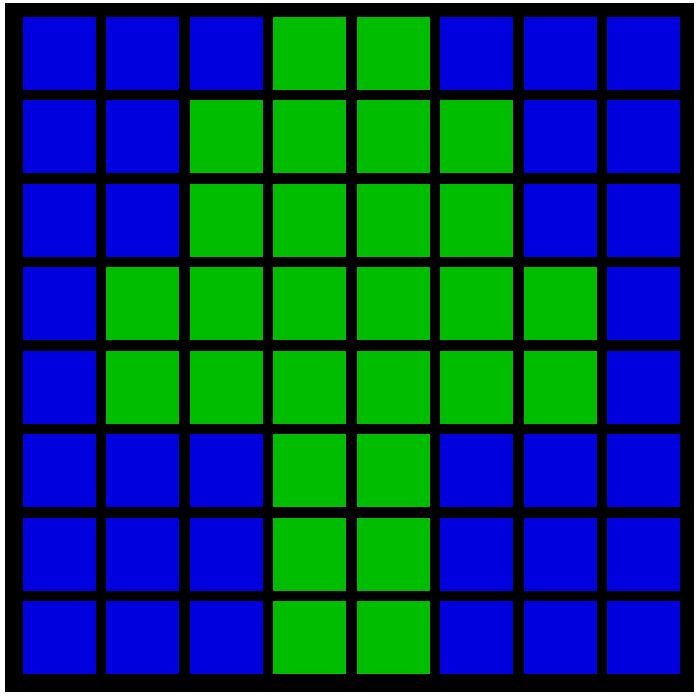
Aplicações de árvores/ Exercícios

30/11 e 2/12

QuadTrees para representação de imagens e

Algoritmo de Huffman para transmissão/compressão de dados

Representação de Imagens

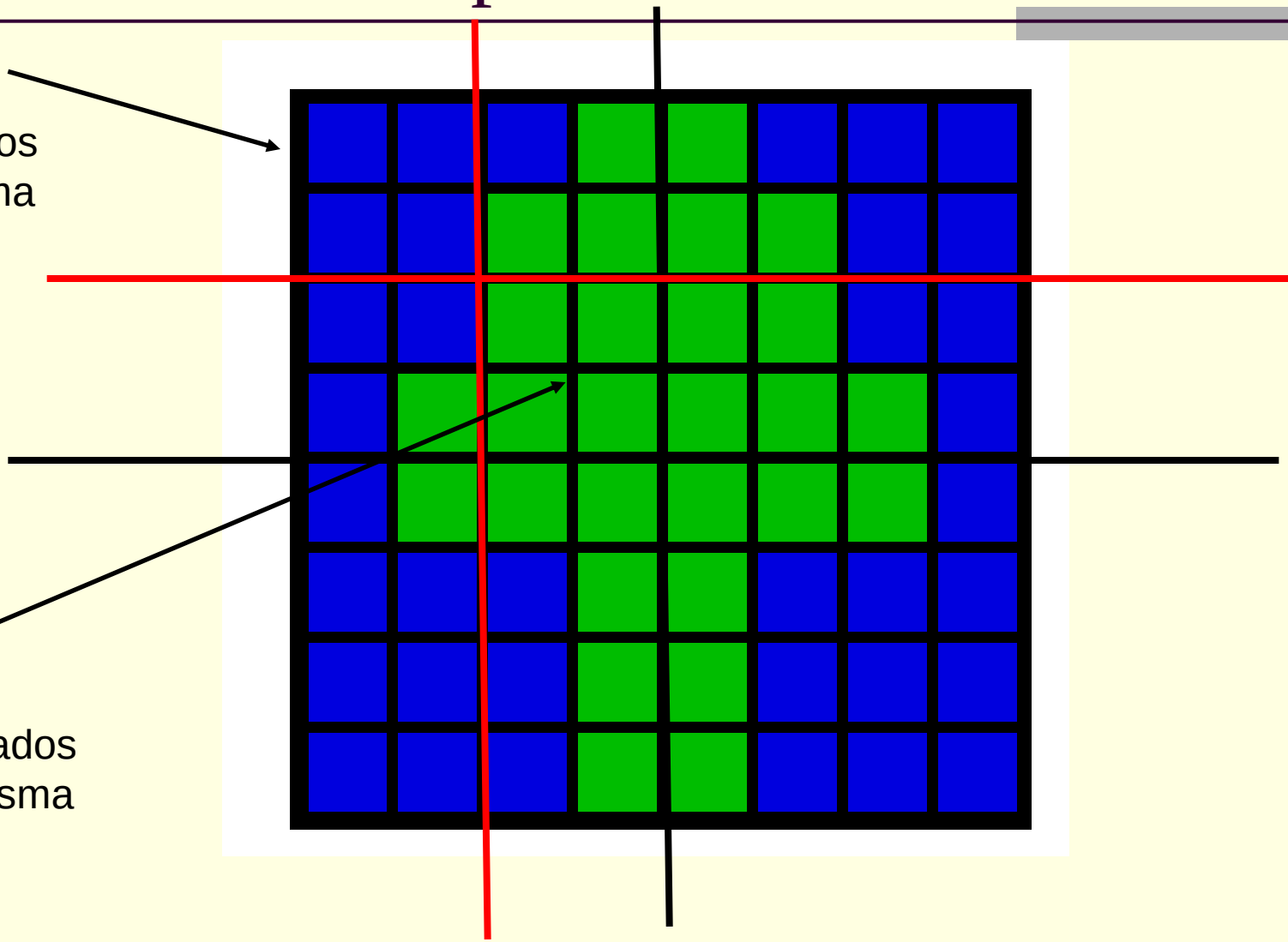


8 x 8 pixel picture represented in a two-dimensional array

Definição mais intuitiva para uma imagem: matriz bidimensional. Cada pixel é um elemento da matriz.

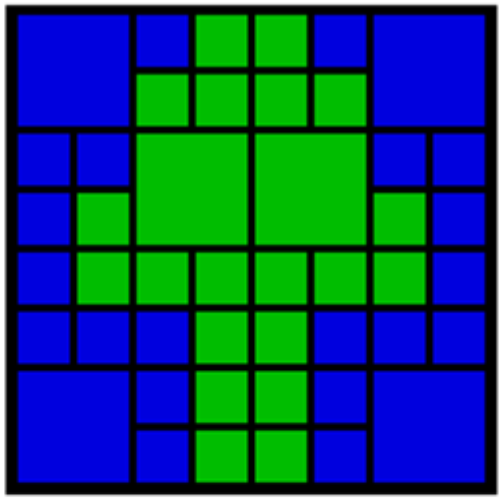
Representação Bitmap (raster) não é a única, nem a mais eficiente.

E se nós dividíssemos a figura em 4 partes e assim por diante?



