



Dal problema all'algoritmo

A cura di A. Orlandini

Obiettivi

Completata questa unità saremo in grado di:

- **Identificare un problema risolubile da un elaboratore automatico**
- **Individuare le istruzioni adatte ad essere comprese dall'esecutore**
- **Descrivere con sequenze di istruzioni non ambigue la risoluzione del problema**
- **Distinguere tra diversi tipi di strategie risolutive quella piu' efficiente**

Contenuti

- **Cosa è un problema**
- **Esempio di problema: il MCD**
- **Cosa è un algoritmo**
- **Problemi decidibili e non**
- **Cosa è un programma**
- **Cosa è una variabile**
- **Algoritmi ed esecutori**
- **Correttezza ed efficienza**
- **Programmazione strutturata**

Introduzione

- Vengono introdotti alcuni concetti base dell'Informatica: **Problema**, **Algoritmo**, **Programma**, **Processo (di elaborazione)**
- In modo sintetico, attraverso semplici esempi, sono presentati alcuni aspetti concettuali e fondamentali dell'Informatica; essi verranno ripresi in maniera più estesa in seguito
- In particolare si considerano aspetti di **correttezza** e di **efficienza** della soluzione automatica di problemi, anche attraverso l'uso di metodi di programmazione strutturata

Problema

- **Problemi di interesse** sono quelli per cui è possibile una **precisa formalizzazione**, in un qualche formalismo espressivo, ad esempio quello offerto dalla matematica
- **Esistono problemi** che sono tanto ben esprimibili e comprensibili da poterne **definire una strategia di soluzione** basata sull'applicazione sistematica di ben precise regole operative che consente di ottenere i risultati attesi a partire dai dati disponibili
- **In molti di questi casi** è possibile affidare l'applicazione delle regole di soluzione ad **un esecutore specializzato** in grado di svolgere (più o meno) rapidamente i compiti affidatigli

Un esempio . . .

**“Determinare il Massimo Comun Divisore di
due numeri interi positivi”**

***Enunciato informale del problema in
linguaggio naturale***

... Un esempio

Determinare $Z = \text{MCD}(X, Y)$

dove $X, Y, Z \in \mathbb{N}^+, X \geq Y$

Formalizzazione del problema in linguaggio naturale e matematico, in cui compaiono i simboli $X, Y, e Z$ da interpretare come oggetti appartenenti al dominio \mathbb{N}^+ dei numeri interi positivi, cioè dei naturali escluso 0, e il simbolo MCD da interpretare come la funzione matematica Massimo Comun Divisore

Massimo Comun Divisore . . .

$$\text{MCD}(X, Y) = \max(D_X \cap D_Y)$$

$$D_X = \{ d \in \mathbb{N}^+ \mid \exists q \in \mathbb{N}^+, x = d \cdot q \}$$

$$D_Y = \{ d \in \mathbb{N}^+ \mid \exists q \in \mathbb{N}^+, y = d \cdot q \}$$

Definizione matematica del MCD

... Massimo Comun Divisore ...

Soluzione 1

Applico la definizione matematica:

- calcolo i due insiemi D_X e D_Y dei divisori di X e di Y , rispettivamente
- costruisco l'insieme intersezione di D_X e D_Y
- determino il valore massimo dell'insieme intersezione

... Massimo Comun Divisore ...

Soluzione 2

- ❑ Individuo il valore m minimo tra X e Y
- ❑ Verifico se m divide sia X sia Y ; se sì allora m è il $MCD(X, Y)$, e ho finito
- ❑ Decremento m di 1 e ripeto il passo precedente, al più fino al valore $m = 1$

... Massimo Comun Divisore ...

Soluzione 3

Cerco altre proprietà del MCD ...

Siano $X, Y, Q \in \mathbb{N}^+$ e $R \in \mathbb{N}$.

Se $X = Q \cdot Y + R$ allora

$$**D_X \cap D_Y = D_Y \cap D_R**$$

Teorema (Proprietà) del MCD

. . . Massimo Comun Divisore . . .

Sulla base di questa proprietà, in questo caso, come anche nei precedenti, si può definire una sequenza di regole da utilizzare per risolvere, in modo molto efficiente, il problema posto:

□ $R = \text{MOD}(X, Y)$

(MOD calcola il resto della divisione di x e y)

□ se $R = 0$ allora Y è il MCD(X, Y), altrimenti il problema si riconduce al più semplice problema che è quello di ottenere MCD(Y, R)

. . . Massimo Comun Divisore

- Ciascuna delle sequenze di regole proposte è detta **Algoritmo** e rappresenta un metodo risolutivo del problema posto.
- Nell'ultimo caso l'algoritmo è noto come **Algoritmo di Euclide (AE)**

Ad esempio il calcolo di **MCD (187,34) = 17** procede così:

$$187 = 5 \cdot 34 + 17 \quad (17 = \text{MOD} (187,34))$$

$$34 = 2 \cdot 17 + 0 \quad (0 = \text{MOD} (34,17))$$

Commento

- ❑ Il problema posto richiede di determinare il Massimo Comun Divisore di due numeri interi positivi
- ❑ Sono state individuate alcune sequenze di regole (nel caso 3) che risolvono il problema
- ❑ Il problema ammette quindi almeno 3 algoritmi che lo risolvono in modi diversi
- ❑ Tra questi algoritmi scegliamo quello più efficiente, in termini di numero di operazioni di divisione: l'**Algoritmo di Euclide**

Cosa è un Algoritmo

- ❑ *Procedimento risolutivo di un problema*
- ❑ *Insieme di regole che, eseguite ordinatamente, permettono di ottenere i risultati del problema a partire dai dati a disposizione*

Perché un insieme di regole possa considerarsi un algoritmo deve rispettare alcune proprietà . . .

Proprietà di un Algoritmo

- ❑ **Non ambiguità** - *Le istruzioni devono essere univocamente interpretabili dall'esecutore dell'algoritmo (nel seguito l'esecutore sarà l'elaboratore, ma il problema si pone anche per esecutori umani)*
- ❑ **Eseguibilità** - *L'esecutore deve essere in grado, con le risorse a disposizione, di eseguire ogni istruzione, ed in tempo finito*
- ❑ **Finitezza** - *L'esecuzione dell'algoritmo deve terminare in un tempo finito per ogni insieme di valori di ingresso*

Ne deriva che . . .

