13 - Gli algoritmi e la risoluzione di problemi

Sommario

- Passi per la risoluzione di problemi
- Problemi di ricerca e ordinamento
- Algoritmi iterativi: la ricerca lineare

Passi per la risoluzione di problemi

1) Comprendere il problema (*analisi dei requisiti*)

2) Progettare una soluzione (*algoritmo)*

3) Realizzare l'algoritmo (*scrittura del codice*)

4) Sottoporre la soluzione a test e correggere eventuali errori (*verifica del programma, testing e debugging*)

5) Manutenzione e aggiornamento

Passi per la risoluzione di problemi

1) Comprendere il problema (*analisi dei requisiti*)
   Può richiedere interazione con la fonte
2) Progettare una soluzione (*algoritmo)*
   La parte più difficile, che richiede creatività
3) Realizzare l'algoritmo (*scrittura del codice*)
   Richiede la conoscenza del linguaggio di prog.
4) Sottoporre la soluzione a test e correggere eventuali errori (*verifica del programma, testing e debugging*)
   Può richiedere più tempo della scrittura del codice
5) Manutenzione e aggiornamento
   Il costo dipende dalla qualità del codice

L'età dei figli...

Non sempre le fasi viste prima possono essere fatte in cascata. Esempio di problema che richiede di intercalare **analisi** e **progettazione**:

- A deve trovare l'età dei tre figli di B.
  - B dice ad A che il prodotto delle età è 36.
  - A risponde che i dati non sono sufficienti.
  - B dice ad A la somma delle età.
  - A risponde che i dati non sono sufficienti.
  - B dice ad A che il maggiore suona il pianoforte.
  - A dice a B l'età dei suoi figli.
- Che età hanno i tre figli?

Verso la soluzione...

a. Triples whose product is 36

(1,1,36)	(1,6,6)
(1,2,18)	(2,2,9)
(1,3,12)	(2,3,6)
(1,4,9)	(3,3,4)

b. Sums of triples from part (a)

1 + 1 + 36 = 38	1 + 6 + 6 = 13
1 + 2 + 18 = 21	2 + 2 + 9 = 13
1 + 3 + 12 = 16	2 + 3 + 6 = 11
1 + 4 + 9 = 14	3 + 3 + 4 = 10

Alcune tecniche per risolvere problemi

- (Sempre) Identificare con chiarezza il risultato richiesto e i dati di ingresso (***leggere con attenzione il testo***)
- Provare a risolvere un problema analogo più semplice
  - es: trovare il massimo di N numeri. ***Cominciamo da tre...***
- Raffinamento: dividere il problema in problemi più piccoli, e combinarne la soluzione (metodologia *top-down*, *divide et impera*)
- Metodologia *bottom-up*: cominciare a risolvere sottoproblemi del problema dato, e estendere le soluzioni
  - Esempio dei puzzle, con e senza immagine della soluzione

Risolvere problemi (problem solving)

- In generale, risolvere un problema richiede una forma di creatività: è una forma di arte.
- **Risolvere un problema** può voler dire
  - trovare un algoritmo per un problema che non ha ancora una soluzione (o non la conosciamo)
  - trovare un **nuovo** algoritmo, possibilmente **più efficiente** di altri che già conosciamo.
- Come per tutte le arti, entro certi limiti si può imparare, ma ci sono persone più dotate e altre meno dotate.
- Noi non faremo ***problem solving***, ma vedremo degli algoritmi conosciuti per alcuni problemi specifici

Strutture di controllo iterative

- Comando iterativo dello pseudocodice:

