Instituto Superior Técnico

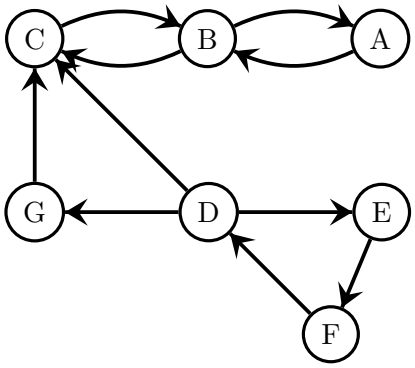
Análise e Síntese de Algoritmos

Ano Lectivo 2020/2021 Exame de Época Especial

RESOLUÇÃO

I. (1.5 + 2 + 2 + 1.5 + 1.5 + 1.5 = 10 val.)

I.a) Considere o grafo dirigido da figura.

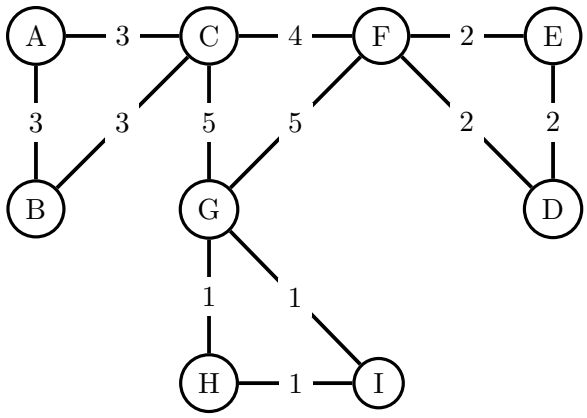


Aplique o algoritmo de Tarjan para identificar os componentes fortemente ligados, considerando o vértice **D** como inicial. Durante a aplicação do algoritmo considere que tanto a escolha dos vértices a visitar, como a pesquisa dos vértices adjacentes são feitas por ordem lexicográfica (ou seja, A, B, C, ...).

Indique os componentes fortemente ligados do grafo pela ordem segundo a qual são identificados pelo algoritmo e o valor *low* calculado para cada vértice. Considere que o tempo de descoberta *d* começa em 1.

	A	B	C	D	E	F	G
low()	**3**	**2**	**2**	**1**	**1**	**1**	**7**
SCCs :	$\{\mathbf{C}, \mathbf{B}, \mathbf{A}\}, \{\mathbf{G}\}, \{\mathbf{F}, \mathbf{E}, \mathbf{D}\}$						

I.b) Considere o grafo não dirigido e pesado da figura.



Considere a execução do algoritmo de Kruskal para determinar árvores abrangentes de menor custo. Durante a aplicação do algoritmo, arcos com o mesmo peso devem ser considerados por ordem lexicográfica.

Utilize a estrutura em árvore para representação de conjuntos disjuntos com a aplicação das heuristicas de união por categoria e compressão de caminhos. Para cada vértice indique os valores de categoria ($rank[v]$) e o valor do seu pai na árvore que representa os conjuntos ($p[v]$).

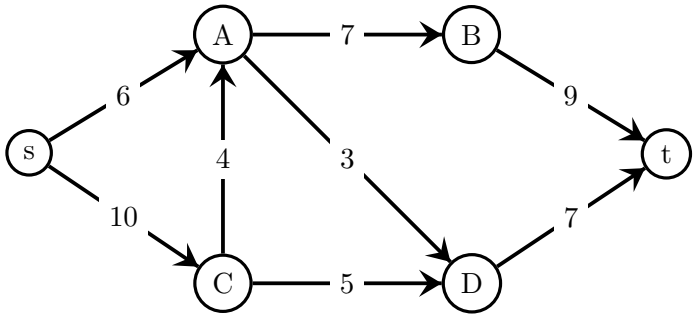
Nota: Na operação $Make\text{-}Set(v)$, o valor da categoria de v é inicializado a 0. Na operação de $Union(u, v)$, em caso de empate, considere que o representante de v é que fica na raíz.

