SBVORIN: Organização e Recuperação da Informação

Aula 11: Conceitos e Algoritmos de Compressão de Dados

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Introdução

- A compressão reduz o tamanho dos arquivos:
  - Para economizar espaço de armazenamento;
  - Para economizar tempo em transmissões;
  - Muitos arquivos tem muitas redundâncias;
- Quem precisa de compressão?
  - Lei de Moore: Número de transistores em um chip dobra a cada 18-24 meses;
  - Lei de Parkinson: os dados se expandem para preencher o espaço disponível;
  - Texto, imagens, som, vídeo, ...

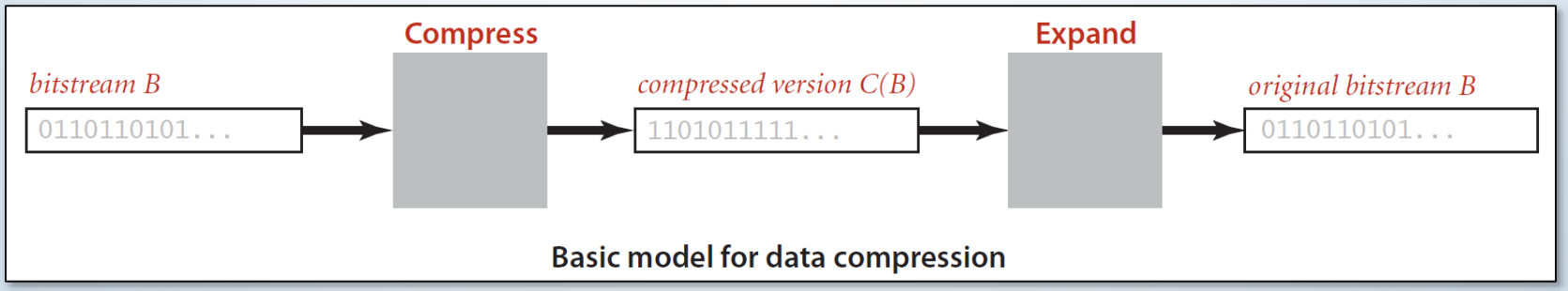
> *" Everyday, we create 2.5 quintillion bytes of data—so much that 90% of the data in the world today has been created in the last two years alone. "* — *IBM report on big data (2011)*

Introdução
Aplicações

- Compressão de arquivos genéricos:
  - Arquivos: GZIP, BZIP, 7z, RAR, ...
  - Sistemas de Arquivos: NTFS, HFS+, ZFS, ...
- **Multimídia:**
  - Imagens: GIF, JPEG, PNG, ...
  - Áudios: MP3, ...
  - Vídeos: MPEG, DivX, HDTV, ...
- **Comunicação:**
  - Fax, Modem, Skype, WhatsApp, Telegram, ...
- **Banco de Dados:**
  - Google, Facebook, Instagram, ...

Compressão e Expansão Sem Perdas

- **Problema:** comprimir um fluxo de bits, ou seja, representar um dado fluxo de bits por outro mais curto;
- **Definições:**
  - **Mensagem:** Dado binário B a ser comprimido;
  - **Compressor (*Compress*):** Gera a representação "comprimida" representada por $C(B)$ utilizando-se menos bits;
  - **Expansor (*Expand*):** Reconstrói o dado binário B original;
- **Modelo Básico para Compressão de Dados:**
  - **Notação:** $|B|$ é o número de bits de B;
  - **Taxa de compressão:** $\frac{|C(B)|}{|B|}$ ➔ Espera-se que a taxa seja estritamente menor que 1;
- **Desafio:** obter a menor taxa de compressão possível!

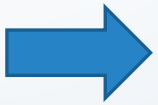


Basic model for data compression

Exemplo
Compressão de Genoma

- **Ideia:** não precisamos usar 8 bits para representar os caracteres de um alfabeto pequeno;
- Em código ASCII, o genoma `ATAGATGCATAGCGCATAGCTAGATGTGCTAGC` usa 264 bits (8×33);
- O alfabeto dos genomas tem apenas 4 letras, podendo usar apenas 2 bits por caractere;
- **Genoma:** Strings a serem utilizadas no alfabeto {`A, C, T, G`}

char	hex	binary
A	41	01000001
C	43	01000011
T	54	01010100
G	47	01000111



char	binary
A	00
C	01
T	10
G	11

Exemplo
Compressão de Genoma

- **Entrada e Saída Padrão Binária (*Binary Standard Input and Standard Output*):** implementadas nas classes `BinaryStdIn` e `BinaryStdOut` contidas no pacote `aesd.utils` do nosso projeto.
- Alguns métodos das classes:

public class BinaryStdIn	
boolean readBoolean()	*read 1 bit of data and return as a* boolean *value*
char readChar()	*read 8 bits of data and return as a* char *value*
char readChar(int r)	*read* r *(between 1 and 16) bits of data and return as a* char *value*
[similar methods for byte *(8 bits);* short *(16 bits);* int *(32 bits);* long *and* double *(64 bits)]*	
boolean isEmpty()	*is the bitstream empty?*
void close()	*close the bitstream*

public class BinaryStdOut	
void write(boolean b)	*write the specified bit*
void write(char c)	*write the specified 8-bit* char
void write(char c, int r)	*write the* r *(between 1 and 16) least significant bits of the specified* char
[similar methods for byte *(8 bits);* short *(16 bits);* int *(32 bits);* long *and* double *(64 bits)]*	
void close()	*close the bitstream*

