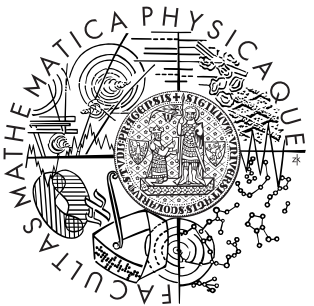
Algoritmy a datové struktury 1

2/2 Z+Zk, NTIN060

úvod + model RAM

Pavel Veselý (IUUK)



vesely+ads1@iuuk.mff.cuni.cz

https://iuuk.mff.cuni.cz/∼vesely/vyuka/LS2122/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

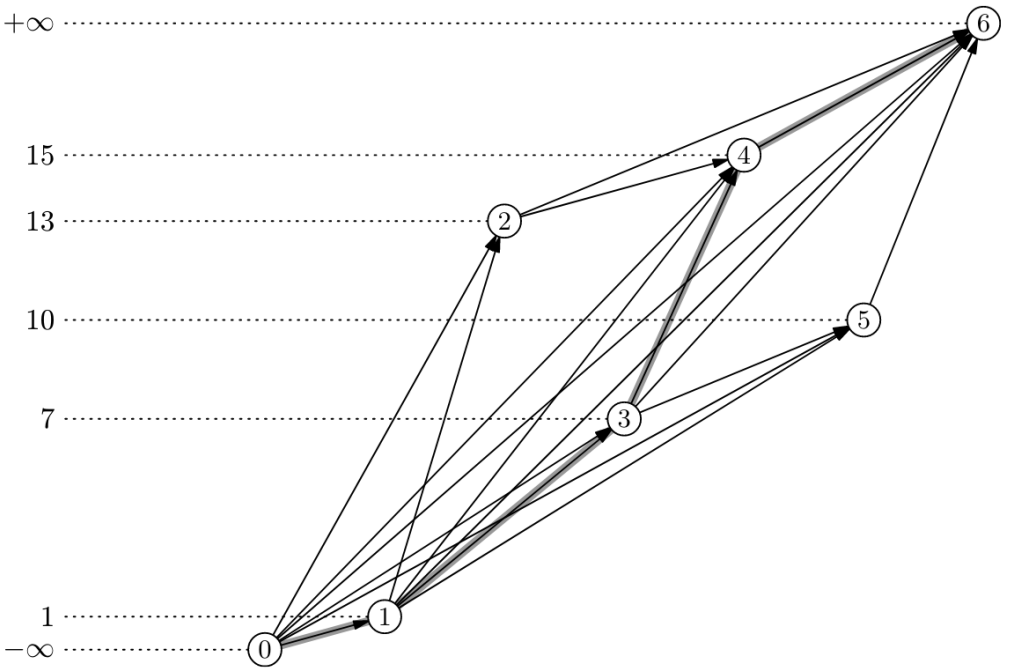
Cvičení: řešení algoritmických úloh

Zk: teorie z přednášky + algoritmické úlohy

Stream z přednášky + nahrávky

Úvodní příklad: nejdelší rostoucí podposloupnost

Úvodní příklad: nejdelší rostoucí podposloupnost



Obrázek 12.4: Graf reprezentující posloupnost $-\infty, 1, 13, 7, 15, 10, +\infty$ a jedna z nejdelších cest

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:

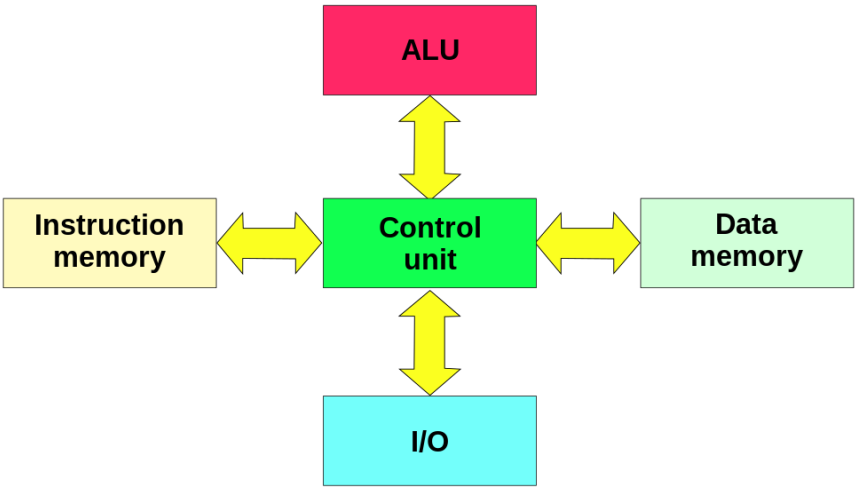
- Turingovy stroje
- Lambda kalkulus
- Random-access machine (RAM)
- ...

Church-Turingova teze (30. léta):

„Ke každému algoritmu existuje ekvivalentní Turingův stroj.“

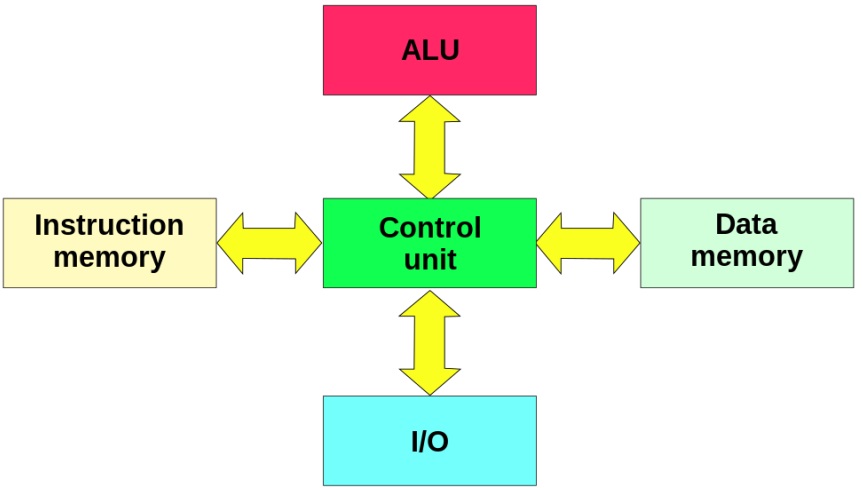
(nebo program pro RAM)

Harvardská architektura počítače



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

Harvardská architektura počítače



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

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
  - `[-5]` – přímá adresace
  - `[[42]]` – nepřímá adresace
  - značení: `A = [-1], B = [-2], ..., Z = [-26]`

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
  - `[-5]` – přímá adresace
  - `[[42]]` – nepřímá adresace
  - značení: `A = [-1], B = [-2], ..., Z = [-26]`

Program: posloupnost instrukcí

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

Výpočet:

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

RAM: doba běhu alg.

1. Počet provedených instrukcí

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

RAM: doba běhu alg.

1. Počet provedených instrukcí

- neomezená velikost čísel $\rightarrow$ 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 buněk
- každý výpočet, který skončí, trvá jen konstantní čas

RAM: doba běhu alg.

1. Počet provedených instrukcí

- neomezená velikost čísel $\rightarrow$ 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 buněk
- 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 lze adresovat „jen“ polynomiálně velký prostor

RAM: doba běhu alg.

1. Počet provedených instrukcí

- neomezená velikost čísel $\rightarrow$ 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 buněk
- 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 lze adresovat „jen“ polynomiálně velký prostor

4. Logaritmická cena instrukce

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

RAM: doba běhu alg.

1. Počet provedených instrukcí

- neomezená velikost čísel $\rightarrow$ 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 buněk
- 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 lze adresovat „jen“ polynomiálně velký prostor

4. Logaritmická cena instrukce

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

5. Relativní logaritmická cena instrukce: $\left\lceil \frac{\text{Logaritmická cena}}{\log_2(n)} \right\rceil$

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

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