3. Jednoduché algoritmy

BAB37ZPR – Základy programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Iterativní algoritmy
  - P3.1 Ciferný součet
  - P3.2 Collatzova posloupnost
  - P3.3 Výpočet odmocniny – binární půlení
  - P3.3 Součet druhých mocnin
  - P3.5 Exponenciální funkce
- Část 2 – Algoritmy pracující s náhodnými čísly
  - P3.6 Simulace házení mincí
  - P3.6 Průměr diskrétní náhodné proměnné
  - P3.7 Simulace volebního průzkumu
  - P3.9 Kámen, nůžky, papír
  - P3.10 Výpočet π metodou Monte-Carlo

Část I

Iterativní algoritmy

Číselné typy

- `int` – celá čísla
- `float`
  - floating-point number
  - čísla s plovoucí desetinnou čárkou
  - reprezentace: mantisa $\times$ báze ^{exponent}
  - nepřesnosti, zaokrouhlování
- `complex` – komplexní čísla

Nepřesnosti

Přesná matematika:

$$((1 + \frac{1}{x}) - 1) * x = 1$$

Nepřesné počítače:

```
>>> x = 2**50
>>> ((1 + 1 / x) - 1) * x
1.0
>>> x = 2**100
>>> ((1 + 1 / x) - 1) * x
0.0
```

Číselné typy – poznámky

- Explicitní přetypování: `int(x)`, `float(x)`
- Automatické nafukování typu `int`
  - viz např. `2**100`
  - pomalejší, ale korektní
  - v ostatních programovacích jazycích zpravidla dochází k přetečení
- Záludné detaily

```
>>> round(2.5)
2
>>> round(4.5)
4
>>> round(2.675, 2)
2.67
```

https://docs.python.org/3/library/functions.html#round

Kvíz

```
>>> n = 1
>>> while n > 0:
...     print(n)
...     n = n / 10
```

- Co udělá program?
- Co když změníme výraz na `n=n*10`
- Co když změníme výraz na `n=n*10.0`
- Jaký bude výsledek programu v jiných jazycích?

I. Iterativní algoritmy

P3.1 Ciferný součet

P3.2 Collatzova posloupnost

P3.3 Výpočet odmocniny – binární půlení

P3.3 Součet druhých mocnin

P3.5 Exponenciální funkce

P3.1 Ciferný součet

- vstup: číslo `x`
- výstup: ciferný součet čísla `x`
- příklady:
  - $8 \to 8$
  - $15 \to 6$
  - $297 \to 18$

Jak na to?

- opakovaně provádíme:
  - dělení 10 se zbytkem – hodnota poslední cifry
  - celočíselné dělení – okrajování čísla

P3.1 Ciferný součet – naivní řešení

```
>>> def pripocti (n):
...     f = 0
...     if n % 10 == 0:
...         f = 0 + f
...     elif n % 10 == 1:
...         f = 1 + f
...     elif n % 10 == 2:
...         f = 2 + f
...     return f
...
>>> n = 12; f = 0
>>> f += pripocti(n)
>>> f += pripocti(n//10)
>>> print('Ciferný součet čísla', n, 'je', f)
Ciferný součet čísla 12 je 3
```

P3.1 Ciferný součet – řešení

```
>>> def digit_sum(n):
...     result = 0
...     while n > 0:
...         result += n % 10
...         n = n // 10
...     return result
...
```

Pro zajímavost

```
>>> def digit_sum(n):
...     return sum(map(int, str(n)))
...
```

Připomenutí

- `print` – výpis hodnoty
- `return` – návratová hodnota funkce, se kterou lze dále pracovat

I. Iterativní algoritmy

P3.1 Ciferný součet

P3.2 Collatzova posloupnost

P3.3 Výpočet odmocniny – binární půlení

P3.3 Součet druhých mocnin

P3.5 Exponenciální funkce

P3.2 Collatzova posloupnost

Collatzův problém lze shrnout následovně:

- Vezměme jakékoliv kladné celé číslo `n`.
- Pokud je `n` sudé číslo, vydělíme jej dvěma, získáme tak `n/2`.
- Pokud je `n` liché číslo, vynásobí se třemi a přičte se jednička, tj. `3n+1`.
- Tento postup opakujeme

Domněnka

- Domněnka je taková, že nezáleží na tom, jaké počáteční číslo `n` je zvoleno — výsledná posloupnost vždy nakonec dojde k číslu `1`
- Experimentálně ověřeno pro velká `n` ($\sim 10^{18}$)
- Důkaz není znám

