VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

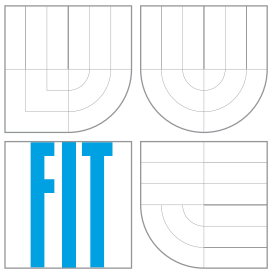
BEZEZTRÁTOVÁ KOMPRESE OBRAZU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAROSLAV NĚMEC
AUTHOR

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

BEZEZTRÁTOVÁ KOMPRESE OBRAZU

LOSSLESS IMAGE COMPRESSION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAROSLAV NĚMEC
AUTHOR

VEDOUCÍ PRÁCE — Ing. DAVID BAŘINA
SUPERVISOR

BRNO 2010

Abstrakt

Tato práce se zabývá bezeztrátovou kompresí obrazu. Je zde rozebrána celá problematika sestavení kodéru/dekodéru. V práci lze nalézt porovnání různých prediktorů, výběr vhodného barevného modelu a dva druhy entropického kódování. V závěru práce je uvedeno porovnání výsledků implementace se stávajícím formátem PNG.

Abstract

This thesis deals with lossless image compression. You can find all the process of assembling lossless image coder/decoder. There are described many predictors, color models and two entropy coders in this thesis. The results of thesis are compared and disscused with current lossless image format PNG at the end of thesis.

Klíčová slova

komprese, obraz, bezeztrátová, predikce, entropie

Keywords

compression, image, lossless, prediction, entropy

Citace

Jaroslav Němec: Bezeztrátová komprese obrazu, bakalářská práce, Brno, FIT VUT v Brně, 2010

Bezeztrátová komprese obrazu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Davida Bařiny.

.....
Jaroslav Němec
15. května 2010

Poděkování

Děkuji svému vedoucímu Ing. Davidu Bařinovi za rady a čas, které mi poskytnul. Rád bych poděkoval i své rodině za podporu při vypracovávání bakalářské práce.

© Jaroslav Němec, 2010.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Rozbor komprese **4**
2.1 Bezeztrátová komprese 4
2.2 Ztrátová komprese 5

3 Volba barevného modelu **7**
3.1 Model RGB 8
3.2 Model YUV 8
3.3 Model CREW/RCT 9
3.4 Model YF_rF_b 9
3.5 Model YC_oC_g 10
3.6 Model RDCT 11
3.7 Model SHIRCT 11
3.8 Model LOCO-I 12
3.9 Model NFS 12

4 Prediktory **13**
4.1 Lineární a nelineární prediktory 14
4.2 Prediktor nultého řádu 14
4.3 Prediktor prvního řádu 14
4.4 Prediktor druhého řádu 15
4.5 Prediktor třetího řádu 15
4.6 Obecné vyjádření prediktoru 16
4.7 Prediktor MED 17

5 Entropické kódování **18**
5.1 Statické Huffmanovo kódování 18
5.2 Adaptivní Huffmanovo kódování 22
5.3 Aritmetické kódování 23
5.3.1 Kódování 23
5.3.2 Dekódování 24
5.3.3 Implementace aritmetického kódování 25
5.3.4 E1 a E2 škálování 26
5.3.5 E3 škálování 26

6 Implementace knihovny **27**
6.1 Třída static_huffman . . . 27
6.2 Třída arithmetic_coder . . . 28
6.3 Spuštění programu . . . 28

7 Dosažené výsledky **30**
7.1 Porovnání prediktorů . . . 30
7.2 Porovnání barevných modelů . . . 32
7.2.1 RGB vs. LYUV . . . 32
7.2.2 RGB vs. YC_oC_g . . . 34
7.2.3 RGB vs. YF_rF_b . . . 35
7.2.4 RGB vs. RCT . . . 35
7.2.5 RGB vs. RDCT . . . 36
7.2.6 RGB vs. SHIRCT . . . 37
7.2.7 Porovnání účinnosti komprese dle barevného modelu . . . 38
7.3 Celkové posouzení komprese . . . 39
7.4 Úvaha nad bezeztrátovou kompresí . . . 40

8 Závěr **41**

A Obsah CD **43**

Kapitola 1

Úvod

Použití komprese má dva hlavní důvody. První důvod je úspora místa na diskové paměti. Někdo by mohl namítnout, že v současné době mají diskové paměti dostatečnou kapacitu. To je sice pravda, ale musíme si uvědomit, že zároveň s kapacitou diskové paměti roste i velikost uložených dat. Nepochybně je pádnějším argumentem přenos dat po počítačové síti. Zcela jistě si dokážeme představit dobu přenosu nekomprimovaného obrázku a dobu přenosu komprimovaného obrázku. Dalším příkladem mluvícím za vše je streaming videa. Uvažujme následující parametry. Přenášíme video v DVD-Video PAL kvalitě, tj. 720×576 pixelů, barevná hloubka 24 bitů, 25 snímků za sekundu. Z toho dostáváme, že datový tok tohoto streamu bude přibližně 240 Mbps. Ale při použití komprese můžeme tento tok významně snížit.

Možná se ptáte, co nám umožňuje data komprimovat. Odpověď je jednoduchá. U všech typů dat se setkáváme s *redundancí*. Do češtiny se překládá jako nadbytečnost. Této vlastnosti plně využíváme při bezeztrátové kompresi. Při ztrátové kompresi využíváme pojmu *irelevance*. Nejedná se přímo o vlastnost dat, ale o nedokonalost lidského sluchu nebo zraku. Bezeztrátová a ztrátová komprese jsou dva způsoby komprese. Oba způsoby jsou popsány v kapitole č. 2.

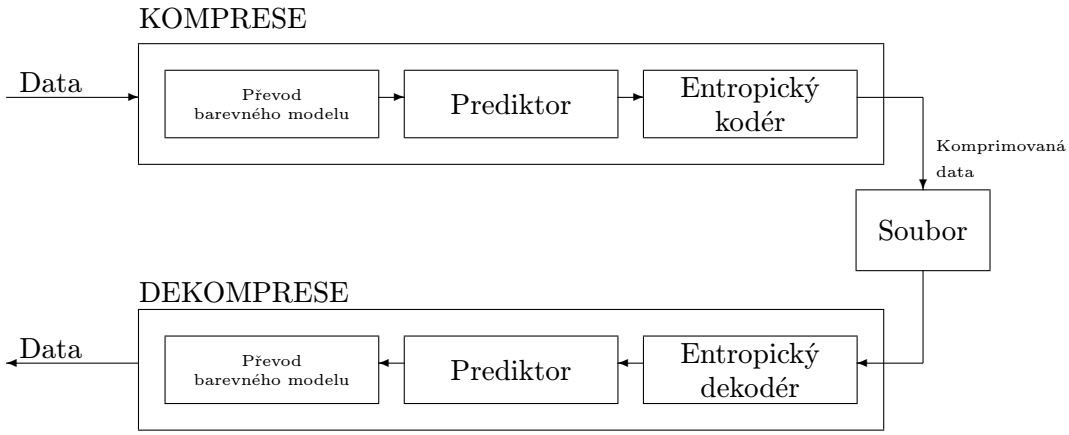
Faktory, které ovlivňují kompresi, jsou popsány v kapitolách č. 3 – 5. Kombinace použití těchto faktorů nám zajišťuje nejlepší výsledky komprese. Vlastnosti obecné komprese jsou diskutovány v kapitole č. 2. Je zde také uvedeno rozmezí hodnot kompresního poměru, kterého dosahují obě komprese (bezeztrátová a ztrátová). Základní barevné modely, jejich vlastnosti a způsoby převodu jsou popsány v kapitole č. 3. V této kapitole se můžete také dočíst, jak se skládají barvy. Následující kapitola nám předvede, jak se dá efektivně využít již zpracovaných pixelů k odhadu současného. Kapitola č. 5 pojednává o způsobu zapsání dat, tak aby byly upřednostněny znaky s vyšší četností. Samozřejmě existují další pomocné algoritmy a způsoby kódování.

Samotný postup implementace a návrhu bezeztrátové komprese je uveden v kapitole č. 6. Návod ke spuštění programu, význam přepínačů a příklady spuštění jsou uvedeny v podkapitole č. 6.3. Pro srovnání vlastností mnou navržené a implementované komprese s formátem PNG je vyhrazena kapitola č. 7. Závěr shrnuje dosažené výsledky komprese a celé bakalářské práce.

Kapitola 2

Rozbor komprese

Komprese se z pohledu zpětné rekonstrukce dat rozděluje na bezeztrátovou a ztrátovou. Pro bezeztrátovou kompresi je typické, že lze data rekonstruovat bez ztráty jakékoliv informace. Používáme ji tehdy, kdy nesmí dojít ke ztrátě dat (dokumenty, spustitelné programy atd.). Protikladem bezeztrátové komprese je komprese ztrátová. Při ztrátové kompresi nejsme schopni úplně zrekonstruovat komprimovaná data. Odstraňuje data, která mají v daném kontextu malý význam. Její použití je například při streamování videa, ukládání fotek atd. Ztrátová komprese má typicky vyšší *kompresní poměr* než komprese bezeztrátová. Není to však pravidlem. Kompresní poměr vyjadřuje podíl velikosti výchozích dat před kompresí vůči velikosti dat komprimovaných.

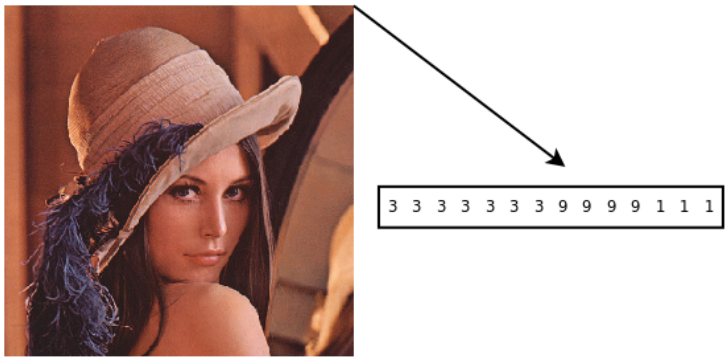


Obrázek 2.1: Obecný postup komprese

2.1 Bezeztrátová komprese

Bezeztrátová komprese odstraňuje redundanci. Jak bylo výše zmíněno, redundance znamená nadbytečnost. Z toho vyplývá, že po odstranění této redundance, se zdají data stejná. Prakticky to znamená, že přestože odstraníme tyto redundantní data, jsme stále schopni

rekonstruovat komprimovaná data bez sebemenší ztráty. Jinými slovy se dá říci, že tato komprese odstraňuje statistickou nadbytečnost. V tomto směru má omezenější schopnost komprese. To je tedy hlavní příčinou nižšího kompresního poměru.



Obrázek 2.2: Hodnoty pixelů v obrázku

Na obrázku vidíme, že pixely mají od levého horního rohu hodnotu p(n) = (3, 3, 3, 3, 3, 3, 3, 9, 9, 9, 9, 1, 1, 1). Tuto posloupnost můžeme napsat i kratším vyjádřením p(n) = (7,3)(4,9)(3,1). Toto vyjádření nám umožňuje zpětně rekonstruovat výchozí data aniž bychom cokoliv ztratili. Celý postup komprese se sestává ze dvou základních kroků. Prvním krokem je předpříprava dat. Aplikují se různé prediktory, převody mezi barevnými modely atd. Druhým krokem je *entropické kódování*. Toto kódování používá již předzpracovaná data a pro každý vzorek dat tvoří kód, který jej bude reprezentovat. Kódování může mít dva druhy. Statické a dynamické. U statického kódování probíhají oba kroky postupně. U dynamické probíhají zároveň. Velikost kompresního poměru se dá ovlivnit použitým barevným modelem, typem prediktoru a způsobem kódování. Mezi běžně používané formáty bezeztrátové komprese patří např. ZIP nebo pro Unix gzip. Pro kompresi obrázku se používá formát PNG. Kompresní poměr se běžně pohybuje v rozmezí 1 – 2,5 : 1.

2.2 Ztrátová komprese

Ztrátová komprese odstraňuje irelevanci. Irelevantní datová složka má v daném kontextu velmi malý význam, a proto je její nepřítomnost nepostřehnutelná. Jejím odstraněním výsledek podstatně nezměníme. Důležitým poznatkem ovšem je, že neexistuje žádný mechanismus, který by dokázal zcela rekonstruovat komprimovaná data na výchozí signál. Je možná pouze částečná rekonstrukce obrazu. Tuto kompresi můžeme použít v případě, že nám nevadí ztráta určité informace. Výsledek je určitým způsobem podobný a jeví se stejně jako výchozí data, přestože stejný není. Využívá se především nedokonalosti lidského zraku (obrázky) a sluchu (zvuk). V porovnání s bezeztrátovou kompresí má mnohem silnější schopnost komprese. Hodnoty kompresního poměru jsou 2 – 10 : 1. Na následujícím příkladu budu demonstrovat odstranění irelevantní složky. Mějme obraz ve stupních šedi. Hodnoty p(n) = (7,6,6,6,6,5,5,6,7) vyjadřují hodnoty pixelů v okolí určitého bodu. Průměrná hodnota v této sekvenci je 6. Pokud bychom přepsali tuto sekvenci na sekvenci g(n) = (6,6,6,6,6,6,6,6,6), můžeme ji snadno zkomprimovat do tvaru (9,6). Prvním pohledem na sekvence p(n) a g(n) zjistíme, že se tyto sekvence výrazně neliší. Rozdíl je v tom, že sekvenci p(n) bychom nemohli tak snadno zkomprimovat jako sekvenci g(n). Malé změny, které jsme ze sekvence

p(n) odstranili, již nejsou rekonstruovatelné ze sekvence g(n). Vizuálně se tyto dvě sekvence jeví stejně, přestože stejné nejsou. Můžeme zvolit kvalitu výsledku, podle které se odvíjí kompresní poměr. Obecně platí, že čím větší kvalitu zvolíme, tím menší bude komprese. Pro kompresi obrázků se hojně využívá formát JPEG, pro zvuk se používá formát MP3 a pro video (i streamování) formát MPEG.

