CS345
Lab 3
Winter 2006

Word Wrapping

Say your American literature class is reading Huckleberry Finn. Everyone else in the class goes to the bookstore to buy used, and heavily highlighted, paperback copies for $12 each. You say to yourself, 'I would really prefer not to spend $12 for a book I'll just resell for $2, and anyway shouldn't that kind of book be free? It was written over 76 years ago and therefore its copyright has expired. Maybe I could find an electronic copy and just read it on my Palm.' You quickly find `http://www.gutenberg.org/`, search for Mr. Twain, and download yourself a plain text version that is 'Free as in Speech' and 'Free as in Beer'. Now to start reading. How? Being a computer scientist you hack out a quick text reader application. But when you start reading you notice the text file has *hard returns* at the end of lines and when viewed on a small screen looks like this:

```
HUCKLEBERRY FINN

Scene:  The Mississippi Valley Time:  Forty to fifty years ago

CHAPTER I.

YOU don't know about me without you have read a
book by the name of The
Adventures of Tom Sawyer; but that ain't no matter.
  That book was made
by Mr. Mark Twain, and he told the truth, mainly.
There was things which
he stretched, but mainly he told the truth.  That is
nothing.  I never
seen anybody but lied one time or another, without
it was Aunt Polly, or
the widow, or maybe Mary.  Aunt Polly--Tom's Aunt
Polly, she is--and
Mary, and the Widow Douglas is all told about in that
book, which is
mostly a true book, with some stretchers, as I said
before.
```

That certainly isn't going to work. You want the text to look natural, with no hard returns. Specifically you'd like it to be able to flow and look nice:

```
YOU don't know about me without you have read a book
by the name of The Adventures of Tom Sawyer; but
that ain't no matter. That book was made by Mr. Mark
```

```
Twain, and he told the truth, mainly. There was things
which he stretched, but mainly he told the truth.
That is nothing. I never seen anybody but lied one
time or another, without it was Aunt Polly, or the
widow, or maybe Mary. Aunt Polly--Tom's Aunt Polly,
she is--and Mary, and the Widow Douglas is all told
about in that book, which is mostly a true book, with
some stretchers, as I said before.
```

For some reason the API for your PDA doesn't have word wrapping capabilities, so you decide to write your own. Since you know a little about algorithms you suspect there is a nice efficient Greedy Algorithm that will suffice. You quickly write out an algorithm that goes something like this.

1. Read lines from the text file and tokenize into words, recording the lengths of each word as you go.
2. Begin writing lines of text to the screen one word at a time as soon as a word will not fit, put in a newline
3. Proceed one line at a time till done.

In fact the Java version looks like Listing 1. In that code, `svec` is a vector of strings of the original text, and `C` is the column number to wrap to.

Dynamic Programming Algorithm

For some reason you just aren't satisfied with your greedy solution. There exists an optimal solution to this problem and you're pretty sure your algorithm doesn't find it. (Greedy solutions are commonly used as highly efficient *approximate* solutions.) What happens if there is a longer word that leaves a bunch of spaces at the end of a line – a situation that could have been avoided by breaking a previous line earlier by one or two words. Certainly there is a way of *minimizing* the number of spaces left at the ends of lines.

In fact there is, but the problem is at first a very daunting one. Deciding where to break a line near the top of the file is affected by where breaks occur further down the file. Your algorithm must be able to *look ahead* and see where better line breaks could occur. If I propose breaking between words 12 and 13, this depends on whether lines are then broken at 14, 15, 16, or in fact any other location. There must be some optimal substructure here, something to go on. Let's put the problem in mathematical terms first.

