

# アルゴリズムとデータ構造②

## ～アルゴリズムの評価～

鹿島久嗣  
(計算機科学コース)

# 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

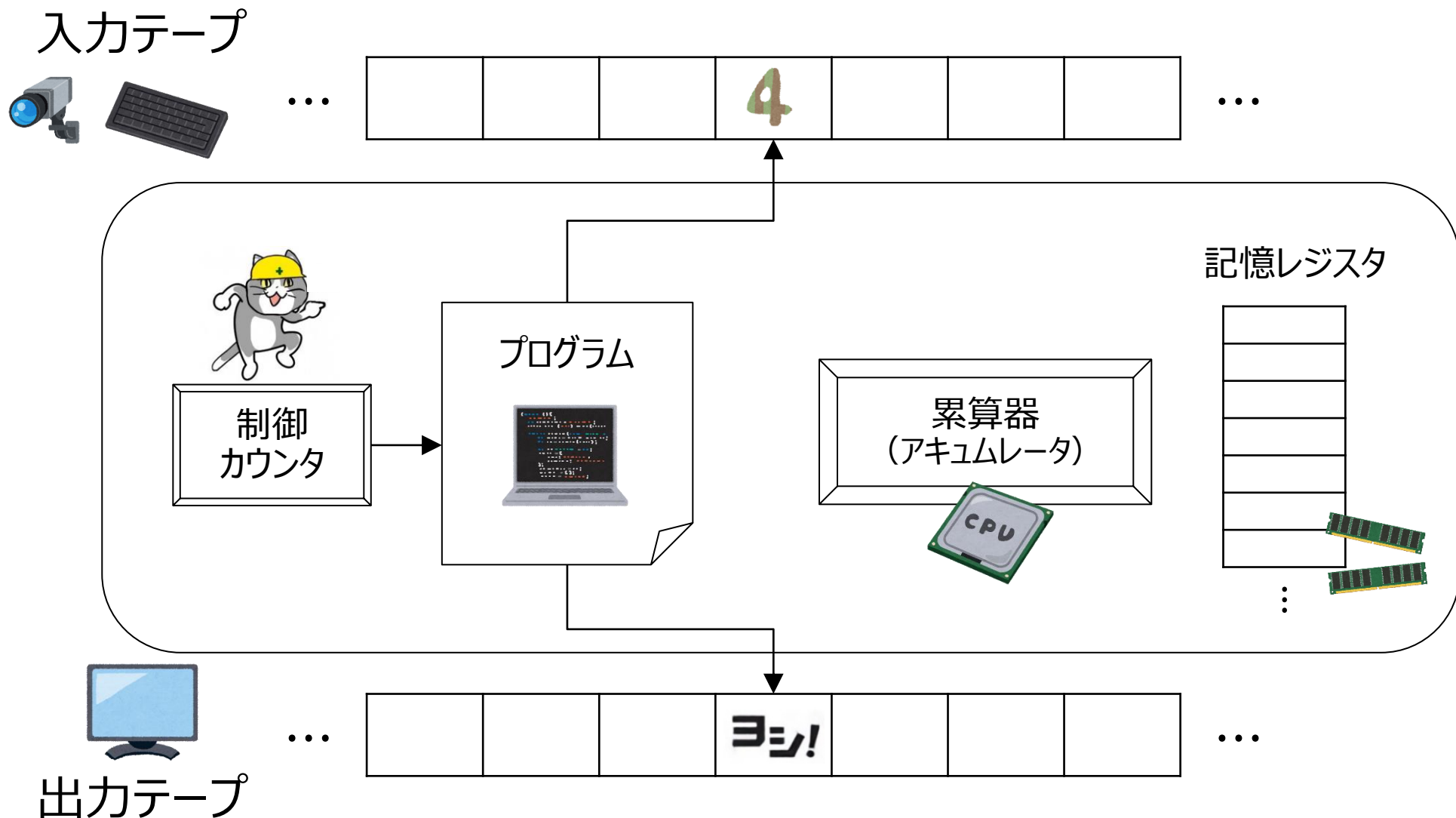
- 我々は特定の言語やハードウェアに依存しない議論がしたい
- 計算のステップ数を数えるためには計算機のモデルを決めておく必要がある
- 計算機の理論モデル：
  - (1930s) Turing機械、 $\lambda$ 計算、Postシステム、帰納的関数、ランダムアクセス機械 (RAM) など様々なモデルが提案される
  - これらによって計算モデルと計算可能性の概念が議論
  - 上記のモデルはすべて同等 → 安定な概念

