Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Kolmogorovská zložitosť a jej využitie

(Diplomová práca)

Bc. Peter Lenčéš

Študijný odbor: 9.2.1. Informatika
Vedúci: doc. RNDr. Dana Pardubská, PhD.
Bratislava, 2010

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Ďakujem svojej školiteľke doc. RNDr. Dane Pardubskej, PhD. za spoluprácu pri písaní práce. Ďalej sa chcem touto cestou poďakovať Džessikovi za jeho ústretovosť, Knuthovi za TEX, Kolmogorovovi za zložitosť a hlavne svojej rodine za podporu počas celého štúdia.

Abstrakt

V práci predstavujeme Kolmogorovskú zložitosť ako zaujímavý prípad popisnej zložitosti a ukazujeme jej vlastnosti a využitie na konkrétnych príkladoch. Na začiatku uvádzame potrebné teoretické základy, v každej kapitole vysvetľujeme teóriu Kolmogorovskej zložitosti, ktorú vzápätí aplikujeme pri riešení príkladov. Práca obsahuje množstvo príkladov, ktoré pomáhajú lepšie pochopiť vedomosti nadobudnuté v teoretickej časti a dopĺňajú ich. Nachádzajú sa v nej riešené príklady ako aj úlohy na precvičenie pre čitateľa.

Kľúčové slová: Kolmogorovská zložitosť, najkratší popis, popisná zložitosť, nestlačiteľnosť, metóda nestlačiteľnosti

Abstract

In the thesis we introduce Kolmogorov complexity as an interesting instance of descriptional complexity and we show some of its properties and application on concrete examples. First we provide essential preliminaries, in each chapter we explain the theory of Kolmogorov complexity which we apply in the examples later on. The thesis contains lots of examples that help in better understanding of the knowledge acquired in the theoretical sections and they make it up. There are examples with solutions as well as exercises meant for reader for practising in the paper.

Keywords: Kolmogorov complexity, shortest description, descriptional complexity, incompressibility, incompressibility method

Obsah

Kapitola 1

Úvod

Tento text vzniká ako diplomová práca na Fakulte matematiky, fyziky a informatiky Univerzity Komenského v Bratislave. Jeho hlavným cieľom je predstaviť zaujímavé aspekty Kolmogorovskej zložitosti a ilustrovať ich na riešených i neriešených príkladoch. Každá kapitola je zameraná na rôzne vlastnosti a aplikácie Kolmogorovskej zložitosti. Po vysvetlení teórie nasledujú riešené príklady objasňujúce a dopĺňajúce podanú teóriu a úlohy určené pre čitateľa, na ktorých si môže precvičiť nadobudnuté vedomosti.

Našou ambíciou pri písaní práce je pomôcť všetkým, ktorí sa o túto oblasť zaujímajú a chcú si vedomosti precvičiť aj na konkrétnych príkladoch. Silnou motiváciou pre písanie tejto práce je aj fakt, že slovenská literatúra s podobným obsahom neexistuje. Veríme, že sa nám toto prázdne miesto podarí zaplniť práve touto prácou, a že bude naozaj slúžiť svojmu účelu.

Práca je určená pre všetkých, ktorí sa chcú naučiť teóriu Kolmogorovskej zložitosti a naučiť sa ju aplikovať, primárne však poslucháčom vysokoškolského štúdia informatiky, ktorým môže čiastočne pomôcť aj ako doplnkový študijný materiál na predmet Kolmogorovská zložitosť prednášaný na našej fakulte. Od čitateľa sa očakávajú základné znalosti z matematiky a teoretickej informatiky. Problematiku Kolmogorovskej zložitosti rozoberáme od základov, žiadne skúsenosti s týmto pojmom preto nie sú vyžadované. V prípade, že má čitateľ záujem o obšírny výklad potrebných teoretických základov ako

aj samotnej teórie Kolmogorovskej zložitosti, odporúčame na štúdium rozsiahle dielo [16] (v anglickom jazyku) z pera Paula Vitányiho a Ming Liho, expertov v tejto oblasti. Pri písaní sme aj my neraz čerpali práve z uvedenej publikácie. Vhodným štartovacím bodom pri štúdiu môže byť taktiež bakalárska práca [8] obsahujúca aj niekoľko veľmi zaujímavých aplikácií KZ, po ktorej sme pri písaní tohto textu siahli tiež.

V úvodnej kapitole stanovujeme cieľ diplomovej práce, popisujeme motiváciu a členenie práce. V druhej kapitole vysvetľujeme všeobecné teoretické základy potrebné na pochopenie problematiky vysvetľovanej v ďalších častiach práce a uvádzame pojem popisná zložitosť. Každá nasledujúca kapitola je rozdelená na teoretickú časť a na praktickú časť obsahujúcu príklady a úlohy. V kapitole číslo 3 zavádzame a definujeme Kolmogorovskú zložitosť a vysvetľujeme jej základné vlastnosti. V nasledujúcej kapitole pracujeme s pojmom *nestlačiteľnosť* reťazca a popisujeme metódu nestlačiteľnosti. Piata kapitola pracuje s Kolmogorovskou zložitosťou ako s nevypočítateľnou celočíselnou funkciou. Náhodné postupnosti a testy náhodnosti spomíname v šiestej kapitole. Kapitola číslo 7 opisuje veľmi zaujímavé a netriviálne aplikácie Kolmogorovskej zložitosti, pričom v nej ukážeme dôkazy odhadu počtu farieb potrebných pri hranovom farbení grafu a odhadu zložitosti triediaceho algoritmu v priemernom prípade využívajúce Kolmogorovskú zložitosť. Posledná – ôsma kapitola tvorí záver tejto práce, zhŕňa jej obsah, popisuje prínos a navrhuje možnosti jej rozšírenia. Nasleduje zoznam obrázkov a zoznam použitej literatúry.

Kolmogorovská zložitosť je „krásna, hlboká a dôležitá“ časť teoretickej informatiky[1]. Teória Kolmogorovskej zložitosti sa zaoberá najmä zložitosťou reťazcov a náhodnými reťazcami. Za kritérium zložitosti reťazca je zvolená dĺžka jeho popisu. Neformálne môžeme Kolmogorovskú zložitosť reťazca definovať ako dĺžku jeho najkratšieho popisu vzhľadom k nejakej popisnej me-

[1] Ako sa nechal počuť Juris Hartmanis, držiteľ Turingovej ceny z roku 1993.

tóde. Reťazce s dlhými popismi majú pre nás špeciálny význam, keďže ich vysokú zložitosť budeme v mnohých prípadoch vedieť využiť vo svoj prospech. Kolmogorovská zložitosť poskytuje aj netradičnú definíciu náhodných reťazcov. Bohatá teória Kolmogorovskej zložitosti nie je samoúčelná a ukazuje sa ako veľmi silný, užitočný a praktický nástroj použiteľný v mnohých oblastiach informatiky a matematiky.

Kapitola 2

Teoretické základy

2.1 Stručný úvod do problematiky

Pod pojmom *zložitosť* zväčša rozumieme zložitosť nejakého algoritmu, či už časovú alebo priestorovú. Časovou zložitosťou rozumieme počet inštrukcií (priradení, porovnaní), ktoré algoritmus vykoná. Pri Turingovych strojoch[1] to môže byť napríklad počet presunov hlavy o jednu pozíciu ľubovoľným smerom alebo zápis znaku na pásku. Čo sa priestorovej zložitosti týka, určuje ju počet použitých polí na páske Turingovho stroja.

Zložitosť ako taká je jednou z vlastností každého objektu. Všeobecne nás preto môže zaujímať zložitosť ľubovoľného objektu, či už je to algoritmus, graf, problém, jazyk, reťazec, množina... Aby sme sa zložitosťou niektorého objektu mohli zaoberať, potrebujeme poznať presnú definíciu zložitosti objektu, aby sme túto vlastnosť vedeli jednoznačne určiť a vyhodnotiť. Ak chceme určovať zložitosti rôznych typov objektov, potrebujeme taktiež definovať ich jednotnú reprezentáciu, aby sme s nimi vôbec mohli pracovať. V tejto práci budeme všetky objekty reprezentovať ako binárne reťazce. Budeme potrebovať rozumné kódovanie objektov, aby bolo jasné aký objekt daný reťazec reprezentuje. Reťazcom a ich kódovaniu bude preto venovaná

[1] Model Turingovho stroja, s ktorým budeme pracovať, definujeme neskôr.

celá podkapitola 2.2.

Už vieme, že objekty budeme reprezentovať reťazcami. Zložitosť týchto objektov bude preto daná zložitosťou ich reťazcovej reprezentácie. Ako ale určiť zložitosť reťazca? Čo to vôbec je zložitosť reťazca? Odpoveďou na tieto otázky bude práve *Kolmogorovská zložitosť*.

Na určenie zložitosti reťazca sa ako prvá možnosť intuitívne ponúka jeho dĺžka. Je to však rozumné kritérium? Všetky krátke reťazce vyhlásime za jednoduché, všetky dlhé budeme považovať za zložité. Najzložitejšími budú nekonečné reťazce. Znamená to, že napr. všetky čísla s neukončeným desatinným rozvojom[2] budú mať „nekonečnú“ zložitosť. Zamyslime sa, je to vhodné kritérium?

Ďalším kritériom na určenie zložitosti reťazca môže byť dĺžka výpočtu tohto reťazca. Nech je to napr. časová zložitosť algoritmu, ktorý tento reťazec vypíše. Zrejme existuje mnoho algoritmov, ktoré dokážu daný reťazec vypísať na výstupe. Ktorý z nich si vyberieme? Môžeme si zvoliť napríklad ten s najmenšou časovou zložitosťou. Nevznikajú výberom takéhoto algoritmu nejaké nežiaduce prekážky? Takto zvolený algoritmus môže byť nepríjemne dlhý, problémom môže byť aj jeho priestorová zložitosť.

Na druhej strane si môžeme zvoliť najkratší taký algoritmus, ktorý vypíše daný reťazec. Ten však môže mať neakceptovateľnú časovú (či priestorovú) zložitosť.

Zvoľme posledný spomínaný prístup, a síce zložitosť reťazca nech je určená dĺžkou jeho najkratšieho popisu. Môžeme povedať, že takto definovaná zložitosť je príkladom *popisnej zložitosti*. Takáto definícia má blízko aj k intuitívnemu pohľadu na komplikovanosť reťazca. Reťazce tvaru napr. $00000\ldots$, $01010101010\ldots$ alebo $0101101110111101111110\ldots$ sú nekonečné, no intuícia

[2]Čísla zapisujeme v dohodnutej číselnej sústave. Prirodzene sme zvyknutí písať čísla v desiatkovej sústave. Každé číslo môžeme v tom prípade chápať ako reťazec nad abecedou $\Sigma = \{0, 1, 2, \ldots, 9\}$. Čísla zapísané v binárnej sústave budú potom reťazce nad dvojprvkovou abecedou $\Sigma' = \{0, 1\}$.

hovorí, že nie sú komplikované alebo zložité. Ich popisy sú veľmi jednoduché a krátke. Prvý môžeme slovne popísať ako „píš samé nuly“. Ďalší vyjadríme ako „striedaj po jednom nuly a jednotky“. Zdá sa, že čím komplikovanejší sa reťazec javí, tým ťažšie sa popíše, resp. tým dlhší popis potrebujeme. Ak sa v reťazci budú znaky (prvky abecedy, písmená) striedať nám neznámym spôsobom, budeme ho považovať (intuitívne) za komplikovaný/zložitý. Nebudeme schopní nájsť žiadny jeho popis.

Pri týchto úvahách je pre nás na ilustráciu vhodným príkladom číslo $\pi = 3.1415926535\ldots$ Pri pohľade naň nevieme ľahko vyjadriť ako sa jednotlivé cifry jeho rozvoja striedajú. Nevidíme medzi nimi súvis, na základe jednej nevieme povedať aká bude nasledovná. π sa ako reťazec javí byť naozaj zložitý. Číslo π ale vieme exaktne popísať a poznáme postupy ako zistiť jeho presnú hodnotu.[3]

Jedným z jeho vyjadrení je napr. Bailey-Borwein-Plouffeov[4] vzorec

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \tag{2.1}$$

Ako vidíme, napriek dĺžke čísla π a jeho očakávanej zložitosti je jeho popis krátky. Za „krátkosť“ tohto popisu však vďačíme aj tomu, že máme krátky popis nekonečna „∞“, ktorý v tomto popise využívame.

Úloha 2.1.1. Uvedený vzorec je síce popisom čísla π, no pri pohľade naň nie je vôbec zrejmé, že je to naozaj tak. Odsimulujte preto aspoň niekoľko počiatočných krokov algoritmu využívajúceho uvedený vzorec na výpočet čísla π.

Koľko iterácií potrebujeme vykonať, aby sme získali prvých 10 platných cifier desatinného rozvoja π?

[3] Všimnime si, že v tomto konkrétnom prípade na popis čísla $3.1415926535\ldots$ stačí triválne krátky popis, a síce je to „π“, jeden všeobecne uznávaný symbol reprezentujúci práve túto konštantu.

[4] Simon Plouffe, [1] (1995). Napriek tomu, že tento vzorec našiel Plouffe, je pomenovaný podľa autorov článku, v ktorom bol vzorec uverejnený.

Z nášho nového pohľadu, z pohľadu dĺžky najkratšieho existujúceho popisu, sa π ukazuje ako pomerne jednoduchý reťazec. Môže sa ale stať, že narazíme na reťazec, ktorý nebudeme vedieť krátko popísať. Z pohľadu popisnej zložitosti to bude veľmi zložitý reťazec.

Koncept, ktorý sme uviedli, opisuje Kolmogorovskú zložitosť ako špeciálny prípad popisnej zložitosti. KZ reťazca je definovaná ako dĺžka najkratšieho možného popisu tohto reťazca. Ako sme už spomínali, voľba najkratšieho popisu reťazca (algoritmu na výpočet reťazca), môže mať za následky extrémnu časovú a priestorovú zložitosť. Od tohto faktu však abstrahujeme, keďže Kolmogorovská zložitosť ponúka *teoretický* pohľad na zložitosť reťazcov a nezaujímame sa o zdroje potrebné na výpočet[5]. Pri uvedenej definícii zložitosti reťazca sa vynárajú na povrch aj iné otázky, ako napr. „Ako budeme jednotlivé popisy vyjadrovať?“ alebo „Ako budeme merať dĺžku ich popisu?“ Tieto, ale i mnohé ďalšie budú zodpovedané v priebehu textu.

2.2 Reťazce a ich kódovanie

V nasledujúcej časti sa budeme venovať zápisu reťazcov a ich kódovaniu v binárnej abecede. Nebudeme však hovoriť o tom, ako objektu priradiť reťazec, ktorý ho bude reprezentovať.

Čísla môžeme zapisovať v rôznych číselných sústavách. My budeme pracovať s číslami v binárnom tvare. Pre ľubovoľnú abecedu Γ vieme nájsť také kódovanie, že každé písmeno abecedy Γ zakódujeme prvkami z $\{0,1\}^*$. Zápis ľubovoľného čísla potom môžeme považovať za reťazec (slovo) nad abecedou $\Sigma = \{0,1\}$ a naopak, každý reťazec nad abecedou Σ môžeme chápať ako zápis čísla v binárnej sústave. Pojmy *číslo* a *reťazec* môžeme preto podľa kontextu zamieňať. Na popis všetkých objektov budeme využívať binárne re-

[5]Existuje aj definícia Kolmogorovskej zložitosti s obmedzenými prostriedkami, v našej práci sa ňou však nezaoberáme. Môžete sa o nej dočítať v [16] (kapitola 7) alebo v [7]

ťazce. Každý objekt preto patrí do $\{0,1\}^*$.

Mohutnosť množiny M budeme označovať $|M|$. Keďže budeme pracovať nad abecedou $\Sigma = \{0,1\}$ a platí $|\Sigma| = 2$, budeme používať logaritmus pri základe 2. Zápisom $\log x$ preto rozumieme $\log_2 x$, pokiaľ nebude uvedené inak. Dĺžku reťazca x ako aj dĺžku binárneho zápisu čísla reprezentovaného reťazcom x budeme označovať $l(x)$. Určuje ju počet bitov zápisu reťazca x, resp. počet bitov (cifier) čísla x. Dĺžka zápisu čísla aj reťazca x sa potom rovná

$$\lfloor \log x \rfloor + 1,$$

kde počítame logaritmus čísla (nie reťazca) x, keďže logaritmus je funkcia definovaná na číslach, nie na reťazcoch.

Zaokrúhlenie hodnoty $\lfloor \log x \rfloor + 1$ na $\log x$ zrejme spôsobuje rozdiel najviac jedného bitu v dĺžke, tento rozdiel preto zanedbáme a budeme písať $l(x) = \log x$.

i-ty bit reťazca x označujeme x_i. Podreťazec x, ktorý budú tvoriť bity x od i-teho po j-ty, označíme $x_{i:j}$. Celý n-bitový reťazec x môžeme označiť $x_{1:n}$, zrejme však $x = x_{1:n}$.

Hovorili sme, že všetky objekty budeme kódovať do binárnych reťazcov. Platí to aj o prirodzených číslach.
Usporiadajme všetky binárne reťazce lexikograficky:

$$\{0,1\}^* = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$$

Každé prirodzené číslo reprezentujeme príslušným binárnym reťazcom podľa jeho poradia v lexikografickom usporiadaní nasledovne

$$(\epsilon, 0), (0,1), (1,2), (00,3), (01,4), (10,5), (11,6), (000,7), \ldots \tag{2.2}$$

Znamená to, že '8' a '001' majú ten istý význam, keďže jedno je reprezentáciou druhého. V dôsledku toho, že reťazce alebo čísla '8' a '001' reprezentujú ten istý objekt, môžeme dokonca písať $8 = 001$. Vďaka takejto reprezentácii prirodzených čísiel sme schopní ušetriť časť bitov potrebných na ich zápis,

keďže v štandardnej binárnej sústave, na rozdiel od práve uvedenej reprezentácie, nie všetky reťazce reprezentujú prirodzené číslo.

Úloha 2.2.1. Reprezentujú reťazce 10, 010 a 0010 to isté prirodzené číslo? Prečo?

Úloha 2.2.2. Pokúste sa nájsť vzorec, pomocou ktorého zistíme, aké prirodzené číslo reprezentuje binárny reťazec $a_n a_{n-1} \dots a_0$.

Pri navrhovaní kódovania reťazcov si musíme uvedomiť, že niekedy budeme potrebovať kódovať jedným reťazcom viacero objektov. Kódovaním rozumieme jednoznačné zobrazenie z množiny *zdrojových*[6] (pôvodných, východiskových) slov do množiny *kódových* slov. Kódovanie musíme navrhnúť tak, aby sme z reťazca vedeli jednoznačne vyčítať aký objekt kóduje. Kód musí túto podmienku spĺňať aj v prípade, že daný reťazec kóduje viacero objektov.

Chceme nájsť funkciu, pomocou ktorej budeme kódovať zdrojové slová do množiny kódových slov ($\{0,1\}^* \to \{0,1\}^*$) tak, aby sme kódové slová spätne vedeli jednoznačne dekódovať.

Uvažujme funkciu $D : \{0,1\}^* \to \mathbb{N}$. Funkciu D voláme *dekódovacia*, ak jej definičný obor tvorí množina *kódových slov* a obor hodnôt tvoria *zdrojové* slová. Ak $D(y) = x$, hovoríme, že y je kódové slovo pre pôvodné slovo x. Pre inverzné zobrazenie k D potom platí $D^{-1}(x) = \{y : D(y) = x\}$. Uvedomme si, že kódovanie $E = D^{-1}$ ešte nemusí byť funkcia.

Definícia 2.2.1. Množina reťazcov M je *bezprefixová* práve vtedy, keď žiadny z jej prvkov nie je prefixom žiadneho iného jej prvku.

Definícia 2.2.2. Hovoríme, že funkcia $E : \{0,1\}^* \to \{0,1\}^*$ definuje *prefixový kód*, ak jej obor hodnôt tvorí bezprefixovú množinu.

[6] source word

Ak kódovacia funkcia E definuje prefixový kód, $D = E^{-1}$ je príslušná dekódovacia funkcia. Z definície 2.2.2 vyplýva, že definičným oborom dekódovacej funkcie je bezprefixová množina. Vďaka tomu dokážeme jednoznačne rozhodnúť, ktoré slovo y bolo zakódované na slovo x.

Zafixujme kódovaciu funkciu E, ktorú budeme odteraz používať.

