1 Algoritmus, vlastnosti, větvení a cykly

Co je algoritmus?

Algoritmus je nejčastěji používán v programování. Jedná se o přesný teoretický postup při řešení daného problému. Dle této definice se za algoritmus dá považovat například i kuhařský recept. Z pravidla jsou na algoritmy ovšem kladena určitá omezení

Základní vlastnosti algoritmu

- Elementárnost
  - Algoritmus se skládá z konečného počtu jednoduchých (elementárních) kroků.
- Konečnost (finitnost)
  - Každý algoritmus musí skončit v konečném počtu kroků. Tento počet kroků může být libovolně velký (podle rozsahu a hodnot vstupních údajů), ale pro každý jednotlivý vstup musí být konečný.
- Obecnost
  - Algoritmus neřeší jeden konkrétní problém (např. „jak spočítat 3×7“), ale obecnou třídu obdobných problémů (např. „jak spočítat součin dvou celých čísel“), má širokou množinu možných vstupů.
- Determinovanost
  - Algoritmus je determinovaný, pokud za stejných podmínek (pro stejné vstupy) poskytuje stejný výstup. Vyjímku tvoří šifrovací nebo generovací algoritmy, které nesmí být determinovatelné, aby splnili svůj účel.
- Determinismus
  - Každý krok algoritmu musí být jednoznačně a přesně definován; v každé situaci musí být naprosto zřejmé, co a jak se má provést, jak má provádění algoritmu pokračovat.
- Výstup
  - Algoritmus má alespoň jeden výstup, veličinu, která je v požadovaném vztahu k zadaným vstupům, a tím tvoří odpověď na problém, který algoritmus řeší (algoritmus vede od zpracování hodnot k výstupu) Wikipedia

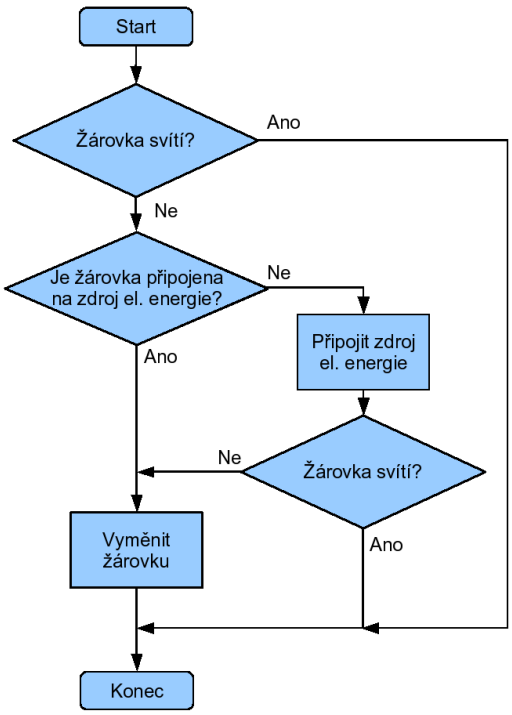
Vývojový diagram

Vývojový diagram je druh diagramu, který slouží ke grafickému znázornění jednotlivých kroků algoritmu, pracovního postupu nebo nějakého procesu. Vývojový diagram obsahuje obrazce různého tvaru (obdélníky, kosočtverce, aj.), navzájem propojené pomocí šipek. Obrazce reprezentují jednotlivé kroky, šipky tok řízení. Vývojové diagramy standardně nezobrazují tok dat, ten je zobrazován pomocí data flow diagramů. Vývojové diagramy jsou často využívány v informatice během programování pro analýzu, návrh, dokumentaci nebo řízení procesu.

