

Concetti di base del pensiero algoritmico: Sequenza, selezione e ripetizione

**Indice dei contenuti**

Introduzione 3

Che cos'è un algoritmo? 3

Sequenziamento 4

Selezione 6

Ripetizione 7

Pseudocodice 9

Problemi risolti 11

Riferimenti 11

Allegati 11

**Introduzione**

Il pensiero algoritmico è la capacità di risolvere i problemi definendo una sequenza chiara e definita di passaggi che, una volta seguiti fedelmente, portano a una soluzione. Durante la nostra vita scolastica, e in particolare durante gli anni della scuola primaria e secondaria, impariamo a risolvere i problemi attraverso un processo di apprendimento mnemonico e di imitazione delle soluzioni fornite dall'insegnante. Per esempio, sebbene molti studenti sappiano calcolare il massimo comune denominatore di due numeri, pochi sono in grado di articolare l'elenco delle operazioni necessarie per raggiungere il risultato.

Questo processo di definizione dei passaggi necessari per arrivare a una soluzione è estremamente importante perché, oltre a rendere possibile l'automazione dei compiti (ad es. attraverso la creazione di una macchina in grado di eseguire i passaggi indicati), porta l'individuo a una maggiore consapevolezza del problema, della sua soluzione e di conseguenza alla capacità di adattare la soluzione trovata a problemi simili o analoghi: per articolare passaggi chiari e ben definiti, bisogna innanzitutto comprendere il problema, le condizioni da cui si parte e quelle a cui si vuole arrivare, gli strumenti e le operazioni che ci permettono di modificare il sistema in modo programmato e prevedibile.

Sebbene i concetti di "algoritmo" e "pensiero algoritmico" siano spesso immediatamente legati al contesto dell'informatica, essi sono strumenti indispensabili nella vita quotidiana. Basti pensare alla cucina e alle ricette utilizzate per cucinare qualsiasi piatto, a un'operazione chirurgica o alla costruzione di un edificio. La definizione di algoritmi è uno dei processi più utilizzati per trasmettere conoscenze e processi definiti.

In questo lavoro si approfondirà il concetto di algoritmo e si esamineranno più da vicino alcuni dei concetti più importanti ad esso correlati, come la sequenza, la selezione e la ripetizione.

**1.Che cos'è un algoritmo?**

Un algoritmo, spesso chiamato anche più semplicemente procedura, è un insieme di passaggi chiari e ben definiti che possono essere utilizzati per completare un compito. Un esempio di algoritmo che può aiutarci a comprendere meglio il concetto è fornito dalla ricetta per fare il caffè con la moka. Nella sua versione più semplice, la ricetta può essere riassunta come segue:

* Smontare la caffettiera nei suoi tre componenti principali: parte inferiore, filtro metallico e parte superiore.
* Versare l'acqua nella parte inferiore della caffettiera moka.
* Posizionare il filtro metallico nell'apposita fessura, sopra la parte inferiore della caffettiera.
* Riempire il filtro di metallo con la polvere di caffè
* Avvitare la parte superiore della caffettiera moka sulla parte inferiore.
* Posizionare la caffettiera su un fornello
* Accendere la pentola
* Attendere che il caffè fuoriesca e riempia la parte superiore della moka
* Spegnere il fornello

Queste istruzioni possono essere seguite efficacemente da chiunque, anche da un robot ragionevolmente sofisticato, e portano sempre allo stesso risultato: un ottimo caffè.

Quindi tutti gli elenchi di operazioni sono algoritmi? In realtà, a livello formale, un algoritmo è sempre caratterizzato da quattro proprietà fondamentali: efficacia, finitezza dell'espressione, finitezza della computazione, determinismo.

**2.Efficacia**

Un algoritmo deve essere effettivamente eseguibile da un esecutore; ciò significa che l'esecutore deve essere in grado di comprendere la descrizione dell'algoritmo e quindi di riconoscere il linguaggio in cui sono espressi i passi che compongono la procedura. Le frasi ben formate in questo linguaggio scelto sono formalmente chiamate istruzioni.

