解説 *New Paradigms for Designing Memory-Constrained Algorithms*

省メモリのための新たなアルゴリズム設計技法

制限された作業用メモリでアルゴリズムをいかに設計するか 後編

浅野 哲夫 北陸先端科学技術大学院大学情報科学研究科

世の中に大量のメモリを必要とするプログラムは大量に存在するが，前回の解説ではいかに少ないメモリで問題を解くかに焦点を当てて，省メモリアルゴリズム設計のための基本的な考え方を紹介した．前回に紹介した省メモリアルゴリズムの例は考え方としては興味深くても，実用的な応用からは遠いのではないかという印象を持たれた読者もおられるだろう．そこで，今回は画像処理と計算幾何学に関する諸問題に省メモリアルゴリズムの考え方がいかに適用できるかを説明しよう．

省メモリアルゴリズム：画像への応用

図 -1 のカラー画像を参照されたい．この写真には赤く塗られた橋があるが，この橋の部分だけを取り出したいものとする．ここでは，カラー画像は赤青緑の 3 つの色成分が 0 から 255 までの 256 段階で表現されているものと仮定している．赤成分の値が 255 で他の成分の値が 0 なら，その画素の色は赤ということになる．すべての成分が最高値 255 なら白，すべて 0 なら黒を表す．問題の写真の赤い部分の色は，大体，赤成分の値が 80 から 255，緑成分が 0 から 80 の範囲にある（青成分の値は何でもよい）ことが分かっているものとする．そこで，入力画像に対して，上記の範囲の色を持つ画素には 1 を，それ以外の画素には 0 を割り当てることにすると同じサイズの 2 値画像が定義できる．このように，赤青緑の各成分の値の範囲を指定することにより任意の色味を指定して，その色味の画素かどうかを表す 2 値画像を計算することができる．実際に計算で求めたのが**図 -2** である．値 1 の画素を白で表示している．

図 -1　入力のカラー画像．中央の赤い橋が抽出対象

■ 入力画像中で指定された色味を持つ領域を抽出したい

図 -2 を見ても分かるように，赤っぽい画素は実にたくさん存在するのである．しかし，図 -2 の 2 値画像において面積（画素数）最大の連結領域を求めて，そこだけを取り出せば，**図 -3** のように，ほぼ橋に対応する部分を取り出すことができている．

■ 省メモリを意識した方式

画像処理の専門家にとって上記の処理は簡単な練習問題であろう．まず，配列に入力画像を読み込み，さらに対応する 2 値画像を計算して，その結果を別のビット配列に蓄える．このビット配列上に連結成分ごとに異なるラベルをつけるアルゴリズムを適用する．そのためにラベルを蓄えるための配列が必要である．ラベルが求まったら，次は連結成分ごとに面積（画素数）を求める．ラベル配列さえあれば，それをラスター順（左から右，下から上の順）に走査す

図 -2　赤：80 ～ 255, 緑：0 ～ 80, 青：0 ～ 255 の範囲にある画素だけで定義される 2 値画像

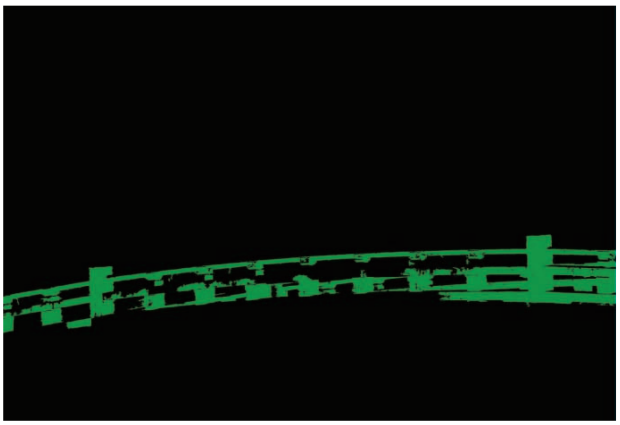

図 -3　図 -2 の画像で面積（画素数）が最大の連結成分の画素にだけ同じ色をつけた画像

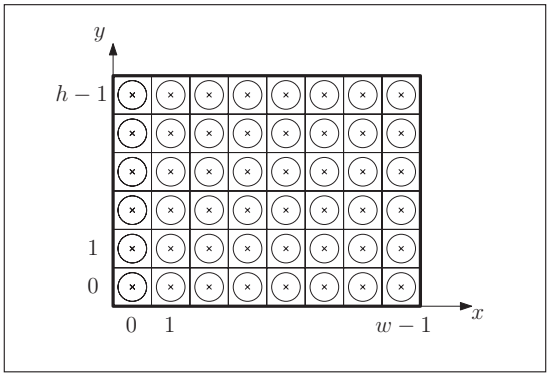

図 -4　$w \times h$ の画像．図中の円は画素に対応．円を囲む正方形が画素領域．

ることで各連結成分のサイズを線形時間で計算することができるが，そのためには連結成分ごとに面積を蓄えるための配列が必要である．連結成分の個数は，市松模様のような最悪の場合には画素数の半分程度になるので，結局 $O(n)$ の配列がここでも必要になる．連結成分の面積さえ求まれば，後の処理は簡単である．