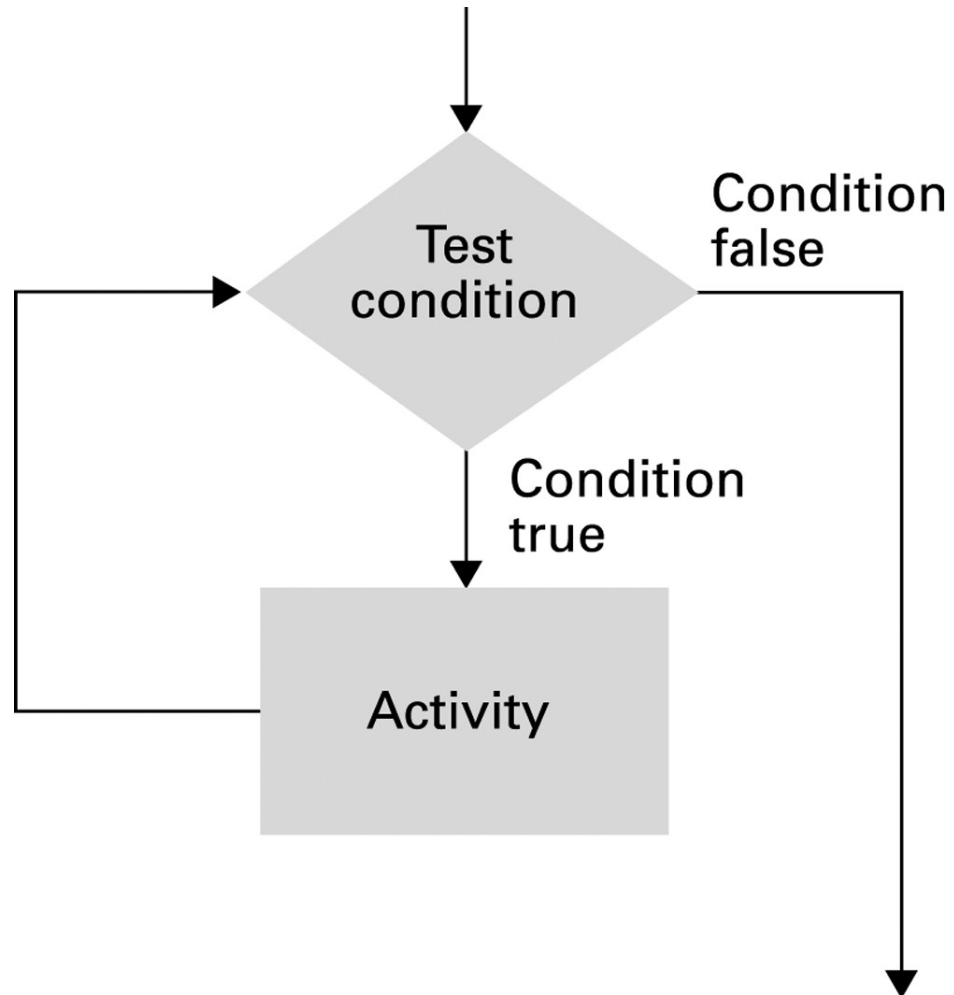
```
while (condizione) do
    (corpo del ciclo)
```

