Základy algoritmizace, Turingův stroj

Matematické algoritmy (K611MA)

Jan Přikryl, Miroslav Vlček

2. přednáška K611MA
čtvrtek 1. října 2009

verze: 2009-10-15 00:55

1 Teoretický základ algoritmizace

1.1 Intuitivní typy algoritmů

Vývojová stádia algoritmu Aneb intuitivní stádia geneze

V literatuře lze narazit na tvrzení, že jakékoliv řešení problému spadá do některé z následujících kategorií:

1. Na první pohled zřejmý postup
2. Metodický postup
3. Chytrý postup
4. Geniální nápad

Celkově jde o postupný vývoj „inteligence“ algoritmu od většinou naivního a časově neefektivního postupu ke složitějším, ale efektivnějším metodám řešení.

Vývojová stádia algoritmu Primitivní a naivní řešení

- Základní úroveň pokusů problém uchopit, pokud se nevyznáme (nebo jsme líní)
- Prohledávání příliš rozsáhlého prostoru možných řešení
- Většinou je to to, co vás napadne jako první ;-).

Příklad

Řazení seznamu záměnou sousedních prvků (bubble-sort) či zatřiďováním vždy jednoho prvku na správnou pozici (insert-sort, ten má ovšem své opodstatnění).

Vývojová stádia algoritmu Metodická řešení

- Hlubší analýza problému
- Netriviální metodický postup
- Rozdělení úlohy na podproblémy

Příklad

Řazení seznamu rekurzivní metodou merge-sort, respektive quick-sort. Možná heap-sort, i když ten aspoň zčásti patří i na další stránku.

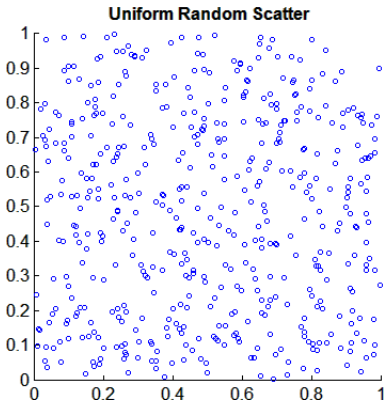
Vývojová stádia algoritmu Chytrá řešení

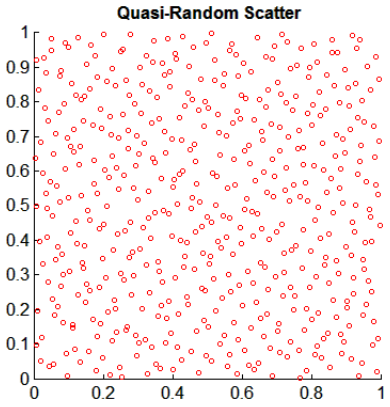
- Často specializovaná
- Vyžadují už jít do hloubky zpracovávaného problému, analyzovat vlastnosti dat
- Většinou není na první pohled jasné, proč to funguje
- Lehká nebývá ani analýza složitosti

Příklad

Snad záměna pseudonáhodných za kvazináhodná čísla pro generování vzorků v Monte-Carlo simulacích. Nebo sekvenční Monte-Carlo.

Odbočka Rozdíl mezi pseudonáhodným a kvazináhodným





1.2 Rozdělení algoritmů podle implementace

Rozdělení algoritmů podle způsobu implementace Když dva dělají totéž, nemusí to být stejným způsobem

Existuje mnoho různých možných přístupů k algoritmizaci postupu řešení:

1. Rekurzivní vs. iterativní přístup
2. Logické programování (algoritmus=logika+řízení)
3. Sériový vs. paralelní výpočet
4. Deterministický vs. nedeterministický přístup
5. Přesné vs. přibližné řešení

Rozdělení algoritmů podle způsobu implementace Rekurzivní vs. iterativní přístup

Rekurze: při řešení úlohy provádí algoritmus sám sebe nad vybranou podmnožinou vstupních dat.

⇳

Iterace: při řešení úlohy provádí algoritmus opakovaně (v cyklu) stejnou úlohu nad měnící se množinou dat.

Example 1. Výpočet Fibonacciho posloupnosti $F(n) = F(n-1) + F(n-2)$ lze provádět oběma způsoby.

Typický příklad rekurze je postup rozděl a panuj.

Počáteční množina dat při iterativním postupu je množinou vstupních dat.

Fibonacciho posloupnost zmíníme detailněji v sedmé přednášce.

Rozdělení algoritmů podle způsobu implementace Logické programování (algoritmus=logika+řízení)

Použití prostředků matematické logiky k vyjádření postupu výpočtu.

