Flow

Network Flow

- 그래프에서 두 정점 사이에 얼마나 많은 유량(flow)을 보낼 수 있는지 계산하는 알고리즘
- 그래프 내 source 정점에서 유량을 발생시켜 간선을 통해 sink 정점에 도달시키는 것이 목표
- = 최대 유량 (Maximum Flow) 알고리즘

기본 용어

- 용량 (Capacity , c(u, v))

  : 정점 u에서 v로 가는 간선에 흐를 수 있는 최대 유량(가중치)

- 유량 (Flow , f(u, v))

  : 정점 u에서 v로의 간선에 실제로 흐르는 유량

- 잔여 용량 (Residual Capacity , r(u, v))

  : c(u, v) - f(u, v)

- 소스 (Source) & 싱크 (Sink)

  : 유량이 시작되는 정점 & 모든 유량이 도착하는 정점

기본 용어

- 증가 경로 (Augmenting Path)

  : 유량이 흐를 수 있는 source에서 sink까지 가는 경로

- 유량 상쇄

  : 모든 경로에 기존에 존재하는 간선들과 반대되는 방향의 간선을 추가한 뒤,

  각 간선으로 유량을 흘려보냈을 때, 반대 방향의 간선으로도 음의 유량을 흘려보냄으로써

  유량을 상쇄 시키는 것.

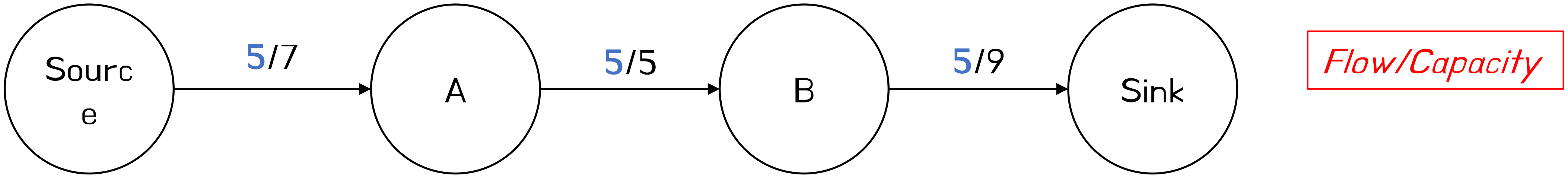
  음의 유량을 기록함으로써, 잔여 용량을 남겨 추가적인 경로를 탐색할 수 있도록 하기 위한 작업.

  두 정점이 서로에게 유량을 보내주는 것은 의미가 없기 때문에 성립 가능

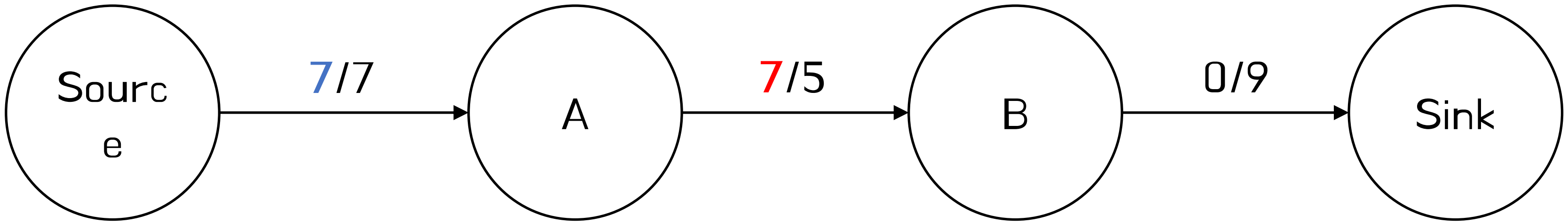
01

속성

1. 특정 경로를 따라 유량 보낼 때, 그 경로에 포함된 간선 중 가장 용량이 작은 간선에 의해 유량이 결정된다.



- 만약 Source에서 7의 유량을 내보낸다면?



- Source에서 Sink까지 막힘없이 흐를 수 있는 최적의 유량 => 5

속성

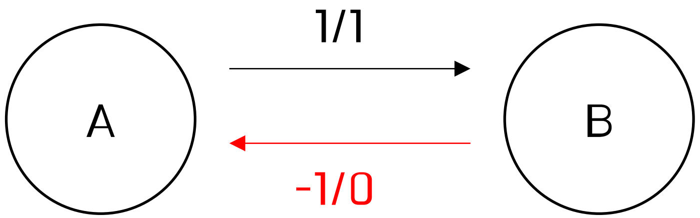
2. 용량 제한 속성

: 흐르는 양이 항상 그 간선의 용량을 넘을 수 없다.

