Algoritmizace - Rekurze, Brute Force, Heuristiky, Nedeterministické algoritmy

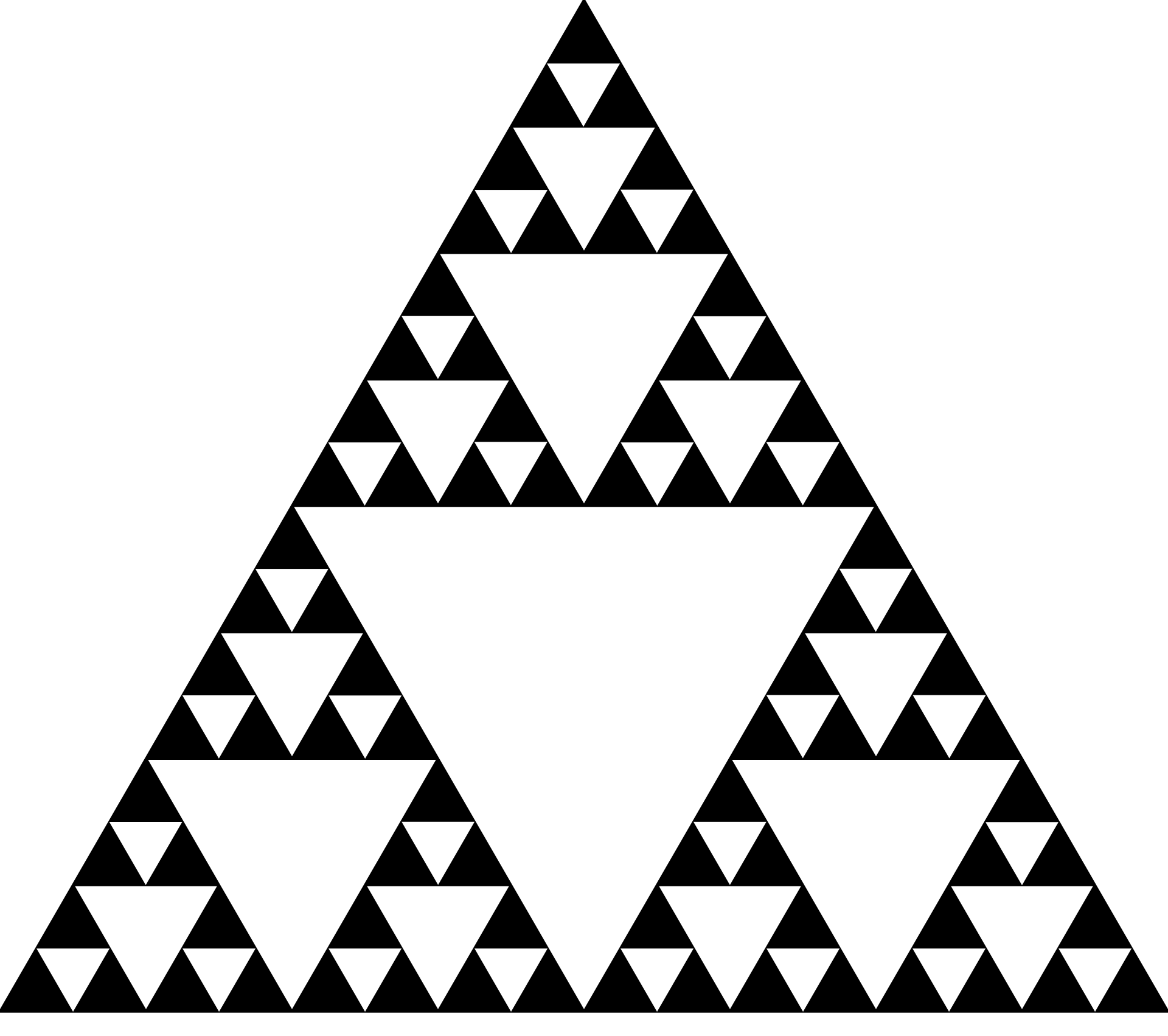
*předmluva: Nejvíc se dá říct o rekurzi a Brute Force. Brute Force doporučuji téměř citovat, zatímco rekurzi brát ze začátku hodně teoreticky, pak až přejít na rekurzi v programování. Heuristika, o které jsem již mluvil v souvislosti se stavovým prostorem se dá opět prezentovat na principu šachisty, ale nenechte se příliš unést. O nedeterministických algoritmech se nedá říct příliš, je to taková třešnička na dortu.*

