アルゴリズムとデータ構造

第2回アルゴリズムと計算量

授業スライドURL:
http://www-ikn.ist.hokudai.ac.jp/~arim/pub/algo/

事務連絡:アルゴリズムとデータ構造　H29　授業予定(改訂)

2017/04/11
第2版

回	日付	曜	内容	担当
1	4月6日	木	ガイダンス	有村
2	4月11日	火	アルゴリズムと計算量	有村
3	4月13日	木	基本的なデータ構造	有村
4	4月18日	火	再帰	有村
5	4月20日	木	探索のためのデータ構造(1)	有村
6	4月25日	火	探索のためのデータ構造(2)	有村
7	4月27日	木	整列のアルゴリズム(1)	喜田
8	5月02日	火	整列のアルゴリズム(2)	喜田
9	5月09日	火	文字列照合アルゴリズム	喜田
10	5月11日	木	グラフとネットワークのアルゴリズム(1)	喜田
11	5月16日	火	グラフとネットワークのアルゴリズム(2)	喜田
12	5月18日	木	グラフとネットワークのアルゴリズム(3)	喜田
13	5月23日	火	計算困難な問題を扱うには？	喜田
14	5月25日	木	予備	休講予定
15	5月30日	火	アルゴリズム関連の最近の話題	有村
16	6月01日	木	試験	

第2回アルゴリズムと計算量

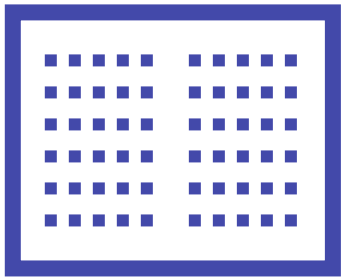
- 今日の内容
  - アルゴリズムの計算量とは？
  - 漸近的計算量
  - オーダーの計算の方法
  - 最悪計算量と平均計算量

- ポイント
  - オーダー記法
  - ビッグオー，ビッグオメガ，ビッグシータ

よいアルゴリズムとは？

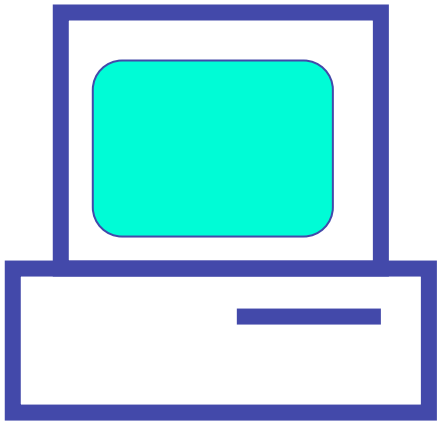
いろいろな性能

- **計算時間**
- **メモリサイズ**
- 外部記憶への入出力数
- ネットワークの通信量



ほかの要素

- モジュラリティ Modularity
- 再利用可能性 Reusability
- 読みやすさ Readability
- 移植しやすさ Portability
- ...



