DNA – Double Helix of Nucleotide Strands.

Nucleotide – Half of one "rung" of DNA

Four choices for the nucleic acid of a nucleotide:

1 – Adenine (A)

2 – Cytosine (C)

3 – Guanine (G) – bonds to C

4 – Thymine (T) – bonds to A.

If we know one strand of DNA, we get the other strands for free because of this "Complementarity."

- Genome sequencing refers to "Reading" the "genome"

- Genome: The nucleotide sequence read down one side of an organism's Chromosomal DNA. A human genome has about 3 billion letters. or nucleotides.

- (An Amoeba) has one of the longest genomes: 670 billion nucleotides.

- For simplicity a Nucleotide is represented by a single character which refers to Nitrogenous base. (A, T, G, or C).

Nucleotide
* Nitrogenous base * Sugar molecule.
* Phosphate group

(*) Nucleic Acids (RNA and DNA) are long chains (polymers) made up of many nucleotides linked together. These chains encode genetic information, which is used in cellular processes like protein synthesis, replication, and gene expression.

- Genome: Complete set of genetic information in an organism.

DNA

ATAGGGTATTTGGCCCCAATCCGTAGCCTATTGGCA

Gene: A DNA sequence with directions for one protein.

An RNA's copy of the genes DNA

- Protein  : Molecular Machines of the cell.

- Origin of Replication:
  Replication begins in a region called replication Origin (denoted Ori)

  ∴ proceeds in both direction.

- Vibro Cholera – bacterium that causes Cholera (~ 500 Nucleotides).

  ∴ The string of 500 nucleotides must contain a hidden msg telling the cell to start replication.

- Hidden Message Problem:
  Input: String text (representing Ori)
  Output: hidden Message in text.

Replication Initiation is mediated by a protein called DnaA.

DnaA binds to a short segment in Ori known as a DnaA box, hidden message saying: "bind here"

Multiple DnaA boxes → higher chance of binding → higher "fitness"

- A K-mer pattern is a most frequent K-mer in a string if no other K-mer is more frequent than this pattern.

K-mer: substring of length K within a biological sequence.

2-mers: AA, AT, TA, TT . .

How many 2-mers are there?

4 options available × 4 = 16 2-mers can exist.

How many K-mers are there?

— × — × . . . K times

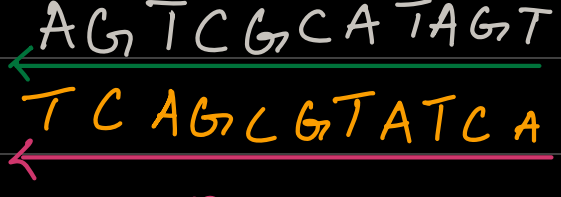
$(4)^k$ k-mers are there.

String or patter : Frequency.

- return one with highest frequency.
- k represent> length of substring pattern or k-mer.
  k length pattern.

T – A (mapping)

G – C (mapping)



1) First map right to left.

2) Then reverse from right to left.

ACTATGCGACT

Q: Given Ori, how do we find "the hidden message" (DnaA box) saying that replication should start here?

- Find the most frequent K-mers that appear in this region (Consider their reverse Complements as well). (DnaA boxes are Usually around 9 nucleotides long).

*) Different Genomes ⇒ Different hidden messages.

GGTGGTAGG CCTACCACC

GGTGGTAGG

Most Frequent 9-mer

Check its reverse Complement as well.

CCTACCACC

| | | | | | | | |

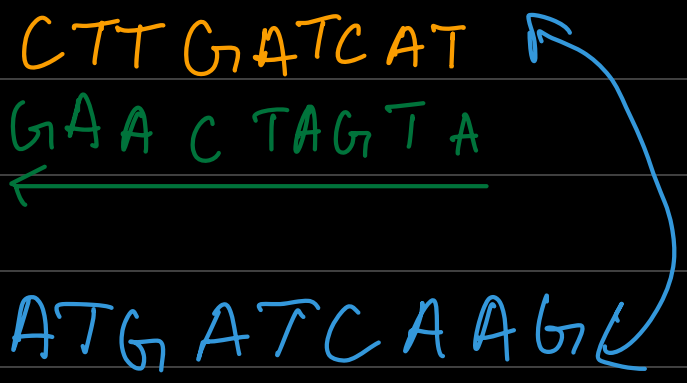
Complement GGATGGTGG

are candidate hidden messages.

