Programação Dinâmica

Ricardo Manhães Savii
ricardosavii@gmail.com

Problemas

Cálculo de Fibonacci

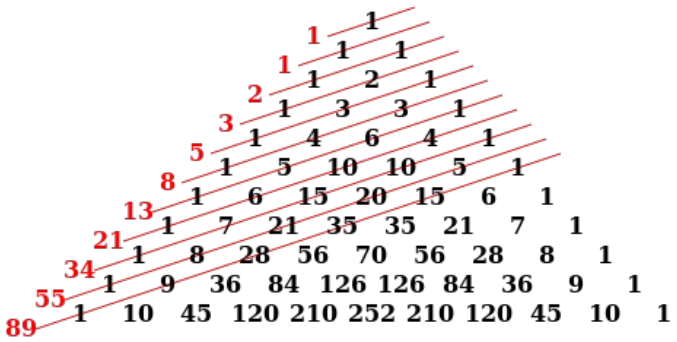
Comparação de sequências de caracteres (aplicação: DNA)

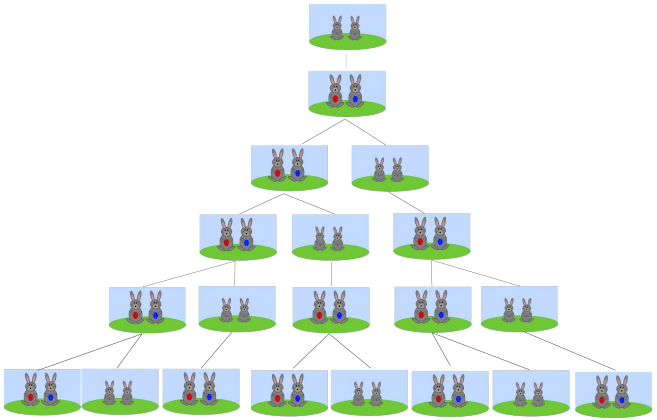
Fibonacci

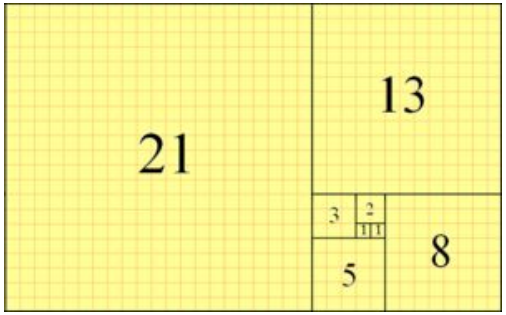
List of Fibonacci numbers [edit]

The first 21 Fibonacci numbers F_n for $n = 0, 1, 2, \ldots, 20$ are:[18]

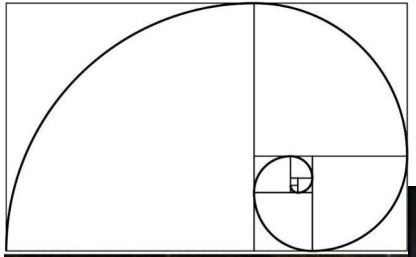
F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