ここでは，計算時間とメモリサイズで測る

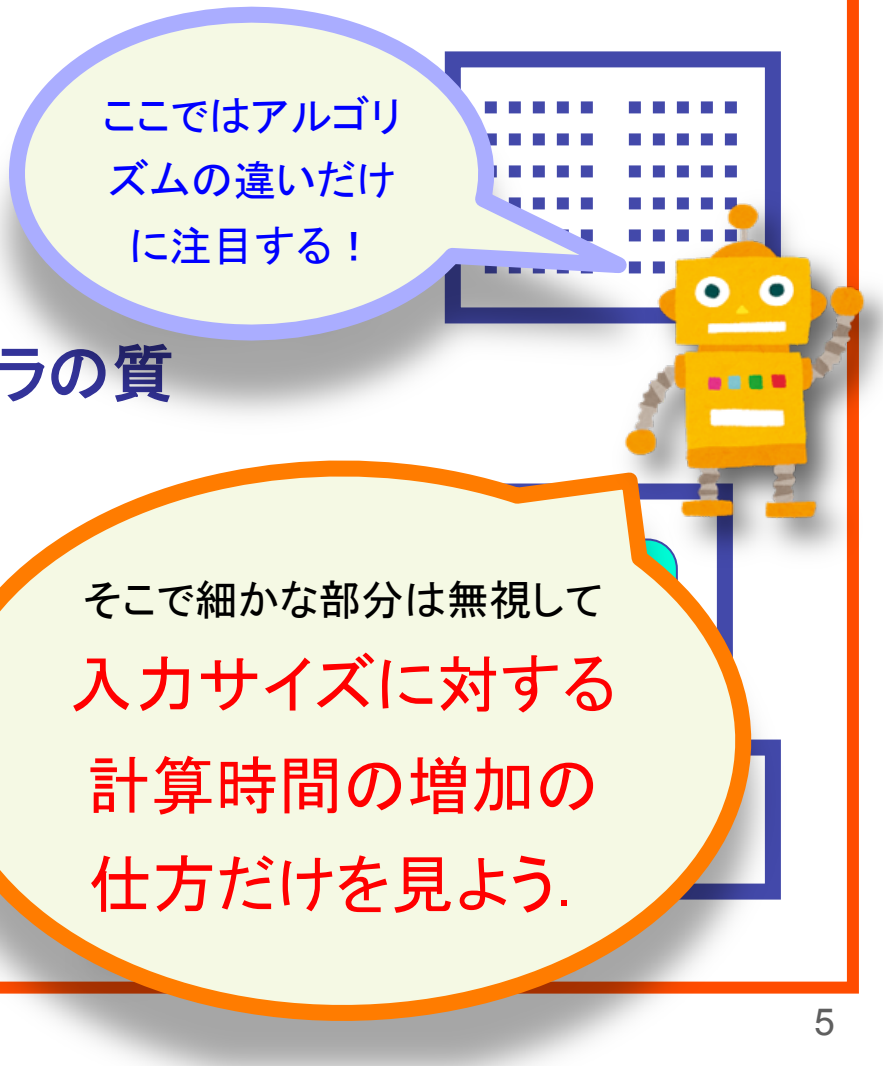
計算時間の測り方

計算時間はいろいろな要素に依存する!

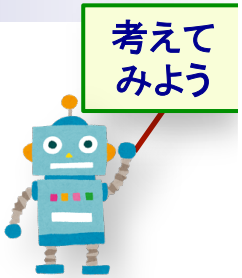
- ◆1. アルゴリズム
- ◆2. コーディングスタイル
- ◆3. 言語の種類とコンパイラの質
- ◆4. ハードウェアの性能
- ◆5. 入力そのもの

だから簡単には比べられ

しかし. . .



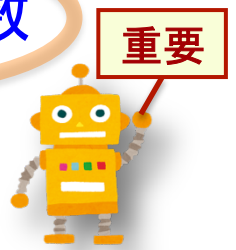
計算量の評価



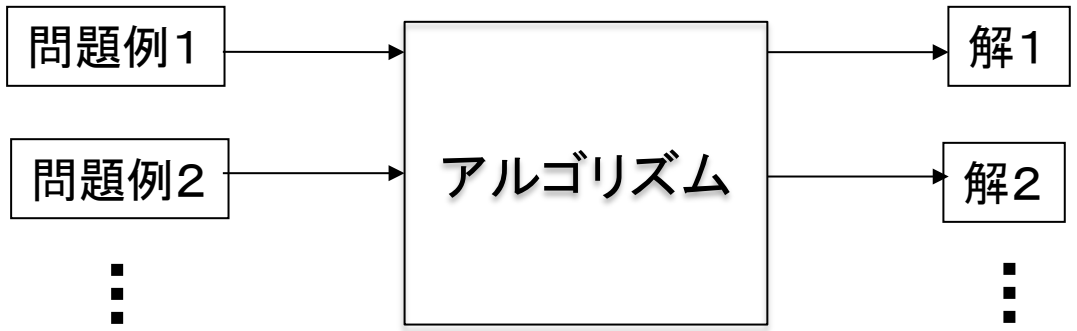
- 計算量の種類
  - 時間計算量(time complexity)
  - 領域計算量(space complexity)

ステップ数

メモリ量



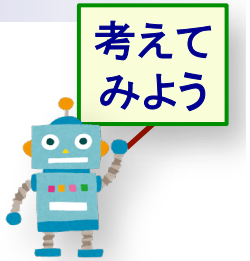
- ある問題を解くアルゴリズム



1つの問題＝無限個の問題例の集合

問題と問題例

問：それぞれの問題（例）で、
入力サイズは何になるか？



問題

GCD (Greatest Common Diviser) [最大公約数を求める問題]

