Radek Pelánek

Programátorská cvičebnice

Algoritmy v příkladech

computer press

Radek Pelánek

Programátorská cvičebnice

Computer Press
Brno
2012

Programátorská cvičebnice

Radek Pelánek

Obálka: Martin Sodomka
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-3751-2

Vydalo nakladatelství Computer Press v Brně roku 2012 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 16 440.

1. vydání

ALBATROS MEDIA a.s.

Obsah

Předmluva

Tato kniha vychází z mých zkušeností s programováním v několika rolích. V roli studenta jsem za svůj „programátorský život“ prošel mnoha programovacími jazyky. Jako nejlepší způsob učení programovacího jazyka mi vždy přišla metoda „skočit do vody a plavat“, tedy vzít si zajímavý, přiměřeně náročný problém, ten zkusit vyřešit a potřebné věci se učit za běhu. Často mi přišlo náročnější vymyslet si zajímavé zadání, než najít řešení konkrétních technických problémů, na které jsem pak při řešení narazil.

Jako učitel mám podobnou zkušenost. S dílčími technickými problémy si většinou studenti zvládnou poradit, důležité je především předložit jim vhodný problém – problém, který pro ně bude adekvátně obtížný a současně bude nějakým způsobem zajímavý, takže je bude bavit. Podobně je to i v programátorských soutěžích, kterých jsem se mnohokrát účastnil v roli soutěžícího i organizátora. Zajímavou soutěž dělají dobře vybrané problémy.

Ve všech těchto rolích jsem se setkal s celou řadou zajímavých příkladů, ale vždy, když nějaký potřebuji, musím složitě vzpomínat nebo hledat. Dobrých učebnic programování existuje spousta, ale většinou kladou důraz na konkrétní jazyk nebo detailní rozbor dílčích algoritmů, příkladů v nich bývá jen pár. Rozsáhlá sbírka zajímavých příkladů mi vždy chyběla. Proto jsem se pokusil ji vytvořit a pevně věřím, že bude užitečná nejen pro mě.

Kniha přímo či nepřímo čerpá z velkého množství zdrojů a zkušeností. Kromě těch literárních, které jsou uvedeny v seznamu literatury, jde hlavně o programátorské soutěže a výuku. Velkou měrou čerpám ze zkušeností s programátorskými soutěžemi a z příkladů, které jsem na těchto soutěžích poznal či pro ně navrhl. Konkrétně jde o soutěž ACM ICPC a její česko-slovenskou verzi CTU Open, programátorskou olympiádu, korespondenční semináře z programování a fakultní soutěž FIbot.

Cenné zkušenosti jsem získal při výuce programátorských předmětů na Fakultě informatiky MU, především pak při vedení předmětu IV104 Seminář řešení programátorských úloh. Na studentech tohoto předmětu jsem mnoho

zde uvedených příkladů testoval a jejich pozorováním jsem získal zkušenosti o obtížnosti a pedagogické vhodnosti jednotlivých úloh.

Tato kniha také částečně vychází k mé předchozí knihy „Jak to vyřešit?“, která se zabývá logickými úlohami a jejich využitím při výuce informatiky. Část příkladů uvedených zde v kapitolách o logických úlohách a hrách vychází z této dřívější knihy.

Chtěl bych tedy poděkovat všem, kdo organizovali uvedené soutěže, účastnili se mé výuky či pomáhali s přípravou předchozí knihy, za jejich přínos pro tuto knihu, byť nevědomý a nepřímý. Konkrétně bych chtěl poděkovat Petrovi Jaruškovi a Janovi Ryglovi za komentáře k dílčím cvičením. Martin Herodek z nakladatelství Computer Press pomohl knihu vylepšit po obsahové i typografické stránce. Poděkování také patří mojí ženě Barče za její vytrvalou podporu nejen při psaní.

Radek Pelánek

1 O knize

Mysl není nádoba, kterou je potřeba naplnit, ale oheň, který je potřeba zapálit.