$f(u, v) <= c(u, v)$

3. 유량의 대칭성

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



01

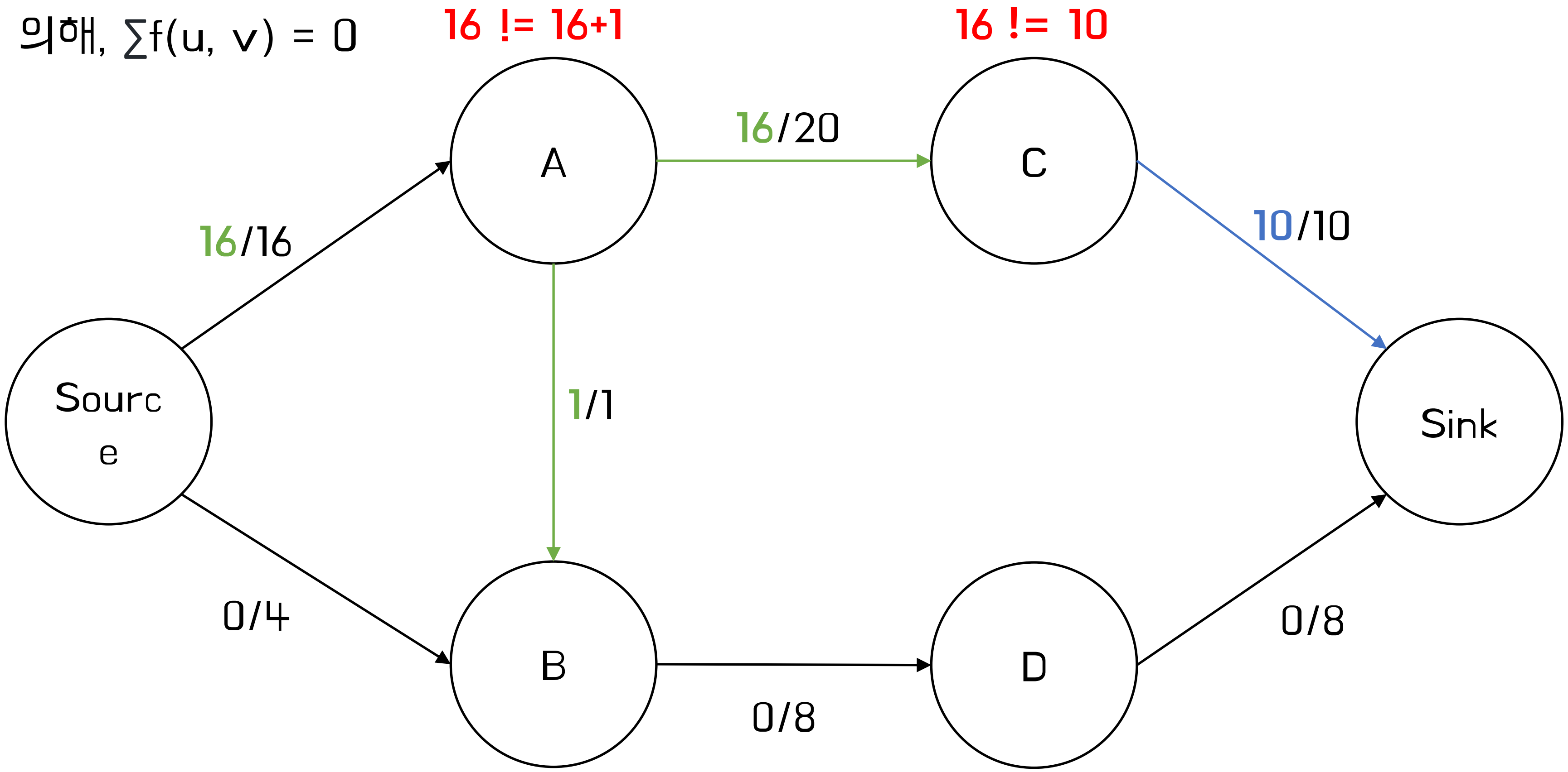
속성

4. 유량 보존의 법칙

: Source와 Sink를 제외한 모든 각 정점에 대해서 들어오는 유량과 나가는 유량의 양은 같다.

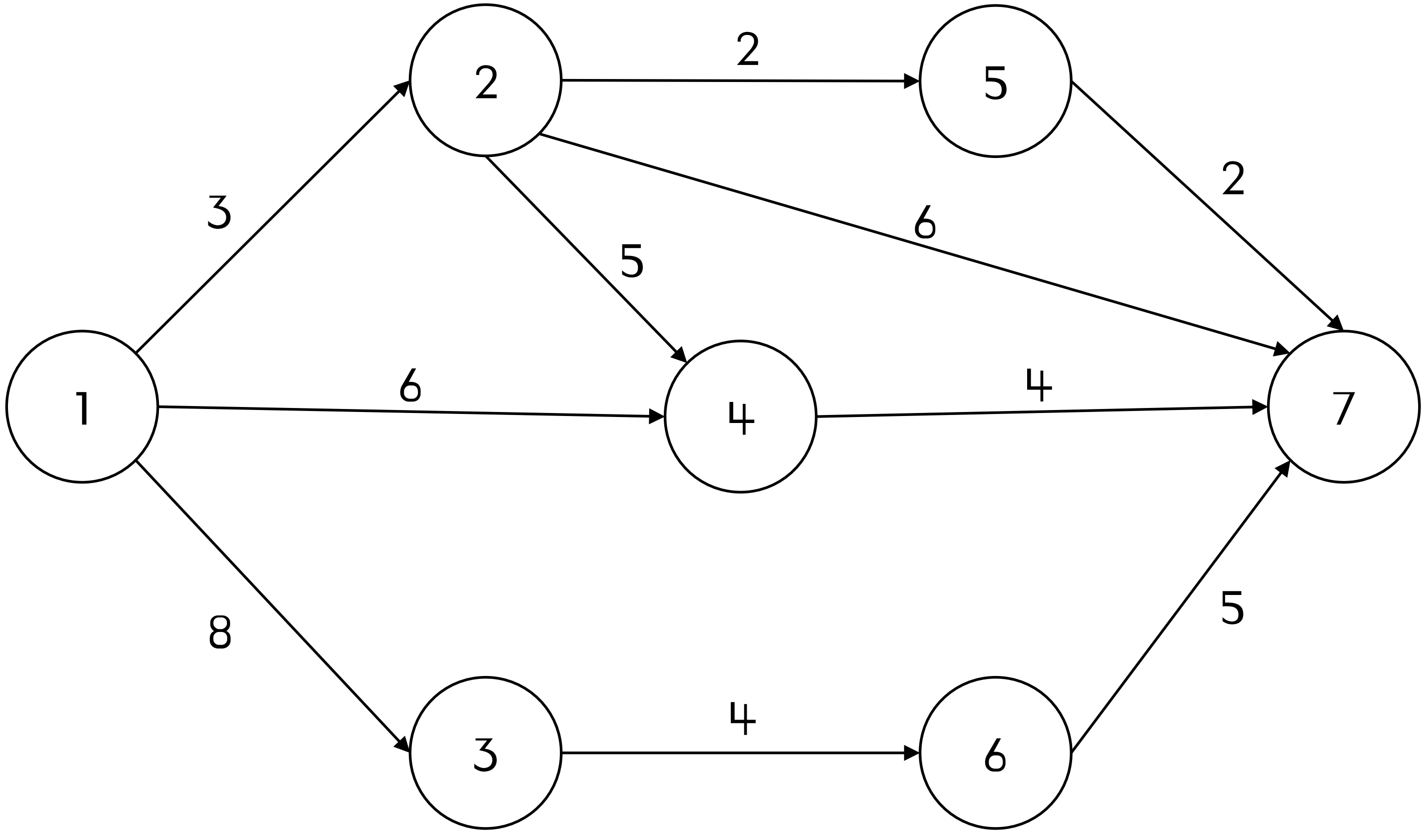
유량의 대칭성(속성3)에 의해, $\sum f(u, v) = 0$

유량 보존의 법칙 지키지 않음!