# 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

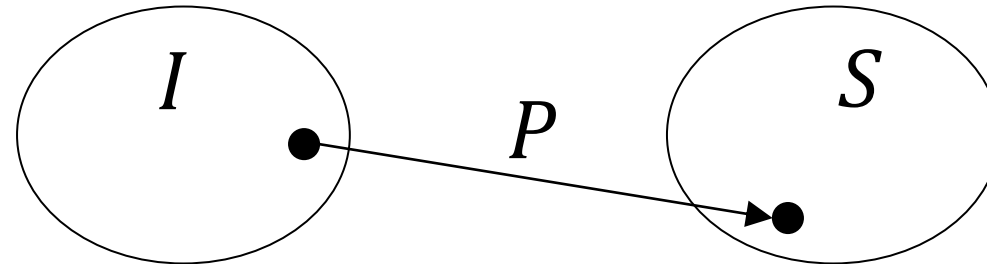
# ランダムアクセス機械: 現在のコンピュータに近い、計算機の理論モデル



「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  $P: I \rightarrow S$



一例：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  $\Leftrightarrow$  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:

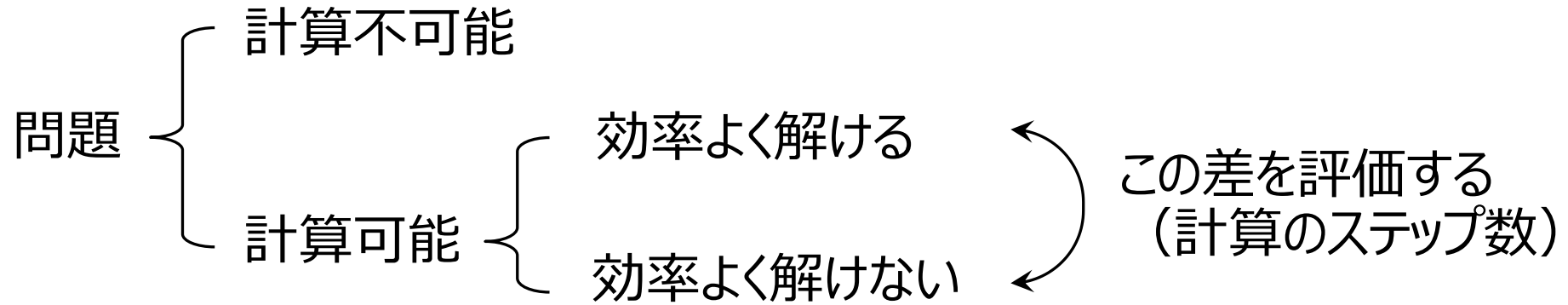
「プログラムの停止問題」は計算不可能な問題

- 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  $x$ （ともに文字列とする）
  - 出力：計算  $(P, x)$  が終了するなら Yes、さもなければ No
- 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（ $(Q', Q')$  を考える）
    - $(Q', Q')$  が停止する  $\Rightarrow$   $(Q', Q')$  は停止しない（矛盾）

# 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  $a_1, a_2, \dots, a_n$  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
  - ※  $\{a_i\}_{i=1, \dots, n}$ 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？

