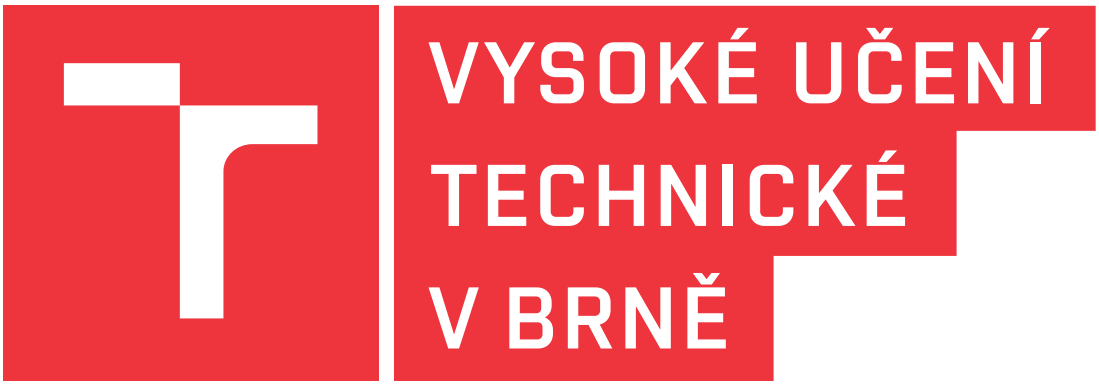


Kódování a komprese dat

Komprese obrazových dat s využitím adaptivního Huffmanova kódování

4. dubna 2021

Autoři: Petr Flajšingr, xflajs00@stud.fit.vutbr.cz
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Úvod

Cílem projektu je implementace programu pro kompresi obrázků v raw formátu za využití adaptivního Huffmanova kódování v kombinaci s adaptivním skenováním obrazu.

Návrh

Tato sekce popisuje vybrané algoritmy pro adaptivní Huffmanovo kódování a skenování obrazu.

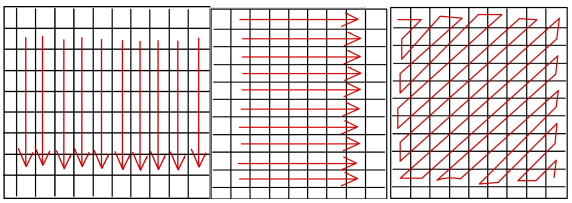
Adaptivní Huffmanovo kódování

Tento způsob kódování umožňuje kódovat data, u kterých neznáme počet výskytů symbolů. Strom se přizpůsobuje symbolům, které jsou mu předávány. Díky tomu je možné data kódovat bez dostupnosti všech data, která chceme zakódovat.

Pro implementaci jsem zvolil Vitter algoritmus. Ve stromě se používá Not Yet Transfered uzel (dále NYT), v jehož pozici se vytváří nové uzly. Pro každý nově příchozí symbol se algoritmus snaží najít jeho pozici ve stromě. Pokud je symbol nalezen, je vypočtena cesta k němu a ta je zároveň výsledným kódem reprezentující tento symbol. V případě, že symbol není nalezen je na výstup předán jeho binární kód a je vytvořen nový uzel stromu, který ho reprezentuje. Každý uzel obsahuje váhu (počet výskytů symbolu) a podle této hodnoty je strom přestrukturován při každé změně.

Adaptivní skenování obrazu

Pro adaptivní skenování obrazu jsou použity tři typy - vertikální, horizontální a tzv. 'zigzag'. Rozhodování o vhodnosti typu skenování je založeno na množství sousedních hodnot, které jsou stejné.



Obrázek 1: Metody skenování obrazu

Implementace

V této sekci je popsána implementace adaptivního Huffmanova kódování a adaptivního skenování obrazu.

Projekt je implementován v jazyce `c++` (standard `c++20`). V kódu je hojně využíváno template metaprogramování, díky čemuž je implementována i částečná podpora symbolů v jiné abecedě než `uint8_t`. Tato podpora ovšem není plně funkční. Celá implementace se nachází v `namespace pf::kko`.

