Compressão de dados

Eduardo Furlan Miranda

2024-02-08

Adaptado dos materiais dos Profs. J. F. de Souza, C. Santos, e A. da S. Freire

Compressão

- Reduzir custo de armazenagem
- Padrões de interoperabilidade e implementação em hardware
- Banda passante (ex.: dispositivos, Internet, etc)
- Tarifas pela quantidade de dados transmitidos
- Arquivos menores são mais rápidos para transferir
- Algoritmos com perda e sem perda

Repetição de série (*Run-Length Encoding* - RLE)

- Substitui uma sequência lona de símbolos por um código curto
  - Valor do símbolo
  - Quantidade de ocorrências
- Se é um arquivo binário, apenas conta as repetições
  - 10001110011111001101
- Aplicações
  - Codificação para facsimiles
  - Pode ser combinado com Huffman, p. ex.
  - Variantes RLE: TIFF, PCX, BMP, RLE(Windows 3.x)

Exemplo RLE

2/3 dos bits originais

Codificação de Huffman

- Algorítimo de compactação de dados sem perda
- A ideia é atribuir códigos de comprimento variável aos caracteres de entrada
- Os comprimentos dos códigos atribuídos são baseados nas frequências dos caracteres correspondentes
- Funciona bem para codificar símbolos separadamente
- Aplicações (e variações): Deflate/PKZIP, JPEG, MP3

Codificação de Huffman

- Usa caracteres (símbolos) com um nº variável de bits
  - Caracteres mais comuns na mensagem são codificados com menos bits e menos comuns com mais bits
- Árvore de prefixos de Huffman
  - Dada uma mensagem, encontra a melhor (menor) codificação
  - Algoritmo:
    - Tabular as frequências dos símbolos
    - Montar uma floresta de Tries unitários com símbolos e frequências
    - Repetir
      - Localizar 2 Tries com menor frequência F_i e F_j
      - Uni-los em um Trie único com frequência $F_i + F_j$

Exemplo

- Frequência dos caracteres
- Árvore de Huffman é um heap
  - Pode-se implementar utilizando um array, assim como é feito no HeapSort

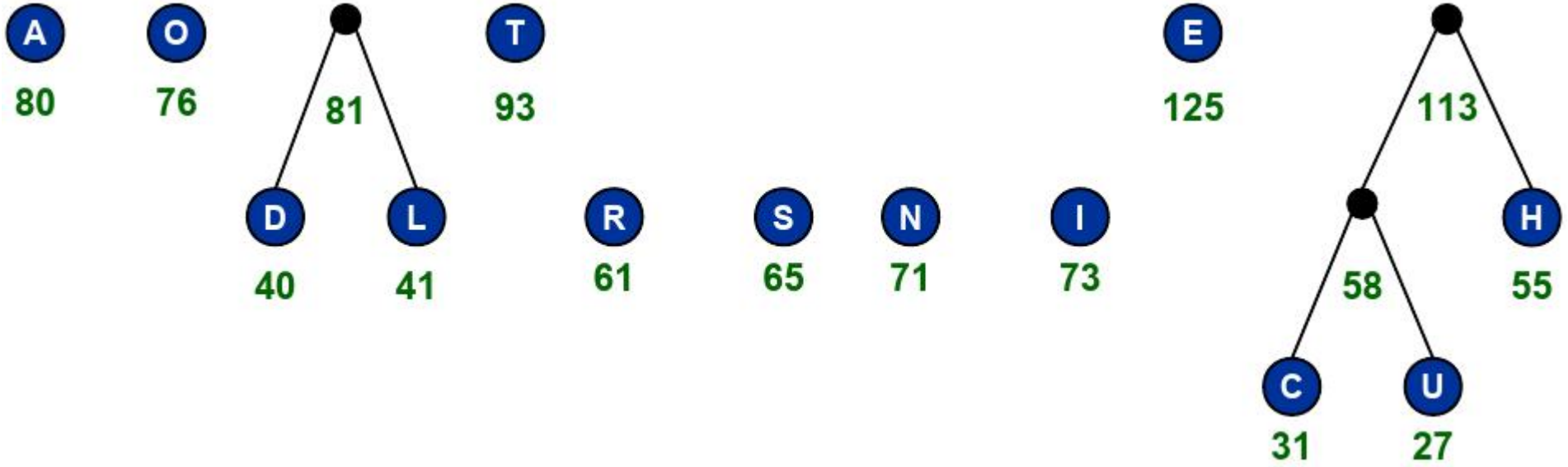
Heap: árvore binária onde cada nó pai é
- Menor ou igual ao seu nó filho (Min Heap), ou
- Maior ou igual ao seu nó filho (Max Heap)

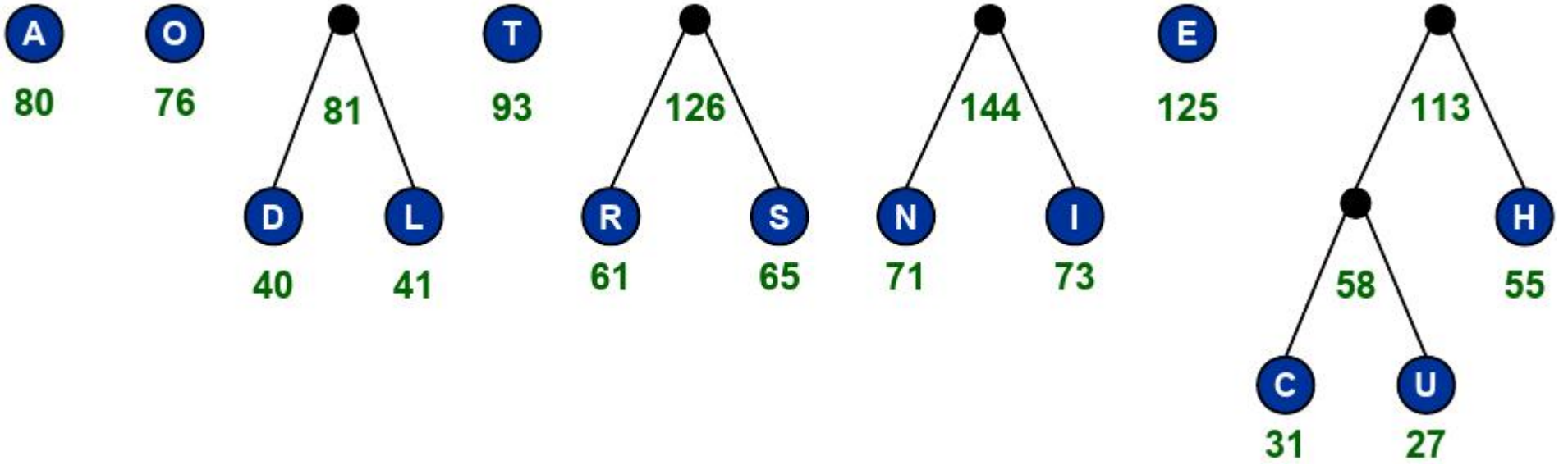
Char	Freq
E	125
T	93
A	80
O	76
I	72
N	71
S	65
R	61
H	55
L	41
D	40
C	31
U	27

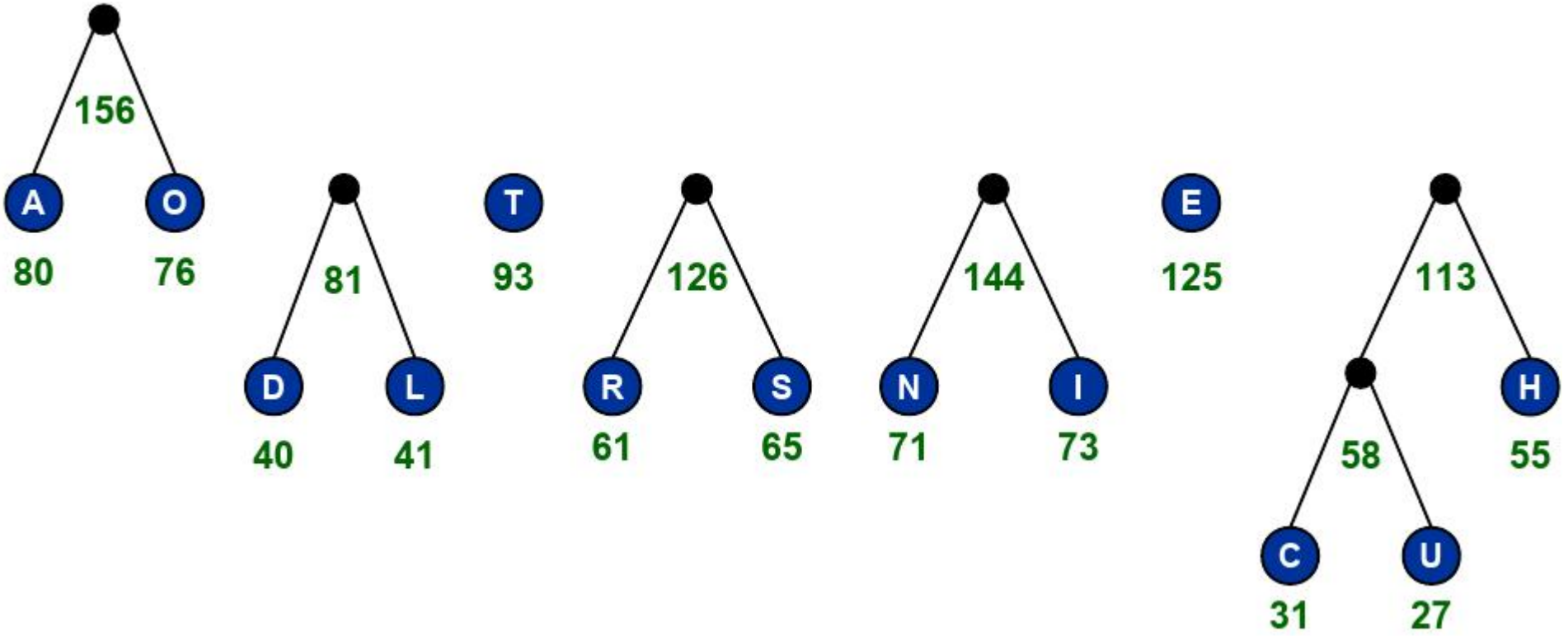
Começa colocando as frequências maiores em cima

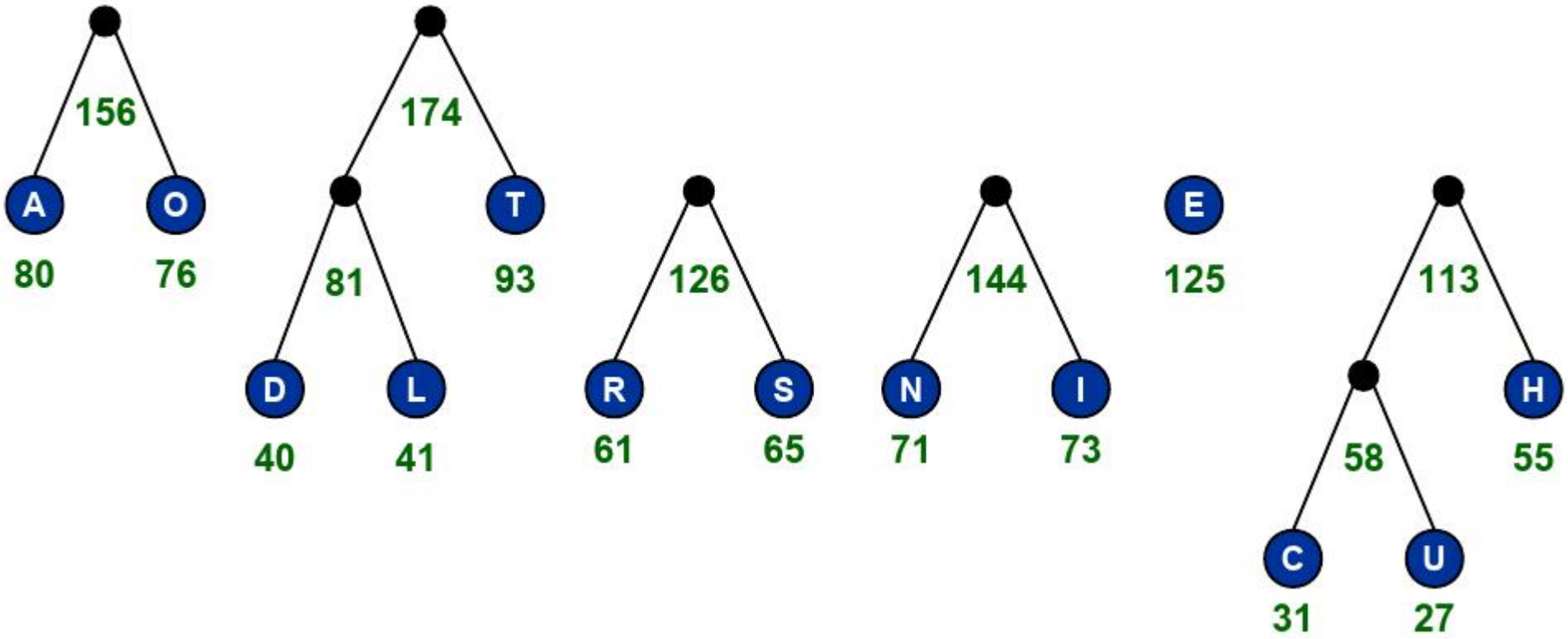
Depois pega “das de baixo” e vai montando a árvore

