Program Promenade

連載

穴を覆う蓋

和田 英一（IIJ 技術研究所）
wada@u-tokyo.ac.jp

■問題の説明

2003年3月の世界大会の問題E, Covering Whole Holesをやってみよう．これはホールホールズと韻を踏んでいるので気持ちいいが，それはそれとして，出場68チーム中，手をつけたのが1チーム，解いたのは0チームと聞き，どんなに難問かと挑戦してみたくなった．

問題に載っていたサンプル入力は以下のとおり．

```
4 4
0 0
0 10
10 10
10 0
0 0
0 20
```
（右上に続く）

```
20 20
20 0
4 6
0 0
0 10
10 10
10 0
```
（右上に続く）

```
0 0
0 10
10 10
10 1
9 1
9 0
0 0
```

最初の4 4は穴の角の座標の個数h(*hole*)が4, 蓋の角の座標の個数c(*cover*) が4の意．以下0 0から10 0まで4行が第1組の穴の角の(xとy)座標；その次の4行が第1組の蓋の角の(xとy)座標である．紙面の節約上，続く入力は右の列に示す．$h+c$行の第1組の座標の表示が済むと，その次の4 6は第2組のデータにつき，穴と蓋の角の座標の個数がそれぞれ4行と6行であることを示す．h, cの対が0 0なら入力の終わりを示す．このサンプル2組のデータの示す穴と蓋を**図-1**のaからdに示す．問題は穴と蓋の対を与えられて，蓋で穴を完全に覆えるかというのである．最初のデータの組ではもちろん覆えるが，2番目のでは蓋の隅が欠けているため覆えない．

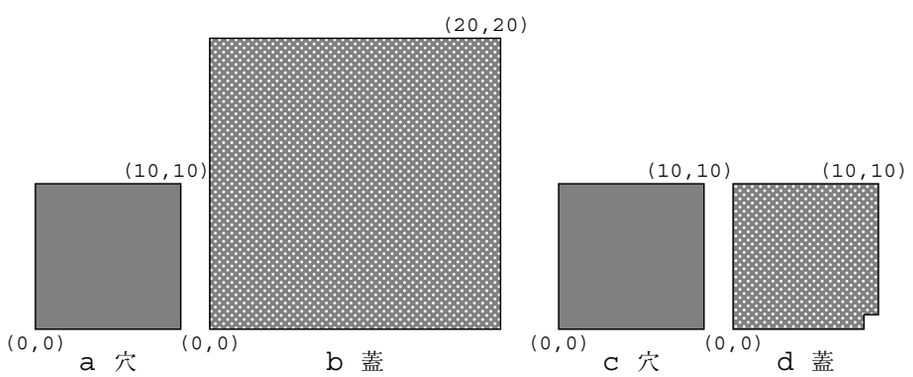


図-1

問題によると穴も蓋も直角からなる多辺形で，周囲の各辺はx軸かy軸に平行になっている．蓋は上下左右に平行にずらすことはできるが，回転はしない．各組の座標点の数は4以上50以下で，座標値は整数．

ただしその値の範囲は与えられていない．穴と蓋が完全に同じときは覆えるというのか，蓋が穴に落ちて覆えないのか，分からない．この辺は世界大会の問題の多少ずさんなところである．

■基本方針

解き方はいろいろ考えられようが，楽そうなのはペントミノ方式かと思う．つまり蓋で穴が覆えるかの代わりに，蓋の形の容器に，穴の形のピースが置けるか調べるのである．図-2の例で考えてみよう．

```
6 12
0 0
-1 0
-1 2
2 2
2 -1
```
（右上に続く）

```
0 -1
0 0
0 -1
2 -1
2 0
3 0
```
（右上に続く）

```
3 2
2 2
2 3
0 3
0 2
-1 2
-1 0
```

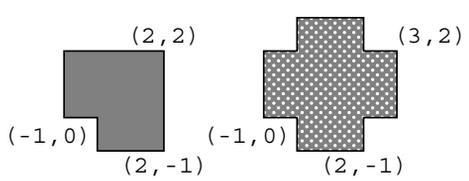


図-2

```
  e      f
111   0110
111   1111
011   1111
      0110
```
図-3

```
   e        f
111??    01100
111??    11110
?11      11110
         0110
```
図-4

