三角形の分割

和田 英一（IIJ 技術研究所）
wada@u-tokyo.ac.jp

■区画内の点の個数

図 -1 に示すように二等辺直角三角形 ABC の内側に点が n 個散在している（n は 3 の倍数）．どの点も各頂点からその点に引いた線分やその延長線上に他の点が載らないようにしてある．図で座標が整数値なのは他意はない．今の条件を調べるのを楽にするためである．

その三角形の内部にもう 1 つの点 Q をたとえば (3, 3) にとり，**図 -2** のように各頂点から Q へ線を引いて三角形を 3 つの区画に分割する．点が区画の境界にあれば，それにかかわる区画のいずれにも含まれるとする．そしてどの区画にも $n/3$ 以上個の点が含まれるようにしたい．その Q の座標を答えるのが今回の問題である．1998 年，東京大会（@ 早稲田大学）の問題 F（問題文は http://www.isse.kuis.kyoto-u.ac.jp/acm-japan/contest.html 参照）．

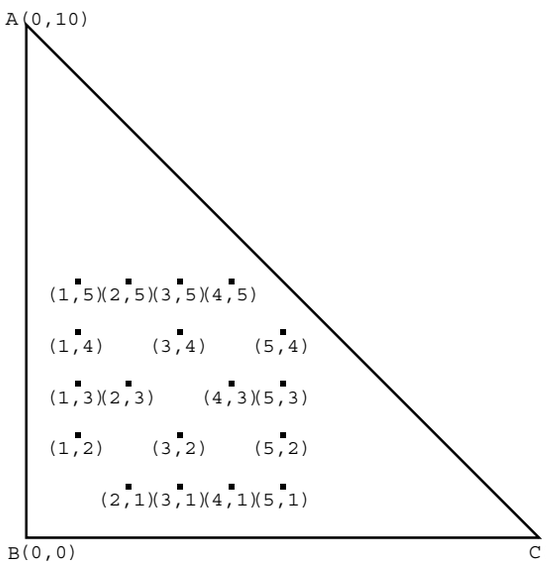

図-1

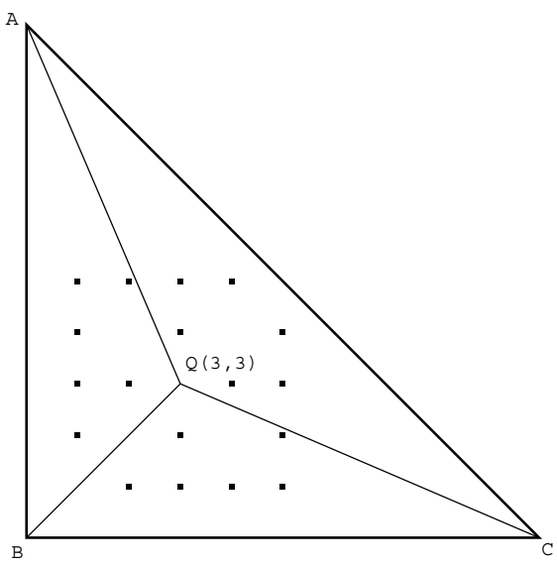

図-2

図 -2 から察せられるように，Q の位置には任意性があり，その Q の位置が各区画に本当に $n/3$ 以上の点が含まれるように分割するか，審判団は計算してチェックしたらしい．**図 -3** の 3 点 P_0, P_1, P_2 のようにそれぞれ各辺の中点のわずかに内側にあれば，人間到る処青山あり（どこにでも骨を埋められるの意）の気分で，ほとんどどこもが Q の候補地である．正確にいえば各頂点からその角を挟む辺の中点付近の点に引いた線で囲まれた 6 角形 (P_0, Q_2, P_1, Q_0, P_2, Q_1) の中が解の存在領域である．

区画内の点の個数だが，3 点 $P_0(x_0,y_0)$, $P_1(x_1,y_1)$, $P_2(x_2,y_2)$ が構成する△ $P_0P_1P_2$ の面積 S は行列式を使うと

$$S = \frac{1}{2}\begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$

である．ただし $P_0 \to P_1 \to P_2$ が反時計回りか時計回りかでこの値は正か負になる（忘れたら (0, 1), (0, 0), (1, 0) の程度の三角形で検算する）．これを利用するが正負の情報だけ必要ゆえ，先頭の 1/2 は今は使わない．3 点が 1 直線上にあれば行列式の値は 0 になる．分割点を Q とし，内部の点 P_i に対して，△ AQP_i, △ BQP_i, △ CQP_i の面積を計算し，その符号をこの順に左から書くと**図 -4** になる．これから分かるように△ BCQ の区画内にある点 P は B→Q→P が時計回りなので負，C→Q→P が半時計回りなので正になっている．一番左の符号は AQ に対して P が左にあるか右にあるかで負か正になり，この区画では問題にならない．これを `?-+` と表す．△ CAQ 内の点 P は `+?-`，△ ABQ 内の点 P は `-+?` となっていることが分かる．したがって Q が与えられたら，今のような符号の組を作り，`?-+`, `+?-`, `-+?` の個数を調べれば各区画内の点の個数が判明する．

