

計算問題を巡って

SHIMURA Masato
jcd02773@nifty.com

2017年12月8日

目次

1	量子コンピューターと計算問題の昨今	1
2	古くて新しいアルゴリズム	2
3	ハノイの塔のアルゴリズム	2
4	組み合わせと計算問題	5
5	魔方陣	6
6	ナップザック問題	9
7	経過と説明	11

アランチューリングが展開した証明や計算問題は永くコンピューター科学者を悩ましてきた古くて新しい問題である。量子コンピューターが手に取れるようになった今、新しい光明が見えるだろうか

1 量子コンピューターと計算問題の昨今

1.1 量子コンピューター

- IBM ワトソン研究所の量子コンピューターがクラウド経由で公開されており、誰でもが iPhone から使える。
5 ビットの量子演算の仕組みは次が詳しい。
<https://www.ibm.com/developerworks/jp/cloud/library/cl-quantum-computing/index.html>
更に 20 ビット拡張がなされ、スパコンが視野に入り始めたようで Python でも動き、更には専用言語の開発が進んでいる。
- NTT、東京大学、国立情報学研究所が共同開発した最適化の研究を目指した量子コンピューターネットワーク・クラウドも公開体験が計画されている。これまでの 30 倍、2000 ビット量子ネットワークのう

たい文句は素晴らしいのか否か想像が難しい。

- IBM は IBM Cloud の機能の一部を期間制限をつけずに無料公開した。ただし、使い続けなければアカウントは削除されるようだが、「作るな！使え」のようだ。
- Google から近々量子マシンが公開されるようだ。

1.2 スーパーコンピューター

- スパコンは性能ランキングが時にニュースになる。
- 最近のスパコンは (1)Intel Zeon,(2)Fujitsu/SGI/Cray Spark ,(3) IBM Power Core のどれかで動いている。
IBM のセコイアは少し古い情報ではパワーPC系のCPUコアが1600万個とか。
- 日本が最後にトップランクになったのは2011年で、ここ5年は中国が独占している。
- スパコンもLinuxで動いていればJ806が使えるようだ。

2 古くて新しいアルゴリズム

アルゴリズムは中世アラブの大数学者アル・フワーリズムに由来し、「アル・フワーリズムかく語りき！」と言うように解法の手順が示される。

K.E.Iverson の著書は *J* のプログラムで正解手順を示し、*Proof* としている。

20世紀の数学で計算手順には何の役にも立たない「反証法」の証明が有力視されている。

計算問題/コンピューターでググると計算科学が音を上げている古典的問題が多く見られる。

- チューリングマシンの停止問題
停止しなければ解がわからない、予測不能問題
- 組み合わせ問題
何処までも計算を続ける一無限か、弥勒菩薩出現まで計算が続く

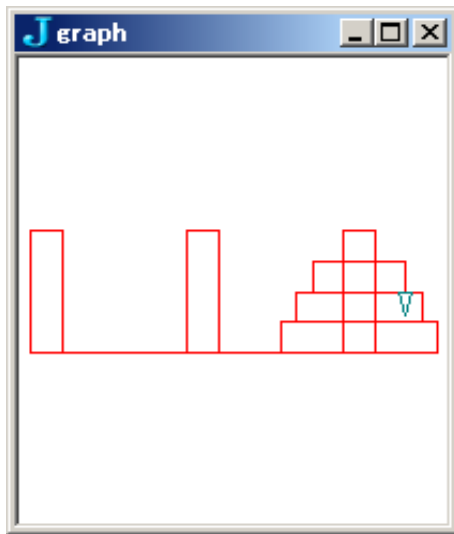
ここでは計算問題の基礎部分をアルゴリズム即ち「アル・フワーリズムかく語りき」に従い、以前に作成したスクリプトを動かしながら見てみよう。

3 ハノイの塔のアルゴリズム

ハノイの塔 バラナシの寺院で64枚の円盤と3本の支柱を使って、終末までの時を図る。

個数	プロセス	回数
2	1 2 1	3
3	1 2 1 3 1 2 1	7
4	1 2 1 3 1 2 1 4 1 2 1 3 1 2 1	15
5	1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1	31

円盤の個数を n とすると回数は $2^n - 1$ となる。円盤2個の軌跡を3個が前後に利用し、3個の軌跡を4個が前後に利用すると簡潔に計算できる。このようなプロセスを効率的に利用するアルゴリズムを再帰という。



```

2^64
1.84467e19
x: 2^64
18446744073709551616

