3 Počítání s čísly

Účel výpočtů je vhled, nikoliv čísla.

R. Hamming

Tato kapitola obsahuje náměty na příklady, které mají velmi jednoduché vstupně-výstupní chování: vesměs pouze načtou čísla a na výstup opět dají čísla. Počítání s čísly pro většinu lidí není tak atraktivní jako třeba grafika, logické úlohy nebo hry. Nicméně příklady s čísly jsou vhodné jako úvodní cvičení, protože k nim stačí velmi málo znalostí o syntaxi konkrétního jazyka. Většinou vystačíme s celočíselnými proměnnými, cykly, funkcemi a případně jednorozměrnými poli.

Většina uvedených zadání navíc ilustruje zajímavé vlastnosti čísel či algoritmů, případně se k úlohám vážou zajímavé souvislosti, jako například historické zajímavosti nebo nevyřešené otevřené problémy. Pokud o těchto souvislostech víme, i jednoduché počítání s čísly dostává na zajímavosti.

3.1 Hrátky s čísly

Nápad:	1
Kódování:	1
Styl úlohy:	*Jednoduché cvičení na základní procvičení práce s celočíselnými proměnnými a s cykly.*

Program načte přirozené číslo n a vypíše výsledek výpočtu nad tímto číslem. Náměty na výpočty:

1. součet prvních n přirozených čísel,
2. faktoriál čísla n, tj. součin prvních n čísel,
3. celou část odmocniny z čísla n, tj. největší celé x takové, že $x^2 \leq n$,
4. ciferný součet čísla n,
5. počet jedniček obsažených v čísle n,
6. číslo n zapsané pozpátku,
7. informace o tom, zda n je platné rodné číslo.

Doplňující komentář

Toto cvičení slouží především k procvičení cyklů. Z tréninkových důvodů je užitečné vyřešit stejný úkol několika různými způsoby. V řešeních na konci knihy je například uvedeno 5 možností, jak zapsat řešení prvního úkolu. U příkladů 4.–7. spočívá základní princip v procházení jednotlivých cifer čísla. Toho snadno dosáhneme tak, že číslo postupně dělíme 10 a zkoumáme zbytek po dělení 10 (operace modulo).

3.2 Posloupnosti

Nápad:	1-3
Kódování:	1-2
Styl úlohy:	*Procvičení práce s celočíselnými proměnnými, s cykly a případně poli.*

Odhalte princip následujících posloupností a poté napište pro každou z nich program, který dostane na vstup číslo n a na výstup vypíše prvních n členů dané posloupnosti.

1. 1, 2, 3, 4, 5, 6, ...
2. 1, 3, 5, 7, 9, 11, ...
3. 1, 5, 9, 13, 17, 21, 25, 29, ...
4. 1, 2, 4, 7, 11, 16, ...
5. 1, 2, 4, 8, 16, 32, ...
6. 1, 2, 6, 24, 120, ...
7. 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, ...
8. 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, ...
9. 1, 2, 4, 7, 9, 12, 18, 24, 32, 38, 42, 50, 56, 64, 71, 73, 75, 81, ...
10. 1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, ...
11. 1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 9, 10, ...

Doplňující komentář

Principy posloupností jsou následující:

1. přirozená čísla,
2. lichá čísla, tj. aritmetická posloupnost s krokem 2,
3. aritmetická posloupnost s krokem 4,
4. aritmetická posloupnost 2. stupně – rozdíly mezi členy tvoří aritmetickou posloupnost s krokem 1, tj. přirozená čísla,
5. mocniny dvojky, tj. geometrická posloupnost s krokem 2,

6. faktoriály,
7. úseky přirozených čísel postupně se prodlužující délky,
8. prvočísla,
9. další člen vznikne tak, že k aktuálnímu členu přičteme počet jeho dělitelů,
10. popis předchozího členu, další člen dostaneme tak, že předchozí člen „přečteme nahlas“: „jedna jednička“, „dvě jedničky“, „jedna dvojka, jedna jednička“,
11. Golombova sebe-popisující posloupnost, neklesající řada kladných čísel, jejíž n-tý člen řady udává, kolikrát se opakuje číslo n.

