Projeto de Algoritmos II
Compressão de Dados

Nelson Cruz Sampaio Neto
nelsonneto@ufpa.br

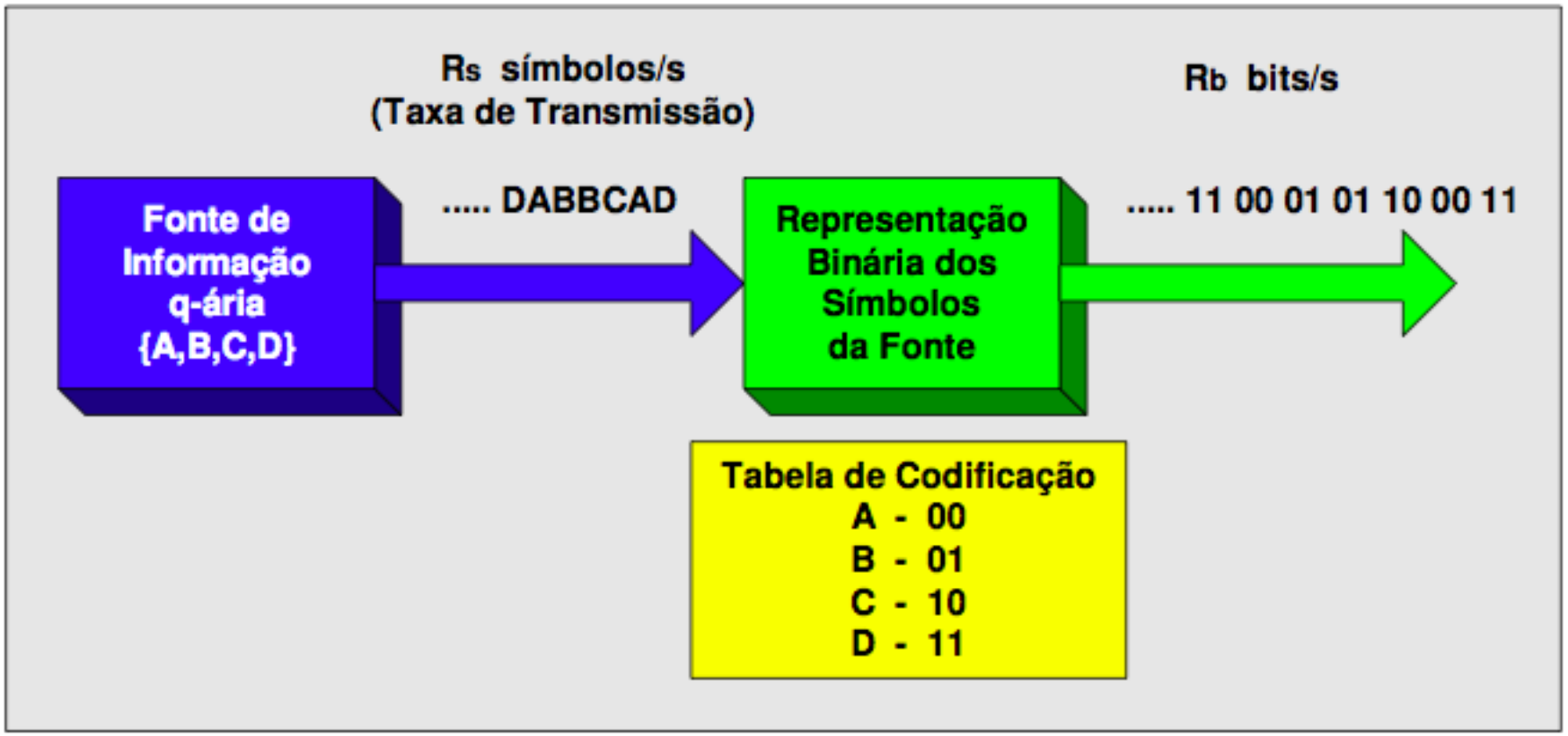
Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Faculdade de Computação

19 de agosto de 2019

Compressão de dados

- A compressão de dados consiste em representar o texto (i.e. sequência de símbolos) original usando menos espaço.
- Para isso, basta substituir os símbolos do texto por outros que possam ser representados usando um número menor de *bits*.
- O arquivo comprimido ocupa menos espaço (armazenamento); leva menos tempo para ser lido do disco, ou transmitido por um canal de comunicação; e para ser pesquisado.

Compressão de dados



Compressão de dados

- Exemplo: Tem-se um arquivo com 100.000 caracteres e deseja-se armazená-lo compactado.

Tabela: 1

	A	**B**	**C**	**D**	**E**	**F**
Frequência (x1000)	45	13	12	16	9	5
Código de comprimento fixo	000	001	010	011	100	101
Código de comprimento variável	0	101	100	111	1101	1100

Compressão de dados

- A Tabela 1 traz um arquivo de dados com 100.000 caracteres que contém no seu alfabeto os caracteres [A..F].
- Se cada caractere for representado com um código de 3 bits, será necessário 300.000 *bits* para codificar o arquivo.
- Em geral, os caracteres do alfabeto não são equiprováveis, ou seja, possuem diferentes probabilidades de ocorrência.
- Logo, é razoável pensar em códigos de comprimento variável, atribuindo códigos mais curtos a caracteres mais frequentes.
- Por exemplo, usando o código de comprimento variável da Tabela 1, é possível codificar o arquivo com 224.000 *bits*.