Kapitola 3

Volba barevného modelu

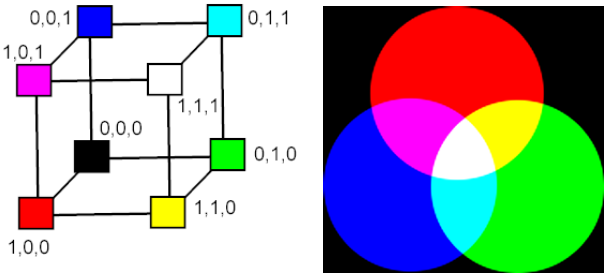
Barevný model je ve své podstatě matematický model popisující způsob reprezentace barvy jako n-tici čísel. Pokud má barevný model tři barevné složky, jedná se o trojici čísel. Tři nebo čtyři barevné složky jsou typické, ne však jediné možné. Každý barevný model dokáže reprezentovat určité množství barev. Všechny barvy, které je daný barevný model schopen reprezentovat, jsou vytvořeny z malé množiny barev základních. Úkolem každého barevného modelu je co nejvěrněji zobrazit danou barvu. Tuto snahu limitují dvě základní omezení – přesnost modelu a složitost modelu. V přírodě je barva tvořena směsicí světla různých vlnových délek. Pokud bychom chtěli naprosto věrně reprodukovat barvy nějakého objektu, bylo by nutné znát spektrální křivku každého bodu tohoto objektu. Pro základní zobrazení existují dva způsoby míchání barev – *aditivní* a *subtraktivní*. Základní rozdíl mezi těmito způsoby je, zda se jednotlivé barevné složky sčítají či odčítají. Další rozdíl je v práci se světlem, nebo s jeho odrazem. Obě metody mají rozdílné použití. Kde potřebujeme barvu zobrazit na černé ploše, použijeme aditivní míchání. Naopak pro bílou plochu použijeme míchání subtraktivní.

- Aditivní míchání je podobné jako skládání barevného světla. Přidáním nového odstínu se výsledná barva zesvětlí. Tento způsob používají monitory, displeje a projektory. Tyto modely pracují se světelnými zdroji barev.
- Subtraktivní míchání funguje opačně. Přidáním nového odstínu vznikne barva tmavší. Tento způsob se využívá u tiskáren. Tyto modely pracují s odrazem světla – bílého.

Pro účely komprese je volba vhodného barevného modelu klíčová. Vhodný barevný model dokáže významně zvýšit kompresní poměr. Je nutné si říci, že každý bod v obraze (pixel) je dán složením tří hodnot jednotlivých barevných složek. Vhodný barevný model dokáže oddělit intenzitu od chromatických složek, a tak lépe zpracovat data pro prediktor. Mezi dva základní barevné modely patří model RGB a CMY. Při kompresi se hojně využívá barevný model YUV. Tento model je pouze součástí vnitřní implementace kodéru, a je nutné provádět přepočet z modelu RGB na YUV a naopak. A protože je celá práce zaměřená na bezeztrátovou kompresi, musí být zcela dodrženo, aby tyto přepočty byly dokonale reversibilní. Jako další barevné modely jsem použil deriváty modelu YUV, které zaručují dokonalou reversibilitu. Existují i jiné deriváty modelu YUV, které reversibilitu nezaručují, a proto nejsou vhodné pro bezeztrátovou kompresi.

3.1 Model RGB

V úvodu této kapitoly bylo řečeno, že se u každého modelu všechny barvy skládají z barev základních. Pro model RGB [8] jsou to, jak napovídá název, červená, zelená a modrá. Filozofie modelu vychází z faktu, že lidské oko obsahuje tři druhy barevných čípků. Tyto čípky jsou citlivé na vlnové délky světla, které přibližně odpovídají červené (630 nm), zelené (530 nm) a modré (450 nm) barvě. Kombinací těchto barev můžeme získat téměř všechny barvy barevného spektra. Model RGB používá aditivní míchání. Je nutné ještě určit minimální a maximální hodnotu, kterých hodnoty barevných složek mohou nabývat. Pro potřeby počítačové grafiky může každá barevná složka nabývat 256 hodnot (0 – 255). Typicky se ukládají na 1 byte, což znamená, že každou barvu můžeme definovat třemi byty. Takto udávané barvy jsou označovány jako true colors. V rámci true colors můžeme tedy zobrazit 256^3 barev. K vyjádření hodnot barevných složek se velmi často používá hexadecimální číselná soustava (hodnoty 00–FF). Na obrázku vidíme jednotkovou krychli – jednu z možných reprezentací modelu RGB a princip aditivního míchání barev.



Obrázek 3.1: Jednotková krychle pro model RGB a princip aditivního míchání barev

Existují i další modely RGB. Nejznámější a nejrozšířenější je varianta sRGB, která je standardem Windows. Jsou zde přesně specifikovány tři základní barvy, bílý bod a gamma. Největší výhodou je, že odpovídá reálným možnostem zobrazení většiny monitorů, a proto se používá ve velkém i na internetu.

3.2 Model YUV

Je všeobecně známo a testováním ověřeno, že barevný model RGB není vhodný pro bezeztrátovou kompresi obrazu. A to hlavně kvůli *korelaci*. Korelací rozumíme velmi podobné hodnoty sousedních pixelů. Vyššího kompresní poměru můžeme dosáhnout převodem obrázku do jiného vhodného barevného modelu. Jak bylo výše zmíněno, tento převod musí být dokonale reversibilní. Pro tento převod jsou uvedeny následující vztahy :

$$\begin{aligned} Y &= 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B \\ U &= B - Y \\ V &= R - Y \end{aligned}$$

Výhodou tohoto modelu je oddělení jasové složky Y od barevných informací. Dalšími převody můžeme dosáhnout převodu RGB do LYUV [2].

$$\begin{aligned} Y_L &= G + [0,299/0.587 \cdot R + 0,114/0,587 \cdot B] \\ U_L &= B - [0,587 \cdot Y_L] \\ V_L &= R - [0,587 \cdot Y_L] \end{aligned}$$

Desetinná čísla v hranatých závorkách zaokrouhlujeme na nejbližší celočíselnou hodnotu. Dokonale reversibilního převodu dosáhneme aplikací následujících vzorců.

$$\begin{aligned} R &= V_L + [0,587 \cdot Y_L] \\ G &= Y_L - [0,299/0,587 \cdot R + 0,114/0,587 \cdot B] \\ B &= U_L + [0,587 \cdot Y_L] \end{aligned}$$

Nyní můžeme převést model RGB na YUV, data zakomprimovat a při dekomprimaci musíme převést dekomprimovaná data do modelu RGB.

3.3 Model CREW/RCT

Nyní si uvedeme další barevný model vhodný pro komprimaci. Tento barevný model získáme transformací CREW/RCT [3]. Jeho použití je široké. Například v standardu JPEG 2000. Tento převod opět slouží k dekorelaci, která se vyskytuje u modelu RGB. Budou zde uvedeny vztahy pro přepočet jednotlivých složek R, G, B na složky Y_0, Y_1, Y_2. V níže uvedených rovnicích se objevují operátory násobení a dělení, které jsou implementovány pomocí bitových posuvů. Rovnice pro převod z modelu RGB vypadají takto:

$$\begin{aligned} Y_0 &= \left\lfloor \frac{R + 2 \cdot G + B}{4} \right\rfloor \\ Y_1 &= B - G \\ Y_2 &= R - G \end{aligned}$$

Pro zpětný převod se používá následujích vztahů:

$$\begin{aligned} R &= Y_2 + G \\ G &= Y_0 - \left\lfloor \frac{Y_2 + Y_1}{4} \right\rfloor \\ B &= Y_1 + G \end{aligned}$$

Zde si musíme dát pozor. Poslední tři vzorce jsou zde uvedeny v pořadí, jak jsou interpretovány v modelu RGB. Při zpětné transformaci nejdříve vypočítáme složku G, poté R a nakonec B.

3.4 Model YF_rF_b

Dalším hojně používaným barevným modelem je model YF_rF_b [9]. Pro snažší výpočet a dosažení reversibility se zde zavádí pomocná proměnná t. Složky výsledného barevného

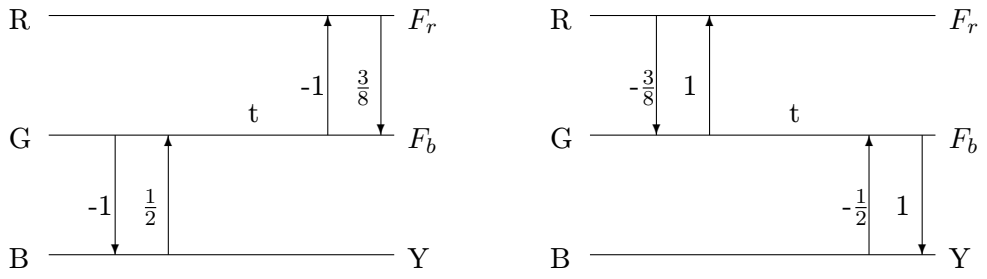
modelu jsou tedy Y, F_r, F_b, nikoliv t. Ukažme si převodní vztahy. Převod z modelu RGB:

$$\begin{aligned} Y &= t + \frac{3}{8} \cdot F_b \\ F_r &= R - B \\ F_b &= G - t \\ t &= b + \frac{F_r}{2} \end{aligned}$$

Převod do modelu RGB:

$$\begin{aligned} R &= B + F_r \\ G &= F_b + t \\ B &= t - \frac{F_r}{2} \\ t &= Y - \frac{3}{8} \cdot F_b \end{aligned}$$

I zde si musíme dát pozor na pořadí. Nejprve vypočítáme hodnotu pomocné proměnné t, poté složky B, G, R. Vztahy mezi jednotlivými složkami se dají vyjádřit pomocí sítí, kdy je na levé straně výchozí barevný model a na pravé straně je cílový barevný model. Mezi oběma modely vedou vodorovné a svislé linky propojující jednotlivé barevné složky. Linky jsou opatřeny konstantou, kterou se násobí daný kanál. Ukažme si příklad takové sítě pro tento barevný model. Tato síť zde bude ukázána pro názornost. U následujícího barevného modelu by šla také použít.



Obrázek 3.2: Převodní síť mezi modely RGB a YF_rF_b

3.5 Model YC_oC_g

U tohoto barevného modelu [5] se opět zavádí pomocná proměnná t. Považuji za důležité říct, že barevných modelů je celá řada, a proto by opravdu postrádalo smysl zde popisovat všechny modely. Převodní vztahy pro převod z modelu RGB jsou zde:

$$\begin{aligned} Y &= t + \frac{C_g}{2} \\ C_g &= G - t \\ C_o &= R - B \\ t &= B + \frac{C_o}{2} \end{aligned}$$

Zpětný převod do modelu RGB provedeme pomocí těchto vztahů:

$$\begin{aligned} R &= B + C_o \\ G &= C_g + t \\ B &= t - \frac{C_o}{2} \\ t &= Y - \frac{C_g}{2} \end{aligned}$$

Tolik k barevnému modelu YC_oC_g. Porovnání všech implementovaných modelů a diskuzi o vhodnosti každého modelu lze nalézt v kapitole č. 7.

3.6 Model RDCT

Převod z RGB do RDCT [4] :

$$\begin{aligned} C_1 &= t + [0,156597 \cdot C3 + 0,507306 \cdot C2] \\ C_2 &= -B - [0,317837 \cdot C3 - 0,577350 \cdot t] \\ C_3 &= -G + [0,292893 \cdot B + 0,408248 \cdot t] \\ t &= R + [0,449490 \cdot G + 0,282561 \cdot B] \end{aligned}$$

Převod z RDCT do RGB :

$$\begin{aligned} R &= t - [0,449490 \cdot G + 0,282561 \cdot B] \\ G &= -C3 + [0,292893 \cdot B + 0,408248 \cdot t] \\ B &= -C2 - [0,317837 \cdot C3 - 0,577350 \cdot t] \\ t &= C1 - [0,156597 \cdot C3 + 0,507306 \cdot C2] \end{aligned}$$

3.7 Model SHIRCT

Převod z RGB do SHIRCT [4] :

$$\begin{aligned} C_1 &= B + G + \frac{t+1}{2} \\ C_2 &= t + \frac{3 \cdot C3 + 2}{4} \\ C_3 &= -G + \frac{C1 + \frac{t}{8} + 1}{2} \\ t &= R - \frac{G + B + 1}{2} \end{aligned}$$

Převod z SHIRCT do RGB :

$$\begin{aligned} R &= t + \frac{G + B + 1}{2} \\ G &= -C3 + \frac{C1 + \frac{t}{8} + 1}{2} \\ B &= C1 - G - \frac{t+1}{2} \\ t &= C2 - \frac{3 \cdot C3 + 2}{4} \end{aligned}$$

3.8 Model LOCO-I

Převodní vztahy [10] jsou velice jednoduché, ale i přesto jsem s tímto modelem dosáhnul nejlepších výsledků. Model LOCO-I je součástí ISO/ITU standardu pro bezeztrátovou kompresi obrazu - JPEG-LS.

$$\begin{aligned} C_1 &= R - G \\ C_2 &= G \\ C_3 &= B - G \end{aligned}$$

Převod z LOCO-I na RGB :

$$\begin{aligned} R &= C_1 + G \\ G &= C_2 \\ B &= C_3 + G \end{aligned}$$

3.9 Model NFS

Převod z RGB do NFS [6]:

$$\begin{aligned} O_1 &= \left\lfloor \frac{R+G+B}{3} + 0,5 \right\rfloor \\ O_2 &= \left\lfloor \frac{R-B}{2} + 0,5 \right\rfloor \\ O_3 &= B - 2 \cdot G + R \end{aligned}$$

Převod z NFS do RGB:

$$\begin{aligned} R &= O_1 + O_2 + O_3 - \left\lfloor \frac{O_3}{2} + 0,5] - \left\lfloor \frac{O_3}{3} + 0,5 \right\rfloor \right. \\ G &= O_1 - \left\lfloor \frac{O_3}{3} + 0,5 \right\rfloor \\ B &= O_1 - O_2 + \left\lfloor \frac{O_3}{2} + 0,5 \right\rfloor - \left\lfloor \frac{O_3}{3} + 0,5 \right\rfloor \end{aligned}$$

Kapitola 4

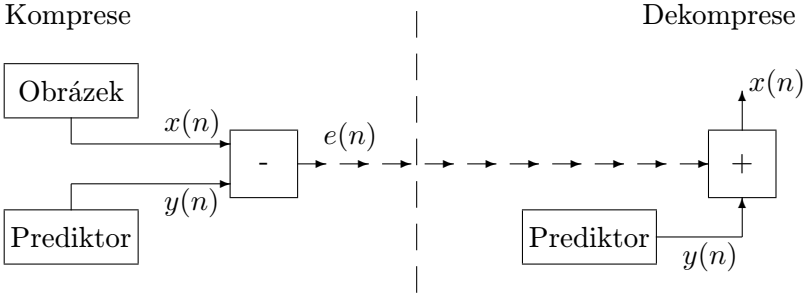
Prediktory

V obrazových datech jsou si sousední pixely velmi často podobné. To je důvod, proč se při bezeztrátové kompresi používají prediktory. Z jejich názvu lze soudit, že se nějakým blíže nespecifikovaným způsobem snaží odhadnout hodnotu následujícího pixelu. Odhadovanou hodnotu určuje prediktor z již zpracovaných pixelů. Pokud prediktor dobře určuje hodnotu následujícího pixelu, liší se skutečná hodnota pixelu od odhadované hodnoty minimálně. Přenáší se rozdíl mezi skutečnou hodnotou a odhadem. Tento rozdíl se nazývá chyba predikce. S použitím ideálního prediktoru by byly všechny přenášené hodnoty pouze hodnoty 0. Což by znamenalo, že prediktor dokáže uhodnout hodnotu naprosto bez chyby. Následující rovnice vyjadřují vztahy mezi odhadovanou hodnotou, skutečnou hodnotou a predikcí. A to při kompresi a dekompresi.

