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

Kolmogorovská zložitosť

(Bakalárska práca)

Peter Lenčéš

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

Bratislava, 2008

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

.....

Ďakujem svojej školiteľke RNDr. Dane Pardubskej, PhD. za zaujímavú tému, študijné materiály, dohodnutý prístup a príjemné konzultácie, na ktorých mi vždy poskytla odbornú pomoc. Ďalej ďakujem svojim rodičom za výraznú podporu pri štúdiu a za všetko ostatné a svojmu bratovi za vzor.

Abstrakt

Kolmogorovská zložitosť sa ukazuje ako jednoduchý, intuitívny a veľmi silný nástroj na dokazovanie tvrdení z mnohých oblastí matematiky a informatiky.

Práca uvádza niektoré základy a terminológiu potrebnú na pochopenie problematiky. Vysvetľuje Kolmogorovskú zložitosť ako takú, čím buduje aparát na zavedenie pojmu nestlačiteľnosti čo vyústi do tzv. metódy nestlačiteľnosti, ktorej výklad a ukážky využitia tvoria nosnú časť práce. Záverečná kapitola poskytuje podrobné dôkazy mnohých tvrdení, ktoré demonštrujú niektoré jej dôležité výsledky. Dôkazy tu uvedené pokrývajú oblasť formálnych jazykov a automatov, zložitosti algoritmov a náhodných grafov.

Kľúčové slová: Kolmogorovská zložitosť, metóda nestlačiteľnosti, formálne jazyky, triediace algoritmy, grafy

Abstract

Kolmogorov Complexity shows to be a simple, intuitive and very powerful tool for proving theorems from a wide range of subfields of mathematics and informatics.

The thesis gives some basics and terminology essential for understanding the topic. It explains Kolmogorov Complexity as it is, by which it builds an apparatus for establishing the notion of incompressibility. This leads to so called incompressibility method, representation of which makes the carrier part of the thesis. The final chapter provides detailed proofs of plenty of theorems, which demonstrate some if its important results. Proofs mentioned cover field of formal languages and automatas, algorithm complexity and random graphs.

Keywords: Kolmogorov Complexity, incompressibility method, formal languages, sorting algorithms, graphs

Obsah

Kapitola 1

Úvod

Motiváciou pre vznik tohto textu bola o.i. neexistencia kvalitnej literatúry v slovenskom jazyku zameranej na Kolmogorovskú zložitosť. Cieľom práce preto bolo napísať študijný text v slovenskom jazyku formulovaný tak, aby bol zrozumiteľný pre čitateľa so základnými vedomosťami z teoretickej informatiky, avšak aby tým neutrpela exaktnosť a potrebný stupeň formálnosti textu. Práca poslúži ako odrazový mostík pri hlbšom štúdiu Kolmogorovskej zložitosti a poskytuje úvodný náhľad ilustrovaný mnohými príkladmi jej využitia.

S rozvojom teoretickej informatiky sa informatici začali viac zaujímať o zložitosť algoritmov. Vznikla snaha vytvárať čo najlepšie, najefektívnejšie algoritmy či už z pohľadu časovej alebo priestorovej zložitosti. Všeobecne však môže byť zaujímavá zložitosť rôznych objektov, nie len algoritmov. Pomocou zložitosti týchto objektov môžeme tiež určiť zložitosti algoritmov, ktoré s týmito objektami pracujú.

Takýto prístup využíva napríklad Kolmogorovská zložitosť (KZ). Je to špeciálny prípad algoritmickej zložitosti, ktorá sa implicitne nezaoberá zložitosťou algoritmov, ale zložitosťou reťazcov. KZ reťazca je meraná dĺžkou algoritmu, ktorý tento reťazec produkuje. Zjednodušene môžeme povedať, že vzhľadom na vybranú popisnú metódu, je Kolmogorovská zložitosť reťazca rovná dĺžke najkratšieho popisu reťazca. Na základe KZ reťazca potom môžeme dokázať aj zložitosť samotného algoritmu ako popisu reťazca. Takýto spôsob uvažovania sa v mnohých prípadoch ukazuje ako najvhodnejšia voľba. Pomocou tejto idey sa podarilo vyriešiť mnoho problémov, ktoré sa klasickým prístupom nedarilo riešiť desiatky rokov.

Na ilustráciu je pre nás veľmi pekným príkladom číslo $\pi = 3.14159265\ldots$ Vieme, že toto číslo tvorí nekonečne dlhý reťazec, lebo má nekonečný desatinný rozvoj. π sa preto intuitívne zdá byť veľmi zložitý reťazec. Zdá sa, že jednotlivé cifry jeho rozvoja sa striedajú vskutku náhodne. Avšak hodnotu π vieme s určitou presnosťou jednoducho vypočítať, jedným z jeho vyjadrení je napr. Bailey-Borwein-Plouffeov vzorec

$$\pi = \sum_{k=0}^{\infty} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \tag{1.1}$$

Vidíme, že nekonečný reťazec sme práve popísali veľmi krátkym popisom. Ak určujeme Kolmogorovskú zložitosť reťazca, kritériom zložitosti je pre nás dĺžka jeho najkratšieho popisu. Vidíme, že π má z tohto hľadiska prekvapivo malú zložitosť. Existuje preň krátky algoritmus, a keďže KZ reťazca je dĺžka jeho najkratšieho popisu, jeho zložitosť je maximálne dĺžka uvedeného popisu, ktorá je rovná konštante.

KZ je aplikovateľná v mnohých oblastiach matematiky a informatiky. Hoci nie každá z týchto oblastí pracuje s reťazcami, prakticky všetky objekty sme schopní kódovať ako binárne reťazce. Reťazec je teda jeden z najpoužívanejších pojmov v tejto práci. Prvá sekcia bude preto pojednávať práve o reťazcoch a spôsobe ich kódovania.

1.1 Reťazce a ich kódovanie

Tak ako vieme vyjadrovať čísla v binárnej sústave, vieme aj každý reťazec kódovať ako sekvenciu núl a jednotiek, tzn. zapísať ho v abecede $\{0, 1\}$. Tým dostaneme binárny reťazec, s ktorým sa nám pracuje lepšie, lebo v abecede máme len dva znaky. Preto budeme predpokladať, že každý reťazec je nad abecedou $\{0, 1\}$ (keby aj pôvodne nebol, vieme ho tak zakódovať). Potom dĺžku reťazca x vieme vyjadriť ako $\log x$. Z tohto dôvodu budeme používať výhradne logaritmus so základom 2. Zápis $\log x$ preto znamená $\log_2 x$. Binárne reťazce (sekvencie, slová) je vhodné mať nejak zoradené, aby sme mohli niektorý z nich zvoliť bez toho, aby sme ho museli celý vypísať. Reťazce utriedime *lexikograficky*, tzn. najskôr podľa ich dĺžky a potom podľa 'numerickej hodnoty'. Dostávame takúto lexikograficky utriedenú postupnosť reťazcov:

$$\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \ldots \tag{1.2}$$

Stotožňujeme množinu prirodzených čísiel s uvedenou lexikograficky zoradenou postupnosťou binárnych reťazcov nasledovne:

$$(\epsilon, 0), (0, 1), (1, 2), (00, 3), (01, 4), (10, 5), (11, 6), (000, 7), \ldots \tag{1.3}$$

Takýmto spôsobom môžeme číslo $x = 2^{n+1} - 1 + \sum_{i=0}^{n} a_i 2^i$ reprezentovať reťazcom $a_n a_{n-1} \ldots a_1 a_0$. Taktiež môžeme písať $x = \sum_{i=0}^{n} (a_i + 1)2^i$ pre $0 \leq i \leq n$.

Takže binárny reťazec '10' a prirodzené číslo v decimálnom tvare 5 budú znamenať ten istý objekt. V skutočnosti môžeme napr. písať $001 = 8$. Dĺžkou konečného binárneho reťazca x rozumieme počet jeho bitov. Budeme ju označovať $l(x)$. Napríklad $l(000) = 3$ alebo pre $x = x_1 x_2 \ldots x_n$ platí $l(x) = n$, pre prázdne slovo ϵ zrejme platí $l(\epsilon) = 0$.

Pri kódovaní viacerých objektov do reťazca môže nastať problém, keď nevieme jednoznačne určiť, ktorá časť kódu reprezentuje ktorý objekt. Preto ak chceme jedným reťazcom popísať viacej objektov, musíme nájsť také kódovanie, aby sme jednotlivé popisy objektov vedeli oddeliť. Na to využívame tzv. *prefixové kódy*.

Hovoríme, že x je prefixom y, ak existuje z také, že $y = xz$. Množine slov hovoríme *bezprefixová*, ak žiadne z jej slov nie je prefixom žiadneho iného slova z tejto množiny. Dekódovacia funkcia $D : A \to B$ definuje *prefixový kód*, ak jej definičný obor A je bezprefixový. Bezprefixová množina a prefixový kód používame ako synonymá. A tvorí množinu tzv. *kódových slov*. V prefixovom kóde teda žiadne z kódových slov nie je prefixom žiadneho iného kódového slova. Ak máme objekty kódované prefixovým kódom ako binárny reťazec, vieme tento reťazec rozdeliť na časti reprezentujúce jednotlivé objekty a vieme ho jednoznačne dekódovať.

Vezmime si napríklad kódovaciu funkciu $E : \{x, y, z\} \to \{0, 1\}$ definovanú takto: $(x, 0), (y, 01), (z, 001)$. Vidíme, že x je prefixom slova y ($y = x1$) ako aj slova z ($z = xy$), preto by sme mohli kódové slovo 0001 dekódovať ako xxy alebo aj ako xz. Toto teda nie je prefixový kód.
Definujme preto kódovaciu funkciu E nasledovne:

Definícia 1.1.1. $E : \{0, 1\}^* \to \{0, 1\}^*$

$$E_0(x) = 1^x 0$$

$$E_i(x) = E_{i-1}(l(x))x \text{ pre } i > 0$$

Pozrime sa bližšie čo vlastne robí funkcia E. Vezmime si napr. $x = 10$. V lexikografickom usporiadaní tento binárny reťazec označuje prirodzené číslo 5. Prípad pre $i = 0$ je zrejme $1^5 0 = 111110$. Ďalej $E_1(10)$ sa podľa predpisu rovná $E_0(l(10))10 = 11010$. Keďže $l(10) = 2$, prvé dve jednotky označujú dĺžku zápisu x, teda koľko bitov treba na jeho zápis, potom nasleduje deliaca 0 a potom zápis samotného x.
Potom

$$E_2(x) = E_1(l(x))x = E_0(l(l(x)))l(x)x = 1^{l(l(x))}0l(x)x.$$

Pre $x =$'10' platí $E_2(x) = 10110$, lebo $l(x) = 2$ a nezabúdajme, že prirodzené číslo 2 sme stotožnili s binárnym kódom '1' (dôležitosť tohto by sme si viac uvedomili napr. v prípade, že dĺžka x by bola 3, pričom 3 je pre nás vlastne '00'). Takže $l(l(x)) = 1$, teda dĺžka zápisu '1' je 1 $\Rightarrow$ preto začína $E_2(x)$ práve jednou jednotkou, nasleduje oddeľujúca 0, potom dĺžka zápisu x, čo je 2 ='1' a potom už len samotné $x =$'10'. Kódovanie $E_1 = 1^{l(x)}0x$ je pre nás dôležité a často ho využívame, preto preň zavedieme označenie

$$\overline{x} = 1^{l(x)}0x \tag{1.4}$$

pričom zrejme $l(\overline{x}) = 2l(x) + 1$. Niekedy budeme využívať aj

$$E_2(x) = \overline{l(x)}x, \tag{1.5}$$

kde platí $l(E_2(x)) = 2l(l(x)) + l(x) + 1$.
$\overline{x}$ nazývame 'samooddeľujúci' (z anglického *self-delimiting*) tvar reťazca x. Tento tvar budeme považovať za *štandardný* zápis reťazca.

Niekedy budeme potrebovať zapísať v samooddeľujúcom tvare viac ako len jeden reťazec. V prípade, že chceme zapísať dva reťazce x a y, využijeme zápis $\langle x, y\rangle$.

Definícia 1.1.2. $\langle x, y\rangle = \overline{x}y = 1^{l(x)}0xy$

Potom $l(\langle x, y\rangle) = 2l(x) + l(y) + 1$.
Čo v prípade, že chceme zapísať ešte viac reťazcov? Ak chceme zapísať reťazce x, y a z, použijeme $\langle x, \langle y, z\rangle\rangle$, pričom zrejme platí

$$\langle x, \langle y, z\rangle\rangle = \langle x, 1^{l(y)}0yz\rangle = 1^{l(x)}0x1^{l(y)}0yz, \tag{1.6}$$

$$resp.\ \langle x, \langle y, z\rangle\rangle = \langle x, \overline{y}z\rangle = \overline{x\overline{y}}z. \tag{1.7}$$

Potom $l(\langle x, \langle y, z\rangle\rangle) = 2l(x) + 2l(y) + l(z) + 2$.
Pre zápis viacerých reťazcov iterujeme uvedený postup. Takýto zápis ešte väčšieho počtu reťazcov ale nebude v priebehu textu potrebný.

1.2 Asymptotická notácia

Na vyjadrenie časovej zložitosti budem používať asymptotickú notáciu, preto uvediem jej presnú definíciu.

Definícia 1.2.1. Nech f a g sú funkcie na reálnych číslach. Potom

$$\mathrm{O}\left(\mathrm{g(n)}\right) = \{\mathrm{f(n)} \mid (\exists \mathrm{c} > 0)(\exists \mathrm{n}_0 > 0)(\forall \mathrm{n} > \mathrm{n}_0)\ 0 \leq \mathrm{f(n)} \leq \mathrm{cg(n)}\},$$

$$\mathrm{o}\left(\mathrm{g(n)}\right) = \{\mathrm{f(n)} \mid (\forall \mathrm{c} > 0)(\exists \mathrm{n}_0 > 0)(\forall \mathrm{n} > \mathrm{n}_0)\ 0 \leq \mathrm{f(n)} < \mathrm{cg(n)}\},$$

alternatívna definícia $f(n) = \mathrm{o}\left(\mathrm{g(n)}\right)$ ak $\lim\limits_{\mathrm{n}\to\infty} \dfrac{\mathrm{f(n)}}{\mathrm{g(n)}} = 0,$

$$\Omega\left(g(n)\right) = \{f(n) \mid (\exists c > 0)(\exists n_0 > 0)(\forall n > n_0)\ 0 \leq cg(n) \leq f(n)\},$$

$$\omega\left(g(n)\right) = \{f(n) \mid (\forall c > 0)(\exists n_0 > 0)(\forall n > n_0)\ 0 \leq cg(n) < f(n)\},$$

alternatívna definícia $f(n) = \omega\left(g(n)\right)$ ak $\lim\limits_{n\to\infty} \dfrac{g(n)}{f(n)} = 0,$

$$\Theta\left(g(n)\right) = \{f(n) \mid f(n) = \Omega\left(g(n)\right) \wedge f(n) = \mathrm{O}\left(\mathrm{g(n)}\right)\},$$

Ak $f(n)$ patrí do $\delta(n)$, píšeme $f(n) = \delta\left(g(n)\right)$, nie $f(n) \in \delta\left(g(n)\right)$.

Kapitola 2

Kolmogorovská zložitosť

Keď už vieme ako môžeme efektívne kódovať ľubovoľné objekty do binárnych postupností, posuňme sa ďalej. Ako už bolo spomenuté, KZ reťazca je vlastne dĺžka najkratšieho programu generujúceho daný reťazec. Pri zisťovaní KZ vlastne hľadáme najkratší možný popis reťazca. Vynára sa otázka akým spôsobom, resp. v akom (programovacom) jazyku tento popis zapísať. Po krátkej úvahe zistíme, že výber jazyka citeľne neovplyvňuje dĺžku popisu. Majme popis ako program v dvoch rôznych programovacích jazykoch. Oba programy počítajú to isté. Líšia sa syntaxou, kľúčovými slovami a štruktúrou. Oba programy musia však vykonávať to isté, preto žiadny z nich nemôže byť v závislosti na dĺžke vstupu markantne dlhší. Ak jeden program vstup nejak spracováva, tak táto položka je čo do zložitosti relevantná. Lenže tak isto musí vstup spracovať aj iný program, preto sa dĺžka programov bude opäť líšiť len kvôli dĺžke kľúčových slov a syntaxe. Nech teda program napíšeme v ktoromkoľvek jazyku, jeho dĺžka sa zmení maximálne o konštantu, ktorá ako vieme pri určovaní zložitosti nehrá rolu.

