Códigos de Huffman

- Códigos de Huffman: técnica de compressão de dados.
- Reduções no tamanho dos arquivos dependem das características dos dados contidos nos mesmos. Valores típicos oscilam entre 20 e 90%.
- Exemplo: arquivo texto contendo 100.000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As *freqüências* de cada caracter no arquivo são indicadas na tabela abaixo.

	a	*b*	*c*	*d*	*e*	*f*
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

- Codificação do arquivo: representar cada caracter por uma seqüência de *bits*
- Alternativas:
  1. seqüências de tamanho fixo.
  2. seqüências de tamanho variável.

Códigos de Huffman

- Qual o tamanho (em *bits*) do arquivo comprimido usando os códigos acima ?
- Códigos de tamanho fixo: $3 \times 100.000 = 300.000$
  Códigos de tamanho variável:

$$(\underbrace{45 \times 1}_{a} + \underbrace{13 \times 3}_{b} + \underbrace{12 \times 3}_{c} + \underbrace{16 \times 3}_{d} + \underbrace{9 \times 4}_{e} + \underbrace{5 \times 4}_{f}) \times 1.000 = 224.000$$

Ganho de $\approx 25\%$ em relação à solução anterior.

Problema da Codificação:

Dadas as freqüências de ocorrência dos caracteres de um arquivo, encontrar as seqüências de *bits* (códigos) para representá-los de modo que o arquivo comprimido tenha tamanho mínimo.

Códigos de Huffman

Definição:

Códigos livres de prefixo são aqueles onde, dados dois caracteres quaisquer *i* e *j* representados pela codificação, a seqüência de *bits* associada a *i* **não** é um *prefixo* da seqüência associada a *j*.

Importante:

Pode-se provar que sempre **existe** uma solução ótima do problema da codificação que é dado por um código *livre de prefixo*.

Códigos de Huffman – codificação

O processo de codificação, i.e, de geração do arquivo comprimido é sempre fácil pois reduz-se a concatenar os códigos dos caracteres presentes no arquivo original em seqüência.

Exemplo: usando a codificação de tamanho variável do exemplo anterior, o arquivo original dado por `abc` seria codificado por `0101100`.

Códigos de Huffman – decodificação

- A vantagem dos códigos livres de prefixo se torna evidente quando vamos decodificar o arquivo comprimido.
- Como nenhum código é prefixo de outro código, o código que se encontra no início do arquivo comprimido não apresenta ambigüidade. Pode-se simplesmente identificar este código inicial, traduzi-lo de volta ao caracter original e repetir o processo no restante do arquivo comprimido.
- Exemplo: usando a codificação de tamanho variável do exemplo anterior, o arquivo comprimido contendo os *bits* `001011101` divide-se de **forma unívoca** em `0 0 101 1101`, ou seja, corresponde ao arquivo original dado por *aabe*.

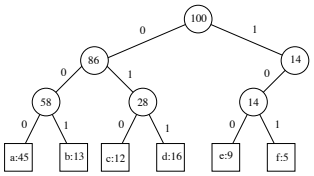
Códigos de Huffman

- Como representar de maneira conveniente uma codificação livre de prefixo de modo a facilitar o processo de decodificação?
- Solução: usar uma árvore binária.
  O filho esquerdo está associado ao *bit* ZERO enquanto o filho direito está associado ao *bit* UM. Nas folhas encontram-se os caracteres presentes no arquivo original.

Códigos de Huffman

Vejamos como ficam as árvores que representam os códigos do exemplo anterior.

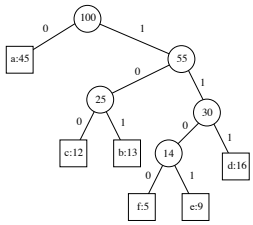
	a	*b*	*c*	*d*	*e*	*f*
Freqüência	45	13	12	16	9	5
Código fixo	000	001	010	011	100	101



Códigos de Huffman

Vejamos como ficam as árvores que representam os códigos do exemplo anterior.

	a	*b*	*c*	*d*	*e*	*f*
Freqüência	45	13	12	16	9	5
Código variável	0	101	100	111	1101	1100



Códigos de Huffman

- Pode-se mostrar (Exercício!) que uma codificação ótima sempre pode ser representada por uma árvore binária cheia, na qual cada vértice interno tem exatamente dois filhos.
- Então podemos restringir nossa atenção às árvores binárias cheias com $|C|$ folhas e $|C| - 1$ vértices internos (Exercício!), onde C é o conjunto de caracteres do alfabeto no qual está escrito o arquivo original.

Códigos de Huffman

Computando o tamanho do arquivo comprimido:

Se T é a árvore que representa a codificação, $d_T(c)$ é a profundidade da folha representado o caracter c e $f(c)$ é a sua freqüência, o tamanho do arquivo comprimido será dado por:

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

Dizemos que $B(T)$ é o **custo** da árvore T.
Isto é exatamente o tamanho do arquivo codificado.

Códigos de Huffman

- **Idéia do algoritmo de Huffman**: Começar com $|C|$ folhas e realizar sequencialmente $|C| - 1$ operações de "intercalação" de dois vértices da árvore.
  Cada uma destas intercalações dá origem a um novo vértice interno, que será o pai dos vértices que participaram da intercalação.
- A escolha do par de vértices que dará origem a intercalação em cada passo depende da soma das freqüências das folhas das subárvores com raízes nos vértices que ainda não participaram de intercalações.