入力：正整数a_0, a_1

出力：a_0とa_1の最大公約数

入力サイズ＝a_0とa_1のビット長

問題例　$(a_0,a_1)=(28,12)$, $(987654321,123456789)$

問題

SUBSET-SUM [部分和問題]

入力サイズ＝数の個数n+1
（または全部の数のビット長の和）

入力：n+1個の正整数$a_0,a_1,\ldots,a_{n-1},b$

出力：$\Sigma_{j\in S}\, a_j=b$となる$S\subseteq\{0,1,\ldots,n-1\}$が存在したらyes、存在しなかったらno

問題例　$(a_0,a_1,a_2,b)=(1,2,3,5)$, $(a_0,a_1,a_2,a_3,a_4,a_5,b)=(1,2,4,8,16,32,60)$

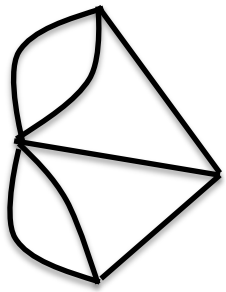
問題と問題例

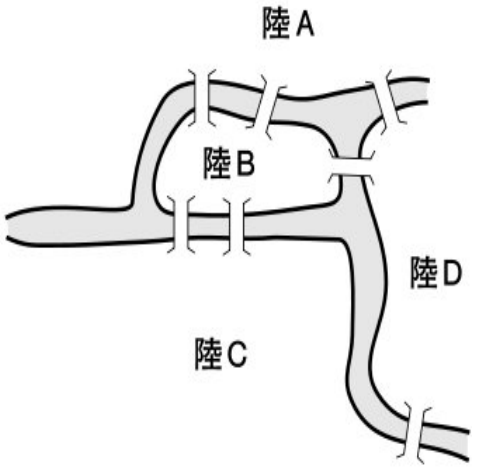
問題

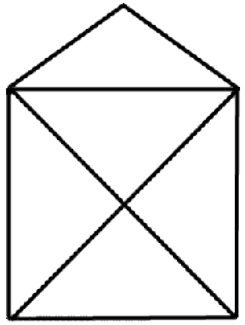
EULER-PATH [オイラー路を求める問題]
入力：グラフ G = (V,E)
出力：オイラー路（各辺を1度だけ通る路）があれば、それらの路の1つ、なければ「なし」

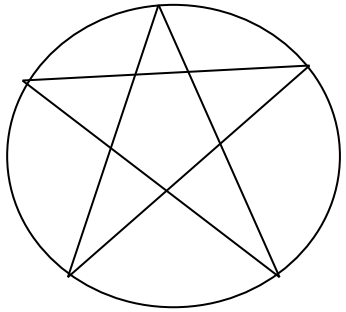
入力サイズ＝
すべての頂点数と辺数の和

問題例









$V=\{v_1,v_2,v_3,v_4\}$
$E=\{e_1,e_2,e_3,e_4,e_5,e_6\}$

ケーニッヒベルグの橋渡り問題

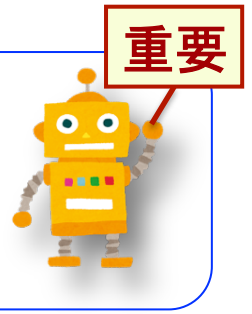
例）入力サイズ＝頂点数と辺数の和 = 4 + 6 = 10

計算量の評価法

問題例の規模によって計算量が変わる

⇒問題例の規模に応じた評価

⇒**入力長nの関数T(n)として計算量を評価**



入力長および計算量は計算コストモデルに依存

- **定数(一様)コストモデル**

  すべての数を1語とみなし、どの基本命令も単位時間で実行できると仮定

- **対数コストモデル**

  各々の数はそれを表現するのに必要なビット数の語数が必要であり、
  基本命令の実行はビット数に応じた時間が必要と考える

大きな数字を扱うことが本質的な問題は対数コストモデル、そうでない場合は定数コストモデル

(例)GCDの入力(a_0,a_1) 対数コストモデル $\lceil \log a_0 \rceil + \lceil \log a_1 \rceil$

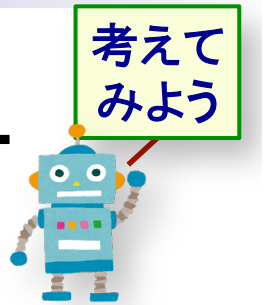
SUBSET-SUMの入力$a_0,a_1,\ldots,a_{n-1},b$ 定数コストモデル $n+1$