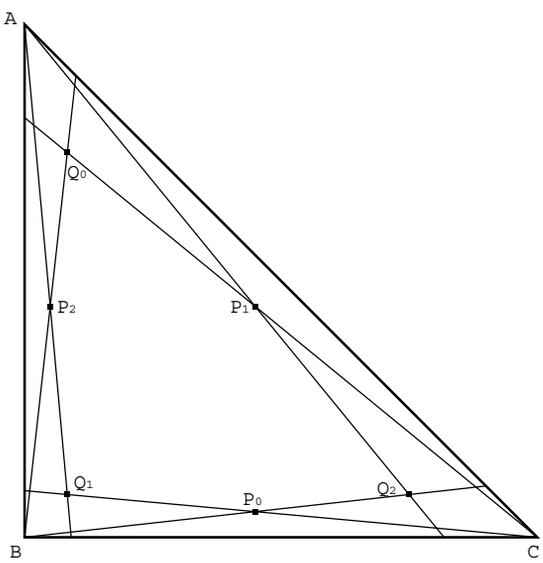

図-3

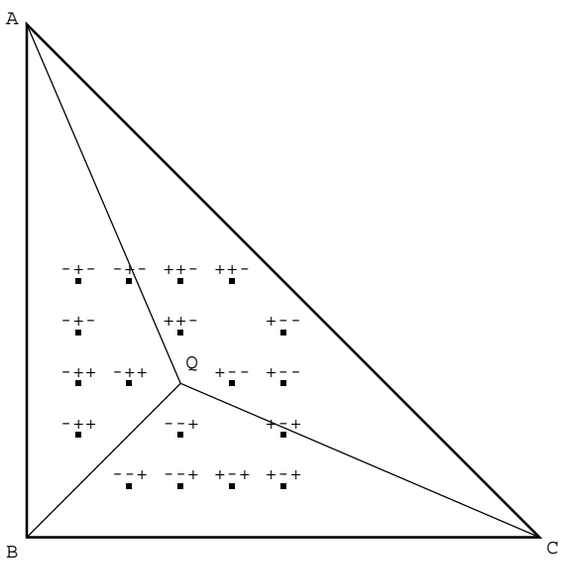

図-4

今回のプログラムは lisp 方言の scheme で書きたい．点の座標の並びなど，C 言語の配列で書いたり，C 言語でリスト処理を書いて使ったりするより分かりやすいからである．

まず下請け関数．crossp は点 p_0, p_1 を結ぶ直線と点 p_2, p_3 を結ぶ直線の交点の座標を返す．座標は x と y のドット対．どうせ傾いている直線なので，気楽に書いてある．side は p_0, p_1, p_2 が反時計回りか時計回りか 1 直線上にあるか判定し，1, −1, 0 のいずれかを返す．count は△ BCq_0, △ CAq_1, △ ABq_2 内にある点の個数をこの順にリストにして返す．リストの要素がすべて $n/3$ 個以上あれば分割成功である．

```
(define (crossp p0 p1 p2 p3)
  (let* ((x0 (car p0)) (y0 (cdr p0))
         (x1 (car p1)) (y1 (cdr p1))
         (x2 (car p2)) (y2 (cdr p2))
         (x3 (car p3)) (y3 (cdr p3))
         (a0 (- y1 y0)) (b0 (- x0 x1))
         (c0 (- (* x0 y1) (* x1 y0)))
         (a1 (- y3 y2)) (b1 (- x2 x3))
         (c1 (- (* x2 y3) (* x3 y2)))
         (d (- (* a0 b1) (* a1 b0))))
    (cons (/ (- (* c0 b1) (* c1 b0)) d)
          (/ (- (* a0 c1) (* a1 c0)) d))))
```

```
(define (side p0 p1 p2)
 (let* ((x0 (car p0)) (y0 (cdr p0)) (x1 (car p1)) (y1 (cdr p1))
        (x2 (car p2)) (y2 (cdr p2))
        (det  (- (+ (* x0 y1) (* x1 y2) (* x2 y0))
                 (+ (* x2 y1) (* x0 y2) (* x1 y0)))))
   (cond ((> det 0) 1)
         ((< det 0) -1)
         (else 0))))

(define (count q0 q1 q2)
   (let* ((sl (map (lambda (p)
                     (list (side (list-ref corners 0) q0 p)
                           (side (list-ref corners 1) q1 p)
                           (side (list-ref corners 2) q2 p))) points)))
    (list
      (length (filter (lambda (x) (and (<= (list-ref x 1) 0)
                                       (>= (list-ref x 2) 0)))
                      sl))       ;(? - +) 下の△内の個数
      (length (filter (lambda (x) (and (<= (list-ref x 2) 0)
                                       (>= (list-ref x 0) 0)))
                      sl))       ;(+ ? -) 中の△内の個数
      (length (filter (lambda (x) (and (<= (list-ref x 0) 0)
                                       (>= (list-ref x 1) 0)))
                      sl)))))    ;(- + ?) 左の△内の個数
```