In altre parole, questa proprietà significa che se si definisce un algoritmo utilizzando una lingua, per esempio l'inglese, qualsiasi esecutore in grado di comprendere quella lingua interpreterà le istruzioni nello stesso modo.

2.1 **Finitezza dell'espressione**

La procedura descritta da un algoritmo deve essere articolata in un numero finito di istruzioni. Non importa quante siano le istruzioni (e si può facilmente pensare a procedure con un numero molto elevato di istruzioni) né quanto sia lunga la procedura, purché il numero di istruzioni non sia infinito.

**2.2 Finitezza del calcolo**

Analogamente alla finitezza dell'espressione, un algoritmo deve sempre essere caratterizzato da un numero finito di passi di esecuzione. In altre parole, un algoritmo deve sempre specificare una condizione per la quale l'esecuzione termina. Anche in questo caso, non è importante il numero di passi, ma la nozione di finitezza dell'esecuzione.

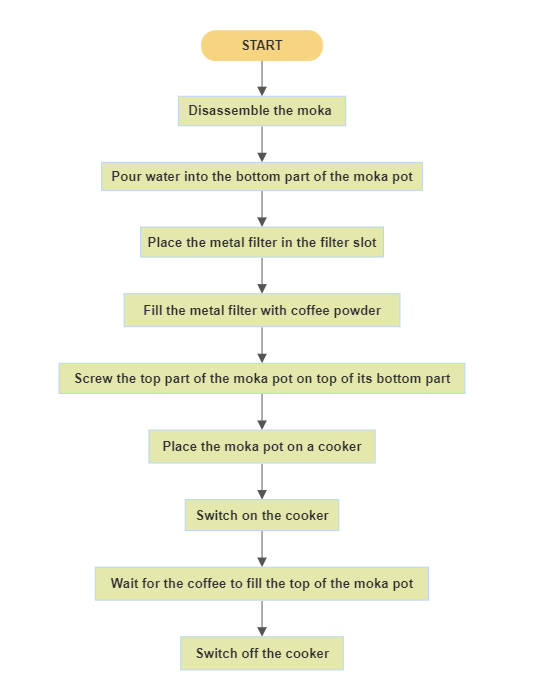
**2.3 Determinismo**

Un algoritmo deve essere deterministico, il che significa che applicando lo stesso algoritmo allo stesso insieme di dati in ingresso si otterranno sempre gli stessi risultati in uscita.

Alla luce di queste quattro proprietà, ci rendiamo conto che per far sì che la nostra ricetta del caffè sia un algoritmo è necessario fare alcune assunzioni: deve essere sempre presente una caffettiera pulita, assemblata e pronta all'uso; deve essere sempre disponibile un fornello; devono essere sempre disponibili acqua e polvere di caffè. E se volessimo rendere il nostro algoritmo più versatile, tenendo conto di queste condizioni iniziali e generalizzando in modo da ottenere una procedura sempre fattibile? Nei prossimi capitoli vedremo da vicino alcuni degli strumenti che ci permetteranno di affinare il nostro algoritmo e, allo stesso tempo, di comprendere meglio questo mondo. In particolare, vedremo tre strutture di base: Sequenza, Selezione e Ripetizione. Queste ultime sono in effetti le più importanti, soprattutto grazie a un teorema definito negli anni Sessanta da due ricercatori: Corrado Böhm e Giuseppe Jacopini [1]. Il teorema afferma che utilizzando solo la Sequenza, la Selezione e la Ripetizione è possibile implementare qualsiasi algoritmo!

