

Combinatorics Mini-Project:

Designing DNA Sequences under Constraints

A DNA sequence can be represented as a word over the alphabet $\Sigma = \{A, C, G, T\}$, where A , C , G , and T represent the four nitrogenous bases.

Throughout this project, the term *sequence* refers to a word over the quaternary alphabet $\Sigma = \{A, C, G, T\}$.

In some applications of computational biology, we want to generate or analyze sequences satisfying certain properties, such as a fixed length, a specific composition of nitrogenous bases, or the absence of certain motifs.

Assignment

The objective of this project is to write a computer program that performs the following two tasks for each of the sets of DNA sequences defined below:

1. List all sequences belonging to the set under consideration.
2. Calculate and display the number of sequences in that set. Whenever possible, this number should be obtained in two ways:
 - using a mathematical formula;
 - by counting the elements in the list generated in Step 1.

Sets of Sequences to Study

- A. The set of all sequences of length n , without any restrictions.
- B. The set of sequences of length n that do not contain the letter A .
- C. The set of sequences of length n that contains an odd number of letter A .
- D. The set of sequences of length n that do not contain two consecutive letters T .
- E. The set of sequences of length n that do not contain the motif TAG as a factor (i.e., as a contiguous substring).
- F. The set of sequences of length 10 containing exactly two letters A , four letters C , three letters G , and one letter T .
- G. The set of sequences of length n containing exactly k nitrogenous bases belonging to the set $\{C, G\}$.

The values of n (and k) will be entered by the user.

Example of a Java program. Below is a Java recursive program generating all ternary sequences of length 4 over the alphabet {0,1,2}. The output of this program is: 0000, 0001, 0002, 0010, 0011, 0012, ..., 2222.

```
class Main {
    public static void generate(int[] genome, int position) {
        // base case: the sequence is completed
        if (position < 0) {
            for (int i = 0; i < genome.length; i++) {
                System.out.print(genome[i]);
            }
            System.out.println();
            return;
        }

        // recursive case: insert the three possible values: 0, 1 et 2
        for (int i = 0; i <= 2; i++) {
            genome[position] = i;
            generate(genome, position - 1);
        }
    }

    public static void main(String[] args) {
        int n = 4;
        int[] genome = new int[n];
        generate(genome, genome.length - 1);
    }
}
```

Example of Python program. The output is the same as previously.

```
def generate(genome, position):
    # base case: the sequence is completed
    if position < 0:
        for i in range(len(genome)):
            print(genome[i], end="")
        print()
        return

    # recursive case: insert the three possible values: 0, 1 et 2
    for i in range(3):
        genome[position] = i
        generate(genome, position - 1)

n = 4
genome = [0] * n
generate(genome, len(genome) - 1)
```