International Journal of Computer Science and Informatics

Volume 3 | Issue 3 Article 4

January 2014

MULTIPLE FIND AND REPLACE TOOL FOR TEXT EDITORS

VIGNESH PRABU
Department of Computer Science Sri Krishna College of Engineering and Technology Coimbatore, India,
vigneshprabuu@gmail.com

SIVAGNANAM MUTHARASU
Department of Computer Science Sri Krishna College of Engineering and Technology Coimbatore, India,
skannanmunna@yahoo.com

RAMA JAYAM
Department of Computer Science Sri Krishna College of Engineering and Technology Coimbatore, India,
ramajayam.g09@gmail.com

MULTIPLE FIND AND REPLACE TOOL FOR TEXT EDITORS

VIGNESH PRABU[1], SIVAGNANAM MUTHARASU[2] & RAMAJAYAM[3]

[1,2&3]Department of Computer Science
Sri Krishna College of Engineering and Technology Coimbatore, India
E-mail: vigneshprabuu@gmail.com, skannanmunna@yahoo.com, ramajayam.g09@gmail.com

Abstract— In text editors like Notepad and vi editor, we can find a string and replace it with another one. It is not possible to find multiple strings and replace with a single string. This paper reveals other 2 approaches: Finding multiple strings and replacing it with a single string, finding single string and replacing it with multiple strings. Java based string manipulations have been carried out on the contents of the file to implement our functionalities. Thus, our tool provides excellent portability option.

Keywords- *find and replace, multiple find, multiple replace, text editors, java swings.*

I. INTRODUCTION

Find and replace itself has a rich legacy, dating back to teletype text editors such as Unix ed, in which the substitute
command was the primary way to change existing text. Although the importance of find and replace has faded somewhat with the rise of direct-manipulation text editing, virtually every text editor still includes a find and replace command [3].

In today's world, text editors have become standard medium of storing data. In academic, governmental and commercial environments, the majority of documents are now being prepared using Text Editors.

When repetitive changes must be made to a text file, there are several approaches to consider. The changes can be performed manually, which is tedious if there are many modifications to be made. Custom programs can be written to perform these changes automatically, but this requires programming skill, and familiarity with either the editor's programming interface or file format [4]. Find and replace is one of the familiar methods to make repetitive changes.

This command in a typical text editor allows the users to choose one of the two alternatives: replace one match at a time or replace all matches at once.

If we want to change n number of different strings by a single string, then we will use the find and replace command n times. This is slow and difficult. Eventually, the user starts getting bored to implement the same operation n times.

Also, if we want to replace data-x with data-1 in its first occurrence, data-2 in its second occurrence, …, data-n in its n^{th} occurrence, data-1 in its $(n+1)^{th}$ occurrence, data-2 in its $(n+2)^{th}$ occurrence and so on then we have to do it individually for all the occurrences. But this method is tedious since the whole document has to be read manually to find all the positions of the string data-x to finish this job.

This paper explores new alternatives to find and replace patterns in a text file. This approach provides the way to search multiple patterns in a file and replace it with a single pattern and, also to search single pattern in the file and replace it with multiple patterns in sequential manner. In this way repetitive changes can be done in a single operation rather than individually capturing the patterns and replacing it. It will be very useful in editing newspapers, scientific documents, etc. where search operations are extensively used.

The rest of this paper describes our experience with designing and evaluating user interface for multiple find and replace. After surveying related works, we present the user interface, highlighting our implementation results and pilot evaluations. Then we present details of our algorithm used. Finally, we discuss some of the conclusions that can be drawn from our experience.

II. RELATED WORKS

There are various algorithms proposed to perform the task find and replace. Some of them are Cluster-Based Find and Replace, Advanced Find and Replace, HandyFile Find and Replace, HTML Search and Replace etc. [3]. Cluster Find and Replace perform conventional find and replace based on clusters i.e. on parts of files [3]. Similarly, HandyFile Find and Replace provides the same functionality on multiple files [2]. Structural find and replace is used extensively on java source codes. It is used to search for instances of the classes, expressions etc and replace it [7]. Find and replace method implemented by new operational transformation algorithm, provides string operations based find and replace [8].

