条件付き整数分割　PARTITION　計算（第７報）

Biased Partitions (Rep. No. 7)

中野嘉弘　（札幌市南区・８５才）

FAX 専　011-588-3354　　yoshihiro@river.ocn.ne.jp

2008年のノーベル物理学賞の快挙の発端の原論文は、 PARTITION 問題に始まる。
今回は、巨大整数分割の総数を求める類の巨大計算から転針して、きめ細かい問題に移ろう。
有名な数学ソフト　Mathematica では、既に、この方面が優先している。
ライバルのソフト Maple V でも、巨大数の処理では最近、連続して ミス が報告されているが、同じく、きめ細かい多くの計算法が内蔵されている。　我らも負けまじ！

０．　は　し　が　き

今年のノーベル物理学賞はまたもや、素粒子物理学 PARTICLE PHYSICS であった。
これは別名、 PARTON パートン（部分子）物理学 であり、1969年に、かのファインマンによって提唱されたものだった。　発端となった小林・益川論文（文献０）を FAX して上げたところ、西川会長・山下老会員から、「こんなところに　PARTITION ！」と歓声が帰って来た（文献　０－ａ，ｂ）。
　　その論文の　２ページ目に　３ケの式　がある。
　　A) 4 = 2 + 2,　B) 4 = 2 + 1 + 1,　C)　4 = 1 + 1 + 1 + 1
整数 ４ の 分割（５通り）中の「３つ」である。
この内、４通りの組み合わせ　(A,C), (A,B), (B,B), (A,A) が肝要なのだそうだ。
原論文はたった６ページ、これでノーベル賞！　しかし、判り易い良い論文だ。

物理の話はさておき、JAPLA では、今年初頭から、PARTITION の話題が、４月の 鈴木義一郎先生の報告「４個のみかん：数の分割」やら西川会長、志村氏らの諸報告が衆目を集めた。（文献１）
それに対して、私の多くの報告は「演算結果は果たして正解か？」（文献２）であって行き着く先は、与数が １４万から２２万 台の巨大計算になり、時間との競争とはなった。（文献３）
結局は、英国コンピュータ学会の VECTOR 誌編集長様から助言で、J 言語 Essay/ Wiki の情報（文献３－ａ）で、こんな単純作業は打ち止め、条件付きの、もっとエレガントな問題を目指す事にした。　本稿はその手始めの報告のつもりである。
もっとも、我が JAPLA の西川会長はすでに今年初頭「数の分割とフェラーズの図」と題して、先行する報告を提出しておられる。　（文献４）
また、志村氏の「分割数の積み上げ方式」（文献４－ａ）は、今回の中野の試みを先取りした感じである。

とにかく、いろいろな興味深い論が、既に出ているのだ。

１．　Mathematica Ver. 2.2 のレベルでは

Mathematica ハンドブックなる有用な参考書が翻訳されている（文献３）。
それによって、色々な条件付きの分割数を計算出来、次の３種類が紹介されている。

１）　PartitionP （無制限分割数＝我らが普通に云う分割数に相当する）
２）　PartitionQ （特異分割数）
３ａ）　Partition （例えば、 長さ 2 とかに限定した時の分割数）
３ｂ）　Partition （例えば 長さ 2 のオフセット付きで、要素数 3 の分割数）

先頭の １） が、我々が今までやって来た普通の分割である。
Mathematica による例を示そう。
例１：　PartitionsP[9999]//Timing　の結果は
{623. Second, 3570990187904736738.....12668644192313000}
これが正解かどうか？は、どうぞ　J 言語にてトライされたい。

２）特異分割とは、分割結果の整数がすべて異なるような分割である。
例えば、整数３の分割法は３通り、即ち　3, 2 1, 1 1 1　であるが、最後の
1 1 1 には、等しい数 1 が含まれるので、除外するもとする。
従って、３の特異分割の個数は　２　となる。

例２：　PartitiosP[61] （我々が普通に計算のもの）の結果は　1121505。
PartitiosQ[61] （要素に等しいものは不可）の結果は　12076。

