



# ELEMENTI DI INFORMATICA

DOCENTE: FRANCESCO MARRA

INGEGNERIA CHIMICA

INGEGNERIA ELETTRICA

SCIENZE ED INGEGNERIA DEI MATERIALI

INGEGNERIA GESTIONALE DELLA LOGISTICA E DELLA PRODUZIONE

INGEGNERIA NAVALE


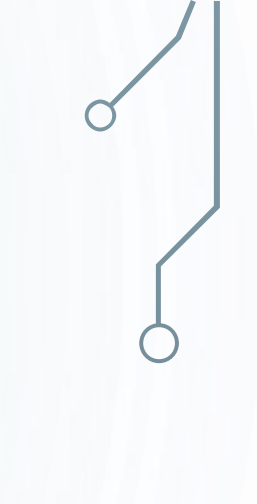
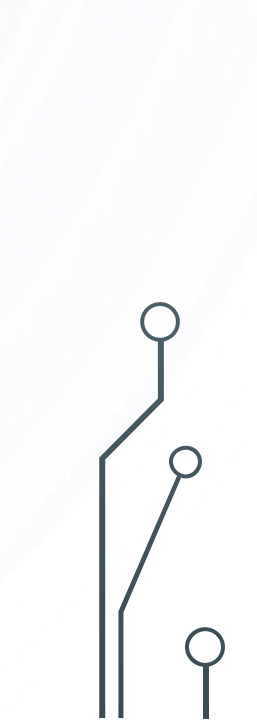
UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II  
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

# COMPLESSITÀ COMPUTAZIONALE E DIAGRAMMI DI FLUSSO





# AGENDA

- Complessità computazionale
  - La descrizione degli algoritmi
  - Diagrammi di flusso
- 
- 
- 

# CALCOLABILITÀ E TRATTABILITÀ

- **Calcolabilità**

- consente di dimostrare l'**esistenza** (ossia, la *decidibilità*) di un algoritmo risolvante un problema assegnato ed indipendente da qualsiasi automa
- classifica i problemi in **risolvibili** e **non risolvibili**
- un problema è risolvibile se esiste una Macchina di Turing in grado di fornire la soluzione al problema in tempo finito

- **Trattabilità**

- studia l'**eseguibilità** degli algoritmi risolvanti
- classifica i problemi in **facili** e **difficili**
- un problema è trattabile se esiste un algoritmo computazionale in grado di giungere alla soluzione in tempi di esecuzione accettabili e con un consumo di risorse (memoria) accettabile

# CALCOLABILITÀ E TRATTABILITÀ

- **Complessità computazionale**
  - studia i costi di esecuzione di un algoritmo risolutivo e come essi variano al crescere della dimensione del problema
  - mira a comprendere le prestazioni massime raggiungibili da un algoritmo risolutivo di un problema
  - corrisponde ad una misura delle **risorse di calcolo** consumate durante la computazione di un algoritmo
- Il concetto di trattabilità dipende da quello di complessità computazionale

# TIPI DI COMPLESSITÀ

- Esistono due tipi fondamentali, fra loro interdipendenti, di complessità computazionale
- **Complessità Spaziale**
  - misura la quantità di memoria necessaria alla rappresentazione dei dati richiesti dall'algoritmo per risolvere il problema
- **Complessità Temporale**
  - misura il tempo richiesto per produrre la soluzione maggiormente considerata in quanto i calcolatori hanno tipicamente molta memoria disponibile

# MISURE DI COMPLESSITÀ

- Le misure di complessità possono essere di due tipi fondamentali:
- *Statiche*
  - basate sulle caratteristiche **strutturali** (ad es. il numero di istruzioni) dell'algoritmo indipendenti dai dati di input su cui esso opera
- *Dinamiche*
  - dipendenti sia dalle caratteristiche **strutturali** dell'algoritmo che dai **dati di input** su cui esso opera

# COMPLESSITÀ E DATI DI INPUT

- Un primo fattore che incide sul tempo impiegato dall'algoritmo è la quantità di dati su cui lavora
- Il tempo di esecuzione di un algoritmo è tipicamente espresso come una funzione  $f(n)$  della dimensione  $n$  dei dati di input
  - ad esempio, un algoritmo ha un tempo di esecuzione  $n^2$  se il tempo impiegato aumenta in funzione del quadrato della dimensione dell'input
- Da sola la dimensione dei dati di input non basta
  - il tempo di esecuzione può dipendere anche dalla particolare **configurazione** che possono assumere i dati in input
  - possono esserci alcune condizioni di esecuzione che in funzione di particolari valori dei dati di ingresso, determinano diversi percorsi nell'algoritmo e quindi diverse quantità di dati da elaborare



# TEMPO DI ESECUZIONE

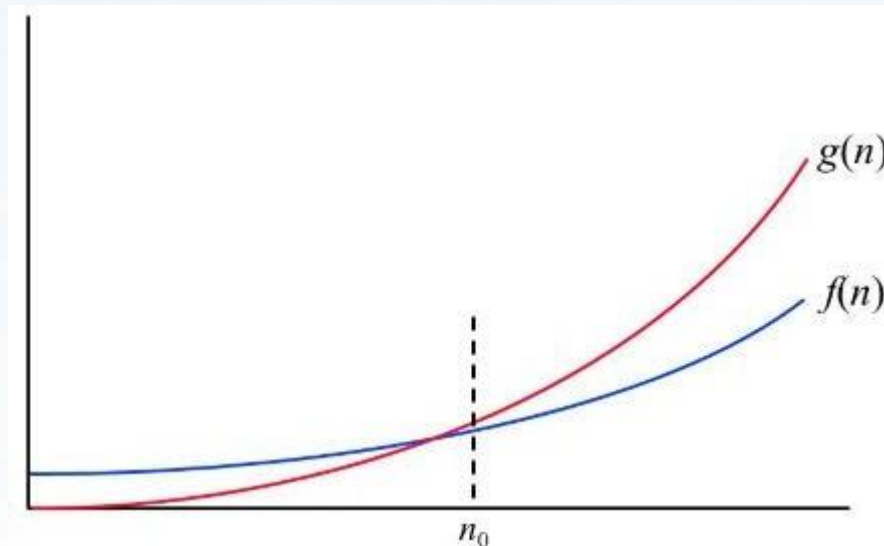
- *Analisi del caso migliore*
  - per calcolare il tempo di esecuzione quando la configurazione dei dati presenta difficoltà minime di trattamento
- *Analisi del caso peggiore*
  - per calcolare il tempo di esecuzione quando la configurazione dei dati presenta difficoltà massime di trattamento
    - si tratta di un'analisi molto utile, perché fornisce delle garanzie sul tempo massimo che l'algoritmo può impiegare
- *Analisi del caso medio*
  - per calcolare il tempo di esecuzione quando la configurazione presenta difficoltà medie di trattamento

