# ALGORITMY

# Obsah Přednášky

- Algoritmus - definice a vlastnosti
- Členění algoritmů
- Terminologie - hodnotící funkce, fitness, optimalizace
- Příklady algoritmů

# Algoritmus

- Slovo algoritmus pochází ze jména významného perského matematika žijícího v první polovině1b 9. století (cca 780 –840), kterým byl Abūk ‘RightGuillemet]Abd Allā01h Muhammad ibn Mūksā01 al - Chwā01rizmī أبو عبد الله محمد بن موسى الخوارزمي (doslova „Otec Abdulláha, Mohameda, syn Mojžíšův, pocházející z mě1bsta Chwā01rizm (Chórézm) "; tento kraj se nachází jižně1b od Aralského moře)

- Tento učenec ve svém díle prakticky vytvořil systém arabských číslic a základy algebry (konkrétně metody řešení lineárních a kvadratických rovnic), jejíž název pochází přímo z titulu tohoto díla (Kitūkb al - jabr waā01l - muqūkbala).

- Jeho jméno bylo do latiny převedeno jako algoritsmus, a původně znamenalo „provádění aritmetiky pomocí arabských číslic "; abacisté počítali pomocí abaku, algoristé pomocí algorismů.

- Postupem času se kvůli neznalosti původu slova jeho podoba měnila, záměnou arabského kořene s kořenem řeckého slova arithmos se z algorismu stal algorithmus.(Později bylo v některých jazycích včetně češtiny th změněno na t, v katalánštině se vrátilo s.) Toto slovo se používalo jako označení různých matematických postupů, např.v 18. století označoval latinský termín algorithmus infitesimalis „metodu výpočtů s využitím nekonečně malých veličin, vynalezenou Leibnizem.” Slovo algoritmus v dnešním významu se používá až zhruba od 20. století.

# Vlastnosti algoritmu

* Konečnost (finitnost)
* Obecnost (hromadnost, masovost, univerzálnost)
* Determinovanost
* Výstup (resultativnost)
* Vstup
* Efektivita

# Vlastnosti algoritmu v detailech

## ✳ Konečnost (finitnost)

- ✳ Každý algoritmus musí skončit v konečném počtu kroků. Tento počet kroků může být libovolně velký (podle rozsahu a hodnot vstupních údajů), ale pro každý jednotlivý vstup musí být konečný. Postupy, které tuto podmínku nesplňují, se mohou nazývat výpočetní metody. Speciálním příkladem nekonečné výpočetní metody je reaktivní proces, který průoběžně reaguje s okolním prostředím. Někteří autoři však mezi algoritmy zahrnují i takovéto postupy.

## ✳ Obecnost (hromadnost, masovost, univerzálnost)

- ✳ Algoritmus neřeší jeden konkrétní problém (např. „jak spočítat 3×7"), ale obecnou třídu obdobných problémů (např. „jak spočítat součin dvou celých čísel"), má širokou množinu možných vstupů.

## ✳ Determinovanost

- ✳ Každý krok algoritmu musí být jednoznačně a přesně definován; v každé situaci musí být naprosto zřejmé, co a jak se má provést, jak má provádění algoritmu pokračovat, takže pro stejné vstupy dostaneme pokaždé stejné výsledky. Protože běžný jazyk obvykle neposkytuje naprostou přesnost a jednoznačnost vyjadřování, byly pro zápis algoritmů navrženy programovací jazyky, ve kterých má každý příkaz jasně definovaný význam. Vyjádření výpočetní metody v programovacím jazyce se nazývá program.

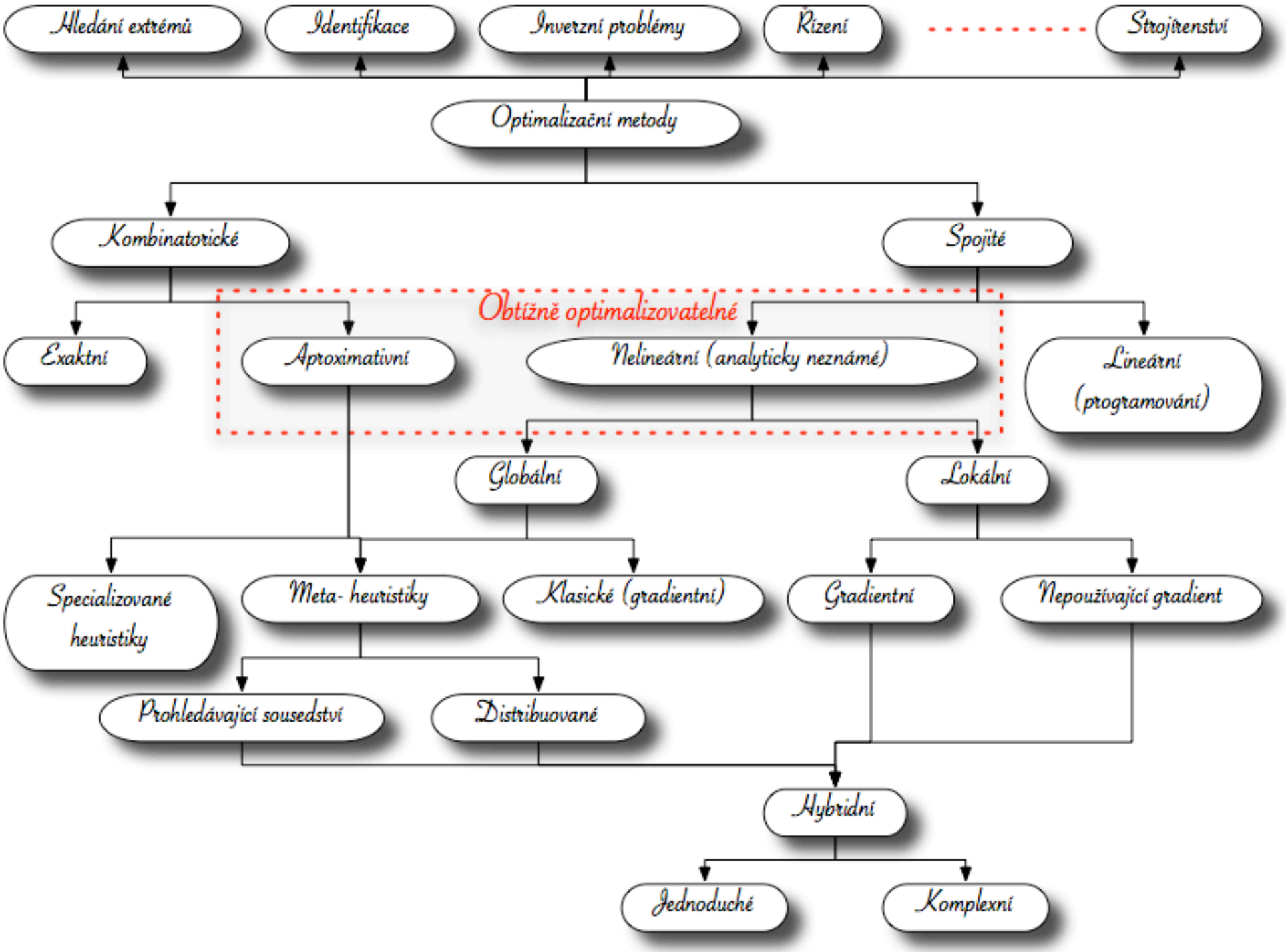
## ✳ Výstup (resultativnost)