Logický program je dán

- výrokovou logikou – zápis odvozovacích pravidel

- řízením – zpracování zapsaných pravidel nějakou formou automatického dokazování

Příklad **deklarativního programování:** výroková logika pouze říká, *jaký* výpočet se má provést a nikoliv, *jak* tento výpočet provést.

K paradigmatu deklarativního programování můžeme kromě logického programování ještě začlenit třeba funkcionální programování (Lisp) nebo regulární výrazy.

Automatické dokazování není konkrétní implementací zapsané logiky postupu výpočtu, je implementované jako nějaký obecně platný algoritmický postup, jehož vstupem je seznam výroků a vstupní data a výstupem je rozhodnutí o tom, jestli pro daný vstup výrok platí nebo ne (respektive výstup generovaný na základě vstupního seznamu výroků).

Rozdělení algoritmů podle způsobu implementace Sériový vs. paralelní výpočet

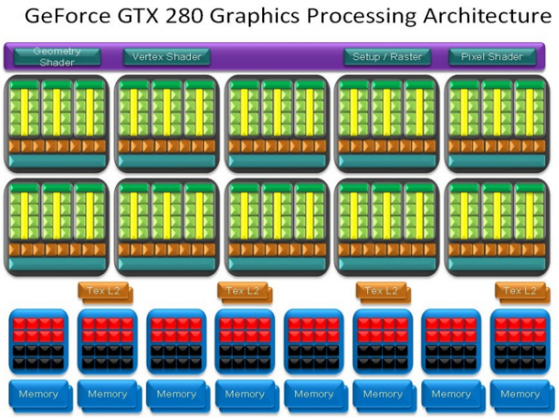
Při sériovém výpočtu počítáme vždy jenom jednu instrukci, algoritmus se vykonává jedinkrát. *Varianta:* SISD.

Paralelní výpočet znamená více instancí algoritmu na více procesorech nebo počítačích. *Varianty:* MISD, SIMD, MIMD.

Example 2. SIMD: GPU moderních grafických karet. MIMD: distribuované výpočty Seti@home, GIMPS a podobné.

Flynnova taxonomie počítačových architektur: SISD (Single Instruction, Single Data stream) je klasický sériově vykonávaných algoritmus, klasický jednoprocesorový jednojádrový počítač. SIMD (Single Instruction, Multiple Data streams) je vektorový přístup k výpočtu, kdy jednu instrukci (například operaci sčítání) provádíme najednou nad vektorem n dat. MISD (Multiple Instruction, Single Data stream) je poněkud neobvyklý postup při němž více procesorů počítá nad stejnými daty; využívá se při ochraně systémů proti chybám. MIMD (Multiple Instruction, Multiple Data streams) označuje zcela distribuovaný výpočet.
`http://en.wikipedia.org/wiki/Flynn's_taxonomy`

Výhoda paralelního přístupu Jak vypadá vevnitř grafická karta



Celkem 10 ALU skupin, v každé 3×8 ALU jednotek (3 stínovací procesory a 8 texturovacích jednotek v jedné skupině).

Výhodou moderních grafických karet nVidia a ATI je možnost využít hardware karty i k běžným matematickým výpočtům – viz CUDA a AMD Stream. Karty se pak chovají přibližně jako SIMD architektura.

Rozdělení algoritmů podle způsobu implementace Deterministický vs. nedeterministický přístup

Deterministický přístup odpovídá striktní definici algoritmu: vždy dospěje ke stejnému výsledku.

Nedeterministický algoritmus obsahuje jeden nebo více bodů, v nichž následující krok není pevně určen a náhodně se zvolí jedna z předem definovaných akcí – nedospěje vždy ke stejnému výsledku.

Example 3. Balení (mého) batohu na hory.

Rozdělení algoritmů podle způsobu implementace Přesné vs. přibližné řešení

Některé úlohy trvají vyřešit příliš dlouho nebo je nelze vyřešit v rozumném čase vůbec.

Pro „těžké“ problémy je často postačující alespoň přibližné (téměř optimální) řešení.

Example 4. Problém obchodního cestujícího: Zkuste si zoptimalizovat cestu přes stovku zastávek pro kurýra UPS/DHL/FedExu.

1.3 Základní postupy při algoritmizaci

Základní postupy při algoritmizaci Aneb rozdělení podle návrhového paradigmatu

1. Rozděl a panuj
2. Redukce
3. Lineární programování
4. Dynamické programování
5. Hladové algoritmy (greedy methods)
6. Probabilistické a heuristické algoritmy

Rozděl a panuj Divide and Conquer

Klasický přístup k dekompozici problému na jednodušší podúlohy stejného problému (divide) a spojení jejich řešení v jeden celek (conquer).