# COMPLESSITÀ ASINTOTICA

- Quando si analizza la complessità computazionale di un algoritmo si valuta come i tempi di esecuzione crescono al crescere della dimensione dei dati
- In pratica si valuta come l'ordine di grandezza del tempo di esecuzione  $f(n)$  cresca al limite (tenda ai suoi asintoti)
  - per dimensioni dell'input sufficientemente grandi
  - si trascurano operazioni non significative concentrando l'attenzione solo su quelle predominanti
- La complessità asintotica dipende solo dall'algoritmo
  - la complessità esatta, invece, dipende da tanti fattori legati alla esecuzione dell'algoritmo

# COMPLESSITÀ ASINTOTICA

- Dati due algoritmi diversi che risolvono lo stesso problema e presentano due diverse complessità  $f(n)$  e  $g(n)$ 
  - se  $f(n)$  è asintoticamente inferiore a  $g(n)$
  - allora esiste una dimensione dell'input  $n_0$  oltre la quale l'ordine di grandezza del tempo di esecuzione del primo algoritmo è inferiore all'ordine di grandezza del tempo di esecuzione del secondo



# ESEMPI DI COMPLESSITÀ TEMPORALE

- Le complessità asintotiche più diffuse sono la logaritmica, la polinomiale, e l'esponenziale (più trattabile la prima)
  - Complessità logaritmica
    - $f(n) = \log n, \dots$
  - Complessità polinomiale
    - $f(n) = n, f(n) = n^2, f(n) = n^3, f(n) = n^5, \dots$
  - Complessità esponenziale (o, in generale, Non Polinomiale, NP)
    - $f(n) = 2^n, f(n) = 3^n, \dots$

	10	20	30	40	50
$n$	0,00001 sec	0,00002 sec	0,00003 sec	0,00004 sec	0,00005 sec
$n^2$	0,0001 sec	0,0004 sec	0,0009 sec	0,0016 sec	0,0025 sec
$n^3$	0,001 sec	0,008 sec	0,027 sec	0,064 sec	0,125 sec
$n^5$	0,1 sec	3,2 sec	24,3 sec	1,7 min	5,2 min
$2^n$	0,001 sec	1,0 sec	17,9 min	12,7 giorni	35,7 anni
$3^n$	0,059 sec	58 min	6,5 anni	3,855 secoli	200.000.000 secoli

# ESEMPIO: TROVARE IL MINIMO

- Trovare il minimo  $m$  in un insieme di  $n$  numeri  $\{x_1, x_2, \dots, x_n\}$
- Possibile algoritmo risolutivo
  - considero  $x_1$  come primo candidato ad essere il minimo  $m$
  - confronto  $m$  con gli elementi  $x_2, \dots, x_n$ 
    - ogni volta che trovo un elemento  $x_i < m \rightarrow$  agguirno il valore temporaneo del minimo  $m = x_i$
  - al termine dei confronti, il valore attuale di  $m$  è il minimo effettivo
- Complessità dell'algoritmo risolutivo  $\rightarrow f(n) = n$ 
  - complessivamente sono stati eseguiti  $n-1$  confronti, che asintoticamente tende a  $n$
  - l'efficienza dell'algoritmo è quindi direttamente proporzionale alla dimensione  $n$  dei dati di input

# ESEMPIO: ORDINAMENTO

- Disporre in ordine crescente un insieme di  $n$  numeri  $\{x_1, x_2, \dots, x_n\}$
- Possibile algoritmo risolutivo
  - per ogni valore di un indice  $i = 1, 2, \dots, n$  ripetere le seguenti operazioni
    - trovare l'elemento più piccolo nel sottoinsieme  $\{x_i, x_{i+1}, \dots, x_n\}$
    - portare il minimo locale  $m$  in prima posizione, scambiando  $m$  con l'elemento  $x_i$
- Complessità dell'algoritmo risolutivo  $\rightarrow f(n) = n^2$ 
  - per  $n-1$  volte si cerca il minimo su  $n-1$  insiemi diversi sempre più piccoli
  - poi si posta il minimo trova nella prima posizione dell'insieme considerato
  - si eseguono  $n + n - 1 + n - 2 + \dots + 2 \approx n^2$  operazioni

# ESEMPIO: ORDINAMENTO

- Analisi del caso migliore
  - il caso ottimo per l'algoritmo è quello in cui la sequenza di partenza sia già ordinata, poiché non bisogna effettuare alcuno scambio
- Analisi del caso peggiore
  - il caso pessimo è invece quello in cui la sequenza di partenza sia ordinata al contrario, poiché ad ogni iterazione si dovrà effettuare uno scambio

# DESCRIZIONE DEGLI ALGORITMI

- Passi per la risoluzione di un problema
  - capire preliminarmente se il problema ammette soluzioni
  - nel caso ne ammetta, individuare un metodo risolutivo (algoritmo)
  - esprimere tale metodo risolutivo in un linguaggio comprensibile all'esecutore a cui è rivolto
- In un algoritmo, tipicamente, si possono individuare due classi fondamentali di frasi del linguaggio
  - quelle che descrivono l'esecuzione di determinate operazioni (istruzioni)
  - quelle che indicano all'esecutore l'ordine in cui tali operazioni devono essere eseguite (strutture di controllo)



