「データ構造とアルゴリズム」第16章
ネットワークフロー

JOI 春季セミナー2024 上級

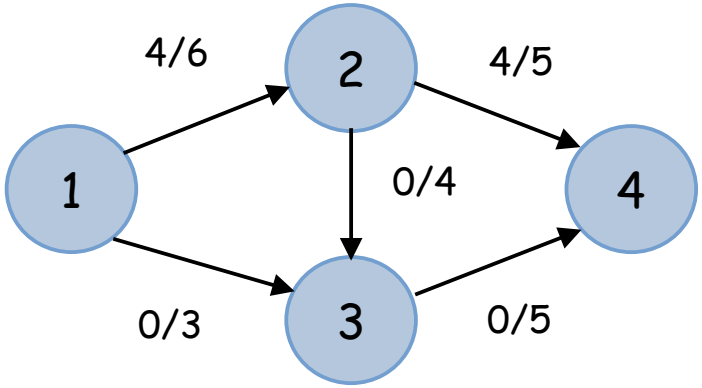
目次

1. ネットワークフローとは？
2. 最大フローを求めるアルゴリズムの紹介（フォード・ファルカーソン法）
3. ネットワークフローの応用例
4. ネットワークフローを用いた問題の例
5. 演習問題など

1. ネットワークフローとは？

- グラフ上における**流れ**を扱う分野
  - グラフに「もの」を流すことを考える
  - グラフの各辺に対して**容量**（「もの」を流せる上限）を導入する
  - また、ある辺について流れている「もの」の量を**流量**という

辺の値は、
流量 / 容量
を表している



例えば、
頂点 1 から 2 へは
4 だけ流れている

1. ネットワークフローとは？

- 今回の目標
  - ネットワークフローに関する簡単な問題が解けるようになる！

    → 今回は特に最大フローについて扱います

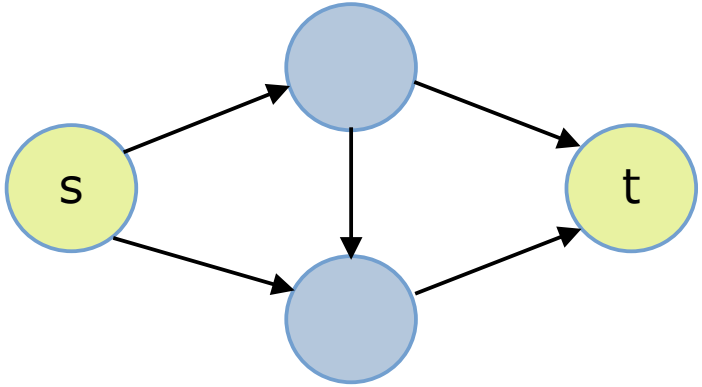
前提知識の説明が多いため、しばらくは用語についての説明になります

1. ネットワークフローとは？

- **辺連結度**について

  s-t 辺連結度とは、グラフのある頂点 s から t を**非連結**にするために必要な取り除く辺の個数の**最小値**

  以下のグラフにおける s-t 辺連結度を求めてみよう

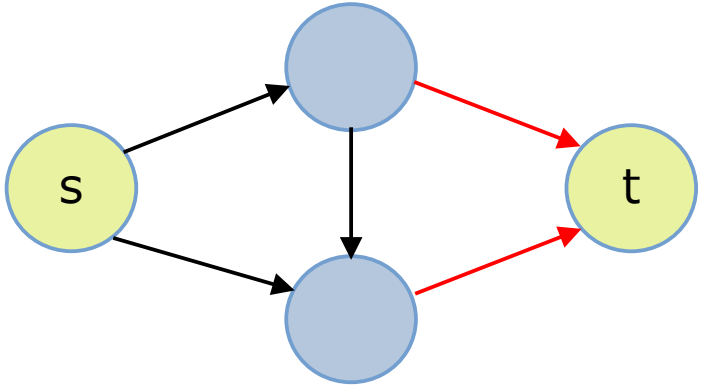


1. ネットワークフローとは？

- **辺連結度**について

  例えば、赤く塗った辺を取り除くことによって s から t について非連結にすることができる

  1 本以下の辺を取り除くことによって非連結にすることはできないので **s-t 辺連結度**は 2



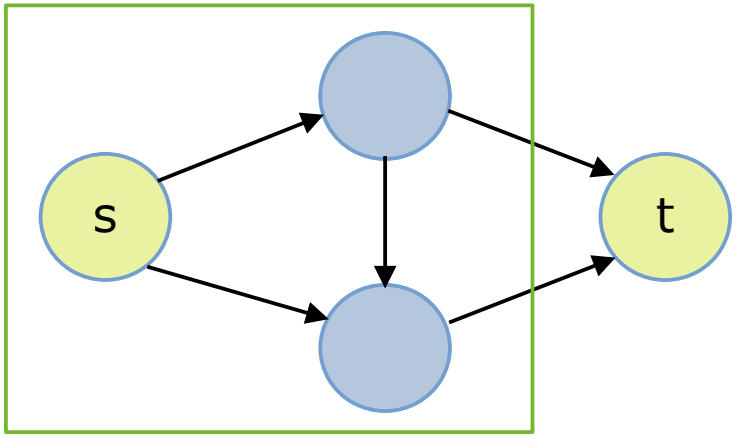
1. ネットワークフローとは？

- **カット**について

  s-t カットとは、頂点集合の分割 S であって、s ∈ S であって、t ∉ S を満たすもの

  → 頂点 s と t でグラフを 2 分割するイメージ

s-t カットの例
(緑の枠線で囲まれた部分)

