Komprese dat (KOD)

Semestrální projekt

Implementace RLE, BWT a LZW

Autor: Bc. Petr Kašpar
Login: KAS265
Datum: 8. května 2009

Úvod

Úkolem tohoto projektu bylo implementovat nějaký komprimační algoritmus. Já jsem si vybral RLE (Run-length encoding) algoritmus doplněný o BWT (Burrows-Wheeler transformace). Dále jsem také implementoval LZW (Lempel-Ziv-Welch) algoritmus. Tyto algoritmy budou v závěru porovnány s programem WinRAR na nějakém reálném vzorku dat. U všech použitých metod jsou v implementaci k dispozici funkce pro komprimaci i dekomprimaci. Doplňující funkce se starají o výpočet kompresního poměru a formátování výstupů na obrazovku. Jako implementační prostředí jsem si zvolil scriptovací jazyk PHP a tedy výsledný ukázkový program bude přístupný jako internetová aplikace. Ukázkový program je vytvořen ve formě didaktické pomůcky a může tedy být prospěšný při studiu těchto algoritmů.

RLE (Run-length encoding)

Jedná se o bezeztrátovou metodu komprese. Komprese probíhá tak, že posloupnosti stejných symbolů ve vstupním slově jsou kódovány do tvaru (znak, délka posloupnosti).

Účinnost této komprese je silně závislá na charakteru vstupních dat. Ideální jsou data, ve kterých se vyskytují delší posloupnosti jednotlivých symbolů. Naopak u dat, ve kterých se žádné posloupnosti symbolů nevyskytují, můžou mít data po kompresi až dvojnásobnou délku.

RLE se používá například v grafickém formátu PCX jako hlavní kompresní metoda. Naopak u grafického formátu JPEG je RLE použito jako pomocná metoda.

Ukázka RLE

Vstupní slovo:

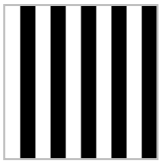
```
aaaaaaaaabbaaaaaaaaaaaaabbbbbbbbbbbbbbaaaabbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaa
```

Komprimované slovo metodou RLE:

```
a9b2a13b13a4b17a17
```

Porovnání RLE na obrázcích s různou kompozicí

V následující tabulce (Tabulka 1) jsou výsledky testu komprese dvou PCX obrázků, které se liší jen svou orientací. V nekomprimovaném TIFF formátu mají oba obrázky stejnou velikost.

Vstupní obraz (PCX)	Velikost
	1,9kB
	6,7kB

Tabulka 1: Porovnání velikostí PCX obrázků

BWT (Burrows-Wheeler transformace)

Algoritmus BWT je pomocný algoritmus, který se často používá s jinými algoritmy (já ho budu používat ve spojení s RLE). Transformace způsobí záměnu pořadí jednotlivých symbolů ve vstupním slově tak, aby ve výstupním slově byly stejná písmena pohromadě. Toto je výhodné právě ve spojení s RLE metodou, kde jsou posloupnosti stejných symbolů velice žádoucí. Algoritmus tedy data nijak nekomprimuje, ale pouze přeskupuje písmena ve slově (přesmyčky). Navíc musí být k datům přidán ještě symbol, který označuje konec slova. Takže nejen, že data nejsou nijak komprimována, ale navíc ještě přidáme jeden symbol navíc. Symbol se nemusí přidávat přímo k datům, v tomto případě si však musíme pamatovat adresu, na které má symbol být a tuto adresu případně zasílat i s daty, aby byla možná zpětná transformace.

Algoritmus funguje tak, že ze vstupního slova jsou provedeny všechny možné rotace, daná rotovaná slova jsou seřazena podle abecedy a výstupem je poslední sloupec těchto slov. Tento postup je prakticky vidět na následující ukázce, ve které je BWT transformace provedena nad slovem:

```
^BANANA
```

Symbol ♠ se na začátku doplní ke slovu a značí konec daného slova, který je potřebný pro provedení inverzní BWT transformace, tedy ke zrekonstruování původního slova. V Tabulce 2 je již k vidění samotný příklad na BWT:

Vstupní slovo	Všechny rotace	Seřazené rotace	Výstup
^BANANA♠	^BANANA♠ ♠^BANANA A♠^BANAN NA♠^BANA ANA♠^BAN NANA♠^BA ANANA♠^B BANANA♠^	ANANA♠^B ANA♠^BAN A♠^BANAN BANANA♠^ NANA♠^BA NA♠^BANA ^BANANA♠ ♠^BANANA	BNN^AA♠A

Tabulka 2: Princip BWT

Symbol označující konec (♠) musí mít nejmenší prioritu při třízení. V samotné implementaci jsem používal ASCII znak s adresou `0xFF`.

