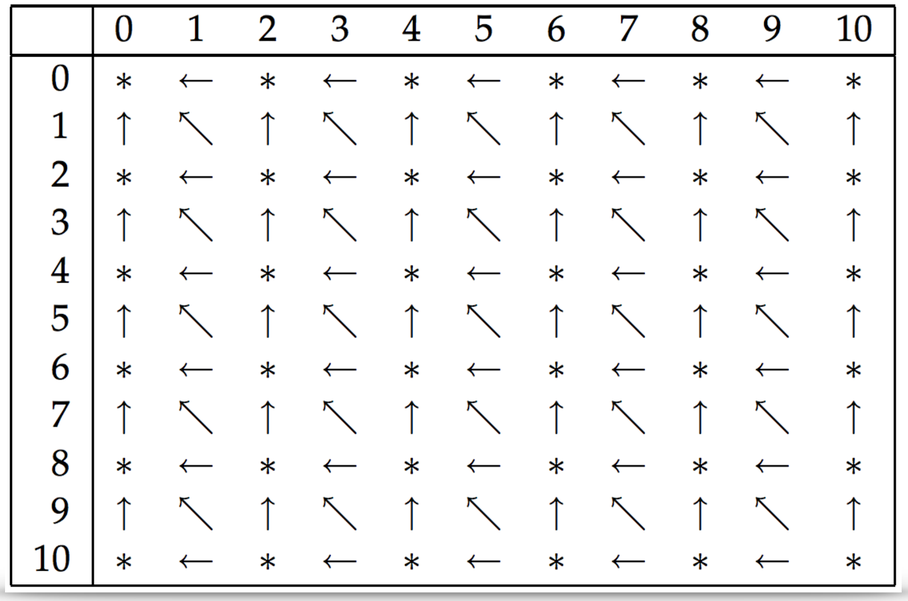
Profa. Dra. Raquel C. de Melo-Minardi
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais



	0	1	2	3	4	5	6	7	8	9	10
0	*	←	*	←	*	←	*	←	*	←	*
1	↑	↖	↑	↖	↑	↖	↑	↖	↑	↖	↑
2	*	←	*	←	*	←	*	←	*	←	*
3	↑	↖	↑	↖	↑	↖	↑	↖	↑	↖	↑
4	*	←	*	←	*	←	*	←	*	←	*
5	↑	↖	↑	↖	↑	↖	↑	↖	↑	↖	↑
6	*	←	*	←	*	←	*	←	*	←	*
7	↑	↖	↑	↖	↑	↖	↑	↖	↑	↖	↑
8	*	←	*	←	*	←	*	←	*	←	*
9	↑	↖	↑	↖	↑	↖	↑	↖	↑	↖	↑
10	*	←	*	←	*	←	*	←	*	←	*

MÓDULO 4
ALGORITMOS PARA BIOINFORMÁTICA
Implementação do algoritmo de Needleman-Wunsch

PROGRAMA PRINCIPAL

```
# PROGRAMA PRINCIPAL
v = ['*', 'A', 'T', 'C', 'G', 'T', 'A', 'C']
w = ['*', 'A', 'T', 'G', 'T', 'T', 'A', 'T']
lcs(v, w)
```

- Esse programa declara e inicializa dois arranjos contendo as sequências *v* e *w* a serem alinhadas e chama a função “lcs" (*Longest Common Subsequence*) passando esses dois arranjos por referência
  - Colocamos um “*“ no início de cada sequência para que os índices da lista estejam em conformidade com o número do caractere na sequência e ainda para acomodar a primeira linha e a primeira coluna que são compostas de “0“s

FUNÇÃO LCS

```
1  def lcs(v, w):
2      pontuacao = []
3      ponteiros = []
4      pontuacao = [0]*len(v)
5      ponteiros = ['']*len(v)

6      for i in range(0, len(w)):
7          pontuacao[i] = [0]*len(v)
8          ponteiros[i] = ['']*len(v)

9      for i in range(0, len(w)):
10         ponteiros[i][0] = '|'
11     for j in range(0, len(v)):
12         ponteiros[0][j] = '_'
```