	A	B	C	D	E	F	G	H	I
$rank[v]$	**0**	**1**	**0**	**0**	**2**	**0**	**0**	**1**	**0**
$p[v]$	**B**	**E**	**E**	**E**	**E**	**E**	**E**	**E**	**H**

Indique ainda o peso da árvore abrangente, bem como o número de total de árvores abrangentes.

Pesos da MST:	**21**
Número de MSTs:	**54**

I.c) Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede. Aplique o algoritmo Relabel-To-Front na rede de fluxo. Considere que as listas de vizinhos dos vértices intermédios são as seguintes:
$N[A] =< B, D, s, C >$
$N[B] =< t, A >$
$N[C] =< D, A, s >$
$N[D] =< t, A, C >$
e que a lista de vértices inicial é $L =< B, D, A, C >$.



Indique a altura final de cada vértice, um corte mínimo da rede, o valor do fluxo máximo, e a sequência de <u>diferentes</u> configurações de L.

	s	A	B	C	D	t
$h()$	**6**	**7**	**1**	**7**	**7**	**0**

Corte :	$\{s, A, C, D\}/\{B, t\}$	f(S,T) =**14**

L:	$< B, D, A, C >, < A, B, D, C >, < B, A, D, C >, < C, B, A, D >,$ $< A, C, B, D >, < D, A, C, B >, < A, D, C, B >, < D, A, C, B >,$ $< A, D, C, B >, < D, A, C, B >, < A, D, C, B >$

Indique ainda o valor do fluxo final de cada aresta da rede de fluxo.

f(s,A)	f(s,C)	f(A,B)	f(A,D)	f(B,t)	f(C,A)	f(C,D)	f(D,t)
5	**9**	**7**	**2**	**7**	**4**	**5**	**7**

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

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

Número de maior sub-sequências comuns: **3**

I.e) 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:
$f(a) = 13, f(b) = 8, f(c) = 5, f(d) = 25, f(e) = 12, f(f) = 37$.

Quando constrói a árvore, considere 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	**011**	**001**	**000**	**10**	**010**	**11**

Total Bits	**23.800**

I.f) Considere o padrão $P = abbabb$ e construa o autómato finito que emparelhe este padrão numa cadeia de caracteres. Indique o estado resultante das seguintes transições:

$\delta(0, a)$	$\delta(1, a)$	$\delta(2, a)$	$\delta(3, b)$	$\delta(4, a)$	$\delta(5, a)$	$\delta(6, a)$	$\delta(6, b)$
1	**1**	**1**	**0**	**1**	**1**	**4**	**0**

Ilustre a sua aplicação no seguinte texto $T = aabbaabbabbabba$.

q	0	**1**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**5**	**6**	**4**	**5**	**6**	**4**

Indique o número total de matches e as posições onde estes ocorrem.

Número total de matches:	**2**
Matches nas posições:	**6 e 9**

II. (1,5 + 1,5 + 2 + 1,5 + 2 + 1,5 = 10 val.)

II.a) Considere a função recursiva:

```
int f(int n) {
  int i = 0, j = n;

  if (n <= 1) return 1;

  while(j > 1) {
    i++;
    j = j / 2;
  }

  for (int k = 0; k < 8; k++)
    j += f(n/2);

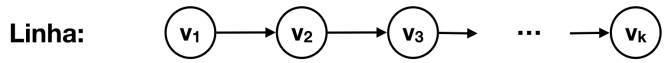
  while (i > 0) {
    j = j + 2;
    i--;
  }
  return j;
}
```

1. Determine o menor majorante assimptótico medido em função do parâmetro n para o número total de iterações dos loops **1** e **2** por cada chamada à função f.
2. Determine o menor majorante assimptótico da função f, em função do parâmetro n, utilizando os métodos que conhece.