Programom budeme preto rozumieť program pre *Turingov stroj*. U čitateľa sa predpokladá znalosť tohto pojmu, avšak pre istotu si k tomuto 'stroju' niečo povedzme.

Turingov stroj (TS) je teoretický model počítača schopný riešiť všetky algoritmicky riešiteľné problémy. TS pomocou hlavy, ktorá môže čítať aj zapisovať, číta vstup na potenciálne nekonečnej vstupnej páske. Páska je rozdelená na časti (políčka, bunky), pričom na každom políčku je buď jeden znak vstupnej abecedy alebo je prázdne, čo označujeme špeciálnym symbolom B (Blank). Čas je diskrétny, jednotlivé časové okamihy môžeme označiť ako $0, 1, 2, \ldots$ TS začína v čase 0, kedy každé políčko pásky obsahuje

B okrem konečnej súvislej postupnosti znakov, ktoré tvoria samotný vstup pre TS. V čase 0 je hlava umiestnená na prvom znaku vstupu. Hlava sa môže pohybovať vľavo i vpravo, v jednom kroku vždy o jedno políčko, prípadne môže ostať na práve čítanom poli. V každom kroku hlava na čítané políčko zapíše jeden znak z pracovnej abecedy. TS sa v každom okamihu nachádza práve v jednom stave z konečnej množiny všetkých svojich stavov. V čase 0 je v začiatočnom stave q_0. Funkčnosť TS je zabezpečená konečnou množinou pravidiel, podľa ktorých hlava TS pracuje. V každom kroku hlava podľa znaku, ktorý práve číta a stavu v akom sa TS nachádza, zapíše na čítané políčko zadaný znak, posunie sa vľavo, vpravo alebo ostane na mieste a TS prejde do stavu špecifikovaného daným pravidlom. TS akceptuje vstupné slovo práve vtedy, ak prejde do akceptačného stavu, nemusí ani prečítať celý vstup.

Každý TS definuje čiastočnú funkciu[1], ktorej hovoríme *čiastočne rekurzívna* alebo *vypočítateľná*. Trieda problémov riešiteľných (jazykov akceptovateľných) Turingovymi strojmi reprezentuje triedu všetkých čiastočne rekurzívnych funkcií.

Formálne je Turingov stroj definovaný ako 6-ica $(K, \Sigma, \Gamma, \delta, q_0, F)$, kde K je množina stavov T, Σ je konečná abeceda vstupných symbolov, Γ je konečná abeceda pracovných symbolov, δ je prechodová funkcia, q_0 je počiatočný stav a F je množina akceptačných stavov TS T. Turingove stroje nie sú najpraktickejšie čo sa písania programov týka, avšak veľmi dobre sa na nich dá argumentovať a dokazovať. Práve preto ich využívame.

Turingovych strojov je ale veľa a pri zadávaní programu pre nejaký TS by bolo dosť nepraktické zakaždým formálne zapísať konkrétny TS. Tento problém riešime tak, že sa zvolí ich efektívne vymenovanie

$$T_0, T_1, T_2, T_3 \ldots \tag{2.1}$$

TS sú teda zoradené a očíslované prirodzenými číslami. Potom vieme jeden konkrétny z nich špecifikovať jeho poradovým číslo. Nemusíme ho vypisovať celý, na jeho určenie nám stačí $\log i$ bitov (na zápis čísla i potrebujeme $\log i$ bitov).

Pre Turingove stroje, tak ako pre všetky objekty, existuje kódovanie do binárnych reťazcov. Kódujeme ich tak, že vieme jednoznačne určiť, či

[1] Čiastočná, lebo nie je definovaná nutne pre všetky vstupné slová, na niektorých sa proste zasekne

binárny reťazec je alebo nie je korektným zápisom nejakého TS. Na základe toho vieme konštruovať tzv. *univerzálne Turingove stroje* (UTS).

UTS U je TS, ktorý vie simulovať ľubovoľný iný TS T_i. UTS dostane na vstupe číslo i Turingovho stroja, ktorý má simulovať a program p [2] pre simulovaný TS. Vstup dostane v tvare $\langle i,p\rangle = 1^{l(i)}0ip$, čo je vlastne $\bar{i}p$. U mal simulovať i-ty Turingov stroj, preto platí

$$U(\langle i,p\rangle) = T_i(p). \tag{2.2}$$

Ešte do toho môžeme zakomponovať vstup y pre samotný T_i, potom môžeme písať

$$U(\langle y,\langle i,p\rangle\rangle) = T_i(\langle y,p\rangle). \tag{2.3}$$

Pár riadkov vyššie bolo spomenuté, že môžeme konštruovať univerzálne Turingove stroje. Tých je v skutočnosti nekonečne veľa. Ktorý z nich teda budeme používať? Dá sa ukázať, že KZ reťazcov vytvorených dvomi rôznymi univerzálnymi TS sa líšia len o konštantu, ktorá je v tomto prípade zanedbateľná. Toto si však ešte rozoberieme neskôr. Takže môžeme si zvoliť jeden (ľubovoľný) UTS U, ktorý budeme považovať za náš referenčný univerzálny počítač.

Spomínal som, že každý TS počíta nejakú čiastočne rekurzívnu funkciu ϕ. Tak isto ako Turingove stroje, vieme efektívne menovať aj tieto funkcie:

$$\phi_1, \phi_2, \phi_3, \ldots \tag{2.4}$$

kde i-ty TS počíta funkciu ϕ_i (týmto funkciám sa niekedy hovorí aj opisné metódy). Tak isto ako existuje univerzálny TS, existuje aj *univerzálna čiastočne rekurzívna funkcia*, prislúchajúca už vybranému UTS U. Z praktických dôvodov môžeme bez ujmy na všeobecnosti túto funkciu označiť ϕ_0 a TS, ktorý ju počíta T_0. Preto, ak sme písali

$$U(\langle y,\langle i,p\rangle\rangle) = T_i(\langle y,p\rangle),$$

môžeme taktiež písať

$$\phi_0(\langle y,\langle i,p\rangle\rangle) = \phi_i(\langle y,p\rangle). \tag{2.5}$$

[2] Pre tých, na ktorých tento program p pôsobí trochu mätúco (prečo by TS mal dostávať na vstupe ešte nejaký program?), tak berte TS ako počítač, na ktorom sa spustí daný program p. Tento program potom samozrejme môže mať ešte svoj vlastný vstup, ako o chvíľu uvidíme.

2.1 Podmienená Kolmogorovská zložitosť

Definícia 2.1.1. Nech x, y, p sú prirodzené čísla. Kolmogorovská zložitosť x podmienená y je definovaná ako

$$C_\phi(x|y) = min\{l(p)|\phi(\langle y, p\rangle) = x\}, \tag{2.6}$$

kde ľubovoľná čiastočne rekurzívna funkcia ϕ spolu s p a y také, že $\phi(\langle y, p\rangle) = x$ sú popisom x. Ak také p neexistuje, $C_\phi(x|y) = \infty$.

Ak teda zisťujeme Kolmogorovskú zložitosť reťazca x podmienenú y počítaného funkciou ϕ, hľadáme najmenšiu dĺžku programu p takého, že ak táto funkcia dostane na vstupe y a p, funkcia vráti práve reťazec x. V prípade, že daná funkcia s daným vstupom nikdy nevráti x, hovoríme, že má nekonečnú zložitosť C_ϕ podmienenú y. Funkcia ϕ vlastne počíta program p so vstupom y.

Veta 2.1.2. *Nech ϕ_0 je univerzálna čiastočne rekurzívna funkcia. Platí*

$$C_{\phi_0}(x|y) \leq C_\phi(x|y) + c_\phi \tag{2.7}$$

pre všetky čiastočne rekurzívne funkcie ϕ a všetky x, y, pričom c_ϕ je konštanta závisiaca od ϕ, ale nie od x alebo y.

Dôkaz. Uvedená rovnosť hovorí, že KZ reťazca získaného funkciou ϕ a funkciou ϕ_0 s tým istým vstupom sa líšia len o konštantu závisiacu len od funkcie ϕ. Funkciu ϕ_0 počíta UTS U a ten so vstupom $\langle y, \langle n, p\rangle\rangle$ simuluje T_n so vstupom $\langle y, p\rangle$. V reči funkcií to môžeme zapísať ako

$$\phi_0(\langle y, \langle n, p\rangle\rangle) = \phi_n(\langle y, p\rangle) \tag{2.8}$$

Vidíme, že univerzálna funkcia potrebuje na vstupe aj číslo n, ktoré však už funkcia ϕ_n nepotrebuje. KZ x počítaného univerzálnou funkciou preto nie je od KZ x počítaného ϕ_n väčšia o viac ako je dĺžka zápisu čísla n.

$$\langle y, \langle n, p\rangle\rangle = 1^{l(y)}0y\mathbf{1}^{\mathbf{l(n)}}\mathbf{0n}p, \tag{2.9}$$

pričom zvýraznená časť je zápis čisla n, na ktorý potrebujeme $l(n)+1+l(n)$ bitov. Preto

$$C_{\phi_0}(x|y) \leq C_{\phi_n}(x|y) + c_{\phi_n}, \tag{2.10}$$

kde $c_{\phi_n} = 2l(n) + 1$. □

Keď už vieme, že univerzálna funkcia ϕ_0 potrebuje len zanedbateľne viac informácie než simulované funkcie, môžeme pri určovaní KZ od tejto funkcie abstrahovať. Preto môžeme písať

$$C(x|y) = C_{\phi_0}(x|y) \tag{2.11}$$

pre univerzálnu čiastočne rekurzívnu funkciu ϕ_0, ktorú budeme označovať ako *referenčná funkcia* pre C.

2.2 Nepodmienená Kolmogorovská zložitosť

Po definícii podmienenej KZ uvádzame už triviálne vyzerajúcu definíciu *nepodmienenej* Kolmogorovskej zložitosti:

Definícia 2.2.1. Nepodmienená Kolmogorovská zložitosť $C(x)$ binárneho reťazca x

$$C(x) = C(x|\epsilon) \tag{2.12}$$

Ako už názov hovorí, nepodmienená KZ reťazca už nijak nezávisí od vstupu.

Skutočnosť, že prvok patrí do nejakej konkrétnej množiny, nám môže veľmi pomôcť pri odhadovaní zložitosti tohto prvku. Ukážem, že pre každú ľahko popísateľnú množinu je zložitosť ľubovoľného jej prvku rovná najviac logaritmu mohutnosti tejto množiny.

Veta 2.2.2. *Nech $A \subseteq \mathbb{N} \times \mathbb{N}$ je rekurzívne vyčísliteľná a $y \in \mathbb{N}$. Predpokladajme, že $Y = \{x : (x, y) \in A\}$ je konečná. Potom pre nejakú konštantu c závisiacu jedine od A a pre všetky $x \in Y$ platí $C(x|y) \leq l(|Y|) + c$.*

Dôkaz. Nech A je Turingovym strojom T vymenovaná bez opakovania ako postupnosť

$$(x_1, y_1), (x_2, y_2), \ldots.$$

Nech $(x_{i_1}, y_{i_1}), \ldots, (x_{i_k}, y_{i_k})$ je podpostupnosť pôvodnej postupnosti, v ktorej sú menované len prvky A obsahujúce na mieste x-u len prvky Y. Označili sme teda $|Y| = k$. Použijúc pevné y zmodifikujme T na T_y tak, že $T_y(p) = x_{i_p}$ pre $1 \leq p \leq |Y|$. Aplikovaním vety 2.1.2 ale dostávame

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

kde c záleží len od A, čím je veta dokázaná. □

2.3 Prefixová Kolmogorovská zložitosť

Táto práca sa zaoberá len jednoduchou Kolmogorovskou zložitosťou, takou, ako bola doteraz definovaná. V skutočnosti sa však vytvorila aj iná verzia, tzv. *prefixová Kolmogorovská zložitosť*. Vznikla miernou modifikáciou pôvodnej kvôli tomu, aby sa odstránili niektoré jej nežiadúce vlastnosti.

Doteraz definovaná KZ napríklad nie je aditívna, nerovnosť $C(x, y) \leq C(x) + C(y)$ neplatí pre všetky x a y. Ďalšou nevýhodou funkcie C je jej nemonotónnosť. Nie vždy platí $C(x) \leq C(xy)$, prefix reťazca nemusí byť nutne menej zložitý ako celý reťazec.

Prefixová KZ má z tohto hľadiska lepšie vlastnosti. Ako už v predošlej kapitole bolo povedané, pracujeme s prefixovými kódmi. U prefixovej KZ sa nezaoberáme čiastočne rekurzívnymi funkciami, ale čiastočne rekurzívnymi *prefixovými* funkciami, ktoré sú definované nasledovne.

Definícia 2.3.1. Čiastočne rekurzívna prefixová funkcia $\phi : \{0,1\}^* \to \mathbb{N}$ je čiastočne rekurzívna funkcia taká, že ak $\phi(p) < \infty$ aj $\phi(p) < \infty$, tak p nie je prefixom q.

Pre čiastočne rekurzívne prefixové funkcie platí ekvivalent vety 2.1.2 o čiastočne rekurzívnych funkciách. V dôkaze sa najskôr ukáže existencia univerzálneho prefixového TS U takého, že $U(\langle y, \langle i, p\rangle\rangle) = T'_i(y, p)$ pre $\forall y, p \in \mathbb{N}$. Zvyšok dôkazu je analogický k dôkazu spomínanej vety.

Definícia 2.3.2. Vyberme aditívne optimálnu čiastočnú rekurzívnu prefixovú funkciu ψ_0 ako referenčnú a definujme prefixovú Kolmogorovskú zložitosť x podmienenú y ako

$$K(x|y) = C_{\psi_0}(x|y). \tag{2.13}$$

Nepodmienená prefixová KZ je potom definovaná ako

$$K(x) = K(x|\epsilon). \tag{2.14}$$

Pre prefixovú KZ potom napr. platí $K(x, y) \leq K(x) + K(y)$, čo je niekedy žiaduca vlastnosť. Taktiež si môžeme všimnúť zbytočnosť zápisu $\langle x, y\rangle$. Vyžadujúc, aby dekódovací algoritmus bol bezprefixový, môžeme teraz popísať oba reťazce bez toho, aby bolo treba špecifikovať ich koniec a začiatok. Preto $K(\langle x, y\rangle) = K(x, y)$.

2.4 Nestlačiteľnosť

Keď už som vybudoval potrebný aparát, definujeme kľúčový pojem pri všetkých dôkazoch robených pomocou KZ. Tým pojmom je *nestlačiteľnosť*. Myšlienka je taká, že reťazec označíme za nestlačiteľný v prípade, že jeho KZ už nemožno stlačiť pod jeho dĺžku (jeho KZ nebude menšia ako jeho dĺžka). To znamená, že nevieme nájsť žiadny program kratší ako samotný reťazec, ktorý by ho exaktne popisoval. Najlepším a najkratším popisom takéhoto reťazca si je potom on sám, najkratšie je vypísať celý reťazec ako taký. Formálne:

Definícia 2.4.1. Hovoríme, že binárny reťazec x je *nestlačiteľný* ak

$$C(x) \geq l(x), \tag{2.15}$$

$$resp.\ C(x|y) \geq l(x). \tag{2.16}$$

Ako sme si už na začiatku ukázali, napr. reťazec π nie je zrovna nestlačiteľný, keďže jeho dĺžka sa rovná ∞, no jeho KZ je rovná konštante. Ako teda vyzerá taký nestlačiteľný reťazec? V jeho zápise zrejme nie sú žiadne časti, ktoré vieme popísať nejakým kratším programom, tak by sme totiž dokázali skrátiť jeho popis. Keď by sme aj niektorú jeho časť vedeli zapísať kratším popisom, zas by sme potrebovali veľa informácie na špecifikáciu umiestnenia tejto časti v reťazci, takže nič neušetríme. Tak isto tento reťazec nebudeme vedieť vypočítať žiadnym krátkym algoritmom. Zrejme v reťazci nenájdeme žiadne opakujúce sa časti, žiadne vzory, žiadne pravidelnosti. Intuitívne sa v postupnosti budú jednotlivé znaky striedať skutočne náhodne, keďže ich nebudeme nijak vedieť odhadnúť. Z tohto dôvodu sa takýmto reťazcom hovorí aj *Kolmogorovsky náhodné*, resp. len *náhodné*. Ukazuje sa totiž, že nestlačiteľné reťazce naozaj úspešne prechádzajú všetkými efektívnymi testami náhodnosti. [3]

V skutočnosti pre každú dĺžku n existuje aspoň jeden nestlačiteľný reťazec s touto dĺžkou. Prečo? Pre každé n existuje 2^n binárnych reťazcov dĺžky n, ale len $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ možných kratších opisov. Ostáva aspoň ten jeden, ktorý bude nestlačiteľný.

