Docente: **Matteo Re**

UNIVERSITÀ DEGLI STUDI DI MILANO

C.d.l. Informatica

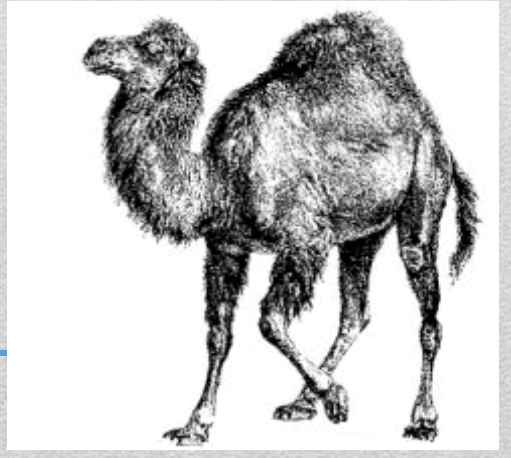
Bioinformatica

A.A. 2013-2014 semestre I

p3 Programmazione dinamica I

Obiettivi

- **Programmazione dinamica in PERL**
  - Implementazione di un algoritmo che utilizzi tecniche di programmazione dinamica
  - Utilizzo di matrici
  - Utilizzo di array associativi
  - Controllo di flusso : **IF**
  - Controllo di flusso : **WHILE**
- **Biologia computazionale**
  - Analisi e manipolazione di biosequenze (allineamento)
  - Implementazione algoritmo **Smith-Waterman**



Linee guida

- **Il livello di complessità di questa esercitazione è medio**
  - Cercate di risolvere il problema dopo averlo suddiviso in sottoproblemi
  - Indipendentemente dal fatto che lo script Perl funzioni o meno l'esercizio **NON verrà valutato** se, insieme allo script, non verrà inviato anche lo pseudocodice.

- **Modalità di svolgimento dell'esercitazione:**
  - Scaricare dal sito web del corso il file SW_exercise.pl
  - Questo script è **incompleto**
  - La posizione delle parti da aggiungere è evidenziata da questo commento:

########## description # FILL IN ####################

description: descrizione dell'operazione da svolgere

- Alcune operazioni sono indicate con **OPT** … esse riguardano, principalmente, operazioni che servono per migliorare l'output dello script e che lo rendono più leggibile. Sono parti opzionali. Se le realizzate avrete dei punti in più per la soluzione dell'esercizio.

Sequenze

```
>gi|40457238|HIV-1 isolate 97KE128 from Kenya gag gene, partial cds
CTTTTGAATGCATGGGTAAAAGTAATAGAAGAAAGAGGTTTCAGTCCAGAAGTAATACCCATGTTCTCAG
CATTATCAGAAGGAGCCACCCCACAAGATTTAAATACGATGCTGAACATAGTGGGGGGACACCAGGCAGC
TATGCAAATGCTAAAGGATACCATCAATGAGGAAGCTGCAGAATGGGACAGGTTACATCCAGTACATGCA
GGGCCTATTCCGCCAGGCCAGATGAGAGAACCAAGGGGAAGTGACATAGCAGGAACTACTAGTACCCCTC
AAGAACAAGTAGGATGGATGACAAACAATCCACCTATCCCAGTGGGAGACATCTATAAAAGATGGATCAT
CCTGGGCTTAAATAAAATAGTAAGAATGTATAGCCCTGTTAGCATTTTGGACATAAAACAAGGGCCAAAA
GAACCCTTTAGAGACTATGTAGATAGGTTCTTTAAAACTCTCAGAGCCGAACAAGCTT
```

In questa esercitazione **NON** leggeremo le sequenze da file FASTA. Vogliamo utilizzare sequenze corte in modo da avere un output facilmente interpretabile. Passeremo quindi le sequenze allo script **direttamente dalla linea di comando**.

Le sequenze che utilizzeremo sono le seguenti (in questo ordine):

AAUGCCAUUGACGG

CAGCCUCGCUUAG

Risultato (pg.1)

```
perl SW_exercise.pl AAUGCCAUUGACGG CAGCCUCGCUUAG

SEQUENCE 1 : AAUGCCAUUGACGG      LENGTH: 14
SEQUENCE 2 : CAGCCUCGCUUAG       LENGTH: 13

DYNAMIC PROGRAMMING MATRICES:

I) Scores:

    A   A   U   G   C   C   A   U   U   G   A   C   G   G

C  0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
A  1.0 1.0 0.0 0.0 0.0 0.7 2.0 0.7 0.0 0.0 1.0 0.0 0.7 0.0
G  0.0 0.7 0.7 1.0 0.0 0.0 0.7 1.7 0.3 1.0 0.0 0.7 1.0 1.7
C  0.0 0.0 0.3 0.3 2.0 1.0 0.0 0.3 1.3 0.0 0.7 1.0 0.3 0.7
C  0.0 0.0 0.0 0.0 1.3 3.0 1.7 0.3 0.0 1.0 0.0 1.7 0.7 0.0
U  0.0 0.0 1.0 0.0 0.0 1.7 2.7 2.7 1.3 0.0 0.7 0.3 1.3 0.3
C  0.0 0.0 0.0 0.7 1.0 1.0 1.3 2.3 2.3 1.0 0.0 1.7 0.3 1.0
G  0.0 0.0 0.0 1.0 0.3 0.7 0.7 1.0 2.0 3.3 2.0 0.7 2.7 1.3
C  0.0 0.0 0.0 0.0 2.0 1.3 0.3 0.3 0.7 2.0 3.0 3.0 1.7 2.3
U  0.0 0.0 1.0 0.0 0.7 1.7 1.0 1.3 1.3 0.7 1.7 2.7 2.7 1.3
U  0.0 0.0 1.0 0.7 0.0 0.3 1.3 2.0 2.3 1.0 0.3 1.3 2.3 2.3
A  1.0 1.0 0.0 0.7 0.3 0.0 1.3 1.0 1.7 2.0 2.0 0.7 1.0 2.0
G  0.0 0.7 0.7 1.0 0.3 0.0 0.0 1.0 0.7 2.7 1.7 1.7 1.7 2.0
```