Exemplo
Compressão de Genoma

- API da classe Alphabet (pacote `aesd.algorithms.strings`):

public class Alphabet	
Alphabet(String s)	*create a new alphabet from chars in* `s`
char toChar(int index)	*convert index to corresponding alphabet char*
int toIndex(char c)	*convert* `c` *to an index between* `0` *and* `R-1`
boolean contains(char c)	*is* `c` *in the alphabet?*
int R()	*radix (number of characters in alphabet)*
int lgR()	*number of bits to represent an index*
int[] toIndices(String s)	*convert* `s` *to base-R integer*
String toChars(int[] indices)	*convert base-R integer to string over this alphabet*

Alphabet API

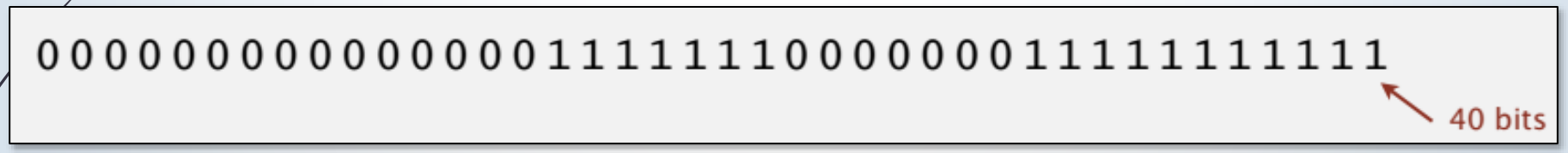
Exemplo
Compressão de Genoma

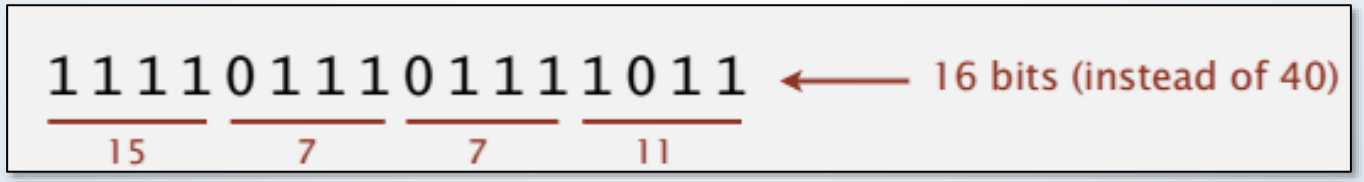
```
public static void compress() {

    Alphabet DNA = Alphabet.DNA;

    String s = BinaryStdIn.readString();
    int n = s.length();

    BinaryStdOut.write( n );

    // escreve o código de 2 bits
    // para cada caractere
    for ( int i = 0; i < n; i++ ) {
        int d = DNA.toIndex( s.charAt( i ) );
        BinaryStdOut.write( d, 2 );
    }

    BinaryStdOut.close();

}
```

```
public static void expand() {

    Alphabet DNA = Alphabet.DNA;

    int n = BinaryStdIn.readInt();

    // lê dois bits e escreve um caractere
    for ( int i = 0; i < n; i++ ) {
        char c = BinaryStdIn.readChar( 2 );
        BinaryStdOut.write( DNA.toChar( c ), 8 );
    }

    BinaryStdOut.close();

}
```

Codificação de Comprimento de Carreira
(Run-length Encoding)

- **Tipo Simples de Redundância em um Fluxo de Bits:** Longas repetições de bits sequenciais
- **Representação:** Contagens de 4 bits para representar execuções alternadas de 0's e 1's
- **Exemplo:** 15 0's, 7 1's, 7 0's e 11 1's





Codificação de Comprimento de Carreira
(Run-length Encoding)

```
public class RunLength
{
   private final static int R   = 256;
   private final static int lgR = 8;

   public static void compress()
   {  /* see textbook */  }

   public static void expand()
   {
      boolean bit = false;
      while (!BinaryStdIn.isEmpty())
      {
         int run = BinaryStdIn.readInt(lgR);
         for (int i = 0; i < run; i++)
            BinaryStdOut.write(bit);
         bit = !bit;
      }
      BinaryStdOut.close();
   }

}
```

