第5章 計算とプログラム

本章の目的

- モデル化した問題を”計算”して解く
- 計算
  - モデル化した問題に対する操作
- 本章で説明すること
  - 計算の概観と記述法
  - 代表的な計算モデル
  - プログラムとプログラム言語

計算とその記述方法

- 計算の例：計数(counting)
  - ある集合Aの要素数を求める
- 計算のやり方
  - Aのデータモデルとして，どのような処理が用意されているか，に依存
- 計数のやり方の種類
  - 取り出し型
    - 要素を1つ1つ，指折り数えていくやり方
  - 分割型
    - 自分の手に余る仕事を下請けに出していくやり方

計算の方法 – 取り出し型

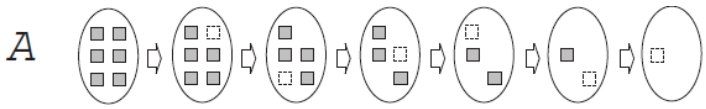
- 集合に対して用意されている処理
  - 空（要素が1つもない）かどうかを判定する
  - 要素を1つ取り出す（集合の要素数は1だけ減る）
- 計算
  - <答>をゼロにする
  - A が空でないあいだ，以下の処理を繰り返す
    - 要素を1つ取り出す
    - <答>を1増やす

計算の方法 – 分割型

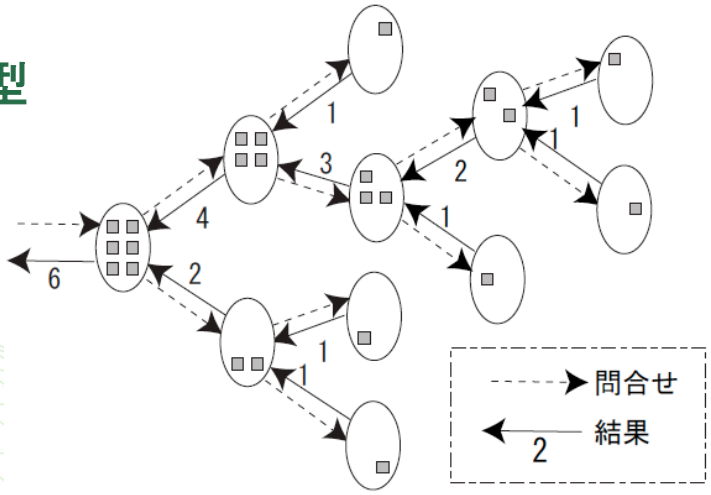
- 集合に対して用意されている処理
  - 空か，要素が1つだけであるかを判定する
  - 空でない2つの集合に分割する
- 計算
  - Aが空なら<答>は0，要素数が1なら<答>は1
  - そうでなければ
    - AをBとCに分割（BもCも空集合ではない）
    - <答> ＝ Bの要素数 ＋ Cの要素数

計算の方法の図

- 取り出し型



- 分割型



計算の記述

問題

- 今年の八十八夜(立春から数えて88日目)は何月何日か．ただし今年の立春は2月4日であり，今年は平年である．

考え方

- 2月4日の87日後は"2月をはみ出す" → 2月の残り日数(28 − 4 = 24日)を引く(87 − 24 = 63日)
- 3月63日は3月を越す → 31日を引く(63 − 31 = 32日)
- 4月32日は4月を越す → 30日を引く(32 − 30 = 2日)
- 5月2日は5月に収まる！→ 最終的な答えは5月2日

計算の記述 − より明確に

2月4日の87日後

- <残り日数> = 4 + 87 → 2月91日

91 > 28(2月の日数)

- <残り日数> = 91 − 2月の日数 → 3月63日

63 > 31(3月の日数)

- <残り日数> = 63 − 3月の日数 → 4月32日

32 > 30(4月の日数)

- <残り日数> = 32 − 4月の日数 → 5月2日

2 < 31(5月の日数)なので計算終了

計算を記述するために必要な要素

- **変数(variable)**
  - 値(例:残り日数)を覚えておく"もの"
  - 変数の値は"代入"により変化させることができる
  - 代入(assignment)
    - 変数に値を設定する操作，"変数名" ← "式" で表す
- **条件付き処理の操作**

if 条件
　then "条件が成立した場合に行なう処理"
　else "条件が成立しない場合に行なう処理"
endif

八十八夜問題の解法手順

<残り日数> ← 4+87
if <残り日数> ＞ 28(2月の日数)
then <残り日数> ← <残り日数> - 28
　if <残り日数> ＞ 31(3月の日数)
　then <残り日数> ← <残り日数> - 31
　　if <残り日数> ＞ 30(4月の日数)
　　then <残り日数> ← <残り日数> - 30
　　　if <残り日数> ＞ 31(5月の日数)
　　　then (6月以降の処理)
　　　else "5月"<残り日数>"日"と表示
　　　endif
　　else "4月"<残り日数>"日"と表示
　　endif
　else "3月"<残り日数>"日"と表示
　endif
else "2月"<残り日数>"日"と表示
endif