```

64 個の円盤で終末が計れるか? 2^{64} は京の桁で 16 乗であるので 1844 京となる。かのスパコンでも手強い。1 秒に 1 回円盤を動かすとすれば 5800 億年! インドの無限も奥が深い

Roger HUI *J* の WIKI/Essay に R.Hui のハノイの塔の解法と詳細で難解な解説が入っている。次はその冒頭部分

```

H=: 4 : 0
if. 1>y do.
  (y,2)$x
else.
  ((0 2 1{x) H y-1), (2{x), ((2 1 0{x) H y-1)
end.
)

```

```

                                0 1 2 H 4
                                0 2
                                0 1
                                2 1
                                0 2
                                1 0
                                1 2
                                0 2
                                0 1 NB. center
                                2 1
                                2 0
                                1 0
                                2 1
                                0 2
                                0 1
                                2 1
                                NB. 15 step

    0 1 2 H 3
    0 1 NB. 0peg -->1peg
    0 2
    1 2
    0 1
    2 0
    2 1
    0 1
    NB. 0 1 2 はペグの位置
    NB. 3 ディスクの数
    NB. 7 step

```

経過 .

```

• if
  (4 ,2) $ 0 1 2
  0 1
  2 0
  1 2
  0 1
• else
  ((0 2 1 { 0 1 2) H 3);(2{. 0 1 2) ;(2 1 0{ 0 1 2) H 3
  +---+---+---+
  |0 2|0 1|2 1|
  |0 1|  |2 0|
  |2 1|  |1 0|
  |0 2|  |2 1|
  |1 0|  |0 2|
  |1 2|  |0 1|
  |0 2|  |2 1|
  +---+---+---+

```

4 組み合わせと計算問題

4.1 組み合わせ

- 組み合わせを作る *J* 言語のイディオム。

```
tap=: i.@! A. i.
```

NB. table of permutations /組み合わせのテーブル

A. *Anagram* 辞書式順序 (組み合わせを順にすべて表示する)

```
(i. ! 3) A. i.3      NB. (i.6) A. i.3 (=tap)
```

```
0 1 2
```

```
0 2 1
```

```
1 0 2
```

```
1 2 0
```

```
2 0 1
```

```
2 1 0
```

- 3 の場合 (0,1,2) の組み合わせは 6 とおり

```
tap 3
```

```
0 1 2
```

```
0 2 1
```

```
1 0 2
```

```
1 2 0
```

```
2 0 1
```

```
2 1 0
```

- 9 の場合は 36 万をこえる。

```
# tap 9
```

```
362880
```

```
! 9      NB. 9*8*7*6*5*4*3*2*1
```

```
362880
```

4.2 組み合わせ論を少し

離散数学の組み合わせ論はインドの数学者バースカラ (1114-1185 頃) に始まり、南フランスのゲルソニデス (1288-1344) など様々な数学者が取り組み、ヤーコプ・ベルヌイ (1654-1705) の名著「推測の技術」でまとめられた。

1. !の単項 階乗/Factorial

$$!5 \longleftrightarrow 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$$

2. !の両項 組合せ/Combination

$${}^3!5 \longleftrightarrow {}_5 C_3 = \frac{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{(3 \cdot 2 \cdot 1)(2 \cdot 1)} = 10$$

$${}_n C_r = \binom{n}{r} = \frac{n!}{r!(n-r)!}$$

3. 順列を用いると

$${}_n P_r = {}_n C_r \cdot !r \longleftrightarrow {}_n C_r = \frac{{}_n P_r}{!r}$$

$$({}^3!5) \cdot !3 \longleftrightarrow 5 \cdot 4 \cdot 3 = \frac{5 \cdot 4 \cdot 3}{3 \cdot 2} * 3 \cdot 2 = 60$$

5 魔方陣

5.1 3×3の魔方陣

1 から 9 までの数字で魔方陣を作る。1 から 9 までの数字を 3×3 のマトリクスに組む方法は 362,880 あった。魔方陣は縦、横、斜めが全て同じ数になる。まずはプログラムの力技で行こう。

```

fm0 ''
-----
| 2 7 6 | 2 9 4 | 4 3 8 | 4 9 2 | 6 1 8 | 6 7 2 | 8 1 6 | 8 3 4 |
| 9 5 1 | 7 5 3 | 9 5 1 | 3 5 7 | 7 5 3 | 1 5 9 | 3 5 7 | 1 5 9 |
| 4 3 8 | 6 1 8 | 2 7 6 | 8 1 6 | 2 9 4 | 8 3 4 | 4 9 2 | 6 7 2 |
-----
(0)      (1)      (2)      (3)      (4)      (5)      (6)      (7)