■ 作業用のメモリをどこまで減らせるか？

至極簡単な操作である．しかし，どれだけ配列を使っただろう？　数えきれないほどである．逆に，どれだけ作業領域を減らすことができるだろう？　極端な場合，定数領域だけでも上記の作業が実行できるだろうか？

信じがたいことかもしれないが，定数領域だけでも上記の作業を実行できる．もう少し正確に言うと，次の通りである．まず，入力画像を蓄えるための配列はあるものとする．この配列を作業用に用いてもよいが，最終的には同じ値を保持していなければならないと仮定すると作業用に用いるのは難しい．出力用の配列は用意しない．出力は 2 値画像なので，各画素の値をラスター順に計算して出力するだけである．入力画像用の配列以外には配列を一切用いない．

これだけ厳しいメモリ制約でも問題が解ける．もちろん，線形時間で実行できるわけではない．しかし，画素数 n の 2 乗程度，正確には $O(n^2 \log n)$ 時間で処理を行うことができる．

しかし，いくら何でも作業領域を定数サイズに限定するのは極端であろう．画像を扱う場合には，バッファを仮定するのは当然のことである．バッファは画像の数行分に対応すると仮定すると，利用できる作業領域として $O(\sqrt{n})$ を仮定するのは妥当である．これだけの作業領域があれば，計算時間を $O(n\sqrt{n})$ に改善することができる．

➤ 連結成分の個数を求める定数領域アルゴリズム

信じられないという読者のために種明かしをしよう．まず最初に基本となるアルゴリズムから説明しよう．**図 -4** に示したのは画像の模式図である．画像を構成する画素を小さな円で表し，それを囲む正方形を画素領域とする．隣り合う画素領域の間の正方形の辺を画像の辺と呼ぶ．

画像の縦と横のサイズ h と w は一般に異なってもよいが，いずれも画素数 n に対して $O(\sqrt{n})$ であ

ると仮定して一般性を失わない．

2 値画像は値が 1 の白画素と値が 0 の黒画素からなる．2 つの白画素が上下または左右に隣接しているとき，それらは連結であるという．この連結関係の推移的閉包を取ったものを連結成分と呼ぶ．ひらたく言えば，連結成分とは，その中のどの 2 画素の間にも，その成分の中だけを通り，上下左右にだけ進む白画素だけの道が存在するような極大な白画素の集合のことである．

図 -5 に簡単な例を示す．全部で 3 個の連結成分があるが，そのうちの 1 つは別の連結成分に完全に包囲されているので，島と呼ばれるものである．

連結成分に対して自然に境界を定義することができる．連結成分の境界に位置する画素（の中心）をつないだものとして境界を定めることもあるが，本稿では各画素を正方形の画素領域と見なして，その境界辺をつないだ多角形状の境界で表現する．図 -5 に示すように，連結成分は内部に穴を含んでもよいので，境界にも外部境界と内部境界がある．穴の個数は任意であるので，1 つの連結成分が複数個の内部境界を持つことはあり得るが，外部境界はただ 1 つである．したがって，外部境界の個数を数えることができれば，連結成分の個数が分かる．これが基本的な考え方である．

境界は閉路に対応している．境界を構成する辺は，画素に対応する小正方形の側辺に対応するので，垂直と水平のどちらかである．ここで，次のような辞書式順序を考えよう．まず，水平辺は垂直辺より辞書式順序で大きいとする．垂直辺どうしでは，上にある方が辞書式順序でも大きいとする．同じ行に存在する場合には，右にある垂直辺の方が大きいと定める．このように辺の間に順序を定めると，どの境界にも（辞書式順序で）最も小さい垂直辺が 1 つだけ存在する．このようにして，それぞれの境界に対して最も下で最も左にある垂直辺のことを，この境界の**代表辺**（canonical edge）と呼ぶことにしよう．

境界上のある垂直辺 e が代表辺かどうかを判定するには，その境界を辿って，上記の辞書式順序で辺 e より小さい垂直辺が存在するかどうかを調べればよい．1 つの方向に境界を辿ってもよいが，それでは最悪の場合に時間がかかるので，前稿で述べたように，双方向に境界を辿ることにする．これを具体的に関数の形にしたのが下に示す IsCanonical(e) である．

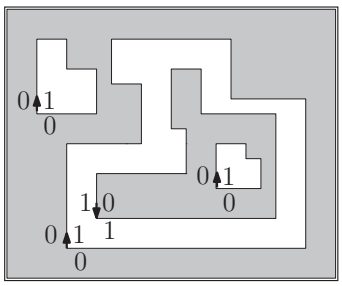

図 -5　画像の幾何モデル．各画素を小さな正方形領域と見なしたとき，隣接する画素の間に水平または垂直の辺を定義することができる．

垂直辺 e が代表辺かどうかを判定する関数

```
IsCanonical(e)
  e の時計回りの次の辺を e_c とする．
  e の反時計回りの次の辺を e_cc とする．
  while(e > e_c and e_cc > e){
    e_c を次の時計回りの辺で置き換える．
    e_cc を次の反時計回りの辺で置き換える．
  }
  if e_c = e または e_cc = e then return TRUE.
  else return FALSE.
```

