Dokumentácia projektu do predmentu Kódovanie a kompresia dát: Huffmanovo kódovanie

Juraj Ondrej Dúbrava (xdubra03)

29. apríla 2019

1 Úvod

Cieľom projektu bolo implementovať v jazyku C/C++ program na kódovanie a dekódovanie súborov použitím statického a adaptívneho Huffmanovho kódovania. Ako ďalší požiadavok bola implementácia modelu slúžiaceho na predspracovanie dát. Na tento účel bol použitý diferenčný model, ktorý pracuje s hodnotami susedných pixelov a uskutočňuje ich rozdiel, ktorý je použitý ako výsledná hodnota. Tým je dosiahnutý menší rozptyl spracovávaných hodnôt.

2 Huffmanovo kódovanie

Huffmanov kód je prefixový kód s premenlivou dĺžkou a používa sa pri bezstratovej kompresii dát. Proces získania tohto kódu sa nazýva Huffmanovo kódovanie. Implementovaný program realizuje 2 typy Huffmanovho kódovania: statické a adaptívne. Jednotlivé metódy sú stručne popísané v nasledujúcich častiach.

2.1 Statické kódovanie

Statická metóda tvorí prefixový kód na základe frekvencií výskytu jednotlivých symbolov vo vstupných dátach. Pred kódovaním je potrebné poznať frekvenciu všetkých symbolov. Symbolom, ktoré sa vyskytujú menej často priraďuje dlhšie kódy a často sa vyskytujúcim symbolom naopak kratšie kódy. Metóda pracuje s vytvoreným zoznamom frekvencií, ktorý je zoradený zostupne. Následne sa zdola nahor vytvára strom, v listoch sa nachádzajú jednotlivé symboly. V každom kroku tvorby stromu sa vyberú 2 uzly pre symboly s najmenšou frekvenciou, vytvorí sa nový uzol s frekvenciou rovnou súčtu frekvencií vybraných uzlov a tieto uzly sa odtránia zo zoznamu symbolov a nahradia sa 1 symbolom, ktorý novo reprezentuje dané symboly. Končí sa v prípade, že ostane len 1 uzol. Kód sa získa priechodom stromom od vrcholu k listu s daným symbolom.

2.2 Adaptívne kódovanie

Adaptívna metóda tvorí strom priamo pri procese spracovania vstupu, nie je potrebné poznať frekvencie symbolov. V priebehu spracovania tak symbol nadobúda rôzne kódy kvôli procesu tvorby a úprave stromu, ktorý musí splňovať určité vlastnosti. Existuje niekoľko algoritmov adaptívneho kódovania, napr. FKG alebo Vitter. Metódy majú rozličné požiadavky na vlastnosti stromu pri jeho zostavovaní. Pri implementácii adaptívneho kódovania bol použitý algortimus FGK a proces kompresie a dekompresie bol inpirovaný textom [1].

3 Implementácia

Program je rozdelený do modulov realizujúcich jednotlivé funkcie pre daný typ kódovania. Hlavný program, kde sú vyberané jednotlivé metódy na kompresiu a dekompresiu podľa zvolených parametrov, je v súbore *huffman_codec.cpp*