EULER-PATHの入力V,E 定数コストモデル $|V|+|E|$

頂点の数と辺の数

実際には要素を分ける区切り文字が必要だがオーダー評価のため定数倍は影響ない

考えてみよう：時間計算量の測り方



- 基本的に、時間計算量は、入力が与えられたときのプログラム実行のステップ数ではかる。
- すごく正確に、ステップ数をはかる必要があるか？
  - 例）ステップ数 $T(n) = 2n^2+5n+1000$
- ステップ数は、おなじアルゴリズムでも、プログラミング言語や、コンパイラ、書き方で変わる。
- 入力nに対して、だいたいの増加具合がわかればよいのではないか。
- → 漸近的計算量（オーダー）
  - 例）ステップ数 $T(n) = O(n^2)$

領域計算量にも同じ議論がなりたつ

ここは大事！

「XXアルゴリズムの計算時間は入力の二乗時間オーダー」
「クイックソートアルゴリズムはO(n log n)時間」
オーダーとは何か？

1.3 計算量の評価、漸近的計算量

「オーダー」記法の勉強

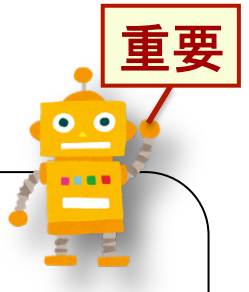
http://www.shutterstock.com/

漸近的計算量：O（ビッグオー）

漸近的上界 = asymptotic upper bound

オーダー評価

- 計算量T(n)は十分大きなnに対して評価する。
- 定数倍の差はないものとみなす。



定義：漸近的上界(ビッグオー記法)　記号O

$T(n) = O(f(n))$ ⇔ ある実数$c>0$と自然数n_0が存在して全ての$n \geqq n_0$に対して$T(n) \leqq cf(n)$が成り立つ

「T(n)は，オーダー f(n) である」または，「T(n)は，ビッグオーf(n) である」と読む

単にオーダー記法ともいう

意味「関数T(n)はf(n)と同じか小さい」

漸近的上界はいくらでも存在するができるだけ単純で精度のよいものがよい

(1) $2n^2+5n+1000 = O(n^2)$ ← 一番良い best!
(2) $2n^2+5n+1000 = O(n^2+n)$ ← (1)より複雑
(3) $2n^2+5n+1000 = O(n^3)$ ← (1)より精度が悪い

