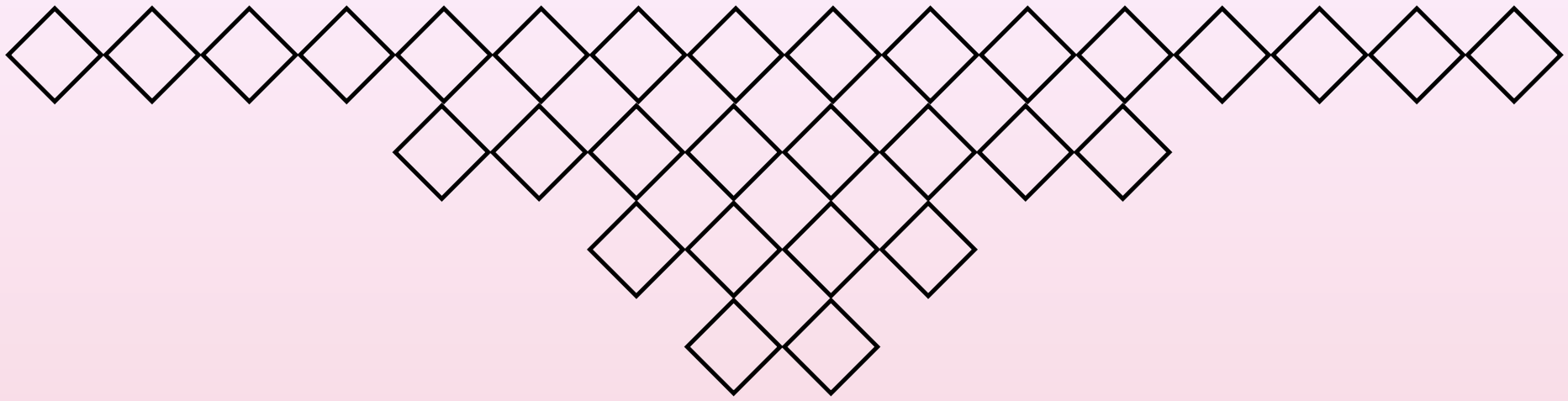
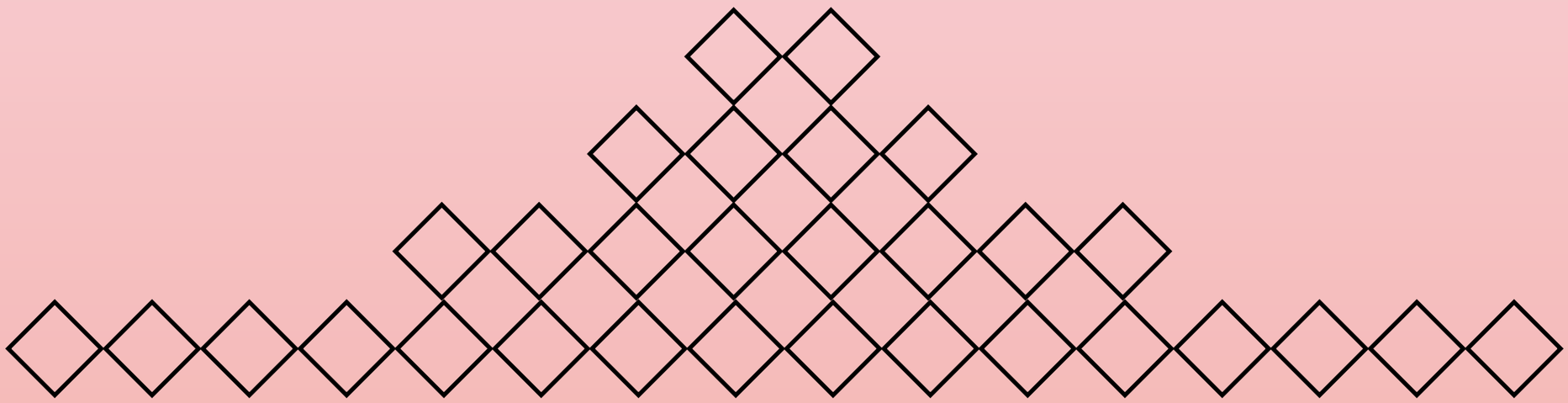