But all these algorithms search a single string and replace one of its occurrence or all of its occurrences by another string.

Consider a scenario in which we have to replace multiple patterns by a single pattern. For example, if we want to replace the entries such as "1000", "2000", "3000" in a file with a single pattern "222". What can we do? We have to perform the usual find

and replace 3 times. Will it be possible to do the same for 100 different patterns? Yes, it is, but we need to perform the find and replace command 100 times. But our new method will achieve this in a single stroke.

Consider another scenario. We want to replace the pattern "ram sundar" with "ram" in its first occurrence, "sundar" in its second occurrence, "ram" in the third occurrence, "sundar" in the fourth occurrence and so on. Is it possible? Yes it is, but it takes n times to perform find and replace operations where n is the number of occurrences of "ram sundar". In this case, we need to make the changes manually i.e. we have to change the i^{th} occurrence of "ram sundar" by "ram" when 'i' is odd and the i^{th} occurrence of "ram sundar" by "sundar" when 'i' is even. This way of approach is very slow and complicated. User concentration is required for every replacement, which doesn't scale to large, complicated task. When the number of strings to be replaced becomes large, the work becomes even more tedious. Eventually, the user starts getting bored and impatient to perform find and replace for each pattern, which leads to error prone situations. This functionality can be achieved using our modified find and replace method in an easy manner.

One of the major advantages of our method is that the replacement operations are carried on character level and hence any kind of pattern can be found and replaced. It can be used on program codes since it has the capability of recognizing numbers and special characters in addition to the alphabets.

The only character that is not recognized in our method is "$" since it is used as a delimiter to separate multiple patterns.

III. PROPOSED METHODOLOGY

A. Multiple find

In this section, we propose an algorithm to spot multiple patterns and replace it with a single pattern. The algorithm takes as input a set of patterns (which are substrings of a document) which has to be replaced and the replacement pattern. Our algorithm has the following steps.

1) Break the replacement pattern into a character array [1].
2) Break the first pattern to be replaced into a character array.
3) Read the lines in the file one by one and create an individual character array for each line.
4) Then extract the characters of the first pattern to be found.
    a) Compare those with the character array of each line and if the comparison extends to be equal to the length of the pattern to be found, a match is said to have occurred.
5) After locating the first match, replacement is done character by character of the replacement array onto the matching pattern iteratively. In this way, all the consecutive patterns are found and replaced.
6) Finally, the character array of the lines are concatenated into lines and written back into the file without any change in the indentation and format after flushing it. Similarly, character array of all the lines are concatenated sequentially and written onto the same file.

B. Multiple Replace

Our algorithm for replacing single pattern with sequential multiple patterns takes a series of inputs such as pattern to be found, multiple replacement patterns.

1) Read the lines in the file one by one and create an individual character array for each line.
2) Then extract the characters of the pattern to be found in an array.
    a. Compare those with the character array of each line and if the comparison extends to be equal to the length of the pattern to be found, a match is said to have occurred.
3) After finding the first occurrence of the match, first replacement string is broken down into a character array and the matching pattern is replaced with this replacement pattern appropriately.
    a. In this way the following consecutive matches are located and the consecutive replacement pattern is broken down into a character array and replacement takes place.
    b. The replacement patterns count is iterated using a modulo function so that found matches gets replaced by the consecutive replacement patterns even if the number of occurrences of the match is greater than the number of replacement pattern.
4) Finally, the broken down array of characters of the line are concatenated into a line and written onto the same file. Similar procedure is carried out for all the lines in the file. Before doing this, the file is flushed so that no overwriting takes place.

One of the significant merit of our algorithm is that, it has the ability to adjust the space needed i.e. if the length of the pattern to be found is greater than the replacement pattern unnecessary spaces in the array of that line are removed during replacement or if the replacement pattern is greater than the pattern to be

found, the space needed are allocated automatically for replacement.

IV. IMPLEMENTATION RESULTS

This section describes user interface of our find and replace tool. The interface was created using Java Swings [5] [6].

A. Multiple Find

In this section, we implement our method of finding multiple strings and replacing with a single string.