Algoritmo de Huffman

Huffman(C)

▷ **Entrada:** Conjunto de caracteres C e as freqüências f dos caracteres em C.
▷ **Saída:** raiz de uma árvore binária representando uma codificação ótima livre de prefixos.

1. $n \leftarrow |C|$;
   ▷ Q é fila de prioridades dada pelas freqüências dos vértices ainda não intercalados
2. $Q \leftarrow C$;
3. **para** $i \leftarrow 1$ **até** $n - 1$ **faça**
4. **alocar novo registro** z; ▷ vértice de T
5. $z.esq \leftarrow x \leftarrow$ EXTRAI_MIN(Q);
6. $z.dir \leftarrow y \leftarrow$ EXTRAI_MIN(Q);
7. $z.f \leftarrow x.f + y.f$;
8. INSERE(Q, z);
9. **retorne** EXTRAI_MIN(Q).

Corretude do algoritmo de Huffman

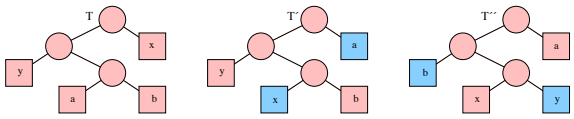
Lema 1: (escolha gulosa)

Seja C um alfabeto onde cada caracter $c \in C$ tem freqüência $f[c]$. Sejam x e y dois caracteres em C com as **menores** freqüências. Então, existe **um** código ótimo livre de prefixo para C no qual os códigos para x e y tem o mesmo comprimento e diferem apenas no último bit.

Prova do Lema 1:

- Seja T uma árvore **ótima**.
- Sejam a e b duas folhas "irmãs" (i.e. usadas em uma intercalação) mais profundas de T e x e y as folhas de T de menor freqüência.
- **Idéia:** a partir de T, obter uma outra árvore **ótima** T' com x e y sendo duas folhas "irmãs".

Corretude do algoritmo de Huffman



$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f[x]d_T(x) + f[a]d_T(a) - f[x]d_{T'}(x) - f[a]d_{T'}(a) \\ &= f[x]d_T(x) + f[a]d_T(a) - f[x]d_T(a) - f[a]d_T(x) \\ &= (f[a] - f[x])(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

Assim, $B(T) \geq B(T')$.

Analogamente $B(T') \geq B(T'')$.

Como T é ótima, T'' é ótima e o resultado vale. □

Corretude do algoritmo de Huffman

Lema 2: (subestrutura ótima)

Seja C um alfabeto com freqüência $f[c]$ definida para cada caracter $c \in C$. Sejam x e y dois caracteres de C com as menores freqüências. Seja C' o alfabeto obtido pela remoção de x e y e pela inclusão de um **novo** caracter z, ou seja, $C' = C \cup \{z\} - \{x, y\}$. As freqüências dos caracteres em $C' \cap C$ são as mesmas que em C e $f[z]$ é definida como sendo $f[z] = f[x] + f[y]$.
Seja T' uma árvore binária representado um código ótimo livre de prefixo para C'. Então a árvore binária T obtida de T' substituindo-se o vértice (folha) z pela por um vértice interno tendo x e y como fihos, representa uma código ótimo livre de prefixo para C.

Corretude do algoritmo de Huffman

Prova do Lema 2:

- Comparando os custos de T e T':
  - Se $c \in C - \{x, y\}$, $f[c]d_T(c) = f[c]d_{T'}(c)$.
  - $f[x]d_T(x) + f[y]d_T(y) = (f[x] + f[y])(d_{T'}(z) + 1) = f[z]d_{T'}(z) + (f[x] + f[y])$.
- Logo, $B(T') = B(T) - f[x] - f[y]$.
- Por contradição, suponha que existe T'' tal que $B(T'') < B(T)$.
  Pelo lema anterior, podemos supor que x e y são folhas "irmãs" em T''. Seja T''' a árvore obtida de T'' pela substituição de x e y por uma folha z com freqüência $f[z] = f[x] + f[y]$. O custo de T''' é tal que

$$B(T''') = B(T'') - f[x] - f[y] < B(T) - f[x] - f[y] = B(T'),$$

contradizendo a hipótese de que T' é uma árvore ótima para C'. □

Corretude do algoritmo de Huffman

Teorema:

O algoritmo de Huffman constrói um código ótimo (livre de prefixo).

Segue imediatamente dos Lemas 1 e 2.

Passos do projeto de algoritmos gulosos: resumo

1. Formule o problema como um **problema de otimização** no qual uma escolha é feita, restando-nos então resolver um único subproblema a resolver.
2. Provar que existe sempre uma solução ótima do problema que atende à **escolha gulosa**, ou seja, a escolha feita pelo algoritmo guloso é segura.
3. Demonstrar que, uma vez feita a escolha gulosa, o que resta a resolver é um subproblema tal que se combinarmos a resposta ótima deste subproblema com o(s) elemento(s) da escolha gulosa, chega-se à solução ótima do problema original.
   Esta é a parte que requer mais engenhosidade!
   Normalmente a prova começa com uma solução ótima *genérica* e a modificamos até que ela inclua o(s) elemento(s) identificados pela escolha gulosa.