Matematická indukce III – V rytmu algoritmů

Milý příteli,
vítáme Tě u třetího a posledního dílu seriálu o matematické indukci. Tentokrát se podíváme na algoritmy, které využívají indukci. Řeč bude o tom, jak najít nejvyššího společného dělitele dvou čísel, jak najít společné kořeny polynomů, ale zabrousíme i do oblasti diofantických rovnic. Poté se podíváme na příklad z trochu jiného soudku, neboť prozkoumáme způsoby seřazení balíčku karet podle jejich hodnoty. Závěrečná kapitola je spíše k zamyšlení nad tím, jak určit, zda algoritmus po nějakém čase doběhne.

Podnětné čtení Ti přejí

autoři

O smyslu algoritmizace

Algoritmus je proces nebo postup, který po konečně mnoha krocích dospěje k nějakému závěru, nejčastěji k vyřešení zadané úlohy. Jednotlivé kroky jsou jednoznačně zadané a často jednoduché na provedení. Zároveň je daný algoritmus v jistém smyslu univerzální, tedy dokáže vyřešit více úloh podobného druhu.

Etymologické okénko. Původ slova algoritmus sahá do 9. století. Název je odvozen od jména perského matematika Al-Chorezmího, který zavedl počítání s arabskými číslicemi (tehdy nazývané indickými) a položil základy algebry. Původní slovo *algorismus* znamenalo pouze počítání v desítkové soustavě. Později se pod vlivem řeckého slova *arithmos* začala používat zkomolenina *algoritmus*. Význam, pod kterým jej známe dnes, však slovo získalo až v 19. století.

Algoritmy však byly známé již starým Babyloňanům 2500 let před naším letopočtem. Staří Řekové zas používali například Eratosthenovo síto pro hledání prvočísel nebo Eukleidův algoritmus, který si představíme i v tomto dílu.

Proč jsou algoritmy užitečné? Ve statistice slouží k analýze dat a také k rozpoznání těch dat, která mají nějaký reprezentativní význam, například při jejich roztřídění do takzvaných clusterů. V aplikované matematice jsou algoritmy používány pro hledání přibližných řešení matematických modelů skutečného světa. Nicméně i v čisté matematice mají algoritmy svůj význam – například pro důkaz existence řešení. Síla algoritmů spočívá také v jejich jednoduchosti a repetitivní podstatě, čehož dokáží počítače využít pro extra rychlé nalezení řešení. Lidé oproti tomu nejsou tak efektivní v provádění ručních opakovaných výpočtů, ale umí napsat program, který to za ně udělá v řádu milisekund!

Dělicí algoritmus

Úmluva. V průběhu budeme pracovat s přirozenými čísly $\mathbb{N}_0 = \{0, 1, 2, \dots\}$ jako jsme je definovali v předchozím díle. Podotkněme, že nulu zde považujeme za přirozené číslo.

Mějme dvě přirozená čísla m, n, kde n je různé od nuly. Již ze základní školy víme, jak se dělí se zbytkem. Při dělení se zbytkem hledáme přirozená čísla q a r, kde r je menší než n, pro která platí

$$m = qn + r.$$

Je tato operace dělení dobře definovaná? Existuje vždy právě jeden výsledek, tedy právě jedna dvojice čísel q a r? Není to na první pohled zřejmé[1], proto si to pojďme dokázat:

Věta. *Nechť m, n jsou dvě přirozená čísla, kde $n \neq 0$. Potom existuje právě jedna dvojice přirozených čísel q a r takových, že $m = qn + r$ a $r < n$.*

Důkaz. Nejprve ukážeme existenci. Postupujme indukcí podle m. Nechť

$$S = \{m \in \mathbb{N}_0;\ m = qn + r \text{ pro nějaká } q, r \in \mathbb{N}_0, r < n\}.$$

Všimněme si, že $0 \in S$, neboť $0 = 0n + 0$. Nyní předpokládejme, že $m \in S$. Potom $m = qn + r$ a $r < n$, tedy

$$m + 1 = qn + r + 1.$$

Jelikož $r < n$, tak $r + 1 \leq n$ (jsme v přirozených číslech). Tedy buď $r + 1 = n$ a dostáváme

$$m + 1 = (q + 1)n + 0,$$

nebo $r + 1 < n$ a potom

$$m + 1 = qn + (r + 1),$$

kde $r + 1 < n$. V obou případech platí $m + 1 \in S$, tedy dle principu indukce (třetího Peanova axiomu) máme $S = \mathbb{N}_0$.

Nyní ukážeme, že daná q a r jsou jednoznačná. Mějme

$$m = qn + r = q'n + r',$$

kde $r, r' < n$. Potom

$$qn \leq m < (q + 1)n,$$
$$q'n \leq m < (q' + 1)n.$$

Z toho dostáváme $qn < (q' + 1)n$, a jelikož $n > 0$, tak $q < q' + 1$, tedy $q \leq q'$, protože pracujeme s přirozenými čísly. Obdobně dostaneme $q' \leq q$, takže dohromady $q = q'$. A z předpokladu pak plyne rovnost zbytků $r = m - qn = m - q'n = r'$. Tedy dvojice čísel q a r je opravdu jednoznačná. □

Přirozeně se můžeme ptát, co se stane, když budeme dělicí algoritmus aplikovat opakovaně. Pojďme to společně probádat.