maximum run-length count

number of bits per count

read 8-bit count from standard input

write 1 bit to standard output

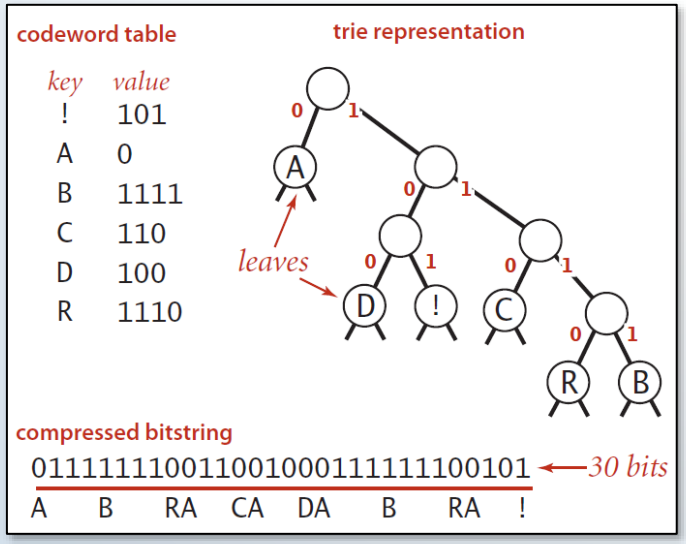
pad 0s for byte alignment

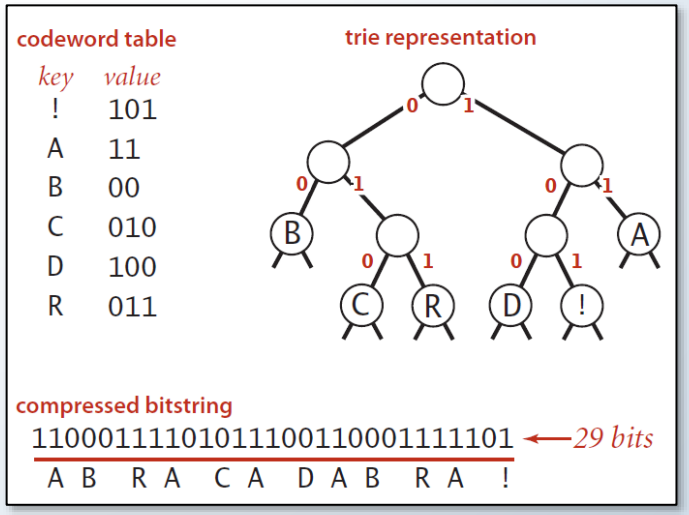
Codificação de Comprimento de Carreira
(Run-length Encoding)

```
public static void compress() {
    char run = 0;
    boolean old = false;
    while ( !BinaryStdIn.isEmpty() ) {

        boolean b = BinaryStdIn.readBoolean();

        if ( b != old ) {
            BinaryStdOut.write( run, LG_R );
            run = 1;
            old = !old;
        } else {
            if ( run == R - 1 ) {
                BinaryStdOut.write( run, LG_R );
                run = 0;
                BinaryStdOut.write( run, LG_R );
            }
            run++;
        }

    }
        BinaryStdOut.write( run, LG_R );
    BinaryStdOut.close();

}
```

Compressão de Huffman
Ideia

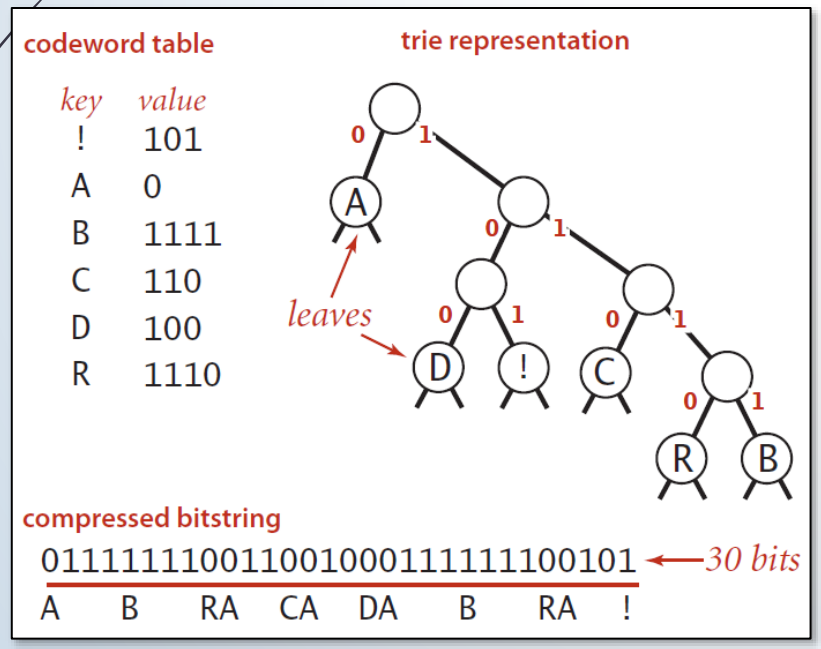
- Utiliza diferentes números de bits para codificar diferentes caracteres;
- Como evitar ambiguidade?
  - Garantir que nenhuma palavra-código (*codeword*) seja prefixo de outra;
- **Exemplos:**
  - Código de comprimento fixo;
  - Adicionar caractere de parada especial a cada palavra-código;
  - Código geral sem prefixo.