代表辺は境界ごとに定義されるので，内部境界に対しても定義される．そこで外部境界の代表辺と区別するために，境界に方向を定める．図 -5 の例では，白画素（連結成分の内部）を右に見るように境界を方向づけている．したがって，外部境界は時計回り，内部境界は反時計回りになる．

さて，求めたいのは外部境界の代表辺である．基本的には，画像を順に左から右，下から上へ順に走査（ラスタースキャン）して，黒から白に変化する個所の垂直辺 e を見つけるたびに，上記の関数 IsCanonical(e) を呼び出して e が代表辺かどうかを調べればよい．しかし，明らかに代表辺でないよう

な垂直辺については関数の適用を避けるのが得策である．そこで，外部境界の代表辺の性質について注目する．図 -5 を見れば明らかなことであるが，外部境界の代表辺（図中で矢印で示した辺）の右下の画素は必ず黒（値 0 の画素）である．これに対して，内部境界の代表辺では右下の画素値が 1 となる．したがって，垂直辺を見つけたときには，まずその右下の画素の値を調べ，それが 0 でなければ，その辺が外部境界の代表辺となることはないので，すぐにその辺を棄却する．このことから，次のようなアルゴリズムが導かれる．

連結成分の個数を求める定数領域アルゴリズム

入力　n 画素からなる 2 値画像 G.

出力　入力画像に含まれる連結成分の個数.

アルゴリズム

```
N = 0. // 連結成分の個数
for y = 1 to h
for x = 1 to w{
  if G[x][y] = 1 かつ G[x − 1][y] = 0 かつ
    G[x][y − 1] = 0 then{
    画素 (x, y) の左の垂直辺を e とする.
    if IsCanonical(e) then
      Nの値を 1 増やす.
  }
}
最後に連結成分の個数 Nを出力する.
```

前回の解説で，直近上位要素発見問題（配列上で，それぞれの配列要素について，それより大きい要素の中で最も近いものを求める問題）については，作業領域が定数に限定されていても，双方向探索を用いることにより，長さ n の配列に対する上記問題が $O(n \log n)$ 時間で解けることを示した．詳しい証明は省略するが，2 値画像に含まれる連結成分の個数（外部境界の代表辺の個数）を求める上記のアルゴリズムも同様の解析により，画素数を n とすると，実行時間は $O(n \log n)$ で与えられることが分かる．

上記のアルゴリズムを拡張すると，計算時間を $O(n \log n)$ に保ったままで，連結成分の個数だけではなく，連結成分の代表辺を見つけるたびに，その連結成分の面積（画素数）も出力することができる．そのときに問題になるのが穴の存在である．穴がなければ，多角形 $P = (p_0, p_1, \ldots, p_n = p_0)$ の面積を求める公式

$$S = \frac{1}{2}\left|\sum_{i=0}^{n-1} x(p_i)\,(y(p_{i+1}) - y(p_{i-1}))\right|$$

を用いることができる．ただし，$x(p_i)$, $y(p_i)$ は点 p_i の x, y 座標を表し，$p_{-1} = p_{n-2}$ とする．2 値画像の連結成分の場合，外部境界は垂直または水平の（単位長の）辺から構成されている．そのような辺を順に辿っていくとき，2 つの連続する辺を知っていれば，$x(p_i)(y(p_{i+1}) - y(p_{i-1}))$ に対応する値を計算することができる．この値を順に足しこんでいくだけで面積が計算できる．したがって，穴のない連結成分の外部境界を辿ることにより，その面積（画素数）を定数領域でしかも線形時間で求めることができる．

基本的には，穴がないものとして，外部境界で定まる多角形の面積を求めたあと，その内部に含まれる穴の面積を引けば正しく面積を求めることができる．しかし，作業領域が限られていると，穴がどの外部境界に含まれているかが分からないなど，さまざまな困難がある．後で効率の良いアルゴリズムを紹介するが，その前に知られている結果を用いるとどうなるかを先に紹介する．

➤ 2 画素の連結度判定の定数領域アルゴリズム

代表辺の概念を用いると，任意に与えられた 2 画素が同じ連結成分に含まれるかどうかを，それぞれの画素を含む連結成分の面積の和に比例する時間で求めることができる．ここでは文献 3）の方法を紹介しよう．

2 つの白画素 p と q が与えられたとしよう．まず，p からまっすぐ左に境界に達するまで進み，その垂直辺が代表辺かどうかを境界を辿ることによって判定する．外部境界の代表辺なら，その代表辺の位置を記録する．内部境界の代表辺なら，そこから再びまっすぐ左に境界に達するまで進み，上と同じ操作

