Network Flow 1

나정휘

https://justiceHui.github.io/

목차

- Network Flow
- Ford-Fulkerson Method
- Max-Flow Min-Cut Theorem
- Edmonds-Karp Algorithm
- Bipartite Matching
- Konig's Theorem
- Dinic's Algorithm
- Min Cost Max Flow
- Circulation
- Push Relabel Algorithm
- Cost Scaling Algorithm

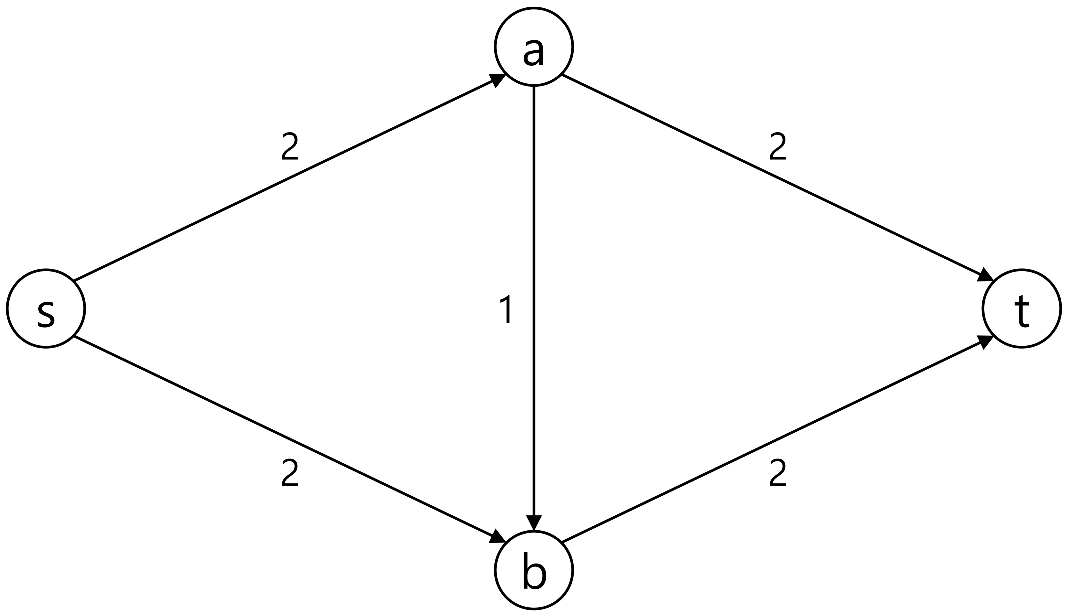
Network Flow

Network Flow

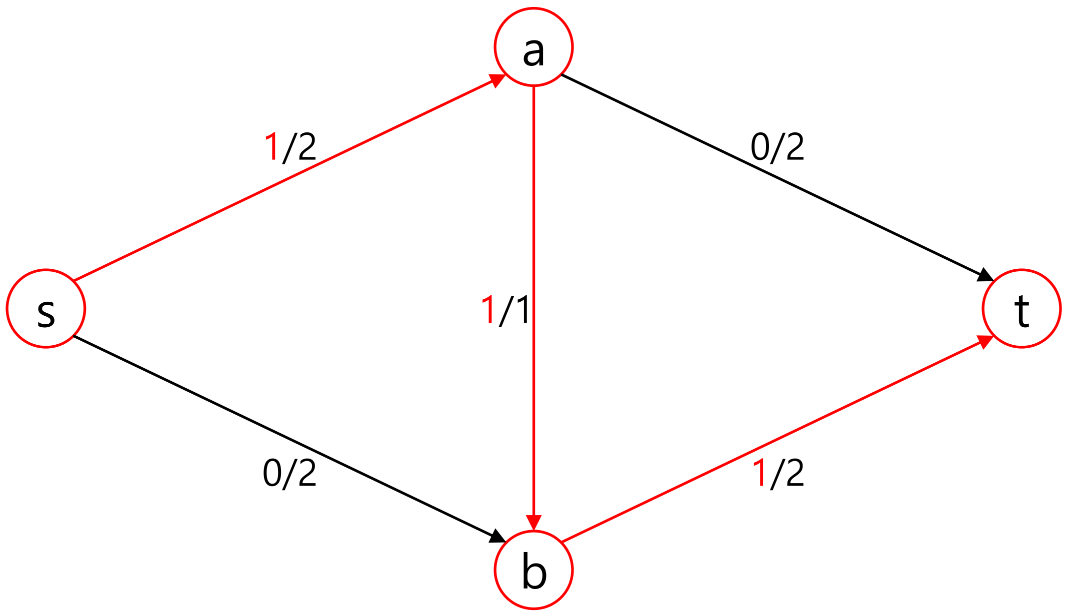
- Flow Network
  - 각 간선마다 용량 $c : E \to \mathbb{R}_+$이 있는 방향 그래프 $G = (V, E)$를 "유량 네트워크"라고 부름
  - 편의상 각 간선 $e \in E$에는 용량이 0인 역방향 간선 e^R이 존재한다고 하자.
- Flow
  - 유량 네트워크의 두 정점 $s, t \in V$가 주어짐
  - 아래 세 조건을 만족하는 함수 $f : E \to \mathbb{R}_+$를 "유량"이라고 부름
    - $f(e) \leq c(e)$
    - $f(e^R) = -f(e)$
    - 모든 $v \in V \setminus \{s, t\}$에 대해 $\sum_{(u,v)} f(u, v) = \sum_{(v,u)} f(v, u)$
      - source와 sink를 제외한 모든 정점에 대해, 들어오는 유량과 나가는 유량의 합이 동일
  - 유량 f의 크기는 $|f| = \sum_v f(v, t)$ 로 정의
    - sink로 들어가는 유량 = source에서 나가는 유량

