Computer Science Journal of Moldova, vol.**1**, no.1(1), 1993

Detecting and correcting spelling errors for the Roumanian language

S.Cojocaru M.Evstunin V.Ufnarovski

Abstract

The implementation of the Roumanian Spelling Checker is discussed. The structure of the morphological vocabulary and similarity word recognition are considered more detailed.

1 Introduction

To create a Spelling Checker for the Roumanian language is of interest from different points of view. Being among highly inflexional languages, it makes the problem of compact representation of its vocabulary nontrivial. For example even for a 30 thousand word dictionary it is necessary to store more than 300 thousand different word-forms (to differentiate them we refer to the latter as vocabulary, saving the word "dictionary" for base word-forms only.) Taking into account the fact that most computers used today in Moldova have no more than 1M of RAM, it may be concluded that the whole this vocabulary (or at least its main part) should be stored on the disk. This, of course, create a new problem of a fast access to the stored word-forms.

Naturally, not only detecting spelling errors, but also the suggesting correct words is a very topical problem. It would be nice to correct more than one mistake in a word, so the suggestion becomes a nontrivial problem too.

The structure of a compact vocabulary for the highly inflexional languages was always of interest. We face more problems when there is some additional information, such as morphology, semantics etc. But this information becomes very important when errors are detected not

only for a single word but also for pairs of words (in disagreement of genders or cases, for example).

Here we want to consider one possible approach, where a vocabulary is provided with morphological features.

To formalize the problem we introduce the notion of binary decomposition of a vocabulary. Then we deal with the task of creating a vocabulary. The formalism of special grammars will be proposed for this aim. Dynamic methods of word-forms decomposition are the subject of the next part.

The related problem of similar words, playing important role in the suggestion, will be our next subject.

The approach described here was applied for the Roumanian Spelling Checker ROMSP, and some implementation details such as increasing efficiency, shorting response time, internal data representation and architecture are discussed.

2 The vocabulary decomposition

First of all the problem of compact representation of the set of word-forms for the given word should be solved. One of the well-known approaches here is to separate roots and endings.

Let us suppose that we have a vocabulary V, containing all possible word-forms (forgetting about the sources of obtaining them. Moreover, the words may be considered in the mathematical sense — as a sequences of letters, independent of their existence in the real language).

Definition 1 *The binary decomposition of vocabulary V consists of two sets of words (namely, roots set R and endings set E) and a map f from R to the set of all the subsets of E, satisfying two conditions:*

- *for every root r from R and every ending e from $f(r)$ the concatenation re is the element of V.*
- *for every word v from V there exists a decomposition: a root r from R and the ending e from $f(r)$ such that $v = re$.*

It is clear that if it is necessary to discuss prefixes too, it is possible to introduce the notion of ternary decomposition (and so on), but for simplicity we are only going to restrict ourselves to binary decompositions (omitting the word "binary").

This definition, evidently, offers different ways for constructing decompositions: it is quite possible that $R = V$ and all endings are empty words, or vice versa, when there is a single root — empty word, but all the elements of V serve as endings. Of course, something intermediate is of interest: when the size of roots set and endings set are significantly less than that one of the vocabulary.

Having V, R, and E, the possible map f may be constructed taking as $f(r)$ the set of all endings e from E such that the concatenation re lies in V. Nevertheless it is to be noted that as we didn't demand the uniqueness of the decomposition of the given word, there may be another map f for given V, R and E. The problem is how to construct a reasonable map f to recover V from R and E, taking minimum of memory (remember: the images of roots are sets of words!). Besides, the search time for the given token in the vocabulary should be reduced too.

If V is the vocabulary of word-forms for a language, there is some hope that taking E and R naturally (in grammar sense, as they are usually described in manuals, but suffixes are included into the roots), the above method leads to the reasonable map. It means that list L of all the possible values of subsets $f(r)$ will be not so large (comparatively with the size of V). In this case it will be sufficient to keep with the every root r only the index of its subset $f(r)$ in list L, so the necessary memory for the vocabulary will consist of two main parts: memory for roots set R (plus memory for index for every root) and memory for list L of possible sets of endings.