Continua…

Risultato (pg.2)

… Continua

II) Pointers:

	A	A	U	G	C	C	A	U	U	G	A	C	G	G
C	*	*	*	*	d	d	*	*	*	*	*	d	*	*
A	d	d	*	*	*	d	d	*	*	*	d	*	d	*
G	*	d	d	d	*	*	u	d	l	d	*	d	d	d
C	*	*	d	d	d	d	*	u	d	l	d	d	d	d
C	*	*	*	d	d	d	l	l	u	d	*	d	d	d
U	*	*	d	*	u	u	d	d	l	l	d	u	d	d
C	*	*	*	d	d	d	d	d	d	d	*	d	l	d
G	*	*	*	d	d	d	d	d	d	d	l	l	d	d
C	*	*	*	*	d	d	d	d	d	u	d	d	l	d
U	*	*	d	*	u	d	d	d	d	u	d	d	d	d
U	*	*	d	d	*	u	d	d	d	d	u	d	d	d
A	d	d	*	d	d	*	d	d	d	d	d	l	d	d
G	*	d	d	d	d	d	u	d	d	d	d	d	d	d

Max score = 3.3 column: 10 (G) row: 8 (G)

Alignment:

```
GCCAUUG
|||  |.|
GCC-UCG
```

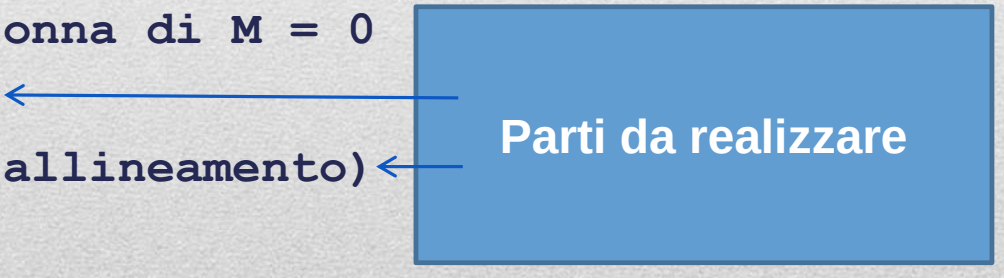
NB: questo è un esempio di output prodotto da uno script in cui sono state implementate anche le parti **opzionali** !

Output minimo:

```
GCCAUUG

GCC-UCG
```

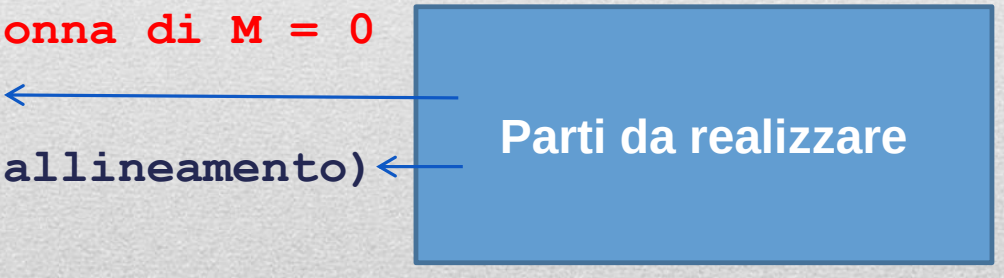
Pseudocodice
(alto livello)

```
1  Leggi sequenza1 (m caratteri)
2  Leggi sequenza2 (n caratteri)
3  Imposta schema di scoring (score match/mismatch e
   gap)
4  Crea matrice M ( n+1 righe , m+1 colonne)
5  Imposta M[0][0] = 0
6  Imposta celle prima riga di M = 0
7  Imposta celle prima colonna di M = 0
8  Compilazione matrice M
9  Traceback (costruzione allineamento)
10 Stampa allineamento
```



Pseudocodice
(alto livello)

```
1  Leggi sequenza1 (m caratteri)
2  Leggi sequenza2 (n caratteri)
3  Imposta schema di scoring (score match/mismatch e
   gap)
4  Crea matrice M ( n+1 righe , m+1 colonne)
5  Imposta M[0][0] = 0
6  Imposta celle prima riga di M = 0
7  Imposta celle prima colonna di M = 0
8  Compilazione matrice M
9  Traceback (costruzione allineamento)
10 Stampa allineamento
```



