Projekt do předmětu MUL 2013L
Komprese dat

Jan Dušek
xdusek17@stud.fit.vutbr.cz

13.5.2013

1 Úvod

V tomto projektu jsme měli za úkol, prostudovat statistické, kontextové a slovníkové metody komprese dat. Z každého typu si vybrat jednu metodu a tu implementovat. Poté bylo úkolem tyto metody porovnat na multimediálních datech (texty, zvuk, obraz a dokumenty).

Z časových důvodů jsme stihli implementovat pouze aritmetické kódování, ze statistických metod a algoritmus Lempel–Ziv–Welch, ze slovníkových metod.

2 Algoritmy komprese

V této části si teoreticky představíme algoritmy, které jsme v tomto úkolu implementovali. Jsou to aritmetické kódování, jež patří do statistických metod a algoritmus Lempel–Ziv–Welch, který se řadí do slovníkových metod.

2.1 Aritmetické kódování

[3] Aritmetické kódování je druh statistického kódování, kdy celý text kódujeme jedním číslem z intervalu $[0, 1)$. Text je tvořen ze symbolů $a_1, a_2, \ldots, a_n$, které se vyskytují s pravděpodobnostmi $p_1, p_2, \ldots, p_n$. Principem je to, že se tento interval rozdělí na n disjunktních intervalů jejichž velikost je určena pravděpodobnostmi. Při zavedení kumulativních pravděpodobností

$$q_0 = 0,\ q_1 = p_1,\ q_2 = p_1 + p_2,\ q_n = p_1 + p_2 + \ldots + p_n = 1$$

patří symbolu a_i interval $[q_{i-1}, q_i)$. Postup aritmetického kódování je takový, že se nepočítá přímo výsledná hodnota, nýbrž se neustále vymezuje interval v němž výsledná hodnota leží, tento interval budeme v následujícím textu označovat symbolem I.

Nejdříve si zvolíme I=[0,1). Poté ze vstupu odebereme znak a_i, určíme jemu odpovídající interval $[q_{i-1}, q_i)$ a ze stávající hodnoty I=[l,h) vypočítáme novou dle vzorce:

$$I \quad = \quad [l + q_{i-1} * (h - l), l + q_i * (h - l))$$

tím jako nový interval vybereme tu část jež odpovídá intervalu znaku a_i. Předchozí krok opakujeme dokud nezakódujeme všechny znaky ze vstupu. Výsledkem pak libovolné číslo c z intervalu I.

Dekódování vychází ze stejného rozdělení na n disjunktních intervalů, v každém kroku pak nalezneme interval ve kterém se nachází zakódovaná hodnota c. Z něho určíme o který znak se jedná a vypočítáme novou hodnotu intervalu I stejným způsobem jako při kódování a pokračujeme v dekódování dalšího znaku.

Aritmetické kódování používá pro interpretaci intervalu I reálná čísla. S každým zakódovaným znakem se interval zmenší a čím menší tím je číslo potřebné k jeho zápisu delší. Princip je tedy takový, že znaky s větší pravděpodobností zmenšují interval méně než čísla s pravděpodobností menší. Jelikož práce s reálnými čísly je v počítači velmi složitá, tak reálné implementace používají celá čísla.

Z praktického hlediska je také důležité jak zjistíme pravděpodobnosti symbolů. Základní metodou je, že pravděpodobnosti známe již před započetím kódování a zůstávají po celou dobu komprese. Vzhledem k tomu, že jak

Algoritmus 1 LZW kódování

```
string = získej vstupní znak
while (máme znaky na vstupu) {
        character = získej vstupní znak
        if (string+character je ve slovníku)
                string = string+character
        else {
                dej na výstup kód pro string
                přidej string+character do slovníku
                string = character
        }
}
dej na výstup kód pro string
```

kodér tak dekodér musí mít pravděpodobnosti stejné, je nutno tyto pravděpodobnosti přenášet spolu s hodnotou c. Druhou metodou je tzv. adaptivní kódování, kdy z počátku nastavíme pravděpodobnosti symbolů stejně, ale po každém zakódování znaku zvýšíme hodnotu pravděpodobnosti tohoto znaku. Dekodér s pravděpodobnostmi pracuje analogicky. a tedy není nutno si pravděpodobnosti nikam ukládat. Rovněž tímto adaptivní kódování bere v potaz změnu pravděpodobností různých znaků v průběhu textu.

