XVII Conference on Applied Mathematics
Đ. Herceg, H. Zarin, eds.
Department of Mathematics and Informatics
Novi Sad, 2007, pp. 193–204

ADAPTATION AND APPLICATION OF DAITCH–MOKOTOFF SOUNDEX ALGORITHM ON SERBIAN NAMES

Petar Rajković[1], Dragan Janković[1]

Abstract. The Daitch-Mokotoff SoundEx algorithm is one of the most effective phonetic-based code resulting string matching algorithms. The input to this algorithm is a word and output is a code or list of codes. It is a phonetic algorithm primarily developed for English names comparison, but it is upgraded and can be used for German and Slavic names too. We have examined the possibilities of using this algorithm for Serbian names. Also, we try to upgrade algorithm rules according lexical standards of the Serbian language. On the other side, one new string comparing strategy is introduced – calculating similarity for generated Daitch-Mokotoff codes using Levenstein and Jaro-Winkler metrics. All mentioned topics are covered with experimental results.

1. Introduction

Government and commercial organizations are increasingly required to store, maintain, search and match identity data from many nations and numerous languages. A variety of algorithms have been published to allow approximate verbal matches to be found in documents or databases. In each case, the user specifies a word and the system retrieves records containing similar ones.

The reasons for wrongly typed words in some text in general are typing errors and spelling errors. The most frequent typing errors are: deleted letter, inserted letter, replaced two letters, added letters, removed letters, used abbreviations, split words, joint words, etc. For different kinds of typing errors different kinds of algorithms have been developed. Some of them detect a set of the above errors.

Two main classes of algorithms can be distinguished: those that determine word similarity by examining the order of the letters, and those that rely principally on phonetics. The second class of the algorithms depends strongly of the given language. Also, there are combinations of the above two approaches (for example editex algorithm [1]).

The majority of the phonetic algorithms are constructed and adapted for English names. So it is interesting to investigate the applicability of these algorithms on Serbian language, and especially on names.

[1]Faculty of Electronic Engineering, University of Niš, A. Medvedeva 14, 18000 Niš, Serbia, e–mail: {rajkovicp, gaga}@elfak.ni.ac.yu.

So we developed a program for the most known string matching algorithms from both kind phonetic and distance matching (Jaro – Winker, Levenstein, NYSIIS, Metaphone, Double metaphone, different SoundEx algorithms, etc) and apply them on Serbian names. Results are reported.

Our future work will be concerned with upgrading of presented algorithms in order to become more suitable for the Serbian language, as well as, enlarge our test base.

2. Classes of String Comparing Algorithms

For implementing our string matching application we used two different kinds of algorithms – similarity-based and code-resulting algorithms. From the group of similarity-based algorithms we have tested Jaro–Winkler and Levenstein, and from the group of code-resulting methods we have used NYSIIS, Metaphone, and different implementation of SoundEx algorithms (Daitch Mokotoff, and four standard modifications – Miracode, Simplified, SQLServer, and Knuth Ed2).

The Jaro–Winkler algorithm is a kind of measure of the similarity between two strings. The Jaro measure [2] is the weighted sum of percentage of matched characters from each file and transposed characters. Winkler increased this measure for matching initial characters, and then rescaled it by a piecewise function, whose intervals and weights depend on the type of string (first name, last name, street, etc.). This is an extension of the Jaro distance metric, from the work of Winkler in 1991 to 1999 [3].

The Levenshtein algorithm is based on calculating distance that is obtained by finding the simplest way to transform one string to another [4]. Transformations are the one-step operations of (single-phone) insertion, deletion and substitution. In the simplest versions substitutions cost two units except when the source and target are identical, in which case the cost is zero. Insertions and deletions cost half that of substitutions. On the basis of these values similarity is computed according to the length of the source string.

NYSIIS is a member of the group of phonetic coding algorithms. Basically, it has been used to convert a name to a phonetic coding of up to six characters [5]. Now, NYSIIS codes can be larger than six characters. NYSIIS is the short form of the *New York State Identification and Intelligence System* Phonetic Code. It features an accuracy increase of 2.7% over the traditional SoundEx algorithm. It is a pretty simple algorithm described in *Name Search Techniques*, New York State Identification and Intelligence System Special Report No. 1, by Robert L. Taft, and it has some seven steps that convert a word to a string that represents its code.

