Algoritmi e programmazione

Andrea Passerini
passerini@disi.unitn.it

Informatica

Elaboratore

Un elaboratore o *computer* è una macchina *digitale*, *elettronica*, *automatica* capace di effettuare trasformazioni o elaborazioni sui dati

digitale
: l'informazione è rappresentata in forma numerica discreta

elettronica
: la logica di manipolazione e la memorizzazione sono implementate con tecnologie di tipo elettronico (piuttosto che di tipo meccanico)

automatica
: è in grado di eseguire una successione di operazioni in modo autonomo (cioè senza intervento di un operatore umano)

Programmi

- le operazioni sono descritte sotto forma di un *programma*
- il programma e i dati su cui deve operare sono registrati in un dispositivo di *memoria*
- un dispositivo detto *unità di controllo* legge il programma e lo esegue su i dati
- questo modo di operare è detto *architettura alla Von Neumann*

Macchina universale

- Un programma è una descrizione delle operazioni che devono essere eseguite per risolvere una certa classe di problemi (e.g. trovare il maggiore tra due numeri)
- Un programma eseguito su dei dati in ingresso risolve lo specifico problema definito dai dati, nella classe di problemi trattati dal programma stesso (e.g. il maggiore tra 3 e 7 è 7)
- Un programma deve essere in una forma che sia rappresentabile nella memoria dell'elaboratore e comprensibile dall'elaboratore stesso.
- L'elaboratore è una *macchina universale* poiché può essere usato per risolvere qualsiasi problema la cui soluzione può essere descritta sotto forma di programma.

Algoritmo

- Il termine algoritmo deriva dal nome di un matematico arabo al-Khuwarizmi (IX sec d.C.)
- Definizione: un algoritmo è una successione ordinata di istruzioni (o passi) che definiscono le operazioni da eseguire su dei dati per risolvere una classe di problemi
- Un programma è la descrizione di un algoritmo in una forma comprensibile (ed eseguibile) dall'elaboratore

Proprietà degli algoritmi

Esistono dei precisi requisiti che devono essere soddisfatti affinchè un determinato elenco di istruzioni possa essere considerato un algoritmo:

Finitezza Ogni algoritmo deve essere finito, ossia ogni istruzione deve essere eseguita in un intervallo finito di tempo ed un numero finito di volte.

Generalità Ogni algoritmo deve fornire soluzione per tutti i problemi appartenenti ad una data classe, ed essere applicabile a tutti i dati apparentenenti al suo *insieme di definizione* o *dominio* producendo risultati che appartengono al suo *insieme di arrivo* o *codominio*.

Non ambiguità Devono essere definiti in modo univoco e non ambiguo i passi successivi da eseguire per ottenere i risultati voluti, evitando paradossi e contraddizioni.

Descrizione degli algoritmi

- Il linguaggio naturale non è adatto a descrivere gli algoritmi in quanto *ambiguo* e *ridondante*.
- Le proposizioni contenute in un algoritmo sono costituite da due componenti fondamentali:

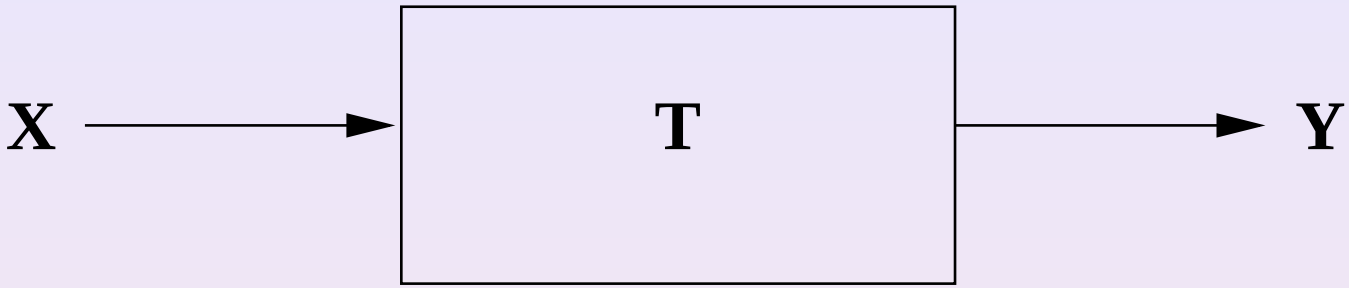
  istruzioni la descrizione delle operazioni da eseguire

