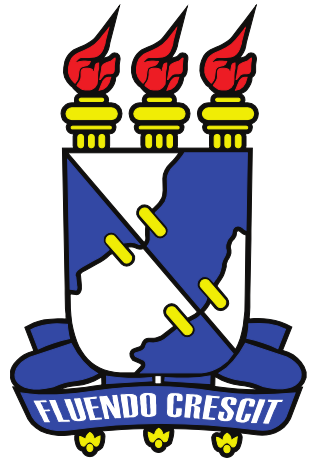


Projeto e Análise de Algoritmos

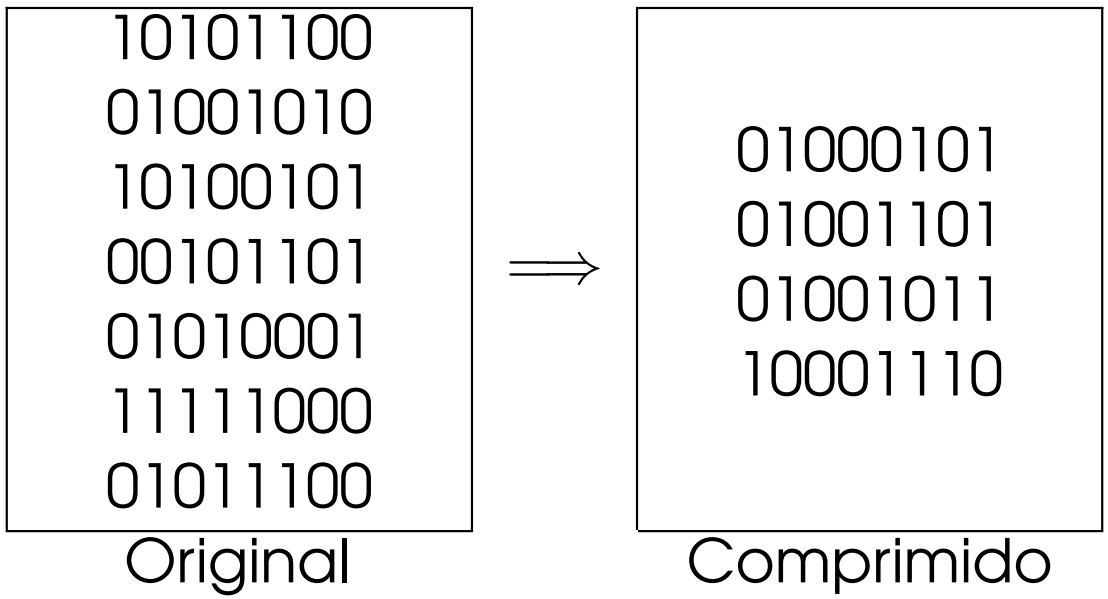
Compressão de dados

Bruno Prado

Departamento de Computação / UFS

Introdução

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



Introdução

- Princípio de operação
  - Explorar a redundância de dados que existem nos diversos tipos de arquivos não processados
    - Letras ou palavras com alto índice de repetição
    - Imagens com grandes áreas homogêneas

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

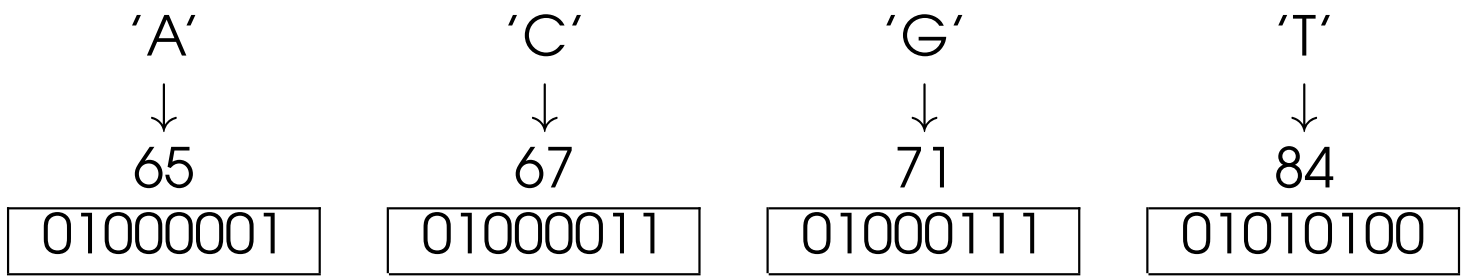
R	R	R	R	G
R	B	B	B	B
B	B	B	B	G
G	G	G	G	G
G	G	G	G	G

Introdução

- Tipos de compressão
  - 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
  - Formato binário
    - Texto simples (ASCII)
    - Arquivos binários (executáveis, imagens, ...)



    - Todos os arquivos são considerados sequências de bits
- A taxa de compressão atingida é dependente das características da entrada utilizada

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

Introdução

- Limitações
  - Não é possível criar um algoritmo universal capaz de comprimir qualquer conjunto de dados em um tamanho menor
    - Se assumirmos que este algoritmo existe, cada execução do algoritmo geraria um arquivo com tamanho menor até que seu tamanho fosse nulo, o que é uma contradição
  - Indecidibilidade
    - Não é possível decidir se um algoritmo é ótimo na compressão de uma determinada cadeia de bits

Compressão de dados

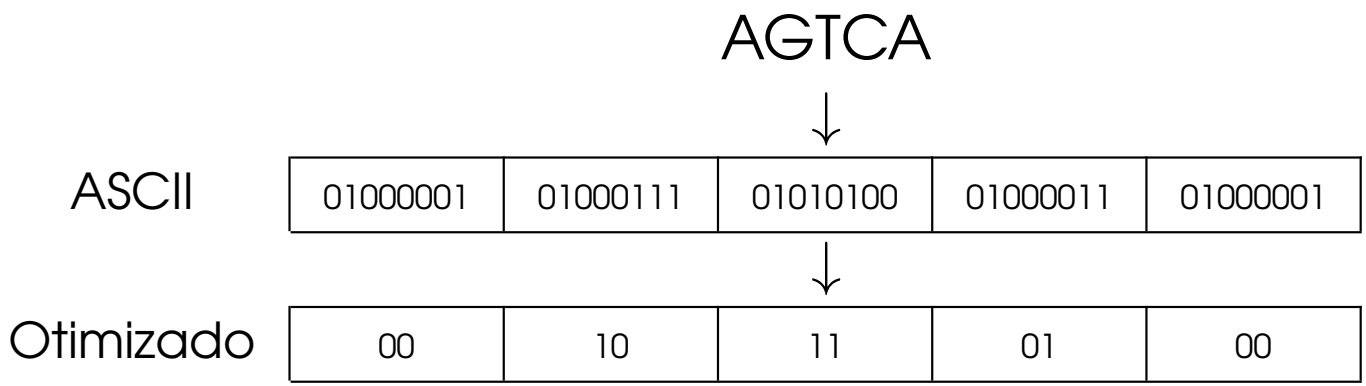
- Representação dos dados
  - Sequência de DNA
    - As cadeias de DNA ou genomas são representadas por um alfabeto de 4 símbolos: A, C, G e T
    - Na representação em texto, são utilizados caracteres de texto que permitem até 256 símbolos distintos

A	C	G	T
01000001	01000011	01000111	01010100
⇓	⇓	⇓	⇓
00	01	10	11

$$\textit{Número de bits} = \log_2 |S|$$

Compressão de dados

- Representação dos dados
  - Sequência de DNA



  - 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% através da representação adequada dos dados
    - O genoma humano possui mais de 10^{10} bits em codificação ASCII

Compressão de dados

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

AAAGGTTTTTTCCCA

↓

00	00	00	10	10	11	11	11	11	11	11	10	10	10	00

↓

3 x A, 2 x G, 6 x T, 3 x C, 1 x A

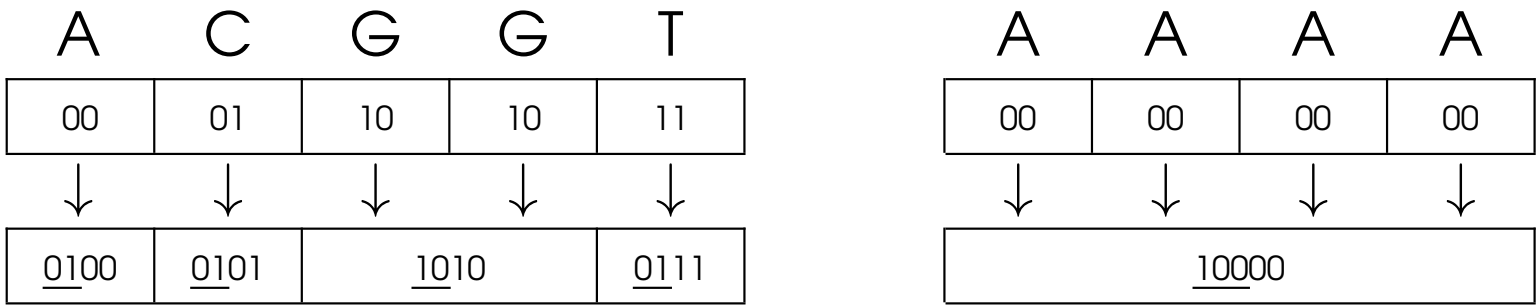
↓

1100	1010	1111	1111	1101	0100

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

Compressão de dados

- Codificação RLE (Run-Length Encoding)
  - Definição do tamanho do contador de símbolos
    - $\downarrow$ Repetição $\longrightarrow \downarrow$ Bits do contador
    - $\uparrow$Repetição $\longrightarrow \uparrow$ Bits do contador



  - O ajuste do tamanho do contador é dependente da entrada utilizada e impacta diretamente na taxa de compressão obtida

Compressão de dados

- Codificação RLE (Run-Length Encoding)
  - Imagem bitmap com 4 bits por pixel
    - Tamanho de 16 x 16 pixels
    - Escala de cinza (0 = preto, F = branco)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	⇒	F010
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	⇒	F010
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	⇒	F010
0	0	0	0	0	0	0	F	F	0	0	0	0	0	0	0	⇒	702F70
0	0	0	0	0	0	0	F	F	0	0	0	0	0	0	0	⇒	702F70
0	0	0	0	0	0	0	F	F	0	0	0	0	0	0	0	⇒	702F70
0	0	0	0	0	0	0	F	F	0	0	0	0	0	0	0	⇒	702F70
0	0	0	F	F	F	F	F	F	F	F	F	F	0	0	0	⇒	30AF30
0	0	0	F	F	F	F	F	F	F	F	F	F	0	0	0	⇒	30AF30
0	0	0	0	0	0	0	F	F	0	0	0	0	0	0	0	⇒	702F70
0	0	0	0	0	0	0	F	F	0	0	0	0	0	0	0	⇒	702F70
0	0	0	0	0	0	0	F	F	0	0	0	0	0	0	0	⇒	702F70
0	0	0	0	0	0	0	F	F	0	0	0	0	0	0	0	⇒	702F70
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	⇒	F010
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	⇒	F010
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	⇒	F010

  - Aplicando RLE de 4 bits nas linhas
    - $\textit{Taxa de compressão} = 100 \times \frac{336\ \textit{bits}}{1024\ \textit{bits}} = 32\%$

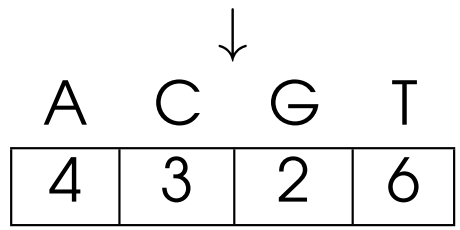
Compressão de dados

- Codificação Huffman
  - Criado em 1952 por David A. Huffman
  - Consiste em utilizar uma quantidade variável de bits para representar os símbolos
    - $\uparrow$ Frequência do símbolo $\longleftrightarrow \downarrow$ # Bits do código
    - $\downarrow$ Frequência do símbolo $\longleftrightarrow \uparrow$ # Bits do código
  - É construída uma árvore de prefixos (trie) para gerar códigos que não são prefixo de nenhum outro, permitindo eliminar a necessidade de delimitadores