Inverzní BWT se provádí tak, že slovo po BWT zapíšeme do tabulky a setřídíme. Poté na začátek přidáme nový sloupec s naším slovem a opět setřídíme. Stejný postup zopakujeme tolikrát, kolik symbolů (včetně ukončovacího) má naše slovo. Původní slovo následně získáme tak, že z tabulky vybereme to slovo, které má na posledním místě ukončovací symbol (♠).

Ukázka BWT

Vstupní slovo:

```
statistika statistika statistika statistika statistika statistika
```

Vstupní slovo po BWT:

```
aaaaakkkkttttttkttttttttttttiiiiii♠iiiiissssssssssssaaaaaaa
```

Vstupní slovo po BWT a RLE:

```
a5k5t6k1t12i6♠1 5i6s12a7
```

Poznámka: ukončující symbol není nutné přímo vkládat do textu. V tom případě si ale musíme pamatovat místo, kde by se měl nacházet.

LZW (Lempel-Ziv-Welch)

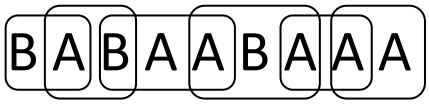
Jedná se o slovníkovou metodu, bezeztrátový algoritmus. Je to vylepšení algoritmů *LZ77* a *LZ78*. Až do roku 2004 byl tento algoritmus zatížen patentem, což například do tohoto data znepříjemňovalo použití grafického formátu GIF, který právě tento kompresní algoritmus používá.

Metoda používá slovník. Na začátku je slovník naplněn všemi jednoznakovými symboly. Typicky 256 znaků na adresy podle jejich *ASCII* kódu. Během komprese se do něj postupně přidávají použité dvouznakové a víceznakové řetězce. Maximální velikost slovníku může být 4095 (12 bitů). Pokud používáme nějaký variabilní či jinak omezený počáteční slovník, je nutné tento počáteční slovník přibalit ke komprimovaným datům, aby byla možná dekomprese.

Následuje praktická ukázka LZW komprese a dekomprese. Pro zjednodušení slovník inicializujeme pouze znaky A a B. Výstupem LZW komprese jsou adresy do slovníku. Každá z těchto adres se dá zakódovat do 12 bitů.

Ukázka komprese

Zde je jednoduchá ukázka LZW komprese. V Tabulce 3 je použitý slovník.

Vstupní slovo: 

1. Inicializujeme počáteční slovník (zde pro jednoduchost jen znaky **A** a **B**)
2. **B** je ve slovníku
   2.1. **BA** není ve slovníku, vložíme **BA** na adresu 3, na výstup adresa prefixu (**B**) → **2**
3. **A** je ve slovníku
   3.1. **AB** není ve slovníku, vložíme **AB** na adresu 4, na výstup adresa prefixu (**A**) → **1**
4. **BA** je ve slovníku
   4.1. **BAA** není ve slovníku, vložíme **BAA** na adresu 5, na výstup adresa prefixu (**BA**) → **3**
5. **AB** je ve slovníku
   5.1. **ABA** není ve slovníku, vložíme **ABA** na adresu 6, na výstup adresa prefixu (**AB**) → **4**
6. **AA** není ve slovníku, vložíme **AA** na adresu 7, na výstup adresa prefixu (**A**) → **1**
7. **AA** je ve slovníku, žádný další znak již není, na výstup adresa (**AA**) → **7**