  dati la descrizione degli oggetti su cui eseguire le operazioni

Linguaggi di programmazione

- Il calcolatore è in grado di comprendere solo istruzioni in linguaggio macchina.
- Il linguaggio macchina non è adatto alla scrittura di programmi poiché troppo distante dal linguaggio naturale.
- I linguaggi di programmazione permettono di scrivere programmi più agevolmente mantenendo le caratteristiche di non ambiguità e non ridondanza necessarie.
- Per poter essere eseguito dal calcolatore, un programma scritto con un linguaggio di programmazione deve essere *tradotto* in linguaggio macchina.

Traduzione



- Dati:
  - una sequenza di istruzioni *X* descritte in un linguaggio *sorgente* (e.g. C).
  - un traduttore *T* in linguaggio macchina per una certa CPU *C*
- L'esecuzione di *T* con ingresso *X* produce in uscita una sequenza di istruzioni *Y* descritte in linguaggio *eseguibile* da *C*.

Traduzione

Esistono due possibili approcci alla traduzione

Compilazione

- Il programma sorgente viene compilato nel suo rispettivo eseguibile una volta per tutte.
- La versione eseguibile viene utilizzata ogni volta si debba eseguire il programma.
- Esempi di linguaggi compilati: Pascal, C

Esempio: C

- Compilazione del file `max.c` nell'eseguibile `max`:

```
gcc -o max max.c
```

- Esecuzione dell'eseguibile `max`

```
./max
```

Traduzione

Interpretazione

- La traduzione in instruzioni in linguaggio macchina avviene solo al momento dell'esecuzione del programma.
- Il programma non viene eseguito direttamente dalla CPU ma tramite un *interprete* che traduce e manda in esecuzione le istruzioni del programma una per una.
- Esempi di linguaggi interpretati: Perl, PHP, Python, bash

Traduzione

Esempio: Python

- Esecuzione del programma `max.py` tramite l'interprete `python`:

```
python max.py
```

- Esecuzione dell'interprete `python` in modalità interattiva:

```
> python
Python 3.6.1 |Anaconda 4.4.0 (x86_64)| ...
[GCC 4.2.1 Compatible Apple LLVM 6.0 ...
Type "help", "copyright", "credits" ...
>>> print("Hello, world")
Hello, world
>>>
```

Compilatori vs Interpreti

tempi di esecuzione L'interprete deve tradurre al volo ogni singola istruzione (impiegando tempo) e non può ottimizzare il codice

qualità della traduzione il compilatore, eseguendo la traduzione al di fuori dell'esecuzione del programma, ha tutto il tempo di ottimizzare la traduzione realizzando codice più efficiente

portabilità del codice un programma eseguibile non può funzionare su una CPU diversa da quella per cui è stato compilato

Compilatori vs Interpreti

modificabilità del codice un programma eseguibile non può essere modificato, e recuperare il sorgente a partire dall'eseguibile è estremamente difficile (*reverse code engineering*)

scrittura dei programmi la scrittura di software tipicamente comporta modifiche successive al codice sorgente via via che viene scritto. In un linguaggio compilato qualunque modifica al programma comporta la ricompilazione del tutto.

interattività nello sviluppo un interprete può eseguire codice in maniera interattiva, facilitando notevolmente la programmazione.

Compilatore + *Virtual Machine*

- è una soluzione ibrida tra compilazione ed intepretazione
- Il codice sorgente viene compilato in un linguaggio intermedio (*bytecode*) (una volta per tutte)
- il bytecode viene interpretato da una *virtual machine* (VM) ossia un interprete di basso livello che emula una CPU.
- vantaggi
  - Le ottimizzazioni più onerose vengono fatte durante la prima compilazione
  - Il bytecode è non modificabile
  - Lo stesso bytecode può essere eseguito su tutte le CPU per cui esiste un'implementazione della VM
  - E' più facile scrivere VM per CPU diverse che compilatori
- svantaggi
  - Il bytecode deve comunque essere convertito in codice macchina a runtime $\rightarrow$ più lento di eseguire direttamente codice macchina