図-2は図-1の倍のスケールになっている．ペントミノの解き方は昨年8月の本欄にあるが，そこではバックトラックの例であった．今回は文字列サーチを使うので，日を同じくして語ることはできない．

まずe(穴)とf(蓋)を図-3のようにし，eの文字列がfの文字列の中に存在するかを見る．ただし，行を跨って存在しても困るので，fには行末の符号 0(箱の外を表す)を置き，eは行長をfの幅にした後，欠けている個所を示す0と行末は，fの1とでも0とでもマッチするという記号?に換える．つまりfとeはそれぞれ図-4のようになり，それぞれ1行にして

```
蓋f 01100111101111100110
穴e 111??111???11
```

で表現する．そうするとeを最後の近くまでずらした

```
蓋f 01100111101111100110
穴e       111??111???11
```

がfにマッチし．ピースが枠に入る，つまり蓋は穴を完全に覆うことができる．したがって解決すべきは，(1)座標のリストから文字列を構成する方法と，(2)文字列から特定のパターンを探す方法である．後者はBoyer-Mooreの方法が応用できる．

■駆動プログラム

いつもと反対に駆動プログラムから説明しよう．

```
(define (holes)
  (let* ((h (read s)) (c (read s)))                    ;hとcを読む
    (if (> (+ h c) 0)            ;h, cがともに0でなければ
        (let* ((hv (readvs h)) (cv (readvs c)))        ;座標を読む(後掲)
          (newline) (display hv) (newline) (display cv) ;デバッグ用
          (let* ((hs (make-str (norm hv) #\?))         ;正規化し文字列に
```

```
                (cs (make-str (norm cv) #\0))
                (hl (string-length (car hs)))
                (cl (string-length (car cs)))
                (hx (map char->integer (string->list   ;蓋の長さにする
                  (conc hs (make-string (- cl hl -1) #\?)))))
                (cx (map char->integer (string->list (conc cs "0")))))
            (newline)
            (if (covered? hx cx) (display 'yes) (display 'no)))
            (holes)))))          ;次の組へ再帰呼び出し
(define s (open-input-file "holes.data"))                ;入力データファイル
(holes)                                                  ;プログラム起動
```

holesが1組のデータに対し解答する手続きである．最初のlet*でhとcを読む．letにすると，hとcのデータのどちらを先に評価するかで，入力が反対になる．前の変数から順にデータを読むためには，let*を使う必要がある．次のifはhとcが0でないことを見ている．0なら終わる．穴hvと蓋cvのデータを読むが，ここもlet*を使う．読み込んだら次のlet*で基本方針によりデータを処理し，文字列（というより，ここでは整数列になっているが）hx, cxに対し，covered?で判定し，yes, noを出力する．そしてholesを再帰的に呼び出し，ループする．それぞれの処理はこれから述べる．

■データの読み込み

*x*座標と*y*座標の組はデータが2個なのでドット対として読む．読込み部分は以下のとおり．

```
(define (readvs n)       ;n組のデータを読み，ドット対のリストにする
  (if (= n 0) '()
      (let* ((x (read s)) (y (read s))) ;x, yの座標を読む
        (cons (cons x y) (readvs (- n 1))))))    ;次の組の前におく
```

これでeとfのデータを読んでみると次のようになる．

```
((0 . 0) (-1 . 0) (-1 . 2) (2 . 2) (2 . -1) (0 . -1)))
((0 . 0) (0 . -1) (2 . -1) (2 . 0) (3 . 0) (3 . 2) (2 . 2) (2 . 3)
  (0 . 3) (0 . 2) (-1 . 2) (-1 . 0)))
```

