5
Operações Eficientes em Inteiros

Como as chaves são números inteiros, algumas operações em inteiros podem ser usadas para acelerar a geração, a manipulação e a conversão de chaves. Dilatação, contração e código adicional local são algumas das operações em inteiros usadas na geração, na manipulação e na conversão de chaves. Schrack em (20) dá uma boa definição de inteiros dilatados e define, além de outras coisas, operações de adição e subtração em inteiros dilatados em *quadtrees lineares*. Com tais operações ele define a operação de adição em códigos locais (seção 2.4.2) em *quadtrees lineares* para encontrar o código local dos nós adjacentes de mesmo nível. Este capítulo apresenta de forma geral inteiros dilatados e contraídos, operações eficientes em inteiros e operações em códigos de Morton, tudo para uma 2^d–*tree* num espaço d-dimensional, utilizando a mesma notação de operações de máquina do capítulo anterior.

5.1
Dilatação e Contração

Dilatação e contração de inteiros são operações usadas em conjunção com sistemas de indexação do espaço, auxiliando na conversão entre a chave e a posição espacial de um elemento multidimensional. Dilatação de inteiros é o processo de inserção de zeros no conjunto de bits que compõe o inteiro na base binária, dependendo do tipo da ordenação espacial. Contração de inteiros é o processo de remoção desses zeros. Esses zeros são inseridos ou removidos através de operações binárias nos inteiros, para o processo de intercalar ou concatenar o inteiro, com o objetivo de formar uma chave. Essas operações dependem da dimensão do espaço e também da precisão da conversão do número para a base binária. Existem muitas maneiras de efetuar tais operações nos inteiros: uma maneira simples, mas não eficiente, é operar cada bit do número inteiro por vez, a qual precisaria de muitas operações. Stocco e Schrack em (19) apresentaram uma maneira muito eficiente de dilatar e contrair inteiros para uma *quadtree linear*. Essa mesma idéia será descrita aqui, mas de forma mais geral, e para uma 2^d–*tree* num espaço multidimensional.

5.1.1 Definições

Seja um inteiro $q = (q_r q_{r-1} \ldots q_1 q_0)_2$ que representa a conversão na base binária com precisão r de um número pertencente ao intervalo [0, 1] onde $q_i = 0$ ou $1 \ \forall i = 0, \ldots, r$. O inteiro $q_{dil} = (q_r 0^{d-1} q_{r-1} 0^{d-1} \ldots 0^{d-1} q_1 0^{d-1} q_0)_2$ é chamado de *dilatação* de q. O inverso dessa operação, que é a remoção dos elementos 0^{d-1} de um inteiro dilatado resultando no inteiro original, é chamada de *contração*. Obviamente, as operações $q_{dil} = dil(q)$ e $q = con(q_{dil})$ são operações inversas uma da outra, isto é, $q_{dil} = dil(con(q_{dil}))$ e $q = con(dil(q))$.

Os processos de dilatação e contração envolvem inteiros auxiliares, chamados de *máscaras*, que são usados para filtrar elementos inconvenientes resultantes das operações usadas para dilatar ou contrair. Necessita também de posições auxiliares, chamadas de *deslocamentos*, que são posições estratégicas para posicionar partes do inteiro original a ser filtrado.

5.1.2 Máscaras

As máscaras usadas para a dilatação e contração de um inteiro que representa a conversão na base binária de um número do hipercubo d-dimensional são da seguinte forma:

1. A primeira máscara possui 1 nas posições 0, d, $2d$, ..., jd, ... até completar o armazenamento e zero nas outras.

2. A segunda máscara possui 1 nas posições 0 até d, d^2 até $d^2 + d$, $2d^2$ até $2d^2 + d$, ..., jd^2 até $jd^2 + d$, ... , até completar o armazenamento e zero nas outras.

3. A i-ésima máscara possui 1 nas posições 0 até d^{i-1}, d^i até $d^i + d^{i-1}$, $2d^i$ até $2d^i + d^{i-1}$, ..., jd^i até $jd^i + d^{i-1}$, ... , até completar o armazenamento e zero nas outras.

