IB002 Algoritmy a datové struktury I

Ivana Černá

Fakulta informatiky, Masarykova univerzita

Jaro 2016

Informace o předmětu

- vyučující předmětu
  - Ivana Černá (přednášky)
  - Vojtěch Řehák (cvičení)
  - Nikola Beneš, Jan Obdržálek, Jaromír Plhák, Kristína Zákopčanová, František Blahoudek, Henrich Lauko, Peter Bezděk, Filip Opálený, Tomáš Zábojník, Jan Ježek, Kristýna Pavlíčková, Tomáš Novotný, Jan Koniarik, Peter Benčík

- **interaktivní osnova předmětu - kompletní informace**
  `is.muni.cz/auth/el/1433/jaro2016/IB002/index.qwarp`
  - organizace výuky (přednášky, cvičení, domácí úkoly, konzultace)
  - hodnocení předmětu (odpovědníky, praktický test, zkouška)
  - studijní materiály (slajdy, skripta, zadání úkolů, rozcestníky)

- diskusní fórum předmětu v IS MU

Literatura

- T. Cormen, Ch. Leiserson, R. Rivest, C. Stein:*Introduction to Algorithms*. Third Edition. MIT Press, 2009
- J. Kleinberg, and E. Tardos:*Algorithm Design*. Addison-Wesley, 2006
- S. Dasgupta, Ch. Papadimitriou, U. Vazirani:*Algorithms*. McGraw Hill, 2007.
- T. Cormen:*Algorithms Unlocked*. MIT Press, 2013
- J. Bentley: *Programming Pearls (2nd Edition)*. Addison-Wesley, 1999.

- Wikipedie (*eng)*

- *online materiály a video přednášky (*doporučuji MIT 6.006 Intro to Algorithms)

obrázky použité v prezentacích jsou částečné převzaty z uvedených monografií

Obsah

algoritmus způsob řešení problému

datová struktura způsob uložení informací

techniky návrhu a analýzy algoritmů
: důkaz korektnosti algoritmu, analýza složitosti algoritmu, asymptotická notace, technika rozděl & panuj a rekurze

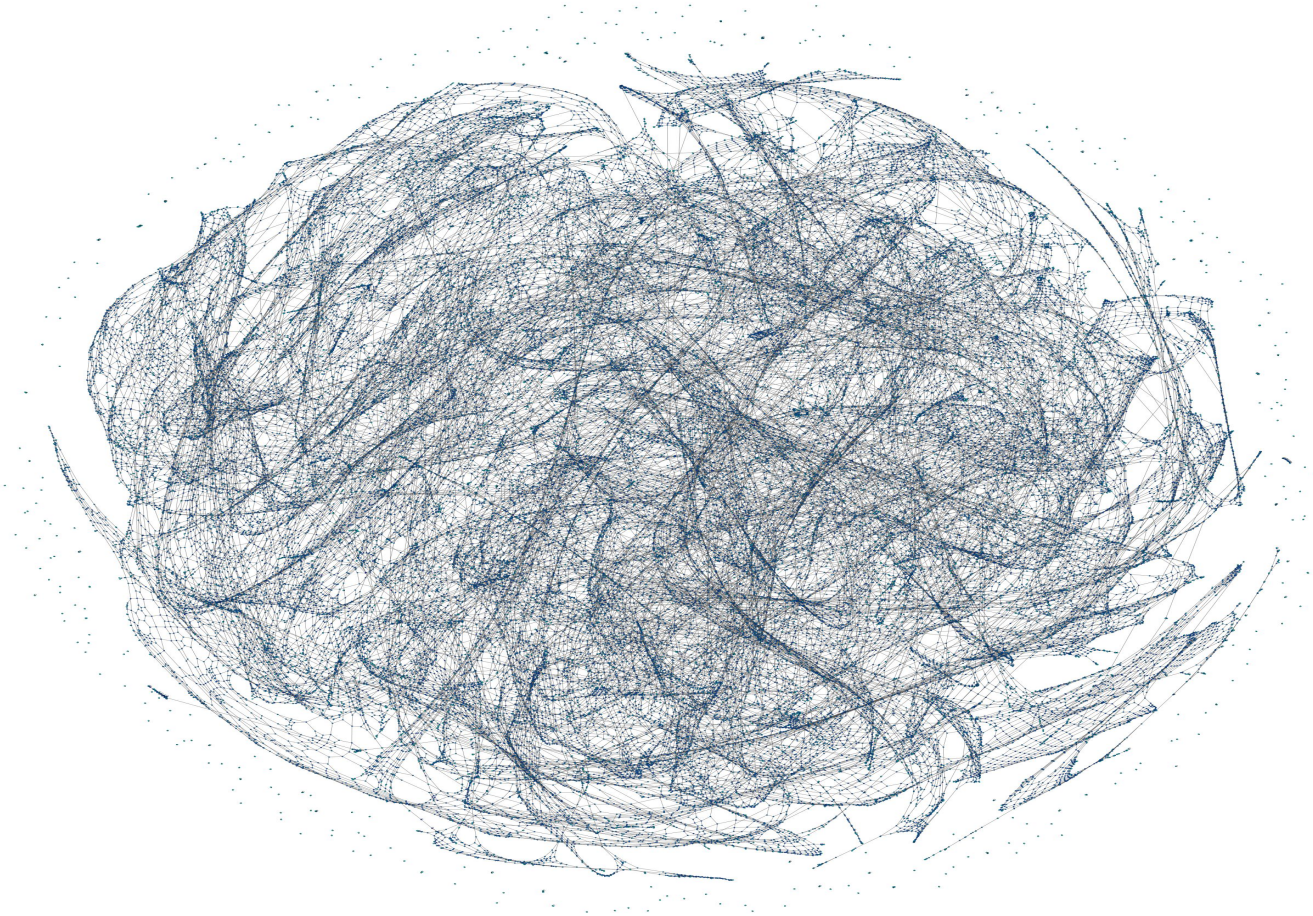
datové struktury
: halda, prioritní fronta, vyhledávací stromy, červeno-černé stromy, B-stromy, hašovací tabulky

algoritmy řazení rozdělováním, slučováním, haldou, v lineárním čase

grafové algoritmy prohledávání grafu, souvislost, cesty

Motivace

řešit jinak neřešitelné problémy...



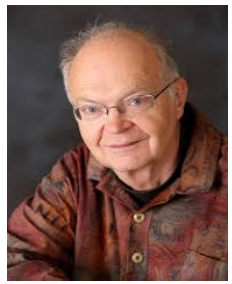
Chicago road networks, http://csun.uic.edu/dataviz.html

Motivace

naučit se něco nového...

An algorithm must be seen to be believed, and the best way to learn about what an algorithm is all about is to try it.

— Donald Knuth, The Art of Computer Programming



Algorithms: a common language for nature, human, and computer.

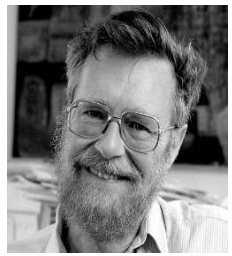
— Avi Wigderson

Motivace

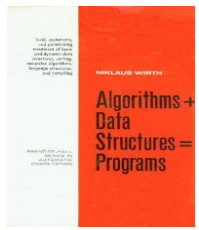
stát se dobrým programátorem...

I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships. — *Linus Torvalds (creator of Linux)*

Progress is possible only if we train ourselves to think about programs without thinking of them as pieces of executable code.
— *Edsger W. Dijkstra*



Algorithms + Data Structures = Programs.
— *Niklaus Wirth*



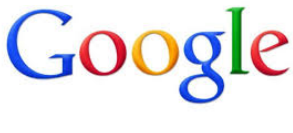
Motivace

široké uplatnění...

- návrh počítačů logické obvody, systém souborů, překladače
- Internet vyhledávání, distribuované sdílení, cloud computing, packet routing
- počítačová grafika virtuální realita, video, hry
- multimédia mp3, jpg, rozpoznávání obrazu
- bezpečnost šifrování, hlasování, e-obchod
- sociální sítě doporučení a predikce, reklama
- fyzika simulace
- biologie projekt lidského genomu, simulace

Motivace

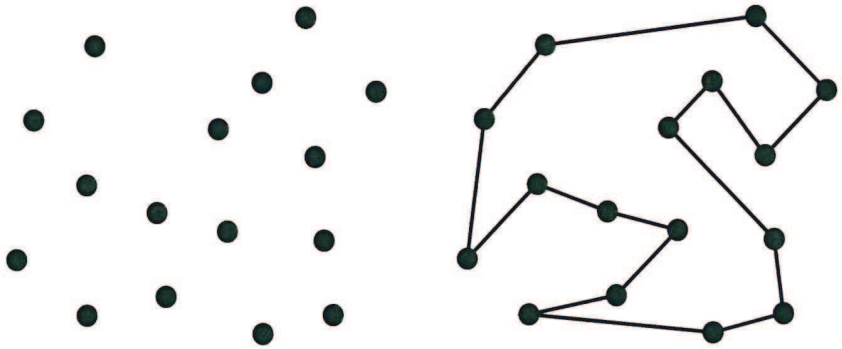
pro zábavu a zisk...