Nella definizione di un algoritmo si deve:

- ❑ adottare un formalismo espressivo adeguato con una sintassi e semantica ben precise
- ❑ garantire che l'esecutore sia in grado di eseguire ciò che gli richiedo di fare, cioè conoscere le potenzialità dell'esecutore
- ❑ garantire che l'esecuzione termini in tempo finito

In tutto questo ci sono le problematiche principali che affronteremo nel seguito

Problemi decidibili

I problemi decidibili sono quelli per cui è possibile definire un algoritmo risolutivo, ad esempio:

- ❑ Dati 2 numeri, calcolarne la somma**
- ❑ Dati n numeri, calcolarne la somma**
- ❑ Dato un elenco nomi/numeri telefonico, trovare il numero telefonico di una data persona di cui si conosce il nome**
- ❑ Consultare una carta geografica**
- ❑ Progettare una rete elettrica / un ponte**

Problemi non decidibili

I problemi non decidibili sono, quindi, quelli per cui non è possibile definire un algoritmo risolutivo, ad esempio:

- ❑ Decidere se $f(x)$ è una funzione costante
- ❑ Determinare il cambio Euro / Dollaro al 1/1/2020

Efficienza

Nella ricerca della soluzione di un problema, cioè nella definizione di un algoritmo, ci si deve porre anche la questione dell'efficienza temporale e spaziale, come nel seguente esempio:

“ Ricerca il valore massimo in un insieme numerico $\{ a_1, a_2, \dots, a_n \}$ ”

Definizione matematica

“Esiste un indice i tale che $a_i \geq a_j$, per $j= 1,2,\dots,n$ ”

Efficienza temporale . . .

Soluzione 1 – (*Ovvia, banale*)

- ❑ **Applico la definizione matematica e verifico la proprietà per ogni valore di i da 1 a n**
- ❑ **Devo quindi eseguire un numero di confronti pari a**

$$(n)^2 = O(n^2)$$

cioè dell'ordine di n^2

... Efficienza temporale

Soluzione 2 – (Migliore)

- ❑ Assumo a_1 come massimo (provvisorio)
- ❑ Confronto con il successivo e decido il nuovo massimo (provvisorio); e così via di seguito ...
- ❑ Devo quindi eseguire un numero di confronti pari a
$$n - 1 = O(n)$$

cioè dell'ordine di n

Efficienza spaziale . . .

Supponiamo di avere una matrice di $n \times n$ numeri interi, di cui solo k (con $k \ll n$) sono non nulli e gli altri sono quindi uguali a 0

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

Vogliamo memorizzarla in modo efficiente

... Efficienza spaziale ...

Soluzione 1 – (standard, poco utile)

Memorizziamo tutti gli elementi, uno dopo l'altro, riga dopo riga, occupando così n^2 unità di spazio (celle di memoria)

Soluzione 2 – (migliore)

Memorizziamo il valore dominante – lo 0 - e le dimensioni della matrice; quindi memorizziamo solo i valori dei k elementi non nulli con le rispettive posizioni di riga e colonna, occupando quindi solo $3(k + 1)$ unità di spazio (celle di memoria)

... Efficienza spaziale

La soluzione 2 è quella più efficiente in termini di occupazione di spazio se

$$3(k + 1) < n^2$$

cioè quando il numero k di elementi non nulli è meno di un terzo del numero totale degli elementi

Ma, attenzione comunque alla possibile diminuzione dell'efficienza temporale delle operazioni matriciali con questo secondo tipo di rappresentazione

Problemi intrattabili

ATTENZIONE !!!

*Esistono dei problemi che, pur rientrando nel mondo del decidibile, per i quali quindi è possibile individuare un algoritmo risolutivo, ammettono **soluzioni non effettive** cioè con un costo temporale e/o spaziale insostenibile in pratica*

Un esempio . . .

Supponiamo di dover assemblare in orbita un apparato costituito da n parti componenti di peso rispettivo p_1, p_2, \dots, p_n , potendo usufruire di missili con portata massima P , con $p_i < P$ e $\sum p_i \gg P$, si tratta di minimizzare il numero di missioni da compiere

Prendiamo il caso $n = 50$, dobbiamo verificare il peso totale trasportabile in una missione sommando i pesi delle parti componenti a due a due, a tre a tre, e così via . . .

Un esempio . . . su cui riflettere

Compiamo quindi 2^{50} operazioni di confronto con **P** e se ciascuna costa **1 ms** il tempo totale sarà di circa **40.000 anni**

Con le attuali tecnologie potremmo ridurre il tempo del confronto a **1 μ s** , ma il tempo totale sarà comunque di circa **40 anni**

COSTO ESPONENZIALE
rispetto a n , cioè dell'ordine di 2^n

Riassumendo . . . (fino a questo punto)

Per delegare ad un esecutore (automatico) la soluzione di un problema, a partire dalla **struttura astratta** di appartenenza degli oggetti da trattare (manipolare), si deve perciò:

- ❑ **individuare una rappresentazione della struttura: degli oggetti e delle operazioni**
- ❑ **individuare una rappresentazione-descrizione dell' algoritmo, come sequenza di passi elementari**

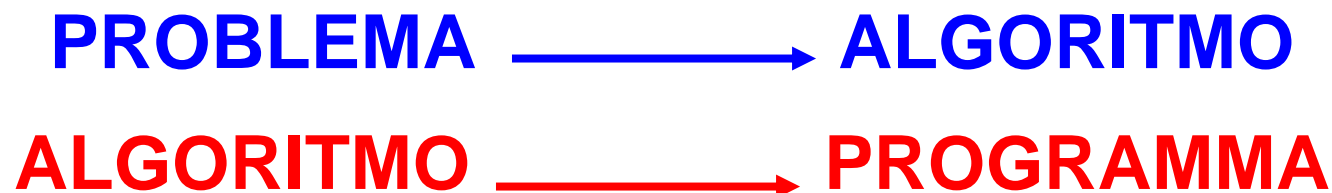
Rappresentazione

Per tali rappresentazioni si fa uso di un opportuno formalismo espressivo o **linguaggio**

Nel caso di un esecutore automatico (**elaboratore**) si fa uso di un adeguato **linguaggio di programmazione** composto di regole comprensibili sia per noi che per l'elaboratore

