TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Komprese měřených dat

v 0.1

Liberec 2007 **V**iktor Bubla

Obsah

1 Proč komprimace?

Nejzásadnějším důvodem, proč vůbec začali vznikat jakékoli komprimační algoritmy, je úspora množství prostor potřebných pro uchování dat a z toho plynoucí další důsledky, jako zvýšení rychlosti toku užitečné informace skrze přenosovou cestu s limitovanou přenosovou rychlostí (vždy), nebo třeba zvýšení využitelnosti úložných zařízení (větší kapacita). Pokaždé dojde ke snížení nákladů, což je vlastně nepřímým hlavním důvodem. Pochopitelně každé plus má i své minusy. Můžeme hovořit o vyšší složitosti softwaru a s tím úzce souvisejícími hardwarovými nároky či o tom, že data nemůžeme přímo číst, ale musíme předtím provést inverzní operaci atd.

2 Filosofie základních komprimačních algoritmů

2.1 Slovníkové

Slovníkové nebo také substituční kodéry pracují na principu hledání shodných řetězců mezi komprimovanými daty a datovou strukturou nazývanou slovník, kterou enkodér postupně upravuje. Ve chvíli, kdy je nalezena shodná část řetězce, je tento nahrazen referencí na svůj předchozí výskyt.

Pro vysoký stupeň komprese je třeba využívat velký slovník, díky kterému je komprese paměťově a výpočetně dosti náročná.

Slovníkové algoritmy jsou univerzálně použitelné a mají obzvláště dobré výsledky při kompresi dat, ve kterých se určité bloky často opakují (text).

Lempel-Ziv

LZ77 je jedním z představitelů slovníkových kompresních algoritmů. Řetězcová reference je zde typicky reprezentována unikátním markerem, offsetem a délkou substituovaného řetězce, přičemž délka reference je proměnná.

Nevýhodou je zdlouhavé porovnávání, kdy pro každý jednotlivý byte vstupních dat může každý předešlý byte být potenciálním počátkem pro porovnávání řetězce, což znamená, že komprese je velice pomalá.

Jako slovník je využíváno tzv. plovoucí okno, kde jsou uchována poslední načtená data tak, aby v nich bylo možné hledat shody řetězců.

Lempel-Ziv-Oberhumer

LZO je zásadně vylepšená varianta LZ. Zaměřuje se na rychlost komprese a extrémní rychlost dekomprese.

2.2 Statistické

Kompresní algoritmy v této kategorii vytvářejí (přesněji řečeno používají) tabulky znaků seřazené podle počtu výskytu ve vstupním souboru dat. Znaky jsou pomocí tabulky převáděny do binární formy, přičemž ty s největším počtem výskytů (nejpravděpodobnější) jsou substituovány nejmenším počtem bitů a ty nejméně četné největším počtem bitů. Tabulka se ukládá na začátek komprimovaného souboru. Podle konkrétní implementace jsou varianty, kde se tabulka (tzv. hufmannův strom) vytváří při prvním průchodu komprimovanými daty, kde je tabulka pevně daná a nemusí se tedy ukládat spolu se souborem, nebo kde je implicitní počáteční tabulka a během komprimace se dynamicky modifikuje.

Huffman

V prvním kroku jsou veškeré znaky seřazeny dle frekvence výskytu od nejčastějšího po nejneobvyklejší. Dále jsou vybrány dva znaky s nejmenší frekvencí výskytu, logicky seskupeny a jejich frekvence sečteny. Tím se započíná stavba tzv. „binárního stromu“. Opět jsou vybrány dva prvky s nejnižší frekvencí, přičemž v posledním kroku sečtená kombinace znaků je nyní považována taktéž za jeden element, jejich výskyty jsou sečteny atd. V poslední iteraci dostaneme jeden jediný kořen huffmanova stromu. Nyní se postupuje od kořene a každému rozvětvení je přidělena 1 a 0. Je dokonce jedno, jaká z dvojice větví dostane přidělenu kterou hodnotu, avšak používá se jeden konzistentní styl. Pokračujeme „do koruny stromu“ ve všech vetvích. Nyní již každému znaku odpovídá binární hodnota daná posloupností 1 nebo 0 od kořene stromu až k místu, kde končí větev s daným znakem. Je jasné, že znak s největší frekvencí výskytu má od kořene ke „své větvi“ nejkratší cestu, tedy nejméně bitové vyjádření.

