

2015

VŠB - TUO

Zdeněk Gold

[KOMPRIMAČNÍ ALGORITMUS RE-PAIR]

Popis algoritmu Re-Pair spolu s testováním a představením, jak algoritmus funguje

Obsah

Algoritmus Re-Pair	3
Analýza algoritmu	3
Implementace.....	4
Testy.....	5
Bibliografie	6

Algoritmus Re-Pair

Algoritmus funguje tak, že opakovaně **hledá nejfrekventovanější páry znaků** v sekvenci a nahrazuje je novými znaky, dokud není v sekvenci každý pár znaků pouze jednou.

1. Identifikuj **nejfrekventovanější pár** symbolů $ab \in T$
2. **Přidej pravidlo** $R \rightarrow ab$ do slovníku, kde R je nový symbol, který není obsažen v T
3. **Nahraď** každý výskyt ab za symbol T
4. **Opakuj celý postup** od 1. bodu, dokud se nebudou všechny dvojice znaků v textu jen jednou

Algoritmus **Re-Pair** může být implementován s **lineární časovou a prostorovou složitostí**. Tento základní algoritmus může být vylepšený různými technikami, jako je například **komprese přepisovacích pravidel**.

Analýza algoritmu

Na začátku mějme **nekomprimovaný text** T obsahující n znaků s abecedou obsahující o znaků, takže $o \leq n$. Algoritmus potom **prochází postupně všechny dvojice** znaků a při každém průchodu celou zprávou vytváří jedno přepisovací pravidlo do slovníku. Definujme si aktuální stav textu během každého kroku komprimace jako $C = c_1, c_2, \dots, c_p$, kde p je počet znaků a d označující velikost slovníku (počet přepisovacích pravidel).

Na začátku algoritmu máme tuto konfiguraci: $C = T$, $p = n$, $d = 0$. Při každém kroku algoritmu se inkrementuje d o 1 a p se sníží nejméně o 2. d zároveň označuje aktuální číslo iterace.

Definujme si funkci $expand(c_i)$, která pro daný symbol vrátí buď stejný symbol, pokud je terminální nebo sekvenci symbolů, které reprezentuje T .

Počet různých symbolů v sekvenci po d krocích je nejvýše $o + d$, a proto budeme potřebovat $\log_2(o + d)$ bitů pro reprezentaci každého symbolu. Pro jednoduchost budeme pesimističtí a budeme počítat s tím, že buňky slovníku budou pro reprezentaci znaku potřebovat $\log_2 n$ bitů.

Lemma 1: V jakémkoliv kroku procesu se bude platit $o + d \leq n$.

Ukážeme si, že nejméně d opakujících se symbolů je identifikováno v T , když tvoříme slovník velikosti d .

Lemma 2: Velikost komprimovaných dat je $p + 2d$ celých čísel, které ne během procesu nezvyšuje.

Lemma 3: Četnost nejfrekventovanějších dvojic symbolů se nezvyšuje oproti předchozímu nejčtetnějšímu páru.

Implementace

Algoritmus byl implementován tak, aby byl obecně využitelný na jakýkoliv obsah (text, data, obrázky, videa, ...). Pro tento požadavek byla reprezentace znaku v podobě bytu (jeho hodnoty). Protože všechen elektronický obsah, lze reprezentovat proudem bytů.

V první fázi algoritmu se vezme pole bytů a uloží se do pomocného pole stejné velikosti. K tomuto poli se potom vytvoří druhé pomocné pole stejné velikosti, které reprezentuje buffer. Důvodem je rychlost algoritmu, kterému stačí vyblokovat pouze 2x velikost komprimovaného souboru za celý průběh komprimace a pouze se technikou double bufferingů využívá jeden či druhý buffer pro zápis nových hodnot po nahrazení.

Časově nejnáročnější fází algoritmu je **analýza všech dvojíc bytů** a nalezení nejčtenější dvojice v proudu bytů. Nejrychlejší možný způsob jak toho dosáhnout je posouvání indexu od začátku do konce pole bytů a inkrementace výskytu každé takové dvojice za indexem.

Pro **uchování tabulky výskytů** můžeme využít slovníkové struktury v jazyce Java, kde nalezení hodnoty klíče a vložení do hashmapy má složitost v nejhorším případě $O(n)$, průměrně $O(1)$. Pro urychlení algoritmu jsem použil **vyalokované pole integerů**, velikosti 2^{16} , kde lze adresovat všechny možné kombinace 2 bytové hodnoty. Přístup pro zápis i čtení je zde konstantní $O(1)$, i pro nejhorší časovou složitost. Nevýhodou je, že je třeba vyalokovat **256 kB paměti**.

