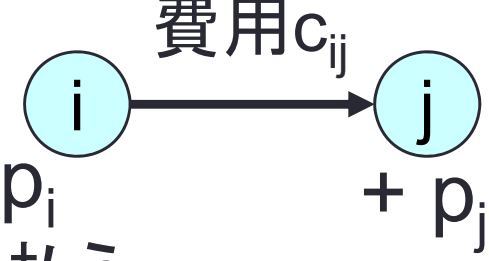
数理経済学
第12回
最小費用流問題
ナップサック問題に対する
近似アルゴリズム

塩浦昭義
東京工業大学 経営工学系 准教授
shioura.a.aa@m.titech.ac.jp

ネットワークのポテンシャル

定義：ポテンシャル $p = (p_i \mid i \in V)$

--- 各頂点に付随する実数変数（からなるベクトル）



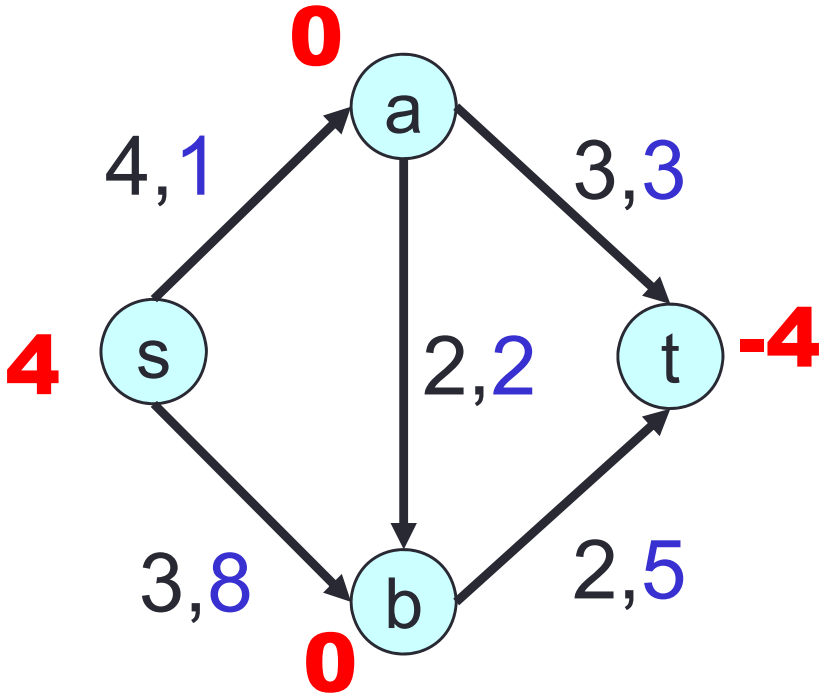
- イメージ：輸送に伴う補助金・手数料
  - 枝(i, j) にフローを1単位 流すとき，
    始点 i では p_i 円もらえる，終点 j では p_j 円支払う
    （始点 i では $-p_i$ 円で財を購入，
    i から j に財を輸送して（コストc_{ij}円）
    終点 j では $-p_j$ 円で財を販売）
  - ➔ 枝 (i, j) でのフローの実質的なコストは $c_{ij} - p_i + p_j$
  - 実質的なコスト > 0 → 枝 (i, j) のフローを減らしたい
  - 実質的なコスト < 0 → 枝 (i, j) のフローを増やしたい