Network Flow

- Flow Network

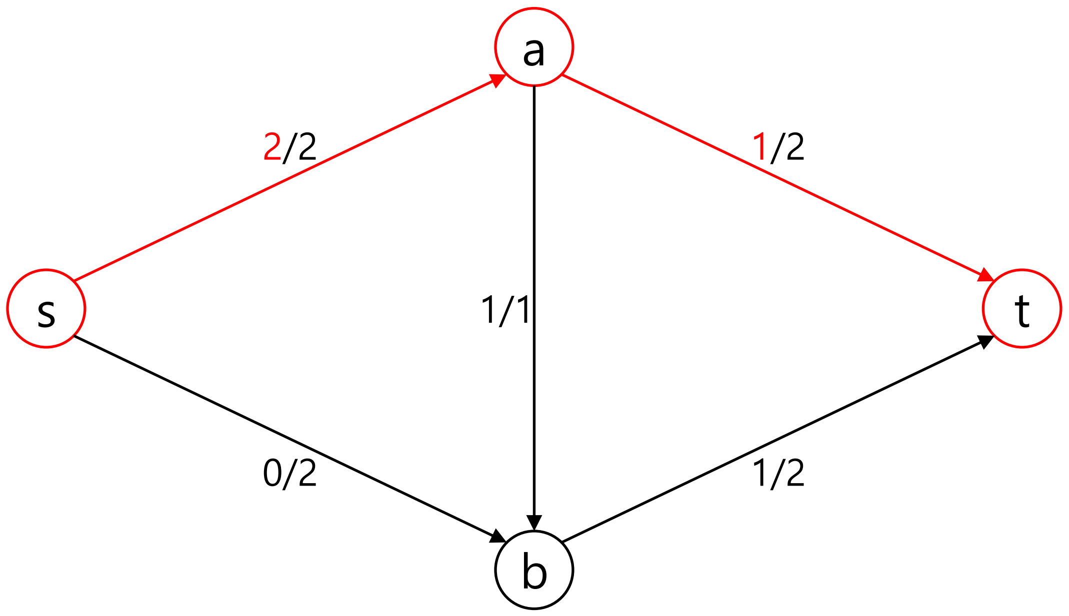


- Flow, |f| = 1

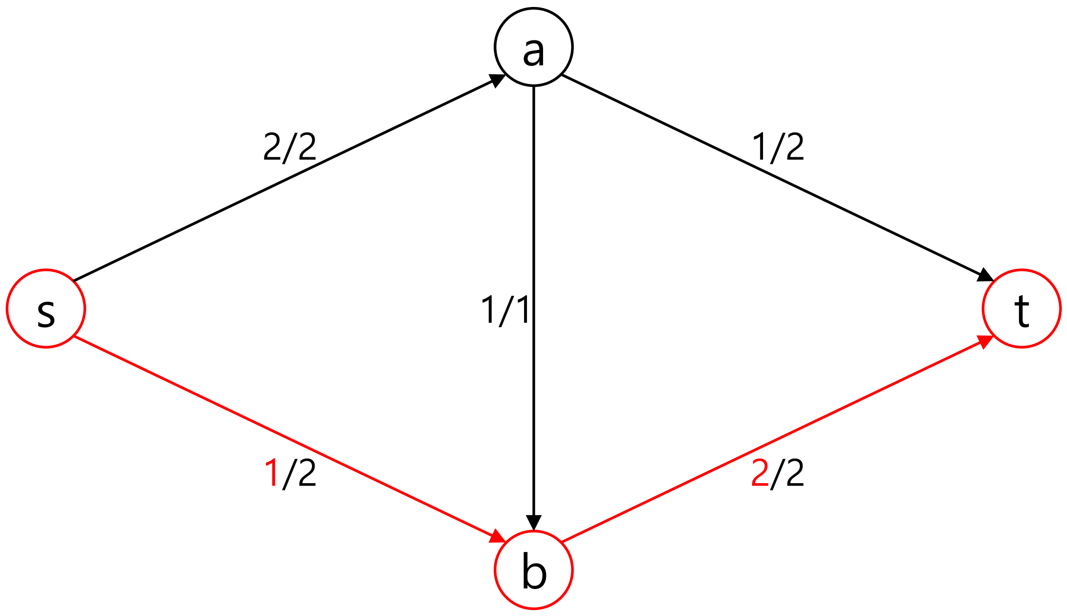


Network Flow

- Flow, |f| = 2

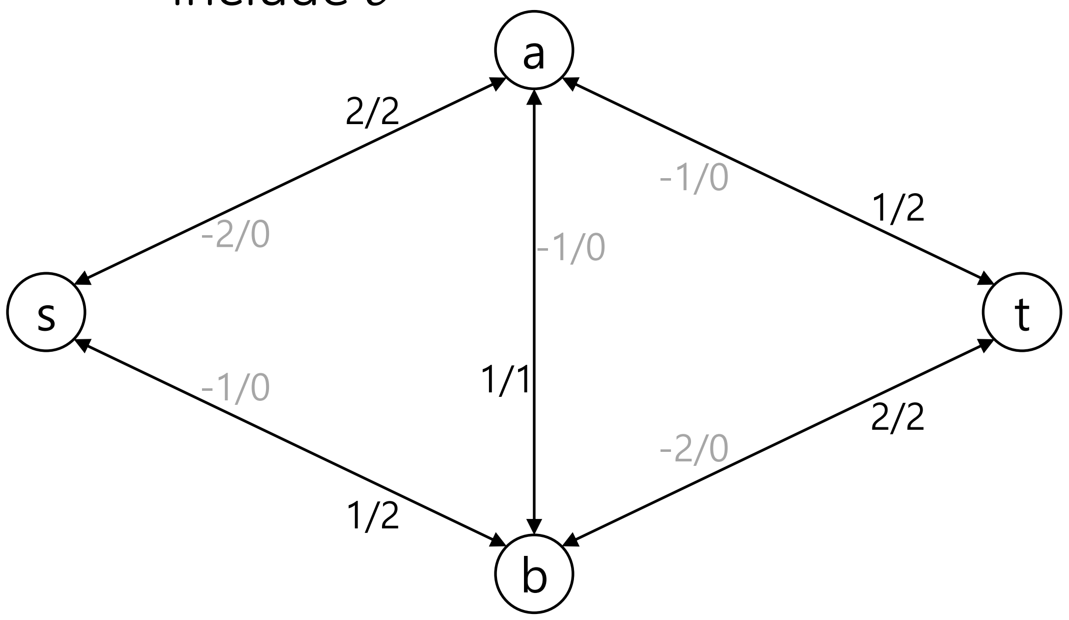


- Flow, |f| = 3

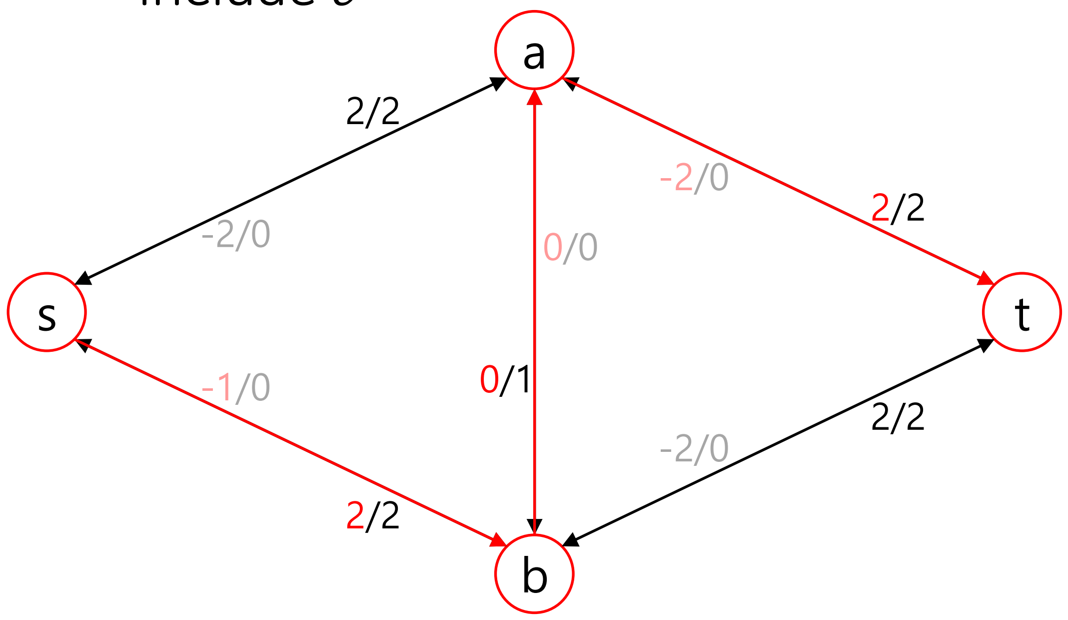


Network Flow

- Flow, |f| = 3
  - include e^R



- Flow, |f| = 4
  - include e^R

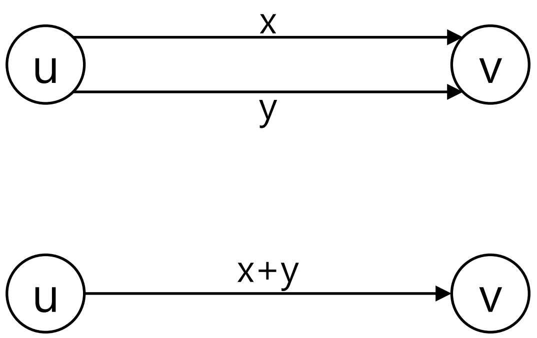


Network Flow

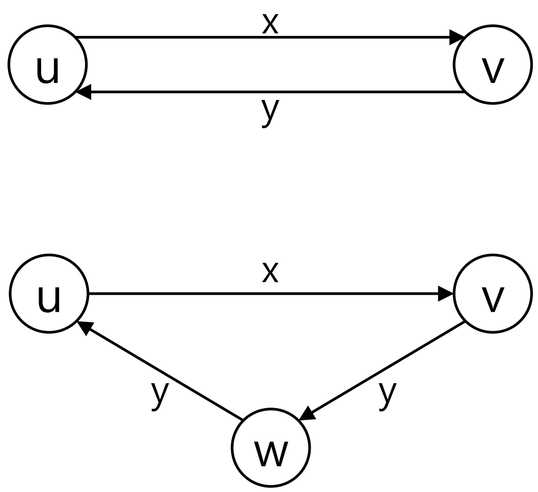
- Maximum Flow
  - 유량 네트워크 $G = (V, E, c)$와 정점 s, t가 주어짐
  - 크기가 최대인 유량 f를 구하는 문제
- Minimum Cost Maximum Flow
  - 유량 네트워크 $G = (V, E, c)$, 간선의 가중치 $w : E \to \mathbb{R}$, 정점 s, t가 주어짐
  - 크기가 최대이면서 $\sum_{(u,v)} f(u, v) \times w(u, v)$가 최소인 유량 f를 구하는 문제
    - 유량의 비용을 최소화하는 문제
- Assume simple graph
  - self loop, parallel edge가 없는 상황만 생각해도 된다.
    - self loop: 유량의 크기에 변화를 주지 않음
    - parallel edge: 적절한 처리를 통해 없앨 수 있음
    - 따라서 $e \in E$이면 $e^R \notin E$

