Instituto Superior Técnico

Análise e Síntese de Algoritmos

Ano Lectivo 2020/2021 Repescagem do 2º Teste - versão A

RESOLUÇÃO

I. (2.5 + 2.5 + 2.5 + 2.5 = 10 val.)

I.a) Considere o algoritmo de Knuth-Morris-Pratt com o seguinte texto $T = aababbabaab$ e o padrão $P = ababaa$. Calcule a função de prefixo para o padrão P:

$P =$	a	b	a	b	a	a
q	1	2	3	4	5	6
$\pi[q]$	**0**	**0**	**1**	**2**	**3**	**1**

Indique ainda todas as posições da sequência de índices q que percorre o padrão P, no final de cada iteração do ciclo principal.

q	0	**1**	**1**	**2**	**3**	**4**	**0**	**1**	**2**	**3**	**1**	**2**

I.b) Considere a maior sub-sequência comum entre as duas strings $ACABCAB$ e $BAACBCBC$ e calcule a respectiva matriz de programação dinâmica $c[i,j]$ para este problema, em que o índice i está associado à string $ACABCAB$. Indique os seguintes valores: $c[1,3]$, $c[2,6]$, $c[3,3]$, $c[4,4]$, $c[5,6]$, $c[6,3]$, $c[7,8]$. Indique ainda o número de maior sub-sequências comuns.

c[1,3]	c[2,6]	c[3,3]	c[4,4]	c[5,6]	c[6,3]	c[7,8]
1	**2**	**2**	**2**	**4**	**2**	**5**

Número de maior sub-sequências comuns: **2**

I.c) Considere o problema de compressão de dados de um ficheiro usando a codificação de Huffman. Indique o código livre de prefixo óptimo para cada carácter num ficheiro com 10 000 caracteres com a seguinte frequência de ocorrências:
$f(a) = 9, f(b) = 13, f(c) = 24, f(d) = 11, f(e) = 37, f(f) = 6$.

Quando constrói a árvore, atribue o bit 0 para o nó com menor frequência. Em caso de empate, atribua o bit 0 ao nó que inclui o caracter que aparece primeiro por ordem alfabética. Analogamente, em caso de empate na *min-priority queue*, considera-se primeiro o nó que inclui o caracter que aparece primeiro por ordem alfabética.

Indique também o total de bits no ficheiro codificado.

	a	b	c	d	e	f
Codificação	**001**	**011**	**10**	**010**	**11**	**000**

Total Bits	**23 900**

I.d) Considere o seguinte programa linear:

$$\begin{array}{rrrcr} \min & -3x_1 & -7x_2 & + & 1 \\ \text{s.a} & x_1 & +2x_2 & \leq & 3 \\ & -2x_1 & -4x_2 & \geq & -6 \\ & & x_1, x_2 & \geq & 0 \end{array}$$

Aplique o algoritmo Simplex para calcular o valor da função objectivo e o respectivo valor das variáveis básicas e não-básicas na solução óptima. Em caso de empate na escolha da variável de entrada ou da variável de saída, aplique a regra de Bland (ou seja, escolha a variável de menor índice).

Z	x_1	x_2	x_3	x_4
$-\frac{19}{2}$	0	$\frac{3}{2}$	0	0

II. (3 + 2 + 2 + 3 = 10 val.)

II.a) Dada uma sequência de inteiros positivos $\langle x_1, ..., x_n \rangle$, pretende desenvolver-se um algoritmo que determina o maior valor suceptível de ser obtido a partir da expressão $x_1/x_2/x_3/.../x_n$, determinando a ordem pela qual as divisões devem ser efectuadas. Por exemplo, dada a sequência $\langle 16, 8, 4, 2 \rangle$, a parentização que resulta no maior valor final é: $(16/((8/4)/2)) = 16$.

1. Seja $M[i,j]$ o maior valor que é possível obter a partir da expressão $x_i/x_{i+1}/.../x_j$ e $m[i,j]$ o menor valor. Por exemplo, dada a sequência $\langle 16, 8, 4, 2 \rangle$, $M[1,4] = 16$ e $m[1,4] = 0.25$. Admitindo que a sequência dada como input é $\langle x_1, ..., x_n \rangle$, defina $M[i,j]$ e $m[i,j]$ recursivamente completando os campos em baixo:

$$M(i,j) = \begin{cases} \boxed{} & \text{se } i = j \\ \boxed{} & \text{se } j > i \end{cases}$$

$$m(i,j) = \begin{cases} \boxed{} & \text{se } j = i \\ \boxed{} & \text{se } j > i \end{cases}$$

2. Complete o template de código em baixo que, dada uma sequência de inteiros $\langle x_1, ..., x_n \rangle$, calcula $m[1,n]$ e $M[1,n]$.

```
GreatestValue(x[1..n])
  let M[1..n, 1..n] be a new matrix of size n × n
  let m[1..n, 1..n] be a new matrix of size n × n
  for i = 1 to n do
    M[i, i] := [          ]
    m[i, i] := [          ]
  endfor
  for s = 1 to n−1 do
    for i = 1 to n−s do
      [                                        ]
    endfor
  endfor
  return M[1, n]
```

