Group 37

|  |  |  |  |
| --- | --- | --- | --- |
| 2012/CS/110 | RANASINGHE | A.A.S.S. | sanjeewa.alahakoon.ranasinghe@gmail.com |
| 2012/CS/111 | RANASINGHE | H.U.R.K.N. | kapilaranasighe49@gmail.com |
| 2012/CS/112 | RANATHUNGA | K.K.T.P. | tharindu.prf@gmail.com |

# Introduction.

This include a dynamic solution for just justification problem (word wrapping). In text display, the line break function to continue on a new line when a line is full, so that each line fits in the window visible, so that the text can be read from top to bottom without horizontal scrolling. Newline eliminate hard coding of the separator line break in pieces and enables dynamic fusion of text with the new automatic line decisions on the fly (for example, when a window is resized). Word wrap is the additional feature of most text editors, word processors and web browsers, the dividing lines between and not in words, except when a single word is longer than one line.

Author have come up with a recursive formula which calculate the minimum total additional space error for a given text sequence having a character limit in a single line. The goal of this report is to justify the algorithm and techniques which have come by the group to get a minimum running time. Also this report include the how the group have successfully achieved the extra goals that are mentioned in the problem namely the Gui and Word Suggestion Dictionary.

At the beginning of the program user have to enter the maximum character limit to have in a single line. And when the user type or copy and paste, after an every key stroke typed text and the limit will be put in to the algorithm and output will be displayed on the fly.

# Recursive Algorithm.

First we construct a table storing the error of each and every combination of starting word and ending word.

For example let’s assume the text as :-

*She is happy but is a blue gal. I am all gone.*

Character limit =15

Word 1:- She Word 9:- I

Word 2:- is Word 10:- am

Word 3:- happy Word 11 :- all

Word 4:- but Word 12:- gone.

Word 5 :- is

Word 6 :- a

Word 7 :- blue

Word 8 :- gal.

According to the above example the content in the cell (1,3) gives the total error if we put word 1 and word 3 to a single line. Which is “She is happy”.

Number of characters:- 12

Error: - (15-12)3 =27

If the number of total characters in a single line exceed the line limit value of the cell will be replaced with the infinity. (In the program that value is 1000000000 for the ease of the programming)

After calculating all the values table will look like this for the above example.

Ending Word

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Starting Word |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 1 | 1728 | 729 | 27 |  |  |  |  |  |  |  |  |  |
| 2 | 0 | 2197 | 343 | 27 |  |  |  |  |  |  |  |  |
| 3 | 0 | 0 | 1000 | 216 | 27 | 1 |  |  |  |  |  |  |
| 4 | 0 | 0 | 0 | 1728 | 729 | 343 | 8 |  |  |  |  |  |
| 5 | 0 | 0 | 0 | 0 | 2197 | 1331 | 216 | 1 |  |  |  |  |
| 6 | 0 | 0 | 0 | 0 | 0 | 2744 | 729 | 64 | 8 |  |  |  |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1331 | 216 | 64 | 1 |  |  |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1331 | 729 | 216 | 8 |  |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2744 | 1331 | 343 | 1 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2197 | 729 | 27 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1728 | 216 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1331 |

Once we have the table constructed, we can calculate total cost using following recursive formula. In the following formula, C[j] is the optimized total cost for arranging words from 1 to j.

For calculating the values for array C following recursive formula can be used.

**C[j]**

if j<=0 0

if j>0 min( c[i-1] + table[I,j] where 1<=i<=j )

When we relating above recursive function to the above example c[4] will give the minimum total error of arranging words from 1 to 4.

# Optimal Substructure Property.

Lowest error that could be for the nth word of text is given by c[n].To find c[n] we have to find the solutions to the sub problems such as c[1],c[2],c[3]…….c[n-1].If we come up with the solutions to those sub problems locally we will be able to find the global solution for the c[n] which will give the lowest total error up to the nth word.

# Overlapping Sub problems.

If we draw the recursion tree for calculating the error of a line having 5 words it will be as follows :-

**C[5]**

**C[3]**

**C[2]**

**C[1]**

**C[3]**

**C[2]**

**C[1]**

**C[2]**

**C[1]**

**C[5]**

**C[4]**

**C[3]**

**C[2]**

**C[1]**

When we look at the above diagram we can see that the same problem appears in different places. For example the solution of sub problem c(2) is used by c(3), C(4) and so on. Therefore we conclude that the above recursion algorithm have so many overlapping sub problems.

