Algoritmus vs. výpočetní metoda – algoritmus musí být konečný, zatímco výpočetní metoda ne

Algoritmus – postup k řešení nějakého problému

- Konečnost – musí skončit v konečném počtu kroků
- Obecnost – neřeší jeden konkrétní problém (např. „jak spočítat 3×7"), ale obecnou třídu obdobných problémů
- Determinovanost – Každý krok algoritmu musí být jednoznačně a přesně definován, pro stejné vstupy dostaneme pokaždé stejné výsledky.
- Výstup – musí mít alespoň jeden výstup

Typy algoritmu:

- Rekurzivní algoritmy
    - algoritmus (metoda) volá sama sebe, dokud není splněna podmínka
    - Po každém kroku volání sebe sama musí dojít ke zjednodušení problému. Pokud nenastane koncová situace, provede se rekurzivní krok.
    - lze přepsat do nerekurzivního tvaru při použití zásobníku nebo jiné paměťové struktury
    - Rozlišujeme dva základní typy dělení:
        - první dělení:
            - přímá (podprogram volá přímo sám sebe),
            - nepřímá rekurze (v příkazové části funkce A je volána funkce B, ve funkci B voláme funkci C, která volá funkci A)
        - druhé dělení (lineární, stromová rekurze)
- Hladové (greedy algorithm)
    - K řešení se propracovávají po jednotlivých rozhodnutích
    - V každém svém kroku vybírá lokální minimum, přičemž existuje šance, že takto nalezne minimum globální.
    - Použití: Problém obchodního cestujícího, problém batohu
- Rozděl a panuj
    - Rozděluje úlohu na dílčí části, které se zpracovávají a na konci se vhodným způsobem sloučí (quicksort)
- Dynamické programování
    - postupně řeší části problému od nejjednodušších po složitější s tím, že využívají výsledky již vyřešených jednodušších podproblémů.
- Pravděpodobnostní
    - provádějí některá rozhodnutí náhodně či pseudonáhodně. (možné rychlejší řešení těžko řešitelných problému (NP úplných)
- generické algoritmy
- heuristické
    - neklade si za cíl nalézt přesné řešení, ale pouze nějaké vhodné přiblížení.
    - Použití: když dostupné zdroje (např. Čas) nepostačují na využití přesných algoritmů (plánovače tras (Problému obchodního cestujícího), výroby)

Výpočetní složitosti (většinou to bývá, že buď jedno, nebo druhé je horší)

- Časová – kolik času je třeba na dokončení algoritmu v závislosti na vstup. datech

- Prostorova (Paměťová) – kolik paměti je třeba na vykonání algoritmu v závislosti na vstup. Datech

Definice

• algoritmy: sada instrukcí/příkazů, která zaručeně najde správné řešení v konečném počtu kroků

• problémy: je abstraktní popis spojený s otázkou, která vyžaduje řešení

• "instance" problému.

Asymptotická složitost nebo analýza algoritmu se týká určení matematických hranic nebo rámců jeho běhové výkonnosti

zjišťuje, jakým způsobem se bude chování algoritmu měnit v závislosti na změně velikosti (počtu) vstupních dat

Čas (nebo prostor) požadovaný algoritmem obvykle spadá do tří kategorií

• Best Case – Minimální čas potřebný pro provedení programu.

• Average Case – Průměrný čas potřebný pro provedení programu.

• Worst Case – to je maximální čas potřebný k provedení programu.