Definícia 2.2.3. Kódovacia funkcia $E : \{0,1\}^* \to \{0,1\}^*$ je definovaná nasledovne.

$$\begin{aligned} E_0(x) &= 1^x 0 \\ E_i(x) &= E_{i-1}\left(l\left(x\right)\right) x \quad \text{pre } i > 0 \end{aligned}$$

Aby bol výraz 1^x korektný, x v definícii $E_0(x)$ reprezentuje prirodzené číslo, nie reťazec. Ukážme si ako funkcia E funguje na niekoľkých príkladoch. Začnime najjednoduchším prípadom a vypočítajme hodnotu $E_0(5)$. Zistíme tak kódové slovo pre reťazec „5“. Úloha je zatiaľ triválna, $E_0(5) = 1^5 0 = 111110$. Keďže $5 = 10$ (viď. 2.2), môžeme písať tiež $E_0(10) = 1^5 0 = 111110$.
Ďalej počítajme $E_1(96)$.

$$\begin{aligned} 96 &= 100001 \\ E_1(96) &= E_1(100001) = E_0\left(l\left(100001\right)\right) 100001 = E_0(\mathbf{11})100001 = \\ &= 1^6 0100001 = 1111110100001 \end{aligned}$$

Zvýraznený reťazec **11** reprezentuje dĺžku reťazca 100001, platí $l(100001) = 6 = 11$.

Kódovanie $E_1(x)$ budeme v celej práci využívať ako štandardné kódovanie, pokiaľ nebude uvedené inak. Vidíme, že reťazec kódovaný funkciou E_1 má tvar $E_1(x) = 1^{l(x)}0x$. Tento tvar nazývame *samooddeľujúci*[7] tvar reťazca. Reťazec x v samooddeľujúcom tvare budeme označovať $\overline{x}$. O dĺžke reťazca v samooddeľujúcom tvare platí

$$l(E_1(x)) = l(\overline{x}) = l(1^{l(x)}0x) = 2l(x) + 1. \tag{2.3}$$

[7] z anglického self-delimiting

Na ilustráciu si ešte predvedieme kódovanie E_2 toho istého reťazca, tj. „96“.

$$\begin{aligned} E_2(96) &= E_2(100001) = E_1(l(100001))100001 = E_1(11)100001 = \\ &= E_0(l(11))11100001 = E_0(1)11100001 = \mathbf{11011}100001 \end{aligned}$$

Prvé dve zvýraznené jednotky hovoria, že za nulou za nimi nasledujú dva bity popisujúce dĺžku kódovaného čísla (100001). Nasledujú spomínané dva bity (**11**) kódujúce dĺžku kódovaného čísla ($11 = 6 = l(100001)$). Posledných 6 bitov tvorí samotné kódované číslo 100001.
Pozorný čitateľ si mohol všimnúť, že platí

$$E_2(x) = \overline{l(x)}x. \tag{2.4}$$

Dĺžku reťazca zakódovaného funkciou E_2 môžeme ľahko vypočítať:

$$l(E_2(x)) = l(\overline{l(x)}x) = 2l\left(l\left(x\right)\right) + l(x) + 1 \tag{2.5}$$

Toto kódovanie využívame v prípade, že potrebujeme kratší prefixový kód. Môžeme použiť aj kódovanie $E_i(x)$ pre $i > 2$, ak máme naozaj dlhé reťazce a potrebujeme skrátiť dĺžky kódových slov. Nám však vo väčšine prípadov bude postačovať kódovanie E_1. Ak použijeme iné kódovanie, bude na to čitateľ upozornený.

Teraz si ukážeme ako budeme kódovať viacero slov jedným reťazcom. Predstavme si, že chceme jedným reťazcom reprezentovať slová x a y. Takýto reťazec budeme označovať $\langle x, y\rangle$. Definujme si tento zápis formálne.

Definícia 2.2.4. $\langle x, y\rangle = \overline{x}y = 1^{l(x)}0xy$

Zrejme potom platí $l\left(\langle x, y\rangle\right) = 2l(x) + l(y) + 1$.
Na samooddeľujúci zápis troch reťazcov použijeme $\langle x, \langle y, z\rangle\rangle$.

$$\langle x, \langle y, z\rangle\rangle = \langle x, \bar{y}z\rangle = \bar{x}\bar{y}z = 1^{l(x)}0x1^{l(y)}0yz \tag{2.6}$$

Potom $l(\langle x, \langle y, z\rangle\rangle) = 2l(x) + 2l(y) + l(z) + 2$.

Úloha 2.2.3. Definujme kódovanie pomocou kódovacej funkcie $E' : \{x, y, z\} \to \{0, 1\}^+$ nasledovne.

$$\begin{aligned} E'(x) &= 0 \\ E'(y) &= 01 \\ E'(z) &= 001 \end{aligned}$$

Definuje kódovanie E' prefixový kód?
Ako by sme dekódovali slovo 0001?

Úloha 2.2.4. Čomu sa rovná $E_2(16)$?

Úloha 2.2.5. Zapíšte reťazce $x = 3, y = 7, z = 12$ ako jeden reťazec v samoodeľujúcom tvare.

2.3 Turingov stroj

Alan Mathison Turing vo svojej slávnej práci [13] z roku 1936 predstavil teoretický model stroja, ktorý si kládol za cieľ pomôcť preskúmať hranice vypočítateľnosti. Voláme ho *Turingov stroj* (TS). Predpokladáme, že čitateľ sa s týmto pojmom už stretol a dokáže s ním pracovať. Keďže existuje viacero definícií TS, aby sme predišli možným nezrovnalostiam, rozhodli sme sa uviesť aj formálnu definíciu modelu TS, s ktorým budeme pracovať.

Základnými „komponentmi“ TS sú riadiaca jednotka, vstupno/výstupná páska a hlava schopná čítať a zapisovať symboly z pásky a na pásku. Formálne môžeme Turingov stroj definovať ako šesticu $(Q, \Sigma, \Gamma, \delta, q_0, F)$, kde Q je konečná množina stavov, v ktorých sa TS môže nachádzať, Σ je konečná abeceda vstupných symbolov, ktoré tvoria vstup TS, Γ je konečná abeceda pracovných symbolov, ktoré na pásku hlava môže zapisovať ($\Sigma \subseteq \Gamma$), δ je prechodová (riadiaca) funkcia, reprezentujúca riadiacu jednotku, $q_0 \in Q$ je špeciálny počiatočný stav, v ktorom TS začína svoj výpočet a $F \subseteq Q$ je konečná množina akceptačných stavov, do ktorých keď stroj prejde, ukončí svoj výpočet. Popíšme si ako Turingov stroj funguje.

Turingov stroj pracuje v krokoch. Vykonanie jedného kroku mu trvá jednu časovú jednotku (časové kvantum). Čas je diskrétny, jednotlivé časové kvantá môžeme označiť $0, 1, 2, \ldots$ V každom čase výpočtu sa TS nachádza v nejakom stave $q \in Q$. Páska je rozdelená na políčka alebo bunky. Na každej pozícii sa môže nachádzať jeden symbol pracovnej abecedy. Vstup pre TS tvorí súvislá postupnosť symbolov vstupnej abecedy na páske, zľava i zprava ohraničených prázdnymi políčkami, ktoré budeme označovať B (Blank, prázdny). Vstupnú abecedu budú spravidla tvoriť symboly $\{0, 1, B\}$. Páska je v jednom smere (zľava doprava) potenciálne nekonečná, avšak v každom okamihu je na nej zapísaný len konečný, hoci ľubovoľne veľký počet symbolov. Znamená to, že páska nás neobmedzuje a poskytuje (potenciálne) nekonečne veľkú pamäť.

TS začína svoj výpočet v čase 0 nachádzajúc sa v odlíšenom počiatočnom stave. Hlava sa nachádza nad prvým vstupným symbolom, tj. nad najľavejším symbolom vstupnej postupnosti. Popíšme si teraz čo všetko spraví TS počas jedného kroku výpočtu.

Hlava prečíta znak na políčku, nad ktorým sa práve nachádza. Na základe tohto symbolu a stavu, v ktorom sa TS nachádza, riadiaca jednotka reprezentovaná δ funkciou rozhodne aký znak pracovnej abecedy hlava zapíše. Po zapísaní symbolu sa hlava presunie vľavo, vpravo alebo zostane na aktuálnej pozícii. V jednom kroku môže hlava vykonať najviac jeden posun o jednu pozíciu. Na záver kroku môže TS prejsť do iného stavu.

Riadiacu δ funkciu môžeme formálne deklarovať

$$\delta : (Q \times \Gamma) \rightarrow (\Gamma \times \{-1, 0, 1\} \times Q),$$

kde hodnoty z $\{-1, 0, 1\}$ reprezentujú posun hlavy doľava, žiadny posun a posun doprava.

Nech je prechodová funkcia pre TS T v stave q čítajúc páskový symbol a definovaná nasledovne.

$$\delta(q, a) = (b, 1, q_F)$$

Tento zápis interpretujeme tak, že ak T v stave q číta z pásky a, zapíše na túto pozíciu symbol b, posunie sa o jednu pozíciu doprava a prejde do stavu q_F.

V prípade, že TS číta v stave q symbol a a prechodová funkcia nie je pre túto usporiadanú dvojicu $(\text{stav}, \text{symbol})$ definovaná, TS sa zasekne. Ak počas výpočtu prejde do akceptačneho stavu, korektne ukončí svoj výpočet. Hovoríme, že TS akceptuje vstupné slovo. Po ukončení výpočtu sa na páske nachádza výstup TS. Posledná možnosť, ktorá môže nastať je, že TS nikdy nedosiahne akceptačný stav, zacyklí sa a jeho výpočet sa nikdy neskončí.

V tejto práci budeme často pracovať s TS, ktorých vstupom bude program p. Výstup TS T s programom p budeme zapisovať $T(p)$ a budeme hovoriť, že p je program pre T. Tento koncept môže pôsobiť trošku mätúco, keďže TS je sám o sebe vlastne program a je riadený prechodovou funkciou, ktorá je jeho „programovou“ zložkou. Vysvetlenie je prosté – TS T môžeme chápať ako počítač, na ktorom sa vykoná program p.

2.3.1 Univerzálny Turingov stroj

V kapitole 2.2 o reťazcoch sme spomínali, že všetky objekty reprezentujeme binárnymi reťazcami. Turingove stroje nie sú výnimkou. Predpokladáme preto, že všetky TS, s ktorými pracujeme, budú tiež kódované do binárnych reťazcov.[8]

Úloha 2.3.1. Vedeli by ste navrhnúť nejaké efektívne kódovanie TS do binárnych reťazcov?

Reťazcová reprezentácia nám umožňuje ľahšie enumerovať Turingove stroje. Vieme generovať binárne reťazce, napr. v lexikografickom poradí a zároveň kontrolovať, či daný reťazec je kódom nejakého TS. Ak je, zaradíme ho do enumerácie s príslušným poradovým číslom. Takto získame jednu z možných efektívnych enumerácií Turingovych strojov $T_0, T_1, T_2, \ldots$ Na špecifiáciu konkrétneho TS T_i potom stačí uviesť jeho poradové číslo v tejto enumerácii, čo vyžaduje $\log i$ bitov plus popis enumerácie namiesto celého zápisu TS T_i.

[8] Detailami tohto kódovania sa nebudeme zaoberať.

Označme si U Turingov stroj, ktorý pracuje tak, že na začiatku prečíta zo vstupu číslo i, čo je poradové číslo TS T_i a vstup p pre T_i a ďalej simuluje tento T_i na vstupe p. TS U nazývame *univerzálny* Turingov stroj (UTS). Je univerzálny práve preto, že dokáže simulovať ľubovoľný TS z danej enumerácie.
Platí

$$U(\langle i, p\rangle) = T_i(p) \tag{2.7}$$

Znamená to, že výstup univerzálneho (simulujúceho) stroja je rovnaký ako výstup pôvodného (simulovaného) stroja. Vstupom pre UTS U je reťazec

$$\langle i, p\rangle = \bar{i}p = 1^{l(i)}0ip, \tag{2.8}$$

z ktorého vie dekódovať aj číslo i Turingovho stroja T_i, ktorý má simulovať, aj program p, ktorý má vykonať.
V prípade, že program p má vlastný vstup y, píšeme

$$U(\langle y, \langle i, p\rangle\rangle) = T_i(\langle y, p\rangle). \tag{2.9}$$

Univerzálnych TS existuje v skutočnosti nekonečne veľa. V ďalšej kapitole ukážeme, že rozdiely medzi týmito strojmi sú z pohľadu popisnej zložitosti, zanedbateľné. Zo všetkých univerzálnych TS preto zvolíme jeden (ľubovoľný) *referenčný* UTS, ktorý budeme používať. Bez ujmy na všeobecnosti budeme predpokladať, že zvolený referenčný UTS je T_0 v našej efektívnej enumerácii Turingovych strojov.

Popri Turingovych strojoch existujú aj iné, rovnako silné výpočtové nástroje. Jedným z nich sú *čiastočne rekurzívne funkcie*.[9] Každý TS v skutočnosti počíta nejakú čiastočne rekurzívnu funkciu (ČRF) a pre každú ČRF existuje TS, ktorý ju počíta, tieto modely sú ekvivalentné. Efektívnej enumerácii TS zodpovedá efektívna enumerácia čiastočne rekurzívnych funkcií

$$\phi_0, \phi_1, \phi_2, \ldots$$

[9]Sú to funkcie, ktoré síce nie sú definované pre všetky vstupné slová, no na tých, pre ktoré sú definované, sú vypočítateľné.

i-tej ČRF ϕ_i v tejto enumerácii zodpovedá i-ty TS T_i. Univerzálnemu TS $U = T_0$ zodpovedá ČRF ϕ_0 univerzálna pre triedu všetkých ČRF. Platí preto

$$\phi_0(\langle y, \langle i, p\rangle\rangle) = \phi_i(\langle y, p\rangle). \tag{2.10}$$

Keďže Kolmogorovská zložitosť je popisnou zložitosťou, týmto funkciám budeme tiež hovoriť *popisné metódy*.

V tejto práci budeme aplikovať oba prístupy – Turingove stroje aj čiastočne rekurzívne funkcie.

2.3.2 Problém zastavenia

V tejto časti práce si ukážeme, že existujú problémy, pre ktoré neexistuje žiadny algoritmus, ktorý by ich riešil. Jedným z takýchto problémov je pre všetky programy a všetky vstupy rozhodnúť, ktoré programy na danom vstupe zastavia, a ktoré programy na danom vstupe nezastavia. Tomuto problému hovoríme aj *problém zastavenia*.

Veta 2.3.1. *Nech $\phi_1, \phi_2, \phi_3, \ldots$ je štandardná enumeráca čiastočne rekurzívnych funkcií. Potom neexistuje rekurzívna funkcia h definovaná ako*

$$h(x,y) = \begin{cases} 1 & \text{ak } \phi_x(y) \text{ je definovaná} \\ 0 & \text{inak} \end{cases}$$

pre všetky x a y.

Dôkaz. Tvrdenie dokážeme *sporom*. Predpokladajme, že takto definovaná rekurzívna funkcia h existuje.

Definujme čiastočne rekurzívnu funkciu ψ. Nech pre ňu platí $\psi(x) = 1$, ak $h(x,x) = 0$, inak nech je funkcia ψ nedefinovaná. Vďaka tomu, že funkcia h je totálna rekurzívna, vieme takto definovanú funkciu ψ skonštruovať. Keďže funkcia ψ je čiastočne rekurzívna, nachádza sa v enumerácii všetkých čiastočne rekurzívnych funkcií. Nech jej poradové číslo v tejto enumerácii je y, tzn. $\psi = \phi_y$. Z definície ψ vidíme, že funkcia $\phi_y(y)$ je definovaná práve vtedy, keď $h(y,y) = 0$.

Funkcia $h(y, y)$ je ale definovaná tak, že $h(y, y) = 1$ práve vtedy, keď je $\phi_y(y)$ definovaná. Dostali sme *spor*. Keďže všetky kroku dôkazu boli korektné, chyba bola v predpoklade. Takto definovaná funkcia h preto neexistuje. □

Problém zastavenia je *nerozhodnuteľný*.

2.4 Asymptotická notácia

V priebehu celého textu budeme často využívať asymptotickú notáciu.

Definícia 2.4.1. Nech f a g sú funkcie na reálnych číslach. Definujme nasledujúce asymptotiky.

$$\begin{aligned}
\mathrm{O}\left(\mathrm{g(n)}\right) &= \{f(n) \mid (\exists c > 0)(\exists n_0 > 0)(\forall n > n_0)\ 0 \leq f(n) \leq cg(n)\}, \\
\Omega\left(g(n)\right) &= \{f(n) \mid (\exists c > 0)(\exists n_0 > 0)(\forall n > n_0)\ 0 \leq cg(n) \leq f(n)\}, \\
\mathrm{o}\left(\mathrm{g(n)}\right) &= \{f(n) \mid (\forall c > 0)(\exists n_0 > 0)(\forall n > n_0)\ 0 \leq f(n) < cg(n)\}, \\
\omega\left(g(n)\right) &= \{f(n) \mid (\forall c > 0)(\exists n_0 > 0)(\forall n > n_0)\ 0 \leq cg(n) < f(n)\}, \\
\Theta\left(g(n)\right) &= \{f(n) \mid f(n) = \Omega\left(g(n)\right) \wedge f(n) = \mathrm{O}\left(\mathrm{g(n)}\right)\}.
\end{aligned}$$

Alternatívne definície pre $\mathrm{o}\left(\mathrm{g}\left(\mathrm{n}\right)\right)$ a $\omega\left(g\left(n\right)\right)$:

$$f(n) = \mathrm{o}\left(\mathrm{g(n)}\right) \text{ ak } \lim_{\mathrm{n}\to\infty} \frac{\mathrm{f(n)}}{\mathrm{g(n)}} = 0,$$

$$f(n) = \omega\left(g(n)\right) \text{ ak } \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0.$$

Ak $f(n)$ patrí do $\delta\left(g\left(n\right)\right)$, kde δ je jednou z práve definovaných asymptotík, píšeme $f(n) = \delta\left(g(n)\right)$, nie $f(n) \in \delta\left(g(n)\right)$. Ak $f(n) = \Theta\left(g\left(n\right)\right)$, hovoríme, že funkcie f a g sú asymptoticky rovnaké.

Kapitola 3

Kolmogorovská zložitosť

3.1 Definície a základné vlastnosti

Po vysvetlení základných pojmov potrebných k pochopeniu Kolmogorovskej zložitosti, posuňme sa ďalej. Hovorili sme, že KZ je popisná zložitosť reťazcov. Je rovná dĺžke najkratšieho popisu reťazca. Aby sme tieto popisy mohli vytvárať a zároveň, aby sme z popisu vedeli extrahovať samotný kódovaný objekt, budeme musieť mať nejakú dohodnutú popisnú metódu, ktorú budeme používať. Našimi popisnými metódami budú čiastočne rekurzívne funkcie.

Nech objekty, s ktorými budeme pracovať, sú prvkami množiny S. Chceme nájsť popisnú metódu, vďaka ktorej z popisu p, vieme zistiť, ktorý prvok x popisuje. Inými slovami hľadáme čiastočne rekurzívnu funkciu f, pre ktorú platí $f(p) = x$.
Definujme zložitosť objektu $x \in S$ vzhľadom k popisnej metóde f nasledovne:

$$C_f(x) = \min\{l(p) : f(p) = x\}, \tag{3.1}$$

pričom $C_f(x) = \infty$ ak také p neexistuje. V počítačovej terminológii môžeme povedať, že f je počítač a p je program. Potom $C_f(x)$ je dĺžka najkratšieho programu programu p s výstupom x pre počítač f.

Pri hľadaní najkratšieho popisu objektu x si treba uvedomiť aj fakt, že pre inú popisnú metódu môže existovať kratší popis x. Ako príklad poslúži

napr. porovnanie reprezentácie čísla v štandardnej binárnej sústave a reprezentácie uvedenej v 2.2.

Nech $f_1, f_2, \ldots, f_r$ je enumerácia metód popisujúcich prvky S. Môžeme vytvoriť novú metódu f, pre ktorú bude platiť, že zložitosť x vzhľadom na f bude prevyšovať najmenšiu z $C_{f_1}(x), C_{f_2}(x), \ldots, C_{f_r}(x)$ najviac o nejakú konštantu c. Metóda f sa bude správať ako metóda f_i, ktorej $C_{f_i}(x)$ je najmenšia spomedzi všetkých, pričom táto metóda bude špecifikovaná na prvých $\log r$ bitoch programu p. Zložitosť x vzhľadom na f bude od $C_{f_i}(x)$ väčšia len o $\log r$, keďže programu p sme pridali prefix dĺžky $\log r$.

