Insert here your thesis' task.

Czech Technical University in Prague

Faculty of Information Technology

Department of theoretical computer science

Master's thesis

Admission procedure Automatic processing of applications for master's study program

Bc. Ján Ondrušek

Supervisor: Ing. Tomáš Kadlec

4th June 2012

Acknowledgements

I would like to thank my family and friends for support during writing this thesis.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act 60 no. 121/2000 (copyright law), and with the rights connected with the copyright act included the changes in the act.

In Prague 4th June 2012 ....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Ján Ondrušek. *Admission procedure
Automatic processing of applications
for master's study program: Master's thesis.* Czech Republic: Czech Technical University in Prague, Faculty of Information Technology, 2012.

Abstract

Primary aim of this thesis is to analyse Conditions for admission and Dean's directive for admission process to master's study programme at CTU FIT. Implement RESTful API, which exposes backend functionality for admission processing using Business Process Management.

Keywords Admission procedure, RESTful API, BPM, jBPM, Spring, Spring Roo

Abstrakt

Primárnym cieľom tejto diplomovej práce je analyzovať Řád přijímacího řízení ČVUT a Směrnici děkana pro přijímací řízení na ČVUT Fakultě informačních technologií. Implementovať RESTful API, ktoré vystaví funkcionalitu backendu pre prijímací proces s použitím Business Process Management stroja.

Klíčová slova Spracovanie prihlášok, RESTful API, BPM, jBPM, Spring, Spring Roo

Contents

List of Figures

List of Tables

Prologue

Motivation and objectives

Every year hundreds of students apply for studies at Czech Technical University, Faculty of informatics. This raises certain requirements including managing, storing, analysing and processing of all these applications. Each application has its own life cycle, which begins with filling out an on-line form and continues through various steps which an applicant has to pass. The life cycle ends when a decision of acceptance is delivered to the applicant and he either enrolls in the studies or not.

Currently all applications are processed rather manually. Many man days of administrative work are consumed during the process. Although an electronic form is filled in and submitted by an applicant, the rest of actions almost exclusively fall into the hands of Study Department staff. The question is: Why?

We live in the world of new era of the Internet. Everything goes on-line, web and the latest trend - everything goes mobile. People want things to happen very quickly. They want to access all the information fast, now.

Students and applicants are no different. They expect from this prestigious University, especially from Faculty of informatics, most modern and useful gadgets when it comes to software and web.

Let's make things better

Taking the above written into account, this might be a good idea for a master's or bachelor's thesis. However if we want to use all available technologies that have become popular in past years and to automatize majority of admission processing, it becomes a very complex project. So why not to build several teams and split it into multiple, both bachelor's and master's, thesis?

This is how project Přiríz was born. It includes web interface for both students and Study Department staff, native Android application and RESTful API with BPM processing machine, which is a subject of my master's thesis.

CHAPTER 1

Data compression

This thesis was submitted at Czech Technical University in Prague (see Figure 1.1).

1.1 Notions and definitions

The source *message* consists of *source units*, which can be defined as *alphabet symbols* or sequence of alphabet symbols *(word, string, phrase)*, where alphabet S is a finite and non-empty set of symbols. The *code unit* is defined as a sequence of bits. The empty sequence of symbols is called *empty string* and it is represented by ε. The set of all symbols from alphabet S, free of empty string, is represented by S^+. The *concatenation* of two phrases $x, y \in S$ is represented by $x.y$.

Code is a triple $K = (S, C, f)$, where

- S is a finite set of source units,
- C is a finite set of codewords (code units),
- f is an injective mapping from S to C^+.

