* Árvores de Busca Ótimas * (Cormen, §15.5)

1. UMA DEFINIÇÃO DO PROBLEMA (*): dada uma árvore binária de busca "A" de "n" nós, seja f[i] o nível -- isto é, a soma entre 1 e a profundidade -- do i-ésimo nó de A, da esquerda para a direita. Além disso, dado também um vetor p[1..n] de números reais quaisquer, seja

   C(A,p) = SOMATÓRIO_{i=1}^n f[i] * p[i] ,

   ou simplesmente C(A), o custo de A. O objetivo do problema é então, dado apenas um vetor p[1..n], encontrar uma árvore binária de busca A de custo mínimo.

   NA VERSÃO SIMPLIFICADA DO PROBLEMA, o objetivo é apenas retornar o custo c(A) de uma árvore A de custo mínimo.

   (*) : A definição do problema no livro-texto é um pouco diferente, mas a ideia da solução é a mesma; veja os últimos exercícios desta nota de aula.

2. OBSERVAÇÃO: o custo de uma árvore A cuja raiz é o k-ésimo nó é

   C(A) = p[k] + SOMATÓRIO_{i = 1}^{k-1} f[i] * p[i]
               + SOMATÓRIO_{i = k+1}^n   f[i] * p[i].

   Além disso, para todo i != k, seja f'[i] o nível do i-ésimo nó de A NA SUBÁRVORE ESQUERDA (se i < k) / DIREITA (se i > k) da raiz de A. Claramente, f[i] = f'[i] + 1, e portanto

   C(A) = p[k] + SOMATÓRIO_{i = 1}^{k-1} (1 + f'[i]) * p[i]
               + SOMATÓRIO_{i = k+1}^n   (1 + f'[i]) * p[i]

        = p[k] + (SOMATÓRIO_{i = 1}^{k-1} p[i])
               + (SOMATÓRIO_{i = k+1}^n   p[i])
               + (SOMATÓRIO_{i = 1}^{k-1} f'[i] * p[i])
               + (SOMATÓRIO_{i = k+1}^n   f'[i] * p[i])

        = ( SOMATÓRIO_{i=1}^n p[i] ) + C(SE,p[1..k-1]) + C(SD,p[k+1..n]) ,

   onde SE e SD denotam respectivamente as subárvores esquerda e direita da raiz de A.

   Por fim, observe ainda que A tem o menor custo dentre todas as árvores cuja raiz é o k-ésimo nó se e somente se tanto SE quanto SD têm custo mínimo. Isso leva ao algoritmo recursivo seguinte.

3. UM ALGORITMO DE DIVISÃO-E-CONQUISTA:

```
============================================================
Algoritmo: abo_dq
Entrada: vetor p[1..n].
Saída: um número real.
------------------------------------------------------------
01. SE n = 1
02. | RETORNE p[1].
03. somap := 0
04. PARA k DE 1 A n
    | somap := somap + p[k]
05. cmín := mín { somap + abo_dq(p[2..n]) , somap + abo_dq(p[1..n-1]) }
07. PARA k DE 2 A n-1
08. | custok := somap + abo_dq(p[1..k-1]) + abo_dq(p[k+1..n])
09. | cmín := mín { cmín , custok }
10. RETORNE cmín.
============================================================
```

4. EXERCÍCIO: como vimos em sala, o algoritmo acima faz muitos cálculos repetidos. Prove então que ele executa em tempo Ω(g), para alguma função "g" superpolinomial em "n" (como, por exemplo, uma função exponencial).

5. UM ALGORITMO DE PROGRAMAÇÃO DINÂMICA:

```
============================================================
Algoritmo: abo_pd
Entrada: vetor p[1..n].
Saída: um número real.
------------------------------------------------------------
01. PARA i DE 1 A n
02. | M[i,i] := p[i]
03. PARA t DE 2 A n
04. | PARA i DE 1 A n -t +1
05. | | j := i +t -1 , somap := 0
06. | | PARA k DE i A j
07. | | | somap := somap + p[k]
08. | | cmín := mín { somap + M[i+1,j] , somap + M[i,j-1] }
09. | | PARA k DE i+1 A j-1
10. | | | cmín := mín { cmín , somap + M[i,k-1] + M[k+1,j] }
11. | | M[i,j] := cmín
12. RETORNE M[1,n].
============================================================
```

6. EXERCÍCIO: claramente, o algoritmo acima executa em tempo O(n^3) e utiliza θ(n^2) de memória auxiliar. Para completar a análise, mostre que ele executa também em tempo Ω(n^3).

7. EXERCÍCIOS:

   a) Estenda o algoritmo "abo_pd" de forma que ele não apenas retorne o custo de uma árvore de custo mínimo, mas também a árvore em si. Dica: uma maneira conveniente de fazer isso é preencher uma matriz "R" n x n tal que R[i,j] é o índice do nó raiz da árvore A de menor custo C(A,p[i..j]).

   b) Na versão do problema definida no livro-texto, além das chaves k_1 ... k_n que são buscadas com frequências p[1] ... p[n], há também nós d_0 ... d_n que correspondem a buscas que sejam feitas na árvore por chaves diferentes de k_1 ... k_n. Mais especificamente, d_i corresponde a uma busca que seja feita por uma chave maior que k_i e menor que k_{i+1}. A entrada do problema consiste então no vetor p[1..n] e num vetor q[0..n], onde q[i] é a probabilidade de a busca corresponder a d_i (isto é, a uma chave entre k_i e k_{i+1}). Nesse caso, uma solução para o problema é uma árvore binária de busca cujos nós k_1 ... k_n sejam internos e os nós d_0 ... d_n sejam folhas, cada d_i estando entre k_i e k_{i+1}. O custo de uma árvore A é dado por

      C(A,p,q) = SOMATÓRIO_{i=1}^n f[i] * p[i] + SOMATÓRIO_{i=0}^n f[i] * q[i] ,

      e o objetivo do problema é encontrar uma árvore de custo mínimo. Mostre então que esse problema admite solução análoga à que demos para a versão que utilizamos acima.

   c) Num artigo de 1971 (http://dx.doi.org/10.1007%2FBF00264289), Knuth mostrou que sempre existem raízes de subárvores ótimas tais que raiz[i,j-1] <= raiz[i,j] <= raiz[i+1,j], para 1 <= i < j <= n. Prove esse fato, e então o utilize para obter uma variação O(n^2) do algoritmo acima.

   d) Leia mais em http://en.wikipedia.org/wiki/Optimal_binary_search_tree .