Definizioni

- ❑ **Dati di ingresso (INPUT)** - le rappresentazioni (fornite all'elaboratore) delle informazioni a disposizione
- ❑ **Programma** - la rappresentazione dell'algoritmo nel linguaggio di programmazione scelto (un programma sarà costituito da un insieme di istruzioni)
- ❑ **Dati di uscita (OUTPUT)** - le rappresentazioni fornite dall'elaboratore al termine della esecuzione del programma



Come intendiamo un Elaboratore

- ❑ L'elaboratore è un **esecutore** di operazioni in sequenza che trasformano input in output
- ❑ Funziona secondo **regole precise** e conosce un numero (molto) limitato di **comandi** (istruzioni)
- ❑ Esegue ciascun comando in **modo univoco** e in **tempo finito**
- ❑ Esegue **azioni** elementari (in numero limitato) su **dati** (che sono rappresentazioni di oggetti astratti) eseguendo **istruzioni**
- ❑ Le azioni procurano **effetti** di cambiamento di **stato**

Programma e Processo

- ❑ Un programma è una sequenza di istruzioni
- ❑ Un processo (di esecuzione di un programma) è una sequenza di azioni (che sono l'esecuzione delle istruzioni del programma)
- ❑ Un processo fa passare l'elaboratore da uno stato iniziale ad uno stato finale

Un esempio di esecutore

Consideriamo un esecutore - denominato

Sommatore - in grado di eseguire operazioni di somma di due numeri interi

Il **Sommatore** **riceve in ingresso** due numeri interi e li memorizza in due celle di memoria denominate **x** e **y**

Produce il **valore della loro somma** nella cella di memoria denominata **z** e fornisce in uscita questo valore calcolato

Le successive diapositive descrivono il comportamento del Sommatore

Sommatore

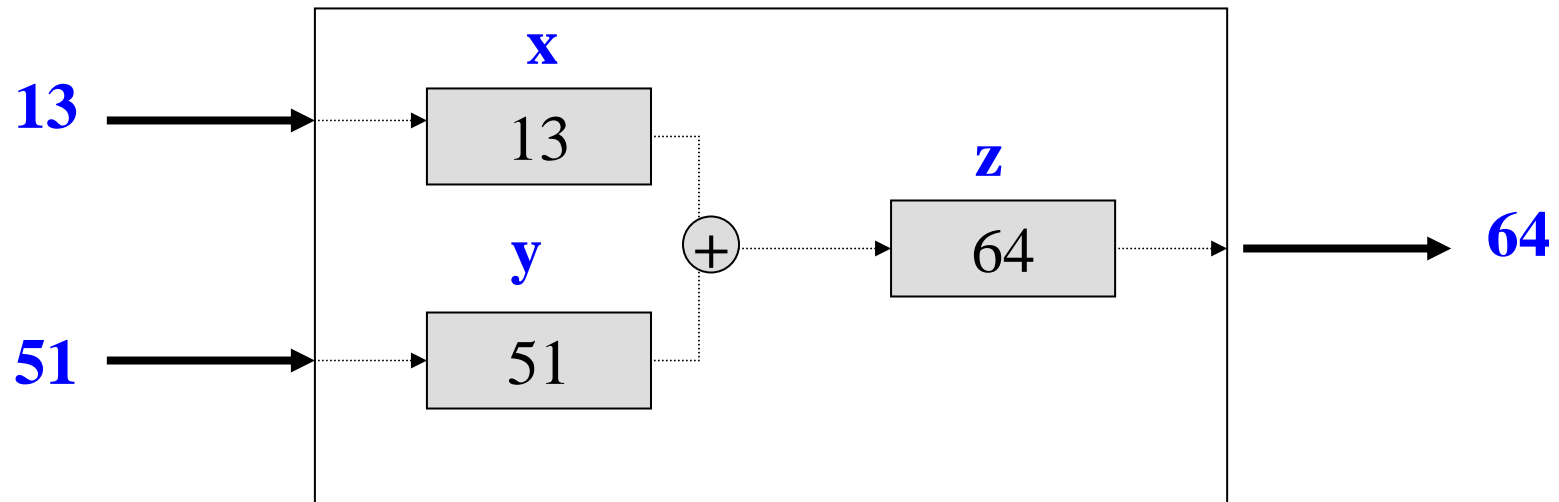
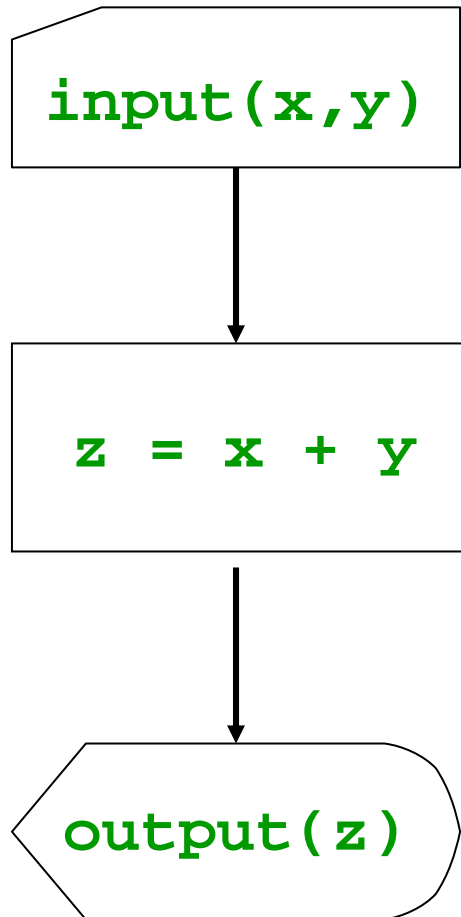


Diagramma a blocchi



Cosa è una variabile . . .