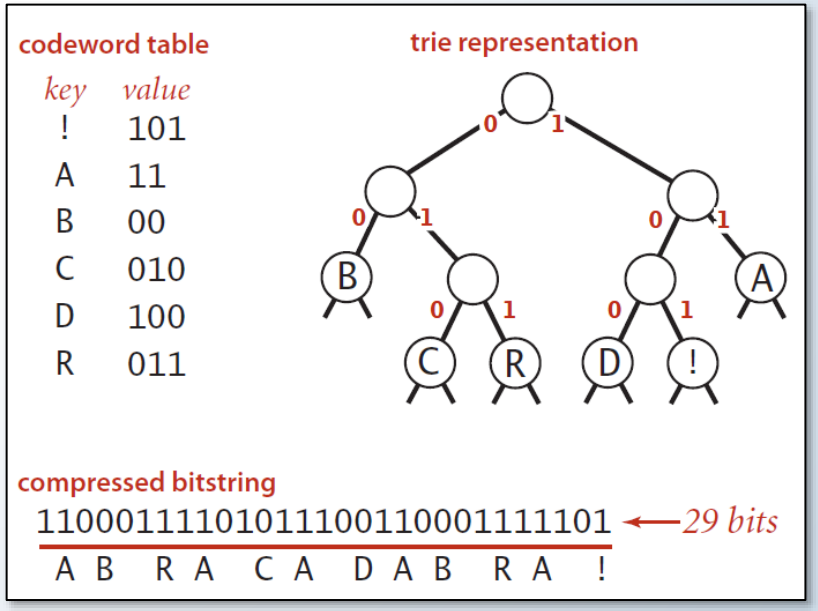




Compressão de Huffman
Representação

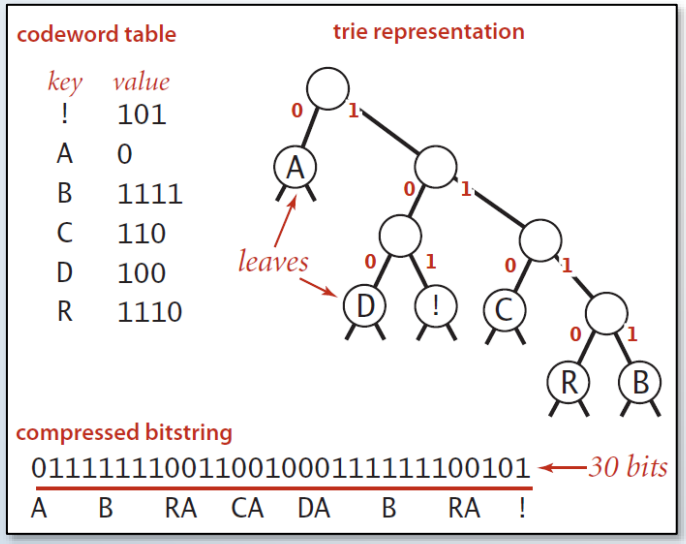
- Como representar o código sem prefixo?
  - Tries
    - Caracteres são as folhas
    - Palavra-código (*codeword*) é o caminho da raiz até a folha.

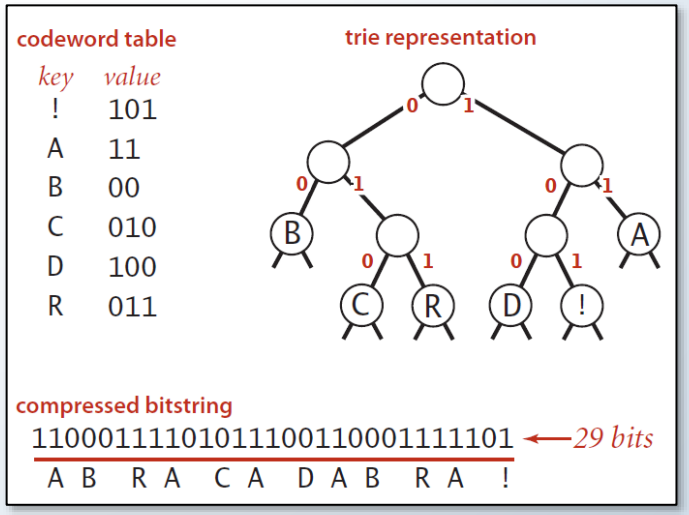




Compressão de Huffman
Compressão e Expansão

- **Compressão:**
  - **Método 01:** Inicia na folha, segue o caminho até a raiz e imprime os bits ao contrário;
  - **Método 02:** Criar a tabela de Strings de pares de chave-valor;
- **Expansão:**
  - Inicia na raiz;
  - Caminhe para a esquerda se o bit for 0; Caminhe para a direita se o bit for 1;
  - Se for nó folha, imprimir o caractere e retornar para a raiz.





