Algoritmo de Huffman

Código de Huffman

- Códigos de tamanho variável

Código ASCII	Código de Huffman
A=01000001	A=? (0)
B=01000010	B=? (110)
.	.
.	.
.	.
a=01100001	a=? (1111110)
b=01100010	b=? (11111111110)

Código de Huffman

- Algoritmo para compressão de arquivos texto
- Codifica cada caracter por uma seqüência de bits
- Código de tamanho variável
  - Atribui códigos menores (mais curtos) para símbolos mais freqüentes, e códigos maiores para símbolos menos freqüentes

Exemplo

- Supondo A e C são mais freqüentes do que C e D no conjunto de valores possíveis

Símbolo	Código
A	0
B	110
C	10
D	111

ABACCDA= 0 110 0 10 111 0

A | B | A | C | D | A

Requisito

Símbolo	Huffman
A	0
B	01
C	1

- O código de um símbolo não pode ser prefixo do código de outro
  - Isso acarretaria ambigüidade na decodificação
- Ex.: ACBA = 01010
  - Os dois bits em vermelho representam 'AC' ou B?
  - Isso porque o código de A é prefixo do código de B, o que não pode ocorrer

Informação de freqüência

- Algoritmo de Huffman produz tabela de códigos baseada em informações de freqüência de ocorrência dos caracteres do alfabeto
- Dependência do tipo de dado primário
- Implicação: cada arquivo comprimido tem o seu próprio conjunto de símbolos

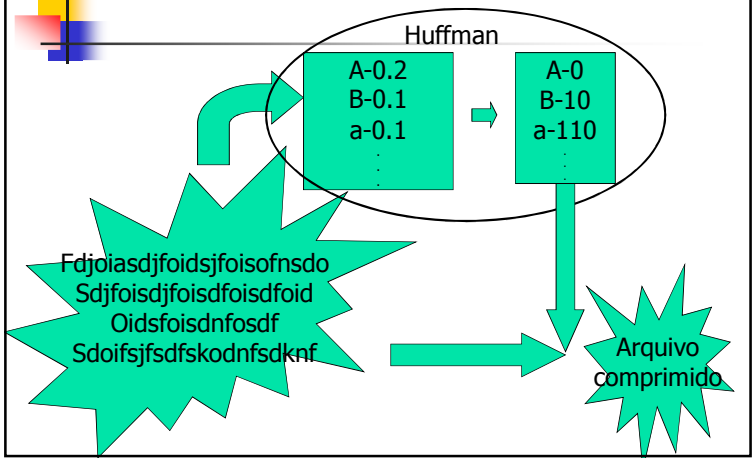
Problema

- Dada uma tabela de freqüências, como determinar o melhor conjunto de códigos, ou seja, o conjunto que comprimirá mais os símbolos?

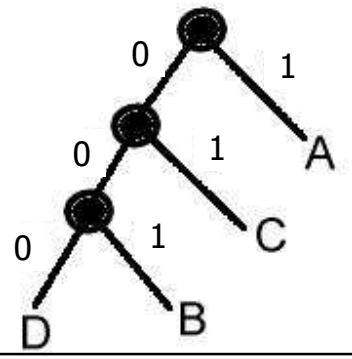
O algoritmo em si

- Dado: Tabela de freqüências dos N símbolos de um alfabeto
- Objetivo: Atribuir códigos aos símbolos de modo que os mais freqüentes tenham códigos menores (menos bits)

O processo de compressão



Exemplo



Símbolo	Código
A	1
B	001
C	01
D	000

Idéia básica

- Construir uma árvore binária tal que
  - A) suas folhas sejam os N símbolos do alfabeto
  - B) cada ramo da árvore está associado a um valor 0 (esquerda) ou 1 (direita)
    - Isso é uma convenção, o contrário também funcionaria
  - O código de um símbolo será a seqüência de bits obtida seguindo o caminho da raiz até a posição do caractere na árvore

A árvore no algoritmo de Huffman

- Árvore é de tamanho fixo (2N-1 nós)
- Construção da árvore é das folhas para a raiz, então os ponteiros serão:

  filho→pai
- Os nós não têm (explicitamente) os campos **esq** e **dir**

Os nós da árvore

- Cada nó tem 4 campos:
  - Father – ponteiro para a posição do pai do nó (no array que armazena a árvore)
  - isLeft (para que este campo?)
    - True se o nó é filho esquerdo
    - False se o nó é filho direito
  - symbol – o caracter representado pelo nó
  - freq – a freqüência do símbolo