Využívají se:

• O notace (Big O notation) - scénář nejhoršího případu

• Omega notace (Big Omega notation) - měří nejlepší časovou složitost

• Theta notace (Big Theta notation) - vyjádřit dolní i horní hranici doby běhu algoritmu

Běžné třídy Asymptotické složitosti

- O(1) - konstantní
- O(log n) - logaritmická
- O(n) - lineární
- O(n^2) - kvadratická
- O(n!) – faktoriálová

Zrychlení algoritmů- musíme zjistit, ve které části programu trávíme nejvíce času (která část je "bottleneck"), a tu vylepšit

- Předpočítání a uložení do paměti
- Optimalizace pro HW
- Odstranění rekurze
- Optimalizace zdrojového kódu
- Heuristika - zkusmé řešení problémů, pro něž neznáme algoritmus nebo přesnější metodu

Gramatika – sada pravidel, uspořádaná množina (čtveřice)

Abeceda: Libovolná neprázdná konečná množina znaku

Slovo: Libovolná konečna, připadne i prázdna posloupnost

Jazyk: Je-li daná abeceda V, potom libovolná podmnožina množiny V* všech slov nad touto abecedou se nazývá jazyk. $L \subseteq V *$

Terminální symbol - prvek abecedy jazyka

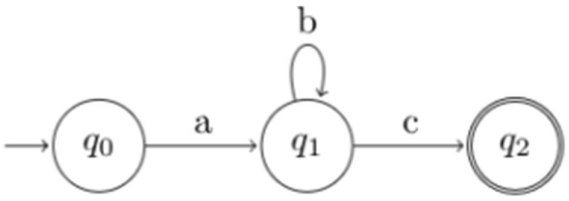
Neterminální symbol - proměnná (prvek abecedy proměnných), dále se nahrazuje (za další terminální nebo neterminální symbol)

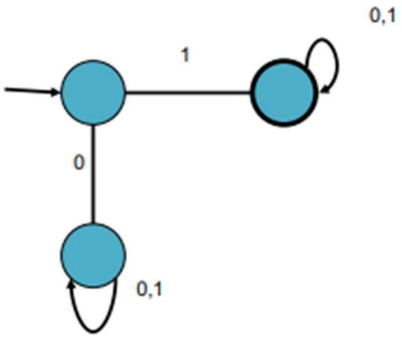
- Gramatika typu 0: Obsahuje všechny gramatiky a jimi generované jazyky L0
- Kontextová gramatika: Leva strana produkčních pravidel obsahuje definice v kontextu
- Bezkontextová gramatika: Na levé straně se nacházejí pouze neterminální symboly
- Regulární gramatika: používají je Konečné automaty
- BNF (Backus- Naur Form): jiná formu zápisu gramatiky (neterminální symboly se zapisuji do špičatých závorek, svislítko znamená nebo

Konečný automat

Abychom přesně formulovali problémy, jako jsou např.: co lze vyřešit pomocí počítače, jaké jsou „otázky“, na které nám dokáže zodpovědět, a jaká bude účinnost výpočtu, je nutné mít formální model, který popisuje práci počítače.

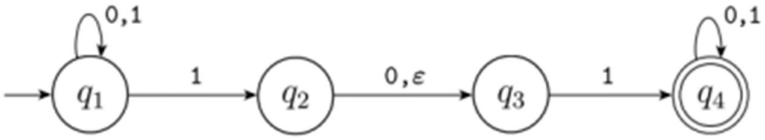
- Konečné automaty jsou teoretickým výpočetním modelem pro studium formálních jazyků.
- stroj, který má konečný počet stavů, ve kterých se může nacházet
- Na základě přečteného znaku vstupního slova a aktuálního stavu, ve kterém se nachází, pak změní svůj stav.
- Stroj proto PŘIJME (po přečtení posledního znaku se nachází v jednom z koncových stavů) nebo ODMÍTNE vstupní slova
- **Deterministický (KA)**
    - existuje pouze jeden stav, do kterého stroj může přejít
    - pouze jeden počáteční stav

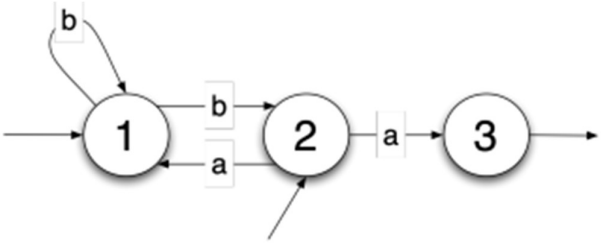




jako stavový diagram

- **Nedeterministický (NKA)**
  - existuje několik stavů, do kterých může automat vstoupit v jednom časovém okamžiku
  - přidáním nedeterminismu můžeme výrazně zvýšit efektivitu výpočtu
  - Epsilon přechod – přechod, u kterého může automat změnit stav, aniž by přečetl symbol vstupního slova
  - někdy je konstrukce nedeterministického automatu jednodušší než konstrukce deterministického automatu.





