Algoritmi e diagrammi di flusso

Alessandro Bugatti

Algoritmi

Per programmare un computer in modo che sia in grado di risolvere un problema dato bisogna come prima cosa essere in grado di risolvere il problema da un punto di vista teorico, poiché la programmazione altro non è che *istruire* un computer a compiere una serie di azioni che poi svolgerà al nostro posto, in maniera più rapida, senza annoiarsi e senza commettere errori. È evidente che se il programmatore non è in grado di risolvere il problema, il computer eseguirà una serie di azioni che non permetteranno di arrivare alla soluzione. L'insieme di azioni che portano alla soluzione di un problema viene in genere definito *algoritmo* e una definizione semplice e efficace potrebbe esser questa:

> **Algoritmo**: *Un algoritmo è una sequenza di passi (istruzioni) elementari che, quando eseguiti, portano alla soluzione di un problema di carattere generale.*

Analizzando questa definizione si possono notare una serie di cose interessanti:

- *una sequenza di passi*: quindi non un insieme in un ordine qualsiasi, ma una sequenza, cioè ogni istruzione deve essere nel posto giusto, con una precisa istruzione che la precede e una che la segue
- *elementari*: il concetto di elementare va riferito rispetto all'esecutore dell'algoritmo. Siccome per un programmatore l'esecutore di un algoritmo è il computer, allora elementari significa istruzioni che possono essere eseguite immediatamente dal computer (come ad esempio fare la somma di due numeri), mentre non sono elementari quelle istruzioni che per essere eseguite hanno bisogno di essere scomposte in istruzioni più semplici (come ad esempio calcolare l'area di un rettangolo)
- *quando eseguiti*: l'algoritmo è una descrizione statica della soluzione, deve essere eseguito per portare a una soluzione effettiva
- *portano alla soluzione*: sembra ovvio ma uno dei requisiti di un algoritmo è che deve portare alla soluzione di un problema. Un algoritmo che non porti alla soluzione di un problema è inutile
- *di carattere generale*: un algoritmo dipende dai valori in ingresso e in questo modo può risolvere tutti i casi di un problema. Se ad esempio avessimo l'algoritmo per il calcolo dell'area di un rettangolo di base 3 e altezza 4 questo sarebbe utile solo nel caso specifico in cui interessa l'area di quel particolare rettangolo, mentre è ovviamente molto più interessante avere un algoritmo che calcoli l'area di qualsiasi rettangolo, dati i valori della base e dell'altezza.

Diagrammi di flusso

Per rappresentare un algoritmo esistono molti modi, alcuni elementari, altri un po' più complessi. Probabilmente il modo più naturale è quello di "raccontarlo" a parole, dandone una descrizione il più precisa possibile. Se ad esempio il problema da risolvere fosse di stabilire se un numero maggiore di 1 è pari o dispari la descrizione dell'algoritmo risolutivo potrebbe essere: una volta letto il valore del numero, guarda se l'ultima cifra è uno 0, un 2, un 4, un 6 o un 8: se è così allora il numero è pari, altrimenti è dispari. Questa descrizione sembra in effetti adeguata ma potrebbe essere soggetta a interpretazioni: ad esempio per il numero 1024 l'ultima cifra è il 4 o l'1? Considerando, per quanto detto sopra, che i nostri algoritmi verranno poi trasformati in programmi per computer, è necessario trovare altre modalità più precise e chiare.

La rappresentazione che utilizzeremo è conosciuta in informatica con il nome di *diagrammi di flusso* (in inglese *flow-chart*) o *diagrammi a blocchi* ed è un sistema grafico di rappresentazione di un algoritmo, costituito da una serie di blocchetti elementari che vanno a unirsi per formare il flusso di esecuzione delle istruzioni (da cui i nomi).

Blocchi elementari

Ogni diagramma di flusso inizia con un'ellissi contenente la parola BEGIN (oppure START o INIZIO) e termina in una o più ellissi contenenti la parola END (o STOP o FINE), come si può vedere in figura 1.

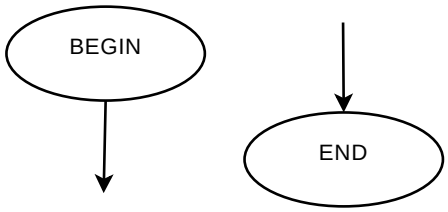

Fig. 1: Blocchi di inizio e fine

Per esprimere invece le istruzioni di input/output, quelle nelle quali si chiede all'utente di inserire un valore oppure gli si mostra un valore calcolato dal programma, si usano dei parallelogrammi, contenenti i nomi delle variabili preceduti, in genere, dalla scritta IN (o LEGGI) o dalla scritta OUT (o scrivi), a seconda del tipo di operazione che si intende fare. Nella figura 2 si può vedere come il primo blocco effettui una lettura delle variabili A e B, mentre il secondo comunichi il contenuto della variabile C.