포드-풀커슨(Ford-Fulkerson) 방법

- 잔여 네트워크에서 증가 경로가 더 이상 존재하지 않을 때까지 유량을 흘려보내는 방식

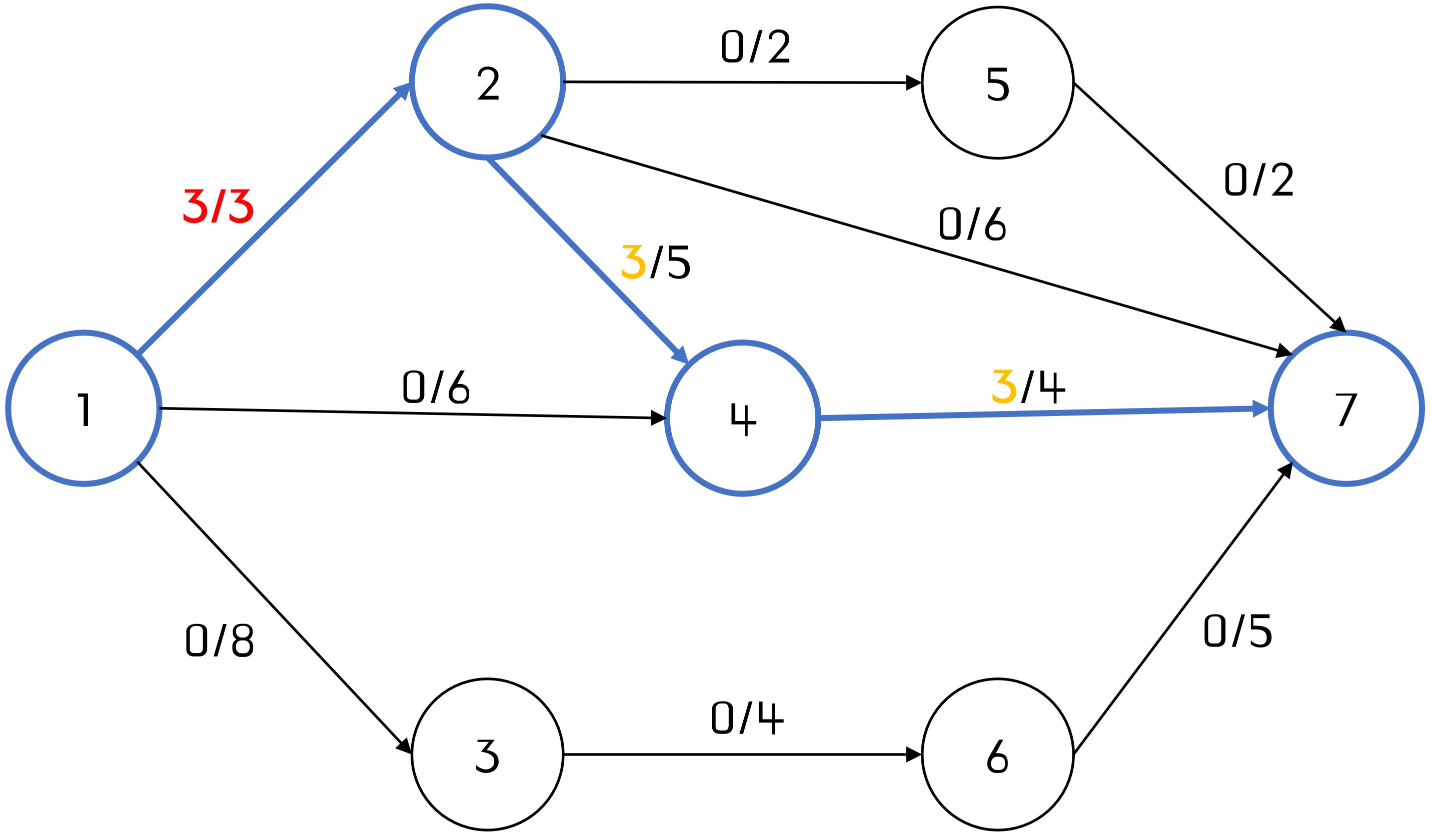


Source : 1번 정점
Sink : 7번 정점

➢ 최대 유량 = 11

포드-풀커슨(Ford-Fulkerson) 방법

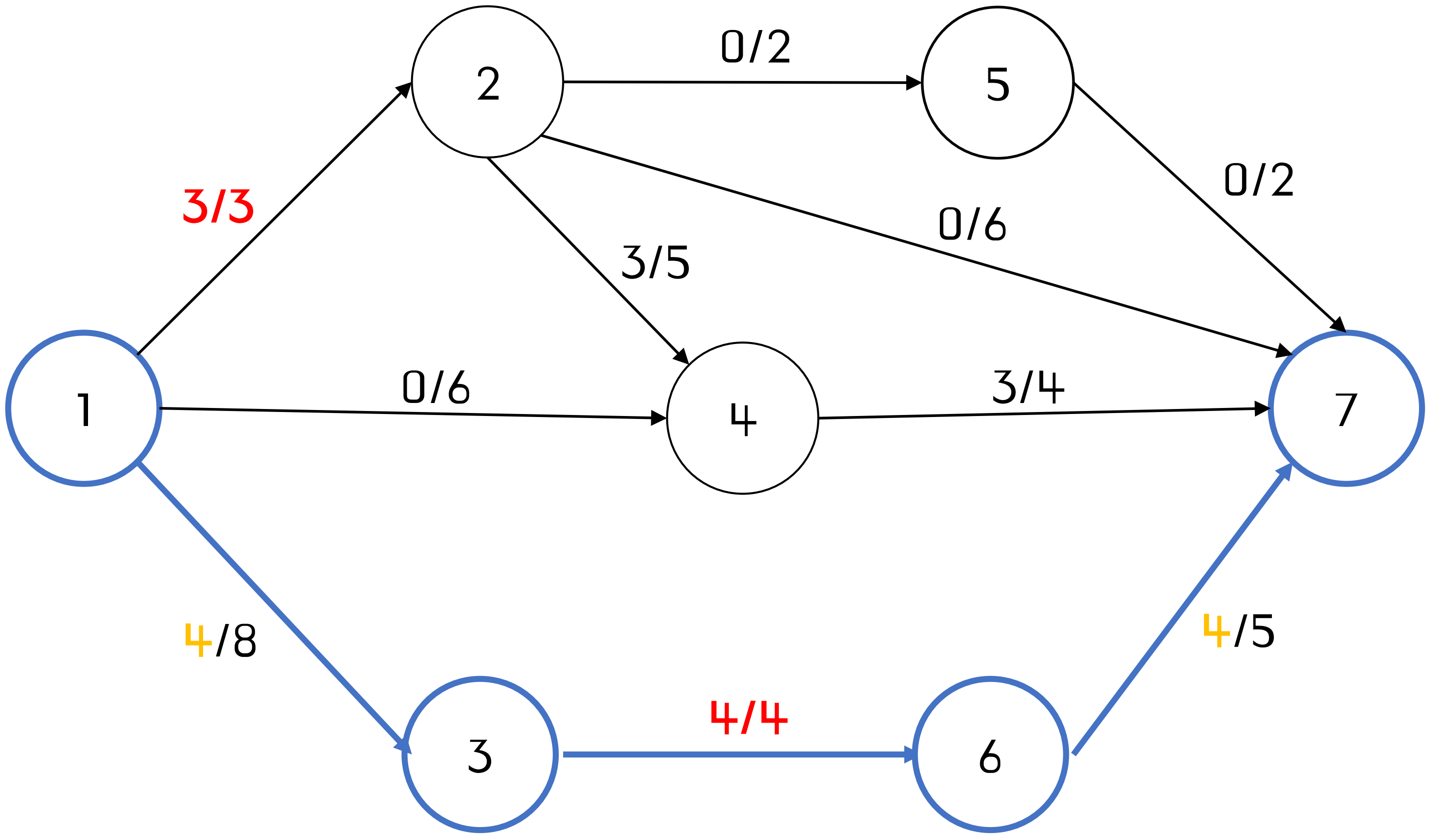
- 증가경로 1->2->4->7 경우



- ➢ 최대 3의 유량 흐를 수 있음
- ➢ 간선 1-2의 유량이 용량에 도달했으므로 더 이상 간선 1-2 포함 불가

