Algoritmus Re–Pair

Moravec Vojtěch

LS 2020

1 Popis algoritmu Re–Pair

Algoritmus Re–Pair, poprvé představen v článku [1], je kompresní algoritmus založený na využití bezkontextové gramatiky. Na vstupu dostává tento algoritmus řetězec znaků, např. text, který převede na řetězec terminálních a neterminálních symbolů bezkontextové gramatiky. Autoři zařadili tento algoritmus do skupiny tzv. *off-line* slovníkových kompresních metod.

Off-line metody využívají celého vstupního řetězce k vytvoření slovníku. Tyto metody tedy musí načíst celý vstupní soubor nebo jeho část do paměti. V minulosti nebyly příliš využívány kvůli limitované paměti počítače, která avšak v současnosti stále roste. Navíc je rychlost komprese *off-line* metod vyšší, než je tomu u druhé skupiny. Uvedli jsme, že může být také načtena pouze část souboru, avšak tato část je stále několikrát větší, než je tomu u *on-line* metod, které většinou využívají plovoucí okno. Algoritmy v *on-line* skupině, tvoří slovník pomocí té části vstupu, kterou již přečetli, jak je tomu například u slovníkových kompresí LZ.

Výhodou *off-line* metod je možnost vytvořit slovníkové objekty, které povedou k lepším kompresním výsledkům. Toto je dáno tím, že díky znalosti větší části textu víme víc o samotném kontextu. Oproti tomu nevýhodou je velká paměťová náročnost, která nedovoluje použití těchto algoritmů na velké soubory.

V tomto dokumentu budeme popisovat kompresi textu a očekáváme od čtenáře základní znalost bezkontextové gramatiky. Algoritmy využívající bezkontextové gramatiky převádějí vstupní text na sekvenci terminálních a neterminálních symbolů. Aby mohli provést tuto transformaci musí nalézt přepisovací pravidla, které budou vést k maximální kompresi. Ukázku, jak můžeme transformovat text `abrakadabra` pomocí bezkontextové gramatiky nalezneme v Tabulce 2. Záměrně jsme využili přepisovací pravidla, které mají na pravé straně pouze dva symboly, jak je tomu u algoritmu Re–Pair.

Text	Pravidla
`abrakadabra`	
`ArakadAra`	$A \to ab$
`BakadBa`	$B \to Ar$
`CkadC`	$C \to Ba$

Tabulka 1: Ukázka přepsání textu podle bezkontextové gramatiky

Je tedy zřejmé, že abeceda vzniklá na konci komprese, bude oproti originálním znakům ze vstupního textu, obsahovat také nové neterminální symboly. Výsledkem algoritmu je tedy řetězec symbolů a množina přepisovacích pravidel. Obě tyto informace musí být uloženy v komprimovaném souboru, nebo přeneseny přes síť. K zakódování se většinou využívá různých entropických metod, generující kódy proměnlivé délky, jako je například Huffmanovo kódování nebo Aritmetické kódování.

Samotná název Re–Pair je zkratkou anglického *Recursive Pairing*, který už sám o sobě

napovídá, že se bude jednat o metodu využívající určité rekurze a párování. Originální algoritmus vždy nahrazuje nejčetnější pár v textu. Párem rozumíme dvojici terminálních nebo neterminálních symbolů. Označíme-li pomocí malých písmen terminální symboly a pomocí velkých písmen neterminální symboly, tak mohou nastat čtyři situace:

- ab
- AB
- aB
- Bc

Algoritmus nijak neupřednostňuje žádnou z nich. Nahrazování probíhá do té doby, dokud se v nahrazeném textu nachází určitý pár alespoň dvakrát. Nutno podotknout, že algoritmus Re–Pair při běhu neprovádí žádné úplné substituce v textu. Substituce by pouze vedli ke zpomalení. Dále bude popsáno, jak dokáže algoritmus postupně redukovat text a přitom vědět, v jakém stavu se text zrovna nachází.

