Network Flow 3

나정휘
https://justiceHui.github.io/

목차

Review

Review

- 지금까지 배운 것
  - Ford-Fulkerson: 증가 경로를 찾아서 유량을 보내는 것을 반복하면 최대 유량 구할 수 있음
  - Edmonds-Karp: 매번 가장 짧은 증가 경로로 유량을 보내면 $O(VE^2)$
  - Max-Flow Min-Cut Theorem: 최대 유량 = 최소 절단
  - Konig's Theorem: 이분 그래프에서 최대 매칭 = 최소 정점 덮개
  - Dilworth's Theorem: Poset에서 최소 경로 덮개 = 최대 반사슬

  - 최소 절단: 최대 유량을 구한 뒤, 잔여 그래프에서 s로부터 도달 가능한 정점 집합
  - 최소 정점 덮개: 최대 매칭을 찾은 뒤, s에서 도달 불가능 왼쪽 정점 + 도달 가능 오른쪽 정점
  - 최대 독립 집합: 최소 정점 덮개의 여집합
  - 최소 경로 덮개: DAG의 간선으로 이분 그래프 만들고 최대 매칭에 대응되는 경로 덮개 찾기
  - 최대 반사슬: 이분 그래프 만들어서 v_l v_r 모두 최소 정점 커버에 속하지 않는 v 찾기

Review

- Edmonds-Karp에서 증명한 명제
  - 매번 최단 증가 경로를 따라 유량을 보내면 각 정점에서 sink까지의 거리는 항상 단조 증가
    - 따라서 source – sink 거리도 단조 증가
  - 길이가 d인 증가 경로로 인해 간선 e가 막혔다면, e는 더 이상 길이가 d인 증가 경로에 안 나옴
    - 즉, 최단 거리가 고정되어 있을 때 막혀 있던 간선이 다시 뚫리지 않음
    - 정확히는 $d + 2$ 이후에서 다시 등장

질문?

Dinic's Algorithm

Dinic's Algorithm

- Dinic's Algorithm
  - 대략적인 작동 과정
    - $f \leftarrow 0$
    - G_f에 증가 경로가 없어질 때까지 다음을 반복 … ①
      - $s - t$ 경로의 최단 거리 d를 구함
      - "특별한 방법"을 이용해 최단 거리가 d인 모든 증가 경로를 찾아서 보냄

  - 시간 복잡도
    - 증가 경로의 최대 거리는 $|V|$ 이므로 ①는 최대 $|V| - 1$번 반복
    - $s - t$ 경로의 최단 거리는 BFS를 이용해 $O(|E|)$에 구할 수 있음
    - "특별한 방법"의 시간 복잡도를 $T(|V|, |E|)$라고 하면 전체 시간 복잡도는 $O(|V| \times T(|V|, |E|))$
      - Edmonds-Karp는 $T(|V|, |E|) = O(|E|^2)$
      - Dinic의 목표는 $T(|V|, |E|)$를 $O(|V||E|)$ 로 만드는 것

Dinic’s Algorithm

- Dinic’s Algorithm
  - 길이가 d인 최단 증가 경로 $P = (s = v_0, v_1, v_2, \cdots, v_{d-1}, v_d = t)$ 를 생각해 보자.
    - G_f에서 $s - v_i$ 최단 거리는 i, $v_i - t$ 최단 거리는 $d - i$
    - 따라서 G_f의 어떤 정점 v가 s와 i 만큼, t와 $d - i$ 만큼 떨어져 있다면 v를 지나는 증가 경로 존재

  - Layered Graph
    - 최단 거리가 d인 잔여 그래프 G_f의 레이어 그래프 $L_f = (\bigcup_{i=0}^{d} V_i^f, \bigcup_{i=0}^{d-1} E_i^f)$
      - $V_i^f = \{v \in V; d_f(s, v) = i, d_f(v, t) = d - i\}$: 증가 경로가 존재하면서 s와 i 만큼 떨어진 정점
      - $E_i^f = \{(u, v) \in E_f; u \in V_i^f, v \in V_{i+1}^f\}$: s와 i 만큼 떨어진 정점에서 $i + 1$ 만큼 떨어진 정점으로 가는 간선
    - BFS를 이용해 $O(|E|)$에 찾을 수 있음

