最大流

国立大学法人 北海道大学
工学部情報エレクトロニクス学科
情報理工学コース情報知識ネットワーク研究室
学部三年　大泉翼

はじめに

本スライドは蟻本に沿った内容になります。

二部マッチングの証明のみ自分で考えたもので、不備がある可能性がございます。どうぞご指摘下さい。

また、グラフに逆辺を張る操作を図にするので、図が見えにくいですが、頑張ってみてください。

目次

- 最大フロー、最小カットのお話
- 競プロのお話

目次

- 最大フロー、最小カットのお話
- 競プロのお話

前提知識

$\delta_{-}(v)$:= 頂点 v へ入ってくる辺の集合
$\delta_{+}(v)$:= 頂点 v から出ていく辺の集合

二部グラフ
以下の性質を満たす頂点集合の分割 $(S, V \setminus S)$が存在する
-グラフのどの辺も S の要素の頂点と、$V \setminus S$ の要素の頂点とに接続している

最大フローとは

有向グラフ $G = (V, E)$ において、各有向辺 e に負でない実数 $c(e), e \in E$ が定められているとする。

[定義]

フローとは、G の各有向辺 e に負でない値 $f(e)$を割り当てる関数 f が存在し、次の性質を満たすものをいう。

・ $\mathrm{e} \in E$に対して $\mathrm{f(e)} \leq c(e)$

・ $\mathrm{v} \in V \setminus \{s, t\}$ に対して $\sum_{e \in \delta_{-}(v)} f(e) = \sum_{e \in \delta_{+}(v)} f(e)$

