3
Compressão de Seqüência de Bits

Informalmente, um algoritmo tem que capaz de construir uma prova de que a saída gerada por ele é correta. Assumimos a partir de agora o modelo computacional baseado em comparações. Por exemplo, um algoritmo executa as operações de comparação $<$, $=$, $>$ e cada uma retorna apenas um bit de informação. Ao final da execução, o algoritmo terá produzido, além da saída correta, uma cadeia binária chamada *prova* (Figura 3.1).

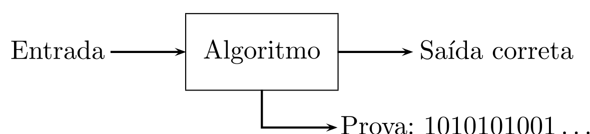

Figura 3.1: Visão geral de um algoritmo no modelo de comparações.

O tamanho da prova é o tempo de execução medido em comparações do algoritmo. Desta forma, o limite inferior do tempo de execução de um algoritmo é o menor tamanho possível da prova codificada. Note que o menor tamanho de uma prova é, na média, diretamente proporcional ao tamanho da entrada e à entropia da prova.

O algoritmo de busca sem limite proposto por Bentley e Yao (Ben76) determina a posição de uma chave em um vetor linear ordenado e sem limites. Ao executarem o algoritmo e verificarem os padrões das comparações, eles perceberam que os resultados das comparações apresentam uma correspondência com a codificação binária de números naturais. Neste caso, a saída do algoritmo é a posição x da chave no vetor e a prova é exatamente a palavra no código Gama (Seção 2.2.3) associado ao número x. Assim, dizemos que o algoritmo de busca sem limite é *isomorfo* ao código Gama. Note que a busca sequencial e binária são isomorfos aos códigos unário e binário, respectivamente.

Neste capítulo, lidamos com a conexão entre os codificadores de fonte e os algoritmos de intercalação (*merge*). Dados duas listas ordenadas A e B, o problema de intercalação consiste em gerar uma lista ordenada C que contem os elementos de $A \cup B$. Esse problema surge naturalmente em diversos domínios de aplicações, tais como gerenciamento de banco de dados e recuperação de informação, sendo um problema clássico de computação (Knu73).

PUC-Rio - Certificação Digital Nº 0711293/CA

A principal contribuição aqui é a demonstração que qualquer algoritmo de intercalação baseado em comparação pode ser naturalmente mapeado em um codificador de fonte através de uma função de conversão que apresentamos aqui. Este resultado é interessante porque torna possível a interpretação da vasta literatura em algoritmos de intercalação (por exemplo, algoritmos *online* e *offline*, estratégias paralelas e distribuídas, implementações de hardware, análise de algoritmo, etc.) no contexto de métodos de codificação.

Na verdade, investigando este processo de conversão, percebemos que alguns dos algoritmos de intercalação mais populares estão intimamente relacionados a alguns codificadores bem conhecidos. Por exemplo, o algoritmo de intercalação binária proposto em (Hwa72) corresponde a um codificador baseado no tamanho das carreiras que usa a código de Rice (Ric79) para representar a repetição de bits. Outro exemplo é o algoritmo recursivo de intercalação proposto em (Dud81). Ele está intimamente relacionado com o codificador de intercalação binária (*Binary Interpolative Coder*) proposto por Moffat et. al. (Mof00) para compressão de índices invertidos.

Além disso, propomos um codificador baseado nos comprimentos das carreiras obtido através da aplicação de nossa função de conversão sobre o algoritmo de intercalação probabilístico apresentado em (Veg93). Descobrimos que é possível extrair, do codificador convertido, um novo variante do código de Rice, intitulado *código de Rice aleatório*. Este novo código usa uma fonte aleatória visando reduzir sua redundância média.

O restante do capítulo está organizado da seguinte forma. Na Seção 3.1, explicamos como mapear algoritmos de intercalação em codificadores binários. Na Seção 3.2, abordamos a conexão de dois algoritmos de intercalação bem conhecidos com dois métodos de codificação conhecidos. Além disso, apresentamos um codificador baseado em comprimento de carreira que é obtido através da aplicação de nossa função de conversão sobre um algoritmo de intercalação probabilístico proposto em (Veg93).