2.2 Lempel–Ziv–Welch

[1, 2] Algoritmus Lempel-Ziv-Welch (LZW) byl vytvořen v roce 1984 jako zdokonalení populárního algoritmu LZ77. Patří do rodiny slovníkových metod. Komprese spočívá v nahrazení řetězce znaků jedním kódem jež slouží jako index do slovníku. Algoritmus nijak neanalyzuje přicházející data pouze si tvoří ze zakódovaných řetězců postupně slovník.

Kódy, které algoritmus generuje mohou být libovolně dlouhé, fixní či variabilní. Typicky se prvních 256 kódů mapuje přímo na jednotlivé byty a zbytek kódů reprezentuje podřetězce obsažené ve slovníku.

Na výpisu 1 vidíme zápis algoritmu komprese v pseudokódu. Algoritmus čte ze vstupu znaky a pokud narazí na podřetězec který ještě nemá ve slovníku tak vloží na výstup kód předcházejícího podřetězce, který zná a uloží si ten neznámý do slovníku.

Dekompresní algoritmus je schopen si slovník vygenerovat, takže si ho nemusíme přenášet spolu se zakódovanými daty. Na výpisu 2 vidíme zápis algoritmu dekomprese v pseudokódu, ten pro každý nový kód si přidá řetězec do slovníků, rovněž přeloží kód na řetězec a ten pošle na výstup. Rovněž musí algoritmus ošetřit výjimku, pokud slovník obsahuje podřetězec *řetězec + znak* a na vstupu je sekvence *řetězec + znak + řetězec + znak + řetězec*, v tento moment kodér vygeneruje kód, který ale ještě dekodér nemá uložen a došlo by k chybě. Pokud tato výjimka nastane, tak je potřeba přidat první znak řetězce kódovaného předchozím kódem na konec řetězce kódovaného předchozím kódem. Takto získáme správný výsledný řetězec pro tento v tuto chvíli neznámý kód a můžeme pokračovat standardně dále.

Při implementaci je potřeba řešit velikost kódů. Pro různě veliké soubory je ideální jiná velikost slovníku, pro menší soubory je lepší méně bitů, jelikož by pak horní bity zůstaly nevyužity a zhoršil by se kompresní poměr, větší soubory pak potřebují více bitů, jelikož jejich slovník vyžaduje více záznamů. Velikost kódů jsme tedy nezvolili fixní nýbrž variabilní, kdy se velikosti postupně zvětšují. Také je možno kódy ještě dále zakódovat nějakou statistickou metodou například aritmetickým kódováním.

3 Implementace

V této kapitole si popíšeme velmi zjednodušeně implementaci. Projekt byl rozdělen na několik částí. Hlavní částí je knihovna ve které je soustředěna veškerá funkcionalita. A dva programy, které provádí kompresi/dekompresi pomocí aritmetického kódování a algoritmu LZW.

Algoritmus 2 LZW dekódování

```
old_code = získej kód ze vstupu
dej na výstup old_code
character = old_code
while (máme kódy na vstupu) {
        new_code = získej kód ze vstupu
        if (new_code není ve slovníku) {
                string = získej překlad old_code
                string = string + character
        } else {
                string = získej překlad new_code
        }

        dej string na výstup
        character = první znak ve string
        přidej překlad old_code + character do slovníku
        old_code = new_code
}
```

3.1 Použité technologie

Projekt byl implementován v jazyce C++ s použitím nové normy C++11. Pro překlad a správu závislostí jsme použili multiplatformní *CMake*. Nebyly použity žádné externí knihovny kromě frameworku *googletest*, který ale není k běhu programů požadován, slouží pouze k vytváření jednotkových testů.