Návrh a analýza algoritmů

Motivace

najdi nejkratší cestu pro rozvoz čerstvé pizzy



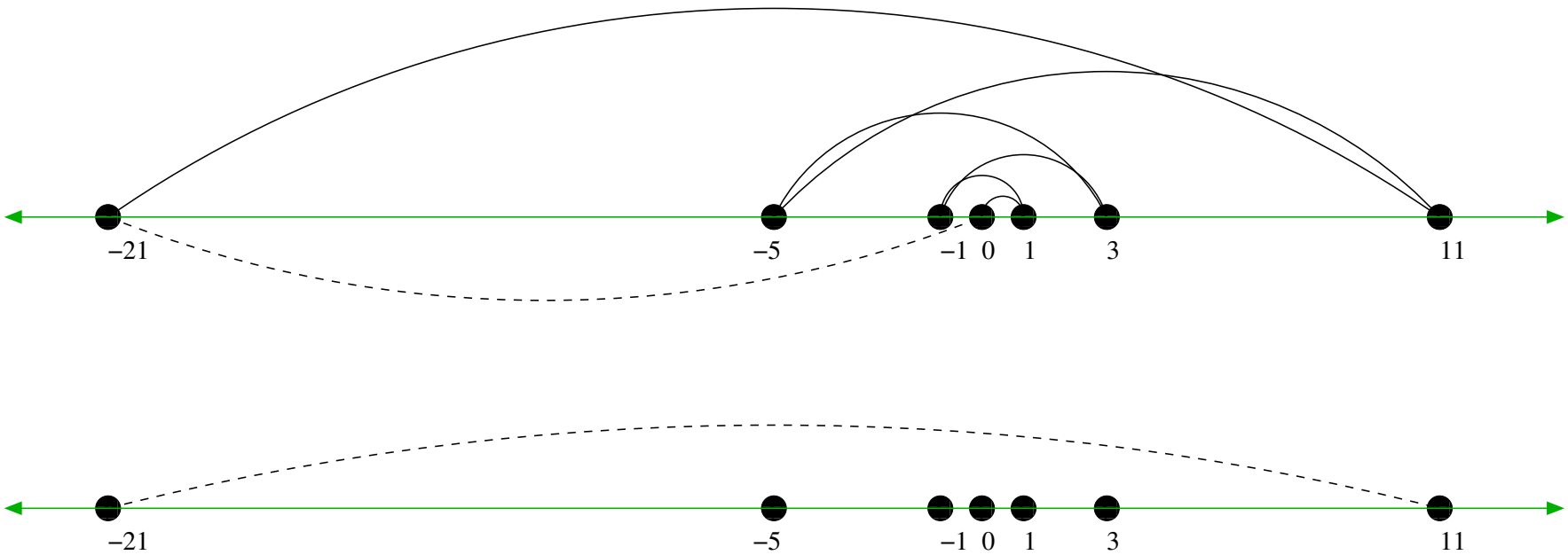
ALGORITMUS???

Řešení 1

vyber počáteční vrchol $v_0, i \leftarrow 0$
while existuje nenavštívený vrchol **do**
    $i \leftarrow i + 1$
    nechť p_i je nenavštívený vrchol nejblíž k p_{i-1}
    navštiv p_i **od**
return vrať cestu z p_0 do p_i

Řešení 1

vyber počáteční vrchol $v_0, i \leftarrow 0$
while existuje nenavštívený vrchol **do**
    $i \leftarrow i + 1$
    nechť p_i je nenavštívený vrchol nejblíž k p_{i-1}
    navštiv p_i **od**
return vrať cestu z p_0 do p_i

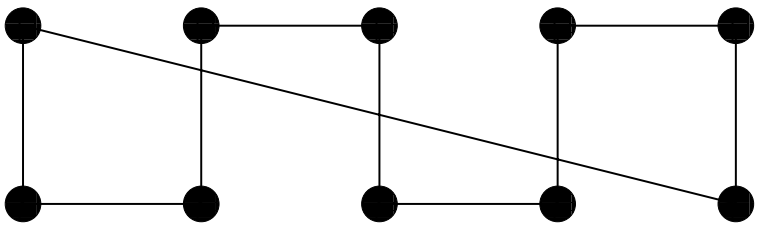


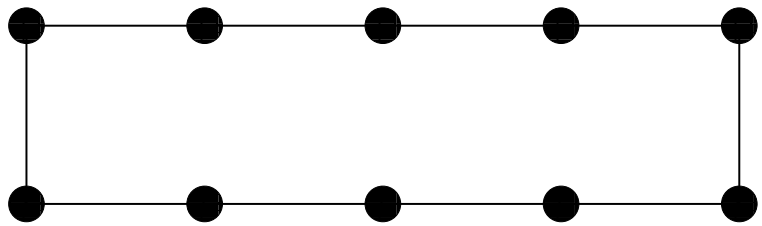
Řešení 2

nechť n je počet vrcholů
for $i = 1$ **to** $n - 1$ **do**
    $d \leftarrow \infty$
    for každou dvojici (x, y) koncových bodů částečných cest **do**
        if $dist(x, y) \leq d$ **then** $x_i \leftarrow x, y_i \leftarrow y, d \leftarrow dist(x, y)$ **fi od**
    spoj vrcholy x_i, y_i hranou **od**
spoj hranou koncové vrcholy cesty

Řešení 2

nechť n je počet vrcholů
for $i = 1$ **to** $n - 1$ **do**
    $d \leftarrow \infty$
    for každou dvojici (x, y) koncových bodů částečných cest **do**
        if $dist(x, y) \leq d$ **then** $x_i \leftarrow x, y_i \leftarrow y, d \leftarrow dist(x, y)$ **fi od**
    spoj vrcholy x_i, y_i hranou **od**
spoj hranou koncové vrcholy cesty





Korektní algoritmus

$d \leftarrow \infty$
for každou z $n!$ permutací Π_i vrcholů **do**
  if $cost(\Pi_i) \leq d$ $cost(\Pi_i) je\ cena\ cesty\ určené\ permutací\ \Pi_i$
    then $d \leftarrow cost(\Pi_i) \wedge P_{min} \leftarrow \Pi_i$ **fi od**
return P_{min}

korektnost algoritmus prověří všech $n!$ možných způsobů projití grafu

Korektní algoritmus

$d \leftarrow \infty$
for každou z $n!$ permutací Π_i vrcholů **do**
  if $cost(\Pi_i) \leq d$ $cost(\Pi_i) je\ cena\ cesty\ určené\ permutací\ \Pi_i$
    then $d \leftarrow cost(\Pi_i) \wedge P_{min} \leftarrow \Pi_i$ **fi od**
return P_{min}

korektnost algoritmus prověří všech $n!$ možných způsobů projití grafu
složitost algoritmus je nepoužitelný již pro velmi malé grafy
??? existuje efektivnější algoritmus ???

Vyhledávání prvku v posloupnosti

Vstup posloupnost prvků $A[1 \ldots n]$ a prvek x

Výstup index i takový, že $A[i] = x$, resp. hodnota No jestliže prvek x se v posloupnosti nevyskytuje

Procedure Linear Search

```
1 answer ← No
2 for i = 1 to n do
3     if A[i] = x then answer ← i fi
4 od
5 return answer
```

Vyhledávání prvku v posloupnosti – optimalizace I

Procedure Better Linear Search

```
1 for i = 1 to n do
2     if A[i] = x then return i fi
3 od
4 return No
```

- první výskyt x ukončí prohledávání
- každý průchod cyklem znamená 2 testy: v řádku 1 testujeme rovnost $i \leq n$, v řádku 2 testujeme rovnost $A[i] = x$
- stačí 1 test?

Vyhledávání prvku v posloupnosti – optimalizace II

Procedure Sentinel Linear Search

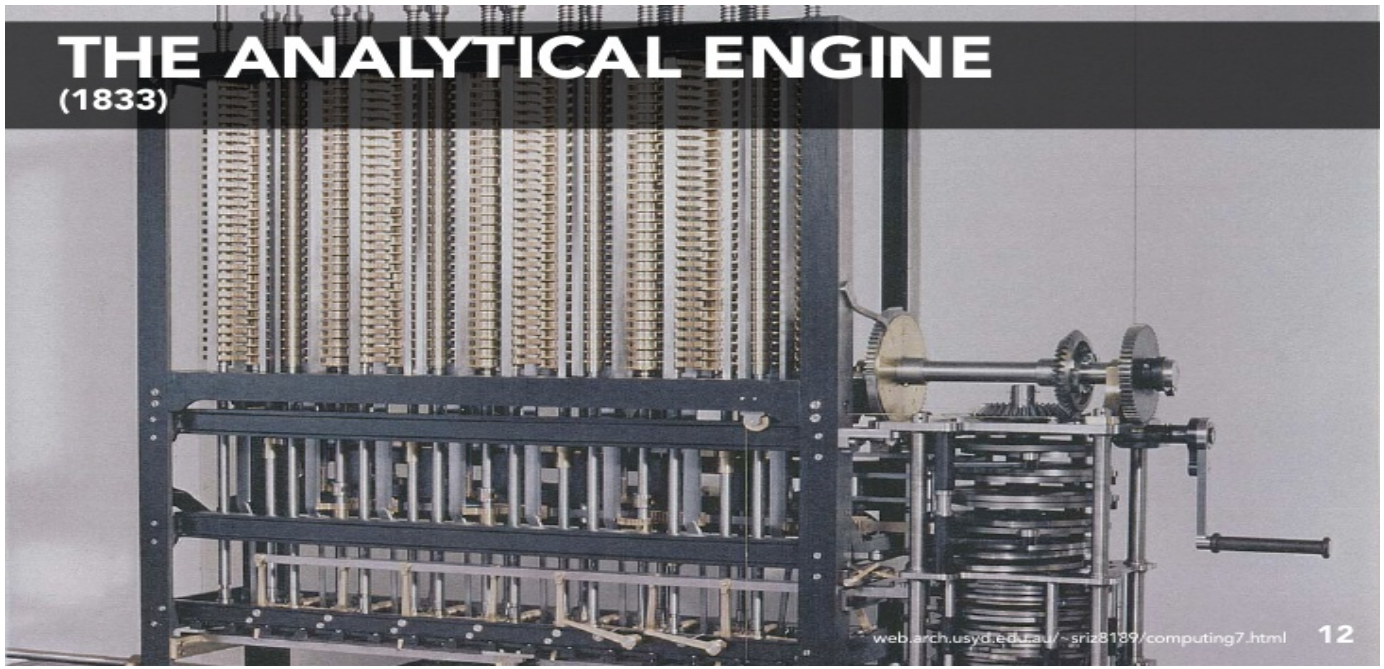
```
1 last ← A[n]
2 A[n] ← x
3 i ← 1
4 while A[i] ≠ x do i ← i + 1 od
5 A[n] ← last
6 if i < n ∨ A[n] = x
7    then return i
8     else return No fi
```

- první výskyt x ukončí prohledávání
- sentinel (zarážka) pro případ, že pole neobsahuje prvek x
- každý průchod cyklem znamená 1 test
- 2 testy na závěr (řádek 6)

Složitost

Charles Babage, 1864

As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any resul is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time?



kolikrát musíme zatočit klikou?