Compressão de dados

- Codificação Huffman
  - Cálculo do histograma: a sequência é processada para contabilizar a frequência de ocorrência dos símbolos do alfabeto $\sum$

AAAGGTTTTTTCCCA



  - Análise de complexidade
    - Espaço $O(\sum)$
    - Tempo $O(n)$

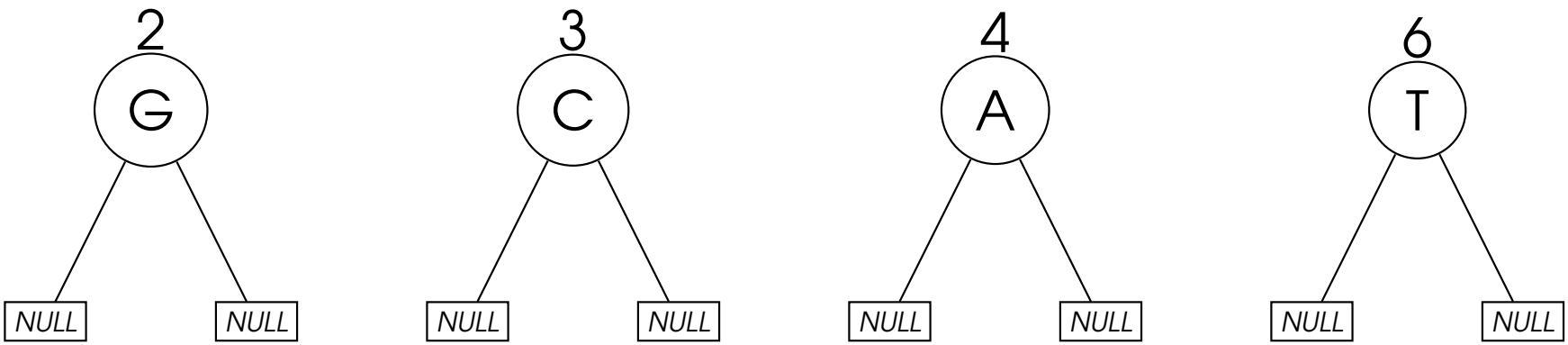
Compressão de dados

- Codificação Huffman
  - Criação da árvore de prefixos: esta árvore permite a codificação binárias de tamanho mínimo e sem repetição de prefixos para eliminar a necessidade de delimitadores entre os códigos gerados

```
// Estrutura de nó
typedef struct no {
        // Frequência
        unsigned int freq;
        // Código do símbolo
        char simb;
        // Nó direito
        no* dir;
        // Nó esquerdo
        no* esq;
} no;
```

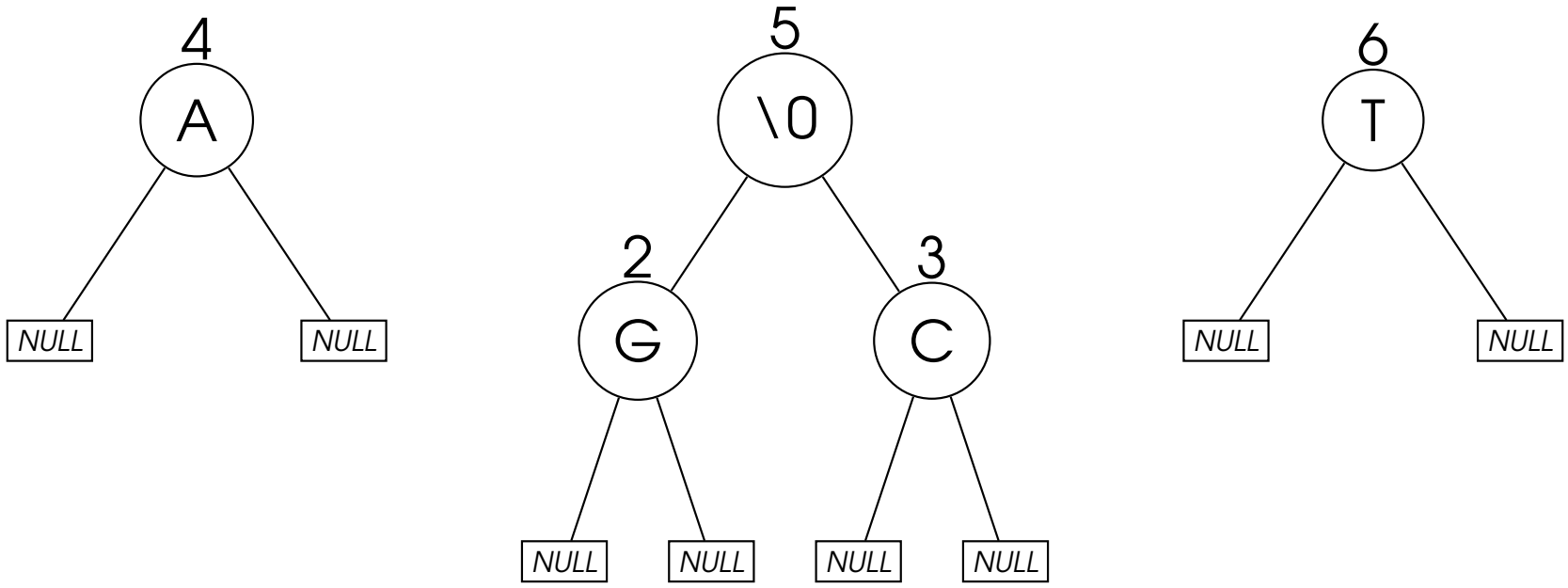
Compressão de dados

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



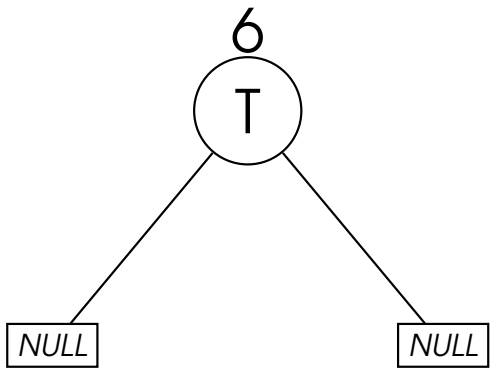
Compressão de dados

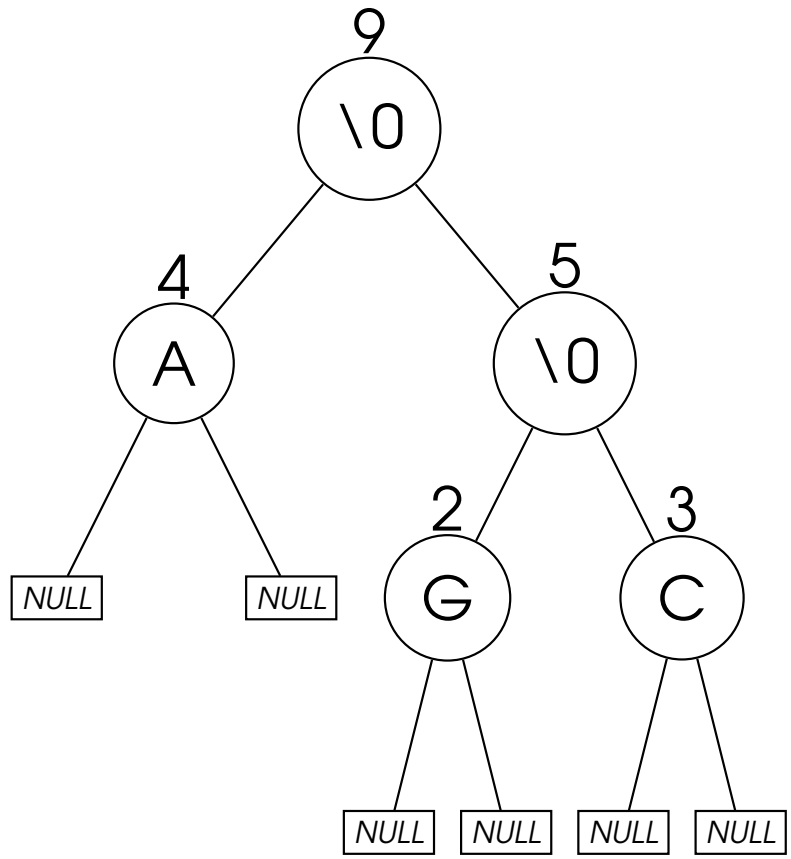
- Codificação Huffman
  - Criação da árvore de prefixos: é feita a remoção dos nós G e C da fila de prioridade mínima e a criação de um nó de símbolo nulo



Compressão de dados

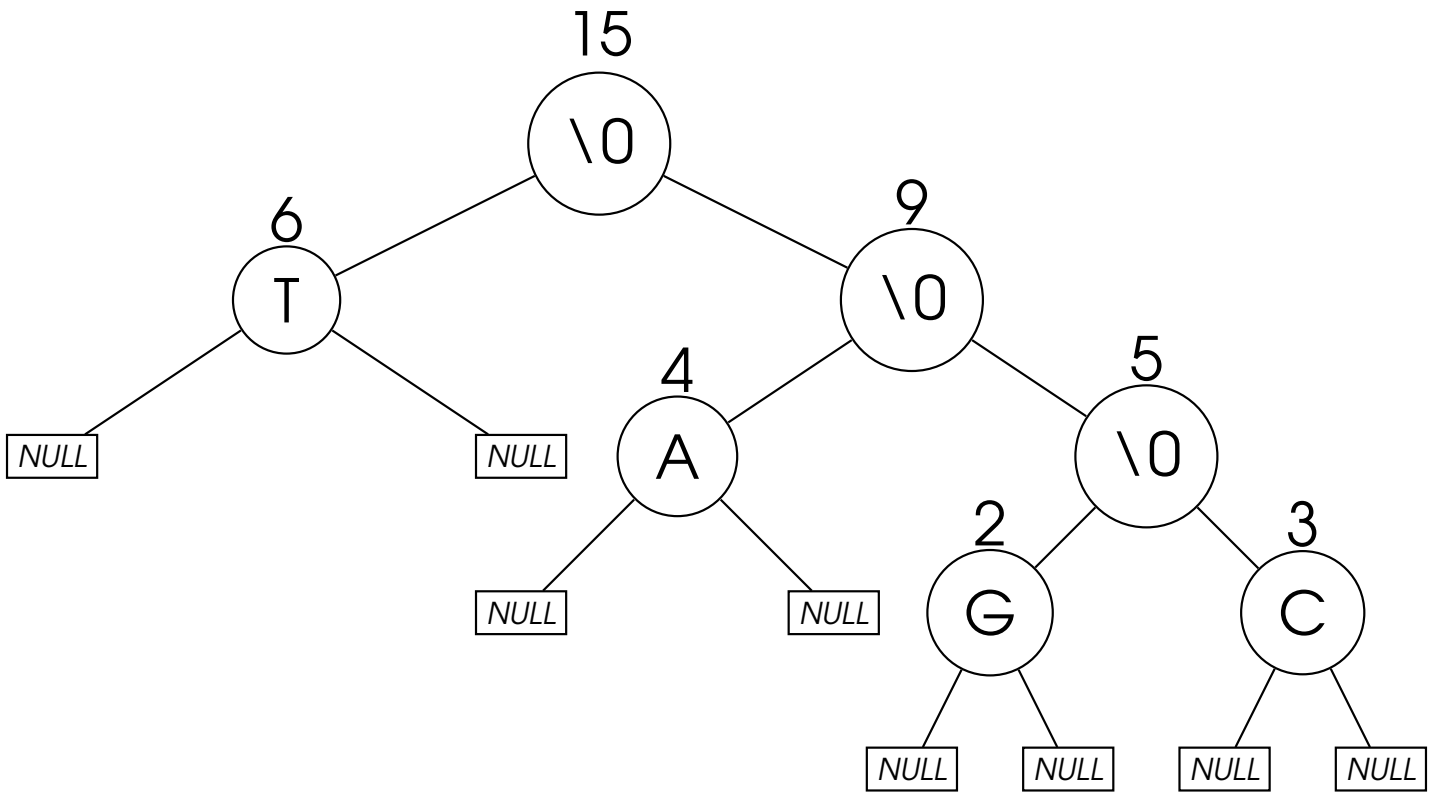
- Codificação Huffman
  - Criação da árvore de prefixos: é feita a remoção dos nós A e \0 da fila de prioridade mínima e a criação de um nó de símbolo nulo