3. Determine a complexidade assimptótica do algoritmo proposto na alínea anterior.

Solução:

1.

$$M(i,j) = \begin{cases} x[i] & \text{se } i = j \\ \mathbf{max}\{M[i,k]/m[k+1,j] \mid i \leq k < j\} & \text{se } j > i \end{cases}$$

$$m(i,j) = \begin{cases} x[i] & \text{se } j = i \\ \mathbf{min}\{m[i,k]/M[k+1,j] \mid i \leq k < j\} & \text{se } j > i \end{cases}$$

2.

```
GreatestValue(x[1..n])
  let M[1..n, 1..n] be a new matrix of size n × n
  let m[1..n, 1..n] be a new matrix of size n × n
  for i = 1 to n do
    M[i, i] := x[i]
    m[i, i]  := x[i]
  endfor
  for s = 1 to n−1 do
    for i = 1 to n−s do
      let j = i + s
      let M[i, j] = −∞
      let m[i, j] = +∞
      for k = i to j−1 do
        M[i, j] := max(M[i, j], M[i, k]/m[k + 1, j])
        m[i, j] := min(m[i, j], m[i, k]/M[k + 1, j])
      endfor
    endfor
  endfor
  return M[1, n]
```

3. Complexidade: $O(n^3)$. O algoritmo tem de preencher a metade diagonal superior das matrizes $M[1..n, 1..n]$ e $m[1..n, 1..n]$, sendo que para cada posição da matriz o algoritmo pode percorrer s posições. Formalmente:

$$\begin{aligned}
&\textstyle\sum_{s=1}^{n-1} \sum_{i=1}^{n-s} \sum_{k=i}^{j-1} O(1) \\
&\quad = \textstyle\sum_{s=1}^{n-1} \sum_{i=1}^{n-s} \sum_{k=i}^{i+s-1} O(1) \\
&\quad = \textstyle\sum_{s=1}^{n-1} \sum_{i=1}^{n-s} O(1).(i + s - 1 - i + 1) \\
&\quad = \textstyle\sum_{s=1}^{n-1} \sum_{i=1}^{n-s} O(s) \\
&\quad = \textstyle\sum_{s=1}^{n-1} O(s).(n - s) \\
&\quad = O(\textstyle\sum_{s=1}^{n-1} n.s - s^3) \\
&\quad \leq O(n. \textstyle\sum_{s=1}^{n-1} s) \\
&\quad \leq O(n^3)
\end{aligned}$$

II.b) Uma fábrica de barras de cereais produz e vende dois tipos de barras: premium e standard. O preço de venda das barras premium é 5 euros por embalagem, enquanto o preço de venda das barras standard é 3 euros por embalagem. Dada a elevada procura de barras por parte dos distribuidores, a fábrica tem sempre conseguido escoar a totalidade da produção. Assim sendo, a produção está apenas limitada pela capacidade dos fornos usados para torrar a mistura de cereais e pelo capital disponível para a compra de matérias primas.

As barras premium requerem 3 horas de tempo de forno por embalagem, enquanto as barras standard requerem 4 horas, sendo que existem 20.000 horas de tempo de forno disponível por embalagem durante o período de um mês.

Os custos directos decorrentes da produção de uma embalagem de barras são: 2 euros por cada embalagem de barras premium e 1 euro por cada embalagem de barras standard. A fábrica dispõe de 4000 euros por mês para investir em produção. Contudo, 45% do rendimento obtido da venda de barras premium e 30% do rendimento obtido das venda de barras standard estará disponível para ser re-investido na produção de mais barras durante o próprio mês.

Finalmente, obrigações contratuais da fábrica com a autarquia onde está instalada exigem que a produção mensal seja superior a 2000 embalagens.

O director de operações da fábrica pretende agora determinar o número de embalagens de cada um dos tipos de barras a produzir mensalmente por forma a maximizar a facturação.

1. Formule o programa linear que permite resolver este problema.
2. A solução básica inicial do programa linear é exequível? Caso não seja, formule o programa linear auxiliar.
3. Formule o programa linear dual.

Solução:

1. Começamos por identificar as variáveis do problema:
   - x_1 - número de embalagens premium produzidas;
   - x_2 - número de embalagens standard produzidas.