Solução:

1. Consideramos os dois loops separadamente.
   - *Loop 1*: $O(log\ n)$
   - *Loop 2*: $O(log\ n)$
2. Equação do tempo:
   $$T(n) = 8.T(n/2) + O(log\ n)$$
   Observando que $log_b\ a = log_2\ 8 = 3$ e que $log\ n \in O(n^{3-\epsilon})$, aplicamos o Teorema Mestre na forma geral, concluindo que: $T(n) \in O(n^3)$.

II.b) Dizemos que um grafo dirigido forma uma *linha* se consiste numa sequência linear de vértices como se ilustra na figura em baixo:



Pretende desenvolver-se um algoritmo que, dado um grafo dirigido $G = (V, E)$, determina se G forma uma linha. Admitindo que o grafo G dado como input é representado com base em listas de adjacências:

1. Proponha um algoritmo que retorna uma lista com os vértices *source* de G e indique a respectiva complexidade assimptótica. Não é necessário apresentar o pseudo-código.

2. Proponha um algoritmo para determinar se G forma uma linha. Deve apresentar o pseudo-código do algoritmo proposto e indicar a respectiva complexidade assimptótica.

Solução:

1. Criamos um vector $\vec{v}$ com n posições, onde $n = |V|$, incializado a 0. Intuitivamente, cada posição i do vector $\vec{v}$ deverá guardar o número de arcos incidentes no vértice i. Para tal, percorremos as listas de adjacências do grafo G, ao processarmos cada arco (i, j), incrementamos de uma unidade a posição j do vector $\vec{v}$ ($\vec{v}[j] = \vec{v}[j]+1$). No final, percorremos o vector $\vec{v}$, inserindo na lista L a retornar os vértices para os quais $\vec{v}$ tem o valor 0; isto é, L deverá conter todos os vértices j tais que $\vec{v}[j] = 0$. *Complexidade:* $O(V + E)$

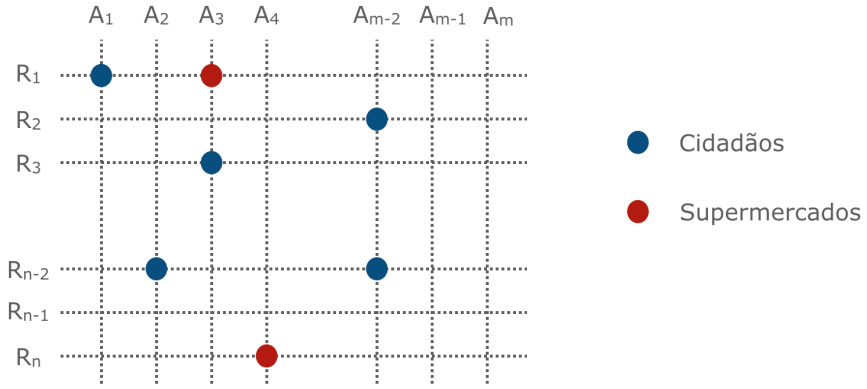
2. **function** `CheckLine(`G`)`

```
1:  v ← findSource(G)
2:  if v == null then
3:     return false
4:  end if
5:  visited[1, .., n] ← new array of size |G.V|
6:  count ← 1
7:  while true do
8:     if G.Adj[v].size > 1 || visited[v] == true then
9:        return false
10:    else if G.Adj[v].size == 0 then
11:       break
12:    else
13:       visited[v] ← true
14:       count ← count + 1
15:       v ← G.Adj[v][0]
16:    end if
17: end while
18: return count == |G.V|
```

   Complexidade: o corpo do ciclo While é percorrido no máximo uma vez por cada vértice pelo que o custo do algoritmo é dominado pela operação `findSource` que tem complexidade $O(V + E)$.