なぜ計算時間をオーダーで測るのか？

質問： 問題のサイズを大きくしていったらどうなるか？

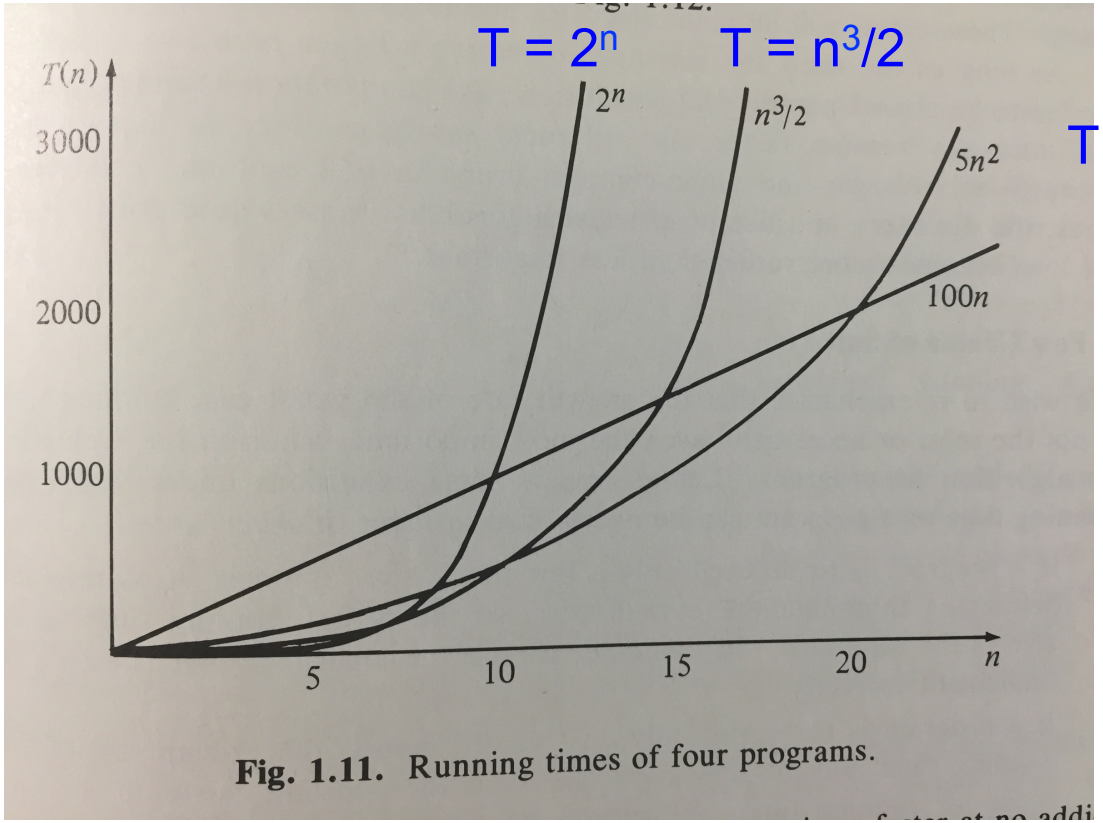


Fig. 1.11. Running times of four programs.

- それぞれ, 2^n, $n^3/2$, $5n^2$, 100nの計算時間をもつ4つのプログラムに対して，その入力nを増やしながら走らせたときの計算時間のグラフ.
- オーダーが大きいほど，係数によらず計算時間が急速に増加する

なぜ計算時間をオーダーで測るのか？

- 質問： 時間をかけた分だけ大きなサイズの問題が解けるか？

smaller percentage increase in problem size. ...
solve only small problems no matter how fast the underlying computer.

Running Time $T(n)$	Max. Problem Size for 10^3 sec.	Max. Problem Size for 10^4 sec.	Increase in Max. Problem Size
$100n$	10	100	10.0
$5n^2$	14	45	3.2
$n^3/2$	12	27	2.3
2^n	10	13	1.3

Fig. 1.12. Effect of a ten-fold speedup in computation time.

- 図）4つプログラムに対して，左欄から，その計算時間と，それぞれ，1千秒と1万秒で解ける最大の問題サイズを示す．
- 観察： 計算時間を10倍にすると，O(n)時間アルゴリズムなら10倍のサイズの問題がとける．しかし，$O(n^3)$時間なら2.3倍，$O(2^n)$時間なら1.3倍しか解ける問題のサイズが大きくならない！

漸近的計算量の性質

性質：$T_1(n)=O(f(n))$, $T_2(n)=O(g(n))$のとき、次の等式が成立

$T_1(n)+T_2(n)=O(\max\{f(n),g(n)\})$ ← いくつかの処理を順次行う場合は一番遅い処理が全体の処理速度を支配する

$T_1(n)T_2(n)=O(f(n)g(n))$ ← 処理を繰り返し行うとその回数分時間がかかる

証明してみよう！

(例)　$n^2+n^3=O(n^3)$,　$n^2 \cdot n^3=O(n^5)$

オーダー表記の注意点　　左辺の精度≧右辺の精度

(例)　○ $2n^2+5n+1000 = O(n^2)$
　　× $O(n^2)=2n^2+5n+1000$

漸近的計算量：Ω（ビッグオメガ）

漸近的下界=asymptotic lower bound

定義：漸近的下界（ビッグオメガ記法）　記号Ω

$T(n)=\Omega(f(n))$ ⇔ あるc,n_0が存在して全てのn≧n_0に対して
$T(n)\geqq cf(n)$が成り立つ

「T(n)はビッグオメガ f(n)である」と読む

発展

意味「関数T(n)はf(n)と同じかもっと大きい」

(例) $2n^2+5n+1000 = \Omega(n^2)$

$$T(n)=\begin{cases} n^2 & \text{if nは奇数} \\ n^3 & \text{if nは偶数} \end{cases} \Rightarrow T(n)=\Omega(n^2)$$

下の定義では
$\Omega(n^3)$

以下の定義をΩの定義として採用することもある。(研究論文ではこちらも多い)

