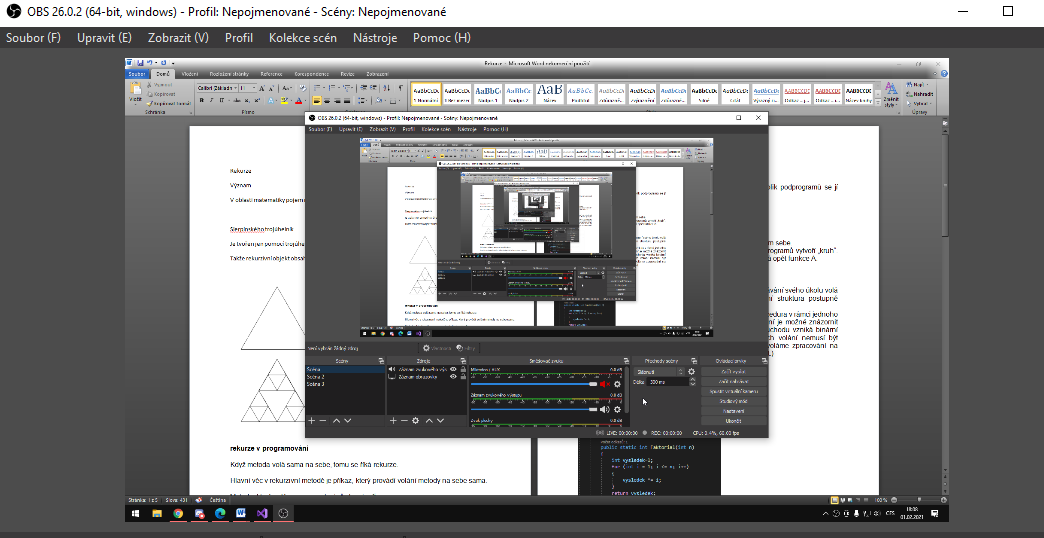
**Rekurze**

Význam

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

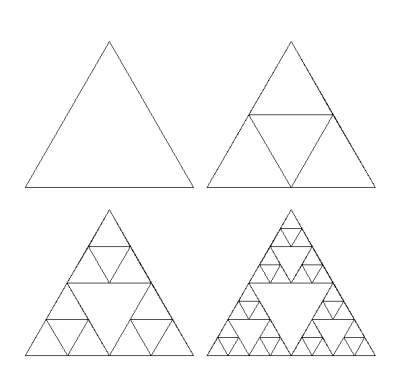


Taky rekurze

Sierpinského trojúhelník

Je tvořen jen pomocí trojúhelníku

Takže rekurzivní objekt obsahuje svoji změnšenou kopii.



**rekurze v programování**

Když metoda volá sama na sebe, tomu se říká rekurze.

Hlavní věc v rekurzivní metodě je příkaz, který provádí volání metody na sebe sama.

Metoda, která volá sama na sebe je “rekurzivní“.

Většinu případů rekurzí lze převést na nerekurzivní algoritmus (a naopak )

Rekurze musí obsahovat podmínku ukončení (konečnost algoritmu), jinak by byla nekonečná

To mže být způsob zefektivnění rychlosti nějakých programů

Rekurze se snaží paměť omezit.

Rekurze je spojená s vysokými nároky na využití paměti zásobníku

Příklady

* strom (z větve jdou větve, co je ještě větev? …)
* většina her - pravidla v každém tahu jsou stejná, princip řešení je tedy stejný i když se data mění.

**K čemu je rekurze dobrá?**

Rekurze se dá jednoduše použít při počítání faktoriálu.

Rekurzivní chování může být různé v závislosti na tom, kolik podprogramů se jí účastní.

Metoda f1 je volána zatímco jedno z předchozích volání f1 ještě nebylo ukončeno návratem.

Rozlišujeme dva základní typy dělení.