*Andreas Dumalas*

## Rekurze

Rekurze v praxi

Co je to rekurze? V pravém slova smyslu je to objekt, který je součástí sebe sama. To je pravděpodobně těžké si představit, ale v praxi to většinou znamená, že objekt bude obsahovat svou zmenšenou kopii, nebo bude pro definování určitého pojmu zapotřebí pojem samotný. Krásným příkladem je tento trojhran zvaný Sierpinského trojúhelník.



Rekurze v programování

Jedná se o speciální techniku programování, při které je určitá procedura nebo funkce volána znovu, než je dokončeno její předchozí volání. To může u některých úloh vést k rychlému a efektivnímu řešení, které ale nemusí nutně být optimální. Naopak, pokud se snažíme program optimalizovat z hlediska paměti, rekurzi se snažíme omezit, či kompletně eliminovat. Některé jazyky používají místo cyklů rekurzi (Lisp, Prolog).

Typy rekurze

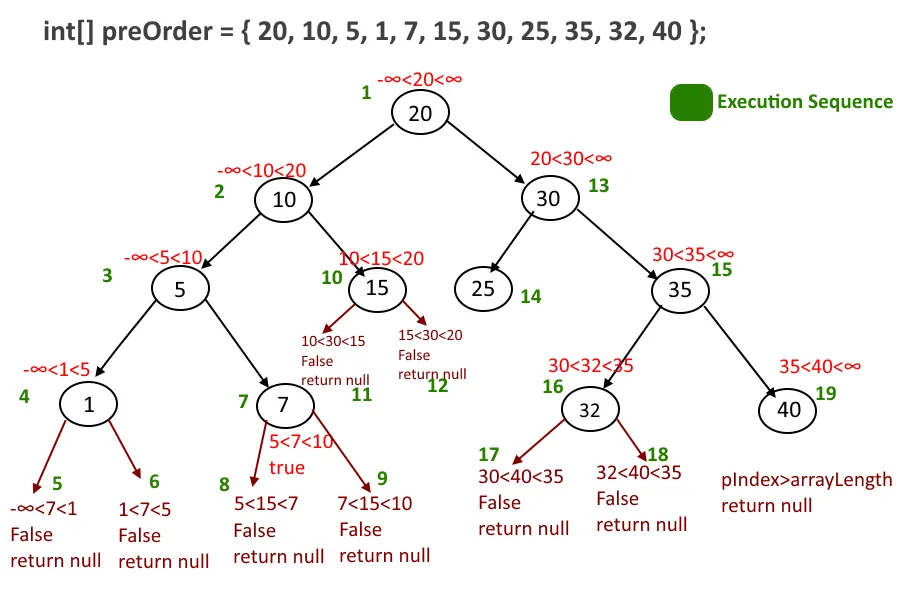
Rekurzi dělíme na dva základní typy na základě toho, kolik podprogramů se jí účastní.

1. Rekurze může probíhat přímo nebo nepřímo:

* Přímá rekurze nastane právě tehdy, když rekurze volá sama sebe
* Nepřímá rekurze nastává v momentě, kdy vzájemné volání podprogramů tvoří kruh. Příkladem je situace, kdy funkce A volá funkci B, která opět volá A.