アルゴリズム・データ構造 I 第14回

計算量について

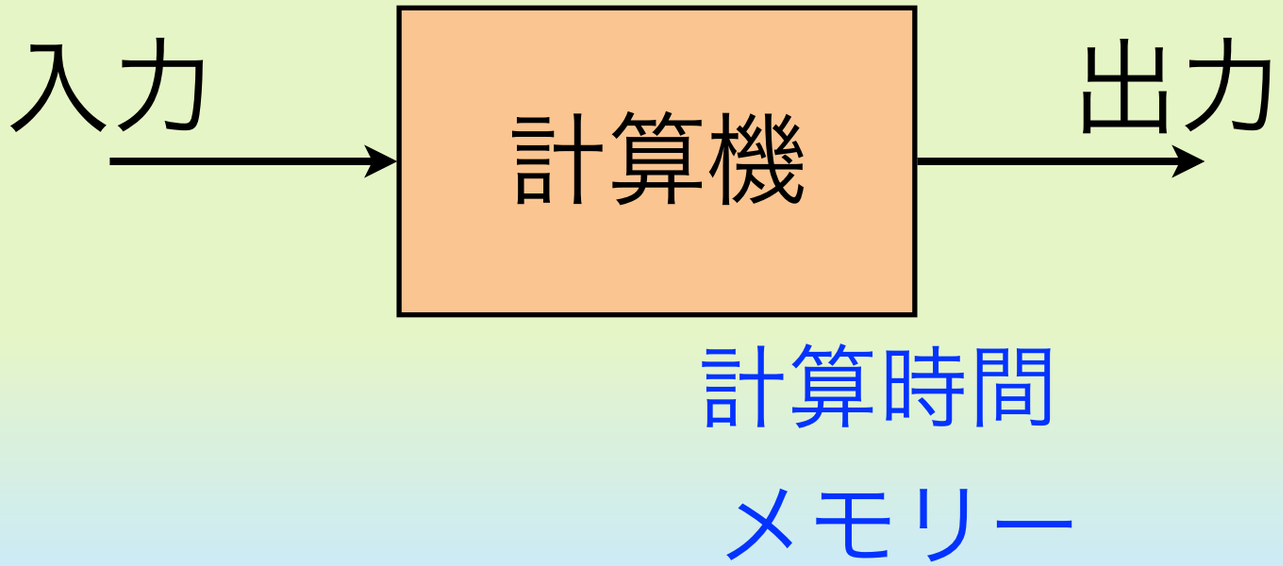
名城大学理工学部情報工学科

山本修身



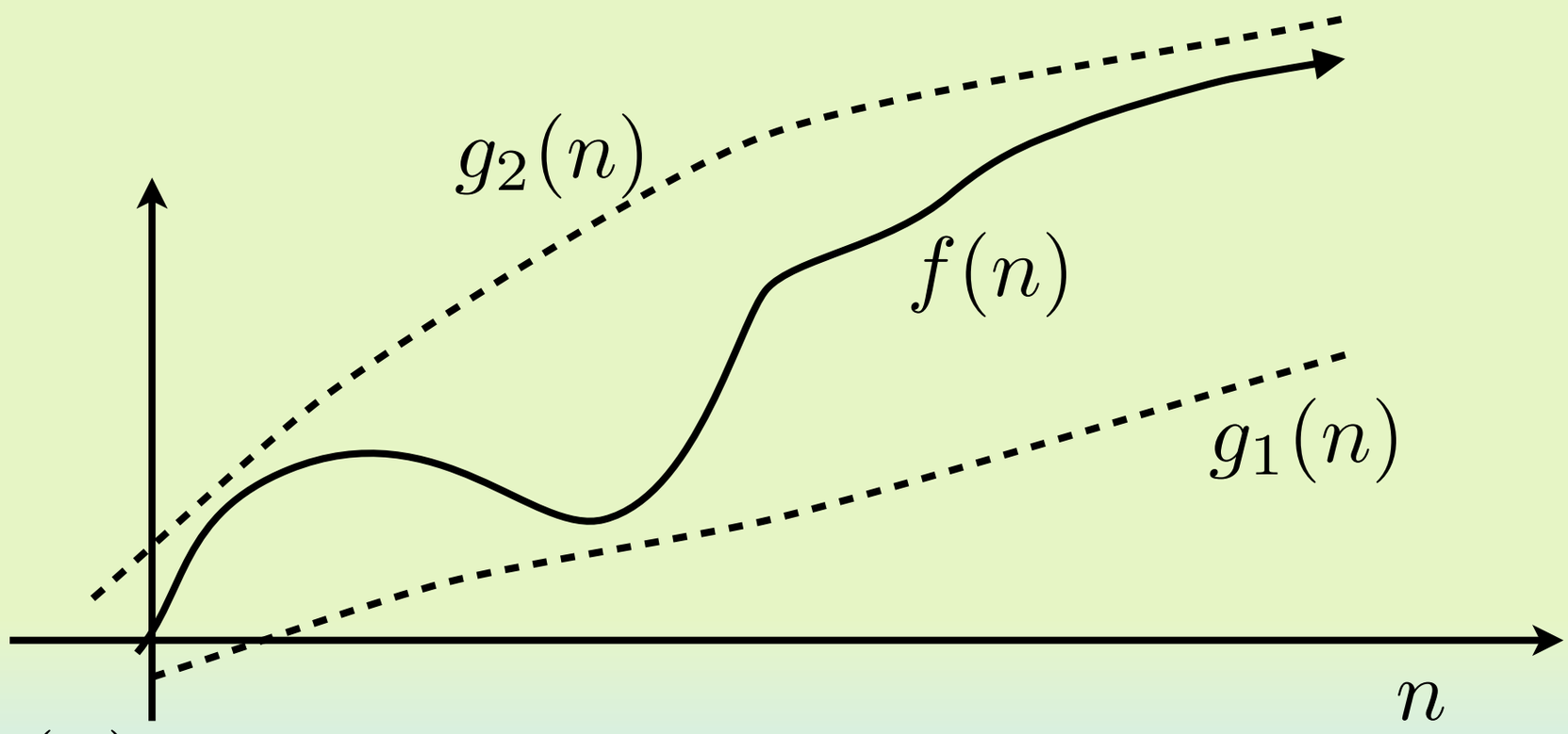
計算量とは

- ある入力データが与えられたとき，それを用いてある計算が行われるとする．大きなデータが与えられれば，計算にはそれなりに時間がかかる．また小さなデータが与えられれば，ある程度早く計算を終了させることが可能かもしれないが，それでも依然としてある程度の計算量が必要となるかもしれない．
- ここで議論する計算量は「**時間計算量**」（計算にどれだけの時間がかかるか）と「**空間計算量**」（計算にどれだけのメモリーが必要になるか）の2種類である．



オーダーという考え方 (1)

- ある変数nが大きくなるにつれて注目している値がどのように変化するかをそれほど厳密ではなく「ゆるく」決める方法としてオーダーという考え方がある.
- ここで問題にするのは，値が大きいか否かではなくて，nが大きくなっていったとき，ずっと，大小関係が維持されるかどうかである.



$$g_1(n) \leq f(n) \leq g_2(n)$$

オーダーという考え方 (2)

nが大きくなれば，an^3の方がbn^2よりもいつか大きくなる（a, bは定数とする）．なぜか？いくらbを大きくしてaを小さくしても，かならず前者が後者を追い抜くことになる.

$$\lim_{n\to\infty} \frac{bn^2}{an^3} = \frac{b}{a} \lim_{n\to\infty} \frac{1}{n} = 0$$

すなわち，ある定数cをうまく選べば，任意のn ≧ 1について,

$$bn^2 \leq c \cdot an^3$$

とすることができる.