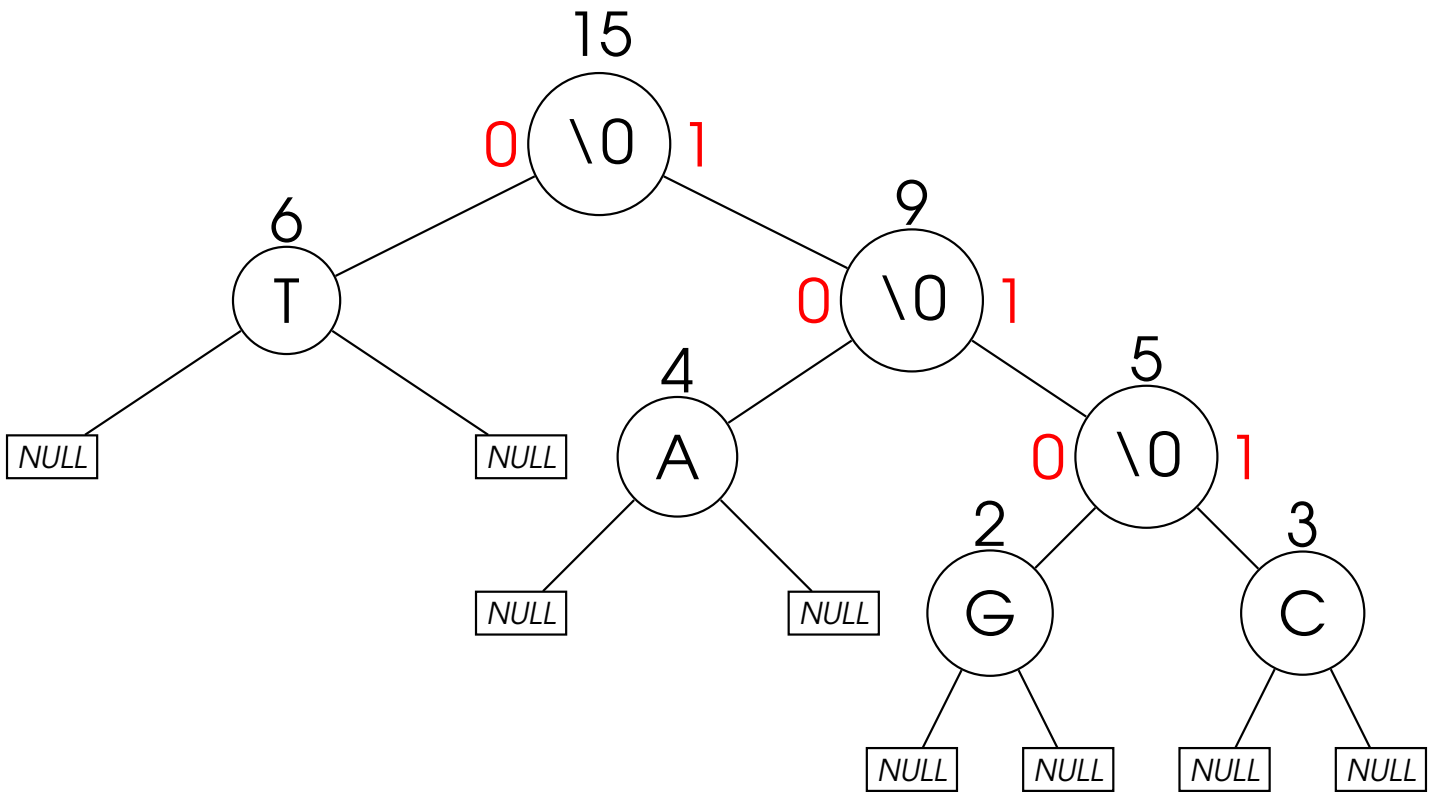
Compressão de dados

- Codificação Huffman
  - Criação da árvore de prefixos: é feita a remoção dos nós T e \0 da fila de prioridade mínima e a criação de um nó de símbolo nulo



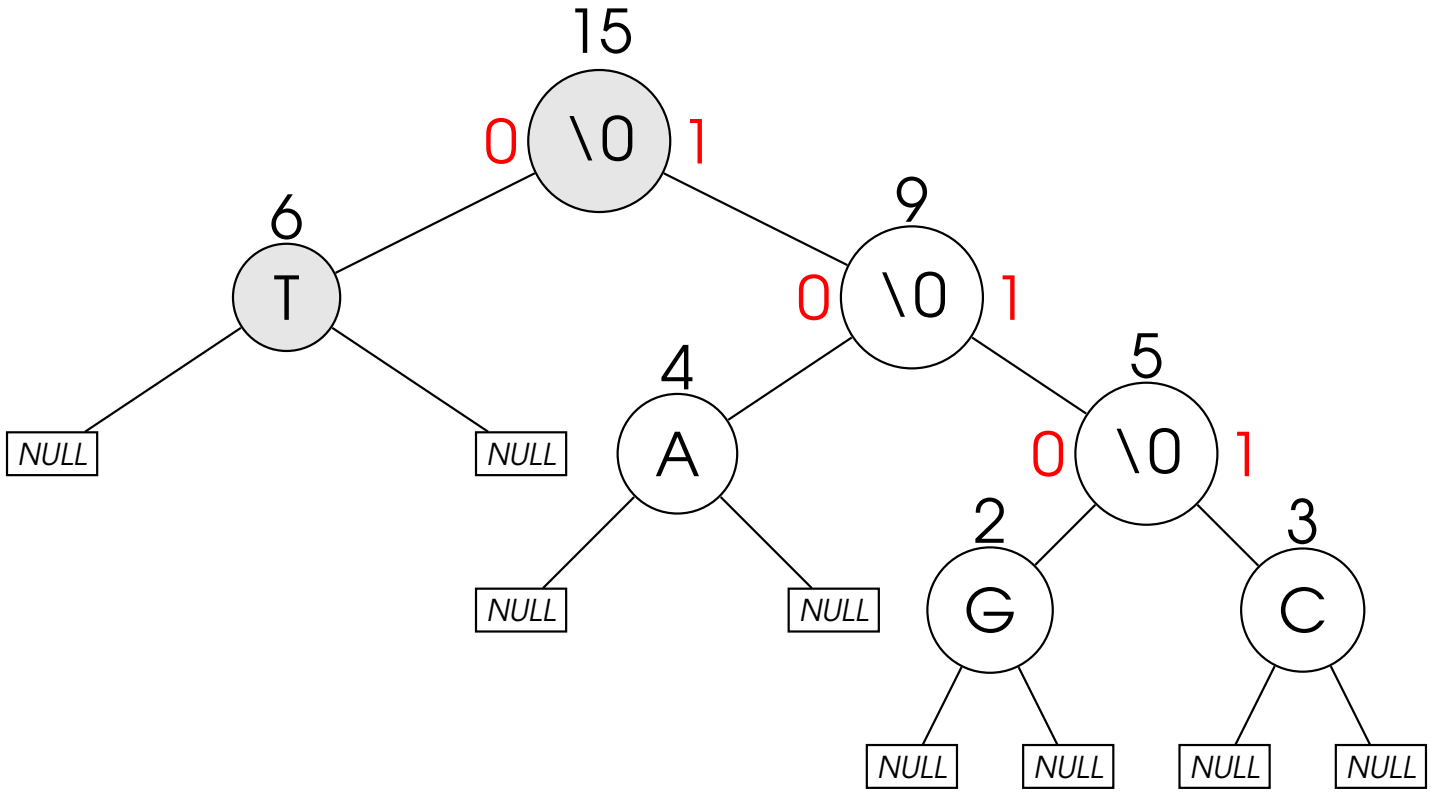
Compressão de dados

- Codificação Huffman
  - Criação da árvore de prefixos: a árvore de prefixos está concluída e é convencionado que o encaminhamento pela esquerda e direita são respectivamente representados pelos bits 0 e 1



Compressão de dados

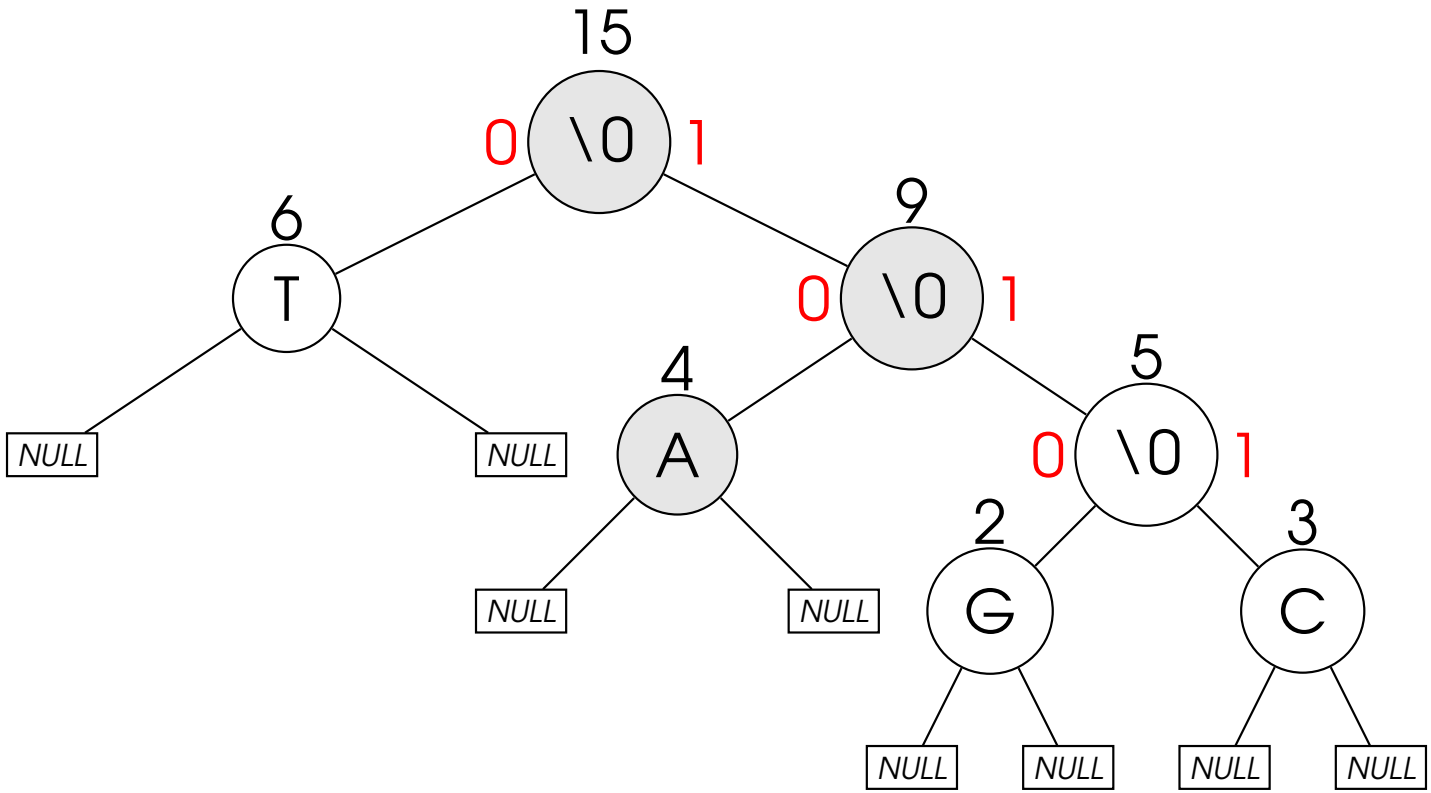
- Codificação Huffman
  - Construção da tabela de codificação: os códigos para os símbolos são gerados através do encaminhamento na árvore de prefixos