GGATGGTGG

CCTACCACC

CCACCATCC



Reverse Complements of each other.

A K-mer forms an (L, t) - Clump inside Genome if there is short (length L) interval of Genome in which it appear many (at least t) times.

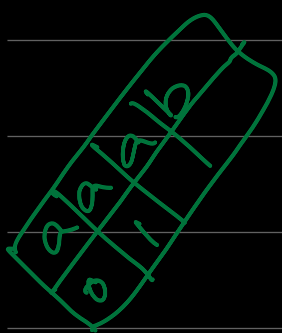
- Clump Finding Problem:

input: A string Genome and integers K (length of a pattern), L (window length) and t (number of patterns in a Clump.

Output: All K-mers forming (L, t) - Clumps in Genome.

```
def Construct_Lps(pattern):
    Lps = [0]
    if len(pattern) == 1: return Lps
    i, j = 1, 0  #
    Count = 0
    While  i <= len(pattern) - 1:
        if patter[i] == pattern[j]:
            Count += 1
```

```
pattern[i] = Count
j += 1
else:  # not same.
    Count = 0
    if j == 0:
        pattern[i] = Count
    else:
        pattern[i] = Count.
        j = 0  # start from beginning.
        Continue.
i += 1
```



k - lps x 1

ababa

Count

return lps

j ↓ (above b at index 3), i ↓ (above a at index 5), ↓ (above the last b)

a	b	a	b	a	a	b
0	0	1	2	3	1	2

Count += 1
- Move both pointers forward } if same.

if not same
set Count = 0
and set that
index element to count.
if j == 0 : # already left most then do nothing
Otherwise move it to the left by 1.

a b a b a a b

0 0 1 2 3 1 2

```
def kmp_search(text, pattern):
    n = len(text)
    m = len(pattern)

    lps = construct_lps(pattern)
    res = []
    i, j = 0, 0
    while i < n
        if text[i] == pattern[j]:
            i += 1
            j += 1

        if j == m:
            res.append(i - j)
            j = lps[j - 1]
        else:
            if j != 0:
```

```
            j = lps[j-1]
        else:
            i += 1

    return res.
```

abcdefghij

i

0+4

3 1+9 1:5

BA2B

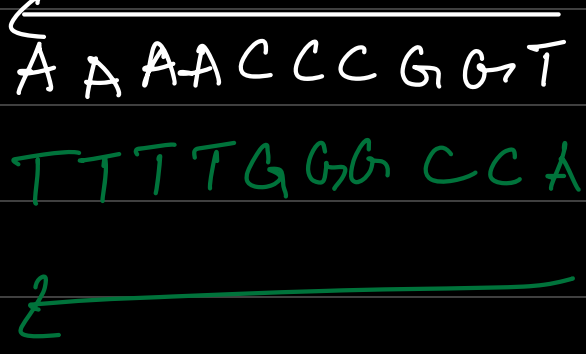
```
def Kmp_search(text, pattern): # How many times string appeared in the text.
    return count.

def frequent_words(text, k):
    freq_table = { }
    words = [ ]
    for i in range(0, len(text)):
        k_mer = text[i : i+k]
```

0:4

1:5

```
        freq = kmp_search(text, k_mer)
        freq_table[k_mer] = freq

    maximum = max(freq_table.values())
    for key in freq_table:
```

```
        if freq_table[key] == maximum:
            words.append(key)

return words
```



ACCGGGTTTT

GGTCA

CAGT

BA1C

TGAC

```
def reverse_complement(pattern):
    mapping = {'G':'C', 'C':'G', 'A':'T',
               'T':'A'}

    new_string = ""
    for i in range(0, len(pattern)):
        new_string += mapping[pattern[i]]
```

```
    new_string = new_string[::-1] # reverse
    return new_string.
```

k_mer
interval
atleast this many times appeared in interval.

```
def Clump_finding (Genome, K, L, t):
    K_mers = [ ]  set
```

- Start with first window of size L, create all Kmers in this window and store frequency in a dictionary.

