Pavel Pudlák

O složitosti

Pokroky matematiky, fyziky a astronomie, Vol. 33 (1988), No. 1, 20--34

Persistent URL: http://dml.cz/dmlcz/139598

O složitosti

Pavel Pudlák, Praha

Tento článek vychází z mé části společné přednášky s Jaroslavem Morávkem, která se konala v květnu r. 1985 *na jedné ze schůzek pražské matematické obce. Chtěl bych jej věnovat památce Jaroslava Morávka, který byl jedním z prvních matematiků v Československu, který se teorií složitosti zabýval.*

1. Co je to složitost a proč se jí zabýváme?

Pojem složitosti, kterým se budeme v tomto článku zabývat, je blízký jeho významu v běžném jazyce. Dalo by se říci, že zkoumáme matematizace tohoto pojmu. Na rozdíl od některých jiných pojmů, které byly matematizovány, neexistuje jen jeden matematický pojem složitosti, ale celá řada možných definic vystihujících různé aspekty. Intuitivní pojem složitosti je svázán s představou množství informace obsažené v daném jevu. Není však zřejmé, jakým způsobem by se měla informace s jevem nebo objektem spojovat. Různé způsoby spojování dávají různé míry složitosti, tím je dána nejednoznačnost tohoto pojmu.

Se slovem „složitost" se v běžném životě setkáváme často. Říkáme dětem, že je něco příliš složité na to, aby to pochopily, říkáme, že mezinárodní situace je složitá, neúspěchy ospravedlňujeme složitostí vnějších podmínek apod. Naopak jednoduchost je obvykle positivní rys: líbí se nám symetrická uspořádání — protože jsou jednodušší, říkáme, že krása teorie tkví v její jednoduchosti a že geniální myšlenky jsou vždy jednoduché apod. Postihnout všechny tyto situace jistě není možné, zejména také proto, že je v nich lidský faktor. Mnohem lépe se studuje otázka, co je složité pro počítač? Zkoumání složitosti výpočtů na počítačích bylo právě hlavním nematematickým faktorem vzniku teorie složitosti. To však neznamená, že teorie složitosti se zabývá jenom složitostí v souvislosti s počítači. Řada zkoumaných tříd složitosti nemá s počítači nic společného a i ty typy složitosti, které souvisí s reálnými výpočty, jsou jen určitým přiblížením ke skutečnosti.

Otázky týkající se prostředků (strojového času, velikosti paměti apod.) nutných k řešení úloh, stručně řečeno optimalizace zpracování informace, jsou jistě převažující motivací pro zkoumání složitosti. Aktuálnost takového výzkumu v současné době je jistě nepopíratelná. Avšak cílem tohoto článku není zdůvodňovat význam teorie složitosti na základě její potenciální aplikovatelnosti. Myslím, že přitažlivost teorie složitosti pramení z toho že je to matematická disciplína, kde hlavní otázky mají kořeny v reálném světě, nicméně mají svůj matematický půvab. Teorie složitosti je i nový pohled na tradiční oblasti matematiky: nestačí nám pouhá existence, ale zkoumáme míry, podle nichž lze určit, jak dobře lze danou věc vyčíslit, zkonstruovat, definovat atd.

Nemá smysl se v krátkém článku pokoušet o souhrnný pohled. Místo toho se spíše zaměřím na příklady; myslím tím nejen příklady úloh, jejichž složitost chceme určit, ale také příklady používaných pojmů a zkoumaných problémů. Všechno bude velice elementární, protože chci, aby článek byl přístupný co největšímu okruhu čtenářů. Podrobnější informace je možno získat např. z přehledu S. Cooka [1], jehož překlad nedávno vyšel v tomto časopise.

První z příkladů je spíše historka. Jak uvádí Pratt [12], začátkem tohoto století F. Cole „strávil 3 roky nedělí" na to, aby dokázal, že číslo $2^{67} - 1$ je složené a vyvrátil tím dvěstě let starou Mersenovu domněnku. Nalezený důkaz je rovnost $2^{67} - 1 = 193707721 \times$ $\times$ 761838257287, kterou si každý může sám ověřit. Všimněme si na tomto příkladě několika věcí. Pro určení, zda nějaké číslo je prvočíslo, máme jednoduchý algoritmus: vyzkoušet všechny potenciální dělitele. Je to tedy úloha z klasického hlediska rozhodnutelná. Avšak pro číslo $2^{67} - 1$ tento algoritmus nemůžeme použít, protože i výkonný počítač by potřeboval příliš mnoho času na provedení všech $2^{67} - 3$ dělení*). Toto číslo není příliš veliké, má 21 cifer v desítkové soustavě a některé jeho vlastnosti lze ne příliš pracně ověřovat. Máme-li například nějakého jeho dělitele, můžeme provést dělení a přesvědčit se, že to je opravdu dělitel. Je tedy přirozené ptát se, které vlastnosti takto velkých čísel ještě jsme schopni rozhodovat a jak je to konkrétně s prvočíselností a hledáním dělitelů. Je třeba říci, že pro jednotlivá čísla můžeme využít nějaké jejich speciální vlastnosti, zde chceme mít algoritmus použitelný pro každé takto velké číslo.

Poznamenejme mimochodem, že pro rozhodování o tom, zda dané číslo je prvočíslo, jsou už dnes známy poměrně dobré algoritmy. Složitější je hledání vlastních dělitelů složených čísel, neboť pro tuto úlohu takové algoritmy nemáme.

Otázky týkající se dělitelů přirozených čísel, ač na první pohled vypadají velice teoreticky, mají pozoruhodné potenciální použití. Jde o tzv. veřejné kódovací klíče. Představme si, že A chce dostávat od B kódované zprávy, ale nemá možnost poslat B kódovací klíč tak, aby se ho nepřítel nemohl zmocnit. A tedy potřebuje kódovací klíč a k němu dekódovací klíč, a to takové, aby znalost kódovacího klíče neumožňovala snadno odvodit dekódovací klíč. (Dekódovací klíč samozřejmě A nepošle B.) Systém, který navrhli Rievst, Shamir a Adleman vypadá takto: A zvolí dvě dostatečně velká prvočísla p, q a spočítá $N = p \,.\, q$, $\Phi(N) = (p - 1) \,.\, (q - 1)$ a zvolí s, t, dvojici inverzních prvků modulo $\Phi(N)$. Jako kódovací klíč A sdělí B čísla N a s. B kóduje číslo $a < N$ tak, že určí $b \equiv a^s \bmod N$. Pro dekódovaní zprávy A spočítá b^t modulo N. Protože

$$st \equiv 1 \bmod \Phi(N)$$

a podle Eulerovy věty

$$x^{\Phi(N)} \equiv 1 \bmod N\,,$$

máme

$$b^t \equiv a^{st} \equiv a^{1+m.\Phi(N)} \equiv a \bmod N\,.$$

Pro umocňování a určování inverzního prvku modulo dané číslo známe jednoduché algoritmy. (Pro určení inverzního prvku je to známý Euklidův algoritmus.) Nutným

*) Při rychlosti jedno dělení za jednu mikrosekundu přes 4 milióny let; samozřejmě, že stačí zkoušet jen dělitele $\leqq \sqrt{(2^{67} - 1)}$, na to by bylo zapotřebí při stejné rychlosti asi 4,5 dne.

předpokladem pro to, aby nebylo možno zjistit dekódovací klíč, tj. t na základě znalosti N a s, je nemožnost nalezení dělitele N. Jakmile je totiž známo řekněme p, potom už snadno určíme postupně q, $\Phi(N)$ a t. Zdá se, že hledat dělitele pro dostatečně velká N je skutečně obtížné, ale ani tento nutný předpoklad není dokázaný.