# ESEMPIO: CORREZIONE DEGLI ERRORI LESSICALI

*La comme osservazione dela  
evoluzione del mondo degli  
elaboratori eletronici non deve trarre  
in ingano. Difati a dispeto  
del'evoluzione tecnologica, l'attivit   
di programmazione di propone come  
un'arte che   ben vero che muta, ma  
che non pu  non contenere i trati  
antropomorfi di chi genera i  
programmi e di chi li utilizza.*



*La comune osservazione della  
evoluzione del mondo degli  
elaboratori elettronici non deve trarre  
in inganno. Difatti a dispetto  
dell'evoluzione tecnologica, l'attivit   
di programmazione si propone come  
un'arte che   ben vero che muta, ma  
che non pu  non contenere i tratti  
antropomorfi di chi genera i  
programmi e di chi li utilizza.*

## ESEMPIO: CORREZIONE DEGLI ERRORI LESSICALI

- 1 leggi un rigo del testo;
- 2 **per** ogni parola del rigo  
**fai:**
- 3     **se** conosci la parola,  
      **allora** controlla come è scritta  
      **altrimenti fai** una ricerca nel vocabolario
- 4     **se** la parola non è riportata in modo corretto  
      **allora** correggila,
- 5     riscrivi la parola nel testo corretto;
- 6 **ripeti** le azioni da 1) a 5)  
   **fino** alla terminazione dei rigi del testo.

# ISTRUZIONI

- ***Elementari***
  - Istruzioni che l'esecutore è in grado di comprendere ed eseguire
- ***Non elementari***
  - Istruzioni non note all'esecutore, la cui specifica dev'essere comunque resa disponibile all'esecutore per eseguire correttamente l'algoritmo
  - consentono di descrivere l'algoritmo con un repertorio di istruzioni più ricco delle istruzioni elementari note all'esecutore
  - la specifica di un'istruzione non elementare tipicamente trasforma l'istruzione stessa in un insieme di istruzioni elementari

# STRUTTURE DI CONTROLLO

- ***Costrutti di sequenza***
  - specificano azioni devono essere svolte una dopo l'altra
- ***Costrutti di selezione***
  - definiscono azioni che devono essere svolte solo se si verificano determinate **condizioni**
- ***Costrutti iterativi***
  - definiscono azioni devono essere **ripetute** un numero di volte prestabilito o determinato dal verificarsi di certe condizioni

# CARATTERISTICHE DELLE OPERAZIONI DI UN ALGORITMO

- *Finitezza*
  - devono avere termine entro un intervallo di tempo finito dall'inizio della loro esecuzione
- *Descrivibilità*
  - devono produrre, se eseguite, degli effetti descrivibili per esempio fotografando lo stato degli oggetti coinvolti sia prima che dopo l'esecuzione dell'operazione
- *Riproducibilità*
  - devono produrre lo stesso effetto ogni volta che vengono eseguite nelle stesse condizioni iniziali
- *Comprensibilità*
  - devono essere espresse in una forma comprensibile all'esecutore

# ESECUZIONE DI UN ALGORITMO

- L'esecuzione di un algoritmo da parte di un esecutore si traduce in una successione di azioni che vengono effettuate nel tempo
  - evoca un processo sequenziale
  - serie di eventi che occorrono uno dopo l'altro, ciascuno con un inizio e una fine identificabili
- Si definisce **Sequenza di esecuzione** la descrizione del processo sequenziale
  - elenco di tutte le istruzioni eseguite, nell'ordine di esecuzione
    - detta anche *sequenza dinamica*
  - un algoritmo può prescrivere più di una sequenza di esecuzione
  - se il processo descritto è ciclico, il numero di sequenze può essere infinito
    - processo che non ha mai termine