corners という変数が見えるが，点や頂点は

```
(define corners '((0 . 10) (0 . 0) (10 . 0)))
(define points '((5 . 4) (4 . 5) (5 . 3) (3 . 5) (5 . 2) (4 . 3)
                 (3 . 4) (2 . 5) (5 . 1) (1 . 5) (4 . 1) (3 . 2)
                 (2 . 3) (1 . 4) (3 . 1) (1 . 3) (2 . 1) (1 . 2)))
```

のように与えることにする．

■山登り法を真似る

各区画内の点の個数の算出法が分かれば，とりあえず，溺れた場合に藁をも掴むが如き方法は考えられる．

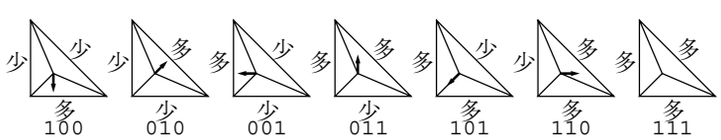

図-5

まず適当に分割中心を仮りに決め，分割を試み，その結果によって分割中心を移動修正するという方策を考える．**図-5** で「多」は $n/3$ 以上の点を含む区画,「少」は $n/3$ 未満の点を含む区画を表す．「多」「少」をそれぞれ 1, 0 で表し，区画を下，中（右上のこと），左の順に並べた 2 進数のような表現が下に記してある．

かくして 111 になれば完成となる．さもなければ，矢印が示す方向に分割中心を移動する．移動の量やその変更が秘妙なノウハウであって，さしたる自信はない．山登り法などと同様に私はこれを「東洋医学」ふうだと思っている．

今書いてあるプログラムは最初の移動量が直角を挟む辺の長さの 1/10 で，向きが変わるたびに 0.8 倍

しているが，必ずしもうまくいくという保証もない．

図 -6 に中心が移動する一例を示す．こういう方法で分散した点を含む最小の円の中心を求めるような場合は最後の精度までつめよらなければならないが，この問題は適当な範囲で 3 分割に成功するから，そういう意味では楽かもしれない．

私のプログラムでは図 -6 の不思議な出発点ではなく，すべての点の位置の平均の位置から開始した．図 -1 のような対称的なデータだと平均の位置が (3, 3) になるから一発で的中する．審判団の用意したデータに，辺の長さ 1000 で点の位置が (1,1), (2,1), ...,(300,1) といういやらしいのがあったが，600 数十ステップで決着した．

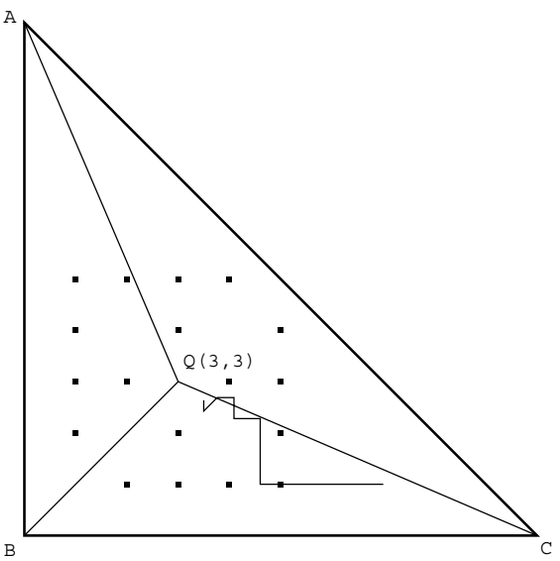

図-6

```
(define (triangle)
  (let ((qx (/ (apply + (map car points)) (length points)))      ;x,y の初期値
        (qy (/ (apply + (map cdr points)) (length points)))
        (th (/ (length points) 3))       ; 閾値
        (dx (/ (cdar corners) 10))       ; 移動量
        (dy (/ (cdar corners) 10))
        (mode0 0)
        (mode1 0))
  (define (advance)
    (define (modify mode)
      (cond
      ((= mode 1) (set! qx (max (- qx dx) 0.0)))         ;mode の値は図 -5 と同じ
      ((= mode 2) (set! qx (+ qx dx)) (set! qy (+ qy dy)))
      ((= mode 3) (set! qy (+ qy dy)))
      ((= mode 4) (set! qy (max (- qy dy) 0.0)))
      ((= mode 5) (set! qx (- qx dx)) (set! qy (- qy dy)))
      ((= mode 6) (set! qx (+ qx dx)))))

   (define (dist count th)       ;2 進→ 10 進変換
     (let ((dist (map  (lambda (x) (if (< x th) 0 1)) count)))
       (+ (* (car dist) 4) (* (cadr dist) 2) (caddr dist))))
                                                          ;advance の本体ここから
    (set! mode1 (dist (count (cons qx qy)(cons qx qy)(cons qx qy)) th))
    (cond ((= mode1 7) (cons qx qy))     ; うまく分割できたら座標を返す
          ((< mode1 7)
           (cond ((not (= mode1 mode0)) (set! mode0 mode1)
```

```
                                (set! dx (* dx 0.8))     ; 移動量を減らす
                                (set! dy (* dy 0.8))
                                (modify (- 7 mode0))))   ; 逆に 1 回動く
            (modify mode1)      ; 新しい方向に動く
            (advance))))        ; 末尾再帰でループする
  (advance)))
```

