

Základné koncepty algoritmického myslenia: Sekvencovanie, výber a opakovanie

Obsah

[Úvod](#_heading=h.gjdgxs) **3**

[Čo je to algoritmus?](#_heading=h.y9fd3ccnzb3t) **3**

[Sekvenovanie](#_heading=h.sx9mm8vv41lj) **4**

[Výber](#_heading=h.7a1rycfnc77v) **6**

[Opakovanie](#_heading=h.c0s0rsaecwjw) **7**

[Pseudokód](#_heading=h.8i64txlf9n76) **9**

[Vyriešené problémy](#_heading=h.8f8zlhq795e6) **11**

[Odkazy](#_heading=h.2et92p0) **11**

[Prílohy](#_heading=h.tyjcwt) **11**

# Úvod

Algoritmické myslenie je schopnosť riešiť problémy definovaním jasnej a definovanej postupnosti krokov, ktoré po presnom dodržaní vedú k riešeniu. Počas školského života, a najmä počas základnej a strednej školy, sa učíme riešiť problémy prostredníctvom procesu mnemotechnického učenia a napodobňovania riešení, ktoré nám poskytuje učiteľ. Napríklad, hoci mnohí žiaci vedia vypočítať najväčšieho spoločného menovateľa dvoch čísel, len málo z nich dokáže vyjadriť zoznam operácií potrebných na dosiahnutie výsledku.

Tento proces definovania krokov potrebných na dosiahnutie riešenia je mimoriadne dôležitý, pretože okrem toho, že umožňuje automatizovať úlohy (napr. vytvorením stroja schopného vykonávať uvedené kroky), vedie jednotlivca k lepšiemu pochopeniu problému, jeho riešenia a následne k schopnosti prispôsobiť nájdené riešenie podobným alebo analogickým problémom: na to, aby sme mohli sformulovať jasné a dobre definované kroky, musíme najprv pochopiť problém, podmienky, z ktorých vychádzame, a podmienky, ku ktorým chceme dospieť, a nástroje a operácie, ktoré nám umožňujú programovo a predvídateľne modifikovať systém.

Hoci sa pojmy "algoritmus" a "algoritmické myslenie" často okamžite spájajú s kontextom počítačovej vedy, sú to nepostrádateľné nástroje v každodennom živote. Stačí si spomenúť na varenie a recepty používané na prípravu akéhokoľvek jedla, chirurgickú operáciu alebo stavbu budovy. Definovanie algoritmov je jedným z procesov, ktoré najviac používame na odovzdávanie vedomostí a definovaných postupov.

V tejto práci sa podrobne rozoberie pojem algoritmus a bližšie sa preskúmajú niektoré najdôležitejšie pojmy, ktoré s ním súvisia, ako napríklad sekvencia, výber a opakovanie.

# Čo je to algoritmus?

Algoritmus, často označovaný aj jednoduchším názvom postup, je súbor jasných a presne definovaných krokov, ktoré možno použiť na dokončenie úlohy. Príklad algoritmu, ktorý nám môže pomôcť lepšie pochopiť tento pojem, poskytuje recept na prípravu kávy pomocou moka kanvice. V najjednoduchšej verzii možno recept zhrnúť takto:[[1]](#footnote-1) :

* Rozložte moka hrniec na tri hlavné časti: spodnú časť, kovový filter a vrchnú časť.
* Do spodnej časti moka hrnca nalejte vodu
* Umiestnite kovový filter do štrbiny na filter v hornej časti moka hrnca.
* Naplňte kovový filter kávovým práškom
* Hornú časť moka hrnca naskrutkujte na jeho spodnú časť.
* Postavte moka hrniec na varnú dosku
* Zapnutie sporáka
* Počkajte, kým sa káva vyleje a naplní hornú časť moka kanvice.
* Vypnite sporák

Tieto pokyny môže účinne dodržiavať ktokoľvek, dokonca aj pomerne sofistikovaný robot, a vždy vedú k rovnakému výsledku: výbornej káve.

Takže všetky zoznamy operácií sú algoritmy? Na formálnej úrovni je algoritmus vždy charakterizovaný štyrmi základnými vlastnosťami: efektívnosť, konečnosť vyjadrenia, konečnosť výpočtu, determinizmus.

* 1. **Účinnosť**

Algoritmus musí byť efektívne vykonateľný vykonávateľom; to znamená, že vykonávateľ musí byť schopný porozumieť opisu algoritmu, a teda musí byť schopný rozpoznať jazyk, v ktorom sú vyjadrené kroky tvoriace postup. Dobre formulované vety v tomto zvolenom jazyku sa formálne nazývajú inštrukcie.

