

Algoritmi

Introduzione.

Un computer non è altro che un elaboratore in grado di eseguire un determinato set di istruzioni. Per poter svolgere qualsiasi compito il calcolatore ha bisogno di qualcosa che gli dica passo passo cosa deve fare, o meglio, quale delle istruzioni che conosce deve eseguire in un determinato momento.

Un software non è altro che un compito da svolgere scritto utilizzando il set di istruzioni primitive che il calcolatore è in grado di eseguire. Quindi una volta dato un problema è necessario “tradurlo” in termini di istruzioni per poterlo fare risolvere da un elaboratore. Descrivere la soluzione di un problema o lo svolgimento di un compito sotto forma di passi discreti, eseguibili e non ambigui significa costruire un algoritmo

Algoritmo.

Algoritmo: descrizione finita e non ambigua del procedimento finito per la soluzione di una classe di problemi.

Le proprietà fondamentali di un algoritmo sono: *non-ambiguità* (il procedimento deve essere interpretabile in modo univoco da chi lo deve eseguire), *eseguibilità* (ogni istruzione dell'algoritmo deve poter essere eseguita senza ambiguità da parte di un esecutore reale o ideale), *finitezza* (sia il numero di istruzioni che compongono l'algoritmo, che il tempo di esecuzione devono essere finiti).

Es. Il seguente algoritmo risolve il problema di un contadino che deve far attraversare il fiume ad una capra, un cavolo e un lupo, avendo a disposizione una barca in cui può trasportare solo uno dei tre alla volta. Se incustoditi, la capra mangerebbe il cavolo e il lupo mangerebbe la capra.

Algoritmo che descrive la soluzione:

1. Inizio
2. Porta la capra sull'altra sponda
3. Porta il cavolo sull'altra sponda
4. Riporta la capra sulla sponda di partenza
5. Porta il lupo sull'altra sponda
6. Porta la capra sull'altra sponda
7. Fine

Algoritmo base dei calcolatori.

Ogni calcolatore funziona secondo un algoritmo di base secondo della forma:

1. finché non trovi un'istruzione di *halt* fai:
2. preleva un'istruzione dalla memoria
3. decodifica l'istruzione
4. esegui l'istruzione

Questo algoritmo è intrinseco nella struttura della CPU e fa in modo che una volta inserito un programma nella memoria, questo possa essere letto istruzione per istruzione ed eseguito. Tale caratteristica sta alla base della programmabilità degli operatori e premette la totale generalità degli elaboratori.

Rappresentazione di un algoritmo.

Al fine di rappresentare gli algoritmi in modo comprensibile per il programmatore, vengono principalmente utilizzate due tecniche:

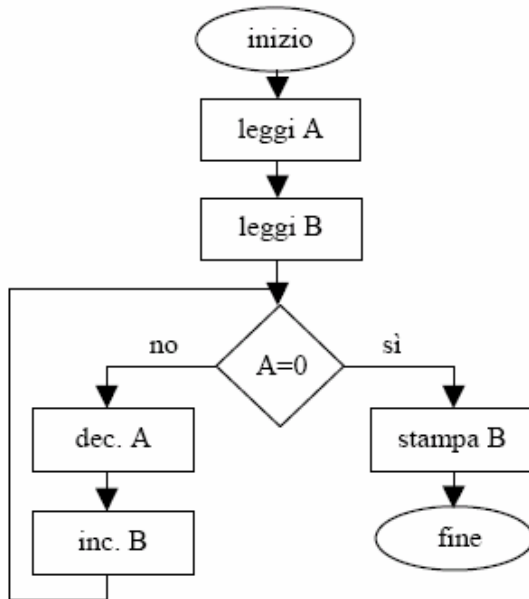
1. rappresentazione mediante *pseudolinguaggio o pseudocodice*
2. rappresentazione mediante *diagramma di flusso (flowchart)*

Pseudolinguaggio:: linguaggio informale (ma non ambiguo) per la descrizione delle istruzioni. Ogni riga rappresenta un'istruzione. Se non diversamente specificato, le righe si leggono dall'alto verso il basso.

Diagramma di flusso (flowchart): rappresentazione grafica in cui ogni istruzione è descritta all'interno di un riquadro e l'ordine di esecuzione delle istruzioni è indicato da frecce di flusso tra i riquadri.

Es: somma di due numeri naturali utilizzando solo incrementi e decrementi

Diagramma di flusso



Pseudolinguaggio

- 1. Leggi A
- 2. Leggi B
- 3. Se A=0 vai all'istruzione 8
- 4. Decrementa A
- 5. Incrementa B
- 6. Vai all'istruzione 4
- 7. Stampa B

Definizione di uno pseudolinguaggio.