Plutarchos

Aby se člověk naučil programovat, musí se naučit základní příkazy a postupy (trochu mysl *naplnit*), především však potřebuje hodně trénovat, zkoušet a procvičovat. Aby u toho trénování člověk vydržel, měl by být *zapálený*. Jenže na čem trénovat programování, aby to bylo zajímavé? Na programování rozsáhlých, prakticky užitečných programů začátečník ještě nemá a učebnicové příklady jsou často nudné, takže u nich člověk nevydrží.

K tomu právě slouží tato kniha – poskytuje náměty na zajímavá programátorská cvičení, sloužící k procvičení a tréninku. Příklady jsou voleny tak, aby byly atraktivní a zajímavé, takže člověka, který má alespoň trochu informatického ducha a nadšení, vtáhnou natolik, že si vydrží s nimi hrát, díky čemuž dobře potrénuje.

Kniha není zaměřena na žádný specifický programovací jazyk. Všechny příklady jsou zvládnutelné ve všech běžně používaných programovacích jazycích, a to většinou s docela základními prvky jazyka, tj. bez použití pokročilých vlastností jazyka a speciálních knihoven. Předpokládá se, že čtenář má k dispozici učebnici konkrétního programovacího jazyka, který používá.

Kniha slouží nejen k procvičení základních programátorských konstrukcí, ale i k procvičení algoritmů. I po této stránce je kniha především cvičebnicí a nikoliv samonosnou učebnicí. Klíčové pojmy jsou v knize stručně popsány a vysvětleny, nicméně tento popis je míněn jako připomenutí pojmů, které již čtenář slyšel, resp. inspirace pro to, co by si měl dostudovat. Opět se předpokládá, že čtenář má k dispozici učebnici algoritmizace (např. Skiena, 1998; Wroblewski, 2004; Cormen et al., 2001; Kučera, 2009) nebo prošel relevantním odborným kurzem. Pokrytá látka většinou spadá do učiva probíraného v 1. ročníku na informatických vysokých školách.

Kniha obsahuje širokou škálu příkladů od velmi jednoduchých až po myšlenkově i programátorsky náročné projekty. Také dílčí úlohy obsahují podúlohy

s různou složitostí. Čtenář tedy vždy může najít příklad, který odpovídá jeho momentálním schopnostem, náladě, časovým možnostem a tomu, co si chce procvičit. Průběžně, jak se bude zlepšovat, zde navíc najde stále nové výzvy.

Kniha přijde vhod několika skupinám čtenářům:

- studentům, kteří se učí programovat a psát efektivní algoritmy, pro samostudium a trénink,
- učitelům, kteří vyučují programování a hledají náměty na cvičení, zadání domácích úloh a projektů,
- zkušeným programátorům, kteří se učí nový jazyk a chtějí si jej procvičit na příkladech, případně si chtějí procvičit své „programátorské svaly“ na zajímavých příkladech.

1.1 Úlohy a jejich popisky

Problémy, které stojí za útok, prokážou svoji cenu tím, že útok opětují.

P. Erdős

Úlohy jsou rozděleny do několika tematických oblastí, které sdílejí základní rysy. Pro každou úlohu je uveden odhad obtížnosti, stručný styl úlohy, popis zadání, doplňující komentář a částečně poznámky k řešením.

Kniha obsahuje velmi rozmanité typy úloh. Některé jsou přímočaré, v zadání je celkem jasně popsáno, co dělat, stačí to „jen“ implementovat. Jiné jsou nápadové – výsledný program je velmi krátký a nevyžaduje nic složitého, ale je potřeba vhodný nápad, na který nemusí být snadné přijít. U jiných příkladů se zase naopak hodí netriviální teorie či pojmy, ale ty jsou stručně vysvětleny, takže jde hlavně o to popis pochopit, dohledat si v případě potřeby detailnější vysvětlení a zvládnout popis převést do funkčního programu. Základní styl úlohy je vždy stručně shrnut na začátku popisu úlohy.