Obecně se dá celý algoritmus popsat tímto velmi zjednodušeným pseudokódem 1. Četností páru, je přímo myšleno počet výskytu daného páru v textu. Během nahrazování může nastat situace, že více párů má maximální četnost. V originální práci uvedli, že volba, který z těchto páru bude nahrazen jako první, hraje minimální roli vzhledem k výsledku komprese. Jednou ze strategií, jak řešit tento problém, je například zvolit pár, který byl zatím nejméně krát použit.

Algorithm 1 Základní princip nahrazování párů

1. Nalezni symboly c a d, takové, že cd je nejčetnější pár v celém textu
2. Vytvoř nový (neterminální) symbol A a nahraď všechny páry cd symbolem A
3. Jestliže existuje nějaký pár s četností alespoň 2 pokračuj 1. krokem

1.1 Implementační detaily

Návrh implementace byla velmi důležitá část originální práce, neboť zde popisovaná komprese se dá řešit několika způsoby, ale dosáhnout lineární časové složitosti $O(n)$ není tak jednoduché. Základem implementace jsou tři různé datové struktury.

1. Sekvence symbolů w
    - implementována jako pole symbolů
    - na začátku je toto pole naplněno vstupním textem
    - každý prvek pole $w[i]$ obsahuje strukturu obsahující tři čísla
        - číslo C – reprezentující symbol,

  - číslo n – kde $n > i$. Následující pozice v sekvenci, na které začíná stejný pár jako na pozici i,
  - číslo p – kde $p < i$. Předešlá pozice v sekvenci, na které začíná stejný pár jako na pozici i.

2. Hashovací tabulka
   - Obsahuje všechny aktivní páry, které mohou být nahrazeny
3. Prioritní fronta
   - implementována jako pole cca $\sqrt{n}$ seznamů, které jsou propojeny
   - na začátku fronty je seznam s páry, maximální četnosti
   - každý další seznam obsahuje páry s menší četností než předcházející seznam

Čísla n a p slouží jako ukazatele na další výskyty páru. Tyto ukazatele nám dovolují pracovat se Sekvencí jako s obousměrně spojovým seznamem. Navíc ukazatelé také slouží k orientaci v "upravovaném"textu. Během běhu algoritmu vznikají v textu tzv. prázdná místa, které jsou těmito ukazateli automaticky přeskakována. Toto se bude hodit při samotném nahrazování. Obě datové struktury, hashovací tabulka i prioritní fronta potřebují stejnou informaci o páru, a proto je tento pár většinou implementován pomocí struktury *Pair*, která obsahuje následující informace.

- *left* – levý symbol páru,
- *right* – pravý symbol páru,
- *frequency* – četnost výskytu,
- *hash_next* – ukazatel na další *Pair* se stejným hashem,
- *queue_next* – ukazatel na další *Pair* se stejnou četností,
- *queue_previous* – ukazatel na předcházející *Pair* se stejnou četností,
- *first_position* – pozice prvního výskytu páru v Sekvenci,
- *last_position* – pozice posledního výskytu páru v Sekvenci.

Inicializace algoritmu probíhá následovně. Do sekvence se naplní vstupní text a lineárním skenem se naleznou všechny páry s jejich četností výskytu. Dle četností je naplněna prioritní fronta a páry jsou vloženy do hashovací tabulky. Složitost inicializace je tedy $O(n)$.

Samotné nahrazení, které bylo druhým krokem v Algoritmu 1, teď popíšeme v Algoritmu 2.

Algorithm 2 Nahrazení páru

1. Najdi první nebo další výskyt nahrazovaného páru cd. Identifikuj sousedící symboly x a y, které dohromady tvoří $xcdy$.
2. Sniž četnost párů xc a dy. Jestliže některý z těchto dvou párů dosáhne četnosti jedna nebo menší, smaž jej z prioritní fronty.
3. Nahraď cd novým symbolem A, z řetězce $xcdy$ dostaneme xAy.
4. Zvyš četnost párů xA a Ay. Pokud pár ještě neexistuje tak jej vytvoř, vlož do fronty a hashovací tabulky.