Dinic's Algorithm

- Dinic's Algorithm
  - 레이어 그래프의 성질
    - 길이가 d인 증가 경로는 항상 L_f에 속한 간선만 사용함
      - 막힌 간선이 다시 뚫리지 않으므로 증가 경로를 보내면 $E(L_f) \supsetneq E(L_{f'})$
    - 증가 경로를 보낼 때마다 매번 최소 한 개의 간선이 막힘
      - Edmonds-Karp와 동일
    - 따라서 L_f에서 찾을 수 있는 최대 $|E|$개의 증가 경로 찾으면 길이가 d인 모든 증가 경로를 찾게 됨

  - 목표: $O(|E|)$개의 증가 경로를 $O(|V||E|)$에 찾고 갱신하기

Dinic's Algorithm

- Dinic's Algorithm
  - Blocking Flow
    - 차단 유량: 레이어 그래프에서 s – t 경로가 존재하지 않도록 하는 유량
    - 차단 유량을 효율적으로 찾아야 함

  - 차단 유량을 찾는 과정
    - 레이어 그래프에서 증가 경로 P를 하나 찾음
      - 이걸 $O(|V|) + amortized\ O(1)$에 해야 함
    - P를 따라 유량을 흘림
      - $O(|V|)$에 가능
    - 레이어 그래프에서 막힌 간선 제거
      - 실제로 제거하지 않고 "막혔음" 이라고 표시만 하면 $O(|V|)$
    - 를 s – t 경로가 없어질 때까지 최대 $O(|E|)$번 반복

Dinic's Algorithm

- Dinic's Algorithm
  - 목표: $O(|E|)$개의 증가 경로를 $O(|V||E|)$에 찾고 갱신하기
    - 각 증가 경로를 $O(|V|) + amortized\ O(1)$에 찾을 수 있을까?
    - DFS/BFS 등에서 연산 낭비 없이 정확히 경로에 포함된 간선만 확인해야 함

  - 차단 유량을 효율적으로 찾는 방법
    - DFS가 $O(|E|)$인 이유: 도착점까지 가지 못하는 막다른 길에 도달했을 때 다른 경로를 찾아야 함
      - 이런 경우 없이 경로만 정확히 뽑아낼 수 있다면 $O(|V|)$ 가능
    - 각 정점 v마다 $idx[v]$를 관리
      - v를 시작점으로 하는 레이어 그래프의 간선 중 인접 리스트에서 가장 앞에 있는 간선의 인덱스
      - 만약 간선 $gph[v][idx[v]]$ 가 막혔다면 $idx[v]$를 증가
      - $idx[v]$를 증가시키기 위해 간선을 보는 것을 제외하면, 정확히 경로 상의 간선만 보게 됨
      - $idx[v]$를 증가시키기 위해 보는 것은 모든 증가 경로에서 총 $O(|E|)$번
    - 따라서 차단 유량을 $O(|V||E| + |E|)$에 찾을 수 있음

Dinic’s Algorithm

- Dinic’s Algorithm
  - 아래 과정을 $O(|V|)$번 반복
    - BFS를 이용해 $O(|E|)$에 레이어 그래프 생성
    - $idx[*]$를 이용한 DFS로 $O(|V||E| + |E|)$에 차단 유량 구함
  - 따라서 전체 시간 복잡도는 $O(|V|^2|E|)$