Kromě slovního popisu stylu je pro snadnější orientaci uvedeno i číselné hodnocení obtížnosti úloh. Toto hodnocení obtížnosti je rozděleno na dva aspekty: *nápad*, tedy jak náročné je přijít na hlavní myšlenku řešení, algoritmus a vhodnou reprezentaci dat, a *kódování*, tedy jak náročné je program napsat a odladit.

Obě kategorie jsou pouze přibližné odhady a slouží především k relativnímu porovnání úloh. Absolutní obtížnost pochopitelně závisí na zkušenostech řešitele. Obě obtížnosti jsou hodnoceny na stupnici 1–5. Význam jednotlivých stupňů pro začátečníka a experta je ilustrován v tabulce 1.1. U většiny úloh je pro nápad i kódování uveden interval, protože úlohy typicky obsahují několik podúloh, jejichž obtížnost se liší. Podúlohy jsou většinou uvedeny v pořadí rostoucí obtížnosti.

Tabulka 1.1: Kategorie obtížnosti

Nápad	Začátečník	Expert
1	vcelku jasné	zřejmé
2	trocha rozmýšlení	rutina
3	těžké	trocha rozmýšlení
4	hranice možností	těžké
5	neřešitelné	hranice možností

Kódování	Začátečník	Expert
1	20 min.	5 min.
2	1 hod.	15 min.
3	půl dne	1 hod.
4	několik dní	2–6 hod.
5	půlroční projekt	intenzivní víkend

Dále pak následuje samotný popis úlohy. *Zadání* je základní popis cílů úlohy. Po přečtení tohoto textu je možné začít řešit, v případě použití ve výuce je tento text určený pro studenty. Následuje *komentář* k úloze, který obsahuje souvislosti a základní rady, jak k problému přistupovat. Komentáře občas prozrazují i základní myšlenky vhodných algoritmů či skryté „finty“, jak k zadání přistoupit. Ambiciózní čtenář by tedy měl úlohu zkusit vyřešit bez čtení komentářů.

Na konci knihy jsou uvedena vybraná *řešení*, která detailněji osvětlují zajímavé body z postupu, ukazují příklady možných programů, případně odpovědi na konkrétní otázky položené v zadání. Jde však opravdu pouze o *vybraná* řešení, rozhodně nejsou rozebrány všechny příklady a varianty. To není žádoucí, některé problémy jsou záměrně ponechány otevřené pro vlastní zkoumání a projekty.

V řešeních jsou uvedeny ukázky kódu v jazyce Python. Jazyk Python byl pro ukázky zvolen proto, že jde o čistý vysokoúrovňový jazyk, který bývá občas označován za „spustitelný pseudokód“. Ukázky kódu by tedy měly být pochopitelné i bez znalosti tohoto konkrétního jazyka. Python je programovací jazyk, který se rozhodně vyplatí naučit, nicméně pro porozumění této knize není jeho znalost nijak zásadní.

1.2 Jak knihu používat

Buď to udělej, nebo ne. Neexistuje žádné „zkusím“.

Yoda ve filmu Impérium vrací úder

Knihu lze využívat různými způsoby: k samostudiu, ve výuce ke krátkým cvičením, k zadávání domácích úloh nebo i rozsáhlých projektů. Následuje několik komentářů a doporučení ke každému stylu.

V případě *samostudia* je důležitá pevná vůle – nespokojit se pouze s jednoduchými příklady a nedívat se hned na řešení. Může být vhodné najít si partnera, se kterým budete příklady řešit souběžně. Můžete tak soutěžit, kdo zvládne úlohy vyřešit rychleji či efektivněji, a také můžete svá řešení porovnávat. U mnoha úloh existuje několik různých řešení a porovnáním více různých řešení se můžete hodně naučit.