を繰り返す．

他方の白画素 q についても同様の処理を行い，q を含む連結成分の外部境界の代表辺を求める．後は，両方の代表辺が同じかどうかを判定するだけである．両者が一致するなら，p と q は同じ連結成分に属し，そうでなければ別の連結成分に属すると言える．また，これらの処理が連結成分の面積に比例する時間で実行できることは明らかであろう．

図 -6 は，任意に指定された画素から，その画素を含む連結成分の外部境界の代表辺までの探索経路を図示したものである．図において，画素 A から左に進むと境界辺にぶつかるが，そこから境界全体を辿って代表辺を求めると，外部境界の代表辺であることが分かるので，そこで処理を終えることができる．画素 B については，複数の内部境界を経る必要があるが，内部境界の代表辺から再び左に境界辺まで進むという操作を繰り返すと，やがて外部境界の代表辺に到達できる．画素 A, B, C については，同じ代表辺で終わるので，同じ連結成分に属することが分かる．画素 D からは別の代表辺で終わるので，別の連結成分に属する．

どんな 2 画素についても，それらが同じ連結成分に属するかは，それらが属する連結成分の総面積に比例する時間で判定できることが分かった．この結果を用いると，与えられた 2 値画像に含まれる各連結成分の面積を求めることができる．

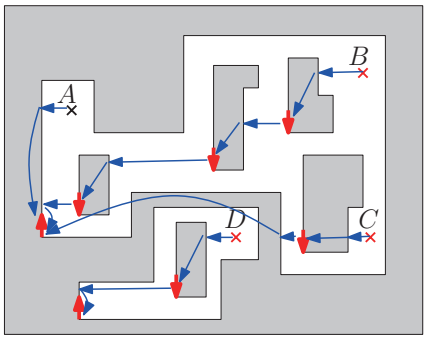

図 -6　質問点から外部境界の代表辺までの経路

連結成分の面積を求める定数領域アルゴリズム 1

入力　n 画素からなる 2 値画像 G.

出力　入力画像に含まれる各連結成分の面積．

アルゴリズム

```
for y = 1 to h
for x = 1 to w{
  if G[x][y] = 1 かつ G[x−1][y] = 0 かつ
  G[x][y−1] = 0 then {
    画素 (x, y) の左の垂直辺を e とする．
    if IsCanonical(e) then {
      S = 0.
      for y′ = 1 to h
        for x′ = 1 to w
          if (x, y) ≠ (x′, y′) かつ G[x′][y′] = 1 かつ
          (x, y) と (x′, y′) が同じ連結成分に属する
          then S = S + 1.
        面積 S と代表辺 e を出力する．
      }
    }
  }
```

確かに上記のアルゴリズムで外部境界の代表辺と連結成分の面積を求めることはできるが，効率はかなり悪い．特に，アルゴリズムの下から 3 行目で 2 画素が同じ連結成分に属するかを判定するのに最悪 $O(n)$ 時間かかるから，それを n 回繰り返すと，連結成分あたり $O(n^2)$ の時間を要する．連結成分は $O(n)$ 個存在し得るので，全体の計算時間は $O(n^3)$ ということになる．もっと効率よくできないだろうか？　実は驚くほど高速化できる．

➤ 連結成分の面積を求める定数領域アルゴリズム

余分な配列を使わなくても，n 画素からなる 2 値画像に含まれる連結成分の個数だけではなく，それぞれの面積（画素数）も出力できることが分かった．ここでは処理の高速化に焦点を当てる．

高速化の秘訣は，連結成分に属する画素を組織的に訪問する方法を考えて，単純に要素数を数えることにある．まず，外部境界の代表辺が分かっているものとする．ここから外部境界を一方向に 1 周する．先に仮定したように，連結成分の内部，したがって

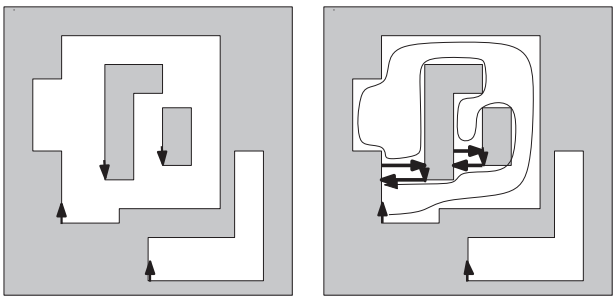

図 -7　連結成分に属する画素の列挙方法．（a）連結成分の外周と内周の代表辺．（b）連結成分内部での探索．

内部境界は境界の右に存在する．そこで，外部境界を辿る間，上方への辺 e に来るたびに，そこから右へ右に進んで境界の辺 e' に出会うと，今度は e' が内部境界の代表辺かどうかを判定する．大事なことは，e' が外部境界上にあるかどうかまで含めて判定しようとすると計算時間がかかるので，e' が内部境界の代表辺かどうかだけを調べるのである．そのために e' から双方向に境界上を探索する．もし e' が内部境界の代表辺であれば，さらにその境界を一方向に辿り，上向きの辺に会うたびに，同じ処理を行う．