# ESEMPIO: CALCOLO DELLE RADICI DI UNA EQUAZIONE DI 2° GRADO

**Calcolo**

$$d = \sqrt{b^2 - 4ac}$$

**Calcolo la prima radice come**

$$x_1 = \frac{-b + d}{2a}$$

**Calcolo la seconda radice come**

$$x_2 = \frac{-b - d}{2a}$$

a) Unica sequenza di esecuzione

**Calcolo**

$$\Delta = b^2 - 4ac$$

**Se  $\Delta \geq 0$**

**Allora Calcolo**

$$d = \sqrt{\Delta}$$

**Calcolo la prima radice come**

$$x_1 = \frac{-b + d}{2a}$$

**Calcolo la seconda radice come**

$$x_2 = \frac{-b - d}{2a}$$

**Altrimenti Calcola radici complesse**

b) Più sequenze di esecuzione



# ESEMPIO: CALCOLO DELLE RADICI DI UNA EQUAZIONE DI 2° GRADO

Calcolo

$$\Delta = b^2 - 4ac$$

Se  $\Delta \geq 0$

Allora Calcolo

$$d = \sqrt{\Delta}$$

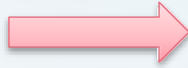
Calcolo la prima radice come

$$x_1 = \frac{-b + d}{2a}$$

Calcolo la seconda radice come

$$x_2 = \frac{-b - d}{2a}$$

Altrimenti Calcola radici complesse



Calcolo

$$\Delta = b^2 - 4ac$$

Verifico che  $\Delta \geq 0$  è risultata vera

Calcolo

$$d = \sqrt{\Delta}$$

Calcolo la prima radice come

$$x_1 = \frac{-b + d}{2a}$$

Calcolo la seconda radice come

$$x_2 = \frac{-b - d}{2a}$$

Calcolo

$$\Delta = b^2 - 4ac$$

Verifico che  $\Delta \geq 0$  è risultata non vera

Calcolo radici complesse

Controllo dell'esistenza di radici reali

Due sequenze di esecuzione  
dipendenti dai dati iniziali



# ESEMPIO: SOLITARIO DEL CARCERATO

**Fintantoché (il mazzo ha ancora carte) ripeti:**

**Mischia le carte**

**Prendi 4 carte dal mazzo e disponile sul tavolo di gioco**

**Se (le 4 carte hanno la stessa figura) allora levali dal tavolo di gioco**



**Mischiate le carte**

**Prese le 4 carte dal mazzo e messe sul tavolo di gioco**

**Verificato che (le 4 carte hanno la stessa figura) è risultata vera**

**Tolte le 4 carte dal tavolo di gioco**

**Sequenza di esecuzione 1) Si eliminano le carte al primo tentativo**

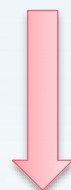
# ESEMPIO: SOLITARIO DEL CARCERATO

**Fintantoché (il mazzo ha ancora carte) ripeti:**

**Mischia le carte**

**Prendi 4 carte dal mazzo e disponile sul tavolo di gioco**

**Se (le 4 carte hanno la stessa figura) allora levali dal tavolo di gioco**



**Mischiate le carte**

**Prese le 4 carte dal mazzo e messe sul tavolo di gioco**

***Verificato che* (le 4 carte hanno la stessa figura) *è risultata falsa***

**Mischiate le carte**

**Prese le 4 carte dal mazzo e messe sul tavolo di gioco**

***Verificato che* (le 4 carte hanno la stessa figura) *è risultata vera***

**Tolte le 4 carte dal tavolo di gioco**

Sequenza di esecuzione 2) Si eliminano le carte al secondo tentativo

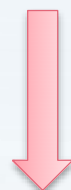
# ESEMPIO: SOLITARIO DEL CARCERATO

Fintantoché (il mazzo ha ancora carte) ripeti:

Mischia le carte

Prendi 4 carte dal mazzo e disponile sul tavolo di gioco

Se (le 4 carte hanno la stessa figura) allora levale dal tavolo di gioco



ripetuto  $n-1$  volte

Mischiate le carte

Prese le 4 carte dal mazzo e messe sul tavolo di gioco

*Verificato che* (le 4 carte hanno la stessa figura) *è risultata falsa*

Mischiate le carte

Prese le 4 carte dal mazzo e messe sul tavolo di gioco

*Verificato che* (le 4 carte hanno la stessa figura) *è risultata vera*

Tolte le 4 carte dal tavolo di gioco

Sequenza di esecuzione 3) Si eliminano le carte dopo  $n$  tentativi

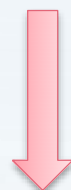
# ESEMPIO: SOLITARIO DEL CARCERATO

**Fintantoché (il mazzo ha ancora carte) ripeti:**

**Mischia le carte**

**Prendi 4 carte dal mazzo e disponile sul tavolo di gioco**

**Se (le 4 carte hanno la stessa figura) allora levali dal tavolo di gioco**



**ripetuto infinite volte**

**Mischiate le carte**

**Prese le 4 carte dal mazzo e messe sul tavolo di gioco**

**Verificato che (le 4 carte hanno la stessa figura) è risultata falsa**

**Sequenza di esecuzione 4) Non si trovano mai quattro carte uguali**

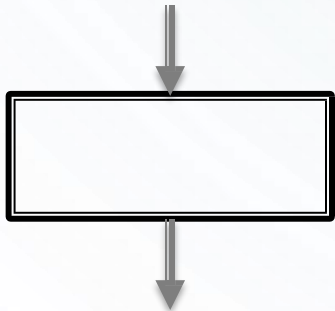
# SEQUENZA DI ESECUZIONE STATICA E DINAMICA

- In un algoritmo, la **sequenza statica** delle istruzioni descrive una pluralità di sequenze dinamiche differenti
  - il numero di sequenze dinamiche non è noto a priori e dipende dai dati da elaborare
- La valutazione sia del tipo che del numero delle sequenze dinamiche è di fondamentale importanza per valutare soluzioni diverse dello stesso problema, in modo da:
  - poter dire quale di esse presenta un tempo di esecuzione migliore
  - poter affermare se una soluzione ha terminazione o meno

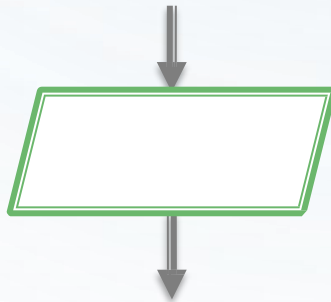
# DIAGRAMMI DI FLUSSO

- I **diagrammi di flusso** o **flow chart** sono un formalismo che consente di rappresentare graficamente gli algoritmi
  - descrivono le *azioni da eseguire* ed il loro *ordine di esecuzione*
- Ogni azione corrisponde ad un simbolo grafico (blocco)
  - ogni blocco ha un ramo in ingresso ed uno o più rami in uscita

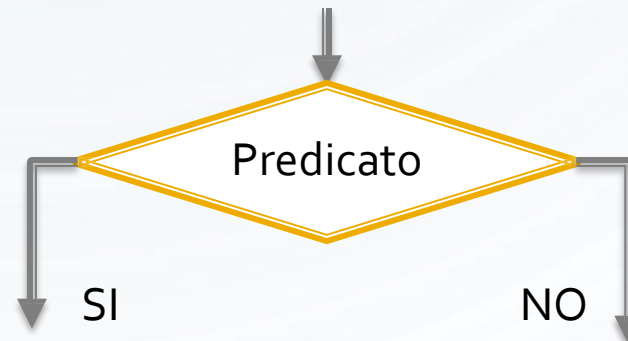
Elaborazione



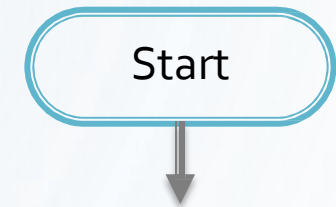
I/O



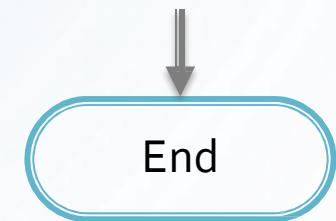
Selezione a due vie



INIZIO



Fine



# DIAGRAMMI DI FLUSSO: ESEMPIO

## SOMMA DI DUE NUMERI

- Definire un algoritmo che calcoli e stampi a video la somma di 2 numeri dati in ingresso

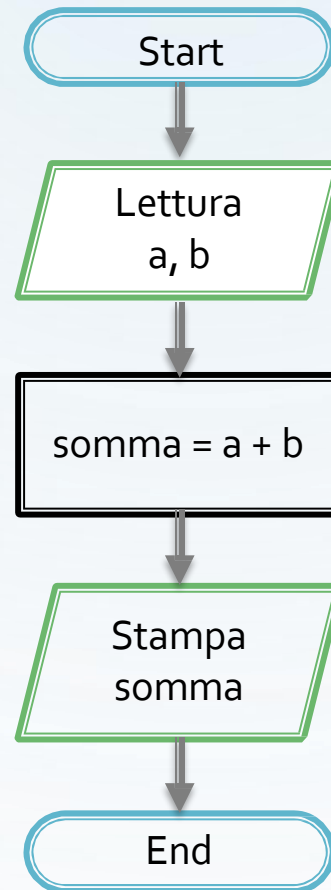


# DIAGRAMMI DI FLUSSO: ESEMPIO

## SOMMA DI DUE NUMERI

Blocco di elaborazione per fare calcoli e operazioni di assegnamento.