Uvedený příklad patří spíše k okrajovým aplikacím. Většina úloh, se kterými se v teorii složitosti setkáváme a které mají potenciálně praktické uplatnění, je z oblasti kombinatorické optimalizace. Jde o úlohy plánování činnosti, rozmístění objektů apod., viz příklad v části o problému $P = NP$?

Schematicky můžeme studované situace nejčastěji popsat jako určení složitosti nějakých funkcí. Tento popis zahrnuje v sobě i případ, kdy zkoumáme složitost množin, protože množina je v podstatě totéž jako funkce s dvouhodnotovým oborem. Abychom terminologicky odlišili ty funkce, jejichž složitost zkoumáme, od jiných funkcí, budeme je nazývat *úlohami*. Úlohy, které jsou dvouhodnotové funkce, se nazývají *rozhodovací úlohy*. Např. uvedená úloha, určit zda dané číslo je prvočíslo, je rozhodovací úloha. Dva typické obory této úlohy jsou všechna 21ciferná čísla a všechna přirozená čísla. Zkoumání úloh, které nejsou rozhodovací, se dá obvykle převést na zkoumání rozhodovacích úloh. Např. úlohu o určení nejmenšího vlastního dělitele n můžeme nahradit úlohou o rozhodnutí o tom, zda n má vlastního dělitele menšího než dané m. V dalším se proto omezíme výhradně na rozhodovací úlohy.

Aby nadhozené otázky získaly matematický charakter, musíme definovat složitost, přesněji řečeno míry složitosti, a tomu se nyní budeme věnovat.

2. Kombinační složitost booleovských funkcí

Jednou ze základních měr složitosti je tzv. kombinační složitost booleovských funkcí. Pro ilustraci popíšeme tento pojem podrobněji a ukážeme také, čím je motivován.

Nejdříve co je to booleovská funkce? Booleovská funkce dimenze n je funkce s n argumenty $f(x_1, \ldots, x_n)$, přičemž argumenty stejně jako funkční hodnoty nabývají pouze hodnot 0 a 1, krátce tedy je to funkce $f\colon \{0, 1\}^n \to \{0, 1\}$. Proč zkoumáme právě takové funkce? Praktické důvody jsou nasnadě: při fyzikální realizaci 0 a 1 odpovídají přítomnosti a nepřítomnosti elektrického napětí, atd. Z teoretického hlediska f je prostě funkce z konečné množiny do $\{0, 1\}$, přičemž na definičním oboru máme strukturu Booleovy algebry. Struktura Booleovy algebry má řadu vlastností, ale zejména umožňuje efektivní kódování jiných struktur. Například je-li n tvaru $\binom{m}{2}$, můžeme zvolit vzájemně jednoznačné přiřazení mezi proměnnými $x_1, \ldots, x_n$ a hranami $e_1, \ldots, e_n$ úplného grafu na m-bodové množině. Potom vektor nul a jedniček $(a_1, \ldots, a_n)$ odpovídá grafu G na m-bodové množině, kde e_i je hrana G, právě když $a_i = 1$. V tomto příkladě můžeme definiční obor f ztotožnit s množinou všech grafů na m-bodové množině a o f mluvit jako o vlastnosti grafů.

Je zde ovšem další důvod pro volbu definičního oboru tvaru $\{0, 1\}^n$. Jde o to, že $\{0, 1\}^n$ je sice z klasického hlediska konečná množina, ale už při poměrně malých n nesmírně velká. S tím jsme se setkali v našem úvodním příkladu, který s použitím dvojkové sou-

stavy můžeme formulovat jako zkoumání booleovské funkce dimenze 67. Jednotlivé vstupy (posloupnosti délky 67) se dají snadno realizovat, ale celá funkce, např. její tabulka, je nerealizovatelná. To je právě situace, která je z hlediska teorie složitosti nejzajímavější.

Podívejme se nyní na definici kombinační složitosti. Nechť B je konečná množina booleovských funkcí; pro jednoduchost uvažujme množinu všech binárních booleovských funkcí $B = \{f \mid f: \{0, 1\}^2 \to \{0, 1\}\}$. B budeme nazývat bází a její prvky spojkami. Prvky B jsou například konjunkce a disjunkce, označované obvykle $\wedge$ a $\vee$. Skládáním funkcí

Obr. 1.

x_1	x_2	$x_1 \wedge x_2$	$x_1 \vee x_2$
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

z B můžeme vytvořit libovolnou booleovskou funkci (říkáme, že B je úplná báze). Funkce je tak složitá, zhruba řečeno, jak je obtížné ji vyjádřit skládáním funkcí z báze. Pro přesnější definici musíme popsat způsob, jak se funkce vytváří skládáním. Pro matematiky je nejpřirozenější uvažovat funkční schéma. Funkční schéma je posloupnost rovnic tvaru

$$\begin{aligned} y_1 &= g_1(u_1, v_1) \\ &\vdots \\ y_k &= g_k(u_k, v_k)\,, \end{aligned} \tag{1}$$

kde $g_1, \ldots g_k \in B$ a pro každé $i = 1, \ldots, k$ je

$$u_i, v_i \in \{x_1, \ldots, x_n, y_1, \ldots, y_{i-1}\}\,.$$

(Ti, kteří více inklinují k informatice, dávají přednost názvu lineární program.) Délka nebo složitost schématu je počet rovnic, tj. k. Funkce, kterou schéma určuje, je dána y_k; dostaneme ji explicitně, provedeme-li postupně všechna dosazení naznačená ve schématu. Protože výsledná formule může být dost velká, je výhodnější pro daný vstupní vektor $\boldsymbol{a} \in \{0, 1\}^n$ provádět výpočet přímo s nulami a jedničkami. Potom stačí provést jen k operací, které jsou skutečně elementární. Kombinační složitost nějaké booleovské funkce f je nejmenší k takové, že f lze definovat schématem délky k.