T = 0

Compressão de dados

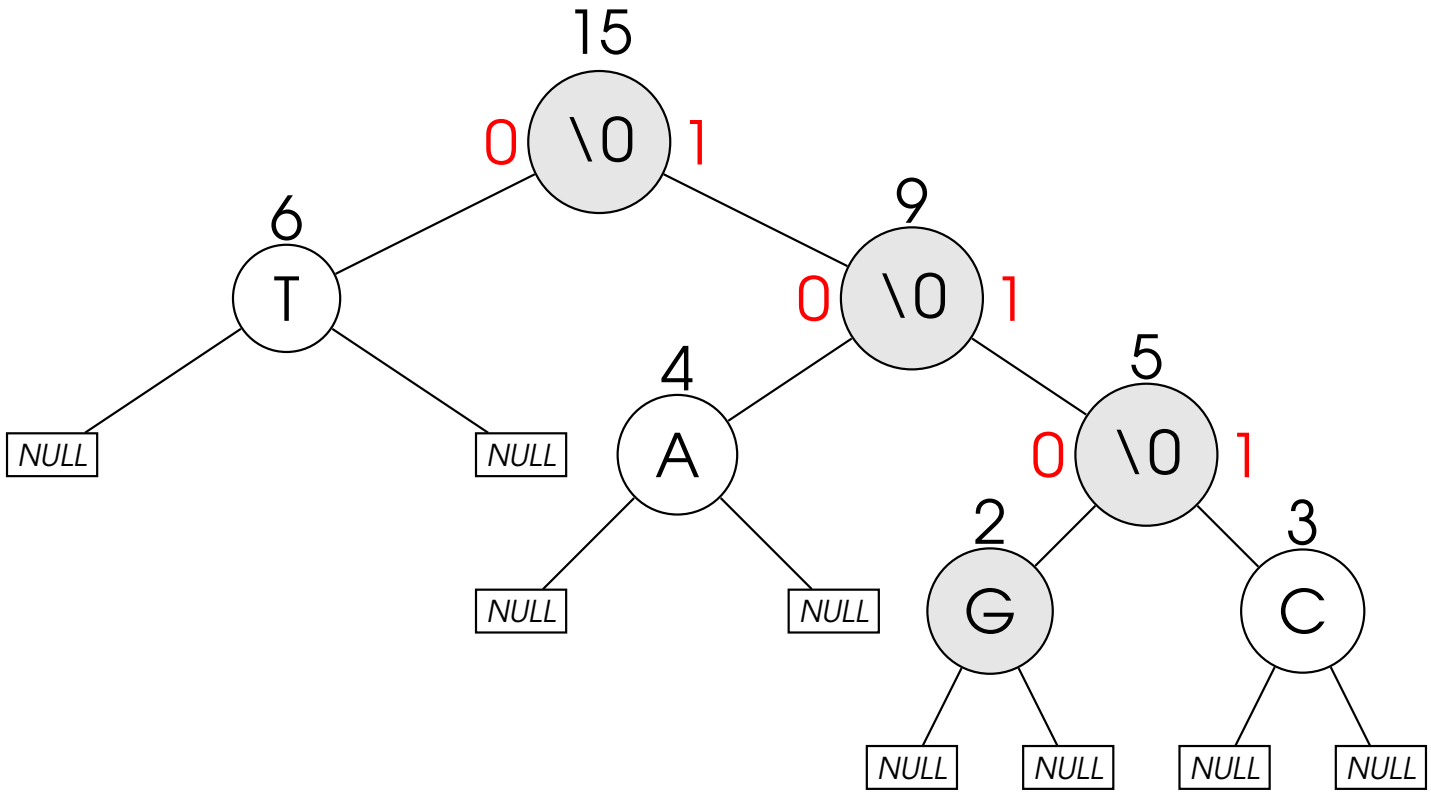
- Codificação Huffman
  - Construção da tabela de codificação: os códigos para os símbolos são gerados através do encaminhamento na árvore de prefixos



A = 10

Compressão de dados

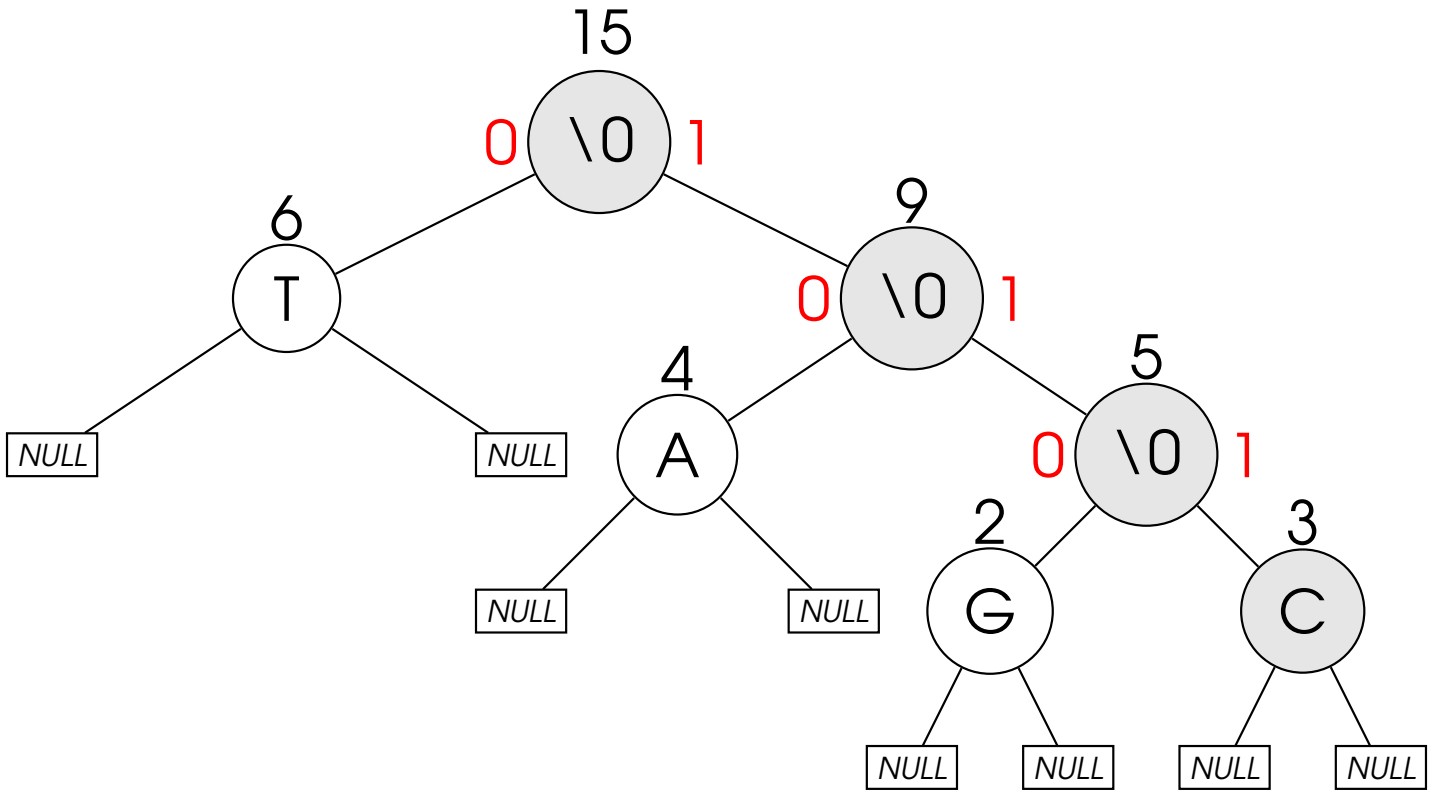
- Codificação Huffman
  - Construção da tabela de codificação: os códigos para os símbolos são gerados através do encaminhamento na árvore de prefixos



G = 110

Compressão de dados

- Codificação 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 Huffman
  - Criação da árvore de prefixos: procedimento para construção da árvore de prefixos do histograma

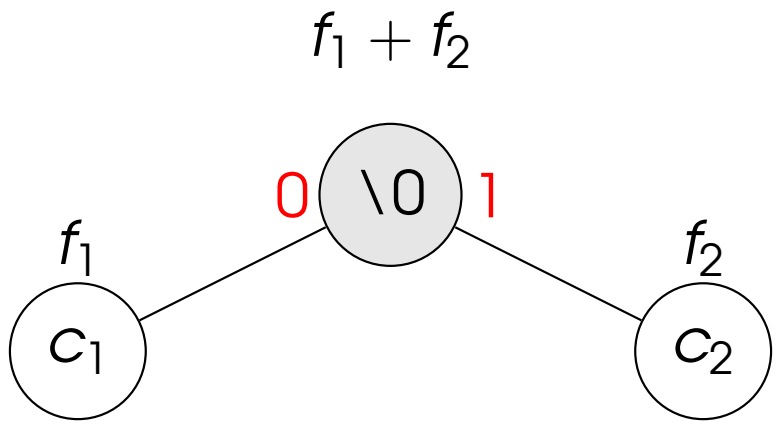
```
no* construir_arvore(unsigned int H(), unsigned int N) {
    unsigned int i;
    fila_p_min* fp_min = criar_fila_p_min();
    for(i = 0; i < N; i++)
        if(H(i) > 0)
            inserir_no(fp_min, H(i), i, NULL, NULL);
    while(tamanho(fp_min) > 1) {
        no* x = extrair_min(fp_min);
        no* y = extrair_min(fp_min);
        inserir_no(fp_min, x.freq + y.freq, '\0', x, y);
    }
    return extrair_min(fp_min);
}
```

Compressão de dados

- Codificação Huffman
  - Análise de complexidade
    - Espaço $O(n)$
    - Tempo $O(n \log_2 n)$

