***ElephantFromFly Application***

***ElephantFromFly*** is a Java console application which solves words chain puzzle.

Coding style: <http://source.android.com/source/code-style.html>

**Task formulation**

It is required to build words chain from start word to end word using words from given vocabulary (start word can be absent in vocabulary). Each word in chain should be different in one character from previous. All words in chain must have equal length. It is desirable to find words chain of shortest length. Name “Elephant from Fly” has sense for Russian language because these words have equal length (муха -> слон), for English language it could be for example “Dog from Cat” (dog -> cat).

Difference between two words is calculated the following way: function counts different characters in appropriate positions of words (word1[i] compared to word2[i]).

Examples

1. difference(“cat”, “dog”) = 3.
2. difference(“cat”, “cot”) = 1.
3. difference(“cat”, “cat”) = 0.
4. difference(“abcd”, “dcba”) = 4.

Example of task solution.

Let’s we have a vocabulary: {cat, fly, dog, elephant, cog, rat, rabbit, cot, pizza, space}

Then we can build words chain: cat -> cot -> cog -> dog

Or we can build words chains: cat -> rat, fog -> dog -> cog -> cot -> cat

But we can’t build words chain: cat -> fly, or space -> pizza

**Application inputs and outputs**

* Input words file - path to the file which contains start and end word (one word in row).
* Input vocabulary file – path to vocabulary text file (one word in each row).
* Maximum words chain length constraint – words chains longer than this constraint are ignored.
* Input timeout value in minutes – defines time after which search process will be interrupted and application outputs result found by this time. This is necessary because of high combinatorial complexity of algorithm for vocabularies of real size.
* Application prints results to console which can be redirected to file. If input parameters are not valid error info is printed.

**Algorithm**

Brief description of algorithm, other details are in source code.

1. Get input parameters.
2. Validate inputs and print error if inputs are not valid.
3. Load input words and vocabulary. Length of input words is equal (verified on inputs validation step). Only words of required length are loaded from vocabulary file and all words are converted to lower case.
4. Apply the following heuristic which can decrease search space: sort words in difference with end word increasing order (first will be the end word for which difference is equal to zero). Because it is desirable on each step to go to the word which is closer to end word.
5. Implement in-depth-search algorithm: on each stage select *<CurrentWord>* word such that *difference(<CurrentWord>, <PreviousWord>) = 1*, add it to chain, exclude from list of remained words and go to next stage. When dead end is found roll back and select another word. When end word is reached store result and roll back to select another word. Algorithm interrupts branches of search when current length of words chain violates maximum length constraint or it reaches earlier found minimum words chain length, because we are only looking for chain of minimum length.
6. On each iteration check if timeout is not expired, otherwise stop search.
7. Provide final results.
8. End.

**Source Code Description**

Source code of ElephantFromFly application is implemented on Java language. Unit tests are implemented using JUnit framework.

**Java classes**

* ElephantFromFly – contains application entry point – main() function. Inside it input parameters are loaded and validated, algorithm is executed, results are printed and errors handled.
* Errors handler: provides error messages for error codes and handles exceptions.
* InputWords – keeps start and end word of words chain. Imposes constraints and doesn’t allow creating instance where words are empty or have different length.
* ListCursor<T> - template class used for iteration through List<T>. This class was implemented because Java’s Iterator<T> class doesn't allow getting reference to current element at which it is pointed to.
* PuzzleException – keeps info about errors which can happen in program.
* TimeCounter – used for counting time since start of some moment.
* Vocabulary – vocabulary of words: provides functionality of loading from file.
* WordInfo – class used by words chain puzzle algorithm: contains word, difference of this word with end word (target word) and flag which defines whether this word is already used or not.
* WordsChainPuzzle - words chain puzzle algorithm. Looks for the shortest words chain from start word to end word using given vocabulary.

**Things to do for enhancement**

* Think how to decrease search space and optimize algorithm.
* Implement possibility to stop search by user.