3.1 Codificadores de Fonte Baseados em Algoritmos de Intercalação

3.1.1 Algoritmos de Intercalação Baseados em Comparação

Dado dois subconjuntos e linearmente ordenados $A = \{a_1 < ... < a_m\}$ e $B = \{b_1 < ... < b_n\}$, com $m \leq n$, de um conjunto linearmente ordenado C, o problema de intercalação consiste em determinar a ordenação linear de sua união (ou seja, para mesclar A e B). Sem perda de generalidade, supomos

PUC-Rio - Certificação Digital Nº 0711293/CA

que os conjuntos A e B são disjuntos e que para todo $a_i \in A$ e $b_j \in B$ temos que $a_i \neq b_j$. Seja $compare(u, v)$ uma operação que toma como entrada dois elementos u e v de um conjunto linearmente ordenado e produz um '$<$' se $u < v$ ou '$>$' se $u > v$. Um algoritmo de intercalação baseado em comparação une A e B através da realização de uma seqüência de operações *compare*, onde um dos argumentos de *compare* pertence a A e outro pertence a B.

Para um algoritmo de intercalação baseado em comparações r, defina $M^r(m, n)$ como o número de comparações executadas por r para mesclar duas listas ordenadas de tamanhos m e n no pior caso. Além disso, defina $M(m, n) = min_{r \in \mathcal{R}} M^r(m, n)$, onde r é otimizado sob a classe $\mathcal{R}$ de todos os possíveis algoritmos de intercalação baseados em comparação. Para todos os $m, n \geq 1$, as seguintes desigualdades são verdadeiras (Knu73):

$$I(m, n) \leq M(m, n) \leq m + n - 1,$$

onde $I(m, n) = \left\lceil \log_2 \binom{m+n}{m} \right\rceil$ é uma cota inferior que pode ser obtido utilizando argumentos baseados em teoria da informação.

A determinação do valor exato de $M(m, n)$ tem sido estudada em vários trabalhos (Knu73)(Sto80)(Chr78a)(Man79). Além disso, há diversos algoritmos (Hwa72) (Dud81) (Veg93) (Chr78a) (Man79) que executam em $O(I(m, n)) = O(M(m, n)) = O\left(m \log_2 \left(\frac{n}{m} + 1\right)\right)$ comparações no pior caso. Todos eles são assintoticamente ótimos no modelo baseado em comparações.

3.1.2
Mapeando Algoritmos de Intercalação em Codificadores de Fonte Binária

Dado uma cadeia binária x, seja $x(i)$ o i-ésimo símbolo de x. Além disso, seja $A^x = \{i | x(i) = 0\}$ e $B^x = \{i | x(i) = 1\}$, onde a_i^x e b_i^x são o i-ésimo elemento do A^x e B^x, respectivamente. Por exemplo, se $x = (11011110010001110111)$ então $A^x = \{3, 8, 9, 11, 12, 13, 17\}$ e $B^x = \{1, 2, 4, 5, 6, 7, 10, 14, 15, 16, 18, 19, 20\}$. Este exemplo será usado novamente na Seção 3.2.

Seja $\mathcal{R}$ a classe de todos os algoritmos de intercalação baseados em comparação e seja $\mathcal{S}$ a classe de todos os codificadores binários (Seção 2.2.5). Apresentamos uma função $\varphi : \mathcal{R} \mapsto \mathcal{S}$ que converte um algoritmo arbitrário de intercalação $r \in \mathcal{R}$ em um codificador binário $s \in \mathcal{S}$. Fixe um algoritmo de intercalação $r \in \mathcal{R}$. O codificador binário $\varphi(r)$ mapeia uma cadeia binária x em uma cadeia binária y, onde $y(i) = 1$ (resp $y(i) = 0$) se e somente se a i-ésima operação *compare*, executado por r para mesclar A^x e B^x, gerar '$>$' (resp '$<$'). Note que y é a prova do algoritmo de intercalação r gerada ao

PUC-Rio - Certificação Digital Nº 0711293/CA

processar a instância A^x e B^x. A quantidade de bits 0 e 1 são adicionadas na codificação da instância.

Figura 3.2 ilustra a árvore de intercalação correspondente ao algoritmo de intercalação linear (*tape or linear merging*) (Sto80) para mesclar os conjuntos $A = \{a_1, a_2\}$ e $B = \{b_1, b_2\}$. As elipses são usadas para representar as comparações e retângulos para representar o conjunto ordenado final. Para codificar a cadeia binária $x = (1001)$, como exemplo, podemos construir primeiro os conjuntos $A^x = \{2, 3\}$ e $B^x = \{1, 4\}$. Em seguida, aplicamos o processo de intercalação sobre A^x e B^x. A seqüência $>, <, <$ é gerada pelas operações de *compare* de forma que o código $y = (100)$ seja produzido.