サンプルデータは第1象限におさまり，*x*, *y*両軸に接し，時計回りに記述してあったが，このデータは座標に負数もあり，データ記述もeは時計回り，fは反時計回りと意地悪くできている．問題文にはこのようなデータ記述の条件はない．まずデータが第1象限にあるように(すべての$x, y \geq 0$)正規化し，次に記述が反時計回りになるようにしよう．

```
(define (norm vs)                                        ;正規化プログラム
  (let ((xmin (apply min (map car vs)))                  ;xの最小値
        (ymin (apply min (map cdr vs))))                 ;yの最小値
    (set! vs                              ;最小値を引いたリストを構成
      (map (lambda (v) (cons (- (car v) xmin) (- (cdr v) ymin))) vs)))
    (let* ((x0 (apply min                                ;y=0のもので最小のx
             (map car (filter (lambda (v) (= (cdr v) 0)) vs))))
           (tail (member (cons x0 0) vs))                ;(x0 . 0)で始まるリスト
           (l0 (length vs))
           (l1 (length tail)))
      (if (> l0 l1)
        (begin
          (set-cdr! (list-tail vs (- l0 l1 1)) '())
          (set-cdr! (list-tail tail (- l1 1)) vs)
          (set! vs tail)))                               ;tailを前にもってくる
      (if (> (cdadr vs) 0)                                     ;時計回りなら逆転
        (set! vs (cons (car vs) (reverse (cdr vs)))))
      vs))                                                     ;normの終わり
```

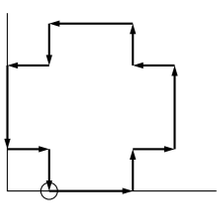
渡された座標の列をvsとする．xmin, yminはそれぞれの座標の最小値．次のmapでそれぞれの座標から最小値を引き，正規化する．次にy座標が0のもので，xが最小の値をx0とし，(x0 . 0)の要素を先頭とするリストtailを用意する．vs, tailの長さをそれぞれl0, l1とし，tailの後にvsのtailの直前までの部分を連結して，改めてvsとする．記述の回る順については，昨年12月のプロムナード「三角形の分割」にあったように，多辺形の面積の正負でも分かるが，今のように最下辺の最左点から出発するように正規化してあれば，最初の座標は(x0 . 0)，次の座標のyの値が0でないなら時計回りに記述してあることになる．(> (cdadr vs) 0)はそれを調べている．

eとfの正規化したものは次のとおり．

```
((1 . 0) (3 . 0) (3 . 3) (0 . 3) (0 . 1) (1 . 1))
((1 . 0) (3 . 0) (3 . 1) (4 . 1) (4 . 3) (3 . 3) (3 . 4) (1 . 4)
 (1 . 3) (0 . 3) (0 . 1) (1 . 1))
```

■文字列への変換

これで穴も蓋も最下段左端からの反時計回りの記述になった．次は1と0と?の文字列を構成する．点の記述が一番下の線の最左の点(○印)から並べ替えられている(図-5の左)から構成は簡単だ．



```
0 1 1 0
1 1 1 1
1 1 1 1
0 1 1 0
```

図-5

これから右のような文字列を作る．記述のy座標を小さい方から走査すると，各y座標の角が見つかるので，その相隣る2点のx座標が増加すればその間の文字列を1に，減少すればその間の文字列を0または?に置き換える．したがってそのプログラムは次のようになろう．

```
(define (make-str vs c)                                    ;文字列へ変換
  (let* ((xmax (apply max (map car vs)))
         (ymax (apply max (map cdr vs)))
         (ymin 0)
         (string (make-string xmax c))          ;長さxmaxの文字列 cで満たす
         (strings '()))                         ;文字列全体を作る場所
    (define (str-amend)
      (define (newstr f t c)             ;f(front)とt(tail)の間をcで満たす
        (string-append (substring string 0 f)
          (make-string (- t f) c) (substring string t xmax)))
      (define (separate l p)
        (if (null? l) (p '() '())
            (separate (cdr l)
              (lambda (edge rem)
                (let ((next (car l)))
                  (if (= (cdr next) ymin)
                      (p (cons next edge) rem)  ;最下行ならedgeに追加
                      (p edge (cons next rem))))))))    ;separateの終わり
      (separate vs                              ;最下行とそれ以外に分離
        (lambda (edge rem) (set! vs rem)
          (do ((i 0 (+ i 2))) ((= i (length edge)))     ;2個ずつとる
            (let ((p (car (list-ref edge i)))           ;前のx座標をp
                  (q (car (list-ref edge (+ i 1)))))    ;後のx座標をq
              (set! string (if (< p q) (newstr p q #\1) ;p-q間を1に変える
```

```
                                        (newstr q p c)))))))      ;q-p間をcに
      (set! ymin (apply min (map cdr vs))))      ;ymin更新 str-amendの終わり
    (do ((y 0 (+ y 1))) ((= y ymax))
      (if (= y ymin) (str-amend))
      (set! strings (cons string strings)))
    strings))                                                ;make-strの終わり
```