- ❑ Dal **punto di vista logico** una variabile è un simbolo (un **nome**) che **identifica** la rappresentazione di un oggetto di interesse (di tipo fissato, ad esempio un intero)
- ❑ Dal **punto di vista fisico** una variabile può essere intesa come un **contenitore** (cella di memoria) per valori di un oggetto di interesse (di tipo fissato)

. . . Cosa è una variabile

- ❑ Nella sua definizione ad ogni variabile viene quindi univocamente associato un **nome, imm modificabile**
- ❑ Ad ogni variabile viene anche associato un **tipo** che caratterizza l'insieme dei valori che essa può assumere nel tempo
- ❑ Variabili diverse **devono** avere nomi diversi, ma evidentemente possono avere lo stesso tipo e valori uguali

Cosa è una assegnazione

Ad una variabile di nome **alfa** possiamo assegnare come valore una costante, ad esempio **137**, oppure **483**, con una assegnazione denotata dal simbolo =

alfa = 137 oppure **alfa = 483**

Ad una variabile possiamo assegnare anche il valore di un'altra variabile, ad esempio

beta = alfa

In una assegnazione la parte sinistra (rispetto al simbolo =) è la variabile ricevente o destinataria del valore che è invece la parte destra, che può essere anche una qualunque espressione da calcolare, come nell'esempio **z = x + y**

Soluzioni di problemi ed esecutori

Come già detto la soluzione di un problema è determinata anche sulla base delle capacità operative dell'esecutore scelto per la soluzione automatica

- ❑ Se l'esecutore ha **elevate capacità** la soluzione è spesso molto semplificata: è come rivolgersi ad un **esperto al quale è sufficiente descrivere il problema per avere la soluzione**
- ❑ Se l'esecutore ha **scarse capacità** la soluzione deve essere descritta con molti dettagli, richiedendo all'esecutore di eseguire passo dopo passo semplici azioni a lui note

Un altro problema . . .

Determinare il prodotto di due numeri interi positivi

Enunciato informale in linguaggio naturale

Determinare $Z = X \cdot Y$, dove $X, Y, Z \in \mathbb{N}^+$

Formalizzazione in linguaggio naturale e matematico

... Un altro problema ...

Se l'esecutore scelto per la soluzione del problema opera sulla seguente rappresentazione della **struttura matematica** degli oggetti (numeri naturali) da trattare:

$$\mathbf{NAT} = \langle \mathbf{N}_{10}, +, -, *, <, =, 0, 1 \rangle$$

cioè se l'esecutore sa eseguire un prodotto, allora il problema è risolto dall'esecuzione della seguente istruzione, scritta in un formalismo, **linguaggio di programmazione**, che supponiamo comprensibile al nostro esecutore

$$\mathbf{z} = \mathbf{x} * \mathbf{y}$$

. . . Un altro problema . . .

In questo linguaggio utilizziamo le **variabili**, nell'accezione precedentemente definita, **x**, **y** e **z**, come denotazione di valori che rappresentano i numeri naturali del problema

$$\mathbf{x} = \text{Rapp}(X) , \mathbf{y} = \text{Rapp}(Y) , \mathbf{z} = \text{Rapp}(Z)$$

il **simbolo** ***** per denotare l'operazione di prodotto e l'istruzione di assegnazione, denotata dal **simbolo** **=**, per assegnare alla variabile **z** il valore ottenuto dall'operazione di prodotto

$$\mathbf{z} = \mathbf{x} * \mathbf{y}$$

Un altro problema ancora . . .

Qualora invece l'esecutore scelto per la soluzione del problema operi su una diversa rappresentazione della struttura degli oggetti da trattare (meno espressiva e potente - **non sa eseguire il prodotto, ma solo somma e sottrazione**) , quale:

$$\mathbf{NAT} = \langle \mathbf{N}_{10}, +, -, <, =, 0 \rangle$$

allora è necessario definire un algoritmo risolutivo basato sull'applicazione di una opportuna sequenza di operazioni elementari tra quelle possibili (eseguibili dall'esecutore)

. . . e ancora

Teorema:

Siano $x, y \in \mathbb{N}^+$

Se $y = 1$ allora $x \cdot y = x$

Altrimenti $x \cdot y = x + (x \cdot (y - 1))$

Cioè per avere il valore z del prodotto di x e y posso sommare x a sé stesso tante volte quanto indicato dal valore di y

$$z = x + (x + (x + \dots (x \cdot (y - (y - 1))) \dots))$$

$$z = x + x + \dots + x$$

In pratica

Inizialmente il valore di **z** è **0**, quindi, verificando passo dopo passo che il valore di **y** resti **maggiore di 0**, ripeto queste due operazioni:

- **incremento il valore corrente di z con x**
- **decremento di 1 il valore di y**