[1] `https://www2.cs.duke.edu/csed/curious/compression/adaptivehuff.html`

3.1 Statické kódovanie

Metódy realizujúce statické kódovanie a dekódovanie, ako aj pomocné metódy, sa nachádzajú v súbore *huffman_static.cpp*. Pomocné štruktúry a deklarácie funkcií sa nachádzajú v súbore *huffman_static.h*. Základnou štruktúrou pre tento typ kódovania je *TreeNode*, reprezentujúca uzol stromu. Pre dáta uzlu bol použitý dátový typ *uint32_t*, aby bolo možné reprezentovať pseudo EOF. Pre frekvencie bol použitý typ *size_t*, aby bolo možné reprezentovať veľké hodnoty frekvencií, aj keď pri veľmi veľkom počte výskytov nemusí stačiť. Uzol ešte obsahuje ukazatele na ľavého a pravého potomka. Tvorba huffmanovho stromu je implementovaná použitím prioritnej fronty, do ktorej sú na začiatku vložené uzly pre jednotlivé symboly a taktiež pre symbol EOF. Tieto uzly reprezentujú listy, tvorba stromu prebieha od listov ku koreňu. Pri procese tvorny stromu sa v cykle vyberú 2 uzly s najnižšími frekvenciami, odstránia sa z fronty, následne je vytvorený nový vnútorný uzol s hodnotou dát 257 a frekvenciou s hodnotou súčtu frekvencií práve spracovávaných uzlov, ktorý je novo pridaný do fronty. Strom je vytváraný funkciou *create_huffman_tree* a vracia ukazateľ na koreňový uzol.

Proces kódovania je implementovaný vo funkcii *encode_raw_static*. Na začiatku sú načítané do pamäti vstupné dáta funkciou *read_raw* a následne sú spracovávané. Prvým krokom po načítaní dát je vytvorenie hash mapy funkciou *get_raw_freqs*, ktorá udržuje frekvencie jednotlivých symbolov. Nasleduje vytvorenie huffmanovho stromu na základe frekvencií podľa vyššie uvedeného spôsobu. Po tejto fáze nasleduje vytvorenie ďalšej hash mapy s kódom pre každý symbol pomocou funkcie *create_codes*, kód je reprezentovaný vo forme reťazca. Poslednou fázou je zakódovanie vstupu získanými kódmi, čo je uskutočnené vo funkcii *encode*, kde sa vytvorí reťazec, ktorý reprezentuje zakódovaný vstup a následne je spracovávaný po znakoch reprezentujúce bity kódu a postupne zapísaný do súboru. Pred samotnými zakódovanými dátami je potrebné uložiť slovník pre potreby dekodéru. Ako prvá je zapísaná veľkosť ukladaného slovníka, pretože sa neukladajú frekvencie pre všetkých 256 hodnôt, ale len pre tie, ktoré sú vo vstupných dátach prítomné. Následne sa ukladajú dvojice symbol a frekvencia, symbol sa uloží na 2 bajty a frekvencia na 4 bajty. Nakoniec je zapisovaná zakódovaná správa.

Proces dekódovania implementuje funkcia *decode_raw_static*. Vstup je načítavaný po bajtoch, najprv sa načíta veľkosť slovníka a načíta sa príslušný počet bajtov reprezentujúcich dvojice slovníka. Po jeho zostavení sa vytvorí strom funciou ako v prípade kódovania a začína prebiehať dekódovanie. V cykle sa načítavajú bajty vstupu a podľa hodnoty jednotlivých bitov sa prechádza strom na zistenie odpovedajúceho symbolu pre načítaný kód. Musí sa načítavať a prechádzať stromom až pokiaľ sa nenarazí na listový uzol značiaci dáta. Ak sa narazí na uzol s hodnotou dát 256, čo reprezentuje pseudo EOF, dekódovanie bolo úspešné.

Funkcie na kódovanie a dekódovanie majú okrem parametrov, ktoré špecifikujú vstup a výstup aj parameter, či sa má s dátami pracovať s použitím modelu. V prípade kódovania je vstup načítaný do pamäti, aby bolo možné modifikovať dáta použitím modelu. Predspracovanie dát pred kódovaním sa deje pomocou funkcie *use_model_encoding*. V prípade dekódovania a použitia modelu nie sú dekódované dáta ihneď zapisované do výstupného súboru, ale vytvára sa pomocný vektor, kde sa dáta najprv uložia a následne sú použítím funkcie *use_model_decoding* vrátené do pôvodných hodnôt ako pred použitím modelu.