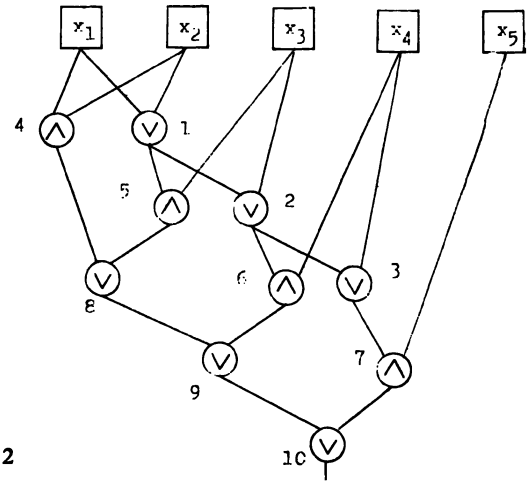
Abychom ukázali, že kombinační složitost vyjadřuje něco skutečně podstatného, uvažujme, jak vůbec může nějaké zařízení pro výpočet booleovské funkce vypadat. Přijměme předpoklad, že zařízení je sestaveno z konečně mnoha elementů tak, že

(1) je nejvýše s stavů, ve kterých se jednotlivé elementy mohou vyskytovat,
(2) zařízení pracuje v diskrétních časových okamžicích a stav elementu v čase $t + 1$ závisí jen na stavu elemetnů v čase t, se kterými je spojen;
(3) každý element je spojen s nejvýše d dalšími elementy (a struktura propojení se nemění);

$$
\begin{aligned}
y_1 &= x_1 \vee x_2 \\
y_2 &= y_1 \vee x_3 \\
y_3 &= y_2 \vee x_4 \\
y_4 &= x_1 \wedge x_2 \\
y_5 &= y_1 \wedge x_3 \\
y_6 &= y_2 \wedge x_4 \\
y_7 &= y_3 \wedge x_5 \\
y_8 &= y_4 \vee y_5 \\
y_9 &= y_8 \vee y_6 \\
y_{10} &= y_9 \vee y_7
\end{aligned}
$$

$$y_{10} = f(x_1, \ldots, x_5) = 1 \leftrightarrow x_1 + \ldots + x_5 \geqq 2$$

Obr. 2.



Obr. 3. Booleovský obvod pro schéma z obr. 2.

(4) ve výchozím okamžiku jsou určené elementy nastaveny podle vstupního vektoru, výstupní hodnota je určena stavem dalšího určeného elementu (tyto stavy se musí lišit od stavu elementu, s nímž zařízení začíná).

Nechť nějaké takové zařízení počítá booleovskou funkci f, nechť C je počet elementů tohoto zařízení a T je počet časových okamžiků nutných pro výpočet f. Ukážeme, že kombinační složitost f se dá odhadnout pomocí veličin C a T. Abychom úvahu ještě více zjednodušili, předpokládejme, že $s = d = 2$ a že stav elementu v čase $t + 1$ závisí jen na stavu jeho sousedů v čase t, ale ne na jeho stavu v čase t. Potom dostaneme, že kombinační složitost funkce f je menší nebo rovna $C \,.\, T$. K tomu stačí najít jedno funkční schéma pro f, které má počet rovnic $\leqq C \,.\, T$. Schéma definujeme takto: Pro každou dvojici (e, t), kde e je element zařízení, t časový okamžik, $1 \leqq t \leqq T$, zvolíme jednu proměnnou y. Proměnné uspořádáme do posloupnosti tak, aby pro každé y odpovídající (e, t) a y' odpovídající (e', t'), y bylo před y', pokud $t < t'$. Rovnice ve schématu budou popisovat závislost stavu elementu na předcházejícím stavu elementů s ním spojených. Protože $s = 2$, můžeme stavy označovat nulou a jedničkou; závislost bude tedy určena booleovskou funkcí. Protože $d = 2$, bude tato funkce binární. Je to tedy funkční schéma v souladu s naší definicí.

V obecném případě, kdy elementy mají více stavů, jsou spojeny s více dalšími elementy a stav elementu závisí i na jeho předchozím stavu, lze sestrojit schéma s délkou $\alpha \,.\, C \,.\, T$, kde konstantu α lze odhadnout pomocí s a d. Předpoklady o zařízení, které jsme nahoře uvažovali, byly velice obecné. To znamená, že pokud se podaří získat dostatečně velké odhady pro kombinační složitost nějaké funkce, má to určitou absolutní platnost. Snadno se dá ukázat, že většina booleovských funkcí dimenze n má kombinační složitost blízkou $2^n/n$. (Tento důkaz je však nekonstruktivní, nedává nám žádnou konkrétní funkci s touto složitostí.) Funkce $2^n/n$ nabývá také „astronomické" hodnoty už pro

malá n. Je tu tedy potenciální možnost ukázat veliký dolní odhad na $C.T$ pro funkce s malým počtem argumentů a tím i jejich fyzikální nerealizovatelnost.

Kombinační složitost se dá definovat také pomocí tzv. booleovských obvodů. Z teoretického hlediska je rozdíl mezi obvody a schémata zcela nepodstatný. Booleovský obvod se z funkčního schématu (1) sestrojí takto: Na množině $\{x_1, \ldots, x_n, y_1, \ldots, y_k\}$ definujeme orientovaný graf tak, že $z \to t$ právě, když $t = y_i$ a $(z = u_i$ nebo $z = v_i)$ pro nějaké $i \leqq k$. K vrcholům y_i, $i = 1, \ldots, k$ připíšeme spojky g_i. Nyní je asi jasné, proč používáme termín obvod. Booleovský obvod je návod, jak realizovat funkci elektronickými prvky. Booleovský obvod je však dosti nerealistický model skutečného obvodu, zejména pokud jde o vysoce integrované obvody. Zanedbává řadu hledisek, která je nutno při konstrukci skutečných obvodů brát v úvahu. Jsou to např. rozmístění prvků na ploše (popř. v prostoru), prostor zaujímaný vodiči, čas nutný pro předání signálu po vodiči, synchronizace elementů a další. Samozřejmě, že modely, které berou zřetel na tato hlediska, byly už také studovány.

Kombinační složitost je jedna z mnoha druhů složitostních měr. Pro jiné míry můžeme odvodit podobné vztahy a je možno je také použít k důkazu toho, že určité funkce se nedají realizovat. Proč tedy studujeme kombinační složitost a ne jiné míry, které jsou blíže k realitě? Abychom na to odpověděli, musíme se zamyslet nad tím, co je naším cílem. Pokud je cílem praktická otázka: odhadnout počet prvků konstruovaného počítače, musíme nutně volit realističtější model. Pokud naším cílem je teoretičtější otázka: oddělit funkce, které ze zásadních důvodů nelze počítat od těch, které by se daly potenciálně počítat při zdokonalení technologie počítačů, potom není vhodné používat příliš speciální model. Model, který zahrnuje mnoho detailů, generuje také mnoho nedůležitých otázek, jež nás odvádějí od podstaty věci. Odhady získané pro takové modely ztrácejí význam, jakmile se změní technologie. Při teoretickém přístupu jde o to stanovit určitou hranici mezi tím, co je složité a co je jednoduché. Je-li tato hranice vhodně zvolena, je rozlišení na jednoduché a složité do značné míry invariantní vzhledem k volbě modelů. To je další důvod, proč dáváme přednost modelu jednoduššímu.

Je nutno přiznat, že teorie složitosti zatím nevyřešila své základní problémy. Konkrétně v případě kombinační složitosti situace vypadá takto: O určitých booleovských funkcích se domníváme, že jsou složité. Abychom to dokázali, potřebujeme najít velké dolní odhady na jejich složitost, např. dolní odhady větší řádově než n^k pro každé k, kde n je počet argumentů funkce. Avšak řádově největší známý dolní odhad pro jakoukoli konkrétní funkci je jenom $3n - o(n)$. Jsou sice známy exponenciální dolní odhady kombinační složitosti pro rozhodnutelnost některých logických teorií, ale to jsou příklady poněkud umělé.