Caractere	Prob.(%)	Cód.I	Cód.II	Cód.III	Cód.IV
A	50	00	0	0	0
B	25	01	1	10	01
C	15	10	00	110	011
D	10	11	11	111	0111

- O Código II parece ser o mais eficiente. Porém, ao contrário dos demais, ele não é univocamente descodificável.
- No Código IV, o bit 0 funciona como separador. Assim, cada palavra de código é descodificada com atraso de 1 bit.
- Já os Códigos I e III são **códigos de prefixo**, ou seja, são univocamente descodificáveis e possuem descodificação instantânea.

Caractere	Prob.(%)	Cód.I	Cód.II	Cód.III	Cód.IV
A	50	00	0	0	0
B	25	01	1	10	01
C	15	10	00	110	011
D	10	11	11	111	0111
$\overline{L}$	-	2	1,25	1,75	1,875

- O Código III apresentou um comprimento médio $\overline{L}$ menor que o do Código I, o que o torna mais eficiente.

$$\overline{L} = \sum_{c=1}^{n} l_c \; p_c \; \textit{bits}/\text{símbolo, onde}$$

l_c é o tamanho da palavra binária que representa o caractere c e p_c é a probabilidade de ocorrência do caractere c.

Exemplo – Considere uma fonte discreta que emite 4 possíveis símbolos a uma taxa de 200 símbolos/s

Símbolo da Fonte	Probabilidade de ocorrência	Representação Binária Usual	Representação Binária Alternativa
A	50%	00	0
B	25%	01	10
C	15%	10	110
D	10%	11	111

$\overline{L} = 2$ bits/símbo lo

$R_b = 400$ bits/s

$\overline{L} = 1{,}75$ bits/símbolo

$R_b = 350$ bits/s

Código mais eficiente

Código de Huffman

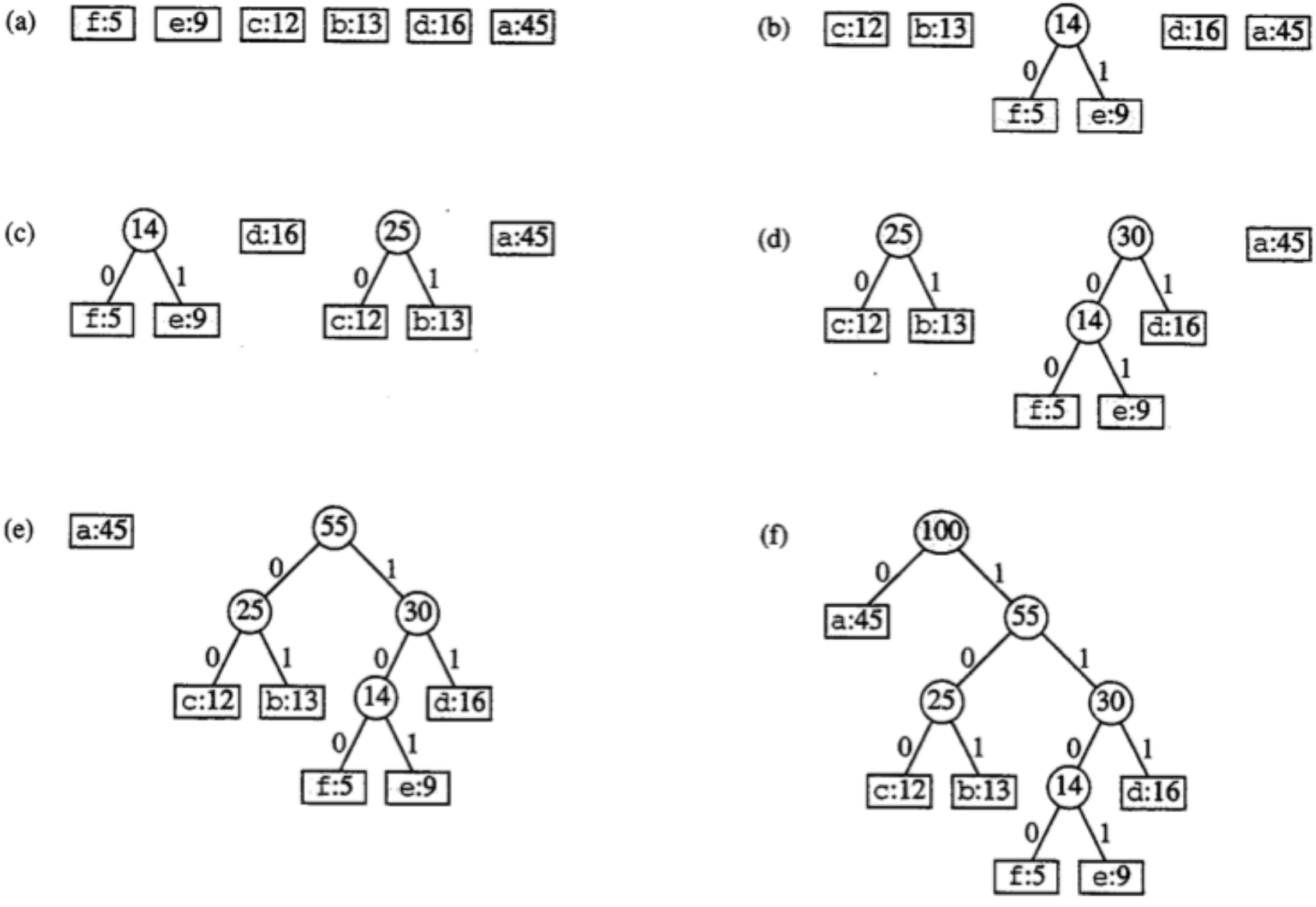
- **Código de Huffman**: Algoritmo eficiente e amplamente utilizado para compressão de dados.
- Sua ideia principal consiste em atribuir códigos mais curtos a símbolos com frequências mais altas.
- Associa códigos binários únicos e de tamanho variável a cada símbolo diferente do alfabeto.
- É gerada uma **árvore binária de prefixo ótima** ao final, ou seja, de comprimento médio mínimo: Árvore de Huffman.