- Šipka
  - Určuje směr zpracování
- Obdélník se zaoblenými rohy
  - Počátek nebo ukončení zpracování
- Kosočtverec
  - Větvení postupu v závislosti na splnění podmínky
- Obdélník s popisem
  - Definuje dílčí krok zpracování
- Kruh
  - Spojka několika šipek

Wikipedia

Například vývojový diagram pro kontrolu žárovky:



Řídicí struktura

Wikipedia Řídicí struktura (též strukturovaný příkaz, anglicky control flow statement) je v informatice konstrukce pro zápis počítačového programu. Řídicí struktury jsou používány ve vyšších programovacích a skriptovacích jazycích. Rozhodují o dalším provádění programu: větví jeho běh, vytváří cykly nebo jinak mění běh programu

Větvení

(či podmíněný příkaz) – v závislosti na splnění podmínky se určitý příkaz buď provede nebo neprovede

- IF
  - "POKUD" je splněna podmínka, tak se něco stane.

```
age = int(input("Věk> "))
if age >= 18:
    print("Můžeš pít alkohol")
```

- ELSE
  - Používá se s IF a určuje, co se má stát, pokud není splněna podmínka v IF

```
age = int(input("Věk> "))
if age >= 18:
    print("Můžeš pít alkohol")
else:
    print("Nemůžeš pít alkohol")
```

- ELSE IF
  - Používá se ve spojení s IF a ELSE. Umožňuje vytvořit víc podmínek, které spadají do stejného bloku => první, co je splňená je použita a zbytek je přeskočen. ELSE je použito pouze pokud není splněná žádná z předešlých podmínek

```
age = int(input("Věk> "))
if age > 19:
    print("Je ti víc, jak 19")
elif age == 18:
    print("Je ti přesně 18")
elif age == 19:
    print("Je ti přesně 19")
else:
    print("Je ti míň jak 18")
```

- SWITCH
  - Slouží k nahrazení většího množství ELSE IF příkazů
  - Python neměl do verze 3.10 SWITCH příkaz. Teď už ho má, ale nejmenuje se SWITCH freecodecamo

```
# kód zkopírován z "https://www.freecodecamp.org/news/python-switch-statement-switch-case-example/"
# komentáře jsem přidal já
lang = input("What's the programming language you want to learn? ")

# V tomto případě mi JS switch přijde praktičtější, proto ho budu přirovnávat k python verz

# v js -> switch(lang){
match lang:
  # v js -> case "Javascript":
    case "JavaScript":
        print("You can become a web developer.")
    # zde by v js musel být příkaz break;
    case "Python":
        print("You can become a Data Scientist")
    # zde by v js musel být příkaz break;
    case "PHP":
        print("You can become a backend developer")
    # zde by v js musel být příkaz break;
    case "Solidity":
        print("You can become a Blockchain developer")
    # zde by v js musel být příkaz break;
    case "Java":
        print("You can become a mobile app developer")
    # v js se přímo používá místo "case" slovo "default" pro označení kódu, který má proběhnout, pokud není splněna podmínka (stejně jako ELSE    v hromadě ELSE IF)
    # python používá "_" jako default
    case _:
        print("The language doesn't matter, what matters is solving problems.")
```

Cykly

Cyklus nebo také smyčka (angl. loop) je řídicí struktura počítačového programu, kde se opakovaně provádí posloupnost příkazů. Opakování i ukončení cyklu je řízeno nějakou podmínku.

- WHILE
    - Většinou se používá pro vytvoření nekonečných cyklů - například u her nebo programů.

```
while True:
    print("Baf")
```

- FOR
    - Většinou slouží k iteraci jednotlivých částí array listů nebo obecně pro zpracovávání dat přes které se dá iterovat, protože narozdíl od while loopu obsahuje promměnou, která udává číslo iterace

```
abeceda = ['a', 'b', 'c', 'd']

for i in range(len(abeceda)):
    print(f"{i+1}. písmeno v abecedě je: {abeceda[i]}")
```

- BREAK
    - Slouží k předčasnému ukončení cyklu (lze použít s FOR, WHILE i FOREACH cyklem)

```
abeceda = ['a', 'b', 'c', 'd']

for letter in abeceda:
    print(letter)
if letter == 'b':
    break
```

