JAPLA シンポジウム資料 2008/12/6

Boss & Hui による数の分割プログラム

西川　利男

数の分割を求める Boss と Hui の J プログラム[1]は強力で高速である。そして実行するのは手軽だが、そのコーディングから処理の内容を理解しようとすると Tacit で fork や hook を駆使していて至難のわざである。

今回、チュートリアルのテーマとして、このプログラムを解析することを試みた。Tacit から Explicit への書き換えなども使って処理の詳細をトレースしてみよう。

１．Boss & Hui 作成の分割プログラムとその実行例

プログラムの主要部分は次のようにたった２行である。

```
new =: (-i.)@#<@:(cat&.>)]
cat =: [;@:(,.&.>)-@(<.#){.]
```

実行は次のように行われる。

初期値を設定する。

```
   y0 =: <<i.1 0
   y0
+--+
|++|
||||
|++|
+--+
```

そして例えば数３までの分割を求めるには、次のようにすれば良い。

```
   y3 =: (, new)^:3 y0
   y3
+--+---+-------+-------------+
|++|+-+|+-+---+|+-+---+-----+|
|||||1|||2|1 1|||3|2 1|1 1 1||
|++|+-+|+-+---+|+-+---+-----+|
+--+---+-------+-------------+
```

これを元に４の分割は次のように求められる。

```
   new y3
+---------------------+
|+-+---+-----+-------+|
||4|3 1|2 2 0|1 1 1 1||
|| | |  |2 1 1|      ||
|+-+---+-----+-------+|
+---------------------+
```

[1] http://www.jsoftware.com/jwiki/Essays/Partitions.html/

(, new)はhookであるので、１からの数の分割は次のように順々に求められる。

```
   y0
+--+
|++|
||||
|++|
+--+
   ]Y1 =: new y0
+---+
|+-+|
||1||
|+-+|
+---+
   ]y1 =: y0, Y1
+--+---+
|++|+-+|
|||||1||
|++|+-+|
+--+---+
   Y2 =: y1
   ]y2 =: y1, Y2
+--+---+-------+
|++|+-+|+-+---+|
|||||1|||2|1 1||
|++|+-+|+-+---+|
+--+---+-------+
   Y3 =: new y2
   ]y3 =: y2, Y3
+--+---+-------+-------------+
|++|+-+|+-+---+|+-+---+-----+|
|||||1|||2|1 1|||3|2 1|1 1 1||
|++|+-+|+-+---+|+-+---+-----+|
+--+---+-------+-------------+


   Y4 =: new y3
   ]y4 =: y3, Y4
+--+---+-------+-------------+---------------------+
|++|+-+|+-+---+|+-+---+-----+|+-+---+-----+-------+|
|||||1|||2|1 1|||3|2 1|1 1 1|||4|3 1|2 2 0|1 1 1 1||
|++|+-+|+-+---+|+-+---+-----+|| |   |2 1 1|       ||
|  |   |       |             |+-+---+-----+-------+|
+--+---+-------+-------------+---------------------+
```

２．Boss & Hui のアルゴリズムの考え方

Boss & Hui のアルゴリズムの優れた点は、数０の分割(null)を導入したことと、逆順の数を以前の分割に結合することによって、次の分割を得るという発想の転換である。これを筆者なりに解釈し、以下のように図とことばで説明してみた。

例えば、y2 から新しく Y3 を生成することを見てみる。これには

① y2 の要素数３の逆順整数ｎ = ３ ２ １を y2 の要素の前にそれぞれ結合する。

②.その際、

ｍ =: ｎ <. #y2 … ｎと#y2（要素の数）のうち小さい方の値

とした上で、y2 の要素のうち、後ろからｍ個だけを結合する。

（y2 の要素のうちｎより大きい値を含む組は結合しない…西川 注）

```
    ++              +-+
3   ||  -------->   |3|
    ++              +-+

    +-+             +---+
2   |1|  ------->   |2 1|
    +-+             +---+

    +-+---+         +---+  +-----+
1   |2|1 1|  --->   |1 2|, |1 1 1|
    +-+---+         +---+  +-----+
```

