https://shorturl.at/K7YKT

KYOTO UNIVERSITY

アルゴリズムとデータ構造①
～ 概要 ～

鹿島久嗣
（情報学科 計算機科学コース）

DEPARTMENT OF INTELLIGENCE SCIENCE AND TECHNOLOGY

講義についての情報：
最新の情報はPandAで確認すること

- 担当教員： 鹿島久嗣
  （工学部情報学科計算機科学コース）
  - 連絡先： kashima@i.kyoto-u.ac.jp
- PandAのページ：
  https://panda.ecs.kyoto-u.ac.jp/x/JNHv4q
- 常に最新のスケジュール等をPandA で確認すること
- 評価方法：中間試験＋期末試験
  - 平常点（クイズ）は補助的に利用

参考書:
標準的なものであればなんでもよいが...

- 基本：

  杉原厚吉「データ構造とアルゴリズム」（共立出版）

  –本講義の多くの内容はこの本に依る

  –とても読みやすい

- より高度な内容：
  Cormen, Leiserson, Rivest, & Stein
  「Introduction to Algorithms」

  –翻訳：「アルゴリズムイントロダクション」（近代科学社）

  –講義内容は部分的に参照

内容（前半）：
アルゴリズムの基本的な概念、評価法、基本的な道具

1. 算法とは・算法の良さの測り方：
   アルゴリズムとデータ構造、計算のモデル、計算複雑度、…
2. 基本算法：
   挿入、削除、整列、検索、…
3. 基本データ構造：
   リスト、スタック、キュー、ヒープ、…
4. 算法の基本設計法：
   再帰、分割統治、動的計画、…
5. 探索：
   二分探索、ハッシュ、…

内容（後半）：
グラフ・計算量・難しい問題への対処法

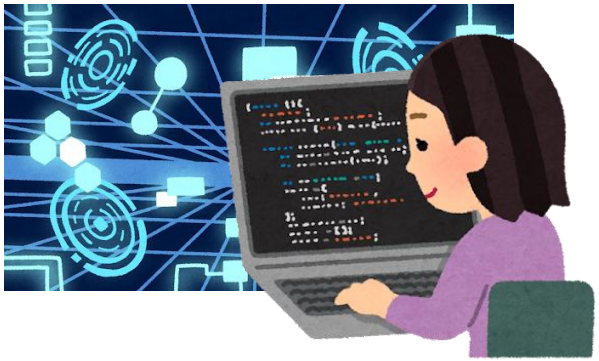
6. グラフ算法：
   深さ・幅優先探索、最短路、最大流
7. 計算複雑度：
   PとNP、NP完全、NP困難
8. 難しい問題の解き方：
   分枝限定法、貪欲法
9. 発展的話題：
   近似アルゴリズム、オンラインアルゴリズム

※ 変更・追加の可能性あり

アルゴリズムとデータ構造は

動機:
「良い」プログラムを書きたい

- プログラムの良し悪しとは：
  - 正しく動く： 想定したように動く
  - 速く動く：プログラムは速いほど良い！
  - 省資源：メモリや電気代
  - 例：お店の顧客管理



- 特定のプログラム言語やハードウェアとはなるべく独立に：
  - プログラムの良し悪しを測りたい
  - ひいては良いプログラムを書きたい

アルゴリズム:
与えられた問題を解くための有限の手続き

- アルゴリズム（algorithm）とは：
  - プログラム言語（CやPythonなど）やハードウェア（CPUやメモリなど）とは別に、どのような手続きを表現しようとするかという「問題の解き方の手順書」
  - もうすこし厳密にいうと「与えられた問題を解くための機械的操作からなる、有限の手続き」
    - 機械的操作：四則演算やジャンプなど
    - かならず有限ステップで終わるべし

      ↕

    - 手続き（procedure）：有限ステップでの終了が保証されない

データ構造:
データを管理し、アルゴリズムを効率化する

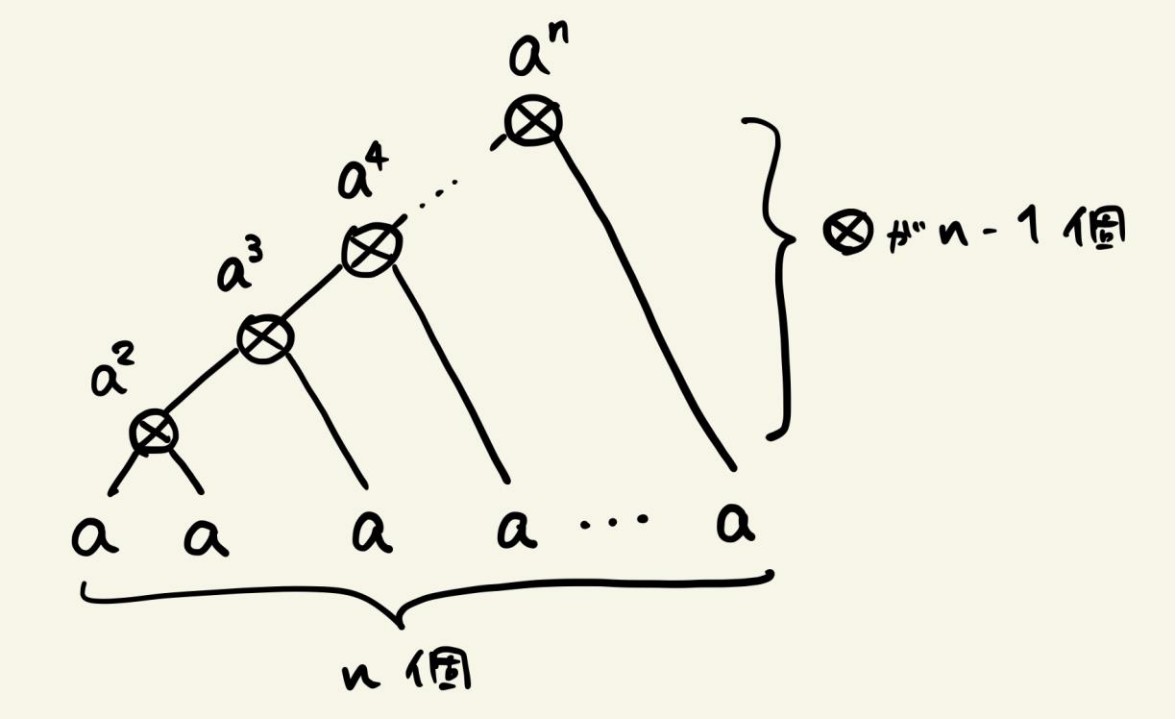
- 多くのプログラムは「データ」を扱う
  - データは繰り返し使用するもの
  - 使用の仕方が予め決められているわけではない
- アルゴリズムがうまく動くためには、データをどのようにもっておくか（＝データ構造）が重要
  - 名前を、入力順に格納？ アイウエオ順？
- データ構造はアルゴリズムと切り離せないもの
  - お互いの良さに影響を与え合う