P3.2 Collatzova posloupnost – řešení

```
>>> def collatz_sequence(n):
...     while n != 1:
...         print(n, end=", ")
...         if n % 2 == 0:
...             n = n // 2
...         else:
...             n = 3*n + 1
...     print(1)
...
>>> collatz_sequence(10)
10, 5, 16, 8, 4, 2, 1
>>> collatz_sequence(11)
11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

I. Iterativní algoritmy

P3.1 Ciferný součet

P3.2 Collatzova posloupnost

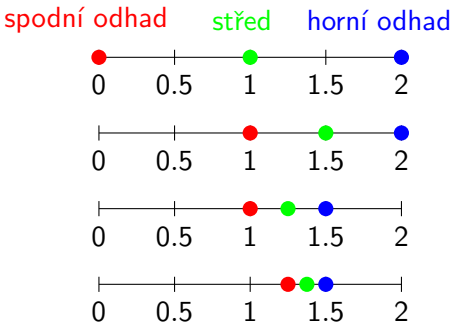
P3.3 Výpočet odmocniny – binární půlení

P3.3 Součet druhých mocnin

P3.5 Exponenciální funkce

P3.3 Výpočet odmocniny – binární půlení

- Vstup: číslo x, přesnost odhadu
- Výstup: odhad $\sqrt{x}$
- Jak na to? Existuje mnoho metod, ukázka metody binárního půlení.



1. Horní odhad nastaven na x, spodní odhad na 0
2. Pokud je druhá mocnina středu intervalu větší než x, horní odhad je nastaven na střed
3. Pokud je druhá mocnina středu intervalu menší než x, spodní odhad je nastaven na střed
4. Postup opakujeme, dokud není dosaženo žádané přesnosti

```
>>> def square_root(x, precision=0.05):
...     upper = x
...     lower = 0
...     middle = (upper + lower) / 2
...     while abs(middle**2 - x) > precision:
...         print(lower, upper, sep='-', end=', ')
...         if middle**2 > x:
...             upper = middle
...         if middle**2 < x:
...             lower = middle
...         middle = (upper + lower) / 2
...     return middle
...
>>> print(square_root(2))
0-2, 1.0-2, 1.0-1.5, 1.25-1.5, 1.375-1.5, 1.40625
```

```
>>> def square_root(x, precision=0.05):
...     upper = x
...     lower = 0
...     middle = (upper + lower) / 2
...     while abs(middle**2 - x) > precision:
...         print(lower, upper, sep='-', end=', ')
...         if middle**2 > x:
...             upper = middle
...         if middle**2 < x:
...             lower = middle
...         middle = (upper + lower) / 2
...     return middle
...
>>> print(square_root(2))
0-2, 1.0-2, 1.0-1.5, 1.25-1.5, 1.375-1.5, 1.40625
```

Drobný problém

- Co program udělá pro čísla $x < 1$
- Proč?
- Jak to opravit?

I. Iterativní algoritmy

P3.1 Ciferný součet

P3.2 Collatzova posloupnost

P3.3 Výpočet odmocniny – binární půlení

P3.3 Součet druhých mocnin

P3.5 Exponenciální funkce

P3.3 Součet druhých mocnin

- Lze napsat zadané číslo jako součet druhých mocnin?
- Příklad: $13 = 2^2 + 3^2$

```
>>> def sum_of_squares_test(n):
...     for i in range(n+1):
...         for j in range(n+1):
...             if i**2 + j**2 == n:
...                 print(n, "=", i**2, "+", j**2)
...
>>> sum_of_squares_test(13)
13 = 4 + 9
13 = 9 + 4
>>> sum_of_squares_test(29)
29 = 4 + 25
29 = 25 + 4
```

P3.3 Součet druhých mocnin

- Lze napsat zadané číslo jako součet druhých mocnin?
- Příklad: $13 = 2^2 + 3^2$

```
>>> def sum_of_squares_test(n):
...     for i in range(n+1):
...         for j in range(n+1):
...             if i**2 + j**2 == n:
...                 print(n, "=", i**2, "+", j**2)
...
>>> sum_of_squares_test(13)
13 = 4 + 9
13 = 9 + 4
>>> sum_of_squares_test(29)
29 = 4 + 25
29 = 25 + 4
```

Je program efektivní

- Zřejmě stačí menší počet iterací – kolik?
- Když je nalezena správná hodnota první mocniny, může trvat dlouho, než je nalezeno druhé číslo
- Jak to vylepšit?

I. Iterativní algoritmy

P3.1 Ciferný součet

P3.2 Collatzova posloupnost

P3.3 Výpočet odmocniny – binární půlení

P3.3 Součet druhých mocnin

P3.5 Exponenciální funkce

P3.5 Exponenciální funkce

Ověřte, že pro dostatečně velké `n` platí

$$e^x \approx \underbrace{\sum_{i=0}^{n} \frac{x^i}{i!}}_{A_n(x)}$$

Problém rozdělíme na podproblémy:

- Výpočet faktoriálu
- Výpočet součtů $A_n(x)$
- Tisk chyby pro různá n
- Tisk chyby pro různá x

P3.5 Faktoriál

