Algoritmo

Formalmente, per algoritmo si intende una successione finita di passi o istruzioni che definiscono le operazioni da eseguire su dei dati (=istanza del problema): in generale un algoritmo è definito per risolvere ogni istanza di un problema di un certo tipo.

La parola algoritmo deriva dal nome del matematico persiano Muhammad ibn Mūsa 'l-Khwārizmī ritenuto uno dei primi a riferirsi al concetto di algoritmo

Generalmente un algoritmo è sempre definito supponendo che esso interagisca con un ambiente esterno dal quale acquisisce dati e verso il quale comunica dati o messaggi.

I dati su cui opera un'istruzione sono forniti all'algoritmo dall'esterno oppure sono il risultato di istruzioni eseguite precedentemente.

Proprietà dell'algoritmo

Finitezza: ogni istruzione va eseguita in un tempo finito e deve essere eseguita un numero finito di volte;

Generalità: un algoritmo deve fornire soluzione per tutti i problemi di una classe;

Non ambiguità: i passi devono essere univoci, evitare paradossi, contraddizioni e ambiguità.

Inoltre un algoritmo deve essere **corretto** ed **efficiente**, ossia arrivare alla soluzione giusta e nel modo più veloce possibile, usando la minore quantità di memoria possibile.

Un semplice esempio: la somma dei primi n numeri naturali.

0. inizio;
1. leggi n;
2. poni i = 0 e s = 0;
3. se i > n
    3.1 stampa s (che è il risultato);
    3.2 vai al passo 7;
4. aumenta s di i (s := s+i);
5. incrementa i di 1 (i := i+1);
6. torna al passo 3.;
7. fine.

Finitezza: **ok**. Però se il passo 5 o il passo 3.2 fossero omessi, l'algoritmo non terminerebbe.

Generalità: **ok**. La quantità n è generica e a meno che non sia grande tanto da non poter essere contenuto in memoria, l'algoritmo funzione per ogni n.

Non ambiguità: **ok.** Però se togliessi il passo 2, non sarebbero definiti s e i al passo 3.

Correttezza: cosa succede se al passo 3 ci fosse ≥ invece di >? L'algoritmo non sarebbe corretto. Come fare per correggerlo mantenendo il ≥?

Provare a riscrivere l'algoritmo utilizzando al passo 3 il controllo con il < e il ≤.

Un altro esempio: il Massino Comune Divisore (MCD) fra due numeri

Si ricorda il procedimento di Eulero:

MCD (a,b) = MCD (b,r) se r ≠ 0;
MCD (a,b) = b se r = 0,

dove a,b sono numeri naturali con a > b e r = a mod b (resto della divisione di a con b).

Esempi:
MCD (105,90) = MCD (90,15) = MCD (15,0) = 15;
MCD (27,10) = MCD (10,7) = MCD (7,3) = MCD (3,1) = MCD (1,0) = 1.

Algoritmo:

1. inizio;
2. acquisisci i valori di a e b;
3. se b > a, scambia i valori di a e b;
4. calcola r (resto della divisione di a con b);
5. se r = 0, salta al passo 8;
6. sostituisci a con b e b con r;
7. torna al passo 4;
8. scrivi il risultato che è b;
9. fine.

Finitezza: **ok**. L'unico problema potrebbe verificarsi al passo 7 dove si dice di tornare al passo 4. Ma notare che r sicuramente assumerà il valore 0.

Non ambiguità: **ok.**

Generalità: **ok.** Notare che l'algoritmo funziona anche se a = b.

In genere negli algoritmi le istruzioni sono semplici:

- lettura e scrittura (acquisizione e stampa);
- operative (operazioni aritmetiche o di assegnamento);
- di controllo;
- di salto.

Dagli esempi risolta inoltre che lo stesso concetto può essere espresso in più modi:

- leggi, acqisisci, ...;
- stampa, scrivi, ...;
- vai a, salta a, ...;
- poni, assegna, ...;
- incrementa, somma, aggiungi, aumenta, ...

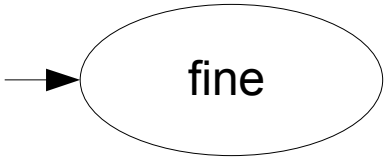
Si rende allora necessario uno pseudolinguaggio il più possibile uniforme per la scrittura degli algoritmi. Notare che non si parla di uno specifico linguaggio di programmazione.