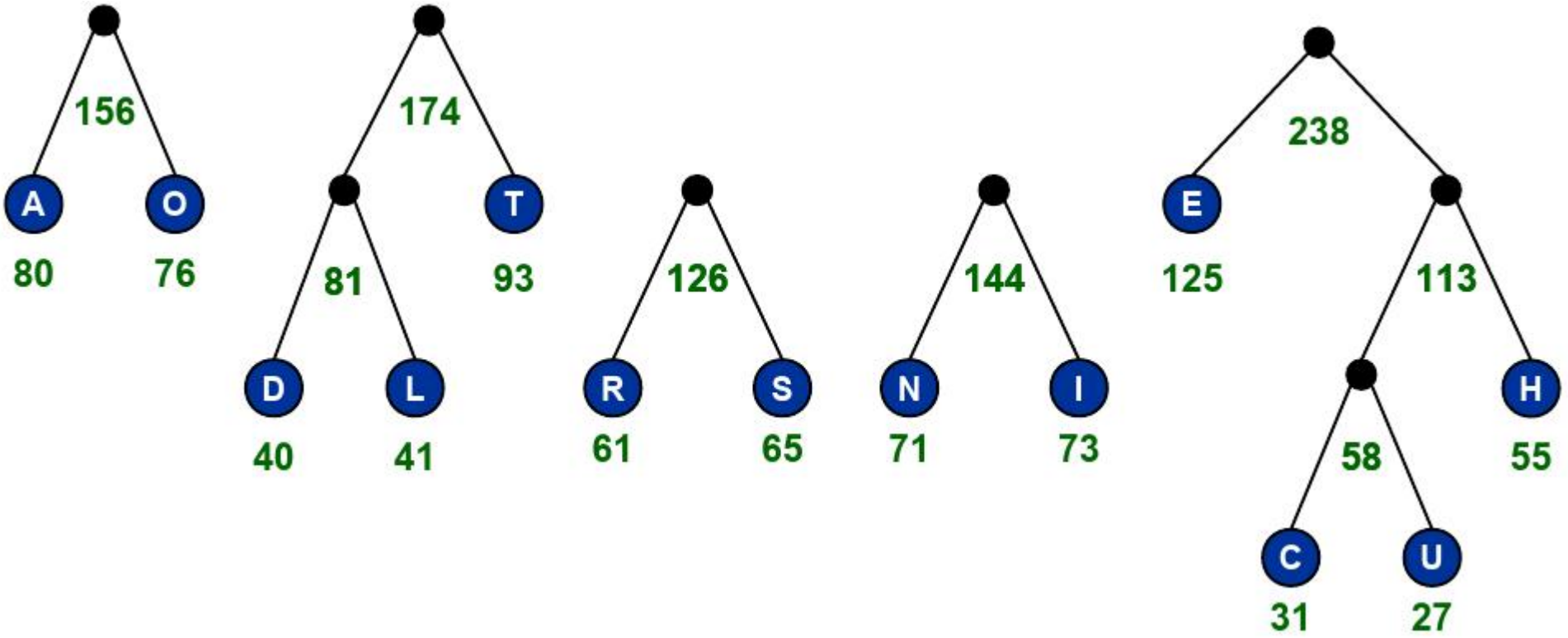

A
80
O
76
81
D
40
L
41
T
93
R
61
S
65
N
71
I
73
E
125
58
C
31
U
27
H
55

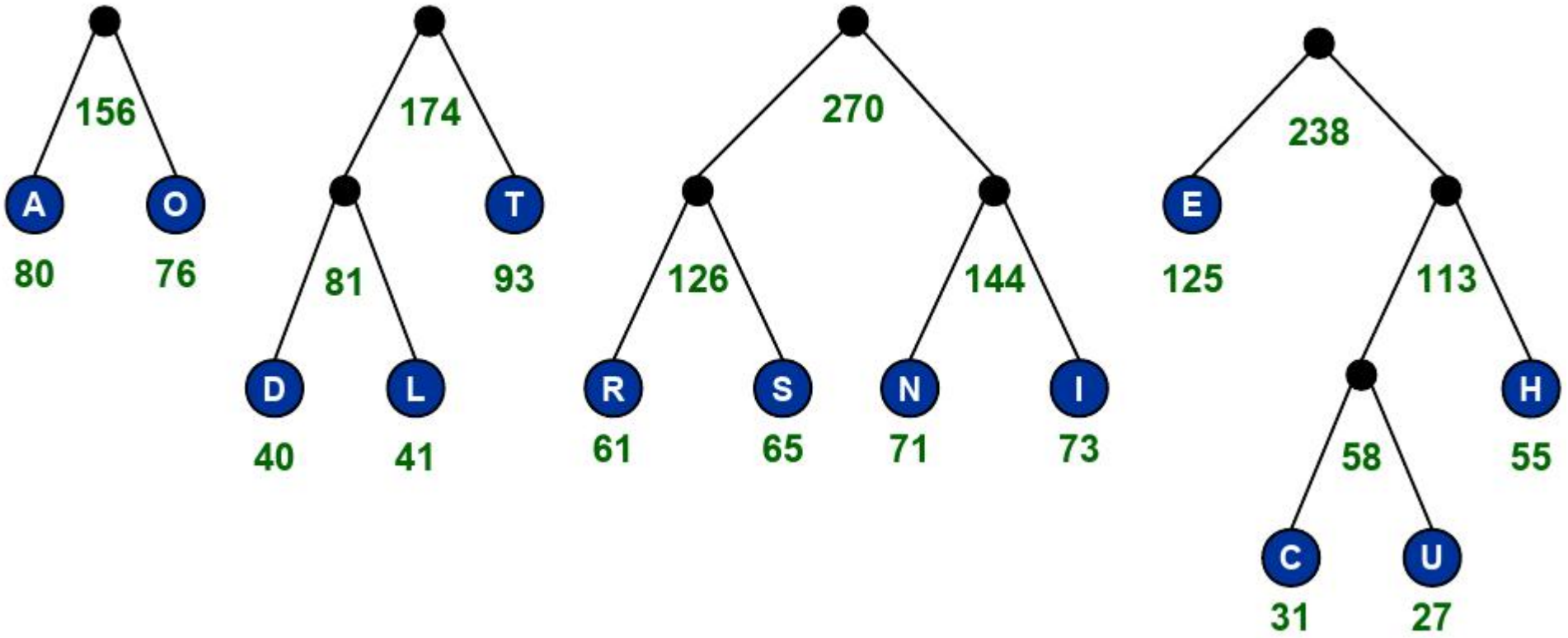

A
80
O
76
81
D
40
L
41
T
93
R
61
S
65
N
71
I
73
E
125
113
58
C
31
U
27
H
55

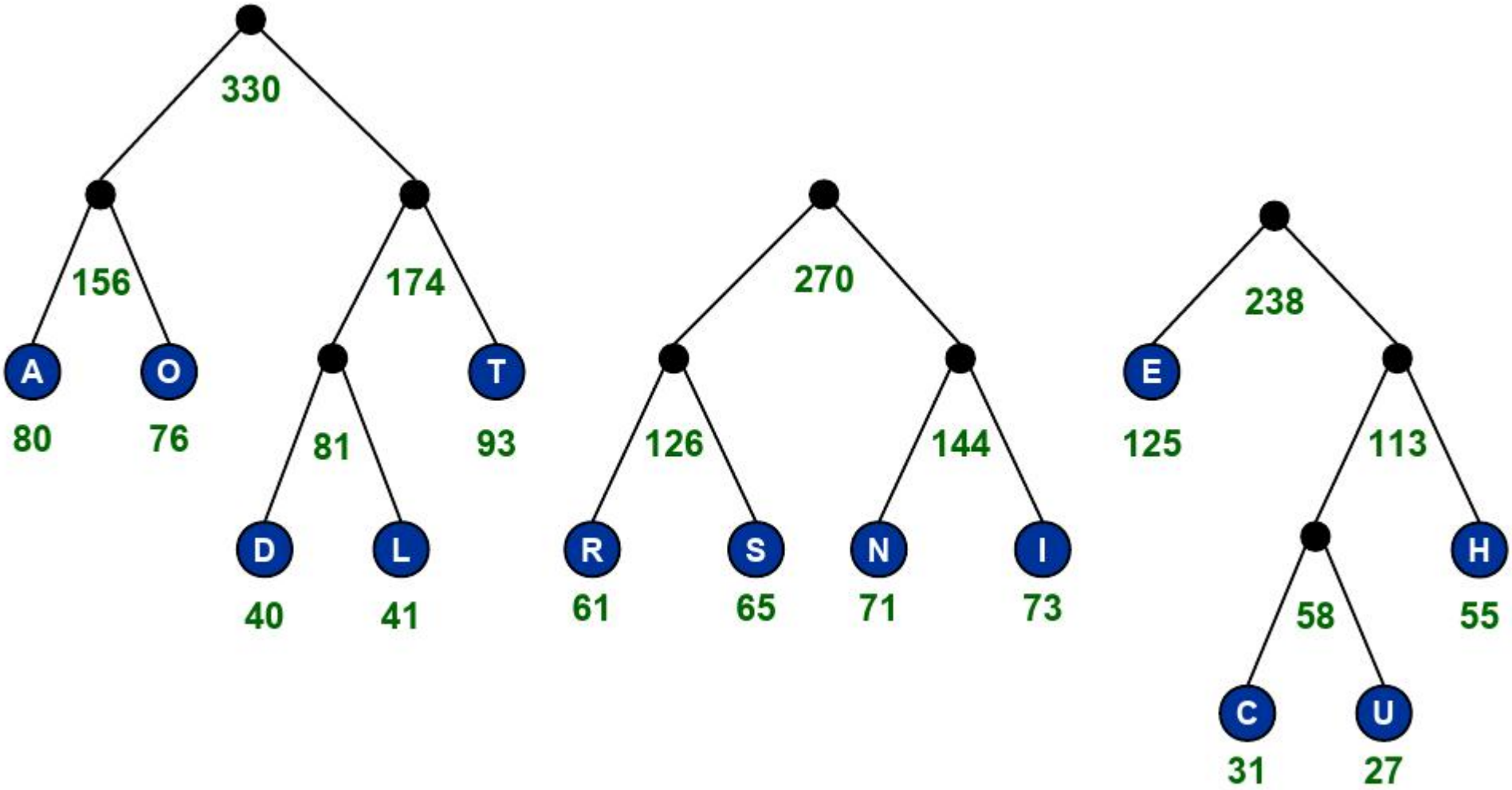

A
80
O
76
81
D
40
L
41
T
93
126
R
61
S
65
144
N
71
I
73
E
125
113
58
C
31
U
27
H
55

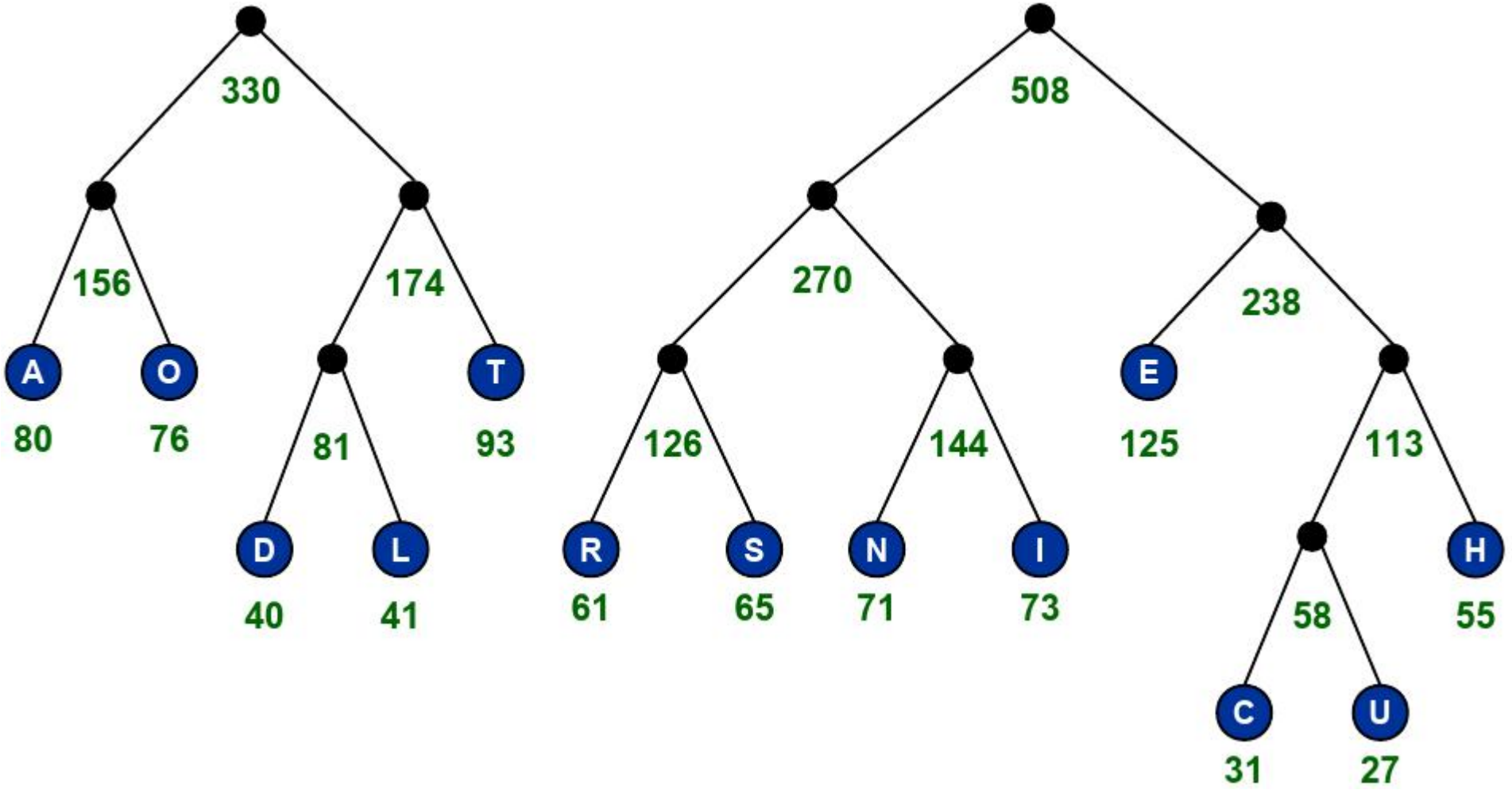

156
A
80
O
76
81
D
40
L
41
T
93
126
R
61
S
65
144
N
71
I
73
E
125
113
58
C
31
U
27
H
55

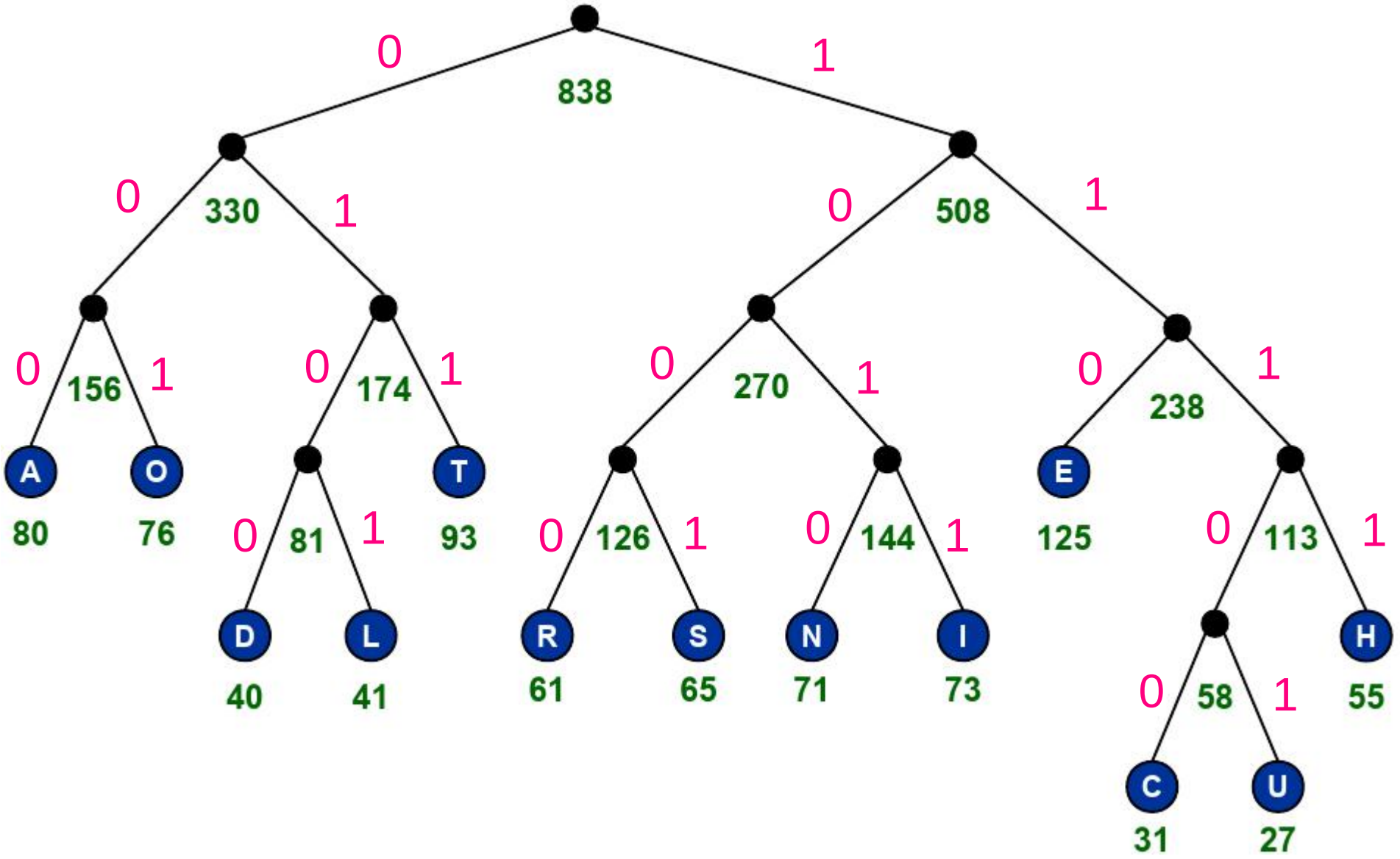

156
A
O
80
76
174
81
T
93
D
L
40
41
126
R
S
61
65
144
N
I
71
73
E
125
113
58
H
55
C
U
31
27


156
A
O
80
76
174
81
T
93
D
L
40
41
126
R
S
61
65
144
N
I
71
73
238
E
125
113
58
H
55
C
U
31
27


156
A
O
80
76
174
81
T
93
D
L
40
41
270
126
144
R
S
N
I
61
65
71
73
238
E
125
113
58
H
55
C
U
31
27


330
156
174
A
O
T
80
76
81
93
D
L
40
41
270
126
144
R
S
N
I
61
65
71
73
238
E
125
113
H
55
58
C
U
31
27


330
156
174
A
O
T
80
76
81
93
D
L
40
41
508
270
238
126
144
E
113
125
R
S
N
I
H
61
65
71
73
58
55
C
U
31
27


0
838
1
0
330
1
0
508
1
0
156
1
0
174
1
0
270
1
0
238
1
A
O
T
E
80
76
93
125
0
81
1
0
