Codificação de Huffman

Prof. Daniel Kikuti

Universidade Estadual de Maringá

Codificação de Huffman

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. **Introduction to Algorithms**, Third Edition. The MIT Press. Chapter 16.
- Kleinberg J., and Tardos E. **Algorithm Design**. 2005. Pearson. Chapter 4.

Problema de codificação

P_1 Dado um texto que usa 32 símbolos (26 letras diferentes, espaço e alguns caracteres de pontuação), como podemos codificar este texto em bits?

P_2 Alguns símbolos são usados com mais frequência que outros. Como podemos usar isto para reduzir nossa codificação?

Problema de codificação

P_1 Dado um texto que usa 32 símbolos (26 letras diferentes, espaço e alguns caracteres de pontuação), como podemos codificar este texto em bits?

R_1 **Codificação de tamanho fixo**.
Cada caracter é representado por uma única *string* binária (**palavra-código**). Podemos codificar 2^5 símbolos diferentes usando palavras de 5 bits de comprimento por símbolo.

P_2 Alguns símbolos são usados com mais frequência que outros. Como podemos usar isto para reduzir nossa codificação?

R_2 **Codificação de tamanho variável**.

$$\text{Caracteres mais frequentes} \implies \text{códigos binários menores}$$
$$\text{Caracteres menos frequentes} \implies \text{códigos binários maiores}$$

Exemplo

	a	**b**	**c**	**d**	**e**	**f**
Frequência (em milhares)	45	13	12	16	9	5
Fixo	000	001	010	011	100	101
Variável	0	101	100	111	1101	1100

Quantos bits são usados na codificação:

- com tamanho fixo?
- com tamanho variável?

Exemplo

	a	**b**	**c**	**d**	**e**	**f**
Frequência (em milhares)	45	13	12	16	9	5
Fixo	000	001	010	011	100	101
Variável	0	101	100	111	1101	1100

Quantos bits são usados na codificação:

- com tamanho fixo? **300.000 bits.**
- com tamanho variável? **224.000 bits.**

Usando tamanho variável seria possível salvar aproximadamente 25% de espaço.

Compressão de dados

Para quê isto?

- Transmissão de dados em redes de comunicação.
- Armazenamento em disco.

O que se espera?

- Quanto mais compacto melhor.
- Que seja possível reconstruir corretamente a informação compactada.

Como reconstruir a informação se o tamanho do código for variável?

Seja $a = 01$, $b = 010$, $c = 1$. O que seria 0101? Como saber quando o próximo símbolo começa?

Compressão de dados

Para quê isto?

- Transmissão de dados em redes de comunicação.
- Armazenamento em disco.

O que se espera?

- Quanto mais compacto melhor.
- Que seja possível reconstruir corretamente a informação compactada.

Como reconstruir a informação se o tamanho do código for variável?

Seja $a = 01$, $b = 010$, $c = 1$. O que seria 0101? Como saber quando o próximo símbolo começa? Para evitar o problema de ambiguidade, podemos definir um caracter para separação ou **garantir que nenhum código é prefixo de outro**.

Código de prefixo (*prefix code*)

Definição

Um **código de prefixo**[1] para um conjunto S é uma função bijetora c que associa cada $x \in S$ a $0s$ e $1s$, de modo que para $x, y \in S$, $x \neq y$, $c(x)$ não é prefixo de $c(y)$.

Exemplo

Seja $c(a) = 11$, $c(b) = 01$, $c(c) = 001$, $c(d) = 10$, $c(e) = 000$. O que seria 1001000001?

[1] Talvez "código livre de prefixo" fosse um mais apropriado, mas "código de prefixo" é padrão na literatura [Cormen et al., 2009].

Código de prefixo (*prefix code*)

Definição

Um **código de prefixo**[1] para um conjunto S é uma função bijetora c que associa cada $x \in S$ a $0s$ e $1s$, de modo que para $x, y \in S$, $x \neq y$, $c(x)$ não é prefixo de $c(y)$.