Algoritmo per il prodotto

1. $z = 0$

2. mentre $y > 0$

2.1. incrementa z di x

2.2. decrementa y di 1

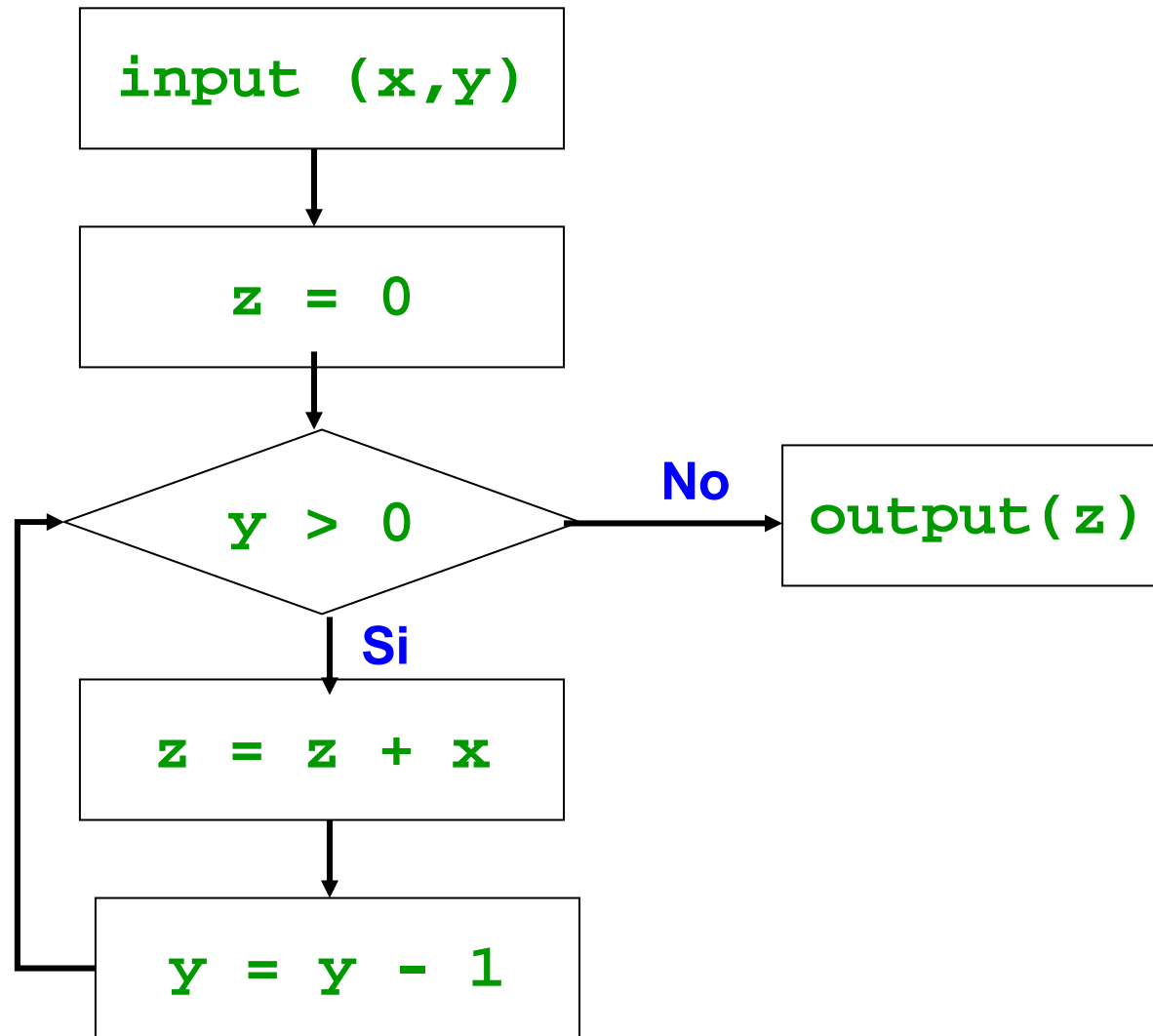
Descrizione dell'algoritmo

Pseudo-programma per il prodotto

Programma:	Prodotto
Variabili:	integer x, y, z;
Istruzioni:	1. input (x, y); 2. z = 0; 3. while (y > 0) { 4. z = z + x; 5. y = y - 1; 6. } 7. output (z);

***Descrizione dell'algoritmo in un linguaggio
(di programmazione)***

Diagramma a blocchi



Traccia del processo di esecuzione

Istruzione	X	Y	Z
1.	<u>17</u>	<u>3</u>	
2.	17	3	0
3.	17	3	0
4.	17	3	17
5.	17	2	17
3.	17	2	17
4.	17	2	34
5.	17	1	34
3.	17	1	34
4.	17	1	51
5.	17	0	51
3.	17	0	51
6.	<u>17</u>	<u>0</u>	<u>51</u>

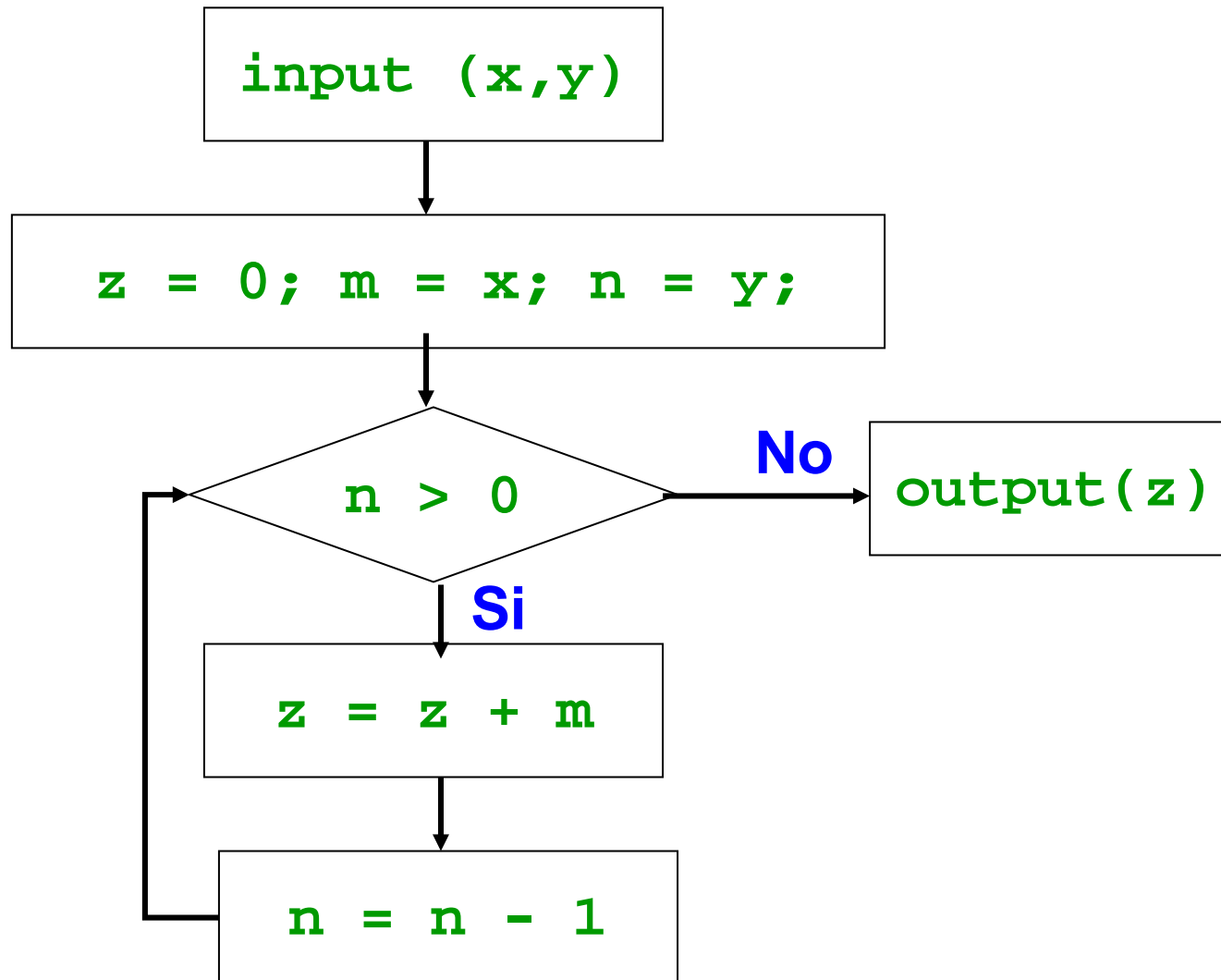
Una prima riflessione

Si dovrebbe poter verificare se il risultato è corretto, cioè se il valore finale di **z** sia o meno effettivamente il prodotto dei valori di **x** e di **y**

Al termine dell'elaborazione tuttavia il valore di **y** è nullo, si è cioè perduto il suo valore iniziale, determinando una impossibilità di verifica del risultato

Si può ovviare a questa carenza !!!