- Formálně lze konečný automat popsat jako uspořádanou pětici:
  - Q (konečná množina stavů)
  - I (vstupní abeceda)
  - F (přechodová funkce)
  - q0 (počáteční stav)
  - P (množina konečných stavů)

- Realizace paměti v KA bez zásobníku je řešena stavy

Pokud má nedeterministický konečný automat n stavů, má deterministický konečný automat 2n stavů

úkol, vyřešitelný bez problémů v rozumném čase nedeterministickými metodami, může být deterministickými metodami skutečně neřešitelný

Mealyho automat

- má oproti klasickému KA ještě výstupní abecedu
- Přechodová funkce f pak určí nejen nový stav, ale i symbol výstupní abecedy
- Výstup Mealyho automatu tedy závisí na zpracovávaném vstupním symbolu a na stavu, ve kterém se automat nachází.

Moorův automat

- výstupní symbol závisí pouze na aktuálním vnitřním stavu stroje
- Nezáleží na tom, jaký prvek vstupního slova se čte
- Příklad: stroj realizující součet dvou přirozených čísel zapsaných ve dvojkové soustavě, tzv. binární sčítačka

Počet stavů se může během převodu měnit (typicky se zvyšuje při převodu na Moorův), ale řešení vždy existuje.

Zásobníkový automat

- obsahuje jednu potenciálně neomezenou paměť – zásobník
- práce ve stylu LIFO
- Pokud je zásobník prázdný, automat svou práci ukončí
- jeho paměť je konečná a zásobníkový přístup možnosti výpočtu značně limituje (potřebovali by jsme alespoň **dva zásobníky**, aby byl schopen vypočítat cokoliv)

Turingův stroj

- konečný automat doplněný o paměť ve formě pásky
- stroj je zařízení, které má čtecí hlavu, nekonečnou pásku a tabulku chování, která určuje chování stroje při čtení aktuálního symbolu z pásky a m-konfiguraci (trojici: stavu, obsahu pásky a pozici hlavy),
- teoretický model počítače – je tvořený procesor. jednotkou, konečným automatem, mezivýsledky se zapisují na pásku

churchuv theorem: pokud zkoumáme pouze to, že problém lze vyřešit, není pro nás účinnost výpočtu podstatná

Postovy stroje

- je vývojový diagram s jedinou proměnnou x typu fronta (FIFO- first in first out)
- Vývojový diagram muže obsahovat následující bloky:
    - počáteční blok „START”
    - koncové bloky „AKCEPT” a „ZAMÍT”
    - bloky příkazu přiřazeni
    - blok logického příkazu

- Na počátku činnosti stroje je slovo w zapsáno do proměnné x, program začíná blokem „START”. Slovo w je akceptováno, jestliže se stroj v konečnem počtu kroků dostane do stavu „AKCEPT”. Slovo w je zamítnuto jestliže se stroj v konečném počtu kroku dostane do stavu „ZAMÍT”. V ostatních případech stroj cykluje

RASP stroje

- stroj s pamětí s přímým přístupem
- podobný Von Neumannově architektuře počítače - pracuje s registry a s řídící jednotkou
- mnohdy rychlejší (díky přímému adresování paměti - TS musí při čtení dvou vzdálených políček na pásce přečíst i vše mezi nimi

Třídy složitosti algoritmů (vyšší třída je vždy nadmnožina té nižší a ty se dají převést na vyšší)