# Recursive Algorithm implementation

Pseudo code to calculate the total lowest error for the above recursion algorithm is given below.

wordWrap(wordNumber,table)

{

If(number=0)

Return(0)

Else

{

Return(min(wordNumber-1,table)+table[i][j])

}

}

# Recursive Algorithm to find the minimum cost of error

If we consider the no of words in the text is “n” then we can define the cost of error as fallows

Cost\_of\_error =

In order to write a recursive solution we need to define a sub problem. We can take a line at a time and guess, what is the best word that fits in to the next line’s first word (jth word) where the remaining next “n-j” words becomes the sub problem.

# Generalizing the Algorithm for Text Justifying problem to make Algorithm efficient

In general recursive algorithm for a problem is slow and run in exponential time. So we must use Dynamic programming techniques to ensure the efficacy of the algorithm we use to solve a problem. The above recursive solution given to the text justifying problem runs on exponential running time. We can use Dynamic programming techniques such as Memorization or Bottom up techniques to reduce the running time of the text justifying algorithm to ⃝ (). Basically both Memorization and Bottom up algorithms are not much deferent with respect to the running time and the computational steps which they have to perform when executing the algorithms. For almost all of the recursive algorithms we can use Memorization and Bottom up techniques and we can use whatever the technique we like

# Analysis of the Running time

Although the most of the recursive algorithms produces exponential time algorithms by carefully handling the redundancies of the recursions we can reduce the running time of the recursive algorithms into polynomial time. If we use a greedy approach to the text justifying problem we can obtain we can solve the problem in ⃝ (n) running time. But the solution we obtain from the greedy approach is not the optimal solution. If we use recursive method without using the dynamic programming techniques we can propose algorithm that runs in exponential time. But by using dynamic programming technique we can achieve a most optimal solution which runs in ⃝ () running time.

* Without dynamic programming techniques.

Let’s assume the no of words in the text is “n”

We will have to consider all the partitions of the words to decide which combination fits bets to the lines. That is we will have to consider sub problems. Therefore the running time is

* With dynamic programming techniques

Running time = NO-of sub problems \* time/sub problem

If we ignore constant time computational steps

Time/sub problem = ⃝ (n)

The maximum no of sub problems can be there is “n” where “n” is the no of words in the text

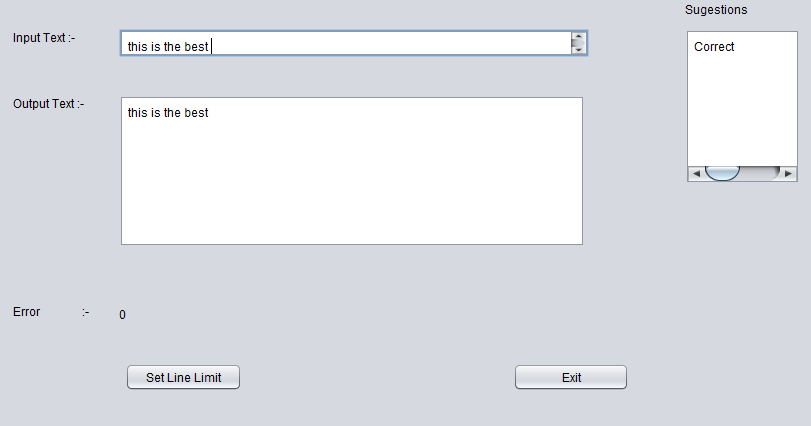
There for

No of sub problems = n

There for the running time of the algorithm is = ⃝ ()

# Spell Checker

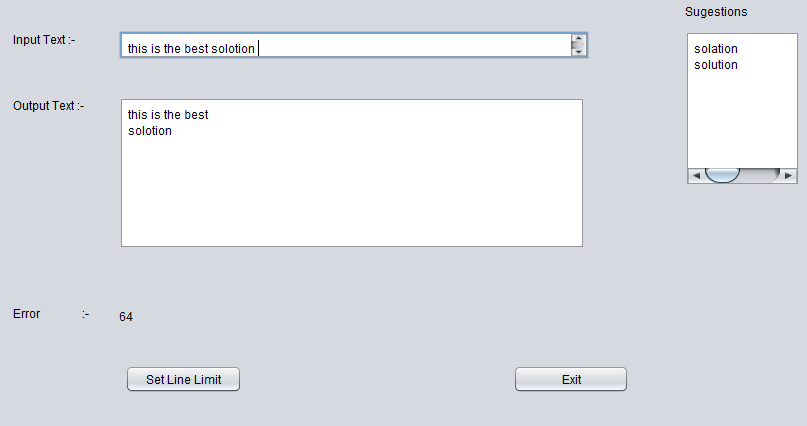
In this part we use edit distance algorithm to solve this problem. In here we have to suggest correct words to incorrect words. First we give chance to user to type any word without checking spell. Then if that word is correct suggestion text area display “correct “.if not display suggestion words



Ref 1

Ref 1 show the when correct word type what will display the suggestion text box. But all suggestions and checking doing only user put space after word. Until user putting space after word program don’t check correctness of the word and suggestions for the wrong words.