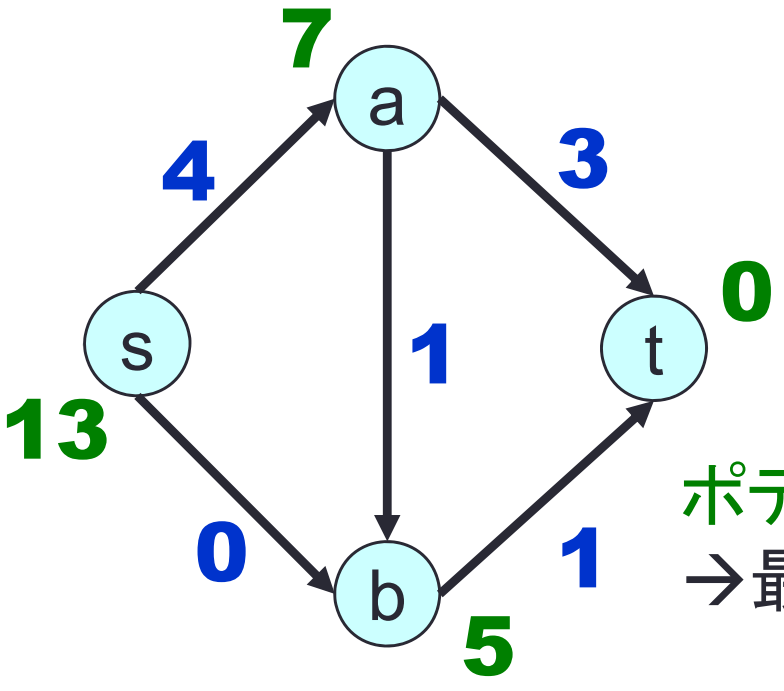
ポテンシャルを用いた最適性条件

定理３：現在のフロー $(x_{ij} \mid (i,j)\in E)$ は**費用最小**
←→ 以下の条件を満たすポテンシャル $p = (p_i \mid i\in V)$ が存在：
(i) $c_{ij} - p_i + p_j > 0$ なる枝 (i, j) に対し，$x_{ij} = 0$
(ii) $c_{ij} - p_i + p_j < 0$ なる枝 (i, j) に対し，$x_{ij} = u_{ij}$
(iii) $0 < x_{ij} < u_{ij}$ なる枝 (i, j) に対し，$c_{ij} - p_i + p_j = 0$

証明は後で

※注意：条件(iii) は (i), (ii) から導かれるので，なくても良い



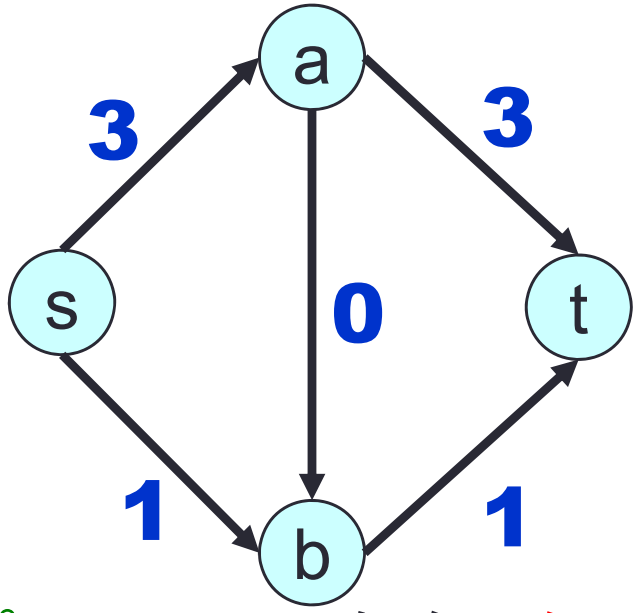


ポテンシャル存在
→最小費用フロー

ポテンシャルを用いた最適性条件

定理３：フロー $(x_{ij} \mid (i,j)\in E)$ は**費用最小**
←→ 以下の条件を満たすポテンシャル $p = (p_i \mid i\in V)$ が存在：
(i) $c_{ij} - p_i + p_j > 0$ なる枝 (i, j) に対し，$x_{ij} = 0$
(ii) $c_{ij} - p_i + p_j < 0$ なる枝 (i, j) に対し，$x_{ij} = u_{ij}$
(iii) $0 < x_{ij} < u_{ij}$ なる枝 (i, j) に対し，$c_{ij} - p_i + p_j = 0$