Časová složitost

časová složitost **výpočtu** = součet (cena operace $\times$ počet opakování)

časová složitost **algoritmu** je funkce délky vstupu

- složitost v nejhorším případě
  maximální délka výpočtu na vstupu délky n
- složitost v nejlepším případě
  minimální délka výpočtu na vstupu délky n
- průměrnásložitost
  průměr složitostí výpočtů na všech vstupech délky n

složitost = časová složitost v nejhorším případě

Časová složitost Linear Search

```
1 answer ← No
2 for i = 1 to n do
3     if A[i] = x
4         then answer ← i fi od
5 return answer
```

- označme t_i cenu operace na řádku i
- operace z řádků 1 a 5 se vykonají jednou
- řádek 2 se vykoná $n + 1$ krát
- řádek 3 se vykoná n krát
- přiřazení v řádku 4 se vykoná úměrně počtu výskytů x v poli

časová složitost v nejlepším případě

$$t_1 + t_2 \cdot (n + 1) + t_3 \cdot n + t_4 \cdot 0 + t_5$$

časová složitost v nejhorším případě

$$t_1 + t_2 \cdot (n + 1) + t_3 \cdot n + t_4 \cdot n + t_5$$

složitost je tvaru $c \cdot n + d$, kde c a d jsou konstanty nezávislé na n
složitost je lineární vzhledem k délce vstupu n

Časová složitost optimalizovaných algoritmů

Better Linear Search

1 **for** $i = 1$ **to** n **do if** $A[i] = x$ **then return** i **fi od**
2 **return** No

- časová složitost v nejhorším případě je lineární
- časová složitost v nejlepším případě je konstantní

Sentinel Linear Search

1 $last \leftarrow A[n], A[n] \leftarrow x, i \leftarrow 1$
2 **while** $A[i] \neq x$ **do** $i \leftarrow i + 1$ **od**
3 $A[n] \leftarrow last$
4 **if** $i < n \vee A[n] = x$ **then return** i **else return** No **fi**

- časová složitost v nejhorším případě je lineární
- časová složitost v nejlepším případě je konstantní
- rozdíl je v konstantních faktorech

Korektnost algoritmu

vstupní podmínka ze všech možných vstupů pro daný algoritmus vymezuje ty, pro které je algoritmus definován

výstupní podmínka pro každý vstup daného algoritmu splňující vstupní podmínku určuje, jak má vypadat výsledek odpovídající danému vstupu

algoritmus je (totálně) korektní jestliže pro každý vstup splňující vstupní podmínku výpočet skončí a výsledek splňuje výstupní podmínku

úplnost (konvergence) pro každý vstup splňující vstupní podmínku výpočet skončí

částečná (parciální) korektnost pro každý vstup, který splňuje vstupní podmínku a výpočet na něm skončí, výstup splňuje výstupní podmínku

Korektnost iterativního algoritmu

analyzujeme efekt jednotlivých operací

analýza efektu cyklu

- u vnořených cyklů začínáme od cyklu nejhlubší úrovně
- pro každý cyklus určíme jeho invariant
- invariantem cyklu je takové tvrzení, které platí před vykonáním a po vykonání každé iterace cyklu
- dokážeme, že invariant cyklu je pravdivý
- využitím invariantu
  - dokážeme konečnost výpočtu cyklu
  - dokážeme efekt cyklu

Invariant cyklu

Inicializace invariant je platný před začátkem vykonávání cyklu

Iterace jestliže invariant platí před iterací cyklu, zůstává v platnosti i po vykonání iterace

Ukončení cyklus skončí a po jeho ukončení platný invariant garantuje požadovaný efekt cyklu

Korektnost Better Linear Search

```
1 for i = 1 to n do if A[i] = x then return i fi od
2 return No
```

Invariant cyklu

Na začátku každé iterace cyklu platí, že jestliže prvek x se nalézá v A, tak se nalézá v části mezi pozicemi i a n.

Inicializace Na začátku je $i = 1$ a proto tvrzení platí.

Iterace Předpokládejme platnost tvrzení na začátku iterace.

Jestliže iterace nevrátí výslední hodnotu, tak $A[i] \neq x$. Proto x musí být na některé z pozic $i + 1$ až n a invariant zůstává v platnosti i po ukončení iterace (tj. před následující iterací).

Jestliže iterace vrátí hodnotu i, platnost tvrzení po ukončení iterace je zřejmá.

Ukončení Cyklus skončí buď proto, že je nalezena hodnota x anebo proto, že $i > n$. V obou případech z platnosti tvrzení po ukončení iterace cyklu plyne korektnost vypočítaného výsledku.

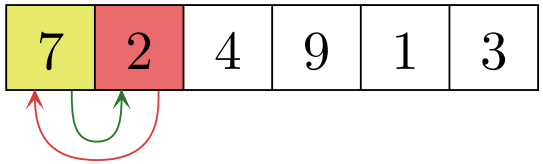
Problém řazení

Vstup posloupnost n čísel $(a_1, a_2, \ldots, a_n)$

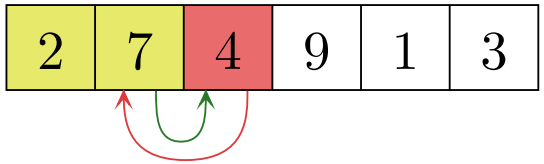
Výstup permutace (přeuspořádání) $(a'_1, a'_2, \ldots, a'_n)$ vstupní posloupnosti taková, že $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Princip řazení vkládáním

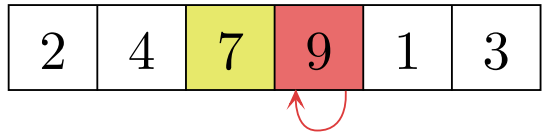
(a)



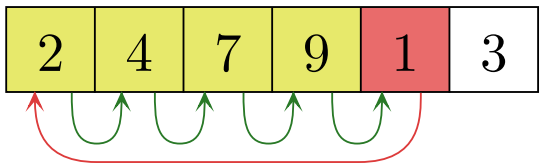
(b)



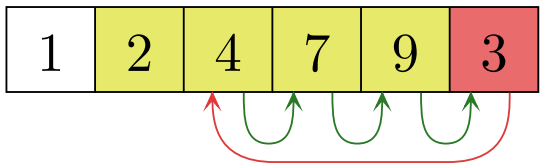
(c)



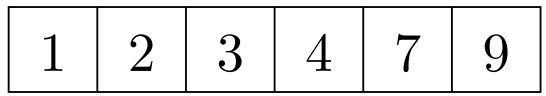
(d)



(e)



(f)



Algoritmus

Insert Sort(A)

```
1 for j = 2 to A.length do
2     key ← A[j]
3     //  Vlož A[j] do seřazené postupnosti A[1 . . . j − 1]
4     i ← j − 1
5     while i > 0 ∧ A[i] > key do
6           A[i + 1] ← A[i]
7           i ← i − 1 od
8     A[i + 1] ← key od
```

Korektnost řazení vkládáním

Invariant cyklu

Na začátku každé iterace **for** cyklu obsahuje pole $A[1 \dots j-1]$ stejné prvky jako na začátku výpočtu, ale seřazené od nejmenšího po největší.

Inicializace Před první iterací je $j = 2$ a tvrzení platí.

Iterace Předpokládejme že tvrzení platí před iterací j, tj. prvky v $A[1 \dots j-1]$ jsou seřazeny. Jestliže $A[j] < A[j-1]$, tak v těle cyklu se prvky $A[j-1], A[j-2], A[j-3], \dots$ posouvají o jednu pozici doprava tak dlouho, až se najde vhodná pozice pro prvek $A[j]$ (ř. 5 - 7). Pole $A[1 \dots j]$ proto na konci iterace cyklu obsahuje stejné prvky jako na začátku, ale seřazené. Po navýšení hodnoty j zůstává tvrzení v platnosti.

Ukončení Cyklus skončí když $j > A.length = n$. Protože v každé iteraci se hodnota j navyšuje o 1, musí platit $j = n + 1$. Z platnosti invariantu cyklu plyne, že $A[1 \dots n]$ obsahuje stejné prvky jako na začátku výpočtu, ale seřazené.

Složitost řazení vkládáním

	Insertion Sort(A)	**cena**	**počet**
1	**for** $j = 2$ **to** $A.length$ **do**	c_1	n
2	$key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0 \wedge A[i] > key$ **do**	c_4	$\sum_{j=2}^{n} t_j$
5	$A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6	$i \leftarrow i-1$ **od**	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$A[i+1] \leftarrow key$ **od**	c_7	$n-1$

t_j označuje počet opakování **while** cyklu pro danou hodnotu j

počet testů v hlavičce cyklu je o 1 vyšší než počet iterací cyklu

Složitost řazení vkládáním -nejlepší případ

Insertion Sort(A)	cena	počet	
1 **for** $j = 2$ **to** $A.length$ **do**	c_1	n	
2 $key \leftarrow A[j]$	c_2	$n-1$	
3 $i \leftarrow j-1$	c_3	$n-1$	
4 **while** $i > 0 \wedge A[i] > key$ **do**	c_4	$\sum_{j=2}^{n} t_j$	$t_j = 1$
5 $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_j - 1)$	
6 $i \leftarrow i-1$ **od**	c_6	$\sum_{j=2}^{n}(t_j - 1)$	
7 $A[i+1] \leftarrow key$ **od**	c_7	$n-1$	