Network Flow

- Parallel edge



- Anti parallel edge

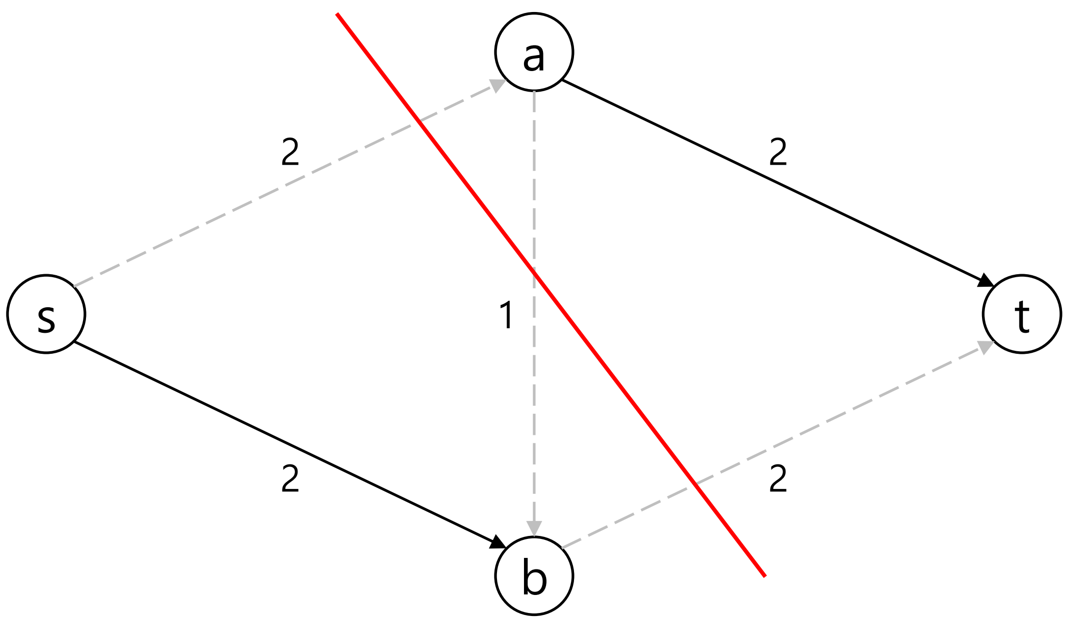


Network Flow

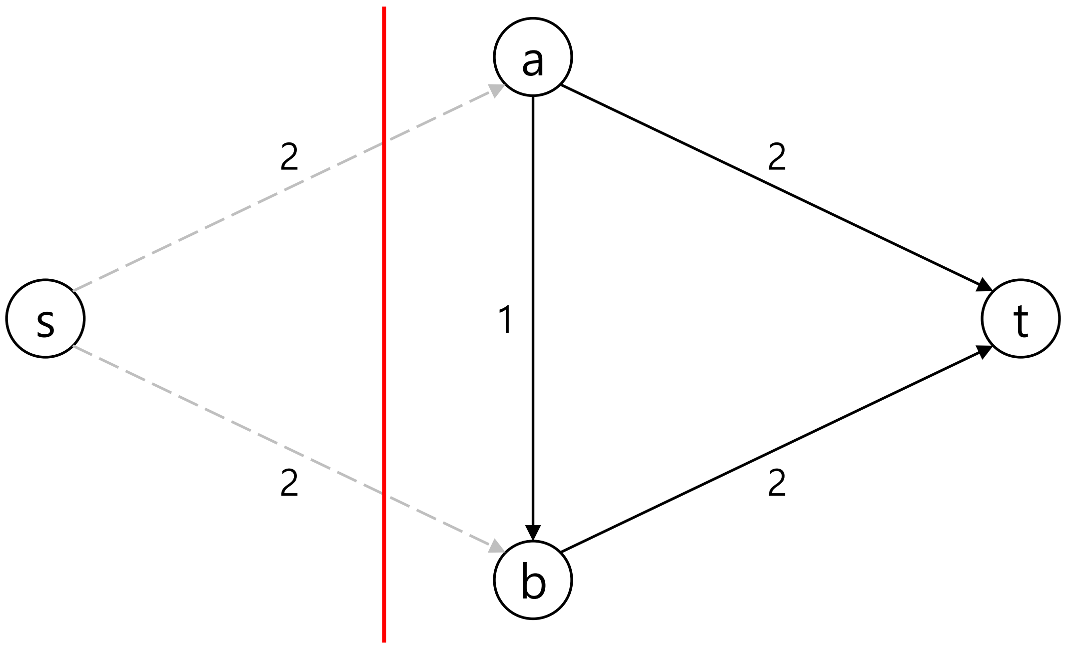
- Minimum Cut
  - 가중치 그래프 $G = (V, E, w)$와 정점 s, t가 주어짐
  - 정점의 부분 집합 $S \subset V$가 $s \in S, t \in V\backslash S$를 만족하면 $(S, V\backslash S)$를 “절단”이라고 부름
  - 절단 $(S, V\backslash S)$의 가중치: $c(S) = \sum_{u \in S, v \in V\backslash S} w(u, v)$
    - S에서 $V\backslash S$로 가는 간선의 가중치 합
  - 가중치가 최소인 절단을 “최소 절단”이라고 부름
    - s와 t가 연결되지 않도록 최소 비용으로 간선을 제거하는 문제
    - 유량과 관련 없어 보이지만…

Network Flow

- Cut (cost = 5)



- Minimum Cut (cost = 4)



Network Flow

- Network Flow 1에서 다루는 내용
  - Ford-Fulkerson Method / Edmonds-Karp Algorithm
    - 최대 유량을 $O(VE^2)$에 구하는 방법
  - Max-Flow Min-Cut Theorem
    - 최대 유량 = 최소 절단이라는 사실
- Network Flow 2에서 다루는 내용
  - Bipartite Matching
    - 이분 그래프의 최대 매칭을 $O(VE)$에 구하는 방법
    - DAG의 최소 경로 덮개를 $O(VE)$에 구하는 방법
  - Konig's Theorem, Dilworth's Theorem
    - 이분 그래프의 최소 정점 커버, 최대 독립 집합, 최대 반사슬을 $O(VE)$에 구하는 방법

Network Flow

- Network Flow 3에서 다루는 내용
  - Dinic's Algorithm
    - 최대 유량을 $O(V^2E)$에 구하는 방법
  - Minimum Cost Maximum Flow
    - 최소 비용 최대 유량을 $O(VEf)$에 구하는 방법 (Successive Shortest Path Algorithm)
- 다루지 않지만 고인물들은 알고 있는 내용
  - Circulation
    - 네트워크 플로우의 일반화 버전, 정점마다 demand/supply가 존재
    - 각 정점마다 $d : V \to \mathbb{R}$이 있어서, $\sum_{(u,v)} f(u,v) - \sum_{(v,u)} f(v,u) = d(v)$를 만족해야 함
  - Push Relabel Algorithm
    - 최대 유량을 $O(V^3)$ 또는 $O(V^2\sqrt{E})$에 구하는 방법
  - Cost Scaling Algorithm
    - 최소 비용 최대 유량을 $O(N^3 \log NC)$에 구하는 방법

질문?