make-strの本体は最後の方doからの3行にある．長さxmaxの文字列をstringに用意する．文字列要素は引数cで貰ってきたもので，穴のときは文字?，蓋のときは文字0である．yを0からymaxまで変えながら，yがyminに等しければここで周囲が横方向を向くのでstr-amendを呼ぶ．いずれにしてもstringをstringsに追加する．

順が前後するが，str-amendは次のように動作する．まず残っている座標の列vsからy座標がyminに等しいものをedgeに取り出し，残りをremに置く．edgeのうち先頭から2個ずつをとり，そのx座標をp, qとし，$p < q$ならstringのpからqまでを文字1に，$p > q$ならqからpまでを文字cにする．

いまのところstringsは文字列の並びである．これを1列にする必要がある．それには蓋では改行のところに0を入れて連結し，穴では蓋の文字列の幅に揃えたうえ，改行の処理もする．その手続きは次のconcである．

```
(define (conc strings pad)       ;stringsの要素をpadを挟みながら連結
  (if (= (length strings) 1) (car strings)
    (string-append (car strings) pad (conc (cdr strings) pad))))
```

■文字パターン探索

蓋の文字列01100111101111001 1の中に穴の文字パターン111??111???11が含まれているか，調べなければならない．常識的には

```
011001111011110011
111??111???11 ← パターン
```

を並べ，先頭から比較する．この場合は0と1なので最初から失敗する．次はパターンを1字右へずらし，

```
011001111011110011
 111??111???11 ← パターン
```

で比較する．最左の2字11は一致するが，次が0と1で失敗，文字列の字数をm, パターンの字数をnとする．$n/2$字くらいで失敗するとし，比較のためにずらす回数は$m-n$なので，パターンが含まれない場合は$mn/2$の比較で判明する．これに対し，Boyer-Mooreの方法はパターンを右から調べる．

```
011001111011110011
111??111???11 ← パターン
cba9876543210 ← 16進法による文字位置
```

最下段はパターンの右から文字位置の16進表記である．パターンの?は相手が0でも1でも合うことにしているので，この場合は12桁目（文字列が0, パターンが1）で初めて失敗する．このときは，（後述の理由により）1字右へシフトし，さらに右から比較する．

```
011001111011110011
 111??111???11 ← パターン
 cba9876543210 ← 16進法による文字位置
```

今度は10桁目で合わない．ではまた1文字右へずらすか．しかしパターンの11桁目も1なので，そこで失敗するに決まっている．2文字ずらしても12桁目が1だからやはりだめである．ではどこまでずらすか

といえば，このパターンにはないが次に?のある位置までの距離だけずらすのである．それは13桁目だから3文字ずらす．

```
011001111011110011
  111??111???11 ← パターン
  cba9876543210 ← 16進法による文字位置
```

そうすると先ほどの0が1と比較されることはなくなる．ただ今度は1桁目で不一致が生じる．このときはすぐ左が?なので，1文字ずらす．つまり要するにパターン中の各1の文字に対し，左側で次の?は何字先にあるか，あらかじめ表をこしらえておけば，ずらす文字数は見つかるのである．いまのパターンなら

c	b	a	7	6	5	1	0
1	2	3	1	2	3	1	2

をこしらえておけばよい．パターンが1の各位置について，左の?までの距離の表である．この表は以下のプログラムではddに用意する．ddにできる表は

```
((12 . 2) (11 . 1) (7 . 3) (6 . 2) (5 . 1) (2 . 3) (1 . 2) (0 . 1))
```