3. Turingovy stroje a další modely výpočtů

Další přístup ke zkoumání složitosti je založen na pojmu Turingova stroje a jeho různých modifikacích. O Turingových strojích se zde musíme zmínit alespoň krátce, protože mají přinejmenším stejný význam jako booleovská schémata a obvody.

Na rozdíl od „booleovské složitosti", která zkoumá konečné funkce, „turingovská složitost" zkoumá spočetné, obecně nekonečné množiny. Objekt, jehož složitost měříme, je zde jazyk nad konečnou abecedou. Jazyk nad abecedou A je libovolná podmnožina množiny A^* všech konečných posloupností prvků z A. Prvky množiny A^* se nazývají slova v abecedě A. Místo podmnožin A^* bychom mohli brát funkce $F : A^* \to \{0, 1\}$, avšak z tradičních důvodů, asi proto, že terminologie pochází z matematické lingvistiky, uvažujeme obvykle spíše podmnožiny. Všimněme si, že $A = A^0 \cup A^1 \cup A^2 \cup \ldots$, tedy $F = F_0 \cup F_1 \cup F_2 \cup \ldots$, kde $F_i : A^i \to \{0, 1\}$. Speciálně, je-li $A = \{0, 1\}$, můžeme F chápat jako posloupnost booleovských funkcí. To naznačuje, že je určitá souvislost mezi turingovskou a booleovskou složitostí. Abychom nezabíhali příliš do detailů, nebudeme tuto důležitou souvislost podrobněji rozvádět.

Původní definice Turingova stroje vychází z představy mechanického zařízení. Toto zařízení má pásku rozdělenou na políčka a automat, který pásku ovládá. Automat je vybaven hlavou, kterou čte symbol napsaný na jednim políčku a pomocí které může symbol na tomto políčku přepsat na jiný. Kromě toho automat může v jednom kroku výpočtu posunout pásku o jedno políčko doprava nebo doleva. Zatímco automat je konečný — má konečně mnoho možných stavů, páska je nekonečná na obě strany. Nakonec ještě předpokládáme, že automat má dva vyznačené stavy — počáteční a koncový. Turingovým strojem rozumíme celé zařízení, tj. včetně pásky.

Nechť je dáno slovo v abecedě, jejíž symboly je Turingův stroj schopen číst. Potom můžeme slovo napsat na pásku, ostatní políčka ponechat prázdná, nastavit pásku tak, aby hlava byla na prvním písmenu slova, dát automat do počátečního stavu a spustit. První věc, která nás bude zajímat je, zda se automat vůbec dostane do koncového stavu. Jestliže ano, potom řekneme, že stroj slovo přijímá. Tímto způsobem stroj určuje jazyk: jazyk všech slov v dané abecedě, která přijímá. Aby stroj mohl přijímat složitější jazyky, povolíme mu, aby při výpočtu na pásce používal větší abecedu, než je abeceda jazyka, který má přijímat.

Je jistě fascinující, že takto primitivní zařízení je schopno provádět libovolný algoritmus. (Jistý profesor dokonce nutil každého svého nového asistenta, aby vyrobil mechanický, později elektronický model Turingova stroje. Vznikla tak zajímavá sbírka dokumentující vývoj technologie zpracování informace.) Dnes, kdy programování pomalu patří k základnímu vzdělání, bychom spíše Turingův stroj definovali jako jednoduchý programovací jazyk, např. takto:

- $P(i)$ je lineární pole, v němž je zapsáno zpracovávané slovo z A, pracovní abeceda je $B \supseteq A$;
- $i = 0$ na začátku programu,
- instrukce jsou průběžně číslovány, přípustné instrukce jsou:

  $i := i + 1$,
  $i := i - 1$,
  $P(i) := b$, (pro všechna $b \in B$),
  if $P(i) = b$ **then go to** N, (kde $b \in B$ a N je číslo instrukce),
  stop.

Tento popis je o krok blíže k přesné matematické definici; není těžké podat formální definici i pro automatovou verzi, ale pro naše účely je to zbytečné.

Stejně jako v případě booleovských obvodů, při definici Turingova stroje nejde o co nejvěrnější popis toho, jak se dají mechanicky realizovat algoritmy. Cílem je zachytit nějakým způsobem určitou představu o zpracování informace. Tato představa je zhruba založena na tom, že

(1) informaci zpracováváme pomocí konečné struktury v konečně mnoha diskrétních krocích;

(2) v každém kroku podle přesně určeného předpisu změníme strukturu lokálně;

(3) velikost této lokální změny je omezena konstantou — nezávisí na velikosti vstupního slova.

V případě Turingova stroje je konečnou strukturou v (1) automat a část pásky, která byla dosud použita, lokální změnou v (2) a (3) je přepsání symbolu na pásce, posuv hlavy a změna stavu stroje.

Pro definice složitostních měr je lepší uvažovat Turingovy stroje, které se zastaví při práci na každém slově. Budeme tedy definici modifikovat tak, že stroj má dva koncové stavy — přijímající a zamítající, a je sestrojen tak, že se *vždy* zastaví po konečném počtu kroků v jednom z těchto stavů. V případě našeho miniaturního programovacího jazyka je potřeba instrukci **stop** nahradit dvěma a za správné považovat jen ty programy, které nutně skončí v jedné z nich. Modifikujeme také přijímání: stroj přijme slovo, pokud se zastaví v přijímacím stavu. Výpočet na daném slovu má dvě základní složitostní charakteristiky: počet kroků stroje, než se zastaví, a počet použitých políček na pásce. Odtud jsou odvozeny dvě míry složitosti jazyka — časová a prostorová. Definice je zde trochu složitější než u booleovských funkcí, protože nemůžeme definovat přímo, že složitost jazyka L je číslo nebo funkce. Místo toho definujeme jen vztah, např.: L lze počítat v čase $t(n)$ (kde t je funkce na přirozených číslech), jestliže existuje Turingův stroj, který přijímá L a pro každé n a každé slovo w délky n se při zpracování slova w zastaví po nejvýše $t(n)$ krocích.

Samozřejmě, že se zkoumá řada modifikací a příbuzných modelů, které zachycují určité aspekty výpočtů lépe než Turingovy stroje. V poslední době je velká pozornost věnována zejména modelům, které umožňují studovat složitost paralelních výpočtů. Paralelismus je v klasickém modelu Turingova stroje zcela potlačen.

