数理計画法 第7回

第3章 ネットワーク計画
§3.2 最大流問題とフロー増加法

担当： 塩浦昭義
(情報科学研究科 准教授)
shioura@dais.is.tohoku.ac.jp

分枝限定法の考え方

- 分枝限定法：組合せ計画問題を分枝操作と限定操作を使って解く解法
- 組合せ計画問題を，場合分けによって部分問題に分解（分枝操作）
  - 0–1ナップサック問題：各変数について 0 の場合と 1 の場合に分ける
  - 分枝の進行の様子は探索木により表現可能
  - 分枝操作により，たくさんの部分問題が生成される
- 解く必要のない（解いても無駄な）部分問題が検出されたら，
  さらなる分枝操作をストップ（限定操作）
  - 暫定解の保持と緩和問題の利用により，無駄をチェック

0-1ナップサック問題の探索木

目的関数 $10x_1 + 7x_2 + 25x_3 \rightarrow$ 最大

制約条件 $2x_1 + x_2 + 6x_3 \leq 7$

$x_j \in \{0, 1\}\ (j = 1, 2, 3)$

x_1を0に固定

目的関数 $7x_2 + 25x_3 \rightarrow$ 最大

制約条件 $x_2 + 6x_3 \leq 7$

$x_2, x_3 \in \{0, 1\}$

部分問題

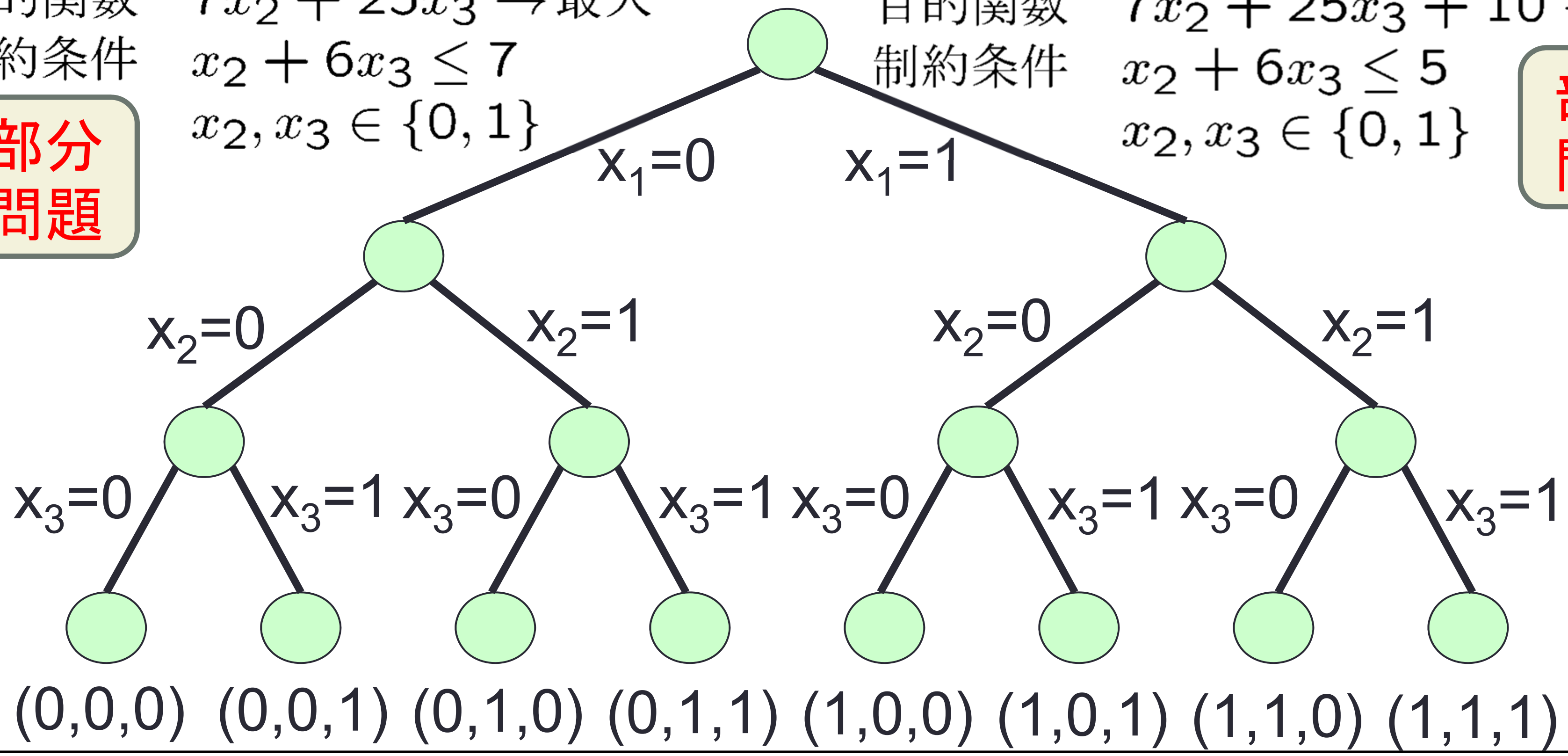
x_1を1に固定