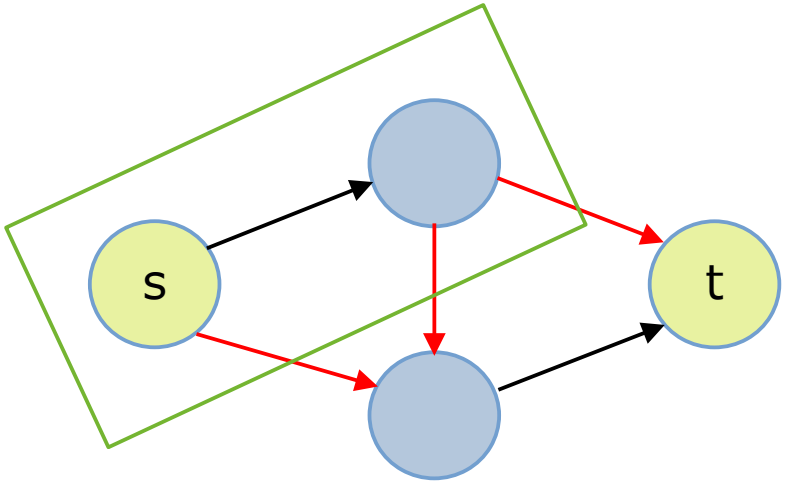


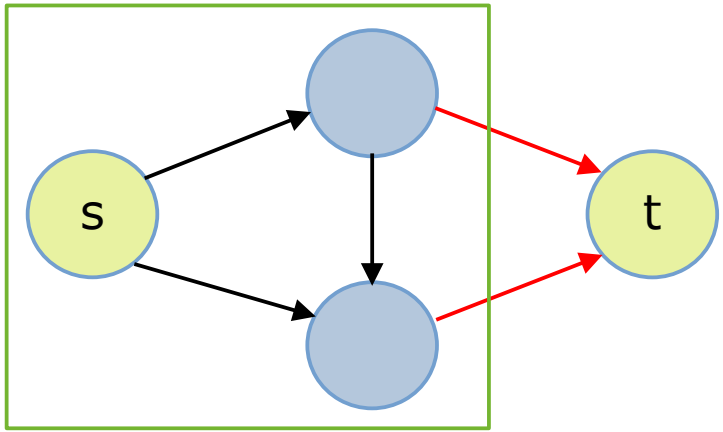
1. ネットワークフローとは？

- **カットセット**について

  頂点集合 V の分割 S に関する**カットセット**とは、
  S から V ＼ S (S に含まれていない頂点集合）**への辺**の集合のこと

緑の枠線を S としたときの
カットセットの例
（赤線の部分）



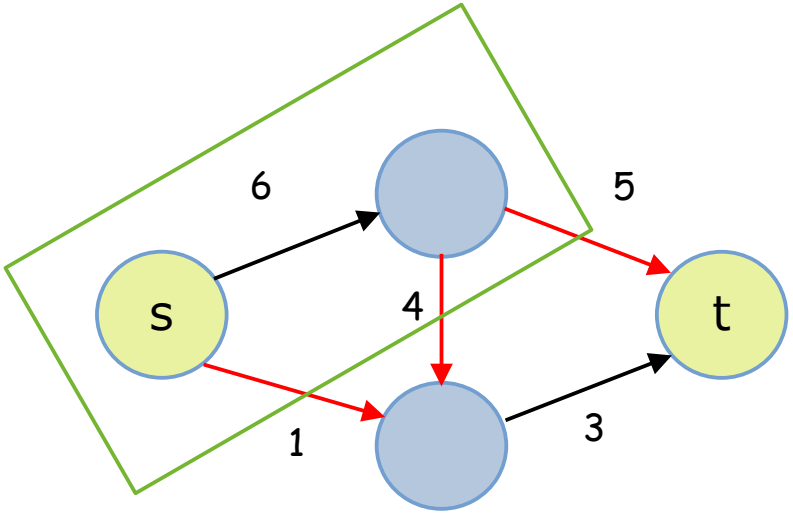


1. ネットワークフローとは？

- カットの**容量**について

  s-t カットにおける**容量**とは、s-t カットに関するカットセットの**重みの総和**のこと

  例えば、以下の s-t カットにおける**容量**は 1 + 4 + 5 = 10

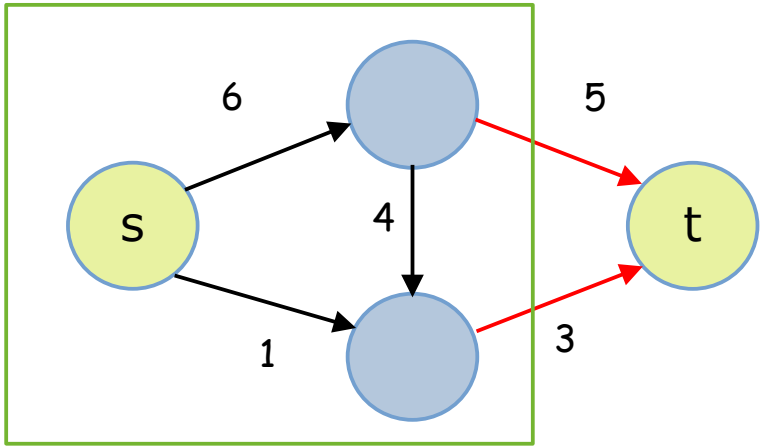


1. ネットワークフローとは？

- カットの**容量**について

  s-t カットにおける**容量**とは、s-t カットに関するカットセットの**重みの総和**のこと

  例えば、以下の s-t カットにおける**容量**は 5 + 3 = 8



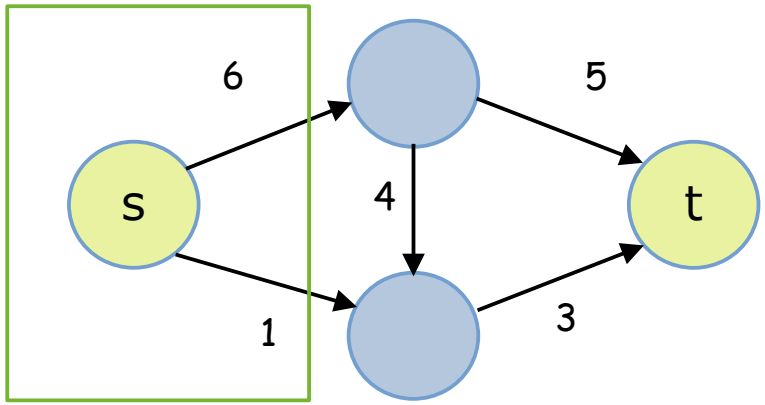
1. ネットワークフローとは？

- **最小カット**について

  最小 s-t カットとは、s-t カットのうち、容量が**最小**のもの

  例えば、先ほどのグラフにおける**最小 s-t カット**は以下のようになる。

  （カットの容量は 6 + 1 = 7 でこれより容量が小さい s-t カットはない）

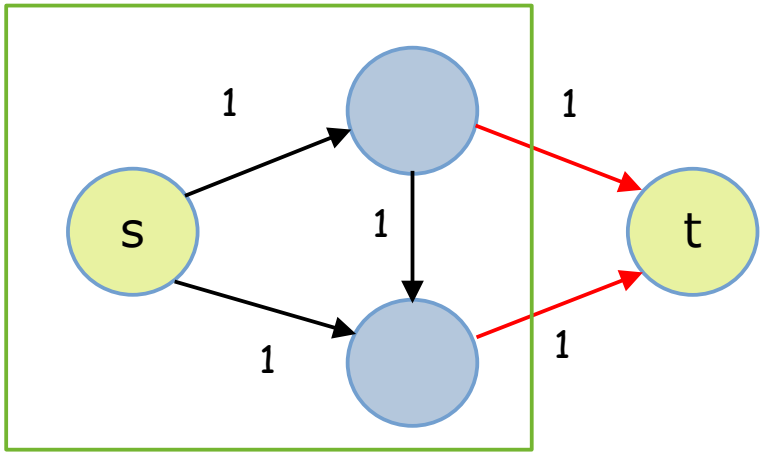


1. ネットワークフローとは？

- **最小カット**について（辺連結度との関係性）

  s-t 辺連結度は、**辺の重みが全て 1 であるグラフ**の最小 s-t カットの容量と一致する（証明は省略）

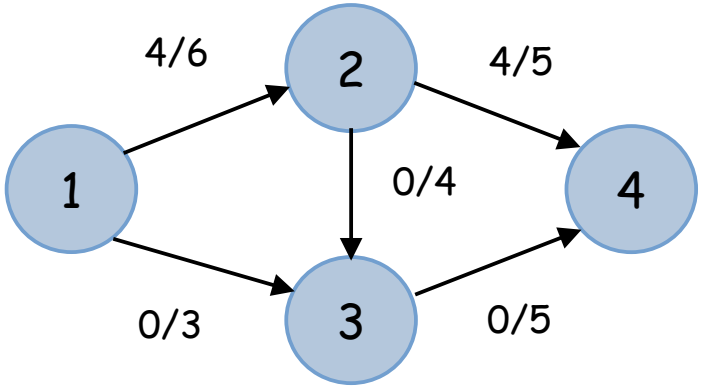
  つまり、最小カットが求められれば辺連結度を求められることになる



1. ネットワークフローとは？

- ここからはフローについての説明
  - グラフに「もの」を流すことを考える
  - グラフの各辺に対して**容量**（「もの」を流せる上限）を導入する
  - また、ある辺について流れている「もの」の量を**流量**という

辺の値は、
流量 / 容量
を表している



例えば、
頂点 1 から 2 へは
4 だけ流れている

1. ネットワークフローとは？

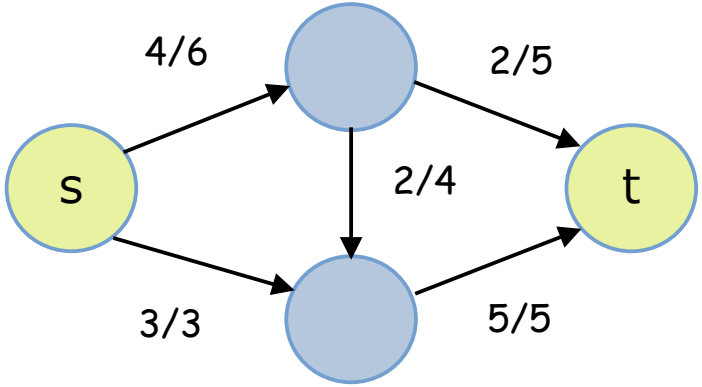
- **フロー**について

  グラフのある頂点からある頂点に対して、「もの」を流す方法

  また、始点から流れている「もの」の量の合計を**総流量**という

  以下は頂点 s から t に対する**フロー（s-t フロー）**の例であり、**総流量**は 7 である。

辺の値は、
流量 / 容量
を表している



1. ネットワークフローとは？

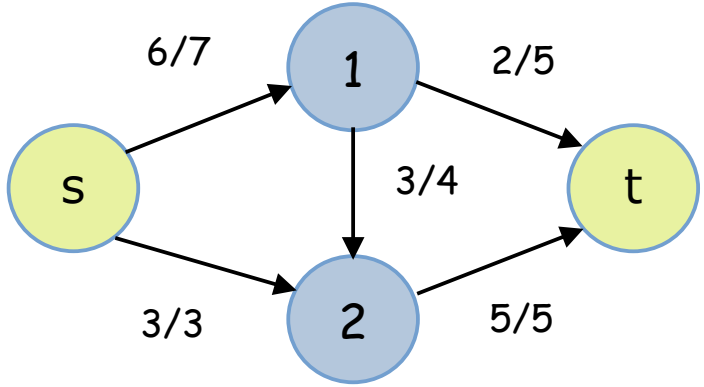
- **フロー（許容フロー）**における注意

  頂点 s から t に対する**フロー（許容フロー）**は以下の条件を満たしていないといけない

  1. 各辺について、流量は **0 以上 容量 以下**である。
  2. ある頂点について、その頂点に流れてくる「もの」の量とその頂点から流れる「もの」の量が**等しい**

     （頂点 s, t を除く）

右の例は
頂点 1 において 2 番目の
条件を満たしていない



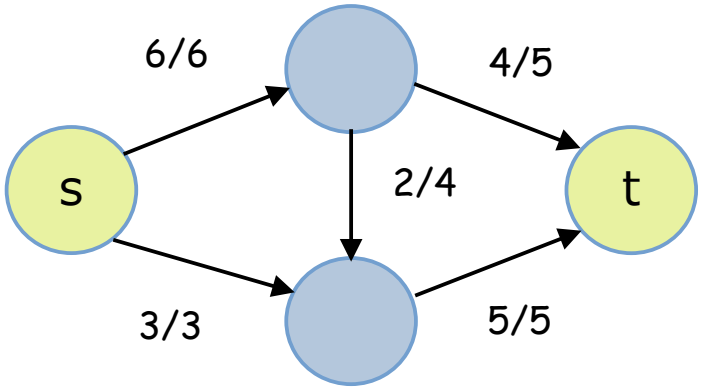
1. ネットワークフローとは？

- **最大フロー（最大流）**について

  グラフのある頂点からある頂点のフローについて、流せる「もの」の量が**最大**のもの

  例えば、頂点 s から t に対する**最大フロー（最大流）**は以下のようになり、総流量は 9 である。

辺の値は、
流量 / 容量
を表している



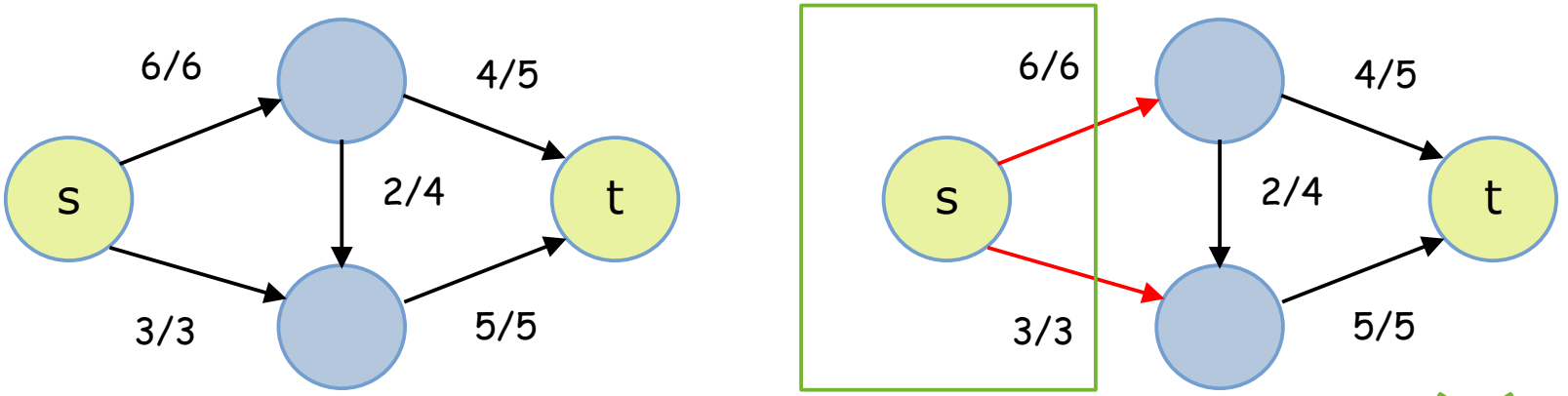
1. ネットワークフローとは？

- **最大フロー・最小カット定理** について

  辺の容量を重みとすれば、以下が成り立つ（正当性の証明は後ほど）

  最大 s-t フローの**総流量**は最小 s-t カットの**容量**と**等しい**

最小カットの容量は
6 + 3 = 9 で
最大フローの総流量と
一致



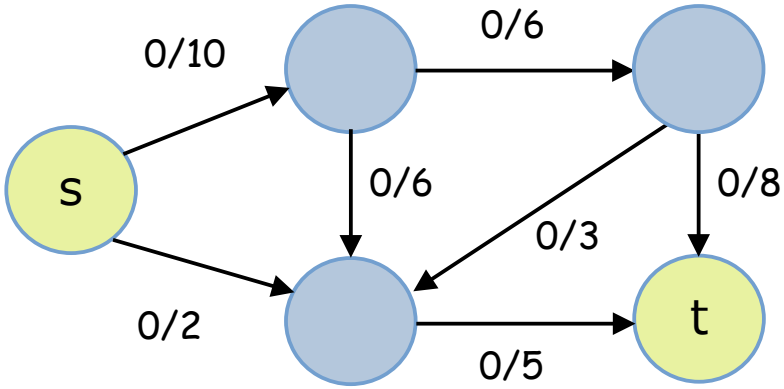
2. 最大フローを求めるアルゴリズムの紹介

- 最大フローを求めるアルゴリズムの気持ち

  基本的な考え方：貪欲法のように求められないか？

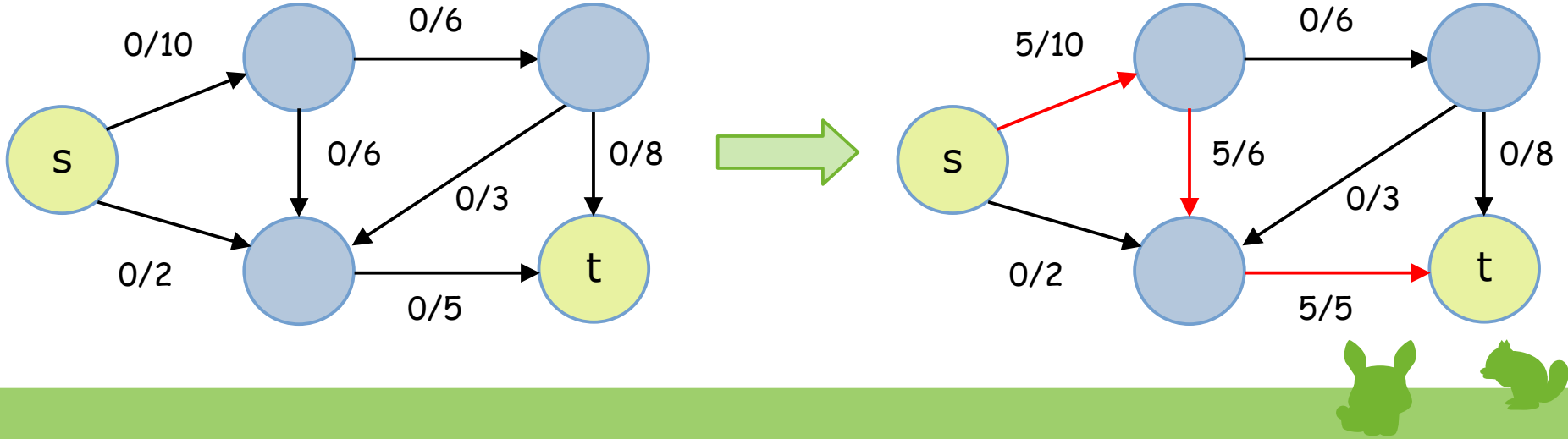
  → 流量が容量に達していない辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる

  - このアルゴリズムは**間違い**だが試しに以下のグラフでやってみる



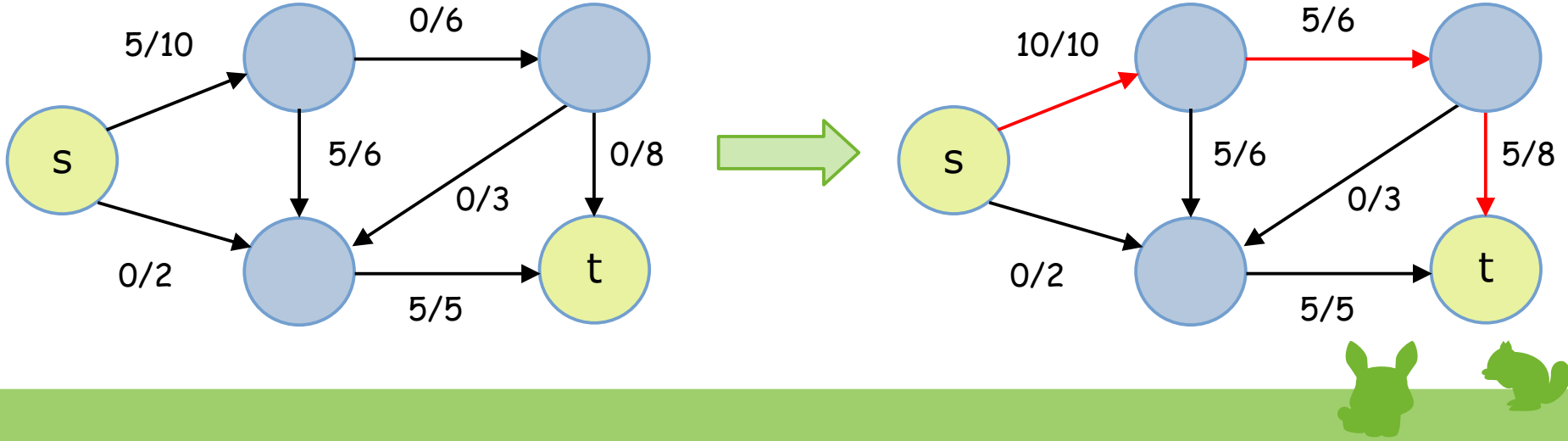
2. 最大フローを求めるアルゴリズムの紹介

- 流量が容量に達していない辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる
  - 赤線の部分が新たに流した辺となる