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n}(t_j - 1) + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7(n-1)$$

$$= c_1 n + c_2(n-1) + c_4(n-1) + c_4(n-1) + c_7(n-1)$$

$$= an + b$$

lineární složitost

Složitost řazení vkládáním -nejhorší případ

Insertion Sort(A)	cena	počet	
1 **for** $j = 2$ **to** $A.length$ **do**	c_1	n	
2 $key \leftarrow A[j]$	c_2	$n-1$	
3 $i \leftarrow j-1$	c_3	$n-1$	
4 **while** $i > 0 \wedge A[i] > key$ **do**	c_4	$\sum_{j=2}^{n} t_j$	$t_j = j$
5 $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_j - 1)$	
6 $i \leftarrow i-1$ **od**	c_6	$\sum_{j=2}^{n}(t_j - 1)$	
7 $A[i+1] \leftarrow key$ **od**	c_7	$n-1$	

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right)$$
$$+ c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$
$$= an^2 + bn + c$$

kvadratická složitost

Asymptotická notace

- asymptotickou notaci využíváme při popisu složitosti algoritmů
- umožňuje abstrahovat od detailů / zdůraznit podstatné

příklad

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n+1)}{2}\right) + c_6\left(\frac{n(n+1)}{2}\right) + c_7(n-1)$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$

$$= an^2 + bn + c$$

$$= \Theta(n^2)$$

Typy notací

- $f(n) = \mathcal{O}(g(n))$ znamená, že $C \times g(n)$ je **horní hranicí** pro $f(n)$
- $f(n) = \Omega(g(n))$ znamená, že $C \times g(n)$ je **dolní hranicí** pro $f(n)$
- $f(n) = \Theta(g(n))$ znamená, že $C_1 \times g(n)$ je horní hranicí pro $f(n)$ a $C_2 \times g(n)$ je dolní hranicí pro $f(n)$

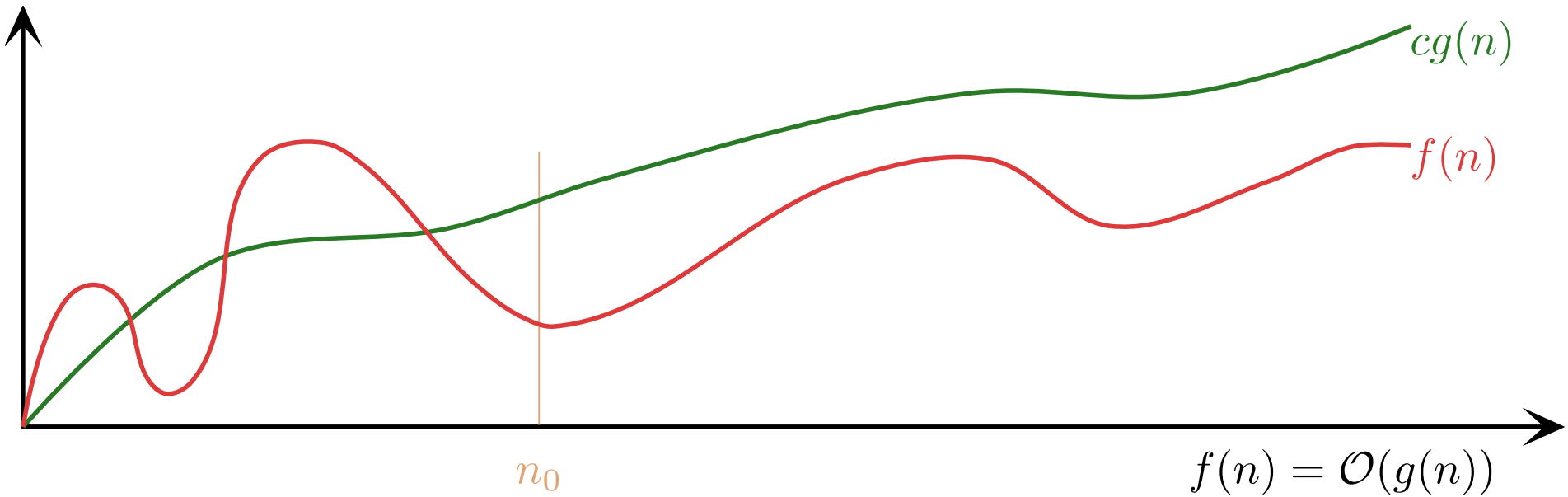
f, g jsou funkce, $f, g : \mathbb{N} \to \mathbb{N}$

C, C_1, C_2 jsou konstanty nezávislé na n

$\mathcal{O}$ notace

Definice

$f(n) = \mathcal{O}(g(n))$ právě když existují kladné konstanty n_0 a c takové, že pro všechna $n \geq n_0$ platí $f(n) \leq cg(n)$



- zápis $f(n) \in \mathcal{O}(g(n))$ vs zápis $f(n) = \mathcal{O}(g(n))$ (*historické důvody*)
- funkce $g(n)$ roste *asymptoticky rychleji* než funkce $f(n)$
- alternativní definice $f(n) = \mathcal{O}(g(n))$ právě když $\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$

$\mathcal{O}$ notace - příklady

- $8n^2 - 88n + 888 = \mathcal{O}(n^2)$

  protože $8n^2 - 88n + 888 < 8n^2$ pro všechna $n \geq 11$

- $8n^2 - 88n + 888 = \mathcal{O}(n^3)$

  protože $8n^2 - 88n + 8 < 1n^3$ pro všechna $n \geq 10$

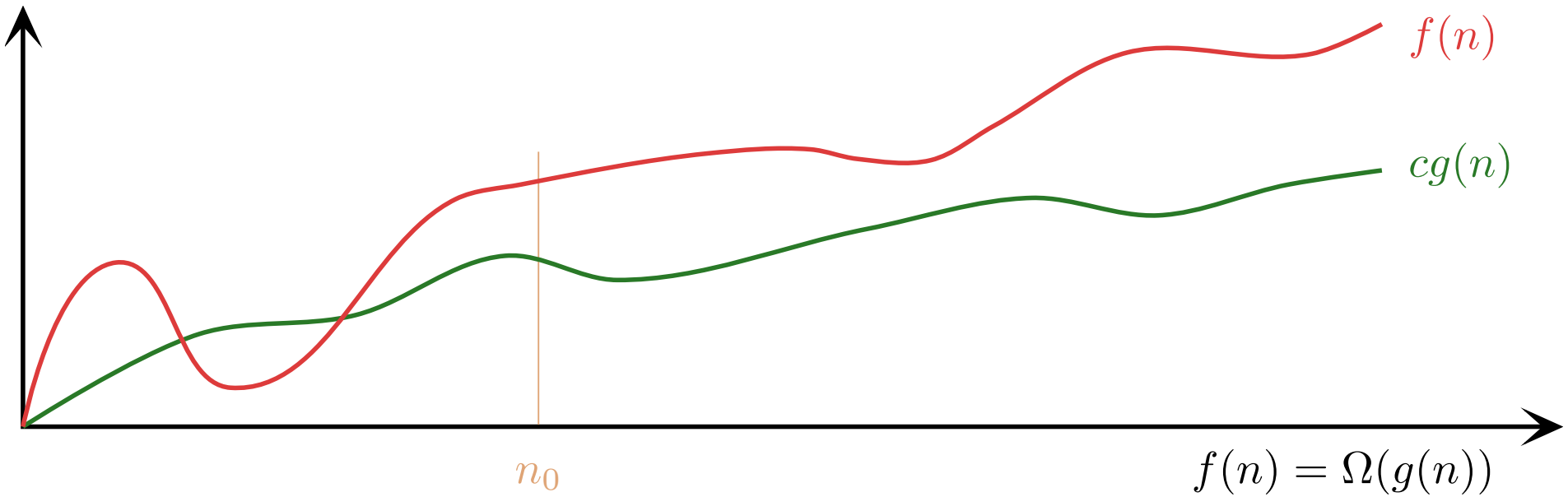
- $8n^2 - 88n + 888 \neq \mathcal{O}(n)$

  protože $cn < 8n^2 - 88n + 888$ pro $n > c$

Ω notace

Definice

$f(n) = \Omega(g(n))$ právě když existují kladné konstanty n_0 a c takové, že pro všechna $n \geq n_0$ platí $f(n) \geq cg(n)$



funkce $g(n)$ roste *asymptoticky pomaleji* než funkce $f(n)$

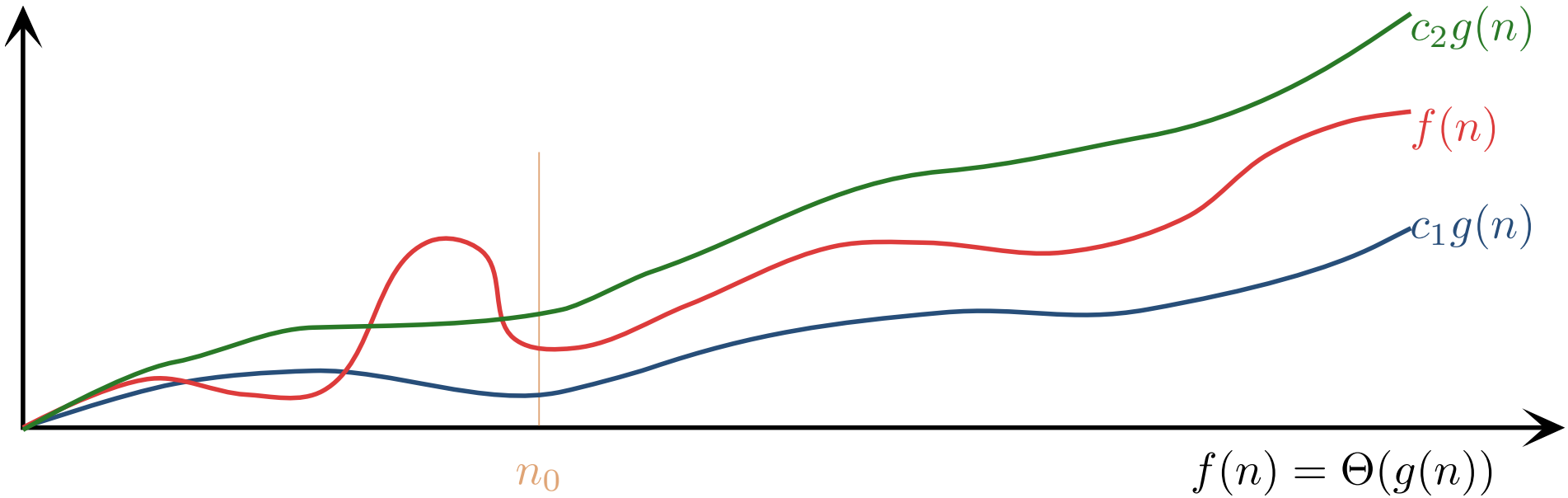
Ω notace - příklady

- $8n^2 - 88n + 8 = \Omega(n^2)$ protože $8n^2 - 88n + 8 > n^2$ pro $n > 13$
- $8n^2 - 88n + 8 \neq \Omega(n^3)$ protože $8n^2 - 88n + 8 < cn^3$ pro $n > \frac{8}{c}$
- $8n^2 - 88n + 8 = \Omega(n)$ protože $8n^2 - 88n + 8 > n$ pro $n > 11$

Θ notace

Definice

$f(n) = \Theta(g(n))$ právě když existují kladné konstanty n_0, c_1 a c_1 takové, že pro všechna $n \geq n_0$ platí $c_1 g(n) \leq f(n) \leq c_2 g(n)$



funkce $f(n)$ a $g(n)$ rostou stejně rychle

Donald E. Knuth: *Big Omicron and big Omega and big Theta*.
ACM SIGACT, Volume 8 Issue 2, April-June 1976, pp. 18 - 24.