Código de Huffman

O algoritmo de Huffman consiste dos seguinte passos, em um alfabeto de c caracteres:

1. Manter uma floresta de árvores com peso igual à soma das frequências de suas folhas.
2. Selecionar as árvores T_i e T_k de menores pesos e formar uma nova árvore com T_i e T_k.
3. Repetir o passo anterior, até obter a árvore binária final.

Código de Huffman



Código de Huffman

- Calculando quantos são os bits necessários para armazenar o arquivo com o código prefixo ótimo obtido pelo algoritmo de Huffman:

  $(45 \times 1 + 13 \times 3 + 12 \times 3 + 16 \times 3 + 9 \times 4 + 5 \times 4) \times 1.000$
  $= 224.000$ *bits*,

  o que representa uma economia de aproximadamente 25%, quando comparado com a codificação de comprimento fixo apresentada na Tabela 1.

Código de Huffman

- No algoritmo abaixo, assume-se que C é um conjunto de n caracteres e que cada caractere $c \in C$ é um objeto com um atributo $c.freq$ que dá a sua frequência.

```
HUFFMAN (C)
1. n = |C|
2. Q = C
3. para i = 1 até n - 1
4.     Aloca um novo nó z
5.     z.esquerda = x = Extract-min(Q)
6.     z.direita  = y = Extract-min(Q)
7.     z.freq = x.freq + y.freq
8.     Insert(Q,z)
9. retorna Extract-min(Q)  // raiz da árvore
```

- A complexidade no tempo é $O(n \log n)$ considerando a fila de prioridade Q implementada com *heap* binário.

Compressão de Huffman usando palavras

- A eficiência da codificação de Huffman pode ser melhorada se considerarmos extensões de fonte de ordem superior.

- Por exemplo, considere, a princípio, a codificação abaixo.

Caractere	**Código**	**Probabilidade**	**Tamanho**
M1	0	0,70	1
M2	10	0,15	2
M3	11	0,15	2

- $\overline{L} = 1,3$ *bits*/símbolo. Considerando a representação usual com 2 *bits*, economia de 35%.

Compressão de Huffman usando palavras

- Agora, a extensão de 2ª ordem do alfabeto original:

Caractere	Código	Probabilidade	Tamanho
M1M1	1	0,49	1
M1M2	010	0,105	3
M2M1	001	0,105	3
M1M3	000	0,105	3
M3M1	0111	0,105	4
M2M2	011011	0,0225	6
M2M3	011010	0,0225	6
M3M2	011001	0,0225	6
M3M3	011000	0,0225	6

- $\overline{L} = 2,395$ *bits*/símbolo. Considerando a representação usual com 4 *bits*, economia de aproximadamente 40%.

Compressão de Huffman usando palavras

- O código da fonte original proporcionou uma economia de 35%, já o código do alfabeto estendido em 2ª ordem trouxe uma economia de 40%.
- Uma forma melhor de aliar os algoritmos de compressão às necessidades dos sistemas de recuperação de informação é considerar **palavras** como símbolos, e não caracteres.
- O método de Huffman baseado em palavras permite acesso randômico a palavras dentro do texto comprimido, que é um aspecto crítico em sistemas de recuperação de informação.
- Nesse caso, a tabela de símbolos do codificador é exatamente o vocabulário do texto. Isso permite uma integração natural entre o método de compressão e o arquivo invertido.

Compressão de Huffman usando palavras

- Codificação de Huffman baseada em palavra:

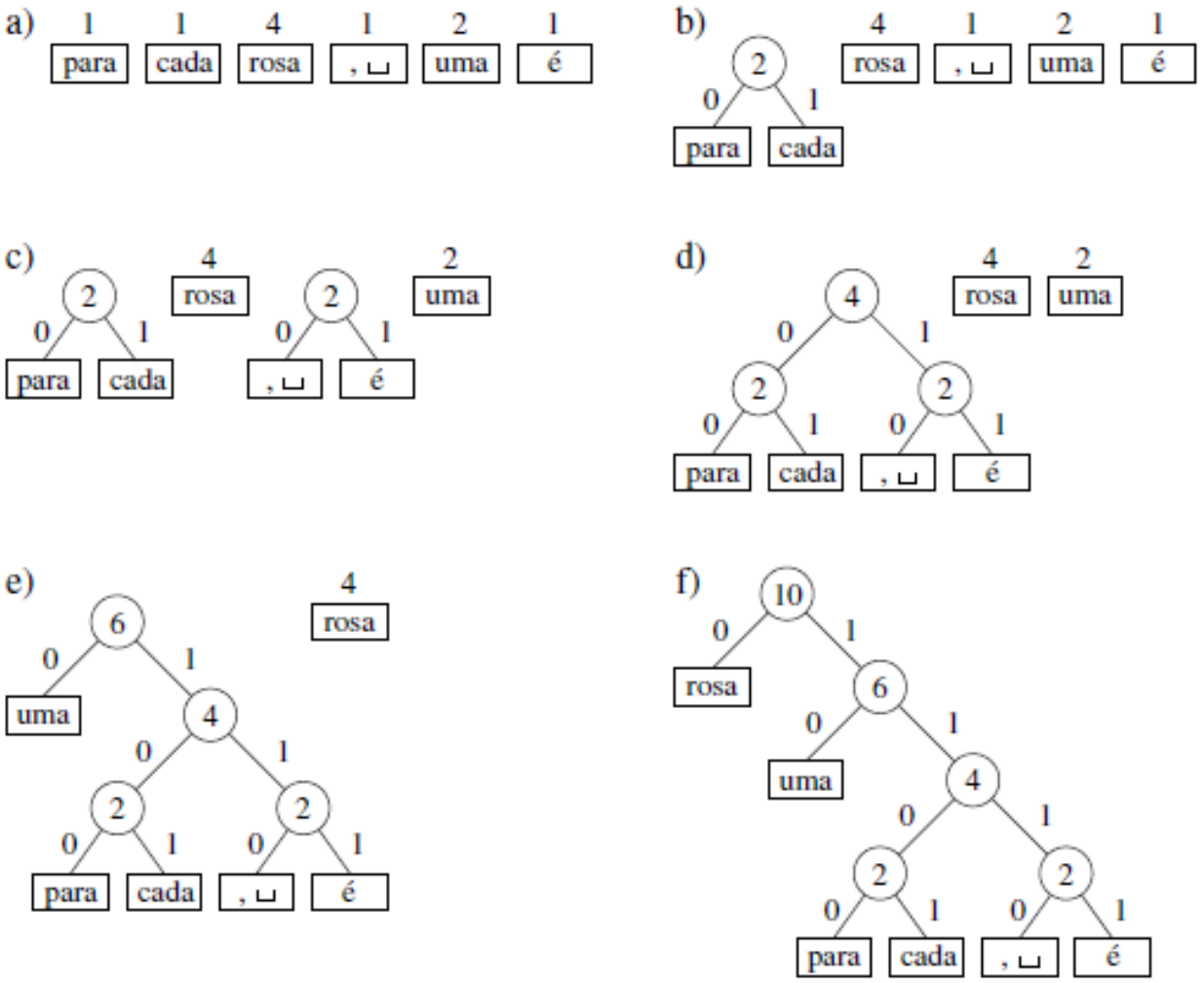
  O método considera cada palavra diferente do texto em linguagem natural como um símbolo, conta suas frequências e gera um código de Huffman para as palavras. A seguir, comprime o texto substituindo cada palavra pelo seu código.