アルゴリズムの例:
指数演算のアルゴリズム

- 問題：べき乗計算
  - 入力： 2つの正整数 a と n
  - 出力：a^n
  - 仮定：許されるのは四則演算のみとする
    （いきなりn乗するのはダメとする）
- a^nの計算には四則演算が何回必要か？

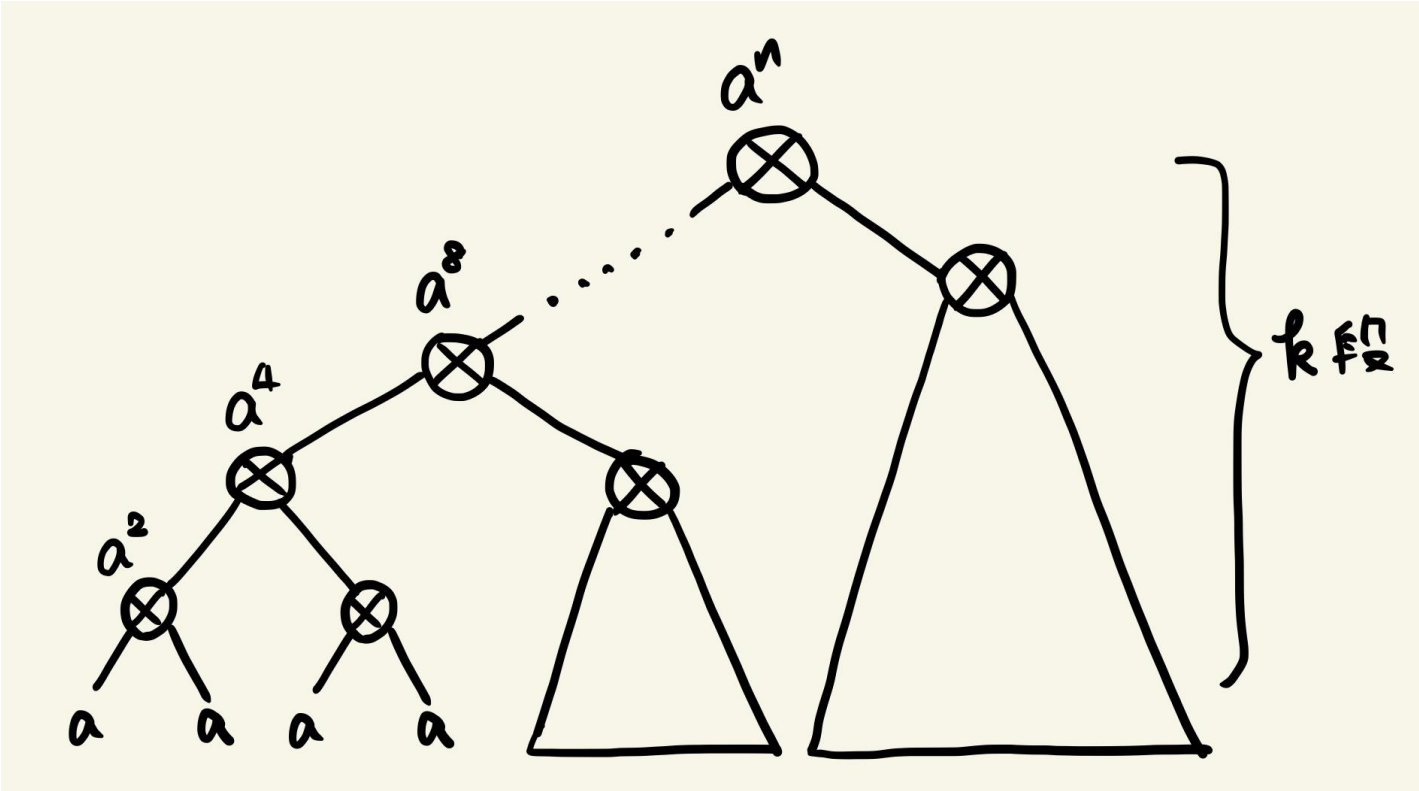
指数演算のアルゴリズム①:
単純な掛け算の繰り返しだと**線形時間**がかかる

- $a^n = ((\ldots(((a \times a) \times a) \times \cdots) \times a)$ で計算
- $n - 1$回の掛け算でできる（演算回数が n の線形関数）



