informatical reality

2013 winter

Ishotihadus

序

informatical reality

本書は，東京大学に入学された者のうち，我々が推定もできないような妙な興味を持った文系生たちや，準必修の名前に騙されてしまった理系生たちに待ち受ける，情報科学という闇教科のクソ期末試験に対する，根本の基礎知識から上辺のテクニックまですべてをなるべく効率よく詰め込むためのシケプリである．

本書は 4 部構成になっている．

第 I 部は，情報科学の授業内で学ぶはずだったことのおさらい．一応試験に出る範囲のものだけを集めたつもりである．もちろん全部を読み切る必要はない．わからなさそうな範囲だけ理解すればいいだけのことである．

第 II 部・第 III 部は過去問およびその解答解説．各節の最初の易しめの 3〜4 問を解いておけば十分だろう．オートマトンなど，旧期末試験にしか出ていない問題も一応掲載している．

第 IV 部は付録．参考文献リストおよび索引を掲載した．索引は日本語と英語の両方を作ってある．必要に応じて開いてもらいたい．

全部で 100 ページを超える分量になってしまったが，問題集とその解答が半分以上を占めているので，実質量はそこまで多くない．自分の能力に合わせて使ってほしい．

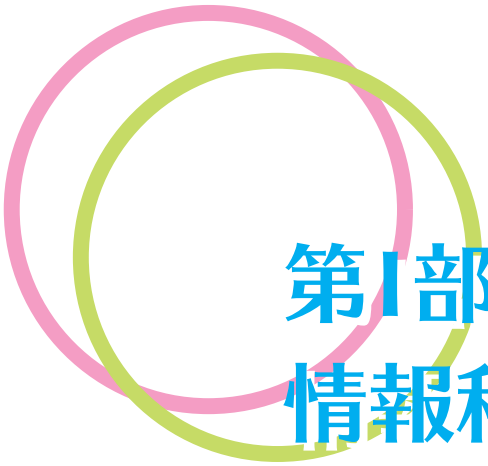
情報科学の講義は基本的に演習中心であるから，期末試験の重みは低い（はずである）．しかし高得点を狙うためには期末試験も手を抜いてはならない．基本の授業の演習を適度にこなしさらに高得点を狙うもの，授業の演習があまり捗らなかったために期末試験で点数稼ぎを狙うものは，ぜひ本書を最後まで読み切り，良い成果を残していってほしい．

最後に，情報科学の教科書を書かれた増原英彦先生，我々のクラスの授業担当である田中哲朗先生，ところどころ意見をくださり質問に請け合ってくださった須田礼仁先生，twitter で絡みつつヒントを与えてくださった久野靖先生，および同クラや twitter のフォロワー等々そのほか数々の周りの不思議な方たちに，心から感謝いたします．

2014 年 1 月 30 日
イショティハドゥス

目次

informatical reality



第I部
情報科学 総おさらい

この部では，授業で扱った情報科学の諸概念を最初から復習していく．基本的には教科書の第 5 章〜第 8 章を中心に，適宜 Ruby のコードをまじえながら各概念を説明していく．ただし，最初の章のみは（この情報科学で扱う）プログラミングにおける基本事項を説明する．これは「まだプログラムのコードがよくわからん」という人のためのものであるから，コードがある程度読める人は読み飛ばして構わない．

第 1 章　基本事項の復習

informatical reality

ここではプログラミングにおける基礎知識をマンガをところどころいれながらざっと解説する．ここから躓かれてはかなり「[「ヤバい」]」状況だが，まあ大丈夫でしょ．

1.1　irb について

irb（Interactive Ruby）とは情報科学の授業で用いた Ruby のメソッドや式などを簡単に実行できるスグレモノ．こちらが 1 つ文を発せば相手（Ruby）が 1 つ答えを返してくれるという対話型のプログラムであるので，初級者には扱いやすい（がところどころ困ったところもある）．

実行すると下みたいな感じになる．

```
irb(main):001:0> 35+7
=> 42
irb(main):002:0> 5**9
=> 1953125
irb(main):003:0> (true && false) || (!true || false)
=> false
```

まあ irb に関してはこれだけで十分．

1.2　使える計算式

計算式にでてくる + とか – とかのことを**演算子**という．演算子には以下のものが使える．

演算子	名前	例
+	加算	3+5 → 8
-	減算	3-5 → -2
*	乗算	3*5 → 15
/	除算	3/5 → 0, 3.0/5 → 0.6
%	剰余	3%5 → 3
**	べき乗	3**5 → 243

3/5 が 0 になって，3.0/5 が 0.6 になるのは，プログラミング言語がそういう仕様だからである．整数どうしで計算するときには答えも整数で，どちらか一方でも小数であれば答えも実数で返すようにできている．なお，複雑な計算式を扱いたいときは ( ) も使える．


対話型のirb
5+3は？
8です
$\frac{\log_8(\pi)}{\cos^2(\frac{\pi}{5}\pi)}$ は？
5.206…です
わたしってかわいい？
てへぺろ☆
ちょっとわかりません……
変数を作りたい
今日は変数をつくろう！
どんな思いつきよ…
ていうかなんだそれ
変数「みかん」にあーちゃんを代入します。
みかん
私は捨て猫か。
変数「私のカレシ♡」にクラスのツキヤマくんを代入します♡
HAHAHA
私のカレシ♡
ツキヤマくんかっこいい！
なつの彼氏はnilでしょ
ほえええん（泣）

1.3 変数

プログラムの中では様々な数字や文字が行き交う．そのデータ（値）を格納しておく箱を変数（variable）という．1 人 1 人名前があって区別されるように，変数にも 1 つ 1 つ名前があって区別される[*1]．

変数は以下 3 つの操作を組み合わせることで利用している．それぞれの操作に名前があって，しかも普通に使われるので，知らなかった人は把握しておこう．

1.3.1 宣言

新たな変数に名前をつけること[*2]．Ruby の場合は仕様上代入も同時に行う必要がある．

```
a = 3 # 3 の入っている a という変数を宣言した
```

1.3.2 代入

変数の箱に新たなデータを入れること[*3]．はじめて代入するときは特に「初期化する」という．Ruby では宣言の際にかならず初期化する必要がある．なお，1 つの変数には 1 つのデータしか入れることができない．ちなみに「何も入っていない」を意味する nil を代入することもできる[*4]．

```
a = 55 # 変数 a に 55 を代入した
```

1.3.3 参照

変数の箱にどんなデータが入っているかを読み取ること．

```
irb(main):007:0> a
=> 55 # a は 55 だと教えてくれた
```

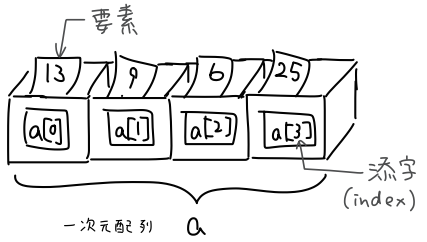
[*1] 無名の変数を作ることもできる．初期化のみを行って変数に代入しなかった場合，一応無名の変数として扱われてメモリ上（プログラムによっては CPU 上）にスタックされる．もちろん使われることがないからすぐに破棄される（はずである）．これは例えば `int length = str.split('\n').length();` などといった形で用いることができる．あとは，c++ などにある無名引数も無名の変数として扱えるだろう．なおこれは筆者独自の解釈である．

[*2] 宣言時に型も指定する必要があるが，Ruby では型を推論してしまうのでその必要はない．逆に型を推論してしまうから最初に初期値が必要になる．しかし Ruby はコロコロと型を変えられるので，あまり気にする必要がない．

[*3] 正確には「データの入っているメモリに紐付けすること」となる．無論データはメモリに格納してあるが，そのメモリの格納してある位置をポインタに，そのデータを該当するメモリに格納することを代入という簡略化した操作で実現している．これは初心者にはラクなことだが，上級者になるとそうでもなくて……?

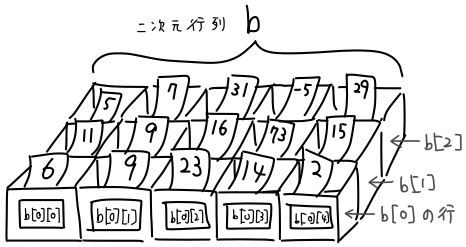
[*4] nil を代入するとその変数は NilClass 型の変数になるため，どんな型の変数にも代入できる．通常は string であれば string 型の変数に nil を格納することになるため，int や boolean などの変数には代入できないのが普通である．なお null とする言語も多数ある．

1.4 配列



いくつかの変数にそれぞれ番号がついて並んだものを収納する変数．番号のことを添字（インデックス / index）という．なお番号は普通 0 から始まる．

配列の個々の中身を要素（element）という[*5]．右図のように 1 列に箱が並んでいて，それぞれの名前が「配列名 [番号]」になっている．



配列は 1 列じゃなくてもいい．たとえば右図の下側のように，縦×横の配列を作ることもできる[*6][*7]．こうすれば，たとえば画像処理なんかに使いやすいデータになる．配列の方向の数を次元（dimension）という．この場合は 2 次元の配列である．同様に 3 次元，4 次元と何次元にも拡張できる．

```
irb(main):009:0> a = [0,5,3,7,9,1] # 配列 a をこのように宣言する
=> [0, 5, 3, 7, 9, 1]
irb(main):010:0> a[3] # 配列 a の 3 番目の要素を参照する
=> 7
irb(main):011:0> a[4] = 2 # 配列 a の 4 番目の要素に 2 を代入する
=> 2
irb(main):012:0> a # 配列 a を参照する
=> [0, 5, 3, 7, 2, 1] # 4 番目の要素はたしかに 2 になっている
irb(main):013:0> b = [[6,9,23,14,2],[11,9,16,73,15],[5,7,31,-5,39]] # 右上図のように配列 b を定義する
=> [[6, 9, 23, 14, 2], [11, 9, 16, 73, 15], [5, 7, 31, -5, 39]]
irb(main):014:0> b[1] # b の 1 番目の要素を参照する（実はこれも配列）
=> [11, 9, 16, 73, 15]
irb(main):015:0> b[2][3] # b の 2 番目の要素である配列の 3 番目の要素を参照する
=> -5
```

[*5] 多くのプログラミング言語では要素はすべて同じ型である必要があるが，Ruby の場合はそれぞれ違う型でもよいのでその説明は省略した．

[*6] 実際には配列自体が複数の変数を内部に持てるオブジェクト（変数）で，ある配列の要素それぞれに配列が入っていることを 2 次元の配列であるというが，変数の箱が 1 次元では 1 列に，2 次元では複数列に，3 次元では立体的に並んでいる，という考え方をしたほうが普通にわかりやすいので，このような説明になっている．

[*7] 図の例では b[0]～b[3] まですべて同じ要素数になっているが，ある次元の要素数がそれぞれ異なる配列を作ることもできる．配列自体がオブジェクトであるという考え方からすれば当然のことである．ただ，多次元配列を配列のなかに配列が入っている，という形でなく定義するプログラミング言語もある．

1.5 条件分岐

もし～だったらどんな処理をする，とかいうのの総称．授業ではおもに if 文をよく使った．「if 分岐条件」の次の行に処理を書く．そうでなかった場合の処理を書きたい場合には else を使う．もしそうではなくてこうなら～という表現をしたい場合には elsif を使う．

Ruby には他にも unless 文（if 文の逆），case 文（同じ変数に関していろいろ議論するときにベンリ），さらには三項間演算子（?:）などがあるが省略している．

```
if month == 2 # month が 2 なら
   28
elsif month == 4 || month == 6 || month == 9 || month == 11 # そうでなくて～
   30
else # どれでもないなら
   31
end # end を書き忘れない
```

使える比較式

正確には条件演算子という．条件を比較する際に使える式．メジャーなものはこれら．返り値は true（真）か false（偽）で表される．

演算子	名前	例
==	等価	3 == 5 → false
!=	異なる	3 != 5 → true
<	より小さい	3 < 5 → true
<=	以下	3 <= 5 → true
=>	以上	3 >= 5 → false
>	より大きい	3 > 5 → true

また，条件式を 2 つつなげる（and とか or）際に使えるものはこちら．

演算子	名前	例
&&	かつ	3 == 5 && 3 < 5 → false
||	または	3 == 5 || 3 < 5 → true

なお，条件式の前に!をつけると否定を表すことができる（このとき条件式を括弧で囲むとわかりやすい）．

演算子	名前	例
!	否定	!(3 == 5) && 3 < 5 → true

1.6 反復処理

ある条件が成立しているうちは何度も実行する，という処理の総称．

授業では while 文を扱った．「while 条件式 ～ end」とすることで，条件式が成り立っている（true である）ときに処理を行って，成り立たなくなった（false になった）時にループを抜ける．

他に for 文をよく扱った．i を 1 から 10 まで順番に動かしてそれぞれ計算するみたいな．あれも反復処理に含まれる．実際には while 文の中に整数が保存してあって，それが 1 回処理されるごとに 1 つずつ増えていく処理をしているだけである．

Ruby には他にも until 文（while 文の逆）があるがこれも省略．

while 文の例

```
irb(main):019:0> str = "abcde" # 文字列 str を宣言し，"abcde"を代入する
=> "abcde"
irb(main):020:0> c = str.length # 変数 c を宣言し，文字列 str の長さを代入する
=> 5
irb(main):021:0> rts = "" # 文字列 rts を宣言し，何もない文字列を代入する
=> ""
irb(main):022:0> while c > 0 # c が 0 より大きいなら何度でも下の処理を行う
irb(main):023:1>  rts = rts + str[(c-1)..(c-1)] # 文字列 rts の後に str の c-1 文字目をくっつける
irb(main):024:1>  c = c - 1 # c を 1 減らす
irb(main):025:1> end
=> # nil
irb(main):026:0> rts
=> "edcba" # rts を参照すると str を逆向きにした文字が入っている
```

範囲指定

str[(c-1)..(c-1)] は，str の c-1 文字目を表している．範囲を指定するときは，整数 a と b を用いて a..b と表せる．点は 2 つ．

```
irb(main):027:0> str = "abcdefg" # 文字列 str をこのように宣言
=> "abcdefg"
irb(main):028:0> str[3..3] # 3 文字目から 3 文字目の文字をとってくる
=> "d"
irb(main):029:0> str[1..4] # 1 文字目から 4 文字目の文字をとってくる
=> "bcde"
irb(main):030:0> str[3] # 3 文字目をとってくる
```

```
=> 100 # 3 文字目（d）の文字コード（100）になってしまう*8
```

配列でも同様のことができる．

```
irb(main):031:0> a = [5,3,9,4,7] # 配列 a をこのように宣言する
=> [5, 3, 9, 4, 7]
irb(main):032:0> a[2..4] # 配列 a の 2 番目から 4 番目を抜き出した配列
=> [9, 4, 7]
```

for 文の例

```
irb(main):037:0> sum = 0 # 変数 sum を 0 として宣言する
=> 0
irb(main):038:0> for i in 1..10 # 1 から 10 まで i を動かしてそれぞれ処理する
irb(main):039:1>  sum = sum + i # sum に i を加える
irb(main):040:1> end
=> 1..10
irb(main):041:0> sum
=> 55 # sum を参照すると 1 から 10 までの和の 55 が入っている
```

for 文における範囲*

for 文中の `for i in 1..10` は，整数の無限配列の中の 1 から 10 までとってくるということを表している*9．だから，in のあとの範囲は配列でもよい．したがって，応用的だが下のような書き方もできる．

```
irb(main):042:0> sum = 0
=> 0
irb(main):043:0> for i in [1,3,5,7,9]
irb(main):044:1>  sum = sum + i
irb(main):045:1> end
=> [1, 3, 5, 7, 9]
irb(main):046:0> sum
=> 25
```

*8 これは処理系によって異なる．最新版の Ruby では `str[3]` としても`"d"`と返す．一応念のため `str[3..3]` としておくことをお勧めする．

*9 正確には Range 型の `1..10` を `to_a()` して，それを `0` から `length-1` まで回している，ということになる．

1.7 メソッド（関数）

授業では関数（function）と呼んでいるが，細かいことを気にしなければどちらでもいい[*10]．このプリントではメソッド（method）と書く．

いくつか（実際には 1 つでも 0 個でもかまわない）のデータをもらって 1 つの値を返すもの．数学の関数も同じだろう．もらうデータのことを引数（arguments）という．

プログラミング言語においては普通，「`paint()`」などというように，メソッドの名前のあとに括弧を付ける．括弧がないと変数と区別がつかないからね[*11]．メソッドを定義する際には括弧の中に必要に応じて引数を入れる（`paint(g)` などというように）．Ruby だと `def 関数名 (引数) 〜 end` で囲まれた部分がメソッドの定義だ．

関数のため返す値を書いておく必要がある．Ruby の場合は最後に書いた式が返す値となっている．

```
def abs(n) # abs という名前のメソッドを定義する．引数は 1 つ
    if n < 0 # もし n が 0 より小さいなら
        -n # -n を答えとして返す
    else # そうでないなら
        n # そのまま n を返す
    end
end
```

```
irb(main):035:0> abs(-5)
=> 5
irb(main):036:0> abs(3)
=> 3
irb(main):003:0> abs() # ちゃんと引数を書いてあげないと
ArgumentError:  wrong number of arguments calling ‘abs‘ (0 for 1) # エラーになる
```

[*10] この違いに関する議論は平行線をたどるし，いろいろな説もあるので省略する．

[*11] Ruby では引数なしのメソッドの場合，() を省略することができる．

第 2 章　計算量

informatical reality

この章では，アルゴリズムの計算量とは何かという基本的な事項について説明する．計算量は，アルゴリズムを評価する上で重要なものであり，情報科学の基礎基本である．試験でも直接聞かれることがあるが，意外と自力では求めにくいものでもあるので，しっかり把握しておこう．

2.1　計算量とは

アルゴリズムの計算量（computational complexity）とは，計算の手間がどのくらいになるかを，与えられたデータの関数として表すものである．

本来は，アルゴリズムにもとづいて作られたプログラムがどのくらい計算に時間がかかるかを表すべきであるが，それはプログラムの書き方や，実行するコンピュータによって異なる．そのためアルゴリズム本来の効率を議論するのには適さない．そこで，計算量という概念では，入力されたデータによって計算時間がどのくらい増えるかといった，**傾向**を表現する．計算量というのは，かかる計算時間の傾向を「なんとなく」表現したものなのである．

具体的には，定数項や定数係数を削除して，プログラムに与えられる値についての変数のうち最も次数が高い変数のみ残した関数を作る．それを $f(n)$ としたときに，「計算量は $O(f(n))$ である」と書く．

明確な定義としては，正の定数 c，n_0 が存在して，$n > n_0$ を満たす全ての n に対して $T(n) \leq cf(n)$ となるとき，$T(n) = O(f(n))$ と表記する．

この O を使った書き方を O 記法（big O notation）とよぶが，そんなことはどうでもよい．O 記法の関数は，n^n，e^n，$n \log n$ などが使える．$3n^2 + 2n + 1$ や $\log_3 n$ などは使わず，それぞれ n^2，$\log n$ [*1]と表す．

これだけではわかりにくいだろうから，例を見ていこう．

2.2　2 分探索*

教養情報の方でも扱った．小さい順に並べられた，長さ n の整数の配列 a に，与えられた b という数が含まれているか，含まれていればその添字はいくつかを求めるアルゴリズムである．

単純に考えれば，1 番最初から 1 つずつ見ていけばよい．これを Ruby のメソッドで定義しよう．

```
def linear_search(a,b)
    n = a.length()
    ans = -1
    for i in 0..n-1
        if a[i] == b
            ans = i
            break # b と同じ要素が見つかればループを抜ける
```

[*1] $\log_3 n = \frac{\log n}{\log 3}$ であるから，log の底が定数の時は無視してよい．

```
        end
    end
    ans
end
```

b と同じ要素がない場合には -1 を返すようにしている．

さて，このアルゴリズムの計算量を求めることを考えよう．かかる作業としては，

- 2 行目の代入
- 3 行目の代入
- 以下を n 回ループ
  - $a[i]$ が b と等しいかを判定
  - 等しければ結果を代入（これは全体で 1 回しか行われない）
- 結果の出力

の 4 つである．n 回ループとしているのは，計算量は**最悪の場合を想定して**求めるものだからである．代入や比較などの演算を 1 とすれば，この計算量は $4+n=O(n)$ と表される．カンタンだね．

このアルゴリズムを，線形探索（linear search）という．次に紹介するのが，これと比較してより効率のよい2分探索（binary search）というアルゴリズムである．いきなりコードで書いても意味不明だと思うので，簡単に流れを説明しよう．

まず，$\{2,5,7,10,13,15,18,20,23,26,28,31,34,36,39,41,44\}$ という単調増加数列[*2]の中に，36 がどこに含まれているかを 2 分探索で判定しよう．「番目」の値は 0 からはじまる．

- まず配列の真ん中の要素の値を調べる．この場合は 23（8 番目）．
- 36 は 23 より大きいので，9 番目～16 番目にあることがわかる．
- 9 番目～16 番目の真ん中の要素の値を調べる．この場合は 34（12 番目）．
- 36 は 34 より大きいので，13 番目～16 番目にあることがわかる．
- 13 番目～16 番目の真ん中の要素の値を調べる．この場合は 39（14 番目）．
- 36 は 39 以下なので，13 番目～14 番目にあることがわかる．
- 13 番目～14 番目の真ん中の要素の値を調べる．この場合は 36（13 番目）．
- 36 は 36 以下なので，13 番目にあることがわかる．
- 確かに 13 番目は 36 だ!!

というわけで，36 は 13 番目であることがわかった．本来は頭から 14 回調べなければいけないところを，4 回調べるだけで場所が特定できていることがわかるだろう．

それじゃあ，37 が含まれていないことはどうやってわかるだろう．「13 番目～14 番目の～」の行までは上と同様だ．

- 13 番目～14 番目の真ん中の要素の値を調べる．この場合は 36（13 番目）．

[*2] どうでもいいが，これは Upper Wythoff sequence と呼ばれる数列の頭 17 個である．$a_n=\lfloor n\times\phi^2\rfloor\left(\phi=\frac{1+\sqrt{5}}{2}\right)$ で定義されている．

- 37 は 36 より大きいので，14 番目にあることがわかる．
- 14 番目は 39 だぞ??? → 37 はない!!

というわけで，最後に本来あるべき番号のところで調べてみて，実は違った，という時に含まれていないことになる．

じゃあ，とりあえず Ruby のメソッドを定義してみよう．

```
def binary_search(a,b)
    n = a.length()
    sindex = 0 # 探している範囲の最初の添字
    eindex = n-1 # 探している範囲の最後の添字
    ans = -2 # 返す値．検索中の時は-2 とする
    while ans == -2 # ans が-2 以外のときはもう見つかっている
        if (eindex-sindex+1) > 1
            med = (eindex+sindex)/2 # 真ん中の要素の添字
            if a[med] > b
                eindex = med - 1
            else
                sindex = med
            end
        else # 探している範囲が 1 つだけのとき
            if a[sindex] == b
                ans = sindex
            else
                ans = -1 # 見つからなかったときは-1 を返す
            end
        end
    end
    ans
end
```

コードを読むのが苦手な人は，焦らずゆっくり把握していこう．上の箇条書きにしている判定法と照らし合わせればわかってくるはずだ．

さて，本題に移ろう．これの計算量を求めたい．while 文があってこれが何回ループするかが一番重要なところである．じゃあ何回ループするのか．

アルゴリズムの観点から見ていこう．最初に n 個の要素があって，真ん中の値を調べる．次に，前後どちらかの $\frac{n}{2}$ 個の要素から，真ん中の値を調べる．さらに，そのなかの前後どちらかの $\frac{n}{2}$ 個（全体の $\frac{n}{4}$ 個）の要素から，真ん中の値を調べる．……というふうに，調べる範囲が 1 つになるまで探す範囲が $\frac{1}{2}$ になっている．つまり，1 回の操作で，1 つ前の操作で絞った範囲からさらに $\frac{1}{2}$ に絞り込むことができ

て，これを 1 個になるまで続ければよく，ループの回数はこの絞り込んだ回数である．数式で表せば，

$$n \times \frac{1}{2} \times \frac{1}{2} \times \cdots \times \frac{1}{2} = 1$$

となるときに何回 $\frac{1}{2}$ をかけたかがわかればいい．これは簡単で，

$$n\left(\frac{1}{2}\right)^k = 1 \quad \Rightarrow \quad k = \log_2 n$$

であるから，計算量は $O(\log n)$ となる．

$O(n)$ と $O(\log n)$ でそんなに差が出るものか，と思うかもしれないが，a の大きさ（n）が大きくなるにつれてその差も大きくなってくる．たとえば，$n = 1$ のときの時間を 1 ミリ秒として，どれくらいの時間がかかるかを計算してみると，表 2.1 のようになる．

n	線形探索	2 分探索
1	1 ミリ秒	1 ミリ秒
1,000	1 秒	10 ミリ秒
1,000,000	17 分	20 ミリ秒
1,000,000,000	12 日	30 ミリ秒

表 2.1　線形探索と 2 分探索の計算時間を単純に比較した表．

いやーそんなの理論上の話でしょーとか言うなら，実際に試してみればいい．図 2.1 は，線形探索と 2 分探索を行った時の，配列の大きさとかかる時間の関係を表すグラフである[*3]．2 分探索のほうはずっと 0 のあたりをさまよっているが，別にイカサマをしているわけではなく，本当に一瞬で計算しているのである．線形探索との差は歴然としている．また，線形探索の計算量が $O(n)$ となっていることもよくわかる．

このように，単純なアルゴリズムでは時間のかかる計算を，ハードウェア（コンピュータ側の設備）を変えることなく，アルゴリズムの変更だけで高速化することができるのだ．

さて，計算量の重要性がわかってきたところで，次は配列要素の並べ替えのアルゴリズムを見ていこう．

2.3　ソートアルゴリズム

配列の中身を小さい順に並べるソート（整列）アルゴリズムには，様々なものがある．試験で名前を聞かれることはまずないから名前は覚えなくていいが，このセクションを通してそれぞれのアルゴリズムの計算量がどれくらいになるかを自力で判断できるようになろう．

[*3] 隣り合う要素どうしの差が 1 以上 100 以下になる単調増加の配列をランダムで生成し，(要素数) × 13,24,…,79 の値を探索させている．グラフに出ている実行時間は，7 回の探索の合計時間である．線形探索は，途中で break を入れることなく，値が見つかっても最後まで探索させている．これは東京大学駒場キャンパスの情報教育棟の iMac 端末を用いて，MacOS 上で行った．

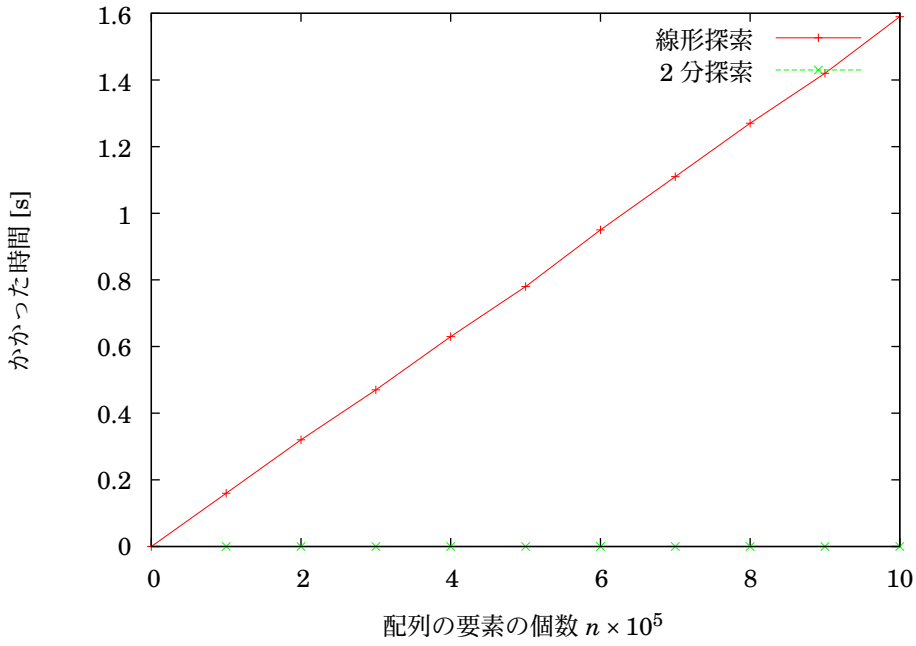


図 2.1　線形探索（赤線）と 2 分探索（緑線）の速度を実際に比較したグラフ．bench.rb によって出力されたグラフをそのまま用いた．横軸は n（配列の要素数），縦軸は探索にかかった時間である．

