アルゴリズム 10－1
（1989. 9. 22）

2値画像の一符号化法と演算アルゴリズム

西谷泰昭, 吉田浩幸†, 清水賢資
群馬大学工学部, †沖電気工業(株)

Pコード列と呼ばれる2値画像表現について述べる．x, y座標の2進数表現を交互に並べることによって画素を整数に符号化したものがPコードである．2値画像はソートされたPコードの列によって表現される．Pコードは次のような性質を持つ: (1)Pコードの組によって4分木(quadtree)のノードに対応する正方領域を表現できる．(2)各座標の加算, 比較は, Pコード上のビット演算, 算術演算で行うことができる．本稿では, 上記の性質, Pコード列のメモリ量, Pコードを使ったアルゴリズムについて述べる．関連した研究にquadcode, リニア4分木があるが, これらの研究ではコードを整数ではなく, 列として扱っている．

Encoding and Operations of Binary Images

Yasuaki Nishitani, Hiroyuki Yoshida†, Kensuke Shimizu
Faculty of Engineering, Gunma University, Kiryu 376 Japan
†Oki Electric Industory, Minato-ku, Tokyo 105, Japan

Representations of binary images, called as pixel-code sequences, are described. A pixel is encoded with an integer termed a pixel-code, by interleaving the binary representaions of the x and y coordinates. A binary image is represented by the sorted sequence of pixel-codes. The pixel-code has the following properties: (1) a pair of pixel-codes may represent a square region corresponding to a node of a quadtree, and (2) addition and comparison for each coordinate may be performed by bit operations and arithmetic operations on pixel-codes. We discuss the above properties, the amount of memory space, and some algorithms using pixel-codes. Related approaches are the quadcode and the linear quadtree. However, these codes are treated as sequences, not integers.

1 まえがき

画像処理，コンピュータグラフィクスなど大量の図形データを計算機で処理するには，莫大な計算量とメモリ量が必要であり，データの効率的な表現とそのデータに対する効率的な演算が重要となる．

2値画像を表現する代表的なデータ構造のひとつに4分木(quadtree)がある[1,5]．これは，対象領域を含む正方形画像の4分割を，分割された画像が対象領域に含まれるか，外になるまで再帰的に行ない，その過程を木構造で表わしたものである．このような画像の階層化表現は，高速な検索などの利点をもつが，4分木をポインタで実現することは，大きなデータ量を必要とする．この点を考慮して4分木の実現法が数多く提案されている．そのひとつに，4分木を根から葉へのパスの集合として実現するquadcode[2]，リニア4分木[3]がある．本稿で提案するPコードは，4分木による画素(最小の画像要素)の表現をquadcodeと同様に符号化したものである．quadcodeはパスを列として扱わざるを得ないのに対して，Pコードは整数として扱うことができるため，大小比較，加減算，ビット演算等が可能である．

以下では，まず，Pコードの定義と4分木との関係を与え，Pコードによる集合演算について述べる．そして，Pコード上の演算について考察し，ベクトルの加減算等がPコード上で簡単に実現できることを示す．さらに，その応用例についても述べる．

2 Pコードの定義

画素はこれ以上分割できない最小の正方形であり，その一辺の長さを1とする．また，画像は$2^N \times 2^N$の画素からなるものとする．通常，画素はx座標，y座標の組(x,y)で表現され，白(0)または黒(1)の画像データを持つ．黒の値を持つ画素の集合が領域である．Pコードは画素を1個の整数に符号化したものである．

以下では，整数zの2進数表現の第iビットを表わすために，$< z >_i$なる記法を用いる．

定義 2.1 *Pコード (pixel-code)*は，(x,y)で表わされる画素を次のような関数Pで符号化したものであり，$2(N+1)$ビットの整数である．

$$P(x,y) = \sum_{i=0}^{N} < x >_i 2^{2i} + \sum_{i=0}^{N} < y >_i 2^{2i+1}$$

(注) 負の座標を扱うために2^{N+1}を法とする座標系を考えている．詳細は5節で述べる．

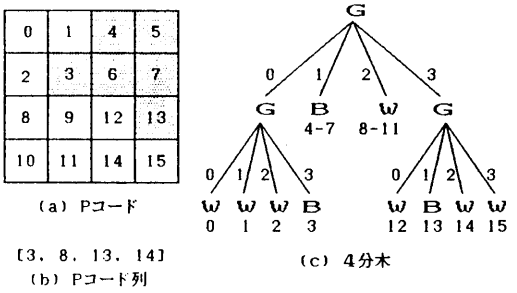

図1 Pコードと4分木

定義 2.2 Pコードpから，x座標，y座標を得る関数を，それぞれ，$X(p)$, $Y(p)$とする．

$$X(p) = \sum_{i=0}^{N} < p >_{2i} 2^i$$
$$Y(p) = \sum_{i=0}^{N} < p >_{2i+1} 2^i$$