- Caso uma palavra seja seguida de um espaço, então, somente a palavra é codificada. Caso contrário, a palavra e o separador são codificados separadamente.

- Na decodificação, supõe-se que um espaço simples segue cada palavra, a não ser que o próximo símbolo seja um separador.

Compressão de Huffman usando palavras

Exemplo: "para cada rosa rosa, uma rosa é uma rosa"



Considerações

- O método de Huffman codifica um texto de forma a obter-se uma compactação ótima usando códigos de prefixo.
- De posse da árvore de prefixo correspondente, o processo de codificação ou decodificação pode ser realizado em tempo linear no tamanho da sequência binária codificada.
- Possibilidade de realizar casamento de cadeia diretamente no texto comprimido, que é uma pesquisa bem mais rápida dado que menos *bytes* têm que ser lidos.
- Permite acesso direto a qualquer parte do texto comprimido, iniciando a descompressão a partir da parte acessada. Isto é, não precisa descomprimir o texto desde o início.

Exercício

- Uma fonte de informação emite 1.000 símbolos/segundo. As probabilidades de ocorrência de cada símbolo do alfabeto são mostradas na tabela abaixo.

Símbolo da fonte	**Probabilidade de ocorrência**
A	0,4
B	0,2
C	0,2
D	0,1
E	0,1

a. Construa um código fonte usando o algoritmo de Huffman.

b. Calcule o comprimento médio das palavras do código.

c. Calcule a taxa média de transmissão em *bits*/segundo após a codificação da fonte.

Exercício

- Suponha que a tabela a seguir apresenta a frequência de cada letra de um alfabeto em uma *string*.

  Quantos *bits* seriam necessários para representar essa *string* usando um código de Huffman?

Letra	A	B	C	D	E	F
Frequência	20	10	8	5	4	2

(A) 392.
(B) 147.
(C) 113.
(D) 108.
(E) 49.

Códigos de Lempel-Ziv

- A maior dificuldade em usar Huffman é a necessidade de se conhecer *a priori* ou estimar as probabilidades dos símbolos que compõem a sequência que se pretende codificar.

- Outra desvantagem é que tanto o *encoder* quando o *decoder* precisam conhecer a árvore de codificação, ou seja, a relação entre as palavras de código e os símbolos.

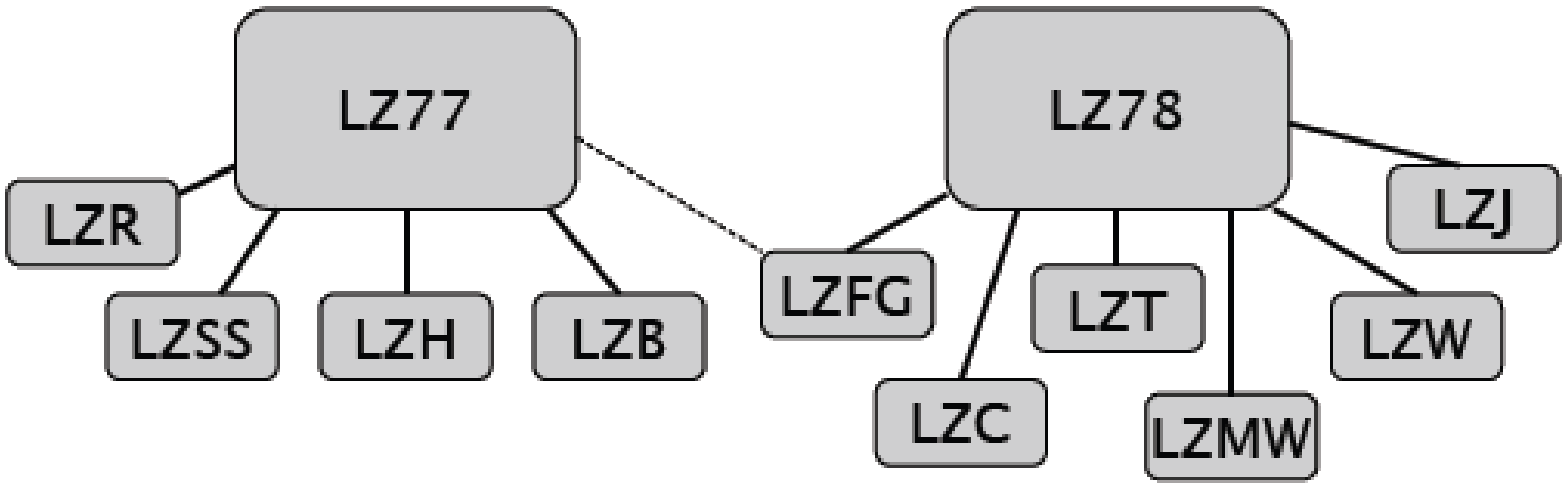
- Já na codificação de Lempel-Ziv, as condições acima **não** são necessárias, o que a torna muito eficiente.