$\sum_{e \in \delta_{+}(s)} f(e) = \sum_{e \in \delta_{-}(t)} f(e)$ = フロー値

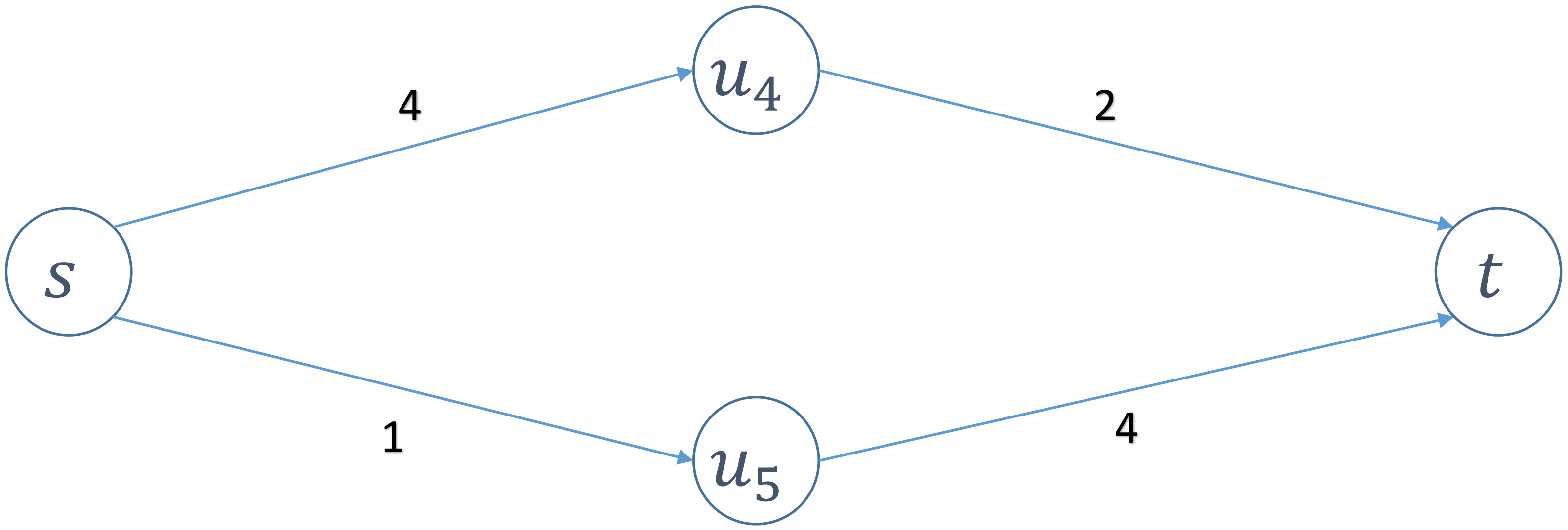
フロー値が最大になるフローを最大フローという

最大フローとは

重みつき有向グラフＧにおいて
各辺の容量を超えずにｓ から ｔに送ることのできる量の最大値

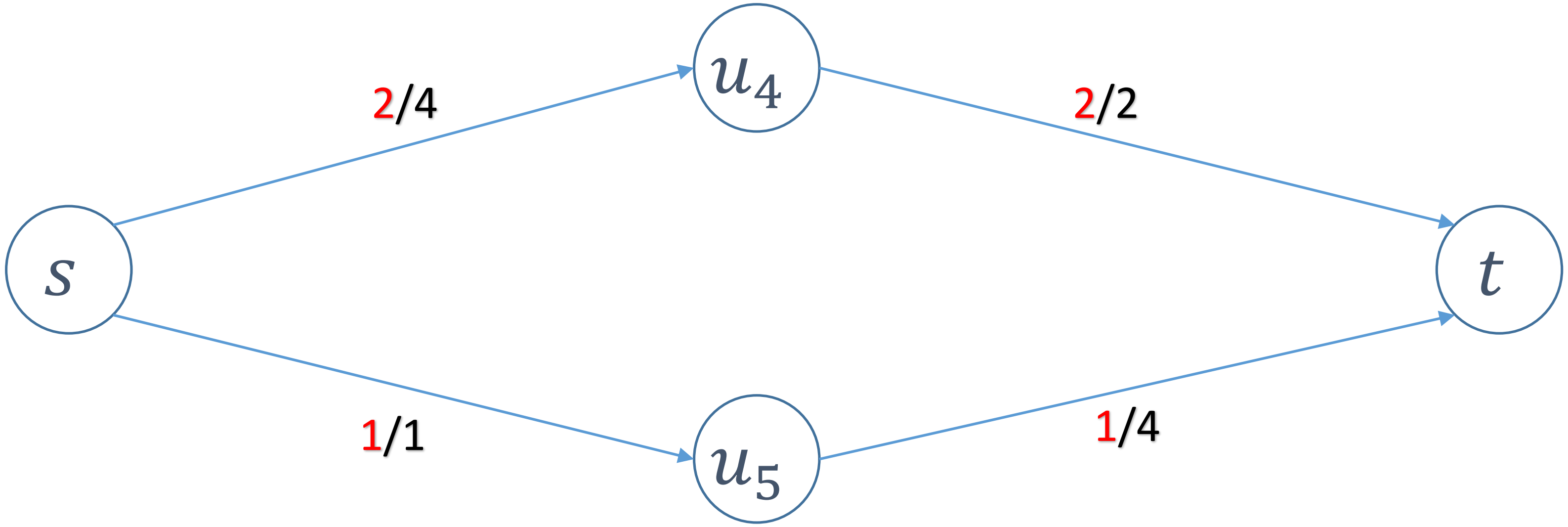
最大フローとは

重みつき有向グラフ G において
各辺の容量を超えずに s から t に送ることのできる量の最大値



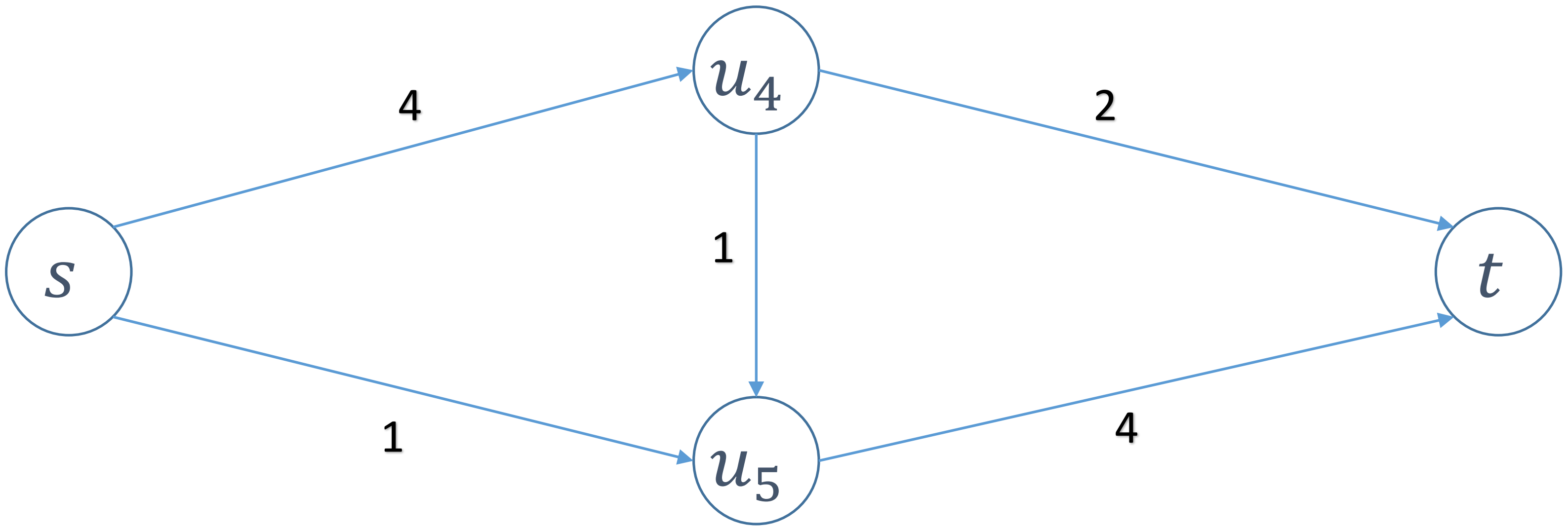
最大フローとは

重みつき有向グラフ G において
各辺の容量を超えずに s から t に送ることのできる量の最大値



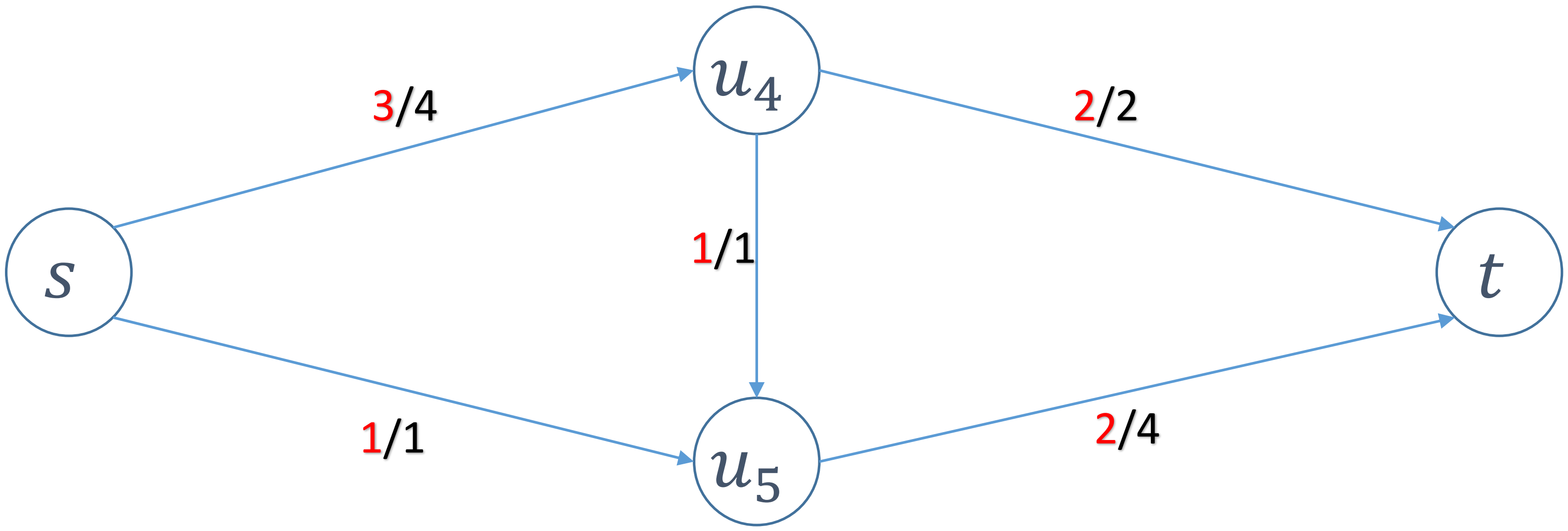
最大フローとは

重みつき有向グラフ G において
各辺の容量を超えずに s から t に送ることのできる量の最大値



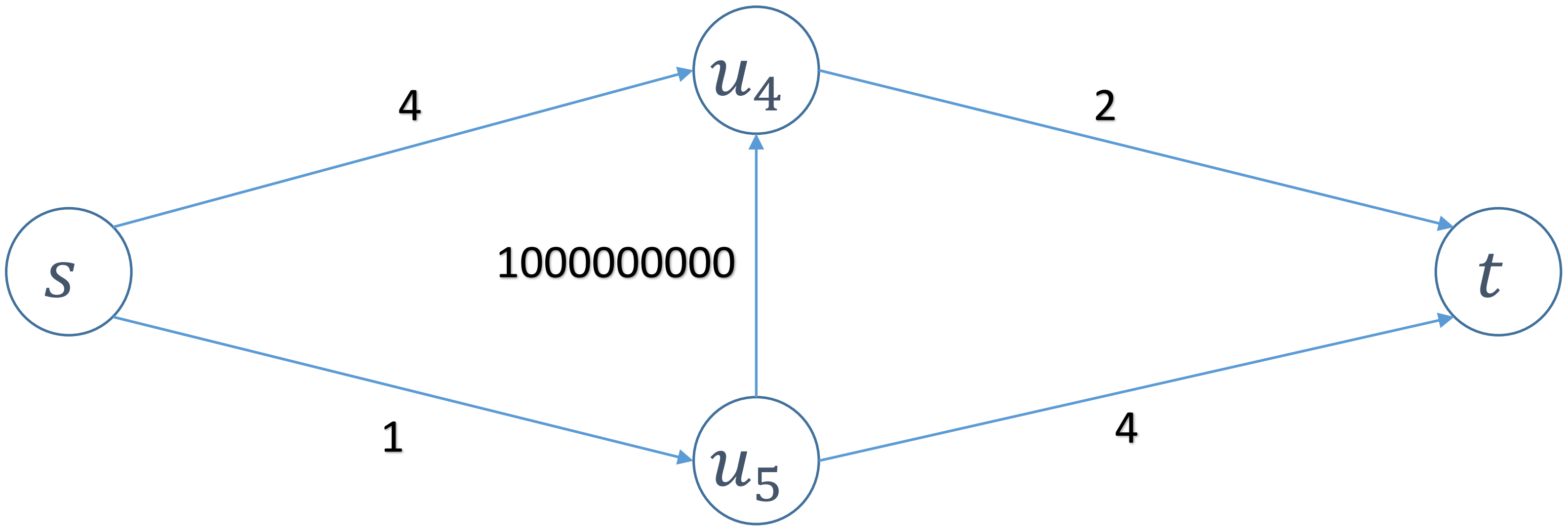
最大フローとは

重みつき有向グラフ G において
各辺の容量を超えずに s から t に送ることのできる量の最大値



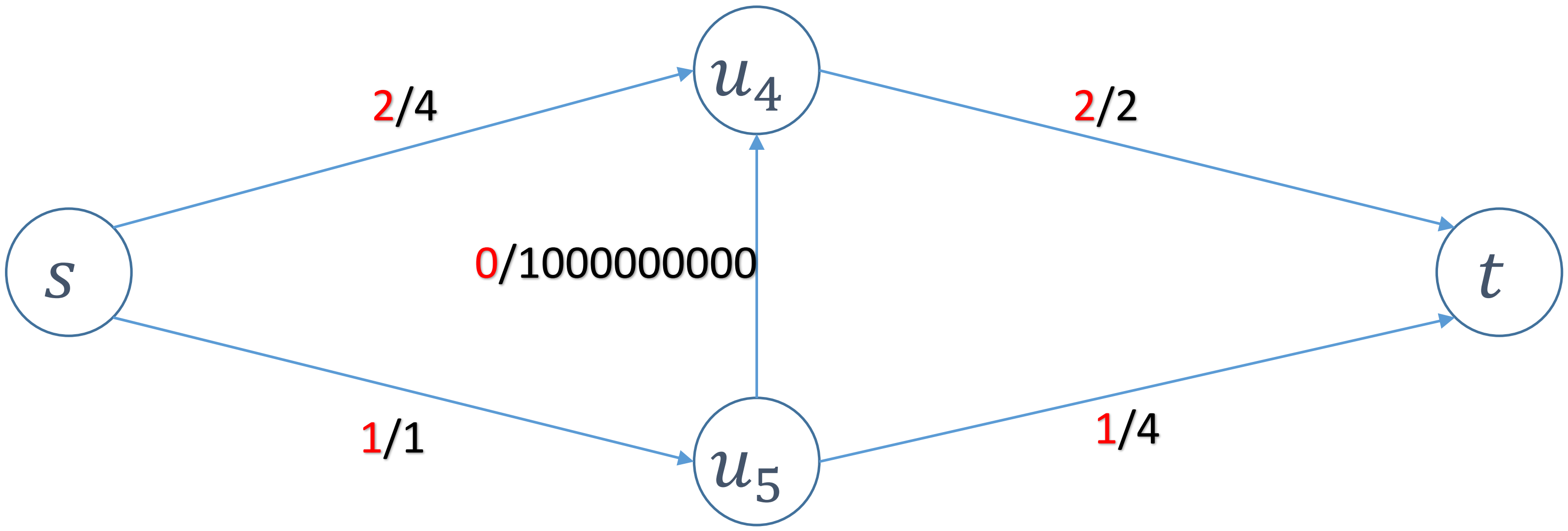
最大フローとは

重みつき有向グラフ G において
各辺の容量を超えずに s から t に送ることのできる量の最大値



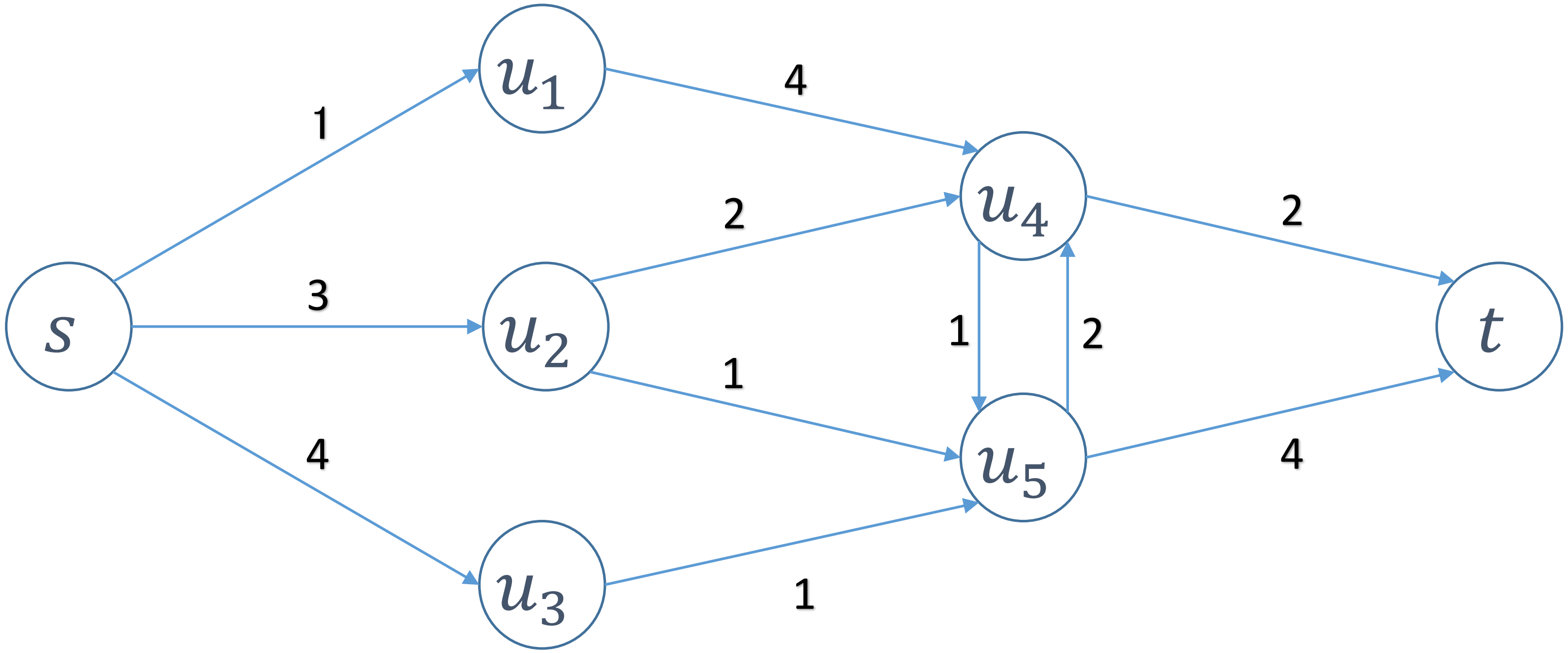
最大フローとは

重みつき有向グラフ G において
各辺の容量を超えずに s から t に送ることのできる量の最大値



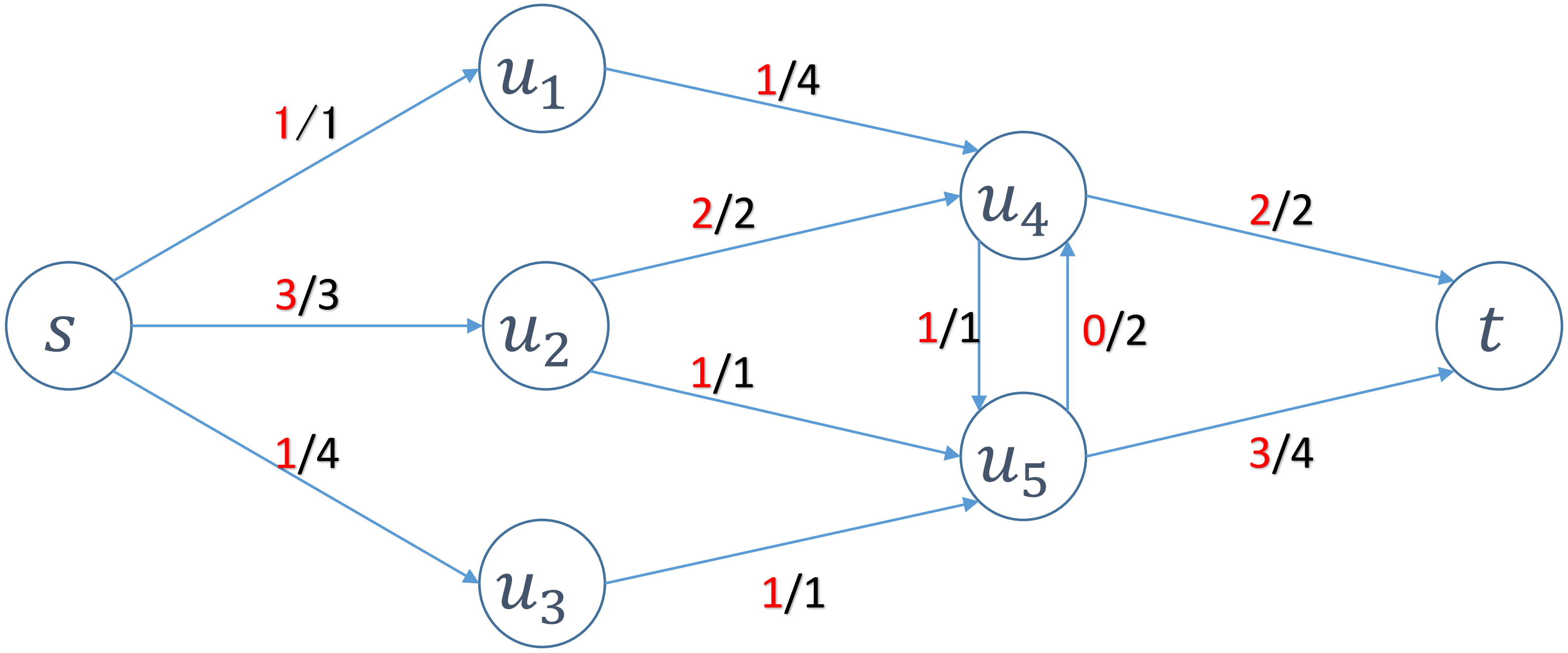
最大フローとは

重みつき有向グラフ G において
各辺の容量を超えずに s から t に送ることのできる量の最大値



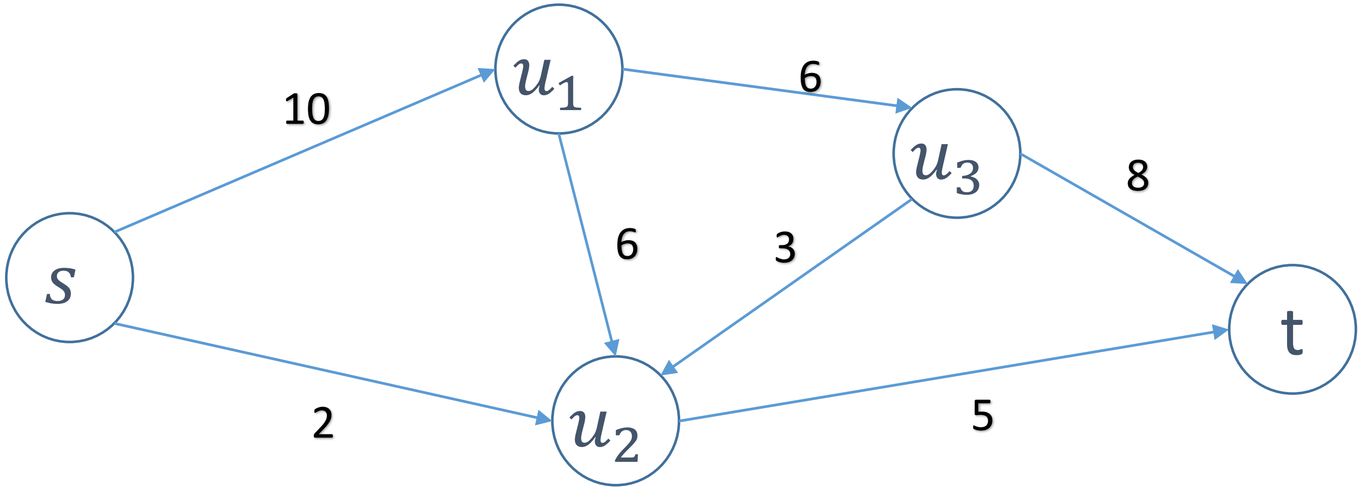
最大フローとは

重みつき有向グラフ G において
各辺の容量を超えずに s から t に送ることのできる量の最大値



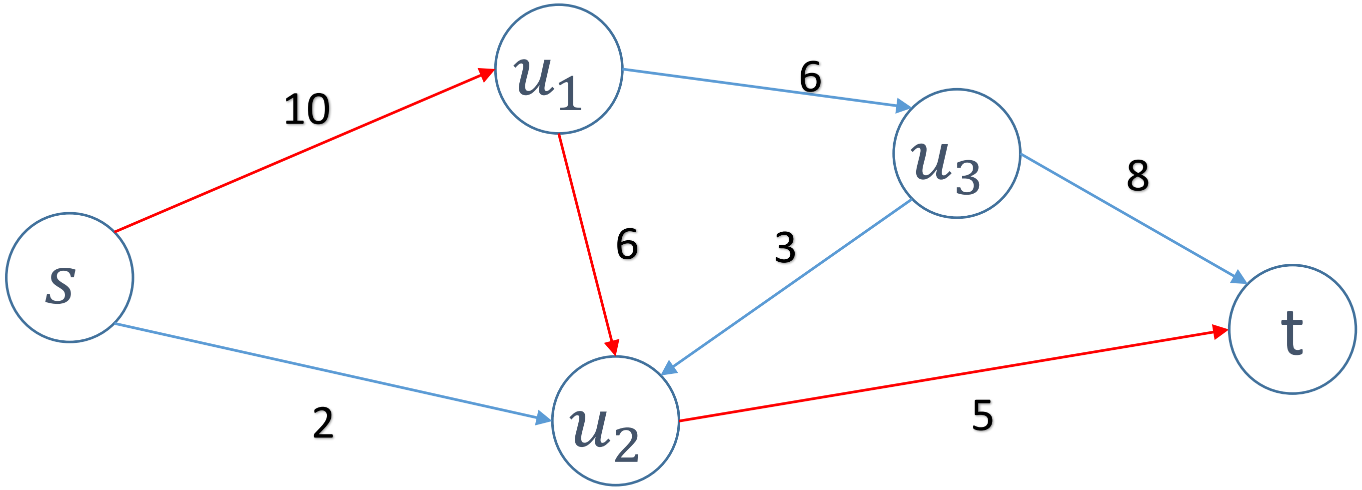
貪欲のアルゴリズム？

- (1)流せる余地のある辺のみを使ってｓ からｔへのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



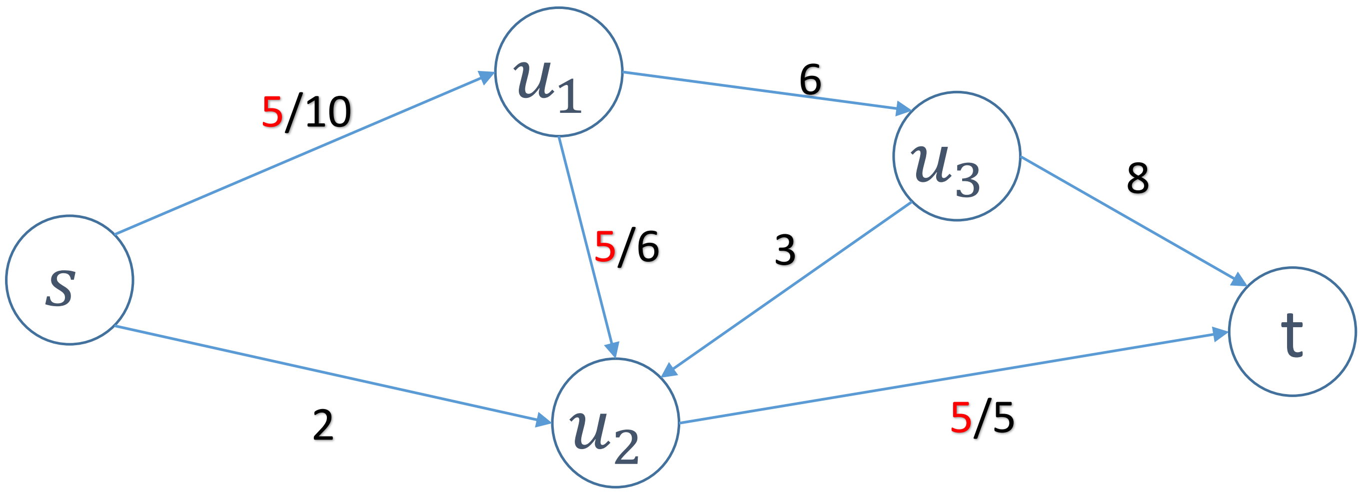
貪欲のアルゴリズム？

- (1)流せる余地のある辺のみを使ってｓ から ｔへのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



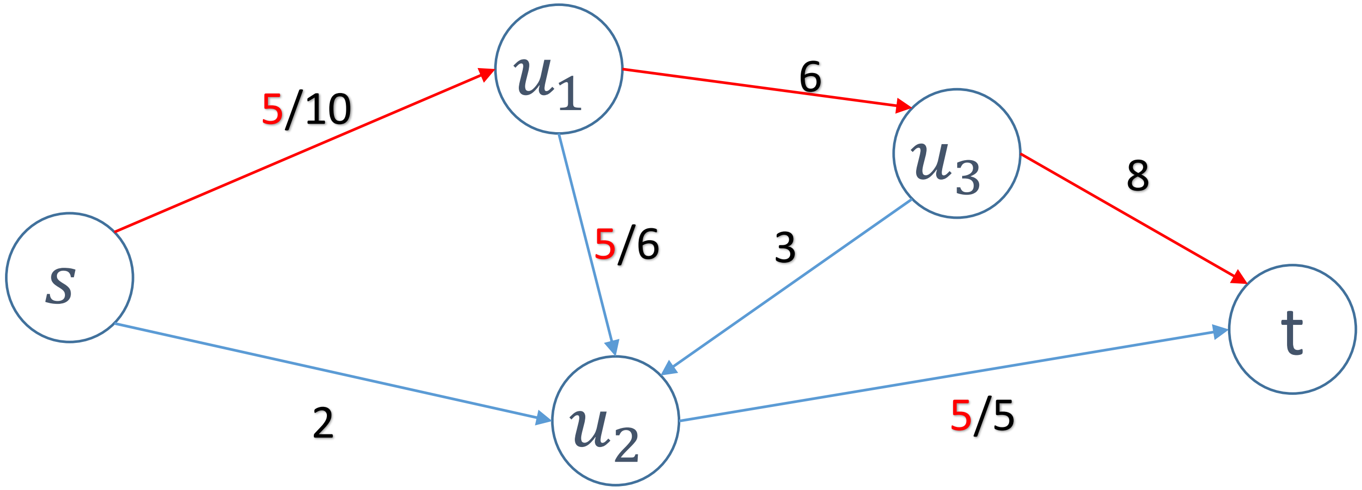
貪欲のアルゴリズム？

- (1)流せる余地のある辺のみを使ってｓ から ｔへのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



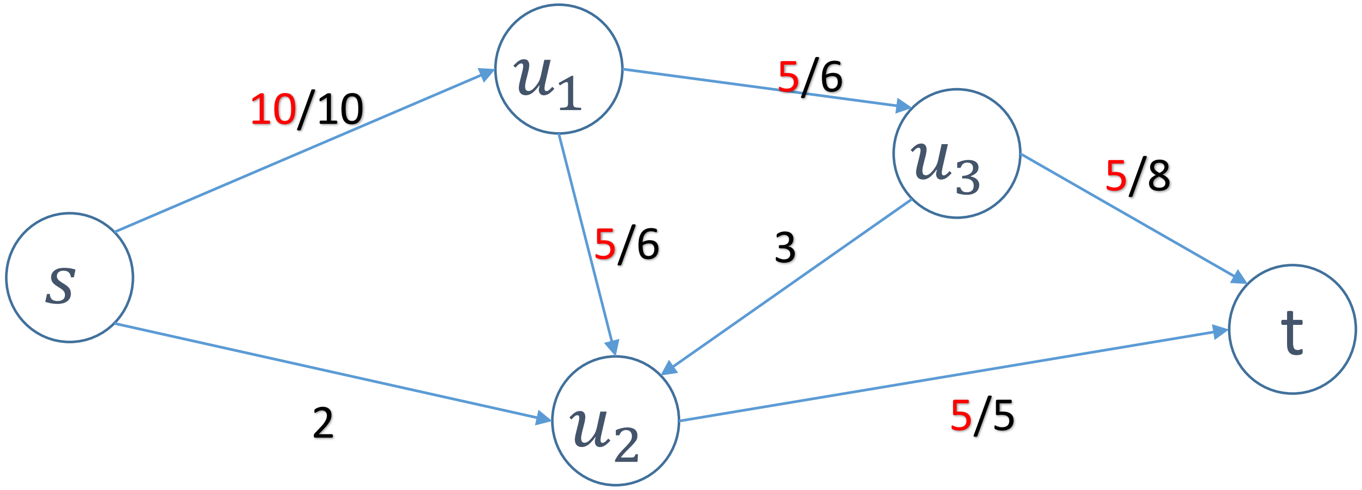
貪欲のアルゴリズム？

- (1)流せる余地のある辺のみを使ってｓ から ｔへのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



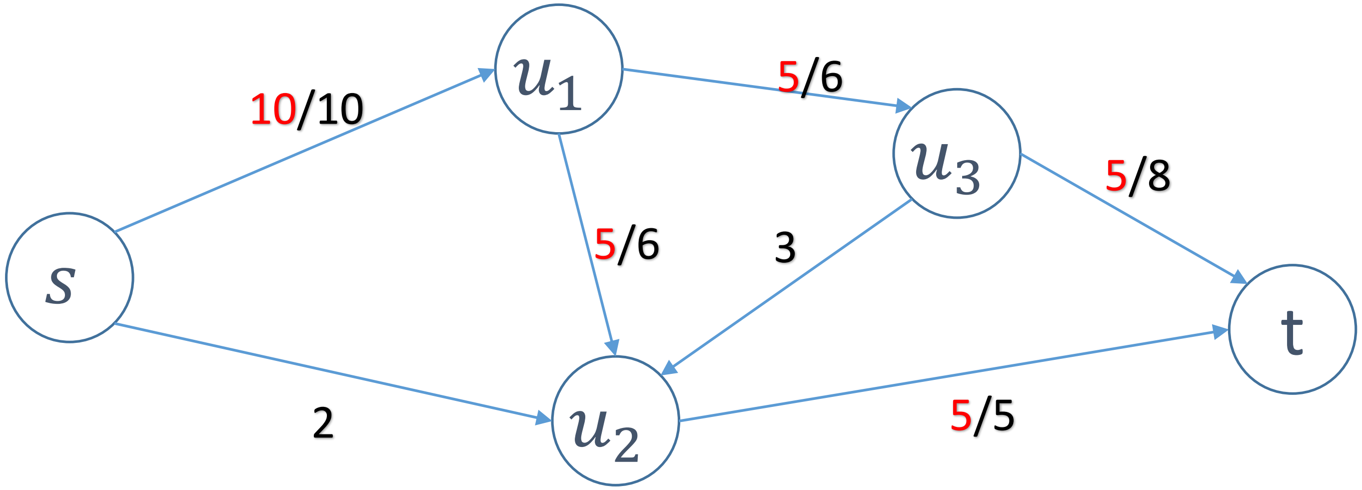
貪欲のアルゴリズム？

- (1)流せる余地のある辺のみを使ってｓ から ｔへのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



貪欲のアルゴリズム？

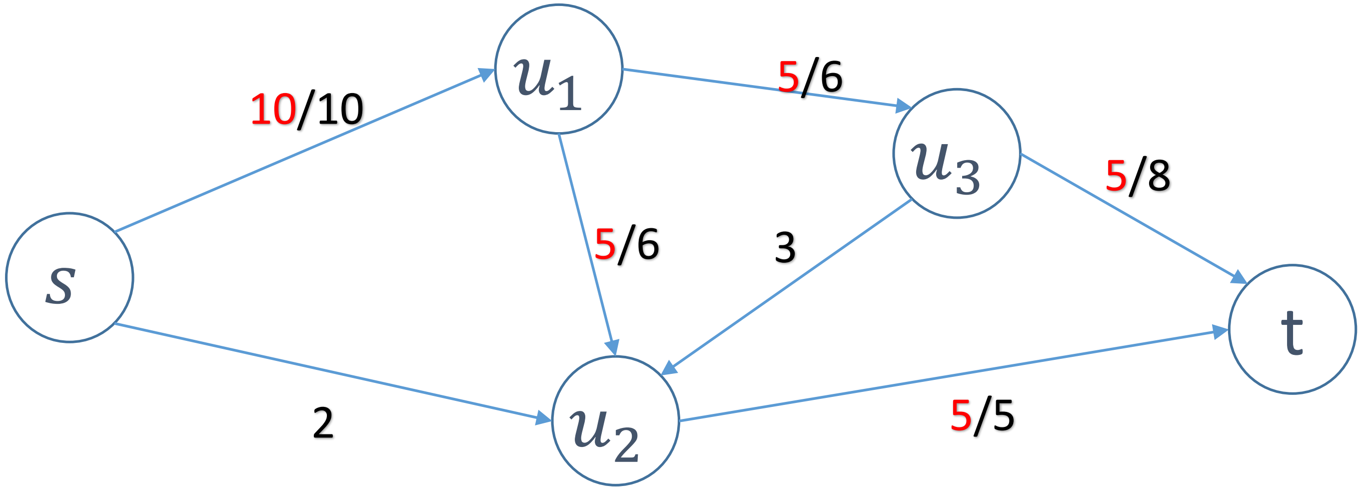
- (1)流せる余地のある辺のみを使って s から t へのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



終了！
答えは10!

貪欲のアルゴリズム？

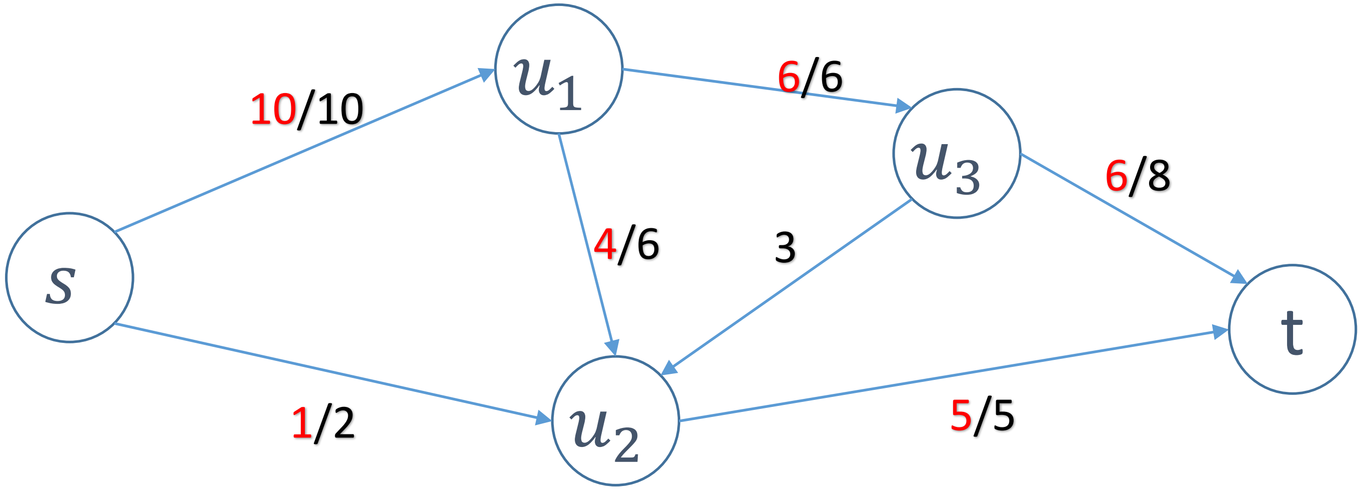
- (1)流せる余地のある辺のみを使ってｓ から ｔへのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



でも、
本当は？

貪欲のアルゴリズム？

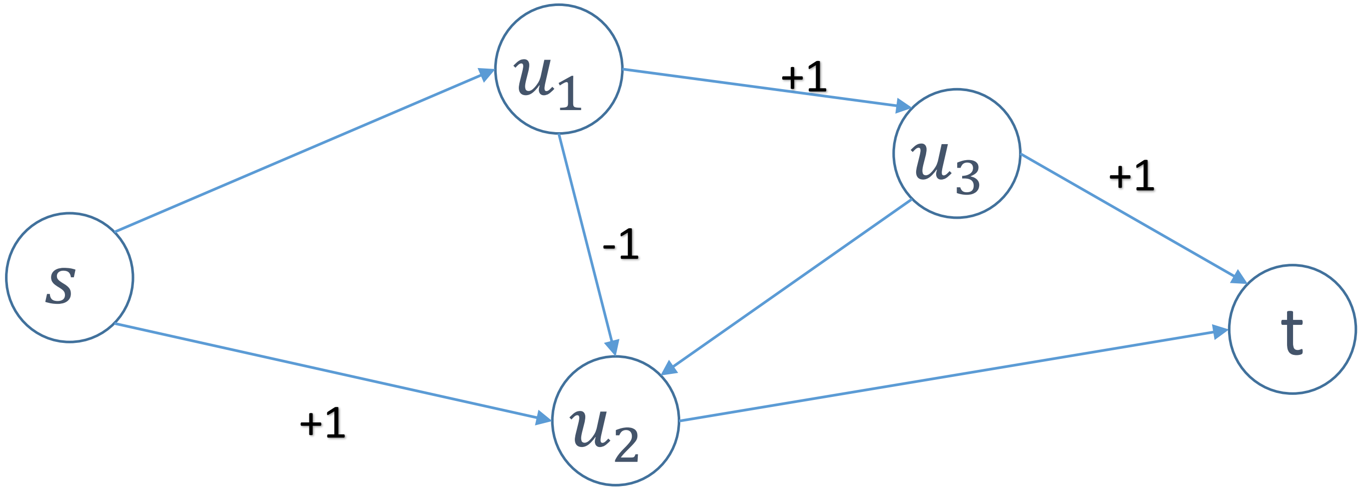
- (1)流せる余地のある辺のみを使ってｓ から ｔへのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



正しい答え
11!

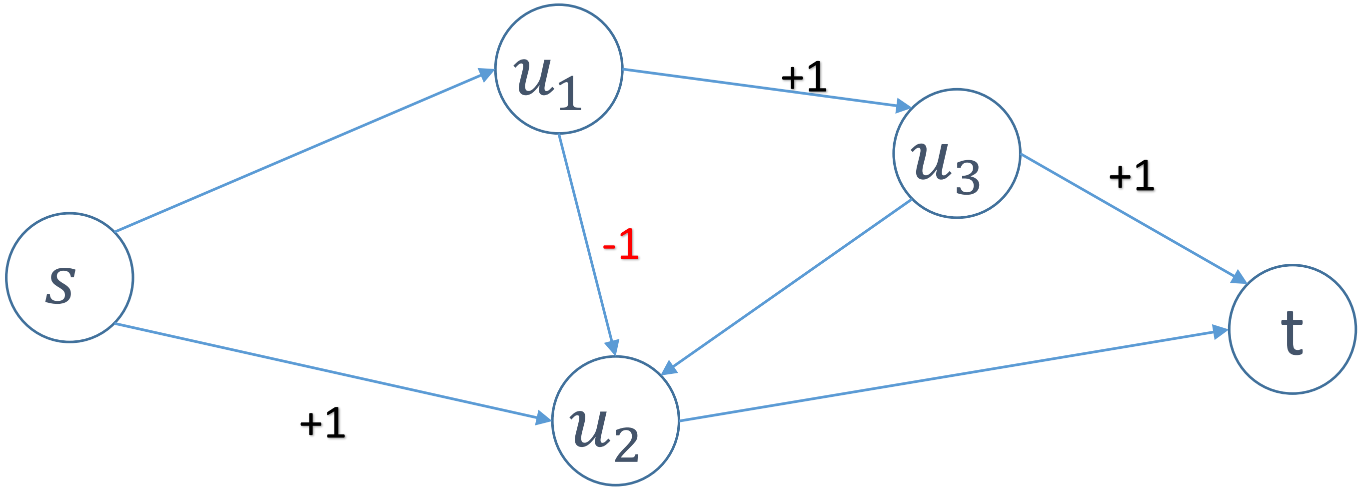
貪欲のアルゴリズム？

- 両者の流量の差に着目してみる



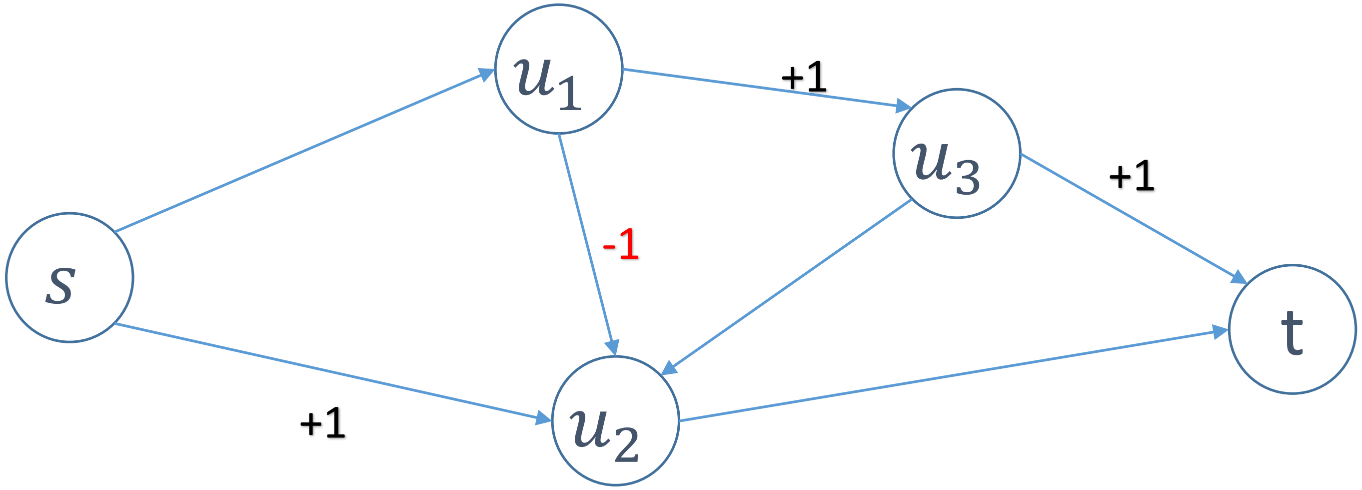
貪欲のアルゴリズム？

- 両者の流量の差に着目してみる。
- -1 はそれまでのフローを押し戻す操作に相等する。



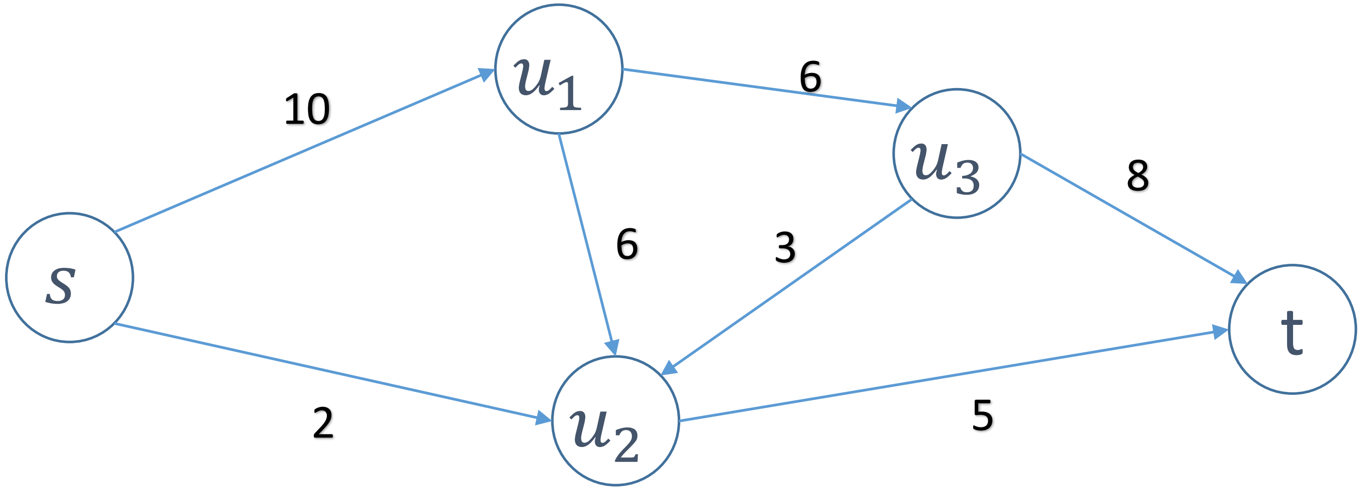
貪欲のアルゴリズム？

- 両者の流量の差に着目してみる。
- -1 はそれまでのフローを押し戻す操作に相当する。
- 流れている流量分だけ逆辺を追加して、フローを押し戻せるようにする。