- Často spojen s rekurzí
- Podproblémy nemusí být nutně *zcela* stejného typu
- *Výhody:* snadná paralelizace, rychlost, vhodný pro konceptuálně složité problémy
- *Nevýhody:* paměťová náročnost, rekurze

Example 5. Již zmíněný merge-sort, jenž dělí řazená data na menší podmnožiny, jež pak řadí stejným přístupem do té doby, než narazí na jednoprvkovou množinu dat – ta je seřazená vždy.

Další příklady: FFT, metoda půlení intervalu v numerice.

Redukce Převod do duální reprezentace

Redukcí nazýváme transformaci řešeného problémy na jiný, ekvivalentní problém, jenž umíme řešit efektivněji.

- Převádíme složité úlohy na úlohy s nižší asymptotickou složitostí
- I za cenu převodu bude řešení trvat kratší dobu

Example 6. Hledání mediánu množiny dat. Prvním krokem může být seřazení celé množiny do vektoru dat, druhým potom výběr prostředního prvku z tohoto vektoru.

Lineární programování Jednoduchý přístup k optimalizaci

Optimalizační postup na nalezení váženého maxima soustavy lineárních rovnic a nerovnic za určitých pevných omezení

- Problém vyjádříme soustavou rovnic, nerovnic a omezení
- Tuto soustavu nám vyřeší nějaký generický řešič (simplexová metoda)
- Není to programovací postup per se
- Častou variantou je celočíselné programování (prostor omezen na celá čísla)

Příklady

Nalezení maximálního toku mezi dvěma uzly orientovaného grafu.

Optimalizace délky zelených pro SSZ snižující délky front vozidel na semaforech.

Dynamické programování Jak si ušetřit práci

Při řešení složitých problémů s určitou strukturou se často velké bloky podúloh počítají vícekrát. Dynamické programování zamezuje opakovanému výpočtu zapamatováním si výsledků vyřešených podproblémů

- Podobné paradigmatu rozděl a panuj – podúlohy ale jsou různého typu
- Pokud se v řešení úlohy nevyskytují opakující se shodné podúlohy, nijak nám DP nepomůže

Example 7. Nalezení nejkratší cesty mezi dvěma uzly orientovaného ohodnoceného grafu.

Hladové algoritmy Greedy Algorithms

Určitou třídu úloh lze řešit výběrem lokálního optima (maxima, minima) v každém kroku algoritmu a mít zaručeno, že dojdeme ke globálnímu optimu.

Example 8. Výběr nejmenšího počtu platidel pro určitou sumu. Nefunguje pro denominace 1,7,10.

Probabilistické a heuristické metody

Probabilistický přístup používají **randomizované algoritmy**, například Monte-Carlo metody. Výsledkem je (rychlý) intervalový odhad řešení na určité hladině pravděpodobnosti.

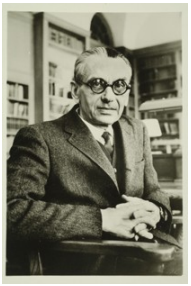
Heuristické metody používají k řešení hrubý odhad na základě zkušeností programátora či odpozorovaných obvyklých výsledků.

2 Konečné automaty a Turingův stroj

2.1 Konečný automat

Matematické modely počítačů

Již v roce 1936 se *Alan Turing*, *Alonso Church* a *Kurt Gödel* zabývali teoretickými základy, na nichž by se dala moderní počítačová věda definovat.



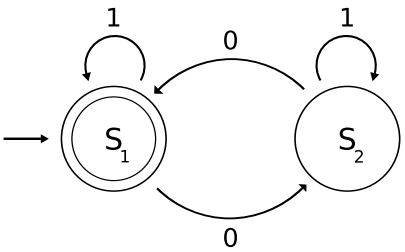
Nezávisle na sobě hledali univerzální model, který by z hlediska funkce bez ohledu na fyzikální vlastnosti popsal chování libovolného výpočtu – *algoritmu*.

Nejčastěji zmiňovaným řešením se stal model Alana Turinga, který byl založen na matematické abstrakci výpočetního stroje jako *konečného automatu*.

Co je to konečný automat Finite State Machine (FSM)

Primitivní automat, jenž čte symboly nějaké definované abecedy na vstupu a podle nich přechází mezi různými vnitřními stavy. Nemá vnitřní paměť.

Může být deterministický nebo nedeterministický.



Nedeterministický konečný automat nemá pevnou tabulku přechodů, ale změna stavu nastane pouze s určitou pravděpodobností.