Nechť a, b jsou dvě přirozená čísla. Potom dle dělicího algoritmu existují q_1 a $r_1 < b$ taková, že

$$a = q_1 b + r_1.$$

Pokračujme dál a aplikujme dělicí algoritmus na čísla b a r_1.

$$\begin{aligned} b &= q_2 r_1 + r_2, \\ r_1 &= q_3 r_2 + r_3, \\ r_2 &= q_4 r_3 + r_4, \\ r_3 &= q_5 r_4 + r_5, \\ &\vdots \\ r_i &= q_{i+2} r_{i+1} + r_{i+2}. \end{aligned}$$

[1] Například při dělení se zbytkem v okruhu *Gaussových čísel* $\mathbb{Z}[i]$ není zbytek jednoznačný, takže to rozhodně není samozřejmost! O Gaussových číslech se můžeš dočíst v prvním díle minulého seriálu *Teorie nejen čísel 1.*

Napřed několik pozorování:

Pozorování první. V každém kroku máme dle dělicího algoritmu zaručená čísla q_{i+2} a $r_{i+2} < r_{i+1}$, dokud není nějaký zbytek roven nule, protože nulou dělit nemůžeme (viz dělicí algoritmus).

Pozorování druhé. Získali jsme klesající posloupnost $b > r_1 > r_2 > r_3 > \cdots$

Pozorování třetí. Proces se po nějakém čase vždy zastaví.

Cvičení 1. Rozmysli si, že třetí pozorování vyplývá z prvního a druhého pozorování a z Úlohy 3 v předchozím dílu.

V následující kapitole se k tomuto procesu vrátíme a prozkoumáme jeho další zajímavé vlastnosti!

Eukleidův algoritmus

Pamatuješ si, jak jste ve škole hledali nejvyššího společného dělitele dvou čísel? Postup byl následující: nejprve obě čísla rozložíme na součin prvočísel a poté hledáme, která prvočísla mají oba rozklady společná. Avšak to jsme to trochu uspěchali – jak například víme, že lze pokaždé prvočíselný rozklad najít, a pokud ho náhodou najdeme, jak víme, že je jen jeden? To nám náš postup trochu komplikuje, skoro to vypadá, že by dvě čísla mohla mít více různých největších společných dělitelů!

Abychom si tento problém vyjasnili, potřebujeme nejdřív přesně zadefinovat potřebné pojmy.

Definice. Nechť m, n jsou přirozená čísla, kde n je různé od nuly. Potom říkáme, že n *dělí* m, pokud existuje přirozené číslo k takové, že $m = nk$. Tuto skutečnost zapisujeme jako $n \mid m$.

Všimněme si, že každé číslo (kromě jedničky) má alespoň dva dělitele, jedničku a sebe sama. Některá čísla jsou výjimečná tím, že je už dál dělit nelze.

Definice. Nechť p je přirozené číslo. Potom p je *prvočíslo*, jestliže p má právě dva různé dělitele.

Poznámka. Poznamenejme, že číslo 1 podle definice není prvočíslo. Proč to tak je? Není za tím žádná magie, důvod je ten, že definice vymýšlí lidé tak, aby byly co nejužitečnější. A prostě se hodí, aby jednička prvočíslem nebyla.

Pokud číslo d dělí m a současně dělí i n, potom říkáme, že d je *společný dělitel* čísel m a n. Číslo 1 je tak pokaždé společným dělitelem. Pokud je to zároveň jediný společný dělitel, říkáme, že čísla m a n jsou *nesoudělná*.

Vraťme se nyní k nejvyššímu společnému děliteli.

Definice. Přirozené číslo d je *největším společným dělitelem* čísel m, n, pokud platí následující dvě podmínky:

(i) d je společný dělitel m a n,
(ii) pokud c je dalším společným dělitelem m a n, pak $c \mid d$.

Potom píšeme $d = \mathrm{NSD}(m, n)$.

Poznámka. Rozmysli si, že tato definice je ekvivalentní s tím, že d je největší ze společných dělitelů.

Poznámka. S naší novou definicí můžeme říct, že čísla m a n jsou nesoudělná právě tehdy, když $\mathrm{NSD}(m, n) = 1$.

Přišel čas vrátit se k opakovanému použití dělicího algoritmu, kterým jsme končili předchozí kapitolu. Vyzbrojeni novou terminologií můžeme učinit další pozorování:

Pozorování čtvrté. Pokud proces skončí u $r_{i+2} = 0$, dostáváme rovnost $r_i = q_{i+1}r_{i+1}$, z čehož plyne, že r_{i+1} dělí r_i. O řádek výš máme rovnost

$$r_{i-1} = q_{i+1}r_i + r_{i+1},$$

tedy pravá strana je dělitelná číslem r_{i+1}, protože r_{i+1} dělí r_i. Proto je i r_{i-1} na levé straně dělitelné r_{i+1}. Takto postupujeme pořád nahoru a iterativně dostáváme, že číslo r_{i+1} dělí jak b, tak a, a tudíž je společným dělitelem výchozích dvou čísel.