```

5.2 3×3の魔方陣のスク립ト

```

fm=: find_mahoujin=:3 : 0
NB. find Mahoujin 3 X 3
NB. Usage: fm ''
nr=. # tmp0=. >: tap 9          NB. 全組合せを求め1を加え1-9の整数に
ans=. < 0
for_ctr. i. nr do.             NB. カウンター付きで32万回のループ
  tmp2=. (+;/+/"1) 3 3 $ ctr{tmp0 NB. 縦横各行の合計
  tmp4=. +/ L:0 (0 4 8;2 4 6){L:0 ctr{tmp0 NB. 斜め方向を取り出し合計
  if. 1 = # ~. ; tmp2,tmp4 do.  NB. ~. nub 8個が全部同じ数なら魔方陣
    ans=. ans,<ctr{tmp0        NB. 魔方陣の書き出し
  end.
end.
3 3 $ L:0 }. ans              NB. 3x3に整形
)

```

5.3 高次の魔方陣

魔方陣は1から n^2 の数を用いて、行、列、対角の和は次になることが知られている。^{*1}

$$\frac{1}{n} \sum_{i=1}^{n^2} i = \frac{n(n^2+1)}{2}$$

*tap 16, tap 25, tap 36*などのA.を用いる*tap*はメモリが厳しいので、逐次打出しのアナグラムを自作しなければならない。

デューラー(ドイツ)の1514年の銅版画に4×4の魔方陣が描かれている。

Wikipediaによると4×4の魔方陣は880、5×5は2億3750万個存在することが知られている。(以降の個数は記されていない。)

Wikipediaにはさらりと書いてあるが、力技法では4×4の魔方陣は $16=21$ 兆回のループを要する。

*1 他のタイプの魔方陣もある

```

!16
2.09228e13
!25
1.55112e25      NB. 15 Jyo

```

4×4の最初の方の魔方陣でもループ809億回。全て求められた方々に敬意を表する。

```

A. 1 2 15 16 13 14 3 4 7 12 10 5 8 11 6 9
80907739690

```

1	2	15	16
13	14	3	4
12	7	10	5
8	11	6	9

関孝和、オイラー、ラマヌジャン、コンウェイも数論として魔方陣を研究している。

5.4 高次の魔方陣のスク립ト

- メモリーが厳しいので、ループで一個ずつ 魔方陣か否か判定して、魔方陣ならば書き出す。
- `for_ctr. i. 1000000000 do.` とするとメモリーオーバーになる。*i.*の全ての数をメモリーに持つため。`while.`ループに変更した。(メモリーはあまり使わない)

```

fm0=: find_mahoujin_any =: 4 : 0
NB. find Mahoujin 3X3 4X4 5X5 ...
NB. Usage: (0, 10000) fm0 3 3
NB. Usage:( 80907762000,80907762100) fm0 4 4
NB. x is pickup zone of permutiation
NB. i.e. 3 8 is 3<->8
NB. ! 9 16 25 is 362880 2.09228e13 1.55112e25
NB. y is i.e 3 3 // 3 X 3
NB. -----
'nrmin nrmax'=. x      NB. input zone //oligin is 0
zone=. (<: nrmax) - nrmin
if. 2 = # y do. size0=. y
  else. size0 =. 2 # y
end.
size=. */ size0      NB. i.e. 9 <-- 3 X 3
NB. -----
mat=. i. size0      NB. explore index matrix
oblic_ind=. index_oblique mat      NB. find oblic index
ans=. < 0
ctr=. nrmin

```



```

NB. -----
while. ctr < nrmax do.
  mmat=. >: ctr A. i. size          NB. target permutation 1++
  tateyoko=. (+;/+/"1) size0 $ mmat  NB. make mat and sum tate&yoko
  naname=. +/ L:0 oblic_ind { L:0 mmat  NB. sum each oblic X
  if. 1 = +/ ~: ; tateyoko,naname do.  NB. NB. 4 pieces is same --> Mahoujin
    ans=. ans,<mmat
  end.
ctr=. >: ctr
end.
NB. -----
if. 1 = 1 < # ans do. ans=. size0 $ L:0 }.ans
else. ans=. 'nothing'              NB. null
end.
)

index_oblique=: 3 : 0
NB. sub find oblic index
NB. usage: index_oblic mat
NB. y is mat
ob0=. ((<:# y) { </. y)
ob1=. ((<:# y) { </. |. y)
oblique=. ob0,ob1      NB. pick index both / & \
)