**Sequenza**

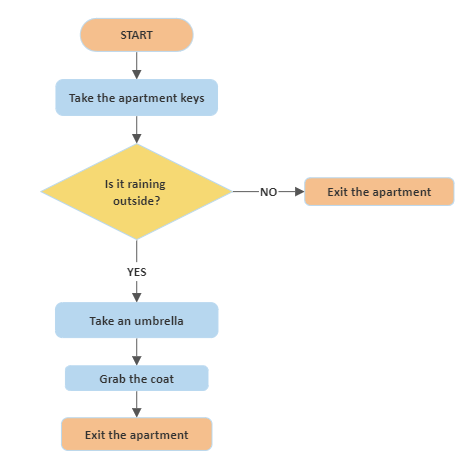
Sequenziare significa definire un ordine per le istruzioni che fanno parte dell'algoritmo. Ad esempio, se consideriamo le istruzioni che compongono la nostra ricetta per fare il caffè:



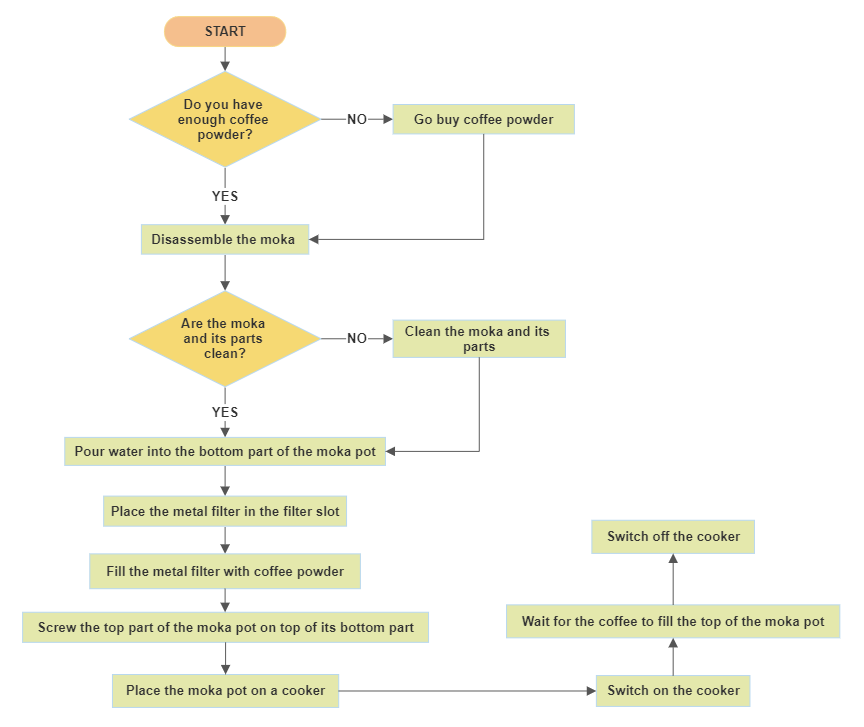
Per garantire che si ottenga sempre lo stesso risultato (assicurando il determinismo), ogni passo deve essere eseguito nell'ordine specificato. Seguendo questo esempio, è facile capire che cambiando l'ordine dei passaggi si otterrebbe un risultato diverso (e non voluto!): cosa succederebbe se si cercasse di riempire d'acqua la parte inferiore della moka prima di smontare la moka stessa?

**Selezione**

La selezione è il costrutto che ci permette di definire diversi percorsi di "esecuzione" e di sceglierne uno piuttosto che un altro a seconda della verifica di una particolare condizione (ad esempio, la condizione è vera o falsa?). Ad esempio, se definissimo un algoritmo per raccogliere oggetti importanti prima di uscire di casa, potremmo definire due percorsi a seconda che fuori piova o meno:



La selezione è un costrutto molto potente che ci permette di rendere un algoritmo flessibile e adattabile. Prendendo come esempio il nostro algoritmo per fare il caffè, possiamo iniziare a testare le condizioni per assicurarci che sia sempre più applicabile nella realtà quotidiana: cosa succede se la caffettiera è sporca? Cosa succede se abbiamo finito la polvere di caffè?