3.2 Ovládání programů

V projektu byli vytvořeny dva programy. Prvním je *ac*, který provádí aritmetické kódování, umí použít jak adaptivní tak statický datový model.

```
ac [-s] INPUT OUTPUT
ad -d INPUT OUTPUT
        -s      Použije se statický datový model. Četnosti jsou pak uloženy ve výstupním soub
        -d      Dekomprese
        INPUT   vstupní soubor pro kompresi/dekompresi.
        OUTPUT  výstupní soubor.
```

Druhým programem je *lzw*, který provádí kompresi algoritmem LZW. Umožňuje použít buď variabilní délku kódů nebo kódy zakódovat aritmetickým kódováním.

```
lzw [-a] INPUT OUTPUT
lzw -d INPUT OUTPUT
        -a      Použije se aritmetické kódování místo variabilní délky kódu.
        -d      Dekomprese
        INPUT   vstupní soubor pro kompresi/dekompresi.
        OUTPUT  výstupní soubor.
```

4 Porovnání algoritmů

Testování probíhalo na sadě testovacích dat rozdělených do čtyř skupin dle typu a to text, zvuk, obraz a dokumenty čítající 6-10 souborů. V tabulce 1 vidíte jednotlivé soubory a jejich velikosti.

Nad těmito soubory pak bylo provedeno porovnání statického a adaptivního aritmetického kódování, LZW s variabilní délkou kódu a LZW s aritmetickým kódováním. Výsledky tohoto porovnání jsou vidět v tabulce 2.

Tabulka 1: Testovací data

Text

číslo	popis	velikost [kB]
1	kniha v textu	148
2	kniha v textu	79
3	kniha v textu	121
4	kniha v textu	830
5	soubor jazyka C++	3,78
6	soubor jazyka C++	4,98
7	seznam url do wikipedie	16 276
8	kniha v textu	454
9	soubor jazyka C++	3,40
10	soubor jazyka C++	9,55

Zvuk

číslo	popis	velikost [kB]
1	píseň v mp3 320 kB/s	11 449
2	promluva v raw 8 kHz 16 bit	8
3	promluva ve wav 44100 kHz 16 bit	69
4	promluva v raw 8 kHz 16 bit	16
5	promluva ve wav 44100 kHz 16 bit	45
6	promluva v raw 8 kHz 16 bit	10
7	promluva ve wav 44100 kHz 16 bit	85
8	píseň v mp3 128 kB/s	2 991
9	píseň v mp3 160 kB/s	3 081

Obraz

číslo	popis	velikost [kB]
1	obrázek v jpg	189
2	fotka v jpg	14
3	nákres v png	30
4	fotka v jpg	4
5	obrázek v png	317
6	nákres v bmp	418
7	nákres v bmp	385
8	screenshot v bmp	3 076
9	obrázek v png	292

Dokumenty

číslo	popis	velikost [kB]
1	prezentace v ppt	453
2	text v pdf	742
3	tabulka v ods	21
4	bakalářka v pdf	2 355
5	prezentace v pdf	109
6	tabulka v xlsx	12

Je vidět, že LZW je velmi dobrý v kompresi textu a nekomprimovaných obrázků, ale u komprimovaných dat i nekomprimovaných zvukových dat je naprosto nevhodný jelikož způsobí zvětšení souboru namísto komprimace, tento jev nastává protože kódy mají více bitů než jednotlivé byty a když není v datech dost společných podřetězců tak algoritmus zcela míjí svůj účel. LZW s aritmetickým kódováním je oproti LZW s proměnlivou délkou kódů v klíčových skupinách většinou o něco horší, vyskytli se ale situace kdy byl zase o dost lepší.

