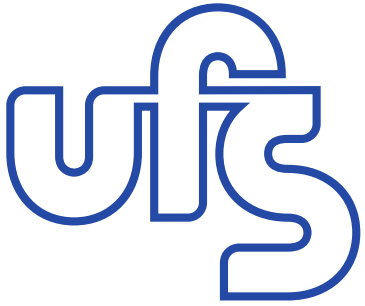
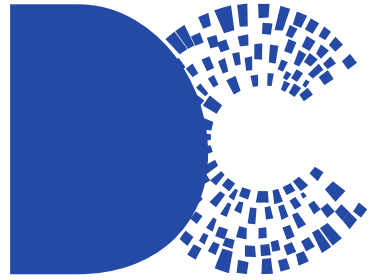




Compressão de dados

Projeto e Análise de Algoritmos

Bruno Prado

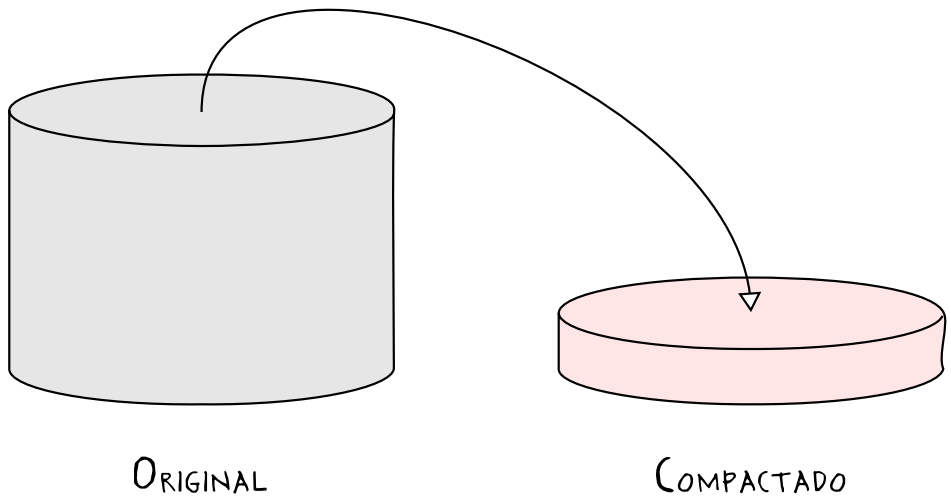
Departamento de Computação / UFS

Introdução

- Por que realizar a compressão de dados?

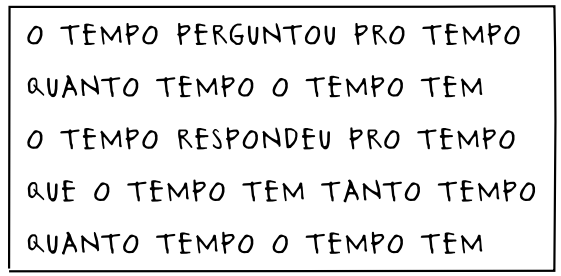
Introdução

- Por que realizar a compressão de dados?
  - Melhorar a eficiência de armazenamento
  - Reduzir o custo para transmissão de dados



Introdução

- Exploração da redundância dos dados que existem nos diversos tipos de arquivos não processados
  - Letras ou palavras com alto índice de repetição

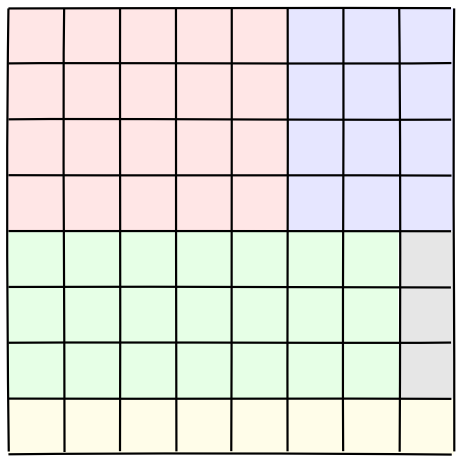


Introdução

- Exploração da redundância dos dados que existem nos diversos tipos de arquivos não processados
  - Letras ou palavras com alto índice de repetição

O TEMPO PERGUNTOU PRO TEMPO
QUANTO TEMPO O TEMPO TEM
O TEMPO RESPONDEU PRO TEMPO
QUE O TEMPO TEM TANTO TEMPO
QUANTO TEMPO O TEMPO TEM

  - Imagens com grandes áreas homogêneas



Introdução

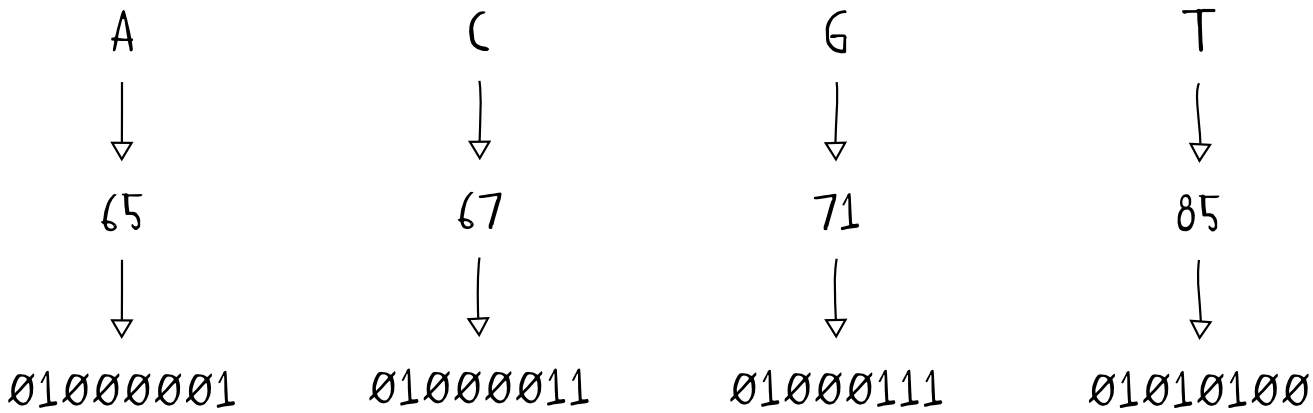
- Tipos de compressão de dados
  - Sem perdas (*lossless*)
    - Os dados comprimidos são reconstruídos exatamente iguais aos dados originais
    - Ex: compactação de arquivos (ZIP, PNG, ...)

Introdução

- Tipos de compressão de dados
  - Sem perdas (*lossless*)
    - Os dados comprimidos são reconstruídos exatamente iguais aos dados originais
    - Ex: compactação de arquivos (ZIP, PNG, ...)
  - Com perdas (*lossy*)
    - Consiste no descarte de parte dos dados para melhorar a taxa de compressão
    - Ex: arquivos multimídia (MP3, JPEG, ...)

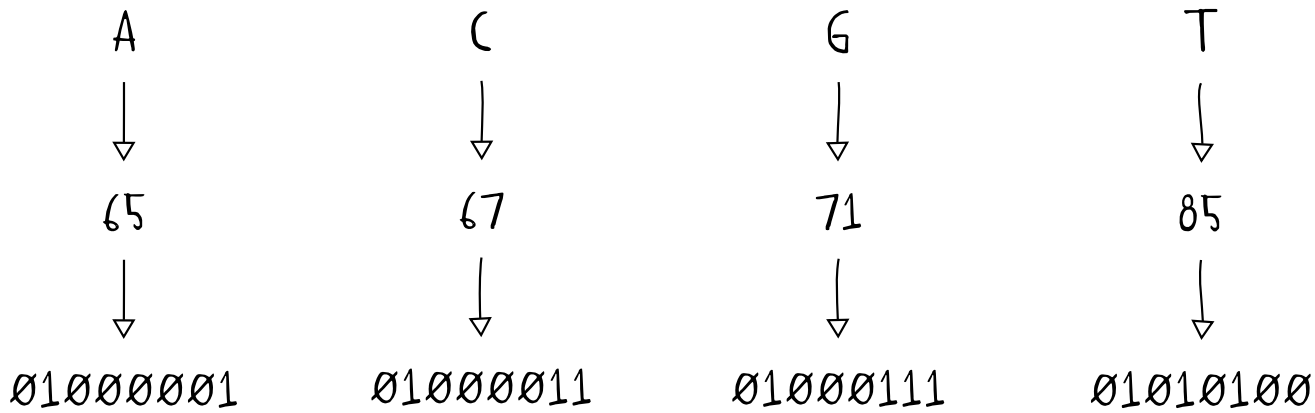
Introdução

- Representação dos dados em binário
  - Texto simples (ASCII)
  - Arquivos binários: arquivos executáveis, imagens, ...



Introdução

- Representação dos dados em binário
  - Texto simples (ASCII)
  - Arquivos binários: arquivos executáveis, imagens, ...



  - A taxa de compressão atingida é dependente das características dos dados da entrada utilizada

$$Taxa\ de\ compressão = 100 \times \frac{Tamanho\ comprimido}{Tamanho\ original}$$

Introdução

- Limitações da compressão de dados
  - É possível existir um algoritmo de compressão universal capaz de sempre reduzir o tamanho qualquer conjunto de dados utilizado como entrada?

Introdução

- Limitações da compressão de dados
  - É possível existir um algoritmo de compressão universal capaz de sempre reduzir o tamanho qualquer conjunto de dados utilizado como entrada?
    - Se assumirmos que isto é possível, a cada execução seria gerada uma saída com tamanho menor

Introdução

- Limitações da compressão de dados
  - É possível existir um algoritmo de compressão universal capaz de sempre reduzir o tamanho qualquer conjunto de dados utilizado como entrada?
    - Se assumirmos que isto é possível, a cada execução seria gerada uma saída com tamanho menor
    - Aplicando uma quantidade suficiente de repetições, o tamanho da saída gerada seria zero (vazia)

Introdução

- Limitações da compressão de dados
  - É possível existir um algoritmo de compressão universal capaz de sempre reduzir o tamanho qualquer conjunto de dados utilizado como entrada?
    - Se assumirmos que isto é possível, a cada execução seria gerada uma saída com tamanho menor
    - Aplicando uma quantidade suficiente de repetições, o tamanho da saída gerada seria zero (vazia)
    - Desta forma, como é gerada uma contradição, a hipótese da existência de um algoritmo de compressão universal não pode ser verdadeira

Introdução

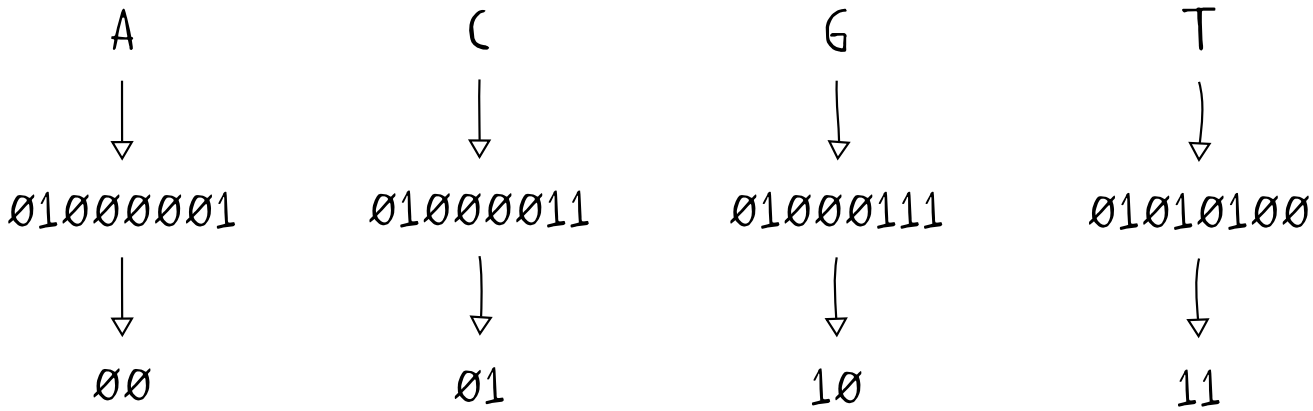
- Limitações da compressão de dados
  - É possível existir um algoritmo de compressão universal capaz de sempre reduzir o tamanho qualquer conjunto de dados utilizado como entrada?
    - Se assumirmos que isto é possível, a cada execução seria gerada uma saída com tamanho menor
    - Aplicando uma quantidade suficiente de repetições, o tamanho da saída gerada seria zero (vazia)
    - Desta forma, como é gerada uma contradição, a hipótese da existência de um algoritmo de compressão universal não pode ser verdadeira
  - Indecidibilidade: não é possível determinar se um algoritmo de compressão é ótimo quando aplicado em uma determinada cadeia de bits

Compressão de dados

- Representação dos dados
  - As cadeias de DNA ou genomas são representadas por um alfabeto de 4 símbolos: A, C, G e T

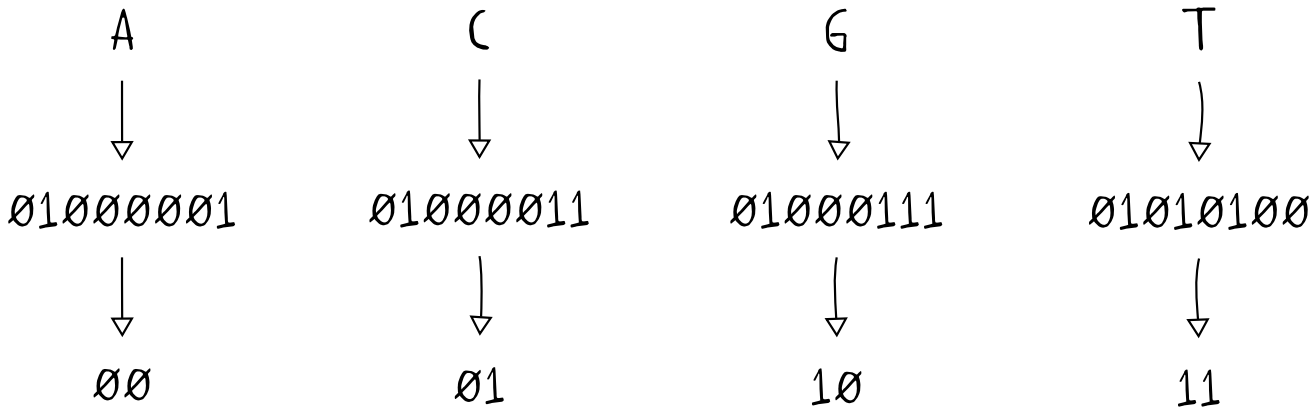
Compressão de dados

- Representação dos dados
  - As cadeias de DNA ou genomas são representadas por um alfabeto de 4 símbolos: *A*, *C*, *G* e *T*
  - Na representação em texto no padrão ASCII, são utilizados caracteres que permitem até 256 símbolos



Compressão de dados

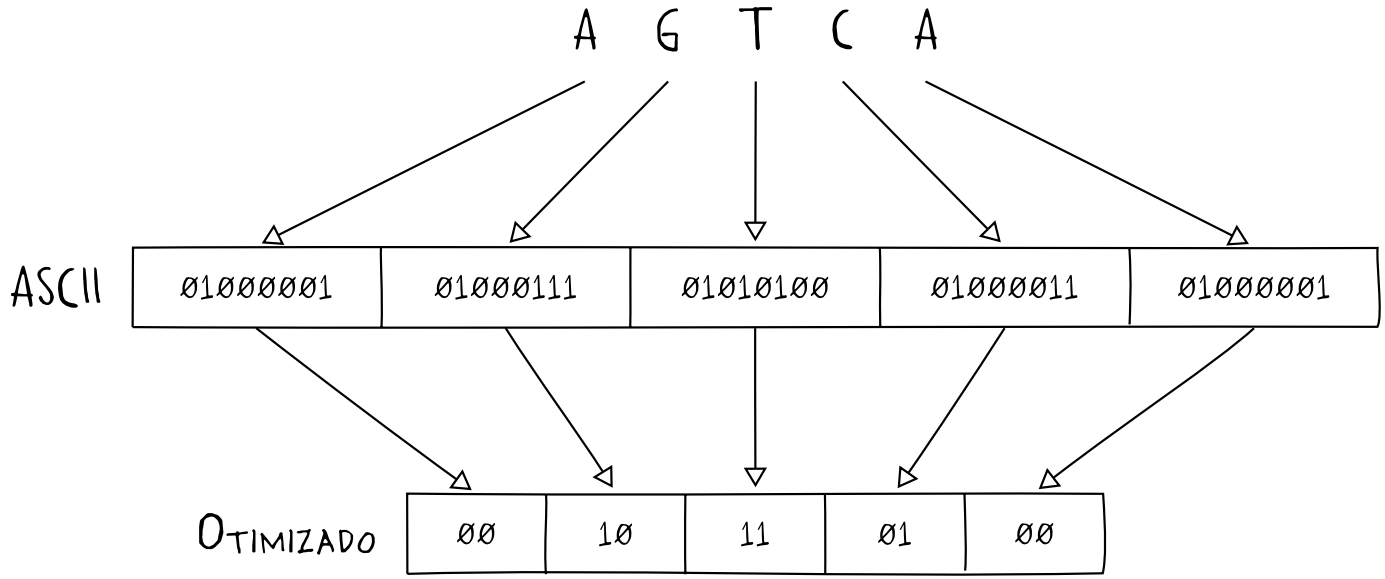
- Representação dos dados
  - As cadeias de DNA ou genomas são representadas por um alfabeto de 4 símbolos: A, C, G e T
  - Na representação em texto no padrão ASCII, são utilizados caracteres que permitem até 256 símbolos



$$\textit{Número de bits} = \log_2 \left| \sum \right|$$

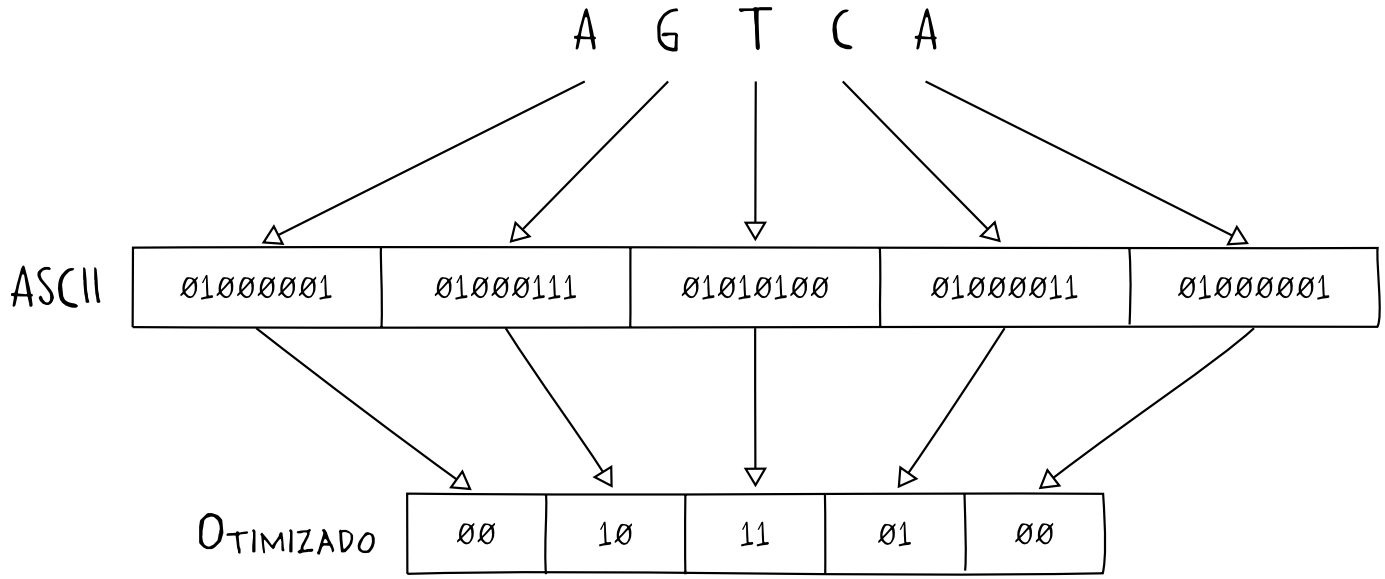
Compressão de dados

- Representação dos dados
  - Para representar os 256 símbolos de texto são necessários 8 bits, enquanto que a representação dos 4 símbolos de DNA são necessários somente 2 bits