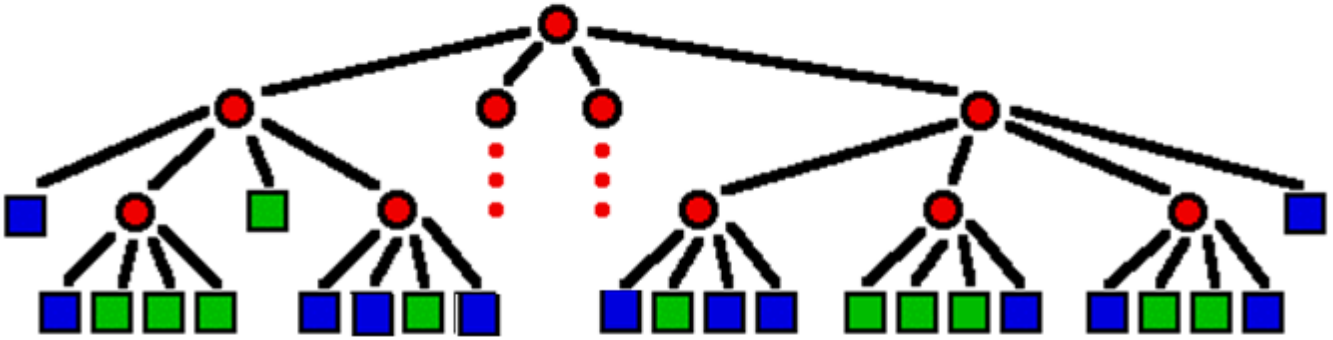
Representação da imagem como uma quadtree

3 1



2 4

8 x 8 pixel picture represented in a quad tree



The quad tree of the above example picture. The quadrants are shown in counterclockwise order from the top-right quadrant. The root is the top node. (The 2nd and 3rd quadrants are not shown.)

Definição

- Uma quadtree é uma árvore cujos nós ou são folhas ou tem 4 filhos.
- Os filhos são ordenados: 1, 2, 3 e 4 (quadrantes)

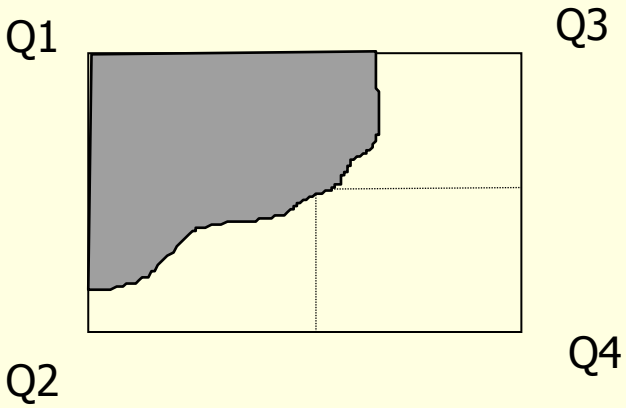
- A estratégia para gerar a árvore:
  - dividir para conquistar!

