

# Základy algoritmizace

## 4. Problémy, algoritmy, data

doc. Ing. Jiří Vokřínek, Ph.D.

Katedra počítačů

Fakulta elektrotechnická

České vysoké učení technické v Praze

# Základy algoritmizace

- Dnes:
  - Řešení problému a algoritmus
  - Rozklad problému na podproblémy
  - Pole

# Výpočet největšího společného dělitele

## ■ Úloha

Najděte největší společný dělitele přirozených čísel  $x$  a  $y$ .

## ■ Řešení

- **Vstup:** dvě přirozená čísla  $x$  a  $y$
- **Výstup:** přirozené číslo  $d$  – největší společný dělitel  $x$  a  $y$
- **Základní varianta:**

```
def getGreatestCommonDivisor(x, y):  
    if (x < y) :  
        d = x  
    else:  
        d = y  
    while ((x % d != 0) or (y % d != 0)):  
        d = d - 1  
    return d
```

*185110 iterací cyklu while*

# Výpočet největšího společného dělitele

- Místo výběru menšího z čísel  $x$  a  $y$  vnoříme do těla hlavního cyklu dva cykly zmenšující hodnoty aktuálních hodnot  $x$  a  $y$

```
def getGreatestCommonDivisorLoops(x, y):  
    while (x != y):  
        while (x > y):  
            x -= y  
        while (y > x):  
            y -= x  
    return x
```

*17650 iterací vnitřních cyklů while*

- Vnitřní cykly počítají nenulový zbytek po dělení většího čísla menším

# Výpočet největšího společného dělitele

- Vnitřní cykly můžeme nahradit přímým výpočtem zbytku po dělení

```
def getGreatestCommonDivisorEuclid(x, y):  
    remainder = x % y  
    while (remainder != 0):  
        x = y  
        y = remainder  
        remainder = x % y  
    return y
```

*4 iterace cyklu while*

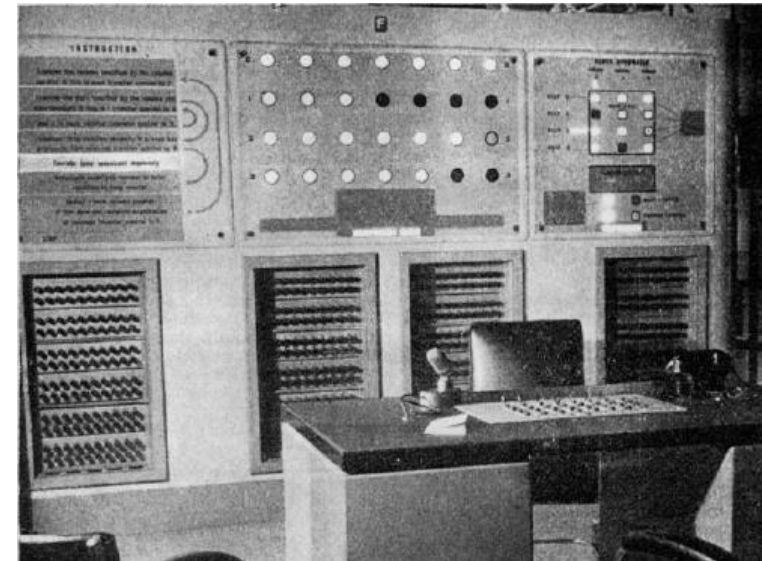
- **Euklidův algoritmus**

- Určíme zbytek po dělení daných čísel
- Zbytkem dělíme dělitele a určíme nový zbytek, až dosáhneme nulového zbytku
- Poslední nenulový zbytek je největší společný dělitel

# Jak najít správný algoritmus?

# Rozklad problému na podproblémy

- Postupný návrh programu rozkladem na podproblémy
  - Zadaný problém rozložíme na podproblémy
  - Pro řešení podproblémů zavedeme **abstraktní příkazy**
  - S abstraktními příkazy sestavíme hrubé řešení
  - Abstraktní příkazy realizujeme jako funkce
  