証明は後で



枝(s,b)に対し，条件(i)（の対偶）より
$8 - p_s + p_b \leq 0 \Rightarrow -8 + p_s - p_b \geq 0$

枝(s,a),(a,b)に対し，条件(ii)（の対偶）より
$1 - p_s + p_a \geq 0, 2 - p_a + p_b \geq 0$

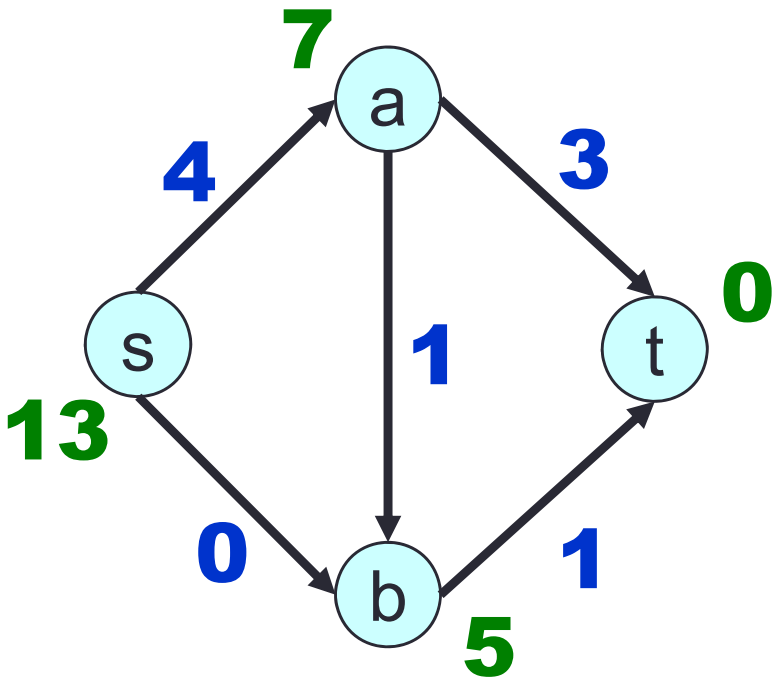
ポテンシャル存在しない
→最小費用フローではない

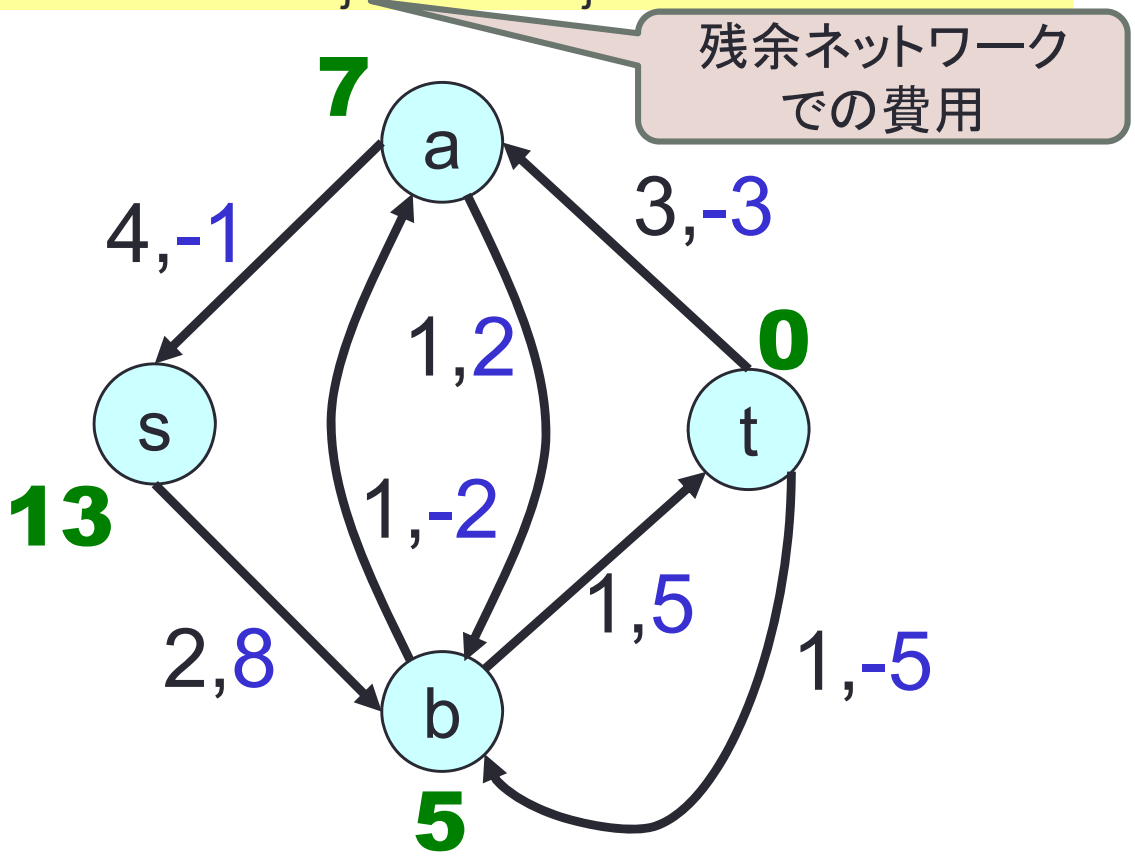
を辺々加えると - 5≧0（矛盾）

ポテンシャルを用いた最適性条件の書き換え

定理3の条件(i), (ii) は，残余ネットワークを使うと簡潔になる

定理４：現在のフロー $(x_{ij} \mid (i,j) \in E)$ は**費用最小**
←→ 以下の条件を満たすポテンシャル $p = (p_i \mid i \in V)$ が存在：
残余ネットワークの各枝 (i, j) に対し，$c_{ij} - p_i + p_j \geqq 0$





