Bioinformatica: Allineamento di Sequenze di Aminoacidi

di Bandiera Roberto

La Bioinformatica è una disciplina che si occupa dell'applicazione dell'informatica nell'ambito biologico per consentire lo studio e la simulazione delle strutture e delle funzioni biomolecolari.
Essa si occupa anche di gestire le informazioni di natura biologica in banche dati e di curare il software per l'accesso alle stesse, si pensi ad esempio alla catalogazione dei geni e delle proteine degli esseri viventi.
Questo articolo intende presentare le problematiche e gli algoritmi usati nell'analisi comparativa della struttura primaria di proteine; dal punto di vista informatico si tratta di affrontare un interessante problema di ottimizzazione combinatoria, fino ad arrivare a sviluppare un programma in Java per risolverlo.

Allineamento di sequenze

Attualmente sono in corso molti studi finalizzati ad identificare i geni dell'uomo: finora ne sono stati identificati 30-40 mila; questi geni contengono il codice di proteine la cui funzione è spesso ancora sconosciuta.
Si ricorda che le proteine presentano una complessa struttura tridimensionale; tuttavia, per consentirne un più agevole studio, in prima approssimazione ci si limita a studiarne la "struttura primaria", ovvero la sequenza lineare degli aminoacidi che le costituiscono.
Per cercare di capire la funzione di nuove proteine è essenziale verificare se esse abbiano una sequenza simile a quella di qualche proteina già nota, nell'uomo o in altri organismi.
Inoltre, tale analisi di similarità consente di effettuare studi il cui scopo è quello di individuare le differenze nella sequenza di geni o proteine per capire quali mutazioni siano avvenute nel tempo oppure per determinare la distanza evolutiva tra vari tipi di organismi.
Ecco alcuni esempi.
L'ala di un uccello e la pinna anteriore di una foca si suppone che si siano entrambi evoluti dall'arto anteriore di un rettile ancestrale. In altre parole si presuppone l'esistenza di un organismo ancestrale comune da cui tali strutture si sono evolute.
Lo studio della mutazione dei geni dell'emoglobina tra una persona sana e una malata di anemia falciforme ha portato a scoprire che è sufficiente una mutazione puntiforme per determinare una differenza nella struttura tridimensionale di tale proteina e quindi per comportare una minore capacità di legare ossigeno e causare, quindi, tale malattia.
L'insulina del maiale e dell'uomo hanno la stessa funzione, eppure sono diverse nella sequenza, infatti appartengono a specie diverse di mammiferi che si sono distanziati evolutivamente.
Per la cura del diabetes mellitus si somministra insulina.
Fino a qualche tempo fa l'insulina veniva prelevata dai maiali, ma recentemente si è riusciti a farla produrre da batteri, nella quantità e qualità desiderate.
Il motivo per cui l'insulina animale poteva essere utilizzata per curare il diabete è dato dalla elevata similarità tra le proteine di origine animale con quella umana.
Per mostrare questo fatto, si riporta l'allineamento di un frammento di insulina dell'uomo con quella del maiale; come spiegato nel box relativo al codice genetico, gli aminoacidi sono simbolicamente rappresentati con lettere dell'alfabeto. La riga centrale (match) riporta gli aminoacidi identici delle due sequenze.

```
FVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQVGQVELGGGPG...(uomo)
FVNQHLCGSHLVEALYLVCGERGFFYTPK RREAE  Q G VELGGG G...(match)
FVNQHLCGSHLVEALYLVCGERGFFYTPKARREAENPQAGAVELGGGLG...(maiale)
```

Risulta che esse sono identiche per l'88%, ovvero 43 aminoacidi su 49.

Punteggi di similarità

Per determinare quanto due sequenze siano simili è essenziale trovare il migliore modo con cui esse possono essere allineate.
Il problema dell'allineamento di sequenze è molto complesso e costituisce un campo di ricerca della bioinformatica che è in continua evoluzione.
Si deve identificare l'allineamento migliore ovvero quello a cui corrisponde il maggiore punteggio di similarità. Per calcolare i punteggi di similarità ci sono diversi criteri.
Un primo criterio è quello di assegnare 1 punto per l'identità e 0 punti per la mancata corrispondenza (mismatch) tra aminoacidi delle sequenze allineate.
Tuttavia non va bene penalizzare tutti i mismatch allo stesso modo: infatti ci sono aminoacidi simili tra loro che possono essere interscambiati senza interferire significativamente sulla struttura tridimensionale della proteina, oppure vi possono essere aminoacidi carichi elettricamente che se vengono scambiati con aminoacidi di carica opposta o neutra determinano notevoli modifiche nella struttura della proteina.
Per quantificare il grado di similarità di un allinamenento si introduce una matrice, detta "matrice di sostituzione", che per ogni possibile appaiamento di aminoacidi dà un punteggio di similarità.
Si tratta di matrici 20x20 simmetriche; come esempio si riporta la matrice PAM 240:

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-4	1	1	1	-6	-4	0
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-5	0	0	-1	2	-4	-3
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-4	-1	1	0	-4	-2	-2
D	0	-1	2	4	-5	2	4	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2
C	-2	-4	-4	-5	12	-6	-6	-4	-4	-2	-6	-6	-5	-5	-3	0	-2	-8	0	-2
Q	0	1	1	2	-6	4	3	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2
E	0	-1	1	4	-6	3	4	0	1	-2	-3	0	-2	-6	-1	0	0	-7	-4	-2
G	1	-3	0	1	-4	-1	0	5	-2	-3	-4	-2	-3	-5	-1	1	0	-7	-5	-1
H	-1	2	2	1	-4	3	1	-2	7	-3	-2	0	-2	-2	0	-1	-1	-3	0	-2
I	-1	-2	-2	-2	-2	-2	-2	-3	-3	5	2	-2	2	1	-2	-1	0	-5	-1	4
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2
K	-1	3	1	0	-6	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-4	-5	-3
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	7	0	-2	-2	-1	-4	-3	2
F	-4	-5	-4	-6	-5	-5	-6	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1
P	1	0	-1	-1	-3	0	-1	-1	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-3	-3	-1
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-4	-4	0	-6	-3	-5	17	0	-6
Y	-4	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-5	-3	7	-5	-3	-3	0	10	-3
V	0	-3	-2	-2	-2	-2	-2	-1	-2	4	2	-3	2	-1	-1	-1	0	-6	-3	4

Tabella: matrice PAM240

Esistono molte matrici di sostituzione, basate su diversi criteri. Le più comuni sono le PAM (Mutation Probability Matrix) e le BLOSUM (Blocks Substitution Matrix) che sono basate

rispettivamente sulle distanze evolutive e sulle frequenze di sostituzione osservate in famiglie di proteine simili.
Ma esistono altre matrici di sostituzione basate su criteri di similarità chimica, oppure sul numero minimo di mutazioni del DNA necessarie per sostituire il codone di un aminoacido in un altro. Ulteriori informazioni possono essere trovate ai seguenti siti: http://www.ncbi.nih.gov/Education/BLASTinfo/Scoring2.html e http://barton.ebi.ac.uk/papers/rev93_1/section3_3.html.
Un altro problema da affrontare è come fare per confrontare sequenze di lunghezza diversa: viene quindi introdotto il concetto di "gap".

Gap e penalità

Nel processo di replicazione del DNA vi possono essere degli errori, i quali provocano delle mutazioni nel DNA stesso: si può trattare di mutazioni puntiformi (che riguardano un singolo nucleotide), ma anche di delezioni, inversioni o duplicazioni interne (che riguardano sequenze formate da più nucleotidi).
Per tenere conto di questa seconda categoria di possibili mutazioni del DNA, e quindi delle proteine da esso codificate, nell'ambito dell'allineamento di due sequenze si introduce il simbolo di "gap", si tratta di un trattino "–", il quale funge da segnaposto di aminoacidi mancanti rispetto alla continuità dell'allineamento.
In questo modo si possono confrontare anche sequenze di lunghezza diversa.
La presenza di un gap introduce un componente negativo nel calcolo del punteggio di similarità di due sequenze: ogni singolo gap potrebbe essere penalizzato con un punteggio negativo (tipicamente –11); talvolta si preferisce differenziare attribuendo una elevata penalità alla creazione di un gap (tipicamente -11) e una penalità più bassa alla prosecuzione dello stesso (tipicamente -1).

Esempio di allineamento

Date le due ipotetiche sequenze: `IKDLLVSSS (alfa)` e `IRNILQPSSVDS (beta),` alcuni possibili allineamenti sono:

```
IKDLLVSSS--- (alfa)
I+++L +SS    (match)
IRNILQPSSVDS (beta)
==> punteggio 8

IKDLLV-SS--S (alfa)
I+++L  SS    (match)
IRNILQPSSVDS (beta)
==> punteggio -1

IKDLL-VSS--S (alfa)
I+++L  SS  S (match)
IRNILQPSSVDS (beta)
==> punteggio 0
```

dove i punteggi sono calcolati con la matrice PAM 240 e le penalità di gap sono 11 per la creazione e 1 per l'estensione.
Il simbolo + nella riga di match indica un punteggio di appaiamento positivo.