U použití ve *výuce*, především pokud programování probíhá v hodině, která má fixní konec, je důležité vybrat příklady správné obtížnosti. Především je důležité, aby obtížnost nebyla příliš vysoká. Pokud studenti v půlce hodiny zjistí, že je nad jejich síly dokončit příklad v dostupném čase, jsou demotivováni, už se příliš nesoustředí a tím pádem i nic moc nenaučí. Přímo ve vyučovací hodině je lepší používat základní varianty příkladů a složitější varianty a rozšíření nechat jako domácí úkol, u kterého není tak krátký časový limit.

U domácích úloh je dnes samozřejmě problém s vyhledáváním na Internetu. Do knihy jsou začleněny úlohy zajímavé a osvědčené, což ovšem znamená, že jde často také o úlohy známé a rozšířené. Řešení mnoha úloh lze tedy často vcelku snadno najít na Internetu (především pokud hledáme anglické verze pojmů). Pokud jsme si tohoto problému vědomi, lze mu vcelku rozumně předcházet. Snadno vyhledatelná řešení většinou nejsou přesně v té podobě, jak je zde zadáno, a pokud navíc úlohu trochu pozměníme či přejmenujeme, často se dá snadnému vyhledání zabránit.

Navíc schopnost „vyhledat si základ řešení a to pak jen vhodně upravit“ není sama o sobě nijak špatná, naopak u praktických aplikací programování to je preferovaný postup. I tuto schopnost může být užitečné trénovat, takže u některých příkladů může být rozumné použití tohoto přístupu podporovat. Konkrétně například u cvičení „Experimenty s řadicími algoritmy“ studenty podporuji v tom, aby si řadicí algoritmy vyhledali a pouze převedli do jednotného stylu a aby se soustředili na experimentální porovnání.

Při použití příkladů jako zadání projektů využívejte zde uvedené zadání pouze jako výchozí bod. Řešitel projektu by měl zkusit vymyslet vlastní rozšíření a variace a zkusit si sám položit zajímavou otázku o úloze.

1.3 Scénáře použití

Čím tvrději pracuji, tím víc se zdá, že mám štěstí.

T. Jefferson

Formát knihy vynucuje lineární řazení příkladů, ovšem možných kritérií, podle kterých lze příklady řadit, je celá řada: například obtížnost, metoda řešení, téma, typ vstupu a výstupu. V této knize je zvoleno tematické řazení příkladů. Příklady, které jsou zařazeny za sebou, tedy sdílejí podobné téma, ale mohou se výrazně lišit v ostatních aspektech, především v obtížnosti. To znamená, že není dobrý nápad řešit příklady v knize čistě sekvenčně.

Pro výběr vhodného příkladu slouží informace uvedené na začátku popisu každého příkladu a dále pak tato a následující kapitola, které udávají přehled příkladů roztříděných podle různých kritérií. V této kapitole jsou příklady roztříděny podle stylu použití, v následující pak podle metod návrhu algoritmů, které se při řešení používají.

Zde uvedené seznamy berte jen jako základní orientaci. To, že příklad není uveden v určité kategorii, ještě neznamená, že by nemohl být pro dané účely vhodný.

Úplní začátečníci

Začneme příklady vhodnými pro úvodní kurz programování, tedy pro ty, kdo neumí vůbec programovat. Příklady jsou řazeny v pořadí, v jakém je vhodné je procházet:

- 4.2 Želví grafika: základy – pokud používáte jazyk, který má snadno použitelnou knihovnu pro interpretaci příkazů želví grafiky (např. Python), je toto cvičení nenáročné a přitom efektní (pomocí pár příkazů jdou kreslit zajímavé obrázky), takže jde o vhodné první cvičení.
- Příklady z kapitoly Počítání s čísly, konkrétně: 3.1 Hrátky s čísly, 3.2 Výpisy posloupností, 3.3 Collatzův problém, 3.4 Náhodná procházka, 3.5 Dělitelnost a prvočísla – u těchto příkladů vystačíme pouze s jednoduchými syntaktickými prvky jazyka (celočíselné proměnné a cykly), příklady jsou také vhodné pro představení konceptu funkce.
- 3.7 Fibonacciho posloupnost – představení jednorozměrného pole, ukázka různých pohledů na problém.
- 4.1 Textová grafika – opět problémy využívající jen základní syntaktické konstrukce, jen více obrázkové a občas vyžadující mírný nápad.
- 5.2 Transpoziční šifry, 5.3 Substituce a kódování – vhodné pro představení řetězců a práce s nimi.