Compressão de dados

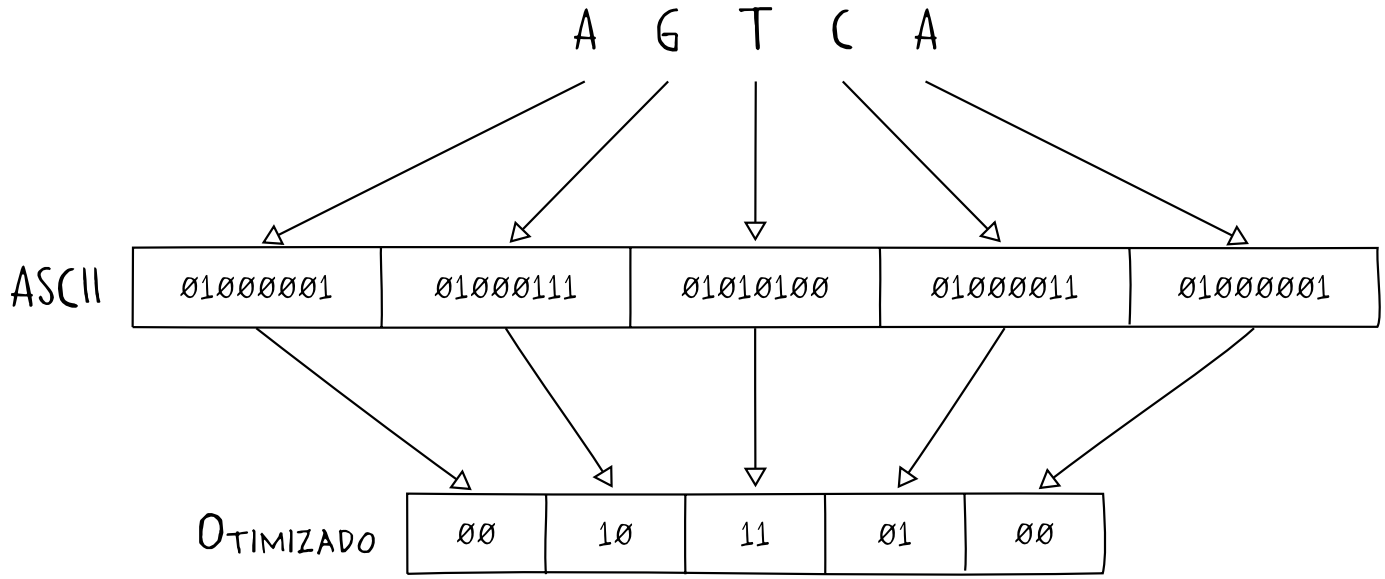
- Representação dos dados
  - Para representar os 256 símbolos de texto são necessários 8 bits, enquanto que a representação dos 4 símbolos de DNA são necessários somente 2 bits



  - É atingida uma taxa de compressão de 25% apenas com uma representação adequada dos dados

Compressão de dados

- Representação dos dados
  - Para representar os 256 símbolos de texto são necessários 8 bits, enquanto que a representação dos 4 símbolos de DNA são necessários somente 2 bits



  - É atingida uma taxa de compressão de 25% apenas com uma representação adequada dos dados
  - No padrão de codificação ASCII, a sequência do genoma humano possui mais de 10^{10} bits

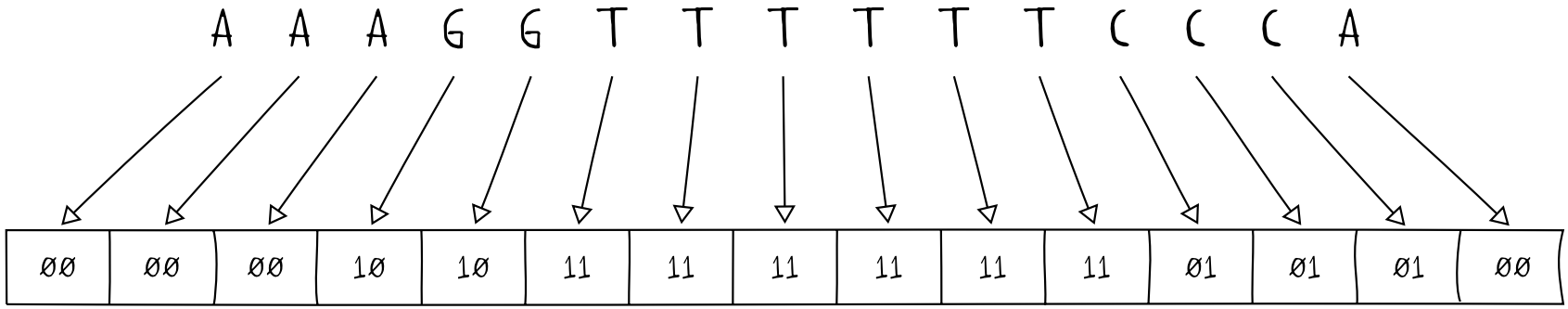
Compressão de dados

- *Run-Length Encoding* (RLE)
  - Esta técnica de compressão consiste em contabilizar a repetição de símbolos em uma sequência

A A A G G T T T T T T C C C A

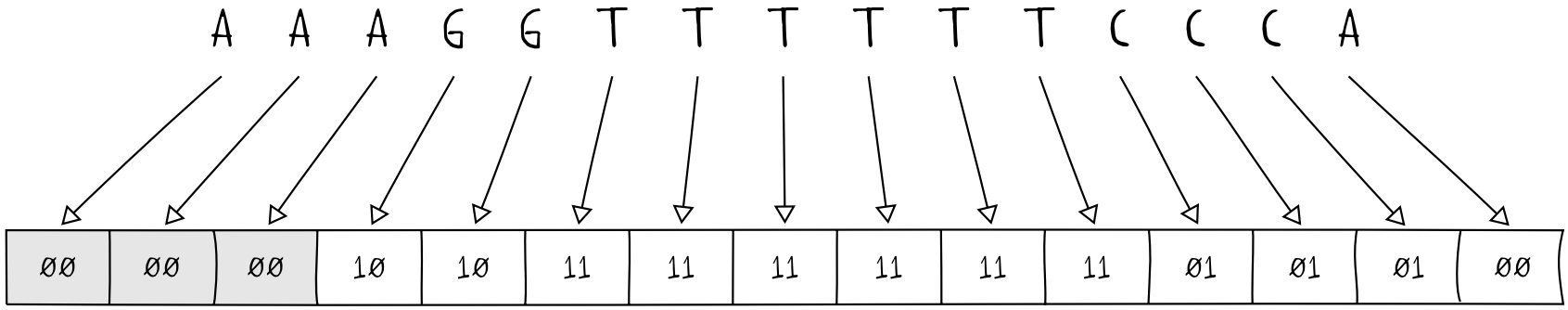
Compressão de dados

- *Run-Length Encoding* (RLE)
  - Esta técnica de compressão consiste em contabilizar a repetição de símbolos em uma sequência



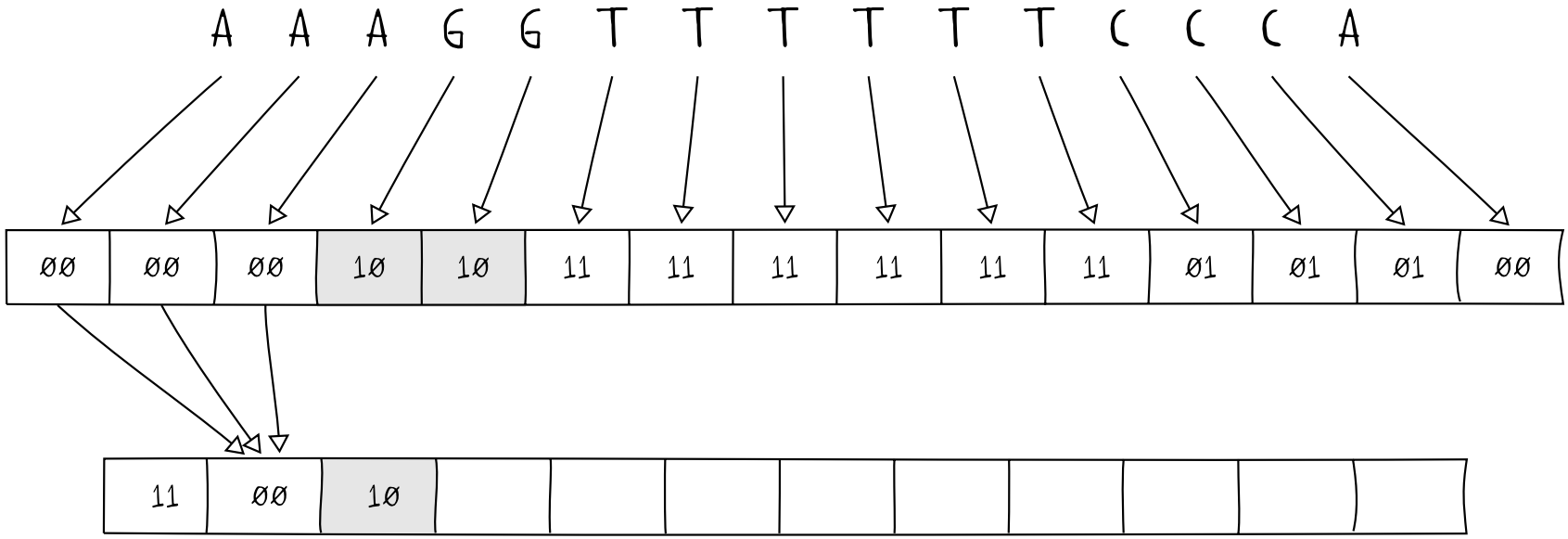
Compressão de dados

- *Run-Length Encoding* (RLE)
  - Esta técnica de compressão consiste em contabilizar a repetição de símbolos em uma sequência



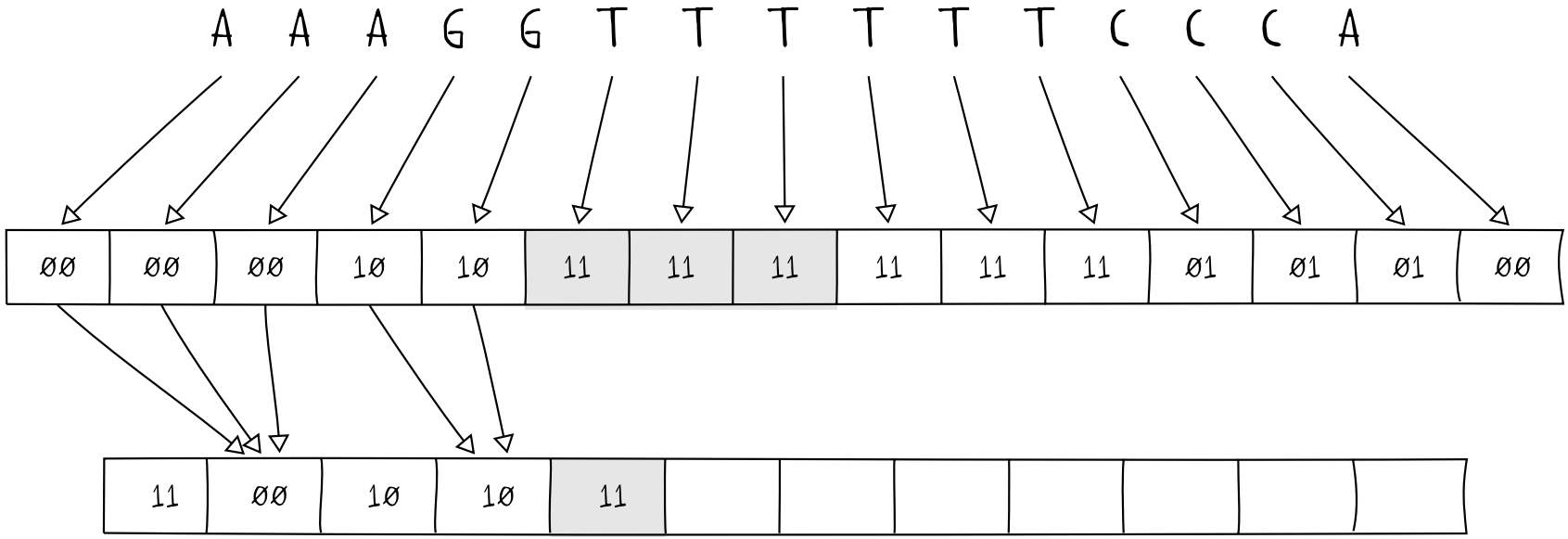
Compressão de dados

- *Run-Length Encoding* (RLE)
  - Esta técnica de compressão consiste em contabilizar a repetição de símbolos em uma sequência



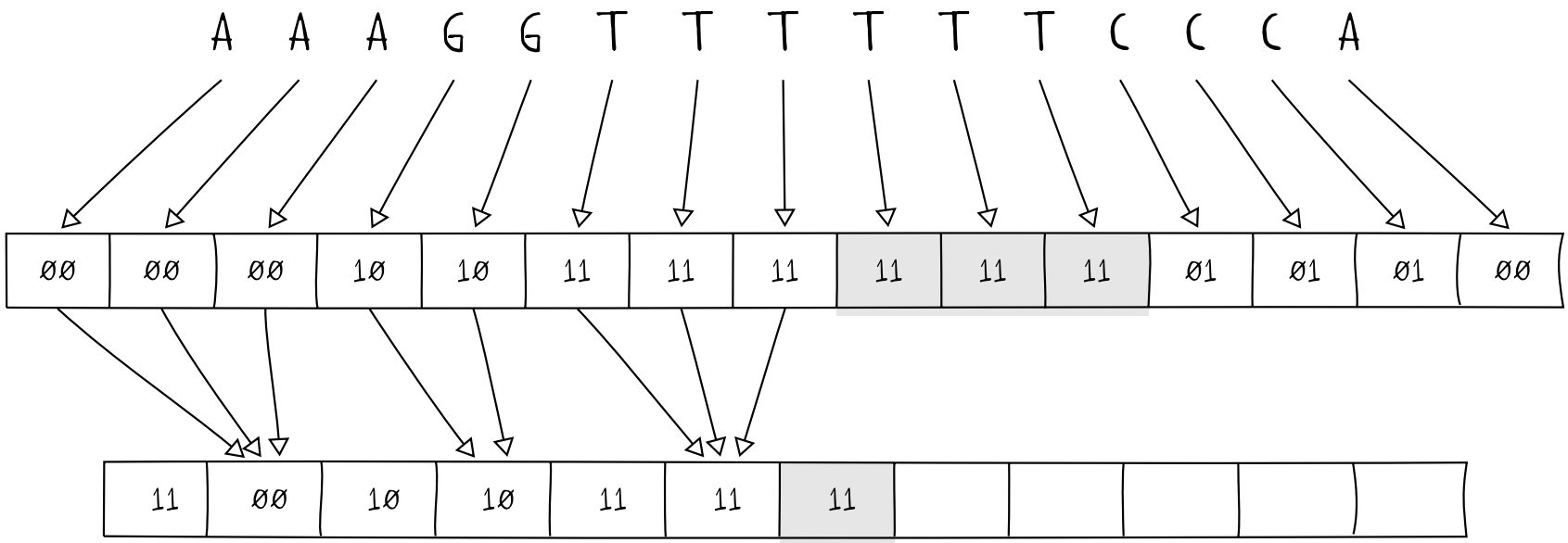
Compressão de dados

- *Run-Length Encoding* (RLE)
  - Esta técnica de compressão consiste em contabilizar a repetição de símbolos em uma sequência



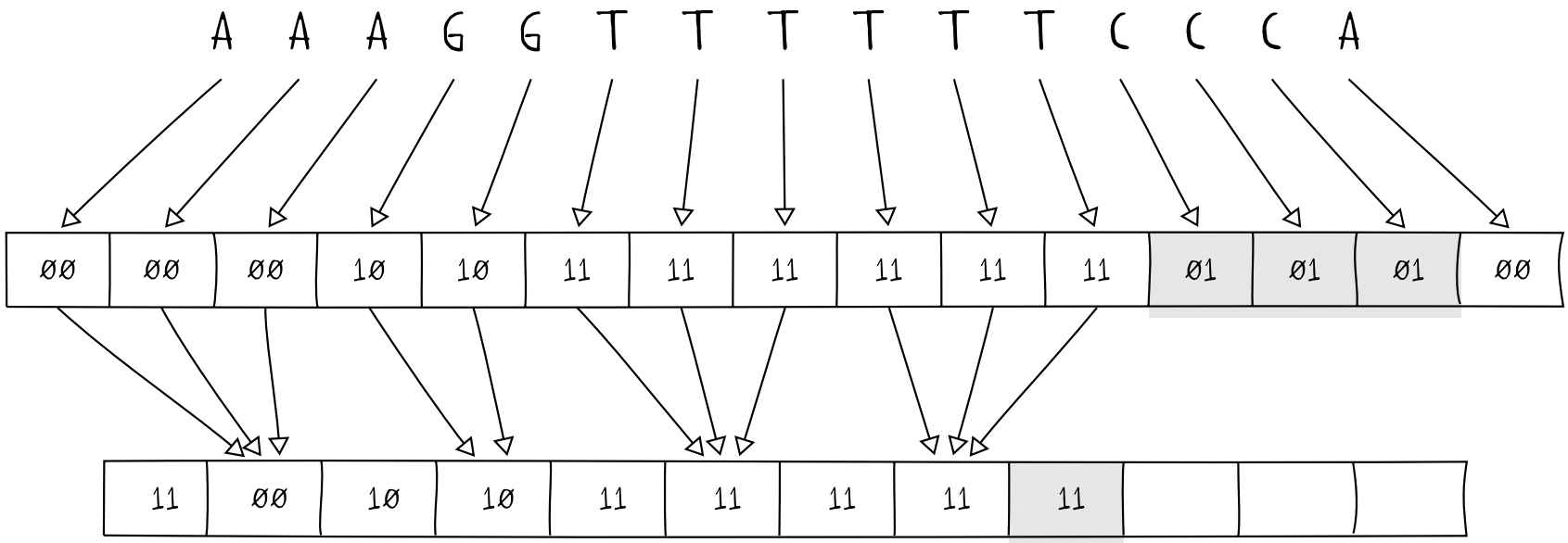
Compressão de dados

- *Run-Length Encoding* (RLE)
  - Esta técnica de compressão consiste em contabilizar a repetição de símbolos em uma sequência