```
>>> def factorial(n):
...     prod=1
...     for i in range(2,n+1):
...         prod*=i
...     return prod
...
>>> factorial(5)
120
```

P3.5 Součet řady

$$A_n(x) = \sum_{i=0}^{n} \frac{x^i}{i!}$$

```
>>> import math
>>> def series_sum(x,n):
...     sum=0.
...     for i in range(n+1):
...         sum+=pow(x,i)/factorial(i)
...     return sum
...
>>> math.exp(1.0)
2.718281828459045
>>> series_sum(1.0,10)
2.7182818011463845
```

P3.5 Vyhodnocení přesnosti

```
>>> def print_accuracy(x):
...     exact = math.exp(x)
...     print("x=%5d  exact=%10g" % (x,exact))
...     for n in [5,10,100]: # smycka pres seznam
...         approx=series_sum(x,n)
...         relerr=abs(exact-approx)/exact
...         print("n=%5d approx=%10g relerr=%10g" % (n,approx,relerr))
...
>>> print_accuracy(1.0)
x=    1  exact=   2.71828
n=    5 approx=   2.71667 relerr=0.000594185
n=   10 approx=   2.71828 relerr=1.00478e-08
n=  100 approx=   2.71828 relerr=1.63371e-16
```

lec03/exponencial.py

Část II

Algoritmy s náhodnými čísly

Zdroj dat – náhodná čísla

- Přesněji: pseudo-náhodná čísla, opravdová náhodná čísla: `https://www.random.org/`
- Bohaté využití v programování: výpočty, simulace, hry, ...
- Python – modul `random`
  - `random.random()` – float od 0 do 1
  - `random.randint(a, b)` – celé číslo mezi a, b
  - `random.randrange(a, b)` – náhodná celá čísla v rozsahu a, b

```
>>> import random
>>> n=10
>>> random.randrange(0,n)
0
>>> random.randrange(0,n)
3
>>> random.randrange(0,n)
2
```

II. Algoritmy s náhodnými čísly

P3.6 Simulace házení mincí

P3.6 Průměr diskrétní náhodné proměnné

P3.7 Simulace volebního průzkumu

P3.9 Kámen, nůžky, papír

P3.10 Výpočet π metodou Monte-Carlo

P3.6 Simulace házení mincí 1/2

```
>>> import random

>>> def hod():
...     if random.randrange(2)==0:
...         print("Hlava", end=" ")
...     else:
...         print("Orel", end=" ")
...
>>> for i in range(10):
...     hod()
...
Hlava Orel Orel Hlava Orel Orel Orel Orel Hlava Hlava
```

- Trochu jiný zápis funkce

```
>>> # trochu jiny zapis funkce
>>> def hod1():
...     text = "Orel" if random.randrange(2) % 2 else "Hlava"
...     print(text, end=" ")
...
```

II. Algoritmy s náhodnými čísly

P3.6 Simulace házení mincí

P3.6 Průměr diskrétní náhodné proměnné

P3.7 Simulace volebního průzkumu

P3.9 Kámen, nůžky, papír

P3.10 Výpočet π metodou Monte-Carlo

P3.7 Průměr diskrétní náhodné proměnné

- Vybíráme náhodně čísla $x_i \in \{1, 2, \ldots, 10\}$
- Jaký bude jejich průměr pro velké N?

$$\frac{1}{N} \sum_{i=1}^{N} x_i$$

```
>>> import random

>>> N=10000
>>> s=0.
>>> for i in range(N):
...     s+=random.randrange(1,11)
...
>>> print("average =", s/N)
average = 5.5057
```

II. Algoritmy s náhodnými čísly

P3.6 Simulace házení mincí

P3.6 Průměr diskrétní náhodné proměnné

P3.7 Simulace volebního průzkumu

P3.9 Kámen, nůžky, papír

P3.10 Výpočet π metodou Monte-Carlo

P3.8 Simulace volebního průzkumu 1/2

- Volební průzkumy se často liší – jaká je jejich přesnost?
- Přístup 1: matematické modely, statistika
- Přístup 2: simulace
- Program:
  - vstup: preference stran, velikost vzorku
  - výstup: preference zjištěné v náhodně vybraném vzorku