Definícia 3.1.1. Hovoríme, že popisná metóda f *aditívne minorizuje* metódu g, ak existuje konštanta c taká, že

$$C_f(x) \leq C_g(x) + c \tag{3.2}$$

Podľa tejto definície platí, že metóda f, ktorú sme uviedli v predošlej úvahe aditívne minorizuje všetky metódy $C_{f_1}(x), C_{f_2}(x), \ldots, C_{f_r}(x)$, pričom hodnota $c \approx \log r$.

Definícia 3.1.2. Nech C je trieda čiastočne rekurzívnych funkcií nad nezápornými celými číslami. Funkcia $f \in C$ je *univerzálna* alebo *aditívne optimálna* pre C, ak pre každú funkciu $g \in C$ existuje konštanta $c_{f,g}$ (závislá len od f a g) taká, že pre všetky x platí

$$C_f(x) \leq C_g(x) + c_{f,g} \tag{3.3}$$

Hovoríme, že dve metódy sú *ekvivalentné* práve vtedy, keď sa navzájom minorizujú. Pre všetky aditívne optimálne metódy zrejme platí

$$|C_f(x) - C_g(x)| \leq c_{f,g} \tag{3.4}$$

Nech x, y a p sú prirodzené čísla (reťazce). Ľubovoľná čiastočne rekurzívna funkcia ϕ spolu s p také, že $\phi(p) = x$ sú popisom x. Ak $\phi(\langle y, p\rangle) = x$, tak popisom x sú funkcia ϕ spolu s p a y.

Definícia 3.1.3. *Kolmogorovská zložitosť* $C_\phi(x)$ reťazca x vzhľadom k metóde ϕ je definovaná ako

$$C_\phi(x) = \min\{l(p) : \phi(p) = x\} \tag{3.5}$$

Ak také p neexistuje, $C_\phi(x) = \infty$.

Definícia 3.1.4. Kolmogorovská zložitosť reťazca x pri znalosti y je definovaná ako

$$C_\phi(x|y) = \min\{l(p) : \phi(\langle y, p\rangle) = x\} \tag{3.6}$$

Ak také p neexistuje, $C_\phi(x|y) = \infty$. Tomuto variantu KZ hovoríme aj *podmienená* Kolmogorovská zložitosť. Jednoduchú KZ definovanú v predošlej definícii voláme aj *nepodmienená*.

Jednoduchú KZ môžeme definovať pomocou podmienenej ako

$$C_\phi(x) = C_\phi(x|\epsilon) \tag{3.7}$$

Ukážme si teraz, že pre univerzálnu čiastočne rekurzívnu funkciu ϕ_0 z našej enumerácie ČRF naozaj platí, že je aditívne optimálna podľa definície 3.1.2.

Veta 3.1.5. *Nech ϕ_0 je univerzálna čiastočne rekurzívna funkcia pre triedu čiastočne rekurzívnych funkcií. Potom pre všetky ČRF ϕ a všetky x a y platí*

$$C_{\phi_0}(x|y) \leq C_\phi(x|y) + c_\phi, \tag{3.8}$$

kde c_ϕ je konštanta závislá od ϕ, ale nie od x alebo y.

Dôkaz. Nech funkciu ϕ_0 počíta univerzálny TS U, ktorý na vstupe $\langle y, \langle n, p\rangle\rangle$ simuluje TS T_n so vstupom $\langle y, p\rangle$. T_n počíta ČRF ϕ_n, preto (ako sme už uviedli v 2.3.1) platí

$$\phi_0(\langle y, \langle n, p\rangle\rangle) = \phi_n(\langle y, p\rangle).$$

Vieme, že $\langle y, p\rangle = 1^{l(y)}0yp$ a $\langle y, \langle n, p\rangle\rangle = 1^{l(y)}0y1^{l(n)}0np$
Vidíme, že univerzálna ϕ_0 prečítala navyše $2l(n) + 1$ bitov popisu.

$$1^{l(y)}0y\underbrace{\mathbf{1^{l(n)}0n}}_{2l(n)+1}p$$

Preto ak zvolíme $c_{\phi_n} = 2l(n) + 1$, pre všetky n platí

$$C_{\phi_0}(x|y) \leq C_{\phi_n}(x|y) + c_{\phi_n}. \tag{3.9}$$

□

Uvedomme si, že predošlá veta nehovorí o tom, že univerzálnej ČRF by stačil ten najkratší popis, ale že žiadna iná ČRF nepotrebuje výrazne dlhší popis. Výrazne dlhší v tomto prípade znamená dlhší o viac ako o c_ϕ bitov.

Dôsledkom vety 3.1.5 je, že pre ľubovoľné dve aditívne optimálne funkcie ψ a ψ' existuje konštanta $c_{\psi,\psi'}$ závislá len od ψ a ψ' taká, že

$$|C_\psi(x|y) - C_{\psi'}(x|y)| \leq c_{\psi,\psi'}. \tag{3.10}$$

Stačí funkcie ϕ_0 a ϕ z uvedenej vety nahradiť aditívne optimálnymi funkciami ψ a ψ', potom ich nahradiť v obrátenom poradí a platnosť dôsledku je zrejmá.

Vďaka tomu, čo sme si práve ukázali, dokážeme odpovedať na otázku aké popisné metódy budeme používať. Vidíme, že použitie univerzálneho TS namiesto ľubovoľného iného TS navýši dĺžku popisu len o aditívnu konštantu c, ktorá je pre nás irelevantná a môžeme ju zanedbať. Zafixujeme preto jednu univerzálnu popisnú metódu, ktorú budeme používať. Odteraz sa budeme opierať o to, že popis objektu spracováva univerzálny TS, ktorý potrebuje naviac len konštantný počet bitov na popis daného TS (jeho poradové číslo v enumerácii TS).

Tieto úvahy ústia do nasledujúcej definície.

Definícia 3.1.6. Napevno zvoľme univerzálnu ČRF ϕ_0. Kolmogorovskú zložitosť $C(x)$ definujeme ako

$$C(x) = C_{\phi_0}(x). \tag{3.11}$$

Funkcia ϕ_0 je našou *referenčnou* funkciou. Univerzálny TS U, ktorý počíta referenčnú funkciu ϕ_0 bude pre nás *referenčný* UTS.

Podmienenú KZ $C(x|y)$ potom definujeme

$$C(x|y) = C_{\phi_0}(x|y).$$

Pritom zrejme stále platí $C(x) = C(x|\epsilon)$.

Už vieme, že binárnymi reťazcami budeme popisovať rôzne objekty rôznych typov. Jednotlivé objekty budú často súčasťou väčších celkov či skupín objektov. Informácia, že daný objekt patrí do nejakej množiny A, nám môže pomôcť výrazne skrátiť jeho popis, ak budeme vedieť generovať prvky tejto množiny. Toto tvrdenie dokazuje nasledujúca veta.

Veta 3.1.7. *Nech $A \subseteq \mathbb{N} \times \mathbb{N}$ je rekurzívne vyčísliteľná*[1] *a $y \in \mathbb{N}$. Ďalej nech množina $Y = \{x : (x, y) \in A\}$ je konečná. Potom pre všetky $x \in Y$ platí*

$$C(x|y) \leq l\left(|Y|\right) + c \tag{3.12}$$

kde c je konštanta závislá len od A.

Dôkaz. Nech množinu A vieme enumerovať pomocou TS T, pričom nech $(x_1, y_1), (x_2, y_2), \ldots$ je enumerácia (bez opakovania) získaná vďaka T. Označme $k = |Y|$ a nech $(x_{i_1}, y_{i_1}), \ldots, (x_{i_k}, y_{i_k})$ je podpostupnosť, v ktorej máme vymenované práve tie dvojice z A, ktorých prvý prvok (teda x_{i_j}) patrí do Y. Množina Y obsahuje tie dvojice z A, ktoré majú ako druhý prvok y. Zvolené y použijeme na úpravu stroja T na stroj T_y. TS T_y bude pracovať tak, že na vstupe p, pričom $1 \leq p \leq |Y|$, vypíše x_{i_p} z dvojice (x_{i_p}, y_{i_p}), kde $y = y_{i_p}$. Platí $T_y(x) = x_{i_p}$.
Z vety 3.1.5 vyplýva

$$C(x|y) \leq C_{T_y}(x) + c.$$

A keďže $1 \leq p \leq |Y|$, platí

$$C(x|y) \leq C_{T_y}(x) + c \leq l\left(|Y|\right) + c,$$

kde konštanta c je závislá len od A. □

[1] Množina A je rekurzívne vyčísliteľná, práve vtedy, keď vieme enumerovať jej prvky, resp. ak pre ľubovoľný prvok $a \in A$ vieme rozhodnúť, že do nej naozaj patrí. Ak $a \notin A$, nemusíme vedieť rozhodnúť, či platí $a \in A$ alebo $a \notin A$.

3.2 Príklady

Dostávame sa k prvej podkapitole venovanej praktickým príkladom. Jej cieľom je pomôcť čitateľovi precvičiť si vedomosti nadobudnuté v doterajších častiach práce, lepšie pochopiť jednotlivé pojmy, pochopiť a nájsť súvislosti. Príklady sú vyberané tak, aby sa nám tieto ciele podarilo naplniť. Každý z príkladov má jasne formulované zadanie, za ktorým nasleduje jeho riešenie. Úlohy bez riešenia sú určené na precvičenie čitateľovi. Považujeme za vhodné, aby tieto úlohy riešil sám čitateľ, nakoľko ich pokladáme za exemplárne. Často majú relatívne nenáročné riešenia, ktoré môžu byť podobné riešeniu predchádzajúceho príkladu alebo vyžadujú aplikáciu metódy uvedenej v predchádzajúcom texte. Na dôkladné pochopenie celej látky odporúčame preriešiť čo najviac úloh. Taktiež odporúčame všetky úlohy riešiť samostatne a riešenie si prečítať až po ich vyriešení, prípadne keď sa vám úlohu nebude dariť vyriešiť. Príklady sme čerpali zväčša z [16].

Príklad 3.2.1. Ukážte, že $C(0^n|n) \leq c$, kde c je konštanta nezávislá od n.

Riešenie. Máme ukázať, že existuje konštanta c, ktorou vieme zhora ohraničiť KZ reťazca 0^n, ak poznáme hodnotu n. Znamená to, že má existovať program p pre nejaký TS T_i dĺžky $l(p) = C(0^n|n) \leq c$, ktorý dokáže vypísať reťazec 0^n pre n, ktoré už pozná. Takýto program vieme ľahko napísať, slovne ho môžeme vyjadriť ako „vypíš n núl.“. Keďže program p pozná hodnostu n, ktorá je preň argumentom, dĺžka programu bude konštantná a nezávislá od n. Podľa definície platí

$$C_{T_i}(0^n|n) = \min\{l(p) : T_i(n, p) = 0^n\} \tag{3.13}$$

Nech dĺžka nášho programu p je $l(p) = c_p$ a nech c_{T_i} je konštanta, pre ktorú platí $C(0^n|n) \leq C_{T_i}(0^n|n) + c_{T_i}$. Konštanta c_{T_i} v sebe zahŕňa aj počet bitov potrebných na samooddeľujúci zápis i na špecifikáciu stroja T_i ($l(\bar{i}) = 2\log i + 1$). Pripomíname, že náš program p nemusí byť najkratším popisom reťazca 0^n, určite však platí $C_{T_i}(0^n|n) \leq l(p)$. Potom

$$C(0^n|n) \leq C_{T_i}(0^n|n) + c_{T_i} \leq c_p + c_{T_i} \tag{3.14}$$

Vidíme, že vzťah $C(0^n|n) \leq c$, ktorý sme chceli dokázať, platí pre $c = c_p + c_{T_i}$. □

Úloha 3.2.2. Ukážte, že $C(\pi_{1:n}|n) \leq c$, kde $\pi = 3.145926535\ldots$ a c je konštanta nezávislá od n.

Úloha 3.2.3. Definujme $C(\omega) = \min\{l(p) : U(p,n) = \omega_{1:n}(\forall n)\}$. Ak také p neexistuje, $C(\omega) = \infty$. Zrejme $C(\omega) < \infty$ alebo $C(\omega) = \infty$. Ukážte, že $C(\omega) < \infty \iff 0.\omega$ je rekurzívne reálne číslo[2]. Ukážte tiež, že pre matematické konštanty π a e platí $C(\pi) < \infty$ a $C(e) < \infty$.

Príklad 3.2.4. Nech $q \in \mathbb{N}$ a x je taký konečný binárny reťazec, že $C(x) = q$. Čomu sa potom rovná $C(x^q)$? (x^q je q-násobné zreťazenie reťazca x, čiže $x^q = \underbrace{xxx\ldots x}_{q\text{-krát}}$)

Riešenie. Vieme, že $C(x) = q$, takže existuje program p dĺžky q a taký TS T_i také, že $T_i(p) = x$. Aby sme dostali x^q, stačí upraviť T_i tak, aby na jeho výstupe nebolo len x, ale x q-krát zreťazené. Na to, aby sme špecifikovali, koľkokrát sa x má reťaziť, stačí $\log q$ bitov plus konštantný počet bitov c_i na úpravu T_i.
Vidíme, že

$$C(x^q) \leq C(x) + \log q + c_i \tag{3.15}$$

Nevedeli by sme ešte efektívnejšie využiť fakt, že $C(x) = l(p) = q$? Ak máme program p na výpočet x a vieme, že jeho dĺžka sa rovná q, môžeme to využiť. Hodnotu q máme vlastne zapísanú v samotnom programe p jeho dĺžkou. T_i upravíme tak, že najskôr prečíta program p a zistí jeho dĺžku. Tak zistíme hodnotu q. Ďalej počíta program p. Po ukončení výpočtu bude na páske x. Na záver $q-1$-krát skopíruje x na pásku za prvé x, ktoré tam vypísal program p. Výstupom preto bude x^q. Dostávame

$$C(x^q) \leq C(x) + c. \quad \square$$

[2]Reálne číslo $0.\omega$ je rekurzívne, ak existuje taká totálna rekurzívna funkcia ϕ, že $\phi(i) = \omega_i$ pre všetky i. Postupnosť ω voláme rekurzívna postupnosť.

Príklad 3.2.5. Ukážte, že existujú nekonečné binárne reťazce ω také, že dĺžka najkratšieho programu pre UTS, ktorý by generoval ω znak po znaku je výrazne menšia ako dĺžka takého programu pre UTS, ktorý by generoval prvých n znakov ($\omega_{1:n}$) reťazca ω pre dostatočne veľké n.

Riešenie. Existujú také reťazce? Ako vyzerajú? Po krátkej úvahe zistíme, že úloha je pomerne jednoduchá. Hneď na úvod je dôležité uvedomiť si, že hovoríme o $C(\omega_{1:n})$, nie o $C(\omega_{1:n}|n)$. Hodnota n nám teda nie je implicitne známa. Hlavným problémom je, že ak chceme generovať prvých n bitov reťazca, potrebujeme poznať n.

Vezmime si ľubovoľný rekurzívny reťazec ω, tzn. taký, že ho vieme generovať pomocou nejakého algoritmu s konštantnou dĺžkou O(1). Nech je to algoritmus A, implementovaný programom p pre stroj T_A. Platí $C_{T_A}(\omega) = \mathrm{O}(1)$, resp. $C(\omega) \leq C_{T_A}(\omega) + c_{T_A}$, kde konštanta c_{T_A} je závislá jedine od T_A a je rovná počtu bitov, ktoré UTS U potrebuje na špecifikáciu TS T_A.Môžeme písať

$$C(\omega) = \mathrm{O}(1). \tag{3.16}$$

Naproti tomu ak chceme vypísať prvých n znakov tohto reťazca, akýkoľvek algoritmus bude potrebovať aspoň $\log n$ bitov na špecifikáciu hodnoty n, aby vedel, koľko znakov má vypísať. Hodnota $\log n$ je zrejme väčšia ako ľubovoľná konštanta pre všetky (až na konečne veľa[3]) n. □

Príklad 3.2.6. Ukážte, že pre každé $x \in \{0,1\}^*$ existuje aditívne optimálna funkcia ϕ_0 taká, že $C_{\phi_0}(x) = 0$. Ukážte, že obdobné tvrdenie platí aj pre podmienenú zložitosť.

Riešenie. Pre každé x vieme nájsť funkciu s požadovanými vlastnosťami. Nech ϕ_0 je naša referenčná aditívne optimálna funkcia.
Platí $(\forall n)(\forall p)\phi_0(\langle n,p\rangle) = \phi_n(p)$ a zároveň $(\forall x)(\forall i)(\exists c_i)C_{\phi_0}(x) \leq C_{\phi_i}(x)+c_i$. Ďalej podľa konvencie platí $\phi_0(\langle \epsilon, p\rangle) = \phi_0(0p) = x$. Upravme teraz túto funkciu na funkciu ϕ_0' tak, aby platilo $\phi_0'(\epsilon) = x$, tzn. že bez akéhokoľvek vstupu

[3]Neplatí to pre niekoľko malých n, pre ktoré $\log n < c_{T_A}$. Tých je zrejme len konečne veľa.

vytlačí práve reťazec x. Funkcia ϕ'_0 je potom presne funkcia s požadovanými vlastnosťami.
Funkciu pre podmienenú zložitosť nájdeme analogicky. □

Definícia 3.2.1. Definujme zápis $C(x, y)$ ako $C(\langle x, y\rangle)$. Tak isto zápisom $C(x|y, z)$ rozumieme $C(x|\langle y, z\rangle)$.

Úloha 3.2.7. Nech x, y a z sú prirodzené čísla. Dokážte nasledujúce tvrdenia:

(a) $C(x|y) \leq C(x) + \mathrm{O}(1)$

(b) $C(x|y) \leq C(x, z|y) + \mathrm{O}(1)$

(c) $C(x|y, z) \leq C(x|y) + \mathrm{O}(1)$

Poznámka 3.2.2. Na tomto mieste je dobré si uvedomiť, že v prípade reťazcov tvaru $\langle x, y\rangle$ nie je nutné zaoberať sa ich detailným spracúvaním Turingovymi strojmi. Na kódovanie reťazcov x, y na $\langle x, y\rangle$, resp. dekódovanie opačným smerom, máme totiž jednoduchý algoritmus, na zápis ktorého stačí $\mathrm{O}(1)$ bitov. Znamená to, že sa nemusíme dopodrobna zaoberať tým ako upraviť nejaký TS, keďže na tieto kódovacie/dekódovacie úpravy máme iný TS, použitie ktorého si vyžiada len konštantne veľa bitov. Pri takýchto príkladoch preto nebudeme popisovať všetky detaily takýchto úprav.

Príklad 3.2.8. Nech x, y a z sú prirodzené čísla. Dokážte nasledujúce tvrdenia:

(a) $C(x, x) = C(x) + \mathrm{O}(1)$

Riešenie. Máme dokázať rovnosť $C(x, x) = C(x) + \mathrm{O}(1)$. Na rozdiel od predchádzajúcich prípadov, v tomto vystupuje rovnosť namiesto (neostrej) nerovnosti. Riešenie bude preto rozdelené na dôkazy oboch nerovností jednotlivo.
Začneme tou ľahšou nerovnosťou:

„$C(x) \leq C(x, x) + \mathrm{O}(1)$“ Máme ukázať, že najkratší popis x nie je dlhší než popis $\langle x, x\rangle$. Na popis $\langle x, x\rangle$ existuje program p dĺžky

$C(x, x)$. Tento program vypíše na výstup reťazec $\langle x, x\rangle$, no my na výstupe potrebujeme x. Program p môžeme upraviť tak, aby na výstupe neodovzdal $\langle x, x\rangle$, ale len x. Stačí zabezpečiť, aby potom ako vypočíta $\langle x, x\rangle$, $\langle x, x\rangle$ upravil na x, na čo stačí konštantný počet bitov. Prvá nerovnosť je dokázaná.