- Simultaneously for that particular window check if any Kmer appeared at least t times and append it to list of Kmers.

- return Kmers.

```
                                    1+1
                                   ACCC
                                   00
def skew(Genome):                  0 1 2
    arr = [0] * (len(genome)+1)
    g_s, c_s = 0, 0 , indices = []
    for i in range(0, len(genome)):
        if genome[i] == 'A' or genome[i]
                                == 'T':
            arr[i+1] = arr[i-1]
        else:
            if genome[i] == 'G':
                g_s += 1
            elif genome[i] == 'C':
                c_s += 1

            arr[i+1] = g_s - c_s

    minimum = min(arr)
    indices = [i for i in range(0,len(arr))
                    if arr[i] == minimum]

    return indices.
```

[0]

CATGCC

[0, -1]

[0, -1, -1]

0, -1 -1 -1

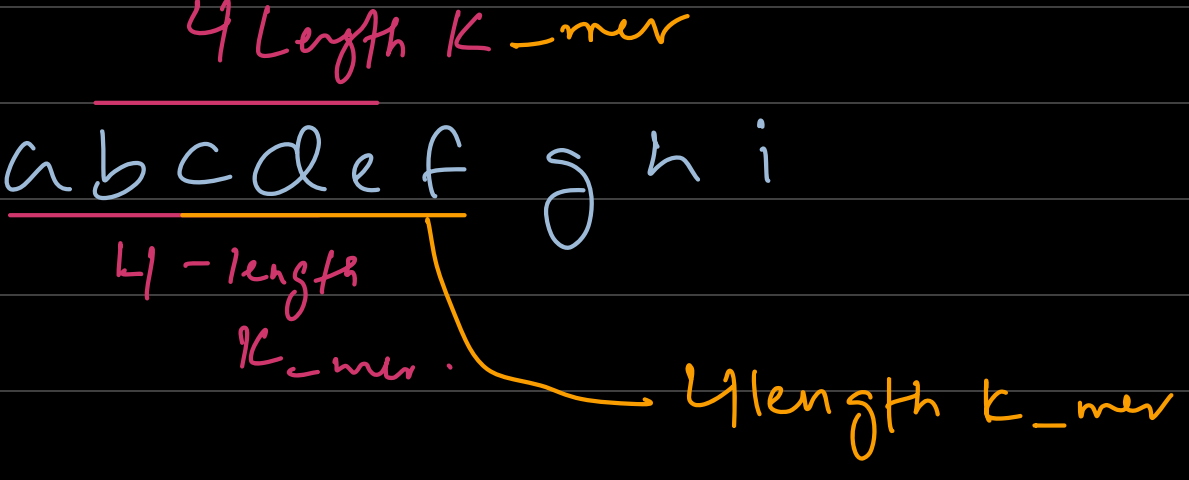
[0, -1 -1 -1 0]

[0 −1 −4 −4 0 −1 −2]

```
def Clump_finding(genome, K, L, t):
    K_mers = set()

    for i in range(len(genome)):
        interval = genome[i:i+L]
        K_mer_count = defaultdict(int)

        for j in range(len(interval)):
            K_mer = kwindow[j:j+K]
            K_mer_count[K_mer]+=1

        for K_mer, count in K_mer_count.
                                items
            if count>=t:
                K_mers.add(K_mer)

    return K_mer
```

*) Find most Frequent words in a String — basically just use dictionary.

K_mer = [i : i+k]



K_mer — key
Count — value

and then return the one with largest Count.