- Altro comando

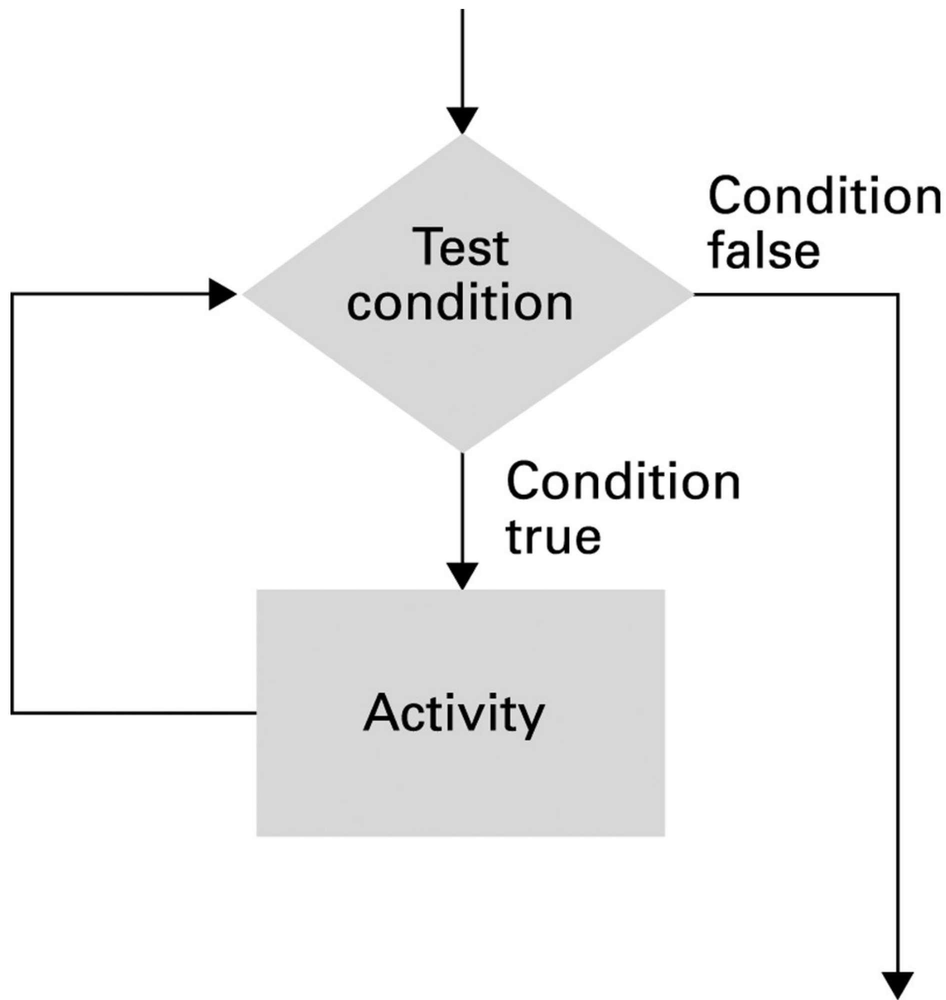
```
repeat (corpo del ciclo)
    until(condizione)
```

- Differenze:
  - Il corpo del **`repeat`** viene sempre eseguito almeno una volta
  - Il **`repeat`** termina quando la condizione è vera

La struttura del while: diagramma di flusso

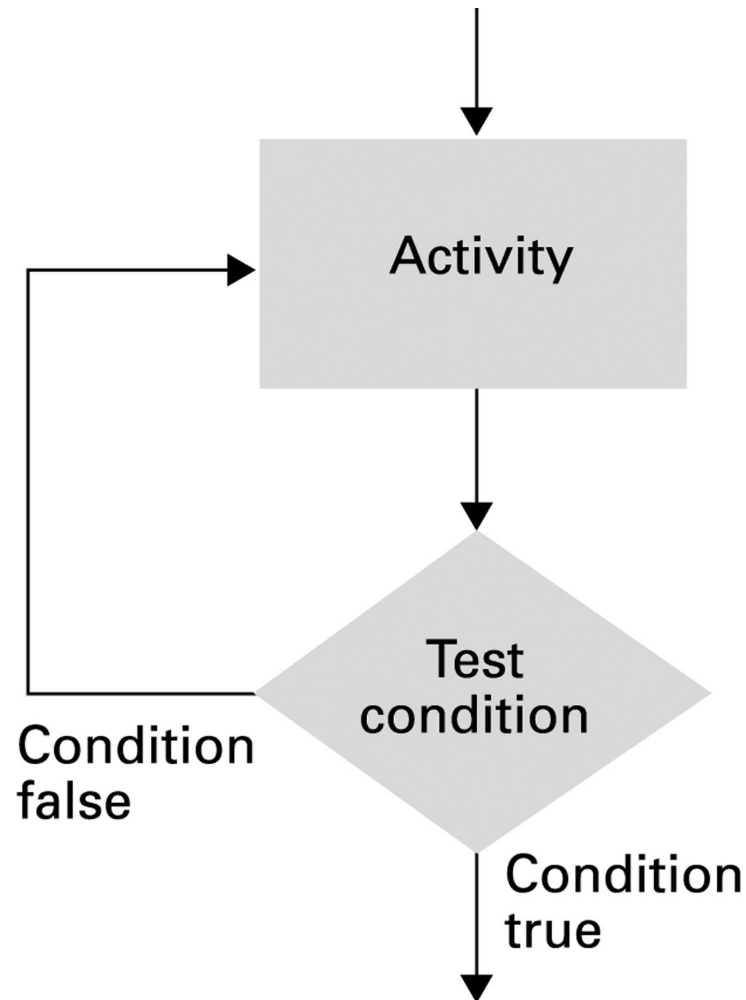


La struttura del while: diagramma di flusso



I diagrammi di flusso sono un'alternativa allo pseudocodice per scrivere algoritmi in modo informale ma preciso. Di moda negli anni '70-'80, oggi sono meno usati.