Θ notace - příklady

- $8n^2 - 88n + 8 = \Theta(n^2)$
  protože $8n^2 - 88n + 8 = \mathcal{O}(n^2)$ a současně $8n^2 - 88n + 8 = \Omega(n^2)$
- $8n^2 - 88n + 8 \neq \Theta(n^3)$ protože $8n^2 - 88n + 8 \neq \Omega(n^3)$
- $8n^2 - 88n + 8 \neq \Theta(n)$ protože $8n^2 - 88n + 8 \neq \mathcal{O}(n)$

Θ notace - příklad

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

- musíme najít **kladné** konstanty c_1, c_2 a n_0 takové, že

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

  platí pro všechna $n \geq n_0$
- po úpravě dostáváme

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

- pravá nerovnost platí pro každé $n \geq 1$ jestliže zvolíme $c_2 \geq 1/2$
- levá nerovnost platí pro každé $n \geq 7$ jestliže zvolíme $c_1 \leq 1/14$
- volba $c_1 = 1/14$, $c_2 = 1/2$ a $n_0 = 7$ dokazuje platnost vztahu $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Asymptotická notace - vlastnosti

tranzitivita

$f(n) = \Theta(g(n))$ a $g(n) = \Theta(h(n))$ implikuje $f(n) = \Theta(h(n))$

$f(n) = \mathcal{O}(g(n))$ a $g(n) = \mathcal{O}(h(n))$ implikuje $f(n) = \mathcal{O}(h(n))$

$f(n) = \Omega(g(n))$ a $g(n) = \Omega(h(n))$ implikuje $f(n) = \Omega(h(n))$

reflexivita

$f(n) = \Theta(f(n))$ podobně pro $\mathcal{O}$ a Ω

symetrie

$f(n) = \Theta(g(n))$ právě když $g(n) = \Theta(f(n))$

transpozice

$f(n) = \mathcal{O}(g(n))$ právě když $g(n) = \Omega(f(n))$

poznámka: ne každá dvojice funkcí je asymptoticky srovnatelná

Návrh a analýza algoritmů

Návrh algoritmů

ideální svět návod (*algoritmus*) „jak konstruovat algoritmy“

realita osvědčené postupy

- iterativní přístup
- rekurzivní přístup (*rozděl a panuj*, *divide et impera*, *divide and conquer*)
- dynamické programování
- hladové techniky
- heuristiky
- náhodnostní techniky
- aproximativní techniky
- parametrizované techniky
- . . . . . .

Rozděl a panuj - principy

Nothing is particularly hard if you divide it into small jobs *Henry Ford*

Rozděl **[divide]** problém na podproblémy, které mají menší velikost než původní problém.

Vyřeš **[conquer]** podproblémy stejným postupem(*rekurzívně*). Jestliže velikost podproblému je malá, použij přímé řešení.

Kombinuj **[combine]** řešení podproblémů a vyřeš původní problém.

Maximální a minimální prvek

- problém nalezení maximálního a minimálního prvku posloupnosti $S[1 \ldots n]$
- složitostní kritérium - počet porovnání prvků

MaxMin Iterative(S)

```
1 max ← S[1]
2 min ← S[1]
3 for i = 2 to n do
4     if S[i] > max then max ← S[i] fi
5     if S[i] < min then min ← S[i] fi
6 od
```

celkem $2(n-1)$ porovnání

Přístup Rozděl a panuj

1. posloupnost rozděl na dvě (stejně velké) podposloupnosti
2. najdi minimální a maximální prvek v obou podposloupnostech
3. kombinuj řešení podproblémů:
   maximálním prvek posloupnosti je větší z maximálních prvků podposloupností
   minimálním prvek posloupnosti je menší z minimálních prvků podposloupností

MaxMin(S, l, r)

```
1 if r = l then return (S[l], S[r]) fi
2 if r = l + 1 then return (max(S[l], S[r]), min(S[l], S[r])) fi
3 if r > l + 1 then (A, B) ← MaxMin(S, l, ⌊(l + r)/2⌋)
4                   (C, D) ← MaxMin(S, ⌊(l + r)/2⌋ + 1, r)
5                   return (max(A, C), min(B, D)) fi
```

iniciální volání $\textsc{MaxMin}(S, 1, n)$

Korektnost

konečnost výpočtu plyne z faktu, že každé rekurzivní volání se provede pro postupnost menší délky

správnost vypočítaného výsledku dokážeme indukcí vzhledem k délce vstupní posloupnosti

$\mathbf{n = 1, n = 2}$ provedou se příkazy v řádku 1, resp. v řádku 2

indukční předpoklad algoritmus vypočítá korektní hodnoty pro všechny posloupnosti délky nejvýše $n - 1$ $(n > 1)$

platnost tvrzení pro $\mathbf{n}$ dle indukčního předpokladu jsou čísla A a B maximálním a minimálním prvkem posloupnosti $S[1, \ldots, \lfloor (1 + n)/2 \rfloor]$, stejně tak čísla C a D jsou maximálním a minimálním prvkem posloupnosti $S[\lfloor (1 + n)/2 \rfloor + 1, \ldots, n]$
větší z čísel A, C je pak maximálním prvkem posloupnost $A[1, \ldots, n]$ a menší z čísel B, D jejím minimem

Složitost

rozděl problém na podproblémy menší velikosti

vyřeš podproblémy

kombinuj řešení podproblémů a vyřeš původní problém

n je délka vstupu, podproblémy mají velikosti $n_1, n_2, \ldots, n_k$
$T(\cdot)$ je časová složitost výpočtu na vstupu délky n

$$T(n) = \text{složitost rozdělení} + T(n_1) + T(n_2) + \ldots T(n_k) + \text{složitost kombinace}$$

konkrétně pro algoritmus MaxMin, složitost je počet porovnání prvků posloupnosti

$$T(n) = \begin{cases} 0 + T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & \text{pro } n > 2 \\ 1 & \text{pro } n = 2 \\ 0 & \text{pro } n = 1 \end{cases}$$

Složitost

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & \text{pro } n > 2 \\ 1 & \text{pro } n = 2 \\ 0 & \text{pro } n = 1 \end{cases}$$

indukcí vzhledem k n ověříme, že pro $n > 1$ platí

$$T(n) \leq \frac{5}{3}n - 2$$

indukční základ $n = 2$ $T(2) = 1 < \frac{5}{3} \cdot 2 - 2$
indukční předpoklad nerovnost platí pro všechny hodnoty i, $2 \leq i < n$
platnost pro n

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 \qquad \textit{využijeme indukční předpoklad}$$
$$\leq \frac{5}{3}\lfloor n/2 \rfloor - 2 + \frac{5}{3}\lceil n/2 \rceil - 2 + 2 = \frac{5}{3}n - 2$$

Kdo je nejrychlejší???

Min1(S, l, r)

$minimum \leftarrow S[l]$
for $i = l + 1$ **to** r **do**
    if $minimum > S[i]$ **then** $minimum \leftarrow S[i]$ **fi od**
return $minimum$

Min2(S, l, r)

if $r = l$ **then return** $S[r]$ **fi**
if $r > l$ **then** $A \leftarrow \textsc{Min2}(S, l, \lfloor (l + r)/2 \rfloor)$
$B \leftarrow \textsc{Min2}(S, \lfloor (l + r)/2 \rfloor + 1, r)$
return $\min(A, B)$ **fi**

Min3(S, l, r)

if $r = l$ **then return** $S[r]$ **fi**
if $r > l$ **then** $A \leftarrow \textsc{Min3}(S, l, r - 1)$
return $\min(A, S[r])$ **fi**

Složitost rekurzivních algoritmů

- složitost obvykle zapíšeme pomocí **rekurentní rovnice**, která vyjadřuje složitost výpočtu na vstupu velikosti n pomocí složitosti výpočtů na menších vstupech

- označme $T(n)$ časovou složitost výpočtu na vstupu délky n
- pro dostatečně malý vstup ($n \leq c$) si přímé řešení vyžaduje konstantní čas
- velký vstup rozdělíme na a podproblémů z nichž každý má velikost $1/b$ velikosti původního vstupu
- řešení každého podproblému si vyžádá čas $T(n/b)$
- označme $D(n)$ čas potřebný na konstrukci podproblémů a $C(n)$ čas potřebný na kombinaci řešení podproblémů a nalezení řešení původního problému

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{jinak} \end{cases}$$

jak najít řešení [1] *rekurentní rovnice???*

[1] nerekurzivní popis funkce, která splňuje podmínky rovnice

Řešení rekurentních rovnic

substituční metoda „uhodneme“ řešení a dokážeme jeho správnost matematickou indukcí

metoda rekurzivního stromu konstruujeme strom, jehož vrcholy vyjadřují složitost jednotlivých rekurzivních volání; výslednou složitost vypočítáme jako sumu ohodnocení vrcholů stromu

kuchařková věta *(master method)* vzorec pro řešení rekurentní rovnice tvaru $T(n) = aT(n/b) + f(n)$