定理4の証明

定理3を用いて，定理4を証明する．
定理3の条件(i), (ii) と条件(＊)が等価であることを示せばよい．

(i) $c_{ij} - p_i + p_j > 0$ なる枝 (i, j) に対し，$x_{ij} = 0$
(ii) $c_{ij} - p_i + p_j < 0$ なる枝 (i, j) に対し，$x_{ij} = u_{ij}$

(＊) 残余ネットワークの各枝 (i, j) に対し，$c_{ij} - p_i + p_j \geqq 0$

(＊)において，
残余ネットワークの順向きの枝(i, j) に対し，$c_{ij} - p_i + p_j \geqq 0$
⇔ $x_{ij} < u_{ij}$ ならば$c_{ij} - p_i + p_j \geqq 0$ ⇔ (ii) の対偶

残余ネットワークの逆向きの枝(i, j) に対し，$c_{ij} - p_i + p_j \geqq 0$
⇔ $x_{ji} > 0$ ならば $-c_{ji} - p_i + p_j \geqq 0$
⇔ $x_{ji} > 0$ ならば $c_{ji} - p_j + p_i \leqq 0$ ⇔ (i) の対偶

定理3の「←」の証明（その1）

$d_{ij} = c_{ij} - p_i + p_j$ と定義，これを枝(i, j)の新しい費用と見なす．

示すこと

（1）条件(i), (ii), (iii) を満たすフロー x は d_{ij} に関する総費用が最小

（2）（フロー x の d_{ij} に関する総費用）

＝（x のc_{ij} に関する総費用）＋（x に依存しない）定数

（つまり，フロー x は d_{ij} に関して費用最小←→c_{ij} に関して費用最小）

（1）の証明：

条件(i), (ii), (iii) を満たすフロー x は，枝ごとに見ても

d_{ij} に関する費用が最小

$d_{ij} > 0$ → $x_{ij} = 0$,　$d_{ij} < 0$ → $x_{ij} = u_{ij}$,　$d_{ij} = 0$ → x_{ij} は何でも良い

よって，総費用も最小

定理3の「←」の証明（その2）

（2）（フロー x の d_{ij} に関する総費用）
　　＝（x の c_{ij} に関する総費用）＋（x に依存しない）定数

（2）の証明:

$$\sum_{(i,j)\in E} d_{ij}\, x_{ij} = \sum_{(i,j)\in E} \left(c_{ij} - p_i + p_j\right) x_{ij}$$

$$= \sum_{(i,j)\in E} c_{ij}\, x_{ij} + \sum_{(i,j)\in E} \left(-p_i + p_j\right) x_{ij}$$