Za chvíli ukážeme ještě mnohem silnější tvrzení, a sice že r_{i+1} je největším společným dělitelem čísel a a b.

Toto opakované použití dělicího algoritmu má svůj vlastní název – *Eukleidův algoritmus*. Výsledkem Eukleidova algoritmu je právě číslo r_{i+1}, tedy poslední nenulový zbytek.

Věta. *Nechť a a $b \neq 0$ jsou přirozená čísla. Potom Eukleidův algoritmus generuje nejvyššího společného dělitele čísel a a b.*

Důkaz. Nejprve si připomeňme značení jednotlivých členů v algoritmu:

$$\begin{aligned}
a &= q_1 b + r_1,\\
b &= q_2 r_1 + r_2,\\
r_1 &= q_3 r_2 + r_3,\\
r_2 &= q_4 r_3 + r_4,\\
r_3 &= q_5 r_4 + r_5,\\
&\vdots\\
r_i &= q_{i+2} r_{i+1} + r_{i+2},\\
&\vdots
\end{aligned}$$

Když v důkazu používáme algoritmus, měli bychom si položit následující otázky:

(1) Doběhne algoritmus někdy, tedy zastaví se proces po nějakém čase?
(2) Pokud se zastaví, vyhodí nám požadovanou odpověď (v našem případě největšího společného dělitele)?
(3) Jak rychle algoritmus konverguje, tj. jak rychle se blíží ke správné odpovědi?

Prvním krokem důkazu je ukázat, že se algoritmus zastaví, což je součástí Cvičení 1. Nechť r_{i+2} je první zbytek roven nule. Chceme ukázat, že r_{i+1} neboli přecházející zbytek je nejvyšším společným dělitelem čísel a a b.

Druhý krok již máme téměř za sebou: ve čtvrtém pozorování jsme nahlédli, že r_{i+1} je společným dělitelem a a b.

Nyní bychom rádi dokázali, že se jedná o nejvyššího společného dělitele. Budeme postupovat podle naší definice. Nechť tedy d je dalším dělitelem a a b. To znamená, že existují přirozená čísla α a β taková, že

$$a = \alpha d \qquad \text{a} \qquad b = \beta d.$$

Jelikož $a = q_1 b + r_1$, tak $r_1 = (\alpha - q_1\beta)d$, takže d dělí r_1. Induktivně dostáváme, že d dělí i další zbytky včetně předposledního, tedy d dělí r_{i+1}. Potom dle definice r_{i+1} je nejvyšším společným dělitelem a a b. □

Cvičení 2. Najdi nejvyššího společného dělitele čísel 3570 a 323 nejprve pomocí rozkladu na součin prvočísel a poté použitím Eukleidova algoritmu.

Cvičení 3. Ukaž, že čísla 19891 a 2022 jsou nesoudělná.

Dělicí algoritmus a Eukleidův algoritmus lze použít i pro dva polynomy. Místo zmenšujících se zbytků však potřebujeme něco jiného, co se bude při každé iteraci snižovat, což zaručí, že algoritmus někdy doběhne. Toto „něco“ je pak stupeň polynomu.

Úloha 1. Vyslov tvrzení podobné dělicímu algoritmu pro polynomy s reálnými koeficienty a dokaž jej.

Eukleidův algoritmus pak funguje úplně stejně i pro dva polynomy. Vyzkoušejte si to na úloze!

Úloha 2. Najdi mnohočlen nejvyššího možného řádu, který dělí $x^3 - 9x^2 - x + 105$ a zároveň $x^2 - 9x + 14$.

Úloha 3. Najdi všechny společné kořeny polynomů $x^4 + x^3 - 21x^2 - x + 20$ a $x^3 - x^2 - 22x + 40$.

Nyní přichází na řadu otázka, jak efektivní Eukleidův algoritmus je. Kolikrát je obecně třeba iterovat dělicí algoritmus, než se dostaneme k výsledku? Počet operací můžeme shora odhadnout podle velikosti b: jelikož $b > r_1 > r_2 > \cdots$, tak zbytek nula dostaneme nanejvýš po b aplikacích dělicího algoritmu.

Avšak je možné dosáhnout i přesnějších odhadů!

Věta. *Nechť $a > b > 0$ jsou dvě přirozená čísla. Předpokládejme, že Eukleidův algoritmus pro a a b sestává z N kroků. Potom nejnižší hodnota, které může a (resp. b) nabývat, je Fibonacciho číslo*[2] *$a = F(N + 2)$ (resp. $b = F(N + 1)$).*

Věta. *Počet iterací v Eukleidově algoritmu nikdy nemůže překročit pět krát počet cifer čísla b.*

Tato věta ukazuje, že horní mez počtu iterací roste úměrně počtu cifer čísla b, což ukazuje na *logaritmický* růst. Růst je měřítkem toho, jaká je cena algoritmu, tedy kolik kroků proces potřebuje, než se zastaví, vzhledem ke vstupním hodnotám, v tomto případě hodnotě čísla b. Je zřejmé, že čím větší čísla do Eukleidova algoritmu vhodíme, tím déle může iterativní proces trvat. Růst potom vyjadřuje konkrétní vztah mezi velikostí vstupu a cenou.