指数演算のアルゴリズム②:
ちょっと工夫すると**対数時間**で計算可能

- 少し工夫すると$k = \log_2 n$ 回の掛け算でできる
  - 仮定： $n = 2^k$ （この仮定はあとで取り除ける）



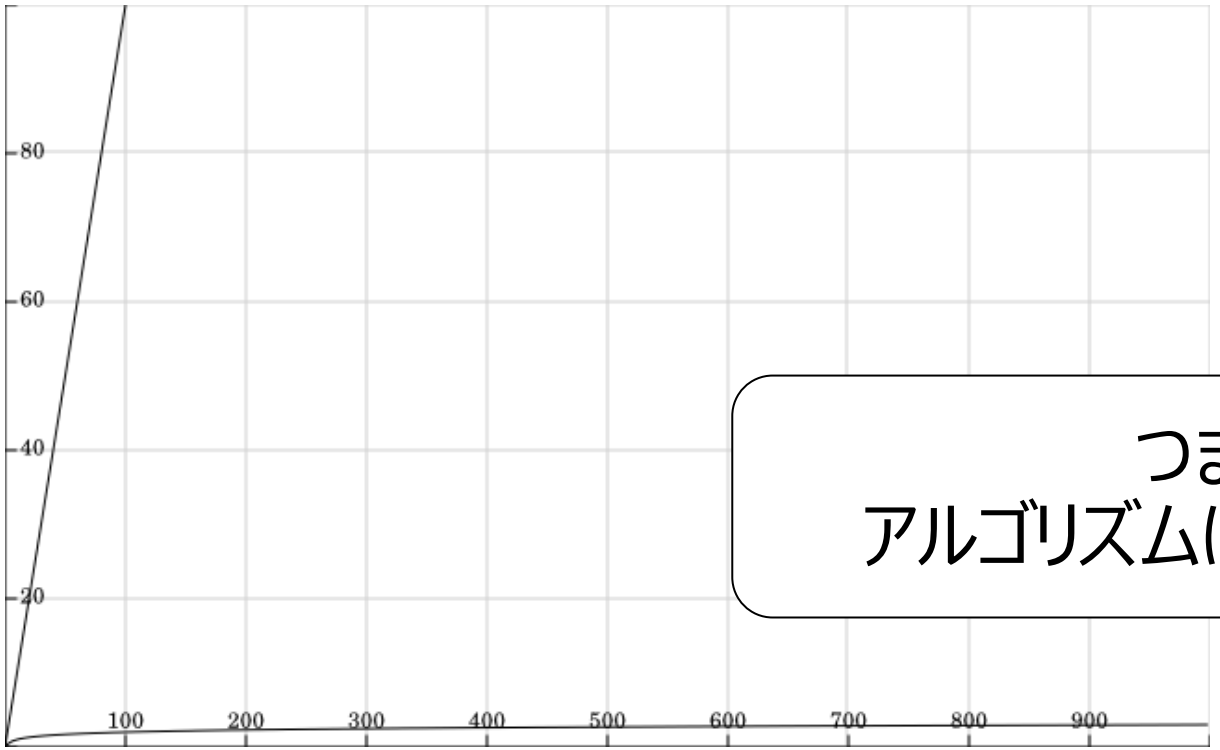
指数演算のアルゴリズム②（補足）：ちょっと工夫すると**対数時間**で計算可能

- なお、$n \neq 2^k$の場合も凡そ $3\log_2 n$ 回の演算で可能
  - nを2進表現する（$\lceil \log_2 n \rceil - 1 = 5$ 回の割り算）
    - 例： $n = 22 = 10110$
  - 1が立っている桁数に対し2の冪を求める（$\lceil \log_2 n \rceil - 1$ 回の掛け算）
    - 例： $n = \cancel{2^0} + 2^1 + 2^2 \cancel{+ 2^3} + 2^4$
  - すべて足す（$\lceil \log_2 n \rceil - 1$回の足し算）

アルゴリズムの重要性:
アルゴリズムの工夫で計算効率に大きな差が生じる

- $n = 1024 = 2^{10}$のとき、掛け算の回数は
  - 解法① では 1023回　（大体 n 回）
  - 解法② では　　10回　（大体 $\log n$回）

} 指数的な差



つまり
アルゴリズムはすごく重要

データ構造の例:
データに対して繰り返し操作を行う場合に有効

- 前のアルゴリズムの例では1回限りの計算を対象としていた
- データに対して繰り返し計算を行う場合には、予めデータを処理して、うまい構造（＝データ構造）を作っておくことでその後の計算を高速に行えるようになる（ことがある）
- 例えば、これから S 回の計算を行うとして

  ① (1回限りの計算時間) × S 回

  ② (データ構造の構築にかかる計算時間) +
  (データ構造を利用した1回分の計算時間) × S

  で① ＞ ② となる場合にはデータ構造を考えることが有効

具体的な問題例:
店舗における顧客情報管理システム

- n 人の顧客情報 $\{(n_i, p_i)\}_{i=1,\ldots,n}$ が載った名簿を考える
  - n_i ：名前、p_i ：情報
  - 例： (元田中 将大, mmototanaka@kyoto-u.ac.jp)
- 客が来るたびに名前を聞いて入力すると、その人の情報が得られるシステムを考える
  - S 人分の問い合わせ $n_{k_1}, n_{k_2}, \ldots, n_{k_S}$ が順に与えられる
  - それぞれに対して p_{k_j} を返す