Kroky	Komprese	Dekomprese
1	Odhadovaná hodnota $y(n)$	Odhadovaná hodnota $y(n)$
2	Skutečná hodnota $x(n)$	Chyba predikce $e(n)$
3	Chyba predikce $e(n) = x(n) - y(n)$	Skutečná hodnota $x(n) = y(n) + e(n)$

Tabulka 4.1: Rovnice pro predikci

Z tabulky vidíme posloupnost kroků, které musíme provést. Při kompresi nejprve musíme určit z předešlých hodnot odhadovanou hodnotu, poté si zjistíme skutečnou hodnotu pixelu. Výsledkem je rozdíl (chyba predikce), který zakódujeme. Při dekompresi je postup opačný. Opět prediktor určí odhad hodnoty, ale poté si načteme chybu predikce a součtem těchto dvou hodnot dostaneme skutečnou hodnotu pixelu v obraze.



Obrázek 4.1: Výpočet predikce při kódování/dekódování

Nespornou výhodou je, že predikce dokáže produkovat hodnoty s vysokou entropií. Jednoduše redukuje výchozí hodnoty na jiné hodnoty s vyšší četností. To nemusí být na první pohled jasné. Celou tuto skutečnost se pokusím ukázat na následujícím obrázku. Uvažujme prediktor prvního řádu, který si pamatuje předchozí hodnotu a tu prohlašuje za odhadnutou hodnotu. Na počátku se hodnota prediktoru inicializuje na hodnotu 0. Obrázek procházíme od levého spodního rohu směrem doprava nahoru.

$x(n)$

16	17	18
13	14	15
10	11	12

$y(n)$

15	16	17
12	13	14
0	10	11

$e(n)$

1	1	1
1	1	1
10	1	1

Obrázek 4.2: Praktická ukázka predikce

Z obrázku lze názorně vyčíst, jak je predikce účinná. Toto je samozřejmě ukázkový příklad. Ve skutečnosti mohou být hodnoty pixelů jiné. V první matici vídíme skutečné hodnoty pixelů obrazu. Na počátku je hodnota prediktoru 0. Hodnota pixelu v levém dolním rohu je 10. Chyba predikce je tedy 10. Prediktor se nastaví na hodnotu 10 (již zpracovaný předchozí pixel). Další hodnota je 11. Chyba predikce je tedy 1. A takto postupujeme celým obrázkem a počítáme chybu predikce. Zde jasně vidíme, že pokud bychom kódovali obrázek bez použití predikce, museli bychom zakódovat a přiřadit kód devíti zcela rozdílným hodnotám. S použitím prediktoru budeme muset zakódovat pouze hodnoty dvě. To je významná úspora místa. O to se v bezeztrátové kompresi obrazu snažíme. Při dekódování vycházíme ze třetí matice. V následujících podkapitolách si ukážeme jednotlivé druhy prediktorů.

4.1 Lineární a nelineární prediktory

Lineární prediktor je lineární kombinací okolních pixelů. To platí pro všechny uvedené prediktory v této práci. Pouze prediktor MED patří do skupiny nelineárních prediktorů.

4.2 Prediktor nultého řádu

Prediktory se rozlišují podle toho, kolik již zpracovaných hodnot potřebují k predikování hodnoty. Nultý prediktor nepotřebuje k predikci žádnou hodnotu. Jeho hodnota je nastavená pořád na 0. Což v konečném důsledku znamená, že žádným způsobem nepomáhá zvyšovat kompresní poměr. Tento prediktor nemá žádné praktické využití. Avšak jeho funkci využívají v jistých krajních případech prediktory vyšších řádů.

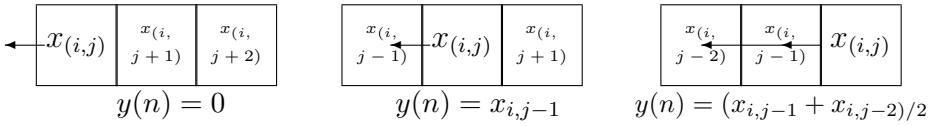
4.3 Prediktor prvního řádu

Jedná se o nejjednodušší prediktor. Jako predikovanou hodnotu bere hodnotu předešlého pixelu. Potřebuje (jako každý prediktor) počáteční inicializaci. V krajním bodě se jeho hodnota nastavuje na 0 (funkce prediktoru nultého řádu). Mohlo by se zdát, že není moc účinný, ale opak je pravdou. I tak jednoduchý prediktor dokáže významně zvýšit kompresní poměr. Jeho činnost je ukázaná na obrázku č. 4.2. U tohoto prediktoru existuje i jeho

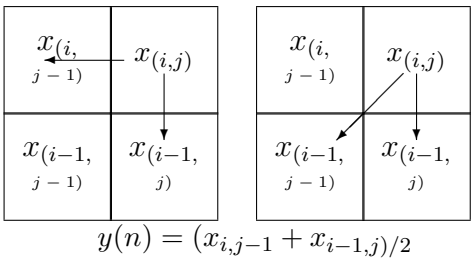
vertikální varianta, kdy se jako predikovaná hodnota bere hodnota pod současným pixelem, nikoliv vedle.

4.4 Prediktor druhého řádu

Tento prediktor používá k odhadu hodnoty dvě předešlé hodnoty. V základním tvaru může mít dvě varianty. Ovšem i ty v sobě v jistých situacích obsahují prediktor prvního řádu. První varianta se dívá na dvě předešlé hodnoty a násobí je vhodnými koeficienty. Nejčastěji se spočítá průměr z těchto dvou hodnot. Problémem je opět inicializace. Na začátku se prediktor opět inicializuje na 0. Při dalším kroku už má prediktor k dispozici jednu předešlou hodnotu, avšak ke své činnosti potřebuje hodnoty dvě. Zde přichází na řadu prediktor prvního řádu. Celá problematika se dá řešit mnoha způsoby. Na následujících dvou obrázcích jsou vyobrazeny pouze dva návrhy situací. Tyto návrhy popisují přístup k predikci v horizontálním směru a ve směru vertikálním. Třetím způsobem může být kombinace dvou předešlých.



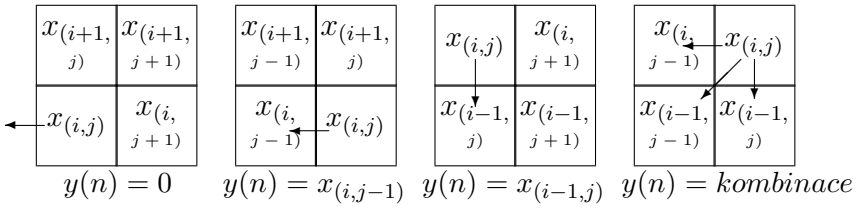
Obrázek 4.3: Horizontální přístup



Obrázek 4.4: Smíšený přístup

4.5 Prediktor třetího řádu

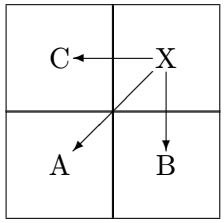
Tento prediktor využívá k určení predikované hodnoty tři hodnoty. Toho lze dosáhnout pouze v případě, že se nenacházíme v krajních bodech. Situace je naznačena na obrázku č. 4.5. Pokud se nacházíme úplně vlevo dole, predikce se inicializuje na 0 (první část obrázku). Pokud procházíme první řádek obrázku, využíváme prediktor prvního řádu ve směru horizontálním (druhá část obrázku). Pokud jsme úplně vlevo na vyšším než první řádku, využíváme prediktor prvního řádu ve směru vertikálním (třetí část obrázku). Ve všech ostatních případech k predikci využíváme levého souseda na stejné úrovni, spodního souseda a souseda na diagonále vlevo dole (čtvrtá část obrázku).



Obrázek 4.5: Prediktor třetího řádu

4.6 Obecné vyjádření prediktoru

Prediktorů existuje celá řada. Všechny výše uvedené přístupy k predikování hodnot můžeme různě kombinovat, nebo vymyslet přístup nový. Platí však, že prediktor je velice účinná technika pomáhající ke zvýšení kompresního poměru. Nedá se s určitostí říct, který prediktor je lepší. Obecně však ten, který kombinuje různé postupy. Výsledky dosažené pomocí různých prediktorů budou ukázány a diskutovány v kapitole Porovnání účinností prediktorů. Nyní si představme, že chceme predikovat hodnotu současného pixelu a máme k dispozici okolí tří bodů (viz obrázek). Tyto body si označíme A, B, C. Dále si určíme velikost konstant a, b, c. Obrázek ukazuje rozmístění bodů a tabulka výpočet odhadované hodnoty.



$$X = a \cdot A + b \cdot B + c \cdot C$$

Obrázek 4.6: Obecné vyjádření prediktoru

Prediktor	$y(n)$
0	žádná predikce
1	A
2	B
3	C
4	$A + B - C$
5	$A + (B - C)/2$
6	$B + (A - C)/2$
7	$(A + B)/2$

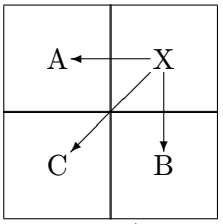
Tabulka 4.2: Rovnice pro predikci

Jiný typ prediktoru může jako predikovanou hodnotu prohlásit hodnotu nejbližší (ze tří sousedních hodnot) ke skutečné hodnotě. Účinnost prediktoru je nejlépe vidět na histogramu. Histogram totiž zobrazuje kolik pixelů v obraze má určitou hodnotu. Prediktor tyto hodnoty značně redukuje. Každý obrázek má jiný histogram a také histogram po použití stejného prediktoru u jiných obrázků vypadá jinak. Úkolem není najít nejlepší prediktor

(ani to nejde), ale spíše pro vybraný obrázek najít a zvolit nejvhodnější prediktor. Další možností je zvolit takový prediktor, který má statisticky lepší schopnost zvyšovat kompresní poměr než ostatní. Statistické vyhodnocení se může provádět na několika desítkách obrázků. Po sérii testů vybereme prediktor, u kterého víme, že nám u většiny obrázku zajistí nejvyšší kompresní poměr.

4.7 Prediktor MED

Speciálním typem prediktoru třetího řádu je prediktor MED [10] (median edge detector). Tento prediktor je používaný ve formátech HuffYUV/FFV1/LOCO–I/JPEG–LS. Při predikování hodnoty pracujeme s pixelem pod aktuálním pixelem, levým sousedem a sousedem na diagonále vlevo dole. Výsledek predikce závisí na velikosti jednotlivých sousedů. Výsledkem je $\hat{x} = (A, B, A + B - C)$. Označení pixelů A, B, C je jiné než na obrázku č. 4.6. Další obrázek ukazuje rozmístění pixelů a rovnice pro výpočet predikované hodnoty.



$$X = \begin{cases} min(A,B) & C \geq max(A,B) \\ max(A,B) & C \leq min(A,B) \\ A + B - C & \text{ostatní} \end{cases}$$

$$X = median(A, B, A + B - C)$$

Obrázek 4.7: Obecné vyjádření prediktoru

Kapitola 5

Entropické kódování

Mohlo by se zdát, že právě entropický kodér je stěžejní část celé komprese. Není tomu tak. Můžeme použít Huffmanovo kódování nebo aritmetické kódování. V samé podstatě neuvidíme velký rozdíl – mezi Huffmanovým kódováním a aritmetickým kódováním. Entropických kodérů existuje mnoho, ale dva výše zmíněné kodéry dosahují nejvyšších kompresních poměrů oproti ostatním entropickým kodérům. Celý proces předpřípravy dat, která se dají vhodně zakódovat, je nejdůležitejší. A takto předpřipravená data musí náš kodér vhodně zakódovat. Proto zde zavádíme pojem *entropický kodér*. Tento pojem značí takový kódér, který jistým způsobem upřednostňuje pixely (obecně data), které se v obrázku objevují častěji. A to tím způsobem, že pixely s vysokou četností zabírají méně místa než pixely s nízkou četností. Toto řešení je logické, protože pixel, který se v obrázku vyskytne například 100× chceme zakódovat způsobem, který zabere méně místa, než pixel, který se v obrázku vyskytne 2×. V následujících kapitolách bude rozebráno Huffmanovo kódování a kódování aritmetické. Obě tyto metody jsou hojně používané a pro účely bezeztrátové komprese vhodné.

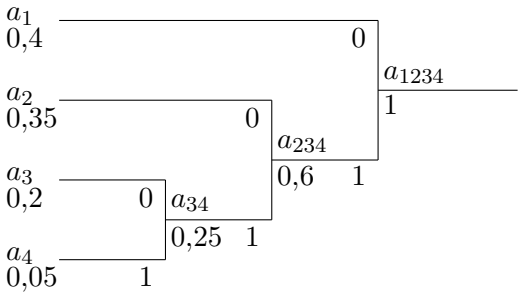
5.1 Statické Huffmanovo kódování

Tento způsob kódování vymyslel a navrhnul v roce 1952 David A. Huffman. Myšlenkou Huffmanova kódování [1] je reprezentovat symboly (pixely) s vyšší četností kratším binárním prexifovým kódem než pixely s nižší četností. Tato snaha se u statické varianty skládá ze dvou částí. První částí je sběr údajů o obrázku. Představme si, že obrázek převedeme do jiného barevného modelu, aplikujeme prediktor a máme sadu hodnot pixelů. V první části procházíme tyto hodnoty a ukládáme si je do struktury [hodnota, počet výskytů]. Tak zjistíme, kterých pixelů (dle hodnoty) je více či méně. Po úspěšném dokončení máme vstupní frontu obsahující prvky struktury a můžeme se pustit do druhé části.