Una possibile stesura degli algoritmi può essere realizzata tramite il *diagramma a blocchi* (o *diagramma di flusso* o *flow chart*).

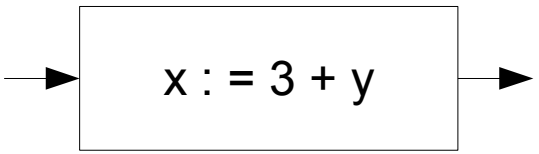
Nel diagramma le varie istruzioni dell'algoritmo sono rappresentate con simboli grafici, detti *blocchi elementari,* opportunamente collegati. Essi sono:

blocco di inizio

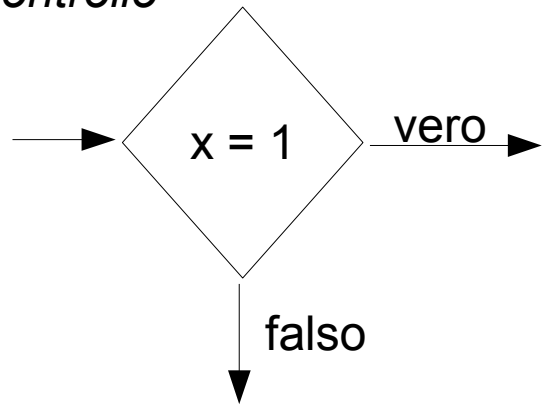
blocco di fine



blocco di elaborazione o azione



blocco di controllo



blocco di input/output

Proprietà e osservazioni

In un diagramma a blocchi sono presenti:

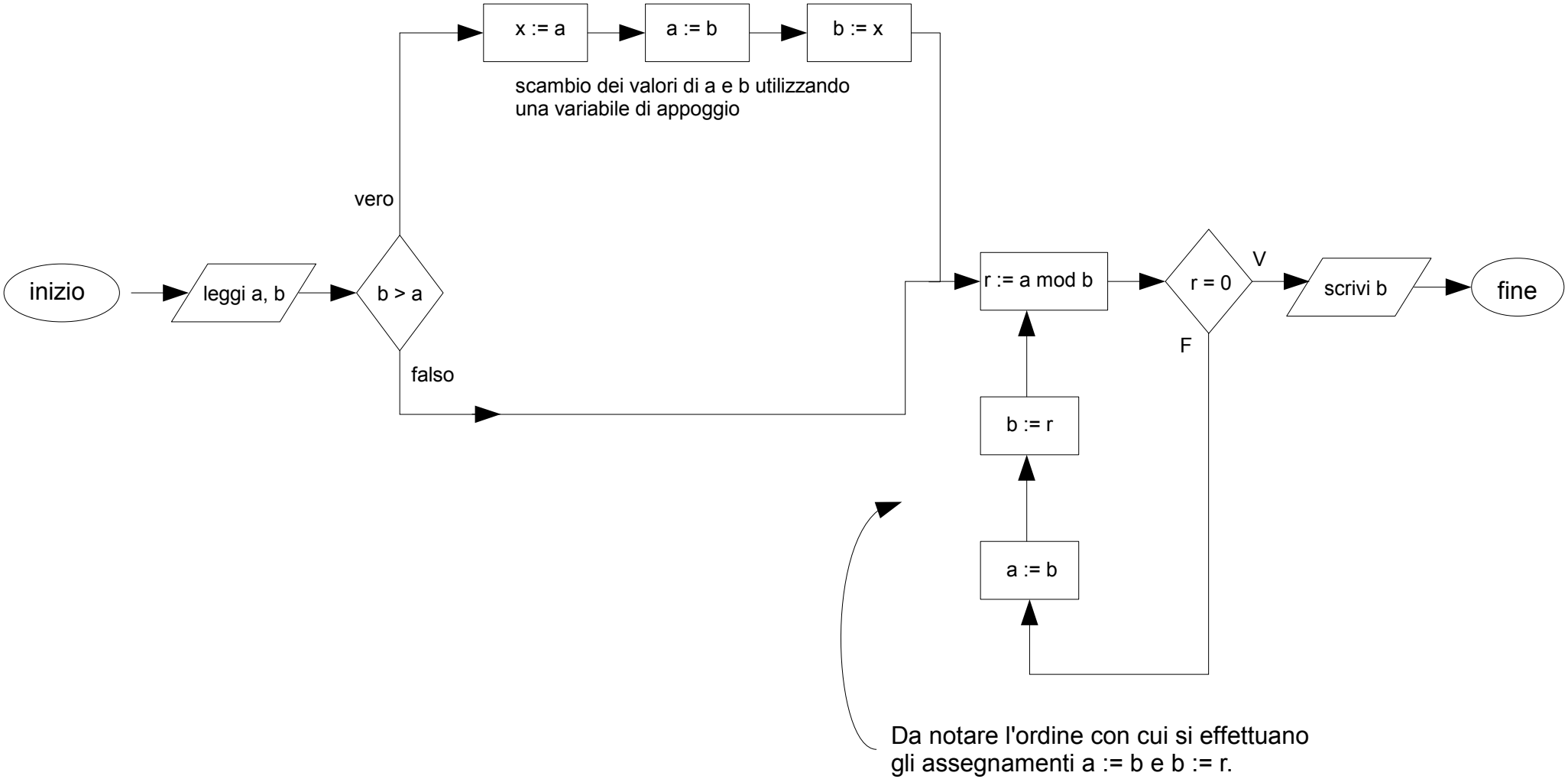
• un blocco di inizio e uno di fine;
• un numero finito di blocchi di elaborazione (o azione) e di input/output;
• un numero finito di blocchi di controllo.

