Aula Prática 5

ASA 2024/2025

Q1 (EE 19/20): Uma *subsequência contígua* de uma dada sequência de inteiros S é uma subsequência de S constituída unicamente por elementos consecutivos de S. Dizemos que o valor de uma subsequência contígua corresponde à soma dos inteiros que a constituem. Por exemplo, dada a sequência $S = \langle 5, 15, -30, 10, -5, 40, 10\rangle$, $S' = \langle 5, 15, -30\rangle$ e $S'' = \langle 10, -5, 40, 10\rangle$ são subsequências contíguas de S com valores -10 e 55, respectivamente. Pretende-se determinar um algoritmo que, dada uma sequência de inteiros S, calcula a subsequência contígua de S de valor máximo.

1. Seja **sub**(i) o máximo de entre os valores de todas as subquências contíguas de S que terminam na posição i. Defina **sub**(i) recursivamente completando os campos em baixo.

$$\mathbf{sub}(i) = \begin{cases} S[i] & \text{se } \boxed{} \\ \boxed{} & \text{caso contrário} \end{cases}$$

2. Complete o template de código em baixo que calcula a quantidade **sub**(i) para $1 \leq i \leq n$ e indique a respectiva complexidade assimptótica.

```
MaxSubSeq(S[1..n])
  let sub[0..n] be a vector of size n+1
  sub[0] = 0
  for i = 1 to n do
    [                    ]
  endfor
  return sub
```

3. Explique como determinar o valor da subsequência contígua de S de valor máximo a partir do vector **sub**.

Solução:

1. $\mathbf{sub}(i) = \begin{cases} S[i] & \text{se } i = 1 \ \lor \ \mathbf{sub}(i-1) \leq 0 \\ S[i] + \mathbf{sub}(i-1) & \text{caso contrário} \end{cases}$

2. Em baixo o código completo:

```
MaxSubSeq(S[1..n])
  let sub[0..n] be a vector of size n+1
  sub[0] = 0
  for i = 1 to n do
    if(sub[i − 1] ≤ 0) then
      sub[i] = S[i]
    else
      sub[i] = sub[i − 1] + S[i]
    endif
  endfor
  return sub
```

   Complexidade: $O(n)$

3. A partir do vector *sub*, calculamos o valor da subsequência contígua de S de valor máximo da seguinte maneira:

$$\max(0, \max\{sub(i) \mid 1 \leq i \leq n\})$$

Q2 (T2 19/20): O Eng. Caracol foi encarregado de apresentar uma proposta para a construção de postos de primeiros socorros ao longo da autoestrada AX, que começa no kilómetro 0 e termina no kilómetro k. O Eng. Caracol dispõe de uma lista de n locais candidatos, ordenada por distância. Cada local candidato, $1 \leq i \leq n$, é associado à sua distância ao kilómetro 0, d_i, e ao seu custo estimado de construção, c_i. Sabendo que dois postos de primeiros socorros consecutivos não podem estar a uma distância superior a D kilómetros, o objectivo do Eng. Caracol é determinar o conjunto de locais candidatos que satisfazem a restrição do problema pelo menor custo possível.

1. Seja $O(i)$ o custo da solução óptima para o troço da autoestrada AX entre o kilómetro 0 e o local candidato i, que atribui necessariamente um posto de primeiros socorros ao local candidato i. Defina $O(i)$ recursivamente, completando os campos abaixo.

$$O(i) = \begin{cases} \boxed{} & \text{se } i > 1 \\ \boxed{} & \text{caso contrário} \end{cases}$$

2. Complete o template de código em baixo que calcula a quantidade $O(i)$ para $1 \leq i \leq n$ e indique a respectiva complexidade assimptótica.

```
FindO(c[1..n], d[1..n])
   let O[1..n] be a new vector of size n
   O[1] = c[1]
   for i = 2 to n do
      [                    ]
   endfor
   return O
```

3. Explique como determinar o custo da melhor solução, da construção até ao kilómetro k, a partir do vector O.

Solução:

1. $O(i) = \begin{cases} \min\{O(j) + c_i \mid j < i \ \wedge \ d_i - d_j \leq D\} & \text{se } i > 1 \\ c_1 & \text{caso contrário} \end{cases}$

2. Em baixo o código completo:

```
FindO(c[1..n], d[1..n])
  let O[1..n] be a new vector of size n
  O[1] = c[1]
  for i = 2 to n do
    O[i] = ∞
    for j = i − 1 to 1 do
      if (d[i] − d[j] < D) then
        if (O[i] > O[j] + c[i]) then
          O[i] = O[j] + c[i]
        endif
      endif
    endfor
  endfor
  return O
```

   Complexidade: $O(n^2)$

3. A partir do vector O, calculamos o valor pretendido da seguinte forma:

$$\min\{O(j) \mid \ k - d_j \leq D\}$$

   O valor mínimo pretendido corresponde a $O(n)$, que pode ser calculado em tempo linear.