Aritmetické kódování je velmi dobré na nekomprimovaných textech a obrazových datech, kde ale LZW dosahuje lepších výsledků. Na nekomprimovaných zvucích je to horší, ale stále je dosaženo nějaké komprese na rozdíl od LZW. U komprimovaných dat dochází v několika procentní kompresi, takže netrpí stejným neduhem jako LZW, že by zvětšovalo velikost souborů. Statické kódování bylo v naprosté většině případů o málo horší než adaptivní, což bylo způsobeno tím, že statické potřebuje mít u sebe uloženy i frekvence souborů, což zhoršuje kompresi a rovněž se nedokáže adaptovat na lokální změny výskytů symbolů.

5 Závěr

V projektu se podařilo implementovat algoritmy aritmetického kódování a LZW, rovněž jsme implementovali jejich kombinaci, kdy je výstup z LZW kódován ještě aritmetickým kodérem. Algoritmy byli otestovány na čtyřech skupinách multimediálních dat.

Algoritmus LZW prokázal vysokou účinnost nad texty a nekomprimovanými obrázky naopak u zvuku a komprimovaných dat totálně pohořel, kdy došlo ke zvětšení dat namísto komprimace. Aritmetické kódování prokázalo svou všestrannost, kdy zvládá slušně texty, zvuk i obraz. Rovněž dokázal zkomprimovat, byť nepatrně, i již zkomprimované formáty dat.

Tabulka 2: Výsledné kompresní poměry

Text

soubor	adaptivní ac	statické ac	LZW	LZW + ac
1	0,655	0,655	0,473	0,473
2	0,658	0,671	0,544	0,557
3	0,653	0,661	0,487	0,521
4	0,635	0,636	0,502	0,418
5	0,696	0,923	0,598	0,739
6	0,685	0,857	0,582	0,735
7	0,646	0,646	0,440	0,328
8	0,568	0,571	0,405	0,410
9	0,688	0,944	0,532	0,782
10	0,685	0,773	0,490	0,659

Zvuk

soubor	adaptivní ac	statické ac	LZW	LZW + ac
1	0,941	0,941	1,329	1,072
2	0,837	0,959	1,161	1,317
3	0,812	0,826	0,956	0,971
4	0,873	0,930	1,234	1,259
5	0,889	0,911	1,156	1,089
6	0,816	0,912	1,156	1,304
7	0,859	0,871	0,953	0,929
8	0,996	0,996	1,223	1,045
9	0,997	0,997	1,232	1,044

Obraz

soubor	adaptivní ac	statické ac	LZW	LZW + ac
1	0,984	0,984	1,038	0,925
2	0,986	1,051	1,384	1,355
3	1,003	1,034	1,366	1,250
4	1,002	1,257	1,273	1,576
5	0,997	1,000	1,255	1,079
6	0,309	0,309	0,050	0,062
7	0,091	0,094	0,031	0,038
8	0,456	0,456	0,215	0,156
9	1,000	1,003	1,260	1,082

Dokumenty

soubor	adaptivní ac	statické ac	LZW	LZW + ac
1	0,936	0,938	1,110	0,938
2	0,998	0,998	1,221	1,037
3	0,980	1,024	1,282	1,233
4	0,995	0,996	1,226	1,015
5	0,981	0,990	1,193	1,028
6	0,888	0,965	1,068	1,143

Reference

[1] Dipperstein, M. *Lempel-Ziv-Welch (LZW) Encoding Discussion and Implementation* [online]. [cit. 13.5.2013]. Dostupné na: <http://michael.dipperstein.com/lzw/>.

[2] Nelson, M. *LZW Data Compression* [online]. 1989 [cit. 13.5.2013]. Dostupné na: <http://marknelson.us/1989/10/01/lzw-data-compression/>.

[3] Vecerka, A. *Komprese dat* [online]. 2008 [cit. 1.5.2013]. Dostupné na: <http://phoenix.inf.upol.cz/esf/ucebni/komprese.pdf>.