Volání může probíhat přímo nebo nepřímo:

1. Přímá rekurze nastává, když podprogram volá přímo sám sebe. = volá se přímo
2. Nepřímá rekurze je situace, kdy vzájemné volání podprogramů vytvoří „kruh“. Např. ve funkci A se volá funkce B a ve funkci B se volá opět funkce A. = volá se zprostředkovaně přes jinou funkci

Podprogram může být volán jednou nebo vícekrát:

1. Lineární rekurze nastává, pokud podprogram při vykonávání svého úkolu volá sama sebe pouze jednou. Vytváří se takto lineární struktura postupně volaných podprogramů.

Třeba faktoriál, nebo hledání hodnoty v setříděném poli

1. Stromová rekurze nastává, pokud se funkce nebo procedura v rámci jednoho vykonání svého úkolu vyvolá vícekrát. Strukturu volání je možné znázornit jako zakořeněný strom. Pro dvě volání v jednom průchodu vzniká binární strom, pro tři ternární strom, atd. (Počet rekurzivních volání nemusí být konstantní, např. při rekurzivním procházení grafu voláme zpracování na všechny sousedy vrcholu, a těch je obecně různý počet.)

Třeba třídicí algoritmy

Druhy

Tail recursion – rekurzní volání je posledním příkazem -> return f()

Strukturální rekurze - volání funkce přes jinou funkci, nebo v jiném místě, než samostatně na konci

Backtracking - zpětné vyčíslování rekurze (obecně funkce)

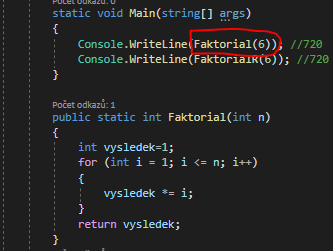
Mutual (indirect) recursion - dva objekty, závisí na sobě

* funkce volající se navzájem
* objekty ukazující na sebe navzájem
* např. binární strom se skládá z uzlu a dvou stromů

Corecursion - duální k rekurzi (iterace), od jednoduchého ke složitému.

Asi nejlepší příklad - faktoriál

Bez rekurze



Cyklus, který přezásobuje do výsledku čísla od 1 do 6, int výsledek a i musí být 1, když budou 0, tak výsledek bude 0

Cyklus:

1 \* 1 = 1

1 \* 2 = 2

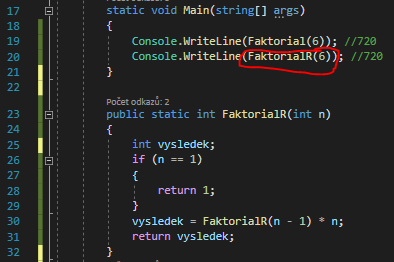
2 \* 3 = 6

6 \* 4 = 24

24 \* 5 = 120

120 \* 6 = 720 konec

S rekurzí – přímá a lineární



Řádek 30 začátek

N = 1, return 1 z IF

N = 2, výsledek = 1 \* 2, return 2

N = 3, výsledek = 2 \* 3, return 6

N = 4, výsledek = 6 \* 4, return 24

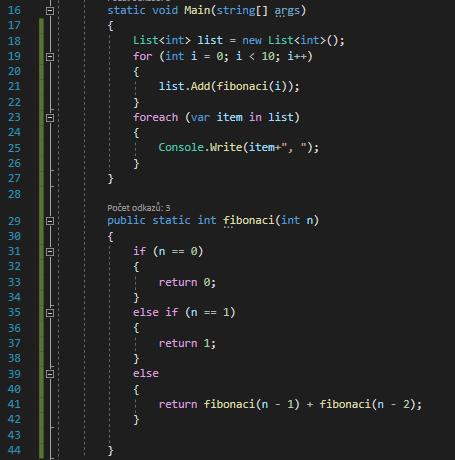
N = 5, výsledek = 24 \* 5, return 120

N = 6, výsledek = 120 \* 6, return 720 konec řádku 30

výpis, protože se konečně executenul řádek 31

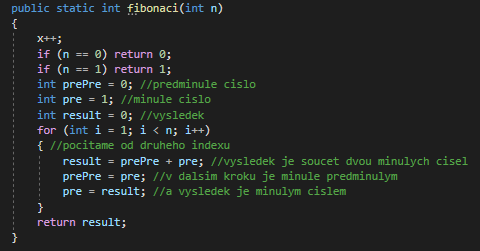
Snazší něž použití cyklu

int výsledek je jedno co bude, protože if dodá 1 místo něj, do něj se jen zapisuje