プログラムではmatch0の引数が減っていくようになっているので，上の表とは左右反対になっている．

```
(define (covered? h c)
  (let ((hl (length h)) (cl (length c)) (dd '()) (q -1))
    (define (match d)            ;パターンをd文字ずらして一致をみる
      (define (match0 i)         ;パターンのi文字目は一致?(右から左)
        (cond ((< i 0) '())
              ((= (+ (list-ref h i) (list-ref c (+ i d))) 97) i)
              (else (match0 (- i 1)))))          ;1文字左へ match0の終わり
      (if (<= d (- cl hl))
          (let ((ml (match0 (- hl 1))))          ;不一致の文字位置をmlへ
            (if ml (match (+ d (cdr (assoc ml dd))))     ;ずらす幅を得る
              #t))
          '()))                                          ;matchの終わり
    (do ((i 0 (+ i 1))) ((= i hl))               ;ddを用意するループ
      (if (= (list-ref h i) 63) (set! q i)
          (set! dd (cons (cons i (- i q)) dd))))
    (match 0)))
```

match0は全部見たらnilを，不一致を見つけたらその位置を返し，一致のときは次の文字対を調べる．matchはずらしている最中ならmatch0を呼び，位置の整数が返っていれば，ddを見て次のずらす位置をとり，nilなら#t(すなわち真)を，ずらし終わっていれば()(すなわち偽)を返す．match0で，2つの文字のコードを足し，97との比較は，hの文字が0，cの文字が1であることを見ている．

■ランレングス方式

この上の部分まで書いて，プロムナード同人に見せたら文句がきた．このやり方だと，たとえば，座標の幅と高さが10億×10億だったりすると，長さ10^{18}，つまり百京の文字列を作ることにならないかというのである．まぁそんなに大きな穴や蓋なら，それにくらべたら1や10の幅は計測誤差みたいなものだから，いい加減なスケールでやればいいとは思ったものの，そこはディジタル計算機だからなんとかする必要もある．

とりあえず文字列の表現と1の文字数だけある連想記憶方式のスキップ距離の表をなんとかしなければならない．文字列の方は幸い1と0，1と?の列だから，ランレングスにすることにした．