- ✳ Algoritmus má alespoň jeden výstup, veličinu, která je v požadovaném vztahu k zadaným vstupům, a tím tvoří odpověď na problém, který algoritmus řeší (algoritmus vede od zpracování hodnot k výstupu).

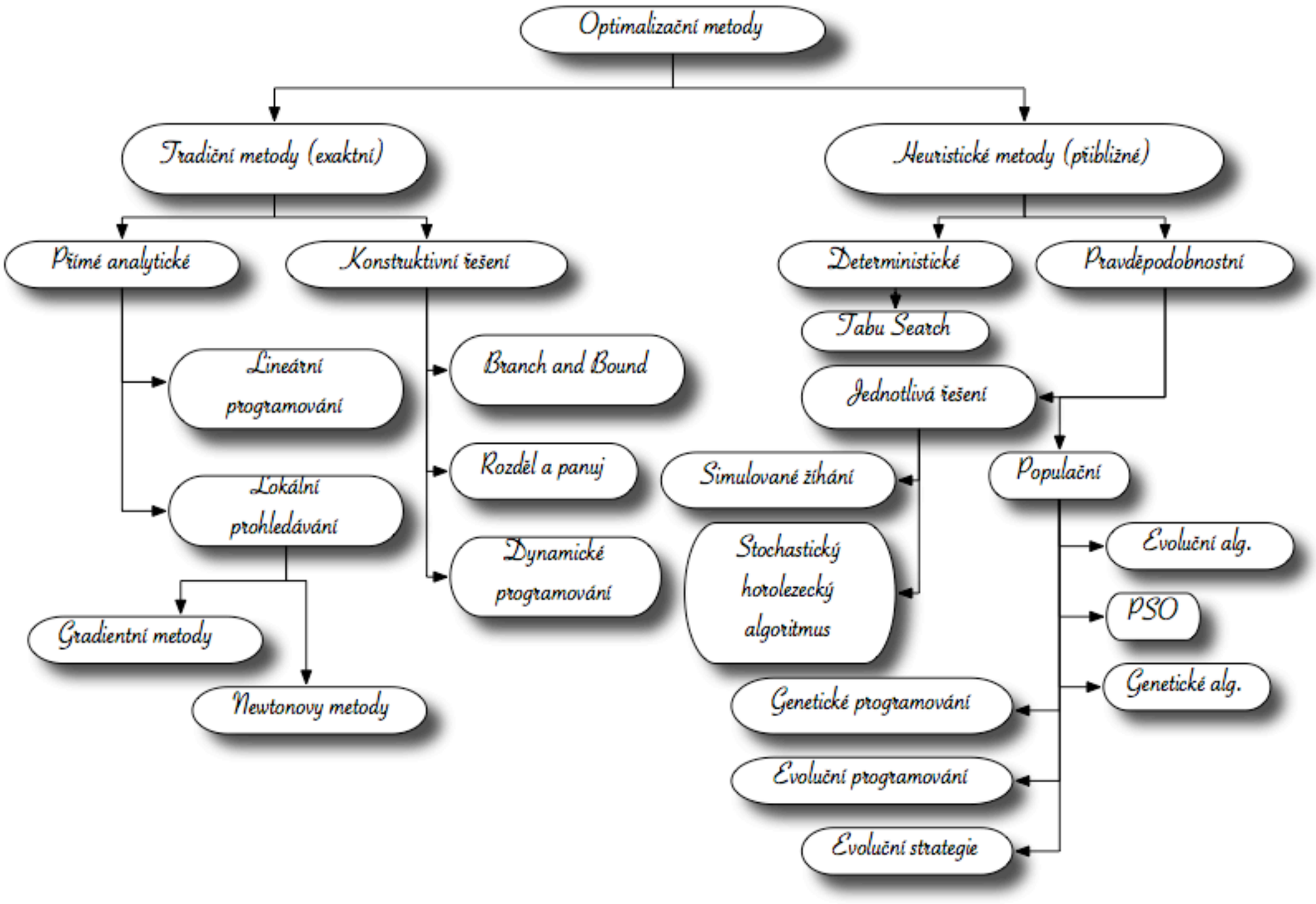
# Členění algoritmů

- Algoritmy lze klasifikovat různými způsoby
- Mezi důležité algoritmy patří:
  - Rekurzivní algoritmy, které využívají (volají) samy sebe
  - Hladové algoritmy, které se k řešení propracovávají po jednotlivých rozhodnutích. Jakmile je učiněno rozhodnutí, není již dále revidováno.
  - Algoritmy typu rozděl a panuj dělí problémy na menší podproblémy, na něž se rekurzivně aplikují (až po triviální podproblémy, které lze vyřešit přímo). Pak se dílčí řešení vhodným způsobem sloučí.
  - Algoritmy dynamického programování
  - Pravděpodobnostní algoritmy
  - Genetické a další evoluční algoritmy (podrobněji v 2. ročníku magisterského stupně)
