Dokumentace zápočtového programu (PRG030)

KOMPRESE TEXTU

Petr Koupý, INF 1/X/32

Anotace

Program je primárně navržen ke komprimování textových souborů. Protože však místo textových znaků pracuje přímo s byty, lze jej použít na kompresi libovolného souboru. Vzhledem k použitým algoritmům bude komprese efektivní zejména na souborech obsahující nenáhodný text a bitmapách s rozsáhlými homogenními plochami. Program je ovládán pomocí parametrů příkazového řádku a při běhu neočekává žádnou interakci s uživatelem. Je tedy vhodný pro použití v dávkových souborech. Volitelně může tvořit logovací soubor, do kterého ukládá informace o činnosti, chybách a některých datových strukturách.

Algoritmus

Huffmanovo kódování je vhodné zejména na soubory, ve kterých frekvence výskytu některých bytů výrazně převyšuje frekvenci ostatních. Je tedy použitelné zejména na texty psané lidským jazykem, kde tuto roli hrají některé samohlásky, případně mezery a nejrůznější formátovací znaky. Byty s největší frekvencí výskytu je vhodné reprezentovat co nejméně bity. Naopak na reprezentaci vzácných bytů lze využít i více než 8 bitů. Pro reprezentace jednotlivých bytů je výhodné použít binární stromovou strukturu (*Huffmanův strom*), kde byty odpovídají listům stromu. Byty s vysokou frekvencí výskytu se nacházejí blíže ke kořeni. Samotná jejich reprezentace odpovídá průchodu stromem od kořene k listům, kde 0 znamená zvolení levého syna a 1 znamená zvolení pravého syna.

1) Projde se vstupní soubor a vytvoří se pole obsahující frekvence výskytu pro každý z možných 256 bytů.
2) Byty o nenulové frekvenci se naskládají do pole, ze kterého je následně vytvořena halda. Byty s vysokou frekvencí probublávají do nižších pater haldy. V kořeni haldy je tedy byte s nejnižším výskytem.
3) K uložení Huffmanova stromu se použije dvou polí. Jednak bude rozšířeno původní pole frekvenčního výskytu, které bude obsahovat hodnoty jednotlivých uzlů stromu (u listů tyto hodnoty stále odpovídají frekvenci, u všech dalších uzlů odpovídají součtu hodnot svých synů). Druhé pole bude udržovat informace o struktuře stromu, tedy o návaznostech každého uzlu na svého otce – prvek pole bude indexem do pole hodnot (frekvencí). Pokud bude tento index záporný, znamená to, že daný uzel je pravým synem svého otce, naopak bude-li kladný, jedná se o levého syna. Samotná tvorbu stromu bude probíhat následovně. Z vrcholu haldy se pokaždé odeberou dva prvky s nejnižším výskytem, tyto jsou sečteny, součet se uloží jakožto otec do pole hodnot a vzápětí také na vrchol haldy, odkud probublá na svoje správné místo. V každém cyklu se tedy halda zmenší o jeden prvek, až nakonec zůstane pouze jednoprvková halda, která obsahuje součet všech svých původních prvků. Index otce v poli hodnot je součtem aktuální velikosti haldy a hraničního indexu pro otce (před tímto hraničním indexem jsou již uloženy frekvence listů). Celkově se tedy v každém cyklu uloží do pole hodnot otec a pro oba jeho syny se ve strukturním poli uloží jeho index.
4) Pro snadnější kódování textu se z nynější reprezentace Huffmanova stromu vytvoří další dvě pole. Pro každý list (byte) se projde stromem nahoru a bude se ve formě binárního čísla zaznamenávat, zda se do vyšší úrovně přichází zleva (0) či zprava (1). Výsledné binární číslo se zarovná na předem určenou délku, která musí být dostatečná pro každou možnou cestu v Huffmanově stromě, a uloží se do kódovacího pole. Původní délka binárního čísla se uloží

do délkového pole. Tato dvě pole se uloží do výstupního souboru, protože nesou nezbytné informace pro rekonstrukci Huffmanova stromu.

5) Pro každý byte vstupního souboru se zjistí jeho kódovaná podoba a tato je uložena do výstupního souboru. Kódovaná podoba odpovídá příslušnému údaji v kódovacím poli, který se ořízne dle údaje v délkovém poli.
6) Při dekompresi se napřed načte kódovací a délkové pole a rekonstruuje se Huffmanův strom. Nyní je vhodné pro binární strom místo polí použít dynamickou datovou strukturu a pro každou položku kódovacího a délkového pole touto strukturou procházet dle binárního čísla v kódovacím poli a postupně tvořit neexistující uzly. Vždy když je vyčerpána délka z délkového pole, je do struktury uložen index těchto polí, který odpovídá kódovanému bytu.
7) Pro každý bit ze zkomprimovaného souboru postupně procházet Huffmanovým stromem od kořene. Vždy když je nalezen list, dojde k dekódování a uložení daného bytu a návratu do kořene stromu.