Jelikož nám po komprimaci vznikne slovník s přesipovacími pravidly, je třeba tento slovník do výsledného souboru uložit. U mé implementace jsem zvolil uložení na začátek souboru. **První byte** reprezentuje **počet záznamů** slovníku a každý záznam obsahuje 1 **B znaku S** levé strany přepisovacího pravidla a 2 **B** pro znaky **ab pravé strany** pravidla. Přepisovací pravidlo je tedy v této podobě: **S** → **ab**.

Výsledný program je implementován v jazyce Java a je přiložen k tomuto dokumentu jako **Java** projekt ve vývojovém prostředí **Eclipse**.

Testy

Testování proběhlo se soubory v **adresáři data**, který je umístěn v projektu aplikace. V tabulce níže jsou uvedené některé z těchto souborů spolu s výsledky komprimace.

Vidíme, že u některých souborů jsme dosáhli **komprimaci dokonce až na 13%** původní velikosti souboru, což je pozoruhodné. Takové soubory obsahují sekvenci opakujících se znaků, která se neztrácí ani po několika iteracích nahrazení nebo má **dostatečně malou abecedu**.

Naproti tomu u **rozsáhlých souborů**, jako je například soubor *Android.pdf* a *BIBLE21.pdf* se komprimovat touto metodou původní soubor **nepodařilo**. Všímavého čtenáře jistě napadne, proč tomu tak bylo. Jelikož algoritmus pracuje s doplňkem množiny abecedy znaků (v našem případě bytů), která se vyskytuje v původním souboru, je pochopitelné, že u takto velkých souborů se využije téměř celá množina znaků abecedy (všechny kombinace hodnot bytů) a pro tvorbu přepisovacích pravidel proto nemáme dostatek znaků. Tuto skutečnost je uvidíme, podíváme-li se na sloupeček „**Počet nahrazení**“.

Název	Velikost (původní)	Velikost (po komprimaci)	Počet nahrazení	Čas komprimace
<i>Alice29.txt</i>	145 kB	50,52 %	181	370 ms
<i>Android.pdf</i>	61,8 MB	99,84 %	6	34 s
<i>asyoulik.txt</i>	122 kB	52,35 %	186	251 ms
<i>bib</i>	109 kB	49,33 %	173	207 ms
<i>BIBLE21.pdf</i>	10,9 MB	98,45 %	6	6 s
<i>Book1</i>	751 kB	53,53 %	172	285 ms
<i>Book2</i>	597 kB	56,50 %	158	908 ms
<i>cp.html</i>	24 kB	46,28 %	168	60 ms
<i>Fields.c</i>	11 kB	44,22 %	164	39 ms
<i>geo</i>	100 kB	76,47 %	6	35 ms
<i>Grammar.lsp</i>	3,6 kB	45,39 %	178	36 ms
<i>Kennedy.xls</i>	0,98 MB	56,03 %	6	148 ms
<i>Lcet10.txt</i>	409 kB	51,31 %	171	675 ms
<i>news</i>	368 kB	62,82 %	156	629 ms
<i>Obj1</i>	21 kB	80,48 %	6	8 ms
<i>Obj2</i>	241 kB	85,82 %	6	91 ms
<i>Paper1</i>	52 kB	55,78 %	159	209 ms
<i>Paper2</i>	80 kB	51,85 %	163	154 ms
<i>pic</i>	501 kB	13,30 %	102	157 ms

Přesto, že je tento algoritmus pro komprimaci **velice účinný**, bude jeho nevýhoda spočívat v relativně **velké časové náročnosti** při komprimaci velkých souborů. V tabulce vidíme, že například pro komprimaci **62 MB** souboru, běžel algoritmus **více jak 30s**. U takto velkých souborů by proto mohlo být řešením, kdyby při analýze všech dvojic symbolů v textu, který je časově nejnáročnější, **prošel** algoritmus **jen část textu** a vyhodnotil by nejlepší dvojici pro nahrazení z této části. Nedosáhli bychom tak dobrých výsledků pro komprimaci, ale v reálných situacích, kde je třeba rychlost, by to bylo nevyhnutelné.

Bibliografie

1. **Navarro, Gonzalo a Russo, Luís.** Re-Pair Achieves High-Order Entropy. *Departamento de Ciencias de la Computación, Universidad de Chile*. [Online] 2007. www.dcc.uchile.cl/TR/2007/TR_DCC-2007-012.pdf.
2. **Lohrey, Markus, Maneth, Sebastian a Mennicke, Roy.** *Cornell University Library*. [Online] 30. červenec 2010. arxiv.org/pdf/1007.5406.