目的関数 $7x_2 + 25x_3 + 10 \rightarrow$ 最大

制約条件 $x_2 + 6x_3 \leq 5$

$x_2, x_3 \in \{0, 1\}$

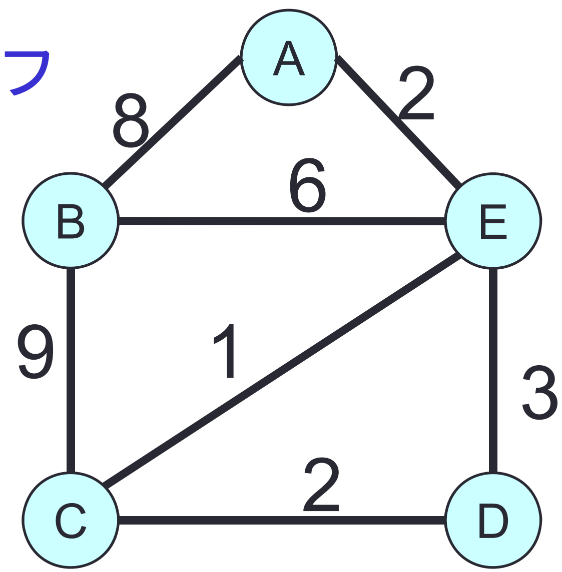
部分問題



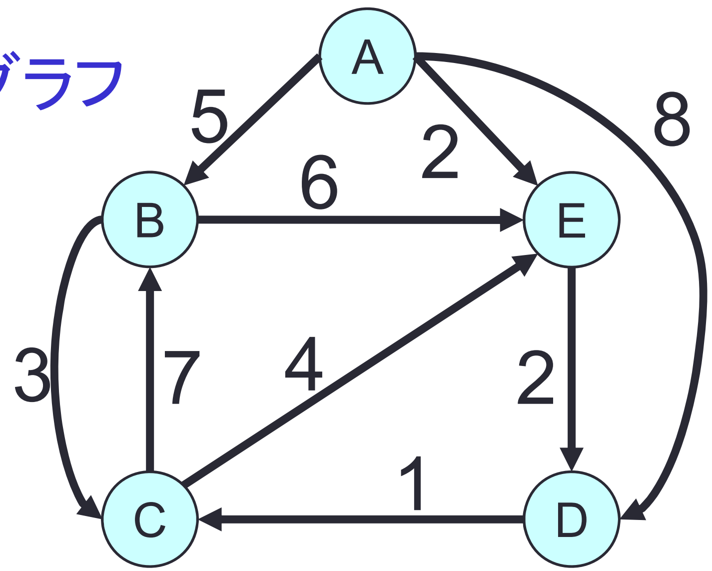
ネットワーク計画問題

- （無向、有向）グラフ
  - 頂点（vertex, 接点、点）が枝（edge, 辺、線）で結ばれたもの
- ネットワーク
  - 頂点や枝に数値データ（距離、コストなど）が付加されたもの
- ネットワーク計画問題
  - ネットワークを使って表現される数理計画問題

無向グラフ



有向グラフ



ネットワーク計画問題の例

「ネットワーク」に関する数理計画問題(最適化問題)

例: 最小木問題
(minimum spanning tree prob.)
最短路問題
(shortest path prob.)

他の講義で扱う
「アルゴリズムとデータ構造」
「情報数学」

最大流問題
(maximum flow prob.)
最小費用流問題
(minimum cost flow prob.)
割当問題
(assignment prob.)

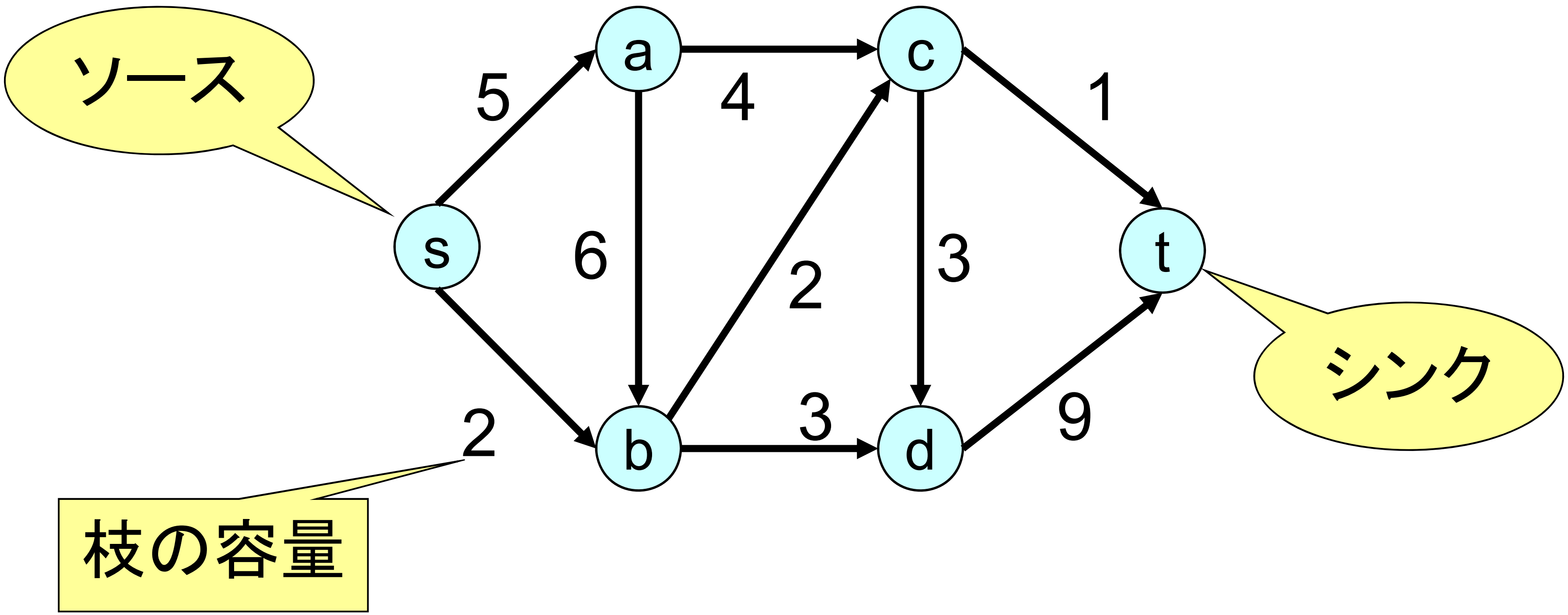
この授業で扱う

最大流問題の定義(その1)

入力：有向グラフ G = (V, E)