３） 例えば、リスト　nums = {-8,9,16,-5,-7,-9,-6,12,8,14} ；を与えて
例３ａ：　Partition[nums, 2]　の結果は
{ {-8, 9} , { {16, -5} } , { { -7, -9} } , { { -6, 12} } , { { 8, 14} }
例３ｂ：　Partition[nums, 3, 2] の結果は
{ {-8, 9, 16} , {16, -5, -7} , {-7, -9, -6} , { -6, 12, 8 } }
要素数は 3 で、　オフセットは　2　である。

３．　Mathematica の　PARTITIONP と　－ Q　を　Ｊ言語　で

以上前節の　例２　と同じ事を　Ｊ言語でトライしよう。

実は、Mathematica で容易に計算出来たものが、Ｊ言語では、与データが　５８
以上では OUT OF MEMORY エラーで不可能であった。
VECTOR誌御推薦の　jwiki/Essays/Partitions　（文献２－ａ）内の関数 PART を用い

● PART 3　からは 分割リスト　[3|2 1|1 1 1] が得られ、分割数は　３　である。
同様な事が可能な上限は
ts 'wr # prt58 =. part 58'　で、結果は次の如し
分割数　715220　通り、　これは　Mathematica とも一致し、正しい。
時間　4.50201 sec、　メモリー　1.30327e9 。
● 次に、分割リストの各BOX内の要素が全て異なる(non dw)の場合を選ぶ操作をする。
ndwp58 =. ndwt each prt58　ここに関数　ndwt =: 3 : ‘ 0= (# y) – (# ~. y)’
ts 'wr # (>ndwp58) # prt58'　から、結果は

分割数　8808 通り、　これは　Mathematica とも一致し、正しい。
時間　0.0655075 sec、　メモリー　1.11558e6 。

４．　Mathematica　の　PARTITION と同じ事を　Ｊ言語　で

以上前節の　例３　と同じ事を　Ｊ言語でトライしよう。

予め、リスト例　nums = ｛-8,9,16,-5,-7,-9,-6,12,8,14｝；の類を与えて置く。
Ｊ言語ならば、　　t1 =. _8 9 16 _5 _7 _9 _6 12 8 14　の類。
ここで判るように、PARTITION 問題は　正整数に限らず、従って文字型でも問題になり得るのである。

例３ａ）要素数サイズ指定での　PARTITION

```
st1=. 5 2 $ t1  ->   _8  9
                    16 _5
                    _7 _9
                    _6 12
                     8 14

ts0 =. (0,0{"1 st1), (1{"1 st1),0
# ts0  ->  12
ts06 =. 2 6 $ ts0
}. (}: </. ts06)
```

から結果は
[_8 9|16 _5|_7 _9|_6 12|8 14]　となり、Mathematica での結果と一致する。

例３ｂ）　要素サイズは３、　オフセットは２の場合

```
4 3 $ t1  ->  _8  9 16
              _5 _7 _9
              _6 12  8
              14 _8  9

t41 =.(0{4 3 $ t1), (1{4 3 $ 0,t1),(2{4 3 $ 0,0,t1),3{4 3 $ 0,0,0,t1

4 3 $ t41  ->  _8  9 16
               16 _5 _7
               _7 _9 _6
               _6 12  8
```

こえから、前例同様にして、期待の分割が得られる。
その為の中野プログラム　condpart　での例を示す。

```
  nums
_8 9 16 _5 _7 _9 _6 12 8 14

  (3 1) condpart nums
┌──────┬──────┬──────┬──────┬──────┬──────┬──┐
│_8 9 16│9 16 _5│16 _5 _7│_5 _7 _9│_7 _9 _6│_9 _6 12│_6 12 8│12 8 14│  │
└──────┴──────┴──────┴──────┴──────┴──────┴──┘

  (3 2) condpart nums
```

```
┌──────────────┬──────────────────┐
│ _8 9 16 | 16 _5 _7 | _7 _9 _6 | _6 12 8 |
└──────────────┴──────────────────┘
```

(3 3) condpart nums

```
┌──────────────┬──────────┐
│ _8 9 16 | _5 _7 _9 | _6 12 8 |
└──────────────┴──────────┘
```

(3 4) condpart nums

```
┌──────────────┬─┐
│ _8 9 16 | _7 _9 _6 |
└──────────────┴─┘
```

(3 5) condpart nums

```
┌──────────────┬──┐
│ _8 9 16 | _9 _6 12 |
└──────────────┴──┘
```