2.3.1　選択ソート

誰でも思いつきやすく，かつかなり効率の悪いアルゴリズムである．

まず配列の中の最小値を探してそれを配列の最初まで持ってきて，つぎにそれを除いた要素の中の最小値を探して配列の 2 番目に持ってきて，……という方法である．Ruby のコードで書くと以下のようになる．

```
def simple_sort(a)
    n = a.length()
    for i in 0..n-1
        min = a[i] # 最小値を覚えておく変数
        minindex = i # 上の最小値を持つ配列の要素の添字を覚えておく変数
        for j in i+1..n-1
            if min > a[j]
                min = a[j]
                minindex = j
            end
        end
        v = a[i]
```

```
        a[i] = min
        a[minindex] = v
    end
    a
end
```

さて，これの計算量を求めよう．内側の for 文は $(n-1)-(i+1)+1=n-i-1$ 回ループして，外側の for 文で i を 0 から $n-1$ まで動かすので，ループの回数は，

$$(n-1)+(n-2)+\cdots+0=\sum_{k=1}^{n-1}k=\frac{n^2-n}{2}$$

となるから，計算量は $O(n^2)$ である．

2.3.2 バブルソート

ここから先ではアルゴリズムの紹介のみにして，コードは載せない．実装する必要がある場合には，君たちの持っているプログラミング力で自前で実装して欲しい．

さて，次に紹介するのは，これも単純なソートアルゴリズムの 1 つである．教科書では，計算量のところではなくループの練習のところ（p.70）で登場した．

バブルソート（bubble sort）は，最初の 2 つを見て前にあるはずのものが後ろにあるなら入れ替える，その次の 2 つを見て前にあるはずのものが後ろにあるなら入れ替える……ということを繰り返して，全体をソートする[*4]．

たとえば，{ 星見月夜（ほしみつきよ），棗 梓（なつめあずさ），小鳥遊まどか（たかなし），藍川有紗（ありさありさ） } という 4 人の名前をあいうえお順に，バブルソートで並べ替えよう．

- 星見月夜，棗梓，小鳥遊まどか，藍川有紗
  星見月夜は棗梓より後なので，月夜と梓を入れ替える
- 棗梓，星見月夜，小鳥遊まどか，藍川有紗
  星見月夜は小鳥遊まどかより後なので，月夜とまどかを入れ替える
- 棗梓，小鳥遊まどか，星見月夜，藍川有紗
  星見月夜は藍川有紗より後なので，月夜と有紗を入れ替える

一周終わった．これで月夜が一番後ろになった．一周目では，かならず一番最後にくるべき要素が一番最後に来る．では，二周目に入ろう．

- 棗梓，小鳥遊まどか，藍川有紗，星見月夜……入れ替える
- 小鳥遊まどか，棗梓，藍川有紗，星見月夜……入れ替える
- 小鳥遊まどか，藍川有紗，棗梓，星見月夜……そのまま

[*4] なお，バブルソートは，一般的には後ろから見ていくソートの方法のことをいう．教科書では前から見ていく方法を紹介していたので，本書でもこちらを採用した．逆から見ていっても基本的にはこれと同じなので，ここでは詳しく解説しない．

3 つめは，上で言ったようにもう揃っているはずなので，行う必要はない．また同様に，二周目でかならず最後から二番目にくるべき要素がその位置に来る．では，三周目に入ろう．

- 小鳥遊まどか，藍川有紗，棗梓，星見月夜……入れ替える

これであいうえお順になった．三周目では，最後から 2 つの要素はそこで決まっているので，最初の $4-2=2$ つのみを見ればよいことがわかるだろう．実は，もともと逆順に並んでいる時が一番多く入れ替える操作をしなければならないのだ．

最後に，このアルゴリズムの計算量を考えよう．n 周目の操作を終えると，最後から n 番目の要素が決まるので，入れ替える操作は最大で，

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

であることがわかる．したがって選択ソートと同じように，計算量は $O(n^2)$ である．

2.3.3 マージソート

ここまで紹介した 2 つは，ともに計算量が $O(n^2)$ であった．これより効率のいい方法はないのだろうか．実はまだまだたくさんある．

その中の 1 つであるマージソート（merge sort）を紹介しよう．小さい順に並んでいる 2 つの配列をうまく組み合わせて，大きな 1 つの配列を作る作業を何回か行うことで，小さい順に並んだ配列を作ることができる．これも説明だけではわかりにくいので，例を見ていこう．

たとえば，{ 天ヶ瀬奈月（あまがせなつき），朝比奈晴（あさひなしん），夏目暦（なつめこよみ），乙音ニコル（おとね），玉樹桜（たまきさくら），小坂井綾（こざかいあや），あずま夜（よる），東雲希（しののめのぞむ），シロクマ } の 9 人の名前をマージソートで名前順にすることを考えよう．

- まずそれぞれを 1 つずつに分離する
- 隣り合う要素どうしで，先に来るものの方が先にくる 2 つずつの配列を作る
  { 朝比奈晴，天ヶ瀬奈月 }，{ 乙音ニコル，夏目暦 }，{ 小坂井綾，玉樹桜 }，{ あずま夜，東雲希 }，{ シロクマ }
- さらにその配列の隣どうし 2 つを併合した配列を作る
  { 朝比奈晴，天ヶ瀬奈月，乙音ニコル，夏目暦 }，{ あずま夜，小坂井綾，玉樹桜，東雲希 }，{ シロクマ }

マージソートでミソなのは，この併合の時に効率がいいという点である．たとえば最初の 2 つの { 朝比奈晴，天ヶ瀬奈月 }，{ 乙音ニコル，夏目暦 } について考えよう．併合した後にできる配列の一番最初の要素は，明らかに朝比奈晴か乙音ニコルのどちらかである（この場合は朝比奈晴）．朝比奈晴の次は，明らかに朝比奈晴の次の要素の天ヶ瀬奈月か，もう 1 つの配列の 1 番最初の乙音ニコルだ．したがって，選択ソートなどのように毎回配列を全部を探索することなく，2 つの配列の最初の要素から順番に見ていけば併合が可能なのである．これがマージソートの計算量を軽くしている一因である．

- さらに配列の隣どうし 2 つを併合した配列を作る
  { 朝比奈晴，あずま夜，天ヶ瀬奈月，乙音ニコル，小坂井綾，東雲希，玉樹桜，夏目暦 }，{ シロク

マ }[*5]

- さらに併合

  { 朝比奈晴，あずま夜，天ヶ瀬奈月，乙音ニコル，小坂井綾，東雲希，シロクマ，玉樹桜，夏目暦 }

これで名前順になった配列が完成した．

さて，マージソートの計算量を考えよう．定数項とならないループによる計算が行われるのは，併合し配列を作る作業の部分である．

まず 1 回の併合の作業をするときに何回ループするか．先ほども説明したように，要素が a 個と b 個の配列を併合するとき，それぞれの配列を頭から見ていき，小さい方を新しい配列の先頭から順番に入れていくので，要素自体は $a+b$ 回しか読み込まない．したがって，この計算量は $O(a+b)$ である．この併合の作業を全体にほどこすと（これを 1 段階の併合と呼ぼう），全体の要素数を n とすれば合計 n 個の要素を順番に見ているので，結局計算量は $O(n)$ となる．

さて，この併合の作業を何段階ほどこすか．これは先に説明した二分探索に似ている．それぞれの配列の要素の数を考えてみると，1 段階の併合で 2 倍の要素数になっている．2 段階の併合で 4 倍の要素数にできる．つまり，1 枚ずつから 2 倍，2 倍……としていったとき何回目で n に到達するか，数式で書けば

$$1 \times 2 \times 2 \times \cdots \times 2 = n$$

となるとき 2 を何回かけたのかがわかればよいのだ[*6]．これは

$$2^k = n \quad \Rightarrow \quad k = \log_2 n$$

の k にあたるから，併合の作業は $\log_2 n$ 段階行われることになる．

1 段階の併合で $O(n)$ の計算量の処理をして，それを $\log_2 n$ 段階行うんだから，全体として $O(n) \times \log_2 n = O(n \log n)$ の計算量で計算できる．

2.3.4 計算量の比較

さて，$O(n^2)$ と $O(n \log n)$ の差はどれくらいなのか．2 分探索の時と同じように，まずは $n = 1$ の時の計算時間を 1 ミリ秒としたときの理論値で考えてみよう（表 2.2）．

これまた極端な表になった．32 年はさすがにかかりすぎだと思うが，それでもこのくらいの差がでるらしい．例によって，やっぱり実際にプログラムを組んで試してみよう．図 2.2 は，様々なソートアルゴリズムの計算時間を，実際にソートさせて測ってグラフにしたものだ[*7]．シケプリ内で紹介していない

[*5] 要素数が $9 = 2^3 + 1$ のためにシロクマが最後まで 1 人ぼっちになったが，シロクマに特に恨みはない．むしろシロクマ大好きです!!!!!

[*6] 正確には上の例であげたように，2^n の形の要素数でない場合には余り物が出てくるから，$1 \times 2 \times 2 \times \cdots \times 2 \geq n$ を求めることになる．しかし，計算量という**大雑把な**概念で考える場合には，そんな余り物のことはどうでもよい．だけどシロクマかわいいからシロクマ好きです．ぐるるしろ!

[*7] 2 分探索のときもそうだったが，表に書いた理論値が長く見積もり過ぎなのはあきらかだ．これは，$n = 1$ のときに 1 ミリ秒かかる，という仮定が間違っているだけ．同じ仮定のもとで n によってどれくらい計算時間が変わるのかだけ比較できれば十分であるから，実際に計算させずとも，机上の空論を述べるだけで十分である．もっとも，コンピュータに計算させるのも机上であるが．

n	$O(n^2)$	$O(n\log n)$
1	1 ミリ秒	1 ミリ秒
100	10 秒	0.7 ミリ秒
10,000	27 時間	2 分
1,000,000	32 年	6 時間

表 2.2　$O(n^2)$ の選択ソート，バブルソートと $O(n\log n)$ のマージソートの計算時間を単純に比較した表．$O(n\log n)$ の方は，$n\log_2(n+1)$ として計算している．

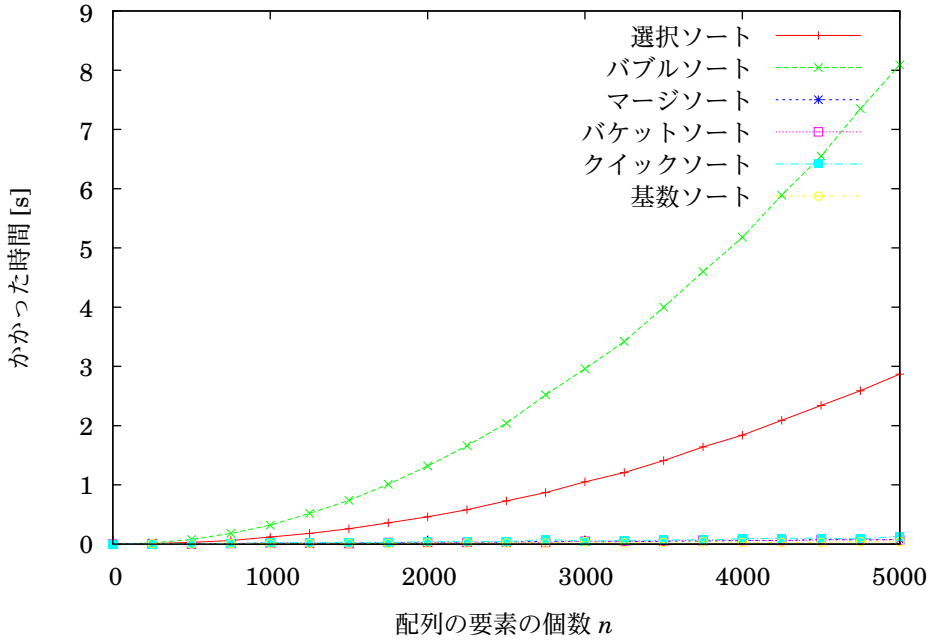


図 2.2　様々なソートアルゴリズムの計算時間を比較したグラフ．

ソートもいくつか入っている[*8]．あきらかに選択ソートとバブルソートだけ時間が大幅にかかっている．2 次関数的にかかる時間が増えていることも確認できる．$O(n^2)$ と $O(n\log n)$ の違いは相当に大きいものであることもはっきりわかる．

しかしこれだけでは他のソートの区別がつかないので，図 2.3 には時間のかかりすぎる 2 つをのぞいたほかのソートアルゴリズムのみのグラフを載せた．みな 1 次関数的に時間がかかっているのがわかる．あまり log の役割が目立たない．しかし，図 2.2 のグラフで用いた配列の 10 倍の要素数の配列で計算しても，全体で 2 秒かかっていないことから，単純にソートするよりも他のアルゴリズムのほうが効率が

[*8] 紹介していないソートは，クイックソート，バケットソート，基数ソートの 3 つである．3 つともまた有名なソート方法である．計算量は，クイックソートが $O(\log n)$ 以上 $O(n^2)$ 以下で平均が $O(n\log n)$，バケットソートと基数ソートは $O(n)$ である．クイックソートの計算量は単純に決まらないところが面倒だが，興味のある人は文献やインターネットで調べてみるとよいだろう．

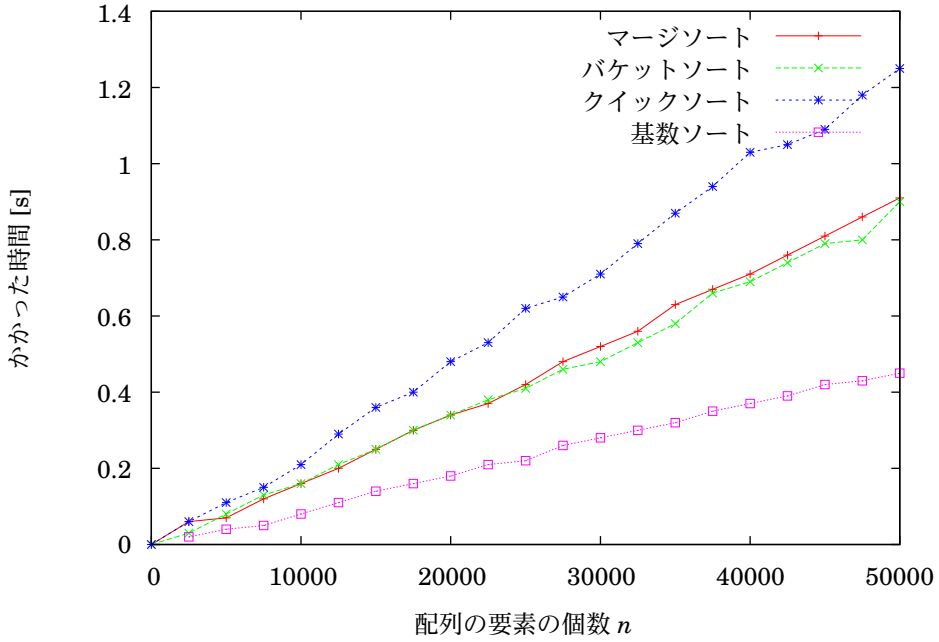


図 2.3　計算量が $O(n^2)$ でないいくつかのソートアルゴリズムの計算時間を比較したグラフ.

よいことがよくわかるだろう.

2.4　フィボナッチ数列

次はフィボナッチ数列の第 n 項を求めるアルゴリズムを通して計算量の概念を学んでいこう.

2.4.1　フィボナッチ数列の再帰的定義

まずは，いわゆる再帰的定義というやつ．フィボナッチ数列の第 n 項を a_n とすれば，フィボナッチ数の定義

$$a_n = \begin{cases} 1 & (n = 0, 1) \\ a_{n-1} + a_{n-2} & (n \geq 2) \end{cases}$$

をそのまま用いてメソッドにしてしまうもの．Ruby のコードで書くと以下のようになる.

```
def fib(n)
    if n < 2
        1
    else
        fib(n-1) + fib(n-2)
    end
end
```

非常にシンプルだ．さて，これの計算量を求めよう．計算量自体は，全体として何回このメソッドが呼び出されるかがわかればよい．それ以外にループする要素がないからだ．

fib(n) を計算するのに必要な計算量を $f(n)$ としよう．fib(n) を計算するには fib($n-1$) と fib($n-2$) の和をとっているから，$f(n)=f(n-1)+f(n-2)$ と考えて問題ないだろう．これをガチャガチャ解くと[*9]，

$$f(n)=\frac{\alpha^{n+1}-\beta^{n+1}}{\sqrt{5}} \quad \left(\alpha=\frac{1+\sqrt{5}}{2}, \beta=\frac{1-\sqrt{5}}{2}\right)$$

となる．しかし計算量の議論では $\beta^n \to 0\,(\mathrm{as}\, n\to\infty)$ であるから β^{n+1} の項は無視してよく，$n+1$ の 1 や分母の $\sqrt{5}$ も無視してよいから，結果として計算量は

$$O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)\approx O(1.618^n)$$

と表される．

「こんなの求められるかアホ」と思うかもしれないが，これくらい情報科学者たちの間では当然の理なので慣れておこう．議論としては単純明快なので，東大生のみんななら十分にその場で対応できるはずだ．

2.4.2 数え上げ

しかしわざわざフィボナッチ数列ごときで再帰的定義をすることはない．メモリも食うし，計算時間も指数時間かかるから遅い．

再帰的定義のどこが非効率的かというと，同じ値を何度も求めているからである．fib(n) を求めるために fib($n-1$)+fib($n-2$) の計算をしているが，fib($n-1$) を求める際に fib($n-2$) を使い，しかもそのとき fib(n) を求めたときの fib($n-2$) の結果を再利用できずにもう一回最初から計算しなおしている．これが，上の再帰的定義の計算量が大きい原因だ．

一度求めたものはもう求めない方向でいきたい．そのためには，フィボナッチ数列を 1 から順番に求めていけばよい．最初の 2 つが分かればその次の 1 つがわかるし，そしたらその次もわかる．これを Ruby のメソッドで定義しよう．

[*9] $f(n)=t_n$ として，$t_n=t_{n-1}+t_{n-2}$．$\alpha=\frac{1+\sqrt{5}}{2}$，$\beta=\frac{1-\sqrt{5}}{2}$，また $t_1=t_0=1$ とすれば $t_n-\alpha t_{n-1}=\beta(t_{n-1}-t_{n-2})\Rightarrow t_n-\alpha t_{n-1}=\beta^{n-1}(t_1-\alpha t_0)=\beta^n$，$t_n-\beta t_{n-1}=\alpha(t_{n-1}-t_{n-2})\Rightarrow t_n-\beta t_{n-1}=\alpha^{n-1}(t_1-\beta t_0)=\alpha^n$．この 2 式から t_{n-1} を消去すれば，$t_n=\frac{\alpha^{n+1}-\beta^{n+1}}{\alpha-\beta}=\frac{\alpha^{n+1}-\beta^{n+1}}{\sqrt{5}}$．なお $\alpha=\frac{1+\sqrt{5}}{2}$ は黄金比として有名な値である．

```
def fib(n)
    f = 1 # これが fib(1) にあたる
    p1 = 1 # これが fib(0) にあたる
    for i in 2..n
        p2 = p1 # p2 に fib(i-2) を代入
        p1 = f # p1 に fib(i-1) を代入
        f = p1 + p2 # fib(i-2)+fib(i-1) から fib(i) を算出
    end
    f
end
```

for 文の 1 周が始まった瞬間，p2 には $\mathrm{fib}(i-3)$ が，p1 には $\mathrm{fib}(i-2)$ が，f には $\mathrm{fib}(i-1)$ が入っていて，これを順番に進めると，p2 に $\mathrm{fib}(i-2)$，p1 に $\mathrm{fib}(i-1)$，f に $\mathrm{fib}(i)$ が入った状態で終わる．その次の 1 周のはじまりの時，また同様に p2 には $\mathrm{fib}(i-3)$ が，p1 には $\mathrm{fib}(i-2)$ が，f には $\mathrm{fib}(i-1)$ が入っていて，という状態になるので，これを繰り返せば最後には $\mathrm{fib}(n)$ が求まる，という仕組みだ．

この計算量はどう見ても $O(n)$ でよいだろう．さきほどの再帰的定義よりも大きく計算量を低減できた．さて，これよりも効率のよい計算を考えよう．

2.4.3 行列の n 乗を用いたアルゴリズム

実は，フィボナッチ数は以下のように行列の積で求めることができる．

$$\begin{pmatrix} \mathrm{fib}(n+1) \\ \mathrm{fib}(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathrm{fib}(n) \\ \mathrm{fib}(n-1) \end{pmatrix}$$

ここから，

$$\begin{pmatrix} \mathrm{fib}(n+1) \\ \mathrm{fib}(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

となることは容易にわかるだろう．

さて，行列のべき乗を効率よく求めることを考えよう．そのまま掛けあわせてもいいが，それでは計算量が $O(n)$ となってしまう．ここで，バイナリ法という方法を紹介しよう．

$Q^{2n} = Q^n \times Q^n$ で出すことができるから，1 回ずつの計算で，それぞれ Q^2，Q^4，Q^8，Q^{16}，$\cdots$ と求めることができる．これを上手く組み合わせれば，任意の自然数乗に対して効率よく計算できる．たとえば $Q^{42} = Q^{32} \times Q^8 \times Q^2$ とすれば，直接 42 回掛けるよりも圧倒的に早いことがわかる．

さて，このアルゴリズムでの計算量を考えることにしよう．Q^n を求めるとき，$n \le 2^p$ となる最小の整数 p をもってくれば，Q^n は $Q^{2^{p-1}}$，$Q^{2^{p-2}}$，$\cdots$，Q の積で求めることができる．

したがって，まず最初に用意するべき Q^{2^p} を求める時の計算量を考えよう．Q^{2^p} を求めるには p 回の計算をすればよいから，$n \le 2^p \Leftrightarrow p \ge \log_2 n$ 回の計算をすればよいことになる．

次に，それを掛け合わせるときのことを考えよう．Q^{2^n-1} の形のときに最も掛け合わせる回数が多い．そのとき，

$$Q^{2^n-1} = Q^{2^{n-1}} \times Q^{2^{n-2}} \times \cdots \times Q$$

となり，全体で $\log_2(n+1)$ 回掛けあわせていることがわかる．

この 2 つから，全体の計算量は $O(\log n)$ となることがわかる．

2.4.4 さらに効率のいいアルゴリズム*

バイナリ法で計算量を $O(\log n)$ まで抑えることができた．これよりも効率のいい方法はあるのだろうか?

実はあるのである．フィボナッチ数は漸化式で表すことができるから，それを解くこともできる．これを解くと，

$$a_n = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right\}$$

となる[*10]．これはただの単純な代数計算であるので，計算量は $O(1)$ となる[*11]．このように，変に頑張ってアルゴリズムを組み立てて頑張るより数学的解法の方が早いこともしばしばである．

2.5 この章のまとめ

計算量は，アルゴリズムを計算するときにかかる時間がどんな感じなのかの表し方だ．同じ目的のアルゴリズムであっても，計算量は異なっていることが多い．どんなアルゴリズムを使うべきなのかは，実際に使うときにその場に応じて選ぶのが大切である．必ずしも計算量が少ないアルゴリズムをいつも使う必要はない．

計算量を求めるときには，同じ操作（ループ）を何回するのかを調べるのがキホン．それではわかりにくいとき（たとえば while 文を使っていたり，再帰的定義を使っていたりするときなど）は，実際に頭の中や手でアルゴリズムを動かしてみて，作業を行う回数を推定する．

試験では，コードを与えられて「これはどれくらいの計算量ですかー」と聞かれる．グダグダ言わずに実際にプログラムのコードを書いたり過去問を解いたりなどして，計算量の概念を掴んでほしい．

[*10] 実はこの値はさきほど再帰的定義の計算量を求めるときにも出した．

[*11] n 乗を計算するときにループが必要なのではないか，と思うかもしれない．しかし，コンピュータは exp と log のテイラー展開を使ってべき乗を計算しており，何回も掛けあわせてべき乗の値を出しているわけではない．したがって計算量は $O(1)$ と表せる．

情報科学を学ぶ意義

情報科学は，情報のプログラミング・アルゴリズムの部分を深く掘り下げた科目だ．これが順必修となっている理由はなんなのか．

キミたちはまだ知らない

プログラミングは高校情報でも扱われることがまれにある程度であり，厨二病時に「俺はプログラムを書きてぇ」と思ってしまったような人をのぞいては，プログラミングをしたことすらない人も多いだろう．

数学や物理化学といったものは高校時代で扱い，大学に入ってそれをもっと詳細にやっていく，という形が多い．しかし，情報科学はそういうわけにはいかない．僕らにとって「情報科学」は未知の学問だ．

今の社会で広がる情報科学

情報科学で学ばれるプログラミングやアルゴリズムなしでは，今の日本の優れた産業はないといっても過言ではない．高性能な CPU により，省電力・小型でも高速な計算ができるようになった今だからこそ，このようなあらゆる電子機器に囲まれた生活ができている．携帯電話だけじゃない．炊飯器だって，冷蔵庫だって，全部中にはプログラムが入っている．

ここ数十年で一気にその重要性をました情報科学．ただ Word が使えてネットサーフィンができればいい，というものではない．その背景になにがあるのか，それがどうやってできているのか．大学はそういうものを学ぶところではないだろうか．

そしてそういうものが教養として重要なのではないだろうか．これは理系・文系関係ないことである．

これからのキミたちに

手の込んだ研究には情報科学は欠かせない．何の分野だってまずはコンピュータでシミュレーションを行う世の中である．どんな物質からどんな化合物ができるのかもシミュレーションで推定することができるのだ．物理，化学，生物学だけでなく，経済学，金融工学にも欠かせない存在だ．

キミたちが今後研究に取り組んだ際，売っているプログラムでは限界が出てくる．自分でプログラムを組み立て，それを自分でスパコン上で動かさないといけなくなるときが訪れるかもしれない．そのときになってプログラムが書けるように，アルゴリズムを思い浮かべられるように，いまここで**教養として**情報科学を知っておいて欲しいのである．

今すぐには役に立たないかもしれない．それでも，たしかな教養を，その手に．

第 3 章　整数型と浮動小数点型

informatical reality

さて，ここからは実際に数字を計算することを考えよう．しかし，計算する前に数を表現する方法が必要だ．コンピュータは 2 進数であらゆる数を表現していることはご存知と思うが，負の数や小数を表現する方法をちゃんと決めねばならない．

3.1　整数

コンピュータで負の数を含めた整数を表現する際は，一般的に 2 の補数表現（two’s complement）を用いる．

まず，一番大きい位の桁が 0 であるときにはそのまま 10 進数の値と対応する．たとえば 00000000～11111111 の 8 ビットの数（表せる数は 256 通り）で考えよう．正の数はそのまま 00000000～01111111（0～127）の 128 通りである．00000011 なら 3，00001010 なら 10 などとなる．素直だ．

次に，負の数だ．負の数は以下のように考える．ここでは同じように 8 ビットの数で考えよう．11111111 を-1 として，-1 増えるごとに 2 進数も-1 減らしていく．11111100 なら-4，11111010 なら-6 となる．これを続けると 10000000 が-128 となり，これより小さな数は表現できない（表現しようとすると正の数と混同してしまう）．

確かに面倒くさい表現である．そのまま-と + を表現するビットを作るとか他の方法も考えられるが，0 に +0 と-0 の 2 つの存在を防げる，大小比較時にラク，などの理由から，普通はこれを用いられることが多い．

なお，正の整数のみをそのまま表現するタイプの変数を作れる言語もたくさん存在する．

3.2　実数

整数はまだ単純に表現できた．しかし，あれでは表現できる数の幅が小さい．しかも実数が表現できない．小数の計算ができないなんて計算機失格だ．でもできたらなるべく表現の幅を広げたいし，計算も高速にしたい．じゃあどうやって表現するか．

現在のコンピュータでは，多く IEEE754 という規格で定められた浮動小数点（floating point）表現を用いる．この表現では，符号（正か負か），小数点の位置（指数，2 の何乗か），小数点以下の数（仮数，1. いくつか）の 3 つの情報からなる．また，浮動小数点表現では普通，単精度（32 ビット）か倍精度（64 ビット）表現で表すことになっている[*1]．

まとめると，表現できる値は 2 進数で $1.\text{仮数部} \times 10^{\text{指数部}}$ で，負なら符号部が 1，ということになる．

なお，試験で細かい数字（バイアス値が 127 だの，仮数部の桁数が 52 だの）は聞かれないので覚えなくてもよい．

[*1] ほかにも 16 ビットで表現する半精度，128 ビットで表現する四倍精度なども用意されている．そのほかに IEEE754 では十進法で浮動小数点を扱うための規格も用意されている．これは後述する 2 進数特有の誤差を防ぐためのものであるが，かなりビット表現が複雑になるため解説しない．

3.2.1 符号部

符号部は 1 ビット．0 のときに正で 1 のときに負．これはまあ単純だ．

3.2.2 指数部

指数部は，単精度であれば-127～128，倍精度であれば-1023～1024 までを表せる．これは 2 の補数表現を**用いない**．指数部が 00…0 のときに最小値（単精度なら-127）を表し，1 増えれば単純に表す数も 1 増えることになっている．このやり方だと 100…0 のときに 1 になる．これくらいは覚えておいてもいいかもしれない．

教科書ではバイアス値というものでこれを表現している．2 進数の数からバイアス値（単精度なら 127，倍精度なら 1023）を引けば本当に表そうとしている数だよ，という考え方である．わかりやすい方で覚えればよい．

3.2.3 仮数部

仮数部は，単精度なら 23 桁，倍精度なら 52 桁を扱える．

これも 2 進数で表すが，小数の 2 進数を表すことになる．たとえば，仮数部が 1100…0 のとき，表される数（の絶対値）は 2 進数で $1.11\times10^{\text{指数部}}$ となるが，これは 10 進数で $\left(1+\frac{1}{2}+\frac{1}{4}\right)\times2^{\text{指数部}}=1.75\times2^{\text{指数部}}$ となる．

計算での有効桁数は仮数部の桁数が決めている．たとえば倍精度の場合，2 進数で 52 桁なので，10 進数では $52\times log_{10}2\simeq16$ 桁 ということになる．

3.2.4 表現できない数

ここまでで，表現できない数があることに気づいたろうか．

0 だ．もともと 1 が加えられた状態で計算を始めているので正確に 0 を示す手段がない．すべての桁が 0 であれば +0 を，符号部の 1 桁のみ 1 でそれ以外が 0 であれば −0 を表すことになっている．

まあそれ以外もちょこちょこあるが専門でないので気にしなくてよい[*2]．

3.2.5 2 つの表現の仕様まとめ

精度	指数部		仮数部	合計ビット数	十進換算有効桁数
単精度	8 桁	-127～128	23 桁	32 ビット	約 7 桁
倍精度	11 桁	-1023～1024	52 桁	64 ビット	約 16 桁

[*2] そもそも指数部が 00…0 のときと 11…1 のときは，特殊な値を表すという決まりがある．指数部が 0 で仮数部が 0 ならば 0 を，仮数部が 0 でないならば非正規化数（これは自分で調べて）を表す．また指数部が 11…1 で仮数部が 0 ならば ∞ を，仮数部が 0 でないならば NaN（非数）を表す．したがって，指数部の最小値と最大値は結果的には用いることができない．ただ，このシケプリではわかりやすさのために最小値と最大値を用いることができるものとして扱っている．

3.2.6 ちょっと練習

少しだけ変換の仕方を見ておこう.