オーダーという考え方 (3)

e^nはどのようなan^kよりもnがある程度大きくなると大きくなる．kをいくつにとってもそうなる．なぜか？

$$e^n = 1 + n + \frac{n^2}{2!} + \frac{n^3}{3!} + \cdots$$

となるので，これを用いれば，

$$\frac{e^n}{an^k} = \frac{1}{an^k} + \frac{1}{an^{k-1}} + \frac{1}{2!an^{k-2}} + \frac{1}{3!an^{k-3}} + \cdots + \frac{1}{k!a} + \frac{n}{(k+1)!a} + \cdots$$

より，

$$\lim_{n\to\infty} \frac{e^n}{an^k} = \infty$$

オーダーという考え方 (4)

前のスライドの結果を用いると,

$$\lim_{n\to\infty} \frac{e^n}{an^k} = \infty \qquad \lim_{n\to\infty} \log n = \infty$$

より，$e^n = m$ とおけば，$n = \log m$

$$\lim_{m\to\infty} \frac{m}{a(\log m)^k} = \infty$$

これより，$(\log m)^k$ はmよりもずっと小さい.

以上より結果をまとめれば

(kは任意の正整数)

$$\log n \leq (\log n)^k \leq n \leq n^k \leq e^n$$

が十分大きなnについて成り立ち，≦のギャップは圧倒的に大きくなる

オーダーという考え方 (5)

以下のような記号を導入する

定義：ある関数 $f(n)$ と $g(n)$ について,

$$f(n) = O(g(n))$$

であるとは，適当な定数 c が存在して，任意のn ≥1について

$$f(n) \leq cg(n)$$

となることである．これを「$f(n)$はオーダー$g(n)$である」という

これを用いると，以下のように表現できる．

$$\log n = O((\log n)^k) \quad (\log n)^k = O(n)$$

$$n = O(n^k) \qquad n^k = O(e^n)$$

オーダーという考え方 (6)

オーダーを用いて計算量を測るのは，詳しい計算量を式として表現することが難しいからである．そのため，適当な多項式などで上から押さえるように，計算量の大きさを見積もるのがオーダーである．

$f(n) = O(n^3 + n^2)$という式はオーダーの定義から考えれば意味を持つが，これは，$f(n) = O(n^3)$ と書くのと同じである．すなわち，

$$\underline{f(n) = O(n^3 + n^2) \Leftrightarrow f(n) = O(n^3)}$$

となる．なぜならば，

$$\begin{aligned} f(n) = O(n^3 + n^2) \quad &\Rightarrow \quad f(n) \leq c(n^3 + n^2) \leq cn^3 + cn^2 \\ &\qquad\quad \leq cn^3 + cn^3 = (2c)n^3 \\ &\Rightarrow \quad f(n) = O(n^3) \end{aligned}$$

$$\begin{aligned} f(n) = O(n^3) \quad &\Rightarrow \quad f(n) \leq cn^3 \leq cn^3 + cn^2 = c(n^3 + n^2) \\ &\Rightarrow \quad f(n) = O(n^3 + n^2) \end{aligned}$$

オーダーという考え方 (7)

もっと一般的には, $g_1(n) = O(g_2(n))$ のとき,

$$f(n) = O(g_1(n) + g_2(n)) \Leftrightarrow f(n) = O(g_2(n))$$

$$\begin{aligned} f(n) = O(g_1(n) + g_2(n)) &\Rightarrow f(n) \leq c(g_1(n) + g_2(n)) \\ &\leq c(dg_2(n) + g_2(n)) = c(d+1)g_2(n) \\ &\Rightarrow f(n) = O(g_2(n)) \end{aligned}$$

$$\begin{aligned} f(n) = O(g_2(n)) &\Rightarrow f(n) \leq cg_2(n) \leq cg_1(n) + cg_2(n) \\ &= c(g_1(n) + g_2(n)) \\ &\Rightarrow f(n) = O(g_1(n) + g_2(n)) \end{aligned}$$

オーダーの計算例

$f(n) = O(n^3)$, $g(n) = O(n^4)$ とする．このとき，以下の性質が成り立つ.