Výstup:

```
2  1  3  4  1  7
```

Adresa	Slovo
1	A
2	B
3	BA
4	AB
5	BAA
6	ABA
7	AA

Tabulka 3: Slovník

Ukázka dekomprese

Vstupní komprimované slovo: **2 1 3 4 1 7**

1. Inicializujeme počáteční slovník (zde pro jednoduchost jen znaky **A** a **B**)
2. Adresa **2** je ve slovníku, na výstup slovo na adrese 2 → **B**
3. Adresa **1** je ve slovníku, na výstup slovo na adrese 1 → **A**, do slovníku vložíme **BA** na adresu 3
4. Adresa **3** je ve slovníku, na výstup slovo na adrese 3 → **BA**, do slovníku vložíme **AB** na adresu 4
5. Adresa **4** je ve slovníku, na výstup slovo na adrese 4 → **AB**, do slovníku vložíme **BAA** na adresu 5
6. Adresa **1** je ve slovníku, na výstup slovo na adrese 1 → **A**, do slovníku vložíme **ABA** na adresu 6
7. Adresa **7** není ve slovníku, na výstup předchozí výstup + jeho první znak → **AA**

Výstupní dekomprimované slovo:

```
BABAABAAA
```

Ukázka velikosti dat a kompresního poměru

Vstupní slovo: **AAABBBBBBAABAABA** → 16 znaků × 8 bitů = 128 bitů

Po LZW komprimaci: **1 125 2 127 128 126 130** → 7 adres × 12 bitů = 84 bitů

V případě, že využijeme pouze 8bitové adresy, tak bude mít komprimované slovo jen 56 bitů. Kompresní poměr je tedy v případě 12bitových adres *1,52 : 1* a v případě 8bitových adres *2,29 : 1*. U bezeztrátových kompresních algoritmů je kompresní poměr kolem 2 : 1 typický.

Popis implementace

Uvedené algoritmy jsou implementovány ve skriptovacím jazyce PHP. Veškeré funkce jsou ve třídě `KOD` v souboru `kod.class.php`. K dispozici jsou následující funkce:

- `bwt($text)` – provede BWT transformaci
- `inverzni_bwt($text)` – provede inverzní BWT transformaci
- `rle($text)` – provede RLE kompresi
- `inverzni_rle($text)` – provede RLE dekompresi
- `lzw($text)` – provede LZW kompresi
- `inverzni_lzw($text)` – provede LZW dekompresi
- `kompresniPomer($vstup, $komprimovane, $lzw = null, $slovnik=null)` – vypočítá kompresní poměr

Jednotlivé funkce následně používáme podle následujícího schématu:

```
<?php

require('kod.class.php');

$k = new KOD();

$text1 = 'statistika statistika statistika statistika statistika';

echo $k->bwt($text1);

echo $k->rle($k->bwt($text1));

?>
```

Komplexní použití navržené třídy je k vidění v souboru `index.php`, ve kterém jsou použity veškeré navržené funkce a jejich kombinace.

Experiment

V této části uvedu výsledky menšího pokusu, ve kterém jsem otestoval výše zmíněné metody a porovnal kompresní poměr s komerčním programem WinRAR. Experiment probíhal s klasickým textem (textovými soubory) v délce do 160 znaků (znaky bez diakritiky), což odpovídá SMS zprávám v mobilních telefonech. Nastavení WinRARu bylo na nejlepší kompresi.

Výsledky tohoto experimentu jsou uvedeny v Tabulce 4. WinRAR měl s takto malými soubory problém a již je nedokázal zkomprimovat. V naprosté většině měl text komprimovaný programem WinRAR ještě větší velikost, jak vstupní nekomprimovaný text. Nejlepší výsledky podávala metoda LZW, která byla u všech textů do 160 znaků nejlepší. RLE + BWT metoda také data nijak nezkomprimovala, avšak byla na tom ve většině případů lépe jak WinRAR.

Poznámka k tabulce: veškeré hodnoty jsou v bajtech, zelenou barvou je označená nejmenší hodnota, červenou barvou je označena největší hodnota. Text #11 je delší jak 160 znaků, ve skutečnosti to je všech 10 předchozích textů v jednom. Zde již WinRAR podal nejlepší výsledek, ale LZW data také zkomprimovalo.