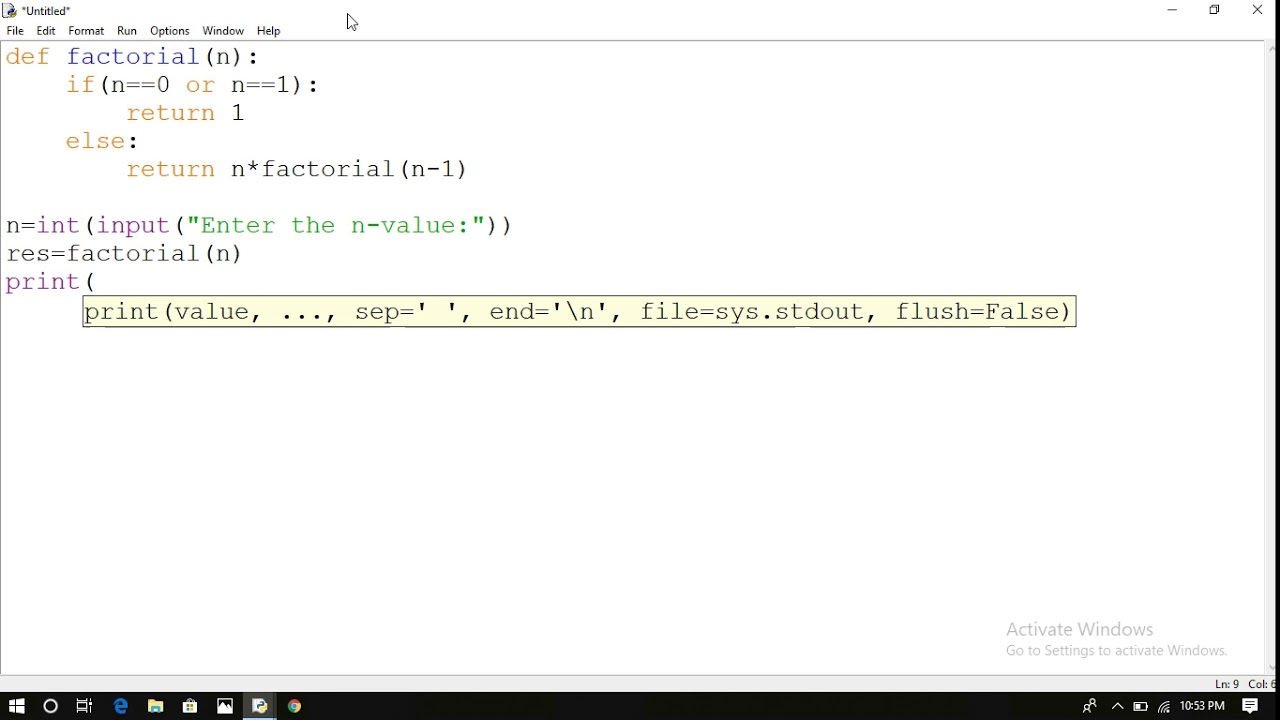
1. Podprogram může být volán jednou, nebo vícekrát

* Lineární rekurze nastane právě tehdy, když podprogram při vykonávání svého úkolu volá sama sebe jen jednou.
* Stromová rekurze nastane tehdy, když se funkce v rámci jednoho vykonávání úkolu zavolá vícekrát. Strukturu volání lze znázornit jako binární strom (obrázek)



Rekurzivní volání je obvykle vykonáno následujícími kroky:

* Kontrola, zdali jsou vstupní parametry v souladu se stanovenými podmínkami
* Rozdělení problému na podproblémy
* Volání funkcí, které řeší daný podproblém (přímé/nepřímé volání)
* Sestavení výsledku
* Vrácení výsledku



## Brute Force

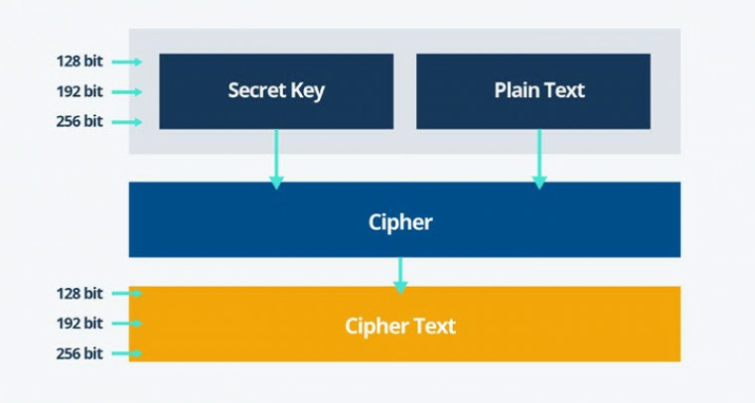
Algoritmus Brute Force, česky zvaný útok hrubou silou je velmi rozsáhle používaný postup, který obvykle slouží k prolomení hesla, či celých přihlašovacích údajů. Tento proces je automatizovaný a vzhledem k tomu, že uživatelé často volí velmi jednoduchá hesla, bývá ve velkém procentu případů úspěšný.