3.2 Adaptívne kódovanie

Metódy adaptívneho kódovania a dekódovania sa nachádzajú v súbore *huffman_adaptive.cpp*, pomocné štruktúry a deklarácie v súbore *huffman_adaptive.h*. Použitým algoritmom adaptívneho kódovania je algoritmus FGK. Tento algoritmus začína s prázdnym stromom, ale je možné začať aj s jedným uzlom, čo je implementované v tomto prípade. Týmto uzlom je tzv. NYT (not yet transmitted) uzol, ktorý má hodnotu 256. Pomocou neho sa indikuje prvý výskyt symbolu. Algoritmus FGK ďalej požaduje zaručenie tzv. súrodeneckej vlastnosti stromu, čo znamená, že každý uzol má súrodenca a uzly je možné zoradiť do neklesajúcej postupnosti zľava doprava a zdola nahor. Podľa vyššie uvedeného textu sa zoradenie do neklesajúcej postupnosti zaručí použitím poradia uzlov, tak, že uzly s vyšším číslom budú mať vyššiu váhu a koreňovaý uzol bude mať najvyššiu váhu. Poradie je potrebné brať do úvahy pri vkladaní uzlu do stromu a pri úprave stromu.

Základnou štruktúrou použitou pri implementácii je *AdaptiveNode*, reprezentujúci uzol stromu. Obsahuje dáta, váhu, odkazy na rodiča, ľavého a pravého syna, ktoré sú realizované ako celočíselné indexy do poľa, ktoré reprezentuje strom. Dáta sú reprezentované dátovým typom *uint32_t* a váha symbolu typom *size_t*. Strom je reprezentovaný ako statické pole ukazateľov na uzly o veľkosti 2*počet symbolov (počet symbolov je 257), čo je maximálny možný počet uzlov stromu pri danom počte symbolov v abecede, indexy určujú poradie uzlu.

Proces kódovania je implementovaný vo funkcii *encode_raw_adaptive*. Na začiatku sú dáta načítané do pamäti funkciou *read_raw* aby ich bolo možné predspracovať v prípade použitia modelu. Ak parameter *use_model* indikuje použitie modelu, dáta sú najprv predpracované funkciou *use_model_encoding_adaptive*.

Po načítaní dát nasleduje krok vytvorenia a inicializácie stromu. Strom je inicializovaný jedným uzlom reprezentujúcim tzv. NYT (not yet transmitted) symbol s hodnotou 256 a váhou 0. Tento uzol je uložený na index 2*N-1, N je počet symbolov abecedy. Následne prebieha spracovanie bufferu so vstupnými dátami. Pomocou funkcie *first_data_occurence* sa zistí, či ide o prvý výskyt symbolu, t.j. overí sa prítomnosť uzlu s danou hodnotou dát v poli uzlov. Ak ide o prvý výskyt, symbol NYT je použitý ako indikátor prvého výskytu a zapíše sa na výstup funkciou *save_encoded_symbol*. Tá pracuje s výstupným bufferom a pozíciou v ňom. Pred samotným zápisom je potrebné získať kód pre spracovávaný symbol. To realizuje funkcia *get_code*, ktorá vracia vektor booleovských hodnôt reprezentujúcich kód. Následne je sú vo výstupnom bufferu nastavované jednotlivé bity na základe kódu a ak je buffer naplnený (je k dispozícií 1 bajt), jeho obsah je zapísaný do výstupného súboru. Pri prvom výskytuje je ďalej pridaný do poľa uzlov nový uzol s hodnotou načítaného symbolu. Ten sa stane pravým synom uzlu, ktorý predtým reprezentoval symbol NYT a ľavým synom sa stane uzol novo reprezentujúci NYT. Pravý syn je ukladaný vždy na index i-1 a ľavý syn na index i-2, čím dosiahneme zachovanie poradia uzlov. Po NYT symbole nasleduje v súbore nezmenená 8 bitová reprezentácia hodnoty symbolu, zápis sa uskutoční opäť funkciou *save_encoded_symbol*. V prípade, že nejde o prvý výskyt symbolu, funkciou *save_encoded_symbol* je zapísaný kód pre spracovávaný symbol.