直観的には，Pコードの偶数ビットがx座標，奇数ビットがy座標を表わす．図1(a)に$2^2 \times 2^2$の画像についての各画素のPコードを示す．例えば，画素$(2,1)$の座標を2進数表現すると$(10_{(2)}, 01_{(2)})$であり，そのPコードの2進数表現は$0110_{(2)}$，10進数表現は6となる．

領域の表現は，Pコードが整数であり大小関係がつけられることを利用する．

定義 2.3 Pコードの組$[p_1, p_2]$$(p_1 < p_2)$は，$P$コードの集合$\{p \mid p_1 \leq p < p_2\}$を表わす．更に，偶数個の$P$コードをソートした列$ps = [p_1, p_2, \ldots, p_{2L}]$$(p_{2i-1} < p_{2i} \leq p_{2i+1})$を$P$コード列*(pcode-sequence)*といい，これは，$\bigcup_{i=1}^{L}[p_{2i-1}, p_{2i}]$を表わす．$L$を$P$コード列の長さといい，$L(ps)$で表わす．

ここで，Pコード列の要素間の関係として，$p_{2i} \leq p_{2i+1}$としているが，列の長さを最小にするため，通常は，不等号<が成立しているものとする．図1(b)に，図1(a)の斜線部分の領域を表わすPコード列を示す．

3 4分木との関係

2次元領域に対して，よく用いられる階層的データ構造のひとつに4分木(quadtree)がある．4分木は次のようにして作られる．まず，$2^N \times 2^N$の画像を再帰的に$2^i \times 2^i$の正方画像に4分割していく．この分割

は，分割された正方画像の画素がすべて黒(1)になるか，すべて白(0)になったときに停止する．分割される正方画像を親，分割された1/4の大きさの正方画像を子として木を構成したものが4分木である．図1(c)に図1(a)の斜線部分の領域を4分木表現したものを示す．4分木のノードは$2^i \times 2^i$の正方画像に対応しており，これをqudrantという．また，4分木において，その葉は黒か白の値を持つ．黒の葉をBノード，白の葉をWノードという．葉でないノードはGノードと呼ばれる．

3.1 Pコードとquadrant

Pコードを4進数として見ると，4分木において4つの子に0, 1, 2, 3と番号付けしたときの根から葉へのパスと一致している．このようなパス表現をquad-codeという．例えば，図1(a)の13という画素は，その4進数表現は$31_{(4)}$であり，対応する4分木(図1(b))の葉13へのパスと一致している．このようにPコードは，画素を4分木のパスで表現したものに対応する．以下では，quadrantとPコードの関係について述べる．

定義 3.1 *Pコードpについて，$smax(p)$を次のように定義する．*

$$smax(p) = max\{i \mid p \wedge (0)^*(11)^i = 0\}$$

ここで，$\wedge$はビット毎の論理積演算であり，$(0)^*(11)^i$は，適当な数の0の後ろに1が$2i$個並んだ2進数を表わす．すなわち，$smax(p)$は，pの下位$2i$ビットがすべて0であるような最大のiである．

Pコードとquadrant(4分木のノード)の間には次の補題が成り立つ．

補題 3.1 *pをPコードとし，$s \leq smax(p)$とする．このとき，$r = [p, p + 4^s]$について次式が成り立つ．このようなrを$quadrant$という．*

$$q \in r \quad \Leftrightarrow \quad X(p) \leq X(q) < X(p) + 2^s \\ \& \; Y(p) \leq Y(q) < Y(p) + 2^s$$

[証明] 付録に与える．

この補題は，4分木のノードに対応する正方領域をPコードの組で表現できることを示している．4分木からPコード列への変換は，木を左優先でたどりながら，Bノードに対応するPコードの組(quadrant)を生成していけばよい．一方，一般のPコードの組は，quadrantでなく，複数のquadrantを含む．以下では，Pコードの組$[p_1, p_2]$からquadrantの列を生成することを考える．

```
δ = p2 − p1 ;
t = min{i | δ ∧ (11)*(00)^i = 0} − 1 ;
p0 = p1 ∧ (11)*(00)^t + 4^t ;
δ1 = p0 − p1; q1 = p1;
for ( i = 0; i < t; i++ ) {
    k = δ1 ∧ (00)*(11)(00)^i >> 2i ;
    for ( j = k; j ≤ 0; j−− ) {
        q2 = q1 + 4^i; put(q1, q2); q1 = q2;
    }
}
δ2 = p2 − p0;
for ( i = t; i ≥ 0; i−− ) {
    k = δ2 ∧ (00)*(11)(00)^i >> 2i ;
    for ( j = k; j ≤ 0; j−− ) {
        q2 = q1 + 4^i; put(q1, q2); q1 = q2;
    }
}
```