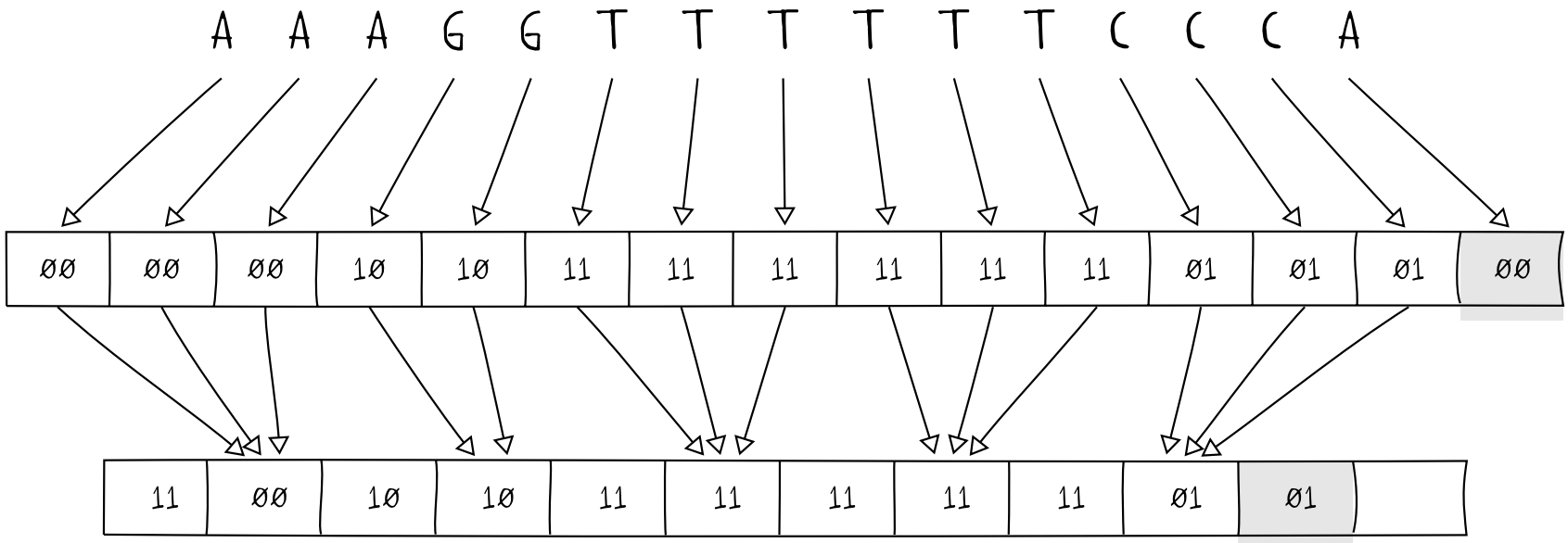
Compressão de dados

- *Run-Length Encoding* (RLE)
  - Esta técnica de compressão consiste em contabilizar a repetição de símbolos em uma sequência



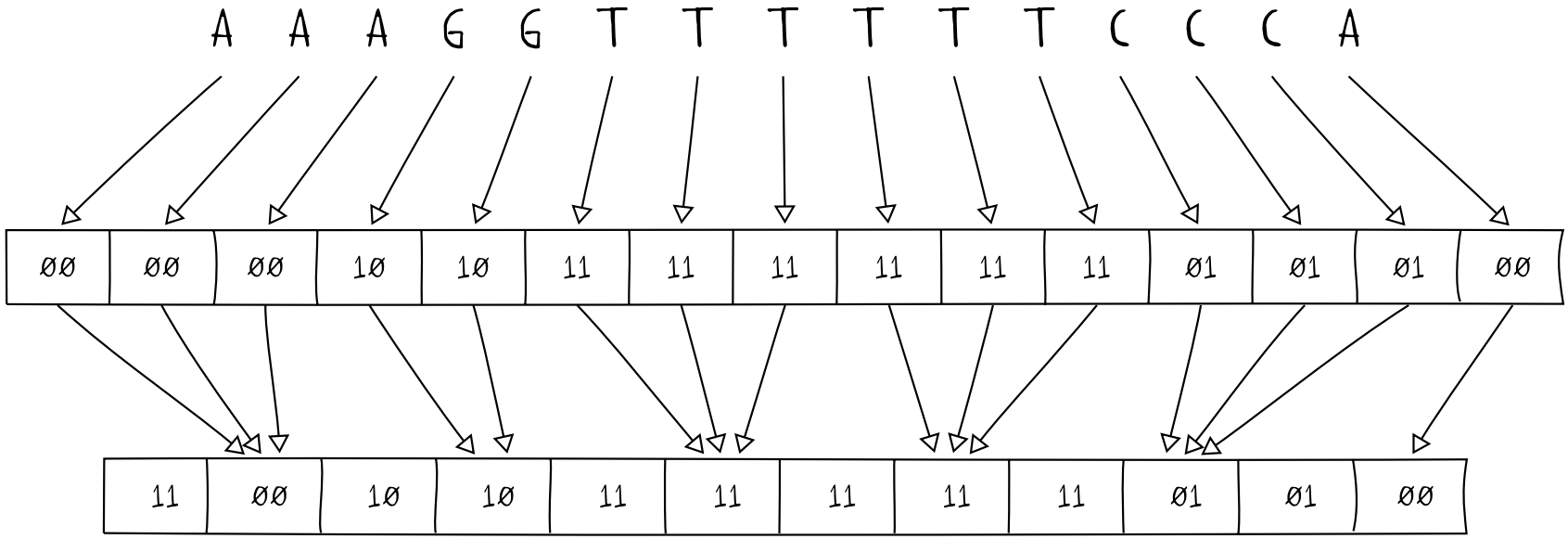
Compressão de dados

- *Run-Length Encoding* (RLE)
  - Esta técnica de compressão consiste em contabilizar a repetição de símbolos em uma sequência



Compressão de dados

- *Run-Length Encoding* (RLE)
  - Esta técnica de compressão consiste em contabilizar a repetição de símbolos em uma sequência



$$\textit{Taxa de compressão} = 100 \times \frac{24\ \textit{bits}}{30\ \textit{bits}} = 80\%$$

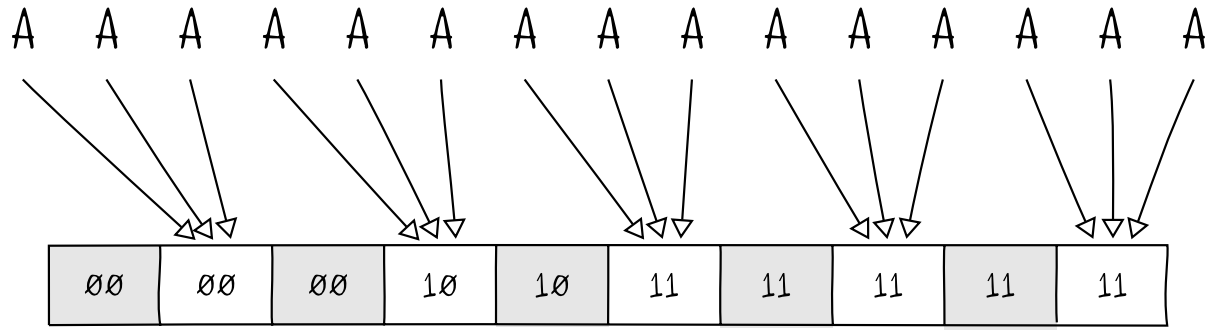
Compressão de dados

- *Run-Length Encoding* (RLE)
  - O ajuste da quantidade de bits do contador é dependente entrada e impacta diretamente a taxa de compressão

A A A A A A A A A A A A A A A

Compressão de dados

- *Run-Length Encoding* (RLE)
  - O ajuste da quantidade de bits do contador é dependente entrada e impacta diretamente a taxa de compressão

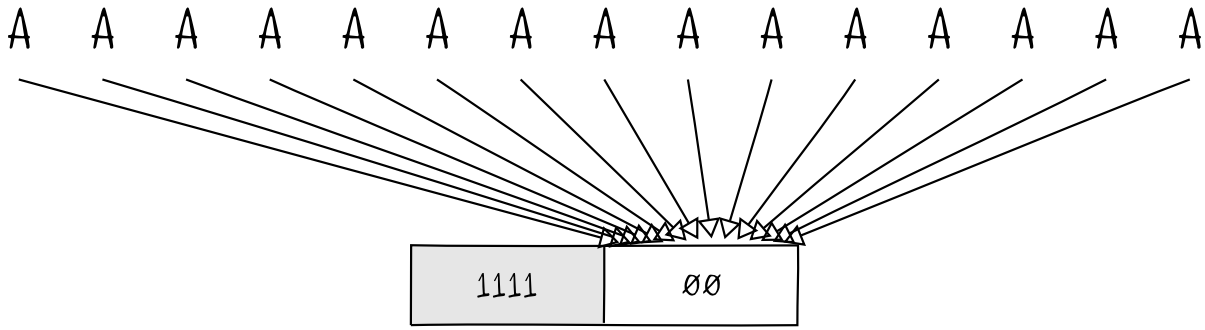


$$Taxa\ de\ compressão = 100 \times \frac{20\ bits}{30\ bits} \approx 67\%$$

$$\downarrow Bits\ do\ contador \leftrightarrow \downarrow Repetições \wedge \downarrow Espaço$$

Compressão de dados

- *Run-Length Encoding* (RLE)
  - O ajuste da quantidade de bits do contador é dependente entrada e impacta diretamente a taxa de compressão

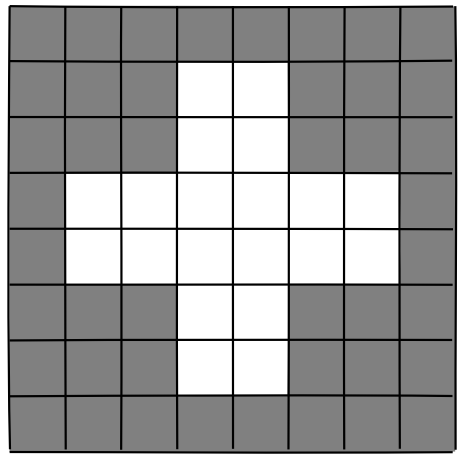


$$Taxa\ de\ compressão = 100 \times \frac{6\ bits}{30\ bits} = 20\%$$

$$\uparrow Bits\ do\ contador \leftrightarrow \uparrow Repetições \wedge \uparrow Espaço$$

Compressão de dados

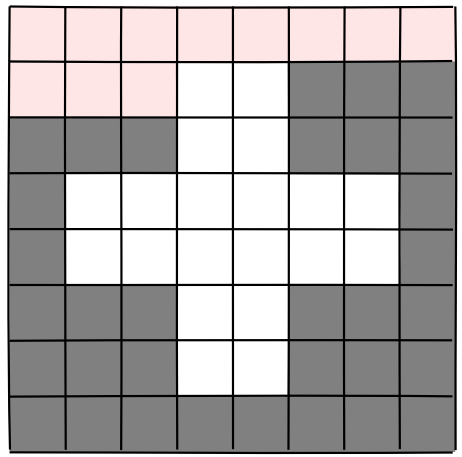
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits:

Compressão de dados

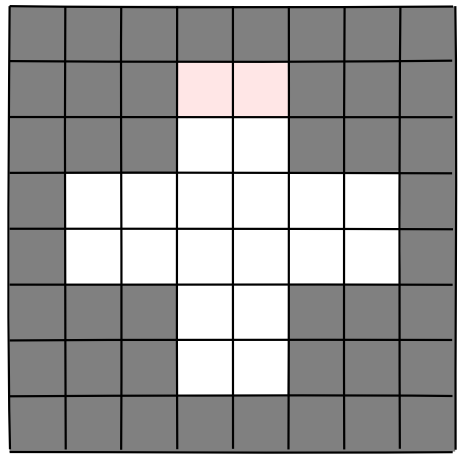
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*0

Compressão de dados

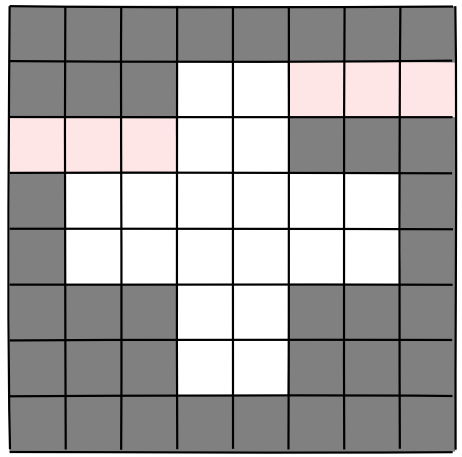
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*

Compressão de dados

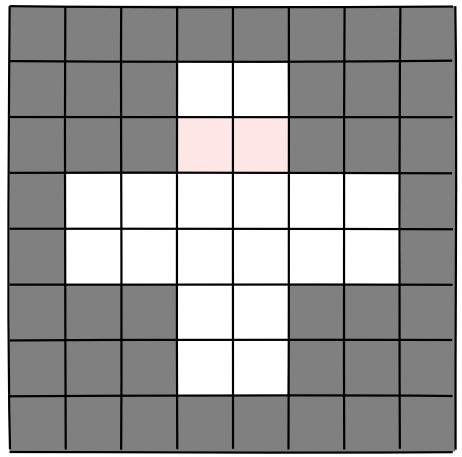
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*60

Compressão de dados

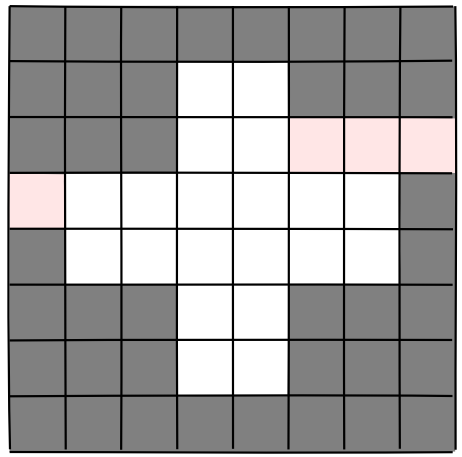
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*

Compressão de dados

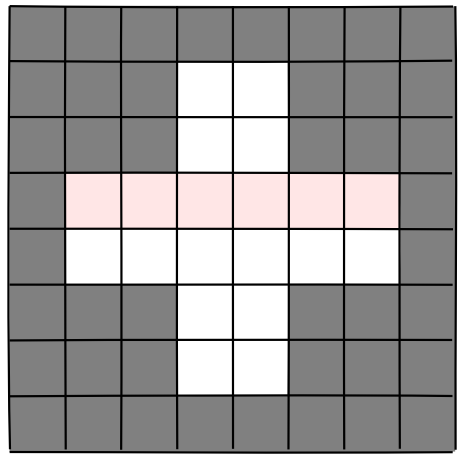
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*40

Compressão de dados

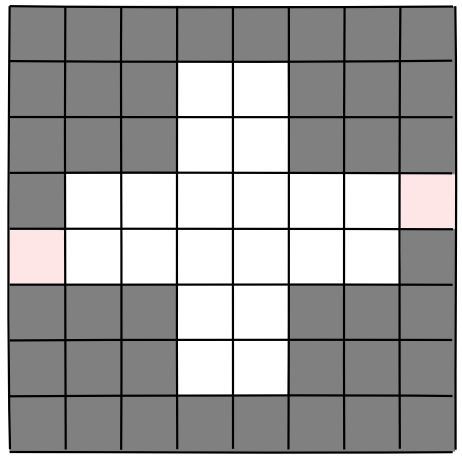
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*406*F*

Compressão de dados

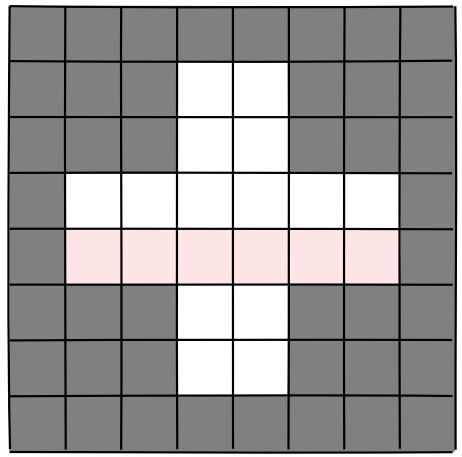
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*406*F*20

Compressão de dados

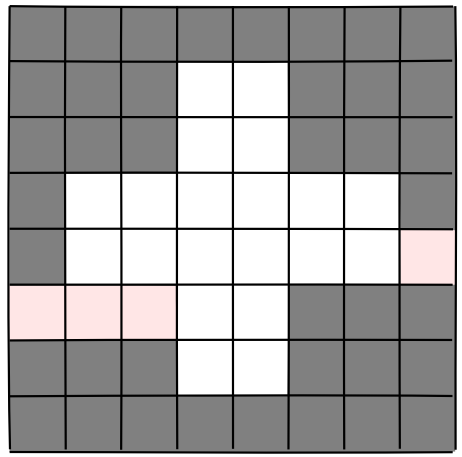
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*406*F*206*F*

Compressão de dados

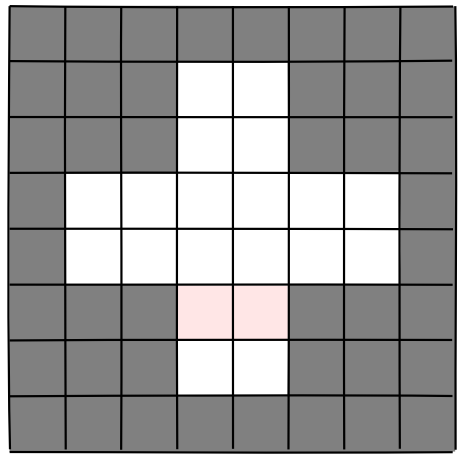
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*406*F*206*F*40

Compressão de dados

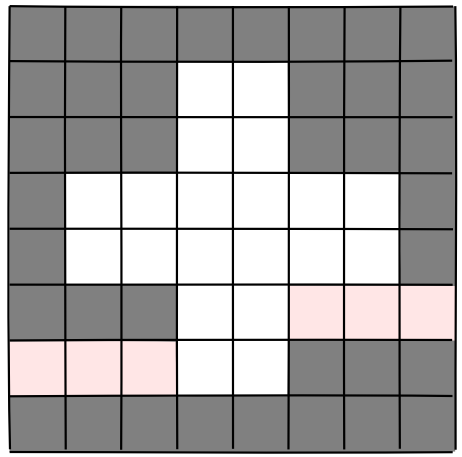
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*406*F*206*F*402*F*

Compressão de dados

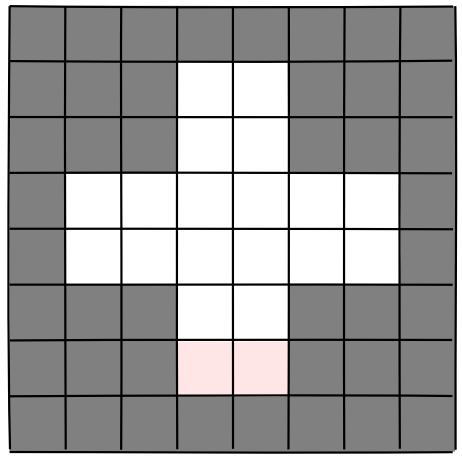
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*406*F*206*F*402*F*60

Compressão de dados

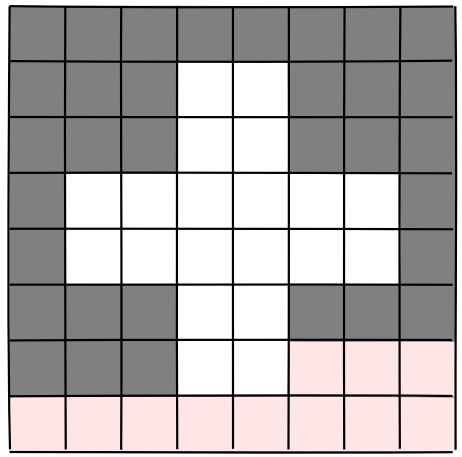
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*406*F*206*F*402*F*602*F*

Compressão de dados

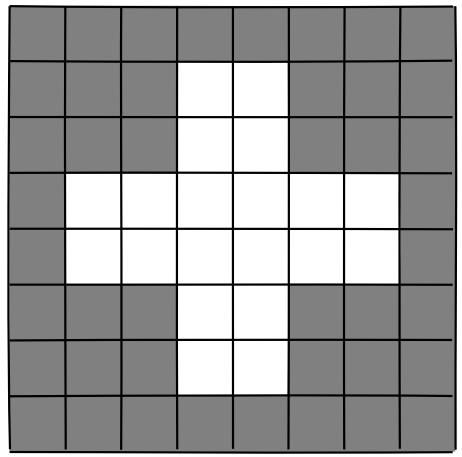
- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: *B*02*F*602*F*406*F*206*F*402*F*602*FB*0