Po zápise prvého alebo ďalšieho výskytu nasleduje krok aktualizácie stromu. Algoritmus FGK vyžaduje dodržanie tzv. súrodeneckej vlastnosti stromu uvedenej vyššie. Samotný proces aktualizácie realizuje funkcia *update_tree* a prebieha po každom spracovaní symbolu vstupu. Začína v uzle pre spracovávaný symbol a končí až v koreňovom uzle. Aktualizácia začína krokom vyhľadania uzlu v poli uzlov s rovnakou frekvenciou a indexom, ktorý je väčší ako index spracovávaného uzlu a zároveň najväčší možný pre danú frekvenciu. Hľadanie začína od indexu spracovávaného uzlu po maximálny index, kde indexy určujú poradie ako bolo uvedené vyššie. V prípade, že nájdeme uzol, ktorý je rozdielny od spracovávaného uzlu a zároveň nie je rodičom tohto uzlu, bol nájdený prípad porušenia súrodeneckej vlastnosti. To znamená, že sa má uskutočniť výmena týchto uzlov, t.j. budú uložené na indexe toho druhého. Ďalším krokom je zvýšenie hodnoty frekvencie spracovávaného uzlu o 1 a pokračovanie v aktualizácii smerom ku koreňu. Po tejto fáze dostávame buď rovnaké alebo pozmenené kódy, ktoré odrážajú poradie uzlov a teda aj ich hodnotu frekvencie. Po zápise kódov symbolov správy je na koniec zapísaný symbol NYT ako EOF symbol na rozpoznanie konca kódovaných dát.

Proces dekódovania je implementovaný vo fukncií *decode_raw_adaptive*. Vstup je načítavaný priamo zo súboru po jednotlivých bajtoch. Po načítanú bajtu sa spracovávajú jednotlivé bity. Na začiatku procesu je nastavený príznak *reading_NYT* na true, ktorý indikuje, že bol prečítaný kód pre symbol NYT a má sa načítať nezakódovaná hodnota prvého výskytu symbolu. To je dôležité z pohľadu prvej načítavanej hodnoty, pretože prvou hodnotou dekódovaného súboru je hodnota prvého symbolu, ktorý nie je zakódovaný, pretože pri ukladaní vo fáze kódovania nemal uzol NYT pred načítaním prvého symbolu vstupu priradený kód, ktorý by mohol byť zapísaný. Ak je príznak pravdivý, načíta sa hodnota symbolu a tá je zapísaná do výstupného súboru. Pri použití modelu sa dáta nezapisujú, ale ukladajú sa do pomocného vektoru, ktorý bude na konci upravený. V prípade, že príznak čítania symbolu NYT nie je pravdivý, podľa hodnoty aktuálne spracovávaného bitu vstupného bajtu sa prechádza pole uzlov (strom) pokiaľ sa nenarazí na listový uzol s hodnotou symbolu, ktorý odpovedá kódu. Ak je načítaný kód pre NYT, nastaví sa príznak *reading_NYT* na pravdivý, v prípade načítania kódu iného symbolu sa hodnota symbolu zapíše na výstup, alebo sa pridá do pomocného vektoru ak je použitý model. Po fázach načítania kódu symbolu alebo načítania nekódovanej hodnoty symbolu za kódom NYT symbolu vždy nasleduje aktualizácia stromu. Ak je na konci dekódovania príznak *reading_NYT* pravdivý, značí to, že posledným načítaným symbolom bol NYT, ktorý slúži ako EOF.