$$= \sum_{(i,j)\in E} c_{ij}\, x_{ij}$$

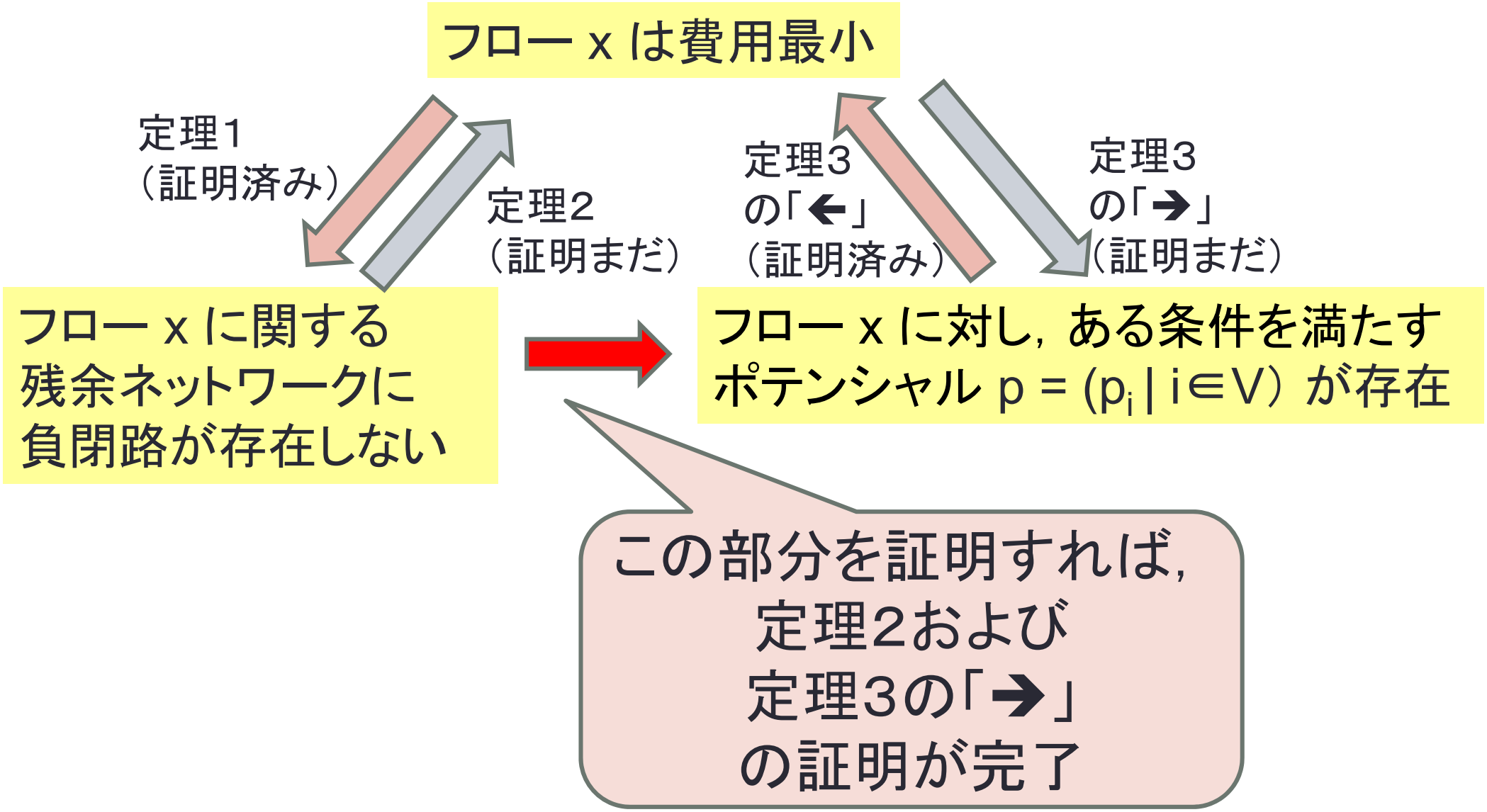
$$- \sum_{k\in V} p_k \left(\sum \{ x_{kj} \,|\, (k,j) \text{は} k \text{から出る枝} \} - \sum \{ x_{ik} \,|\, (i,k) \text{は} k \text{に入る枝} \} \right)$$

$$= \sum_{(i,j)\in E} c_{ij}\, x_{ij} - \sum_{k\in V} p_k\, b_k$$

各頂点ごとに
和をとる

流量保存条件より

最適性条件のまとめ


フロー x は費用最小
定理1
（証明済み）
定理2
（証明まだ）
定理3
の「←」
（証明済み）
定理3
の「→」
（証明まだ）
フロー x に関する
残余ネットワークに
負閉路が存在しない
フロー x に対し，ある条件を満たす
ポテンシャル p = (p_i | i∈V) が存在
この部分を証明すれば，
定理2および
定理3の「→」
の証明が完了

「負閉路なし➔ポテンシャル存在」の証明

命題: フロー x に関する残余ネットワークに負閉路が存在しない
➔ その残余ネットワークの各枝 (i, j) に対し, $c_{ij} - p_i + p_j \geqq 0$
を満たすポテンシャル $p = (p_i \mid i \in V)$ が存在

（証明）
ネットワーク上の最短路問題に関する既知の結果を利用

- 残余ネットワークにおいて,
  費用を枝の長さとみなした最短路問題を考える.
- 任意に選んだ頂点 u から, 各頂点 v への最短路を求める.
  - u から v へのパスが存在しない場合:
    長さが十分に大きい枝 (u, v) を追加
    （これにより, 負閉路は発生しない）
- 負閉路が存在しない➔最短路問題の既知の結果より,
  u から各頂点 v への最短路は必ず存在. その長さを q_v とおく

「負閉路なし➔ポテンシャル存在」の証明

（証明のつづき）

- 負閉路が存在しない➔最短路問題の既知の結果より，

  u から各頂点 v への最短路は必ず存在．その長さを q_v とおく

（注：負閉路が存在する

➔任意に短いパスが存在し，最短路が定まらない）

- 最短路の長さの性質：

  残余ネットワークの各枝 (i, j) に対し，$q_i + c_{ij} \geqq q_j$

（∵ i への最短路経由で j に行くパスの長さ ≧ j への最短路の長さ）

- $p_i = -q_i$ $(i \in V)$ とおくと，上記の不等式より $c_{ij} - p_i + p_j \geqq 0$

ナップサック問題に対する近似アルゴリズム

離散最適化問題

- 離散最適化問題とは:
  - 有限個の「もの」の組合せの中から，目的関数を最小または最大にする組合せを見つける問題
  - 例1:最小全域木，最短路，マッチング（グラフの枝の組合せ）
  - 例2:最小頂点被覆（グラフの頂点の組合せ）
  - 例3:ナップサック問題（「もの」の組合せ）
  - 例4:巡回セールスマン問題（都市の順列，枝の組合せ）