Compressão de dados

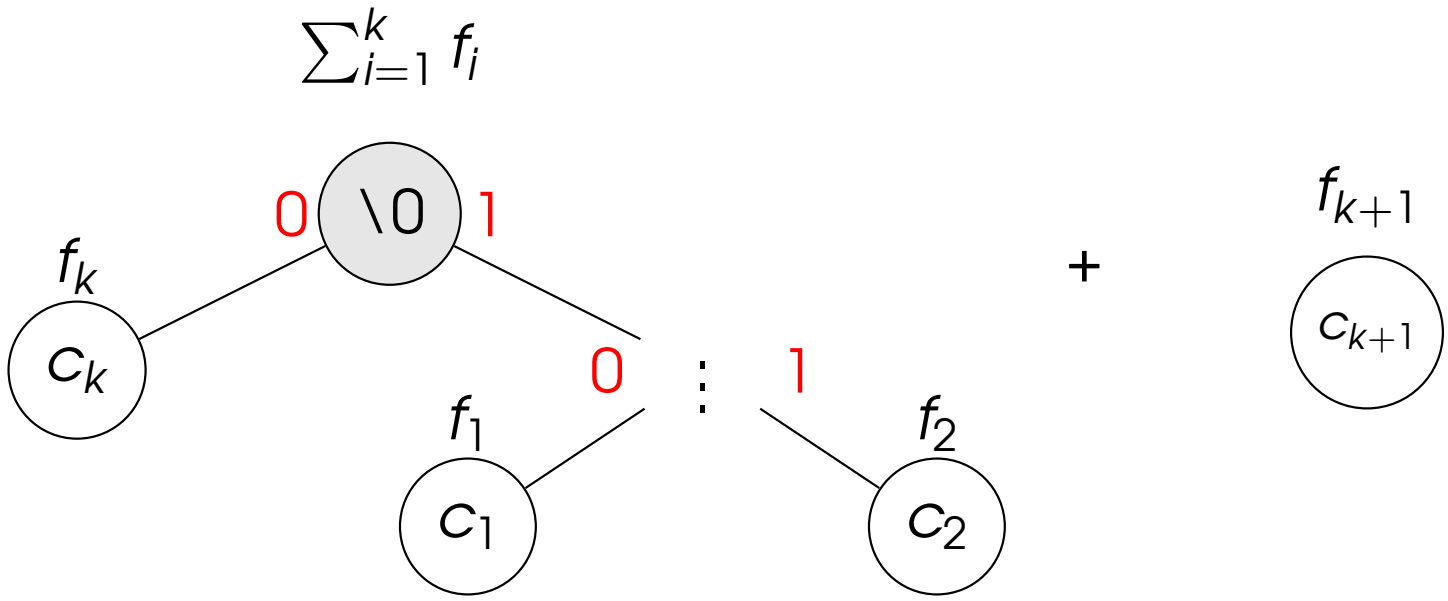
- Codificação Huffman
  - Prova por indução da otimalidade da codificação
    - Considere o conjunto de símbolos $C = \{c_1, c_2, \ldots, c_N\}$ com suas frequências denotadas por $f_1 \leq f_2 \leq \ldots \leq f_N$ em uma determinada sequência



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

Compressão de dados

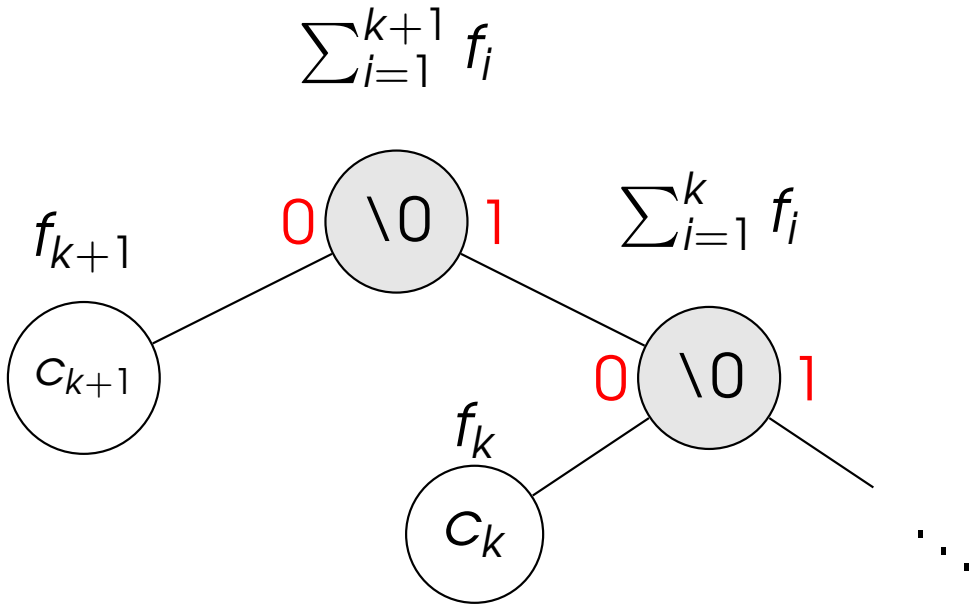
- Codificação Huffman
  - Prova por indução da otimalidade da codificação
    - Considere o conjunto de símbolos $C = \{c_1, c_2, \ldots, c_N\}$ com suas frequências denotadas por $f_1 \leq f_2 \leq \ldots \leq f_N$ em uma determinada sequência



Hipótese indutiva: $N = k$ (m bits)

Compressão de dados

- Codificação Huffman
  - Prova por indução da otimalidade da codificação
    - Considere o conjunto de símbolos $C = \{c_1, c_2, \ldots, c_N\}$ com suas frequências denotadas por $f_1 \leq f_2 \leq \ldots \leq f_N$ em uma determinada sequência



Tese: $N = k + 1$ ($m + 1$ bits)

Compressão de dados

- Codificação Huffman
  - Compressão da sequência

Símbolo	Código
A	10
C	111
G	110
T	0

```
void compactar(char* C, char* TAB, char* U) {
        unsigned int i;
        for(i = 0; i < strlen(U); i++)
                anexar(C, TAB(U(i)));
}
```

AAAGGTTTTTTCCCA

↓

10	10	10	110	110	0	0	0	0	0	0	111	111	111	10

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

Compressão de dados

- Codificação Huffman
  - Características chave
    - É eficiente em diversos domínios de aplicação
    - Permite a codificação sem utilização de delimitadores
    - Em situações onde os símbolos não estão bem distribuídos, como em caracteres de texto, a taxa de compressão obtida é expressiva

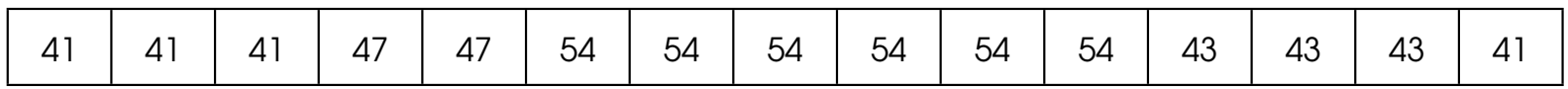
Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - 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 variável 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 LZW (Lempel-Ziv-Welch)
  - Os símbolos de entrada são representados em código ASCII de 7 bits em formato hexadecimal

AAAGGTTTTTTCCCA



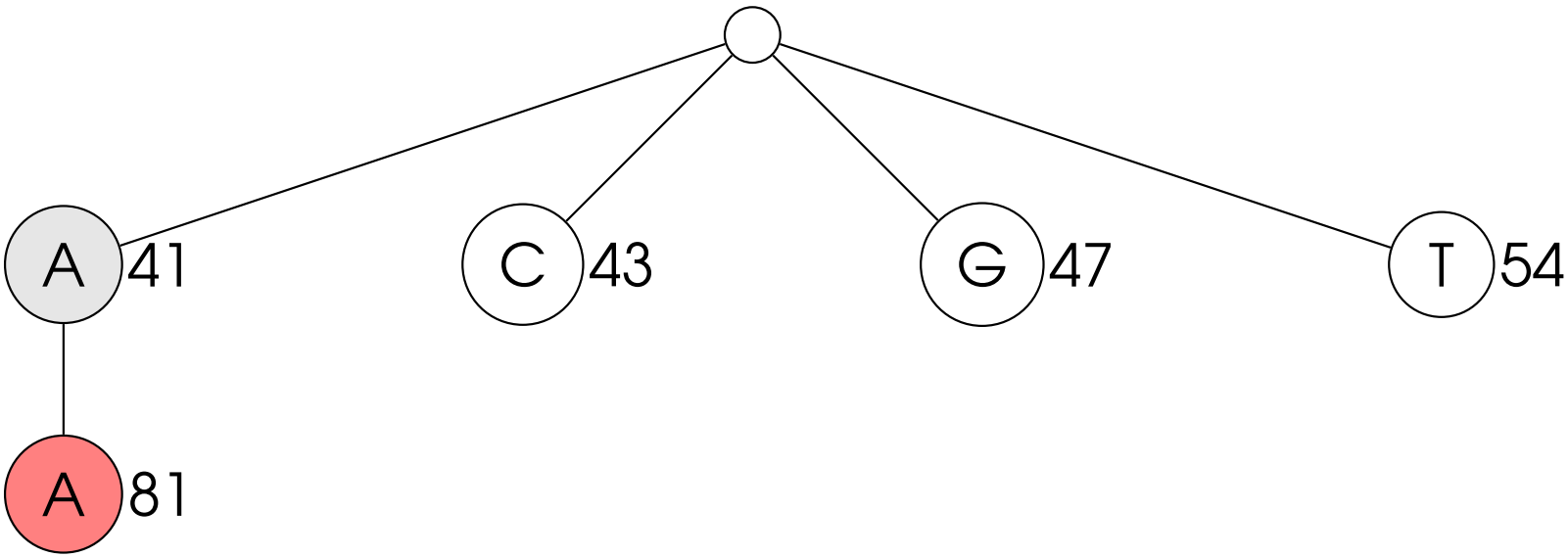
120 bits

  - A codificação dos símbolos é feita com 8 bits
  - Em implementações de propósito geral são utilizados símbolos de entrada com 8 bits e codificação de 12 bits para se adequar a arquivos maiores

Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

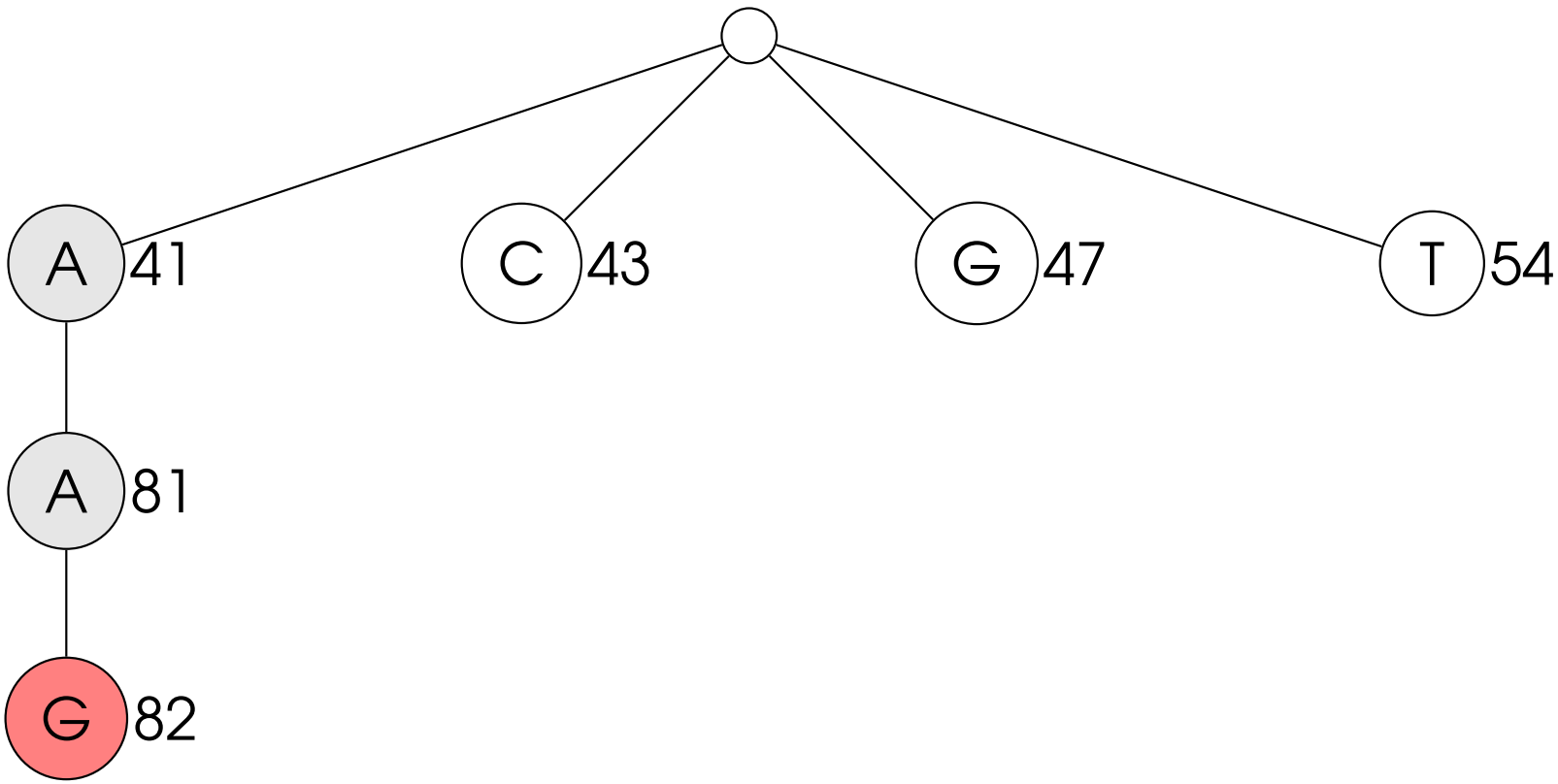
AAAGGTTTTTTCCCA $\rightarrow$ 41



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

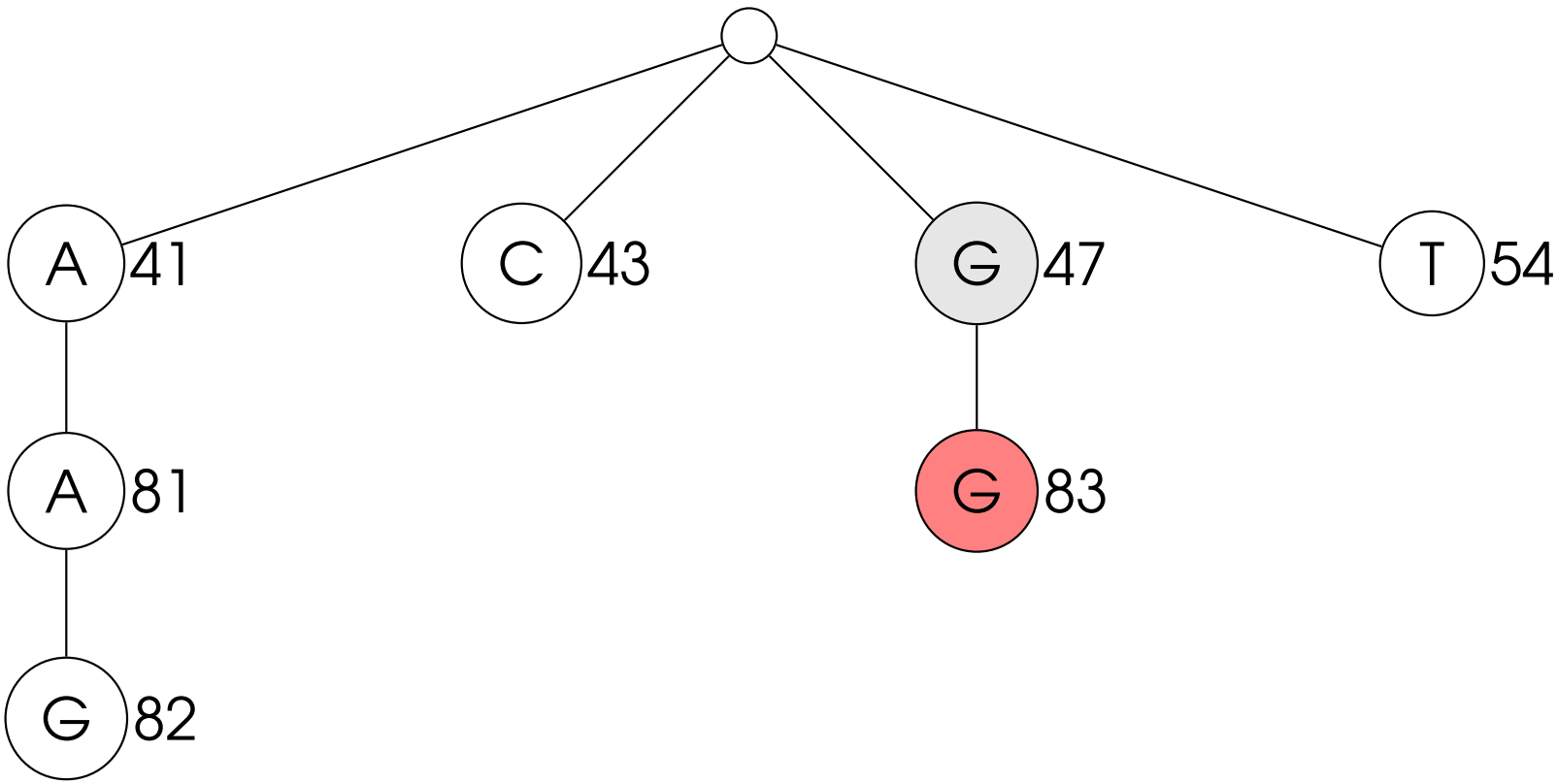
AAAGGTTTTTTCCCA $\rightarrow$ 41 81



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

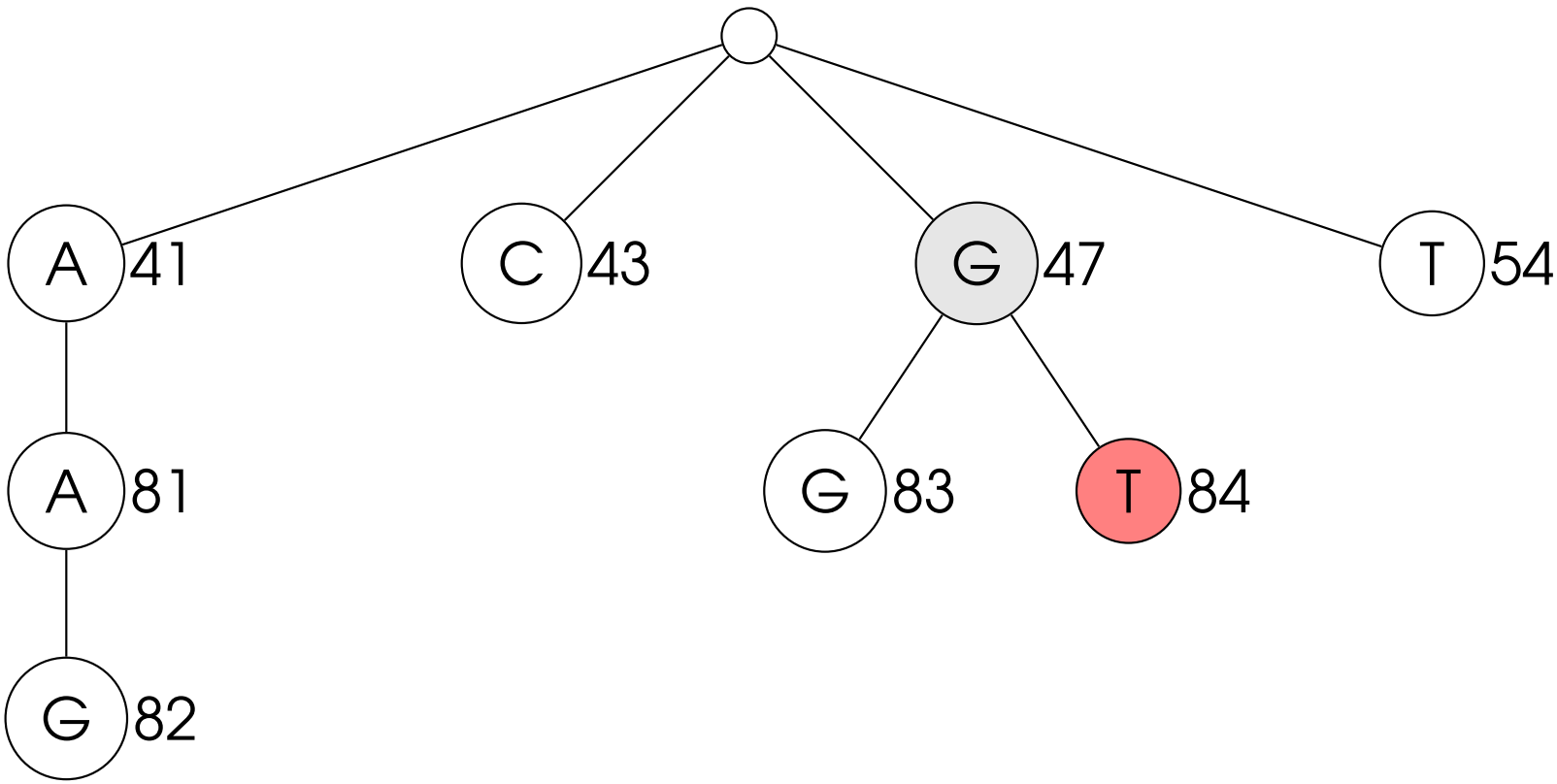
AAAGGTTTTTTCCCA $\rightarrow$ 41 81 47



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

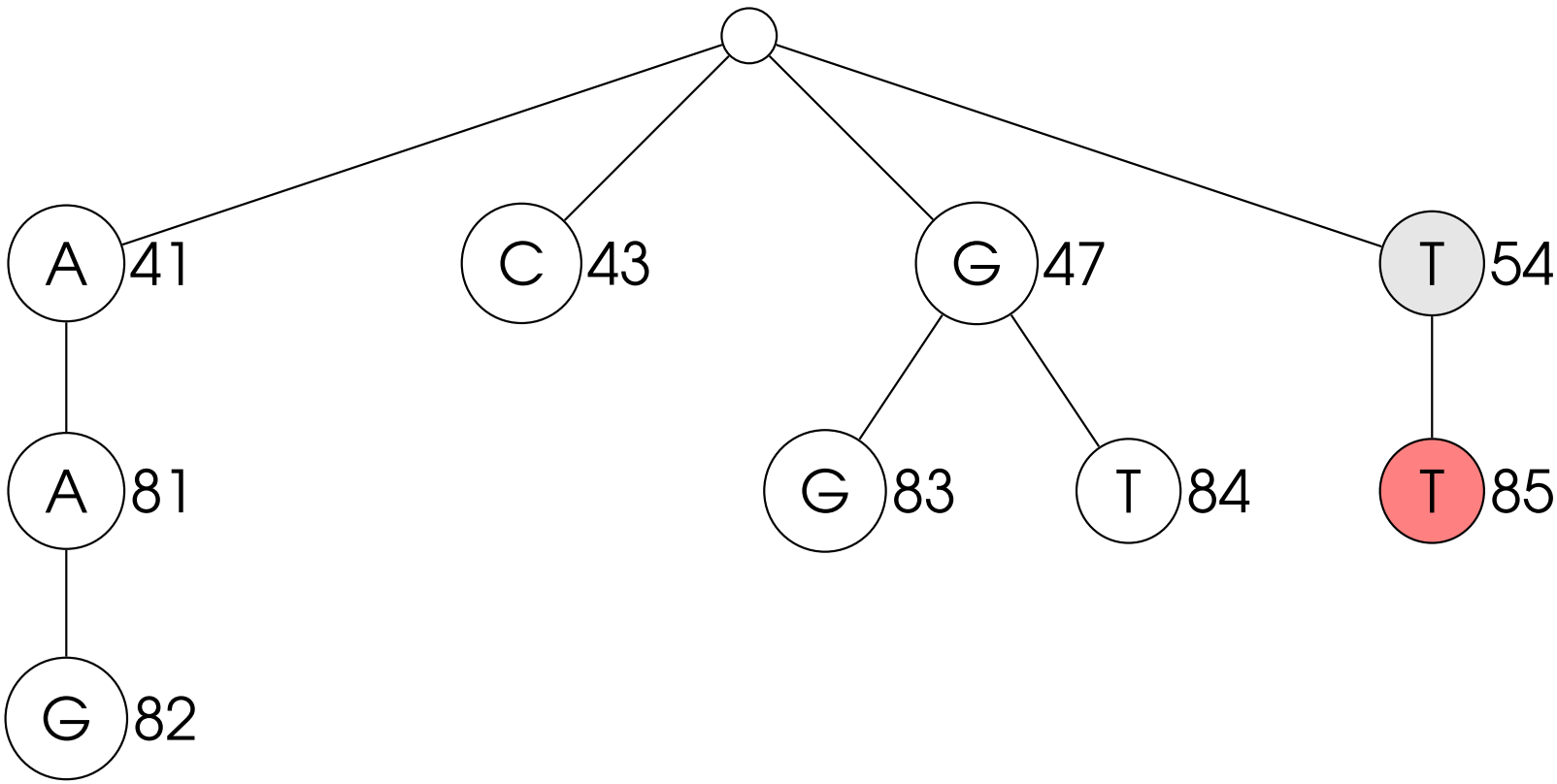
AAAGGTTTTTTCCCA $\rightarrow$ 41 81 47 47