単純なアルゴリズム：並び順がでたらめな場合は最悪で約nS 回のチェックが必要

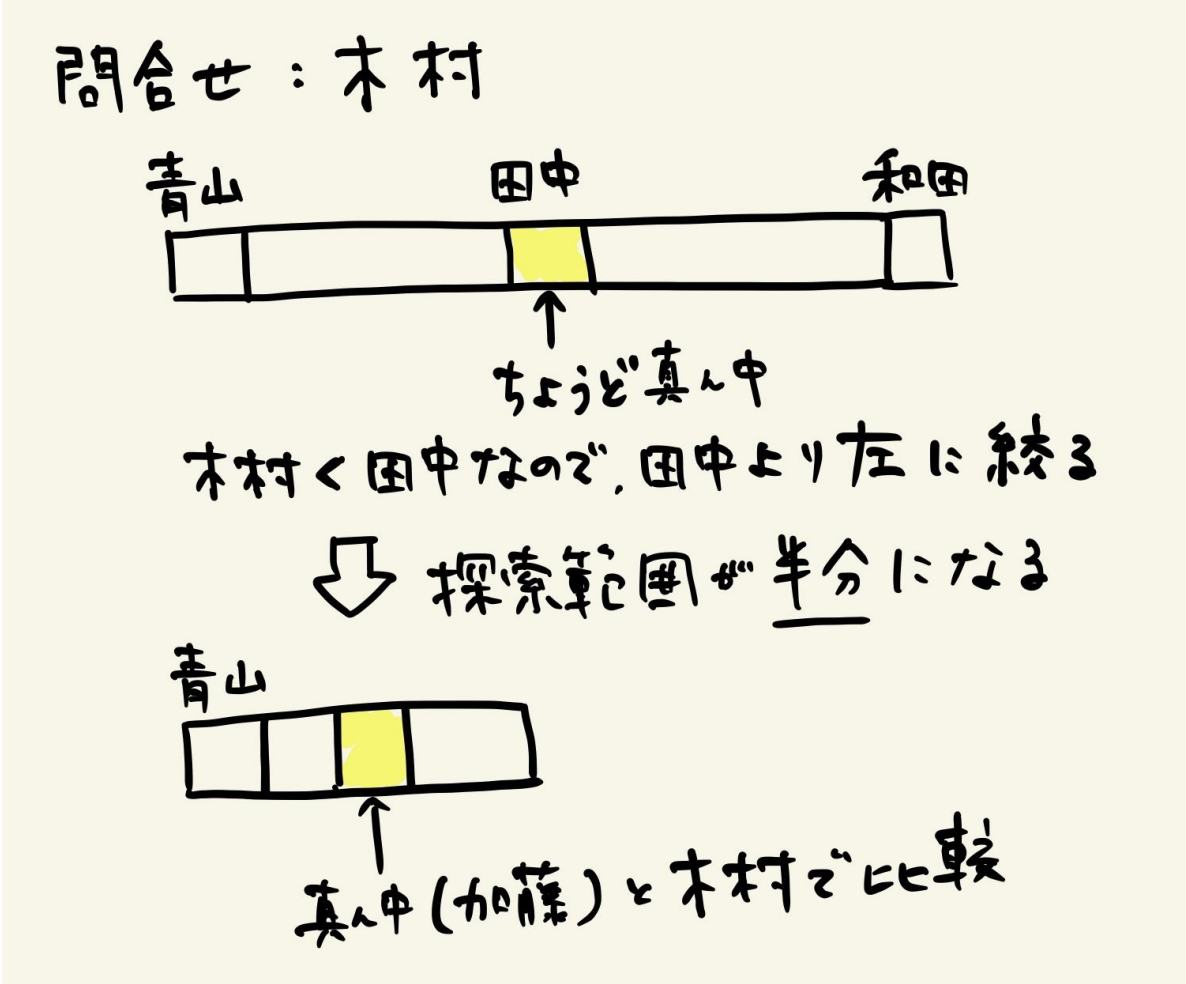
- 名簿の並び順が登録順（つまり、特に規則性なく並んでいる）の場合を考える
- 探索のアルゴリズムとしては、前から順に探していく（しかない）とする
- この場合、各問い合わせで、最悪n回のチェックが必要
  - 名簿上の位置（ページ）を指定してチェックすることは単位時間でできるものとする
- 合計 約 nS 回のチェックが（最悪ケースで）必要となる

ちょっと頭を使ったアルゴリズム：辞書順に並んでいる場合は$S \log n$回のチェックで可能

- 予め名簿を辞書順に並べてデータを保存しておくとする
  - そのような「データの持ち方」をする＝「データ構造」を作る
- 問い合わせ名を名簿の「ほぼ真ん中」の人の名前と比較
  - 前者が辞書順で前ならば、目的の人は名簿の前半分にいるはず
  - 今後は前半分だけを調べればよい
  - こんどは前半分の「ほぼ真ん中」の人と比較
  - ・・・
  - 計$S \log n$回のチェックで可能

ちょっと頭を使ったアルゴリズム：辞書順に並んでいる場合は$S \log n$回のチェックで可能

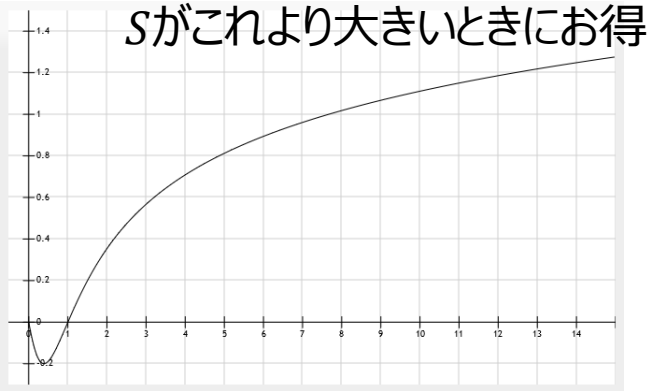
- 計$S \log n$回のチェックで発見可能



データ構造の重要性:
データを正しく持つことで計算コストが大きく削減される

- データ構造＝「データをどのように管理するか」
- 先ほどの「賢いほうのアルゴリズム」の恩恵にあずかるにはデータが予め整列（ソート）されている必要がある
  - 一般に整列は $n \log n$ 回 に比例する演算回数が必要
- よって ① $n\,S$ と ② $n \log n + S \log n$ の比較
  - S が大きくなると②の方がお得になってくる