Run-Lenght Encoding (RLE) je určen na soubory, ve kterých se dá předpokládat dlouhé sekvence stejných znaků (bytů). Typickým zástupcem takového souboru je obrazová bitmapa. Každá dlouhá sekvence stejných bytů lze reprezentovat pouze počtem opakování a bytem samotným, čímž se ušetří spoustu místa. Aby se takovou komprimovanou sekvenci dalo odlišit od nekomprimovaných, je uvedena tzv. *escape characterem*, který by měl být v daném souboru velmi vzácným, případně se tam vůbec nevyskytoval.

1) Projde se vstupní soubor a vytvoří se pole obsahující frekvence výskytu pro každý z možných 256 bytů.
2) V poli frekvencí se nalezne byte s nejnižší frekvencí. Tento se stane escape characterem. Je výhodné jej uložit na začátek výstupního souboru.
3) Prochází se vstupním souborem a počítá se délka souvislých sekvencí stejných bytů. Jakmile je načten byte, který již do sekvence nelze zařadit, je sekvence uložena a čítač resetován. Pokud byl čítač větší než 3, je sekvence uložena ve formě (escape character | počet opakování | daný byte). V opačném případě by se komprese nevyplatila a byte je uložen přímo. Jedinou výjimkou z výše uvedeného postupu je načtení bytu, který odpovídá escape characteru. V takovém případě se escape character ukládá ve formě (escape character | nulový byte). Jako jediný tedy zabírá více místa než ve vstupním souboru.
4) Při dekompresi se načte escape character a postupně se prochází souborem. Vždy když je nalezen escape character, je načten počet opakování. Pokud je nulový, výsledkem je escape charactere, je-li nenulový, načte se následující byte a dle počtu opakování je z něj vytvořena dekomprimovaná sekvence. Pokud je načten jakýkoliv jiný byte, je uložen přímo.

Diskuse výběru algoritmu

Rozmýšlel jsem se mezi implementací Huffmanova kódování a slovníkového LZW. Nakonec zvítězilo Huffmanovo kódování spíše kvůli subjektivní sympatii. Jestli bude možnost, k LZW bych se rád někdy v budoucnu vrátil. Původně jsem plánoval implementovat pouze jeden algoritmus, ale po schválení specifikace jsem byl vyzván k implementaci dvou algoritmů. Protože jsem si nebyl jistý časovou náročností na pochopení dalšího pokročilého algoritmu, zvolil jsem jednoduché RLE, které bylo sázkou na jistotu.

Program

Procedury a funkce programu by se daly roztřídit do několika skupin dle svého účelu – řídící, vstupně-výstupní, interface pro kompresní algoritmy, a nakonec samotné kódovací či dekódovací procedury. Hlavní program se částečně stará o kontrolu zápisu parametrů. Pokud by dle parametrů nebyl schopen dále pokračovat, zobrazí nápovědu. Jinak zavolá funkce pro otevření vstupního souboru a výstupního souboru. Pokud tyto ohlásí úspěch, je dále dle parametrů rozhodnuto, zda se bude komprimovat či dekomprimovat a následně jsou zavolány řídící procedury, které dle dalších parametrů zvolí daný algoritmus. Pro kódovací a dekódovací procedury je určen jednotný univerzální

interface, který je tvořen procedurami pro načítání a ukládání dat, pohyb ve vstupním souboru, či výpočet kompresního poměru. Jakmile je komprese nebo dekomprese dokončena, dojde k návratu do hlavního programu, který zavře všechny otevřené soubory a ukončí se.

Vstupně-výstupní procedury jsou vytvořeny tak, aby správně reagovali na nestandardní situace. Při otevírání souborů tedy dochází ke kontrole, zda existují. Pokud se navíc má existující soubor přepsat, je o tom rozhodnuto dle parametru pro přepis. Pokud tedy uživatel takový parametr při spuštění programu nezapsal, existující soubor se z bezpečnostních důvodů přepisovat nebude a program skončí. Do této skupiny procedur patří také procedura pro založení logovacího souboru *log.txt*, jehož vznik je opět podmíněn volitelným parametrem při spuštění programu. Tato procedura se již neohlíží na existenci předchozího log souboru a vždy jej přepíše.

Řídící procedury jsou velmi krátké a tvoří pouze křižovatku, na které se rozhodne jaký konkrétní kompresní algoritmus zvolit. Lze je v budoucnu rozšířit, pokud by bylo třeba implementovat další kompresní algoritmy.

