### **The data structures used in the kmer analysis**

In the script kmer\_analyzer.py, two main Python dictionaries are used. I chose these data structures to store each kmer and its following character as a unique key and their corresponding counts as values. The use of dictionaries as data structures makes it easy to access complex data sets when processing large sequences.

1. counts dictionary   
   This dictionary was used as part of the get\_kmer\_counts function to store the total number of times each unique k-mer appears in the DNA sequence.  
   Example: {"ACG": 2, "CGA": 1, "GAC": 1}, where each kmer is a unique key
2. follow\_counts dictionary  
   This is a nested dictionary which was used as part of the get\_following\_counts function to store for each unique k-mer, a sub-dictionary of characters that immediately follow the k-mer and their counts.

Example: {"AA": {"A": 1}}

### **My thoughts on handling Edge Cases**

My approach on handling edge cases for this analysis is as follows:

**Edge Case 1**

It tests that if k (length of kmer) is greater than the length of the sequence, it is impossible to extract even a single valid k-mer. Internally, the function uses a loop for i in range(len(sequence) - k + 1). If k is greater than the length of the sequence, this range becomes negative or zero, so the loop never runs. As a result, the function returns an empty dictionary, correctly indicating that there are no valid k-mers. Without handling this case, the function might throw an error (e.g., index out of range), enter an infinite loop or return incorrect, misleading data.

**Edge case 2**

This test prevents accessing out-of-bounds indexes and avoids falsely assigning follow characters where there are none. For example, the last k-mer that starts at position len(sequence) - k may not have a character following it. In such cases, the function excludes the final k-mer if it doesn’t have a next character and does not add an entry in the follow\_counts dictionary.

My current edge cases do not account for special characters (like N, -, or other non-ACGT characters) in the sequence since they are not present in the input file. However, DNA sequences can contain ambiguous or special characters. As such, to effectively manage this edge case, cleaning the sequence to remove or handle special characters before analysis and adding a test case to exclude such characters is a better approach.

### **Avoiding Overcounting or Missing Context**

* The script ensures accurate counting by iterating over every valid position i in the sequence where a k-mer of length k can be extracted.
* In the get\_kmer\_counts function, the loop iterates from position 0 to len(sequence) - k + 1, ensuring that each valid k-mer is counted exactly once per occurrence in the sequence. This prevents duplicate counting of overlapping or adjacent k-mers beyond what is present.
* In the get\_following\_counts function, it only counts the character that comes right after each k-mer—but only if there is a character after it. This avoids counting anything past the end of the sequence that doesn’t exist. The loop runs from position 0 to len(sequence) - k, which excludes any k-mer that appears at the very end of the sequence without a character following it.
* Using dictionaries to maintain unique keys for each k-mer and their associated following character frequencies, ensured that each event is recorded and updated only once per true observation.