- il risultato del calcolo viene assegnato alla variabile di uscita che si chiama somma.



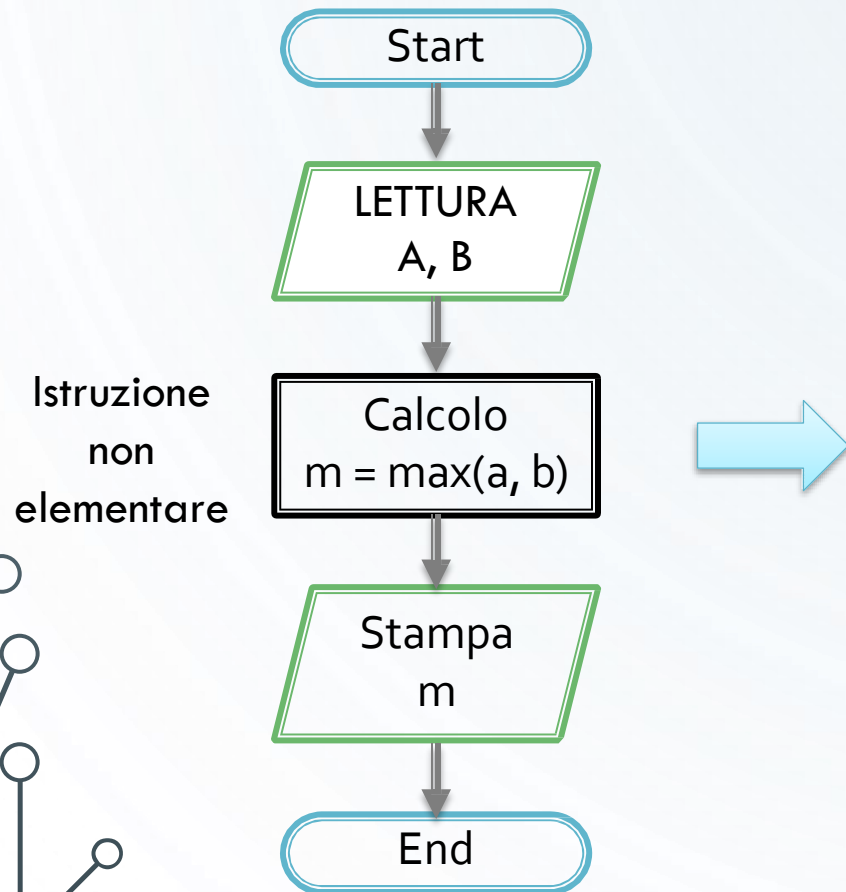
Uso il parallelogramma per effettuare operazioni di I/O.

- lettura in input di 2 numeri, e salvataggio dei valori in 2 variabili di ingresso che si chiamano a e b.

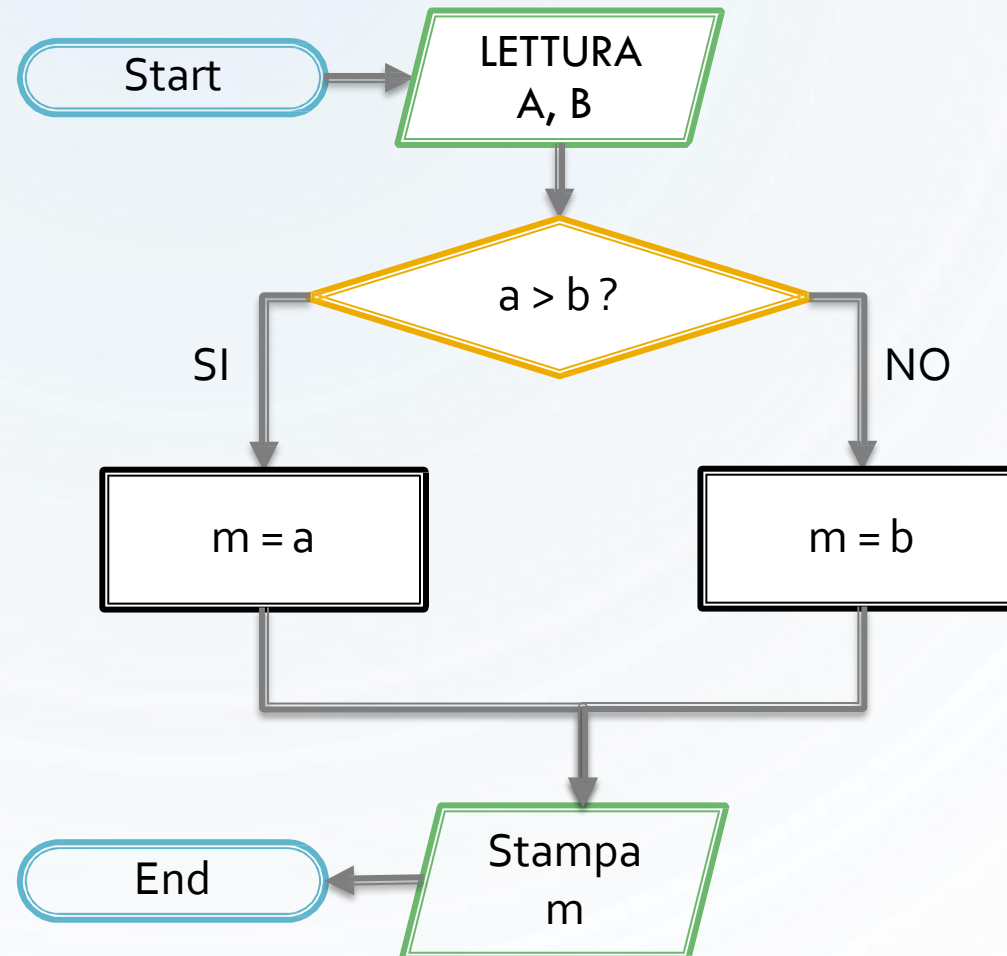
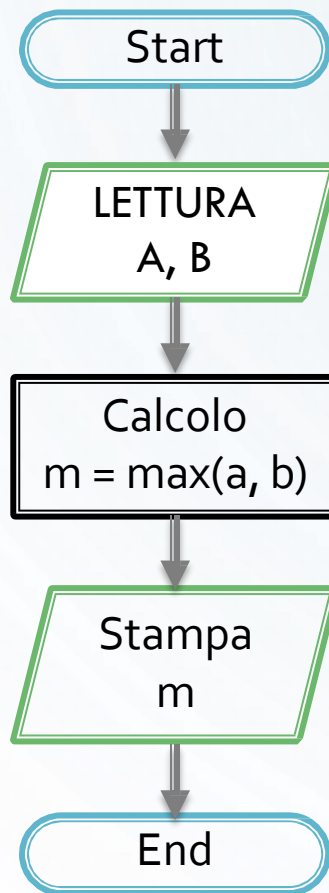
Stampa a video la variabile somma, quindi si utilizza il parallelogramma.



