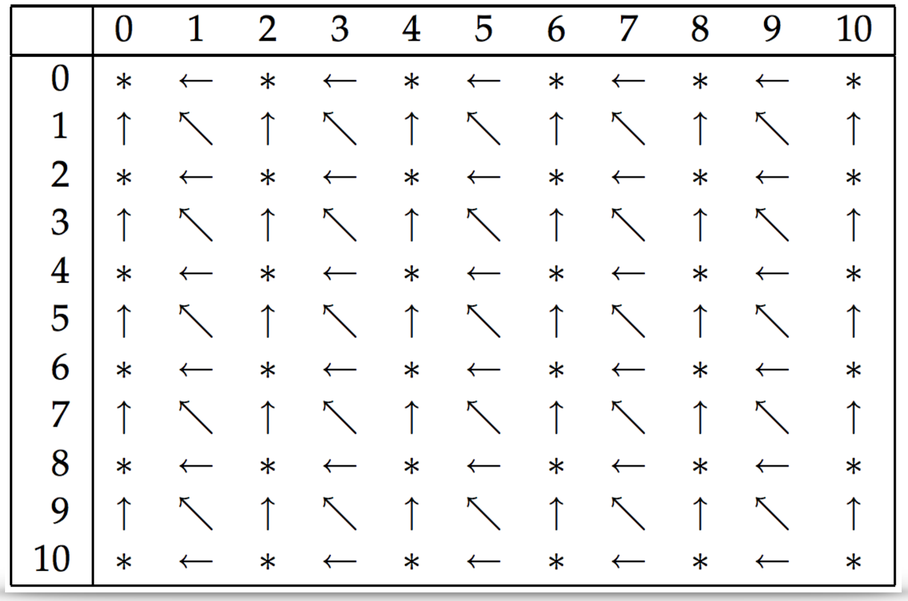
Profa. Dra. Raquel C. de Melo-Minardi
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais



	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
Algoritmo de Needleman-Wunsch

ALGORITMO DE NEEDLEMAN-WUNSCH

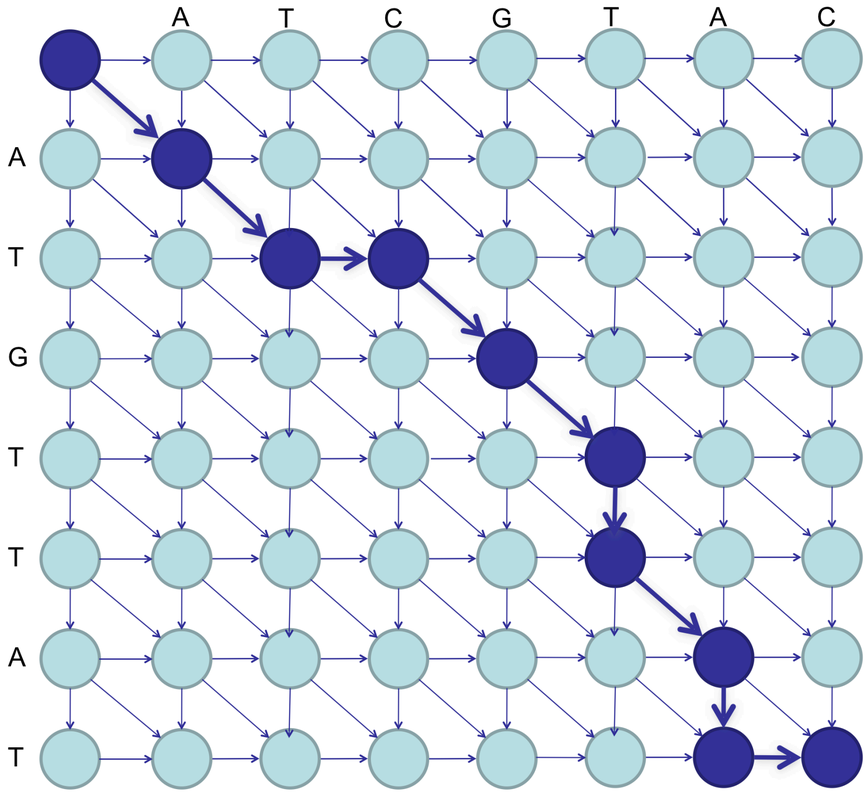
- Esse algoritmo foi descoberto e redescoberto inúmeras vezes em áreas que vão desde reconhecimento de fala até a biologia molecular
- Apesar de haverem detalhes que diferem nas diversas aplicações, todas são essencialmente **programação dinâmica**
- O alinhamento de duas sequências *v* (com *n* caracteres) e *w* (com *m* caracteres) pode ser visualizado assim