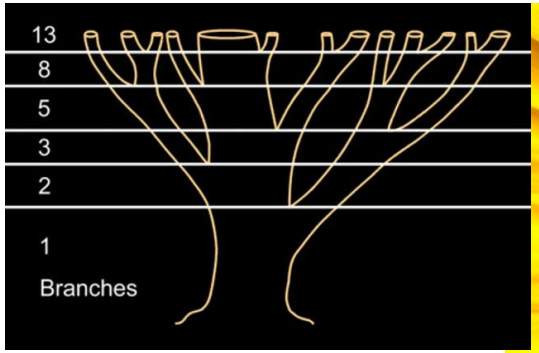


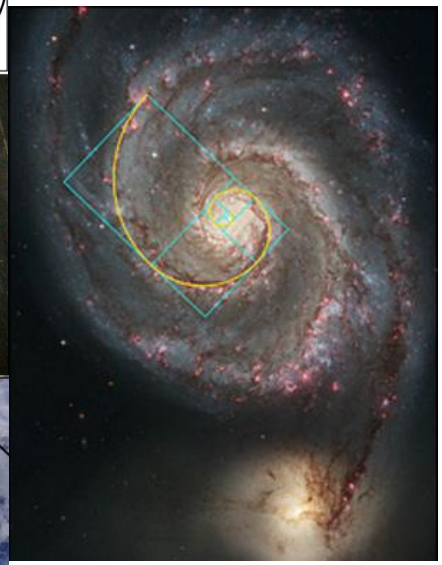


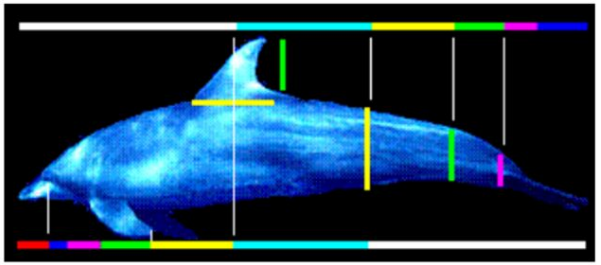


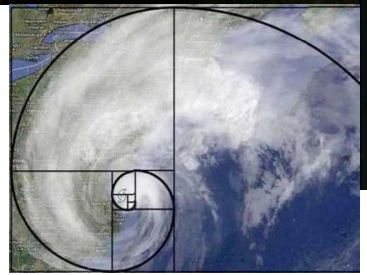
Fibonacci

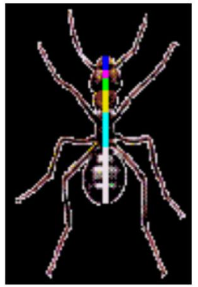












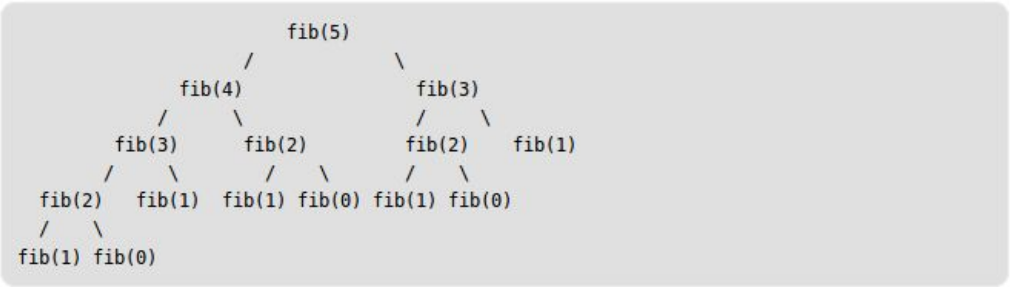
Fibonacci (computação)

$$F(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ F(n-1) + F(n-2) & \text{caso contrário} \end{cases}$$

```c
//Fibonacci Series using Recursion
#include<stdio.h>
int fib(int n)
{
   if (n <= 1)
      return n;
   return fib(n-1) + fib(n-2);
}

int main ()
{
  int n = 9;
  printf("%d", fib(n));
  getchar();
  return 0;
}
```

Time Complexity: T(n) = T(n-1) + T(n-2) which is exponential.



Vamos alinhar DNAs :)

Atividade: construa a matriz de Programação dinâmica alinhando o para de sequências abaixo:

seq1: GAATTCAGTTA

seq2: GGATCGA

Resultado esperado (submeter):

A) Programa de alinhamento local (Programação dinâmica)

B) A matriz de alinhamento local (Programação dinâmica)

C) O alinhamento e o escore total das duas sequencias como a Fig. 1 do link abaixo.

http://vlab.amrita.edu/?sub=3&brch=274&sim=1433&cnt=1

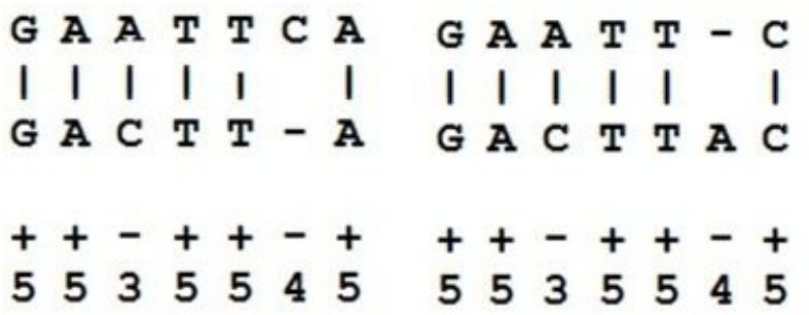


Figure 1: Alinhamento de duas sequências quaisquer.