図2 quadrantへの変換

$\delta = p_2 - p_1$とすると，$\delta = \sum k_i 4^i$ $(k_i = 0, 1, 2, 3)$と表現でき，δは対象領域内の画素の個数である．従って，$2^i \times 2^i$のquadrantをk_i個ずつ生成すればよい．しかしながら，$smax(p) < i$である場合，$[p, p + 4^i]$はquadrantとはならないため，その生成順序が問題となる．そこで，次のようにしてquadrantを生成する(図2参照)．

tを$4^t \leq \delta < 4^{t+1}$であるような整数とする．$p_0 = p_1 \wedge (11)^*(00)^t + 4^t$とすると，$p_1 < p_0 \leq p_2$，$smax(p_0) \geq t$が成り立つ．従って，$\delta_2 = p_2 - p_0 = \sum_{i=0}^{t} k_i^{(2)} 4^i$とすると，$smax(p_0 + \sum_{i=j}^{t} k_i^{(2)} 4^i) \geq j$であり，$p_0$から$p_2$の画素については，$\delta_2$の上位ビットから順次quadrantを生成すればよい．一方，p_1からp_0については，$\delta_1 = p_0 - p_1 = \sum_{i=0}^{t-1} k_i^{(1)} 4^i$とすると，$smax(p_1 + \sum_{i=0}^{j} k_i^{(1)} 4^i) \geq j$であるので，$\delta_1$の下位ビットから順次quadrantを生成すればよい．なお，このアルゴリズムの時間計算量はO(quadrantの個数)である．

3.2 データ量

前節で述べたようにPコードと4分木には，密接な関係がある．ここでは，Pコード列と4分木のデータ量について述べる．4分木のノード数について次の式が知られている．

性質 3.1 n, n_B, n_W, n_Gを，それぞれ，4分木のノード数，Bノード数，Wノード数，Gノード数とすると，次の式が成り立つ．

$$n \leq (4^{N+1}-1)/3$$
$$n_B + n_W = (3n+1)/4$$
$$n_G = (n-1)/4$$

Pコード列の長さと4分木のノード数の間には次の補題が成り立つ．

補題 3.2 2次元領域Rの4分木表現とPコード列表現psを考える．n_B, n_Wを，それぞれ，4分木表現のBノードの個数，Wノードの個数とすると，psの長さについて，次の式が成り立つ．

$$L(ps) \leq (n_B + n_W + 1)/2$$

[証明] 補題3.1より，psの長さは，Bノードの個数より小さい．すなわち，$L(ps) \leq n_B$が成り立つ．

一方，2次元領域Rの補集合$\overline{R}$を考え，そのPコード列表現を$\overline{ps}$とすると，$\overline{R}$の4分木表現におけるBノードの個数はn_Wであり，補題3.1より$L(\overline{ps}) \leq n_W$が成り立つ．$ps$と$\overline{ps}$の長さは高々1しか違わないので，$L(ps) \leq L(\overline{ps}) + 1 \leq n_W + 1$が成立する．

以上の議論($L(ps) \leq n_B$, $L(ps) \leq n_W + 1$)より，$L(ps) \leq min(n_B, n_W + 1) \leq (n_B + n_W + 1)/2$が成立する． (証明終)

Pコード列を$m_p = 2(N+1)$ビットの整数の配列で実現することにすると，性質3.1と上記の補題より，次の定理が成り立つ．

定理 3.1 2次元領域の4分木表現のノード数をnとし，$m_p = 2(N+1)$ビットの整数でPコードを実現するとする．このとき，Pコード列の長さLとデータ量M(ビット)について次の性質が成り立つ．

$$L \leq (3n+5)/8,$$
$$M \leq m_p(3n+5)/4 = (N+1)(3n+5)/2$$

実現法	データ量(ビット)	
ポインタ型	$(4M_p+1)n$	$(8N+5)n$
改良ポインタ型	$(M'_p+1)(n-1)$	$2N(n-1)$
DF表現	$2n$	
修正DF表現	$(M_p+1)n$	$(2N+4)n$