Uno pseudolinguaggio si basa sul concetto di azione "primitiva". L'utilizzo di azioni primitive per descrivere un procedimento fa sì che questo possa essere compreso senza alcuna ambiguità. In altre parole, nel caso in cui si descriva una azione in modo superficiale o si utilizzino vocaboli dal significato non univoco una frase potrebbe essere interpretata in diversi modi. Si prenda, per esempio, la frase "andare a lezione può essere irritante". Un lettore che non conosca gli argomenti della lezione, il docente o il luogo in cui la lezione si svolge potrebbe interpretare la frase come: "La lezione causa irritazione" oppure "il tragitto per arrivare dove si svolge la lezione è irritante" oppure in entrambi i modi.

Per evitare un'interpretazione ambigua, quindi, viene definito un set di attività primitive, note agli interlocutori (uomo – macchina) e un insieme di regole per utilizzare le primitive. Tutto questo, set di primitive più insieme di regole, costituisce un normale *linguaggio di programmazione*.

Ogni pseudolinguaggio deve necessariamente definire un insieme minimo di elementi, quali:

- 1. Istruzioni di inizio e di fine, start, end
Sono i punti d'ingresso e di uscita del flusso di esecuzione
- 2. Istruzioni di assegnamento nome ← espressione
L'esecuzione di un'istruzione di assegnamento avviene in due fasi: la valutazione dell'espressione (utilizzando i valori correnti delle eventuali variabili che in essa compaiono) e l'aggiornamento del valore della variabile a sinistra del segno ←

3. Scelta tra due possibili attività `if(condizione) then (attività 1)`
`else(attività 2)`

Biforcazione del flusso in due percorsi alternativi (mutuamente esclusivi) la cui esecuzione è condizionata al verificarsi di una condizione. L'esecuzione di un'istruzione condizionale comporta innanzitutto la valutazione della condizione, e quindi l'esecuzione delle istruzioni che compongono uno dei due cammini.

4. Strutture iterative `while(condizione) do (azione) end while`

Esecuzione ripetuta di una o più istruzioni. La ripetizione dipende dall'esito di un controllo. È fondamentale che l'esito del controllo dipenda da variabili il cui valore può essere modificato dall'esecuzione delle istruzioni del ciclo. In caso contrario il ciclo verrebbe eseguito o mai (risultando inutile) o all'infinito (violando la definizione di algoritmo). Le variabili da cui dipende il controllo sono dette variabili di controllo del ciclo. Ogni ciclo è composto da 4 elementi: l'inizializzazione delle variabili di controllo, il controllo della condizione da cui dipende l'iterazione, il corpo del ciclo (sequenza delle istruzioni da eseguire ciclicamente) e l'aggiornamento delle variabili di controllo.

5. Istruzione di salto `goto(riga destinazione)`

L'istruzione di salto sposta il flusso di esecuzione in un particolare punto dell'algoritmo. La riga di destinazione rappresenta, appunto, il punto in cui l'esecuzione riprenderà dopo il goto.

6. Procedure `procedure NomeProcedura (`
`Istruzione 1`
`Istruzione 2`
`Istruzione 3`
`)`

Unità di programma utilizzabili attraverso invocazione da un qualsiasi punto del codice. Tutti i linguaggi di programmazione utilizzano questa strategia per rendere più leggibile il codice. Sinonimi di procedura sono: funzione, metodo, subroutine, sottoprogramma ecc. Esempio di utilizzo:

```

procedure SalutaTutti (
    contatore ← 24
    while(contatore > 0) do (
        Stampa("Saluti ")
        contatore ← contatore - 1
    )end while
)
1. Start
2. Stampa("Ora invoco la procedura")
3. SalutaTutti
4. end

```

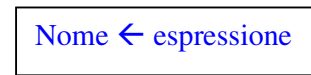
Diagramma di flusso.

Il flusso inizia e termina in blocchi fittizi (detti inizio e fine) di forma generalmente arrotondata. I riquadri del diagramma di flusso possono avere forme diverse (convenzionali) per rappresentare graficamente il tipo di istruzione che contengono. Noi utilizzeremo solo tre tipi di riquadri: rombi con una freccia entrante e due uscenti per indicare diramazioni di flusso condizionate, rettangoli per indicare qualsiasi istruzione che non sia una condizione, ellissi per indicare inizio e fine.

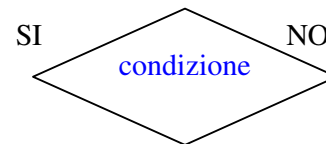
- 1. Inizio e fine



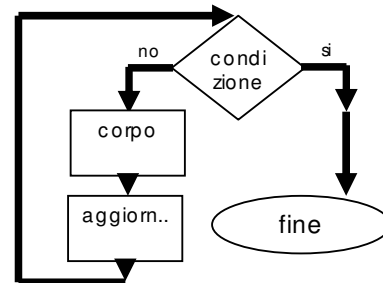
- 2. Assegnamento ed elaborazione



- 3. Scelta condizionata



- 4. Iterazione e salti



Esecuzione di un algoritmo.