**Ripetizione**

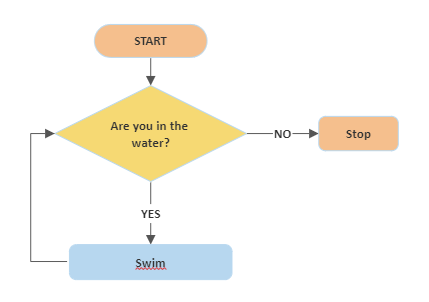
La ripetizione, spesso chiamata anche iterazione o loop, è un costrutto che consente di eseguire un'istruzione (o un insieme di istruzioni) più di una volta senza doverla ripetere ogni volta nel contesto di un blocco di istruzioni sequenziali. Comunemente, esistono tre tipi di ripetizione: controllata dal conteggio (chiamata anche "ciclo for" a causa delle parole chiave utilizzate in vari linguaggi di programmazione per riferirsi a questo tipo di ripetizione), while e repeat-until.

**Ripetizione "controllata" dal conteggio**

La ripetizione controllata dal conteggio è un tipo di ripetizione che consente di ripetere un blocco di istruzioni per un numero predefinito di volte. Ad esempio, se volessimo rendere esplicite le istruzioni per contare alla rovescia da 10 a 0, potremmo dire "sottrai 1 dal conteggio corrente per 10 volte di seguito". Questo costrutto è utile nel contesto di operazioni che sappiamo già in anticipo che verranno ripetute un numero specifico di volte e la cui ripetizione non dipende da nessun'altra condizione.

**Ripetizione di "WHILE"**

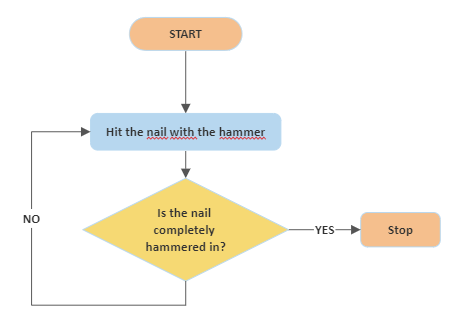
A differenza del controllo del conteggio, la ripetizione while consente di ripetere un blocco di istruzioni finché non viene soddisfatta una condizione. Questo costrutto consente di definire il fatto che le istruzioni devono essere eseguite più volte senza sapere in anticipo quante volte devono essere ripetute. Un esempio di questo tipo di esecuzione è il seguente: "mentre nell'acqua nuota".



È importante notare che quando si utilizza una ripetizione while non si ha la certezza che il blocco di istruzioni destinato a essere ripetuto venga eseguito almeno una volta. Nell'esempio riportato, se l'esecutore è fuori dall'acqua fin dall'inizio, l'istruzione "nuota" non verrà mai eseguita.

**Ripetizione "fino a quando"**

La ripetizione fino a quando è un costrutto molto simile al while: anch'essa viene utilizzata quando un blocco di istruzioni deve essere potenzialmente ripetuto più volte e non si sa in anticipo quante volte dovrà essere ripetuto. L'unica differenza è che il controllo della condizione di stop viene effettuato dopo la prima iterazione dell'istruzione. In altre parole, ci assicuriamo che le istruzioni vengano eseguite almeno una volta prima di decidere se ripeterle o meno.



**Pseudocodice**

I diagrammi di flusso sono uno strumento eccellente per formalizzare un algoritmo in un linguaggio comprensibile e scrivibile. Tuttavia, per quanto utili, presentano importanti svantaggi che ne rendono spesso scomodo l'utilizzo: la quantità di spazio richiesta e la mappatura non immediata dei tipici costrutti di ripetizione (ad esempio, il conteggio controllato). Un'alternativa popolare è quella fornita dallo pseudocodice: un linguaggio che assomiglia ai linguaggi di programmazione, ma che astrae da essi specifiche parole chiave, rimanendo generico e più vicino al linguaggio naturale che usiamo ogni giorno.