Let w be the word-form to be find in the vocabulary. Being small, the set E can be kept in RAM, and possible endings e for w may be found rather quickly. For every e the possible root r is determined uniquely ($w = re$) and it remains only to check that r belongs to R, and (if it is the case, and i is the index in L, corresponding to r) to make sure that e lies in the set $L[i]$ (for at least one of endings e.)

Note that there is another way of checking: first to find the root r, which is the left segment of w, and (if such r exists), second to check that the corresponding right segment lies in $f(r)$.

For the given decomposition more than one of the candidates for the possible roots (or endings) can be found. To avoid ambiguity (and improve access time) let us introduce some restrictions.

Let V be any set of words.

Definition 2 *Let E be the set of words, including the empty word. The set $R = R_E(V)$, consisting of the minimal left segments r of words w from V, such that the corresponding right segment e ($w = re$) lies in E, is named the roots set for V relative to E.*

It is goes without saying that selecting the maximal right segment of w, the possible ending is determined uniquely and the only root checking is sufficient (nevertheless, the ambiguity remains for the second method).

3 A grammar for the generating vocabulary

In the previous part we suggested to decompose the vocabulary into the two parts. Evidently it is more reasonable to create this decomposition directly instead of creating the full vocabulary and selecting roots and endings sets after. Here we want to discuss one approach to realize this idea.

Being different from the decomposition, described above, it looses the advantage of uniqueness of the word decomposition, but has another profits.

The starting point for this approach was the book [1], where main part of Roumanian inflective words were classified according to the methods of creating the word-forms. There were 100 groups for masculine nouns, 273 for verbs etc in the book, and about 30000 words with their group numbers were listed. The classification was made from the linguistical point of view, and, for example, the accents were taken into account. Nevertheless, this classification was useful and have lead to the idea to introduce the special grammar to formalize word-forms producing.

Let us consider a grammar rule:

$$[/]^*[\#][N_1]a_1\overline{b_1}a_2\ldots a_{n-1}\overline{b_{n-1}}a_n \longrightarrow a_1^{'}\overline{b_1}a_2^{'}\ldots a_{n-1}^{'}\overline{b_{n-1}}a_n^{'}\ N_2,$$

where a_i , $a_i^{'}$ are arbitrary words and either b_i is nonempty word or the special symbol $*$ stands instead of $\overline{b_i}$. N_j — endings set numbers. The interpretation of this rule is as follows.

Let w be the word to produce word-forms (basic word-form).

Every sign / indicates cutting the last letter from w. The obtained (after the deletions) word v is considered as a root (if N_1 exists) and N_1 is its index in endings sets list L. In any case the word v should have the form

$$f_0a_1f_1a_2f_2\ldots a_{n-1}f_{n-1}a_nf_n,$$

where every f_i is arbitrary (possible empty) word, not containing (for $i = 1, 2, \ldots, n-1$) the veto subword b_i . If there exists more then one of representation of this kind the first (scanning v from left to the right or vice versa if the sign # is present) should be selected. The special character $*$ instead of $\overline{b_i}$ admits arbitrary f_i .

After the evident substitution the word $f_0a_1^{'}f_1a_2^{'}\ldots a_n^{'}f_n$ serves as a second (or first, if N_1 is absent) root and N_2 is its endings set number.

Note that the special (but widely used) case of this rule

$$[/]^* \longrightarrow N_2,$$

gives the possibility to include v directly.

Using these grammar rules, we can formalize the process of creating of the decomposed vocabulary. According to the classification in [1], it is possible to build the grammar rules for every group. Sometimes more than two roots arise and more than one grammar rule is necessary.

Veto for b_i is conditioned by the necessity to determine the position of the subword a_i to be substituted. It appears that for the Roumanian language veto subwords can be avoided (being restricted by asterisks only).