ポインタの大きさは，$M_p = 2N+1$, $M'_p = 2N-1$としている．

表1 4分木の実現に必要なデータ量

4分木を実現するためのデータ構造が数多く提案されている．最も単純なものは，4分木をそのままポインタを用いて実現したもので，ポインタ型4分木と呼ばれる．同じくポインタを用いるもので，子の情報を親に持たせることにより4分木の葉を削除し，そのメモリ量を約1/4に削減した改良ポインタ型4分木がある[4]．一方，ポインタを用いないものとしてDF表現がある[1]．これは，4分木のノードを深さ優先の順序で並べたもので必要なメモリ量は最も少ない．しかしながら，アクセス効率がよくないため，効率を上げるために各ノードに子孫の個数等を保持することもある．表1にそれぞれの実現法で必要となるデータ量を示す．表1と定理3.1を較べるとPコードの方がデータ量が少ないことがわかる．

4 集合演算とPコード列の生成

4.1 集合演算

画素が領域に属するかどうかを探索する問題(member)，領域の補集合，共通集合，和集合を求める問題について考える．これらの問題については，Pコード列がソートされた列であることを利用する．member問題については，2分探索法により$O(\log L)$で探索が可能である．また，共通集合，和集合については，2つのソートされた列をマージすることにより，$O(L_1 + L_2)$のアルゴリズムを作ることができる(L, L_1, L_2はPコード列の長さ)．補集合については，列の先頭と終端に0，4^Nを挿入あるいは削除することにより，$O(1)$のアルゴリズムが作成できる．

図3に，共通集合の計算時間についての実験結果を，Pコードと改良されたポインタ型4分木について示す(Sun3/60, C言語)．横軸は生成された共通集合の大きさで，4分木のノード数に換算したものである．全体としては，係数が小さいPコードの方が優れている．

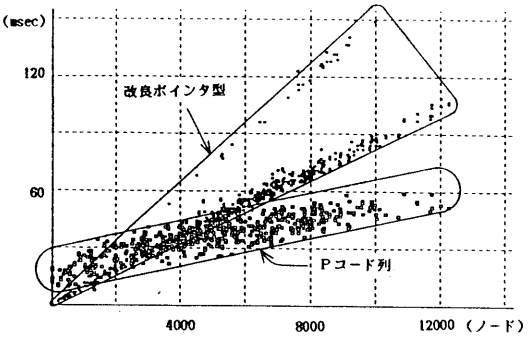

図3　共通集合の計算時間

なお，これらのアゴリズムの時間計算量は，定理3.1より$O(L) = O(4$分木のノード数$)$であるため，4分木の他の実現法についてのものと(補集合以外は)オーダとして変わらない．

4.2　Pコード列の生成

$2^N \times 2^N$の画像のラスタ表示(2次元配列)から，Pコード列を生成する手順について述べる．ラスタ表示であるため，各行を左$(0, y)$から右$(2^N - 1, y)$へ走査しながら，y方向に処理を進めなければならない．基本的な考えは，各行において画素のPコードが左から右に昇順になっていることから，各行毎にPコード列を生成し，それらの和集合をとるものである．このとき，計算時間を効率化するために，画像全体を再帰的にy方向に$1/2^i$に分割して処理を行なう．図4にそのアルゴリズムを示す．$mkps(y_0, y_1)$はy_0から$y_1 - 1$までの行に対応するPコード列を返し，$mkps_row$は1行のPコード列を返す関数である．

定理 4.1 $2^N \times 2^N$の画像に対して，$mkps(0, 2^N)$を実行したときの時間計算量は$O(N4^N)$である．

[証明] $T(y_0, y_1)$を$mkps(y_0, y_1)$の時間計算量とする．$L(y_0, y_1)$をy_0から$y_1 - 1$までの行に対応するPコード列の長さとする．このとき，$union$の計算時間はPコード列の長さの和に比例するので，$i > 0$に対して，

$$T(0, 2^i) = 2T(0, 2^{i-1}) + 2c_U L(0, 2^{i-1})$$

が成立する．$2^j \times 2^j$の画像のPコード列の長さは，補題3.2，性質3.1より$(3n + 5)/8 = (4^j + 1)/2$以下であるので，$L(0, 2^j) \leq ((4^j + 1)/2)(2^N/2^j) = (2^{N+j} + 2^{N-j})/2$である．従って，

$$T(0, 2^j) = 2T(0, 2^{j-1}) + c_U(2^{N+j-1} + 2^{N-j+1})$$

である．$T(0, 1) = c2^N$として，$j = N$について計算すると，

$$T(0, 2^N) = (c + 2c_U/3 + c_U N/2)4^N - 2c_U/3$$

となり，$T(0, 2^N) = O(N4^N)$が示される．　(証明終)

```
mkps(y0, y1) {
    if y0 + 1 = y1
        mkps_row(y0) ;
    else
        union(mkps(y0, (y0 + y1)/2),
              mkps((y0 + y1)/2, y1));
}
```