Introdução

O algoritmo de Smith-Waterman realiza o alinhamento local e utiliza programação dinâmica para tal função.
Para isso ele precisa de uma matriz de score, à qual nos foi dada pelo enunciado do exercício como:

- match = +5
- mismatch = -3
- gap = -4

Logo nossa matriz terá o seguinte fomato para sequências de DNA:

```
matriz_score <- matrix(data = c(5, -3, -3, -3, -4,
                                -3, 5, -3, -3, -4,
                                -3, -3, 5, -3, -4,
                                -3, -3, -3, 5, -4,
                                -4, -4, -4, -4, -4), nrow = 5, ncol = 5, byrow = TRUE )
aa <- c('A', 'G', 'T', 'C', '-')
rownames(matriz_score) <- aa
colnames(matriz_score) <- aa
matriz_score
```

```
##    A  G  T  C  -
## A  5 -3 -3 -3 -4
## G -3  5 -3 -3 -4
## T -3 -3  5 -3 -4
## C -3 -3 -3  5 -4
## - -4 -4 -4 -4 -4
```

Salvaremos agora as duas sequências de teste em duas variáveis no programa. Salvei previamente as sequências do enunciado em um arquivo **seq_teste.fasta** e farei a leitura do arquivo utilizando a função **read.fasta** do pacote **seqinr**.

```
squencias <- seqinr::read.fasta("seq_teste.fasta", seqtype = "AA")
seq1 <- squencias[[1]]
seq2 <- squencias[[2]]
seq3 <- squencias[[3]]
seq4 <- squencias[[4]]
seq1
```

```
##  [1] "G" "A" "A" "T" "T" "C" "A" "G" "T" "T" "A"
## attr(,"name")
## [1] ""
## attr(,"Annot")
## [1] "> seq1:"
## attr(,"class")
## [1] "SeqFastaAA"
```

	-	C	G	T	G	A	A	T	T	C	A	T
-	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	5	1	5	1	0	0	0	0	0	0
A	0	0	1	2	1	10	6	2	0	0	5	1
C	0	5	1	0	0	6	7	3	0	5	1	2
T	0	1	2	6	2	2	3	12	8	4	2	6
T	0	0	0	7	3	0	0	8	17	13	9	7
A	0	0	0	3	4	8	5	4	13	14	18	14
C	0	5	1	0	0	4	5	2	9	18	14	15

Figure 2: Matriz preenchida e os back pointers.

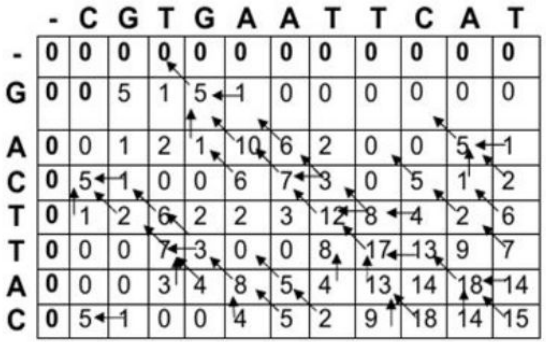


Figure 2: Matriz preenchida e os back pointers.

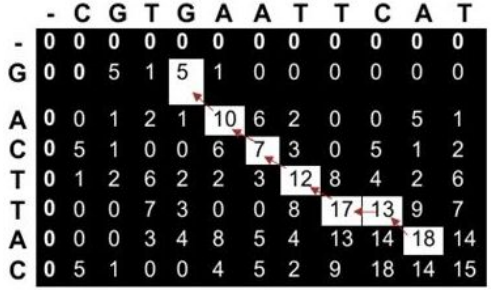


Figure 3: Trace back of first possible alignment

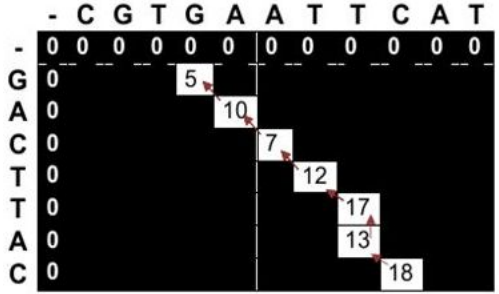


Figure 4: Trace back of second possible alignment

```
G A A T T C A    G A A T T - C
| | | | |   |    | | | | |   |
G A C T T - A    G A C T T A C

+ + - + + - +    + + - + + - +
5 5 3 5 5 4 5    5 5 3 5 5 4 5
```