つまり011001111011110011を(1 2 2 4 1 4 2 2)と表現する．実際には先頭と末尾の0は関係な

いから捨て，1から始まるランレングスにする．上の例は(2 2 4 1 4 2 2)となる．先頭の0からのランレングスのときは，1が0個あると思い，(0 1 2 2 4 1 4 2 2)とする．考えてみれば穴の場合も？にすることはなかったので，normのあとのmake-strに相当するmake-segは引数1個にした．

```
(define (make-seg vs)
  (let* ((xmax (apply max (map car vs)))
         (ymax (apply max (map cdr vs)))
         (ymin 0) (seg '()) (segs '()))
  (define (newseg p q)
    (define (addseg p q seg)            ;p,q間が1のセグメントと追加する
       (cond ((null? seg) (list (cons p q)))
             ((< p (caar seg)) (cons (cons p q) seg))
             (else (cons (car seg) (addseg p q (cdr seg))))))
    (define (unify seg)          ;セグメントの隣同士が同じならまとめる
      (cond ((< (length seg) 2) seg)
            ((= (cdar seg) (caadr seg))
              (unify (cons (cons (caar seg) (cdadr seg)) (cddr seg))))
            (else (cons (car seg) (unify (cdr seg))))))
    (define (subseg p q seg)            ;(-.q) (p.-)とセグメントを分割する
      (cond ((null? seg) seg)
            ((and (<= (caar seg) q) (<= p (cdar seg)))
             (cons (cons (caar seg) q) (cons (cons p (cdar seg))
                                             (cdr seg))))
            (else (cons (car seg) (subseg p q (cdr seg))))))
    (define (cancel seg)                ;長さ0のセグメントを削除する
      (cond ((null? seg) seg)
            ((= (caar seg) (cdar seg)) (cancel (cdr seg)))
            (else (cons (car seg) (cancel (cdr seg))))))
    (if (< p q) (set! seg (unify (addseg p q seg)))
                (set! seg (cancel (subseg p q seg)))))
  (define (separate l p)                          ;文字列版のseparateと同様
    (if (null? l) (p '() '())
        (separate (cdr l)
          (lambda (edge rem)
            (let ((next (car l)))
               (if (= (cdr next) ymin)
                   (p (cons next edge) rem)
                   (p edge (cons next rem))))))))       ;separateの終わり
  (do ((y 0 (+ y 1))) ((= y ymax))
    (if (= y ymin)                                  ;最下行の座標があれば
      (begin                                ;それをedgeに，その他をremにいれ
        (separate vs
          (lambda (edge rem) (set! vs rem)              ;remをvsにもどす
            (do ((i 0 (+ i 2))) ((= i (length edge)))   ;座標を2個ずつとり
              (let ((p (car (list-ref edge i)))         ;前のx座標をp
                    (q (car (list-ref edge (+ i 1)))))  ;後のx座標をq
                (newseg p q)))))                        ;としnewsegで処理
        (set! ymin (apply min (map cdr vs)))))          ;yminを更新する
    (set! segs (cons seg segs)))
  segs))                                            ;make-segの終わり
(define (maxx ss)               ;ドット対のリストのリストからxのmaxを得る
  (apply max (map
               (lambda (s) (apply max (map (lambda (x) (cdr x)) s))) ss)))
(define (append-seg segs width) ;segsから1行がwidth幅のランレングスを作る
  (let ((last-x '()) (run '())) ;last-xは直前のセグメントの右端の座標
    (for-each (lambda (seg)                         ;1行分をsegにとり，
      (for-each (lambda (pair)                      ;1対の座標をpairにとる
        (if last-x (set! run (cons (- (car pair) last-x) run)))
        (set! run (cons (- (cdr pair) (car pair)) run))
        (set! last-x (cdr pair))) seg)
```

```
      (set! last-x (- last-x width))) segs)
    run))                                                    ;append-segの終わり
```

たとえば図-2 fの正規化した座標列

```
((1 . 0) (3 . 0) (3 . 1) (4 . 1) (4 . 3) (3 . 3)
 (3 . 4) (1 . 4) (1 . 3) (0 . 3) (0 . 1) (1 . 1))
```

に対し，make-segでは文字列版と同様にソートし，同じ横軸ごとの1の両端の座標のドット対のリストのリスト，segsが得られる．

```
(((1 . 3)) ((0 . 4)) ((0 . 4)) ((1 . 3))))
```

次にmaxxが1の最大幅を計算する（eでは3, fでは4）．そして最後のappend-segがランレングス(2 2 4 1 4 2 2)を作る．図と見比べると最初の2は図-2 fの一番上の1の長さ，次の2はその左の0の長さ＋行末分の1，4は上から2段目の1の長さ，次は行末，3段目の1，行末＋4段目の右の0の長さ，そして最後が4段目の1の長さである．穴に相当するランレングスは(3 2 3 2 2)である．fの幅に揃えてあるから，行末が伸びている．

hとcからそのランレングスhr, crは以下のように作る．

```
(let* ((hv (readvs h)) (cv (readvs c))                       ;データ入力
       (hs (reverse (make-seg (norm hv))))       ;正規化してセグメントにする
       (cs (reverse (make-seg (norm cv))))
       (hl (maxx hs)) (cl (maxx cs))                         ;x座標の最大値
       (hr (append-seg hs (+ cl 1)))                         ;ランレングスにする
       (cr (append-seg cs (+ cl 1)))) )
```

Boyer-Mooreは文字列のある範囲の終わりの方から比較するから，ランレングスをある範囲で切り出す必要がある．それが次の手続き(cut n c)で，ランレングスcからn文字分を切り出す．上のランレングスから12文字，15文字，17文字で切り出し，逆転すると下のようになる．

```
12  (3 1 4 2 2)
15  (0 2 4 1 4 2 2)
17  (2 2 4 1 4 2 2)

(define (cut n c)                          ;ランレングスcからn文字分切り出す
  (define (ct n c)                ;0の長さから始まるランレングスの処理
    (if (>= (car c) n) (list n 0)        ;0が十分長い，1の長さを0として追加
        (let ((a (car c)))
          (cons a (cut (- n a) (cdr c))))))                  ;ctの終わり
  (if (>= (car c) n) (list n)                          ;最初の1が十分長い
    (let ((a (car c)))
      (cons a (ct (- n a) (cdr c))))))                       ;cutの終わり
```

