La programmazione: gli algoritmi

- **Gli elaboratori sono macchine capaci di effettuare un numero limitato di operazioni molto semplici, ma a grande velocità.**

- **Combinando insieme sequenze di operazioni semplici è possibile realizzare funzioni comunque complesse, che possono soddisfare le esigenze più disparate.**

- **È possibile quindi realizzare programmi per l'elaborazione di testi, per la gestione di grosse banche dati, per il controllo di robot e di processi, e così via.**

- **Condizione fondamentale affinché si possa risolvere un problema mediante l'elaboratore è che esso sia stato definito con precisione.**

- **Non si può infatti dire alla macchina semplicemente:**

  “Calcola la criticità di un reattore”

  “Evadi gli ordini di un'azienda”

  “Aggiorna i conti correnti di una banca”

  “Traduci in lingua inglese”

- **Per le applicazioni di tipo scientifico, si dovranno definire i problemi in termini di sistemi di equazioni, di formule da calcolare che a loro volta dovranno essere poste in termini elementari (i soli comprensibili all'elaboratore).**
- **Per le elaborazioni aziendali si dovrà redigere un processo di ragionamento logico assai dettagliato, comparazioni di codici, letture di tabelle, confronti fra quantità, ecc.**

- **Quindi per risolvere un determinato problema occorre esaminarlo nei dettagli più minuti.**
- **La sequenza di passi che porta alla soluzione di un problema è detto *algoritmo*.**
- **La sequenza di operazioni, incorporate nella logica della macchina, utilizzate per lo svolgimento di una determinata linea risolutiva, si dice *programma*.**

- **L'esperienza quotidiana suggerisce numerosi esempi di algoritmi: una ricetta da cucina, le istruzioni per l'uso di un televisore, le indicazioni per un lavoro a maglia, le regole per calcolare la somma o la moltiplicazione di due numeri.**

- **Le operazioni che un algoritmo può prescrivere possono essere di natura assai differente, ma devono possedere, per poter essere considerate tali, delle caratteristiche ben precise.**

- **ogni operazione deve avere termine entro un intervallo di tempo finito dall'inizio della sua esecuzione.**

  “Cavalcare un centauro” o “calcolare le cifre decimali di π” non possono essere considerate operazioni: la prima, infatti, non è eseguibile affatto e la seconda non può avere termine in un tempo finito.

- ogni operazione deve produrre, se eseguita, un effetto osservabile che possa essere descritto.

  Ad esempio, l'esecuzione dell'operazione di “spostare il tavolo” produce un effetto che può essere descritto specificando la disposizione dei mobili nella stanza prima e dopo lo spostamento.

  Invece, “pensare” o “pensare al numero 5” non possono essere considerate operazioni poiché non è affatto chiaro come il loro effetto possa essere “osservato” e descritto.

- **ogni operazione deve produrre lo stesso effetto ogni volta che venga eseguita a partire dalle stesse condizioni iniziali (*determinismo*).**

  Così se X vale 5 e Y vale 10, tutte le volte che eseguiamo la somma di X e Y con quei valori iniziali, il risultato dovrà essere sempre 15.

- **L'esecuzione di un algoritmo da parte di un “esecutore” - uomo o macchina - si traduce in una successione di operazioni che vengono effettuate nel tempo evocando un “processo sequenziale”.**
- **Per “processo sequenziale” s’intende una serie di eventi che occorrono uno dopo l'altro, ognuno con un inizio e una fine bene identificabile.**

- **Talvolta il processo è fisso, cioè è sempre lo stesso ad ogni diversa esecuzione.**

  Esempio: algoritmo per calcolare l'importo di una fattura;

- **cerca l'aliquota IVA sulla tabella;**
- **moltiplica l'importo netto per l'aliquota trovata;**
- **somma il risultato all'importo netto.**

- **Questo algoritmo è composto da tre *istruzioni*, che devono essere eseguite in sequenza.**
- **L'elencazione, una dopo l'altra, di tutte le istruzioni eseguite, nell'ordine di esecuzione è detta *sequenza di esecuzione* dell'algoritmo.**
- **Più frequentemente lo stesso algoritmo può evocare più processi sequenziali differenti, a seconda delle condizioni iniziali.**