126
1
0
144
1
0
113
1
D
L
R
S
N
I
H
40
41
61
65
71
73
55
0
58
1
C
U
31
27

Frequência

125/838=0,15

Dígitos fixos

Dígitos Huffman

multiplicação

Char	Freq	Fixo	Huff					
E	125	0000	110	0,15	4	0,60	3	0,45
T	93	0001	011	0,11	4	0,44	3	0,33
A	80	0010	000	0,10	4	0,38	3	0,29
O	76	0011	001	0,09	4	0,36	3	0,27
I	73	0100	1011	0,09	4	0,35	4	0,35
N	71	0101	1010	0,08	4	0,34	4	0,34
S	65	0110	1001	0,08	4	0,31	4	0,31
R	61	0111	1000	0,07	4	0,29	4	0,29
H	55	1000	1111	0,07	4	0,26	4	0,26
L	41	1001	0101	0,05	4	0,20	4	0,20
D	40	1010	0100	0,05	4	0,19	4	0,19
C	31	1011	11100	0,04	4	0,15	5	0,18
U	27	1100	11101	0,03	4	0,13	5	0,16
Total	838							
				Totais		1,00		4,00		3,62

Huffman adaptativo

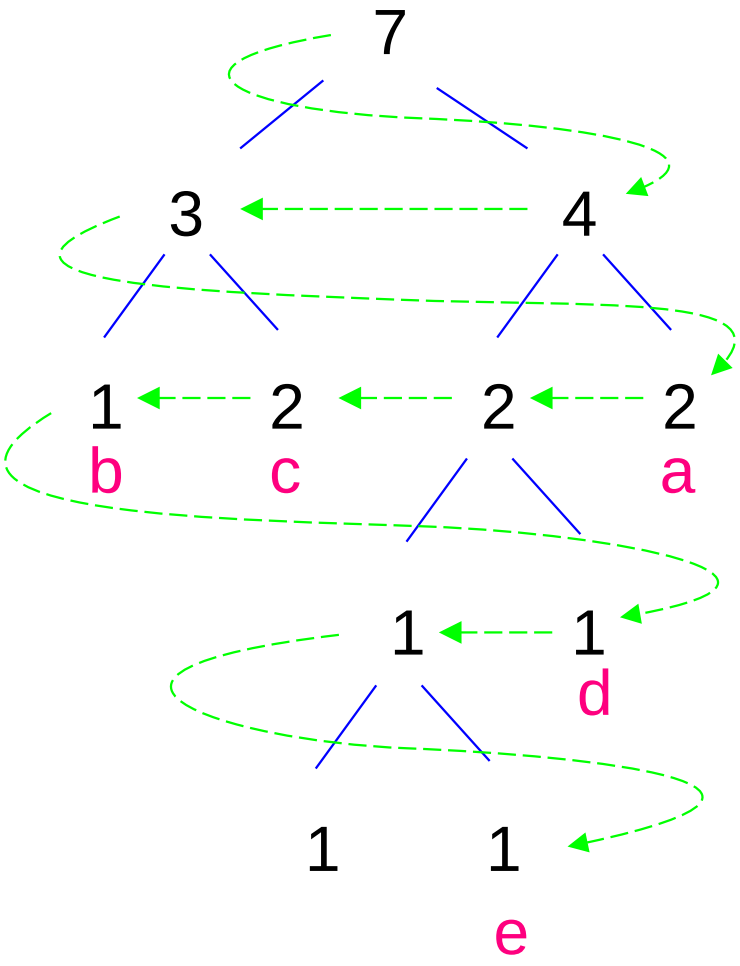
- O algoritmo foi desenvolvido independentemente por Faller (1973) e Gallager (1978) e melhorado por Knuth (1985), ficando conhecido como algoritmo FGK
- Existe ainda outro melhoramento feito por Vitter (1987), conhecido também como algoritmo V
- O Huffman adaptativo foi criado para que se pudesse usar a compressão em casos nos quais não se conhecem as frequências de ocorrência de cada caractere no texto