Compressão de Huffman

Representação na Trie

```
private static class Node implements Comparable<Node>
{
   private final char ch;   // used only for leaf nodes
   private final int freq;  // used only for compress
   private final Node left, right;

   public Node(char ch, int freq, Node left, Node right)
   {
      this.ch    = ch;
      this.freq  = freq;
      this.left  = left;
      this.right = right;
   }

   public boolean isLeaf()
   {  return left == null && right == null; }

   public int compareTo(Node that)
   {  return this.freq - that.freq;  }
}
```

Compressão de Huffman
Compressão

```
public static void compress() {
   String s = BinaryStdIn.readString();
   char[] input = s.toCharArray();

   // cálculo da tabela de códigos st[]
   // discutido mais adiante

   for (int i = 0; i < input.length; i++) {
      String code = st[input[i]];
      for (int j = 0; j < code.length(); j++)
      if (code.charAt(j) == '1')
           BinaryStdOut.write(true);
      else BinaryStdOut.write(false);
   }
   BinaryStdOut.close();
}
```

Compressão de Huffman
Expansão

```
public void expand()
{
   Node root = readTrie();
   int N = BinaryStdIn.readInt();

   for (int i = 0; i < N; i++)
   {
      Node x = root;
      while (!x.isLeaf())
      {
         if (!BinaryStdIn.readBoolean())
            x = x.left;
         else
            x = x.right;
      }
      BinaryStdOut.write(x.ch, 8);
   }
   BinaryStdOut.close();
}
```

read in encoding trie

read in number of chars

expand codeword for i^{th} char

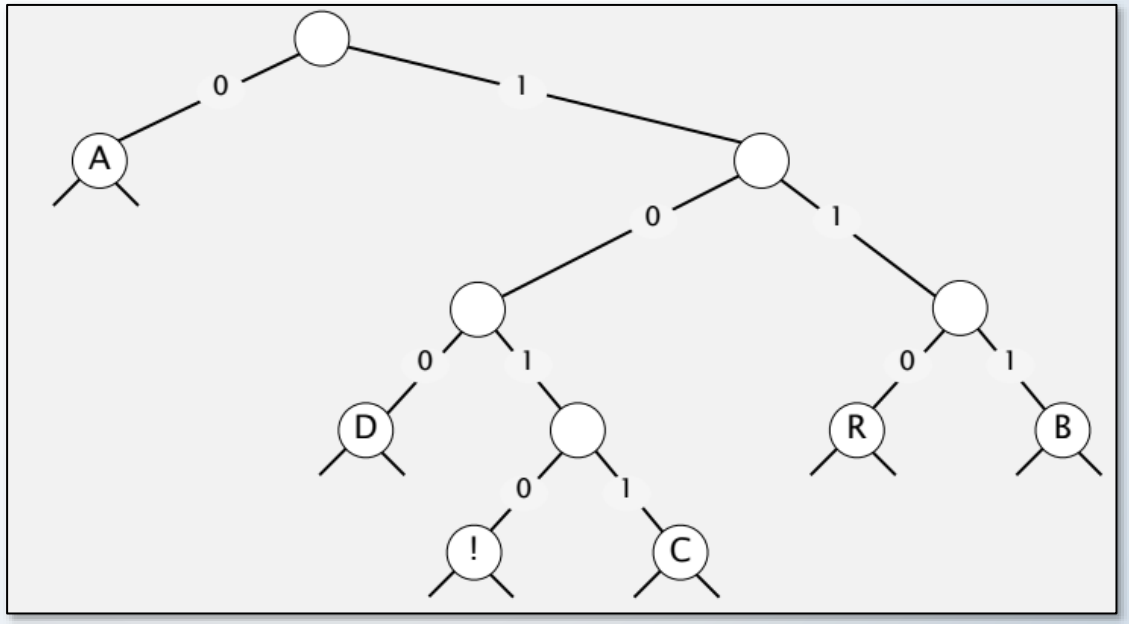
Compressão de Huffman
Construção da Trie

- Como encontrar o melhor código sem prefixo?
  - A tabela de códigos de Huffman não é universal, pois é construída sob medida para a String a ser codificado de tal modo que o comprimento da cadeia codificada seja o menor possível;
  - Primeiro, deve-se construir a Trie do código, depois extrair dela a tabela de códigos. Dada a String a ser codificada, a Trie é construída assim:
    1. Determine a frequência, ou seja, o número de ocorrências, de cada caractere da String original;
    2. Para cada caractere, crie um nó e armazene o caractere e sua frequência nos campos `ch` e `freq`;
    3. Organizar o conjunto de nós em uma Trie conforme o seguinte algoritmo;
  - **Algoritmo de Construção da Trie:**
    1. No início de cada iteração, há um conjunto de Tries mutuamente disjuntas, ou seja, no início da primeira iteração, cada Trie tem um único nó;
    2. Escolha duas Tries cujas raízes, diga-se `x` e `y`, tenham frequência mínima;
    3. Crie um novo nó pai (`parent`) e faça com que `x` e `y` sejam filhos desse nó;
    4. Faça `parent.freq` igual a `x.freq + y.freq`;
    5. Repita esse processo até que o conjunto de Tries tenha uma só Trie.