- Na linha (1), recebemos como argumento as referências para duas listas com as sequências *v* e *w*
- Em (2-3) declaramos duas matrizes que receberão as pontuações dos alinhamentos “pontuacao” e os ponteiros para recuperar o caminho do alinhamento máximo “ponteiros“. Note que a matriz ilustrada anteriormente é aqui armazenada em duas matrizes diferentes indexadas pelos mesmos índices
- Em (4-8), inicializamos com “0”s a matriz pontuação e com vazio a matriz de ponteiros

FUNÇÃO LCS

```
1  def lcs(v, w):
2      pontuacao = []
3      ponteiros = []
4      pontuacao = [0]*len(v)
5      ponteiros = ['']*len(v)

6      for i in range(0, len(w)):
7          pontuacao[i] = [0]*len(v)
8          ponteiros[i] = ['']*len(v)

9      for i in range(0, len(w)):
10         ponteiros[i][0] = '|'
11     for j in range(0, len(v)):
12         ponteiros[0][j] = '_'
```

- Em (9-12), inicializamos com “|”s a primeira coluna da matriz de ponteiros e com “_”s a primeira linha desta mesma matriz
  - Esse procedimento é necessário para garantir que o alinhamento seja gerado até o fim nos casos em que há uma cauda inicial em uma das duas sequências
    - Exemplo: ACTG e **AAA**ACTG

FUNÇÃO LCS

```
13 for i in range(1, len(w)):
14     for j in range(1, len(v)):
15         pontuacao[i][j] = maximo(v[j], w[i], pontuacao[i-1][j], pontuacao[i][j-1], pontuacao[i-1][j-1])
16.        ponteiros[i][j] = ponteiro(v[j], w[i], pontuacao[i-1][j], pontuacao[i][j-1], pontuacao[i-1][j-1])
```

- Os laços em (13) e (14) são a implementação do algoritmo para preenchimento da matriz
- Em (15), implementamos o critério $s_{i,j} = max\ (s_{i-1,j}, s_{i,j-1}$ e $s_{i-1,j-1}+1)$ (desde que *v[i] = w[j]*). A função “maximo” será implementada a seguir
- Na linha (16), o mesmo critério é aplicado através da implementação que será explicada a seguir da função “ponteiro” que serve para decidir segundo o critério de pontuação para qual das três células vizinhas ($s_{i-1,j}$, $s_{i,j-1}$ e $s_{i-1,j-1}$) a célula atual deverá apontar (↑, ← e ↖)

FUNÇÃO LCS

```
17          imprimeMatriz(v, w, pontuacao, ponteiros)
18          geraAlinhamento(v, w, pontuacao, ponteiros)
```

- Por fim, as linhas (17) e (18) são chamadas a outras funções que detalharemos a seguir que imprimem as matrizes de pontuação e ponteiros e realizam a impressão do alinhamento de pontuação máxima calculado

FUNÇÃO MAXIMO

```
1 def maximo(c1, c2, cima, lado, diagonal):
2       if (c1 == c2 and (diagonal+1) >= cima and (diagonal+1) >= lado):
3             diagonal = diagonal+1
4             return diagonal
5       elif (lado >= cima and lado >= diagonal):
6             return lado
7       else:
8             return cima
```

- Nas linhas (1) a (8) definimos a função “maximo" que conforme explicado implementa o critério $s_{i,j} = max\ (s_{i-1,j},\ s_{i,j-1}$ *e* $s_{i-1,j-1}+1)$ (desde que *v[i] = w[j]*)
- Ela recebe como argumentos (1) os caracteres “c1" e “c2" que estão sendo avaliados e os valores das pontuações nas células vizinhas (“lado”, “cima”, “diagonal”)
- Note que, caso os caracteres sejam iguais e a pontuação da célula em diagonal seja a maior, retornamos o valor somado de “+1" (4)
- Caso contrário, podemos decidir vir da célula do lado (6) ou de cima (8)