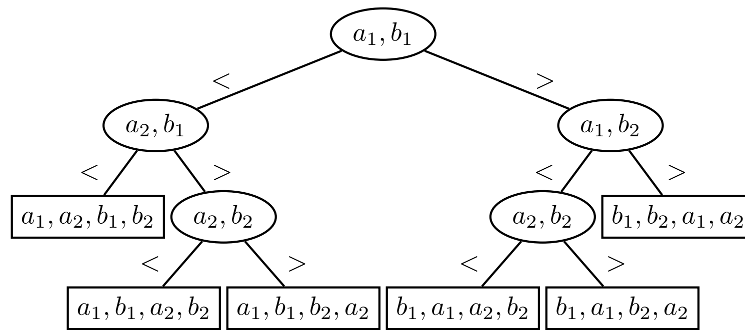

Figura 3.2: Exemplo de árvore de intercalação.

Note que para um algoritmo de intercalação $compare(u, v)$ e $compare(v, u)$ tem o mesmo efeito. No entanto, para fins de codificação devemos fixar que lista vem à esquerda e à direita nos argumentos de *compare*. Como exemplo, se substituirmos $compare(a_1, b_2)$ por $compare(b_2, a_1)$ na Figura 3.2, então a codificação de $x = (1001)$ torna-se (110) em vez de (100).

Dado uma cadeia de caracteres codificada y e um algoritmo de intercalação r, podemos recuperar a cadeia de caracteres original x. Para isso, vamos supor que os números m e n de bits com valores respectivamente de 0 e 1 em x são conhecidos. Podemos construir dois conjuntos linearmente ordenados $A = \{a_1 < \ldots < a_m\}$ e $B = \{b_1 < \ldots < b_n\}$. Em seguida, aplicamos o algoritmo r sobre A e B. Embora não saibamos os valores dos elementos em A e B, podemos mesclar esses conjuntos, porque o resultado da i-ésima comparação executada por r é armazenada em $y(i)$. A saída produzida pelo algoritmo de intercalação r é um conjunto linearmente ordenado $C = \{c_1 < \ldots < c_{m+n}\}$ contendo os elementos de $A \cup B$. Podemos reconstruir x definindo $x(i) = 0$ se $c_i \in A$ e $x(i) = 1$, caso contrário.

Ilustramos o processo de decodificação com o seguinte exemplo. Suponha que a cadeia binária $x = (1001)$ foi codificada em $y = (100)$ usando o algoritmo de intercalação linear. Sabemos que existem $m = 2$ 0-bits e $n = 2$ 1-bits em x.

PUC-Rio - Certificação Digital Nº 0711293/CA

Portanto, temos que $A = \{a_1, a_2\}$, $B = \{b_1, b_2\}$ e $C = \{c_1, c_2, c_3, c_4\}$. Simulando os passos do algoritmo de intercalação linear, temos que:

1) O resultado de $compare(a_1, b_1)$ é $'>'$ pois $y(1) = 1$. Logo, segue-se que $a_1 > b_1$ e $c_1 = b_1$. Como $c_1 \in B$, resulta que $x(1) = 1$;

2) O resultado de $compare(a_1, b_2)$ é $'<'$ pois $y(2) = 0$. Logo, segue-se que $a_1 < b_2$ e $c_2 = a_1$. Como $c_2 \in A$, resulta que $x(2) = 0$;

3) O resultado de $compare(a_2, b_2)$ é $'<'$ pois $y(3) = 0$. Logo, segue-se que $a_2 < b_2$ e $c_3 = a_2$. Como $c_3 \in A$, resulta que $x(3) = 0$;

4) Resta apenas que $c_4 = b_2$. Como $c_4 \in B$, resulta que $x(4) = 1$.