„$C(x, x) \leq C(x) + \mathrm{O}(1)$“ V tomto prípade potrebujeme TS upraviť tak, aby namiesto výpisu x, vypísal x, x, resp. $\langle x, x\rangle$. Stačí, aby po výpočte x toto x skopíroval a vypísal ho dvakrát.[4] Takáto úprava si vyžaduje len $\mathrm{O}(1)$ bitov, čím sme ukázali platnosť druhej nerovnosti, a teda aj celej rovnosti. □

(b) $C(x, y|z) = C(y, x|z) + \mathrm{O}(1)$

Riešenie. Máme ukázať, že reťazce $\langle x, y\rangle$ a $\langle y, x\rangle$ majú rovnakú KZ (až na konštantu), ak poznáme z. Obe nerovnosti tejto rovnosti sa ukazujú rovnako, ukážeme si preto len jednu z nich. Nech je to $C(x, y|z) \leq C(y, x|z) + \mathrm{O}(1)$. Majme program p pre TS T_i, ktorý keď pozná z, vypíše na výstupe $\langle y, x\rangle$. Chceme ho upraviť tak, aby mal na výstupe $\langle x, y\rangle$. Oba reťazce, $\langle x, y\rangle$ aj $\langle y, x\rangle$, v sebe obsahujú tú istú informáciu, tj. samooddeľujúci zápis reťazcov x a y, len v inom poradí. Program p upravíme tak, aby po výpočte reťazca $\langle y, x\rangle$, z tohto reťazca dekódoval reťazce x a y pomocou TS uvedeného v poznámke 3.2.2 a tie následne zakódoval do reťazca $\langle x, y\rangle$. Túto úpravu sme zapísali na konštantný počet znakov (bitov), preto zložitosť navýšime len o konštantu. Dokázali sme $C(x, y|z) \leq C(y, x|z) + \mathrm{O}(1)$.
Dôkaz druhej nerovnosti je analogický. □

(c) $C(x|y, z) = C(x|z, y) + \mathrm{O}(1)$

Riešenie. V tomto prípade máme ukázať, že dĺžka najkratšieho popisu x nezávisí od poradia vstupov. Nezáleží na tom, či program p,

[4] Ak chceme vypísať $\langle x, x\rangle$, teda samooddeľujúci zápis, použijeme TS spomínaný v poznámke 3.2.2.

ktorý počíta x, ak pozná y a z má tieto reťazce zakódované v tvare $\langle z, y\rangle$ alebo $\langle y, z\rangle$. V oboch prípadoch musí ich samooddeľujúci zápis dekódovať a potom získa obe hodnoty – z aj y. V prípade, že má $\langle y, z\rangle$ (v druhom prípade analogicky), môže tento vstup dekódovať na y a z a zakódovať ich do $\langle z, y\rangle$. Je to úprava vyžadujúca O(1) bitov. Platia preto obe nerovnosti tejto rovnosti a teda aj celá rovnosť. □

Úloha 3.2.9. Nech x, y a z sú prirodzené čísla. Dokážte nasledujúce tvrdenia:

(a) $C(x, y|x, z) = C(y|x, z) + \mathrm{O}(1)$

(b) $C(x|x, z) = C(x|x) + \mathrm{O}(1) = \mathrm{O}(1)$

Príklad 3.2.10. Nech x, y a z sú prirodzené čísla a ϕ_k je v poradí k-ta v enumerácii čiastočne rekurzívnych funckií. Ukážte:

(a) $C(x, y) \leq C(x) + 2l\left(C\left(x\right)\right) + C(y|x) + \mathrm{O}(1)$

Riešenie. Chceme popísať oba reťazce x aj y. Vezmime si program $p, l(p) = C(x)$ pre nejaký stroj T_i ako popis x. Nech q je popis y pre stroj T_j, ktorý už pozná x, $l(q) = C(y|x)$. Potom reťazec $\overline{l(p)}pq$ je popisom x, y dĺžky $2l\left(C\left(x\right)\right) + C(x) + C(y|x)$ plus O(1) bitov na diskusiu, resp. popisy jednotlivých TS. □

(b) $C\left(\phi_k\left(x\right)|y\right) \leq C(x|y) + 2l(k) + \mathrm{O}(1)$

Riešenie. $\phi_k(x)$ môže na prvý pohľad pôsobiť odstrašujúco, ale je to tak len do vtedy, kým si neuvedomíme, že je to proste nejaká čiastočne rekurzívna funkcia z našej enumerácie všetkých čiastočne rekurzívnych funkcií, a keďže ich vieme enumerovať, na jej špecifikáciu nám stačí poznať k. Nech p je program dĺžky $C(x|y)$ pre TS T_i, ktorý pozná y a na výstupe odovzdá x. Nech T_k je TS, ktorý počíta funkciu ϕ_k. Potom $\phi_k(x)$ vieme popísať reťazcom $\overline{k}p$, pričom dĺžka tohto reťazca je $C(x|y) + 2l(k)$. □

(c) $C\left(\phi_k\left(x,y\right)\right) \leq C(x) + 2l\left(C\left(x\right)\right) + C(y|x) + 2l(k) + \mathrm{O}(1) \leq$
$C(x) + 2l\left(C\left(x\right)\right) + C(y) + 2l(k) + \mathrm{O}(1)$

Riešenie. Pozrime sa na reťazec $\overline{l(p)}p\overline{k}q$, kde p dĺžky $C(x)$ je program pre T_i, ktorý počíta x, a q také, že $l(q) = C(y|x)$ je program pre T_j, ktorý počíta y ak už pozná x. Vidíme, že z tohto popisu vieme zistiť x, y. Najskôr z neho vyčítame program p, pomocou ktorého zistíme x, prečítame k, ktoré nám stačí na špecifikáciu ϕ_k, prečítame q, pomocou ktorého zistíme y a potom už len vypočítame hodnotu $\phi_k(x,y)$.
Tým sme ukázali prvú nerovnosť, platnosť druhej triviálne plynie z nerovnosti $C(y|x) \leq C(y) + \mathrm{O}(1)$. □

Úloha 3.2.11. Nech x, y a z sú prirodzené čísla a ϕ_k je v poradí k-ta v enumerácii čiastočne rekurzívnych funckií. Ukážte, že platí

$$C\left(y|\phi_k\left(x\right)\right) \geq C(y|x) - 2l(k) + \mathrm{O}(1).$$

Príklad 3.2.12. Nech x, y a z sú prirodzené čísla a ϕ_k je v poradí k-ta v enumerácii čiastočne rekurzívnych funckií, ktorá je navyše aj injektívna (one-to-one). Ukážte

$$|C(x) - C\left(\phi_k\left(x\right)\right)| \leq 2l(k) + \mathrm{O}(1).$$

Riešenie. Riešenie môžeme rozdeliť na dve časti. Buď platí $C\left(\phi_k\left(x\right)\right) \geq C(x)$ alebo $C\left(\phi_k\left(x\right)\right) < C(x)$. V prvom prípade môžeme nerovnosť upraviť na $C\left(\phi_k\left(x\right)\right) \leq C(x) + 2l(k) + \mathrm{O}(1)$ a tú sme už dokázali v 3.2.10 (b). V druhom prípade, $C(x) \leq C\left(\phi_k\left(x\right)\right) + 2l(k) + \mathrm{O}(1)$, využijeme injektívnosť ϕ_k. Nech p je popis hodnoty $\phi_k(x)$, $l(p) = C\left(\phi_k\left(x\right)\right)$ a $\overline{k}$ je samoodeľujúci zápis k. Keďže ϕ_k je injektívna, platí

$$(\forall m)(\forall n)\left(\left(\phi\left(m\right) = \phi\left(n\right)\right) \Leftrightarrow \left(m = n\right)\right).$$

Ak nájdeme také y, že $\phi_k(y) = \phi_k(x)$, platí $y = x$. Na nájdenie takého y použijeme metódu, ktorú budeme využívať aj v iných úlohách.

	$\phi_k(1)$	$\phi_k(2)$	$\phi_k(3)$	$\phi_k(4)$	$\cdots$	
1	1	3	6	10		
2	2	5	9			
3	4	8				
4	7					
$\vdots$						

Obr. 3.1: Tabuľkové znázornenie použitej metódy. Riadky reprezentujú jednotlivé kroky výpočtu funkcie na danom vstupe, v rámci stĺpca označeného $\phi_k(i)$ sa počíta hodnota ϕ_k na vstupe i. Kroky sú vykonávané postupne v poradí ako je uvedené v jednotlivých políčkach.

Metóda spočíva v tom, že funkciu spustíme na všetkých vstupoch prichádzajúcich do úvahy a hľadáme vstup, na ktorom funkcia vráti požadovanú hodnotu. V tomto prípade budeme spúšťať výpočet ϕ_k postupne na všetkých číslach $y \geq 1$. Jednotlivé výpočty rozdelíme na kroky. Najskôr sa vykoná prvý krok výpočtu $\phi_k(1)$. Potom druhý krok výpočtu $\phi_k(1)$ a prvý krok $\phi_k(2)$. Ďalej tretí krok $\phi_k(1)$, druhý krok $\phi_k(2)$ a prvý krok $\phi_k(3)$. Označme $\phi_k^i(j)$ i-ty krok výpočtu $\phi_k(j)$. Poradie vykonávania krokov je potom nasledovné.

$$\phi_k^1(1) \to \phi_k^2(1) \to \phi_k^1(2) \to \phi_k^3(1) \to \phi_k^2(2) \to \phi_k^1(3) \to \phi_k^4(1) \to \ldots$$

Toto poradie znázorňuje aj obrázok 3.1.

Uvedomme si, že si nemôžeme dovoliť spúšťať celé výpočty ϕ_k, nevieme totiž, kedy daný výpočet skončí. Nevieme dokonca ani to, či výpočet vôbec skončí. Ak by sme spustili výpočet $\phi_k(j)$ pre nejaké j, mohlo by sa stať, že výpočet by trval nekonečne dlho, resp. nikdy by neskončil. Tým pádom by

sme sa nikdy nedostali k výpočtu ϕ_k na ďalších vstupoch. Použitou metódou zabezpečíme, že postupne počítame hodnotu $\phi_k(j)$ pre všetky $j \geq 1$.

Celý výpočet zastavíme v momente, keď niektorý z výpočtov $\phi_k(y)$ zastaví a vráti hodnotu rovnú $\phi_k(x)$. Znamená to, že y, na ktorom sme výpočet spustili, sa vďaka injektívnosti ϕ_k rovná x. Získali sme ho z popisu dĺžky $C\left(\phi_k\left(x\right)\right) + 2l(k) + \mathrm{O}(1)$. Preto určite platí $C(x) \leq C\left(\phi_k\left(x\right)\right) + 2l(k) + \mathrm{O}(1)$. □

Úloha 3.2.13. Nech x, y a z sú prirodzené čísla a ϕ_k je v poradí k-ta v enumerácii čiastočne rekurzívnych funckií, ktorá je navyše aj injektívna (one-to-one). Ukážte, že platí

$$C(x|y,z) \leq C\left(x|\phi_k\left(y\right),z\right) + 2l(k) + \mathrm{O}(1)$$

Príklad 3.2.14. Nech ϕ je napevno zvolená bijektívna funkcia $\phi : \{0,1\}^* \to \{0,1\}^*$. Ukážte, že pre všetky $x \in \{0,1\}^*$ platí

$$C(x) - C\left(x|\phi\left(x\right)\right) = C(x) + \mathrm{O}(1) = \mathrm{C}\left(\phi\left(\mathrm{x}\right)\right) + \mathrm{O}(1).$$

Riešenie. Začnime dôkazom druhej rovnosti $C(x)+\mathrm{O}(1) = \mathrm{C}\left(\phi\left(\mathrm{x}\right)\right)+\mathrm{O}(1)$. Rozdelíme ho na dôkaz dvoch nerovností:

„$C(x) + \mathrm{O}(1) \geq \mathrm{C}\left(\phi\left(\mathrm{x}\right)\right) + \mathrm{O}(1)$“ Nech $p, l(p) = C(x)$, je program na výpočet x. p' sa bude chovať rovnako ako p, ale po výpočte x ešte vypočíta hodnotu $\phi(x)$, ktorá bude jeho výstupom. Keďže ϕ je nám známa napevno zvolená funkcia, nepotrebujeme na jej špecifikáciu jej poradové číslo v enumerácii ako to bolo v doterajších prípadoch.

„$C(x) + \mathrm{O}(1) \leq \mathrm{C}\left(\phi\left(\mathrm{x}\right)\right) + \mathrm{O}(1)$“ Druhú nerovnosť ste mali dokázať v úlohe 3.2.13 na strane 40, s tým rozdielom, že tu je funkcia ϕ napevno zvolená, preto nepotrebujeme $2l(k)$ bitov na samoodeľujúci zápis jej špecifikácie.

Prvú rovnosť môžeme upraviť na

$$C\left(x|\phi\left(x\right)\right) = \mathrm{O}(1).$$

Ak poznáme hodnotu $\phi(x)$, na overenie x nám stačí informácia konštantnej veľkosti. Funkcia ϕ je napevno zvolená a poznáme aj hodnotu $\phi(x)$. Podobne ako v úlohe 3.2.12 budeme po krokoch počítať $\phi(y)$ pre všetky $y \geq 1$, kým nenájdeme také y, že $\phi(y) = \phi(x)$. Vďaka bijektívnosti ϕ takto zistíme hodnotu x. Nepotrebovali sme žiadne informácie nekonštantnej dĺžky, preto $C\left(x|\phi\left(x\right)\right) = \mathrm{O}(1)$. □

Príklad 3.2.15. Ukážte, že platí $C\left(x + C\left(x\right)\right) \leq C(x) + \mathrm{O}(1)$

Riešenie. Nech p je program dĺžky $C(x)$ pre Turingov stroj T_i, ktorý počíta x. T_i môžeme upraviť na T_i' tak, aby okrem programu p, ktorý vykonal, aj spočítal dĺžku p a pripočítal ju k výstupu programu p, teda k x. Takže T_i' spustí program p, zároveň aj vie zistiť jeho dĺžku, vypočíta x a pripočíta k nemu $C(x)$ ako dĺžku p. Táto úprava si vyžiada konštantý počet bitov, takže tvrdenie sme dokázali. □

Príklad 3.2.16. Ukážte, že ak $m \leq n$, tak $m + C(m) \leq n + C(n) + \mathrm{O}(1)$

Riešenie. Ak $m = n$, tak tvrdenie triviálne platí. Predpokladajme, že $m < n$. Hodnoty $C(m)$ a $C(n)$ nevieme len tak porovnať na základe predpokladu. Vieme však, že určite platí $C(m) \leq C(n) + 2l(n-m) + \mathrm{O}(1)$. Totiž ak máme program na výpočet n a poznáme aj hodnotu rozdielu $(n-m)$, vieme poľahky vypočítať aj m. Hodnota $l(n-m)$ sa rovná $\log(n-m)$. Ďalej zrejme platí $2l(n-m) \leq n-m$. Toto zistenie môžeme využiť a dosadiť

$$C(m) \leq C(n) + 2l(n-m) + \mathrm{O}(1) \leq C(n) + (n-m) + \mathrm{O}(1)$$
$$C(m) + m \leq C(n) + n + \mathrm{O}(1) \quad \square$$

Príklad 3.2.17. Nech ϕ_k je k-ta v enumerácii čiastočne rekurzívnych funkcií a a je napevno zvolené prirodzené číslo také, že množina $A = \{x : \phi_k(y) = \langle a, x\rangle, \text{ pre nejaké } y \in \mathbb{N}\}$ je konečná. Ukážte, že potom $(\forall x \in A)$ platí

$$C(x|a) \leq l\left(|A|\right) + 2l(k) + \mathrm{O}(1)$$

Riešenie. Ak poznáme a a k, vieme po krokoch simulovať výpočty $\phi_k(y)$ na všetkých $y \in \mathbb{N}$ ako v úlohe 3.2.12. Vždy, keď dostaneme nejakú dvojicu

$\langle a, x\rangle$, vieme, že x patrí do A. Takto vieme enumerovať množinu A. Preto z reťazca $\overline{k}i$, kde k je poradové číslo ϕ_k v enumerácii všetkých čiastočne rekurzívnych funkcií a i je poradové číslo x v popísanej enumerácii prvkov množiny A, vieme zrekonštruovať x. Keďže A je konečná, vieme, že na zápis i nám stačí $l(|A|)$ bitov.

Našli sme popis x ak poznáme a a jeho dĺžka je $l(|A|) + 2l(k) + \mathrm{O}(1)$, preto aj $C(x|a) \leq l(|A|) + 2l(k) + \mathrm{O}(1)$. □

Kapitola 4

Nestlačiteľnosť

4.1 Definície a základné vlastnosti

Ako sme si už ukázali v úvodnej kapitole a aj v príklade 3.2.2, popisná zložitosť reťazca π je malá. Dokázali sme, že jeho Kolmogorovská zložitosť je rovná konštante. Všetku informáciu, ktorú v sebe reťazec π nesie, sme skomprimovali do krátkeho popisu. Je to príklad reťazca, ktorého dĺžku popisu vieme veľmi dobre stlačiť. Z asymptotického hľadiska rastie pomer dĺžky $\pi_{1:n}$ (s rastúcim n) a dĺžky popisu, ktorá je konštantná, rovnako rýchlo ako rastie samotné n. Je to práve vďaka tomu, že popisná zložitosť tohto reťazca je konštantná. Existujú aj reťazce, pre ktoré neexistuje krátky[1] popis?

Reťazec má väčšiu KZ, ak na jeho popis potrebujeme veľa informácií. Znamená to, že má dlhý najkratší popis, neexistuje krátky algoritmus, na základe ktorého tento reťazec vieme zostrojiť. Môžeme tiež povedať, že reťazec v sebe obsahuje veľa mnoho informácií. V tom prípade zrejme nie je ľahké zachytiť regularitu, pravidelnosti a vzory v danom reťazci. Ak sú totiž v reťazci nejaké vzory či pravidelnosti, vieme ich zachytiť algoritmom. Keďže takýto dostatočne krátky algoritmus alebo popis neexistuje, hodnoty jednotlivých bitov v reťazci sa zrejme striedajú vskutku náhodne. V takom prípade sa môže stať, že bude pre nás výhodnejšie zapísať celý reťazec ako zapísať jeho popis,

[1] Čo znamená krátky popis?

pretože tento popis môže byť dokonca dlhší ako samotný reťazec. Vtedy je ale samozrejme lepšie písať samotný reťazec namiesto iného popisu, keďže tento reťazec si je zároveň najkratším popisom. Takéto reťazce budeme nazývať *nestlačiteľné* alebo niekedy aj *Kolmogorovsky náhodné*. Zadefinujme si presne kedy bude reťazec nestlačiteľný, resp. kedy hovoríme, že preň neexistuje dostatočne „krátky“ popis.

Definícia 4.1.1. Binárny reťazec x voláme *nestlačiteľný*, ak

$$\begin{aligned} C(x) &\geq l(x), \\ \text{resp. } C(x|y) &\geq l(x|y) \end{aligned}$$

Reťazec voláme nestlačiteľný, ak jeho KZ rovná aspoň jeho dĺžke. Znamená to, že najkratším popisom si je on sám.

Môže sa však stať aj to, že reťazec má vysokú KZ, no podľa predošlej definície nie je nestlačiteľný. Na pomenovanie rôznych „stupňov“ nestlačiteľnosti zavádzame všeobecnejšiu definíciu nestlačiteľnosti.

Definícia 4.1.2. Nech c je nezáporná celočíselná konštanta. Binárny reťazec x, pre ktorý platí

$$C(x) \geq l(x) - c, \tag{4.1}$$

voláme *c-nestlačiteľný*.
Ďalej nech g je celočíselná funkcia. Binárny reťazec x, pre ktorý platí

$$C(x) \geq l(x) - g\left(l\left(x\right)\right), \tag{4.2}$$

voláme *g-nestlačiteľný*.
Pre podmienenú KZ sú definície analogické.

Poznámka 4.1.3. Môžeme používať aj pojmy *c-náhodný* reťazec a *g-náhodný* reťazec.

Poznámka 4.1.4. Pôvodná defninícia nestlačiteľnosti definuje 0-nestlačiteľné reťazce.

Otázkou ostáva, či naozaj existuje reťazec, ktorý nevieme žiadnym spôsobom popísať kratšie ako je jeho dĺžka. Túto otázku vieme ľahko zodpovedať. Vezmime si všetky n-bitové reťazce. Vieme, že ich je 2^n. Koľko z nich vieme popísať programom kratším ako n? Určite ich bude najviac toľko, koľko existuje programov kratších ako n. Všetkých takých programov[2] je $\sum_{i=0}^{n-1} 2^i = 2^n - 1$. Vidíme, že ich je o jeden menej ako je všetkých n-bitových reťazcov. Znamená to, že jeden z 2^n všetkých n-bitových reťazcov, nech je to x, nevieme popísať programom kratším ako n bitov. Platí preto $C(x) \geq n$. Pre ľubovoľnú dĺžku n teda existuje aspoň jeden nestlačiteľný reťazec dĺžky n.

Teraz môžeme dokázať všeobecnejšie tvrdenie o počte nestlačiteľných reťazcov v konečných množinách. Tým tvrdením je *Veta o nestlačiteľnosti*, na ktorú sa budeme často odvolávať.