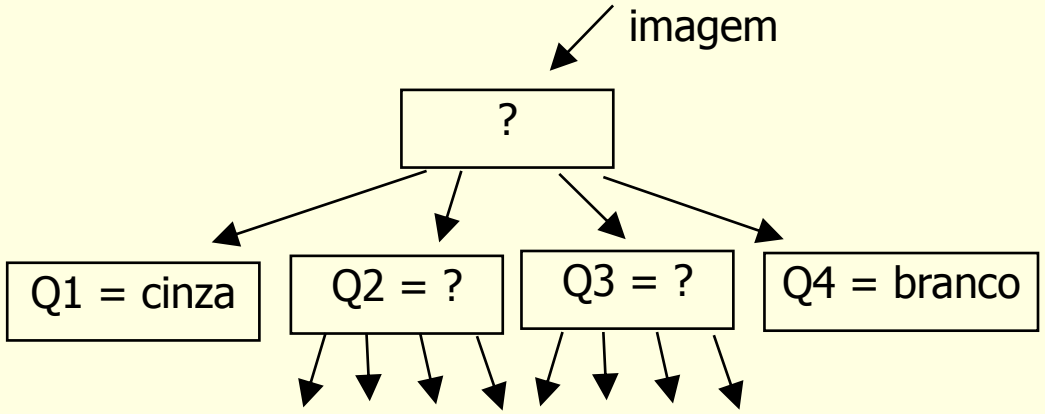
- O limite da parada da divisão:
  - Limite de espaço
  - Limite de tempo de processamento
  - Limite da resolução do dispositivo de saída

- Um pixel é a menor subseção de uma quadtree

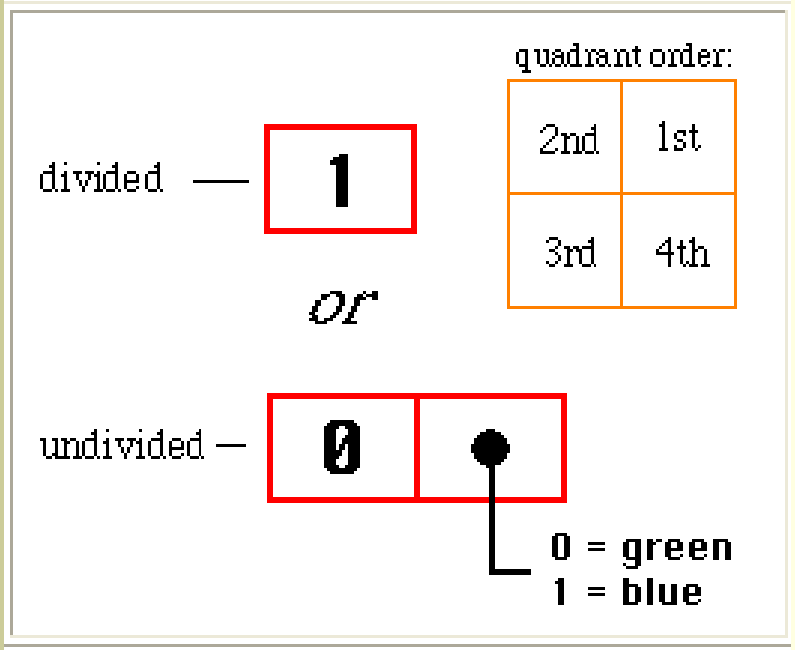
- Um quadrante pode ser:
  - De uma única cor
  - Composto de 4 sub-quadrados menores
- Cada folha:
  - Representa uma área uniforme da imagem
  - Se a imagem é B/P, nós precisamos de 1 único bit para representar a cor em cada folha
- Nenhum nó deve permitir que todos os seus descendentes tenham a mesma cor!
  - Não haveria razão para dividir

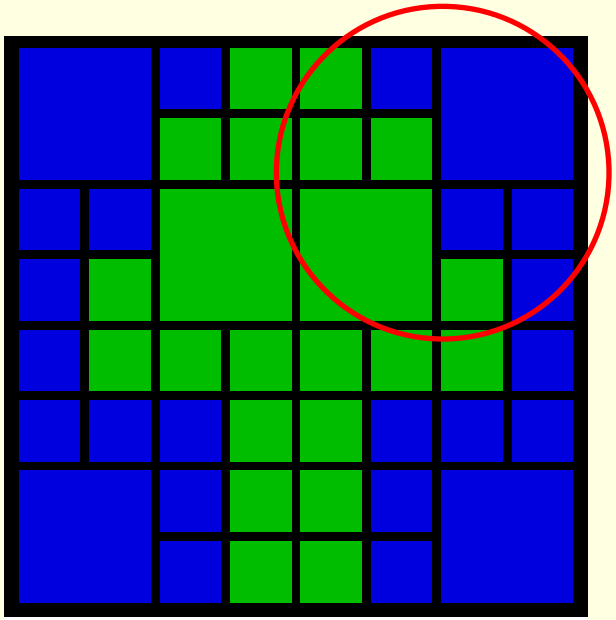
Exemplo da árvore, com outra disposição de quadrantes





