

Úvod do problematiky

Karel Richta a kol.

Přednášky byly připraveny i s pomocí materiálů, které vyrobili Marko Berezovský, Petr Felkel, Josef Kolář, Michal Píše a Pavel Tvrdík

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

© Karel Richta a kol., 2024

Datové struktury a algoritmy, B6B36DSA
02/2024, Lekce 1

<https://cw.fel.cvut.cz/b232/courses/b6b36dsa/start>



Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Obsah:

- Základní pojmy
 - Algoritmus
 - Datová struktura
 - Správnost
 - Částečná správnost
 - Úplná správnost
 - Efektivita
 - Časová složitost algoritmu
 - Paměťová složitost algoritmu
 - Příklad
- Matematická indukce
- Rekurentní rovnice

Výpočetní problém

- Specifikace výpočetního problému má dvě části:
 - definici vstupu (vstupních dat) a
 - definici výstupu (výstupních dat).
- Instance výpočetního problému → nějaká konkrétní přípustná vstupní data. Obecně může k jedné instanci problému existovat několik správných výstupů (obecně se jedná o výpočet, představující relaci).
- Příklad: problém řazení čísel (Sorting Problem for Numbers).
 - Vstup: Posloupnost n čísel $Inp = \langle a_1, \dots, a_n \rangle$.
 - Výstup: Taková permutace $Out = \langle a'_1, \dots, a'_n \rangle$ stejných čísel z Inp , pro kterou platí, že $a'_1 \leq \dots \leq a'_n$.
- Instance problému řazení je např. zadání: seřadíte vzestupně posloupnost $Inp = \langle 75, 11, 34, 176, 59, 6, 54 \rangle$. Správným výstupem řazení je pak posloupnost $Out = \langle 6, 11, 34, 54, 59, 75, 176 \rangle$.
- Pozn.: Zde je výstup určen jednoznačně, neboť se žádné číslo neopakuje.

Algoritmus

- Neformálně: algoritmus je jakýkoli dostatečně přesný popis postupu výpočtu – dostatečně přesný znamená:
 - v každém okamžiku postupu víme, který krok bude následovat (postup je deterministický),
 - pro libovolná vstupní data postup vždy skončí (je konečný) a
 - lze jej spustit pro všechny instance problému (je hromadný).
- Mezi algoritmy a výpočetními problémy nemusí být vztah jedna ku jedné – stejný problém lze zpravidla řešit různými algoritmy.
- Algoritmus řešící nějaký výpočetní problém je správný (korektní), pokud pro každou vstupní instanci skončí se správným výstupem. Správnost algoritmu se musí dokázat (proč?).
- Algoritmus není program.
 - Program může být realizací algoritmu v určitém výpočetním prostředí.
- Různé algoritmy pro stejný problém mohou mít různou složitost.

Korektnost algoritmu

- Korektní algoritmus A pro problém V :
 - pro každou instanci problému V , algoritmus A skončí v konečném čase se správným výstupem.
- Pak říkáme, že algoritmus A řeší problém V (nebo je řešením pro V).
- Čili: algoritmus A není korektní, pokud:
 - pro některou instanci problému neskončí (např. se “zacyklí”) nebo
 - pro některou instanci problému skončí s nesprávným výstupem.
- Můžeme pak nejvýše hovořit o jisté proceduře (postupu).
- V DSA uvažujeme téměř výhradně pouze korektní algoritmy – tj. algoritmy, které skutečně řeší odpovídající problém.
- Důkaz korektnosti nemusí být jednoduchý – instancí problému bývá zpravidla nekonečně mnoho – ověřit postup pro všechna data nelze. Je nutno použít sofistikovanější metody, např. matematickou indukci.

Program

- Program P je zápis (implementace) algoritmu A v nějakém programovacím jazyce, spustitelný (proveditelný) na nějakém počítači s nějakým OS a SW prostředím.
- U návrhu a implementace programu nás zajímají SW inženýrské otázky, jako např.:
 - modularita,
 - ošetření výjimek,
 - datová abstrakce a ošetření vstupu a výstupu,
 - dokumentace.
- Jednomu algoritmu A mohou odpovídat různé programy P_1, P_2, \dots dokonce v různých programovacích jazycích.
- Složitosti provedení jednotlivých programů P_i téhož algoritmu se mohou lišit v závislosti na architektuře počítače — nikoli ale řádově.

Jak měřit a určovat složitost algoritmu?

- Analýza složitosti algoritmu: výpočet/odhad/predikce požadavků na výpočetní prostředky, které provedení algoritmu bude vyžadovat v závislosti na velikosti vstupních dat.
- Velikost (složitost) vstupních dat závisí na specifikaci daného problému, může se jednat např. o:
 - počet bitů reprezentace vstupního čísla,
 - délka vstupní posloupnosti čísel,
 - rozměry vstupní matice,
 - počet znaků vstupního textu,
 - ...
- Pro jeden problém V mohou existovat různé algoritmy A_1, A_2, \dots , pro jeho řešení. Jejich složitosti se mohou výrazně (= řádově) lišit svými nároky na výpočetní prostředky.
- Zvláště se mohou lišit svojí rychlostí (operační složitostí).
- Nebo nároky na použitou paměť (paměťovou složitostí).

