

オブジェクト指向 プログラミング

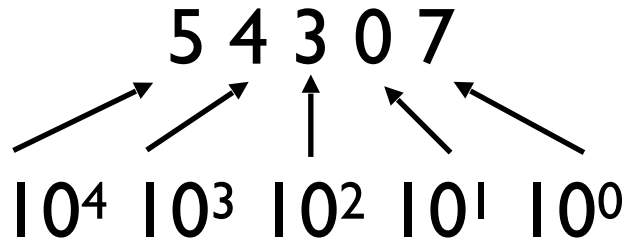
第7回
箕原辰夫

整数のアルゴリズムと演算

- n進数との変換
- 約数や倍数
- 階差方程式
- 整数のビット演算子

n進数と位取り法

- それぞれの桁が基数のべき乗との掛け算である。



$$5 \times 10000$$

$$4 \times 1000$$

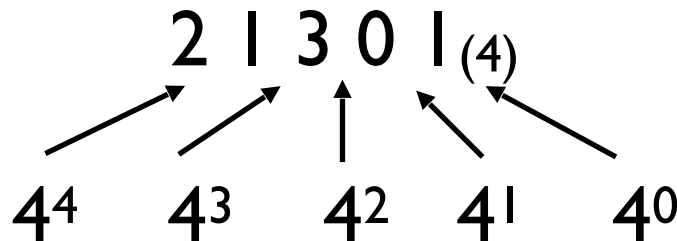
$$3 \times 100$$

$$0 \times 10$$

$$7 \times 1$$

n進数の値を求める

- 各桁をnのべき乗数と掛け算し、それらの総和を求める



Question 1:
12031₍₄₎

$$\begin{array}{r} 2 \times 256 \\ 1 \times 64 \\ 3 \times 16 \\ 0 \times 4 \\ + \quad 1 \times 1 \\ \hline \end{array} = 625_{(10)}$$

Question 2:
356₍₇₎

2進数と16進数

- 2進数を最下位（右側）の桁から4桁ずつ区切って読むと16進数になる。

→ 11101010010101
→ 11/1010/1001/0101
→ 3 A 9 5

2進数	16進数	2進数	16進数
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

n進数への変換

- 基数のnで割っていく

検算してみる

4) 8473
4) 2118 ... 1
4) 529 ... 2
4) 132 ... 1
4) 33 ... 0
4) 8 ... 1
4) 2 ... 0
0 ... 2



2 0 1 0 1 2 1
↗ ↗ ↗ ↑ ↖ ↖ ↖
4⁶ 4⁵ 4⁴ 4³ 4² 4¹ 4⁰

2 x 4096
0 x 1024
1 x 256
0 x 64
1 x 16
2 x 4
1 x 1

Question 3:
9876を7進数で

n進数の分解

- 基数での整数除算と整数剰余
- 基数で割っていく + 基数で整数剰余を取る
 - それぞれの桁が求まる

1234567 ↓ $\frac{1}{10}$
123456 ↓ $\frac{1}{10}$
12345 ↓ $\frac{1}{10}$
1234 ↓ $\frac{1}{10}$
123 ↓ $\frac{1}{10}$
12 ↓ $\frac{1}{10}$
1 ↓ $\frac{1}{10}$
0 ↓ $\frac{1}{10}$

456 ↓ $\times 10$
4560 ↓ $\times 10$
45600 ↓ $\times 10$
456000 ↓ $\times 10$
4560000 ↓ $\times 10$
45600000 ↓ $\times 10$
456000000 ↓ $\times 10$
4560000000 ↓ $\times 10$

基数と整数剰余・整数除算

- 各桁に分解できる (**div**は、Pythonの場合、`//`)
 - $3456 \% 10 = 6$
 - $3456 \text{ div } 10 \% 10 = 5$
 - $3456 \text{ div } 10 \text{ div } 10 \% 10 = 4$
 - $3456 \text{ div } 10 \text{ div } 10 \text{ div } 10 \% 10 = 3$
- n進数においても同じ
 - $39 \% 5 = 4$
 - $39 \text{ div } 5 \% 5 = 2$
 - $39 \text{ div } 5 \text{ div } 5 \% 5 = 1$

n進数の必要性

- 2進数、16進数はコンピュータで基本的に使われている
- 7進数は暦の上で曜日（週）との関連性が強い
- 12進数は、時間や月などで用いられている
- 60進数は、時間（分や秒）で用いられている
- 360進数は、角度で用いられている
- n進数はこのように実は広く生活の中に浸透している。ただ、それをn進数として学んでこなかっただけ