- **Esempio: il precedente algoritmo, se è richiesto do dover considerare la possibilità che la merce in esame non sia soggetta a IVA.**

- **SE la merce da fatturare è soggetta a IVA ALLORA**
  - **Cerca la corretta aliquota IVA sulla tabella e moltiplica l'importo per l'aliquota trovata.**
  - **Somma il risultato all'importo netto.**
- **ALTRIMENTI**
  - **Tieni conto solo dell'importo di partenza.**

- In questo caso, il processo evocato non è fisso, ma dipende dai dati da elaborare, in particolare dal tipo di merce da fatturare: l'algoritmo descrive un insieme costituito da due sequenze di esecuzione diverse.
- In altre situazioni, la stessa sequenza di operazioni può dover essere eseguita più volte. Ad esempio, consideriamo l'algoritmo per effettuare una telefonata:

1. Solleva il ricevitore
2. Componi il numero
3. SE qualcuno risponde ALLORA
   SALTA al PUNTO 6
4. Deponi il ricevitore
5. TORNA AL PUNTO 1
6. Conduci la conversazione

- **Negli algoritmi fino ad ora visti sono indicate due classi fondamentali di costrutti:**
  - **costrutti che prescrivono l'esecuzione di determinate operazioni (come “deponi il ricevitore”);**
  - **costrutti che indicano l'ordine di esecuzione delle operazioni (come “torna al punto 1”).**

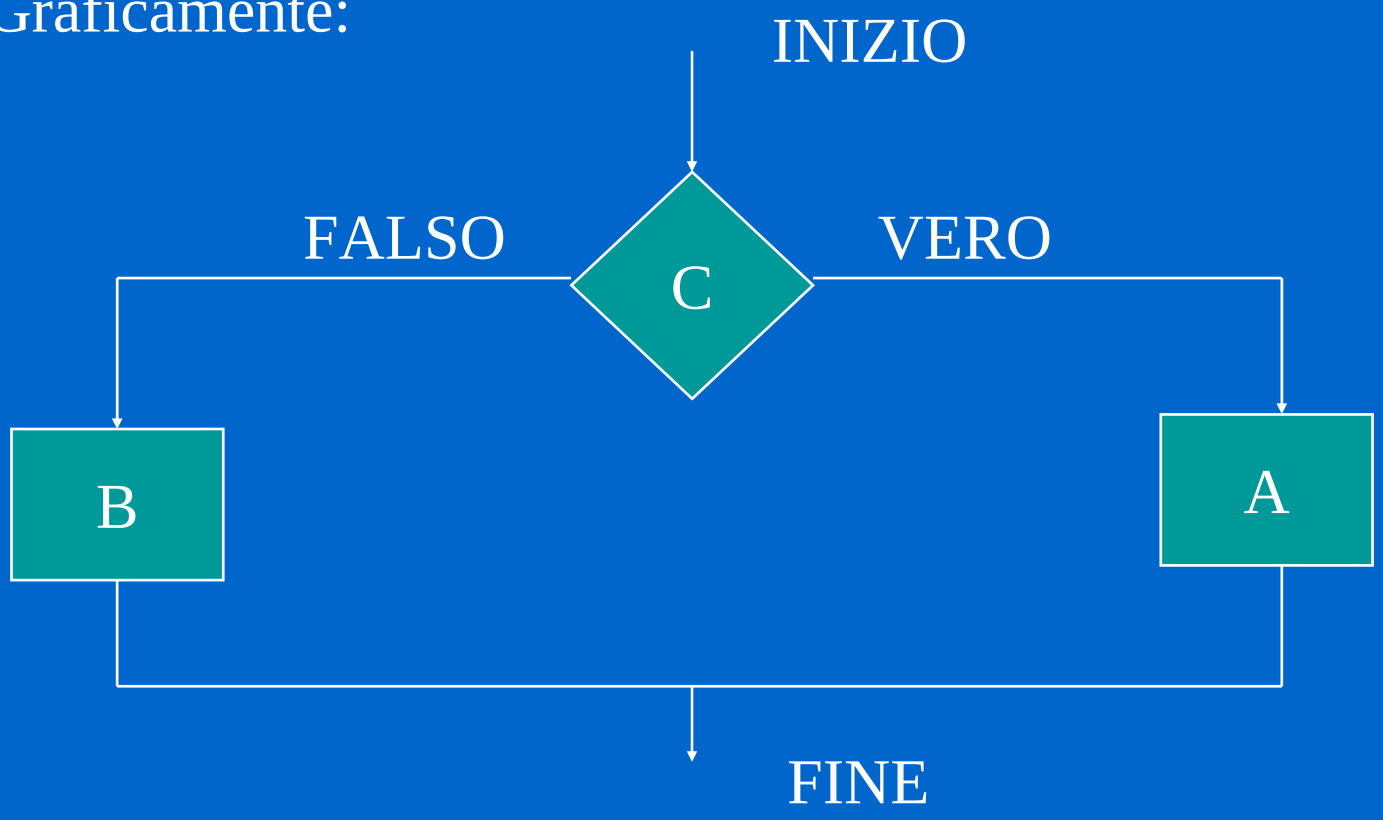
- **I primi sono detti *istruzioni* o *comandi*, i secondi *schemi di controllo* o *istruzioni di controllo*.**