Dinic’s Algorithm

```cpp
struct Dinic{
    struct edge_t{ int v, r, c; };
    int n;
    vector<vector<edge_t>> g;
    vector<int> lv, idx;
    Dinic(int n) : n(n), g(n), lv(n), idx(n) {}
    void add_edge(int s, int e, int c1, int c2=0){
        g[s].push_back({e, (int)g[e].size(), c1});
        g[e].push_back({s, (int)g[s].size()-1, c2});
    }
    bool bfs(int s, int t){
        fill(lv.begin(), lv.end(), 0);
        queue<int> que; que.push(s); lv[s] = 1;
        while(!que.empty()){
            int v = que.front(); que.pop();
            for(const auto &e : g[v]) if(!lv[e.v] && e.c != 0) que.push(e.v), lv[e.v] = lv[v] + 1;
        }
        return lv[t] != 0;
    }
    int dfs(int v, int t, int fl=1e9){
        if(v == t || fl == 0) return fl;
        for(int &i=idx[v]; i<g[v].size(); i++){
            auto &e = g[v][i];
            if(lv[e.v] != lv[v] + 1 || e.c == 0) continue;
            int now = dfs(e.v, t, min(fl, e.c));
            if(now == 0) continue;
            e.c -= now; g[e.v][e.r].c += now;
            return now;
        }
        return 0;
    }
    int maximum_flow(int s, int t){
        int flow = 0, augment = 0;
        while(bfs(s, t)){
            fill(idx.begin(), idx.end(), 0);
            while((augment=dfs(s, t)) != 0) flow += augment;
        }
        return flow;
    }
};
```

질문?

Dinic's Algorithm

- Dinic's Algorithm with Unit Capacity
  - 모든 간선의 용량이 0 또는 1이면 최대 유량을 $O(\min(V^{2/3}, E^{1/2})\,E)$에 구할 수 있음
    - 차단 유량은 $O(|E|)$에 찾을 수 있음
      - 증가 경로의 모든 간선이 막히므로 각 간선은 최대 1번씩 사용됨

    - 레이어 그래프를 최대 $O(\min(V^{2/3}, E^{1/2}))$번 찾는다는 것을 보이면 됨
      - $let\ d_f(v) = d_f(v, t)$
      - Claim 1. $d_f(s) \geq E^{1/2}$이면 크기가 $E^{1/2}$ 이하인 절단 존재
        - 레이어 그래프 $E^{1/2}$번 찾으면 $d_f(s) \geq E^{1/2}$인 상태가 됨
        - G_f에서 최소 절단 = 최대 유량이 $E^{1/2}$이하이므로 레이어 그래프를 최대 $E^{1/2}$번 더 찾으면 종료
      - Claim 2. $d_f(s) \geq 2 \times V^{2/3}$이면 크기가 $V^{2/3}$ 이하인 절단 존재
        - 위와 동일한 이유로 레이어 그래프를 최대 $3 \times V^{2/3}$번 찾게 됨
      - Claim들을 증명해 보자.

Dinic's Algorithm

- Dinic's Algorithm with Unit Capacity
  - Claim 1. $d_f(s) \geq E^{1/2}$이면 크기가 $E^{1/2}$ 이하인 절단 존재
    - $D_k = \{v \in V; d_v = k\}, S_k = \{v; d_v \geq k\}$를 정의하자.
      - $k > 0$이면 $s \in S_k, t \notin S_k$이므로 $(S_k, V \setminus S_k)$는 올바른 절단
      - $S_k \rightarrow V \setminus S_k$ 간선은 모두 D_k에서 D_{k-1}로 가는 간선
    - $d_f(s) \geq E^{1/2}$이므로 $c(S_k) \leq E/E^{1/2} = E^{1/2}$인 k 존재
    - 따라서 $d_f(s) \geq E^{1/2}$이면 최소 절단 = 최대 유량은 $E^{1/2}$ 이하

Dinic’s Algorithm

- Dinic’s Algorithm with Unit Capacity
  - Claim 2. $d_f(s) \geq 2 \times V^{2/3}$이면 크기가 $V^{2/3}$ 이하인 절단 존재
    - $|D_k| \leq V^{1/3}, |D_{k-1}| \leq V^{1/3}$을 만족하는 k 존재
      - D_k 중 절반 이상은 $|D_k| \leq V^{1/3}$을 만족해야 함

    - $D_k \rightarrow D_{k-1}$ 간선은 최대 $V^{2/3}$개 존재하므로 $c(S_k) \leq V^{2/3}$
    - 따라서 $d_f(s) \geq V^{2/3}$이면 최소 절단 = 최대 유량은 $V^{2/3}$ 이하

질문?