2. 最大フローを求めるアルゴリズムの紹介

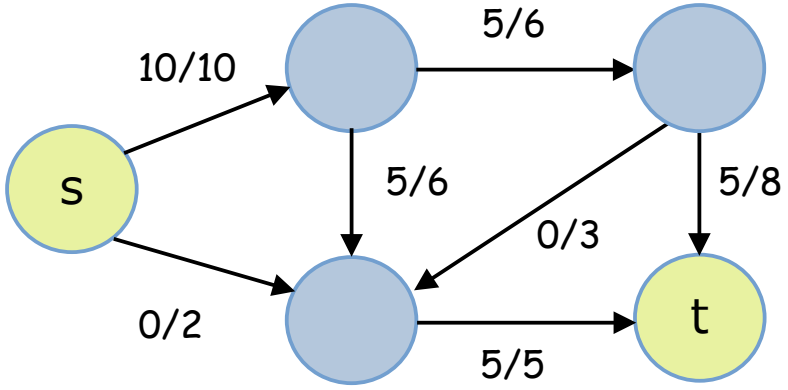
- 流量が容量に達していない辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる
  - 赤線の部分が新たに流した辺となる



2. 最大フローを求めるアルゴリズムの紹介

- 流量が容量に達していない辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる
  - これ以上は流せないので終了する

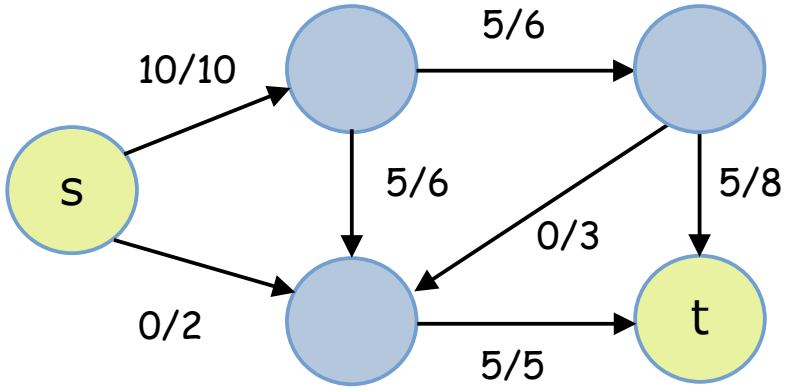
最大流量は 10？

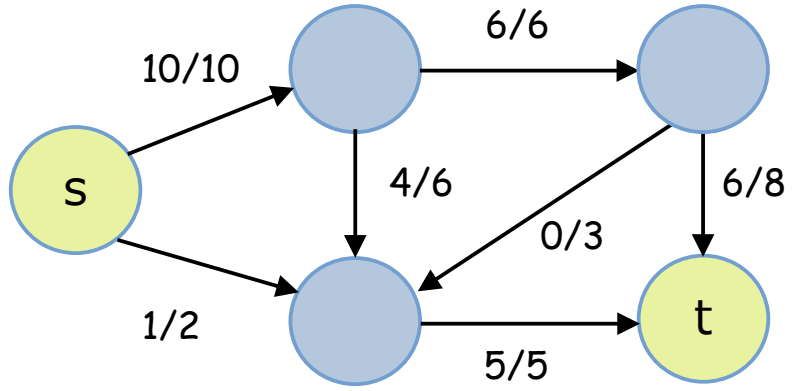


2. 最大フローを求めるアルゴリズムの紹介

- 流量が容量に達していない辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる
  - これ以上は流せないので終了する

最大流量は 10？ → 右図のようにすれば総流量 11 を達成できる

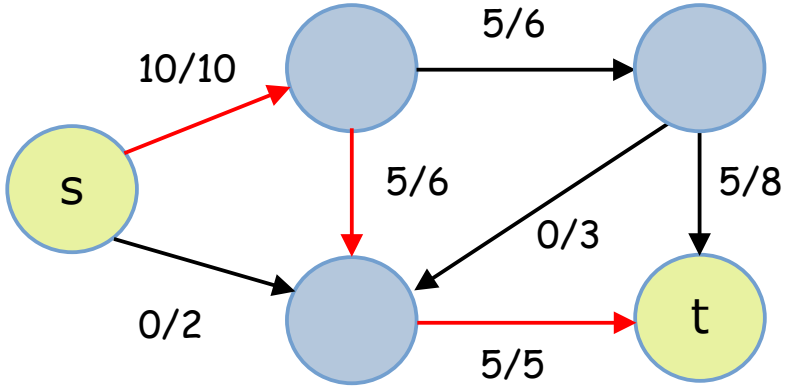


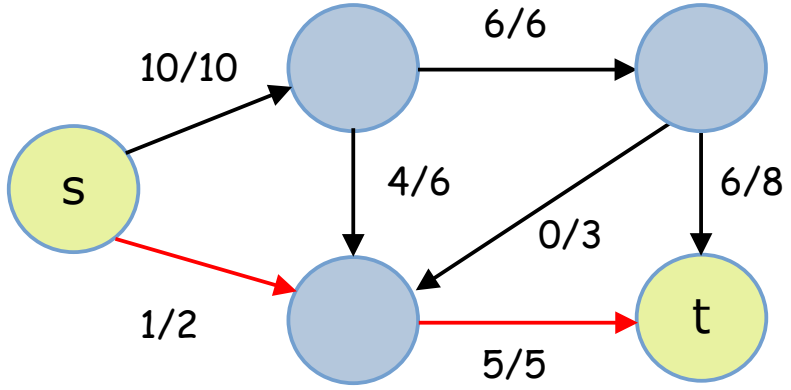


2. 最大フローを求めるアルゴリズムの紹介

- なぜこのアルゴリズムではダメなのか？
  - 他のパスで流した方が良い場合の経路に先に流してしまっている

    → さらに良いパスが見つかった際に今までに流したものをキャンセルできる機構が必要



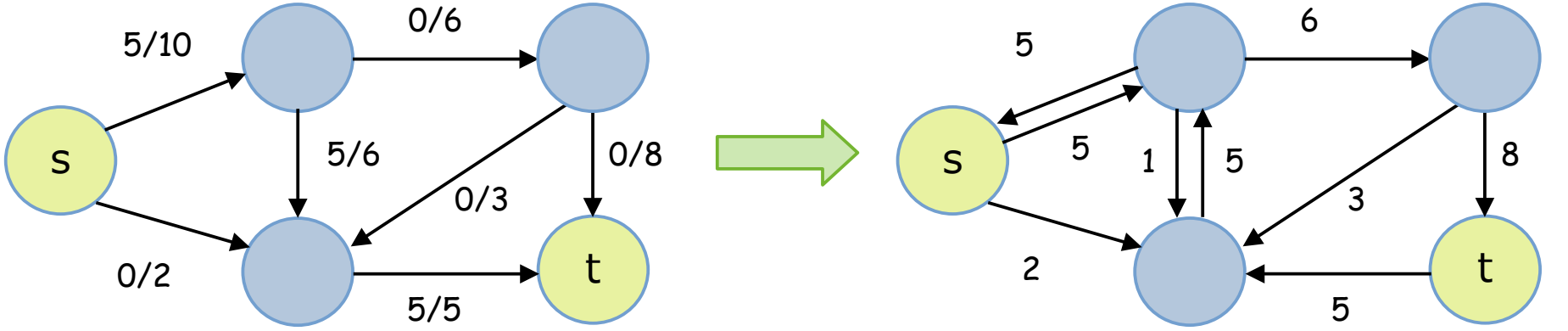


2. 最大フローを求めるアルゴリズムの紹介

- **残余グラフ** について

  流量と容量を表すグラフを **どれだけ流せるかを重みとした辺** と **流量を重みとした逆向きの辺**のグラフに変換したもの（右側の図）

  （重みが 0 となる辺は便宜上図示していない）

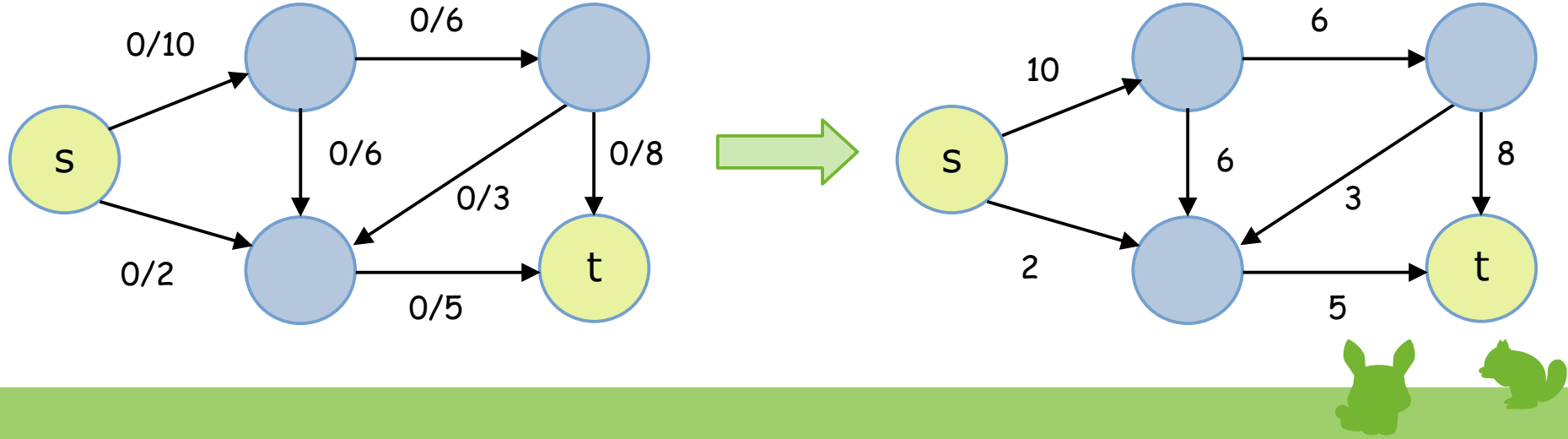


2. 最大フローを求めるアルゴリズムの紹介

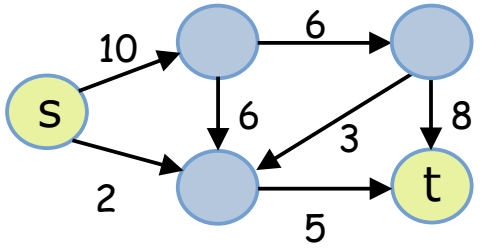
- 最大フローを求めるアルゴリズムの気持ち

  残余グラフ上で先ほどのアルゴリズムをやってみよう

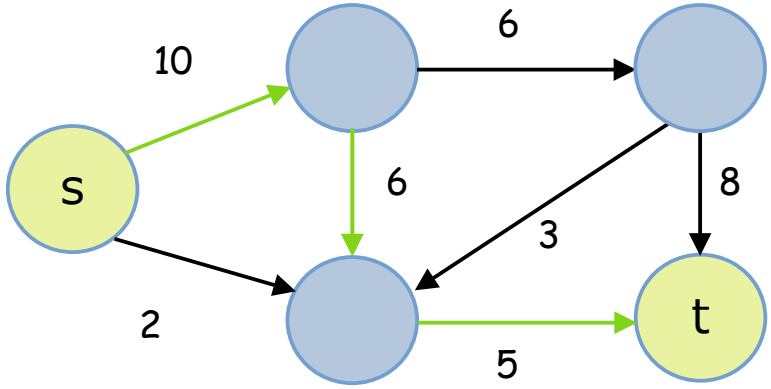
  - 重みが 0 より大きい辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる

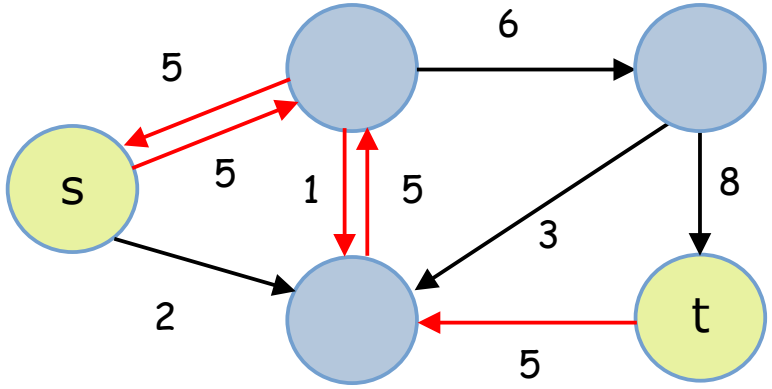


2. 最大フローを求めるアルゴリズムの紹介

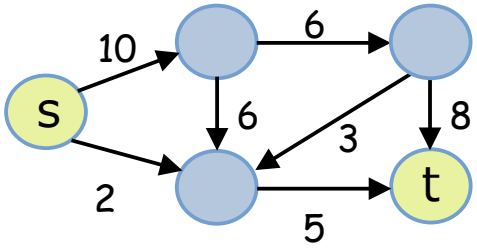


- 重みが 0 より大きい辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる
  - 見つけた s-t パスを緑線、赤線の部分が更新された辺となる
  - スライドの右上には最初のグラフをメモ用に図示してある

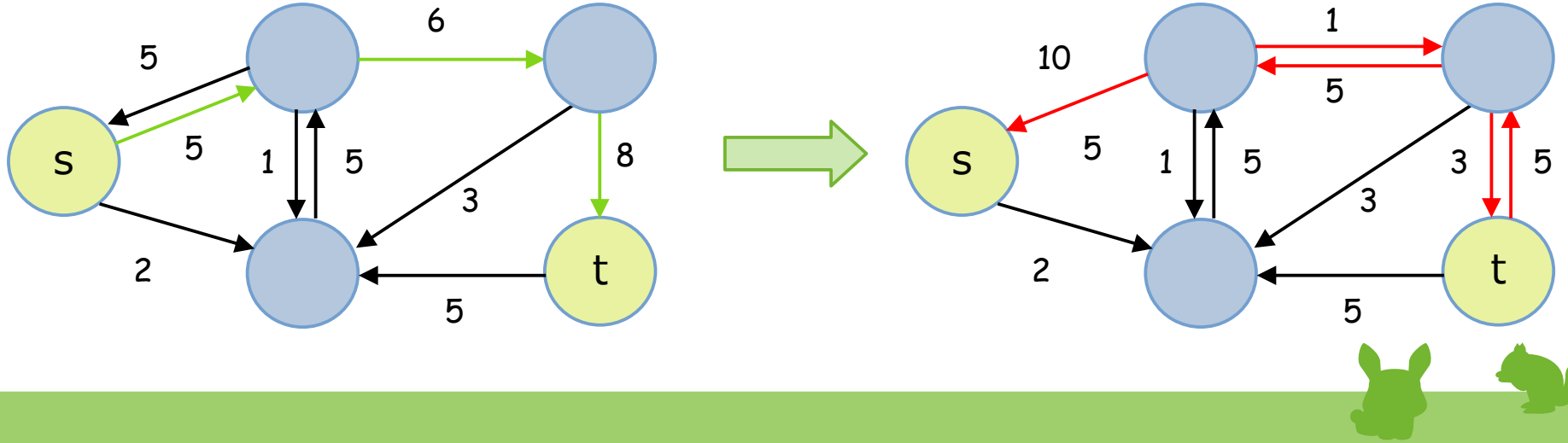




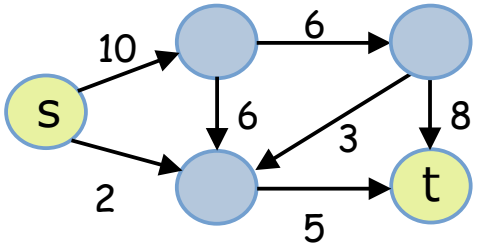
2. 最大フローを求めるアルゴリズムの紹介



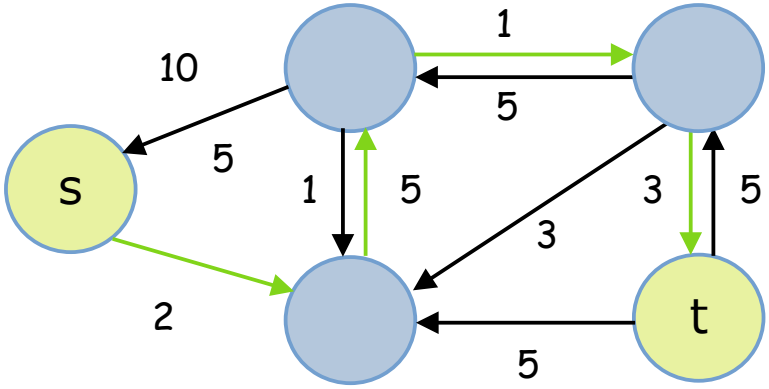
- 重みが 0 より大きい辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる
  - 見つけた s-t パスを緑線、赤線の部分が更新された辺となる
  - スライドの右上には最初のグラフをメモ用に図示してある

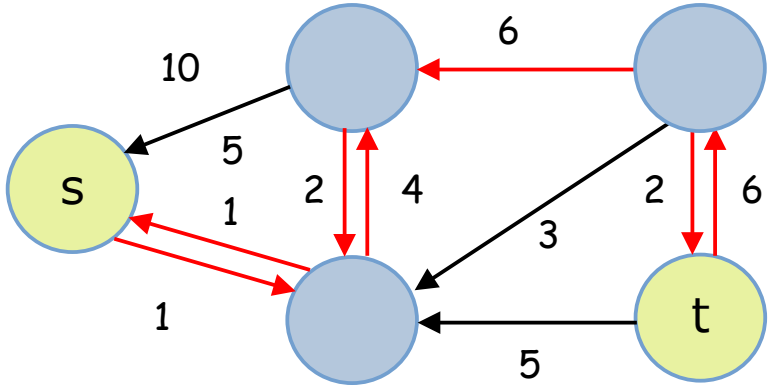


2. 最大フローを求めるアルゴリズムの紹介

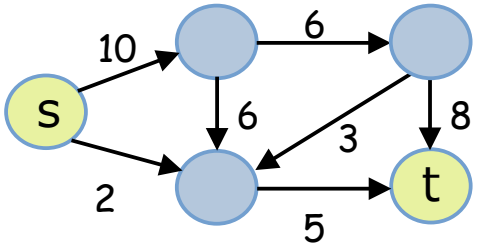


- 重みが 0 より大きい辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる
  - 見つけた s-t パスを緑線、赤線の部分が更新された辺となる
  - スライドの右上には最初のグラフをメモ用に図示してある