```
>>> def pruzkum(size, pref1, pref2, pref3):
...     count1 = 0
...     count2 = 0
...     count3 = 0
...     for i in range(size):
...         r = random.randint(1, 100)
...         if r <= pref1: count1 += 1
...         elif r <= pref1 + pref2: count2 += 1
...         elif r <= pref1 + pref2 + pref3: count3 += 1
...     print("Strana 1:", 100 * count1 / size)
...     print("Strana 2:", 100 * count2 / size)
...     print("Strana 3:", 100 * count3 / size)
...
```

- řešení není dobré: funguje jen pro 3 strany
- lepší řešení – využití seznamů

II. Algoritmy s náhodnými čísly

P3.6 Simulace házení mincí

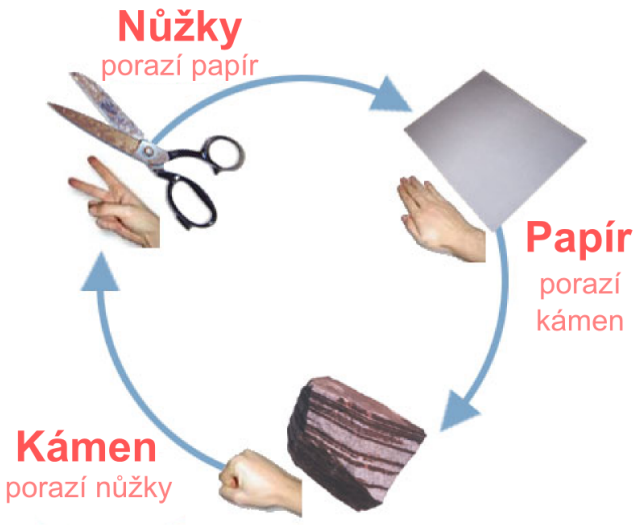
P3.6 Průměr diskrétní náhodné proměnné

P3.7 Simulace volebního průzkumu

P3.9 Kámen, nůžky, papír

P3.10 Výpočet π metodou Monte-Carlo

P3.9 Kámen, nůžky, papír



P3.9 Kámen, nůžky, papír – volba strategie

- hráč náhodně zkouší štěstí podle rovnoměrného rozdělení pravděpodobnosti

```
>>> def strategy_uniform():
...     r = random.randint(1, 3)
...     if r == 1:
...         return "R"
...     elif r == 2:
...         return "S"
...     else:
...         return "P"
...
```

- hráč zkouší pouze jednu možnost – kámen

```
>>> def strategy_rock():
...     return "R"
...
```

P3.9 Kámen, nůžky, papír – vyhodnocení tahu

```
>>> def evaluate(symbol1, symbol2):
...     if symbol1 == symbol2:
...         return 0
...     if symbol1 == "R" and symbol2 == "S" or \
...        symbol1 == "S" and symbol2 == "P" or \
...        symbol1 == "P" and symbol2 == "R":
...         return 1
...     return -1
...
>>> evaluate('R', 'R')     # nevítězí nikdo
0
>>> evaluate('S', 'R')     # vítězí počítač (nůžky < kámen)
-1
>>> evaluate('P', 'R')     # vítězí hráč (papír > kámen)
1
```

P3.9 Kámen, nůžky, papír – tah

```
>>> def game(rounds):
...     points = 0
...     for i in range(1, rounds+1):
...         print("Round ", i)
...         symbol1 = strategy_uniform()
...         symbol2 = strategy_uniform()
...         print("Symbols:", symbol1, symbol2)
...         points += evaluate(symbol1, symbol2)
...         print("Player 1 points:", points)
...
>>> game(1)
Round  1
Symbols: R P
Player 1 points: -1
```

P3.9 Kámen, nůžky, papír – obecnější strategie

```
>>> def strategy(weightR, weightS, weightP):
...     r = random.randint(1, weightR + weightS + weightP)
...     if r <= weightR:
...         return "R"
...     elif r <= weightR + weightS:
...         return "S"
...     else:
...         return "P"
...
```

P3.9 Kámen, nůžky, papír – obecnější strategie

```
>>> def strategy(weightR, weightS, weightP):
...     r = random.randint(1, weightR + weightS + weightP)
...     if r <= weightR:
...         return "R"
...     elif r <= weightR + weightS:
...         return "S"
...     else:
...         return "P"
...
```

Jak hru ještě vylepšit?

- turnaj různých strategií
- strategie pracující s historií
  - kopírování posledního tahu soupeře
  - analýza historie soupeře (hraje vždy kámen? → hraj papír!)
- rozšíření na více symbolů

II. Algoritmy s náhodnými čísly

P3.6 Simulace házení mincí

P3.6 Průměr diskrétní náhodné proměnné

P3.7 Simulace volebního průzkumu

P3.9 Kámen, nůžky, papír

P3.10 Výpočet π metodou Monte-Carlo

P3.10 Výpočet π metodou Monte-Carlo

```
>>> import random
>>> def mcpi(interval):
...     circle_points = 0
...     square_points = 0
...
...     for i in range(interval**2):
...         rand_x = random.uniform(-1, 1)
...         rand_y = random.uniform(-1, 1)
...         origin_dist = rand_x**2 + rand_y**2
...         # Checking if (x, y) lies inside the circle
...         if origin_dist <= 1:
...             circle_points += 1
...         square_points += 1
...
...     return 4 * circle_points / square_points
...
```