

# Algoritmy a datové struktury 1

2/2 Z+Zk, NTIN060

úvod + model RAM

Pavel Veselý (IUUK)



`vesely+ads1@iuuk.mff.cuni.cz`

<https://iuuk.mff.cuni.cz/~vesely/vyuka/ls24/ads1.html>

# O čem tento předmět bude?

# O čem tento předmět bude?

Navazujeme na Algoritmizaci

Předběžný plán

1. Časová a paměťová složitost, výpočetní model RAM
2. Grafové algoritmy: DFS, nejkratší cesty, minimální kostry, ...
3. Binární vyhledávací stromy: AVL,  $(a, b)$ -stromy, červeno-černé
4. Hešování
5. Rozděl a panuj
6. Dynamické programování
7. Pravděpodobnostní analýza algoritmů



# O čem tento předmět bude?

Navazujeme na Algoritmizaci

Předběžný plán

1. Časová a paměťová složitost, výpočetní model RAM
2. Grafové algoritmy: DFS, nejkratší cesty, minimální kostry, ...
3. Binární vyhledávací stromy: AVL,  $(a, b)$ -stromy, červeno-černé
4. Hešování
5. Rozděl a panuj
6. Dynamické programování
7. Pravděpodobnostní analýza algoritmů

Cvičení: řešení algoritmických úloh

Zk: teorie z přednášky + algoritmické úlohy

Zdroje: Nahrávky z přednášky (online stream) + Průvodce



## Motivační příklad: jak moc jsou podobné dva řetězce?

- Např.: ALGORITMUS a LOGARITMUS
- Nebo: ALGORITMUS a AL-CHOREZMI





## Editační vzdálenost (Levenštajnova)

```
AACGTGTTGCG-ATCA
|||| | |: ||| ||||
AACG-GTAGCGCATCA
```



## Editační vzdálenost (Levenštajnova)

AACGTGTTGCG-ATCA  
| | | | | : | | | | |  
AACG-GTAGCGCATCA

	p	o	t	e	m	n	í	k	
p	3	4	4	4	4	4	5	6	7
o	4	3	3	3	3	3	4	5	6
u	4	3	3	2	2	2	3	4	5
t	4	3	2	2	1	1	2	3	4
n	5	4	3	2	1	0	1	2	3
í	6	5	4	3	2	1	0	1	2
k	7	6	5	4	3	2	1	0	1
	8	7	6	5	4	3	2	1	0

Obrázek 12.5: Tabulka T pro slova poutník a potemník

# Co je vlastně algoritmus?

# Jak definovat algoritmus?

Potřebujeme definovat počítač!

Výpočetní modely (abstraktní stroje) v literatuře:

- Turingovy stroje
- Lambda kalkulus
- Random-access machine (RAM)
- ...

# Jak definovat algoritmus?

## Potřebujeme definovat počítač!

Výpočetní modely (abstraktní stroje) v literatuře:

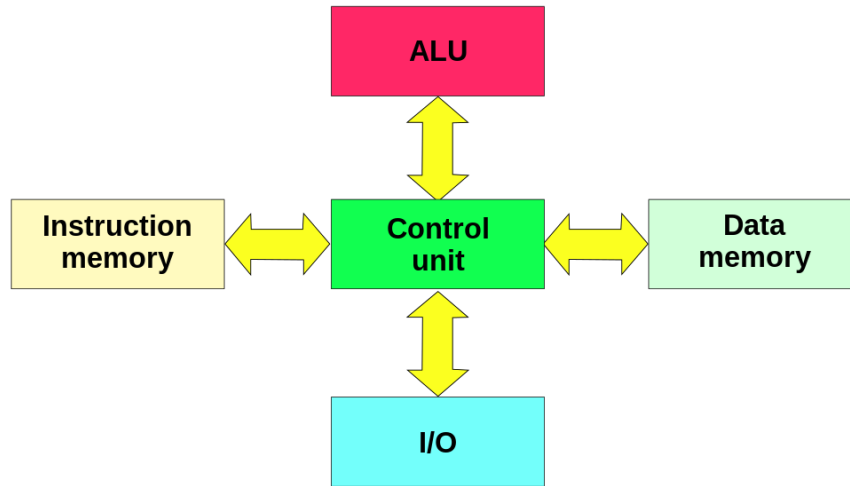
- Turingovy stroje
- Lambda kalkulus
- Random-access machine (RAM)
- ...