Pro dokreslení pestrosti studovaných modelů výpočtů a složitostních měr je potřeba uvést ještě alespoň jeden model, který nelze zařadit ani do turingovské ani do booleovské složitosti. Začneme s úlohou. Nechť je dán vektor reálných čísel $(\alpha_0, \alpha_1, \ldots, \alpha_n, \alpha_{n+1})$. Máme určit, zda všechna čísla $\alpha_1, \ldots, \alpha_n$ leží v intervalu $\langle \alpha_0, \alpha_{n+1} \rangle$. Aniž bychom dále upřesňovali, jakým způsobem se to má určit, většina lidí by asi navrhla provést $2n$ srovnání $\alpha_0 \lesseqgtr \alpha_i$, $\alpha_i \lesseqgtr \alpha_{n+1}$, $i = 1, \ldots, n$. Připustíme-li, že můžeme srovnávat α_0 a α_{n+1} s α_i, pak jistě bychom měli povolit srovnávat libovolná dvě α_i, α_j. Potom ale existuje kratší způsob, jak řešit úlohu:

(1) nejdříve srovnáme α_1 s α_2, α_3 s α_4 atd. až α_{m-1} s α_m, kde $m = 2\rceil n/2 \lceil$ (tj. m je největší sudé číslo menší nebo rovné n);

(2) menší číslo v každé z těchto m dvojic srovnáme s α_0, větší s α_{n+1};

(3) pokud je n liché (tj. $m < n$), srovnáme ještě α_n s α_0 a α_{n+1}.

Tímto způsobem stačí na řešení úlohy jen $[\frac{3}{2}n]$ srovnání.*) Abychom dokázali, že menší počet srovnání nestačí (což je pravda), museli bychom definovat model výpočtu. Nicméně otázka nutného počtu srovnání je natolik intuitivně názorná, že se do této definice zde nemusíme pouštět.

Podívejme se raději na to, v čem se tato složitost podstatně liší od předchozích. Základní odlišnost je v tom, že povolujeme jedinou elementární operaci – srovnání dvou čísel, a tato operace je velice speciální. V tomto modelu můžeme řešit jen některé úlohy a používat jen některé algoritmy. Například kdybychom tutéž úlohu řešili pomocí Turingova stroje, např. pro přirozená čísla zadaná v desítkové soustavě, potom můžeme uvažovat algoritmy, které postupují úplně jiným způsobem. Přirozeně by bylo třeba hledat mezi $\alpha_1, \ldots, \alpha_n$ čísla s nejdelším a nejkratším zápisem. Takový postup se nedá formulovat jen pomocí srovnání α_i s α_j. I přes určitou omezenost má zkoumání nutného počtu srovnání a podobné úlohy své oprávnění. Můžeme si přestavit situaci, kdy vektor $(\alpha_0, \ldots, \alpha_{n+1})$ nám není zadán explicitně, ale máme možnost provést sérii testů, které nám určí, zda α_i je $<$ nebo $=$ nebo $>$ než α_j pro $i, j = 0, \ldots, n+1$. Jiná možná interpretace je, že hledáme algoritmy, které jsou použitelné na libovolné lineárně uspořádané množině.

Nakonec ještě uveďme geometrickou interpretaci této úlohy: určete, zda vektor $(\alpha_1, \ldots, \alpha_n)$ leží v krychli $\langle \alpha_0, \alpha_{n+1} \rangle^n$. Mnohé další zajímavé úlohy se dají interpretovat jako úlohy o náležení bodu do sjednocení konvexních polyedrických množin. Místo pouhého srovnávání dvou čísel, což lze napsat jako určení znaménka lineární funkce $x - y$ pro α_i a α_j, je přirozené uvažovat znaménka libovolných lineárních funkcí. V této oblasti byl Jaroslav Morávek průkopníkem a dosáhl řady fundamentálních výsledků, viz také jeho knihu [9].

4. Jak stanovit hranici mezi složitým a jednoduchým a třídy složitosti

Pro kombinační složitost by bylo možno stanovit jedno přirozené číslo jako hranici mezi jednoduchými a složitými booleovskými funkcemi. Booleovské funkce, které by měly kombinační složitost menší než toto číslo, bychom prohlásili za jednoduché a ty, které by měly složitost větší za složité. Jak ale určit toto číslo? Nemůžeme vycházet z toho, čeho jsou schopny současné počítače, neboť technologie se velice rychle mění. Mohli bychom se pokusit o hledání absolutních fyzikálních mezí, ale názory na velikost vesmíru, znalosti o elementárních částicích atd. se také mění. Místo toho se nabízí jiný přístup. Většina úloh, se kterými se při výpočtech setkáváme, má parametr, který můžeme interpretovat jako velikost nebo dimenze vstupních dat. Například v úlohách o grafech – velikost nosné množiny, při řešení soustav rovnic – počet rovnic, počet proměnných apod. Jestliže takovou úlohu modelujeme pomocí booleovských funkcí, znamená to obvykle, že nemáme jedinou funkci, ale nějakou posloupnost funkcí $f_1: \{0,1\}^{n_1} \to \{0,1\}$, $f_2: \{0,1\}^{n_2} \to \{0,1\}$, ..., kde $n_1 < n_2 < \ldots$. V tomto modelu je přirozené položit velikost vstupu rovnou příslušnému n_i, tj. délce vstupního vektoru.

*) $]x[$ je označení pro celistvou část x, $[x]$ označuje nejmenší celé číslo větší nebo rovné x.

Složitost takovéto parametrické úlohy pak není jedno číslo, ale posloupnost přirozených čísel čili funkce na přirozených číslech. Pokud studujeme turingovskou složitost, nemusíme úvahy uvedené nahoře vůbec provádět, protože turingovská složitost se měří funkcemi.

Otázku hranice mezi složitým a jednoduchým tak převádíme na otázku klasifikace funkcí. Jednoduché bude odpovídat pomalu rostoucím funkcím, složité rychle rostoucím funkcím. Vědomě se tím vzdalujeme od původního zdroje otázky – při skutečných výpočtech nás vždy budou zajímat jen konečné počáteční úseky těchto funkcí a ne jejich asymptotické chování. Skýtá nám to ovšem větší možnosti, protože množina funkcí má bohatší strukturu než samotná přirozená čísla.

Ukončeme tedy obecné úvahy a podívejme se, jak se to např. řeší v případě Turingových strojů. Postulujeme, že jazyk L je jednoduchý, pokud existuje polynom $p(n)$, takový, že L lze počítat v čase $p(n)$, (tj. existuje Turingův stroj, který přijímá jazyk L a na každém vstupu délky n se zastaví po $\leqq p(n)$ krocích). Obvyklá terminologie nepoužívá slova „jednoduchý" a „složitý". Místo toho se uvedenou podmínkou definuje třída jazyků P a říkáme, že L patří, resp. nepatří, do P; někdy též říkáme jenom, že L lze (nelze) počítat v polynomiálním čase.

Tato definice se v praxi až překvapivě osvědčila. Pokud se o nějaké konkrétní úloze dokázalo, že se dá počítat v polynomiálním čase, potom se také obvykle dříve nebo později dokázalo, že příslušný polynom je kvadratický nebo že lze úlohu řešit ještě rychleji. (Přesněji řečeno, to platí pro poněkud silnější verzi Turingových strojů, kterou zde nebudeme uvádět.) Dá se tedy říci, že úlohy z praxe, o kterých bylo dokázáno, že jsou v P, se dají skutečně prakticky řešit pro malé vstupní hodnoty.