Compressão de dados

- *Run-Length Encoding* (RLE)
  - Imagem de tamanho 8 x 8 pixels em escala de cinza (0 = preto, F = branco), com 4 bits por pixel



RLE de 4 bits: $B02F602F406F206F402F602FB0$

$$Taxa\ de\ compressão = 100 \times \frac{104\ bits}{256\ bits} \approx 41\%$$

Compressão de dados

- Codificação de Huffman
  - Foi criada em 1952 por David A. Huffman e consiste em utilizar uma quantidade variável de bits para codificar os símbolos, utilizando menos bits para os símbolos que possuem maior frequência

Compressão de dados

- Codificação de Huffman
  - Foi criada em 1952 por David A. Huffman e consiste em utilizar uma quantidade variável de bits para codificar os símbolos, utilizando menos bits para os símbolos que possuem maior frequência
  - Para eliminar a necessidade de delimitadores na codificação de tamanho variável, é construída uma árvore binária de prefixos (*trie*) para gerar códigos que não são prefixo de nenhum outro símbolo

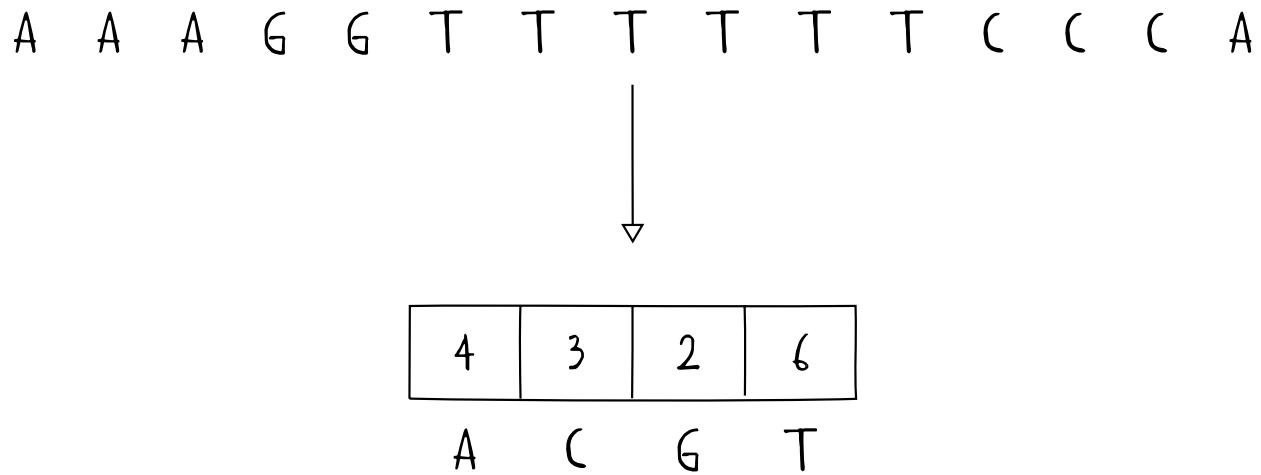
Compressão de dados

- Codificação de Huffman
  - A entrada é processada para contabilizar a frequência de ocorrência dos símbolos do alfabeto $\sum$

A A A G G T T T T T T C C C A

Compressão de dados

- Codificação de Huffman
  - A entrada é processada para contabilizar a frequência de ocorrência dos símbolos do alfabeto $\sum$



Espaço $\Theta(|\sum| + n)$ e tempo $\Theta(n)$

Compressão de dados

- Codificação de Huffman
  - A árvore de prefixos (*trie*) permite a criação de codificação binárias de tamanho mínimo e sem repetição de prefixos, desta forma eliminando a necessidade de delimitadores

```
1  // Padrão de tipos por tamanho
2  #include <stdint.h>
3  // Estrutura do nó
4  typedef struct no {
5      // Frequência
6      uint32_t F;
7      // Código do símbolo
8      char S;
9      // Nó direito
10     no* D;
11     // Nó esquerdo
12     no* E;
13 } no;
```

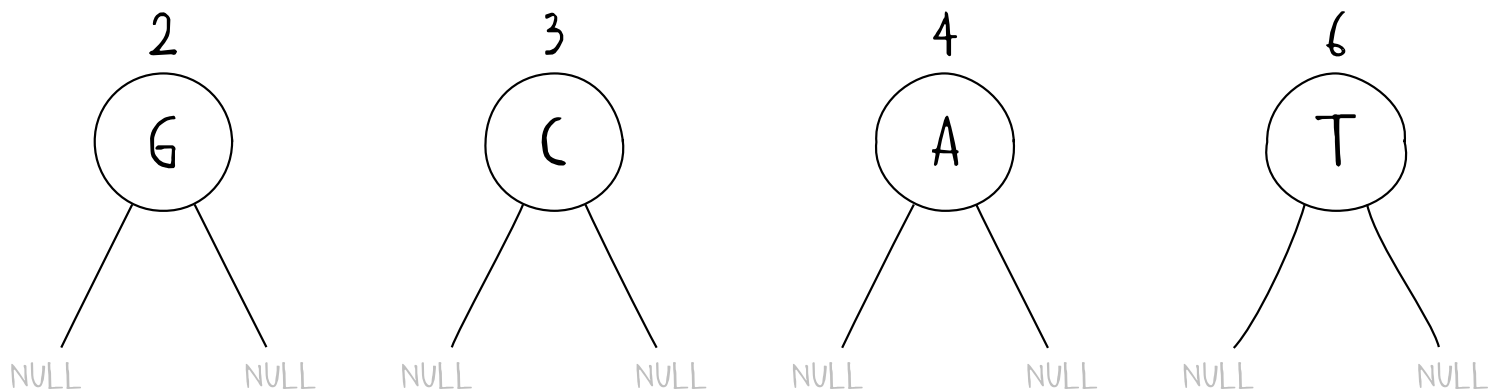
Compressão de dados

- Codificação de Huffman

```
14 // Construção da árvore de prefixos (trie)
15 no* construir_arvore(uint32_t H[], uint32_t n) {
16     // Criação de fila de prioridade mínima
17     fila_p_min* fpm = criar_fila_p_min();
18     // Inserindo símbolos não nulos na fila
19     for(uint32_t i = 0; i < n; i++)
20         if(H[i]) inserir(fpm, H[i], i, NULL, NULL);
21     // Combinação dos nós com menor frequência
22     while(tamanho(fpm) > 1) {
23         no* x = extrair_min(fpm);
24         no* y = extrair_min(fpm);
25         inserir(fpm, x.freq + y.freq, '\0', x, y);
26     }
27     // Retornando a raiz da árvore
28     return extrair_min(fpm);
29 }
```

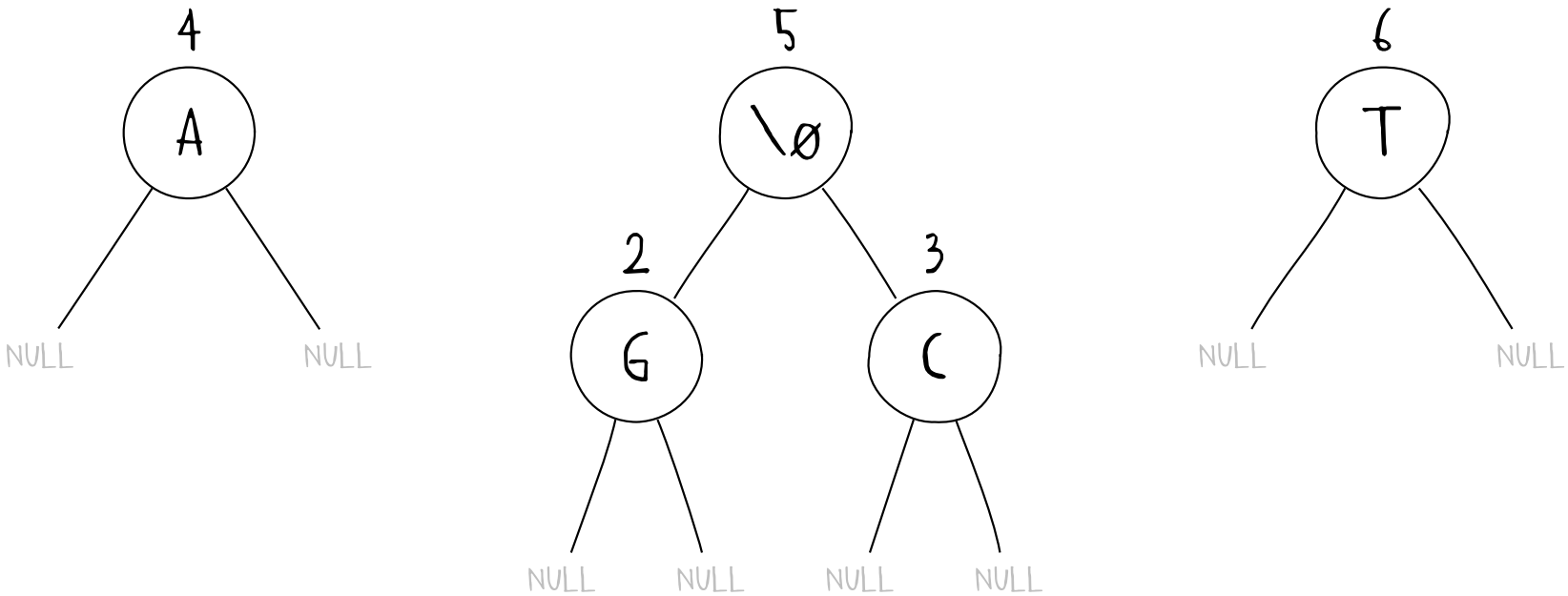
Compressão de dados

- Codificação de Huffman
  - Criação da árvore de prefixos: instanciação dos nós da árvore e criação da fila de prioridade mínima



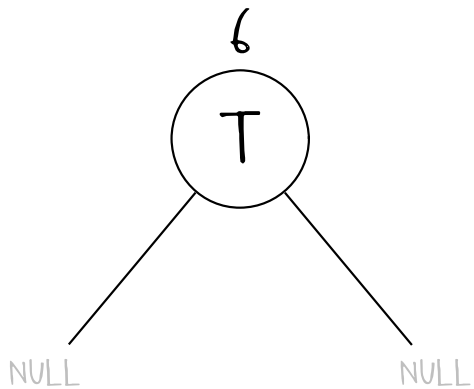
Compressão de dados

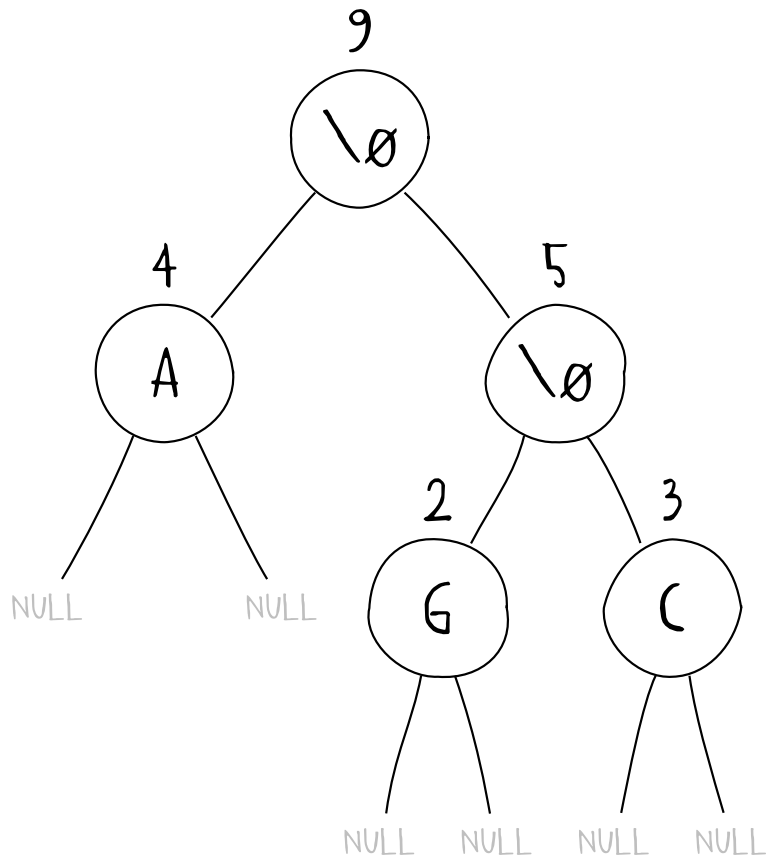
- Codificação de Huffman
  - É feita a remoção dos nós *G* e *C* da fila de prioridade mínima e a criação de um nó de símbolo nulo \0



Compressão de dados

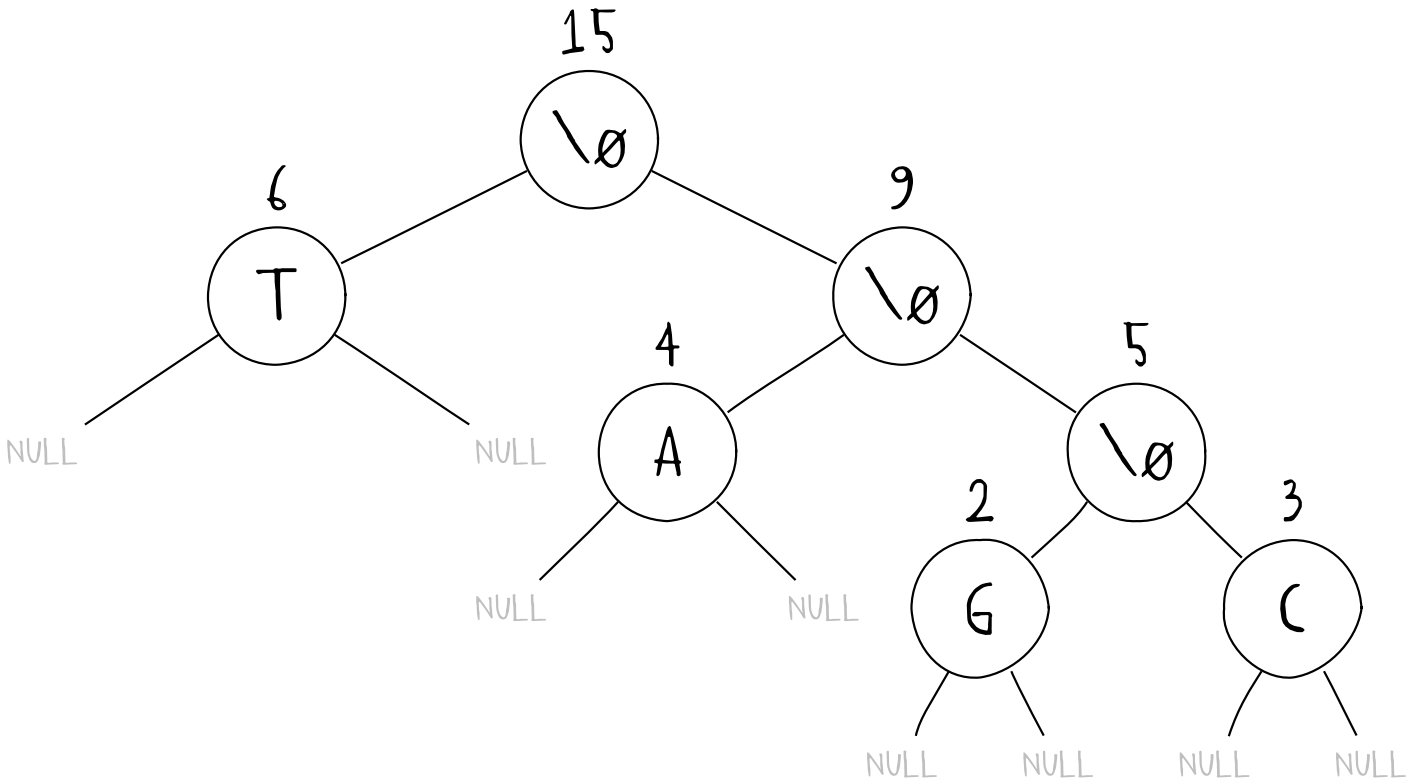
- Codificação de Huffman
  - É feita a remoção dos nós *A* e \0 da fila de prioridade mínima e a criação de um nó de símbolo nulo \0





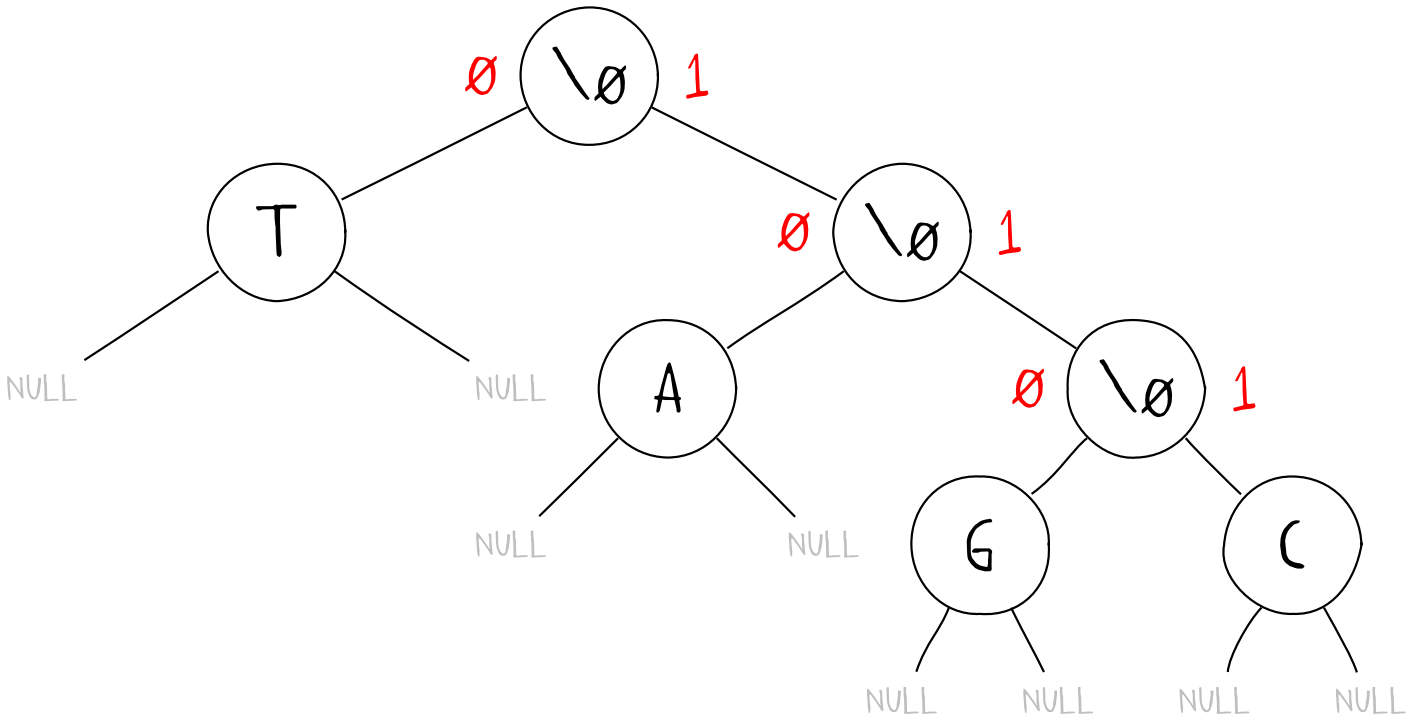
Compressão de dados

- Codificação de Huffman
  - É feita a remoção dos nós *T* e \0 da fila de prioridade mínima e a criação de um nó de símbolo nulo \0



