A. Jančařík
Početní algoritmy III - složitost

Učitel matematiky, Vol. 15 (2007), No. 3, 143–150

Persistent URL: http://dml.cz/dmlcz/150679

POČETNÍ ALGORITMY III – SLOŽITOST

ANTONÍN JANČAŘÍK[3]

V předchozím článku o početních algoritmech jsem představil několik algoritmů věnovaných násobení. V případě, že máme k dispozici několik různých algoritmů, naskýtá se poměrně logická otázka, který z nich je nejlepší.

Lze algoritmy vůbec nějak porovnávat a hodnotit? Má takové srovnání smysl? Mohou být některé algoritmy efektivnější než jiné? Otázkami, jak je algoritmus efektivní, jak rychle pracuje, kolik spotřebuje místa, či jak efektivnost algoritmů porovnávat, se zabývá *teorie složitosti*. Teorie složitosti je samostatný obor na pomezí mezi matematikou a teoretickou informatikou. Jednou z otázek, kterou se zabývá, je i jeden z nejznámějších nevyřešených problémů současnosti – zda existuje úloha, která se nedá řešit v „rozumném" čase, ale jejíž řešení se dá v „rozumném" čase ověřit (tzv. P/NP problém). Cílem tohoto článku je demonstrovat některé základní myšlenky teorie složitosti na jednoduchých příkladech a praktických ukázkách srozumitelných i žákům základních a středních škol.

Časová náročnost – Hornerovo schéma

Důležitou otázkou, kterou je vhodné u každého algoritmu zodpovědět, je, jaký počet kroků je nutný dle algoritmu provést pro řešení úlohy. Jako příklad uveďme výpočet hodnoty polynomické funkce v bodě. Např. chceme zjistit hodnotu polynomu $5x^3 - 6x^2 + 4x - 5$ v bodě 3. Pokud tuto úlohu řešíme běžným dosazováním, tak potřebujeme dvě násobení na výpočet třetí mocniny čísla 3, dále pak tři násobení na vypočítaní jednotlivých sčítanců ($5 \cdot 3^3$, $6 \cdot 3^2$, $4 \cdot 3$) a tři součty (resp. rozdíly) na sečtení všech členů výrazu dohromady. Pro výpočet funkční hodnoty polynomu stupně

[3]Příspěvek byl vypracován s podporou grantu GAČR 406/05/P561.

tři tedy potřebujeme použít pětkrát operaci násobení a třikrát operaci sčítání. (Již zde postupujeme poměrně efektivně, když počítáme třetí mocninu čísla tři, již si pamatujeme, i hodnotu druhé mocniny, kterou jsme vypočítali v průběhu výpočtu.)

Tento výsledek je možné zobecnit pro výpočet funkční hodnoty v bodě polynomu stupně n. Je poměrně zřejmé, že budeme potřebovat $n-1$ násobení pro výpočet všech mocnin až do stupně n zadaného čísla. Potom n násobení na vynásobení vybrané mocniny příslušným koeficientem a nakonec n sčítání na sečtení všech členů polynomu dohromady.

Nyní je dobré položit si otázku, zda nemůžeme počítat efektivněji, tedy s využitím menšího počtu početních operací. Určitě můžeme v případě některých specialních polynomů použít menší počet operací. Typickým příkladem jsou polynomy s velkým počtem nulových (či jednotkových) koeficientů (např. pro výpočet funkční hodnoty polynomu $x^4 - 5$ si vystačíme s dvěma operacemi násobení a jedním sčítáním), nebo u polynomů, v nichž se koeficienty shodují s bodem, ve kterém počítáme funkční hodnotu (např. pro výpočet funkční hodnoty polynomu $x^3 - 3 \cdot x^2 + 3 \cdot x + 1$ v bodě 3 nám stačí použít pouze dvě násobení a tři sčítání, pokud si navíc uvědomíme, že $x^3 - 3 \cdot x^2$je v bodě tři nula bez výpočtu, stačí nám jedno násobení a jedno sčítání).

Tyto trikové případy představují pouze výjimky z obecného pravidla. Zkusme se zabývat otázkou, zda není možné hodnotu polynomu spočítat rychleji, tedy s použitím menšího počtu operací, i v případě, kdy pracujeme s obecným polynomem. Odpověď na tuto otázku je kladná, existuje způsob, jak hodnotu obecného polynomu spočítat rychleji, nazývá se Hornerovo schéma.