Hlavní důvod, proč se tento pojem tak rychle rozšířil, je však jiný. Třída P nám totiž umožňuje zbavit se závislosti na pojmech, které jsou až příliš konkrétní, jako je třeba Turingův stroj. Při prvním setkání s definicí časové složitosti pomocí Turingova stroje nás možná trochu zarazilo, že jedna ze základních otázek je zkoumání počtu kroků, které vykoná tak zvláštní zařízení, jako je Turingův stroj. Když už tedy počítáme počet kroků, měli bychom vzít nějaký „přirozenější" nebo „kanoničtější" pojem algoritmu. Byla navržena řada alternativních definic a těžko lze posoudit, zda některé z nich jsou z tohoto hlediska podstatně lepší než Turingův stroj. Důležité je, že většina těchto pojmů je ekvivalentní Turingovu stroji v tom smyslu, že příslušné časové složitosti se neliší více než polynomiálně. V důsledku toho můžeme v definici třídy P použít kterýkoli z těchto pojmů. Tím se stává mnohem více opodstatněné zkoumání nejen třídy P, ale vůbec časové turingovské složitosti. Dále také můžeme v souvislosti s třídou P mluvit o polynomiálním počtu elementárních kroků, aniž bychom příliš pedantně ověřovali jejich elementárnost. Místo elementárních kroků můžeme používat i podprogramy pro úlohy, které jsou řešitelné v polynomiálním čase.

V mnoha případech lze řešitelnost úlohy v polynomiálním čase nahlédnout snadno tím, že se spočítá počet kroků ve známém algoritmu pro tuto úlohu. Vezměme například úlohu: pro daná přirozená čísla m a n určit, zda m dělí n. K důkazu polynomiálnosti stačí vzít algoritmus písemného dělení přirozených čísel. (Tento algoritmus je velice jednoduchý v dvojkové soustavě.) Snadno se nahlédne, že při dělení dvou nanejvýš k ciferných čísel je potřeba napsat nanejvýš const. k^2 cifer. Při simulování tohoto algo-

ritmu Turingovým strojem bychom potřebovali asi více kroků, kvůli přenosu informace mezi různými částmi pásky, ale polynomiálnost by nebylo těžké dokázat. V některých případech je cesta k důkazu řešitelnosti úlohy v polynomiálním čase spletitější. Příkladem takové úlohy je úloha lineárního programování. Řešitelnost této úlohy v polynomiálním čase byla dlouho otevřeným problémem, viz [7].

Vraťme se ještě na chvíli k našemu úvodnímu příkladu: určit, zda dané číslo n je prvočíslo. Zde je důležité stanovit, co je velikost vstupu pro dané n. Pokud bychom definovali velikost vstupu pro n rovnou n, potom triviální algoritmus založený na dělení n všemi čísly mezi 2 a $]\sqrt{n}[$ včetně by dokazoval, že úloha je řešitelná v polynomiálním čase. Přirozenější (a zajímavější) je brát jako velikost $\log n$, protože s použitím např. dvojkové soustavy můžeme kódovat číslo n posloupností (tj. slovem) z nul a jedniček o délce řádově $\log n$. Zcela formálně by to znamenalo určit složitost jazyka L v abecedě $\{0, 1\}$, kde $w \in L$ právě, když w je dvojkovým zápisem prvočísla. Algoritmus, který probírá čísla $2, \ldots,]\sqrt{n}[$ není v tomto případě polynomiální, ale je exponenciální — pro vstup délky k potřebuje vykonat 2^k dělení (navíc jedno dělení není elementární operací). Zda existuje algoritmus (Turingův stroj), který rozhodne v polynomiálně mnoha krocích (v závislosti na $\log n$) zda n je prvočíslo, je dosud nevyřešený problém. Řada výsledků však nasvědčuje tomu, že odpověď na tuto otázku bude kladná. Např. podle [4] k tomu stačí, aby mezi každým dostatečně velkým n a $n + \log^2 n$ existovalo prvočíslo.

Kromě P bylo definováno ještě mnoho dalších tříd složitostí. Základní problémy teorie složitosti se dají formulovat jako rovnosti nebo inkluze určitých tříd. O nejdůležitějším z těchto problémů se zmíníme dále.

5. Problém $P = NP$?

Jak už jsme uvedli, nemáme zatím dostatečně silné metody pro důkazy dolních odhadů složitosti. Proto pro mnoho zajímavých i prakticky důležitých úloh není znám jejich statut z hlediska náležení do P; až na výjimky jsme totiž zatím schopni pouze dokázat, že úloha patří do P, ale nejsme s to dokázat, že nepatří. Ukazuje se, že většina takových úloh patří do třídy, která je určitým způsobem blízká ke třídě P. Opět začneme s příklady:

(1) Pro dané přirozené číslo n určete, zda n je složené číslo (tj. že není prvočíslo; z důvodů, které vyplynou dále, formulujeme náš úvodní příklad tímto způsobem).

(2) Pro daný graf G a přirozené číslo k určete, zda G má kliku velikosti k (tj. existuje podmnožina vrcholů grafu mohutnosti k taková, že každé její dva body jsou spojeny hranou v G).

(3) Plánování pracovních úkonů: je dána konečná množina — pracovní úkony, každému pracovnímu úkonu je přiřazen (a) interval na přirozených číslech — termíny, ve kterých je nutno úkon provést, (b) přirozené číslo — délka pracovního úkonu. Otázka zní, zda je možno naplánovat pracovní úkony v souladu s termíny tak, aby se žádné dva nepřekrývaly (tj. zda je může vykonat jeden pracovník).

To, co mají tyto tři úlohy společného z hlediska teorie složitosti, je něco jako princip Kolumbova vejce: je-li řešení předloženo, je snadné se přesvědčit, že to je správné řešení.

Poněkud přesněji, v případě kladné odpovědi pro daný vstup je možno předložit něco jako důkaz, přičemž fakt, že je to důkaz pro daný vstup se dá ověřit snadno; zejména nesmí být důkaz příliš dlouhý. Jakým způsobem se tento „důkaz" získá, je nepodstatné. V případě záporné odpovědi nepožadujeme nic. V našich příkladech tyto „důkazy" jsou: vlastní dělitel čísla n, k-bodová klika v grafu G, plán úkonů. Kdybychom však příklad (2) modifikovali tak, že bychom požadovali, aby k bylo klikové číslo grafu G, tj. mohutnost největší kliky v G, pak bychom tuto vlastnost pravděpodobně ztratili. Jedna maximální klika o velikosti k nedokazuje, že klikové číslo G není větší než k; kdybychom např. chtěli doplnit tento důkaz všemi $k + 1$ indukovanými podgrafy, narazili bychom na problém, že je jich příliš mnoho.

