# NMIN112 Programování 2 2/4 Z+Zk 8 kr.

Pavel Töpfer
Katedra softwaru a výuky informatiky MFF UK
MFF Malostranské nám., 4. patro, pracovna 404

pavel.topfer@mff.cuni.cz
https://ksvi.mff.cuni.cz/~topfer

**Algoritmizace**
- metody řešení úloh na počítači
- algoritmy, datové struktury, programovací techniky

**Správnost algoritmu**
- konečnost
- parciální korektnost (správné výsledky, když výpočet skončí)

**Efektivita algoritmu** (složitost algoritmu)
- časová (počet vykonaných operací)
- prostorová (pracovní paměť potřebná na uložení dat)

# Učivo

- algoritmus, časová a prostorová složitost
- dělitelnost čísel, Eukleidův algoritmus
- test prvočíselnosti, Eratosthenovo síto
- rozklad čísla na cifry
- aritmetika s vyšší přesností („dlouhá čísla“), polynomy
- Hornerovo schéma, poziční číselné soustavy
- algoritmy vyhledávání v poli (sekvenční, binární, zarážka)
- třídění čísel v poli - přímé metody, heapsort, mergesort, quicksort
- složitost problému třídění
- přihrádkové metody třídění
- vnější třídění
- reprezentace dat v paměti
- zásobník, fronta, halda, slovník
- spojový seznam

## **Učivo** (pokračování)

- rekurze – princip, příklady, efektivita, rekurzivní generování
- binární a obecný strom – reprezentace, průchod, použití
- binární vyhledávací strom, princip vyvažování
- notace aritmetického výrazu – vyhodnocení, převody
- reprezentace grafu
- prohledávání grafu, základní grafové algoritmy
- prohledávání stavového prostoru do hloubky a do šířky
- metody zrychlení backtrackingu – ořezávání, heuristiky
- programování her, algoritmus minimaxu
- metoda Rozděl a panuj
- dynamické programování
- metody ukládání dat, hešovací tabulka

# **Studijní zdroje** (přednášky)

**Prezentace z přednášek**
- po každé přednášce na webu přednášejícího
- materiály často rozšířené a doplněné oproti přednášce
- ukázkové programy v Pythonu

# **Studijní zdroje** (algoritmy)

**Pavel Töpfer: Algoritmy a programovací techniky**
Prometheus Praha 1995, 2. vydání 2007
starší text, ale obsahově téměř odpovídá (programové ukázky v Pascalu)
tištěná kniha v knihovnách, nyní k zakoupení pouze jako e-kniha
https://www.prometheus-eknihy.cz/

**Programátorské kuchařky KSP**
krátké studijní texty k jednotlivým tématům algoritmizace a programování
http://ksp.mff.cuni.cz/kucharky/

**Martin Mareš, Tomáš Valla: Průvodce labyrintem algoritmů**
CZ.NIC Praha 2017
text pdf zdarma ke stažení
http://pruvodce.ucw.cz/
https://knihy.nic.cz/files/edice/pruvodce_labyrintem_algoritmu.pdf

# **Studijní zdroje** (Python)

**Mark Pilgrim: Ponořme se do Python(u) 3**
CZ NIC Praha 2011
text pdf zdarma ke stažení
https://knihy.nic.cz/files/edice/python_3.pdf

odkazy na různé on-line materiály na webu
https://python.cz/zacatecnici/

například kurz PyLadies
https://naucse.python.cz/course/pyladies/

# Zápočet

- uděluje cvičící („praktický“ po souhlasu „teoretického“)
- domácí úkoly z teoretické i z praktické části cvičení
- praktický zápočtový test
- zápočtový program
- další požadavky cvičícího (účast na cvičeních, práce na cvičeních)

# Zkouška

- povinná písemná a doplňková ústní část
- přihlašování prostřednictvím SISu
- k účasti na zkoušce není nutné předchozí získání započtu

# Algoritmus

Intuitivní pojem, v literatuře mnoho různých definic, například:

„Konečná posloupnost elementárních příkazů, jejichž provádění umožňuje pro každá přípustná vstupní data mechanickým způsobem získat po konečném počtu kroků příslušná výstupní data.“

*Typické vlastnosti (nemusí platit vždy):*

- konečnost
- hromadnost
- resultativnost
- jednoznačnost
- determinismus

## Formální modely algoritmu

rekurzivní funkce (Kurt Gödel, 1934)
Turingův stroj (Alan Turing, 1936)
lambda kalkul (Alonzo Church, 1941)
RAM počítač (Random access machine)

Churchova-Turingova teze:

*Ke každému algoritmu existuje ekvivalentní Turingův stroj.*

## Popis a zápis algoritmu

slovní popis v přirozeném jazyce
pseudokód
program (zjednodušené konstrukce programovacího jazyka)

# Největší společný dělitel

X, Y – dvě kladná celá čísla

→ určit největší společný dělitel NSD(X, Y)

**Algoritmy:**

1. NSD(X, Y) = největší z celých čísel od 1 do *min(X*, Y), které je dělitelem obou čísel X a Y

– postupně zkoušet, nejlépe v pořadí od *min(X*, Y) dolů k 1, do nalezení prvního takového společného dělitele

2. Nalézt prvočíselné rozklady čísel X a Y

– jejich maximální společná část určuje NSD(X, Y)