Diagramma a blocchi



Correttezza . . .

Prodotto

```
integer x, y, m, n, z;  
input (x, y);  
m = x;  
n = y;  
z = 0;  
while (n > 0) {  
    z = z + m;  
    n = n - 1;  
}  
output (z);
```

... Correttezza

	x	y	m	n	z	
1.	<u>17</u>	<u>3</u>	17	3		
2.	17	3	17	3	0	$x*y = z+m*n$
3.	17	3	17	3	0	
4.	17	3	17	3	17	
5.	17	3	17	2	17	$x*y = z+m*n$
3.	17	3	17	2	17	
4.	17	3	17	2	34	
5.	17	3	17	1	34	$x*y = z+m*n$
3.	17	3	17	1	34	
4.	17	3	17	1	51	
5.	17	3	17	0	51	$x*y = z+m*n$
3.	17	3	17	0	51	
6.	<u>17</u>	<u>3</u>	17	0	<u>51</u>	$x*y = z+m*n$

Un'altra riflessione

In questo caso il problema della correttezza può essere gestito tenendo anche conto della possibilità di determinare la **validità della seguente proprietà** durante l'esecuzione del programma

$$x * y = z + m * n$$

*Questo tipo di proprietà è detto
invariante di ciclo*

Efficienza

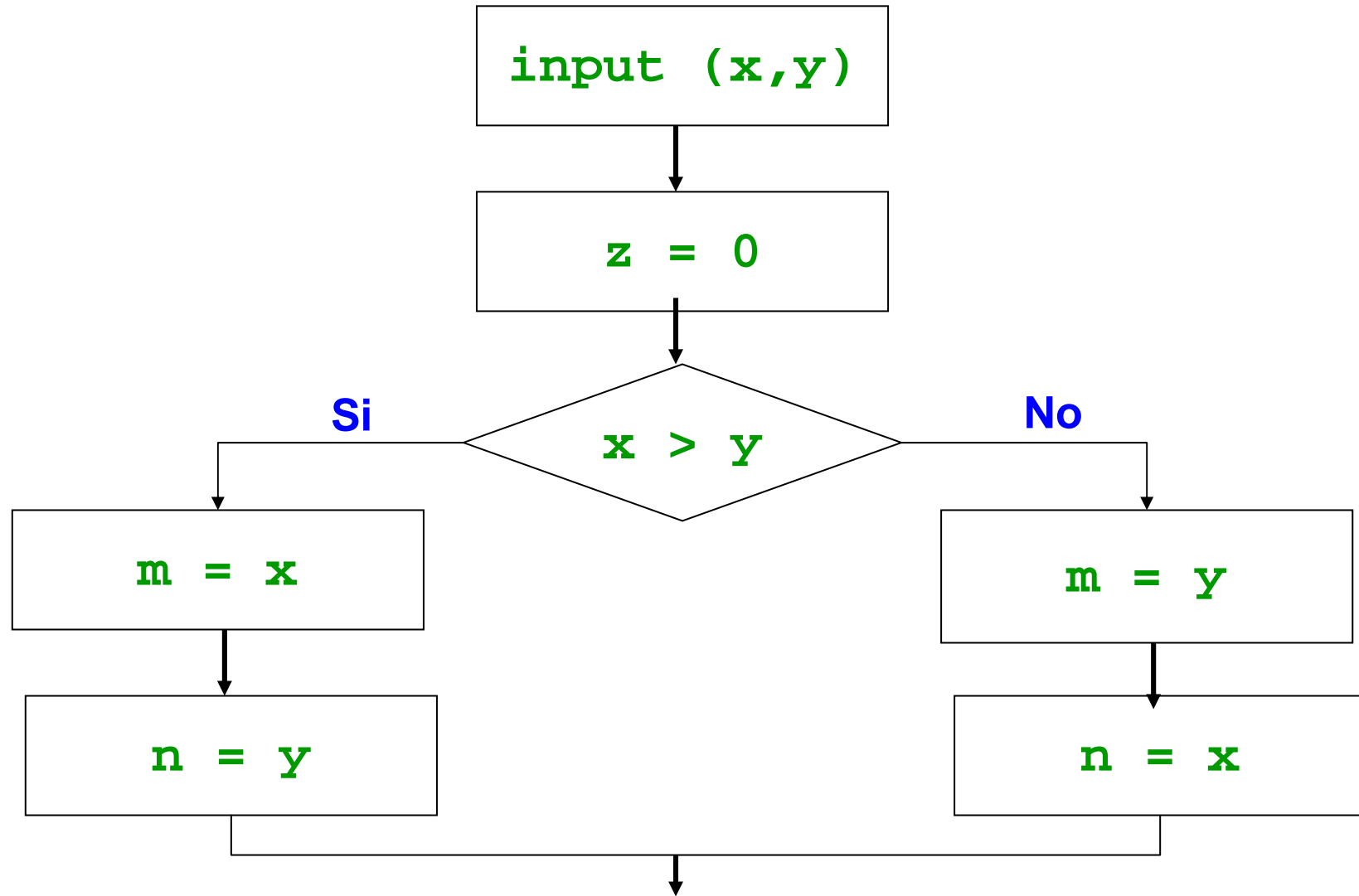
Ragioniamo ora sugli aspetti di efficienza della soluzione fin qui adottata

Il **numero di addizioni complessive** che vengono eseguite è pari ad **n** cioè a **y**, che nel nostro esempio vale **3**