ソース（供給点） $s \in V$, シンク（需要点） $t \in V$

各枝 $(i, j) \in V$ の容量 $u_{ij} \geqq 0$



最大流問題の定義(その2)

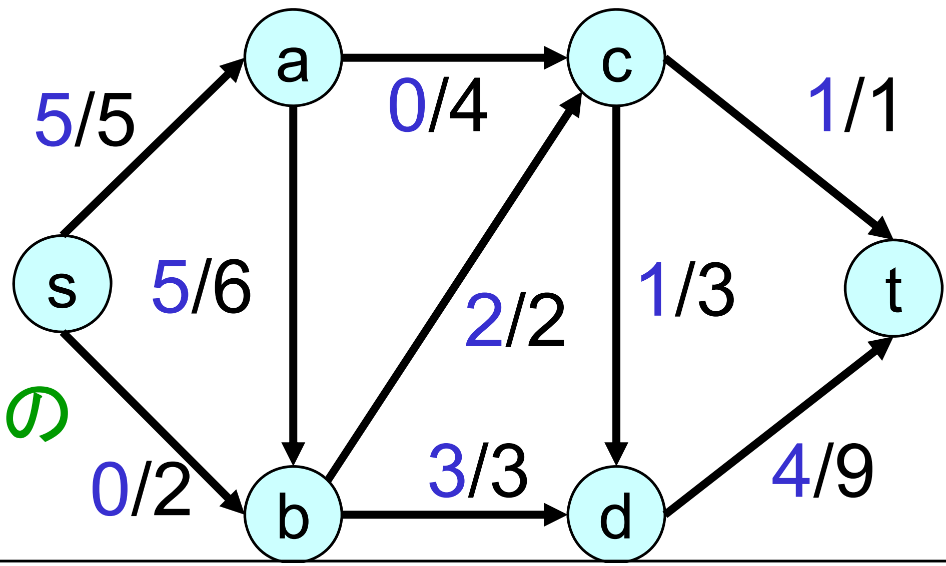
目的:ソースからシンクに向けて、枝と頂点を経由して「もの」を出来るだけたくさん流す

条件1(容量条件, capacity constraint):

0 ≦ 各枝を流れる「もの」の量 ≦ 枝の容量

条件2(流量保存条件, flow conservation constraint):

頂点から流れ出す「もの」の量＝ 流れ込む「もの」の量

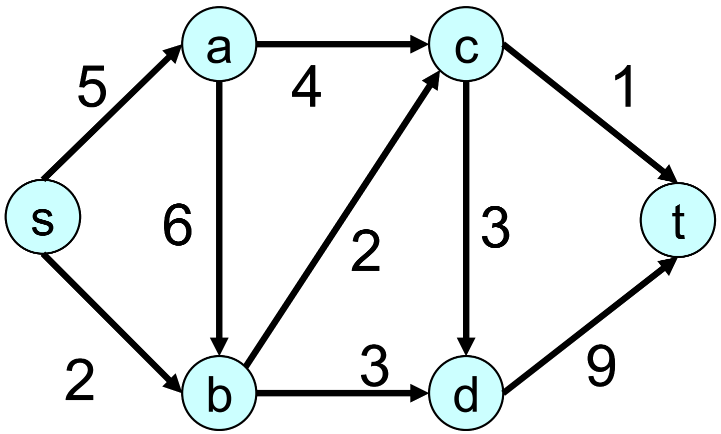


実行可能解の例

最大流問題の定式化:
変数，目的関数と容量条件

変数 x_{ij}: フロー＝枝 (i, j) を流れる「もの」の量
変数 f: ソースからシンクへの総流量
＝シンクに流れ込む「もの」の量＝ ソースから流れ出す「もの」の量

目的:ソースからシンクに「もの」をたくさん流したい
⇒ 目的関数 f → 最大
容量条件:0 ≦ 各枝を流れる「もの」の量 ≦ 枝の容量
⇒ $0 \leqq x_{ij} \leqq u_{ij} \quad ((i,j) \in E)$



具体例
目的: 最大化 f
容量条件:
$0 \leqq x_{sa} \leqq 5$, $0 \leqq x_{sb} \leqq 2$, $0 \leqq x_{ab} \leqq 6$,
$0 \leqq x_{ac} \leqq 4$, $0 \leqq x_{bc} \leqq 2$,
…

最大流問題の定式化: 流れ保存則

流れ保存則(流量保存条件):

(頂点から流れ出す「もの」の量) － (流れ込む「もの」の量) ＝ 0

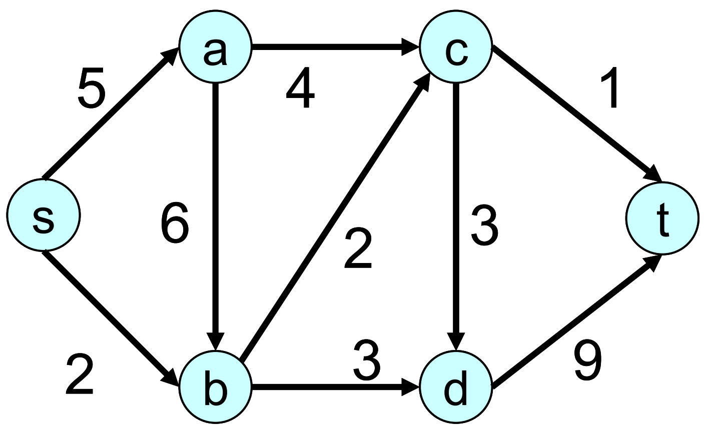
⇒ $\sum\{x_{kj}$ | 枝 (k,j) は 頂点 k から出る$\}$

　　－ $\sum\{x_{ik}$ | 枝 (i,k) は 頂点 k に入る$\}$ ＝ 0 　$(k \in V - \{s, t\})$

ソースとシンクに関する条件:

$\sum\{x_{sj}$ | (s,j) は s から出る$\}$ - $\sum\{x_{is}$ | (i,s) は s に入る$\}$ = f

$\sum\{x_{tj}$ | (t,j) は t から出る$\}$ - $\sum\{x_{it}$ | (i,t) は t に入る$\}$ = - f



流れ保存則の例:

$x_{ac} + x_{ab} - x_{sa} = 0$

$x_{bc} + x_{bd} - x_{ab} - x_{sb} = 0$

$x_{ct} + x_{cd} - x_{ac} - x_{cb} = 0$

$x_{dt} - x_{cd} - x_{bd} = 0$

$x_{sa} + x_{sb} = f,$ 　　$- x_{ct} - x_{dt} = - f$

最大流問題の定式化:まとめ

目的関数 f → 最大

条件 $0 \leqq x_{ij} \leqq u_{ij}$ $((i,j) \in E)$

$\sum\{x_{kj} \mid (k,j)$ は k から出る$\}$

$- \sum\{x_{ik} \mid (i,k)$ は k に入る$\} = 0$ $(k \in V - \{s, t\})$

$\sum\{x_{sj} \mid (s,j)$ は s から出る$\}$

$- \sum\{x_{is} \mid (i,s)$ は s に入る$\} = f$

$\sum\{x_{tj} \mid (t,j)$ は t から出る$\}$

$- \sum\{x_{it} \mid (i,t)$ は t に入る$\} = - f$

この問題の許容解 x_{ij} --- フロー(flow)

フローの目的関数値 f --- 流量

最大流問題の応用例

- 物流
- シーズン途中でのプロ野球チームの優勝可能性判定
  - 残り試合全勝しても優勝の可能性がないかどうか？
- 画像処理における物体の切り出し
  - 画像内の物体のみ取り出す
- その他多数

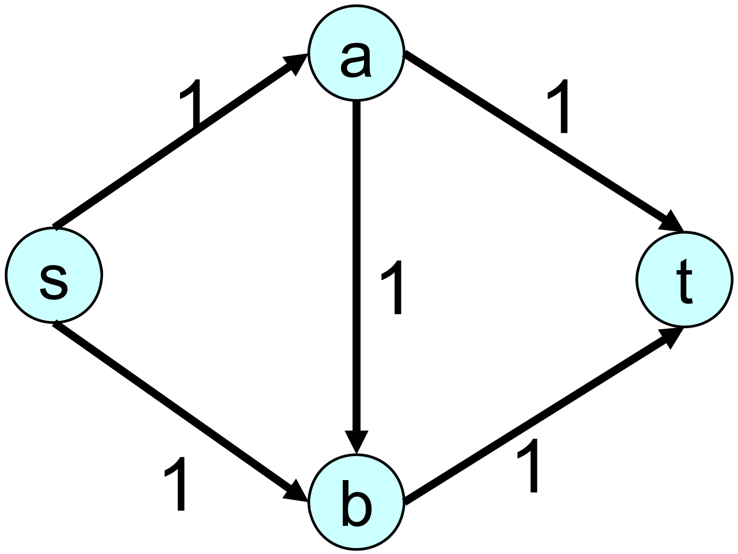
最大流問題の解法

最大流問題は線形計画問題の特殊ケース
⇒　単体法で解くことが可能！

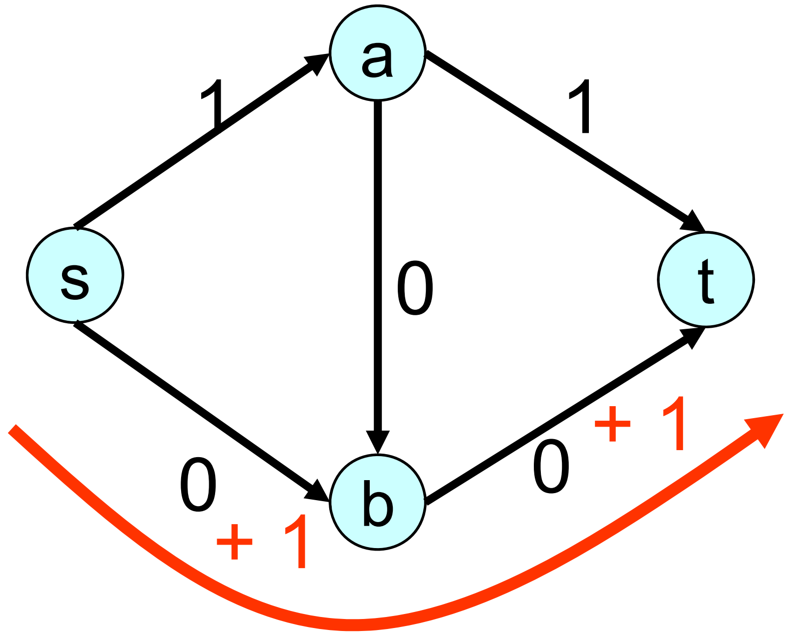
最大流問題は良い（数学的な）構造をもつ
⇒　この問題専用の解法（フロー増加法など）
を使うと，より簡単，かつより高速に解くことが可能