포드-풀커슨(Ford-Fulkerson) 방법

- 증가경로 1->3->6->7 경우

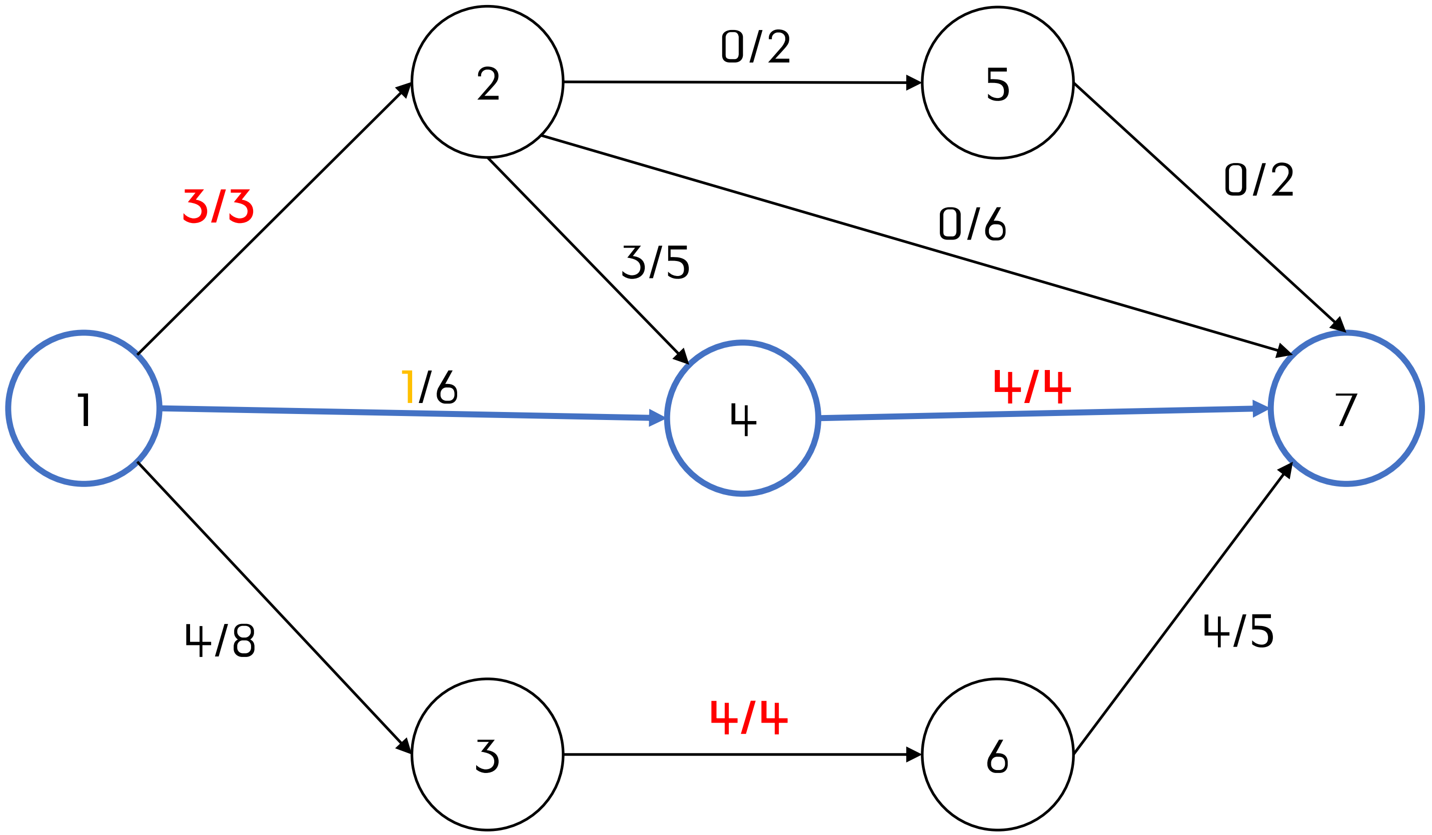


➢ 최대 4의 유량 흐를 수 있음

➢ 더 이상 간선 3-6 포함 불가

포드-풀커슨(Ford-Fulkerson) 방법

- 증가경로 1->4->7 경우



➢ 최대 1의 유량 흐를 수 있음

➢ 더 이상 간선 4-7 포함 불가

➢ 지금까지의 유량 = 8

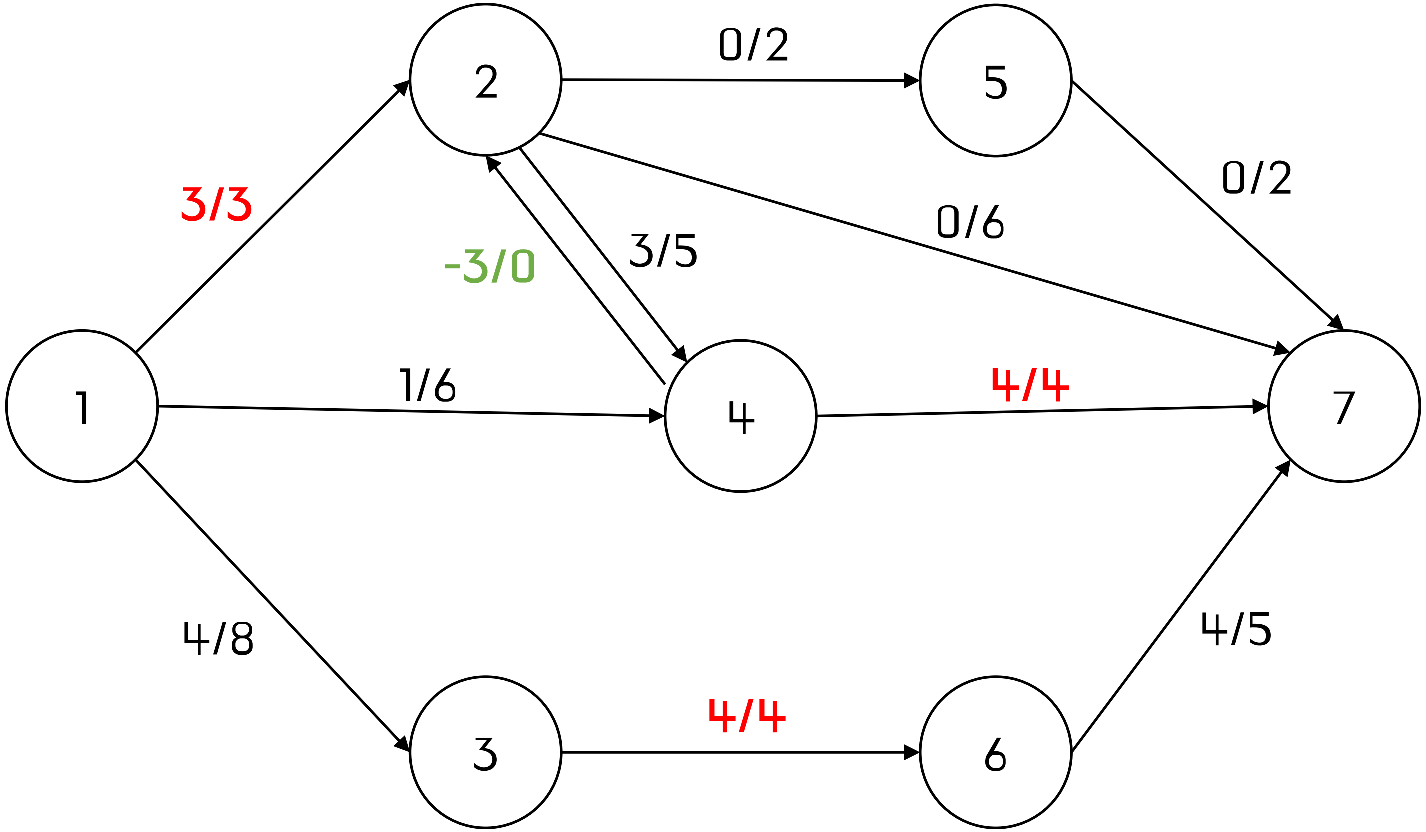
왜 결과는 11?

=> 음의 유량 계산

(유량의 대칭성)