Metaphone (we use its *double metaphone* variant) is an algorithm to code English words (and foreign words often heard in the United States) phonetically by reducing them to 12 consonant sounds [6]. This reduces matching problems from wrong spelling in English language.

Soundex is a phonetic algorithm for indexing names by their sound when pronounced in English [7]. The basic aim is to encode names with the same pronunciation to the same string, so that matching can occur despite minor

differences in spelling. Soundex is the most widely known of all phonetic algorithms and is often used (incorrectly) as a synonym for "phonetic algorithm". Soundex was developed by Robert Russell and Margaret Odell and patented in 1918 and 1922. A variation called American Soundex (U.S. SoundEx) was used in the 1930s for a retrospective analysis of the US censuses from 1890 through 1920. The Soundex code for a name consists of a letter followed by three numbers: the letter is the first letter of the name, and the numbers encode the remaining consonants. Similar sounding consonants share the same number so, for example, the labial B, F, P and V are all encoded as 1. Vowels can affect the coding, but are never coded directly unless they appear at the start of the name.

3. Daitch–Mokotoff SoundEx Algorithm–Application and Adaptation

The one of the latest significant improvements of the basic SoundEx is the Daitch-Mokotoff algorithm. In 1985, this author indexed the names of some 28,000 persons who legally changed their names while living in Palestine from 1921 to 1948, most of whom were Jews with Germanic or Slavic surnames. It was obvious that there were numerous spelling variants of the same basic surname and the list should be "soundexed". It is a modification to U.S. SoundEx.

Names are coded using Daitch–Mokotoff SoundEx (DMS coding) with an array of digits [8]. The length of coding array is in the most cases six, but it can vary from four up to ten. Each digit in coding array has a purpose for representing some sound listed in coding charts (Table 1). When a name lacks enough sounds for coding in six digits, the coding algorithm uses zeros to fill to six digits. The word GOLDEN has only four sounds that can be coded (G – L – D – N). The result of this coding is 5836. In the system that is based on six coding digits this result will be enlarged to 583600, and for ten digits based coding the result would be 5836000000. The examples of DMS coding which will be mentioned in the further text, are coded in the six-digit base.

The vowels (A, E, I, O, U, Y) are not coded in this system, except at the beginning of the name as in ALPERT. In these cases the vowels are coded with digit 0 (ALPERT is 087930). The other case when the vowels are not ignored is the existence of a pair of vowels before another vowel, as in BREUER (791900), but not in the word FREUD.

The letter H is coded at the beginning of the word or preceding a vowel, otherwise it is not coded. By example, the word HABER is coded with 579000 and MANHEIM is 665600.

The main quality of this algorithm is the fact that it calculates the so-called adjacent sounds. Adjacent sounds are those which can be combined in order to create larger sound (sounds T and Z, one before another, are not treated like two, but like one sound – "C" in Serbian language). Word MINTZ are not coded as M – N – T – Z , but M – N – TZ (664000). For dealing with adjacent letters which produce one sound there is one more rule. If the adjacent

Table 1: Coding table of Daitch–Mokotoff SoundEx (Meaning of column headers: Lt - letter, St - start of a name, Bf - before of a vowel, AOS - any other situation)