最初の let で変数を初期化し，あとは advance でループしている．どの方向に進んでいるかは mode が覚えている．

■全数検査でやると

こういう東洋医学手法は信念に合わないというときは確実に答えに至るアルゴリズムで解きたくなるであろう．

それには次の方法を考えつく．

1. すべての点が分割点になり得ないか確かめる．
2. すべての相異なる 2 点に頂点から引いた線の延長上の交点（図 -3 の Q のような点）が分割点になり得ないか確かめる．

1. は点の個数に対してオーダ n であり，2. はオーダ n^2 である．1. では分割点になる点を提供した区画はちょうど $n/3$ 個含むことになるが，あとの 2 区画は 1 個ずつ余徳がつく．これは区画内個数算出の手続きが用意できれば，プログラムは簡単だ．

2. には解決すべきことがまだある．まず 2 点を p_0, p_1 として，それぞれにどの頂点から線を引くかだ．答えは（v_0, $v_1 \in \{A,B,C\}$, p_0, $p_1 \in \{P_0,P_1,\cdots, P_{n-1}\}$ として）v_0 から p_0, v_1 から p_1 に線を引くとき，p_1 は v_0p_0 に対して v_1 側にあり，p_0 は v_1p_1 に対して v_0 側にあるべし．

A から p_0 に線を引くとする．p_1 が Ap_0 に対し B 側にあれば p_0 が Bp_1 に対し A 側にあればよい．つまり，$\triangle Ap_0p_1 < 0$ かつ $\triangle Bp_1p_0 > 0$ このときは Ap_0, Bp_1 を引く．反対に，p_1 が Ap_0 に対し C 側にあれば p_0 が Cp_1 に対し A 側にあればよい．つまり，$\triangle Ap_0p_1 > 0$ かつ $\triangle Cp_1p_0 < 0$ このときは Ap_0, Cp_1 を引く．B や C から p_0 に線を引く場合も同様に考えればよい．面積を 6 種類計算しているみたいだが，$\triangle vp_0p_1 = -\triangle vp_1p_0$ が利用できることに注意しよう．

次は，各区画内の点の個数を調べるには，交点の座標ではなく，頂点からの勾配だけを知ればよいから，たとえば**図 -7** で，Ap_0 と Cp_2 の延長線の交点 D へ BD を引いたとき，その勾配を，D の座標を明示的に計算せずに得る方法の開発である．

図 -7 をみて考えよう．E, G は H からそれぞれ AB, BC に下ろした垂線の足，F, H はそれを逆に伸ばして斜辺 CA との交点．

この図では 3 点 p_0, p_1, p_2 の座標は $p_0(a, 10-b)$, $p_1(c, d)$, $p_2(10-e, f)$ のつもり．したがって $a, b, \ldots, f$（それぞれ > 0, また $b > a$, $e > f$）は図に示すような長さになる．

この図で重要なのは，直角二等辺三角形ゆえ，DF＝DH ということ．さらに ED : DF＝a : $b-a$, GD : DH＝f : $e-f$ である．この a, b と e, f のスケール比は $e-f$: $b-a$, 勾配は (a, b スケールの a):(e, f スケールの f) である．$\therefore$ ED : GD＝c : d＝$a(e-f)$: $f(b-a)$.

次に Ap_0, Bp_1 が先に与えられて，Cp_2 の勾配 e : f が欲しい場合はどうか．

$cf(b-a)=ad(e-f)$ だから $e-f=(b-a)cf/ad$, $(e-f)/f=e/f-1=(b-a)c/ad$,

$\therefore e : f=(b-a)c+ad : ad$.

Bp_1, Cp_2 が先なら a : b は $b-a=(e-e)ad/cf$, $(b-a)/a=b/a-1=(e-f)d/cf$,

$\therefore a : b=cf : (e-f)d+cf$.

