組合せ　COMBINATION　計算の話題

（附）　J 602　新機能　M. の盲点

中野嘉弘　（札幌市南区・８５才）

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

順列・組合せの計算法のＪ言語プログラムは、何処まで有効か？　確かめて見た。
必要な改良の例を示す。

０．　は　し　が　き

JAPLA １月例会の西川論文「J602 の新機能　M. について（続き）」（文献１）を拝見中に気付た話題がある。　新機能 M. (Memorize) を用いれば、メモリーオーバーになるかも知れぬ問題が、瞬時に解ける例があるとの事だ。
西川の例は「数の分割 (Partition)が幾通りあるか？」であった。
例えば、 p(1) = 1, p(2) = 2, p(5) = 7,　p(10) = 42,　p(100) = 190569292,
p(200) = 3972999029388 であった。
この際、計算をカナダ産の有名な数学ソフト　Maple V （学生版）でのトライでは、p(200) 相当の　numbpart(200) で　STACK OVERFLOW エラー　を発するので、 Ｊ言語の方がベター云々の記述が見えた。　しかし、中野のトライでは、格別悲観的でも無い。多少遡って、例えば numbpart(150) 辺りから、再出発して、再計算すれば、何とか突破出来るのだ。　その繰り返しで、遥かに大数の、なんと　numbpart(1550)　が　５０桁の個数　　6648312965925656816271400679772663779731　通りと計算出来た。
この件を中野は西川宛に ＦＡＸ した。　（文献２）

ところで、同様に大数になるであろう「順列・組合せ」の個数ならば、どうか？
１０ケの順列の個数 ！１０ ＝ 3628800、 ！１１ ＝ 39916800　は簡単だが、その「順列・組合せ」の　リスト を表示することは、至難の技である。
有名な数学ソフト Mathematica や Maple でも難儀らしく、解説や例題も乏しい。
しかし、リストの末尾の１０ケほどをを表示せよ程度ならば、出来てもよかろうに？
前記の有名数学ソフトでも、そもそも、そんな機能は避けているらしい。
最も簡単な場合、「１０ケの数から、１ケ宛選んだ場合のリストを示せ」ならば、暗算でも可能な筈だから、必ず出来そうなものだ。　Ｊ言語なら出来るか？
我らＪＡＰＬＡの会友、鈴木・北野の両氏の名著「Ｊ言語による数学計算」（文献３）にプログラム例があるので、それは「出来る」筈だ！
しかし １０ を超えて「１１ケの数から・・・」とすれば　" limit error " を発し、不可能となる。　その理由は、先ず　！１１　ケのリストを作成してから、すべての処理を始める算法なので、この場合は最初の段階でつまずいて仕舞うのだ。
暗算で可能なレベルの計算が J で「出来ぬ」とあっては、何とかせやならぬ。
これが、「はしがき」である。　ただし、簡単な「組合せ」の問題のみを考える。

実は、西川は取り上げていなかったが、 J602 beta （文献４）では、　M. 関数を Combination 関数　comb　に適用しようとして、実は何もしていない、奇妙に中途半端な解説があるのだ。　（文献５）
従って、西川も、全く、ノー　コメント　であったものだ。

１．　改良案　　その１

鈴木・北野の書の「 Combination 組合せ」は、n ケから r ケ のリストを選ぶ場合、関数 sample　を用い、演算　r sample n　で行う。（文献３ pp.33-35 ）
sample 関数では、最初に、順列リスト関数 plist でデータ数 n の階乗　!n ケの順列リストを用意してから始める。
ここで、limit error に引っ掛かるのだから、実用性には今一つである。
本稿では「組合せ」の記号には、印刷に不便な添字を用いずに、 C(n,r) 等で示す。時には　cn_r 等を用いる事もある。　さて、以下の事は容易に可能である。

C(n,n) のリストは、Ｊ言語流では i.n 、これは暗算！
即ち、例えば n=. 10 ならば （0 1 2 3 ....9)であり、その「組合せ」の個数は１ケ　である。

C(n, 1) の「組合せ」数は n ケ、そのリストは単列行列　cn_1 =. (n,1)$ i.n の個別要素であるから、これも、暗算！
前述の plist 関数では、この簡単な処理をも考慮していなかった。