Lt	Alt. spell	St	Bf	AOS	Lt	Alt. spell	St	Bf	AOS
NC = not coded					**NC = not coded**				
AI	AJ, AY	0	1	NC	**P**	PF, PH	7	7	7
AU		0	7	NC	**Q**		5	5	5
A		0	NC	NC	**RZ, RS**	Try RTZ (94) and ZH (4)			
B		7	7	7	**R**		9	9	9
CHS		5	54	54	**SCHTSCH**	SCHTSH, SCHTCH	2	4	4
CH	Try KH (5) and TCH (4)				**SCH**		4	4	4
CK	Try K (5) and TSK (45)				**SHTCH**	SHCH, SHTSH	2	4	4
CZ	CS, CSZ, CZS	4	4	4	**SHT**	SCHT, SCHD	2	43	43
C	Try K (5) and TZ (4)				**SH**		4	4	4
DRZ	DRS	4	4	4	**STCH**	STSCH, SC	2	4	4
DS	DSH, DSZ	4	4	4	**STRZ**	STRS, STSH	2	4	4
DZ	DZH, DZS	4	4	4	**ST**		2	43	43
D	DT	3	3	3	**SZCZ**	SZCS	2	4	4
EI	EJ, EY	0	1	NC	**SZT**	SHD, SZD, SD	2	43	43
EU		1	1	NC	**SZ**		4	4	4
E		0	NC	NC	**S**		4	4	4
FB		7	7	7	**TCH**	TTCH, TTSCH	4	4	4
F		7	7	7	**TH**		3	3	3
G		5	5	5	**TRZ**	TRS	4	4	4
H		5	5	NC	**TSCH**	TSH	4	4	4
IA	IE, IO, IU	1	NC	NC	**TS**	TTS, TTSZ, TC	4	4	4
I		0	NC	NC	**TZ**	TTZ, TZS, TSZ	4	4	4
J	Try Y (1) and DZH (4)				**T**		3	3	3
KS		5	54	54	**UI**	UJ, UY	0	1	NC
KH		5	5	5	**U**	UE	0	NC	NC
K		5	5	5	**V**		7	7	7
L		8	8	8	**W**		7	7	7
MN			66	66	**X**		5	54	54
M		6	6	6	**Y**		1	NC	NC
NM			66	66	**ZDZ**	ZDZH, ZHDZH	2	4	4
N		6	6	6	**ZD**	ZHD	2	43	43
OI	OJ, OY	0	1	NC	**ZH**	ZS, ZSCH, ZSH	4	4	4
O		0	NC	NC	**Z**		4	4	4

letters have the same code number they should be coded as one sound. Without this rule, the word TOPF would be coded 377000, but, with this rule, the code for word TOPF is 370000. Exceptions to this rule are the letter combinations MN and NM whose letters are coded separately, as in KLEINMAN, which is coded 586660 not 586600.

When a surname that comes from Jewish culture consists of more than one word, it is treated as one word (surname Ben Aaron should be treated as BENARRON). For Anglo–Saxon, Slavic, and German names this rule should be ignored, otherwise it produces irrelevant results.

Also, several letter combinations cause the problem of multiple codes producing. In every other coding system the rule one word – one code is certain. But here, the letter combinations CH, CK, C, J, and RS are assigned two possible code numbers. The DMS coding of the Polish name ROSOCHOWACIEC results in two code words 944744 and 945744. One harder case is coding of the name CHICAGO. It results in the array of four codes 445000, 545000, 455000, and 555000.

The rules that came from Slavic languages which are incorporated in the basic DMS coding algorithm are mainly based on the phonetic rules of Polish and Russian languages. So, they cannot be used in the most appropriate way for every Serbian name. For example, the word GOLUBICA (Serbian variant of this word which means small dove) is last name that one can find in Poland, Slovakia, Czech Republic and Serbia. In each of mentioned countries this word is pronounced the same way, but it is written very differently – GOLUBICA, GOLUBITSA, HOLUBICA, but results of DMS coding, in every case, are 5874 and 5875. But in many cases, Serbian names, interpreted on different ways could not be marked as equal. For example, last name ĐANKOVIĆ would be interpreted from some person that is not of Slavic origin probably as DANKOVITS (365740). Otherwise, Slavic people would probably transform it into DJANKOVITSI (316574, 346574).

The main problem here is the existence of characters (letters) that are specific of the Serbian language. The letters Š, Đ, Č, Ć, and Ž, found in Serbian and other languages from South Slavic group, are not present in almost any other language. So, first adaptation of the Daitch–Mokotoff algorithm that we made was the transformation of Serbian letters in order to present Serbian names in a proper way (Table 2).

Table 2: Serbian Letters Transformation

letter	**conversion (Variant 1)**	**conversion (Variant 2)**
š	sh	sh
đ	dy	gi
č	tch	ch
ć	ty	ci
ž	zh	zy

Transformation of Serbian names is not incorporated in the algorithm, but it is implemented as a preprocessing step. We try to implement two different sets of rules for Serbian letters transformation. Variant 1 is based upon the presentation of similar sounds in English and German languages, and Variant 2 is based on Romanic representation for sounds represented by Serbian letters

in Table 2. The only problem was the representation of letter *ć* whose sound is not present in the most of languages. It has been replaced by a letter group *ty* according to Serbian phonetics. Variant 1 gave us more appropriate results, and so we chose to implement this kind of transformation.