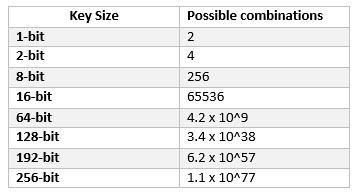
Čas potřebný k prolomení hesla roste exponenciálně s délkou klíče (délka klíče se uvádí v bitech), neboť se tím zvětšuje prostor klíče. Velký prostor klíčů je tak nutnou podmínkou pro bezpečnost šifry. (V kryptografii je klíč informace, která určuje průběh algoritmu. Při šifrování klíč určuje transformaci zprávy do šifrovaného textu, při dešifrování je tomu naopak.)

Z historického hlediska byla délka klíčů na základě předpisů USA definována na maximálně 56 bitů. To se ale ukázalo jako nevhodné, prostor klíčů byl příliš náchylný na prolomení. Dnes se používají klíče o délce 128, či 256 bitů.

Pro dešifrování je optimální použít hardware, jenž je schopen paralelního zpracování dat. Tím by z hlediska dostupnosti mohla být Grafická karta, nebo z hlediska energetické úspory programovatelné hradlové pole (integrovaný logický obvod, který je naprogramován až u zákazníka/uživatele).

K prolomení symetrického (používá k šifrování i dešifrování jeden klíč) klíče o délce 256 bitů je zapotřebí 2128 krát vyšší výkon, než k prolomení 128 bitového klíče. Za předpokladu, že by jsme disponovali strojem se schopností ověřit trilion (1018) klíčů za sekundu (což je mnoho násobek výkonu nejvýkonnějších superpočítačů, stále by dešifrování trvalo 3x1051 let.

****



## Heuristiky

Heuristické algoritmy jsou takové algoritmy, které používají ke svému výpočtu heuristiku. Heuristika je v podstatě zkusmé řešení problému (pomocí odhadu budoucích událostí) vhodné tehdy, pokud neznáme přesný postup, jak dojít k cíli. Toto řešení nemusí být příliš přesné, ani nijak zvlášť rychlé. Dobrým příkladem heuristické metody je ta, která se stará o rozhodnutí vašeho oponenta ve hře šachy proti umělé inteligenci.