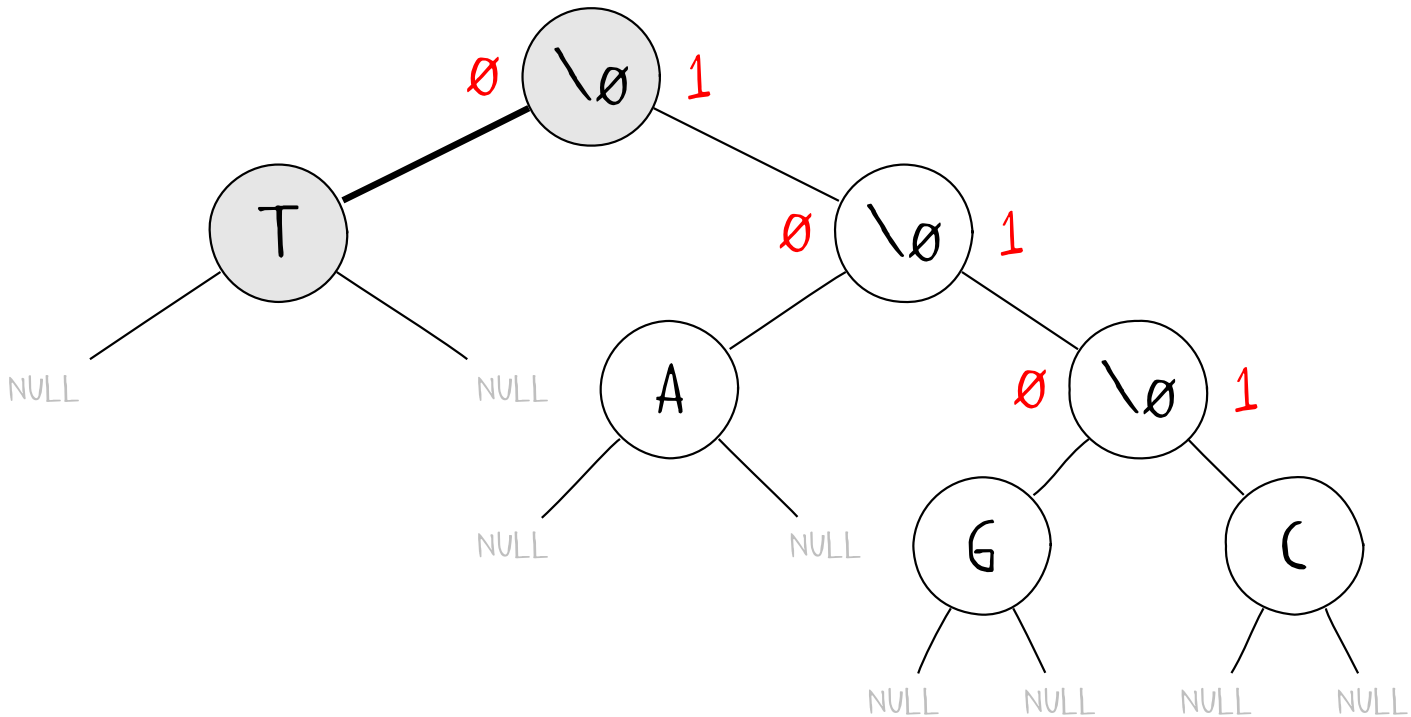
Compressão de dados

- Codificação de Huffman
  - A árvore de prefixos está construída e é convencionado que o encaminhamento pela esquerda e direita são respectivamente representados pelos bits 0 e 1



Compressão de dados

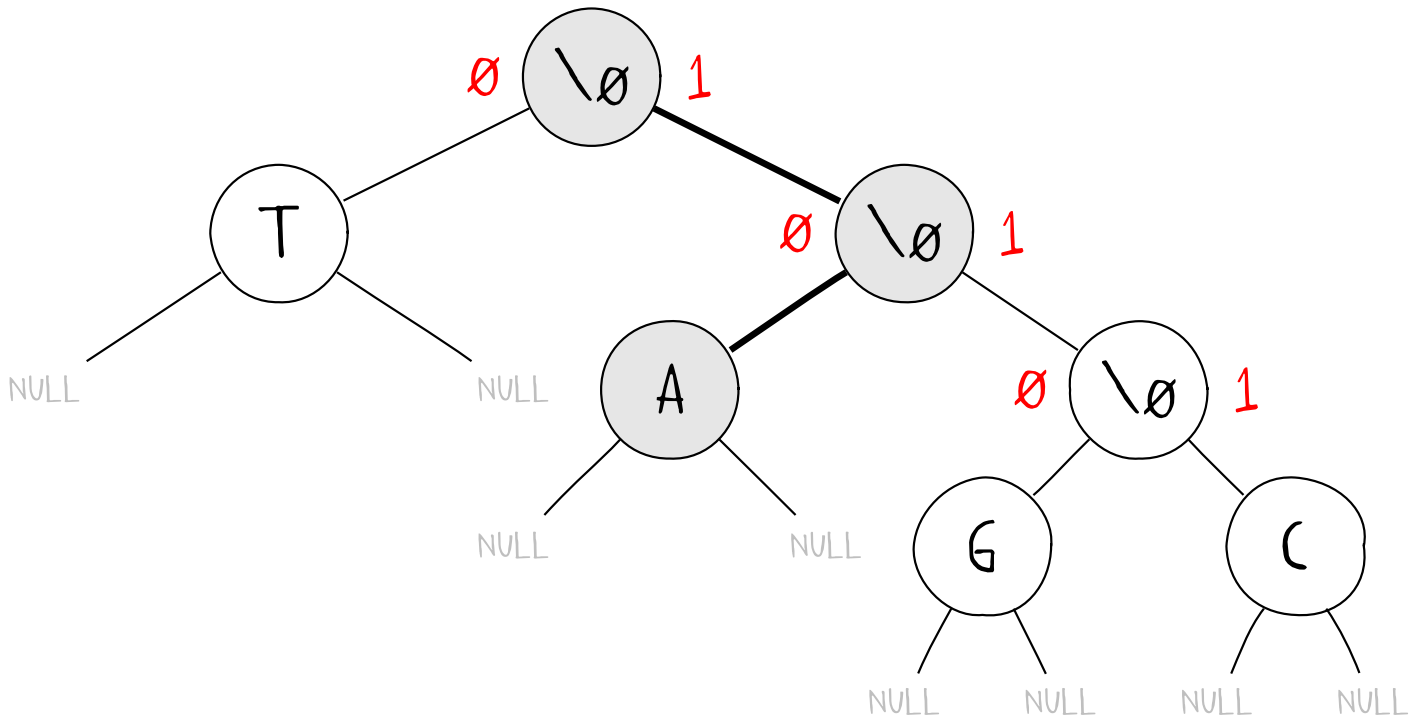
- Codificação de Huffman
  - Construção da tabela de codificação: os códigos para os símbolos são gerados através do encaminhamento na árvore de prefixos



$$T = 0$$

Compressão de dados

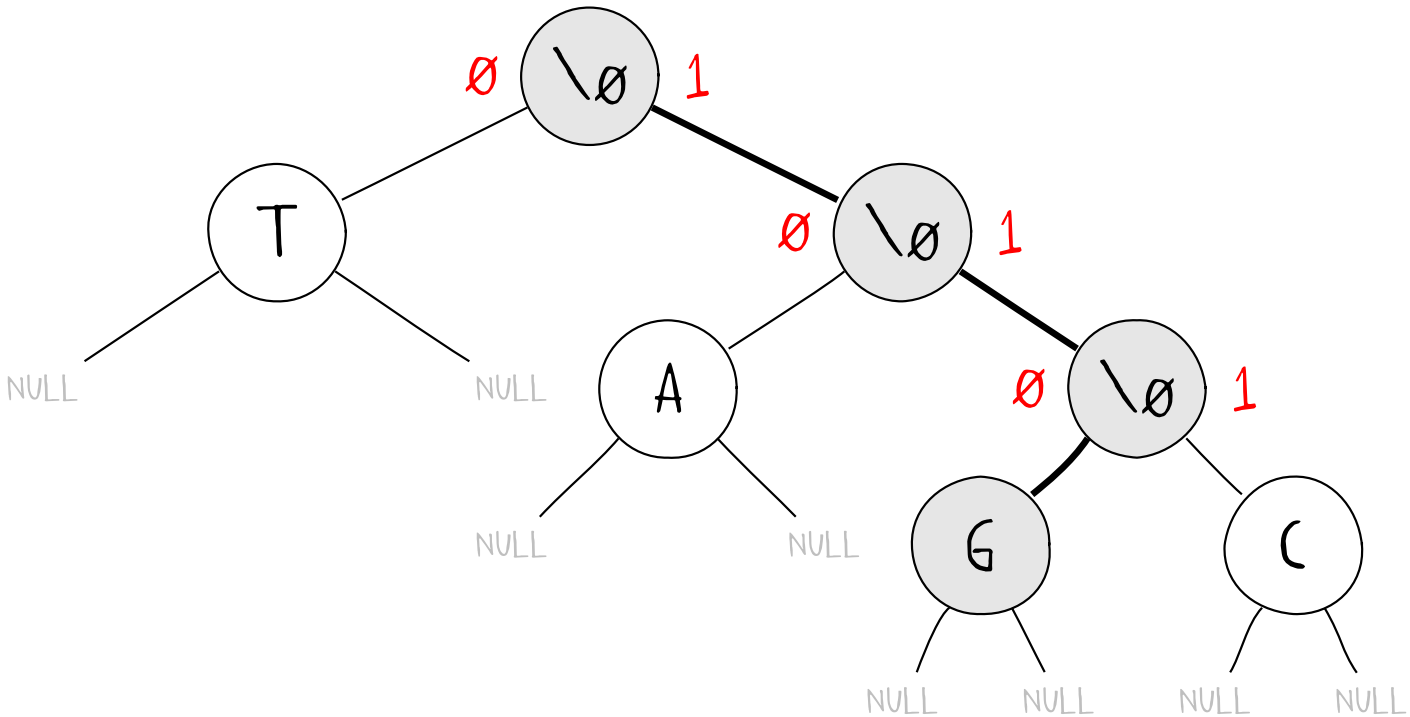
- Codificação de Huffman
  - Construção da tabela de codificação: os códigos para os símbolos são gerados através do encaminhamento na árvore de prefixos



$$A = 10$$

Compressão de dados

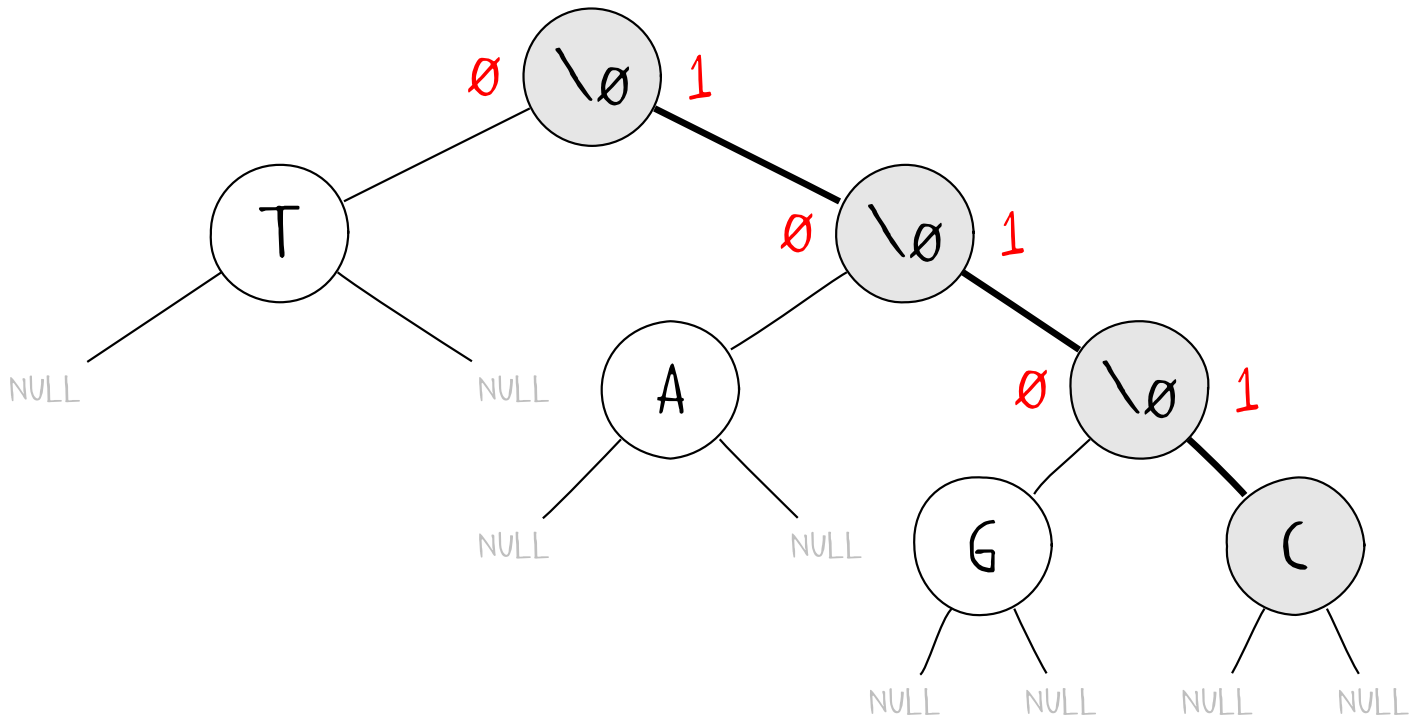
- Codificação de Huffman
  - Construção da tabela de codificação: os códigos para os símbolos são gerados através do encaminhamento na árvore de prefixos



$$G = 110$$

Compressão de dados

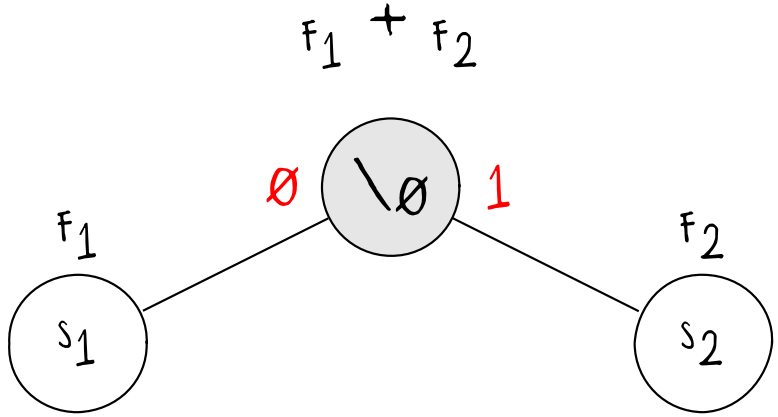
- Codificação de Huffman
  - Construção da tabela de codificação: os códigos para os símbolos são gerados através do encaminhamento na árvore de prefixos



$$C = 111$$

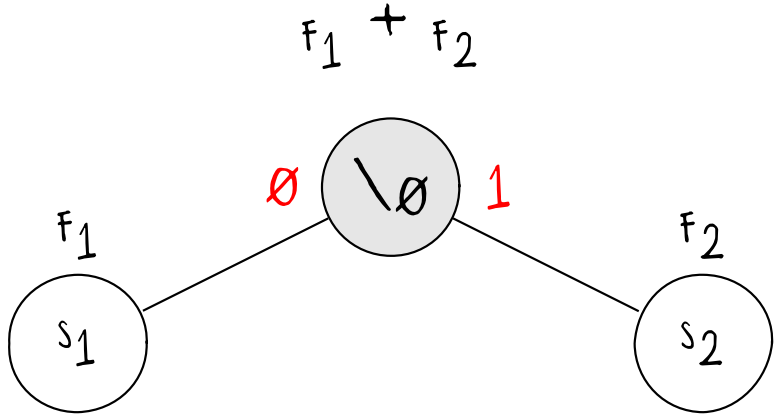
Compressão de dados

- Codificação de Huffman (prova da otimalidade)
  - Considere o conjunto de símbolos $\sum = \{s_1, s_2, \ldots, s_n\}$ com suas frequências denotadas por $f_1 \leq f_2 \leq \ldots \leq f_n$ para uma determinada sequência de entrada



Compressão de dados

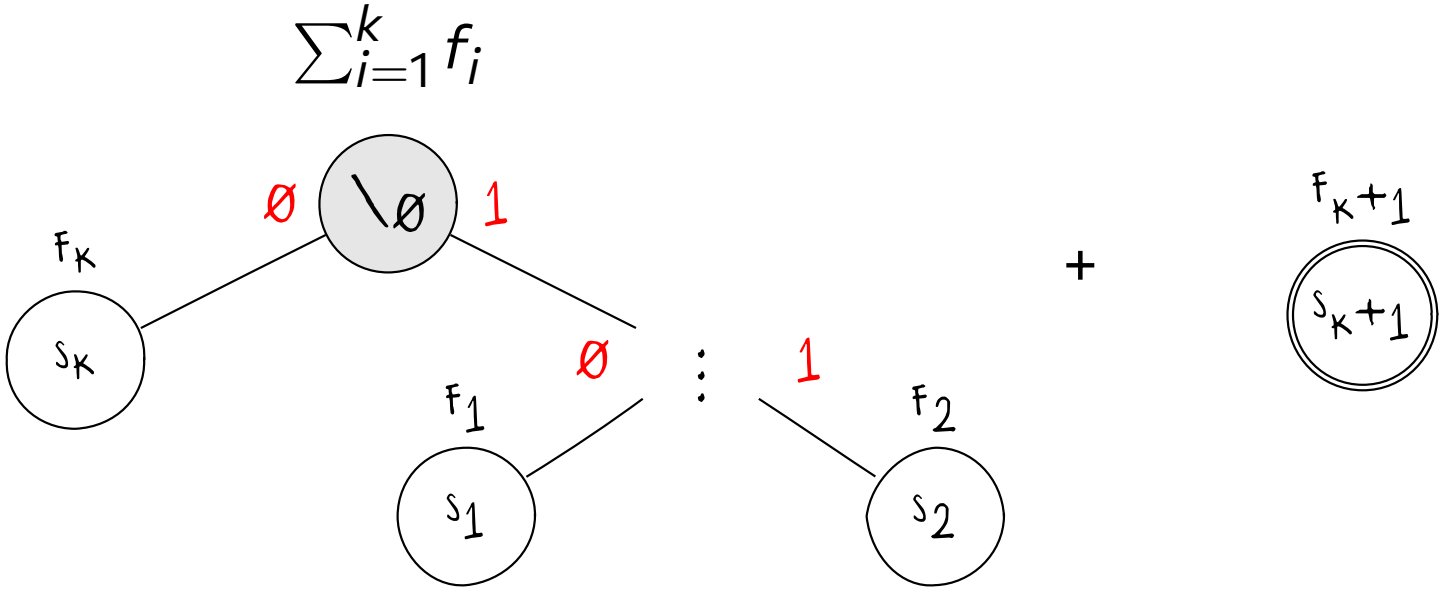
- Codificação de Huffman (prova da otimalidade)
  - Considere o conjunto de símbolos $\sum = \{s_1, s_2, \ldots, s_n\}$ com suas frequências denotadas por $f_1 \leq f_2 \leq \ldots \leq f_n$ para uma determinada sequência de entrada



Caso base: $N = 1$ ou $N = 2$ (1 bit)