# DIAGRAMMI DI FLUSSO: ESEMPIO MASSIMO TRA DUE NUMERI

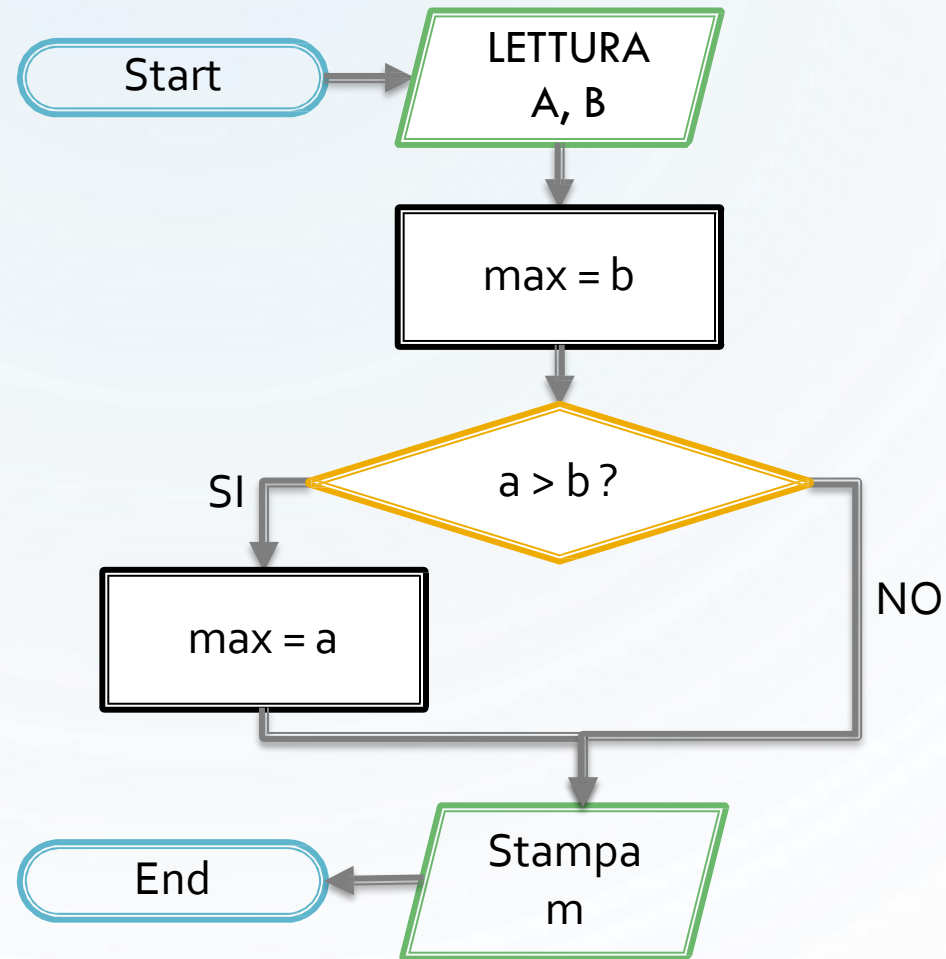
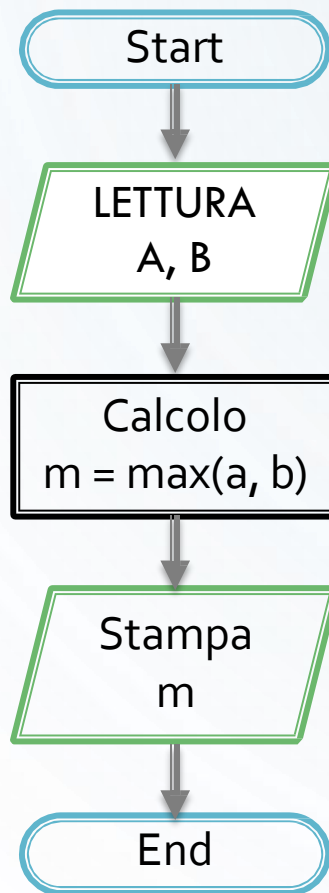