C(n, n-1) の「組合せ」数も　n ケ、そのリストを求める関数も容易で、これを CN としよう。　そのプログラム（中野）は、

```
CN =: 3 : 0
  n=. y
  n1=. n - 1
 if. n = 2  do. cn=. 2 1 $ 0 1
                else. cn=. (i.n1), ((CN (n1)), "1 (1 # n1)) end.
)
```

C(n, 2) の「組合せ」数は　n*(n-1)% 2 ケ　である。　そのリストを求める関数を CN2 としよう。　そのプログラム（中野）は、

```
CN2 =: 3 : 0
  n=. y
  in=. i.n
  n1=. n - 1

   i =. 0
  c2=. (n1 # 0), ((0, n1# 1) # in)
  c2=. |: "1 c2

   i =. i + 1
 while.  i <: n1  do.
  c2i=. ((n1- i)# i ), (((i+1)#0), (n1- i)# ) # in
  c2i=. |: (2,(n1- i)) $ c2i
  c2=. c2, c2i
   if. i >: (n-2) do. goto_e.  end.
       i =. i + 1
 end.
 label_e.  c2
)
```

これだけの関数を追加するだけで、例のいやらしい **limit error** の幾つかは回避出来よう。

例１）　**C(11,11)**　個数は１、　リストは　**0 1 2 3 4 5 6 7 8 9 10**
こんな事が　**plist** 利用の方法では **limit error** で、不可能だったとは？！
例２）　**C(11, 1)** 個数は　**11** ケ, リストは単列行列の個別の各要素。

例３）　**C(11, 10)**　本稿での演算は　**CN 11** 、　結果のリストは
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 10
0 1 2 3 4 5 6 7 9 10
0 1 2 3 4 5 6 8 9 10
0 1 2 3 4 5 7 8 9 10
0 1 2 3 4 6 7 8 9 10
0 1 2 3 5 6 7 8 9 10
0 1 2 4 5 6 7 8 9 10
0 1 3 4 5 6 7 8 9 10
0 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10

例４）　**C(11, 2)**　本稿での演算は　**CN2 11** 、　結果のリストは
スペース節約の為、横長に転置して表現すれば
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
1 2 3 4 5 6 7 8 9 10 2 3 4 5 6 7 8 9 10 3 4 5 6 7 8 9 10 4 5 6 7 8 9 10

4 4 4 4 4 4 5 5 5 5 5 6 6 6 6 6 7 7 7 8 8 8 9
5 6 7 8 9 10 6 7 8 9 10 7 8 9 10 8 9 10 9 10 10

２．　改良案　　その２

「組合せ」　**C(n,r) r = n, 1, n-1, 2** なる簡単な場合は、前節１．で済んだ。
それ以外の場合でも、組合せ論の初等的公式
C(n, r) = C(n-1, r) + C(n-1, r-1)　　・・・・　（１）
を、利用すれば、逐次的に処理出来る。　例示しよう。

例１）　**C(4,3) = C(3,3) + C(3,2)**　　**r= n、n-1、 2** では、１．節通りで出来る。

0 1 2　0 1 ,3　　右辺、第２項に　新数字要素　３　を
0 2 ,3　　**append　, 3**　する。
1 2 ,3　リストの合体が求める左辺（４ ケ）である。

例２）　**C(5,3) = C(4,3) + C(4,2)**　前例１）の結果を利用して、右辺第１項は

0 1 2　0 1 ,4　　右辺第２項は **r = 2** 故、すぐ計算出来る。
0 1 3　0 2 ,4　それに、新数字要素　４　を　アペンドする。
0 2 3　0 3 ,4　右辺リストを合体した全体（１０ケ）が
1 2 3　1 2 ,4　左辺の「組合せ」 **C(5,3)** である。
1 3 ,4
2 4 ,4