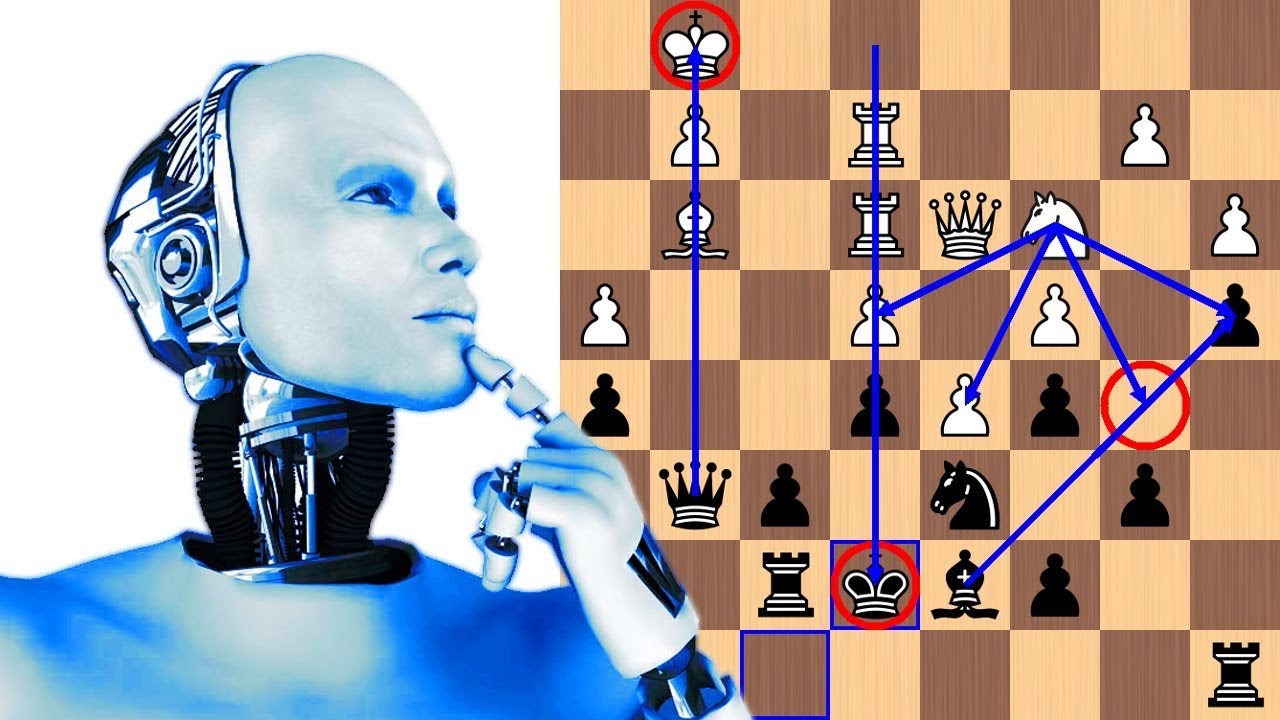
Jeho užitečnost ale tkví v tom, že výsledky jsou dostatečně přesné a dostatečně rychle získány i v situaci, kdy byla metoda pro přímý výpočet neúměrně složitá. Z toho důvodu by bylo možné umělou inteligenci porazit, ale jen pomocí druhé inteligence s lepší heuristickou funkcí, nebo připraveným programem pro veškeré situace, které by ve hře mohly nastat.

Heuristický algoritmus je obvykle označení pro algoritmy, které neposkytují záruku kvality řešení, nebo pokud nevíme, jestli heuristika uspěje.

Je potřeba si uvědomit, že heuristika nemusí uspět. Nemusí vydat správné řešení, optimální řešení, také nemusí podat řešení vůbec (viz. šachy). Lze kombinovat více heuristických metod najednou.

Nejčastější metody Heuristiky:

* **Generický algoritmus** - algoritmus založený na principu přirozeného výběru. Na základě předem daných kritérií rozhoduje, jakou hodnotu upřednostní. Dokáže najít kvalitní řešení i složitého problému, v oblasti IT je velmi rozsáhle používané.
* **Metoda lokálního hledání** - tyto metody vyhodnocují jen své nejbližší okolí a vydají se při prohledávání zkrátka některým směrem, který se v tu chvíli zdá metodě lokálně optimální na základě vyhodnocení funkce. Lokální metody ale zcela zapomínají předcházející uzly a postrádají tak možnost návratu.
* **Iterativní metoda** - využívá postupného hledání řešení ve stále se zužující oblasti řešení (postupně se z dobrého řešení dopracovává k ještě lepšímu řešení).



## 

## Nedeterministické algoritmy

+, někdy také nazývaný stochastický, je algoritmus takový, který v určitých krocích volí z několika možností. Jeho opakem je již podle názvu Deterministický algoritmus, pro který je další krok vždy definován jednoznačně. Při stejném vstupu může nedeterministický algoritmus dojít k různým výsledkům.

Lze zkoumat množinu všech výsledků nedeterministického algoritmu a určovat

* zda existuje alespoň jeden výsledek vyhovující zadání
* Pravděpodobnost provedení některých kroků algoritmu, pokud jsou známy pravděpodobnosti výběru dalších kroků algoritmu