```

6 ナップザック問題

- ナップザック問題も古くて新しい問題である。容れるものを一種類一個とすると *NP* 完全のクラスになる。
- 解法は動的計画法やツリー探索、双方のハイブリッド法がある。
- 組合わせ問題は次元が少し増えるだけで急激に計算量が増えるのでマトリクス計算は個人の *PC* では直ぐにオーバーフローする。
- 何れにしても計算して後の書き換えか結果を全部持つかの問題であるならば大差はない。(計算する前に不要ブロックを除外できるか → *AI* の領域)

例題：高級縫いぐるみ工場に押し入って、人気の熊さん人形を物色している。

- 熊さん人形は4種類あって次の表のとおり。人形は沢山ある。

<i>SS</i>	2kg	16万円
<i>S</i>	3kg	19万円
<i>M</i>	4kg	23万円
<i>L</i>	5kg	28万円

- 逃走用のナップザックは古く、7kgより多くの荷物を入れると底が抜けてしまう。
- 転売価格が最大になるにはどのような熊さん人形を持って逃げればよいか

出題 久保 p124

6.1 ナップザック問題とアルゴリズム

入力 次のように縦型とする

DATA

2 16
3 19
4 23
5 28

ソート スクリプトでは入力に昇順ソート (/ : ~) をかけて、出力もソート済みである。結果を読むときに、入力もソートしておくとも便利である。

sort=: /:~

sort DATA

最大個数 $n = \frac{\text{制限重量}}{\text{最軽ピースの重量}}$ の切り捨て (*floor* <.) 値

組み合わせの方法 数学の順列や組み合わせは個数を求める。ここでは内訳が必要。

$${}_n H_r = \frac{(n+r-1)!}{(n-1)!r!} = {}_{n+r-1} C_r$$

4人の子供 (*n*) に5個のみかん (*r*) を重複を許して分ける方法

$${}_4 H_5 = \frac{8!}{5!3!} = {}_8 C_5 = 56$$

組み合わせの内訳を求めるのに *J* のイディオム `tap=:i.@!` **A.** *i.* があり、順列 ${}_n P_n$ とした全個数を打ち出す。ここでは重複が入り、重複具合が分からないので、愚直に全組み合わせを打ち出す

アルゴリズム ナップザック問題のアルゴリズム

ここでは単純な力技法を用い次のアルゴリズムによる。

1. **n** を元に全組み合わせを作成する

2. 途中に重量チェックを組み入れる (枝切、分岐限定プロセス)
3. 価格による最適評価は重量制限下で可能な全組み合わせ求めたうえで一度に一覧を打ち出す
4. 整数でなくともよい

6.2 例題の実行

0 は何も取らない。2 3 4 5kg を指標 1 オリジンとして順に取った

```

                                7 napsack DATA
                                指標の組み合わせ重量とその価格を付加
組み合わせの指標
                                0 0 0 0 0
                                1 0 0 2 16
                                2 0 0 3 19
                                3 0 0 4 23
                                4 0 0 5 28
                                1 1 0 4 32
                                2 1 0 5 35
                                2 2 0 6 38
                                3 1 0 6 39
                                3 2 0 7 42
                                4 1 0 7 44
                                1 1 1 6 48
                                2 1 1 7 51
                                index kg/price
7 napsack0 DATA
0 0 0
1 0 0
1 1 0
1 1 1
2 0 0
2 1 0
2 1 1
2 2 0
3 0 0
3 1 0
3 2 0
4 0 0
4 1 0

```

最適化の順にソートしている

久保は最大値 51 を求めるのにツリー構造や *Bellman* の動的計画法を用いて、価格まで同時に探索して最大価格を求めている。マシンパワーを生かした力技法の方がブラックボックスがないので明快である

7 経過と説明

7.1 組み合わせを求める

```

napsack0=: 4 : 0
NB. Usage: 7 napsack0 DATA
limit=. x
' weight price'=. { |: /:~ y NB. upsort
maxnum=. {. <. limit % weight NB.floor
ans=: (maxnum # 0) , >({@> >:i.maxnum) # (L:0) 1

```

```

for_ctr. i. <: # y do.
  ind0=. }. i. maxnum
  left=. ({@> ind0) # (L:0) 2+ ctr
  index=. maxnum - L:0 {@> ind0
  right=. index {"1 (L:0) ans
  left=>L:1((# ans) # L:1 { L:0 left)      NB. fine
  tmp0=(,./,./>,.left ,. L:0 right), maxnum # 2+ctr
  ans=. ~. (limit;weight) check_weight ans, tmp0
      NB. nub

end.
)

