* CÓDIGOS DE HUFFMAN * (Cormen, §16.3)

1. DEFINIÇÃO DO PROBLEMA: informalmente, nós desejamos encontrar uma codificação binária "livre de prefixos" (isto é, o código de um caractere nunca pode ser um trecho inicial do código de um caractere diferente) que minimize o tamanho da codificação resultante de um dado texto. A entrada do problema é, porém, apenas um vetor "o" indexado pelos caracteres do texto, sendo o[c] o número (POSITIVO) de ocorrências de um caractere "c" qualquer do texto. Nós supomos que "o" tem pelo menos dois elementos, isto e, que o texto tem pelo menos dois caracteres diferentes.

   Como vimos em sala, o fato de só admitirmos codificações livres de prefixos permite que representemos uma SOLUÇÃO por uma árvore binária, na qual as folhas correspondem aos caracteres da entrada. Nesse caso, o CUSTO de uma solução S é dado por

     c(S) = SOMATÓRIO_{c ∈ C} o[c] * d(c,S) ,

   onde

   * C é o conjunto dos caracteres da entrada;
   * d(c,S), ou apenas d(c), é a profundidade ("depth") de "c" na árvore S.

   Finalmente, nós dizemos que uma SOLUÇÃO ÓTIMA é uma solução de custo mínimo.

2. EXERCÍCIO: escreva uma demonstração precisa do seguinte fato, argumentado em sala: "nenhuma solução ótima S possui um nó interno com um filho apenas -- ou seja, todo nó interno de S tem exatamente dois filhos".

3. LEMA: seja S uma solução para uma entrada "o" do problema. Além disso, sejam x,a ∈ C tais que o[x] <= o[a] e d(x,S) <= d(a,S). Finalmente, seja T a solução obtida a partir de S pela troca das posições de "x" e "a". Nesse caso, temos c(T) <= c(S).

==========

PROVA:

Observe que

```
c(S) - c(T)  =   o[x]*d(x,S) + o[a]*d(a,S) + SOMATÓRIO_{c != x,a} o[c]*d(c,S) -
               ( o[x]*d(x,T) + o[a]*d(a,T) + SOMATÓRIO_{c != x,a} o[c]*d(c,T) )

  = o[x]*d(x,S) + o[a]*d(a,S) - ( o[x]*d(x,T) + o[a]*d(a,T) )     (1)

  = o[x]*d(x,S) + o[a]*d(a,S) - ( o[x]*d(a,S) + o[a]*d(x,S) )     (2)

  = ( o[x] - o[a] )*d(x,S) + ( o[a] - o[x] )*d(a,S)

  = ( o[a] - o[x] )*( d(a,S) - d(x,S) )

 >= 0 ,                                                           (3)
```

pois

  (1) d(c,T) = d(c,S) para todo c != a,x ("c" não mudou de posição na árvore);
  (2) De S para T, "x" e "a" trocaram de posição;
  (3) o[a] >= o[x] e d(a,S) >= d(x,S).

Logo, c(S) >= c(T), CQD.

==========

4. LEMA: se "o" é uma entrada do problema na qual há caracteres x != y tais que
o[x] <= o[y] <= o[c] para todo c != x,y , então existe uma solução
ótima S para "o" na qual "x" e "y" são irmãos e estão no nível mais
profundo de S.

==========
PROVA:

Seja R uma solução ótima para "o". Se "x" e "y" ocorrem como irmãos no nível mais profundo de R, então R é uma testemunha para o enunciado, CQD. Em caso contrário, seja "a" um nó do nível mais profundo de R. Como |C| >= 2, então "a" não pode ser a raiz. Além disso, pelo exercício 2, o pai de "a" possui exatamente dois filhos. Adicionalmente, o irmão de "a" tem que ser uma folha, pois em caso contrário os filhos de "a" teriam nível maior que "a", um absurdo pela escolha de "a". Seja, então, "b" o irmão de "a", e suponhamos, sem perda de generalidade, que o[a] <= o[b]. Logo, pelo enunciado, temos o[x] <= o[a] e o[y] <= o[b]. Além disso, temos d(a,R) >= d(x,R) e d(b,R) >= d(y,R). Logo, pelo lema 3 acima, a árvore S que obtemos a partir de R pela troca das posições de "x" e "a" é tal que c(S) <= c(R), e portanto também é uma solução ótima para "o". Se b = y, então "x" e "y" são irmãos no nível mais profundo de S, o que prova o resultado. Se b != y, seja T a árvore obtida a partir de S pela troca das posições de "b" e "y". Pelo lema 3, temos que c(T) <= c(S), e portanto que T também é solução ótima. Além disso, "x" e "y" são irmãos no nível mais profundo de T, o que prova o resultado.
==========