- CONTINUE
    - Slouží k přeskočení jedné iterace cyklu, opět lze použít se všemi druhy cyklů

```
abeceda = ['a', 'b', 'c', 'd']

for letter in abeceda:
    if letter == 'b':
        continue
    print(letter)
```

MATURITA

Co bude pravděpodobně potřeba:

- Vědět, co je algoritmus
- Znát jednotlivé vlastnosti algoritmu a umět je popsat
- Vědět, co je vývojový diagram. K čemu slouží. Jak vypadá a co znamenají jednotlivé značky. Umět ho nakreslit na základě algoritmu v kódu. Umět napsat kód na základě vývojového diagramu
- Znát jednotlivé řídící struktury. Co je větvení a cykly? Umět popsat jejich využití. Umět napsat jednoduchý program na tabuli (například pro výpis sudých čísel)
- Ukázka (viz složka FizzBuzz v python 1 lekce)

Úlohy z hodiny:

- Rozdělit číselné hodnoty na sudá, lichá čísla a čísla dělitelná 5

```
# Očekáván vstup: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
all_numbers = input("Vložte čísla (Oddělená \" \" (mezerou))>  ").split("
")

even_numbers = []
odd_numbers = []
divisible_by_five_numbers = []

for number in all_numbers:
    number = int(number)
    if number % 2 == 0:
        even_numbers.append(number)
    else:
      odd_numbers.append(number)
    if number % 5 == 0:
        divisible_by_five_numbers.append(number)

print(f"Sudá čísla: {even_numbers}")
print(f"Lichá čísla: {odd_numbers}")
print(f"Dělitelná pěti čísla: {divisible_by_five_numbers}")
```

- 1. Najdi ve 2D poli průměr všech hodnot a následně vypiš všechna čísla, jejichž vzdálenost od průměru je menší nebo rovná 3.
  2. U čísel vypiš jejich souřadnice v poli

```
#Input data
array_2D = [[1, 2, 3], [4, 5, 6], [7, 20, 9]]
number_distance = 3

#Starting values
number_count = 0
addition = 0
all_numbers = []
final_numbers = []

#Getting an average
for row in array_2D:
    for colum in row:
        number_count+=1
        addition+= colum
        all_numbers.append(colum)

average = addition/number_count

#Finding numbers in correct distance
for number in all_numbers:
    if abs(number-average) <= number_distance:
        final_numbers.append(number)

#Data printing
print("Vstupní údaje")
for row in array_2D:
    print(row)
print(f"Průměrná hodnota: {average}")
print(f"Čísla ve vzdálenosti {number_distance} od průměrné hodnoty:
{final_numbers}")

# Getting the cords
cords = ""
while True:
    if(len(final_numbers) == 0):
        break
    # first is a number, second is na array
    for row_cords, row in enumerate(array_2D):
        for colum_cords, colum in enumerate(row):
            if colum in final_numbers:
                final_numbers.remove(colum)
                cords += f"Číslo: {colum}, Souřadnice řádku {row_cords},
Souřadnice sloupce: {colum_cords}\n"
#Final output
print(cords)
```

- Hra, kde si počítač myslí číslo (1-100) a hráč jej má uhodnout s tím, že mu napovídá větší/menší (Program také počítá počet pokusů)

```
import random

random_number = -1
guess_count = 0

while True:
    try:
        if random_number == -1:
            random_number = random.randint(1, 100)
            guess_count = 0

        guess = int(input("Myslím si číslo od 1 do 100. Jaké?> "))
        guess_count+=1

        if guess == random_number:
            print(f"Správně, číslo bylo {random_number}.\n Uhádl jsi ho na
{guess_count}. pokus.")
            print("-----------------------------------")
            random_number = -1
        elif guess < random_number:
            print(f"Číslo {guess} je moc malé")
        elif guess > random_number:
            print(f"Číslo {guess} je moc velké")
    except Exception as error:
        print(f"You failed horribly {error}")
```