Un algoritmo può essere visto come un risolutore di problemi che acquisisce dati d’ingresso e produce risultati. Generalmente, uno stesso algoritmo può essere applicato più volte per risolvere lo stesso tipo di problema con dati d’ingresso diversi, ovvero istanze diverse dello stesso problema. Chiamiamo esecuzione di un algoritmo la sua applicazione a determinati dati d’ingresso. Un algoritmo è detto lineare se l’esecuzione segue un flusso lineare dalla prima all’ultima istruzione, è detto non lineare altrimenti. (Es: L’algoritmo per la somma di due numeri naturali dell’esempio visto in precedenza è non lineare). Chiamiamo passo d’esecuzione l’esecuzione di un’istruzione. Il numero di passi d’esecuzione è generalmente diverso dal numero di istruzioni utilizzate per descrivere l’algoritmo e dipende dai dati d’ingresso.

Grandezze.

Costante: quantità nota a priori il cui valore non dipende dai dati d’ingresso e non cambia durante l’esecuzione dell’algoritmo (Es. lo 0 dell’istruzione 4 dell’algoritmo per la somma di due numeri naturali descritto precedentemente). *Variabile*: nome simbolico cui è assegnato un valore che può dipendere dai dati d’ingresso e cambiare durante l’esecuzione dell’algoritmo (Es: A e B nell’ dell’algoritmo per la somma di due numeri naturali descritto precedentemente). *Espressione*: sequenza di variabili, costanti o espressioni combinate tra loro mediante operatori (Es: $a+b*4$).

Nella valutazione di un'espressione si sostituisce ad ogni variabile il suo valore corrente e si seguono le operazioni secondo regole di precedenza prestabilite (o indicate da parentesi).

Rappresentazione dell'esecuzione di un algoritmo.

Traccia d'esecuzione: tabella le cui righe rappresentano i passi d'esecuzione e le cui colonne rappresentano (per ogni passo) l'istruzione eseguita e il valore delle variabili (prima e dopo l'esecuzione dell'istruzione).

passo n.	istr. n.	valore di A		valore di B	
		prec.	succ.	prec.	succ.
1	1	-	2	-	-
2	2	2	2	-	4
3	3	2	2	4	4
4	4	2	1	4	4
5	5	1	1	4	5
6	6	1	1	4	5
7	3	1	1	4	5
8	4	1	0	5	5
9	5	0	0	5	6
10	6	0	0	6	6
11	3	0	0	6	6
12	7	0	0	6	6

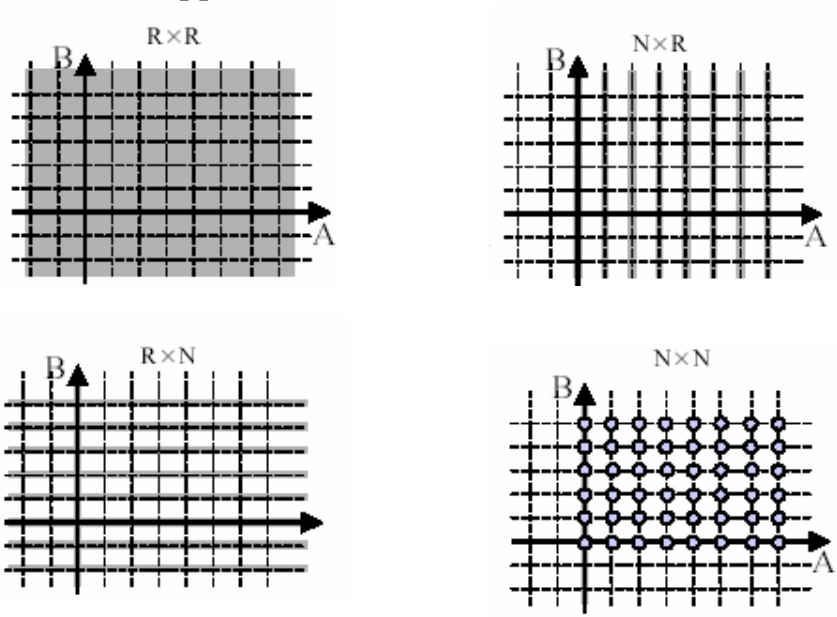
Es: Traccia d'esecuzione dell'algoritmo per la somma di due numeri naturali descritto precedentemente. (i valori iniziali sono rispettivamente $A = 2$ e $B = 4$)

Dominio di un algoritmo.

Il dominio di applicazione di un algoritmo è l'insieme dei valori dei dati d'ingresso ai quali può essere applicato. Il dominio di un algoritmo può non coincidere con quello del problema che intende risolvere. Un algoritmo che risolve alcune istanze di un problema (cioè un algoritmo che risolve il problema per alcuni valori dei dati d'ingresso) può produrre risultati errati (o non produrre risultati) per altre istanze.

Es: Il dominio di definizione della somma è $\mathbb{R} \times \mathbb{R}$, ma il dominio in cui l'algoritmo che utilizza solo incrementi e decrementi può essere utilizzato per il calcolo della somma di due numeri è $\mathbb{R} \times \mathbb{N}$ perché se il primo numero non è intero il risultato non è corretto, se il primo numero non è positivo l'algoritmo non termina.

Rappresentazione di domini numerici



Ref: J. Gleen Brookshear, "Informatica una panoramica generale"