最大流の判定

問題の例



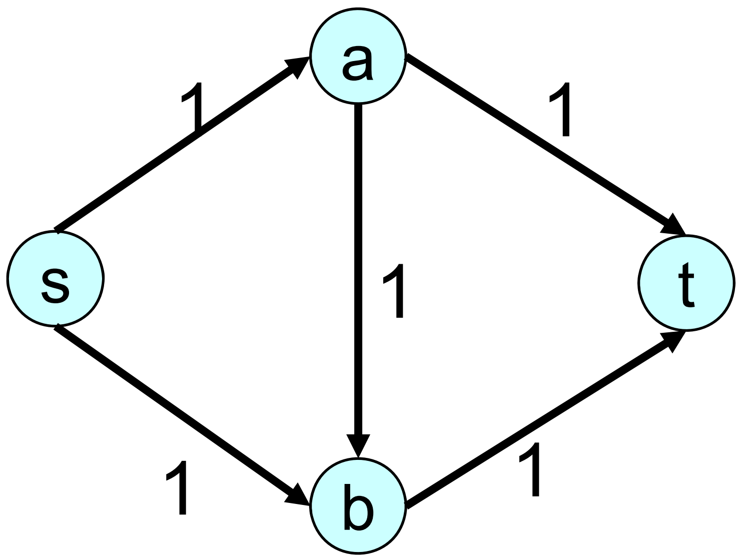
フローの例1：最大？



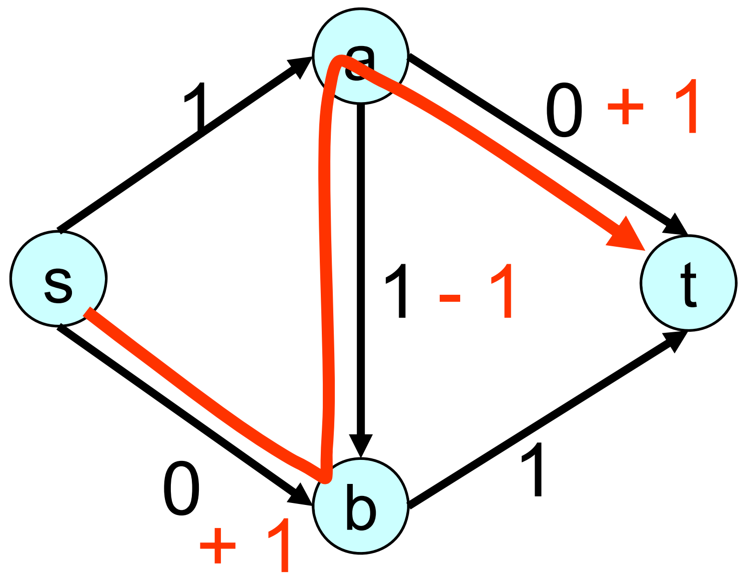
最大流ではない

最大流の判定

問題の例



フローの例2：最大？



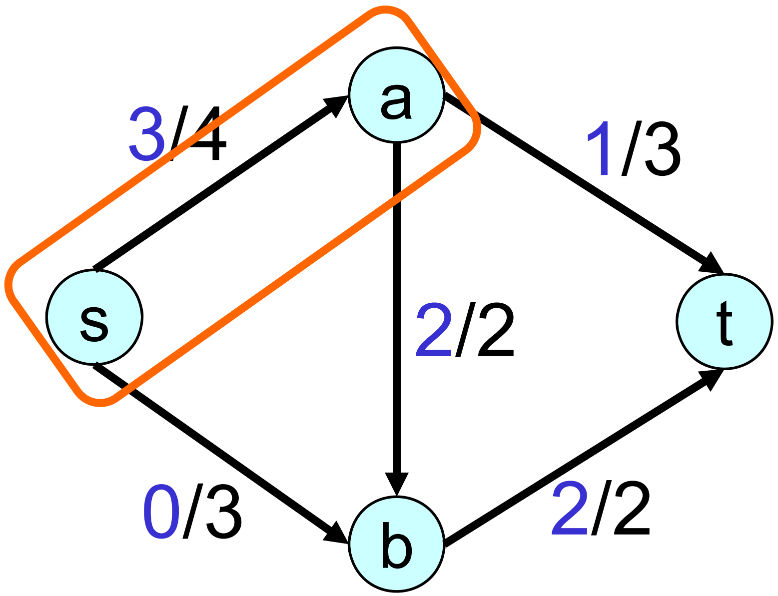
最大流ではない

最大流であることの判定を効率よく行うには？

⇒　残余ネットワーク(residual network)を利用

残余ネットワークの定義

残余ネットワークの作り方



問題例とフロー

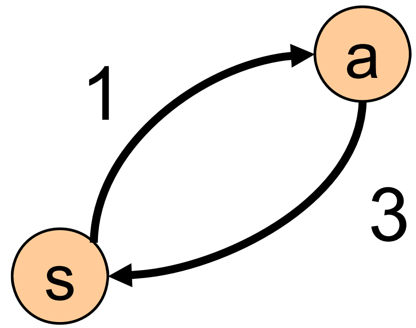