Veta 4.1.5. *Nech c je nezáporné celé číslo a A nech je konečná množina, $|A| = m$. Pre každé y platí, že množina A obsahuje aspoň*

$$m(1 - 2^{-c}) + 1$$

prvkov x takých, že $C(x|y) \geq \log m - c$.

Dôkaz. Všetkých reťazcov (možných popisov reťazca x) kratších ako $\log m - c$ je

$$\sum_{i=0}^{\log m - c - 1} 2^i = 2^{\log m - c} - 1 = m2^{-c} - 1.$$

Počet reťazcov v A, ktoré nemajú popis kratší ako $\log m - c$ je potom rovný $m - m2^{-c} + 1 = m(1 - 2^{-c}) + 1$. □

Tvrdenie, že spomedzi všetkých 2^n n-bitových binárnych reťazcov je minimálne jeden nestlačiteľný, je len špeciálnym prípadom tvrdenia dokázaného v predchádzajúcej vete. Naozaj, $2^n(1 - 2^0) + 1 = 1$.

Koľko je potom c-nestlačiteľných reťazcov v množine všetkých 2^n binárnych reťazcov? Pre každé c ich je $2^n - 2^{n-c} + 1$. 0-nestlačiteľný je preto

[2]Každý program je zároveň reťazec.

minimálne jeden, 1-nestlačiteľných je aspoň polovica, 2-nestlačiteľných aspoň tri štvrtiny, 3-nestlačiteľných je sedem osmín, ...

Už vo vete 3.1.7 na strane 31 sme si ukázali, že nám môže pomôcť, ak vieme, že reťazec je prvkom nejakej množiny. KZ reťazca x, o ktorom vieme, že je prvkom množiny A, môžeme zapísať ako $C(x|A)$. Podľa vety 3.1.7 potom platí

$$C(x|A) \leq l(|A|) + c.$$

Uvedenú nerovnosť môžeme upraviť na $C(x|A) - l(|A|) \leq c$. Rozdiel medzi samotnou KZ reťazca a dĺžky zápisu mohutnosti množiny A vieme zhora ohraničiť konštantou. Znamená to, že KZ reťazca nebude výrazne prevyšovať logaritmus mohutnosti množiny, v ktorej sa reťazec nachádza. Uvedené nás vedie k ďalšej definícii.

Definícia 4.1.6. *Odchýlka náhodnosti*[3] reťazca x vzhľadom na A je definovaná ako

$$\delta(x|A) = l(|A|) - C(x|A). \tag{4.3}$$

Z definície plynie platnosť $\delta(x|A) \geq -c$, pre nejakú konštantu c nezávislú od x.

Ak má x vysokú odchýlku náhodnosti, znamená to, že informácia, že x patrí do A nám pomohla nezanedbateľne skrátiť popis x. Koľko je reťazcov s vysokou odchýlkou náhodnosti?

Lema 4.1.6.1.

$$|\{x : \delta(x|A) \geq k\}| \leq |A|/2^{k-1} \tag{4.4}$$

Dôkaz. $\delta(x|A) \geq k \Rightarrow$ rozdiel medzi $l(|A|)$ a $C(x|A)$ je prinajmenšom k, preto $C(x|A) \leq l(|A|) - k$. Počet reťazcov x, pre ktoré platí $\delta(x|A) \geq k$, je rovný

$$2^{l(|A|)-k+1} = \frac{2^{l(|A|)}}{2^{k-1}} = \frac{|A|}{2^{k-1}}$$

□

[3] randomness deficiency

4.2 Metóda nestlačiteľnosti

Fakt, že nestlačiteľné reťazce existujú, a dokonca že skoro všetky sú nestlačiteľné, sa veľmi často využíva pri rôznych dôkazoch. Metóde, ktorá nestlačiteľnosť reťazcov využíva, hovoríme *metóda nestlačiteľnosti*. Použitím tejto metódy sa podarilo dokázať tvrdenia, ktoré boli desaťročia otvorené. Príkladom je napr. dolný odhad zložitosti Shellsortu v priemernom prípade, dôkaz ktorého nájdete v 7. kapitole.

Dôkazy pomocou metódy nestlačiteľnosti majú typickú štruktúru. Najskôr si z množiny objektov, o ktorých chceme dané tvrdenie dokázať, zvolíme niektorý nestlačiteľný objekt (objekty sú reprezentované reťazcami). Je dôležité uvedomiť si, že my si uvedený reťazec (ako reprezentáciu objektu) s požadovanou zložitosťou síce zvolíme, no nevieme presne povedať, ktorý reťazec z danej množiny to je. V skutočnosti to ani nepotrebujeme. Veta o nestlačiteľnosti nám zaručuje jeho existenciu a to nám postačuje.[4] Keďže taký reťazec v danej množine určite existuje, aj preň musí platiť dokazovaná vlastnosť. Ďalej sporom ukážeme, že ak by požadovaná vlastnosť pre tento objekt neplatila, existoval by preň krátky popis, čím dospejeme k sporu s jeho nestlačiteľnosťou. Dôkazy písané týmto spôsobom sa stávajú jednoduchšie, keďže pracujeme len s jedným zvoleným nestlačiteľným reťazcom.

Iný spôsob ako nestlačiteľnosť reťazca využiť v dôkaze spočíva v tom, že skoro všetky reťazce sú nestlačiteľné. Preto ak tvrdenie dokážeme pre jeden nestlačiteľný reťazec, platí aj pre všetky ostatné prvky z množiny nestlačiteľných reťazcov. Dokázané tvrdenie preto platí v priemere, čo sa využíva napr. pr dokazovaní zložitosti algoritmov v priemernom prípade (viď. spomínaný Shellsort).

Na ilustráciu metódy nestlačiteľnosti si uvedieme jej jednoduché využitie v dôkaze. Dokážeme pomerne elementárne tvrdenie, a síce, že prvočísiel je nekonečne veľa.

[4] Aj keby sme sa snažili presne určiť spomínaný nestlačteľný reťazec, neuspeli by sme, keďže, ako neskôr ukážeme, Kolmogorovskú zložitosť reťazca nevieme presne vypočítať.

Dôkaz. Tvrdenie dokážeme *sporom*. Kvôli sporu predpokladajme, že ich existuje konečne veľa. Nech ich je n a sú to

$$p_1, p_2, p_3, \ldots, p_n.$$

Každé prirodzené číslo k potom môžeme vyjadriť ako súčin prvočísiel nasledovne

$$k = p_1^{\alpha_1}, p_2^{\alpha_2}, \ldots, p_n^{\alpha_n},$$

kde pre každé $1 \leq i \leq n$ platí $0 \leq \alpha_i \leq \log k$ (α_i je nezáporné prirodzené číslo). Zvoľme si také číslo k, že platí

$$l(k) = m \wedge C(k) \geq m.$$

Vieme, že také k existuje. Vďaka tomu, že $\alpha_i \leq \log k$, platí $l(\alpha_i) \leq \log \log k$. Z definície 2.2.4 a rovnosti 2.6 dostávame

$$l\left(\langle \alpha_1, \langle \alpha_2, \langle \alpha_3, \ldots \langle \alpha_{n-2}, \langle \alpha_{n-1}, \alpha_n \rangle \rangle \ldots \rangle \rangle \rangle\right) \leq 2n \log \log k.$$

Z platnosti $l(k) = m$ vyplýva $k \leq 2^{m+1}$, preto

$$l\left(\langle \alpha_1, \langle \alpha_2, \langle \alpha_3, \ldots \langle \alpha_{n-2}, \langle \alpha_{n-1}, \alpha_n \rangle \rangle \ldots \rangle \rangle \rangle\right) \leq 2n \log (m+1).$$

Reťazec $\langle \alpha_1, \langle \alpha_2, \langle \alpha_3, \ldots \langle \alpha_{n-2}, \langle \alpha_{n-1}, \alpha_n \rangle \rangle \ldots \rangle \rangle \rangle$ ako samooddeľujúci popis všetkých exponentov v prvočíselnom rozklade čísla k je jednoznačným popisom tohto čísla. Preto platí

$$C(k) \leq 2n \log (m+1) + c,$$

pre nejakú konštantu c. Pre všetky m okrem konečného počtu dostávame *spor*, pretože sme si zvolili k také, že $C(k) \geq m$. □

4.3 Príklady

Pri výbere príkladov sme čerpali z [16].

Príklad 4.3.1. Dokážte, že pre všetky binárne reťazce x, y platí

$$|C(x+y) - C(x)| = 2l(y) + \mathrm{O}(1) \tag{4.5}$$

Riešenie. Rozdeľme dôkaz rovnosti na dôkaz dvoch nerovností.

„$|C(x+y) - C(x)| \leq 2l(y) + \mathrm{O}(1)$“ Uvažujme dva prípady. Najskôr nech $C(x+y) \geq C(x)$. Dokazovanú nerovnosť môžeme prepísať na

$$C(x+y) \leq C(x) + 2l(y) + \mathrm{O}(1) \tag{4.6}$$

Nech p je popis x, $l(p) = C(x)$. Z reťazca $\overline{y}p$ vieme ľahko vypočítať $x+y$. Reťazec $\overline{y}p$ dekódujeme, spustíme program p, vypočítame x a sčítame ho s y, ktoré už poznáme. Platí $C(x+y) \leq C(x) + 2l(y) + \mathrm{O}(1)$, kde v $\mathrm{O}(1)$ bitoch je zakódovaná aj inštrukcia sčítať x a y.

Ak platí $C(x+y) \leq C(x)$, dôkaz je analogický, len namiesto sčítania x a y bude treba od $x+y$ odčítať y, aby sme získali x. Na záver vidíme, že platí $C(x+y) \leq C(x) + 2l(y) + \mathrm{O}(1)$ aj $C(x) \leq C(x+y) + 2l(y) + \mathrm{O}(1)$, preto platí aj $|C(x+y) - C(x)| \leq 2l(y) + \mathrm{O}(1)$.

„$2l(y) \leq C(x+y) - C(x)| + \mathrm{O}(1)$“ Dôkaz druhej nerovnosti prenechávame na čitateľa ako cvičenie.

Príklad 4.3.2. Nech $C(x) \geq n - \mathrm{O}(1)$, pričom $l(x) = n$. Dokážte, že ak $x = yz$ a $l(z) = l(y)$, tak $C(y) \geq n/2 - \mathrm{O}(1)$ aj $C(z) \geq n/2 - \mathrm{O}(1)$.

Riešenie. Tvrdenie dokážeme *sporom*. Nech aspoň jeden z reťazcov x, y má KZ menšiu ako $n/2 - \mathrm{O}(1)$. Bez ujmy na všeobecnosti môžeme predpokladať, že je to y. Keďže $C(y) < n/2 - \mathrm{O}(1)$, môžeme písať $C(y) = n/2 - f(n)$, pre funkciu $f(n) < \mathrm{O}(1)$. Nech program p_y je najkratším popisom podreťazca $y, l(p_y) = n/2 - f(n)$ a program p_z je najkratším popisom $z, l(p_z) \geq n/2 - \mathrm{O}(1)$. Potom $l(p_y p_z) = n - f(n) - \mathrm{O}(1)$. Pridajme k reťazcu $p_y p_z$ ešte reťazec $p_f = \overline{f(n)} = 1^{l(f(n))}0f(n)$. Reťazec p_f tvorí samooddeľujúci zápis hodnoty $f(n)$.

Nech $p = p_f p_y p_z$. Po prečítaní prvých $2\log f(n) + 1$ bitov reťazca p poznáme hodnotu $f(n)$. Vieme, že ďalej nasleduje $n/2 - f(n)$-bitový reťazec p_y

a $n/2 - \mathrm{O}(1)$-bitový reťazec p_z. Hodnota $f(n)$, ktorú sme získali z p_f nám poslúži na to, aby sme zistili, kde v reťazci $p_y p_z$ (bez oddeľovača) končí zápis p_y. Keď vydelíme hodnotu $l(p_y p_z) = n - f(n) - \mathrm{O}(1)$ dvomi, dostaneme $n/2 - \mathrm{O}(1) - \mathrm{f(n)}/2$. Zistili sme, kde je stred reťazca $p_y p_z$. Vidíme, že posledný bit reťazca p_y je na pozícii $n/2 - \mathrm{O}(1) - \mathrm{f(n)}/2$.

Takto získame podreťazce p_y a p_z, z ktorých vypočítame y a z, ktoré tvoria reťazec $x = yz$.

Z popisu $p = p_f p_y p_z$, $l(p) = n - f(n) - \mathrm{O}(\log \mathrm{f(n)})$ sme získali x. Zrejme však pre všetky $f(n) < \mathrm{O}(1)$ a všetky n okrem konečného počtu platí

$$n - f(n) - \mathrm{O}(\log \mathrm{f(n)}) < \mathrm{n} - \mathrm{O}(1),$$

čím sme dostali *spor* s predpokladom $C(x) \geq n - \mathrm{O}(1)$. Spor mohol spôsobiť len chybný predpoklad $C(y) = n/2 - f(n)$ pre $f(n) < \mathrm{O}(1)$, preto platí $C(y) \geq n/2 - \mathrm{O}(1) \wedge C(x) \geq n/2 - \mathrm{O}(1)$. □

Úloha 4.3.3. Nech $C(x) \geq n - \mathrm{O}(1)$, pričom $l(x) = n$. Dokážte, že ak $x = yz$ a $l(z) = 2l(y)$, tak $C(y) \geq n/3 - \mathrm{O}(1)$ a zároveň $C(z) \geq 2n/3 - \mathrm{O}(1)$.

Úloha 4.3.4. Nech $x = x_1 x_2 \ldots x_{\log n}$, pričom $l(x_i) = n/\log n$ pre všetky $1 \leq i \leq \log n$. Ukážte, že pre všetky $1 \leq i \leq \log n$ platí $C(x_i) \geq n/\log n - \mathrm{O}(\log \log \mathrm{n})$.

Príklad 4.3.5. Nech $C(x) \geq n - \mathrm{O}(1)$, kde $l(x) = n$. Ukážte, že pre všetky rozdelenia x na dve podslová y a z (teda $x = yz$) platí[5]

$$n - \log n - 2 \log \log n \leq C(y) + C(z) \tag{4.7}$$

Riešenie. Nech $p, l(p) = C(y)$, je program na výpočet y a $q, l(q) = C(z)$, nech je program na výpočet z. Reťazec pq na popis x nestačí, keďže z neho nevieme určiť, kde končí program p a kde začína q. Na zápis $p, l(p) \leq n$, stačí n bitov. Na zápis hodnoty $l(p)$ stačí $\log l(p) \leq \log n$. Na špecifikovanie dĺžky zápisu reťazca $l(p)$ použime reťazec $l\,(l\,(p))$ v samooddeľujúcom tvare, tj. $\overline{l\,(l\,(p))} = 1^{l(l(l(p)))}0l\,(l\,(p))$.

[5] až na aditívnu konštantu

Uvedené popisy spojme do jedného popisu P reťazca x.

$$P = \overline{l\,(l\,(p))}l(p)pq = 1^{l(l(l(p)))}0l\,(l\,(p))\,l(p)pq$$

Z popisu P vieme vyčítať postupne hodnoty $l\,(l\,(p))$, $l(p)$, p a na záver q. Z reťazcov p a q ďalej získame reťazec $yz = x$.

Zrejme

$$l(P) \leq 2\log\log n + 1 + \log n + C(y) + C(z).$$

Potom platí

$$\begin{aligned} n - \mathrm{O}(1) \leq \mathrm{C(x)} &\leq 2\log\log \mathrm{n} + 1 + \log \mathrm{n} + \mathrm{C(y)} + \mathrm{C(z)} \\ n - \log n - \log\log n - \mathrm{O}(1) &\leq \mathrm{C(y)} + \mathrm{C(z)}, \end{aligned}$$

čo sme chceli dokázať. □

Úloha 4.3.6. Nech $C(x) \geq n - \mathrm{O}(1)$, kde $l(x) = n$. Ukážte, že existujú také rozdelenia $x, x = yz$ na podslová y a z, pre ktoré platí

$$C(y) + C(z) \leq n - \log n + \log\log n. \tag{4.8}$$

Príklad 4.3.7. Nech $g : \mathbb{N} \to \mathbb{N}$ je neohraničená funkcia. Nech $I(n)$ je počet g-nestlačiteľných reťazcov dĺžky najviac n. Ukážte, že

$$\lim_{n\to\infty} \frac{I(n)}{2^{n+1}} = 1 \tag{4.9}$$

Dokazovaná rovnosť hovorí o tom, že skoro všetky reťazce sú g-nestlačiteľné.

Riešenie. Koľko je g-nestlačiteľných reťazcov dĺžky najviac n? Reťazcov, tým pádom aj popisov dĺžky kratšej ako $n - g(n)$ je

$$\sum_{i=1}^{n-g(n)-1} 2^i = 2^{n-g(n)} - 1$$

Reťazcov dĺžky najviac n, ktoré nemajú popis kratší ako $n - g(n)$ je potom $(2^{n+1} - 1) - (2^{n-g(n)} - 1) = 2^{n+1} - 2^{n-g(n)}$. Pozrime sa potom na hodnotu spomínanej limity.

$$\lim_{n\to\infty} \frac{2^{n+1} - 2^{n-g(n)}}{2^{n+1}} = 1 - \lim_{n\to\infty} \frac{2^{n-g(n)}}{2^{n+1}} = 1 - \lim_{n\to\infty} \frac{1}{2^{g(n)+1}} = 1 \tag{4.10}$$

Keďže g je neohraničená, $1/2^{g(n)+1}$ je pre $n \to \infty$ limitne rovné nule a celá limita bude rovná 1, čo je presne to, čo sme chceli dokázať. □

Príklad 4.3.8. Dokážte, že ku každému binárnemu reťazcu x dĺžky n existuje reťazec y, ktorý sa od x líši len v jednom bite a súčasne

$$C(y|n) \leq n - \log n + \mathrm{O}(1). \tag{4.11}$$

Riešenie. Nech n-bitové binárne reťazce x a y sa navzájom líšia len v jednom bite.[6] Tento fakt budeme označovať ako $\Delta(x, y) = 1$. Nech H_x je množina všetkých n-bitových binárnych reťazcov y, pre ktoré platí $\Delta(x, y) = 1$. Množina H_x obsahuje n reťazcov, keďže každý z nich sa od x líši práve v jednom bite.

Poznámka 4.3.1. Uveďme si príklad. Množinu H_z pre 10 bitový reťazec $z =$ 0011010011 tvorí týchto 10 reťazcov: **1**011010011, 0**1**11010011, 00**0**1010011, 001**0**010011, 0011**1**10011, 00110**0**0011, 0011011011, 0011010**1**11, 0011010**0**01, 0011010010.
Zvýraznené sú tie bity, v ktorých sa daný reťazec líši od reťazca z.

Koľko navzájom disjunktných množín H_x v priestore všetkých 2^n binárnych reťazcov dĺžky n môžeme vytvoriť? Keďže každú z nich tvorí n reťazcov, spolu ich je $2^n/n$. Množiny H_y vieme ľahko generovať tak, že budeme generovať všetky binárne reťazce dĺžky n. Keď vygenerujeme reťazec x_j, pre ktorý $\Delta(x_j, y = 1)$, pridáme ho do H_y. Ak vygenerujeme reťazec x_j, pre ktorý $\Delta(x_j, x_i = 1)$ pre nejaký už vygenerovaný a zaradený x_i, ignorujeme ho.

Už vieme ako generovať reťazce y také, že pre každé x existuje y také, že $\Delta(x, y) = 1$. Zostáva dokázať, že pre nájdené y platí $C(y|n) \leq n - \log n + \mathrm{O}(1)$.

Ak máme algoritmus, ktorým y generujeme (ak poznáme n, aby sme vedeli aké dlhé reťazce generovať), popísať y už vieme ľahko. Stačí špecifikovať

[6] Tejto vlastnosti hovoríme aj *hammingovská vzialenosť* reťazcov x a y (Richard Wesley Hamming, [6], 1950) V tomto prípade je Hammingovská vzdialenosť x a y rovná 1. O Hammingovych kódoch sa dočítate napr. aj v [9] a [10].

poradové číslo reťazca v enumerácii popísanej uvedeným algoritmom. Keďže vygenerovaných reťazcov je $2^n/n$, poradové číslo vieme zapísať na

$$\log 2^n/n = \log 2^n - \log n = n - \log n$$

bitov. Na zápis algoritmu (plus diskusiu) potrebujeme ešte O(1) bitov. Preto platí

$$C(y|n) \leq n - \log n + \mathrm{O}(1). \quad \square$$

Príklad 4.3.9. Nech A je množina binárnych reťazcov dĺžky n, B je jej podmnožina. Ukážte, že ak $x \in B$, tak

$$\log \frac{|A|}{|B|} - C(B|A) \leq \delta(x|A) + \mathrm{O(log\,n)} \tag{4.12}$$

Riešenie. Do predchádzajúceho vzťahu dosaďme za $\delta(x|A)$ z definície $\delta(x|A) = \log d(A) - C(x|A)$ a upravme ho.

$$\begin{aligned} \log|A| - \log|B| - C(B|A) &\leq \log|A| - C(x|A) + \mathrm{O(log\,n)} \\ -\log|B| - C(B|A) &\leq -C(x|A) + \mathrm{O(log\,n)} \\ C(x|A) &\leq C(B|A) + \log|B| + \mathrm{O(log\,n)} \end{aligned}$$

Pri prvej úprave sme využili aj fakt, že $\delta(x|A) = \log d(A) - C(x|A)$. Vďaka týmto úpravám jasnejšie vidíme ako potrebujeme zhora ohraničiť $C(x|A)$.