Esquema para representação de imagens





1

Primeiro quadrante:

Eficiente para armazenamento de imagens de satélites.
O problema é espaço (ponteiros). A vantagem é a rapidez da Manipulação da imagem

```
1 1 01 1 01 00 00 00 00 1 01 01 00 01 ...
```

blue green

TAD Quadtree

Definições e decisões de projeto:

Nó transparente: composto de sub-quadrados

Nó neutro: “nenhuma cor” atribuída

Um nó não pode ser usado (fica inacessível) se pelo menos se pelo menos um de seus ancestrais é não transparente. Acontece em processamentos da árvore.

1. Checagem de ancentral
2. Divisão
3. Travessia
4. Remontagem

Ancestor Check/ Division to a Quad

- **Ancestor Check**
  - Given a node as a parameter, Ancestor Check returns TRUE if the node is accessed to display the picture. If one of its ancestors is not transparent, then that quadrant will never be drawn. The algorithm must check all the ancestors of the given node, starting at the root. If one of the ancestors is not transparent, the algorithm quits and returns FALSE.

- **Division to a Quad**
  - During manipulation of an image, we may need to divide a uniform square into sub-squares. Given a node in the quad tree, it divided it into quadrants.
  - The algorithm is relatively simple. The algorithm checks the parent of the given node. If the parent is not transparent:
    1.) store the colour of the parent
    2.) make the parent transparent
    2.) the 3 remaining children are created and the colour of the parent is transferred to the children.
  - The algorithm is then recursively called on the parent of the given node.

Tree Traversal/ Reassembly

- **Tree Traversal**
  - Given a node, Tree Traversal generates the sons of the node, and then repeats the process on each of the sons.
- **Reassembly**
  - After the colour of a square is changed, it is possible that it has the same value of its tree brother nodes. If that is the case, the colour is transferred to the father to maintain the requirement of a minimum level of division.
  - Given a node as a parameter, the algorithm generates the brothers of that node. If all the nodes have the same colour, the colour is transferred to the parent and the algorithm is recursively called on the parent. Otherwise, the algorithm quits.

ED

```
struct TquadTree {
        Cor : String;
        struct TquadTree *  NO;
        struct TquadTree *  NE;
        struct TquadTree *  SO;
        struct TquadTree *  SE;
    }

typedef struct TquadTree *QuadTree;
```

Vantagens/Desvantagens

- Muito usada em CG
  - Manipulada e acessada rapidamente
  - Popular para fractais, que são recursivos
  - Apagar: um só passo: setar raiz como neutra
  - Zoom de quadrante: um passo
  - Reduzir a complexidade da imagem: remover os nós finais
- Desvantagem: usa muito espaço, com ponteiros. Deve-se usar métodos de compactação para transferência eficiente de dados.
  - Link: http://www.inf.unisinos.br/~ari/estrut/quad/Quadtree.htm

Algoritmo de Huffman

- Codificação de mensagens
  - Transmissão de dados, criptografia, compressão, etc.
- Suponha que tenhamos um alfabeto de n símbolos e uma extensa mensagem consistindo desses símbolos para codificar
- Uma mensagem é codificada como uma cadeia de bits
  - Atribuindo-se uma seqüência de bits para cada símbolo do alfabeto
  - Concatenando-se os bits correspondentes a cada símbolo da mensagem

- Exemplo com alfabeto de 4 símbolos, 3 bits cada

Símbolo	Código
A	010
B	100
C	000
D	111

- A mensagem ABACCDA: 21 bits
  - 010100010000000111010

Ineficiente. Essa codificação pode ser melhorada?

- Uso de código de 2 bits

Símbolo	Código
A	00
B	01
C	10
D	11

- ABACCDA: 14 bits
  - 00010010101100

Essa codificação pode ser melhorada?

E se símbolos mais freqüentes recebessem códigos menores?

Símbolo	Código
A	0
B	110
C	10
D	111

- ABACCDA: 13 bits
  - 0110010101110

A aparece 3 vezes, então ganha o menor código

Boa solução: códigos de tamanho variável

- Símbolos mais freqüentes no alfabeto recebem códigos menores que os distinguem dos demais na interpretação do código (da esquerda para a direita)

- Qual a mensagem representada por 0110010101110?

Símbolo	Código
A	0
B	110
C	10
D	111

Código para um símbolo não pode ser o pré-fixo para outro

- Símbolos mais freqüentes no alfabeto recebem códigos menores que os distinguem dos demais na interpretação do código (da esquerda para a direita)

- Qual a mensagem representada por 0110010101110?
- Começa com A, pois é 0. Próximo pode ser (B|C|D)...