Exemplo

Seja $c(a) = 11$, $c(b) = 01$, $c(c) = 001$, $c(d) = 10$, $c(e) = 000$. O que seria 1001000001? Para reconstruir o texto:

1. varra a sequência de bits da esquerda para a direita;
2. formou uma palavra-código, imprima a letra correspondente (pois nenhum prefixo mais curto ou longo poderia codificar outra letra);
3. apague a palavra-código no início da mensagem e repita.

Decodificada: $dbec$.

[1] Talvez "código livre de prefixo" fosse um mais apropriado, mas "código de prefixo" é padrão na literatura [Cormen et al., 2009].

Código de prefixo ótimo

Definição

O número de bits necessários para codificar um arquivo usando o código de prefixo c é igual a soma de todas as frequências dos símbolos multiplicado pelo número de bits usados na codificação:

$$B(c) = \sum_{x \in S} f_x \cdot |c(x)|. \tag{1}$$

Objetivo: encontrar um código de prefixo que tenha o menor custo $B(c)$.

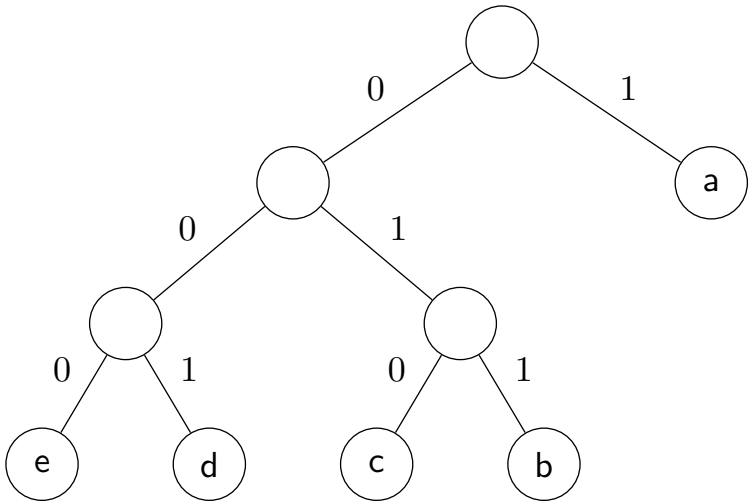
Código de prefixo e árvores binárias

- Folhas $=$ caracteres (cada folha é um $x \in S$);
- Todas as letras $x \in S$ cuja codificação começa com um 0 serão folhas na subárvore à esquerda da raiz;
- Todas as letras $x \in S$ cuja codificação começa com um 1 serão folhas na subárvore à direita da raiz;

Não há ambiguidade

Uma codificação $c(x)$ é um prefixo de uma codificação $c(y)$ se e somente se o caminho de $c(x)$ é prefixo do caminho de $c(y)$.

Exemplo



$$a = 1$$
$$b = 011$$
$$c = 010$$
$$d = 001$$
$$e = 000$$

Exercício

1. Faça a árvore para a codificação: $a = 11$, $b = 01$, $c = 001$, $d = 10$ e $e = 000$.
2. Faça a árvore para a codificação: $a = 000$, $b = 001$, $c = 010$, $d = 011$ e $e = 100$.

Código de prefixo ótimo

Definição

Uma árvore binária é **cheia** se cada nó que não é uma folha possui dois filhos.

Proposição

Uma árvore binária representando um código de prefixo ótimo é cheia.

Demonstração (por contradição)

- Suponha que T é uma árvore binária representando um código de prefixo ótimo e que não é cheia.
- Isto significa que existe um nó u (não folha) que contém apenas um filho v.
  - Caso 1: u é a raiz; remova u e use v como raiz.
  - Caso 2: seja w o pai de u; remova u e faça de v um filho de w.
- Em ambos os casos, o número de bits necessários para codificar nós folhas descendentes de v diminuiu. O resto da árvore permaneceu inalterado. Portanto, para esta árvore T' temos $B(T') < B(T)$.