Druhá část se týká vhodného zakódování sesbíraných statistik o obrázku. Jak bylo již zmíněno, budeme jednotlivé hodnoty pixelů reprezentovat jako binární prefixový kód. Vlastnost prefixového kódu je taková, že prefix nějakého symbolu (pixelu) není nikdy stejný jako prefix jakéhokoliv jiného symbolu. A k tomu využijeme binární strom. Další kroky budou vést k vytvoření binárního stromu. Tento strom bude mít jako listy symboly (pixely) obrazu. Binární strom sestavujeme odspodu a to tak, že vždy ze vstupní fronty vybíráme dva prvky (zároveň je ze vstupní fronty smažeme) s nejmenším ohodnocením (četností). Nad nimi vytvoříme nový uzel binárního stromu. Na nově vytvořený uzel napojíme, jako levého a pravého syna, právě dva vybrané prvky. Ohodnocení nově vytvořeného uzlu je dáno sou-

čtem ohodnocení dvou napojených prvků. Nově vytvořený prvek vkládáme do fronty. Tuto činnost opakujeme, dokud není ve vstupní frontě pouze jeden prvek. Pak máme celý strom vytvořený. Z implementačního hlediska je více než žádoucí, aby uzly měly ukazatel na uzel otcovský. V okamžiku, kdy máme strom sestavený, se provádí průchod stromem, který se provádí od shora. Na zjednodušeném příkladu si ukážeme postup při Huffmanově kódování.

Máme za úkol zakódovat čtyři vstupní znaky a_1, a_2, a_3, a_4. Pravděpodobnost výskytu je 0,4, 0,35, 0,2, 0,05. Vybereme dva znaky s nejmenší pravděpodobností (a_3, a_4) a ty spojíme v prvek a_{34}. Dále uvažujeme jen tři prvky a_1, a_2, a_{34}. Celý postup opakujeme, dokud nedostaneme pravděpodobnost 1.



Symbol	Kód
a_1	0
a_2	10
a_3	110
a_4	111

Obrázek 5.1: Ukázka Huffmanova kódování

Smyslem příkladu bylo ukázat, jak funguje entropické kódování. Pokud bychom chtěli zakódovat bez jakéhokoliv náznaku zohlednění pravděpodobností výskytu znaku zprávu skládající se ze symbolů $a_1a_2a_3a_4$, potřebovali bychom k tomu 4 byty (1 znak = 1 byte). S využitím Huffmanova kódování jsme schopni tuto zprávu binárně reprezentovat jako posloupnost 010110111, což je 9 bitů. To je velká úspora dat. V tomto případě je kompresní poměr roven $\frac{4 \cdot 8}{9} \doteq 3{,}6$. Zde také vidíme sílu prefixových kódů. Žádný znak nemá stejný prefix jako znak jiný. Proto je možno jednoznačně od sebe rozlišit jednotlivé znaky.

Další příklad detailně ukáže práci se vstupní frontou a také tvorbu binárního stromu. Naším úkolem bude posbírat statistiku o jednotlivých pixelech v obrázku, sloučit do množin pixely stejné hodnoty a určit jejich počet v obrázku. Mějme obrázek, který má po aplikaci prediktoru tento charakter a proveďme sběr statistik o tomto obrázku.

2	0	0	5
0	0	2	2
0	0	0	4
1	2	3	0

$\Rightarrow$

Pixel	Výskyt
0	8x
2	4x
1	1x
3	1x
4	1x
5	1x
Celkem	16

Obrázek 5.2: Sběr statistik z obrázku

Statistika nám určuje, že se v obrázku 8× vyskytuje pixel o hodnotě 0, 4× s hodnotou 2 a poté čtyři další pixely s hodnotami 1, 3, 4, 5. Celkem jsme tedy na počátku měli 16 pixelů, které jsme zredukovali na 6 množin (dle hodnoty). Každá tato hodnota bude v binárním

stromu zastoupena samostatným listem. V našem případě dostaneme binární strom o šesti listech. Nyní využijeme výše zmíněné vstupní fronty. Do této fronty umístíme sestupně dle četnosti hodnoty pixelů. Horní hodnota znamená hodnotu pixelu, spodní hodnota četnost. V praktické realizaci Huffmanova kódování není dobré počítat přímo s pravděpodobnostmi, stačí nám pouze vybírat prvky s nejnižší četností, což je potažmo pravděpodobnost výskytu.

0 8	2 4	1 1	3 1	4 1	5 1

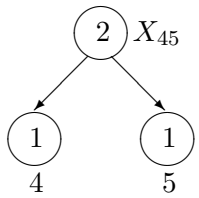
Obrázek 5.3: Vstupní fronta na začátku kódování

Do fronty jsme umístili hodnoty pixelů. Dále z této fronty budeme vybírat dva prvky s nejmenším počtem výskytů (spodní číslo) a z nich vytvoříme listy. Musíme si dát pozor, zda vybíráme hodnotu pixelu, která má být list, nebo zda vybíráme již vytvořený uzel. Po vybrání dvou prvků zde vrátíme jeden, jehož počet výskytů bude dán součtem dvou vytažených. Obrázek níže zobrazuje práci s frontou a postavení celého stromu. Na počátku uvažujme, že jsme již vybrali dva poslední prvky z fronty, tj. prvek s hodnotou 4 (1 výskyt) a prvek s hodnotou 5 (1 výskyt). Na jejich místo jsme vrátili prvek X_{45}.

1)

Obsah vstupní fronty:

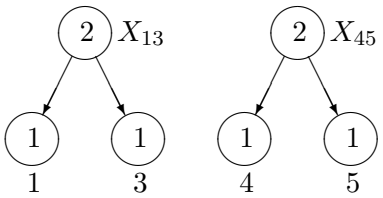
0 8	2 4	X_{45} 2	1 1	3 1



2)

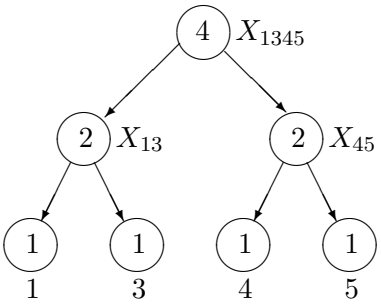
Obsah vstupní fronty:

0 8	2 4	X_{45} 2	X_{13} 2



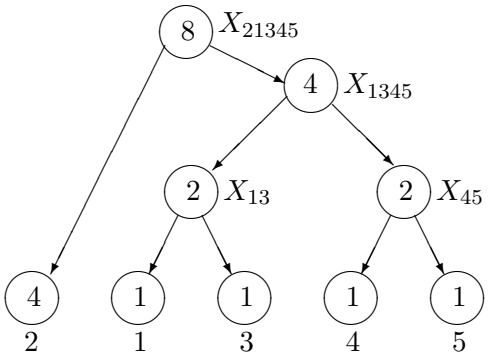
3) Obsah vstupní fronty:

0 8	2 4	X_{1345} 4

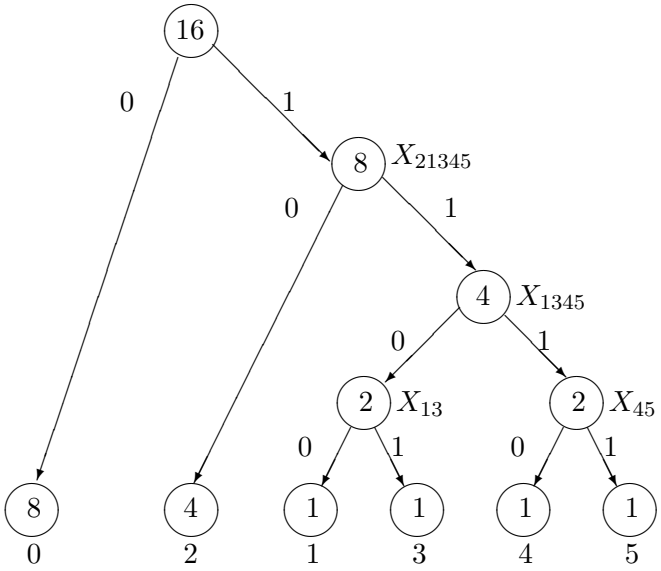


4) Obsah vstupní fronty:

0 8	X_{21345} 8



Vidíme, že nám ve vstupní frontě zbývají poslední dva prvky. První prvek je ještě nevytvořený uzel a další je podstrom celého Huffmanova stromu. Nyní už nezbývá, než vybrat tyto prvky z fronty a vytvořit tak finální verzi Huffmanova stromu.



Hodnota	Kód
0	0
2	10
1	1100
3	1101
4	1110
5	1111

Obrázek 5.4: Finální verze Huffmanova stromu

Celý proces tvorby Huffmanova stromu byl dokončen. V tabulce vidíme jednotlivé hodnoty pixelů a jejich kódy. Tyto kódy je reprezentují. To znamená, že nyní projdeme opět obrázek, ale ne pro sběr statistik, ale pro účely kódování. Porovnáme hodnotu v obrázku s hodnotou listu ve stromu a celý strom projdeme až ke kořeni.

Poté pomocí bitových maker zapíšeme výsledný kód do souboru. Celý algoritmus by se v pseudokódu dal zapsat následovně.

```
while (vstupni_fronta.velikost() != 1)
{
    vyber dva prvky s nejmensim ohodnocenim (prvni, druhy);
    vytvor treti (jako potomky ustanov dva vybrane);
    treti.ohodnoceni = prvni.ohodnoceni + prvni.ohodnoceni;
    vstupni_fronta.vloz(treti);
}
```

Při dekompresi je nutné opět sestavit binární strom. Teď už ovšem nemáme k dispozici vzorový obrázek, abychom provedli sběr statistiky a na jeho základě sestavili strom. Řešení je jednoduché. Uložit si statistiku (nebo celý strom) k zakódovaným datům obrázku. Když budeme provádět dekompresi, tak si načteme statistiku, sestavíme strom a poté načítáme prefixové bitové kódy a rekonstruujeme obrázek. Pokud jsme použili prediktor, musíme pamatovat na to, že dekompresí nedostáváme přímo hodnoty pixelů, ale pouze chybu predikce.

Huffmanovo kódování je hojně využíváno. Může být použito jako samostatné kódování, nebo součást. Výhodou tohoto kódování je jeho rychlost oproti adaptivní metodě (komprese i dekomprese) a nepříliš velká paměťová náročnost. Nevýhodou je, že optimální kód přiřazuje pouze hodnotám, jejichž pravděpodobnost je rovna záporným mocninám čísla 2.

5.2 Adaptivní Huffmanovo kódování

Variací statického Huffmanova kódování je adaptivní Huffmanovo kódování. Ve statické verzi procházíme obrázek dvakrát. Poprvé při sběru statistik, podruhé při samotném kódování pixelů. Adaptivní metoda tento problém řeší. K výstavbě odpovídajícího stromu potřebuje pouze jeden průchod obrázkem. Celý trik spočívá v přestavbě Huffmanova stromu v každém kroku. To znamená, že s každým načteným symbolem procházíme strom a sledujeme, jestli nebyla porušena *sourozenecká vlastnost*. Jednoduše řečeno, musíme zkontrolovat, zda neexistuje uzel blíže ke kořeni stromu se stejným počtem výskytů. Pokud tomu tak je, uzly vyměníme. Je zcela logické, že přestavování binárního stromu zabere více času. Adaptivní metoda dokáže generovat lepší kódy a tím zvyšuje kompresní poměr. Když si představíme správně sestavený Huffmanův strom, je nutné zajistit, aby listy, které mají vysokou četnost, byly nejblíže ke kořeni. Listy s nižším ohodnocením jsou pak dále od kořene. Proto pokud bychom chtěli celý algoritmus zapsat do pseudokódu, vypadal by následovně.

```
for(celý obrázek)
{
   načteme symbol;
   if (první čtení tohoto symbolu) {
      vytvoř_nový_uzel();
      nový_uzel->četnost++;
      aktualizuj_strom(nový_uzel);
   } else {
      aktualizuj_strom(uzel);
   }
}

void aktualiuj_strom(uzel)
{
   for(celý strom) {
      if(existuje uzel se stejnou četností a je blíže ke kořeni) {
         vyměň oba uzly;
         uzel->četnost++;
      }
      uzel->četnost++;
      uzel = předek_uzlu;
   }
}
```

Po sestavení stromu je situace stejná jako u statické varianty. Procházíme strom a pomocí bitových maker zapisujeme posloupnosti bitů do souboru. Při dekomprimaci postupujeme tak, že začínáme s prázdným stromem a při čtení tvoříme strom. Adaptivní metoda je také používaná díky své schopnosti generovat optimální kódy a nutnosti pouze jednoho průchodu. Její nevýhodou je časová náročnost a z implementačního hlediska je složitější než statická metoda. Zvláště co se týká aktualizace stromu. Autorem této metody není David A. Huffman. Algoritmus publikoval v roce 1973 Faller, v roce 1978 Gallager. Obměnu provedl v roce 1985 Knuth. Proto se můžeme setkat s názvem FGK.

5.3 Aritmetické kódování

Huffmanovo kódování přiřadí každému ze vstupních symbolů nějaký kód. Zato aritmetické kódování [7] pracuje naprosto odlišně. Proč přiřazovat symbolům nějaký kód? Proč nepřiřadit celému vstupnímu souboru nějaké jednoznačné číslo. Například číslo z intervalu $<0;1)$. Reálných čísel v tomto intervalu je nekonečně mnoho. S každým dalším symbolem se tento interval zmenšuje. Celá technika zmenšování intervalu musí brát v potaz četnosti jednotlivých symbolů, abychom mohli mluvit o entropickém kódování. Navíc tato technika odstraňuje problém Huffmanova kódování – přidělování celočíselných kódů. Princip spočívá v tom, že symboly s vyšší četností tento interval zmenšují méně než symboly s menší četností. V konečném důsledku symboly s vyšší četností přispívají do výstupního souboru menším počtem bitů. Huffmanovo a aritmetické kódování mají však jednu část společnou. Je to sběr statistiky. Dále pak dělíme interval na podintervaly. Kolik je množin hodnot symbolů, tolik máme podintervalů. Ukažme si tedy, jak celý proces kódování funguje.