Símbolo	Código
A	0
B	1
C	10
D	111

Processo de decodificação: esquerda para direita

- Cada “prefixo” de um código identifica as possibilidades de codificação
  - Se primeiro bit é 0, então A; se 1, então é B, C ou D, dependendo do próximo bit
  - Se segundo bit é 0, então C; se 1, então é B ou D, dependendo do próximo bit
  - Se terceiro bit é 0, então B; se 1, então D
  - Quando o símbolo é determinado, começa-se novamente a partir do próximo bit

0110010101110?

AB AC C D A

Símbolo	Código
A	0
B	110
C	10
D	111

- Como representar todas as mensagens possíveis de serem expressas em língua portuguesa
  - Como determinar os códigos de cada letra/sílaba/palavra?
    - Qual a unidade do “alfabeto”?
- Com certeza a freqüência das letras deve ser usada.

Algoritmo de Huffman, David A. Huffman, 1952

- Algoritmo de Huffman
  - Calcula-se a freqüência de cada símbolo do alfabeto
    1. Encontre os dois símbolos que tem menor freqüência (B/1 e D/1)
    2. O último símbolo de seus códigos deve diferenciá-los (B=0 e D=1)
    3. Combinam-se esses símbolos e somam-se suas freqüências (BD/2, indicando ocorrência de B ou D)
    4. Repete-se o processo até restar um símbolo
       - C/2 e BD/2 ➔ 0 para C e 1 para BD ➔ CBD/4
       - A/3 e CBD/4 ➔ 0 para A e 1 para CBD ➔ ACBD/7

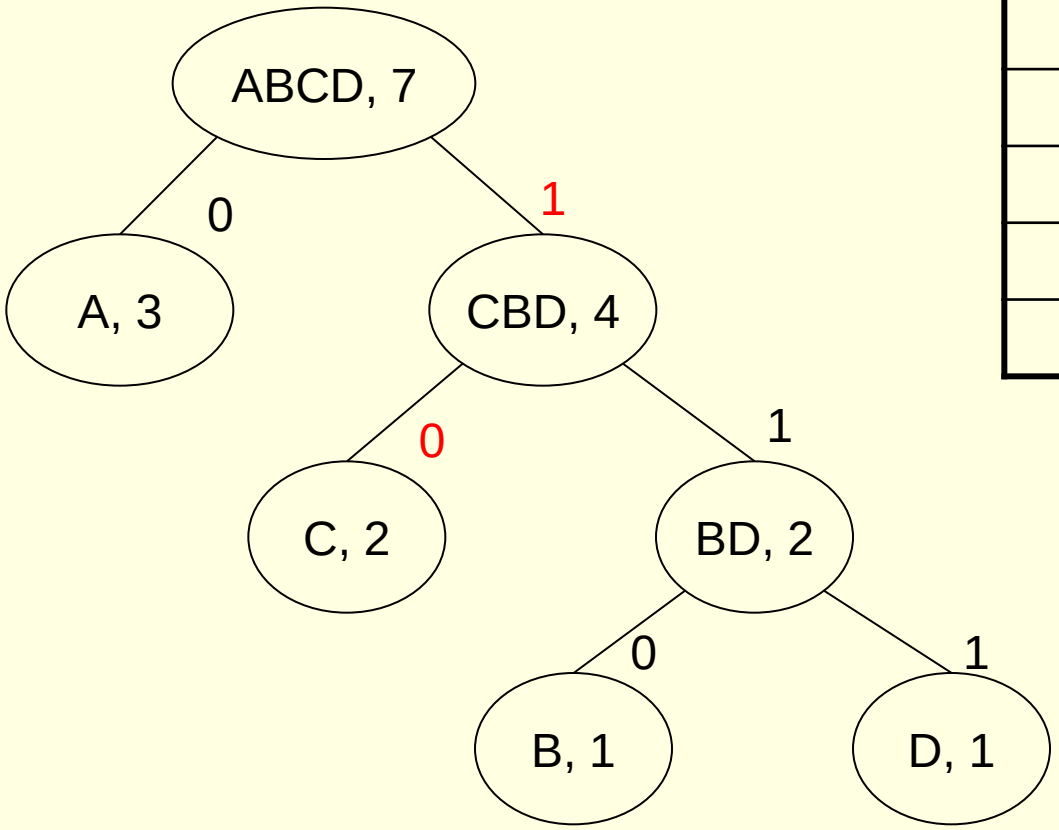
ABACCDA ⟶ Freq.: A/3, B/1, C/2 e D1

Algoritmo de Huffman: árvores binárias

- Combinação de dois símbolos em 1 (sugere a. binária)
  - Árvore de Huffman
    - Construída passo a passo após cada combinação de símbolos
    - Árvore binária
      - Cada nó da árvore representa um símbolo (e sua freqüência)
      - Cada nó folha representa um símbolo do alfabeto original
  - Ao se percorrer a árvore de uma folha X qualquer para a raiz, tem-se o código do símbolo X
    - Escalada por ramo esquerdo: 0 no início do código
    - Escalada por ramo direito: 1 no início do código

Árvores de Huffman: nós com símbolos e sua freqüência

- Exemplo



Símbolo	Código
A	0
B	110
C	10
D	111

Símbolos mais freqüentes mais à esquerda e mais próximos da raiz.