図4　Pコード列の生成アルゴリズム

5　Pコードの演算

領域の移動，隣接関係，外周の計算などは，4分木では効率の良いアルゴリズムを作ることが難しい．これは，これらのアルゴリズムが，x座標，y座標の算術演算や比較を基礎としているためである．原理的にはPコードをx(y)座標に変換してそのような演算を実行すればよいが，その変換には$O(N)$の時間計算量が必要となり，無駄が多い．そこで，Pコードをx(y)座標に変換せずに，Pコード上で加減算，比較を行なうことを考える．

減算を行なうために，負の座標を扱うことが必要である．そこで，$0 \leq x, y < 2^N$を正の座標，$2^N \leq x, y < 2^{N+1}$を負の座標として，2^{N+1}を法とする加減算を行なう．以下では，x座標，y座標の演算は2^{N+1}を法とする演算とする．また，Pコードについては$2^{2(N+1)}$を法とする演算とする．正の座標xの逆数$-x$は，$2^{N+1} - x$であり，これは$N+1$ビットの2進表現の場合xの補数となる．

以上のようにx(y)座標を定めた上で，Pコード上の加算$\oplus$と減算$\ominus$を次のように定義する．

定義 5.1 p_1, p_2をPコードとする．このとき，Pコー

ド上の加算⊕と減算⊖は次のように定義される.

$$p_1 \oplus p_2 = P(X(p_1) + X(p_2), Y(p_1) + Y(p_2))$$
$$\ominus p_1 = P(-X(p_1), -Y(p_1))$$

⊕, ⊖は, ベクトルの加算, 減算に対応している. 以下の議論では, x 座標と y 座標を独立に扱うため, x(y) 座標に対応する P コードを与える関数$P_X(P_Y)$を次のように定義する.

定義 5.2

$$P_X(x) = P(x, 0) = \sum_{i=0}^{N} < x >_i 2^{2i}$$
$$P_Y(y) = P(0, y) = \sum_{i=0}^{N} < y >_i 2^{2i+1}$$

次の性質が成り立つ.

性質 5.1 x, y, pをそれぞれ, x座標, y座標, Pコードとする.

$$P(x, y) = P_X(x) \vee P_Y(y)$$
$$P_X(X(p)) = p \wedge (01)^*$$
$$P_Y(Y(p)) = p \wedge (10)^*$$

補題 5.1 x_1, x_2, y_1, y_2を, それぞれx座標, y座標とすると, 次の式が成り立つ.

$$P_X(x_1 + x_2) = (P_X(x_1) + P_X(x_2) \vee (10)^*) \wedge (01)^*$$
$$P_Y(y_1 + y_2) = (P_Y(y_1) + P_Y(y_2) \vee (01)^*) \wedge (10)^*$$

[証明] 付録に与える.

補題 5.2 x, yをそれぞれ, x座標, y座標とすると, その逆数について次の式が成り立つ.

$$P_X(-x) = (-P_X(x)) \wedge (01)^*$$
$$P_Y(-y) = (-P_Y(y)) \wedge (10)^*$$

[証明] 付録に与える.

定理 5.1 p_1, p_2をPコードとしたとき, その加減算は次の式で計算される.

$$p_1 \oplus p_2 =$$
$$(p_1 \wedge (01)^* + p_2 \vee (10)^*) \wedge (01)^*$$
$$\vee (p_1 \wedge (10)^* + p_2 \vee (01)^*) \wedge (10)^*$$
$$p_1 \ominus p_2 =$$
$$(p_1 \wedge (01)^* + (-(p_2 \wedge (01)^*) \vee (10)^*)) \wedge (01)^*$$
$$\vee (p_1 \wedge (10)^* + (-(p_2 \wedge (10)^*) \vee (01)^*)) \wedge (10)^*$$