各枝のデータは

（フロー量/容量）

枝(s,a)において

☆さらに4－3＝1だけフロー
を流せる

⇒　残余ネットワークに
容量1の枝(s,a)を加える

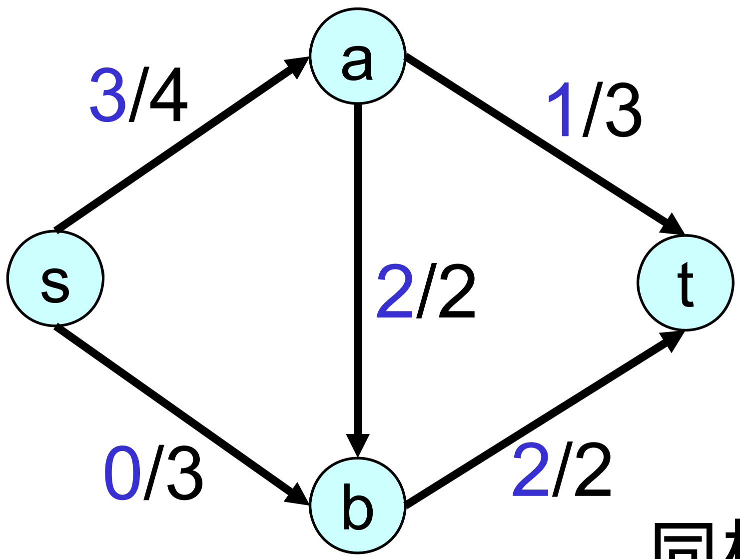


☆現在のフロー3を逆流させて
0にすることが出来る

⇒　容量3の枝(a,s)を加える

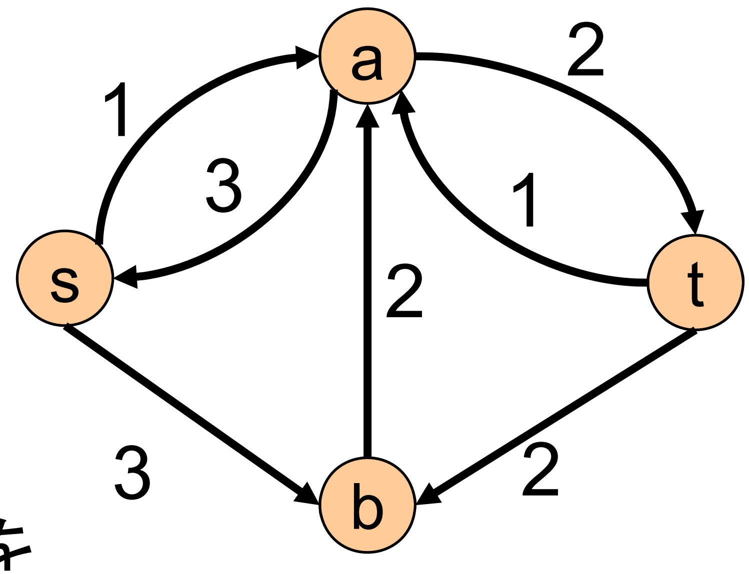
残余ネットワークの定義

残余ネットワークの作り方



問題例とフロー

同様の操作を
各枝に行う



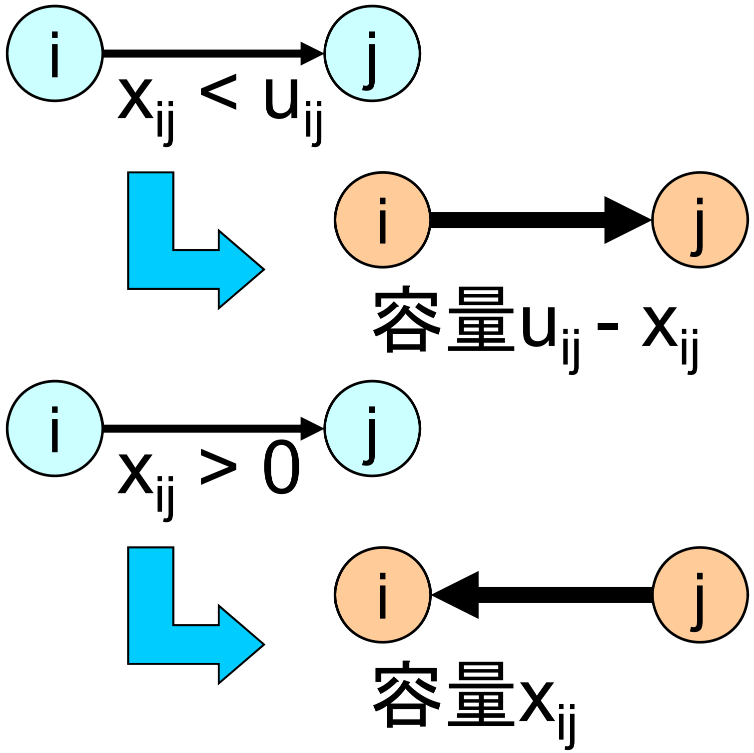
残余ネットワーク
の完成

残余ネットワークの定義(まとめ)