Huffman adaptativo

- A árvore é construída durante a compressão/descompressão
- Inicialmente não precisa da tabela de frequência
- Ajusta a árvore conforme os dados vão sendo transmitidos
- Algoritmo
  - Reservar um símbolo temporário na árvore para representar os que ainda não ocorreram
  - Atualizar os pesos de todos os nós da árvore com o respectivo número de ocorrências (permite obter a estatística da fonte)
  - Atualizar a árvore de forma que todos os nós tenham pesos ordenados em ordem decrescente de cima para baixo, e da direita para a esquerda

Huffman adaptativo

- A idéia do FGK é baseada na propriedade de irmandade
  - Se cada nó tem um irmão (exceto a raiz), e
  - Caminhando na árvore de *cima para baixo* e da *direita para a esquerda* gera uma lista de nós com contadores de frequência decrescentes (ou iguais)
    - Também é possível caminhar no sentido contrário, com contadores crescentes

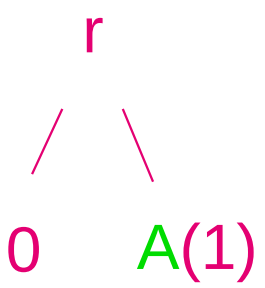


Huffman adaptativo

- O algoritmo verifica se a propriedade de irmandade está sendo violada
  - Em caso positivo, deve-se reestruturar a árvore
  - Para reestruturar basta fazer a troca entre os nós adjacentes
- A árvore Huffman é inicializada com um único nó, conhecido como Not-Yet-Transmited (NYT) ou código de escape
- Este código será enviado toda vez que um novo caractere, que não esteja na árvore, for encontrado, seguido da codificação ASCII do caractere
  - Isso permite que o descompressor distinga entre um código e um novo caractere

Exemplo: ABRACADABRA → “A”

- Primeira ocorrência de “A”



  - ‘r’ → nó raiz
  - ‘0’ → nó nulo (onde outros nós podem ser adicionados)
  - ‘A(1)’ → ocorrência de “A” com a frequência “1”

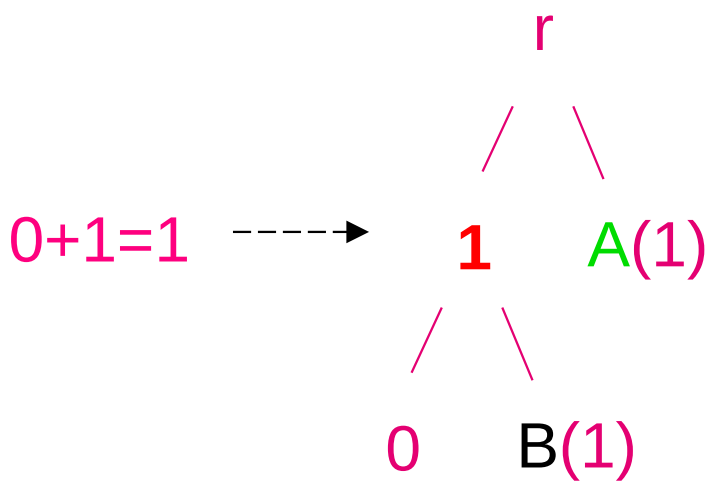
- Ordem: 0,A(1)

  Neste caso, de baixo para cima, da esquerda para a direita, em ordem crescente

  Poderia ser também decrescente, no sentido inverso

Exemplo: ABRACADABRA → “AB”

- Primeira ocorrência de “B”



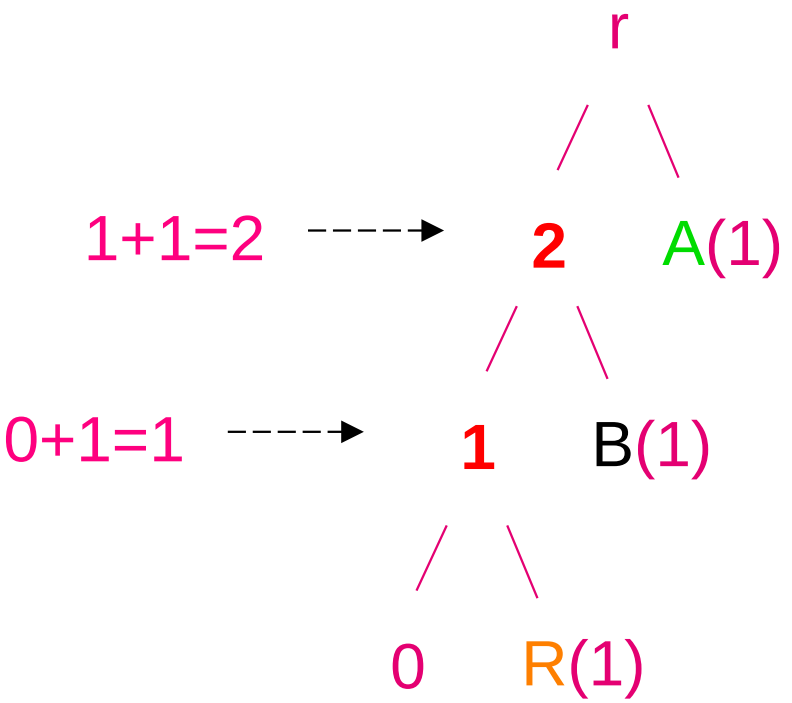
- Ordem: 0,B(1),1,A(1)

Considerar o índice entre parêntesis, para ordenar

Não considerar o caractere em si

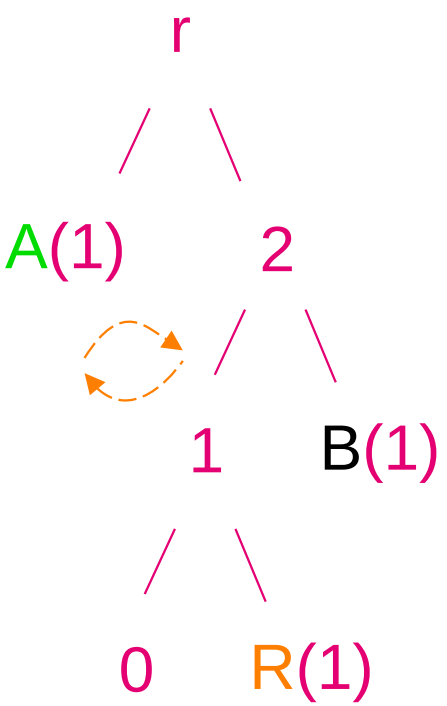
ABRACADABRA → “ABR”

- Primeira ocorrência de “R”



- Ordem: 0,R(1),1,B(1),2,A(1)
  - Está fora de ordem → vamos trocar as subárvores

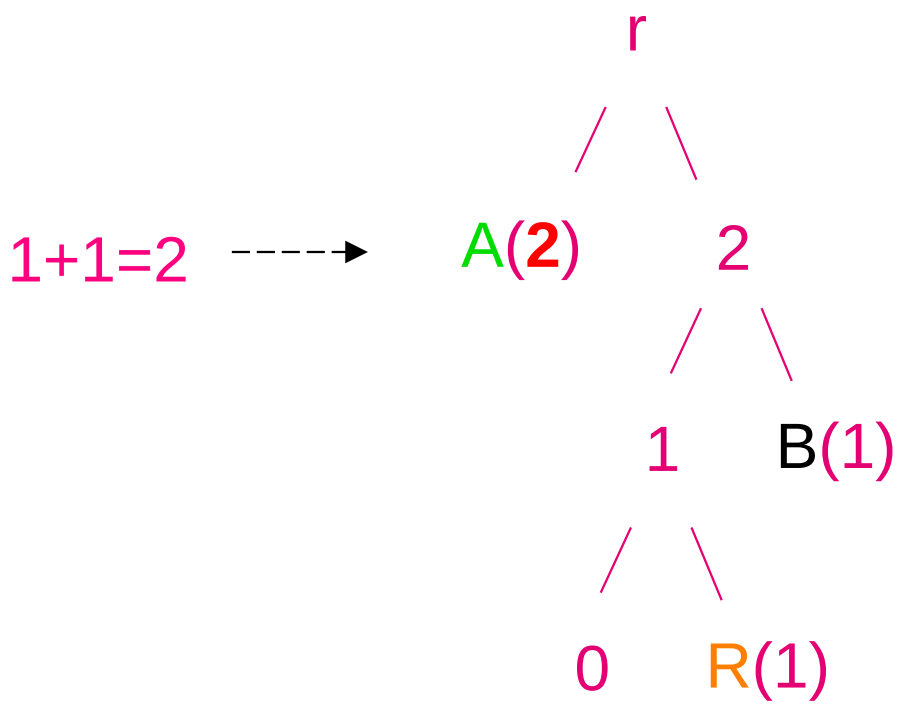
Reordenar “ABR”



- Ordem: 0,R(1),1,B(1),A(1),2

ABRACADABRA → “ABRA”

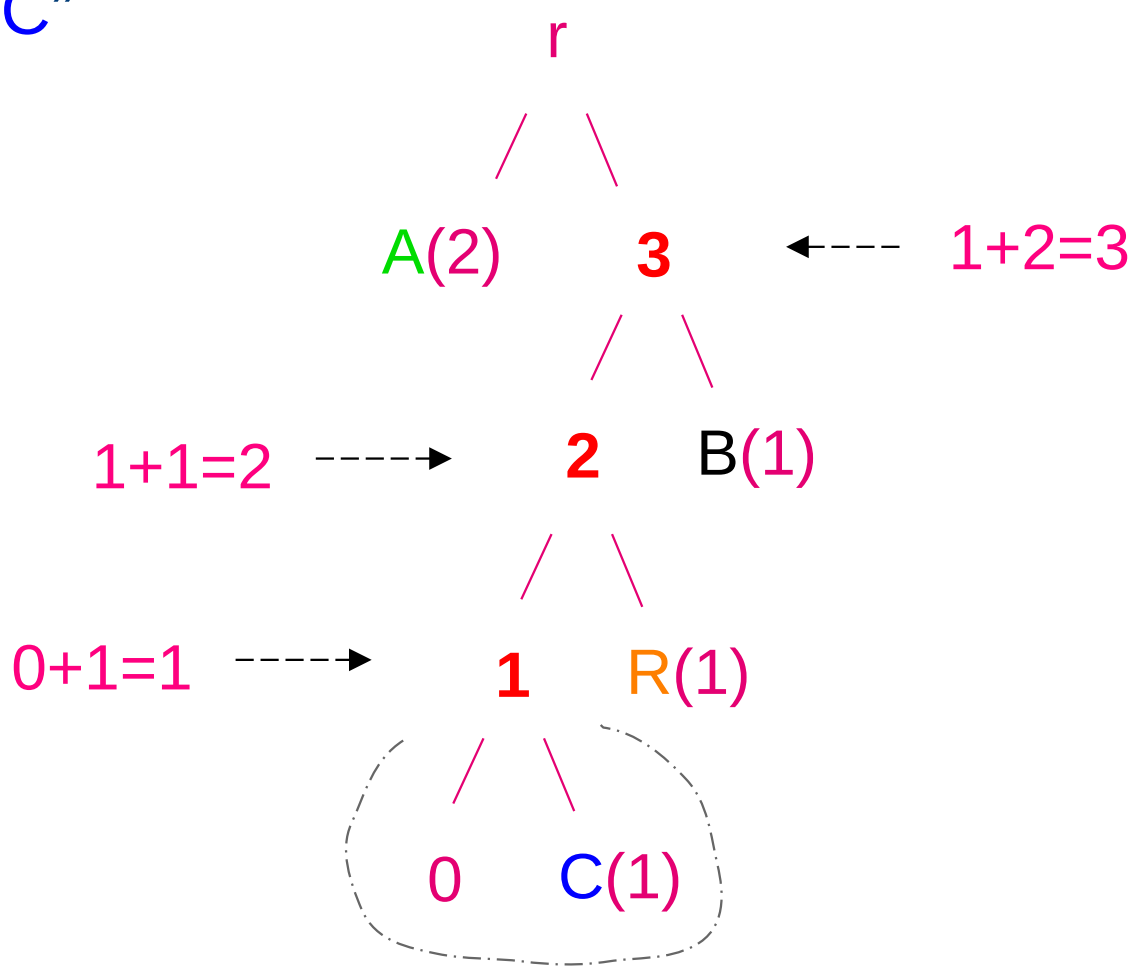
- Inserir “A” (existente)