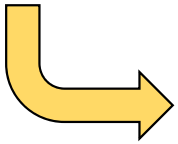
- 解きやすい問題と解きにくい問題
  - 解きやすい問題≒多項式時間で解ける問題
  - 解きにくい問題≒NP困難な問題
    （多項式時間で解けないと信じられている問題）

※離散最適化問題の解は有限個➔有限時間で必ず解ける！

ナップサック問題

ハイキングの準備

- n個の品物の中から持って行くものを選択
- ナップサックには b kg まで入れられる
- 品物 $i = 1,2, \ldots, n$ の重さは a_i kg, 価値は c_i
- 利用価値の合計を最大にしたい



最大化: $\sum_{i \in S} c_i$

制約: $\sum_{i \in S} a_i \leq b$

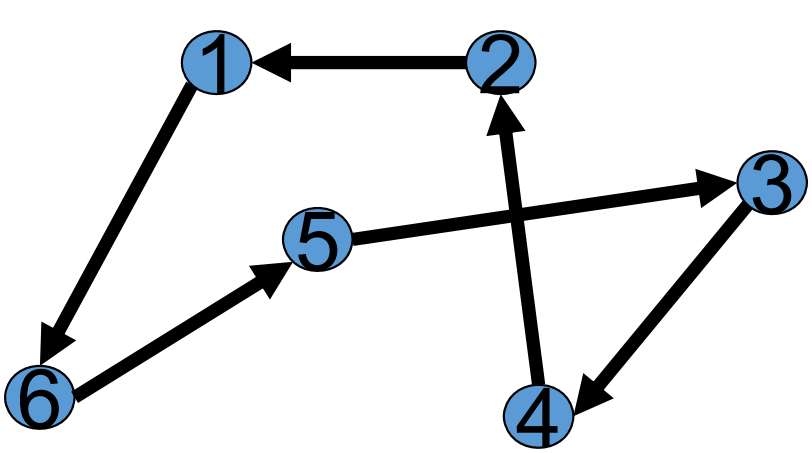
$S \subseteq \{1,2, \ldots, n\}$

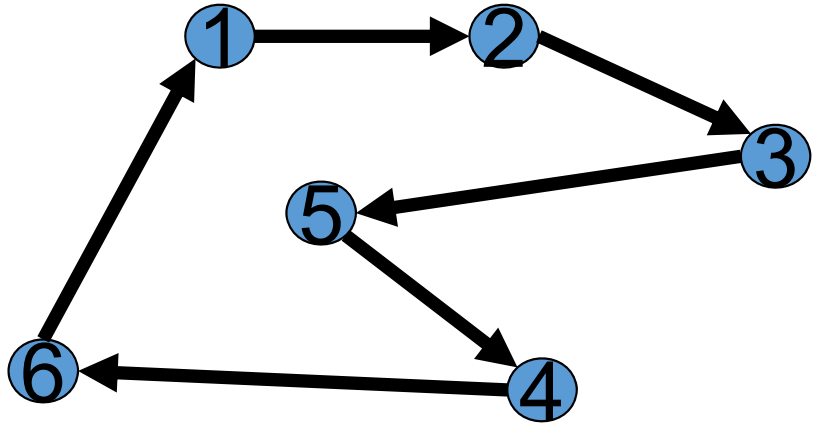
別の例: 財の購入

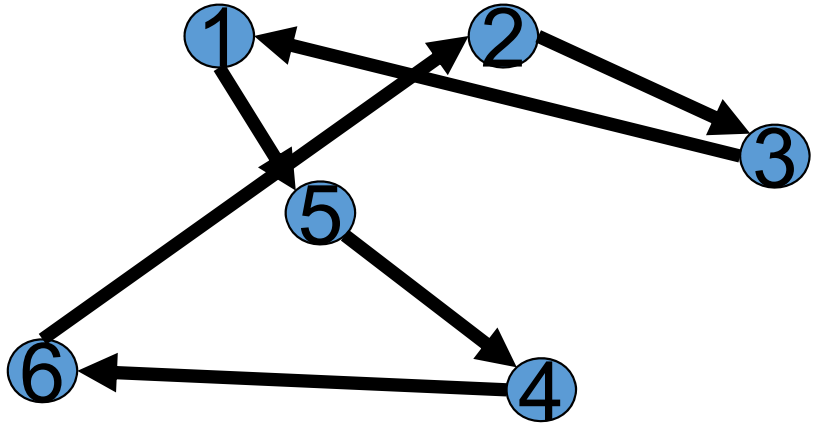
- n個の財の中から購入するものを選択
- 予算は b 円
- 財 $i = 1,2, \ldots, n$ の価格は a_i 円, 満足度は c_i
- 予算範囲内で満足度の合計を最大にしたい

