CS 5112 HW4
Dynamic Programming

In this assignment, you will come up with and implement two dynamic programming algorithms.

1 Problems

1.1 Respacing

The respacing problem is to put spaces back into a string that has lost them, given a dictionary. For example, given the string "itwasthebestoftimes" and an English dictionary, we would like to reconstruct the original sentence: "it was the best of times".

1.2 Diffing with costs

We saw in lecture that 'diffing' two strings can be accomplished with dynamic programming. In this problem we will solve a more general version. We would like to align two strings, `s` and `t`, in a way to produce a *minimum-cost* alignment.

To produce an alignment on two strings `s` and `t`, we insert the special character '-' some number of times into each string to produce `align_s` and `align_t` so that:

- `align_s` and `align_t` have the same length, and
- There is no `i` such that `align_s[i]` and `align_t[i]` are both '-'.

The cost of an alignment is given by a cost function, which we will call `cost`. The cost of an alignment is the sum over all `i` of `cost(align_s[i], align_t[i])`. The cost of aligning a letter with itself is always 0.

For example, given this cost table:

		Letter from s			
		a	b	c	-
Letter from t	a	0	1	5	3
	b	5	0	5	3
	c	3	4	0	3
	-	2	2	1	·

The best alignment of `acb` and `baa` is given by:

```
align_s:  -acb
align_t:  ba-a
```

With a total cost of 5.

This problem arises, for example, in DNA sequencing; given two strands of DNA, there are many sequences of mutations (insertions, deletions, etc.) which would have transformed one to the other; we would like to find the most probable. We know that certain mutations are more likely than other, and these probabilities are reflected in the cost table.

2 Dynamic Programming Framework

As usual, we have provided a skeleton of code for you, in `dynamic_programming.py`, `diffing.py`, and `respacing.py`. It is designed to let you just think about the algorithm, while also ensuring that the algorithm you design is in fact a dynamic program. You will write some code to fill in a dynamic programming table, and then to compute the final answer from this table.

The table for respacing is $(N+1)\times(N+1)$, where N is the length of the word to be respaced. Each cell may contain a single boolean and a single integer.

The table for diffing is $|s|+1\times|t|+1$, where s and t are the strings to be diffed. Each cell may contain a single integer and two characters (a character is a length-1 string).

Your job for each problem is to fill in three functions.

2.1 Respacing

fill_cell . You should fill in `fill_cell(T, i, j, string, is_word)`, which will be called once per cell. T is the table as filled in so far; use its `get` method to find these values. Your function should not mutate T directly, but rather return a `RespaceTableCell`. `is_word` is a function which encodes a dictionary: given a string, it returns true or false, according to whether the string is a word. Each call to `fill_cell` should call it no more than once. `fill_cell` should run in $O(N)$ time.

cell_ordering . You should fill in `cell_ordering(N)`, which will be called once. N is the length of the string to be respaced. `cell_ordering` should return a list of two-tuples; these are the coordinates of cells in the order you would like `fill_cell` to be called. No cell should be repeated, but you do not need to use all cells.

respace_from_table . You should fill in `respace_from_table(s, table)`. `s` is the string being respaced, and `table` is the table that is already filled in. `respace_from_table` should run in $O(N^2)$ time, and make no calls to `is_word`.

An example for putting these all together with our dynamic programming framework is given at the bottom of `respacing.py`.

2.2 Diffing

fill_cell . You should fill in `fill_cell(table, i, j, s, t, cost)`, which will be called once per cell. `table` is the table as filled in so far; use its `get` method to find these values. Your function should not mutate the table directly, but rather return a `DiffingTableCell`. `cost` is a function which encodes a cost function: it takes two characters and returns their cost. Each call to `fill_cell` should call it no more than four times. `fill_cell` should run in $O(1)$ time.

cell_ordering . You should fill in `cell_ordering(n,m)`, which will be called once. n and m are the lengths of the strings s and t. `cell_ordering` should return a list of two-tuples; these are the coordinates of cells in the order you would like `fill_cell` to be called. No cell should be repeated, but you do not need to use all cells.

diff_from_table . You should fill in `diff_from_table(s,t, table)`. `s` is the string being respaced, and `table` is the table that is already filled in. `diff_from_table` should run in $O(n + m)$ time, and make no calls to `cost`.

An example for putting these all together with our dynamic programming framework is given at the bottom of `diffing.py`.

3 Testing and logistics

3.1 Outside resources

This homework is different from previous assignments – instead of implementing a well-known algorithm, we are asking you to come up with new algorithms using dynamic programming. We expect that finding an appropriate optimal substructure will be the bulk of the work – the code is fairly short. So, you should not use or seek any resources that describe how to solve these problems, even in pseudocode.

3.2 Generated tests

As in HW3, we have provided a list of generated tests; you can run them by using `test_diffing.py` and `test_respace.py`. We do not guarantee that these tests are exhuastive; we suggest you write your own, smaller tests as well.

3.3 Python etc.

You can assume that all strings (words in respacing, input to respacing, and strings in diffing) consist of lowercase letters a,b,c,...z, have length at least 1, and are not None.

Please use python2.

Please do not use any additional imports.

You will need to submit your `diffing.py` and `respacing.py` on CMS. Please be careful not to swap them by accident!

3.4 Groups

You may work in groups of up to 3 on this assignment.