Algoritmi e linguaggi di programmazione

Giorgio Valentini

e –mail: *valentini@dsi.unimi.it*

DSI – Dipartimento di Scienze dell' Informazione

Università degli Studi di Milano

Definizione intuitiva di algoritmo

Un **algoritmo** si può definire come un *procedimento* che consente di *ottenere* un *risultato* eseguendo, in un determinato ordine, un insieme di *passi semplici* corrispondenti ad azioni scelte solitamente da un insieme finito.

Esempi:

- Una procedura per il calcolo del minimo comune multiplo fra due numeri naturali
- La ricetta di una torta
- Una procedura per ordinare un insieme di oggetti in base ad un criterio.

Il termine algoritmo deriva dall'arabo

Abū Ja-far Muhammad ibn Mūsā Khwārizmī - in arabo: محمد خوارزمی - Baghdad, IX secolo matematico, astronomo, e geografo persiano.

Una pagina dall' *Algebra* di al- Khwārizmī

Caratteristiche degli algoritmi

- La sequenza di istruzioni deve essere finita (*finitezza*);
- La procedura deve portare ad un risultato (*effettività*);
- Le istruzioni devono essere eseguibili materialmente (*realizzabilità*);
- Le istruzioni devono essere espresse in modo non ambiguo (*non ambiguità*).

Problemi e algoritmi

Un algoritmo risolve un *problema* (come funzione di corrispondenza fra spazio delle istanze e delle soluzioni)

Esempio:

Problema: Allineamento di due sequenze di DNA

Istanze: Coppie di sequenze

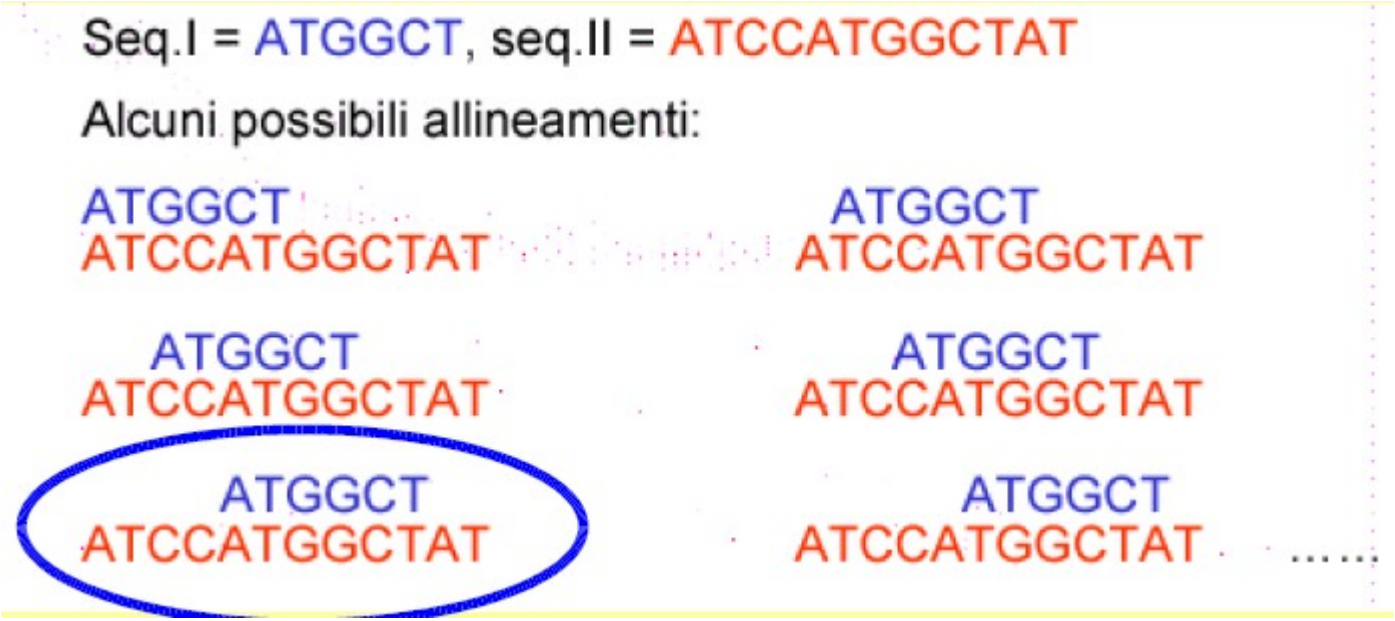
Soluzioni: Giustapposizioni delle due sequenze (che massimizzino la similarita')