Explanation [edit]

A matrix H is built as follows:

$$H(i,0) = 0,\ 0 \le i \le m$$

$$H(0,j) = 0,\ 0 \le j \le n$$

$$H(i,j) = \max \left\{ \begin{array}{cc} 0 & \\ H(i-1,j-1) + s(a_i,b_j) & \text{Match/Mismatch} \\ \max_{k \ge 1}\{H(i-k,j) + W_k\} & \text{Deletion} \\ \max_{l \ge 1}\{H(i,j-l) + W_l\} & \text{Insertion} \end{array} \right\},\ 1 \le i \le m, 1 \le j \le n$$

Where:

- a, b = Strings over the Alphabet Σ
- $m = \text{length}(a)$
- $n = \text{length}(b)$
- $s(a,b)$ is a similarity function on the alphabet
- $H(i,j)$ – is the maximum Similarity-Score between a suffix of a[1...i] and a suffix of b[1...j]
- W_i is the gap-scoring scheme

Example [edit]

- Sequence 1 = ACACACTA
- Sequence 2 = AGCACACA
- $s(a,b) = +2 \text{ if } a = b \text{ (match)},\ -1 \text{ if } a \ne b \text{ (mismatch)}$
- $W_i = -1$

$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 1 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 1 & 3 & 2 & 3 & 2 & 3 & 2 & 2 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 4 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 6 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 8 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 10 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix} \quad T = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ G & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow & \uparrow \\ C & 0 & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow \end{pmatrix}$$

Desenvolvimento do código e testes

Pensando na função, seus parâmetros precisam ser as duas sequências à serem alinhadas e a matriz de escore. Como resultado devemos obter o escore final da comparação local entre as duas sequências, a matriz de comparação entre as duas sequências e a matriz de direções das decisões tomadas para adquirir o melhor alinhamento. Tendo a matriz conseguiremos gerar o alinhamento e o escore total das duas sequências conforme a figura do enunciado.

```
compara <- function(seq1, seq2, matriz_score) {
  seq1 <- c('-',seq1)
  seq2 <- c('-',seq2)

  matriz <- matrix(nrow = length(seq1), ncol = length(seq2))
  direcoes <- matrix(nrow = length(seq1), ncol = length(seq2))

  rownames(matriz) <- seq1
  colnames(matriz) <- seq2
  rownames(direcoes) <- seq1
  colnames(direcoes) <- seq2

  matriz[1,] <- matriz[,1] <- 0
  for(i in 2:length(seq1)) {
    for(j in 2:length(seq2)) {
      matriz[i,j] <- max(matriz[i-1,j-1] + matriz_score[seq1[i],seq2[j]],
                         matriz[i  ,j-1] + matriz_score['-',seq2[j]],
                         matriz[i-1,j  ] + matriz_score[seq1[i],'-'],
                         0)
      if(matriz[i-1,j-1] + matriz_score[seq1[i],seq2[j]] >=
         matriz[i  ,j-1] + matriz_score['-',seq2[j]] &&
         matriz[i-1,j-1] + matriz_score[seq1[i],seq2[j]] >=
         matriz[i-1,j  ] + matriz_score[seq1[i],'-']) {
        direcoes[i,j] <- '/'
      } else {
        if(matriz[i  ,j-1] + matriz_score['-',seq2[j]] >=
           matriz[i-1,j-1] + matriz_score[seq1[i],seq2[j]] &&
           matriz[i  ,j-1] + matriz_score['-',seq2[j]] >=
           matriz[i-1,j  ] + matriz_score[seq1[i],'-']) {
          direcoes[i,j] <- '-'
        } else {
          if (matriz[i-1,j  ] + matriz_score[seq1[i],'-'] >
              matriz[i-1,j-1] + matriz_score[seq1[i],seq2[j]] &&
              matriz[i-1,j  ] + matriz_score[seq1[i],'-'] >
              matriz[i  ,j-1] + matriz_score['-',seq2[j]]) {
            direcoes[i,j] <- '|'
          }
        }
      }
    }
  }

  list(matriz,direcoes)
}
```