内部境界上を一方向に辿っているとき，再度，現在の辺が代表辺かどうかを判定する．あらかじめ代表辺の位置を記憶しておけば問題なさそうに思えるが，上にも述べたように，1 つの連結成分の中に多数の穴が存在する場合には，内部境界を辿るうちに別の内部境界に移動することがあるので，十分な作業領域がないかぎり，すべての代表辺を記憶しておくことは不可能である．

さて，内部境界の代表辺 e に再び戻ってくると，対応する内部境界上の辺はすべて調べ終わったことになる．この代表辺から始まる内部境界をなぜ調べることになったかというと，その左に上向きの辺があったからである．そこで，代表辺 e から今度は左に辿り，呼び出し側の上向き辺 e' を求める．辺 e' が求まったら，そこでの処理を続ける．具体的には，その次の辺に移って，それが上向き辺なら右に境界辺を探索するという処理を行う．

このように処理を続けると，やがては外部境界の代表辺に戻ってくる．このとき，それまで調べていた連結成分に関する処理を終わることができる．

以上の処理を模式的に説明したのが**図 -7** である．外部境界の代表辺から始めて，どのような順序で探索が進んでいるかを説明している．

アルゴリズム全体は次の通りである．

連結成分の面積を求める定数領域アルゴリズム 2

入力　n 画素からなる 2 値画像 G.

出力　入力画像に含まれる各連結成分の面積.

アルゴリズム

```
for y = 1 to h
for x = 1 to w
  if G[x][y] = 1 かつ G[x−1][y] = 0 かつ
  G[x][y−1] = 0 then {
    画素 (x, y) の左の垂直辺を e_s とする.
    if IsCanonical(e) then {
      S = 0. // 面積(画素数)
      e = e_s の次の辺.
      repeat {
        if e は上向きの辺 then {
          e′ = e.
          repeat {
            p = e′の右の画素 .
            S = S + 1.
            e′ = p の右の垂直辺 .
          } until(e′は境界辺，つまり e′の右は 0)
          if IsCanonical(e′) then e = e′の次の辺 .
          else e = e の次の辺 .
        if IsCanonical(e) then{
          e′ = e.
          repeat {
            p = e′の左の画素 .
            e′ = p の左の垂直辺 .
          } until(e′は境界辺，つまり e′の左は 0)
          e = e′.
      } until(e = e_s)
      最後に，e_s の右にある画素数を S に加える.
      面積 S と代表辺 e_s を出力する.
    }
  }
```

定理 1　n 画素の 2 値画像が与えられたとき，画像に含まれる連結成分の面積(画素数)と代表辺の位置を $O(n \log n)$ 時間で求める定数領域アルゴリズムが存在する．

上の定理ではアルゴリズムが存在するとだけ述べているが，上に与えたアルゴリズムがそれである．アルゴリズムを注意深く眺めると，連結成分の各画素は，左右の方向に一度ずつ訪問されていることが分かる．右に進むときだけに画素数をカウントしているので，連結成分の面積が正確に求まっていることが理解できるだろう．境界上の辺については，一方向には一度だけ調べている．そのほかに，代表辺かどうかを調べるために，上向きの辺からそれぞれ 2 回ずつ双方向の探索を行っている．したがって，連結成分の境界に関する計算時間は，境界の長さの和を n_i とすると，$O(n_i \log n_i)$ となる．一方，連結成分の面積を S_i とすると，画素を調べるのに要した時間は $O(S_i)$ である．よって，定理の計算時間を得る．

➤ 最大連結成分からなる画像を求める定数領域アルゴリズム

2 値画像が与えられたとき，連結成分の個数だけでなく，連結成分の面積なども定数領域だけで求められることが分かった．では，最大の連結成分だけを残して，それ以外の連結成分をすべて消し去ることはできるだろうか？　より正確には次の問題を考えよう．

最大連結成分抽出問題　n 画素のカラー画像と，カラー情報を 2 値に変換する関数 $f()$ が与えられたとき，この関数によって定まる 2 値画像における面積最大の連結成分に属する画素での値を 1 とし，それ以外の画素での値を 0 とするような 2 値画像をラスター順に出力せよ．

この問題は定数領域で解けるだろうか？　実は，答を知ってしまえば意外に簡単である．アルゴリズムは次の通りである．

まず，先に述べたアルゴリズムにより，最大の連結成分の外部境界上の代表辺 e を求める．これが求まれば，それぞれの画素について，2 画素の連結度判定のアルゴリズムを用いて，その画素が属する連結成分の外部境界上の代表辺が e に一致するかを求める．一致すれば 1 を出力し，そうでなければ 0 を出力する．