Compressão de Huffman
Construção da Trie

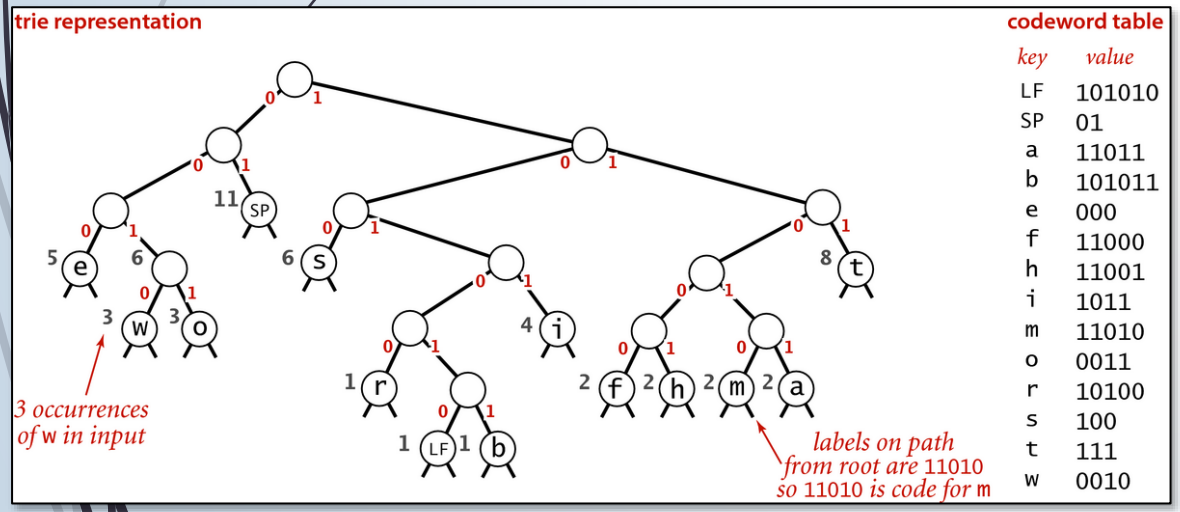
- **Exemplo de Construção:**
  - **Entrada:** A B R A C A D A B R A !

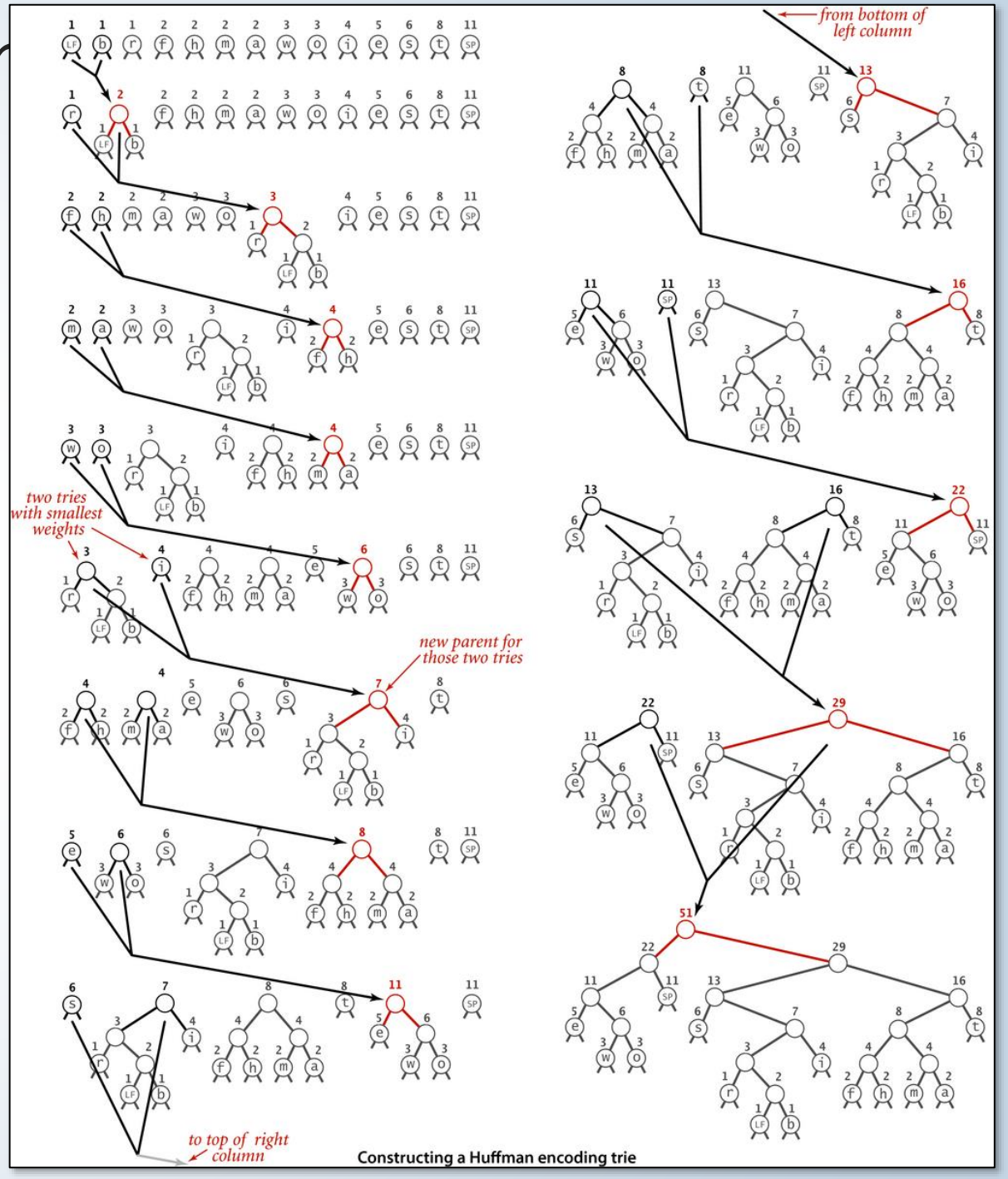
char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	