[3] Pojem test náhodnosti nás pre naše účely trápiť nemusí. Vedzme len, že je to akýsi algoritmus, ktorý nám na základe nejakých vlastností povie či je reťazec (s nejakou pravdepodobnosťou) náhodný. Viac sa dočítate v [16].

Definícia 2.4.2. Hovoríme, že reťazec x je *c-nestlačiteľný* (c-náhodný) pre nejakú konštantu c, ak

$$C(x) \geq l(x) - c. \tag{2.17}$$

Definícia 2.4.3. Podobne pre celočíselnú funkciu $g(n)$ je x dĺžky n *g-nestlačiteľný* (g-náhodný), ak

$$C(x) \geq n - g(n) \tag{2.18}$$

c-nestlačiteľné reťazce (analogicky pre $g(n)$-nestlačiteľné) sú teda také, ktorých popis vieme skrátiť o c bitov oproti dĺžke pôvodného reťazca. Použijúc rovnaký výpočet ako pri prvej definícii nestlačiteľnosti dostávame, že reťazcov, ktoré vieme stlačiť o c je $\sum_{i=0}^{n-c} 2^i = 2^{n-c} - 1$. Tých nestlačiteľných je potom $2^n - (2^{n-c} - 1) = 2^n - 2^{n-c} + 1$. Všimnime si, že definícia nestlačiteľnosti hovorí vlastne o 0-nestlačiteľných reťazcoch. A z uvedeného výpočtu dostávame aj to, že aspoň polovica reťazcov dĺžky n je 1-nestlačiteľná, aspoň tri štvrtiny sú 2-nestlačiteľné..., až napokon, že $(1 - 1/2^c)$-tý zlomok všetkých reťazcov dĺžky n sú c-nestlačiteľné reťazce. Takže ako vidíme, väčšina reťazcov je c-nestlačiteľná pre nejaké c (samozrejme pre $c < n$). Tieto zistenia môžeme všeobecne zhrnúť v jednoduchej, no o to užitočnejšej vete.

Veta 2.4.4 (Veta o nestlačiteľnosti)**.** *Nech c je kladné celé číslo. Pre každé y, každá konečná množina A s mohutnosťou m obsahuje aspoň $m - m/2^c + 1$ prvkov x takých, že*

$$C(x|y) \geq \log m - c. \tag{2.19}$$

Dôkaz. Ako už vieme, počet reťazcov z A, kde $|A| = m$ kratších ako $\log m - c$ je

$$\sum_{i=0}^{\log m - c - 1} 2^i = 2^{\log m - c} - 1. \tag{2.20}$$

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

Takže vidíme, že značná časť všetkých reťazcov, skoro všetky, sú nestlačiteľné. Tento fakt budeme vďačne využívať a budeme predpokladať, že skoro všetky reťazce sú nestlačiteľné.

Zadefinujme si ešte jeden pojem, ktorý bude v priebehu textu využitý. Ak vieme, že x patrí do nejakej podmnožiny celých čísel A, môžeme jeho zložitosť brať ako $C(x|A)$. Napríklad $C(x) = C(x|\mathbb{N})$, keď vieme, že x je prirodzené číslo.

Definícia 2.4.5. Odchýlka náhodnosti[4] x relatívna k A je definovaná ako

$$\delta(x|A) = l(|A|) - C(x|A). \tag{2.21}$$

Keďže z vety 2.2.2 už vieme, že $C(x|A) \leq l(|A|) + c$, dostávame $\delta(x|A) \geq -c$ pre nejakú konštantu c nezávislú od x.

[4] z anglického randomness deficiency

Kapitola 3

Metóda nestlačiteľnosti

Ako som už ukázal, takmer všetky reťazce sú nestlačiteľné (Veta o nestlačiteľnosti 2.4.4). Tento fakt budeme využívať pri dôkazoch pomocou tzv. *metódy nestlačiteľnosti*. Metódoa nestlačiteľnosti využíva KZ a nestlačiteľnosť objektov v dôkazoch. Je aplikovateľná do všemožných oblastí nielen matematiky a informatiky. Metóda funguje všeobecne zhruba takto:

Vezmeme si ľubovoľný nestlačiteľný objekt z množiny, s ktorou pracujeme. Vieme, že taký existuje, preto si ho môžeme zvoliť. Fakt, že objekt je nestlačiteľný platí pre takmer všetky objekty z danej množiny, preto môžeme vybraný objekt považovať za akéhosi reprezentanta tejto množiny. Nemusíme hľadať nijaký konkrétny objekt, vezmeme si ľubovoľný, vieme totiž, že taký nestlačiteľný objekt určite existuje a jeho existencia je pre nás postačujúca. Jeho nestlačiteľnosť v skutočnosti nevieme dokázať, ale to ani nie je nutné. Ďalej ukážeme platnosť dokazovanej vlastnosti na tomto objekte. Dôkazy sú v drvivej väčšine sporom, preto sa využije argument, že keby daná vlastnosť pre tento objekt neplatila, mohli by sme ho vyjadriť kratším popisom, čím dostaneme spor s jeho nestlačiteľnosťou. V dôkazoch teda pracujeme len s jedným objektom, čím sa dôkazy stávajú jednoduchšími. V mnohých prípadoch touto metódou dospejeme k výsledku platnému v priemernom prípade, keďže nestlačiteľnosť je vlastnosťou skoro všetkých objektov.

Ukážeme si využitie tejto metódy v dôkazoch tvrdení z oblasti formálnych jazykov a automatov, triediacich algoritmov a grafov.

3.1 Formálne jazyky a automaty

3.1.1 Pumpovacia lema

Ako jednoduchú ukážku využitia Kolmogorovskej zložitosti vo formálnych dôkazoch, si na začiatok ukážeme dôkaz neregulárnosti jazyka. Na dokazovanie (ne)príslušnosti jazykov do triedy regulárnych jazykov sa v teórii jazykov využíva dobre známa tzv. *pumpovacia lema*. Hoci sa u čitateľa predpokladá jej znalosť, uvediem aspoň jej znenie bez dôkazu.

Lema 3.1.1 (Pumpovacia lema)**.** *Ku každému regulárnemu jazyku L existuje konštanta p taká, že ku každému slovu $w, w \in L, |w| \geq p$ existujú slová x, y a z, pre ktoré platí:*

(i) $w = xyz$

(ii) $|xy| \leq p$

(ii) $|y| \geq 1$

(iv) $\forall i, i \geq 0$ *slovo* xy^iz *patrí do* L

Názov lemy vznikol z poslednej menovanej vlastnosti, ktorá hovorí, že strednú časť slova môžeme 'pumpovať' môžeme iterovať ľubovoľne veľakrát (hoci aj nulakrát) a slovo bude stále patriť do jazyka.

3.1.2 Regulárnosť jazyka

Už máme definované všetko potrebné, pozrime sa preto na prvú ukážku použitia metódy nestlačiteľnosti. Dokážeme, že jazyk $L = \{0^k1^k | k \geq 1\}$ nie je regulárny. Najskôr si na porovnanie uveďme dôkaz pomocou pumpovacej lemy.

Dôkaz 1. Máme jazyk $L = \{0^k1^k | k \geq 1\}$. Zvoľme z neho slovo $w = 0^p1^p$, kde p je číslo z pumpovacej lemy. Podľa lemy toto slovo vieme zapísať ako xyz. Nech však w rozdelíme akokoľvek (podľa pravidiel lemy), podslovo xy ako aj samotné y bude obsahovať len nuly, keďže $|xy| \leq p$ a na začiatku slova w máme p núl. No a keď toto slovo začneme pumpovať, začnú nám pribúdať v slove nuly, avšak počet jednotiek sa nezmení, čím sa pokazí štruktúra slova a to už nebude patriť do jazyka L. Slovo $w = xy^iz$ už teda (pre $i \neq 1$) nepatrí do L. Preto jazyk L *nie je* regulárny. □

A teraz si predveďme dôkaz pomocou metódy nestlačiteľnosti

Dôkaz 2. Dôkaz bude *sporom*, tak ako sú aj takmer všetky ostatné dôkazy pomocou tejto metódy.

Predpokladajme, že jazyk L je regulárny. Potom existuje deterministický konečný automat (DKA) A, ktorý ho akceptuje. Uvažujme k, pre ktoré platí

$$C(k) \geq \log k, \tag{3.1}$$

teda k je nestlačiteľné. [1] Z vety o nestlačiteľnosti (2.4.4) vieme, že také k existuje. Nech A je po prečítaní prvých k núl vstupného slova v stave q. Všimnime si, že tento stav a samotné A sú pre nás popisom čísla k. To preto, lebo A začínajúc v q na reťazci obsahujúcom len jednotky akceptuje práve po k prečítaných jednotkách, keďže akceptuje len slová tvaru 0^k1^k. Dĺžka zápisu A a q je rovná konštante. Takže existuje nejaká konštanta c nezávislá od k taká, že $C(k) < c$. A by podľa tohto mal akceptovať L len pre k také, že $\log k \leq C(k) < c$, čím dostávame *spor*. Pre dostatočne veľké k (konkrétne také k, že $\log k \geq c$) by totiž A už neakceptoval. Jazyk $L = \{0^k1^k | k \geq 1\}$ preto *nie je* regulárny. □

3.1.3 Regulárnosť definovaná pomocou Kolmogorovskej zložitosti

Keďže čitateľ je zrejme zvyknutý skôr na dôkazy pomocou pumpovacej lemy, než metódou nestlačiteľnosti, uvediem ešte dva jednoduché príklady, na ktorých si môže porovnať tieto dve metódy a zvyknúť si na uvažovanie v pojmoch KZ. Neskôr budú už dôkazy čiste pomocou KZ.

Využívať niektoré popisy regulárnych jazykov na dôkaz ich *ne*regulárnosti je niekedy veľmi náročné. Tieto popisy sú zväčša užitočné najmä na dokazovanie ich regulárnosti (napr. Myhill-Nerodova veta). Tu nachádzame ďalšie výrazné pozitívum metódy nestlačiteľnosti. Uvediem interpretáciu práve spomínanej Myhill-Nerodovej vety v reči Kolmogorovskej zložitosti, ktorú môžeme použiť dokonca aj v prípadoch, keď nám pumpovacie lemy nepomôžu.

Veta 3.1.1. *Nech L je regulárny jazyk. Potom pre nejakú konštantu c závisiacu len od L a pre každý reťazec x, ak y je n-tý reťazec v lexikografickom*

[1] Uvedomme si, že k je vlastne binárny reťazec reprezentujúci prirodzené číslo k.

poradí v $L_x = \{y \mid xy \in L\}$ *(prípadne v doplnku* L_x*), tak*

$$C(y) \leq C(n) + c. \tag{3.2}$$

Poznámka 3.1.2. L_x je vlastne jazyk všetkých suffixov slova x takých, že slovo xy patrí do L.

Dôkaz. Slovo y môžeme popísať pomocou

- popisu DKA akceptujúceho L, tj. O(1) bitov
- stavu DKA po spracovaní x, tj. O(1) bitov
- čísla n, $C(n)$ bitov,

čo dáva dokopy $C(n) + c$ bitov, pre nejakú konštantu c. □

3.1.4 Regulárnosť-prvočísla

Lema 3.1.2. *Jazyk* $L = \{1^p \mid p$ *je prvočíslo*$\}$ *nie je regulárny.*

Dôkaz (Pumpovacia lema). Predpokladajme, že L je regulárny, tzn. dôkaz *sporom*. Nech n je najmenšie prvočíslo väčšie alebo rovné p, kde p je celé číslo z pumpovacej lemy (3.1.1). Nech $w = 1^n$, zrejme $w \in L$. Podľa pumpovacej lemy 1^n môžeme prepísať ako xyz, kde $|y| \geq 1$, a $xy^iz \in L$ pre $\forall i$. Taktiež platí $|xy| \leq p$. Nech $i = n + 1$. Podľa pumpovacej lemy musí slovo $xy^{n+1}z$ patriť do L. Toto slovo vieme prepísať ako $xyzy^n = 1^n y^n$ keďže obsahuje len jednotky. Vieme, že pumpovaná časť y má dĺžku $1 \leq |y| \leq n$, nech $y = 1^m$. Potom $1^n y^n = 1^{n+nm} = 1^{n(m+1)}$, ale $n(m+1)$ je zrejme zložené číslo, preto toto slovo do L napatrí, čím dostávame *spor*. Jazyk L *nie je* regulárny. □

Dôkaz (KZ). Dôkaz bude prebiehať opäť *sporom*. Predpokladajme, že L je regulárny. Nech pre reťazec xy z vety 3.1.1 platí $xy = 1^p$, kde p je $(k+1)$-vé prvočíslo. Nech $x = 1^{p'}$, kde p' je k-te prvočíslo. Potom $y = 1^{p-p'}$ a n z vety 3.1.1 je rovné 1. Z toho dostávame, že $C(p-p') = \mathrm{O}(1)$. Avšak rozdiel medzi prvočíslami rastie neohraničene s ich veľkosťou, z čoho dostávame, že neohraničené množstvo celých čísiel má $KZ = \mathrm{O}(1)$. Ale možných popisov dĺžky O(1) je predsa len O(1), nie je ich neohraničene veľa, dostávame *spor*. Jazyk L *nie je* regulárny. □

3.1.5 Regulárnosť-najväčší spoločný deliteľ

Lema 3.1.3. *Jazyk* $L = \{0^i 1^j \mid nsd(i,j) = 1\}$ *nie je regulárny.*

Poznámka 3.1.3. $nsd(i,j)$ označuje najväčšieho spoločného deliteľa čísiel i a j.

Dôkaz (Pumpovacia lema). Aj tento dôkaz bude *sporom*. Nech L je regulárny a n je najmenšie prvočíslo také, že $n \geq p$, kde p je konštanta z pumpovacej lemy. Nech $w = 0^n 1^{(n-1)!}$ Slovo w zrejme patrí do L, ľahko vidno, že $nsd(n,(n-1)!) = 1$. Podľa pumpovacej lemy môžeme w prepísať ako xyz, kde $1 \leq |y|$ a $xy^i z \in L$ pre $(\forall i)(i \geq 0)$. Tiež $|xy| \leq p$ a preto xy je prefixom slova 0^n. Nech $i = 0$. Potom aj slovo $0^a 1^{(n-1)!}$, kde $a = n - |y| < n$ by malo patriť do L ale zrejme nepatrí, pretože $nsd(a,(n-1)!) \neq 1$ pre žiadne $2 \leq a < n$. Máme *spor*, jazyk L *nie je* regulárny. □

Dôkaz (KZ). Nech slovo x z vety 3.1.1 je 0^p, kde p je ľubovoľné prvočíslo. Potom 1^p je lexikograficky prvé neprázdne slovo v $L_x^c \cap 1^*$, kde L_x^c označuje komplement jazyka L_x. $L_x^c \cap 1^*$ sú vlastne všetky slová skladajúce sa zo samých jednotiek, ktoré nemôžu byt sufixom slova x, lebo xy by už nepatrilo do L. No a keďže p je prvočíslo, slovo 1^p je v $L_x^c \cap 1^*$ lexikograficky prvé, ich dĺžky sú totiž súdeliteľné, resp sú rovné. Keby bolo v $L_x^c \cap 1^*$ lexikograficky prvé nejaké iné, kratšie slovo, jeho dĺžka by nedelila p, preto by malo patriť do $L_x \cap 1^*$ a nie do $L_x^c \cap 1^*$. Podľa tohto nám ale z vety 3.1.1 plynie, že existuje konštanta c taká, že pre všetky prvočísla p platí $C(p) < c$, čo je samozrejme nepravda. Opäť sme dospeli k *sporu* s predpokladom, že jazyk L je regulárny. □

3.1.6 Konverzia nedeterministického konečného automatu na deterministický

V tejto časti sa nebudeme venovať algoritmu ako prerobiť nedeterministický konečný automat (NKA) na deterministický (DKA). Bude nás zaujímať len minimálny počet stavov, ktoré takto získaný DKA musí mať. Keďže nedeterminizmu sa u DKA zbavíme tak, že stavy budú reprezentované podmnožinami množiny stavov, je známe, že DKA vytvorený z NKA môže potrebovať až 2^n stavov, kde n je počet stavov transformovaného NKA. V tejto časti si pomocou metódy nestlačiteľnosti ukážeme dôkaz nasledovného tvrdenia.

