EE3980 Algorithms

HW8 Printing Paragraphs

104061212 馮立俞

2018/5/6

Introduction

Given a text file with each paragraph separated by line break('\n') character, print the file left aligned, and the length of printed lines should not surpass a constant N. However, while moving on to the next printing line, sometimes we'll leave some blank spaces. Except for the last line of a paragraph, where we use a line break character to terminate the paragraph. Therefore, no blank spaces are needed for the last line of a it. The goal of this assignment is to print the file using minimum spaces for each paragraph, and with greedy method and dynamic programming.

Approach

Suppose there's n words in the paragraph. For a line that prints from i-th to j-th word in the paragraph. The extra space needed for such line is N minus total word length from i to j, and minus the spaces between words. i.e.

$$\text{Extra Spaces of line} = \text{N} - \sum_{k=i}^{k=j} wordlength_k - (j - i)$$

And the total spaces used for a paragraph is just adding the result of each line in the paragraph.

Greedy Method

A greedy method approach needs a benchmark for all legal moves in current step. Then, to optimize the overall result, a greedy method takes the step with highest benchmark result. The taken step should reduce the problem. After several steps the problem is solved, the output is the combination of steps taken.

In this problem, the benchmark of printing a line is the number of spaces required the at the end of line. To optimize we'll print as much as we can in a line, leaving minimum spaces each line. The time complexity is $O(n)$ since we only need to evaluate each word one time.

```
1.  Algorithm greedy(paragraph) {
2.      while (not finished printing paragraph)
3.          while (the next word can fit in current line) print_word;
4.      print("\n")
5.  }
```

Dynamic Programming

Another way of viewing this problem is to see it as recursion of printing.

$$Space_{0\sim n} = \min(\ Space_{current\ line} + Space_{rest\ of\ paragraph})$$

For printing a paragraph, the result is equivalent to the combination of current line result and the rest of the paragraph result. We can then treat the rest of the paragraph as a new paragraph. Such recursion ends as the residue can be printed in a single line. This is somewhat like the rod cutting problem in textbook. And we've

known from rod cutting problem that we could record computation result in arrays to avoid repetitive operations. In this problem we minimize the cost rather than maximize the profit. So an array recording the minimum amount of total spaces required if the paragraph starts from word[i], Space[ ], is needed. As for the solution, instead of indicating the length to cut, we need information about to which word should we stop printing if the line starts with word[i]; therefore another matrix which records Solution, Sol[ ], is used.

```
1.  Algorithm DP(paragraph, Sol[], Spaces[], i) {
2.      if (Spaces[i] != 0) return Space[i];  //have calculated result before
3.      for (k = i; line can fit from word i to word k; k++)
4.            temp = (space left when print from word i to word k)
5.                  + DP(paragraph, Sol[], Spaces[], k);
6.      if (temp < Spaces[i]) {
7.          Sol[i] = k;
8.          Spaces[i] = temp;
9.      }
10.     return Spaces[i];
11. }
```

We can then print the paragraph using constructed Sol[ ]

```
1.  Algorithm DPPrint(paragraph, Nwords, Sol[]) {
2.      for (i = 0, j = Sol[0]; i < Nwords;){
            print from word i to word j
            i = j + 1;
3.          j = Sol[i];
4.      }
5.  }
```

The time complexity is not clear due to recursion call. Yet it's obvious that it's under-bounded by $O(n)$ for filling Space[ ], Sol[ ], and printing result, because they're all of size n. So dynamic programming would theoretically take more time due to larger coefficient in time complexity.

The space complexity for both method are $O(n)$. Though coefficient of dynamic Programming is greater.

Results and analysis

Below is a snippet of the output of greedy method with N being 50, the numbers on the right are the additional spaces added in that line.

```
Stewart and his team put out several issues of The|0
Whole Earth Catalog, and then when it had run its |1
course, they put out a final issue. It was the    |4
mid-1970s, and I was your age. On the back cover  |2
of their final issue was a photograph of an early |1
morning country road, the kind you might find     |5
yourself hitchhiking on if you were so            |12
adventurous. Beneath it were the words: "Stay     |5
Hungry. Stay Foolish." It was their farewell      |6
message as they signed off. Stay Hungry. Stay     |5
Foolish. And I have always wished that for myself.|0
And now, as you graduate to begin anew, I wish    |4
that for you.|0
----Space in this paragraph: 45

Stay Hungry. Stay Foolish.|0
----Space in this paragraph: 0

Thank you all very much.|0
```

And Below is the output from DP function.

```
Stewart and his team put                          |26
out several issues of The Whole Earth Catalog, and|0
then when it had run its course, they put out a   |3
final issue. It was the mid-1970s, and I was your |1
age. On the back cover of their final issue was   |3
a photograph of an early morning country road, the|0
kind you might find yourself hitchhiking on if you|0
were so adventurous. Beneath it were the words:   |3
"Stay Hungry. Stay Foolish." It was their farewell|0
message as they signed off. Stay Hungry. Stay     |5
Foolish. And I have always wished that for myself.|0
And now, as you graduate to begin anew, I wish    |4
that for you.                                     |0
----Total spaces in this paragraph is 45

Stay Hungry. Stay Foolish.                        |0
----Total spaces in this paragraph is 0

Thank you all very much.                          |0
```

We can see that though some difference exists, the overall spaces used per paragraph are the same for two approaches. The output of dynamic programming approach is leaves weird long blank (as indicated above with red line).

Observations and Conclusion

From the given test case, greedy method seems to give the optimal solution as dynamic programming can do. Yet result from greedy method looks more uniform between lines. Plus, greedy method has less time complexity and requires easier coding effort by nature. It seems greedy method should be favored over dynamic programming when we encounter this problem.