同様にして、y3 から新しく Y4 を生成する。

```
    ++                  +-+
4   ||  ------------->  |4|
    ++                  +-+

    +-+                 +---+
3   |1|  ------------->  |3 1|
    +-+                 +---+

    +-+---+             +---+  +-----+
2   |2|1 1|  --------->  |2 2|, |2 1 1|
    +-+---+             +---+  +-----+

    +-+---+-----+       +---+  +-----+  +-------+
1   |3|2 1|1 1 1|  --->  |1 3|, |1 2 1|, |1 1 1 1|
    +-+---+-----+       +---+  +-----+  +-------+
```

このようにして、それまでの分割に上のような操作をおこなうことで、新しい分割が得られ、これを以前の分割に付け加えるというアルゴリズムである。

３．定義動詞 new と cat の解析

まず定義された new と cat の構造を見てみる。

```
   new
+-----------+----------------+-+
|+------+-+-+|+-+--+----------+|]|
||+-+--+|@|#|||<|@:|+---+--+-+|| |
|||-|i.|| | ||| |  ||cat|&.|>||| |
||+-+--+| | ||| |  |+---+--+-+|| |
|+------+-+-+|+-+--+----------+| |
+-----------+----------------+-+


   cat
+-+---------------+-----------------+
|[|+-+--+--------+|+-----------+--+-+|
| ||;|@:|+--+--+-+|||+-+-+------+|{.|]||
| || |  ||,.|&.|>|||||-|@|+--+-+||  | ||
| || |  |+--+--+-+|||| | ||<.|#|||  | ||
| |+-+--+--------+|||| |  |+--+-+||  | ||
| |                ||+-+-+------+|  | ||
| |                |+-----------+--+-+|
+-+---------------+-----------------+
```

３．１　new と new0 の動作

上で分かるように動詞 new は fork であり、これを次のように分けて考える。

```
   new0 =: (-i.)@#
   new1 =: <@:(cat&.>)
```

とすると、new は次のような new0, new1,] から成る fork になる。

```
   new =: new0 new1 ]
```

したがって、引数 y に作用させると、new y は

```
   (new0 y) new1 y
```

のように働く。

まず、new0 を考える。ここで、new0 における(-i.)は hook であるので、先に得た y1, y2, y3 に new0 を作用させると、次のようになる。

```
   (-i.)@#y1 ⇔ 2-i.2 = 2-(0 1) = 2 1
```

から

```
   new0 y1
2 1
   new0 y2
3 2 1 ⇔ 3-i.3 = 3-(0 1 2) = 3 2 1
   new0 y3
4 3 2 1 ⇔ 4-i.4 = 4-(0 1 2 3) = 4 3 2 1
```

したがって new を作用させることは、次のようになる。

```
   (2 1) new1 y1
   (3 2 1) new1 y2
   (4 3 2 1) new1 y3
```

これは、先の逆順整数nを作ることに相当する。

３．２　new1とcatの動作

new1の処理を見ると、

```
   new1 =: <@:(cat&.>)
```

これはサブプログラムとしてcatを呼んでいる。

ここで、cat&.>を見ると、Jの副詞的定形処理である‘それぞれ演算(each)’を利用している。

```
   each =: &.>
```

すなわち、new1は

①引数のそれぞれのボックスを開いて
②それぞれにcatを作用させ
③その結果をそれぞれボックス化する

そして、最後に全体をさらにボックスで囲む

さらにcatを細かく見てみる。catは２項動詞でforkである。

```
   cat =: [;@:(,.&.>)-@(<.#){.]
```

これも前と同様に、分けて考える。

```
   f =: ;@:(,.each)
   g =: -@(<.#){.]
```

とすると、

```
   cat =: [ f g
```

と置き換えられる。

そこで、例えばy2を例にとって、動詞catの動作をひとつずつ見ていく。

```
   y2 =: Y0, Y1, Y2
   y2
+--+---+-------+
|++|+-+|+-+---+|
|||||1|||2|1 1||
|++|+-+|+-+---+|
+--+---+-------+
   3 2 1 ([ f g) each y2
```

をやってみよう。これは左右の引数を別々にほぐしてそれぞれ実行する。つまり

```
   3 f (3 g Y0)
   2 f (2 g Y1)
   1 f (1 g Y2)
```