A seqüência de máscaras segue até a m-ésima máscara, interrompendo-se quando d^m for maior que a quantidade de armazenamento. Por exemplo, para um armazenamento de 64 bits, no caso 2D temos seis máscaras, em 3D temos quatro máscaras e em 4D são apenas três máscaras. Essas máscaras estão escritas no formato hexadecimal na tabela abaixo:

2D	3D	4D
0x5555555555555555	0x9249249249249249	0x1111111111111111
0x3333333333333333	0x01C0E070381C0E07	0x000F000F000F000F
0x0F0F0F0F0F0F0F0F	0x7FC0000FF80001FF	0x0000000000000FFFF
0x00FF00FF00FF00FF	0x0000000007FFFFFF	
0x0000FFFF0000FFFF		
0x00000000FFFFFFFF		

5.1.3 Deslocamentos

Para as operações de dilatação e contração, antes de aplicar as máscaras é preciso replicar partes do inteiro, estrategicamente posicionadas. Assim, quando for filtrado pelas máscaras, devem aparecer as partes do inteiro nas posições desejadas. Tais posições auxiliares dependem da dimensão d do espaço e da precisão r do inteiro convertido na base binária. Conhecidas essas grandezas, existem $m = \lceil \log_d r \rceil$ grupos de deslocamento contendo $s = \lceil \frac{r}{d^{m-1}} \rceil - 1$ posições. São eles:

1. Primeiro grupo são as posições:
   $(d-1)$, $2(d-1)$, ..., $s(d-1)$.

2. Segundo grupo são as posições:
   $(d-1)d$, $2(d-1)d$, ..., $s(d-1)d$.

3. i-ésimo grupo são as posições:
   $(d-1)d^{i-1}$, $2(d-1)d^{i-1}$,..., $s(d-1)d^{i-1}$.

4. m-ésimo grupo são as posições:
   $(d-1)d^{m-1}$, $2(d-1)d^{m-1}$, ..., $s(d-1)d^{m-1}$.

Por exemplo, no caso 2D para um inteiro com precisão 32 bits temos cinco grupos com apenas uma posição; já no caso 3D para um inteiro com precisão 21 bits temos três grupos com duas posições cada; e no caso 4D para um inteiro com precisão 16 bits são dois grupos com três posições cada. Esses grupos e suas posições encontram-se na tabela abaixo:

2D	3D	4D
1		
2	2 4	3 6 9
4	6 12	12 24 36
16	18 36	
32		

5.1.4 Algoritmos

Algorithm 3 dil(q): faz a dilatação do inteiro q.

```
1: static d                                   // dimensão do espaço
2: static r                                   // precisão da conversão no inteiro
3: static m = ⌈log_d r⌉                       // número de máscaras menos um
4: static s = ⌈r/d^(m-1)⌉ − 1                 // número de deslocamentos
5: static shift[m+1][s]                       // vetor do grupo de deslocamentos
6: static mask[m+1]                           // vetor de máscaras
7: for j = m a 1 do
8:    for i = 1 a s do
9:       q |= (q ≪ shift[j][i])
10:   end for
11:   q &= mask[j]
12: end for
13: return q
```

Dado um inteiro que representa a conversão de precisão r de um número pertencente ao hipercubo d-dimensional, são necessários pelo menos $r * d$ bits para armazenar a sua dilatação. Então o número de bits n_{bits} usado para o armazenamento deve ser maior ou igual a $r * d$: $n_{bits} \geq r * d$. Para tal armazenamento, são usadas nas operações de dilatação e contração $m + 1$ máscaras e m grupos de deslocamento contendo s posições, como definido acima. Para dilatar são usadas as primeiras m máscaras da seguinte forma: primeiramente é aplicado o m-ésimo grupo de deslocamentos no inteiro e depois este é filtrado com a m-ésima máscara. Depois é aplicado o $(m-1)$-ésimo grupo de deslocamentos no inteiro e depois este é filtrado com a $(m-1)$-ésima máscara, e assim por diante, até aplicar o primeiro grupo de deslocamentos, e filtrando-o com a primeira máscara. O algoritmo 3 descreve em detalhes esse procedimento. Observe que s foi definido de forma que satisfaça as desigualdades: $(s+1)d^{m-1} < r \leq (s+2)d^{m-1}$. Veja o que acontece com o inteiro no processo de dilatação:

1. Inteiro Original:
   $1\ 0 \ldots 0\ b_r \ldots b_1$.

2. Passo 1: $[g = (d-1)d^{m-1}]$
   $1\ 0 \ldots 0\ b_{d^r} \ldots b_{(s+1)d^{m-1}+1} \ldots 0^g b_{2d^{m-1}} \ldots b_{d^{m-1}+1}\ 0^g b_{d^{m-1}} \ldots b_1$.