別定義：漸近的下界（ビッグオメガ記法の別定義）

$T(n)=\Omega(f(n))$ ⇔ あるcが存在して無限個のnに対して
$T(n)\geqq cf(n)$が成り立つ

上の定義の方が厳しいので上の定義を満たせばこちらの定義も満たす

漸近的計算量：Θ（ビッグシータ）

定義：漸近的にタイトな限界(asymptotic tight bound)　記号Θ

発展

$T(n)=\Theta(f(n))$ ⇔ ある実数$c_0, c_1>0$と自然数n_0が存在して全ての$n \geqq n_0$に対して$c_0 f(n) \leqq T(n) \leqq c_1 f(n)$が成り立つ

「T(n)はビッグシータ f(n) である」と読む

意味「関数T(n)はf(n)とだいたい同じ」

(例) $2n^2+5n+1000 = \Theta(n^2)$

$$T(n)=\begin{cases} n^2 & \text{if } n\text{は奇数} \\ n^3 & \text{if } n\text{は偶数} \end{cases} \Rightarrow \quad T(n)\neq\Theta(n^3)$$

一般に

$T(n)=\Theta(f(n))$ ⇔ $T(n)=O(f(n)), T(n)=\Omega(f(n))$

Ωの別定義であれば`⇒'のみ成立

教養としてoとωも知っておこう！

T(n)はf(n)より真に小さいこと

発展

漸近的にタイトでない上界　　記号o

T(n)=o(f(n)) ⇔ **任意の**定数c>0に対し,あるn_0が存在して
全てのn≧n_0に対してT(n)≦cf(n)が成り立つ

「T(n)はスモールオー f(n) である」

T(n)はnが無限大に近づくにつれてf(n)に対して相対的に小さくなる、

つまり、 $\lim_{n\to\infty} \frac{T(n)}{f(n)} = 0$ となる。

(例) $2n = o(n^2)$, $2n^2 \neq o(n^2)$

T(n)=o(f(x))
⇕
f(x)=ω(T(n))

漸近的にタイトでない下界　　記号ω

T(n)はf(n)より真に大きいこと

T(n)=ω(f(n)) ⇔ **任意の**定数c>0に対し,あるn_0が存在して
全てのn≧n_0に対してT(n)≧cf(n)が成り立つ

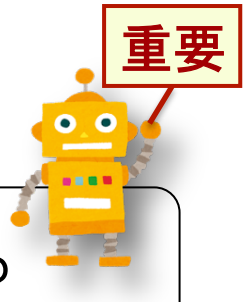
「T(n)はスモールオメガ f(n) である」

T(n)はnが無限大に近づくにつれてf(n)に対して相対的に大きくなる、

つまり、 $\lim_{n\to\infty} \frac{T(n)}{f(n)} = \infty$ となる。

(例) $2n^2 = \omega(n)$, $2n^2 \neq \omega(n^2)$

オーダーの計算法：基本のやりかた



規則1： T(n)がnの多項式ならば，最大次数の項のオーダーになる

例： $2n^2 + 3n + 100 = O(n^2)$

例： $10n + 2\sqrt{n} + 5 = 10n^1 + 2n^{0.5} + 5 = O(n)$

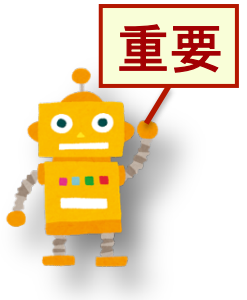
規則2： 次のオーダーの式が成立する：

- log(n) = O(n)
- n = $O(2^n)$
- 任意のc>0に対して，log n = $O(n^c)$

規則3： T(n)がいくつかの項の和ならば，最大次数の項のオーダーになる

例： $3n^2 + 2\sqrt{n} + 100\log n + 5 = O(\log n)$

オーダーの計算法：自分で計算してみる
増加のオーダーによる比較(1/2)



計算時間が$T_1(n)$と$T_2(n)$のアルゴリズムではどちらが速いか？
⇒増加のオーダーが小さい方が速い

規則4：　$T_1(n)$よりも$T_2(n)$の方が増加のオーダーが大きい

$$\Leftrightarrow \lim_{n\to\infty}\frac{T_1(n)}{T_2(n)}=0 \qquad \Leftrightarrow \lim_{n\to\infty}\frac{T_2(n)}{T_1(n)}=\infty$$

規則5：$T_1(n)$と$T_2(n)$は増加のオーダーが等しい

$$\Leftrightarrow T_1(n)=\Theta(T_2(n)) \qquad \Leftrightarrow T_2(n)=\Theta(T_1(n))$$