0, 1, 1, 2, 3, 5, 8, 13, 21, 34 výpis prvních 10, ale metoda se provedla 276krát

Funguje ale rychlost tohodle je o(n^2)



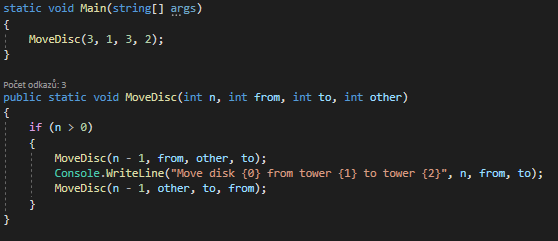
Stejný výpis, ale metoda se provedla 10krát

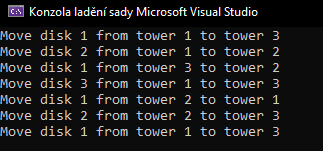
Rychlost O(n)

Hanojská věž

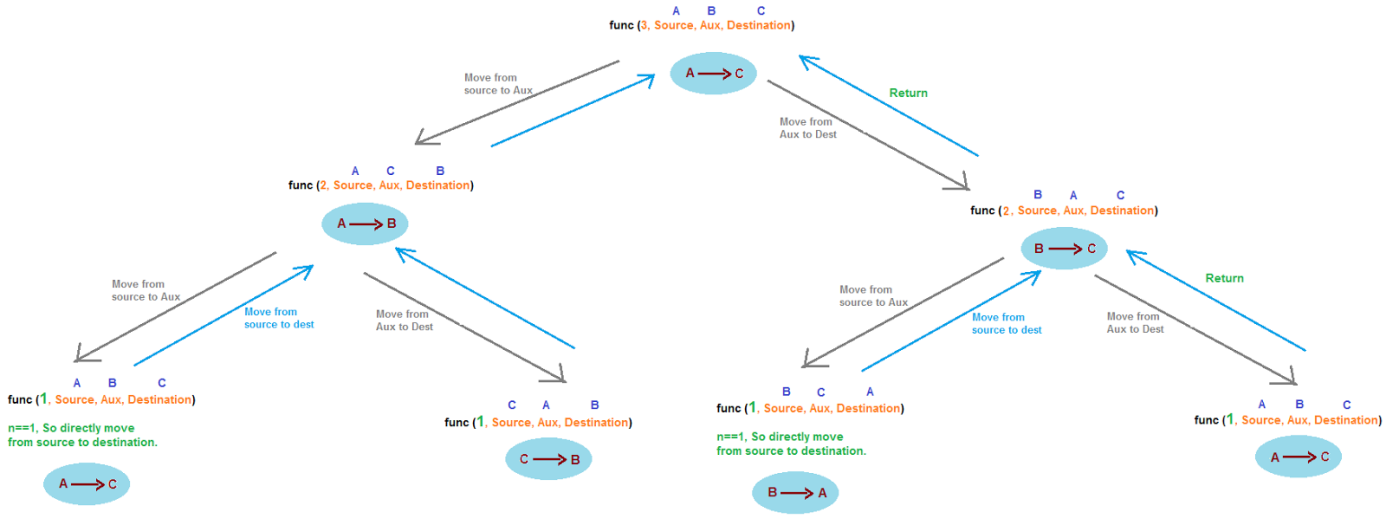
1. Je-li *n*>1, pak rekurzívním voláním této procedury přesuneme *n*–1 kotoučů (tj. všechny kromě největšího) z počáteční věže na odkládací.
2. Přesuneme největší kotouč z počáteční věže na cílovou.
3. Je-li *n*>1, pak rekurzívním voláním této procedury přesuneme *n*–1 kotoučů z odkládací věže na cílovou.