例３）　**C(6,3) = C(5,3) + C(5,2)**　右辺第１項は、前例２）の計算結果（左辺）

である。　第２項は、 r = 2 故、すぐ計算
可能で、　前例２）の如く処理出来る。

例４）　C(6,4) = C(5,4) + C(5,3)　右辺第１項は、r = n-1 故、すぐ計算可能。
第２項は、前例２）の計算結果（左辺）で
あるが、それに新数字要素４をアペンドする。
右辺の合体が、左辺を与える。

例５）　C(6,6), C(6,5),C(6,2),C(6,1) は、暗算や、稿末のプログラムで容易に
処理出来る。

以下等々で、n = 7, 8, 9, 10, 11 etc　も、処理出来る。

３．　改良案　　その３

また、逐次法がここまで來れば、前節で問題の　鈴木・北野両氏の sample 関数を動作可能な範囲から、遡って、前述の如き逐次法を用いる事も可能であろう。
この際、「組合せ法」の第２公式（次の２段階飛躍法）を用いるのも、賢明かと思う。

C(n,r) = C(n-2,r)+2*C(n-2,y-1)+C(n-2,r-2)

これによれば、 n= 12 の場合を、　n = 10 の場合から、２段階飛ばして、一挙に逐次近似出来る。

４．　改良案　　その４

ここまでくれば、任意の　C(n,r) について、処理出来る一般関数が作れよう。

```
 CNR =:  3 : 0    NB. ( M.  with  or without)  and ( 3 : 0  or 4 : 0 )
  :
     if. y < x do. return. end.
wr  n=. y        NB. wr があれば　iteration の進行状況が判る便利あり
wr  r=. x        NB. wr は、不要ならば、 共に省略可
     in=. i.n
     n1=. n-1
     r1=. r-1
if.  r= n  do.  cnr=. in
                              goto_e.  end.
if . r= n1 do.  cnr =. CN n
                              goto_e.  end.
if.  r= 1  do.  cnr =. (n,1) $ in
                                    goto_e.  end.
if.  r= 2  do.  cnr =. CN2 n          NB. CN2 は用いずも可
                                    goto_e.  end.    NB. 計算を少し早めるのみ

cnr1 =. r1  CRN  n
       i =. 0
    cnr =. (1,r)$ i.r )
    ne=. 0 { $ cnr1
while.  i < ne  do.
 icn1 =. i { cnr1
 ce=. {: icn1
```

```
    ce1=. ce + 1
    nre=. n - ce1
      if. nre= 0  do. goto_i.  end.

     cr =. ((nre, r1) $ icn1), "1 (nre,1) $ ( ce1 }. in )
     cnr=. cnr, cr
    label_i.  i =. i + 1
    if. i=1 do. cnr =. }. cnr end.
   end.
    label_e.  cnr
  )
```

この関数 CNR は、後節の例の如く、万能である。

５．　Ｊ６０２－ｂｅｔａ 内の機能　M. の話題

冒頭の　０．はしがき　の末尾で触れたが、 J602-beta 、Help、 Vocabulary、 M. Memo 内の「組合せ関数」　comb から

```
  combm =: 4 : 0  M.       NB.  4 : 0  with  M.,
   if. (x>:y)+.0 = x  do. i. (x<:y),x
        else. (0,.x comb &. <: y), 1 + x comb y-1 end.
  )

 comb0 =: 4 : 0        NB.  4 : 0   without  M.
```

を用意する。

テスト例：　中野のマシンは　SHARP （ノート型）　Mebius PCAL70F　Win-XP Home Ed. CPU モバイル　AOD 2000: 333MHz 768 MB 、 HDD 60 GB

```
 1）      3 combm 5
0 1 2
0 1 3
0 1 4
0 2 3
0 2 4
0 3 4
1 2 3
1 3 4
2 3 3     １０ ケ リスト

   timer ' 3 combm 5 '
4.1345e_5 sec ( ～  0.04 msec )
   timer '3 comb0 5'
0.000436368 sec ( ～  0.4 msec )
```

M. 関数の効果で、１０倍も速くなった。　（後節に議論あり！）

２）鈴木・北野関数　sample （plist 使用） 、中野関数　CNR （iteration 法）等との比較

```
   timer ' 3  sample  5 '
```