Il primo dei tre allineamenti è il migliore possibile, ovvero è l'allineamento ottimo, a queste condizioni di calcolo dei punteggi.

L'algoritmo di allineamento

L'algoritmo che viene generalmente utilizzato per risolvere questo problema di determinare il migliore allineamento di due sequenze è l'algoritmo di programmazione dinamica di Needleman-Wunsch.
Esso sfrutta il fatto che il punteggio totale di allineamento è dato dalla semplice somma di tanti punteggi di accoppiamento di singoli aminoacidi.
L'allineamento ottimo viene calcolato ricorsivamente per sottosequenze via via più lunghe, a partire dal caso banale di sottosequenze vuote.
Questo grazie al fatto che per calcolare l'allineamento ottimo di due sequenze è sufficiente conoscere l'allineamento ottimo delle sottosequenze più corte di una unità rispetto a quelle considerate.
Si debbano allineare le sequenze a e b, di lunghezza, rispettivamente, n e m: $a_1..a_n$ e $b_1..b_m$.
Il migliore allineamento delle sottosequenze $a_1..a_i$ e $b_1..b_j$ (con i <= n e j <= m), al quale corrisponde il punteggio di similarità S(i, j), si ottiene in uno dei seguenti 3 modi alternativi:

1) estendendo l'allineamento delle sottosequenze $a_1..a_{i-1}$ e $b_1..b_{j-1}$ con l'appaiamento di a_i con b_j,
2) oppure estendento quello di $a_1..a_{i-1}$ e $b_1..b_j$ con l'appaiamento di a_i con un gap,
3) oppure estendento quello di $a_1..a_i$ e $b_1..b_{j-1}$ con l'appaiamento di b_j con un gap.

In questi 3 casi si otterrebbero rispettivamente i seguenti punteggi di similarità:

1) $S'(i, j) = S(i-1, j-1) + m(a_i, b_j)$
2) $S''(i, j) = S(i-1, j) – d$
3) $S''(i, j) = S(i, j-1) – d$

dove $m(a_i, b_j)$ è il punteggio di appaiamento derivante dalla prescelta matrice di sostituzione, mentre d è la penalità del gap.
Di queste tre possibilità si sceglie quella che comporta il punteggio maggiore:

$S(i, j) = \max \{ S'(i, j), S''(i, j) , S'''(i, j)\}$

Si tratta di una procedura ricorsiva per calcolare S(i, j) le cui condizioni iniziali sono banali:

- l'allineamento ottimo di due sequenze vuote ha punteggio nullo: S(0, 0) = 0
- l'allineamento ottimo di una sequenza con una sequenza vuota ha punteggio pari ad una sequenza di gap: S(i, 0) = -d * i e S(0, j) = -d * j

Si può costruire una matrice per mantenere i valori di S(i, j) con i da 0 a n e j da 0 a m.
Pertanto, a partire dalla prima riga e dalla prima colonna della matrice S, si può procedere riga per riga per calcolare tutte le celle della stessa.
Si termina quando si arriva a calcolare S(n, m), che corrisponde al punteggio dell'allineamento ottimo delle due sequenze.

L’allineamento ottimo, che non è necessariamente unico, può essere facilmente ricostruito, se per ogni passo è stato mantenuto memoria di come si è arrivati a calcolare S(n, m).

Ad esempio, si vogliono allineare le sequenze AWGH e AGH, si costruisce la seguente matrice per tenere traccia dei punteggi ottimi S(i, j).
Si usa la matrice PAM 240 e la penalità di gap pari a 11.

S	-	A	W	G	H
-	*0*	-11	-22	-33	-44
A	-11	*2*	*-9*	-20	-31
G	-22	-9	-5	*-4*	-15
H	-33	-20	-12	-7	*3*

Per la realizzazione dell’algoritmo si deve tenere in memoria il predecessore di ciascuna cella, mediante una opportuna matrice; si codificano le direzioni Nord = 4, Ovest = 1, NO = 2, in questo modo è possibile codificare anche combinazioni di più predecessori, nel caso di uguaglianza di punteggi: ad esempio Nord + NO = 6.

