2008年度　情報科学　期末試験　略解

khRules

平成 22 年 2 月 9 日

1　問題 1

(a)

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

このようなプログラムにする場合、ループの回数は指定された範囲の長さの半分にしましょう。以下のようなプログラムはダメですよ（結果的に何も変化しません）。

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

あと、当然ですが、ここで言う変数 t のように、一時記憶用の場所もちゃんと用意してください。変数 t のようなものを用意せずに単に

```
x[p + i] = x[q - i]
```

としたのでは値が複製されるのであって、互いに交換されるのではありませんからね。

(b)

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

計算量は $O(n^2)$

「計算量」には、本当は最悪な事態を考える最大計算量と、実態に則して期待値を考える平均計算量の 2 種類があるのですが、この試験では算出の楽な最大計算量で考えてよいでしょう。というか、平均計算量は下手すると結論が出なくなる可能性があります（計算の得意な人はどうぞ）。

とりあえず、「計算量」が最大計算量であると仮定して、メソッド `transpose1` の内容だけで $O(n)$、更に、内部で呼び出しているメソッド `rotate` の内容で $O(n)$ ですから、合わせて計算量は $O(n^2)$ です。

(c)

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

計算量は $O(n)$

この問いはヒントがないとキツイですね（笑）。やはり数学が得意な人が有利かも（私は違いますよ）。

ヒントはメソッド `reverse` を 3 回使うと言っています。よくよく考えてみると、塊の間で反転するだけで、塊の中は反転しないので、この結果的に反転しない部分は反転を 2 回かけることで実現できる（「裏」の「裏」は「表」）というところが鍵でしょう。

「計算量」（最大計算量）は、メソッド `transpose2` の内容だけでは繰り返し（ループ）がないので $O(1)$、そして、、内部で呼び出しているメソッド `reverse` の内容で $O(n)$ ですから、合わせて計算量は $O(n)$ です。はい、めでたくメソッド `transpose1` よりも「優秀な」メソッドができました。

なお、計算量が小さいことは、具体的な処理がより短時間で終わることを必ずしも意味しません。単に対象となるデータ数が大きいほど有利になりやすいというだけですので注意しましょう（もっとも、アルゴリズムが優秀かどうかは、計算量だけではなくメンテナンス性など別の要素もありますけどね、それはソフトウェア工学の話）。

2 問題 2

(a)

> 値域の範囲内では全ての値が一定の確率で取り出されるような乱数。

ちなみに、「一定の確率」は、実際に値を取り出してみたら全ての種類の値が同じ回数だけ出てくることを意味しませんので注意してください。単に実際に値を取り出してみたら全ての種類の値が同じ回数だけ出てくることが「期待される」（そうなることが多いのだろうという予想・推測）、あるいは、既に出た結果から将来を全く予想できない、というだけのことですので、勘違いしないようにしましょう（と言うか、これは高校数学の話ですよね）。

(b)

あたかも乱数のように見えるように作られたもの。コンピューターは原則として特定の入力に基づいた出力しかできないので、特定の入力が変化しないのに出力が毎回変化することはありえない（出力はいつでも確定的である）。そこで、本物の乱数の特徴である無相関性などを備えることで、普通の使い方をすれば乱数のように見えるように工夫されている。

「疑似乱数」の基本的な意味は「乱数のようで乱数でないもの」ですが、これだけでは情報科学の範疇には入らない、あまりにも一般的過ぎる解答なのでダメです。きちんと情報理論またはコンピューター理論に則して解答しましょう。

コンピューター（computer）はその名の通り、計算をするものです。その計算とは四則演算（足し算、引き算、かけ算、割り算）とちょっとした論理演算（「〜かつ〜」、「〜または〜」とか）くらいなもので、あとはコンピューターは命令を実行する順番を変える機能を備えているだけの単純なものです。そのようなコンピューターが、（入力に依存せず）毎回出力を変えてくるような高等なことはしません。もし入力を変えていないのに出力が毎回変わるようであれば、そのコンピューターは故障しています！　さっさと修理に出しましょう（ただし、ハードウェア乱数生成器は除きます）。

というわけで、そのよ９うなちょっとした計算を組み合わせて乱数を作っているように見せかけるのです。そのよい例が線型合同法で、コンピューターがすることと言えば、メモリーに保存されている数に決まった数をかけて、決まった数を足して、最後に決まった数で割ったあまりを求めるだけです。でも、内部構造を知らなければ立派な乱数に見えます（ただし、線型合同法による乱数の場合、取り出す値の範囲を狭めるときには、上位の桁を取り出さないと乱数の周期が短くなってしまうことに注意）。

(c)

モンテカルロ法とは、シミュレーションや数値計算を乱数を使用して行う方法である。抽象的・一般的にかつ正確に問題の答を出そうとすると不合理なほど時間がかかってしまう場合に、不確定さは残るもののある程度の正確さで比較的短時間に問題を解きたいときに効果的である。

モンテカルロ法で真っ先に思いつくのは円の面積の算出ではないかと思います。円の面積は歴史的にはかなり大変な計算で、しばしば高度な数学の技術が要求されますが、モンテカルロ法を利用するのであれば、適当な正方形にでたらめに点を打ち、その中でその正方形に内接する円の中にどれだけ入っているのかを数えるだけですから極めて原始的で簡単です。