Consider the text file "santa.txt" in which the words such as "Black Peter", "Nikolaus" and "Saint Papa Noel" has to be replaced by "Santa Claus".

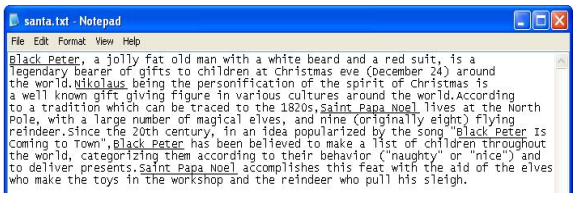


Figure 1. Contents of "santa.txt" before the execution of multiple find command.

Fig. 1 shows the content of the file "santa.txt" in which the strings to be found are highlighted by underline.

TABLE-I.

Label	Strings
Strings to be Found	Nikolaus$Saint Papa Noel Black Peter
Replacement String	Santa Claus

TABLE-II.

String to be found	Number of occurrences
Nikolaus	1
Saint Papa Noel	2
Black Peter	3

TABLE-I provides the details about the strings to be found and the string to be replaced whereas TABLE-II provides the number of occurrences of each pattern to be found.

Fig. 2 shows the multiple find and replace tool in action. Browse button in figure 2 is used to choose the file to be operated on ("santa.txt"). Also we can select the options whether to operate on partial words or whole words by selecting the appropriate radio buttons.

Fig. 3 shows the contents of the file after the execution of multiple find command. We can see that the strings "Nikolaus", "Saint Papa Noel" and "Black Peter" have been replaced by the string "Santa Claus".

B. Multiple Replace

Consider the updated file "santa.txt" to implement multiple replace method. Figure 3 shows the contents of the updated file.

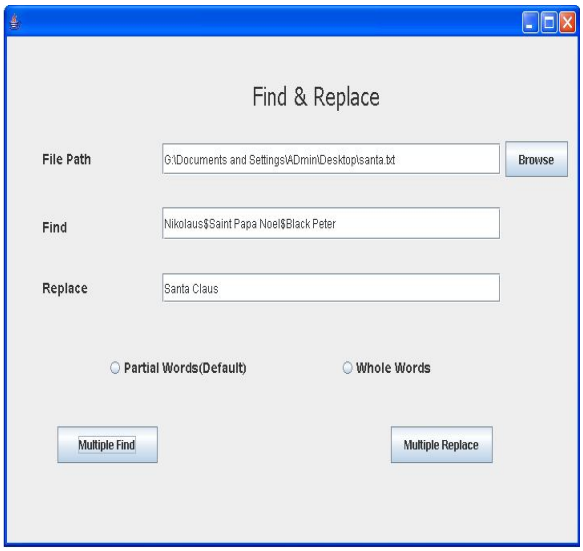


Figure 2. Multiple Find and Replace Interface

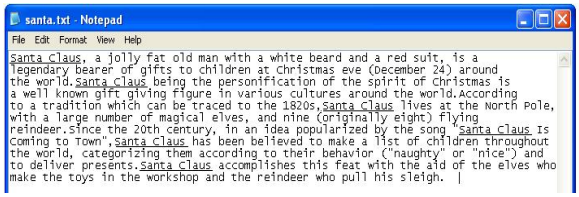


Figure 3. Contents of "santa.txt" after the execution of multiple find command.

TABLE-III.

Label	Strings
String to be found	Santa Claus
Replacement strings in sequential order	Black Peter$Nikolaus$Saint Papa Noel

TABLE-IV.

String to be found	Number of occurrences
Santa Claus	6

TABLE-III provides the information about the string to be found and the replacement strings that are replaced in sequential order. TABLE-IV provides the number of occurrences of the searched string.