P	-	A	W	G	H
-	*0*	1	1	1	1
A	4	*2*	*1*	1	1
G	4	4	2	*2*	1
H	4	4	2	2	*2*

L’allineamento ottimo ha punteggio +3:

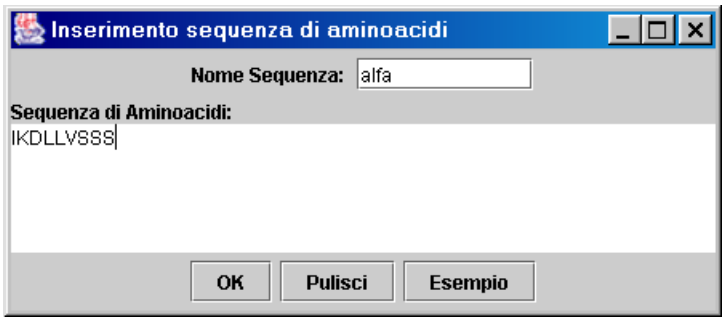
```
AWGH (a)
A GH (match)
A-GH (b)
```

Il suddetto algoritmo può anche tenere conto della differenziazione delle penalità per la creazione e l’estensione di un gap, sfruttando la memoria data dalla matrice dei predecessori per distinguere tra le due suddette situazioni.

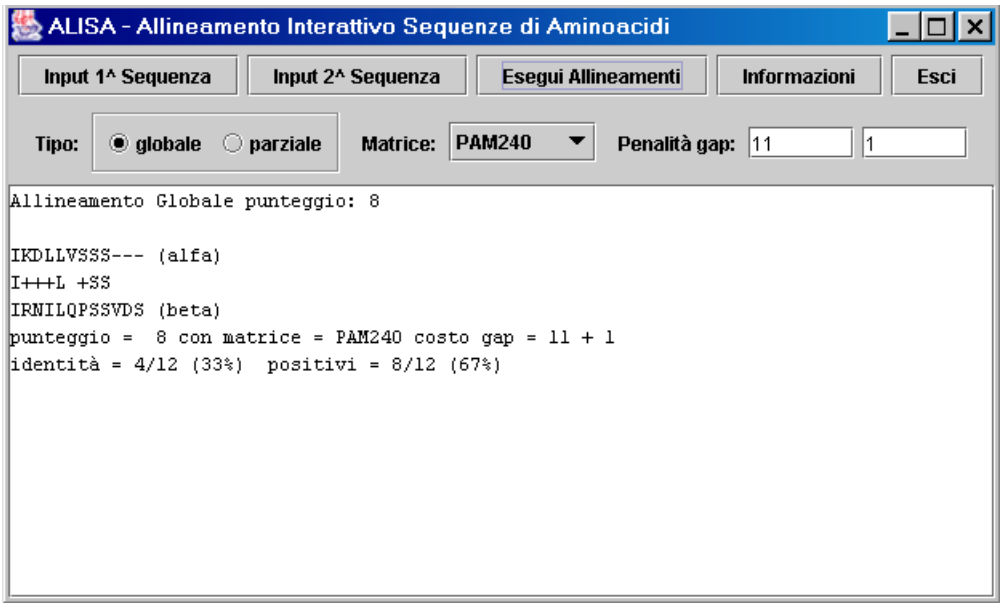
Finora è stato cercato il migliore allineamento “globale” di due sequenze, ma quando si vuole cercare il migliore allineamento di una sequenza breve (a1..an) all’interno di una sequenza molto lunga (b1..bm) non si devono penalizzare i gap iniziali, che corrispondono alla ricerca del punto iniziale dell’allineamento, e nemmeno quelli finali; pertanto si applica il suddetto algoritmo con S(0, j) = 0 e S(0, i) = 0; si termina quando si arriva a calcolare S(n, j) per un qualche j, oppure S(i, m) per un qualche i, ovvero quando è stato completato l’allineamento della sequenza più breve rispetto all’altra. In questo caso si parla di allineamento “parziale”.

Il programma ALISA

Viene ora descritta l'architettura del programma ALISA, scritto in Java, per effettuare l'allineamento di sequenze di aminoacidi secondo il suddetto algoritmo.