Es: algoritmo di allineamento fra due sequenze

Input: 2 sequenze Seq.I e Seq.II

1. Confrontare le sequenze (partendo dal I nucleotide di ogni sequenza)
2. Assegnare un punteggio all'allineamento sulla base di criteri fissati per la similarità
3. Ripetere l'operazione, "facendo scorrere" in tutti i modi possibili una sequenza rispetto all'altra

Output: allineameno con il punteggio massimo



Complessità dell’algoritmo elementare di allineamento

- L’ algoritmo fa scorrere ciclicamente una sequenza sull'altra, spostandosi ad ogni ciclo di una posizione e verificando quante posizioni hanno un'identità.
- Per ogni ciclo si devono verificare tutte le posizioni, quindi alla fine dovremo fare un numero di verifiche pari al numero di cicli per numero di posizioni: *più o meno dell'ordine del prodotto delle lunghezze delle due sequenze*.
- Se la lunghezza dele due sequenze e’ di n nucleotidi, si dovranno quindi effettuare n^2 verifiche (operazioni elementari): *la complessità dell’algoritmo è quindi quadratica* rispetto alla lunghezza delle sequenze di ingresso.

Ogni algoritmo e’ caratterizzato da una complessità temporale e spaziale rispetto alle dimensioni dei dati di ingresso.

Linguaggi di programmazione: strumenti per comunicare ad una macchina come risolvere un problema.

- Strumenti per la comunicazione uomo-macchina
- Permettono di esprimere e rappresentare *programmi = algoritmi + strutture dati* comprensibili ed eseguibili da una macchina

Caratteristiche dei linguaggi di programmazione

- Sono analoghi ai linguaggi naturali, con la differenza che vengono usati per comunicare con una macchina

Come i linguaggi naturali sono caratterizzati dalle seguenti componenti:

• Insieme di simboli (*alfabeto*) e di parole (*dizionario*) che possono essere usati per formare le frasi del linguaggio

• Insieme delle regole grammaticali (*sintassi*) per definire le frasi corrette composte dalle parole del linguaggio

• Significato (*semantica*) delle frasi del linguaggio

• Per utilizzare correttamente un linguaggio è necessario conoscerne la *pragmatica* (ad es: quali frasi è opportuno usare a seconda del contesto).

• *I linguaggi di programmazione, a differenza di linguaggi naturali, non devono essere ambigui e devono essere formalizzati* (definiti in maniera non equivocabile).

Linguaggi macchina

- Linguaggi immediatamente comprensibili per una macchina:
  - Istruzioni e dati sono sequenze di numeri binari
  - Le istruzioni operano direttamente sull' hardware (registri, locazioni di memoria, unità fisiche di I/O del calcolatore)
  - Sono specifici per un determinato processore o famiglia di processori
  - Assumono il modello computazionale di Von Neumann

Esempio:

Calcolo della somma S di due numeri A e B

Linguaggio macchina	Linguaggio assembly
00000010101111001010	LOAD A
00000010111111001000	ADD B
00000011001110101000	STORE S

Un *linguaggio assembly* è la forma simbolica di un linguaggio macchina: si usano nomi al posto dei codici binari per le operazioni e locazioni di memoria delle macchine.