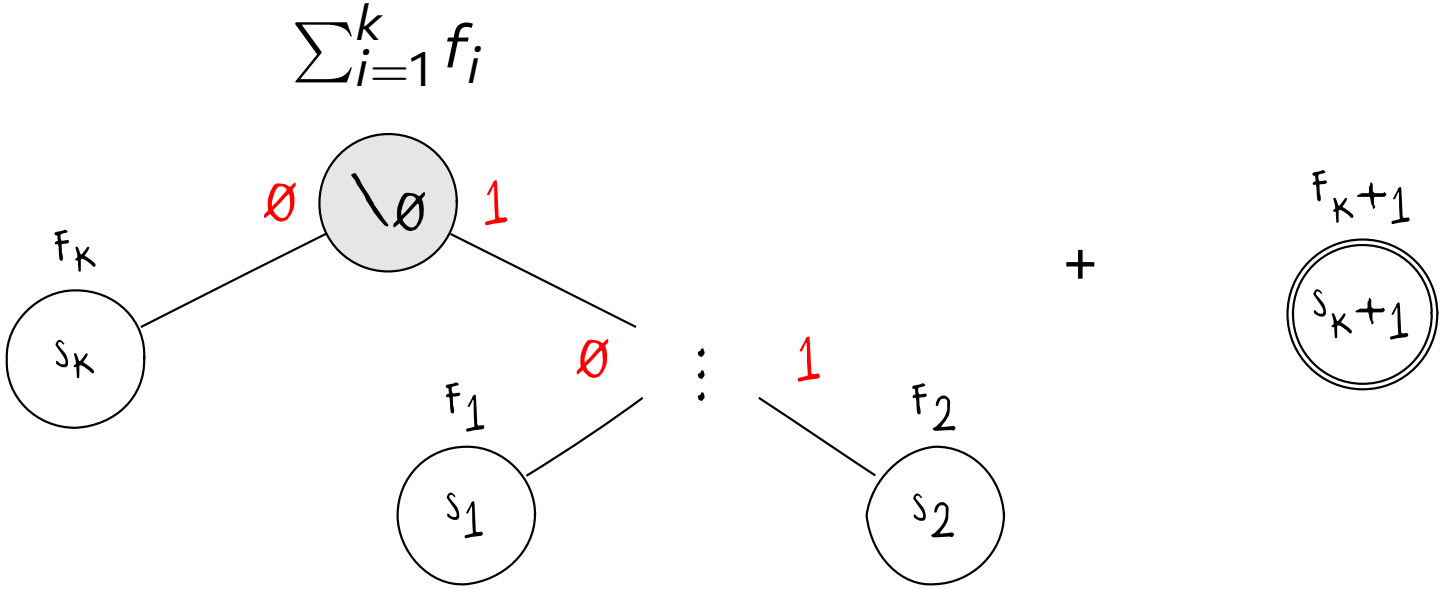
Compressão de dados

- Codificação de Huffman (prova da otimalidade)
  - Considere o conjunto de símbolos $\sum = \{s_1, s_2, \ldots, s_n\}$ com suas frequências denotadas por $f_1 \leq f_2 \leq \ldots \leq f_n$ para uma determinada sequência de entrada



Compressão de dados

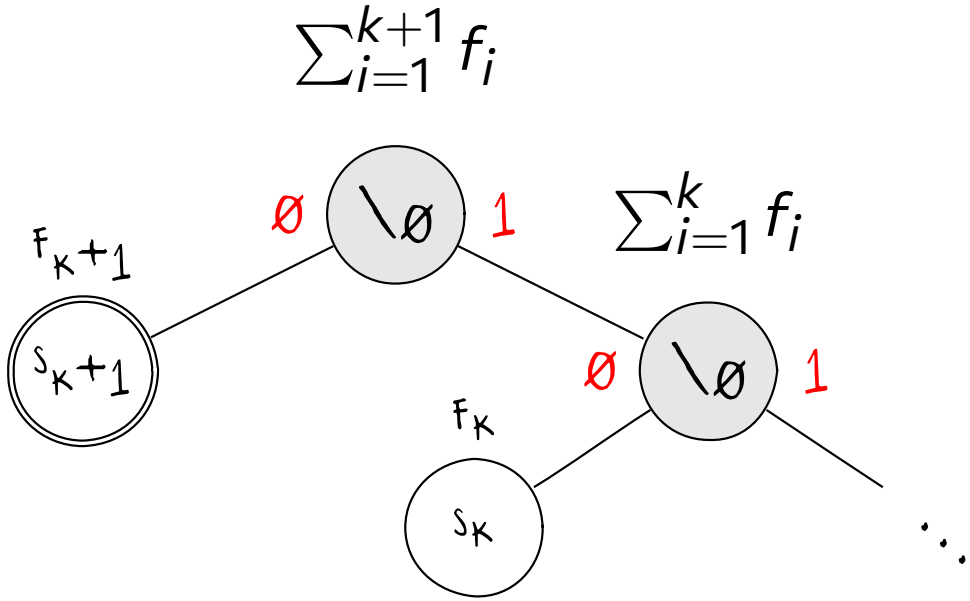
- Codificação de Huffman (prova da otimalidade)
  - Considere o conjunto de símbolos $\sum = \{s_1, s_2, \ldots, s_n\}$ com suas frequências denotadas por $f_1 \leq f_2 \leq \ldots \leq f_n$ para uma determinada sequência de entrada



Hipótese indutiva: $N = k$ (m bits)

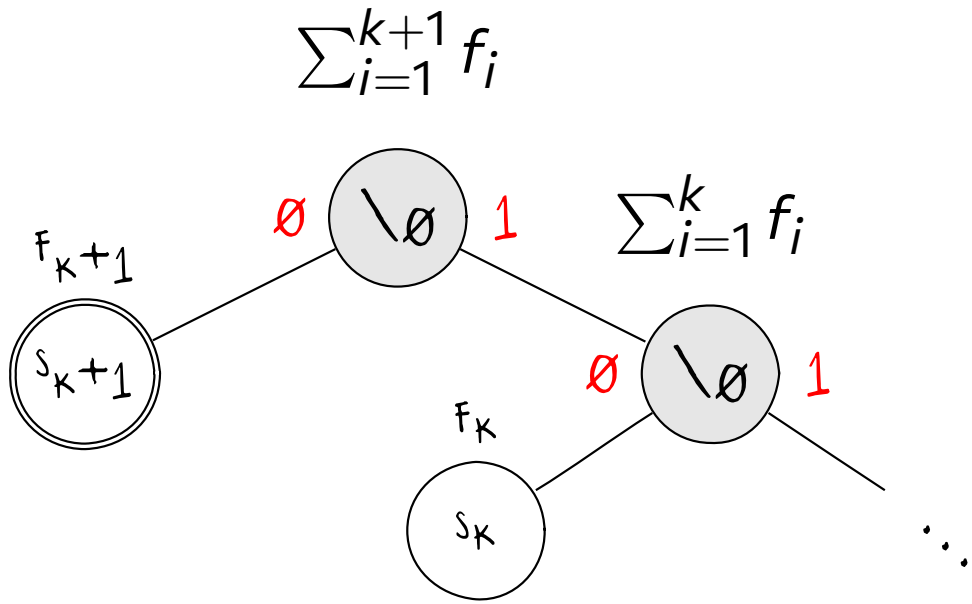
Compressão de dados

- Codificação de Huffman (prova da otimalidade)
  - Considere o conjunto de símbolos $\sum = \{s_1, s_2, \ldots, s_n\}$ com suas frequências denotadas por $f_1 \leq f_2 \leq \ldots \leq f_n$ para uma determinada sequência de entrada



Compressão de dados

- Codificação de Huffman (prova da otimalidade)
  - Considere o conjunto de símbolos $\sum = \{s_1, s_2, \ldots, s_n\}$ com suas frequências denotadas por $f_1 \leq f_2 \leq \ldots \leq f_n$ para uma determinada sequência de entrada

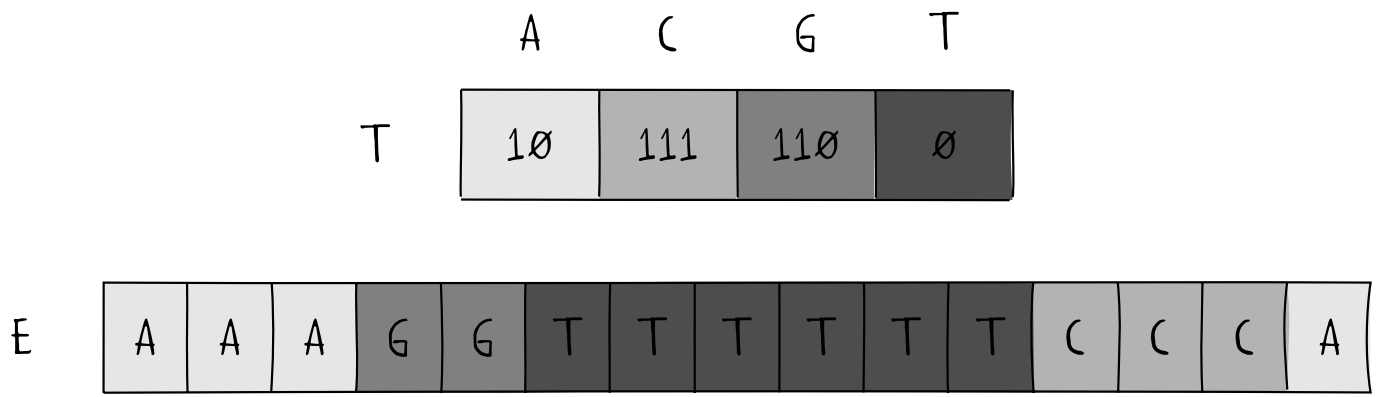


Tese: $N = k + 1\,(m + 1\,\text{bits})$

Compressão de dados

- Codificação de Huffman

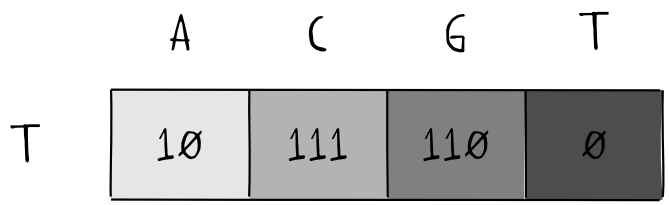
```
30 // Procedimento de compactação dos dados
31 void compactar(char* C, char* E, char* T) {
32     // Anexando codificação na saída compactada C
33     for(uint32_t i = 0; i < strlen(E); i++)
34         anexar(C, T[E[i]]);
35 }
```

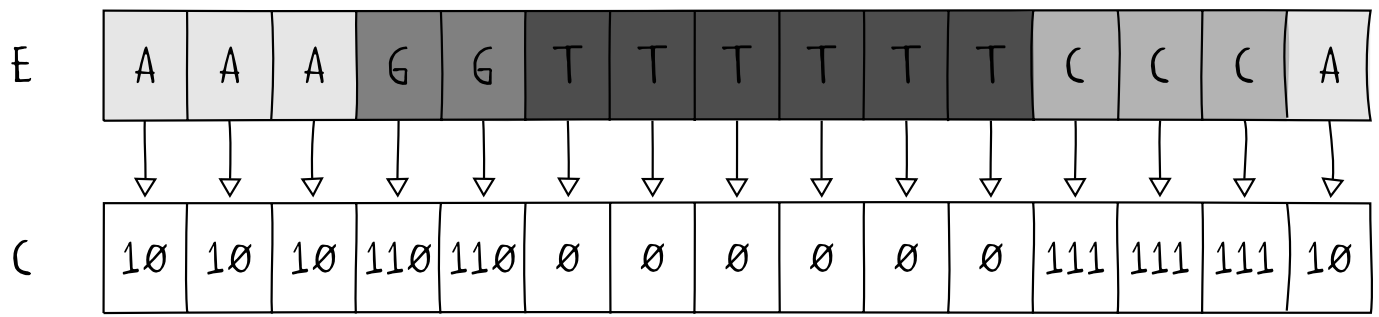


Compressão de dados

- Codificação de Huffman

```
30 // Procedimento de compactação dos dados
31 void compactar(char* C, char* E, char* T) {
32     // Anexando codificação na saída compactada C
33     for(uint32_t i = 0; i < strlen(E); i++)
34         anexar(C, T[E[i]]);
35 }
```





$$Taxa\ de\ compressão = 100 \times \frac{29\ bits}{120\ bits} = 24\%$$

Compressão de dados

- Características da codificação de Huffman
  - ✓ Espaço $\Theta\left(|\sum| + n\right)$, tempo $\Omega\left(n\right)$ e $O\left(n \log n\right)$

Compressão de dados

- Características da codificação de Huffman
  - ✓ Espaço $\Theta\left(|\sum| + n\right)$, tempo $\Omega\left(n\right)$ e $O\left(n \log n\right)$
  - ✓ Eficiência em diversos domínios de aplicação

Compressão de dados

- Características da codificação de Huffman
  - ✓ Espaço $\Theta\left(|\sum| + n\right)$, tempo $\Omega\left(n\right)$ e $O\left(n \log n\right)$
  - ✓ Eficiência em diversos domínios de aplicação
  - ✓ Não utiliza delimitadores na codificação dos dados

Compressão de dados

- Características da codificação de Huffman
  - ✓ Espaço $\Theta\left(|\sum| + n\right)$, tempo $\Omega\left(n\right)$ e $O\left(n \log n\right)$
  - ✓ Eficiência em diversos domínios de aplicação
  - ✓ Não utiliza delimitadores na codificação dos dados
  - ✓ Quanto maior a redudância ou a frequência dos símbolos dos dados, melhor será a taxa de compressão obtida

Compressão de dados

- Características da codificação de Huffman
  ✓ Espaço $\Theta\left(|\sum| + n\right)$, tempo $\Omega\left(n\right)$ e $O\left(n \log n\right)$
  ✓ Eficiência em diversos domínios de aplicação
  ✓ Não utiliza delimitadores na codificação dos dados
  ✓ Quanto maior a redudância ou a frequência dos símbolos dos dados, melhor será a taxa de compressão obtida
  ✗ O processo de descompactação pode ser mais complexo, devido à codificação dos dados com tamanho variável

Compressão de dados

- Codificação Lempel-Ziv-Welch (LZW)
  - Foi desenvolvido no início da década de 1980 por Abraham Lempel, Jacob Ziv e Terry Welch

Compressão de dados

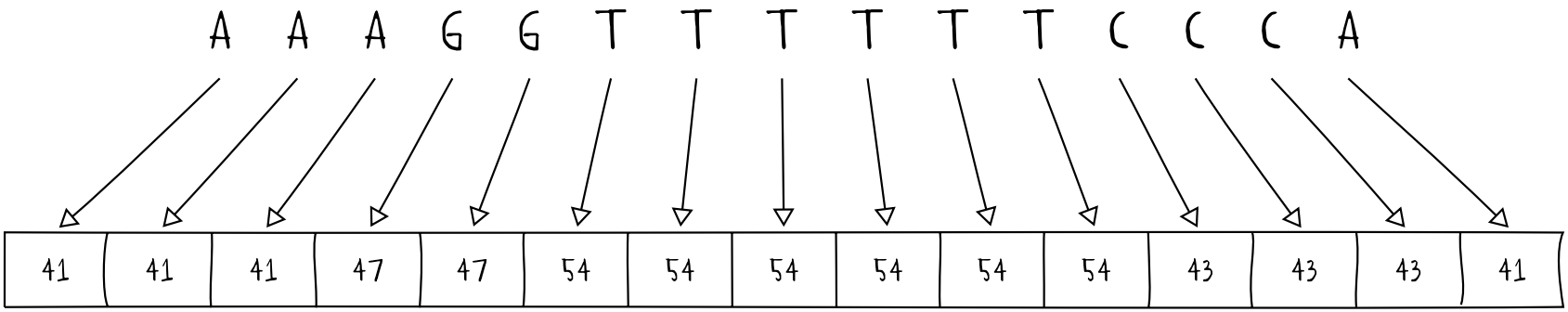
- Codificação Lempel-Ziv-Welch (LZW)
  - Foi desenvolvido no início da década de 1980 por Abraham Lempel, Jacob Ziv e Terry Welch
  - Ao invés de utilizar codificações variáveis para os símbolos, utiliza uma codificação de tamanho fixo para padrões de tamanho variáveis da entrada

Compressão de dados

- Codificação Lempel-Ziv-Welch (LZW)
  - Foi desenvolvido no início da década de 1980 por Abraham Lempel, Jacob Ziv e Terry Welch
  - Ao invés de utilizar codificações variáveis para os símbolos, utiliza uma codificação de tamanho fixo para padrões de tamanho variáveis da entrada
  - Não demanda o uso de delimitadores, pois os códigos possuem tamanho fixo, portanto sua tabela de códigos não precisa ser codificada

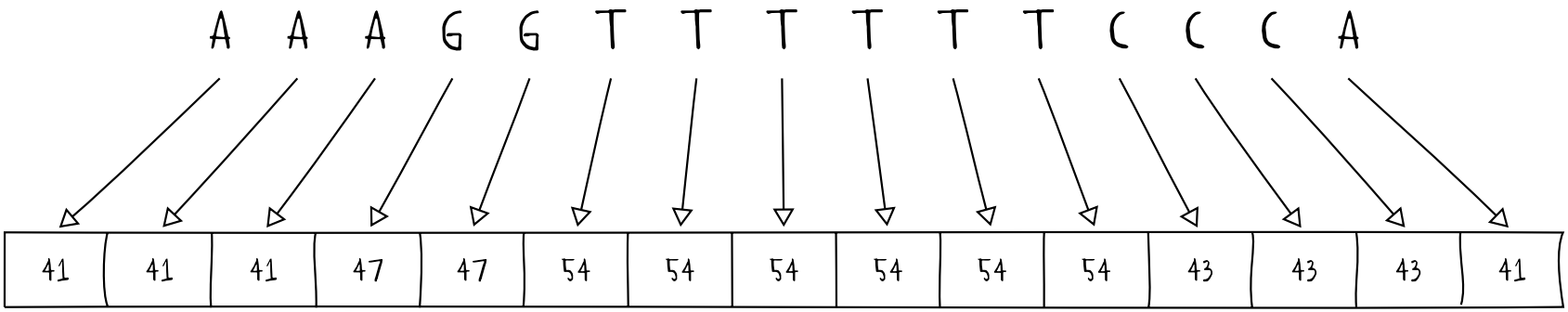
Compressão de dados

- Codificação Lempel-Ziv-Welch (LZW)
  - Os símbolos de entrada são representados em código ASCII de 7 bits (sem os códigos estendidos) em formato hexadecimal, permitindo uma codificação com 8 bits