Inoltre:

- ogni blocco di elaborazione (o azione) o di input/output ha una freccia in ingresso e una in uscita;
- ogni blocco di controllo ha una freccia in ingresso e due in uscita;
- ogni freccia entra in un blocco e si raccorda con un'altra freccia (escluso la freccia in uscita del blocco di inizio e quella in entrata del blocco di fine);
- ogni blocco è raggiungibile dal blocco di inizio;
- il blocco di fine è raggiungibile da ogni altro blocco.

Il blocco di controllo contiene una *condizione* solitamente un'espressione che può essere vera o falsa.

Diagramma di flusso dell'algoritmo per l'MCD



Nello pseudolinguaggio si usano spesso dei costrutti fondamentali.

- *if* Si usa nella forma:

  if B then C1 else C2,

  dove B è un'espressione booleana, C1 e C2 cono comandi. Se B è vera allora si esegue C1, altrimenti C2.

  Si può trovare anche senza else:

  if B then C1.

  In questo caso se B è falsa non si esegue C1 ma si passa al comando successivo all'if.

  Il costrutto if è detto *comando condizionale* (o di *selezione*)

- *iterazione indeterminata*

  - ciclo *while*

    while B do C

    1. valutare l'espressione booleana B;
    2. se B è vera eseguire il comando C e tornare al passo 1,
       altrimenti il ciclo termina.

  - ciclo *until repeat*

    repeat C until B

    1. eseguire il comando C;
    2. valutare B: se B è falsa torna al passo 1, altrimenti il ciclo termina.

Notare che
- nel ciclo *until repeat* il comando C viene eseguito almeno una volta;
- il ciclo *unitl repeat* equivale a:

```
esegui C;
while (not B) do C;
```

(not B è la negazione di B)

- *iterazione determinata*

  - ciclo *for* Generalmente è usato nella forma:

    for I = **inizio** to **fine** by **passo** do

    C

    I è l'indice o contatore o variabile di controllo, **inizio** e **fine** sono espressioni di tipo intero, **passo** è una costante.

    1. inizializzare I con il valore di **inizio** (I:=**inizio**);
    2. se I > **fine**, il ciclo termina;
    3. eseguire il comando C e incrementare I del valore di **passo** (I:=I+**passo**);
    4. tornare al punto 3.

Notare che:
- è sconsigliato modificare la variabile di controllo nel corpo del ciclo (comando C);
- usato in questa forma il ciclo *for* ha sempre termine, a differenza dell'iterazione indeterminata.

I cicli while, repeat-until e for sono *comandi iterativi.*
Il ciclo for è un'iterazione *determinata* poiché conoscendo il valore di **inizio**, **fine** e **passo** (che sono praticamente numeri) è possibile sapere quante volte sarà eseguito C. Ciò non è possibile nei cicli while e repeat-until, che sono iterazioni *indeterminate*.