Dinic's Algorithm

- Dinic's Algorithm with Capacity Scaling
  - 간선 용량의 최댓값을 2^k라고 하자.
    - 잔여 용량이 2^k 이상인 간선들만 사용했을 때의 최대 유량을 구하고
    - 잔여 용량이 2^{k-1} 이상인 간선들만 사용했을 때의 최대 유량을 구하고
    - 잔여 용량이 2^{k-2} 이상인 간선들만 사용했을 때의 최대 유량을 구하고
    - …
    - 잔여 용량이 2^0 이상인 간선들만 사용했을 때의 최대 유량을 구하자.
  - 전체 시간 복잡도는 $O(|V||E|k) = O(|V||E| \log U)$
    - 잔여 용량이 2^i 이상인 간선만 사용하는 상황
      - 레이어 그래프 $O(|V|)$번 생성
      - unit capacity 그래프와 동일한 이유로 차단 유량을 $O(|E|)$에 구할 수 있음
      - 따라서 여기까지 $O(|V||E|)$
    - 이 과정을 $k = O(\log U)$번 반복하므로 전체 시간 복잡도는 $O(|V||E| \log U)$

Dinic’s Algorithm

- Dinic’s Algorithm
  - 일반적인 그래프에서 $O(|V|^2|E|)$
  - 스케일링 적용하면 $O(|V||E| \log U)$
  - 용량이 0 또는 1이면 $O(\min(|V|^{2/3}, |E|^{1/2}) \times |E|)$
    - 레이어 그래프를 최대 $\min(3|V|^{2/3}, 2|E|^{1/2})$번만 찾음
    - 차단 유량은 $O(|E|)$에 구할 수 있음

질문?

Minimum Cost Maximum Flow

Minimum Cost Maximum Flow

- Minimum Cost Maximum Flow
  - 유량 네트워크 $G = (V, E, c)$, 간선의 가중치 $w : E \to \mathbb{R}$, 정점 s, t가 주어짐
    - 역방향 간선의 가중치 $w(e^R) = -w(e)$
    - G에 음수 사이클이 없는 상황만 생각하자.
  - 크기가 최대이면서 $\sum_{(u,v)} f(u, v) \times w(u, v)$가 최소인 유량 f를 구하는 문제
    - 유량의 비용을 최소화하는 문제

  - Successive Shortest Path Algorithm
    - 매번 비용이 가장 작은 증가 경로로 유량을 보내면 MCMF를 구할 수 있음
      - Why?
    - 시간 복잡도
      - 음수 가중치 있으므로 Bellman-Ford 또는 SPFA를 사용해야 함
      - $O(|V||E||f|)$, SPFA가 $O(|E|)$라고 가정하면 $O(|E||f|)$

Minimum Cost Maximum Flow

- Minimum Cost Maximum Flow
  - Successive Shortest Path Algorithm
    - 이 알고리즘이 항상 올바른 정답을 구함을 증명하기 위해서 보여야 하는 것
      - 원본 그래프에 음수 사이클이 존재하지 않으면 잔여 그래프에도 음수 사이클이 존재하지 않음
        - 음수 사이클이 있으면 최단 경로를 구할 수 없음
      - 알고리즘이 항상 종료됨
      - 매번 최단 경로를 따라 유량을 보내면 항상 최소 비용 유량을 구할 수 있음

      - Lemma 1. G_f에 음수 사이클이 없으면 최단 경로 P에 유량을 δ 만큼 흘린 $G_{f'}$에도 음수 사이클 없음
      - Lemma 2. G_f에 음수 사이클이 없는 것은 크기가 $|f|$인 유량 중 f가 최소 비용인 것과 동치
    - 두 보조 정리가 모두 참이라면…
      - 항상 최단 경로를 따라 유량을 보내면 음수 사이클이 존재하지 않으므로
      - 크기가 $1,2,3,\cdots$인 최소 비용 유량을 모두 구할 수 있음
      - Ford-Fulkerson Method는 최대 유량을 찾으므로 최소 비용 최대 유량을 찾을 수 있음