5.3.1 Kódování

Mějme řetězec *aaab*. Vidíme, že četnost znaku a je 3 a četnost znaku b je 1. V tomto poměru rozdělíme interval $<0;1)$. Dostáváme, že znak a patří do intervalu $<0;0,75)$ a znak b do intervalu $<0,75;1)$. Tímto jsme provedli sběr statistik a základní rozdělení intervalu na podintervaly. Nyní můžeme přistoupit k samotnému kódování. Budeme číst jednotlivé vstupní znaky a vždy tento interval zmenšíme. Načítáme znak $a \Rightarrow$ vybíráme interval $<0;0,75)$, protože právě čtený znak do něj patří. Odteď budeme dále dělit interval $<0;0,75)$. Tento interval opět rozdělíme podle četností jednotlivých znaků. Dalším čteným znakem je opět a. Dostáváme interval $<0;0,5625)$. Opět čteme a. Interval před čtením posledního znaku je $<0;0,421875)$. Poté čteme znak b. Nyní dostaneme interval $<0,31640625;0,421875)$. Tím je kódování skončeno. Vstupní řetězec můžeme zakódovat některým z čísel v intervalu $<0,31640625;0,421875)$, logicky nejméně náročným na velikost při reprezentaci.

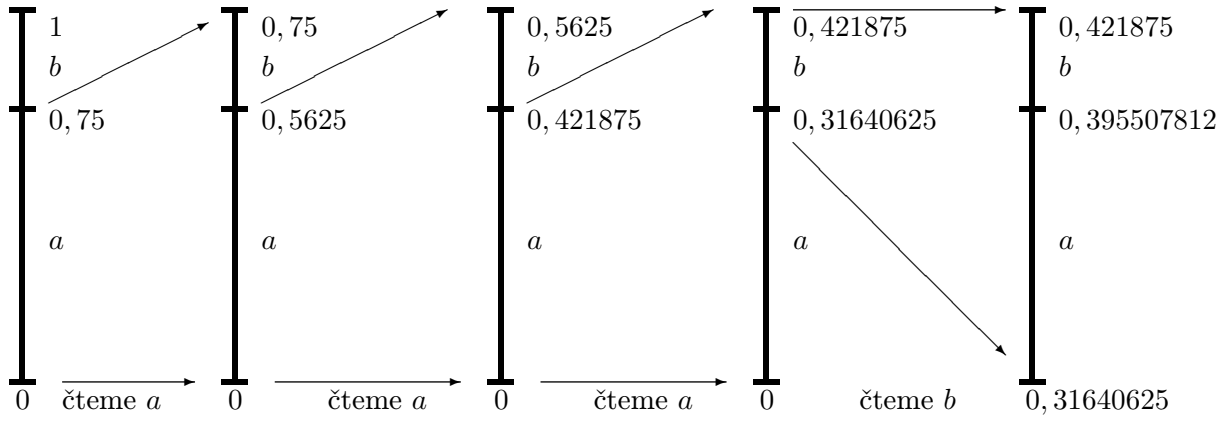
Znak	Interval
a	$<0;0,75)$
b	$<0,75;1)$

Aktuální znak	Interval
a	$<0;0,75)$
a	$<0;0,5625)$
a	$<0;0,421875)$
b	$<0,31640625;0,421875)$

Tabulka 5.1: Zmenšování intervalu při kódování

Pro lepší pochopení je na další stránce obrázek, který názorně ukazuje práci s intervalem. Nezáleží na pořadí symbolů po dokončení sběru statistik. Je však podmínkou, že kodér i dekóder je musí zpracovávat ve stejném pořadí. V tomto příkladě si z výsledného intervalu vybereme číslo 0,4.

Pokud bychom chtěli celý postup kódování zapsat pseudokódem, je vhodné si aktuální hranice intervalu označit jako $High, Low$. Nejprve provedeme inicializaci hranic intervalu a poté čteme všechny vstupní symboly. Při každém čtení se znovu přepočítávají krajní hodnoty intervalu a počítáme aktuální velikost tohoto intervalu, která je označena jako $Range$. K výpočtu ještě využijeme funkce $HighRange(s), LowRange(s)$, které vrací krajní hodnoty intervalu pro právě načtený znak. I při opakovaném čtení stejného symbolu se výsledný interval mění. Pseudokód kódování vypadá následovně:



Obrázek 5.5: Postup aritmetického kódování

```
High = 1;
Low  = 0;

for (všechny vstupní symboly)
{
   načti symbol;
   Range = High - Low;
   Low   = Low + Range * HighRange(s);
   High  = Low + Range * LowRange(s);
}

výsledek = <Low;High);
```

Vidíme, že celý proces není vůbec složitý. Zbývá však vyřešit jediný problém. Zvolit způsob, jakým dáme vědět, že jsou všechny symboly již načteny. Můžeme použít speciální symbol eof, nebo dopředu říci, kolik se bude číst znaků a ty potom načítat v počítaném cyklu for.

5.3.2 Dekódování

Dekomprese probíhá opačně. Dekodér si nejdříve zkonstruuje tabulku č. 5.1. Poté přečte číslo reprezentující vstup. V našem případě to bylo číslo 0,4. Dekodér podle rekonstruované tabulky ví, že toto číslo patří do intervalu $<0; 0,75)$. Dekódovaný znak je tudíž a. Poté eliminuje vliv znaku a na číslo 0,4 odečtením hodnoty Low znaku a, následně výsledek vydělí délkou intervalu znaku a. Dostáváme číslo $0,533333333$. Toto číslo opět patří do intervalu pro znak a. Znovu odečteme hodnotu Low a podělíme délkou intervalu. Dostáváme číslo 0,711111111. Opět dostáváme znak a. Naposledy provádíme tuto akci a získáváme číslo 0,948148148. Toto číslo patří do intervalu $<0,75; 1)$. Dekódovaným znakem je b. Celý řetězec jsme správně dekódovali jako $aaab$. V následující tabulce jsou ukázany jednotlivé výpočty.

Kód	Interval	Dekódovaný znak	Nový kód
0,4	$<0; 0,75)$	a	$(0{,}4 - 0)/0{,}75 = 0{,}533333333$
0,533333333	$<0; 0,75)$	a	$(0{,}533333333 - 0)/0{,}75 = 0{,}711111111$
0,711111111	$<0; 0,75)$	a	$(0{,}711111111 - 0)/0{,}75 = 0{,}948148148$
0,948148148	$<0,75; 1)$	b	–

Tabulka 5.2: Postup při dekompresi

Opět vidíme stejný problém jako při kódování. Určení délky dekódovaných dat. Řešením může být opět uložení počtu dekódovaných symbolů před samotný kód, nebo zařadit speciální znak eof. Tomuto znaku by se přiřadila malá pravděpodobnost (1 / počet všech symbolů). Do tabulky se statistikou by se musela přidat hodnota i pro tento znak. V pseudokódu jsou použity funkce $LowRange(s)$, která vrací spodní hranici daného znaku. Dále funkce $Range(s)$, která vrací délku intervalu daného znaku. Pseudokód dekódování vypadá takto:

```
načti a sestav tabulku se statistikou;
Code = číslo reprezentující vstupní data;
while(dekódovaný symbol není eof)
{
   najdi interval, do kterého patří Code;
   dle tohoto intervalu vypiš příslušný znak;
   Code = (Code - LowRange(s)) / Range(s);
}
```

Všechno vypadá jednoduše. V teoretické rovině je tento algoritmus naprosto bezchybný a plně vyhovující. Bohužel jen v teoretické rovině. Dopustili jsme se mnoha úvah, které při implementaci tohoto algoritmu na počítači nejsou proveditelné. Jdou však různými technikami obejít, aby bylo možné aritmetické kódování implementovat. Těmto technikám se bude věnovat následující kapitola.

5.3.3 Implementace aritmetického kódování

První chybou je přemýšlení v reálných číslech. V počítači můžeme ukládat čísla s omezenou délkou a tudíž i přesností. A protože jsou operace s čísly v plovoucí řádové čárce složité, je nutné implementaci provést za použití celých čísel. Po procesu kódování tedy nedostáváme jedno reálné číslo, ale posloupnost bitů. Další chybou, které se můžeme dopustit, je počítání pravděpodobností. Představme si pravděpodobnost symbolu eof, která je rovna 1/počet všech symbolů. Tento zlomek je tak malý, že se zcela jistě zaokrouhlí na hodnotu 0. Řešením je použití tzv. komutativních četností symbolů. Interval tedy nebude určen pravděpodobností jako takovou, ale pomocí četnosti symbolů. Označme komutativní četnosti symbolů jako $lowCount, highCount$. Hodnota $lowCount$ je součet všech četností symbolů v tabulce pod tímto symbolem. Hodnota $highCount$ je pak $lowCount$ navýšena o četnost symbolu. Poté můžeme tedy jednotlivé hranice intervalů $(lowRange(s), highRange(s))$ naprosto bez problému vyjádřit takto:

$$lowCount(s) \quad = \quad \sum_{i=0}^{n} count(i)$$

$$
\begin{aligned}
highCount(s) &= \sum_{i=0}^{n} lowCount(n) + count(n) \\
lowRange(s) &= \frac{lowCount(s)}{pocet\,vsech\,symbolu} \\
highRange(s) &= \frac{highCount(s)}{pocet\,vsech\,symbolu}
\end{aligned}
$$

5.3.4 E1 a E2 škálování

Při implementaci narazíme na problém přibližování obou mezí (Low a $High$). Tento problém nastává již po pár operacích dělení intervalu. Vše končí situací, kdy se obě meze rovnají. To je nežádoucí stav. Při hlubším zkoumání tohoto problému zjistíme, že pokud se rovnají nejlevější bity obou mezí, nastává tento problém. Další úvahou také zjistíme, že pokud se příslušené bity rovnají, tak se dále nemění (ani v případě úspěšného kódování). Proč tedy udržovat v proměnných Low a $High$ informaci, která se nemění? Příslušný počet bitů, které se rovnají rovnou zapíšeme na výstup. A na jejich místo zprava doplníme další bity (pro mez $High$ s hodnotou 1, pro Low s hodnotou 0). Co jsem tím udělali? Zvýšili přesnost a možnost uložit do těchto proměnných větší číslo. Při praktické implementaci tuto shodu mezí ohalíme testem, zda obě proměnné leží ve stejné polovině intervalu.

5.3.5 E3 škálování

Samotné techniky škálování E1 a E2 nepostačují k plné činnosti kodéru. Můžeme se dostat (a zcela jistě se dostaneme) do situace, kdy obě meze konvergují v blízkosti středu intervalu. Zde by nám test shodnosti dvou nejvýznamnějších bitů selhal a kódování by se neprovádělo správně. Vyjmutím druhého nejvýznamějšího bitu (z obou mezí) provedeme škálování. Nesmíme ještě zapomenout na vložení bitu zprava do obou proměnných $High, Low$. Tato technika se provádí, dokud si nejsou rovny nejvýznamnější bity $High, Low$. Musíme si také pamatovat, kolik bitů bylo takto vyjmuto. Dalším krokem je zapsaní nejvýznamějšího bitu na výstup a poté vyjmutých bitů. Celý algoritmus kódování je nastíněn zde:

```
for(cely obrazek) {
  přepočítej meze;
  while ((high < half) || (low >= half)) { // E1, E2 škálování
    if (high < half)
        proveď škálování;
    else if (low >= half)
        proveď škálování;
  }
  // E3 škálování
  while (( FirstQuarter <= low ) && ( high < ThirdQuarter )) {
       proveď škálování;
  }
}
```

Kapitola 6

Implementace knihovny

Cílem této bakalářské práce je implementovat multiplatformní knihovnu pro bezeztrátovou kompresi obrazu. Tato knihovna implementuje dvě metody komprese. Statické Huffmanovo kódování (viz kapitola č. 5.2) a aritmetické kódování (kapitola č. 5.4). Uživatel si může zvolit, kterou kompresní metodu použije. Pro samotnou práci s obrázky je použita grafická knihovna OpenCV. Tato knihovna implementuje funkce pro práci s jednotlivými pixely, funkce pro načítání obrázků, ukládání obrázku a získání informací o obrázcích (výška, šířka, počet barevných kanálů). Pojďme se nyní podívat, jak je knihovna implementována a jakých výsledků lze dosáhnout. Knihovna se skládá z modulu `coder.cpp` a jeho rozhraní `coder.h`. Ovládání programu je implementováno v modulu `main.cpp`.

6.1 Třída static_huffman

Prvním úkonem je načtení obrázku. Vstupem může být obrázek v libovolném grafickém formátu. Ovšem smysluplným vstupem je pouze vstup nekomprimovaný. Zde proto doporučuji obrázky ve formátu `.bmp,.ppm`. Po úspěšném načtení obrázku si objekt této třídy uloží do svých proměnných informace o obrázku. Ukládá se výška obrázku, šířka a počet barevných kanálů. Pokud si uživatel zvolil jiný barevný model než RGB (viz kapitola č. 3), převedeme obrázek do zvoleného barevného modelu. Když je převod dokončen, následuje predikce hodnot. Obrázek se prochází od levého dolního rohu po řádcích. Uživatel si prediktor nemůže zvolit. Přestože si uživatel nemůže prediktor v nastavení změnit, tato třída má devět metod implementující funkci ruzných prediktorů (viz tabulka č. 4.2). Prediktor lze tedy změnit, ale pouze zásahem do zdrojového kódu. Teď už nezbývá, než vypracovat statistiku a chybu predikce zakódovat. Obě tyto činnosti se provádí v metodě `coding()`. Je vypracován záznam o počtu skupin chyb predikce (dle velikosti chyby). Tato statistika je uložena do položek struktury `BinaryTree`. Následuje vytvoření binárního stromu. Algoritmus je popsán v kapitole č. 5.1. Abychom poznali, zda se jedná o list, je ve struktuře `BinaryTree` také dvouhodnotová proměnná určující tuto vlastnost. Strom je tedy sestaven odspodu nahoru. Čeká nás opětovný průchod obrázkem, kdy přečteme chybu predikce a k ní si vyhledáme patřičný list binárního stromu (k tomu slouží kontejner `leafs`). Uložíme si pro tuto chybu predikce prefixový kód a pokračujeme dalším bodem v obrázku. V momentě, kdy máme přidělené kódy všem chybám predikce, musíme do souboru (před samotný kód) uložit informace o obrázku a vypracovanou statistiku. Toto je nesmírně důležité, protože dekodér nemá prakticky žádné informace o vstupním obrázku. Také musí mít k dispozici statistiku, aby mohl zpětně sestavit binární strom a dekódovat kód uložený v souboru.