3. Passo 2: $[g = (d-1)d^{m-2}]$
   $1\ 0 \ldots 0\ b_{d^r} \ldots b_{(s+1)d^{m-1}+(d-1)d^{m-2}+1} \ldots 0^g b_{d^{m-1}+d^{m-2}} \ldots b_{d^{m-1}+1}$
   $0^g b_{d^{m-1}} \ldots b_{(d-1)d^{m-2}+1} \ldots 0^g b_{2d^{m-2}} \ldots b_{d^{m-2}+1}\ 0^g b_{d^{m-2}} \ldots b_1$.

4. Passo i: $[g = (d-1)d^{m-i}]$
   $1\ 0 \ldots 0\ b_{d^r} \ldots b_{(s+1)d^{m-i}+(d-1)d^{m-i-1}+1} \ldots 0^g b_{d^{m-i}+d^{m-i-1}} \ldots b_{d^{m-i}+1}$
   $0^g b_{d^{m-i}} \ldots b_{(d-1)d^{m-i-1}+1} \ldots 0^g b_{2d^{m-i-1}} \ldots b_{d^{m-i-1}+1}\ 0^g b_{d^{m-i-1}} \ldots b_1.$

5. Passo m: $[g = (d-1)]$
   $1\ 0 \ldots 0\ b_{d^r} \ldots 0^g b_{(s+1)d^{m-1}} \ldots 0^g b_{d^2}\ 0^g b_1.$

Para contrair são usadas as últimas m máscaras da seguinte forma: primeiramente é aplicado o primeiro grupo de deslocamentos no inteiro e depois este é filtrado com a segunda máscara. Depois é aplicado o segundo grupo de deslocamentos no inteiro e depois este é filtrado com a terceira máscara, e assim por diante, até aplicar o m-ésimo grupo de deslocamentos, filtrando-o com a $m+1$-ésima máscara. O algoritmo 4 descreve em detalhes esse procedimento.

Algorithm 4 con(q_{dil}): faz a contração do inteiro dilatado q_{dil}.

```
1: static d                                          // dimensão do espaço
2: static r                                          // precisão da conversão no inteiro
3: static m = ⌈log_d r⌉                              // número de máscaras menos um
4: static s = ⌈r/d^(m-1)⌉ − 1                        // número de deslocamentos
5: static shift[m+1][s]                              // vetor do grupo de deslocamentos
6: static mask[m+1]                                  // vetor de máscaras
7: for j = 2 a m + 1 do
8:    for i = 1 a s do
9:       q_dil |= (q_dil ≫ shift[j][i])
10:   end for
11:   q_dil &= mask[j]
12: end for
13: return q_dil
```

A Figura 5.1 exemplifica as etapas do processo de dilatação, e na ordem contrária o processo de contração. A Figura 5.1(a) mostra em 2D, para uma precisão 8; já a Figura 5.1(b) mostra em 3D, para uma precisão 9; e a Figura 5.1(c) mostra em 4D, para uma precisão 8, onde os espaços vazios são zeros e as letras gregas simbolizam os bits da conversão.

5.1.5 Acelerando o Processo de Intercalação

O processo de intercalação dos bits para a geração da chave pelo método de códigos de Morton pode ser feito usando as operações de dilatação e contração em inteiros da seguinte forma: dada uma posição $p = (x_1, \ldots, x_d) \in [0,1]^d$ no hipercubo unitário do espaço d-dimensional e uma precisão m, a chave $k(p)$ é gerada a partir das conversões de precisão m das coordenadas