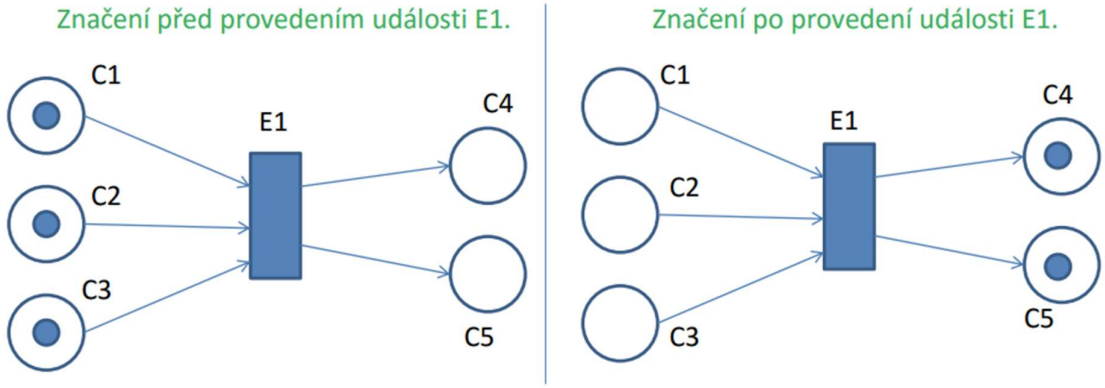
- P – považují se za zvládnutelné i když mohou být pro větší hodnoty náročné, obsahuje všechny problémy řešitelné pomoci determ. Turingova stroje, nebo RAM/RASP strojů
- NP
  - množina problémů, které lze řešit v polynomiálně omezeném čase na nedeter Turingově stroji. Namísto rozhodování stroj uhodne správnou cestu.
  - Každý problém P je zároveň problémem NP, takže platí NP = P. Otázka rovnosti P = NP je matematickým problémem tisíciletí a stále není vyřešena
- NPH – jsou minimálně tak těžké, jako nejtěžší NP problém
- NPC – třídu NP-úplných úloh tvoří v jistém smyslu ty nejtěžší úlohy z NP
  - Pokud bychom znali algoritmus pro řešení jakéhokoli problému NPC, vyřešili bychom všechny ostatní problémy NP a NPC současně
  - problém prvočísel – je jich mnoho a nelze je všechny najít (to samé hodnota Pi)
  - problém obchodního cestujícího – nalezení nejkratší možné cesty mezi body
  - problém batohu – problém kombinatorické optimalizace

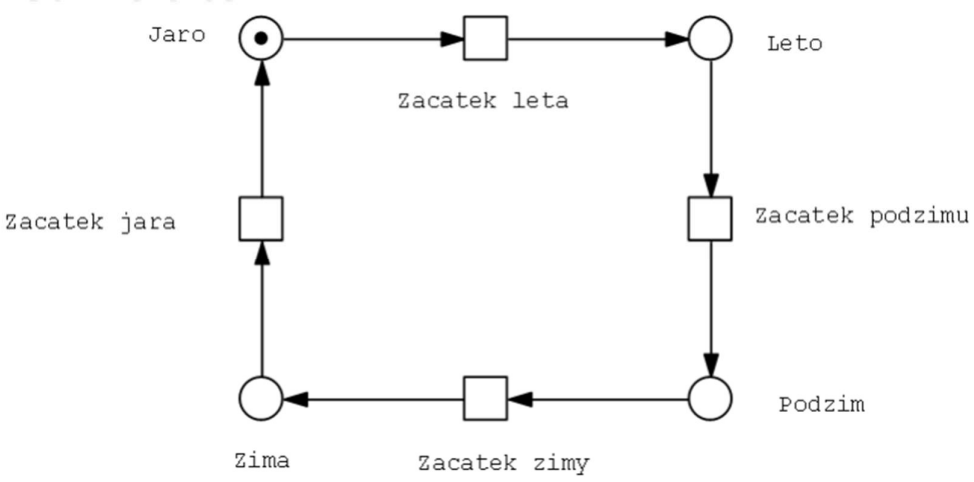
Petriho sítě

- nástroj pro modelování paralelních systémů a systémů s diskrétním časem
- Grafický popis a analýza systémů, ve kterých se vyskytují synchronizační, komunikační a zdroje sdílející procesy

1) C/E (Condition / Event) Petriho sítě.