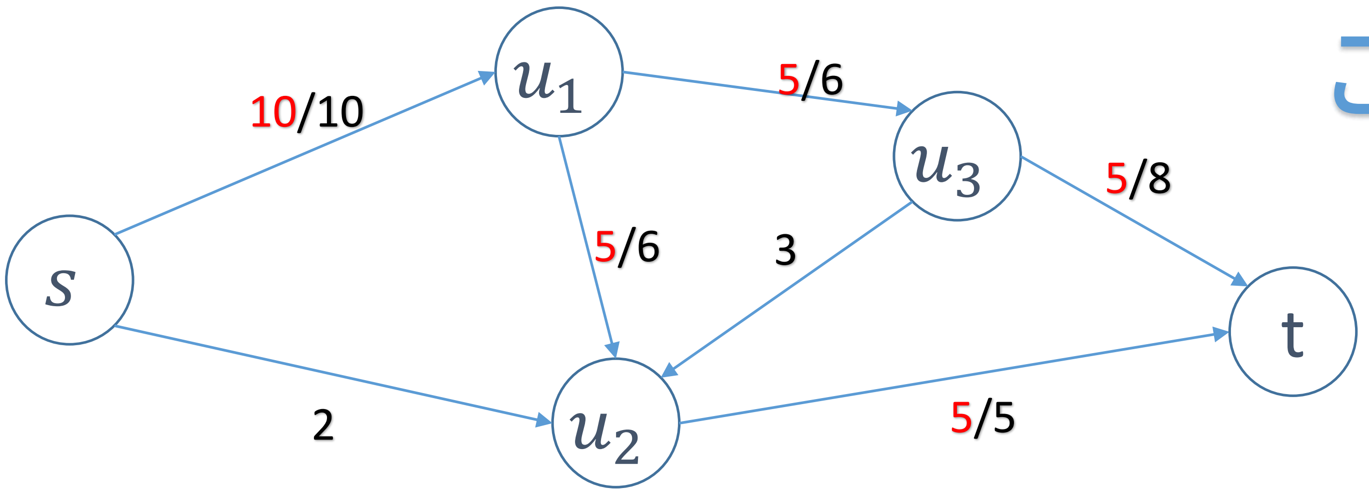
Ford-Fulkersonのアルゴリズム

- (1)流す余地のある辺と、流した容量分の逆辺のみを使って、s から t へのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



Ford-Fulkersonのアルゴリズム

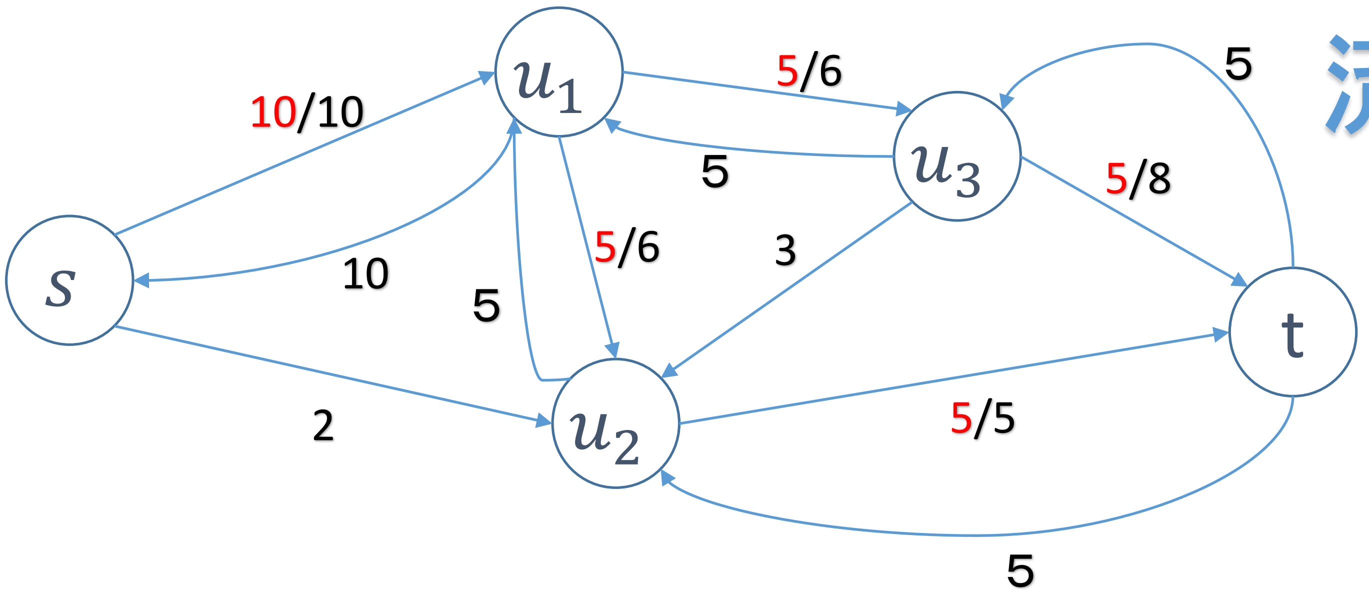
- (1)流す余地のある辺と、流した容量分の逆辺のみを使って、s から t へのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



この状態から

Ford-Fulkersonのアルゴリズム

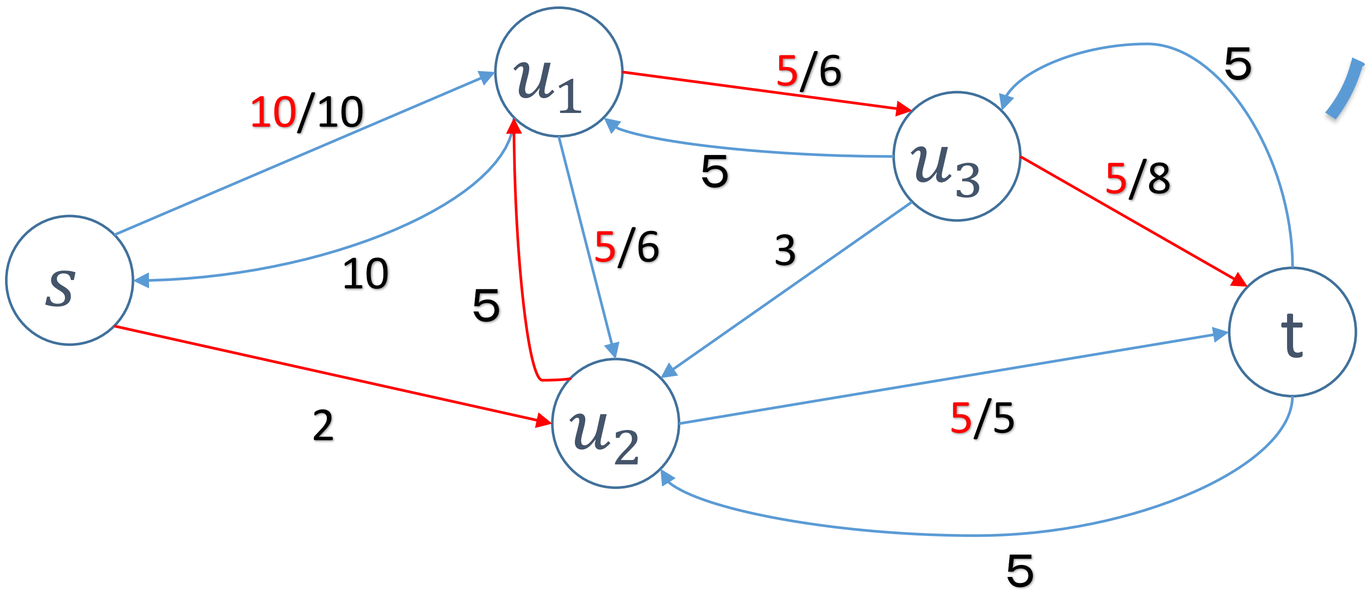
- (1)流す余地のある辺と、流した容量分の逆辺のみを使って、s から t へのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



流れている分だけ逆辺をはり

Ford-Fulkersonのアルゴリズム

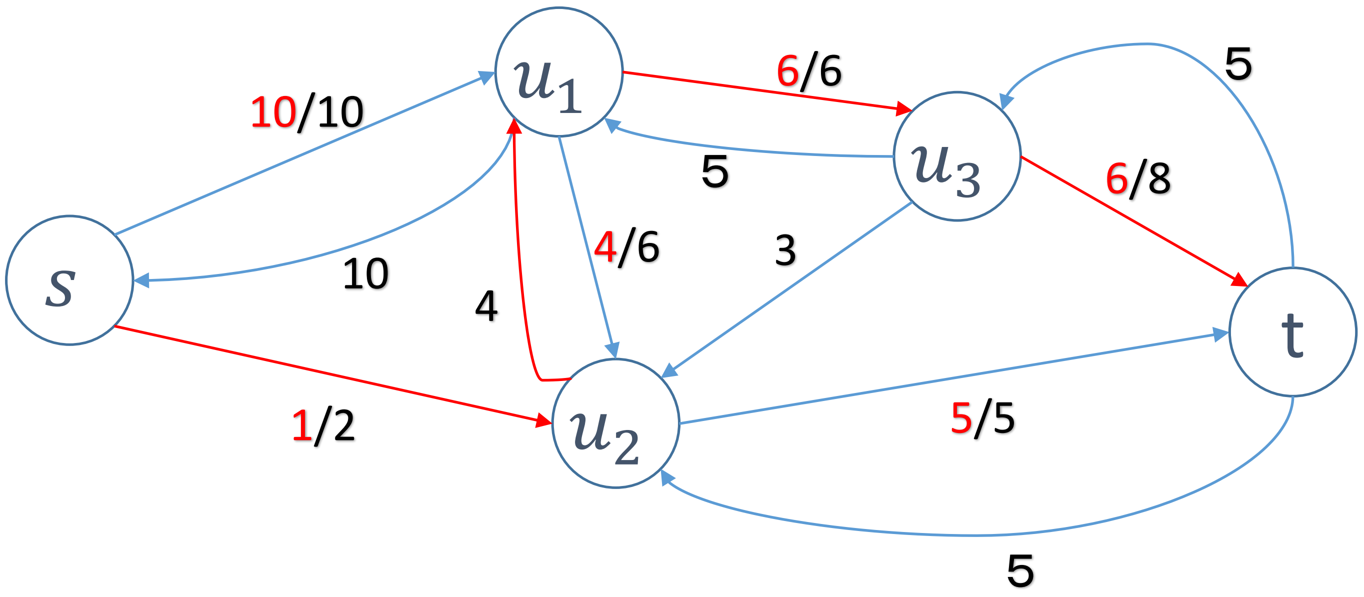
- (1)流す余地のある辺と、流した容量分の逆辺のみを使って、s から t へのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



パスを見つける！

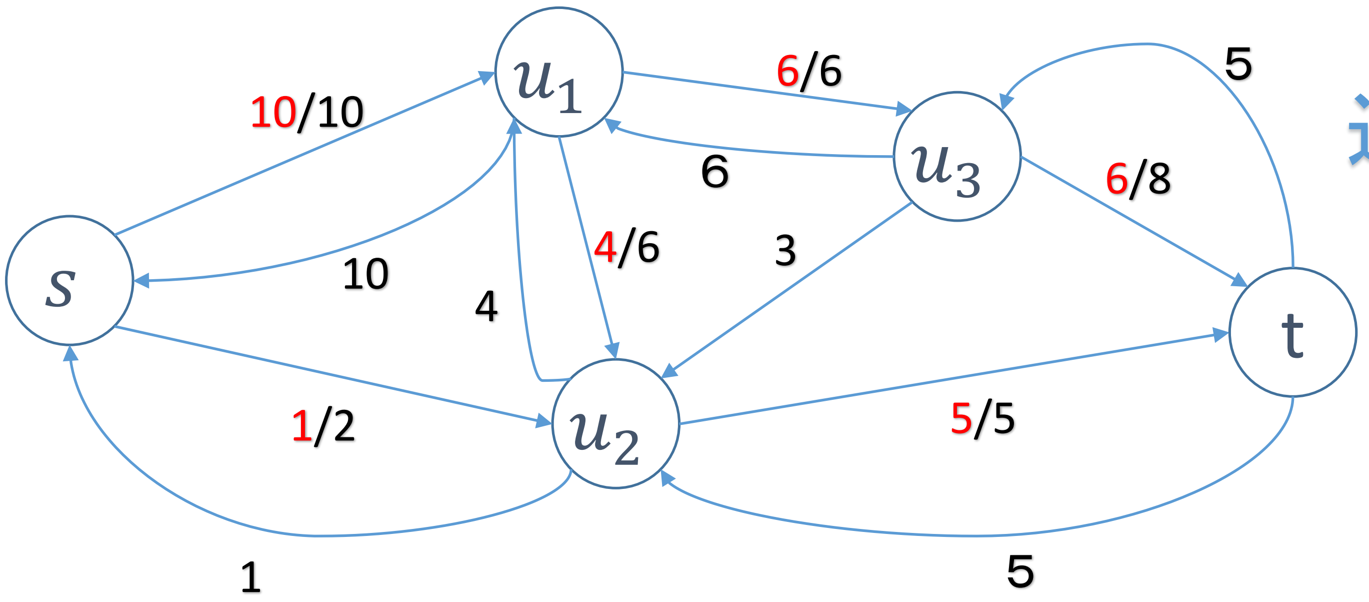
Ford-Fulkersonのアルゴリズム

- (1)流す余地のある辺と、流した容量分の逆辺のみを使って、s から t へのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



Ford-Fulkersonのアルゴリズム

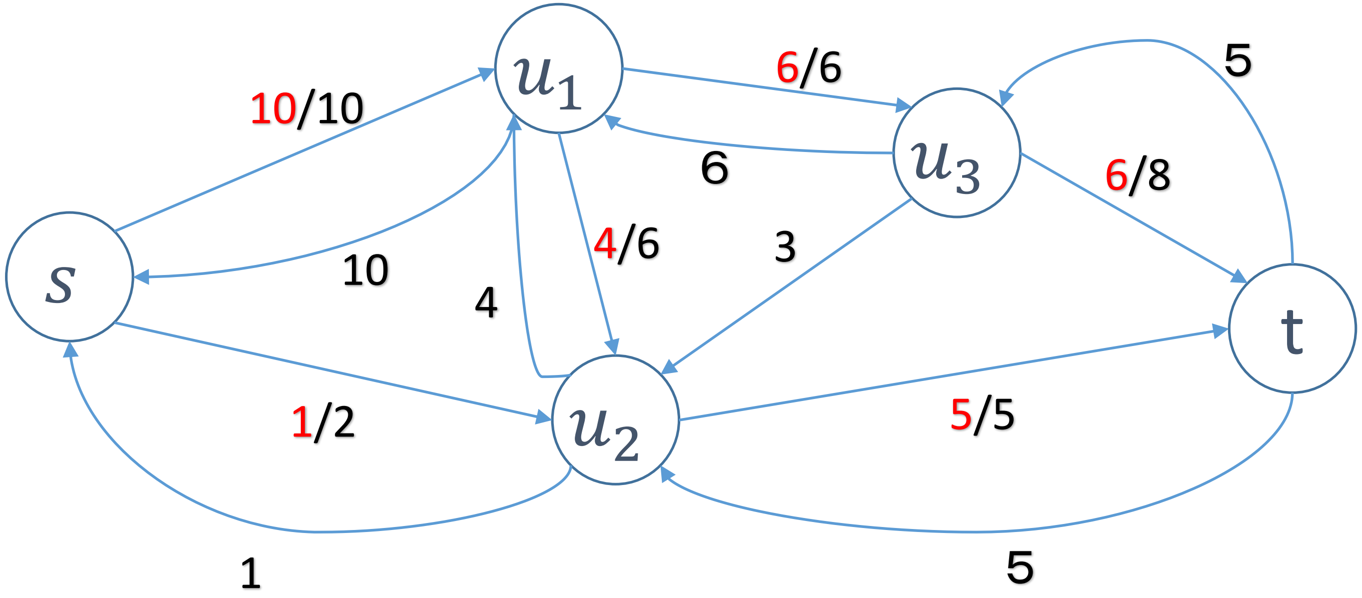
- (1)流す余地のある辺と、流した容量分の逆辺のみを使って、s から t へのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



逆辺を張りなおしても、
パスが見つからない

Ford-Fulkersonのアルゴリズム

- (1)流す余地のある辺と、流した容量分の逆辺のみを使って、s から t へのパスを見つける
- (2)見つかれば目一杯流して、(1)に戻る。見つからなければ終了



逆辺含めたグラフを残余グラフという

Ford-Fulkersonのアルゴリズム

- アルゴリズムの正当性は後程
- パスを見つけては更新の繰り返し
- DFS でパスを見つける
- 最大フローの流量を F とすると高々 F 回 DFS が行われる
- 最悪計算量は $O\big(F(|E| + |V|)\big)$

カットとは

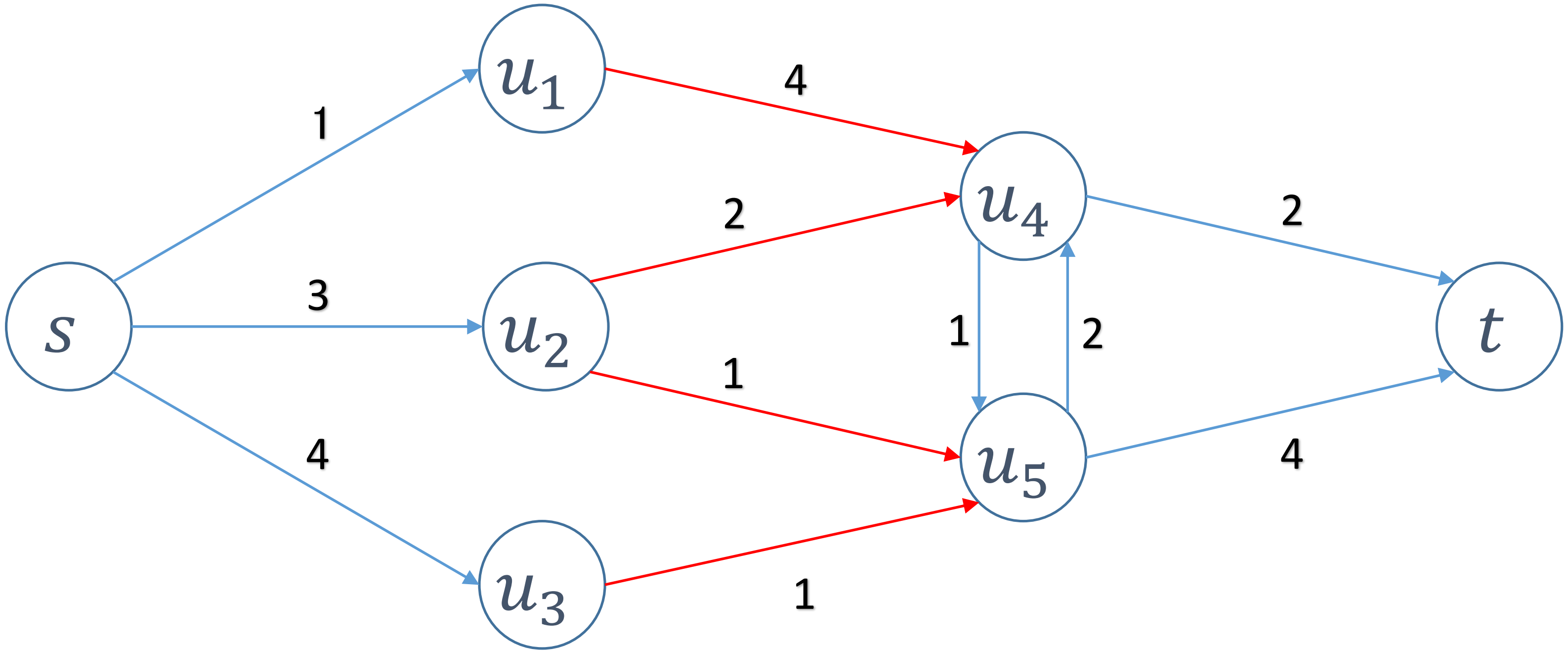
[定義]
頂点集合の任意の分割 $\mathrm{S}, \mathrm{V} \setminus S$ を考える。
S から $\mathrm{V} \setminus S$ に出ていく辺の集合をカットと言い、$(\mathrm{S}, \mathrm{V} \setminus S)$ と表す。

特に、$\mathrm{s} \in S$, $\mathrm{t} \in V \setminus S$のとき$s - t$ カットという
s-t カットに含まれる辺全てを削除すると、$s - t$パスがなくなる
カットに含まれる辺の容量の総和をカットの容量という
最小カットとは、任意の $s - t$ カットのうち、カットの容量が最小のもの

カットとは

C $= \{ (u_1, u_4), (u_2, u_4), (u_2, u_5), (u_3, u_5) \}$ はカット?　はい。

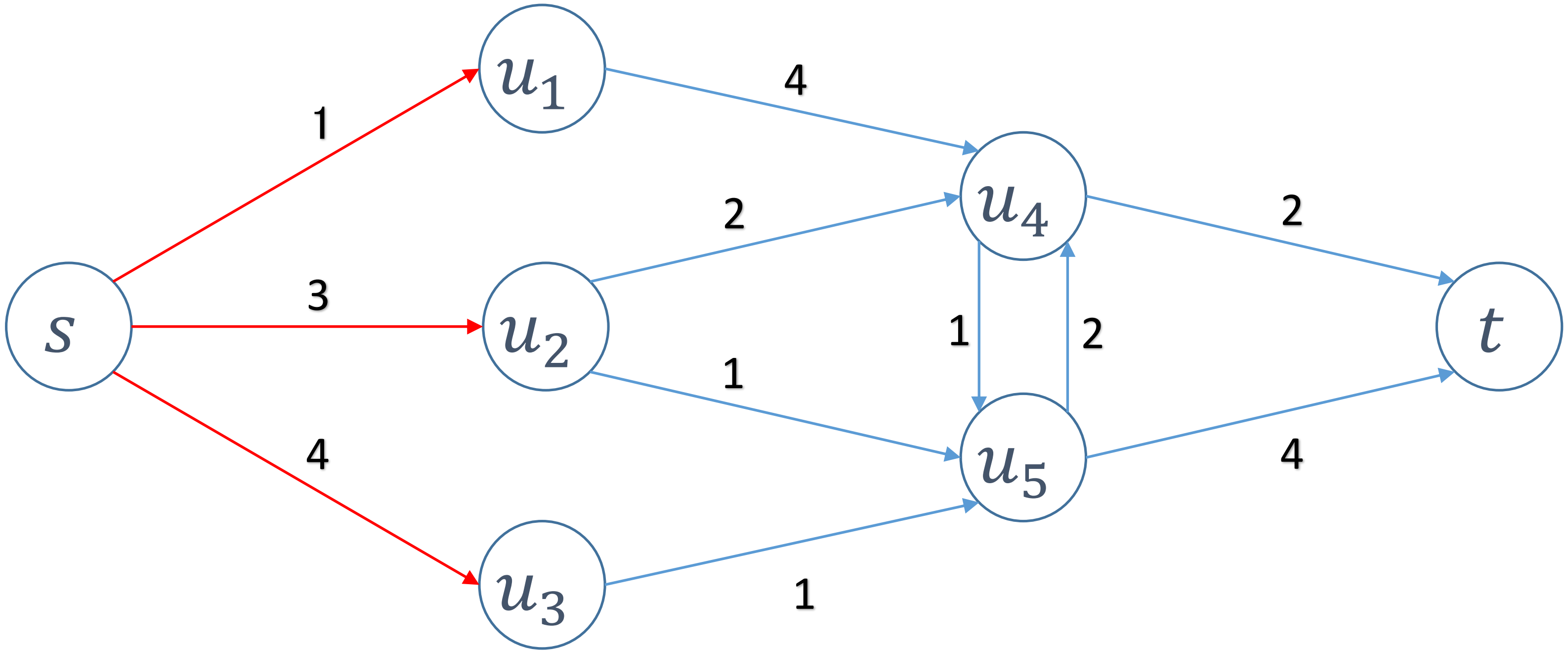
カットの容量は?　8です。



カットとは

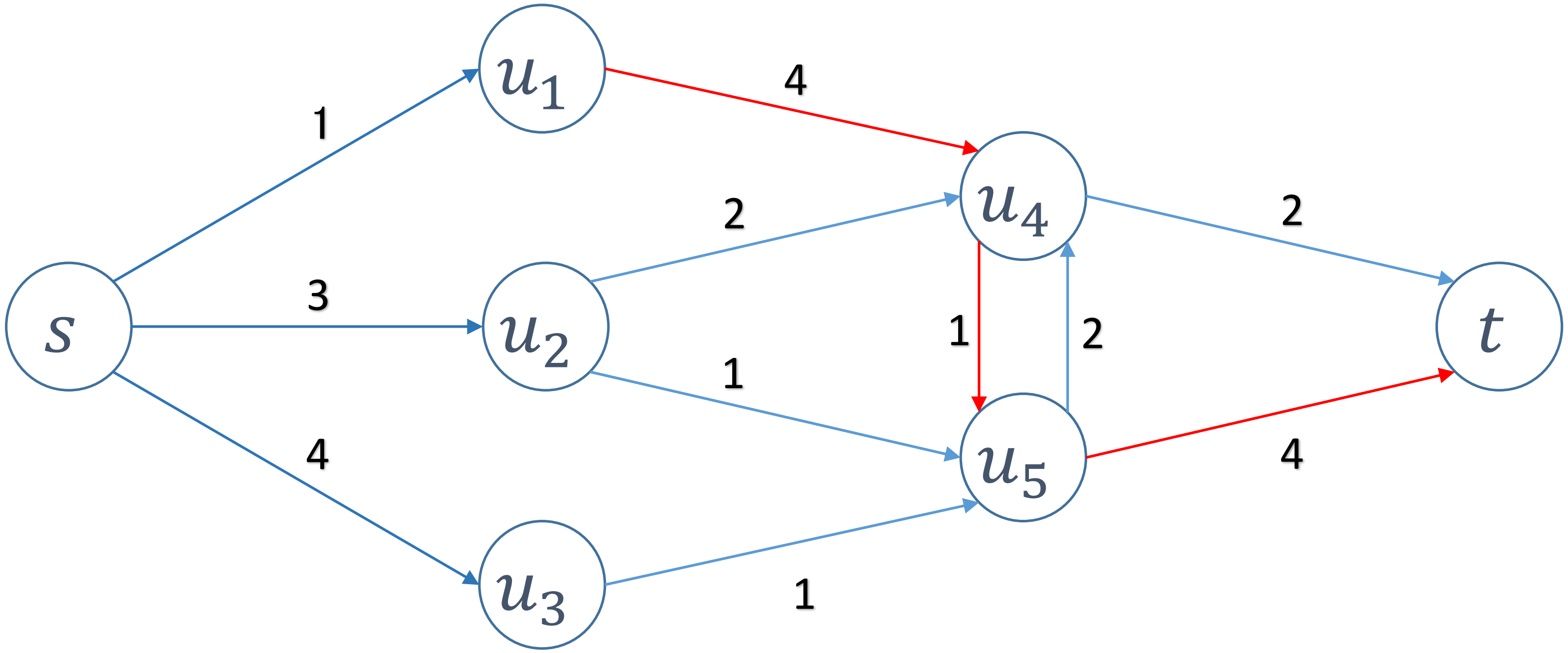
C = $\{\,(s,u_1),(s,u_2)\ (s,u_3)\,\}$ はカット?　はい。