[証明] ⊕については, 性質5.1より, $P_X(X(p_1)+X(p_2)) = (p_1 \wedge (01)^* + p_2 \vee (10)^*) \wedge (01)^*$, $P_Y(Y(p_1)+Y(p_2)) = (p_1 \wedge (10)^* + p_2 \vee (01)^*) \wedge (10)^*$を示せばよい. 補題5.1, 5.2を用いることにより, 次の式が成り立つ.

$$P_X(X(p_1) + X(p_2))$$
$$= (P_X(X(p_1)) + P_X(X(p_2)) \vee (10)^*) \wedge (01)^*$$
$$= (p_1 \wedge (01)^* + (p_2 \wedge (01)^*) \vee (10)^*) \wedge (01)^*$$
$$= (p_1 \wedge (01)^* + p_2 \vee (10)^*) \wedge (01)^*$$

P_Yについても同様の式が成り立つ. また, ⊖についても補題5.2を用いることによって, 同様に証明できる. (証明終)

上記の定理により, 移動などの計算において最も重要なベクトルの加算をP コード上で行なうことができる. また, 比較についても次のような関係が成立するので, x 座標, y 座標についての性質をP コード上の演算で調べることが可能になる.

性質 5.2 x_1, x_2をx座標とすると, 次の関係が成立する. y座標についても同様の関係が成立する.

$$x_1 = x_2 \quad \Leftrightarrow \quad P_X(x_1) = P_X(x_2)$$
$$x_1 < x_2 \quad \Leftrightarrow \quad P_X(x_1) < P_X(x_2)$$

6 応用例

6.1 隣接関係

2 つの quadrant r_1=[p_1, q_1], r_2=[p_2, q_2]が隣接しているかどうかを調べる問題を考える. x(y) 座標方向で接している条件を$Adj_X(Adj_Y)$とし, 重なっているための条件を$Overlap_X(Overlap_Y)$とすると, 隣接のための条件は,

$$(Adj_X \; and \; Overlap_Y) \quad or \quad (Overlap_X \; and \; Adj_Y)$$

となる. $Adj_X, Adj_Y, Overlap_X, Overlap_Y$は次のように記述できる.

quadrant r=[p, q]の 1 辺の長さを$size(r)$で表わすことにし, 簡単のため, $X(p_1) \leq X(p_2)$, $Y(p_1) \leq Y(p_2)$とすると,

$$Adj_X : \quad X(p_2) = X(p_1) + size(r_1)$$
$$Adj_Y : \quad Y(p_2) = Y(p_1) + size(r_1)$$
$$Overlap_X : \quad X(p_2) < X(p_1) + size(r_1)$$
$$Overlap_Y : \quad Y(p_2) < Y(p_1) + size(r_1)$$

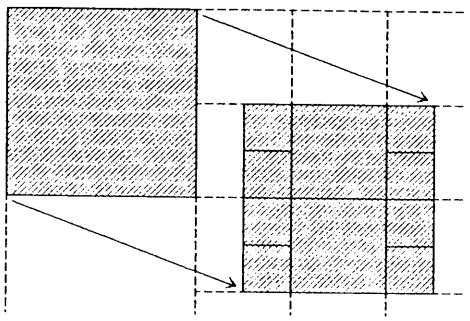

図5 quadrantの移動

である．例えば，Adj_XをPコード上で調べるためには，$P_X(X(p_2)) = P_X(X(p_1) + size(r_1))$を調べればよい．$P_X(size(r_1)) = q_1 - p_1$なので，前節の結果を用いるとこの等式は次のようになる．

$$p_2 \wedge (01)^* = (q_1 - p_1 + q_1 \vee (10)^*) \wedge (01)^*$$

他の条件についても，上式のようなPコード上のビット演算，加減算で表現できる．

画素pと領域Rの隣接関係については，$p \notin R$と，pの隣接画素$(p \oplus 1,\ p \ominus 1,\ p \oplus 2,\ p \ominus 2)$の少なくとも1個が$R$に含まれることを調べればよい．これは，4.1節のmember問題であるので，Rを表現するPコード列の長さをLとすると，時間計算量は$O(\log L)$である．一般に，領域と領域のの隣接関係についても，共通集合が空であることと，一方の領域の隣接画素と他方の領域との共通集合を調べればよい．

4分木表現における多くのアルゴリズムが，隣接するquadrantを求めることをベースとしているため，隣接quadrantを求める問題は重要である[1]．Pコードの場合は，座標毎の加減が簡単にできることため，この問題は容易に計算できる．例えば，r=$[p, p+4^s]$の右に隣接する同じ大きさののquadrantは，$p' = p \oplus 4^s$とすると，$[p', p'+4^s]$であり(6.2参照)，隣接する画素は，$p' \oplus P_Y(y)\ (0 \le y < 2^s)$である．