Prvních 6 posloupností lze vyřešit jednoduše pomocí jednoho `for` cyklu a několika celočíselných proměnných, další 3 posloupnosti vyžadují vnořené cykly nebo použití pomocné funkce. Z uvedených posloupností jsou nejnáročnější dvě poslední, i když i u nich je náročné přijít především na princip posloupnosti, programátorsky jsou vcelku přímočaré. V případě posloupnosti s popisem předchozího členu může být výhodnější pracovat s členy jako řetězci spíše než jako s celými čísly. Golombova sebe-popisující řada vyžaduje použití jednorozměrného pole.

Mnoho dalších zajímavých námětů na zadání lze najít v internetové encyklopedii celočíselných posloupností (The On-Line Encyclopedia of Integer Sequences, *oeis.org*). V této encyklopedii je uvedena také řada doplňujících komentářů a zajímavostí.

Zajímavým rozšířením této úlohy je automatické doplňování řad – implementace „umělé inteligence“, která odhalí princip posloupnosti a automaticky ji doplní. Toto je reálné jen pro relativně jednoduché posloupnosti,jako je prvních 5 uvedených posloupností. I tak tato varianta představuje úkol obtížnosti 4–5.

3.3 Collatzův problém

Nápad:	1
Kódování:	1
Styl úlohy:	*Programátorsky jednoduché cvičení, které ilustruje velmi zajímavý problém z teorie čísel.*

Předmětem tohoto cvičení je matematický problém, který je znám pod mnoha různými názvy – kromě názvu Collatzův problém se můžete setkat též s označením $3n+1$ problém, Ulamův problém, Syrakuský problém a spoustou dalších jmen. Problém se týká následujícího postupu: „vezmi přirozené číslo, pokud je sudé, vyděl jej dvěma, pokud je liché, vynásob jej třemi a přičti jedničku;

tento postup opakuj, dokud nedostaneš číslo jedna“. Program tedy vypadá následovně:

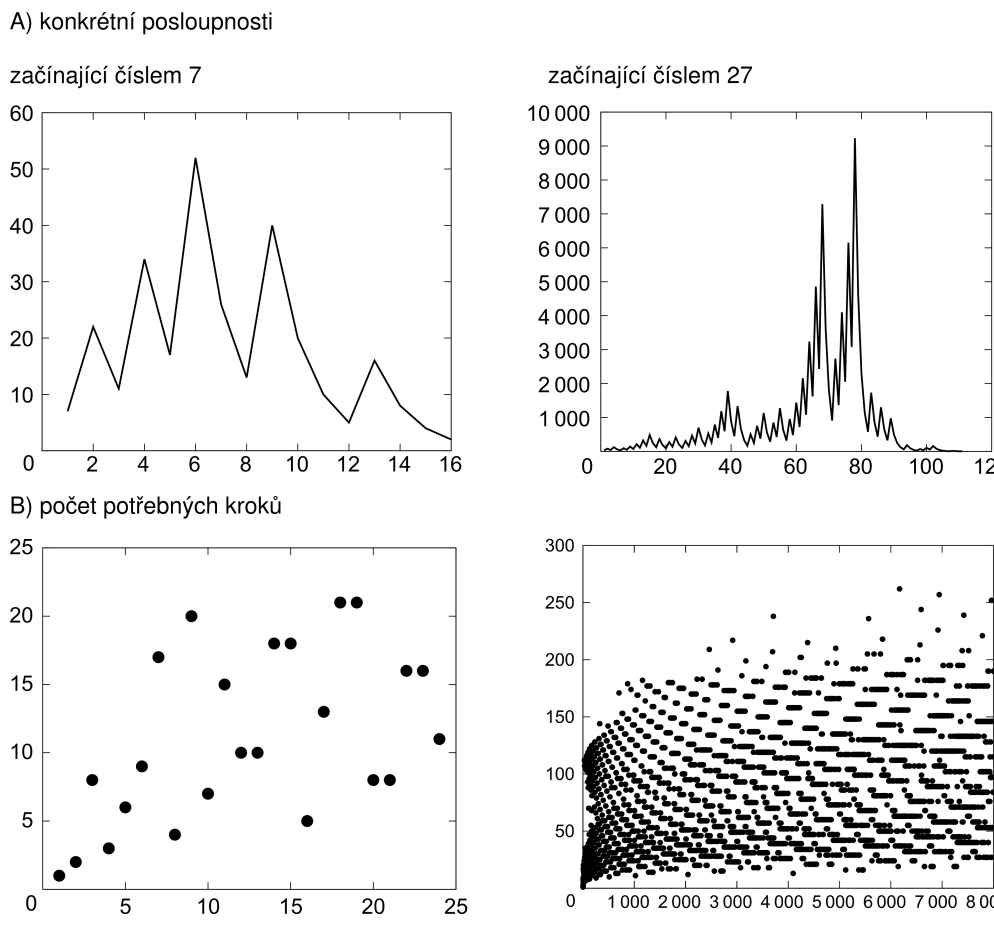
```
def collatz(n):
    while n != 1:
        print n
        if n % 2 == 0:
            n = n / 2
        else:
            n = 3*n + 1
```