Veta 3.1.4. *Na konverziu NKA s n stavmi na DKA potrebujeme v najhoršom prípade* $\Omega(2^n)$ *stavov.*

Dôkaz. Vezmime si jazyk $L_k = \{x \mid k\text{-ty bit zprava je } 1\}$. Zrejme existuje nejaký NKA A' s $k+1$ stavmi akceptujúci L_k (A' potrebuje len napočítať do k a v nejakom stave musí začať, preto práve tých $k+1$ stavov). Nech L_k akceptuje nejaký DKA A. Dôkaz bude opäť *sporom.*

Predpokladajme, že A má len $o(2^k)$ stavov.Vezmime si nejaký nestlačiteľný reťazec x dĺžky k, $C(x|k, A) \geq k$ (kde A si je sám sebe popisom). Opäť vieme, že takýto reťazec existuje. Pridajme na koniec reťazca x k núl, dostávame $x0^k$ (pre naše účely v skutočnosti stačí na koniec prilepiť akýkoľvek binárny reťazec dĺžky k). Uložme si stav automatu A po prečítaní vstupu x. Slovo x patrí do L_k práve vtedy, keď jeho prvý bit je 1, lebo $|x| = k$ a slovo z L_k má k-ty bit zprava 1. Takže podľa tohto stavu (podľa toho, či je akceptačný) vieme určiť prvý bit x. Keď bude A čítať ďalej prvú nulu z k pridaných núl, prejde do nejakého stavu. Opäť ak je tento stav akceptačný, k-ty bit zprava od práve čítaného (teda v x druhý bit zľava) je 1, inak 0. Takto postupne bit po bite vieme zistiť každý z k bitov x, až zistíme celé x. [2] Vidíme, že za predpokladu, že poznáme k a A, potrebujeme vedieť len stav q automatu A po prečítaní x. Keďže A má len $o(2^k)$ stavov, na zápis stavu q potrebujeme jasne menej ako k bitov. Tým sme ale dostali popis reťazca x kratší ako je jeho dĺžka, preto $C(x|k, A) < k$, čo je *spor* s predpokladom, že x je nestlačiteľný. Takže DKA A potrebuje v najhoršom prípade $\Omega(k)$ stavov. □

3.1.7 Hľadanie podreťazcov jednosmerným DKA s tromi hlavami

Hľadanie podreťazcov v reťazci sa tiež nazýva *string matching.* Existuje preň niekoľko algoritmov, nás bude zaujímať riešenie pomocou konečných automatov (KA). KA rieši túto úlohu, ak akceptuje jazyk $SUBSTR = \{\$x\#y\$ \mid x \; je$ podreťazcom $y \; a \; x, y \in \{0,1\}^*\}$. Pomocou KZ sa postupne podarilo dokázať, že string matching nemožno robiť pomocou DKA s dvoma hlavami, ani s troma hlavami [11], ba dokonca žiadnym DKA s k hlavami pre žiadne k [7]. Všetky tieto výsledky ilustrujú silu tejto metódy. My sa

[2] To isté sme mohli docieliť aj tak, že automatu A začínajúcemu v stave, do ktorého sa dostal po prečítaní x, by sme na vstupe dali binárny reťazec dĺžky $k-1$. Ďalej by sme takým istým spôsobom zrekonštruovali celé x.

nebudeme zaoberať samotným hľadaním podreťazcov pomocou DKA, ani si nepredvedieme kompletný dôkaz jedného z vyslovených tvrdení. Ako už podnadpis naznačuje, bude nás zaujímať prípad, keď počet hláv automatu je 3, konkrétne si ukážeme dôkaz tzv. *pohybovej lemy*, ktorá zohrala rozhodujúcu úlohu pri dôkaze, že tri hlavy nestačia.

Najskôr si však definujme jednosmerný deterministický konečný automat s troma hlavami. Je to šestica $A = (\Sigma, Q, \delta, q_0, F, 3)$, kde Σ je abeceda, Q je množina stavov, δ je prechodová funkcia, q_0 je počiatočný stav a F je množina akceptačných stavov. Zaveďme preň označenie 3-DKA. V každom kroku na základe aktuálneho stavu a trojice symbolov práve čítaných hlavami h_1, h_2, h_3 A deterministicky mení stav a posúva niektoré zo svojich hláv o jednu pozíciu do prava. Pre každý vstup každá z hláv dosiahne ukončovací znak \$. A akceptuje, ak je v akceptačnom stave a všetky tri hlavy už prečítali posledný znak. Predpokladáme, že hlavy detekujú ich vzájomné kolízie, keď čítajú znak na tej istej pozícii.

Teraz už sľúbený dôkaz pohybovej lemy. Originál dôkazu nájdete napr. v [11, 9].

Lema 3.1.4 (Pohybová lema)**.** *Nech a je blok textu taký, že $|a| = n$ a $C(a) \geq n - \mathrm{O}(\log \mathrm{n})$. Predokladajme, že 3-DKA A je v stave q a jedna z hláv h_i, $1 \leq i \leq 3$, je na prvom bite a pričom $n > |Q|$.*
Potom buď

(1) kým h_i prečíta a, niektorá ďalšia hlava sa pohne aspoň o jednu pozíciu,alebo

(2) h_i sa bude pohybovať, kým nedosiahne koniec vstupu alebo nestretne niektorú inú hlavu, pričom hlavy rôzne od h_i stoja nehybne.

Dôkaz. Ak neplatí (2), to znamená, že nejaká hlava rôzna od h_i sa pohne, kým h_i dočíta celý vstup alebo stretne inú hlavu. Chceme dokázať, že potom musí platiť (1), teda že niektorá hlava sa pohne kým h_i číta a. Predpokladajme opak, a síce, že žiadna z hláv sa nepohne, kým h_i číta a. Kľúčovým v dôkaze bude, že ak h_i prečíťa a bez pohybu iných hláv, bude a stlačiteľené pomocou A.

Z každého stavu A existuje postupnosť symbolov dĺžky najviac $|Q|$, ktorá spôsobí pohyb inej hlavy, inak by sme dostali cyklus, a v nejakých stavoch sa musia pohnúť aj ďalšie hlavy. A má len $|Q|$ stavov a najkratšia cesta (ak vôbec cesta existuje) zo stavu do iného stavu je maximálne $|Q|$. Takže A

môže prečítať najviac $|Q|$ symbolov, aby sa počas čítania nepohla aj dalšia hlava. Kým h_i číta a, nehýbe sa žiadna iná hlava. Preto z každého bitu a existuje postupnosť symobolov dĺžky najviac $|Q|$, ktorá sa na páske nemôže vyskytnúť, pretože by viedla k rozhýbaniu inej hlavy. Takže na každom bite máme najviac $2^{|Q|} - 1$ postupností, ktoré sa môžu vyskytnúť.

Postupností a s týmito vlastnosťami je najviac $(2^{|Q|} - 1)^{\frac{n}{|Q|}}$, pretože $n > |Q|$ a každá z možných postupností nemôže začínať na každom z n bitov postupnosti a, ale len na každom z $\frac{n}{|Q|}$ bitov a (teda na každom $|Q|$-tom bite), keďže každá z týchto možných postupností má dĺžku $|Q|$. Ak chceme vybrať jednu z týchto postupností a, môžeme ju jednoznačne identifikovať jej indexom v efektívnom vymenovaní všetkých týchto postupností. Potrebujeme na to len

$$\log(2^{|Q|} - 1)^{\frac{n}{|Q|}} = \frac{n}{|Q|}(|Q| - \epsilon) = (1 - \epsilon)n \tag{3.3}$$

bitov, kde ϵ je konštanta závislá len od $|Q|$. Takže na popis a potrebujeme len

- $(1 - \epsilon)n$ bitov na popis indexu a,
- O(1) bitov na popis A,
- O(1) bitov na popis aktuálneho stavu q automatu A,
- O(1) bitov na popis aktuálnych bitov čítaných inými hlavami,

čo je dokopy

$$(1 - \epsilon)n + \mathrm{O}(1) < \mathrm{C(a)}. \tag{3.4}$$

Dostali sme teda spor s predpokladom, že a nestlačiteľný reťazec, takže musí platiť (1), lema platí. □

Celý dôkaz v [11] využíva dokázanú lemu. Prvá hlava musí najskôr prečítať vstup po znak #. Pomocou lemy sa dokáže, že na vstupe $x\#a^{2n}x$ h_2 už dočíta x po znak #, kým h_1 príde na koniec a^{2n}. Ak by to nebola pravda, h_1 by dočítala celý vstup, kým h_2 a h_3 by podľa pohybovej lemy stáli na mieste. To by h_1 prišla na koniec vstupu skôr ako by h_2 prečítala znak #. Potom kým h_2 prečíta a^{2n}, hlava h_3 už prešla znak # alebo h_2 dočíta celý vstup, zatiaľ čo h_3 sa nehýbe. Každopádne výskyt x nebude zistený.

3.2 Zložitosť triediacich algoritmov

V časti o triediacich algoritmoch sa budeme zaoberať dokazovaním ich časovej zložitosti v priemernom prípade. Dokazovanie zložitosti v priemernom prípade však býva ošemetnou záležitosťou, keďže tá sa získava tak, že sa zistí zložitosť všetkých vstupov požadovanej dĺžky a potom sa spraví priemer. My už vieme ako fungujú dôkazy metódou nestlačiteľnosti a práve tu sa Kolmogorovská zložitosť, resp. metóda nestlačiteľnosti ukazuje ako výborná voľba. Tento problém totiž vyriešime tak, že si zvolíme len jeden vstup, ktorý bude pre nás v istom zmysle *reprezentatívny*. Na ňom ukážeme platnosť požadovanej vlastnosti. Jeho reprezentatívnosť spočíva v tom, že keďže nestlačiteľné sú takmer všetky reťazce, plynie nám, že jeho daná vlastnosť platí v priemere pre všetky vstupy.

3.2.1 Bubblesort

Sekciu venovanú triediacim algoritmom začneme analýzou dobre známeho triediaceho algoritmu *Bubblesort*. O tomto triedení je všeobecne známe, že jeho časová zložitosť je $\Theta(n^2)$. Analýzu jeho zložitosti klasickými metódami môžete nájsť napr. v[8]. Pri tomto dôkaze som čerpal z [15] a kvôli niektorým nejednoznačnostiam v dôkaze aj z konzultácií s pánom Paulom M. B. Vitányim.

Bubblesort pracuje v prechodoch. Prechody robí zľava doprava, avšak rovnako sa dá implementovať aj opačným smerom. V každom prechode sa deje nasledovné: Začneme prvým prvkom postupnosti a porovnáme ho s jeho nasledovníkom. Ak je prvý prvok väčší ako druhý, sú vymenené, inak sa nedeje nič. Pokračuje sa prvkom, ktorý je teraz na druhej pozícii. Opäť sa porovná s ďalším prvkom, väčší z nich sa ocitne na tretej pozícii. Pre n prvkové pole vždy vezmeme i-ty prvok postupne pre $1 \leq i \leq n-1$ a porovnáme ho s prvkom na pozícii $i+1$. Ak je prvok na i-tej pozícii väčší, vymeníme ich, na $i+1$-ej pozícii bude teda vždy väčší z nich. Prechod končí vykonaním týchto krokov na prvku na pozícii $n-1$. Nasleduje ďalší prechod. Bubblesort pokračuje, kým v prechode nastane výmena prvkov. Ak v niektorom prechode už nebudú žiadne dva prvky vymenené, znamená to, že pole je už utriedené, algoritmus sa zastaví. Väčšie prvky sa presúvajú zľava doprava. Ak pri presúvaní prvok narazí na väčší prvok, väčší z nich sa posúva ďalej. Prvky sa takto presúvajú v prechodoch, kým nie sú utriedené.

Je zrejmé, že na utriedenie postupnosti n prvkov nám stačí najviac $n-1$ prechodov, jednak preto, lebo v každom prechode je Bubblesort schopný presunúť aspoň jeden prvok na správnu pozíciu, a jednak preto, že po premiestnení $n-1$ prvkov na správne pozície bude určite aj ten posledný n-tý prvok na správnom mieste (už nemôže byť na zlej pozícii, všetky ostatné pozície sú totiž už okupované ostatnými prvkami, ktoré sú už na svojich miestach). V každom kroku algoritmus zrejme môže spraviť najviac $n-1$ výmien dvoch susediacich prvkov. Z toho nám jasne plynie, že počet uskutočnených výmien môžeme odhadnúť ako $\mathrm{O(n^2)}$. To už máme triviálny horný odhad. My sa budeme zaoberať ohraničením zdola, o ktorom vieme, že sa rovná $\Omega(n^2)$.

Veta 3.2.1. *Bubblesort vykoná v priemernom prípade aspoň $\Omega(n^2)$ výmien prvkov.*

Dôkaz. V dôkaze využijeme, že na vstupe dĺžky n môžeme dostať $n!$ postupností, resp. permutácií. Ak sa dozvieme ako presne prebiehal proces triedenia, teda kam sa prvky presúvali, vieme z utriedenej postupnosti zistiť pôvodnú neutriedenú postupnosť. Preto uvažujeme $n!$ navzájom rôznych procesov triedenia. Vieme určiť dolný odhad celkového počtu výmien prvkov potrebných na utriedenie postupnosti.

Označme si náš algoritmus Bubblesortu B. Chceme vzostupne utriediť permutáciu π prvkov množiny $\{1, 2, 3, \dots n\}$. Nech

$$\pi = a_1 a_2 \dots a_n, \tag{3.5}$$

kde a_i pre $1 \leq i \leq n$ je z $\{1, 2, \dots, n\}$. Vieme, že existuje $n!$ takýchto permutácií. Nech m_i označuje vzdialenosť prvku i v počiatočnej permutácii π od jeho pozície vo finálnej utriedenej postupnosti. Intuícia hovorí, že táto vzdialenosť je rovná počtu miest, o ktoré je prvok na začiatku posunutý od svojej finálnej pozície. Tu je ale vzdialenosť myslená trochu inak. Pri určení vzdialeností postupujeme nasledovne.