Ref 2 show how the program action if given word is incorrect. After typing and put space, program check creativity about word and if that word is incorrect then display suggestions in the suggestions text box.



Ref 2

We create a Dictionary class for check correctness about word. We use hash set for store all word for check correctness.

public class Dictionary {

HashSet dic=new HashSet();

Then we read given library text file and create hash table including all words in the text file.

File inputFile = new File("ass.txt");

Scanner inFile = null;

inFile = new Scanner(inputFile);

String theWord;

while(inFile.hasNext()) {

theWord = inFile.nextLine();

dic.add(theWord); }

Then check correctness about words using these hash table. If word correct return true, else return false.

public boolean check(String word)

{

return(dic.contains(word));

}

We use hash table for reduce time complexity of computing correctness about words. Hash table method is get low time than reading and matching with each words of text file.

If any word incorrect then using edit distance algorithm program me find best similar words for incorrect word.

Edit Distance algorithm

a = a.toLowerCase()

b = b.toLowerCase();

int [] costs = new int [b.length() + 1];

for (int j = 0; j < costs.length; j++)

costs[j] = j;

for (int i = 1; i <= a.length(); i++) {

costs[0] = i;

int nw = i - 1;

for (int j = 1; j <= b.length(); j++) {

int cj = Math.min(1 + Math.min(costs[j], costs[j - 1]), a.charAt(i - 1) == b.charAt(j - 1) ? nw : nw + 1);

nw = costs[j];

costs[j] = cj;

}

}

return costs[b.length()];

Compute cost for all words in the dictionary .then return words have minimum cost. So Transposition errors are common in written text. A transposition can be treated as a deletion plus an insertion, but a simple variation on the algorithm can treat a transposition as a single point mutation.

# Compute edit distance

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | M | O | N | K | E | Y |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| M | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| O | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| N | 3 | 2 | 1 | 0 | 1 | 2 | 3 |
| E | 4 | 3 | 2 | 1 | 1 | 1 | 2 |
| y | 5 | 4 | 3 | 2 | 2 | 2 | 1 |

Edit Distance =1

# Complexity of the edit distance algorithm

The time-complexity of the algorithm is O(|s1|\*|s2|), i.e. O(n2) if the lengths of both strings is about `n'. The space-complexity is also O(n2) if the whole of the matrix is kept for a trace-back to find an optimal alignment. If only the value of the edit distance is needed, only two rows of the matrix need be allocated; they can be "recycled", and the space complexity is then O(|s1|), i.e. O(n)

T(I,j) : minimum cost to transform A[1….j] in to B[1….j]

C d +T(i-1,j)

T(I,j) = min T(i,j-1) + Ci

T(i-1,j-1) if A[i]=B[j]

T(i-1,j-1) + CR if A[i]≠B[j]