- **È stato dimostrato (Bohm - Jacopini) che qualsiasi algoritmo può essere realizzato utilizzando due soli costrutti di controllo:**
  - **costrutto di *decisione binaria***
  - **costrutto di *ripetizione* (o *loop*)**

- **Il costrutto di *decisione binaria* è anche detto costrutto *IF - THEN - ELSE (SE - ALLORA - ALTRIMENTI)* e segue lo schema:**

```
IF condizione THEN
        sequenza A
ELSE
        sequenza B
```

Graficamente:



- **Nei casi in cui manca la parte ELSE il costrutto condizionale diventa semplicemente:**

  IF condizione THEN

  sequenza A

  SE la *condizione* è vera ALLORA esegui la *sequenza A* (ovvero "esegui la sequenza A se e solo se la condizione è vera").

- **Il meccanismo di *ripetizione*, detto anche meccanismo DO-WHILE, si può manifestare in due forme diverse.**

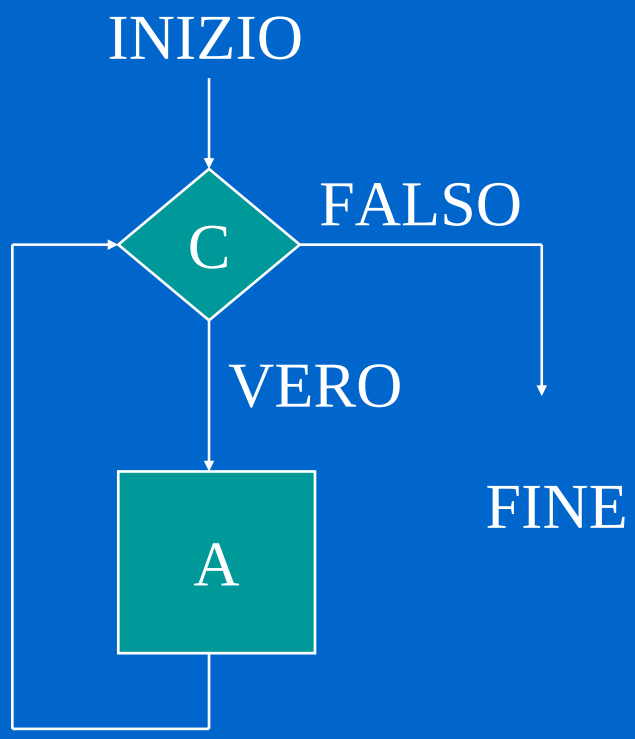
- **La prima è detta WHILE-DO (*FINCHÉ-ESEGUI*), la cui rappresentazione è:**

  WHILE condizione DO
  sequenza istruzioni

  Continua ad eseguire la sequenza di *istruzioni* finché la *condizione* è vera.

  Se già alla prima verifica la *condizione* risulta falsa, la sequenza di *istruzioni* non verrà mai eseguita.