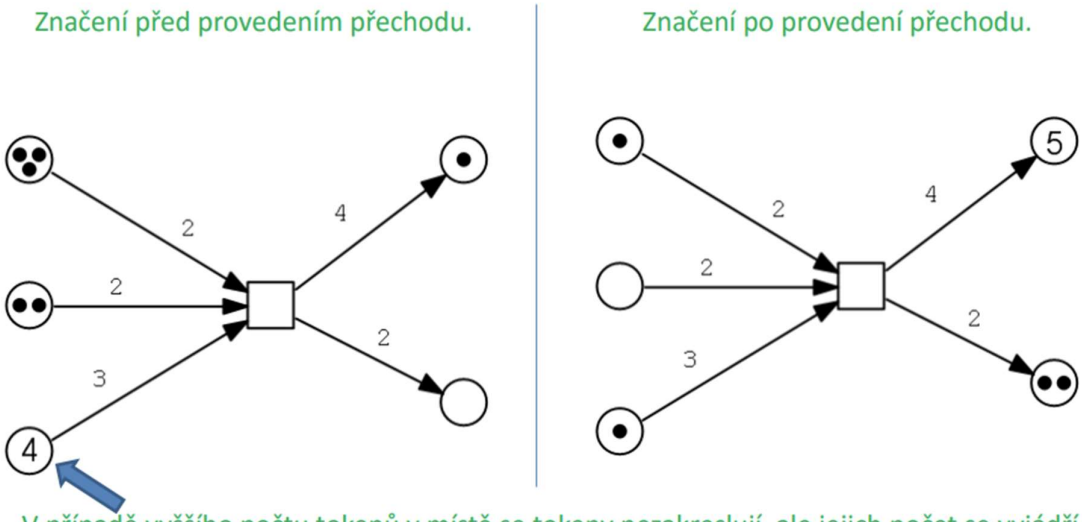
- Model se skládá z událostí (events) a podmínek (conditions), které musí být splněny, aby mohla nastat určitá událost
- Vazby mezi událostmi a podmínkami jsou znázorněny pomocí orientovaných hran (arcs)
- Obsahuje tyto prvky: podmínky, udalosti, orientovane hrany, tokeny, počateční značení
- Př.





2) P/T (Place / Transition) Petriho sítě.

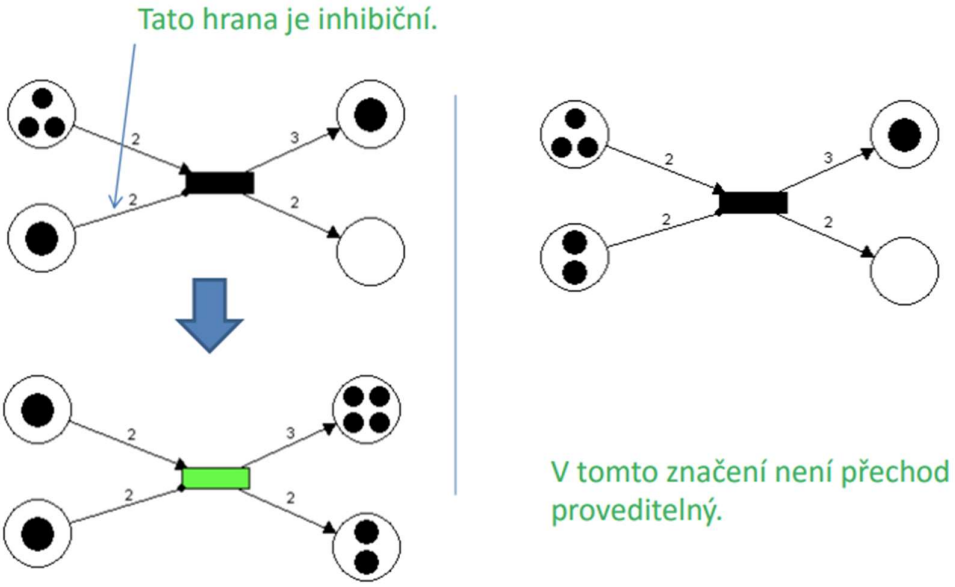
- Místo podmínek jsou místa (places), místo událostí jsou přechody (transitions)
- Pomocí míst zpravidla modelujeme stavy systému a pomocí přechodů změny stavů systému. Vazby jsou opět znázorněny pomocí orientovaných hran.
- Používaji také kapacitu (udává maximální počet tokenů, které se mohou v místě v jeden okamžik nacházet) a váhu (kolik tokenů se při provedení přechodu po dané hraně přesouvá)
- C/E Petriho síť je speciálním případem P/T Petriho sítě, kde kapacita každého místa je rovna 1 a váha každé hrany je taky rovna 1.