(d)

（メソッド `rand` が擬似的な一様乱数を生成するものだとしても、）点は単位円内に均一に分布しない。なぜなら、動径（r）方向で値が出る確率を一様にすると、動径（r）方向で同じ幅であってもその領域の面積は単位円の中心に近いほど小さくなるので、単位円の中心に点が密に集まることになるから。

このような問題を考える際には、おそらく確率としては高いであろう、「もし値が等間隔に、それぞれ同じ回数だけ取り出されたら」という場合を考えてみましょう。例えば、動径方向では $r =$

$0, \frac{1}{10}, \frac{2}{10}, \frac{3}{10}, \ldots$、偏角（$\theta$）方向では$\theta = 0, \frac{\pi}{16}, \frac{2\pi}{16}, \frac{3\pi}{16}, \ldots$のようにして点を打ってみると、点は放射状に分布しますが、単位円の中心に密に集まるのがよく分かると思います。

ところで、問題文に対してツッコミを入れたいのですが、Ruby のメソッド rand が（擬似的であっても）一様乱数を生成する保証はどこにもないはずです。なぜなら、規格書にそういう記述がないかもしれません。おそらく、たいていの実装では一様乱数になっていますので、実際にプログラムを実行してみると一様乱数が取り出されると思うのですが、ある日突然そうでなくなるかもしれませんので、規格にない場合は注意が必要です（特にソフトウェア工学に携わろうとする方はご注意ください）。

3 問題3

(a)

> rec([2, 3], 7) は rec([2, 3], 5) と rec([2, 3], 4) を呼び出し、rec([2, 3], 5) は rec([2, 3], 3) と rec([2, 3], 2) を呼び出し、rec([2, 3], 4) は rec([2, 3], 2) と rec([2, 3], 1) を呼び出し、rec([2, 3], 3) は rec([2, 3], 1) と rec([2, 3], 0) を呼び出し、rec([2, 3], 2) は rec([2, 3], 0) を呼び出すので、合計で 11 回。

実際に紙とペンを用いながらプログラムを実行してみてください。このときには、変数に格納されている値を逐次記録するとよいでしょう（ただし、再帰呼び出しなどで同じ名前のメソッドが呼び出されても、そのようなメソッドは互いに区別して、変数の値を個別に記しましょう）。また、再起と関わりのある問題では、樹形図を描いていくとよいでしょう。

ところで、この問題でメソッドが呼び出される回数を数えていくと、例えば rec([2, 3], 2)（rec([2, 3], 5) からの呼び出しと、rec([2, 3], 4) からの呼び出し）のように、メソッドの引数に全く同じ値が代入されてメソッドが呼び出されているのが何回かあります。これが次の問題への布石になります。

(b)

> メソッド rec の引数 n は再帰でメソッド rec が呼び出されるごとに、引数 a に格納されている値のうちいずれかを使用して減算されていく。ここで、単に数を減算する順番が異なるだけで、数を減算する回数が数の種類毎に同じならば、引数 n には同じ値が代入される。ここで、全く同じ動作が展開されるため、重複になる。

(c)

> （ア） 0 （イ） k （ウ） n

このようにプログラムを完成させる問題では、どうしても元となるプログラムを理解しなければなりません。プログラムを理解する方法は大雑把に 2 種類あり、一つはプログラムを実際に実行してみて、その様子を見るというもの、もう一つは、プログラム中における変数や処理構造の組み合わせを手がかりとするというのがあります。後者について述べると、例えば、

```
for i in 0..a.length()-1
```

で使用されている変数 i は、0 から配列 a の要素数から 1 を引いたものまで 1 ずつあがっていきますが、これは a の要素数を使用していることからも分かるように、a の添字（配列 a から何番目の値を取り出すか）を表しています。

(d)

繰り返しで使用している変数の値は単調に増加しており、一度変化したら二度と戻ってこないので、変数の値の組合せが全く同じになることがないから。

4 問題 4

(a)

まず得点の配列を別の場所に複製した上で、得点を順番に整列させ、あとは保存しておいた得点とそれに対応する学生の氏名と学生証番号を取り出し、その得点が整列後の得点の配列でどの場所に位置づけられているのか（ただし、まだ学生の氏名と学生証番号が書き込まれていないものに限る）を探し、そこに学生の氏名と学生証番号を書き込む。

(b)

学生の性質を表す、氏名、学生証番号、得点。

(c)

比較の対象となる得点を取得するメソッド。

(d)

学生を表すオブジェクトに記された得点を基に、学生を表すオブジェクト全体を移動させる。

具体的にどのようなプログラムに仕立て上げるのかで答えが異なるかもしれません。問題がちょっと抽象的過ぎるのではないかと思います。

(e)

オブジェクト全体を移動させるようにすれば、学生の氏名や学生証番号は比較の対象となる得点と一緒に動いてくれるので、氏名、学生証番号や得点を別々の場所に記録する場合のように、得点を整列させた後で、氏名や学生証番号といった要素を個別に得点と対応を付け直す必要がない。

解説を作る時間がなくなってしまったのでとりあえずこの辺で。