5. LEMA: Se

(1) "o" é uma entrada de tamanho >= 3 na qual há caracteres x != y tais que o[x] <= o[y] <= o[c] para todo c != x,y ;

(2) "o'" é uma entrada igual a "o", exceto por não possuir os caracteres "x" e "y", e por possuir um caractere "z" não presente em "o", sendo o'[z] = o[x] + o[y];

(3) S é uma solução ótima para o',

então a solução T obtida a partir de S pela adição de "x" e "y" como filhos esquerdo e direito de "z", respectivamente, é ótima para "o".

==========
PROVA:

Observe primeiramente que, sendo R = SOMATÓRIO_{c != x,y,z} o[c] * d(c,S), então

```
  c(S) = R + o'[z] * d(z,S)
       = R + (o[x] + o[y]) * d(z,S)
```

e

```
  c(T) = R + o[x]*d(x,T)       + o[y]*d(y,T)
       = R + o[x]*(d(z,S) + 1) + o[y]*(d(z,S) + 1)
       = R + (o[x] + o[y]) * d(z,S) + o[x] + o[y]
       = c(S) + o[x] + o[y] .
```

Suponha agora, por absurdo, que T não é uma solução ótima para "o". Logo, pelo lema 4 acima existe uma solução ótima T* para "o" tal que "x" e "y" ocorrem como irmãos no nível mais profundo de T*. Seja, então, S* a solução para o' que se obtém a partir de T* pela substituição da subárvore enraizada no pai de "x" e "y" pela folha "z". Analogamente a acima, segue que

```
  c(T*) = c(S*) + o[x] + o[y] .
```

Logo, como c(T*) < c(T), então

  c(S*) + o[x] + o[y] < c(S) + o[x] + o[y] => c(S*) < c(S) ,

ou seja, S não é solução ótima para o', um absurdo com a hipótese "2" do enunciado.

Logo, T é necessariamente ótima para "o", CQD.
==========

6. ALGORITMO DE HUFFMAN, baseado no lema acima:

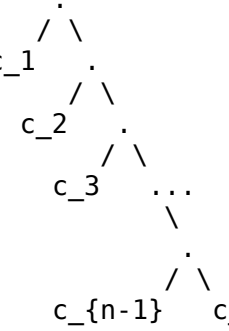
```
==========================================================================
Algoritmo: huffman
Entrada: o vetor o[1..n] descrito na definição do problema.
Saída: uma árvore binária.
--------------------------------------------------------------------------
01. Q := monte ("heap") de mínimo vazio cujos elementos são "n" folhas; cada
        folha corresponde a um caractere "c" de "o" e tem prioridade o[c].
02. FAÇA n-1 vezes
03. | px := prioridade_mín(Q) , x := remover_mín(Q)
04. | py := prioridade_mín(Q) , y := remover_mín(Q)
05. | Crie um nó "z" cujos filhos esquerdo e direito sejam "x" e "y"
06. | Insira z em Q com prioridade px+py
07. RETORNE remover_mín(Q).
==========================================================================
```

O algoritmo acima pode ser implementado de forma a executar em tempo O(n*lg n), pois a linha 1 custa O(n) (construção de um monte binário) e cada uma das n-1 iterações custa O(lg n).

7. EXERCÍCIOS:

   (a) Certifique-se de que você entende o algoritmo acima e os lemas nos quais ele se baseia. (Uma opção é tentar reescrever tudo sem consulta.)

   (b) Em sala foi considerada a estratégia abaixo para o mesmo problema, onde c_1 ... c_n são os caracteres da entrada considerados em ordem decrescente de frequência. Para que tipo de entrada essa estratégia necessariamente leva a soluções ótimas?



   (c) Veja mais em: http://en.wikipedia.org/wiki/Huffman_coding
                      http://dx.doi.org/10.1109%2FJRPROC.1952.273898