Při kompresi se prochází zdrojová data a každý znak je konvertován do svého binárního obrazu daného huffmanovým stromem.

Shannon-Fano

Shannon-Fano algoritmus je ve své podstatě totožný s huffmanovým. Rozdíl je pouze ve způsobu výstavby binárního stromu. Zatímco Huffman ho staví odspodu nahoru, Shannon-Fano odshora dolu. Dle svých frekvencí výskytu seřazené znaky jsou vždy rozděleny na dvě poloviny tak, aby součty výskytů obou skupin byly co nejbližší. Stejně se pokračuje při dalším dělení. Přiřazení binárních hodnot jednotlivým znakům již probíhá naprosto totožně. Vytvořený strom je dokonce také shodný jako strom vytvořený huffmanovým algoritmem.

Komprese Shannon-Fano vznikla dříve a huffmanův způsob výstavby stromu byl pouze jejím vylepšením a zefektivněním.

2.3 Ostatní

RLE - Run Length encoding

Funkce jednoho z nejjednodušších algoritmů spočívá v nahražení za sebou se opakujícího symbolu escape sekvencí, počtem opakování vyjádřeným binárním číslem a opakujícím se symbolem.

```
Veta s opakujiciiiiiiiiiiiiiiim se i.
   Veta s opakujic<ESC><16>im se i.
```

Při kompresi textových souborů nemá RLE téměř význam používat, avšak často se využívá jako první stupeň složitých algoritmů složených z několika různých způsobů komprese, jako je například JPEG. Kupříkladu u monochromatického obrazového souboru je možné dosáhnout provedením RLE i o několik řádů menší velikosti.

Jako nevýhodu můžeme označit nemožnost více než 256 násobného opakování a problém související s možností výskytu escape sekvence (znaku) v komprimovaném souboru. První nevýhoda se řeší použitím čtyřbytového popisu opakování, čímž je zvětšen rozsah na 32768 opakujících se znaků, a v druhém případě se na výstup zapíše dvakrát po sobě znak escape sekvence. Tím se bohužel zbytečně navyšuje výstupní velikost a proto je znak volen tak, aby se vyskytoval s nejmenší frekvencí výskytu, nebo vůbec.

3 Hlediska hodnocení

3.1 Rychlost

Rychlost komprese vypovídá o složitosti algoritmu. Dá se říci, že rychlost je nepřímo úměrná „kvalitě“ komprese, tedy kompresnímu poměru (viz 3.2), a přímo úměrná pamětové náročnosti (viz 3.3). Jasně se zde nabízí otázka, čemu dát přednost. Výběr je samozřejmě otázkou kompromisu mezi rychlostí, kompresním poměrem a paměťovou náročností. V ideálním případě bychom vždy mohli použít „dokonalý“ kompresní algoritmus, který by ve všech ohledech zvítězil. V praxi to není možné už jen z toho důvodu, že pokaždé komprimujeme data jiného charakteru a náš dokonalý algoritmus by tedy musel být velice složitý kvůli dosažení velkého kompresního poměru a tedy by nemohl konkurovat v rychlosti jednodušším variantám.

Rychlost komprese je údaj porovnavatelný pouze s hodnotami získanými za stejných podmínek na stejném stroji. Zásadně jiné výsledky získáme při komprimování na různém hardwaru.

Na rozdíl od kompresního poměru je zde výsledek vlastně pokaždé jiný. Ovlivňují ho například další spuštěné procesy, nebo třeba i jen pohyb ukazatelem myši. Měřením času, s využitím časovačů běžících pro každý proces zvlášť, lze vliv minimalizovat.