アルゴリズムの解析も簡単である．最初に代表辺 e を求める処理が $O(n \log n)$ でできることはすでに知っている．次に 2 画素の連結度判定は $O(n)$ 時間でできることも分かっているから，全体で必要な計算時間は $O(n^2)$ ということになる．実際には，入力画像の同じ行で値 1 の画素が連続していれば，それらは当然同じ出力値を持つから，連結度判定のアルゴリズムを使うまでもない．したがって，最大連結成分が複雑な形状でなければ，計算時間は $O(n\sqrt{n})$ 程度で抑えられることになる．

➤ 2 値画像のラベル付け問題

上のアルゴリズムを拡張すると，入力の 2 値画像に対してラベル付けを行った結果の行列を定数領域で求めることも可能である．作業領域が限定されているので，この場合も，出力すべきラベル行列を蓄えることはせず，単にラベル行列の値をラスター走査の順に出力するだけである．

考え方は至って簡単である．入力の 2 値画像をラスター順に走査しているものとし，現在の画素を p とする．画素 p の値が 0 ならば，単に 0 を出力するだけなので，問題はない．値が 1 ならば，p が属する連結成分の番号(ラベル)を求める必要がある．では，連結成分に番号を付けるとして，自然な付け方とは何だろう．連結成分には 1 つだけ外部境界があり，その外部境界上には唯一に定まる代表辺があることを知っている．つまり，連結成分と代表辺をペアにして考えることができる．辺の間には自然な辞書式順序があったから，それを用いるのはごく自然なことである．そこで，ここでは辞書式順序で定まる番号を連結成分のラベルとして採用することにしよう．

さて，現在の (値 1 の) 画素を p とするとき，p に付けるべき番号は，p が属する連結成分の代表辺 e

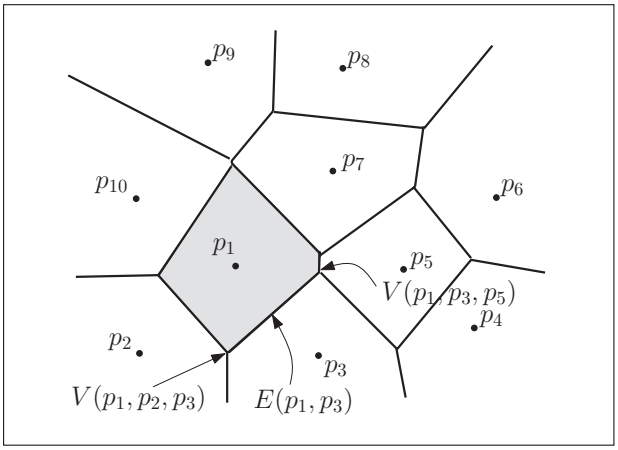

図 -8　平面をどの点に近いかという関係で分割して得られるボロノイ図．網掛けした領域は，p_1 が最も近くなる領域．

の順序によって決まる．e が何番目の代表辺かを知るには，もう一度最初からラスター走査を行って，e に至るまでの各辺について，それが外部境界上の代表辺かどうかを調べればよい．この処理が定数領域で $O(n \log n)$ 時間でできることはすでに分かっている．つまり，それぞれの画素での計算時間は $O(n \log n)$ 時間で抑えられる．よって，全体の計算時間は $O(n^2 \log n)$ となる．

➤ $O(\sqrt{n})$ の作業領域を用いるアルゴリズム

n 個の画素からなる画像を扱うのに $O(\log n)$ ビットしか作業領域に使えないというのはあまりにも厳しい制限である．画素数が n なので，画像の行数と列数は共に $O(\sqrt{n})$ であると仮定してもよいだろう．では，$O(\sqrt{n})$ の作業領域（正確には，$O(\sqrt{n}) \log n$) ビットの作業領域）を用いると高速に処理することができるだろうか？　紙数の関係で詳しい方法を解説することはできないが，まったく違う考え方により，次のようなアルゴリズムを得ることができる．興味がある読者は筆者に連絡されたい．

連結成分の個数を求める問題　ラスタースキャンを 1 回だけ実行することにより，$O(n)$ の時間で解くことができる．

連結成分の面積を求める問題　ラスタースキャンを 1 回実行するだけで十分である．

2 画素の連結度判定　2 画素が同じ連結成分に属するかどうかも，ラスタースキャンを 1 回だけ実行することにより判定できる．

2 値画像に対するラベル行列　各行でラスター走査を 1 回ずつ実行することにより，ラベル行列をラスター順に出力することができる．実行時間は $O(n\sqrt{n})$ 時間である．

省メモリアルゴリズム：計算幾何学への応用

省メモリアルゴリズムは筆者の専門である計算幾何学にも着実に応用されている．本稿の最初に定数領域データ構造について述べたが，幾何対象物に対しても同様のデータ構造を考えることができる．