Church-Turingova teze (30. léta):

„Ke každému algoritmu existuje ekvivalentní Turingův stroj.“

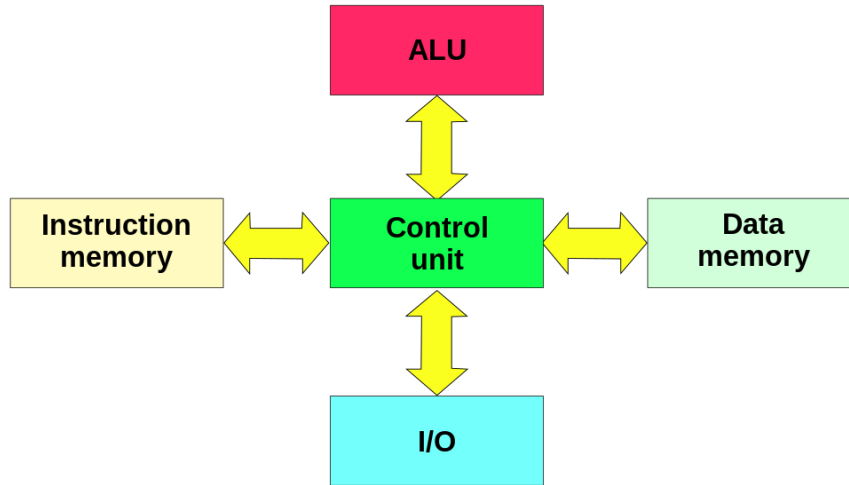
(nebo program pro RAM)

# Harvardská architektura počítače



Zdroj: Nessa Ios, CC BY-SA 3.0, via Wikimedia Commons

# Harvardská architektura počítače



Zdroj: Nessa Ios, CC BY-SA 3.0, via Wikimedia Commons

Versus: von Neumannova architektura (1945): sdílená paměť pro data i program

## Random-access machine (RAM) obecně

Paměť: **neomezené** pole buněk adresovaných  $\mathbb{Z}$

- buňka obsahuje celé číslo

Program: posloupnost instrukcí

# Random-access machine (RAM) obecně

Paměť: **neomezené** pole buněk adresovaných  $\mathbb{Z}$

- buňka obsahuje celé číslo

Program: posloupnost instrukcí

- Výpočet:
- init: *smluvené* buňky obsahují vstup (ostatní nedefinované)
  - RAM provádí instrukce sekvenčně
  - po skončení: *smluvené* buňky obsahují výstup



# Random-access machine (RAM) obecně

**Paměť:** **neomezené** pole buněk adresovaných  $\mathbb{Z}$

- buňka obsahuje celé číslo
- $[0]$ ,  $[100]$ ,  $[-5]$  – **přímá adresace**
- značení:  $A = [-1]$ ,  $B = [-2]$ ,  $\dots$ ,  $Z = [-26]$
- $[[42]]$ ,  $[I]$  – **nepřímá adresace**

**Program:** posloupnost instrukcí

- Výpočet:**
- init: *smluvené* buňky obsahují vstup (ostatní nedefinované)
  - RAM provádí instrukce sekvenčně
  - po skončení: *smluvené* buňky obsahují výstup

# Random-access machine (RAM) obecně

**Paměť:** **neomezené** pole buněk adresovaných  $\mathbb{Z}$

- buňka obsahuje celé číslo
- $[0]$ ,  $[100]$ ,  $[-5]$  – **přímá adresace**
- značení:  $A = [-1]$ ,  $B = [-2]$ ,  $\dots$ ,  $Z = [-26]$
- $[[42]]$ ,  $[I]$  – **nepřímá adresace**

**Program:** posloupnost instrukcí

- Aritmetické:  $\text{kam} \leftarrow \text{co}$
- př.:  $A \leftarrow [[5]] * 3$ 
  - kam je buňka:  $A$ ,  $[42]$  či  $[[0]]$
  - operandy: literály či buňky:  $[42]$  či  $[[0]]$
  - aritmetika:  $+$   $-$   $*$   $/$   $\%$
  - bitové operace:  $\&$   $|$   $\wedge$   $\ll$   $\gg$