浮動小数点表現から十進実数に

単精度で

```
0 10110110 01101010010010100100101
```

が何を表すかを考えよう（区切りは見やすさのために作ってあるだけでなくてもいい）.

■符号部　0 なので正の数.

■指数部　10110110 は 10 進数になおすと 182．0 のときに-127 なので，182-127=55.

■仮数部　2 進数で 1.01101010010010100100101 は,

$$1+\frac{1}{2^1}\times 0+\frac{1}{2^2}\times 1+\frac{1}{2^3}\times 1+\frac{1}{2^4}\times 0+\cdots = 1.41519594192504$$

を表す（面倒なので最後の方は適当に丸めている）.

■全部まとめる　したがって，これはだいたい $1.415\times 2^{55}=5.10\times 10^{16}$ という数を表している.

試験本番ではこんなグチャグチャな数は出ないので安心して計算しよう.

十進実数から浮動小数点表現に

6.8 を単精度浮動小数点表現で表すことを考えよう．まず,

$$6.8 = 4+2+\frac{1}{2^1}+\frac{1}{2^2}+\frac{1}{2^5}+\frac{1}{2^6}+\cdots+\frac{1}{2^{4n+1}}+\frac{1}{2^{4n+2}}+\cdots$$

であるから，6.8 を 2 進数であらわすと

$$110.110011001100\cdots = 1.10110011001100\cdots\times 10^{10}$$

となる（10^{10} も 2 進数であることに注意）.

さて，それぞれの部分について考えよう

■符号部　正の数なので 0.

■指数部　100…0 のときに 1 だから，10（10 進数で 2）は 100…01.

■仮数部　上の指数表示の小数点以下を書けばよいので，1011001100….

最後に桁数に注意してすべてを合わせれば，6.8 は単精度浮動小数点表現で

```
0 10000001 10110011001100110011001
```

となる（れいによって区切りは視認性のためだけでありなくてもよい）.

3.3 この章のまとめ

事実上コンピュータを扱う上で，IEEE754 で規定されている浮動小数点表示を考えることはほとんどない．そのため，符号部・指数部・仮数部の桁数を覚えることはほとんど意味が無い．

試験でも浮動小数点表現と 10 進実数の変換が聞かれることが多い．そのときも必ず仮数部および指数部の桁数と，バイアス値（指数部の最小値の絶対値）は必ず用意されている．だから，仕様だけ与えられた上で相互変換ができるようになっておけば十分であろう．まあもちろん，52 だの 1023 だのを覚えておいて損はないと思うが．

10 進実数を 2 進数にする方法

本文中ではいきなり 2 進数で表示してしまったため，戸惑った人もいるかと思う．Google 先生に各自聞いてもらうのが早いと思うが，一応ここでも説明しておく．ここで紹介するのは 1 例であり他にも方法があるので，これでわかりにくかったら各自調べてみてね．

ここではたとえば **13.7** を例にとろう．

整数部分

1 つ前の 2 で割った商を 2 で割った余りを順につなげていくだけ．

- $13 \div 2 = 6 \cdots \mathbf{1}$
- $6 \div 2 = 3 \cdots \mathbf{0}$
- $3 \div 2 = 1 \cdots \mathbf{1}$
- $1 \div 2 = 0 \cdots \mathbf{1}$

よって整数部分は **1011** となる．

小数部分

1 つ前の小数部分を 2 倍した整数部分をつなげていくだけ．

- $0.7 \times 2 = \mathbf{1}.4$
- $0.4 \times 2 = \mathbf{0}.8$
- $0.8 \times 2 = \mathbf{1}.6$
- $0.6 \times 2 = \mathbf{1}.2$
- $0.2 \times 2 = \mathbf{0}.4$ ←ここで 4 → 8 → 6 → 2 のループに入る

よって小数部分は **10110011001100**… となる．

循環小数になる場合は，どこでどんな循環が起きるかをちゃんと把握しよう．

あとは合わせればよい．すなわち，10 進数で 13.7 は，2 進数で $\mathbf{1011.10\dot{1}1\dot{0}0}$ となる．

第 4 章　誤差

informatical reality

コンピュータは有限な数しか扱えない．したがって，循環小数でさえも完全には表せない[*1]．そのために想定した解とは異なる解を返す場合がある．その差を**誤差**（error）という．

誤差にはいろいろ種類があって，それぞれできる理由も違う[*2]．ここでは 5 種類紹介するが，これらはすべて今後コンピュータであらゆる計算をするキミたちが把握しておかねばならない大事なものである．しっかり身につけよう．

4.1　オーバーフローとアンダーフロー*

教科書では紹介されていないが，まあ普通に当たり前のものである．理解にも難くないだろう．

IEEE754 で規定されている浮動小数点表示では表示できる値に限界がある．たとえば単精度では，指数部は-127～128 しか扱えない[*3]ため，これを超えるたとえば $2^{129} \approxeq 6.8 \times 10^{38}$ といった大きすぎる値や，$2^{-130} \approxeq 7.3 \times 10^{-40}$ といった小さすぎる値は扱えない[*4]．

身近なところで言えば，12 桁しか扱えない電卓で 999999999999+1 とやるとエラーが出てしまうアレである．

4.2　丸め誤差

無限小数（循環小数や無理数など）を有限桁しか扱えないことにより起きる誤差を総称していう．

たとえば $\log_{10} 2$ が 0.3010 に近いことはよく知られており，手計算で用いられることが多い．しかし実際には 0.30102999566… である．すなわちこの値には 0.00002999566… の誤差が出ている．

さて，実際にコンピュータで計算するときに現れる丸め誤差について考えよう．教科書ではこのような例が出ていた．

```
0.1 * 3 - 0.3 = 5.55111512312578e-17
```
[*5]

れいによってこれがなぜ起きるかを考えよう．

[*1] 循環小数は有理数であるから，分母・分子を整数として有理数を扱う型を用意すれば完全な計算をすることもできる．しかしメモリには限界があるから，大きすぎる整数は扱えないのもまた事実である．結局どこにでも限界はあるのだ．

[*2] もちろん，すべては「コンピュータが有限な数しか扱えないから」という理由に起因するのだが．

[*3] 前の章で書いたとおり，正確には-126～127 しか扱えない．

[*4] IEEE754 で定められた精度を超えた計算能力をそなえたプログラミング言語やライブラリも多数ある．RSA 暗号を復号化するとなると 1024bit だの 3072bit だの巨大な整数を扱うことになるため，32bit の整数型が不十分であることもしばしば．もちろんライブラリを使わなくても自前で実装できるが，実際に利用する際は既存のものを上手に使おう．

[*5] 授業中に説明があったと思うが，コンピュータ等では 10^n のことを en と表す．3.5e3 なら 3.5×10^3 を，2.7e-5 なら 2.7×10^{-5} を表している．

4.2.1 0.1 とはなんなのか

コンピュータにおいて 10 進法で 0.1 という数は存在しない．0.1 は 2 進法で $0.000\dot{1}10\dot{0} = 1.100110011\cdots\times10^{-4_{(10)}}$ だから，倍精度表現だと仮数部は 52bit だから

$$1.\underline{1001100110011001100110011001100110011001100110011001}1001\cdots\times10^{-4_{(10)}}$$

の下線部のみがデータとなる．0 捨 1 入する（丸められる一番大きな桁が 0 であればそのまま，1 であれば 1 つ大きな桁に 1 を加える）と，コンピュータでは 0.1 というのは

$$1.1001100110011001100110011001100110011001100110011010\times10^{-4_{(10)}}$$

という 2 進数で扱われることになる．

4.2.2 0.1×3 とはなんだったのか

さて，さっき 0.1 が，コンピュータでは

$$1.1001100110011001100110011001100110011001100110011010\times10^{-4_{(10)}}$$

だということを説明した．これを 3 倍（2 進数で 11 倍）しよう．

$$\begin{aligned}1.1001100\cdots11010\times10^{-4_{(10)}}\times11 &= 1.1001100\cdots11010\times10^{-4_{(10)}}\times(10+1)\\ &= 1.1001100\cdots11010\times10^{-3_{(10)}}+0.11001100\cdots11010\times10^{-3_{(10)}}\\ &= 10.01100\cdots11001110\times10^{-3_{(10)}}\\ &= 1.001100\cdots11001110\times10^{-2_{(10)}}\end{aligned}$$

すなわち，

$$1.\underline{0011001100110011001100110011001100110011001100110011001}10\times10^{-2_{(10)}}$$

であり，仮数部が 52bit なので 0 捨 1 入すれば，0.1×3 がコンピュータでは

$$1.0011001100110011001100110011001100110011001100110011010\times10^{-2_{(10)}}$$

として扱っているということがわかる．

4.2.3 0.3 とはなんぞや

こちらも倍精度で浮動小数点表示をしてみよう．0.3 は 2 進数で $1.\dot{0}01\dot{1}\times10^{-2_{(10)}}$ だから，0.3 はコンピュータでは

$$1.0011001100110011001100110011001100110011001100110011\times10^{-2_{(10)}}$$

という数で表されていることがわかる．

4.2.4 0.1×3 と 0.3 の違い

もうわかっただろう．0.1×3 は

$$1.0011001100110011001100110011001100110011001100110011001100110\mathbf{100} \times 10^{-2_{(10)}}$$

で，0.3 は

$$1.0011001100110011001100110011001100110011001100110011001100110\mathbf{011} \times 10^{-2_{(10)}}$$

だから，太字部分の最後の 3 桁に差が出ていることがわかる．その差はわずか $2^{-54} \approxeq 5.55 \times 10^{-17}$ である．これは irb で計算させた際の結果と一致している．

間違ってもこんなことを Web 上の質問コーナーで聞いてしまうような人[*6]にはならないようにしよう．

4.2.5 その他の例*

他にも `0.1*7-0.7` などでも同じようにできる．また，`(1.001-0.001)-1.0` といったものでも丸め誤差があらわれる．それぞれ実際に 2 進数になおして確かめてみよう．

4.3 桁落ち誤差

たんに桁落ちともいう．値が極端に近い 2 つの数の差をとろうとした時に有効数字が著しく落ちて起こる誤差．有効数字が 5 桁の 2 つの数 2.9997 と 2.9995 の差をとると 0.0002 となり，有効数字が 1 桁になってしまう．

実際の計算手順の中での例としては以下のようなものがよく見られるので取り上げておく．回避する方法も書いておくが，こういった計算でけた落ち誤差が発生する，ということを知っておけば十分だろう．

4.3.1 平方根の差*

$\sqrt{1001} - \sqrt{999}$ など値の近い平方根の差を計算すると，当然のごとく桁落ち誤差が発生する．たとえば有効数字 8 桁で計算すると

$$\sqrt{1001} - \sqrt{999} = 31.638584 - 31.606961 = 0.031623$$

となり，有効数字が 5 桁まで落ちてしまった．

この桁落ち誤差には回避する方法がある．有理化してやればよいのだ．

[*6] http://stackoverflow.com/questions/19526266/why-is-0-3-not-to-0-1-3

$$\begin{aligned}\sqrt{1001}-\sqrt{999} &= \frac{\left(\sqrt{1001}-\sqrt{999}\right)\left(\sqrt{1001}+\sqrt{999}\right)}{\sqrt{1001}+\sqrt{999}} \\ &= \frac{1001-999}{\sqrt{1001}+\sqrt{999}} \\ &= \frac{2}{\sqrt{1001}+\sqrt{999}}\end{aligned}$$

こうすれば，有効数字が落ちずに計算できる[*7]．同じように有効数字 8 桁で計算してやると，

$$\sqrt{1001}-\sqrt{999} = \frac{2}{\sqrt{1001}+\sqrt{999}} = \frac{2}{31.638584+31.606961} = 0.031622781$$

となる．実際の値は約 0.03162278055 なので有効数字の範囲で正しい値が出ていることがわかる．

4.3.2 微分

微分の公式は

$$f'(x) = \lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

というものであるのは知っているだろう．h をなるべく小さくしていくと本来の微分の値に近くなる[*8]，しかし小さすぎると $f(x+h)$ と $f(x)$ の差が小さくなりすぎて桁落ち誤差が発生してしまう，というものだ．

この微分の公式を用いる際は，丸め誤差の定数を ε（倍精度なら 10^{-16} である）として，$h=\varepsilon^{\frac{1}{2}}$ 程度にすると誤差が $\varepsilon^{\frac{1}{2}}$ 程度で一番小さくなることが知られている[*9]．

微分の公式を使わなければならないのは，微分を解析的に出すのが面倒な場合だ．単純な三角関数や指数・対数関数くらいではそもそも導関数を導ける．したがって，もとの関数 $f(x)$ をいじることによってこの誤差を小さくすることは無理がある．

誤差を回避する方法としては，2 次の微分の公式を使うという方法がある．

$$f'(x) = \lim_{h\to 0}\frac{f(x+h)-f(x-h)}{2h}$$

という公式を用いれば，$h=\varepsilon^{\frac{1}{3}}$ 程度にすると誤差が $\varepsilon^{\frac{2}{3}}$ 程度になる[*10]．

実際は微分の際に桁落ち誤差が発生するよ，ということくらいを知っておけばいい．あとは教科書の p.127 にのっているグラフは非常に有名なグラフなので，頭の片隅に止めておくのがよいだろう．

4.3.3 2 次方程式の解

2 次方程式 $ax^2+bx+c=0 (a>0)$ の解が

$$x = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

[*7] ただ，どこまで正確な計算ができているかということは別の議論になるが，平方根の結果が正しく出る有効数字の分は正しく計算できることが想定される

[*8] h があまり小さくなくて本来の微分の値から離れてしまう誤差は，このあとの節で説明する「打ち切り誤差」である．

[*9] テイラー展開を考えればすぐにわかる．

[*10] ほかに Richardson 加速を用いればもっと大きな h からもっと誤差の小さい $f'(x)$ を導くことができるが，そんなことは情報科学科に進まない限り知らなくてよいし，たかが微分ごときでそんな大げさなことはしない．

であることは周知の事実であろう．ここで $4ac$ が b^2 に比べて非常に小さいとき，$\sqrt{b^2-4ac}$ と b の差が非常に小さくなる[*11]ため，$-b+\sqrt{b^2-4ac}$ を計算するときに桁落ち誤差が発生してしまう．

これを回避するには，解と係数の関係を使えばよい．すなわち，$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}$ と $x_1x_2=\frac{c}{a}$ を使ってやればよい[*12]．

桁落ち誤差は引き算によってしか起こらないため，このように式変形などを工夫することで避けることができる．

4.4 情報落ち誤差

有効数字から外れるくらいちっちゃいところで足したり引いたりしても，無視されちゃうことで起こる誤差．$2.0\times10^{10}+0.30=2.0\times10^{10}$ みたいな感じ．たんに情報落ちともいう．

1 回の足し引きくらいなら問題ない[*13]のだが，何回も何回も足して蓄積していってもとの有効数字内でも変化が出るような場合だと問題になる．たとえば，

$$S=\sum_{k=1}^{\infty}\frac{1}{k^2}$$

という計算を実直にしようと考える[*14]と，倍精度では $k=10^8$ くらいになったところで S の値が変わらなくなってしまう．これを防ぐには，絶対値の小さなものから計算するとよい．すなわち $k=10^{10}$ くらいからさかのぼって足しあわせていけば，小さな数の蓄積がちゃんと最後の大きな S に反映される．

テイラー展開により計算する場合などでは注意が必要である．

4.5 打ち切り誤差

離散化誤差，切り捨て誤差ともいう．

テイラー展開による計算などで，無限大まで計算したいところを途中までで打ち切ってしまうことにより，本当の収束値から離れた値が出てしまう，という誤差．たとえば，

$$\sin x=\sum_{n=1}^{\infty}\frac{(-1)^n}{(2n+1)!}x^{2n+1}$$

である[*15]．これで sin を計算しようと思うが，時間は有限なので $n=\infty$ まで計算することは不可能だ．途中で打ち切らねばならない．その n をどう決めるかは浮動小数点の精度と相談しながらになる．

なお微分の公式の h があまり小さくないことによる誤差も，打ち切り誤差とよんでいる．2 次の微分の公式を使えばある程度回避できることが知られている．

[*11] これは b が正のときに限る．b が負の場合は $b+\sqrt{b^2-4ac}$ を計算するときに桁落ち誤差が発生する．このときの桁落ち誤差の回避も，誤差が出にくい方の解をそのまま出してから，別の解に関して解と係数の関係を使ってやればよい．

[*12] $x_1+x_2=-b/a$ を使ってもできそうだが，桁落ち誤差が発生する可能性があるので避けた．

[*13] 月と地球の表面の距離と，月と地球上に立ってる人の頭のてっぺんの距離の違いなど誰も考えない．

[*14] なお，この値はフーリエ級数を上手く使うと $\pi^2/6$ に収束することが示せる．

[*15] そのままマクローリン展開しただけ．

4.6 この章のまとめ

試験ではよく誤差の内容を聞かれる．名前を答えさせることもあった．（オーバーフロー・アンダーフローをのぞいた）4 種類の誤差の名前とその内容および原因をしっかり把握しておけば試験では問題ないだろう．

第 5 章　数値計算

ここからは本格的な数値計算に入る．前章で述べた誤差の概念をところどころ要する．特に 1 次方程式に関してはしっかり理解しよう．

5.1　乱数

乱数は暗号化技術，シミュレーションなどに欠かせないものである．この後で積分に応用することも考える．だがここではそこまで細かいことを考えない．

乱数[*1]は 2 通りの方法で区別できる．その区別だけを知っておけばよい．

5.1.1　真の乱数と擬似乱数

乱数の定義は，それまで出てきた乱数から推定がまったく不可能である，すなわち独立している数のことである．サイコロを 5 回振ってすべて 6 が出ようが 6 が一度も出なかろうが，次に 6 が出る確率はいつでも 6 分の 1 であり，次に何の数字が出るかは誰にも予想し得ない．これが（真の）乱数だ．

しかしコンピュータが乱数を生成しようとするとそうもいかない．コンピュータには適当なことが苦手なのだ．だから，統計的に乱数だと認められるような数列を，**それまで出てきた数字から**作る必要がある．これまで登場した数字から次の数字を決めるようでは，そんなもの乱数ではない．でも見た目的には，そして統計的には乱数である．こういったコンピュータの作る乱数を**擬似乱数**（pseudorandom number）という．

擬似乱数の生成アルゴリズム*

教科書では紹介されていないが一応触れておこう．

現在最も多く使用されている乱数のアルゴリズムは，1997 年に松本眞と西村拓士によって開発されたメルセンヌ・ツイスタというアルゴリズムである．

生成が非常に高速であること，（同じ数列に戻ってくるまでの）周期が $2^{19937}-1 \fallingdotseq 4.31 \times 10^{6001}$ と長いこと，統計的に十分ランダムであることから，このアルゴリズムが多く用いられている．しかし線形漸化式を使っているために予測可能であるから暗号化には向かない．

5.1.2　一様乱数と正規乱数

一様乱数は，分布をとると一様になるやつ．サイコロを 1 つ振ってできる乱数がこれ．

正規乱数は，分布をとると正規分布になるやつ．平均値だと確率が大きく，平均値から離れると確率が少ない．サイコロを同時に複数振ってその和から乱数を作るとこうなる．

その他もいろいろ作れると思うけどまあ……お察しください．

[*1] 本来は「乱数列」などと言うべきなのであろうが，言葉の区別が面倒になりそうなのでここではすべて「乱数」で統一した．

5.2 数値積分

数値積分は基本的に，原始関数が求められないなど数学的に解くに厳しい関数を扱う際に用いる．計算自体はリーマン積分の定義通り，面積を足しあわせていっているだけである．ありがたいことに 1 変数関数しか扱わないので安心しよう．

計算式自体が煩雑なため試験で聞かれることはあまりないが，出ることもあるので把握しておこう．もちろん丸暗記必須ではなく理論だてて覚えられるぞ．

5.2.1 台形公式

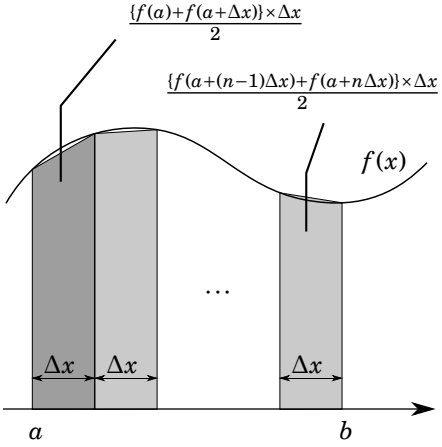
$\int_a^b f(x)dx$ を右の図のように n 個の台形を集めて求めることを考えよう．

ここで $a+n\Delta x=b$ となるように Δx と n をとる．そうすれば台形の数が n 個になりその高さが Δx となるからだ．

左[*2]から数えて k 番目の台形の面積は，上底 $f(a+(k-1)\Delta x)$，下底 $f(a+k\Delta x)$，高さ Δx であるから

$$\frac{\Delta x}{2}\{f(a+(k-1)\Delta x)+f(a+k\Delta x)\}$$

であり，これを 1 番目から n 番目まで足し合わせるので，



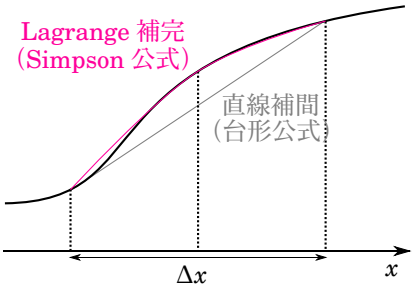
$$\begin{aligned}\int_a^b f(x)dx &\approx \sum_{k=1}^{n}\frac{\Delta x}{2}\{f(a+(k-1)\Delta x)+f(a+k\Delta x)\}\\ &=\frac{\Delta x}{2}\{f(a)+2f(a+\Delta x)+2f(a+2\Delta x)+\cdots+2f(a+(n-1)\Delta x)+f(a+n\Delta x)\}\\ &=\Delta x\left\{\frac{f(a)+f(b)}{2}+\sum_{k=1}^{n-1}f(a+k\Delta x)\right\}\end{aligned}$$

となる．これが世に言う**台形公式**（trapezoid rule）である．なお台形則ともいう．

5.2.2 Simpson 公式

厳密には合成 Simpson 公式という場合もある．

台形公式ではなめらかな関数の時になんとなく効率が悪い気がする．そこで，台形公式のときに 2 点間を直線で近似していたものを，ここでは Lagrange 補完という方法で 2 次関数で近似しようというものである．



[*2] 左端の台形を 1 番目の台形と呼んでいる．

Lagrange 補完

ここでは 3 点 x_1, x_m, x_2 から 2 次関数を近似する Lagrange 補完を考える．考え方は簡単だ．$\tilde{f}(x_1) = f(x_1)$，$\tilde{f}(x_m) = f(x_m)$，$\tilde{f}(x_2) = f(x_2)$ となるような 2 次関数 $\tilde{f}(x)$ を考えればいいだけだ．そのためには

$$\tilde{f}(x) = \frac{(x-x_2)(x-x_m)}{(x_1-x_2)(x_1-x_m)} f(x_1) + \frac{(x-x_1)(x-x_2)}{(x_m-x_1)(x_m-x_2)} f(x_m) + \frac{(x-x_1)(x-x_m)}{(x_2-x_1)(x_2-x_m)} f(x_2)$$

とすればよい．

第 1 項は $x = x_2, x_m$ のときには係数が 0 となり反映されず，逆に $x = x_1$ のときに係数が 1 となるようになっている．第 2 項も第 3 項も同じようになっており，結果的に $x = x_1, x_m, x_2$ では $\tilde{f}(x)$ の値が $f(x)$ と等しくなるようになっている．

Lagrange 補完を積分する

上で考えた $\tilde{f}(x)$ を $[x_1, x_2]$ の範囲で積分すること，すなわち $\int_{x_1}^{x_2} \tilde{f}(x)dx$ を考えよう．ここで，x_m は x_1 と x_2 の中点，すなわち $x_m = \frac{x_1+x_2}{2}$ としよう．

まず，それぞれの項に対して $[x_1, x_2]$ で積分しよう．まず第 1 項．

$$\int_{x_1}^{x_2} \frac{(x-x_2)(x-x_m)}{(x_1-x_2)(x_1-x_m)} f(x_1)dx = \frac{f(x_1)}{(x_1-x_2)(x_1-x_m)} \int_{x_1}^{x_2} (x-x_2)(x-x_m)dx$$

ここで $x = x_1 + t$ と置換し，$x_2 - x_1 = 2h$ とすれば，

$$\begin{aligned} &= \frac{f(x_1)}{2h^2} \int_0^{2h} (t-2h)(t-h)dt \\ &= \frac{f(x_1)}{2h^2} \left[\frac{1}{3}t^3 - \frac{3h}{2}t^2 + 2h^2 t \right]_0^{2h} \\ &= \frac{h}{3} f(x_1) \end{aligned}$$

同様に

$$\int_{x_1}^{x_2} \frac{(x-x_1)(x-x_2)}{(x_m-x_1)(x_m-x_2)} f(x_m)dx = \frac{4h}{3} f(x_m)$$
$$\int_{x_1}^{x_2} \frac{(x-x_1)(x-x_m)}{(x_2-x_1)(x_2-x_m)} f(x_2)dx = \frac{h}{3} f(x_2)$$

となる．これを全部足し合わせれば

$$\int_{x_1}^{x_2} \tilde{f}(x)dx = \frac{x_2-x_1}{6} \{f(x_1) + 4f(x_m) + f(x_2)\}$$

となる．

得られた積分値を足し合わせる

台形公式と同じように，分割を n 個にするために $a + n\Delta x = b$ として $\int_a^b f(x)dx$ を考えよう．

左から数えて k 番目の図形の面積は, 上の積分公式に $x_1 = a+(k-1)\Delta x$, $x_m = a+(k-\frac{1}{2})\Delta x$, $x_2 = a+k\Delta x$ を代入して[*3],

$$\int_{a+(k-1)\Delta x}^{a+k\Delta x} \tilde{f}_k(x)dx = \frac{\Delta x}{6}\left\{f(a+(k-1)\Delta x)+4f\left(a+\left(k-\frac{1}{2}\right)\Delta x\right)+f(a+k\Delta x)\right\}$$

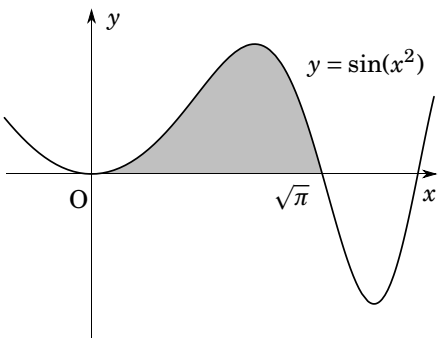
となる．これを 1 番目から n 番目まで足し合わせれば,

$$\begin{aligned}\int_a^b f(x)dx &\approx \sum_{k=1}^{n}\int_{a+(k-1)\Delta x}^{a+k\Delta x}\tilde{f}_k(x)dx \\ &= \sum_{k=1}^{n}\frac{\Delta x}{6}\left\{f(a+(k-1)\Delta x)+4f\left(a+\left(k-\frac{1}{2}\right)\Delta x\right)+f(a+k\Delta x)\right\} \\ &= \frac{\Delta x}{6}\left[f(a)+4f\left(a+\frac{1}{2}\Delta x\right)+f(b)+2\sum_{k=1}^{n-1}\left\{f(a+k\Delta x)+2f\left(a+\left(k+\frac{1}{2}\right)\Delta x\right)\right\}\right]\end{aligned}$$

となる．これが Simpson 公式だ．めっちゃ汚いけど．

台形公式と Simpson 公式の比較

さて，ここで出てきた 2 つの公式を使って実際に積分値を求めてみよう．ここでは $\int_0^{\sqrt{\pi}}\sin(x^2)dx$ [*4]を使ってみよう（右図の塗りつぶされた部分）.