   Programa linear primal:

$$\begin{array}{rrrcl} \max & 5x_1 & +3x_2 & & \\ \text{s.a} & 3x_1 & +4x_2 & \leq & 20000 \\ & -0.25x_1 & +0.1x_2 & \leq & 4000 \\ & -x_1 & -x_2 & \leq & -2001 \\ & & x_1, x_2 & \geq & 0 \end{array}$$

2. A solução básica inicial não é exequível (a terceira restrição não é satisfeita). Programa linear auxiliar:

$$\begin{array}{rrrrcl} \max & -x_0 & & & & \\ \text{s.a} & 3x_1 & +4x_2 & -x_0 & \leq & 20000 \\ & -0.25x_1 & +0.1x_2 & -x_0 & \leq & 4000 \\ & -x_1 & -x_2 & -x_0 & \leq & -2001 \\ & & x_1, x_2 & -x_0 & \geq & 0 \end{array}$$

3. Programa linear dual:

$$
\begin{array}{lrrrcl}
\max & 20000y_1 & +4000y_2 & -2001y_3 & & \\
\text{s.a} & +3y_1 & -0.25y_2 & -1y_3 & \geq & 5 \\
 & +4y_1 & +0.1y_2 & -1y_3 & \geq & 3 \\
 & & y_1, y_2, y_3 & & \geq & 0
\end{array}
$$

II.c) Considere o algoritmo de Knuth-Morris-Pratt para o emparelhamento de cadeias de caracteres. Seja $n \in \mathbb{N}$ e P o padrão $\underline{aba}\ \underline{ab^2a}\ \underline{ab^3a}\ \ldots \underline{ab^{n-1}a}\ \underline{ab^na}$ tal que $a \neq b$, $a, b \in \Sigma$ e $n \geq 2$. Considere o cálculo da função de prefixo $\pi[i]$ para o padrão P. Indique em função de n para quantos valores diferentes de i é que temos que:

1. $\pi[i] = 0$
2. $\pi[i] = 1$
3. $\pi[i] = 2$
4. $\pi[i] > 2$

Deve apresentar os cálculos.

Solução:

1. $\pi[i] = 0$. Number: $2 + \sum_{i=1}^{n-1} i = 2 + \frac{(n-1).n}{2} = \frac{n^2-n+4}{2}$
2. $\pi[i] = 1$. Number: $2 * (n-1) + 1 = 2n - 2 + 1 = 2n - 1$
3. $\pi[i] = 2$. Number: $n - 1$
4. $\pi[i] > 2$. Number: 0

II.d) Recorde o problema da mochila não fraccionária sem repetição estudado nas aulas. Dada uma mochila com capacidade K e n items com pesos $p_1, ..., p_n$ e valores $v_1, ..., v_n$, o problema consiste em determinar o valor máximo que podemos transportar na mochila respeitando a sua restrição de capacidade. Este problema pode ser modelado como o seguinte problema de decisão:

$$\textbf{Knapsack} = \{\langle K, \vec{p}, \vec{v}, v^* \rangle \mid \exists I \subseteq \{1, ..., n\}. \sum_{i \in I} v_i = v^* \wedge \sum_{i \in I} p_i \leq K\}$$

1. Mostre que o problema **Knapsack** está em **NP**.
2. Mostre que o problema **Knapsack** é **NP**-difícil por redução a partir do problema **Subset-Sum** que se recorda em baixo.

Problema Subset-Sum: Seja $\mathcal{X} = \{v_1, ..., v_n\}$ e k um inteiro arbitrário; o problema **Subset-Sum**, define-se formalmente da seguinte maneira:

$$\textbf{Subset-Sum} = \{\langle \mathcal{X}, k \rangle \mid \exists I \subseteq \{1, ..., n\}. \sum_{i \in I} v_i = k\}$$

Solução:

1. O algoritmo de verificação recebe como input uma possível instância $\langle K, \vec{p}, \vec{v}, v^* \rangle$ e um certificado na forma de um conjunto de índices I tal que: $\sum_{i \in I} v_i = v^*$ e $\sum_{i \in I} p_i \leq K$. Em primeiro lugar, observamos que o certificado tem tamanho $O(n)$. O algoritmo de verificação tem de verificar $\sum_{i \in I} v_i = v^*$ e $\sum_{i \in I} p_i \leq K$, o que pode ser feito em tempo $O(n)$.
2. Dada uma instância $\langle \mathcal{X}, k \rangle$ do problema **Subset-Sum** temos de construir uma instância $\langle K, \vec{p}, \vec{v}, v^* \rangle$ do problema **Knapsack** tal que:

   $$\langle \mathcal{X}, k \rangle \in \textbf{Subset-Sum} \Leftrightarrow \langle K, \vec{p}, \vec{v}, v^* \rangle \in \textbf{Knapsack}$$

   Admitindo que os elementos de X se encontram numerados: $X = \{x_1, ..., x_n\}$, definimos formalmente a instância $\langle K, \vec{p}, \vec{v}, v^* \rangle$ como se segue:

   - $K = k$
   - $\vec{p} = \langle x_1, ..., x_n \rangle$
   - $\vec{v} = \langle x_1, ..., x_n \rangle$
   - $v^* = k$

   A redução proposta tem complexidade: $O(n)$.