$x = (x_{ij} \mid (i,j) \in E)$: 現在のフロー

フロー x に関する残余ネットワーク $G^x = (V, E^x)$

$E^x = F^x \cup R^x$



順向きの枝集合

$F^x = \{ (i, j) \mid (i, j) \in E, x_{ij} < u_{ij} \}$

各枝の容量 $u^x_{ij} = u_{ij} - x_{ij}$

逆向きの枝集合

$R^x = \{ (j, i) \mid (i, j) \in E, x_{ij} > 0 \}$

各枝の容量 $u^x_{ji} = x_{ij}$

注意！ : 現在のフローが変わると残余ネットワークも変わる

残余ネットワークに関する定理

フロー増加路：残余ネットワークでのソースからシンクへのパス（路）

定理１：残余ネットワークに フロー増加路が存在する
➔ 現在のフローは増加可能

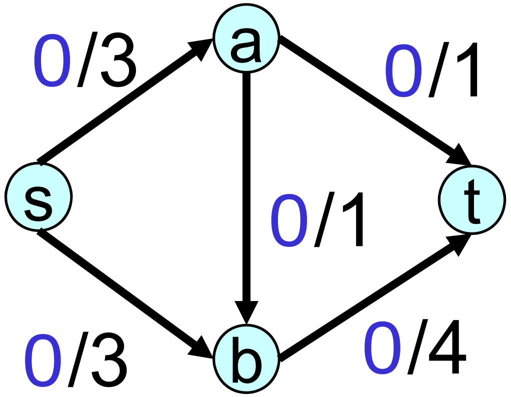
定理２：残余ネットワークに フロー増加路が存在しない
➔ 現在のフローは最大流

定理1の例

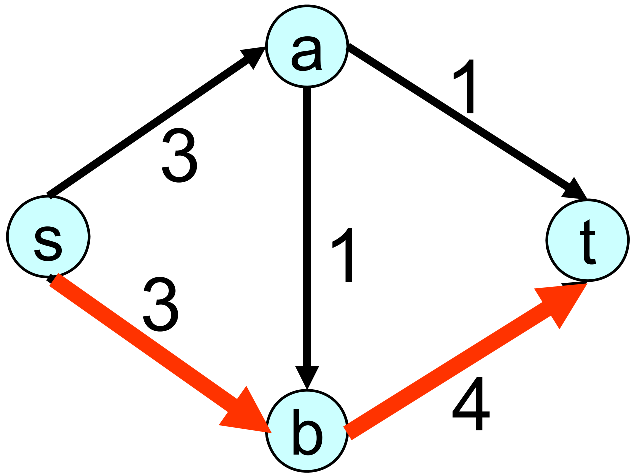
定理１：残余ネットワークに s-t パスが存在する
➔ 現在のフローは増加可能

証明: s-t パスを使うことで，実際にフローを増加させることが出来る

与えられた問題と現在のフローx

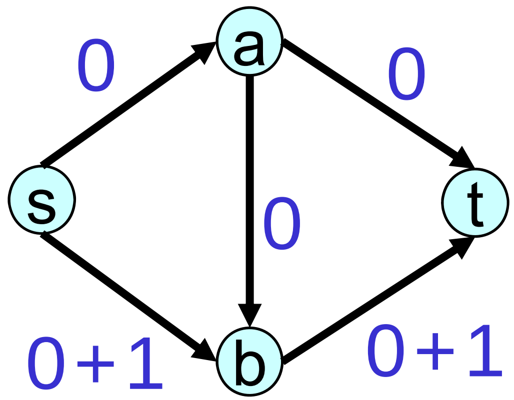


残余ネットワーク



s–t パスが存在

新しいフローx’



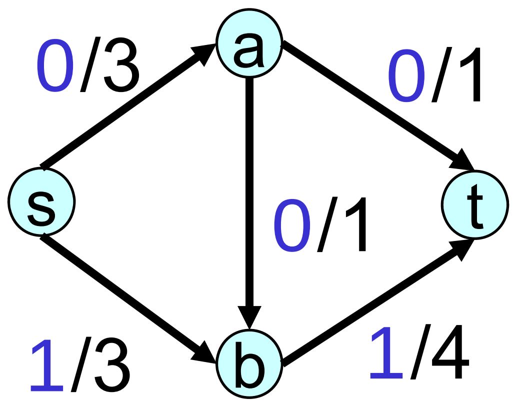
フロー値が
１増えた

定理1の例

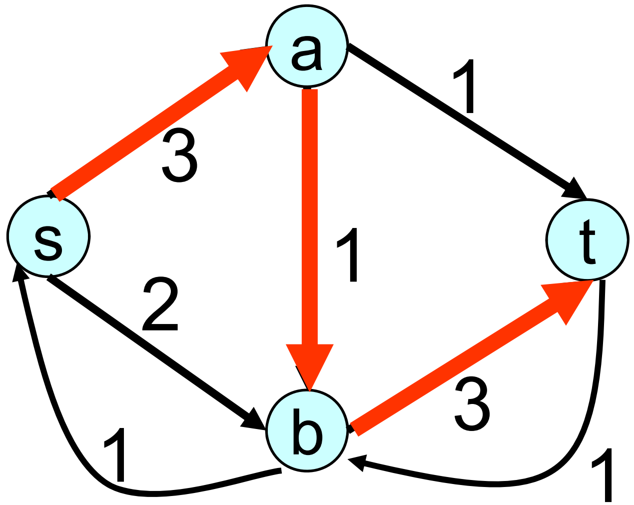
定理１：残余ネットワークに s-t パスが存在する
　　　➔ 現在のフローは増加可能

証明：s-t パスを使うことで，実際にフローを増加させることが出来る

与えられた問題と
現在のフローx

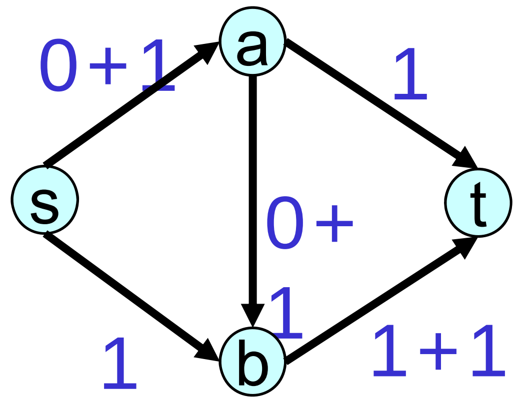


残余ネットワーク



s−t パスが存在

新しいフローx’



