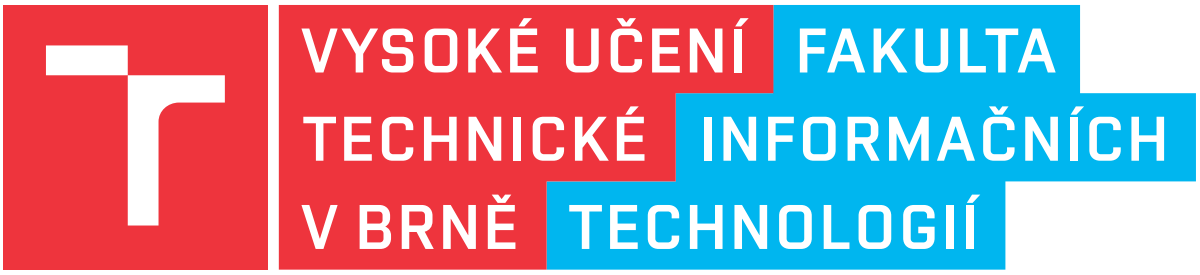

Kódování a komprese dat (KKO)

Komprese obrazových dat s využitím statického a adaptivního Huffmanova kódování

v Brně
14. dubna 2019

Václav Martinka
xmarti76

1 Zadání

Cílem projektu je v programovacím jazyce C/C++ vytvořit aplikaci pro kompresi šedo tónových obrazových dat, kde se uplatní principy statického a adaptivního Huffmanova kódování. Vytvořené řešení musí být spustitelné coby tzv. konzolová aplikace (tedy z příkazového řádku) pod OS Linux v prostředí počítačové sítě FIT VUT v Brně.

2 Rozbor problému

Zadání vyžaduje nastudovat kompresi dat pomocí statického i adaptivního kódování dat. K tomu je dále potřeba zvolit vhodný model pro předzpracování dat.

2.1 Statické Huffmanovo kódování

Z principu se jedná o dvou-průchodovou metodu. Při prvním průchodu daty je změřen výskyt jednotlivých symbolů.[1] V rámci projektu jsem se rozhodl jako symbol použít jeden byte (pixel) vstupního souboru.

Na základě této statistiky je pak vytvořen Huffmanův strom, kde znaky s vyšším výskytem jsou umístěny v nižší hloubce stromu. Průchod stromem ke konkrétnímu symbolu pak lze převést na binární vyjádření a to zapsat do výstupního (komprimovaného) souboru.

Aby mohl být takto kódovaný soubor dekódován, je nutné do souboru uložit i samotný Huffmanův strom (nebo alespoň data potřebná k jeho vytvoření).

2.2 Adaptivní Huffmanovo kódování

Každý nově načtený symbol je přímo zapsán na výstup (spolu se speciální *NYT* značkou, taktéž kódovanou Huffmanovým kódem) a následně je přidán do stromu. Jeho další výskyty jsou už kódovány stejně jako u statického kódování, přičemž po každém vyhledání libovolného symbolu je strom (respektive počet výskytů daného symbolu) aktualizován a v případě potřeby přeskládán.

Výhodou tohoto přístupu je absence nutnosti dvou průchodů souborem, což lze využít např. ke kódování souvislého toku dat. Stejně tak není nutné do výstupního souboru ukládat strukturu stromu, jelikož je sama vytvářena až při samotném čtení.

Nevýhodou je nutnost pro každý načtený symbol aktualizovat strom, což negativně ovlivňuje dobu komprese.

2.3 Předzpracování dat

Ke zvýšení účinnosti komprese je vhodné data určitým způsobem upravit. V kombinaci s Huffmanovým kódováním je např. výhodné snížit počet symbolů, což vede ke stromu s nižší hloubkou. Toho lze dosáhnout mimo jiné i pomocí diference pixelů, což znamená, že jednotlivé pixely nejsou popisovány svojí hodnotou, nýbrž rozdílem od předchozího pixelu. Pokud obrázek neobsahuje velké množství náhlých změn, bude výsledný souboru obsahovat převážně nižší hodnoty. Tato metoda byla použita v rámci projektu.

3 Vlastní řešení

Struktura programu odpovídá popisu metod v kapitole 2. Níže uvádím podrobnější popis několika částí programu.

[1] symbolem se obecně myslí jeden znak vstupu, což nic nevyjadřuje o jeho délce nebo významu.

3.1 Komprese statickým kódováním

Zpracovávaný i výstupní soubor je po celý běh programu uložen v paměti RAM. To zvyšuje rychlost zpracování (není nutné přistupovat na disk) na úkor paměťových nároků (paměťová náročnost je cca dvojnásobek velikosti souboru).