Ilustrujme postup na příkladu. Pokud začneme v čísle 7, dostaneme následující posloupnost: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. Ačkoliv je definice postupu jednoduchá, výsledné chování je velmi zajímavé. Již na příkladu začínajícím v čísle 7 je vidět, že chování vypadá docela „chaoticky“. A když začneme číslem 27, potřebujeme dokonce 111 kroků, než se dostaneme na číslo 1 a jako jeden z mezivýsledků dostaneme číslo 9 232. Průběh těchto dvou posloupností je graficky znázorněn na obrázku 3.1.

Když máme takto definovaný postup, nabízí se následující otázka. Platí, že ať začneme v libovolném přirozeném čísle, skončí posloupnost v jedničce? Tato otázka činí uvedenou posloupnost známou, protože nikdo totiž zatím nezná s jistotou odpověď. Collatzova (nebo též Ulamova) hypotéza říká, že odpověď na uvedenou otázku je „ano“. Experimentálně bylo ověřeno, že i pro velmi vysoká čísla se výpočet nakonec dostane do jedničky, takže většina lidí věří, že hypotéza platí. Nicméně matematický důkaz zatím nemáme a problém je stále otevřený.

V rámci našeho cvičení se nebudeme pouštět do řešení obtížných matematických problémů, ale využijeme téma k několika jednoduchým programátorským cvičením:

1. Pro zadané číslo vypočítejte posloupnost vygenerovaných čísel a tuto posloupnost zobrazte graficky (podobně jako na obrázku 3.1 A).
2. Pro čísla od 1 do n vypočítejte, kolik kroků potřebujeme, než se dostaneme do jedničky. Hodnoty vykreslete grafem – pro malé n vypadá výsledný graf vcelku náhodně, pro velké n se však objeví zajímavá struktura (viz obrázek 3.1 B).
3. Podobně jako předchozí bod, nezobrazujte však počet kroků, ale maximální hodnotu, na kterou v průběhu výpočtu narazíme (např. pro začátek v čísle 7 je touto maximální hodnotou 52).
4. Pro které číslo od 1 do 100 000 potřebujeme nejvíce kroků, než dospějeme do jedničky? Jaké je maximální číslo, které se v příslušné posloupnosti objeví?



Obrázek 3.1: Collatzův problém – grafy

5. Rekordman je takové číslo x, pro které je počet potřebných kroků větší než pro jakékoliv $y < x$. Posloupnost rekordmanů začíná 1, 2, 3, 6, 7, 9, 18, 25, 27, 54. Najděte všechny rekordmany do 100 000.
6. Vyzkoušejte jiné funkce podobného typu a prozkoumejte jejich chování. Co se například stane, když výraz $3n + 1$ nahradíme za $3n - 1$? Pak už uvedená hypotéza neplatí – když začneme ve vhodném čísle, výpočet se zacyklí a nikdy se nedostane do jedničky. Najděte takové číslo.