Isto demonstra o principal resultado:

Teorema 3.1 *Qualquer algoritmo de intercalação baseado em comparação pode ser mapeado em um codificador de fonte binária.*

Um subconjunto interessante de $\mathcal{R}$ é o subconjunto $\mathcal{R}'$ contendo apenas os algoritmos de intercalação baseados em comparação assintoticamente ótimos. A principal característica do $\mathcal{R}'$ é que o número de comparações realizadas por qualquer algoritmo de intercalação $r \in \mathcal{R}'$ é $O(I(m, n))$ no pior caso. Como

$$I(m, n) \in O\left((m + n)H\left(\frac{m}{m + n}, \frac{n}{m + n}\right)\right),$$

segue-se que φ converte qualquer algoritmo de intercalação $r \in \mathcal{R}'$ em um codificador binário de entropia. Finalizamos esta seção com o seguinte resultado:

Corolário 3.2 *Qualquer algoritmo de intercalação baseado em comparação e assintoticamente ótimo pode ser mapeado em um codificador de entropia binário.*

3.2 Aplicações

Nesta seção, mostramos como alguns dos algoritmos de intercalação mais populares se relacionam com alguns codificadores de entropia conhecidos. Na descrição dos algoritmos de intercalação, omitimos as etapas de atribuição e alguns detalhes de implementação, porque apenas os passos de comparações são relevantes para o processo de conversão.

PUC-Rio - Certificação Digital Nº 0711293/CA

3.2.1 Algoritmo de Intercalação Binário

O algoritmo de intercalação binário (Hwa72), abreviado por BM, é um conhecido algoritmo que foi originalmente concebido para mesclar dois arquivos armazenados em fitas. Em cada etapa, ele verifica se o primeiro elemento da menor lista é maior do que o 2^t-ésimo elemento da maior lista, onde t é um número inteiro que depende do tamanho de ambas as listas. Em caso positivo, os primeiros 2^t elementos da maior lista precedem os primeiros elementos da menor lista. Em caso negativo, ele usa uma busca binária para determinar quais elementos da maior lista são menores do que o primeiro elemento da menor lista. O seu pseudocódigo é apresentado no Algoritmo 1.

Algoritmo 1 Algoritmo de intercalação binário

```
 1: procedure BM(A, B)
 2:     U ← menor conjunto entre A e B
 3:     V ← maior conjunto entre A e B
 4:     if U ≠ ∅ e V ≠ ∅ then
 5:         t = ⌊log_2 (|V|/|U|)⌋
 6:         if u_1 < v_{2^t} then                          ▷ compare(u_1, v_{2^t})
 7:             Faça uma busca binária para encontrar o inteiro q tal que v_q < u_1 < v_{q+1}
 8:             BM(U[2, ..., |U|], V[q + 1, ..., |V|])
 9:         else
10:             BM(U[1, ..., |U|], V[2^t + 1, ..., |V|])
11:         end if
12:     end if
13: end procedure
```

Aqui, convertemos uma ligeira variação do Algoritmo 1 em um codificador binário. Nesta variação, em cada etapa, uma pesquisa híbrida (uma pesquisa seqüencial combinada com uma pesquisa binária) é executada para encontrar o número de elementos na maior lista que são menores do que o primeiro elemento da menor lista, ou seja, a posição de u_1 em V. Este novo algoritmo é obtido do BM, substituindo as linhas 6-11 por:

i) Faça uma busca sequencial para encontrar o menor inteiro i no conjunto $\{1, 2, 3, \ldots\}$ tal que $u_1 < v_{i2^t}$;

ii) Faça uma busca binária no intervalo $[i2^t - 2^t, i2^t - 1]$ para encontrar um inteiro q tal que $v_q < u_1 < v_{q+1}$;

iii) $BM(U[2, \ldots, |U|], V[q + 1, \ldots, |V|])$.

Conversão. Convertendo esse algoritmo modificado, obtemos um codificador baseado no tamanho das carreiras que usa o código de Rice para representar

PUC-Rio - Certificação Digital Nº 0711293/CA

Algoritmo 2 Codificador resultante do algoritmo de intercalação binário

```
1: procedure ENCODE(x)
2:     m ← número de bits 0 em x
3:     n ← número de bits 1 em x
4:     if m > 0 e n > 0 then
5:         t = ⌊log2 (max{m,n} / min{m,n})⌋
6:         if m > n then
7:             j* ← min{j|x(j) = 1}
8:         else
9:             j* ← min{j|x(j) = 0}
10:        end if
11:        Adicione o código de Rice para j* − 1, com parâmetro t, na saída y
12:        Encode(x[j* + 1, ..., |x|])
13:    end if
14: end procedure
```

repetições do mesmo bit. O pseudocódigo para codificar uma cadeia binária x em uma cadeia binária y é dado abaixo:

Para ilustrar o esquema de codificação, usamos o exemplo da Seção 3.1.2. O método de codificação calcula, em cada recursão, um valor para j^* que gera a seqüência de valores $(3, 5, 1, 2, 1, 1, 4)$. Ele codifica esses valores usando código de Rice com a respectiva seqüência de parâmetros $(0, 0, 0, 0, 1, 1, 2)$ produzindo a seqüência de códigos de Rice $(001, 00001, 1, 01, 10, 10, 111)$.