Linguaggi a basso ed alto livello

- Linguaggi assembly e macchina sono *linguaggi a basso livello*
- *I linguaggi ad alto livello* permettono di scrivere programmi con un linguaggio piu' vicino a quello naturale

Esempio:

"Stampa sullo schermo la somma fra C ed il prodotto di A e B":

Linguaggio ad alto livello (C++):

```
cout << A * B + C;
```

Linguaggio Assembly:

```
mov eax,A
mul B
add eax,C
call WriteInt
```

Linguaggio macchina:

```
A1     00000000
F7 25 00000004
03 05 00000008
E8 00500000
```

Esempio: funzione per il calcolo della media in C ed in linguaggio assembly

Linguaggio C (alto livello)

```
double mean (double* x, unsigned n)
{
  double m = 0;
  int i;
  for (i=0; i<n; i++)
     m += x[i];
  m /= n;
  return m;
}
```

Linguaggio Assembly

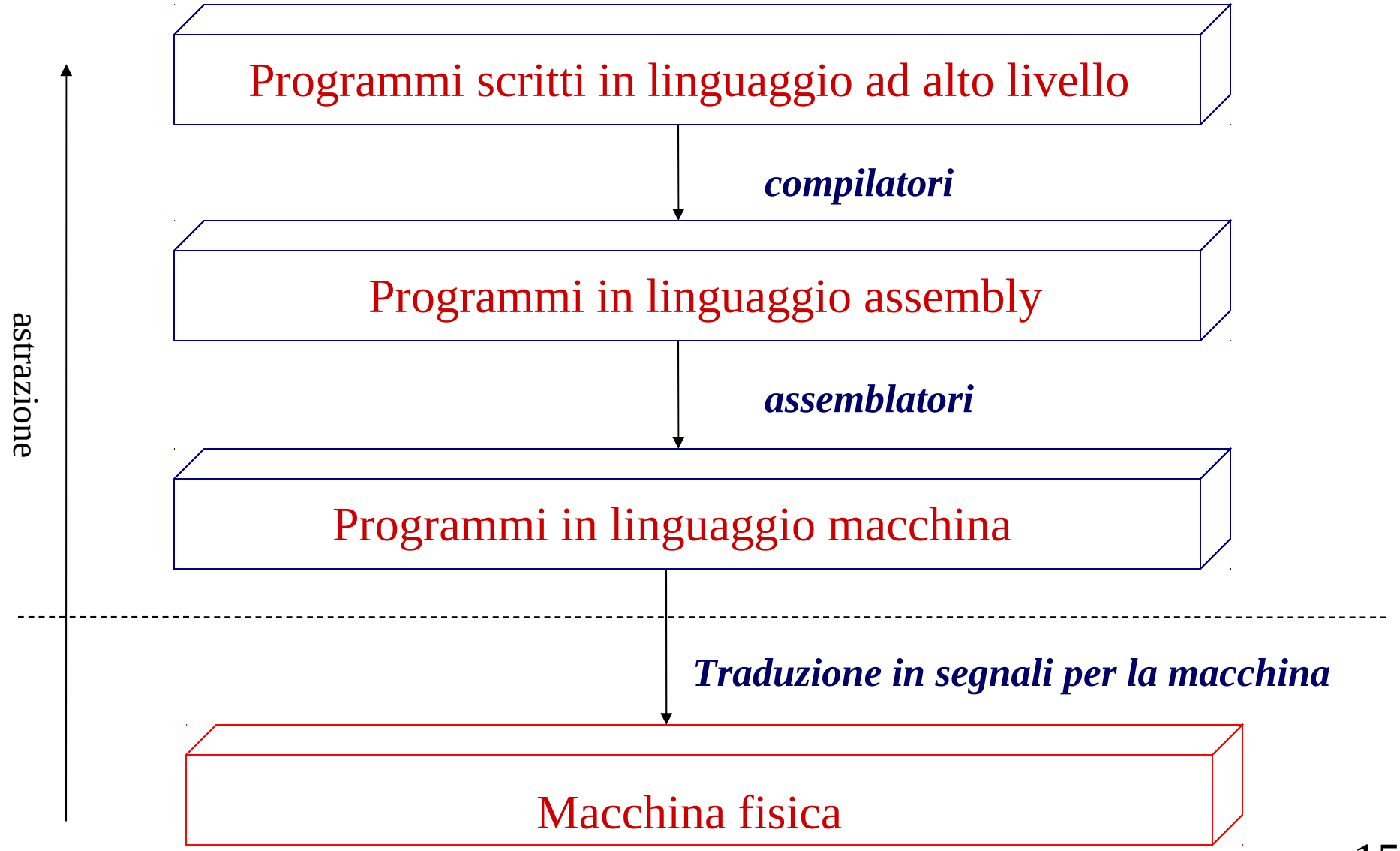
```
            .file       "qq.c"
            .text
.globl mean
            .type       mean,@function
mean:
            pushl       %ebp
            movl        %esp, %ebp
            subl        $24, %esp
            movl        $0, -8(%ebp)
            movl        $0, -4(%ebp)
            movl        $0, -12(%ebp)
.L2:
            movl        -12(%ebp), %eax
            cmpl        12(%ebp), %eax
            jb          .L5
            jmp         .L3
.L5:
            movl  -12(%ebp), %eax
            leal  0(,%eax,8), %edx
            movl  8(%ebp), %eax
            fldl  -8(%ebp)
            faddl (%eax,%edx)
            fstpl -8(%ebp)
            leal  -12(%ebp), %eax
            incl  (%eax)
            jmp   .L2
.L3:
            movl  12(%ebp), %eax
            movl  $0, %edx
            pushl %edx
            pushl %eax
            fildll      (%esp)
            leal  8(%esp), %esp
            fldl  -8(%ebp)
            fdivp %st, %st(1)
            fstpl -8(%ebp)
            movl  -8(%ebp), %eax
            movl  -4(%ebp), %edx
            movl  %eax, -24(%ebp)
            movl  %edx, -20(%ebp)
            fldl  -24(%ebp)
            leave
            ret
.Lfe1:
            .size mean,.Lfe1-mean
            .ident      "GCC: (GNU) 3.2.2
```