Códigos de Lempel-Ziv

- Os algoritmos de Lempel-Ziv não usam métodos estatísticos.
- Baseiam-se no princípio de **compressão por substituição**.
- A sequência de símbolos de entrada é dividida em segmentos (dicionário) e, depois, os segmentos já vistos são representados por palavras de código de tamanho constante (referências).
- Em um dado momento, a compressão é obtida porque as referências ocupam menos espaço do que a segmentos que elas substituem.
- A codificação de Lempel-Ziv é **universal**, ou seja, ela não depende da distribuição da fonte de dados e o dicionário não precisa ser armazenado ou enviado.

Códigos de Lempel-Ziv

Principais versões dos algoritmos de Lempel-Ziv: **LZ77** e **LZ78**.



Applications:
- zip
- gzip
- Stacker
- ...

Applications:
- GIF
- V.42
- compress
- ...

Algoritmo LZ77

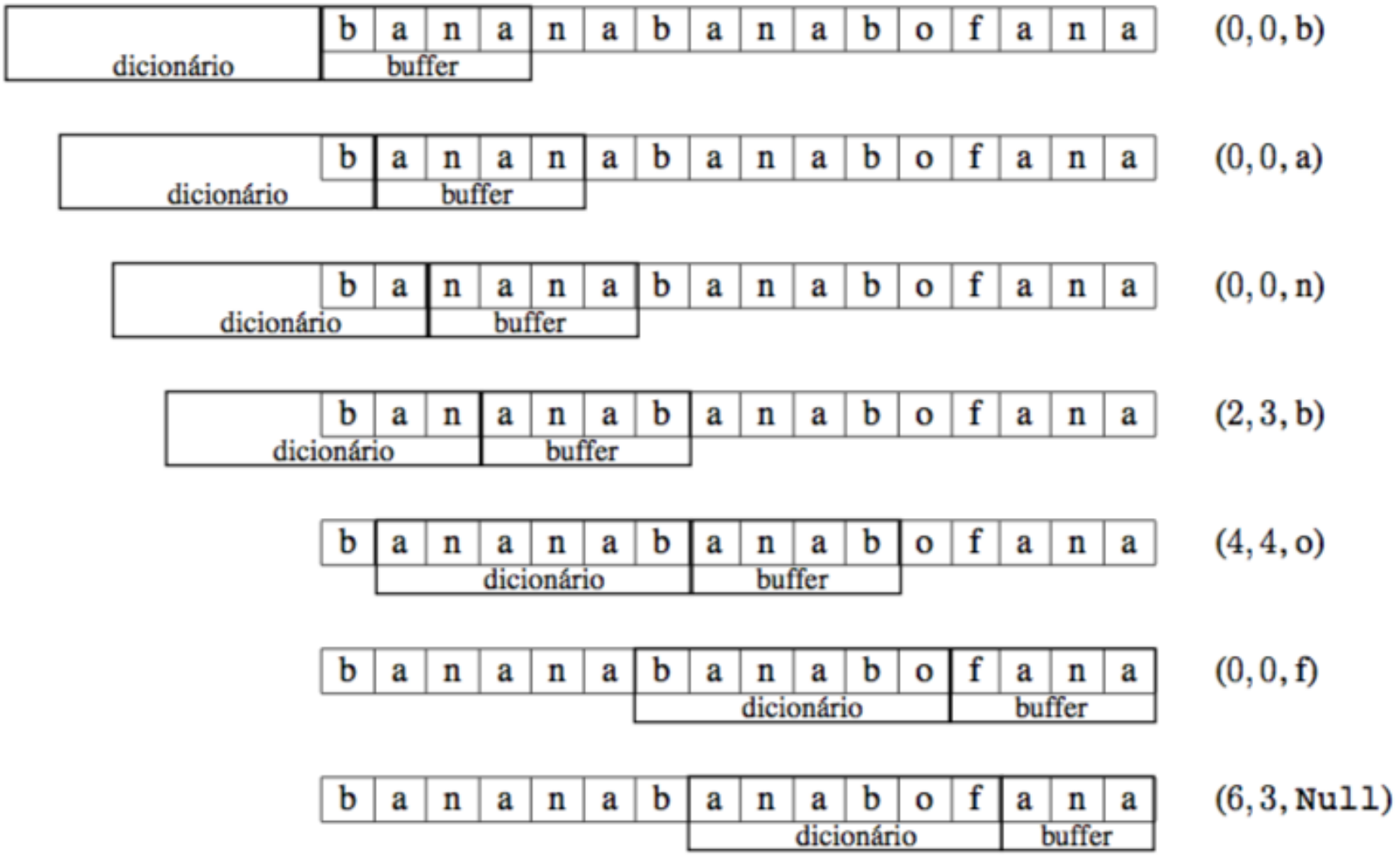
- Essa versão baseia-se em duas janelas contíguas de tamanho fixo: o **dicionário** (últimos símbolos codificados) e o **buffer** (próximos símbolos a codificar).
- A posição das janelas vai sendo deslocada à medida que os símbolos vão sendo codificados.
- O algoritmo LZ77 divide a sequência de símbolos a codificar num conjunto de subsequências, e associa a cada subsequência um código composto por dois inteiros e um caractere.