Nech $p, l(p) = C(B|A)$ je popis množiny B ak poznáme množinu A. Nech je to napr. popis enumerácie jej prvkov (alebo samotný výpis všetkých jej prvkov). Ľubovoľný jej prvok vieme špecifikovať pomocou $\log|B| \leq \log 2^n = n$ bitov uvedením jeho poradového čísla i v popísanej enumerácii. Hodnotu $l(i) = \log|B| \leq n$ špecifikujeme reťazcom $\overline{l(i)}$, pričom platí $l(\overline{l(i)}) \leq 2\log n + 1$.

Po skombinovaní uvedených popisov do reťazca

$$P = 1^{\log l(i)} 0 l(i) i p$$

získame popis reťazca x, ak poznáme A. Platí preto

$$C(x|A) \leq 2\log\log n + 1 + \log|B| + C(B|A) = C(B|A) + \log|B| + \mathrm{O(log\,n)}.$$

$\square$

Definícia 4.3.2. Reťazec x, pre ktorý platí $l(x) = n$ a súčasne $x = n0^{n-l(n)}$, voláme *n-reťazec.*

Príklad 4.3.10. (a) Ukážte, že pre každý n-reťazec existuje taká konštanta c, že $C(x|n) \leq c$.[7]

Riešenie. Nech p je program, ktorý pozná hodnotu n. Nech program p pracuje tak, že vypíše hodnotu n a výstup doplní nulami tak, aby dĺžka výstupu bola n. Vypíše teda reťazec $n0^{n-l(n)}$. Dĺžka p je zrejme konštantná. Ľubovoľný n-reťazec x preto vieme vypísať programom konštantnej dĺžky, ak poznáme n. □

(b) Vezmime si konštantu c ako v predošlom prípade a reťazec x dĺžky n taký, že $C(x|n) \gg c$. Ukážte, že reťazec y v tvare $y = x00\ldots0$, pričom $l(y) = x$ má zložitosť $C(y|x) \leq c$.

Riešenie. Nech x je ľubovoľný nestlačiteľný reťazec dĺžky n, platí $C(x|n) \geq n \gg c$. Vidíme, že T_k z predošlého bodu poznajúc x má na svojom výstupe reťazec tvaru $x00\ldots0$ dĺžky práve x. Označme ho z. Pripomíname, že x môžeme považovať za (binárne) číslo aj za (binárny) reťazec. Program pre univerzálny TS má preto dĺžku c tak ako aj v predošlom prípade, a preto $C(y|x) \leq c$. □

(c) Uvažujme x v tvare $n-$bitového prefixu slova $nn\ldots n$. Ukážte, že existuje konštanta c taká, že pre všetky n platí $C(x|n) \leq c$.

Riešenie. Na zápis x v požadovanom tvare stačí jednoduchý algoritmus vykonávaný Turingovym strojom T_j. Keďže hodnotu n pozná, na výstup bude cyklicky vypisovať bit po bite n. Ak už vypíše posledný bit n, pokračuje opäť prvým. Pri tom si počíta koľko bitov už vytlačil. Výpočet končí, keď vytlačí n-tý bit. □

Poznámka 4.3.3. Z práve dokázaných tvrdení plynie, že funkcia $C\left(x|l\left(x\right)\right)$ nie je monotónna na prefixoch. To znamená, že existujú prirodzené čísla m

[7] c záleží len od voľby referenčného TS

a n, $m < n$, pre ktoré neplatí $C(m|l(m)) \leq C(n|l(n))$. Vezmime si nestlačiteľné $m, C(m) \geq l(m)$ a $n, n = m0^{m-l(m)}$. Potom platí $C(n|l(n)) \leq c$. Z platnosti $C(m) \leq C(m|l(m)) + C(l(m))$ ale dostávame $C(m|l(m)) \geq C(m) - C(l(m)) \geq \log m - 2\log\log m$.

Príklad 4.3.11. Ukážte, že existuje taká konštanta $d > 0$, že pre každé n existuje aspoň $\lfloor 2^n/d \rfloor$ reťazcov x dĺžky n, pre ktoré platí

$$C(x|n) \geq n. \tag{4.13}$$

Riešenie. Nech platí $l(x) \leq n - c$, potom $C(x|n) \leq n$. Existuje $2^n - 1$ možných programov dĺžky menšej ako n pre reťazce dĺžky n a $2^{n-c} - 1$ možných programov dĺžky menšej ako $n - c$ pre reťazce dĺžky $n - c = l(x)$. Spolu zostáva najviac $2^n - 2^{n-c}$ programov dĺžky menšej ako n pre reťazce dĺžky n. Z toho vyplýva, že spomedzi všetkých 2^n reťazcov dĺžky n je aspoň $2^n - (2^n - 2^{n-c}) = 2^{n-c}$ takých, pre ktoré platí $C(x|n) \geq n$. Hľadaná konštanta d má hodnotu 2^c.

Označme m počet reťazcov dĺžky n takých, že $C(x|n) \geq n$. Ak poznáme m a n, vieme enumerovať všetkých $(2^n - m)$ n-bitových reťazcov so zložitosťou $C(x|n) < n$ tak, že budeme postupne spúšťať všetky programy dĺžky $< n$. Výpočet programov zastavíme, keď už $2^n - m$ z nich zastaví. Všimnime si, vieme kedy máme prestať vykonávať programy, keďže vieme koľko z nich má zastaviť. KZ ostatných m reťazcov je $\geq n$. Pre lexikograficky prvý reťazec, ktorý nebude výstupom žiadneho programu, ktorý zastavil, platí

$$\log m + \mathrm{O}(1) \geq \mathrm{C(x|n)} \geq \mathrm{n}.$$

□

Príklad 4.3.12. Ukážte, že existujú také konštanty $c, d > 0$, že pre každé dosť veľké n existuje aspoň $\lfloor 2^n/d \rfloor$ reťazcov x dĺžky $n - c \leq l(x) \leq n$, pre ktoré platí

$$C(x|n) > n. \tag{4.14}$$

Riešenie. Tak ako existuje $2^{n+1} - 1$ reťazcov dĺžky najviac n, aj možných programov dĺžky najviac n, ktoré ich popisujú je $2^{n+1} - 1$. Keďže nie všetky z týchto programov musia zastaviť, pre dosť veľké n existuje taký reťazec dĺžky $l(x) \leq n$, že $n < C(x|n) \leq l(x) + c$. Nech takých reťazcov existuje m. Ak poznáme m a n, vieme enumerovať všetkých $2^{n+1} - 1 - m$ reťazcov x dĺžky najviac n, pre ktoré platí $C(x|n) \leq n$ tak ako v predošlej úlohe. Dôkaz ďalej pokračuje obdobne ako v predchádzajúcej úlohe. □

Úloha 4.3.13. Ukážte tvrdenia v predošlých dvoch úlohách (4.3.11 a 4.3.12) pre nepodmienenú zložitosť, tzn. pre $C(x) \geq n$, resp. $C(x) > n$.

Poznámka 4.3.4. Požadovaný dôkaz tohto tvrdenia ako aj dôkazy v úlohách 4.3.11 a 4.3.12 nájdete v [3].

Kapitola 5

Kolmogorovská zložitosť ako funkcia

5.1 Definície a základné vlastnosti

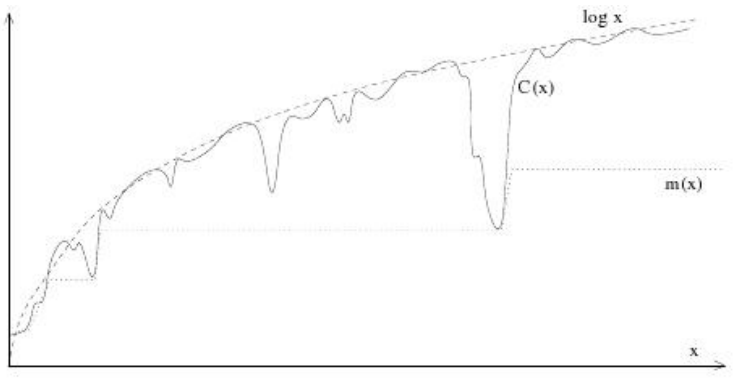
V tejto kapitole sa pozrieme na Kolmogorovskú zložitosť ako na celočíselnú funkciu $C : \mathbb{N} \to \mathbb{N}$. Triviálny odhad $C(x)$ zhora už poznáme, existuje taká konštanta c, že pre všetky $x \in \mathbb{N}$ platí $C(x) \leq l(x) + c$. Z Vety o nestlačiteľnosti 4.1.5 (str. 45) tiež vieme, že pre väčšinu reťazcov je tento odhad pomerne presný.

Uveďme si ďalšie tri vlastnosti Kolmogorovskej funkcie:

(a) Funkcia $C(x)$ je neohraničená.

(b) Funkcia $m(x) = \min\{C(y) : y \geq x\}$ je tiež neohraničená. Funkcia m je najväčšou monotónne rastúcou funkciou, ktorá ohraničuje C zdola.

(c) Nech $\phi(x)$ je ľubovoľná čiastočne rekurzívna funkcia, ktorá od nejakého x_0 monotónne rastie do nekonečna. Potom pre všetky x okrem konečného počtu platí $m(x) < \phi(x)$. Znamená to, že hoci m rastie do nekonečna, rastie pomalšie ako ktorákoľvek čiastočne rekurzívna funkcia.



Obr. 5.1: Porovnanie funkcií $\log x, C(x)$ a $m(x)$. Zdroj: [16].

Dôkaz týchto troch tvrdení môžete nájsť v [16].
Obrázok 5.1 ilustruje porovnanie funkcií $\log x, C(x)$ a $m(x)$.
Ďalšiu dôležitú vlastnosť KZ ukážeme vo vete o nevypočítateľnosti $C(x)$.

Veta 5.1.1. *Funkcia $C(x)$* nie je *rekurzívna. Navyše, žiadna čiastočne rekurzívna funkcia $\phi(x)$ definovaná na nekonečnej množine sa nezhoduje s $C(x)$ na celom svojom definičnom obore.*

Dôkaz. Tvrdenie dokážeme *sporom*, predpokladajme, že $C(x)$ *je* rekurzívna a ČRF $\phi(x)$ je s ňou na celom svojom nekonečnom definičnom obore totožná. Nech A je nekonečná rekurzívna podmnožina definičného oboru ϕ. Potom funkcia

$$\psi(m) = \min\{x : C(x) \geq m, x \in A\}$$

je totálna rekurzívna, lebo $C(x) = \phi(x)$ pre $x \in A$. Podľa definície ψ zároveň platí $C(\psi(m)) \geq m$. Z vety 3.1.5 platí $C(\psi(m)) \leq C_\phi(\psi(m)) + c_\phi$. Keďže zrejme $C_\phi(\psi(m)) \leq l(m)$, dostávame

$$m \leq C(\psi(m)) \leq l(m) + c_\psi,$$

čo platí len pre niekoľko malých m. □

Uvedieme ešte jednu vetu, ktorá hovorí o tom, že napriek tomu, že $C(x)$ nevieme vypočítať, vieme ju aproximovať.

Veta 5.1.2. *Existuje totálna rekurzívna funkcia $\phi(t,x)$ monotónne klesajúca v t, pre ktorú platí*

$$\lim_{t\to\infty}\phi(t,x) = C(x).$$

Dôkaz. Vieme, že pre každé x platí $C(x) \leq l(x) + c$. Budeme po krokoch simulovať všetky programy $p, l(p) \leq l(x) + c$ až po t-ty krok. Definujme $\phi(t,x)$ ako dĺžku najkratšieho programu p, ktorého výpočet na x zastaví najviac po t krokoch, inak $\phi(t,x) = l(x) + c$. Znamená to, že po zbehnutí t krokov všetkých programov dĺžky $\leq l(x) + c$ na reťazci x, bude hodnota $\phi(t,x)$ rovná dĺžke najkratšieho p, ktorý na x zastavil. Ak žiadny program na x po t krokoch ešte nezastavil, $\phi(t,x) = l(x) + c$.

Funkcia $\phi(t,x)$ aproximuje $C(x)$ zhora. Čím väčšie t zvolíme, tým lepšiu aproximáciu dostaneme, zrejme totiž pre všetky x platí $\phi(t_i,x) \leq \phi(t_j,x)$, pre $t_i > t_j$. Funkcia $\phi(t,x)$ je totálna, rekurzívna, monotónne klesajúca v t. □

Uvedomme si, že nevieme rozhodnúť, či $\phi(t,x) = C(x)$. Ak by sme vedeli, dokázali by sme počítať hodnotu $C(x)$. Platí len, že $\phi(t,x)$ sa limitne blíži k $C(x)$ pre $t \to \infty$.

5.2 Príklady

Pri voľbe príkladov sme čerpali z [16].

Definícia 5.2.1. Nech $g_1, g_2, g_3, \ldots$ je postupnosť funkcií. Funkciu f nazveme *limitou* tejto postupnosti, ak platí

$$f(x) = \lim_{t\to\infty} g_t(x). \tag{5.1}$$

Limita f postupnosti týchto funkcií sa nazýva *rekurzívne uniformná*, ak pre každé racionálne ϵ existuje totálna rekurzívna funkcia $t(\epsilon)$ taká, že pre $\forall x$ platí

$$|f(x) - g_{t(\epsilon)}(x)| \leq \epsilon. \tag{5.2}$$

Príklad 5.2.1. Nech $\phi(t,x)$ je rekurzívna funkcia a $\lim_{t\to\infty}\phi(t,x) = C(x)$ pre všetky x. Pre každé t definujme $\psi_t(x) = \phi(t,x)$. Funkcia $C(x)$ je potom limitou postupnosti funkcií $\psi_1, \psi_2, \psi_3, \ldots$ Ukážte, že pre ľubovoľné ϵ a každé t platí

$$\psi_t(x) - C(x) > \epsilon$$

pre nekonečne veľa x.

Riešenie. Nerovnosť dokážeme *sporom*. Nech pre ľubovoľné ϵ a každé t platí $\psi_t(x) - C(x) > \epsilon$ pre nekonečne veľa x, čo znamená, že $C(x)$ je rekurzívne uniformnou limitou uvedenej postupnosti funkcií. Keďže potom pre *ľubovoľne* malé ϵ platí

$$|C(x) - \psi_t(x)| \leq \epsilon,$$

vedeli by sme počítať $C(x)$. Z vety 5.1.1 však vieme, že $C(x)$ nie je rekurzívna. Dostali sme spor. Pre nekonečne veľa x preto platí $\psi_t(x) - C(x) > \epsilon$ pre ľubovoľné ϵ a každé t. □

V ďalšej úlohe sa pozrieme na vlastnosti zložitosti s mierne pozmenenou definíciou. Budeme hľadať popis, resp. program pre reťazec, ktorý dokáže vygenerovať nie len reťazec samotný, ale ľubovoľný jeho prefix.

Definícia 5.2.2. Definujme *uniformnú zložitosť* reťazca konečnej dĺžky n vzhľadom na ϕ ako

$$C_\phi(x;n) = \min\{l(p) : \phi(m,p) = x_{1:m} \text{ pre } \forall m \leq n\}. \tag{5.3}$$

Slovne môžeme povedať, že je to dĺžka programu p (pre funkciu ϕ z enumerácie všetkých čiastočne rekurzívnych funkcií) schopného generovať m-bitové prefixy[1] slova x, pre všetky $m \leq n$. Ak taký program neexistuje, $C_\phi(x;n) = \infty$.

Úloha 5.2.2. Ukážte, že aj pre práve definovaný variant zložitosti si môžeme zvoliť referenčnú univerzálnu funkciu ϕ_0, pre ktorú bude platiť $C_{\phi_0}(x;n) \leq C_\phi(x;n) + c$.

[1]Turingov stroj môže mať hodnotu m uloženú napríklad na pomocnej páske.

Na základe tvrdenia dokázaného v predošlej úlohe definujeme

$$C(x;n) = C_{\phi_0}(x;n). \tag{5.4}$$

Vidíme, že program p v tejto definícii dokáže robiť viac ako programy, ktoré nás zaujímali doteraz. Ich dĺžka sa preto zrejme bude trochu líšiť. Túto domnienku dokážeme v nasledujúcom príklade.

Príklad 5.2.3. Ukážte, že pre všetky konečné reťazce x platí

$$C(x|l(x)) \leq C\left(x;l\left(x\right)\right) \leq C(x),$$

až na konštanty nezávislé od x.

Riešenie. Začneme druhou nerovnosťou.

(a) $C\left(x;l\left(x\right)\right) \leq C(x) + \mathrm{O}(1)$
Nech program p je popisom reťazca x, pričom $l(p) = C(x)$. Chceme ho upraviť tak, aby dokázal vypísať ľubovoľný prefix slova x dĺžky m pre všetky $m \leq n$.
Upravíme ho tak, že potom ako vypíše na (pomocnú) pásku celý reťazec x, môže prečítať hodnotu m a odstráni z vypísaného n posledných $n-m$ bitov. Výslednú reťazec odovzdá ako svoj výstup. Takto dostaneme len prvých m bitov slova x, čo je presne to, čo potrebujeme. Táto jednoduchá úprava si vyžiada len (1) bitov.

(b) $C(x|l(x)) \leq C\left(x;l\left(x\right)\right) + (1)$
V tomto prípade upravíme program p dĺžky $C\left(x;l\left(x\right)\right)$ tak, aby so znalosťou dĺžky x tento reťazec vedel vypísať. Ale program p predsa dokáže vypísať ľubovoľný prefix dĺžky $m \leq n$ slova x. Preto dokáže vypísať m-bitový prefix aj pre $m = n = l(x)$, ktorým je reťazec $x_{1:n}$, čo je práve x. □

Príklad 5.2.4. Ukážte, že na rozdiel od $C(x|l(x))$, pre $C(x;l(x))$ neexistuje taká konštanta c, že pre všetky n-reťazce x platí $C(x;l(x)) \leq c$.

Riešenie. Existenciu takej konštanty pre $C(x|l(x))$ sme dokázali v úlohe 4.3.10 (str. 54). Aby sme dokázali (ne)existenciu požadovanej konštanty, potrebujeme nájsť program p, ktorý dokáže vypísať nielen celý n-reťazec, ale aj ľubovoľný jeho prefix. Pre lepšie pochopenie môžeme tvar n-reťazca pre ľubovoľné n zapísať ako

$$n_1 n_2 n_3 \dots n_{l(n)} \underbrace{000\dots0}_{(n-l(n))},$$

kde n_i označuje i-ty bit čísla n.
Potom m-bitový prefix tohto reťazca je tvaru

$$\begin{aligned} n_1 n_2 n_3 \dots n_m & \quad \text{pre } m \leq l(n) \\ n_1 n_2 n_3 \dots n_{l(n)} \underbrace{000\dots0}_{(m-l(n))} & \quad \text{pre } m > l(n) \end{aligned}$$

Vidíme, že aj na zápis m-bitového prefixu n-reťazca potrebujeme poznať hodnotu n, aby sme vedeli písať jednotlivé n_i. Poznáme však len hodnotu $m \leq n$, ktorá nám o n nič nehovorí (okrem nerovnosti $m \leq n$). V prípade $C(x; l(x))$ hodnotu n nepoznáme a na konštantný počet bitov zrejme nevieme zapísať program, ktorý by poznal všetky n.

Neexistuje teda žiadna konštanta c, pre ktorú by platilo $C(x; l(x)) \leq c$ pre *všetky* n-reťazce x. □

Príklad 5.2.5. Ukážte, že $C(x; l(x))$ na rozdiel od $C(x|l(x))$ *je* monotónna na prefixoch. To znamená, že ak $m \leq n$, tak $C(x_{1:m}; m) \leq C(x_{1:n}; n)$ pre všetky x.