Z těchto důvodů je struktura výsledného souboru členěna takto: `šířka, výška, počet barevných kanálů, krok, počet položek ve statistice, statistika, kód`. Při dekódování si dekodér přečte tyto hodnoty, tím si vytvoří představu o tom, kolik hodnot bude muset rozkódovat, aby sestavil celý obrázek. Načte si statistiku, sestaví binární strom. Binární stromy kodéru a dekodéru musí být naprosto stejné, jinak nedosáhneme odpovídajícího výsledku. V poslední fázi činnosti dekodéru prochází dekodér uložený kód v souboru a z něj určuje chybu predikce. Zde si musíme uvědomit, že jsme kódovali chybu predikce, nikoliv skutečnou hodnotu pixelu. Takže opět přichází na řadu prediktor, který musí plnit stejnou funkci jako prediktor kodéru. Jinak bychom opět nedosáhli požadovaného výsledku. Skutečná hodnota pixelu je tedy dána součtem predikce a chyby predikce. Pokud si uživatel zvolil model RGB, činnost dekodéru končí uložením dekódovaného obrázku například do formátu PNG. Je-li však použitý model jiný, je nezbytné převést vybraný model do modelu RGB (toto řeší samotná knihovna). Po tomto převodu je poslední krok stejný.

6.2 Třída arithmetic_coder

Co se týká metod pro práci prediktorů nebo převodů na jiné barevné modely, jsou stejné jako u Huffmanova kódování. Jsou zde navíc metody potřebné pro řízení kódování/dekódování (`coding()/decoding()`). Dále metody na práci s bity (`SetBitFlush(), GetBit()`). Celý proces opět začíná načtením obrázku, případným převodem do jiného barevného modelu a použitím prediktoru. Vypracujeme statistiku z obrazových dat (položka statistiky má typ `TSymbol`). Tuto statistiku ještě doplníme o hodnoty $low_count, high_count$ jednotlivých symbolů. Do výstupního souboru uložíme informace o obrázku v tomto pořadí `šířka, výška, počet barevných kanálů, krok, počet položek v statistice`. Dále ukládáme statistiku. Poté můžeme přikročit ke kódování samotných dat. Po úspěšném kódování uzavíráme soubor a končíme s kompresí. Při dekompresi si nejprve načteme informace o obrázku, statistiku a poté můžeme dekódovat. Výsledný dekódovaný obrázek si můžeme dle libosti uložit například do formátu PNG. Jak bylo výše uvedeno, metody pro práci prediktorů a pro převod do různých barevných modelů jsou u obou kodérů stejné. A to z důvodu vzájemné nezávislosti. Pokud chceme v dalším projektu použít statické Huffmanovo kódování, můžeme použít třídu `static_huffman` a její metody. To samé platí pro aritmetické kódování. Pokud bych třídu `arithmetic_coder` implementoval pomocí dědičnosti z třídy `static_huffman`, musel bych upravovat třídu aritmetického kódování a doplňovat metody z Huffmanova kódování. Tato implementace zajišťuje naprostou nezávislost.

6.3 Spuštění programu

V této podkapitole si povíme, jak spouštět program. Jedná se o konzolovou aplikaci, takže se všechny parametry ovlivňující výsledek komprese zadávají přes terminál pomocí tzv. přepínačů. Existuje celkem pět přepínačů, které mají přesně určené pořadí a hodnoty, kterých mohou nabývat. Jedná se o přepínače s názvem `-method, -act, -model, -input, -output`. V následující tabulce vidíme konkrétní význam jednotlivých přepínačů a množinu přípustných hodnot. K jednomu přepínači můžeme vybrat v rámci jedné komprese pouze jednu hodnotu z množiny přípustných hodnot.

Pro názornost uvádím příklad zakódování obrázku s názvem `lena.bmp`.

```
./main -method huffman -act code -model rgb -input lena.bmp -output result.bin
```

Přepínač	Význam	Množnina přípustných hodnot
–method	Volba entropického kodéru	huffman, arithm
–act	Volba kódování / dekódování	code, decode
–model	Volba barevného modelu	rgb, yuv, rct, shr, rdct, ycc, yff, loco, nfs
–input	Volba vstupního souboru	picture.bmp
–output	Volba výstupního souboru	result.bin

Tabulka 6.1: Význam přepínačů

A jeho dekódování a uložení do formátu PNG s názvem `lena.png`.

```
./main -method huffman -act decode -model rgb -input result.bin -output lena.png
```

Pokud kódujeme obrázek s výstupním barevným modelem RGB (přepínač `-model`), musíme jej i tak rozkódovat. Nelze zakódovat obrázek s výsledným modelem RGB a poté jej dekódovat a uvést, že byl zakódován do modelu LYUV.

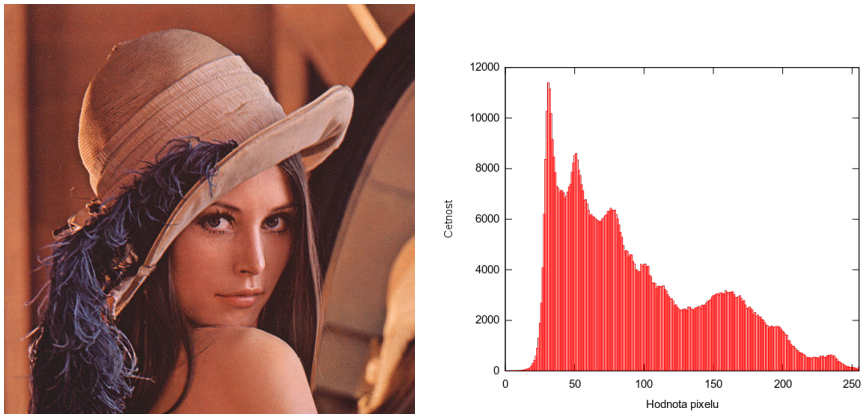
Kapitola 7

Dosažené výsledky

Protože jsme v této práci probrali mnoho metod predikce, barevných modelů a způsobů entropického kódování, je čas na porovnání jejich účinnosti. Zajímá nás, který prediktor je lepší. Nebo který barevný model. Jak již bylo zmíněno, způsob entropického kódování není stěžejní problém bezeztrátové komprese obrazu. Budeme se snažit na referenčním obrázku vydedukovat, který prediktor je pro většinu případů nejvhodnější. Jako prostředek pro porovnání účinností prediktorů a barevných modelů použijeme histogram. Jedná se o graf, který má na ose x hodnoty pixelů a na ose y jejich četnosti. Důvod použití je jeho názornost. Histogram totiž nepřímo zobrazuje pravděpodobnosti výskytu hodnot na ose x. Naší snahou je, aby hodnoty na ose x nabývaly co nejrozdílnějších pravděpodobností. Právě tuto vlastnost zobrazuje histogram. Pokud je histogram špičatý, hodnoty mají rozdílné pravděpodobnosti. Pokud je však hladký, znamená to, že pravděpodobnosti hodnot jsou vyrovnané a při entropickém kódování jim budou přiřazovány delší kódy.

7.1 Porovnání prediktorů

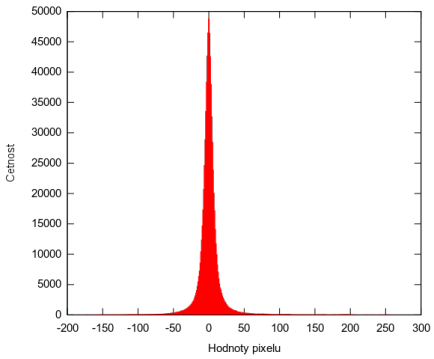
Na referenčním obrázku `lena.bmp` si ukážeme vlastnosti jednotlivých prediktorů. Obrázek má rozlišení 512×512 bodů v barevném modelu RGB. Podívejme se na jeho histogram.



Obrázek 7.1: Referenční obrázek a jeho histogram

Jak bylo zmíněno v kapitole č. 2, hodnoty u modelu RGB se pohybují od 0–255, což vidíme na ose x. Na ose y vidíme jejich četnosti. Nejvíce je tedy pixelů s hodnotou asi 30.

Co lze z obrázku jistě říct je, že je tento graf hladký. Pravděpodobnosti jsou vyrovnané. Žádná hodnota svou pravděpodobností výrazně nepřekračuje ostatní. Proto by nebylo rozumné kódovat takový obrázek. My však použijeme prediktor prvního řádu (viz kapitola č. 4.2). Podívejme se, jak se histogram změní. Očekáváme, že nám tento prediktor silně zredukuje počet množin pixelů. Zde již neuvádím snímek obsahující chyby predikce, protože jsou některé záporné a to v modelu RGB nelze zobrazit.



Obrázek 7.2: Vliv prediktoru na histogram

Výsledek splňuje naše očekávání. Drtivá většina pixelů se pohybuje v okolí bodu 0. Což je nulová chyba predikce. Na tomto obrázku je nutné si uvědomit, že již nezobrazujeme hodnoty pixelů, ale chybu predikce. V ideálním případě by byly všechny hodnoty v bodě 0. Těchto hodnot je šířka · výška · počet barevných kanálů. V současném stavu můžeme s velkou výhodou oproti předchozímu stavu obrázek zakódovat. V čem je nyní rozdíl? Máme několik hodnot s několikanásobně vyšší četností než ostatní. Pokud budeme kódovat pomocí Huffmanova kódování, symboly s vysokou četností dostanou krátké kódy a s nízkou delší. Kdežto v předchozím případě bylo těchto hodnot s vyšší četností podstatně více, čímž se kódy prodlužovaly a v konečném výsledku nelze dosáhnout solidního kompresního poměru. Z praktických důvodů zde neukazuji histogramy pro ostatní prediktory, protože vypadají velice podobně.

V dalším příkladu budeme opět předpokládat referenční obrázek `lena.bmp` a kompresi v barevném modelu RGB. Obrázek má velikost 768,1 kB. Uvažujme prediktory z tabulky č. 4.2, ze které vychází i označení prediktorů. Jako entropický kodér nám poslouží Huffmanovo kódování. V tabulce lze vidět účinnost jednotlivých prediktorů, která se projeví ve velikosti komprimovaného souboru. Pro úplnost uvádím i počet množin hodnot pixelů.

	Velikost souboru (kB)	Počet množin hodnot pixelů	Kompresní poměr
Prediktor č.0	722	255	1,064
Prediktor č.1	523	389	1,469
Prediktor č.2	491	366	1,564
Prediktor č.3	506	341	1,518
Prediktor č.4	564	480	1,362
Prediktor č.5	533	398	1,441
Prediktor č.6	522	382	1,471
Prediktor č.7	498	369	1,542
Prediktor č.8	484	253	1,587
Prediktor MED	465	240	1,652

Tabulka 7.1: Velikost komprimovaných dat dle prediktoru

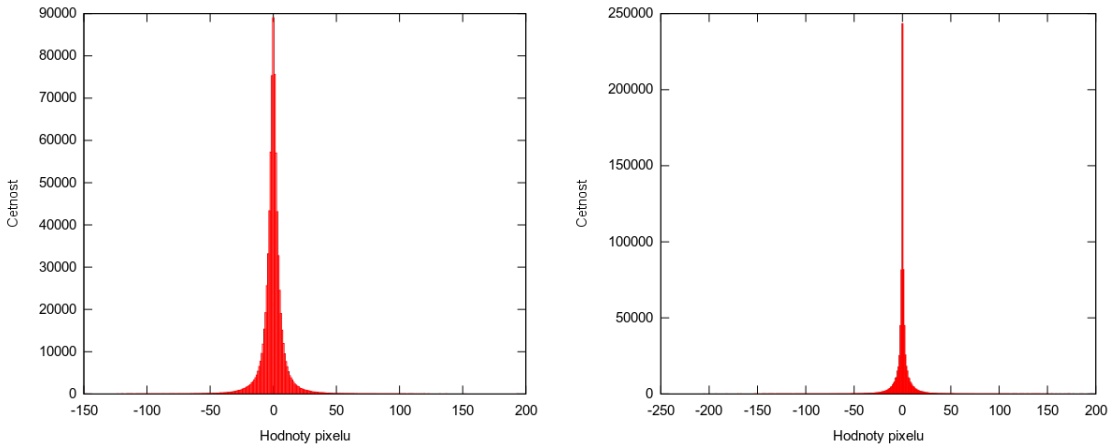
Na předposledním řádku tabulky je prediktor, který není označen v tabulce č. 4.2. Jedná se o prediktor, který vychází z prediktoru č. 6, ale pokud může přečíst všechny tři hodnoty (dle obrázku č. 4.6), tak z nich vypočítá průměr a ten prohlašuje za predikovanou hodnotu. Jako jednoznačně nejlepší prediktor (nejen z této tabulky) označuji prediktor MED, který jsem ve své implementaci použil jako implicitní. Pro srovnání uvádím, že komprimovaný obrázek `lena.bmp` ve formátu PNG měl 443,8 kB.

7.2 Porovnání barevných modelů

Myšlenka použitá u vyhodnocení vhodnosti prediktorů se prolíná i do barevných modelů. Opět chceme vědět, který je nejvhodnější. Jenže to nelze se stoprocentní jistotou určit. Obecně je však dokázáno, že model RGB není vhodný pro kompresi obrazu kvůli silné korelaci mezi sousedními pixely. Existují však výjimky, u kterých platí, že například model RGB je pro kompresi vhodnější než model LYUV. Porovnání bude provedeno tak, že vždy porovnáme model RGB s nějakým jiným. K porovnání opět použijeme histogramy. V každé podkapitole budeme uvažovat jiný referenční obrázek.

7.2.1 RGB vs. LYUV

Znovu si připomeňme, co potřebujeme k dosažení co nejlepšího kompresního poměru. Potřebujeme skupiny pixelů se stejnou hodnotou a s maximální četností. Důvodem pro převod do jiného barevného modelu je snížení změn hodnot pixelů na chromatických složkách. Pokud toho docílíme, tak kódujeme méně symbolů a s kratším kódem (u Huffmanova kódování). Pokud bychom toto chtěli převést na zobrazení v histogramu, chceme histogram špičatý, nikoliv hladký. Dalším požadavkem je umožnění co nejpřesnější predikce, která následuje po převodu barevného modelu. V histogramu jsou zaznamenány hodnoty chyby predikce. Jako referenční prediktor budeme v následujích podkapitolách uvažovat prediktor MED a jako referenční obrázek použijeme `lena.bmp`. Provedli jsme převod do modelu LYUV a poté aplikovali prediktor MED.