Doplňující komentář

Pokud bychom se chtěli problémem zabývat pro vysoká čísla, bylo by smysluplné snažit se program optimalizovat například pomocí ukládání mezivý-

sledků. Pro hodnoty uvedené v zadání (do 100 000) však bohatě stačí přímočaré řešení bez jakýchkoliv optimalizací – prostě přepíšeme funkci pro počítání posloupnosti a pak ji opakovaně voláme. Pro grafické zobrazení výsledků můžeme využít knihovnu pro zobrazení grafů, jež je dostupná ve většině programovacích jazyků. Klidně ale můžeme použít i běžný tabulkový procesor, do kterého zkopírujeme výsledky našeho programu.

3.4 Náhodná procházka

Nápad:	1-2
Kódování:	1-3
Styl úlohy:	*Jednoduchá cvičení s využitím generátoru náhodných čísel.*

Náhodná procházka je postup spočívající v mnoha opakovaných náhodných krocích. Jde o jednoduchý matematický princip, který však dobře modeluje mnoho reálných jevů a je intenzivně studován v oblastech, jako je fyzika, finance nebo biologie. Pro účel úvodního programátorského cvičení se zaměříme pouze na jednoduché variace na toto téma:

1. „Opilcova procházka“ probíhá na jednorozměrném plánu velikosti n. Na levém konci plánu je domov, na pravém hospoda. Opilec se na začátku nachází na poli číslo k. V každém kole se opilec s pravděpodobností p pohne směrem k hospodě a s pravděpodobností $1 - p$ směrem domů. Procházka končí, když opilec dorazí do hospody nebo domů.

2. „Náhodná procházka ve 2D“ probíhá na mřížce velikosti $n \times n$, začínáme uprostřed, v každém kroku se pohneme se stejnou pravděpodobností na jednu ze 4 sousedních buněk. Procházka končí, když narazíme na kraj mřížky.

3. Zjednodušené „Člověče, nezlob se“ se hraje na hracím plánu velikosti n a pouze s 1 figurkou. Figurka se posunuje podle hodů kostkou, tj. podle náhodně generovaných čísel od 1 do 6. Když padne číslo 6, házíme znova a čísla sčítáme. Figurka se posunuje o celkový součet. Hra končí, jakmile dorazíme na poslední pole, přičemž se musíme trefit přesně na poslední pole. Pokud se netrefíme, figurka stojí.

Základní úkol pro všechny varianty spočívá v napsání programu, který pro zadané parametry odsimuluje průběh procházky a textově či graficky jej znázorní. Rozšiřující úkol spočívá v opakovaném spuštění simulace a výpočtu průměrných charakteristik simulace: Jaká je pro dané n, k, p pravděpodobnost, že opilec dorazí do hospody? Jaká je průměrná délka 2D procházky na mřížce $n \times n$? Jak závisí u zjednodušeného „Člověče, nezlob se“ délka hry na velikosti plánu?

Doplňující komentář

Jde o využití základních programátorských konstrukcí, takže úlohy nebudeme rozebírat. Jen poznamenejme, že minimálně pro některé varianty lze uvedené průměrné charakteristiky vypočítat i matematicky, tj. bez provádění simulací. Čtenář s matematickými ambicemi se tedy může pokusit odvodit příslušný výsledek a pak jej pomocí programu ověřit. Případně je možné se zabývat nejen průměrnými hodnotami, ale i dalšími parametry příslušných pravděpodobnostních distribucí, například mediánem nebo rozptylem.