カットの容量は?　8です。



カットとは

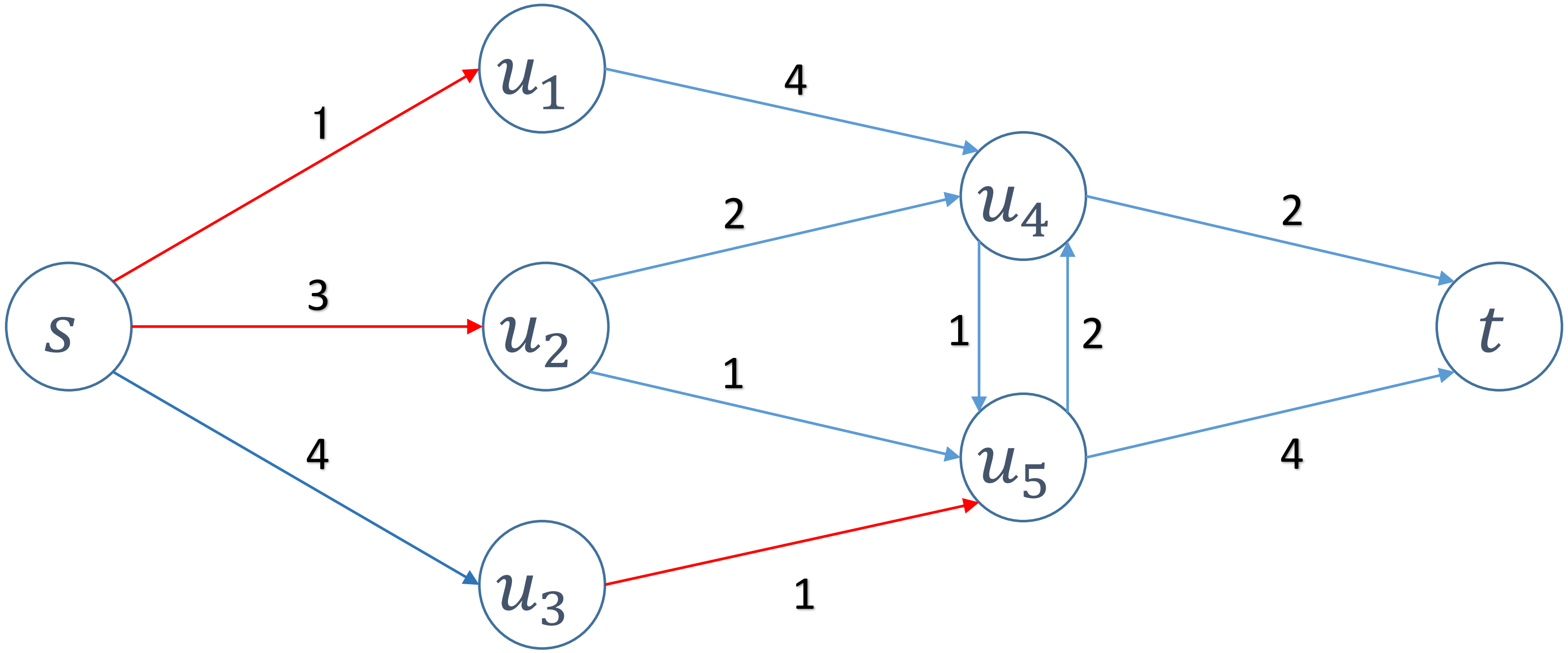
$C = \{\,(u_1, u_4), (u_4, u_5)\ (u_5, t)\,\}$ はカット? いいえ。



カットとは

C = $\{\,(s, u_1), (s, u_2)\ (u_3, u_5)\,\}$ はカット? はい。

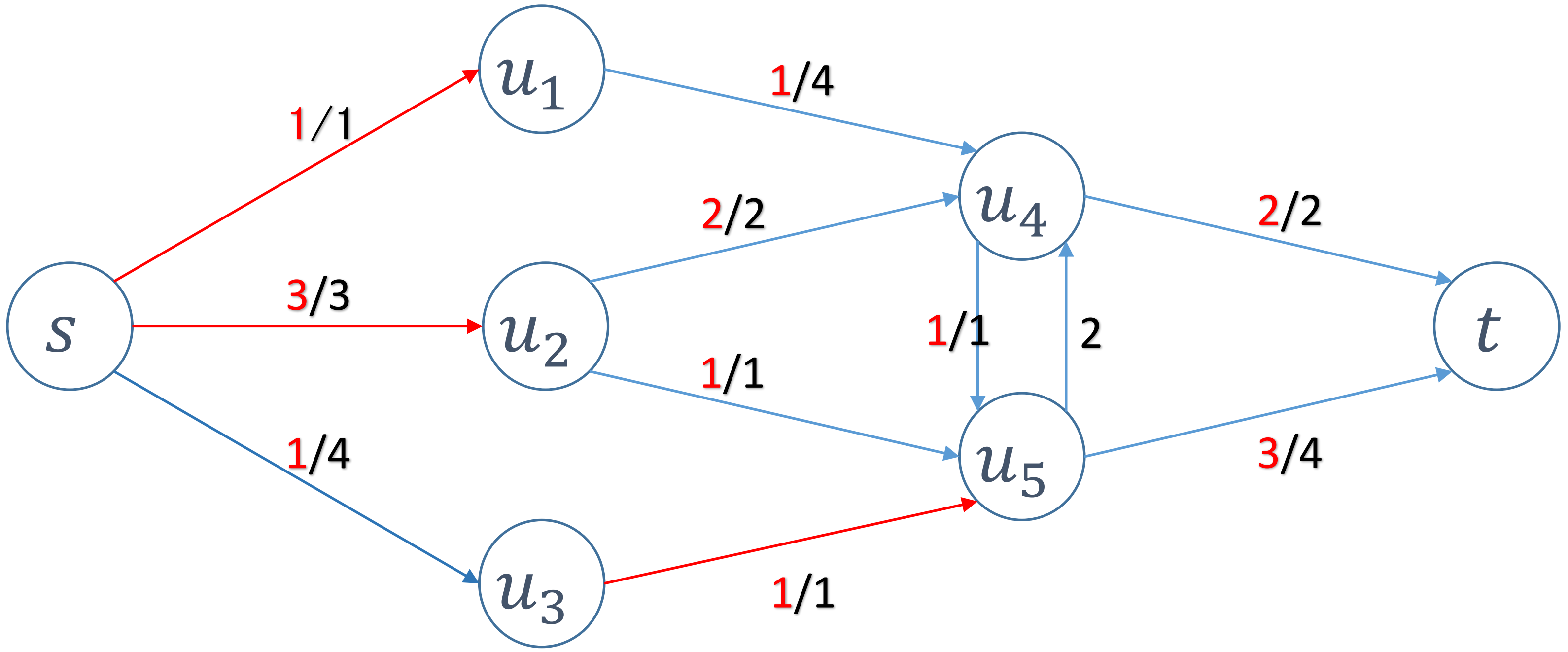
カットの容量は? 5です。



最大フロー・最小カットの定理

[定理]

任意のネットワークにおいて、最大フローの値は、最小カットの容量に等しい



証明

- 証明したいことは二つ
- Ford-Fulkerson のアルゴリズムの正当性
- 最大フロー・最小カットの定理

証明

[証明の流れ]

1) 任意のフロー f の流量 $\leq$ 最小カットの容量

2) Ford-Fulkerson で得られたフロー f'の流量 = 最小カットの容量

この二つを示すことができれば、

f' が最大フローであり、同時に最大フロー・最小カットの定理を示せる。

任意のフロー f の流量 ≤ 最小カットの容量

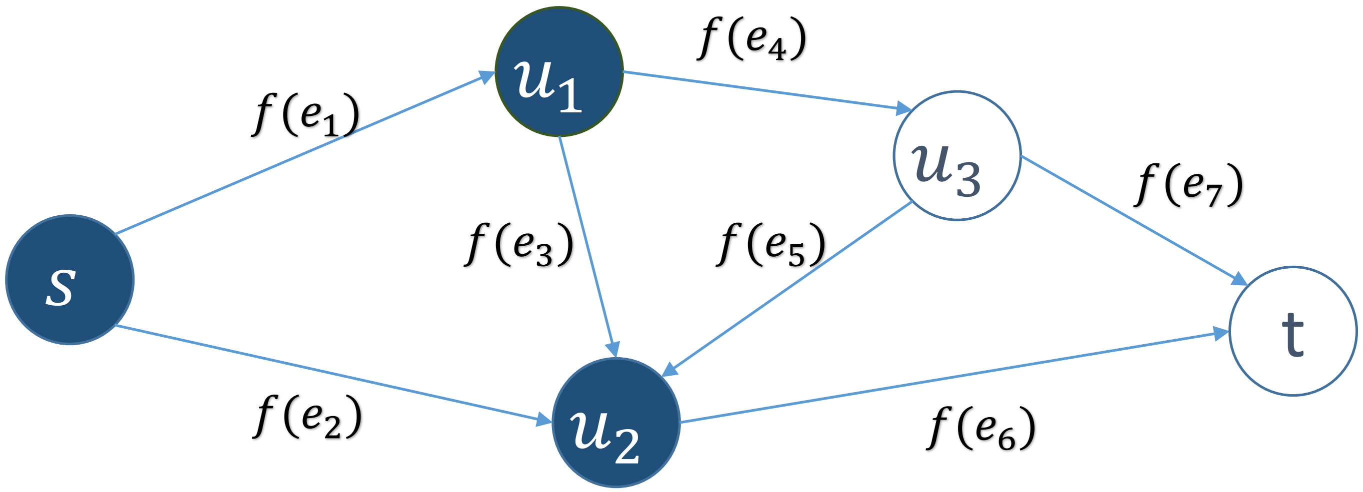
任意のフロー fと任意のカット (S, V\S) を考える

(fの流量) = $\sum_{e \in \delta_+(s)} f(e)$ であり、

$v \in S \setminus \{s\}$ に対して $\sum_{e \in \delta_-(v)} f(e) = \sum_{e \in \delta_+(v)} f(e)$であるので

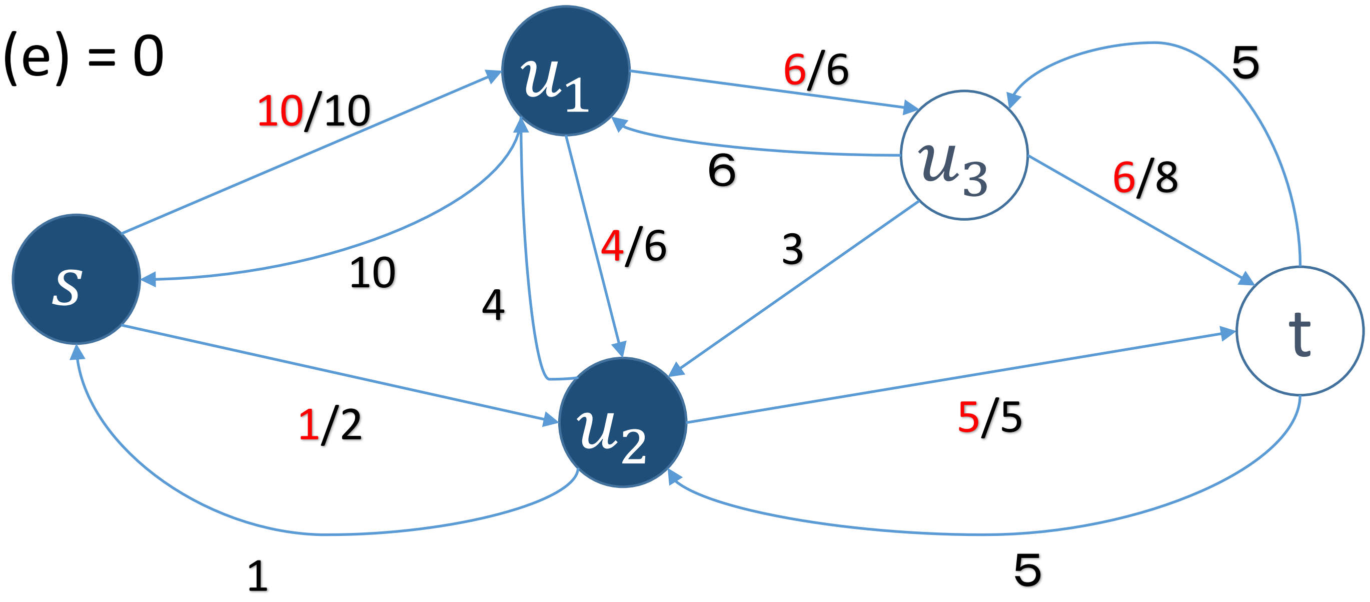
(fの流量) = (Sから出ていく辺の流量) – (Sに入ってくる辺の容量)

≤ (カットの容量)



f’の流量 = 最小カットの容量

f’ に対する残余グラフにおいて、
s-vパスの存在するような頂点 v からなる集合を S とする。
f’ の残余グラフにおいてs-tパスが存在しないので、(S, V\S)は s-t カットとなる。
また, S の性質より
S から V \ Sに向かう辺 e について f’(e) = c(e)
V \ S から S に向かう辺 e について f’(e) = 0
したがって
(f’の流量) = (Sから出る辺の流量)
- (Sに入る辺の流量)
= (カットの容量)



実装例(C++)

```cpp
#include<bits/stdc++.h>
using namespace std;
#define MAX_V 1000000
#define INF 1e9

//辺を表す構造体 (行先、容量、逆辺の場所(id))
struct edge{
    int to, cap, rev;
};

vector<vector<edge> > G(MAX_V);        //グラフの隣接リスト表現
bool used[MAX_V];                      //DFSですでに調べたかのグラフ


// from から to へ向かう容量 cap の辺をグラフに張る
void add_edge(int from, int to, int cap){

    edge F;
    edge T;
    F.to = to;
    F.cap = cap;
    F.rev = G[to].size();
    G[from].push_back(F);

    T.to = from;
    T.cap = 0;
    T.rev = G[from].size() - 1;
    G[to].push_back(T);
}
```

```cpp
//増加パスをDFSで探す
int dfs(int v, int t, int f){
    if(v == t) return f;
    used[v] = true;
    for(int i = 0; i < G[v].size(); i++){
        edge &e = G[v][i];                   //辺の容量を更新するのでアドレスを持ってくる
        if(!used[e.to] && e.cap > 0){
            int d = dfs(e.to, t, min(f, e.cap));
            if(d > 0){
                e.cap -= d;                  //使った分容量を減らす
                G[e.to][e.rev].cap += d;     //使った分逆辺の容量を増やす
                return d;
            }
        }
    }
    return 0;
}

//s から t への最大流を求める
int max_flow(int s, int t){
    int flow = 0;
    while(1){
        memset(used, 0, sizeof(used));
        int f = dfs(s, t, INF);
        if(f == 0) return flow;
        flow += f;
    }
}
```

目次

- 最大フロー、最小カットのお話
- 競プロのお話

競プロの話

- フローのアルゴリズムは複雑なので、ライブラリ化するのがおすすめ
- 最大流を計算するクラスを用意
- メンバ関数は以下の二つ

- add_edge(int from, int to, int cap) := 辺を追加する
- max_flow(int s, int t) := s から t への最大フロー値を求める

競プロの話

ブラックボックス

```
int main(){

    int n, m; cin >> n >> m;
    Dinic g(n);
    for(int i = 0; i < m; i++){
        int u, v, c; cin >> u >> v >> c;
        g.add_edge(u, v, c);
        //g.add_edge(u - 1, v - 1, c);
    }

    cout << g.max_flow(0, n - 1) << endl;

    return 0;
}
```

main関数はたったの14行！？

身も蓋もない話

- 最大流問題を解くアルゴリズムはFord-Fulkerson法のほかに Dinic法が知られている。
- Dinic法のオーダーはO(E V^2) だが、実際には爆速に動く
- こっちを使っているひとが多いように感じる

問題の解き方

- 考察する
- あ、これフローっぽいな or カットっぽいな
- グラフをいい感じに作る
- ライブラリをペタリ
- AC

問題(KUPC2014 H)

https://qupc2014.contest.atcoder.jp/tasks/qupc2014_h

[問題概要]

M 頂点 N 辺 の重みつき有向グラフと整数 P、さらに頂点番号 $L_0, L_1, \ldots L_{G-1}$ が与えられる。各頂点には魔法使いが一人づつ存在し、L_i $(0 \leq i \leq G-1)$ にいる魔法使いは無制限に魔力を得ることができる。各辺の重みを超えないようにうまく魔力を伝達した時に、頂点 0 に P 以上の魔力を送ることができるかどうかを答えよ

[制約]

$1 \leq N \leq 500$

$1 \leq M \leq 500$

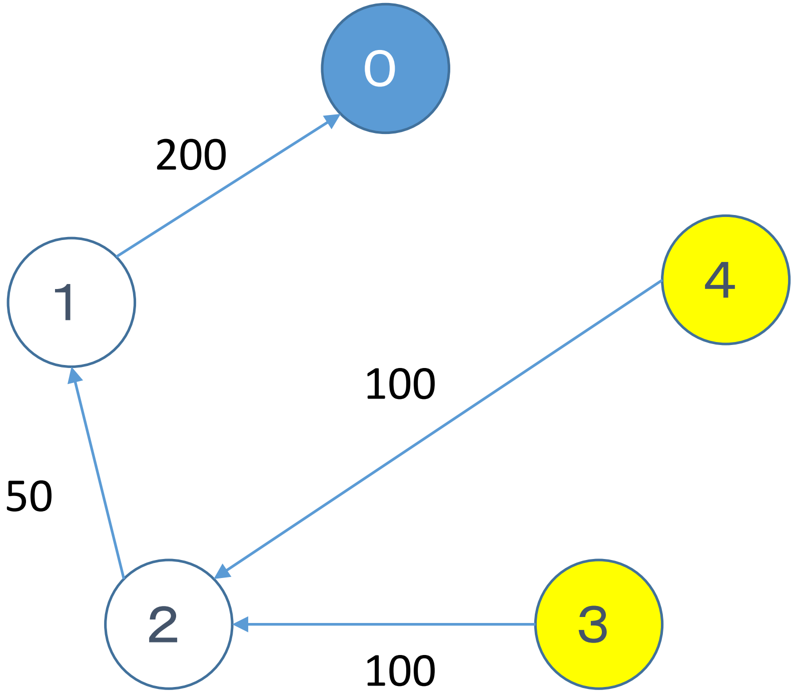
考察

最大フローを求める問題に帰着できそう

頂点が複数設定される。どう処理するか？

入力例3

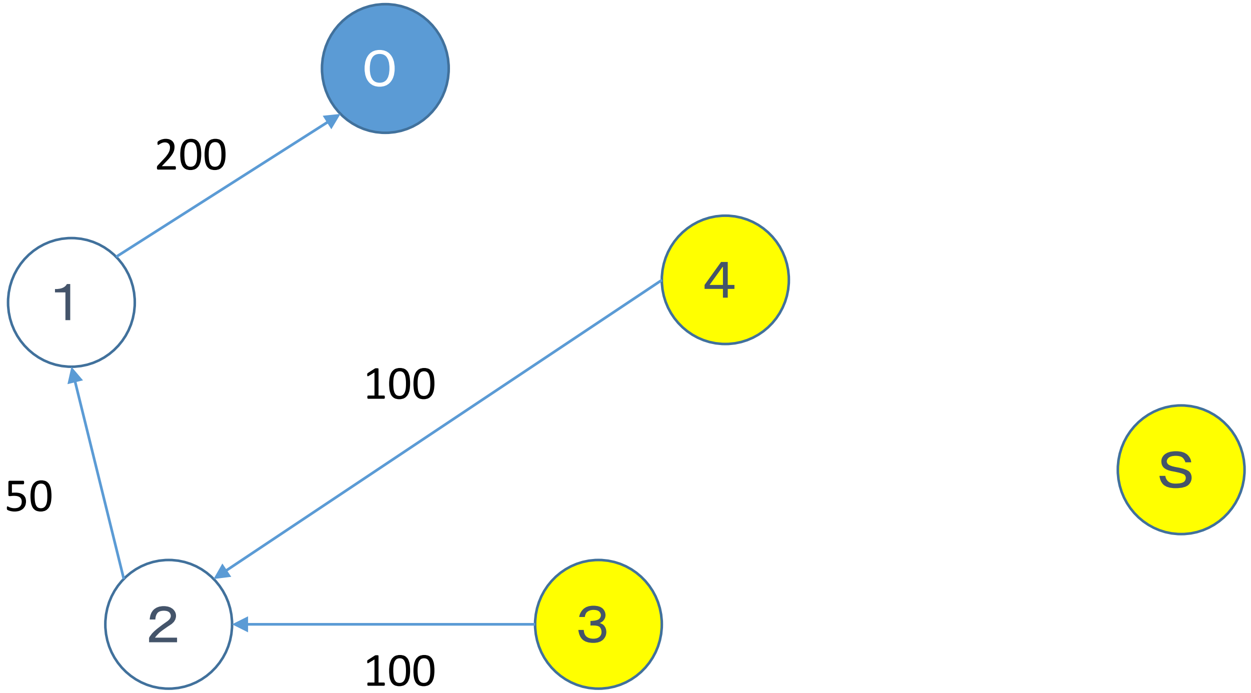
4 5 100 2
3 4
1 0 200
2 1 50
3 2 100
4 2 100