6.2 移動

5節のPコードの加減算を用いて，領域を，x方向にδ_x，y方向にδ_yだけ移動することを考える．$\delta = P(\delta_x, \delta_y)$とする．画素$p$の移動はベクトルの和が計算されればよいので，定理5.1に従って，$p \oplus \delta$を計算すればよい．

領域の移動はquadrantの移動を繰り返せばよいので，ここでは，quadrant $r = [p, p+4^s]$ $(s \le smax(p))$の移動について述べる．quadrantの場合，単純に$[p \oplus \delta, (p \oplus \delta) + 4^s]$としても，これがquadrantになるとは限らない(図5)．しかし，$s \le smax(\delta)$の場合は，$[p \oplus \delta, (p \oplus \delta) + 4^s]$が移動先の正方領域となる．これは，補題3.1，$\oplus$の定義，および$smax(p \oplus \delta) \ge min\{smax(p), smax(\delta)\} \ge s$から簡単に示される．

一方，$s < smax(\delta) = s_0$の場合は，rを

$$r_i = [p + i4^{s_0}, p + (i+1)4^{s_0}]\ (0 \le i < 4^{s-s_0})$$

に分割し，各r_iについて移動を行なえばよい．この場合，移動後のquadrantの列がソートされているとは限らないため，ソートする必要がある．

上で述べた移動の手順は簡便なものであるが，ソートをしないですむ方法がある．これは，対象となる矩形を適当な基点で4分割し，左上，右上，左下，右下の各象限について分割された矩形をquadrantの列に変換するものである．以下ににそのアルゴリズムを示す．アルゴリズムは理解し易さのため，Pコードではなく，x，y座標で記述してあるが，これは容易にPコードの演算に変換できる．

```
move(p, s, δ) {
    /* 移動後の正方領域の端点の座標
       x1, x2, y1, y2の計算 */
    x1 = X(p) + X(δ); x2 = x1 + 2^s;
    y1 = Y(p) + Y(δ); y2 = y1 + 2^s;
    /* 分割の基点(x0, y0)の計算 */
    x0 = x1 ∧ (1)*(0)^s + 2^s;
    y0 = y1 ∧ (1)*(0)^s + 2^s;
    rect0(x0, y0, x0 - x1, y0 - y1);
    rect1(x0, y0, x2 - x0, y0 - y1);
    rect2(x0, y0, x0 - x1, y2 - y0);
    rect3(x0, y0, x2 - x0, y2 - y0);
}
rect0(x0, y0, x, y); {
    if (x = 0 or y = 0) return;
    /* 矩形内で最大のquadrantの
       大きさsの計算 */
    s = smax(P(x, y));
    /* 新しい基点の計算 */
    x0 = x0 - 2^s; y0 = y0 - 2^s;
    rect0(x0, y0, x - 2^s, y - 2^s);
```

$rect_1(x_0, y_0, 2^s, y - 2^s)$;
$rect_2(x_0, y_0, x - 2^s, 2^s)$;
$put(P(x_0, y_0), P(x_0, y_0) + 4^s)$;
}
$rect_{1,2,3}$は省略.

7 おわりに

2値画像の符号化Pコードを提案し，4分木との関係を明らかにした．また，そのデータ量，集合演算がこれまでの4分木の実現法よりも優れていることを示した．さらに，座標毎の加減や比較がPコード上のビット演算，算術演算で可能であることを示し，その応用についても触れた．Pコードの利点は，算術演算が可能であることであり，この性質を利用して，多くの問題が見通しよく解決できるであろう．また，3次元への拡張も自明であり，画素数の多い3次元物体の表現にはPコードは有用であると考える．

参考文献

[1] H. Samet, The quadtree and related hierarchical data structure, Comput. Surv., 16, 2, (1984).
[2] S. X. Li and M. H. Loew, The quadcode and its arithmetic, CACM, 30, 7, (1987).
[3] I. Gargantini, An effective way to represent quadtrees, CACM, 25, 12, (1982).
[4] 大川原，清水，Region Quadtreeのデータ量削減と集合演算アルゴリズム，第36回情処全大，4B-9，(昭63).
[5] 坂内，大沢，画像データベース，昭晃堂，(昭62).