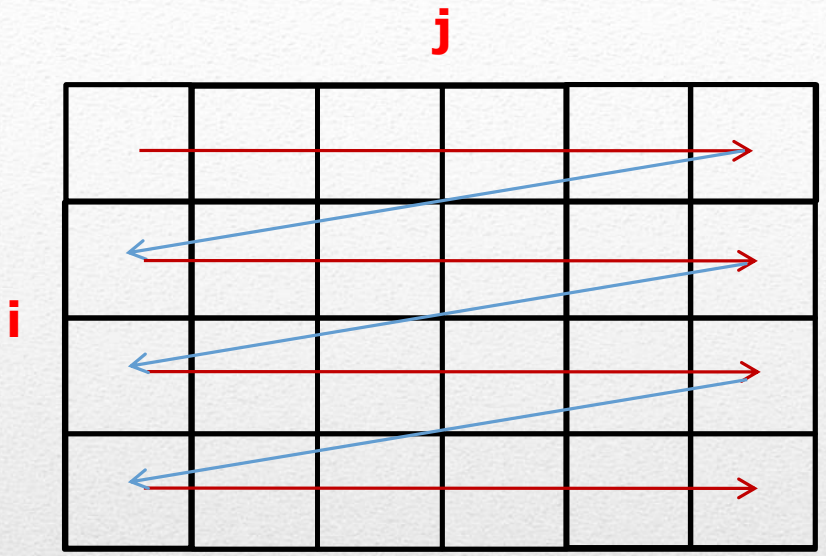
Step 1-7 : Perl

```
# usage statement
die "usage: $0 <sequence 1> <sequence 2>\n" unless @ARGV == 2;

# get sequences from command line
1  $seq1 = shift;
2  $seq2 = shift;

3  # scoring scheme
   my $MATCH    =  1;      # +1 for letters that match
   my $MISMATCH = -(1/3);  # - (1/3) for letters that mismatch
   my $GAP      = -(4/3);  # - (1 + 1/3) for any gap

   # initialization
4  my @matrix;
5  $matrix[0][0]{score}   = 0;
   $matrix[0][0]{pointer} = "*";
6  for(my $j = 1; $j <= length($seq1); $j++) {
       $matrix[0][$j]{score}   = 0;
       $matrix[0][$j]{pointer} = "*";
   }
7  for (my $i = 1; $i <= length($seq2); $i++) {
       $matrix[$i][0]{score}   = 0;
       $matrix[$i][0]{pointer} = "*";
   }
```

NB: nelle celle della matrice NON salviamo degli scalari … usiamo invece un **HASH**. In questo modo possiamo salvare sia i valori della matrice degli **score** che i **puntatori**

Step 8: Compilazione Matrice

E' realizzata attraverso **due cicli** , uno dentro all'altro. Il ciclo **esterno** attraversa le RIGHE della matrice mentre il ciclo **interno** attraversa le COLONNE.




```
for(my $i = 1; $i <= length($seq2); $i++) {
    for(my $j = 1; $j <= length($seq1); $j++) {



    }
}
```

Step 8: Compilazione Matrice

Mentre attraversiamo la matrice dobbiamo svolgere **3** operazioni:

```
for(my $i = 1; $i <= length($seq2); $i++) {
    for(my $j = 1; $j <= length($seq1); $j++) {
```

PSEUDOCODE

- Calcolare lo score da inserire nella cella corrente (score di una cella preesistente **+** score per match/mismatch o gap)
- Salvare un puntatore (UP (u), LEFT (l) o DIAGONAL (d)) che ci ricordi quale score abbiamo utilizzato come "base" per lo score attuale
- Verificare se il valore che abbiamo calcolato per la cella corrente è il massimo che abbiamo visto finora … se è così **salviamo la posizione corrente (indice riga, indice colonna)**

```
    }
}
```

Step 8: Compilazione Matrice

Come calcoliamo i valori da inserire nella matrice H (che sarebbe la variabile @matrix) ?

I valori della matrice H_{ij} vengono determinati secondo la formula:

$$\mathbf{H_{i,j}} = \mathbf{max}(H_{i-1,j-1} + \mathbf{s}(a_i , b_j) , H_{i-1,j} - \mathbf{pgap}, H_{i,j-1} - \mathbf{pgap} , 0)$$

NB: s(a_i , b_j**)** rappresenta una funzione che attribuisce uno score al match o al mismatch osservato e **pgap** è la penalità fissa per i gap.
Il fatto che è presente una funzione **max()** implica che dobbiamo calcolare tutti e tre i possibili valori e poi scegliere il massimo !

Step 8: Compilazione Matrice

Calcolare lo score da inserire nella cella corrente (score di una cella preesistente **+** score per match/mismatch o gap):

PSEUDOCODICE (dichiaraz. variabili):

Imposta max_i = 0
Imposta max_j = 0
Imposta max_score = 0

FUORI dai cicli che percorrono la matrice : vogliamo il massimo valore di TUTTA la matrice

Imposta diagonal_score = 0
Imposta left_score = 0
Imposta up_score = 0
Imposta lettera1 = simbolo corrente in sequenza1
Imposta lettera2 = simbolo corrente in sequenza2