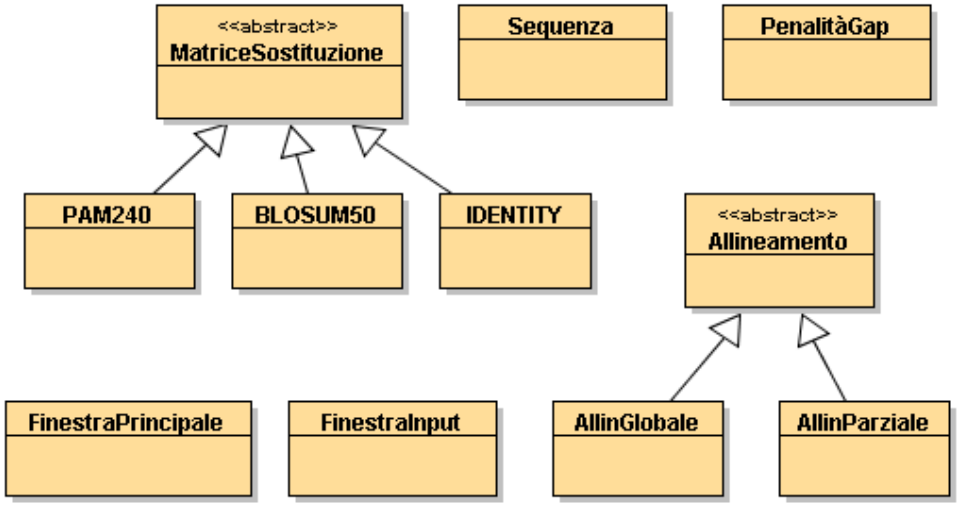


"Input di una sequenza"



Allineamento ottimo con ALISA"

Il diagramma delle classi che costituiscono ALISA, mostra le classi principali che sono la classe Sequenza, per memorizzare e gestire le sequenze di aminoacidi, e la classe astratta Allineamento, che definisce i metodi astratti per la determinazione dell'allineamento ottimo di due sequenze.
Dalla classe Allineamento derivano le due classi AllinGlobale e AllinParziale, che realizzano concretamente i metodi dichiarati dalla superclasse e differiscono unicamente per le condizioni iniziali poste all'algoritmo di Needleman-Wunsch.
A corollario di queste classi c'è la classe PenalitàGap, che gestisce la semplice struttura dati che memorizza il criterio di penalizzazione dei gap, e la classe astratta MatriceSostituzione, concretizzata da un elenco di matrici concrete, come la PAM240.
In particolare, la matrice IDENTITY assegna il punteggio 1 all'accoppiamento di due aminoacidi identici e 0 altrimenti.
Infine, vi sono le due classi relative alle finestre che realizzano l'interfaccia grafica, sui cui dettagli non ci si sofferma.



"Il diagramma delle classi"

Vengono ora illustrati i tratti salienti del codice.
La classe Sequenza rappresenta una proteina come sequenza di aminoacidi, ovvero come una stringa di caratteri e corrispondentemente come un array di indici numerici relativi alle posizioni che tali aminoacidi assumono nelle matrici di sostituzione.

```
public class Sequenza
{
 private String nome;
 private String a; //la sequenza di aminoacidi
 private int[] b; //gli indici numerici degli aminoacidi
```

In particolare risulta interessante il metodo che assegna un elenco di aminoacidi alla sequenza, esso riceve come parametro una stringa contenente l'elenco di aminoacidi da assegnare e restituisce 0 se tutto è OK, -1 se invece ci sono degli aminoacidi non validi nella stringa fornita in input.

```
public int impostaAminoacidi(String elencoAminoacidi)
{ int esito = 0;
  elencoAminoacidi = elencoAminoacidi.trim(); //elimina eventuali spazi
  elencoAminoacidi = elencoAminoacidi.toUpperCase(); //converte in maiuscolo
  a = elencoAminoacidi;
  int n = a.length();
  b = new int[n];
  int i = 0;
  char c;
  //stringa che presenta gli aminoacidi nell'ordine giusto
  String ordine = "ARNDCQEGHILKMFPSTWYV";
  while ((i < n) && (esito == 0))
  {
    c = a.charAt(i);
    b[i] = ordine.indexOf(c);
    if (b[i] == -1)  //aminoacido non valido
      { esito = -1; }
    i++;
  }
  return esito;
}
```

Risulta molto utile anche il metodo che ritorna l'indice numerico dell'i-esimo aminoacido della sequenza.

```
public int indiceAminoacido(int i)
  { //deve essere (i < lunghezza())
    return b[i];
  }

... //altri metodi omessi

} //fine classe
```

La classe astratta MatriceSostituzione, fornisce concretamente tutti i metodi per la gestione di una matrice di sostituzione, ma lascia alle sottoclassi concrete la specificazione del contenuto della matrice.

```
public abstract class MatriceSostituzione
{
  private int[][] m; //matrice di punteggi
  private String tipo;
```