Z rekurzívního řešení lze dokázat matematickou indukcí, že pro n kotoučů potřebujeme 2^n–1 tahů

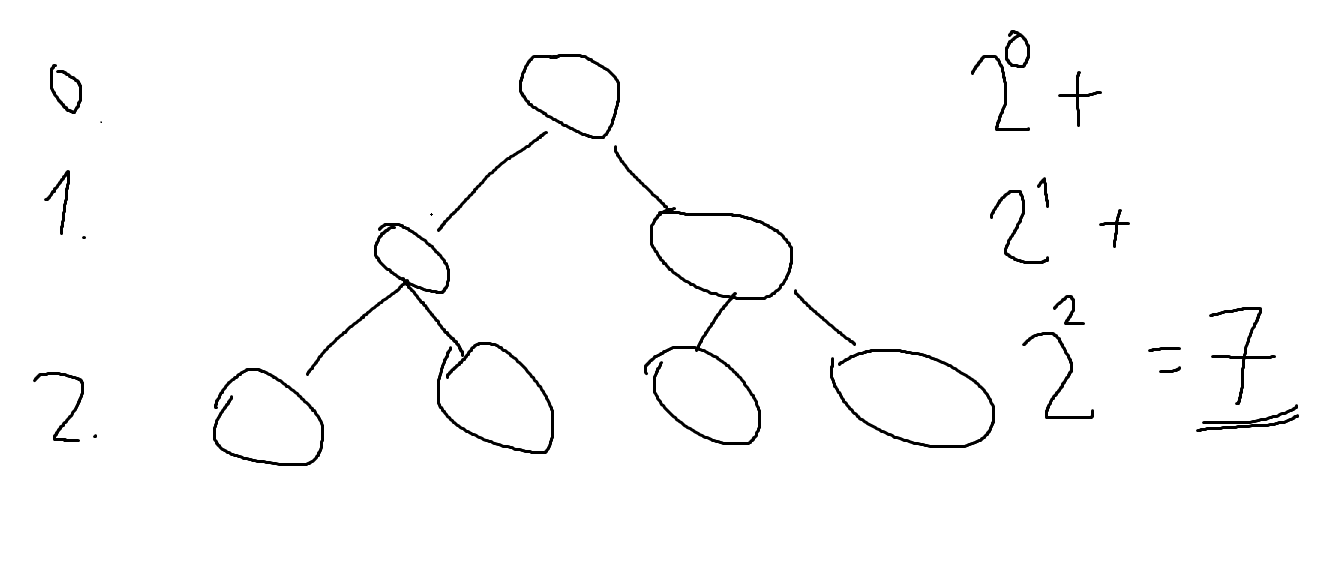








Strom



Prohledání stromovového prostoru

Vyhledání ve windows

Stromová

Metoda se vnoří na disk C a prohledá všechny adresáře ve všech adresářích

Brute Force

Brute force útok neboli **útok hrubou silou** je druh kyberútoku, jehož cílem je nejčastěji **prolomení hesla,** či celých přihlašovacích údajů. Útočníci používají software (prolamovač hesel), který postupně **zkouší různé kombinace znaků**, dokud neuhádne skutečné heslo. Tímto způsobem se mohou útočníci dostat do internetových služeb, zamčených souborů nebo do jakéhokoli digitálního prostoru, který vyžaduje uživatelské jméno a heslo. Tento proces je automatizovaný a vzhledem k tomu, že uživatelé často volí velmi jednoduchá hesla, bývá ve velkém procentu případů úspěšný.

Variantou brute force útoku je **slovníkový útok**, který nezkouší náhodné kombinace znaků, ale pracuje s **databází potenciálních hesel**, například zkouší nejčastěji používaná hesla.

Brute force útok je velmi snadný a může ho provést i nezkušený útočník. Jeho slabinou je **časová náročnost**, a proto jsou účinnou obranou**silná hesla**. Pokud vaše heslo kombinuje různé druhy znaků (písmena, čísla a další), velká a malá písmena a má alespoň 8 znaků, prolamovači hesel může trvat i **stovky let**, než přijde na správnou kombinaci. 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.