- Příklad – hra NIM
  - „První“ počítačová hra (1951)
  - Hra dvou hráčů v odebírání herních kamenů (nebo zápalek)







# Hra NIM

## ■ Pravidla

- Hráč zadá počet herních kamenů
- Pak se střídá se strojem v odebírání
  - Lze odebrat 1, 2, nebo 3 kameny
- Prohraje ten, kdo odebere poslední herní kámen

## ■ Dílčí problémy

- Zadání počtu kamenů
- Odebrání kamenů hráčem
- Odebrání kamenů strojem



# Příklad průběhu hry

- Zadej počet kamenů (15 až 35): 18
- Stav hry
  - Počet kamenů ve hře
  - Kdo je na tahu – počítač (P) nebo hráč (H)
- Průběh
  1. Počet kamenů 18; Kolik odeberete? **1** (H)
  2. Počet kamenů 17; Odebírám **1** (P)
  3. Počet kamenů 18; Kolik odeberete? **3** (H)
  4. Počet kamenů 17; Odebírám **1** (P)
  5. Počet kamenů 18; Kolik odeberete? **3** (H)
  6. Počet kamenů 17; Odebírám **1** (P)
  7. Počet kamenů 18; Kolik odeberete? **3** (H)
  8. Počet kamenů 17; Odebírám **1** (P)
  9. Počet kamenů 18; Kolik odeberete? **3** (H)
  10. Počet kamenů 17; Odebírám **1** (P)
- Hráč vyhrál, počítač odebral poslední kámen

# NIM – pravidla pro odebírání

- Počet kamenů nevýhodných pro protihráče je 1, 5, 9, ...
- Obecně  $4n + 1$ , kde  $n$  je celé nezáporné číslo
- Stroj musí z počtu kamenů  $K$  odebrat  $x$  kamenů, aby platilo

$$K - x == 4n + 1$$

- $x == (K - 1) - 4n$ , tj. hledáme zbytek po dělení 4
- $x = (K - 1) \% 4$
- Je-li  $x == 0$ , je okamžitý počet kamenů pro stroj nevýhodný a bude-li protihráč postupovat správně, stroj prohraje

# NIM – hrubý návrh řešení

```
numberOfStones = getNumberOfStones(limits)
machine = True
```

```
while (numberOfStones):
    if machine:
        machineTurn()
    else:
        humanTurn()
    machine = not machine
if machine: print("Machine wins!")
else: print("Human wins!")
```

- Podproblémy `getNumberOfStones`, `machineTurn` a `humanTurn` reprezentují abstraktní příkazy, které implementujeme jako funkce vracující počet kamenů k odebrání

# NIM – podrobnější návrh

- Funkce na zadání počtu kamenů v rámci limitu
  - Ošetříme vstup na rozsah hodnot
  - Možno ošetřit proti špatnému zadání

```
def getNumberOfStones(min, max):  
    n = int(input("Enter number of stones in range [  
                  + str(min) + ", " + str(max) + "]: "))  
  
    if n < min: return min  
    if n > max: return max  
    return n
```

# NIM – podrobnější návrh

- Funkce pro tah hráče
  - Akceptujeme jen validní tahy – 1, 2 nebo 3 kameny

```
def humanTurn(n):  
    r = ""  
    while (r != "1" and r != "2" and r != "3"):  
        r = input("Number of stones is "  
                + str(n) +  
                ". How many stones you will take: ")  
    return int(r)
```

# NIM – podrobnější návrh

- Funkce pro tah stroje
  - Pomocí výpočtu optimální strategie

```
def machineTurn(n):  
    r = (n-1) % 4  
    if r == 0: r = 1  
    print("Number of stones is "  
          + str(n) + ". I take: " + str(r))  
    return r
```

# NIM – podrobnější návrh

- Finální program

- Vyzkoušejte si naprogramovat a otestovat pro případ, kdy začíná počítač nebo člověk