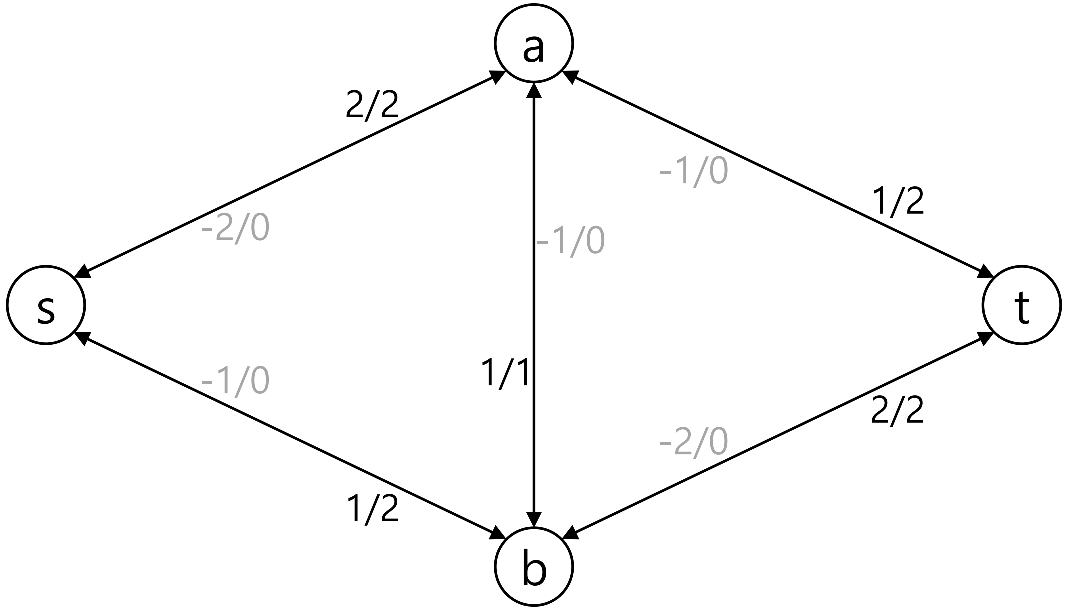
Ford-Fulkerson Method

Ford-Fulkerson Method

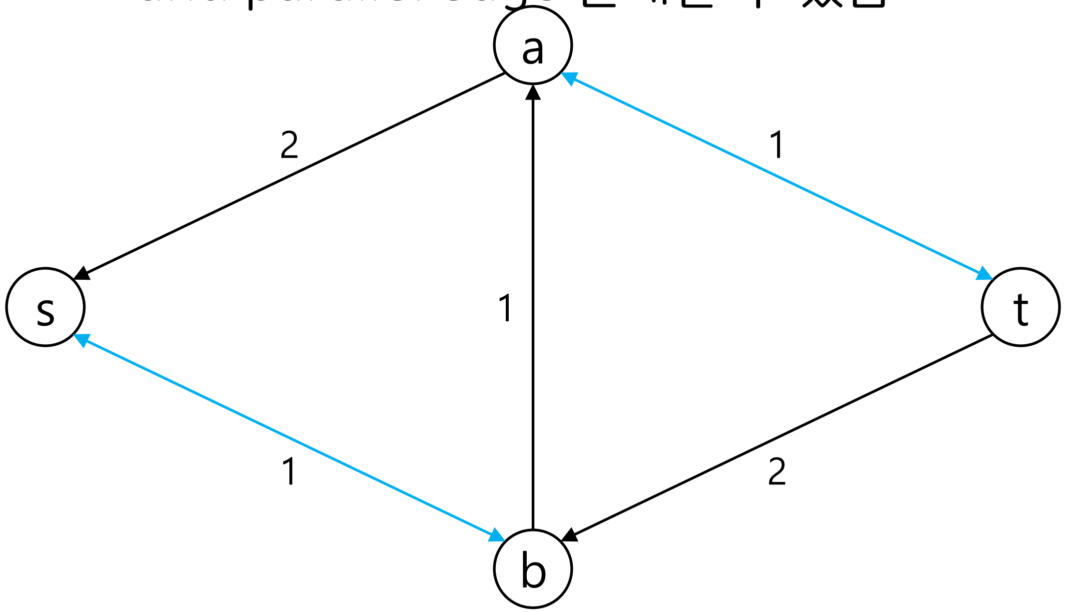
- Residual Network
  - 잔여 그래프 $G_f = (V, E_f, c_f)$: 유량을 f 만큼 보냈을 때, 용량이 남은 간선들로 구성된 그래프
    - $E_f = \{e; e \in E, f(e) < c(e)\} \cup \{e; e^R \in E, f(e^R) > 0\}$
    - $c_f(e) = \begin{cases} c(e) - f(e), e \in E \\ f(e^R), \qquad e^R \in E' \end{cases}$ 따라서 $c_f(e)$는 항상 양수임을 알 수 있음
- Augmenting Path
  - 잔여 그래프에 있는 임의의 $s - t$단순 경로를 "증가 경로"라고 부름
    - 증가 경로 P를 구성하는 간선을 $e_1, e_2, \cdots, e_k$라고 하면
    - P를 따라서 유량을 $f_P = \min\{c_f(e_1), c_f(e_2), \cdots, c_f(e_k)\}$ 만큼 흘릴 수 있음
    - 이때 유량의 크기는 $f_p > 0$ 만큼 증가함
  - 만약 f가 최대 유량이면 G_f에 $s - t$경로가 존재하지 않음

Ford-Fulkerson Method

- Flow, |f| = 3
  - include e^R

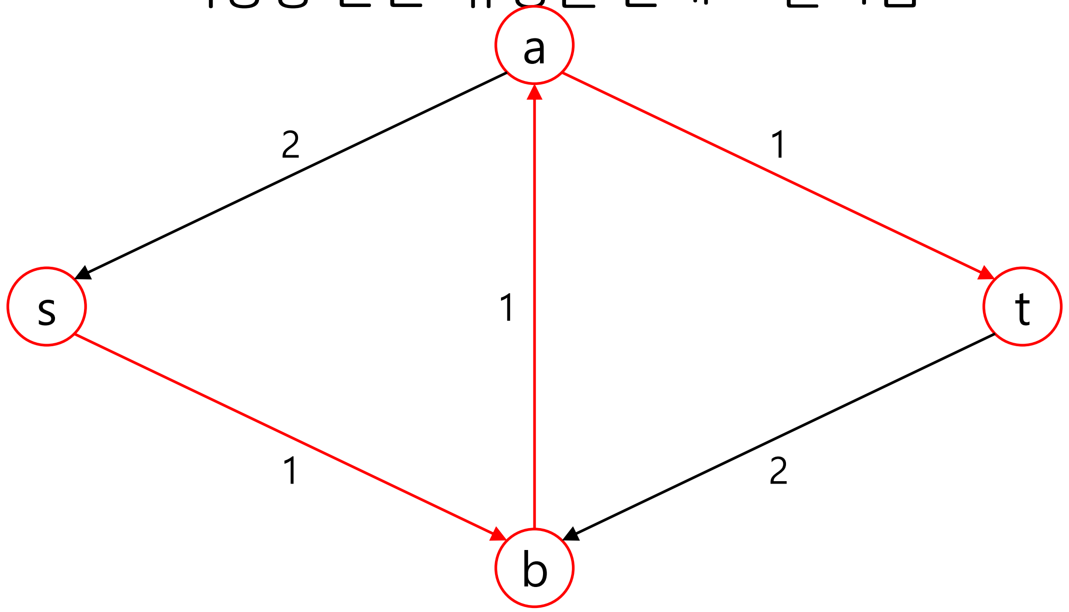


- Residual Graph
  - anti parallel edge 존재할 수 있음

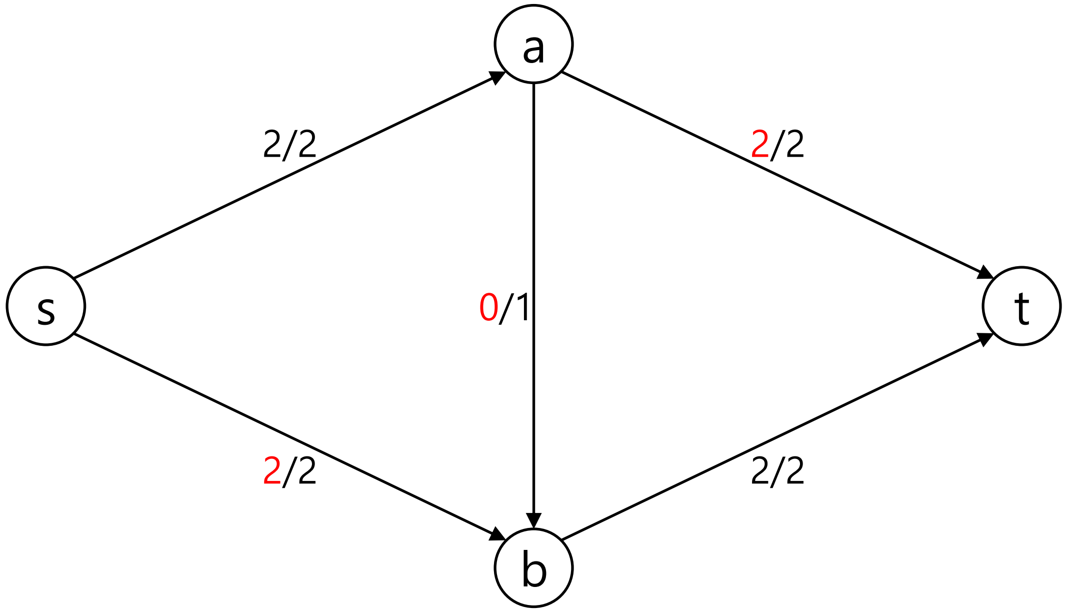


Ford-Fulkerson Method

- Augmenting Path
  - 역방향 간선: 유량을 반대로 밀어냄



- Flow, |f| = 4



Ford-Fulkerson Method

- Ford-Fulkerson Method
  - $f \leftarrow 0$
  - while G_f에 $s - t$단순 경로 P가 존재하면
    - $f \leftarrow f + f_P$
  - f는 최대 유량
- 이게 진짜 될까?
  - 알고리즘은 항상 종료함
    - 매번 $|f|$가 최소 1씩 증가하고 최대 유량은 유한하므로 종료함
  - f가 최대 유량이면 G_f에 증가 경로가 존재하지 않음
    - 이미 증명함
  - G_f에 증가 경로가 존재하지 않으면 f는 최대 유량
    - 이걸 증명해 보자.