付録

A.1 補題3.1の証明

[⇒の証明] $q \in [p, p+4^s]$, $\delta = q - p$とする．$\delta < 4^s$であり，pの下位$2s$ビットは0であるので，

$$\begin{aligned} X(q) &= X(\delta + p) \\ &= X(\sum_{i=0}^{2s-1} <\delta>_i 2^i + \sum_{i=2s}^{2N+1} <p>_i 2^i) \\ &= \sum_{i=0}^{s-1} <\delta>_{2i} 2^i + \sum_{i=s}^{N} <p>_{2i} 2^i \\ &= X(\delta) + X(p) \end{aligned}$$

である．$0 \le X(\delta) < 2^s$は明らかなので，

$$X(p) \le X(q) < X(p) + 2^s$$

が成立する．$Y(q)$についても同様.

[⇐の証明] $X(p) \le x < X(p) + 2^s$, $Y(p) \le y < Y(p) + 2^s$とする．$\delta_x = x - X(p)$, $\delta_y = y - Y(p)$とすると，$0 \le \delta_x, \delta_y < 2^s$であり，また，$p$の下位$2s$ビットは0であるので，

$$\begin{aligned} P(x,y) &= P(X(p) + \delta_x, Y(p) + \delta_y) \\ &= \sum_{i=s}^{N} <p>_{2i} 2^{2i} + \sum_{i=0}^{s-1} <\delta_x>_i 2^{2i} \\ &\quad + \sum_{i=s}^{N} <p>_{2i+1} 2^{2i} + \sum_{i=0}^{s-1} <\delta_y>_i 2^{2i+1} \\ &= p + P(\delta_x, \delta_y) \end{aligned}$$

である．$0 \le P(\delta_x, \delta_y) < 4^s$は明らかなので，

$$p \le P(x,y) < p + 4^s$$

が成立する． (証明終)

A.2 補題5.1の証明

$x = x_1 + x_2$とする．c_i $(0 \le i \le N+1)$をiビット目のキャリア，すなわち，次式のように定義する．

$$\begin{aligned} c_0 &= 0 \\ c_{i+1} &= <x_1>_i + <x_2>_i + c_i \ \ (div\ 2) \end{aligned}$$

$<x>_i = <x_1>_i + <x_2>_i + c_i \pmod 2$なので，$<x_1>_i + <x_2>_i + c_i = 2c_{i+1} + <x>_i$が成立し，この式を用いて，$P_X(x_1) + P_X(x_2) \vee (10)^*$を次のように変形する．

$$\begin{aligned} & P_X(x_1) + P_X(x_2) \vee (10)^* \\ &= \sum_{i=0}^{N} <x_1>_i 2^{2i} + \sum_{i=0}^{N} <x_2>_i 2^{2i} + \sum_{i=0}^{N} 2^{2i+1} \\ &= \sum_{i=0}^{N} (<x>_i + 2c_{i+1} - c_i) 2^{2i} + \sum_{i=0}^{N} 2^{2i+1} \\ &= \sum_{i=0}^{N} <x>_i + \sum_{i=0}^{N} (1 - c_{i+1}) 2^{2i+1} + c_{N+1} 2^{2(N+1)} \\ &\quad - c_0 \end{aligned}$$

$c_0 = 0$, $c_{N+1} 2^{2(N+1)} = 0 \pmod{2^{2(N+1)}}$であるので，

$$(P_X(x_1) + P_X(x_2) \vee (10)^*) \wedge (01)^* = \sum_{i=0}^{N} <x>_i 2^{2i}$$

が成立する．P_Yについても同様. (証明終)

A.3 補題5.2の証明

$-x = 2^{N+1} - x = 1 + \sum_{i=0}^{N} (1 - <x>_i) 2^i$であるので，補題5.1を用いると，$P_X(-x)$は次のように変形される．

$$\begin{aligned} & P_X(-x) \\ &= P_X(1 + \sum_{i=0}^{N} (1 - <x>_i) 2^i) \\ &= (P_X(1) + P_X(\sum_{i=0}^{N} (1 - <x>_i) 2^i) \vee (10)^*) \wedge (01)^* \\ &= (1 + \sum_{i=0}^{N} (1 - <x>_i) 2^{2i} + \sum_{i=0}^{N} 2^{2i+1}) \wedge (01)^* \\ &= (1 + \sum_{i=0}^{2N+1} 2^i - \sum_{i=0}^{N} <x>_i 2^{2i}) \wedge (01)^* \\ &= (2^{2(N+1)} - P_X(x)) \wedge (01)^* \\ &= -P_X(x) \wedge (01)^* \end{aligned}$$

P_Yについても同様. (証明終)