Teorie algoritmů

Petr Horáček

Obsah

Kapitola 1

Úvod

Teorie algoritmů je obor související s matematikou a informatikou, který se zabývá vlastnostmi a popisem algoritmů. Díky pochopení způsobu fungování algoritmu je možné dané algoritmy zkoumat, porovnávat s různými druhy stejného algoritmu, optimalizovat a popsat jej.

Teorie algoritmů je důležitou součástí softwarového inženýrství, které se zabývá navrhováním a sestavováním algoritmů a převádění do specifického programovacího jazyka.

Kapitola 2

Algoritmus

Algoritmus je přesný popis postupu, kterým lze vykonat nějakou přesně definovanou činnost (úlohu), který je prováděn pomocí konečného počtu přesně definovaných kroků. Jako jistý druh algoritmu lze chápat například recept na přípravu jídla. V softwarovém inženýrství se algoritmus používá pro popis funkce výsledného programu. V prostředí teorie algoritmů pojem **procesor** označuje objekt (fyzický nebo i virtuální), který vykonává daný algoritmus. Procesor může být nějaký jednoúčelový robot, cnc soustruh, nebo dokonce člověk. V případě počítačů je za procesor považován integrovaný obvod (CPU), který umožňuje vykonávat elementární operace, ze kterých je daný algoritmus sestaven.

Základem algoritmu je **příkaz**, který popisuje elementární operaci, kterou může procesor vykonat. Elementární operace je pro daný procesor jiná a proto nelze porovnávat konkrétní algoritmus mezi různými procesory. Ale lze pomocí jednoho algoritmu porovnávat efektivitu více různých procesorů. Vedle základních příkazů procesor zpravidla disponuje i řídícími příkazy, které umožňují vytvářet složitější příkazy, zkrátit výsledný algoritmus a umožnit jeho větší univerzálnost.

2.1 Vlastnosti algoritmu

Algoritmus se vyznačuje určitými vlastnostmi, které musí splňovat k tomu, aby byl v praxi proveditelný. Pokud by danému algoritmu scházela některá z jeho vlastností, došlo by při jeho vykonávání v počítači k nežádoucím výsledkům (zatuhnutí programu, chybné výsledky, ...).

2.1.1 Začátek a konec algoritmu

Každý algoritmus musí někde začínat, tedy musí mít přesně definované místo, kde začíná, jinak by mohl začít od libovolné své části, což by způsobilo, že by se některé jeho části neprovedly. Tyto části mohou být důležité pro správný výsledek algoritmu. V programovacích jazycích je toto místo označeno nějakých speciálním identifikátorem (funkce main), který procesoru říká odkud daný algoritmus začíná. Algoritmus musí mít také konec, jinak by nikdy neskončil. To by mohlo způsobit zablokování procesoru. V případě, že je potřeba daný algoritmus vykonat pouze jednou by po jeho skončení procesor přešel opět na jeho začátek a pokračoval dále. Tím by došlo k zacyklení procesoru a vykonávání algo-

ritmu by bylo potřeba přerušit vnějším zásahem. Algoritmus musí vždy skončnit s konečným počte kroků. Takový algoritmus se nazývá **konečný algoritmus**.

2.1.2 Elementárnost

Algoritmus se skládá z konečného množství jednoduchých (elementárních) a snadno srozumitelných kroků. Díky tomu je možné porovnávat efektivnost různých algoritmů, které podávají na základě stejných vstupních dat i stejné výstupy. Algoritmus, který se skládá z menšího počtu kroků, je pro danou aplikaci efektivnější (rychlejší).

2.1.3 Determinovanost

Deterministický (jednoznačný) algoritmus musí vždy za stejných vstupních podmínek svým vykonáváním vytvořit vždy stejné výsledky (je předvídatelný). V každém kroku musí jít jednoznačně určit zda daný algoritmus již zkončil a pokud ne, musí jít určit jaký krok následuje.

V případě deterministického algoritmu nesmí nastat situace, kdy algotitmus neví co má dělat. To může nastat v případě, kdy není dostatečně ošetřen proti nepovoleným vstupům. Jedná se o nejčastější chybu při tvorbě programů. Při návrhu algoritmu je nutné dopodrobna promyslet jaké stavy mohou při jeho vykonávání nastat a vhodným způsobem je ošetřit aby nemohlo dijít k zakázaným stavům (dělení nulou, ...). Pokud nejsou všechny eventuality algoritmu vhodným způsobem ošetřeny, může nastat jeho zablokování, nebo může dojít k nějakému nevyžádanému stavu programu, který bude ovlivňovat jeho další vykonávání. Hledáním těchto neošetřených stavů se zabývá **testování algoritmů** (softwaru).

Opakem deterministického algoritmu jsou **nedeterministické algoritmy**, kterné nemusejí mít vždy určen následující krok (může záležet na situaci). Takový algotimus může například volit z několika různých možností, mezi kterými vybírá na základě různých kritérií. Nedeterministické algoritmy jsou takové algoritmy, které se projevují určitým typem vlastní inteligence rozhodování - jsou základem inteligentních algoritmů.

2.1.4 Věcná správnost

Věcná správnost (korektnost) algoritmu říká, že daný algoritmus zkutečně dělá to co se po něm chce. Porušení věcné správnosti algoritmu může být způsobená například zvolením špatného vzorce pro daný výpočet, nebo syntaktickou chybou v zápise. Princip fungování algoritmu může být sice správný, ale při jeho sestavování došlo k nějaké chybě.

Porušení věcné správnosti se projeví tak, že algoritmus nemusí vůbec fungovat a nebo vrací špatné (nepředpokládané) výsledky.