Graficamente:

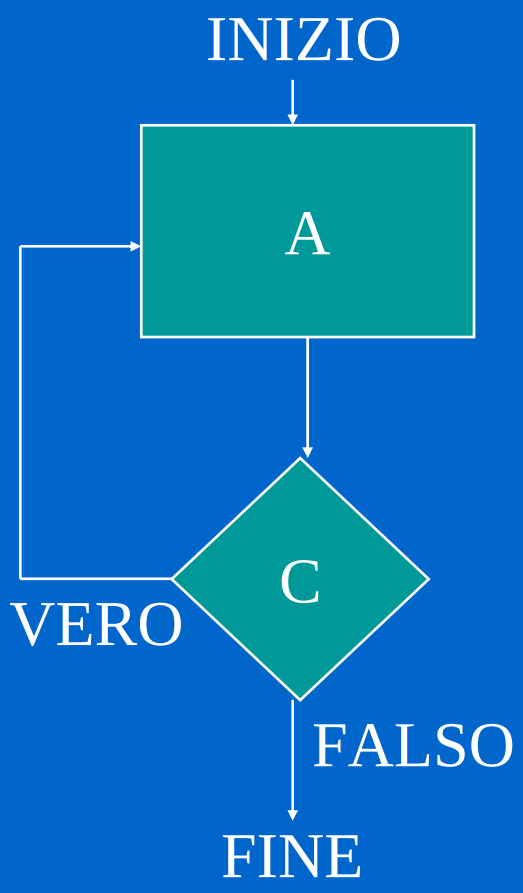


- **La seconda è detta *REPEAT-UNTIL (RIPETI-FINCHÉ*) ed è simmetrica alla precedente:**

  REPEAT
  sequenza istruzioni
  UNTIL condizione

- **Qui il blocco di istruzioni precede il controllo: pertanto verrà sempre eseguito *almeno una volta* anche se la *condizione* risulta falsa alla prima verifica.**

Graficamente:



- **Una derivazione interessante del primo dei due costrutti è il *ciclo a contatore*, che ha la forma:**

  FOR *variabile* FROM *valore_iniz* TO *valore_fin* DO
  sequenza istruzioni

  A *variabile* si attribuiscono tutti i valori compresi tra *valore_iniz* e *valore_fin* e, per ogni attribuzione, si esegue la sequenza di *istruzioni*.

  Se all'inizio *valore_iniz* è maggiore di *valore_fin*, le istruzioni non vengono mai eseguite.

- **Come specificare le azioni?**
- **Supponiamo di voler effettuare il prodotto di due numeri interi: un metodo potrebbe essere quello delle addizioni successive. L'algoritmo potrebbe essere:**

  Si sommi il moltiplicando a se stesso un numero di volte uguale al valore del moltiplicatore.

- **Così è troppo vago.**
- **Per eliminare incertezze e ambiguità, si ricorre a *simboli* o *identificatori* per riferirsi agli oggetti (di solito contenitori) manipolati.**
- **L’algoritmo per il prodotto diventa così:**

1 Si chiami M il valore del moltiplicando ed N il valore del moltiplicatore e sia M1 il risultato (inizialmente zero).

2 Si ripetano le seguenti operazioni fino a che il valore di N non diventi uguale a 0:

2.1 si sommi il valore del moltiplicando M al valore di M1 e si chiami il risultato ancora M1;

- 2.2 si sottragga 1 dal valore di N, e si chiami il risultato ancora N.

3 Alla fine il valore di M1 è il risultato: lo si visualizza.

- **I simboli M, N e M1 sono detti *identificatori di variabili* .**
- **È conveniente che siano di tipo mnemonico:**

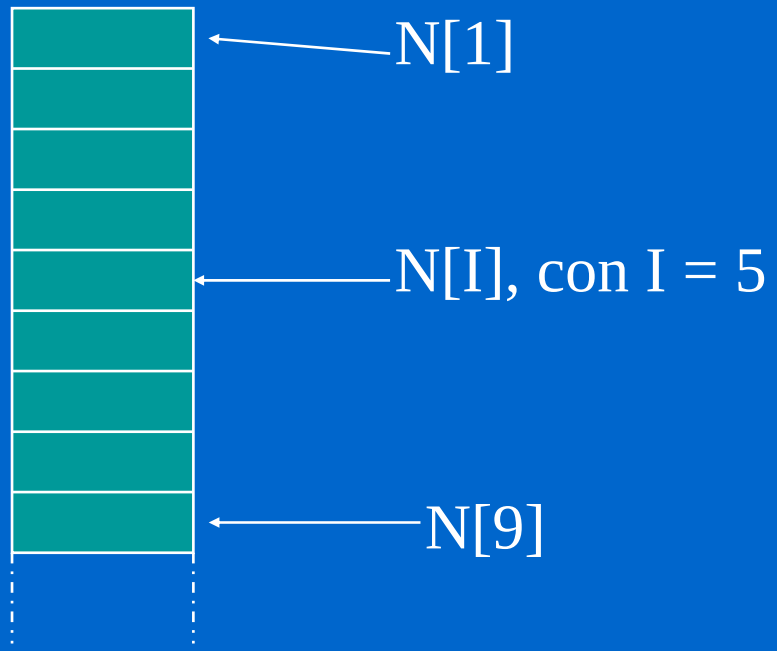
  meglio *moltiplicando*, *moltiplicatore*, *prodotto*.
- **Riferirsi a singoli elementi può non bastare. Esempio:**

