Text Justification

Given an array of words and a length *L*, format the text such that each line has exactly *L* characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces `' '` when necessary so that each line has exactly *L* characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

For example,
words: `["This", "is", "an", "example", "of", "text", "justification."]`
L: `16`.

Return the formatted lines as:

```
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
```

Note: Each word is guaranteed not to exceed *L* in length.

click to show corner cases.

Corner Cases:

- A line other than the last line might contain only one word. What should you do in this case?
  In this case, that line should be left-justified.

Solution 1

```
vector<string> fullJustify(vector<string> &words, int L) {
    vector<string> res;
    for(int i = 0, k, l; i < words.size(); i += k) {
        for(k = l = 0; i + k < words.size() and l + words[i+k].size() <= L - k; k
++) {
            l += words[i+k].size();
        }
        string tmp = words[i];
        for(int j = 0; j < k - 1; j++) {
            if(i + k >= words.size()) tmp += " ";
            else tmp += string((L - l) / (k - 1) + (j < (L - l) % (k - 1)), ' ');
            tmp += words[i+j+1];
        }
        tmp += string(L - tmp.size(), ' ');
        res.push_back(tmp);
    }
    return res;
}
```

For each line, I first figure out which words can fit in. According to the code, these words are words[i] through words[i+k-1]. Then spaces are added between the words. The trick here is to use mod operation to manage the spaces that can't be evenly distrubuted: the first (L-l) % (k-1) gaps acquire an additional space.

written by qddpx original link here

Solution 2

In some of the texts that I have been able to find I see that this problem admits a dynamic programming solution that is superior to greedy solutions. (MSWord vs LATEX). I think, that to solve this question specifically (meaning something that OJ accepts) requires a greedy solution.

As far as I understand the "idea" of text justification is not to distribute spaces as evenly as possible within all the words of an individual line; But instead lower the overall cost of the way you justify text, which means that even though you may have some lines that have uneven spaces between words than others, but this lowers the overall cost of a justification in other lines.

In this question's description the correct answer is described as a very specific way to do text justification that seems to be not what the superior solution is.

Do you think its right to actually post this question as an exercise at all? What does this question aim to teach as far as good text justification algorithms are concerned?

written by graham2181 original link here

Solution 3

```
class Solution {
public:
    vector<string> fullJustify(vector<string> &words, int L) {
        vector<string> ans;
        int begin = 0;
        while (begin < words.size()) {
            int last = begin;
            int linesize = words[begin++].size();
            while (begin < words.size() && linesize + 1 + words[begin].size() <=
L) {
                linesize += 1 + words[begin].size();
                begin++;
            }

            int spaces = 1, extra = 0;
            if (begin < words.size() && begin != last + 1) {
                spaces = (L - linesize) / (begin - last - 1) + 1;
                extra = (L - linesize) % (begin - last - 1);
            }

            ans.push_back(words[last++]);
            while (extra--) {
                ans.back().append(spaces+1, ' ');
                ans.back().append(words[last++]);
            }
            while (last < begin) {
                ans.back().append(spaces, ' ');
                ans.back().append(words[last++]);
            }
            ans.back().append(L-ans.back().size(), ' ');
        }

        return ans;
    }
};
```

written by xuewuxiao original link here