いま , $a=2$, $b=5$, $c=2$, $d=3$, $e=6$, $f=3$ でチェックすると

$c : d = 2(6-3) : 3(5-2) = 2\times3 : 3\times3 = 2 : 3$,

$e : f = (5-2)\times 2+2\times3 : 2\times3 = 3\times2+6 : 6 = 12 : 6 = 2 : 1$,

$a : b = 2\times3 : (6-3)3+2\times3 = 6 : 3\times3+6 = 2 : 5$,

これを使えば交点をいちいち計算しないで済むが，問題に答えるには最後にどうしても交点の計算がいる．2点 (p_0, p_1) と2点 (p_2, p_3) を通る直線の交点を計算する cross を使う．しかしこういう工夫をしても，計算時間はかなりかかり，予定時間を超えると打ち切られる恐れがある．

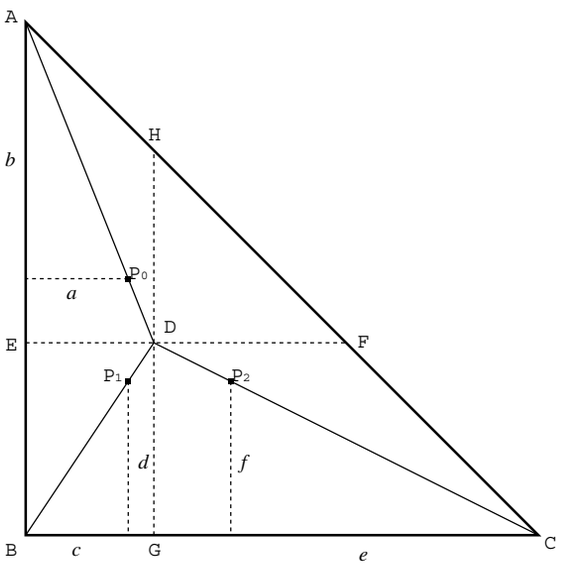

図-7

```
(define (triangle)
  (let* ((a (list-ref corners 0)) (b (list-ref corners 1))
         (c (list-ref corners 2)) (pleng (length points)) (plen3 (/ pleng 3))
         (plen1 (- plen3 1)) (lat (cdar corners)))

  (define (test1 p)                          ; 点 p がうまく 3 分割するかテスト
    (count p p p))

  (define (choosecorners p0 p1)              ;p0, p1 がどの頂点と対応するか
    (let ((s0 (side a p0 p1)) (s1 (side b p0 p1)) (s2 (side c p0 p1)))
      (cond ((and (< s0 0) (< s1 0)) (cons 0 (list p0 p1 '()))) ;corner への点
            ((and (> s0 0) (> s2 0)) (cons 1 (list p0 '() p1)))
            ((and (< s1 0) (< s2 0)) (cons 2 (list '() p0 p1)))
            ((and (> s1 0) (> s0 0)) (cons 0 (list p1 p0 '())))
            ((and (< s2 0) (< s0 0)) (cons 1 (list p1 '() p0)))
            ((and (> s2 0) (> s1 0)) (cons 2 (list '() p1 p0))))))

  (define (test2 p0 p1)                      ;2 点の延長線の交点が 3 分割するか
    (let* ((corner (choosecorners p0 p1)) (ci (car corner))(cc (cdr corner)))
      (cond ((= ci 0)
             (let* ((a (car (list-ref cc 0))) (b (- lat (cdr (list-ref cc 0))))
                    (c (car (list-ref cc 1))) (d (cdr (list-ref cc 1)))
                    (e (+ (* (- b a) c) (* a d))) (f (* a d)))
               (count (cons a (- lat b)) (cons c d) (cons (- lat e) f))))
            ((= ci 1)
             (let* ((a (car (list-ref cc 0))) (b (- lat (cdr (list-ref cc 0))))
                    (e (- lat (car (list-ref cc 2))))
                    (f (cdr (list-ref cc 2)))
```

```
                (c (* a (- e f))) (d (* f (- b a))))
           (count (cons a (- lat b)) (cons c d) (cons (- lat e) f))))
        ((= ci 2)
         (let* ((c (car (list-ref cc 1))) (d (cdr (list-ref cc 1)))
                (e (- lat (car (list-ref cc 2))))
                (f (cdr (list-ref cc 2)))
                (a (* c f)) (b (+ (* (- e f) d) (* c f))))
           (count (cons a (- lat b)) (cons c d) (cons (- lat e) f)))))))

(define (dist count th)
  (and (>= (car count) th) (>= (cadr count) th) (>= (caddr count) th)))
  (call-with-current-continuation     ; ループから飛び出す準備
    (lambda (exit)
      (do ((i 0 (+ i 1))) ((= i pleng))
        (let ((p (list-ref points i)))
          (cond ((dist (test1 p) plen3)
                 (exit p)))))          ;p が分割した．飛び出す．
      (do ((i 0 (+ i 1))) ((= i (- pleng 1)))
        (let ((p0 (list-ref points i)))
          (do ((j (+ i 1) (+ j 1))) ((= j pleng))
            (let ((p1 (list-ref points j)))
              (cond ((dist (test2 p0 p1) plen3)       ;p0 p1 でテスト
                     (let* ((corner (choosecorners p0 p1))
                            (ci (car corner)) (cc (cdr corner)))
                       (exit (cond ((= ci 0)          ; もう一度交点を計算
                                    (crossp a (car cc) b (cadr cc)))
                                   ((= ci 1)
                                    (crossp a (car cc) c (caddr cc)))
                                   ((= ci 2)
                                    (crossp b (cadr cc) c (caddr cc))
))))))))))))))
```