Výpočetní (operační) složitost

- Varianty měření operační složitosti:
 - doba běhu algoritmu,
 - doba výpočtu,
 - počet provedených operací,
 - počet provedených instrukcí (aritmetických, logických, paměťových),
 - počet transakcí nad databází.
- Případová analýza složitosti:
 - Typický algoritmus obsahuje podmíněné příkazy nebo cykly \Rightarrow složitost algoritmu může obecně záviset nejen na velikosti vstupních dat, ale také na jejich hodnotách (říkáme pak, že algoritmus je citlivý na vstupní data).
- Proto je obecně potřebné vyjádřit složitost algoritmu:
 - v nejlepším případě,
 - v nejhorším případě,
 - v průměrném případě.

Umění analýzy složitosti algoritmu

- Odhad složitosti v průměrném případě je nejobtížnější, protože není vždy zřejmé, co to jsou vstupní data v "průměrném" případě.
- Typicky náhodně vygenerovaná data, ale to nemusí být v praxi splněno.
- Platí pro časovou, paměťovou, komunikační, transakční, . . .
- Umění analyzovat složitost algoritmu vyžaduje řadu znalostí a schopností:
 - znalosti odhadu řádu rychlosti růstu funkcí,
 - schopnost abstrakce výpočetního modelu,
 - matematické znalosti z diskrétní matematiky, kombinatoriky,
 - znalosti sčítání řad a řešení rekurentních rovnic,
 - znalosti základu teorie pravděpodobnosti,
 - a další.

RAM model

- Pro analýzu složitosti algoritmu potřebujeme model výpočetního systému, na kterém bude algoritmus hypoteticky implementován.
- Nejčastěji se používá matematická abstrakce stroje s přímým přístupem do paměti - jednoprocesorový model RAM (Random Access Machine):
 - data jsou uložena v adresovatelné paměti,
 - stroj umí aritmeticko-logické, řídicí, paměťové instrukce,
 - časová složitost instrukcí je jednotková nebo konstantní,
 - instrukce jsou prováděny postupně (sekvenčně).
- Neumí manipulace s větším množstvím dat naráz a práci s neomezeně velkými čísly (řetězci).
- Zpracování každé instrukce trvá přesně jeden krok.
- Nemá „keše“, disky a vstupně/výstupní zařízení (lze je ale simulovat pomocí přidružených datových struktur).
- Díky své obecnosti přežil tento model desetiletí.

Turingův stroj a RAM počítač

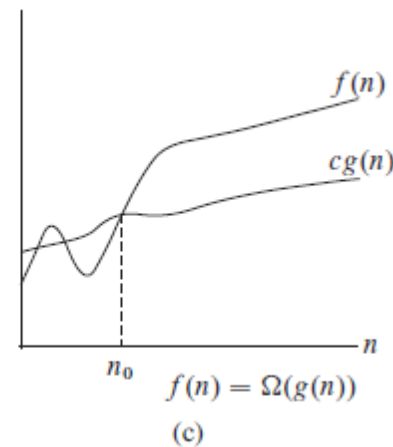
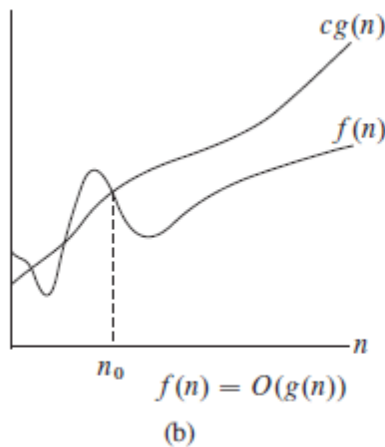
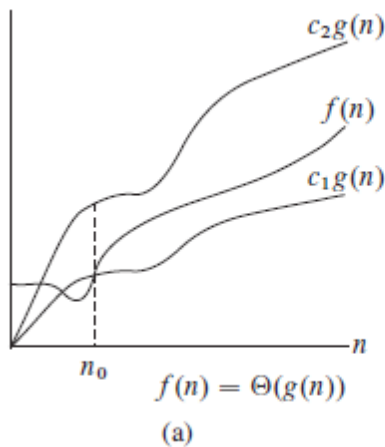
- RAM počítač je dostatečnou abstrakcí pokud neuvažujeme vstup ani výstup. Na druhé straně je paměť RAM počítače neomezená.
- Pokud bychom chtěli tento stroj přiblížit standardním modelům výpočtů, je možno rozšířit RAM počítač o vstupně/výstupní zařízení – např. Turingovu pásku.
- Ta je nekonečná na obě strany (vlevo i vpravo). Můžeme na ní zapisovat symboly (data) a poté je číst. Na pásku zapisujeme data pomocí čtecí/zápisové hlavy, která se může také pohybovat vlevo či vpravo.
- RAM počítač s Turingovou páskou tvoří tzv. Turingův stroj. Lze dokázat, že každá funkce realizovatelná Turingovým strojem je také realizovatelná RAM počítačem s nekonečnou pamětí.
- Existuje tzv. Churchova teze, která tvrdí, že všechny rekurzivně vyčíslitelné funkce jsou realizovatelné Turingovým strojem nebo RAM počítačem s nekonečnou pamětí. To ale dokázat nelze (proto teze), nikdy nebude existovat algoritmus, který by např. řešil pro libovolný program (resp. pro libovolný Turingův stroj) problém zastavení Turingova stroje.