Il costruttore verrà utilizzato dalle sottoclassi concrete:

```
public MatriceSostituzione(String unTipo)
 {
   tipo = unTipo;  //es. “PAM240”
 }
```

Ecco il metodo che assegna effettivamente la matrice di punteggi

```
public void impostaMatrice(int[][] unaMatrice)
 {
   m = unaMatrice;
 }
```

Infine ecco il metodo che ritorna il valore di una cella

```
public int punteggio(int i, int j)
 { //deve essere ((i < 20) && (j < 20))
  return m[i][j];
 }

} //fine classe
```

La classe concreta PAM240, si limita ad assegnare il tipo e il contenuto della matrice di sostituzione

```
public class PAM240 extends MatriceSostituzione
{

// costruttore
public PAM240()
{
  super("PAM240");
  int[][] m;  //matrice 20x20
  m = { {2, -2, ...,  0},
        {-2, 6, ..., -3},
        ...
      };
  impostaMatrice(m);
}

} //fine classe
```

E’ interessante il modo in cui nella FinestraPrincipale si crea e istanzia un oggetto di tipo MatriceSostituzione, prelevando dal ComboBox la stringa s contenente il nome della classe della matrice e richiamando il costruttore della classe selezionata mediante il metodo newInstance() applicato alla classe specificata a run-time con l’istruzione Class.forName(s):

```
...
MatriceSostituzione mat;
String s = (String) comboBox.getSelectedItem();  //es. “PAM240”
mat = (MatriceSostituzione) Class.forName(s).newInstance();
...
```

il risultato è lo stesso che se si fosse fatto:

```
...
if (s.equals(“PAM240”))
   { mat = new PAM240(); }
else if (s.equals(“BLOSUM50”))
   { mat = new BLOSUM50(); }
else if (s.equals(“IDENTITY”))
   { mat = new IDENTITY(); }
```

ma con ovvia scomodità di implementazione qualora si avessero molti tipi di matrici.
Per maggiore chiarezza ecco un altro esempio di utilizzo di questa tecnica per creare un oggetto di classe Persona, dove il nome della classe Persona viene fornito a tempo di esecuzione:

```
Class c = Class.forName("Persona");
Persona p = (Persona) c.newInstance();
System.out.println("ho creato una " + c.getName()); // “ho creato una Persona”
```

Per quanto riguarda la classe AllinGlobale, essa contiene la matrice dei punteggi di similarità e la matrice dei predecessori, per consentire di ricostruire l’allineamento ottimo.

```
public class AllinGlobale extends Allineamento
{
  private int[][] s; // matrice dei punteggi di similarità
  private byte[][] p; // matrice dei percorsi: da cella x alla precedente
   // che possono essere 1, 2, 4, 1+2=3, 1+4=5, 2+4=6, 1+2+4=7
   //    [2]  [4]  []
   //    [1]  [x]  []
   //    []   []   []
```

Ecco il metodo che calcola l'allineamento globale ottimo secondo l’algoritmo di Needleman-Wunsch, riempendo le due matrici s e p

```
public void eseguiAllineamento()
{ //prelevo le sequenze da allineare, la mat di sostituzione e le penalità
  //dalla superclasse Allineamento
  Sequenza a = sequenza1();
  Sequenza b = sequenza2();
  MatriceSostituzione mat = matriceSostituzione();
  PenalitàGap pg = penalitàGap();
  int n = a.lunghezza();
  int m = b.lunghezza();
  int d = pg.costoCreazione();
  int e = pg.costoEstensione();
  //inizializzo le celle della prima colonna e della prima riga come previsto
  ...//codice omesso
  //ed ora calcolo tutti i punteggi di similarità
  int s1, s2, s4;
  int x, y;
  int max, prec;
  i = 1; //si parte dalla seconda riga
  while (i < n+1) //le righe sono n+1 perché la prima riga è per il gap iniziale
  {
```

```
    j = 1; //si parte dalla seconda colonna
    while (j < m+1)
    {
      x = a.indiceAminoacido(i-1);
      y = b.indiceAminoacido(j-1);
      //se non si dovesse tener conto delle differenti penalità di gap
      //calcolo i tre possibili punteggi
      s1 = s[i][j-1] - d;
      s2 = s[i-1][j-1] + mat.punteggio(x,y);
      s4 = s[i-1][j] - d;
      //cerco il massimo punteggio
      prec = 1; //vai a sinistra
      max = s1;
      if (s2 > max)
      { prec = 2; //vai in diagonale
        max = s2;
      }
      else if (s2 == max)
      { prec = prec + 2; } //vai anche in diagonale
      if (s4 > max)
      { prec = 4; //vai in alto
        max = s4;
      }
      else if (s4 == max)
      { prec = prec + 4; } //vai anche in alto
      s[i][j] = max;
      p[i][j] = (byte) prec;
      j++;
    }
    i++;
  }
}
```