$\frac{n \log n}{n - \log n}$



つまり
データ構造もすごく重要

ここまでのまとめ:
アルゴリズムとデータ構造を学ぶ意義

- アルゴリズムはソフト／ハードに（あまり）依存しない、計算機による問題解決の手順書
- データ構造は、データの管理を行うアルゴリズム
- いずれも賢く設計すると、すごく得する
  （逆に、賢くやらないと、すごく損する）

アルゴリズムの評価基準

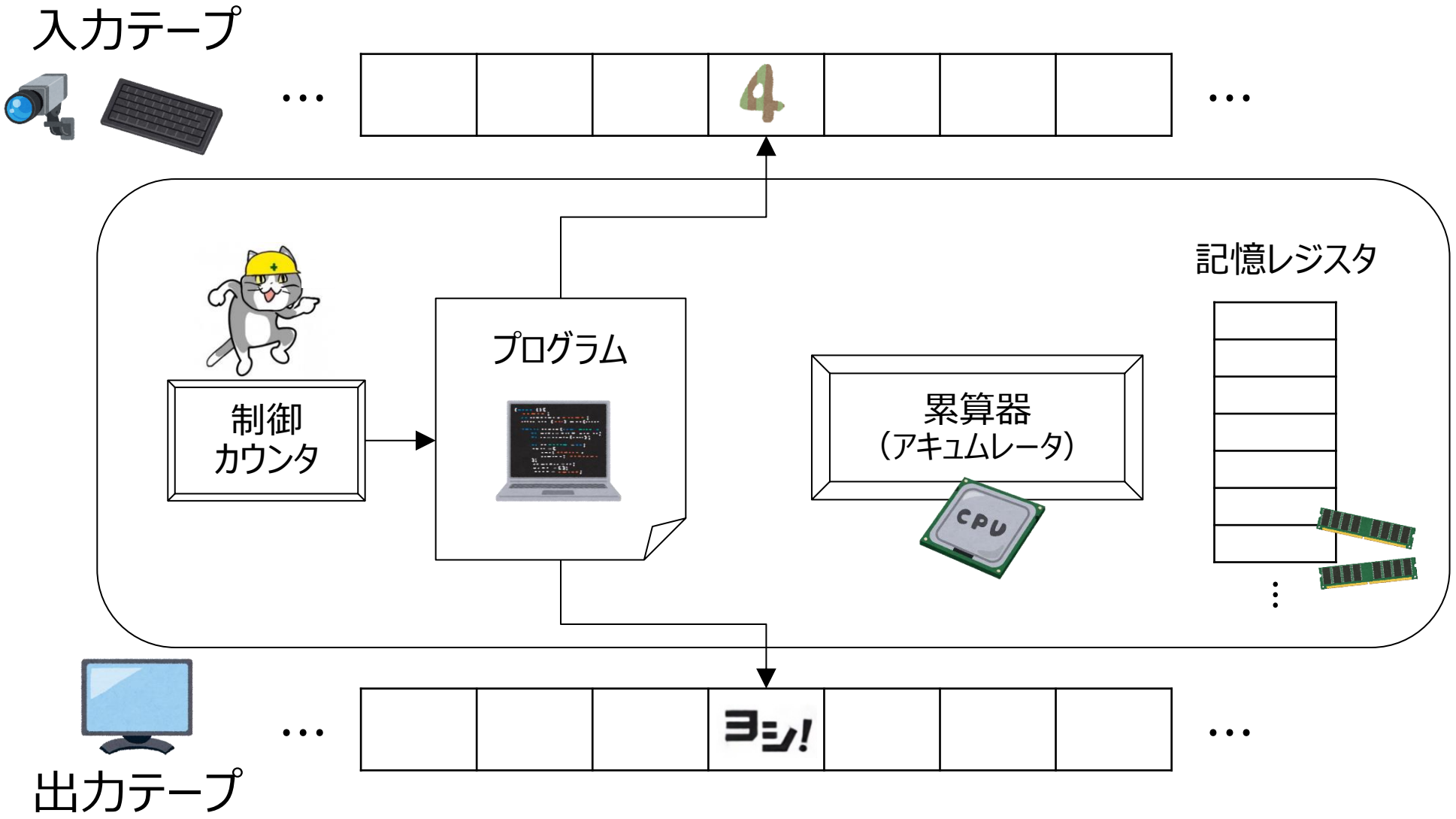
計算機の理論モデル:
計算の効率性を評価するための抽象的な計算機

- 我々は特定の言語やハードウェアに依存しない議論がしたい（＝時代が変わっても有効な議論がしたい）
- 計算のステップ数を数えるためには計算機のモデルを決めておく必要がある
- 計算機の理論モデル：
  - （1930s）Turing機械、λ計算、Postシステム、帰納的関数、ランダムアクセス機械（RAM）など様々なモデルが提案される
  - これらによって計算モデルと計算可能性の概念が議論
  - 上記のモデルはすべて同等 → 安定な概念