Substituční metoda

1. „uhodni“ řešení
2. matematickou indukcí dokaž jeho korektnost

Příklad

$$T(n) = \begin{cases} 1 & \text{pro } n = 1 \\ 2T(\lfloor n/2 \rfloor) + n & \text{jinak} \end{cases}$$

1. $T(n) = \mathcal{O}(n \log n)$
2. indukcí dokážeme, že $T(n) \leq cn \log n$ pro dostatečně velké n a vhodně zvolenou konstantu c

dokazujeme $T(n) = \mathcal{O}(n \log n)$ pro rovnici

$$T(n) = \begin{cases} 1 & \text{pro } n = 1 \\ 2T(\lfloor n/2 \rfloor) + n & \text{jinak} \end{cases}$$

Indukční krok

- předpokládejme, že tvrzení platí pro všechna $m < n$, tj. speciálně pro $m = \lfloor n/2 \rfloor$ platí $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)$
- využitím indukčního předpokladu dokážeme platnost tvrzení pro n

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)) + n \\ &\leq cn \log(n/2) + n \\ &= cn \log n - cn \log 2 + n \\ &= cn \log n - cn + n \\ &\leq cnlogn \end{aligned}$$

dokazujeme $T(n) = \mathcal{O}(n \log n)$ pro rovnici

$$T(n) = \begin{cases} 1 & \text{pro } n = 1 \\ 2T(\lfloor n/2 \rfloor) + n & \text{jinak} \end{cases}$$

- jako indukční základ nemůžeme použít případ $n = 1$, protože neplatí $T(1) \leq cn \log n = c1 \log 1 = 0$
- využijeme, že dokazujeme asymptotický vztah a proto stačí, aby nerovnost platila pro všechna dostatečně velká n
- tvrzení dokážeme pro $n \geq 2$
- základem indukce bude platnost vztahu pro $n = 2$ a $n = 3$
- pro $n > 3$ závisí hodnota $T(n)$ jenom od $T(2)$ a $T(3)$

Indukční základ $n = 2$ a $n = 3$

- dosazením do rovnice zjistíme, že $T(2) = 4$ a $T(3) = 5$
- zvolíme konstantu $c \geq 1$ tak, aby pro $n = 2$ a $n = 3$ platilo $T(n) \leq cn \log n$
- dobrá volba je $c \geq 2$, protože platí $T(2) \leq c \cdot 2 \log 2$ i $T(3) \leq c \cdot 3 \log 3$

Metoda rekurzivního stromu

- „rozbalování rekurze“
- přehledný zápis pomocí stromu, jehož vrcholy vyjadřují složitost jednotlivých rekurzivních volání
- vrchol stromu je ohodnocen složitostí dekompozice a kompozice
- synové vrcholu odpovídají jednotlivým rekurzivním voláním
- výslednou složitost vypočítáme jako sumu ohodnocení vrcholů, obvykle sečítáme po jednotlivých úrovních stromu

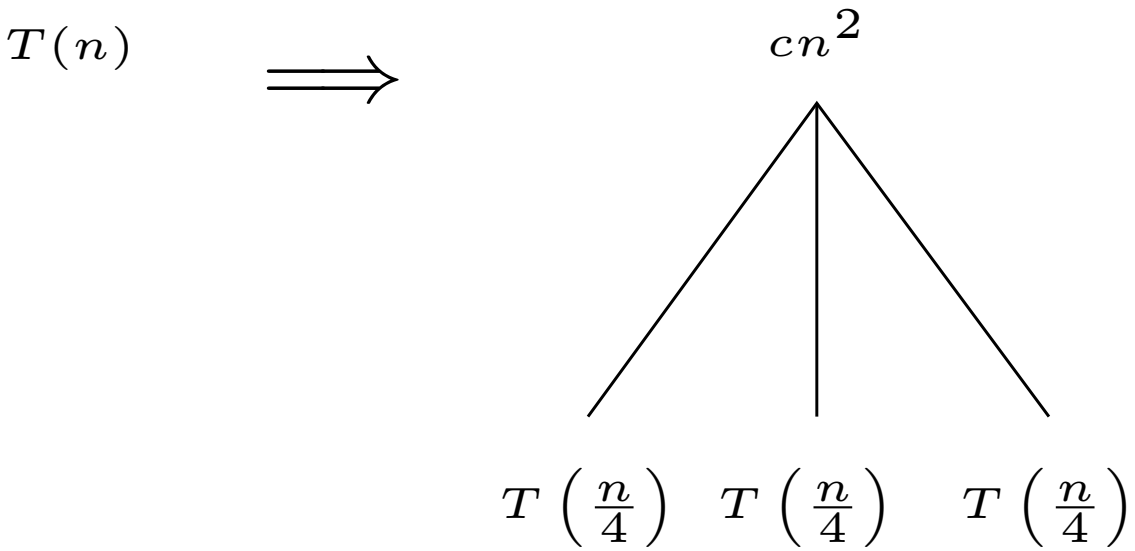
metodu můžeme použít pro

- nalezení přesného řešení (je nutné přesné počítání)
- pro získání odhadu na řešení rekurentní rovnice; pro důkaz řešení se pak použije substituční metoda