  - Heuristické algoritmy, které nemají za cíl nalezení přesného řešení, ale pouze vhodného přiblížení.

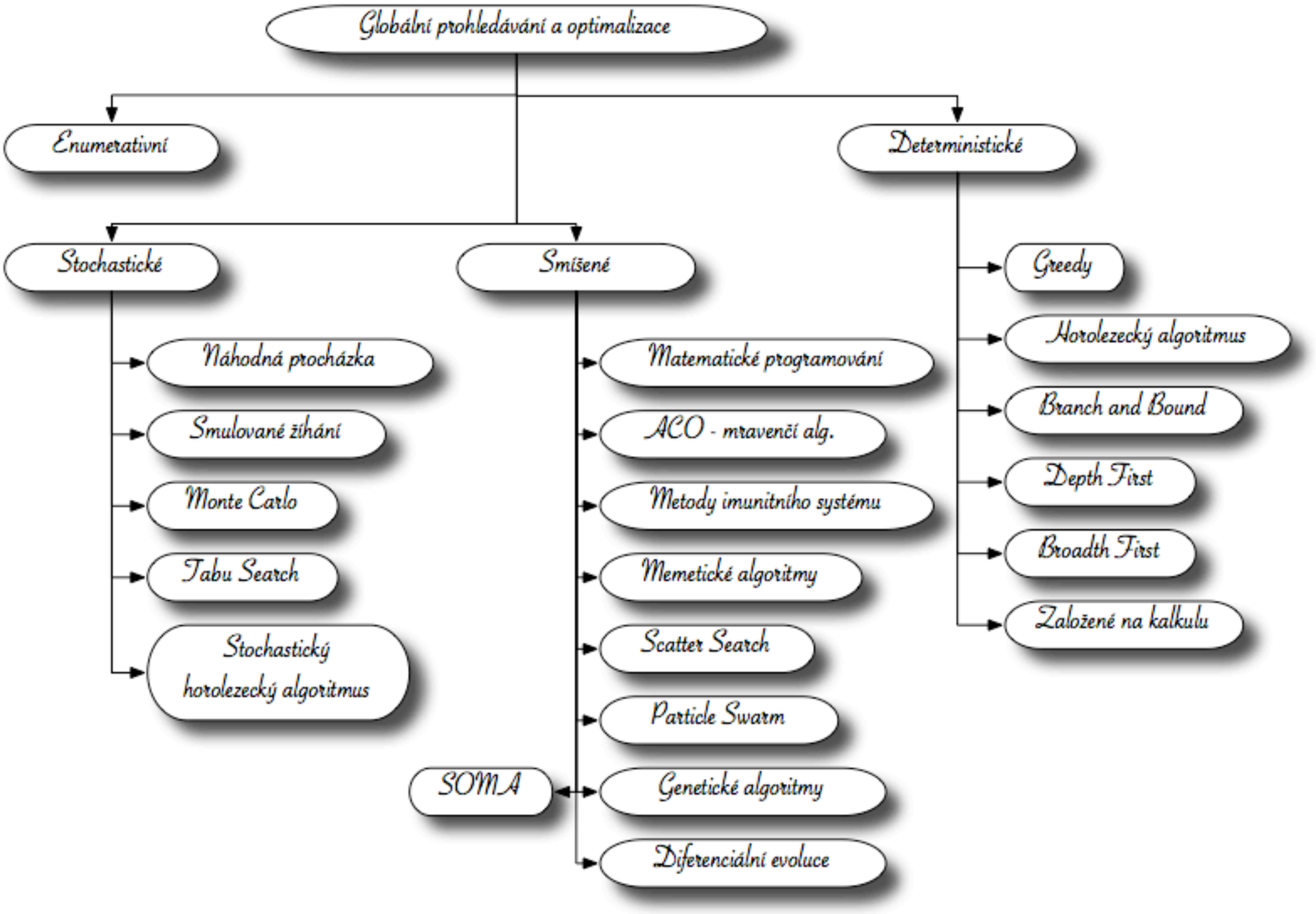
# Členění algoritmů - další způsoby 1



# Členění algoritmů - další způsoby 2



# Členění algoritmů - další způsoby 3



# Rekurzivní algoritmy

- V oblasti matematiky pojem rekurze chápeme jako definování objektu pomocí volání sebe sama.

- Využívá se například pro definici přirozených čísel, stromových struktur a některých funkcí.

- V programování rekurze představuje opakované vnořené volání stejné funkce (podprogramu), v takovém případě se hovoří o rekurzivní funkci. Nedílnou součástí rekurzivní funkce musí být ukončující podmínka určující, kdy se má vnořování zastavit. Jelikož bývá nejčastějším zdrojem chyb, je třeba ji navrhnout dostatečně robustním způsobem a prověřit veškeré možné stavy.

