Problem 3: RNA Folding (30 points; individual only)

Copied from: https://www.cs.hmc.edu/twiki/bin/view/CS5/BlackRNA on 3/22/2017

This is the individual-only problem for this week. Please put your solution in a file called `hw3pr3.py`. There is are also optional bonus problems that you can place in this same file.

Part 1: RNA Folding...

RNA is a single-stranded polymer of nucleotides. RNA folds to form complex structures, which have a variety of functions in the cell. For our purposes, RNA is a string made up of a sequence of letters from the alphabet 'A', 'U', 'C', and 'G'.

- RNA folds in a way that, in general, puts it into the most stable, low-energy state. A good first-order approximation, and the one that we will use here, is to assume that RNA folds in a way that maximizes the total number of paired bases.
- Any 'A' (adenine) on the RNA strand can pair with any 'U' (uracil) on the same strand, even if they are adjacent to each-other. Similarly, any 'G' (guanine) can pair with any 'C' (cytosine). When a base pairs with another, it cannot then pair a second time—it becomes inert.
- When two nucleotides pair to each other, they pinch together and any bases in between them are scrunched up into a loop. Within this loop, pairing can take place. Outside of the loop on any continuous strand, pairing can also take place. Pairing *cannot* take place between a nucleotide inside a loop and one outside the loop! (This is called the "no pseudoknots" assumption. In fact, pseudoknots do occur in some cases in folded RNA, but most RNA folding algorithms exclude them from consideration because they are believed to be best examined in the tertiary structure of RNA rather than at this secondary level.)

Write a function `fold(RNA)` that accepts an RNA nucleotide sequence—that is a string made up of the letters 'A', 'U', 'G', and 'C'—as its sole argument. The function then returns a single number, which is the maximum number of matches possible in a valid folding of this RNA string.

You'll need to use `map` and the *x* `if` *condition* `else` *y* idea that we saw in class for use with `lambda`.

Your `fold` function should be about 4 or 5 lines of Python code (not including the `def fold` line and the docstring).

Test your function and make sure it produces the correct result on the following strings:

```
In [1]: fold("ACCCCCU")
Out[1]: 1

In [2]: fold("ACCCCGU")
Out[2]: 2

In [3]: fold("AAUUGCGC")
Out[3]: 4

In [4]: fold("ACUGAGCCCU")
Out[4]: 3

In [5]: fold("ACUGAGCCCUGUUAGCUAA")
Out[5]: 8
```

You may find the built-in `max` function useful. This function can be invoked with a pair of numbers as arguments, or it can take a whole list, as demonstrated here:

```
In [1]: max(42, 10)
Out[1]: 42

In [2]: max( [13, 42, 18, 0, 7] )
Out[2]: 42
```

Memoized `fold`

The `fold` function is nice, but it's way too slow to be useful for real RNA strings. It would literally take millions or billions of years for `fold` to find solutions for actual RNA data.

Memoization is the secret to all happiness! Write a function called `mfold(RNA)` that takes an RNA string and uses a global Python dictionary for memoization. We'll always call this function with the dictionary initially being empty (that is, `{}`). This function should do the same thing as `fold` but *much* faster.

In particular, test your function with the same arguments as above and also the following real RNA strand (5S rRNA sequence(s) for: Acanthamoeba castellanii):

```
testRNA =
"GGAUACGGCCAUACUGCGCAGAAAGCACCGCUUCCCAUCCGAACAGCGAAGUUAAGCUGCGCC
AGGCGGUGUUAGUACUGGGGUGGGCGACCACCCGGGAAUCCACCGUGCCGUAUCCU"
```

Here's the `mfold` function in action:

```
In [1]: mfold(testRNA)
Out[1]: 52
```

Submit

Submit your functions in a file called `hw3pr3.py`.

Optional Bonus Part A (5 Points)

Your `mfold` function quickly found the maximum number of matches possible in a RNA folding. In this optional problem, you will write a new version of this function called `getFold`that again uses memoization but this time returns a list in which the first element is the maximum number of matches and the second element is a list. Each element of that list is itself a list of two elements of the form `[x, y]`, where `x` and `y` are the indices of two nucleotides that are matched in an optimal folding. In other words, `getFold` lets us actually see the pairs that are matched in an optimal solution.

Here are some examples of `getFold` in action. **Keep in mind that in many cases there will be more than one possible solution (folding) for a given RNA sequence.**Therefore, if your solutions look different than the ones below, they may still be correct! However, since all optimal solutions will have the same number of matches, make sure that your solutions have the same number of matches as ours. Then, check that your solutions are valid by checking to see that the pairs of indices that are found are really matching pairs!

```
In [1]: getFold("ACCCCCU")
Out[1]: [1, [[0, 6]]]

In [2]: getFold("ACCCCGU")
Out[2]: [2, [[0, 6], [4, 5]]]

In [3]: getFold("AAUUGCGC")
Out[3]: [4, [[0, 3], [1, 2], [4, 5], [6, 7]]]
```

```
In [4]: getFold("ACUGAGCCCU")
Out[4]: [3, [[1, 3], [4, 9], [5, 6]]]

In [5]: getFold("ACUGAGCCCUGUUAGCUAA")
Out[5]: [8, [[0, 2], [3, 7], [5, 6], [8, 10], [11, 18], [12,
13], [14, 15], [16, 17]]]
```

Warning: We haven't talked about `for` loops and you may be tempted to use them here. However, `for` loops are not necessary because `map` can do everything that you might want a `for` loop to do and using `map` will result in cleaner code! Do use `map` (or equivalently, list comprehensions).

Optional Bonus Part B (5 Points)

Use the turtle to draw the nucleotide pairings! See BlackRNATurtleFun.

Please do note that this extra-credit problem should be submitted under `hw3pr4.zip`, ***not*** here!