Ve zdrojových souborech je obsažena i implementace statického Huffmanova kódování, ale ta nebyla v zadání a proto zde není popsána.

Adaptivní Huffmanovo kódování

Společné struktury pro kódování a dekódování (práce se stromem) jsou umístěny v souboru `adaptive_common.h`. Implementován je Viterr algoritmus, který je popsaný v předchozí sekci. Pro urychlení zpracování je použita cache k rychlejšímu dohledání uzlů podle symbolu - není nutné procházet celý strom. Data z disku jsou čtena za využití třídy `RawGrayscaleImageDataReader`. Pro strom je využita generická struktura `Tree<T>`, která binární strom. Pro zápis binárních dat existuje třída `BinaryEncoder`, která umožňuje serializaci dat, či přímý zápis bitových hodnot.

Pro statické skenování jsou ve výstupních datech pouze data nutná k rekonstrukci stromu - kódy symbolů, či Huffmanův kód. Implementace je v souborech `adaptive_encoding.h` a `adaptive_decoding.h`.

Pro adaptivní skenování strom obsahuje hlavičku - 4B velikost obrázku, 2B velikost bloku. Dále každý blok obsahuje hlavičku, kdy jsou využity 3 bity pro uložení typu průchodu.

Adaptivní skenování

Adaptivní skenování je implementováno třídou `AdaptiveImageScanner`. Třída přijímá v konstruktoru vstupní data, implementace konceptu `BlockScorer` pro hodnocení vhodnosti typu průchodu bloku a implementaci konceptu `Model` pro transformaci dat. Třída je implementována jako `range` objektů typu `Block`. Tento objekt reprezentuje vyhrazenou oblast ve vstupních datech, která je také `range`. Pomocí iterátorů lze procházet data bez závislosti na typu průchodu.

Pro hodnocení vhodnosti jsem implementoval 2 typy. Prvním je `SameNeighborsScorer`, který má tím vyšší skóre, čím vícekrát se po sobě opakují symboly v sekvenci. Druhým je `NeighborDifferenceScorer`, který hodnotí nejlépe ten typ průchodu, kdy je součet rozdílů všech sousedních dat nejnižší. Vhodnějším se ukázal být `SameNeighborsScorer`.

Model

Implementace modelů je limitována konceptem `Model`. Statický polymorfismus namísto dynamického byl zvolen kvůli výrazně lepší optimalizaci - např. `IdentityModel` je optimalizátorem naprosto odstraněn. `Model` vyžaduje 2 metody; `apply(T)` pro transformaci a `revert(T)` pro zpětnou transformaci.

Použité knihovny

Program využívá následující knihovny:

- `libfmt` [1] – formátování řetězců.
- `spdlog` [2] – logování.
- `argparse` [3] – zpracování argumentů terminálu.
- `tl::expected` [4] – implementace P0323R8 návrhu pro c++.

[1] https://github.com/fmtlib/fmt
[2] https://github.com/gabime/spdlog
[3] https://github.com/p-ranav/argparse
[4] https://github.com/TartanLlama/expected

- `magic_enum` [5] – statická reflexe pro enumy.
- `nanobench` [6] – benchmark.

Vyhodnocení

Doba zpracování

Měření bylo prováděno na serveru `merlin.fit.vutbr.cz` za pomocí knihovny `nanobench` - knihovna sama detekuje počet opakování pro zaručení stability výsledku.