Na začiatku sú všetky vzdialenosti rovné nule. Začíname so vstupnou postupnosťou zľava pre $i = 1$. Ak prvok a_i je už na svojom správnom mieste, pokračujeme s a_{i+1}. Vzdialenosť m_{a_i} prvku a_i od jeho správnej pozície a_i, môžeme vyjadriť ako $a_i - i + k$, kde k je počet prvkov väčších ako a_i, ktoré sa po posunutí prvku a_i ocitnú vľavo od neho. Samozrejme ak sa po navýšení tejto vzdialenosti o k ďalšie väčšie prvky ocitnú vľavo od prvku $a - i$, musíme o ich počet vzdialenosť opäť zväčšiť. Totiž väčšie prvky vľavo od a_i sa

počas triedenia tiež presnunú na svoje správne pozície, čím prvok a_i posunú doľava. Prvok a_i posunieme o jeho vzdialenosť m_i. Na 1. pozícii v postupnosti sa teraz ocitol iný prvok, a_j, s ktorým postupujeme analogicky. Postup opakujeme, kým nezískame utriedenú postupnosť. Potom

$$M := \sum_{i=1}^{n} m_i \tag{3.6}$$

je vlastne celkový počet porovnaní/výmien potrebných na utriedenie postupnosti π. Ak sa už prvok i v niektorom prechode Bubblesortu začne presúvať vpravo, zrejme to neznamená, že sa v danom prechode premiestni na svoju finálnu pozíciu. Cestou totiž môže naraziť na nejaký väčší prvok j, ktorý sa bude presúvať namiesto neho ďalej. To však znamená, že zvyšok cesty m_i, ktorú prvok i ešte neprecestoval, prejde v niektorých ďalších prechodoch. Takže každý prvok prejde počas triedenia práve toľko miest, aká je jeho počiatočná vzdialenosť od finálnej pozície. Zrejme $M \leq n^2$, pretože každý z n prvkov môže byť od svojho finálneho umiestnenia vzdialený najviac o $n-1$ miest. Keď teraz na vstupe dostaneme $m_1, m_2, \ldots, m_n$, vieme zrekonštruovať pôvodnú permutáciu π.

Samotná rekonštrukcia je jednoduchá, keď už vieme ako zistiť vzdialenosti prvkov. Na vstupe máme utriedenú postupnosť $1, 2, \ldots, n$ a postupnosť $m_n, \ldots, m_1$ v tomto poradí. V pôvodnej postupnosti jednoducho presunieme postupne prvok i pre každé i začínajúc od $i = n$ končiac s $i = 1$ o m_i pozícií vľavo a dostaneme pôvodnú postupnosť.
Uvediem dva príklady.

Príklad 1. Majme na vstupe 6-prvkovú postupnosť

$$3, 6, 5, 1, 4, 2.$$

Číslo 3 je od svojej správnej pozície vzdialené o 2 pozície, avšak po jeho posunutí sa vľavo od neho ocitnú aj čísla 6 a 5, preto jeho vzdialenosť určíme ako $m_3 = 4$. Posunieme ho teda o 4 miesta vpravo, dostávame postupnosť
$6, 5, 1, 4, 3, 2$.
Prvý prvok postupnosti je teraz číslo 6. To je od svojej pozície vzdialené o 5 miest, keďže je v postupnosti najväčšie, vzdialenosť už nemusíme navýšiť, preto $m_6 = 5$. Po jeho posunutí dostávame
$5, 1, 4, 3, 2, 6 \Rightarrow m_5 = 4$
$1, 4, 3, 2, 5, 6$

m_1 ostáva na nule, keďže číslo 1 je na svojej správnej pozícii.
Ideme na číslo 4, $m_4 = 2$
$1, 3, 2, 4, 5, 6$
1 je opäť na správnej pozícii, 3 musíme ešte posunúť o jedno miesto, preto navýšime m_3 na 5.
$1, 2, 3, 4, 5, 6$
Už nemusíme nič presúvať.
Takto teda vieme určiť našu vzdialenosť m_i prvkov postupnosti.

Samotná rekonštrukcia:
Na vstupe máme utriedenú postupnosť a $m_6 = 5, m_5 = 4, m_4 = 2, m_3 = 5$, $m_2 = m_1 = 0$ v tomto poradí.
$1, 2, 3, 4, 5, 6$
$m_6 = 5$, preto číslo 6 posuniem o 5 miest vľavo
$6, 1, 2, 3, 4, 5$
Keďže $m_5 = 4$, ďalej dostávame
$6, 5, 1, 2, 3, 4$
$m_4 = 2 \Rightarrow 6, 5, 1, 4, 2, 3$
$m_3 = 5 \Rightarrow 3, 6, 5, 1, 4, 2$
$m_2 = m_1 = 0 \Rightarrow$ už máme pôvodnú postupnosť.

Príklad 2. Nech vstupná postupnosť je $2, 4, 3, 1$. Potom číslo 2 je od svojej správnej pozície vzdialená o 1 miesto, to by sa však vľavo od nej ocitla štvorka, tak vzdialenosť dvojky zvýšime na 2, avšak po posunutí o dve miesta by bola vpravo aj od 3, preto vzdialenosť zvýšime na 3. Takže $m_2 = 3$.
$4, 3, 1, 2$
m_4 sa potom rovná 3
$3, 1, 2, 4 \Rightarrow m_3 = 2$
$1, 2, 3, 4$

Teraz ako z utriedenej postupnosti a postupnosti vzdialeností získame vstupnú postupnosť.
$1, 2, 3, 4$
$m_4 = 3 \Rightarrow 4, 1, 2, 3$
$m_3 = 2 \Rightarrow 4, 3, 1, 2$
$m_2 = 3 \Rightarrow 2, 4, 3, 1$

$m_1 = 0$, už máme pôvodnú postupnosť.

Položme si otázku, koľko rôznych postupností $m_1, m_2, \ldots, m_n$ existuje. Vlastne sa pýtame, koľko existuje rozkladov čísla M na n nezáporných sčítancov, resp. koľkými spôsobmi možno rozdeliť M objektov do n podmnožín (môžu byť aj prázdne, keď prvok neposúvame). Pridajme k týmto M objektom $n-1$ oddeľovačov. Keď chceme totiž označiť n podpostupností, stačí nám $n-1$ oddeľovačov, keď beriem do úvahy, že prvá podpostupnosť (m_1) začína pred prvým oddeľovačom a posledná (m_n) končí za posledným na konci postupnosti. Úlohou teda je z terajších $M+n-1$ objektov vybrať $n-1$ oddeľovačov. Každý výber jednoznačne popisuje práve jedno rozdelenie M objektov do n množín. A to sú predsa jednoduché kombinácie. Preto existuje práve

$$B(M) = \binom{M+n-1}{n-1} \tag{3.7}$$

možností ako rozdeliť M na n usporiadaných nezáporných sčítancov. Permutáciu π teraz vieme popísať pomocou M, n, indexom logaritmu $\log B(M)$ na popísanie postupnosti $m_1, \ldots, m_n$ a programu P, ktorý nám pôvodnú permutáciu z týchto parametrov zrekonštruuje (a utriedenú postupnosť samozrejme poznáme). Vezmime si počiatočnú postupnosť π, pre ktorú platí

$$C(\pi \mid n, B(M), P) \geq \log n! - \log n \tag{3.8}$$

Podľa Vety o nestlačiteľnosti existuje takých permutácii n prvkov aspoň $n!(1-1/n)$. Taktiež môžeme predpokladať $M \geq n$. V opačnom prípade (potrebujeme vykonať menej ako n výmien) by sme túto postupnosť vedeli utriediť v čase O(n). Popis, podľa ktorého chceme počiatočnú postupnosť skonštruovať, musí mať dĺžku aspoň $C(\pi|n, B(M), P)$ (pretože KZ reťazca je dĺžka jeho najkratšieho popisu). Takže potrebujeme zapísať M a $B(M)$ ako popis pôvodnej postupnosti. Na zápis týchto dvoch údajov v samooddeľujúcom tvare potrebujeme

$$\log\log M + 1 + \log\log M + \log M + \log B(M) \tag{3.9}$$

bitov, pričom vieme, že musí platiť

$$2\log\log M + \log M + \log B(M) + \mathrm{O}(1) \geq \mathrm{n}\log\mathrm{n} - \mathrm{O}(\log\mathrm{n}), \tag{3.10}$$

nemôžeme totiž dostať popis lepší ako $C(\pi \mid n, B, P)$. Výraz $\log n!$ môžeme upraviť na $n \log n$ podľa Stirlingovho vzorca

$$n! \approx \frac{n^n}{e^n}\sqrt{2\pi n}, \tag{3.11}$$

Upravme teraz výraz $\log B(M)$. Využijeme nasledujúcu rovnosť použitú napr. aj v [16]:

$$\log \binom{a}{b} = b \log \frac{a}{b} + (a-b) \log \frac{a}{a-b} + \frac{1}{2} \log \frac{a}{b(a-b)} + \mathrm{O}(1) \tag{3.12}$$

Dôkaz tejto rovnosti pre jeho náročnosť nebudem uvádzať, v prípade záujmu je prenechaný na čitateľa. Táto rovnosť ešte však bude v priebehu práce využitá. Aplikujeme ju na $\log B(M)$ a dostávame

$$\log \tbinom{M+n-1}{n-1} = (n-1) \log \tfrac{M+n-1}{n-1} + M \log \tfrac{M+n-1}{M} + \tfrac{1}{2} \log \tfrac{M+n-1}{M(n-1)} + \mathrm{O}(1)$$

Na prvý pohľad sa možno nezdá, že sme tým niečo získali, no výraz $M \log \frac{M+n-1}{M} = \log \left(1 + \frac{n-1}{M}\right)^M$ môžeme zjednodušiť podľa vzorca

$$e^a > (1 + \frac{a}{b})^b, \tag{3.13}$$

ktorý platí pre $\forall a > 0$ a kladné celé číslo b. Takže uvedený výraz nahradíme výrazom $\log e^{n-1}$. Takže zatiaľ máme

$$2 \log \log M + \log M + (n-1)(\log (1 + \tfrac{M}{n-1}) + \log e) + \tfrac{1}{2} \log \tfrac{M+n-1}{M(n-1)} \geq n \log n - \mathrm{O}(\log \mathrm{n})$$

Celú rovnicu predelíme n-kom a predpokladajúc $2 < n \leq M \leq n^2$ môžeme škrtať zbytočné výrazy a dostávame

$$\log (1 + \frac{M}{n-1}) \geq \log n + \mathrm{O}(1) \tag{3.14}$$

Z toho potom vidíme, že $M \geq n(n-1)$, teda

$$M \geq n^2. \tag{3.15}$$

Keďže uvedené vzťahy platia pre $(1 - 1/n)$-tý zlomok všetkých permutácií n prvkov a M nám ako celkový počet potrebných výmien/porovnaní vlastne udáva zložitosť algoritmu, dostávame pre toto triedenie ohraničenie $\Omega(n^2)$ v priemernom prípade. □

3.2.2 Shellsort

Ďalšou zastávkou na prehliadke triediacich algoritmov bude *Shellsort* (podľa pána D. L. Shella). Určenie netriviálneho dolného odhadu zložitosti tohto triedenia v priemernom prípade bolo otvoreným problémom vyše štyri desaťročia. Tento problém bol vyriešený len nedávno, a to práve pomocou Kolmogorovskej zložitosti. Takýto dôkaz môžete nájsť v [15, 3]. Neskôr sa ukázalo, že použitý argument sa dá preniesť aj do klasických metód zisťovania zložitosti. Dôkaz bez KZ nájdete v [13].

Shellsort triedi n-prvkovú postupnosť π v p prechodoch využívajúc postupnosť tzv. *inkrementov* $h_1, h_2, \ldots, h_p$. V k-tom prechode rozdelí postupnosť na h_k 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 $i, (i + h_k), (i + 2h_k), \ldots, (i + \lceil n/h_k \rceil h_k)$ (resp. posledný člen podpostupnosti môže mať index $i + \lfloor n/h_k \rfloor h_k$). Teda index prvku mod h_k je vlastne nejaké $j - 1$, pričom j je z $\{1, 2, \ldots, h_k\}$. Každá z týchto podpostupností je utriedená jendoduchým *Insertsortom*. Efektívnosť tohto 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 zostať utriedená len čiastočne, aj keby jednotlivé podpostupnosti boli pekne utriedené).

Pre istotu uvediem 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 spomínané podpostupnosti. Tieto sú utriedené Insertsortom

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

a dostávame čiastočne utriedenú postupnosť $1, 7, 2, 5, 8, 3, 6, 9, 4, 10$.
Ďalej by sme postupovali pre inkrement h_2 atď. Všimnime si, že keďže $h_p = 1$, na koniec dostaneme už len jednu podpostupnosť (stĺpec), ktorá bude rovná celej doterajšej čiastočne utriedenej postupnosti s n prvkami. Posledný krok teda len Insertsortom utriedi celú doteraz utriedenú postupnosť.

Výpočet Shellsortu pozostáva z porovnaní a inverzií (výmien) prvkov. Budeme počítať len pohyby dátami=výmeny prvkov, avšak ten istý odhad, ako ku ktorému sa dopracujeme, platí aj pre počet porovnaní.

Veta 3.2.2. *Priemerný počet porovnaní a tiež aj inverzií v p-prechodovom Shellsorte na n-prvkovej postupnosti je aspoň $\Omega(pn^{1+1/p})$ pre ľubovoľnú postupnosť inkrementov, pre $p = \mathrm{o}(\log \mathrm{n})$. Priemer sa berie zo všetkých postupností n prvkov s rovnakou pravdepodobnosťou (rovnomerné rozdelenie pravdepodobnosti).*

Dôkaz. V dôkaze sa najskôr dopracujeme k odhadu počtu potrebných výmien prvkov. Z nich sme schopní zrekonštruovať pôvodnú postupnosť. Podobne ako pri Bubblesorte využijeme, že vieme koľko je rôznych rozdelení tohto počtu na sčítance, teda na 'inverzie' konkrétnych prvkov (bude vysvetlené v dôkaze). Za vstupnú permutáciu si zvolíme nestlačiteľnú, čo v kombinácii s množstvom informácie potrebnej na kódovanie triedenej postupnosti využijeme na získanie dolného odhadu.

Nech vstupná postupnosť je permutáciou π prvkov $\{1, 2, \ldots, n\}$. Uvažujme $(h_1, h_2, \ldots, h_p)$ algoritmus Shellsortu A, kde h_k je inkrement v k-tom prechode a $h_p = 1$. Triviálnym odhadom zdola by mohlo byť $\Omega(pn)$, 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

$$M = \sum_{k=1}^{p}\sum_{i=1}^{n} m_{i,k} \tag{3.16}$$

Postupnosť $m_{1,k}, \ldots m_{n,k}$ jasne špecifikuje počiatočnú permutáciu prechodu k. Preto ak dostaneme $m_{1,k}$ pre všetky $1 \leq i \leq n$ a $1 \leq k \leq p$ (dokopy ich bude np), vieme zrekonštruovať počiatočnú permutáciu π.

Ukážme si jednu časť tejto rekonštrukcie. Pracujme s postupnosťou dĺžky 8. Práve pracujeme na i-tom prechode, chceme zistiť počiatočnú postupnosť prechodu i. Zatiaľ sme sa dopracovali k postupnosti
$2, 1, 6, 3, 4, 8, 7, 5$ a vieme, že $m_{1,i} = 2$, $m_{2,i} = 1$, $m_{3,i} = 1$, $m_{4,i} = 0$, $m_{5,i} = 0$, $m_{6,i} = 1$, $m_{7,i} = 0$, $m_{8,i} = 0$. Vieme, že $h_i = 3$. Postupnosť rozdelíme na podpostupnosti

$$\begin{matrix} 2 & 1 & 6 \\ 3 & 4 & 8 \\ 7 & 5 \end{matrix}$$

Na základe $m_{1,i}, \ldots m_{8,i}$ vieme, že pred utriedením jednotlivých podpostupností to vyzeralo takto

$$\begin{matrix} 7 & 4 & 8 \\ 2 & 5 & 6 \\ 3 & 1 \end{matrix}$$

Takže postupnosť na začiatku i-teho prechodu bola $7, 4, 8, 2, 5, 6, 3, 1$. Takto vieme rekonštruovať postupnosti v jednotlivých prechodoch až zrekonštruujeme pôvodnú postupnosť.