01

포드-풀커슨(Ford-Fulkerson) 방법

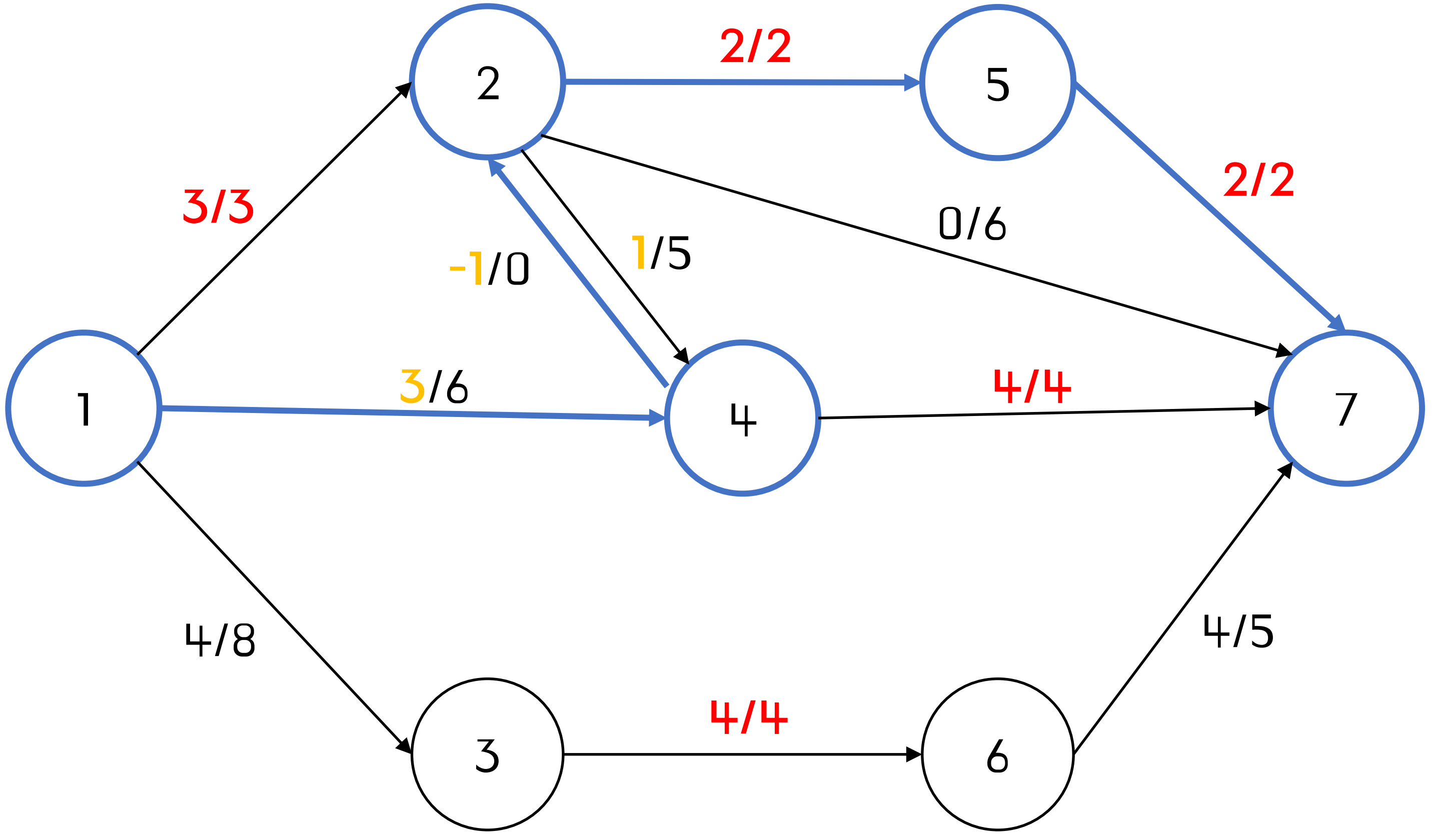


$\boxed{f(v, u) = -f(u, v)}$

➢ $r(4, 3) = c(4, 3) - f(4, 3)$
$= 0 - (-3) = 3$

포드-풀커슨(Ford-Fulkerson) 방법

- 증가경로 1->4->2->5->7 경우



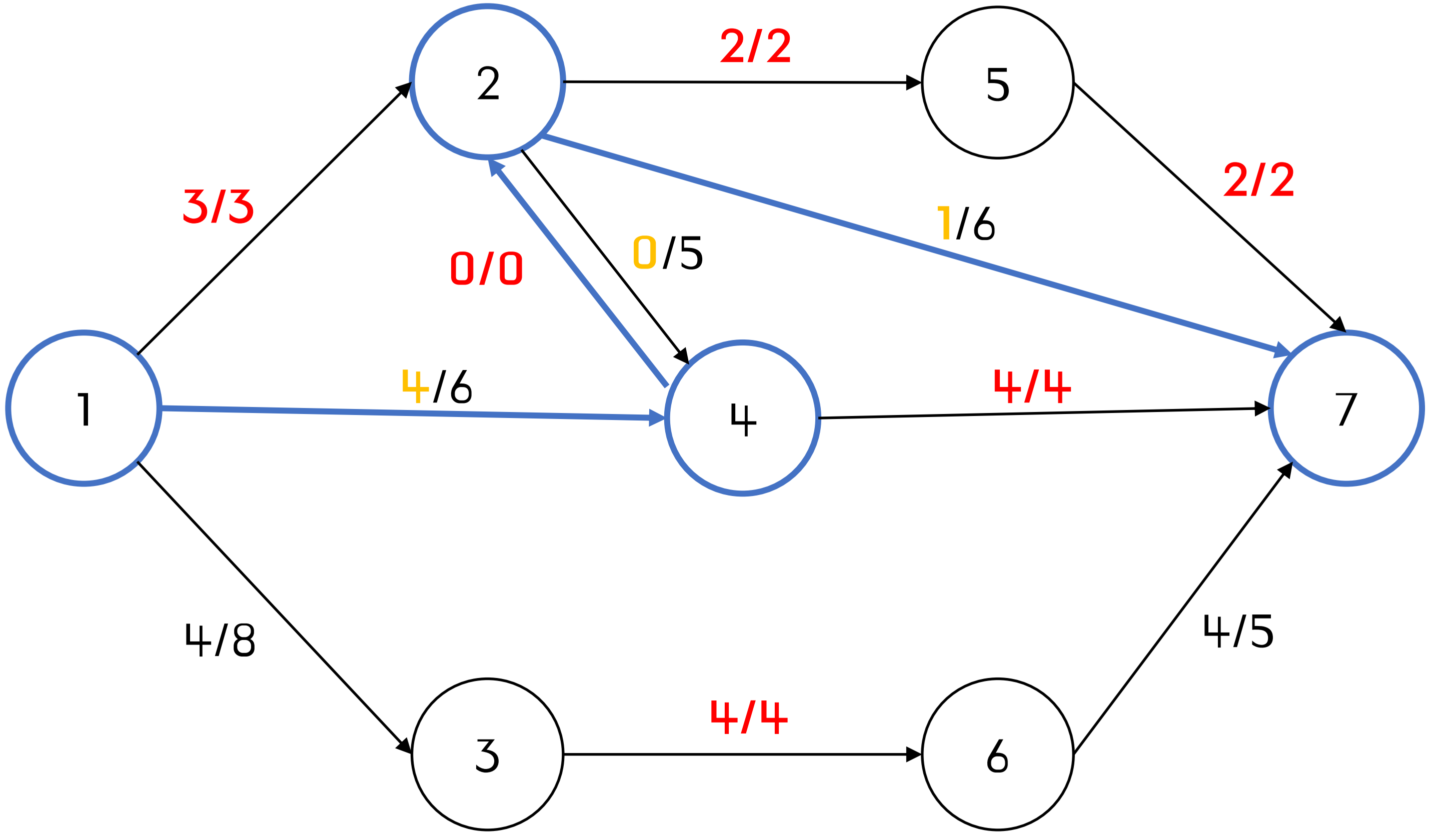
➢ 최대 2의 유량 흐를 수 있음

➢ 더 이상 간선 2-5, 5-7 포함 불가

➢ 역방향 간선 : +유량
방향 간선 : -유량

포드-풀커슨(Ford-Fulkerson) 방법

- 증가경로 1->4->2->7 경우



➢ 최대 1의 유량 흐를 수 있음

➢ 더 이상 가능한 증가경로 없음

➢ 최대 유량 = 11

포드-풀커슨(Ford-Fulkerson) 방법