- Výpočet:**
- init: *smluvené* buňky obsahují vstup (ostatní nedefinované)
  - RAM provádí instrukce sekvenčně
  - po skončení: *smluvené* buňky obsahují výstup

# Random-access machine (RAM) obecně

**Paměť:** **neomezené** pole buněk adresovaných  $\mathbb{Z}$

- buňka obsahuje celé číslo
- $[0]$ ,  $[100]$ ,  $[-5]$  – **přímá adresace**
- značení:  $A = [-1]$ ,  $B = [-2]$ ,  $\dots$ ,  $Z = [-26]$
- $[[42]]$ ,  $[I]$  – **nepřímá adresace**

**Program:** posloupnost instrukcí

- Aritmetické:  $\text{kam} \leftarrow \text{co}$
- př.:  $A \leftarrow [[5]] * 3$ 
  - kam je buňka:  $A$ ,  $[42]$  či  $[[0]]$
  - operandy: literály či buňky:  $[42]$  či  $[[0]]$
  - aritmetika:  $+$   $-$   $*$   $/$   $\%$
  - bitové operace:  $\&$   $|$   $^$   $\ll$   $\gg$
- logické:
  - halt
  - if  $A = B$  then  $\langle \text{instrukce} \rangle$
  - podmínky:  $=$   $>$   $<$   $\leq$   $\geq$   $\neq$
  - goto  $\langle \text{návěstí} \rangle$

**Výpočet:** • init: *smluvené* buňky obsahují vstup (ostatní nedefinované)

- RAM provádí instrukce sekvenčně
- po skončení: *smluvené* buňky obsahují výstup

## Příklad: Bublínkové třídění

```
procedure bubbleSort(A : list of sortable items)
  n := length(A)
  repeat
    swapped := false
    for i := 1 to n - 1 inclusive do
      if A[i - 1] > A[i] then
        swap(A[i - 1], A[i])
        swapped := true
      end if
    end for
    n := n - 1
  until not swapped
end procedure
```

## Příklad: Bublínkové třídění

```
procedure bubbleSort(A : list of sortable items)
  n := length(A)
  repeat
    swapped := false
    for i := 1 to n - 1 inclusive do
      if A[i - 1] > A[i] then
        swap(A[i - 1], A[i])
        swapped := true
      end if
    end for
    n := n - 1
  until not swapped
end procedure
```

```
1 |# Bubble sort, cell 0 is length of input and input is in cells 1 .. [0]
2 | C := 1
3 | outer:
4 | if C = 0 then halt
5 | C := 0
6 | A := 1
7 | inner:
8 | B := A + 1
9 | if B > [0] then goto outer
10| if [A] <= [B] then goto next
11| T := [A]
12| [A] := [B]
13| [B] := T
14| C := 1
15| next:
16| A := B
17| goto inner
```

## RAM: doba běhu alg.

### 1. Počet provedených instrukcí

## RAM: doba běhu alg.

### 1. Počet provedených instrukcí

- neomezená velikost čísel → triky s velkými čísly (násobení matic v  $O(n^2)$ )

## RAM: doba běhu alg.

### 1. Počet provedených instrukcí

- neomezená velikost čísel → triky s velkými čísly (násobení matic v  $O(n^2)$ )

### 2. Číslo v buňkách omezeno $w$ bity (např. $w = 64$ )

- lze adresovat jen  $2^w$  buňek
- každý výpočet, který skončí, trvá jen **konstantní** čas



## RAM: doba běhu alg.

### 1. Počet provedených instrukcí

- neomezená velikost čísel → triky s velkými čísly (násobení matic v  $O(n^2)$ )

### 2. Číslo v buňkách omezeno $w$ bity (např. $w = 64$ )

- lze adresovat jen  $2^w$  buňek
- každý výpočet, který skončí, trvá jen konstantní čas

### 3. Číslo v buňkách omezeno $k \cdot \log_2 n$ bity — model wordRAM

- $n$  je velikost vstupu,  $k \geq 1$  je konstanta
- lze adresovat „jen“ polynomiálně velký prostor

## RAM: doba běhu alg.

### 1. Počet provedených instrukcí

- neomezená velikost čísel → triky s velkými čísly (násobení matic v  $O(n^2)$ )