Myšlenka výpočtu podle Hornerova schématu je velice jednoduchá a budeme si ji demonstrovat na polynomu $5x^3 - 6x^2 + 4x - 5$, který již známe z předchozího výpočtu. Tento polynom můžeme vhodným přezávorkováním upravit do tvaru $((5 \cdot x - 6) \cdot x + 4) - -5$. Je naprosto zřejmé, že pro výpočet hodnoty polynomu (pokud je zapsán v tomto tvaru), stačí použít pouze třikrát operaci násobení a třikrát operaci dělení. Výpočet se obvykle zapisuje do následující tabulky:

	5	−6	4	−5
v bodě 3		$5 \cdot 3 = 15$	$9 \cdot 3 = 27$	$31 \cdot 3 = 93$
	5	$15 - 6 = 9$	$27 + 4 = 31$	$93 - 5 = \mathbf{88}$

Hodnota polynomu $5x^3 - 6x^2 + 4x - 5$ v bodě tři je tedy osmdesát osm.

V případě obecného polynomu stupně bychom potřebovali n-krát operaci násobení a n operací sčítání, což je lepší výsledek, než který dostáváme v případě obvyklého dosazování. Hornerovo schéma je tedy efektivnější než běžný postup, který při výpočet hodnoty polynomické funkce používáme. Hornerovo schéma zapsané ve tvaru tabulky je navíc přehlednější, obvykle se v něm v porovnání s dosazováním vyskytují menší čísla a dosažený výsledek má i další využití, neboť Hornerovo schéma lze zároveň i využít pro dělení polynomu lineárním členem.

Umocňování

V případě výpočtu v předchozím příkladu jsme v jednom případě pro výpočet funkční hodnoty v bodě polynomu stupně n potřebovali $2n - 1$ násobení a ve druhém případě pouze n násobení. V postupech byl tedy rozdíl, ale nebyl nijak drastický, neboť obě závislosti jsou lineární. Nyní se podívejme na další příklad, na kterém budeme demonstrovat, že rozdíl v počtu operací může být v některých případech mnohem větší.

Příklad. Spočítejte 2^{10}.
Nejjednodušší způsob, jak tuto úlohu vyřešit, je počítat přesně podle definice a číslo 2 devětkrát vynásobit samo se sebou $2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$. V tomto případě potřebujeme devět operací násobení. Pokud používáme tento postup, tak pro výpočet n-té mocniny potřebujeme $n - 1$ operací násobení.

Aktivita pro žáky

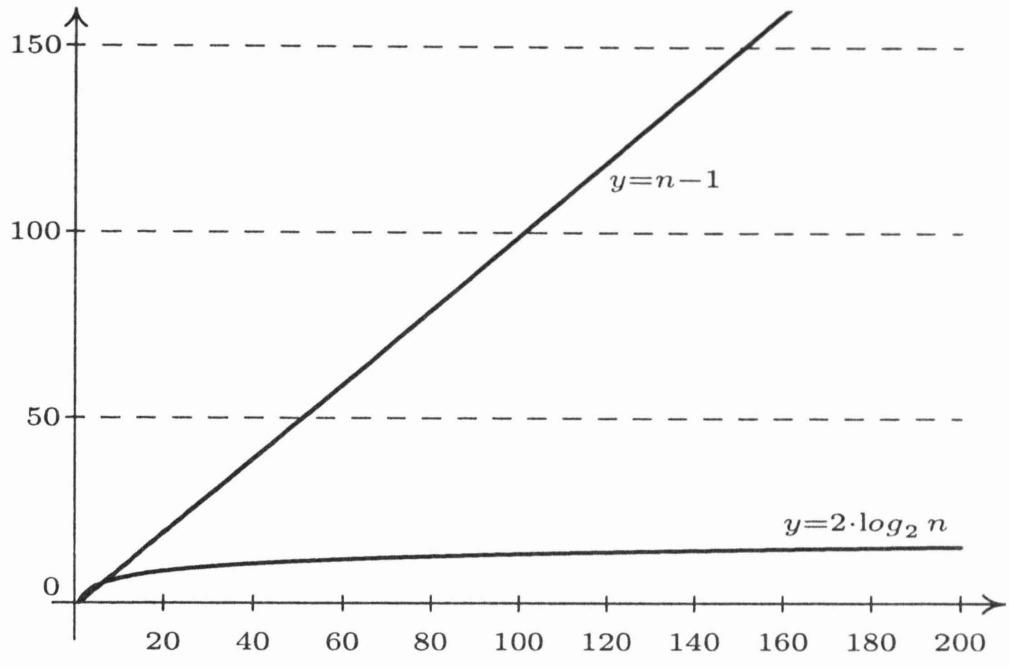
Můžete požádat žáky, aby se pokusili najít způsob, který bude rychlejší než standardní (výše uvedený) postup na pro výpočet

n-té mocniny, a nechat je soutěžit, kdo dokáže vypočítat zadanou mocninu s použitím menšího počtu násobení.