*Příklad:*

396 = 2.2.3.3.11, 324 = 2.2.3.3.3.3

$\rightarrow$ NSD(396, 324) = 2.2.3.3 = 36

## 3. **Eukleidův algoritmus**

*Eukleidés: Základy (řecky Stoicheia, 13 knih), cca 300 př.n.l.*

*Idea:*

| | | |
|---|---|---|
| | když X < Y | NSD(X, Y) = NSD(X, Y-X) |
| | když X > Y | NSD(X, Y) = NSD(X-Y ,Y) |
| | když X = Y | NSD(X, Y) = X |

*Příklad:*

NSD( 396, 324) =
NSD( 72, 324) =
NSD( 72, 252) =
NSD( 72, 180) =
NSD( 72, 108) =
NSD( 72, 36) =
NSD( 36, 36) = 36

*Algoritmus (pro kladná celá čísla X, Y):*

dokud X ≠ Y dělej
   od většího z čísel X, Y odečti menší z čísel X, Y

```
while x != y:
    if x > y:
        x -= y
    else:
        y -= x
print(x)
```

*Možnost urychlení (pro některé vstupní hodnoty):*
místo odčítání použít zbytek po celočíselném dělení

```
while y > 0:
    z = x % y
    x = y
    y = z
print(x)
```

Program funguje i v případě X < Y, jenom vykoná o jednu iteraci více a při první iteraci se hodnoty X, Y navzájem prohodí.

```
while y > 0:
    x, y = y, x % y
print(x)
```

**Správnost Eukleidova algoritmu**

*1. Konečnost*

= *výpočet pro jakákoliv vstupní data skončí*

- na začátku výpočtu i stále v jeho průběhu je X > 0, Y > 0

- v každém kroku výpočtu se hodnota X+Y sníží alespoň o 1

$\rightarrow$ nejpozději po X+Y krocích výpočet skončí, je tedy konečný

*2. Parciální (částečná) správnost*
*= když výpočet skončí, vydá správný výsledek*

- pro X = Y zjevně platí NSD(X, Y) = X

- ukážeme, že pro X > Y platí NSD(X, Y) = NSD(X-Y, Y)

*Důkaz sporem:*
Nechť N = NSD(X, Y), tedy N dělí X a zároveň N dělí Y.
Proto také N dělí X-Y a je tedy N společným dělitelem X-Y a Y.
Pokud by neplatilo, že N = NSD(X-Y, Y), musí existovat A > 1 tak, že N.A = NSD(X-Y, Y).
Tedy N.A dělí X-Y i Y, takže N.A dělí i jejich součet, což je X.
Jelikož N.A dělí Y a zároveň N.A dělí X, je N.A společným dělitelem čísel X, Y, což je spor s předpokladem, že N = NSD(X, Y).
Proto skutečně N = NSD(X-Y, Y).

# Efektivita algoritmu (složitost algoritmu)

- **časová**

  počet vykonaných operací *(kterých?)*
  → rychlost výpočtu programu

- **prostorová (paměťová)**

  velikost datových struktur využívaných algoritmem
  → paměť potřebná na uložení dat při výpočtu programu

Kritéria pro hodnocení kvality algoritmu a programu
(příp. praktické použitelnosti programu).

Obě kritéria mohou mířit proti sobě (nelze najednou optimalizovat paměť i čas), musíme zvolit, čemu dáme přednost (dnes obvykle čas).

***„výměna času za paměť“***

***Funkce*** vyjadřující počet vykonaných operací (resp. velikost potřebné paměti) v závislosti na velikosti vstupních dat. Jako velikost vstupních dat obvykle stačí uvažovat např. počet zpracovaných čísel, nikoliv konkrétní hodnoty (jedná-li se o hodnoty „standardní“ velikosti). Obecně by se musela uvažovat délka zápisu vstupních dat v bitech.

Funkce časové a prostorové složitosti jsou většinou ***rostoucí*** (nad většími daty bývá výpočet delší).

*Jaké operace započítat:*
- elementární, vyžadující konstantní čas
- aritmetické, logické, přiřazení
- typické pro řešený problém (převažující)

*Kterou paměť započítat:*
- do prostorové složitosti nepočítáme paměť potřebnou na uložení vstupních dat, pokud z ní data pouze čteme (neměníme její obsah)

Přesné vyjádření funkce časové složitosti (např. $3.N^2 + 2.N - 4$)
je jednak obtížné, jednak většinou zbytečné.
Podstatná je řádová rychlost růstu této funkce pro rostoucí $N$.

Zanedbáme pomaleji rostoucí členy a konstanty
→ **asymptotická časová složitost** – zapisujeme např. $O(N^2)$

**Symbol „velké O“**
f, g: **N** → **R**
$f \in O(g) \Leftrightarrow \exists\, c > 0 \;\; \exists\, n_0 > 0 \;\forall\, n \geq n_0 : \;\; 0 \leq f(n) \leq c.g(n)$

tzn. funkce f se dá **shora** odhadnout funkcí g
(až na multiplikativní konstantu a pro dostatečně velká n)

Opačný odhad **zdola**: $f \in \Omega(g)$ $\quad 0 \leq c.g(n) \leq f(n)$
**Přesný** (těsný) odhad: $f \in \Theta(g) \Leftrightarrow f \in O(g) \;\&\; f \in \Omega(g)$

*Poznámka:*

Funkce $3.N^2 + 2.N - 4$ patří nejen do třídy $O(N^2)$, ale podle definice také třeba do třídy $O(N^5)$. Odhad složitosti $O(N^5)$ je zde sice správný, ale zbytečně hrubý a nepřesný, vždy se snažíme o co nejlepší (nejnižší) horní odhad složitosti.

Proto je pro nás cennější, ale také obtížnější, odvodit těsný odhad asymptotické časové složitosti algoritmu $\Theta(N^2)$ než jenom horní odhad $O(N^2)$.