Transformation of specific letters is not the only problem when one wants to convert some Serbian word into a representation that is most adequate for DMS coding. The bigger problem is consonant transformation avoiding. Serbian language has a few rules for transformation of consonant depending of its position in a word, and depending of its preceding letter in the word, for example, adjective "Serbian", would be written as "SRPSKI" but not "SRBSKI". Also the term "Absolute", translated into Serbian would be "APSOLUTAN", instead of "ABSOLUTAN". Analysis of this problem is very complex, because there are also irregular words where this consonant transformation rules are not applied. Further discussion of this problem will not be given here, because of its complexity and large scale.

Table 3 represents a set of twelve different consonant transformations that should be always avoided in Serbian words in order to have DMS coding produced relevant results. This table is product of a heuristic and statistical analysis of a source of several thousands Serbian names.

Table 3: Avoiding Consonant Transformation

source	**target**	**source**	**target**
tk	dk	zlj	slj
st	zt	st	zt
tp	dp	šč	zč
ks	gs	jj	j
bdž	pdž	dd	d
ps	bs	šć	sć

4. Testing

In order to test all the previously described algorithms we have developed a simple Windows-based application in Visual Studio 2003, using C#.Net named Word matcher. We have implemented the Jaro–Winkler and Levenstein similarity algorithms, as well as the following code-resulting methods: NYSIIS, Metaphone (as Double Metaphone), Caverphone, Daitch-Mokotoff SoundEx (in order to speed up this complex algorithm hashing of tested words is enabled), and four variants of the standard SounEx algorithms (Knuth Ed2, Simplified, Miracode, and SQLServer SoundEx).

This application provides us the possibility to test two words or two sentences (Figure 1), or one word (or sentence) with strings from some source file. Source files only have to be placed in the same folder with the executable file, and they will be loaded. The comparison results can be saved in the text file and processed later on.

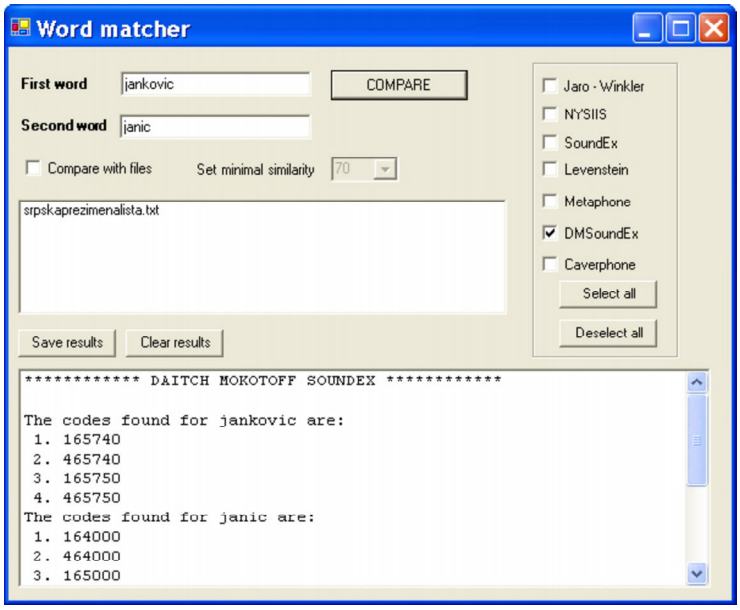

Figure 1: Comparing two words by selected method.

As an example of the usability of previously described algorithms we will present results of testing similarity of the last name *Jankovic*. All testing results are presented by tables that are consisted of three columns – the count of found similar words, minimal similarity (in percents), and duration of this operation in seconds. All tests are done on a computer with Pentium Mobile processor at 1.8 GHz with 512MB of RAM.

In the text to follow, different comparing strategies are tested and some of testing results are presented. Generally, there are four strategies for testing similarity between two strings:

- Exact matching – comparing two strings in order to determine if they are equal.
- Using similarity method – comparing two strings using some algorithm that will return us some value (between 0 and 1) that will be information about similarity level.
- Using code resulting method – comparing codes that are generated by using some code resulting algorithm, in order to determine if they are equal.
- Using similarity for generated codes (code + similarity) – apply similarity method to determine level of similarity of passed strings' phonetic codes.