Figure 1.1: CTU logo

The mapping f does not map two different source units from S to the same codeword from C, as shown Formula 1.1.

$$\forall a_1, a_2 \in S, a_1 \neq a_2 \Rightarrow f(a_1) \neq f(a_2) \tag{1.1}$$

The string $x \in C^+$ is *uniquely decodable* with f, when Formula 1.2 is true.

$$\forall y_1, y_2 \in S^+, f(y_1) = f(y_2) = x \Rightarrow y_1 = y_2 \tag{1.2}$$

The code $K = (S, C, f)$ is *uniquely decodable*, when all strings $x \in C^+$ are uniquely decodable in f. The code K is called *prefix code*, when none of codewords is a prefix of another codeword. If all codewords are exactly n symbols length in code K, the code K is called *block code*. The prefix codes and block codes are often used by compression algorithms because of their unique decode-ability during the left-to-right reading (decoding).

$$\textit{Compression ratio} = \frac{\textit{Length of compressed data}}{\textit{Length of original data}} \tag{1.3}$$

The compression efficiency can be expressed by many units of measure. The amount of data reduction gained by the compression process is *compression ratio*. This compression ratio is a ratio of the length of compressed data to the original size of data (Formula 1.3). For example, the compression ratio is measured in *bpb* (bits per bit), *bpc* (bits per character) or *bpp* (bits per pixel).

The compression algorithms use specific *compression models* to encode the data. For example, these models could follows:

- The algorithm assigns code to each source unit irrespective to its position (statistical compression methods).
- The Markov's model of n-th order look at previous n source units to assign code. The simplist of this codes, 0th order, are mentioned above.
- The models based on finite automata.

1.1.1 Entropy

The entropy is only theoretical minimal length but it is possible to reach this border in some special cases. It is very difficult to measure real entropy of source message in common usage, because not only statistical model of 0th order (context of source units with length 1) exists. For example, the probabilities of appearances of source units pairs (context of source units with length 2) are considered in 1st order statistical model.

Table 1.1: RLE example

Method	Data to encode	Encoded data
RLE 1	*babaaaaabbbaabbbba11aaa*	1*b*1*a*1*b*5*a* 3*b*2*a*4*b*1*a* 2*1*3*a*
RLE 2	*babaaaaabbbaabbbba11aaa*	*bab*@5*a*@3 *baa*@4*ba1 1*@3*a*

1.1.2 Classification

Data compression/decompression is classified by many factors. The first classification depends on information loss during the compression process. The data compression, as data compression algorithms, is divided into two main parts:

- *Lossy*—some information loss is possible. These compression methods achieve higher compression (better compression ratio) but they are useful only in special cases (images, video, voice...).
- *Lossless*—information is acquired in original form. These compression methods are best suited for data where loss is unacceptable (documents, programs, scripts...).

1.2 Elementary methods

Many elementary compression methods are currently known, but only the Run-Length Encoding (RLE) is mentioned on as very simple compression method, to show how compression methods work.

1.2.1 RLE

The RLE technique was created especially for data with strings of repeated symbols (the length of string of repeated symbols is called *run*). The main idea of RLE compression is to encode repeated symbols as a pair—the length of string and the symbol.

There are shown some ways of RLE compression of 23-characters-long string *"babaaaaabbbaabbbba11aaa"* in Table 1.1. The first method (RLE 1) is the simplest way to encode string (20 characters) which exactly follows idea of RLE. The problem is that in worst case the size of output data can be two times longer than size of input data. This problem is solved by second method (20 characters), for simplicity called RLE 2, where only three or more characters long strings are encoded, but we pay for it by another character (@ in example), which precede all of encoded pairs, to differentiate between the length of string and the number as a character. The third method (RLE 3) solves this problem by map of bits (18 characters). RLE compression method is very useful for graphic files coding but it is practically useless in text compressions without using it cooperatively with other methods.

1.3 Dictionary methods

1.3.1 LZ77

It is obvious that the value of offset and the match length have to be limited to some constant. The usually chosen value for the match length is 255 (8 bits) and the offset is commonly encoded on 12–16 bits, so the search buffer is limited to 4 095–65 535. In so far that there is no need to remember more than 65 535 already encoded symbols during compression process.

CHAPTER 2

Implementation and testing

2.1 Details of realised tests

The scaliness of implemented algorithms were tested on chosen files from Calgary and Canterbury corpus too. The framework to scale the files is different of the testing one. Each file is equally split to 1 000 parts (files with number of lines less than 1 000 are split to 100 parts) by number of lines—the n th part consists of $\frac{n}{1000}$ ($\frac{n}{100}$) lines. Each test runs only 100–10 times (depending up the n—the size of compressed data) because of time complexity. This cycle runs 10–100 times (10 was chosen for this tests) and the minimum time is taken. The file splitting by the number of lines was chosen because of the character of algorithms (word-based)—the splitting by the block of the same size is not so predicative.