Asymptotická složitost

- Zajímá nás, jak složitost algoritmu závisí na velikosti vstupních dat v limitě, pokud velikost instance problému poroste nade všechny meze.
- Pro první porozumění chování algoritmu nás nemusejí zajímat ani multiplikativní konstanty - říkáme, že nás zajímá asymptotická složitost.
- Je to složitost vyjádřená pro instance problému dostatečně velké, aby se jasně projevil řád růstu funkce složitosti v závislosti na velikosti vstupních dat.
- Asymptoticky lepší algoritmus bude lepší pro všechny instance problému kromě případného konečného počtu menších instancí.
- Asymptotická notace nám umožňuje zjednodušovat výrazy zanedbáním nepodstatných částí.
- Co znamená zápis $f(n) = 4n^3 + \Theta(n^2)$?
- Výraz $\Theta(n^2)$ zastupuje anonymní funkci z množiny $\Theta(n^2)$ (nějaká kvadratická funkce), jejíž konkrétní podobu prozatím zanedbáváme.

Grafické znázornění pro vysvětlení

- Pokud uvažujeme asymptotickou složitost, zajímá nás zejména chování pro velká vstupní data, pro singulární malá data (menší než n_0) se může chovat trochu jinak. Varianta (a) ukazuje tzv. průměrnou složitost – $f(n) = \Theta(g(n))$ se drží mezi $c_2g(n)$ a $c_1g(n)$ (pro $n \geq n_0$). Varianta (b) ukazuje tzv. asymptotickou horní mez - $f(n) = O(g(n))$ se drží pod $cg(n)$. Naopak varianta (c) ukazuje tzv. asymptotickou dolní mez - $f(n) = \Omega(g(n))$ se drží nad $cg(n)$.



Příklad

- Uvažme problém řazení a dva vývojáře.
- Jeden má rychlý počítač C_1 a navrhl řazení pomocí přímého zatřídování (INSERTIONSORT) – výstup tvoříme tak, že do prázdné posloupnosti postupně zatřídíme jednotlivé prvky vstupní posloupnosti.
- Druhý má pomalý počítač C_2 (např. 1000x pomalejší než C_1), ale navrhl řazení slučováním (MERGESORT) – vstup rozdělíme na dvě části které (rekurzivně) setřídíme a poté sloučíme do výsledku.
- Lze ukázat (to se naučíme později), že asymptotická složitost INSERTIONSORT bude $\Theta(n^2)$, zatímco složitost MERGESORT bude $\Theta(n \cdot \log_2 n)$.
- Pokud budeme třídit 10 milionů (10^7) čísel, počítač C_1 provede 10^{10} instrukcí za vteřinu, počítač C_2 provede 10^7 instrukcí za vteřinu (je 1000x pomalejší), pak:
$$(10^7)^2 / (10^{10}) = 10^4 \text{ vteřin} \geq (10^7 \cdot \log_2 10^7) / (10^7) = 161 \text{ vteřin}$$
- Tj. rychlejší C_1 počítá téměř 3 hodiny, pomalejší C_2 ani ne 3 minuty.

Základní funkce jedné proměnné

- dolní celá část $\lfloor x \rfloor$, horní celá část $\lceil x \rceil$,
- polynomiální
 - exponent > 1 , např. n^2 ,
 - lineární, např. n ,
 - exponent < 1 , např. \sqrt{n} ,
- exponenciální, např. 2^n ,
- polylogaritmické, např. $\log^3 n$,
- faktoriální $n!$,
- logaritmická iterace $\log^* n$,
- Fibonacciho čísla $F(n)$.
-

Srovnání časových složitostí

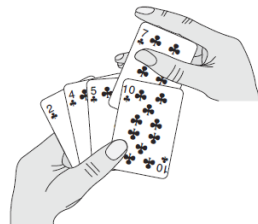
- Předpokládejme, že 1 operace trvá čas $1\mu s$ a že počet operací je dán funkcí v prvním sloupci. Pak doba výpočtu bude pro různé hodnoty n následující: (Připomeňme, že počet atomů ve vesmíru se odhaduje na 10^{80} a stáří vesmíru na 14×10^9 let.)

n	10	20	40	80	500	1000
$\log n$	$3,3 \mu s$	$4,3 \mu s$	$5 \mu s$	$5,8 \mu s$	$9 \mu s$	$10 \mu s$
n	$10 \mu s$	$20 \mu s$	$40 \mu s$	$60 \mu s$	$0,5 ms$	$1 ms$
$n \log n$	$33 \mu s$	$86 \mu s$	$0,2 ms$	$0,35 ms$	$4,5 ms$	$10 ms$
n^2	$0,1 ms$	$0,4 ms$	$1,6 ms$	$3,6 ms$	$0,25 s$	$1 s$
n^3	$1 ms$	$8 ms$	$64 ms$	$0,2 s$	$125 s$	$17 min$
n^4	$10 ms$	$160 ms$	$2,56 s$	$13 s$	$17 h$	$11,6 dní$
2^n	$1 ms$	$1 s$	$12,7 dní$	$3600 let$	$10^{137} let$	$10^{287} let$
$n!$	$3,6 s$	$77100 let$	$10^{34} let$	$10^{105} let$	$10^{1110} let$	$10^{2554} let$

Příklad: Řazení přímým vkládáním (INSERTIONSORT)

Algoritmus: vstup – posloupnost s:

1. Pokud je posloupnost s kratší než 2, skonči, $Out = Inp$.
2. Jinak postupně vyber prvek, který je na j -tém místě (j nabývá hodnot od 2 do délky posloupnosti – aktuálně je v čele) a zařaď jej na správné místo v posloupnosti – před nejbližší větší prvek (pokud existuje) v levé části posloupnosti.

