

Concepte de bază în gândirea algoritmică: secvențiere, selecție și repetare

Clasă: K11

Cuprins

[1. Introducere 3](#_Toc125750547)

[2. Ce este un Algoritm? 3](#_Toc125750548)

[3. Secvenţierea 5](#_Toc125750549)

[4. Selecţia 6](#_Toc125750550)

[5. Repetarea 7](#_Toc125750551)

[6. Pseudocodul 9](#_Toc125750552)

[7. Probleme rezolvate 12](#_Toc125750553)

[8. Surse 14](#_Toc125750554)

# Introducere

Gândirea algoritmică este capacitatea de a rezolva probleme prin definirea unei secvențe clare și definite de pași care, odată urmați cu fidelitate, conduc la o soluție. În timpul vieții noastre școlare, și în special în anii de școală primară și gimnazială, învățăm să rezolvăm probleme printr-un proces de învățare mnemonică și imitare a soluțiilor oferite de profesor. De exemplu, deși mulți studenți știu să calculeze cel mai mare numitor comun al două numere, puțini pot articula lista operațiilor necesare pentru a ajunge la rezultat.

Acest proces de definire a pașilor necesari pentru a ajunge la o soluție este extrem de important deoarece, pe lângă faptul că face posibilă automatizarea sarcinilor (de exemplu, prin crearea unei mașini capabile să efectueze pașii indicați), conduce individul la o mai mare conștientizarea problemei, a soluționării acesteia și, în consecință, a capacității de a adapta soluția găsită la probleme similare sau analoge: pentru a articula pași clari și bine definiți, trebuie mai întâi să înțelegem problema, condițiile de la care se pleacă și cele de care se dorește să ajungă și instrumentele și operațiunile care ne permit să modificăm sistemul într-un mod programat și previzibil.

Deși conceptele de „algoritm” și „gândire algoritmică” sunt adesea legate imediat de contextul informaticii, ele sunt instrumente indispensabile în viața de zi cu zi. Nu trebuie decât să ne gândim la gătit și la rețetele folosite pentru a găti orice fel de mâncare, o operație chirurgicală sau construcția unei clădiri. Definirea algoritmilor este unul dintre procesele pe care le folosim cel mai mult pentru a transmite cunoștințe și procese definite.

În această lucrare, conceptul de algoritm va fi detaliat, iar unele dintre cele mai importante concepte legate de acesta, cum ar fi secvențierea, selecția și repetiția, vor fi examinate mai îndeaproape.

# Ce este un Algoritm?

Un algoritm, adesea denumit și mai simplu procedură, este un set de pași clari și bine definiți care pot fi utilizați pentru a finaliza o sarcină. Un exemplu de algoritm care ne poate ajuta să înțelegem mai bine conceptul este oferit de rețeta de preparare a cafelei folosind o oală moka. În versiunea sa cea mai simplă, rețeta poate fi rezumată după cum urmează[[1]](#footnote-1):

* Dezasamblați vasul de moka în trei componente principale: partea inferioară, filtrul metalic și partea superioară.
* Turnați apă în partea de jos a vasului de moka
* Așezați filtrul metalic în fanta filtrului, deasupra părții inferioare a vasului de moka
* Umpleți filtrul metalic cu praf de cafea
* Înșurubați partea superioară a vasului de moka peste partea inferioară
* Puneți vasul de moka pe un aragaz
* Porniţi aragazul
* Așteptați ca cafeaua să se reverse și umpleți partea de sus a vasului de moka
* Opriți aragazul

Aceste instrucțiuni pot fi urmate eficient de oricine, chiar și de un robot rezonabil de sofisticat și conduc întotdeauna la același rezultat: o cafea excelentă.

Deci toate listele de operațiuni sunt algoritmi? De fapt, la nivel formal, un algoritm este întotdeauna caracterizat de patru proprietăți fundamentale: eficacitatea, caracterul finit al expresiei, caracterul finit al calculului, determinismul.

* 1. **Eficacitatea**

Un algoritm trebuie să fie executabil efficient de către un executant; aceasta înseamnă că executantul trebuie să fie capabil să înțeleagă descrierea algoritmului și, prin urmare, să poată recunoaște limbajul în care sunt exprimați pașii care compun procedura. Propozițiile bine formate în această limbă aleasă sunt denumite formal instrucțiuni.