Inými slovami, táto vlastnosť znamená, že ak definujeme algoritmus pomocou jazyka, napríklad angličtiny, každý vykonávací program schopný porozumieť tomuto jazyku bude interpretovať inštrukcie rovnakým spôsobom.

* 1. **Konečnosť vyjadrenia**

Postup opísaný algoritmom musí byť vyjadrený konečným počtom inštrukcií. Nezáleží na tom, koľko inštrukcií je (a ľahko si možno predstaviť procedúry s veľmi veľkým počtom inštrukcií), ani na tom, aký dlhý je postup, pokiaľ počet inštrukcií nie je nekonečný.

* 1. **Konečnosť výpočtu**

Podobne ako konečnosť výrazu, aj algoritmus musí byť vždy charakterizovaný konečným počtom krokov vykonávania. Inými slovami, algoritmus musí vždy špecifikovať podmienku, pre ktorú sa vykonávanie končí. Ani v tomto prípade nie je dôležitý počet krokov, ale pojem konečnosti vykonávania.

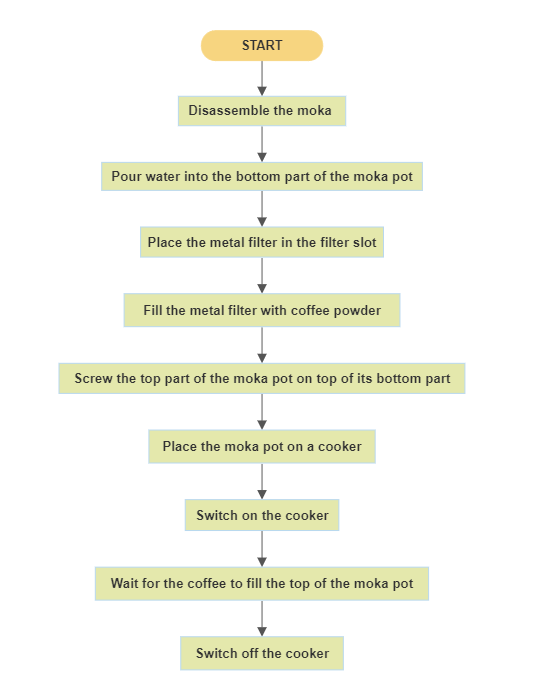
* 1. **Determinizmus**

Algoritmus musí byť deterministický, čo znamená, že pri použití rovnakého algoritmu na rovnaký súbor vstupných údajov sa vždy získajú rovnaké výstupné výsledky.

Vzhľadom na tieto štyri vlastnosti si uvedomujeme, že na to, aby bol náš recept na kávu algoritmom, je potrebné prijať určité predpoklady: vždy musí byť k dispozícii čistá, zostavená a na použitie pripravená moka kanvica; vždy musí byť k dispozícii varič; vždy musí byť k dispozícii voda a kávový prášok. Čo keby sme chceli náš algoritmus urobiť univerzálnejším, zohľadniť tieto počiatočné podmienky a zovšeobecniť ho tak, aby sme získali postup, ktorý je vždy uskutočniteľný? V nasledujúcich kapitolách sa bližšie pozrieme na niektoré nástroje, ktoré nám umožnia zdokonaliť náš algoritmus a zároveň lepšie pochopiť tento svet. Konkrétne si ukážeme tri základné štruktúry: Sekvenovanie, výber a opakovanie. Tie sú v skutočnosti najdôležitejšie, najmä vďaka vete, ktorú v 60. rokoch 20. storočia definovali dvaja výskumníci: Corrado Böhm a Giuseppe Jacopini [1]. V teoréme sa uvádza, že len pomocou Senquencing, Selection a Repetition možno implementovať akýkoľvek algoritmus!