Assume the document has N words, labeled $1, 2, \ldots, N$. These words have lengths $l_1, l_2, \ldots, l_N$. The number of lines in the resulting document is determined by where we choose to break lines. Let's say we allow lines to have no more than C letters in them (having exactly C letters is fine, and we don't need a space at the end of a line). But we must have exactly one space between words. This includes one space between sentences rather than two. We can represent the **optimal solution** to the entire problem as minimum number of spaces $= M[1, N]$, meaning the solution

for placing words 1 through N on some number of lines, which is not yet known. What is this optimal solution composed of? Whatever it is, it must be something like this. The *first* line break has to occur somewhere, let's say it is after word j. If this is the case then our overall problem can be broken down to say that the overall solution, $M[1, N] = numSpacesLeft_{1,j} + M[j+1, N]\}$. That is, the number of spaces left over on *this* line (containing words 1 through j, plus the number of spaces left over from solving the *subproblem* of line breaking for words $j+1$ through N. The problem is that we don't know where j is. So we must iterate over all possible j values and choose the smallest. Let's say that $R_{1,j} = numSpacesLeft_{1,j}$ for easier writing. Then our overall solution is

$$M[1, N] = Minimum\{R_{1,1} + M[2, N], R_{1,2} + M[3, N], R_{1,3} + M[4, N], \ldots, R_{1,N-1} + M[N, N]\}$$

But what is $M[2, N]$ etc.? We don't know yet because this solution is written recursively. And you should also be able to easily see the overlapping subproblems here. The solution to 1 needs the $N-1$ subproblem solutions for $2, 3, 4, 5, \ldots$ while the solution to 2 needs the $N-2$ subproblem solutions for $3, 4, 5, 6, \ldots$. That's a whole lot of overlap by the time we get to N. **Let's not even think about doing this one recursively.**

What about the base case. We must be able to solve at least $M[N, N]$ without recursion. Yes, we'll consider that the last line of the file does not contribute any left over spaces to the problem. Effectively we don't count any spaces left over if they're on the last line. So certainly $M[N, N] = 0$ and probably $M[N-1, N] = 0$. The question is how far back are the zeros. To find that, you'll need to work backwards from the end and figure out how many of the last words will fit on one line.

Formally then, optimal solution to this problem can be represented by the following recursion relation.

$$M[i, N] = \begin{cases} Minimum\{R_{i,j} + M[j+1, N]\}_{i \le j \le N} & 1 \le i \le N \\ 0 & \text{if words } i \text{ through } N \text{ fit on one line} \end{cases}$$

The number of spaces at the end of a line is just

$$R_{i,j} = C - (j - i + \sum_{k=i}^{j} l_k)$$

which accounts for spaces between words i through j and none at the beginning or end of a line.

Your job is to develop a bottom-up iterative, or memoized solution to this problem. For complete credit you'll need to solve the problem and report back the minimum number of spaces (over all lines) at the end of lines in the resulting file. For extra credit, keep track of where the line breaks occur and actually word wrap the file, outputting it as shown. I've provided starting Java code as well as some sample files. I'll post a table of known solutions for some of the sample files.

For the bottom up solution you will find that the runtime efficiency is $O(N^3)$.

Listing 1: Greedy Version

```
private static int wrapGreedy( )
{
   try{out = new PrintWriter( outputFilename );}
   catch( FileNotFoundException e )
   {
      System.out.println("Cannot create or open " + outputFilename +
                  " for writing.  Using standard output instead." );
      out = new PrintWriter(System.out);
   }
   ListIterator<String> it = svec.listIterator();
   it.next();
   int col = 1;
   int spacesRemaining = 0;
   while( it.hasNext() )
   {
      String str = it.next();
      int len = str.length();
      if( col == 1 )
      {// Write first word on a line
         out.print(str);
         col += len;
      }// See if we need to wrap to the next line
      else if( (col + len) >= C )
      {
         out.println();
         spacesRemaining += ( C - col ) + 1;
         col = 1;
         it.previous();
      }
      else
      {// Typical case of printing the next word on the same line
         out.print(" ");
         out.print(str);
         col += (len + 1);
      }
   }
   out.println();out.flush();out.close();
   return spacesRemaining;
}
```