Compressão de Huffm
Construção da Trie

- **Outro Exemplo de Construção:**
  - **Entrada:** `it was the best of times it was the worst of timesLF`
    - **LF:** o caractere que indica fim de linha
    - **SP:** o caractere que indica espaço





Compressão de Huffman
Construção da Trie

- O método de construção da Trie utiliza uma fila de prioridades;

```
private static Node buildTrie(int[] freq) {
   MinPQ<Node> pq = new MinPQ<Node>();
   for (char c = 0; c < R; c++)
      if (freq[c] > 0)
         pq.insert(new Node(c, freq[c], null, null));

   while (pq.size() > 1) {
      Node x = pq.delMin();
      Node y = pq.delMin();
      Node parent = new Node('\0', x.freq + y.freq, x, y);
      pq.insert(parent);
   }
   return pq.delMin();
}
```

- A tabela de códigos `st[]`, usada na codificação, é calculada a partir da Trie construída.

```
private static String[] buildCode(Node root) {
   String[] st = new String[R];
   buildCode(st, root, "");
   return st;
}

private static void buildCode(String[] st, Node x, String s) {
   if (x.isLeaf()) {
      st[x.ch] = s;
      return;
   }
   buildCode(st, x.left, s + '0');
   buildCode(st, x.right, s + '1');
}
```

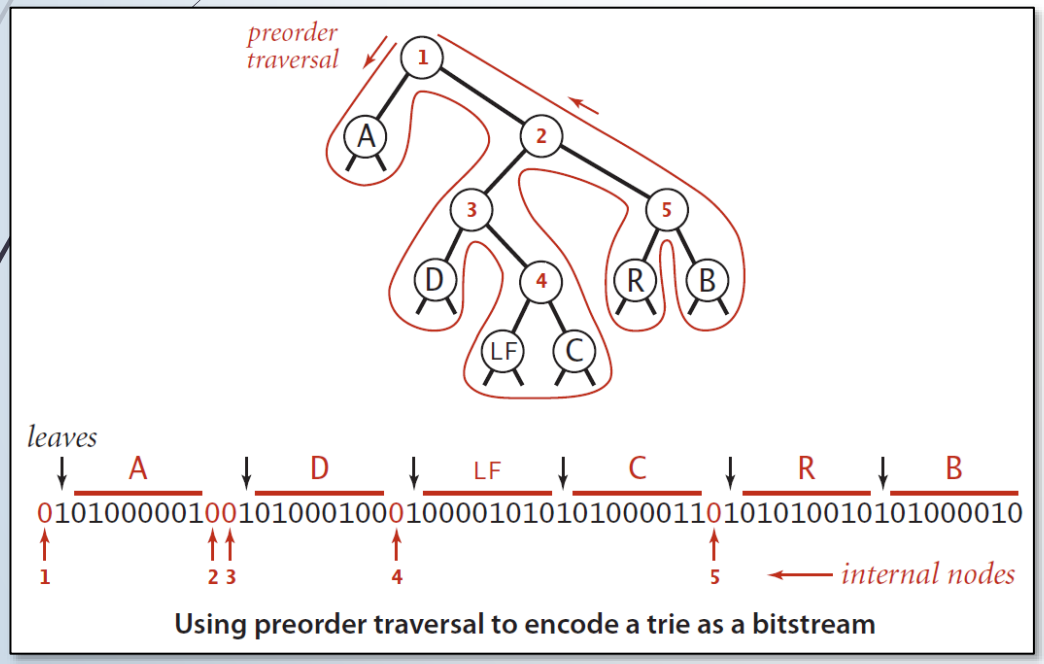
Compressão de Huffman
Construção da Trie

- Conhecidos os métodos `buidTrie()` e `buildCode()`, pode-se completar o método de compressão:

```
public static void compress() {
   String s = BinaryStdIn.readString();
   char[] input = s.toCharArray();

   int[] freq = new int[R];
   for (int i = 0; i < input.length; i++)
      freq[input[i]]++;

   Node root = buildTrie(freq);

   String[] st = new String[R];
   buildCode(st, root, "");

   writeTrie(root);  // discutido abaixo

   BinaryStdOut.write(input.length);

   for (int i = 0; i < input.length; i++) {
      String code = st[input[i]];
      for (int j = 0; j < code.length(); j++)
      if (code.charAt(j) == '1')
           BinaryStdOut.write(true);
      else BinaryStdOut.write(false);
   }
   BinaryStdOut.close();
}
```

Compressão de Huffman
Como Transmitir

- Como escrever a Trie?
  - Escrever o percurso pré-ordem da Trie ➔ Marcar as folhas e nós internos com um bit;
  - Se a mensagem for longa, a sobrecarga da Trie de transmissão é pequena.

