Breaking Paragraphs into lines
an implementation for Emacs

Kundan Kumar
Rishabh Nigam
Satyadev Nandakumar
Dept. of Computer Science and Engineering, IIT Kanpur
{kundankr,rishabhn,satyadev}@cse.iitk.ac.in

25th June 2012

Abstract

The problem of text justification involves the aspect of putting line-breaks at appropriate points. In this project we try to do text justification for monospace character. The Knuth's algorithm for typesetting in Tex uses different sizes of characters giving it more freedom and better text justification. So our version of the algorithm uses some of Knuth's principles and tries to build upon it. We try to build it within emacs using elisp. One of the important aspect of justification is hyphenation. We used the algorithm as described in [2] from Frank Liang's thesis for hyphenation of words.

About the Algorithm

- **Hyphenation**
  We used [2] for the algrorithm. Liang's thesis describes the algorithm in terms of some generated patterns. These patterns store the likelihood of a possible break point at a point. For e.g. some pattern like ".a3n4t." this means that there is a good hyphenation point after a(value 3 being odd) and a bad hyphenation point after n(value 4 being

odd). So the approach is look after all these patterns for all possible substrings of the word and see if a possible point is a good hyphenation point.

- **Breaking paragraphs into lines**
  For this we used a dynamic programming approach similar to that used by Knuth in [1]. We needed to calculate the penalty associated with the rest of the paragraph after putting a line break after the jth character of ith word. This involved looking at the min(penalty for the line + penalty at the next breakpoint). The time complexity of this algorithm happens to be less than O(n*n*k*k + n*h) where k is the length of a word and n is the number of words and h is the hyphenation time of the word.

Our Approach

We considered the penalty to comprise of three quantities. The first one being the number of spaces to be filled. The second the hyphenation penalty (currently set to 0) and the third the penalty of filling spaces between words (currently set to 10 times number of extra spaces). With these three penlties we tried to minimize the sum of penalty over the entire paragraph. This used a dynamic programming approach as mentioned.

About the Code

There are a lot of defined functions which have different functions. So here I describe what are the various functions and what is there purpose

- isnum(c)
  check if a character is num or not
- extractchar(s)
  This function takes input a string and puts alphabets into list chars and number in the list points.Convert the a pattern like 'a1bc3d4' into a list of chars (a b c d) and a list of points (1 0 3 4).
- listtostring(L)
  This function converts a list L in string and returns the string

- binarysearch (pat binary_i binary_j _arr)
  It search a specific pattern from sorted array of patterns(strings). If pattern is found it return index of pattern in array. If not found -1 is returned. binary_i and binary_j are the initial and final indexes of array.

- runoverstring(runover-str)
  It generates all the possible substrings of the given string and binary searches them in the pattern. and if a string is found it uses extract char and updates pointsstring with max(pointsstring,point)

- exceptions (exceptions-string)
  It looks after some exceptions that the Liang's thesis mentions.

- hyphenate(hystr)
  It returns the hyphenated string and stores the corresponding values in pointstring.If a value in pointsstring is odd it is a possible break point. so if pointstring[i] is odd u can break the strin hystr after i-2 e.g. for "elephant" pointsstring[4] is odd so u can hyphenate as "ele-phant"

- loadinit()
  This function initializes four 2d arrays(dpa,dpb,dpc,dpd) of size 200,15. dpb stores for the ith word if u can break at its jth character. dpa stores what is the penalty associated with breaking after ith word's jth character.dpc,dpd stores if I break at ith word's jth character where next should I break dpc stores the word index and dpd the character index.

- file : pattern.el
  loading this file initializes arr to the list of patterns and arr1 to the list of exceptions.

- bake(i,j)
  this function reads byte from the buffer using get-byte and finally returns the string of buffer from i to j inclusive.

- store(i,j,linewidth)
  This functions creates three global variables. First Storelist - it is the list of all words from the buffer. Second listlen - its ith index corresponds to the length after (i-1)th word in storelist. Third liststretch -its ith index corresponds to the possible stretch after (i-1)th word in storelist.

- givehyphenpoints(hystr)
  This returns a list of indexes where we could possibly hyphenate hystr.

- penalty(x,y)
  here x and y represent stretch available and spaces to put. (penalty 5 3) = 3 and penalty(3,5)= 23
- filldp(i,j,linewidth)
  This is the most important function. This function implements the dynamic programming approach for breaking paragraphs into line. After this function is executed we get 4 global arrays(dpa,dpb,dpc,dpd) filled with values. The total penalty associated with the paragraph can be seen from (elt(elt dpa 0) 0).
- putnewline(i,j,linewidth)
  This function uses dpc and dpd to find where to put new line in the paragraph. It returns a string which contains the paragraph along with appropriately put new lines.
- runoverpara(i,j,linewidth)
  This function looks for 2 or more newline characters to separate the text into paragraphs. Where it finds two newline characters it calls the putnewline function to print the output of the paragraph into the message buffer. After calling this function you can find the output on the message buffer. It additionally puts the starting and ending time of the execution of this function. The output ends with "endoffile"

Some constraints

- Here each word is assumed to have less than 40 characters and the total words in a paragraph should not exceed 500. There is not limit on the number of the words in the entire text.

Bibliography

[1] Texbook by Knuth [101-118]

[2] Word Hy-phen-a-tion by Com-put-er by Franklin Mark Liang, Department of Computer Science, Stanford University PHD thesis
http://www.tug.org/docs/liang/

[3] http://defoe.sourceforge.net/folio/knuth-plass.html