Cu alte cuvinte, această proprietate înseamnă că dacă se definește un algoritm folosind o limbă, de exemplu engleza, orice executant capabil să înțeleagă acea limbă va interpreta instrucțiunile în același mod.

* 1. **Caracterul finit al expresiei**

Procedura descrisă de un algoritm trebuie articulată într-un număr finit de instrucțiuni. Nu contează câte instrucțiuni există (și ne putem gândi cu ușurință la proceduri cu un număr foarte mare de instrucțiuni) și nici cât de lungă este procedura, atâta timp cât numărul de instrucțiuni nu este infinit.

* 1. **Caracterul finit al calculului**

Similar cu caracterul finit al expresiei, un algoritm trebuie întotdeauna caracterizat printr-un număr finit de pași de execuție. Cu alte cuvinte, un algoritm trebuie să specifice întotdeauna o condiție pentru care se încheie execuția. Din nou, nu numărul de pași este important, ci noțiunea de caracter finit al execuției.

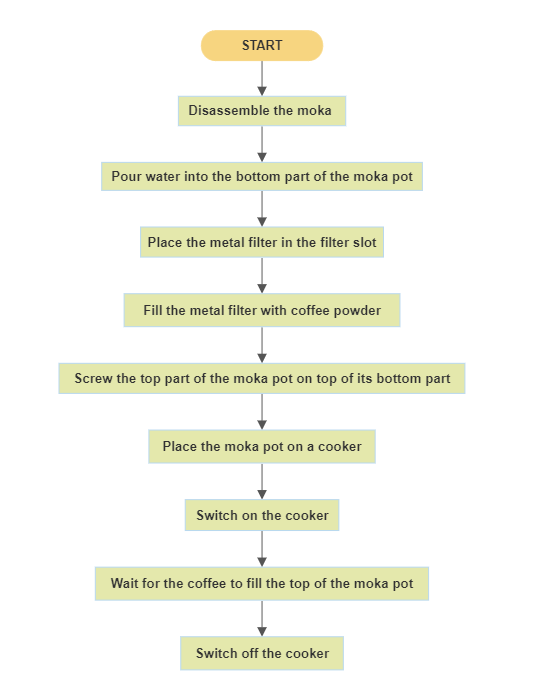
* 1. **Determinismul**

Un algoritm trebuie să fie determinist, ceea ce înseamnă: aplicarea aceluiași algoritm la același set de date de intrare va produce întotdeauna aceleași rezultate de ieșire.

În lumina acestor patru proprietăți, ne dăm seama că trebuie făcute anumite ipoteze pentru a menține rețeta noastră de cafea un algoritm: trebuie să fie întotdeauna prezent un vas de moka curat, asamblat și gata de utilizare; un aragaz trebuie să fie întotdeauna disponibil; apa și pudra de cafea trebuie să fie întotdeauna disponibile. Ce se întâmplă dacă am dori să facem algoritmul nostru mai versatil, ținând cont de aceste condiții inițiale și generalizând astfel încât să obținem o procedură care este întotdeauna fezabilă? În capitolele următoare vom arunca o privire mai atentă asupra unora dintre instrumentele care ne vor permite să ne perfecționăm algoritmul și, în același timp, să înțelegem mai bine această lume. În special, vom vedea trei structuri de bază: secvențiere, selecție și repetiție. Acestea sunt de fapt cele mai importante, mai ales datorită unei teoreme definite în anii 1960 de doi cercetători: Corrado Böhm și Giuseppe Jacopini [1]. Teorema afirmă că folosind doar Senquencing, Selection și Repetition, orice algoritm poate fi implementat!