Pro každý kompresní algoritmus existuje kódovací a dekódovací procedura, která uvnitř sebe zastřešuje potřebné množství dalších procedur a datových struktur, které jsou specifické pro daný kompresní algoritmus. Tyto procedury nevyužívají přímo žádné datové struktury, které nejsou jejich součástí. Jediný způsob, jakým komunikují s okolím, je volání interfacových procedur, které jim poskytují či od nich přebírají data. Fungování mnou implementovaných procedur je vysvětleno v části o popisu algoritmů. Zde bych jen doplnil, že obě dekódovací procedury jsou doplněny o kontrolu chyb, takže jakmile vstupní data neodpovídají tomu, co algoritmus očekává, program se ukončí a případně uloží informaci o této chybě do log souboru.

Interface je tvořen především procedurami pro načítání a ukládání dat. Konkrétně jsou implementovány procedury pro načtení a uložení jednobytových a dvoubytových dat. Dále jsou zde procedury pro pohyb ve vstupním souboru, výpočet kompresního poměru a logování činnosti programu. Protože některé kompresní algoritmy potřebují pracovat s daty na úrovni bitů, jsou zde také procedury, které rozloží vstupní data na jednotlivé bity nebo naopak složí pole bitů do výstupní datové jednotky. Dle výše zmíněného jsou zatím implementovány pouze procedury na rozložení či složení jednobytových či dvoubytových dat. Některé z těchto interfacových procedur by se dalo mírně vylepšit. Například procedura pro logování by mohla každou hlášku doplnit o časový údaj. Nic také nebrání doplnění interface o další procedury, které by se při implementaci jiných kompresních algoritmů ukázaly jako potřebné.

Alternativní programová řešení

Pro vytvoření programu byl použit jazyk Pascal. Bohužel jsem se až v průběhu vývoje dozvěděl, že Pascal nedovoluje přímou práci s bity při načítání a ukládání souborů, na kterou jsem zvyklý z jazyka C. Kvůli tomu bylo nutné implementovat procedury, které budou tuto funkčnost suplovat. V těchto procedurách se byte (případně word) rozkládá na bity klasickým způsobem div 2, mod 2. To jistě není příliš efektivní a je to i znát na výsledném výkonu programu. Zkomprimovat soubor větší než 1MB Huffmanovým kódováním trvá řádově minuty.

Další výraznou výkonovou brzdou je procházení celého souboru za účelem výpočtu frekvencí. U Huffmanova kódování by toto šlo eliminovat použitím nějaké defaultní frekvenční tabulky (např. pro český jazyk). U RLE by se zase dalo předpokládat nějaký statický escape character. V obou případech by ale taková úprava narušila univerzálnosti programu, které jsem chtěl docílit. Místo Huffmanova kódování by také šlo použít jednoprůchodové adaptivní Huffmanovo kódování. To však možná někdy příště.

Kromě efektivity programu z hlediska rychlosti je také třeba uvažovat efektivitu z hlediska komprese. Při vhodném použití program dává poměrně pěkné výsledky, které dosahují třeba i 50% komprese. Ve srovnání s komerčním softwarem, jako je WinZIP nebo WinRAR je však dosti pozadu, což je ale pochopitelné. Přestože je program vytvořen tak, aby díky zpracování souborů po bytech byl univerzální, někdy ani tento stupeň univerzálnosti nestačí. Např. při aplikaci metody RLE na bitmapy, které mají barevnou hloubku více než 8 bitů, algoritmus nezkomprimuje ani byte. Stejně tak Huffmanovo kódování nebude plně využívat svůj potenciál na textových dokumentech Unicode. Tuto

skutečnost jsem si uvědomil až v poměrně pozdním stadiu vývoje. Řešením by bylo zřejmě přidání dalšího parametru, který by programu řekl po jak velkých úsecích bytů má se soubory pracovat. Při této úpravě by se museli zřejmě úplně jinak navrhnout procedury interfacu a dále rozsahy některých datových struktur v použitých kompresních algoritmech.