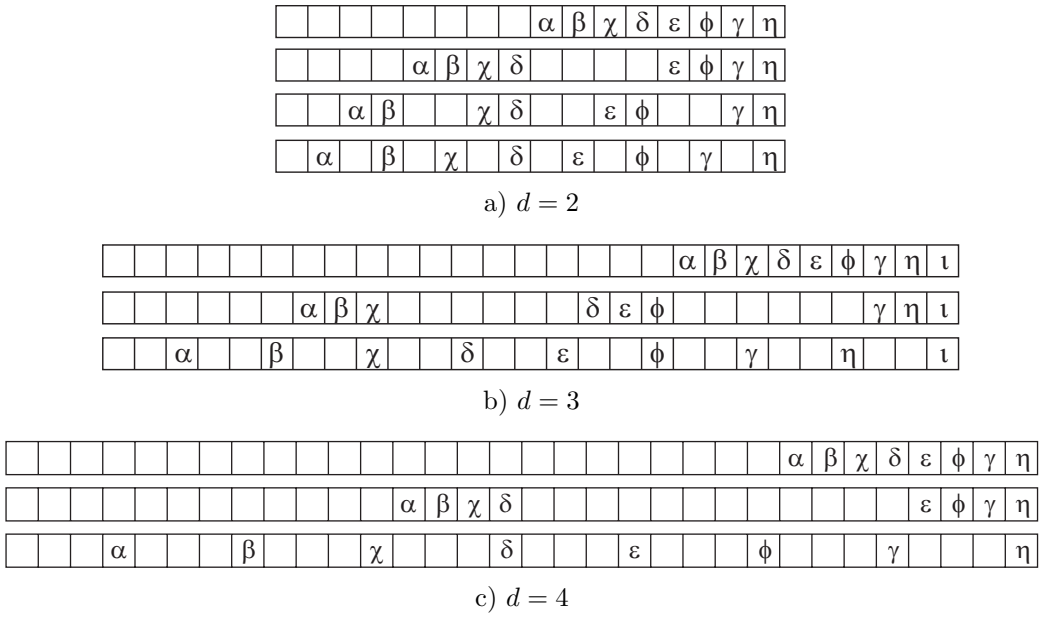

Figura 5.1: Etapas de dilatação e contração de inteiros.

de p na base binária $b^i = (b^i_m, \ldots, b^i_1)_2 = \lfloor x_i \times 2^m \rfloor$ para $i = 1, \ldots, d$ e usando operações de dilatação:

$$k(p) = (1b^d_m \ldots b^1_m \ldots b^d_2 \ldots b^1_2 \ldots b^d_1 \ldots b^1_1)_2 = \bigmid_{i=1}^{d} \big(dil(b^i) \ll (i-1)\big)$$

e o processo inverso de obter as coordenadas do ponto p com uma precisão m a partir de sua chave $k(p)$ utiliza operações de contração:

$$x_i = \frac{1}{2^m} \times con(k(p) \gg (i-1)).$$

5.2 Adição em Inteiros Dilatados

A operação de adição em inteiros dilatados evita conversões desnecessárias quando se está operando um inteiro. Operações em inteiros dilatados devem resultar também em inteiros dilatados, e para isso é preciso separar e filtrar os bits importantes do inteiro original após o uso dessas operações. Para isso, é preciso definir máscaras binárias para auxiliar essas operações. Dada uma precisão r, tais máscaras binárias são dadas por r blocos de tamanho d cada um, que é o tamanho de um inteiro dilatado de precisão r: $t_1 = (0 \ldots 01)^r, t_2 = (0 \ldots 10)^r, \ldots, t_{d-1} = (010 \ldots 0)^r$ e $t_d = (10 \ldots 0)^r$. As máscaras binárias opostas (negação) às anteriores que também auxiliam a operação de adição, basta trocar 0 por 1 e 1 por 0 nas máscaras anteriores: $\tilde{t}_1 = (1 \ldots 10)^r, \tilde{t}_2 = (1 \ldots 01)^r, \ldots, \tilde{t}_{d-1} = (101 \ldots 1)^r$ e $\tilde{t}_d = (01 \ldots 1)^r$, é fácil

ver que $t_i = t_{i-1} \ll 1$ e $\tilde{t}_i = \tilde{t}_{i-1} \ll 1$. Um caso particular é o espaço bidimensional, pois existem apenas duas máscaras, sendo uma oposta à outra. É mais intuitivo denotar a primeira máscara por t_x ao invés de t_1, a segunda máscara por t_y ao invés de t_2, a terceira máscara por t_z ao invés de t_3 e a quarta máscara por t_w ao invés de t_4, para dimensões menores. Veja as chaves no formato hexadecimal na tabela abaixo:

2D
$t_x = \tilde{t}_y$=0x5555555555555555
$t_y = \tilde{t}_x$=0xAAAAAAAAAAAAAAAA

3D
t_x=0x4924924924924924
t_y=0x2492492492492492
t_z=0x9249249249249249
$\tilde{t}_x$=0xB6DB6DB6DB6DB6DB
$\tilde{t}_y$=0xDB6DB6DB6DB6DB6D
$\tilde{t}_z$=0x6DB6DB6DB6DB6DB6

4D
t_x=0x1111111111111111
t_y=0x3333333333333333
t_z=0x4444444444444444
t_w=0x8888888888888888
$\tilde{t}_x$=0xEEEEEEEEEEEEEEEE
$\tilde{t}_y$=0xDDDDDDDDDDDDDDDD
$\tilde{t}_z$=0xBBBBBBBBBBBBBBBB
$\tilde{t}_w$=0x7777777777777777

A operação de adição em inteiros dilatados q_{dil} e p_{dil} de mesma precisão r é definida como

$$q_{dil} \oplus p_{dil} = \big((q_{dil} \mid \tilde{t}_1) + p_{dil}\big) \ \& \ t_1$$

onde o operador $\oplus$ representa a adição de inteiros dilatados. A adição é definida de forma que a soma $s_{dil} = q_{dil} \oplus p_{dil}$ respeite o princípio de unicidade da dilatação, preservando o inteiro original após operá-lo de forma dilatada. Então a soma deve satisfazer $con(s_{dil}) = q + p$ ou então $s_{dil} = dil(q + p)$. Um exemplo no espaço bidimensional: sejam os inteiros 7 e 6, suas conversões na base binária de precisão 4 são $7_2 = 0111$ e $6_2 = 0110$. Seja 10 o número de bits usado para a dilatação, então $dil(7) = 0000010101$ e $dil(6) = 0000010100$. Com as máscaras $t_x = \tilde{t}_y = 0101010101$ e $t_y = \tilde{t}_x = 1010101010$, temos $dil(7) \mid t_y = 1010111111$, logo $(dil(7) \mid \tilde{t}_1) + dil(6) = 1011010011$ e então $dil(7) \oplus dil(6) = ((dil(7) \mid t_y) + dil(6)) \ \& \ t_x = 0001010001 = dil(13)$.

5.3 Adição em Códigos de Morton

A adição em chaves geradas pelo método códigos de Morton pode ser feita separando as adições de cada coordenada das chaves e recombinando os resultados em uma nova chave resultante da soma das duas primeiras. Inspirado na operação de adição em inteiros dilatados, dadas as chaves k e h geradas pelo

método códigos de Morton num espaço d-dimensional de mesmo nível, a soma delas é:

$$k \underset{Morton}{\oplus} h = \bigmid_{i=1}^{d} (\{[(k \,\&\, t_i) \gg (i-1)] \oplus [(h \,\&\, t_i) \gg (i-1)]\} \ll (i-1))$$

onde o operador $\oplus_{Morton}$ representa a adição de chaves geradas pelo método códigos de Morton e o operador $\oplus$ representa a adição de inteiros dilatados. A parcela $(k \,\&\, t_i) \gg (i-1)$ remove da chave os bits relativos à coordenada i, montando um inteiro dilatado, para que possa ser adicionado com o outro inteiro dilatado da outra chave. A parcela final $\ll (i-1)$ recoloca o resultado da soma dos inteiros dilatados na posição certa relativa à coordenada i para a geração da chave resultante da soma que ainda é uma chave gerada pelo método de códigos de Morton. Substituindo a adição de inteiros dilatados temos a seguinte simplificação:

$$k \underset{Morton}{\oplus} h = \bigmid_{i=1}^{d} \left(\left[(k \mid \tilde{t}_i) + (h \,\&\, t_i) \right] \,\&\, t_i \right).$$

Para verificar tal simplificação note que na parcela $[(k \,\&\, t_i) \gg (i-1)] \mid \tilde{t}_1$ os bits relativos à i-ésima coordenada estão na primeira posição dos blocos de tamanho d da chave, e o resto preenchido por uns, e para a parcela $(k \mid \tilde{t}_i)$ os bits relativos à i-ésima coordenada estão na i-ésima posição dos blocos de tamanho d da chave, e o resto preenchido por uns, dispensando assim o uso de operações de deslocamento.