Se invece si vuole tenere conto delle diverse penalità di gap, cambia il modo in cui vengono calcolati i punteggi s1 e s4, perché si deve guardare nella matrice p se precedentemente c'era un gap oppure no.

```
  if (p[i][j-1] == 1)  //se precedentemente c'era un gap
  { s1 = s[i][j-1] - e; }
  else
  { s1 = s[i][j-1] - d; }
  if (p[i-1][j] == 4) //se precedentemente c'era un gap
  { s4 = s[i-1][j] - e; }
  else
  { s4 = s[i-1][j] - d; }
```

Il punteggio dell'allineamento ottimo viene dato da un metodo che restituisce il contenuto della cella di indici n, m della matrice s.
Per determinare l'allineamento ottimo, si deve utilizzare la matrice p dei percorsi, con l'accortezza, però, di agire in modo ricorsivo in quanto si potrebbero avere più percorsi ottimi: è come seguire tutti i cammini in un albero dalla radice alle foglie.

```
//il metodo ritorna un array di stringhe, ciascuna delle quali rappresenta
//una singola riga di testo dell'output che verrà visualizzato nella finestra
//principale del programma
public String[] percorsoOttimo()
 { //codice omesso
 }
```

Conclusioni

La complessità computazionale dell'algoritmo di Needleman-Wunsch è quadratica O(n*m); essa è troppo elevata se si vuole confrontare una sequenza con tutte quelle presenti in una banca dati.
Pertanto si ricorre ad algoritmi più efficienti basati su tecniche euristiche (che però non sempre individuano la soluzione ottima); tra i pacchetti software più usati c'è BLAST.
In conclusione l'uso di software aiuta il biologo nelle sue ricerche ma il fatto che esso si basa su modelli semplificati della realtà può portare a false interpretazioni, risulta quindi indispensabile usare sempre una buona dose di buon senso e di conoscenze biologiche.

BOX LATERALI

In allegato

Il programma sorgente e l'eseguibile ALISA.JAR (dove la main class è FinestraPrincipale).

La programmazione dinamica

Il metodo della programmazione dinamica per la ricerca della soluzione di problemi di ottimizzazione combinatoria si basa sul seguente principio di ottimalità: se è dato il percorso ottimo tra i punti A e B e si prende un qualsiasi punto C di questo percorso come nuovo punto di partenza, allora il nuovo percorso ottimo coincide con quello originario da C fino a B.
Grazie a questo principio, per la ricerca della soluzione ottima si parte dal punto di arrivo e si torna indietro esplorando le diverse alternative.
Si assume che i possibili percorsi siano costituiti di un numero finito di passi e che tutti i possibili percorsi siano anch'essi in numero finito.

La struttura delle proteine

E' importante conoscere la struttura tridimensionale delle proteine perché da essa dipende la loro funzione biologica. Tale struttura dipende dalla sequenza di aminoacidi che la compongono e dalle proprietà chimiche degli stessi.
Ma la determinazione della struttura delle proteine è ancora un problema aperto: solo per macromolecole molto piccole gli attuali algoritmi di modellazione sono in grado di predire una struttura che corrisponde ad un minimo di energia.
A tale scopo talvolta si utilizzano anche programmi di simulazione basati su "algoritmi genetici", i quali permettono di determinare la struttura di una proteina come soluzione di un complesso problema di ottimizzazione.

BLAST

Ci sono vari programmi preposti all'analisi delle sequenze; il più importante è BLAST http://www.ncbi.nlm.nih.gov/BLAST/ oppure http://blast.wustl.edu.
Lo scopo è quello di valutare la similarità di sequenze di nucleotidi o di aminoacidi: l'utente inserisce una particolare sequenza, detta "query", che verrà confrontata con tutte quelle presenti in un database, come ad esempio GenBank contenente oltre 1.000.000 di sequenze, e si otterrà l'elenco delle sequenze con i più alti punteggi di similarità con la query.
Esempio estratto da una query con BLAST:

```
Score =  251, Identities = 50/79 (63%), Positives = 62/79 (78%)
Matrix: BLOSUM62    Gap Penalties: Existence -11, Extension -1

Query: IKDLLVSSSTDLDTTLVLVNAIYFKGMW
       I+++L  SS D  T +VLVNAI FKG+W
Sbjct: IRNILQPSSVDSQTAMVLVNAIAFKGLW


   KTAFNAEDTREMPFHVTKQESKPVQMMCMNNS
   + AF AEDT+ +PF VT+QESKPVQMM    S
   EKAFKAEDTQTIPFRVTEQESKPVQMMYQIGS


   FNVATLPAEKMKILELPFA  (ovalbumin-related protein chicken)
   F VA++ +EKMKILELPFA
   FKVASMASEKMKILELPFA  (ovalbumin - Japanese quail)
```