4 Vyhodnotenie

Implementované verzie huffmanovho kódovania boli vyhodnotené na priložených súboroch typu RAW o veľkosti 512x512 pixelov s veľkostou 262.1 kB. Parametre, ktoré boli použité na vyhodnotenie sú priemerný počet bitov potrebných na zakódovanie 1 bajtu, dosiahnutý kompresný poemer, čas kódovania a dekódovania. Pri vyhodnocovaná času kompresie a dekompresie bolo pre kopmresiu aj dekompresiu uskutočnených 5 meraní a z nich priemer ako výsledná hodnota. Všetky merania boli uskutočnené na serveri Merlin. Výsledky vyhodnotenia je možné vidieť v tabuľkách 1, 2 a 3.

	Bez modelu		S modelom	
	Priemerný počet bitov na zakódovanie 1 bajtu			
Súbor	Static	Adaptive	Static	Adaptive
HD01.raw	3.94	3.88	3.47	3.40
HD02.raw	3.76	3.70	3.39	3.33
HD07.raw	5.67	5.61	3.96	3.84
HD08.raw	4.26	4.24	3.56	3.53
HD09.raw	6.72	6.67	4.74	4.69
HD12.raw	6.26	6.20	4.44	4.39
NK01.raw	6.56	6.51	6.07	6.07

Tabuľka 1: Tabuľka priemerného počtu bitov na bajt pre jednotlivé metódy

	Bez modelu				S modelom			
	Kompresia		Dekompresia		Kompresia		Dekompresia	
Súbor	Static	Adaptive	Static	Adaptive	Static	Adaptive	Static	Adaptive
HD01.raw	0.0336 s	0.948 s	0.019 s	0.695 s	0.031 s	0.959 s	0.021 s	0.716 s
HD02.raw	0.0372 s	0.966 s	0.0266 s	0.699 s	0.038 s	0.91 s	0.03 s	0.68 s
HD07.raw	0.0472 s	1.191 s	0.0298 s	0.933 s	0.0426 s	1 s	0.0338 s	0.726 s
HD08.raw	0.0368 s	0.934 s	0.306 s	0.696 s	0.0366 s	0.917 s	0.0224 s	0.64 s
HD09.raw	0.0448 s	0.219 s	0.0264 s	0.998 s	0.0382 s	1.081 s	0.0246 s	0.759 s
HD12.raw	0.0432 s	1.283 s	0.023 s	0.96 s	0.034 s	1.057 s	0.0242 s	0.758 s
NK01.raw	0.0528 s	1.238 s	0.0258 s	0.994 s	0.046 s	1.245 s	0.0316 s	0.968 s

Tabuľka 2: Tabuľka časov kompresie a dekompresie jednotlivých metód

	Bez modelu				S modelom			
	Veľkost po kompresii		Kompresný pomer		Veľkosť po kompresii		Kompresný pomer	
Súbor	Static	Adaptive	Static	Adaptive	Static	Adaptive	Static	Adaptive
HD01.raw	129.2 kB	127.3 kB	49.2 %	48.5%	113.7 kB	111.7 kB	43.3 %	42.6 %
HD02.raw	123.5 kB	121.5 kB	47.1 %	46.3 %	111.2 kB	109.3 kB	42.4 %	41.7 %
HD07.raw	186 kB	184 kB	70.9 %	70.2 %	128 kB	126 kB	48.8 %	48 %
HD08.raw	139.6 kB	139 kB	53.3 %	53 %	116.9 kB	115.6 kB	44.6 %	44.1 %
HD09.raw	220.2 kB	218.4 kB	84 %	83.3 %	155.5 kB	153.6 kB	59.3%	58.6 %
HD12.raw	205.2 kB	203.4	78.3 %	77.6 %	145.6 kB	144 kB	55.6 %	54.9 %
NK01.raw	215.2 kB	213.4 kB	82.1 %	81.4 %	200.9 kB	199 kB	76.7 %	75.9 %

Tabuľka 3: Tabuľka dosiahnutej kompresie pre jednotlivé metódy