# Sekvenovanie

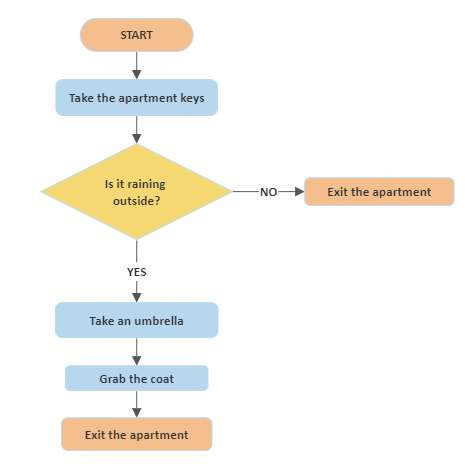
Sekvencovanie znamená definovanie poradia inštrukcií, ktoré sú súčasťou algoritmu. Ak napríklad uvažujeme o inštrukciách, ktoré tvoria náš recept na prípravu kávy:



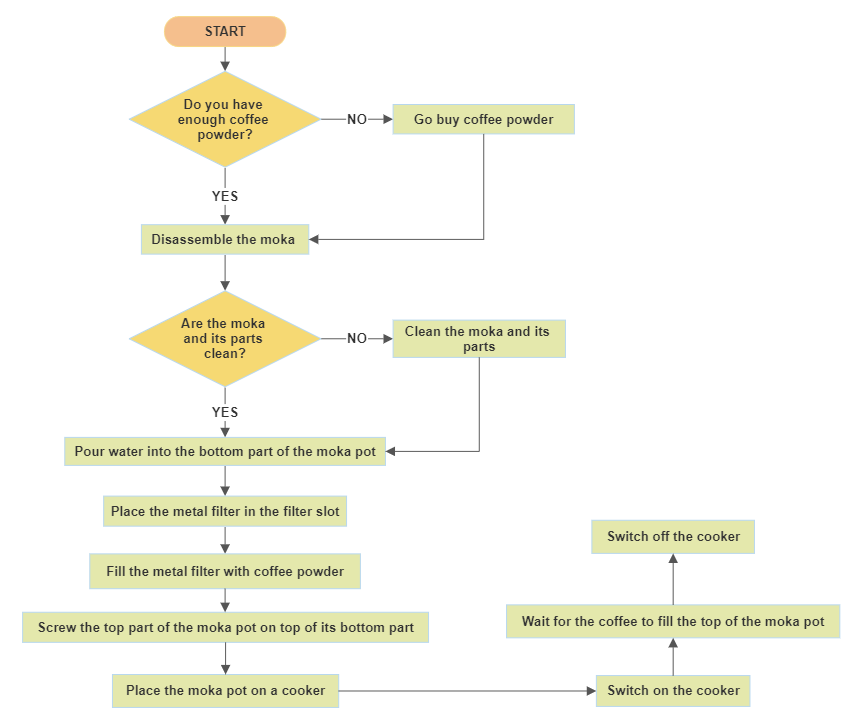
Aby sa zabezpečilo, že sa vždy dosiahne rovnaký výsledok (zabezpečenie determinizmu), každý krok sa musí vykonať v určenom poradí. Podľa tohto príkladu je ľahké vidieť, že zmena poradia krokov by viedla k odlišnému (a nezamýšľanému!) výsledku: čo by sa stalo, keby ste sa pokúsili naplniť spodnú časť moka hrnca vodou skôr, ako by ste rozobrali samotný moka hrniec?

# Výber

Selekcia je konštrukcia, ktorá nám umožňuje definovať rôzne cesty "vykonávania" a vybrať jednu z nich namiesto inej v závislosti od overenia konkrétnej podmienky (napr. je podmienka pravdivá alebo nepravdivá?). Ak by sme napríklad definovali algoritmus na zozbieranie dôležitých predmetov pred odchodom z domu, mohli by sme definovať dve cesty v závislosti od toho, či vonku práve prší, alebo nie:



Výber je veľmi silná konštrukcia, ktorá nám umožňuje urobiť algoritmus flexibilným a prispôsobivým. Ak si zoberieme ako príklad náš algoritmus na prípravu kávy, môžeme začať testovať podmienky, aby sme sa uistili, že je čoraz viac použiteľný v každodennej realite: čo sa stane, ak je kanvica na kávu špinavá? Čo sa stane, ak nám dôjde kávový prášok?



# Opakovanie

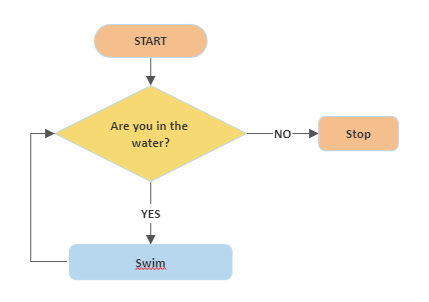
Opakovanie, často nazývané aj iterácia alebo cyklus, je konštrukcia, ktorá umožňuje vykonať inštrukciu (alebo sadu inštrukcií) viac ako raz bez toho, aby sa musela zakaždým opakovať v rámci sekvenčného bloku inštrukcií. Bežne existujú tri typy opakovania: *riadené počítaním* (nazývané aj "for loop" kvôli kľúčovým slovám používaným v rôznych programovacích jazykoch na označenie tohto typu opakovania), *while* a *repeat-until*.