- $f(n) + g(n) = O(n^4)$. 証明：適当な定数c_1, c_2を用いて，$f(n) \leq c_1 n^3$, $g(n) \leq c_2 n^4$ と書ける．これより，$f(n) + g(n) \leq c_1 n^3 + c_2 n^4 \leq (c_1 + c_2)\, n^4 = O(n^4)$.
- $f(n)\, g(n) = O(n^7)$. 証明：前の証明の結果を用いて，$f(n)\, g(n) \leq c_1 c_2 n^7 = O(n^7)$.
- $f(g(n)) = O(n^{12})$. 証明： 最初の証明より，$f(n) \leq c_1 n^3$. これより，$f(g(n)) \leq f(c_2 n^4) \leq c_1 (c_2 n^4)^3 = c_1\, c_2^3\, n^{12}$ となり，$f(g(n)) = O(n^{12})$ となる.

◇◇ ソーティングの計算量のオーダー ◇◇

計算量を正確に見積もることは難しいし，入力データの中身によって動作時間が変化する場合がほとんどである．したがって，正確に求めようとしてもあまり意味がない．そこでオーダーを用いる．

アルゴリズム	最悪実行時間	平均実行時間	空間計算量
セレクションソート	$O(n^2)$	$O(n^2)$	$O(1)$
バブルソート	$O(n^2)$	$O(n^2)$	$O(1)$
クイックソート	$O(n^2)$	$O(n \log n)$	$O(n)$
ヒープソート	$O(n \log n)$	$O(n \log n)$	$O(\log n)$
マージソート	$O(n \log n)$	$O(n \log n)$	$O(n)$

すべての問題にはそれを解くプログラムが存在するか？（1）

実はある種の問題に対するプログラムが存在しないことが知られている.

停止性問題は計算不能である (Turing, 1936)

> あるプログラムAとそのプログラムへの入力xが与えられたとき，それを実行したとき，停止するか否かを判定するプログラムは存在しない.

このようなプログラムは存在しない！

すべての問題にはそれを解くプログラムが存在するか？（2）

- 証明は比較的簡単である.

背理法で示す．任意のAとxを入力とするプログラムHが存在して，A(x)が停止するときH(A, x) = 1, A(x)が停止しないときH(A, x) = 0 となるとする.

H(A, A) = 1のとき停止せず，H(A, A) = 0のとき停止するプログラムM(A)を作ることができる.

H(A,A)の右側のAはAのプログラムをデータとみて，エンコードしたものであると考える.

```
function M(A){
    if (H(A, A) == 1)
        while (true){}
    else
        return 1
}
```