考察

図のように架空の s 頂点を追加して…

入力例3
4 5 100 2
3 4
1 0 200
2 1 50
3 2 100
4 2 100



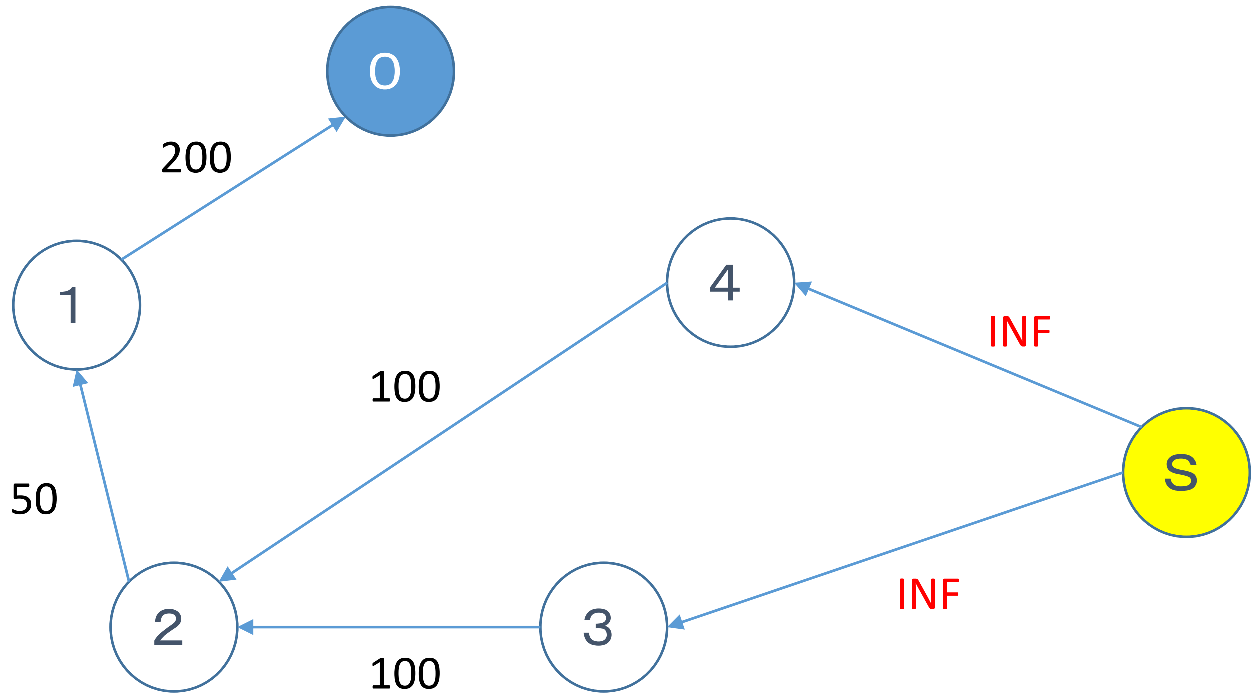
考察

図のように架空の s 頂点を追加して…

容量 INF の辺を張り…

入力例3

4 5 100 2
3 4
1 0 200
2 1 50
3 2 100
4 2 100



考察

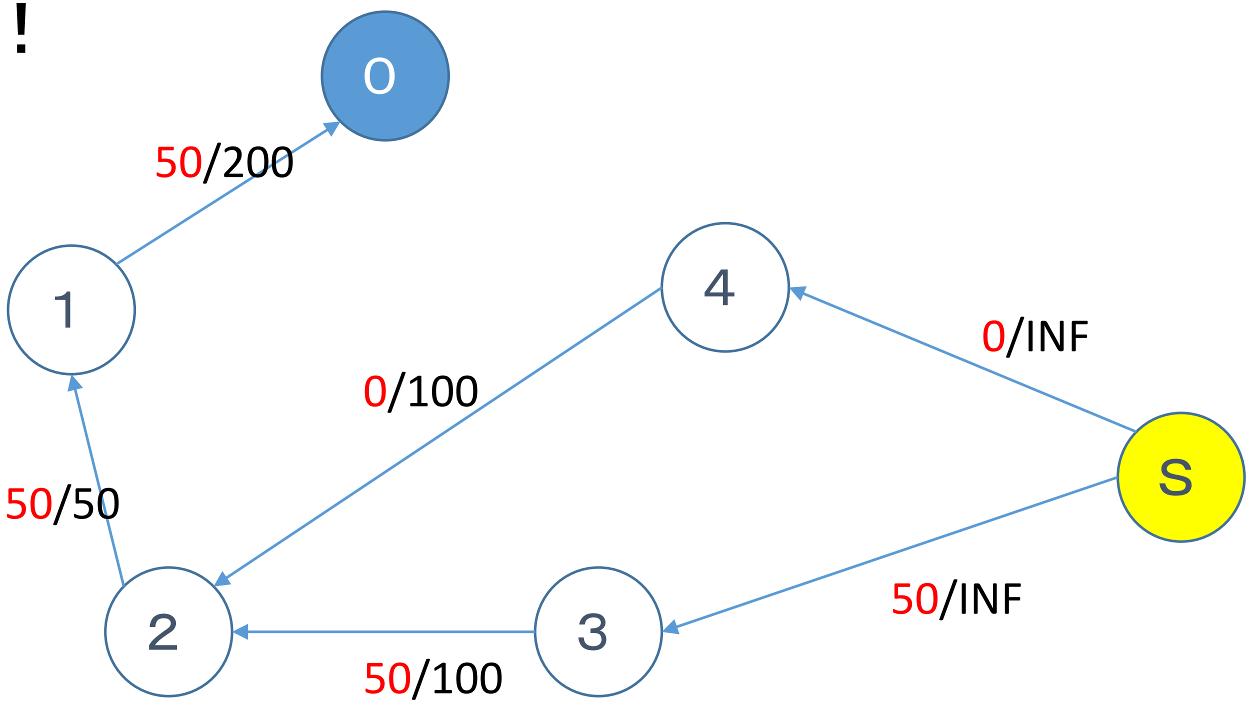
図のように架空の s 頂点を追加して…

容量 INF の辺を張り…

最大フローを流せばよい！

入力例3

4 5 100 2
3 4
1 0 200
2 1 50
3 2 100
4 2 100



実装例 (C++)

```cpp
int main(){

    //入力受け取り
    int n, m, p, g; cin >> n >> m >> p >> g;

    //アスナ := 0. 架空の源泉 := m
    Dinic G(m + 1);
    for(int i = 0; i < g; i++){
        int l; cin >> l;
        G.add_edge(m, l, INF);
    }

    for(int i = 0; i < n; i++){
        int u, v;
        long long c;
        cin >> u >> v >> c;
        G.add_edge(u, v, c);
    }

    if(G.max_flow(m, 0) >= p) cout << "Yes" << endl;
    else cout << "No" << endl;
    return 0;
}
```

問題 (ABC010 D)
https://beta.atcoder.jp/contests/abc010/tasks/abc010_4

[問題概要]

SNS の友人関係を表す無向グラフが与えられる。

なぎさちゃんは高橋君に自分以外の女と連絡を取ってほしくない。

なぎさちゃんは以下のような工作ができる。

- 友人関係を一つ解消
- 一人のパスワードを変更し、メッセージを閲覧不可能にする

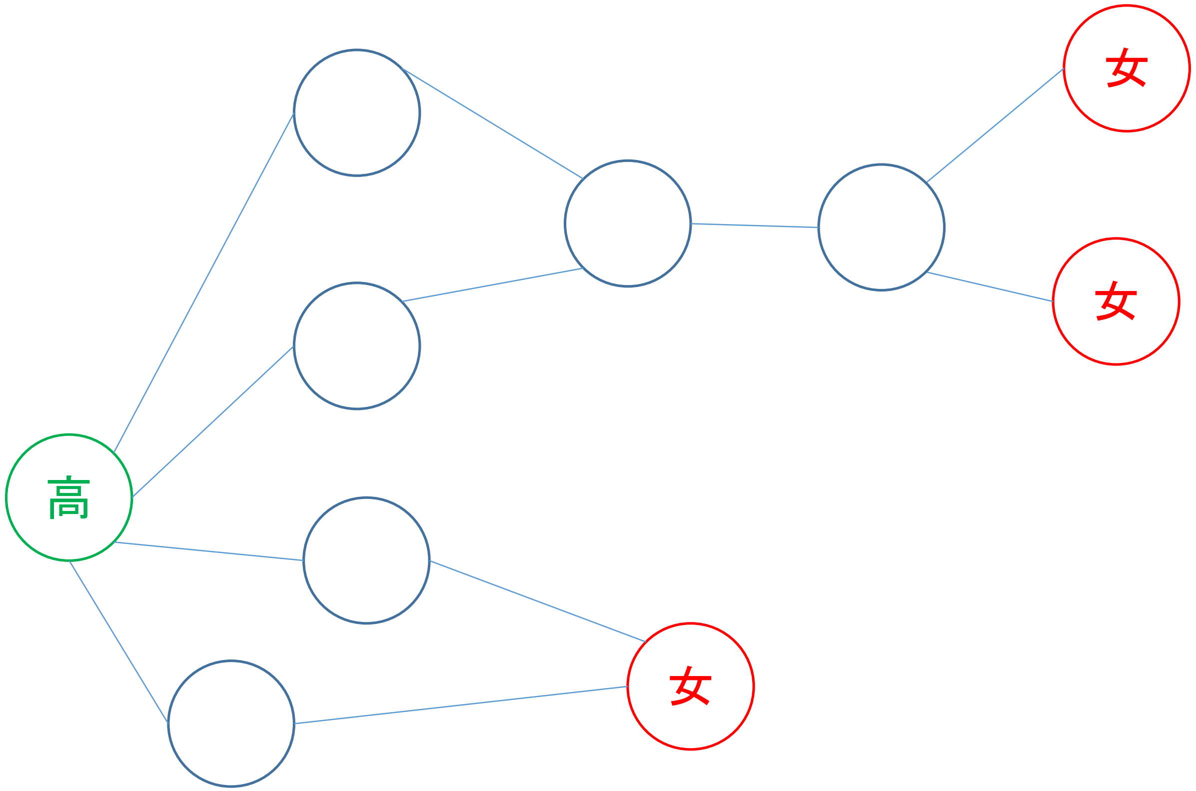
なぎさちゃんが工作を行う回数の最小値を出力してください

[制約]