Podobne ako v dôkaze zložitosti Bubblesortu, aj tu využijeme, že existuje

$$D(M) = \binom{M + np - 1}{np - 1} \tag{3.17}$$

rôznych rezdelení M na np zoradených nezáporných celých sčítancov $m_{i,k}$. Každé takéto rozdelenie môže byť jednoznačne popísané prirodzeným číslom j, ktoré je vlastne jeho poradovým číslom vo vymenovaí týchto rozdelení. Takže popis M a j popisujú postupnosť prvkov $m_{i,k}$. Nech teda P je program pre UTS, ktorý rekonštruuje zoznam prvkov $m_{i,k}$ z tohto popisu. Existuje $n!$ rôznych permutácií n prvkov. Nech je permutácia π nestlačiteľná s Kolmogorovskou zložitosťou

$$C(\pi \mid n, A, P) \geq \log n! - \log n, \tag{3.18}$$

Ako už vieme, takýchto permutácií existuje veľa. Zrejme existuje program, ktorý na vstupe A, P, n zrekonštruuje π z popisu prvkov $m_{i,k}$. Preto minimálna dĺžka popisu prvkov $m_{i,k}$ spolu s $\mathrm{O}(1)$ bitmi na popis tohto programu musí prevyšovať zložitosť π:

$$C(m_{1,1}, \ldots, m_{n,p} \mid n, A, P) + \mathrm{O}(1) \geq \mathrm{C}(\pi|\mathrm{n}, \mathrm{A}, \mathrm{P}) \geq \log \mathrm{n}! - \log \mathrm{n} \tag{3.19}$$

M opäť môže slúžiť ako dolný odhad počtu vykonaných inverzií. Takže zaujíma nás popis M a j, ktoré spolu popisujú postupnosť prvkov $m_{i,k}$, lebo z nich už vieme zostrojiť pôvodnú permutáciu π. Dĺžka tohto popisu musí opäť prevyšovať Kolmogorovskú zložitosť popisovanej permutácie π. M a j teda zapíšeme ako samooddeľujúci reťazec, potrebujeme na to

$$2\log\log M + \log M + 1 + \log D(M) \geq \log n! - \log n - \mathrm{O}(1) \tag{3.20}$$

bitov. Vieme, že $M \leq pn^2$, keďže každé $m_{i,k} \leq n$. Taktiež môžeme predpokladať $p < n$ (keby bolo viac ako n prechodov, mohli by sme dostať aspoň n^2 porovnaní). Môžeme teraz tento výraz upravovať:

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

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

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

$$\log D(M) \geq \log n! - \log n - 2\log\log n - 3\log n - \mathrm{O}(1)$$

$$\log D(M) \geq \log n! - 4\log n - 2\log\log n - \mathrm{O}(1) \tag{3.21}$$

Ako už bolo spomínané pri Bubblesorte, opäť využijeme odhad $\log D(M)$ podľa už použitého vzorca (3.12). Dostávame

$$\log \binom{M+np-1}{np-1} =$$
$$(np-1)\log\frac{M+np-1}{np-1} + M\log\frac{M+np-1}{M} + \frac{1}{2}\log\frac{M+np-1}{M(np-1)} + \mathrm{O}(1)$$

Druhý sčítanec na pravej strane môže byť odhadnutý podľa (3.13) na

$$\log\left(1+\frac{np-1}{M}\right)^M < \log e^{np-1} \tag{3.22}$$

pre kladné M a $np-1>0$. Po menšej úprave

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

Vidíme, že pre $n \to \infty$

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

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

$$(np-1)\left(\log\left(1+\frac{M}{np-1}\right)+\log e\right) \tag{3.23}$$

Na $\log n!$ môžeme aplikovať Stirlingovu aproximáciu (3.11) a pravá strana nerovnosti (3.21) konverguje k $n \log n$ pre $n \to \infty$. Môžeme písať

$$(np-1)\left(\log\left(1+\frac{M}{np-1}\right)+\log e\right) \geq n \log n \tag{3.24}$$

a postupnými úpravami sa pre $p = \mathrm{o}(\log \mathrm{n})$ dostaneme k výsledku

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

$$p\log\left(1+\tfrac{M}{np-1}\right) \geq \log n$$

$$1+\tfrac{M}{np-1} \geq n^{1/p}$$

$$M \geq pn^{1+1/p}, \tag{3.25}$$

teda

$$M = \Omega(pn^{1+1/p}) \tag{3.26}$$

Získali sme odhad pre permutáciu π, pre ktorú platí $C(\pi \mid n, A, P) \geq \log n! - \log n$. Takýchto permutácií je však aspoň $n!(1-1/n)$ a preto priemerná zložitosť je aspoň

$$\left(1-\frac{1}{n}\right)\Omega(pn^{1+1/p}) = \Omega(pn^{1+1/p}) \tag{3.27}$$

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

3.2.3 Varianty Shellsortu

Spomeňme si aj dva známe varianty Shellsortu a ich zložitosti. Dobosiewicz v [4] navrhol pre každú vzniknutú podpostupnosť podľa inkrementov namiesto jej utriedenia Insertsortom len jeden prechod Bubblesortu. Tento algoritmus nesie meno Dobosiewicz sort. Naproti tomu Incerpi a Sedgewick v [6] navrhujú pre každú podpostupnosť dva prechody Bubblesortu, jeden zľava a druhý zprava. Toto triedenie nazývame Shaker sort, keďže evokuje miešanie shakerom. Oba tieto varianty však takto môžu zrejme ponechať vstupnú postupnosť neutriedenú, hoci posledný inkrement je 1. Jeden ani

dva prechody Bubblesortu nemusia postupnosť utriediť. Preto sa vyžaduje záverečná fáza, a síce je to Insertsort aplikovaný na túto 'skoro utriedenú' postupnosť.

Tieto varianty neboli ani zďaleka tak študované ako samotný Shellsort, preto sa o ich zložitostiach vie menej. Nejaké odhady sa v minulosti už podarilo získať (viď. [4, 14, 6, 19]). Ale len nedávno Bronislava Brejová [3] v [3] dokázala, že Shaker sort triedi v najhoršom prípade v čase $\mathrm{O(n^{3/2})\log^3 n}$. Pomocou metódy nestlačiteľnosti dokázala aj odhady zdola v priemernom prípade oboch variantov, a to je to, čo nás teraz zaujíma.

Veta 3.2.3. *Nech H je postupnosť p inkrementov $h_1, \ldots, h_p$. V priemernom prípade s inkrementmi H má Dobosiewicz sort zložitosť $\Omega(n^2/2^p)$ a Shaker sort $\Omega(n^2/4^p)$.*

Dôkaz. Dokážem vetu pre Dobosiewicz sort. Vieme, že každé triedenie založené na porovnávaní prvkov potrebuje v priemere aspoň $\Omega(n \log n)$, čo jasne platí pre $p > \log n - \log\log n$, ako môžeme vidieť, pre $p = \log n - \log\log n$ platí:

$$\frac{n^2}{2^p} = \frac{n^2}{2^{\log n - \log\log n}} = \frac{n^2 2^{\log\log n}}{n} = n \log n. \tag{3.28}$$

Preto budeme ďalej predpokladať $p < \log n - \log\log n$.
Nech vstupnou postupnosťou je nejaká permutácia π množiny $\{1, 2, \ldots, n\}$. Nech π' je permutácia, ktorú dostaneme po p prechodoch Dobosiewicz sortu (ešte pred finálnym Insertsortom). Nech X je počet všetkých inverzií [4] v π'. Potom záverečný Insersort pracuje v čase $\Omega(X)$. Dokážeme, že $X = \Omega(n^2/2^p)$ v priemernom prípade.

Vytvoríme unikátny popis permutácie π, prvá časť popisu umožňuje skonštruovať samotné π z π' a druhá časť popíše permutáciu π'. Prvá časť sa skladá z p reťazcov dĺžky n $(x_1, \ldots, x_n)$, pričom j-ty bit i-teho reťazca nám hovorí, či prvok x_j bol v i-tom prechode vymenený s prvkom x_{j-h_i}. Z takýchto postupností a permutácie π' vieme ľahko simulovať kroky vzad a krok po kroku rekonštruovať jednotlivé 'medzipermutácie' až nakoniec pôvodnú permutáciu π.

Permutáciu π' popíšeme jednoducho postupnosťou inverzií $a_1, \ldots, a_n$, kde a_i je počet prvkov vľavo od π'_i (π'_i je i-ty prvok v π'), ktoré sú od neho

[3] Bronislava Brejová, absolventka našej fakulty
[4] v poli $a_1, \ldots, a_n$ máme inverziu (a_i, a_j) ak $i < j$ a $a_i > a_j$

väčšie. Zrekonštruovať pôvodnú permutáciu z tohto popisu už pre čitateľa so skúsenosťami z rekonštrukcií v predošlých dôkazoch dozaista nebude problém.

Potom $\sum_{i=1}^{n} a_i = X$. Ako už vieme, existuje presne $D(X) = \binom{X+n-1}{n-1}$ postupností n nezáporných sčítancov so súčtom X. π' môže byť preto popísané číslom X a indexom postupnosti $a_1, \ldots, a_n$ v enumerácii všetkých takýchto postupností. Na to potrebujeme

$$L(\pi) = 2 \log \log X + \log X + \log D(X) + 1 \tag{3.29}$$

bitov. Na popis π potrebujeme spolu

$$np + 2 \log \log X + \log X + \log D(X) + 1, \tag{3.30}$$

tam sme zarátali aj np bitov potrebných na popis p postupností dĺžky n na popis π'. Použijeme už nám známy vzorec (3.12) pre $\log \binom{a}{b}$ a dostávame

$$\log D(X) \leq \log \binom{X+n}{n} =$$
$$n \log \frac{X+n}{n} + X \log \frac{X+n}{X} + \frac{1}{2} \log \frac{X+n}{Xn} + \mathrm{O}(1).$$

Druhý sčítanec opäť môžeme odhadnúť najviac na $n \log e$. Z predošlých použití tohto vzorca už vieme, že ďalej môžeme písať

$$\log D(X) \leq n \log \left(\frac{X}{n} + 1 \right) + \mathrm{O(n)}. \tag{3.31}$$

Potom

$$L(\pi) \leq np + n \log \left(\frac{X}{n} + 1 \right) + \mathrm{O(n)} \tag{3.32}$$

Nech P je program, ktorý z n, p a H vytvorí π. Vieme, že

$$L(\pi) \geq C(\pi | n, p, H, P) \geq \log n! - \log n, \tag{3.33}$$

pričom existuje veľa postupností π s takouto zložitosťou. Z práve uvedeného nám plynie, že

$$np + n \log \left(\frac{X}{n} + 1 \right) + \mathrm{O(n)} \geq \mathrm{n \log n} - \Theta(\mathrm{n}) \tag{3.34}$$

$$\log \left(\frac{X}{n} + 1 \right) \geq \log n - p - \Theta(1) = \log n - \log 2^p - \Theta(1) = \log \frac{n}{2^p \Theta(1)}$$

$$\frac{X}{n} + 1 \geq \frac{n}{2^p \Theta(1)}$$

$$X \geq \frac{n^2}{2^p \Theta(1)} - n = \Omega(\frac{n^2}{2^p}) \tag{3.35}$$

Zložitosť algoritmu pre permutáciu π je preto $\Omega(\frac{n^2}{2^p})$. Tento odhad platí pre $n!(1 - 1/n)$ permutácií, preto priemerná zložitosť je

$$\left(1 - \frac{1}{n}\right) \Omega(\frac{n^2}{2^p}) = \Omega(\frac{n^2}{2^p}). \tag{3.36}$$

Dolný odhad pre Shaker sort získame úplne analogicky, akurát na zápis jedného kroku algoritmu potrebujem $2n$ bitov namiesto n. Preto dostávame dolný odhad $\Omega(\frac{n^2}{4^p})$. □

3.2.4 Heapsort

Pozrime sa teraz na ďalšie triedenie, ktorého zložitosť v priemernom prípade bola otvoreným problémov tridsať rokov. Bol vyriešený práve pomocou metódy nestlačiteľnosti. Pri písaní dôkazu som čerpal z [16, 15, 9].

Heapsort vymyslel v roku 1964 pán J. W. J. Williams, vzápätí bol mierne vylepšený pánom R. W. Floydom, ktorý znížil počet vykonaných porovnaní. Tento algoritmus patrí k tým najznámejším a je často využívaný aj kvôli tomu, že zaručene pracuje v čase $n \log n$. Heapsort je triedením *'in situ'* (*'na mieste'*), teda triedenie prebieha len v rámci triedeného poľa $A[1 \ldots n]$ bez prídavnej pamäte. Pracuje s dátovou štruktúrou zvanou *halda*.[5]

Definícia 3.2.4. Pole prvkov $k_1, \ldots, k_n$, kde $k_i \in \mathbb{N}$ nazývame haldou, ak tieto prvky sú čiastočne usporiadané tak, že platí

$$k_{\lfloor j/2 \rfloor} \geq k_j \; pre \; 1 \leq \lfloor j/2 \rfloor < j \leq n. \tag{3.37}$$

Halda môže byť vizualizovaná ako úplný binárny strom (až na poslednú úroveň, kde je len zľava úplný, keďže tam môžu tie najpravejšie prvky chýbať, podľa n) s n uzlami. Každý uzol reprezentuje prvok poľa, ktorý uchováva hodnotu tohto uzla. Najväčšia hodnota je uložená v koreni stromu a každý ďalší uzol má hodnotu menšiu alebo rovnú ako jeho otec. V halde je každý podstrom tiež haldou.

Kvôli dôkazu tohto triedenia považujem za vhodné uviesť presný algoritmus triedenia. Algoritmus bude zapísaný v pseudokóde.[6] Heapsort najskôr

[5] z anglického *heap*

[6] Poznámka. length[A] je dĺžka poľa A, zatiaľ čo heap-size[A] je veľkosť haldy. Halda nemusí stále obsahovať všetky prvky poľa, preto jej veľkosť môže byť menšia ako veľkosť poľa. LEFT(i) (RIGHT(i) vracia ľavého (pravého) syna

zo vstupného poľa vytvorí haldu procedúrou BULID-HEAP a potom v nej usporadúva prvky (HEAPSORT).

```
BUILD-HEAP(A)
1  heap-size[A]←length[A]
2  for i← ⌊length[A]/2⌋ downto 1 do
3     HEAPIFY(A,i)
```

```
HEAPIFY(A,i)
1  l←LEFT(i)
2  r←RIGHT(i)
3  if (l≤heap-size[A]) and (A[l]>A[i])
4     then largest←l
5     else largest←i
6  if (r≤heap-size[A]) and (A[r]>A[largest])
7     then largest←r
8  if largest≠i then
9     exchange A[i]↔A[largest]
10    HEAPIFY(A,largest)
```

Vidíme, že BULID-HEAP zavolá procedúru HEAPIFY postupne na prvkoch na pozíciách $\lfloor length[A]/2 \rfloor, \ldots, 1$. Nemusí ju volať až od pozície n, pretože tie zvyšné prvky sú v strome len listami, teda sú bez synov a na nich netreba nič upravovať (uzol sám o sebe je haldou). HEAPIFY len jednoducho skontroluje, či je otec väčší ako jeho synovia. Ak nie je, vymení otca s jeho najväčším synom a v tom prípade sa zavolá HEAPIFY aj na uzol, do ktorého sa otec teraz dostal. A takto až na dno stromu (k listom). Všimnime si, že BULID-HEAP funguje tak, že keď sa zavolá na nejakom uzle, tak podstromy s koreňmi v jeho synoch už sú haldami.

Už vieme ako sa vytvára halda, ukážme si algoritmus samotného triedenia.

HEAPSORT(A)

```
1 BUILD-HEAP(A)
2 for i←length[A] downto 2 do
3   exchange A[1]↔A[i]
4   heap-size[A]←heap-size[A]−1
5   HEAPIFY(A,1)
```

Ako som už spomínal, HEAPSORT najskôr zo vstupného poľa vytvorí haldu. Potom od posledného prvku poľa po druhý, tento prvok vždy vymení s koreňom. Teda na koniec poľa sa dostane najväčší prvok, ten je potom z haldy odstránený, lebo už je na správnom mieste. A na prvok z konca, ktorý sa teraz stal koreňom, je zavolaná procedúra HEAPIFY, ktorá ho umiestni na jeho správne miesto v aktuálnej halde.

Ako už bolo v úvode spomenuté, existujú dve verzie tohto triedenia: Williamsova a Floydova. Floydova verzia potrebuje vykonať približne dvakrát menej porovnaní, avšak pre určenie zložitosti bude pre nás kľúčový počet výmien prvkov, ktorý je v oboch verziách rovnaký. Takže dokážeme zložitosť pre obe verzie. V tomto dôkaze je obrovskou výhodou metódy nestlačiteľnosti fakt, že sa nemusí zaoberať rozdelením jednotlivých háld, ktoré počas triedenia vznikajú. Ostatné metódy totiž narazia na problém, že hoci s počiatočným rovnomerným rozdelením pravdepodobností vstupov, je nesmierne náročné počítať rozdelenie vzniknutých háld a ešte ťažšie rozdelenie zmenšujúcich sa háld vznikajúcich počas procedúry HEAPSORT.