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

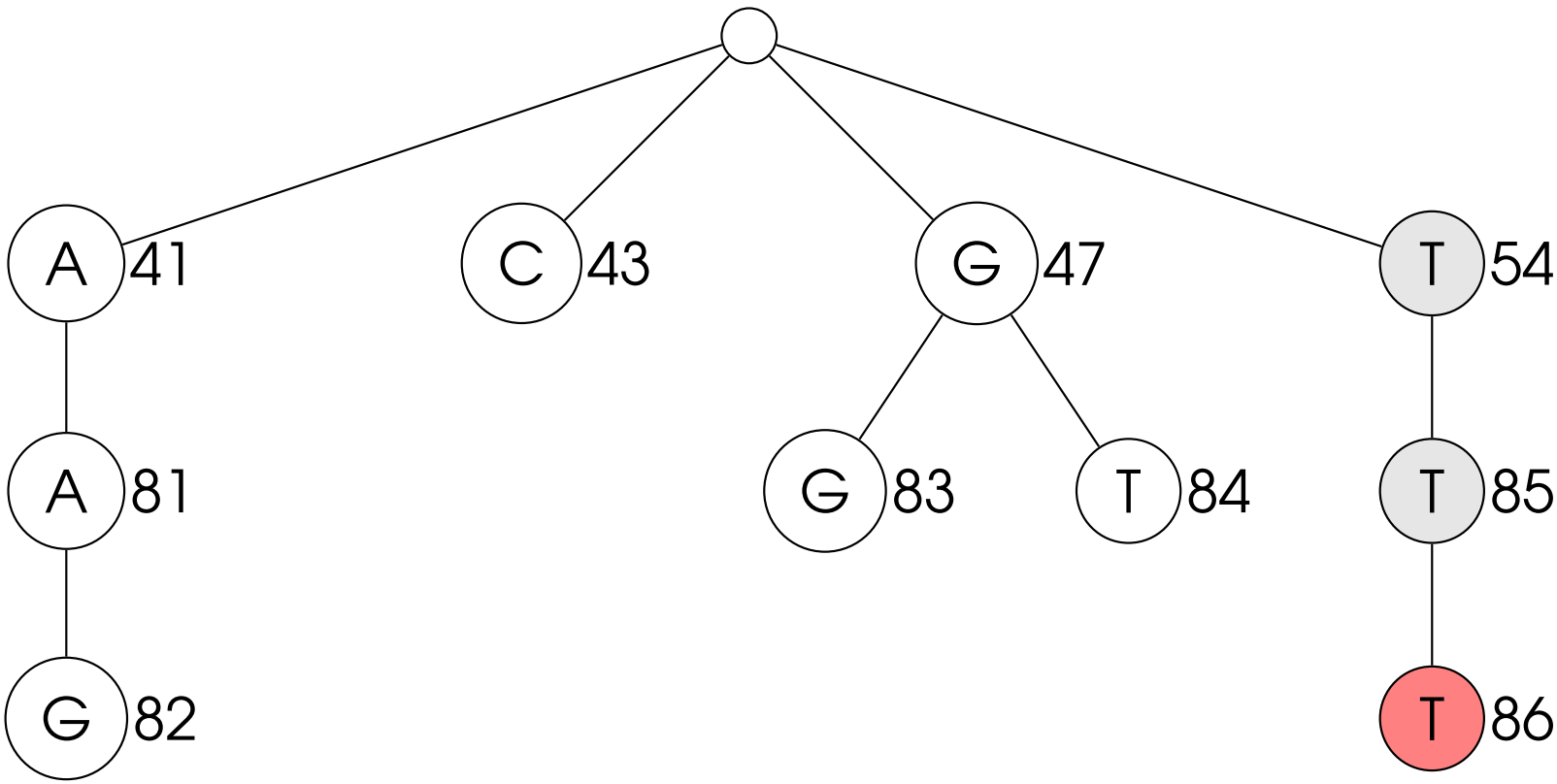
AAAGGTTTTTTCCCA $\rightarrow$ 41 81 47 47 54



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

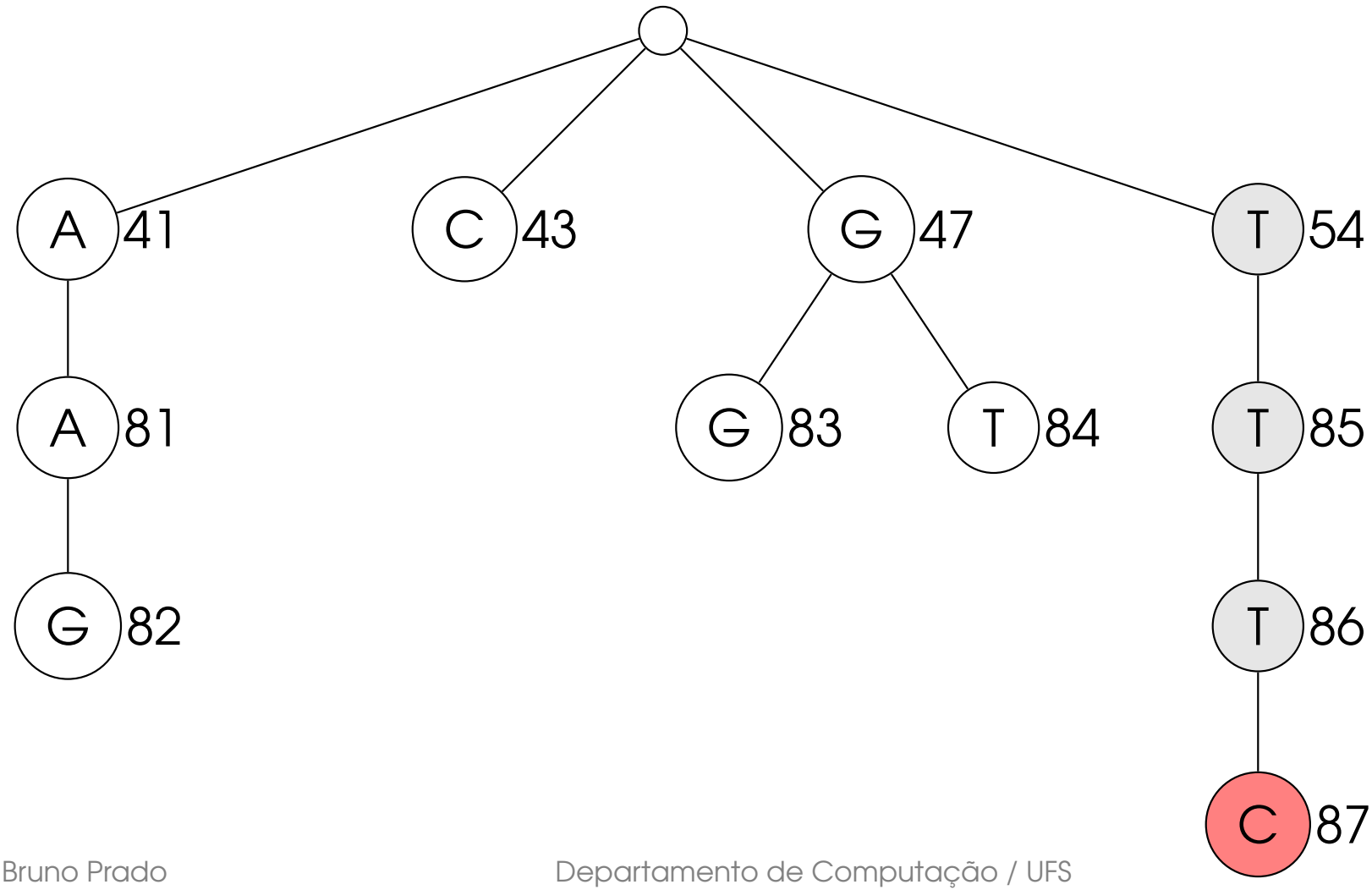
AAAGGTTTTTTCCCA $\rightarrow$ 41 81 47 47 54 85



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

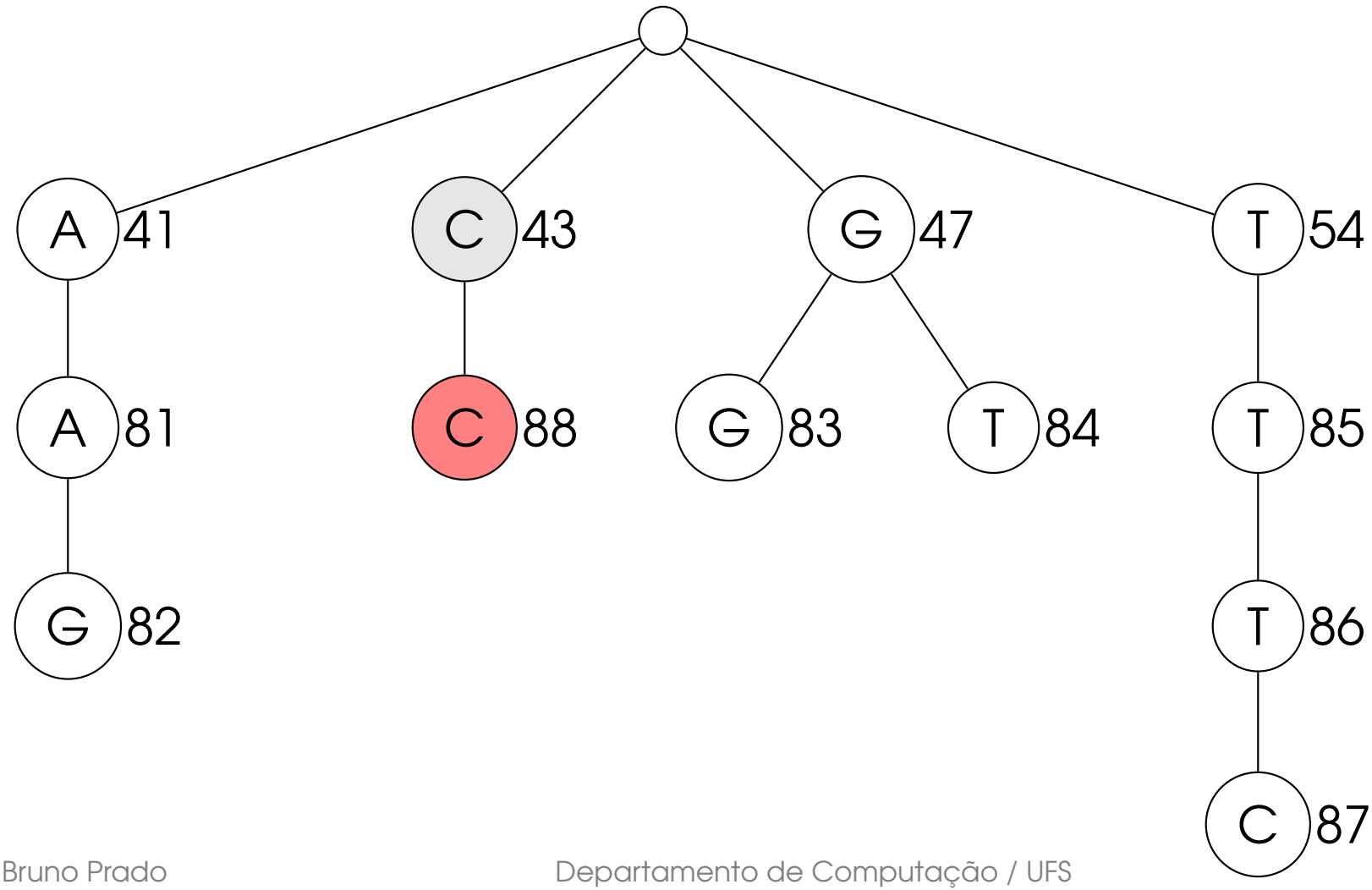
AAAGGTTTTTTCCCA $\rightarrow$ 41 81 47 47 54 85 86