実際の値は 0.894831469484 であることがわかっている．これは有効数字内で十分に正確な値である．台形公式と Simpson 公式を用いて n を動かしながら積分値を計算してみると以下のようになる（太字が理論値と一致している桁である）.

分割数 n	台形公式	Simpson 公式
10	0.**8**85489513242	0.**8948**46843350
100	0.**894**738657911	0.**894831**471011
1000	0.**894830**541429	0.**894831469484**
10000	0.**894831460**204	0.**894831469484**

n を増やしていくほど理論値に近づいていることがわかる．まあ当然のことだろう．台形公式よりも Simpson 公式の方が速く理論値に近づいていることもわかる．これは $\sin(x^2)$ という関数が Simpson 公式で計算するのに適しているからだ．必ずしも Simpson 公式のほうがいいというわけでもない．無限積分などは台形公式のほうが得意だとされている.

なお振動する関数の積分などは他の手段を考えるのが一般的である．二重指数関数型数値積分公式などを用いて効率化することも考えるとさらに捗る.

[*3] 教科書における Δx の定義とは異なるので注意．本書における Δx を $2\Delta x$ とすれば教科書と同じ式が得られる.

[*4] $S(x) = \int_0^x sin(t^2)dt$ は光学系で用いられるフレネルの積分とよばれる有名な積分式である．なお $\lim_{x\to\infty} S(x) = \sqrt{\pi/2}$ であることが知られている.

台形公式や Simpson 公式で考えられる誤差

分割数が少ないと理論値から遠くなる．この誤差は**打ち切り誤差**だ．

もう 1 つ重要な誤差がある．分割数を多くし過ぎると総和の計算をしているとき，途中まで足しあわせて得られた値よりも，次に足しあわせる値があまりにも小さくなる可能性がある．これによって**情報落ち誤差**が発生することが考えられる．

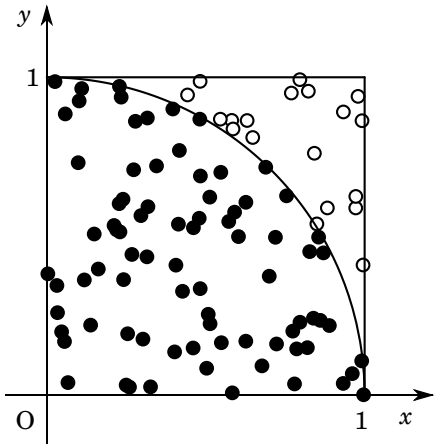
5.2.3 Monte Carlo 法

Monte Carlo 法は，乱数をもちいて様々なシミュレーションを行う方法をいう．カジノで有名なモナコ公国の町モンテカルロからその名がとられている．数値積分はその 1 つの例である．とくに多次元の際に，台形公式や Simpson 公式よりも正確に求めることができるとされている．実際は積分というより面積や体積を解析的に出せない場合などに用いられる．

たとえばここでは半径 1 の円の面積を求めることを考えよう．ここでは簡単のためその 4 分の 1 を求めたあと 4 倍することにする．

右図のように 1×1 の正方形の中にランダムに点を打つ．そのうち面積として計算されるべき領域に入った点の個数の割合を求めればよい．右図の場合，点が全部で 100 個でそのうち四分円の中に入っているもの（黒点）が 81 個，入っていないもの（白点）が 19 個であるから，四分円の面積は

$$(1\times 1)\times\frac{81}{100}=0.81$$



と見積もられる．したがって半径 1 の円の面積（円周率）は $0.81\times 4=3.24$ くらいだとわかる．表 5.1 のとおり，当然のごとく試行回数（打つ点の数）が増えれば増えるほど実際の値に近くなる．しかしその分時間もかかるため，適切な試行回数を考える必要がある．

試行回数 n	見積もられた円周率	誤差
100	3.24	3.1%
1000	3.108	1.1%
10000	3.1588	0.5%
100000	3.13632	0.2%
1000000	3.142152	0.02%
10000000	3.1411152	0.03%

表 5.1　モンテカルロ法によって円周率を見積もった結果

5.3 多元 1 次方程式

多元 1 次方程式を解くためのアルゴリズムは，Gauss 消去法，Gauss-Jordan 法，LU 分解といった消去法，Gauss-Seidel 法，Jacobi 法，SOR 法といった反復法などがある．規模の小さいものであれば Gauss 消去法で十分だ．だが規模が大きくなると並列化なども考える必要があるため反復法のほうが適していることが多い．

ここでは Gauss 消去法や Gauss-Jordan 法，およびそこにあらわれる誤差とその回避法を説明していく．

5.3.1 Gauss 消去法と Gauss-Jordan 法

簡単にいえば，$\boldsymbol{Ax}=\boldsymbol{b}$ を解く際に，A を三角行列までもっていくのが Gauss 消去法，さらに対角化までやっちゃうのが Gauss-Jordan 法だ．

さて，細かく見ていこう．これらの消去法では，1 つの式を定数倍する，別の式に定数倍を加える，の 2 つの操作によりなんとかして解が簡単に出せる状態まで持っていく．実際にここでは次の 3 元 1 次方程式を解くことを考えよう．

$$\begin{cases} 3x + 2y - 4z = 1 \\ x + 5y + 7z = 4 \\ 2x - y - 6z = 1 \end{cases}$$

Gauss 消去法

1. 第 1 式の $-\frac{1}{3}$ を第 2 式に，$-\frac{2}{3}$ を第 3 式に加える．

$$\begin{cases} 3x + 2y - 4z = 1 \\ \frac{13}{3}y + \frac{25}{3}z = \frac{11}{3} \\ -\frac{7}{3}y - \frac{10}{3}z = \frac{1}{3} \end{cases}$$

2. 第 2 式の $\frac{7}{13}$ を第 3 式に加える．

$$\begin{cases} 3x + 2y - 4z = 1 \\ \frac{13}{3}y + \frac{25}{3}z = \frac{11}{3} \\ \frac{15}{13}z = \frac{30}{13} \end{cases}$$

3. 第 1 式を $\frac{1}{3}$ 倍に，第 2 式を $\frac{3}{13}$ 倍に，第 3 式を $\frac{13}{15}$ 倍にする．

$$\begin{cases} x + \frac{2}{3}y - \frac{4}{3}z = \frac{1}{3} \\ y + \frac{25}{13}z = \frac{11}{13} \\ z = 2 \end{cases}$$

4. 変数を逆順に求めていく**後退置換**を行う．

$$\begin{cases} z = 2 \\ y = \frac{11}{13} - \frac{25}{13}z = -3 \\ x = \frac{1}{3} + \frac{4}{3}z - \frac{2}{3}y = 5 \end{cases}$$

5. 完成♪

Gauss-Jordan 法

1. 第 1 式の $-\frac{1}{3}$ を第 2 式に，$-\frac{2}{3}$ を第 3 式に加える．（これは Gauss 消去法と同じ）

$$\begin{cases} 3x + 2y - 4z = 1 \\ \frac{13}{3}y + \frac{25}{3}z = \frac{11}{3} \\ -\frac{7}{3}y - \frac{10}{3}z = \frac{1}{3} \end{cases}$$

2. 第 2 式の $-\frac{6}{13}$ を第 1 式に，$\frac{7}{13}$ を第 3 式に加える．

$$\begin{cases} 3x - \frac{102}{13}z = -\frac{9}{13} \\ \frac{13}{3}y + \frac{25}{3}z = \frac{11}{3} \\ \frac{15}{13}z = \frac{30}{13} \end{cases}$$

3. 第 3 式の $-\frac{34}{5}$ を第 1 式に，$-\frac{65}{9}$ を第 2 式に加える．

$$\begin{cases} 3x = 15 \\ \frac{13}{3}y = -13 \\ \frac{15}{13}z = \frac{30}{13} \end{cases}$$

4. 第 1 式を $\frac{1}{3}$ 倍に，第 2 式を $\frac{3}{13}$ 倍に，第 3 式を $\frac{13}{15}$ 倍にする．

$$\begin{cases} x = 5 \\ y = -3 \\ z = 2 \end{cases}$$

5. 完成♪

まあこんな感じ．結局，式同士を足し引きしてうまく式の中の変数の数を減らしていくってことだ．頭のいいキミたち東大生なら把握したよね．

手計算なら適宜式を定数倍するとか，どの式でどの変数を残すとか，うまい方法が考えられるが，プログラムをするとなるとそうもいかない．そのため基本的には「第 n 式で第 n 変数を残す」という操作をするのが基本だ．

5.3.2 ピボット選択

教科書では **pivoting** と紹介されていた．ちょっとネーミングがイヤなのでピボット選択とよぶことにする[*5]．枢軸選択ともいう．

先ほど見たような機械的な操作だとバグが起きる可能性がある．たとえば次のような例だ．

$$\begin{cases} y + z = -1 \\ x + 5y + 7z = 4 \\ 2x - y - 6z = 1 \end{cases}$$

[*5] そもそもいろいろな名前をアルファベットで書くのが嫌いだ．「Gauss-Jordan 法」だって「ガウスジョルダン法」と書けばよいではないか．日本にはカタカナという素晴らしい道具がある．教科書にならって一応アルファベットで書いてはいるが，カタカナを忘れない日本人になりたい．

ここで第 1 式に第 1 変数 x を残そうとしても無理だ．機械的にやろうとすると，$-\frac{1}{0}$ なんて計算をさせてしまうことになる．これでは答えが出ない．

また，次のような例を考えてみよう．解はさっきと同じだ．

$$\begin{cases} 0.0007x - 0.0005y + 4z = 8.005 \\ x + 5y + 7z = 4 \\ 2x - y - 6z = 1 \end{cases}$$

ここで第 2 式・第 3 式の第 1 変数 x を消去しようとして第 1 式を $\frac{1}{0.0007} = 1429$ 倍すると，他の変数の係数や定数項が非常に（絶対値が）大きな値になる．すなわち，第 k 式の第 k 変数の係数（の絶対値）が他の式でのそれよりも非常に小さい場合，つまり 0 に近い場合，その第 k 式が大きな数倍されてしまう．すると絶対値が大きいものと小さいものを加減算したときに小さいものが無視されてしまう誤差「情報落ち誤差」が起こってしまう．

0 での除算および情報落ち誤差を避けるためには，計算するときに残す変数の係数（の絶対値）が 1 番大きくなるような式をもってくるのがよい．この操作をピボット選択という．

上の例でいえば，変数 x を残す際，いったん式をしたのように交換する．

$$\begin{cases} 2x - y - 6z = 1 \\ x + 5y + 7z = 4 \\ 0.0007x - 0.0005y + 4z = 8.005 \end{cases}$$

ここからは実際に有効数字 5 桁で計算していってみよう．左がピボット選択なし，右がピボット選択ありだ．

そのままの状態．

$$\begin{cases} 0.0007x - 0.0005y + 4z = 8.005 \\ x + 5y + 7z = 4 \\ 2x - y - 6z = 1 \end{cases}$$

第 1 式の $-\frac{1}{0.0007} = -1428.6$ 倍を第 2 式に，$-\frac{2}{0.0007} = -2857.1$ 倍を第 3 式に加える．

$$\begin{cases} 0.0007x - 0.0005y + 4z = 8.005 \\ 5.7143y - 5707.4z = -11432 \\ 0.42855y - 11434z = -22870 \end{cases}$$

左の式のほうが大きな数字や細かい数字がたくさん出てきて混沌としている．明らかに丸めがたくさん行われているのは確かだ．右の式でも第 3 式では丸めが起きているが，あまり気になるほどではない．続けていこう．

第 2 式の $\frac{0.0005}{5.7143} = 0.000087500$ 倍を第 1 式に，$-\frac{0.42855}{5.7143} = -0.074996$ 倍を第 3 式に加える．

$$\begin{cases} 0.0007x + 3.5006z = 7.0047 \\ 5.7143y - 5707.4z = -11432 \\ - 11006z = -22013 \end{cases}$$

第 1 式と第 3 式を交換しておいた状態．

$$\begin{cases} 2x - y - 6z = 1 \\ x + 5y + 7z = 4 \\ 0.0007x - 0.0005y + 4z = 8.005 \end{cases}$$

第 1 式の $-\frac{1}{2} = -0.5$ 倍を第 2 式に，$-\frac{0.0007}{2} = -0.00035$ 倍を第 3 式に加える．

$$\begin{cases} 2x - y - 6z = 1 \\ 5.5y + 10z = 4.5 \\ -0.00015y + 4.0002z = 8.0054 \end{cases}$$

第 2 式の $\frac{1}{5.5} = 0.18182$ 倍を第 1 式に，$\frac{0.00015}{5.5} = 0.000027273$ 倍を第 3 式に加える[*6]．

$$\begin{cases} 2x - 4.1818z = 1.8182 \\ 5.5y + 10z = 4.5 \\ 4.0005z = 8.0055 \end{cases}$$

第 3 式の $\frac{3.5006}{11006} = 0.00031806$ 倍を第 1 式に，$-\frac{5707.4}{11006} = -0.51857$ 倍を第 2 式に加える．

$$\begin{cases} 0.0007x & & & = & 0.0032452 \\ & 5.7143y & & = & -16.719 \\ & & -11006z & = & -22013 \end{cases}$$

第 3 式の $\frac{4.1818}{4.0005} = 1.0201$ 倍を第 1 式に，$-\frac{10}{4.0005} = -2.4997$ 倍を第 2 式に加える．

$$\begin{cases} 2x & & & = & 9.9846 \\ & 5.5y & & = & -15.511 \\ & & 4.0005z & = & 8.0055 \end{cases}$$

さあ，答えまで近づいてきた．この次点で，左の方は値の大きさが極端だが，右の方は落ち着いていることがわかる．

答えを出す．

$$\begin{cases} x &= 4.636 \\ y &= -2.9258 \\ z &= 2.0001 \end{cases}$$

答えを出す．

$$\begin{cases} x &= 4.9923 \\ y &= -2.8202 \\ z &= 2.0011 \end{cases}$$

うーん．なかなか微妙な感じだ．y と z に関しては左の方が答えに近いが，それでも x の方の誤差が右よりもかなり大きい．とりあえずピボット選択があったほうが全体的にはマシになるってことはわかったでしょ?（無理矢理）[*7]

5.4 この章のまとめ

この章では 3 つの事柄を扱った．

■乱数　該当箇所でも書いたとおり，「擬似乱数 or 真の乱数」と「一様乱数 or 正規乱数」の 2 つの区別だけを理解しておけばよい．これは**モンテカルロ法**で用いられることも覚えておこう．

■数値積分　台形公式，Simpson 公式のそれぞれを概念として理解しておこう．できれば導出までやってみよう．モンテカルロ法は黒点と白点が散らばった絵を理解できていれば OK．多次元のときに特に有効であることも把握しておこう．

■多元 1 次方程式　Gauss 消去法と Gauss-Jordan 法で多元 1 次方程式を求められるようにしておこう．あとは pivoting が「第 n 変数の係数が第 n 式で一番大きくなるようにする」操作で，これは 0 で割ったり情報落ち誤差が起きたりするのを防ぐためにあることを覚えておこう．

[*6] 本来はここでもピボット選択を行うため第 2 式と第 3 式を入れ替えることがあるが，y の係数の絶対値は第 2 式が一番大きいのでそのままとした．

[*7] まあきれいな例を作るのは難しいってことだよ（言い訳）．

LU 分解法

多元 1 次方程式の解き方の 1 つだ．知っておいて損はない．

連立 1 次方程式は

$$A\boldsymbol{x} = \boldsymbol{b}$$

と表せる．ここで，

$$A = LU$$

となるような下三角行列 L と上三角行列 U があるとしよう．すると

$$A\boldsymbol{x} = LU\boldsymbol{x} = \boldsymbol{b}$$

とできる．ここで $U\boldsymbol{x} = \boldsymbol{y}$ とすれば

$$L\boldsymbol{y} = \boldsymbol{b} \qquad と \qquad U\boldsymbol{x} = \boldsymbol{y}$$

の 2 式が連立 1 次方程式としてでてくる．L と U は三角行列だから，$L\boldsymbol{y} = \boldsymbol{b}$ から $\boldsymbol{y}$ を計算したあと $U\boldsymbol{x} = \boldsymbol{y}$ を計算すれば容易に解を得ることができる．

さて，問題は L と U をどうやって導くか．そもそも未知数が $n^2 + n$ 個存在するのに式は n 個しかないため一意には決まらない．そこで L の対角成分 n 個をすべて 1 としてしまう．すなわち，

$$LU = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ a_{21} & 1 & 0 & & 0 \\ a_{31} & a_{32} & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 1 & 0 \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ 0 & b_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & b_{n,n-1} \\ 0 & \cdots & 0 & b_{nn} \end{pmatrix}$$

とする．

ここからはドゥーリトル法とクラウト法の 2 通りの方法で L と U を計算できるが，ごちゃごちゃと成分計算すればわりと簡単に計算できたりもする．

LU 分解は連立 1 次方程式だけでなく，逆行列の計算や行列式の計算にも応用できる．また LU 分解にもピボット選択が不可欠である．その辺の細かいことはまたいつの日か．

第 6 章　動的計画法

informatical reality

動的計画法というのは，問題をいくつかの部分で分けて解いている間に同じ計算をしようと思ったとき，前にでた結果を再利用する，というアルゴリズムの組み立て方のことをいう．いきなり全体に対して取り掛かろうとせずに，小さくわけて順番にやっていくと上手くいきやすい．とにかく，コンピュータがなるべくラクになるように計算をサボらせることを考えるのがポイント．

6.1　アラインメント

ここでの例としてみんなの嫌いなアラインメントを挙げよう[*1]．

ここで紹介するアラインメントは，2 つの文字列の似ている度を計算するものである．たとえば「adce」と「ace」なら 4 点，「adce」と「abcd」なら 2 点，などといったふうに計算できる．

6.1.1　点数をどうやって求める?

2 つの文字列を，文字の順番は変えないように縦に並べて比較する．

一致しているところが多ければ多いほど高いとしたい．だから縦で文字が一致してたら 2 点（一致得点），一致していなかったら-1 点（不一致得点）にしよう．

あとは同じ長さにしたい．同じ長さにするためには空白が必要だから，空白を適宜入れられるようにしておこう．もし片方に空白を入れてしまったらそれは-2 点（ギャップ得点）ということにしよう．ただ，空白と空白が並ぶことはないとしよう．それは詰めればいい話だからね．

2 つの文字列の並べ方はいくつかある．たとえば adce と ace を比較するとき，

```
ad-ce
-ac-e
```

なら $(-2)+(-1)+(-2)+(-2)+2=-5$ 点，

```
adce
a-ce
```

なら $2+(-2)+2+2=4$ 点となる．その中で一番大きい点数のものが最大点として評価される．つまり，いろんな並べ方で試してみて一番いいヤツを点数とするってことだ．

ただ全部のパターンを調べるのはキツい．っていうか全部のパターンが何なのかすらわからない．だからこんな方法を考えてみた．

6.1.2　再帰的な方法

簡単にいえば，後ろのほうから攻めていく方法である[*2]．

同じように adce と ace に関して考えよう．一番後ろは

```
e
e
```

```
e
-
```

```
-
e
```

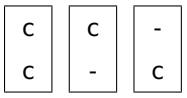
の 3 通りがあるはずだ．

[*1] 本来はパターン認識の入門として扱うべきなのだが，パターン認識など扱うような内容になっていないため，ただ動的計画法の 1 つの例として挙げた．ただ，パターン認識としてのアラインメントは非常に大事な概念で，情報科学の初歩として欠かせないものである．

[*2] 攻められるならシタからよりウエからがいいよね．

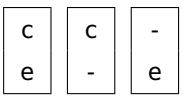
e / e で終わるとき

まずこの時点で 2 点入った．やったね☆お兄ちゃん！

このあとは adc と ac の点数を考えればいい．そのときやっぱり終わり方は  の 3 パターンがある．

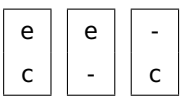
e / - で終わるとき

まずこの時点で-2 点になっちゃった．残念．

このあとは adc と ace の点数を考えればいい．そのときやっぱり終わり方は  の 3 パターンがある．

- / e で終わるとき

これもこの時点で-2 点になっちゃった．残念．

このあとは adce と ac の点数を考えればいい．そのときやっぱり終わり方は  の 3 パターンがある．

こういうふうに繰り返していけば，どこかでどっちかの文字列がなくなる．そしたらあとは空白で埋めてしまえばいい．両方同時になくなったらそのままでめでたしめでたし．

6.1.3 再帰的な方法の欠点

実はこれ，無駄な計算をしてしまう箇所がある．

e / e で終わるときにも e- / -e で終わるときにも adc と ac の点数を呼び出してそれに 2 点もしくは-4 点加える，という計算をしなければいけなくなる．それぞれ独立に呼び出されるから，どちらかで計算済みでも結果の再利用は難しい．

じゃあどうやって計算をサボるか．こんな方法を考えよう．

6.1.4 動的計画法

再帰的な方法と違って頭から考えていく．このときこんな表を持ってこよう．

		a	c	e
	0			
a				
d				
c				
e				

さて，この表に何を埋めるのかを説明しよう．便宜的に上から 3 行目のことを「a の行」，左から 4 列目のことを「c の列」などという．

○の行と□の列の空欄には，「1 つめの文字列の○まで と 2 つめの文字列の□まで のアラインメントの点数を計算したときの最高得点」を入れる．たとえば d の行 c の列であれば，ad と ac のアラインメントの最高得点である 1 点（ad / ac のとき）を入れる．

このときすぐにわかるのは，片方の文字列になにもないときだ．たとえば 1 行目（なにもない行）と 2 列目（c の列）の空欄には，「なにもない文字列 と ac のアラインメントの点数を計算したときの最高得点」を入れることになるが，これは唯一 -- / ac の-4 点のみしかない．同じようにすればとりあえずこんな風に表が埋まる．

		a	c	e
	0	-2	-4	-6
a	-2			
d	-4			
c	-6			
e	-8			

さて，これで埋まってない部分はどう求めればよいか．

まず a の行 a の列のことを考えよう．a と a のアラインメントの最高得点は，a / a （2 点），a- / -a (-4 点)，-a / a- (-4 点) のうちの最大値の 2 点だ．a / a は，両方とも何もないアラインメント + a / a であり，両方とも何もないアラインメントというのは 1 行 1 列目（0 の部分）を指す．また，a- / -a は，a / - + - / a であり，a / - というのは 2 行 (a の行) 1 列目を指す．そして，-a / a- は，- / a + a / - であり，- / a というのは 1 行 2 列（a の列）目を指す．

すなわち，○行□列の空欄は，左上 + ○ / □ と 左 + - / □ と 上 + ○ / - のアラインメントの最大値が入るのだ!

a の行 a の列は，0（左上 a/a → 2 点） + 2 = 2 点，−2（左 - / a → -2 点） + −2 = −4 点，−2（上 a / - → -2 点） + −2 = −4 点の最大値である 2 点が入る．

		a	c	e
	0	-2	-4	-6
a	-2	2		
d	-4			
c	-6			
e	-8			

a の行 c の列は，−2（左上 a/c → 1 点） + 1 = −1 点，2（左 - / c → -2 点） + −2 = 0 点，−4（上 a / - → -2 点） + −2 = −6 点の最大値である 0 点が入る．

さて，みんなで埋めてみよう! 全部埋めるとこんなかんじになる．

		a	c	e
	0	-2	-4	-6
a	-2	2	0	-2
d	-4	0	1	-1
c	-6	-2	2	0
e	-8	-4	0	4

こうすれば同じ計算を 2 度しなくてもアラインメントの点数を求めることができた．

6.1.5 トレースバック

アラインメントの得点は 4 点だった．さきほどの表からどのような文字の並べ方をすれば 4 点になるかを調べることができる．この調べ方をトレースバックという．

まず最後の点数に注目しよう．4 点だ（見ればわかる）．4 点は，左上（2）+ e と e の一致得点（2）と左（0）+ ギャップ得点（-2）と 上（0）+ ギャップ得点（-2）のうち，左上 + e と e の一致得点 に該当する．したがって，‘e’ の部分では 2 つが揃っている（一致している）ということがわかる．

左上から来たことがわかったので，次にその左上の点数を見てみよう．2 点だ（見ればわかる：2 回目）．さっきと同じように考えると，これも 左上 + c と c の一致得点 に該当するようだ．したがって，‘c’ の部分でも 2 つが揃っている（一致している）ということがわかる．

さらに左上から来たことがわかったので，またその点数を見てみよう．0 点だ（見ればわかる[*3]）．0 点は，左上（-2）+ a と d の不一致得点（-1）と 左（-4）+ ギャップ得点（-2）と 上（2）+ ギャップ得点（-2）のうち，上 + ギャップ得点 に該当する．上に進んだので，d のカップルはいないことになる[*4]．したがって，‘d’ のペアはギャップ（空白である）．

そんでもって上から来たことがわかったので，またその点数を見てみよう．2 点だ（見ればわかる[*5]）．これも 左上 + a と a の一致得点 にあたるので，‘a’ の部分も 2 つが揃っている．

これで一番左上の 0 に到達した．ここがゴール．ここまでを合わせると，4 点は

```
adce
a-ce
```

という並べ方をすればとれることがわかった．

6.2 この章のまとめ

動的計画法を用いる問題はいろいろある．ナップサック問題，最短路問題，最長共通部分列問題など有名なものも多数ある．試験で出る際には，どういった考え方にのっとればラクに解けるかは示してある．問題をちゃんと理解していればちゃんと解けるはずだ．

ただし，アラインメントに関してはそうもいかない．なにせ授業で扱っているからだ．アラインメントに関してはここ 5 年間出題されていないが，今後出題されないとも限らない．どうしても理解できな

[*3] 3 回目．
[*4] 筆者のようだ．
[*5] もういい．

かった人は，表の書き方だけ覚えておこう．

- 何もない行と列に 0 を書く．
- 片方がなにもない行と列は −2 の倍数を順に書いていく．
- あとは，

$$\max\left(\begin{array}{l} \text{左上} + \begin{cases} 2 & \text{行と列の文字が同じとき} \\ -1 & \text{行と列の文字が異なるとき} \end{cases} \\ \text{左} - 2 \\ \text{上} - 2 \end{array}\right)$$