3.5 Dělitelnost a prvočísla

Nápad:	2-3
Kódování:	1-3
Styl úlohy:	*Jednoduchá cvičení řešitelná různými algoritmickými přístupy, na kterých lze ilustrovat rozdíly v efektivitě algoritmů.*

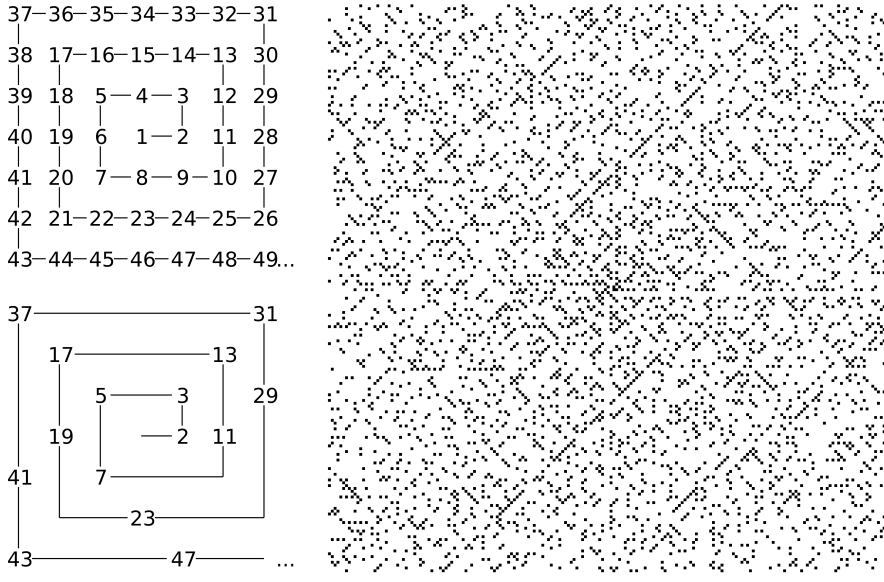
Základní zadání spočívají v tom, že program načte přirozená čísla a vypíše informace související s dělitelností či prvočíselností:

1. Výpis všech dělitelů čísla n.
2. Test prvočíselnosti čísla n.
3. Výpis prvních n prvočísel.
4. Výpis největšího společného dělitele čísel n a m.
5. Hledání „prvočíselných dvojčat“: program načte číslo n a najde nejmenší $p \geq n$ takové, že p a $p + 2$ jsou obě prvočísla.

Pokročilejší ztvárnění tématu dostaneme za využití bitmapové (případně textové) grafiky, kdy zakreslujeme distribuci prvočísel graficky pomocí takzvané Ulamovy spirály. Ta vznikne tak, že si napíšeme všechna čísla „do šneka“ a potom vybereme pouze prvočísla. Na obrázku 3.2. máme takto znázorněno prvních 40 tisíc přirozených čísel, prvočísla jsou černě. Podobným způsobem můžeme znázornit třeba čísla dělitelná zadaným číslem n.

Doplňující komentář

Prvočísla jsou velmi zajímavý matematický objekt, protože nejsou nijak striktně pravidelná, ale přitom vykazují řadu dílčích pravidelností. Tuto vlastnost prvočísel názorně ilustruje Ulamova spirála, kterou objevil Stanislav Ulam v šedesátých letech, když si čmáral po papíře během nudné přednášky. Výsledný obrázek nemá žádný zjevný řád, nicméně jsou na něm vidět výrazné „diagonály“. S prvočísly se také váže mnoho těžkých a často i nevyřešených



Obrázek 3.2: Ulamova spirála: ilustrace principu a spirála velikosti 200×200

matematických problémů. Jeden takový známý problém souvisí s prvočíselnými dvojčaty zmíněnými v zadání: Existuje nekonečně mnoho prvočíselných dvojčat?