Minimum Cost Maximum Flow

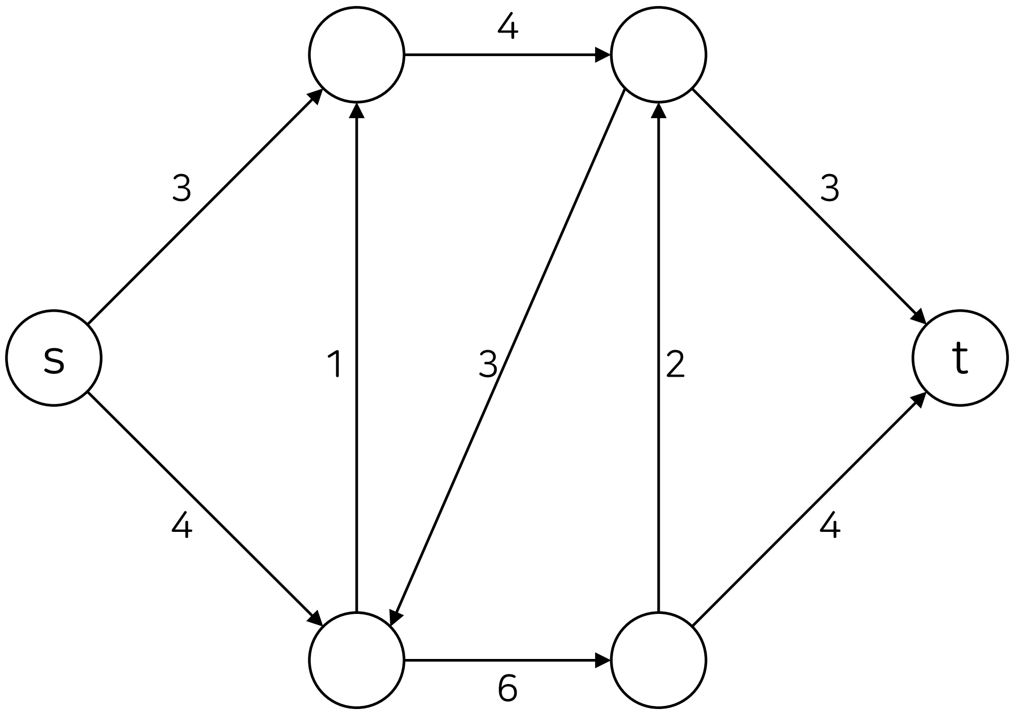
- Minimum Cost Maximum Flow
  - 유량의 비용에 대한 몇 가지 성질
    - $G = (V, E, c, w)$의 두 유량 f, g에 대해…
    - $f' = f + g$ 에서 $cost(f') = cost(f) + cost(g)$ 성립
    - $f' - f$ 에서 $cost(f' - f) = cost(f') - cost(f)$ 성립
    - 비용이 같은 평행 간선이 없다면 $\left(G_f\right)_g = G_{f+g}$