Compressão de dados

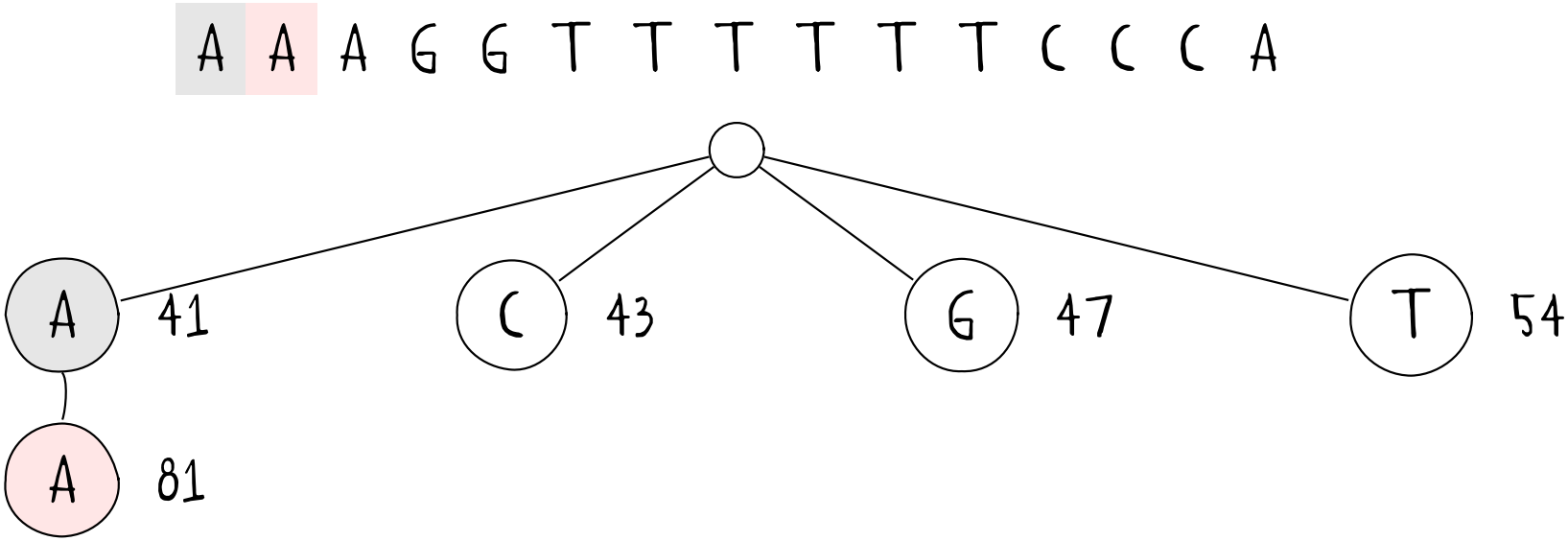
- Codificação Lempel-Ziv-Welch (LZW)
  - Os símbolos de entrada são representados em código ASCII de 7 bits (sem os códigos estendidos) em formato hexadecimal, permitindo uma codificação com 8 bits



Para implementações de propósito geral são utilizados símbolos de entrada com 8 bits e codificação de 12 bits para atender conjuntos de dados bem maiores

Compressão de dados

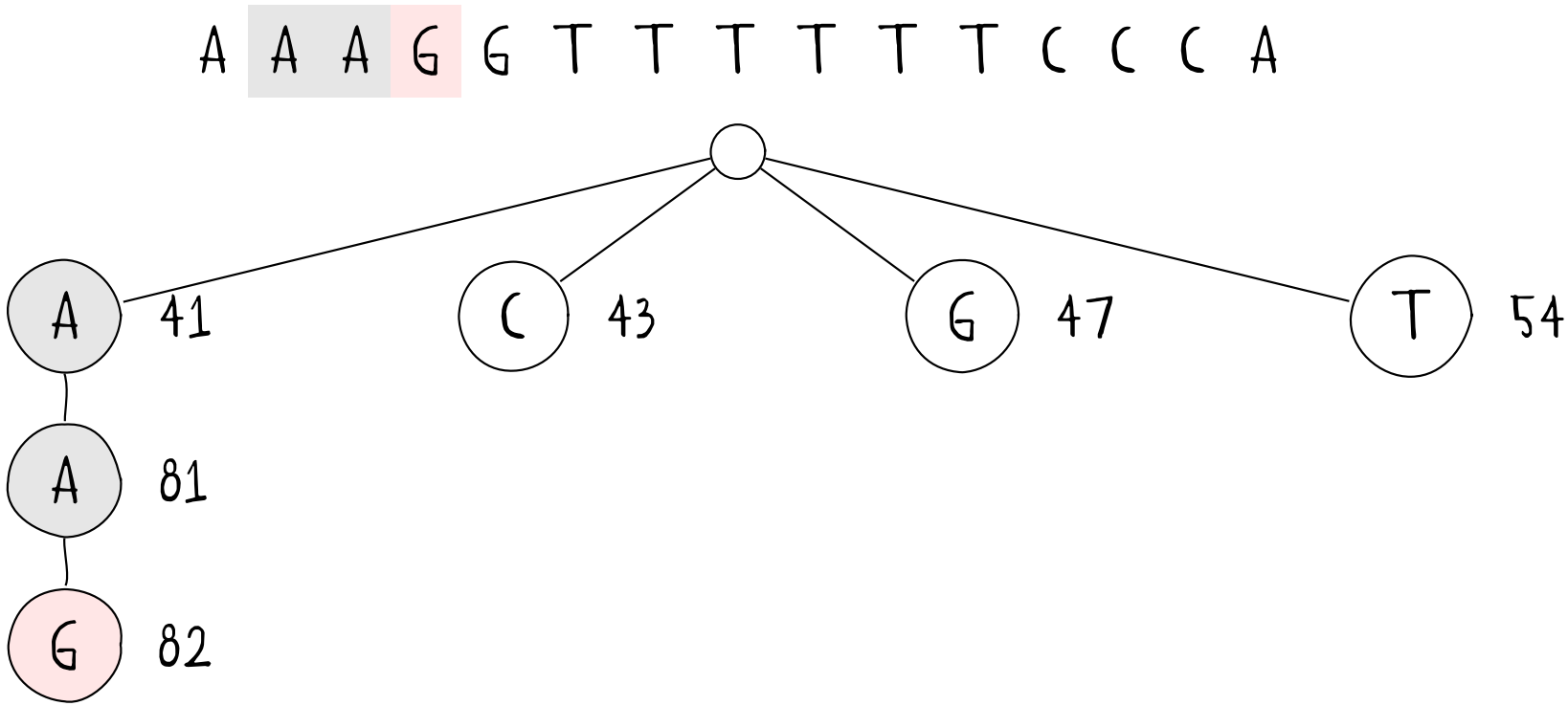
- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 41

Compressão de dados

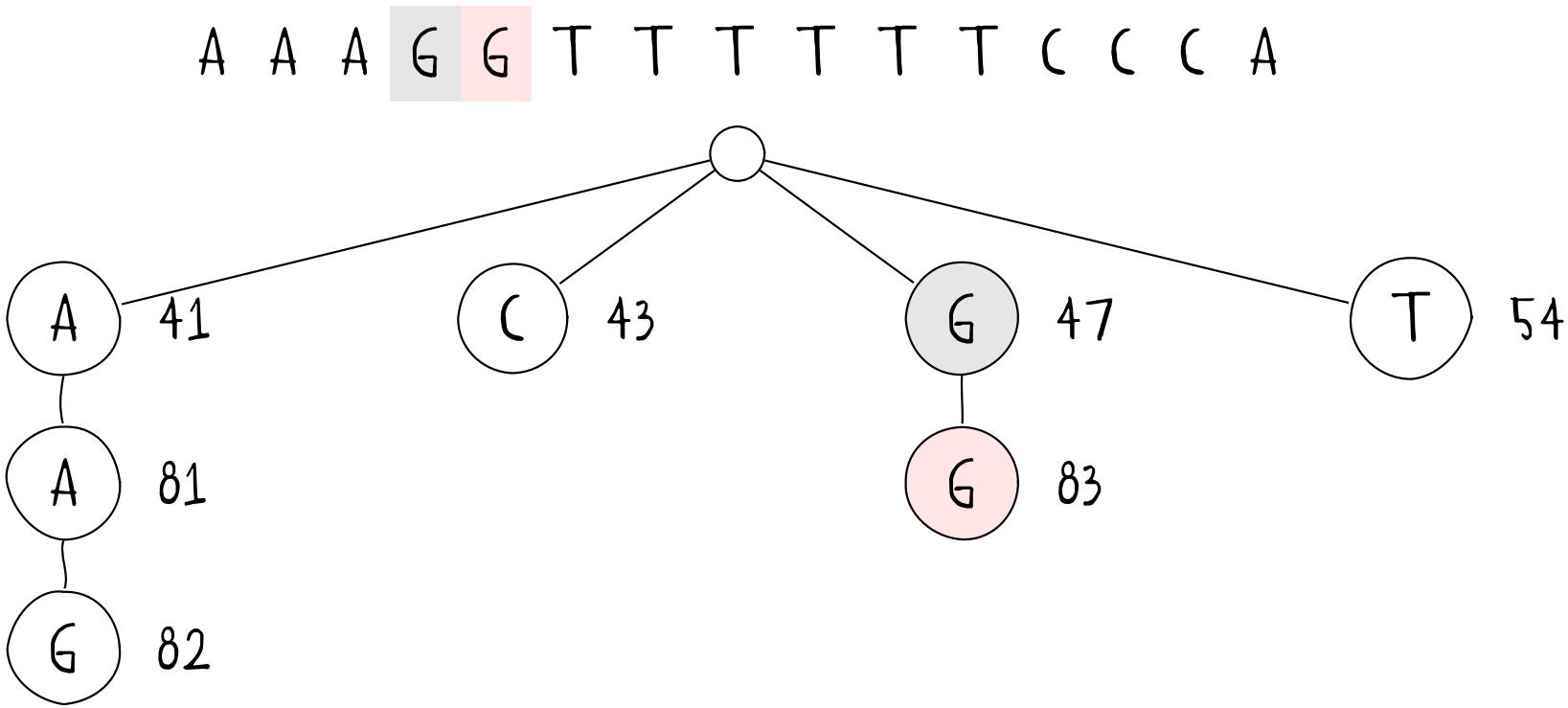
- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 4181

Compressão de dados

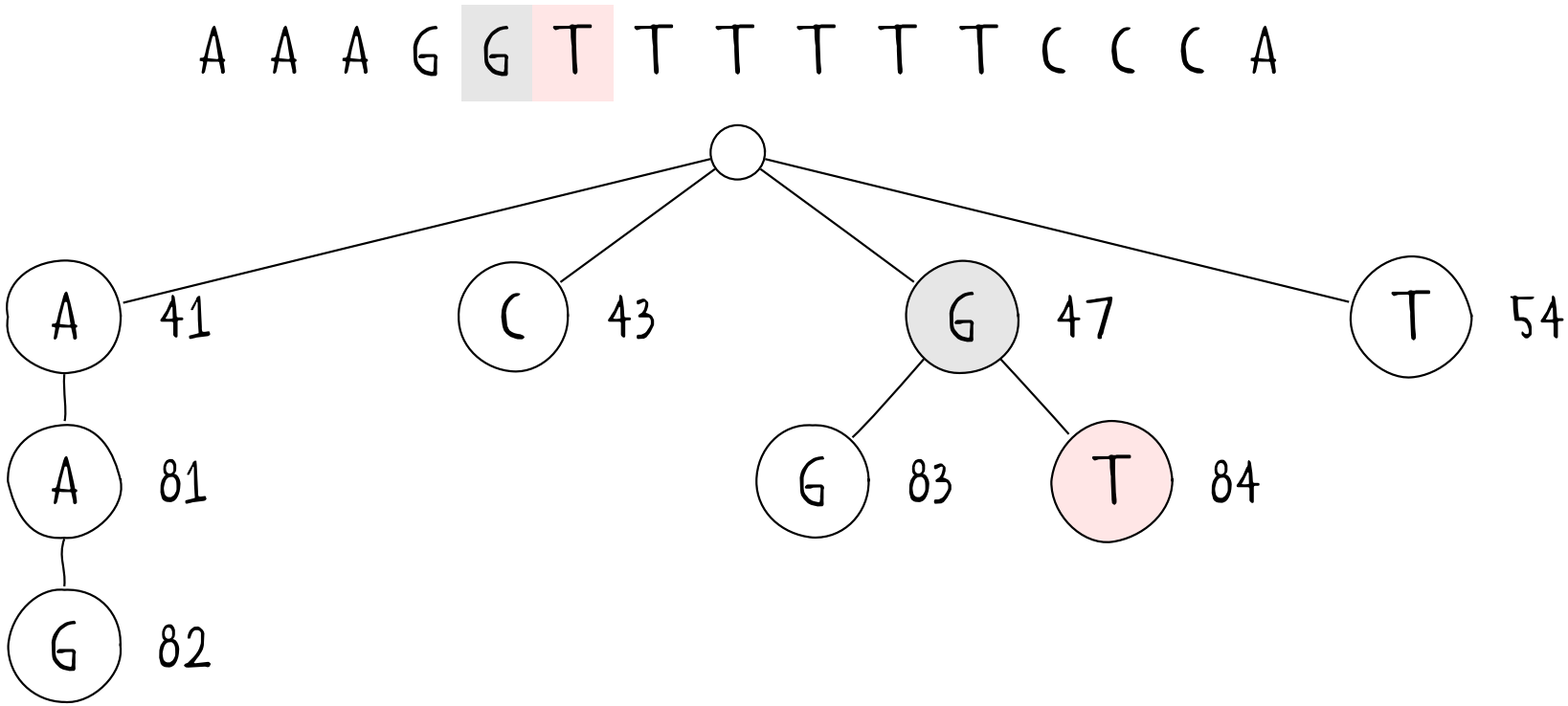
- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 418147

Compressão de dados

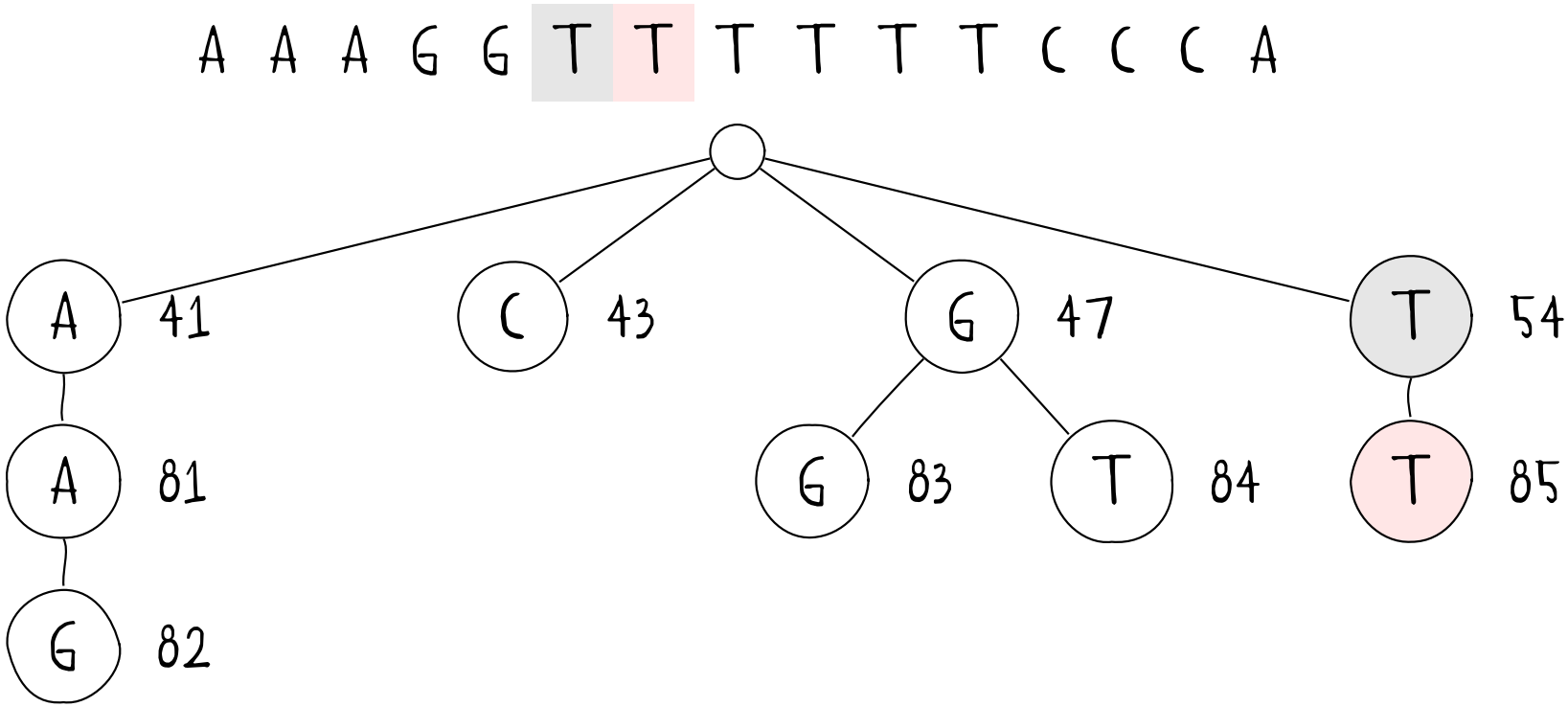
- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 41814747

Compressão de dados

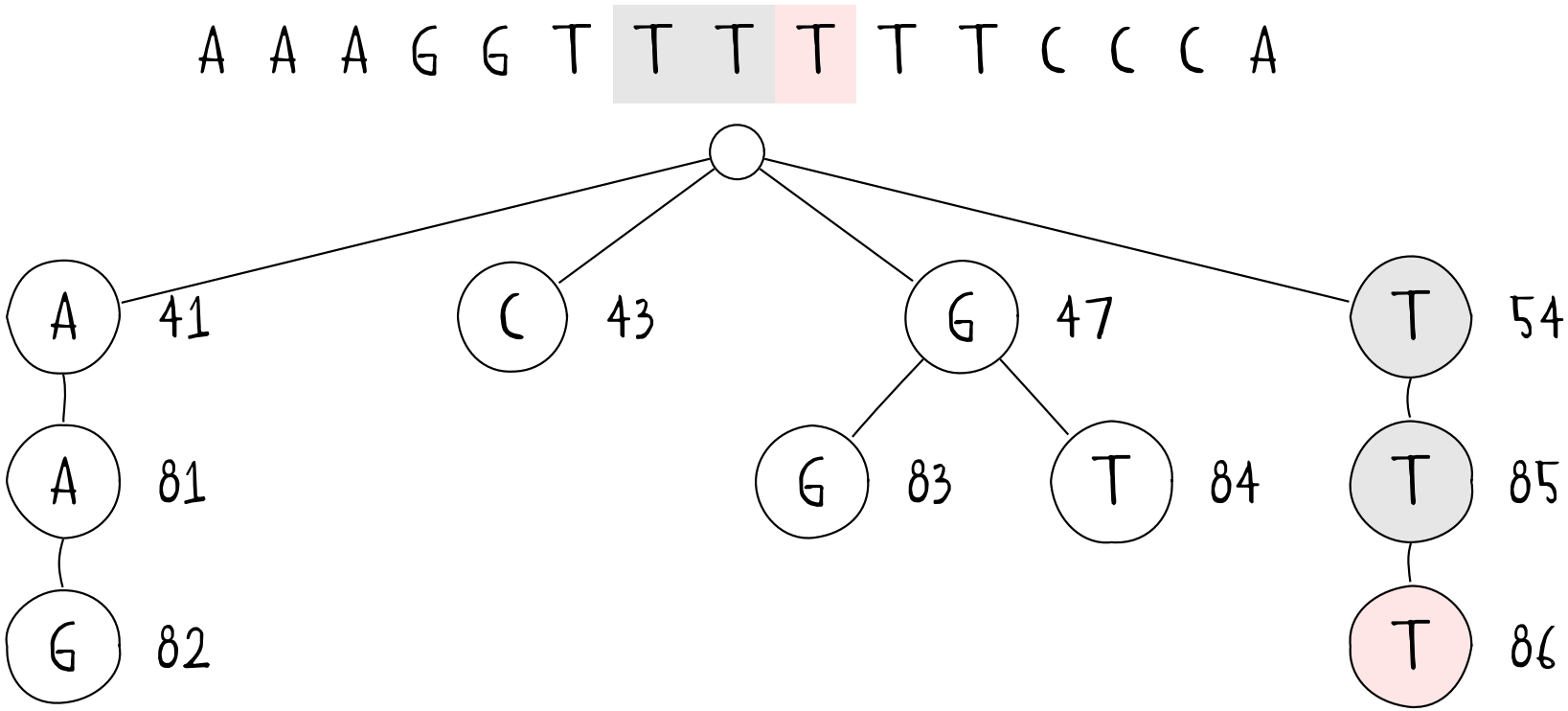
- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 4181474754