질문?

Max-Flow Min-Cut Theorem

Max-Flow Min-Cut Theorem

- 지금까지 알아낸 것
  - f가 최대 유량이면 G_f에 증가 경로가 존재하지 않음
  - G가 단순 그래프이므로 G_f에도 루프와 평행 간선 없음
    - G_f에 역평행 간선은 존재 가능
    - $e \in E_f$이면 e와 e^R 중 정확히 하나만 E에 존재
    - $e \notin E_f$이면 $f(e) = c(e)$이고, $e^R \notin E_f$이면 $f(e) = 0$
    - G_f에 역평행 간선 e, e^R가 존재하면 $0 < f(e) < c(e)$
- 증명하고 싶은 것
  - G_f에 증가 경로가 존재하지 않으면 f는 최대 유량

Max-Flow Min-Cut Theorem

- 용어 정의
  - 유량 네트워크 G_f와 절단 $(S, V\backslash S)$에 대해
    - $\delta_f^+(S) = \{(u, v); (u, v) \in, E_f\ u \in S, v \in V\backslash S\}$: S에서 $V\backslash$S로 가는 간선
    - $\delta_f^-(S) = \{(u, v); (u, v) \in, E_f\ u \in V\backslash S, v \in \mathrm{S}\}$: $V\backslash$S에서 S로 가는 간선
    - 절단의 비용 $c(S) = \sum_{e \in \delta_f^+(S)} c(e)$
    - 절단의 유량 $f(S) = \sum_{e \in \delta_f^+(S)} f(e) - \sum_{e \in \delta_f^-(S)} f(e)$: S에서 $V\backslash$S로 가는 유량

Max-Flow Min-Cut Theorem

- 몇 가지 성질
  - 유량 네트워크 G_f와 절단 $(S, V\backslash S)$에 대해
    - 간선의 용량보다 더 많이 유량을 보낼 수 없으므로 $f(S) \leq c(S)$임을 알 수 있음
    - 잘 생각해 보면 $f(S) = |f|$임을 알 수 있음
      - 어떤 증가 경로 P가 $S \to V\backslash\mathrm{S}$방향으로 k번 이동했다면 $S \leftarrow V\backslash\mathrm{S}$방향으로 $k-1$번 이동함
      - 각 증가 경로는 $f(S)$를 $(k-(k-1)) \times F_p$만큼 증가시키므로 $f(S) = |f|$가 성립함

    - 최소 절단 $(S', V\backslash S')$에 대해 $|f| = f(S') \leq c(S')$
    - $|f| = c(S)$이면 f와 $(S, V\backslash\mathrm{S})$는 각각 최대 최대 유량, 최소 절단
      - 최대 유량 $f' \neq f$가 있다고 가정하면 $c(S) = |f| \leq |f'| \leq c(S') \leq c(S)$이므로 f도 최대 유량

Max-Flow Min-Cut Theorem

- Ford-Fulkerson Method 증명의 완성
  - 증가 경로가 없는 G_f에서 s로부터 도달 가능한 정점 집합을 S라고 하면 $|f| = c(S)$
    - 정의에 의해 $S \to V \backslash S$간선은 존재하지 않음 ($\delta_f^+(S) = \emptyset$)
    - $e \in \delta_f^-$가 $e \in E$를 만족하는 경우: $e^R \notin E_f$이므로 $f(e) = 0$
    - $e^R \in \delta_f^-$가 $e^R \notin E$를 만족하는 경우: $e \notin E_f$이므로 $f(e) = c(e)$

    - $|f| = f(S) = \sum_{e^R \in \delta_f^-(S), e^R \notin E_f} f(e) - \sum_{e \in \delta_f^-(S), e \in E_f} f(e) = \sum c(e) - \sum 0 = c(S)$
- Max-Flow Min-Cut Theorem
  - f가 최대 유량이면 G_f에 증가 경로가 존재하지 않음
  - G_f에 증가 경로가 존재하지 않으면 $(S, V \backslash S)$는 최소 절단이고 f는 최대 유량

Max-Flow Min-Cut Theorem

- 정리
  - G_f에 증가 경로가 존재하지 않으면 f는 최대 유량
  - 이때 s에서 도달할 수 있는 정점의 집합을 S라고 하면 $(S, V \backslash S)$는 최소 절단

질문?