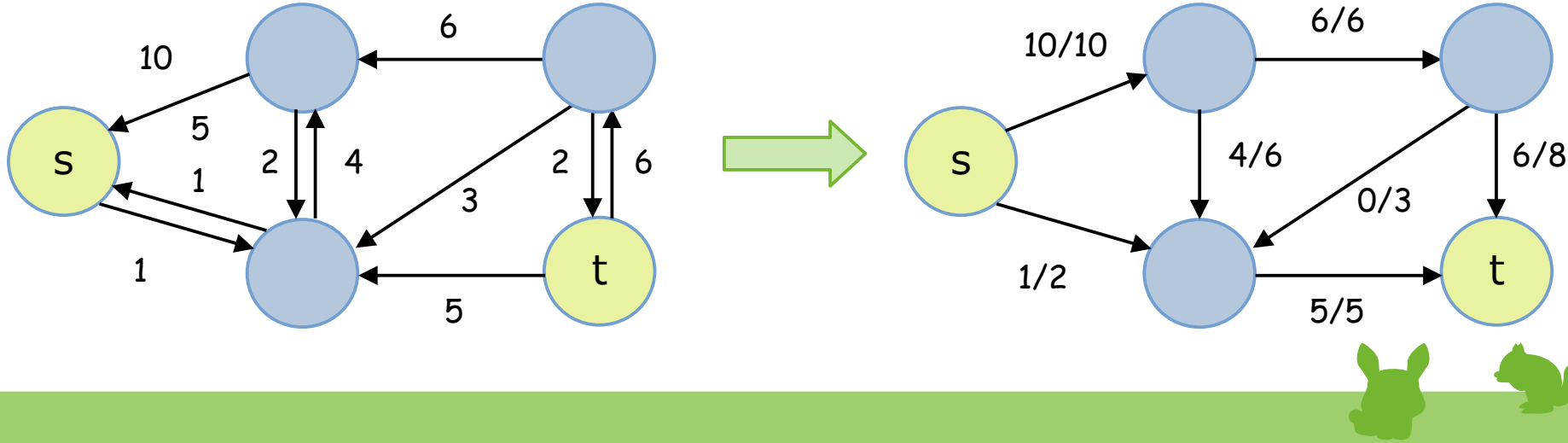


2. 最大フローを求めるアルゴリズムの紹介



- 重みが 0 より大きい辺からなる s-t パスを見つけて「もの」を流せるだけ流してみる
  - これ以上は流せないので終了する

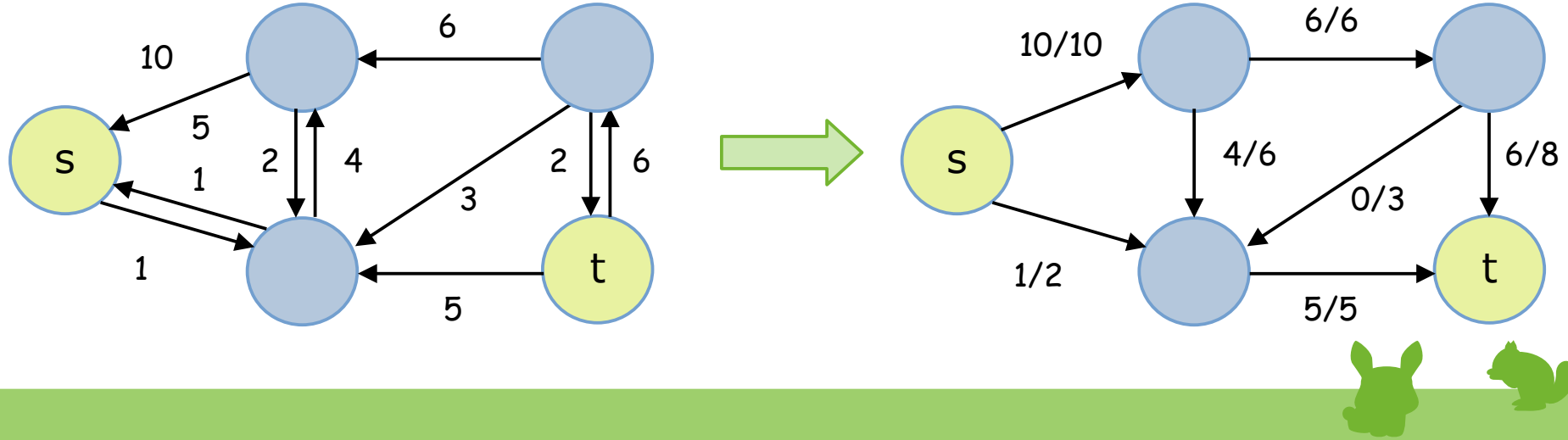
残余グラフを元に戻してみると最大フローが求められている！（最大流量は 11）



2. 最大フローを求めるアルゴリズムの紹介

- なぜこのアルゴリズムで最大フローを求められるのか？
  - 逆の辺を追加することによって既に流したものを押し戻すことができる

    → さらに良いパスが見つかった際に押し戻すことができるため正しい最大フローが求められる



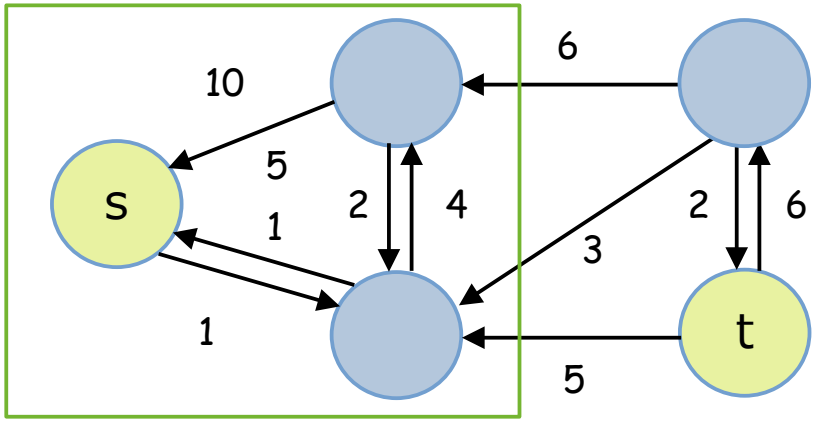
2. 最大フローを求めるアルゴリズムの紹介

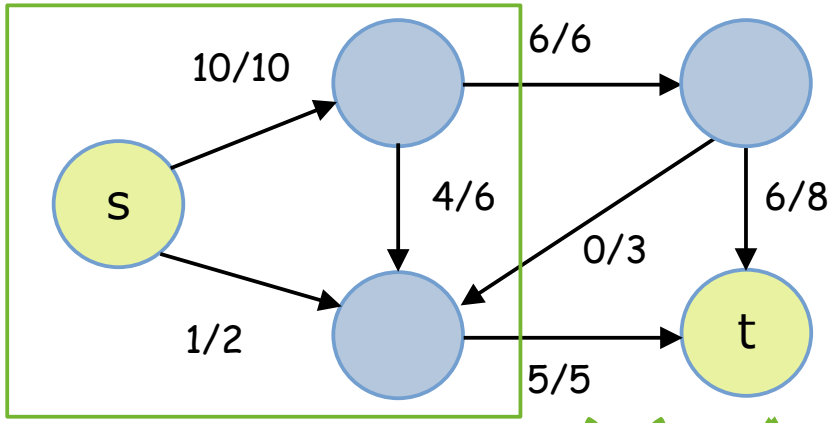
- カットとの関係性

  最終的な残余グラフにおいて、頂点 s から到達することのできる頂点集合（緑線）は **s-t カット**となっている

  → **s-t カット**とは、頂点集合の分割 S であって、s ∈ S であって、t ∉ S を満たすもの であった

  最終的な残余グラフにおいて、s-t パスは存在しないため、上記の条件を満たす



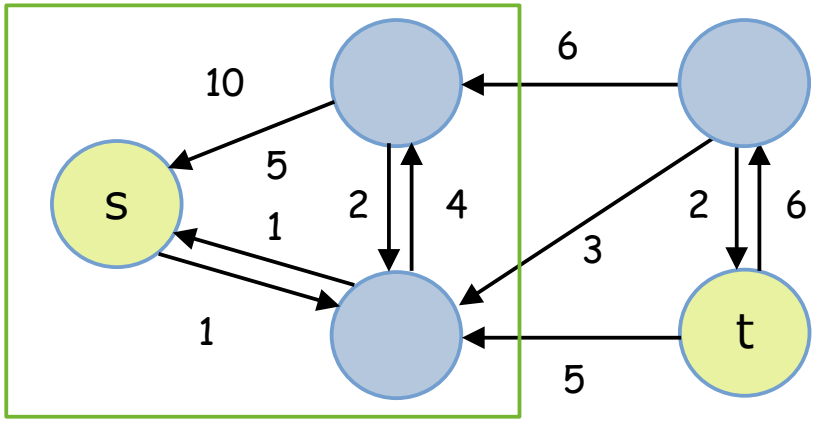


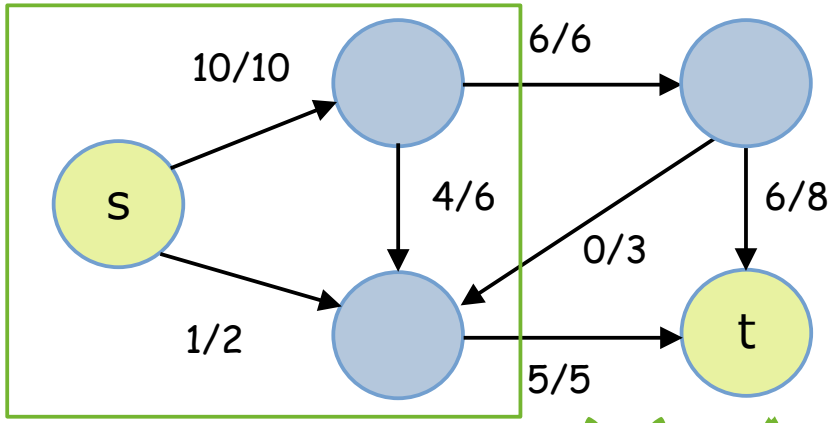
2. 最大フローを求めるアルゴリズムの紹介

- カットとの関係性

  さらに、そのカットにおける容量は**最大流量**と**等しい**（実際、以下の s-t カットの容量は 5 + 6 = 11 で等しい）

  最終的な残余グラフにおいて、s-t パスが存在しないということは、
  カットセットの辺には限界（容量の分）だけ「もの」が流れていることになるため**等しくなる**





2. 最大フローを求めるアルゴリズムの紹介

- カットとの関係性

  ここまでのことをまとめると、以下のようになる

  - アルゴリズムにより得られた s-t カット の容量 = 最大 s-t フローの総流量

  アルゴリズムにより得られた s-t カットの容量については、最小カットの定義より

  - アルゴリズムにより得られた s-t カットの容量 ≥ 最小 s-t カットの容量

  であるから、結果的に以下の事実が成り立つ

  - **最大 s-t フローの総流量 ≥ 最小 s-t カットの容量（重要 ①）**

2. 最大フローを求めるアルゴリズムの紹介

- カットとの関係性

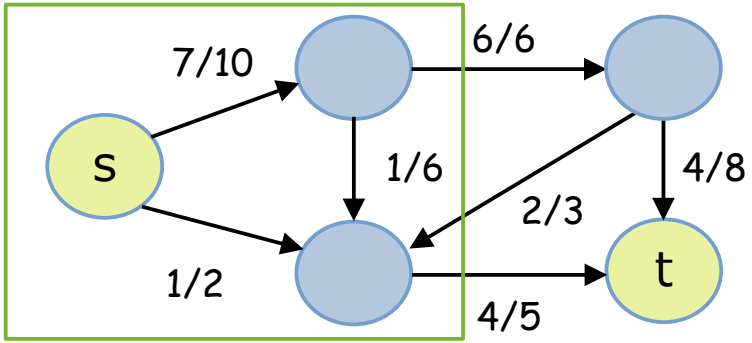
  また、任意の s-t フロー f と任意の s-t カット S について以下が成り立つ

  - **f の総流量 ≤ S の容量**

  f の総流量は、カットを大きな頂点とみなせば以下のようになる

  f の総流量 = S から出ていく辺の総流量 − S に入ってくる辺の総流量

  例えば、右の例では 6 + 4 − 2 = 8 である



  また、S の容量 = S から出ていく辺の容量の総和

  上記に加えて、各辺について 流量 ≤ 容量 であるため成り立つ！

2. 最大フローを求めるアルゴリズムの紹介

- カットとの関係性
  - 任意の s-t フローの総流量 ≤ 任意の s-t カットの容量

  が成り立つということは、
  - 任意の s-t フローということは、最大 s-t フローとしても成り立つ
  - 任意の s-t カットということは、最小 s-t カットとしても成り立つ

  結果として、以下の事実が成り立つ
  - **最大 s-t フローの総流量 ≤ 最小 s-t カットの容量 （重要 ②）**

2. 最大フローを求めるアルゴリズムの紹介

- カットとの関係性

  先ほどの重要な事実な 2 つの事実をまとめると、

  - 最大 s-t フローの総流量 ≥ 最小 s-t カットの容量
  - 最大 s-t フローの総流量 ≤ 最小 s-t カットの容量

  よって、以下のことが言える

  - 最大 s-t フローの**総流量（最大流量）**は最小 s-t カットの**容量**と**等しい**

    → **最大フロー・最小カット定理** が示せた！

2. 最大フローを求めるアルゴリズムの紹介

- 先ほどの例で示したアルゴリズムは**フォード・ファルカーソン法**と呼ばれている

2. 最大フローを求めるアルゴリズムの紹介

- 先ほどの例で示したアルゴリズムは**フォード・ファルカーソン法**と呼ばれている

＜ **フォード・ファルカーソン法**の手順 ＞

1. フロー流量を表す変数 F を F ← 0 と初期化する
2. 残余グラフ G’ を元のグラフ G で初期化する
3. While 残余グラフ G’ において s-t パス P が存在するならば：

   f をパス P 上の各辺の容量の最小値とする

   F += f とする

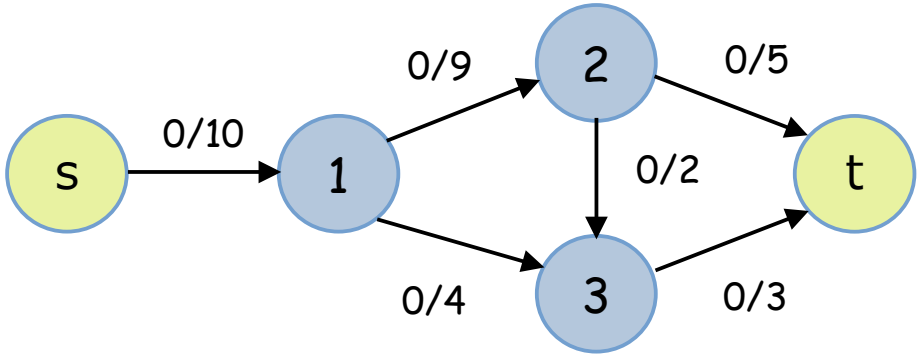
   パス P 上に大きさ f のフローを流す

   残余グラフ G’ を更新する
4. F が求める最大流量となる

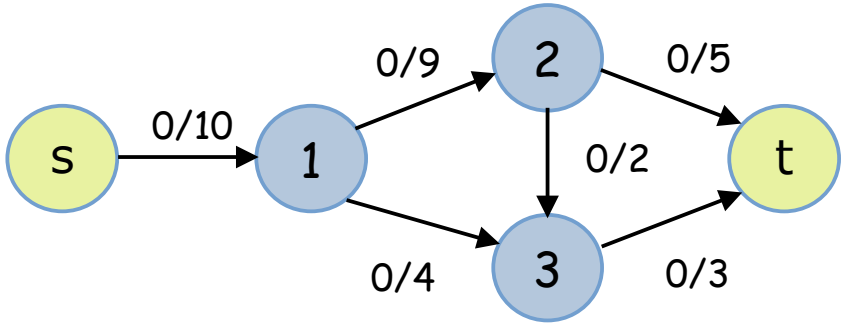
2. 最大フロー問題を解くアルゴリズムの紹介

- 実際にやってみよう

  以下のようなグラフで最大フローを求めてみよう



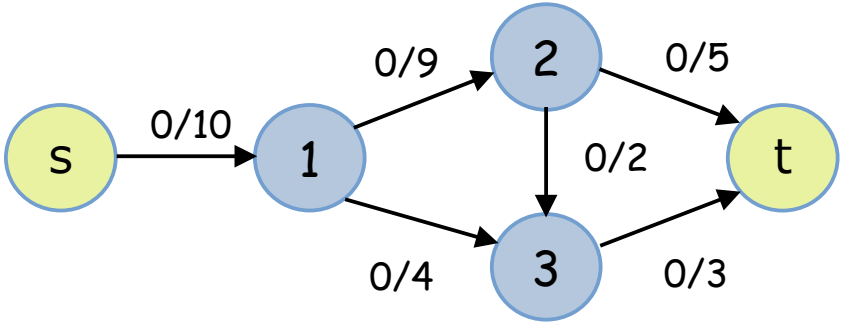
2. 最大フロー問題を解くアルゴリズムの紹介

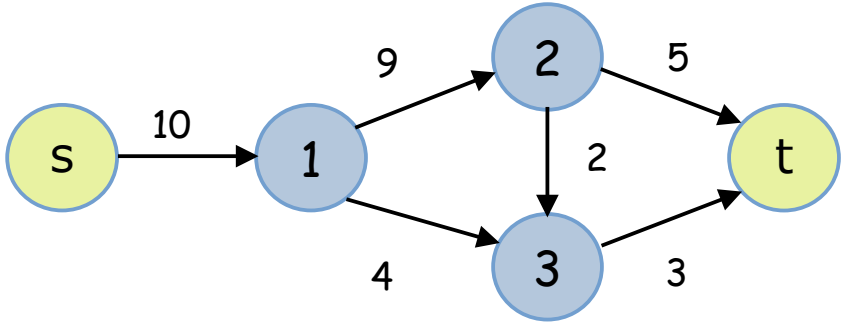


- 実際にやってみよう

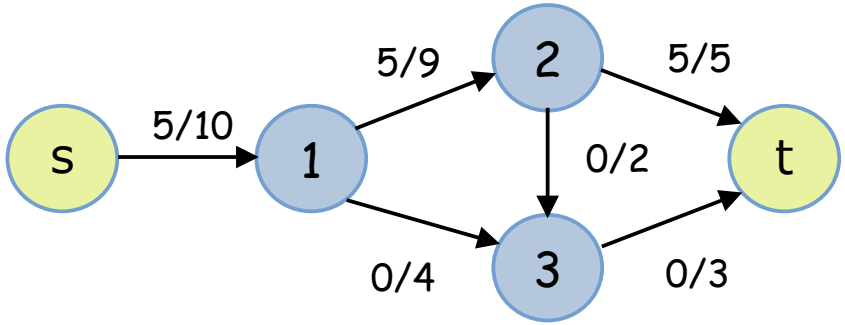
  以下のようなグラフで最大フローを求めてみよう（今回は右上に現在のグラフを図示）