各桁への分解

- 基数の10で割っていく

時分秒の場合は60と24

$$\begin{array}{r} 10 \overline{) 81473} \\ 10 \overline{) \underline{8147}} \dots 3 \\ 10 \overline{) \underline{14}} \dots 7 \\ 10 \overline{) \underline{1}} \dots 4 \\ 10 \overline{) \underline{}} \dots 1 \\ 0 \dots 8 \end{array}$$

$$\begin{array}{r} 60 \overline{) 81473} \\ 60 \overline{) \underline{1357}} \dots 53 \\ 24 \overline{) \underline{22}} \dots 37 \\ 365 \overline{) \underline{0}} \dots 22 \\ 0 \dots 0 \end{array}$$

秒
分
時
日

n進数変換

- n進数を10進数に変換する
 - n進数で表記された各桁をnのべき乗（最下位は0から始まる）と掛け算して、足し合わせる
 - 11進数以上は、アルファベットへの対応が必要
- 10進数をn進数に変換する
 - 数をnで割っていき（整数除算）、余りを下の桁から文字列として足していく
 - 商が0になった時点で終了

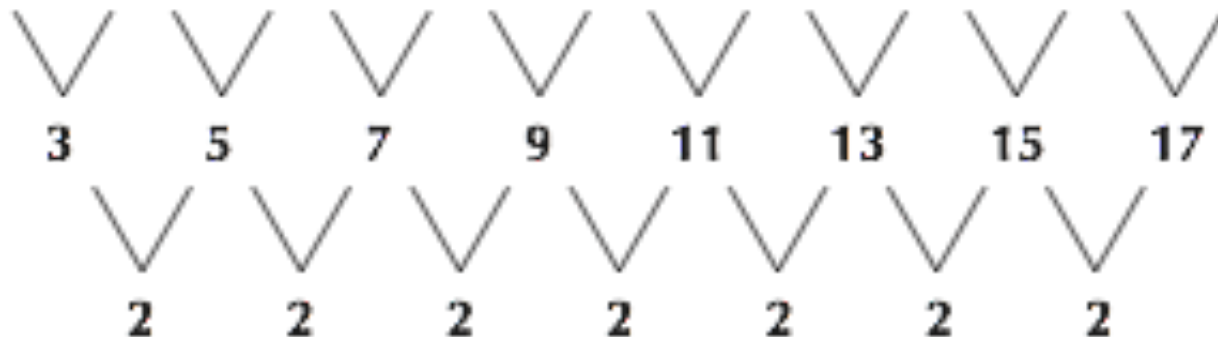
約数・倍数を求める

- 約数の個数を求める
 - 約数があるか数を数えるための変数を用意
- 約数の和を求める
 - 約数の和を求めるための変数を用意
- 倍数の個数を求める
 - 倍数があるか数を数えるための変数を用意
- 素数と完全数
- ピタゴラスの平方数を求める
- 素因数分解