- Ordem: 0,R(1),1,B(1),A(2),2

ABRACADABRA → “ABRAC”

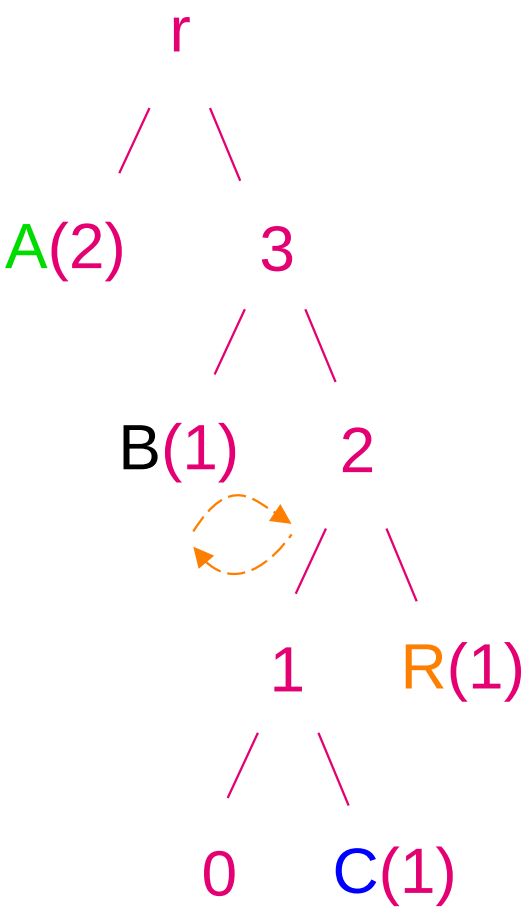
- Primeira ocorrência de “C”



- Ordem: 0,C(1),1,R(1),2,B(1),A(2),3

Fora de ordem

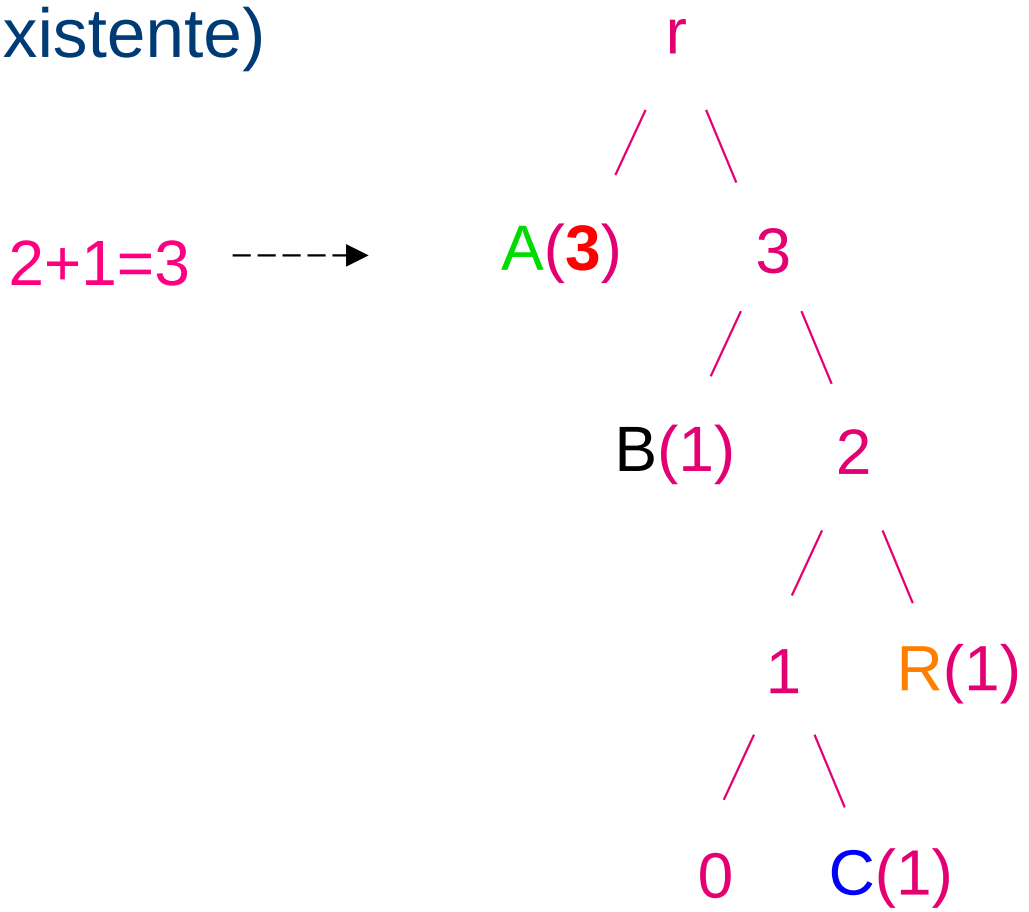
Reordenar



- Ordem: 0,C(1),1,R(1),B(1),2,A(2),3

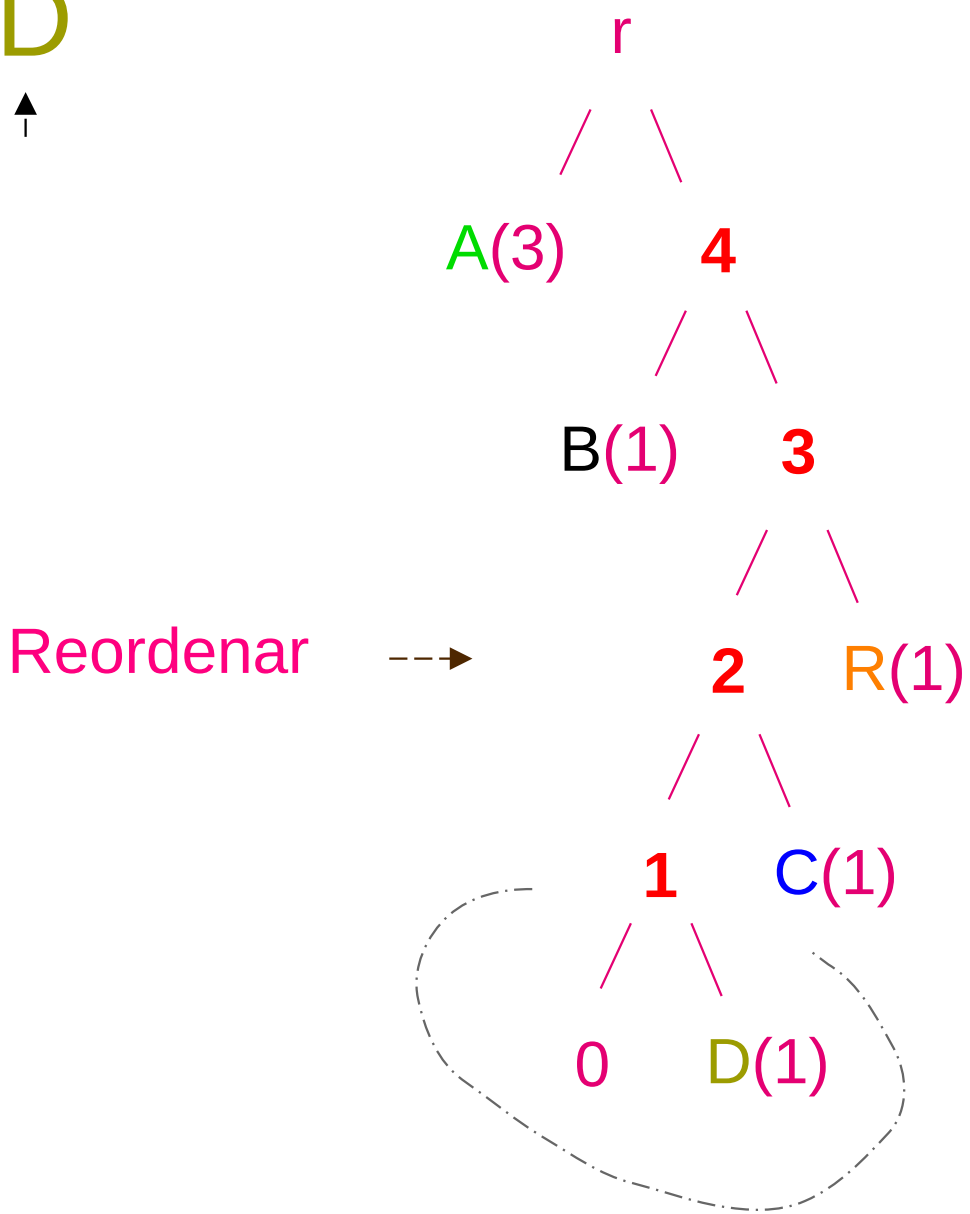
ABRACADABRA → “ABRACA”

- Inserir “A” (existente)