  として埋めていけばよい．
- ただし，一致の点数が 2 点，不一致の点数が-1 点，ギャップの点数が-2 点になっているので，出題されたときに違えばそれに合わせて変えること．

フォントのはなし

フォントの話しようか．まったく本編と関係ないけどね．

欧文書体

この本文では，欧文書体は New Century Schoolbook 使ってる．こう，なんというか無難なフォントというか．いろいろ考えたんだけどこれに落ちついた．

TEX の初期フォントの Computer Modern も嫌いじゃないんだけどね．とくに「n」とかは好き．

ページ数のフォントは Palatino を使ってる．このオールドスタイルの数字．かっこ良くていいよね（厨二）．まあ普通の文章中で使われると読みにくいんだけど．

見出しとかにつかってるサンセリフは Optima．この線の太さがメッチャイイ感じ．超超超イイカンジ．

日本語書体

明朝体は，モリサワの黎ミンを使ってる．決め手は「の」．ヒラギノとかＭＳ 明朝（笑）とかだと，「の」の入りから丸にはいるところが，丸っこくないんだよね．この「くるっ」と丸い感じがイイ．

見出しに使ってるサンセリフのフォントは，タイプバンクの UD タイポス．このなんかなんとも言えない感じが好き．ムニェッとしてる感じが．

ところどころで出てくる丸ゴシックは，モリサワの秀英丸ゴシック．丸ゴシックの中だと筑紫丸ゴシックと並んで好き．

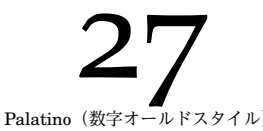
フォント厨にとって TEX を使うこと

種類が少なかったり，設定がめんどくさかったりで，あまりフォント厨に優しいとは言えないんだよね．ただマクロが作れるから Word で作るよりよっぽどラク．InDesign をマスターするのが一番いいとは思うんだが，あれはあれで難しすぎる．勉強したい．

フォント厨は，こんなことを思いながら車内広告を見つめてるのです．

New Century Schoolbook

Computer Modern



Palatino（数字オールドスタイル）

Optima

黎ミン R

UD タイポス 58

秀英丸ゴシック B

第II部
分野別問題集 問題編

この部では，情報科学の過去問を分野別にし，各分野の中で易しい方から順に掲載した．第 0 章に，情報科学の問題の傾向や難易度，またこの過去問集の使い方に関して書いておいたので，そちらを読んでから使って欲しい．情報科学に限らず，こういった理系教科は演習量につきる．第 I 部だけ読んで満足することなく，過去問を解くことでその力をモノにしよう．

第 0 章　情報科学の試験について

informatical reality

試験形式

情報科学の試験は，いつもどおりであれば，2 月の中旬ごろに 2 限の 90 分間で行われる．問題用紙は 2 枚（もしくは両面印刷で 1 枚），答案用紙は 1 枚（両面印刷），計算用紙は 1 枚，持ち込みは不可．大問は 4 つで 1 大問あたりも小問は 3〜4 問であることが多い．

出題傾向

第 1 問・第 2 問が計算量や再帰的定義などの問題，第 3 問が誤差や乱数，第 4 問が動的計画法によるアルゴリズム，となっている．

第 1 問・第 2 問では，計算量を直接求めさせたり，再帰的なアルゴリズムの穴埋めをさせたり，アルゴリズムを書き換えさせたりするような問題が多く出題される．Ruby のコードを直接書かせる問題や，アルゴリズムを文章で説明させる問題が出題されたこともある．とくに第 1 問は全体の中でも基本的で易しい問題であることが多く，得点源となりうる．

第 3 問は，誤差および乱数に関する問題．誤差や乱数の定義を知ってさえいれば書ける記述問題もあり，こういった問題は落とせない．逆に誤差や乱数をからめたアルゴリズムの問題も毎回出題されているが，こちらは少し難易度が高め．

第 4 問は，基本的に動的計画法によるアルゴリズム．動的計画法ではなくただのプログラミングコンテストのような問題が出題されるようなこともある．問題文がかなり長く難易度も高いため，問題の状況や掲載されている Ruby のコードをちゃんと理解して，落ち着いて解く必要がある．ただ，小問による誘導があるので，たとえ苦手であっても最初の方は解ける可能性が高い．

難易度について

計算機プログラミング I から名前が変わり，情報科学の科目ができたのが 2006 年度である．その当時から 2009 年度までは少々難易度が高く理解に苦しむような問題も出題されていた．2010 年度からは授業の方針が「演習中心，テストは $+\alpha$」という形に変更され，問題は比較的易しめになった．

それでもやはり状況把握を問う問題や，アルゴリズムを直接書かせる問題は難易度が高いため，満点を取ることはそう簡単ではない．かなりの**情報科学力**が必要となる．

まずは，誤差や乱数などの定義をしっかりと覚えること．そして Ruby のコードを臆せず理解していくこと．このあたりが大事になってくるだろう．

プログラミングに関して得意であれば，丁寧に問題を解くだけでなく，よりシンプルで例外の発生しにくいアルゴリズムを記述するなどの工夫をすることで，さらなる高得点を狙えるようになるだろう．

過去問について

この過去問集は，情報科学の発足した 2006 年度から出題されたすべての問題，また内容が類する他科目の期末試験の過去問を集めたものである．

はじめの方の易しめの問題それぞれせめて 3〜4 題ずつくらいは解いておくと安心だろう．余裕があったら全部解いちゃおう♪

また，現在の情報科学の試験範囲ではないが，過去に出題されたことのある問題も別途掲載した．こちらは必ずしも解いておく必要はないだろうが，これから情報科学と共に歩むだろうと思う人はやっておいて損しないだろう．

現在の試験範囲からして出題されないもの，また出題に適さない程度の難問に関しては難問マーク*をつけているので参考にしてほしい．

なお過去問は編集の都合上一部改変している場合がある．

第 1 章　計算量・再帰的定義

informatical reality

以下の問題で用いられている Ruby の式の意味は次のとおりである．x && y は x と y の値がともに真の場合にのみ真となる．x || y は x と y の値がどちらか真の場合にのみ真となる．!x は x の真偽を逆転させる．また，関数 min(x,y)，max(x,y) は計算量 $O(1)$ で，x，y のそれぞれ小さい方，大きい方の値を返す．関数 abs(x) は x の絶対値を求める．関数 is_even(x) は x が偶数である場合に真（true）を，それ以外の場合に偽（false）を返す．関数 make1d(n) と Array.new(n) は大きさ n の配列を作る．関数 make2d(h,w) は h 行 w 列の 2 次元配列を作る．関数 rand() は 0 以上 1 未満の一様乱数を 1 つ作る．include(Math) を実行していた場合，PI は円周率を表す定数となる．問題においても解答の際も，これらの関数は定義済みであるとしてよい．

▶ 解答 p.93

問 1　再帰的定義とそれと同値の繰り返しによる定義　〔2012 年度　第 1 問〕

1. 下の関数定義の下で f(5) が何を計算するか述べよ．

```
def f(n)
    if n >= 2
        n * f(n - 1)
    else
        1
    end
end
```

2. 上の小問の関数 f は再帰呼び出しを使って書かれている．再帰呼び出しを使わず while や if, for だけを使って同じ計算をするように f を書き直せ．
3. 下の関数定義の下で g(5) が何を計算するか述べよ．

```
def g(n)
    if n >= 2
        g(n - 1) + g(n - 2)
    else
        1
    end
end
```

4. 上の小問の関数 g を，再帰呼び出しを使わないように書き直したものが下の関数 h である．空欄を埋めよ．

```
def h(n)
```

```
    result = make1d(n+1)
    i = [ ア ]
    while [ イ ]
        if i >= 2
            result[i] = [ ウ ]
        else
            result[i] = 1
        end
        i = [ エ ]
    end
    result[ [ オ ] ]
end
```

▶ 解答 p.94

問 2　${}_nC_r$ を求める 2 つの方法とその計算量 〔2010 年度 第 1 問〕

${}_nC_k$ を求める関数 combination_loop(n,k) に関して以下の問に答えよ.

```
def combination_loop(n,k)
    c = make2d(n+1,n+1)
    for i in 0..n
        c[i][0] = 1
        for j in 1..(i-1)
            c[i][j] = c[i-1][j-1]+c[i-1][j] #（ア）
        end
        [ イ ]
    end
    c[n][k]
end
```

1. プログラムを正しく動かすために，[イ]に何を入れればよいか.
2. combination_loop(3,2) と呼び出したとき，（ア）の行の文が実行されて，どのような代入がされるかを，代入される順に，添字 i，j の値と代入される値を示せ.
3. n に関して，この計算方法の計算量を示せ．求める過程も示せ.

${}_nC_k$ を求める関数 combination(n,k) について以下の問に答えよ.

```
def combination(n,k)
    if k > n
```

```
        0
    else
        if k == 0
            1
        else
            combination(n-1,k-1) + combination(n-1,k)
        end
    end
end
```

4. combination(3,2) と呼び出したとき，combination(n,k) がどのように再帰的に呼び出されるかを示せ．ただし，式 a+b は a を b より前に評価するものとする．
5. ${}_nC_n$ の値は，combination_loop(n,n) では，どのように求めているか．combination(n,n) では，どのように求めているか．

▶ 解答 p.95

問 3　差の絶対値の最小値 〔2012 年度 第 2 問〕

配列 a に，n 個の整数値が入っているとする（$n \geq 2$）．a の中の任意の 2 つの値の差の絶対値の最小値を求める問題について考える．例えば a = [9,3,5] の場合には 2 と答えるようなものである．この問題を解くアルゴリズムについて以下の小問に答えよ．

1. 計算量が $O(n^2)$ であるようなアルゴリズムを，文章あるいは Ruby の関数として書け．
2. 配列 a が大きさ順に整列されていたと仮定する．この仮定の下で計算量 $O(n)$ となるアルゴリズムを考え，文章あるいは Ruby の関数として書け．
3. 配列 a が整列されているとは限らないときのアルゴリズムのうち，1 よりも計算のオーダーが良いものを 1 つ考えよ．その計算量のオーダーと理由を書け．

▶ 解答 p.96

問 4　配列のソートアルゴリズム 〔2013 年度 第 1 問〕

1. 以下に示すのは，教科書で扱った併合整列法のプログラムである．空欄 ア ～ ウ を埋めよ．関数 merge を呼び出すと，昇順の配列を 2 つ併合したものが返る．例えば merge([1,3,5,7],[2,4,6,8]) と呼び出すと，[1,2,3,4,5,6,7,8] が返る．

```
def mergesort(a)
    n = a.length()
    from = make2d(n,1)
    for i in 0..(n-1)
        from[i][0] = a[i]
```

```
    end
    while n > 1 #（エ）
        to = make1d((n+1)/2)
        for i in 0..(n/2-1)
            to[i] = merge(from[i*2], from[i*2+1])
        end
        if !is_even(n)
            to[(n+1)/2-1] = [ ア ]
        end
        from = to
        n = [ イ ]
    end
    [ ウ ]
end
```

2. `mergesort([7,2,8,3,1,4,6,5])` の計算の途中で（エ）が実行される時の `from` の内容を，実行されるたびごとに書き出せ．
3. 教科書で扱った単純整列法と併合整列法を計算量のオーダーと使用するメモリ量の観点で比較せよ．

▶ 解答 p.97

問 5　線形探索と二分探索 〔計算機プログラミング I（植田）1999 年度 第 3 問〕

貴方が友達と数当てゲームをするとしよう．数当てゲームとは「1 から n までの整数のうち友達が最初に考えた整数を当てる」ゲームのことである（もちろん友達は，ゲームの最中に考えを変えないものとする）．貴方が 1 から n までの整数の任意の 1 つを当てずっぽうで言うと，その言った数字が友達の考えていた数字と同じである（正解）かそうでないか（不正解）かをその友達は答える．これを正解にたどり着くまで繰り返す．このとき以下の質問に答えなさい．

1. 貴方はランダムに数字を言っても混乱すると思い，1 から順に整数を言っていくことにした．このとき正解にたどり着くまでの平均回数（貴方が数字を言う回数の平均）を計算しなさい．
2. 上の方法だと時間がかかるので，友達は単に正解か不正解かを教えるだけでなく，貴方が言った数字が自分が考えている数字よりも，(1)大きいか (2)小さいか (3)同じ（つまり同じ）か，のいずれかを偽りなく教えることにした．友達が教えてくれるこの情報を効率的に使った場合（つまり，友達の情報を用いて正しく正解の候補を絞り込んでいった場合），正解にたどり着くまでの平均回数（貴方が数字を言う回数の平均）を計算しなさい．

▶ 解答 p.98

問 6　線形探索と二分探索 〔2006 年度 第 2 問〕

配列 a の中からある値 v がしまわれている場所（添字）を探すことを考える．ただし，a は整数がし

まわれている配列で，v は探す値（整数）であり，a の中には必ず v と同じ値が 1 つだけあるとする．

1. 先頭から順に配列の要素を調べ，最初に v と同じ値がしまわれている場所を見つけたときにその添字を返すようなアルゴリズムに基づいたメソッド linear(a,v) の定義を書け．
2. 配列 a が昇順（数が大きくなる順）に整列されているとする．このとき，次のようなアルゴリズムが考えられる．
   (a) 最初は調べる範囲を全体とする．
   (b) 調べる範囲の中央の値と v を比べ，（v と等しくなかった場合には）大小関係を用いて，調べる範囲を前半分あるいは後半分に絞り込むことを続ける．
   このアルゴリズムに基づいたメソッド binary(a,v) の定義を書け．
3. 1 と 2 で定義したアルゴリズムの計算量をそれぞれ求めよ．

※ 参考のために「正の数から成る配列 a の中で，先頭から最初に出現する 0 までの間（0 がなければ全範囲）の最大値を見つけて返す」Ruby のメソッド max(a) の定義例を以下に示す．

```
def max(a)
   v = 0
   i = 0
   while i < a.size && a[i] != 0
      if v <= a[i]
         v = a[i]
         i = i + 1
      else
         i = i + 1
      end
   end
   v
end
```

(2) 上の方法だと時間がかかるので、友達は単に正解か不正解かを教えるだけでなく、貴方が言った数字が自分が考えている数字よりも、(1) 大きいか (2) 小さいか (3) 同じ (つまり正解) か、のいずれかを偽りなく教えることにした。友達が教えてくれるこの情報を効率的に使った場合 (つまり、友達の情報を用いて正しく正解の候補を絞り込んでいった場合)、正解にたどり着くまでの平均回数（貴方が数字を言う回数の平均）を計算しなさい。(10 点)

▶ 解答 p.99

問 7　配列の操作と計算量 〔2008 年度 第 1 問〕

配列 x の部分的な添字範囲 p..q（$0 \le p \le q \le x.length() - 1$）に関して，配列の各要素を巡回的に 1 要素分左にシフトする Ruby の関数 rotate を，下のように書いたとする．

```
def rotate(x,p,q)
```

```
    t = x[p]
    for i in p+1..q
        x[i-1] = x[i]
    end
    x[q] = t
end
```

この関数の実行の様子は，下のようである．

```
irb(main):013:0> a = [2,3,5,7,11]
=> [2, 3, 5, 7, 11]
irb(main):014:0> rotate(a,0,4)
=> 2*
irb(main):015:0> a
=> [3,5,7,11,2]
```

* この 2 は rotate の中で最後に代入された値で，とくに意味はない．

1. この rotate のプログラムを参考として，x の p..q の範囲の要素を逆順に並べ替える関数 reverse(x,p,q) を書け．たとえば，x の p..q に [2,3,5,7,11] が入っていたら，同じ範囲の内容を [11,7,5,3,2] とするような関数である．ただし，rotate と同様に，関数に引数として与えられた配列の上で中身を変更するものとし，新しい配列を生成して使ってはいけない．

x.length() を n としたとき，x の $0..n-1$ の区間を，はじめの k 個の要素と後の $n-k$ 個の要素に分け，$0..k-1$ と $k..n-1$ の 2 つの区間に区切ったとする．この全部を x_a，後部を x_b としたとき，これをひっくり返して $x_b x_a$ とする関数を作ることを考える．たとえば，$n=7$，$k=3$ で，x=[2,3,5,7,11,17,19] とすると，x_a = [2,3,5]，x_b = [7,11,17,19] となるが，その x を [7,11,17,19,2,3,5] に変えるような関数である．

2. このような関数 transpose1(x,k) を rotate を用いて記述し，その計算量を評価せよ．ただし，引数の x と k の意味は上で説明したとおりとする．
3. 同じ機能を持つ関数 transpose2(x,k) を reverse を用いて記述し，その計算量を評価せよ．ただし，transpose2 のほうが transpose1 より効率よくできるはずなので，その工夫をすること．

▶ 解答 p.101

問 8　フィボナッチ数列と木構造 〔2007 年度 第 1 問〕

非負整数 n に対するフィボナッチ数の再帰的定義は

$$f(n)=\begin{cases} f(n-1)+f(n-2) & \text{if } n>1 \\ 1 & \text{otherwise} \end{cases}$$

と表され，これを素直に Ruby で書くと（ただし，f は fib と表記），下のようになる．

```
def fib(n)
   if n > 1
      fib(n-1) + fib(n-2)
   else
      1
   end
end
```

$n>1$ とすると，fib(n) は fib($n-1$) と fib($n-2$) という 2 つの再帰呼び出しをする．この関係を fib(n) を親とし，fib($n-1$) と fib($n-2$) を子とする木構造で表すと，fib(n を根とし，fib(1) または fib(0) を葉とする木構造ができる．

1. fib(3) を根とする木構造を，根を上にして図示せよ．
2. この木構造のすべての枝（エッジ）の数は，fib(n) を計算するために必要な再帰呼び出しの回数と等しくなる．これを n の関数 $F(n)(n \geq 0)$ とし，$F(n)$ の再帰的定義を書け．なお，$F(1)=F(0)=0$ である．再帰的定義は上の $f(n)$ の定義にならって数学的な式の形で書け．
3. fib(n) を根とする木構造の葉の個数を答えよ．
4. $F(n)$ と $f(n)(n \geq 0)$ の間には簡単な関係がある．$F(n)$ を $f(n)$ で表せ．厳密な証明は不要だが，その理由を 2〜3 行で書け．

第 2 章　数値計算と誤差

informatical reality

▶ 解答 p.103

問 9　記述問題・常微分方程式の解　〔2006 年度　第 3 問〕

1. 計算機で数値計算のプログラムを書くときには様々な誤差に気をつけなくてはならないが，以下の各々の誤差の定義とその誤差が現れる例を簡潔に説明せよ．
   (a) 丸め誤差
   (b) 情報落ち誤差
   (c) 打ち切り誤差
   (d) 桁落ち誤差
2. * 常微分方程式の数値解法には Euler 法と RungeKutta 法があるが，それぞれの特徴を誤差の観点から簡潔に述べよ．

▶ 解答 p.104

問 10　記述問題・2 次方程式の解　〔2010 年度　第 3 問〕

1. 丸め誤差とは何か，例を挙げて説明しなさい．
2. Ruby（irb）において，以下の下線部のような入力をしたところ，結果はどのようになるか述べなさい．また，この理由を簡単に述べなさい．

```
irb(main):001:0> 0.3==0.1*3
```

3. 桁落ち誤差とは何か，例を挙げて説明しなさい．
4. A 君と B 君が 2 次方程式 $ax^2+bx+c=0$ の解を求める以下のような関数 fa と fb をそれぞれ作成した．（これらの関数は 2 次方程式が実数解を持つときに使われると仮定する．）

```
include(Math)

def fa(a,b,c)
    d = b**2-4.0*a*c
    x1 = (-b+sqrt(d))/a/2
    x2 = (-b-sqrt(d))/a/2
    [x1, x2]
end

def fb(a,b,c)
    d = b**2-4.0*a*c
    x1 = (-b+sqrt(d))/a/2
```

```
    x2 = (-b+sqrt(d))/a/2
    if b > 0
        x1 = (1.0*c)/a/x2
    else
        x2 = (1.0*c)/a/x1
    end
    [x1, x2]
end

def check(a,b,c,x)
    [a*x[0]**2+b*x[0]+c, a*x[1]**2+b*x[1]+c]
end
```

2 次方程式 $x^2 - 100x + 1 = 0$ の解を両方のプログラムで計算し，結果を関数 check で検査したところ，結果は以下のようになった．

```
irb(main):002:0> check(1,-100,1,fa(1,-100,1))
[0.0, 1.22124532708767e-13]
irb(main):003:0> check(1,-100,1,fb(1,-100,1))
[0.0, 0.0]
```

B 君のプログラムでは A 君のプログラムにない部分が追加されているが，この部分は何を目的としているか述べなさい．

▶ 解答 p.104

問 11　Gauss-Jordan 法とピボット選択 〔2012 年度 第 3 問〕

1. 以下に示すのは Gauss-Jordan 法によって n 変数の連立 1 次方程式を解く関数 gj(a) である．［ア］と［イ］に適切な式を入れて，プログラムを完成させよ．

```
def gj(a) # Gauss-Jordan method without pivoting
    row = a.length()
    col = a[0].length()
    for k in 0..(col-2)
        akk = a[k][k]
        for i in 0..(col-1)
            a[k][i] = a[k][i] * [ア]
        end
        for i in 0..(row-1)
            if i != k
```

```
            aik = a[i][k]
            for j in k..(col-1)
                a[i][j] = a[i][j] - [ イ ]
            end
        end
    end
    a
end
```

2. (a) のプログラムによる計算の際に生じる可能性がある誤差について説明せよ．
3. Pivoting について説明し，(a) のプログラムをどのように修正すれば pivoting ができるかを示せ．配列 a で k 列目の絶対値が最大となる行を k 行目以降から探す関数 `maxrow(a,k)` と，配列 a の k 番目と max 番目を入れ替える `swap(a,k,max)` は定義済みとして使ってよい．
4. 不能（解が存在しない場合）や不定（複数の解が存在する場合）となる連立 1 次方程式に，(c) で修正したプログラムを適用した時に，不能や不定であることができるだけ早く判定でき，計算を打ち切れるようにプログラムを修正せよ．「計算を打ち切る」には関数 `abort()` を呼ぶだけでよい．関数 `abort()` を呼ぶとプログラムの実行はそこで止まる．

▶ 解答 p.106

問 12　台形公式とその誤差 〔2009 年度 第 2 問〕

関数 $f(x)$ の数値積分 $\int_{x_s}^{x_e} f(x)dx$ を求めたい．これに関して以下の問に答えよ．

1. 台形公式について，計算式を示しながら，考え方を説明せよ．

以下，台形公式は授業で説明したプログラムの計算順序で計算するものとする．

2. 台形公式を用いて $f(x)=1$ を $x_s=0$ から $x_e=1$ まで積分する場合，どのような誤差が生じる可能性があるかを，その誤差が発生する理由を含めて，述べよ．
3. 台形公式を用いて $f(x)=x^2$ を $x_s=0$ から $x_e=1$ まで積分する場合，問 2 に加えてどのような誤差が生じる可能性があるかを，その誤差が発生する理由を含めて，述べよ．
4. 台形公式以外の公式を用いて，問 3 で生じる誤差を減らすことを考える．そのような公式の名前を 1 つあげ，考え方を説明せよ．

▶ 解答 p.107

問 13　Gauss-Jordan 法とピボット選択 〔2007 年度 第 2 問〕

下に連立一次方程式を Gauss-Jordan 法で解く Ruby プログラムを示す．係数行列を表す 2 次元配列 m と定数ベクトルを表す 1 次元配列 v を引数として，解のベクトルを 1 次元配列で返す関数 `linear(m,v)` が定義されている．

```
def linear(m,v)
    # 係数行列 m と列ベクトル v が与えられる
    # m と v を組み合わせた行列 a をつくる
    row=m.size
    col=m.size+1
    a = Array.new(row)
    for i in 0..(row-1)
        a[i] = Array.new(col)
        for j in 0..(col-2)
            a[i][j] = m[i][j]
        end
            a[i][col-1] = v[i]
        end

        # 一行ずつ非対角成分を消去する
        for k in 0..(row-1)
            akk = a[k][k]
            for i in 0..(col-1)
                a[k][i] = a[k][i]/akk # k 行目を対角成分で割り正規化する
            end
            for i in 0..(row-1)
                if (i != k)
                    aik = a[i][k] # k 行目を用いて i 行目の非対角成分を消去する
                    for j in k..(col-1)
                        a[i][j] = [ ア ]
                    end
                end
            end
        end
        s = Array.new(row)
        for i in 0..(row-1) # 解を列ベクトル s として取り出す
            s[i] = [ イ ]
        end
    s # 解の列ベクトル s を戻り値として終了
end
```

以下は実行例である.

```
> m = [[1.0,1.0,-1.0],[3.0,5.0,-7.0],[2.0,-3.0,1.0]]
```

```
> v = [2.0,0.0,5.0]
> linear(m,v)
=> [3.0, 1.0, 2.0]
```

1. 空欄アとイにあてはまる式を答えよ．
2. 1 のプログラムには数値計算上の問題がある．その問題を指摘（複数ある場合は列挙）し，解決策を簡単に述べよ．

第 3 章　乱数

informatical reality

▶ 解答 p.109

問 14　乱数の理解・モンテカルロ法およびその応用　〔2008 年度　第 2 問〕

1. 一様乱数とは何か説明せよ．
2. 擬似乱数とは何か説明せよ．
3. モンテカルロ法とは，どのような問題を解くために用いられるかを説明せよ．
4. xy 平面上の単位円 $P = \{(x,y) | 0 \leq x^2 + y^2 \leq 1\}$ の中に均一に n 個の点を配置することを考える．以下に示す `cover(n)` は，上記のような n 個のデータを返すことを意図して書かれたプログラムである．`cover(n)` は，n 個の点のデータを，各点の xy 座標の配列とし，さらにそれを n 個含む配列を返す．このプログラムにより出力されるデータは意図した通りに単位円 P 内に均一に分布するか，理由と共に述べよ．もし，意図した通りにならない場合は，どこが間違っているかを指摘せよ．プログラムの文法の誤りを問うているわけではないことに注意せよ．

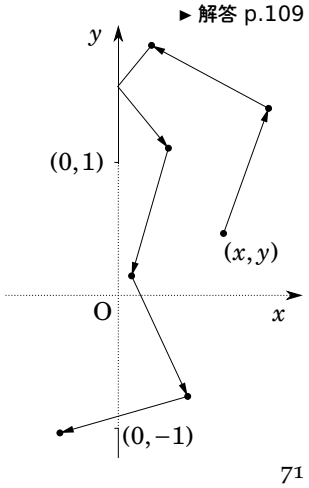
```
include(Math)

def cover(n)
    a = Array.new(n)
    for i in 0..n-1
        r = rand()
        theta = 2*PI*rand()
        a[i] = [r*cos(theta), r*sin(theta)]
    end
    a
end
```

▶ 解答 p.109

問 15　ランダムウォーク　〔2011 年度　第 3 問〕

右図のように，平面上の x 座標が皮膚の領域の上を，一回のステップでランダムな方向に直線距離で 1 だけ移動する点がある．y 軸にぶつかる場合は，y 軸によって完全反射する．この場合，ぶつかるまでに移動した距離とぶつかってから移動した距離の和が 1 であるとする．ただし，$(0,-1)$ から $(0,1)$ までは穴が空いていると仮定する．この点の初期値を (x,y) として，穴を抜けるまでに移動するステップ数を求める手続き `escapesteps(x,y)` を下に掲げる．以下の問に答えよ．



```
include(Math)

def escapesteps(x,y)
    n = 0
    escaped = false
    while !escaped
        r = rand()
        dx = cos(2*PI*r)
        dy = sin(2*PI*r)
        x1 = x + dx
        y1 = y + dy
        if x1 < 0
            if  ア 
                escaped = true
            else
                 イ 
                y = y1
            end
        else
             ウ 
            y = y1
        end
        n = n + 1
    end
    n
end
```

1. `escapesteps(x,y)` 関数で利用している `rand()` 関数は擬似乱数を実現している．擬似乱数とは何かを説明せよ．
2. 擬似乱数が満たすべき望ましい条件を一つ述べよ．
3. ア は，点が穴を抜ける条件を表している．ここに入るべき Ruby の式を書け．
4. イ と ウ は，x 座標の更新を行う．これらに入るべき Ruby の命令を書け．