2.1.5 Obecnost

Obecný algoritmu je takový algoritmus, který neřeší jeden konkrétní problém, ale lze ho využít pro řešení dané třídy problému. Každý algoritmus musí být co nejobecnější, aby bylo možné pomocí něj řešit co nejširší množství úloh stejného druhu. Díky tomu lze daný algoritmus využívat jako menší součást různých větších algoritmů. To zrychluje vývoj nových algoritmů a zvětšuje jejich použitelnost. Typickým příkladem je součet dvou čísel. Pokud je zapotřebí sečíst nějaká dvě konkrétní čísla například $1 + 3$, je lepší daný algoritmus zobecnit na libovolná dvě čísla $A + B$, protože je možné, že v budoucnu bude potřeba sestavit stejný algoritmus pro jiná dvě čísla.

Obecnost algoritmu umožňuje vstup parametrů (parametrizace algoritmu) do algoritmu. To znamená, že do programu lze vložit hodnoty, které jej ovlivní. Na základě vstupních parametrů je pak vytvořen žádaný výstup.

2.1.6 Opakovatelnost

Základní podmínkou pro správný algoritmus je, že je možné jej kdykoli zopakovat a za stejných okolností se bude chovat vždy stejně. To znamená, že při zadání stejných vstupních dat algoritmus dospěje vždy ke stejným výsledkům.

Takové chyby jsou většinou způsobeny chybou při implementaci algoritmu. Algoritmus sám je navržen správně, ale je špatně fyzicky sestaven.

2.1.7 Výstup

Algoritmus musí mít vždy 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). To algoritmu umožňuje komunikovat s okolím.

2.1.8 Srozumitelnost

Algoritmus je nutné určitým způsobem zaznamenat. Způsobů jak algoritmus zaznamenat je více, ale obecně musejí splňovat podmínku srozumitelnosti. Zaznamenanému algoritmu musí porozumět ne jen programátor, který podle něj vytváří daný program, ale i jeho tvůrce, který může daný algoritmus v budoucnosti nějak upravovat. Každý způsob se využívá

v jiných situacích (kontextu použití) a k různým účelům. Způsoby zaznamenávání algoritmů lze rozdělit na textové a grafické.

Pro zápis algoritmů se nejčastěji používají tyto metody:

- Slovní vyjádření
- Matematický zápis
- Rozhodovací tabulky
- Vývojové diagramy
- UML - jednotný modelovací jazyk
- Počítačové programy

Slovní vyjádření algoritmu je dostatečně srozumitelný a přesný popis nějaké činnosti pomocí slov z běžného života. Typickým příkladem jsou různé návody a příručky. Výhodou slovního vyjádření je jeho jednoduchost a srozumitelnost. Nevýhoda je ale jeho nízká přehlednost. Při složitějších a rozsáhlejších algoritmech je těžké se v něm orientovat a uhlídat podmínku jednoznačnosti. Slovní vyjádření algoritmů se používá například jako část dokumentace softwarového projektu. V případě vývoje softwaru se jako doplněk vývojových diagramů používají slova, která připomínají operace v programu. Jedná se tedy o postup popisu algoritmů složený z textové i grafické formy. Pro slovní vyjádření algoritmů pro potřeby vývoje programů se používá kódový jazyk, který se nazývá **pseudokód**. Jedná se seznam slov, které připomínají výrazy a příkazy z programovacích jazyků vyšších úrovní.

Matematický zápis je formou vyjádření algoritmu, kterou je vhodné použít všude tam, kde je možné danou problematiku popsat matematicky. Výhodou matematického zápisu je jednoznačnost, podle rovnice jsou přesně definovány výstupy algoritmu. Nevýhodou je ale, že bývá méně podrobný a nelze ho přímo zadat do počítače. Navíc je po přepsání do počítačového jazyka ještě potřeba ošetřit zakázané hodnoty (dělení nulou, ...).

Rozhodovací tabulky jsou vhodné v případech kdy se v algoritmu vyskytuje několik možností a vlastní řešení je pro každou možnost jednoznačně popsatelné. Příkladem je **tabulka pravdivostních hodnot**, která se skládá z n sloupců a 2^n řádků. Tabulka je horizontálně rozdělena na dvě části, kde v jedné části se nacházejí vstupní hodnoty a v druhé části výstupy jejichž hodnoty závisejí na vstupních hodnotách. Výhodou je naprostá jednoznačnost, jednoduchost a přehlednost. Nevýhodou je ale, že se nehodí pro všechny typy úloh. U některých úloh by tabulka byla

příliš rozsáhlá a ztratila by svou přehlednost.

Vývojové diagramy jsou typem symbolického (grafického) algoritmického jazyka, který se používá pro názorné zobrazení algoritmů. Jedná se o jednu z nejdokonalejších forem zápisu algoritmu, ale většinou bývá pro úplnost doplněn ještě některými jinými formami zápisu. Vývojové diagramy se skládají z jednotlivých grafických symbolů, které jsou mezi sebou spojeny orientovanými hranami - čárami které jsou doplněny šipkou vyjadřující směr postupu algoritmu. Obecný postup psaní algoritmů je od shora dolů a zleva doprava. To může být v určitých případech porušeno - cykly, skoky v algoritmu, ...

UML je zkratka pro **jednotný modelovací jazyk** ...

Kapitola 3

Analýza algoritmů

3.1 Asymptotická složitost

Asymptotická složitost je nástroj, který umožňuje určit výkon algoritmů a porovnávat je vzájemně mezi sebou.