  - グラフを残余グラフとして初期化する
  - F = 0





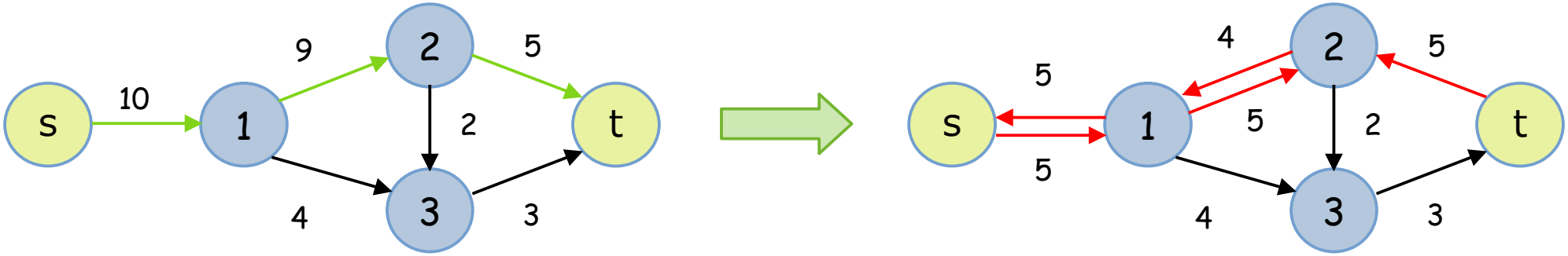
2. 最大フロー問題を解くアルゴリズムの紹介



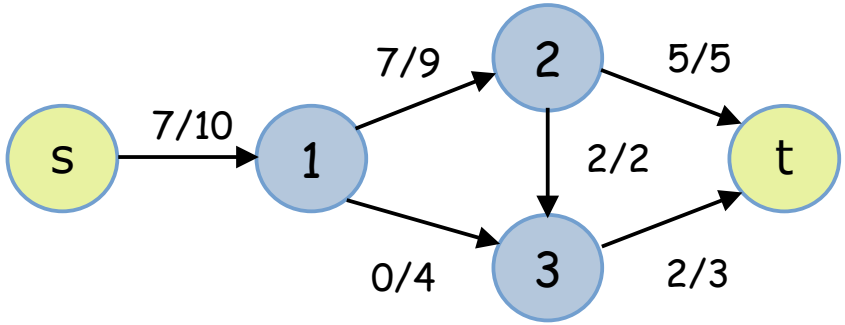
- 実際にやってみよう

  以下のようなグラフで最大フローを求めてみよう（今回は右上に現在のグラフを図示）

  - s-t パスを見つけて（緑線）、残余グラフを更新（赤線）する
  - F = 0 → 5



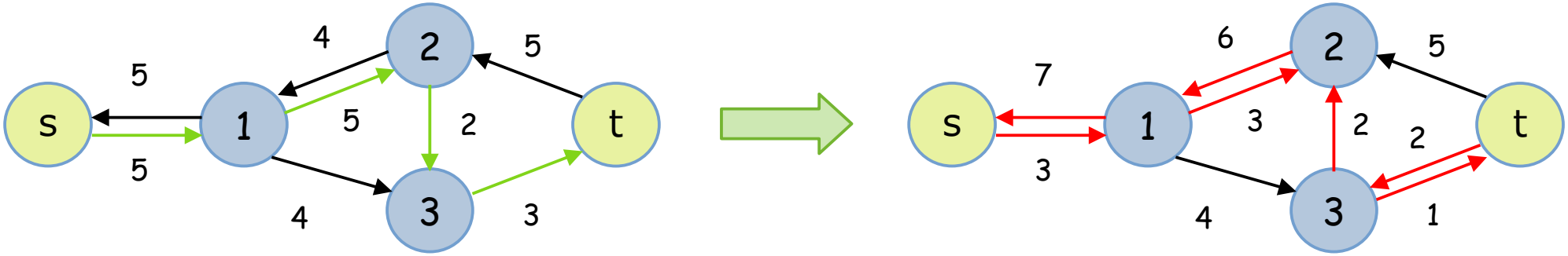
2. 最大フロー問題を解くアルゴリズムの紹介



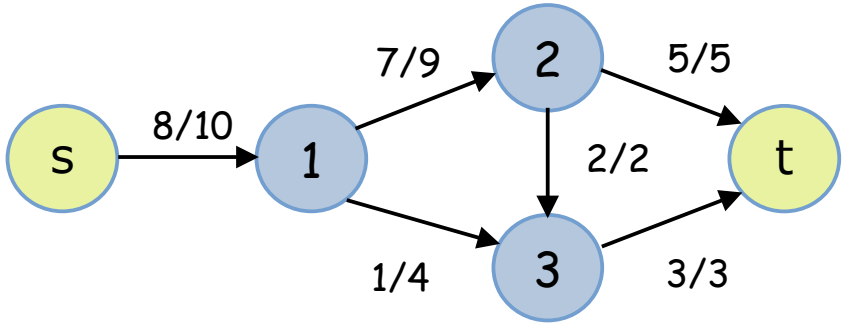
- 実際にやってみよう

  以下のようなグラフで最大フローを求めてみよう（今回は右上に現在のグラフを図示）

  - s-t パスを見つけて（緑線）、残余グラフを更新（赤線）する
  - F = 5 → 7



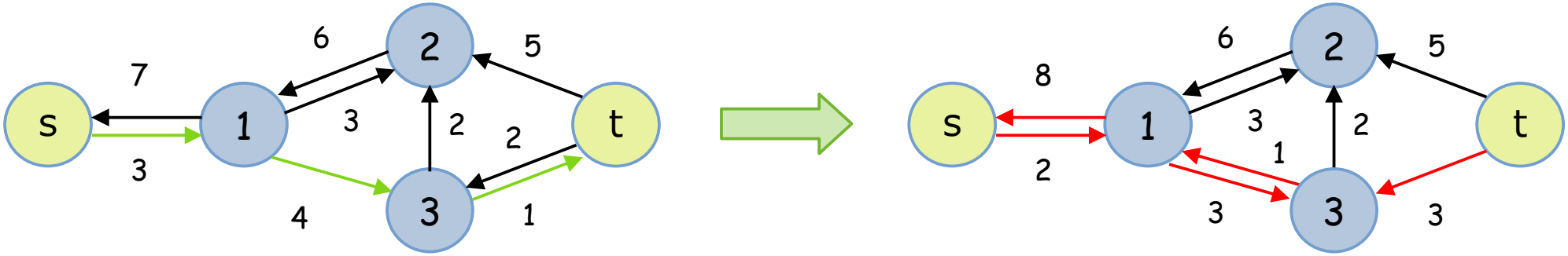
2. 最大フロー問題を解くアルゴリズムの紹介



- 実際にやってみよう

  以下のようなグラフで最大フローを求めてみよう（今回は右上に現在のグラフを図示）

  - s-t パスを見つけて（緑線）、残余グラフを更新（赤線）する
  - F = 7 → 8

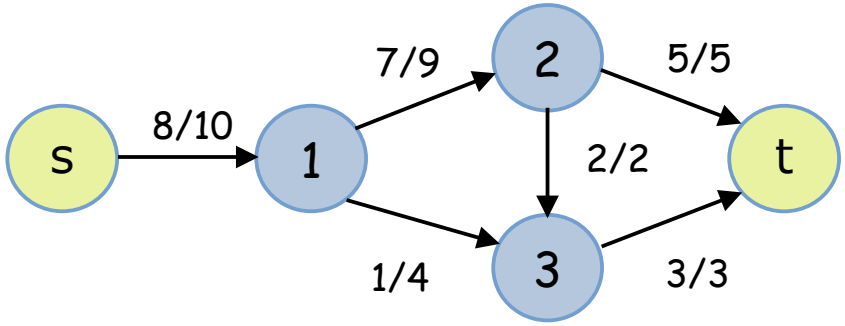


2. 最大フロー問題を解くアルゴリズムの紹介

- 実際にやってみよう

  以下のようなグラフで最大フローを求めてみよう

  - s-t パスが存在しないので、終了する → 最大フローが求められた！
  - F = 8



2. 最大フローを求めるアルゴリズムの紹介

- **フォード・ファルカーソン法**の計算量

  計算に時間がかかりそうなところを考える（ E を辺集合とする ）

  - s-t パスを 1 つ求める
  - パス上にフローを流す
  - 残余グラフの更新

    → いずれもグラフの全体を調べれば可能なため O(|E|)

  また、反復するたびに総流量は少なくとも 1 増加する

  → 最大フローの流量を F とすると、全体の計算量は O(F |E|)

2. 最大フローを求めるアルゴリズムの紹介

- その他の最大フローを求めるアルゴリズム

  最大フローを求めるその他のアルゴリズムとしては Dinic 法がある

  計算量が頂点集合を V、辺集合を E として $O(|V|^2|E|)$ で流量に依存しないため、実際に問題を解く場合には Dinic 法を使ったほうが良い

  有名なライブラリである ACL (AtCoder Library) の実装は Dinic 法となっている

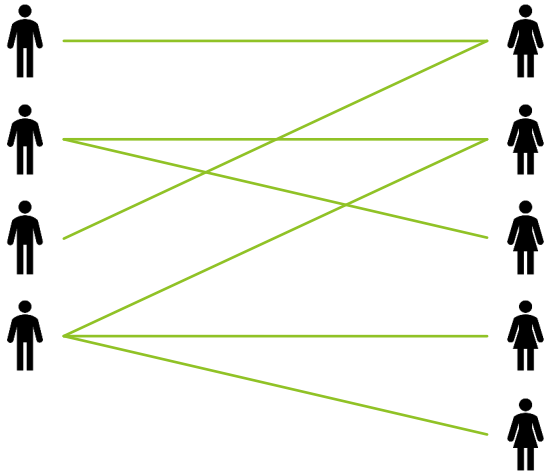
  参考：https://atcoder.github.io/ac-library/production/document_ja/maxflow.html