第 4 章　動的計画法

informatical reality

▶ 解答 p.111

問 16　アラインメント　〔2007 年度　第 3 問〕

次のようにアラインメントに関するスコアを定める．

一致	不一致	ギャップ
+2	−1	−2

1. 文字列 AGGTAA と文字列 AGTGGGA の最適（類似度が最大）のアラインメントを求めるために，以下のテーブルのア～ケにあてはまる数を答えよ．

		A	G	T	G	G	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	2	0	-2	-4	-6	-8	-10
G	-4	0	4	2	0	-2	-4	-6
G	-6	-2	2	3	4	2	0	-2
T	-8	-4	0	4	2	ア	イ	ウ
A	-10	-6	-2	2	3	エ	オ	カ
A	-12	-8	-4	0	1	キ	ク	ケ

2. トレース・バックによってギャップを挿入した結果を以下のように示せ．

```
ATAG
A-AC
```

▶ 解答 p.111

問 17　経路の数　〔2010 年度　第 4 問〕

横 $M+1$，縦 $N+1$ 個のマス目を持つ盤がある．駒が，座標 $(0,0)$ で表された左下のマス目から，座標 (M,N) まで移動するとする．一度に駒が動ける範囲は 3 通りに制限されており，右，上，斜め右上，だけとする．このとき以下の問いに答えなさい．

1. 座標 $(0,0)$ から座標 (m,n) まで駒が取り得る経路の数を $T_{m,n}$ とする（ただし $1 \leq m \leq M, 1 \leq n \leq N$ とする）．このとき

$$T_{m,n} = T_{m,n-1} + T_{m-1,n} + T_{m-1,n-1}$$

となることを示しなさい．このとき初項

$$T_{0,0}, T_{0,i}, T_{j,0} \quad (1 \leq i \leq M, 1 \leq j \leq N)$$

はどのように決めたらよいか示しなさい．

2. $T_{M,N}$ を求めるために動的計画法を用いたアルゴリズムの指針を示しなさい．

▶ 解答 p.112

問 18　組合せ最適化問題 〔2008 年度 第 3 問〕

配列 a に正の整数がいくつか格納されているとする．配列 a と非負の整数 n が与えられたとき，配列 a の中の各整数を 0 回以上使った和が n に等しくできるならば，整数を使う回数の総和の最小値を返す関数を定義したい．配列 a の整数をどのように組み合わせてもその和が n に等しくできない場合は，−1 を返す．たとえば，a=[1000,2000,5000,10000]，n=9000 とすると，9000 円を作ることのできる紙幣の最小の枚数を返すことになる．実際に，$2000\times2+5000\times1=9000$ が最小の枚数を与えるので，3 という答えが返る．

下は，この計算を行う再帰的な関数 rec である．以下の 1〜4 の問に答えよ．

```
def rec(a,n)
   if n==0
      0
   else
      i = 0
      k = -1
      for i in 0..a.length()-1
         if n >= a[i]
            r = rec(a,n-a[i])
            if r >= 0 && (k < 0 || r + 1 < k)
               k = r + 1
            end
         end
      end
      k
   end
end
```

1. rec([2,3],7) を実行したとき，最初の呼び出しも含めて関数 rec は何回呼び出されるか．
2. この再帰的関数は，同じ計算を重複して行うために効率が悪い．どうして重複が起こるのか，簡単に説明せよ．
3. 次の関数 ite(a,n) は，繰り返しを用いて計算の重複を避けている． ア ～ ウ の部分を埋めてプログラムを完成させよ．

```
def ite(a,n)
   b = Array.new(n+1)
   b[0] = ア
   for m in 1..n
```

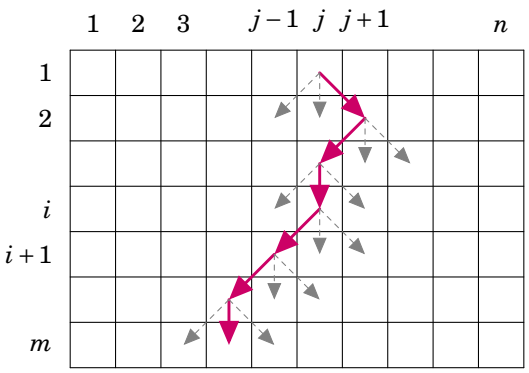
```
        i = 0
        k = -1
        for i in 0..a.length()-1
            if m >= a[i] && b[m-a[i]] >= 0 && (k < 0 || b[m-a[i]] + 1 < k)
                k = b[m-a[i]] + 1
            end
        end
        b[m] = [ イ ]
    end
    b[ ウ ]
end
```

4. どうして計算の重複を避けられるのか，簡単に説明せよ．

▶ 解答 p.113

問 19　経路と累積利益 〔2011 年度 第 4 問〕

縦 m 行，横 n 列からなる格子領域において，i 行 j 列の位置を (i,j) と表記する．右図のように，i 行目の (i,j) から移動する際には，$i+1$ 行目の $(i+1,j-1)$，$(i+1,j)$，$(i+1,j+1)$ のいずれかのみに移動して領域全体を縦断する．位置 (i,j) に存在することで正の利益 $\mathrm{gain}(i,j)$ を得られるものとすると，ある経路で縦断する際の**累積利益**は $\sum_{i=1}^{m}\mathrm{gain}(i,j_i)$ となる（j_i は当該の経路で i 行目における位置 (i,j_i) を表す）．出発点の 1 行目と終着点の m 行目の列の位置は任意として，縦断によって得られる累積利益の最大値を求めたい．以下の問いに答えよ．



1. 1 行目の任意の位置から (i,j) までの経路は複数存在するが，それらの累積利益の最大値（以下，**最大累積利益**と呼ぶ）を $\mathrm{maxGain}(i,j)$ とする．1 行目における最大累積利益 $\mathrm{maxGain}(i,j)$ は，その位置 $(1,j)$ における利益のみであるから，

$$\mathrm{maxGain}(1,j)=\mathrm{gain}(1,j)$$

となる．$i>1$ の場合の i 行目の最大累積利益 $\mathrm{maxGain}(i,j)$ を，$i-1$ 行目の最大累積利益，$\mathrm{gain}(i,j)$ ならびに $\max(a,b)$ を用いて漸化式で表現せよ．ただし，位置 (i,j) が範囲外の場合，すなわち $i<1, i>m, j<1, j>n$ のいずれかが真となる場合には，$\mathrm{maxGain}(i,j)$ は 0.0 を返すと仮定して良い．

2. 上の議論をもとに最大累積利益 $\mathrm{maxGain}(i,j)$ を求める Ruby の関数 `maxGain(i,j,m,n)` を再帰的に定義すると，以下のようになる．（引数として m,n を補っていることに注意せよ．）このよう

な再帰関数は i の値が大きくなると急速に計算時間を必要とする．領域全体を縦断する際の最大累積利益 maxGain(m,m,m,n) を計算する際に，関数 maxGain は何回呼び出されるか? ただし，$n > 2m$ であるものとする．

```
def maxGain(i, j, m, n)
    if i < 1 || i > m || j < 1 || j > n
        0.0
    else
        if i == 1
            gain(1, j)
        else
            [ 1 の解答に相当する計算 ]
        end
    end
end
```

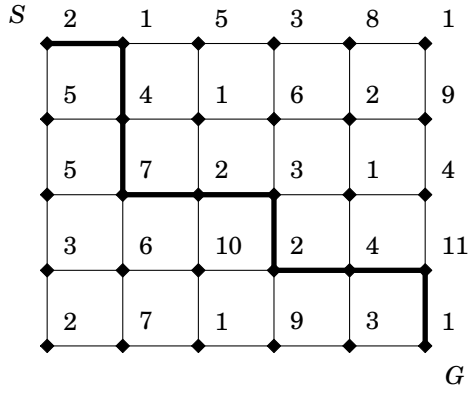
3. この計算を効率よく行うために，位置 (i,j) までの最大累積利益を与える $m+1$ 行 $n+2$ 列の 2 次元配列 totalGain[i][j] を動的計画法によって求める関数 calculateTotal(m, n) は下のようになる．[ア]～[ウ]を埋めよ．

```
def calculateTotal(m, n)
    totalGain = make2d(m+1, n+2)
    for i in 1..m
        totalGain[i][0]   = [ ア ]
        totalGain[i][n+1] = [ イ ]
        for j in 1..n
            if i == 1
                totalGain[i][j] = [ イ ]
            else
                totalGain[i][j] = [ ウ ]
            end
        end
    end
    totalGain
end
```

▶ 解答 p.114

問 20　最短路問題 〔2009 年度 第 3 問〕

下図のように m 行 n 列の格子状に並んだ都市（◆）が道路で結ばれている．西北の町 S から出発して，東南の町 G へ最小費用で行きたい．S，G を含め，町では必ず一泊し，1G 以上の宿泊料を払う（都市の右肩の数値）．以下の各問に答えなさい．



1. 町から南の道へ行くのを 0，東の道へ行くのを 1 と表せば，最小宿泊経路はたどった順に 0 と 1 からなる 1 次元の配列で表現できる．図の中で太字に描いた経路を，Ruby の配列の表記法（`[1,0,…]` のように要素を順に書く）に従って書きなさい．また，この経路での費用（宿泊料の合計）を書きなさい．
2. S から G に行く最小宿泊数の経路が何通りあるかを m と n の式で書きなさい．
3. 以下では町の座標を (行番号,列番号) のように書く．最小費用を求める問題は，町 S,G の座標をそれぞれ $(0,0),(m-1,n-1)$ としたとき，町 (i,j) の宿泊料を `c(i,j)` で表す関数でモデル化できる．経路が問 1 のように表現されているとき，費用を求めるプログラムを Ruby で下のように書いた．空欄［ア］，［イ］，［ウ］に適当なプログラム断片を入れなさい．`path` は上記のように表現された経路を，`m()`，`n()` は m，n をそれぞれ意味する．

```
def get_cost(path)
    cost = c(0,0)
    i = 0
    j = 0
    for k in 0..(m()+n()) [ア] )
        if path[k] == 0
            i = i+1
        else
            [イ]
        end
        cost = [ウ]
```

```
    end
    cost
end
```

4. S から G へ行く最小費用を求めるために，町 (i,j) から G へ行く最小費用を求める関数 get_min_cost を下のように定義した．get_min_cost(0,0) を計算するのに get_min_cost(4,5) が何回呼ばれるかを答えなさい（$4<m, 5<n$ とする）．なお，1000000 はここでは無限大を意味する．

```
def get_min_cost(i,j)
    if i == m()-1 && j == n()-1
        c(i,j)
    else
        south = 1000000
        east = 1000000
        if i < m()-1
            south = get_min_cost(i+1,j)
        end
        if j < n()-1
            east = get_min_cost(i,j+1)
        end
        c(i,j) + min(south,east)
    end
end
```

5. 一度計算した get_min_cost(i,j) の値を，関数の外側で定義した m 行 n 列の配列 $min_cost[i][j] に入れておくと無駄な計算が省ける．このアイデアに基づいて問 4 のプログラムを手直ししたプログラムと，その外側で行う$min_cost の定義や初期化の文を書きなさい．関数名は同じとし，上のプログラムのうちオレンジ色で示された範囲の行を必ずそのまま再利用すること．（注意：Ruby では，ある関数の中で，その関数の外や他の関数で定義された変数を使うためには，変数名を$min_cost のように$ではじめなければならない．）

第 5 章　その他

informatical reality

ここでは分類しかねる問題を掲載する．基本的にアルゴリズムを理解して実際に形にする問題である．オブジェクト指向など，分類できるが範囲外のものに関しては次以降の章で紹介する．

▶ 解答 p.117

問 21　計算の特殊定義 〔2011 年度 第 2 問〕

下の Ruby プログラムを読んで以下の問いに答えよ．

```
def f(x,y)
    z = 1
    while y != 0
        while y % 2 == 0
            x = x * x
            y = y / 2
        end
        y = y - 1
        z = z * x
    end
    z
end
```

1. f(2,4) および f(3,5) を実行すると，それぞれどのような値が得られるか．結果のみ答えよ．
2. c = f(a,b) としたとき，c と a，b の関係を，簡潔に述べよ．ただし，a と b は非負の整数とする．

▶ 解答 p.117

問 22　整数の数え上げ 〔2010 年度 第 2 問〕

大きさ n の配列 a に m 種類の正の整数が格納されているとき，a に現れる整数を配列 b に求め，各整数の a に現れる回数を配列 c に求めたい．たとえば，配列 a が [3,1,4,1,5,9,2,6,5,3] のであるとき，n は 10，m は 7 である．このとき，b には [3,1,4,5,9,2,6]，c には [2,2,1,2,1,1,1] を返すことができる．

1. 次のプログラムは，a から b と c を求めるプログラムの一例である．このプログラムの計算量を n と m を用いて表せ．なお，引数 b と c には大きさ m の配列が渡される．配列 b の各要素は 0 で初期化されていると仮定する．

```
def intcount(a,b,c)
```

```
    for i in 0..(a.length()-1)
        x = a[i]
        j = 0
        while b[j] != 0 && b[j] != x
            j = j + 1
        end
        if b[j] == 0
            b[j] = x
            c[j] = 1
        else
            c[j] = c[j] + 1
        end
    end
end
```

2. a が整列されていると仮定する．この場合に計算量が $O(n)$ で済むように，上のプログラムを参考に新たに intcount(a,b,c) の定義を書け．

▶ 解答 p.118

問 23　整数列の和 〔2011 年度 第 1 問〕

N 個の整数の列 a = [$x_0, x_1, \cdots, x_{N-1}$] の i 番目から $j-1$ 番目（$0 \le i \le j \le N$）までの数の和を $s(\mathrm{a}, i, j)$ とする（ただし $i = j$ のとき $s(\mathrm{a}, i, j) = 0$）．以下の問いに答えよ．

1. a = [8,-4,-5,2,4,-5,5,3,-7,8] のとき，$s(\mathrm{a},0,0) = 0$，$s(\mathrm{a},0,1) = 8$ である．$s(\mathrm{a},0,2)$，$s(\mathrm{a},0,3)$，$s(\mathrm{a},0,4)$ を求めよ．
2. $s(\mathrm{a},0,N)$ を単純な反復計算によって求めるのに必要な計算量のオーダーを N の式で書け．
3. $x \le i \le j \le y$ を満たす全ての i, j について $s(\mathrm{a}, i, j)$ の最大値を $\mathrm{mss}(\mathrm{a}, x, y)$ とする（ただし $0 \le x \le y \le N$ とする）．下のように全ての i, j の組み合わせについて $s(\mathrm{a}, i, j)$ を計算し，その最大値を求める場合の $\mathrm{mss}(\mathrm{a}, 0, N)$ の計算量のオーダーを N の式で書け．

```
def mss(a,x,y)
    m = 0
    for i in x..y
        for j in i..y
            m = max(m,s(a,i,j))
        end
    end
    m
end
```

4. $\mathrm{mss}(\mathtt{a},0,z-1)=s(\mathtt{a},x,z-1)$ かつ $s(\mathtt{a},x,z)<0$ のとき，a の $z-1$ 番目は $\mathrm{mss}(\mathtt{a},0,y)$ を与えるような区間に含まれないことが知られている（つまり $\mathrm{mss}(\mathtt{a},0,y)=s(\mathtt{a},i,j)$ ならば，$i\le z\le j$ ではない）．このことを利用して $\mathrm{mss}(\mathtt{a},0,m)$ を求める関数 mss0(a,m) は下のように定義できる．

```
def mss0(a,m)
    t = 0
    sum = 0
    for i in 0..(m-1)
        sum = sum + a[i]
        if sum > t
            t = sum
        else
            if sum < 0
                sum = 0
            end
        end
        # ※
    end
    t
end
```

a=[8,-4,-5,2,4,-5,5,3,-7,8] の下で mss0(a,10) を実行したとき，繰り返し各回の最後（※）における sum，t の値を下表のような形で書け．

i	0	1	2	3	4	5	6	7	8	9
sum										
t										

5. 小問 4 で示した mss0(a,N) の計算量のオーダーを N の式で書け．

▶ 解答 p.119

問 24　再帰関数による探索 〔2012 年度 第 4 問〕

文字列の配列が 2 つ与えられているとする．これらを a と b とする．a と b の大きさ（文字列の数）は同じであるとする．たとえば，a=["0", "11"]，b=["101", "1"] とする．

a から重複を許して文字列をいくつか選んで連結した結果と，b から対応する添字の文字列を選んで同じ順序で連結した結果が，まったく同じ文字列になるようにしたい．たとえば，上の例の場合，a の 1,0,1 番目つまり a[1] と a[0] と a[1] を連結すると"11011"が得られる．これに対して，b の 1,0,1 番目つまり b[1] と b[0] と b[1] を連結すると，まったく同じ文字列である"11011"が得られる．

1. a=["1", "0", "00"]，b=["0", "011", "0"] としたとき，連結してまったく同じ文字列が得られるような文字列の選び方を添字の並びとして答えよ．

2. 次のプログラムは，上の問題を解こうとするものである．関数 post の最初の 2 つの引数として，2 つの配列を指定する．3 番目の引数としては配列の大きさを指定する．

```
def post(a, b, m, n, as, bs)
    if as != "" && as == bs
        as
    else
        if n == 0
            ""
        else
            k = 0
            p = ""
            while p == "" && k < m
                p = post(a, b, m, n-1, as+a[k], bs+b[k])
                k = k + 1
            end
            p
        end
    end
end
```

なお，演算子 + は，文字列に対しては，連結の操作を意味する．たとえば，"010" + "00"は"01000"になる．""は空文字列を表し，"" + "00"は"00"になる．また，比較演算子==（!=）は，文字列に対しては，同じ文字列であるか（ないか）を判定する．

post(["0", "11"], ["101", "1"], 2, 1, "110", "1101") を実行すると何が返るか．

3. 小問 1 の例に対して，上の問題を解くためには，関数 post をどのように呼び出せばよいか．また，結果として何が返るか．
4. どのように文字列を選んでもまったく同じ文字列を作れない場合に対して，関数 post の m と n に関する計算量のオーダーを答えよ．ただし，文字列の演算は文字列の長さによらずに一定時間しかかからないものと仮定する．

▶ 解答 p.120

問 25　バケツの問題 〔2009 年度 第 1 問〕

a リットル入るバケツ A と b リットル入るバケツ B がある．このバケツで水を正確に q リットルはかり取りたい．ただし，a,b,q は正の整数とする．可能な操作は以下であるとき 1 から 3 の問に答えよ．

fill(X)	バケツ X に一杯の水を汲む．ただし，X は A または B.
empty(X)	バケツ X を空にする．ただし，X は A または B.
move(X, Y)	バケツ X の水を可能な限り Y に移す．可能な限りという意味は，X が空になるか Y が一杯になるまでという意味である．X, Y はそれぞれ A, B または B, A.

1. 目標は，A および B が空の状態から始めて，A または B に q リットルの水が入っている状態を作ることである．$a = 5, b = 3, q = 4$ として，その目標状態に至る操作の列の 1 つを例を示しなさい．たとえば以下のように記述すればよい．

   fill(B), move(B, A), empty(A), ⋯

2. 一般の a, b, q に対して，目標状態に至る操作の列を発見するために，この問題を次のようにモデル化する．まず，ある地点における問題の状態を，A と B に入っている現在の水の量を (x, y) として，xy 平面上の点によって表現する．以下ではこれを問題平面と呼ぶ．
   （a）問題平面上で，fill，empty，move という操作が一般にどのような動きとなるかを述べなさい．
   （b）問題平面上で，目標となる状態はどのような点に対応するかを述べなさい．
   （c）解が存在すると仮定して，目標状態に至る操作列を問題平面上で探索する方法について，簡単に述べなさい．

3. $a = 8, b = 5, q = 4$ の場合について，目標状態に至る最短の操作列を 1 つ示しなさい．

第 6 章　オブジェクト指向*

informatical reality

▶ 解答 p.125

問 26　メソッド定義 〔2006 年度 第 1 問〕

時間の長さを，時間，分，秒で表す Ruby のクラス Time が次のように定義されているとする．

```
class Time
    def initialize(h,m,s)
        @hour = h
        @minute = m
        @second = s
    end
    def to_string
        @hour.to_s + "時間" + @minute.to_s + "分" + @second.to_s + "秒"
    end
end
```

以下で，初期値が与えられた Time 型の値は，条件

$$0 \leq @hour, \quad 0 \leq @minute < 60, \quad 0 \leq @second < 60$$

が成立しているものと仮定してよい．しかし，メソッドを実行して状態が変化した Time 型の値は，上の条件を満たすようにする必要がある．

1. 秒単位のデータを引数として与えられ，その秒数だけ現在の時間に加えたクラス Time のメソッド，advance を定義しなさい．与えられるデータは 0 以上の整数と仮定してよい．メソッド定義だけ書けばよく，クラス定義を写す必要はない．
2. 秒単位のデータを引数として与えられ，その秒数だけ現在の時間から引いたクラス Time のメソッド，back を定義しなさい．与えられるデータは 0 以上の整数と仮定してよい．また，引いた結果の時間がマイナスになることはないと仮定してよい．メソッド定義だけ書けばよく，クラス定義を写す必要はない．

たとえば，次のように実行すると，

```
t = Time.new(5,3,43)
puts t.to_string
t.advance(10000)
puts t.to_string
t.back(20000)
puts t.to_string
```

次のような結果が得られる．

```
5 時間 3 分 43 秒
7 時間 50 分 23 秒
2 時間 17 分 3 秒
```

▶ 解答 p.126

問 27　クラスの構造 〔2008 年度 第 4 問〕

N 人の学生の試験の成績を，得点順に並べて氏名，学生証番号，得点を印刷したい．与えられるデータは学生証番号順に記録されているので，得点順になるように整列しなければいけない．このとき，以下の問に答えよ．

1. 学生の氏名は文字列の配列で，学生証番号と得点はそれぞれ整数の配列で与えられる場合，どのようにして整列を行うべきかを述べよ．ただし，得点の配列を整列するプログラムがすでにあるときに，氏名と学生証番号の配列をどのように操作すべきかだけを述べよ．
2. 次のこの整列を行う別の方法として，学生に対応するクラスを定義して，そのクラスのオブジェクトで学生 1 人分のデータを表すことを考える．学生を表すオブジェクトはどのような値を表すインスタンス変数を持つかを書け．
3. 学生を表すオブジェクトを定義するクラスはどのようなメソッドを持つべきか，各メソッドの働きを書け．
4. 2 の方法で，学生を表すオブジェクトが配列として与えられた場合，どのように整列を行うべきかを述べよ．
5. 2 の方法は，1 の場合と比べてどのような点が優れているかを説明せよ．

▶ 解答 p.126

問 28　オブジェクト指向の利点 〔2007 年度 第 4 問〕

カプセル化（情報隠蔽でもよい），継承（インヘリタンス），多態性（ポリモルフィズム）のそれぞれについて，

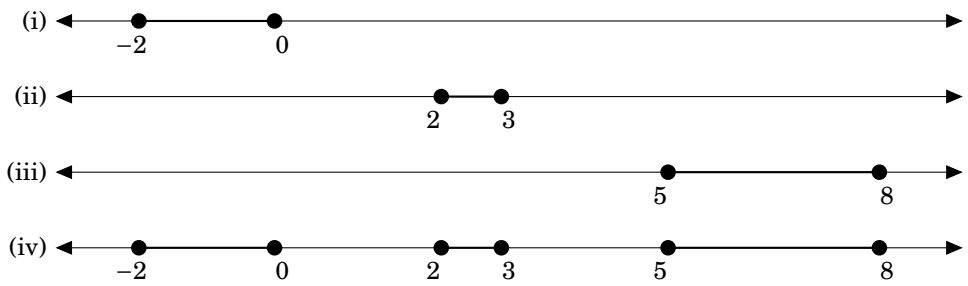
1. どのような概念であるか，
2. どのような問題を解決するために導入されたか，

を，それぞれ 3 行以内で簡潔に説明せよ．

▶ 解答 p.127

問 29　オブジェクト指向の利用 〔2009 年度 第 4 問〕

下図 (i), (ii), (iii) はそれぞれ −2 以上 0 以下，2 以上 3 以下，5 以上 8 以下の数から成る集合，(iv) は，これら 3 つの集合の合併を表している．このような数直線上の数の集合をオブジェクトとして表現したい．なお，この問題では数値はすべて整数で与えられるものと仮定せよ．



1. 下のように Interval クラスを定義する.

```
class Interval
   attr_accessor("from", "to")

   def initialize(f,t)
      self.from = f
      self.to = t
   end

   def is_member(x)
      ア
   end
end
```

このクラスのオブジェクトは下のように，2 つの数を指定して生成する．s1，s2，s3 のオブジェクトは，それぞれ上図の (i)，(ii)，(iii) の集合を表す.

```
s1 = Interval.new(-2,0)
s2 = Interval.new(2,3)
s3 = Interval.new(5,8)
```

is_member(x) というメソッドが，各々のオブジェクトが表す集合に x が属しているときに true，属していないときに false を返すように，上の ア の部分を埋めよ．たとえば，s1.is_member(-1) は true，s2.is_member(-1) は false となる.