Jelikož při každém nahrazení se délka textu zmenšuje, je celkový počet nahrazení $O(n)$. Všechny operace uvedeny v Algoritmu 2 mohou být provedeny v konstantním čase, díky datovým strukturám, které jsme uvedli dříve. První a třetí krok algoritmu jsou provedeny v konstantním čase pomocí ukazatelů n a p v Sekvenci. Zbylé dva kroky modifikují četnosti párů, a je tedy nutná aktualizace prioritní fronty. Díky tomu, že fronta je implementována pomocí spojových seznamů je přesun ze seznamu do méně prioritního seznamu proveden v konstantním čase. Můžeme si všimnout, že četnost aktivních párů nikdy neroste. Četnost pouze klesá, když je jeden ze symbolů páru absorbován nahrazením sousedního páru. Podobně, četnost nově vytvořených párů není nikdy větší než četnost nahrazeného páru. Celkově četnost tedy postupně klesá.

Při nahrazování je třeba si dát pozor na sekvence opakujících se symbolů, jako je např. $aaaa$, tento řetězec by měl být nahrazen jen dvěma novými symboly a ne třemi.

1.2 Varianty algoritmu

V průběhu let vzniklo několik vylepšení základního algoritmu, kde každé se snažilo zlepšit jiný aspekt, ať už se jedná o paměťovou složitost nebo menší kompresní poměr.

V práci [2] se setkáváme s technikou tvoření slovníku z celých slov, raději než jednotlivých symbolů. Na pravá straně přepisovacích pravidel tedy nalezneme dvě slova místo dvou symbolů. Tento upravený algorimus dále nabízí možnost upravit definici slova.

Dále v [3] se setkáváme s úpravou finálního kódování, které se používá pro kompresi vygenerovaných přepisovacích pravidel. V originálním algoritmu jsou pravidla kódována pomocí kódů proměnné délky, kdežto zde se rozhodli využít kódu pevné délky. Tato úprava vedla k vylepšení finálních kompresního poměru [3].

Podobně jako v [2] se v článku [4] setkáváme s úpravou dovolující nahrazovat více než jen dva symboly. V tomto článku využívají tzv. *Maximal Repeats*. Při nahrazování nejčetnějšího páru se snažíme tento pár co nejvíce zvětšit. Například pro náš text v Tabulce 2 by hned první pravidlo mělo na pravé straně řetězec `abra`, který se dá získat rozšířením páru `ab`. V článku je dále uvedeno, že tato technika se vyplácí hlavně na texty, obsahující často opakující se sekvence.

Poslední variantou, kterou si zde uvedeme je *Space–Efficient Re–Pair*, představen v [5]. Jak už název napovídá, tento algoritmus se snaží vylepšit hlavně paměťovou náročnost originálního algoritmu, která byla $5n + 4\sigma^2 + 4d + \sqrt{n}$. Kde n značí délku textu, σ ve-

likost abecedy a d počet neterminálních symbolů (velikost vygenerované abecedy). V této práci byly představeny celkem dva algoritmy, kde časová složitost druhého je $O(n \log n)$ a vylepšené nároky na paměť jsou $n + \sqrt{n}$. Vylepšeních bylo dosaženo pomocí rozdělení algoritmu do dvou fází. První fáze, nahrazuje páry s vysokou četností. To jsou ty, které se vyskytují alespoň $\sqrt{n}/3$ krát. Druhá fáze následně zpracována zbylé páry s menší četností. Obě tyto fáze se liší implementací prioritní fronty, kterou využívají.

2 Testování komprese

V této sekci bychom rádi prezentovali data, na kterých budeme zkoušet kompresi, spolu s výsledky, kterých jsme dosáhli. Budeme zde uvádět kompresní poměr, který počítáme podle Rovnice 1. Zároveň budeme uvádět BPS neboli počet bitů na symbol. Symbolem budeme rozumět jeden ASCII znak neboli byte. Toto si můžeme dovolit, neboť všechny testované soubory jsou v anglickém jazyce a nemusíme tedy počítat se speciálními symbole jako je například diakritika.

$$\mathrm{CR} = \frac{\text{Počet bytů po kompresi}}{\text{Počet bytů před kompresí}} \tag{1}$$

Název	Typ souboru	Velikost (B)
bible.txt	Anglický text	4047392
dblp.xml.100MB	XML dokument	104857600
english.100MB	Anglický text	104857600
sources.100MB	Kód v jazyce C a Java	104857600