$1 \leq V \leq 100$

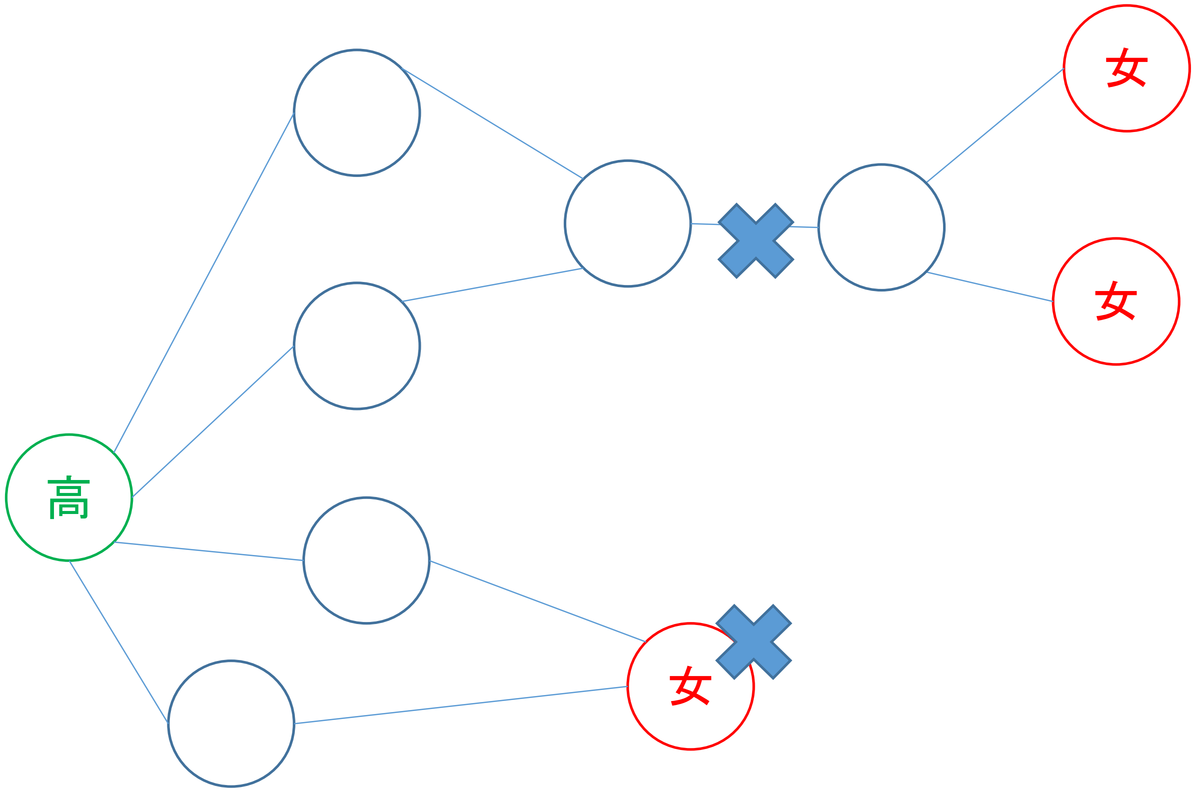
考察

- このような図で考えてみる



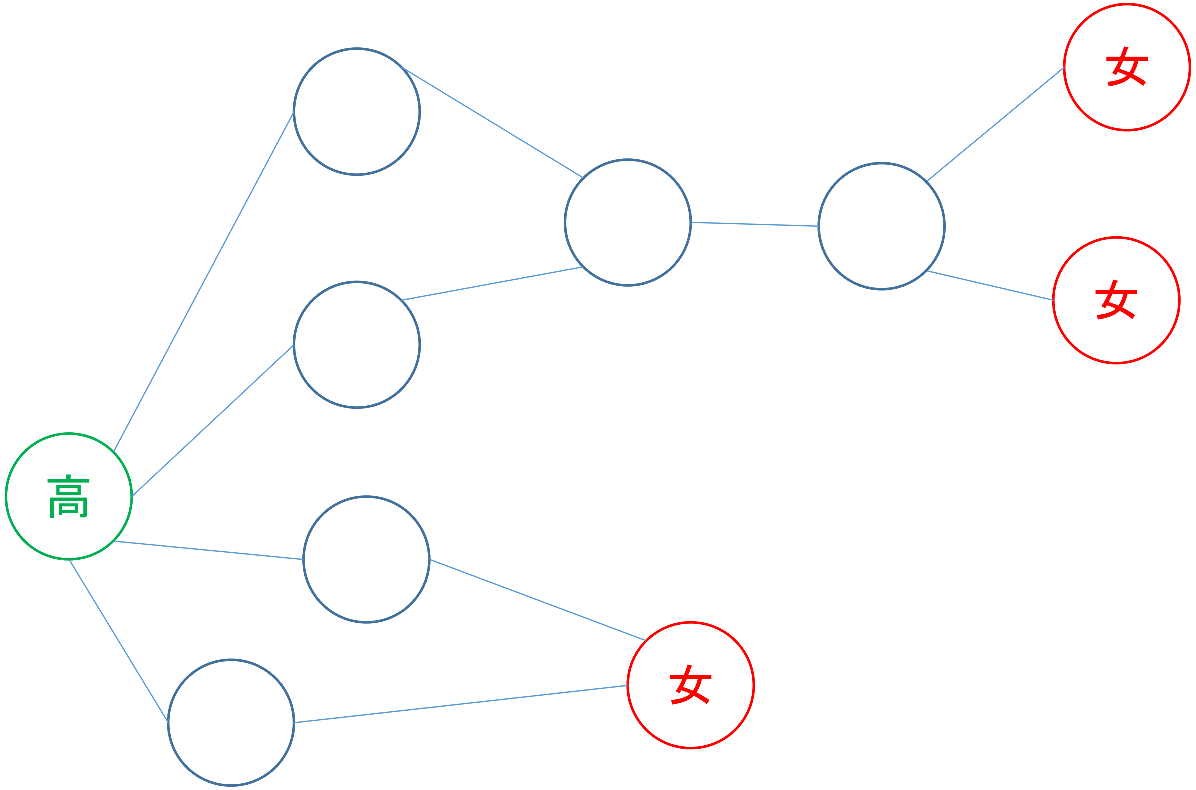
考察

- 最適解は 2



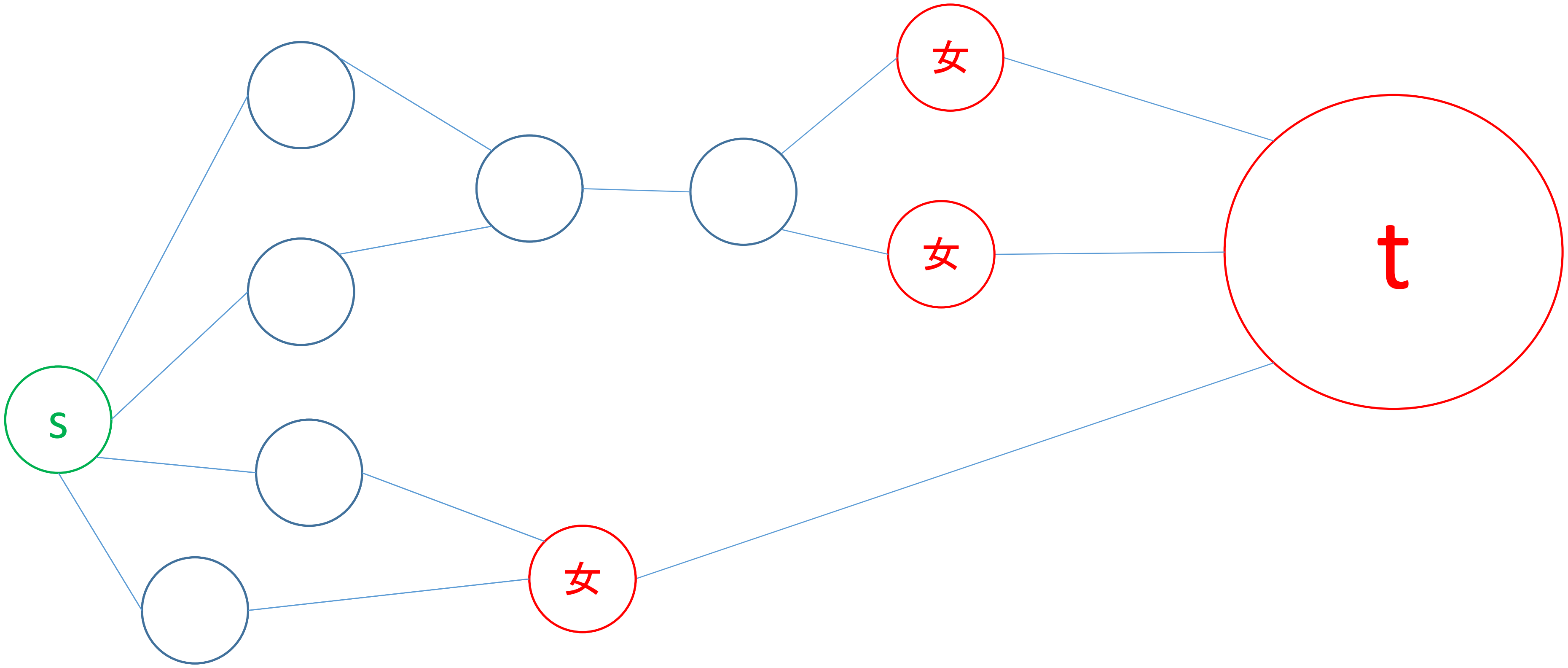
考察

- 工作のパターンが二種類あるのが面倒



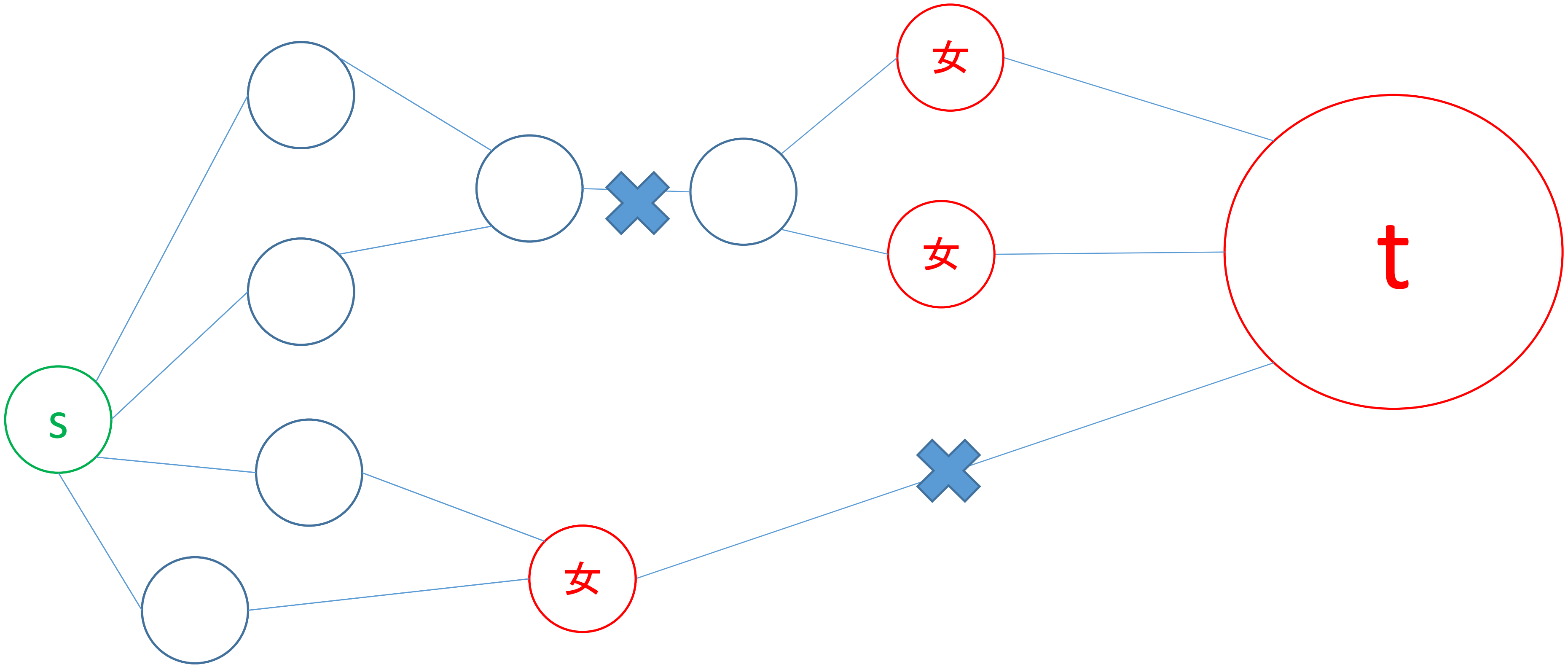
考察

- 図のように架空の頂点を追加すると、辺の削除だけを考えることができる



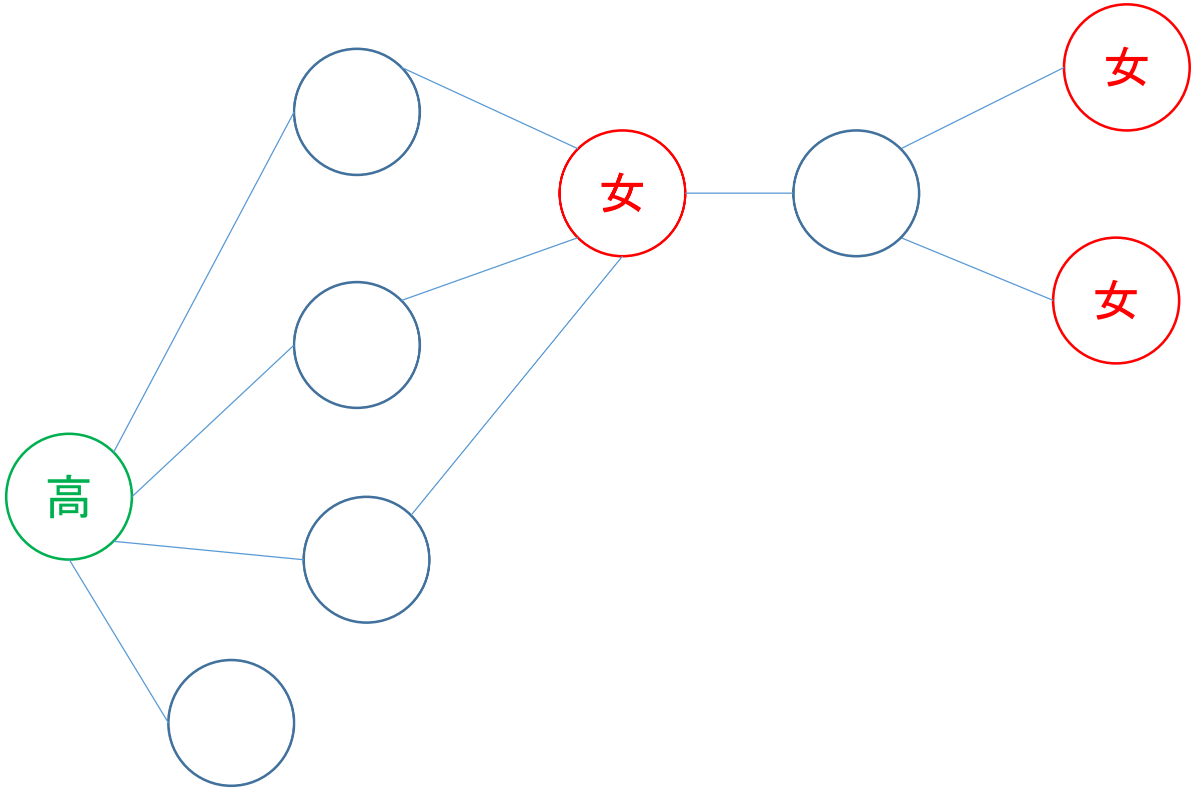
考察

- 最小カットに帰着できた！！



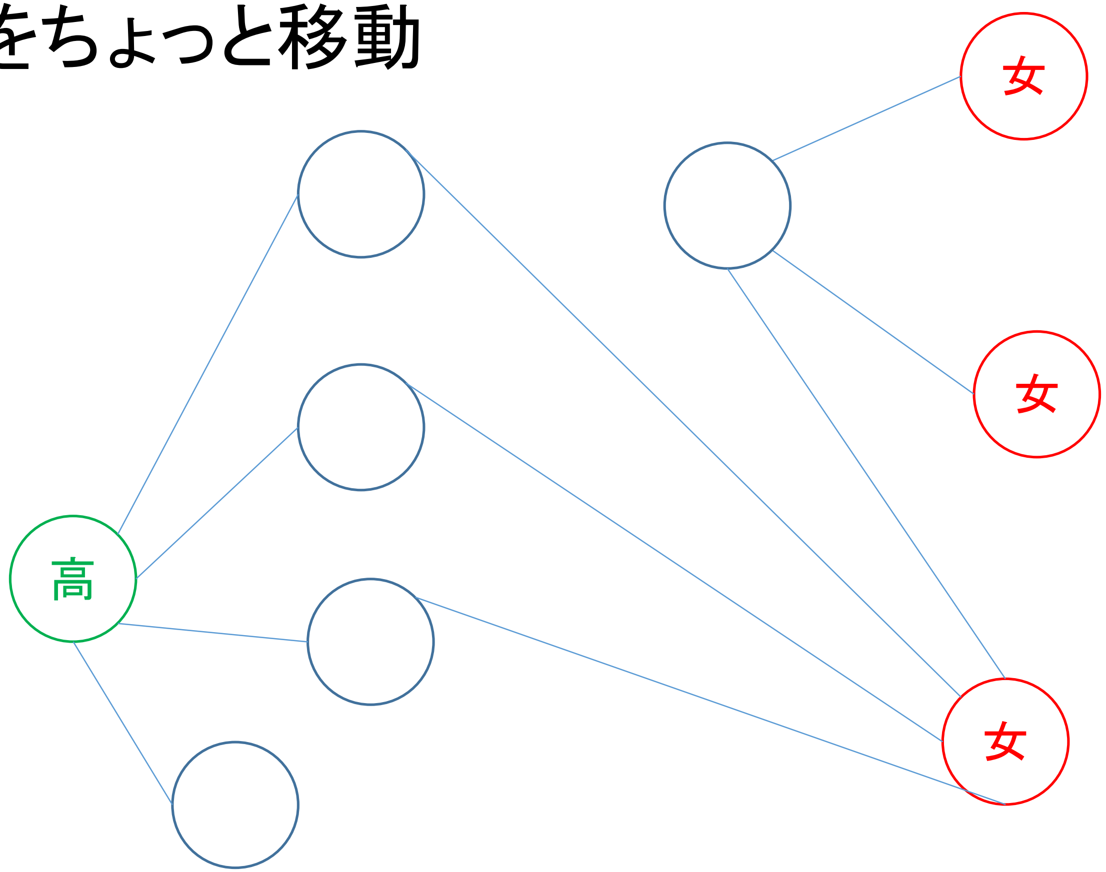
考察

- 別の例



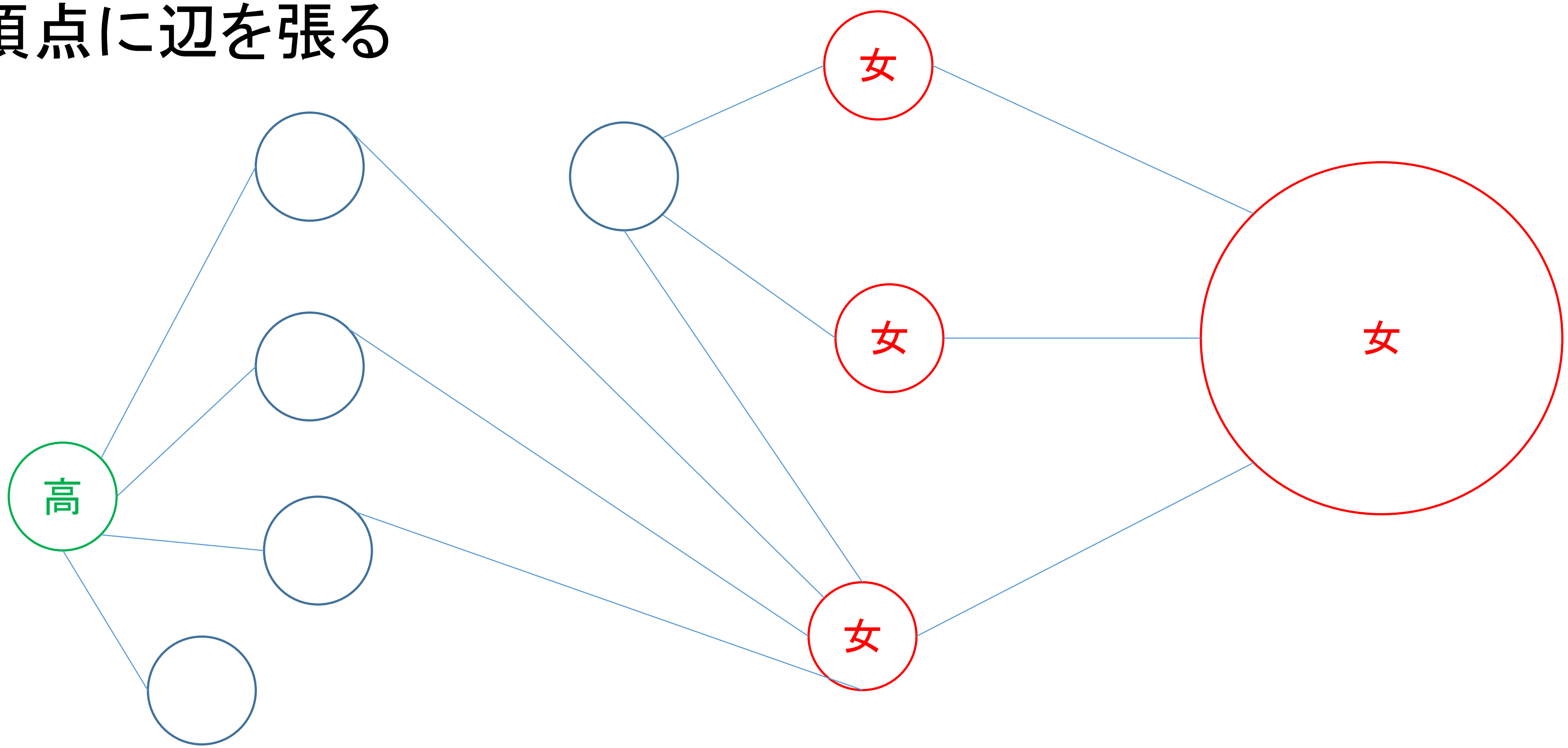
考察

- グラフをちょっと移動



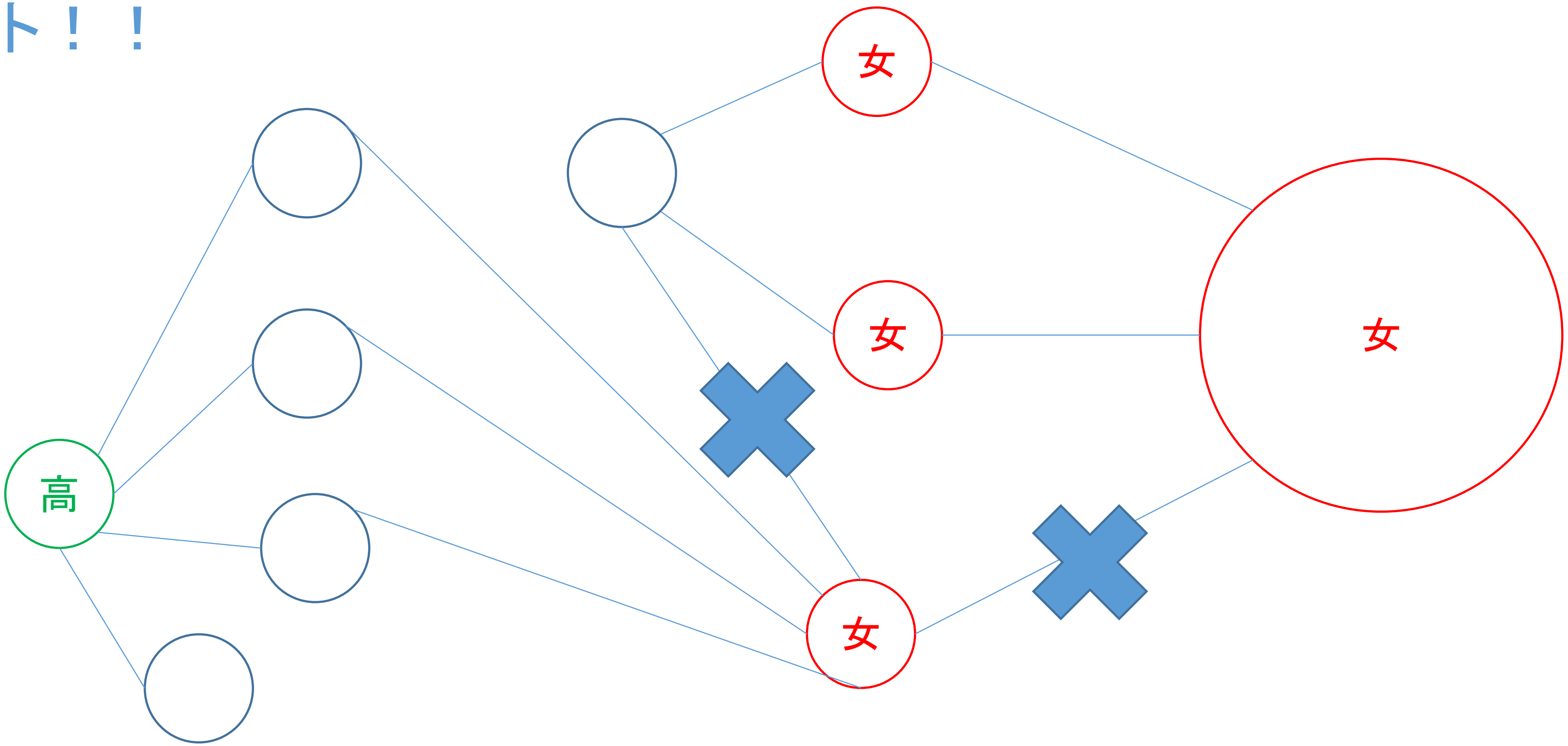
考察

架空の頂点に辺を張る



考察

最小カット！！



実装例(C++)

```
int main(){

    int n, g, e; cin >> n >> g >> e;

    //高橋君 := 0, ゴール := n
    Dinic G(n + 1);

    //女の子からゴールへ
    for(int i = 0; i < g; i++){
        int p; cin >> p;
        G.add_edge(p, n, 1LL);
    }

    //辺を追加
    for(int i = 0; i < e; i++){
        int a, b; cin >> a >> b;
        G.add_edge(a, b, 1LL);
        G.add_edge(b, a, 1LL);
    }

    cout << G.max_flow(0, n) << endl;
    return 0;
}
```

s ノードと t ノードに何番を割り振るかを定義してしまう。

s := 0, t := n

問題 (ACPC 2018 I)

https://onlinejudge.u-aizu.ac.jp/beta/room.html#ACPC2018Day2/problems/I

[問題]

文字列のリストが与えられ、しりとりを以下のルールで行う。

1) まず最初に、リストの中から好きな文字列を一つ選び、その文字列をリストから除外する。

2) 続いて、一つ前に選んだ文字列の最後の一文字が、最初の一文字である文字列をリストから一つ選ぶ。

3) 選んだ文字列をリストから除外する。

この後、2., 3.を交互に繰り返すことになる。

さて、このしりとりを2.でリストから選べる文字列がなくなるまで続けたとしよう。このときに、最後に選んだ文字列の「最後の一文字」としてあり得る文字をすべて列挙したい。

問題 (ACPC 2018 I)

https://onlinejudge.u-aizu.ac.jp/beta/room.html#ACPC2018Day2/problems/I

[制約]

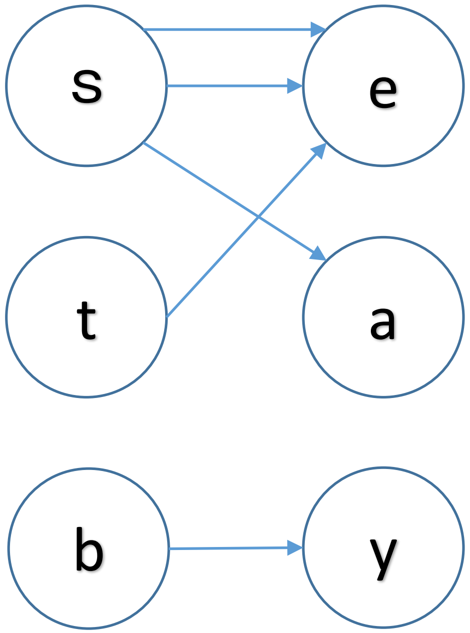
$1 \leq N \leq 10^4$

$1 \leq |s_i| \leq 100$

考察

与えられた入力をグラフにしてみる(自己ループは無視)

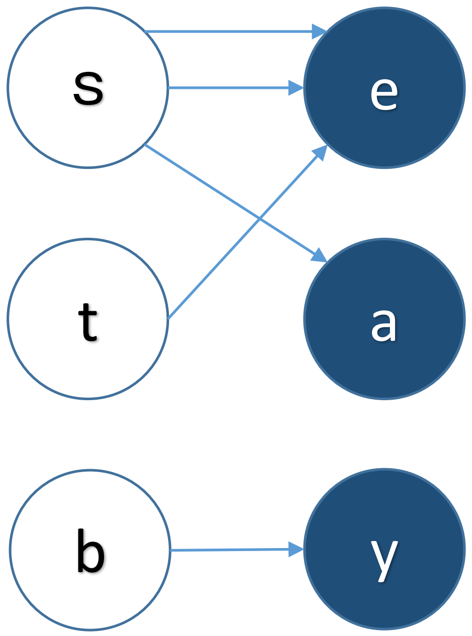
[入力]
7
she
sells
sea
shells
by
the
seashore



考察

与えられた入力をグラフにしてみる(自己ループは無視)

[入力]
7
she
sells
sea
shells
by
the
seashore



考察

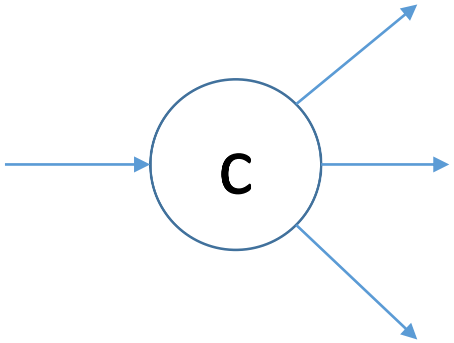
- 各文字 (a – z) に対してしりとりの最後になりうるかを判定したい
- 入次数と出次数に着目？

考察

- 各文字 (a – z) に対してしりとりの最後になりうるかを判定したい
- 入次数と出次数に着目？

(入次数) < (出次数)のとき

NG!

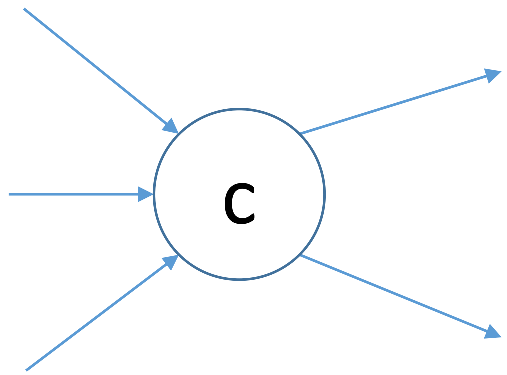


考察

- 各文字 (a – z) に対してしりとりの最後になりうるかを判定したい
- 入次数と出次数に着目？

(入次数) ≥ (出次数)のとき

OK?

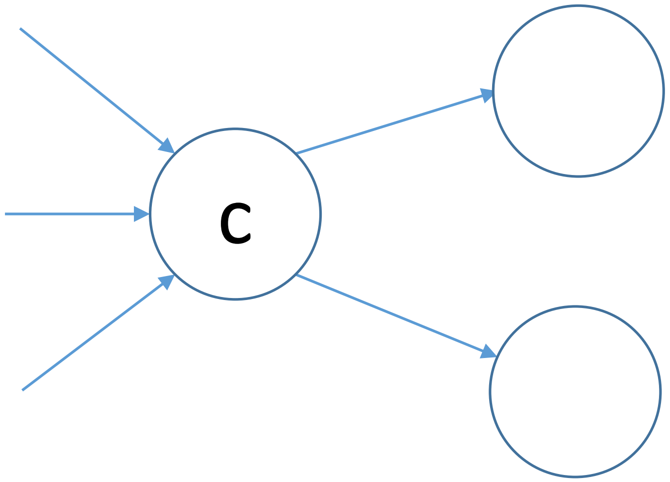


考察

- 各文字 (a – z) に対してしりとりの最後になりうるかを判定したい
- 入次数と出次数に着目？

(入次数) ≥ (出次数)のとき

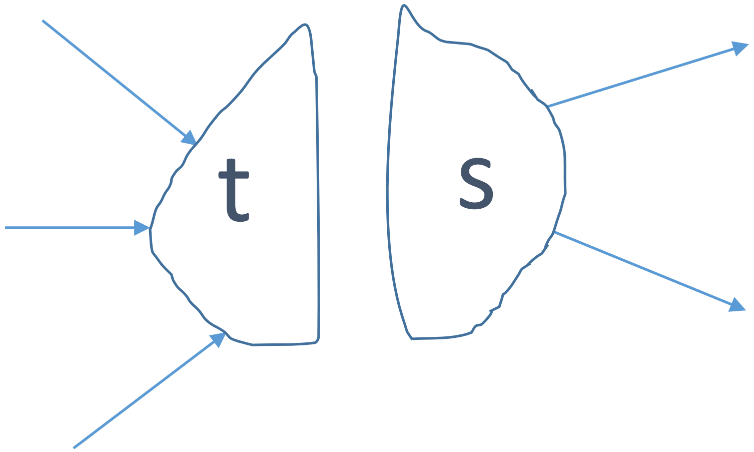
OK?　戻ってこれなかったらダメ



考察

- 各文字 (a – z) に対してしりとりの最後になりうるかを判定したい
  下の図のように頂点を分解して
  s から t への辺素なパスが出次数と同じになればOK!

辺素なパスの本数は容量 1 のグラフにおけるフロー値!



実装例(C++)

```
int main(){

    int n; cin >> n;
    vector<string> str(n);
    for(int i = 0; i < n; i++) cin >> str[i];

    //最後の文字となりうるか
    vector<bool> last(26, false);
    for(int i = 0; i < n; i++) last[str[i].back() - 'a'] = true;

    for(char c = 'a'; c <= 'z'; c++) if(last[c - 'a']) {

        long long s = 26, t = 27;
        long long cnt = 0;
        Dinic G(28);


        for(int i = 0; i < n; i++){

            //u -> v に有向辺を張る
            long long u = str[i].front() == c ? s : (long long)(str[i].front() - 'a');
            long long v = str[i].back() == c ? t : (long long)(str[i].back() - 'a');
            G.add_edge(u, v, 1);
            if(str[i].front() == c) cnt++;
        }

        //辺素なパスの本数が 出次数と等しかったら出力
        if(G.max_flow(s, t) == cnt) cout << c << endl;
    }

    return 0;
}
```

問題(ICPC 国内予選 2009 E)
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1163&lang=jp

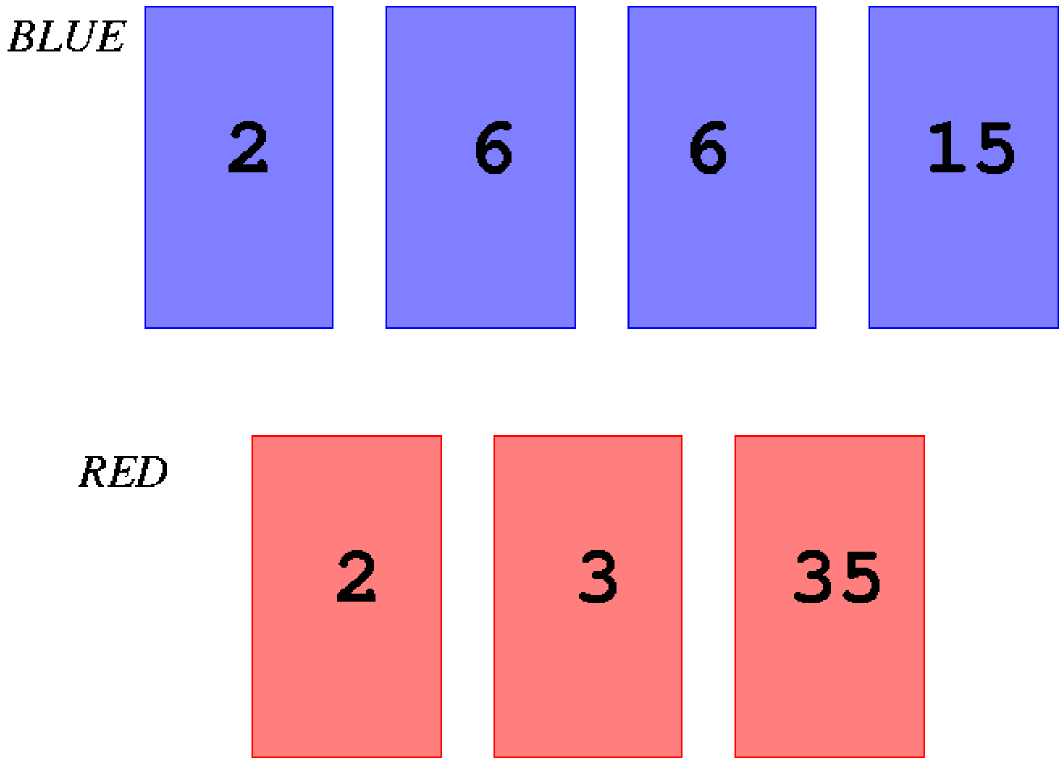
[問題概要]

青いカードが m 枚、赤いカードが n 枚あり、それぞれに$1 \leq x \leq 10^7$の整数が書かれている。赤のカードと青のカードから互いに素でないペアを選んで取り除く操作を行う。適切に操作を行った時、最大何組のペアが取り除かれるか？

[制約]

$1 \leq m, n \leq 500$

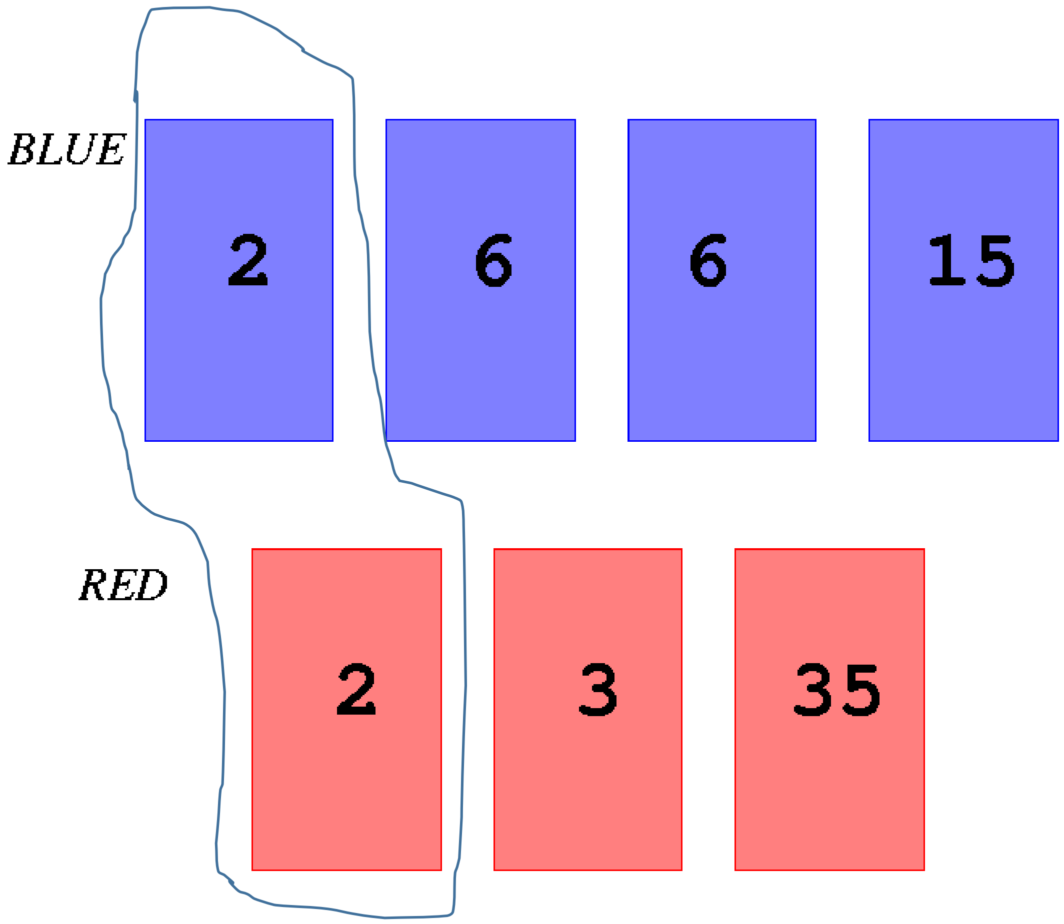
考察



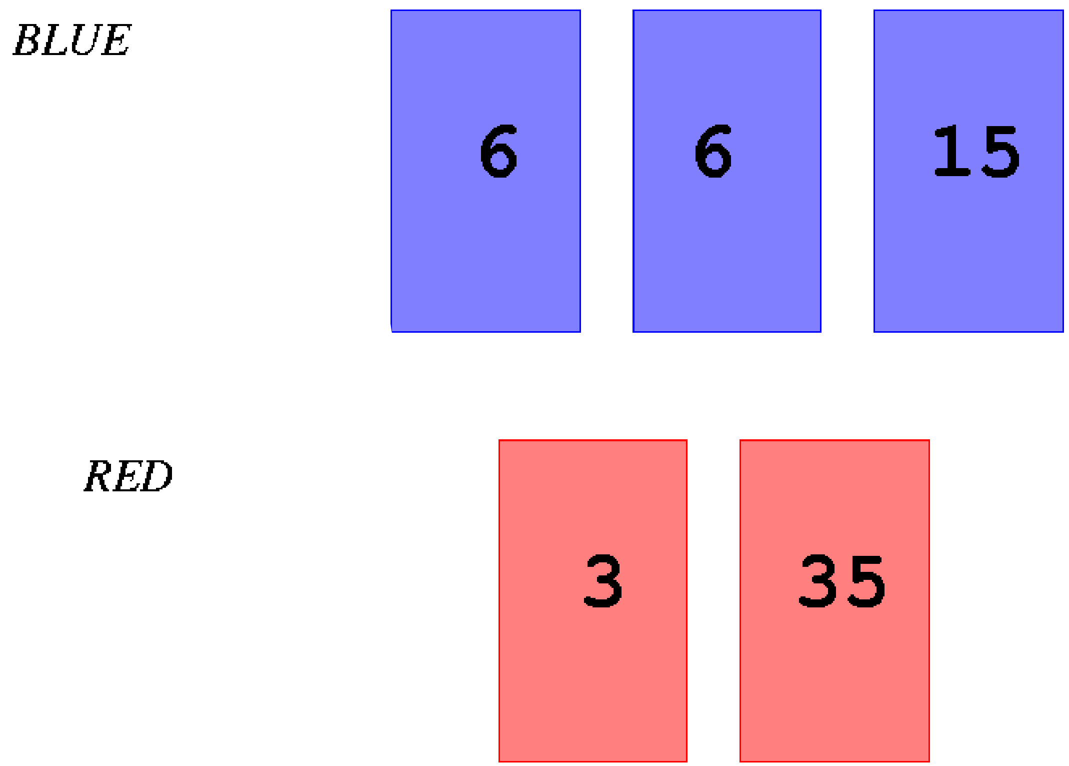
引用　http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1163&lang=jp