バベッジの階差機関

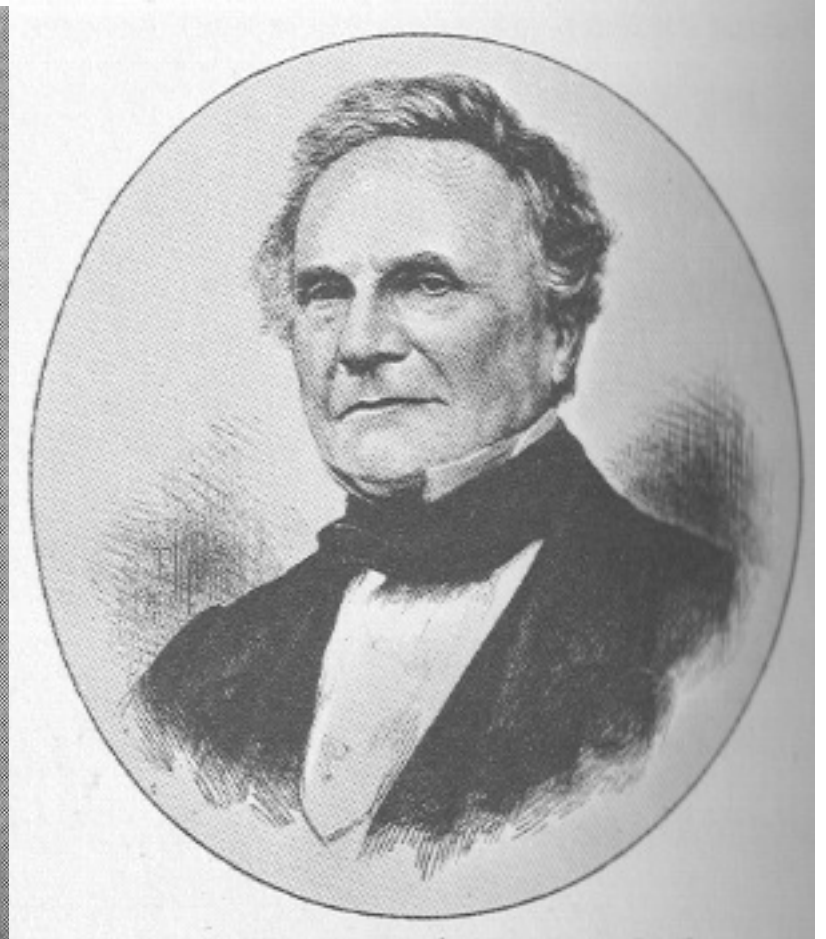
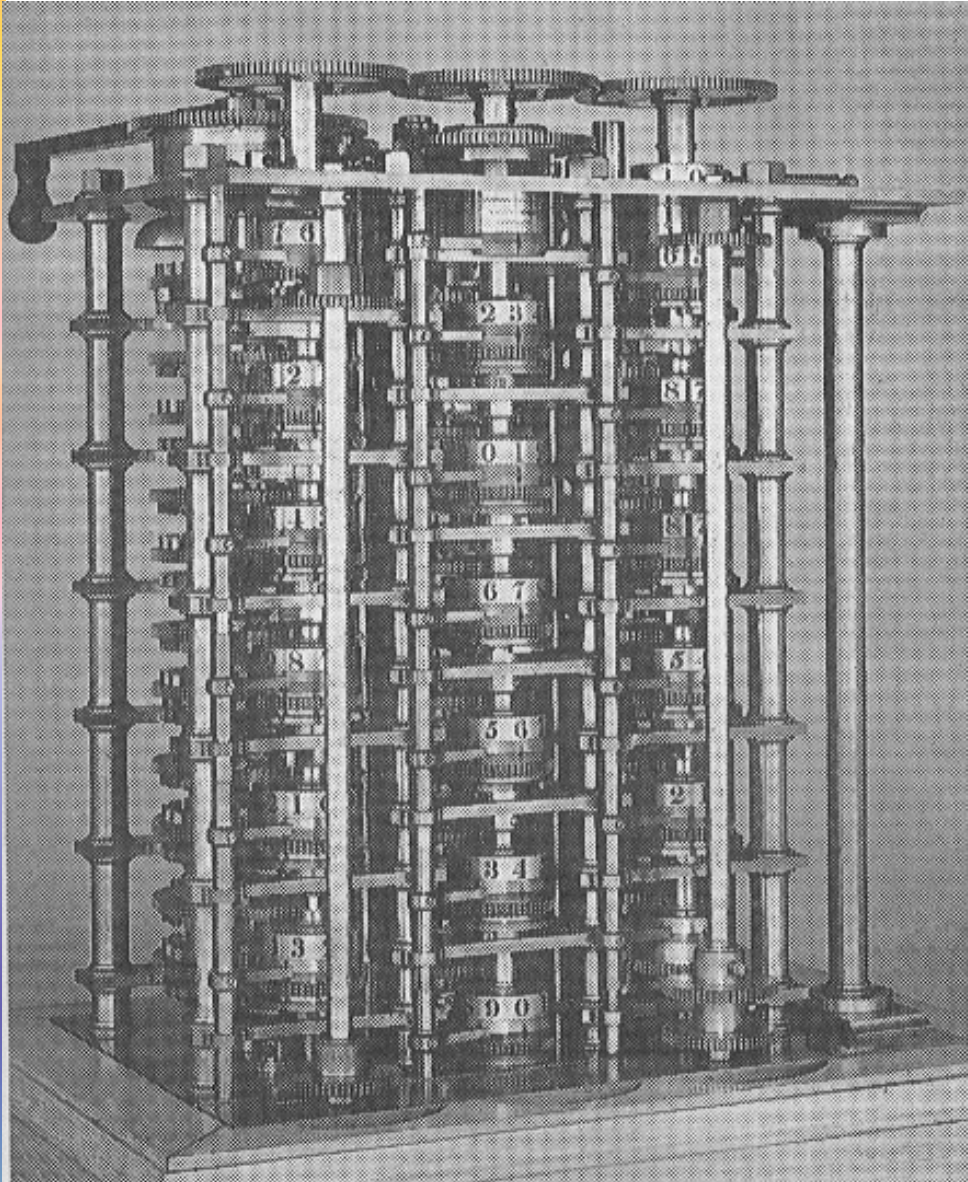
- 2次多項式に値を代入して計算値を求める
- 掛け算を足し算で計算していく方法

x :	1	2	3	4	5	6	7	8	9
x^2 :	1	4	9	16	25	36	49	64	81
第1階差:		3	5	7	9	11	13	15	17
第2階差:			2	2	2	2	2	2	2



The diagram illustrates the step-by-step calculation of a quadratic polynomial using Babbage's difference engine. It shows the initial values of x and x^2 , followed by the first differences (第1階差) and second differences (第2階差). The first differences are calculated by subtracting the previous x^2 value from the current one. The second differences are constant at 2, which is a characteristic of a quadratic polynomial. The diagram uses a series of downward-pointing chevrons to show the flow of calculations from the initial values to the final differences.