- Pro uplatnění rekurzivních algoritmů je zapotřebí, aby programovací jazyk umožňoval volání podprogramu ještě před ukončením jeho předchozího volání.

- Po každém kroku volání sebe sama musí dojít ke zjednodušení problému. Pokud nenastane koncová situace, provede se rekurzivní krok.

- Každý algoritmus využívající rekurzi lze přepsat do nerekurzivního tvaru při použití zásobníku nebo jiné paměťové struktury.

# Rekurzivní algoritmy - základní dělení

- Rekurzivní chování může být různé v závislosti na tom, kolik podprogramů se jí účastní.
- Rozlišujeme dva základní typy dělení - první dělení:
  - Přímá rekurze nastává, pokud podprogram volá přímo sám sebe.
  - Nepřímá (vzájemná) rekurze je situace, kdy vzájemné volání podprogramů vytvoří „kruh ".Např.v příkazové části funkce A je volána funkce B, ve funkci B voláme funkci C, která volá funkci A.
- Druhé dělení:
  - Lineární rekurze nastává, pokud podprogram při vykonávání svého úkolu volá sama sebe pouze jednou. Vytváří se takto lineární struktura postupně volaných podprogramů.
  - Stromová rekurze nastává, pokud se funkce nebo procedura v rámci jednoho vykonání svého úkolu vyvolá vícekrát.Vzniklou strukturu je možné znázornit jako strom. Pro dvě volání v jednom průchodu vzniká binární strom, pro tři ternární strom, atd. Při kombinaci zmíněných typů rekurze lze docílit velmi komplikovaných struktur. Je třeba připomenout, že již z charakteru takovéhoto volání podprogramů nelze docílit jiné, než symetrické struktury.

# Rekurzivní algoritmy - příklady ze sw Mathematica

```
Nest[f, x, 3]
```

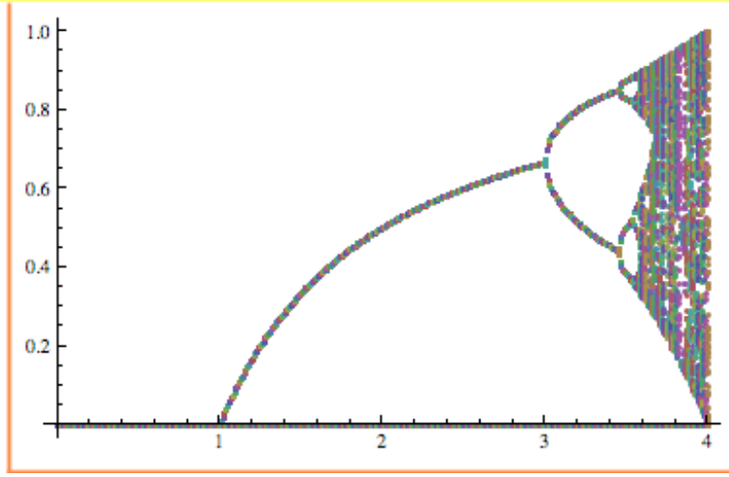
```
f[f[f[x]]]
```

```
Nest[1 / (1 + #) &, x, 5]
```

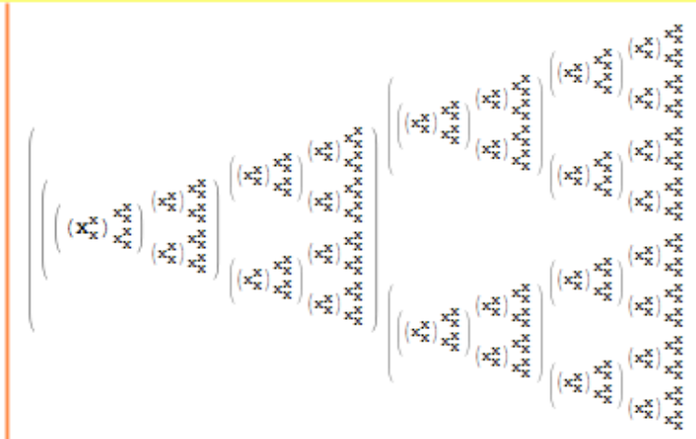
$$\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+x}}}}}$$

* fraktální chování, či chování deterministického chaosu (více ve vyšších ročnících)

```
a = ListPlot[Table[Thread[{r, Nest[r # (1 - #) &, Range[0, 1, 0.01], 1000]}], {r, 0, 4, 0.01}]]
```




```
Nest[Subsuperscript[#, #, #] &, x, 5]
```