* 1. **"Počítaním riadené" opakovanie**

Opakovanie riadené počtom je typ opakovania, ktorý umožňuje opakovať blok inštrukcií vopred definovaný počet krát. Napríklad, ak by sme chceli explicitne uviesť inštrukcie na odpočítavanie od 10 do 0, mohli by sme povedať "od aktuálneho počtu odčítaj 1 10-krát za sebou". Táto konštrukcia je užitočná v kontexte operácií, o ktorých už vopred vieme, že sa budú opakovať určitý početkrát a ktorých opakovanie nezávisí od žiadnej inej podmienky.

* 1. **Opakovanie "While"**

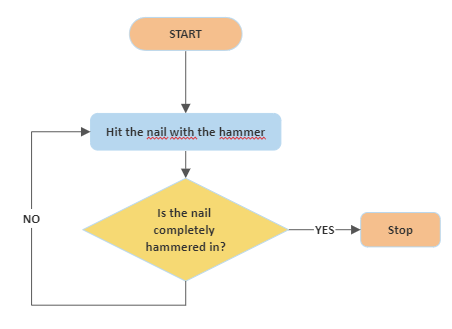
Na rozdiel od kontroly počítania, opakovanie while umožňuje opakovať blok inštrukcií, kým nie je splnená podmienka. Táto konštrukcia umožňuje definovať skutočnosť, že inštrukcie sa majú vykonať niekoľkokrát bez toho, aby sa vopred vedelo, koľkokrát sa majú opakovať. Príkladom tohto typu vykonávania je nasledujúci postup: "while in the water swim".



Je dôležité si uvedomiť, že pri použití opakovania while nemáme istotu, že sa blok inštrukcií, ktorý sa má opakovať, vykoná aspoň raz. V uvedenom príklade, ak je vykonávateľ od začiatku mimo vody, inštrukcia "plávať" sa nikdy nevykoná.

* 1. **Opakovanie "Repeat until"**

Opakovanie Repeat-until je v skutočnosti veľmi podobná konštrukcia ako while: tiež sa používa, keď sa blok inštrukcií potenciálne musí opakovať niekoľkokrát a my vopred nevieme, koľkokrát sa bude musieť opakovať. Jediný rozdiel je v tom, že kontrola podmienky zastavenia sa vykoná po prvej iterácii inštrukcie. Inými slovami, zabezpečíme, aby sa inštrukcie vykonali aspoň raz, a až potom sa rozhodneme, či ich budeme opakovať alebo nie.



# Pseudokód

Vývojové diagramy sú vynikajúcim nástrojom na formalizáciu algoritmu v zrozumiteľnom a zapisovateľnom jazyku. Akokoľvek sú však užitočné, majú dôležité nevýhody, ktoré často spôsobujú, že ich používanie je nepohodlné: množstvo potrebného priestoru a neimplicitné mapovanie typických opakovacích konštrukcií (napr. riadené počítanie). Obľúbenou alternatívou je tá, ktorú poskytuje pseudokód: jazyk, ktorý sa podobá programovacím jazykom, ale abstrahuje z nich konkrétne kľúčové slová, zostáva všeobecný a bližší prirodzenému jazyku, ktorý používame každý deň.

V tejto kapitole si stručne predstavíme zjednodušený pseudokód, ktorý nám umožní jednoducho písať vlastné algoritmy s využitím konštrukcií, ktoré sme si predstavili. V rámci tohto jazyka budeme používať nasledujúce pravidlá:

* každý riadok predstavuje inštrukciu

|  |
| --- |
| toto je jeden z pokynov  toto je ďalší pokyn  toto je ďalší pokyn |

* bloky inštrukcií musia byť reprezentované použitím odsadenia: rôzne bloky budú mať rôzny počet prázdnych miest pred riadkom

|  |
| --- |
| táto inštrukcia patrí do **bloku A**  táto inštrukcia tiež patrí do **bloku A**  táto inštrukcia patrí do **bloku B**  táto inštrukcia tiež patrí do **bloku B**  táto inštrukcia tiež patrí do **bloku B**  táto inštrukcia patrí do **bloku A** |