Úlohy tohoto typu tvoří třídu NP. Problém je tedy, zda $NP \subseteq P$, což lze formulovat také jako $P = NP$, neboť NP je definováno tak, že $P \subseteq NP$. Název NP je zkratka od „nedeterministicky polynomiální"; „nedeterministický" pochází od jistého zobecnění Turingových strojů. Třídu NP lze však definovat, aniž bychom toto zobecnění zaváděli. Nechť $|w|$ označuje délku slova w, nechť wu označuje zřetězení slov w a u. Potom NP je třída jazyků definovaná podmínkou: $L \in NP$, jestliže existuje jazyk $L' \in P$ a polynom p tak, že

$$L = \{w \mid \exists u(|u| \leqq p(|w|) \,\&\, wu \in L')\} .$$

Pro dané w je u splňující hořejší podmínku to, čemu jsme neformálně říkali důkaz. Tedy požadujeme, aby množina dvojic vstup-důkaz patřila do P. Podmínka $|u| \leqq p(|w|)$ nám zaručuje, že fakt, že u je důkaz pro w, se dá ověřit v polynomiálně mnoha krocích v *závislosti na délce* w.

Důležitou podtřídu NP tvoří tzv. NP-úplné úlohy. Jsou to úlohy, které jsou v určitém smyslu nejsložitější ze všech NP úloh. V důsledku toho platí, že $P = NP$, jakmile jedna NP-úplná úloha je v P. Metody důkazu, že nějaká úloha je NP-úplná, jsou značně propracované. Garey a Johnson [3] uvádějí přes 300 NP-úplných úloh. Pro většinu známých úloh z NP se dá dokázat buď, že jsou v P, nebo že jsou NP-úplné. Z našich tří příkladů (2) a (3) jsou NP-úplné, (1) patří zatím k výjimkám, ale pravděpodobně patří do P. Vzhledem k tomu, že většina matematiků věří, že $P \neq NP$, je důkaz NP-úplnosti náhražkou za důkaz složitosti úlohy.

Otázka, zda $P = NP$ patří mezi nejdůležitější současné matematické problémy. Někteří matematici ho řadí dokonce před známou Riemannovu hypotézu. Samozřejmě mezi $P = NP$? a Riemannovou hypotézou je velký rozdíl. Teorie čísel je jedna z nejrozvinutějších teorií, je tedy velice málo pravděpodobné, že by řešení Riemannovy hypotézy bylo jednoduché, což o $P = NP$? nelze s určitostí říci. Problém, zda $P = NP$, byl poprvé formulován Cookem v roce 1971 [2]. Je to tedy poměrně mladý problém, ale přesto si už získal jméno. Je to asi tím, že se s ním setkává skoro každý, kdo navrhuje efektivní algoritmy, zejména se často vyskytuje při aplikaci kombinatorických metod. Příklad (3) je typickou ukázkou optimalizační úlohy, jejíž řešitelnost závisí na tom, zda $P = NP$. S tímto problémem se setkáváme i v teoretických oblastech — vlastně skoro vždy, když nás zajímá složitost zkoumaných pojmů; kromě teorie grafů a kombinatoriky je to zejména logika a teorie čísel.

Zajímavá je souvislost problému $P = NP$? se starými metodologickými otázkami o vztahu konstrukce a existence v matematice. Jde o otázky typu, zda jsou oprávněné ryze existenční důkazy, tj. důkazy, kterými se existence matematického objektu dokáže, aniž by tento objekt byl zkonstruován. V konečné kombinatorice je řada vět, která mají takové důkazy, třebaže konstrukce nejsou známy. Jsou to důkazy založené na počítání objektů. Existenci dokážeme např. tak, že ukážeme, že objektů, které nemají požadovanou vlastnost, je méně, než je počet všech objektů. Kdyby bylo $P = NP$, existovaly by pro většinu těchto vět také příslušné konstrukce. V PMFA se touto tematikou zabýval článek J. Nešetřila [10].

$P = NP$? je nejdůležitějším velkým problémem teorie složitosti, ne však jediným. Další základní problémy se týkají vztahu času k prostoru pro Turingovy stroje (připomeňme, že prostorem rozumíme počet políček pásky použitých v průběhu výpočtu, odpovídá to tedy velikosti paměti počítače) a vztahu kombinační složitosti k některým dalším mírám složitosti booleovských funkcí. Otevřený problém je také např., zda NP je uzavřeno na doplňky, tj. zda doplněk jazyka z NP je také v NP.

6. Historie a perspektivy teorie složitosti

Teorie složitosti se zabývá kvantitativními otázkami výpočtů a konstrukcí – kolik kroků algoritmus potřebuje, jak musí být velký obvod pro danou funkci apod. V historii matematiky těmto kvantitativním otázkám předcházely otázky rázu spíše kvalitativního – zda se dá určitými konstrukcemi daný objekt sestrojit, zda je možno danou úlohu řešit určitými početními metodami apod. Důkazy nemožnosti určitých konstrukcí a výpočtů patří k nejslavnějším výsledkům v matematice (např. nemožnost trisekce úhlu pravítkem a kružítkem, neřešitelnost rovnic pátého stupně radikály). Blíže ke kvantitativnímu pojetí jsou různé klasifikace zkoumaných objektů, se kterými se setkáváme v matematice skoro všude. V analýze třídíme funkce podle toho, zda jsou spojité, mají spojitou n-tou derivaci atd. V topologii máme množiny otevřené, uzavřené, F_σ, G_δ borelovské atd. Celé jedno odvětví topologie, deskriptivní teorie množin, se takovými klasifikacemi zabývá. V některých případech je souvislost s teorií složitosti tak těsná, že se dá využít (viz [16]).

Jeden ze základních pojmů z teorie složitosti je pojem algoritmu. Intuitivní pojem algoritmu je velice starý (název „algoritmus" vznikl zkomolením jména arabského matematika Al Chvárizmího z 9. století). Formální definice byly podány teprve ve třicátých letech našeho století (Turing, Kleene, Church, Post). Pro teorii složitosti má největší význam Turingova formalizace tohoto pojmu, kterou jsme popsali v části 4. Už v této době bylo zřejmé, že některé algoritmy jsou prakticky neproveditelné již pro malé vstupní hodnoty, protože vyžadují příliš velký počet kroků (např. známá Ackermannova funkce). V padesátých letech se objevují práce popisující hierarchie rekursívních (= algoritmicky vyčíslitelných) funkcí (Grzegorczyk). Hlavní rozmach nastává až v polovitě šedesátých let v souvislosti se studiem časové a prostorové složitosti na Turingových strojích.

Téměř paralelně se rozvíjel výzkum matematických modelů z počátku releových a později elektronických obvodů. Dnes tyto modely nazýváme booleovskými obvody.

První práce Shannona byly publikovány koncem třicátých let. Shannona už od začátku zajímala kvantitativní stránka věci. Později se tento směr nejvíce rozvíjel v Moskvě. Vztah mezi booleovskou a turingovskou složitostí byl zřejmě objeven teprve na přelomu šedesátých a sedmdesátých let.

Dalším mezníkem je článek S. Cooka [2] z roku 1971, kde jsou definovány třídy P, NP a pojem NP-úplnosti. Pojem NP-úplnosti byl pak modifikován R. Karpem [5] do podoby, která se dnes převážně používá. Tento pojem umožnil dokázat ekvivalenci řady problémů v konečné kombinatorice a teorii grafů, které byly zkoumány nezávisle. Tím se rozšířil okruh matematiků, kteří se zajímají o otázky složitosti. Otázka, zda $P = NP$, se stala centrálním problémem teorie složitosti a získala respekt i mimo tento obor.