# Secvenţierea

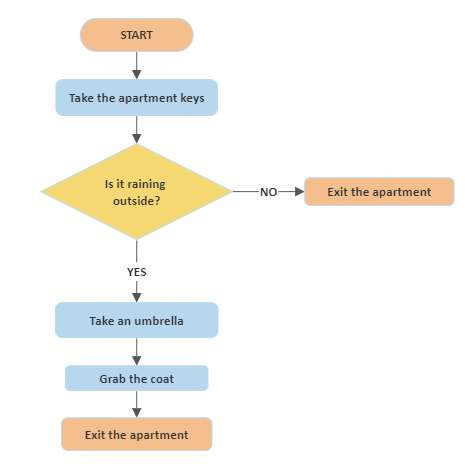
Secvențierea înseamnă definirea unei ordini pentru instrucțiunile care fac parte din algoritm. De exemplu, dacă luăm în considerare instrucțiunile care compun rețeta noastră de preparare a cafelei:



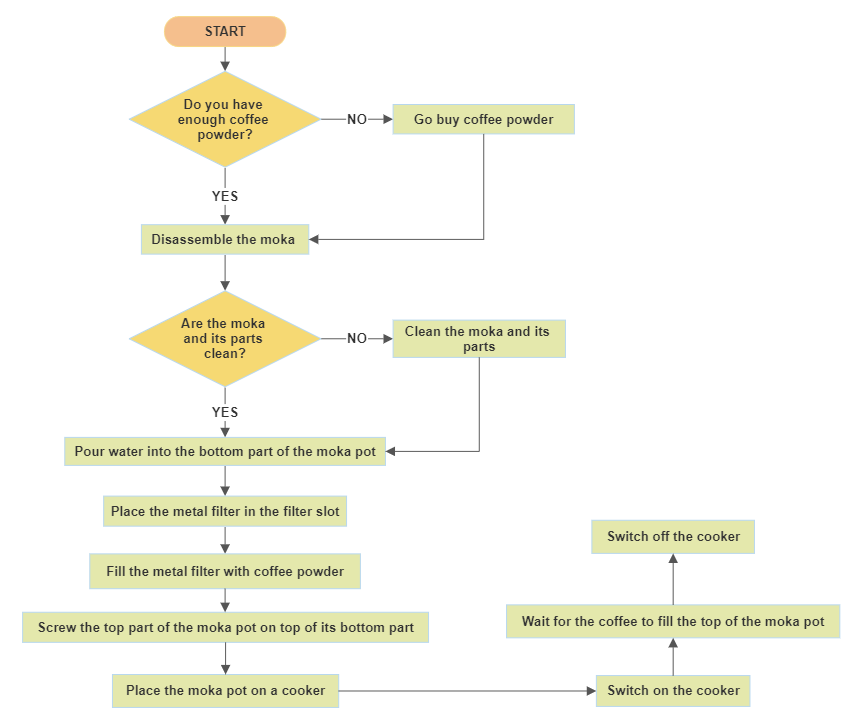
Pentru a ne asigura că se obține întotdeauna același rezultat (asigurând determinismul), fiecare pas trebuie efectuat în ordinea specificată. Urmând acest exemplu, este ușor de observat că schimbarea ordinii pașilor ar duce la un rezultat diferit (și neintenționat!): ce s-ar întâmpla dacă ați încerca să umpleți partea de jos a vasului de moka cu apă înainte de a demonta moka în sine?

# Selecţia

Selecția este constructul care ne permite să definim diferite căi de „execuție” și să alegem una mai degrabă decât alta, în funcție de verificarea unei anumite condiții (de exemplu, condiția este adevărată sau falsă?). De exemplu, dacă am definit un algoritm pentru a colecta obiecte importante înainte de a părăsi casa, am putea defini două căi în funcție de dacă afară plouă sau nu în acest moment:



Selecția este o construcție foarte puternică care ne permite să facem un algoritm flexibil și adaptabil. Luând ca exemplu algoritmul nostru de preparare a cafelei, putem începe să testăm condițiile pentru a ne asigura că este din ce în ce mai aplicabil în realitatea de zi cu zi: ce se întâmplă dacă cafetierul este murdar? Ce se întâmplă dacă rămânem fără cafea pudră?