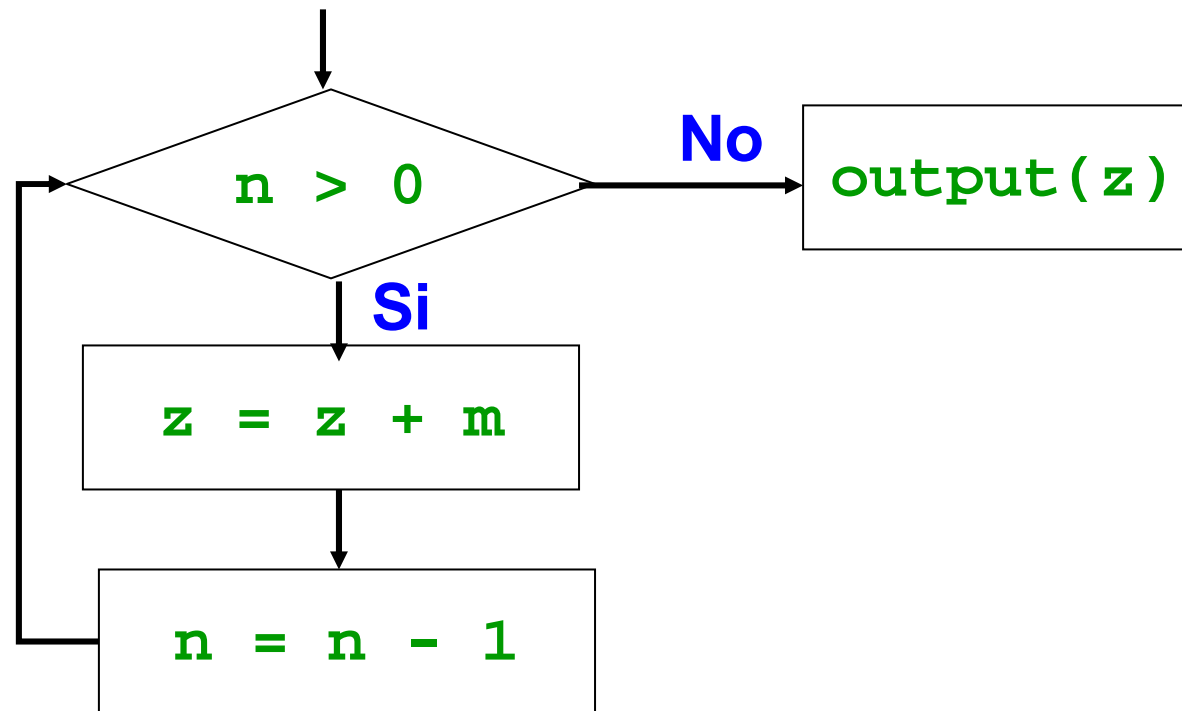
Pensiamo allora al caso di moltiplicare **3** per **17** utilizzando il nostro programma

Forse dobbiamo fare delle modifiche . . .

Diagramma a blocchi . . .



... continua



Programma finale per il prodotto

Prodotto

```
integer x, y, m, n, z;  
input (x, y);  
if (x > y) {  
    m = x; n = y;  
} else {  
    m = y; n = x;  
};  
z = 0;  
while (n > 0) {  
    z = z + m;  
    n = n - 1;  
}  
output (z);
```

Commento

- Il problema posto richiede di determinare il prodotto di due numeri interi positivi
- Sono stati individuati 2 algoritmi che lo risolvono in modi diversi in relazione alle potenzialità dell'esecutore
- Per il secondo algoritmo abbiamo definito diversi programmi sui quali abbiamo via via ragionato sulla correttezza e sull'efficienza, pervenendo ad una versione finale da ritenere soddisfacente

Tipi di istruzioni

Nell'esempio si è fatto uso di diversi tipi di istruzioni:

❑ Input / Output

❑ Assegnazione

❑ Operazioni aritmetiche e logiche

❑ Controllo

a) Sequenza (Composizione)

b) Selezione (Alternativa)

c) Iterazione (Ciclo o Ripetizione)

Esempio del programma prodotto

Prodotto

```
integer x, y, m, n, z;  
input (x, y);  
if ( x > y ) {  
    m = x; n = y;  
} else {  
    m = y; n = x;  
};  
z = 0;  
while ( n > 0 ) {  
    z = z + m;  
    n = n - 1;  
}  
output (z);
```

intestazione dichiarazioni input/output assegnazione
op. aritmetiche op. logiche selezione iterazione

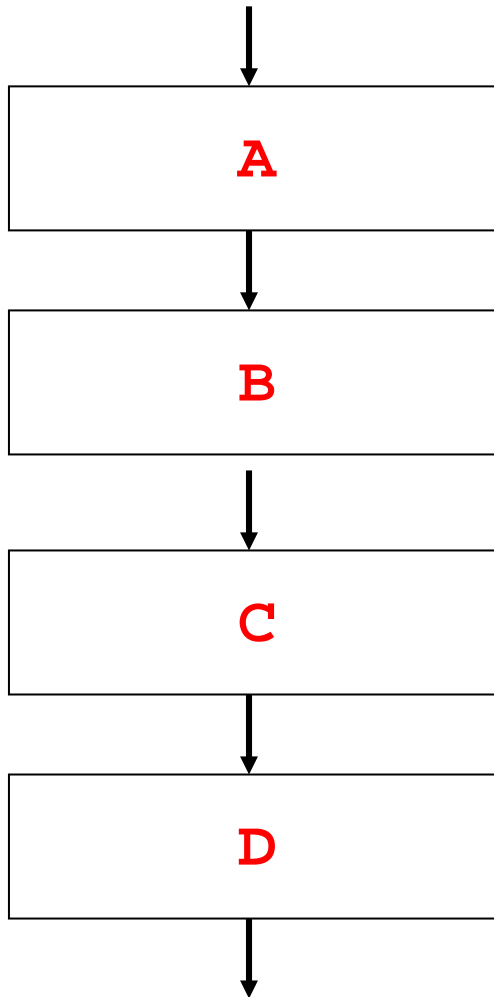
Strutture di controllo

Le strutture di controllo utilizzate nel nostro programma per il calcolo del prodotto di due numeri interi positivi sono