Minimum Cost Maximum Flow

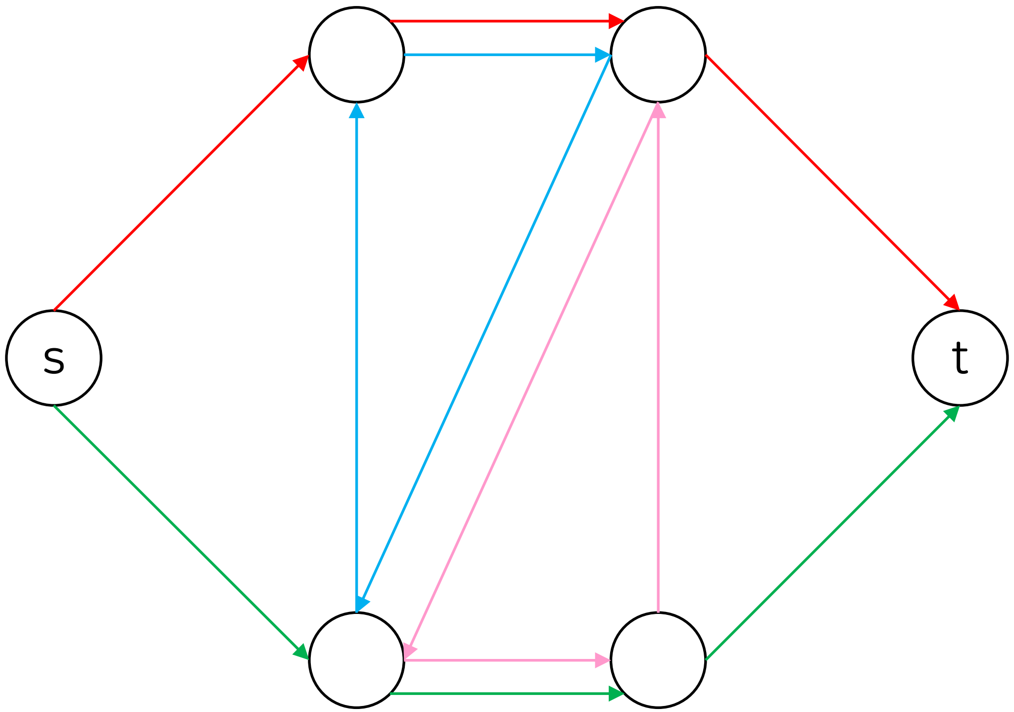
- Minimum Cost Maximum Flow
  - Flow Decomposition
    - 유량 네트워크 $G = (V, E, c)$와 유량 f에 대해, f는 다음과 같은 경로와 사이클로 분할 가능
      - 경로 집합 $P = \{p_1, p_2, \cdots, p_k\}$, 유량 집합 $F = \{f_1, f_2, \cdots, f_k\}$
        - f_i는 p_i를 따라서 흐르는 $|f_i|$ 만큼의 유량으로만 구성되어 있음
      - 사이클 집합 $C = \{c_1, c_2, \cdots, c_l\}$, 유량 집합 $F' = \{f'_1, f'_2, \cdots, f'_k\}$
        - f'_i는 c_i를 따라서 흐르는 $|f'_i|$ 만큼의 유량으로만 구성되어 있음
      - $|f| = \sum |f_i|$
    - 증명은 생략

Minimum Cost Maximum Flow

- Flow

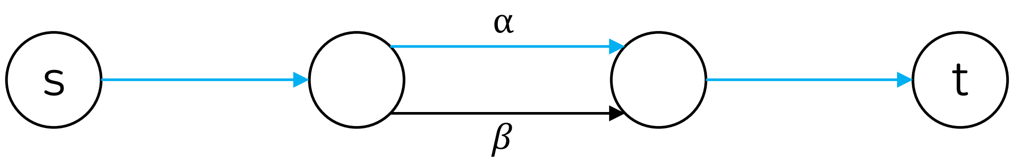


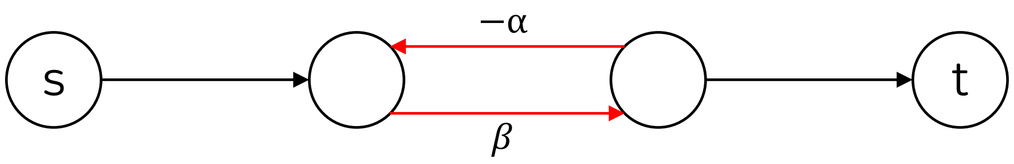
- Flow Decomposition



Minimum Cost Maximum Flow

- Minimum Cost Maximum Flow
  - Successive Shortest Path Algorithm
    - Lemma 1. G_f에 음수 사이클이 없으면 최단 경로 P에 유량을 δ 만큼 흘린 $G_{f'}$에도 음수 사이클 없음
      - α를 포함하는 경로 P를 따라 흘렸을 때 α의 역방향과 β로 구성된 음수 사이클이 생겼다고 하자.
      - 최단 경로 P에 β가 아닌 α가 포함되었으므로 $\alpha \leq \beta$
      - α의 역방향 가중치는 $-\alpha$이므로 사이클의 가중치는 $\beta - \alpha$
      - $\alpha \leq \beta$이므로 $\beta - \alpha \geq 0$이 돼서 음수 사이클이 생길 수 없음