# Repetarea

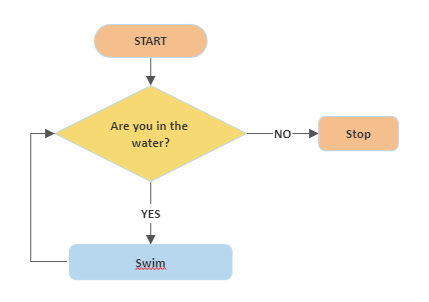
Repetarea, numită adesea și iterație sau buclă, este o construcție care permite ca o instrucțiune (sau un set de instrucțiuni) să fie executată de mai multe ori fără a fi nevoie să le repeți de fiecare dată în contextul unui bloc de instrucțiuni secvențiale. În mod obișnuit, există trei tipuri de repetiție: controlată de numărare (numită și „for loop” din cauza cuvintelor cheie folosite în diferite limbaje de programare pentru a se referi la acest tip de repetiție), while și repeat-until.

* 1. **Repetare „controlată prin numărare”**

Repetarea controlată de numărare este un tip de repetiție care permite unui bloc de instrucțiuni să fie repetat de un număr predefinit de ori. De exemplu, dacă dorim să explicăm instrucțiunile de numărare inversă de la 10 la 0, am putea spune „scădeți 1 din numărul curent de 10 ori la rând”. Acest construct este util în contextul operațiilor despre care știm deja dinainte că se vor repeta de un anumit număr de ori și a căror repetare nu depinde de nicio altă condiție.

**5.2. Repetarea „while”**

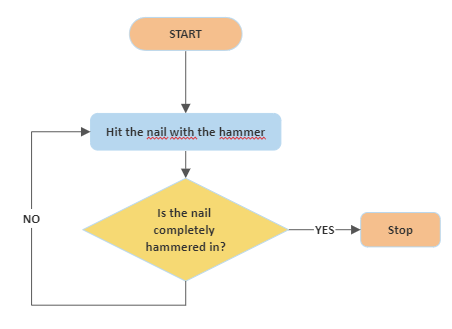
Spre deosebire de controlul numărării, repetarea *while* permite repetarea unui bloc de instrucțiuni până când este îndeplinită o condiție. Acest construct face posibilă definirea faptului că instrucțiunile urmează să fie executate de mai multe ori fără a ști în prealabil de câte ori urmează să fie repetate. Un exemplu de acest tip de execuție este următorul: „în timp ce înotați în apă”.



Este important de remarcat că atunci când folosim o repetare *while*, nu avem nicio asigurare că blocul de instrucțiuni menit să fie repetat va fi executat cel puțin o dată. În exemplul dat, dacă executantul este în afara apei de la început, instrucțiunea „înot” nu se execută niciodată.

**5.2. Repetarea „Repeat until”**

Repetarea de *Repeat until* este de fapt un construct foarte asemănător cu *while*: este folosit și atunci când un bloc de instrucțiuni poate fi repetat de mai multe ori și nu știm dinainte de câte ori va trebui repetat. Singura diferență este că verificarea stării de oprire este efectuată după prima iterație a instrucțiunii. Cu alte cuvinte, ne asigurăm că instrucțiunile sunt executate cel puțin o dată înainte de a decide dacă să le repetăm sau nu.



# Pseudocodul

Diagramele de flux sunt un instrument excelent pentru formalizarea unui algoritm într-un limbaj ușor de înțeles și de scris. Cu toate acestea, oricât de utile sunt, au dezavantaje importante care le fac adesea incomod de utilizat: cantitatea de spațiu necesară și maparea neimediată a constructelor tipice de repetiție (de exemplu, numărare controlată). O alternativă populară este cea oferită de pseudocod: un limbaj care seamănă cu limbajele de programare, dar abstrage anumite cuvinte cheie din ele, rămânând generic și mai apropiat de limbajul natural pe care îl folosim în fiecare zi.

În acest capitol, vom introduce pe scurt un pseudocod simplificat care ne va permite cu ușurință să scriem proprii algoritmi folosind constructele pe care le-am introdus. În cadrul acestui limbaj, vom folosi următoarele reguli:

* fiecare linie reprezintă o instrucţiune

|  |
| --- |
| aceasta este o instrucțiune  aceasta este o altă instrucțiune  aceasta este o altă instrucțiune |