巡回セールスマン問題

- セールスマンが n 都市をちょうど一回ずつ巡回する
- 都市 i から j への距離は c_{ij}（平面上の距離で与えられるケースも）
- 目的：都市を巡回する際の総距離を最小にする







離散最適化問題を解くのは簡単？

- 離散最適化問題の選択肢（解）は有限個 ➔ 有限時間で解ける
- 例1：ナップサック問題
  - 品物が100個 ➔ 組合せは $2^{100} \fallingdotseq 1.27 \times 10^{30}$ 通り
- 例2：巡回セールスマン問題
  - 10か所を訪問する順番： 3,628,800 通り

訪問地点数 n	訪問する順番 $n!$
10	3,628,800
20	2.4×10^{18}
30	2.7×10^{32}

京コンピュータを
使っても
10億年以上必要！

➔ 最も良い選択肢（最適解）を見つけるのは難しい！
- 人間の力だけでは無理
- コンピュータを使っても限界がある

離散最適化問題に対するアプローチ

- 解きやすい問題の場合
  - 多項式時間アルゴリズムを構築

    例：最小全域木 --- 頂点数20, 枝数100のグラフの場合

    全域木の総数は約 1.3×10^{20} 個

    → プリムのアルゴリズムでは，枝数^2 回程度の計算で最小全域木が得られる
  - さらに高速な解法へ
- 解きにくい問題の場合
  - 絶対に最適解が必要➔厳密解法
    - 分枝限定法←現在の主流
    - 動的計画法
  - 高速に解を求めたい，ある程度良い解であれば十分
    - 精度保証付き近似アルゴリズム

      （解の良さに対する理論保証あり）
    - ヒューリスティックス（解の良さは実験的に証明）

今日の授業で紹介するアプローチ

ナップサック問題と連続版

ナップサック問題の定式化

最大化: $\sum_{i \in S} c_i$
制約: $\sum_{i \in S} a_i \leq b$
$S \subseteq \{1,2,\ldots,n\}$

最大化: $\sum_{i=1}^{n} c_i\, x_i$
制約: $\sum_{i=1}^{n} a_i\, x_i \leq b$
$x_1, x_2, \ldots, x_n \in \{0,1\}$

$x_i = 1 \leftrightarrow i \in S$
$x_i = 0 \leftrightarrow i \notin S$

連続ナップサック問題: 変数の条件を$x_i \in \{0,1\}$から$0 \leq x_i \leq 1$へ

最大化: $\sum_{i=1}^{n} c_i\, x_i$
制約: $\sum_{i=1}^{n} a_i\, x_i \leq b$
$0 \leq x_i \leq 1 \ \ (i = 1,2,\ldots,n)$

連続ナップサック問題に対する貪欲アルゴリズム

- 資源（重量，金額）1単位あたりの価値（満足度）の高いものを選ぶ

貪欲アルゴリズム

ステップ1：n 個の品物を $\frac{c_{i_1}}{a_{i_1}} \geq \frac{c_{i_2}}{a_{i_2}} \geq \cdots \geq \frac{c_{i_n}}{a_{i_n}}$ を満たすように並べる

ステップ2：$a_{i_1} + a_{i_2} + \cdots + a_{i_{k-1}} \leq b$

$< a_{i_1} + a_{i_2} + \cdots + a_{i_{k-1}} + a_{i_k}$ を満たす k を求める

ステップ3：$x_{i_1} = x_{i_2} = \cdots = x_{i_{k-1}} = 1,$

$x_{i_k} = \frac{b - \sum_{j=1}^{k-1} a_{i_j}}{a_{i_k}}$, $x_{i_{k+1}} = x_{i_{k+2}} = \cdots = x_{i_n} = 0$ とする.