(3 6) condpart nums

```
┌────────┐
|_8 9 16|
└────────┘
```

５．　カナダ産の数学ソフト　Maple V ：

西川会長は、先に、これに拘った事があった。（文献　１－ｂ）
package　の　with (combinat); で　numnpart(294);　の結果は、正解を与える。
ただし、対象の数が大きいと、演算で　**STACK OVERFLOW** エラーを発する
対策としては、**numbpart(n)** の 引数 **n** を、小さな数　**50** 程度から出発して
50 刻み程度で数を増加するなどの工夫をすれば、この種の **STACK OVERFLOW** エラーは突破出来る。　中野は前報（文献２）では　**n = 1550**、　分割数 **50** 桁　を報じたが、今回鈴木報告に刺激されて、一応　**n = 3601**、　分割数　**63**　桁まで調べた。

しかし、この **numpart** 以外にも多くの **PARTITION**　の　**functions** が用意されている。
例えば：
partition, firstpart, nextpart, prevpart, lastpart,

conjpart,　encodepart, decodepart, randpart, inttovec　等々　（文献６）

仲々　器用な方法であるから若干は例題にて解説しなければなるまい。

1) numpart(4); -> 5, numpart(10); -> 42 の類で、分割の総数を与える。
我々が苦労している対象で、 **Mathematica** の **PartitionsP[4]**相当。

2) partition(4) -> [[1,1,1,1],[1,1,2],[2,2],[1,3],[4]] の　分割リスト

3) firstpart(4) -> [1,1,1,1] の類、分割リストの最初

4) nextpart(") -> [1,1,2] 上記の分割リストの次を与える。

5) lastpart(4) -> [4] 分割リストの最後

6) prevpart(") -> [1,3] 上記の分割リストの直前を返す。

7) conjpart(") -> [1,1,2] 上記の分割リストに **conjugate**（共役）なリスト。

8) decodepart(4,1) -> [1,1,1,1] 整数 ４ の分割リストの１番目を与える。

decodepart(4,2) -> [1,1,2]　整数 ４ の分割リストの２番目を与える。
decodepart(4,3) -> [2,2]　整数 ４ の分割リストの３番目を与える。
decodepart(4,4) -> [1,3]　整数 ４ の分割リストの４番目を与える。
decodepart(4,5) -> [4]　整数 ４ の分割リストの５番目を与える。

9) encodepart([1,1,1,1]) -> 1　和が４なる分割リストに１番目である。
encodepart([1,1,2]) -> 2　和が４なる分割リストに２番目である。
encodepart([2,2]) -> 3　和が４なる分割リストに３番目である。
encodepart([1,3]) -> 4　和が４なる分割リストに４番目である。
encodepart([4]) -> 5　和が４なる分割リストに５番目である。

10) randpart(4) -> [1,3]　和が４なる分割リストを乱数的に求めた。
〃　**-> [1,1,1,1]**　〃
randpart(10) -> [2,3,5]　和が１０なる分割リストを乱数的に求めた。
〃　**-> [1,1,8]**　〃
〃　**-> [2,8]**　〃

ベクトルのリスト処理　（下例は　３次元）

vector成分	番号	記号（Monomial）
[0,0,0]	0	1
[1,0,0]	1	x
[0,1,0]	2	y
[0,0,1]	3	z
[2,0,0]	4	x^2
[1,1,0]	5	x*y
[1,0,1]	6	x*z
[0,2,0]	7	y^2

etc
.....

11) vectoint([1,0,1]) -> 6　上表参照

12) inttovec(6,3) -> [1,0,1]　〃

以上の如く、きめ細かい処理法が用意されている。
これを、Ｊ言語で実行する事は容易だと思うので、問題提起のみに止めよう。

なお、Ｍａｐｌｅの **numpart** や **Mathematica** の **Pa PartitionP** に相当する演算結果は、すでに、 分割数の一覧表 が出来ている。
例えば、インターネットの検索機能を利用すれば、日本国内外の研究レベルが痛切に判る。　**http://www.asahi-net.or.jp/　Hisanori Mishima**
分割数の桁数や、また分割数が素数か、またはその値の素因数分解まで併記すると云う親切さ！
例えば、 **P(302) = 10657331232548839 (17 桁) = prime** 素数
あたかも三角関数表を見る類であるので、思考の面白みは少ない。
しかし、「巨大数の素因数分解」の例題と見れば、面白い。 我が老友・山下氏は目下、せっせと計算中と聞く（文献 ８）。 期待したい。