すべての問題にはそれを解くプログラムが存在するか？（3）

- このとき，M(M)は停止するか？

M(M)が停止する→H(M, M) = 0→ M(M)は停止しない

M(M)が停止しない→H(M, M) = 1→ M(M)は停止する

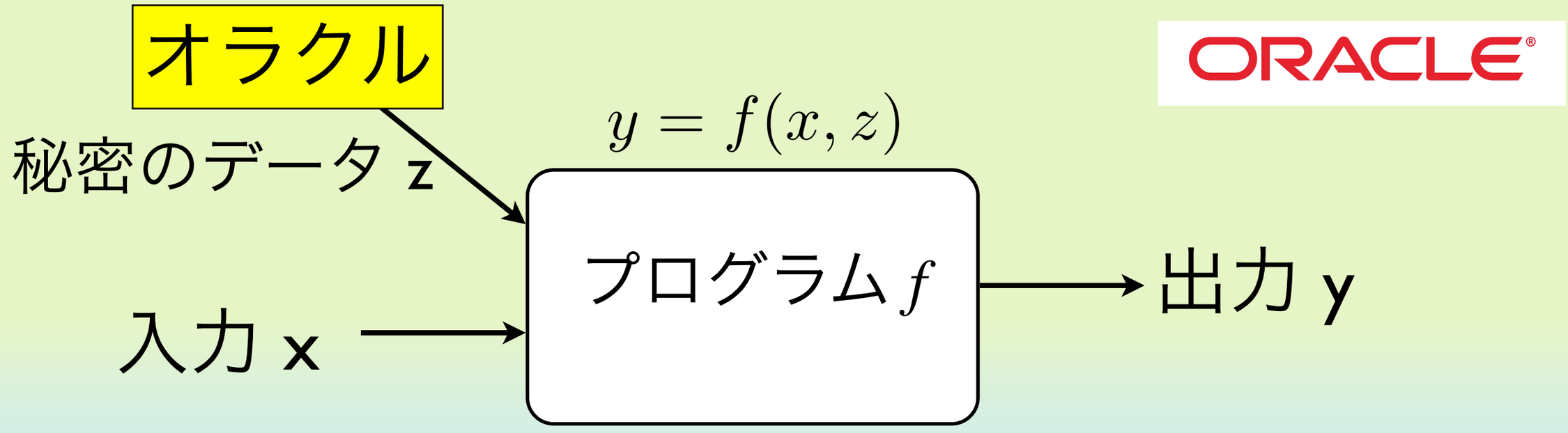
↓

どちらにしても矛盾．このことより，Hは存在しない．

非決定的な計算（1）

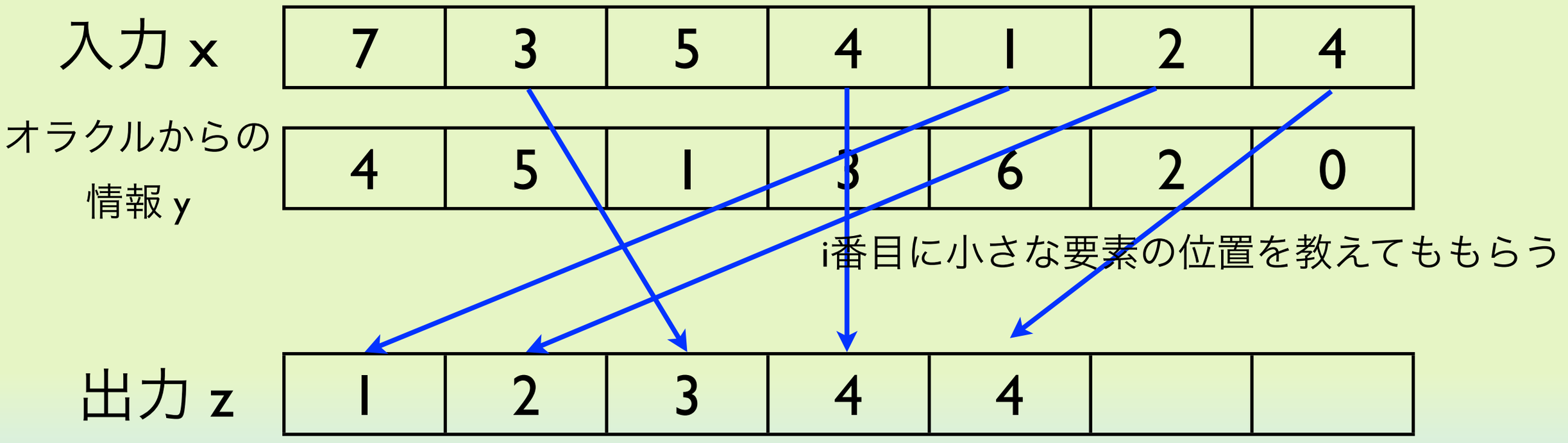
現実的ではないように思えるかもしれないが，「計算できる」ということの定義をここで変更する.