Časy provádění kódování, uváděno v milisekundách(SH = static Huffman, AH = adaptive Huffman, AS = adaptive scanning):

Soubor	SH	SH + model	AH	AH + model	AH + AS	AH + AS + model
df1h.raw	10.1	2.1	2547.8	15.7	2501.4	117.8
df1hvx.raw	4.7	2.9	340.5	397.9	363.9	47.1
df1v.raw	7.7	2.1	1912.7	18.6	1971.3	98.8
hd01.raw	4.8	4.9	1426.2	1300.2	1460.3	1301.6
hd02.raw	4.9	4.9	1448.5	1347.9	1417.7	1290.3
hd07.raw	6.1	5.5	1823	793.4	1847.9	1188.6
hd08.raw	4.6	5.1	489.2	798.1	502.4	829.6
hd09.raw	6.9	6.6	2010.4	1532.6	2028.8	1633.1
hd12.raw	8.4	6.2	2074	1122.2	2131.7	1461.5
nk01.raw	8	8.1	2088.2	1881	1949.3	1913

Časy provádění dekódování, uváděno v milisekundách(SH = static Huffman, AH = adaptive Huffman, AS = adaptive scanning):

Soubor	SH	SH + model	AH	AH + model	AH + AS	AH + AS + model
df1h.raw	3.9	0.9	2301.3	4.2	2261.8	55.9
df1hvx.raw	2.4	1.4	286.3	234.6	301.8	22.8
df1v.raw	3.9	0.9	1754.9	4.9	1796.1	48.8
hd01.raw	2.7	2.9	1112.5	1011.1	1184.4	1070.7
hd02.raw	2.7	2,8	1094.3	989.3	1106.1	1002.9
hd07.raw	3.7	3.5	1536.3	685.3	1530.7	1050.7
hd08.raw	2.7	3.3	399.9	618.8	409.2	624.4
hd09.raw	4.6	4.8	1841	1312.7	1864.2	1415.5
hd12.raw	3.8	3.7	1865	957.8	1812.2	1306.7
nk01.raw	5	5.1	1737.1	1814	1854.5	1761.3

Úroveň komprese

Následující tabulka zobrazuje BPC (bits per character) pro jednotlivé typy komprese. (SH = static Huffman, AH = adaptive Huffman, AS = adaptive scanning).

[5] https://github.com/Neargye/magic_enum
[6] https://github.com/martinus/nanobench

Soubor	SH	SH + model	AH	AH + model	AH + AS	AH + AS + model
df1h.raw	8.0079	1.00015	8.0127	1.00006	8.05975	1.28992
df1hvx.raw	4.58051	1.83655	4.58453	1.83804	4.63165	1.40891
df1v.raw	8.0079	1.00015	8.0127	1.00204	8.06021	1.31583
hd01.raw	3.88	3.40598	3.88501	3.40964	3.93228	3.4917
hd02.raw	3.70486	3.3324	3.71033	3.3367	3.75735	3.38239
hd07.raw	5.61279	3.85666	5.61771	3.85974	5.66525	4.01248
hd08.raw	4.23663	3.52628	4.24118	3.52911	4.28848	3.43051
hd09.raw	6.65979	4.68561	6.6669	4.68961	6.7142	4.80814
hd12.raw	6.20026	4.39148	6.20694	4.39508	6.25467	4.48077
nk01.raw	6.50854	6.06931	6.51367	6.07309	6.56082	5.90204

Ne příliš dobré výsledky pro adaptivní skenování jsou pravděpodobně následek nevhodného hodnocení při výběru typu průchodu blokem obrazu.

Překlad a použití programu

Překlad je prováděn pomocí příkazu `make`. V defaultním režimu je vytvořen program `huff_codec`, jehož použití je následující:

```
Usage: huff_codec [options]

Optional arguments:
-h --help           show this help message and exit
-c                  Compress input file
-d                  Decompress input file
-m                  Activate model for preprocessing
-a                  Adaptive scanning
-i                  Path to input file[Required]
-o                  Path to output file[Required]
--static            Static huffman
-w                  Input image width[Required]
```

Další možností je `make bench`. Tímto příkazem je vytvořen program `bench`, který slouží k verifikaci algoritmu a výpočtu úrovně komprese či změření doby běhu algoritmů. Použití programu:

```
Usage: bench [options]

Optional arguments:
-h --help           show this help message and exit
-i                  Path to folder with input files[Required]
--validate          Test encoding validity
```

Literatura

[1] Data compression. Navštíveno 4.4.2021, https://www.ics.uci.edu/~dan/pubs/DC-Sec4.html.