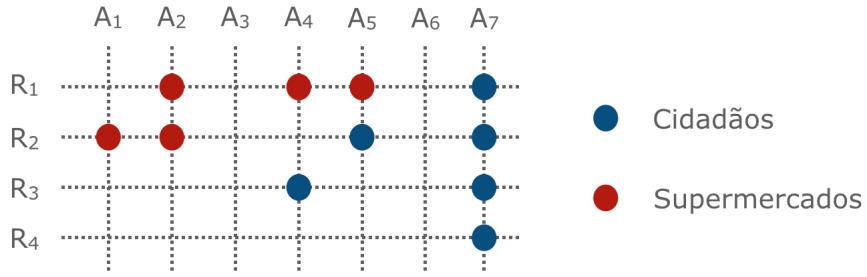
II.c) Por causa do COVID-19, o Eng. António Caracol foi encarregue de projectar um sistema que permita aos cidadãos de Manhattan deslocar-se a um supermercado sem se cruzarem com nenhum outro cidadão. Estão disponíveis no mapa da cidade, as localizações dos supermercados, que se situam todos em esquinas e as moradas dos cidadãos que, para este efeito, se situam também nas esquinas.

Tratando-se de Manhattan, as ruas têm um arranjo em quadriculado absolutamente regular, e considera-se que em todas as ruas se circula em ambos os sentidos. As avenidas estão numeradas de 1 a m, enquanto que as ruas estão numeradas de 1 a n, como se ilustra em baixo:



Os cruzamentos são definidos por um par de números, sendo que o par (i, j) corresponde ao cruzamento da rua i com a avenida j.

Dados um conjunto de supermercados abertos $\{S_1, ..., S_k\}$ e de cidadãos que querem fazer compras a uma dada hora $\{C_1, ..., C_l\}$, o sistema deverá determinar qual o número máximo de cidadãos que pode deslocar-se a um supermercado, sem correr o risco de se encontrar com outro cidadão, numa rua, avenida ou cruzamento, inicial, intermédio ou final do seu percurso. Por exemplo, dada a grelha ortogonal em baixo, apenas 4 cidadãos se podem deslocar a um supermercado simultaneamente sem se cruzarem.



Dados adicionais: Podem existir dois supermercados no mesmo cruzamento, mas apenas um deles poderá ser usado numa soluação, para evitar contactos nesse local. Dois ou mais cidadãos podem morar no mesmo cruzamento, mas apenas um deles poderá sair à rua de cada vez, os que ficam em casa não levantam problemas de contágio. Da mesma forma, se um ou mais cidadãos morarem num cruzamento mas não saírem à rua, o cruzamento pode ser usado por outro cidadão para passar ou aceder a um supermercado, nesse ou noutro cruzamento.

1. Modele o problema descrito em cima como um problema de fluxo máximo.

2. Indique o algoritmo que utilizaria para a calcular o fluxo máximo, bem como a respectiva complexidade assimptótica medida em função dos parâmetros do problema: número de avenidas m, número de ruas n, número de supermercados k e

número de cidadãos l. Pode admitir que $l >> k$ e $l >> n.m$. De entre os algoritmos de fluxo estudados nas aulas deve escolher aquele que garanta a complexidade assimptótica mais baixa para o problema em questão.
Nota: A resposta deverá necessariamente incluir as expressões que definem o número de vértices e de arcos da rede de fluxo proposta ($|V|$ e $|E|$, respectivamente) em função dos parâmetros do problema, bem como um upper-bound para o valor do fluxo máximo.

Solução:

1. *Construção da rede de fluxo:* $G = (V, E, c, s, t)$. Na construção da rede de fluxo consideramos dois vértices por cruzamento (um vértice de entrada e um vértice de saída), um vértice por residente, um vértice por supermercado e dois vértices adicionais s e t, respectivamente a fonte e o sumidouro. Formalmente:

   - $$\begin{array}{rll} V = & \{s,t\} & \text{fonte e sumidouro} \\ & \cup\, \{S_i \mid 1 \le i \le k\} & \text{supermercados} \\ & \cup\, \{C_i \mid 1 \le i \le l\} & \text{cidadãos} \\ & \cup\, \{P_{ij}, P'_{ij} \mid 1 \le i \le n, 1 \le j \le m\} & \text{cruzamentos} \end{array}$$