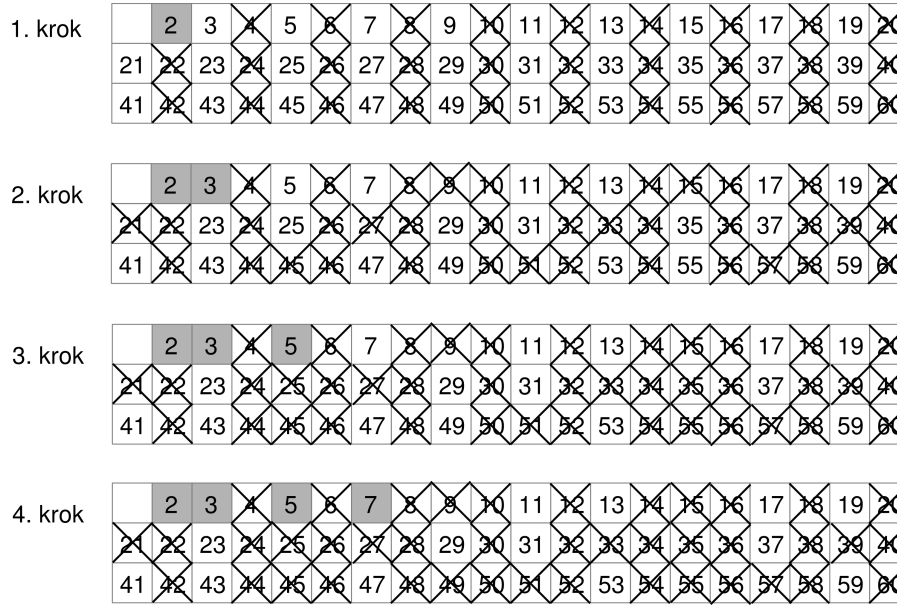
Výpis všech dělitelů lze realizovat přímočaře pomocí jednoho `for` cyklu. Test prvočíselnosti můžeme udělat přímočaře spočítáním dělitelů. Program můžeme mírně optimalizovat, například tak, že zkoušíme dělit pouze dvojkou a lichými čísly, a to jen do odmocniny z n. Jsou pochopitelně možné i výraznější optimalizace – testování prvočísel je důležitý praktický problém například v moderní kryptografii. Pokročilejší optimalizace ale jdou vysoce nad rámec úvodních programátorských cvičení.

Výpis prvočísel můžeme udělat dvěma základními způsoby, z tréninkových důvodů je vhodné implementovat oba dva. První metoda je prostá hrubá síla. Procházíme postupně přirozená čísla a pro každé z nich zjistíme pomocí výše popsaného testu, zda je prvočíslem. Druhá metoda se nazývá Eratosthenovo síto a je ilustrována na obrázku 3.3. Napíšeme si čísla od 2 do n, v každém kroku vždy vybereme nejmenší neoznačené číslo a označíme všechny násobky tohoto čísla. Všimněte si, že v uvedeném obrázku jsou již po 4 krocích výpočtu všechna neoznačená čísla prvočísla.

Také hledání největšího společného dělitele můžeme dělat více způsoby. Základní řešení je opět hrubá síla: procházíme sestupně čísla od n do 1 a zkou-

šíme jimi dělit n i m, dokud nenajdeme společného dělitele. Lepší řešení je Euclidův algoritmus, který je založen na pozorování, že $NSD(n, m) = NSD(m, n \bmod m)$, kde $\bmod$ značí zbytek po dělení. Tento vztah můžeme snadno převést na algoritmus výpočtu, například pro čísla 504 a 180 probíhá výpočet následovně: $NSD(504, 180) = NSD(180, 144) = NSD(144, 36) = NSD(36, 36) = NSD(36, 0) = 36$. Alternativní zápis stejného výpočtu ukazuje tabulka 3.1.

Euclidův algoritmus je výrazně efektivnější než naivní algoritmus, což jde snadno experimentálně ověřit. Příklad slouží jako dobrá ilustrace rozdílů mezi výpočetní náročností různých algoritmů.



Obrázek 3.3: Eratosthenovo síto: první 4 kroky výpočtu

Tabulka 3.1: Euclidův algoritmus: příklad

krok	n	m
1	504	180
2	180	144
3	144	36
5	36	36
6	36	0