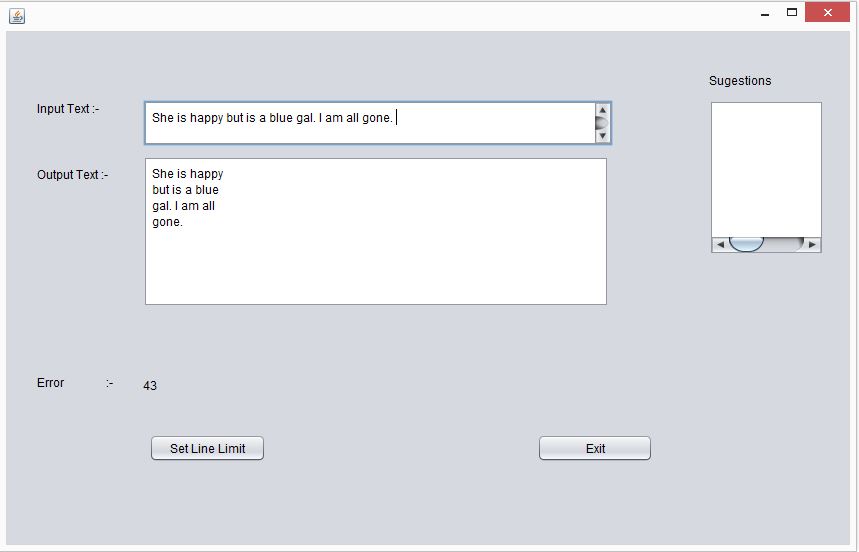
0(n,m) running time

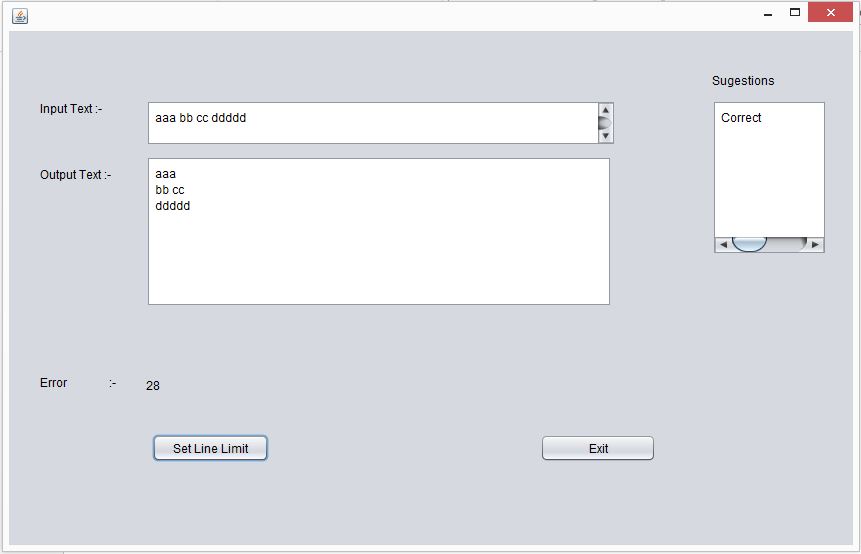
T(n.m) optimal Solution

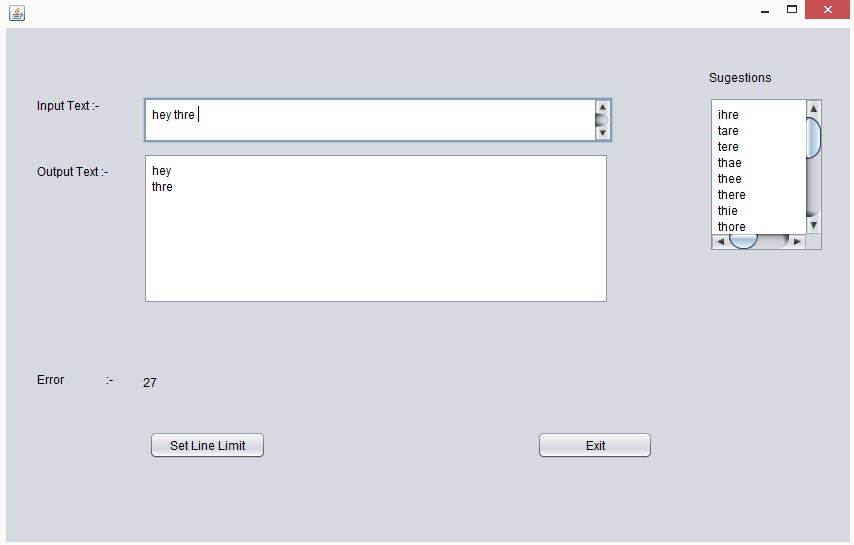
# Variations

The costs of the point mutations can be varied to be numbers other than 0 or 1. Linear gap-costs are sometimes used where a run of insertions (or deletions) of length `x', has a cost of `ax+b', for constants `a' and `b'. If b>0, this penalizes numerous short runs of insertions and deletions.

# Test Cases







# Compiling Instructions

There are 6 java files and main.java is the file that should be compiled to run the above program.

“ass.txt” is the file that include words in the dictionary and that file should be there at the compiling directory.

# References

Dynamic Programming | Set 19 (Word Wrap Problem) | GeeksforGeeks. 2014. Dynamic Programming | Set 19 (Word Wrap Problem) | GeeksforGeeks. [ONLINE] Available at:<http://www.geeksforgeeks.org/dynamic-programming-set-18-word-wrap/>. [Accessed 23 June 2014].