Text #	Originál	RLE+BWT	LZW	WinRAR	LZW kompr. poměr
1	150	254	112	184	1,34 : 1
2	20	38	18	93	1,11 : 1
3	90	152	73	151	1,23 : 1
4	19	38	18	92	1,06 : 1
5	50	90	43	123	1,16 : 1
6	92	158	76	154	1,21 : 1
7	141	142	109	181	1,29 : 1
8	63	64	55	136	1,15 : 1
9	45	86	41	118	1,10 : 1
10	34	64	31	107	1,10 : 1
11	714	1022	618	497	1,16 : 1

Tabulka 4: Test jednotlivých algoritmů a porovnání s WinRAR

Ukázka implementace

Na Obrázku 1 v Příloze A je vidět výsledná implementovaná aplikace. Je možno nastavit vlastní vstupní texty samostatně pro RLE + BWT i pro LZW. Aplikace také provede výpočet velikosti jednotlivých slov v bitech, vypíše kompresní poměr. Pokud se data skutečně zkomprimovala, bude kompresní poměr vypsán zelenou barvou, u nezkomprimovaných dat bude poměr vypsán červenou barvou a u dat beze změny bude použitá modrá barva.

U LZW metody je možno si zobrazit vytvořený slovník. Při vytváření tohoto slovníku máme na výběr, zda se při inicializaci slovníku použijí pouze znaky, které jsou ve vstupním slově, anebo všechny jednoznakové znaky. Tato volba se nastavuje pomocí checkboxu *"Omezená abeceda"* při zadávání vstupního slova. Při zaškrtnutí se použijí jen nezbytně nutná písmena.

U BWT metody je možnost si zobrazit průběh této transformace. Po kliknutí na odkaz *"Zobrazit postup BWT"* se zobrazí v přehledné tabulce jednak jednotlivé rotace a také jejich seřazení. Červenou barvou v této tabulce jsou zvýrazněna písmena, která patří do výstupního slova.

Závěr

Implementoval jsem celkem 2 algoritmy pro kompresi dat. RLE doplněný o BWT transformaci se hodí pro data, ve kterých jsou větší posloupnosti stejných znaků. Pro tato data je tato kompresní metoda účinná. U dat bez posloupnosti stejných znaků není tato kompresní metoda vhodná, jelikož výsledný řetězec může mít až dvojnásobnou délku. Obecně se RLE spíše než u textu používá u obrazových souborů. V rámci experimentu se také potvrdilo, že tato metoda není vhodná pro komprimaci krátkých textů (SMS zprávy), jelikož v těchto textech není dostatek posloupností stejných znaků a výsledná komprimovaná data mají většinou ještě větší velikost

LZW komprimace je vhodná pro text i obrázky. Předností této metody je její rychlost a také to, že slovník není nutné posílat spolu s daty. Slovník se vytváří dynamicky při kompresi i dekompresi. Mezi nevýhody bych uvedl, že maximální počet položek ve slovníku je 4095. Při zaplnění je nutné některé slova ze slovníku vymazat (nejčastěji ty nejméně používaná). Z praktického experimentu vyplynulo, že tato metoda je vhodná i pro komprimaci kratších textů (typicky SMS zprávy), u kterých podávala lepší výsledky jak komerční program WinRAR.

Použité metody je možno vyzkoušet na stránkách http://blog.mynameisearl.cz/skola/kod/, kde je tento projekt vystaven. Součástí jsou také různé prvky (zobrazení slovníku, postupu BWT), díky kterým je možno tuto aplikaci použít také jako didaktickou pomůcku k výuce těchto algoritmů.

Použitá literatura

[1] Lempel-Ziv-Welch [online]. 2009 [cit. 2009-04-19]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Lempel-Ziv-Welch>.