Minimum Cost Maximum Flow

- Minimum Cost Maximum Flow
  - Successive Shortest Path Algorithm
    - Lemma 2. G_f에 음수 사이클이 없는 것은 크기가 $|f|$인 유량 중 f가 최소 비용인 것과 동치
      - case 1. f는 최소 비용
        - G_f에 음수 사이클 C가 존재한다고 하자.
        - C의 간선들을 따라 흐르는 유량 g를 정의할 수 있다.
        - g에서 sink로 들어가는 유량이 없으므로 $|g| = 0$이지만 음수 사이클을 따라 흐르므로 $cost(g) < 0$
        - $|f + g| = |f| + |g| = |f|$인데 $cost(f + g) < cost(f)$, f가 최소 비용이라는 가정에 모순
        - 따라서 G_f에 음수 사이클 없음
      - case 2. f는 최소 비용이 아님
        - $|f'| = |f|, cost(f') < cost(f)$인 f'이 존재한다고 하자.
        - $|f' - f| = 0$이므로 $f' - f$의 flow decomposition은 사이클로만 구성되어 있음
        - $cost(f' - f) < 0$ 이므로 최소 하나의 음수 사이클 존재

Minimum Cost Maximum Flow

- Minimum Cost Maximum Flow
  - Successive Shortest Path Algorithm
    - 따라서 매번 가장 짧은 증가 경로를 따라 유량을 보내면 MCMF를 구할 수 있음
    - 또한, 크기가 $1,2,\cdots,|f|$인 최소 비용의 유량도 구할 수 있음
    - 시간 복잡도는 $O(|V||E||f|)$

  - Maximum Cost Maximum Flow
    - 간선 가중치에 -1 곱하면 됨

Minimum Cost Maximum Flow

```
struct MCMF{
    struct edge_t{ int v, r; flow_t c; cost_t d; };
    vector<vector<edge_t>> g;
    vector<int> in, prv, idx;
    vector<cost_t> dst;
    MCMF(int n) : g(n), in(n), prv(n), idx(n), dst(n) {}
    void add_edge(int s, int e, flow_t c, flow_t d){
        g[s].push_back({e, (int)g[e].size(), c, d});
        g[e].push_back({s, (int)g[s].size()-1, 0, -d});
    }
    bool augment(int s, int t){
        fill(in.begin(), in.end(), 0);
        fill(dst.begin(), dst.end(), INF);
        queue<int> que; que.push(s); dst[s] = 0; in[s] = 1;
        while(!que.empty()){
            int v = que.front(); que.pop(); in[v] = 0;
            for(int i=0; i<g[v].size(); i++){
                const auto &e = g[v][i];
                if(e.c != 0 && dst[e.v] > dst[v] + e.d){
                    dst[e.v] = dst[v] + e.d;
                    prv[e.v] = v; idx[e.v] = i;
                    if(!in[e.v]) in[e.v] = 1, que.push(e.v);
                }
            }
        }
        return dst[t] < INF;
    }
    pair<flow_t, cost_t> minimum_cost_flow(int s, int t){
        flow_t flow = 0; cost_t cost = 0;
        while(augment(s, t)){
            flow_t path = INF;
            for(int i=t; i!=s; i=prv[i]) path = min(path, g[prv[i]][idx[i]].c);
            flow += path; cost += path * dst[t];
            for(int i=t; i!=s; i=prv[i]){
                auto &e = g[prv[i]][idx[i]];
                e.c -= path; g[i][e.r].c += path;
            }
        }
        return {flow, cost};
    }
};
```

질문?