Linguaggi ad alto livello

- Sono definiti *astraendo* rispetto alla macchina fisica
- Realizzano una *macchina virtuale* soprastante alla macchina fisica e visibile al programmatore
- Richiedono di essere implementati su un particolare sistema di calcolo tramite strumenti opportuni (*compilatori* o *interpreti*)

Esempi: *fortran*, *C*, *lisp*, *java*, *R*

Livelli di rappresentazione e macchine astratte



Linguaggi compilati e linguaggi interpretati

- I *programmi compilati* vengono tradotti completamente dalla prima all' ultima istruzione nel linguaggio macchina del sistema sottostante (netta distinzione fra compile-time e run-time).
  Es: programmi in *C*, *fortran*, *C++*
- I *programmi interpretati* vengono tradotti ed eseguiti immediatamente riga per riga (l' interprete simula una macchina astratta, no distinzione netta fra compile-time e run-time).
  Es: programmi in R
- Esistono casi "intermedi": es: *java*.
- I *compilatori* e gli *interpreti* sono i programmi che effettuano la traduzione dal linguaggio ad alto livello al linguaggio macchina

Linguaggi di programmazione e produzione del software

Modello tradizionale “a cascata” per la produzione del sw:

- Analisi e specificazione dei requisiti
- Progetto (design) del sistema
- Implementazione: Produzione del codice nel linguaggio prescelto
- Verifica e validazione
- Manutenzione

In realtà il processo di produzione è ciclico.

Linguaggi di programmazione e ambienti di sviluppo

Ogni fase dello sviluppo del sw può essere supportato da *ambienti di sviluppo*.

Ambienti di sviluppo per l'implementazione del sw:

- Text editor
- Compilatori
- Linker
- Librerie
- Debugger
- ...

Linguaggi di programmazione per la bioinformatica

- In linea di principio qualsiasi linguaggio ad alto livello può essere utilizzato.
- Esistono comunque *linguaggi con librerie e package specifici* specializzati per la bioinformatica:

Progetti Open Source

  - *Perl* e *BioPerl*: http://bioperl.org
  - *Python* e *Biopython*: http://biopython.org
  - *Java* e *BioJava*: http://biojava.org
  - *R* e *Bioconductor*: http://www.bioconductor.org

  - Matlab e toolbox per la bioinformatica