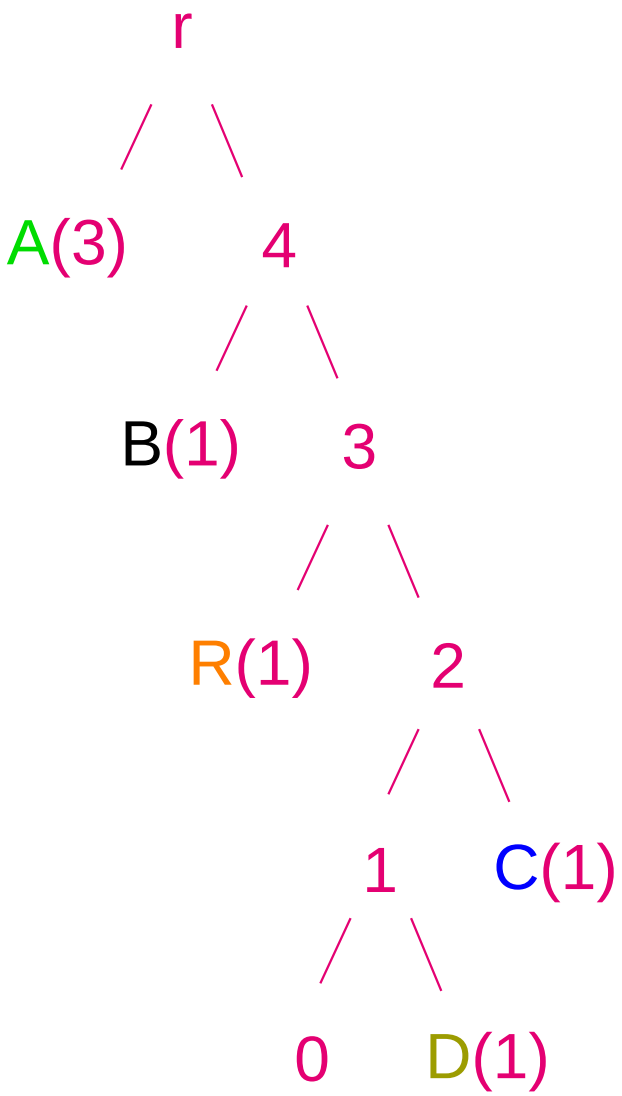
- Ordem: 0,C(1),1,R(1),B(1),2,A(3),3

ABRACAD



- Ordem: 0,D(1),1,C(1),2,R(1),B(1),3,A(3),4

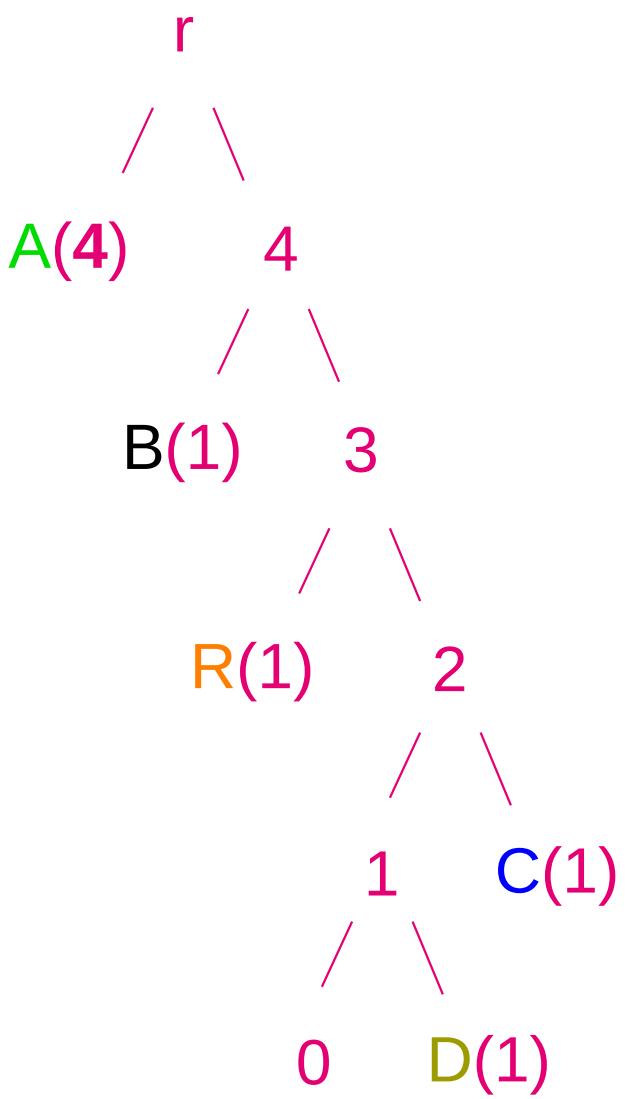
Reordenar



- Ordem: 0,D(1),1,C(1),R(1),2,B(1),3,A(3),4

ABRACADA

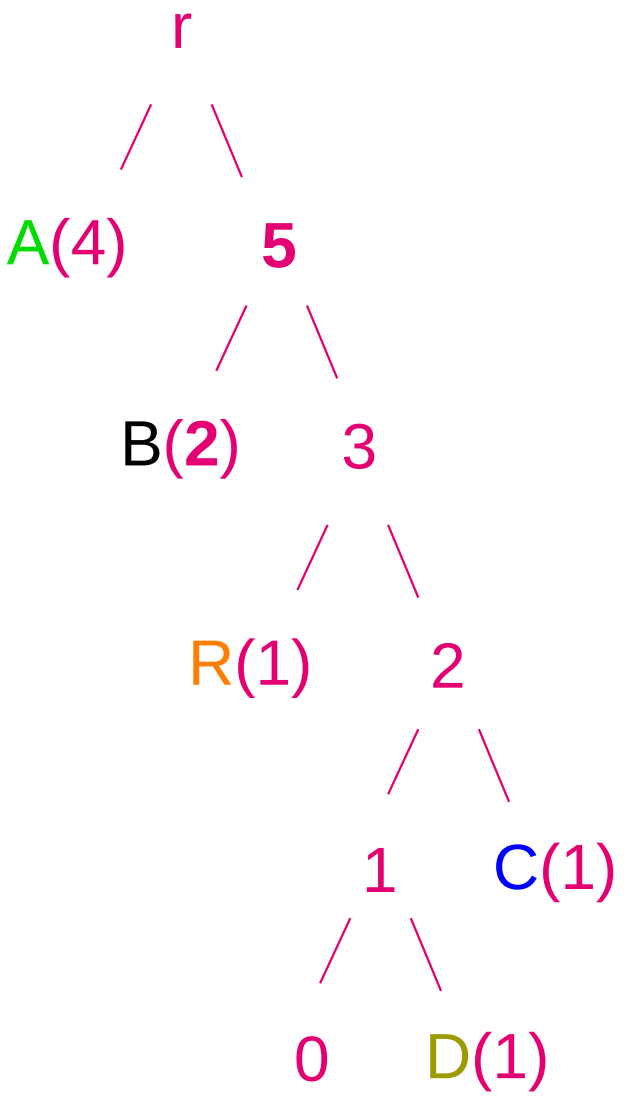
- Inserir “A” (existente)



- Ordem: 0,D(1),1,C(1),R(1),2,B(1),3,A(4),4

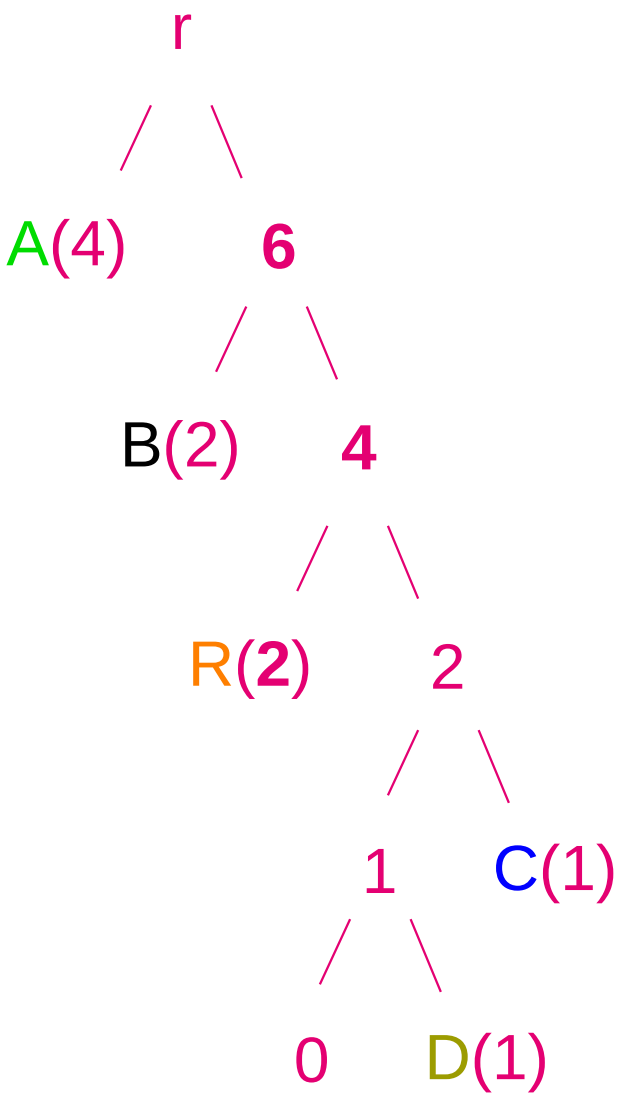
ABRACADAB

- Inserir “B” (existente)



- Ordem: 0,D(1),1,C(1),R(1),2,B(2),3,A(4),5

ABRACADABR

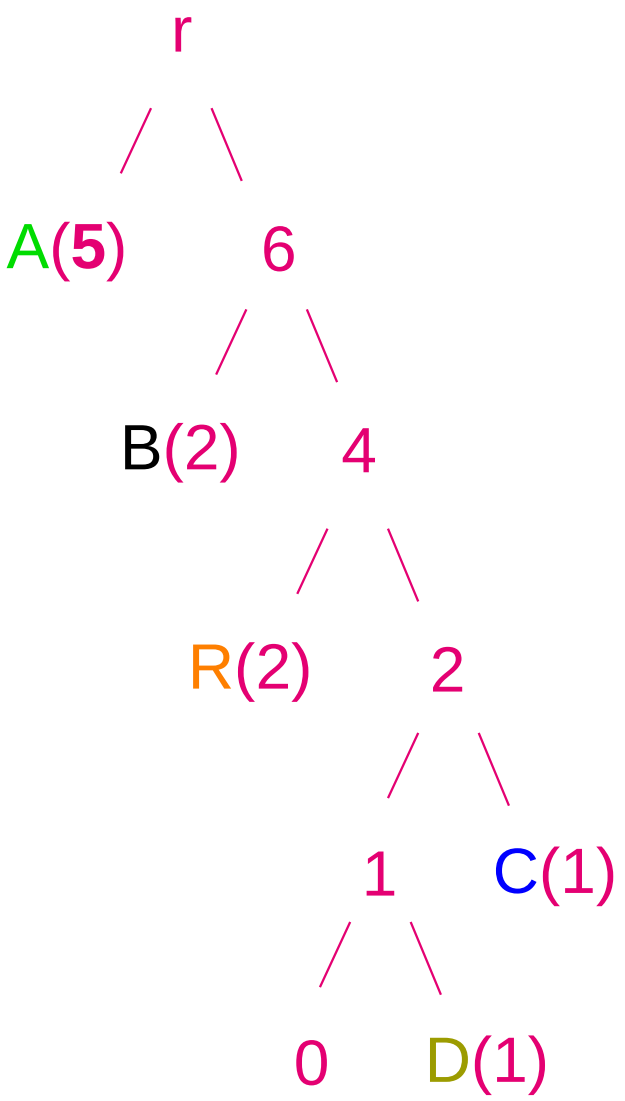


- Inserir “R” (existente)

- Ordem: 0,D(1),1,C(1),R(2),2,B(2),4,A(4),6

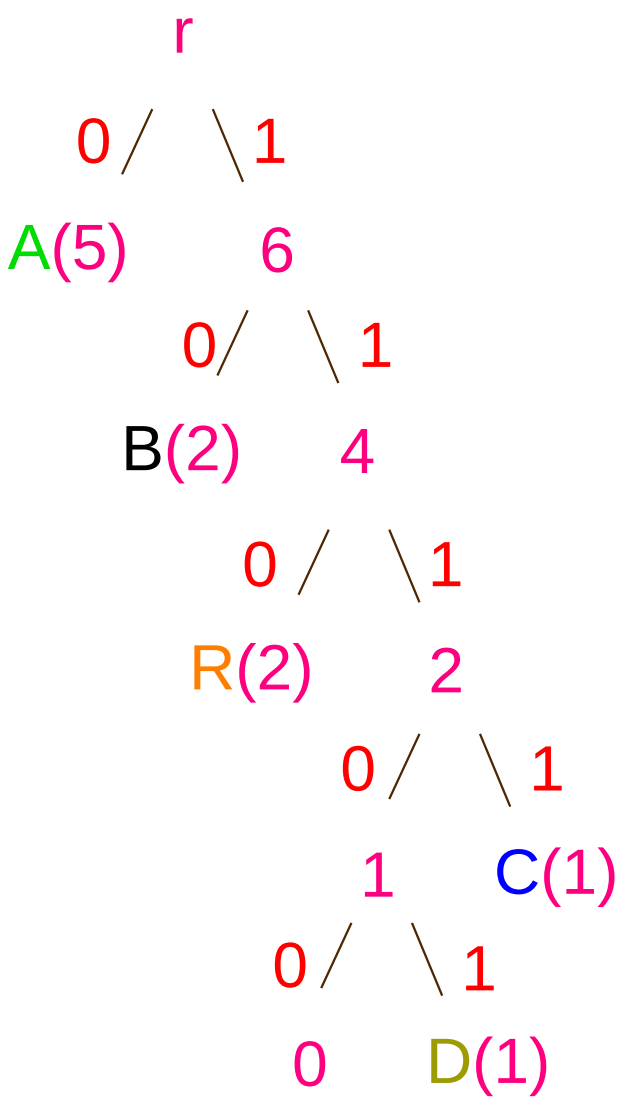
ABRACADABRA

- Inserir “A” (existente)