In questo capitolo, introdurremo brevemente uno pseudocodice semplificato che ci permetterà di scrivere facilmente i nostri algoritmi utilizzando i costrutti che abbiamo introdotto. All'interno di questo linguaggio, utilizzeremo le seguenti regole:

* ogni riga rappresenta un'istruzione

| questa è un'istruzione  questa è un'altra istruzione  questa è un'altra istruzione |
| --- |

* I blocchi di istruzioni devono essere rappresentati con l'uso dell'indentazione: blocchi diversi avranno un numero diverso di spazi vuoti prima della riga.

| questa istruzione appartiene al blocco A  anche questa istruzione appartiene al blocco A  questa istruzione appartiene al blocco B  anche questa istruzione appartiene al blocco B  anche questa istruzione appartiene al blocco B  questa istruzione appartiene al blocco A |
| --- |

* La selezione è identificata dalla parola chiave IF, seguita da una condizione da verificare e da un nuovo blocco di istruzioni da eseguire se la condizione è vera. Dopo questo blocco, è possibile utilizzare un altro blocco di istruzioni per definire l'istruzione da eseguire se la condizione è falsa; questo blocco deve essere introdotto dalla parola chiave ELSE.

| un'istruzione  un'altra istruzione  Condizione IF  questa istruzione viene eseguita se la condizione è vera  anche questa istruzione verrà eseguita se la condizione è vera  ELSE  questa istruzione verrà eseguita se la condizione è falsa  **questa istruzione viene eseguita anche se la condizione è falsa**  questa istruzione verrà eseguita indipendentemente dalla condizione specificata dopo IF |
| --- |

* La ripetizione controllata dal conteggio è identificata dalla parola chiave FOR X TIMES DO seguita da un blocco di istruzioni da ripetere X volte.

| un'istruzione  un'altra istruzione  FOR X TIME DO  questa istruzione verrà ripetuta X volte  anche questa istruzione sarà ripetuta X volte  questa istruzione non verrà ripetuta |
| --- |

* La ripetizione è identificata dalla parola chiave WHILE <CONDITION> DO seguita da un blocco di istruzioni da ripetere finché <CONDITION> non è più vera. Ricordate che se la condizione non è inizialmente vera, le istruzioni del blocco successivo non verranno eseguite.

| un'istruzione  un'altra istruzione  WHILE CONDITION DO  questa istruzione viene ripetuta finché la condizione non è vera  anche questa istruzione verrà ripetuta fino a quando la condizione sarà vera  questa istruzione non verrà ripetuta |
| --- |

* La ripetizione fino a quando è identificata dalla parola chiave REPEAT seguita da un blocco di istruzioni. Questo blocco di istruzioni deve essere seguito dalla parola chiave UNTIL <CONDITION> che specifica la condizione da verificare per determinare se il blocco di istruzioni deve essere ripetuto o meno.