   - $$\begin{array}{rll} E = & \{(s, C_i, 1) \mid 1 \le i \le l\} & C_i \text{ quer ir a um supermercado} \\ & \cup\{(C_o, P_{ij}, 1) \mid 1 \le o \le l\} & \text{se } C_o \text{ vive no cruzamento } (i,j) \\ & \cup\, \{(P'_{ij}, S_o, 1) \mid 1 \le o \le k\} & \text{se } S_o \text{ se encontra em } (i,j) \\ & \cup\, \{(S_o, t, 1) \mid 1 \le o \le k\} & \\ & \cup\, \{(P_{ij}, P'_{ij}, 1) \mid 1 \le i \le m, 1 \le j \le n\} & \text{uma pessoa por cruzamento} \\ & \cup\, \{(P'_{ij}, P_{(i+1)j}, 1) \mid 1 \le i < n, 1 \le j \le m\} & \text{ligar ao cruzamento de baixo} \\ & \cup\, \{(P'_{ij}, P_{(i-1)j}, 1) \mid 1 < i \le n, 1 \le j \le m\} & \text{ligar ao cruzamento de cima} \\ & \cup\, \{(P'_{ij}, P_{i(j-1)}, 1) \mid 1 \le i \le n, 1 < j \le m\} & \text{ligar ao cruzamento da esquerda} \\ & \cup\, \{(P'_{ij}, P_{i(j+1)}, 1) \mid 1 \le i \le n, 1 \le j < m\} & \text{ligar ao cruzamento da direita} \end{array}$$

2. *Complexidade:*

   - $|V| = 2.n.m + k + l + 2 = O(n.m + l) = O(l)$
   - $|E| = 2.l + 2.k + n.m + (n-1) * m * 2 + n * (m-1) * 2 = O(n.m + l) = O(l)$
   - $|f^*| \le k = O(k)$
   - Edmonds Karp (upper bound de FF): $O(|f^*|.E) = O(k.l)$
   - Edmonds Karp (upper bound EK): $O(E^2.V) = O(l^3)$
   - Relabel-To-Front: $O(l^3)$

   O algoritmo a utilizar é o algoritmo de Edmonds-Karp.

II.d) Considere o seguinte programa linear:

$$\begin{array}{lrrcl} \max & x_1 & +x_2 & & \\ \text{s.a} & -x_1 & -x_2 & \leq & -1 \\ & -x_1 & +x_2 & \leq & 5 \\ & 4x_1 & +x_2 & \leq & 20 \\ & & x_1, x_2 & \geq & 0 \end{array}$$

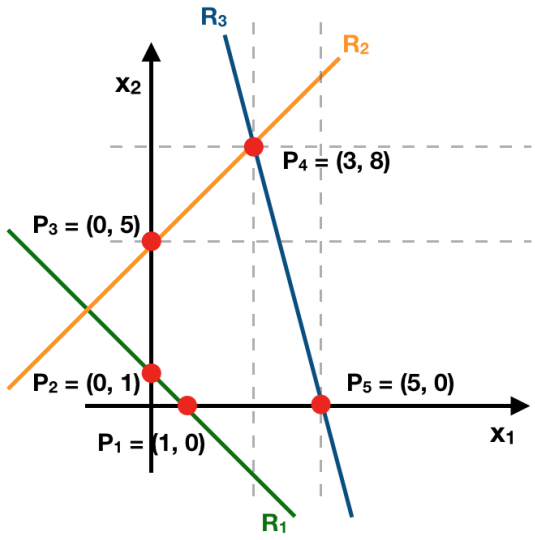
1. Desenhe o conjunto exequível e resolva geometricamente o programa linear. A resposta deve incluir: o valor máximo, as coordenadas onde esse valor é atingido e as equações das rectas que delimitam a região exequível.