  cercare il massimo fra quattro numeri.

1 Leggi N1, N2, N3, N4
2 *max* ← N1
3 IF N2 > *max* THEN
  3.1 *max* ← N2
4 IF N3 > *max* THEN
  4.1 *max* ← N3
5 IF N4 > *max* THEN
  5.1 *max* ← N4
6 stampa “Il massimo è” *max*

- Quando si devono effettuare operazioni su elementi simili, cioè della stessa specie, conviene “strutturare” i dati, ad esempio in un “vettore” e utilizzare etichette con indici.
- Si individua ogni elemento con una etichetta, ad esempio N, più un indice che lo distingue dagli altri dello stesso tipo.
- Normalmente si rappresenteranno così: N[1], N[2] o, più genericamente, N[I], dove I funge da etichetta dell'indice.

Graficamente:



1 Per I da 1 a 4

  1.1leggi N[I]

2 *max* ← N[1]

3 Per I che va da 2 a 4

  3.1IF N[I] > *max* THEN

    3.1.1*max* ← N[I]

4 stampa “Il massimo è” *max*

- **Meglio ancora generalizzare l'algoritmo rispetto al numero di elementi trattati. Si può ottenere in due modi:**
  - **fissando come *parametro* il numero di elementi, cioè come *costante* simbolica definita in testa all'algoritmo;**
  - **definendolo come *variabile* il cui valore è caricato da un dispositivo di ingresso all'inizio del programma.**

1 Leggi *num_dati*

2 FOR *indice* FROM 1 TO *num_dati*

  3.1 leggi N[*indice*]

3 *max* ← N[1]

4 FOR *indice* FROM 2 TO *num_dati*

  4.1 IF N[*indice*] > *max* THEN

    4.1.1 *max* ← N[*indice*]

5 stampa “Il massimo è” *max*

- **Altro esempio: diagramma di flusso che effettua la somma di N numeri.**
- **Verrà utilizzato il ciclo while invece del for.**

1 Leggi *num_dati*
2 *indice* ← 1
3 While *indice* ≤ *num_dati*
  3.1 leggi N[*indice*]
  3.2 incrementa *indice*
4 *somma* ← 0
5 *indice* ← 1
6 While *indice* ≤ *num_dati*
  6.1 *somma* ← somma + N[*indice*]
  6.2 incrementa *indice*
7 stampa “La somma è” *somma*

Algoritmi

Che cosa s'intende comunemente per algoritmo ?

"Un algoritmo è costituito da un insieme finito di passi distinti e non ambigui che, eseguiti a partire da assegnate condizioni iniziali, producono l'output corrispondente e terminano in un tempo finito".

La definizione precedente implica aver identificato uno o più “passi” che si è in grado di eseguire:

occorre disporre di una serie di operazioni (elementari o complesse) e di un “esecutore” capace di realizzare tali operazioni.

La descrizione di un algoritmo dipende dunque da quell'insieme di operazioni.

Per la soluzione degli esercizi proposti si suppone di disporre delle seguenti operazioni:

- somma / sottrazione / prodotto tra numeri interi o numeri reali: il risultato è rispettivamente un numero intero o un reale

- divisione tra numeri reali: il risultato è un numero reale
- divisione tra numeri interi: il risultato è la parte intera del quoziente (*quota*)
- modulo tra numeri interi: il risultato è il resto della divisione tra numeri interi

- confronto per >, ≥, =, <, ≤ tra numeri interi e numeri reali: il risultato è un valore logico, `VERO` o `FALSO`

- and (“*e insieme*”) / or (“*oppure*”) / not (*negazione logica*) tra valori logici: il risultato è ancora di tipo logico

Alcune altre operazioni su dati strutturati (accesso ad elementi di un vettore, ecc.) verranno introdotte in seguito.