*Kdo vyhrává při aplikování optimální strategie?*

```
MIN_S = 15
```

```
MAX_S = 35
```

```
machine = True
```

```
numberOfStones = getNumberOfStones(MIN_S, MAX_S)
```

```
while (numberOfStones > 0):
```

```
    if machine:
```

```
        numberOfStones -= machineTurn(numberOfStones)
```

```
    else:
```

```
        numberOfStones -= humanTurn(numberOfStones)
```

```
    machine = not machine
```

```
if machine: print("Machine wins!")
```

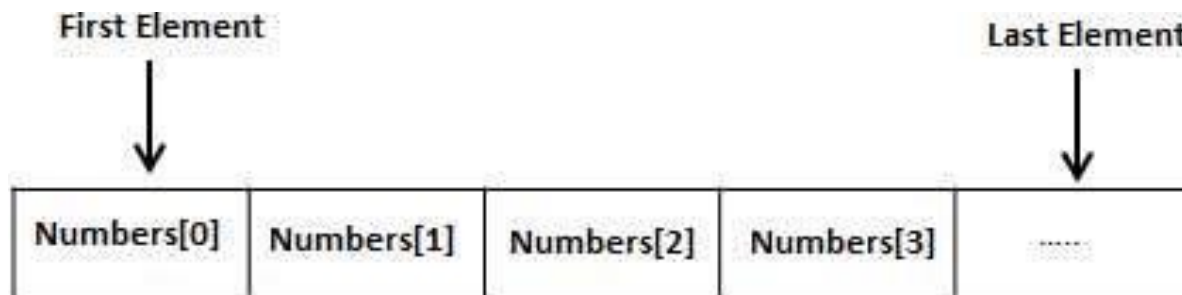
```
else: print("Human wins!")
```



# Když proměnná nestačí

# Stav hry

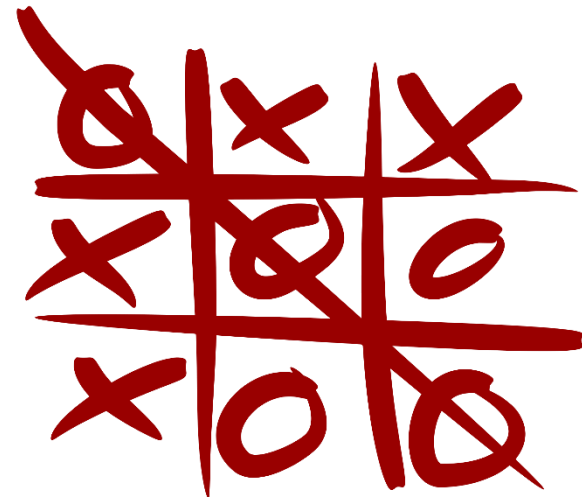
- U hry NIM je stav udržován v jedné proměnné
- Varianta „multiple-heap“ musí udržovat více stavových proměnných
  - heap1 = 10
  - heap2 = 20
  - heap3 = 30
- Co když máme tisíc hromádek?
  - Použijeme pole



- Vícerozměrná data?

# Pole

- Vícerozměrná data
  - „pole polí“
  - Používáme seznamy, či tuple
- Matice, šachovnice, ...
- Příklad reprezentace – piškvorky (tic-tac-toe)



# Tic Tac Toe

- Jak se liší od hry NIM?
  - Reprezentace dat
  - Složitější výpočet strategie
  - Test konce hry
  - Test přípustnosti tahu
  - ...

```
sizeOfGame = getSizeOfGame(limits)
machine = True
while (not gameFinished()):
    if machine:
        machineTurn()
    else:
        humanTurn()
    machine = not machine
if machine: print("Machine wins!")
else: print("Human wins!")
```

# Tic Tac Toe

## ■ Příklad generování hry

```
def getGame():
    n = int(input("Enter size of the game (minimum 3): "))
    if n < 3: n = 3
    game = []
    for x in range(n):
        row = []
        for y in range(n):
            row.append("-")
        game.append(row)
    return game
```