V dôkaze pomocou Kolmogorovskej zložitosti si najskôr ako vždy zvolíme nestlačiteľnú vstupnú permutáciu. Z jej zložitosti je v Tvrdení 1. odvodená zložitosť haldy jej prislúchajúcej. Využijeme to, že z haldy vieme zrekonštruovať pôvodnú permutáciu, čo je dokázané v Tvrdení 2. Následne ale aj z utriedeného poľa budeme schopní reprodukovať haldu, vďaka čomu dospejeme k priemernému počtu pohybov prvkov v uzloch.

Veta 3.2.5. *Heapsort vykoná v priemernom prípade (rovnomerné rozdelenie)* $n \log n + \mathrm{O(n)}$ *pohybov prvkami.*

Dôkaz. Nech vstupnou permutáciou pre Heapsort je nestlačiteľná permutácia p taká, že

$$C(p|n) \geq n \log n - 2n. \tag{3.38}$$

Jej existenciu môžeme uvažovať vďaka vete o nestlačiteľnosti 2.4.4 (+Stirling 3.11).

Tvrdenie 1. *Potom pre haldu h, ktorú vytvorí procedúra BULID-HEAP, platí*

$$C(h|n) \geq n \log n - 6n. \tag{3.39}$$

Dôkaz. Ukážeme, že keby toto neplatilo, vieme popísať permutáciu p menej ako $n \log n - 2n$ bitmi.

Predpokladajme $C(h|n) < n \log n - 6n$. Pre každé j, keď procedúra BULID-HEAP (resp. HEAPIFY) posúva prvok A[j] z koreňa haldy smerom dole na jeho správnu pozíciu, zakódujeme si cestu, ktoru bol tento prvok posúvaný. Uložíme si 0, ak šiel prvok do ľava (vymenil sa s ľavým synom), 1 ak šiel do prava a 2 keď sa už zastaví. Na zakódovanie tejto procedúry budeme potrebovať

$$n \log 3 \sum_j \frac{j}{2^{j+1}} \tag{3.40}$$

bitov, lebo máme n prvkov, [7] v každom kroku si ukladáme jednu z troch možností (0,1,2), na zápis jednej z nich potrebujem $\log 3$ bitov a priemerná dĺžka cesty, ktorou pôjde prvok z koreňa dole, je $\sum_j \frac{j}{2^{j+1}}$, lebo existuje 2^{j+1} ciest dĺžky j nanajvýš j. Prvok pri zostupovaní dolu pri HEAPIFY nemusí zastaviť až v liste, preto beriem do úvahy aj cesty kratšie ako j. Využitím rovnosti

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2 \tag{3.41}$$

dostávame

$$n \log 3 \sum_j \frac{j}{2^{j+1}} \leq 2n \log 3. \tag{3.42}$$

Z haldy h a takéhoto popisu ciest vieme jednoducho simulovať BUILD-HEAP odzadu a zrekonštruovať p. Preto

$$C(p|n) < C(h|n) + 2n \log 3 + \mathrm{O}(1) < \mathrm{C(h|n)} + 4\mathrm{n} < \mathrm{n \log n} - 2\mathrm{n}, \tag{3.43}$$

[7] Hodnota n je to použitá ako v spomínaných publikáciách, z ktorých som čerpal. Oku pozorného čitateľa však určite neušlo, že v tomto kroku by sme odhad mohli urobiť trochu tesnejším. Keď si všimneme, že procedúra BUILD-HEAP volá HEAPIFY nie pre všetkých n prvkov, ale len od $\lfloor n/2 \rfloor$ nižšie, mohli by sme n nahradiť $n/2$. Týmto ťahom by sme však v skutočnosti nič nezískali, pretože aj tak by sme dospeli k sporu tak isto ako k nemu dospejeme s pôvodnou hodnotou n.

čím dostávame spor, takže platí $C(h|n) \geq n \log n - 6n$. □

Z utriedenej postupnosti potrebujeme získať h. Na to nám stačí uchovať históriu posledných $n-1$ reorganizácií haldy počas procedúry HEAPSORT. V skutočnost nám stačí uchovať si konečné pozície len pre $i := n-1, \ldots, 2$, na ktoré bol prvok A[i] v halde vložený. Pozíciu zakódujeme popisom cesty, ktorú prvok od koreňa prejde, až niekde zastaví. Cesta bude reprezentovaná postupnosťou núl a jednotiek, pričom 0 znamená vľavo a 1 vpravo, označme ju s, resp. s_i. Potom $\delta_i = \log n - l(s_i)$ je vlastne akýsi skrátený zápis dĺžky postupnosti s_i. Tento zápis sa nám hodí, pretože v každej nižšej úrovni haldy je dvakrát viac prvkov ako v tej vyššej. Preto veľa prvkov pôjde od koreňa po dlhých cestách dole, zatiaľ čo krátkych ciest bude málo. A na zápis dlhšej cesty potrebujem predsa viac bitov ako na zápis kratšej cesty, preto je vhodné túto dĺžku odrátať od $\log n$, čo je dĺžka najdlhšej cesty. Tým sa mi obráti pomer počtu dlhších a kratších ciest. Cesty budeme kódovať samoodeľujúcimi reťazcami v tvare $1^{\log \delta_i} 0 \delta_i s_i$, na čo potrebujeme

$$2 \log \delta_i + l(s_i) + 1 \tag{3.44}$$

bitov. Zápis všetkých týchto ciest za sebou bude tvoriť postupnosť H. O dĺžke tejto postupnosti potom platí

$$l(H) = \sum_{i=1}^{n} (2 \log \delta_i + l(s_i) + 1) \tag{3.45}$$

Tvrdenie 2. *Z postupnosti H a n vieme skonštruovať haldu h.*

Dôkaz. Spätne simulujeme HEAPSORT. Na začiatku je v poli A[1...n] utriedená permutácia s najmenším prvkom v A[1]. Pre $i = 2, 3, \ldots, n-1$ budeme robiť nasledovné: prvky počnúc A[1] na ceste $(n-i)$ v H[8] posunieme o uzol nižšie, teda do syna na tejto ceste. Posledný prvok na tejto ceste už nebude kam posunúť, pretože cesta sa skončila, preto ho vložíme do A[i] a hodnotu, ktorá bola pred tým v A[i] vložíme do A[1]. Takže začneme na $i = 2$. Hodnotu v uzle A[2] si zapamätáme, prvky na ceste s_{n-2} počnúc A[1] postupne posúvame po tejto ceste nižšie, až posledný prvok tejto cesty nemáme kam dať, tak ho vložíme do A[2] a do A[1] vložíme doterajšiu hodnotu A[2]. A ďalej ideme na uzol A[3]... a tak postupne až po A[n-1].

[8] teda na poslednej ceste v H. V H čítame cesty od poslednej po prvú.

Posledná úprava v postupnosti je potom zrejmá, ostáva už len presunúť hodnotu v A[n] do A[1] s tým, že opäť posúvame dole prvky počnúc koreňom stromu na ceste od A[1] do A[n]. □

Uvediem príklad. Postupnosti a haldy budem zapisovať ako jednoduché polia, bez vizualizácie na binárne stromy. Čitateľovi pre názornosť odporúčam spraviť si tento príklad na binárnych stromoch. Pripomínam, že synmi uzla A[i] sú uzly A[2i] a A[2i+1]. Ukážeme si ako získať jednotlivé cesty s_i počas procedúry HEAPSORT.

Nech pôvodná halda je $A[1 \dots 7] = 10, 7, 8, 4, 3, 1, 5$. Najskôr sa vymenia hodnoty koreňa a uzla A[7], dostávame
$5, 7, 8, 4, 3, 1, \mathbf{10}$, pričom hrubo vyznačená hodnota 10 je už na svojom správnom mieste. Hodnota 5 v koreni sa teraz počas HEAPIFY premiestni do pravého syna koreňa, dostávame $8, 7, 5, 4, 3, 1, \mathbf{10}$. Cesta $s_7 = 1$. Opäť si koreň a posledný neutriedený prvok vymenia hodnoty, dostávame $1, 7, 5, 4, 3, \mathbf{8}, \mathbf{10}$. Hodnota 1 sa presunie po ceste $s_6 = 00$ a dostávame $7, 4, 5, 1, 3, \mathbf{8}, \mathbf{10}$. Ďalej
$3, 4, 5, 1, \mathbf{7}, \mathbf{8}, \mathbf{10} \rightarrow 5, 4, 3, 1, \mathbf{7}, \mathbf{8}, \mathbf{10} \rightarrow s_5 = 1$
$1, 4, 3, \mathbf{5}, \mathbf{7}, \mathbf{8}, \mathbf{10} \rightarrow 4, 1, 3, \mathbf{5}, \mathbf{7}, \mathbf{8}, \mathbf{10} \rightarrow s_4 = 0$
$4, 1, 3, \mathbf{5}, \mathbf{7}, \mathbf{8}, \mathbf{10} \rightarrow 3, 1, \mathbf{4}, \mathbf{5}, \mathbf{7}, \mathbf{8}, \mathbf{10} \rightarrow s_3 = \epsilon$, lebo 3 je najväčšia hodnota, preto sa nebude nikam posúvať.
$3, 1, \mathbf{4}, \mathbf{5}, \mathbf{7}, \mathbf{8}, \mathbf{10} \rightarrow 1, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{7}, \mathbf{8}, \mathbf{10} \rightarrow s_2 = \epsilon$, lebo 1 je opäť najväčšia hodnota, keďže už je len jediná.
$\mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{7}, \mathbf{8}, \mathbf{10} \rightarrow s_1$ triviálne.

Teraz k samotnej rekonštrukcii haldy z postupnosti $s_2, \dots, s_6$.
Na začiatku máme utriedenú postupnosť $1, 3, 4, 5, 7, 8, 10$. Začíname pre $i = 2$. $s_2 = \epsilon$, máme prázdnu cestu, preto len vymeníme hodnoty uzlov A[1] a A[2]. Dostávame $3, 1, 4, 5, 7, 8, 10$.
s_3 je opäť prázdna cesta, opäť vymeníme hodnoty koreňa a A[3]. Dostávame $4, 1, 3, 5, 7, 8, 10$.
$s_4 = 0$, prvky na tejto ceste posúvame o uzol nižšie. Takže hodnotu 5 koreňa do uzla A[2], ale pôvodnú hodnotu uzla A[2] už niet kam posunúť, lebo cesta sa skončila. Preto hodnotu uzla A[2] premiestnime do uzla A[4] a jeho pôvodná hodnota ide do koreňa. Dostávame $5, 4, 3, 1, 7, 8, 10$.
$s_5 = 1 \rightarrow 7, 4, 5, 1, 3, 8, 10$
$s_6 = 00 \rightarrow 8, 7, 5, 4, 3, 1, 10$
Tu sme vyčerpali postupnosť H. Tu máme však jedinú možnosť, presunúť

prvky na ceste od koreňa k A[7] o uzol nižšie, pričom hodnota uzla A[7] pôjde do koreňa.
$1, 3, 4, 5, 7, 8, 10$

Už vieme ako rekonštruovať haldu pomocou postupnosti H. Pokračujme ďalej v dôkaze.

Platí

$$n \log n - 6n \leq C(h|n) \leq l(H) + \mathrm{O}(1), \tag{3.46}$$

keďže tento popis musí byť kratší ako je KZ haldy h. Z toho ale dostávame

$$\sum_{i=1}^{n}(2 \log \delta_i + l(s_i) + 1) = \sum_{i=1}^{n}(2 \log \delta_i + \log n - \delta_i + 1) \geq n \log n - 6n, \tag{3.47}$$

lebo $l(s_i) = \log n - \delta_i$.
Ale keďže $\sum_{i=1}^{n} \log n < n \log n$, tak toto je možné len ak $\sum_{i=1}^{n} \delta_i = \mathrm{O(n)}$. Preto

$$\sum_{i=1}^{n} l(s_i) = n \log n - \sum_{i=1}^{n} \delta_i = n \log n - \mathrm{O(n)} \tag{3.48}$$

a priemerná dĺžka cesty je potom (po vydelení predošlého výrazu n) $\log n - c$, pre nejakú konštantu c. Pre každý prvok, ktorý sa počas procedúry HEAPSORT presúval na svoju správnu pozíciu, sa dĺžka cesty, po ktorej bol premiestnený zrejme rovná počtu jeho presunov. Preto celkový počet pohybov prvkov je aspoň

$$n \log n - nc = n \log n - \mathrm{O(n)}. \tag{3.49}$$

Keďže temer všetky permutácie sú Kolmogorovsky náhodné, tento odhad pre jednu z nich platí pre všetky permutácie v priemere. □

3.3 Kolmogorovsky náhodné grafy

V poslednej kapitole tejto práce ukážem využitie metódy nestlačiteľnosti v teórii grafov. Tieto dôkazy môžete nájsť v [10].

Každý ohodnotený graf $G = (V, E)$ s n vrcholmi môžeme reprezentovať ako binárny reťazec $E(G)$ dĺžky $\binom{n}{2}$, kde každý bit hovorí o existencii hrany medzi dvojicou vrcholov spomedzi všetkých možných $\binom{n}{2}$ dvojíc vrcholov v n-vrcholovom grafe. Jednotlivé dvojice budú zoradené, napr. lexikograficky, a i-ty bit bude rovný 1 v prípade existencie hrany i, inak bude nulový. Obrátene každý binárny reťazec dĺžky $\binom{n}{2}$ kóduje n-vrcholový graf. Mierne modifikovaná rekapitulácia definície 2.4.5:

Definícia 3.3.1. Ohodnotený graf s n vrcholmi má odchýlku náhodnosti najviac $\delta(n)$ a nazýva sa δ-náhodný, ak preň platí

$$C(E(G)|n) \geq \binom{n}{2} - \delta(n) \tag{3.50}$$

Ohodnotené grafy s vysokou KZ majú paradoxne veľa topologických vlastností, čo vyvoláva zdanie sporu s ich náhodnosťou. Avšak náhodnosť v tomto prípade implikuje striktné štatistické pravidelnosti. Štatistické vlastnosti reťazcov s vysokou KZ boli skúmané aj v [18].

3.3.1 Disjunktné cesty medzi vrcholmi, diameter

Definícia 3.3.2. Nech $x = x_1 \dots x_n$ je binárny reťazec dĺžky n a y je omnoho kratší binárny reťazec dĺžky l. Nech $p = 2^{-l}$ a $\#y(x)$ je počet (možno prekrývajúcich sa) rôznych výskytov y v x. Rátame aj výskyt y začínajúci na konci x a pokračujúci na začiatku x, teda akoby celé x bolo zapísané na kružnici.[9]

Keď napríklad $x = 0100110100101$ a $y = 1010$, tak v x sú celkom $\#y(x) = 3$ výskyty y: 0100**1010**0101, **0**100110100**101**, **010**0110100**1**0**1**.

Uveďme si znenie vety, ktorú budeme v ďalšom potrebovať. Jej dôkaz môžete nájsť v [16] (Veta 2.6.1, str. 162).

Veta 3.3.3. *Uvažujme notáciu z predošlej definície s $l \leq \log n$. Existuje konštanta c pre všetky n a $x \in \{0,1\}^n$ taká, že ak $C(x) \geq n - \delta(n)$, tak*

$$|\#y(x) - pn| \leq \sqrt{\alpha pn}, \tag{3.51}$$

[9] Keď vypíšeme všetky bity x v poradí, v akom sú v x, je vlastne jedno ktorým bitom začneme. Mohli by sme ho napísať napr. ako $x_i, x_{i+1}, \dots, x_n, x_1, x_2, \dots, x_{i-1}$

kde $\alpha = \frac{3l}{\log e}[K(y|n) + \log l + \delta(n) + c]$.

To v skutočnosti znamená, že

$$\#y(x) \leq pn \pm \sqrt{\alpha pn} \tag{3.52}$$

$\#y(x)$ vlastne označuje frekvenciu výskytu y v x.
Dokážme si teraz prvé tvrdenie v tejto sekcii.