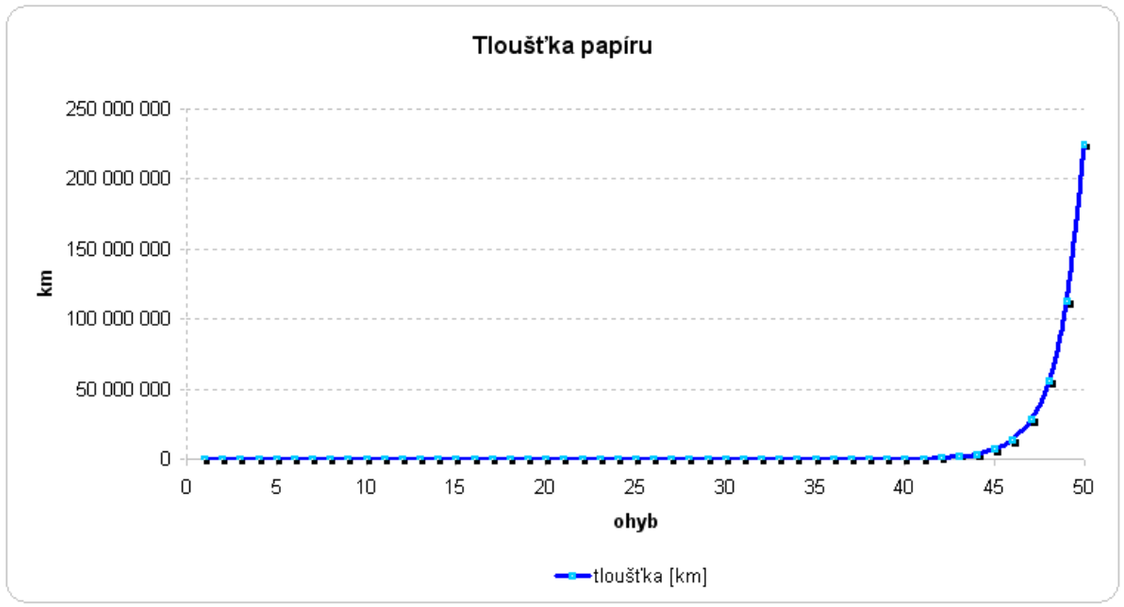
# Rekurzivní algoritmy - zajímavý příklad

| n | tloušťka | tloušťka [mm] | tloušťka [m] | tloušťka [km] |
|---|---|---|---|---|
| 1 | 2 | 0,4 | 0,0004 | 0 |
| 2 | 4 | 0,8 | 0,0008 | 0 |
| 3 | 8 | 1,6 | 0,0016 | 0 |
| 4 | 16 | 3,2 | 0,0032 | 0 |
| 5 | 32 | 6,4 | 0,0064 | 0 |
| 6 | 64 | 12,8 | 0,0128 | 0 |
| 7 | 128 | 25,6 | 0,0256 | 0 |
| 8 | 256 | 51,2 | 0,0512 | 0 |
| 9 | 512 | 102,4 | 0,1024 | 0 |
| 10 | 1024 | 204,8 | 0,2048 | 0 |
| 11 | 2048 | 409,6 | 0,4096 | 0 |
| 12 | 4096 | 819,2 | 0,8192 | 0 |
| 13 | 8192 | 1638,4 | 1,6384 | 0 |
| 14 | 16384 | 3276,8 | 3,2768 | 0 |
| 15 | 32768 | 6553,6 | 6,5536 | 0 |
| 16 | 65536 | 13107,2 | 13,1072 | 0 |
| 17 | 131072 | 26214,4 | 26,2144 | 0 |
| 18 | 262144 | 52428,8 | 52,4288 | 0 |
| 19 | 524288 | 104857,6 | 104,8576 | 0 |
| 20 | 1048576 | 209715,2 | 209,7152 | 0 |
| 21 | 2097152 | 419430,4 | 419,4304 | 0 |
| 22 | 4194304 | 838860,8 | 838,8608 | 1 |
| 23 | 8388608 | 1677721,6 | 1677,7216 | 2 |
| 24 | 16777216 | 3355443,2 | 3355,4432 | 3 |
| 25 | 33554432 | 6710886,4 | 6710,8864 | 7 |
| 26 | 67108864 | 13421772,8 | 13421,7728 | 13 |
| 27 | 134217728 | 26843545,6 | 26843,5456 | 27 |
| 28 | 268435456 | 53687091,2 | 53687,0912 | 54 |
| 29 | 536870912 | 107374182,4 | 107374,1824 | 107 |
| 30 | 1073741824 | 214748364,8 | 214748,3648 | 215 |
| 31 | 2147483648 | 429496729,6 | 429496,7296 | 429 |
| 32 | 4294967296 | 858993459,2 | 858993,4592 | 859 |
| 33 | 8589934592 | 1717986918 | 1717986,918 | 1 718 |
| 34 | 17179869184 | 3435973837 | 3435973,837 | 3 436 |
| 35 | 34359738368 | 6871947674 | 6871947,674 | 6 872 |
| 36 | 68719476736 | 13743895347 | 13743895,35 | 13 744 |
| 37 | 1,37439E+11 | 27487790694 | 27487790,69 | 27 488 |
| 38 | 2,74878E+11 | 54975581389 | 54975581,39 | 54 976 |
| 39 | 5,49756E+11 | 1,09951E+11 | 109951162,8 | 109 951 |
| 40 | 1,09951E+12 | 2,19902E+11 | 219902325,6 | 219 902 |
| 41 | 2,19902E+12 | 4,39805E+11 | 439804651,1 | 439 805 |
| 42 | 4,39805E+12 | 8,79609E+11 | 879609302,2 | 879 609 |
| 43 | 8,79609E+12 | 1,75922E+12 | 1759218604 | 1 759 219 |
| 44 | 1,75922E+13 | 3,51844E+12 | 3518437209 | 3 518 437 |
| 45 | 3,51844E+13 | 7,03687E+12 | 7036874418 | 7 036 874 |
| 46 | 7,03687E+13 | 1,40737E+13 | 14073748836 | 14 073 749 |
| 47 | 1,40737E+14 | 2,81475E+13 | 28147497671 | 28 147 498 |
| 48 | 2,81475E+14 | 5,6295E+13 | 56294995342 | 56 294 995 |
| 49 | 5,6295E+14 | 1,1259E+14 | 1,1259E+11 | 112 589 991 |
| 50 | 1,1259E+15 | 2,2518E+14 | 2,2518E+11 | 225 179 981 |
| 51 | 2,2518E+15 | 4,5036E+14 | 4,5036E+11 | 450 359 963 |
| 52 | 4,5036E+15 | 9,0072E+14 | 9,0072E+11 | 900 719 925 |
| 53 | 9,0072E+15 | 1,80144E+15 | 1,80144E+12 | 1 801 439 851 |
| 54 | 1,80144E+16 | 3,60288E+15 | 3,60288E+12 | 3 602 879 702 |
| 55 | 3,60288E+16 | 7,20576E+15 | 7,20576E+12 | 7 205 759 404 |
| 56 | 7,20576E+16 | 1,44115E+16 | 1,44115E+13 | 14 411 518 808 |
| 57 | 1,44115E+17 | 2,8823E+16 | 2,8823E+13 | 28 823 037 615 |
| 58 | 2,8823E+17 | 5,76461E+16 | 5,76461E+13 | 57 646 075 230 |
| 59 | 5,76461E+17 | 1,15292E+17 | 1,15292E+14 | 115 292 150 461 |
| 60 | 1,15292E+18 | 2,30584E+17 | 2,30584E+14 | 230 584 300 921 |
| 61 | 2,30584E+18 | 4,61169E+17 | 4,61169E+14 | 461 168 601 843 |