解法手順の改良

- 6月以降については"(6月以降の処理)"としか書いていない
  - 実際に6月以降の処理が必要になったときに正しい答えが求まる保証がない
  - もっとすっきりと，12月まで対処できる方法は？
- ”m(n月の日数)”という表現が何度も現れている
  - 例：28(2月の日数)
  - もっとすっきりと記述する方法は？

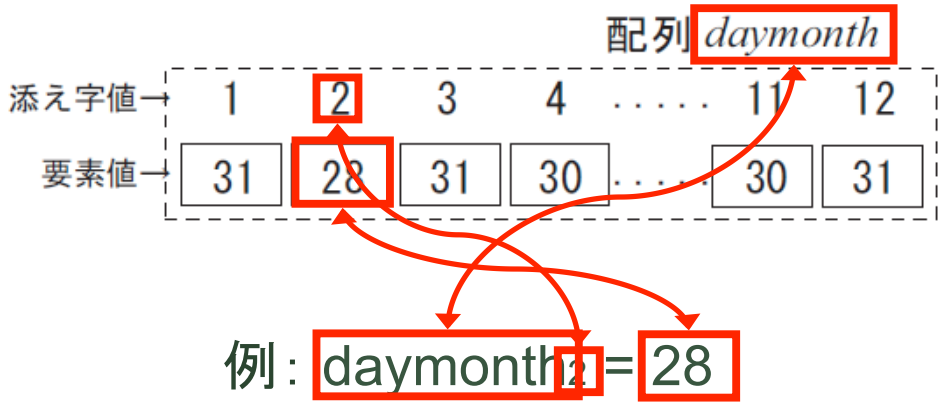
反復処理と配列

- 反復処理(repetitive processing)
  - 条件が成立している限り”処理”を繰り返し実行する

```
while 条件 do
    ”処理”
done
```

- 配列(array)
  - 要素の集合から，”添え字”を使って値を取り出し，変数として扱うことができる
  - 要素全体をまとめて扱うことができる

配列の例



配列名 = daymonth
添え字 = 2
値 = 28

八十八夜問題の解法手順 – 改良版

<残り日数> ← 4+87

m ← 2

while <残り日数> ＞ daymonth$_m$ do

<残り日数> ← <残り日数> - daymonth$_m$

m ← m + 1

done

- ”<残り日数>と月の日数の比較を繰り返し行う”手順を，”反復処理”と”配列”を使うことですっきりと記述することができた

第6章 問題の解決

この章のねらい

- モデルの記述（第4章）
- モデル上の操作による計算とプログラム（第5章）
- 本章
  - アルゴリズム：計算手順
    - よいアルゴリズムとよくないアルゴリズムがある
  - 計算のモデル化
    - 有限状態機械，チューリング機械
  - 計算量と計算可能性

ハノイの塔問題

発展

円盤の山を左から中央に移すのにかかる時間は?

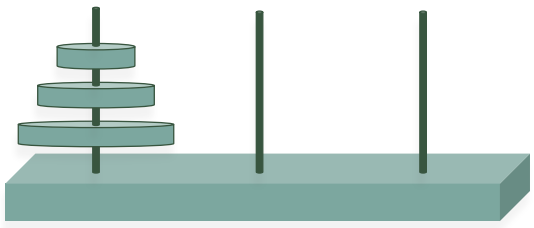
- n: 円盤の枚数
- 1枚動かすのに10^{-9}秒かかるとする (cf. 1GHz)
- n=100のときにかかる時間は?