６．　む　す　び

整数分割問題のおかげで、楽しませて頂いた。　副産物として、長大計算に於いては、Ｊ言語の限界が見えて来た。　また、プログラミングの難しさも具体化して来た。比較に利用した数学ソフト **Maple V** や　**Mathematica** の優秀さには驚かざるを得ない。　また、日本内外の同好の研究者にも、多大の敬意を表する。

文　　献

０）　**Makoto KOBAYASI and Toshihide MASKAWA:**
" CP-Violation in the Renormalizable Theory of Weak Interaction ", Feb 1973, Progress of Theoretical Physics, Vol.49, No.pp.652-657
－ａ）西川 **FAX ('08.10.8 pm.7:51)** 「坂田模型、小林・益川論文」
－ｂ）山下 **FAX ('08.10.11 am.9:32)**「ノーベル賞原論文拝受　**partition** ビックリ」

１）鈴木義一郎「４個のみかん：数の分割**(Partition)**」 **JAPLA 2008/Apr/26**
－ａ）志村メール **JAPLA discussion <JAPLA@aplsoft.co.jp> May 02,2008 9:36 AM**
「鈴木先生の力生の力作 **Partition** を試してみてください。 とても速く、 **2000** ぐらいでも平気でこなします。」
－ｂ）西川利男：「Ｊ６０２の新しい機能　Ｍ．について（続き）
数の分割（**Partition**）とそのＪプログラム」
JAPLA 2008/1/26 pp.6

２）中野嘉弘「整数分割　**PARTITION** 計算の話題（第２報相当）
演算結果は果たして正解か？」 **JAPLA 2008/Apr/26 pp. 8**

３）　**Yoshihiro NAKANO:" PARTITION Huge Integers Upto 140,005"**
VECTOR BCS UK (submitted 17 July 2008)

3－ａ）　Stephen Taylor: " Re: Offer Vector " 2008/9/20
http://www.jsoftware.com/jwiki/Essays/Partitions
provides functions
part, newn cat, final, pnv, 6!:2
－ｂ）中野嘉弘「＜速報＞　超々長大整数（２２万台）の分割」
JAPLA 2008/Sep/27 pp.4

４）西川利男：「数の分割とフェラーズの図」　**JAPLA 2008/2/23 pp.3**
－ａ）志村正人：「分割数（**partition number**）」　**JAPLA 2008/4/22 pp.4**

５）　「**Mathematica** ハンドブック」 **M.L.**アベル、**J.P.**ブレイセルトン共著、
川瀬宏海・五島奉文・佐藤　穂・田澤義彦　共訳
東京電機大学出版局、**1994.12.10** 第１版第１刷発行、**pp.342-343**

６）　Ｍａｐｌｅ Ｖ （**Windows**版）　**Help** コマンド　にて検索可能

７）　山下 **FAX** （**2008.10.20 am.9:55**）：「**Partition Numbers** の素因子分解表を
作成中　**n = 94** ～ **122** ～ **9973**」

```
NB.
    condpart =:  3 : 0
    :
    nX=. # X =. x
   Y =. y
   if. nX > 2 do. return. end.
      nY=. # Y
   if. nX =1 do. eln=. X
                 goto_1.  end.
   if. (# X)=2 do. eln=. 0{ X
                   ofst=. 1 { X
                 goto_2.  end.
  label_1.
      nYX =. <.  nY % eln
     t1 =. (nYX, eln) $ Y
 }.{}: </. (eln, >: nYX)$ t01=.(0,0{"1 t1),(1{"1 t1),0)
     goto_e.
   label_2.
     ry=. <. nY % ofst
      t02=. eln {. Y
     bt02=. < t02
    i=.1
   while. i < ry    do.
    t2i=. eln {. Y=. ofst }. Y
    bt2i=. < t2i
   bt02=. bt02,bt2i
    i=.i+1
    if. (eln +  i * ofst) > nY do. goto_ee. end.
   end.
   label_ee.
   bt02
   label_e.
   )
```