2. Formule o programa linear auxiliar e indique duas soluções diferentes para o mesmo.

3. Formule o programa linear dual e calcule a respectiva solução a partir da solução do programa primal. Indique tanto o valor mínimo como as coordenadas onde esse valor é atingido.

Solução:

1. Representamos a região exequível no diagrama em baixo.



   O Teorema Fundamental da Programação Linear estabelece que o valor óptimo da função objectivo, a existir, ocorre num vértice da região exequível. Assim sendo, concluímos que o valor óptimo é 11 e ocorre no vértice $P_4 = (3, 8)$. Equações das rectas que delimitam o conjunto exequível:

   - **R1:** $x_2 = 1 - x_1$
   - **R2:** $x_2 = 5 + x_1$
   - **R3:** $x_2 = 20 - 4.x_1$

2. O programa linear auxiliar é definido em baixo:

$$\begin{array}{lrrrcl} \max & -x_0 & & & & \\ \text{s.a} & -x_1 & -x_2 & -x_0 & \leq & -1 \\ & -x_1 & +x_2 & -x_0 & \leq & 5 \\ & 4x_1 & +x_2 & -x_0 & \leq & 20 \\ & & x_1, x_2, x_0 & & \geq & 0 \end{array}$$

Qualquer ponto da região exequível é uma solução do programa auxiliar. Assim podemos considerar, por exemplo, quaisquer dois pontos de P_1, P_2, P_3, P_4 e P_5, definidos na figura (estendidos com a coordenada $x_0 = 0$):

- $P_1' = (1, 0, 0)$
- $P_2' = (0, 1, 0)$
- $P_3' = (0, 5, 0)$
- $P_4' = (3, 8, 0)$
- $P_5' = (5, 0, 0)$

3. O programa linear dual é definido em baixo:

$$\begin{array}{rrrrrr} \min & -1y_1 & +5y_2 & +20y_3 & & \\ \text{s.a} & -y_1 & -y_2 & +4y_3 & \geq & 1 \\ & -y_1 & +y_2 & +y_3 & \geq & 1 \\ & & y_1, y_2, y_3 & & \geq & 0 \end{array}$$

Do Teorema da Dualidade Forte concluímos que o valor mínimo do programa dual coincide com o valor máximo do programa primal, 11. Da inspecção da geometria do programa primal, concluímos que as restrições activas no vértice da solução correspondem às variáveis y_2 e y_3 do problema dual. Segue, por isso, que $y_1 = 0$ no ponto óptimo do problema dual. Resolvendo o sistema:

$$\begin{cases} -y_2 + 4y_3 = 1 \\ y_2 + y_3 = 1 \end{cases}$$

concluímos que o valor mínimo do programa dual se encontra no ponto $(0, 3/5, 2/5)$.

II.e) 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:

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

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

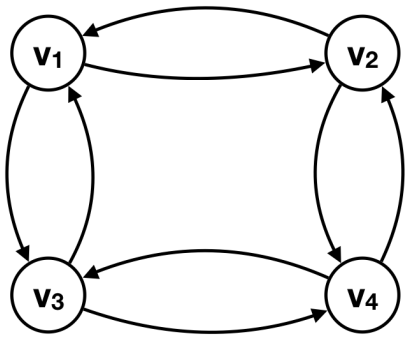
2. ```
   LongestContiguousCommonSubstring(x[1..n], y[1..m])
     let B[0..n, 0..m] be a new matrix of size (n + 1) × (m + 1)
     B[0, 0] := 0
     for i = 1 to n do
       B[i, 0] := 0
     endfor
     for j = 1 to m do
       B[0, j] := 0
     endfor
     let max = 0
     for i = 1 to n do
       for j = 1 to m do
         if x[i] == y[j] then
           B[i, j] := B[i − 1, j − 1] + 1
           max := 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, $O(1)$.
   ```

II.f) Dado um grafo dirigido $G = (V, E)$, o problema **LongestPath** consiste em determinar o tamanho do caminho mais longo entre quaisquer dois vértices em G que não passa pelo mesmo vértice duas vezes. Formalmente, o problema **LongestPath** pode ser modelado através do seguinte problema de decisão:

$$\textbf{LongestPath} = \{\langle G, k\rangle \mid G \text{ contém um caminho de tamanho } k \text{ sem vértices repetidos}\}$$

1. Ao contrário do problema do caminho mais curto, o problema do caminho mais longo não possui sub-estrutura óptima. Explique a afirmação fornecendo um exemplo com base no grafo que se ilustra em baixo.



2. Mostre que o problema **LongestPath** está em **NP**.

3. Mostre que o problema **LongestPath** é NP-difícil por redução a partir do problema **HamiltonianPath** que se define em baixo e que é sabido tratar-se de um problema NP-completo. Não é necessário provar formalmente a equivalência entre os dois problemas; é suficiente indicar a redução e a respectiva complexidade.

Problema do Caminho Hamiltoniano: Um grafo dirigido $G = (V, E)$ contém um *caminho Hamiltoniano* sse existe uma sequência de vértices $\langle v_{i_1}, ..., v_{i_n}\rangle$ em G tal que:

$$V = \{v_{i_1}, ..., v_{i_n}\} \ \wedge \ n = |V| \ \wedge \ \forall_{i=1,..,n-1} \,.\, (v_i, v_{i+1}) \in E$$

O problema do caminho Hamiltoniano, **HamPath**, define-se formalmente da seguinte maneira:

$$\textbf{HamPath} = \{\langle G\rangle \mid G \text{ contém um caminho Hamiltoniano}\}$$

Solução:

1. O problema do caminho mais longo não tem sub-estrutura óptima porque um caminho mais longo não é necessariamente constituído por caminhos mais longos. Por exemplo, a sequência $\langle v_1, v_2, v_4\rangle$ é um caminho mais longo entre v_1 e v_4. Contudo, $\langle v_2, v_4\rangle$ não é um caminho mais longo entre v_2 e v_4. O caminho mais longo entre v_2 e v_4 é: $\langle v_2, v_1, v_3, v_4\rangle$.

2. O algoritmo de verificação recebe como input uma possível instância $\langle G, k\rangle$ e uma sequência de vértices em G, $\vec{v} = \langle v_{i_1}, ..., v_{i_k}\rangle$, que constitui o certificado. O algoritmo tem de verificar que:

   - *Restrição 1:* A sequência $\vec{v}$ contém todos os vértices de G e não contém vértices repetidos;
   - *Restrição 2:* Cada dois vértices consecutivos de $\vec{v}$ correspondem a um arco de G.

Em primeiro lugar, observamos que o certificado tem tamanho $O(V)$. Analisamos cada restrição separadamente:

- *Restrição 1:* Percorrer a sequência $\vec{v}$ mantendo um vector de Booleanos de tamanho n, inicialmente inicializados a $false$; sempre que um vértice é encontrado, colocamos o seu Booleano correspondente a $true$; se encontrarmos um vértice cujo Booleano seja $true$, retornamos $false$. Se não encontrarmos nenhum vértice repetido, verificamos adicionalmente que o tamanho de $\vec{v}$ coincide com o número de vértices. Complexidade: $O(V)$.
- *Restrição 2:* Percorrer a sequência $\vec{v}$. Verificar para cada dois vértices consecutivos que $(v_i, v_{i+1}) \in E$. Complexidade: $O(E)$, no pior caso percorremos todos os arcos do grafo.

3. Dada uma instância $\langle G \rangle$ do problema **HamPath** temos de construir uma instância $\langle G', k \rangle$ do problema **LongestPath** tal que:

$$\langle G \rangle \in \textbf{HamPath} \Leftrightarrow \langle G', k \rangle \in \textbf{LongestPath}$$

Para tal, basta escolher $G' = G$ e $k = |V|$. Complexidade da redução: $O(V + E)$.