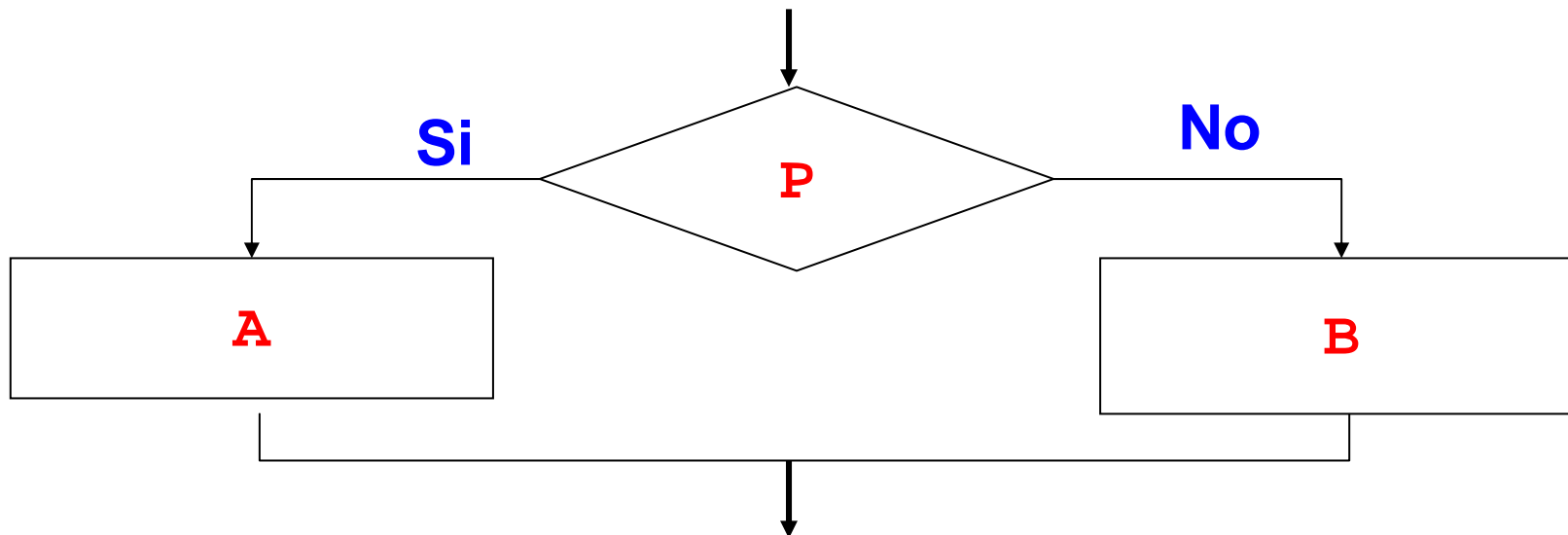
- ❑ Sequenza
- ❑ Alternativa
- ❑ Iterazione

Rivediamole analiticamente attraverso i rispettivi schemi diagrammatici

Sequenza (Composizione)

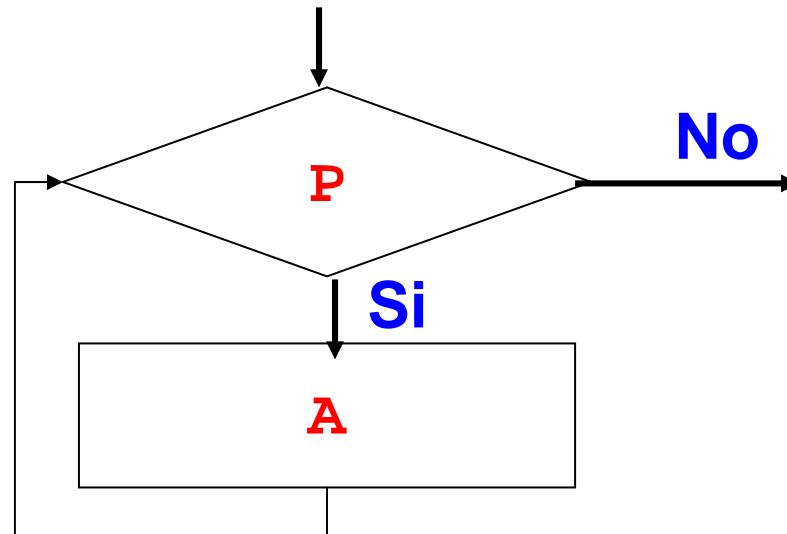


Selezione (Alternativa)



if *condizione-P* *istruzione-A* **else** *istruzione-B*

Iterazione (Ripetizione)



`while` *condizione-P* *istruzione-A*

Programmazione strutturata

Ciascuna delle strutture di controllo viste precedentemente gode della fondamentale proprietà di avere un flusso con una sola entrata ed una sola uscita; possiamo quindi **concatenarle** tra loro anche **annidandole a vari livelli**

“Qualsiasi programma può essere costruito utilizzando esclusivamente questi tre tipi di strutture di controllo”

Ovviamente c'è un teorema su tutto questo:
“Teorema di Böhm-Jacopini”

Riassumendo . . .

Gli esempi introdotti sono stati volutamente semplici ma ci consentono di trarre delle conclusioni generali

- Per ogni problema (decidibile) posso costruire, cioè sintetizzare, diversi algoritmi tra i quali sceglierò sulla base della loro efficienza**
- Per ogni algoritmo posso sintetizzare diversi programmi tra i quali sceglierò sulla base della loro efficienza avendone evidentemente verificata la correttezza**

Vedremo meglio cosa significa tutto questo e soprattutto come si fa

E ancora . . .

- Le soluzioni dei problemi le posso **costruire** (attraverso algoritmi e programmi) **astruendo** da ciò che conosco
 - ad esempio risolvo il problema del prodotto mediante l'iterazione della somma
 - similmente risolvo il problema dell'elevamento a potenza intera mediante l'iterazione del prodotto

Attenzione !!!

Non è quasi mai vero che la prima soluzione che viene in mente sia la migliore e soprattutto non dobbiamo pensare di automatizzare ciò che faremmo a mano : l'uomo e la macchina si comportano in modo diverso . . .

. . . e ancora

Riflettiamo su questo problema

“ Dati 100 numeri interi positivi di 4 cifre ciascuno, è più semplice (più rapido) calcolare la loro somma oppure determinare il valore massimo tra loro ? ”

E ancora . . .

“ Se per verificare come stanno le cose mi rivolgo ad un esecutore umano o ad un elaboratore ottengo la stessa risposta ? ”

Cosa abbiamo visto finora

- Cosa è un problema, e l'esempio del MCD
- Cosa è un algoritmo
- Cosa sono i problemi decidibili e non
- Cosa sono un programma e un processo
- Cosa è una variabile
- Che rapporto c'è tra un algoritmo e le potenzialità del suo esecutore
- In che consistono le proprietà di correttezza ed efficienza
- In cosa consiste la tecnica della programmazione strutturata