を行なう。

そしてY2からやってみる。

```
   Y2 =:  >2{y2
   Y2
+-+---+
|2|1 1|
+-+---+
```

まず

```
   1 g Y2
```
をやってみる。
この結果は
```
   (1 (-@(<.#)) Y2) {. Y2
+---+
|1 1|
+---+
```
となるが、これは細かく実行した以下のことから明らかになろう。

まず、<.#はhookとなるので
```
   1 (<.#) Y2
1
   1 (-@(<.#)) Y2
_1
   (1 (-@(<.#)) Y2) {. Y2
+---+
|1 1|
+---+
```
これにfを作用させることは、ボックスを開いて左引数と結合(,.)して戻す（つまりeach）、さらにリストにほぐす(;)。したがって、次のようになる。
```
   1 f (1 (-@(<.#)) Y2) {. Y2
1 1 1
```

y2の次の要素についてもやってみる。
```
   Y1 =: >1{y2
   Y1
+-+
|1|
+-+
   2 ([ f g) Y1
2 1
```

y2のさらに別の要素についても行なう。
```
   Y0 =: >0{y2
   Y0
++
||
++
   3 ([ f g) Y0
3
```

これらをまとめて行なうと
```
   3 2 1 (cat &.>) y2
```

```
+-+---+-----+
|3|2 1|1 1 1|
+-+---+-----+
```

さらに new1 はこれをボックス化するものである。

```
 3 2 1 new1 y2
+-------------+
|+-+---+-----+|
||3|2 1|1 1 1||
|+-+---+-----+|
+-------------+
```

ちなみに、動詞 f, g に対応する Explicit の動詞定義を以下のように fn, gn として作ってみた。

```
fn =: 3 : 0
:
; x. ,"(1) L:0 y.
)

gn =: 3 : 0
:
(- x. <. # y.) {. y.
)
```

これによる実行は次の通りである。

```
   1 fn (1 gn Y2)
1 1 1
   2 fn (2 gn Y1)
2 1
   3 fn (3 gn Y0)
3
```

注) －－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

なお、先の注に従った西川版 gnn でも同じ結果が得られる。

```
NB. modified version by T.N.
gnn =: 3 : 0
:
(> */ L:(0), L:(0) x. >: L:(0) y.) # y.
)

   1 fn (1 gnn Y2)
1 1 1
   2 fn (2 gnn Y1)
2 1
   3 fn (3 gnn Y0)
```

3

同様にして、y3 に対してもやってみる。

```
   y3
+--+---+-------+-------------+
|++|+-+|+-+---+|+-+---+-----+|
|||||1|||2|1 1|||3|2 1|1 1 1||
|++|+-+|+-+---+|+-+---+-----+|
+--+---+-------+-------------+
   4 3 2 1 cat each y3
+-+---+-----+-------+
|4|3 1|2 2 0|1 1 1 1|
| |   |2 1 1|       |
+-+---+-----+-------+

   4 3 2 1 new1 y3
+--------------------+
|+-+---+-----+-------+|
||4|3 1|2 2 0|1 1 1 1||
|| |   |2 1 1|       ||
|+-+---+-----+-------+|
+--------------------+

   y4 =: y3, 4 3 2 1 new1 y3
```

さらに y4 に対してやってみる。

```
   y4
+--+---+-------+-------------+--------------------+
|++|+-+|+-+---+|+-+---+-----+|+-+---+-----+-------+|
|||||1|||2|1 1|||3|2 1|1 1 1|||4|3 1|2 2 0|1 1 1 1||
|++|+-+|+-+---+|+-+---+-----+|| |   |2 1 1|       ||
|  |   |       |             |+-+---+-----+-------+|
+--+---+-------+-------------+--------------------+
   5 4 3 2 1 new1 y4
+------------------------------+
|+-+---+-----+-------+---------+|
||5|4 1|3 2 0|2 2 1 0|1 1 1 1 1||
|| |   |3 1 1|2 1 1 1|         ||
|+-+---+-----+-------+---------+|
+------------------------------+
```

４．おわりに

Boss & Hui の２行のプログラムはひとつずつ追っていけば、決してマジックのようなものではない。

しかしながら、筆者に言わせれば、真の価値はマニアックでトリッキーな Tacit プログラムにあるのではなく、その発想の逆転のアルゴリズムにある。その意味では Explicit プログラムに書き換えてみることは教育的に大いに意義がある。