**Tloušťka novinového papíru je cca 0,2mm**

**Představ si obří novinový papír (neber v potaz kolikrát lze přehnout) tloušťka novinového papíru je 0,2 mm .... odhadem si tipni, jak velká by byla harmonika po 50-ti přeloženích na půl (tzn. papír na půl, pak zase atd. 50x) ?**

**řešení vpravo a graf :)**




Vzdálenost Země - Měsíc (povrch) = 363 104 km
Vzdálenost Země - Slunce (povrch) = 1 AU = 150 milionů kilometrů

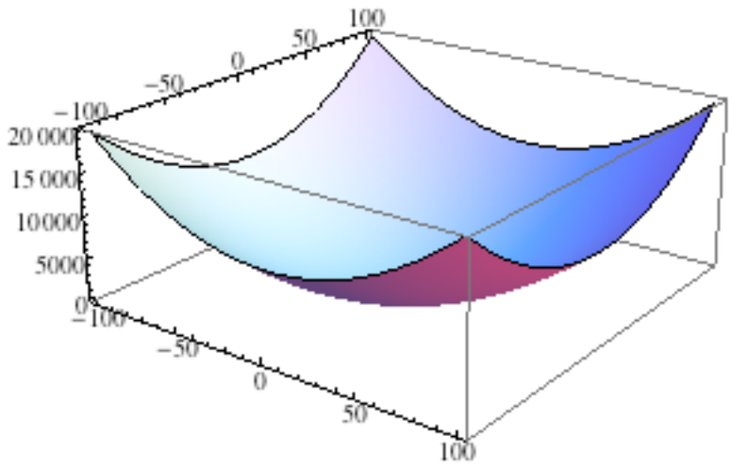
abychom dosáhli papírem ke slunci, stačilo by 60 přehnutí :) (230 mil.km)

