2008 年度冬学期 情報科学 共通問題 解答・解説

問題 1

(a)

```
1 def reverse(x, p, q)
2   for i in p..(p+q)/2
3     t = x[i]
4     x[i] = x[q-(i-p)]
5     x[q-(i-p)] = t
6   end
7 end
```

【解説】

添字に注意．p+q が偶・奇のどちらでも動作することに注意．

t=a a=b b=t は a と b をスワップするときの定石．ただし，Ruby では a,b = b,a のような書き方もできる．

(b)

```
1 def transpose(x, k)
2   k.times{ rotate(x, 0, x.length-1) }
3 end
```

計算量について，1 回の rotate(x, 0, n) は $O(n)$ となるから，$O(k) \times O(n) = O(kn)$ となる．

【解説】

times メソッドを用いることでも繰り返しが可能です．

1 回の代入を $O(1)$ とすると，rotate(x, 0, n) について，

$$O(1) + O(n) + O(1) = O(n+2) = O(n)$$

(c)

```
1 def transpose2(x, k)
2   reverse(x, 0, x.length-1)
3   reverse(x, 0, k-1)
4   reverse(x, k, x.length-1)
5 end
```

計算量について，1 回の reverse(x, p, q) は $O(3 \cdot (q-p)/2) = O(q-p)$ となるので，求めるものは

$$O(n) + O(k) + O(n-k) = O(n) \quad (\because 0 \leq k \leq n)$$

【解説】

定数項は無視します.

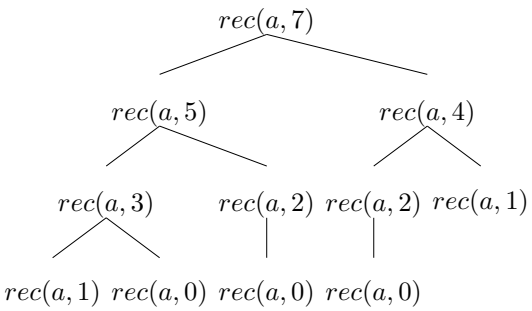
問題 2

(a) 出現確率が値によらず一定である数.
(b) 乱数のように見えるが，計算によって確定的に求めることができる数.
(c) 多次元積分，π の値を求める等の，決定論的問題の近似的解決，また，自然科学，社会科学において一般に定式が困難な，非決定的問題に対するシミュレーション.
(d) スライド参照.

問題 3

(a) 11 回

$a = [2, 3]$ とする.



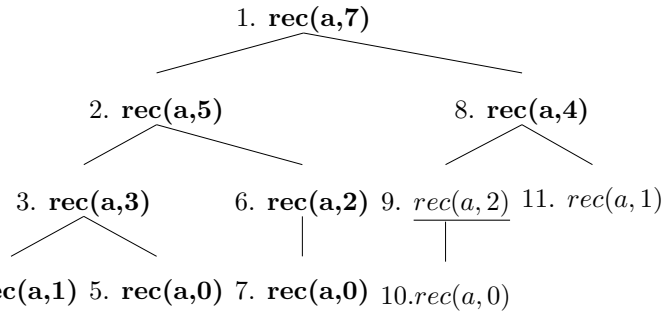
【解説】

recursive tree を描くと分かりやすい.

(b)

ある再帰呼び出しが行われている関数は，その関数の呼び出し元の関数の情報を知ることができないから.

【解説】



上の recursive tree（番号は呼び出される順番を示しています）で，例えば 9 の呼び出し（下線）は，それ以上に含まれる呼び出しおよびその呼び出しから呼び出される呼び出し（太字）で何が起こっているか知ることができません．よって，6 ですでに 9 は計算してあるにもかかわらず，重複して計算してしまうことになります.

