Tópicos Avançados em Algoritmos
uma resolução sumária da Prova III

Nota. **Ver o enunciado completo na página da disciplina.**

1. **Verdadeiro ou falso?** V F V V F V V

2. **Multiplicação óptima de matrizes**

   (a) **Usando indução matemática em n mostre que para $n \geq 7$, $f(n) \geq 2^n$. Pode usar o facto $f(7) = 132$.**
   R: seja $P(n)$ a propriedade a demonstrar. A demonstração é por indução em n.

   - Caso base, $n = 7$, temos $f(7) = 132 > 2^7 = 128$.
   - Vamos demonstrar a implicação $P(n-1) \Rightarrow P(n)$.

   $$\begin{array}{rcll} f(n) & = & \sum_{1 \leq p < n} f(p) f(n-p) & \\ & \geq & f(1)f(n-1) + f(n-1)f(1) & \text{(excluiram-se termos)} \\ & = & 2f(n-1) & (f(1) = 1) \\ & \geq & 2 \times 2^{n-1} & \text{(hipótese indutiva)} \\ & = & 2^n & \end{array}$$

   Temos portanto $[f(n-1) \geq 2^{n-1}] \Rightarrow [f(n) \geq 2^n]$.

   (b) **A utilização da Programação Dinâmica permite obter um algoritmo polinomial em n para determinar a "parentização" óptima de um produto de n matrizes. Descreva em linhas gerais esse algoritmo e mostre que a sua eficiência é $O(n^3)$.**
   R: Ver apontamentos. Relativamente à eficiência, repare-se que o algoritmo baseado na Programação Dinâmica é da forma

```
for k=2 to n          // produtos de 2,3,...,n matrizes   n-1 vezes
  for i=1 to n-k+1    // início do "bloco"              n-k+1 vezes
    determine o sub-produto óptimo...  //                 k-1 comparações
```

   pelo que é evidente que o número total de comparações não excede $(n-1)(n-1)(n-1) \in O(n^3)$.

3. **Procurar x**
   Considere o problema de procurar um valor x num vector `v[1..n]` (não necessariamente ordenado) com n elementos distintos. A resposta é o índice `i` tal que `v[i]`$= x$, ou `-1` se x não ocorre em `v`. Utiliza-se o modelo externo dos dados com a análise no pior caso; todos os acessos aos dados são comparações entre x e elementos do vector (cada comparação tem 3 resultados possíveis). Seja $c(n)$ o número de acessos aos dados (comparações) no pior caso.

   (a) **Usando o Princípio da Informação necessária, determine um minorante de $c(n)$.**
   R: Admitindo que cada comparação pode fornecer 1 de 3 resultados[1], com $c(n)$ comparações pode haver, no máximo, $3^{c(n)}$ resultados (comportamentos distintos do algoritmo). Como o número possível de resultados é $n+1$, temos $3^{c(n)} \geq n+1$, $c(n) \geq \lceil \log_3(n+1) \rceil$.

   (b) **Determine o majorante seguinte: $c(n) \leq n$.**
   R: o algoritmo da pesquisa sequencial faz, no máximo, n comparações, donde resulta o minorante pretendido

```
function pesq(x,v[1..n]):
  for i=1 to n:
    if x==v[i]:   //   n comparações (máximo)
      return i
  return -1
```

   (c) **Mostre que se verifica o minorante $c(n) \geq n$.**
   R: Por contradição, vamos supor que x não é comparado com um dos elementos do vector, seja `v[i]`. Vamos mostrar que há dados (x e vector `v`) que levam o algoritmo a dar uma resposta errada. Nas hipóteses seguintes estamos sempre a supor que x é diferente de todos os `v[j]` com `j`$\neq$`i`:

   - Se o algoritmo responder `-1`, fazemos $x =$`v[i]` e a resposta está errada.
   - Se o algoritmo responder `i`, fazemos x diferente de `v[i]`: a resposta está errada.
   - Finalmente, se o algoritmo responder `j` com `j`$\neq$`i`, a resposta está naturalmente errada.

   Assim, o algoritmo tem que comparar x com todos os valores de `v`, o que obriga a n comparações.