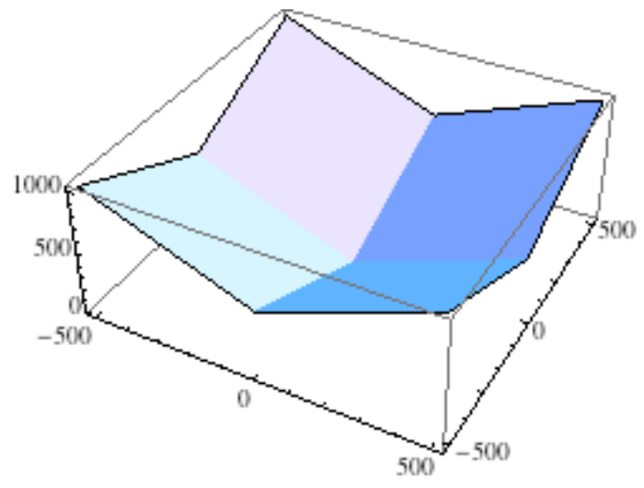
✳ překládání papíru

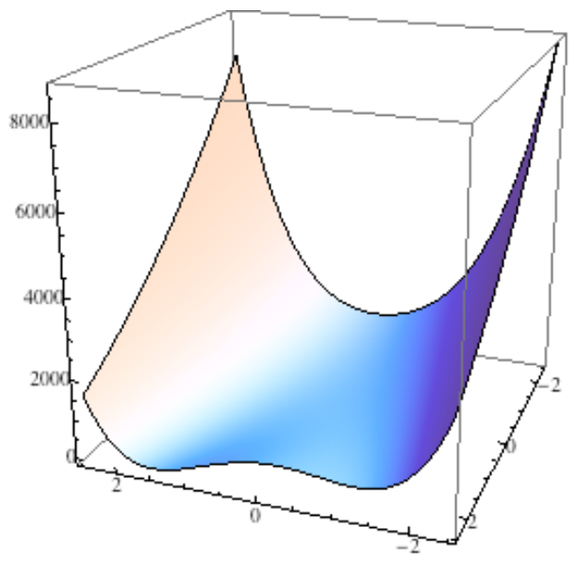
# Optimalizace - účelová funkce 1

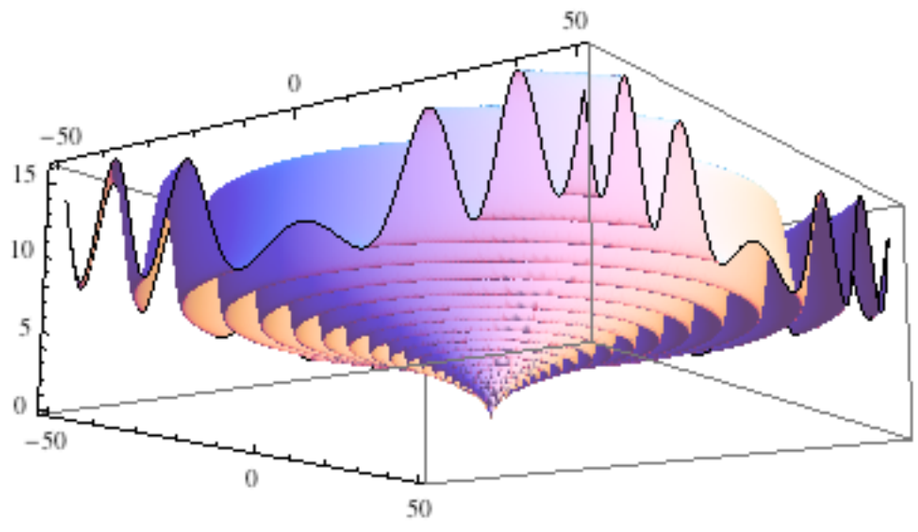
* Účelová funkce (synonyma: hodnotící funkce, ohodnocení funkce, cost function, někdy též fitness)
    * Účelová funkce je matematický model řešeného optimalizačního problému. Řešení je obvykle dosaženo minimalizací resp maximalizací funkce. Tvar funkce je různý a obvykle je formulován matematikou spadající do příslušné problematiky.
    * Hodnota účelové funkce vyjadřuje kvalitu řešení.
    * Účelové funkce jsou unimodální (s jedním globálním extrémem) a multimodální (s více extrémy - lokální a globální). Na unimodální stačí deterministické metody, na multimodální jsou vhodné heuristické algoritmy.
* Fitness (někdy se může zaměnit s účelovou funkcí)
    * Fitness je konverze účelové funkce do intervalu <0,1>, využívá se především u genetického algoritmu, kde se využívá binární reprezentace jedinců.

# Optimalizace - účelová funkce 2



# Optimalizace - účelová funkce 3

- Definice účelové funkce je jedním z nejkritičtějších kroků v rámci optimalizačního procesu.
- Jeho správné provedení může citlivě ovlivnit kvalitu výsledků.
- Vzhledem k tomu, že třída problémů, jež lze definovat jako optimalizační, je vskutku nepřeberná, neexistuje kuchařka konkrétních postupů, jak účelovou funkci sestavit. Dají se pouze naznačit obecné principy.
- Při tvorbě účelové funkce je nutné vědět čeho se má dosáhnout a z čeho lze vycházet.

# Hladové algoritmy

- Hladový algoritmus (anglicky “greedy search”) je jedním z možných způsobů řešení optimalizačních úloh v matematice a informatice.
- V každém svém kroku vybírá lokální minimum, přičemž existuje šance, že takto nalezne minimum globální.
- Hladový algoritmus se uplatní v případě, kdy je třeba z množiny určitých objektů vybrat takovou podmnožinu, která splňuje jistou předem danou vlastnost a navíc má minimální (případně maximální) ohodnocení.
- Ohodnocení je obvykle reálné číslo w, přiřazené každému objektu dané množiny, ohodnocení množiny A je definováno jako:

$$w(A) = \sum_{a \in A} w(a)$$

# Hladové algoritmy - příklady použití

- Hladové algoritmy se uplatňují například v následujících úlohách :
  - Hledání minimální kostry grafu: Kruskalův algoritmus, Jarníkův algoritmus a Borůvkův algoritmus
  - Problém obchodního cestujícího: Je dáno $n$ měst a je nutno nalézt optimální permutaci (trasu) např z pohledu vzdálenosti. Je možno i uvažovat více faktorů, např: čas, ekologii trasy atd...
  - Problém batohu: máme dáno n předmětů. Pro každý předmět $i = 1,..., n$ máme dánu hmotnost $W[i]$ a cenu $P[i]$. Je dána též maximální kapacita $C$. Úkolem je najít takovou podmnožinu množiny úkolů, pro niž platí že kapacita není překročena a zároveň je celková cena batohu co největší ($x$ je vektor; je - li $x[i] = 1$, pak i - tý předmět do dané podmnožiny patří, je - li $x[i] = 0$, pak do ní nepatří). Pro řešení této úlohy pomocí hladového algoritmu stačí setřídit předměty podle rostoucího poměru cena/hmotnost, podmínka na množinu je, že součet hmotností předmětů musí být menší nebo roven $C$.

# Hladové algoritmy - vzorový algoritmus (teoretický)

- 1. všechny prvky původní množiny setřídíme do posloupnosti podle rostoucí nebo klesající váhy podle toho, zda chceme výsledek minimalizovat nebo maximalizovat
- 2. položíme $A_0 = \varnothing$
- 3. postupně procházíme posloupnost a vytváříme množiny Ai
  - - splňuje - li množina $A_{i-1} \cup \{i\}$ danou podmínku, položíme $A_i = A_{i-1} \cup \{i\}$
  - - jinak $A_i = A_{i-1}$
- projdeme - li takto celou původní množinu, obsahuje množina An prvky, splňující danou vlastnost, a to takové, že součet jejich ohodnocení je minimální (maximální)

# Algoritmy typu rozděl a panuj

- Metoda rozděl a panuj (angl.divide and conquer) označuje ty algoritmy pro práci s daty, které řeší problém rozdělením řešené úlohy na dílčí části (podproblémy), nad kterými se provádí algoritmická operace.

- Často se tato metoda implementuje rekurzivně nebo iterativně a původní úloha se dělí na stále menší části. Pak se dílčí řešení vhodným způsobem sloučí.