Rychlost komprese budu uvádět v kB/s. Jako příklad můžeme zkomprimovat 50 kB dat za 10 ms:

$$Speed = \frac{InSize}{Time} = \frac{50\,kB}{10\,ms} = 5000\,kBs^{-1} \tag{1}$$

Jak z výpočtu vyplývá, počítá se se vstupní velikostí dat.

3.2 Kompresní poměr

Teoreticky by dokonalý kompresní algoritmus mohl být schopen ze signálu či informace odstranit veškerou redundanci a na jeho výstupu by měl být balík dat o velikosti dané množstvím informace vstupního signálu dle shannonova teorému. V praxi se tomuto ideálu snažíme co nejvíce přiblížit a jedním z kritérií výběru

je právě hodnota vypovídající o schopnosti zkomprimovat data, jíž je poměr mezi vstupní a výstupní velikostí.

Rozhodl jsem se používat hodnotu v procentech. Na příkladu ilustruji způsob výpočtu, pokud jsme našich 50 kB dat zkomprimovali na 30 kB:

$$Ratio = \frac{OutSize}{InSize} \cdot 100 = \frac{30\,kB}{50\,kB} \cdot 100 = 60\,\% \tag{2}$$

3.3 Paměťová náročnost

TOHLE JE PROBLÉM. MUSÍ SE JEŠTĚ POŘEŠIT...

4 Požadavky pro cílovou aplikaci

Komprese měřených dat má být prováděna v embeded měřicím zařízení, na kterém běží OS Linux. Úloha komprimování bude využívána periodicky vždy po zachycení určitého množství dat před tím, než budou výsledky měření odeslány do společného úložného a monitorovacího zařízení, v němž budou trvale uchovávána.

Vzhledem k hardwarovým parametrům zařízení je nejzásadnějším kritériem pro výběr kompresního algoritmu jeho paměťová náročnost. Z už tak malého množství dostupné paměti využívají běžící procesy tolik, že pro veškerou kompresní činnost zbývají jednotky kB.

Další vlastností, již je třeba zvážit, je výpočetní náročnost — tedy rychlost komprese. Ač se jedná o komprimaci několika kB naměřených dat, velmi pomalá komprese by mohla činit potíže, protože musejí být pravidelně obsluhovány další procesy a výkon CPU v embeded zařízení není něčím, čím by se dalo plýtvat.

Na posledním místě, i když v neposlední řadě, je dosahovaný kompresní poměr. Podle něj bychom se rozhodovali v případě, že bychom měli na výběr z více kompresních algoritmů s vlastnostmi popsanými výše na stejné úrovni.

5 Závěr

V TUTO CHVÍLI SEM NEMÁM CO PSÁT. PŘED ZÁVĚREM BYCH RÁD UVEDL POPIS TESTOVACÍHO PROGRAMU A ZPŮSOB GENEROVÁNÍ DAT PRO KOMPRESI, DÁLE TABULKY VÝSLEDKŮ PRO RŮZNÉ ZPŮSOBY ULOŽENÍ DAT, NĚJAKÉ TY GRAFY A NAKONEC TABULKU „VÍTĚZŮ“ SEŘAZENOU PODLE VŠECH KRITÉRIÍ. DO ZÁVĚRU SAMOZŘEJMĚ ROZEBRAT PROČ TA A TA KOMPRESE DOPADLA TAK A TAK A PROČ JE TEDY MINI-LZO NEJLEPŠÍ :-).

Reference

[1] Introduction to Losseless Data Compression:
http://web.archive.org/web/20060410170411/http://www.vectorsite.net/ttdcmp1.html

[2] LZ77:
http://en.wikipedia.org/wiki/LZ77_and_LZ78

[3] Basic Compression Library Manual
http://bcl.comli.eu/home-en.html

[4] LZO:
http://www.oberhumer.com/opensource/lzo/