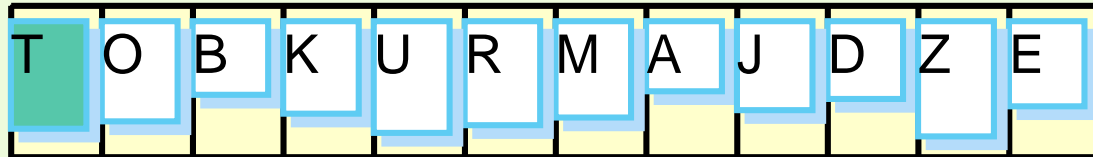


Program (pseudokód):

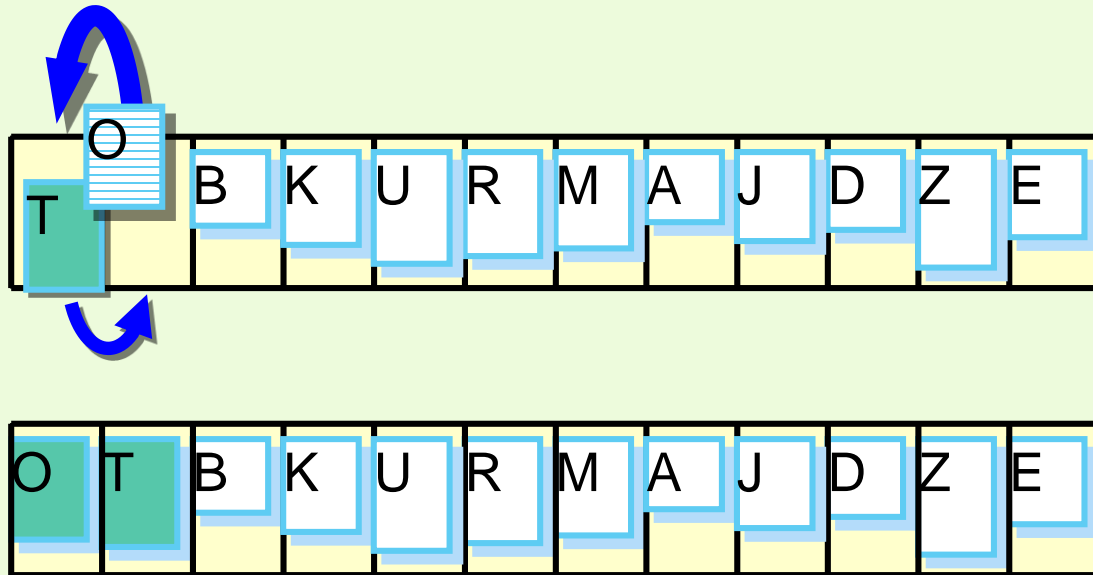
```
for (i = 1; i < n; i++) { // find & make
                               // place for s[i]
    insVal = s[i];
    j = i-1;
    while ((j >= 0) && (s[j] > insVal)) {
        s[j+1] = s[j];
        j--;
    }
    s[j+1] = insVal; // insert s[i]
}
```