これらと穴のランレングスを逆転したものを，先頭から比較するのである．それには与えられたランレングスのある文字目から探して最初の1の位置，最初の0の位置を返す手続きnext1, next0を用意する．

```
(define (next1 r n)                                ;ランレングスrのn文字目
  (define (nx1 r n c)                              ;からの最初の1の位置を返す
    (let ((rc (+ (car r) c)))
      (if (< n rc) n
          (if (null? (cdr r)) '()                  ;範囲を超えた nilを返す
            (let ((rrc (+ rc (cadr r))))
              (if (< n rrc) rrc (nx1 (cddr r) n rrc)))))))
  (nx1 r n 0))                                             ;next1の終わり
(define (next0 r n)                                ;ランレングスrのn文字目
  (define (nx0 r n c)                              ;からの最初の0の位置を返す
    (let ((rc (+ (car r) c)))
```

```
      (if (null? (cdr r)) '()                    ;範囲を超えた nilを返す
          (if (< n rc) rc
            (let ((rrc (+ rc (cadr r))))
              (if (< n rrc) n (nx0 (cddr r) n rrc)))))))
  (nx0 r n 0))                                          ;next0の終わり
```

文字列をランレングスにすると同時にBoyer-Mooreの不一致を見つけたときのずらす文字数の表も，前のような連想リストにするのではなく，右から見て次の0の位置までのリストで表す．つまり図-2のeに対しては(2 7 12)とする．この変更のせいで，make-ddとnextを用意する．

```
(define (make-dd h)             ;ランレングスhに対するddのリストを作る
  (if (null? (cdr h)) (list (car h))
      (cons (apply + h) (make-dd (cddr h)))))           ;make-ddの終わり
(define (next j dd)             ;ddを見てjから最初の0の位置を知る
  (- (car (filter (lambda (x) (> x j)) dd)) j))         ;nextの終わり
```

さて中心のcovered?は次のようになる．

```
(define (covered? h c)
  (let ((hl (apply + h)) (cl (apply + c))
        (dd (reverse (make-dd h))) (hr (reverse h)))
    (define (match i)
      (if (> i (- cl hl)) '()           ;パターンが文字列を飛び出した
          (let ((cc (reverse (cut (+ hl i) c))) (j 0) (k 0))
            (define (search)
              (set! j (next0 cc k))
              (if j (begin (set! k (next1 hr j))
                           (if k
                              (if (= j k)
                                  (match (+ i (next j dd)))
                                  (search))) #t)) #t))       ;searchの終わり
          (search))))                                       ;matchの終わり
  (match 0)))                                               ;covered?の終わり
```

簡単に説明すると先頭のletで，穴と蓋のランレングスの長さhlとclを計算し，ddを用意し，穴のランレングスを逆転したhrを作る．そして先頭からi文字目から一致を調べる(match i)がある．

(match i)はパターンをi文字ずらして文字列との一致を見る．まずパターンが文字列の範囲内にあることを確かめ(i < cl - hl)，cからhl + i文字を切り出し逆転してccとする．ccの0を先頭から探し，その位置をjとする．hrの1をjから探し，その位置をkとする．j = kなら不一致があったので，nextを使い，ずらす数dを得，(match (+ d i))を実行．k > jならccの最初の0をkから探し，同様な探索を繰り返す．jやkがnilだったら，そういう文字はないわけで，終わりまで探索したことになる．

世界大会の問題はサンプル以外のテストデータが得られないので，自分で作ったテストデータに頼るよりしかたがない．いちおう意地悪そうなデータでも動くようだが，時間のかかりそうなデータを作ったりはしていないので，どの程度の速さで計算できるかはよく分からない．こうしてみると審判団が苦労して用意するデータのありがたみを実感する．

(平成15年10月27日受付)