Example 1 *Consider the first group of masculine nouns, described in [1]. One of the representatives of this group is the word "pom" (a tree).*

Let us introduce the notations for some morphological features of the Roumanian language: N — nominative; A — accusative; G — genitive; D — dative; S — singular, P — plural, V — vocative. Besides that the noun word-forms have two different forms in Roumanian: definite and indefinite ones. Let us denote them FD and FI correspondingly. Using these notations, we shell list all word-forms for noun "pom":

pom — NASFI or GDSFI,
pomi — NAPFI or GDPFI,
pomul — NASFD,
pomului — GDSFD,
pomii — NAPFD,
pomilor — GDPFD or VP,
pomule — VS.

Taking "pom" as a single root, we get an ordered set of endings (denoting the empty one by "-"):

$$T_1 = \{ \text{-, } i, \text{ } ul, ului, \text{ } ii, \text{ } ilor, \text{ } ule\}$$

It is essential that all the words from this group have the only root, coinciding with the NSFI, and the same endings set. So, the grammar rule for this group looks very simple:

$$\longrightarrow T_1$$

Example 2 *The word "ied" (a kid) is the typical representative of the another group. During the decline the alternation "d - z" appears here:*

ied — NASFI or GDSFI,
iezi — NAPFI or GDPFI,
iedul — NASFD,
iedului — GDSFD,
iezii — NAPFD,
iezilor — GDPFD or VP,
iedule — VS.

Here we have two roots: "ied" with the endings set:

$$T_2 = \{\text{-, } ul, ului, \text{ } ule\},$$

and "iez" with the endings set:

$$T_3 = \{i,\ ii,\ ilor\},$$

and the grammar rule for the whole group:

$$T_2\ d \longrightarrow z\ T_3.$$

Example 3 *Here is the grammar rule for the words from one of the more complicated verbs groups:*

$$/\#\ T_4\ \breve{a} * \breve{a} \longrightarrow a * \breve{a}\ T_5;$$

$$\#\ \breve{a} * \breve{a} \longrightarrow a * e\ T_6.$$

The equivalent form for this rule is:

$$/T_4\ \breve{a}\bar{\breve{a}}\breve{a}\bar{\breve{a}} \longrightarrow a\bar{\breve{a}}\breve{a}\bar{\breve{a}}\ T_5;$$

$$\breve{a}\bar{\breve{a}}\breve{a}\bar{\breve{a}} \longrightarrow a\bar{\breve{a}}e\bar{\breve{a}}\ T_6.$$

where veto contexts different from the stars were employed.

Three roots are generated for this group (for brevity, we omit three corresponding endings sets). So, for the verb "dărăpăna" (to collapse) the roots are "dărăpăn", dărapăn", "dărapen".

It is important that the real list of possible endings sets isn't large: the same ending sets serve for different groups. For example, 100 different groups of masculine nouns use only 15 different sets.

It is time to think about the morphological information. Evidently, according to its construction, every ending from the sets T_i , can be supplied with a morphological attributes (e.g. "-" in T_1 corresponds to NASFI or GDSFI; "i" to NAPFI or GDPFI and so on). The following method can be used to extract this attributes and simultaneously to compact the endings set.

Let us include all the possible endings for the masculine nouns in one common list, joining two equal (as words) endings in one, if they have the equal morphological features, but storing them separately, if

their morphological attributes differ. Every ending (with its attributes) can now be uniquely determined by its index in this list. So every T_i can be replaced by the corresponding bits scaleB_i, where 1 in j-th position signifies that the j-th ending from the list (with its attributes) belongs to T_i . The common list consists of

{ (-,NASFI or GDSFI), (i, NAPFI or GDPFI), (ul, NASFD), (ului, GDSFD), (ii, NAPFD), (ilor, GDPFD or VP), (ule, - VS), (e,NASFI or GDSFI), (ele, NASFD), (elui, GDSFD), (e, - VS), (u,NASFI or GDSFI), (i, NAPFD), (-, NAPFI or GDPFI), (lor, GDPFD or VP), (ă,NASFI or GDSFI), (a, NASFD), (ei, GDSFD), (ii, GDSFD), (ă, - VS), (-, - VS), (ăi, GDSFD) }.