ランダムアクセス機械（RAM）：現在のコンピュータに近い、計算機の理論モデル

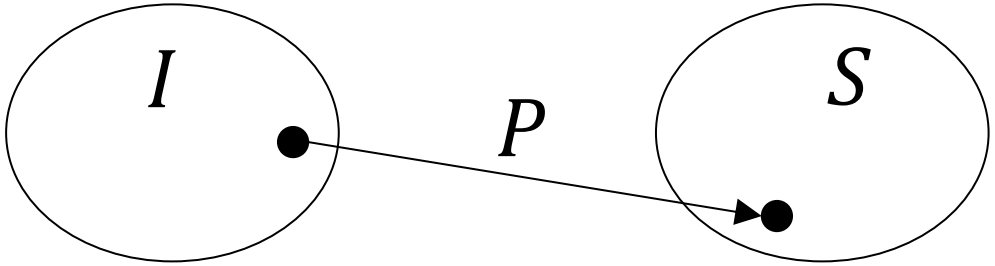
- ランダムアクセス機械：理想化された計算機
  - 入力テープ、出力テープ（モニタ出力）、制御カウンタ、プログラム、累算器（アキュムレータ；一時的な記憶）、記憶レジスタ（記憶装置）をもつ
  - 入・出力テープは無限の長さがあると仮定
  - 累算器、レジスタには任意の桁数のデータが入り、単位時間でアクセス可能と仮定
- 命令：それぞれ単位時間で実行できると仮定
  - 読み取り／書き込み（入力・出力テープを1つ進める）
  - 四則演算、ジャンプ、条件分岐、停止

ランダムアクセス機械:
現在のコンピュータに近い、計算機の理論モデル



「問題」の定義：入力（問題）と出力（解）の対応関係

- 問題：問題例集合Iと解集合Sの対応関係$P: I \to S$



– 例：素数判定問題
  - 問題例：正整数 n （問題例集合 I は正整数全体）
  - 解集合 S ： n が素数なら Yes, そうでないなら No
    （$S = \{\text{Yes}, \text{No}\}$）
    – 決定問題 ⇔ 探索問題（例：代数方程式）
  - 関係 P ：n が決まれば Yes か No かは決まっている

「アルゴリズム」の定義：「問題」を解くための有限の手続き

- 計算モデルと問題に対して定義される
- 定義：アルゴリズム
  - 問題Aに対する計算モデルCでのアルゴリズムとは、有限長の（Cで許された）機械的命令の系列であり、Aのどんな入力に対しても有限ステップで正しい出力をするもの
    - 注意：問題Aの一部の問題例が解けるとかではダメ

計算可能性:
世の中にはアルゴリズムが存在しない問題が存在する

- 問題Aが計算モデルCで計算可能：
  Aに対するCのアルゴリズムが存在すること
- 計算可能な問題があるなら、計算不可能な問題もある
- 計算不可能な問題の一例：プログラムの停止問題
  - 入力：プログラム P、データ x
  - 出力：計算(P, x)が有限ステップで終了するならYes
    そうでなければNo
  - データxはプログラムでもよいことを考えると、この計算不可能性はプログラムの自動的なデバッグは難しいことを示唆している(?)

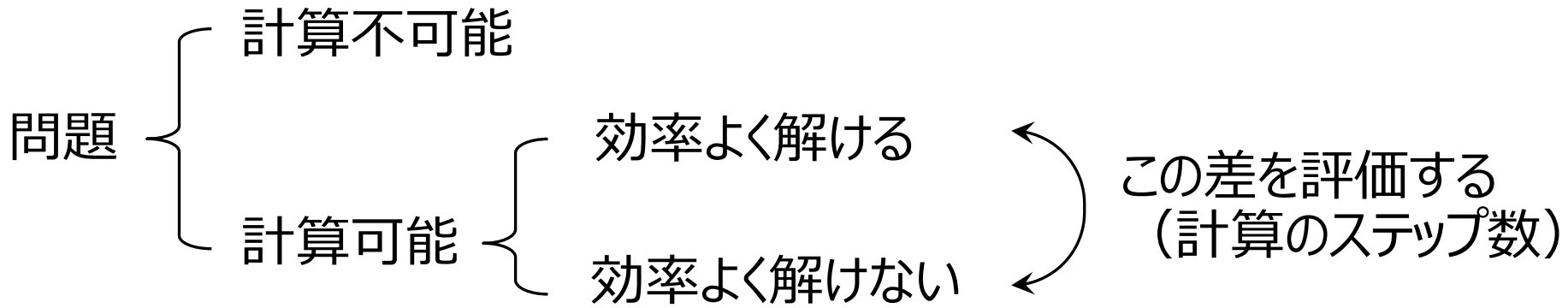
計算可能性:
「プログラムの停止問題」は計算不可能な問題

- 証明の概略：この問題を解けるプログラムQが存在するとして矛盾を示す

1. Qを使って新しいプログラムQ′をつくる：
   - Q′はプログラムPを入力として、(P, P)が停止するなら停止しない、停止しないなら停止するようなプログラム
2. Q′にQ′を入力して矛盾を示す（$Q'(Q') = (Q', Q')$を考える）
   - もしも(Q', Q')が停止するならば
     $\Rightarrow Q'(Q') = (Q', Q')$は停止しない（矛盾）
   - プログラムQが存在するという仮定がおかしい

アルゴリズムの性能評価:
通常は最悪な入力に対するアルゴリズムの計算量を考える

- 計算可能な問題に対するアルゴリズムの評価



- 計算量 (computational complexity)
  - 時間量 (time complexity)：計算時間の評価
  - 空間量 (space complexity)：使用メモリ量の評価
- 最悪計算量： サイズnの全ての問題の入力の中で最悪のものに対する計算量 （⇔ 平均計算量）

最悪計算量と平均計算量：探索問題の場合の例

- （前述の）名簿から名前を見つける探索問題
  - 入力： $n+1$個の正整数 $a_1, a_2, \ldots, a_n$ と k
  - 出力： $a_i = k$となるiが存在すればi；なければ No
- 前から順に探すアルゴリズムを考える
  - 最悪ケースではn回の比較が必要
  - 平均ケースでは約 $\frac{n}{2}$ 回