### 2. Čísla v buňkách omezena $w$ bity (např. $w = 64$ )

- lze adresovat jen  $2^w$  buňek
- každý výpočet, který skončí, trvá jen konstantní čas

### 3. Čísla v buňkách omezena $k \cdot \log_2 n$ bity — model wordRAM

- $n$  je velikost vstupu,  $k \geq 1$  je konstanta
- lze adresovat „jen“ polynomiálně velký prostor

### 4. Logaritmická cena instrukce

- Doba instrukce =  $\sum$  počtu bitů výsledku a operandů (včetně adres)
- sečtení dvou čísel s  $\log_2 n$  bity trvá  $O(\log_2 n)$  😞

# RAM: doba běhu alg.

## 1. Počet provedených instrukcí

- neomezená velikost čísel → triky s velkými čísly (násobení matic v  $O(n^2)$ )

## 2. Čísla v buňkách omezena $w$ bity (např. $w = 64$ )

- lze adresovat jen  $2^w$  buňek
- každý výpočet, který skončí, trvá jen konstantní čas

## 3. Čísla v buňkách omezena $k \cdot \log_2 n$ bity — model wordRAM

- $n$  je velikost vstupu,  $k \geq 1$  je konstanta
- lze adresovat „jen“ polynomiálně velký prostor

## 4. Logaritmická cena instrukce

- Doba instrukce =  $\sum$  počtu bitů výsledku a operandů (včetně adres)
- sečtení dvou čísel s  $\log_2 n$  bity trvá  $O(\log_2 n)$  😞

## 5. Relativní logaritmická cena instrukce:

- práce s  $\log_2(n)$  bitovými čísly v čase  $O(1)$  😊

$$\left[ \frac{\text{Logaritmická cena}}{\log_2(n)} \right]$$

## Asymptotická složitost

- $f \in O(g)$ : „ $f$  je asymptoticky menší nebo rovno  $g$ “
- $f \in \Omega(g)$ : „ $f$  je asymptoticky větší nebo rovno  $g$ “
- $f \in \Theta(g)$ : „ $f$  je asymptoticky stejné jako  $g$ “

## Asymptotická složitost

- $f \in O(g)$ : „ $f$  je asymptoticky menší nebo rovno  $g$ “
- $f \in \Omega(g)$ : „ $f$  je asymptoticky větší nebo rovno  $g$ “
- $f \in \Theta(g)$ : „ $f$  je asymptoticky stejné jako  $g$ “
- $f \in o(g)$ : „ $f$  je asymptoticky *ostře* menší než  $g$ “
- $f \in \omega(g)$ : „ $f$  je asymptoticky *ostře* větší než  $g$ “

## Asymptotická složitost

- $f \in O(g)$ : „ $f$  je asymptoticky menší nebo rovno  $g$ “
- $f \in \Omega(g)$ : „ $f$  je asymptoticky větší nebo rovno  $g$ “
- $f \in \Theta(g)$ : „ $f$  je asymptoticky stejné jako  $g$ “
- $f \in o(g)$ : „ $f$  je asymptoticky *ostře* menší než  $g$ “
- $f \in \omega(g)$ : „ $f$  je asymptoticky *ostře* větší než  $g$ “

### Výhody:

- snadné určení
- konstanty strojově závislé
- vhodná pro „dost velké“ vstupy
- vhodná pro teorii, kdy nevíme, jaké budou vstupy

## Asymptotická složitost

- $f \in O(g)$ : „ $f$  je asymptoticky menší nebo rovno  $g$ “
- $f \in \Omega(g)$ : „ $f$  je asymptoticky větší nebo rovno  $g$ “
- $f \in \Theta(g)$ : „ $f$  je asymptoticky stejné jako  $g$ “
- $f \in o(g)$ : „ $f$  je asymptoticky *ostře* menší než  $g$ “
- $f \in \omega(g)$ : „ $f$  je asymptoticky *ostře* větší než  $g$ “

### Výhody:

- snadné určení
- konstanty strojově závislé
- vhodná pro „dost velké“ vstupy
- vhodná pro teorii, kdy nevíme, jaké budou vstupy

### Nevýhody:

- konstanty **jsou** v praxi důležité
- zaměření na **nejhorší případy**

Pro „malé“ vstupy se může hodit nasadit  
jednodušší, ale asymptoticky pomalejší algoritmus