2. 下のように Union クラスを定義する.

```
class Union
   attr_accessor("set1", "set2")

   def initialize(s1, s2)
      self.set1 = s1
      self.set2 = s2
```

```
    end

    def is_member(x)
        イ
    end
end
```

このクラスのオブジェクトは以下のように，集合を表す 2 つのオブジェクトを指定して生成する．

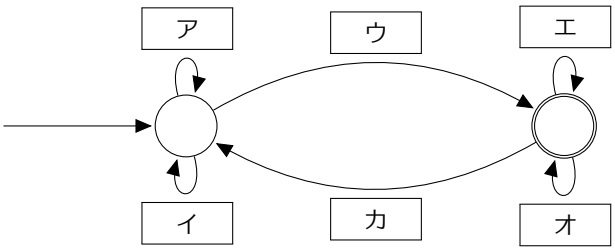
```
s4 = Union.new(Union.new(s1, s2), s3)
```

s4 のオブジェクトは，s1，s2，s3 の表す集合の合併を表す．s1，s2，s3 が問 1 のオブジェクトを表すとすると，s4 のオブジェクトは，上図 (iv) の集合を表す．たとえば，s4.is_member(-1) は true となる．問 1 と同様にして，上の イ を埋めよ．

▶ 解答 p.129

問 30　3 進法のオートマトン　〔2006 年度　第 4 問〕

3 進法をアルファベット $\Sigma = \{0,1,2\}$ 上の文字列によって表現するとき，奇数を表す文字列を受理するオートマトンを以下の図のように定義する．［ア］から［カ］までの空欄に入る文字をひとつずつ書け．



第III部
分野別問題集 解答編

この部では，第 II 部で掲載した問題集の解答・解説を掲載する．もちろん解答の方法はひと通りでないから，必ずしもこの通りでなくともよい．これが解答の指標・参考になれば幸いである．大切なのは，試験の感覚をつかむことである．なお，試験範囲外の問題に関しては，解説を省いて解答のみの掲載とさせてもらう．また，一部解答を作成する上で過去のシケプリを利用したが，そちらに関しては別途参考文献リストで紹介する．

第 1 章　計算量・再帰的定義

informatical reality

▶ 問題 p.57

問 1　再帰的定義とそれと同値の繰り返しによる定義　〔2012 年度　第 1 問〕

解法のポイント

基本的な問題．与えられた定義から漸化式を思い浮かべ，何を求める関数なのかを把握する．

解答

1. 5 の階乗（120）
2.
```
def f(n)
    ans = 1
    for i in 1..n
        ans = ans * i
    end
    ans
end
```
3. 5 番目のフィボナッチ数（8）
4. ア　0
   イ　i <= n
   ウ　result[i-1] + result[i-2]
   エ　i + 1
   オ　n

解説

1. 関数の定義から $f(n) = n * f(n-1)$ とわかるのでこれは階乗を計算する関数．3 も同様だが，「何を計算するか」と問題にあるので，返される数字だけ答えるのは適切とは言えないだろう．
2. 1 から n まで順に掛け算していくことを考える．
3. 関数の定義から $g(n) = g(n-1) + g(n-2)$ とわかるのでこれはフィボナッチ数を計算する関数．「$g(0) = g(1) = 1$ のフィボナッチ数」といったほうが親切かもしれないが省略した．
4. 順に小さいほうから計算していくような定義．すべてにおいて `result[i-1] + result[i-2]` を計算したいが，`i` が 0 と 1 のときではこれが使えないので条件分岐している．

▶ 問題 p.58

問 2 ${}_nC_r$ を求める 2 つの方法とその計算量 〔2010 年度 第 1 問〕

解法のポイント

こちらも基本的な問題．コンビネーションがパスカルの三角形や漸化式から求められることを思い出す．

解答

1. `c[i][i] = 1`
2. - `i` = 2, `j` = 1 のとき 2 を代入
   - `i` = 3, `j` = 1 のとき 3 を代入
   - `i` = 3, `j` = 2 のとき 3 を代入
3. 1 重目の for 文内では計算量 O(`i`) の計算が行われ，`i` が 1 から `n` まで順に動いているので $1+2+\cdots+n=\frac{1}{2}n(n+1)$ より計算量は $O(n^2)$.
4. - `combination(3,2)` が呼び出される.
     - `combination(2,1)` が呼び出される.
       - `combination(1,0)` が呼び出される.
       - `combination(1,1)` が呼び出される.
         - `combination(0,0)` が呼び出される.
         - `combination(0,1)` が呼び出される.
     - `combination(2,2)` が呼び出される.
       - `combination(1,1)` が呼び出される.
         - `combination(0,0)` が呼び出される.
         - `combination(0,1)` が呼び出される.
       - `combination(1,2)` が呼び出される.
5. `combination_loop(n,n)` ではパスカルの三角形を 2 次元配列上に書くことによって ${}_0C_0$ から ${}_nC_n$ まで順番に，それまでの結果を用いてすべて求めることで導いている．一方 `combination(n,n)` では単純な ${}_nC_k = {}_{n-1}C_{k-1} + {}_{n-1}C_k (0 < k \leq n)$ という漸化式を利用して再帰的に解いており，重複計算による効率の悪さがあるものの必要なコンビネーションの値のみから計算している．

解説

1. パスカルの三角形を使った解法．n 行目では必ず 0 番目から n 番目までの要素が 0 以外の値でなければならない（1）．for 文では $n-1$ 番目の要素までしか計算していないので，n 番目の要素はここで指定する．
2. そのままガリガリ実行するだけ．内側の for 文は `i` が 2 以上のときにしか使われないことに注意．

	0	1	2	3	4	⋯
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
⋮	⋮					⋱

表 1.1　ここで処理するパスカルの三角形の概念図．プログラム中の 2 次元配列 c[][] にあたる．

3. 2 重になっている for 文に注目．
4. こちらもそのままガリガリ実行するだけ．解答のようにインデントをつけるとわかりやすい．
5. 問題が何を聞いているのかよくわからないが，アルゴリズムの説明をしておけば十分だろう．

▶ 問題 p.59

問 3　差の絶対値の最小値 〔2012 年度 第 2 問〕

解法のポイント

計算量を与えられた状態でアルゴリズムを考える問題．1 と 2 は単純だろう．3 はソートを使えばよいことに気付きたい．

解答

1. i 番目の値と，$i+1$ 番目から $n-1$ 番目までの値の差の絶対値をそれぞれ調べることを，i を 0 から $n-1$ まで動かして行い，その最小値をとる．

```
def minabs(a)
    n = a.length()
    min = abs(a[1]-a[0])
    for i in 0..n-1
        for j in i+1..n-1
            t = abs(a[j] - a[i])
            if t < min
                min = t
            end
        end
    end
    min
end
```

2. (i 番目の値) − ($i-1$ 番目の値) の最小値を，i を 0 から $n-1$ まで順に動かして調べる．

```
def minabs(a)
    n = a.length()
    min = abs(a[1]-a[0])
    for i in 2..n-1
        t = abs(a[i] - a[i-1])
        if t < min
            min = t
        end
    end
    min
end
```

3. マージソートなど計算量が $O(n^2)$ 未満のソートアルゴリズムを用いて配列の中身を小さい順に整列したあと，2 で示したアルゴリズムによって調べる．たとえばマージソートの場合は計算量が $O(\log n)$ であるから，全体の計算量は $O(\log n)+O(n)=O(n+\log n)$ となる．

解説

1. 解答のとおり．日本語が下手で申し訳ない．
2. こちらも解答のとおり．for 文の範囲が 2 からになっているのは，1 の場合がすでに min に格納されているからである．
3. おそらくこれが最善の解答．バケットソート（計算量 $O(n)$）なども有効だろう．クイックソート，基数ソートに関しては与えられた数値や状況によって計算量が変わってくるので解答にするのは少し危ない．ソートせずに求める方法はとりあえず思いつかなかった．

▶ 問題 p.59

問 4　配列のソートアルゴリズム　〔2013 年度　第 1 問〕

解法のポイント

配列の整列のためのアルゴリズムに関する問題．教科書のコードそのままではあるのものの，マージソートの内容を知らなければ解くことは厳しいだろう．

解答

1. ア　from[n-1]
   イ　(n+1)/2
   ウ　from[0]

2.
   - `[[7],[2],[8],[3],[1],[4],[6],[5]]`
   - `[[2,7],[3,8],[1,4],[5,6]]`
   - `[[2,3,7,8],[1,4,5,6]]`
   - `[[1,2,3,4,5,6,7,8]]`
3. 計算量のオーダーとしては，配列の要素数を n としたとき，単純整列法は $O(n)$ で併合整列法は $O(\log n)$ であるため，併合整列法の方が優れている．一方メモリ量の観点から見れば，単純整列法は 1 つの 1 次元配列を操作するだけで良いが，併合整列法だと併合前と併合後の 2 つの 2 次元配列を用意せねばならず，単純整列法の方が優れているといえる．

解説

1. 教科書に掲載されているコードとまったく同じ．ただ，ちゃんとマージソートに関して理解していれば覚えていなくてももちろん解けるはずだ．
   ア 2 つどうしで併合するため，奇数個のものを併合して 2 分の 1 の個数にしようとしたとき，最後の 1 つだけ処理が変わってくる．それ．
   イ 次に操作する配列の要素数を n としている．したがって，`from.length()` や `to.length()` でも問題ない．
   ウ 併合し終わったあとに残る配列の形に注意しよう．小問 2 でもわかるとおり，配列の中に配列が入っている状態で出てくるが，求められているのは単なる配列である．
2. そのまま追っていけば良い．というかマージソートの方法を知っていればコードをなぞらなくても適当に書けるだろう．なお最後に `from` が `[[1,2,3,4,5,6,7,8]]` となっているとき，while 文の比較の真偽値自体は偽だが，それでも while 文の 1 行目は実行されている，と解釈している．
3. 計算量のオーダーに関しては教科書通りの内容．メモリ量に関しては，配列を使ってる個数に関して述べれば十分だろう．なお，ファイルに一時保存するなどすればマージソートの使用メモリ量をもっと抑えることも可能である．

▶ **問題** p.60

問 5　線形探索と二分探索 〔計算機プログラミング I（植田）1999 年度 第 3 問〕

解法のポイント

探索アルゴリズムの文章題．探索アルゴリズムとその計算量がわかっていればカンタン．正直このような問題が出るとは思えないがおもしろかったので掲載した．

解答

1. $\frac{n+1}{2}$ 回
2. 約 $\log_2 n$ 回

解説

1. 答えが 1 から n までで同様に確からしい．答え k を探すまで k 回かかるので，$\sum_1^n \frac{k}{n} = \frac{n+1}{2}$.
2. 二分探索を用いれば，1 回聞くと検索範囲が $\frac{1}{2}$ に，2 回聞くと検索範囲が $\frac{1}{4}$ になるので，聞く回数 p は $n \times \frac{1}{2^p} = 1 \Leftrightarrow p = \log_2 n$．実際にはそこまで簡単にならない（これよりも小さくなる）が，情報科学の試験問題の答えとしてはこれでいいだろう．

▶ **問題** p.60

問 6　線形探索と二分探索 〔2006 年度 第 2 問〕

解法のポイント

基本的な探索アルゴリズムの問題．1 と 2 は定義をすべて書く必要があるが問題文にのっとれば難しくない．3 は常識問題．

解答

1.
```
def linear(a,v)
    ans = -1
    i = 0
    while ans == -1 && i < a.size
        if a[i] == v
            ans = i
        end
        i = i+1
    end
    ans
end
```

2.
```
def binary(a,v)
    i_s = 0
    i_e = a.size - 1
    while i_e - i_s > 1
        i_m = (i_s + i_e) / 2
        if a[i_m] < v
            i_s = i_m + 1
        else
            i_e = i_m
```

```
        end
    end
    if a[i_s] == v
        i_s
    else
        -1
    end
end
```

3. 配列の要素数を n とすれば，1 が $O(n)$，2 が $O(\log n)$.

解説

1. 問題に載っているコードを参考にした．a.size は配列 a の要素数を表している．a.length() でも構わない．ans をとりあえず-1 にしておいて，v と同じ値の要素が見つかったもしくは最後まで調べたらループを抜けるようにしている．
2. 問題文で指示されたとおりの定義．i_s が探索する最初の添字，i_e が最後の添字である．絞り込み続けて i_s と i_e が等しくなったときにループを抜けるようにしている．最後に本当にその値の要素が配列 a に含まれているかを if 文によって調べているが，こちらは指示されていないので素直に i_s を返してもよいだろう．
3. 常識問題．理由については第 I 部の該当部を参照のこと．

▶ **問題** p.61

問 7　配列の操作と計算量 〔2008 年度 第 1 問〕

解法のポイント

関数の定義を自ら考え，その計算量を評価する問題．できる範囲でかしこい方法を考えよう．

解答

1.
```
def reverse(x,p,q)
    m = (p - q + 1)/2
    for i in 0..m-1
        t = x[p+i]
        x[p+i] = x[q-i]
        x[q-i] = t
    end
end
```

2.
```
def transpose1(x,k)
    n = x.length()
    for i in 0..k-1
        rotate(x,0,n-1)
    end
end
```

計算量は $O(kn)$.

3.
```
def transpose2(x,k)
    n = x.length()
    reverse(x,0,n-1)
    reverse(x,0,k-1)
    reverse(x,k,n-1)
end
```

計算量は $O(n)$

解説

1. まずどうやってひっくり返すかを考える．たとえば 1〜5 をひっくり返したかったら，1,2,3,4,5 のうち「1 と 5」と「2 と 4」を交換すればいい．1〜6 なら 1,2,3,4,5,6 のうち「1 と 6」，「2 と 5」，「3 と 4」を交換すればいい．上の解答では，この交換する回数を m とおいている．
   for 文の中では，x[p] と x[q] を交換し，x[p-1] と x[q+1] を交換し……という作業を行っている．直接交換はできないので変数 t を介している．これは変数を交換するときの常套手段．どうでもいいが Ruby では x[p],x[q] = x[q],x[p] とすると変数の値が交換できたりする．
2. 何回か回すと目的の配列になることはわかるだろう．[1,2,3,4,5,6] → [2,3,4,5,6,1] → [3,4,5,6,1,2] → … となるので，k 回 rotate すればいいことがわかる．
   計算量は (rotate を実行する回数) × (1 回の rotate の計算量) となるので，$O(k) \times O(n) = O(kn)$.
3. もともとの問題ではヒントとして「reverse を 3 回呼び出すようにすればよい」と書いてあったが，問題掲載の際に省略した．
   [1,2,3,4,5,6] → [6,5,4,3|2,1] → [3,4,5,6|1,2] というように，全体をひっくり返したあと前半と後半をそれぞれひっくり返すとうまくいく．
   計算量は (reverse(0,n-1) の計算量)+(reverse(0,k-1) の計算量)+(reverse(k,n-1) の計算量) = $O(n) + O(k) + O(n-k) = O(2n) = O(n)$.

▶ 問題 p.62

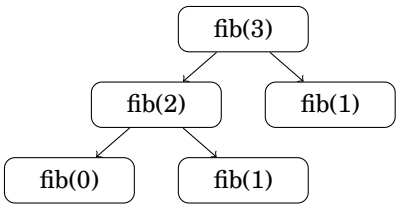
問 8　フィボナッチ数列と木構造 〔2007 年度 第 1 問〕

解法のポイント

フィボナッチ数列の再帰的定義を木構造で表す問題．教科書 p.87 にもそれらしきものがある．アルゴリズムの理解というよりは，木構造をうまく操作できる能力が必要だろう．木構造での表現を求める問題は出題の可能性があるが，ここまで木構造にこだわる問題が今後出ることは考えにくい．

解答

1.



2.

$$F(n) = \begin{cases} F(n-1)+F(n-2)+2 & \text{if } n > 1 \\ 0 & \text{otherwise} \end{cases}$$

3. $f(n)$
4. 完全 2 分木における葉の枚数とエッジの本数を考えればよい．ある状態で 1 枚葉が節になると，葉は全体で 1 枚，エッジは全体で 2 本増える．したがって，$F(n) = 2f(n) + c$ と表せる（c は定数）．単純な場合を考えて，根が 1 つと葉が 2 枚の 2 分木はエッジが 2 本で葉が 2 枚だから，$c = -2$ とわかる．したがって $F(n) = 2f(n) - 2$.

解説

1. 図のとおり．
2. $\mathrm{fib}(n)$ は $\mathrm{fib}(n-1)$ と $\mathrm{fib}(n-2)$ を呼び出しているので，まずそれらによって作られるエッジの本数（それぞれ $F(n-1)$ と $F(n-2)$）を足し合わせる．そこに $\mathrm{fib}(n)$ から $\mathrm{fib}(n-1)$ と $\mathrm{fib}(n-2)$ に伸びるエッジ 2 本を加える．
3. 教科書にも書いてある．この再帰的定義で値が生じる原因は $\mathrm{fib}(0) = 1$ と $\mathrm{fib}(1) = 1$ のみであるので，$\mathrm{fib}(n)$ と葉の枚数が一致するのは当然のことである．
4. 完全 2 分木（根と節からつねに 2 つの枝が伸びている木構造）は常にこれが成り立つ．$F(n)$ と $f(n)$ の漸化式がわかっているので，ここから数式をいじっても導ける．

第 2 章　数値計算と誤差

▶ 問題 p.65

問 9　記述問題・常微分方程式の解 〔2006 年度 第 3 問〕

解法のポイント

どれも知っていれば書ける問題．2 の常微分方程式の解法は試験範囲外であるため解けなくても支障はない．

解答

1. 丸め誤差　計算中に値の絶対値に比べてとても小さな桁を丸めた結果，計算結果が本来の値とずれること．
   例：`0.1*3-0.3` を浮動小数点で計算させると，0.1 のときに起きた丸めが 3 倍して増大し 2^{-54} の誤差が生じる．

   情報落ち誤差　ある数に対して極端に絶対値が小さい数をその数に足したり引いたりしたときに，小さい数が無視されてしまうこと．
   例：$3\times10^{20}-0.1$ を計算したら本来 `29999999999999999999.9` となるはずだが `3.0e+20` としかならない。

   打ち切り誤差　無限に続ければより本来の値に近くなる計算処理を途中で打ち切るために起こる誤差のこと．
   例：自然対数の底は $\sum_{n=0}^{\infty}\frac{x^n}{n!}$ で計算できるが，無限に計算できないために途中で打ち切ると本来の値よりも小さくなる．

   桁落ち誤差　計算結果が 0 に極端に近くなるような加減算を行ったときに有効数字の桁数が極端に少なくなること．
   例：有効数字 4 桁で $2.008-2.006$ を引き算すると，$2.008-2.006=0.002$ となって有効数字が 1 桁になってしまう．
2. Euler 法は単純でわかりやすいが，区間の出発点のみで微分を調べて計算するため計算間隔が大きいと精度が非常に悪くなるし，解が曲線の場合は最悪の場合本来の曲線からどんどん離れてしまう．RungeKutta 法は，1 次で近似する Euler 法と異なり多次（基本的には 4 次）で近似するため打ち切り誤差はほとんど現れない．

解説

1. 情報落ち誤差　Ruby の処理系だと $3\times10^{20}-1$ はちゃんと 29999999999999999999 として処理されるので小さい数の方は小数とするべき．

   打ち切り誤差　例にはそのほか微分や常微分方程式や積分などが使える．
2. Euler 法は打ち切り誤差のパラダイスなので，数値計算で常微分方程式解きたかったら（基本的に

は）RungeKutta 法使おうねってことだ．

▶ 問題 p.65

問 10　記述問題・2 次方程式の解 〔2010 年度 第 3 問〕

解法のポイント

基本的な問題．4 のプログラム例が長いが大したことはない．

解答

1. 無理数である円周率を 3.14 として計算するなど，本来の値の小さい部分を丸めたために起きる誤差のこと．
2. false（偽）と表示される．0.1 は 2 進法だと循環小数になるため丸めが起こり，3 倍した際にそれぞれが増大し，結果的に丸め誤差が蓄積して 0.3 よりも 2^{-54} 大きい値となる．
3. 3.092 と 3.088 はともに有効数字 4 桁だが 3.092 − 3.088 を計算すると 0.004 となって有効数字が 1 桁まで落ちるように，計算結果が 0 に極端に近くなるような加減算を行ったときに有効数字の桁数が極端に少なくなる現象のこと．
4. 2 次方程式の解の公式における $-b \pm \sqrt{b^2 - 4ac}$ が 0 に極端に近くなる際に起きる桁落ち誤差を防ぐため．

解説

1. 例はかんたんで十分だろう．ただ，`0.1*3-0.3` の話は使えないので今回はこのようにした．
2. 循環小数を処理するときにまるめが起きる，ということが書けていればたぶん OK.
4. 授業では扱わないがプリントには掲載している．2 次方程式の桁落ち誤差はほんとに大事．

▶ 問題 p.66

問 11　Gauss-Jordan 法とピボット選択 〔2012 年度 第 3 問〕

解法のポイント

Gauss-Jordan 法および pivoting といった，連立 1 次方程式の基本的な問題．

解答

1. ア　`1.0 / akk`
   イ　`aik * a[k][j]`
2. 浮動小数点による計算をしているため，与えられたり計算して出てきたりした 10 進実数を 2 進数に変換する際などに丸め誤差が発生する可能性がある．また，2 つの式の係数の値が非常に近い場合，対角化のために引き算した際の答えが 0 に近くなるので桁落ち誤差が発生する可能性がある．

さらに，a[k][k] が 0 に非常に近かった場合に a[k][i] が大きな値になり，空欄イの行で情報落ち誤差が発生する可能性がある．

3. 2 で記述したうちの情報落ち誤差やプログラム 7 行目での 0 除算を防ぐため，係数行列の対角成分（a[k][k]）の絶対値がなるべく大きくなるように行を入れ替えることを pivoting という．このプログラムで pivoting を行うには以下のように修正すればよい．

```
                                    ⋮
for k in 0..(col-2)
    swap(a,k,maxrow(a,k)) # この行を挿入する
    akk = a[k][k]
    for i in 0..(col-1)
                                    ⋮
```

4. 以下のように修正すればよい．

```
                                    ⋮
for k in 0..(col-2)
    swap(a,k,maxrow(a,k)) # 問題 3 で挿入した行
    akk = a[k][k]
    if akk == 0 # ここから
        abort()
    end # ここまでを挿入する
    for i in 0..(col-1)
                                    ⋮
```

解説

1. プログラムコードは教科書および配布プログラムそのまま．

   ア　ここでは対角成分が 1 になるように第 k 行を $\frac{1}{\texttt{a[k][k]}}$ 倍している（正規化している）．空欄の直前に∗があるのは，1 ではなく 1.0 と書いて欲しいと示唆するためのものだろう．

$$\begin{pmatrix}2 & 3\\ 5 & 2\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}12\\ 19\end{pmatrix}\rightarrow\begin{pmatrix}\mathbf{1} & 1.5\\ 5 & 2\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}6\\ 19\end{pmatrix}$$

   イ　ここではさきほど正規化した行の a[i][k] 倍を第 i 行から引いている．こうすることで第 k 行以外の k 番目の変数の係数を 0 にしている．

$$\begin{pmatrix}1 & 1.5\\ 5 & 2\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}6\\ 19\end{pmatrix}\rightarrow\begin{pmatrix}1 & 1.5\\ 5-5\times1 & 2-5\times1.5\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}6\\ 19-5\times6\end{pmatrix}\rightarrow\begin{pmatrix}1 & 1.5\\ \mathbf{0} & -5.5\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}6\\ -11\end{pmatrix}$$

2. いろいろ考えられるため列挙するのが最善だが，おそらく pivoting する必要性（情報落ち誤差）を説明すればいい．

3. 教科書では 2 行で書かれていたが，解答では 1 行に省略した．k 番目の変数の係数が k 行目で一番大きくなるように入れ替える．
4. 結局 pivoting しても a[k][k] で割れない場合に解が求められないことになる．すなわち pivoting した結果が 0（もしくはそれに非常に近い値）であるときに求められない．「できるだけ早く」と書いてあるので，相似な式があること（あとは係数がすべて 0 の行があるとか）を最初に確認するのが一番早い気もするが，おそらくこの解答を求めているのだろう．

▶ **問題** p.67

問 12　台形公式とその誤差 〔2009 年度 第 2 問〕

解法のポイント

数値積分の方法と誤差に関する理解さえあれば解けるだろう．1 は台形公式の計算式を示すことが必要なため，その場で導ける力が必要だ．

解答

1. 積分範囲を n 等分し，その幅を Δx とする．k 番目（いちばん左を 0 番目とする）の区間における面積を，上底 $f(x_s+k\Delta x)$，下底 $(x_s+(k+1)\Delta x)$，高さ Δx の台形として近似するとその値は

$$\frac{1}{2}\{(x_s+k\Delta x)+(x_s+(k+1)\Delta x)\}\Delta x$$

となる．0 番目から $n-1$ 番目までの台形を足し合わせることで求める積分値を近似し，その値は

$$\sum_{k=0}^{n-1}\frac{1}{2}\{(x_s+k\Delta x)+(x_s+(k+1)\Delta x)\}\Delta x=\Delta x\left\{\frac{f(x_s)+f(x_e)}{2}+\sum_{k=1}^{n-1}f(x_s+k\Delta x)\right\}$$

である．
2. 分割を多くすると足し合わせられる台形（この場合は長方形）の面積が非常に小さくなるので，単純に足しあわせていくと，途中まで足しあわせて得られた面積に比べてさらに足し合わせるべき面積が非常に小さくなるため情報落ち誤差が発生する．
3. 分割が少ないと，台形公式では関数を折れ線と見て計算しており曲線の関数では近似が荒くなるため，本来の値より遠くなる（この場合は小さくなる）という打ち切り誤差が発生する．
4. 台形公式のように 1 区間中の関数を直線に近似するのではなく，ラグランジュ補完によって 2 次関数に近似する（合成）シンプソン公式がある．

解説

1. 台形公式の説明．その場で導出する過程を記述する．本編では区間を 1 番目から n 番目と呼んでいたが，ここでは教科書に合わせて 0 番目から $n-1$ 番目と呼んでいる．結果の公式は変わらない．

2. あまりに小さい数で $\sum$ を順番にたくさんやっていくと情報落ち誤差が発生する．今まで足しあわせて出てきた値に比べて，次に足しあわせる値があまりにも小さいからだ．
3. これは単純に近似が荒いせいで起こる誤差．2 と 3 の誤差に関しては教科書 p.127 にグラフが載っているので，これもあわせて見ておこう．
4. ガウス求積法，ロバット積分則，クレンショウ・カーチス則などいろいろあるが，ここではシンプソン公式について説明しておけばよいだろう．なお紹介したいくつかの積分方法はすべてサンプルを取る点を工夫することで本来の値に近づけるというものである．

▶ **問題** p.67

問 13　Gauss-Jordan 法とピボット選択 〔2007 年度 第 2 問〕

解法のポイント

Guass-Jordan 法と pivoting に関する問題．コードが教科書のものとは違うために戸惑うかもしれないが，やっていることは同じ．ゆっくり考えて正しい答えを導き出そう．

解答

1. ア　`a[i][j] = a[i][j] - a[k][j]*aik`
   イ　`a[i][col-1]`
2. まず対角成分が 0 となる行があると，`a[k][i] = a[k][i]/akk` のところで 0 で割り算を行うことになりエラーを起こしてしまう．またそうでなくとも絶対値が非常に小さい値をとるときに，この式で `a[k][i]` が非常に大きな値となり，空欄アの行の引き算で情報落ちが発生する．そのため，対格要素の絶対値がなるべく大きくなるように行を入れ替える pivoting を行えばよい．

解説

1. 対角成分が 1 になった `k` 行目の `a[i][j]` 倍を，`i` 行目の値から引くところ．
   一番右側の列を取り出している．左側が単位行列になっているため，右側の列が解となっている．
2. pivoting の必要性を説明すればよいだろう．

第 3 章　乱数

informatical reality

▶ 問題 p.71

問 14　乱数の理解・モンテカルロ法およびその応用　〔2008 年度　第 2 問〕

解法のポイント

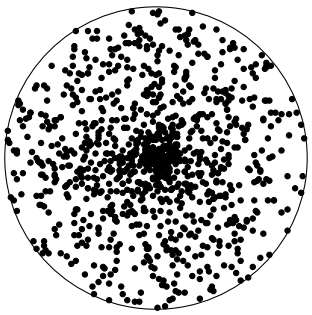
乱数に関する基本的な理解をしておこう．

解答

1. 出現確率が値によらず一定である乱数．
2. サイコロを振ったときに出る値などの真の乱数とは異なり，今まで出た値をもとにして統計的に乱数であることが有意であるように作られた数．
3. 解析的に解くことが困難または不可能な問題や，確率的な動きをする対象のシミュレーションに用いられる．
4. r が一様乱数になっているため r が大きくなっても角度に対する点の分布確率は等しいので，単位面積当たりで考えると r が大きい場所（すなわち円の外側）の方が密度が薄くなり，均一に分布しない．

解説

3. モンテカルロ法は単なる数値積分手段だけでなく，ランダム性のあるミクロな物質の運動や，確定的には決められない金融の計算など，確率的な事象をシミュレーションするのにも多く用いられている．
4. 実際に点を打ってみると右の図のようになる．あからさまに中心の方が濃い．解決策としては，`r = sqrt(rand())` とする方法，または点を何度も打って円の中に入るものを採用する方法が考えられる．



▶ 問題 p.71

問 15　ランダムウォーク　〔2011 年度　第 3 問〕

解法のポイント

乱数をからめたプログラム理解の問題．後半は乱数の知識がたとえまったくなくとも解けるのであきらめずいこう．

解答

1. 一見するとでたらめな乱数のようであるが，実際は今まで出た値を使って統計的に乱数であると有意になるように算出した数.
2. 十分に周期が長いこと．値の出現の偏りがないこと．計算が速いこと．など
3. `(y-x*(dy/dx)) ** 2 < 1`
4. イ　`x = -x1`
   ウ　`x = x1`

解説

2.「周期が十分に長い」がおそらくもっともよい解答だろう.
3. (x,y) と (x_1,y_1) を結ぶ直線の切片 y_0 が $-1 < y_0 < 1$ であればよい.

$$y_0 = y + \frac{x}{x-x_1}dy = y - \frac{xdy}{dx}$$

となる．$x \geq 0$ で $x_1 < 0$ なので dx が 0 になることはない．解答では 2 乗することで 2 度 y_0 を計算することを避けているが，素直に `y-x*(dy/dx) < 1 && y-x*(dy/dx) > -1` と書いてもよいだろう．イコールが入るかどうかは問わないと思う．なお，手抜きして`-1 < y1 && y1 < 1` としてもある程度点数は来そうである.

4. イ　仮置きした x 座標 `x1` に対し，反射すると x 座標は正負が逆になるので，最終的な x 座標は `-x1`.
   ウ　仮置きした値そのままでよい.

第 4 章　動的計画法

informatical reality

▶ 問題 p.73

問 16　アラインメント　〔2007 年度　第 3 問〕

解法のポイント

教科書や本書で扱ったアラインメントを求める作業を再現するだけ．ミスなくこなそう．

解答

1. ア 3　イ 1　ウ -1　エ 1　オ 2　カ 3　キ 2　ク 0　ケ 4
2.
```
AGTGGGA
AG-GTAA
```

解説

1. 教科書どおり．表を完成させると以下のようになる．

		A	G	T	G	G	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	2	0	-2	-4	-6	-8	-10
G	-4	0	4	2	0	-2	-4	-6
G	-6	-2	2	3	4	2	0	-2
T	-8	-4	0	4	2	**3**	**1**	**-1**
A	-10	-6	-2	2	3	**1**	**2**	**3**
A	-12	-8	-4	0	1	**2**	**0**	**4**

2. こちらも教科書通り，遡っていけばいい．
   4 →左上（2）→左上（3）→左上（4）→左上（2）→左（4）→左上（2）→左上（0）．

▶ 問題 p.73

問 17　経路の数　〔2010 年度　第 4 問〕

解法のポイント

非常にカンタン．1 はそこにまっすぐ 1 通りでしか行けないことに気づこう．2 はもう全部問題に答えが書いてある．

解答

1. $T_{0,0} = T_{0,i} = T_{j,0} = 1$.