* blocurile de instrucțiuni trebuie să fie reprezentate prin utilizarea indentării: diferite blocuri vor avea un număr diferit de spații goale înainte de linie

|  |
| --- |
| această instrucţiune aparţine **blocului A**  această instrucţiune aparţine **blocului A**  această instrucţiune aparţine **blocului B**  această instrucţiune aparţine **blocului B**  această instrucţiune aparţine **blocului B**  această instrucţiune aparţine **blocului A** |

* Selectarea este identificată prin cuvântul cheie **IF**, urmat de o condiție de verificat și apoi de un nou bloc de instrucțiuni care trebuie executat dacă condiția este adevărată. În urma acestui bloc, un alt bloc de instrucțiuni poate fi utilizat pentru a defini instrucțiunea de executat dacă condiția este falsă; acest bloc trebuie introdus prin cuvântul cheie **ELSE**.

|  |
| --- |
| o instructiune  o altă instrucțiune  condiţia **IF**  **această instrucțiune va fi executată dacă condiția este adevărată**  **această instrucțiune va fi executată și dacă condiția este adevărată**  **ELSE**  **această instrucțiune va fi executată dacă condiția este falsă**  **această instrucțiune va fi executată și dacă condiția este falsă**  această instrucţiune va fi executată indiferent de condiţia specificată după IF |

* Repetarea controlată de numărare este identificată prin cuvântul cheie **FOR X TIMES DO** urmat de un bloc de instrucțiuni care trebuie repetat de X ori.

|  |
| --- |
| o instructiune  o altă instrucțiune  **FOR X TIMES DO**  **această instrucțiune se va repeta de X ori**  **această instrucțiune se va repeta și de X ori**  această instrucțiune nu se va repeta |

* În timp ce repetarea este identificată prin cuvântul cheie **WHILE <condiţie> DO** urmat de un bloc de instrucțiuni care trebuie repetat până când < condiţie > nu mai este adevărat. Rețineți că dacă condiția nu este inițial adevărată, instrucțiunile din următorul bloc nu vor fi executate.

|  |
| --- |
| o instructiune  o altă instrucțiune  **WHILE condiţie DO**  **această instrucțiune se va repeta până când condiția este adevărată**  **această instrucțiune va fi de asemenea repetată până când condiția este adevărată**  această instrucțiune nu se va repeta |

● Repetarea *repeat until* este identificată prin cuvântul cheie **REPEAT** urmat de un bloc de instrucțiuni. Acest bloc de instrucțiuni trebuie să fie urmat de cuvântul cheie **UNTIL <condiţie>** care specifică condiția care trebuie verificată pentru a determina dacă blocul de instrucțiuni trebuie repetat sau nu.