Algoritmo de Huffman

- Exercício: construir a árvore para os dados abaixo

Símbolo	Freqüência
A	15
B	6
C	7
D	12
E	25
F	4
G	6
H	1
I	15

Se há n símbolos no alfabeto, a árvore, que tem n folhas, pode ser representada por um vetor de tamanho 2n-1

Solução

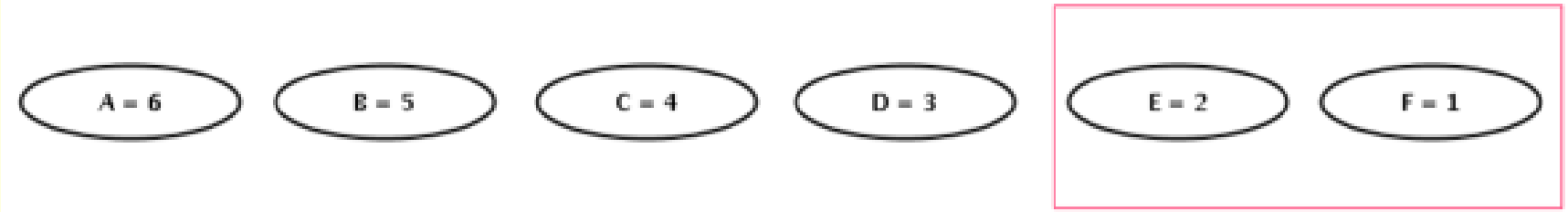
Símbolo	Freqüência	Código
A	15	111
B	6	0101
C	7	1101
D	12	011
E	25	10
F	4	01001
G	6	1100
H	1	01000
I	15	00