- 잔여 네트워크에서 증가 경로가 더 이상 존재하지 않을 때까지 유량을 흘려 보내는 방식

- 진행 과정

1. 네트워크에 존재하는 모든 간선(역방향 포함)의 유량을 0으로 초기화
2. source에서 sink로 갈 수 있는, 잔여 용량이 남은 경로를 DFS로 탐색
3. 해당 경로에 존재하는 간선들의 잔여 용량 중 가장 작은 값을 유량으로 흘려보냄
4. 해당 유량에 음수 값을 취해, 역방향 간선에도 흘려보냄 (유량 상쇄)
5. 더 이상 잔여 용량이 남은 경로가 존재하지 않을 때까지 반복

에드몬드-카프 알고리즘 (Edmonds-Karp Algorithm)

- 포드-풀커슨 방법을 BFS로 구현한 알고리즘

- BFS 사용하는 이유

  α. 시간 복잡도

  >> DFS

  : O((|E| + |V|) * F)

  => 스택 오버플로우 발생 가능

  >> BFS

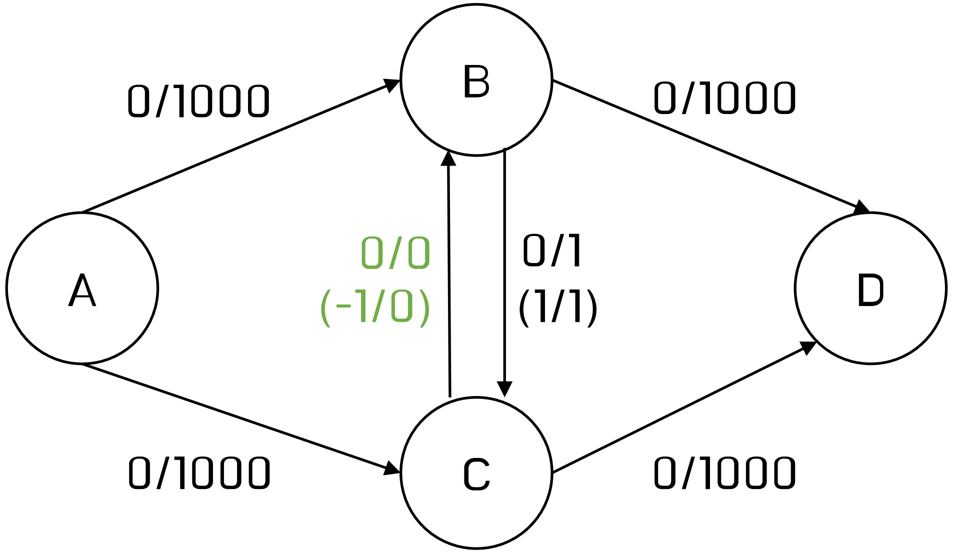
  : O(|V| * (|E| ^ 2))