V případě, že je aktivován model pro předzpracování, tak jsou vstupní data převedena pomocí modelu diference pixelů. Následně je vytvořena frekvenční charakteristika výskytu jednotlivých symbolů (bohužel s ohledem na návrh programu je to řešeno druhým průchodem souborem) a na jejím základě je vytvořen Huffmanův strom.

Nyní je již možní přistoupit k samotnému kódování. Z důvodu zrychlení je strom převeden na pole struktur `MaskedByte`[2], kde indexem do pole je hodnota kódovaného symbolu. Tyto kódované hodnoty jsou ukládány do bufferu a následně po bytech zapisovány do výstupního souboru.

3.2 Komprese adaptivním kódováním

Adaptivní kódování probíhá obdobně jako statické, viz 3.1. Jak bylo zmíněno v 2. kapitole, adaptivní kódování nevyžaduje průchod souborem pro vytvoření frekvenční charakteristiky. Místo toho je strom aktualizován (a případně přeskládán) po každém zapsaném symbolu. Aby byl i v tomto případě zápis dostatečně rychlý, tak kodér obsahuje pole ukazatelů na koncové listy stromu a jednotlivé uzly obsahují ukazatel na svého rodiče, což zaručuje velmi rychlý průchod stromem.

3.3 Hlavička souboru

Aby bylo možné soubor správně dekódovat, je nutné si do něj uložit hlavičku s potřebnými údaji. A to především počet bitů (0 – 7), které bylo nutné doplnit na konec výstupního souboru. Dále je potřeba uložit Huffmanův strom při statickém kódování a pro zvýšení uživatelské přívětivosti je v hlavičce uložena i informace o použitém typu kódování (statické/adaptivní), modelu pro předzpracování dat a existenci rozšířené hlavičky (příprava na případné rozšíření). Všechny tyto informace lze uložit do jednoho bytu, tudíž vliv na velikost souboru je zanedbatelná.

Uložení statického stromu je následující. Vstupní symboly jsou kódovány postupně (tedy nejdříve je uložen `MaskedByte` pro `0x00`, pak `0x01`, atd.). Pro každý symbol je uložena jeho délka v bitech a pak samotný symbol. To znamená, že jeden symbol tak může obsadit 2 – 5 bytů z výstupního souboru. V případě, že se daný symbol (symboly) ve stromě nevyskytují, tak se namísto délky zapíše záporné číslo, jehož hodnota vyjadřuje počet symbolů k přeskočení. Toho lze využít pro efektivnější kódování souborů, které neobsahují všech 256 kombinací bytů, což může být např. ASCII text.

3.4 Dekódování

Dekódování probíhá přesně opačně. Komprimovaný soubor je čten bit po bitu a na základě jeho hodnoty se ukazatel do stromu pohybuje vpravo nebo vlevo. V případě, že narazí na list, zapíše do výstupního souboru daný symbol a přesune se zpět na kořen. U adaptivního kódování je opět nutné aktualizovat strom. Po přečtení celého souboru případně proběhne zpětná transformace dat pomocí diference pixelů.

4 Vyhodnocení účinnosti komprese

Následující tabulky ukazují výsledek měření na běžném pracovním notebooku s procesorem *Intel i5*. *Stat.* značí statické kódování, *Adap.* adaptivní. Písmeno *M* symbolizuje použití modelu pro předzpracování dat.

[2] Tato struktura obsahuje 32 bitový integer pro uložení kódovaného symbolu (čímž vzniká omezení na maximální hloubku stromu 32) a počet využitých bitů, jelikož Huffmanův kód využívá proměnnou délku symbolu.

	Stat.	**Stat. *M***	**Adap.**	**Adap. *M***
hd01	3,89	3,42	3,88	3,41
hd02	3,72	3,35	3,71	3,34
hd07	5,63	3,87	5,62	3,86
hd08	4,24	3,54	4,24	3,53
hd09	6,67	4,70	6,67	4,69
hd12	6,21	4,40	6,21	4,39
nk01	6,52	6,08	6,51	6,07
průměr	5,27	4,19	5,26	4,18

Tabulka 1: Průměrný počet bitů potřebný k zapsání jednoho bytu

	Stat.	**Stat. *M***	**Adap.**	**Adap. *M***
hd01	60,24 ms	70,05 ms	111,68 ms	113,35 ms
hd02	59,95 ms	68,82 ms	102,48 ms	99,04 ms
hd07	71,94 ms	68,37 ms	123,23 ms	99,67 ms
hd08	53,77 ms	70,97 ms	104,14 ms	104,15 ms
hd09	99,70 ms	73,82 ms	149,93 ms	125,24 ms
hd12	79,54 ms	69,74 ms	144,37 ms	106,75 ms
nk01	88,97 ms	81,27 ms	141,29 ms	132,52 ms
průměr	73,44 ms	71,86 ms	125,30 ms	111,53 ms