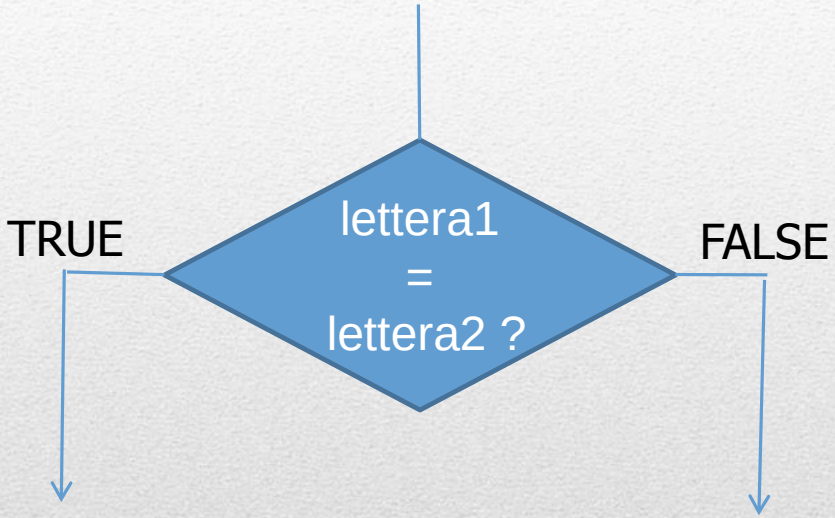
DENTRO ai cicli che percorrono la matrice : servono per calcolo valore cella

Step 8: Compilazione Matrice

Calcolare lo score da inserire nella cella corrente (score di una cella preesistente **+** score per match/mismatch o gap):

3 CASI POSSIBILI:

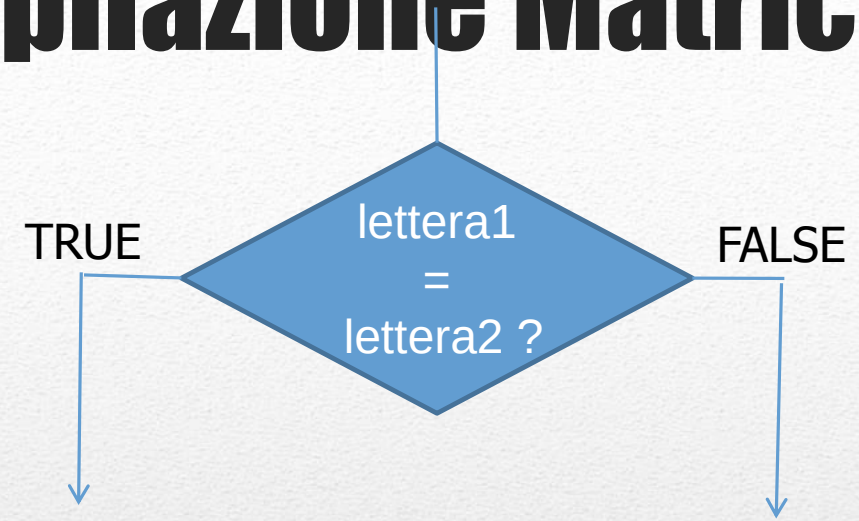
1. match/mismatch
2. gap UP
3. gap LEFT



$diagonal_score = $matrix[$i-1][$j-1]{score} + $MATCH

?

Step 8: Compilazione Matrice



$diagonal_score = $matrix[$i-1][$j-1]{score} + $MATCH

?

```
if($letter1 eq $letter2){
```

TRUE

```
  $diagonal_score = $matrix[$i-1][$j-1]{score} + $MATCH;
}else{
```

FALSE

```
  $diagonal_score = #### FILL IN ######
}
```

Step 8: Compilazione Matrice

3 CASI POSSIBILI:

1. match/mismatch
2. gap UP
3. gap LEFT

```
# calculate gap scores (unweighted linear mode)
 $up_score = #### FILL IN ######
 $left_score = #### FILL IN ######
```

Step 8: Compilazione Matrice

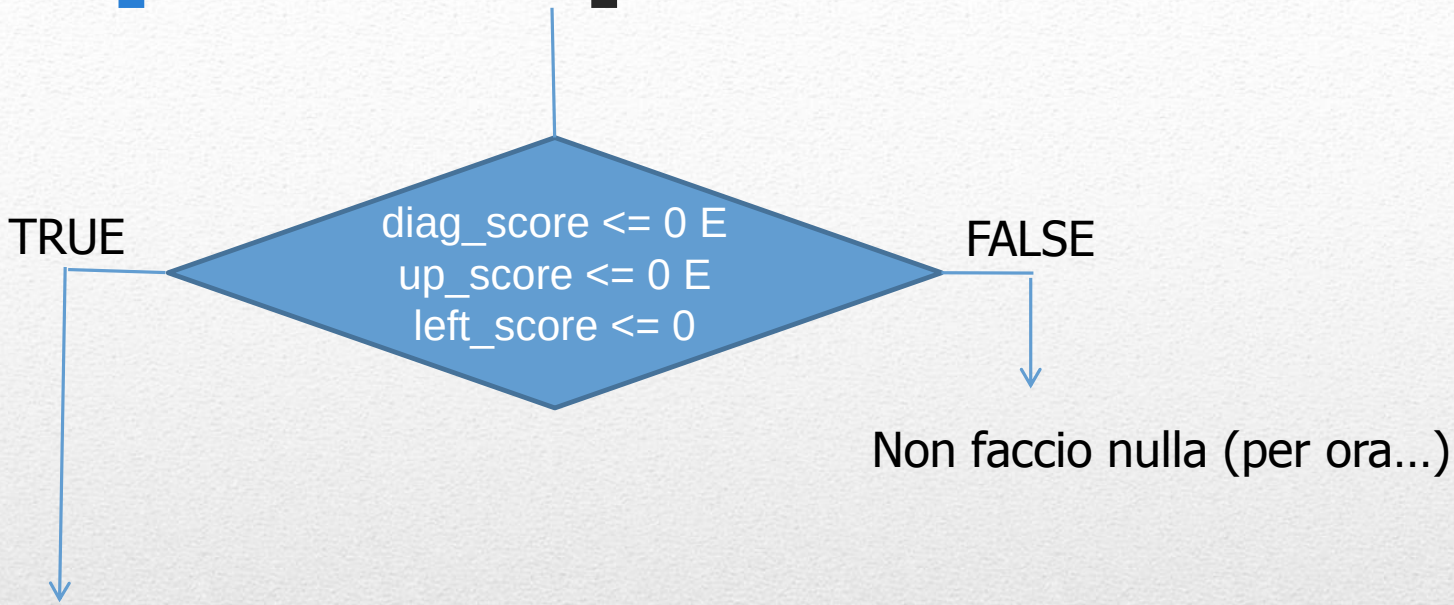
Ora abbiamo gli score di tutti i casi possibili. Possiamo decidere il valore da assegnare alla cella corrente.