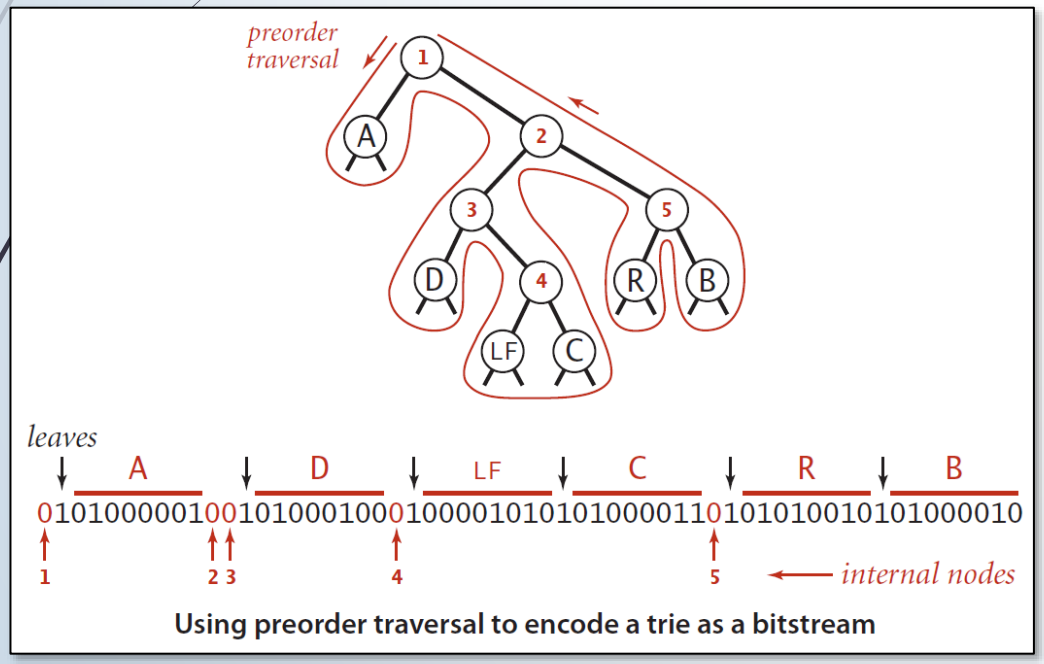


Using preorder traversal to encode a trie as a bitstream

```
private static void writeTrie(Node x)
{
   if (x.isLeaf())
   {
      BinaryStdOut.write(true);
      BinaryStdOut.write(x.ch, 8);
      return;
   }
   BinaryStdOut.write(false);
   writeTrie(x.left);
   writeTrie(x.right);
}
```

Compressão de Huffman
Como Transmitir

- Como ler na Trie?
  - Reconstruir a partir do percurso pré-ordem da Trie.



Using preorder traversal to encode a trie as a bitstream

```
private static Node readTrie()
{
   if (BinaryStdIn.readBoolean())
   {
      char c = BinaryStdIn.readChar(8);
      return new Node(c, 0, null, null);
   }
   Node x = readTrie();
   Node y = readTrie();
   return new Node('\0', 0, x, y);
}
```

arbitrary value
(value not used with internal nodes)

Compressão de Huffman
Visão Geral

- **Modelo Dinâmico:** Use um código sem prefixo personalizado para cada mensagem;
- **Compressão:**
  - Ler a mensagem;
  - Construir o melhor código sem prefixo para a mensagem. Como?
  - Escrever o código sem prefixo (como um teste) no arquivo;
  - Comprimir a mensagem usando um código sem prefixo;
- **Expansão:**
  - Ler o código sem prefixo (como uma Trie) do arquivo;
  - Ler a mensagem compactada e expandir usando a Trie.

Compressão de Lempel-Ziv-Welch (LZW)

- Individualmente, prepare um relatório explicando e exemplificando essa técnica;
- O material se encontra a partir da página 839 da obra SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Boston: Pearson Education, 2011. 955 p.

Codificação de Shannon-Fano

- A codificação de Shannon-Fano é uma técnica de codificação probabilística;
- Individualmente, prepare um relatório explicando e exemplificando essa técnica;
- Busque por artigos, exemplos etc. na web.

Bibliografia

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Boston: Pearson Education, 2011. 955 p.

GOODRICHM M. T.; TAMASSIA, R. **Estruturas de Dados & Algoritmos em Java**. Porto Alegre: Bookman, 2013. 700 p.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos – Teoria e Prática**. 3. ed. São Paulo: GEN LTC, 2012. 1292 p.