Algoritmo LZ77 - Codificador

1. Inicializa-se o *buffer* (de dimensão N_b) com os primeiros N_b símbolos da sequência a codificar.
2. Inicialmente o dicionário não contém nenhum símbolo ($N_d = 0$).
3. Enquanto houver símbolos para codificar:
   - Identificar no dicionário a maior sequência de símbolos que também esteja presente no *buffer*.
   - Associar à essa sequência o código (p, l, c), onde
     - p é a quantidade de símbolos que se deve recuar no dicionário a contar da primeira posição do *buffer*.
     - l é o comprimento da sequência.
     - c é o símbolo que se segue à sequência.
   - Deslocar as janelas (dicionário e *buffer*) de $l + 1$ símbolos.

Algoritmo LZ77 - Codificador

Sequência: “bananabanabofana” com $N_d = 6$ e $N_b = 4$.

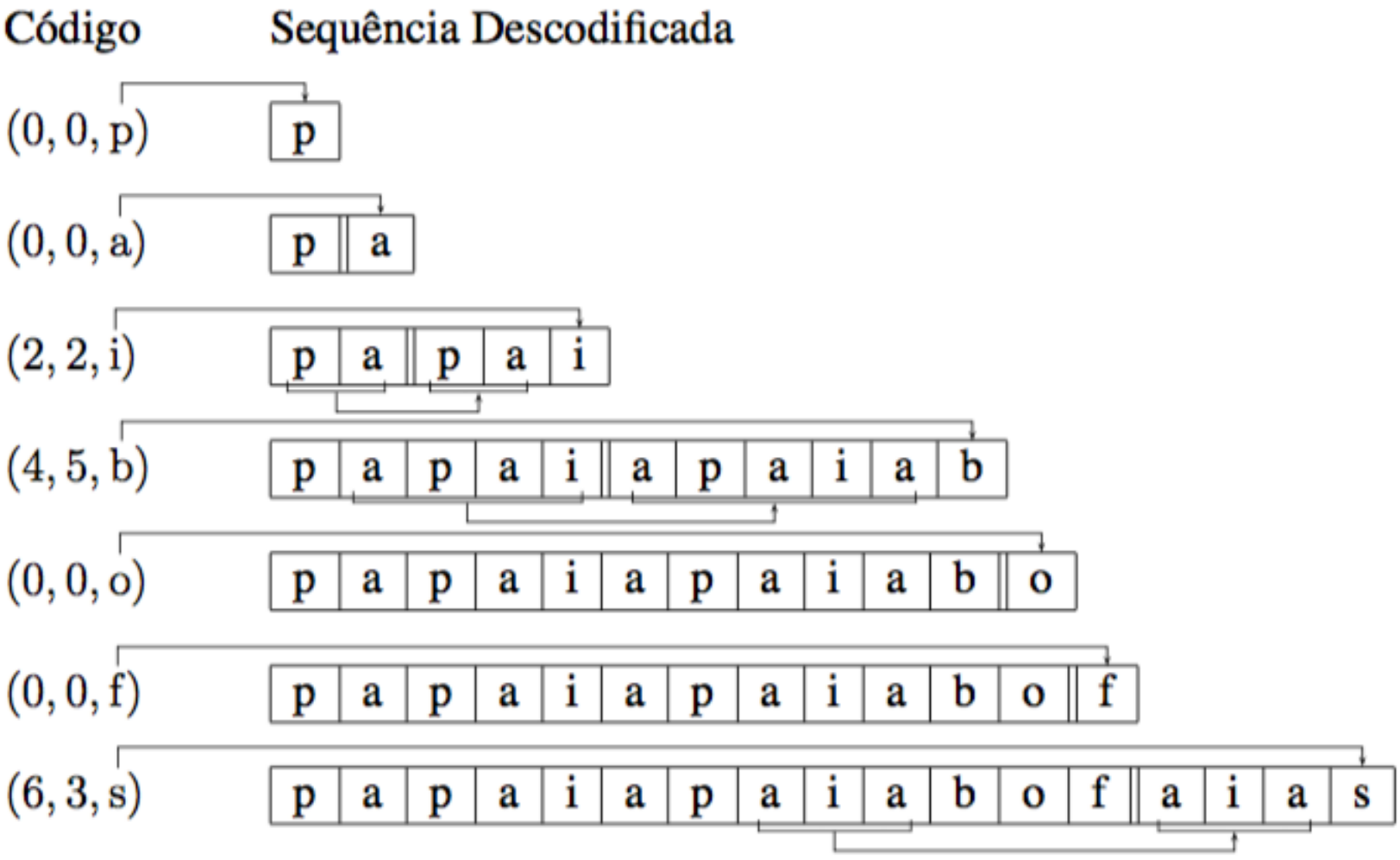


Algoritmo LZ77 - Decodificador

- O processo de descodificação consiste somente em interpretar as palavras de código recebidas.
- À medida que a sequência é formada, serve de base para descodificar futuras palavras de código.
- O slide seguinte mostra o esquema de descodificação da seguinte sequência de palavras de código:

  (0,0,p), (0,0,a), (2,2,i), (4,5,b), (0,0,o), (0,0,f), (6,3,s).

