# Komprese obrazových dat s využitím statického a adaptivního Huffmanova kódování

Katarína Grešová

20. apríla 2019

## 1 Úvod

Kompresia dát je proces konverzie vstupného dátového toku do iného dátového toku, ktorý má menšiu veľkosť. Rôzne metódy majú rôzne prístupy v priraďovaní kódov k symbolov, s ktorými pracujú. Štatistické metódy (medzi ktoré patrí Huffmanovo kódovanie [1]) používajú kódy s premenlivou dĺžkou, pričom kratšie kódy sú priradené symbolom alebo skupinám symbolov, ktoré majú vyššiu pravdepodobnosť výskytu [2].

## 2 Statické Huffmanovo kódovanie

Klasické statické Huffmanovo kódovanie počíta s tým, že pred začiatkom kódovania je známy zoznam frekvencií výskytov jednotlivých kódovaných symbolov. V tejto práci však statickým Huffmanovým kódovaním budeme nazývať semistatické Huffmanovo kódovanie, kde je zoznam frekvencií získaný počas prechodu vstupných dát ešte pred samotným kódovaním.

### 2.1 Kodér

Statické Huffmanovo kódovanie pracuje so zoznamom frekvencií výskytov jednotlivých kódovaných symbolov. Tento zoznam musí byť získaný ešte pred samotným kódovaním. Zoznam frekvencií výskytov môže byť odhadnutý, alebo daný, ak kódujeme dáta s vopred známou charakteristikou. Bežnejší prístup, ktorý bol využitý aj v tejto práci, je prechod vstupných dát a zostavenie zoznamu frekvencií výskytu symbolov. Kódovanie symbolov je vykonané až počas druhého prechodu vstupných dát.

Zoznam frekvencií symbolov obsahuje počty výskytov jednotlivých symbolov vo vstupných dátach. Tento zoznam frekvencií je celý (aj nulové hodnoty) zapísaný na začiatok výstupného súboru. Zápis zoznamu frekvencií oproti zápisu Huffmanovho stromu bol zvolený z dôvodu jednoduchosti. Každá položka zoznamu zaberá 4B vo výstupnom súbore. Priestor 4B na uloženie počtu výskytov symbol umožňuje uložiť až číslo $2^{32}$, čo umožňuje kódovať súbory s veľkosťou až 4GB.

Huffmanov strom bol vytvorený za pomoci std::priority_queue[1]. Táto fronta s prioritou má konfigurovateľné porovnanie, podľa ktorého sa určuje poradie vo fronte. Na zostavenie Huffmanovho stromu potrebujeme postupovať od symbolov s najnižšou frekvenciou a preto bolo porovnávanie vo fronte s prioritou nastavené tak, uzol s nižšou frekvencou mal vyššiu prioritu. Na začiatku sú do fronty vložené uzly odpovedajúce jednotlivým symbolom. Kým fronta obsahuje viac ako len koreňový uzol, vyberú sa z fronty dva uzly s najnižšími frekvenciami, spoja sa do rodičovského uzlu a ten sa vloží do fronty s prioritou.

Zakódovaný vstup je na výstup zapisovaný po bajtoch. Ale môže sa stať, že zakódovaný vstup nevypĺňa celý posledný bajt a na výstup sú v poslednom bajte zapísané nulové bity, ktoré nie sú súčasťou kódu. Tento problém je vyriešený tým, že do výstupu je zapísaný celý frekvenčný zoznam a dekodér si vie spočítať, koľko symbolov má dekódovať a teda vie, kedy má začať ignorovať koncové nadbytočné nulové bity.

### 2.2 Dekodér

Dekodér pri statickom Huffmanovom kódovaní najskôr načíta zoznam frekvencí z úvodu zakódované vstupu. Zoznam frekvencií má pevnú veľkosť a poradie, takže dekodér presne vie, koľko bajtov má načítať a ako ich interpretovať. Po načítaní zoznamu frekvencií je zostavený Huffmanov strom rovnakým spôsobom, ako s prípade kodéru.

Hlavný dekódovací cyklus má pevne daný počat opakovaní. Tento počet opakovaní odpovedá počtu symbolov, ktoré sa majú dekódovať a dá sa zistiť z koreňového uzlu Huffmanovho stromu. Frekvencia v koreňovom uzli odpovedá súčtu frekvencií všetkých symbolov a teda počtu symbolov, ktoré majú byť dekódované.

[1] https://en.cppreference.com/w/cpp/container/priority_queue

V dekódovacom cykle sú postupne načítavané bity zo vstupného súboru. Tieto bity sú ako znaky 0 a 1 pridávané do pomocnej premennej a v každom cykle je testované, či už sme načítalí celý kód nejakého symbolu. Ak áno, tak je odpovedajúci symbol zapísaný na výstup a pomocná premenná sa vynuluje. Ak nie, tak pokračujeme v pridávaní znakov do tejto premennej.