(4)

```
1  def ite(a, n)
2    b = Array.new(n+1)
3    b[0] = 0 #(ア)
4    for m in 1..n
5      i = 0
6      k = -1
7      for i in 0..a.length()-1
8        if m >= a[i] && b[m-a[i]] >= 0 &&
9            (k < 0 || b[m-a[i]] + 1 < k)
10         k = b[m-a[i]] + 1
11       end
12     end
13     b[m] = k #(イ)
14   end
15   b[n] #(ウ)
16 end
```

【解説】

1 次元の DP(動的計画法)．$b[i]$ の定義は以下の通り．

$b[i]$:= 配列 a の中の各整数を 0 回以上使った和が非負整数 i に等しくできるときの整数を使う回数の最小値．存在しない場合は-1．$(0 \leq i \leq n)$

$b[i]$ さえ分かれば，再帰版のソースコードが与えられているので，類推から解くことができます．

(d)

以前に計算した内容を記録しておき，再度用いることができるから．

【解説】

DP の一般論とからめて，

「計算に部分最適解のみ用いるから」

でもよいかも．

問題 4

(a)

「得点の配列を整列するプログラムがすでにある」という条件があいまいなので，いくつか考えられる条件を付け加えて考えてみます．

- そのプログラムを書き換えることができない場合，すなわち，`sort_score(a)` のようなメソッドを用いる必要がある場合．

  n を得点の最大値として，それぞれの要素が整数を要素とする配列の，要素数 n の配列 b を用意する．要素を配列とした理由は，同じ点数の学生を考慮するためである．b には得点の配列の添字を挿入する．これによって，得点を整列

した後，配列 b を用いることで整列する前の得点の配列の添字を得ることができ，それによって氏名，学生番号を得ることができる．

- そのプログラムを書き換えることができる場合

  得点の配列に対して行う操作と同じ操作を，それに対応する氏名と学生番号の配列に対して適応すればよい．

【解説】

後者ははじめは，

「得点の配列の要素を交換する際に，同じく対応する氏名と学生番号の配列の要素も交換すればよい」

としようとしましたが，この方法はプリミティブな操作として交換を基とする整列法，すなわち最も原始的な操作が swap のみである整列法にしか適応できないので，抽象的に答えました．

前者が文章ではわかりにくいと思うので，ソースコードで示します．n が得点の最大値．p, q, r がそれぞれ得点，氏名，学生番号の配列．`sort_score(p)` が与えられているメソッドとします．

```
1  def sort_elms(n, p, q, r)
2    b = Array.new(n+1).map!{Array.new()}
3
4    for i in 0...p.length
5      b[p[i]].push(i)
6    end
7
8    sort_score(p);
9
10   for i in 0...p.length
11     for j in 0...b[p[i]].length
12       idx = b[p[i]][j];
13       q[idx], q[i] = q[i], q[idx]
14       r[idx], r[i] = r[i], r[idx]
15     end
16   end
17 end
```

補足：ここで用いた配列 b は direct-address table といい，hash table を導入するための基礎となります．

(b)

氏名（文字列），学生証番号（整数），点数（整数）

(c)

- **コンストラクタ**
  クラスのインスタンスが生成された際に呼び出される．

- **アクセサ**
  インスタンス変数を読み書きする.
- **（オペレータ）**
  学生クラスを演算に用いる場合に使用される演算子.

【解説】

整列をしようと思うならば，オペレータを定義するのが普通です．でも授業でやってないので，あまり深入りしません．Ruby ではソートの際にブロック（後述）を渡すことで，ソートの方法を指定することができます.

(d)

(手法 1)

配列クラスの sort メソッドを用います．このメソッドに，配列のソート規則を与えるためには，メソッドの**ブロック呼び出し**という手法を用います．|〜|ではブロック変数を指定します．使用できるブロック変数はメソッドによって異なります．sort メソッドの場合は，配列の要素を 2 つ参照します．さらに，その 2 つの要素について，整列を行う際の比較に用いる演算子を指定します．Ruby では <=> という特殊な演算子が用意されています.

```
1  class Student
2    attr_reader :id, :name, :score
3
4    def initialize(id, name, score)
5      @id = id
6      @name = name
7      @score = score
8    end
9  end
10
11 def sort_by_score(v)
12   v.sort{ |a, b|
13     a.score <=> b.score
14   }
15 end
```

(手法 2)

与えられたプログラムを利用して，比較に用いる変数を Student クラスの score に書き換えます.

【解説】

手法 2 について，例えば bubble_sort メソッドが与えられた場合を考えてみます.

```
1  def bubble_sort(a)
2    i = a.length-1
3    while i > 0
4      j = 0
5      while j < i
```

```
6       if a[j+1].score < a[j].score
7         a[j+1], a[j] = a[j], a[j+1]
8       end
9       j += 1
10    end
11    i -= 1
12  end
13 end
```

書き換えた部分は `a[j+1].score < a[j].score` だけです.