Riešenie. Túto vlastnosť ukážeme pomerne ľahko. Ak totiž máme program p, ktorý dokáže vypísať ľubovoľný k-bitový prefix pre $k \leq n$, dokáže zrejme vypísať aj k-bitový prefix pre $k \leq m \leq n$. □

Definícia 5.2.3. Definujme stavovú zložitosť konečného binárneho reťazca x ako najmenšie n také, že existuje Turingov stroj s n stavmi, ktorý na slove x zastaví. Budeme ju označovať $S(x)$. Túto definíciu môžeme prepísať ako

$$S(x) = \min \{n : |Q_T| = n, T(x) < \infty\} \tag{5.5}$$

Ďalej definujme množinu B

$$B = \{\langle x, p\rangle \,|S(x) \leq p\} \tag{5.6}$$

B je teda množina všetkých dvojíc $\langle x, p\rangle$, pre ktoré platí, že existuje TS s počtom stavov menším alebo rovným p, ktorý zastaví na slove x.

Príklad 5.2.6. Dokážte, že práve definovaná množina B je rekurzívne vyčísliteľná, ale nie je rekurzívna.

Riešenie. Množina je rekurzívne vyčísliteľná práve vtedy, keď (je prázdna alebo) je oborom hodnôt nejakej totálnej rekurzívnej funkcie. Znamená to, že existuje Turingov stroj, ktorý postupne generuje všetky prvky tejto množiny. Prvky generuje v nejakom poradí a niektoré prvky môže vygenerovať viackrát. V konečnom čase však vygeneruje každý prvok množiny aspoň raz.

Množina je rekurzívna, ak jej charakteristická funkcia je rekurzívna. Znamená to, že existuje Turingov stroj, ktorý nám povie, či prvok a patrí do množiny alebo nie. V oboch prípadoch nám určite dá pravdivú odpoveď. To je zásadný rozdiel medzi rekurzívnymi a rekurzívne vyčísliteľnými množinami. Každá rekurzívna množina je teda aj rekurzívne vyčísliteľná. Pre každú rekurzívnu množinu platí, že ona sama ako aj jej doplnok sú rekurzívne vyčísliteľné (keďže o každom prvku mimo množiny vieme, že patrí do jej doplnku, vieme generovať aj doplnok).

Chceme dokázať, že množina B je rekurzívne vyčísliteľná, potrebujeme preto nájsť algoritmus, ktorý bude generovať všetky jej prvky. Pri vytváraní tohto algoritmu využijeme ideu spúšťania TS na rôznych vstupoch po jednotlivých krokoch výpočtu popísanú v 3.2.12. Budeme spúšťať rôzne TS na rôznych vstupoch a vždy, keď niektorý z nich ukončí svoj výpočet, získame dvojicu $\langle$slovo, počet stavov$\rangle$.

Rozdielom medzi modelom, ktorý sme používali doteraz a tým, ktorý použijeme teraz, je, že tentokrát budeme spúšťať rôzne TS na rôznych vstupoch, zatiaľ čo pred tým sme spúšťali jeden TS na rôznych vstupoch. Budeme preto potrebovať nielen enumeráciu binárnych reťazcov, ale aj enumeráciu Turingovych strojov. Celý výpočet názorne ilustruje obrázok 5.2, ktorý znázorňuje,

	$T_1(\omega_1)$	$T_1(\omega_2)$	$T_2(\omega_1)$	$T_2(\omega_2)$	$T_1(\omega_3)$	$\cdots$
1	1	3	6	10	15	
2	2	5	9	14		
3	4	8	13			
4	7	12				
5	11					
$\vdots$						

Obr. 5.2: Tabuľkové znázornenie použitej upravenej metódy. Kroky sú vykonávané postupne v poradí ako je uvedené v jednotlivých políčkach.

ako budeme jednotlivé výpočty spúšťať. Riadky reprezentujú jednotlivé kroky výpočtu TS. V každom stĺpci prebieha výpočet jedného TS na jednom vstupnom slove.

Najskôr spustíme Turingov stroj, ktorý bude postupne generovať všetky TS T_i aj Turingov stroj, ktorý bude generovať všetky binárne reťazce ω_j (vstupy pre generované Turingove stroje). Generované Turingove stroje tak ako aj slová môžeme číslovať v poradí, v akom sú generované. Pripomíname, že nevieme v akom poradí budú objekty generované. Ak bude ktorýkoľvek z týchto objektov vygenerovaný znovu, ignorujeme ho. Vždy, keď sa vygeneruje nový stroj T_i, spustia sa postupne výpočty $T_i(\omega_j)$ pre všetky slová ω_j, ktoré už boli vygenerované. Podobne keď sa vygeneruje nové slovo ω_j, spustia sa postupne výpočty $T_i(\omega_j)$ pre všetky TS T_i, ktoré už boli vygenerované.[2]

[2]Znamená to, že po vygenerovaní nového objektu (TS alebo slova) do tabuľky pribudnú pre ne ďalšie stĺpce.

Takto zabezpečíme, že v konečnom čase spustíme každý TS z enumerácie TS na každom slove z enumerácie binárnych reťazcov.

Keď niektorý z výpočtov v konečnom čase zastaví, získame prvú dvojicu $\langle x, n\rangle$, ktorá hovorí, že existuje TS s n stavmi, ktorý zastaví svoj výpočet na slove x. Nezistili sme tým hodnotu $S(x)$, no získali sme horný odhad tejto hodnoty. Zistili sme totiž, že existuje TS s n stavmi, ktorý zastaví na slove x, čo znamená, že hodnota $S(x)$ už nemôže byť väčšia ako n. Dvojica $\langle x, n\rangle$ preto patrí do B. Platí totiž $S(x) \leq n$. Zároveň však vieme, že do B patria aj všetky dvojice $\langle x, m\rangle$, kde $m > n$, lebo $S(x) \leq n < m$.

Keď opäť niektorý TS (môže to byť ten istý TS alebo ktorýkoľvek iný) zastaví svoj výpočet na slove x, porovná sa jeho počet stavov s najmenším počtom stavov doteraz nájdeného TS, ktorý už zastavil svoj výpočet na slove x. Ak sme našli TS s menším počtom stavov, do množiny B môžeme pridať ďalšie dvojice – tú, ktorú sme práve našli $\langle x, n\rangle$ a aj všetky $\langle x, m\rangle$ pre $m > n$, ktoré v B ešte nie sú. V prípade, že TS s počtom stavov n zastavil svoj výpočet na inom slove y, opäť sme získali horý odhad hodnoty funkcie $S(y)$ a do B môžeme pridať popri dvojici $\langle y, n\rangle$ aj všetky $\langle y, m\rangle$ pre $m > n$.

Z popisu algoritmu je zrejmé, že každá dvojica, ktorú do B vložíme, tam naozaj patrí a zároveň každú dvojicu, ktorá do B podľa jej popisu patriť má, vygenerujeme a vložíme do B. Postupne takto generujeme množinu B.

Ukázali sme, že množina B *je* rekurzívne vyčísliteľná. Ostáva len si uvedomiť, že nie je rekurzívna.

Ak máme rozhodnúť, či dvojica $\langle x, n\rangle$ patrí alebo nepatrí do B, musíme zistiť, či existuje nejaký TS T_l s n stavmi, ktorý akceptuje x. Problém zastavenia Turingovho stroja je nerozhodnuteľný (veta 2.3.1), nevieme preto rozhodnúť, či stroj T_l na vstupe x zastaví. Množina B je preto rekurzívne vyčísliteľná, ale nie je rekurzívna. □

Kapitola 6

Náhodné postupnosti

6.1 Definície a základné vlastnosti

Ako sme už spomínali v kapitole 4 o nestlačiteľnosti, nestlačiteľnosť reťazca implikuje jeho náhodnosť. V tejto kapitole si povieme niečo viac o testoch náhodnosti,

Predpokladajme, že ak z nejakej množiny reťazcov náhodne zvolíme jeden reťazec, bude to v istom zmysle typický reťazec z danej množiny. Znamená to, že ak vhodne zvolíme kritériá a podľa týchto kritérií množinu rozdelíme, tento náhodne zvolený reťazec bude patriť podľa každého kritéria do väčšiny v tejto množine. Ak by patril len do nejakej malej špecifickej podmnožiny, nemožno ho považovať za typický/náhodný (náhodný z pohľadu Kolmogorovskej zložitosti). Túto ideu využijeme pri definícii testu náhodnosti.

Nech P je pravdepodobnostné rozdelenie nad priestorom S. Chceme rozhodnúť, či nejaký reťazec $x \in S$ patrí do väčšiny $M \subseteq S$. Každé z týchto rozhodnutí učiníme na určitom stupni významnosti ϵ, kde $\epsilon = 1 - P(M)$. Test definujeme tak, aby vedel, pre ktoré x má hypotézu „$x \in S$ patrí do väčšiny $M \subseteq S$“ zamietnuť.

Definícia 6.1.1. Množinu $V \subseteq N \times S$ tzv. *kritických regiónov* definujeme

ako

$$\begin{aligned} V_m &= \{x : (m, x) \in V\} \\ V_m &\supseteq V_{m+1}, \text{ pre } m = 1, 2, 3, \ldots \end{aligned}$$

Hovoríme, že V_m je kritický región na *úrovni významnosti*[1] $\epsilon = 2^{-m}$, čomu zodpovedá

$$(\forall n) \sum_x \{P(x|l(x) = n) : x \in V_m\} \leq \epsilon \tag{6.1}$$

Ak $x \in V_m$, hypotéza „x patrí do väčšiny M," resp. „x je náhodné" je zamietnutá na stupni významnosti $\epsilon = 2^{-m}$. Hovoríme taktiež, že x zlyhal v teste na úrovni kritického regiónu V_m.

Čím väčšie m si zvolíme, tým spoľahlivejší výsledok dostaneme. Doplnok ku kritickému regiónu V_m sa preto tiež nazýva aj $(1-\epsilon)$ *interval spoľahlivosti.*

Test náhodnosti si teraz definujeme formálne a tak, aby sme získali *efektívny* spôsob ako rozhodovať o náhodnosti reťazcov.

Definícia 6.1.2. Nech P je rekurzívne pravdepodobnostné rozdelenie na priestore $\mathbb{N}$. Totálna funkcia $\delta : \mathbb{N} \to \mathbb{N}$ je P-test[2] práve vtedy, keď

(a) množina $V = \{(m, x) : \delta(x) \geq m\}$ je rekurzívne vyčísliteľná

(b) $(\forall n) \sum_x \{P(x|l(x) = n, \delta(x) \geq m)\} \leq 2^{-m}$.

Kritický región V_m je potom $V_m = \{x : \delta(x) \geq m\}$, pre $m \geq 1$. Vlastnosť $V_{m+1} \subseteq V_m$ pre $m \geq 1$ vyplýva z toho, že $\delta(x) \geq m + 1$ implikuje $\delta(x) \geq m$. Všimnime si taktiež, že v prípade rovnomerného rozdelenia nám namiesto ukázania vlastnosti (b) stačí ukázať

$$d(\{x : l(x) = n, x \in V_m\}) \leq 2^{n-m}.$$

Ak totiž túto mohutnosť predelíme počtom všetkých n-bitových binárnych reťazcov (ktorých je 2^n), dostaneme práve 2^{-m}.

[1] alebo kritický región pre úroveň významnosti ϵ, prípadne len kritický región úrovne významnosti ϵ

[2] nazývaný aj Martin-Löfov test náhodnosti. Pomenované po Švédskom matematikovi Per Martin-Löfovi, ktorému vďačíme za mnoho výsledkov týkajúcich sa testov náhodnosti.

6.2 Príklady

Príklad 6.2.1. Nech $x = x_1x_2 \dots x_n$ je náhodná (binárna) postupnosť, pričom $C(x|n) \geq n$. Ukážte, že postupnosť x tvare $x_10x_30x_5 \dots x_{n-1}0x_n$ *nie je* náhodná vzhľadom na rovnomerné rozdelenie. (Zdroj: [16])

Riešenie. Test budeme chcieť definovať ako totálnu funkciu vracajúcu počet núl nachádzajúcich sa na párnych bitoch postupnosti x. Definujme si P-test, kde P je rovnomerné rozdelenie, nasledovne.

$\delta(x) =$ najväčšie i také, že $x_2 = x_4 = \ldots = x_{2i} = 0$. Ak $x_2 = 0$, tak $\delta(x) = 0$. Treba ukázať, že takto definovaná funkcia $\delta(x)$ skutočne je P-test.

V prvom rade treba ukázať, že $V = \{(m, x) : \delta(x) \geq m\}$ je rekurzívne vypočítateľná. Táto vlastnosť je zrejmá. Pre každé x a m totiž vieme jednoznačne vyhlásiť, či $\delta(x) \geq m$, stačí sa pozrieť na párne pozície a overiť, či na každej z nich (vzostupne) je nula.

Ak chceme ukázať aj druhú potrebnú vlastnosť, pozrieme sa koľko existuje takých n-bitových reťazcov x, že $\delta(x) \geq m$. Keďže $\delta(x) \geq m$ a dĺžka x je n, platí $n \geq 2m$, lebo nuly sa nachádzajú minimálne na prvých m párnych bitoch x. Nevieme čo sa nachádza na nepárnych bitoch, taktiež nevieme aké hodnoty majú bity na vzdialenejších pozíciách, tzn. pozíciách s indexom väčším ako $2m$. Prefix $x_{1:2m}$ obsahuje m nepárnych bitov, ktoré môžu tvoriť jeden z 2^m možných podreťazcov. Celý reťazec x však tvorí celkom n bitov. Zostávajúcich $n - 2m$ bitov môže preto tvoriť jeden z 2^{n-2m} možných podreťazcov. Potom reťazcov s $\delta(x) \geq m$, ktoré nás zaujímajú môže byť celkom

$$2^m \cdot 2^{n-2m} \leq 2^{n-m}. \tag{6.2}$$

Teraz ešte tento počet predelíme počtom všetkých možných n-bitových reťazcov (2^n) a získame pravdepodobnosť s akou spomedzi všetkých n-bitových binárnych reťazcov nájdeme práve taký, že $\delta(x) \geq m$. Dostaneme $2^{n-m}/2^n = 2^{-m}$. □

Úloha 6.2.2. Pomocou Martin-Löfovho testu ukážte, že reťazec $x_1x_2 \dots x_n$, ktorý na začiatu obsahuje veľa núl, nie je náhodný vzhľadom na rovnomerné rozdelenie. (Zdroj: [16])

Kapitola 7

Zaujímavé aplikácie Kolmogorovskej zložitosti

V nasledovnom texte definujeme pojem *informačná vzdialenosť* a pomocou Kolmogorovskej zložitosti a hranového farbenia grafov dokážeme jednu z jej vlastností. Farbenie grafov bude prebiehať netradičným spôsobom – formou hry Alice a Boba. Alicinou úlohou bude generovať hrany grafu a Bob ako jej protivník bude mať za úlohu ich farbiť. Alica bude generovaním hrán vytvárať veľmi špeciálny graf tak, aby hrany spĺňali konkrétnu požiadavku týkajúcu sa informačnej vzdialenosti. V závere dôkazu získame algoritmus na farbenie Alicou generovaných grafov a vďaka dĺžke informácie potrebnej na korektné farbenie dokážeme požadované tvrdenie.

7.1 Hranové farbenie grafov a informačná vzdialenosť

Definícia 7.1.1. *Informačná vzdialenosť* reťazcov x a y pre algoritmy A a B, označujeme ju$C_{AB}(x \leftrightarrow y)$, je definovaná ako

$$C_{AB}(x \leftrightarrow y) = \min\{|p| : A(p, y) = x, B(p, x) = y\} \tag{7.1}$$

Dolný odhad pre hodnotu informačnej vzdialenosti je zrejme

$\max\{C(x|y), C(y|x)\}$. Totiž ak $A(p, y) = x$, tak zrejme $C(x|y) \leq |p| + \mathrm{O}(1)$. Obdobne pre algoritmus B. Ukážeme si teraz ďalšiu vlastnosť takto definovanej informačnej vzdialenosti.

Veta 7.1.2. *Pre všetky binárne reťazce x a y platí*

$$C_{AB}(x \leftrightarrow y) = \max\{C(x|y), C(y|x)\} + \mathrm{O}(\log \max\{\mathrm{C(x|y)}, \mathrm{C(y|x)}\}). \quad (7.2)$$

Ako sme už spomínali, tvrdenie dokážeme trochu netradičným, no o to zaujímavejším postupom, a síce pomocou farbenia grafov. Dôkaz v originálnom znení, z ktorého sme čerpali, nájdete v [14].

Dôkaz. Nech G je neorientovaný graf. Graf G je správne hranovo ofarbený, ak pre každú dvojicu hrán platí, že ak majú spoločný vrchol, majú aj rôznu farbu. Stupeň grafu definujeme ako maximálny stupeň vrchola v grafe a budeme ho označovať $d(G)$. Zrejme na zafarbenie grafu G potrebujeme minimálne $d(G)$ rôznych farieb. Keď je totiž v grafe $d(G)$ hrán incidentných s nejakým vrcholom g, každá z $d(G)$ hrán musí mať inú farbu ako všetky ostatné hrany incidentné s týmto vrcholom. Podľa Vizingovej vety[1] platí, že každý graf vieme zafarbiť $d(G)+1$ farbami. Situácia sa ale zmení, keď pri farbení nebudeme vopred poznať celý graf, ale bude nám odhaľovaný postupne - po jednotlivých hranách a my budeme musieť každú novú hranu zafarbiť „online“. V takom prípade už $d(G) + 1$ farieb stačiť nemusí.

Pri dôkaze nám pomôže hra spočívajúca v generovaní a farbení grafu. Hrajú ju proti sebe Alica a Bob. Alica generuje hrany grafu tak, aby neprekročila daný stupeň grafu d. Ak ho prekročí, prehráva. Jej protivník Bob tieto hrany farbí. Ak Alica vygeneruje hranu, Bobovou úlohou je zafarbiť ju jednou z l farieb, ktoré má k dispozícii. V prípade, že počet použitých farieb prekročí l, prehráva. Naopak, vyhrá, ak sa mu celý graf podarí ofarbiť najviac l farbami.

Pozrime sa najskôr na hornú hranicu počtu farieb, ktoré bude Bob potrebovať. Stupeň grafu G je najviac d, preto existuje aspoň jeden vrchol

[1] viď. [4],[17]

incidentný s d hranami. V najhoršom prípade môžu byť hranou spojené dva vrcholy stupňa d. Nech sú to vrcholy a a b a nech sú spojené hranou e. Z vrchola a vychádza okrem hrany e ďalších $d-1$ hrán, tak isto z vrchola b vychádza okrem hrany e ďalších $d-1$ hrán. Ľubovoľná hrana grafu G preto susedí najviac s $2(d-1)$ ďalšími hranami. Keď zarátame jednu farbu potrebnú na zafarbenie e, $2(d-1)+1 = 2d-1$ farieb určite stačí na ofarbenie grafu G. Navyše pre Boba existuje víťazná stratégia využívajúca maximálne tento počet farieb (vezmime si jednoduchý greedy algoritmus, ktorý zafarbí hranou najmenšou možnou farbou, resp. farbou s najmenšou hodnotou).

Nech $k = \max\{C(x|y), C(y|x)\}$. Graf G, ktorého množinu vrcholov tvoria binárne reťazce a dva vrcholy (reťazce) x a y sú spojené hranou práve vtedy, keď platí $(C(x|y) \leq k) \wedge (C(y|x) \leq k)$, označujeme G_k. Počet všetkých binárnych reťazcov dĺžky menšej alebo rovnej k je $\sum_{i=1}^{k} = 2^{k+1} - 1$, preto aj hrán v grafe G_k je najviac $2^{k+1} - 1$. Stupeň grafu G_k je preto tiež najviac $2^{k+1} - 1$.

Alica generujúc hrany vykoná svoj ťah (x, y), keď nájde novú hranu, resp. dvojicu vrcholov x a y takú, že $(C(x|y) \leq k) \wedge (C(y|x) \leq k)$. Bob vykonáva svoj greedy algoritmus a túto hranu zafarbí.

Celý tento postup popisuje algoritmus (označme ho T) na farbenie grafu G_k používajúc najviac $2d - 1 \leq 2(2^{k+1} - 1) - 1 = 2^{k+2} - 3$ farieb. Vstupom pre T je hodnota k a na výstupe odovzdá všetky dvojice ⟨hrana grafu G_k, jej farba⟩.