計算幾何学とは，幾何的に定義された計算問題を解く効率の良いアルゴリズムを開発したり，その問題の本質的な計算困難さを解析することを目的とする理論計算機科学の一分野である．計算幾何学ではさまざまな新たな概念が生み出されているが，中でも平面上に多数の点が与えられたとき，どの点に最も近いかという関係に基づいて平面を分割することによって得られるボロノイ図 (Voronoi diagram) が有名である（**図 -8** 参照）．筆者も計算幾何学の研究を始めたころ，ボロノイ図を描くアルゴリズムに興味があり，計算幾何学のテキストを読んでプログラムを書こうと考えたことがあるが，複雑なアルゴリズムとデータ構造のせいで挫折した経験がある．また，アルゴリズム関係のテキスト一般について言えることであるが，ある問題を解くのに理論的に効率の良いアルゴリズムがあれば，効率は悪いが簡単だというアルゴリズムは紹介されないことが多い．反対意見も多いと思われるが，最近の計算機の速度の向上により，問題のサイズが大きくなければ入力サイズの 2 乗に比例するアルゴリズムは十分に実行可能である．

平面上に 2 点 a と b だけが与えられたとき，それらの垂直 2 等分線を引けば，平面全体を，b より a の方が近い領域と，a より b の方が近い領域に分割することができる．多数の点が与えられたときも，基本的には 2 点の垂直 2 等分線を多数引くことによ

り，それぞれの点が最も近くなる領域に分割することができる．図 -8 に一例を示している．同図において，$E(p_1, p_3)$ というのは，p_1 と p_3 の垂直 2 等分線の一部で，ボロノイ辺と呼ばれるものである．このボロノイ辺は別のボロノイ辺との交点に端点を持つが，この例では左の端点が $V(p_1, p_2, p_3)$ で，右の端点が $V(p_1, p_3, p_5)$ である．そのような端点のことはボロノイ頂点と呼ばれる．

ボロノイ図は計算幾何学の初期から研究され，平面上の n 個に対するボロノイ図を $O(n \log n)$ 時間で構成するアルゴリズムが知られている．分割統治法や平面走査法を用いるものなど，多数のアルゴリズムが知られているが，いずれもデータ構造を工夫しないと，$O(n \log n)$ を実現することはできない．

本稿では省メモリのアルゴリズムについて述べてきた．最初に述べたのが定数領域データ構造であったが，実は，ボロノイ図に対しても考えることができる．そもそも，ボロノイ図を計算するとは何を意味するのであろうか？　1 つは，ボロノイ図を画面上に描くことであり，もう 1 つはボロノイ図を構成するボロノイ辺とボロノイ頂点を列挙することにより別の問題を解くということである．この 2 つの目的を達成するために，ボロノイ辺とボロノイ頂点に関する操作を効率よく行えるようにするデータ構造を構成することが，ボロノイ図を求めるということの真の意味であろう．つまり，目的はボロノイ辺とボロノイ頂点に関する操作を実現することである．十分大きな作業領域が利用できる場合には，複雑なデータ構造を構成して，それらの操作を効率よく，たとえば n 点の場合には各操作を $O(\log n)$ 時間とか，$O(1)$ 時間で実行することが要求される．定数領域アルゴリズムのように作業領域が極端に限定される場合には，各操作を実行する時間を増やさなければ対応できない．ここでは，$O(1)$ の情報を蓄えておくだけで，必要な各操作を $O(n)$ 時間で実行することを目標とする．

ボロノイ図に対する定数領域データ構造を説明してもよいが，ここではボロノイ図の双対にあたる Delaunay 三角形分割に対するデータ構造について説明しよう．**図 -9** は同じ点集合に対する Delaunay 三角形分割を示している．

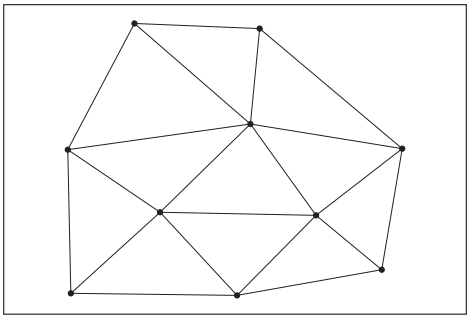

図 -9　平面上に与えられた点集合に対する Delaunay 三角形分割

一般に，平面上の点集合 S に対する三角形分割とは，次のように定義される図形である．まず，与えられた点の中から適当に 2 点を選んで，それらを線分で結ぶ．次に別の 2 点を選び，それらを結ぶ線分がすでに引かれた線分と端点以外で交差しないなら，それらも線分で結ぶ．この操作をすべての点対について行うと，すべての面が三角形であるような図形が得られる．もし三角形以外の面があったとすると，その面は多角形の形をしているが，対角線を引くと，より頂点数の少ない面に分割されるからである．

n 点からなる点集合 S が与えられたとき，三角形分割の仕方は n の指数通り存在するが，その中でで き上がったどの三角形についても，その外接円の内部に S の他の点が存在しないような三角形分割が存在することが知られている．そのような三角形分割は，任意の点集合に対して縮退がないかぎり（すなわち，どの 3 点も同一直線上にはなく，どの 4 点も同一円周上にないかぎり）唯一に定まるが，そのような三角形分割のことを Delaunay 三角形分割 (Delaunay triangulation) と呼んでいる．実は，この三角形分割は，可能な三角形分割の中でも最小の内角が最大となるという良い性質を持っていることも知られている．