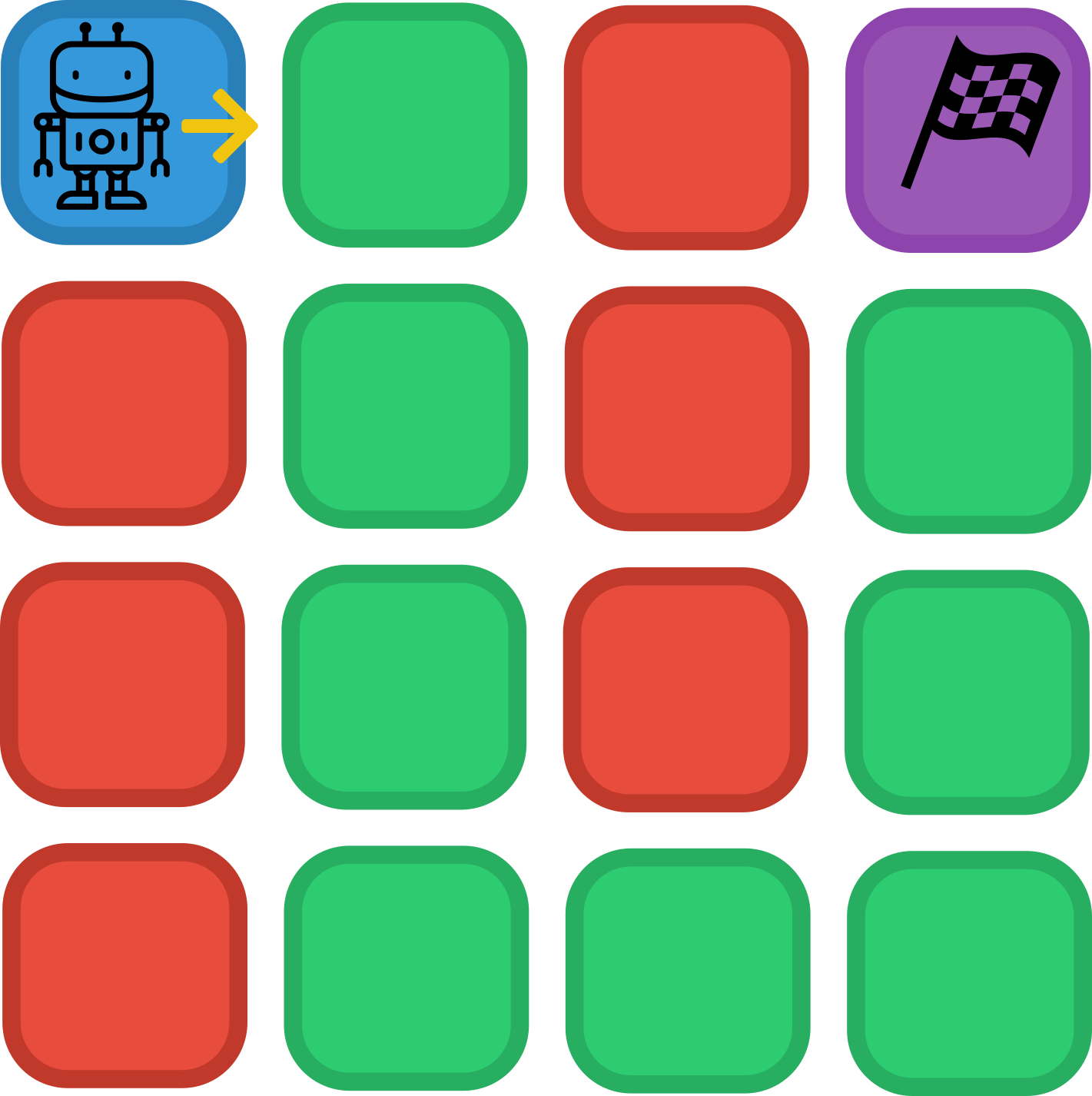
|  |
| --- |
| o instructiune  o altă instrucțiune  **REPEAT**  **această instrucțiune se va repeta până când condiția este adevărată**  **această instrucțiune va fi de asemenea repetată până când condiția este adevărată**  **UNTIL CONDITION**  această instrucțiune nu se va repeta |

# Probleme rezolvate

1) Bob este un mic robot. Fiind unul foarte simplu, efectuează doar 3 operațiuni:

* virați la stânga 90°
* virați la dreapta 90°
* mergeți cu o piesă înainte

Referindu-ne la următoarea imagine, îl puteți ajuta pe Bob să treacă de la țiglă albastră la țigla violet, mișcându-se doar prin plăcile de culoare verde? Scrieți doi algoritmi pentru această sarcină, unul fără și unul cu utilizarea constructelor de repetiție. Vă rugăm să rețineți că săgeata galbenă indică direcția către care începe Bob să privească.



|  |  |
| --- | --- |
| **Soluție fără repetare** | **Rezolvare cu utilizarea constructelor repetitive** |
| mergi cu o țiglă înainte  virați la dreapta 90°  mergi cu o țiglă înainte  mergi cu o țiglă înainte  mergi cu o țiglă înainte  virați la stânga 90°  mergi cu o țiglă înainte  mergi cu o țiglă înainte  virați la stânga 90°  mergi cu o țiglă înainte  mergi cu o țiglă înainte  mergi cu o țiglă înainte | mergi cu o țiglă înainte  virați la dreapta 90°  FOR 3 TIMES DO  mergi cu o țiglă înainte  virați la stânga 90°  FOR 2 TIMES DO  mergi cu o țiglă înainte  virați la stânga 90°  FOR 2 TIMES DO  mergi cu o țiglă înainte |

# Surse

[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

1. Pictogramă vas Moka creată de Dooder și descărcată de pe Flaticon - „https://www.flaticon.com/free-icons/moka-pot” [↑](#footnote-ref-1)