v	A	T	C	G	T	-	A	-	C
w	A	T	-	G	T	T	A	T	-

- Esse alinhamento poderá ter, no máximo (*n*+*m*) posições
- As colunas que contêm o mesmo caracter são chamadas de ***matches*** (casamentos)
- As colunas que contêm espaços são chamadas ***indels*** (inserções e deleções)

ALGORITMO DE NEEDLEMAN-WUNSCH

- Esse alinhamento poderia ser representado como o seguinte grid ou matriz na qual a primeira sequência está representada nas colunas e a segunda nas linhas



v	A	T	C	G	T	-	A	-	C
w	A	T	-	G	T	T	A	T	-

ALGORITMO DE NEEDLEMAN-WUNSCH

- Essa matriz é central em todo algoritmo de alinhamento e o caminho destacado representa o melhor alinhamento entre $s_{0,0}$ e $s_{n,m}$

- É interessante notar que as arestas no caminho máximo representam impressões

  - **Horizontais**: significam deleções na primeira sequência *v*, ou seja, um caracter na primeira sequência é impresso e um *gap* é impresso na segunda sequência *w*

  - **Verticais**: significam inserções na primeira sequência *v*, ou seja, um *gap* é impresso na primeira sequência e um caracter é impresso na segunda sequência *w*

  - **Diagonais**: significam um *match* ou *mismatch*, ou seja, um caracter é impresso em cada uma das sequências *v* e *w* podendo ser iguais ou diferentes

ESQUEMAS DE PONTUAÇÃO

- Falta um esquema de pontuação que seja capaz de julgar o mérito dos diversos possíveis alinhamentos
  - Algo similar ao que o número de pontos turísticos em cada quarteirão fazia no Problema do Turista de Manhattan
    - Custo de admitir *matches* e *mismatches* bem como dos *indels*

- Por simplicidade, podemos pontuar “+1" para um *match* e “0”, caso contrário

- Nessa primeira versão do algoritmo, não contemplamos um *mismatch*, visto que o esquema de pontuação permite caminhamento em diagonal apenas no caso de igualdade entre os caracteres

- Não há perda de generalidade

Problema da Máxima Subsequência Comum

Encontre a máxima subsequência comum entre duas sequências

Entradas: Duas sequências, *v* e *w*
Saída: A máxima subsequência comum entre *v* e *w*

ESQUEMAS DE PONTUAÇÃO

- Para resolver o LCS, criamos uma matriz de programação dinâmica colocando a primeira sequência como as colunas e a segunda como as linhas conforme a seguir e seguindo o seguinte critério

  - $s_{i,j} = max\ (s_{i-1,j},\ s_{i,j-1}\ e\ s_{i-1,j-1}+1$ (desde que $v[i] = w[j]$)

	v	A	T	C	G	T	A	C
w	0	0	0	0	0	0	0	0
A	0							
T	0							
G	0							
T	0							
T	0							
A	0							
T	0							

MATRIZ DE PROGRAMAÇAO DINAMICA

- Preenchemos a primeira linha ($s_{0,j}$) e coluna ($s_{i,0}$) com 0s para inicializar a pontuação do alinhamento

- Continuamos a partir da célula $s_{1,1}$ que corresponde ao problema de se alinhar as sequências “A” com “A”: *match* pontuando “+1” e deixando uma marca diagonal

	v	**A**	T	C	G	T	A	C
w	0	0	0	0	0	0	0	0
A	0	**1**↖						
T	0							
G	0							
T	0							
T	0							
A	0							
T	0							

MATRIZ DE PROGRAMAÇAO DINAMICA

- Calculamos toda a linha $s_{1,j}$
- Note que a célula $s_{1,2}$ é um *mismatch* de “A” com “T" e pontua “0”
- Essa pontuação tem de ser somada com a de $s_{1,1}$ pois a posição corrente indica o alinhamento de “A" com “AT” e valeria “1" com a marcação de horizontal pois veio de $s_{1,1}$

	v	**A**	**T**	C	G	T	A	C
w	0	0	0	0	0	0	0	0
A	0	1↖	**1←**					
T	0							
G	0							
T	0							
T	0							
A	0							
T	0							

MATRIZ DE PROGRAMAÇAO DINAMICA

- Preenchendo a linha

	v	A	T	**C**	**G**	**T**	**A**	**C**
w	0	0	0	0	0	0	0	0
A	0	1↖	1←	**1←**	**1←**	**1←**	**1↖**	**1←**
T	0							
G	0							
T	0							
T	0							
A	0							
T	0							