Delaunay 三角形分割に対する定数領域データ構造では次の 2 つの操作をサポートすれば十分である．

- FirstDelaunayEdge(u)：点 u に接続する最初の Delaunay 辺を返す．

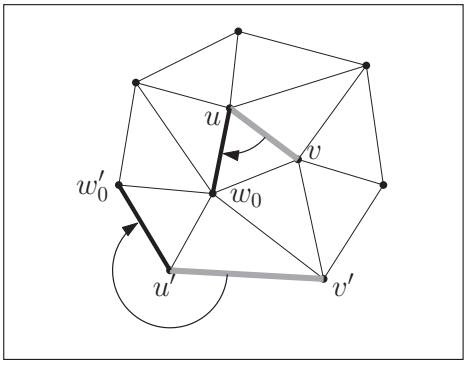

図 -10　ある点に接続する多数の Delaunay 辺の中で，時計回りの順で，任意に指定した辺の次の辺.

- ClockwiseNextDelaunayEdge(u, v)：点 u に接続する Delaunay 辺の中で時計回りの順に (u, v) の次の辺を返す.

点 u に接続する最初の Delaunay 辺を求めるのは容易である．点集合 S の中で u に最も近い点 v を求めれば，uv を直径とする円の内部に S の他の点は存在しないので，(u, v) が Delaunay 辺であることが分かる.

次に，u に接続する Delaunay 辺を時計周りの順に並べるとして，(u, v) の次の Delaunay 辺を求めたい．**図 -10** に示すように，2 つの場合を考える必要がある．1 つは，点 u が点集合 S の凸包（点集合を包含する面積最小の凸多角形）の内部の点の場合で，他方は凸包上の点の場合である．内部の点の場合には連続する 2 つの Delaunay 辺がなす角度は 180 度以上になることはないが，凸包上の点の場合には 180 度を超えるところがある．凸包の内部の点の場合には，連続する Delaunay 辺で三角形が定まるが，その外接円は S の他の点を含まない．したがって，$S-\{u, v\}$ のすべての点 w について，u から v への有向直線の右にあって，かつ u, v, w の 3 点を通る円の中で半径が最小となる点 $w \in S$ を求めれば，(u, w) が求める次の Delaunay 辺である．凸包上の点の場合には，時計回りの順に次の凸包上の点を求めればよい．したがって，どちらの場合も $O(n)$ の時間で次の Delaunay 辺を求めることができる.

この定数領域データ構造を用いると，与えられた点集合に対するユークリッド最小木を求めることもできる．詳細は論文 [2] に譲るが，定数領域だけでもユークリッド最小木が $O(n^3)$ 時間で求まることはまったく自明ではない.

このほかにも，単純な多角形（穴を含まない多角形）の内部だけを通って任意に指定された 2 点を結ぶ最短経路を $O(n^2)$ 時間で求める定数領域のアルゴリズムも知られている [1].

省メモリアルゴリズムの今後

省メモリアルゴリズムの研究は始まったばかりである．もちろん，対数領域アルゴリズムの名の下に長い研究の歴史はあるが，主に計算複雑度の観点から研究されてきた．典型的には $O(\sqrt{n})$ 程度の作業領域を用いてどれだけのことができるかに興味があるが，ほとんど分かっていないのが実情である．計算幾何学への応用でも，$O(1)$ の作業領域を用いるアルゴリズムはあるが，$O(\sqrt{n})$ の作業領域をどのように使えば高速化が達成できるかは知られていない.

参考文献

1) Asano, T., Mulzer, W. and Wang, Y. : Constant-Work-Space Algorithm for a Shortest Path in a Simple Polygon, (Invited talk) Proc. 4th International Workshop on Algorithms and Computation, WALCOM, Dhaka, Bangladesh, pp.9-20 (Feb. 2010). (Lecture Notes of Computer Science, LNCS 5942, Springer).
2) Asano, T. and Rote, G. : Constant Working-Space Algorithms for Geometric Problems, Proc. Canadian Conference on Computational Geometry, pp.87-90, Vancouver (2009).
3) Malgouyresa, R. and Moreb, M. : On the Computational Complexity of Reachability in 2D Binary Images and Some Basic Problems of 2D Digital Topology, Theoretical Computer Science 283, pp.67-108 (2002).

(2011 年 9 月 5 日受付)

■**浅野 哲夫（正会員）** t-asano@jaist.ac.jp
1977 年大阪大学大学院工学博士．大阪電気通信大学教授を経て 1997 年より北陸先端科学技術大学院大学教授．計算幾何学に関する研究に従事．本会，ACM, 電子情報通信学会各フェロー.