- 7.2 Hádání čísla, 7.3 Oběšenec, 7.5 Hra Nim – vhodné pro představení načítání vstupu od uživatele (případně ze souboru) a vyzkoušení interakce s uživatelem. Příklady také ilustrují zajímavé a přitom vcelku snadno zvládnutelné algoritmy.
- 4.3 Želví grafika: fraktály, 6.3 Hanojské věže – představení konceptu rekurze.
- 6.1 Číselné bludiště, 6.5 Hledání cest v bludišti – práce s dvojrozměrným polem, představení základních grafových algoritmů (prohledávání do šířky).
- 8.4 Experimenty s řadicími algoritmy – práce s polem, ilustrace klasických algoritmů.

Příklady z kapitoly 3 Počítání s čísly jsou realizovatelné třeba i v tabulkovém editoru a mohou tak posloužit jako základní úvod do algoritmizace i v případě, kdy není ve výuce čas na probrání obecného programovacího jazyka.

Trénink algoritmizace

Následující příklady přijdou vhod, pokud se chceme zaměřit primárně na trénink algoritmů a metod návrhu algoritmů a chceme příklady, které jsou zvládnutelné při dobrém nápadu relativně rychle a pomocí krátkého kódu:

- 3.10 Permutace, kombinace, variace
- 4.7 Konvexní obal
- 4.8 Triangulace
- 5.5 Přesmyčky
- 6.3 Hanojské věže
- 6.6 Generování bludišť
- 7.7 Jednorozměrné piškvorky
- 8.1 Rozměňování mincí
- 8.3 Problémy s řetězci a posloupnostmi
- 8.4 Experimenty s řadicími algoritmy

Podrobnější rozbor různých metod návrhu algoritmů je uveden v následující kapitole.

Učení nového jazyka

Pro ty, kdo už umí programovat, pouze se učí nový programovací jazyk a chtějí jej natrénovat na zajímavých příkladech, se mohou hodit následující příklady:

- 3.9 Výpočet Pí – zajímavé experimenty, které přitom vyžadují jen manipulaci s číselnými proměnnými (vhodné pro první seznámení s jazykem).

- 6.1 Číselné bludiště – jednoduchá logická úloha pro procvičení práce s dvojrozměrným polem.
- 5.2 Transpoziční šifry, 5.3 Substituce a kódování – procvičení základní práce s řetězci a poli.
- 7.6 Tetris (interaktivní verze) – komplexní procvičení mnoha aspektů jazyka (interakce s uživatelem, reprezentace dat, čas), přičemž celkově je program docela krátký a výsledek relativně efektní.

Výzvy pro zkušené programátory

Pro zkušené programátory, kteří se nechtějí učit nový jazyk nebo algoritmus, ale chtějí výzvu svých schopností, zkusit si něco „pro radost z programování“ nebo potrénovat na programátorskou soutěž, se mohou hodit následující příklady.

- Výše uvedené příklady na trénink algoritmizace.
- Příklady z kapitoly 8 Klasické informatické problémy, a to nejlépe bez čtení doprovodného komentáře.
- Úlohy 3.10 Permutace, kombinace, variace, 4.3 Želví grafika: fraktály, 4.4 Sierpińského fraktál. Tyto úlohy mají krátké elegantní řešení, ale není úplně snadné je vymyslet.
- Hry a logické úlohy, konkrétně např. 6.11 Sokoban, 7.6 Tetris, 7.8 Piškvorky. Základní verze her v textovém režimu lze napsat docela rychle, také heuristiky pro umělou inteligenci mohou být s dobrým nápadem krátké, poskytují prostor pro soutěžení o to, kdo napíše nejúspěšnější program, a také dávají široký prostor pro rozšíření.
- 5.4 Rozlomení šifer – tato úloha obsahuje otevřenou a zajímavou výzvu napsat program tak, aby byl co nejúspěšnější v lámání šifer.