MATRIZ DE PROGRAMAÇAO DINAMICA

- Por fim, podemos continuar o preenchimento da matriz linha a linha

	v	**A**	**T**	**C**	**G**	**T**	**A**	**C**
w	0	0	0	0	0	0	0	0
A	0	1↖	1←	1←	1←	1←	1↖	1←
T	0	**1↑**	**2↖**	**2←**	**2←**	**2↖**	**2←**	**2←**
G	0	**1↑**	**2↑**	**2←**	**3↖**	**3←**	**3←**	**3←**
T	0	**1↑**	**2↖**	**2←**	**3↑**	**4↖**	**4←**	**4←**
T	0	**1↑**	**2↖**	**2←**	**3↑**	**4↖**	**4←**	**4←**
A	0	**1↖**	**2↑**	**2←**	**3↑**	**4↑**	**5↖**	**5←**
T	0	**1↑**	**2↖**	**2←**	**3↑**	**4↖**	**5↑**	**5←**

MATRIZ DE PROGRAMAÇAO DINAMICA

- Dessa matriz, podemos perceber que o alinhamento de pontuação máxima terá valor 5
- Além disso, seguindo os ponteiros partindo da célula $s_{n,m}$ até chegar a célula $s_{0,0}$, podemos gerar o alinhamento à direita

	v	**A**	**T**	**C**	**G**	**T**	**A**	**C**
w	0	0	0	0	0	0	0	0
A	0	1↖	1←	1←	1←	1←	1↖	1←
T	0	**1↑**	**2↖**	**2←**	**2←**	**2↖**	**2←**	**2←**
G	0	**1↑**	**2↑**	**2←**	**3↖**	**3←**	**3←**	**3←**
T	0	**1↑**	**2↖**	**2←**	**3↑**	**4↖**	**4←**	**4←**
T	0	**1↑**	**2↖**	**2←**	**3↑**	**4↖**	**4←**	**4←**
A	0	**1↖**	**2↑**	**2←**	**3↑**	**4↑**	**5↖**	**5←**
T	0	**1↑**	**2↖**	**2←**	**3↑**	**4↖**	**5↑**	**5←**

v	A	T	C	G	T	-	A	-	C
w	A	T	-	G	T	T	A	T	-

MATRIZ DE PROGRAMAÇAO DINAMICA

- Note que esse alinhamento é construído de trás para frente, ou seja, da direita para a esquerda
  - Partindo da célula $s_{7,7}$, na qual temos um “5←”, imprimiremos no alinhamento o caracter “C" da sequência *v* e um *gap* na sequência *w*

	v	**A**	**T**	**C**	**G**	**T**	**A**	**C**
w	0	0	0	0	0	0	0	0
A	0	1↖	1←	1←	1←	1←	1↖	1←
T	0	**1↑**	**2↖**	**2←**	**2←**	**2↖**	**2←**	**2←**
G	0	**1↑**	**2↑**	**2←**	**3↖**	**3←**	**3←**	**3←**
T	0	**1↑**	**2↖**	**2←**	**3↑**	**4↖**	**4←**	**4←**
T	0	**1↑**	**2↖**	**2←**	**3↑**	**4↖**	**4←**	**4←**
A	0	**1↖**	**2↑**	**2←**	**3↑**	**4↑**	**5↖**	**5←**
T	0	**1↑**	**2↖**	**2←**	**3↑**	**4↖**	**5↑**	**5←**

v	A	T	C	G	T	-	A	-	C
w	A	T	-	G	T	T	A	T	-

Desafio

Implemente em Python o algoritmo da Máxima Subsequência Comum.

Dica: use uma matriz para armazenar as pontuações e outra para armazenar os ponteiros.