Primeiro vamos comparar as sequências que foram dadas a matriz resposta, assim verificaremos se nossa função está funcionando corretamente:

```
resultado_teste <- compara(seq3, seq4, matriz_score)
resultado_teste[[1]]
```

```
##    - C G T G  A A  T  T  C  A  T
## - 0 0 0 0 0  0 0  0  0  0  0  0
## G 0 0 5 1 5  1 0  0  0  0  0  0
## A 0 0 1 2 1 10 6  2  0  0  5  1
## C 0 5 1 0 0  6 7  3  0  5  1  2
## T 0 1 2 6 2  2 3 12  8  4  2  6
## T 0 0 0 7 3  0 0  8 17 13  9  7
## A 0 0 0 3 4  8 5  4 13 14 18 14
## C 0 5 1 0 0  4 5  2  9 18 14 15
> resultado_teste[[2]]
   -  C  G  T  G  A  A  T  T  C  A  T
- NA NA NA NA NA NA NA NA NA NA NA NA
G NA "/" "/" "-" "/" "-" "/" "/" "/" "/" "/" "/"
A NA "/" "|" "/" "|" "/" "/" "-" "-" "/" "/" "-"
C NA "/" "-" "/" "/" "|" "/" "/" "/" "/" "-" "/"
T NA "|" "/" "/" "-" "|" "/" "/" "/" "-" "/" "/"
T NA "/" "/" "/" "/" "/" "/" "/" "/" "-" "-" "/"
A NA "/" "/" "|" "/" "/" "/" "|" "|" "/" "/" "-"
C NA "/" "-" "|" "/" "|" "/" "/" "|" "/" "-" "/"
```

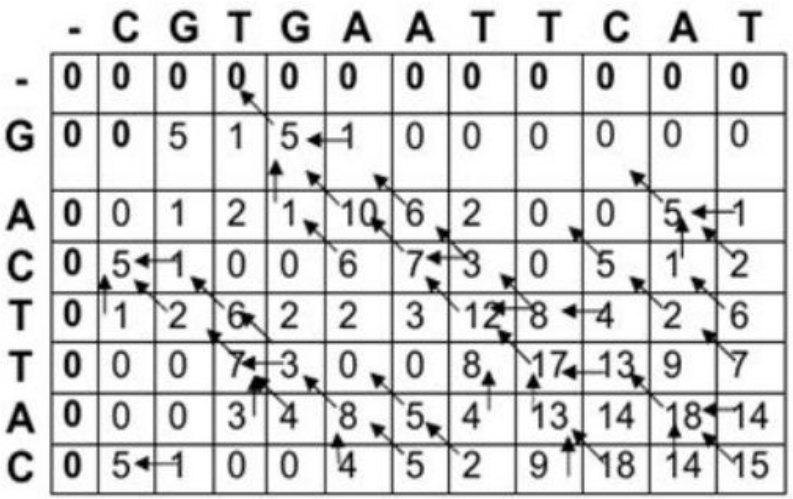


Figure 2: Matriz preenchida e os back pointers.

Agora precisamos de uma segunda função que faça a busca retro-ativa na matriz para encontrar os maiores scores e nos mostre os melhores alinhamentos entre as duas sequências. Para representar os alinhamentos vou pensar na imagem como uma matriz e alimentar as duas primeiras linhas com os alinhamentos gerados e a terceira linha com os valores de escore. Logo podemos gerar uma nova função:

```
alinhar <- function(matriz) {
  res <- list()
  maximos <- which(matriz[[1]] == max(matriz[[1]]), arr.ind = TRUE)

  for (i in 1:dim(maximos)[1]) {
    linha = maximos[i,1]
    coluna = maximos[i,2]
    seq1 <- c()
    seq2 <- c()
    score <- c()

    while(TRUE) {
      if (matriz[[1]][linha, coluna] == 0) break
      if (matriz[[2]][linha, coluna] == '/') {
        seq1 = c( rownames(matriz[[1]])[linha], seq1 )
        seq2 = c( colnames(matriz[[1]])[coluna], seq2 )
        score = c( matriz[[1]][linha,coluna] - matriz[[1]][linha-1,coluna-1], score )

        linha = linha-1
        coluna = coluna-1

      } else if (matriz[[2]][linha, coluna] == '-') {
        seq1 = c( '-', seq1 )
        seq2 = c( colnames(matriz[[1]])[coluna], seq2 )
        score = c( matriz[[1]][linha,coluna] - matriz[[1]][linha,coluna-1], score )

        coluna = coluna-1
      } else if (matriz[[2]][linha, coluna] == '|') {
        seq1 = c( rownames(matriz[[1]])[linha], seq1 )
        seq2 = c( '-', seq2 )
        score = c( matriz[[1]][linha,coluna] - matriz[[1]][linha-1,coluna], score )

        linha = linha-1
      }
    }
    res[[i]] <- rbind(seq1, seq2, score)
  }
  res
}
```