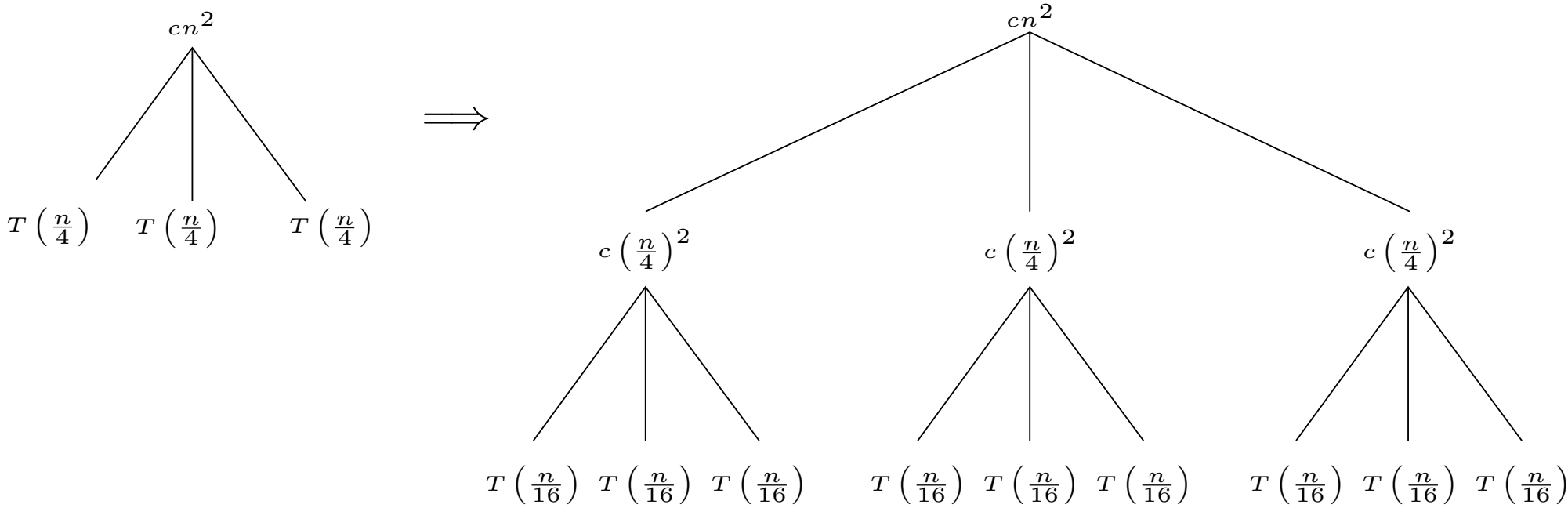
Metoda rekurzivního stromu - příklad

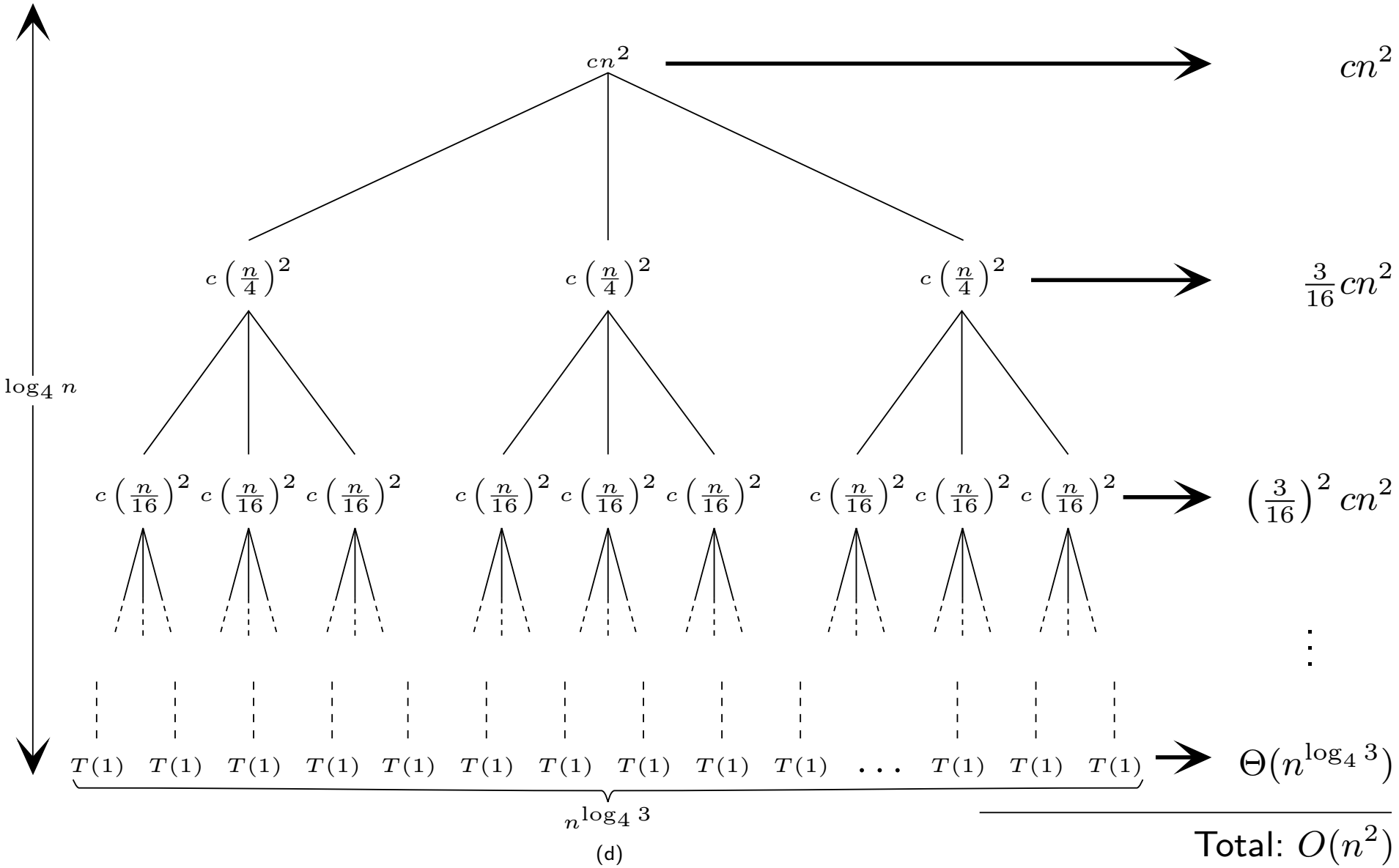
$$T(n) = \begin{cases} 1 & \text{pro } n = 1 \\ 3T(\lfloor n/4 \rfloor) + cn^2 & \text{jinak} \end{cases}$$

metodu použijeme pro získání odhadu řešení, můžeme proto předpokládat, že n je mocninou 4



$$T(n) = \begin{cases} 1 & \text{pro } n = 1 \\ 3T(\lfloor n/4 \rfloor) + cn^2 & \text{jinak} \end{cases}$$




cn^2
cn^2
$c\left(\frac{n}{4}\right)^2$
$c\left(\frac{n}{4}\right)^2$
$c\left(\frac{n}{4}\right)^2$
$\frac{3}{16}cn^2$
$\log_4 n$
$c\left(\frac{n}{16}\right)^2$ $c\left(\frac{n}{16}\right)^2$ $c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$ $c\left(\frac{n}{16}\right)^2$ $c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$ $c\left(\frac{n}{16}\right)^2$ $c\left(\frac{n}{16}\right)^2$
$\left(\frac{3}{16}\right)^2 cn^2$
$T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ ... $T(1)$ $T(1)$ $T(1)$
$\Theta(n^{\log_4 3})$
$n^{\log_4 3}$
(d)
Total: $O(n^2)$

- **kořen** má hloubku 0

- **(vnitřní) vrchol** v hloubce i je označen složitostí $c(n/4^i)^2$
- počet vrcholů s hloubkou i je 3^i
- součet složitostí vrcholů v hloubce i je $3^i c(n/4^i)^2 = (3/16)^i cn^2$

- **list** je označen složitostí 1 (základ rekurentní rovnice) a má hloubku $i = \log_4 n$ (protože $n/4^{\log_4 n} = 1$)
- počet listů je $3^{\log_4 n} = n^{\log_4 3}$

- sumací přes všechny úrovně dostáváme

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + n^{\log_4 3}$$

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{1}{1 - (3/16)} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13}cn^2 + n^{\log_4 3}$$

$$= \mathcal{O}(n^2)$$

hodnotu $T(n) = \mathcal{O}(n^2)$ použijeme jako odhad pro substituční metodu

Kuchařková věta (Master method)

Nechť $a \geq 1$ a $b > 1$ jsou konstanty, $f(n)$ je polynomiální funkce, a nechť $T(n)$ je definována na nezáporných číslech rekurentní rovnicí

$$T(n) = aT(n/b) + f(n)$$

Potom platí

$$T(n) = \begin{cases} \Theta(f(n)) & \text{když } af(n/b) = \kappa f(n) \text{ pro nějakou konstantu } \kappa < 1 \\ \Theta(n^{\log_b a}) & \text{když } af(n/b) = Kf(n) \text{ pro nějakou konstantu } K > 1 \\ \Theta(f(n) \log_b n) & \text{když } af(n/b) = f(n) \end{cases}$$

Kuchařková věta - alternativní varianta

Nechť $a \geq 1$, $b > 1$ a $d \geq 0$ jsou konstanty a nechť $T(n)$ je definována na nezáporných číslech rekurentní rovnicí

$$T(n) = aT(n/b) + \Theta(n^c)$$

Potom platí

$$T(n) = \begin{cases} \Theta(n^c) & \text{když } a < b^c \quad \textit{případ 1} \\ \Theta(n^c \log n) & \text{když } a = b^c \quad \textit{případ 2} \\ \Theta(n^{\log_b a}) & \text{když } a > b^c \quad \textit{případ 3} \end{cases}$$

věta platí i ve variantě pro $\mathcal{O}$ a Ω

Příklady použití kuchařkové věty I

- $T(n) = 4T(n/2) + 1 \implies T(n) = \Theta(n^2)$
  případ 3, $a = 4, b = 2, c = 0,\ 4 > 2^0$
- $T(n) = 4T(n/2) + n \implies T(n) = \Theta(n^2)$
  případ 3, $a = 4, b = 2, c = 1,\ 4 > 2^1$
- $T(n) = 4T(n/2) + n^2 \implies T(n) = \Theta(n^2 \log n)$
  případ 2, $a = 4, b = 2, c = 2,\ 4 = 2^2$
- $T(n) = 4T(n/2) + n^3 \implies T(n) = \Theta(n^3)$
  případ 1, $a = 4, b = 2, c = 3,\ 4 < 2^3$

Příklady použití kuchařkové věty II

- $T(n) = 2T(n/2) + 1 \implies T(n) = \Theta(n)$
  případ 3, $a = 2, b = 2, c = 0,\ 2 > 2^0$
- $T(n) = 2T(n/2) + n \implies T(n) = \Theta(n \log n)$
  případ 2, $a = 2, b = 2, c = 1,\ 2 = 2^1$
- $T(n) = 2T(n/2) + n^2 \implies T(n) = \Theta(n^2)$
  případ 1, $a = 2, b = 2, c = 2,\ 2 < 2^2$
- $T(n) = 2T(n/2) + n^3 \implies T(n) = \Theta(n^3)$
  případ 1, $a = 2, b = 2, c = 3,\ 2 < 2^3$

Příklady

Hanojské věže

$$T(n) = 2T(n-1) + 1, \text{ základní případ } T(0) = 0$$

$$T(n) = 2^n - 1$$

MergeSort

$$T(n) \leq 2T(n/2) + \Theta(n)$$

$$T(n) = \Theta(n \log n)$$

násobení celých čísel

$$T(n) = 4T(n/2) + \mathcal{O}(n)$$

$$T(n) = \mathcal{O}(n^2)$$

Strassenův algoritmus pro násobení celých čísel

$$T(n) = 7T(n/2) + \mathcal{O}(n^2)$$
$$T(n) = \mathcal{O}(n^{\log_b a}) = \mathcal{O}(n^{\log_2 7}) = \mathcal{O}(n^{2.81})$$

Jak nepoužívat rekurzi - nedefinovaný výpočet

Bad_Factorial(n)

return $n \cdot \textsc{Bad_Factorial}(n-1)$

chybí základ rekurze

Jak nepoužívat rekurzi - nekonečný výpočet

Awful_Factorial(n)

if $n = 0$ **then return** 1
else return $\frac{1}{n+1}\textsc{Awful\_Factorial}(n+1)$ **fi**

Beta(n)

if $n = 1$ **then return** 1
else return $n(n-1)\textsc{Beta}(n-2)$ **fi**

Jak nepoužívat rekurzi - složitost

Fibonacciho posloupnost

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

RecFibo(n)

if $n < 2$ **then return** n
else return $\textsc{RecFibo}(n-1) + \textsc{RecFibo}(n-2)$ **fi**

časová složitost algoritmu

$$T(0) = 1, \quad T(1) = 1, \quad T(n) = T(n-1) + T(n-2) + 1$$

$$T(n) = \Theta(\phi^n), \quad \phi = (\sqrt{5} + 1)/2$$

Jak nepoužívat rekurzi - složitost

Fibonacciho posloupnost

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

IterFibo(n)

$F[0] \leftarrow 0$
$F[1] \leftarrow 1$
for $i = 1$ **to** n **do**
    $F[i] \leftarrow F[i-1] + F[i-2]$ **od**
return $F[n]$

časová složitost algoritmu

$$T(n) = \Theta(n)$$

Jak používat rekurzi - význam definice podproblémů

problém maximální podposloupnosti

- dané je pole celých čísel $A[1..n]$
- cílem je najít takové indexy $1 \leq i \leq j \leq n$, pro které je suma $A[i] + \cdots + A[j]$ maximální

- 13, -3, -25, 20 -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7
- řešením je 18, 20, -7, 12

- řešení *hrubou sílou* - prozkoumat všechny dvojice indexů i, j
- kvadratická složitost
- existuje lepší řešení
- *rozděl a panuj*?

Rozděl a panuj

daná je posloupnost $A[low \dots high]$ a hodnota mid

hledané řešení $A[i \dots j]$

(A) je podposloupností $A[low \dots mid]$ $(low \leq i \leq j \leq mid)$

(B) je podposloupností $A[mid + 1 \dots high]$ $(mid + 1 \leq i \leq j \leq high)$

(C) zasahuje do obou podposloupností $(low \leq i \leq mid < j \leq high)$

- **(A)** a **(B)** jsou problémy stejného typu jako původní problém
- **(C)** ????