### 2.3 It's not a bug, it's a feature

Počas testovania projektu bolo objavené nečakané správanie statického Huffmanovho kódovania bez modelu. Ak vstupný súbor obsahuje všetky bajty rovnakej hodnoty, potom zakódovaný súbor obsahuje len zoznam frekvencií a dosahuje sa bps blízke nule. Dekóder je však schopný obnoviť pôvodný súbor.

Táto význačná kompresia je spôsobená tým, že ak zoznam frekvencií výskytov obsahuje len jednu nenulovú hodnotu, potom aj Huffmanov strom obsahuje len jeden uzol. Neošetrenie tohto okrajového prípadu pri prechode stromom a získavaní Huffmanovho kódu spôsobilo, že výsledný kód je prázdny reťazec. Kodér zapisuje tento kód do výstupného súboru, ale stále je to len prázdny reťazec.

Spôsob, akým si dekodér poradí spočíva v tom, že počet symbolov, ktoré sa majú dekódovať, je vopred známy. Dekodér načíta zoznam frekvencií a zostaví Huffmanov strom. Frekvencia uložená v koreňovom uzli odpovedá počtu všetkých symbolov, ktoré majú byť dekódované. Dekódovací cyklus teda trvá presne daný počet cyklov. Dekodér sa síce pokuša čítať bity zo zakódovaného súboru, ale koniec súboru ho nezastaví. A keďže Huffmanov strom obsahuje jediný uzol, na výstup sa v každom cykle dostane ten správny symbol.

## 3 Adaptívne Huffmanovo kódovanie

Nevýhoda statického (respektíve semistatického) Huffmanovho kódovanie sú dva prechody vstupnými dátami. Túto nevýhodu odstraňuje adaptívne Huffmanovo kódovanie, ktoré vytvára a modifikuje Huffmanov strom adaptívne, v jednom prechode spolu so samotným kódovaním [2]. Základom algoritmu tohto kódovania je správna práca so stromom. Vizualizácia je na stránke „Visualizing Adaptive Huffman Coding“ [3].

### 3.1 Kodér

Kodér pri adaptívnom Huffmanovom kódovní začína so stromom, ktorý obsahuje jediný uzol – tzv. NYT (not yet transmitted) uzol. Tento uzol je miesto, kde je strom rozšírený o dva dcérske uzly v prípade, že na vstupe je symbol, ktorý sa ešte nenachádza v strome. Pravý dcérsky uzol potom obsahuje nový symbol a ľavý dcérsky uzol sa stane novým NYT uzlom.

Pre rýchle overenie toho, že sa symbol už nachádza v strome, sú všetky doterajšie symboly držané v std::map[2] spolu s uzlami stromu, ktoré im prislúchajú.

V prípade, že sa kódovaný symbol už nachádza v strome, je jeho kód zapísaný do výstupného súboru. Ak je na vstupe nový symbol, je tento symbol vložený do stromu. Na výstup je najskôr zapísaný kód odpovedajúci NYT uzlu. Tento kód sa nezhoduje s kódom žiadneho symbolu a je poznávacím znakom pre dekodér, že bude nasledovať 1B, ktorý reprezentuje nový symbol. Teda kóder po zapísaní kódu NYT uzlu, zapíše na výstup nezmenený nový symbol.

Po každom spracovaní vstupného symbolu je potrebné aktualizovať Huffmanov strom, aby bolo zaručené, že stále produkuje optimálne kódy. Aktualizácia začína uzlom, ktorý reprezentuje práve kódovaný symbol a postupuje až ku koreňovému uzlu. Pre každý aktualizovaný uzol sa zisťuje, či existuje uzol, ktorý má rovnakú váhu, ale má vyššie poradie (order). Ak takýto uzol existuje, tak je zamenený s aktuálne aktualizovaným uzlom. Pri zámene sú ošetrené prípady, aby nenastala zámena koreňového uzlo a aby nenastala zámena uzlu som svojím rodičom. Súčasťou aktualizácie je aj zvýšenie váhy uzlu. Zvýši sa váha uzlu, ktorý reprezentuje aktuálne kódovaný symbol a taktiež sa zvýši váha všetkých jeho rodičovských uzlov až po koreňový uzol.

Kóder ukončí svoju činnosť po prečítaní a zakódovaní celých vstupných dát. Na výstup je zapísaný posledný bajt, ktorý môže obsahovať nadbytočné nuly. Preto je na záver výstupného súboru pridaný ešte jeden bajt, ktorý popisuje, koľko bitov je platných v predposlednom bajte.