Argumentamos que o codificador acima é obtido através de nossa função de conversão sobre o BM modificado. Vamos supor que u_1 é sempre o argumento à esquerda do operador *compare* na etapa das linhas i-iii. A principal observação é que a seqüência de comparações realizadas pelo algoritmo de intercalação para encontrar o inteiro q na linha ii corresponde, no sentido da nossa função de conversão, ao código de Rice de q com o parâmetro t. Na verdade, o código unário corresponde à busca seqüencial enquanto o código binário corresponde à busca binária.

Assim, falta mostrar que o inteiro q é exatamente $j^* - 1$. Primeiro, note que $m = |A^x|$, $n = |B^x|$ e

$$t = \left\lfloor \log_2 \frac{\max\{m, n\}}{\min\{m, n\}} \right\rfloor = \left\lfloor \log_2 \frac{\max\{|A^x|, |B^x|\}}{\min\{|A^x|, |B^x|\}} \right\rfloor .$$

Se $|A^x| > |B^x|$ então q é tal que $a^x_q < b^x_1 < a^x_{q+1}$. Decorre da definição de A^x e B^x que $j^* = b^x_1 = q + 1$ e como conseqüência, $q = j^* - 1$. Um argumento semelhante é válido quando $|A^x| < |B^x|$.

Discussão. Note que quando $m = n$, o algoritmo de intercalação binário reduz-se ao algoritmo de intercalação linear e, quando $m = 1$, reduz-se à uma busca binária descentralizada. Ele apresenta um bom equilíbrio para os intervalos de m e n (Knu73). Os autores do BM demonstraram que o número

PUC-Rio - Certificação Digital Nº 0711293/CA

de comparações M^{BM} executado pelo algoritmo de intercalação binária tem cota superior dado por:

$$\left\lceil \log_2 \binom{m+n}{m} \right\rceil + m.$$

Pelo Apêndice 1, verificamos que:

$$M^{BM}(m,n) < \left\lceil \log_2 \binom{m+n}{m} \right\rceil + m < (m+n)H\left(\frac{m}{m+n}, \frac{n}{m+n}\right) + m.$$

Como $m \leq n$, decorre que:

$$M^{BM}(m,n) < (m+n)\left[H\left(\frac{m}{m+n}, \frac{n}{m+n}\right) + 0,5\right].$$

3.2.2
Algoritmo de Intercalação Recursiva

O algoritmo de intercalação recursivo (Dud81), abreviado por RM, é também um simples e bem conhecido algoritmo que une dois conjuntos A e B de tamanhos m e n, respectivamente, da seguinte forma: primeiro, ele encontra a posição de a_i, onde $i = \lfloor m/2 \rfloor$, na lista de B, ou seja, o inteiro q para que $b_q < a_i < b_{q+1}$. Em seguida, ele recursivamente une as listas $A[1,\ldots,i-1]$ e $B[1,\ldots,q]$ e, em seguida, as listas $A[i+1,\ldots,m]$ e $B[q+1,\ldots,n]$. O pseudocódigo é apresentado a seguir:

Algoritmo 3 Algoritmo de intercalação recursivo

1: **procedure** RM(A, B)
2: Seja $m = |A|$, $n = |B|$ e $i = \lfloor m/2 \rfloor$;
3: **if** $n = 0$ ou $m = 0$ **then**
4: Saia do procedimento;
5: **end if**
6: **if** $n < m$ **then**
7: Troque m com n e A com B;
8: **end if**
9: Faça uma busca binária para encontrar a posição de a_i em B, ou seja, o inteiro q para qual $b_q < a_i < b_{q+1}$;
10: $RM(A[1,\ldots,i-1], B[1,\ldots,q])$;
11: $RM(A[i+1,\ldots,m], B[q+1,\ldots,n])$.
12: **end procedure**

Aplicando a função de conversão sobre este algoritmo de intercalação obtemos um codificador de entropia binário que está intimamente relacionado com o *Binary Interpolative Coder* (BIC) proposto por Moffat *et al.* (Mof00) para compactação de índices invertidos. Primeiro, descrevemos o BIC e, em seguida, explicamos sua conexão com o RM.