## ■ Nebo

```
def getGame():
    n = int(input("Enter size of the game (minimum 3): "))
    if n < 3: n = 3
    return [ ["-" for x in range(n)] for x in range(n) ]
```

# Tic Tac Toe

## ■ Příklad vykreslování hry

```
def printGame(game):  
    for row in game:  
        print("| ", end = ' ')  
        for char in row:  
            print(char+" | ", end = ' ')  
        print()
```

```
| X | - | - | O | - | X | O | - | - | - |  
| - | - | - | - | O | - | X | X | - | O |  
| - | - | X | - | O | - | - | - | - | - |  
| - | O | - | - | - | - | - | X | - | - |  
| X | X | X | X | - | - | X | - | - | X |  
| - | - | - | - | - | - | - | - | O | - | - |  
| - | O | X | O | - | O | - | - | O | O |  
| - | - | X | - | - | - | X | O | - | - |  
| - | - | - | - | - | - | - | X | - | - | O |  
| - | X | O | - | - | - | - | - | O | - | O |
```

# Tic Tac Toe

## ■ Příklad testu tahu

```
def isAllowed(game, x, y):  
    if x<0 or x >= len(game): return False  
    if y<0 or y >= len(game): return False  
    return game[x][y] == "-"
```

## ■ Příklad testu remízy

```
def boardFull(game):  
    for row in game:  
        for cell in row:  
            if cell == "-": return False  
    return True
```

# Tic Tac Toe

## ■ Tah hráče

```
def humanTurn(game, char):  
    x, y = -1, -1  
    while (not isAllowed(game, x, y)):  
        printGame(game)  
        x = int(input("Enter X position of your move: "))  
        y = int(input("Enter Y position of your move: "))  
    game[x][y] = char
```



# Tic Tac Toe

- Tah počítače

```
import random
```

```
def machineTurn(game, char):  
    x, y = -1, -1  
    while (not isAllowed(game, x, y)):  
        x = random.randint(0, len(game)-1)  
        y = random.randint(0, len(game)-1)  
    game[x][y] = char
```

- Kde je AI????

*Udělejte si v klidu doma*

- Zacyklí se?

*Statistika vs. pseudo-náhodná čísla*

# Tic Tac Toe

- Test konce hry

```
def gameFinished(game):  
    for x in range(len(game)):  
        for y in range(len(game)):  
            char = game[x][y]  
            if char == "-": continue  
            if (y < len(game)-2 and game[x][y+1] == char  
                and game[x][y+2]) == char:  
                return char  
            if (x < len(game)-2 and game[x+1][y] == char  
                and game[x+2][y] == char):  
                return char  
            # add other two directions
```

# Tic Tac Toe

- A celá hra ....

```
machine = True
game = getGame()
while True:
    if boardFull(game):
        print("No winner!")
        break;
    winner = gameFinished(game)
    if not winner:
        if machine:
            machineTurn(game, "O")
        else:
            humanTurn(game, "X")
        machine = not machine
    else:
        if winner == "O": print("Machine wins!")
        else: print("Human wins!")
        break
```

# Práce s polem

- Viz. seznamy a řetězce – druhá přednáška
- Mohou být vnořené
- Tvoření pomocí append nebo zřetězení

```
array = [0 for x in range(n)]
```

- Přístup přes indexy
- Možno modifikovat obsah
- Iterace pomocí cyklu for nebo while

```
for x in array: doSomething(x)
```

```
i = 0  
while i < len(array):  
    doSomething(array[i])  
    i += 1
```

# Základy algoritmizace

- Dnes:
  - Řešení problému a algoritmus
    - Příklad – výpočet společného dělitele
  - Rozklad problému na podproblémy
    - Příklad – hra NIM
  - Pole
    - Příklad – hra Tic Tac Toe

**Příště** vyhledávání a řazení