0.0995168 sec (～ 0.1 sec)

timer ' 3 CNR 5 ' NB. without M.
0.02777069 sec (～ 0.03 sec) 約 ３ 倍 速い

３）より大数でのテスト例

● 鈴木・北野関数
timer '3 sample 10'
4.66937 sec (～ 5 sec)
timer '3 sample 11'
limit error

◎ 中野関数
timer ' 3 CNR 10 ' NB. without M.
0.0018385 sec (～ 2 msec)
timer ' 3 CNR 10 ' NB. with M.
0.00183236 sec (～ 2m sec) 両者、差無し

timer ' 3 CNR 11 ' NB. without M.
0.00214022 sec (～ 2 msec)
timer ' 3 CNR 11 ' NB. with M.
0.00216284 sec (～ 2 msec) 両者、差無し

timer ' 3 CNR 13 ' NB. without M.
0.002888 sec (～ 3 msec)
timer ' 3 CNR 13 ' NB. with M.
0.00285371 sec (～ 3 msec) 両者、差無し

timer ' 3 CNR 15 ' NB. without M.
0.00378903 sec (～ 4 msec)
timer ' 3 CNR 15 ' NB. with M.
0.00397984 sec (～ 4 msec) 両者、差無し

◆ Ｊ６０２ｂｅｔａ 内の 「組合せ」関数 **comb** での例
timer ' 3 comb0 13 ' NB. without M.
0.0105368 sec 中野関数より５倍遅い
timer ' 3 combm 13 ' NB. with M.
6.34159e_5 sec (～ 0.06 msec) 中野関数より５倍速い

timer ' 3 comb0 14 '
0.0137928 sec 中野関数より５倍遅い
timer ' 3 combm 14 '
0.000300876 sec (～ 0.3 msec) 中野関数より１０倍速い

timer ' 3 comb0 15 '
0.0166211 sec 中野関数より５倍遅い
timer ' 3 combm 15 '
0.000359543 sec (～ 0.4 msec) 中野関数より１０倍速い

即ち、Ｊ６０２ｂｅｔａ 内の「組合せ関数 **COMB** 」は、新機能 **M.** が無い時は、中野関数より 約５倍 遅い。 **M.** を利用すれば、逆に、約１０倍速くなる。

かくて、 M. MEMO 関数のメリットが明白になった。
Ｊ６０２ｂｅｔａ 内の当該解説の重点項目になり得ると思う。

そこで、機能 M. 以前の版では、「組合せ」 COMBINATION 計算には、中野関数を利用する方がベターかも知れぬ。

Ｊ６版以上の作業でも、新機能 M. を正しく使わない（使えない）時には同じ事が云えそうだ。 では「正しく使う」とは、どんな事か？

６． M. 関数の真価

前節までで、M. MEMO 関数のメリットを明かにしたつもりであるが、事は簡単では無いのだ。 ０．はしがき で述べたが（ＦＡＸの件、文献２）、その返事の西川ＦＡＸ（文献６）では、「 M. （memorize） の効果は未だ、明かでは無い。会友の鈴木義一郎氏の近著「幸運数：改訂版」（文献７） でも、M. の有無は、小差に過ぎない。」とあった。 そこで、この辺をもう少し子細に調べよう。

前節５．のテスト例
１） 3 comb 5 で再調査する。

■ with M. の combm 関数　　without M. の comb0 関数

新 timer ' 3 combm 5 '
0.0734397 sec 遅いな！
再 timer ' 3 combm 5 '　　新 timer ' 3 comb0 5 '
3.15683e_5 何故か？ 俄然、速い。　　0.000429943 sec 最初から速い。
再三 timer ' 3 combm 5 '　　再 timer ' 3 comb0 5 '
3.12889e_5 同様に速い！　　0.000434692 上 同様に速い！
四回目 timer ' 3 combm 5 '　　幾度繰り返しても同様。
3.12889e_5 同じ値に収束
最初は遅いが、次回から、迅速化される。

▲　　新 timer ' 3 comb0 14 '
0.0133704 sec
新 timer ' 3 combm 14 '　　再 0.0135732
0.000298641 sec　　三 0.013288
再 6.31365e_5 sec　　四 0.013288
三 6.39746e_5 sec　　幾度繰り返しても同様。
最初は遅いが、次回から、迅速化される。