# DIAGRAMMI DI FLUSSO: ESEMPIO MASSIMO TRA DUE NUMERI



# DIAGRAMMI DI FLUSSO: ESEMPIO

## MASSIMO TRA DUE NUMERI (VARIANTE)



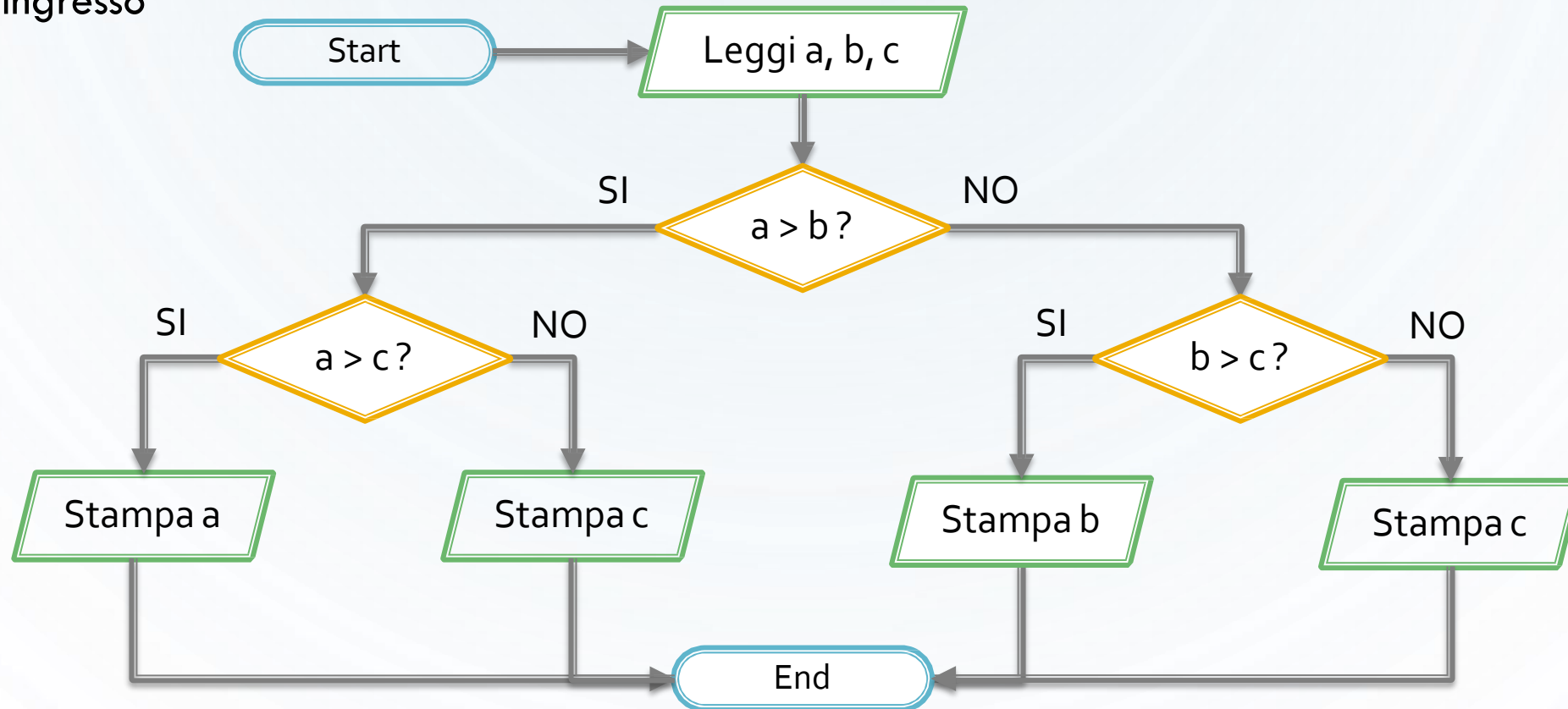
# DIAGRAMMI DI FLUSSO: ESERCIZIO MASSIMO TRA TRE NUMERI

- Progettare un algoritmo che calcoli e stampi a video il valore massimo di 3 numeri inseriti in ingresso



# DIAGRAMMI DI FLUSSO: ESERCIZIO MASSIMO TRA TRE NUMERI

- Progettare un algoritmo che calcoli e stampi a video il valore massimo di 3 numeri inseriti in ingresso



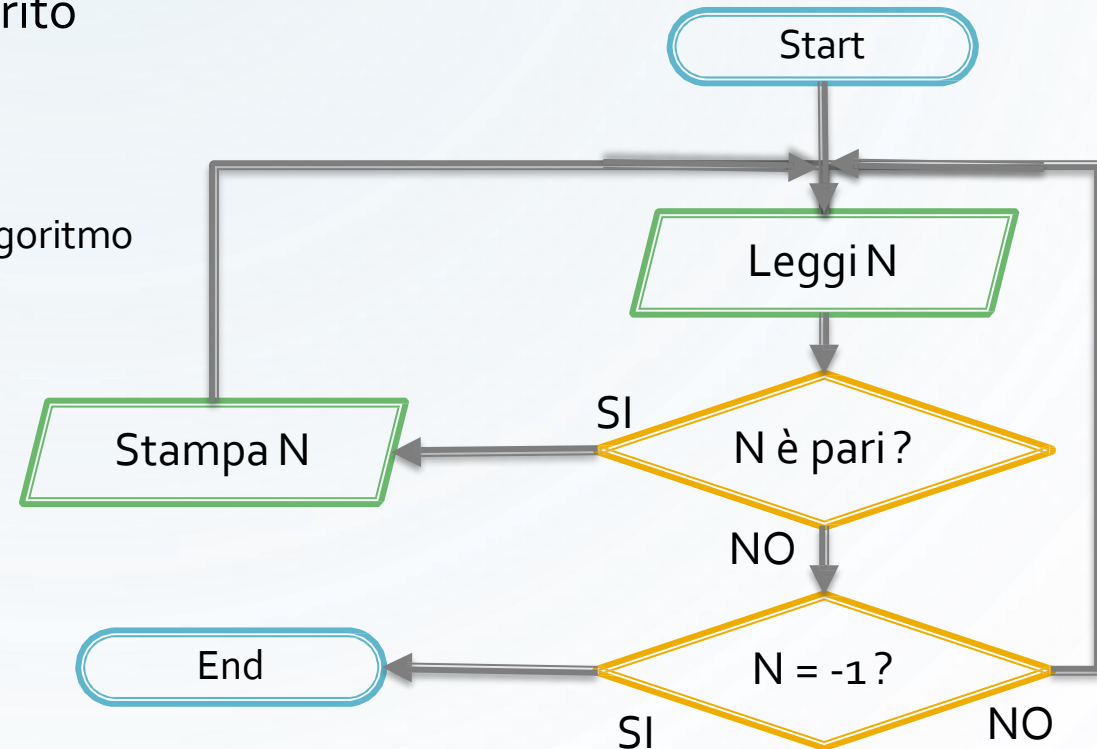
# DIAGRAMMI DI FLUSSO: ESERCIZIO

## STAMPA A VIDEO

- Definire un algoritmo che permetta di stampare il valore di un numero pari N fornito in ingresso dall'utente, e che termini quando l'utente inserisce -1.
  1. Leggere un numero da input
  2. Controllare il valore del numero inserito
    - se il numero è pari, stamparlo a video
    - se il numero è dispari tornare al punto 1
    - se il numero è uguale a "-1", terminare l'algoritmo