$$H_{i,j} = \mathbf{max}(H_{i-1,j-1} + s(a_i , b_j) , H_{i-1,j} - pgap, H_{i,j-1} - pgap , 0)$$

$diagonal_score $up_score $left_score

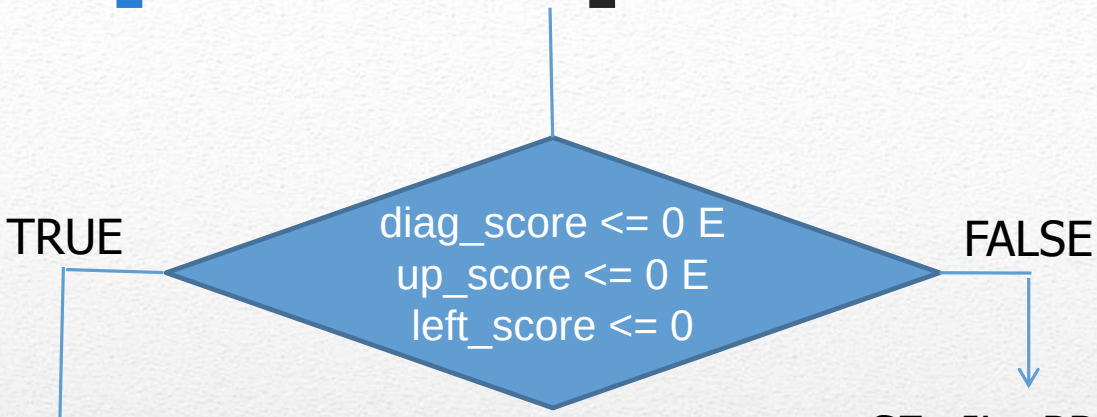
DOBBIAMO SCEGLIERE IL VALORE MASSIMO (TRA 4 VALORI POSSIBILI)

Step 8: Compilazione Matrice




```
if($diagonal_score <= 0 and $up_score <= 0 and $left_score <= 0){

        $matrix[$i][$j]{score} = 0;
        $matrix[$i][$j]{pointer} = "*";
        next;
}
```