* Výber sa označuje kľúčovým slovom **IF,** za ktorým nasleduje podmienka, ktorá sa má overiť, a potom nový blok inštrukcií, ktorý sa vykoná, ak je podmienka pravdivá. Po tomto bloku sa môže použiť ďalší blok inštrukcií na definovanie inštrukcie, ktorá sa vykoná, ak je podmienka nepravdivá; tento blok musí byť uvedený kľúčovým slovom **ELSE**.

|  |
| --- |
| pokyn  iný pokyn  **Podmienka IF**  **táto inštrukcia sa vykoná, ak je podmienka true**  **táto inštrukcia sa vykoná aj vtedy, ak je podmienka true**  **ELSE**  **táto inštrukcia sa vykoná, ak je podmienka false**  **táto inštrukcia sa vykoná aj vtedy, ak je podmienka false**  táto inštrukcia sa vykoná bez ohľadu na podmienku zadanú za IF |

* Opakovanie riadené počtom sa označuje kľúčovým slovom **FOR X TIMES DO,** za ktorým nasleduje blok inštrukcií, ktorý sa má opakovať X-krát.

|  |
| --- |
| pokyn  iný pokyn  **PRE X ČASOV DO**  **tento pokyn sa zopakuje X-krát**  **tento pokyn sa tiež zopakuje X-krát**  tento pokyn sa nebude opakovať |

* Opakovanie sa označuje kľúčovým slovom **WHILE <CONDITION> DO,** za ktorým nasleduje blok inštrukcií, ktorý sa má opakovať, kým <CONDITION> prestane platiť. Pamätajte, že ak podmienka nie je na začiatku pravdivá, inštrukcie v nasledujúcom bloku sa nevykonajú.

|  |
| --- |
| pokyn  iný pokyn  **WHILE PODMIENKA DO**  **tento pokyn sa bude opakovať, kým sa podmienka nestane pravdivou**  **táto inštrukcia sa bude tiež opakovať, kým sa podmienka nestane pravdivou**  tento pokyn sa nebude opakovať |

* Opakovanie sa označuje kľúčovým slovom **REPEAT,** za ktorým nasleduje blok inštrukcií. Za týmto blokom inštrukcií musí nasledovať kľúčové slovo **UNTIL <CONDITION**>, ktoré špecifikuje podmienku, ktorá sa má kontrolovať, aby sa určilo, či sa má blok inštrukcií opakovať alebo nie.