### 3.2 Dekodér

Dekodér pri adaptívnom Huffmanovom kódovaní, podobne ako kodér, začína so stromom, ktorý obsahuje iba uzol NYT. Pred dekódovaním sa ešte zistí veľkosť vstupného súboru, prečíta sa hodnota posledného bajtu súboru a podľa nej sa upraví použiteľná veľkosť súboru. Vstupný súbor je potom čítaný len po túto veľkosť a zvyšné nadbytočné nulové bity sa ignorujú.

[2] https://en.cppreference.com/w/cpp/container/map

V hlavnom dekódovacom cykle sú postupne načítané bity zo vstupného súboru a podľa ich hodnôt sa prechádza aktuálny Huffmanov strom až kým sa nenarazí na listový uzol. Ak nájdený listový uzol je NYT uzol, potom vieme, že vo vstupnom súbore bude nasledovať 1B, ktorý reprezentuje nezákodovanú hodnotu symbolu a táto hodnota je zapísaná aj do výstupného súboru. V prípade, že nájdený listový uzol obsahuje symbol, potom je tento symbol z uzlu získaný a zapísaný do výstupného súboru.

Po zápise symbolu do výstupného súboru ešte nasleduje aktualizácia Huffmanovho stromu rovnakým spôsobom ako v prípade kodéru.

# 4 Model

V tejto práci bol použitý model založený na rozdiele medzi susednými vzorkami definovaný nasledovne:

$$y_n = \begin{cases} x_n - x_{n-1} & \text{for n} > 1 \\ x_1 & \text{for n} = 1 \end{cases}$$

Model funguje dobre v prípade, že hodnoty kódovanej sekvencie sú si podobné [4].

# 5 Merania

Merania boli vykonané na školskom serveri merlin na dátach priložených k zadaniu projektu.

## 5.1 Kompresná účinnosť

Kompresná účinnosť je vyjadrená ako priemerný počet bitov potrebný na zakódovanie bajtu obrazu.

| | static | static -m | adaptive | adaptive -m |
|---|---|---|---|---|
| nk01.raw | 6.53 | 6.09 | 6.51 | 6.07 |
| hd12.raw | 6.22 | 4.42 | 6.21 | 4.39 |
| hd09.raw | 6.68 | 4.71 | 6.67 | 4.69 |
| hd08.raw | 4.26 | 3.55 | 4.24 | 3.53 |
| hd07.raw | 5.66 | 3.88 | 5.62 | 3.86 |
| hd02.raw | 3.73 | 3.36 | 3.71 | 3.34 |
| hd01.raw | 3.90 | 3.43 | 3.88 | 3.41 |
| priemer | 5.28 | 4.21 | 5.26 | 4.18 |

Tabuľka 1: Kompresná účinnost vyjadrená v jednotkách bpc

## 5.2 Čas kompresie

Čas kompresie je súčet časov potrebných na zakódovanie vstupu a následné dekódovanie vstupu.

| | static | static -m | adaptive | adaptive -m |
|---|---|---|---|---|
| nk01.raw | 1881 | 1970 | 2215 | 1696 |
| hd12.raw | 1930 | 1311 | 2885 | 1024 |
| hd09.raw | 2181 | 1586 | 2979 | 1020 |
| hd08.raw | 1167 | 1031 | 1024 | 861 |
| hd07.raw | 1686 | 1137 | 2046 | 748 |
| hd02.raw | 1187 | 1127 | 1724 | 1168 |
| hd01.raw | 1249 | 1091 | 1745 | 1099 |
| priemer | 1612 | 1322 | 2088 | 1088 |

Tabuľka 2: Čas kompresie v milisekundách

# 6 Návod k prekladu

Aplikácia je napísaná v jazyku C++ a na jej preloženie je priložený súbor Makefile. Aplikácia je preložená pomocou príkazu *make*. Vzniknutý spustiteľný súbor má názov $huff_codec$ a spúšťa sa nasledovne: $./huff\_codec < argumenty >$. Pri spustení s argumentom -h je zobrazená nápoveda, ktorá obsahuje popis ďalších akceptovaných argumentov.

## Literatúra


[1] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[2] David Salomon. *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[3] Ben Tanen. Visualizing adaptive huffman coding. `http://ben-tanen.com/adaptive-huffman/`. Accessed: 2019-04-15.

[4] Zdeněk Vašíček. Úvod do problematiky kódování a komprese dat. `https://wis.fit.vutbr.cz/FIT/st/cfs.php?file=%2Fcourse%2FKKO-IT%2Flectures%2FKKO-01.pdf&cid=12830`. Accessed: 2019-04-13.