test1 は各点が分割中心になるかを調べ（1. の場合），test2 は p_0, p_1 とそれぞれの対応する頂点を結ぶ直線の交点が分割点になるかを調べ（2. の場合）ている．ループの途中で脱出したいので，call-with-current-continuation を使っているが，それに続くラムダ式の変数（ここでは exit）を呼ぶと脱出する．test2 では脱出してから交点を計算する．

■敵は幾万ありとても

東洋医学ふうではなくなったが，300 点もあるとオーダ n^2 はやはり苦しい．無駄な計算をできるだけ省き，解に向かって進みたいと思って工夫したのがこの算法である．**図 -8** で説明する．
図中の各点に β，γ という値を配する．点を P として β は ∠ CBP，γ は∠ ACP である．

2 つのスィーパ，C を中心に右回転する Sc と，B を中心に右回転する Sb がある．最初 Sc は CB に，Sb は BA に一致している．Sb と BC のなす角を S_β，Sc と CA のなす角を S_γ と呼ぶことにする．S_β の初期値は 90 度，S_γ は 45 度である．

Sc は右回転を始め，点を $n/3$ 個通過した時点で停止する（図 -8 では C から放射線状に出ている線の一番下）．この点の γ の値を S_γ として，また点の番号を γ インデックスとして記録する．次に Sb を右回転し，S_γ の線より下にある最初の点を探す（図では P_{17} に到達し，B から放射線状に出ている線の一番左にいることになる）．この点の β を S_β として，点の番号を β インデックスとして記録する．

この時点で B と β インデックスの点，C と γ インデックスの点を結ぶ線の延長線上の交点は $n/3$ 個の点を含む．

次に Sc をさらに右回転し，S_β より小さい β を持つ最初の点を探す（図では P_{12} が見つかる．C からの

放射状の下から2番目）．この点のγをS_γとして，γインデックスとともに記録する．今度はSbの右回転を続け，S_γより小さいγを持つ最初の点を探す（図ではP_{12}が見つかる）．この点のβをS_βとし，Bインデックスとともに記録（このように2つの点の延長線上の交点ではなく，1つの点になることもある）．

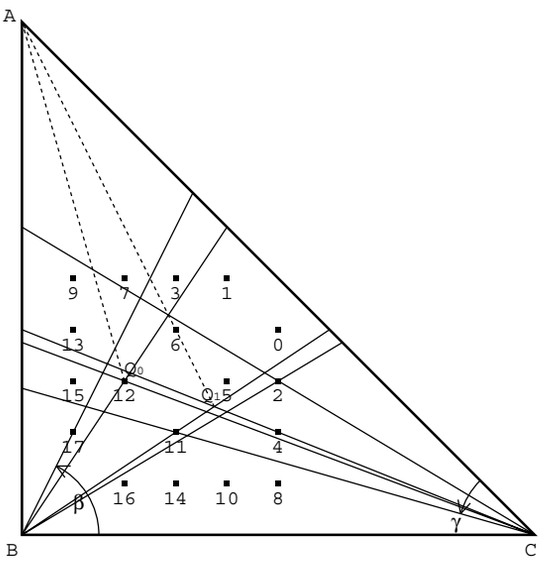

図-8

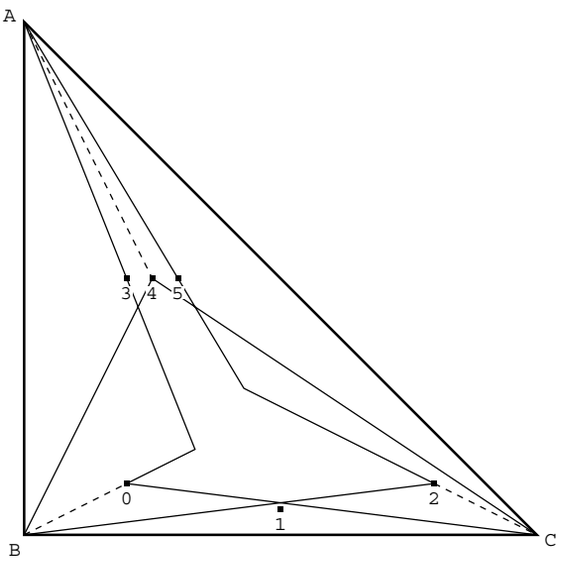

図-9

この作業をScがCAに到達するまで繰り返す．図でいえば4本のSbと4本のScが得られる．Sbの右からi番目とScの下からi番目が対になり，この各対で囲まれた区画は$n/3$個の点が含まれる．この各対の交点で三角形を3つに分割した場合，各区画に$n/3$個以上の点が含まれれば，分割に成功したことになる．この方法ではScは点のγ値の順にソートしたリストを，γ値の大きい方から小さい方へ順に1回辿るだけで作業が終るので，これは一見オーダnの算法に見えるが，中でソートを使っているので，それよりは悪い．