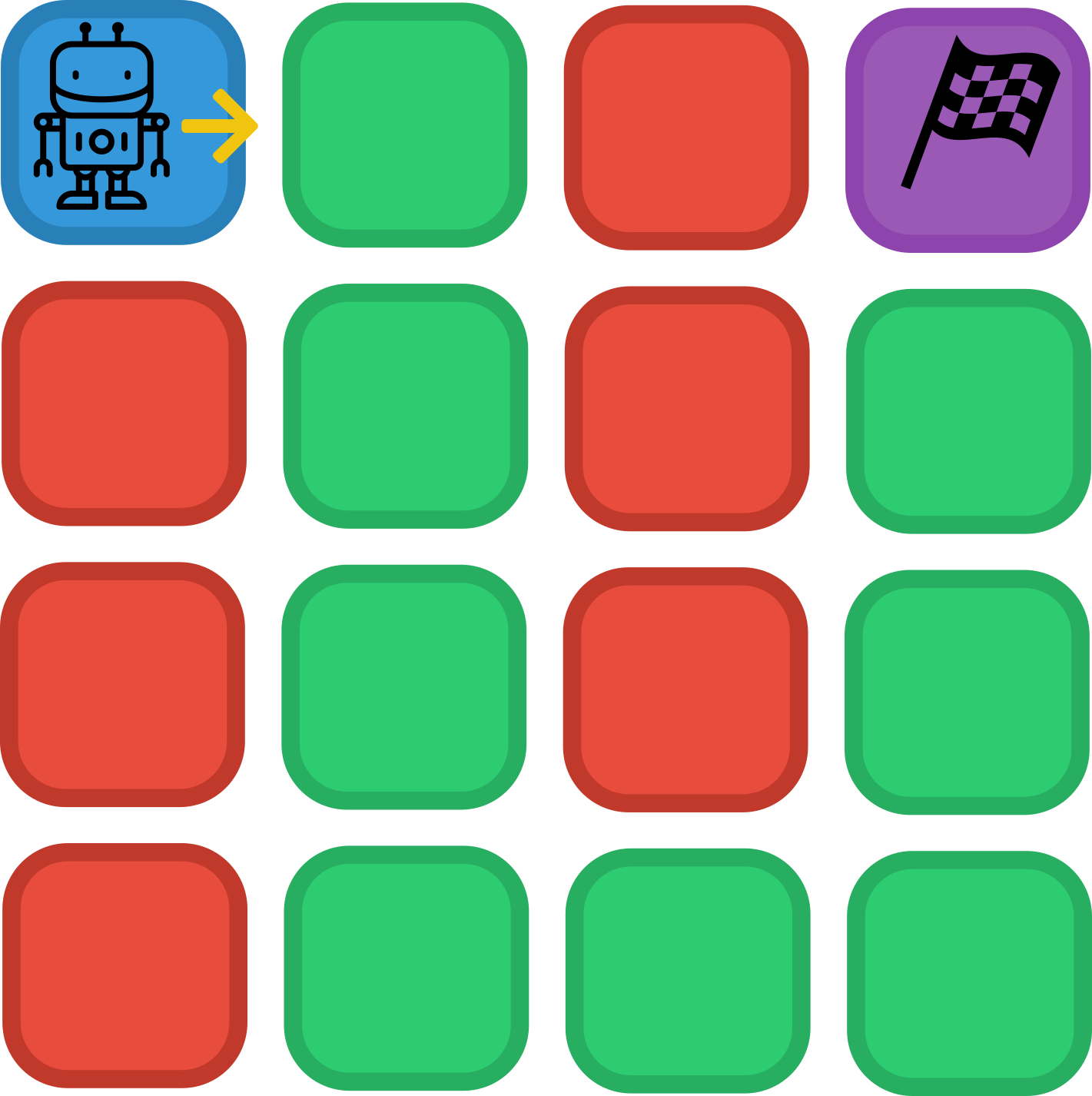
|  |
| --- |
| pokyn  iný pokyn  **OPAKOVAŤ**  **tento pokyn sa bude opakovať, kým sa podmienka nestane pravdivou**  **táto inštrukcia sa bude tiež opakovať, kým sa podmienka nestane pravdivou**  **AŽ DO SPLNENIA PODMIENKY**  tento pokyn sa nebude opakovať |

# Vyriešené problémy

1) Bob je malý robot. Keďže je veľmi jednoduchý, vykonáva len 3 operácie:

* odbočiť doľava o 90°
* odbočiť doprava o 90°
* prejsť o jednu dlaždicu dopredu

Pomôžete Bobovi dostať sa z modrého dielika na fialový dielik, pričom sa budete pohybovať len po dielikoch zelenej farby? Napíšte dva algoritmy pre túto úlohu, jeden bez a druhý s použitím konštrukcií opakovania. Všimnite si, že žltá šípka označuje smer, ktorým sa Bob začína pozerať.



|  |  |
| --- | --- |
| **Riešenie bez opakovania** | **Riešenie s použitím konštrukcií opakovania** |
| prejsť o jednu dlaždicu dopredu  odbočiť doprava o 90°  prejsť o jednu dlaždicu dopredu  prejsť o jednu dlaždicu dopredu  prejsť o jednu dlaždicu dopredu  odbočiť doľava o 90°  prejsť o jednu dlaždicu dopredu  prejsť o jednu dlaždicu dopredu  odbočiť doľava o 90°  prejsť o jednu dlaždicu dopredu  prejsť o jednu dlaždicu dopredu  prejsť o jednu dlaždicu dopredu | prejsť o jednu dlaždicu dopredu  odbočiť doprava o 90°  PRE 3 ČASY DO  prejsť o jednu dlaždicu dopredu  odbočiť doľava o 90°  PRE 2 ČASY DO  prejsť o jednu dlaždicu dopredu  odbočiť doľava o 90°  PRE 3 ČASY DO  prejsť o jednu dlaždicu dopredu |

# Odkazy

[1] Böhm, C., & Jacopini, G. (1966). Vývojové diagramy, Turingove stroje a jazyky s iba dvoma pravidlami tvorby. *Commun. ACM, 9*, 366-371.

[2] druhý odkaz

1. Ikonu Moka kanvice vytvoril Dooder a stiahol ju z Flaticonu - "https://www.flaticon.com/free-icons/moka-pot" [↑](#footnote-ref-1)