  A: 約5ミリ秒

  B: 約50秒

  C: 約50億年

  D: 約50兆年



- ヒント: n=20のとき約1ミリ秒

配布不可

コンピュータによる問題解決

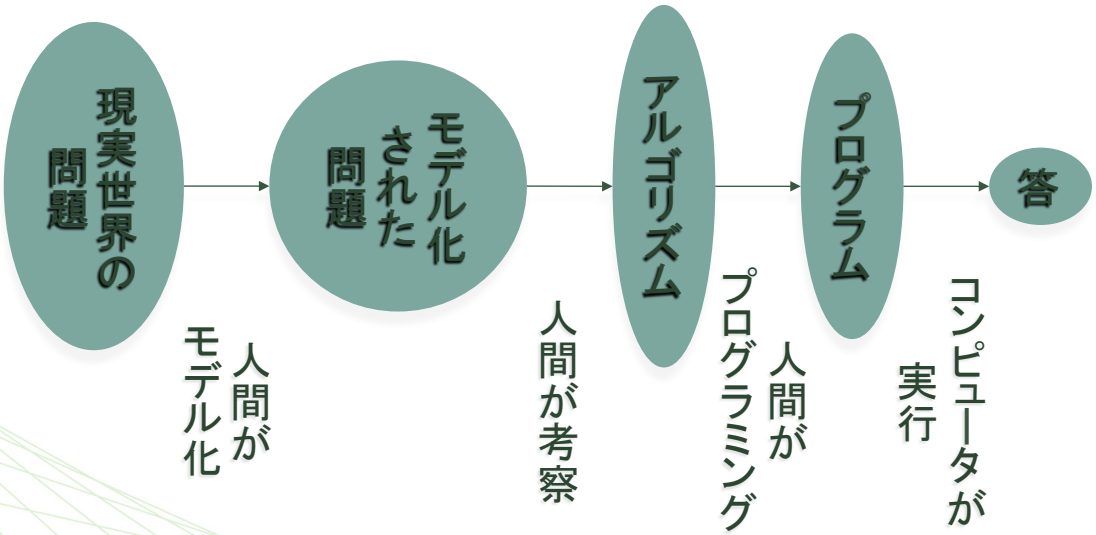
A/B

問題解決の手順

1. 問題の明確化（モデル化）（目的、入力と出力が何かをはっきりさせる。）
2. 問題の解き方(アルゴリズム)を考える
3. アルゴリズムを計算機に実行させるために、プログラミング言語でプログラムを作る。
4. プログラムを実行する

コンピュータによる問題解決：実際

↘ 問題解決の手順



アルゴリズムの重要性

- 性能に大きな違いが出る
  - 同じ問題を解く複数のアルゴリズムがある
  - アルゴリズムによって計算時間が桁違いに変わることがある
- 類型化されている
  - 全く違う問題を解くアルゴリズムが同じものになることがある
  - 性能に関する考察・プログラミングを共通化できる

アルゴリズムの実例

- 目的
  - 「アルゴリズム」がどのようなものかを具体的な問題について知る
  - 同じ問題について複数のアルゴリズムを見て、計算時間が変わることを知る
- 紹介される例:

問題	平方根の計算	フィボナッチ数の計算
アルゴリズム	反復法 二分法	再帰法 メモ化法

問題: 平方根の計算

- $\sqrt{x}$ を求める
- 注意:
  - 小数の計算は有限の精度で行われる
  - →近似値しか求められない
- 問題:
  - ある正の実数 x が与えられたときに、2乗すると x に近くなる正の実数 y を精度 d で求める.
  - つまり, $|\sqrt{x}-y|<\delta$ となるような y を 1 つ求める

平方根のアルゴリズム: 反復法

- x = 90, d = 1 の場合を考える
- 「y = 0, 1, 2, 3, ... を順に検討してゆき $(y+d)^2$ が 90 より大きくなったら、その1つ前が解」

アルゴリズム1
(反復による
平方根の計算)

$y \leftarrow 0$
while $(y+\delta)^2 < x$ **do**
  $y \leftarrow y+\delta$
done
return y

- 実際の動作 x =2, d =0.0001 の場合、

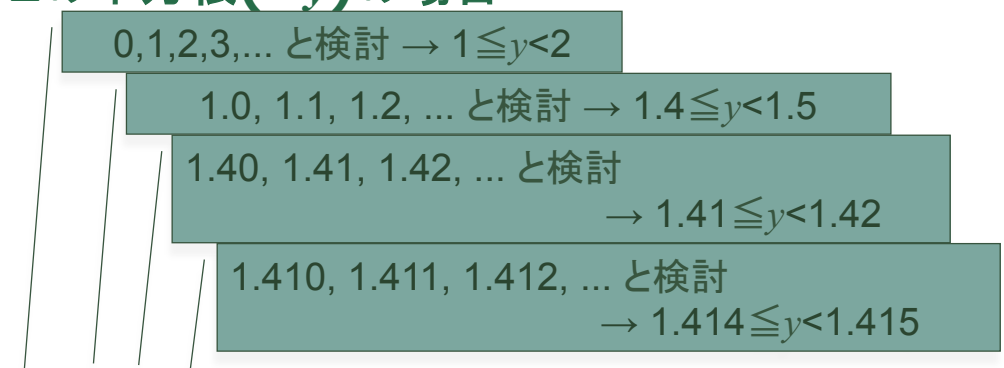
回数	0	1	2	···	14140	14141	14142
候補 (y)	0.0000	0.0001	0.0002	···	1.4140	1.4141	1.4142
$(y+\delta)^2$	0.00000	0.00000	0.00000	···	1.99968	1.99996	2.00024

