CSCI 102 assignment 3 – Recursion

Alan Amin

Sept 29

In this assignment you'll be implementing a simple dynamic programming method for performing sequence alignments using recursion. We'll be looking for alignments of a sequence / list `PositionList<E> seq1` in a longer sequence `PositionList<E> seq2`; `int numAli(PositionList<E> seq1, PositionList<E> seq2)` will return the number of alignments. We'll be looking at `String`s as running examples assuming they're position lists of characters. An alignment is a sequence of elements in `seq2` that is the same order as those in `seq1`, for example:

`numAli("ABA", "AABABA")=7` since we have

$$\mathbf{A}A\mathbf{BA}BA, \mathbf{A}A\mathbf{B}AB\mathbf{A}, \mathbf{A}ABA\mathbf{BA}, A\mathbf{ABA}BA, A\mathbf{AB}AB\mathbf{A}, A\mathbf{A}BA\mathbf{BA}, AAB\mathbf{ABA}.$$

Now write out a bunch of examples to get some intuition! Try calculating `numAli("ABBA", "AABAAABABBABA")`; notice any easy strategy?

1 Recursive implementation

We can calculate `numAli` by looking for the first letter of `seq1` in `seq2` first:

```
numAli("ABA", "AABABA")=numAli("BA", "ABABA")+numAli("BA", "BABA")
                        +numAli("BA", "BA")+numAli("BA", "")
```

- Write a recursive implementation of `int numAli(PositionList<E> seq1, PositionList<E> seq2)`.
- If `seq1` and `seq2` have lengths L_1 and L_2, the worst case asymptotic computational complexity is $O(L_2 \times \binom{L_2}{L_1})$ (you don't need to prove this), which can be as bad as $O(L_2 \times 2^{L_2})$. In future courses you'll learn efficient "dynamic programming" methods that will bring the complexity down to $O(L_1 L_2)$. **Where is computation wasted in your recursive implementation?**
- Write a `main` method that calculates `numAli("ABBA", "AABAAABABBABA")`.

2 Maximum gaps

We can count the number of gaps in each alignment: ex AAB**ABA** has 0 gaps but **A**A**B**AB**A** has 3 gaps. Say we want to calculate the number of alignments that have fewer than `int maxGaps` gaps. Again write out a bunch of examples to get some intuition!

- Write a recursive implementation of `int numAli(PositionList<E> seq1, PositionList<E> seq2, int maxGaps)` by keeping track of the number of gaps during recursion (don't count gaps when aligning the first letter!) and modifying the sum in the above recurrent relation to only count alignments that don't add too many gaps. **Explain** why your code has this property!
- Write a `main` method that calculates `numAli("ABBA", "AABAAABABBABA", 3)`.
- Write a method `int totalNumGaps(PositionList<E> seq1, PositionList<E> seq2, int maxGaps)` that counts the total number of gaps across all alignments which obeys a similar recurrent relation to `numAli`. You'll probably want to keep track of `numAli` as an auxiliary variable.

Submission Please submit your code and answers to the questions in a zipped folder on brightspace by Oct 6. Remember to use the principles of encapsulation and least privilege!