考察

- 最初に青の 2 と赤の 2 を取り除く

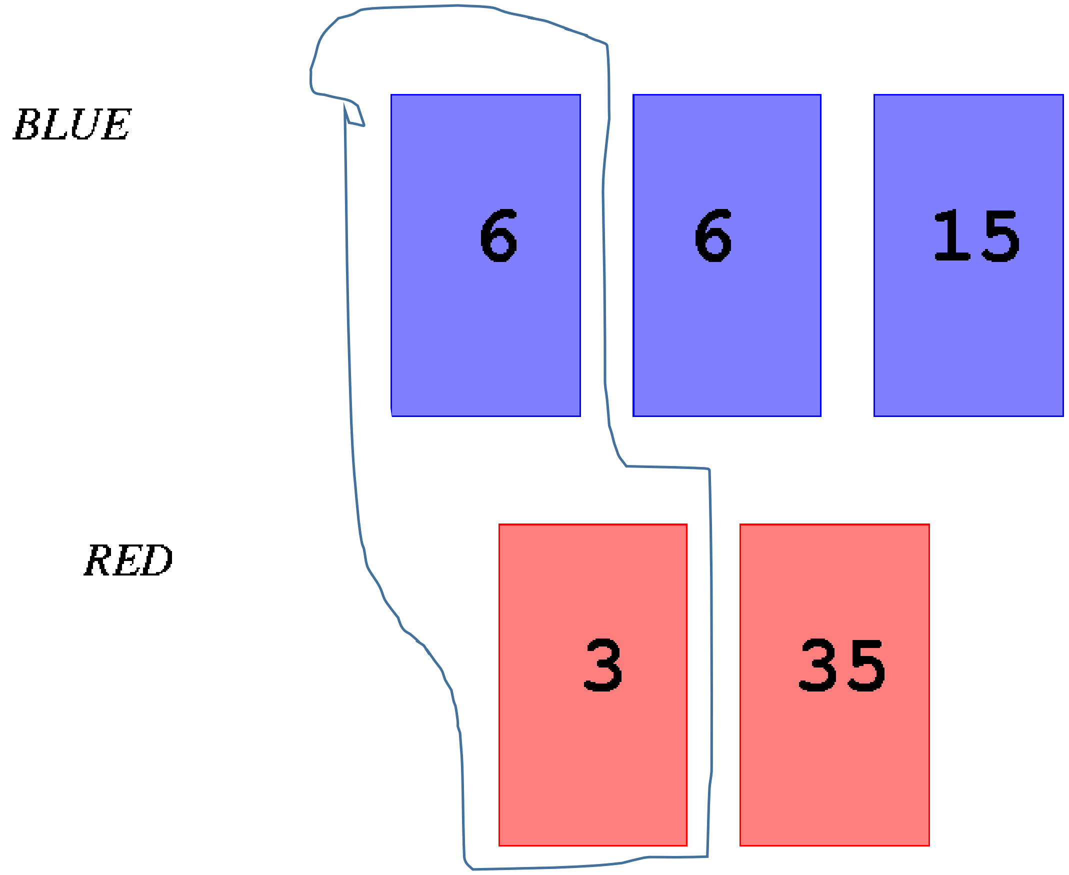


考察

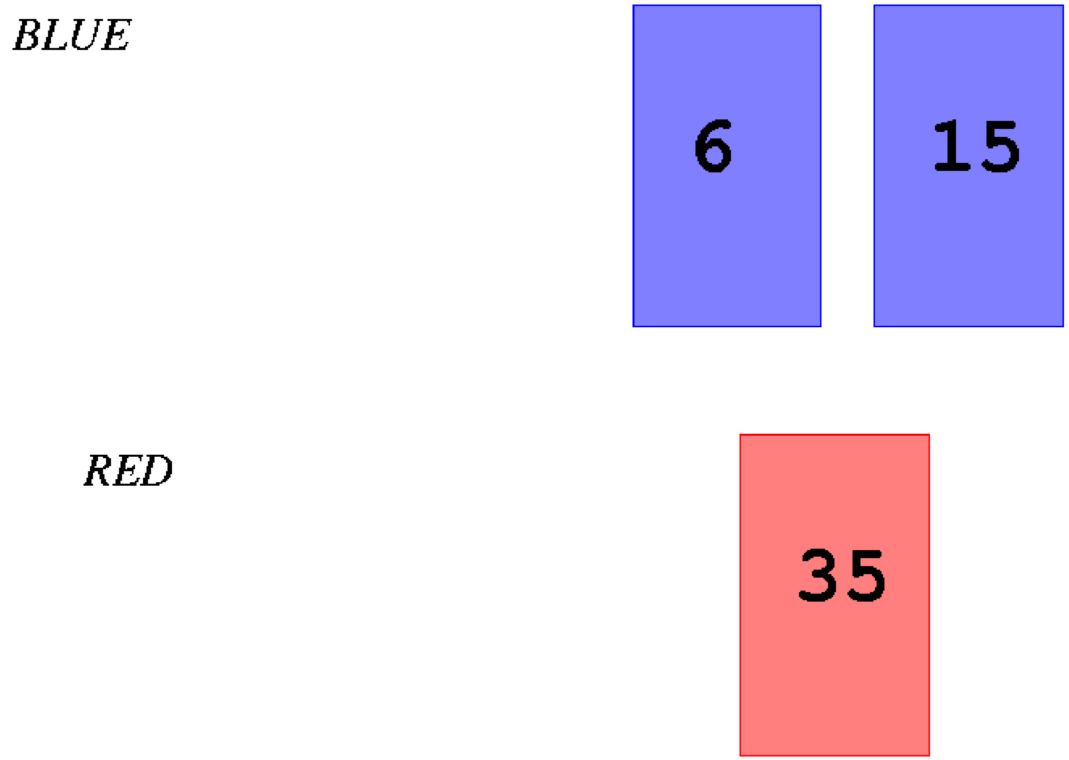


考察

- 次に青の 6 と赤の 3 を取り除く

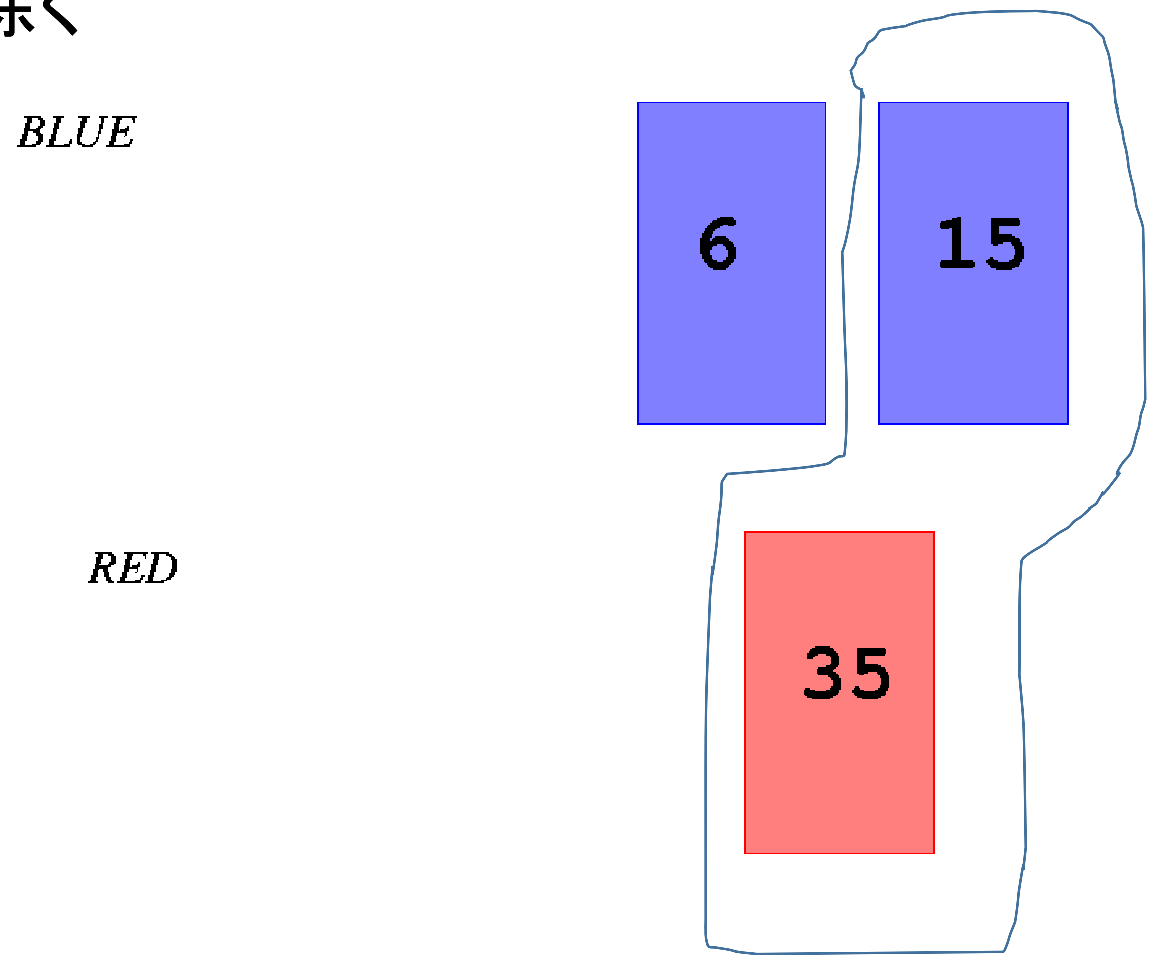


考察



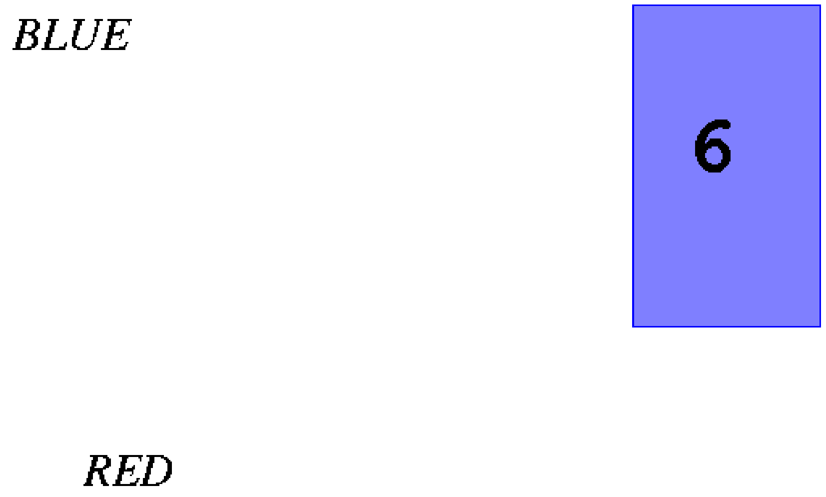
考察

- 最後に青の 15 と 赤の 35 を取り除く



考察

- 答えは 3 !!



貪欲法？

取り除けるペアを見つけて、貪欲に取り除く

貪欲法？

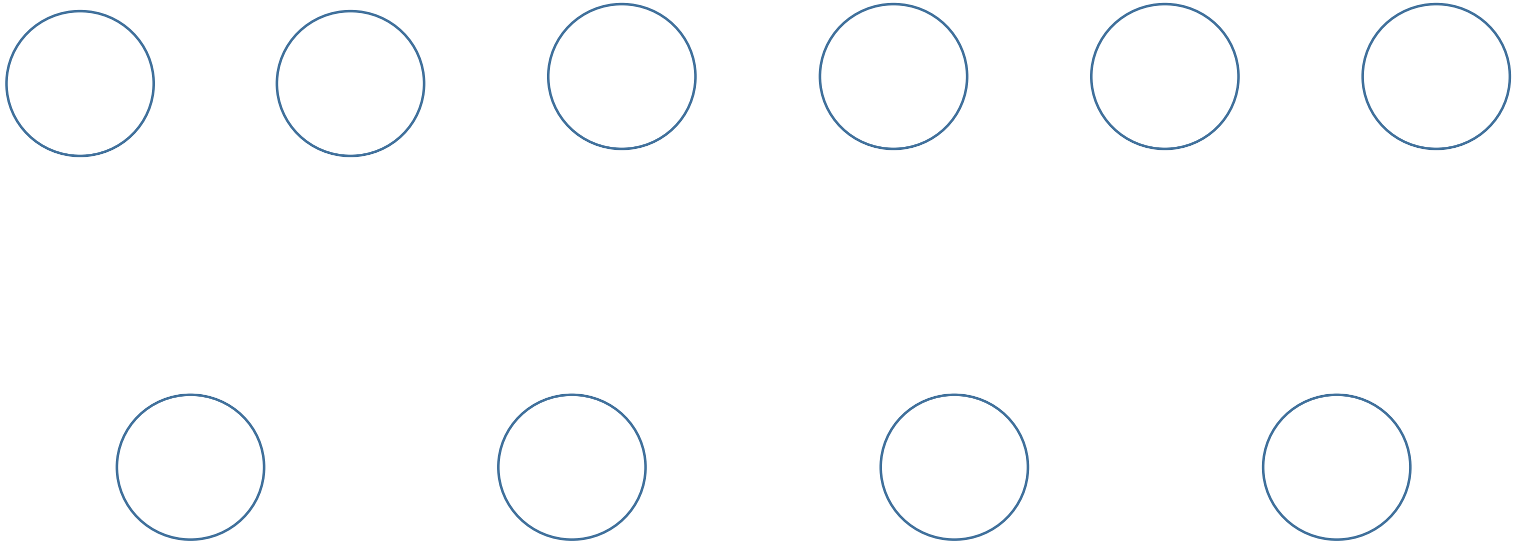
例えば以下のようなとき、貪欲法ではだめなことがわかる。

Blue 2 3

Red 6 4

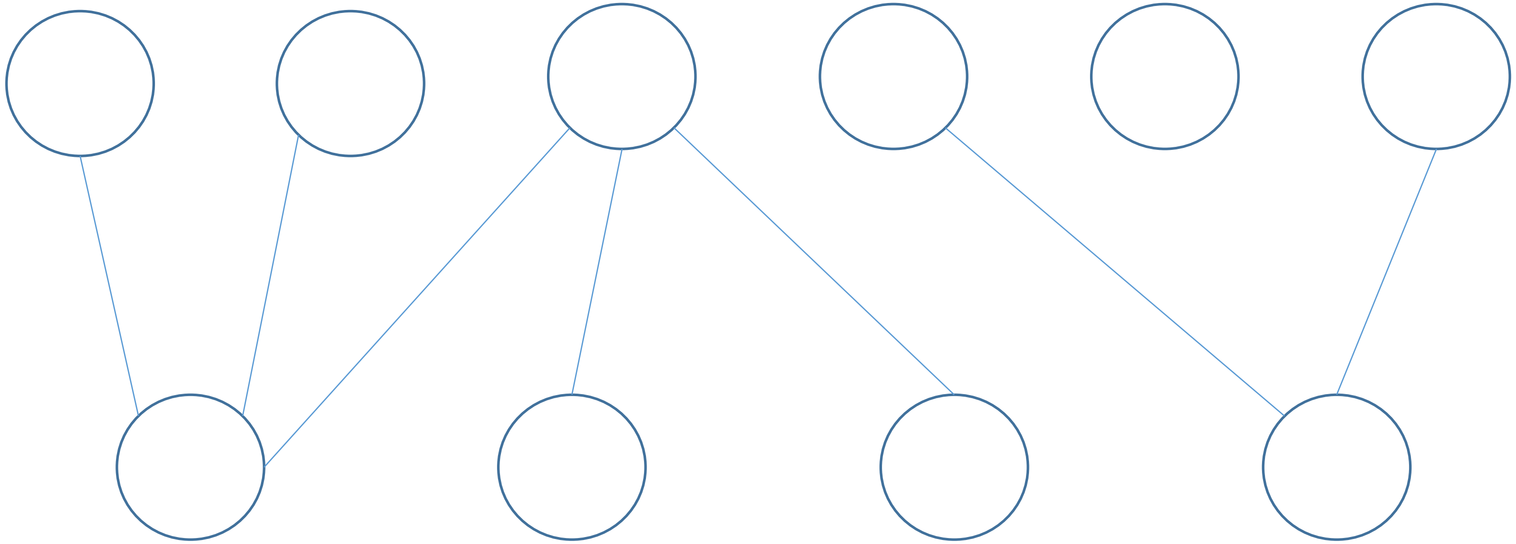
考察

- グラフに落とし込んで考察



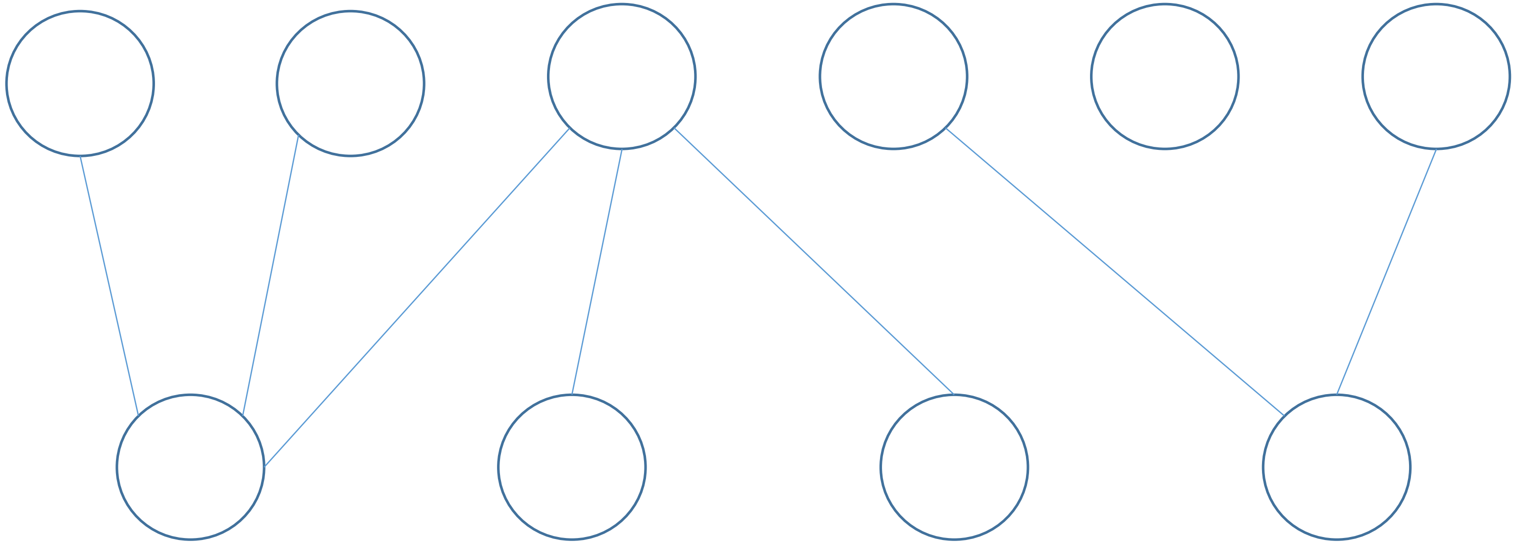
考察

- 前処理で互いに素な頂点間に辺を張る



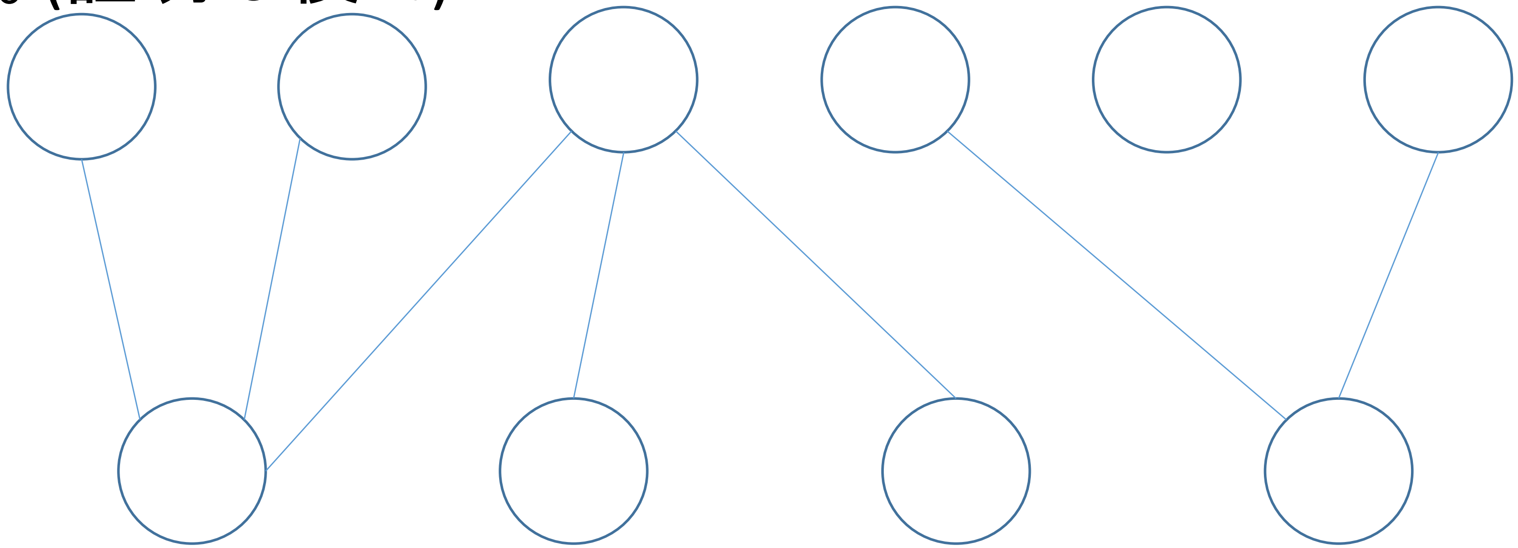
考察

- 作れるペアの最大値を求める問題 = 最大マッチング！！



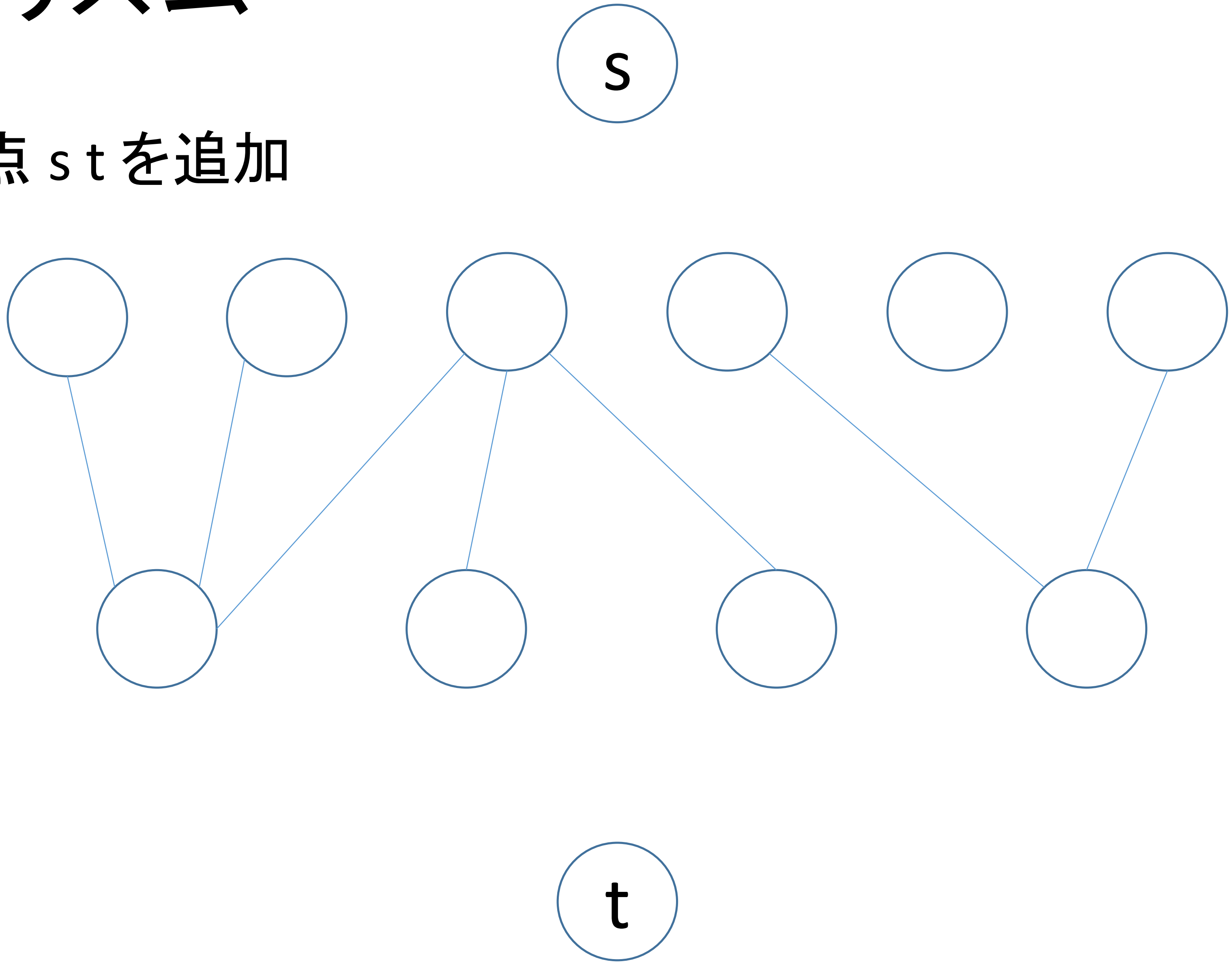
考察

- 二部グラフの最大マッチングはフローを流すことで解けることが知られている。(証明は後で)