Com a função definida acima conseguimos agora verificar se conseguiremos com as matrizes escore e direções em **resultado_teste** o mesmo resultado visto na **Figura 1**.

```
teste1 <- alinhar(resultado_teste)
teste1
```

```
## [[1]]
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## seq1  "G"  "A"  "C"  "T"  "T"  "A"  "C"
## seq2  "G"  "A"  "A"  "T"  "T"  "-"  "C"
## score "5"  "5"  "-3" "5"  "5"  "-4" "5"
##
## [[2]]
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## seq1  "G"  "A"  "C"  "T"  "T"  "-"  "A"
## seq2  "G"  "A"  "A"  "T"  "T"  "C"  "A"
## score "5"  "5"  "-3" "5"  "5"  "-4" "5"
```

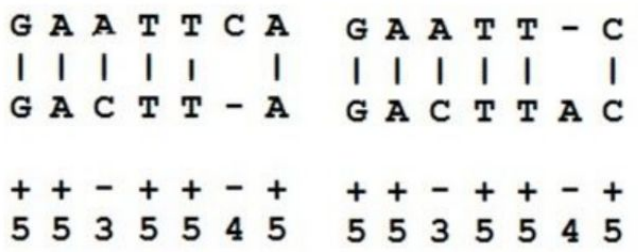


Figure 1: Alinhamento de duas sequências quaisquer.

DNA (importância da corretude do algoritmo)

Ter um bom projeto de indução (e não programar na tentativa e erro) é importante porque em problemas de comparação de DNA se comparam sequências gigantescas (exemplo **<u>pequeninníssimo</u>** abaixo)

>seq 1 frame 1 tamanho 1158
atgaataaacttaaagtgaattctgtcgttgaaagaaaaatcaaatcaggtgctcagttactggaaaaaaaagattttgataccagtttagttaaccagttggttc
aacttttttcacagtcaaatcaattcttagggatggcctatctttcaccacaaaataaagggattggttggttactatcaagacaaatttttgattttaaccatgattact
ttgtatcgctattcgaaaaatctagagagaagcgtcaaaaatttgaaaaatctagccaaacaacagcctaccgcttgtttaatcaggatggagataactttggtg
gcctaactatagatttttatagtgactatgctctttttcgtggtacaatgaatttgtttatactaatcgacaaatgattgttgcagcctttaagcaggtctatcctaatatta
aaggggcatatgaaaaaattcgtttcaaaggtttagactttgaaagtgctcatttgtacggtcaagaggctcctgaatcatttttgattttagaaaataatatcaaat
atagtgtctttttgaatgatgggttaatgacaggtattttccttgaccaacatgatgtcagaaaagccttagcaactaatctatcagaaggtaaaaaagttctaaata
tgttttcatacactgcggctttttctgtagctgcagctgttggaggagcattagagacaacttctgttgatttagcaaaacgctcgcgtgaactttcaaaagcacactt
tgatgctaatcagattgtcacagataaccatcgatttatcgttatggatgtttttgaatattataagtatgccaaaagaaaacatctatcatatgatgtgattgttattga
tccaccaagttttgctcgaaataaaaaacaaacttttcggttactaaagattactataaattaattgaacaagctttagatattttagcccctggtggaactatcatt
gcatcaactaacgcagctaatctaaccgtatcacaatttaagaaacaattggaaaaggggtttggtaaagcttcacataattacattagcttacagcagttgcct
gaagatttcactgtaaatgataaggaccaacaaagtaattatttaaaagtatttacaataaaggtaaaataa

Exercícios:

1) implementar cálculo de Fibonacci com programação dinâmica:

```c
//Fibonacci Series using Recursion
#include<stdio.h>
int fib(int n)
{
   if (n <= 1)
      return n;
   return fib(n-1) + fib(n-2);
}

int main ()
{
  int n = 9;
  printf("%d", fib(n));
  getchar();
  return 0;
}
```