Minimum Cost Maximum Flow

- Minimum Cost Maximum Flow
  - 중요한 성질: 증가 경로의 최단 거리는 단조 증가함
    - Successive Shortest Path Algorithm의 연속한 단계에서 얻은 세 유량 f, g, h를 생각해 보자.
      - G_f의 최단 경로 P를 따라 유량을 $\alpha > 0$ 만큼 흘린 $g = f + f_P$
      - G_g의 최단 경로 Q를 따라 유량을 $\beta > 0$ 만큼 흘린 $h = g + g_Q$
      - 정의에 의해 f, g, h는 각자의 크기에서 최소 비용
    - G_f에 추가할 수 있는 유량 $h - f$을 생각해 보자.
      - $|h - f| = \alpha + \beta, cost(h - f) = \alpha cost(P) + \beta cost(Q)$
      - $|h - f|$의 크기를 α로 줄인 $\frac{\alpha}{\alpha+\beta}(h - f) = k$를 정의하자.
      - $cost(k) = \frac{\alpha}{\alpha+\beta}\left(\alpha cost(P) + \beta cost(Q)\right) \geq \alpha cost(P)$, g가 최소 비용이기 때문에 부등호 성립
      - 양변 α로 나누고 이항하면 $\frac{\beta}{\alpha+\beta} cost(Q) \geq \left(1 - \frac{\alpha}{\alpha+\beta}\right) cost(P) = \frac{\beta}{\alpha+\beta} cost(P)$
      - $0 < \frac{\beta}{\alpha+\beta} < 1$이므로 $cost(P) \leq cost(Q)$

Minimum Cost Maximum Flow

- Minimum Cost Maximum Flow
  - 중요한 성질: 증가 경로의 최단 거리는 단조 증가함
    - 따라서 minimum cost flow는 아래로 볼록한 함수라고 생각할 수 있음
    - maximum cost flow는 위로 볼록한 함수

    - 이게 왜 중요할까?
      - 나중에 Aliens Trick에서 다시 알아보자.

질문?

더 공부하고 싶다면...

- 다양한 내용이 있어요
  - Johnson’s Algorithm
    - 음수 사이클이 없을 때, 그래프의 최단 거리를 유지하면서 간선의 가중치를 0 이상으로 바꾸는 방법
    - $O(|V||E|)$ 전처리를 하면, 이후 MCMF에서 증가 경로를 찾을 때 Dijkstra’s algo로 찾을 수 있음
  - Hopcroft-Karp Algorithm
    - Dinic’s algorithm의 응용, 이분 매칭을 $O(|E|\sqrt{|V|})$에 구하는 알고리즘
  - MPM Algorithm
    - Dinic’s algorithm의 응용, 최대 유량을 $O(|V|^3)$에 구하는 알고리즘, 근데 별로 안 좋음
  - Dinic’s Algorithm with Dynamic Tree
    - link/cut tree를 사용하면 차단 유량을 $O(|E|\log|V|)$에 찾을 수 있음
    - 따라서 전체 시간 복잡도는 $O(|V||E|\log|V|)$
    - 근데 별로 안 좋음

더 공부하고 싶다면...

- 다양한 내용이 있어요
  - Circulation
    - flow network의 일반화 버전
  - Push-relabel Algorithm
    - 최대 유량을 $O(|V|^2|E|), O(|V|^3), O(|V|^2\sqrt{|E|})$에 구하는 알고리즘
    - PS 범위에서는 가장 효율적인 알고리즘
  - Capacity-scaling Algorithm
    - 최대 유량에서의 capacity scaling과 비슷한 아이디어, $O(|E|^2 \log|V| \log U)$에 구하는 알고리즘
  - Cost-scaling Algorithm
    - push-relabel의 응용, MCMF를 $O(|V|^3 \log |V|C)$에 구하는 알고리즘
    - PS 범위에서는 가장 효율적인 알고리즘
  - Double-scaling Algorithm
    - 둘 다 스케일링하면 $O(|V||E| \log C \log\log U)$라고 하던데...