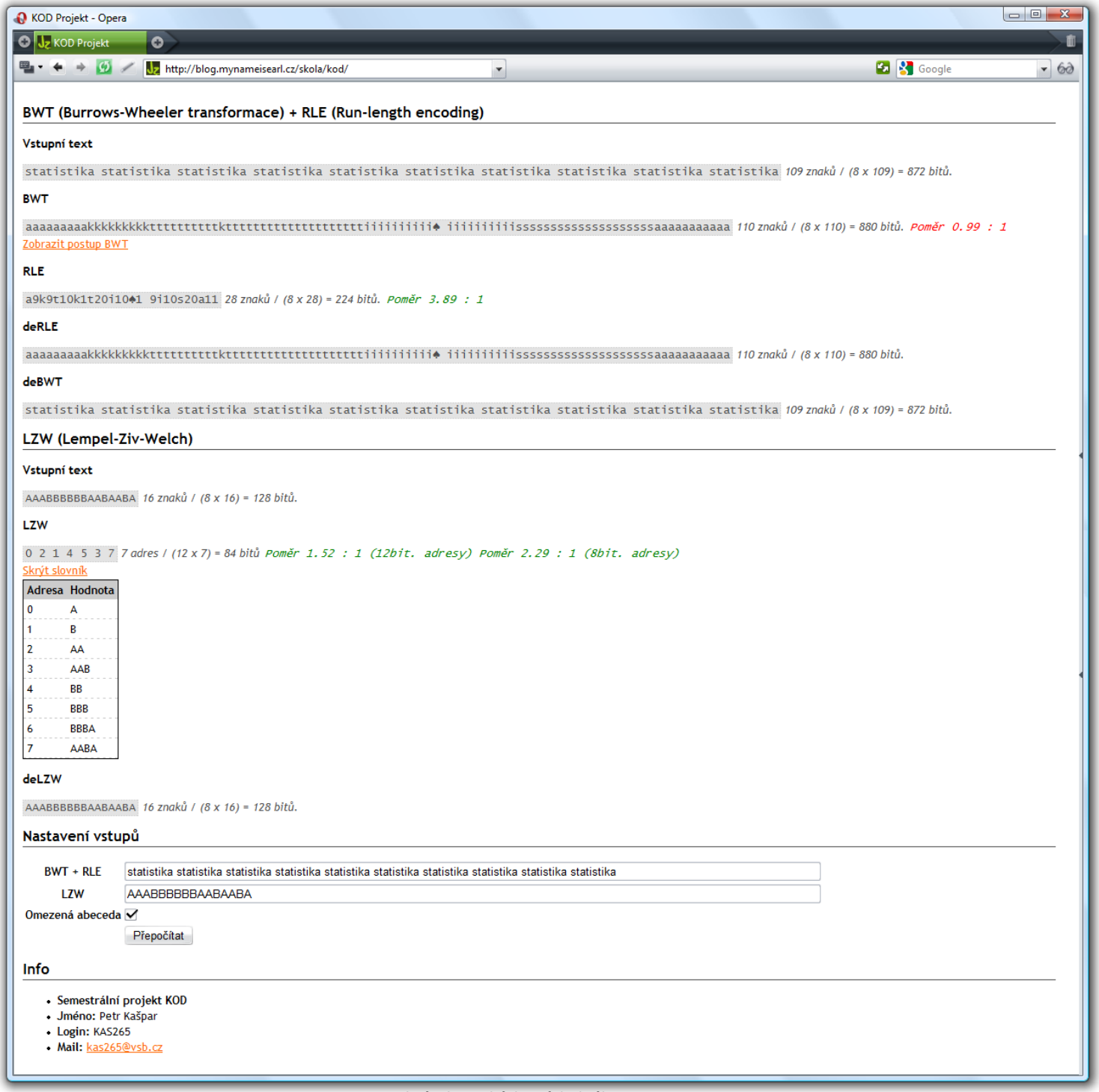
[2] Run-length encoding [online]. 2009 [cit. 2009-04-19]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Run-length_encoding>.

[3] Burrows-Wheeler transform [online]. 2009 [cit. 2009-04-19]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Burrows-Wheeler_transform>.

[4] KOJECKÝ, Marek. Komprese binárních textových a grafických dat. [s.l.], 2005. 46 s. Bakalářská práce.

[5] Lempel-Ziv-Welch (LZW) Compression Algorithm [online]. [2002] [cit. 2009-04-19]. Dostupný z WWW: <faculty.kfupm.edu.sa/ICS/jauhar/ics202/Unit32_LZW.ppt>.

Příloha A – Ukázka výsledného programu



Obrázek 1: Ukázka výsledného programu