Il metodo

I punti salienti del metodo adottato possono essere così riassunti:

- scomposizione dei problemi complessi in problemi più semplici (si suppone che siano noti un certo numero di problemi elementari di cui si conosce la soluzione);

- focalizzazione dell’attenzione: riconoscimento di uno o più “passi” che risultano risolutivi (la soluzione può essere ottenuta, ad esempio, ripetendo più volte lo stesso passo);

- minimizzazione delle differenze tra “descrizione attuale della soluzione” ed “obiettivo”; si noti che ciò implica individuare e tentare di risolvere dapprima il problema – o l’aspetto del problema – più rilevante, rimandando a fasi successive il raffinamento della soluzione;

- utilizzo delle analogie: se il problema che si sta affrontando può essere ricondotto alla *forma* di un problema che si è già risolto, si può riutilizzare lo stesso tipo di soluzione.

Esercizio 1

Realizzare il diagramma di flusso per il calcolo dell'area geometrica di un triangolo.

I. Leggi la *base*
II. Leggi l'*altezza*
III. Calcola *base* * *altezza* / 2, risultato in *area_triangolo*
IV. Fine.

Esercizio 2

Realizzare un algoritmo per il calcolo della tabellina pitagorica di un numero.

Occorre identificare un *passo significativo*:

1 Calcola *numero* ∗ *fattore*, risultato in *prodotto*

Stabilisco *quante volte* devo ripetere l’operazione:

1 Con *fattore* che assume valori da 1 a 10

1.1 Calcola *numero* * *fattore*, risultato in *prodotto*

Completo con i dettagli:

1 Leggi *numero*
2 Con *fattore* che assume valori da 1 a 10
  2.1 Calcola *numero* $*$ *fattore*, risultato in *prodotto*
  2.2 Visualizza *prodotto*
3 Fine

Esercizio 3

Realizzare un algoritmo per il calcolo della tabellina pitagorica dei numeri da 1 a 10.

Occorre identificare un *passo significativo*:

1 Calcola la tabellina di *numero* (operazione già nota)

Stabilisco *quante volte* devo ripetere l'operazione:

1 Con *numero* che assume valori da 1 a 10

1.1 Calcola la tabellina di *numero*

L’algoritmo completo è:

1 Con *numero* che assume valori da 1 a 10
  1.1 Con *fattore* che assume valori da 1 a 10
    1.1.1 Calcola *numero* * *fattore*, risultato in *prodotto*
    1.1.2 Visualizza *prodotto*
2 Fine

Esercizio 4

Trasformare il seguente algoritmo non strutturato in un equivalente algoritmo strutturato:

1 si pone 1 in *num_val* e 0 in *accum*
2 con *indice* che va da 1 a 100 si esegue
  2.1 si legge un dato intero in *varint*
  2.2 se *varint* < 0 si va a 3.
  2.3 si somma *varint* ad *accum*
  2.4 si incrementa *num_val*
3 si decrementa *num_val*
4 si visualizza *accum*
5 fine.

I. L’algoritmo contiene un ciclo a contatore basato su *indice*, ma dal ciclo si esce anche se si verifica l’evento “*varint* < 0”.

Occorre pertanto trasformare il ciclo a contatore in un ciclo basato su evento (ciclo WHILE o ciclo REPEAT-UNTIL).

Si esce dal ciclo se sono stati trattati tutti i dati (test su *indice*) oppure se il dato letto in *varint* è negativo.

Nell'adottare un ciclo basato su evento occorre invece determinare la condizione per continuare il ciclo:

si permane nel ciclo finché non sono stati trattati tutti i dati e allo stesso tempo *varint* è positivo o nullo.

1 si pone 0 in *accum*
2 si pone *indice* a 0, *varint* a 0
3 finché (*indice* < 100) e insieme
(*varint* ≥ 0) si esegue
3.1 si legge un dato intero in *varint*
3.2 se *varint* ≥ 0
3.2.1 si somma *varint* ad *accum*
3.2.2 si incrementa *indice*
4 si pone *indice* in *num_val*
5 si visualizza *accum*
6 fine.