V případě vyššího počtu tokenů v místě se tokeny nezakreslují, ale jejich počet se vyjádří číslem. Kdyby toto místo obsahovalo např. pouze 2 tokeny, přechod by nebyl proveditelný.

3) P/T Petriho sítě s inhibičními hranami.

- je speciální hrana, která může směrovat pouze z místa k přechodu. U inhibiční hrany se zpravidla místo šipky zakresluje kolečko. Inhibiční hrany upravují proveditelnost přechodů.



4) P/T Petriho sítě s prioritami. - každému přechodu přiřazeno celé nezáporné číslo udávající prioritu přechodu.

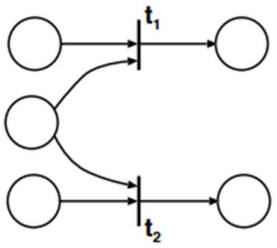
5) TPN Časované (Timed) Petriho sítě.

6) CPN Barevné (Coloured) Petriho sítě. - je obyčejná P/T Petriho síť, ve které je možno pracovat s více typy tokenů („tokeny různých barev“).

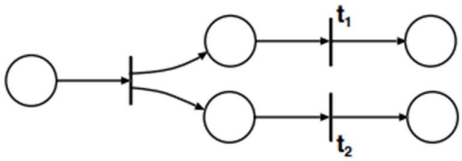
7) HPN Hierarchické (Hierarchical) Petriho sítě - umožňují členit vytvářenou síť na jednotlivé podsítě, které jsou navzájem propojeny

8) WPN Workflow Petriho sítě.- rozšíření jednotlivých typu petriho sítě

- Modelování vzájemné výlučnosti:
  t_1 a t_2 jsou vzájemně vyloučeny
  (konfliktní přechody)



- Modelování paralelnosti (simultánnosti):
  t_1 a t_2 jsou simultánní
  (nezávislé přechody)



Predikátový počet – část matematické logiky, která se zabývá popisem vnitřní (sémantické) struktury výroků. Důležitou součástí abecedy predikátového počtu jsou KVANTIFIKÁTORY.

Ekvivalence – stejný výsledek, trochu jiný postup

Izomorfismus – stejný program, stejný výsledek

Vstupní predikát – jak mají vypadat vstupní data

Výstupní predikát –požadovaný vztah mezi proměnnými při ukončení výpočtu

Vstupní vektor-nabývá vždy daných vstupních hodnot a během výpočtu se nemění

Výstupní vektor – obsahuje výstupní hodnoty v okamžiku skončeni vypočtu

Vektor programu –obsahuje všechny proměnné (mezivýsledky programu)

Verifikace programu: počítačový program dělá přesně to, co je uvedeno ve specifikaci programu, která udává, co má program realizovat

Testováním programu lze pouze zjistit přítomnost chyby, nikoliv její nepřítomnost!