Código de prefixo: um algoritmo de divisão e conquista

Onde em uma árvore representando um código de prefixo ótimo devemos colocar as letras que aparecem com mais frequência?

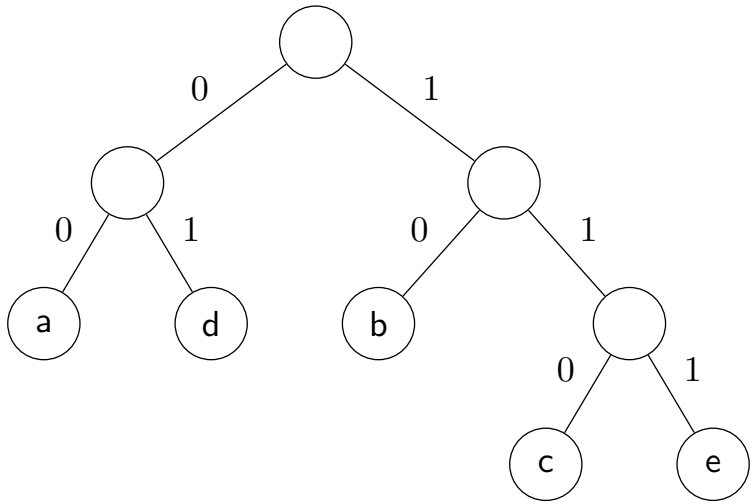
Shannon-Fano, 1949

Crie uma árvore dividindo S em dois subconjuntos S_1 e S_2 com frequências aproximadamente iguais. Recursivamente crie subárvores para S_1 e S_2.

Exercício

Mostre que esta abordagem falha para $f_a = 0.32$, $f_b = 0.25$, $f_c = 0.2$, $f_d = 0.18$ e $f_e = 0.05$.

Resolução do exercício



Por que a árvore gerada não é ótima?

Porque $f_c > f_d$, mas na codificação gerada, $|c(d)| = 2$ e $|c(c)| = 3$ (se trocássemos de posição c e d nesta árvore, teríamos um código de prefixo com custo menor).

Pensando na solução ótima T^*

Proposição

Se u e v são folhas em T^* associadas respectivamente a $x, y \in S$, tal que $|c(u)| < |c(v)|$, então $f_x \geq f_y$.

Demonstração [por contradição]

Suponha que $f_x < f_y$. Considere então o código de prefixo T' obtido pela troca entre os carateres associados aos nós u e v.
Na expressão $\sum_{k \in S} f_k.|c(k)|$, teríamos as seguintes mudanças nos multiplicadores de f_x e f_y:

$$f_x\big(|c(u)| + (|c(v)| - |c(u)|)\big) + f_y\big(|c(v)| - (|c(v)| - |c(u))|\big)$$

Aplicando distributiva e agrupando os termos temos:

$$f_x|c(u)| + f_y|c(v)| + (f_x - f_y)(|c(v)| - |c(u)|)$$

Como $f_x < f_y$ e $|c(v)| > |c(u)|$, a parte destacada será negativa, indicando que $B(T') < B(T^*)$ (contradição).

Código de Huffman

Observações importantes

1. Caracteres com menos frequência devem ficar no nível mais baixo de uma árvore representando um código de prefixo ótimo.
2. Para $n > 1$, o nível mais baixo sempre contém pelo menos duas folhas.
3. A ordem em que os caracteres aparecem em um nível não importa.

Proposição

Existe um código de prefixo ótimo com árvore T^* tal que dois caracteres com menores frequências são atribuídos a folhas que são irmãs em T^*.
(Ver demonstração no lema 16.2 [Cormen et al.] ou 4.31 [Kleinberg-Tardos].)

Código de Huffman

Escolha gulosa [Huffman, 1952]

Crie a árvore de baixo para cima. Crie duas folhas para os caracteres com menores frequência y e z. Recursivamente construa uma árvore para o restante usando um meta-caracter para yz.