Banche dati

Ci sono molte banche dati di sequenze di DNA e di proteine. I due maggiori centri bioinformatici mondiali sono il N.C.B.I. di Washington negli Stati Uniti e l'E.B.I. a Cambridge in Europa.
Inoltre ci sono altre importanti banche dati come ad esempio il database di tassonomia che contiene le informazioni relative a molti organismi biologici e la loro classificazione sistematica; oppure il database di referenze bibliografiche di interesse biologico e medico PubMed che costituisce un'importante fonte di informazione in cui possono essere facilmente reperite informazioni bibliografiche e riassunti delle pubblicazioni scientifiche.

Il codice genetico

Il DNA è formato da una catena di nucleotidi contenenti basi azotate, chiamate Adenina, Citosina, Guanina e Timina, abbreviate con le loro iniziali: A, C, G, T.
Pertando il DNA, dal punto di vista computazionale, può essere visto come una sequenza di diverse migliaia di caratteri; ad esempio:

```
GTTAATTACTAATCAGCCCATGATCATAACATAACT...
```

Il DNA contiene la codifica delle proteine, che sono composte di sequenze di aminoacidi.
Ci sono 20 diversi aminoacidi, ciascuno dei quali è codificato da una sequenza di 3 basi, detta tripletta o codone.

La sequenza con cui gli aminoacidi si succedono determina le proprietà di ogni proteina.
Esistono proteine di lunghezze molto diverse, da pochi a diverse migliaia di aminoacidi.
Il codice genetico è lo stesso per tutti gli organismi; esso traduce le diverse possibili triplette di nucleotidi nei 20 aminoacidi.
La tabella riporta i 20 aminoacidi con le loro abbreviazioni a 3 e a una lettera e le triplette di nucleotidi che li codificano (gli * indicano T, C, A o G). Alcune particolari triplette (start e stop) indicano l'inizio e la fine della codifica di una proteina nell'ambito della catena del DNA.

Nome Aminoacido	**Simboli**		**Triplette**
Alanine	Ala	A	GC*
Arginine	Arg	R	CG* AGA AGG
Asparagine	Asn	N	AAU AAC
Aspartic Acid	Asp	D	GAU GAC
Cysteine	Cys	C	UGU UGC
Glutamine	Gln	Q	CAA CAG
Glutamic Acid	Glu	E	GAA GAG
Glycine	Gly	G	GG*
Histidine	His	H	CAU CAC
Isoleucine	Ile	I	AUU AUC AUA
Leucine	Leu	L	UUA UUG CU*
Lysine	Lys	K	AAA AAG
Methionine	Met	M	AUG
Phenylalanine	Phe	F	UUU UUC
Proline	Pro	P	CC*
Serine	Ser	S	UC* AGU AGC
Threonine	Thr	T	AC*
Tryptophan	Trp	W	UGG
Tyrosine	Tyr	Y	UAU UAC
Valine	Val	V	GU*
	start		AUG
	stop		UAA UAG UGA

TABELLA: Il codice genetico

Si nota che ci sono 4^3 = 64 possibili triplette per codificare 20 aminoacidi, di conseguenza molti aminoacidi sono codificati da più codoni.
Pertanto anche le proteine sono riconducibili, in prima approssimazione, a stringhe di caratteri.
Ad esempio: `QIKDLLVSSSTDLDTTLVLVNAIYFKGM..`

Bibliografia

Allineamento di coppie di sequenze – Nicola Vitacolonna (Univ. di Udine) (http://www.dimi.uniud.it/~vitacolo)
A Short Introduction to Biocomputing (http://www.uni-mainz.de/~cfrosch/bc4s/welcome.html)
BioComputing Hypertext Coursebook (http://www.techfak.uni-bielefeld.de/bcd/welcome.html)

Genetic Code (MIT course of Information and Entropy) (http://www-

mtl.mit.edu/Courses/6.095/notes/genetic_code.html)