3. ネットワークフローの応用例

- 応用例１：二部マッチング

  男女が何人かおり、相性の良い男女の組が分かっている

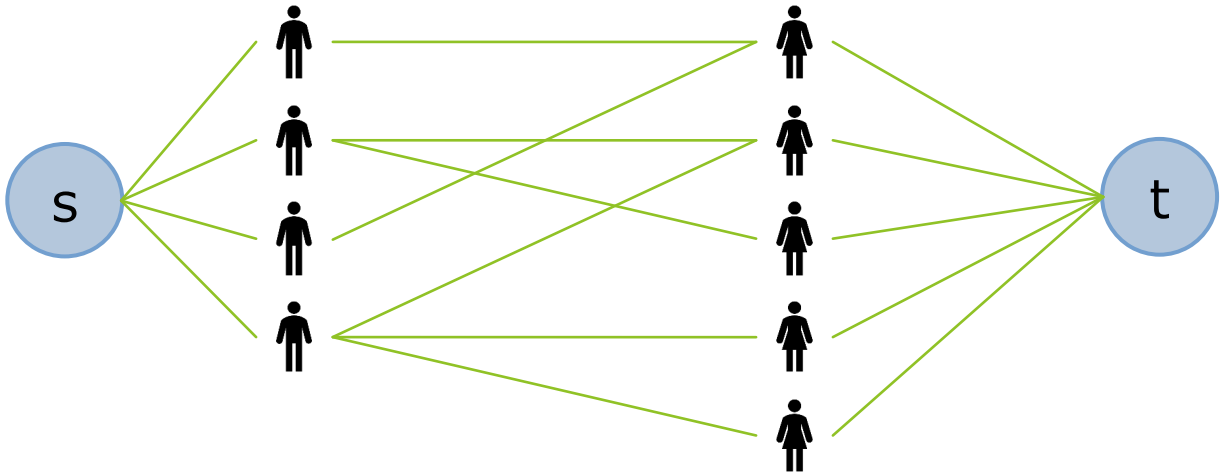
  最大で何組の男女のペアを作ることができるか？



3. ネットワークフローの応用例

- 応用例１：二部マッチング

  新たな頂点 s, t を用意し、以下のように辺を張ると頂点 s から t の最大フローを求める問題に帰着される。（辺の容量は１とする）

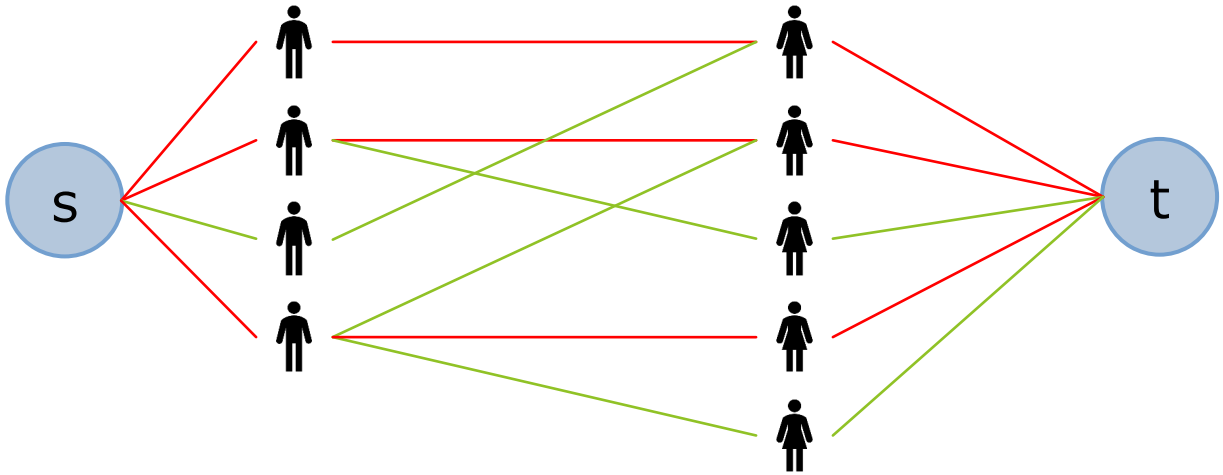


3. ネットワークフローの応用例

- 応用例 1 ：二部マッチング

  実際に最大フローを求めてみると以下のようになる。（赤線の部分）

  最大流量が 3 であるため、男女のペアを 3 組作れることが分かる。



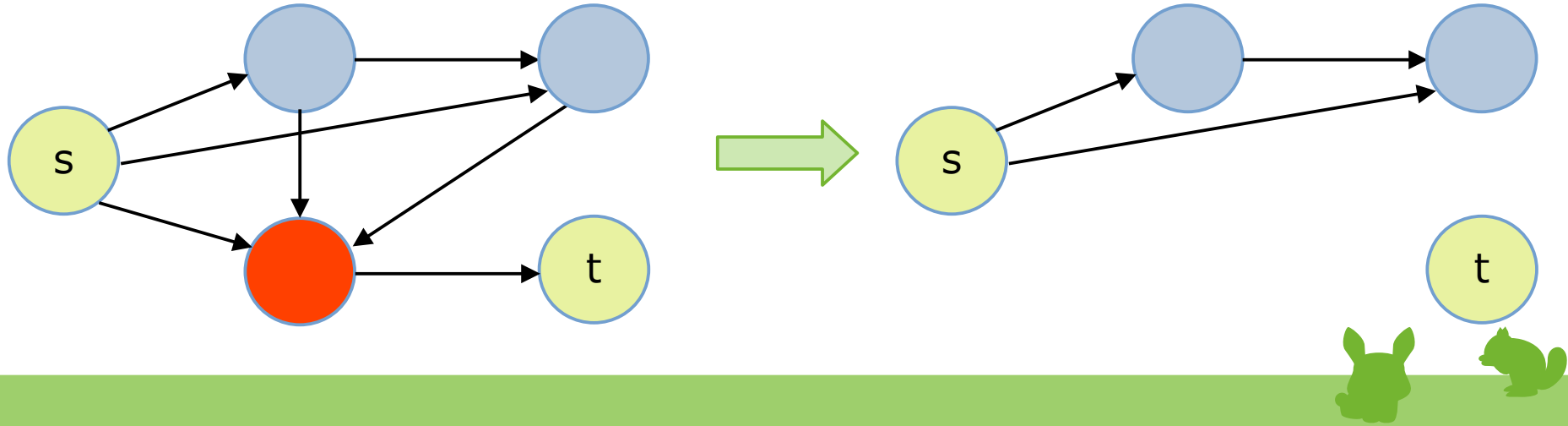
3. ネットワークフローの応用例

- 応用例２：点連結度

  s-t 点連結度とは、グラフのある頂点 s から t を**非連結**にするために必要な取り除く頂点の個数の**最小値**

  → s-t 辺連結度の頂点版みたいなもの

  例えば、以下のグラフにおける **s-t 点連結度**は 1 （頂点 s, t は取り除けないことに注意）

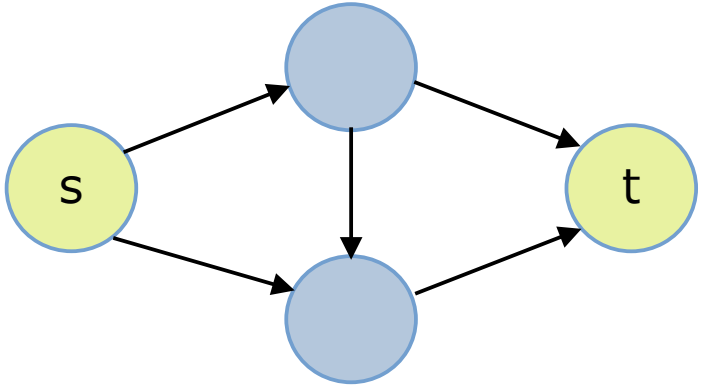


3. ネットワークフローの応用例

- **辺連結度**について（再掲）

  s-t 辺連結度とは、グラフのある頂点 s から t を**非連結**にするために必要な取り除く辺の個数の**最小値**

  例えば、以下のグラフにおける **s-t 辺連結度**は 2



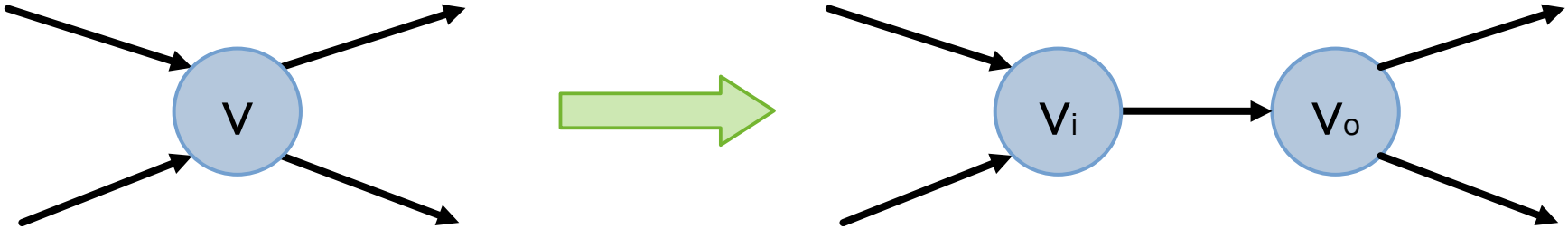
3. ネットワークフローの応用例

- 応用例２：点連結度

  s-t 点連結度とは、グラフのある頂点 s から t を**非連結**にするために必要な取り除く頂点の個数の**最小値**

  → s-t 辺連結度の頂点版みたいなもの

  s-t 辺連結度は、最大フローを求めることによって求められるが、s, t 点連結度もグラフの形を変形させることにより、辺連結度の場合に帰着できる



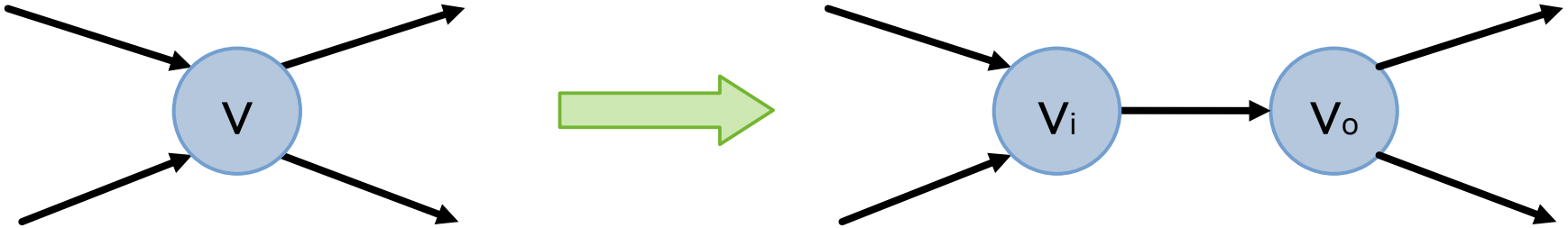
3. ネットワークフローの応用例

- 応用例２：点連結度

  ある頂点に対して 辺が**入ってくる**専用の頂点 (V_i) と 辺が**出ていく**専用の頂点(V_o) に分割する

  そうすると、頂点を取り除くということが V_i から V_o への辺を取り除くことと等価になる

  → 辺連結度と同じ問題に帰着される



3. ネットワークフローの応用例

- 応用例３：プロジェクト選択問題

  プロジェクト選択問題と呼ばれる問題も最大フローを求めることによって解くことができる

  プロジェクト選択問題とは以下のような問題

  N 個のボタンがあり、i 番目のボタンを押すと A_i 、押さない場合は B_i のコストがかかる。

  しかし、以下のような制約がいくつかある。

  - U_i 番目のボタンを押したならば V_i 番目のボタンを押さなければならない

  かかる合計のコストの最小値を求めよ。

3. ネットワークフローの応用例

- 応用例 3 ：プロジェクト選択問題

  具体例で考えてみよう

  2 個のボタンがあり、それぞれのボタンを押した場合に、

  - 1 番目のボタンを押すと 10、押さない場合は 30 のコスト
  - 2 番目のボタンを押すと 50、押さない場合は 40 のコスト

  が得られる。

  しかし、以下のような制約がある。

  - 1 番目のボタンを押したならば 2 番目のボタンを押さなければならない

  かかる合計のコストの最小値を求めよ。

3. ネットワークフローの応用例

メモ	押す	押さない
ボタン1	10	30
ボタン2	50	40

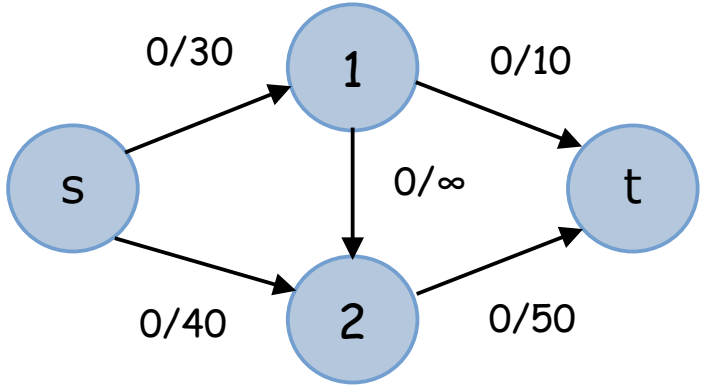
- 応用例３：プロジェクト選択問題

  考え方：制約は以下のように置き換えられる

  - 1 番目のボタンを押したならば 2 番目のボタンを押さなければならない

  → 1 番目のボタンを押して 2 番目のボタンを押さない場合は ∞ のコストがかかる

  s, t カットにおいて、s に含まれる頂点を押すボタン、t に含まれる頂点を押さないボタンとすれば、以下のグラフの最小カットが求める答えとなる



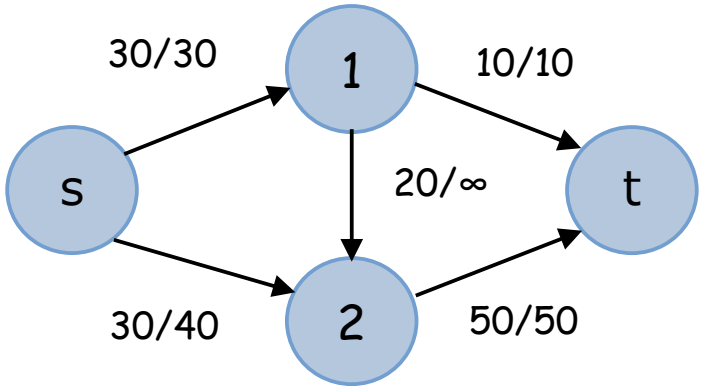
3. ネットワークフローの応用例

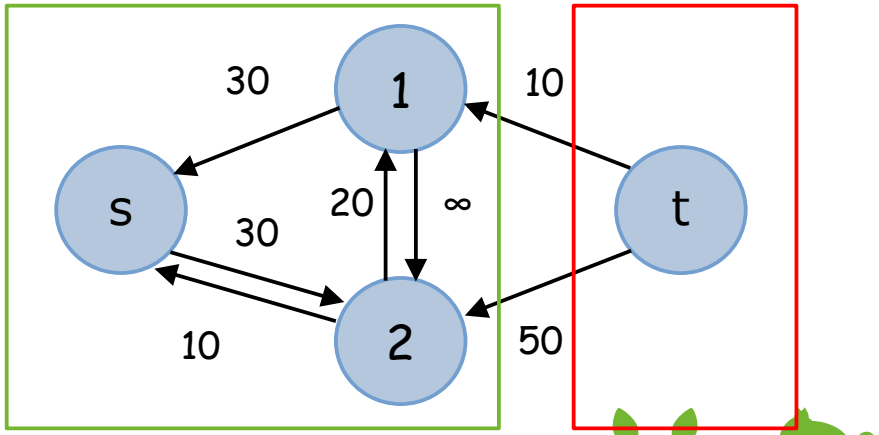
メモ	押す	押さない
ボタン1	10	30
ボタン2	50	40

- 応用例３：プロジェクト選択問題

  実際に最大フローを求めてみると以下のようになる

  右側の残余グラフより最小カットを求めることができ、両方のボタンを押した場合にコストが最小になることが分かる





3. ネットワークフローの応用例

- 応用例３：プロジェクト選択問題

  余談：この問題の素朴な解法を考えてみる

  素朴な解法：ビット全探索によって、ボタンの押し方を全探索する

  → N についての指数時間となるので、N が大きいと時間がかかる……

  最大フローを求めれば多項式時間で解けるため、フローは非常に強力であることが分かる

4. ネットワークフローを用いた問題の例

- 例題を 1 問やってみよう（ ABC239-G Builder Takahashi ）

  URL: https://atcoder.jp/contests/abc239/tasks/abc239_g

（問題文が長かったため次のページに載せています）

4. ネットワークフローを用いた問題の例

問題文

N 頂点 M 辺の単純連結無向グラフがあります。
頂点には頂点 1, 頂点 2, $\ldots$, 頂点 N と番号が振られています。
辺には辺 1, 辺 2, $\ldots$, 辺 M と番号が振られています。辺 i は頂点 a_i と頂点 b_i を双方向に結んでいます。また、頂点 1 と頂点 N を直接結ぶ辺は存在しません。
各頂点の状態は「何もない頂点」か「壁がある頂点」のいずれかです。はじめ、全ての頂点は何もない頂点です。