The source for this testing is a text file that contains list of 22505 different words. The list members are first names, last names, names of settlements

and other commonly used words that are collected by different organization in Serbia and Montenegro. Some of them are written using Serbian alphabet specific characters (š, č, ć, đ, ž) and some of them are written using English alphabet. The large number of the collected lat names is presented in both ways (e.g. you can find both *Rajkovic* and *Rajković* in the list).

The results of comparison of word *jankovic* with the representative list of names using different code resulting methods is shown in Table 4. We can assume that DMS coding is slowest of all four algorithms, but let us see the relevance of obtained results.

Table 4: The results of comparison of word *jankovic* with representative list of names using different code resulting methods

Used algorithm	Number of matching words	Duration (seconds)
NYSIIS	1	0.625
Metaphone	17	0.1875
Daitch Mokotof SoundEx	4	49.421875
Daitch Mokotof SoundEx (with Hash table)	**4**	**0.875**
Knuth Ed2 SoundEx	19	0.15625

- The NYSIIS code returns only one word – janković. Thus, using pure NYSIIS is like using standard exact string matching. However, in this case, the strategy code + similarity should give much better results.
- The Metaphone returns a list of seventeen words (jankov, jankovic, janković, anković, jankovića, jankovići, junkovića, junković, inković, inkovići, janaković, jankovi, junaković, onković, jankovići, unkašević, unković), but some of them cannot be used as relevant similar words in the Serbian language.
- Like Metaphone, standard SoundEx gives similar results. It returns even more words (19: jankov, jankovic, janković, johanes, jankovica, janjević, jankovica, junkovica, janjušević, junković, janjevica, janjuševi, janaković, janićijević, jankovi, junaković, junuzovci, jankovići, janjević), but not all of them can be assigned as relevant.
- DMSoundEx detects 4 very relevant words: jankov, jankovic, janković, jankovci.

In the case of modified DMS coding for Serbian names the application of the code + similarity strategy did not bring any new quality, because the count of returned results grow too fast with decreasing of minimal similarity level (Table 5 and Table 6). The conclusion could be that the DMS coding algorithm, together with proposed modifications, is designed to return highly sophisticated

values (codes) for each entered word. So, further transformation of these codes seems obsolete.

Using of similarity resulting methods can be interesting, too. When comparing one word (test word) with a list of words we will obtain as a result a list of the words with similarities corresponding to the test word. In the case when comparing list has 2000 words, the list of results will have 2000 results. In order to reduce the size of resulting list we should determine some kind of threshold for placing certain word to the list of the results. And this threshold is the level of minimal similarity.

Table 5: The results of comparison of the word *jankovic* with a representative list of names using combination of *modified DMSoundEx* code resulting method and *Jaro–Winkler* similarity method

Found similar words (count)	Minimal similarity level (percent)	Duration (seconds)	Duration (seconds) with Hash-Table
4340	70	52.125	0.93275
2722	75	50.031	0.93275
1665	80	51.203	0.93275
178	85	50	0.875
94	90	52.172	0.875
5	95	50.141	0.875
4	99	50.75	0.875

Table 6: The results of comparison of the word *jankovic* with a representative list of names using combination of *modified DMSoundEx* code resulting method and *Levenstein* similarity method

Found similar words (count)	Minimal similarity level (percent)	Duration (seconds)	Duration (seconds) with Hash-Table
46	70	49.7969	1.0275
46	75	50.7031	1.0275
46	80	49.2344	0.875
4	85	51.4531	0.875
4	90	50.625	0.825
4	95	51.9844	0.825
4	99	49.8438	0.825

If we want to use matching strings by similarity for some spelling helper we need a result list with not more than 10 to 12 words. If we want to solve this kind of problem we have to use matching string by similarity, otherwise exact matching will provide only one solution – the searched word. The results of using Jaro–Winkler and Levenstein similarity methods are presented in the following two tables.