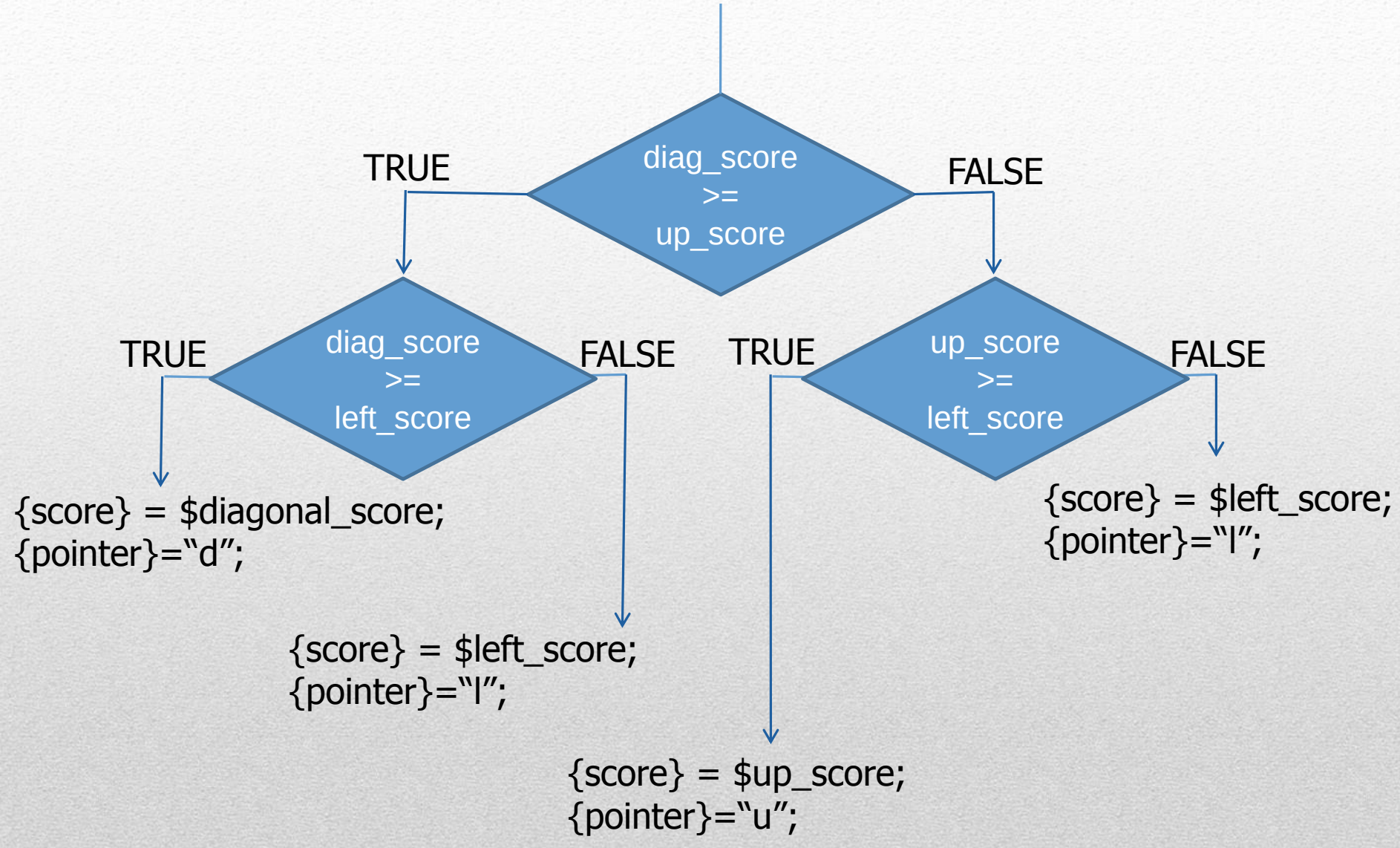
Step 8: Compilazione Matrice



SE IL PROGRAMMA ARRIVA QUI VUOL DIRE CHE **ALMENO UNA** DELLE TRE VARIABILI è > 0 ! **DEVO** SCEGLIERE IL VALORE PIU' ALTO.

```
if($diagonal_score <= 0 and $up_score <= 0 and $left_score <= 0){

        $matrix[$i][$j]{score} = 0;
        $matrix[$i][$j]{pointer} = "*";
        next;
}
```

Step 8: Compilazione Matrice



Step 8: Compilazione Matrice

```
# choose best score

        if ($diagonal_score >= $up_score) {
            if ($diagonal_score >= $left_score) {
                $matrix[$i][$j]{score}   = ### FILL IN ###
                $matrix[$i][$j]{pointer} = ### FILL IN ###
            }
            else {
                ### FILL IN ###################
            }
        } else {
            if ($up_score >= $left_score) {
                ### FILL IN ###################
            }
            else {
                ### FILL IN ###################
            }
        }
```

Step 8: Compilazione Matrice

Abbiamo salvato nella cella corrente sia lo score che il puntatore alla cella che abbiamo utilizzato come “base” per lo score vincente.

L’ultima cosa che resta da fare prima di passare alla cella successiva è verificare se lo score appena salvato è il **MASSIMO** che abbiamo visto fino a questo momento (se è così **salviamo** le coordinate della cella ad il valore dello score).

```
 # set maximum score

        if ($matrix[$i][$j]{score} > $max_score) {
              $max_i = $i;
              $max_j = $j;
              $max_score = $matrix[$i][$j]{score;}
}
```

NB: ricordatevi che queste sono variabili GLOBALI dichiarate FUORI dai cicli for che attraversano la matrice !

Compilazione Matrice

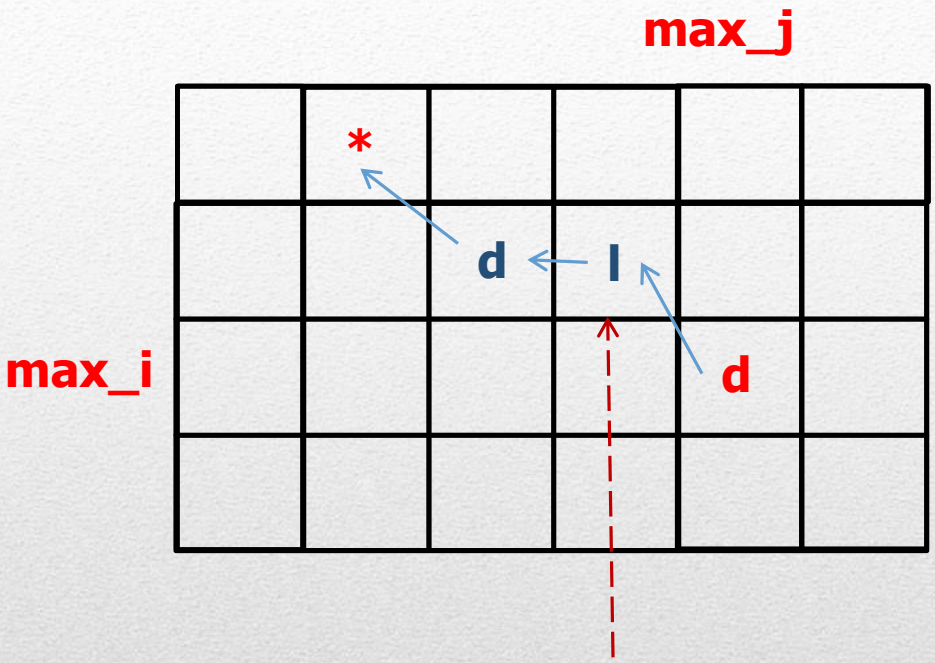
Alla fine dei cicli che attraversano la matrice disponiamo delle seguenti informazioni:

- Matrice degli score
- Matrice dei puntatori
- Valore e coordinate della posizione nella matrice degli score della cella con il punteggio **MASSIMO** (che identifica il **punto di partenza** per la costruzione dell'allineamento durante la fase di traceback)

Step 9: Traceback

E’ realizzato attraverso **un ciclo**. Esso parte dalle coordinate della cella con score massimo identificata durante la compilazione della matrice degli score, segue la direzione indicata dai puntatori e termina SE INCONTRA UN PUNTATORE “NULLO”, che indichiamo con *.

Il ciclo continua FINCHE’ il valore di puntatore NON E’ * ! E’ un ciclo **WHILE**

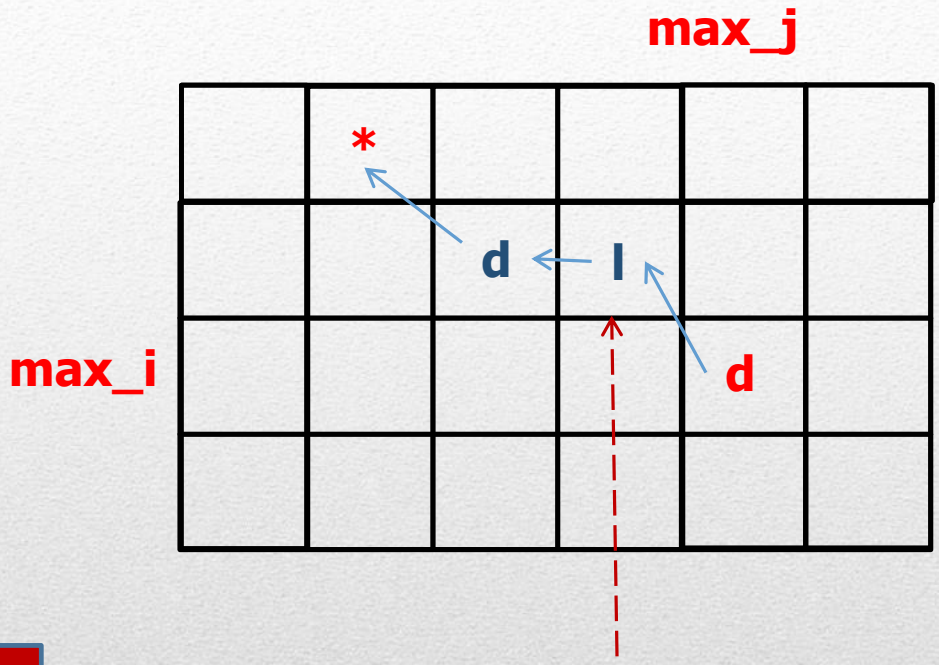


Left => gap nella SECONDA seq. (righe)
Up => gap nella PRIMA seq. (colonne)

Step 9: Traceback

In questa fase:

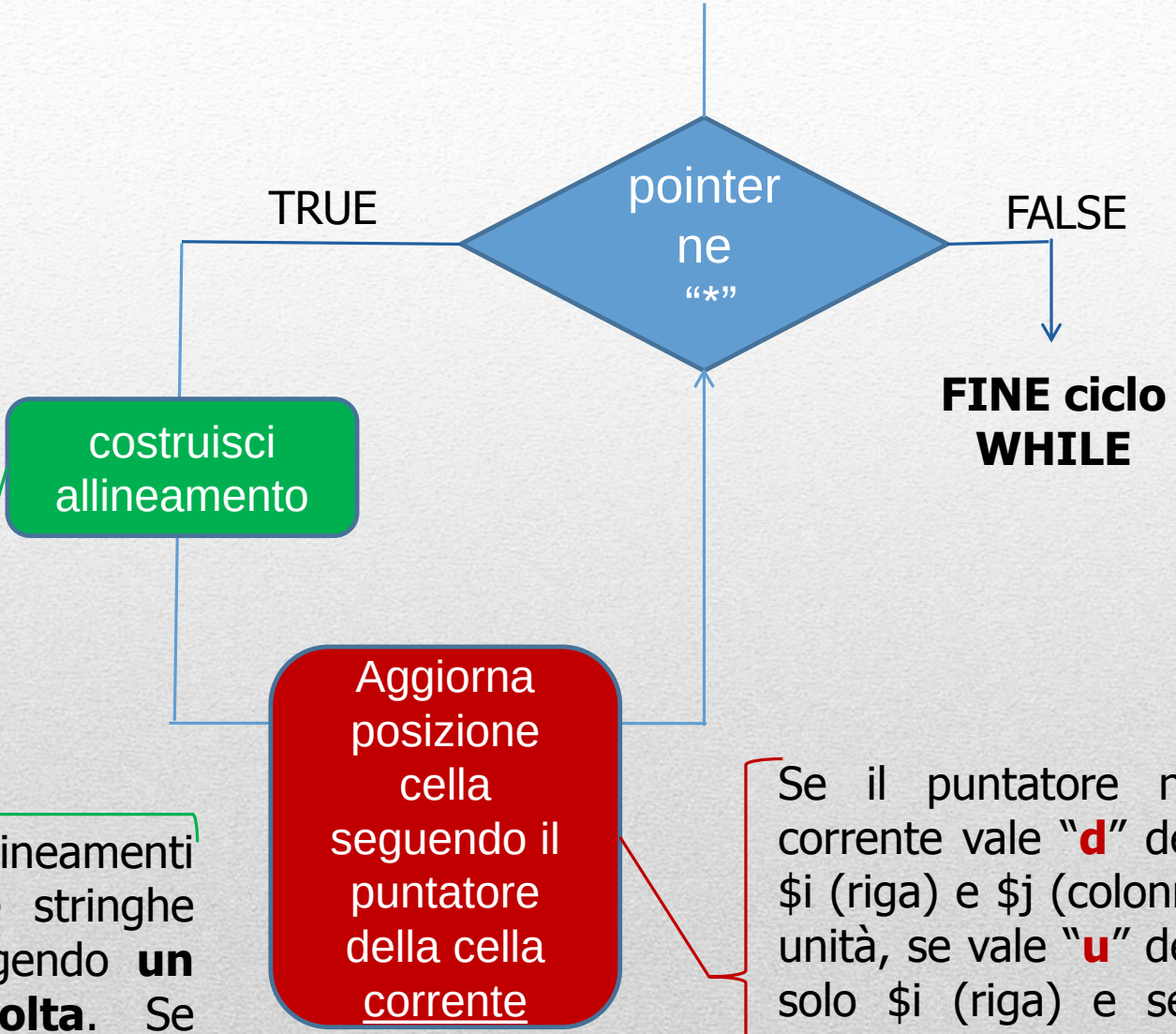
- Coordinate score massimo ci forniscono il punto di **partenza**.
- Primo puntatore nullo incontrato fornisce il punto di **arresto**.