Ovládání

Program neobsahuje uživatelský interface. Jediný způsob, jak jej lze ovládat, je pomocí parametrů příkazové řádky. Parametrů může být maximálně 6, z toho 4 jsou povinné a 2 volitelné. První parametr je název vstupního souboru, tedy toho, který se má zkomprimovat či dekomprimovat. Druhý parametr je název výstupního souboru, který bude programem vytvořen. Oba názvy souborů je nutné uvádět bez jakékoliv cesty. Implicitně se předpokládá, že tyto soubory budou umístěny ve stejné složce, ve které se nachází spouštěný program. Názvy souborů také v žádném případě nemohou obsahovat mezery, protože právě mezera odděluje jednotlivé parametry. Třetím parametrem je určení kompresního (-c) nebo dekompresního (-d) režimu. Čtvrtý parametr určuje jaký kompresní algoritmus se pro zvolenou činnost má použít – na výběr je Huffmanovo kódování (-h) nebo RLE (-r). Výše zmíněné parametry jsou povinné a je nutné je uvést přesně v popsaném pořadí. Pokud se v zápisu těchto 4 povinných parametrů objeví chyba nebo bude zápis neúplný, program zobrazí nápovědu nebo neudělá nic. Zbylé dva volitelné parametry mohou být zapsány v libovolném pořadí a nemusí být použity oba zároveň. Jeden z nich (-w) určuje, zda se může výstupní soubor přepsat, pokud by již náhodou existoval. Jedná se o bezpečnostní pojistku, aby nedošlo k nechtěné ztrátě dat. Druhým volitelným parametrem (-l) je aktivace logovacího souboru, který bude pod názvem log.txt vytvořen ve stejné složce jako spouštěný program. Do tohoto souboru budou průběžně ukládány informace o činnosti programu, chybách a některých datových strukturách. Asi nejhodnotnější informace, která lze z logovacího souboru při praktickém použití vyčíst je údaj o kompresním poměru, který se typicky nachází blízko konce logovacího souboru. Následuje několik názorných příkladů volání programu:

- vstup.txt bude zkomprimován Huffmanovým kódováním do souboru packed.dat
  `komprese.exe vstup.txt packed.dat –c –h`
- packed.dat bude dekomprimován Huffmanovým kódováním do souboru vystup.txt
  `komprese.exe packed.dat vystup.txt –d –h`
- picture.bmp bude zkomprimován pomocí RLE do souboru packed.dat, přitom pokud soubor packed.dat již existuje, bude přepsán, dále bude vytvořen logovací soubor
  `komprese.exe picture.bmp packed.dat –c –r –w –l`
- packed.dat bude dekomprimován pomocí RLE do souboru picture.bmp, přičemž pokud soubor picture.bmp již existuje, celý proces bude zastaven a informace o zastavení bude uložena do logovacího souboru
  `komprese.exe packed.dat picture.bmp –d –r –l`

Průběh práce

Ještě před samotným výběrem tématu zápočtového programu jsem si nastudoval potřebnou teorii ohledně Huffmanova kódování, abych věděl, že moje programátorské schopnosti budou stačit na jeho implementování. Dále jsem se rozmýšlel, zda budu programovat v Pascalu nebo v C. Protože jsme celý semestr pracovali s Pascalem, přišlo mi logické jej použít. Nakonec se však ukázalo, že program by byl v C pravděpodobně mnohem výkonnější. Dalším rozhodnutím bylo, jestli má smysl v Pascalu implementovat nějaké uživatelské rozhranní. To jsem zamítl, protože učit se v Pascalu pokročilejší práci s grafikou asi dnes nemá příliš smysl. Navíc kompresní program to ani příliš nevyžaduje a jeho ovládání pomocí parametrů příkazové řádky mi přijde celkem použitelné.

Samotný vývoj programového kódu probíhal ve třech fázích. Napřed jsem vytvořil kostru programu, která uměla zpracovat parametry, otevřít soubory a rozhodnout o další činnosti. Ve druhé fázi jsem se zamyslel, co by měl obsahovat interface pro kompresní procedury. Zprvu jsem

implementoval některé základní pro načítání a ukládání dat. Další jsem doplňoval v průběhu vývoje, když se ukázalo, že jsou potřeba. Třetí, nejdelší fáze zahrnovala implementaci kompresních algoritmů. Mezi každou fází vývoje jsem se snažil program ladit. Při finálním testování a ladění se ukázalo jako celkem komplikované zajistit správné kódování a dekódování posledního bytu souboru při použití Huffmanova kódování. Každopádně po podrobné analýze souborů v hexadecimálním editoru jsem za pomocí log souboru objevil chyby a nepřesnosti v algoritmu, takže nakonec byl soubor po dekompresi identický s původním souborem.

Závěr

Vždy mě zajímalo, jak fungují různé kompresní a šifrovací algoritmy. Šifrováním jsem se zabýval již na střední škole, a tak mi teď připadalo zajímavé zkusit implementovat právě kompresi dat, která někdy s šifrováním úzce souvisí. Obtížnost úlohy mi připadala tak akorát, vzhledem k mým schopnostem a časovému rámci, ve kterém musel být program dokončen. Z časových důvodů bych si asi netroufnul na něco komplikovanějšího.