PUC-Rio - Certificação Digital Nº 0711293/CA

Conversão. BIC recebe três parâmetros de entrada (A, lo, hi), onde $A = \{a_1 < \ldots < a_m\}$ é um conjunto ordenado de inteiros e lo e hi são inteiros que satisfazem $lo \leq a_1$ e $a_m \leq hi$, respectivamente. Seja $i = \lfloor (m+1)/2 \rfloor$ e $m > 1$. Uma vez que todos os inteiros de A são distintos, resulta que a_i pertence ao intervalo $[lo + i - 1, hi + i - m]$ do tamanho $w = hi - lo - m + 2$. A primeira etapa do BIC consiste de codificação a_i. Para isso, ele usa o código binário centralizado mínimo (Mof00) que gera $2^{\lceil \log_2 w \rceil} - w$ palavras-código de tamanho $\lfloor \log_2 w \rfloor$ e $2w - 2^{\lceil \log_2 w \rceil}$ palavras-código de tamanho $\lceil \log_2 w \rceil$. Em seguida, BIC é recursivamente chamado para a entrada $(A[1, ..., i-1], lb, a_i - 1)$ e, em seguida, para a entrada $(A[i+1, ..., m], a_i + 1, hi)$. Para $m = 1$, BIC codifica a_1 usando o código binário centralizado mínimo no intervalo $[lo, hi]$.

A conexão entre RM e BIC baseia-se no fato de que a codificação produzida por RM para uma cadeia binária x é similar ao produzido pela BIC para entrada $(A^x, 1, |x|)$. Na verdade, as duas codificações são semelhantes se a etapa (7) for removida de RM e a busca binária realizada por RM na etapa (9) for isomorfa à codificação de a_i na primeira etapa do BIC.

Podemos ilustrar essa conexão através de um exemplo. Usamos novamente a cadeia binária x do exemplo da Seção 3.1.2. O primeiro elemento a ser mesclado é $A^x[4] = 11$. Através da aplicação da busca binária induzida pela árvore da Figura 3.3, as seguintes operações são executadas $compare(11, 10)$, $compare(11, 16)$ e $compare(11, 14)$ que geram código 100. Em seguida, RM é recursivamente chamado com entrada $(A^x[1, \ldots, 3], B^x[1, \ldots, 7])$ e então com entrada $(A^x[5, \ldots, 7], B^x[8, \ldots, 13])$. Observe que, para cada chamada recursiva, a busca binária é induzida por uma diferente árvore binária mínima centralizada em seu intervalo.

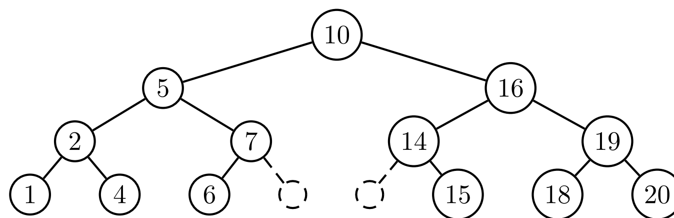

Figura 3.3: Árvore binária mínima centrada.

Por outro lado, BIC recebe $(A^x, 1, 20)$ como parâmetros de entrada. Primeiro, ele codifica o número 11 usando o intervalo $[1 + 3, 20 - 3] = [4, 17]$. Uma vez que há $17 - 4 + 1 = 14$ números possíveis, então ele precisa de 4 bits para codificar qualquer número em $[4..9, 12..17]$ e 3 bits para codificar qualquer número em $[10..11]$. Portanto, o número 11 é codificado como 100. Em seguida, BIC é recursivamente chamado com entrada $(A^x[1, \ldots, 3], 1, 10)$ e então com entrada $(A^x[4, \ldots, 7], 12, 20)$.