- All’interno del ciclo si è inserito un test su *varint* per ottenere un comportamento equivalente a quello dell’algoritmo specificato dal testo dell’esercizio.

- Si sono eliminati dei passi inutili.

Esercizio 5

Realizzare un algoritmo per calcolare la media aritmetica di una sequenza forniti da un dispositivo di input.

Analisi nel dominio della matematica:

- la media si calcola effettuando la somma di tutti i numeri e dividendo per la quantità di numeri;
- la divisione è lecita se il divisore non è nullo.

Identifico un *passo significativo*:

1 Leggi *numero*
2 Calcola *numero* + *accumulatore*, risultato in *accumulatore*
3 Incrementa *quanti_numeri*

Determino quante volte effettuare il *passo significativo*, tengo conto di dover effettuare le inizializzazioni e della condizione per effettuare correttamente la divisione.

L’algoritmo è:

1 Inizializza *accumulatore* e *quanti_numeri* con il valore 0
2 Finché ci sono numeri
  2.1 Leggi *numero*
  2.2 Calcola *numero* + *accumulatore*, risultato in *accumulatore*
  2.3 Incrementa *quanti_numeri*
3 Se *quanti_numeri* > 0
  3.1 Calcola *accumulatore* / *quanti_numeri*, risultato in *media*
4 Altrimenti
  4.1 Visualizza “media non calcolabile”
5 Fine.

Esercizio 6 (MCD I versione)

Realizzare un algoritmo per il calcolo del massimo comun divisore (M.C.D.) di due numeri interi positivi.

Analisi nel dominio della matematica:

il M.C.D. tra A e B è il numero più grande che divide pienamente A e B (divisione con resto nullo).

- **Individuo un passo significativo:**

  Se ((A mod *divisore*) = 0) AND ((B mod *divisore*) = 0)

  pongo *divisore* in *mcd*

- **Mi concentro sulla ricerca del massimo:**

  se provo per tutti i valori da 1 fino al minimo tra A e B, in *mcd* resta il massimo cercato.
- **Ne discende quante volte eseguo il passo fondamentale:**

  Con *divisore* da 1 a *min_a_b* (che dovrà contenere il minimo tra A e B)

- **Algoritmo:**

Leggi A

Leggi B

Se A > B

pongo B in *min_a_b*

altrimenti

pongo A in *min_a_b*

Con *divisore* da 1 a *min_a_b*

Se ((A mod *divisore*) = 0) AND

((B mod *divisore*) = 0)

pongo *divisore* in *mcd*

Stampo “ Il m.c.d tra A e B è” *mcd*

Commento:

se A e B sono numeri grandi, il numero di iterazioni è alto.

L’algoritmo è corretto ma *inefficiente*.

Esercizio 6 (MCD II versione)

Realizzare un algoritmo per il calcolo del massimo comun divisore (M.C.D.) di due numeri interi positivi.

Analisi nel dominio della matematica:

Euclide ha già proposto un algoritmo efficiente, che sfrutta la seguente proprietà: se *a* > *b*, il M.C.D. tra *a* e *b* è anche il M.C.D. tra *a*-*b* e *b*;

il procedimento può essere reiterato e termina quando, in seguito a sottrazioni successive, si ottengono due numeri uguali.

Algoritmo:

Leggi *dato_a* e *dato_b*

Finché *dato_a* ≠ *dato_b*

Se *dato_a* > *dato_b*

calcola *dato_a* - *dato_b*,

risultato in *dato_a*

altrimenti

calcola *dato_b* - *dato_a*,

risultato in *dato_b*

Assegna a *massimo_comun_divisore* il valore di *dato_a*

Fine.

M.C.D. (III Soluzione)

Si può utilizzare il procedimento di scomposizione in fattori primi.

Si ipotizza di poter disporre di un vettore *vett_primi*, di lunghezza adeguata, che contenga i numeri primi almeno fino al più piccolo dei numeri considerati.


1
2
3
5
7
11
13
...

Analizzo il problema nel dominio della matematica:

Il M.C.D. è costituito dal prodotto dei numeri primi per i quali i due numeri sono pienamente divisibili.

Occorre tener presente che un numero primo può contribuire al M.C.D. con molteplicità superiore ad 1.