2. まず, $(M+1)\times(N+1)$ の表を 2 次元配列として作成する．1 のように $T_{0,0}, T_{0,i}, T_{j,0} (1 \leq i \leq M, 1 \leq j \leq N)$ を定める．そして問題文にある漸化式を用いて $T_{1,1}$ から順番に $T_{M,N}$ まで求めていく．最後に求められた $T_{M,N}$ を参照して返せばよい．

解説

1. 回り道しないため，初期位置に行く方法は 1 通りしかないし，辺上にたどるのも 1 通りしかない．
2. 表を書いてそこに数字を埋めていくイメージ．まず一番下と一番左に 1 を埋めて，そのあと空いているマスに，小問 1 の問題文にある通り 左下＋下＋左（$T_{m,n} = T_{m,n-1} + T_{m-1,n} + T_{m-1,n-1}$）を書けばよい．実際に表を書いてみるとこんな感じ．

$n \setminus m$	0	1	2	3	4	5	6	7
6	1	13	85	377	1289	3653	8989	**19825**
5	1	11	61	231	681	1683	3653	7183
4	1	9	41	129	321	681	1289	2241
3	1	7	25	63	129	231	377	575
2	1	5	13	25	41	61	85	113
1	1	3	5	7	9	11	13	15
0	1	1	1	1	1	1	1	1

► **問題** p.74

問 18　組合せ最適化問題 〔2008 年度 第 3 問〕

解法のポイント

後ろの方の難易度が高い．1 はやるだけだし 2 もお決まりのことを書くだけなので，この 2 つは取っておきたい．

解答

1. 11 回.
2. 呼び出した rec の計算結果を保存できないから．たとえば `rec([2,3],7)` を呼び出したとき，まず 2 を使ってその後 3 を使ったときに `rec([2,3],2)` が呼び出されるし，まず 3 を使ってその後 2 を使ったという状況でも `rec([2,3],2)` が呼び出されてしまう．
3. ア　`0`
   イ　`k`
   ウ　`n`
4. `rec[a,m]` のデータが配列 `b` の要素 `b[m]` として保存されているから．

解説

1. 実際にやるだけ．書き下してみると下のようになる．[2,3] は a と略記した．
   - rec(a,7)
     - rec(a,5)
       - rec(a,3)
         - rec(a,1)
         - rec(a,0)
       - rec(a,2)
         - rec(a,0)
     - rec(a,4)
       - rec(a,2)
         - rec(a,0)
       - rec(a,1)
2. 上の書き下したものを見れば，同じものが 2 回呼び出されていることがわかるだろう．
3. 小さい方から表を埋めている．方針としては，
   (a) 0 を作るには 0 個の数を組み合わせればいい．
   (b) ある値を作るには，(その値)−(使える値) を作れる最低の個数 +1 で作ることができる．全部調べれば，最も少ない個数が調べられる．たとえば，2 と 3 が使える値で 6 を作るためには，$6-2=4$ を作る個数 +1 か，$6-3=3$ を作る個数 +1 で作ることができる．この場合後者が 2 となるので，6 は 2 つ（3 と 3）で作ることができる．
   求められない数字に関しては，もともと −1 が入っている．(その値)−(使える値) が −1 になってしまった場合にはカウントされない．なお，0 より大きくて，使える値よりも小さい値は自明の −1 である．
   (c) これを求めるべき値までくり返す．
   という感じ．

▶ **問題** p.75

問 19　経路と累積利益 〔2011 年度 第 4 問〕

解法のポイント

いかにも動的計画法向きの問題．まずはその位置に来るにはどうしたらよいのかを把握して漸化式を組み立てていこう．1 が解ければ 2 と 3 はカンタン．

解答

1. $\mathrm{maxGain}(i,j) = \max(\max(\mathrm{maxGain}(i-1,j-1),\mathrm{maxGain}(i-1,j)),\mathrm{maxGain}(i-1,j+1)) + \mathrm{gain}(i,j)$.

2. $\frac{3^m-1}{2}$ 回.
3. ア　`0`
   イ　`gain(i,j)`
   ウ　`max(max(totalGain[i-1][j-1], totalGain[i-1][j]), totalGain[i-1][j+1]) + gain(i,j)`

解説

1. (i,j) には，$(i-1,j-1)$，$(i-1,j)$，$(i-1,j+1)$ の 3 通りの方法で来ることができる．それぞれの最大累積利益に，(i,j) での利益を加えれば，上の漸化式が得られる．
2. 1 つの `maxGain` から，3 つの `maxGain` が呼び出されている．m 段目の `maxGain` は 1 回，$m-1$ 段目の `maxGain` は 3 回，$m-2$ 段目の `maxGain` は 9 回呼び出されているので，これを 1 段目まで続けると，呼び出される回数 x は

$$3^0+3^1+\cdots+3^{m-1}=x \Leftrightarrow x=\frac{3^m-1}{2}$$

   となる．
3. $(m+1)\times(n+2)$ の表を書いて求めている．まず，格子領域に収まっていない部分の利益を 0 としている（空欄ア）．1 行目の最大利益はそこでの利益にきまっているので，そのように初期化している（空欄イ）．あとは，余ったところを 1 で求めた漸化式にのっとって埋めている（空欄ウ）．

▶ **問題** p.77

問 20　最短路問題　〔2009 年度　第 3 問〕

解法のポイント

問題はいかついが，実際に難しいのは 4 と 5．この 2 つはできなくともほかを取りこぼさないようにしたい．

解答

1. `[1,0,0,1,1,0,1,1,0]`
2. ${}_{m+n-2}C_{m-1}$
3. ア　`-3`
   イ　`j = j+1`
   ウ　`cost+c(i,j)`
4. 126 回.
5.
```
$min_cost = make2d(m(), n())
for i in 0..m-1
    for j in 0..n-1
```

```
        $min_cost[i][j] = -1
    end
end
$min_cost[m()-1][n()-1] = c(m()-1,n()-1)
def get_min_cost(i,j)
    if $min_cost[i][j] != -1
        $min_cost[i][j]
    else
        south = 1000000
        east = 1000000
        if i < m()-1
            south = get_min_cost(i+1,j)
        end
        if j < n()-1
            east = get_min_cost(i,j+1)
        end
        $min_cost[i][j] = c(i,j)+min(south,east)
    end
end
```

解説

1. 問題文の図から，東，南，南，東，東，南，東，東，南，の順に進んでいる．
2. 中学受験の問題．
3. 順番に道のりをたどりながらコストを加えていくだけ．
4. `get_min_cost(i,j)` は，`get_min_cost(i-1,j)` と `get_min_cost(i,j-1)` からしか呼ばれない．また，`get_min_cost(0,0)` は 1 度だけ呼ばれる．したがって，これを満たすように下のように表を書く．

$i \setminus j$	0	1	2	3	4	5
0	1	1	1	1	1	1
1	1	2	3	4	5	6
2	1	3	6	10	15	21
3	1	4	10	20	35	56
4	1	5	15	35	70	**126**

   すると `get_min_cost(4,5)` の呼ばれる回数が 126 回であることがわかる．単純に手でゴリゴリやって出すのは苦しいだろう．
5. `$min_cost` に登録されているかを確認し，登録されていればそれを呼び出すようにすればよい．

そのまま再利用せよ，との指示があるのでこのような形になるが，実際には表を順番に埋めていったほうが早いだろう．

第 5 章　その他

informatical reality

▶ 問題 p.79

問 21　計算の特殊定義　〔2011 年度　第 2 問〕

解法のポイント

何をやっているのか 1 の段階でわからなくとも，その答えから推測できればよいだろう．

解答

1. f(2,4)　16
   f(3,5)　243
2. $c = a^b$.

解説

1. 気付かなくとも手でゴリゴリやれば出る．
2. いわゆる繰り返し二乗法．うまい説明ができないので解説は略．1 から推測するもよし．うだうだ考えるもよし．

▶ 問題 p.79

問 22　整数の数え上げ　〔2010 年度　第 2 問〕

解法のポイント

そこまで難しくはない．ただ，1 は迷いどころがちょっとあるかも．

解答

1. $O(mn)$.
2.
```
def intcount(a,b,c)
    b[0] = a[0]
    c[0] = 1
    j = 0
    for i in 1..(a.length()-1)
        if b[j] != a[i]
            j = j+1
            b[j] = a[i]
```

```
        end
        c[j] = c[j] + 1
    end
end
```

解説

1. 外側のループが n 回なのはいいだろう．問題は内側のループだが，今探している値が b のどれに入っているかを順番に探している．b の要素数は最大で m なので，内側のループの平均回数は $\frac{m}{2}$ くらいだと見積もっていいだろう．したがって，全体の計算量は $O(mn)$ となる．
2. 小さい方から順番に見ていく．さっきまで出ていた値と今の値が違った場合，その値は今まで出てきていないので，新しくしてしまっていい．計算量は $O(n)$.

▶ 問題 p.80

問 23　整数列の和　〔2011 年度　第 1 問〕

解法のポイント

小問の数が多いが，ほとんど問題文にのっとってやっていけばできるはず．

解答

1. $s(\mathrm{a},0,2)=4$，$s(\mathrm{a},0,3)=-1$，$s(\mathrm{a},0,4)=1$.
2. $O(N)$.
3. $O(N^3)$.
4.

i	0	1	2	3	4	5	6	7	8	9
sum	8	4	0	2	6	1	6	9	2	10
t	8	8	8	8	8	8	8	9	9	10

5. $O(N)$.

解説

1. 和を求めるだけ．たとえば $s(\mathrm{a},0,4)=8+(-4)+(-5)+2=1$.
2. 配列の要素を 1 つずつ足しあわせていけばいいので，明らかに計算量は $O(N)$.
3. $s(\mathrm{a},i,j)$ の計算量は $O(j-i)$．内側のループで j は i から $y(=N)$ まで動くので，内側のループ内での計算量は

$$\sum_{j=i}^{N} O(j-i) = O\left(\frac{i(2N-i)}{2}\right) = O(Ni-i^2)$$

となる．さらに，外側のループで i は 0 から $x(=N)$ まで動くので，

$$\sum_{i=0}^{N} O(Ni - i^2) = O\left(\frac{N^2(N+1)}{2} + \frac{N(N+1)(2N+1)}{2}\right) = O(N^3)$$

となる．

4. 書いてあるプログラム通りにやるだけ．sum が和を順番にとっていって，t はその中の最大を求めている．ただ，sum が 0 未満になりそうなときは 0 にしてしまう．だって，0 未満になっちゃうならそれまでのやつを含めなければいいもんね．
5. 掲載されているプログラムから明らかに $O(N)$ であることはわかるだろう．

▶ **問題** p.81

問 24　再帰関数による探索 〔2012 年度 第 4 問〕

解法のポイント

問題文は長いが，やっていることは単純．パズル感覚で解いていこう．

解答

1. 1, 0, 0, 2, 2.
2. `"11011"`
3. `post(["1", "0", "00"], ["0", "011", "0"], 3, 5, "", "")`
4. $O(m^n)$

解説

1. コードをなぞってもいいが，おそらく手作業で探したほうが早い．b の`"011"`を最初にもってくると結構すぐにわかるだろう．つなげると`"0110000"`となるはずである．
2. コードを追っていこう．if 文だの while 文だのを通って `post` が内部から呼び出される．`k=1` のときに `as+a[k] == bs+b[k]` となるので，p には `as+a[k]` すなわち`"11011"`が代入される．それが答え．
3. as と bs はもともと用意されている文字列，n はあと最大何個の要素をつなげるか，ということを表しているようである．そこから考えれば上のような答えが妥当．なお n の部分の値は調べる量を表しているので 5 以上であれば間違いではない（満点とは限らないが……）．
4. 結果的に m 種類の要素を n 個つなげる組み合わせを全部調べている．その並べ方は m^n 通りあり，すなわち計算量は $O(m^n)$ である．

▶ 問題 p.82

問 25　バケツの問題 〔2009 年度　第 1 問〕

解法のポイント

座標上に状態を設定してバケツの水汲みを行う問題．同じような操作を繰り返していけば解答にはたどり着ける．

解答

1. fill(A), move(A, B), empty(B), move(A, B), fill(A), move(A, B).
   別解: fill(B), move(B, A), fill(B), move(B, A), fill(B), move(A, B).
2. (a) fill　対応する座標が最大値（a または b）となる．
   empty　対応する座標が 0 になる．
   move　(x, y) を通る傾き -1 の直線上を移動する．
   (b) x 座標もしくは y 座標の値が q と等しくなる．
   (c) まず最初に $(x, y) = (a, 0)$ などとしておく．そこから上に向かって，$x = 0, x = a, y = 0, y = b$ のどれかにぶつかるまで傾き -1 の直線上を移動させる．軸にぶつかればそちらの座標を max にし，$x = a$ または $y = b$ にぶつかればその座標を 0 にする．この直線上の移動を，x 座標または y 座標が q となるまでくり返す．
3. fill(B), move(B, A), fill(B), move(B, A), empty(A), move(B, A), fill(B), move(B, A), fill(B), move(B, A). 別解: fill(A), move(A, B), empty(B), move(A, B), fill(A), move(A, B), empty(B), move(A, B), empty(B), move(A, B), fill(A), move(A, B).

解説

1.「ビル・ゲイツの面接試験」として有名な問題[*1]．これは単純に考えればよい．本解は以下の様な操作をしている．別解も確認してみてね．

操作	A: 5L	B: 3L
A を満タンに	5L	0L
A から B に移す	2L	3L
B を空に	2L	0L
A から B に移す	0L	2L
A を満タンに	5L	2L
A から B に移す	**4L**	3L

[*1] 『ビル・ゲイツの面接試験――富士山をどう動かしますか?』（ウィリアム・パウンドストーン，松浦俊輔訳，2003）にも掲載されている．

2. (a), (b) はよいだろう．この設定された座標平面上でぐるぐるうまく点を移動させて，q まで辿り着くようにすればよい．
3. 2 を使っても使わなくても解けるだろう．これも本解の手順を書いておく．別解の方は 2 手順多いが，こちらでもある程度の点数はくるだろう．

操作	A: 8L	B: 5L
B を満タンに	0L	5L
B を A に移す	5L	0L
B を満タンに	5L	5L
B を A に移す	8L	2L
A を空に	0L	2L
B を A に移す	2L	0L
B を満タンに	2L	5L
B を A に移す	7L	0L
B を満タンに	7L	5L
B を A に移す	8L	**4L**

おまけ

本問題に解答する Ruby のメソッドを書いておく．

```ruby
def backet(a,b,q)
    ta = 0 # a のバケツを 0 リットルで初期化
    tb = 0 # b のバケツを 0 リットルで初期化

    # まずは a をいっぱいにしてからのパターン
    astr = "" # a の手順を入れておく文字列
    ta = a # まず a を満タンにする
    counta = 1 # カウンター
    astr += "fill(A)"
    while ta != q && tb != q # どちらかが q になったらおしまい
        counta += 1
        if ta != 0 # a がいっぱいでなくて，
            if tb != b # b がいっぱいでないなら移す
                if b - tb > ta # a に入ってるものが全部うつせる
                    tb += ta
                    ta = 0
                else
                    ta -= b - tb
```

```
                tb = b
            end
            astr += ", move(A,B)"
        else # a がいっぱいでなくて b がいっぱい
            tb = 0 # b を空に
            astr += ", empty(B)"
        end
    else # a が空なら
        ta = a # a を満タンに
        astr += ", fill(A)"
    end
    if ta == a && tb == 0 # 初期状態に帰ってきたら
        counta = -1 # 解なしの処理
        astr = "no answer"
        break
    end
end
astr += " (count:" + counta.to_s + ")"

# b をいっぱいにしてからのパターン a と同様
bstr = ""
ta = 0
tb = b
countb = 1
bstr += "fill(B)"
while ta != q && tb != q
    countb += 1
    if tb != 0
        if ta != a
            if a - ta > tb
                ta += tb
                tb = 0
            else
                tb -= a - ta
                ta = a
            end
            bstr += ", move(B,A)"
        else
```

```
            ta = 0
            bstr += ", empty(A)"
        end
    else
        tb = b
        bstr += ", fill(B)"
    end
    if ta == 0 && tb == b
        countb = -1
        bstr = "no answer"
        break
    end
  end
  bstr += " (count:" + countb.to_s + ")"

  # 出力処理
  if counta == -1 && countb == -1
    "no answer"
  elsif counta == -1 || (countb < counta && countb != -1)
    bstr
  else
    astr
  end
end
```

第 6 章　オブジェクト指向*

▶ 問題 p.85

問 26　メソッド定義 〔2006 年度 第 1 問〕

解答

1.
```
def advance(s)
    @second += s
    @minute += @second / 60
    @second %= 60
    @hour += @minute / 60
    @minute %= 60
end
```

別解:

```
def advance(s)
    @second += s
    while @second > 60
        @second -= 60
        @minute += 1
    end
    while @minute > 60
        @minute -= 60
        @hour += 1
    end
end
```

2.
```
def back(s)
    @second -= s
    while @second < 0
        @second += 60
        @minute -= 1
    end
    while @minute < 0
        @minute += 60
        @hour -= 1
```

```
    end
end
```

1 が本解であった場合のみ有効な別解:

```
def back(s)
    advance(-s)
end
```

▶ 問題 p.86

問 27　クラスの構造 〔2008 年度 第 4 問〕

解答

1. 得点の配列を入れ替えるのと同時に，氏名と学生証番号の配列も入れ替えればよい．
2. 氏名，学生証番号，得点．
3. コンストラクタとそれぞれの値を参照・変更するメソッド．
4. オブジェクトの持つ得点の情報をもとにして，オブジェクト自体をマージソートなどで整列させればよい．
5. オブジェクトの中に新たな変数（たとえば生年月日などの情報）を加えても，得点順に整列するメソッドを変更しなくてよい．レコードがひとまとまりになっているので，学生の追加，削除などが行いやすい．

▶ 問題 p.86

問 28　オブジェクト指向の利点 〔2007 年度 第 4 問〕

解答

カプセル化

1. オブジェクト内部のメソッドや変数のうち，ユーザーが直接必要としないものや，ユーザーによって変更されたくない・実行されたくないものをプライベートにすること．
2. 変数定義やメソッド定義をかえたくなった場合に，プログラム全体の大幅な変更を要する問題．また，具体化しすぎて中身がわかりにくくなるという問題．

継承

1. あるオブジェクトが，別のオブジェクトの変数やメソッドなどの特性を引き継ぐこと．
2. 似たようなオブジェクトを作る際に，同じコードをもう一度書く必要があるという問題．似た型であっても，型が特定されている配列などに入れられないという問題．

ポリモルフィズム

1. オブジェクトの型が違っていても，メソッドや式などを同じように使えるようになっていること．
2. 複数の型に対して同じようなメソッドを用意しようとしたときに，具体的にそれぞれ別の名前の別のメソッドを用意しなければならず，可読性が低くなりコードが煩雑になるという問題．

▶ **問題** p.86

問 29　オブジェクト指向の利用 〔2009 年度 第 4 問〕

解答

1. `f <= x && x <= t`
2. `self.set1.is_member(x) || self.set2.is_member(x)`

第 7 章　オートマトン*

informatical reality

▶ 問題 p.89

問 30　3 進法のオートマトン　〔2006 年度　第 4 問〕

解法のポイント

一番下の位に 0, 1, 2 が増えたとき，それぞれどうなるかを考えよう．

解答

ア，イ　0，2（順不同）
　　ウ　1
エ，オ　0，2（順不同）
　　カ　1

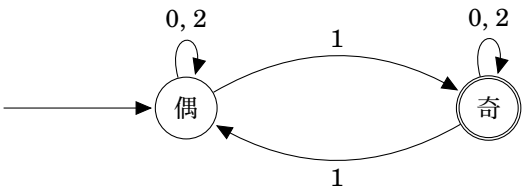
解説

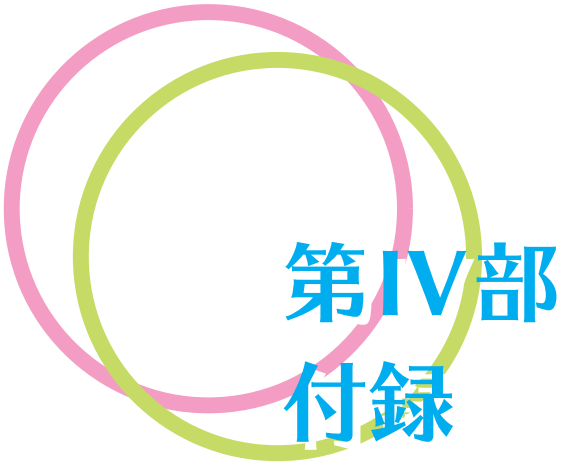
3 進法の数の 1 番下の位に 0, 1, 2 が増えたときのことをそれぞれ考えよう．

0　3 倍されるだけ．
1　3 倍されて 1 足される．
2　3 倍されて 2 足される．

3 倍されても偶奇は変わらないため，下の位に 1 がつくと偶奇が変わり，0 と 2 がついても偶奇はそのまま，ということがわかる．

埋めるとこうなる．





第IV部
付録

この部では付録として，本書を作成する上で使わせてもらった参考文献と，各概念を逆引きで確認することのできる索引を掲載した．索引は日本語のみでなく英語の名前でも掲載している．うまく有効活用して欲しい．

付録 A　参考文献

専門書籍

[1] 増原英彦，東京大学情報教育連絡会．情報科学入門 Ruby を使って学ぶ．東京大学出版会，2010，237p.，ISBN 4-13-062452-7.

[2] 五十嵐健夫．データ構造とアルゴリズム．数理工学社，2007，116p.，ISBN 4-901683-49-4.

[3] デビット・トーマス，アンドリュー・ハント．プログラミング Ruby．田和勝訳．ピアソン・エデュケーション，2001，665p.，ISBN 4-89471-453-1.

[4] 秋葉拓也，岩田陽一，北川宜稔．プログラミングコンテスト チャレンジブック．第 2 版，マイナビ，2012，367p.，ISBN 4-8399-4106-2.

講義資料

[5] 須田礼仁．連続系アルゴリズム 講義スライド．[n.d.].

[6] 久野靖．“情報科学 2013（久野クラス）資料等”．Information Sciences 2013 Kuno's Class．http://lecture.ecc.u-tokyo.ac.jp/~kuno/is13/siryou/.

シケプリ

[7] 三浦拳（2009 年度理科 I 類 13 組）．情報科学シケプリ（06 冬～08 冬過去問解答例）．2012，6p. http://todai.info/sikepuri/search/show.php?id=1190.

[8] khRules．2008 年度 情報科学 期末試験 略解．2012，5p. http://todai.info/sikepuri/search/show.php?id=1192.

[9] 暇人．情報科学 2010 年度 冬学期 解答&解説．2011，10p. http://todai.info/sikepuri/search/show.php?id=944.

[10] 2012 年度理科 I 類 10 組情報シケ対．情報科学 2011 年度 共通問題 解答例・解説．2013，4p. http://todai.info/sikepuri/search/show.php?id=1617.

一般書籍

[11] 奥村晴彦．LaTeX 2_ε 美文書作成入門．第 5 版，技術評論社，2010，433p.，ISBN 4-7741-4319-4.

同人誌

[12] 梱枝りこ．むじんのくにのアリス．無人少女，2012，24p.

ウェブサイト

[13] 東京大学教養学部 情報図形科学部会．“情報科学 共通資料（2013 年度）”．http://lecture.ecc.u-tokyo.ac.jp/johzu/joho-kagaku/2013/

[14] “Ruby 1.8.7 リファレンスマニュアル”．るりま．http://docs.ruby-lang.org/ja/1.8.7/doc/index.html.

[15] 戯画．“キスアト Official Web Site”．http://products.web-giga.com/kissato/.

[16] SAGA PLANETS．“SAGA PLANETS OFFICIAL HOMEPAGE”．http://sagaplanets.product.co.jp/.

[17] “うぃっきー！ ういっきぺでぃあーっ！ ”. http://ja.wikipedia.org/wiki/.

付録 B　索引

あ〜お

か〜こ

さ〜そ

た〜と

な〜の

は〜ほ

ま〜も

や〜よ

ら〜ろ

あとがき

informatical reality

小さい頃からプログラムコードを書き続けて十数年．長いこといろいろやってきたわりには，基本的なアルゴリズムすら知らなかったんだなぁって反省してしまう．そもそも二分探索を知ったのが大学 1 年だった．『実践的プログラミング』とかいう主題科目で出てきて「はぁー!」っていう感じだったね．

そういえば大学入りたての頃は完璧にプログラムが書けなくなってた気がする．それとももともとあんなレベルだったんだろうか．とりあえず，配られた資料のアルゴリズムもわからず適当にプログラムを組み立てて OK もらって単位をもらうっていう謎な授業だった．何も理解せずにプログラムを書いていた．一体なにが得られたんだ．

話を聞くと，プログラムを書いてる先生たちやそういった仕事の人たちは，ずっとずっとプログラムを書き続け，定年になると「やっと自由にプログラムが書けるぞー!」とか言い出すらしい．頭おかしいのでは．自分もそんな風になってしまうのかと思うと恐ろしいのだが，まあならないだろう……．なるかもしれないな．じんせいわからんし．

そういうわけで情報科学とかいう授業を取ってシケ対にならざるを得ない状況で Ruby のコードを夏休みからひたすらに書きまくってなんとかプログラミング力をつけてこの本の完成．実際には中身にあんまりコードがなくてなんかアレって感じだけど，まあ過去問のところにたくさんあるからいいでしょ．

過去問多いよね．あれ全部載せたの馬鹿らしくなってきた．まあでも全部解いてよ．ヒマでしょ?

実際自分としても情報科学っていう科目はプログラミング力つけるのに役に立ったなぁという感じがしてる．あれだけプログラムまともにガツガツ書いたの始めてだったのかもね．

ノンリニアっていうサークルでプログラム書いてるけど，つらい．やっぱり大きなプログラム書こうとすると，いろいろうまくいかないことも多いよね．うまい方法を試行錯誤しながら探していく．そうやって自分としても経験を積み重ねて学んでいくんだなぁ感じ．まだまだ書かなきゃダメだね．

プログラムを書く能力だけじゃなくて，いかによく書くかも大事だよね．線形探索じゃなくて二分探索で探すことにどれくらいの意味があるのかわからないけど，それでもやっぱりそういう小さな努力で大きなパフォーマンスが得られるんじゃないかな．

まだ並列処理とかそういうのもやってないからなにもわからないけどさ．

TEX だってプログラミング言語なんだよね．TEX で素因数分解のプログラムかけるし．

こう，いかにうまいデザインを簡単にやってのけるかっていうのが問われる気がする．結構これも頑張ったつもりだけど，やっぱり Perfect Reader 見ると「あぁ ossan-arrow 氏にはかなわねぇなぁ」ってなる．

うーん，センスがほしい．それを実現する能力もほしい．結局これも練習あるのみ，なんかな．

どうでもいいんですが，北海道に行ったんですよ．

新千歳空港の中にアニメイトがあって，そこに売ってたアニメイト新千歳空港店限定「キタキツネの夕張メロンケーキ」のパッケージがかわいい．

行ったときは 12 月の末だったけど，めちゃめちゃ吹雪いてて寒かった．北海道だった．素敵だった．

雪っていいよね．なんか．

こう，しんしんと静かにふる雪は，寂しさを呈しているというか，それでいて優しさがあるというか．逆に猛吹雪は自然の威力を感じるし，そこに惹かれもする．

やっぱり雪っていうの日本人の感性にあってるんじゃないのかな．北の町で寂しく凍えるの，風流だよね．

そういえば山手のメニューにゆきあるね．ゆき．まだ食べてない．焦がしねぎうまい．今度誰か行きませんか．

なにもかもとけてしまえばいいんだよ．雪みたいに．

informatical reality —2013 winter—

2014 年 1 月 30 日　　初版発行
2014 年 3 月 10 日　　第 2 版発行

著者　イショティハドゥス

本プリントに関するお問い合わせは，Twitter アカウント @Ishotihadus までお願いします。

編者公式サイト：`https://sites.google.com/site/ishotihadus/`