Uvažujme algoritmus A, ktorý pozná y a so vstupom $\overline{k}i$ simuluje $T(k)$ a vypíše x práve vtedy, keď $T(k)$ vypíše dvojicu $\langle (x, y), i \rangle$. Keďže algoritmus T nájde práve také dvojice (x, y), že $(C(x|y) \leq k) \wedge (C(y|x) \leq k)$, hrana (x, y) patrí do grafu G_k. Môžeme písať $A(\overline{k}i, y) = x$, analogicky $A(\overline{k}i, x) = y$, keď A pozná x namiesto y.

Čo vieme povedať o hodnote $C_{AB}(x \leftrightarrow y)$? Nech algoritmus B je identický

s algoritmom A. Potom

$$C_{AA}(x \leftrightarrow y) \leq |\overline{k}i| = 2\log k + 1 + k = k + \mathrm{O}(\log \mathrm{k})$$

Algoritmus T potrebuje najviac $2^{k+2} - 3$ farieb, na špecifikáciu jednej z nich preto stačí rádovo k bitov.

$$C_{AA}(x \leftrightarrow y) \leq |\overline{k}i| = 2\log k + 1 + k = k + \mathrm{O}(\log \mathrm{k}) \tag{7.3}$$

Potom už ľahko vidieť, že

$$C(x \leftrightarrow y) \leq C_{AA}(x \leftrightarrow y) + \mathrm{O}(1) \leq \mathrm{k} + \mathrm{O}(\log \mathrm{k}) \tag{7.4}$$

Keďže sme zvolili $k = \max\{C(x|y), C(y|x)\}$, dostávame

$$C_{AB}(x \leftrightarrow y) = \max\{C(x|y), C(y|x)\} + \mathrm{O}(\log \max\{\mathrm{C(x|y)}, \mathrm{C(y|x)}\}).$$

□

7.2 Dolný odhad zložitosti Shellsortu v priemernom prípade

Ďalším zaujímavým využitím Kolmogorovskej zložitosti bude odvodenie dolného odhadu zložitosti triediaceho algoritmu *Shellsort*[2] v priemernom prípade. Určenie netriviálneho dolného odhadu zložitosti tohto triedenia v priemernom prípade bolo otvoreným problémom vyše štyri desaťročia. Túto úlohu sa podarilo vyriešiť práve pomocou Kolmogorovskej zložitosti a metódy nestlačiteľnosti. Dôkaz, ktorý si predvedieme, môžete nájsť v [15, 2]. Hlavným zdrojom pri písaní dôkazu bola bakalárska práca [8], v ktorej sme tento odhad tiež ukázali. Dôkaz bez použitia metódy nestlačiteľnosti nájdete v [11].

Shellsort triedi n-prvkovú postupnosť π v p prechodoch využívajúc postupnosť *inkrementov* $h_1, h_2, \ldots, h_p$. V k-tom prechode rozdelí postupnosť na h_k

[2]Donald L. Shell, [12] (1959)

podpostupností dĺžky $\lceil n/h_k \rceil$ (resp. $\lfloor n/h_k \rfloor$). V každej podpostupnosti π_{i,h_k} pre $i \in \{1, 2, \ldots, h_k\}$ sú prvky s indexami $i, (i + h_k), (i + 2h_k), \ldots, (i + \lceil n/h_k \rceil h_k)$ (posledný člen podpostupnosti môže mať index $i + \lfloor n/h_k \rfloor h_k$). Index prvku je z množiny $\{1, 2, \ldots, h_k\}$. Každá z týchto podpostupností je utriedená jednoduchým *Insertsortom*. Efektívnosť triedenia závisí od voľby p a jednotlivých inkrementov $h_1, \ldots, h_p$, pričom $h_p = 1$, aby bolo zaručené úplné utriedenie prvkov (inak by postupnosť mohla ostať len čiastočne utriedená).

Na ilustráciu si uveďme príklad. Nech sme na vstupe dostali postupnosť $5, 8, 3, 1, 9, 2, 10, 7, 4, 6$. Nech $h_1 = 3$, postupnosť je rozdelená na tri podpostupnosti dĺžok 4, 3 a 3 nasledovne

$$\begin{array}{ccc} 5 & 8 & 3 \\ 1 & 9 & 2 \\ 10 & 7 & 4 \\ 6 & & \end{array}$$

pričom stĺpce tvoria vzniknuté podpostupnosti. Tieto podpstupnosti (stĺpce) sú následne utriedené Insertsortom

$$\begin{array}{ccc} 1 & 7 & 2 \\ 5 & 8 & 3 \\ 6 & 9 & 4 \\ 10 & & \end{array}$$

čím dostávame čiastočne utriedenú postupnosť $1, 7, 2, 5, 8, 3, 6, 9, 4, 10$.
Ďalej by sme postupovali pre inkrement $h_2, h_3, \ldots$ Všimnime si, že keďže $h_p = 1$, podpostupnosť, ktorá vznikne v poslednom, p-tom kroku, bude celá čiastočne utriedená postupnosť s n prvkami (dostaneme jeden stĺpec). Shellsort na záver teda len Insertsortom utriedi celú čiastočne utriedenú postupnosť. Uvedomme si, že v tomto kroku je postupnosť už skoro utriedená, záverečný Insertsort bude mať preto len malú zložitosť.

Veta 7.2.1. *Priemerný počet porovnaní aj inverzií v p-prechodovom Shellsorte, kde* $p = \mathrm{o}(\log \mathrm{n})$*, na n-prvkovej postupnosti je aspoň* $\Omega(pn^{1+1/p})$ *pre ľubovoľnú postupnosť inkrementov. Priemer sa berie zo všetkých postupností n prvkov s rovnakou pravdepodobnosťou (rovnomerné rozdelenie).*

Dôkaz. V dôkaze sa najskôr dopracujeme k odhadu počtu vykonaných inverzií (vzájomných výmien prvkov). Z nich budeme schopní zrekonštruovať pôvodnú postupnosť. Vstupnú permutáciu si zvolíme nestlačiteľnú, čo v kombinácii s množstvom informácie potrebnej na kódovanie triedenej postupnosti využijeme na získanie požadovaného dolného odhadu.

Sila metódy nestlačiteľnosti, ktorú využijeme, spočíva v tom, že namiesto počítania priemeru cez všetky vstupné permutácie, argumentáciou na jednej nestlačiteľnej vstupnej permutácii dokážeme dolný odhad v priemernom prípade. Využívame pri tom fakt, že skoro všetky vstupné permutácie sú nestlačiteľné a dokázané tvrdenie bude platiť pre všetky z nich. Dokázaný výsledok bude preto platiť v priemernom prípade. Ďalšou výhodou použitej metódy je, že zvolenú nestlačiteľnú permutáciu nemusíme hľadať, či presne špecifikovať. Veta o nestlačiteľnosti garantuje jej existenciu a to nám postačuje na to, aby sme ju v dôkaze mohli použiť.

Nech vstupná postupnosť π je permutácia prvkov $\{1, 2, \ldots, n\}$. Nech A je algoritmus implementujúci $(h_1, h_2, \ldots, h_p)$ Shellsort, pričom h_k je inkrement v k-tom prechode a $h_p = 1$. Triviálnym odhadom zložitosti A zdola je $\Omega(pn)$, keďže každý z n prvkov musí byť porovnaný aspoň raz v každom z p prechodov.

Nech $m_{i,k}$ pre každé $1 \leq i \leq n$ a $1 \leq k \leq p$ je počet takých prvkov podpostupnosti obsahujúcej i na začiatku k-teho prechodu, ktoré sú vľavo od prvku i a sú od neho väčšie. Všimnime si potom, že Insertsort v prechode k potrebuje práve $\sum_{i=1}^{n}(m_{i,k} + 1)$ porovnaní.

Označme počet všetkých inverzií

$$M = \sum_{k=1}^{p} \sum_{i=1}^{n} m_{i,k} \tag{7.5}$$

Postupnosť $m_{1,k}, \ldots m_{n,k}$ jednoznačne popisuje počiatočnú permutáciu prechodu k. Preto ak poznáme $m_{i,k}$ pre všetky $1 \leq i \leq n$ a $1 \leq k \leq p$ (spolu ich je np), dokážeme zrekonštruovať počiatočnú permutáciu π.

Dokážeme odhadnúť koľko existuje rôznych postupností $m_{i,k}$ pre $1 \leq i \leq n$ a $1 \leq k \leq p$?

Pri odhade nám pomôže celkový počet inverzií. Zistiť počet rôznych postupností $m_{i,k}$ (pre $1 \leq i \leq n$ a $1 \leq k \leq p$) znamená zistiť koľko existuje rozdkladov čísla M na np nezáporných sčítancov. Táto úloha je ekvivalentná úlohe zistiť koľkými spôsobmi môžeme postupnosť M prvkov rozdeliť na np podpostupností. Keďže $m_{i,k}$ môže byť rovné nule, niektoré z np podpostupností môžu byť prázdne. Všetkých np podpostupností M-prvkovek postupnosti môžeme oddeliť pomocou $np - 1$ oddeľovačov. Spolu tak budeme postupnosť mať $M+np-1$ objektov. Počet rôznych výberov $np-1$ oddeľovačov spomedzi $M + np - 1$ objektov vieme vypočítať:

$$D(M) = \binom{M + np - 1}{np - 1} \tag{7.6}$$

Rozdelenia M-prvkových postupností na np podpostupností pomocou $np-1$ oddeľovačov vieme ľahko generovať. (Ako?) Každé rozdelenie potom môžeme popísať jeho poradovým číslom $j \leq \log D(M)$ vo vymenovaní týchto rozdelení. Čísla M a j preto jednoznačne popisujú postupnosť prvkov $m_{i,k}$.

Spomeňme si, že existuje $n!$ rôznych permutácií n prvkov. Nech pre vstupnú permutáciu π platí

$$C(\pi|n, A, P) \geq \log n! - \log n, \tag{7.7}$$

kde P je program, ktorý z M a j rekonštruuje zoznam prvkov $m_{i,k}$. Veta 4.1.5 zaručuje existenciu takej postupnosti. Vieme, že existuje algoritmus, ktorý dokáže z popisu postupnosti $m_{i,k}$ zrekonštruovať pôvodnú postupnosť π.[3] Dĺžka zápisu tohto algoritmu (O(1) bitov) spolu s dĺžkou popisu postupnosti

[3]Takýto algoritmus budete hľadať pri riešení úlohy 7.2.1.

$m_{i,k}$ musia prevyšovať Kolmogorovskú zložitosť π:

$$C(m_{1,1},\ldots,m_{n,p}|n,A,P)+\mathrm{O}(1)\geq \mathrm{C}(\pi|\mathrm{n},\mathrm{A},\mathrm{P})\geq \log \mathrm{n}!-\log \mathrm{n}$$

Dĺžka popisu M a j, z ktorých program P dokáže rekonštruovať pôvodnú postupnosť $m_{i,k}$, musí byť aspoň $C(m_{1,1},\ldots,m_{n,p}|n,A,P)$. Po zapísaní M a j v samooddeľujúcom tvare dostávame

$$2\log\log M+\log M+\log D(M)+\mathrm{O}(1)\geq \log \mathrm{n}!-\log \mathrm{n}-\mathrm{O}(1).$$

Keďže každé $m_{i,k}\leq n$, zrejme $M\leq pn^2$. Predpokladajme, že $p<n$ (inak by $M\leq n^3$).
Uvedený výraz upravíme:

$$2\log\log pn^2+\log pn^2+\log D(M)\geq \log n!-\log n-\mathrm{O}(1)$$

Použijeme $pn^2<n^3$.

$$\begin{aligned} 2\log 3\log n+3\log n+\log D(M) &\geq \log n!-\log n-\mathrm{O}(1)\\ 2\log 3+2\log\log n+3\log n+\log D(M) &\geq \log n!-\log n-\mathrm{O}(1)\end{aligned}$$

$$\log D(M)\geq \log n!-4\log n-2\log\log n-\mathrm{O}(1) \tag{7.8}$$

Na úpravu $\log D(M)$ využijeme rovnosť[4]

$$\log\binom{a}{b}=b\log\frac{a}{b}+(a-b)\log\frac{a}{a-b}+\frac{1}{2}\log\frac{a}{b(a-b)}+\mathrm{O}(1) \tag{7.9}$$

Po úprave:

$$\begin{aligned}\log\binom{M+np-1}{np-1} &= (np-1)\log\frac{M+np-1}{np-1}+M\log\frac{M+np-1}{M}+\\ &\quad \frac{1}{2}\log\frac{M+np-1}{M(np-1)}+\mathrm{O}(1)\end{aligned}$$

Druhý sčítanec na pravej strane rovnice môžeme odhadnúť podľa

$$e^a>(1+\frac{a}{b})^b. \tag{7.10}$$

[4]Túto rovnosť môžete nájsť v [16]. Uvádzame ju bez dôkazu, kvôli jeho náročnosti a dĺžke.

Po úprave:

$$\log\left(1+\frac{np-1}{M}\right)^{M} < \log e^{np-1} \tag{7.11}$$

pre kladné M a $np-1>0$. Ďalej dostávame

$$\log\binom{M+np-1}{np-1} =$$
$$(np-1)\left(\log\left(1+\frac{M}{np-1}\right)+\log e+\frac{1}{2(np-1)}\log\frac{M+np-1}{M(np-1)}\right)$$

Pre $n\to\infty$

$$\frac{1}{2(np-1)}\log\frac{M+np-1}{M(np-1)}\to 0$$

Potom $\log D(M)$ pre $n\to\infty$ môžeme vyjadriť ako

$$\log D(M)=(np-1)\left(\log\left(1+\frac{M}{np-1}\right)+\log e\right)$$

Na $\log n!$ môžeme aplikovať Stirlingovu aproximáciu

$$n!\approx\frac{n^n}{e^n}\sqrt{2\pi n}. \tag{7.12}$$

Pravá strana nerovnosti (7.8) potom pre $n\to\infty$ konverguje k $n\log n$. Môžeme písať

$$(np-1)\left(\log\left(1+\frac{M}{np-1}\right)+\log e\right)\geq n\log n.$$

Postupne upravujeme pre $p=\mathrm{o(log\,n)}$.

$$\begin{aligned} np\log\left(1+\frac{M}{np-1}\right)+np\log e &\geq n\log n\\ p\log\left(1+\frac{M}{np-1}\right) &\geq \log n\\ 1+\frac{M}{np-1} &\geq n^{1/p}\\ M &\geq pn^{1+1/p} \end{aligned}$$

Dostávame

$$M = \Omega(pn^{1+1/p}) \tag{7.13}$$

Získali sme dolný odhad na počet inverzií pre permutáciu π, pre ktorú platí $C(\pi \mid n, A, P) \geq \log n! - \log n$. Takýchto permutácií je aspoň $n!(1 - 1/n)$, získaný odhad platí pre každú z nich, preto priemerná zložitosť je aspoň

$$\left(1 - \frac{1}{n}\right) \Omega(pn^{1+1/p}) = \Omega(pn^{1+1/p}), \tag{7.14}$$

pre $p = \mathrm{o}(\log \mathrm{n})$. Pre $p = \Omega(\log n)$ je dolný odhad triviálne $pn = \Omega(pn^{1+1/p})$.

Vďaka tomu, že skoro všetky permutácie sú nestlačiteľné, sme získaním dolného odhadu pre jednu nestlačiteľnú permutáciu dokázali dolný odhad platný v priemere cez všetky permutácie. □

Úloha 7.2.1. Zvoľte si n, postupnosť $m_{i.k}$ ($1 \leq i \leq n$ a $1 \leq k \leq p$, kde $m_{i,k}$ je počet takých prvkov podpostupnosti obsahujúcej i na začiatku k-teho prechodu, ktoré sú vľavo od prvku i a sú od neho väčšie) pre nejaké k a pokúste sa z postupnosti $m_{i,k}$ zrekonštruovať postupnosť na začiatku prechodu Shellsortu.

Úloha 7.2.2. Na základe dôkazu vety 7.2.1 sa pokúste dokázať, že Bubblesort vykoná v priemernom prípade aspoň $\Omega(n^2)$ výmien prvkov.

Kapitola 8

Záver

Cieľom práce bolo predstaviť Kolmogorovskú zložitosť a predviesť jej vlastnosti a aplikácie na rozmanitých príkladoch. U čitateľa sme sa snažili vzbudiť záujem o túto zaujímavú, aj keď miestami náročnú oblasť teoretickej informatiky. Prácu sme písali ako študijný materiál, môže poslúžiť aj ako doplnkový text k predmetu Kolmogorovská zložitosť vyučovaného na našej fakulte.

Vysvetlili a popísali Kolmogorovskú zložitosť, ukázali jej základné atribúty a využitie na množstve riešených i neriešených príkladoch. Postupne sme prebrali viacero aspektov Kolmogorovskej zložitosti. V každej kapitole sme najskôr uviedli podstatné definície, spomenuli a dokázali fundamentálne vlastnosti, za ktorými nasledovala časť venovaná praktickým príkladom a úlohám.

V úvodnej časti práce sme uviedli teoretické základy vyžadované v ďalších častiach práce. Zaviedli sme pojem Kolmogorovská zložitosť reťazca ako špeciálny prípad popisnej zložitosti. Od Kolmogorovskej zložitosti vzhľadom k určitej popisnej metóde sme prešli k definícii Kolmogorovskej zložitosti bez závislosti na konkrétnej popisnej metóde. Definovali sme podmienenú aj nepodmienenú Kolmogorovskú zložitosť. Dopracovali sme sa k pojmu nestlačiteľnosť, ktorý sme formálne definovali a dokázali sme, že skoro všetky reťazce sú nestlačiteľné. Objasnili sme princípy metódy nestlačiteľnosti. Vysvetlili sme prepojenie medzi nestlačiteľnosťou a náhodnosťou reťazca, defino-

vali sme testy náhodnosti. Na záver sme predviedli veľmi inšpiratívne využitia Kolmogorovskej zložitosti a metódy nestlačiteľnosti, ktoré demonštrujú, že teória Kolmogorovskej zložitosti je v praxi naozaj využiteľná a využívaná.

Práca poskytuje možnosti na jej rozšírenie. V budúcnosti je možné obohatiť jednotlivé kapitoly, možno ukázať ďalšie vlastnosti preberaných pojmov, zaviesť nové pojmy a uviesť nové príklady a úlohy na precvičenie. Taktiež možno pridať nové kapitoly zamerané na komplexnejšie stránky Kolmogorovskej zložitosti ako sú napr. prefixová Kolmogorovská zložitosť či Kolmogorovská zložitosť s obmedzenými prostriedkami.

Dúfame, že toto dielo ako aj Kolmogorovská zložitosť samotná čitateľov zaujali a veríme, že mnohým študentom aj informatickým nadšencom pomôže pri zoznamovaní sa s Kolmogorovskou zložitosťou a podnieti ich k jej hlbšiemu štúdiu. Budeme radi, ak táto práca pomôže spopularizovať Kolmogorovskú zložitosť medzi študentmi i odbornou verejnosťou.

Zoznam obrázkov

Literatúra

[1] David Bailey, Peter Borwein, and Simon Plouffe. On the rapid computation of various polylogarithmic constants. *Math. Comput.*, 66(218):903–913, 1997.

[2] Bronislava Brejová. Analyzing variants of shellsort. *Information Processing Letters*, (79):223–227, 2000.

[3] Harry Buhrman, Tao Jiang, Ming Li, and Paul Vitányi. New applications of the incompressibility method: part ii. *Theor. Comput. Sci.*, 235(1):59–70, 2000.

[4] S. Fiorini and R. J. Wilson. Edge-colorings of graphs. Pitman, 1977.

[5] Lance Fortnow. Kolmogorov complexity. lecture notes, 2000.

[6] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, 1950.

[7] Lance Fortnow Sophie Laplante Harry Buhrman. Resource-bounded kolmogorov complexity revisited, 2001.

[8] Peter Lenčéš. Kolmogorovská zložitosť. bakalárska práca, jún 2008.

[9] David J. C. Mackay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, 1st edition, June 2002.

[10] Todd K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, June 2005.

[11] Tao Jiang Ming Li Paul Vitányi. A lower bound on the average-case complexity of shellsort. *Journal of the ACM*, 47(5):905–911, September 2000.

[12] D. L. Shell. A high-speed sorting procedure. *Commun. ACM*, 2(7):30–32, 1959.

[13] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, 2(42):230–265, 1936.

[14] Nikolay Vereshchagin. Kolmogorov complexity and games, 2008.

[15] Paul Vitányi. Analysis of sorting algorithms by kolmogorov complexity (a survey). *Entropy, Search, Complexity*, (16):209–232, 2000.

[16] Paul Vitányi and Ming Li. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, third edition, 2008.

[17] R. J. Wilson. Graphs, colorings and the four-color theorem. Oxford University Press, 2002.