Q3 (R2 19/20): Acredita-se que uma data string de texto $T[1..n]$ corresponde a uma versão corrompida de uma string de texto original à qual foram removidos os espaços entre as palavras; por exemplo "Aquelapraiaextasiadaenua". O Eng. João Caracol foi encarregado de verificar se é possível obter uma sequência de palavras válidas a partir da string de texto T usando um algoritmo baseado em programação dinâmica. Para tal, dispõe de uma função de dicionário **dict** que, dada uma cadeia de caracteres s, verifica se s é uma palavra válida; formalmente:

$$\mathbf{dict}(s) = \begin{cases} 1 & \text{se } s \text{ é uma palavra válida} \\ 0 & \text{caso contrário} \end{cases}$$

1. Seja $B(i)$ o valor booleano que indica se a cadeia de caracteres $T[1..i]$ forma uma sequência de palavras válidas. Defina $B(i)$ recursivamente completando os campos em baixo.

$$B(i) = \begin{cases} \boxed{} & \text{se } i \geq 1 \\ \boxed{} & \text{se } i = 0 \end{cases}$$

2. Complete o template de código em baixo que calcula a quantidade $B(n)$ e indique a respectiva complexidade assimptótica.

```
FindWords(t[1..n])
  let B[0..n] be a new array of size n
  B[0] = true
  for i = 1 to n do
    [ ]
  endfor
  return B[n]
```

Solução:

1. $$B(i) = \begin{cases} \bigvee\{B(j) \wedge \mathbf{dict}(t[(j+1)..i]) \mid j < i\} & \text{se } i \geq 1 \\ \mathbf{true} & \text{se } i = 0 \end{cases}$$

2. Em baixo o código completo:

```
FindWords(t[1..n])
  let B[0..n] be a new array of size n+1
  B[0] = true
  for i = 1 to n do
    B[i] = false
    j = i − 1
    while j ≥ 0 ∧ (not B[i]) do
      B[i] = B[j] ∧ dict(t[(j + 1)..i])
      j = j−1
    endwhile
  endfor
  return B[n]
```

Complexidade: $O(n^2)$

Q4 (T2 20/21): Uma sequência diz-se um *palíndromo* se é simétrica, isto é, se permanece igual quando lida de trás para diante; por exemplo, são palíndromos as sequências: a, aa, $abbba$ e $abbaabba$. Pretende-se desenvolver um algoritmo que, dada uma sequência de caracteres arbitrária, retorne o tamanho do maior palíndromo que esta contém. Por exemplo, dada a sequência $abbaabbababaabc$, o algoritmo deve retornar 8, que corresponde ao tamanho do palíndromo $abbaabba$.

1. Seja $x[1..n]$ a string de texto dada como input e $B(i,j)$ o valor Booleano que indica se a cadeia de caracteres $x[i..j]$ forma um palíndromo. Defina $B(i,j)$ recursivamente completando os campos em baixo:

$$B(i,j) = \begin{cases} \textbf{true} & \text{se } j < i \\ \boxed{} & \text{se } j = i \\ \boxed{} & \text{c.c.} \end{cases}$$

   Admite-se, para simplificar a formulação, que $B(i,j) = \textbf{true}$ quando $j < i$.

2. Complete o template de código em baixo que calcula o tamanho do maior palíndromo contido no array dado como input, $x[1..n]$. Para obter a cotação máxima, o algoritmo deve retornar o valor pretendido assim que encontra o palíndromo de tamanho máximo, não devendo de efectuar o preenchimento completo da matriz $B[1..n, 1..n]$.

   `BiggestPalindromeSize`$(x[1..n])$
   let $B[1..n, 1..n]$ be a new matrix of size $n \times n$ with all cells initialised to true

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

Solução:

1. $B(i,j) = \begin{cases} \textbf{true} & \text{se } j \leq i \\ B(i+1, j-1) \ \wedge \ (x[i] == x[j]) & \text{c.c.} \end{cases}$

2.

```
BiggestPalindromeSize(x[1..n])
  let B[1..n, 1..n] be a new matrix of size n × n with all cells initialised to true
  let not_found = 0
  for s = 1 to n−1 do
    let found = false
    for i = 1 to n−s do
      B[i, i+s] := B[i + 1, (i+s)−1] && (x[i] == x[i+s])
      found := found || B[i, i+s]
    endfor
    if(not found){
      not_found := not_found+1
      if(not_found == 2) return s−1
    } else not_found := 0
  endfor
  if(not_found == 1) return n−1
  else return n
```

3. Complexidade: $O(n^2)$. No pior caso o algoritmo terá de preencher a metade diagonal superior da matriz B.

Q5 (R2 20/21): 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:

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

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

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:

$$\begin{aligned}
&\textstyle\sum_{s=1}^{n-1} \sum_{i=1}^{n-s} \sum_{k=i}^{j-1} O(1) \\
&\quad = \textstyle\sum_{s=1}^{n-1} \sum_{i=1}^{n-s} \sum_{k=i}^{i+s-1} O(1) \\
&\quad = \textstyle\sum_{s=1}^{n-1} \sum_{i=1}^{n-s} O(1).(i + s - 1 - i + 1) \\
&\quad = \textstyle\sum_{s=1}^{n-1} \sum_{i=1}^{n-s} O(s) \\
&\quad = \textstyle\sum_{s=1}^{n-1} O(s).(n - s) \\
&\quad = O(\textstyle\sum_{s=1}^{n-1} n.s - s^2) \\
&\quad \leq O(n. \textstyle\sum_{s=1}^{n-1} s) \\
&\quad \leq O(n^3)
\end{aligned}$$

Q6 (EE 20/21): Dadas duas sequências de caracteres $\vec{X} = \langle X_1, ..., X_n \rangle$ e $\vec{Z} = \langle Z_1, ..., Z_k \rangle$, $\vec{Z}$ diz-se uma *subsequência contígua* de $\vec{X}$ se existir um inteiro $0 \leq i < n$ tal que: $X_{i+1} = Z_1$, $X_{i+2} = Z_2$, ..., $X_{i+k} = Z_k$. Por exemplo, a sequência de caracteres *abb* é uma subsequência contígua de *ababb* (basta escolher o deslocamento $i = 2$).

Dadas duas sequências de caracteres $\vec{X} = \langle X_1, ..., X_n \rangle$ e $\vec{Y} = \langle Y_1, ..., Y_m \rangle$, pretende desenvolver-se um algoritmo que determine o tamanho da sua maior subsequência contígua comum.

1. Dadas duas sequências de caracteres $\vec{X} = \langle X_1, ..., X_n \rangle$ e $\vec{Y} = \langle Y_1, ..., Y_m \rangle$, seja $B(i,j)$ o tamanho do maior sufixo comum entre $\langle X_1, ..., X_i \rangle$ e $\langle Y_1, ..., Y_j \rangle$. Por exemplo, para $\vec{X} = abaabb$ e $\vec{Y} = abbbbb$, temos que $B(3,3) = 0$ e $B(6,3) = 3$. Defina $B(i,j)$ recursivamente completando os campos em baixo:

   $$B(i,j) = \begin{cases} 0 & \text{se } i = 0 \lor j = 0 \\ \boxed{} & \text{se } \boxed{} \\ \boxed{} & \text{c.c.} \end{cases}$$

   Admite-se, para simplificar a formulação, que $B(i,j) = 0$ quando $i = 0$ ou $j = 0$.

2. Complete o template de código em baixo que, dadas duas sequências de caracteres $\langle X_1, ..., X_n \rangle$ e $\langle Y_1, ..., Y_m \rangle$, calcula o tamanho da sua maior subsequência contígua comum.

```
LongestContiguousCommonSubstring(x[1..n], y[1..m])
  let B[0..n, 0..m] be a new matrix of size (n + 1) × (m + 1)
  B[0, 0] := [          ]
  for i = 1 to n do
    B[i, 0] := [          ]
  endfor
  for j = 1 to m do
    B[0, j] := [          ]
  endfor
  let max = 0
  for i = 1 to n do
    for j = 1 to m do
      [                                        ]
    endfor
  endfor
  return max
```

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

Solução:

1.
$$B(i,j) = \begin{cases} 0 & \text{se } i = 0 \vee j = 0 \\ B(i-1, j-1) + 1 & \text{se } X_i = Y_j \\ 0 & \text{c.c.} \end{cases}$$

2.

LongestContiguousCommonSubstring($x[1..n], y[1..m]$)
  let $B[0..n, 0..m]$ be a new matrix of size $(n+1) \times (m+1)$
  $B[0,0] := 0$
  for $i = 1$ **to** n **do**
    $B[i,0] := 0$
  endfor
  for $j = 1$ **to** m **do**
    $B[0,j] := 0$
  endfor
  let $max = 0$
  for $i = 1$ **to** n **do**
    for $j = 1$ **to** m **do**
      if $x[i] == y[j]$ **then**
        $B[i,j] := B[i-1, j-1] + 1$
        $max := \mathbf{max}(B[i,j], max)$
      else $B[i,j] := 0$
    endfor
  endfor
  return max

3. Complexidade: $O(n^2)$. O algoritmo tem de preencher toda a matriz $B[0..n, 0..m]$. O preenchimento de cada célula faz-se em tempo constante, $O(1)$.