しかし待てよ，と思わぬでもない．もう一度図-8を見よう．$n/3$個の点を含む2つのスィーパの組の相隣る交点，たとえばP_{12}（Q_0としよう），それとP_{11}とP_4の延長線上の交点（Q_1としよう）に着目する．それらの点とスィーパの軸でなかった頂点（今はA）を結ぶ図の2本の破線の間の領域に多くの点が散在し，Q_0を選ぶとAQ_0の右には多数の点が含まれ，次にAQ_1を選ぶと，今度はその分割線の左に多くの点が含まれ，ちょうど半々にならないのではないかという疑問は当然出てくる．

図-9の点は(2, 1), (5, 0.5), (8, 1), (2, 5), (2.5, 5), (3, 5)に取ってある．辺BCの上に2点を含む三角形を作ると$\triangle BP_0C$, $\triangle BP_2C$ができる（2本の線が重なることを破線で示す）．しかし線分AP_0は左が1個，右が6個と，線分AP_2は左に6個，右に0個と分割するので，どちらでもだめである．そうなると次は頂点CとAでスィープするか，頂点AとBでスィープするかしなければならない．

以下のプログラムはパラメータvを0, 1, 2と変えて3つの方向でテストするようにできている．しかし本体はβとγを使うように書いてあるから，上の説明とあっている．実際にはvを変えるとα，β，γが循環して使われる．

```
(define qpi (atan 1))              ; π /4

(define (triangle)
  (let* ((pleng (length points)) (plen3 (/ pleng 3)) (plen1 (- plen3 1))
         (lat (cdar corners)) (cc '()) (pointlist '()) (bdeg '()) (gdeg '())
         (blist '()) (gindex '()) (bindex '()) (gammalist '()))

  (define (v-iter v)               ; 頂点の対を順にテストする . v=0 なら B, C.
```

```
(let ((v1 (+ (remainder (+ v 1) 3) 3)) (v2 (+ (remainder (+ v 2) 3) 3))
    (bs (list (* 2 qpi) qpi qpi (* 2 qpi))))

(define (gfind gg)
  (if (null? gg) '()
      (let* ((p (list-ref pointlist (caar gg)))
             (g (list-ref p v2)) (b (list-ref p v1)) (i (list-ref p 2)))
        (cond ((and (< b bdeg) (< g gdeg))  ;点を取り込む条件
               (set! gindex (list-ref p 2)) (set! gdeg (list-ref p v2))
               (set! blist
                     (cons (list (list-ref p 2) (list-ref p v1)) blist))
               '#t)
              (else (gfind (cdr gg)))))))   ;末尾再帰

(define (update)
  (set! blist (sort blist (lambda (x y) (> (cadr x) (cadr y)))))
  (set! bdeg (cadar blist))
  (set! bindex (caar blist))
  (set! blist (cdr blist))
  (set! cc (crossp (list-ref corners (- v1 3)) (list-ref points bindex)
                   (list-ref corners (- v2 3)) (list-ref points gindex))))

(define (g-iter)              ;γ角を順に調べるための反復
  (cond ((= (length (filter (lambda (x) (>= x plen3)) (count cc cc cc))) 3)
         cc)                  ;うまく3分割できたら交点の座標をもって帰る
        (else (cond ((gfind gammalist) (update) (g-iter))
                    (else '()))))

(set! gammalist     ;v-iterの本体 ここから γ角の順にソートしたリストを作る
  (sort
    (map (lambda (x) (list (list-ref x 2) (list-ref x v2))) pointlist)
    (lambda (x y) (> (cadr x) (cadr y)))))
(set!  bdeg (list-ref bs v))          ;β角の初期化
(set!  gdeg (list-ref bs (+ v 1)))    ;γ角の初期化
(set!  blist '())                     ;βリストの初期化
(do ((j 0 (+ j 1))) ((= j plen3))
  (gfind gammalist))                  ;まずn/3個の点を確保
(update)
(cond ((g-iter))
      (else (v-iter (+ v 1))))))    ;v-iterの末尾再帰

(define  i -1)          ;triangleの本体 ここから
(set!  pointlist
   (map (lambda (p)                   ;各点のα，β，γを計算する.
          (let* ((x (car p)) (y (cdr p))
                 (alpha (atan (/ x (- lat y))))
                 (beta (atan (/ y x)))
                 (gamma (- (atan (/ (- lat x) y)) qpi)))
            (set! i (+ i 1)) (list x y i alpha beta gamma)))
        points))
(v-iter 0)))            ;まず辺BCに対してv-iterを呼ぶ
```

点が300個ある審判団のデータでも，すぐに答えが得られて，快適である.