Processo

- Cria-se um nó (nó folha) para cada símbolo da tabela de freqüência
- Cria-se um vetor que aponta para cada um desses nós
- Insere-se esses nós em uma uma fila de prioridades (nós menos freqüentes primeiro)
  - Notem: temos uma árvore E uma fila de prioridades
  - A fila de prioridades é usada para construir a árvore
- O processo termina quando a fila de prioridades ficar vazia
  - Os últimos dois juntam-se e formam a raiz da árvore

Processo

- Árvore construída de baixo para cima, a partir dos 2 símbolos de menor freqüência, e recursivamente tomando-se sempre as 2 sub-árvores menos freqüentes
- Definição: A freqüência de uma sub-árvore é a soma das freqüências de seus 2 filhos

Processo(visão geral)

- Enquanto existir mais de 1 nó na fila
  - Retira-se os dois primeiros
  - Gera-se um novo nó a partir destes
  - Insere o nó associado a estes dois na árvore
- No final restará um nó na fila de prioridades

N-nº de símbolos tratados

Frequencias – uma tabela com os símbolos e suas freqüências

Code – Saída do algoritmo. Uma tabela com os símbolos e os seus respectivos códigos

Rootnodes – fila de prioridades

Position – vetor de ponteiros para os nós iniciais (nós folhas)

```
Code Huffman(N,Frequencias)
        rootnodes=FilaVazia //inicializa o conjunto de "root nodes"
        for(i=0;i<n;i++){
                P=makeTree(frequencias[i]);
                position[i]=P; //P ponteiro para folha
                pqinsert(rootnodes,P);
        } //este laço cria todos os nós folhas
        ...continua no próximo slide
```

```
//gerando os códigos a partir da árvore
root=pqmindelete(rootnodes);
for (i=0; i<n; i++) {
   P= position[i]; // i-esimo símbolo
   code[i]="\"; // string de bits nula
   while (P != root) { // sobe na árvore
        if (isleft(P))
                code[i]= 0 + code[i];
        else
                code[i]= 1+code[i];
        P= father(P); // sobre para o pai
   } //end while
}//end for

Fim do algoritmo de Huffman
```

Algoritmo de Huffman

Gera os nós iniciais e Constrói a fila de prioridades

Gera a árvore binária

Gera os códigos a partir da árvore

pqMinDelete – retorna o primeiro nó da fila de prioridades

makeTree – gera um novo nó da árvore

Setleft(P,P1) – seta o pai de P1 e seta que P1 é filho esquerdo

Setright(P,P2) – seta o pai de P2 e seta que P2 pe filho direito

Pqinsert – insere um nó na fila de prioridades

```
// geração da árvore
   while (size(rootnodes) > 1) {
        P1= pqMinDelete(rootnodes); // remove da fila
        P2= pqMinDelete(rootnodes); // os 2 de menor freq.
        //combina P1 e P2 em um nó
        P= makeTree(info(P1)+info(P2)) // cria árv. c/ raiz em P
        setleft(P,P1); setRight(P,P2);
        pqinsert(rootnodes,P); //insere nó pai na fila de
   prioridades
   }// este laço constrói a árvore
Continua....
```

A tabela de códigos (code)

a	1	0
b	2	10
c	3	11

Codificando

a	b	c	a

01100001	01100010	01100011	01100001

a	1	1
b	2	01
c	3	00

101001			

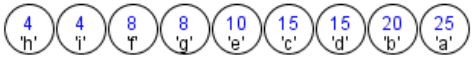
Exemplo (1)

Tabela de freqüência

aaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbcccc
ccccccccccddddddddddddddeeeeeeeeeeffffffffgggggggh
hhhiiii

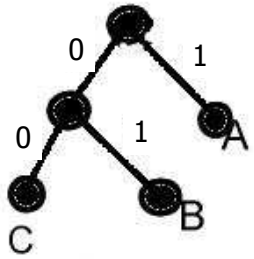
A	25
B	20
C	15
D	15
E	10
F	8
G	8
H	4
I	4

Fila de prioridades



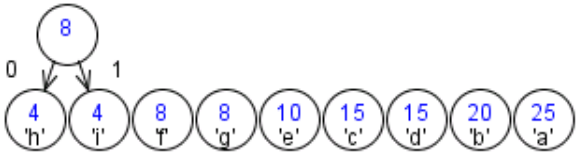
Decodificando

101001

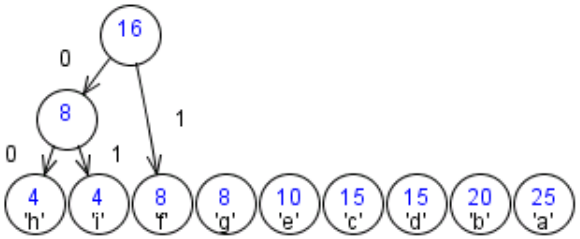


a	b	c	a

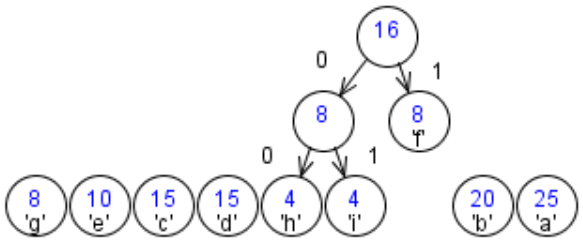
Exemplo (2)