Zkušení programátoři si pak také mohou úlohy rozšířit tím, že překročí rámec psaní klasických sekvenčních programů využitím paralelizace u výpočetně náročných problémů, např. 6.11 Sokoban, 6.8 Jak navštívit všechna pole mřížky? nebo 4.6 Mandelbrotova množina s velkou přesností. Můžete zkusit co nejefektivněji využít vícejádrové procesory nebo provést výpočet v distribuovaném prostředí. Jiným způsobem rozšíření zadání může být realizace úloh formou interaktivní webové aplikace nebo aplikace pro mobilní telefon – pro tento styl jsou vhodné především hry a logické úlohy, ale zajímavé mohou být i některé geometrické problémy a úlohy s obrázky.

Projekty

Jako témata semestrálních (ročníkových) projektů, případně i jako inspirace pro témata bakalářských či diplomových prací, se mohou hodit následující příklady:

- Šifrovací systém – kombinace témat z kapitoly 5 Šifrování a práce s textem. Program dostane text a bude ho umět zašifrovat a dešifrovat různými způsoby.
- Úloha 6.6 Generování bludišť a generování zadání u dalších logických úloh.
- Složitější logické úlohy a hry: 6.9 Polyomina, 6.10 Sudoku, 6.11 Sokoban, 7.6 Tetris, 7.8 Piškvorky.
- 7.9 Souboje virtuálních robotů.
- 8.6 Grafové algoritmy.
- Problémy s experimentální složkou, jako jsou 8.4 Experimenty s řadicími algoritmy.
- Náměty z kapitoly 9 Další náměty.

2 Přehled pojmů

Do průšvihu nás nikdy nedostane to, co nevíme. Dostane nás tam to, co víme příliš jistě, a ono to tak prostě není.

Y. Berry

Tato kniha slouží jako cvičebnice, nikoliv jako kompletní učebnice. U jednotlivých příkladů jsou většinou hlavní myšlenky vysvětleny, předpokládá se však, že čtenář umí programovat v nějakém programovacím jazyce a zná základní pojmy z informatiky. Pro základní přehled pozadí, na kterém kniha staví, slouží tato kapitola, která podává stručný přehled pojmů použitých v knize. Rozhodně není nezbytné všechny uvedené pojmy ovládat, některé z nich se využijí pouze v několika málo těžších příkladech.

Pro každý pojem je dán stručný popis, který slouží primárně pro připomenutí, nikoliv pro vysvětlení. Pokud se čtenář s některým z pojmů nesetkal, je vhodné si před řešením příkladů souvisejících s daným pojmem téma blíže dostudovat. Výklad zmíněných pojmů lze najít v mnoha učebnicích, viz např. Skiena, 1998; Wroblewski, 2004; Cormen et al., 2001; Kučera, 2009. U mnoha základních pojmů nabízí dobré vysvětlení i Wikipedie.

Ke každému pojmu je uveden i seznam příkladů, ve kterých se daný pojem vyskytuje. Tyto příklady poslouží jako názorná ilustrace uvedeného stručného popisu. Současně lze tuto kapitolu využít i jako zdroj námětů ve chvíli, kdy se učíte nějaký pojem a chcete si jej dobře procvičit.

2.1 Složitost

Dříve než se podíváme na techniky návrhu algoritmů, připomeňme si, jak poměřujeme výkon algoritmů. Přímočarý přístup by byl spustit algoritmy na konkrétním vstupu, změřit jejich rychlost a podle toho určit „vítěze“. Takový přístup má však zřejmé nevýhody: výsledek závisí na počítači, na kterém algoritmy spouštíme, a na volbě konkrétního vstupu. Proto se k porovnání algoritmů většinou používá jiný přístup: neměříme čas běhu na konkrétním