Implement a Greedy Motif Finder

The Motif-Finding Problem asks to find a list of k-mers `Motifs` that minimizes the `Score(Motifs)`. We have shown that designing an algorithm that tries all combinations of potential `Motifs` will be very, very slow, even if we enumerate all k-mers to solve the Median String Problem.

Instead, you will implement a greedy algorithm to find a list of k-mers `Motifs` that is "pretty good." There are three places that you can find (slightly different) pseudocode for this algorithm: (1) below, (2) in the slides, and (3) on p. 85. in the textbook.[1] Use datasets from Rosalind to check your work (log into Rosalind to get access to the URLs) - however, I will only grade the code submitted to Moodle.

```
GreedyMotifSearch(Dna, k):
    bestMotifs  = First k-mers in each string from Dna
    for each kmer in Dna[0]:
        Motifs = [kmer]
        for each dna_string in Dna[1]...Dna[len(Dna)-1]:
            Make a profile from Motifs (*include pseudocounts*)
            Append the Profile-most probable k-mer in dna_string to Motifs
        if Score(Motifs) < Score(bestMotifs):
            bestMotifs = Motifs
    return bestMotifs
```

1. **Copy relevant functions from Lab6.** You may use `Lab6-solution.py` posted on Moodle; "cite" where you copied the functions from.
2. Read in the file `simulated-motifs.txt`. This is the dataset with the (15,4) motif from class.
3. **Write a function to compute the Profile-most probable k-mer.** This function takes as input a Profile (frequency table) and a text string (e.g., `Dna[i]`) and returns a k-mer from the text string with the highest probability of being generated from the Profile.

   

   Profile-most Probable k-mer: `http://rosalind.info/problems/ba2c/?class=406`
4. Implement `GreedyMotifSearch()` (without pseudocounts at first).

   

   Greedy Motif Search: `http://rosalind.info/problems/ba2d/?class=406`
5. Modify your code to handle pseudocounts.

   

   Search w/ Pseudocounts: `http://rosalind.info/problems/ba2e/?class=406`
6. Hand in your code that runs `GreedyMotifSearch()` on `simulated-motifs.txt` with $k = 15$.

Extra Exercise (Optional). How would you modify `GreedyMotifSearch` to improve the accuracy? Implement a function that tries to improve upon the pseudocode above. The `random` package (`https://docs.python.org/2/library/random.html`) may be useful, but it's not necessary.

[1] Note that the version in the textbook doesn't include pseudocounts!