アルゴリズムの速度

- 繰り返しの回数で比べる
  - プログラムの実行時間の非常におおざっぱな近似
  - 実際のコンピュータの性能と無関係に検討できる
  - 異なる種類の計算の速度差も無視してしまう
- 反復法の場合:
  - 繰り返しの回数は約 $\sqrt{x}/\delta$ 回
  - 精度を1桁増やすと回数も10倍に増える

平方根のアルゴリズム: 二分法の考え方

- アイデア: 1桁ずつ求めてゆく
- 例: 2の平方根(=y)の場合



1. 4 1 4 2 1 3 5 6 ...

- 特徴: 解がある範囲を1/10ずつ狭めてゆく
- 単純化: →二分法 (次スライド)

平方根のアルゴリズム: 二分法

アルゴリズム2 (二分法による平方根の計算)
x の平方根を精度 d で求める (ただし $x > 1$):

$a \leftarrow 0$

$b \leftarrow x$

while $b - a > \delta$ **do**

　　$c \leftarrow \frac{a+b}{2}$

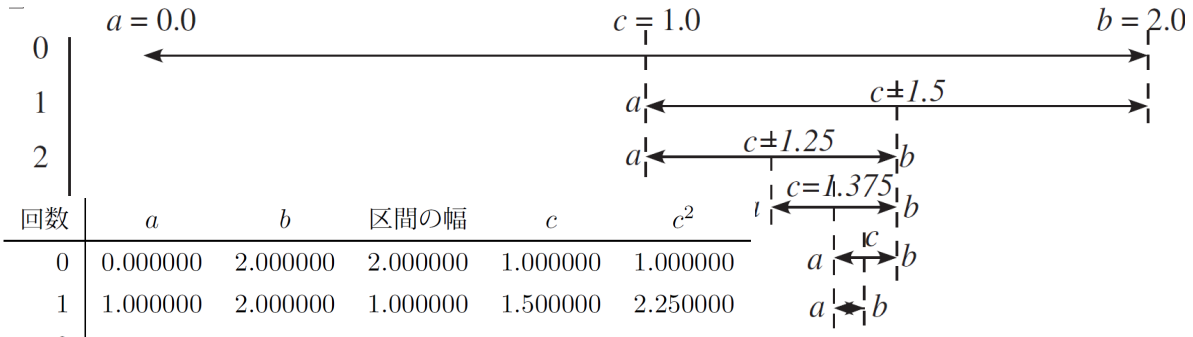
　　if $c^2 > x$ **then** $b \leftarrow c$ **else** $a \leftarrow c$ **endif**

done

return a

平方根のアルゴリズム: 二分法の実際

「区間の幅」が1/2ずつ減ってゆく



回数	a	b	区間の幅	c	c^2
0	0.000000	2.000000	2.000000	1.000000	1.000000
1	1.000000	2.000000	1.000000	1.500000	2.250000
2	1.000000	1.500000	0.500000	1.250000	1.562500
3	1.250000	1.500000	0.250000	1.375000	1.890625
4	1.375000	1.500000	0.125000	1.437500	2.066406
			(中略)		
13	1.414062	1.414307	0.000244	1.414185	1.999918
14	1.414185	1.414307	0.000122	1.414246	2.000091
15	1.414185	1.414246	0.000061		

アルゴリズムの速度

- **反復法: 約 $\sqrt{x}/\delta$ 回**
- **二分法:**
  - 1回繰り返すごとに区間の幅が1/2になる
  - *n*回繰り返し後の区間の幅は $x/2^n$
  - これが *d*以下になるのに要する回数
    → 約 $\log_2 \frac{x}{\delta}$ 回
- **比較: *x*=2, *d*=0.0000000001 のとき**
  - 反復法: 約141億回
  - 二分法: 35回

小数点以下
10桁まで求める