■所要時間

それぞれのプログラムで実験した所要時間（秒）を**表-1**に示す.

	図 -1 のデータ	審判団データ 1	審判団データ 2
点の個数	18	300	300
山登り法	0	32	1.5
全数検査	0.25	1200	335
敵は幾万	0	7.3	3.1

表 -1

■なぜうまく分割できるか

もとの問題文には，こういうふうに分割できることは証明可能としか書いてない．しかしなぜ分割可能か少し考えてみたい．**図 -10** は図 -8 をすべての頂点の組に対して書いたものである．2 本の線が重なるところを黒の破線で示すのは図 -9 と同じである．

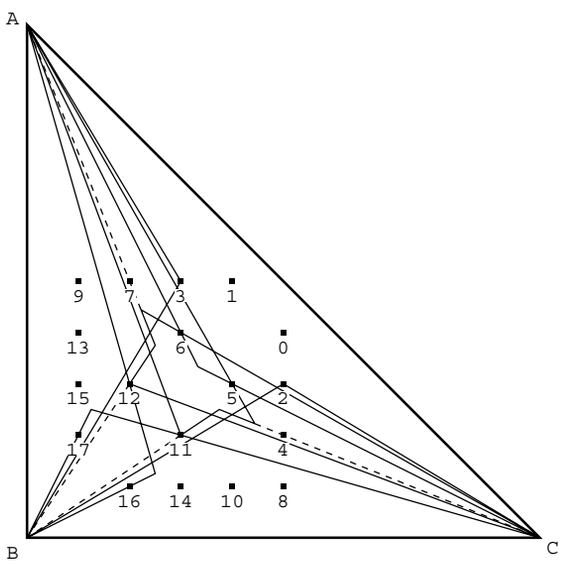

図-10

2 つの頂点と内部の他の 1 点，または 2 点で構成する三角形は，ちょうと $n/3$ 個の点を含む最小の三角形である．たとえば B P_{17} P_{11} C には P_6, P_{10}, P_{11}, P_{14}, P_{16}, P_{17} の 6 個の点がある．B P_{12} C では P_{17} を失った代わりに P_{12} を得た．B P_{11} P_4 C では P_{12} を失った代わりに P_4 を得た．B P_2 C では P_{11} を失った代わりに P_2 を得た．つまり常に 6 個の点がある．B, C を頂点としちょうど 6 個の点がある三角形はこれだけだ．他の頂点の組についても同様．

さて全部で点が n 個あって，各辺に立つ小三角形にそれぞれ $n/3$ 個の点があれば，各辺から適切に小三角形を選べば，各点がいずれかの小三角形に含まれるように選べる．いまの例では (B P_{11} P_4 C), (C P_5 P_6 A), (A P_7 P_{12} B) を選べばよい．そうしたら各頂点から，選ばれた小三角形の辺の中間へ向かって分割線を引き，その 3 本の交点を Q とすればよい．

もう一度図 -9 を見てみよう．この場合は最小三角形の取り方は一意ではなく，

(B P_0 C), (C P_2 P_5 A), (A P_4 B)

の組と

(B P_2 C), (C P_4 A), (A P_3 P_0 B)

の組が存在する．

■問題の難易度はどうか

この問題はやってみると見かけに相違して，存外手ごわかったという感じである．プロムナードの原稿としては，こればかりにとりかかっているわけではないが，たっぷり1カ月くらい，あれこれ考えさせられた．

すべてがこれほど手ごわいとも思わないが（この問題はFであった），本番のコンテストでは5時間に8問題解くのだからコンテストに参加する学生さんは大変である．プログラムは審判団の用意したテストデータに対して規定の時間内にしかるべき答えが出力できればよく，多少違っていてもそのテストにさえ合格すればしめたものというわけだが，特に今回のような問題は，本来ならプログラムを書くときの方針とか，プログラムの読みやすさ，きれいさなども評価するのが望ましい．しかしさて実行のことを考えると不可能であろう．

今回はどこかにも書いたようにschemeのプログラムを示すことで勘弁してもらった．私はだいたいいつもutilispでプログラムを書く，あるいは考える．しかし今回は幾何学的な問題なので，プログラムはpostscriptでかなり書いた．最後のスィープのプログラムもオリジナルはpostscriptである．プログラムを書きながら，少しずつ描画して確認作業をした．したがってpostscript, utilisp, schemeと3種類のプログラムが手元にそろっている．そのうちschemeのはなるべく読みやすくするように努めて書き直しを繰り返したが，まだ納得のいくレベルにはならなかったのは気になっている．

8月に亡くなったDijkstraならどういうプログラムを書いただろうか．見てみたいものである．

（平成14年10月24日受付）