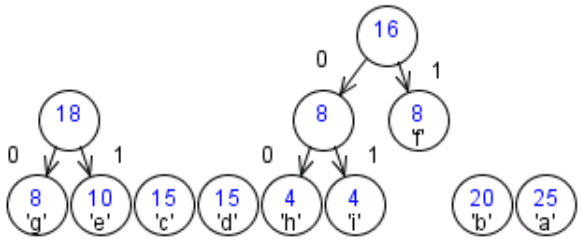
Exemplo (3)



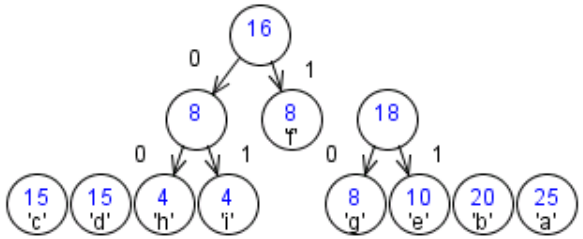
Exemplo (4)



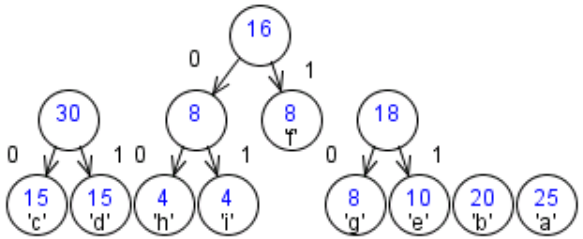
Exemplo (5)



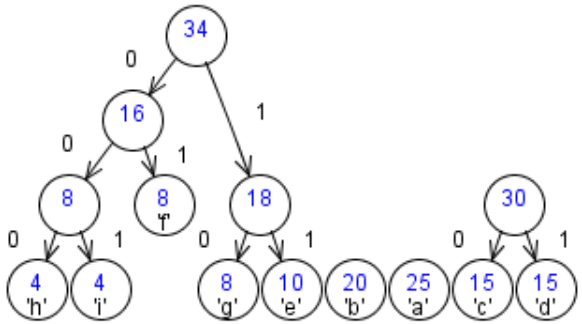
Exemplo (6)



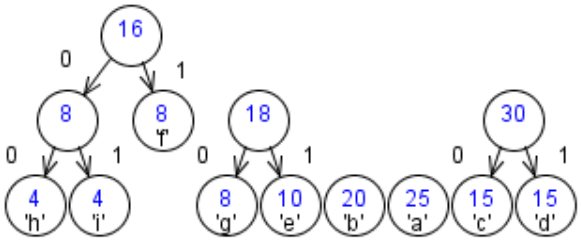
Exemplo (7)



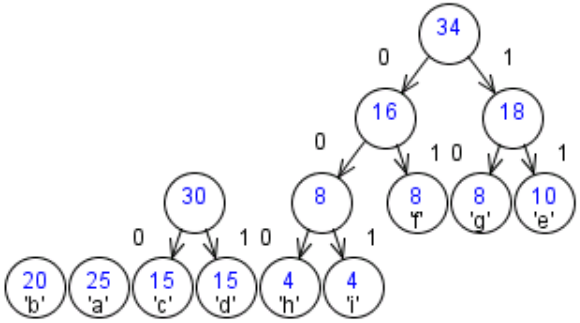
Exemplo (9)



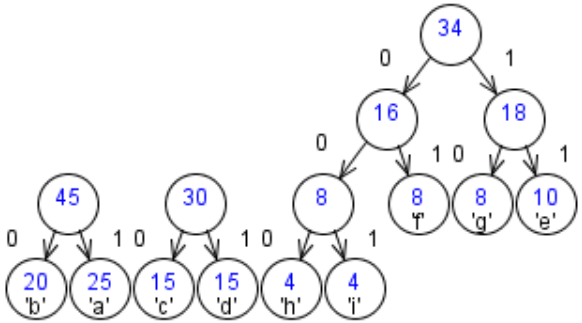
Exemplo (8)



