資料結構

Report 1

姓名： 彭敬樺 學號：0416106

This report will briefly describe the process and algorithm used to accomplish the homework. The main output required to be printed in an output file consists of mainly two parts. One for calculating the frequency of the pattern without considering substring and the other is to search the position of the pattern inside the text including the appearance in substring.

Firstly, it should make sure the file pointer point to the correct file that is inputted by user before the calculation progressed. If the output file didn’t exist, then it will create a new file.

Next, we calculate the number of character of the pattern (keyword) that will be compared. This calculation is done to make sure that the memory used to save the pattern to be as minimal as possible. After calculating, we insert the pattern from the file that serves as input. Also, not forget to point the file pointer back to the beginning of the file to ensure no mistake when comparing the pattern and the full text.

After completing the pattern array, we start calculating the failure function that will be used later for fast substring compare. Simple explanation about how to generate failure function: Two integer representing the address of the character in the pattern, namely first and second. If the character at address “first” and “second” is the same then the value of failure function at address “first” is equal to second + 1 and we need to increment “first” and “second” by one. If it is not the same, then we track back from the failure function to search where we should start the search again.

Failure function discard excess search from unneeded comparison. We could skip the useless comparison for better efficiency time by looking at the value of the failure function value of that specific character that has been calculated beforehand.

For counting words frequency without including substring, we need to check whether the word is identical or not with the pattern. If the character that is read contains non-character value, then we start re-checking the pattern from the beginning. The reason to restart the search is because we assume that each word is separated by some non-character. If the character in the text and the pattern is the same then keep checking if it is the same throughout the pattern. If we found all match, we still have to confirm that the next character in the text won’t be a character anymore then we could be sure that it is identical one. If this condition, is not fulfilled then we have to start from the beginning of the pattern all again.

Most important thing that has to be done before doing search again is to reset the file pointer back to the beginning of the input file. For the second part, we need to print out the positions of the pattern inside the text including the substring. The variable word will represent and save the number of words we are on. The “word” variable will be increment if they meet the requirement that needs the input to be non-character. For this part, failure function plays a great role for speeding up thing and reducing the time complexity for the searching algorithm. If we found the character in text and pattern to be matched then keep searching, else then we should go back to the address position saved in the failure function of the character before in the pattern. This address reposition will try its best to reduce the useless comparison as much as possible.

All character that is being compared has been converted to lower case. Therefore, this program won’t differ the result from the word being in upper case or lower cases. However, the usage of number and punctuation will be considered as different.

Because in this program written by me has used some dynamic array. Therefore, “free” function is a must to be used. And also the file need to be closed before terminating program.

This is all the report Thank you for reading. All the explanation is being described briefly only.