Jedním z nejrychlejší řešení je následující postup (zkuste jej srovnat s egyptským algoritmem násobení z předcházejícího dílu). Nejprve spočítáme druhou mocninu ($2 \cdot 2 = 4$), tu vynásobíme samu se sebou a dostaneme čtvrtou mocninu $4 \cdot 4 = 16$ a tu opět vynásobíme samu se sebou a dostaneme osmou mocninu ($16 \cdot 16 = 256$ – tento výpočet je možné provést na prstech, viz předchozí díl seriálu). Na závěr vynásobíme mezi sebou osmou a druhou mocninu čísla 2 a dostaneme desátou mocninu, tedy požadovaný výsledek ($256 \cdot 4 = 1\,024$). V celém výpočtu jsme použili pouze čtyřikrát operaci násobení. Zkusme se nyní zamyslet a odhadnou, kolik operací budeme potřebovat pro výpočet n-té mocniny zadaného čísla. Počet operací, které budeme potřebovat, velice úzce souvisí s tím, jak vypadá zápis čísla n ve dvojkové soustavě.

Úplně nejhorší situace může nastat v případě, kdy se číslo n ve dvojkové soustavě skládá ze samých jedniček. Naopak nejlepší je případ, kdy zápis čísla n ve dvojkové soustavě obsahuje pouze jednu jedničku a samé nuly. Například pokud potřebujeme spočítat $2^{1\,024}$, můžeme násobit pouze desetkrát. Tedy i pro výpočet tak obrovského čísla, jakým $2^{1\,024}$ bezesporu je, nám stačí pouze malý počet operací.

Nyní se ale vraťme k nejhoršímu možnému případu. Pokud se číslo n ve dvojkové soustavě skládá ze samých jedniček a jejich počet je k, budeme, dle našeho algoritmu, potřebovat $k-1$ násobení pro výpočet všech potřebných pomocných mocnin a $k-1$ násobení pro jejich vzájemné vynásobení. Celkem se tedy počet násobení pohybuje mezi $k-1$ v nejlepším případě a $2 \cdot (k-1)$ v nejhorším případě. Pro zapsání výsledku by bylo vhodné nalézt funkci, která pro zadané číslo vrací počet cifer při zápisu čísla ve dvojkové soustavě. Takovou funkcí je logaritmus při základu dva (definovaný vztahem $n = 2^{log_2 n}$), resp. jeho dolní celá část. Můžeme tedy říci, že pro výpočet n-té mocniny zadaného čísla potřebujeme nejvýše $2 \cdot log_2 n$ operací. Následující graf ukazuje srovnání obou funkcí. Z grafu je zřejmé, že zvláště pro větší je rozdíl v porovnání obou



algoritmů velmi výrazný.

Cena operací – Násobení zdvojováním

Při odhadování časové náročnosti algoritmu musíme někdy vzít v úvahu i rozdílnou časovou náročnost jednotlivých kroků. Je zřejmé, že vynásobit dvě stomístná čísla bude zřejmě složitější (a časově náročnější) než je pouze sečíst. Proto například algoritmus, který potřebuje pro výpočet n násobení, bude určitě složitější než algoritmus, který potřebuje pouze n sčítání. K přiblížení této skutečnosti dětem lze použít následující aktivitu, která volně navazuje na algoritmy násobení popsané v předchozím díle.

Aktivita pro žáky

Vypočítejte, kolik je 17 krát 23 pouze pomocí zdvojnásobování libovolného čísla, přičítání čísla 23 a sčítání dvou libovolných čísel dohromady. Každá z uvedených operací má různou cenu a vaším úkolem je provést výpočet co nejlevněji. Zdvojnásobování libovolného čísla stojí jeden zlatý, přičtení čísla 23 k libovolnému číslu stojí dva zlaté a sečtení dvou libovolných čísel stojí tři zlaté. Za-

daná čísla i cenu jednotlivých operací můžete libovolně měnit. V uvedeném případu by nejnižší cena měla být do 6 zlatých.

Faktorizace

U některých úloh se setkáme i s algoritmy, které jsou tak pomalé, že pro praktické použití jsou naprosto nepoužitelné. Jednou z praktických ukázek takového algoritmu je algoritmus na rozklad čísla na prvočísla, který používají žáci na základních školách. Zatímco vynásobit dvě celá čísla je úloha velice jednoduchá, tak úloha zjistit k zadanému číslu, zda lze napsat jako součin dvou celých čísel a tato čísla nalézt, je velice obtížné. Na obtížnosti hraničící s neřešitelností této úlohy je založeno zabezpečování informací na internetu. RSA algoritmus, který je používán k šifrování dat na internetu (např. bankovních transakcí), stojí a padá na tom, že neumíme zpětně najít, součinem jakých dvou velkých prvočísel zadané číslo vzniklo. Proces hledání dvou součinitelů (faktorů), z nichž vzešlo zadané číslo, se nazývá *faktorizace*. S otázkou faktorizace souvisí také rozklad čísel na prvočísla a ověření, zda zadané číslo je prvočíslo.

