Homework 7 Solutions

Problem 1:

You were not required to develop software. So, this solution exceeds what is expected of you.

The sub-problems for the dynamic programming approach are the layout of a paragraph consisting of a suffix of the array of words. The array of words is text $= [w_0, w_1, w_2, \ldots, w_{n-1}]$.

We define a function that calculates the space used by setting words $[w_i, w_{i+1}, \ldots, w_{j-1}]$ on a single line. Because we want to use memoization we need to get around the requirement that keys for a dictionary have to be immutable. Our solution is to use the text, — the array of words — as globals. This is ***bad programming practice*** and the best way around this is to use classes.

The space function is simple (using list comprehension, which you need to know for a Python Code interview):

```
@cache
def space(i,j):
    global text
    return sum([len(a) for a in text[i:j]]) + j-i-1
```

The **subproblems** we are going to use is setting the text starting with word w_j, i.e. text[j :].

We can use a bottom-up approach, where we find the minimum costs b_k to type-set text[k :]. We start with the last words. As long as text[k :] fits into the last line, $b_k = 0$. Otherwise, we break the text into two parts, text[$k : k + j$] and text[$k + j$:], but only if text[$k : k + j$] fits into a single line. The costs of setting text[k :] is the costs of setting text[$k : k + j$], which is the combined length of the words plus $j - 1$, plus the costs of type-setting text[$k + j$:], which we already determined as b_{k+j}. We chose the minimum as the value of b_k.

```
best = {}

for j in range(len(text), 0, -1):
    if(space(j,len(text)+1)) < line_length:
        best[j] = 0
    else:
        possibilities = []
        for k in range(1,len(text)-j):
            if space(j, j+k) > line_length:
                break
            else:
                possibilities.append( (space(j, j+k)- line_length)**2
                                      + best[j+k])
        best[j]=min(possibilities)
```

Our process proceeds backwards. With a bit more work (because the last line is different), we could also try to work forwards. Finally, a more efficient implementation would not use a list of possible values and then take the minimum.