入力 x に対して，出力yを計算するための計算するには，まず，**オラクル（データベース会社ではない）**にお願いして秘密のデータ z をもらう．その z と x を用いて，出力 y = f(x, z) を計算する．ただし，データ z はif文の分岐でしか使われない．オラクルは万能の神様であって，いつでも適切なデータを与えてくれる．また，fは普通のコンピュータ上のプログラムである.



非決定的な計算（2）

- 非決定的な計算でソーティングを行うと，n個のデータはO(n)でソートすることができる.
- n番目に大きなデータが元の配列の何番目にあるかをまとまったデータとしてもらえば，それを用いて，n回でデータを入れ換えることが可能である.



計算クラスの階層

- 入力サイズnに対してどの程度の計算時間や計算領域を消費するかによって，扱う問題を分類することができる.

計算不能
アルゴリズムは存在しない

計算可能
(recursive)
アルゴリズム
が存在する

決定性の計算

指数時間計算可能 $O(\mathrm{e}^{p(n)})$

EXPTIME

多項式領域計算可能 $O(p(n))$

PSPACE

多項式時間計算可能 $O(p(n))$

P

非決定性の計算

非決定性多項式時間計算可能

非決定性計算機で $O(p(n))$

NP

NP完全問題

NP = P なのか，NP ⊃ P なのかは未だにわかっていない．オラクルに聞いてもzを利用しなければ普通の計算と同じなので，NP ⊇ P は明らか．普通は NP ≠ P と信じられているが，本当のところはわからない．この問題はミレニアム懸賞問題（クレイ数学研究所によって懸賞金がかけられている問題）である．解ければ100万ドル（１億円くらい）もらえる．

P = NPかどうかはわからないが，Cookは1970年にNPのある部分クラスでこのクラスの問題の１つでもPに含まれることが示せれば，P = NPであるようなものが存在することを示した．とくにそのようなクラスに SAT （論理式の充足可能性問題）が含まれることを示した．このクラスの問題を**NP-完全 (NP-complete)**と呼ぶ．その後，ナップザック問題などがNP-完全であることも示されている．また，NPに含まれることを仮定するとNP-完全になる問題のことを**NP-困難（NP-hard)** と呼ぶ．

◇◇ 良く知られたNP完全問題の例 (1) ◇◇

- **SAT (充足可能性問題)** n個の変数 $x_1, x_2, \ldots, x_n$ をもつ論理式を真にするような値の組み合せが存在するか否かを判定する問題. 論理式とは, n個の変数と論理和（∨）と論理積（∧）および否定（¬）によって作られる式のことで, 変数の値が決まれば, その式の値を機械的に評価することができる. 与えられた論理式を真にする答をあらかじめ教えてもらえば, それが正しい答えであるか否かを判定することは簡単にできる. したがって, この問題がNPに入ることは明らかである.
- SATよりも束縛のきつい問題として, CNF-SAT（CNF = conjunctive normal form: いくつかの論理和の論理積の形), 3-SAT（CNF-SATのうち, 論理和の中身が高々3つしかないもの）がある. これらもNP完全であることが知られている（2-SATは多項式時間計算可能）.