アルゴリズム

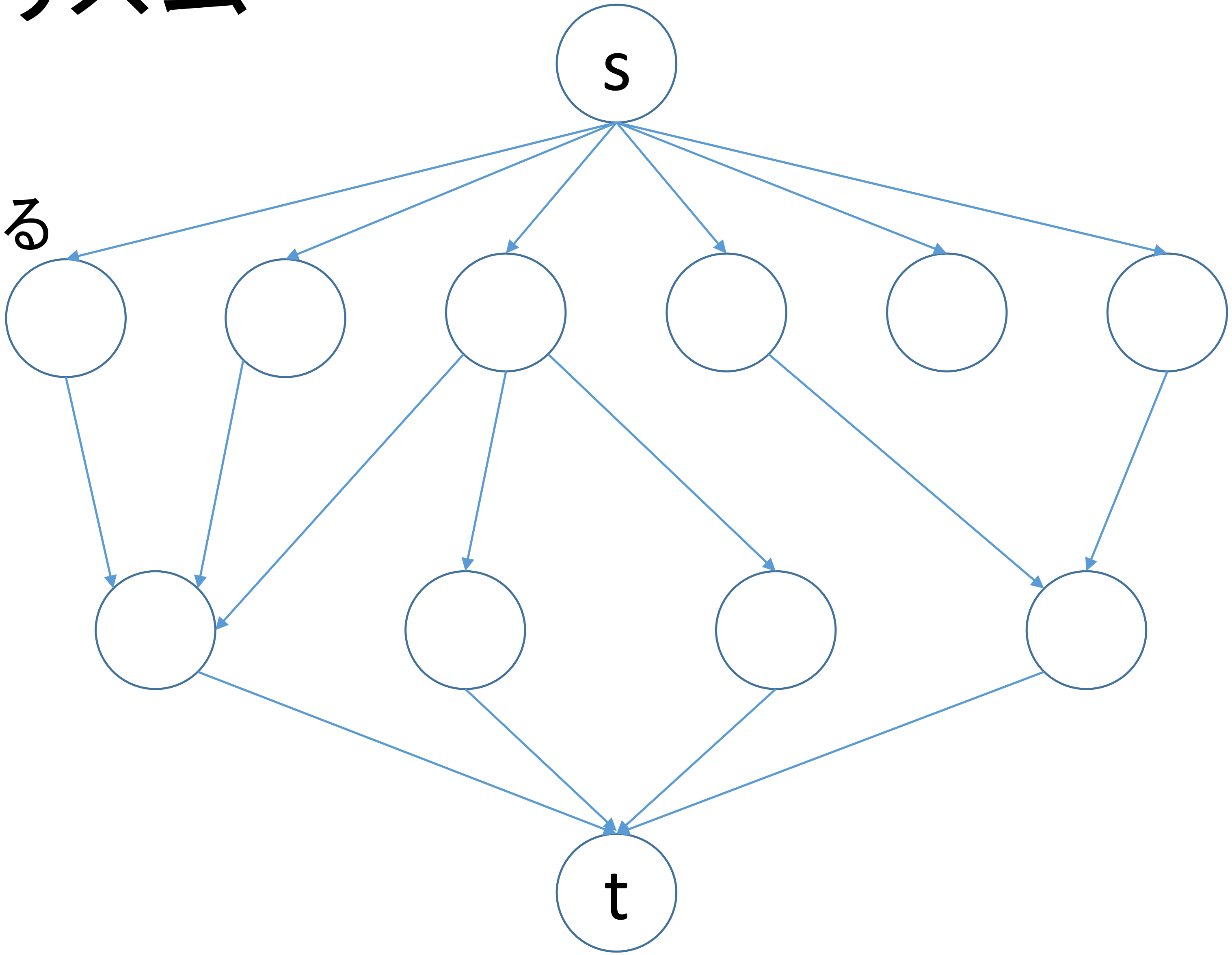
架空の頂点ｓｔを追加



アルゴリズム

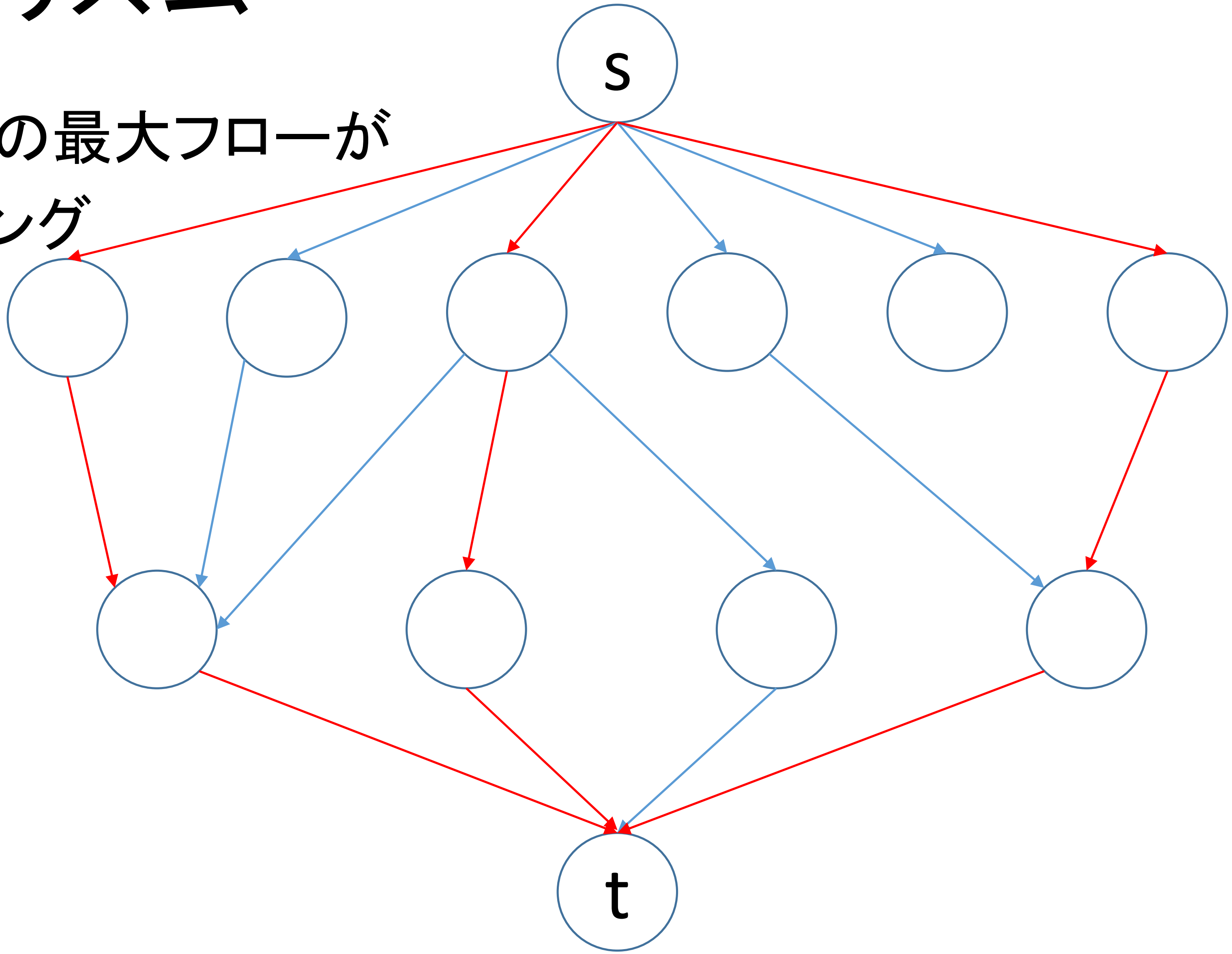
図のように
有向辺を張る

辺の
容量は1



アルゴリズム

ｓ から ｔへの最大フローが
最大マッチング
になる！



証明

- 復習 (最大マッチング)

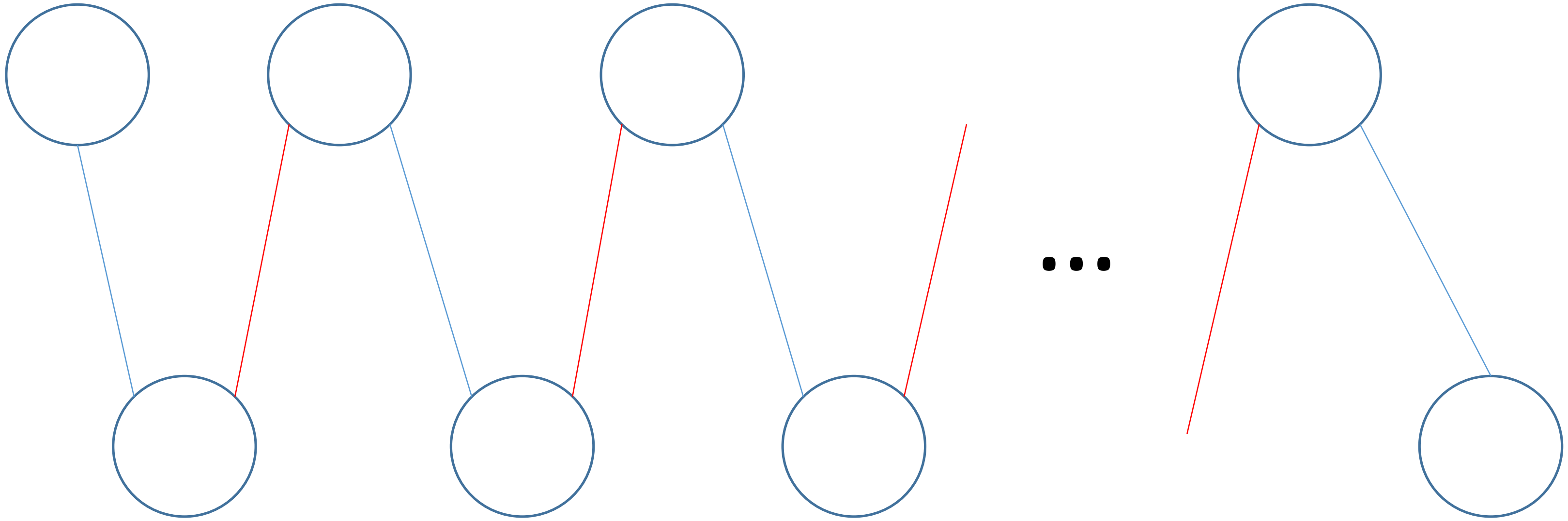
マッチングが最大である ⇔ 増加パスを持たない

フローによって得られたマッチングが増加パスを持たないことを示せばよい

増加パスを持つことを仮定して矛盾を導く(背理法)

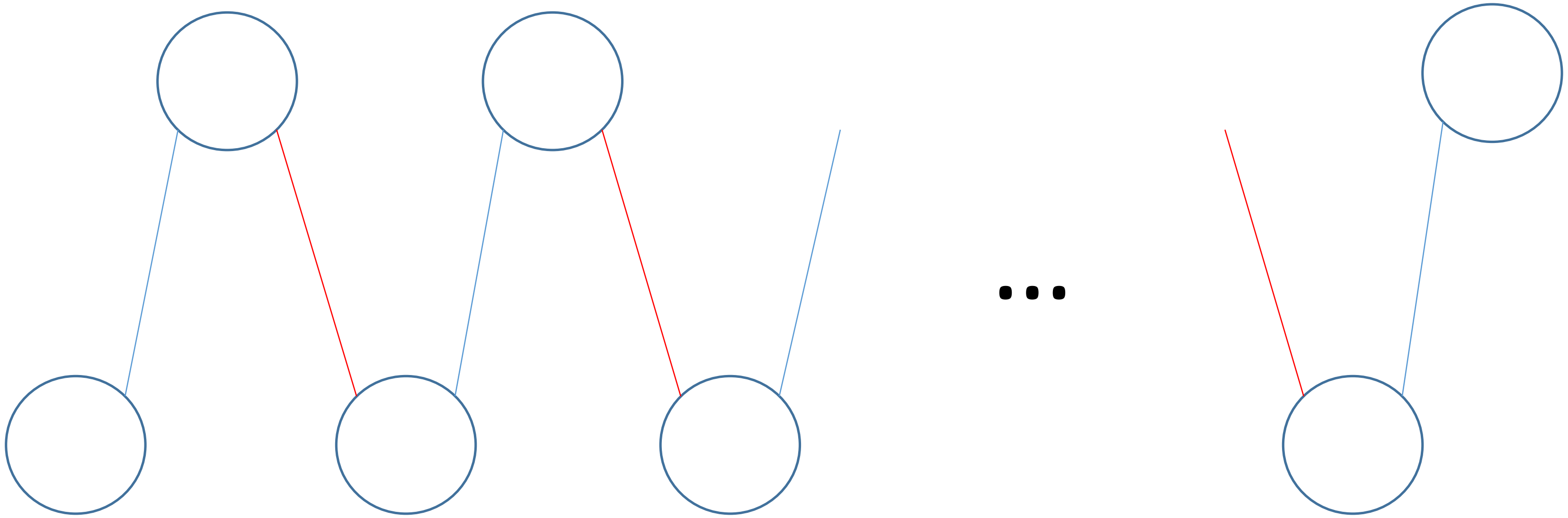
証明

増加パスを持つと仮定する。
以下の二つの部分構造のどちらかを持つ。



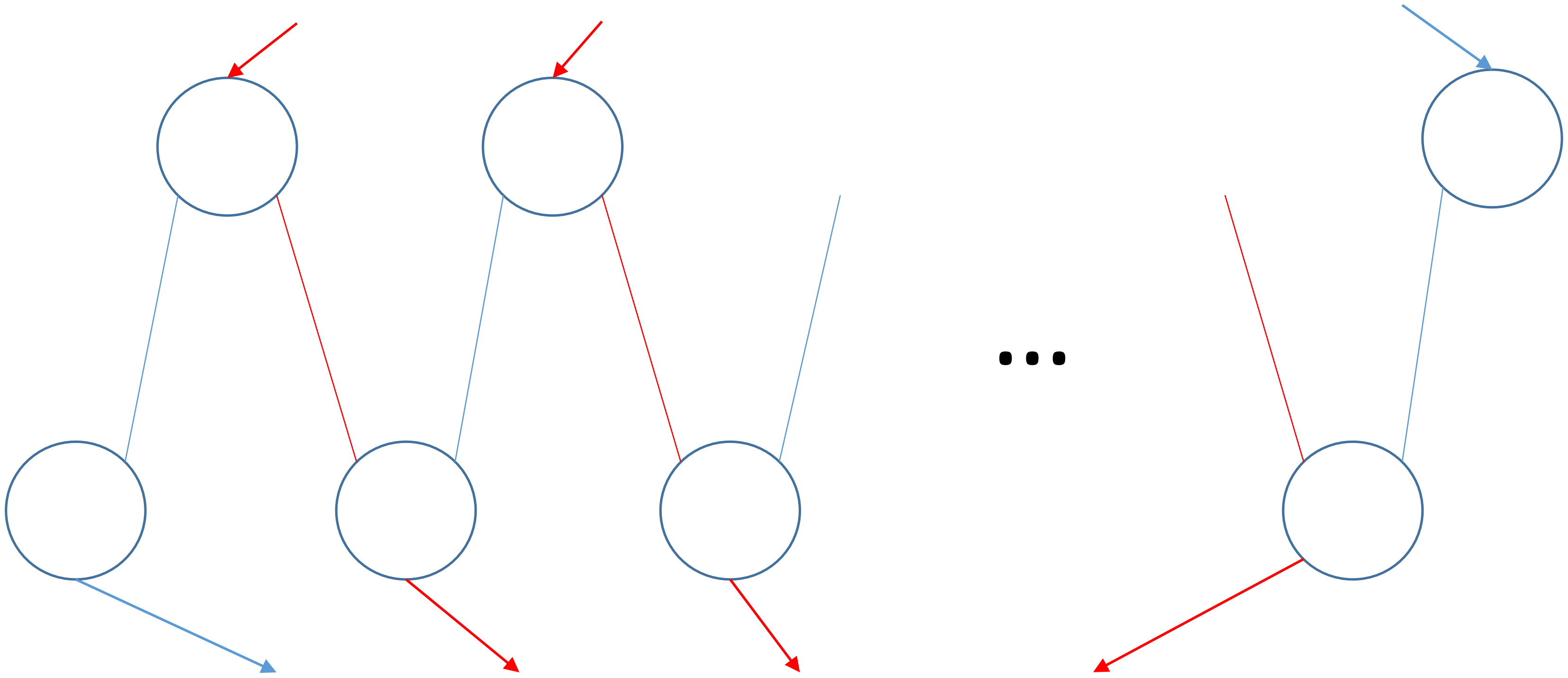
証明

増加パスを持つと仮定する。
以下の二つの部分構造のどちらかを持つ。



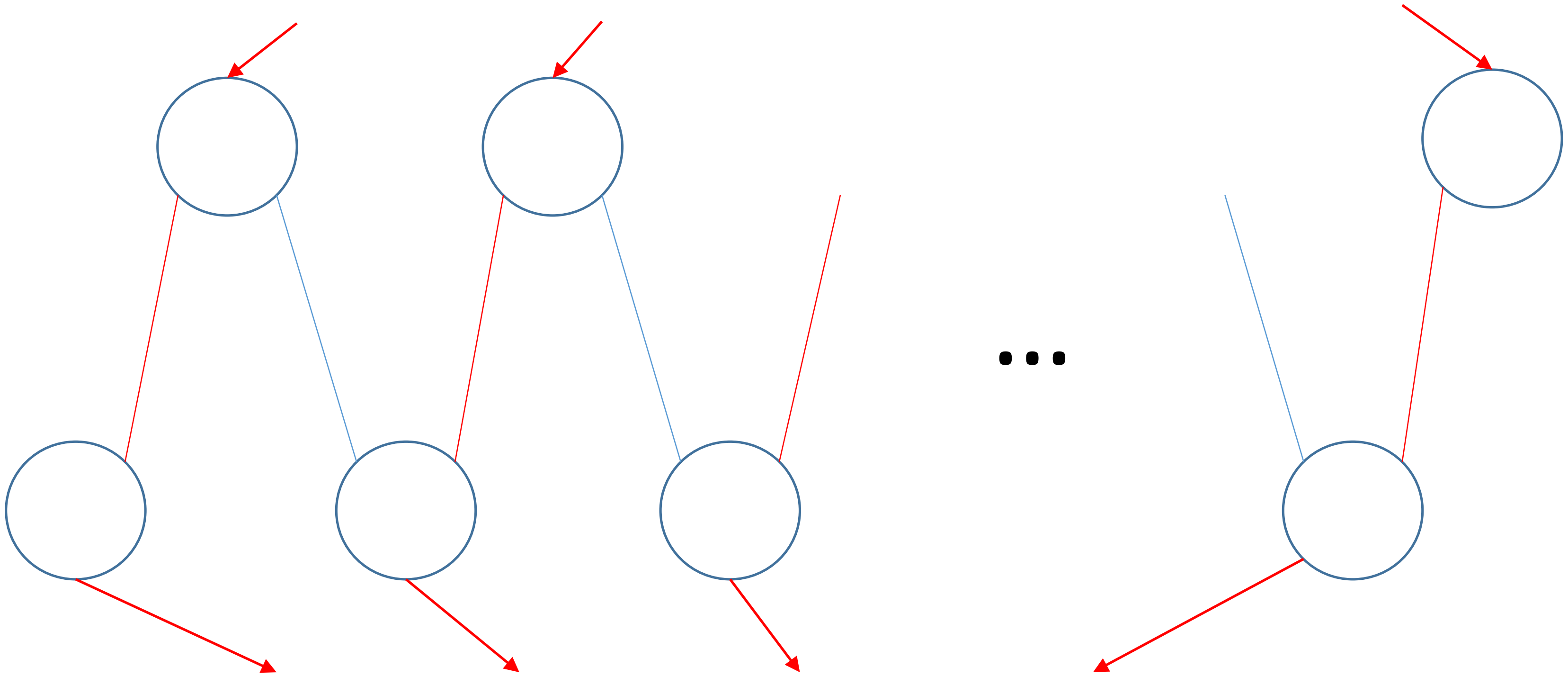
証明

- このマッチングを得るフローを考える



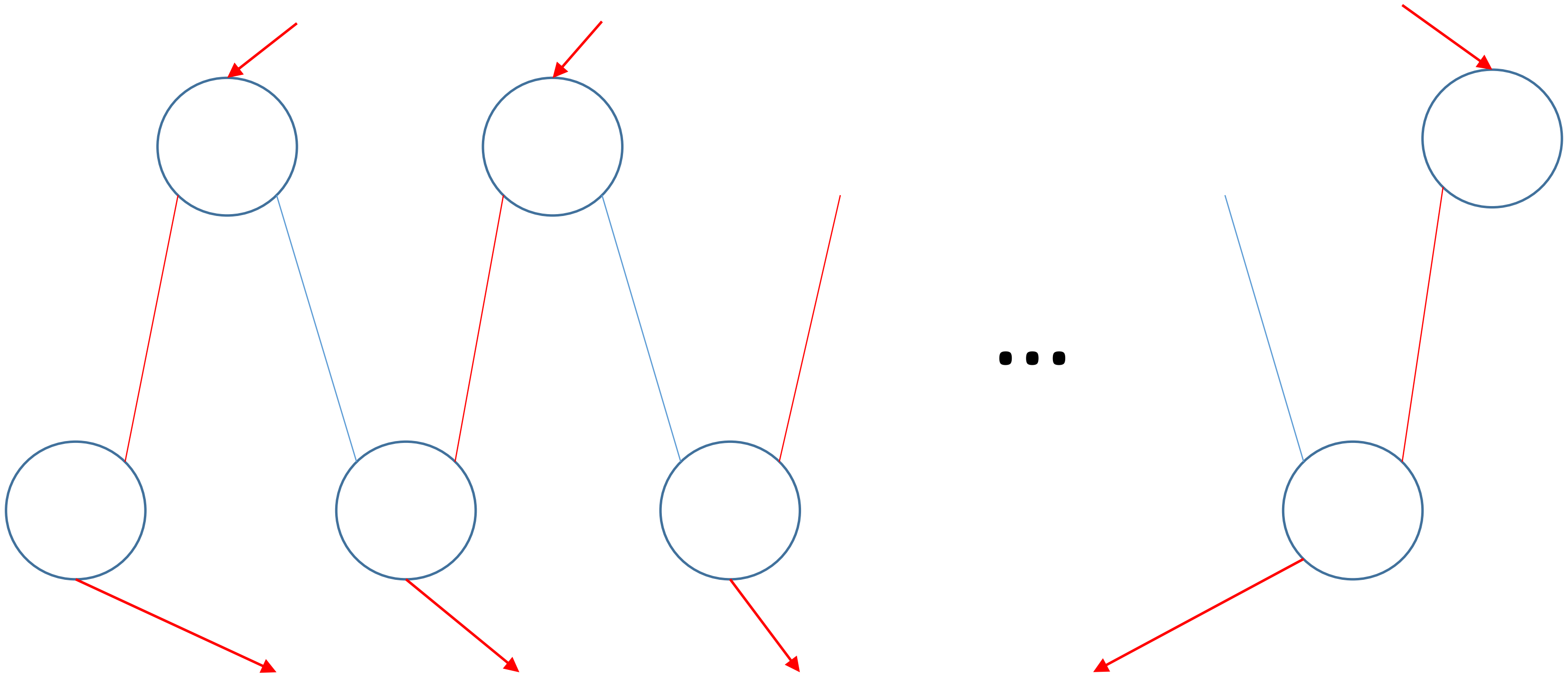
証明

- このマッチングを得るフローを考える
- 増加パス上の辺を反転させるようなフローを構成でき、フロー値を大きくできる。



証明

- 最大フローであることに矛盾！



ソースコード (C++)

```
int main(){

    while(1){

        int m, n; cin >> m >> n;
        if(!n) break;
        vector<int> b(m);
        vector<int> r(n);
        for(int i = 0; i < m; i++) cin >> b[i];
        for(int i = 0; i < n; i++) cin >> r[i];

        //0 := start, [1, m] := blue, [m + 1, m + n] := red, m + n + 1 := goal
        Dinic G(n + m + 2);

        for(int i = 1; i <= m; i++) G.add_edge(0, i, 1);
        for(int i = m + 1; i <= m + n; i++) G.add_edge(i, m + n + 1, 1);

        //辺を張る
        for(int i = 0; i < m; i++){
            for(int j = 0; j < n; j++){
                if(gcd(b[i], r[j]) > 1){
                    G.add_edge(i + 1, m + j + 1, 1);

                }
            }
        }

        cout << G.max_flow(0, m + n + 1) << endl;
    }

    return 0;
}
```

練習問題