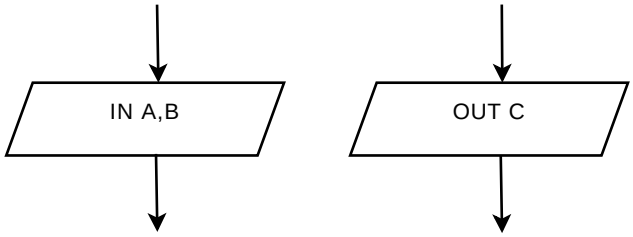

Fig. 2: Blocchi di I/O

Le azioni vengono rappresentate all'interno di un rettangolo, descritte in una forma comprensibile per chi dovrà leggere il diagramma. Nell'esempio di figura 3 si può vedere che viene calcolata *c* come somma delle variabili *a* e *b* e successivamente viene incrementata *n* di un'unità. Quando due o più azioni vengono eseguite l'una di seguito all'altra si parla di *sequenza*.

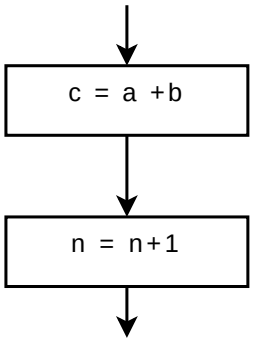

Fig. 3: Azioni

L'ultimo mattone necessario a poter rappresentare ogni possibile algoritmo è il costrutto di *selezione*, che ci permette, arrivati a un certo punto del programma, di scegliere che strada seguire in base alla verità o falsità di un'affermazione. Nell'esempio in figura 4 si può vedere che l'affermazione da verificare è $a>b$ e se risulta vera allora viene assegnato a *c* il valore di *b*, altrimenti a *c* viene assegnato il doppio del valore di *a*. Successivamente il flusso di esecuzione dell'algoritmo prosegue per affrontare altri passi.

Il costrutto di *selezione* è inoltre necessario per rappresentare quello che nella terminologia degli algoritmi viene definito costrutto di *iterazione*, cioè un'azione o insieme di azioni che viene ripetuta un certo numero di volte oppure finché non si verifica una certa condizione e che permette di rappresentare in maniera elegante moltissimi algoritmi che al loro interno prevedono questo tipo di ripetizioni. Nell'esempio in figura 5 si può vedere come l'istruzione che assegna ad *a* il doppio del proprio valore venga ripetuta finché

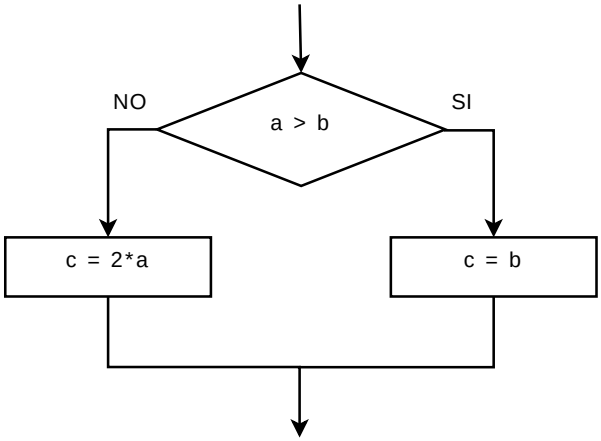

Fig. 4: Selezione

a non diventa maggiore di *b*: in quel momento si scende nel ramo di destra della selezione e *b* viene posto uguale ad *a*, per poi proseguire nelle successive istruzioni dell'algoritmo.

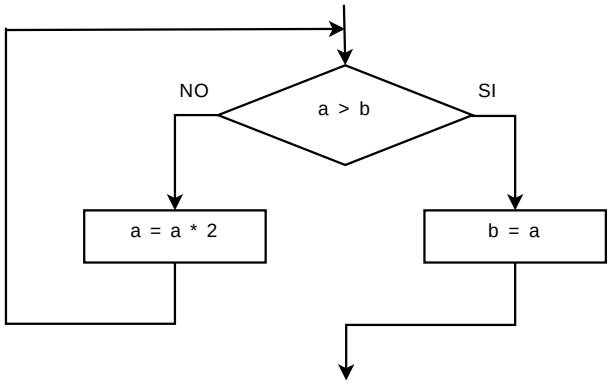

Fig. 5: Iterazione

Tutti questi blocchi elementari possono essere combinati per dare origine a qualsiasi algoritmo: c'è difatti un teorema conosciuto come teorema di Bohm-Jacopini che afferma che qualunque algoritmo può essere implementato utilizzando le tre sole strutture di sequenza, selezione e iterazione.