- Ordem: 0,D(1),1,C(1),R(2),2,B(2),4,A(5),6

Codificação



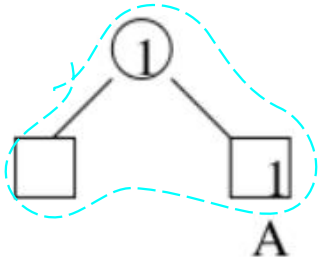
Letra
Frequência
Código
Comprimento

L	F	Código	C
A	5	0	1
B	2	10	2
R	2	110	3
C	1	1111	4
D	1	11101	5

- ABRACADABRA: 0 10 110 0 1111 0 11101 0 10 110 0

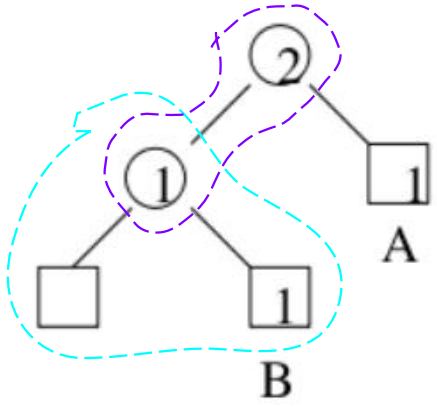
Exemplo - resumo

Insere A e incrementa



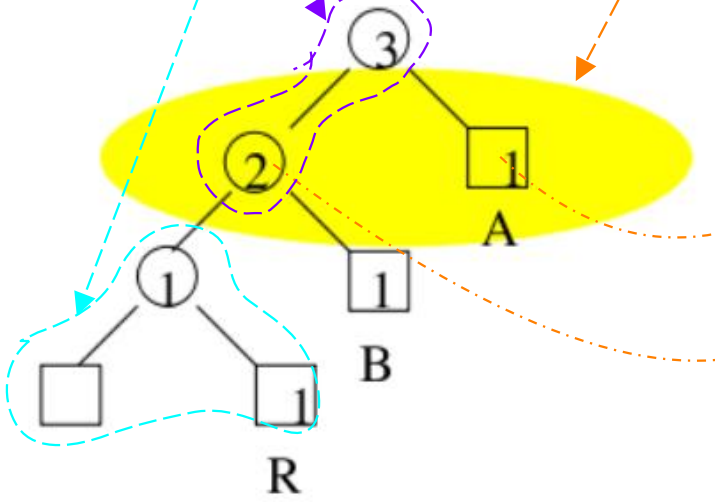
Bloco

Insere B e incrementa

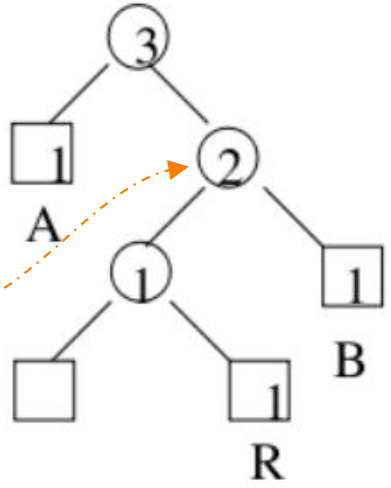


Insere R e incrementa

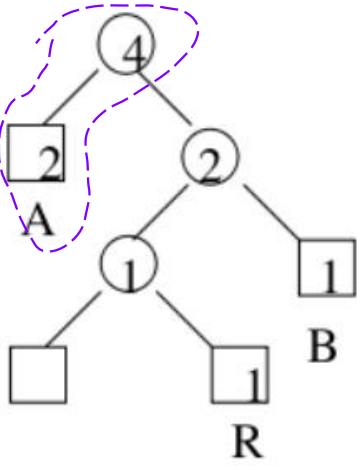
violação



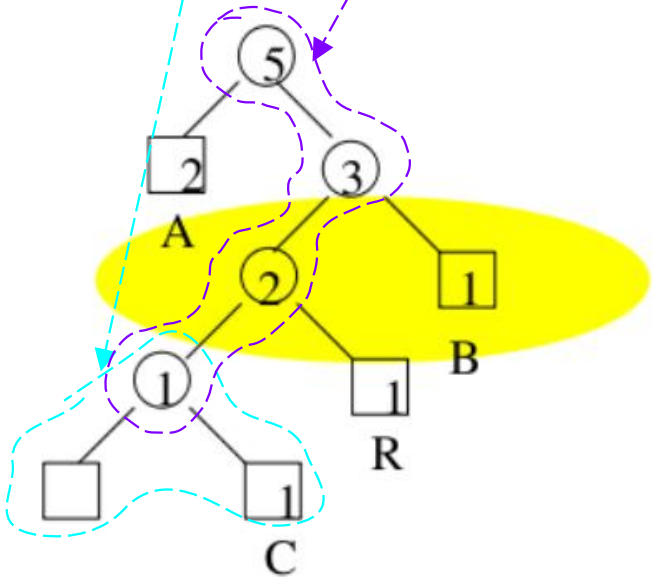
Atualiza a árvore



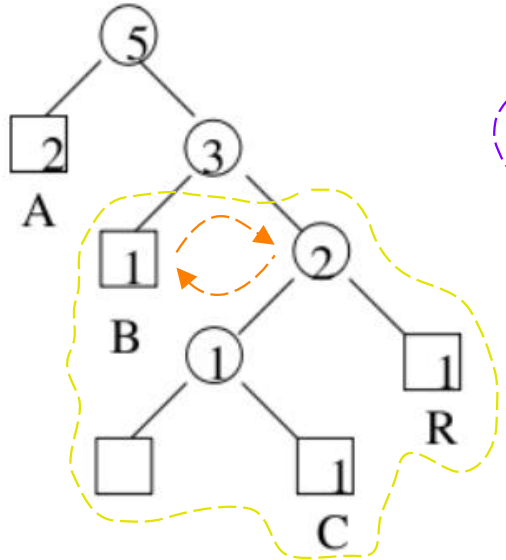
Insere A e incrementa



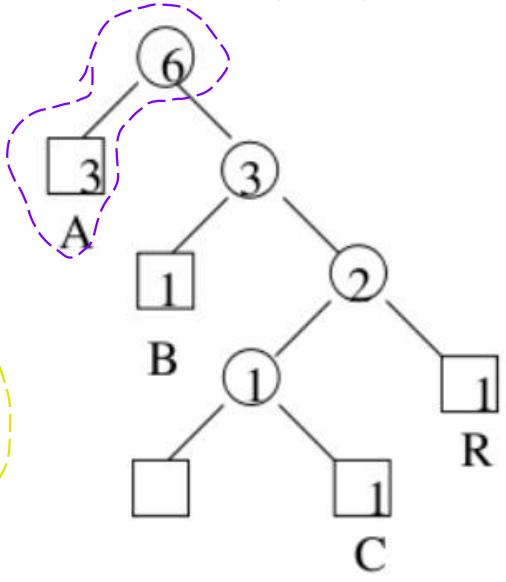
Insere C e incrementa



Atualiza a árvore

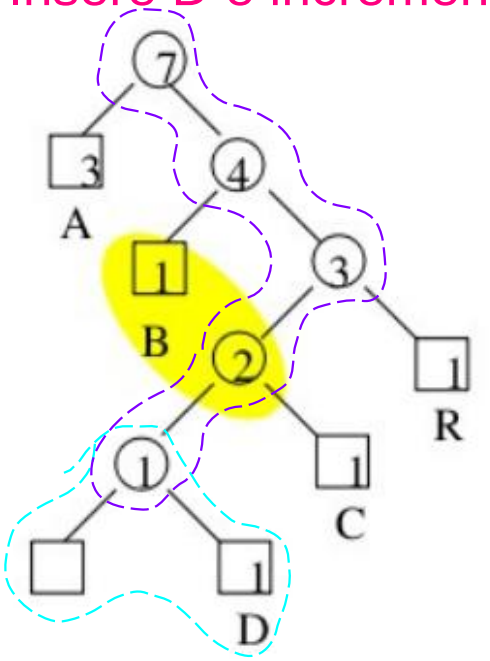


Insere A e incr.

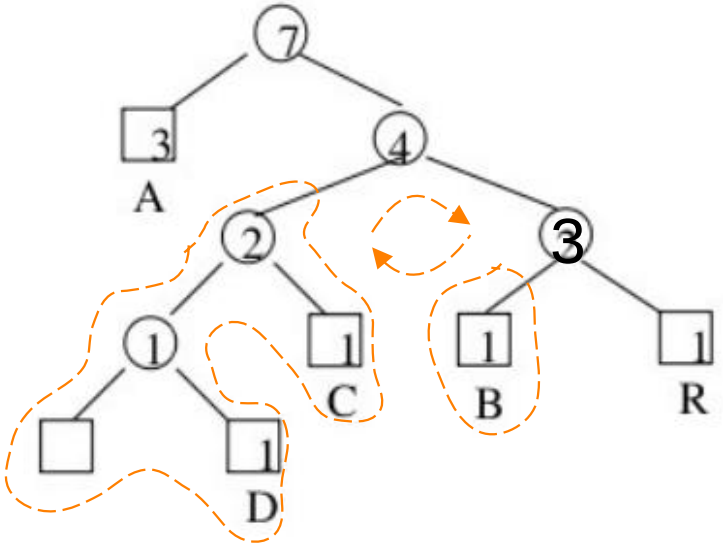


Exemplo - resumo

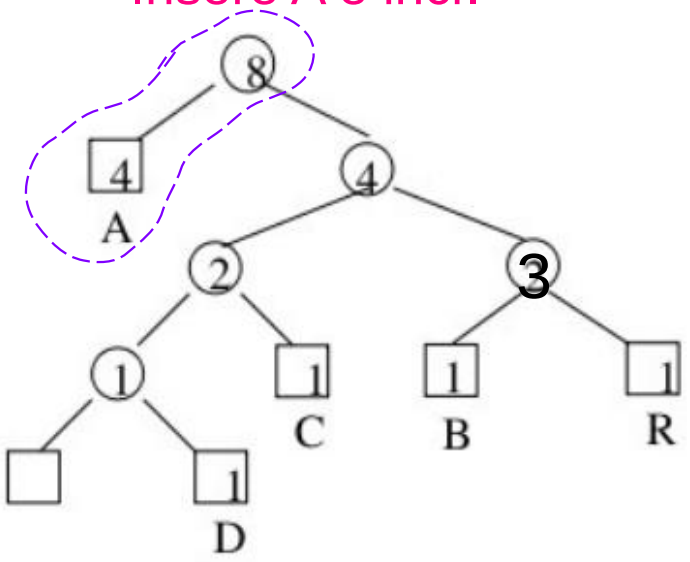
Insere D e incrementa



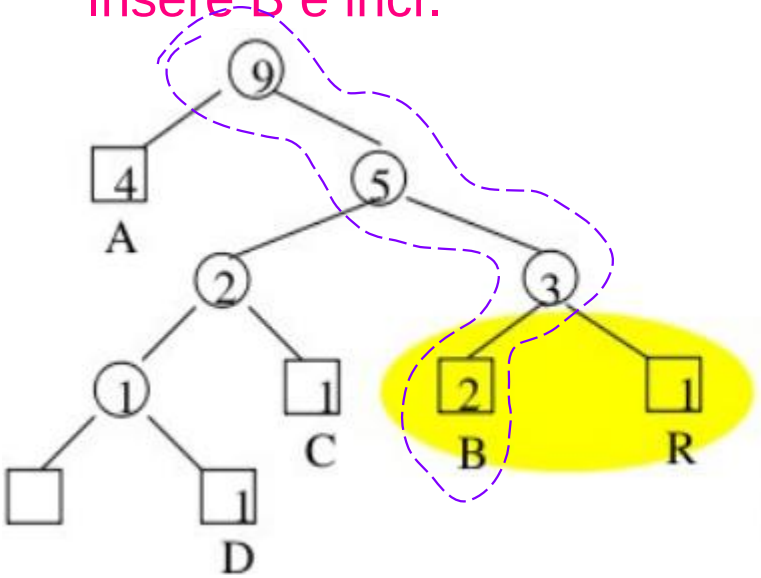
Atualiza a árvore



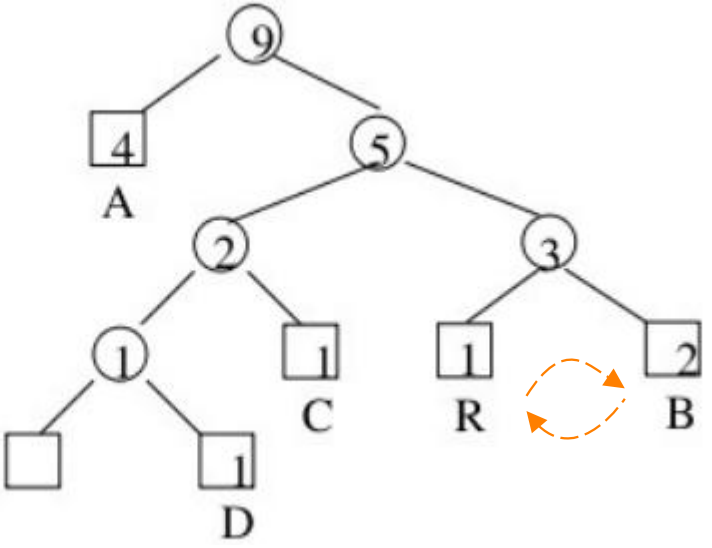
Insere A e incr.