Obrázek 7.3: RGB/LYUV

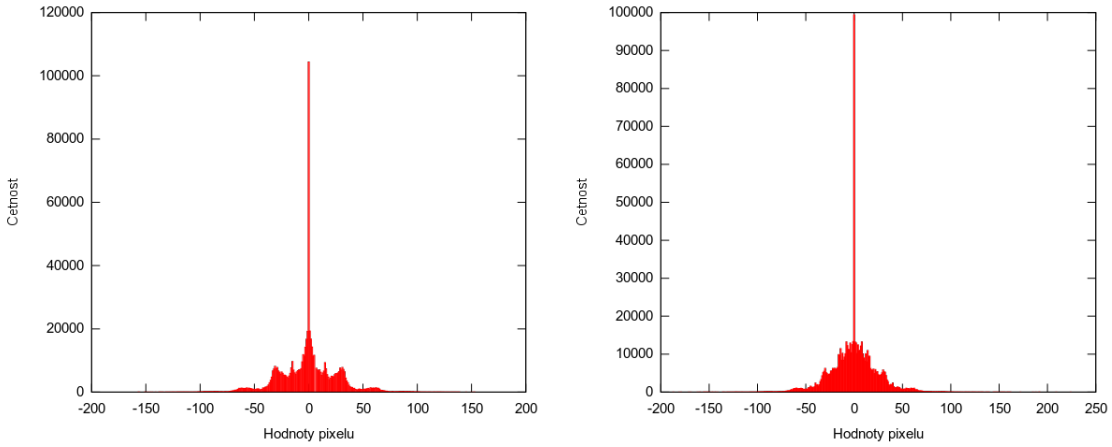
Vidíme, že v obou případech prediktor splnil svou práci. Oba histogramy mají stejný tvar a požadovaný rozsah (většina hodnot je v okolí 0). Rozdíl je však v četnosti. V levém histogramu je zobrazen model RGB. Četnost predikce o hodnotě 0 je 90 000. Na pravém histogramu (model LYUV) je četnost 250 000. Je tedy vidět, jak nám změna barevného modelu vhodněji předpřipravila data pro kodér.

Pro ilustraci uvádím tabulku reflekující schopnost zvýšení kompresního poměru pouze výměnou modelu LYUV za model RGB. Bude zde uvedeno šest obrázků reprezentujích tuto schopnost. Jako poslední bude uveden obrázek, v jehož případě barevný model LYUV snížil kompresní poměr. Výběr obrázků probíhal z několika databází korpusů, což jsou obrázky, na kterých se zkouší účinnost komprese. Proto zde uvádím pouze reprezentativní vzorek, ale komprese byla testována řádově na stovkách obrázků.

Obrázek	Originální velikost	Velikost v RGB	Velikost v LYUV	Kompresní poměry
test1.bmp	768,1 kB	465,2 kB	405, 7 kB	1,651 / 1,893
test2.bmp	1224,8 kB	767,2 kB	637,2 kB	1,596 / 1,922
test3.bmp	1396,8 kB	637,6 kB	639, 6 kB	2,191 / 2,184
test4.bmp	263,7 kB	73,9 kB	70,9 kB	3,568 / 3,719
test5.bmp	1329, 8 kB	678,6 kB	625,7 kB	1,960 / 2,125
test6.bmp	10276 kB	4071,8 kB	4834,6 kB	2,524 / 2,125

7.2.2 RGB vs. YC_oC_g

V této podkapitole budeme uvažovat jako referenční obrázek `baboon.bmp`. Tento obrázek je velmi obtížný ke kompresi a z histogramů poznáme, že ani změna barevného modelu mnohdy nemusí nutně znamenat významné zvýšení kompresního poměru. Výsledné histogramy tedy vypadají takto:



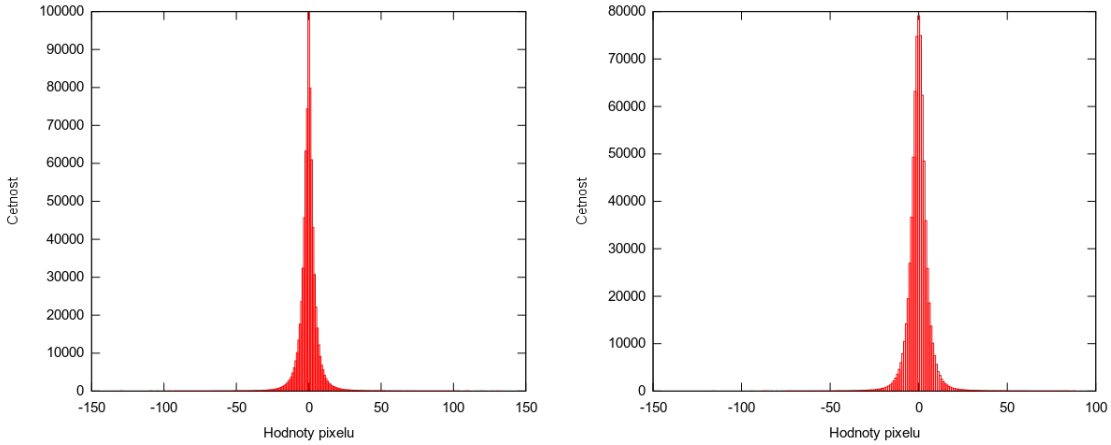
Obrázek 7.4: RGB/YC_oC_g

Na tomto těžkém obrázku ukázal prediktor, že taktéž nemusí vždy vykazovat chybu v blízkosti hodnoty 0. Tvar histogramů je přibližně stejný, avšak četnost chyby predikce o velikosti 0 je u modelu RGB přes 100 000 a u modelu YC_oC_g přibližně 100 000. Už jen z tak letmého pohledu na histogram můžeme celkem přesně usoudit, že velikosti obou komprimovaných souborů se výrazně neliší. Je tomu skutečně tak. Velikost komprimovaného souboru u modelu RGB je 560,1 kB a u modelu YC_oC_g 556,6 kB. Rozdíl je minimální. Opět uvádím tabulku s přehledem obrázků a účinností komprese za použití jiného barevného modelu.

Obrázek	Originální velikost	Velikost v RGB	Velikost v YC_oC_g	Kompresní poměry
test1.bmp	768,1 kB	465,2 kB	382, 9 kB	1,651 / 2,006
test2.bmp	1224,8 kB	767,2 kB	606,4 kB	1,596 / 2,020
test3.bmp	1396,8 kB	637,6 kB	691, 1 kB	2,191 / 2,021
test4.bmp	263,7 kB	73,9 kB	69, 7 kB	3,568 / 3,783
test5.bmp	1329, 8 kB	678,6 kB	592,9 kB	1,960 / 2,243
test6.bmp	10276 kB	4071,8 kB	4924,3 kB	2,524 / 2,087

7.2.3 RGB vs. YF_rF_b

Uvažujme opět prediktor MED a referenční obrázek `fruits.bmp`. Výsledek bude podobný jako v předchozí podkapitole. V tomto případě uvidíme vysoký pokles četnosti.



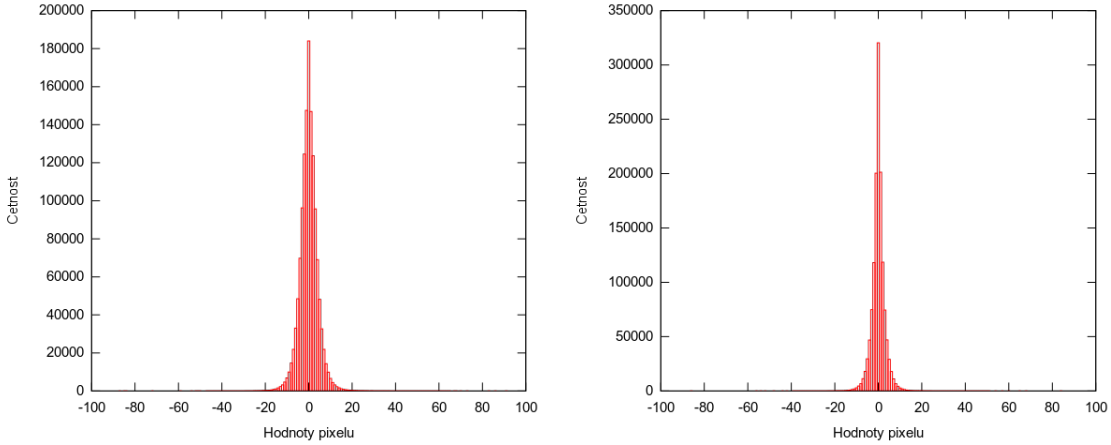
Obrázek 7.5: RGB/YF_rF_b

Na rozdíl od histogramů z předchozí podkapitoly, zde prediktor splnil svůj účel. Zmiňovaný počet četnosti vidíme na pravém histogramu. Je to pokles přibližně o 20 000 oproti modelu RGB. Rozdíl velikosti souborů bude minimální. Velikost komprimovaného souboru v RGB je 401,5 kB a v YF_rF_b 399,6 kB. Pokud bychom se měli tedy rozhodnout, jaký zvolit barevný model (z těchto dvou), bylo by to v tomto případě bezpředmětné.

Obrázek	Originální velikost	Velikost v RGB	Velikost v YF_rF_b	Kompresní poměry
test1.bmp	768,1 kB	465,2 kB	384, 2 kB	1,651 / 1,999
test2.bmp	1224,8 kB	767,2 kB	610,4 kB	1,596 / 2,007
test3.bmp	1396,8 kB	637,6 kB	623, 0 kB	2,191 / 2,242
test4.bmp	263,7 kB	73,9 kB	69, 5 kB	3,568 / 3,794
test5.bmp	1329, 8 kB	678,6 kB	598,6 kB	1,960 / 2,222
test6.bmp	10276 kB	4071,8 kB	5014,3 kB	2,524 / 2,049

7.2.4 RGB vs. RCT

V tomto okamžiku by se mohlo zdát, že je volba správného barevného modelu velice vrtkavá záležitost. U některých obrázků pomůže, u některých ne. Obrázků, u kterých kompresní poměr výrazně nezvýší, je minimum. Pravou sílu volby vhodného barevného modelu poznáme až v následujích třech podkapitolách. Ukažme si proto příklady, kdy tato volba velice silně ovlivňuje kompresní poměr. Budeme opět předpokládat prediktor MED a jako referenční obrázek `ZeldaColor.bmp`.



Obrázek 7.6: RGB/RCT

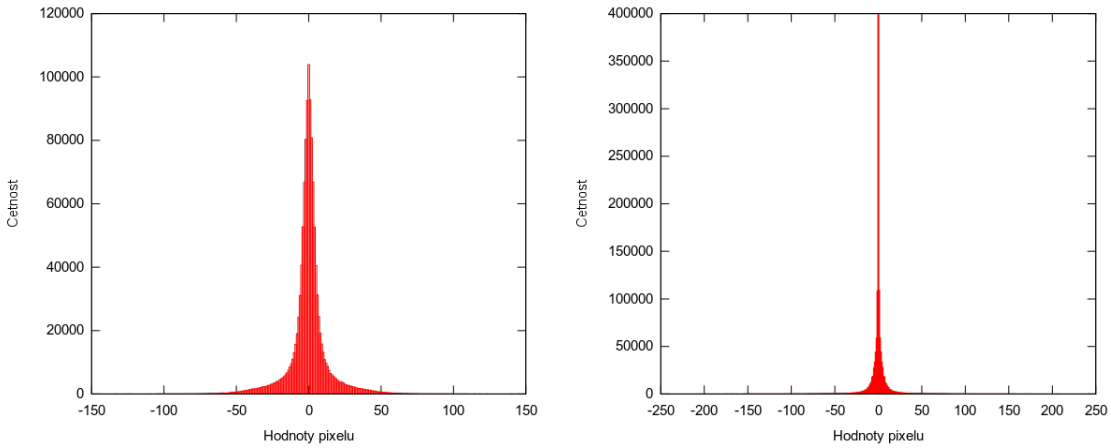
Co se týká činnost prediktoru, je vše v pořádku. I tvar histogramů je podobný. Opět se liší v četnostech. U modelu RGB je nejvyšší četnost přes 180 000. U modelu RCT je to 320 000. Při tomto pohledu na histogram můžeme říct, že velikost komprimovaného souboru v modelu RCT bude značně menší než v modelu RGB. Konkrétně je to 678,6 kB v RGB a 587,5 kB v RCT. Téměř 100 kB rozdíl pouze změnou barevného modelu.

Obrázek	Originální velikost	Velikost v RGB	Velikost v RCT	Kompresní poměry
test1.bmp	768,1 kB	465,2 kB	378, 6 kB	1,651 / 2,029
test2.bmp	1224,8 kB	767,2 kB	597,9 kB	1,596 / 2,049
test3.bmp	1396,8 kB	637,6 kB	634, 4 kB	2,191 / 2,202
test4.bmp	263,7 kB	73,9 kB	62, 6 kB	3,568 / 4,212
test5.bmp	1329, 8 kB	678,6 kB	587,5 kB	1,960 / 2,262
test6.bmp	10276 kB	4071,8 kB	4623,2 kB	2,524 / 2,222

7.2.5 RGB vs. RDCT

Opět stejný prediktor a referenční obrázek `barbara.bmp`.

Obrázek	Originální velikost	Velikost v RGB	Velikost v RDCT	Kompresní poměry
test1.bmp	768,1 kB	465,2 kB	412 kB	1,651 / 1,864
test2.bmp	1224,8 kB	767,2 kB	647,1 kB	1,596 / 1,893
test3.bmp	1396,8 kB	637,6 kB	635, 2 kB	2,191 / 2,200
test4.bmp	263,7 kB	73,9 kB	76, 1 kB	3,568 / 3,465
test5.bmp	1329, 8 kB	678,6 kB	642,5 kB	1,960 / 2,070
test6.bmp	10276 kB	4071,8 kB	5130,9 kB	2,524 / 2,003

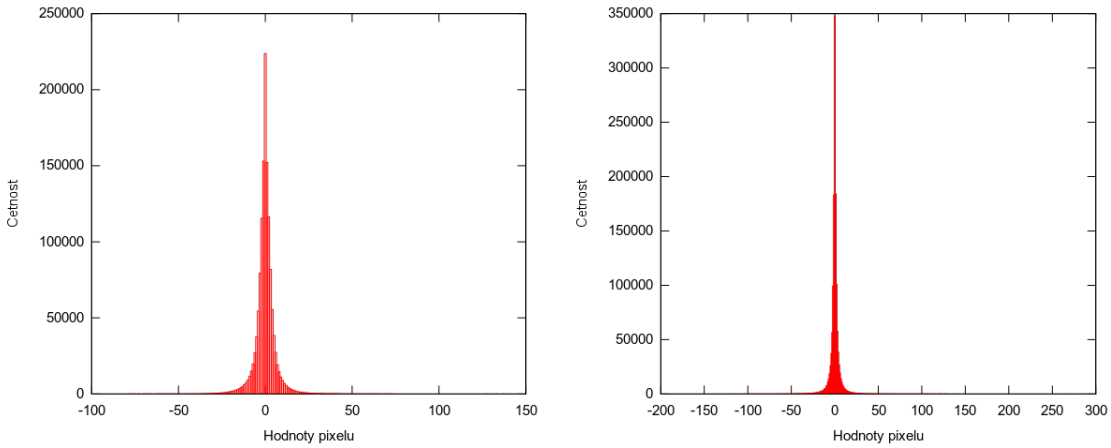


Obrázek 7.7: RGB/RDCT

Čtyřnásobně vyšší četnost u modelu RDCT. Převedeno na velikost komprimovaného souboru - u modelu RGB 811,9 kB a RDCT 674,2 kB. Více jak 100 kB rozdíl.