Tabulka 2: Testované soubory

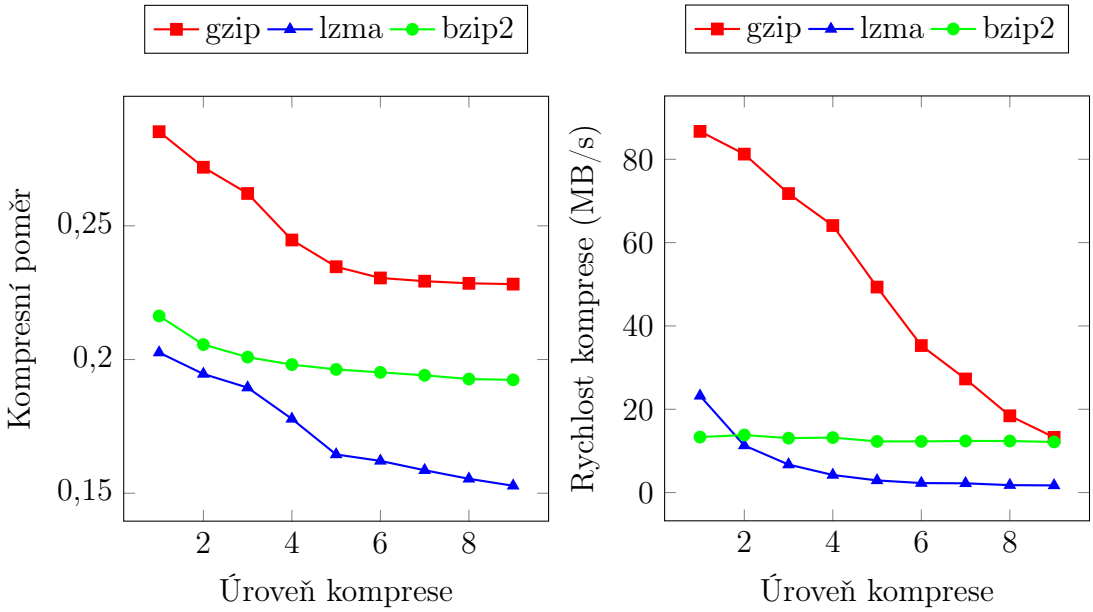
Všechny testované soubory jsou součástí standardních korpusu pro testování bezztrátové komprese textu. První soubor bible.txt je ještě ze starší doby, proto je jeho velikost menší a je k dispozici na `http://corpus.canterbury.ac.nz/index.html`. Zbylé tři dokumenty jsou součástí *Pizza & Chili* korpusu na webových stránkách `http://pizzachili.dcc.uchile.cl/texts.html`.

Co se týče algoritmů, se kterými budeme srovnávat Re–Pair tak to budou:

1. gzip
2. lzma
3. bzip2

Gzip je založen na metodě DEFLATE [6], která kombinuje dva různé algoritmy, LZ77 [7] a Huffmanovo kódování. Další zvolenou metodou komprese je algoritmus LZMA [8], který znovu kombinuje více algoritmů. Součástí LZMA najdeme algoritmy LZ77, Markův model predikce a aritmetické kódování, přesněji *Range Coding*. Třetí metodou je algoritmus bzip2 [9], který podobně jako LZMA kombinuje více metod komprese a současně přidává několik transformací. Pomocí těchto transformací se snaží vylepšit výslednou kompresi. Detailní vysvětlení uvedených algoritmů je nad rámec této práce, zde zkoumáme pouze jejich výsledky. Čtenáři, doporučujeme následovat zdroje uvedené v literatuře.

U všech tří těchto algoritmů můžeme nastavit tzv. úroveň komprese neboli číslo od 0 do 9. Nultá úroveň znamená žádnou kompresi a 9. úroveň naopak maximální kompresi. Toto se u algoritmu Re–Pair nastavit nedá.