FUNÇÃO PONTEIRO

```
11 def ponteiro(c1, c2, cima, lado, diagonal):
12       if (c1 == c2 and (diagonal+1) >= cima and (diagonal+1) >= lado):
13                return '\\'
14       elif (lado >= cima and lado >= diagonal):
15                return '_'
16       else:
17                return '|'
```

- A função “ponteiro” implementada nas linhas (11) a (17) é praticamente idêntica à função “maximo” com o diferencial de retornar símbolos “|”, “_” e “\” para representar respectivamente ↑, ← e ↖

FUNÇÃO IMPRIMEMATRIZ

```
21 def imprimeMatriz(v, w, pontuacao, ponteiros):
22       print('\t', end='');
23       for j in range(0, len(v)):
24               print(v[j], end='\t')
25       print()
26       for i in range(0, len(w)):
27               print(w[i], end='\t')
28               for j in range(0, len(v)):
29                       print(pontuacao[i][j], ponteiros[i][j], end='\t', sep='')
30               print()
31       print()
```

FUNÇÃO IMPRIMEMATRIZ

- A função “imprimeMatriz” implementada nas linhas (21) a (31) recebe como argumentos (21) as sequências *v* e *w* e as matrizes de pontuação e ponteiros e as percorre imprimindo a matriz de programação dinâmica gerada para simples conferência e entendimento

FUNÇÃO GERAALINHAMENTO

```
def geraAlinhamento(v, w, pontuacao, ponteiros):
    ali_v = ''
    ali_w = ''
    i = len(w)-1
    j = len(v)-1
    while ((i!=0) or (j!=0)):
        if (ponteiros[i][j] == '\\'):
            ali_v = v[j] + ali_v
            ali_w = w[i] + ali_w
            i-=1
            j-=1
        elif (ponteiros[i][j] == '_'):
            ali_v = v[j] + ali_v
            ali_w = '_' + ali_w
            j-=1
        else:
            ali_v = '_' + ali_v
            ali_w = w[i] + ali_w
            i-=1
    print(pontuacao[len(w)-1][len(v)-1])
    print(ali_v)
    print(ali_w)
```

FUNÇÃO GERAALINHAMENTO

- A função “geraAlinhamento” reconstrói o alinhamento a partir das matrizes de programação dinâmica calculadas pela função “lcs”

- Recebe como argumentos (1) as sequências *v* e *w* e as matrizes de pontuação e de ponteiros e as percorre da célula $s_{n,m}$ em direção à célula $s_{0,0}$ emitindo nas variáveis arranjo “ali_v“ e “ali_w” (2-3) caracteres ou *gaps* conforme direção do ponteiro da célula da matriz de ponteiros (7), (12) ou (16)

  - Caso seja um ↖, (“\”), emitimos um caracter em *v* e outro em *w* indicando um *match*

  - Caso seja um ← (“_”), emitimos um caracter em *v* e um *gap* em *w* indicando uma deleção;

  - Caso seja um ↑, (“|”), emitimos um caracter em w e um *gap* em *v* indicando uma inserção

  - As variáveis “i” e / ou “j” são decrementadas seguindo o caminhamento indicado

FUNÇÃO GERAALINHAMENTO

- Note que essa implementação pontua apenas em caso de *matches* não penalizando *indels* e nem permitindo *mismatches*
  - Esse é a base do algoritmo Needleman-Wunsch [Needleman e Wunsch, 1970], proposto por Saul Needleman e Christian Wunsch na década de 1970
  - Trata-se de um algoritmo para **alinhamento global par-a-par de sequências**
  - Há quatro tipos de algoritmos para alinhamento:
    - um alinhamento pode ser para-a-par ou múltiplo
    - e pode ser ao mesmo tempo global ou local