$T_1(n)$と$T_2(n)$は増加のオーダーが等しい $\Leftarrow \not\Rightarrow$ $\lim_{n\to\infty}\frac{T_1(n)}{T_2(n)}=c$ for some $0<c<\infty$

オーダーの計算法：増加のオーダーによる比較(2/2)

発展

規則6： l'Hospitalの法則

1. f(x),g(x):R →Rが微分可能でf(a)=g(a)=0,x=a以外でg'(x)≠0であれば

$$\lim_{x\to a}\frac{f(x)}{g(x)}=\lim_{x\to a}\frac{f'(x)}{g'(x)}$$

2. f(x),g(x):R →Rが微分可能で$\lim_{x\to\infty}f(x)=\lim_{x\to\infty}g(x)=\infty$であれば

$$\lim_{x\to\infty}\frac{f(x)}{g(x)}=\lim_{x\to\infty}\frac{f'(x)}{g'(x)}$$

(例) $n^{0.01}$と$\log^{100}n$ではどちらが漸近的増加率が大きいか？

$$\lim_{n\to\infty}\frac{\log^{100}n}{n^{0.01}}=\lim_{n\to\infty}\frac{100\log^{99}n\times(1/n)}{0.01\,n^{-0.99}}=\lim_{n\to\infty}\frac{100\log^{99}n}{0.01\,n^{0.01}}$$

$$=\lim_{n\to\infty}\frac{100\cdot 99\log^{98}n}{0.01^2\,n^{0.01}}=\cdots=\lim_{n\to\infty}\frac{100!}{0.01^{100}\,n^{0.01}}=0$$

ここは大事！

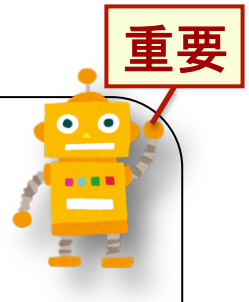
実際に、ユークリッドのGCDアルゴリウムの時間計算量を求めてみよう

計算量

アルゴリズムの計算量

計算量を問題例の入力長Nの関数としてオーダー評価したもの

以下の2つの評価法がある。



最悪計算量（worst case complexity）

入力長がNである問題例の中で最大の計算量

平均計算量(average case complexity)

入力長がNである問題例の計算量の期待値

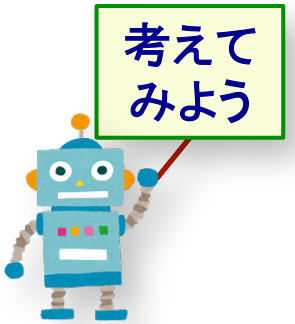
（例） 入力長がNの問題例に対し確率(N-1)/NでO(N)、確率1/NでO(N^2)であるようなアルゴリズムの計算量

最悪時間計算量 O(N^2)　　平均時間計算量 O(N)

通常は，最悪計算量で評価することが多い．

GCDを解くのにどちらがよいアルゴリズム(1/2)？

入力： a_0, a_1 $(a_0 \geqq a_1)$　　入力長はN = $O(\log a_0)$



(a)単純なアルゴリズム

Step 1　n←a_1

Step 2　nがa_0とa_1の約数であれば
　　nを出力して停止

Step 3　n←n-1としてStep 2へ

⇔

(b) ユークリッドの互除法

Step 1　a←a_0, b←a_1

Step 2　r←aをbで割った余り

Step 3　r=0ならばbを出力して停止

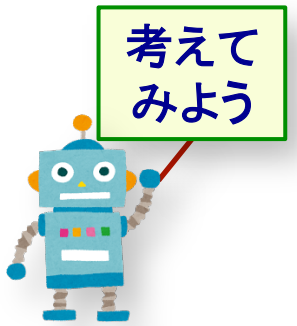
Step 4　a←b, b←rとしてStep 2へ

どっちが速い？

GCDを解くのにどちらがよいアルゴリズム(1/2)？

入力： a_0, a_1 $(a_0 \geqq a_1)$　　入力長はN = O(loga_0)

(a)単純なアルゴリズム
Step 1　n←a_1
Step 2　nがa_0とa_1の約数であれば
　　nを出力して停止
Step 3　n←n-1としてStep 2へ



単純なアルゴリズムの計算量

(1) 最悪の場合　ループの回数はa_1($\leqq a_0$)
　これは入力長N=log(a_0)の指数

(2) ループ内において割り算がO(log$^2a_0$)
ステップ、その他の部分はO(loga_0)
ステップ。全体ではO($a_0$log$^2a_0$)ステップ。