5.4 Chaves Adjacentes

Um nó pertencente a uma 2^{d}–*tree* num espaço d-dimensional, possui 3^d-1 direções de adjacência, são elas: $\Delta v = (v_1, \ldots, v_d)$ com $v_i \in \{-1, 0, 1\}$ $\forall i = 1, \ldots, d$ e com $v_i \neq 0$ para algum $i \in \{0, \ldots, d\}$. A tabela abaixo mostra essas direções no espaço bidimensional:

NO	N	NL
O	X	L
SO	S	SL

(-1, 1)	(0, 1)	(1, 1)
(-1, 0)	X	(1, 0)
(-1,-1)	(0,-1)	(1,-1)

Torna-se possível o cálculo em tempo constante das chaves dos nós adjacentes a um nó a partir de sua chave, utilizando-se a operação de adição de chaves geradas pelo método de códigos de Morton, satisfazendo assim o terceiro requerimento da seção 4.1.4. Para isto, basta adicionar à chave do nó uma das $3^d - 1$ chaves de mesmo nível que representam as direções

adjacentes. Dada a chave $k(n)$ de um nó n de uma 2^{d}–*tree*, a chave $k(n_{adj})$ de algum dos nós adjacentes n_{adj} de mesmo nível é gerada da seguinte forma: $k(n_{adj}) = k(n) \oplus_{Morton} \Delta k$, onde Δk simboliza a chave gerada no mesmo nível que n pelo método de códigos de Morton de alguma das $3^d - 1$ direções de adjacência $\Delta k = k(\Delta v)$, fazendo uma intercalação das coordenadas da direção de adjacência, as quais assumem apenas os valores $-1 = 11\ldots11_2$, $0 = 00\ldots00_2$ e $1 = 00\ldots01_2$.

Em 2D temos 8 direções adjacentes. As chaves de tamanho 64 bits dessas direções no formato hexadecimal na tabelas abaixo:

2D	
(-1,-1)=0xFFFFFFFFFFFFFFFF	(0, 1)=0x0000000000000001
(-1, 0)=0xAAAAAAAAAAAAAAAA	(1,-1)=0x5555555555555557
(-1, 1)=0xAAAAAAAAAAAAAAAB	(1, 0)=0x0000000000000002
(0,-1)=0x5555555555555555	(1, 1)=0x0000000000000003

Em 3D temos 26 direções adjacentes. As chaves de tamanho 64 bits dessas direções no formato hexadecimal na tabela abaixo:

3D	
(-1,-1,-1)=0xFFFFFFFFFFFFFFFF	(0, 0, 1)=0x0000000000000001
(-1,-1, 0)=0x6DB6DB6DB6DB6DB6	(0, 1,-1)=0x924924924924924B
(-1,-1, 1)=0x6DB6DB6DB6DB6DB7	(0, 1, 0)=0x0000000000000002
(-1, 0,-1)=0xDB6DB6DB6DB6DB6D	(0, 1, 1)=0x0000000000000003
(-1, 0, 0)=0x4924924924924924	(1,-1,-1)=0xB6DB6DB6DB6DB6DF
(-1, 0, 1)=0x4924924924924925	(1,-1, 0)=0x2492492492492496
(-1, 1,-1)=0xDB6DB6DB6DB6DB6F	(1,-1, 1)=0x2492492492492497
(-1, 1, 0)=0x4924924924924926	(1, 0,-1)=0x924924924924924D
(-1, 1, 1)=0x4924924924924927	(1, 0, 0)=0x0000000000000004
(0,-1,-1)=0xB6DB6DB6DB6DB6DB	(1, 0, 1)=0x0000000000000005
(0,-1, 0)=0x2492492492492492	(1, 1,-1)=0x924924924924924F
(0,-1, 1)=0x2492492492492493	(1, 1, 0)=0x0000000000000006
(0, 0,-1)=0x9249249249249249	(1, 1, 1)=0x0000000000000007