Rozděl a panuj

daná je posloupnost $A[low \dots high]$ a hodnota mid

hledané řešení $A[i \dots j]$

(A) je podposloupností $A[low \dots mid]$ ($low \leq i \leq j \leq mid$)

(B) je podposloupností $A[mid + 1 \dots high]$ ($mid + 1 \leq i \leq j \leq high$)

(C) zasahuje do obou podposloupností ($low \leq i \leq mid < j \leq high$)

- **(A)** a **(B)** jsou problémy stejného typu jako původní problém
- **(C)** ????
- pro řešení **(C)** stačí poznat podposloupnosti tvaru $A[i \dots mid]$ a $A[mid + 1 \dots j]$ s maximální sumou

Případ C

FIND_MAX_CROSSING_SUBARRAY

F_M_C_S$(A, low, mid, high)$

```
1  leftsum ← −∞
2  sum ← 0
3  for i = mid downto low do
4       sum ← sum + A[i]
5       if sum > leftsum then leftsum ← sum
6                             maxleft ← i fi od
7  rightsum ← −∞
8  sum ← 0
9  for j = mid + 1 to high do
10      sum ← sum + A[j]
11      if sum > rightsum then rightsum ← sum
12                             maxright ← j fi od
13 return (maxleft, maxright, leftsum + rightsum)
```

Algoritmus

Find_Maximum_Subarray

F_M_S$(A, low, high)$

```
1  if high = low
2     then return (low, high, A[low])
3      else mid = ⌈(low + high)/2⌉
4           (leftlow, lefthigh, leftsum) ← F_M_S(A, low, mid)
5           (rightlow, righthigh, righttsum) ← F_M_S(A, mid + 1, high)
6           (crosslow, crosshigh, crosssum) ← F_M_C_S(A, low, mid, high) fi
7  if leftsum ≥ rightsum ∧ leftsum ≥ crosssum
8     then return (leftlow, lefthigh, leftsum) fi
9  if leftsum ≤ rightsum ∧ rightsum ≥ crosssum
10    then return (rightlow, righthigh, rightsum)
11     else return (crosslow, crosshigh, crosssum) fi
```

Složitost

procedura FIND_MAX_CROSSING_SUBARRAY$(A, low, mid, high)$

- označme $n = high - low + 1$
- jedna iterace obou cyklů má konstantní složitost
- počet iterací cyklu pro levou část posloupnosti je $mid - low + 1$
- počet iterací cyklu pro pravou část posloupnosti je $high - mid$
- celková složitost je $\Theta(n)$

algoritmus FIND_MAXIMUM_SUBARRAY

dekompozice a kompozice v konstantním čase, řešení problému (C) v čase $\Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 2T(n/2) + \Theta(n) & \text{jinak} \end{cases}$$

$$T(n) = \Theta(n \log n)$$

Lokální maximum

- dané pole $A[1 \ldots n]$ celých čísel
- úkolem je najít lokální maximum v A, tj. takový index i, pro který platí $A[i] \geq A[i+1]$ a současně $A[i] \geq A[i-1]$
- krajní prvek je lokálním maximem právě když $A[1] \geq A[2]$ resp. $A[n] \geq A[n-1]$
- 1 2 6 5 4 7 3 8

Algoritmus 1

- *hrubá síla*
- otestuj, zda $A[1], A[2], \ldots, A[n]$ jsou lokálním maximem
- každý test složitosti $\Theta(1) \Longrightarrow$ složitost $\mathcal{O}(n)$

Algoritmus 2

- najdi maximální prvek v A
- složitost $\Theta(n)$

Lokální maximum

Algoritmus 3 - rozděl a panuj

- zkontroluj prvek na pozici i
- jestliže $A[i]$ je lokálním maximem, hotovo
- v opačném případě má $A[i]$ většího souseda
- jestliže $A[i-1] > A[i]$, tak lokální maximum je vlevo
- jestliže $A[i+1] > A[i]$, tak lokální maximum je vpravo
- pokračuj s posloupností $A[: i-1]$ resp. $A[i+1 :]$

- v nejhorším případě rekurze s $\max\{i-1, n-i\}$ prvky
- balance: $i - 1 = n - i$, tj. $i = (n-1)/2$

- $T(n) = T(n/2) + \Theta(1)$
- $T(n) = \Theta(logn)$

Lokální maximum - dvourozměrné

- daná $n \times n$ matice vzájemně různých celých čísel
- lokálním maximem je prvek $M[i, j]$, který je větší než všichni jeho sousedé, tj. prvky $M[i+1, j]$, $M[i-1, j]$, $M[i, j+1]$ a $M[i, j-1]$
- prvek na hranici matice je lokálním maximem právě když je větší než jeho sousedé v matici

Algoritmus 1

- *hrubá síla*
- otestuj všech n^2 prvků matice
- složitost $\Theta(n^2)$

Algoritmus 2

- redukce na problém lokálního maxima v poli
- najdi maximální prvek v každém sloupci
- v posloupnosti maximálních prvků najdi lokální maximum
- složitost $\Theta(n^2)$ *(složitost hledání maximálních prvků v sloupcích!!!)*

Lokální maximum - dvourozměrné

- daná $n \times n$ matice vzájemně různých celých čísel
- lokálním maximem je prvek $M[i, j]$, který je větší než všichni jeho sousedé, tj. prvky $M[i+1, j]$, $M[i-1, j]$, $M[i, j+1]$ a $M[i, j-1]$
- prvek na hranici matice je lokálním maximem právě když je větší než jeho sousedé v matici

Algoritmus 3

- redukce na problém lokálního maxima v poli
- v posloupnosti maximálních prvků najdi lokálním maximum
- maximální prvek v sloupci hledej až když ho potřebuješ
- složitost $\Theta(n \log n)$

existuje efektivnější řešení???

je možné najít lokálním maximum v $n \times n$ matici v čase $\Theta(n)$???

Lokální maximum - rozděl a panuj

- **okno** - $n \times n$ matice
- **rám okna** - první, prostřední a poslední řádek okna, první, prostřední a poslední sloupec okna
- najdi maximální prvek g mezi $6n - 6$ prvkami rámu
- jestliže g je větší než jeho sousedé, hotovo
- v opačném případě má g většího souseda, který určitě nepatří do rámu
- pokračuj s podoknem, do kterého patří větší soused prvku g

Korektnost

1. zvolené podokno obsahuje lokální maximum

- nechť max je maximální prvek podokna
- $max \geq$ soused prvku $g > g$, , tj. všichni sousedé prvku max jsou menší a max je lokálním maximem

2. konečnost

- v případě, že lokální maximum není prvkem rámu podokna, algoritmus rekurzívně hledá maximum v jeho podokně
- rekurze se zastaví když rám okna pokrývá celé okno (tj. počet řádků / sloupců je nejvýše 3), v takovém případě algoritmus prověří každý prvek okna a podle tvrzení 1 musí toto okno obsahovat lokální maximum

3. prvek, který najde algoritmus ve zvoleném podokně, je lokálním maximem

- nechť m je maximální prvek rámu zvoleného podokna, platí $m \geq g$
- když algoritmus vrátí m, tak m je určitě větší než všichni jeho sousedé v podokně a současně je i větší než jeho sousedé obklopující podokno (všichni jsou menší než g)
- v opačném případě algoritmus vrátí prvek, který nepatří do rámu podokna, je větší než všichni jeho sousedé v podokně a tedy je lokálním maximem

Složitost

- vstupem problému je matice rozměrů $n \times n$, jako velikost problému uvažujeme hodnotu n
- $T(n) = T(n/2) + \Theta(n)$
- $T(n) = \Theta(n)$

Rekurzivní vs iterativní přístup

pro

- intuitivní, jednoduchý návrh
- důkaz korektnosti využitím matematické indukce
- analýza složitosti využitím rekurentní rovnice
- efektivní řešení

proti

- neefektivní implementace
- ne vždy podpora ze strany programovacího jazyka
- neefektivní řešení

- každý rekurzivní algoritmus lze převést na iterativní
- simulace zásobníku volání
- *(resp. dynamické programování)*
- jednoduchý přepis v případě *tail rekurze*

Rekurzivní vs iterativní přístup

tail rekurze -speciální případ rekurze, kde se po rekurzivním volání nedělá žádný výpočet

ANO

$\mathrm{F}(x, y)$
if $y = 0$ **then return** x
else $F(x \cdot y + x, y - 1)$ **fi**

NE

$\mathrm{G}(x)$
if $y = 0$ **then return** x
else $y \leftarrow G(x - 1)$ **fi**
return $x \cdot y$

Rekurzivní vs iterativní přístup

```
F(x, y)
if y = 0 then return x
         else F(x · y + x, y − 1) fi
```

```
F(x, y)
label : if y = 0 then return x
                 else x ← x · y + x
                      y ← y − 1
                      goto label fi
```

```
F(x, y)
ret ← x
for y = 1 to y do
    ret ← ret · y + ret od
return ret
```

Rekurzivní vs iterativní přístup

```
BIN SEARCH(x, A, left, right)
if right = left then return A[left] == x fi
if right < left then mid = ⌊(left + right)/2⌋
                     if A[mid] = x then return true fi
                     if A[mid] < x then left ← mid + 1
                                   else right ← mid fi
                     BIN SEARCH(x, A, left, right)
fi
```

```
BIN SEARCH(x, A, left, right)
while right < left do mid = ⌊(left + right)/2⌋
      if A[mid] = x then return true fi
      if A[mid] < x then left ← mid + 1
                    else right ← mid fi
od
```

Domácí úkol

Vstup: pole $A[1 \ldots n]$ *celých čísel*

CoToDela(n)

if $n = 1$ **then write** A
else for $i = 1$ **to** n **do**
CoToDela$(n-1)$
if n je liché **then** swap $A[1]$ a $A[n]$
else swap $A[i]$ a $A[n]$ **fi**
od fi

- co je výstupem algoritmu?
- jaká je složitost výpočtu?