Důkaz těchto vět přesahuje odbornost tohoto textu, přesto je zmiňujeme pro zajímavost. O těchto vlastnostech algoritmu pravděpodobně Eukleides nevěděl, protože například druhá z vět byla dokázána až v roce 1844.

Diofantické rovnice

Ukážeme si zajímavé použití Eukleidova algoritmu při řešení diofantických rovnic. Diofantické rovnice jsou rovnice, jejichž řešení hledáme pouze v oboru celých čísel $\mathbb{Z}$. Taková rovnice může například vypadat takto:

$$13x + 7y = 2,$$

kde $x, y \in \mathbb{Z}$. Zkuste dosadit $(x, y) = (5, -9)$. Jak ale na toto řešení přijít? Existuje obecný postup? Nejsou možná i jiná řešení? Jak najít všechna řešení?

Zkusme použít Eukleidův algoritmus na koeficienty proměnných:

$$\begin{aligned} 13 &= 1 \cdot 7 + 6, \\ 7 &= 1 \cdot 6 + 1. \end{aligned}$$

Dobrá, to nám řeklo jen to, co už jsme dávno věděli: že čísla 13 a 7 jsou nesoudělná. Bodejť, vždyť jsou to prvočísla. Avšak můžeme z toho získat víc ...

[2]Fibonacciho čísla jsou definována rekurzí $F(0) = 0$, $F(1) = 1$ a $F(n + 1) = F(n) + F(n - 1)$ pro $n \geq 1$. Více o Fibonacciho číslech najdeš v prvním dílu seriálu.

Postupujme pozpátku, z druhé rovnosti vyjádříme $1 = 7 - 1 \cdot 6$ a z první nerovnosti pak $6 = 13 - 1 \cdot 7$. Dohromady dostáváme

$$1 = 7 - 6 = 7 - (13 - 7) = (-1) \cdot 13 + 2 \cdot 7.$$

Stačí tuto rovnost vynásobit dvěma, abychom dostali

$$2 = (-2) \cdot 13 + 4 \cdot 7,$$

takže jsme zkonstruovali další řešení naší rovnice, a to $(x, y) = (-2, 4)$.

Cvičení 4. Pomocí Eukleidova algoritmu najdi jedno celočíselné řešení

$$27x - 15y = 21.$$

Umíme tedy generovat alespoň jedno řešení (pokud nějaké existuje). Dokonce platí i něco silnějšího: pokud existuje jedno řešení, tak jich už nutně existuje nekonečně mnoho. Než se však pustíme do jejich hledání, zkus si vyřešit následující cvičení, která tě nasměrují na cestu k nalezení odpovědí na otázky v úvodu kapitoly.

Cvičení 5. Ukaž, že rovnice

$$72x - 117y = 42$$

nemá celočíselná řešení.

Cvičení 6. Nechť a a b jsou nesoudělná přirozená čísla. Najdi všechna celočíselná řešení (x, y) rovnice

$$ax + by = 0$$

a ukaž, že jiná řešení nejsou.

Možná už na základě svých pozorování při řešení předchozích cvičení zvládneš sám (sama) dokázat následující větu:

Věta. *Předpokládejme, že (x_0, y_0) je řešení diofantické rovnice $ax + by = c$, kde $a, b, c \in \mathbb{Z}$ jsou dané konstanty. Potom (x, y) řeší tuto rovnici právě tehdy, když $x = x_0 + x_h$ a $y = y_0 + y_h$, kde (x_h, y_h) je (nějakým) řešením* **homogenní** *rovnice*

$$ax + by = 0.$$

Důkaz. Tvrzení, které chceme dokázat, je ekvivalence, takže musíme dokázat oba směry implikace.

Nejprve tedy předpokládejme, že (x, y) řeší (nehomogenní) rovnici $ax + by = c$. Jelikož (x_0, y_0) je také řešením, tak

$$\begin{aligned} 0 &= c - c \\ &= (ax + by) - (ax_0 + by_0) \\ &= a(x - x_0) + b(y - y_0). \end{aligned}$$

Pokud tedy položíme $x_h = x - x_0$ a $y_h = y - y_0$, tak (x_h, y_h) je řešením homogenní rovnice. Potom $(x, y) = (x_0 + x_h, y_0 + y_h)$ má požadovaný tvar.