Then the bits scales for T_1 , T_2, T_3 look like

(11111110000000000000),

(10110010000000000000),

(01001100000000000000),

correspondingly.

The experiments show that 866 grammar rules and 320 endings sets were sufficient for the Roumanian language to realize vocabulary decomposition, using 312 endings.

4 Dynamic methods

The above method of the decomposition is based on the knowledge about the morphological group of the given word. Nevertheless it is necessary to have the possibility to include a new set of word-forms for the given item w without this knowledge. In other words, we need to detect the group number dynamically. Three steps should be done to solve this problem.

1. First, the word-forms themselves should be obtained. The special program can be elaborated to facilitate this boring work. The information concerning part of speech (verb, noun etc) and may be

something else (e.g. gender) can be obtained interactively. Then deep linguistic analysis (and may be one-two additional questions (usually about the alternation or suffixes)) permits to predict the possible structure of the base-forms. The corresponding practical algorithm, elaborated by E.Boian, A.Danilchenco and L.Topal [2] for Roumanian gives correct word-forms in 95% of cases.

2. The second step is to detect the roots from the word-forms. An ingenious method, suggested for this purpose in [3] has the advantage of universality (it can be applied to every highly inflexional language). But in this method the chosen roots are usually the longest ones, though for the effectiveness it is more preferable to have short roots (the only restriction is the possible size of endings sets). For instance, had we the endings "d", "zi", "dul", "dului", "zii", "zilor", "dule" in the list of possible endings, the only (and shorter) root "ie" for example 2 would be sufficient.

   Our approach is based on different ideas. Let us suppose that for every grammar attribute the corresponding empty ending is included in E. Then for every word-form v there exists at least one decomposition of form $v = re$, where e lies in E. Taking into account all possible decompositions of this kind, we get the set of all possible words r with the sets of word-forms, where this r was used. The union of all these sets is the set of all the word-forms, so we have a covering of the last set. Choosing the minimal (in the corresponding sense, depending on the subject to be minimized) covering, we select the roots.

   Example 4 *Let word-forms and endings set E be as in example 2. Adding empty endings to E, we get the possible decompositions for word-forms:*

   "ied" = "ied" + "";

   "iezi" = "iez" + "i" = "iezi" + "";

   "iedul" = "ied" + "ul" = "iedul" + "";

   "iedului" = "ied" + "ului"="iedului" + "";

"iezii" = "iez" + "ii" = "iezi" + "i" = "iezii" + "";

"iezilor"= "iez" + "ilor"="iezi" + "lor"="iezilor" + "";

"iedule" = "ied"+"ule" ="iedul" + "e" = "iedule" + "".

The candidates for roots and their sets are (omitting those, which are the only representatives of their sets):

"ied" — (1011001);

"iez" — (0100110);

"iezi" — (0100110);

"iedul"— (0010001);

Two different minimal coverings may be selected: ("ied","iez") and ("ied", "iezi"). The first one is shorter and it is quite natural that this pair should be selected if the main criterion is memory for the roots. But suppose that (in contrary to example 2) the endings set for the root "iez" has not been introduced yet. Had we the endings set for "iezi" introduced and the memory for new endings sets restricted, it would be possible that the second pair were more preferable.

Being flexible the definition of the binary decomposition permits to build different ones, depending on purpose. It means that varying the sets E, R and map f, we can solve different problems with the same vocabulary V. For instance, a sufficiently small endings set (consisting of traditional endings), is convenient to create a vocabulary from the very beginning. Having a complete vocabulary we can redecompose it, according to the new criterion.

For example, using the given size of endings sets, we can add some more endings (e.g. by counting the frequency of their appearances). Then we may try to solve either memory problems or (/and) time ones. Let us consider one possible solution.

For simplicity suppose for a moment that the morphological information is not available. Then the decomposition $(E, R_E(V))$, described in the beginning, minimizes the access time, because

for searching it is sufficient to check the only root. The simplest way to get it is:

- First, to select a new endings set E.
- Second, to create the full vocabulary and empty list L of endings sets.
- Third, to make a new decomposition word by word. More exactly, to put a word w in, the longest right segment e from E should be found. Then the root r can be obtained from the decomposition $w = re$. Two cases are to be considered.