INSERTIONSORT

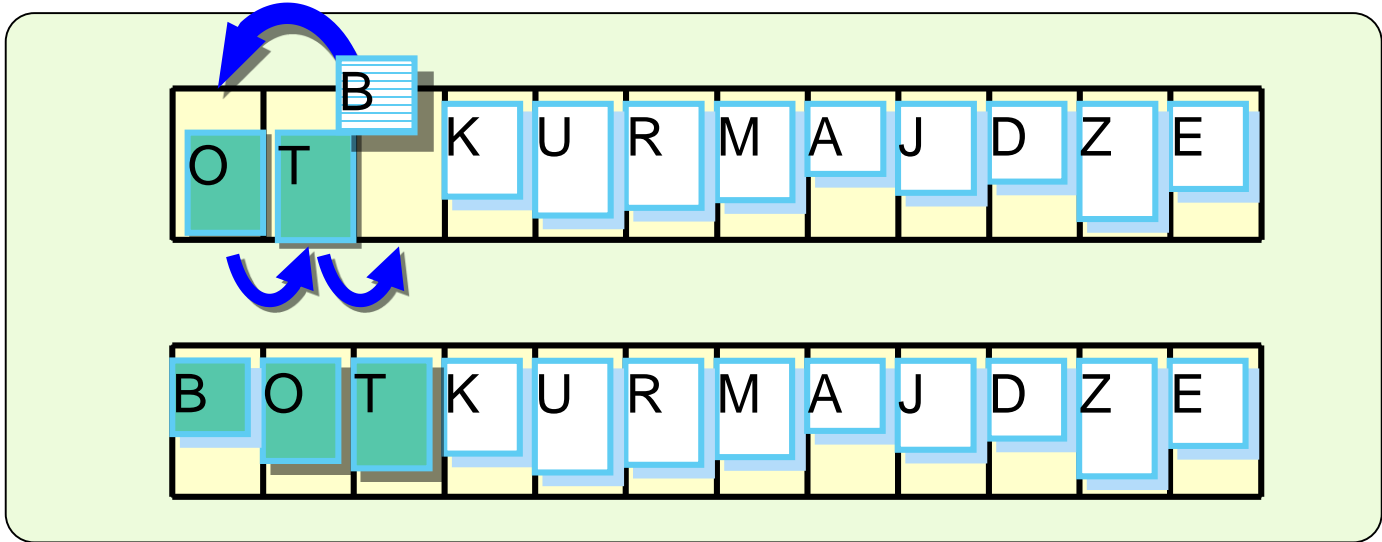
Start



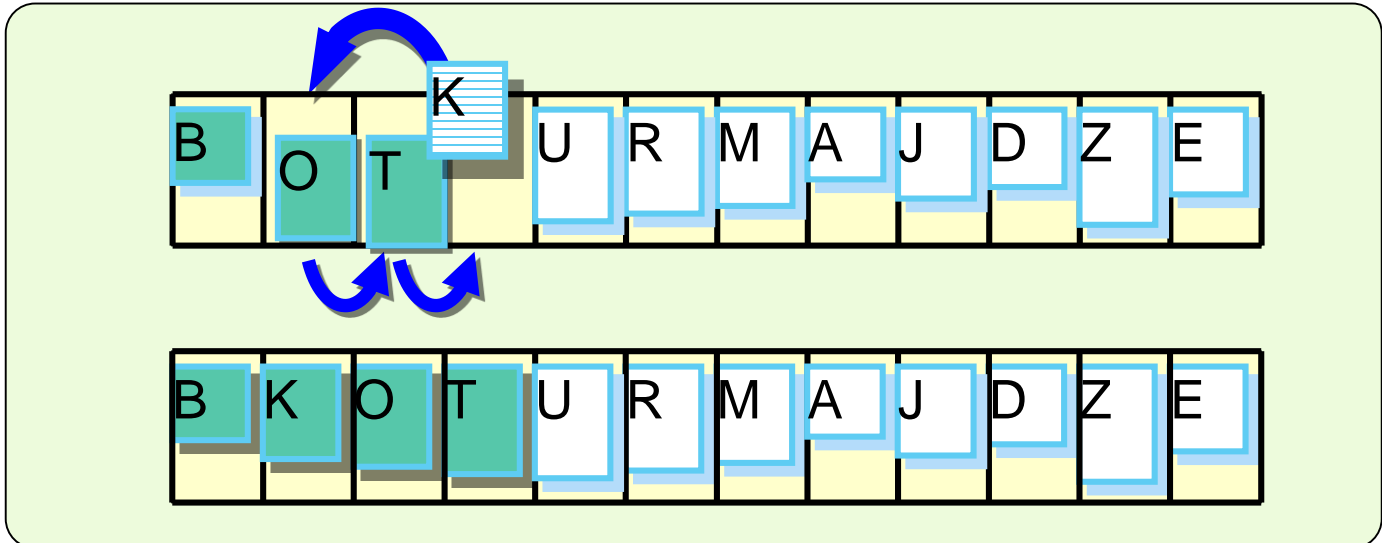
Krok 1



Krok 2



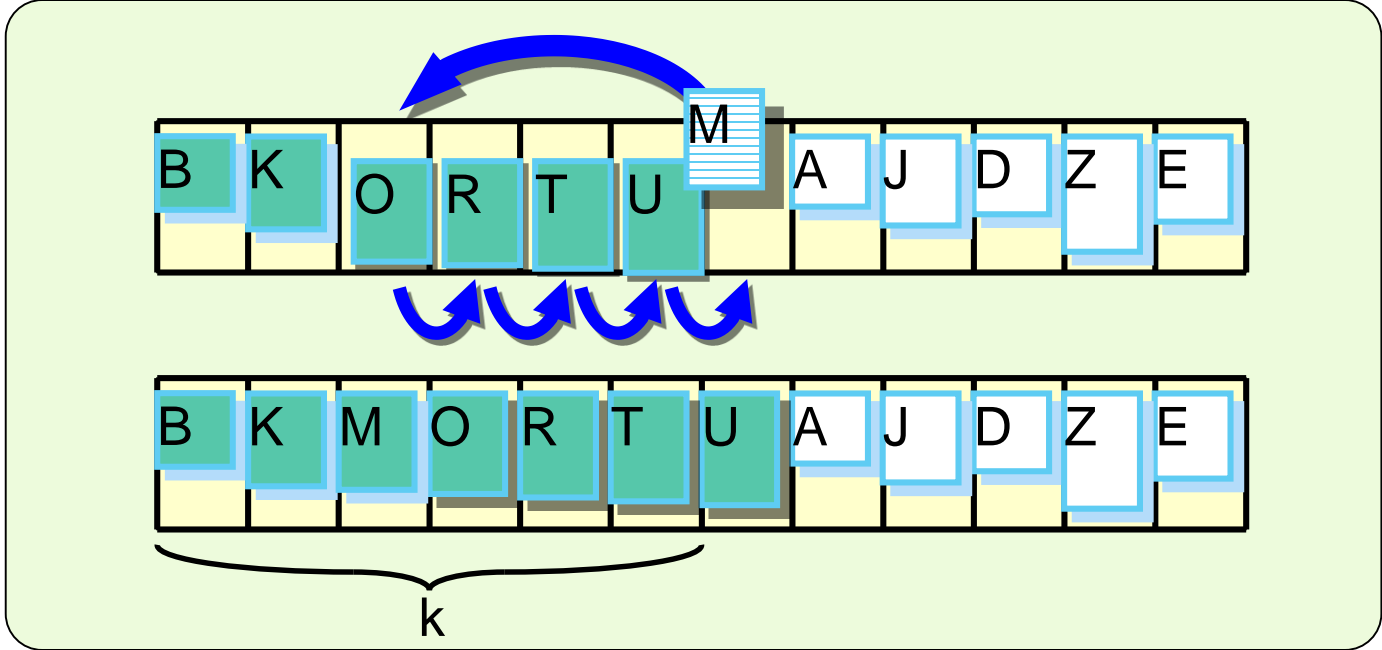
Krok 3



...

...

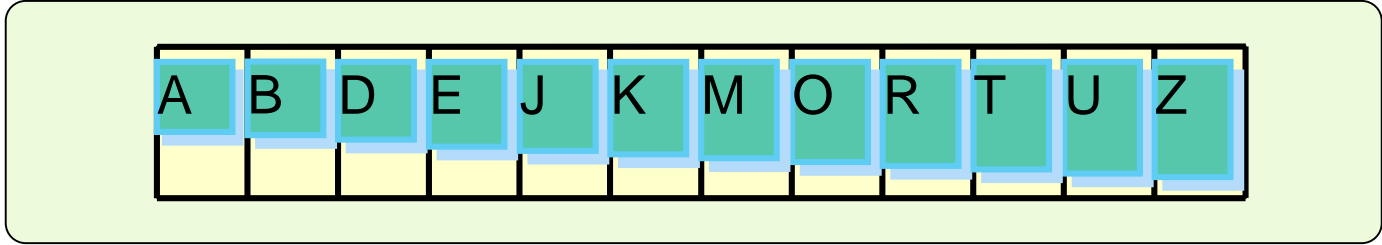
Krok k



...

...

seřazeno

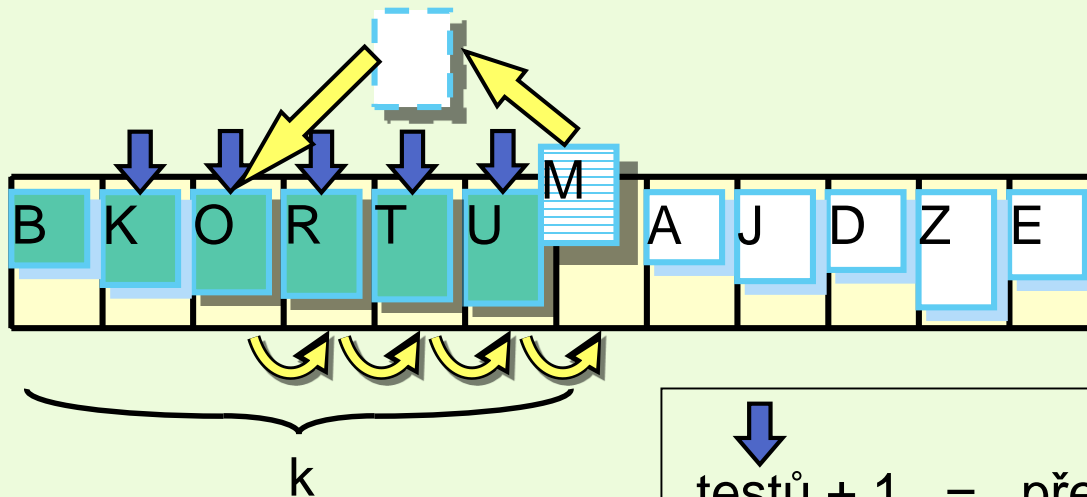




Je tato implementace korektní?

- **Formálně:** $\forall s$: posloupnost platí: výstup $\text{INSERTIONSORT}(s)$ je seřazená posloupnost a obsahuje permutaci prvků z s .
- Náznak důkazu:
- Ad a) konečnost – evidentně algoritmus po n krocích skončí.
- Ad b) správnost (indukcí podle délky posloupnosti s):
 - Pokud je s prázdná posloupnost ε , $Out = \text{INSERTIONSORT}(Inp)$ – tj. $\varepsilon = \text{INSERTIONSORT}(\varepsilon)$ a výstup je korektní.
 - Pokud je s posloupnost délky 1, $Inp = \langle a \rangle$ – tj. $Out = \langle a \rangle = \text{INSERTIONSORT}(\langle a \rangle)$ a výstup je korektní.
 - Předpokládejme, že INSERTIONSORT funguje správně pro posloupnost s délky n , musíme ověřit, že funguje správně i pro posloupnost s' délky $n+1$, tj.: $\text{INSERTIONSORT}(s')$ je seřazená, neboť $\text{INSERTIONSORT}(s)$ je seřazená (indukční předpoklad) a zařazením nového prvku do posloupnosti $\text{INSERTIONSORT}(s)$ vznikne seřazená posloupnost $\text{INSERTIONSORT}(s')$, protože jeden průběh cyklem zařadí nový prvek před nejbližší větší prvek v $\text{INSERTIONSORT}(s)$.