Nyní ukážeme opačnou implikaci, a sice že $(x, y) = (x_0 + x_h, y_0 + y_h)$, kde (x_h, y_h) řeší homogenní rovnici, je řešením (nehomogenní) rovnice $ax + by = c$. Máme

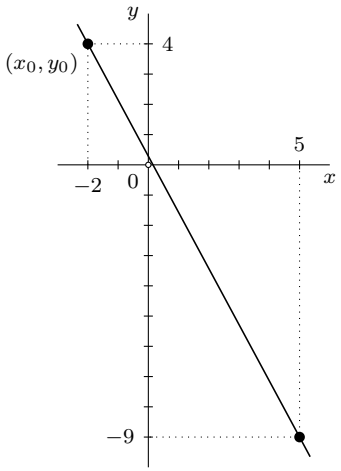
$$\begin{aligned} ax + by &= a(x_0 + x_h) + b(y_0 + y_h) \\ &= (ax_0 + by_0) + (ax_h + by_h) \\ &= c + 0 \\ &= c \end{aligned}$$

a jsme hotovi. □

Dobrá, víme tedy, že stačí najít jedno řešení (x_0, y_0), a pak nejenže $(x, y) = (x_0 + x_h, y_0 + y_h)$ je řešením, ale navíc všechna další řešení mají nutně tento tvar! To nám patřičně zužuje výběr a za chvíli nám to umožní ukázat, že jsme nalezli všechna řešení.

Nyní pojďme prozkoumat, co jsou zač ta řešení homogenní rovnice. Dobrá zpráva je, že jsi už většinu práce odvedl(a), protože jsi jistě poctivě vyřešil(a) Cvičení 6. Důležité je podotknout, že ve Cvičení 6 jsi vyřešil(a) homogenní rovnici pro a a b nesoudělná. Naštěstí je tu „easy fix“: Pokud jsou a a b soudělná, prostě vydělíme celou rovnici jejich největším společným dělitelem. A pokud jím není dělitelné číslo c, tak už díky Cvičení 5 víme, že potom rovnice nemá řešení (v oboru celých čísel) – stačí zobecnit tvrzení pro rovnici $ax + by = c$, která nemá řešení, pokud c není dělitelné největším společným dělitelem čísel a a b.

Řešení diofantické rovnice si můžeme představit i graficky. V rovině $\mathbb{R}^2$ s osami x a y jsou body s celočíselnými souřadnicemi právě mřížové body celočíselné mřížky. Nalezení jednoho konkrétního řešení (x_0, y_0) odpovídá jednomu bodu mřížky. Potom řešení homogenní rovnice určuje, kterým směrem se vydat z tohoto bodu. Všechna řešení pak leží na jedné přímce, která prochází některými mřížovými body, a tyto body jsou hledaná řešení rovnice.



Na obrázku je konkrétní řešení $(x_0, y_0) = (-2, 4)$ rovnice $13x + 7y = 2$, od kterého se vydáme směrem homogenního řešení. V tomto případě přičítáme násobky vektoru $(7, -13)$.

Úloha 4. Najdi všechna řešení diofantické rovnice

$$321x + 17y = 1.$$

Úloha 5. Najdi všechna celočíselná řešení rovnice

$$2022x + 312y = 18.$$

Úloha 6. Odvoď Bézoutovo lemma: Pokud a a b jsou nenulová přirozená čísla, potom existují celá čísla s a t taková, že

$$as + bt = \mathrm{NSD}(a, b).$$

Úloha 7. Nechť a, b a c jsou přirozená čísla taková, že c dělí ab. Ukaž, že pokud $\mathrm{NSD}(a, c) = 1$, tak c dělí b.

Úloha 8. Nechť a, b a c jsou taková přirozená čísla, že $\mathrm{NSD}(a, c) = 1 = \mathrm{NSD}(b, c)$. Dokaž, že potom $\mathrm{NSD}(ab, c) = 1$.

Jak co nejrychleji seřadit balíček karet?

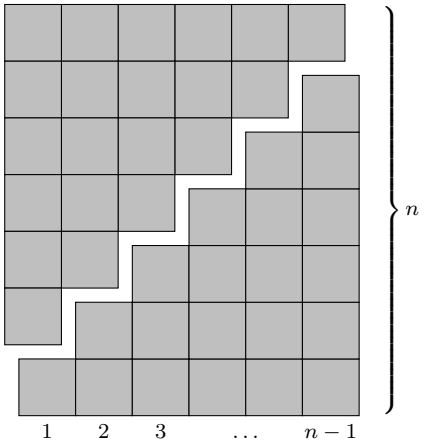
V této kapitole se zaměříme na ryze praktický problém: máme seřadit karty v balíčku vzestupně podle jejich hodnoty a chceme to provést co nejefektivněji.

Intuitivní přístup je brát postupně karty z balíčku a umístit je na správné místo do řady. Představme si, že v balíčku jsou karty s hodnotami 3, 7, 2, 4, 5 v tomto pořadí. Vytáhneme trojku, poté sedmičku položíme napravo od ní, pak vytáhneme dvojku a umístíme ji nalevo od trojky, potom čtyřku dáme mezi trojku a sedmičku a nakonec pětku vložíme mezi čtyřku a sedmičku. Tak jsme čísla seřadili jako 2, 3, 4, 5, 7.