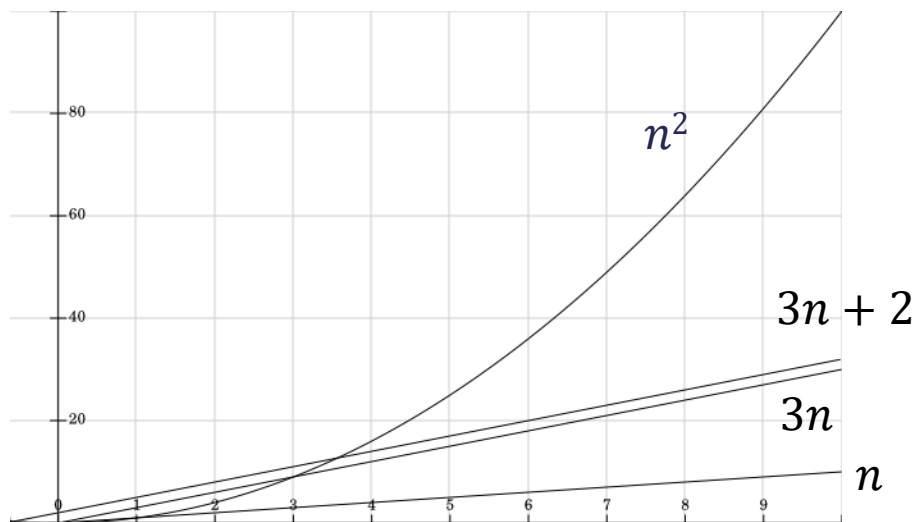
## オーダー評価：

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

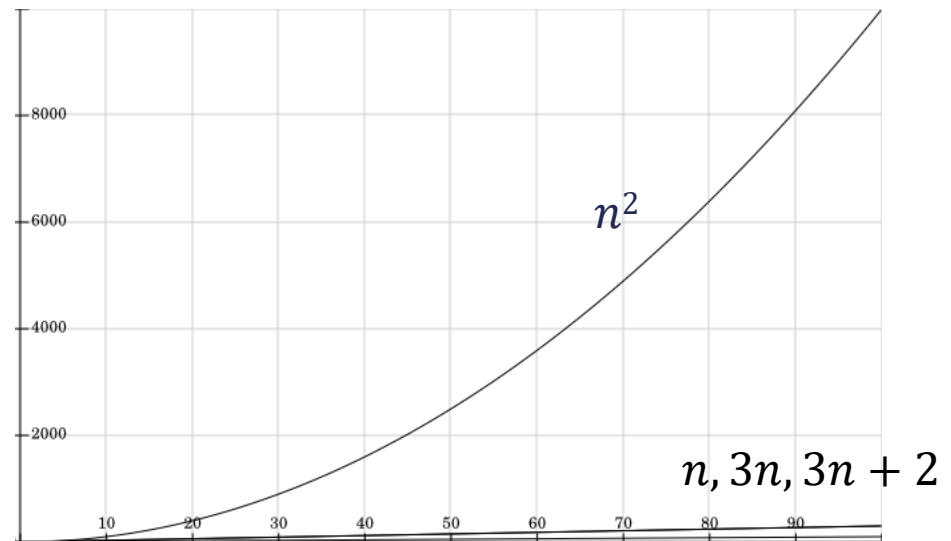
- 以降、特になにも言わない場合は最悪ケースで考える
- (前述の) 名簿を前から順に探すアルゴリズムでは、最悪ケースでは $n$ 回のチェックが必要
  - $n$ 回の比較、 $n$ 回のカウンタ移動、 $n$ 回のデータ読み取りを考えると、 $3n$ 回の操作が必要
  - 加えて、出力と停止を加えると計 $3n + 2$ ステップ？
  - 本質的には $n$ が重要 → オーダー記法で  $O(n)$  と書く
- オーダー記法： $n$ が大きいときの振舞いを評価する
  - $n$ が大きくなったとき、 $n, 3n, 3n + 2$ の違いは、 $n^2$ に比べると極めて小さ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