2) comparar duas sequências de caracteres:
   a) GAATTCAGTTA
   b) GGATCGA

Respostas Fibonacci

C

```c
//Fibonacci Series using Dynamic Programming
#include<stdio.h>

int fib(int n)
{
  /* Declare an array to store Fibonacci numbers. */
  int f[n+1];
  int i;

  /* 0th and 1st number of the series are 0 and 1*/
  f[0] = 0;
  f[1] = 1;

  for (i = 2; i <= n; i++)
  {
      /* Add the previous 2 numbers in the series
         and store it */
      f[i] = f[i-1] + f[i-2];
  }

  return f[n];
}

int main ()
{
  int n = 9;
  printf("%d", fib(n));
  getchar();
  return 0;
}
```

Time Complexity: O(n)

Extra Space: O(n)

Java

```java
// Fibonacci Series using Dynamic Programming
class fibonacci
{
   static int fib(int n)
    {
        /* Declare an array to store Fibonacci numbers. */
    int f[] = new int[n+1];
    int i;

    /* 0th and 1st number of the series are 0 and 1*/
    f[0] = 0;
    f[1] = 1;

    for (i = 2; i <= n; i++)
    {
       /* Add the previous 2 numbers in the series
          and store it */
        f[i] = f[i-1] + f[i-2];
    }

    return f[n];
    }

    public static void main (String args[])
    {
        int n = 9;
        System.out.println(fib(n));
    }
}
/* This code is contributed by Rajat Mishra */
```

Resposta Fibonacci (melhorada)

Method 3 (Space Optimized Method 2)

We can optimize the space used in method 2 by storing the previous two numbers only because that is all we need to get the next Fibonacci number in series.

```
//Fibonacci Series using Space Optimized Method
#include<stdio.h>
int fib(int n)
{
  int a = 0, b = 1, c, i;
  if( n == 0)
    return a;
  for (i = 2; i <= n; i++)
  {
     c = a + b;
     a = b;
     b = c;
  }
  return b;
}

int main ()
{
  int n = 9;
  printf("%d", fib(n));
  getchar();
  return 0;
}
```

Time Complexity: O(n)

Extra Space: O(1)

E dá para melhorar ainda mais… veja a fonte:

fonte: http://www.geeksforgeeks.org/program-for-nth-fibonacci-number/

Respostas parciais DNA

```
resultado <- compara(seq2, seq1, matriz_score)
resultado[[1]]
```

```
##   - G  A A  T T  C  A  G  T T  A
## - 0 0  0 0  0 0  0  0  0  0 0  0
## G 0 5  1 0  0 0  0  0  5  1 0  0
## G 0 5  2 0  0 0  0  0  5  2 0  0
## A 0 1 10 7  3 0  0  5  1  2 0  5
## T 0 0  6 7 12 8  4  1  2  6 7  3
## C 0 0  2 3  8 9 13  9  5  2 3  4
## G 0 5  1 0  4 5  9 10 14 10 6  2
## A 0 1 10 6  2 1  5 14 10 11 7 11
```

```
> resultado[[2]]
  -  G   A   A   T   T   C   A   G   T   T   A
- NA NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
G NA "/" "-" "/" "/" "/" "/" "/" "/" "-" "/" "/"
G NA "/" "/" "/" "/" "/" "/" "/" "/" "/" "/" "/"
A NA "|" "/" "/" "-" "-" "/" "/" "-" "/" "/" "/"
T NA "/" "|" "/" "/" "/" "-" "|" "/" "/" "/" "-"
C NA "/" "|" "/" "|" "/" "/" "-" "-" "|" "/" "/"
G NA "/" "-" "/" "|" "/" "|" "/" "/" "-" "-" "-"
A NA "|" "/" "/" "-" "/" "|" "/" "-" "/" "/" "/"
```

```
aux <- alinhar(resultado)
aux
```

```
## [[1]]
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## seq1  "G"  "G"  "A"  "-"  "T"  "C"  "G"  "A"
## seq2  "G"  "A"  "A"  "T"  "T"  "C"  "-"  "A"
## score "5"  "-3" "5"  "-4" "5"  "5"  "-4" "5"
##
## [[2]]
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## seq1  "G"  "G"  "A"  "-"  "T"  "C"  "-"  "G"
## seq2  "G"  "A"  "A"  "T"  "T"  "C"  "A"  "G"
## score "5"  "-3" "5"  "-4" "5"  "5"  "-4" "5"
```