Tomuto přístupu se říká *řazení vkládáním*. Pokud má tento algoritmus provést počítač na n prvcích, hodí se vědět, kolik jednotlivých operací musí v nejhorším možném případě provést. Operacemi myslíme jednotlivá porovnávání dvou prvků. V každém kroku vezmeme jednu kartu z neseřazeného balíčku a postupně ji porovnáváme s prvky v seřazené řadě. Začneme zprava (od největšího): pokud je vkládaná karta větší nebo stejná, umístíme ji napravo a jsme hotovi, je-li však nižší, pokračujeme porovnáním s kartou nalevo, a tak dále. V nejhorším případě musíme kartu porovnat se všemi již vyloženými kartami. Pro n karet tedy musíme provést nanejvýš

$$0 + 1 + 2 + 3 + 4 + \cdots + (n-1)$$

jednotlivých porovnání. Že to je v součtu $\frac{n(n-1)}{2}$, to si můžeš dokázat (např. indukcí[3]), pro stručnost to však zde nahlédneme jen pomocí obrázku:



Tedy v nejhorším možném případě musíme provést řádově n^2 operací – to je kvadratický růst, což je pro objemné balíčky karet opravdu hodně.

Řazení vkládáním, při kterém v každém kroku do $n-1$ již seřazených karet přidáme n-tou na správné místo, však není jediný přístup k řešení našeho problému.

Dalším možným způsobem je *řazení výběrem*. Proces probíhá tak, že z neseřazené řady vybereme nejnižší prvek a umístíme ho na první místo v seřazené řadě. Toto opakujeme s druhým nejnižším

[3]Viz první díl seriálu.

prvkem a tak dále. Kolik porovnání musíme provést? Nejnižší prvek můžeme najít následovně: porovnáme první dva prvky v řadě a z nich vezmeme ten nižší, který porovnáme s následujícím prvkem v řadě, a z těchto dvou opět vybereme ten nižší. Obdobně pokračujeme až na konec řady, kdy nám zbyde ten úplně nejnižší prvek. Hledáme-li tedy nejnižší prvek z řady o n prvcích, musíme provést $n-1$ porovnání. Celkem tedy řazení výběrem pracuje v

$$(n-1)+(n-2)+\cdots+2+1$$

krocích. To už víme, že je dohromady $\frac{n(n-1)}{2}$ kroků. Tedy řazení výběrem trvá stejně dlouho jako řazení vkládáním! Růst je opět kvadratický.

Pro zajímavost si představíme ještě jeden poměrně neefektivní způsob řazení, a tím je *bublinkové řazení*[4]. Nejprve porovnáme první a druhý prvek na začátku řady a případně je prohodíme, pokud je první prvek větší než druhý. Dále porovnáme druhý a třetí prvek a případně jejich pořadí prohodíme. Takto projdeme celou řadu, v každém kroku se soustředíme pouze na dvojici sousedních prvků. Pravděpodobně po tomto prvním projetí řadou nejsou prvky ve správném pořadí. Proto tento proces opakujeme znovu a znovu, dokud není řada seřazená.

Samozřejmě se nabízí otázka, zda nebudeme náhodou „probublávat“ naší řadou donekonečna. Zastaví se algoritmus někdy? Případně jak pozná, že se má zastavit?

Snadno nahlédneme, že nejvyšší číslo vždy probublá na konec řady. Můžeme tedy proces v každém procházení řadou o jeden krok snížit. Postupně tedy zafixujeme poslední číslo, pak předposlední a induktivně dojdeme k závěru, že potřebujeme projet řadou nanejvýš $(n-1)$-krát. Takže algoritmus se po nějakém čase zastaví. Navíc se může zastavit i dřív, pokud jej naprogramujeme tak, že se může zastavit už tehdy, když projede řadou a žádné dva prvky neprohodí. V nejhorším možném případě však musí postupně udělat $(n-1)$ probublání řadou, tedy celkem

$$(n-1)+(n-2)+\cdots+2+1$$

jednotlivých porovnání. Dostali jsme opět stejnou odpověď – kvadratický růst. Není to frustrující – vyzkoušeli jsme již tři různé způsoby řazení a nedostali jsme nic lepšího než kvadratický růst? Naštěstí existují i efektivnější řazení. Pojďme se na jedno podívat.

Představme si, že máme dvě hromádky karet, v nichž je dohromady n karet, které jsou již seřazené. Dokážeme je nyní spojit dohromady? Jistě, otázkou je spíš jak rychle, tedy v kolika krocích.

Cvičení 7. Vymysli způsob, jakým bys spojil(a) dva již seřazené balíčky do jednoho seřazeného. Z kolika kroků (porovnávání) se Tvůj algoritmus skládá?

Chceme seřadit celý balíček, místo toho si však představíme, že řadíme dva menší balíčky, které poté spojíme.

Poznámka. Všimněme si, že předchozí algoritmus řazení vkládáním je vlastně speciálním případem tohoto spojování dvou balíčků dohromady, kdy jeden balíček čítá $n-1$ karet a druhý jen jednu.