  If r is a new element from R, consider the set S, consisting of one ending e only. If S was not in L, it should be added here with the weight 1, otherwise its weight should be increased by 1. Surely, $f(r) = S$.

  If r was in R before that, let $T = f(r)$. Let S be a union of T with e. The above procedure can be applied. As to set T, its weight should be decremented by 1, and T itself should be eliminated from L, if the weight became zero.

It should be noted that this method of redecomposition takes a lot of additional memory: first — for full vocabulary, second — for the temporary (intermediate) sets. It is more reasonable to create a new decomposition in statu, changing the existing one by adding (or deleting) endings. For example, to include a new ending e we have to check those word-forms, which have e as the longest ending. The corresponding roots and endings sets should only be changed.

It is curious that this method of decomposition often joins endings of different parts of speech into one set. If the grammar features are to be taken into account it is not the case, but here the unique decomposition property can be loosed: it is possible that ending e_1 for another part of speech can be longer, than the longest ending e for this part of speech. To avoid this problem the new ending e_1 with the corresponding grammar attribute should be included.

3. Roots being selected, it is easy to check if there exists the corresponding grammar rule for the generating of this roots (with the corresponding endings sets). If not, the new grammar rule should be created (and, perhaps, new endings sets should be included). Sometimes it is possible to change the existing rule. Here the veto words can be of help.

Let us note that, being obtained with the help of grammar,the vocabulary itself contains whole grammar information, but for the most applications the grammar rules can be omitted. In the case, when they are really useful (e.g. for the knowledge acquisition), the problem of similar roots arises [3]. So it is high time to consider similarity.

5 The similar words detecting

It is well known that every natural language contains some irregular words (e.g. auxiliary verbs in English and Roumanian), which are the exclusions to rules of inflexion. From the point of view of grammar rules constructing they are very unsuitable and sometimes it is more preferable to keep every word-form separately instead of constructing rather complicated rules.

The question is how to determine the degree of irregularity of the word-forms and (if it isn't large) how to create the corresponding grammar rules. To answer this question it is sufficient to be able to estimate the similarity of two given words.

A possibility to determine that two given words are similar may be useful for different purposes. For example it was helpful to realize the correct word suggestion when the given word had one or more mistakes. But even for this case different criteria appear. There may be at least three different causes of mistakes: the mistakes can arise because of bad knowledge of the word (here the similar words are those, which sound similarly); because of scanner errors (here the similar words are those, whose letters looks similar in the given alphabetic design) and, at last, the mistakes may be due to wrong key pressing (here the similarity depends on the arrangement of the keyboard). As to the grammar

rules the similar letters are those, which are essential for grammatical alternation (see "d" and "z" in Example 2).

To solve the similarity problem independently we suggested to divide it into two problems. First, to solve the similarity problem for letters only, getting as result a similarity letters matrix M. Depending on the purpose, this matrix can be filled independently (in our programs by integers between 0 and 9).

The second task is to find the algorithm for the calculation of the similarity degree of two given words. The trivial solution is to give this function the maximal value if the words differ in at least one letter or in one permutation of two neighboring letters. In other cases the function has zero values. (It solves the problem of one mistake and can be satisfactory for the simplest Spelling Checker suggestion.)

To detect more than one mistake (or to create a grammar rule) more complicated algorithms are employed. First of all consider the following variation of the well-known algorithm for searching the maximal common subsequence for the two given words v and w (n and m long correspondingly).