(a) Kompresní poměr vzhledem k úrovni (b) Ča komprese vzhledem k úrovni

Obrázek 1: Vliv úrovně komprese na kompresní poměr a rychlost, soubor sources.100MB

Pro ilustraci se můžeme podívat na grafy v Obrázku 1, kde nalezneme kompresní poměr a rychlost komprese vzhledem k úrovni komprese pro soubor sources.100MB. Z těchto grafů vyčteme, že algoritmus lzma dosahuje nejlepších kompresních výsledků, ale za to je taky nejpomalejší. Nejrychlejší je gzip, který je jednoduchým algoritmem, a proto taky dosahuje nejhorších výsledků v rámci komprese.

V následující Tabulce 3 jsou již uvedeny výsledky pro všechny testované algoritmy a soubory. Algoritmem RP značíme originální algoritmus Re–Pair a pod RP–SE najdeme upravenou variantu představenou v [5]. Testování probíhalo na počítači s procesorem Intel Core i7-8750H bez využití více jader procesoru.

Podíváme-li se na kompresní poměry, tak si všimneme, že algoritmy Re–Pair dokážou dosáhnout podobných výsledků jako algoritmy lzma a bzip2. Algoritmus gzip je ze všech nejhorší. Celkově nejlepších výsledku na těchto textových souborech dosahuje algoritmus lzma, který se u XML souboru dokázal dostat i pod hranici 1 BPS. Dále si všimneme, že vylepšený algoritmus Re–Pair dosahuje obecně menších kompresních poměrů než originální algoritmus Re–Pair, ale za cenu mnohem pomalejší komprese. Obecně jsou Re–Pair algoritmy řádově pomalejší nežli zbylé. Toto můžeme nejspíš připisovat poněkud horší implementaci než u zbylých algoritmů, které se standardně používají v programech jako je např. 7-zip. Algoritmy Re–Pair nalezneme spíše v odborných pracích a pokusech.

Soubor	Algoritmus	CR	BPS	Rychlost (kB/s)
bible.txt	gzip	0,2909	2,3272	11142,8690
	lzma	0,2187	1,7496	2174,4037
	bzip2	0,2089	1,6712	13220,2216
	RP	0,2604	2,0832	5899,9880
	RP–SE	0,2426	1,9408	381,9375
dblp.xml.100MB	gzip	0,1686	1,3488	34997,8193
	lzma	0,1103	0,8824	2475,6255
	bzip2	0,1110	0,8880	9917,5981
	RP	0,1289	1,0312	6093,5380
	RP–SE	0,1219	0,9752	211,3574
english.100MB	gzip	0,3764	3,0112	12649,8978
	lzma	0,2158	1,7264	1162,6733
	bzip2	0,2807	2,2456	12178,6030
	RP	0,2789	2,2312	3759,9540
	RP–SE	0,2476	1,9808	64,0705
sources.100MB	gzip	0,2282	1,8256	12851,3163
	lzma	0,1528	1,2224	1824,9400
	bzip2	0,1924	1,5392	12326,4306
	RP	0,2321	1,8568	3872,5710
	RP–SE	0,2212	1,7696	76,5095

Tabulka 3: Výsledky komprese

Reference

[1] N. J. Larsson and A. Moffat, "Off-line dictionary-based compression," *Proceedings of the IEEE*, vol. 88, no. 11, pp. 1722–1732, 2000.

[2] R. Wan, *Browsing and searching compressed documents*. PhD thesis, 2003.

[3] S. Yoshida and T. Kida, "Effective variable-length-to-fixed-length coding via a re-pair algorithm," in *2013 Data Compression Conference*, pp. 532–532, IEEE, 2013.

[4] I. Furuya, T. Takagi, Y. Nakashima, S. Inenaga, H. Bannai, and T. Kida, "Mr-repair: Grammar compression based on maximal repeats," in *2019 Data Compression Conference (DCC)*, pp. 508–517, IEEE, 2019.

[5] P. Bille, I. L. Gørtz, and N. Prezza, "Space-efficient re-pair compression," in *2017 Data Compression Conference (DCC)*, pp. 171–180, IEEE, 2017.

[6] Aladdin Enterprises, *DEFLATE Compressed Data Format Specification version 1.3*, May 1996.

[7] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, pp. 337–343, May 1977.

[8] I. Pavlov, "7-zip, lzma," 3 2019.

[9] F. Mena, "bzip2," 3 2019.