◇◇ 良く知られたNP完全問題の例 (2) ◇◇

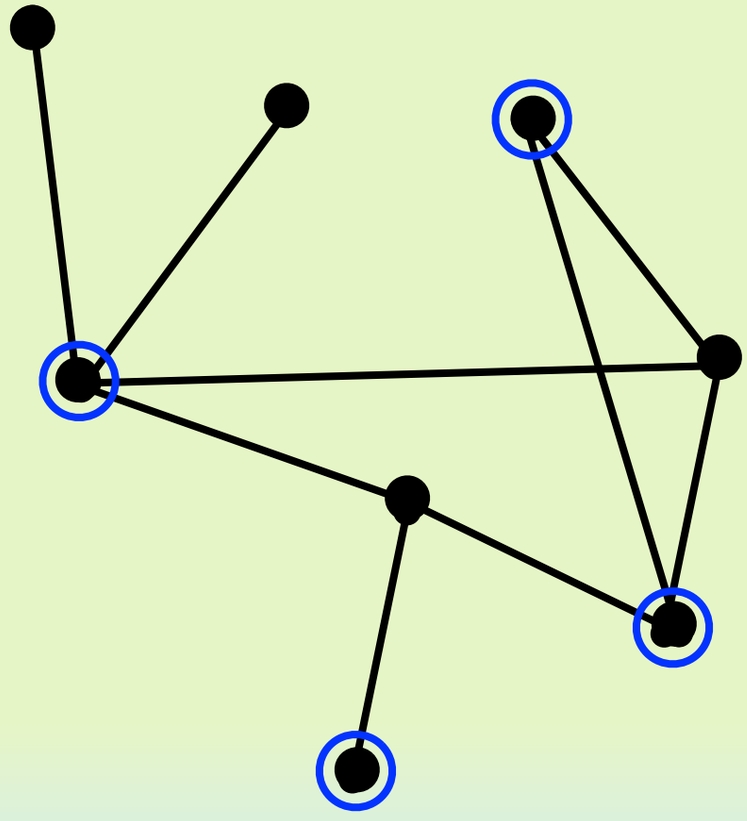
- **ナップザック問題.** n個の正の数 $p_1, p_2,...,p_n$が与えられたとき，このうちのいくつかの数を足し合わせてV以上 C 以下にすることが可能かを判定する問題.

$$V \leq \sum_{k=1}^{m} p_{i(k)} \leq C$$

- 普通はいくつかを足し合わせてCを超えないで最大にする問題のこと．また，「C以下にする」という場合の数と「V以上にする」という場合の数をそれぞれの部品について別々に用意する場合もある（こちらの方が普通である．価値の総和を最大化（V以上に）して．なおかつ費用の総和を最小化（C以下）にするということ）

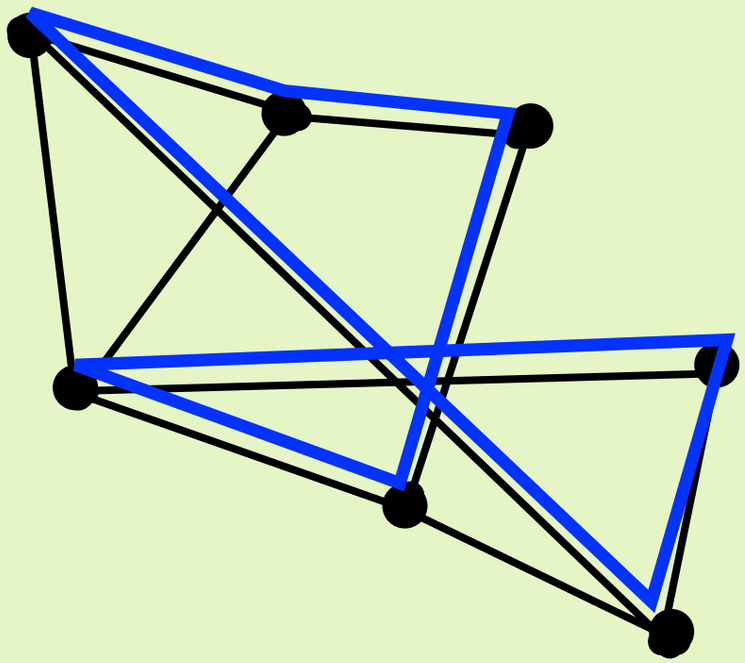
◇◇ 良く知られたNP完全問題の例 (3) ◇◇

- **頂点被覆問題.** グラフが与えられたとき，いくつかの頂点にマークをつけて，すべての辺のいずれかの端にマークがついているようにしたものを頂点被覆と呼ぶ．頂点被覆のうちちょうどv個のものが存在する否かを判定する問題が頂点被覆問題である.



◇◇ 良く知られたNP完全問題の例 (4) ◇◇

- **ハミルトン閉路問題.** 与えられたグラフのすべての頂点を通って一周する経路が存在するか否かを調べる問題.



ハミルトン閉路