PUC-Rio - Certificação Digital Nº 0711293/CA

Discussão. O algoritmo de intercalação recursivo é assintoticamente ótimo e os autores do RM fizeram uma análise simples para mostrar que é executado em:

$$M^{RM}(m,n) = O\left(m \log_2\left(\frac{n}{m}+1\right)\right).$$

Como já mencionamos, os autores do codificador BIC desenvolveram este codificador para o problema da compactação de índices invertidos. De acordo com a pesquisa deles, este codificador é adequado para dados que apresentam distribuição não-uniforme (com *clusters*). É interessante notar que vários algoritmos de intercalação (Dem00)(Car90) têm sido estudados para este tipo de dados e, à luz do nosso resultado, eles podem ser convertidos em codificadores para fontes não-estacionárias, tais como, índices invertidos e imagens de documentos binárias.

3.2.3
Algoritmo de Intercalação Probabilístico

O algoritmo de intercalação probabilístico (Veg93), abreviado por PM, utiliza técnicas de aleatorização para reduzir o número de comparações. Seja $\delta_1 = (\sqrt{5}-1)/2 \approx 0.618$ e $\delta_2 = (\sqrt{2}-1+\sqrt{2}\delta_1)^2 \approx 1.659$. De La Vega *et al.* (Veg93) mostraram que PM é mais eficiente do que o algoritmo de intercalação binária, apresentado na Seção 3.2.1, para $|B|/|A| > 1+\delta_1$ onde A e B são conjuntos linearmente ordenados a serem unidos. Eles ainda recomendam o uso do algoritmo de intercalação linear (Sto80) quando $|B|/|A| \leq 1+\delta_1$.

O algoritmo recebe dois outros parâmetros t e p, além de dois conjuntos linearmente ordenados A e B. O parâmetro t está relacionado com a variável t do BM, mas permanece constante durante o processo de intercalação. O parâmetro p é usado para definir a probabilidade de realizar algumas comparações durante a execução do algoritmo. Uma discussão sobre como definir os valores de p e t para minimizar o número médio de comparações empregado pelo PM é apresentada em (Veg93).

O algoritmo PM pode ser visto como uma variação do BM, onde uma moeda viciada é jogada, em cada etapa, para determinar se b_{2^t} ou $b_{2^{t+1}}$ devem ser comparados com o primeiro elemento da lista A. Nossa apresentação do PM é um pouco diferente da que consta em (Veg93). Ele é apresentado no Algoritmo 4 a seguir.

De forma a manter compatibilidade com o algoritmo original de (Veg93), temos que inserir uma lista ordenada $\{l_1 < \ldots < l_{2^t-1}\}$ no início do conjunto B tal que $l_{2^t-1} < b_1$. Além disso, partimos do princípio que $b_k = \infty$ para todo $k > |B|$.

PUC-Rio - Certificação Digital Nº 0711293/CA

Algoritmo 4 Algoritmo de intercalação probabilístico

```
1: procedure PM(A, B, p, t)
2:     if A = ∅ or B = ∅ then
3:         Saia do procedimento;
4:     end if
5:     Inicialize i = 0 e j = 0;
6:     Jogue uma moeda viciada que gera 'H' com prob. p e 'T' com prob. 1 − p;
7:     if 'T' then                                                    ▷ compare(a_1, b_{2^t})
8:         if a_1 < b_{2^t} then
9:             Faça uma busca binária para a_1 na lista {b_1 < ... < b_{2^t−1}} e faça i = 1;
10:        else
11:            Faça j = 2^t;
12:        end if
13:    else if 'H' then                                               ▷ compare(a_1, b_{2^{t+1}})
14:        if a_1 < b_{2^{t+1}} then
15:            Faça uma busca binária para a_1 na lista {b_1 < ... < b_{2^{t+1}−1}}, faça j = 2^t se
    a_1 > b_{2^t} e, finalmente, faça i = 1;
16:        else
17:            Faça j = 2^{t+1};
18:        end if
19:    end if
20:    PM(A[1 + i, ..., |A|], B[1 + j, ..., |B|], p, t).
21: end procedure
```

Conversão. Convertendo PM, obtemos um codificador baseado no comprimento das carreiras que usa uma nova variante do código de Rice para representar a posição relativa dos 0-bits, supondo-se que há mais 1-bits do que de 0-bits na cadeia binária de entrada.

O pseudocódigo para codificação uma cadeia binária x em uma cadeia binária y é dada abaixo. Usamos $u \circ v$ para denotar a concatenação das cadeias binárias u e v. Mais uma vez, para manter a compatibilidade com o algoritmo de intercalação, inserimos no início de x a cadeia binária $(1, \ldots, 1)$ de tamanho $2^t - 1$. Os valores de t e p são calculados da mesma forma que o algoritmo de intercalação.