Složitost řazení vkládáním (INSERTIONSORT)

Krok k



 testů + 1 =  přesunů

testů	1	nejlepší případ
	k	nejhorší případ
	$(k+1)/2$	průměrný případ

přesunů	2	nejlepší případ
	k+1	nejhorší případ
	$(k+3)/2$	průměrný případ

Shrnutí

Celkem
testů

$n - 1$	$= \Theta(n)$	nejlepší případ
$(n^2 - n)/2$	$= \Theta(n^2)$	nejhorší případ
$(n^2 + n - 2)/4$	$= \Theta(n^2)$	průměrný případ

Celkem
přesunů

$2n - 2$	$= \Theta(n)$	nejlepší případ
$(n^2 + n - 2)/2$	$= \Theta(n^2)$	nejhorší případ
$(n^2 + 5n - 6)/4$	$= \Theta(n^2)$	průměrný případ

Asymptotická složitost INSERTIONSORT je $O(n^2)$ (!!)

Základní diskrétní posloupnosti a řady

- **Aritmetická posloupnost:**

Základní: $1 + 2 + \dots + n = \sum_{i=1}^n i = \frac{1}{2}n(n+1)$

Obecná: $a_1 + \dots + a_n = \frac{1}{2}n(a_1 + a_n)$.

- **Geometrická posloupnost:** Pro $x \neq 1$:

$$1 + x + x^2 + \dots + x^n = \sum_{i=0}^n x^i = \frac{x^{n+1} - 1}{x - 1}.$$

- **Obecná posloupnost:** Je-li $f(n)$ **monotonně rostoucí** funkce, lze součet řady $\sum_{k=m}^n f(k)$ aproximovat určitým integrálem:

$$\int_{m-1}^n f(x) dx \leq \sum_{k=m}^n f(k) \leq \int_m^{n+1} f(x) dx$$

Aplikace:

- ▶ **Příklad 1: Harmonická posloupnost:**

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n} = \sum_{k=1}^n \frac{1}{k} = \ln n + O(1).$$

- ▶ **Příklad 2: Součet kvadrátů:**

$$1 + 2^2 + 3^2 + \dots + n^2 = \sum_{k=1}^n k^2 \doteq \frac{n^3}{3} + \frac{n^2}{2}.$$

Základní kombinatorika

- **Permutace:** (prvky se neopakují, záleží na pořadí)
Počet k -permutací z n prvků je $\frac{n!}{(n-k)!} = n(n-1)\dots(n-k+1)$.
Počet permutací z n prvků je $n!$.
- **Permutace s opakováním:** (prvky se opakují, záleží na pořadí)

$$\frac{n!}{k_1!k_2!\dots k_d!},$$

kde $\sum_{i=1}^d k_i = n$.

- **Kombinace:** (prvky se neopakují, nezáleží na pořadí).
Počet k -kombinací z n prvků je $\binom{n}{k} = \frac{n!}{(n-k)!k!}$.
= Počet k -podmnožin z n -prvkové množiny.
= Počet n -bitových řetězců s k bity 1.
= Permutace s opakováním pro $d = 2$.
- **Variace** (prvky se opakují libovolně, záleží na pořadí):
Počet k -variací z n -prvkové množiny je n^k .

Techniky matematických důkazů

- Důkaz je postup, kdy pro nějaké tvrzení dokazujeme, že je pravdivé.
- Jeden typ důkazu je tzv. **konstruktivní důkaz**, kdy zkonstruujeme řešení, které má požadované vlastnosti.
- Jiný typ důkazu je **důkaz sporem** – hledáme protipříklad, kde tvrzení neplatí. Často to lze obrátit tak, že uvažujeme negaci nějakého tvrzení a dokazujeme, že neplatí.
- Pro nekonečné (spočetné) případy musíme použít metody **důkazu matematickou indukcí**. Skládá se ze dvou kroků – nejprve dokazujeme nějaký triviální počáteční krok a poté předpokládáme, že tvrzení platí pro problém velikosti n a dokazujeme, že pak platí i pro $n+1$. Z toho pak indukcí usuzujeme, že platí pro všechna n . Věříme totiž, že když z platnosti pro n plyne platnost pro $n+1$, bude to platit pro libovolně velké n – to je ale jen naše víra v indukci, třeba to za obzorem může být jinak.

(Pokud Darwin tvrdí, že předkem člověka je opice a platí, že opici se může narodit jen opice, pak je důsledek odvozený indukcí zřejmý.)

Konstruktivní důkaz

Zpravidla postupujeme tak, že hledáme řešení, pro které by tvrzení platilo.

Příklad tvrzení: Existuje prvočíslo větší než 8.

- Konstruktivní důkaz - konstruujeme řešení – probíráme čísla > 8 :
- Přirozené číslo 9 je dělitelné 1, 3 a 9 - není tedy prvočíslo.
- Přirozené číslo 10 je dělitelné 1, 2, 5 a 10 - není tedy prvočíslo.
- Přirozené číslo 11 je dělitelné pouze 1 a samo sebou. Pak ale je 11 prvočíslo a protože $11 > 8$, tvrzení platí.

Důkaz sporem

Zpravidla postupujeme tak, že předpokládáme opak dokazovaného tvrzení a hledáme protipříklad, který by to vyvrátil. Neplatí-li opak, musí platit dokazované tvrzení.