(3) したがって入力長をNとすれば⇐　指数時間アルゴリズム
最悪時間計算量はO($N^2 2^N$)。

GCDを解くのにどちらがよいアルゴリズム(2/2)？

入力： a_0, a_1 $(a_0 \geqq a_1)$　　入力長は$O(\log a_0)$

発展

(b) ユークリッドの互除法

Step 1　$a \leftarrow a_0$, $b \leftarrow a_1$

Step 2　rをaをbで割った余り

Step 3　r=0ならばbを出力して停止

Step 4　$a \leftarrow b$, $b \leftarrow r$としてStep 2へ

ユークリッドの互除法の計算量

ループを回るごとに、次のような計算を順次して、aの値が小さくなり、最後に$a_{k+1}=0$となる。

$$a_0=q_1a_1+a_2,$$
$$a_1=q_2a_2+a_3,\ldots,$$
$$a_{k-1}=q_ka_k+a_{k+1}$$

$a_0-a_2=q_1a_1 \geqq a_1>a_2$であるから$a_2<a_0/2$。

観察：同様に、任意のiに対して、$a_{i+2}<\lceil a_i/2 \rceil$。つまり、ループを2回まわるごとにaの値が半分以下になる。

したがって、$a_{k+1}=0$ になるまでk回ループが回ると、$a_{2k}<\lceil a_0/2^k \rceil$ (1式)。

ここで右の計算より、最悪の場合でも、ループの回数は$O(\log a_0)$回。各ループ内において、割り算が$O(\log^2 a_0)$ステップ、その他の部分は$O(\log a_0)$ステップ。

よって全体では$O(\log^3 a_0)$ステップ。 ⇦ 多項式時間アルゴリズム

こちらがずっと速い！

ループ回数のオーダーの導出計算

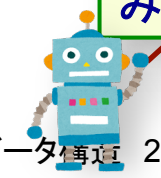
$0 = a_{2k} \leqq a_0/2^k$　（(1)式より）

→ $\log_2 0 \leqq \log_2 a_0 - k \log_2 2$
（両辺の$\log_2$をとる）

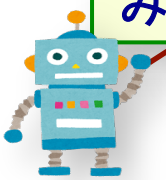
→ $k+1 \leqq \log_2 a_0$　(整理する)

→ $k = O(\log_2 a_0)$　(オーダー)

考えてみよう



演習（発展問題）



任意の$a,b>0$に対し
$\lim_{n\to\infty} \frac{\log^a n}{n^b} = 0$
であるから十分大きなnに対しては、
常に $\log^a n < n^b$ である。

(5) つぎのオーダー表記を簡略化せよ。

(a) $O(n\log n+n^2)+O(n^{1.83}\log n)$

明らかに$n\log n<n^2$, $n\log n<n^{1.83}\log n$である。
また、

$$\lim_{n\to\infty} \frac{n^{1.83}\log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n^{0.17}} = \lim_{n\to\infty} \frac{1/n}{0.17n^{-0.83}}$$

$$= \lim_{n\to\infty} \frac{1}{0.17n^{0.17}} = 0$$

慣れてきたら
この知識から
直接結論付けても
OK

よって十分大きなnに対して$n^{1.83}\log n<n^2$であるから

$O(n\log n+n^2)+O(n^{1.83}\log n)=O(n^2)$

(b) $O(n^{\log n}+n^{100}+n^{30}\log n)$

$n^{30}\log n<n^{100}$は明らか。
また十分大きなnに対して$n^{100}<n^{\log n}$が成り立つ。
よって

$O(n^{\log n}+n^{100}+n^{30}\log n)=O(n^{\log n})$

(c) $O(n^3\sin^2 n)O(2^n/\log n)$

$\sin^2 n \leqq 1$であるから

$O(n^3\sin^2 n)O(2^n/\log n)$
$=O(n^3 2^n/\log n)$

第2回アルゴリズムと計算量

- 今日の内容
  - アルゴリズムの計算量評価
  - 漸近的計算量
  - オーダーの計算の方法
  - 最悪計算量と平均計算量
  - 例）GCDアルゴリズムの時間計算量の計算

- ポイント
  - オーダー記法
  - ビッグオー，ビッグオメガ，ビッグシータ