Identifico un *passo significativo*:

cerca fattore comune di *dato_a* e *dato_b* in *vett_primi*, determina se *trovato* e restituisce il valore in *fatt_comun*

Se *trovato* è vero

moltiplica il valore precedente di *massimo_comun_denominatore* per *fatt_comun*

riduci (dividendo) *dato_a* e *dato_b* di *fatt_comun*

Nota: il primo passo (“cerca...”) può essere ritenuto un’azione primitiva, se esiste un “*modulo*” che lo attua.

Lo si trasforma in *procedura* o *funzione*, in cui *dato_a* e *dato_b* sono parametri d’ingresso e *trovato* e *fatt_comun* sono parametri d’uscita.

Occorre determinare quante volte occorre ripetere il *passo significativo*, e successivamente le inizializzazioni.

Riscrivendo l'algoritmo in modo più formale, si ha:

Main

Legge *dato_a* e *dato_b*

Pone 1 in *mcd*

Pone VERO in *trovato*

Finché *trovato* è VERO

***Cerca_fattore_comune* (ingressi: *dato_a*, *dato_b*; uscite: *trovato*, *fatt_comun*)**

Se *trovato* è VERO

Calcola *fatt_comun* * *mcd*, risultato in *mcd*

Calcola *dato_a* / *fatt_comun*, risultato in *dato_a*

Calcola *dato_b* / *fatt_comun*, risultato in *dato_b*

Visualizza *mcd*

Fine.

Passo alla soluzione del sottoproblema.

Devo verificare se un generico elemento di *vett_primi* divide pienamente *dato_a* e *dato_b*.

Il *passo significativo* è:

Se ((*dato_a* MOD *vett_primi*[*indice*]) = 0)

e insieme ((*dato_b* MOD

vett_primi*[*indice*]) = 0)

Pone VERO in *trovato*

Pone *vett_primi*[*indice*] in *fatt_comun*

altrimenti

Incrementa *indice*

Determino quante volte devo eseguire questo passo:

Smetto di cercare appena trovo il fattore comune.

Il fattore cercato non può essere superiore al minimo tra *dato_a* e *dato_b*.

Determino le inizializzazioni: devo ignorare il primo numero primo, cioè 1: pertanto l'indice di scansione del vettore *vett_primi* deve iniziare da 2.

L’algoritmo completo è:

***Cerca_fattore_comune*(ingressi: *dato_a*, *dato_b*; uscite: *trovato*, *fatt_comun*)**

Pone 2 in *indice*

Se *dato_a* < *dato_b*

pone *dato_a* in *minimo_ab*

altrimenti

pone *dato_b* in *minimo_ab*

Finché (NO *trovato*) e insieme
(*vett_primi*[*indice*] ≤ *minimo_ab*)
Se ((*dato_a* MOD *vett_primi*[*indice*]) = 0)
e insieme ((*dato_b* MOD
***vett_primi*[*indice*]) = 0)**
Pone VERO in *trovato*
Pone *vett_primi*[*indice*] in *fatt_comun*
altrimenti
Incrementa *indice*
Fine.

Conclusioni

Ai criteri generali per creare buoni algoritmi si possono aggiungere i seguenti suggerimenti:

analizzare il problema nel suo dominio specifico, cercando di evidenziare tutte le specifiche esplicite ma anche quelle *implicite*;

creare un esempio, il più generale possibile;

creare esempi dei casi limiti;

nella realizzazione dell’algoritmo, cercare di trattare tutti i dati allo stesso modo, evitando di trattare a parte il primo dato;

prendere come riferimento i dati dell’esempio tipico e partire non dall’inizio, né dalla fine, ma dal centro, immaginando che i dati precedenti siano stati già trattati: individuare così il passo fondamentale;

mettere a posto le inizializzazioni affinché l’algoritmo operi correttamente alla partenza;

verificare che con questo vengano trattati correttamente anche i casi limite.

Se ciò non accade, tentare di effettuare piccole modifiche per far rientrare anche i casi limiti: se le modifiche sono onerose, valutare l'opportunità di *rigettare completamente* l'algoritmo e di crearne un altro su nuove basi.

Data la stretta dipendenza tra azioni e base dati, anche quest'ultima dovrebbe essere valutata con occhio critico.