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

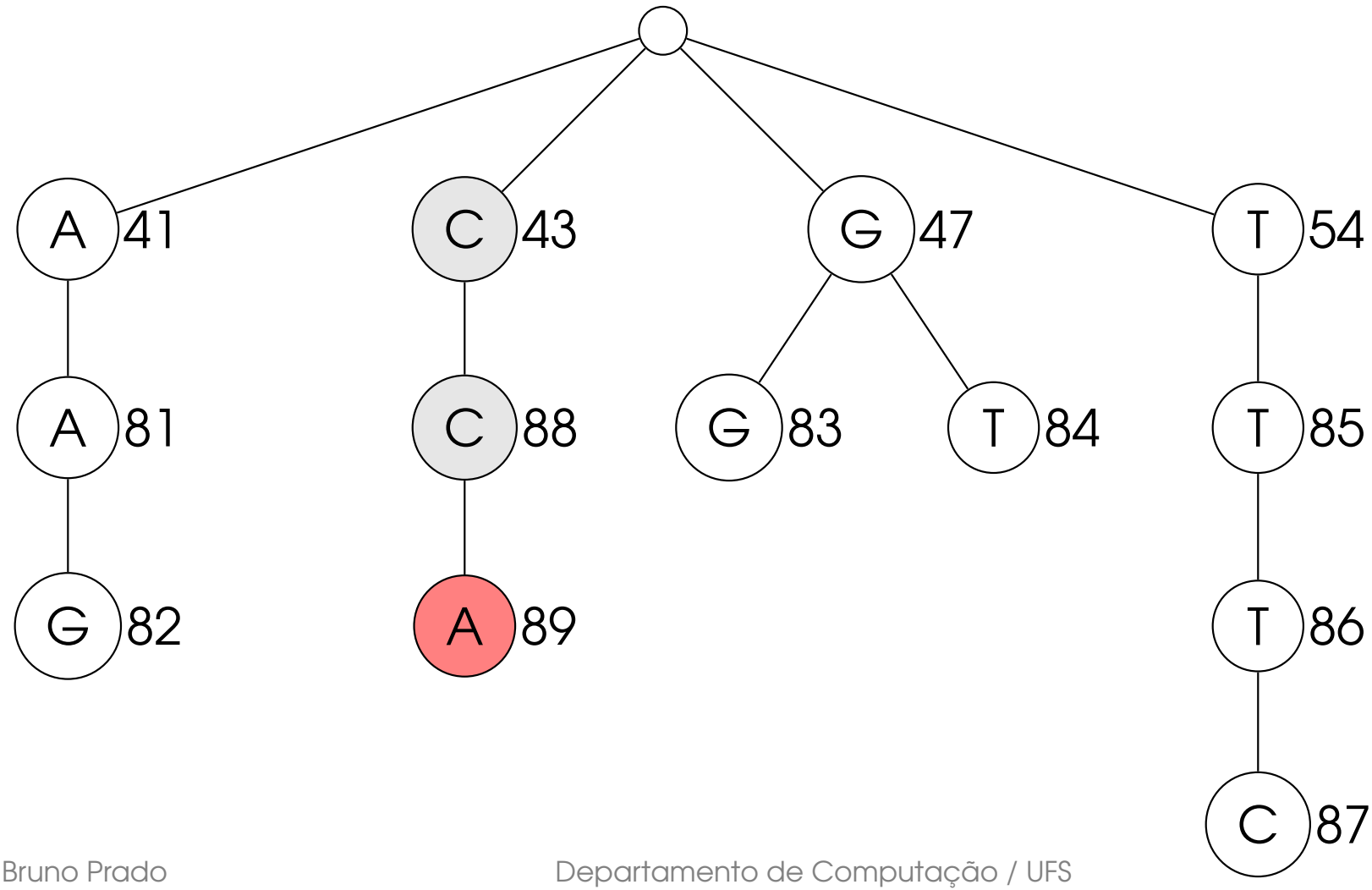
AAAGGTTTTTTCCCA $\rightarrow$ 41 81 47 47 54 85 86 43



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

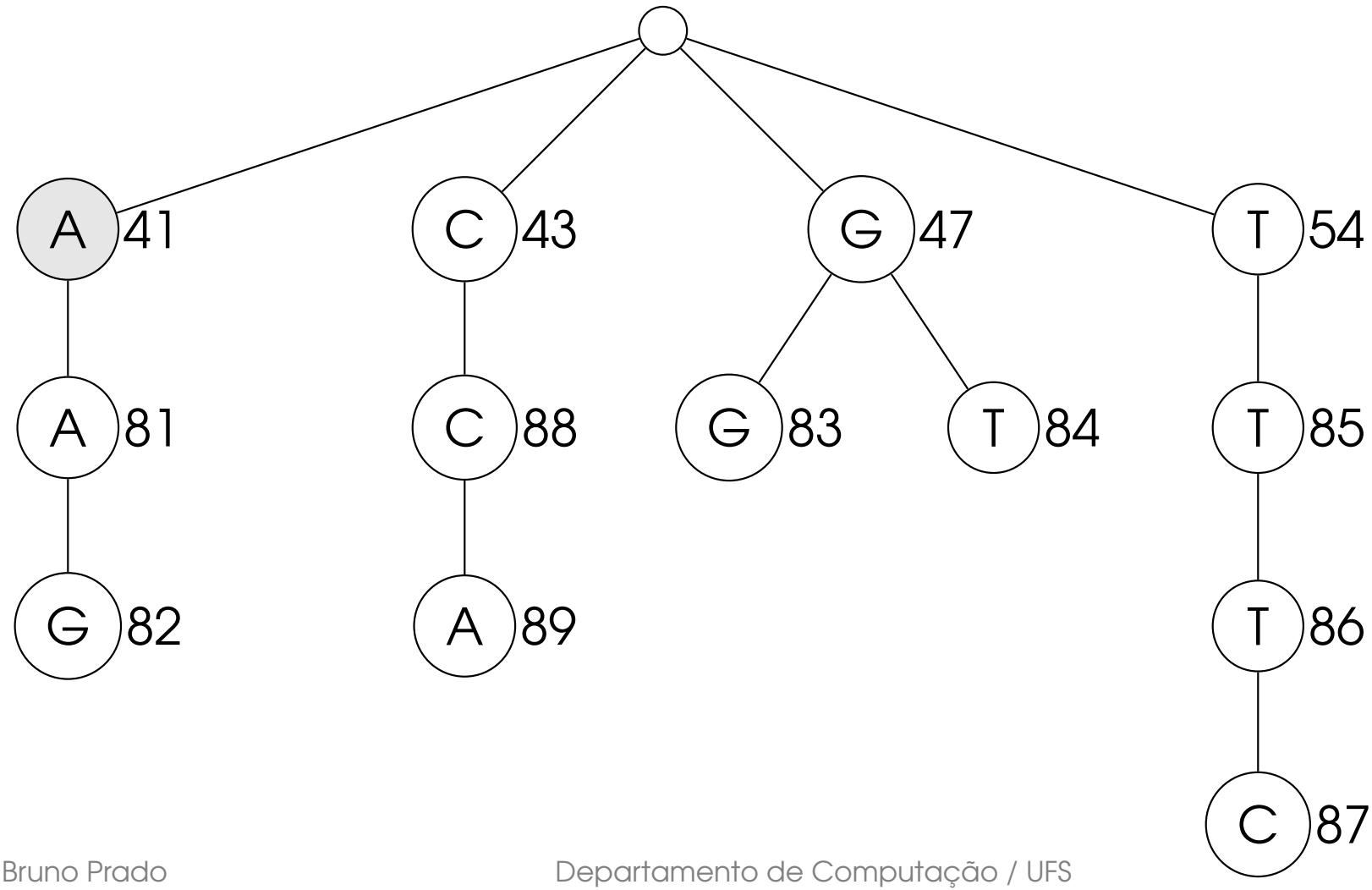
AAAGGTTTTTTCCCA $\rightarrow$ 41 81 47 47 54 85 86 43 88



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Criação da árvore de prefixos

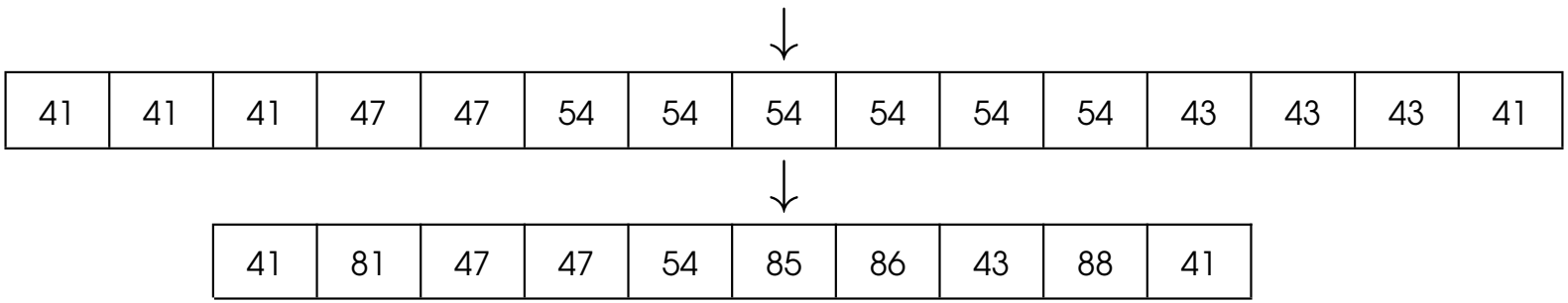
AAAGGTTTTTTCCCA $\rightarrow$ 41 81 47 47 54 85 86 43 88 41



Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Compressão da sequência

AAAGGTTTTTTCCCA



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

Compressão de dados

- Codificação LZW (Lempel-Ziv-Welch)
  - Características chave
    - Possui aplicações de propósito geral em ferramentas de compactação de arquivos (compress) e na especificação de formato de imagens (GIF, TIFF e PDF)
    - A codificação gerada possui tamanho fixo, eliminando a necessidade de delimitadores
    - Este algoritmo é mais eficiente em situações onde longos padrões da entrada se repetem com uma alta frequência, como em arquivos de texto ou de imagens

Exercício

- A empresa de telecomunicações Poxim Tech está desenvolvendo um sistema para compressão de dados para minimizar a utilização de banda na transmissão dos dados, avaliando qual técnica apresenta a melhor taxa de compressão
  - São fornecidas sequências de bytes em formato hexadecimal que possuem valores entre 0x00 até 0xFF, com tamanho máximo de 10000 caracteres
  - As codificações de 8 bits Run-Length Encoding (RLE) e de Huffman (HUF) são utilizadas para compressão
  - 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] [$BN_1 \ldots BN_m$]

```
4
5 0xAA 0xAA 0xAA 0xAA 0xAA
7 0x10 0x20 0x30 0x40 0x50 0x60 0x70
9 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
4 0xFA 0xFA 0xC1 0xC1
```

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 é um número real com duas casas decimais de precisão
  - 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%] 0x00
1: [HUF 42.86%] 0x9C 0x6B 0x50
2: [HUF 22.22%] 0x00 0x00
2: [RLE 22.22%] 0x09 0xFF
3: [HUF 25.00%] 0xC0
```