Toto jo základní myšlenka algoritmu typu *rozděl a panuj*. Daný problém rozdělíme na menší části, které lze vyřešit, a tyto části pak spojíme. Jak vyřešíme ty menší části? Znovu vyvoláme náš algoritmus – rozdělíme je na ještě menší části, které vyřešíme! Takto pokračujeme dál, dělíme na menší a menší části, až se dostaneme k těm, které lze nějak triviálně vyřešit (v našem případě je triviální jednotkou balíček o jedné kartě, který je rovnou srovnaný). Brzy zjistíme, že tento rekurzivní přístup je rychlejší než řazení vkládáním.

Definujme $P(n)$ jako počet kroků, kolik potřebuje algoritmus rozděl a panuj na seřazení n karet. Pro jednoduchost předpokládejme, že n je mocnina dvou, abychom mohli bez obav dělit balíček na

[4]To nejlepší video najdeš pod tímto odkazem: *https://youtu.be/lyZQPjUT5B4*.

půlky. V triviálním případě pro $n = 1$ máme $P(n) = 0$. Pro $n > 1$ pak dostáváme rekurzivní relaci

$$P(n) = 2 \cdot P\left(\frac{n}{2}\right) + n - 1,$$

což znázorňuje to, že balíček rozdělíme na půlky, které zvlášť vyřešíme v $P\left(\frac{n}{2}\right)$ krocích a poté je spojíme dohromady v $n - 1$ krocích[5]. Vyřešením této rekurzivní rovnice dostaneme počet kroků algoritmu v závislosti na n.

Úloha 9. Pomocí indukce (nebo jinak) ukaž, že $P(n) \leq n \log_2 n$ pro $n = 2^k$, kde k je přirozené číslo.

Tedy počet kroků v tomto algoritmu roste nanejvýš jako $n \log_2 n$, což je pro velká n o mnoho lepší než řazení vkládáním!

Úloha 10. Řazení vkládáním si lze také představit jako rekurenci. Napiš jaké vztahy splňuje příslušné $P(n)$ a indukcí ukaž, že řešením je opravdu $\frac{n(n-1)}{2}$.

Halting problem

Dostaneme-li nějaký algoritmus, vždy bychom si měli položit některé důležité otázky. Ku příkladu výše jsme zkoumali rychlost algoritmu, tedy kolik kroků potřebuje, než dospěje k výsledku. Avšak možná nejdůležitější otázkou je, zda se algoritmus vůbec někdy zastaví. U Eukleidova algoritmu a u bublinkového řazení jsme dokázali, že po nějakém čase se proces vždy zastaví. Existuje však obecný postup, kterým by šlo dokázat, že se algoritmus zastaví či nezastaví?

Problém zastavení zní takto: Máme před sebou algoritmus a nějakou jeho vstupní hodnotu (například dvě čísla, jejichž společného dělitele hledáme). Vložíme-li do algoritmu tuto vstupní hodnotu, zastaví se někdy nebo poběží donekonečna? A teď pojďme o úroveň výš – zkusme navrhnout obecný postup, který by pro daný algoritmus ukázal, zda se zastaví. To znamená navrhnout algoritmus, který toto rozhodne.

Bohužel (nebo naštěstí) se o to pokoušet nemusíme – nemělo by to cenu:

Věta. *Neexistuje žádný algoritmus, který by byl vždy schopen správně rozhodnout, zda se jiný algoritmus vůbec někdy zastaví při zadaném vstupu.*

Důkaz. Naznačíme jen základní myšlenku stojící za důkazem, nebudeme zde zabíhat do detailů. Důkaz využívá takzvanou *diagonální metodu*, která se často objevuje v teorii množin a můžeš ji znát třeba z Cantorova důkazu, že reálná čísla jsou nespočetná množina.

Pro spor připusťme, že existuje algoritmus $\text{Halt}(T, t)$, který pro daný vstup t rozhodne, zda se daný algoritmus T zastaví. Tedy $\text{Halt}(T, t) = 1$, pokud T zastaví při vstupu t, a $\text{Halt}(T, t) = 0$, pokud proces T při vstupu t poběží donekonečna.

Úmluva. Pokud do algoritmu vložíme vstup, který s ním není kompatibilní, předpokládáme, že algoritmus vyhodí chybovou hlášku a zastaví se.

Nyní definujme diagonální funkci $\text{Diagonal}(s)$ takto:

$$\text{Diagonal}(s) = 1, \quad \text{pokud } \text{Halt}(s, s) = 0,$$

zatímco v opačném případě, kdy $\text{Halt}(s, s) = 1$, se proces $\text{Diagonal}(s)$ zacyklí v nekonečné smyčce.

Poznámka. Podle naší úmluvy výše definice dává smysl. Existují algoritmy, které na vstupu přijímají jiný algoritmus. Proto je možné dát do Haltu jako vstup jiný algoritmus, a pokud jej Halt nezvládne zpracovat, prostě se zastaví.

[5] Viz Cvičení 7.