HUFFMAN(S)

if $|S| = 2$ **then**
  $T \leftarrow$ Árvore com raiz e duas folhas.
else
  Seja y e z os caracteres com menor frequência em S
  $S' \leftarrow S - \{y\} - \{z\} \cup \{yz\}$, com $f_{yz} = f_y + f_z$
  $T' \leftarrow$ HUFFMAN(S')
  $T \leftarrow$ Adicione y e z como folhas de yz em T'
return T

Algoritmo guloso recursivo

HUFFMAN(S)

1 **if** $|S| = 2$ **then**
2 | $T \leftarrow$ Árvore com raiz e duas folhas.
3 **else**
4 | Seja y e z os caracteres com menor frequência em S
5 | $S' \leftarrow S - \{y\} - \{z\} \cup \{yz\}$, com $f_{yz} = f_y + f_z$
6 | $T' \leftarrow$ HUFFMAN(S')
7 | $T \leftarrow$ Adicione y e z como folhas de yz em T'
8 **return** T

Complexidade de tempo

Algoritmo guloso recursivo

HUFFMAN(S)

1 **if** $|S| = 2$ **then**
2 | $T \leftarrow$ Árvore com raiz e duas folhas.
3 **else**
4 | Seja y e z os caracteres com menor frequência em S
5 | $S' \leftarrow S - \{y\} - \{z\} \cup \{yz\}$, com $f_{yz} = f_y + f_z$
6 | $T' \leftarrow$ HUFFMAN(S')
7 | $T \leftarrow$ Adicione y e z como folhas de yz em T'
8 **return** T

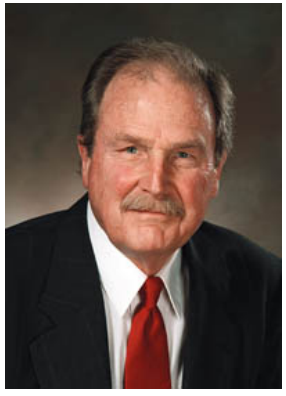
Complexidade de tempo

- Usando uma maneira pouco eficiente para encontrar os caracteres com menores freqência:
  $T(n) = T(n-1) + O(n) \implies O(n^2)$
- Usando fila de prioridades para S:
  $T(n) = T(n-1) + O(\lg n) \implies O(n \lg n)$

Algoritmo guloso iterativo

```
HUFFMAN(S)
1 n ← |S|
2 Q ← S
3 for i ← 1 to n − 1 do
4 |   aloque um novo nó z
5 |   z.left ← x ← EXTRACT-MIN(Q)
6 |   z.rigth ← y ← EXTRACT-MIN(Q)
7 |   z.freq ← x.freq + y.freq
8 |   INSERT(Q, z)
9 return EXTRACT-MIN(Q)
```

David Albert Huffman



(09/08/1925 – 07/10/1999)

Conhecido pela invenção do Código de Huffman, uma técnica importante de compressão sem perda de dados com uma codificação de comprimento variável. Principal campo de atuação: **Teoria da Informação.**

Curiosidade

The story of the invention of Huffman codes is a great story that demonstrates that students can do better than professors. David Huffman (1925-1999) was a student in an electrical engineering course in 1951. His professor, Robert Fano, offered students a choice of taking a final exam or writing a term paper. Huffman did not want to take the final so he started working on the term paper. The topic of the paper was to find the most efficient (optimal) code. What Professor Fano did not tell his students was the fact that it was an open problem and that he was working on the problem himself. Huffman spent a lot of time on the problem and was ready to give up when the solution suddenly came to him. The code he discovered was optimal, that is, it had the lowest possible average message length. The method that Fano had developed for this problem did not always produce an optimal code. Therefore, Huffman did better than his professor. Later Huffman said that likely he would not have even attempted the problem if he had known that his professor was struggling with it.

Fonte: `https://www.maa.org/press/periodicals/convergence/discovery-of-huffman-codes`