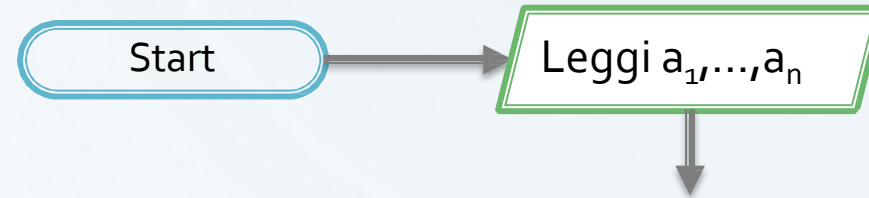
# DIAGRAMMI DI FLUSSO: ESERCIZIO STAMPA A VIDEO

1. Leggere un numero da input
2. Controllare il valore del numero inserito
  - se il numero è pari, stamparlo a video
  - se il numero è dispari tornare al punto 1
  - se il numero è uguale a "-1", terminare l'algoritmo



# DIAGRAMMI DI FLUSSO: ESERCIZIO MASSIMO TRA N NUMERI

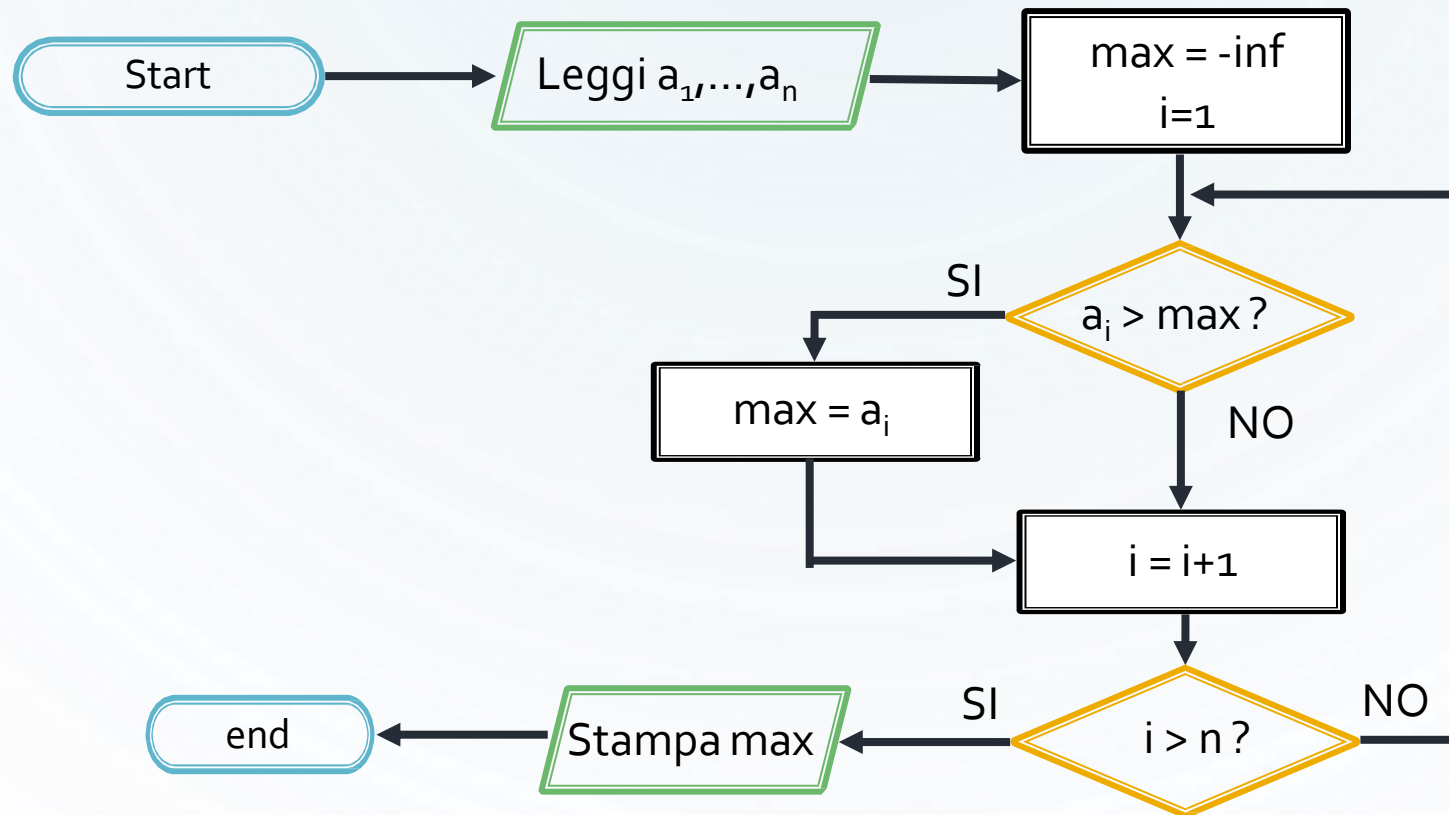
- Progettare un algoritmo che calcoli e stampi a video il valore massimo di  $n$  numeri inseriti in ingresso





# DIAGRAMMI DI FLUSSO: ESERCIZIO MASSIMO TRA N NUMERI

- Progettare un algoritmo che calcoli e stampi a video il valore massimo di  $n$  numeri inseriti in ingresso



**DOMANDE, DUBBI, PERPLESSITÀ**