Consider n by m matrix L, where by induction $L[i,j] = \max(L[i-1,j], L[i,j-1], L[i-1,j-1] + M[v[i], w[j]])$, and $L[0,j]$, $L[i,0]$ are considered to be equal zero. Taking L[n,m] as a criterion of the similarity we can get a rather good similarity function (e.g. $f(w,v) = L[n,m]/(n+m)$). Nevertheless for the purposes of suggestion this method cannot be considered as a satisfactory one: it uses too many useless calculations. To improve it, let us restrict our suggestion to a reasonable bound: the similar word should be found if there no more than two mistakes were made. The possibility to correct three mistakes remains, but only in the case of "natural" mistakes. This restriction gives us a possibility to construct only 5 diagonals from matrix $L[i,j]$ (where the difference between i and j is at most 2, $L[i,j]$ being zero in other cases). Moreover, it is not necessary to calculate all diagonals: if the values are too small we can be sure, that more than two mistakes were made and stop calculations). The real method was slightly more complicated (suggestion shouldn't be too talkative), but the main idea was exactly this one. To accelerate the search similar roots and similar

endings are looked for separately.

The last remark is that instead of letter similarity matrix M the corresponding word similarity tree could be used (it helps, for instance, to display more adequately the alternation of a letter with several ones, which is typical of Roumanian).

6 Implementation notes

Now we will discuss some implementation aspects in the case when the morphological information is not available. It should be taken into account that we want to achieve two contradicting aims:

1. to shorten data base volume;

2. to reduce the access time for achieving reasonable work time.

Obviously Lempel-Ziv, Lempel-Ziv-Welsh packing methods and their modifications give a good compress coefficient but they are not applicable in our case because the response time will be too large. Therefore the authors of this article decided to use semantic information during the word packing into the constant base because just this one contains the main word fund having then a huge volume. Note that, shorting size of data base allows scanning it more quickly approaching aim 2.

Consider the representation of a word in the data base (Figure 1).

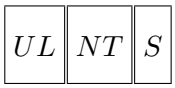
It can be seen that a word is divided into three parts:

1. the first two characters saved separately (this point will be discussed later);

2. the rest of the root;

3. the set index of valid terminations set for this root.

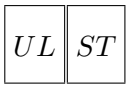
The UL field contains the length of the root in characters including $C1$ and $C2$. The root S is saved in a special encoded form. We use the source words length opposed to the encoded word length due to



UL — length of a source word root (including the first two letters $C1$ and $C2$)
NT — valid terminations set index
S — common source word root for all terminations from the pointed set (without $C1$ and $C2$)

Figure 1:Constant base page element structure.

the possibility to calculate the encoded length from the source one (the inverse is not the case). This field is used for an effective word search. We do not need to compare words when their length is not equal. It saves a very expansive string comparison. In the case when the length and the root are the same, we have to find out if there is the source word termination in the set pointed out by field NT. The oversimplification was made for the user's base. The word is not divided into a root and a termination (Figure 2).



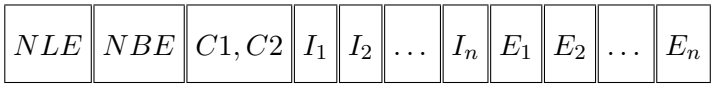
UL — length of a source word root (including $C1$ and $C2$)
ST — encoded word without $C1$ and $C2$

Figure 2:User's base page element structure

The size of the element for the word is limited and constant. So the following three cases are possible: the size of a word

1. is equal to the element size;
2. is less than one;
3. is greater than one.

The first case is ideal. In the second case there is an unused memory. The latter is not very efficient but there is no problem. The third variant is the most complicated. The rest of the word which is not contained in the first element will lay over the next one. So we must know how to distinguish between the beginning of the word and its continuation. For this purpose we consider the structure of a page (Figure 3).



NLE — logical elements quantity
NBE — physical elements quantity
$C1, C2$ — the first two characters of each word on the page
I_i — indirect indexed vector
E_i — page element

Figure 3:Constant and user's data base page

The first field shows how many roots are stored on the page and the second field shows how many elements are occupied by the roots. Pages as well as elements have a fixed size. So it is possible to access directly to an indexing file by the page number. Hence we have fast access only by operating system means.