Left => gap nella SECONDA seq. (righe)
Up => gap nella PRIMA seq. (colonne)

NB: siamo in una struttura BIDIMENSIONALE ma, grazie alla programmazione dinamica ci stiamo muovendo in un'UNICA DIMENSIONE (definita dalla serie di puntatori)

Step 9: Traceback



Costruiremo gli allineamenti partendo da due stringhe vuote ed aggiungendo **un carattere alla volta**. Se incontreremo i puntatori up e left aggiungeremo dei gap.

Se il puntatore nella cella corrente vale "**d**" decremento $i (riga) e $j (colonna) di una unità, se vale "**u**" decremento solo $i (riga) e se vale "**l**" decremento solo $j (colonna). Se non aggiorno $i e $j ... il ciclo WHILE non finisce **MAI**.

Step 9: Traceback

```
my $align1 = "";
my $align2 = "";

my $j = $max_j;
my $i = $max_i;

while (1) {
    last if $matrix[$i][$j]{pointer} eq "*";

    if ($matrix[$i][$j]{pointer} eq "d") {
        $align1 .= substr($seq1, $j-1, 1);
        $align2 .= substr($seq2, $i-1, 1);
        $i--; $j--;
    }
    elsif ($matrix[$i][$j]{pointer} eq "l") {
        ### FILL IN ##########################
    }
    elsif ($matrix[$i][$j]{pointer} eq "u") {
        ### FILL IN ##########################
    }
}
```

Alla fine del ciclo queste variabili contengono l'allineamento !

Ci posizioniamo sulla cella con score massimo

START

Se incontro il puntatore * fine WHILE

STOP

Step 10: Stampa allineamento

NB: abbiamo costruito le stringhe dell'allineamento accodando ad una stringa vuota i caratteri che incontravamo percorrendo l'allineamento **DALLA FINE AL PUNTO DI PARTENZA**! Quindi abbiamo l'allineamento … ma AL CONTRARIO.

```
$align1 = reverse $align1;
$align2 = reverse $align2;
print "$align1\n";
print "$align2\n";
```

OUTPUT

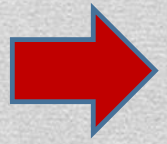
GCCAUUG

GCC-UCG

Esercizi

- Rendere lo script funzionante (**4 pt**) (COMMENTARE IN MANIERA DETTAGLIATA)
- Modificare lo script in modo da passare come argomenti non solo le sequenze ma anche gli score per match, mismatch e gap (**2 pt**)
- Provare ad allineare le sequenze utilizzando questo schema di scoring: match = +1, mismatch = +1/3, gap = 0. Inviate l'allineamento ottenuto e commentate le differenze rispetto all'allineamento ottenuto con lo schema di scoring utilizzato in questa esercitazione (**3 pt**)
- **OPZIONALE:** aggiungere all'output anche la stampa della matrice degli score e la matrice dei puntatori (niente decorazioni … solo le matrici) (**3 pt**)
- **OPZIONALE:** Modificare lo script in modo da ottenere un allineamento globale. Suggerimento: cercate nelle slide delle lezioni teoriche sull'allineamento … ne esiste una che **contiene la soluzione**! Invece di allineare sequenze nucleotidiche allineate queste sequenze proteiche: **HEAGAWGHEE e PAWHEAE** . Al posto dei punteggi fissi per match/mismatch leggete I valori da una matrice BLOSUM50 ed utilizzate una penalità per I **GAP** di -8. (**8 pt**)

cfr. Durbin et al, Biological sequence analysis
Pg. 21



HEAGAWGHE-E
--P-AW-HEAE