Insere B e incr.



Atualiza a árvore



etc...

Huffman adaptativo

- Código gerado para “ABRACADABRA”:
  - 0 10 110 0 1111 0 11101 0 10 110 0
  - A B R A C A D A B R A
- Para entender o código, é preciso verificar o passo a passo da construção da árvore
- Cada letra significa uma inserção na árvore
- Bits que antecedem a letra indicam a posição do nó de escape
  - Sempre que for enviada uma letra não pertencente à árvore
    - É enviado os bits correspondente ao nó de escape e,
    - Em seguida, a letra
- Ao decodificar, a árvore final deverá ser idêntica

Codificação por Dicionário

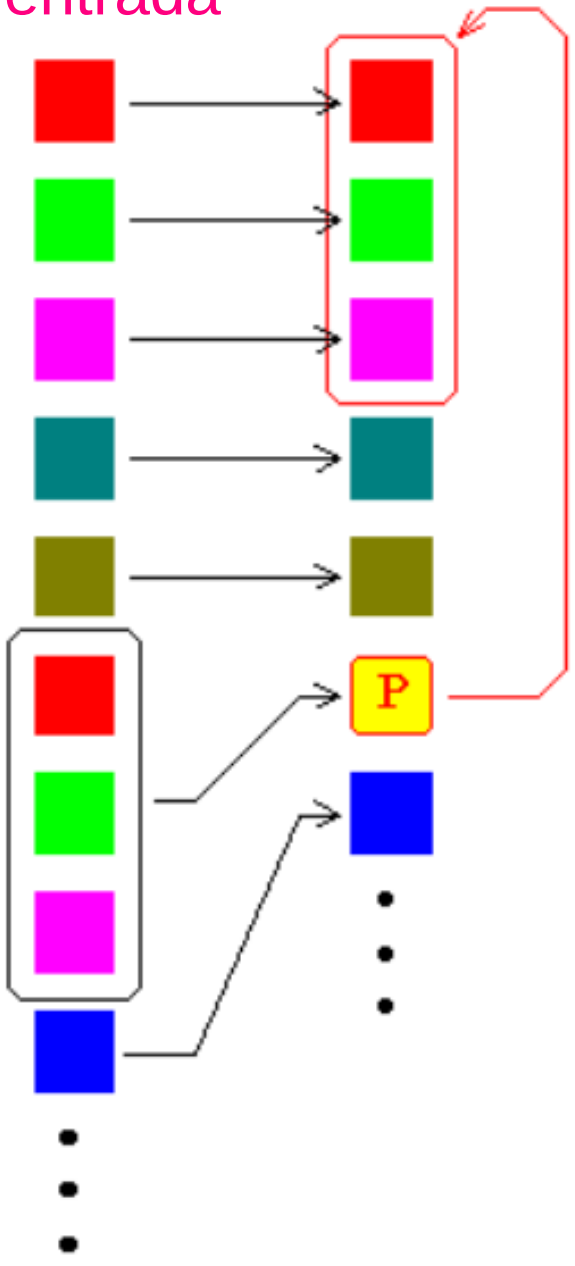
- Lempel e Ziv (1970) : LZ77 e LZ78
- Símbolos são lidos e comparados com entradas de um dicionário (tabela hash)
  - Se o símbolo existe, a posição da tabela é gerada como saída
- O resultado da compressão é o próprio dicionário combinado com os dados
- Os dados podem ser comprimidos “on-the-fly” sem a necessidade de geração de tabelas e de maneira dinâmica
- Variantes implementados no zip/unzip, gzip, e outros

Métodos de dicionário: 2 grupos

- 1o. grupo:
  - 1. Verifica se a sequência que deve ser comprimida já apareceu anteriormente
  - 2. Ao invés de repeti-la, define um ponteiro para essa ocorrência na própria sequência
- 2o. grupo:
  - 1. Cria um dicionário de “frases” que ocorrem na sequência de entrada
  - 2. Quando a frase é novamente encontrada, simplesmente o índice da frase no dicionário é codificada

Exemplos

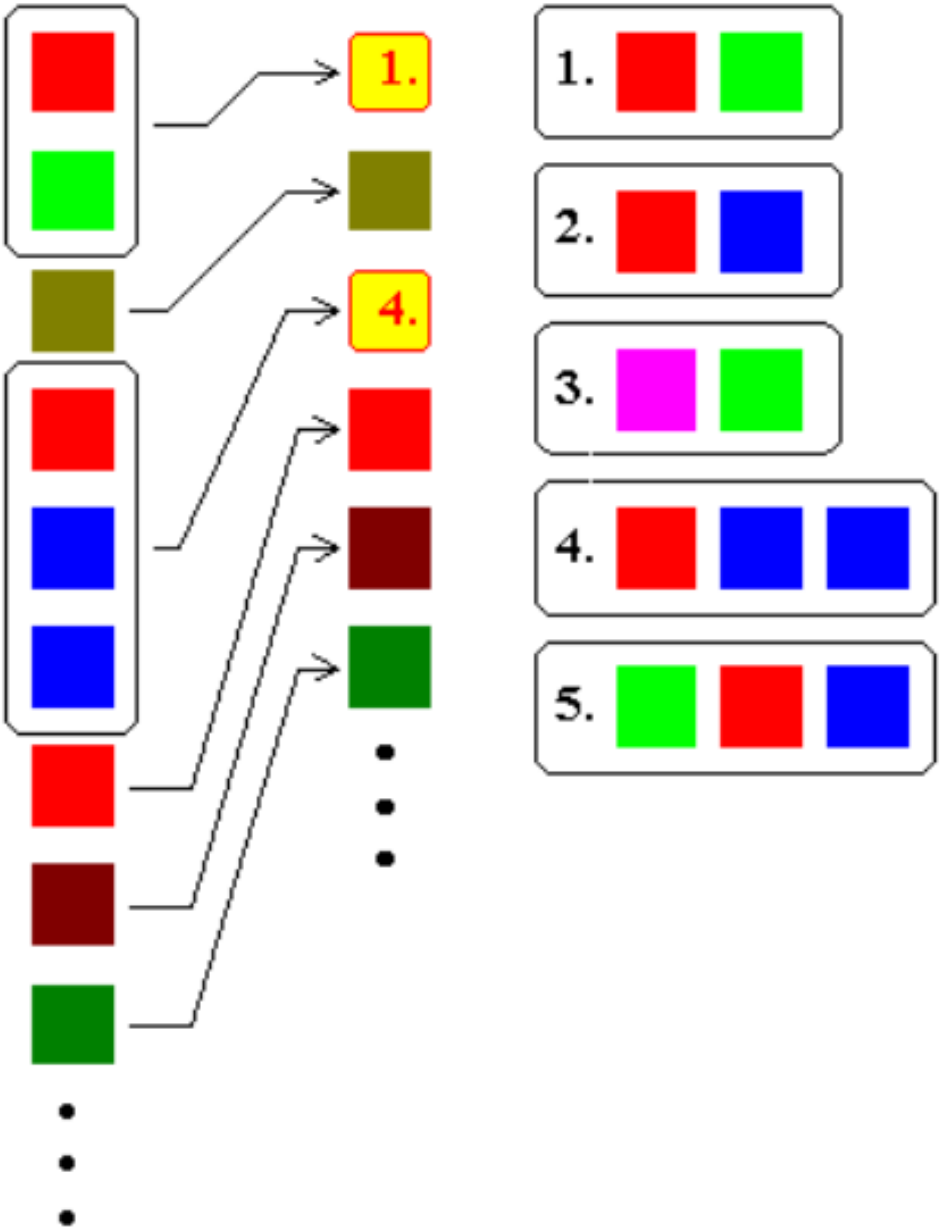
Fluxo de entrada



Fluxo de entrada

Fluxo de saída

Dicionário



Exemplo - LZ78

- Considere um arquivo binário com o conteúdo:

```
00010111000100000010011
```

- Basicamente a ideia é particionar a cadeia de bits em trechos
- De maneira que cada trecho corresponda à menor subsequência que não apareceu entre os trechos encontrados anteriormente

```
0|00|1|01|11|000|10|0000|001|0011
```

Algoritmo compressor

- Os trechos são numerados sequencialmente
- É definido um trecho vazio no início da cadeia, com índice “0”
- No exemplo anterior o trecho “0” recebe o índice 1
  - O trecho “00” recebe o índice 2
  - E assim por diante
- Cada trecho T é então associado a um par
  - Composto pelo índice do trecho dentre os anteriores que corresponde ao prefixo de T, e
  - O bit (0 ou 1) que finaliza T

Índice do trecho	1	2	3	4	5	6	7	8	9	10
Subsequência	0	00	1	01	11	000	10	0000	001	0011
Prefixo	-	0	-	0	1	00	1	000	00	001
(Índice do Prefixo, Bit Final)	(0,0)	(1,0)	(0,1)	(1,1)	(3,1)	(2,0)	(3,0)	(6,0)	(2,1)	(9,1)

- A sequência de pares obtida é a “codificação” da cadeia de bits
- A codificação deve ser convertida para a base binária
- À medida que avançamos na cadeia, os pares representarão subsequências cada vez mais longas, o que tende a tornar a codificação mais compacta do que a cadeia que a gerou

Representação binária da codificação

- Para produzir o arquivo binário de saída deve-se converter os índices de prefixo de cada par, de base decimal para binária
- O número de bits usados ao se escrever o índice do prefixo do n-ésimo par em binário corresponde ao número de bits necessários para representar o número n-1 em binário
  - Que é o maior valor que o índice do prefixo do n-ésimo par pode assumir
- O enésimo par possuirá, portanto, n bits, sendo
  - n-1 bits para representar o índice do prefixo, e
  - 1 bit para representar o último bit do trecho correspondente

Índice do trecho	1	2	3	4	5	6	7	8	9	10
Par	(0,0)	(1,0)	(0,1)	(1,1)	(3,1)	(2,0)	(3,0)	(6,0)	(2,1)	(9,1)
Nº bits necessários	0+1	1+1	2+1	2+1	3+1	3+1	3+1	3+1	4+1	4+1
Par codificado	0	10	001	011	0111	0100	0110	1100	00101	10011

- A codificação da cadeia de bits do exemplo fica

```
0100010110111010001101100010110011
```

0 0
1 1
2 10
3 11
4 100
5 101
6 110
7 111
8 1000
9 1001

Referências

DYNAMIC HUFFMAN CODING “ABRACADABRA”. [S. l.: s. n.], 18 maio 2017. Disponível em: https://www.youtube.com/watch?v=h5_ix_DnOtU.

SOUZA, J. F. de. Strings (Compressão). [S. l.]: Universidade Federal de Juiz de Fora, UFJF, 2010. Disponível em: https://docplayer.com.br/3862587-Strings-compressao-estrutura-de-dados-ii-jairo-francisco-de-souza.html.

SANTOS, C. Compressão de Dados Multimídia. [S. l.: s. n.], 2010. Disponível em: https://docplayer.com.br/3862431-Compressao-de-dados-multimidia.html.

FREIRE, A. da S. Compressor de Ziv e Lempel. [S. l.]: USP, 2018. Disponível em: https://edisciplinas.usp.br/mod/resource/view.php?id=2423088.