Come esempio finale di un diagramma di flusso che descrive un algoritmo si può guardare la figura 6, che rappresenta la soluzione al problema di determinare se un dato numero intero, maggiore o uguale a due, è primo oppure no[1]. Questo diagramma inizia leggendo il valore inserito dall'utente e copiandolo nella variabile *N*, poi inizializza la variabile *a* al valore 2 e controlla se il numero *N* è divisibile per *a*. Se sì allora il numero non è sicuramente primo perché abbiamo trovato un divisore più piccolo di *N*. Altrimenti *a* viene incrementato di un'unità e si controlla se è diventato uguale a *N*: se la risposta è affermativa allora il numero è primo perché vuol dire che non ci sono divisori di *N*, altrimenti si riparte dal controllo sulla divisibilità di *N* per *a* (è evidente che l'algoritmo è destinato prima o poi a terminare).

[1] Per semplicità l'algoritmo è nella sua versione più banale, ma il controllo potrebbe essere fatto in modo da ridurre il numero massimo di iterazioni.

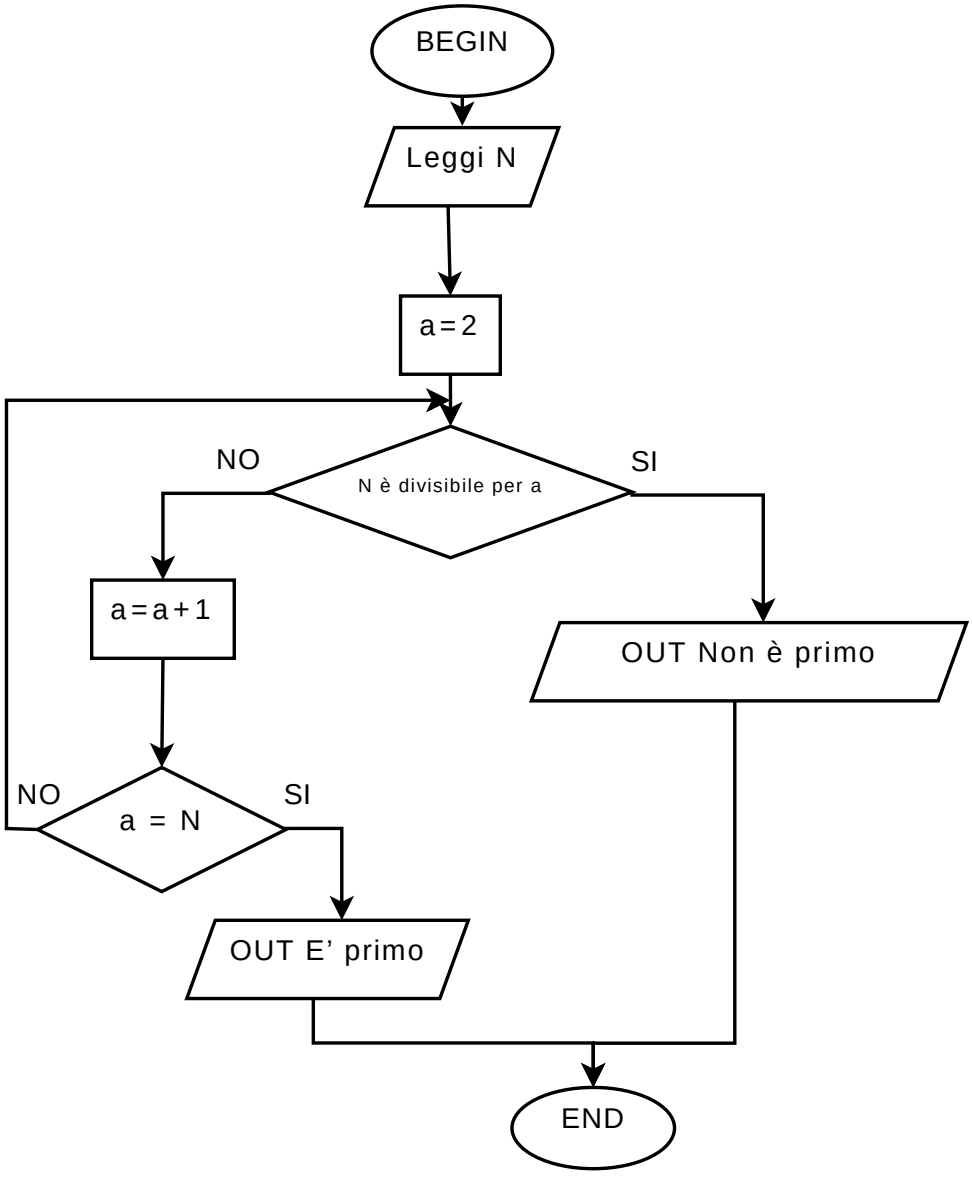

Fig. 6: Algoritmo per verificare la primalità di un numero