定理　貪欲アルゴリズムで得られた解は，
連続ナップサック問題の最適解

連続ナップサック問題に対する貪欲アルゴリズム：実行例

i	1	2	3	4	5	6	7	8
c_i	350	400	450	20	70	8	5	5
a_i	25	35	45	5	25	3	2	2

Martello, Toth: Knapsack Problems, Wiley (1990) より

b=104

c_i/a_i	14	11.4	10	8	2.8	2.7	2.5	2.5

25+35≦104<25+35+45

x_i	1	1	44/45	0	0	0	0	0

最適値＝1190

ナップサック問題に対する貪欲アルゴリズム

- 資源（重量，金額）1単位あたりの価値（満足度）の高いものを選ぶ

貪欲アルゴリズム

ステップ1：n 個の品物を $\frac{c_{i_1}}{a_{i_1}} \geq \frac{c_{i_2}}{a_{i_2}} \geq \cdots \geq \frac{c_{i_n}}{a_{i_n}}$ を満たすように並べる．

$S = \emptyset,\ j := 1$ とおく

ステップ2：$\sum_{i \in S} a_i + a_{i_j} \leq b$ ならば S に i_j を追加

ステップ3：$j = n$ ならば終了．

そうでなければ，j:=j+1 としてステップ2へ戻る

得られた解はどのくらい良い解か？

解の精度の評価

- 最適解（最適値）との比較で評価
- 定義： 近似比＝（得られた近似解の目的関数値）／最適値
  - 最大化問題の場合：近似比≦１，＝１ならば最適解
  - 最小化問題の場合：近似比≧１，＝１ならば最適解

- ナップサック問題の場合：近似比が1に近い解が欲しい

ナップサック問題に対する貪欲アルゴリズム：実行例

i	1	2	3	4	5	6	7	8
c_i	350	400	450	20	70	8	5	5
a_i	25	35	45	5	25	3	2	2

Martello, Toth: Knapsack Problems, Wiley (1990) より

b=104

c_i/a_i	14	11.4	10	8	2.8	2.7	2.5	2.5
	○	○	×	○	○	○	○	○
	79	44	44	39	14	11	9	7

x_i	1	1	0	1	1	1	1	1

目的関数値＝858　近似比＝0.9533

x_i^*	1	0	1	1	1	0	1	1

目的関数値＝900

ナップサック問題に対する貪欲アルゴリズム：悪い例

近似比が任意に悪くなる例が存在

i	1	2
c_i	1	b-1
a_i	1	b

bは任意の正整数

貪欲アルゴリズムの解＝(1,0), 目的関数値＝1

最適解＝(0,1), 目的関数値=b-1

貪欲アルゴリズムの解の近似比＝1/(b-1)

b→∞とすると近似比→0

ナップサック問題に対する貪欲アルゴリズムの改良

最初に容量オーバーになる品物（連続ナップサックのときのk）を利用

改良版貪欲アルゴリズム

ステップ1：n 個の品物を $\frac{c_{i_1}}{a_{i_1}} \geq \frac{c_{i_2}}{a_{i_2}} \geq \cdots \geq \frac{c_{i_n}}{a_{i_n}}$ を満たすように並べる．

$S = S' = \emptyset,\ j := 1$ とおく

ステップ2：$\sum_{i \in S} a_i + a_{i_j} \leq b$ ならば S に i_j を追加．

$\sum_{i \in S} a_i + a_{i_j} > b$ かつ $S' = \emptyset$ ならば S' に i_j を追加．

ステップ3：$j = n$ ならば S と S’ の良い方を出力して終了．

そうでなければ，j:=j+1 としてステップ2へ戻る

得られた解はどのくらい良い解か？

悪い例に対する
改良版貪欲アルゴリズムの適用

i	1	2
c_i	1	b-1
a_i	1	b

bは任意の正整数

改良版貪欲アルゴリズムの解　S＝(1,0), S’=(0,1),

よって，最適解が得られる

改良版貪欲アルゴリズムの近似比

定理　ナップサック問題に対し，
　改良版貪欲アルゴリズムで得られた解の近似比は 1/2 以上

（証明）連続ナップサック問題の方が解の選択肢が多いので，

　ナップサック問題の最適値≦連続ナップサック問題の最適値

連続ナップサック問題の最適解：{1,2,…,k-1} は全部，{k} は一部分

改良版貪欲アルゴリズムにおいて {1,2,…,k-1}⊆S,　　k∈S’

➔（Sの目的関数値）＋（S’の目的関数値）

　　≧連続ナップサック問題の最適値

∴改良版貪欲アルゴリズムの解

≧(1/2){（Sの目的関数値）＋（S’の目的関数値）}

≧(1/2)連続ナップサック問題の最適値

≧(1/2)ナップサック問題の最適値