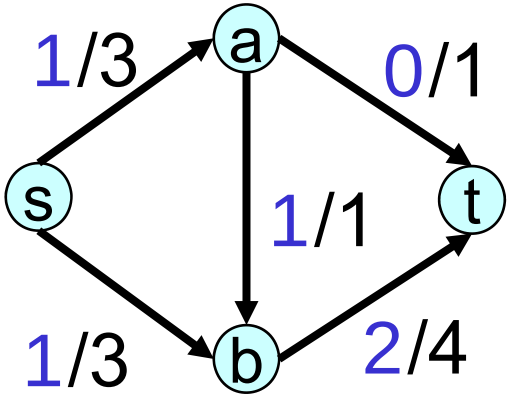
フロー値が
１増えた

定理1の例

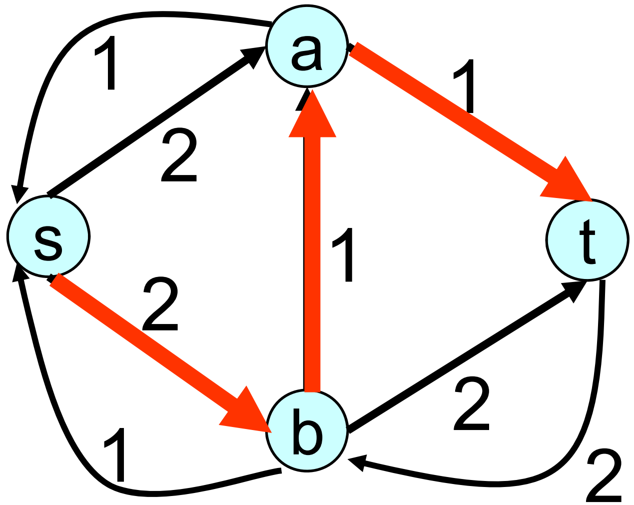
定理１：残余ネットワークに s-t パスが存在する
➔ 現在のフローは増加可能

証明：s-t パスを使うことで，実際にフローを増加させることが出来る

与えられた問題と
現在のフローx

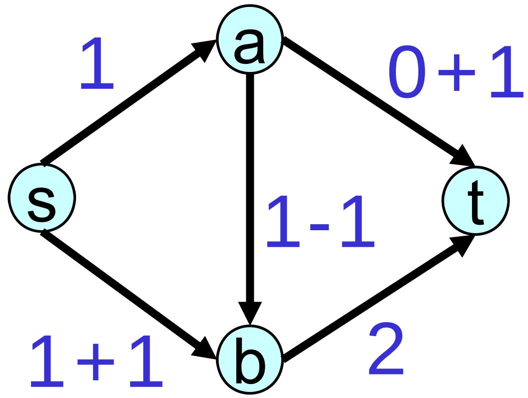


残余ネットワーク



s–t パスが存在

新しいフローx’

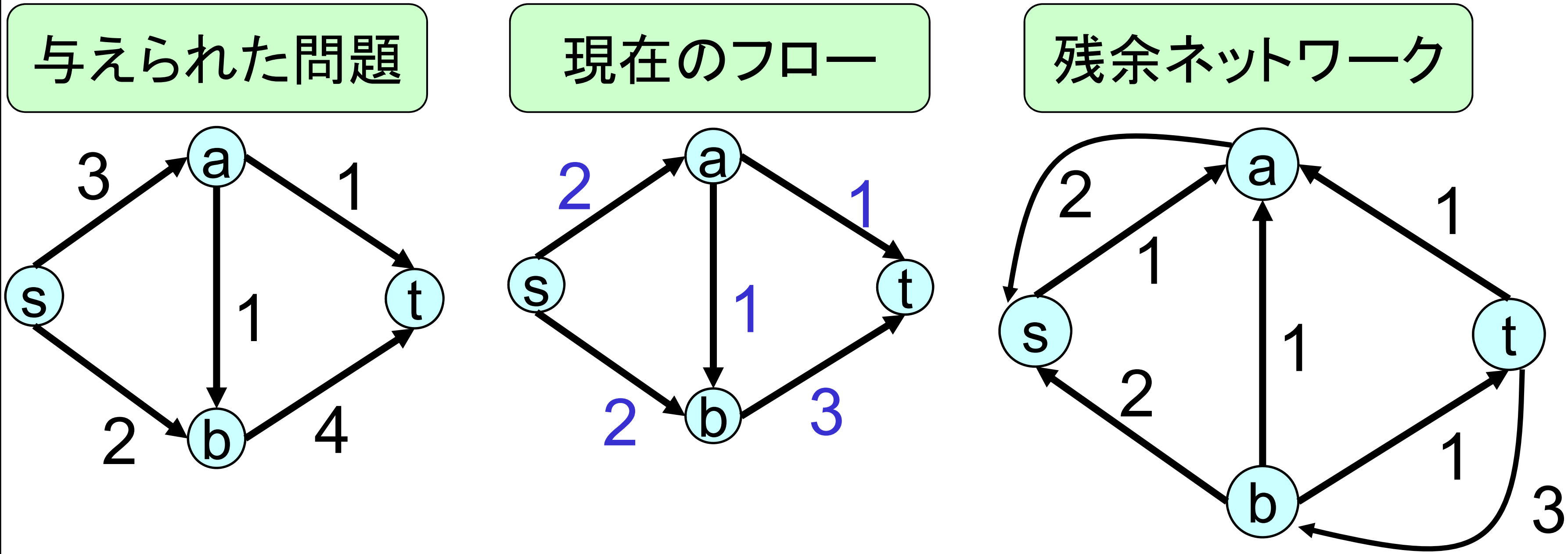


フロー値が
１増えた

定理2の例

定理2：残余ネットワークに s-t パスが存在しない
➔ 現在のフローは最大流

証明は次回



s-t パスがない
➔現在のフローは最適！

フロー増加法 flow augmenting algorithm

最大流を求めるためのアルゴリズム

ステップ0：初期フローとして、全ての枝のフロー量を
0とする

ステップ1：現在のフローに関する残余ネットワークを作る

ステップ2：残余ネットワークに フロー増加路が存在しない
⇒ 終了

ステップ3：残余ネットワークのフロー増加路をひとつ求め、
それを用いて現在のフローを更新する

ステップ4：ステップ1へ戻る

フロー増加法の計算時間

※各枝の容量は整数と仮定
U = 容量の最大値
m = 枝の数，n = 頂点の数

各反復においてフローが1以上増加
➔反復回数 ≦ 最大流量 ≦ m U

各反復での計算時間
＝ 残余ネットワークのフロー増加路を求める時間
➔深さ優先探索，幅優先探索などを使うと O(m + n) 時間

∴ 計算時間は O((m+n) m U)
（入力サイズは m + n + log U なので，指数時間）

フロー増加法の改良

フロー増加法の反復回数を少なくしたい
➔ 各反復でのフロー増加路の選び方を工夫する

（改良法１） 各反復でのフロー増加量を大きくする
➔各反復で容量最大のフロー増加路を選ぶ
➔反復回数 O(m log (n U)), 計算時間 $O(m^2 \log (n U))$

（改良法2） 各反復で最短（枝数最小）のフロー増加路を選ぶ
➔反復回数 O(m n), 計算時間 $O(m^2 n)$

※この他にも，フロー増加法の計算時間を短縮するための様々なテクニックが存在
全く違うアイディアのアルゴリズム➔プリフロープッシュ法（§3.5, 3.6）

レポート問題

問1：次の2つの最大流問題に対する定式化を書きなさい

問2：次の2つの最大流問題に対して、フロー増加法で最大流を求めよ（各反復での残余ネットワークやフローも省略せずに書くこと）

締切：次回講義（12／15）の13：05まで