Foi possível extrair do PM uma nova variante do código de Rice, intitulado *código de Rice aleatório*, gerado pelo procedimento acima. Cada palavra é composto por uma parte unária (passos 13 e 17) e uma parte binária (etapa 28) como no código original de Rice (Ric79). No entanto, ambas as partes dependem de uma probabilidade p, além do parâmetro habitual t. O parâmetro p é usado para jogar uma moeda viciada para cada bit e, em seguida, decidir por quanto o valor de j é decrementado e o comprimento da parte binária é incrementado.

Usamos o exemplo da Seção 3.1.2 para ilustrar o método de codificação. Partimos do princípio que $p = 0,618$ e $t = 0$. Em cada recursão, o método de codificação calcula um valor para j^* que gera a seqüência de valores $(3, 5, 1, 2, 1, 1, 4)$. Ele codifica esses valores usando código de Rice aleatório de

PUC-Rio - Certificação Digital Nº 0711293/CA

Algoritmo 5 Codificador resultante do algoritmo de intercalação probabilístico

```
1: function ProbEncode(x, t, p)
2:     if |x| = 0 then
3:         return cadeia binária vazia;
4:     end if
5:     j* ← min{j|x(j) = 0};
6:     Adicione a saída de RandomizedRiceEncode(j* − 1, t, p) ao final da saída de y;
7:     Seja x' a cadeia binária (1, ..., 1) de tamanho j* − 1 mod 2^t;
8:     Adicione a saída de ProbEncode(x' ∘ x[j* + 1, ..., |x|], t, p) ao final da saída de y;
9:     return y.
10: end function

11: function RandomizedRiceEncode(j, t, p)
12:     Inicialize a palavra-código ω como vazia;
13:     while j ≥ 2^(t+1) do                                           ▷ {j ≥ 2^(t+1)}
14:         Adicione um bit 1 ao final de ω;
15:         Faça z = 0 com prob. 1 − p e z = 1 com prob. p; decremente 2^(t+z) de j;
16:     end while
17:     Faça z = 0 com probabilidade 1 − p e z = 1 com probabilidade p;  ▷ {j < 2^(t+1)}
18:     if z = 0 then
19:         if j ≥ 2^t then                                            ▷ Compare j com 2^t
20:             Adicione um bit 1 ao final de ω, decremente 2^t de j e volte à linha 17;
21:         else
22:             Adicione um bit 0 ao final de ω;
23:         end if
24:         if z = 1 then
25:             Adicione um bit 0 ao final de ω;
26:         end if
27:     end if
28:     Adicione o código binário de j usando t + z bits ao final de ω;  ▷ {j < 2^(t+z)}
29:     return ω.
30: end function
```

acordo com a cadeia binária aleatória (0011110111101) produzindo a seqüência de palavras $(1100, 1100, 0, 01, 00, 00, 1100)$. Se empregarmos uma seqüência aleatória binária diferente, digamos (10111010111) então obtemos a seqüência de palavras $(10, 1100, 0, 01, 0, 00, 101)$.

Discussão. É interessante notar que o codificador usa uma fonte binária aleatória para reduzir a redundância esperada da cadeia de saída. Por outro lado, o decodificador tem que receber uma semente, que pode ser um número inteiro, para gerar a mesma seqüência binária aleatória empregada pelo codificador. Além disso, o decodificador tem também que receber os valores de m e n, se ele implementa um modelo estático.

Analisamos o número esperado de bits geradas pelo codificador usando o número esperado de comparações $E^{PM}(m, n)$ feita pela PM. De La Vega *et al.* (Veg93) mostrou que:

$$E^{PM}(m, n) < I(m, n) + 0,471m,$$

PUC-Rio - Certificação Digital Nº 0711293/CA

para valores de m e n suficientemente grandes. Observamos ainda que:

$$E^{PM}(m,n) < (m+n)\left[H\left(\frac{m}{m+n},\frac{n}{m+n}\right)+0,2355\right].$$

O número de bits gerado por $ProbEncode$ pode ser reduzido se o passo de compatibilidade inicial e o passo 7 forem removidos. Igualmente, o número de comparações de PM pode ser reduzido se as etapas correspondentes na PM forem removidas.

PUC-Rio - Certificação Digital Nº 0711293/CA