7.2.6 RGB vs. SHIRCT

Jako prediktor použijeme i teď prediktor MED. Pro demonstraci nám poslouží obrázek `BoatsColor.bmp`.



Obrázek 7.8: RGB/SHIRCT

Model SHIRCT má nejvyšší četnost 350 000. Model RGB pouhých 220 000. I zde dosáhneme vyššího kompresního poměru změnou barevného modelu. Pokud bychom měli porovnat činnost prediktoru, je v pořádku. Velikost komprimovaného souboru v RGB je 709,7 kB a 655,9 kB v SHIRCT.

Obrázek	Originální velikost	Velikost v RGB	Velikost v SHIRCT	Kompresní poměry
test1.bmp	768,1 kB	465,2 kB	406, 1 kB	1,651 / 1,891
test2.bmp	1224,8 kB	767,2 kB	646,9 kB	1,596 / 1,893
test3.bmp	1396,8 kB	637,6 kB	691, 2 kB	2,191 / 2,021
test4.bmp	263,7 kB	73,9 kB	68, 0 kB	3,568 / 3,878
test5.bmp	1329, 8 kB	678,6 kB	648,5 kB	1,960 / 2,051
test6.bmp	10276 kB	4071,8 kB	5208,3 kB	2,524 / 1,973

7.2.7 Porovnání účinnosti komprese dle barevného modelu

Předchozí podkapitoly ukázaly, jak lze z histogramu vyčíst účinnost komprese v závislosti na barevném modelu. Rozebírat každý barevný model by jistě nemělo smysl. Mnohem lepší bude porovnání na určitých obrázcích. Uvedeme si zde 10 základních obrázků, na kterých bude provedena komprese pomocí Huffmanova entropického kodéru. Všechny obrázky jsou k dohledání na internetu. Nazývají se :

- `baboon.bmp`
- `barbara.bmp`
- `ZeldaColor.bmp`
- `lena.bmp`
- `yacht.bmp`
- `monarch.bmp`
- `flowers.bmp`
- `BoatsColor.bmp`
- `fruits.bmp`
- `goldhill.bmp`

Do tabulky uvádím i velikost ve formátu PNG pro představu účinnosti komprese. Referenční prediktor je prediktor MED.

Obrázek	BMP	RGB	LYUV	YC_oC_g	YF_rF_b	PNG
baboon.bmp	703,2 kB	560,1 kB	569 kB	557,2 kB	556,6 kB	520 kB
barbara.bmp	1215 kB	811,9 kB	760 kB	608 kB	610,9 kB	801,6 kB
ZeldaColor.bmp	1330 kB	678,6 kB	625,7 kB	592,9 kB	598,6 kB	690,2 kB
lena.bmp	768,1 kB	465,2 kB	405,7 kB	382,9 kB	384,2 kB	445,6 kB
yacht.bmp	720,1 kB	398,9 kB	382,2 kB	374,2 kB	376,8 kB	404 kB
monarch.bmp	1152 kB	604,5 kB	528,4 kB	493,5 kB	497,9 kB	604,2 kB
flowers.bmp	530,3 kB	341,1 kB	328,5 kB	322,6 kB	322,9 kB	340,4 kB
BoatsColor.bmp	1330 kB	709,7 kB	631 kB	599 kB	604 kB	713,8 kB
fruits.bmp	720,1 kB	401,5 kB	401,1 kB	395,9 kB	399,6 kB	410,9 kB
goldhill.bmp	1215 kB	746,8 kB	651,8 kB	595,3 kB	597,6 kB	740,3 kB

Obrázek	BMP	RCT	RDCT	SHIRCT	LOCO-I	NFS
baboon.bmp	703,2 kB	554,2 kB	578,4 kB	577,3 kB	560,2 kB	557,7 kB
barbara.bmp	1215 kB	609,1 kB	674,2 kB	661,6 kB	610,3 kB	610,5 kB
ZeldaColor.bmp	1330 kB	587,5 kB	642,5 kB	648,5 kB	588,8 kB	602,5 kB
lena.bmp	768,1 kB	378,6 kB	412 kB	406,1 kB	379,1 kB	383,7 kB
yacht.bmp	720,1 kB	371,8 kB	393,9 kB	401,4 kB	375,2 kB	380,6 kB
monarch.bmp	1152 kB	498,6 kB	541,3 kB	552,5 kB	500,6 kB	509,9 kB
flowers.bmp	530,3 kB	318,9 kB	333,6 kB	338,9 kB	332,6 kB	324 kB
BoatsColor.bmp	1330 kB	593 kB	641,7 kB	655,9 kB	594,1 kB	605,5 kB
fruits.bmp	720,1 kB	388,8 kB	421,1 kB	422,7 kB	387,9 kB	401,3 kB
goldhill.bmp	1215 kB	592,5 kB	664 kB	634,3 kB	593,3 kB	599,8 kB

7.3 Celkové posouzení komprese

Cílem předchozích kapitol bylo ukázat vliv prediktoru a barevného modelu na účinnost komprese. Uvažovali jsme také jeden entropický kodér (Huffmanovo statické kódování). Tato kapitola právě bude uvažovat dva implementované entropické kodéry. Porovnáme tedy Huffmanovo statické kódování a aritmetické kódování. Nemělo by totiž smysl ukazovat histogramy pro aritmetické kódování, protože ty jsou stejné jako u Huffmanova kódování. Co činí aritmetické kódování lepší, je přidělování optimálního kódu. Pojďme se podívat, jak to vypadá. Mějme opět sadu testovacích obrázků a uvažujme referenční prediktor MED. Očekáváme lepší výsledky u aritmetického kódování. Budeme začínat s modelem RGB. Uživatel si může sám porovnat účinnost obou kompresních metod u všech nabízených barevných modelů. Účelem této kapitoly není ukazovat kompresní poměry u všech barevných modelů, ale naznačit a dokázat, že aritmetické kódování je obecně lepší jako Huffmanovo kódování.

Obrázek	Huffmanovo kódování	Aritmetické kódování	Velikost v PNG
baboon.bmp	560,1 kB	557,8 kB	520 kB
barbara.bmp	811,9 kB	807,5 kB	801,6 kB
ZeldaColor.bmp	678,6 kB	673,2 kB	690 kB
lena.bmp	465,2 kB	461,9 kB	445,6 kB
yacht.bmp	398,9 kB	396 kB	404 kB

Všechny soubory zakódované pomocí aritmetického kódování mají menší velikost než pomocí Huffmanova kódování. Pouze ve dvou případech jsme dosáhli lepšího kompresního poměru než formát PNG. To je ovšem pochopitelné, protože používáme model RGB. Už dopředu předesílám, že při použití prediktoru MED a vhodného barevného modelu, je možné u všech testovacích obrázků (kromě `baboon.bmp`) dosáhnout lepší komprese než PNG. Dále si vyzkoušíme model LOCO-I.

Obrázek	Huffmanovo kódování	Aritmetické kódování	Velikost v PNG
monarch.bmp	500,6 kB	491,1 kB	604,2 kB
flowers.bmp	332,6 kB	320,9 kB	340,4 kB
BoatsColor.bmp	594,1 kB	588,8 kB	713,8 kB
fruits.bmp	387,9 kB	385,2 kB	410,9 kB
goldhill.bmp	593,3 kB	584,4 kB	740,3 kB

U každého obrázku máme lepší kompresní poměr než formát `.png`. Někdy je rozdíl i více jak 100 kB. A vždy je aritmetické kódování lepší než Huffmanovo. Další porovnání nechávám na uživateli. V každém případě platí, že téměř vždy (až na opravdu malé výjimky) zkomprimuje má implementace obrázek lépe než stávající formát PNG.

7.4 Úvaha nad bezeztrátovou kompresí

Bezeztrátová komprese má tři základní kroky. Převod do jiného barevného formátu, predikce hodnoty a entropické kódování. Pro kompresi obrázků volíme jiný model, než je RGB. Rozdíl mezi barevnými modely je hlavně v tom, že se více mění hodnoty na chromatických složkách, což způsobuje větší chybu prediktoru. Vhodný barevný model je tedy takový, který dokáže zminimalizovat změny hodnot na chromatických složkách. Pro lepší pochopení uvádím tabulku, ve které jsou jednotlivé kompresní poměry (dle barevného modelu) a jejich zlepšení vůči kompresnímu poměru u modelu RGB.

	RGB	LYUV	YC_oC_g	YF_rF_b	RCT	RDCT	SHIRCT	LOCO-I	NFS
K. P.	1,651	1,893	2,006	1,999	2,029	1,864	1,891	2,026	2,002
%	100	115	122	121	123	113	115	123	121

V dokumentu [2] vidíme, že procentuální vyjádření komprese je stejné jako v této tabulce. Šly by použít i techniky, které mění přeuspořádání dat a entropii.

Ke zvýšení efektivity implementace by bylo vhodné použít adaptivní entropický kodér. Ten vykazuje lepší kompresní poměr než jeho statická verze (u Huffmanova kódování). Další výhodou je pouze jeden průchod obrázkem.

Oproti formátu PNG, který má čtyři prediktory (Sub, Up, Average, Paeth), používám prediktor MED, který lépe predikuje hodnoty. V tomto je má implementace lepší. Má implementace má několik možných barevných modelů. Tato skutečnost mi umožnila dosáhnout vyššího kompresního poměru než PNG. Vzhledem k tomu, že formát PNG používá algoritmus Deflate, což je metoda založená na variantě LZ77 s Huffmanovým kódem, tak jsem implementoval aritmetické kódování, které generuje optimální kódy, a proto je lepší než Huffmanovo kódování. Tyto tři vlastnosti mé implementace (prediktor MED, převody barevných modelů, aritmetické kódování) činí kompresní poměr lepší než u PNG.

Kapitola 8

Závěr

Cílem bakalářské práce byla implementace multiplatformní knihovny pro bezeztrátovou kompresi obrazu. Podařilo se mi implementovat knihovnu, ve které je celkem devět prediktorů – prediktor MED je vybrán jako implicitní. Dále je zde osm různých barevných modelů, které lze libovolně měnit. Jako entropické kodéry jsem implementoval Huffmanovo statické kódování a aritmetické kódování. Testováním jsem zjistil, že ve velké většině případů lze nejlepšího kompresního poměru dosáhnout kombinací aritmetického kodéru, prediktoru MED a barevného modelu LOCO-I. Při této kombinaci vykazuje v drtivé většině obrázků má implementace lepší kompresní poměr než formát PNG. V některých případech je velikost komprimovaného souboru pomocí mé knihovny až o 25% menší než stejný obrázek ve formátu PNG.

V případném pokračování práce navrhuji prozkoumat využití korelace mezi měřítky obrazů (DWT – diskrétní vlnková transformace), vyzkoušet značně více prediktorů. Dalším vylepšením může být aplikace algoritmů pro přeuspořádání dat nebo změny entropie (RLE – kódování délek sledů, MTF – přesun na začátek, BWT – Burrowsova-Wheelerova transformace). Navrhuji také implementovat kódování pravděpodobnosti výskytu symbolu v nějakém kontextu (kontextové kompresní metody). V mé implementaci slučuji všechny barevné kanály do sebe a jejich hodnoty kóduji. Zde by také bylo možné vylepšení, a to pomocí kódování každého kanálu zvlášť. Protože aritmetické kódování zlepšilo kompresní poměr, bylo by vhodné doplnit knihovnu o jeho adaptivní verzi.

Literatura

[1] CORMAN, T. H.: *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001, ISBN 0-262-03293-7.

[2] EGNER, S.; VLEUTEN, R.: *Lossless and Fine-Granularity Scalable Near-Lossless Color Image Compression* [online]. (Eindhoven, The Netherlands), [cit. 2010-05-10]. URL http://www.win.tue.nl/wic2004/28.pdf

[3] GORMISH, M. J.; aj.: *Lossless and nearly lossless compression for high quality images. SPIE*, ročník 3025, (San Jose, CA), 1997: s. 62–70, ISSN 1818-2259.

[4] HAO, P.; SHI, Q.: *Comparative study of color transforms for image coding and derivation of interger reversibile color transform*. ICPR 2000, (Barcelona, Spain), 2000: s. 224–227.

[5] MALVAR, H.; SULLIVAN, G.: $YC_OC_g - R$*: A color space with RGB reversibility and low dynamic range*. JVT, no. JVT-I015e8, (San Diego, CA), 2003.

[6] NAKACHI, T.; FUJII, T.; SUZUKY, J.: *Lossless and near-lossless compression of still images*. ICIP'99, (Kobe, Japan), 1999.

[7] NOSEK, A.: *Implementace kompresní metody PPM*. Diplomová práce, Univerzita Karlova v Praze, 2006.

[8] POYNTON, C. A.: *Digital Video and HDTV:Algorithms and Interfaces*. Morgan Kaufmann, 2003, ISBN 15586607927.

[9] TOPIWALA, P.; TU, C.: *New Invertible Integer Color Transforms Based on Lifting Steps and Coding of 4:4:4 Video*. JVT, no. JVT-I0r3, (San Diego, CA), 2003.

[10] WEINBERGER, G. M.: *The LOCO-I losslesss image compression algorithm: principles and standardization into jpeg-ls. Image Procession 9*, 2000: s. 1309–1324, ISSN 1057-7149.

Dodatek A

Obsah CD

Kompaktní disk obsahuje :

- Zdrojové kódy knihovny pro bezeztrátovou kompresi obrazu
- Programovou dokumentaci knihovny
- Technickou zprávu ve formátu PDF
- Technickou zprávu v LaTeXu
- Sadu testovacích obrázků