Edmonds-Karp Algorithm

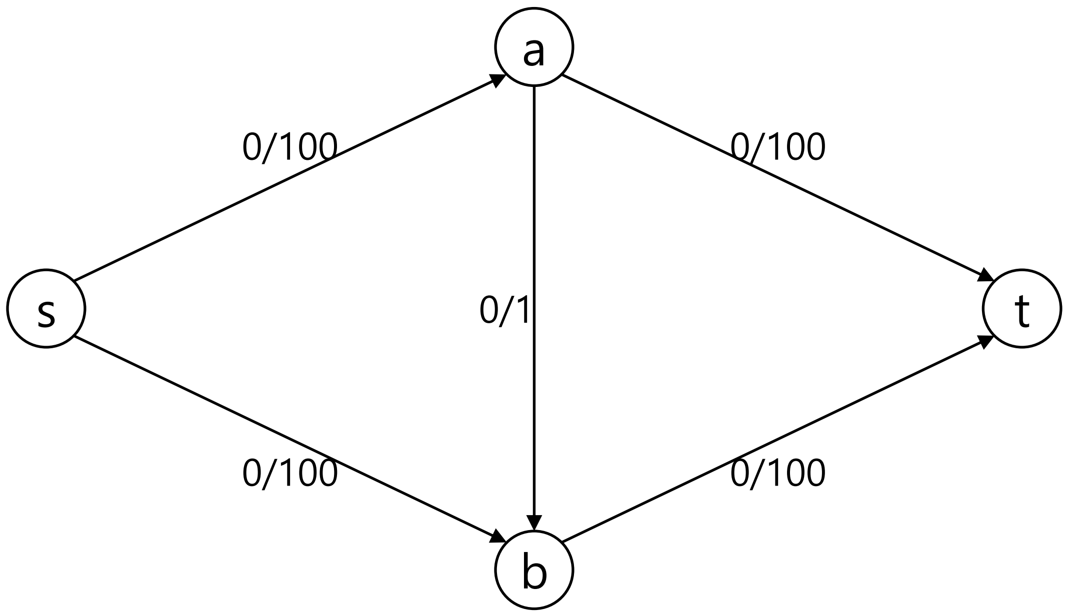
Edmonds-Karp Algorithm

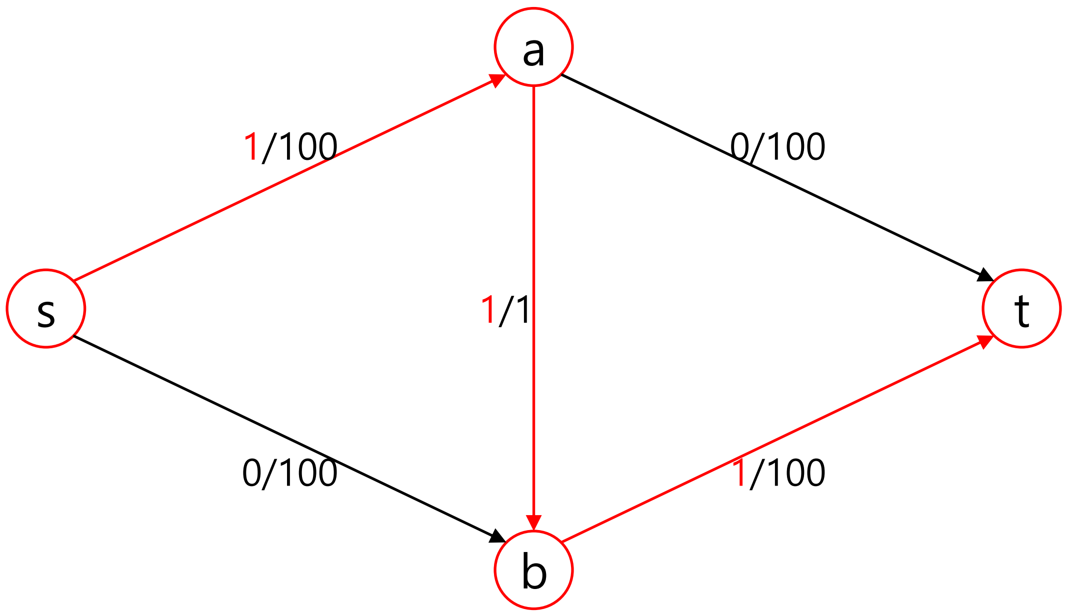
- Ford-Fulkerson Method의 시간 복잡도
  - 증가 경로를 찾아서 유량을 보낼 때마다 $|f|$가 최소 1씩 증가
  - 증가 경로를 찾는 건 DFS/BFS 등으로 $O(E)$에 가능
  - 최악의 경우 $O(Ef)$가 되고 이건 다항 시간이 아님

  - 다항 시간에 최대 유량을 구할 수 없을까?

Edmonds-Karp Algorithm

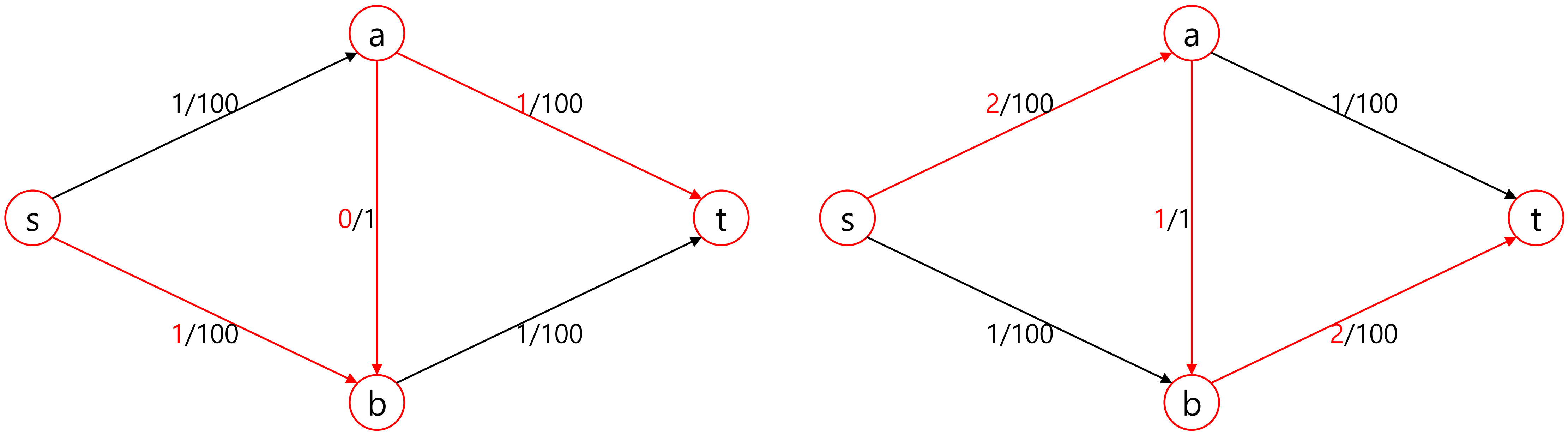
- Ford-Fulkerson Method의 worst case





Edmonds-Karp Algorithm

- Ford-Fulkerson Method의 worst case



Edmonds-Karp Algorithm

- Edmonds-Karp Algorithm
  - 매번 가장 짧은 증가 경로로 유량을 보내면 $O(VE^2)$
    - 증가 경로의 길이는 매번 단조 증가함 – 뒤에서 증명
    - 경로의 길이로 가능한 것은 최대 $V - 1$가지

    - 증가 경로의 길이가 고정되어 있을 때, 증가 경로를 찾을 때마다 최소 1개의 간선이 꽉 참
    - 증가 경로의 길이가 고정되어 있을 때 증가 경로는 최대 E번 찾게 됨
    - 따라서 증가 경로를 찾는 횟수는 최대 $O(VE)$

    - 가장 짧은 증가 경로는 BFS를 이용해 $O(E)$시간에 찾을 수 있으므로 전체 시간 복잡도는 $O(VE^2)$

Edmonds-Karp Algorithm

- Edmonds-Karp Algorithm
  - 증가 경로의 길이는 매번 단조 증가함
    - G_f에서 증가 경로 P를 찾아서 유량을 보낸 상태를 $f' = f + f_P$라고 하자.
    - G_f에서 v에서 t까지의 거리를 $d_f(v)$라고 하자. $d_{f'}(v) \geq d_f(v)$를 보일 것이다.