# 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
- 係数にほとんど意味がなくなる



$n$  が小さい場合



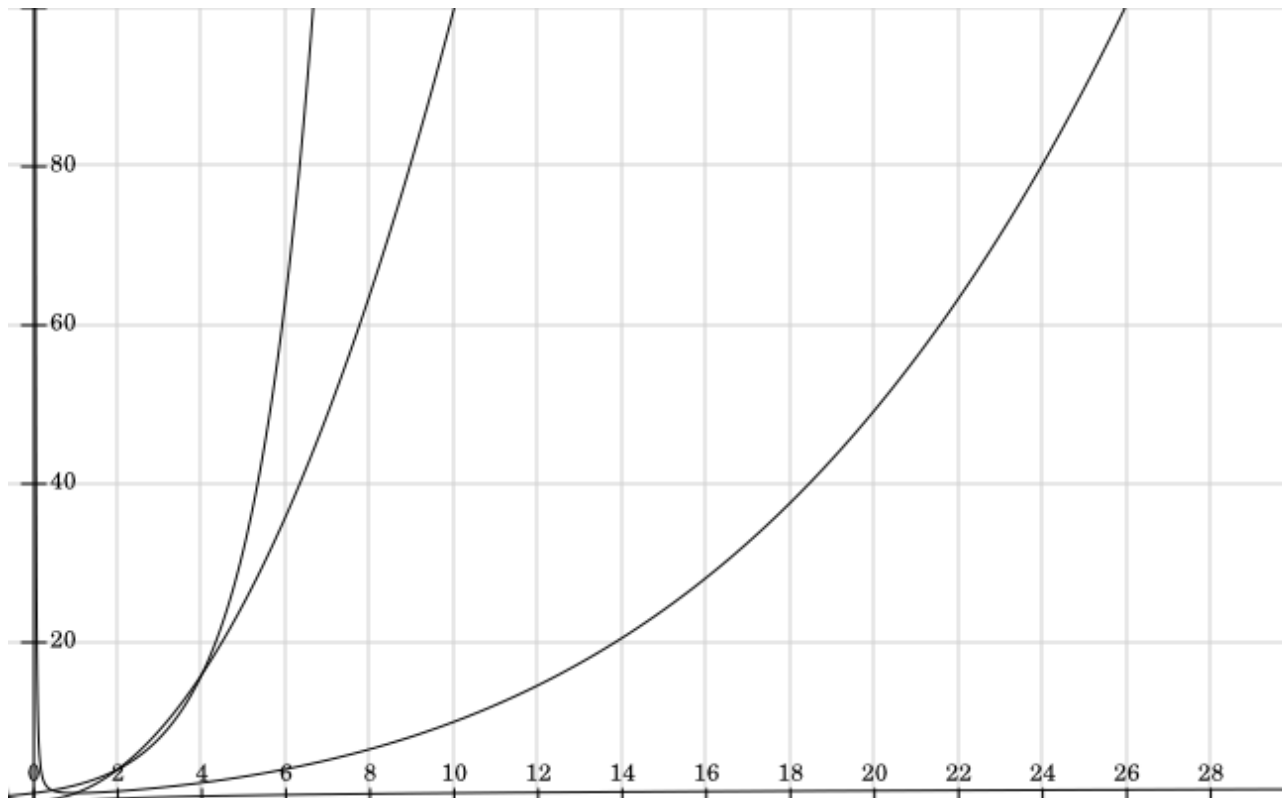
$n$  が大きい場合

# オーダー記法の評価：

ざっくり言えば「係数を見捨てて一番大きいものをとる」

- 直観的には係数を見捨てて一番大きいものをとればよい

$$-T(n) = 3n^2 + n^{\log n} + 5n \log n + 2^n \text{ ならば } O(2^n)$$



# 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

---

$$1. \quad f(n) = O(h(n)), \quad g(n) = O(h(n)) \\ \rightarrow f(n) + g(n) = O(h(n))$$

$$2. \quad f(n) = O(g(n)), \quad g(n) = O(h(n)) \\ \rightarrow f(n) = O(h(n))$$



# 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： $a_0, a_1, \dots, a_n, x$
- 出力： $a_n x^n + \dots + a_1 x + a_0$

### ■ 方法：

#### • 単純な方法：

$$a_k x^k = a_k \times x \times \dots \times x \text{ (} k + 1 \text{回の掛け算)}$$

- $O(\sum_{k=1}^n k + n) = O(n^2)$

#### • Horner法：

$$(((\dots (a_n x + a_{n-1})x + a_{n-2})x + a_{n-3}) \dots )x + a_0$$

- $O(n)$

# 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：

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

同じところを2度通らない道順の数  
Number of routes that do not pass for the same place twice

$6 \times 6$

5億7578万0564 通り

We have an answer. 575,780,564 ways! Wow!

あ、なんか出てるね。  
5億7578万0564通り!  
すごいね!