- Typickými představiteli metody rozděl a panuj jsou algoritmy třídění QuickSort, výpočet rychlé Fourierovy transformace (FFT) nebo binární vyhledávání.

# Algoritmy dynamického programování

- Algoritmy dynamického programování pracují tak, že postupně řeší části problému od nejjednodušších po složitější s tím, že využívají výsledky již vyřešených jednodušších podproblémů.
- Mnoho úloh se řeší převedením na grafovou úlohu a aplikací příslušného grafového algoritmu.

# Pravděpodobnostní algoritmy 1

- Pravděpodobnostní algoritmy (někdy též probabilistické) provádějí některá rozhodnutí náhodně či pseudonáhodně.
- V případě, že máme k dispozici více počítačů, můžeme úlohu mezi ně rozdělit, což nám umožní ji vyřešit rychleji; tomuto cíli se věnují paralelní algoritmy
- Pravděpodobnostní (náhodnostní) algoritmy jsou nedeterministické algoritmy, které se snaží najít řešení rychleji nebo řešení těžko řešitelných problémů, často tzv.NP - úplných problémů.

# Pravděpodobnostní algoritmy 2

- Pravděpodobnostní algoritmus se může náhodně rozhodovat mezi různými možnostmi jak pokračovat.

- Pro stejný vstup může dávat takový algoritmus různé výsledky, které mohou být dokonce nesprávné. Mnohdy se tedy na daném vstupu spustí pravděpodobnostní algoritmus vícekrát, aby se s větší pravděpodobností dospělo ke správnému výsledku.

- Varianty pravděpodobnostních algoritmů:
  - Výpočetní strom je binární, v každém uzlu se provede hod mincí.
  - V každém výpočetním uzlu je definováno pravděpodobnostní rozložení na hranách.
  - Na začátku se vybere náhodně deterministický algoritmus, který provede výpočet.

- Všechny tři výše uvedené varianty (strategie) jsou ekvivalentní.

# Heuristické algoritmy 1

- Heuristický algoritmus si za cíl neklade nalézt přesné řešení, ale pouze nějaké vhodné přiblížení.

- Používá se v situacích, kdy dostupné zdroje (např.čas) nepostačují na využití exaktních algoritmů (nebo pokud nejsou žádné vhodné exaktní algoritmy vůbec známy).

- Přitom jeden algoritmus může patřit zároveň do více skupin; například může být zároveň rekurzivní a typu rozděl a panuj.

# Heuristické algoritmy 2

- Informatika jako věda o informacích a jejich zpracování chápe heuristiku jako postup získání řešení problému, které však není přesné a nemusí být nalezeno v krátkém čase.
- Slouží však nejčastěji jako metoda rychle poskytující dostatečné a dosti přesné řešení, které však nelze obecně dokázat. Nejčastější použití heuristického algoritmu nalezneme v případech, kde není možné použít jiného lepšího algoritmu, poskytujícího přesné řešení s obecným důkazem.

# Heuristické algoritmy 3

- Jiný, avšak podobný pohled na heuristiku
- Heuristikou nazýváme postup, který, i když neprobere všechny možnosti, je schopný v některých případech podat výsledek.
- Výsledek je podáván zpravidla jedním ze dvou možných stavů. Buď jako kladný výsledek (odpověď) nebo jako výrok neurčitosti.

# Heuristické algoritmy 4 - použití algoritmu

- Heuristické algoritmy se začaly používat s příchodem výpočetní techniky.
- Tyto algoritmy jsou rozvíjeny a často používány pro řešení složitých problémů. Jsou vhodné zejména pro řešení složitých funkcí s mnoha parametry a složitým průběhem s mnoha extrémy.
- Například algoritmy deterministické jsou v řešení složitých problémů velmi špatně použitelné, protože jejich náročnost (zejména časová) roste, s lineárně rostoucím rozsahem problému, exponenciálně. Naopak výhodou je oproti heuristickým algoritmům nalezení přesného, optimálního řešení, které lze dokázat.
- Velmi často se setkáváme s metodami kombinujícími heuristické algoritmy s deterministickými.

# Heuristické algoritmy 5 - metody

- Genetický algoritmus - algoritmus založený na principu přirozené selekce. S úspěchem se využívá jako počítačový heuristický algoritmus schopný nalézt kvalitní řešení i složitého problému. V oblasti IT ho využijeme pro řešení mnoha technických a matematických problémů.

- Metoda lokálního hledání - jedná se o jednu z nejjednodušších metod. Dobře použitelná pro řešení matematických problémů.

- Iterativní metoda - využívá postupného hledání řešení ve stále se zúžující oblasti řešení (postupně se z dobrého řešení dopracovává k ještě lepšímu řešení).

- Hladový algoritmus

- Metod je celá řada a každá má své klady a zápory. Mezi klady řadíme možnost nalezení přesného řešení. Mezi zápory dobu, která je k provedení algoritmu potřeba, případně množství upřesňujících parametrů.Větší jistotu, že nalezený výsledek je přesný, můžeme u některých metod zvýšit opakováním heuristického algoritmu (Ze statistického hlediska min. 30x).

# Heuristické algoritmy 6 - použití

- Použití heuristických algoritmů je velmi dobře použitelné při řešení již známého "Problému obchodního cestujícího", který má nejkratší cestou projít města vyznačená na mapě.

- Běžně používané algoritmy využívající heuristiku, jejichž služeb využíváme každodenně, jsou například algoritmy pro heuristickou analýzu integrované v antivirových software, satelitní navigace, plánovače tras, výroby...atd.

- Dále se heuristické algoritmy využívají pro odhalení e - mailového spamu, ve specializovaných vývojových a simulačních softwarech určeném pro různé oblasti použití (od strojírenství, přes oblast IT až po lékařství ...).