Žáci na základní škole se učí faktorizovat pomocí zkoušení všech prvočísel menších než zadané číslo (resp. jeho odmocnina). Zkusme si na praktické ukázce ukázat, jak je uvedený algoritmus neefektivní.

Předpokládejme, že chceme rozlomit šifru založenou na algoritmu RSA s délkou klíče 512 bitů. Máme tedy najít dva prvočíselné dělitele čísla, které má v binárním zápise 512 cifer. Stačí tedy prověřit všechna prvočísla od 2 do odmocniny ze zadaného čísla, celkem asi 10^{70} prvočísel. Předpokládejme, že uspějeme asi v desetině všech čísel, tedy pro prověření 10^{69}. Do výpočtu zapojíme všechny lidi na Zemi a každý člověk bude mít k dispozici 1 000 počítačů. Celkem na výpočtu bude pracovat asi 5 000 000 000 000 počítačů. Nyní předpokládejme, že všechny počítače budou super výkonné s procesorem o frekvenci 200 THz (to znamená, že za sekundu zvládnou provést 200 000 000 000 000 operací). Navíc předpokládejme, že každý z počítačů dokáže jedinou operací ověřit jedno prvočíslo. Za těchto předpokladů budete potřebovat pro dokončení výpočtu asi 31 623 153 207 852 661 404 573 973 miliard

let. Vzhledem k tomu, že tato doba mnohonásobně přesahu stáří vesmíru, je velice pravděpodobné, že se konce výpočtu nedočkáte.

Přesto již šifrovací klíč o délce 512 bitů není považován za bezpečný a v praxi došlo k jeho prolomení. Pro faktorizaci tak velkého čísla bylo použito velice sofistikovaných algoritmů, které hledají čísla x a y taková, že zbytek po dělení x^2 a y^2 faktorizovaným číslem je stejný. Tudíž zbytek $x^2 - y^2$ je po dělení nulový a můžeme použít vzorec $x^2 - y^2 = (x - y) \cdot (x + y)$, kde čísla $x - y$ a $x + y$ mají s velkou pravděpodobností s faktorizovaným číslem netriviální největší společný dělitel. Tento největší společný dělitel je hledaným faktorem šifrovacího klíče.

Praktická aplikace – Eukleidův algoritmus

Poslední algoritmus představovaný v tomto článku je Eukleidův algoritmus, který je datován do dob starého Řecka (*Eukleides* (asi 325–265 př.n.l)). Pomocí Eukleidova algoritmu je možné vypočítat velice rychle největší společný dělitel dvou čísel bez toho, abychom znali jejich rozklad na prvočísla.

Postup je velice jednoduchý, vezmeme obě čísla a větší z nich nahradíme zbytkem po dělení většího čísla menším (popřípadě rozdílem většího a menšího čísla). Celý postup opakujeme, dokud jedno z čísel není nula. Potom druhé z čísel je hledaný největší společný dělitel zadaných čísel.

Příklad

Nalezněte největší společný dělitel čísel 3 565 a 5 704.

3 565	5 704
3 565	2 139
1 426	2 139
1 426	713
0	713

Uvedený výpočet ukazuje, že největším společným dělitelem čísel 3 565 a 5 704 je číslo 713. Efektivita uvedeného algoritmu jej dovoluje užívat i pro velmi vysoká čísla.

Závěr

V matematice se setkáváme s velkým množství nejrůznějších algoritmů. Některé z nich jsou poměrně nové, jiné již jsou známé po celá staletí až tisíciletí. U každého algoritmu je rozumné se ptát nejen zda je algoritmus správný, to znamená jednoznačně zadaný se správným výsledkem, ale i jak je efektivní. Vědomí, že různé postupy jsou různě efektivní, že některé výpočty jsou natolik složité, že je nezvládnou ani nejmodernější počítače či představa o tom, jak efektivitu algoritmů porovnávat, by měly patřit mezi věci, se kterými se žáci seznamují již na základní a střední škole. Vzhledem k velice omezenému počtu hodin, které jsou věnovány informační výchově a jejímu všeobecnému zaměření na zvládnutí informační gramotnosti, patří tyto otázky především do hodin matematiky. Vhodným doplňkem učiva matematiky je i seznámení žáků s některými starými, přesto velmi efektivními algoritmy, jako je například v tomto článku uváděný Eukleidův algoritmu a Hornerovo schéma.

RNDr. Antonín Jančařík, Ph.D.
Katedra matematiky a didaktiky matematiky PdF UK
M. D. Rettigové 4
116 39 Praha 1
e-mail: antonin.jancarik@pedf.cuni.cz