Brute force zkouší všechny myslitelné otázky

Bývá lineární, někdy i horší, je protivní, hackne každý bankovní účet

Nedá se použít, bude mu to trvat hodně dlouho

= dá se použít na problémy, které se dají vyřešit rychle

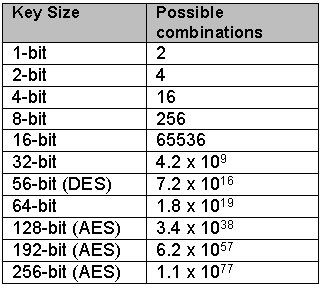
Každej problém se dá řešit brute forcem

Weby se mohou brute force útokům bránit **omezeným počtem pokusů** pro zadání přihlašovacích údajů a povinnými časovými intervaly mezi jednotlivými pokusy. Doplňující metodou je **captcha** nebo jiný způsob kontroly, zda přihlašovaný uživatel není robot.

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.

Dnes se používají klíče o délce 128, či 256 bitů.

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 bychom 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.

Vezme se algoritmus brute force a zapojí se do toho nějaký trik. Upraví se algoritmus a to může být efektivnější

Algortimus, do kterého přidáte nějakou nějakou chytrost, která ten algoritmus zlepší

Když dáte vyhledávat něco ve windows, tak třeba odřízne a neprohledá složky jako systémové drivery a logy, tím je rychlejší

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

1. def

Deterministický konečný automat (DKA, Deterministic finite automaton, DFA) je automat, který má konečné množství stavů a každý přechod je jednoznačný (neexistuje více možností přechodu z daného uzlu do jiných uzlů).

Má přesně daný co se bude dít, má přesně daný co se bude dít, každej stav závisí na předchozím stavu.

Nedeterministický konečný automat (NKA, Non-deterministic finite automaton, NDFA) tuto podmínku nerespektuje, je zde tedy více možností, kam z daného uzlu při daném vstupu přejít. Nedeterministický automat je „inteligentní“ - půjde vždy takovým směrem, aby nakonec skončil v koncovém stavu (pokud mu to vstup umožňuje).

Třeba házení kostkou, v programování náhodný ho čísla (pseudo), počítačem by se nemuselo být nedeterministické, vzniká v procesoru vzniká v procesoru a to je stroj z křemíku

Máme konstantní šanci

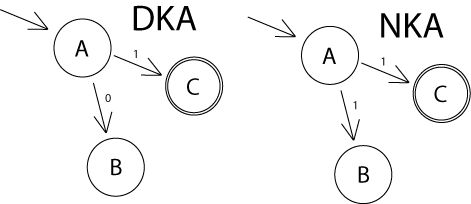
Z tohoto je zřejmé, že se daleko lépe vymýšlí nedeterministický automat, protože je jeho návrh daleko intuitivnější, ale na druhou stranu se výrazně hůře implementuje (vždy je zapotřebí jít všemi možnými směry, protože dopředu nevíme, který je správný, zároveň je pak tato implementace výpočetně náročnější). Je ale dokázáno, že oba druhy automatů jsou stejně výpočetně silné a lze je mezi sebou převést.

1. def

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

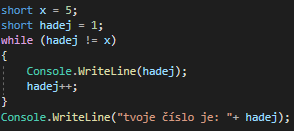


Monte carlo –

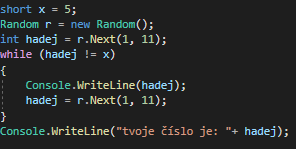
Příklady

Hádání náhodného čísla od 1 do 10

Deterministický má šanci 1/10, 1/9, 1/8, 1/7….



Nedeterministicky má šanci 1/10, 1/10, 1/10….



Hra lodě

Brute force - špatnej, jede políčko po políčku a měl by vždy být porážen

Heuristiky – hodně dobrej algoritmus by mohl být dobrej, ale může být odhalenej

Nedeterministický algortmus – třeba rozdělit pluchu na kvadranty a bude stejný počet střel do kvadrantu, nejlepší