Algoritmo LZ77 - Decodificador



Algoritmo LZ78

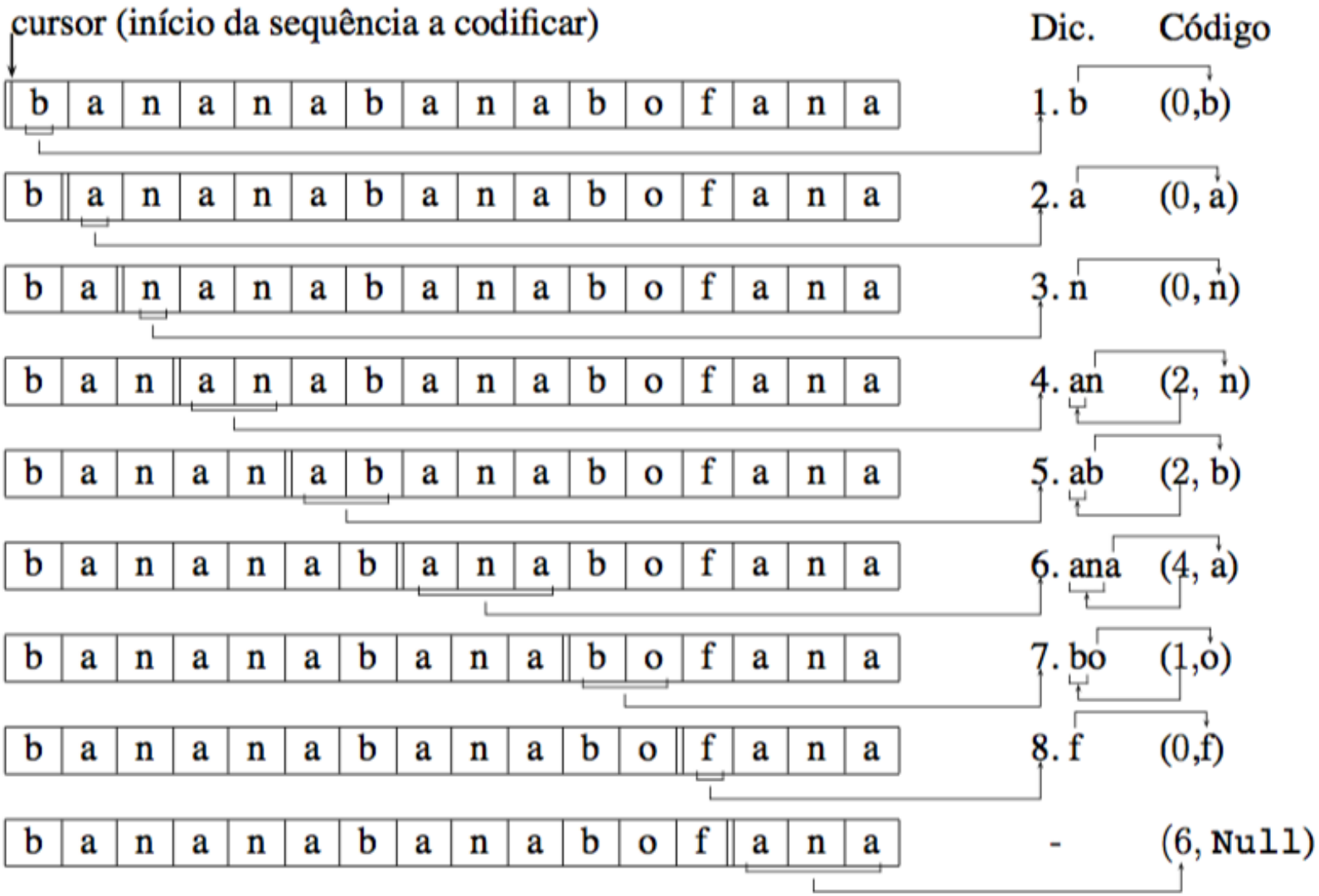
- Esse algoritmo de compressão Lempel-Ziv difere do LZ77 na forma como atualiza o dicionário.
- Agora, em vez de ter uma janela deslizante, o dicionário é uma tabela de sequências que já tenham sido analisadas no processo de codificação.
- Não existe limite no número de sequências que formam o dicionário, o que pode se tornar um problema!
- O algoritmo LZ78 codifica novas sequências de símbolos com apenas um inteiro e um caractere.

Algoritmo LZ78 - Codificador

1. Inicialmente o dicionário não contém nenhuma sequência.
2. $string = Null$.
3. Enquanto houver símbolos para codificar:
   - c = próximo símbolo.
   - Se $string + c$ é uma sequência presente no dicionário:
     - $string = string + c$.
   - Caso contrário:
     - Enviar o código: (índice da *string* no dicionário, c).
     - Atualizar o dicionário com $string + c$. Ao ser adicionada no dicionário, essa sequência recebe um índice.
     - $string = Null$.

Algoritmo LZ78 - Codificador

Sequência a codificar: “bananabanabofana”.