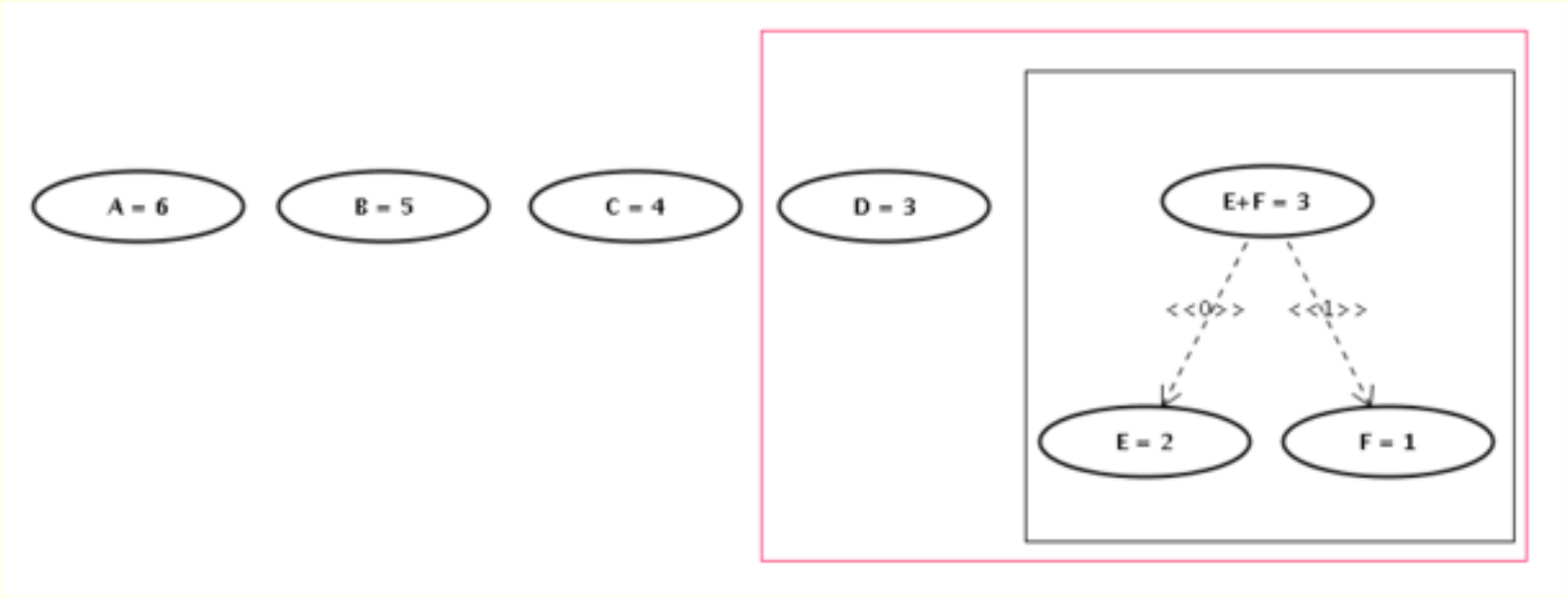
Algoritmo de Huffman

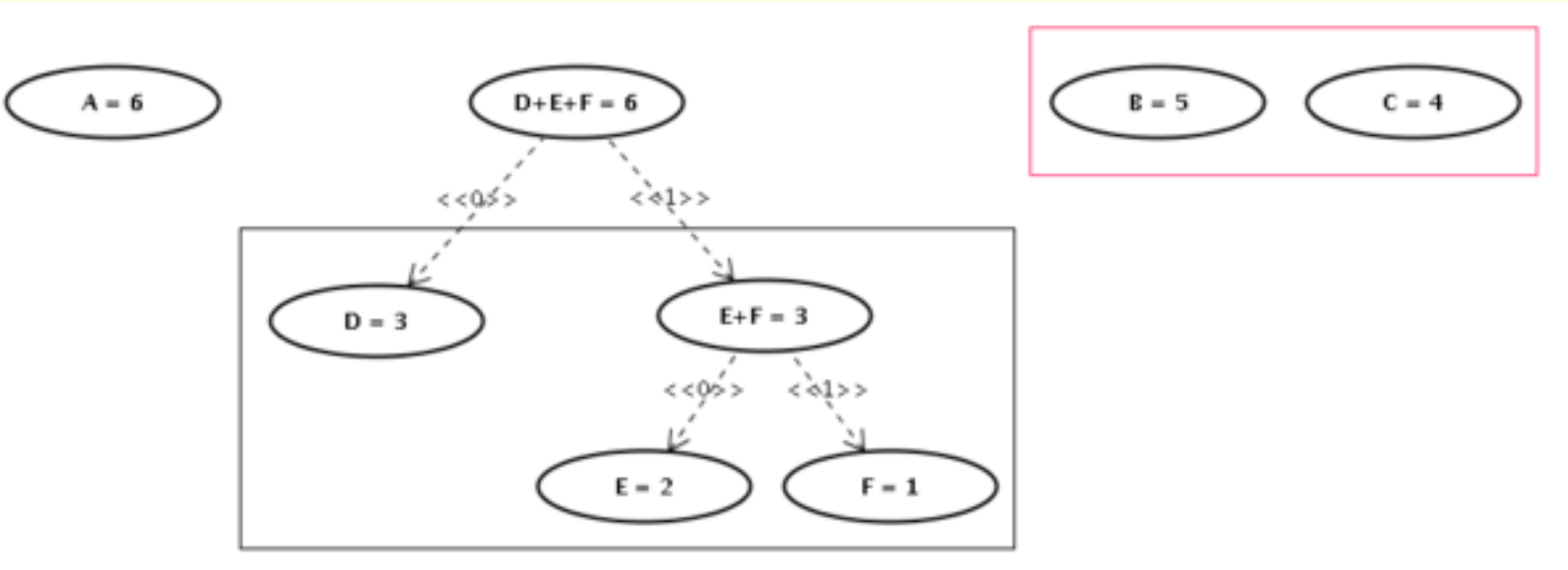
```
enquanto tamanho(alfabeto) > 1:
  S0 := retira_menor_probabilidade(alfabeto)
  S1 := retira_menor_probabilidade(alfabeto)
  X  := novo_nó
  X.filho0 := S0
  X.filho1 := S1
  X.probabilidade := S0.probabilidade + S1.probabilidade
  insere(alfabeto, X)
fim enquanto

X = retira_menor_símbolo(alfabeto) # nesse ponto só existe um símbolo.

para cada folha em folhas(X):
  código[folha] := percorre_da_raiz_até_a_folha(folha)
fim para
```

tamanho retorna a quantidade de nós ou folhas armazenados no nosso repositório, chamado aqui de *alfabeto*.
retira_menor_símbolo(alfabeto) retorna o nó ou folha de menor probabilidade no nosso repositório, e remove este símbolo do repositório.
insere(alfabeto, X) insere o símbolo *X* no nosso repositório
percorre_da_raiz_até_a_folha(folha) percorre a árvore binária da raiz até a folha acumulando os valores associados com as arestas em seu valor de retorno.