2.2 Turingův stroj

Ve své první velké publikaci z roku 1936 nazvané *On computable numbers, with an application to the Entscheidungsproblem* představil Turing svůj abstraktní *Turingův stroj* (anglicky je to *Turing Machine*, Tu-

ring jej ve svém článku označuje jen jako „výpočetní stroj“, na Turingův stroj jej pokřtil až americký logik Alonzo Church). Článek 'On computable numbers' je průkopníkem základní myšlenky důležité pro moderní počítače – konceptu ovládání operací výpočetního stroje prostřednictvím programu zakódovaného v instrukcích uložených v paměti zařízení. Toto dílo mělo hluboký vliv na vývoj digitálních počítačů s programem uloženým v elektronické formě ve 40. letech 20. století – vliv často zanedbávaný nebo dokonce popíraný počítačovými historiky.

Turingův stroj je abstraktním konceptuálním modelem. Skládá se ze skeneru a neomezené paměťové-páska. Páska je rozdělena do čtverců, z nichž každý může být prázdný nebo může obsahovat právě jeden symbol (například '0' nebo '1' nebo nějaký jiný symbol převzatý z konečné abecedy szmbolů). Skener se pohybuje tam a zpět přes paměťovou pásku, zkoumá v jednom okamžiku právě jeden čtverec ('naskenovaný čtverec'). Stroj tedy čte symboly na pásce, a případně na pásku zapisuje další symboly. Páska tak tvoří jak paměť stroje, tak i prostředek pro vstup a výstup. Páska může obsahovat také program instrukcí. (I když páska sám o sobě je nekonečná – Turingovým cílem bylo ukázat, že existují úkoly, jež Turingovy stroje nemohou vykonávat, a to i bez omezení pracovní paměti a výpočetního času dobu – každý vstupní text napsaný na pásce se musí skládat z konečného počtu symbolů.)

Turingův stroj Formální popis

Motivace: Existuje mechanický proces, kterým je možno rozhodnout o pravdivosti libovolného matematického teorému nebo výroku.

Turingův stroj simuluje práci matematika při vytváření důkazu:

- nekonečná tabule (pro psaní i čtení)
- mozek (řídící jednotka)

Formalizace Turingova stroje

- místo tabule oboustranně nekonečná páska
- místo křídy čtecí a zapisovací hlava, kterou lze posouvat
- místo mozku konečná řídicí jednotka

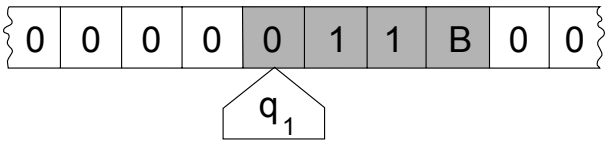
Turingův stroj Vlastnosti (1)

Takový stroj měl několik základních vlastností:

- musel nahradit složitou symboliku matematických kroků. V takovém případě šlo každou konečnou množinu symbolů nahradit pouze dvěma symboly (jako je 0 a 1) a prázdnou mezerou, která by oba symboly oddělovala.
- podobně jako si matematik zapisuje poznámky na papír, má Turingův stroj k zápisu nekonečnou pásku skládající se z buněk, do/ze kterých se symbol zapisuje/čte.
- nad touto páskou je možné provádět za pomocí čtecí hlavy operace čtení, zápisu a posunu pásky (*read*, *write*, *shift left*, *shift right*).

Turingův stroj Vlastnosti (2)

- protože je možné symboly číst, zapisovat nebo se posunovat po pásce, je pro Turingův stroj důležitý vnitřní stav, ve kterém operaci čtení provádíme (čtený symbol a stav určují další akci a přechod do dalšího stavu)



Protože se chování tohoto stroje vyvíjí podle tabulky přechodů, můžeme říci, že každý následující stav lze jednoznačně určit na základě čteného symbolu a aktuálního stavu.

Chování Turingova stroje je proto *deterministické*.

Turingův stroj Formální definice

Klasický Turingův stroj je konečný automat, jenž můžeme popsat sedmicí $T \equiv \{Q, \Gamma, b, \Sigma, \delta, q_0, F\}$, kde

- Q je konečná množina vnitřních stavů,
- Γ je konečná množina symbolů abecedy na pásce,
- $b \in \Gamma$ je symbol pro prázdné políčko,
- $\Sigma \subseteq \Gamma \setminus \{b\}$ je množina vstupních symbolů na pásce,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, P\}$ je přechodová funkce, popisující změnu stavu, zápis na pásku a posun hlavy,
- $q_0 \in Q$ je počáteční stav stroje,
- $F \subseteq Q$ je množina koncových (akceptovaných) stavů.

Cokoliv, co odpovídá tomuto formálnímu zápisu, je Turingův stroj.

Bližší popis Turingova stroje naleznete například v knize B. J. Copelanda *Alan Turing's Automatic Computing Engine: The Master Codebreaker's Struggle to Build the Modern Computer* (Oxford Univeristy Press, 2005) či na Wikipedii.