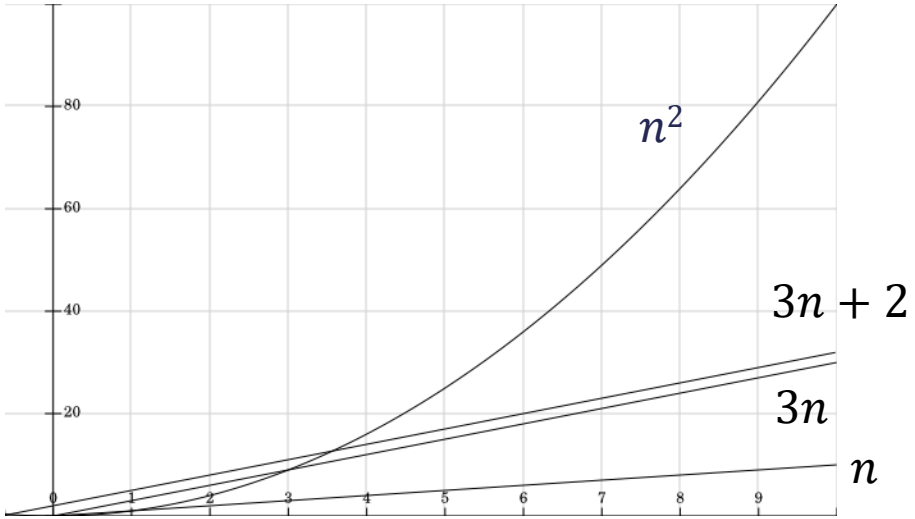
    ※ $\{a_i\}_{i=1,\ldots,n}$の要素がすべて異なり、
    $n+1$通りの場合が等しい確率で起こると仮定する
  - これらは異なるといってよいだろうか？

オーダー評価：アルゴリズムの計算量は最悪ケースのオーダーで評価する

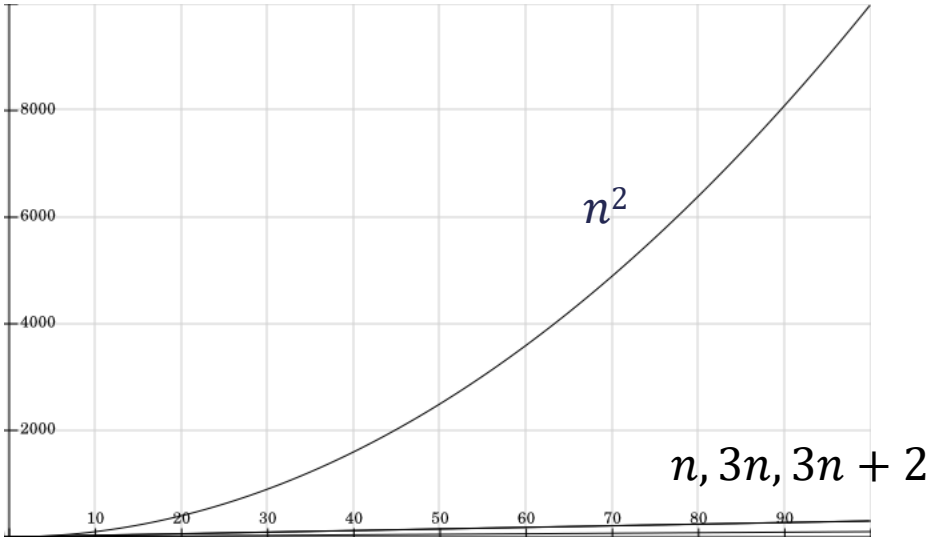
- 以降、特になにも言わない場合は最悪ケースで考える
- （前述の）名簿を前から順に探すアルゴリズムでは、最悪ケースにおいて、n回のチェックが必要
  - もし、n回の比較、n回のカウンタ移動、n回のデータ読み取りと数えるならば、合計で $3n$ 回の操作が必要
  - さらに、出力と停止を加えるなら 計 $3n + 2$ ステップに？
  - 本質的には n が重要 → オーダー記法で $O(n)$ と書く
- オーダー記法： nが大きいときの振舞いを評価する
  - nが大きくなったとき、$n, 3n, 3n + 2$ の違いは、n^2 に比べると極めて小さい

オーダー記法の性質：問題サイズ n の指数部分が支配的になる

- $n, 3n, 3n+2$ と n^2 の比較
- nが大きくなると、n^1とn^2の差しか意味をもたない
- 「遠くから見ると」定数係数にほとんど意味がなくなる