Algoritmo LZ78 - Decodificador

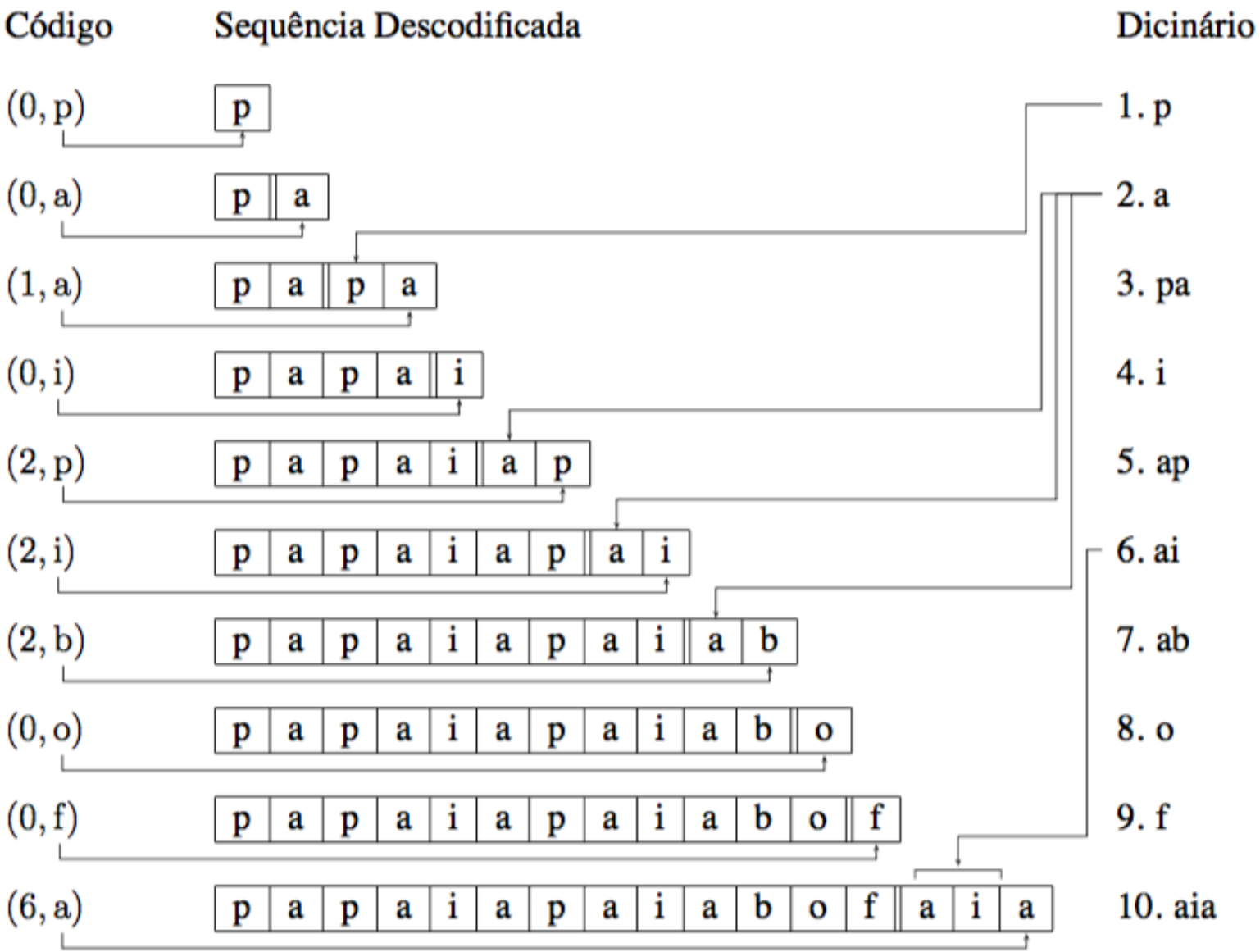
- O processo de descodificação é extremamente simples: a saída é composta pela sequência correspondente ao índice enviado e acrescenta-se no fim dessa sequência o símbolo enviado.

- Por fim, a sequência mais o símbolo enviados são adicionados ao dicionário, com índice associado, e o processo é repetido para a próxima palavra de código.

- O slide seguinte mostra o esquema de descodificação da seguinte sequência de palavras de código:

  (0,p), (0,a), (1,a), (0,i), (2,p), (2,i), (2,b), (0,o), (0,f), (6,a)

Algoritmo LZ78 - Decodificador



Considerações

- A ideia dos métodos de compressão Lempel-Ziv é substituir uma sequência de símbolos por um apontador (ou referência) a uma ocorrência anterior dessa sequência.
- Os algoritmos Lempel-Ziv são populares pela sua velocidade e "universalidade", pois são capazes de produzir um código sem conhecer a estatística da fonte.
- Principais versões dos algoritmos Lempel-Ziv: LZ77 e LZ78.
- O algoritmo LZ77 é baseado em janelas contíguas de tamanho fixo: dicionário e *buffer*. Já o LZ78 não impõe limite para o tamanho do dicionário.

Considerações

- Ao não limitar o tamanho do dicionário, um problema clássico do algoritmo LZ78 é a memória necessária.
- Outro problema que isso causa é de espaço de endereçamento: os códigos das entradas no dicionário crescem junto com ele.
- Várias abordagens podem ser adotadas para lidar com esses problemas, as mais simples são: congelamento (mais nenhuma entrada é adicionada) e esvaziamento (total ou das entradas mais antigas) do dicionário.

Considerações

- Na codificação Lempel-Ziv, ganhos são obtidos apenas para grandes sequências de dados de entrada.
- Os métodos Lempel-Ziv apresentam desvantagens importantes em um ambiente de recuperação de informação.
- Uma muito crítica é a necessidade de iniciar a decodificação desde o início do arquivo comprimido, o que torna o acesso randômico muito caro.
- Outra dificuldade é pesquisar no arquivo comprimido sem descomprimir.

Exercício

- Dada a seguinte sequência S com 16 símbolos:

  $S = [a\ b\ a\ a\ b\ a\ b\ b\ b\ b\ b\ b\ b\ a\ b\ a]$

- Codifique a sequência S usando o algoritmo LZ77, dado que o comprimento do dicionário é 4 e do *buffer* é 6.

- Codifique a sequência S usando o algoritmo Lempel-Ziv 78.