Tabulka 2: Čas potřebný ke kompresi souboru

	Stat.	**Stat. *M***	**Adap.**	**Adap. *M***
hd01	84,84 ms	84,95 ms	101,36 ms	98,45 ms
hd02	79,41 ms	93,90 ms	99,07 ms	100,58 ms
hd07	100,49 ms	104,61 ms	126,78 ms	99,96 ms
hd08	138,33 ms	91,67 ms	106,65 ms	101,05 ms
hd09	123,49 ms	110,00 ms	159,99 ms	128,25 ms
hd12	116,26 ms	98,94 ms	182,13 ms	112,66 ms
nk01	118,59 ms	114,48 ms	145,59 ms	178,60 ms
průměr	108,77 ms	99,79 ms	131,65 ms	117,08 ms

Tabulka 3: Čas potřebný k dekompresi souboru

Je zde vidět několik zajímavostí. I primitivní metoda diference pixelu dosáhla měřitelného zlepšení a to na všech obrázcích, přičemž čím větší byl průměrný počet pixelů na byte bez jejího použití, tím efektivněji fungovala.

Jako špatně komprimovatelný obrázek se ukázal `nk01.raw`, který obsahoval velké množství šumu. V tomto případě by bylo vhodné použít např. dokonalejšího kódování, které by jako symbol pracovalo s více pixely nebo využít nějakou z transformací. Druhým řešením by mohlo být použití vhodné metody pro redukci šumu, což by ale znamenalo ztrátovou kompresi.

Pozitivní informací je, že metoda pro předzpracování dat neovlivňuje čas komprese. Naměřené hodnoty dokonce vykazují mírné zrychlení. To je ovšem nutné brát s rezervou, jelikož data v tabulce vychází pouze z jednoho měření na jednom počítači. Na vině může být operační systém a jeho optimalizace při opakovaném přístupu k souboru.

Podle očekávání probíhá dekomprese staticky kódovaného souboru pomaleji, jelikož zde není využita žádná optimalizace a je tak nutné pro každý symbol procházet celý strom. Oproti tomu v případě adaptivního kódování je strom procházen pro každý symbol jak při kódování, tak i při dekódování, tudíž k žádnému zpomalení nedochází.

5 Překlad a spuštění

Program lze přeložit pomocí přiloženého `Makefile`, který obsahuje následující příkazy:

`all` popř. pouze `make` přeloží program

`doxygen` vygeneruje *doxygen* dokumentaci

`zip` vytvoří archiv se zdrojovými kódy

`clean` smaže soubory vzniklé při překladu

`test` spustí testovací skript

Chování programu lze ovlivnit pomocí následujících parametrů:

`--help (-h)` Zobrazí nápovědu a ukončí program

`--version` Zobrazí verzi programu a ukončí program

`--compression (-c)` Vstupní soubor bude komprimován (výchozí režim)

`--decompression (-d)` Vstupní soubor bude dekomprimován (nelez kombinovat s `-compression`)

`--model (-m)` Aktivuje model pro předzpracování vstupních dat. (Automaticky detekováno při dekompresi.)

`--input (-i) <soubor>` Vstupní soubor

`--output (-o) <soubor>` Výstupní soubor. Výchozí je `out.raw`

`--verbous (-v)` Povolí vypisování podrobných informací o průběhu

`--quiet (-q)` Tichý režim, pouze výpisy na `stderr`

`--huffman (-h) static` Použije statické Huffmanovo kódování (výchozí)

`--huffman (-h) adaptive` Použije dynamické Huffmanovo kódování. Nelze kombinovat se statickým kódováním, automaticky detekováno při dekompresi

`--width (-w)` Šířka vstupního obrázku v pixelech – **nepoužito**

6 Závěr

Výsledný program na vstupních datech dosahuje až 58% úspory dat, přičemž průměrně pro všechny možnosti komprimuje testovací soubory o 41 % původní velikosti. I jednoduchá metoda pro předzpracování dat pomocí diference pixelů se ukázala jako efektivní, jelikož v průměru zlepšila kompresi o 14 % (v nejlepším případě dokonce o 25 %) a to za minimální cenu co se týče výpočetních zdrojů.

Porovnání statického a adaptivního kódování vede k závěru, že míra komprese je u obou metod srovnatelná (liší se v desetinách procent). I časem (de)komprese se od sebe výrazně neliší, tudíž určit která z metod je vhodnější by záleželo především na konkrétní situaci.

Při tvorbě programu jsem vycházel především ze studijním materiálů k tomuto předmětu.