adef numbertoremain(index, k):

remain = []

for i in range(k):

remain.append(index%4)

index = index//4

remain.reverse()

return remain

def remaintoseq(remain):

seq = ''

for i in remain:

dict = {0: 'A', 1:'C', 2: 'G', 3:'T'}

seq = seq + dict[i]

return seq

def numbertopattern(index, k):

pattern = remaintoseq(numbertoremain(index, k))

return pattern

print(numbertopattern(5437, 8))

* Pattern to number: Visualización que al asignar un valor A = 0, C =1, G=2, T=3, se le puede dar un valor numérico que puede ser procesado luego, esto al usar la propiedad que al tener cuatro bases existen 4^k posibilidades en las que un fragmento puede tener un acomodo.

def Patterntonum(Text):

i = 0

num = 0

while i < len(Text):

if Text[i] == 'C':

num += 1 \* 4 \*\*(len(Text)-i-1)

elif Text[i] == 'G':

num += 2 \* 4 \*\*(len(Text)-i-1)

elif Text[i] == 'T':

num += 3 \* 4 \*\*(len(Text)-i-1)

i += 1

return num

* Patterntonumber:

def NumberToPattern(index, k):

pattern = ''

for k in range(k):

remainder = index%4

index = index // 4

if remainder == 3:

pattern = "T" + pattern[:k]

if remainder == 2:

pattern = "G" + pattern[:k]

if remainder == 1:

pattern = "C" + pattern[:k]

if remainder == 0:

pattern = "A" + pattern[:k]

k += 1

return pattern

True or false: a motif of lowest score with respect to a collection of strings must appear as a substring of one of the strings.

True, esta mal

* Computing Frequencies:

def SymbolToNumber(s):

if s=="A":

return 0

elif s=="C":

return 1

elif s=="G":

return 2

elif s=="T":

return 3

def LastSymbol(p):

n = len(p)

return p[n-1]

def Prefix(p):

newstr = ""

n = len(p)

newstr = p[:n-1]

return newstr

def PatternToNumber(Pattern):

if Pattern == "":

return 0

symbol = LastSymbol(Pattern)

prefix = Prefix(Pattern)

return (4 \* PatternToNumber(prefix) + SymbolToNumber(symbol))

def ComputingFrequencies(Text,k):

FreqArr = {}

for i in range(0, 4\*\*k):

FreqArr[i] = 0

for i in range(len(Text)-k+1):

Pattern = Text[i:i+k]

j = PatternToNumber(Pattern)

FreqArr[j] += 1

Result = ""

for item in FreqArr.values():

Result = Result + " " + str(item)

return Result

* FrequentWordswith mismatches: Most frequent k-mers from a string with mismatches.

def FrequentWordsWithMismatches(Text, k, d):  
    counts = dict()  
    for i in range(len(Text)-k+1):  
        neighbor = Neighbors(Text[i:i+k], d)  
        for n in neighbor:  
            counts[n] = counts.get(n, 0) + 1  
    m = max(counts.values())  
    return [t for t, v in counts.items() if v == m]

from itertools import combinations, product  
def Neighbors(kmer, d):  
    mismatches = [kmer]   
    alt\_bases = {'A':'CGT', 'C':'AGT', 'G':'ACT', 'T':'ACG'}  
    for dist in range(1, d+1):  
        for change\_indices in combinations(range(len(kmer)), dist):  
            for substitutions in product(\*[alt\_bases[kmer[i]] for i in change\_indices]):  
                new\_mistmatch = list(kmer)  
                for idx, sub in zip(change\_indices, substitutions):  
                    new\_mistmatch[idx] = sub  
                mismatches.append(''.join(new\_mistmatch))  
    return mismatches