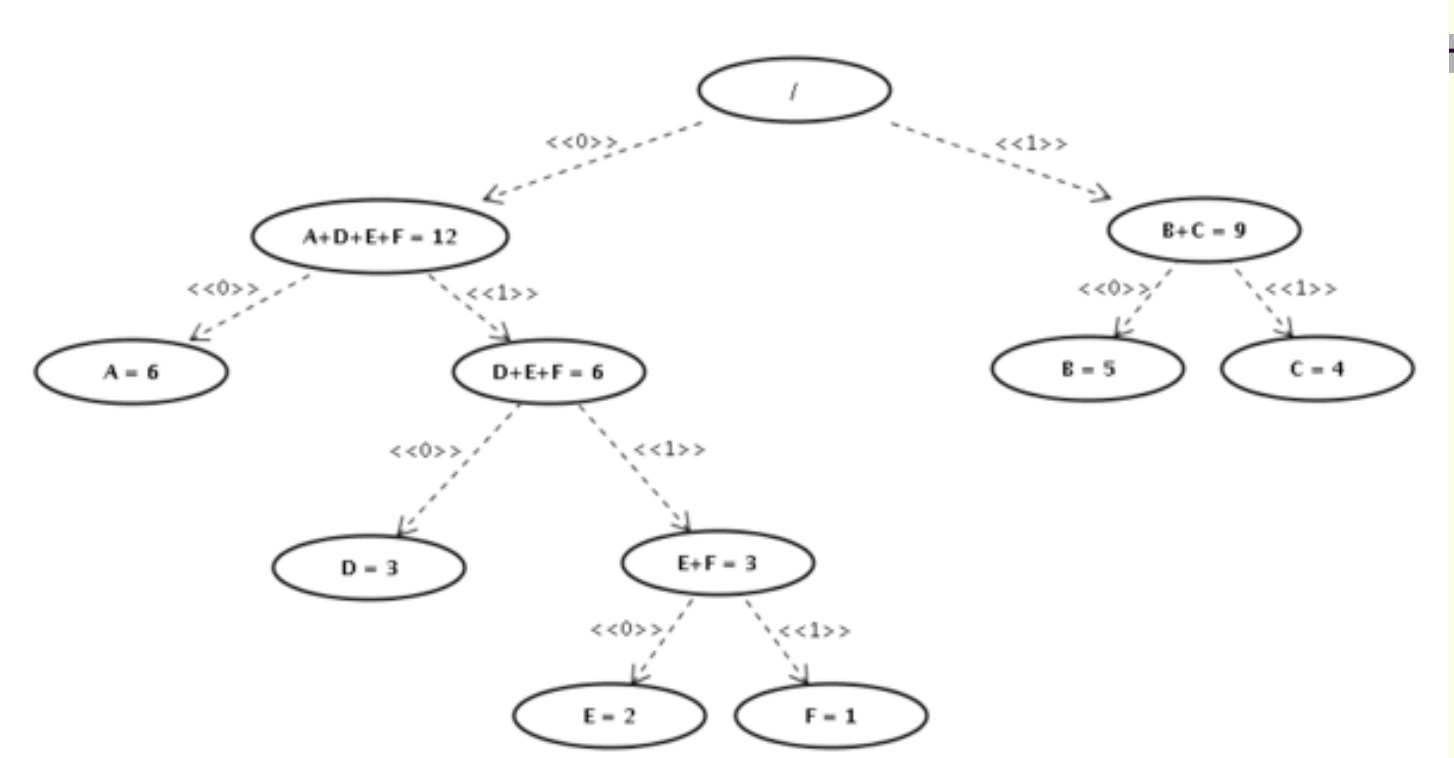
Caractere	A	B	C	D	E	F
Contagem	6	5	4	3	2	1






A = 6
D+E+F = 6
B = 5
C = 4
<<0>>
<<1>>
D = 3
E+F = 3
<<0>>
<<1>>
E = 2
F = 1

Para AAAAAABBBBBCCCCDDDEEF



Caractere	A	B	C	D	E	F
Contagem	00	10	11	010	0110	0111

000000000000101010101011111111010010010011001100111 totalizando apenas 51 bits. A compressão foi de 12 bits, ou cerca de 20%. Com 3 bits seriam 63 bits.

Exercícios

1. Duas ABBs são SIMILARES se possuem a mesma distribuição de nós (independente dos valores nos mesmos). Em uma definição mais formal, duas ABBs são SIMILARES se são ambas vazia, ou se suas subárvores esquerdas são similares e suas subárvores direitas também são similares
   - Implemente a sub-rotina que verifica se duas ABBs são similares

2. Duas ABBs são IGUAIS se são ambas vazias ou então se armazenam valores iguais em suas raízes, suas subárvores esquerdas são iguais e suas subárvores direitas são iguais
   - Implemente a sub-rotina que verifica se duas ABBs são iguais

3. Uma ABB é estritamente binária se todos os nós da árvore tem 2 filhos ou nenhum filho

- Implemente uma sub-rotina que verifica se uma ABB é estritamente binária

4. Implemente uma sub-rotina para verificar se uma árvore binária é uma ABB