JAPLA 研究会資料 2010/9/25

Ｊのコード探訪－１　クィックソート
－英単語の辞書順ソートへの応用－

西川　利男

インターネットで J-Wikipedia を見ていたら、クィックソートのコードが出ていた。Ｊにはソートのプリミティブとして /: や \: があるので、これをわざわざコーディングする必要はない。しかしアルゴリズムの学習として、クィックソートはどんな教科書にも取り上げられ、よく Pascal のコーディングが載っている。*)
ここでは、Ｊのコンパクトで強力なプログラミングスタイルを学ぶ意味で、われわれにとっても興味深いと思われる。

クィックソートのアルゴリズムの原理は簡単である。

ソートしたい集合から例えば先頭の要素を取り出しキーとする。このキーと、残りの集合との値を比較して、小さい要素の集合と大きい要素の集合に分割する。そして次のように並べる。
　(小さい要素の集合), (キー), (大きい要素の集合)
そして、それぞれのサブ集合について同じ操作を再帰的におこなう。
つまり、小さい要素の集合について、キーと分割の操作をおこない、また、大きい要素の集合についても、キーと分割の操作をおこなう。このそれぞれの操作を再帰的におこない、要素が１つになったら終了する。
なお、キーは集合の中央値でもよいし、ランダムに取り出したものでもよい。

Ｊではこのアルゴリズムのとおりをコーディングするだけでよい。まず、Explicit のプログラムを実行例を示す。

```
NB. Quicksort from J Wikipedia

sel =: adverb def 'u # ['

quick =: verb define
  if. 1 >: #y do. y
  else.
     e =. y{~?#y
     (quick y < sel e), (y = sel e), (quick y > sel e)
  end.
)
   quick 5 7 2 1 8 4 6 3
1 2 3 4 5 6 7 8
```

---

*) たとえば、R.Sedgewick, “Algorithm” p.115-131, Addison Wesley(1988).

このコーディングを検討してみよう。
まず、小さな副詞(sel)が定義されている。

```
   sel
+-+-+-----+
|1|:|u # [|
+-+-+-----+
```

これは、動詞といっしょに用いて、動詞の操作に合った要素を選び出す。
例えば、

```
   1 2 3 4 5 < sel 3
1 2
   1 2 3 4 5 = sel 3
3
   1 2 3 4 5 > sel 3
4 5
```

副詞を使って、一般的にしているので分かり難いが、具体的な動詞でおこなえば、例えば、次のようである。

```
   1 2 3 4 5 (< # []) 3
1 2
```

コード < # [は、いわゆる fork である。段階ごとに実行すれば次のようになる。

```
   1 2 3 4 5 < 3            NB. 3より小さいかどうかのテストのフラグを返す。
1 1 0 0 0                   NB. 真は1、偽は0
   1 2 3 4 5 [ 3            NB. 単に左引数を返すだけである。
1 2 3 4 5
   1 1 0 0 0 # 1 2 3 4 5    NB. 左引数が1の場所の右引数を返す。
1 2
```

これを用いて、以下の動詞quickを再帰的におこなう。つまりquickの定義の中で、動詞quickを呼んでいる。その際、1 >: #y で示すように引数の数が一つになるまで続ける。 コード e =. y{~?#y は大小比較のキーをランダムに取りだす。これでクィックソートのプログラムが出来上がる。

```
   quick
+-+-+-------------------------------------------------+
|3|:|  if. 1 >: #y do. y                              |
| | |  else.                                          |
| | |     e =. y{~?#y                                 |
| | |     (quick y < sel e), (y = sel e), (quick y > sel e)|
| | |  end.                                           |
+-+-+-------------------------------------------------+
```

同じクィックソートをJの特徴とするtacitで定義すると、以下のようにたった一行でコーディングできる。

```
   quicksort =: (($: @(<#[) , (=#[) , $: @(>#[)) ({~ ?@#)) ^: (1:<#)

   quicksort 5 7 2 1 8 4 6 3
```

1 2 3 4 5 6 7 8

Jのこのコードの意味を知るには、かなり大変のように見えるが、次のようにfork, hookの連続(train)として、理解すればそれほどのことはない。

```
   quicksort =: (($: @(<#[) ,  (=#[) , $: @(>#[)) ({~ ?@#))   ^: (1:<#)
                (- @(---)) - ((---) - (- @(---)) ((--)(-@-)) -  (- - -)
                      f         f          f       h              f
               (---------) -  (-----------------) (---------) -  (-----)
                          fork
               (--------------------------------) (---------) -  (-----)
                                                hook
               (-----------------------------------------) -  (-----)
                                                verb          conj  verb
```

なお、つぎのようにして、文字の値が動詞(val)により簡単に得られるので、接続詞(&. under)を使って、文字列を文字の集合と見たときのソートも簡単に行える。

```
val=: a. & i.
    val 'abc'
97 98 99
    quicksort &. val 'quick sort examples.'
  .aceeiklmopqrsstux
```

さらに、文字列の語単位での辞書順のソートも次のようにして可能である。

```
Alpha =: ,((97+i.26){a.),.((65+i.26){a.)
Alphas =: ' ', Alpha
   Alphas
 aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ
```

このように、スペースを先頭にした文字順のキーAlphasを作る。これを用いて、文字列'aAbBcC...'などを53進数の数値として扱うことによりソートを行う。

```
valalp =: Alphas & i.
ord =: (53"_) #. valalp   NB. return 53-based number of character string
rmsp =: (' '&~:)# ]       NB. remove space
```

次のように実行される。

```
   DD =. 'apl';'apple';'ape';'angel'
   DD
+---+-----+---+-----+
|apl|apple|ape|angel|
+---+-----+---+-----+
   quicksort &. ord &. (>"(1)) DD
+-----+-----+-----+-----+
|angel|ape  |apl  |apple|
+-----+-----+-----+-----+
   rmsp L:0 quicksort &. ord &. (>"(1)) DD
+-----+---+---+-----+
```

```
|angel|ape|apl|apple|
+-----+---+---+-----+
```