VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

Kódování a komprese dat

Komprese textových souborů s využitím Burrows-Wheelerovy transformace

May 4, 2018

Autor: Jan Pawlus, xpawlu00@stud.fit.vutbr.cz

1 Využití knihovny a aplikace

Samostatně využitelnými funkcemi knihovny jsou:

- `int BWTEncoding(tBWTED *bwted, FILE *inputFile, FILE *outputFile)`, která zakóduje vstup ze souboru `inputFile` (případně `stdin`), uloží jej do souboru `outputFile` (případně `stdout`) a uloží informace o kódování do logovacího souboru `bwted`.
- `int BWTDecoding(tBWTED *ahed, FILE *inputFile, FILE *outputFile)`, která dekóduje vstup ze souboru `inputFile` (případně `stdin`), uloží jej do souboru `outputFile` (případně `stdout`) a uloží informace o kódování do logovacího souboru `ahed`.

Ostatní funkce knihovny není vhodné zvenku využívat, jelikož potřebují speciální vstup definovaný implementací této knihovny.

Program `bwted` lze spouštět s následujícími argumenty:

- `-i [inputFile]` definuje vstupní soubor `inputFile`. Pokud není zadán, vstup se načte z `stdin`.
- `-o [outputFile]` definuje výstupní soubor `outputFile`. Pokud není zadán, výstup se vypíše na `stdout`.
- `-l [logFile]` definuje soubor pro uložení informací o průběhu kódování či dekódování, a to velikost zakódovaného souboru a velikost dekódovaného souboru v bytech.
- `-c` říká, že má program využít knihovní funkci pro kódování.
- `-x` říká, že má program využít knihovní funkci pro dekódování.
- `-h` vypíše nápovědu a program skončí.

2 Implementace

Program byl implementován ve čtyřech krocích - jako první je na vstup aplikována *Burrows-Wheelerova transformace* (dále jen *BWT*), dále kódování *Move-To-Front* (dále jen *MTF*) a *Run-Length-Encoding* (dále jen *RLE*). Nakonec je výstup zakódován pomocí *Huffmanova kódu*.

Jelikož je časová složitost (některých) těchto algoritmů exponenciální, byl prvotní vstup rozsekán na bloky o velikosti dané hodnotou `BLOCK_SIZE`, definovanou v souboru `bwted.h`. Následně jsou na všechny tyto bloky aplikovány výše zmíněné algoritmy a výstupy se následně spojí (oddělené vhodným oddělovačem).

Jako první je nutné v každém bloku nalézt symbol nevyskytující se v textu - ten je přidán na konec každého vstupního bloku.

2.1 Burrows-Wheelerova transformace

Ze vstupního bloku o délce n je vytvořena matice $n \cdot n$ obsahující všechny možné rotace daného textu. Tato matice je následně lexikograficky seřazena a z každého řádku je zkopírován poslední znak - tím je kódování *BWT* bloku hotovo.

Díky vlastnostem algoritmu *BWT* není problém následně text dekódovat. Vstupní zakódovaná sekvence je očíslována indexy a lexikograficky seřazena - tím je dosažen zisk prvního sloupce matice. Následně je vytvořeno pole inverzních indexů, které udává pořadí zisku původního řetězce.

2.2 Move-To-Front

Vstupem do této části algoritmu je řetězec zakódovaný *BWT*. Je vytvořen nový vektor, který mapuje znaky zakódovaného textu na pozice znaku v abecedě. Po přidání každého indexu je navíc raný symbol abecedy přesunut na začátek abecedy.

Dekódování probíhá jednoduše tak, že je znova vytvořena abeceda a výstupním řetězcem je inverzní mapování do vektoru indexů.

2.3 Run-Length-Encoding

Vstupem je posloupnost symbolů z *MTF*. Algoritmus *RLE* pracuje tak, že pro každou posloupnost stejných čísel tuto posloupnost redukuje na jedno číslo, které předchází délka původní sekvence. Pro vstup o dlouhých posloupnostech může být poté úspora prostoru enormní.

Dekódování probíhá tak, že ve vstupním vektoru značí každý lichý index délku sekvence a sudý samotný symbol. Tedy pro každý symbol na sudém indexu se tento symbol zduplikuje tolikrát, kolik značí číslo na předchozím lichém indexu.

2.4 Huffmanův kód

Vstupem je posloupnost symbolů z *RLE*. Nejprve je nutné spočítat veškeré symboly ve vstupním řetězci. K vytvoření kódu je nutné symboly seřadit dle počtu jejich výskytů a sestavit binární strom, kde listy představují symboly z abecedy, a na základě rekurzivního průchodu stromem přiřadit dané kódové slovo všem listům. Následně je z tohoto stromu vytvořena struktura `map` obsahující mapování Huffmanova kódu ke každému symbolu.

Samotné zakódování řetězce probíhá takto:

- pro každý vstupní znak je nalezena jeho binární reprezentace, a tato reprezentace je ve formě datového typu `string` konkatenována
- pro každých osm textových bitů z předchozí konkatenace je za pomoci `bitset` vytvořen jeden byte ve formě `char`. Tyto byty jsou následně konkatenovány do dalšího `stringu`.
- pokud již zbývá zakódovat méně než osm textových bitů, je vytvořen poslední byte, který je zprava doplněn o nulové bity. Z tohoto důvodu je třeba uložit si původní délku vstupní posloupnosti, aby bylo možné při dekódování jednoznačně zjistit, kolik posledních bytů sloužilo pouze pro zarovnání.

Aby bylo možné v mé implementaci dekódovat cokoliv pomocí *Huffmanova kódování*, je třeba pro každý zakódovaný blok držet informaci o mapování *Huffmanových kódů* na původní znaky. Výše zmíněná mapa je tedy serializována do textu a při dekódování je znova deserializována.

Dekódování probíhá tak, že z každého bytu vstupního řetězce je vytvořen `bitset`, který je následně serializován do `stringu`. Vznikne tedy řetězec textových bitů. Následně se postupně tento řetězec prochází a pokud je nalezena shoda s některou položkou z mapy, je obnoven původní symbol.

2.5 Výstup kódování

Výstupem pro každý blok je řetězec zakódovaný pomocí *Huffmanova kódování* společně se serializovanou mapou kódů a délkou *RLE* řetězce. Jelikož pro *BWT* bylo třeba najít unikátní oddělovač, jsou tyto informace odděleny rovnou tímto oddělovačem. Jednotlivé bloky se poté od sebe oddělují vložením několika oddělovačů za sebe. Na úplný začátek výstupního souboru je navíc přidán tento oddělovač, aby bylo při dekódování jasné, jak vůbec oddělovač vypadá.

Dekódování tedy probíhá zjištěním oddělovače z první pozice, zjištění délky řetězce *RLE*, zjištění a deserializování mapy a následně postupného dekódování popsaného výše.