La struttura del repeat: diagramma di flusso



Due problemi fondamentali: ricerca e ordinamento

- **Ricerca:** dato un valore X e una sequenza di valori S, dire se X compare in S.
  - Ricerca lineare
  - Ricerca binaria
- **Ordinamento:** data una sequenza S di valori confrontabili tra loro, riorganizzare S in modo che contenga gli stessi valori, ma in ordine crescente
  - Ordinamento per inserimento (insertion sort)
  - Bubble sort, Quick sort, Merge sort

La ricerca lineare (o sequenziale)

- Identifichiamo dati d'ingresso (***input***) e risultato (***output***)
- **Input**: il valore **X** e la sequenza **S = $s_1, s_2, \ldots, s_N$**
  Quindi indichiamo con **N** la lunghezza della sequenza.
- **Output**: un valore booleano (TRUE o FALSE) che indica se **X** è presente nella sequenza oppure no.
  Usiamo il comando RETURN per restituire il risultato
- Cominciamo con una sequenza di due elementi (assumiamo N = 2):

```
if (X = s1) then RETURN TRUE;
else  if (X = s2) then RETURN TRUE
      else RETURN FALSE
```

- Quindi confrontiamo **X** con un elemento alla volta, restituiamo TRUE se è uguale, e solo alla fine, se è diverso da tutti, restituiamo FALSE

La ricerca lineare (2)

- Se la sequenza ha lunghezza arbitraria, non possiamo usare una catena di **if-then-else**, perché non sappiamo quanti ne servono
- Usiamo un **while-do** che ad ogni ciclo confronta **X** con il prossimo elemento della sequenza, individuato dalla variabile **k**. Il **while** termina quando abbiamo esaminato tutta la sequenza (**k>N**)

```
procedure RicLin (X, S)
  N ← lunghezza della sequenza S;
  k ← 1;     //indice del prossimo elemento di S
  while (k <= N) do  // termina quando k = N+1
     if ( X = Sk)      // se ho trovato X
     then  RETURN TRUE; // restituisco TRUE e
  esco
     else  k ← k+1; // passo al prossimo elemento
  RETURN FALSE; // se arrivo qui, X non c'è in S
```

La ricerca lineare (3)

- Presentiamo una piccola variante della procedura RichLin. Se troviamo **X**, invece di uscire "brutalmente" dal ciclo **while** con RETURN TRUE, assegnamo TRUE a una variabile booleana (**trovato**) in modo che la condizione del **while** diventi falsa.

```
procedure RicLin (X, S) // in verde i cambiamenti
  N ← lunghezza della sequenza S;
  k ←  1;    //indice del prossimo elemento di S
  trovato ← FALSE; // variabile booleana
  while (NOT trovato AND  k <= N) do
     if ( X = Sk)
     then  trovato <=  TRUE;
     else   k ← k + 1;
  RETURN trovato;
```

Ricerca lineare: variazioni sul tema

- Supponiamo che la sequenza sia ordinata in modo crescente. Cosa cambia?
- Si può interrompere la ricerca appena si trova un valore maggiore di **X**, perché **X** non può comparire oltre (altrimenti la sequenza non sarebbe ordinata).
- La complessità della ricerca lineare. Intuitivamente:
  - numero di "operazioni elementari" in funzione della dimensione dell'input (cioè della lunghezza di **S**)
  - per algoritmi di ricerca, si considerano ***i confronti*** come "operazioni elementari"
  - nel caso pessimo (**X** non compare) dobbiamo fare **N** confronti, con **N** lunghezza di **S**.
- La complessità della ricerca lineare è **lineare** (o **O(n)**).