Table 7: The results of comparison of the word *jankovic* with a representative list of names using the *Jaro–Winkler* similarity method

Found similar words (count)	Minimal similarity level (percent)	Duration (seconds)
1270	70	0.375
416	75	0.21875
164	80	0.203125
36	85	0.1875
13	90	0.171875
6	95	0.171875
1	99	0.171875

Table 8: The results of comparison of the word *jankovic* with a representative list of names using the *Levenstein* similarity method

Found similar words (count)	Minimal similarity level (percent)	Duration (seconds)
24	70	0.171875
7	75	0.15625
3	80	0.15625
3	85	0.21875
1	90	0.15625
1	95	0.15625
1	99	0.15625

As has been discussed in the previous part of this paper, a critical issue in these problems is setting of minimal similarity in order to obtain a list of synonyms that has a reasonable number of members (less than 10). The Jaro–Winkler algorithm returns the following words when minimal similarity is set at 95%: *jankov* 0.967, *jankovic* 1, *janković* 0.967, *jankovica* 0.954, *jankovići* 0.954, and *jankovi* 0.983. Each word is followed by its similarity level with word *jankovic*. Comparing word *rajkovic* with the mentioned list of names will return six words for minimal similarity of 95% (*rajkov* 0.967, *rajkovica* 0.954, *rajkovići* 0.954, *rajkovac* 0.967, *rajković* 0.967, and *rajkovci* 0.983) and 14 words for 90%. On the basis of comparing many other Serbian last names we could say that a "reasonable" threshold for minimal similarity level for the Jaro–Winkler method should be set between 90 and 95%. For example, the level of 92% will return a list of 10 words similar with word *rajkovic* and 8 words similar with word *jankovic*.

In the case of the Levenstein algorithm minimal similarity level could be set lower – 70 to 80 percent. For the word *jankovic* and a threshold of 75% our testbench application returns seven words: *jankovic* 1, *stankovic* 0.778, *janković* 0.875, *jankovica* 0.778, *jankovići* 0.778, *janaković* 0.778, *jankovi* 0.875. For a

similarity of 70%, the number of similar words is 24. Testing the Levenstein algorithm on word rajkovic returns 3 words (*rajkovac* 0.875, *rajković* 0.875, and *rajkovic* 1) for 80% threshold, 10 words (brajković 0.778, brajkovac 0.778, raškovice 0.778, rašković 0.778, rajkovic 1, ranjković 0.778, rajkovac 0.875, rajković 0.875, rajčković 0.778, trajković 0,778) for 75% and 35 words for 70%.

5. Conclusion

This paper gives an overview of the well known string matching algorithms (that are generally divided in two groups – similarity and code resulting) and, at the same time, explores their possible application for Serbian and other Slavic names. Different comparing strategies are tested: comparing names by similarity (using the Jaro–Winker or Levenstein algorithms), comparing names by their phonetic codes directly (NYSIIS, Metaphone, different SoundEx codes, and Daitch–Mokotoff SounEx) or calculating similarity between codes in order to enlarge the set of results.

It can be concluded that similarity-based methods are, generally, faster, but the minimal similarity level has to be precisely adjusted. Code-resulting methods are slower, but they work without any additional adjustment. Using code + similarity strategy does not bring relevant improvements in the case of the Daitch–Mokotoff SoundEx. From this class of algorithms *Standard SoundEx* is fastest, but *DMSoundEx* gives the most appropriate results for Serbian names comparison.

In the further work, we will try to upgrade the presented algorithms in order to make them more suitable for the Serbian language, as well as, enlarge our test base.

References

[1] J. Zobel, Ph. Dart. Phonetic String Matching: Lesson from Information retrieval. In Proc. 19th Inter. Conf. on Research and Development in Information Retrieval (SIGIR'96), pp. 166–172, Aug. 1996.

[2] M. A. Jaro. Advances in Record-linkage Methodology a Applied to Matching the 1985 Census of Tampa, Florida. Journal of the American Statistical Association, 89: 414–420.

[3] W. E. Winkler, Y. Thibaudeau. An Application of the Fellegi-Sunter Model of Record Linkage to the 1990 U.S. Decennial Census. Statistical Research Report Series RR91/09, U.S. Bureau of the Census, Washington, D.C., 1991.

[4] P. Kleiweg. Implementation and Visualization of Levenstein Algorithm. *http://www.let.rug.nl/~kleiweg/lev/levenshtein.html*

[5] P. E. Black. NYSIIS, from *Dictionary of Algorithms and Data Structures*, Paul E. Black, ed., *NIST*, *http://www.nist.gov/dads/HTML/nysiis.html*

[6] Lawrence Philips Metaphone algorithm. *http://aspell.net/metaphone/*

[7] How To: Understanding Classic SoundEx Algorithms. *http://www.creativyst.com /Doc/Articles/SoundEx1/SoundEx1.htm*

[8] Definition of DMSoundEx. *http://www.jewishgen.org/InfoFiles/soundex.html*