青木君は頂点 1 を出発して、グラフ上を辺に沿って移動して頂点 N へ行こうとしています。ただし、壁がある頂点への移動を行うことはできません。

高橋君は、青木君が移動を開始する前にいくつかの頂点を選んで壁を建てることで、青木君がどのように移動しても頂点 N へ行くことができないようにすることにしました。
高橋君が頂点 i に壁を建てるには c_i 円を払う必要があります。また、**頂点 1 および頂点 N には壁を建てられません。**

高橋君が条件を満たすように壁を建てるときに最小で何円払う必要があるでしょうか。また、その時の壁の立て方を 1 つ出力してください。

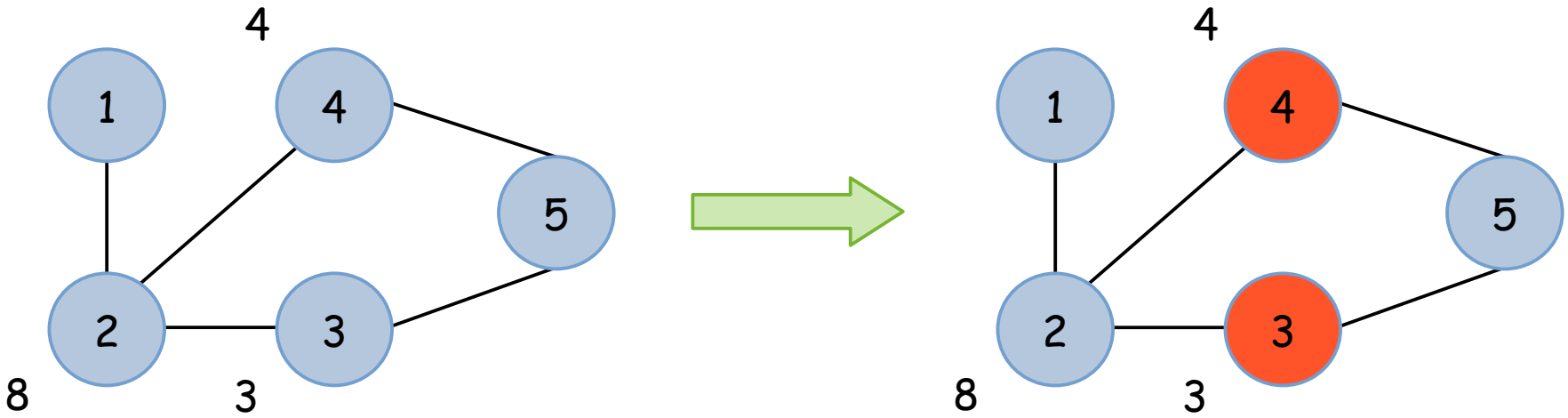
制約

- $3 \leq N \leq 100$
- $N-1 \leq M \leq \frac{N(N-1)}{2} - 1$
- $1 \leq a_i < b_i \leq N\ (1 \leq i \leq M)$
- $(a_i, b_i) \neq (1, N)$
- 与えられるグラフは単純かつ連結である。
- $1 \leq c_i \leq 10^9\ (2 \leq i \leq N-1)$
- $c_1 = c_N = 0$
- 入力はすべて整数である。

4. ネットワークフローを用いた問題の例

- 例題を 1 問やってみよう（ ABC239-G Builder Takahashi ）

＜ サンプル 1 ＞



頂点 3 と頂点 4 に壁を立てれば
3 + 4 = 7 円で最小

4. ネットワークフローを用いた問題の例

- 例題を 1 問やってみよう（ ABC239-G Builder Takahashi ）

＊少し考えてみよう

ヒント：青木くんがどのように移動しても頂点 1 から N へ移動できない

→ グラフの最小カット？

4. ネットワークフローを用いた問題の例

- 解法について

  頂点 1 から N へ移動できないようにしたい

  → 最小カットを求めたい

  → 最小カットは最大流を求めれば求められる

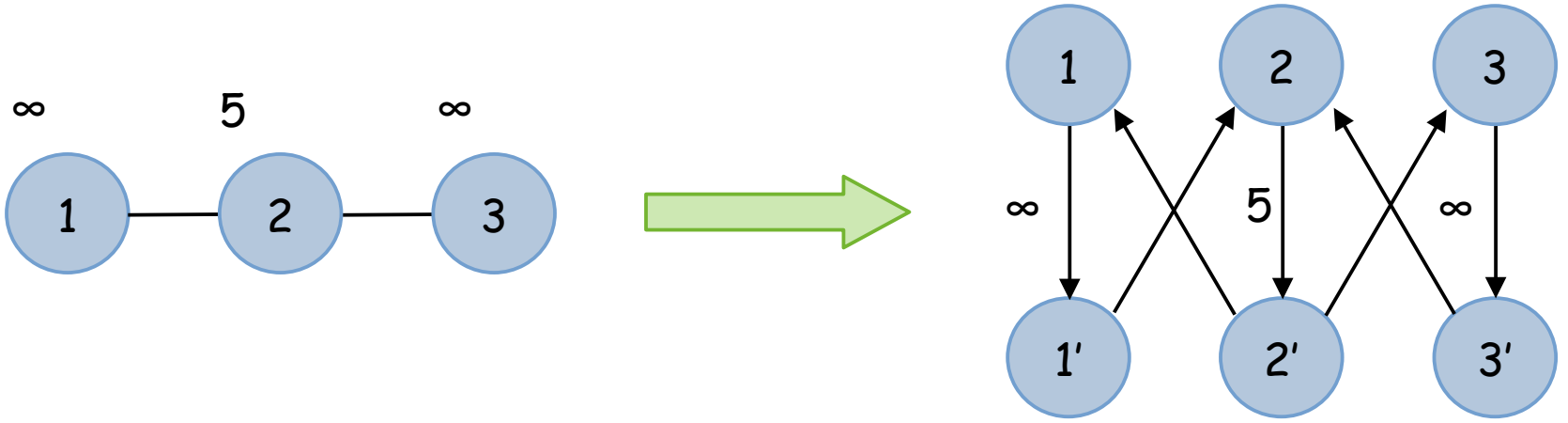
  しかし、今回は辺ではなく頂点で分かれてしまう……

  → 先ほどの応用例２のようにグラフを変形して最小カットを求める問題に置き換える

4. ネットワークフローを用いた問題の例

- グラフを以下のように変形させる (変形後のグラフで何も書いていない辺は ∞ とする)

  無向グラフを双方向の有向グラフと見なし、辺を出す専用の頂点と入れる専用の頂点に分ける

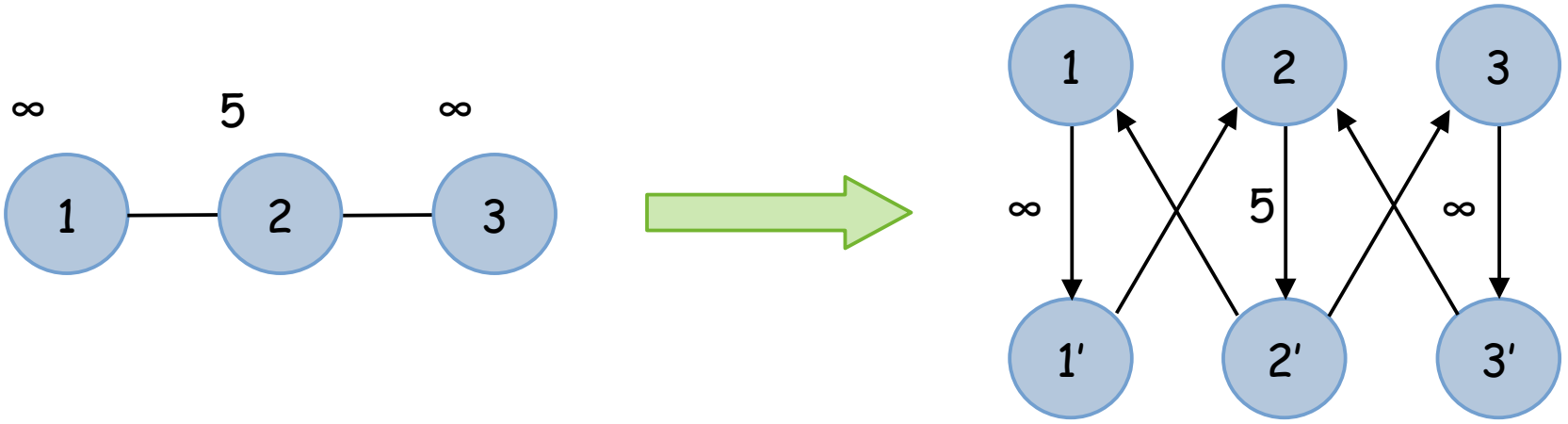


4. ネットワークフローを用いた問題の例

- このように変形させると何が良いのか

  頂点 v に壁を立てるということは頂点 v から v′ への辺を取り除くのと同じ

  → 最小カットに帰着できる！



4. ネットワークフローを用いた問題の例

- 解法について

  頂点 1 から N へ移動できないようにしたい

  → 最小カットを求めたい

  → 最小カットは最大流を求めれば求められる

グラフを上手く変形すれば最小カットを求める問題に帰着できる！

4. ネットワークフローを用いた問題の例

- 実装例（C++）

```cpp
const long long INF = 1LL << 40;

int main(){
  // 入力を受け取る
  int n, m; cin >> n >> m;
  vector<int> a(m), b(m), c(n);
  for(int i = 0; i < m; i++){
    cin >> a[i] >> b[i];
    a[i]--, b[i]--;
  }
  for(int i = 0; i < n; i++){
    cin >> c[i];
  }

  // 変形させたグラフを構築する
  // 2i ... 辺が入る専用の頂点
  // 2i + 1 ... 辺が出る専用の頂点
  mf_graph<long long> G(2 * n);
  for(int i = 0; i < m; i++){
    G.add_edge(a[i] * 2 + 1, b[i] * 2, INF);
    G.add_edge(b[i] * 2 + 1, a[i] * 2, INF);
  }

  for(int i = 0; i < n; i++){
    if(i == 0 || i == n - 1){
      G.add_edge(i * 2, i * 2 + 1, INF);
    }else{
      G.add_edge(i * 2, i * 2 + 1, c[i]);
    }
  }

  // 最大フローを求める
  long long f = G.flow(0 * 2, (n - 1) * 2 + 1);
  vector<bool> mincut = G.min_cut(0 * 2);
  vector<int> ans;
  for(int i = 0; i < n; i++){
    // 残余グラフ上で 頂点 2i から 2i + 1 に移動できない
    // -> 壁が立っている
    if(mincut[i * 2] && !mincut[i * 2 + 1]){
      ans.push_back(i);
    }
  }

  // 答えを出力する
  cout << f << endl;
  cout << (int) ans.size() << endl;
  for(auto x : ans) cout << x + 1 << " ";
  cout << endl;
}
```

5. 演習問題など

下の問題ほど難しいです！（「MUL」から下の問題は同じくらいの難易度かも）

- 典型アルゴリズム問題集 上級～エキスパート編「最大流」
- ABC010-D「浮気予防」
- ABC239-G「Builder Takahashi」（ 今回扱った例題 ）
- ARC085-E「MUL」
- ABC274-G「Security Camera 3」
- ABC193-F「Zebraness」