```

- 軽い順にソートし、*weight price* に代入
'*weight price*'=. { :/: y—NB. upsort
- 最大個数 (*n*) を求める
maxnum=. {. <. limit % weight NB.floor
- *n* と *l* 番目の指標の組み合わせ。1 まで処理完了
ans=: (maxnum # 0) , >({@> >:i.maxnum) # (L:0) 1

```

ans
0 0 0
1 0 0
1 1 0
1 1 1

```

- 2 番目の組み合わせ

```

({@> }. i.3) # (L:0) 2
+-+----+
|2|2 2|
+-+----+

```

これを *ans* の行数分コピーして、右に足りない列分の *ans* を取り出し拡張する

```

tmp0
2 0 0
2 1 0
2 1 1
2 1 1
-----
2 2 0

```

```

2 2 1
2 2 1
2 2 1
-----
2 2 2

```

最後に 2222 を加える

- ここで次項のリミットチェックを行う。数値はインデックス
- 枝切である

7 napsack0 DATA

```

0 0 0
1 0 0
1 1 0
1 1 1
2 0 0
2 1 0
2 1 1
2 2 0

```

- 3項目.. と繰り返す
- $L:1$ は2重ボックスの中での演算

7.2 重量チェック

```

check_weight=: 4 : 0
'limit weight'=. x
ind=.limit>: +/"1 y { 0, weight
ind# y
)

```

- `0,weight` で0番目の0を入れる
- 重量内かオーバー化の 10 の指標を作成し、指標に従って 1 の部分をコピーして取り出す

7.3 最適化

ブラックボックスで最適解を求めるより、一覧表の方がよくわかる。

```

napsack=: 4 : 0
NB. Usage: 7 napsack DATA
limit=. x
' weight price'=.{. { |: /:~ y NB. upsort

```

```
ind=. limit napsack0 y
ind,. |: > +/"1 L:0 ind{ L:0 {|: 0,y
)
```

- 取り出した指標ごとの価格を計算し、一覧表を作る。

```
7 napsack DATA
指標の組み合わせ重量とその価格を付加
```

```
7 napsack DATA
0 0 0 0 0
1 0 0 2 16
2 0 0 3 19
3 0 0 4 23
4 0 0 5 28
1 1 0 4 32
2 1 0 5 35
2 2 0 6 38
3 1 0 6 39
3 2 0 7 42
4 1 0 7 44
1 1 1 6 48
2 1 1 7 51
index kg/price
```

- 出力が膨大な時は最適解に近い部分が直ぐにみられるよう列ソートを入れた
`sort_fit=: 3 : '|. "1 /:~ |."1 y'`

7.4 幾つかの例題

ネットでいくつかの例題を集め、数え上げ法と比較してみよう。いずれも動的計画法を駆使した名作である。同時に動的計画法の的確さも実感することができる。

1. 例題2 「病みつきになる動的計画法 その深淵に迫る」ITメディア *Enterprize*

この例題は1品ずつという制限が入っている。`napsack_one`には次の1行を加えた
`order=: 3 : '|. /:~ "1 ~. "1 y'`

```

_5 { . 10 napsack_one DATA3
/:~ DATA3
0 0 1 4 5 8 11
1 2
0 0 3 4 5 10 11
2 3
0 0 2 4 5 9 12
3 2
0 1 2 3 4 9 13
3 6
0 1 2 4 5 10 14
4 3
kg/price
limit 10kg

```

同じ重量なら最適化で価格が安い方が選ばれることはない。

2. 例題3 チャリティーオークションへの出品。自分で運ぶので重さは 10kg (片鱗懐古のブログ)

	kg	yen	DATA2	/:~ DATA2
カメラ三脚	3	7000	3 7000	1 1 3000
手提げ金庫	2	4000	2 4000	2 1 5000
ゲーム機	2	8000	2 8000	3 2 4000
高圧洗浄機	5	9000	5 9000	4 2 8000
木彫り置物	1	3000	1 3000	5 3 7000
電気ポット	1	5000	1 5000	6 5 9000

こちらを参照

```

_5 { . 10 napsack_one DATA2
0 0 1 2 5 6 10 24000
0 0 0 4 5 6 10 24000
0 0 1 2 4 6 9 25000
0 0 2 3 4 6 10 26000
0 1 2 3 4 5 9 27000

```

高圧洗浄機は置いて行こう

References

- 久保幹雄「組み合わせ最適化とアルゴリズム」 共立出版 2000
 スクリプトは次から DL できます。 japla.sakura.ne.jp
 J 言語は次から入手できます。 www.jsoftware.com