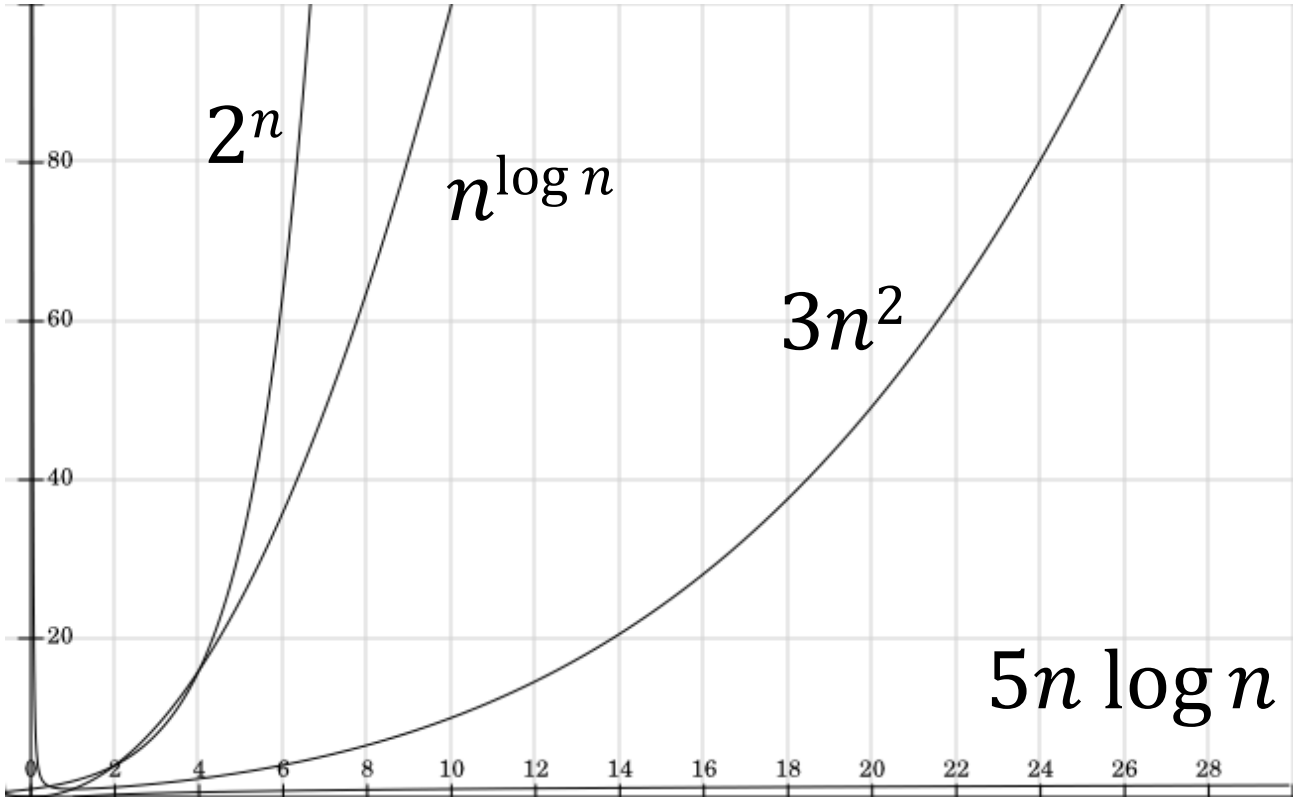
nが小さい場合



nが大きい場合

オーダー記法の評価：ざっくり言えば「係数を無視して一番大きいものをとる」

- 大雑把には係数を無視して一番大きいものをとればよい
  - $T(n) = 3n^2 + n^{\log n} + 5n \log n + 2^n$ ならば $O(2^n)$



オーダー記法の定義：計算量の上界・下界を見積もる

- 関数の上界：$T(n) = O(f(n))$
  - ある正整数n_0とcが存在して、任意の$n \geq n_0$に対し $T(n) \leq c\ f(n)$が成立すること
    - 例： $4n + 4 \leq 5n\ (n \geq 4)$: $c = 5, n_0 = 4$とする
- 関数の下界：$T(n) = \Omega(f(n))$
  - $T(n) \geq c\ f(n)$ （厳密には「あるcが存在し無限個のnに対し」）
- 上界と下界の一致：$T(n) = \Theta(f(n))$
  - c_1とc_2が存在して、$c_1\ f(n) \leq T(n) \leq c_2\ f(n)$
  - 例：$T(n) = 3n^2 + 3n + 10n \log n = \Theta(n^2)$

オーダー記法の性質：２つの性質

1. $f(n) = \mathrm{O}\big(h(n)\big),\ g(n) = \mathrm{O}\big(h(n)\big)$
$\rightarrow f(n) + g(n) = O\big(h(n)\big)$

2. $f(n) = \mathrm{O}\big(g(n)\big),\ g(n) = \mathrm{O}\big(h(n)\big)$
$\rightarrow f(n) = O(h(n))$

オーダー評価の例：多項式の計算（単純法 vs. Horner法）

- 問題：多項式の評価
  - 入力：$a_0, a_1, \ldots, a_n, x$
  - 出力：$a_n x^n + \cdots + a_1 x + a_0$
- 方法：
  - 直接的な計算方法：
    $a_k x^k = a_k \times x \times \cdots \times x$ (k回の掛け算)
    - これは $O(\sum_{k=1}^{n} k + n) = O(n^2)$
  - Horner法：
    $(((\cdots(a_n x + a_{n-1})x + a_{n-2})x + a_{n-3})\cdots)x + a_0$
    - これは $O(n)$

計算量のクラス：多項式時間で解けるものが効率的に解ける問題

- 一秒間に100万回の演算ができるとすると：

$O(\ )$	$n = 10$	$n = 30$	$n = 60$
n	$0.00001\mathrm{s}$	$0.00003\mathrm{s}$	$0.00006\mathrm{s}$
n^2	$0.0001\mathrm{s}$	$0.0009\mathrm{s}$	$0.0036\mathrm{s}$
n^3	$0.001\mathrm{s}$	$0.027\mathrm{s}$	$0.216\mathrm{s}$
2^n	$0.001\mathrm{s}$	$1074\mathrm{s}$	$10^{12}\mathrm{s}$
3^n	$0.06\mathrm{s}$	$2 \times 10^8\mathrm{s}$	$4 \times 10^{22}\mathrm{s}$

- スパコン「京」は1秒間に8162兆回（100億=10^{10}倍速い）
- 「富岳」は1秒間に44京2010兆回（10兆=10^{12}倍速い）

計算量のクラス：多項式時間で解けるものが効率的に解ける問題とする

- P： 多項式時間アルゴリズムを持つ問題のクラス
- NP完全問題、NP困難問題など（おそらく）多項式時間では解けない問題のクラス
  - ただし、実用上現れる重要な問題に、このクラスに属するものが多い
  - 理論的な保証はないが「実用上」有用な様々な戦略によって多くの場合良い解が得られる方法
    - 分枝限定法、局所探索、…
  - 近似アルゴリズムのような、最適解の保証はないが、最適解からどの程度悪いかという保証がある方法

数え上げお姉さん：
単純な解法が破たんする例