２） 中野関数 CNR 定義は動詞 3 : 0 を再調査する。
with M. 3 : 0　　without M. 3 : 0
新 timer ' 3 CNR 5 '　　新　　左に同じ
0.1888458 sec
0.000634997 sec　　再 0.000615721
再 5.95048e_5　　三 0.000615441
三 5.67111e_5　　四 0.000707073
四 6.00635e_5　　次回から、ある値に収束する。
次回以降、圧倒的に迅速化される。

３）　動詞定義　3 : 0　と　接続詞定義　4 : 0　とでの比較
　a, comb 関数ではどちらか片方で、 domain error を起こす事がある。
　b. 中野関数　CNR では、どちらでも「可」であった。

４）　鈴木氏「幸運数：改訂版」（文献７）　中の計算について
「 p.2 中央以下： timer=: 6!:2　として　timer ' r15=: fortune 15 或いは 100
或いは 1000 或いは　10000, 100000, 1000000' の計算を、　without M. の
fortune0 と　with M. の　fortunem　とで比較計算する例 」
これを、初回計算のみの比較と、再度以上計算した場合との比較とを、調べる。

without M.　　with M.
timer ' r15=: fortune0 15'　初　timer ' r15=: fortunem 15'
0.00288305 sec　0.00292886 sec　時間は左と差なし、
r15 のリストは　1 7 10 13　むしろ 若干遅い感じ

timer ' r10p6=: fortune0 10^6'　初　timer ' r10p6=: fortunem 10^6'
243.31 sec　243.634 sec
個数 # r10p6 は 143071ケ　再度での時間　5.78286e_5 sec　極短！

より詳しくは、読者自らトライされれば、中野の申す事が御納得出来よう。
（なお、中野マシンは鈴木マシンより２倍ほど高速だと思われた。）

「 新機能　M.　は、まさに stack 等　something の　" memorize "　である。
初回にメモれば、次回からは、それを利用するので圧倒的に迅速！」

以上は、所見のみであるが、 M.　関数の有効な使い方及び評価の参考になろう。
詳しい事は JAPLA 会友諸賢に委ねる。

７．　む　す　び

「組合せ」 Combination を計算するのに、旧法（先ず plist 順列リストを用意してから行うやり方）は、避けた方が良い。　時間が掛かるし、メモリの limit error を生じて、停止になる可能性がある。
それに対する幾つかの改良案を示した。　取りとりあえずのものである。
もっとエレガント化も可能だろう。　諸賢の御工夫に期待する。
なお、Ｊ６０２ｂｅｔａ　内の新機能　M.　（memorize）などの該当解説や関連事項をも話題にした。　我が会友間には、この M.　の効能・利用法について、悲観的な誤解があるらしい。　中野は楽観的意見を報告した。

文　　献

１）西川利男：「J602 の新機能　M. について（続き）」 JAPLA資料 2008/1/26 pp.6
－ａ）　西川利男：「J602 の新機能　M. について」JAPLA symp 2007/12/8 pp.2
２）中野ＦＡＸ：「numbpart (1550) までのテスト」 2008/1/30 pp.5
３）鈴木義一郎・北野利雄：「Ｊ言語による数学計算」森北出版、1996.10.14, p.35
§２．１０　標本のリストアップ
４）志村 Ｅメール： <jcd02773@nifty.ne.jp>、 多々あるが、 最近では、例えば
JAPLA discussion <JAPLA@aplsoft.co.jp>

(SHIMURA)J602-kbeta Feb. 15, 2008

５） **M. 関数： J602-beta タスクバー　Help　Vocabulary　M. Memo**
　　関数　timer, fib, pn 、 関数　comb と　例.　3 comb 5 のリスト
６）**西川 FAX(2008-1-31)：「分割数、MapleV (numbpart (100))、 M. の効果？」**
７）**鈴木義一郎：「幸運数　fortune number ：　改訂版」 JAPLA 2008/01/26 pp.5**
「M. の利用は計算時間の短縮には、ほとんど寄与しない事が検証された」と云う。