    - 모든 정점을 $d_{f'}(v)$ 오름차순으로 정렬해서 $v_1, v_2, \cdots, v_{|V|}$라고 하자.
    - t와 가까운 정점부터 수학적 귀납법을 이용해 증명한다.
      - $d_{f'}(v_i) = \infty$이면 $d_f(v_i) \leq \infty$이므로 $d_{f'}(v) < \infty$인 정점만 고려하면 됨
      - $d_{f'}(v_1 = t) = d_f(v_1)$은 자명함
      - $G_{f'}$에서 v_i에서 t로 가는 최단 경로에서 v_i 직후에 방문하는 정점을 $v_j(i > j, d_{f'}(v_j) \geq d_f(v_j))$라고 하자.
      - $(v_i, v_j) \in E_f$였다면 $d_{f'}(v_i) = d_{f'}(v_j) + 1 \geq d_f(v_j) + 1 \geq d_f(v_i)$
      - $(v_i, v_j) \notin E_f$였다면 $(v_j, v_i) \in P$이므로 $d_f(v_j) = d_f(v_i) + 1$
        - $d_{f'}(v_i) = d_{f'}(v_j) + 1 \geq d_f(v_j) + 1 = d_f(v_i) + 2 \geq d_f(v_i)$

질문?

Edmonds-Karp Algorithm

- 정리
  - Ford-Fulkerson Method의 시간 복잡도는 $O(Ef)$
  - 하지만 매번 가장 짧은 증가 경로를 찾으면 $O(VE^2)$
    - 매번 증가 경로의 길이가 단조 증가하기 때문

- 구현
  - 실제로 구현할 때는 루프, 평행 간선 따로 처리하지 않아도 잘 돌아감
  - 만약 무향 그래프에서 최대 유량을 구해야 하는 경우, 역방향 간선의 용량도 c로 설정하면 됨
    - https://www.inf.ufpr.br/elias/papers/2004/RT_DINF003_2004.pdf
    - ex. BOJ 6086

```cpp
struct maximum_flow{
    struct edge_t{ int v, c, r; };
    vector<vector<edge_t>> gph;
    vector<int> dst, prv, idx;
    maximum_flow(int n) : gph(n), dst(n), prv(n), idx(n) {}
    void add_edge(int s, int e, int c1, int c2=0){
        gph[s].push_back({e, c1, (int)gph[e].size()});
        gph[e].push_back({s, c2, (int)gph[s].size()-1}); // reverse edge
    }
    int augment(int s, int t){
        fill(dst.begin(), dst.end(), -1);
        queue<int> que; que.push(s); dst[s] = 0;
        while(!que.empty()){
            int v = que.front(); que.pop();
            for(int i=0; i<gph[v].size(); i++){
                const auto &e = gph[v][i];
                if(e.c > 0 && dst[e.v] == -1){
                    que.push(e.v); dst[e.v] = dst[v] + 1;
                    prv[e.v] = v; idx[e.v] = i;
                }
            }
        }
        if(dst[t] == -1) return 0;

        int flow = 1e9;
        for(int i=t; i!=s; i=prv[i]){
            const auto &e = gph[prv[i]][idx[i]];
            flow = min(flow, e.c);
        }
        for(int i=t; i!=s; i=prv[i]){
            auto &e = gph[prv[i]][idx[i]];
            e.c -= flow; gph[e.v][e.r].c += flow;
        }
        return flow;
    }
    int run(int s, int t){
        int flow = 0, path = 0;
        while((path = augment(s, t)) != 0) flow += path;
        return flow;
    }
};
```

질문?

Applications

Applications

- BOJ 6086 최대 유량
  - 무향 그래프의 최대 유량을 구하는 문제
  - G.add_edge(u, v, c, c)

Applications

- BOJ 14286 간선 끊어가기 2
    - 최소 절단의 가중치를 구하는 문제 = 최대 유량을 구하는 문제

Applications

- BOJ 17412 도시 왕복하기 1
  - 1번 정점에서 2번 정점으로 가는 경로 중, 간선이 겹치지 않는 서로 다른 경로의 최대 개수
  - 서로소 경로의 개수를 최대화하는 문제

  - 모든 간선의 용량을 1로 잡으면 됨
    - flow decomposition: 그래프의 최대 유량을 경로들의 집합으로 분해할 수 있음
    - 이때 모든 간선의 용량이 1이므로 각 간선은 최대 1개의 경로에 들어감

Applications

- BOJ 2316 도시 왕복하기 2
  - 정점과 간선이 겹치지 않는 서로 다른 경로의 최대 개수

  - 정점에 용량을 줄 수 있을까?
    - 각 정점을 2개로 분할: $v_i \rightarrow v_{in}(i), v_{out}(i)$
    - 간선 $(s, e) \rightarrow \left(v_{out}(s), v_{in}(e)\right)$
    - 정점 용량은 간선 $(v_{in}(i), v_{out}(i))$로 표현

    - 보통 정점 분할이라고 부름

Applications

- BOJ 11375 열혈강호
  - N명의 직원과 M개의 작업
  - 각 직원은 최대 한 개의 작업만 할 수 있고, 각 작업은 한 명이 담당함
  - 각 직원이 할 수 있는 작업이 주어졌을 때 완료할 수 있는 작업의 최대 개수

  - 그래프 모델링
    - source와 직원을 가중치가 1인 간선으로 연결
    - 직원과 작업을 가중치가 1인 간선으로 연결
    - 작업과 sink를 가중치가 1인 간선으로 연결

    - s에서 t로 가는 경로 = 직원이 작업을 하는 것
    - 최대 유량 = 직원과 작업을 최대한 많이 매칭하는 것

Applications

- BOJ 11376 열혈강호 2
  - N명의 직원과 M개의 작업
  - 각 직원은 최대 두 개의 작업만 할 수 있고, 각 작업은 한 명이 담당함
  - 각 직원이 할 수 있는 작업이 주어졌을 때 완료할 수 있는 작업의 최대 개수

  - 그래프 모델링
    - source와 직원을 가중치가 2인 간선으로 연결
    - 직원과 작업을 가중치가 1인 간선으로 연결
    - 작업과 sink를 가중치가 1인 간선으로 연결

    - s에서 t로 가는 경로 = 직원이 작업을 하는 것
    - 최대 유량 = 직원과 작업을 최대한 많이 매칭하는 것

Applications

- BOJ 11377 열혈강호 3
  - N명의 직원과 M개의 작업
  - K명의 직원은 최대 2개, 나머지 N-K명은 최대 1개의 작업을 할 수 있음
  - 각 작업은 한 명이 담당함
  - 각 직원이 할 수 있는 작업이 주어졌을 때 완료할 수 있는 작업의 최대 개수

  - 일을 2개 할 수 있는 K명을 어떻게 정하지?

Applications

- BOJ 11377 열혈강호 3
  - 일을 최대 x개 할 수 있는 직원: source에서 직원까지 가는 경로의 용량이 x가 되어야 함
    - 일단 source와 모든 직원을 용량이 1인 간선으로 연결하자.

  - "추가 근무 관리자"라는 직원을 새로 만들어서 관리자가 직원들에게 적절히 2번째 작업을 분배
    - source에서 관리자로 가는 용량 K짜리 간선
    - 관리자는 각 직원에게 최대 1개의 추가 작업을 줄 수 있음
    - 따라서 관리자에서 직원으로 가는 용량 1짜리 간선

질문?