- E : 간선
- V : 정점
- F : 최대 유량

에드몬드-카프 알고리즘 (Edmonds-Karp Algorithm)

b. 가장 짧은 경로의 증가 경로 탐색

: 최단거리로 최대의 유량을 보냄 -> 중간에 용량이 1인 간선 끼어 있어도 돌아가는 길이면 무시



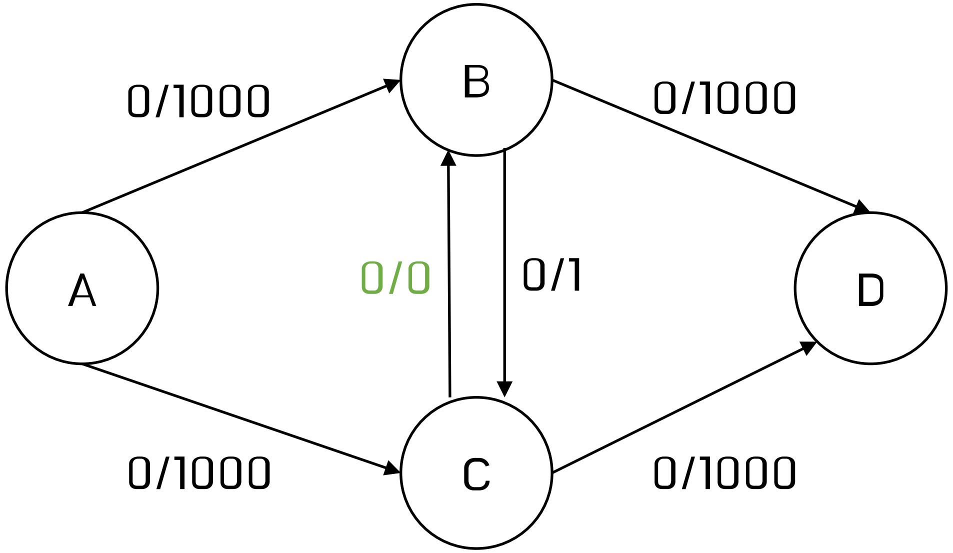
>> DFS 사용 시

- A->B->C->D : 1의 유량 보냄
- A->C->B->D : 1의 유량 보냄(역간선)
- A->B->C->D : 1의 유량 보냄
- A->C->B->D : 1의 유량 보냄(역간선)
- 반복...
- => 2000번의 탐색으로 최대 유량 발견

에드몬드-카프 알고리즘 (Edmonds-Karp Algorithm)

b. 가장 짧은 경로의 증가 경로 탐색

: 최단거리로 최대의 유량을 보냄 -> 중간에 용량이 1인 간선 끼어 있어도 돌아가는 길이면 무시



>> BFS 사용 시

- A->B->D : 1000의 유량 보냄
- A->C->D : 1000의 유량 보냄
- => 2번의 탐색으로 최대 유량 발견

에드몬드-카프 알고리즘 (Edmonds-Karp Algorithm)

- 진행 과정

1. 네트워크에 존재하는 모든 간선(역방향 포함)의 유량을 0으로 초기화
2. source에서 sink로 갈 수 있는, 잔여 용량이 남은 경로를 BFS로 탐색
3. 해당 경로에 존재하는 간선들의 잔여 용량 중 가장 작은 값을 유량으로 흘려보냄
4. 해당 유량에 음수 값을 취해, 역방향 간선에도 흘려보냄 (유량 상쇄)
5. 더 이상 잔여 용량이 남은 경로가 존재하지 않을 때까지 반복

01

에드몬드-카프 알고리즘 (Edmonds-Karp Algorithm)

```
vector <int> v[MAX];
int cap[MAX][MAX] = {0,};   // 용량
int flow[MAX][MAX] = {0,};  // 현재 유량
int pre[MAX] = {0,};        // 현재 탐색 중인 증가 경로에서 이전 정점을 저장하여 경로를 기억하는 역할
int total_flow = 0;         // 최대 유량

void maxFlow(int source, int sink){
    memset(flow, 0, sizeof(flow));

    while (1){
        queue<int> q;
        memset(pre, -1, sizeof(pre));

        q.push(source);
        pre[source] = source;

        while (!q.empty()){
            int now = q.front();
            q.pop();

            if (now == sink)
                break;

            for (int i=0; i<v[now].size(); i++){
                int next = v[now][i];

                // 방문하지 않은 정점 중 용량이 남는 경우
                if (cap[now][next] - flow[now][next] > 0 && pre[next] == -1){
                    q.push(next);
                    pre[next] = now;
                }
            }
        }
```

```
        // 더 이상 증가 경로 없음
        if (pre[sink] == -1)
            break;

        // 증가 경로로 새로 흘려줄 유량 = 경로 중 최소 잔여 용량
        int min_flow = INF;
        for (int i=sink; i!=source; i=pre[i]){
            int j = pre[i];
            min_flow = min(cap[j][i] - flow[j][i], min_flow);
        }

        // 증가 경로는 유량 증가, 역방향 경로는 유량 감소
        for (int i=sink; i!=source; i=pre[i]){
            int j = pre[i];
            flow[j][i] += min_flow;
            flow[i][j] -= min_flow;
        }

        total_flow += min_flow;
    }
    return;
}
```

02 MCMF

MCMF (Minimum Cost Maximum Flow)

- 최소 비용 최대 유량 문제
- 비용을 최소로 하는, source에서 sink까지 흐를 수 있는 최대 용량을 구하는 문제
- 음의 가중치가 있는 그래프에서도 잘 작동해야 함
- SPFA를 사용하여 에드몬드-카프 알고리즘보다 빠르게 최대 유량 탐색 가능
- 시간복잡도 : O(VEF)

- SPFA (Shortest Path Faster Algorithm)
  - 벨만-포드 알고리즘의 변형
  - 음의 가중치가 있는 그래프에서 최단경로를 빠르게 구할 수 있다.
  - 최단거리 갱신 시킬 간선들의 큐가 있고, 갱신된 점과 연결된 간선들이 큐에 없다면 넣는 것
  - 이를 반복하면 최단 경로 계산이 완료되었을 때 큐가 비게 된다.
  - isInQ 배열 : 큐에 해당 정점이 들어가 있는지 여부 저장