Příklad tvrzení: Neexistuje největší přirozené číslo.

- Důkaz sporem: Předpokládejme opak - že existuje největší přirozené číslo, např. X . Pak ale existuje i přirozené číslo $X+1$ (podle definice přirozených čísel). $X+1$ je ale větší než X , tj. X není největší a je zde spor.

Příklad tvrzení: Číslo 8 není prvočíslo.

- Důkaz sporem: Předpokládejme opak – 8 je prvočíslo. Pak musí být dělitelné pouze 1 a 8. Ale 8 je dělitelné 2, což je spor a 8 není prvočíslo.

Důkaz indukcí

Při důkazu indukcí zpravidla postupujeme tak, že nejprve dokazujeme počáteční krok – tvrzení musí platit pro minimální případ (např. pro 0). Poté zavedeme předpoklad (indukční hypotézu), že tvrzení platí pro problém rozsahu n a dokazujeme, že pak platí i pro $n+1$.

Příklad tvrzení: Součet prvních n přirozených čísel je $n(n+1)/2$.

- Počáteční krok: Ukažme, že toto tvrzení platí pro $n=1$. Dosadíme-li za n do vztahu: $n(n+1)/2 = 1 \cdot 2/2 = 1$ a tvrzení platí. (Lze uvažovat i 0.)
- Indukční krok: Předpokládejme, že toto tvrzení platí pro n , tj. že součet prvních n přirozených čísel je $\sum_{i=1}^n i = n(n+1)/2$. Dokažme, že pak tvrzení platí i pro $n+1$, tj. že součet prvních $n+1$ přirozených čísel je $(n+1)((n+1)+1)/2$:

$$\begin{aligned} \sum_{i=1}^{n+1} i &= \sum_{i=1}^n i + n + 1 = n(n+1)/2 + n + 1 = (n(n+1) + 2n + 1)/2 = (n^2 + n + 2n + 1)/2 = \\ &= (n^2 + 3n + 1)/2 = (n^2 + n + 2n + 2)/2 = (n+1)(n+2)/2 = (n+1)((n+1)+1)/2. \quad \text{q.e.d.} \end{aligned}$$

Důkaz indukcí (pokr.)

Příklad tvrzení: Libovolnou hodnotu známky v korunách větší než 3 lze vyjádřit pomocí známek s hodnotou 2 Kč a 5 Kč.

- Počáteční krok: Ukažme, že toto tvrzení platí pro $n=4$ (pro $n=1$ a $n=3$ tvrzení neplatí). Zde $4=2+2$, tj. hodnotu vyjádřit lze.
- Indukční krok: Předpokládejme, že toto tvrzení platí pro hodnotu n , tj. že tuto hodnotu lze vyjádřit pomocí 2 a 5. Musíme ukázat, že pak tvrzení platí i pro $n+1$.
 - Pokud vyjádření n obsahuje známku s hodnotou 5, nahradíme ji 3-mi známkami s hodnotou 2 – výsledkem je hodnota $n+1$.
 - Pokud vyjádření n neobsahuje známku s hodnotou 5, musí obsahovat nejméně 2 známky s hodnotou 2 ($n \geq 4$). Ty nahradíme známkou s hodnotou 5 – výsledkem je hodnota $n+1$.

q.e.d.

Důkaz indukcí (pokr.)

Příklad tvrzení: Každé přirozené číslo $n > 1$ je dělitelné nějakým prvočíslem.

- Počáteční krok: Pro $n = 2$, je zřejmě dělitelné 2 – což je prvočíslo.
- Indukční krok: Předpokládejme, že toto tvrzení platí pro hodnotu n , tj. že všechna x , taková že: $2 \leq x \leq n$ jsou dělitelná nějakým prvočíslem. Musíme ukázat, že pak tvrzení platí i pro $n+1$.
 - Pokud je $n+1$ prvočíslo, je dělitelné samo sebou a tvrzení platí.
 - Pokud $n+1$ není prvočíslo, pak $n+1 = x \times y$, kde $2 \leq x \leq n$ a $2 \leq y \leq n$. Podle indukčního předpokladu je x dělitelné nějakým prvočíslem q , tj. $x = a \times q$ - tímto prvočíslem je pak dělitelné i $n+1 = a \times q \times y$.

q.e.d.
- Pozn. Zde se jedná o tzv. silnou indukci, neboť musíme v indukčním kroku předpokládat, že tvrzení platí pro všechna čísla x , kde: $2 \leq x < n$.

Rekurzivní rovnosti

- Rekurzivní rovnosti představují způsob definice funkce (procedury, postupu, algoritmu) pomocí odkazu na ně samé.
- Teoretická informatika vyjadřuje tezi, že každou parciálně vyčíslitelnou funkci, která může být implementovaná na von Neumannovském počítači, lze vyjádřit rekurzivně.
- Později budeme studovat vztah mezi rekurzí a iterací – někdy lze rekurzivní řešení nahradit iterací, která bývá zpravidla efektivnější.

Př.: Funkce faktoriál se často vyjadřuje pomocí rekurzivní definice (pro $n \geq 0$):

$$\mathbf{faktoriál}(n) = n * \mathbf{faktoriál}(n-1)$$

$$\mathbf{faktoriál}(0) = 1$$

The End