Lema 3.3.1. *Vo všetkých* o(n)*-náhodných ohodnotených grafoch je* $n/4 +$ o(n) *disjunktných ciest dĺžky 2 medzi každými dvoma vrcholmi* i *a* j*. Navyše, všetky* o(n)*-náhodné ohodnotené grafy majú diameter 2.*

Dôkaz. Diameter je dĺžka najdlhšej cesty v grafe medzi dvoma vrcholmi. Úplné grafy nemusíme uvažovať, tie sú totiž jediné, čo majú diameter 1 a vieme ich opísať na O(1) bitov, tie teda nie sú náhodné. Ostáva nám uvažovať o(n)-náhodné grafy G s diametrom väčším alebo rovným 2. Nech i, j je dvojica vrcholov grafu G spojených r disjunktnými cestami dĺžky 2. Potom G môžeme popísať modifikáciou pôvodného kódu pre G pomocou nasledovného

- program na rekonštrukciu grafu z rôznych častí tohto kódovania na O(1) bitov
- zápis i a j, $i < j$ na $2 \log n$ bitov
- pôvodný kód E(G) grafu G s vymazanými $2(n-2)$ bitmi určujúcimi existenciu hrán $(i,k),(k,j)$ pre každé $k \neq i, j$, spolu $\binom{n}{2} - 2(n-2)$ bitov
- najkratší program pre reťazec $e_{i,j}$ obsahujúci preusporiadaných $n-2$ párov bitov vymazaných v predošlom bode. Jeden bit z páru popisuje existenciu hrany (i,k), druhý (k,j). Blok '11' nám teda hovorí o existencii cesty dĺžky 2 medzi týmito dvoma vrcholmi.

Vytvorili sme popis grafu G na

$$\mathrm{O(\log n)} + \binom{\mathrm{n}}{2} - 2(\mathrm{n}-2) + \mathrm{C(e_{i,j}|n)}$$

bitov, čo musí byť samozrejme viac ako je KZ grafu G, ktorý je o(n)-náhodný.

$$\mathrm{O(\log n)} + \binom{\mathrm{n}}{2} - 2(\mathrm{n}-2) + \mathrm{C(e_{i,j}|n)} \geq \binom{\mathrm{n}}{2} - \mathrm{o(n)}$$

$$C(e_{i,j}|n) \geq l(e_{i,j}) - \mathrm{o(n)}, \tag{3.53}$$

lebo vieme, že $l(e_{i,j}) = 2(n-2)$.
Potom z vety 3.3.3 dostávame

$$\left|\#'11'(e_{i,j}) - \frac{n}{4}\right| \leq \sqrt{\alpha\frac{n}{4}}. \tag{3.54}$$

Po dosadení za α dostávame jednoduché o(n):

$$\left|\#'11'(e_{i,j}) - \frac{n}{4}\right| \leq \sqrt{\mathrm{o(n)}\frac{\mathrm{n}}{4}}, \tag{3.55}$$

z čoho napokon dostávame frekvenciu výskytu 2-bitového bloku '11', ktorá sa z konštrukcie rovná počtu disjunktných ciest dĺžky 2 medzi i a j, rovnú

$$\frac{n}{4} + \mathrm{o(n)}. \tag{3.56}$$

No a keďže nám toto hovorí o výskyte cesty dĺžky dva medzi každými dvoma vrcholmi, vidíme, že každé dva vrcholy sú spojené cestou dĺžky dva, takže každý o(n)-náhodný graf má diameter 2. □

3.3.2 Štatistiky podgrafov

Pokračujeme definíciou ohodnoteného podgrafu ohodnoteného grafu.

Definícia 3.3.4. Nech $G = (V, E)$ je ohodnotený graf s n vrcholmi. Uvažujme ohodnotený graf H s k vrcholmi $\{1, 2, \ldots, k\}$. Každá k-prvková podmnožina množiny V grafu G indukuje podgraf G_k grafu G. Podgraf G_k je usporiadaným ohodnoteným výskytom H, ak dostaneme H zmenou hodnôt vrcholov G_k $i_1 < i_2 < \ldots < i_k$ na $1, 2, \ldots, k$.

Z vety 3.3.3 si ľahko vieme odvodiť frekvenciu výskytu ohodnotených dvojvrcholových podgrafov. Tie existujú totiž len dva

(1) podgraf pozostávajúci z dvoch izolovaných vrcholov

(2) podgraf pozostávajúci z dvoch vrcholov prepojených hranou

Všimnime si, že každý bit kódovania E(G) grafu G hovorí o existencii takéhoto podgrafu z 2 vrcholov. Každý bit totiž z konštrukcie hovorí o existencii hrany medzi dvomi vrcholmi, preto ak sa bit rovná 1, tak máme podgraf (2), ak sa rovná 0, existuje podgraf (1). Preto do vzorca na frekvenciu výskytu ohodnotených dvojvrcholových podgrafov z vety 3.3.3 za p dosadíme $1/2^1$,

lebo dĺžka podreťazca, ktorého výskyt zisťujeme je $l = 1$, dĺžka zápisu E(G) je $\binom{n}{2} = \frac{n!}{2(n-2)!} = \frac{n(n-1)}{2}$. Dostávame

$$\frac{n(n-1)}{4} \pm \sqrt{\frac{3n(n-1)}{4\log e}(\delta(n) + \mathrm{O}(1))} \tag{3.57}$$

ako frekvenciu výskytu ohodnoteného podgrafu s dvoma vrcholmi. Avšak určenie frekvencie výskytu ohodnotených podgrafov s k vrcholmi pre $k > 2$ je komplikovanejšie než ako hľadať frekvenciu výskytu k-bitového podreťazca. Je to preto, lebo všetkých $\binom{n}{k}$ k-prvkových podmnožín množiny vrcholov grafu sa môže prekrývať aj inak ako sa môžu prekrývať podreťazce.

Nech $\#H(G)$ je počet výskytov H ako ohodnoteného podgrafu G, pričom tieto výskyty sa môžu navzájom rôznymi spôsobmi prekrývať. Nech p je pravdepodobnosť, s akou zvolíme práve H spomedzi všetkých $2^{\binom{k}{2}}$ k-prvkových podmnožín množiny vrcholov grafu G. Takže

$$p = \frac{1}{2^{\binom{k}{2}}} = 2^{\frac{-k(k-1)}{2}} \tag{3.58}$$

Pred nasledujúcou vetou ešte zadefinujem pojem *vrcholové pokrytie* potrebný v dôkaze.

Definícia 3.3.5. Vrchlové pokrytie grafu G je množina $C = \{S_1, S_2, \ldots, S_N\}$, kde $N = \dfrac{n}{k}$ a S_i sú navzájom disjunktné podmnožiny V a platí $\bigcup_{i=1}^{N} S_i = V$.

Veta 3.3.6. *Berúc do úvahy práve spomenutú terminológiu, nech $G = (V, E)$ je graf s n vrcholmi, k je kladné celé číslo, pričom n delí k, H je ohodnotený graf s $k \leq \sqrt{2\log n}$ vrcholmi. Nech*

$$C(E(G)|n) \geq \binom{n}{2} - \delta(n). \tag{3.59}$$

Potom

$$\left|\#H(G) - p\binom{n}{k}\right| \leq \binom{n}{k}\sqrt{\alpha p\frac{k}{n}}, \tag{3.60}$$

kde $\alpha := \dfrac{3}{loge}\left(K(H|n) + \log\dfrac{k}{n}\binom{n}{k} + \delta(n) + \mathrm{O}(1)\right)$.

Dôkaz. Podľa [2] vieme rozdeliť $\binom{n}{k}$ rôznych k-vrcholových podmnožín do $h = \frac{\binom{n}{k}}{N}$ rôznych pokrytí G, pričom každé pokrytie bude obsahovať

$N = \frac{n}{k}$ disjunktných podmnožín. Každá k-vrcholová podmnožina V patrí do práve jedného pokrytia. Vymenujme tieto pokrytia ako $C_0, C_1, \ldots, C_{h-1}$. Nech pre každé $i \in \{0, 1, \ldots, h-1\}$ a každý k-vrcholový ohodnotený graf H je $\#H(G, i)$ počet (teraz už neprekrývajúcich sa) výskytov podgrafu H v pokrytí C_i.

Teraz uvažujme experiment, počas ktorého spravíme N pokusov, pričom každý z nich môže dopadnúť $2^{\frac{k(k-1)}{2}}$ spôsobmi. Intuitívne každý pokus reprezentuje prvok pokrytia a každý výsledok pokusu reprezentuje k-vrcholový podgraf. Pre každé i vytvorme reťazec s_i obsahujúci N blokov po $\binom{k}{2}$ bitov, pričom bloky reprezentujú jednotlivé prvky pokrytia a bity hovoria o prítomnosti hrán vnútri indukovaných podgrafov každej z N podmnožín C_i. Každé s_i je teda kódovaním pokrytia C_i.

Vidíme, že graf G môžeme získať pomocou n, i, s_i a na zistenie zvyšných hrán neuvedených v s_i potrebujeme zostávajúcich $\binom{n}{2} - N\binom{k}{2}$ bitov $E(G)$. Keďže tento popis opäť nemôže byť kratší ako $C(E(G)|n)$, platí

$$\log n + \log h + C(s_i|n) + \binom{n}{2} - N\binom{k}{2} \geq \binom{n}{2} - \delta(n) \tag{3.61}$$

Uvedomujúc si, že $l(s_i) = N\binom{k}{2}$ dostávame

$$C(s_i|n) \geq l(s_i) - \log h - \delta(n) \tag{3.62}$$

Frekvenciu výskytu $\binom{k}{2}$-bitového bloku $E(H)$ medzi N blokmi dĺžky $\binom{k}{2}$ nám určuje veta 3.3.3 ako

$$\#H(G, i) = Np \pm \sqrt{Np\alpha} \tag{3.63}$$

Takto to možno spraviť nezávisle pre každé i nehľadiac na závislosť medzi frekvenciami podgrafov v rôznych pokrytiach. Totiž, tento argument záleží jedine od nestlačiteľnosti G. Ak by množstvo výskytov nejakého podgrafu v ľubovoľnom pokrytí bolo privysoké alebo prinízke, vedeli by sme stlačiť G. Všetkých k-prvkových podmnožín n je $\binom{n}{k}$.
Takže

$$\left|\#H(G) - p\binom{n}{k}\right| = \sum_{i=0}^{h-1} |\#H(G, i) - Np|, \tag{3.64}$$

čo sa po dosadení z (3.63) sa to rovná

$$\sum_{i=0}^{h-1} \sqrt{Np\alpha}.$$

Z toho dostávame horný odhad

$$\left|\#H(G) - p\binom{n}{k}\right| \leq h\sqrt{Np\alpha} = \frac{\binom{n}{k}}{N}\sqrt{Np\alpha} = \frac{\binom{n}{k}}{\sqrt{N}}\sqrt{p\alpha}$$

Po dosadení $N = \frac{n}{k}$

$$\left|\#H(G) - p\binom{n}{k}\right| \leq \binom{n}{k}\sqrt{\frac{k}{n}}\sqrt{p\alpha} = \binom{n}{k}\sqrt{\alpha p\frac{k}{n}} \tag{3.65}$$

□

Kapitola 4

Záver

Kvalitná literatúra o Kolmogorovskej zložitosti v slovenčine prakticky neexistuje, čo bolo aj výraznou motiváciou pre vznik tejto práce. Snahou bolo napísať študijný text v slovenskom jazyku formulovaný tak, aby bol zrozumiteľný pre čitateľa so základnými vedomosťami z teoretickej informatiky, avšak aby tým neutrpela exaktnosť a potrebný stupeň formálnosti textu.

Úvodná kapitola ponúka určitý základ potrebný na pochopenie textu a načrtáva motiváciu vzniku popisovanej metódy.

Druhá kapitola zavádza a vysvetľuje samotný pojem Kolmogorovská zložitosť reťazca ako dĺžku najkratšieho programu generujúceho daný reťazec. Dopracuváva sa k pojmu nestlačiteľnosti reťazca, tj. vlastnosti reťazca, ktorá hovorí o tom, že jeho Kolmogorovská zložitosť je väčšia než jeho dĺžka.

Prvé dve kapitoly budujú aparát pre vysvetlenie tzv. metódy nestlačiteľnosti, ktorá je témou najdlhšej, tretej kapitoly. Tu je ukázané jej využitie v mnohých dôkazoch z rôznych oblastí. Taktiež sú tu ukázané dôkazy získané prvýkrát práve metódou nestlačiteľnosti, pričom sa dokážu problémy, ktorý boli otvorené desiatky rokov. Hlavným zameraním práce sú práve detailné ukážky využitia metódy neslačiteľnosti.

Práca poskytuje možnosti na jej rozšírenie v rámci diplomovej práce. Z každej oblasti môže byť uvedených viac príkladov dôkazov pomocou popisovanej metódy. Taktiež možno rozšíriť záber práce o iné, tiež zaujímavé zamerania matematiky a informatiky. Je možné ukázať ešte viac vlastností Kolmogorovskej zložitosti. Taktiež je možná prípadná implementácia niektorých algoritmov.

Problematika bola aj pre mňa úplne nová, preto som sa najskôr musel venovať jej štúdiu. Taktiež v oblastiach, v ktorých som ukazoval využitie metódy nestlačiteľnosti som nadobudol nové vedomosti, resp. som prehĺbil či utvrdil moje doterajšie znalosti. Téma ma zaujala a teším sa, že som v rámci svojej bakalárskej práce mohol písať práve o tomto. Pozitívnym vedľajším efektom písania práce bolo aj to, že som sa naučil pracovať v TEX-u, čo určite v budúcnosti využijem.

Jednoznačným prínosom v porovnaní s ostatnou cudzojazyčnou literatúrou je i datailnosť výkladu a snaha o jeho zápis v ľahko stráviteľnej forme, čo inej literatúre zaoberajúcej sa danou problematikou bohužiaľ chýba. Preto dúfam, že tento text ocenia všetci, ktorí chcú získať základné vedomosti o uvedenej problematike.

Literatúra

[1] Algoritmy a dátové štruktúry, skriptá.

[2] Zs. Baranyai. On the factorization of the complete uniform hypergraph. *Infinite and Finite Sets*, 10:91–108, 1995.

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

[4] W. Dobosiewicz. An efficient variant of bubble sort. *Information Processing Letters*, (11):5–6, 1980.

[5] Michal Forišek. Formálne jazyky a automaty, skriptá, 2007.

[6] J. Incerpi and R. Sedgewick. Practical variations of shellsort. *Information Processing Letters*, pages 37–43, 1980.

[7] Tao Jiang and Ming Li. k one-way heads cannot do string-matching. *Journal of Computer and System Sciences*, 53(3):513–524, 1996.

[8] D. E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, second edition, 1998.

[9] Vasileios Megalooikonomou. Kolmogorov incompressibility method in formal proofs, a critical survey, 1997.

[10] Harry Buhrman Ming Li, John Tromp and Paul Vitányi. Kolmogorov random graphs and the incompressibility method. In *Conference on Compression and Complexity of Sequences*, 1997.

[11] M. Geréb-Graus Ming Li. Three one-way heads cannot to string-matching. *Journal of Computer and System Sciences*, 47(1):1–8, 1994.

[12] Ming Li Paul Vitányi, Tao Jiang. Average-case analysis of algorithms using kolmogorov complexity, 1999.

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

[14] R. Sedgewick. Analysis of shellsort and related algorithms. In *Fourth Annual European Symposium on Algorithms*, 1996.

[15] Paul Vitányi. Analysis of sorting algorithms by kolmogorov complexity (a survey), 2000.

[16] Paul Vitányi and Ming Li. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, second edition, 1993.

[17] Paul M. B. Vitányi. personal communication, 2008.

[18] Paul M. B. Vitányi and Ming Li. Statistical properties of finite sequences with high kolmogorov complexity. *Math. System Theory*, 27:365–376, 1994.

[19] M. A. Weiss and R. Sedgewick. Bad cases for shaker-sort. *Information Processing Letters*, pages 133–136, 1988.