The next field keeps the first two characters of all the words saved on the page. This permits one to save some memory without any loss of efficiency. Vector I_i was introduced to simplify both the modification of data base and addition of the new words. Using this vector we can distinguish between the beginning of a word and the continuation of a word. Its usage as an indirect address map sets the lexicographical order of the elements. We can use a direct element indexing because of a fixed element size. Summarizing the above given we can use binary search methods. Note that this method allows both effective searching and adding words. As it is known if there are N elements, we can find out necessary one by log2(N) comparisons. Suppose we have W words

in the data base and P words in the page, then $D = \log_2 \frac{W}{P} = \log_2 W - \log_2 P$ disk accesses are needed. Hence, minimizing parameter D we shorten access time to the data base. For this purpose the following method was elaborated. The first word from each page is stored in the special table named HASH-table (Figure 4).

H_1	H_2	...	H_n

H_i — table element

SA	NP

SA — word in ASCII code, the first from the pointed page
NP — page number in the data base

Figure 4:The HASH-table structure

The size of this table will be D elements. Controlling P we can make the HASH-table be in RAM completely. Note that pages of the constant base contain the word roots, hence the volume of pages grows. Applying binary search to HASH-table we can find out the page possibly containing searched word. After reading the page we will know whether it contains the necessary word. Thus we minimize D to 1 (i.e. there is only one disk access). Thus we obtain the following data base general structure (Figure 5).

The following method was used for increasing effectiveness. More often used words were extracted into a separate data base permanently being in the RAM. Its structure is shown in Figure 6.

Using words length and array L_{ij} we get the hash-function for the minimization of operation number for a binary search. All words are represented in ASCII code because the volume of this base is rather small.

In conclusion we give some evaluation of ROMSP. All measurements have been made on IBM PC AT compatible computer — 12 MHz, 40

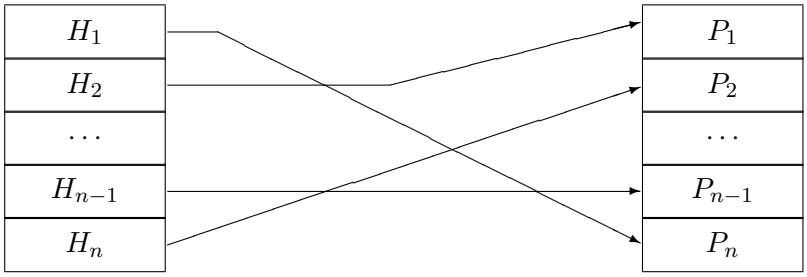


Figure 5:The general data base structure

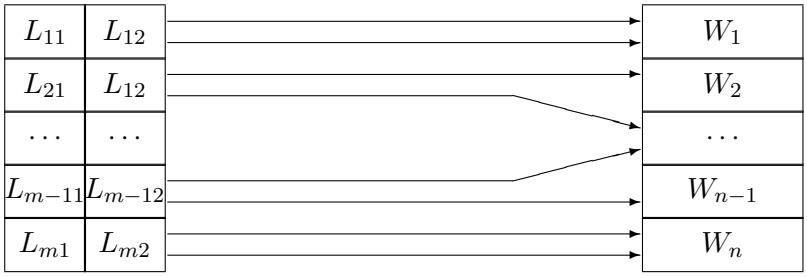


Figure 6:Data base structure of the more often used words

MB HDD, 28 msec average seek.

- Searching speed — 70-100 words per second;
- the volume of the data base containing about 330000 words (3.6 MB) is equal to 700 KB;
- the ROMSP software volume — 300 KB;
- programming language — TurboPascal.

References

[1] A.Lombard , C.Gâdei. Dictionnaire morphologique de la langue roumaine. Bucuresti, Editura Academiei. 1981.

[2] E.Boian, A.Danilchenco, L.Topal. The automation of speech parts inflexion process. Computer Science Journal of Moldova. 1993, Vol. 1, No 2 (to appear).

[3] D. Tufis. Paradigmatic morphology learning. Computers and Artificial Intelligence. 1990, Vol. 9, No 3. p. 273-290.

S.Cojocaru, Ph.D.,
M.Evstunin,
V.Ufnarovski, Ph.D,
Institute of Mathematics,
Academy of Sciences, Moldova
5, Academiei str., Chişinău,
277028, Moldova
e-mail: *ufn@math.moldova.su*

Received January 3, 1993