Compressão de dados

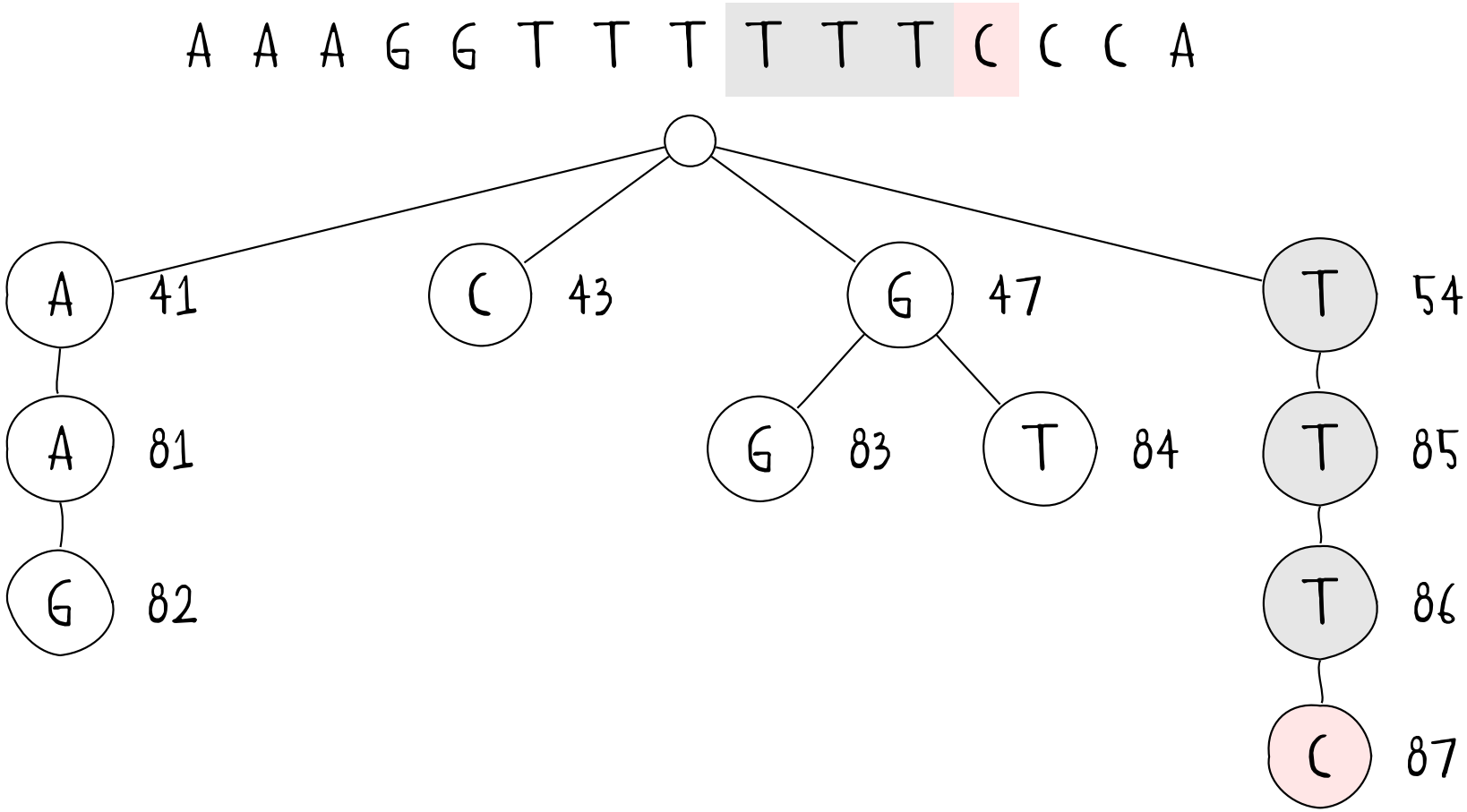
- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 418147475485

Compressão de dados

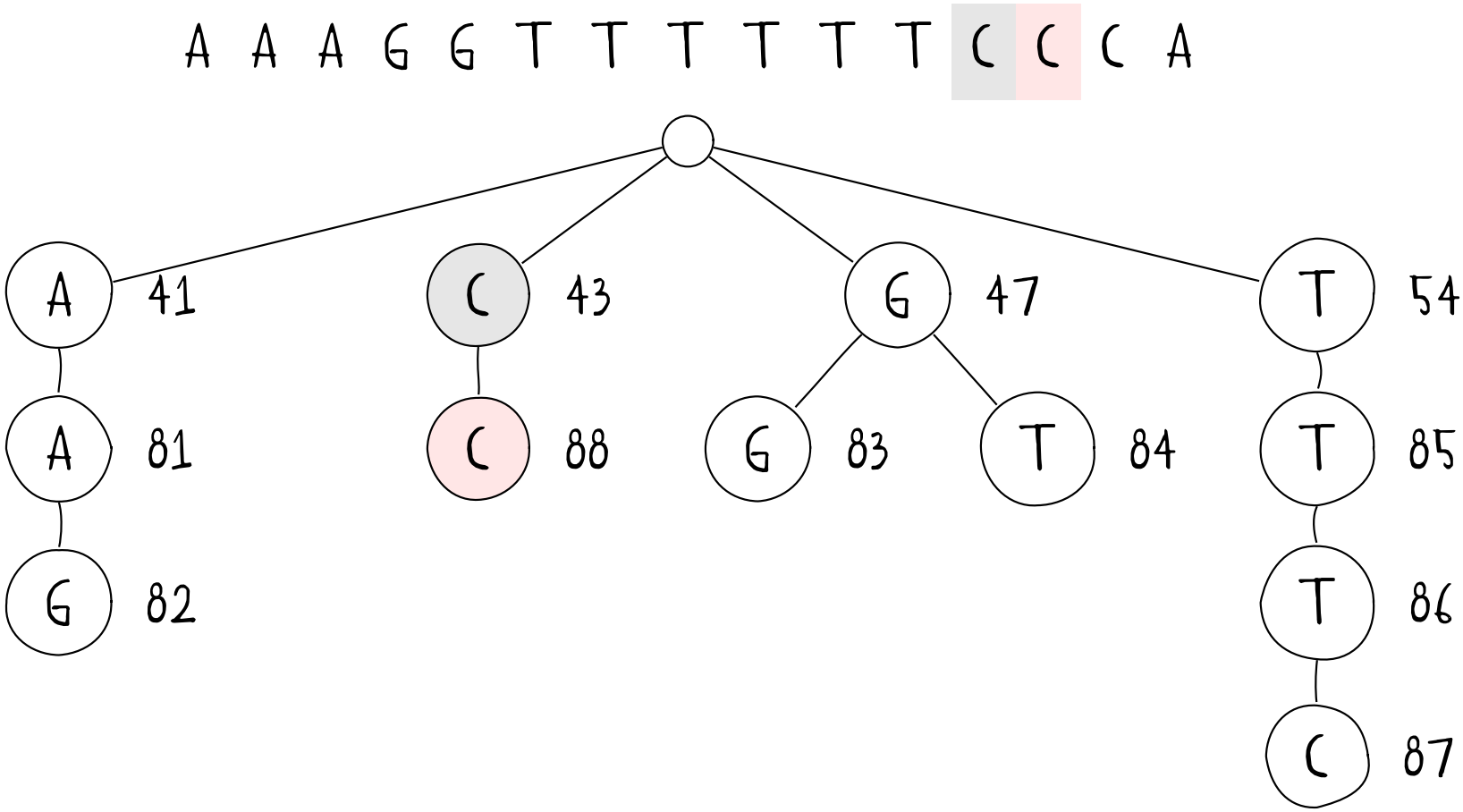
- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 41814747548586

Compressão de dados

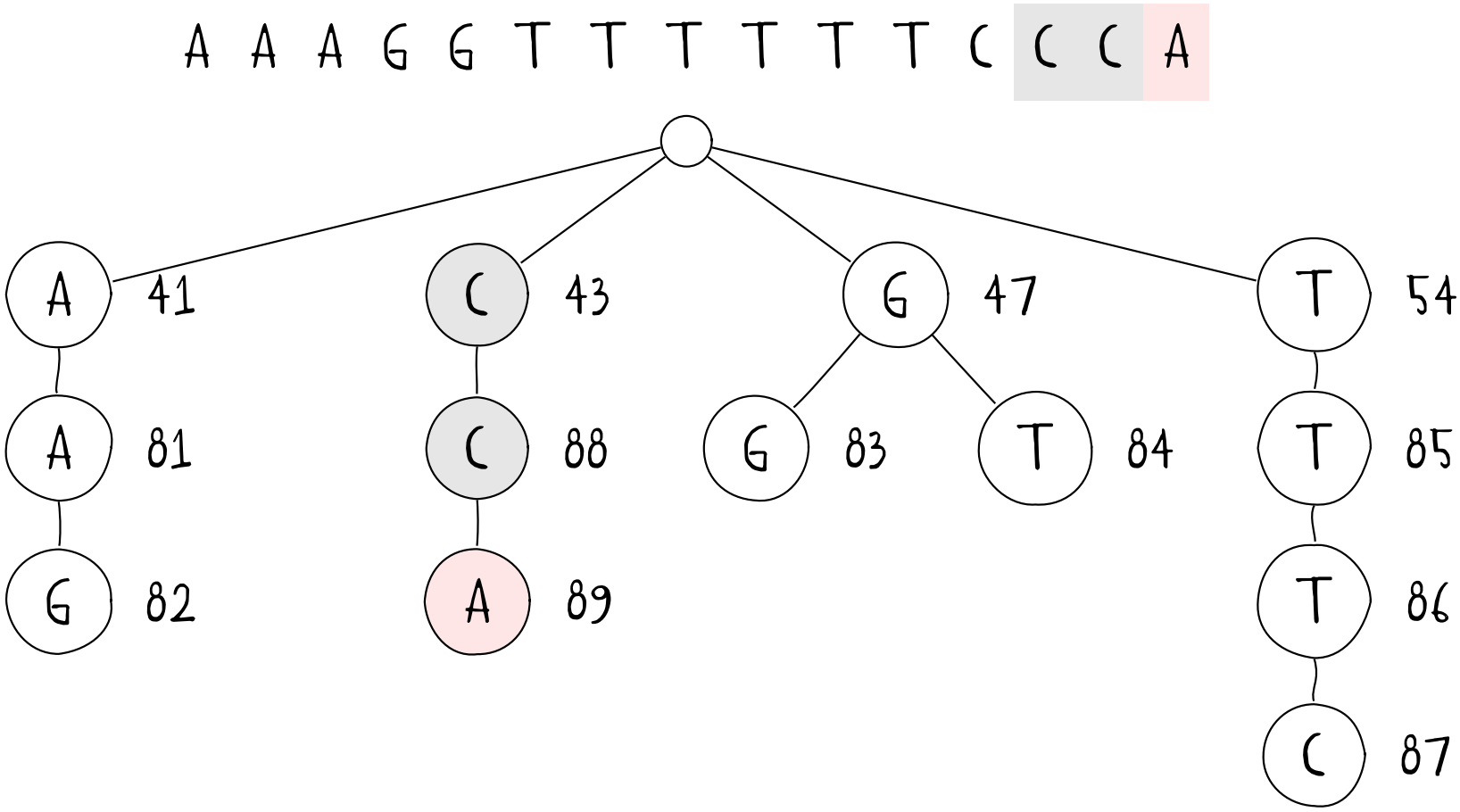
- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 4181474754858643

Compressão de dados

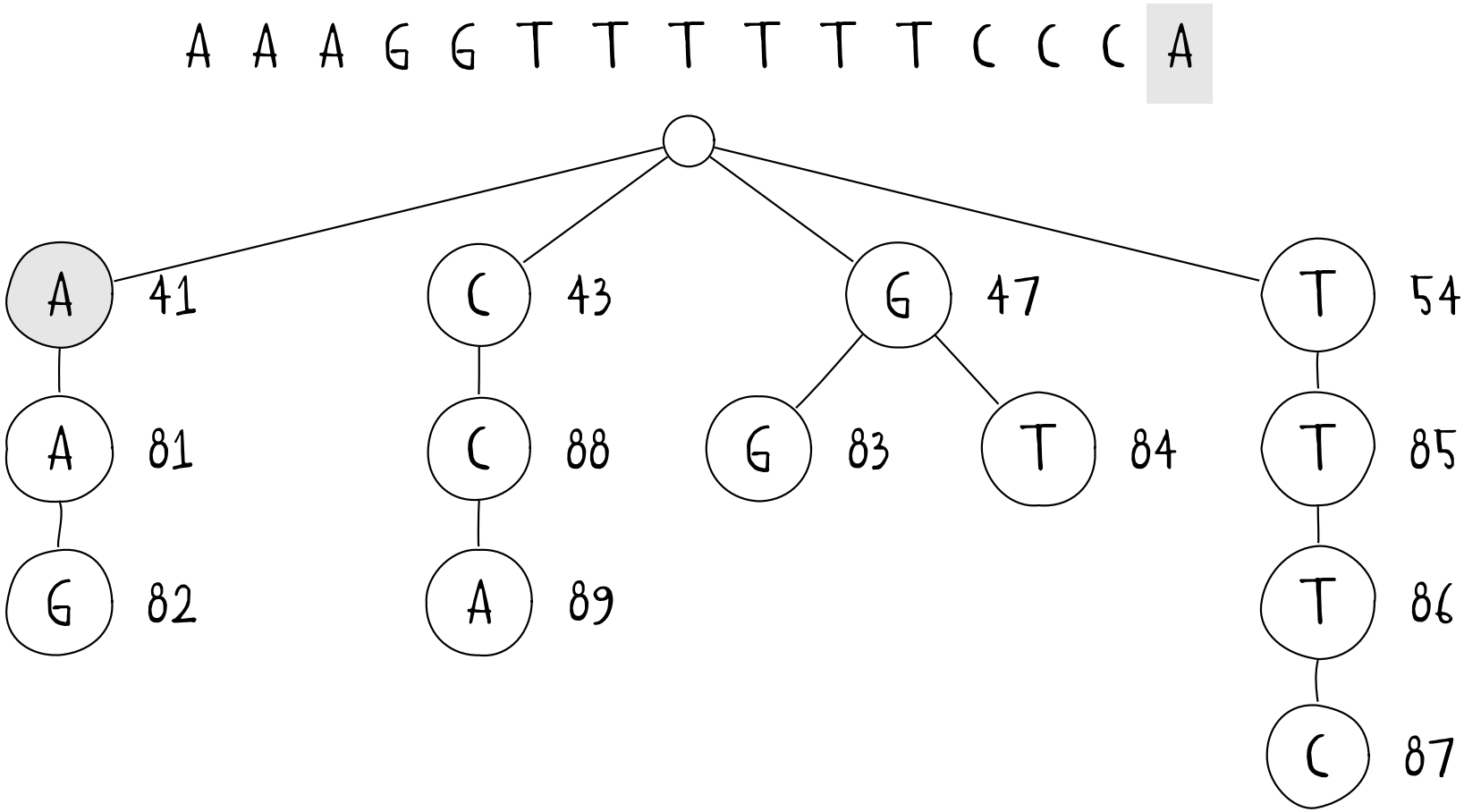
- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 418147475485864388

Compressão de dados

- Codificação Lempel-Ziv-Welch (LZW)
  - Construção da árvore de prefixos (*trie*)



LZW: 418147475486586438841

$$\textit{Taxa de compressão} = 100 \times \frac{80\ \textit{bits}}{120\ \textit{bits}} \approx 67\%$$

Compressão de dados

- Codificação Lempel-Ziv-Welch (LZW)
  ✓ Espaço $\Theta\left(|\sum| + n\right)$, tempo $\Omega\left(n\right)$ e $O\left(n \times m\right)$, onde m é tamanho médio das cadeias de prefixos

Compressão de dados

- Codificação Lempel-Ziv-Welch (LZW)
  ✓ Espaço $\Theta\left(|\sum| + n\right)$, tempo $\Omega\left(n\right)$ e $O\left(n \times m\right)$, onde m é tamanho médio das cadeias de prefixos
  ✓ É utilizado em aplicações de propósito geral em ferramentas de compactação de arquivos (compress) e na especificação de formato de imagens, como GIF, TIFF e PDF, por exemplo

Compressão de dados

- Codificação Lempel-Ziv-Welch (LZW)
  ✓ Espaço $\Theta(|\sum| + n)$, tempo $\Omega(n)$ e $O(n \times m)$, onde m é tamanho médio das cadeias de prefixos
  ✓ É utilizado em aplicações de propósito geral em ferramentas de compactação de arquivos (compress) e na especificação de formato de imagens, como GIF, TIFF e PDF, por exemplo
  ✓ Como a codificação gerada possui tamanho fixo, não existe a necessidade utilização de delimitadores

Compressão de dados

- Codificação Lempel-Ziv-Welch (LZW)
  ✓ Espaço $\Theta\left(|\sum| + n\right)$, tempo $\Omega\left(n\right)$ e $O\left(n \times m\right)$, onde m é tamanho médio das cadeias de prefixos
  ✓ É utilizado em aplicações de propósito geral em ferramentas de compactação de arquivos (compress) e na especificação de formato de imagens, como GIF, TIFF e PDF, por exemplo
  ✓ Como a codificação gerada possui tamanho fixo, não existe a necessidade utilização de delimitadores
  ✓ É mais eficiente em situações onde longos padrões se repetem com alta frequência, como em textos ou de imagens

Compressão de dados

- Codificação Lempel-Ziv-Welch (LZW)
  ✓ Espaço $\Theta(|\sum| + n)$, tempo $\Omega(n)$ e $O(n \times m)$, onde m é tamanho médio das cadeias de prefixos
  ✓ É utilizado em aplicações de propósito geral em ferramentas de compactação de arquivos (compress) e na especificação de formato de imagens, como GIF, TIFF e PDF, por exemplo
  ✓ Como a codificação gerada possui tamanho fixo, não existe a necessidade utilização de delimitadores
  ✓ É mais eficiente em situações onde longos padrões se repetem com alta frequência, como em textos ou de imagens
  ✗ Apesar da simplicidade do algoritmo, o gerenciamento da tabela de códigos pode ser complexo

Exercício

- A empresa de telecomunicações Poxim Tech está desenvolvendo um sistema para compressão de dados, para minimizar o uso de banda na transmissão dos dados, avaliando qual técnica tem a melhor taxa de compressão
  - São fornecidas sequências de bytes em formato hexadecimal que possuem valores entre 00 até *FF*, com tamanho máximo de 10.000 caracteres
  - As codificações de 8 bits *Run-Length Encoding* (RLE) e de Huffman (HUF) são utilizadas para compressão
  - Na implementação da fila de prioridade mínima é utilizada uma estrutura de *heap*
    - Os símbolos são inseridos ordenados em um vetor
    - É feita a construção da árvore em tempo linear
  - A técnica que apresentar menor quantidade de bytes é selecionada para a transmissão dos dados

Exercício

- Formato do arquivo de entrada
  - [#*Quantidade de sequências*]
  - [#T_1] [$B1_1 \ldots B1_n$]
  - $\vdots$
  - [#T_N] [$BN_1 \ldots BN_m$]

```
1 4
2 5 AA AA AA AA AA
3 7 10 20 30 40 50 60 70
4 9 FF FF FF FF FF FF FF FF FF
5 4 FA FA C1 C1
```

Exercício

- Formato do arquivo de saída
  - Cada linha da saída gerada deve conter o algoritmo utilizado na compressão dos dados (RLE ou HUF) e o valor da taxa de compressão com duas casas decimais
  - Em uma situação onde ambos as técnicas apresentarem o mesmo número de bytes na codificação, devem ser impressas ambas as saídas, seguindo a ordem HUF e RLE

```
0->HUF(20.00%)=00
1->HUF(42.86%)=9C6B50
2->HUF(22.22%)=0000
2->RLE(22.22%)=09FF
3->HUF(25.00%)=C0
```