Zavedení složitostních měr pozitivně stimulovalo navrhování nových algoritmů pro konkrétní úlohy, neboť bylo možné srovnávat efektivnost algoritmů. Ze dvou algoritmů je lepší ten, který vyžaduje méně kroků (= čas), případně menší paměť (= prostor). Samozřejmě, že je to teoretické měřítko a při implementaci může být někdy výhodnější algoritmus s horšími parametry. Ve většině případů však další zdokonalení algoritmů nebo nutnost vyšetřovat větší vstupní data dá za pravdu tomuto hledisku.

Navrhování algoritmů, obvodů apod. pro konkrétní úlohy patří spíše k aplikacím. Na rozdíl od toho důkazy, že algoritmy s určitými parametry neexistují, jsou jistě výsledky teoretické. Můžeme to také vyjádřit pomocí pojmů *horní odhady* a *dolní odhady*. Parametry konkrétního algoritmu jsou horním odhadem pro složitost úlohy. Důkaz složitosti dané úlohy je ekvivalentní nalezení dolních odhadů pro parametry všech algoritmů, které danou úlohu řeší. Například pro důkaz $P = NP$ by stačilo najít polynomiální algoritmus pro jednu NP-úplnou úlohu, přičemž by to mělo značný praktický dosah. Abychom dokázali, že $P \neq NP$, je potřeba najít více než polynomiální dolní odhad na časovou složitost nějaké úlohy ze třídy NP; praktický význam by byl jen ten, že by se potvrdilo to, čemu většina matematiků věří.

Důkazy dolních odhadů patří k nejzajímavější, ale také nejobtížnější části teorie složitosti. Zdá se, že místo přímého řešení problémů, jako je $P = NP$?, je lepší se zaměřit na hledání jakýchkoli technik, které dávají dolní odhady složitosti. Protože nás potom nezajímá, pro jaké úlohy dostáváme dolní odhady, je úspěšnost nějaké metody dolních odhadů měřena jen velikostí dosaženého dolního odhadu. Je tedy možno zaznamenávat „rekordy" pro jednotlivé míry složitosti, popř. třídy úloh. Existence takového objektivního měřítka hodnoty dosažených výsledků je jistě ku prospěchu věci.

Pro zajímavost uvádím tabulku rekordů pro jeden typ složitosti, kde byly v poslední době rekordy překonány několikrát (jde o velikost obvodů s hloubkou k).

1981	Furst, Saxe, Sipser	$c_k \cdot n^{\log^{3(k-2)} n}$
1983	Ajtai	$n^{c'_k \log n}$
1984	Yao	$2^{n^{c''_k}}$
1985	Hastad	$2^{c'''_k \cdot n^{1/(k-1)}}$

($c_k, \ldots, c'''_k$ jsou konstanty >0, $\log^i$ označuje i-krát iterovaný logaritmus).
Většinou pokrok nejde tak přímočaře kupředu, nicméně rekordy byly překonány v poslední době i v dalších oblastech.

Pro další vývoj lze předpokládat, že aplikovaná část, tj. zejména horní odhady složitosti konkrétních úloh, se bude vyvíjet podobným způsobem jako dosud. Zajímavější je budoucnost oblasti dolních odhadů. Zde je těžké cokoli předpovědět. Je možné, že řešení budou jednoduchá, ale je i možné, že bude potřeba nejdříve vypracovat nový aparát. Možná, že bude stačit jeden nápad a bude možno vyřešit všechny problémy; ale je také možné, že bude potřeba pro každý problém něco jiného. Jsou i úvahy o nezávislosti některých tvrzení teorie složitosti vzhledem k používaným formálním teoriím. Ale i kdyby to byla pravda, v současné době nemáme v matematické logice potřebný aparát pro důkaz takových nezávislostí.

Teorie složitosti je ještě mladým oborem: neexistují zde rozsáhlé monografie, hranice oboru nejsou zcela vyjasněny, základní problémy jsou dosud otevřené a jediný článek může svým významem znehodnotit značnou část předchozích výsledků. (Není proto také překvapivé, že řada zásadních objevů byla v poslední době učiněna právě mladými matematiky; např. sovětský matematik A. A. Razborov, který je v současné době pouze vědeckým aspirantem, byl v r. 1986 díky dvěma článkům v tomto oboru pozván na Mezinárodní matematický kongres k přednesení referátu.) Bude tedy jistě zajímavé i nadále sledovat vývoj teorie složitosti.

Literatura

[1] Cook, S. A.: *Prehľad teórie výpočtovej zložitosti.* PMFA *32* (1987), 12—29.

[2] Cook, S. A.: *The complexity of theorem proving procedures.* 3rd Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio, 1971, 151—158.

[3] Garey, M. R., Johnson, D. S.: *Computers and Intractability — A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco 1979. (též ruský překlad)

[4] Goldwasser, S., Kilian, J.: *Almost all primes can be quickly certified.* Proceedings of the 18-th Annual ACM Symp. on Theory of Computing, Berkeley 1986, 316—329.

[5] Karp, R. M.: *Reducibility among combinatorial problems.* Ve sborníku Symposium on the Complexity of Computer Computations, editoři R. E. Miller a J. W. Thatcher. Plenum Press, New York 1972, 85—103.

[6] Kučera, L.: *Kombinatorické algoritmy.* SNTL, Praha 1983.

[7] Lawler, E. L.: Velký matematický sputnik roku 1979. PMFA *27* (1982), 39—47.

[8] Manders, K. L.: *Computational complexity of decision problems in elementary number theory.* Ve sborníku Model Theory of Algebra and Arithmetic, editoři L. Pacholski. J. Wierzejewski a A. J. Wilkie. Springer-Verlag, Berlin—Heidelberg—New York, 211—227.

[9] Morávek, J.: *Složitost výpočtů a optimální algoritmy.* Academia, Praha 1984.

[10] Nešetřil, J. *Kombinatorické konstrukce, jejich složitost a praktický význam,* PMFA *23* (1978), 16—27.

[11] Nigmatullin, R. G.: *Složnost' bulevych funkcij.* Izdatelstvo Kazanskovo universiteta, Kazaň 1983.

[12] Pratt, V.: *Every prime has a succint certificate.* SIAM J. Comput. *4*, 1975, 214—220.

[13] Razborov, A. A.: *Nižnyje ocenki monotonnoj složnosti nekotorych bulevych funkcij.* Doklady AN SSSR *281/4*/(1985), 798—801.

[14] Razborov, A. A.: *Nižnyje ocenki monotonnoj složnosti logičeskogo permanenta.* Matematičeskije zametki *37/6* (1985), 887—900.

[15] Savage, J. E.: *The Complexity of Computing.* John Wiley & Sons, New York—London—Sydney, 1976.

[16] Sipser, M.: *A topological view of some problems in complexity theory.* Proceedings, Mathematical Foundations of Computer Science '84, Praha. Springer-Verlag, Berlin—Heidelberg—New York—Tokyo 1984, 567—572.