   (d) **Suponha agora que o vector está ordenado e que se usa a pesquisa binária. Mostre que o número de comparações $c(n)$ satisfaz $c(n) \leq \log(n+1)$, onde se supõe que n é da forma $n = 2^p - 1$. Que conclui neste caso (pesquisa binária) do minorante obtido na alínea 3a?**
   R: O número de comparações expresso em termos de p satisfaz a recorrência

   $$\begin{cases} c(1) = 1 \\ c(p) = 1 + c(p-1) \quad \text{para } p \geq 2 \end{cases}$$

   cuja solução é $c(p) = p$; em termos de n vem $c(n) = \log(n+1)$. Comparando com o minorante referido, verificamos que a distância entre eles não é muito grande, essencialmente o majorante iguala o minorante multiplicado por $\log 3$.

4. **Comparações no "merge"**

[1] Uma análise mais cuidada permitiria melhorar este minorante: quando o resultado de uma comparação entre x e `v[i]` é `igual`, qualquer algoritmo pode terminar imediatamente.

(a) Explique porque é que o número de respostas possíveis é igual ao número de sequências de $m+n$ caracteres `u` ou `v`, com m caracteres `u` e n caracteres v.
R: É claro que o algoritmo de "merge" especifica univocamente uma sequência da forma indicada: basta produzir `u` ou `v` conforme o elemento seleccionado para ser colocado no vector de saída seja originário de `u[]` ou de `v[]`, respectivamente. Reciprocamente, a partir dos dados de uma qualquer palavra da forma indicada, podemos obter o vector que resulta do "merge" dos 2 vectores dados. Assim o número possível de respostas é $\binom{m+n}{m} = \binom{m+n}{n}$.

(b) Usando o Princípio da Informação Necessária determine um minorante do número de comparações efectuadas entre elementos de `u` e elementos de `v`.
R: Dado que não há elementos comuns aos 2 vectores, cada comparação pode ter apenas 2 resultados. Temos então

$$\begin{aligned} 2^{c(n)} &\geq \binom{m+n}{m} \\ c(n) &\geq \left\lceil \log \binom{m+n}{m} \right\rceil \end{aligned}$$

(c) Supondo que $m = n$ e usando a fórmula de Stirling, $n! \approx \sqrt{2\pi n}(n/e)^n$, determine uma forma fechada (sem usar factoriais nem combinações) para o minorante obtido na alínea anterior.
R: vem

$$\begin{aligned} c(n) &\geq \left\lceil \log \binom{2n}{n} \right\rceil \\ &\approx \log \left(\frac{\sqrt{4\pi n}(2n/e)^{2n}}{2\pi n (n/e)^{2n}} \right) \\ &= \log \left(\frac{2^{2n}}{\sqrt{\pi n}} \right) \\ &= 2n - \frac{1}{2}\log(\pi n) \end{aligned}$$

Como o algoritmo clássico do "merge" efectua, no máximo, $2n-1$ comparações, este minorante é bastante bom.

5. Transformando uma palavra noutra
Mostre que a distância entre duas "strings" s e t é dada por

$$d(s,t) = (m - \mathtt{ms}(s,t)) + (n - \mathtt{ms}(s,t)) = m + n - 2 \times \mathtt{ms}(s,t)$$

onde $\mathtt{ms}(s,t)$ é o comprimento da maior sequência (não necessariamente consecutiva) comum às "strings" s e t.
R: consideremos uma transformação de s em t.

- Os caracteres de s que se mantêm em t formam uma determinada sequência w comum (não necessariamente consecutiva) a s e a t,
- Os caracteres de s que não pertencem a w têm que ser eliminados, o que obriga a $|s| - |w|$ eliminações.
- Os caracteres de t que não pertencem a w têm que ser inseridos, o que obriga a $|t| - |w|$ eliminações.
- Numa transformação arbitrária pode haver outras operações: caracteres que são inseridos e posteriormente eliminados, mas essas operações nunca fazem parte de uma transformação mínima.

Em resumo, qualquer transformação mínima tem necessariamente custo $(|s| - |w|) + (|t| - |w|)$; como $|s|$ e $|t|$ são fixos, o custo mínimo obtém-se maximizando $|w|$, o comprimento da sequência comum, donde resulta o resultado pretendido. Note-se que, como podem existir várias sequências de comprimento máximo, podem também existir várias transformações de custo mínimo.