There are parameters of the computer used for tests shown in Table 2.1.

2.2 Integers distribution and encoding

The integers encoding of indexes of phrases from the word (non-word) dictionary is possible cause of the only average results of compression ratio of implemented algorithms. The decision is to get the distribution of indexes during the encoding process (and decoding process too). The length of indexes

Table 2.1: Testing computer parameters

Part	**Description**
CPU	2.2 GHz AMD Athlon(tm) 64 Processor 3200+
MEM	2.5 GB
OS	x86_64 GNU/Linux Fedora release 7 (Moonshine)

located in shown graphs is only hypothetical—the binary code with minimal length.

The graphs of index distribution also shows the differencies between the algorithms with sorted dictionaries (*WLZWS* and *WLZWES*) and the algorithms with unsorted dictionaries (*WLZW* and *WLZWE*). The most frequently used phrases are moved to the front of dictionary in algorithms with sorted dictionary so they get lower indexes. This feature is demonstrated by the growth of number of indexes at the beginning of the distribution. The compression process of algorithms with sorted dictionaries becomes more efficient when the code with variable length of code words (Fibonacci code) is used but the compression efficiency is supposed to be the same at the transition from the *WLZWE2* algorithm to the *WLZWES2* algorithm—the encoding by block code.

Conclusion

The word-based dictionary data compression algorithms (a part of lossless data compression) are the subject of this thesis. The lossless data compression is a very important field of research because the data compression allows to reduce the amount of space needed to store data.

The background of a data compression field was presented in Chapter 1. There are basic notions and definitions followed by the description of character-based dictionary algorithms. The word-based dictionary compression methods were investigated and discussed at the end of this chapter too.

There is the investigation of index distribution of tested files in Section 2.2. It led to the new modification of semi-adaptive word-based Lempel Ziv Welch (LZW) algorithm—*WLZWE2*. The compression efficiency of this algorithm applied to the large files is better than the other implemented algorithms. However, the compression efficiency of *WLZWE2* algorithm is much worse when it is applied to the small files. The experiments with *WLZWE2* and *WLZWES2* algorithms confirm the assumption from Section 2.2—the compression efficiency of version with unsorted dictionaries (*WLZWE2*) is analogous to version with sorted ones (*WLZWES2*).

The testing of memory used during compression and/or decompression process is one of the possibilities of further research. The experiments with files of greater size or multilingual files could be also good opportunity to gain new improvements of algorithms. The static part of dictionaries could improve the compression efficiency too.

The implemented methods achieve fairly good compression ratio (25–30% at large files) with acceptable compression and decompression time. There are possibilities of further improvements especially at semi-adaptive methods. However, the gain of these improvements is not good enough to top the compression efficiency of other lossless data compression methods (context methods from PPM family). The results of implemented algorithms were not as good as it was expected but the work on this thesis showed new ways of possible further research—word-based version of grammar-based compression algorithms and another possibilities in the field of word-based context methods of data compression.

The Gnuplot 4.2 utility was very useful for generation of graphs in this thesis. There was the drawing editor Ipe 6.0 used for figures creation.

APPENDIX A

Content of CD

readme.txt	- the file with CD content description
data/	- the data files directory
graphs/	- the directory of graphs of experiments
*.eps	- the B/W graphs in PS format
*.png	- the color graphs in PNG format
*.dat	- the graphs data files
exe/	- the directory with executable WBDCM program
wbdcm	- the WBDCM program executable (UNIX)
wbdcm.exe	- the WBDCM program executable (MS Windows)
src/	- the directory of source codes
wbdcm/	- the directory of WBDCM program
Makefile	- the makefile of WBDCM program (UNIX)
thesis/	- the directory of LaTeX source codes of the thesis
figures/	- the thesis figures directory
*.eps	- the figures in PS format
*.pdf	- the figures in PDF format
*.tex	- the LaTeX source code files of the thesis
text/	- the thesis directory
thesis.pdf	- the Diploma thesis in PDF format
thesis.ps	- the Diploma thesis in PS format