Je to vcelku zvláštní funkce, protože využívá algoritmu Halt pro rozhodnutí, zda algoritmus s zastaví „sám na sobě“. Nyní toto paradoxní odkazování na sebe sama pojďme pozvednout na vyšší úroveň – zeptejme se, co se stane, když do Diagonal vložíme jako vstupní hodnotu algoritmus Diagonal.

Jsou pouze dvě možnosti:

(1) Pokud Diagonal(Diagonal) = 1, znamená to, že Halt(Diagonal, Diagonal) = 0, tedy program Diagonal nikdy nezastaví, je-li vstupem Diagonal. To je ale ve sporu s tím, že Diagonal(Diagonal) = 1, tedy že Diagonal sám na sobě zastaví (a vyhodí hodnotu 1).

(2) Pokud Diagonal(Diagonal) nikdy nezastaví a zacyklí se v nekonečné smyčce, musí platit Halt(Diagonal, Diagonal) = 1. To znamená, že algoritmus Diagonal po nějakém čase zastaví, je-li na vstupu sám Diagonal. To je však spor s předchozím předpokladem, že se Diagonal(Diagonal) zacyklí v nekonečné smyčce.

V obou případech dojdeme ke sporu. Nějaký předpoklad, který jsme v průběhu použili, tedy nebyl pravdivý. A protože jediné tvrzení, jehož pravdivost jsme předpokládali, je to, že existuje algoritmus Halt, tak nutně žádný takový algoritmus neexistuje. Tím je důkaz hotov. □

Tuto větu dokázal v roce 1936 anglický matematik Alan Turing. O Turingově práci na rozluštění Enigmy (přístroje, kterým Němci za druhé světové války šifrovali tajné zprávy) se můžeš dozvědět například ve filmu Kód Enigmy (The Imitation Game) z roku 2014.

Ne vždy lze rozhodnout, zda se algoritmus zastaví, případně sice víme, že se zastaví, ale až po nepřiměřeně dlouhé době, přičemž přibližný výsledek dostaneme už za kratší čas. V praxi proto na vstupu často zadáváme, kolik iterací má algoritmus provést, či jak přesný výsledek potřebujeme (například přesnost na čtyři desetinná místa).

Poděkování a rozloučení

Gratulujeme, že ses dočetl(a) až sem! Věříme, že ses při čtení dozvěděl(a) něco nového a hlavně že Tě některá část seriálu podnítila k zamyšlení a hledání odpovědí na otázky, které Ti vyvstaly na mysli. Přejeme mnoho zdaru při řešení seriálových úloh!

Zároveň bychom rádi poděkovali všem orgům, kteří se na seriálu podíleli, a že jich nebylo poskrovnu. Zejména děkujeme Hedvice za jazykové korektury a užitečné postřehy, Matějovi za TEXnickou podporu, krásné obrázky a další nápady a Radovi za odborné korektury. Poděkování patří také Lence a Radkovi, kteří pro vás připravili čokoLeanovou výzvu. A samozřejmě děkujeme všem orgům, kteří pomáhali s výběrem a opravováním úloh.

Návody ke cvičením

2. Největším společným dělitelem je číslo 17. Který způsob byl v tomto případě rychlejší?

3. Použij Eukleidův algoritmus k zjištění, že jejich nejvyšší společný dělitel je 1.

4. $(x, y) = (-7, -14)$.

5. Jaký je největší společný dělitel čísel 72 a 117?

6. Uprav rovnost na $ax = -by$ a podívej se na to, čím jsou jednotlivé strany dělitelné. Jakého tvaru jsou pak x a y?

Odpověď je $x = nb$ a $y = -na$, kde n je celé číslo.

7. Dává smysl vždy porovnat nejmenší dvě karty, z každé hromádky jednu, a tu nižší umístit na následující místo v nové řadě. V nové řadě je potom n karet, tedy proběhlo $n - 1$ porovnávání.

Návody k úlohám

1. Tvrzení: Nechť $a(x)$ a $b(x)$ jsou dva polynomy. Potom existuje právě jedna dvojice polynomů $q(x)$ a $r(x)$ taková, že $a(x) = q(x)b(x) + r(x)$, přičemž stupeň $r(x)$ je ostře nižší než řád $b(x)$.

Důkaz je obdobou důkazu dělicího algoritmu pro přirozená čísla.

2. Nejprve vyděl první mnohočlen druhým, poté pokračuj v duchu Eukleidova algoritmu.

3. Pomocí Eukleidova algoritmu najdi největšího společného dělitele $x^2 + x - 20$. Kořeny tohoto kvadratického polynomu již snadno spočítáš.

4. $(x, y) = (8 - 17n, -151 + 321n)$, kde $n \in \mathbb{Z}$.

5. $(x, y) = (75 - 52n, 486 + 337n)$, kde $n \in \mathbb{Z}$.

6. Použij Eukleidův algoritmus pozpátku.

7. Použij Bézoutovo lemma.

8. Použij Bézoutovo lemma, rovnosti vynásob a uvaž, co musí splňovat jakýkoli společný dělitel ab a c.

9. Použij indukci podle k.

10. $P(1) = 0$ a $P(n) = P(n-1) + n$.