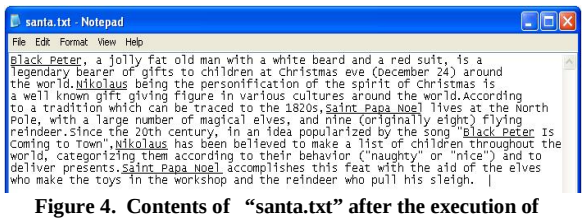
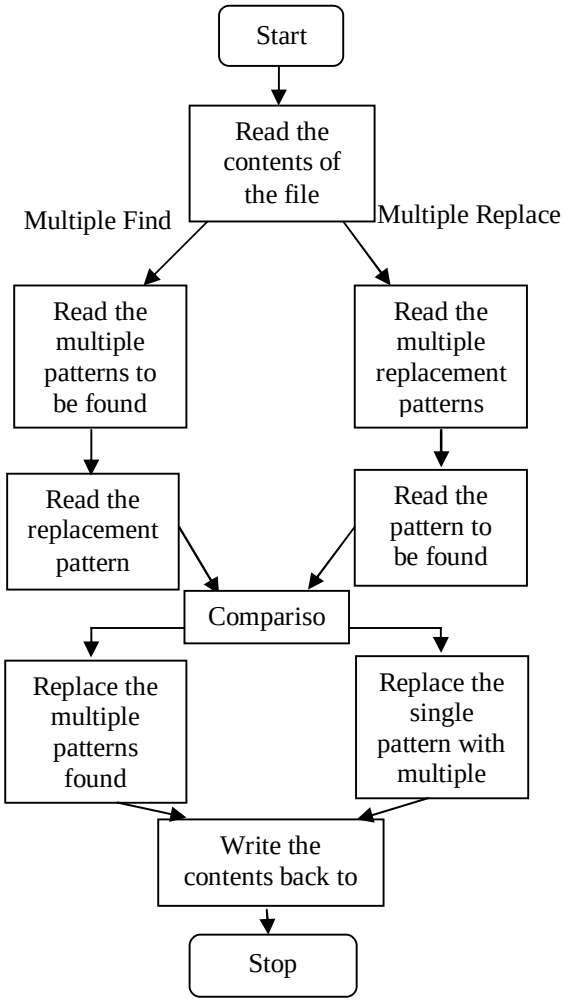


Figure 4. Contents of "santa.txt" after the execution of multiple replace command.

Fig. 4 shows the contents of the file after execution of the multiple replace command. We can see that the 3 occurences of the string "Santa Claus" has been replaced to "Black Peter", "Nikolaus" and "Saint Papa Noel" in the order in which it is given.

V. WORK FLOW DIAGRAM



VII. FUTURE DISCUSSION

As discussed earlier, our multiple find and replace method differs from conventional find and replace methods. But it cannot perform multiple find and multiple replace simultaneously i.e. it cannot find multiple patterns and replace them with multiple patterns. We are working on that which takes as input the pattern to be found and the replacement pattern in pairs so that multiple patterns can be found and replaced.

VIII. CONCLUSION

Unlike the traditional replace-all and replace-with-confirmation approaches, multiple find and replace allows users to focus on different strings to be found and replaced.

We compared multiple find and replace with traditional find and replace in a small user study. And most of the users prefer to use our multiple find and replace over the conventional one because of its efficient search and replacement mechanism.

Multiple find and replace suggest ways that search and replace operation are carried in an efficient way. We hope that these techniques will evolve to reduce tedium and increase correctness in difficult tasks that demand human attention.

REFERENCES

[1] Herbert Schildt, The Complete Reference: Java 2, 5th Edition, Tata McGraw Hill Publication.

[2] HandyFile Find And Replace: Office Edition, Silverage Software, Inc.

[3] Robert C. Miller, Cluster-Based Find and Replace, MIT Computer Science and AI Lab Cambridge, USA.

[4] David Kurlander & Steven Feiner, Interactive Constraint-BASED Search and Replace, Columbia University, USA.

[5] Maxim Mossienko, Structural Find and Replace, JetBrains.

[6] http://docs.oracle.com/javase/tutorial/index.html.

[7] http://netbeans.org/index.html

Articles in journals:

[8] Santosh Kumawa & Ajay Khuntet, String Based New Operations – Find and Replace by New Operational Transformation Algorithms for Wide-Area Collaborative Applications, Poornima College of Engg. Jaipur, Rajasthan, India, International Journal of Computer Applications, Volume 6 – No.7, September 2010.