バベッジの階差機関



求め方

- 計算結果を求める変数
 - $result = result + \text{第1階差}$;
- 第1階差を求める変数
 - $diff = diff + \text{第2階差}$;
- 多項式が $px^2 + qx + r$ の場合、
 - 最初の結果は、 r
 - 最初の第1階差は、 $p + q$
 - 第2階差は、一定で、 $p + p (=2p)$

ビット演算

- `~` ビットの反転（単項演算子）
 - 例： `~x`
- `&` ビットのAND（二項演算子）
 - 例： `x & 7`
- `|` ビットのOR（二項演算子）
 - 例： `x | 5`
- `^` ビットの排他的論理和（二項演算子）
 - 例： `x ^ 0b1101`

8 bitで演算

- 0xa3 と ~0xa3

→ 0b10100011 各ビットを反転する(Pythonでは-(n+1))

↓↓↓↓↓↓↓↓

→ 0b01011100

- 0xa3 & 0x7a

→ 0b10100011 0xa3

→ 0b01111010 0x7a

→ 0b00100010 0x22 両方とも1のビットだけ1

- 0xa3 | 0x7a

→ 0b10100011 0xa3

→ 0b01111010 0x7a

→ 0b11111011 0xfb いずれかのビットが1なら1

ビット演算続き

- $0xa3 \wedge 0x7a$
 - `0b10100011` $0xa3$
 - `0b01111010` $0x7a$
 - `0b11011001` $0xd9$ どちらかのビットが1なら1

A	B	A&B	A B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

ビット演算を目に見える形で

- 入力された 2 つの数をそれぞれ 2 進数で表示する
- 入力された文字列の 2 進数を整数に変換するには、`int( 文字列, base=2 )`を用いる
- 整数を 2 進数で表示するには、`bin( )`関数を用いる

シフト算

- 2 のべき乗とのかけ算・割り算になる

→ `>>` 右シフト 2 のべき乗で割った

例： `12 >> 2` $\equiv$ `12 // (2*2)`

→ `<<` 左シフト 2 のべき乗と掛けた

例： `3 << 4` $\equiv$ `3 * (2 * 2 * 2 * 2)`

format関数とzfill関数

- format関数
 - format(数値, 書式文字列)
 - "b"... 2進数
 - "d"... 10進数
 - "o"... 8進数
 - "x"... 16進数
- 文字列のzfill関数
 - 文字列.zfill(0で埋める桁数)

2進数の表示

- `format( 式, "b" ).zfill( 桁数 )`
 - 式の値を、桁数分の0で埋めてから、2進数を表示する
- 表示する有効の下位の桁数を指定したいときは、
「値 & 0b1111」などを使う
- 例：`format( ~(0b101) & 0xff, "b" ).zfill( 8 )`
 - 下位 8 桁だけを表示する

補数

- 足してその数になる数の組み合わせ
- 10の補数
 - 足して10になる数の組み合わせ

例： 1と9, 2と8, 3と7, 4と6, 5と5

6に対する10の補数は？ … 4

9の補数 0と9, 1と8, 2と7, 3と6, 4と5

補数

- 補数は引き算の代わりに用いられる
- 基数の補数 (2の補数、10の補数)
- 基数-1の補数 (1の補数、9の補数)
- 基数の補数 = 基数-1の補数 + 1
- 例 : $10000 - 5678 = ?$
 - $9999 - 5678 + 1 = 4321 + 1 = 4322$
 - $15678 - 9876 = 5678 + 10000 - 9876$
 - $= 5678 + 9999 - 9876 + 1 = 5678 + 123 + 1$
 - $= 5802$

2進数では？

- $10010 - 01101 = 18 - 13$
 - $= 10010 + \sim(01101) + 1$
 - $= 10010 + 10010 + 1$
 - $= 100100 + 1 = 100101 = 00101 = 5$
- $00000 - 01101 = 0 - 13$
 - $= \sim(01101) + 1 = 10010 + 1 = 10011$