MCMF (Minimum Cost Maximum Flow)

```cpp
void mcmf(int source, int sink){
    memset(flow, 0, sizeof(flow));

    while (1){
        queue<int> q;
        bool isInQ[MAX];

        memset(pre, -1, sizeof(pre));
        memset(isInQ, false, sizeof(isInQ));

        // dist 배열 초기화
        for (int i=0; i<MAX; i++)
            dist[i] = INF;

        q.push(source);
        pre[source] = source;
        dist[source] = 0;
        isInQ[source] = true;

        while (!q.empty()){
            int now = q.front();
            q.pop();
            isInQ[now] = false;

            for (int i = 0; i < v[now].size(); i++){
                int next = v[now][i];

                // 방문하지 않은 정점 중 용량이 남는 경우
                if (cap[now][next] - flow[now][next] > 0 && dist[now] + cost[now][next] < dist[next]){
                    pre[next] = now;
                    dist[next] = dist[now] + cost[now][next];

                    if (!isInQ[next]){
                        q.push(next);
                        isInQ[next] = true;
                    }
                }
            }
        }
```

```cpp
vector <int> v[MAX];
int cap[MAX][MAX] = {0,};   // 용량
int flow[MAX][MAX] = {0,};  // 현재 유량
int pre[MAX];               // 경로를 기억하는 역할
int cost[MAX][MAX];         // 노드에서 노드로 이동하는데 필요한 비용
int dist[MAX];              // source 노드에서 해당 번호의 노드로 이동하는데 필요한 비용

int costSum = 0;            // 최소 비용
int totalFlow = 0;          // 최대 유량
```

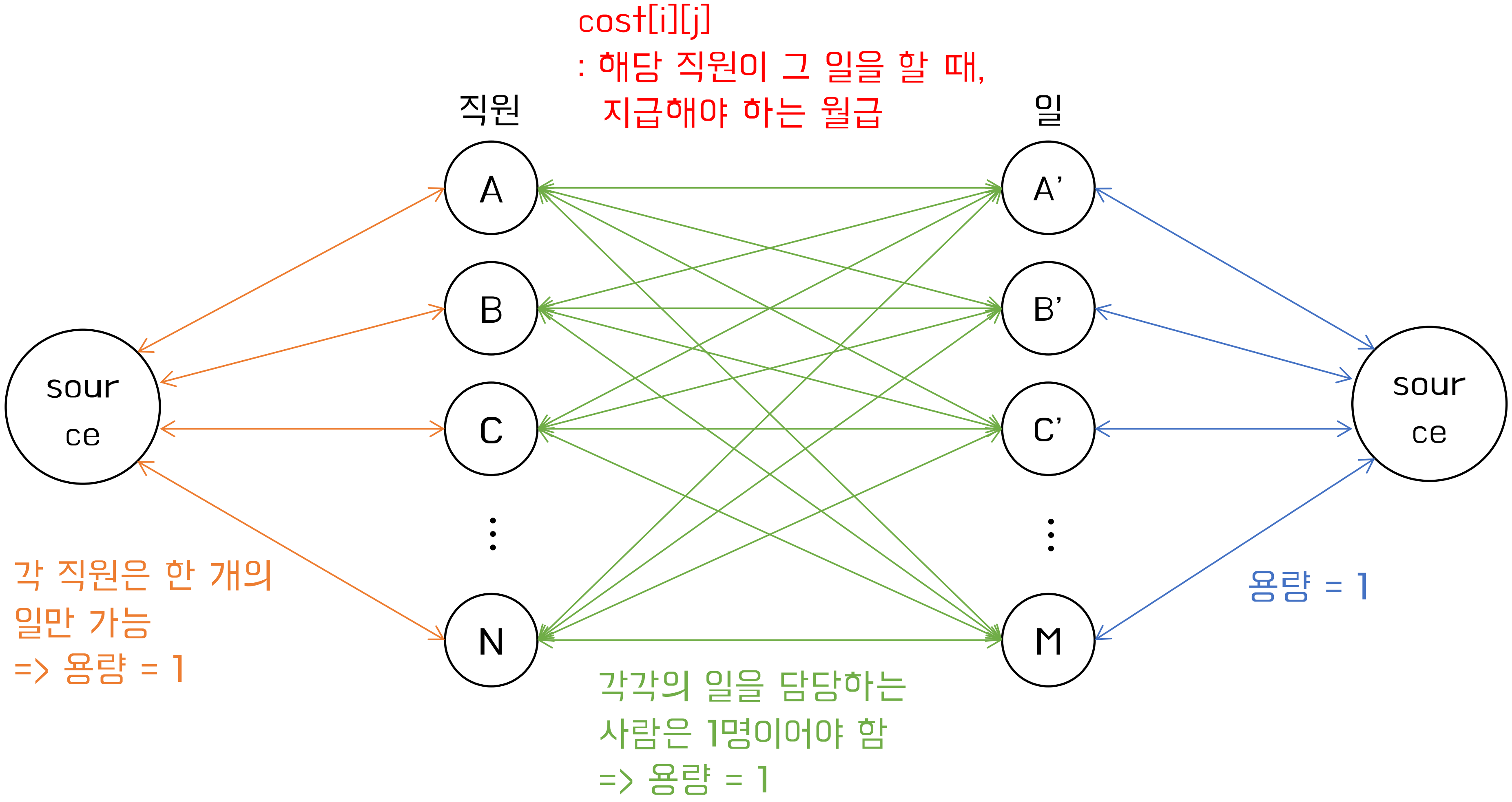
```cpp
        // 더 이상 증가 경로 없음
        if (pre[sink] == -1)
            break;

        // 증가 경로로 새로 흘려줄 유량 = 경로 중 최소 잔여 용량
        int minFlow = INF;
        for (int i = sink; i != source; i = pre[i]){
            int j = pre[i];
            minFlow = min(cap[j][i] - flow[j][i], minFlow);
        }

        // 증가 경로는 유량 증가, 역방향 경로는 유량 감소
        for (int i = sink; i != source; i = pre[i]){
            int j = pre[i];
            flow[j][i] += minFlow;
            flow[i][j] -= minFlow;
            costSum += minFlow * cost[j][i];
        }
        totalFlow += minFlow;
    }
    return;
}
```

03 추천 문제

추천 문제

[G3] 6086. 최대 유량
[P4] 2316. 도시 왕복하기 1
[P3] 11377. 열혈강호 3
[P3] 11408. 열혈강호 5
[P3] 11405. 책 구매하기

참고 (*열혈강호 5)



cost[i][j]
: 해당 직원이 그 일을 할 때,
지급해야 하는 월급
직원
일
A
B
C
N
A'
B'
C'
M
sour
ce
sour
ce
각 직원은 한 개의
일만 가능
=> 용량 = 1
각각의 일을 담당하는
사람은 1명이어야 함
=> 용량 = 1
용량 = 1

감사합니다.