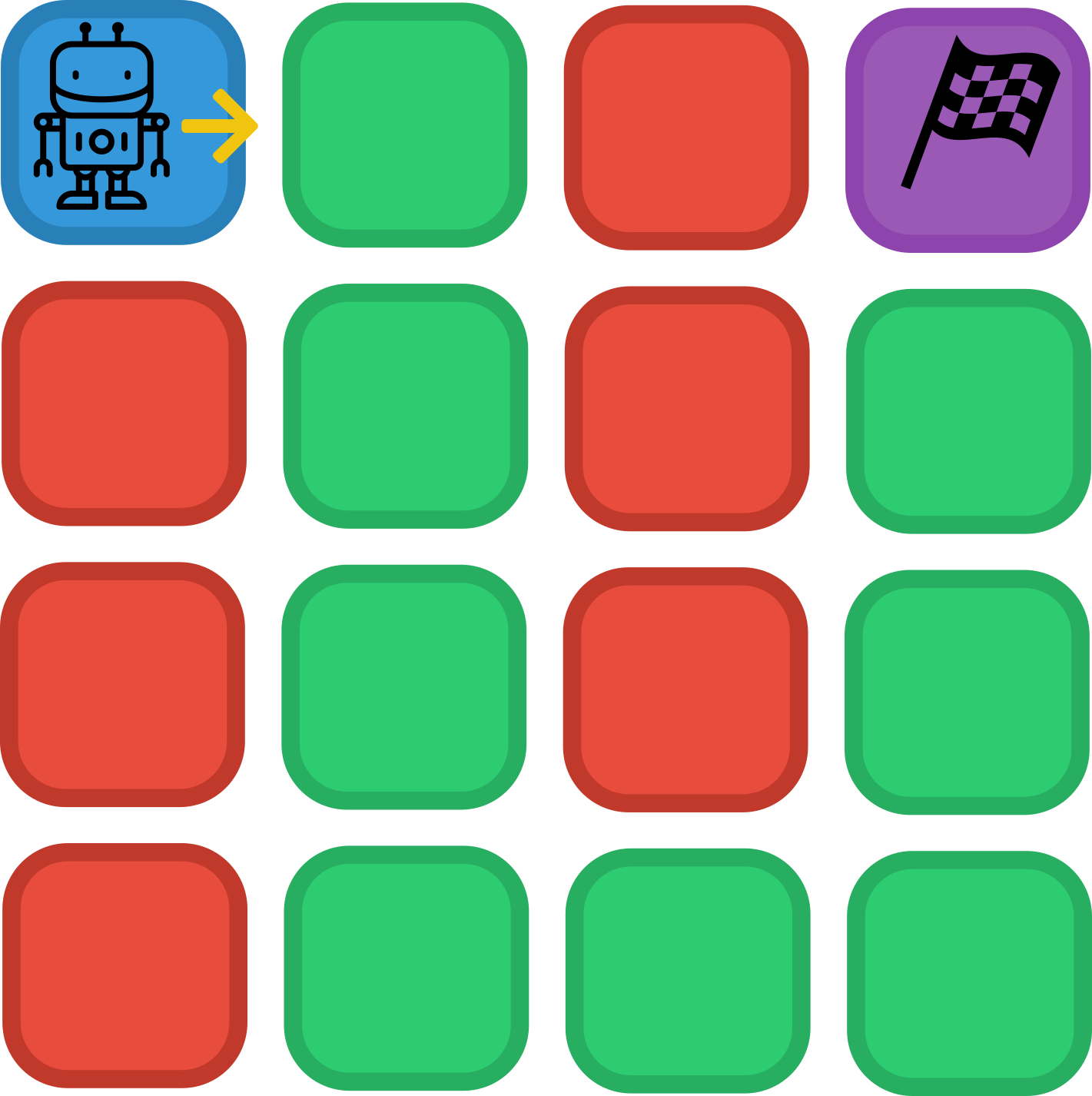
| un'istruzione  un'altra istruzione  RIPETIZIONE  questa istruzione sarà ripetuta fino a quando la condizione sarà vera  anche questa istruzione verrà ripetuta fino a quando la condizione sarà vera  FINO A QUANDO LA CONDIZIONE  questa istruzione non verrà ripetuta |
| --- |

**Problemi risolti**

1) Bob è un piccolo robot. Essendo molto semplice, esegue solo 3 operazioni:

* girare a sinistra di 90°
* girare a destra di 90°
* camminare di una tessera in avanti

Facendo riferimento all'immagine seguente, potete aiutare Bob ad andare dalla piastrella Blu alla piastrella Viola, muovendosi solo attraverso le piastrelle di colore verde? Scrivete due algoritmi per questo compito, uno senza e uno con l'uso di costrutti di ripetizione. Notate che la freccia gialla indica la direzione in cui Bob inizia a guardare..



| **Soluzione senza ripetizioni** | **Soluzione con l'uso di costrutti di ripetizione** |
| --- | --- |
| camminare di una piastrella in avanti  girare a destra di 90°  camminare una piastrella in avanti  camminare una piastrella in avanti  camminare una piastrella in avanti  girare a sinistra di 90°  camminare una piastrella in avanti  camminare una piastrella in avanti  girare a sinistra di 90°  camminare una piastrella in avanti  camminare una piastrella in avanti  camminare una piastrella in avanti | camminare di una piastrella in avanti  girare a destra di 90°  PER 3 VOLTE FARE  camminare una piastrella in avanti  girare a sinistra di 90°  PER 2 VOLTE FARE  camminare una piastrella in avanti  girare a sinistra di 90°  PER 3 VOLTE FARE  camminare una piastrella in avanti |

1. **Riferimenti**

[1] Böhm, C., & Jacopini, G. (1966). Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM, 9*, 366-371.

[2] second reference