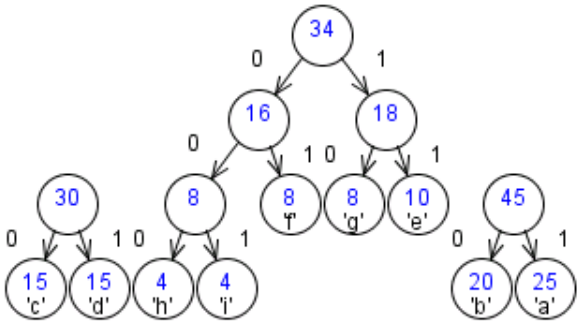
Exemplo (10)



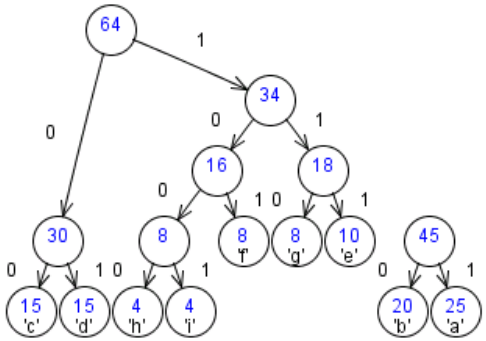
Exemplo (11)



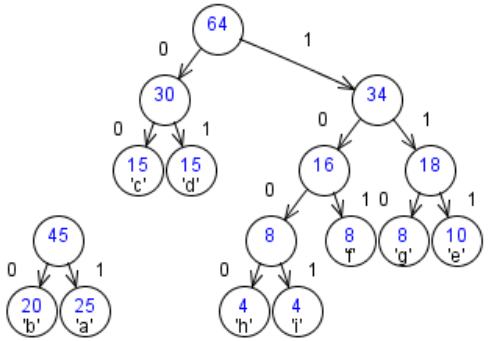
Exemplo (12)



Exemplo (13)



Exemplo (14)



Exemplo (15)

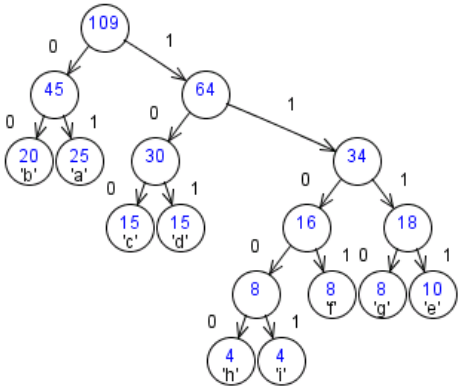
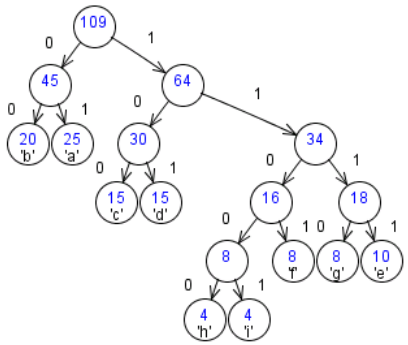


Tabela de códigos

Símbolo	Nº bits	Código
A	2	01
B	2	00
C	3	100
D	3	101
E	4	1111
F	4	1101
G	4	1110
H	5	11000
I	5	11001

Exemplo do algoritmo



A	25	01
B	20	00
C	15	100
D	15	101
E	10	1111
F	8	1101
G	8	1110
H	4	11000
I	4	11001

Operadores bit a bit no C

&	AND
|	OR
^	OR exclusivo (XOR)
>>	Deslocamento à direita
<<	Deslocamento à esquerda

- Operam sobre char e int

AND bit a bit

AND bit a bit
1 1 0 0 0 0 0 1
0 1 1 1 1 1 1 1
&----------
0 1 0 0 0 0 0 1

```
void main(void) {
        char a,b,c;
        a=193;
        b=127;
        c=a & b;//c=65
        printf("%i\n",c);
};
```

XOR bit a bit

XOR bit a bit
1 1 0 0 0 0 0 1
0 1 1 1 1 1 1 1
^----------
1 0 1 1 1 1 1 0

```
void main(void) {
        char a,b,c;
        a=193;
        b=127;
        c=a & b;//c=190
        printf("%i\n",c);
};
```

OR bit a bit

OR bit a bit
1 1 0 0 0 0 0 1
0 1 1 1 1 1 1 1
|----------
1 1 1 1 1 1 1 1

```
void main(void) {
        char a,b,c;
        a=193;
        b=127;
        c=a & b;//c=255
        printf("%i\n",c);
};
```

Deslocamento à direita e à esquerda

```
void main(void) {
        char x;
        x<<1;
        x<<3;
        x<<2;
        x>>1;
        x>>2;
};
```

X=7	0000 0111	7
X<<1	0000 1110	14
X<<3	0111 0000	112
X<<2	1100 0000	192
X>>1	0110 0000	96
X>>2	0001 1000	24

- Fonte: Estruturas de Dados Usando C. Tenembaum et al., cap. 5. Makron Books