IBS 3

Assignment 2

KAUSHIK.M
CB.EN.U4AIE19036

DATE:21.08.2020

1. Write a code to create a k-mer composition for the given nucleotide sequence ‘TAATGCCATGGGATGTT’ ?
   (hint: take ‘k’ as 3)

CODE:

```
using LinearAlgebra
#Function to find all the possible kmers from the string
function pattern(String, k)
    #Creating a list to handle kmers obtained
    s = [];
    #Pushing them into the list
    for i in 1:length(String)-k+1
        s = push!(s,String[i:i+k-1]);
    end
    #Returning the list and the deictionary
    return s;
end
#Storing the list and dictionary obtained from the function
k = 3;
kmer1 = pattern("TAATGCCATGGGATGTT",k)
```

OUTPUT:

```
#Function to find all the possible kmers from the string
function pattern(String, k)
    #Creating a list to handle kmers obtained
    s = [];
    #Pushing them into the list
    for i in 1:length(String)-k+1
        s = push!(s,String[i:i+k-1]);
    end
    #Returning the list and the deictionary
    return s;
end
```

```
pattern (generic function with 1 method)
```

```
#Storing the list and dictionary obtained from the function
k = 3;
kmer1 = pattern("TAATGCCATGGGATGTT",k)
```

```
15-element Array{Any,1}:
 "TAA"
 "AAT"
 "ATG"
 "TGC"
 "GCC"
 "CCA"
 "CAT"
 "ATG"
 "TGG"
 "GGG"
 "GGA"
 "GAT"
 "ATG"
 "TGT"
 "GTT"
```

2. Write a code to rearrange the k-mer composition into its lexicographic order.

CODE:

```
#Function to find out the lexicographic ordering of the kmers
function lexico_order(kmers)
    #Creating a local variable sorted_kmer
    sorted_kmer = kmers;
    for i in 1:length(sorted_kmer)
        for j in i:length(sorted_kmer)
            #Alphabets Unicode values are increasing in order from A to Z.
            #So a string will be arranged in alphabetical order in this fashion
            if(cmp(sorted_kmer[i], sorted_kmer[j]) >= 0)
                #Swapping takes place
                swap = sorted_kmer[i];
                sorted_kmer[i] = sorted_kmer[j];
                sorted_kmer[j] = swap;
```

```
            end
        end
    end
    #Returning the values obtained after sorting
    return sorted_kmer;
end
#Storing of the sorted kmer list to a variable
ord_kmer = lexico_order(kmer1)
```

OUTPUT:

```
#Storing of the sorted kmer list to a variable
ord_kmer = lexico_order(kmer1)

15-element Array{Any,1}:
 "AAT"
 "ATG"
 "ATG"
 "ATG"
 "CAT"
 "CCA"
 "GAT"
 "GCC"
 "GGA"
 "GGG"
 "GTT"
 "TAA"
 "TGC"
 "TGG"
 "TGT"
```

3. Write a code to obtain the original nucleotide sequence.

Note: Due to the presence of kmer ("GGG") and the duplicates, the reconstruction of original sequence is a problem and thus, I have taken a smaller sequence "TACTGTT". So, the following activity alone was done with the mentioned sequence.

CODE:

```
#Creating a Dictionary to map kmer to its prefix => used for
reconstruction
function Prefix_Dict(kmer)
    #Initialise dictionary
    P = Dict();
    for k in kmer
        #Storing kmer mapped to its prefix
        P[k] = k[1:2];
```

```
    end
    #Returning the value;
    return P
end
PrefDict = Prefix_Dict(ord_kmer)
#Function used to find the next Prefix accessed from the Kmer-Prefix Dictionary
function next_str(inter_str, PrefDict)
    #If prefix is equal to the inter_str, the key,i.e, the kmer is returned
    kmer_key = [key for (key, value) in PrefDict if value == inter_str];
    return kmer_key;
end
#Function to identify the starting prefix
function Starting(ord_kmer)
    #Creating 2 lists - Prefix and Suffix
    Suffix = []
    Prefix = []
    for i in 1:length(ord_kmer)
        #Pushing suffix of each kmer
        Suffix = push!(Suffix,ord_kmer[i][2:k]);
        #Pushing prefix of each kmer
        Prefix = push!(Prefix,ord_kmer[i][1:k-1]);
    end
    #Storing Prefix list as a set in set1
    set1 = Set(Prefix);
    #Storing Suffix list as a set in set2
    set2 = Set(Suffix);

    #st is defined as set {Prefix set} - {Suffix set}
    st = setdiff(set1, set2);
```

```
    #ed is defined as set {Suffix set} - {Prefix set}
    ed = setdiff(set2, set1);

    st = string(st);
    ed = string(ed);

    #Starting prefix
    start = st[10:11];
    #Ending Suffix
    End = ed[10:11];
    return start;
end
function reconstruction(kmer, PrefDict, k)
    #Obtaining the starting kmer
    start_kmer = Starting(kmer);
    #Obtaining the next kmer in the reconstruction string
variable
    str = next_str(start_kmer,PrefDict);
    #Re-initialising str
    str = str[1][1:2];
    for i in 1:length(kmer)
        #Finding next kmer
        inter_str =  next_str(start_kmer,PrefDict)
        #Appending the last character from the obtained kmer
        str = string(str,inter_str[1][k])
        #Re-initialising start_kmer value
        start_kmer = str[end-1:end]
    end
    #Returning the string
    return str;
end
```

```
reconstructed_string = reconstruction(ord_kmer, PrefDict, k)
```

OUTPUT:

```
In [9]: function reconstruction(kmer, PrefDict, k)
            #Obtaining the starting kmer
            start_kmer = Starting(kmer);
            #Obtaining the next kmer in the reconstruction string variable
            str = next_str(start_kmer,PrefDict);
            #Re-initialising str
            str = str[1][1:2];
            for i in 1:length(kmer)
                #Finding next kmer
                inter_str =  next_str(start_kmer,PrefDict)
                #Appending the last character from the obtained kmer
                str = string(str,inter_str[1][k])
                #Re-initialising start_kmer value
                start_kmer = str[end-1:end]
            end
            #Returning the string
            return str;
        end

Out[9]: reconstruction (generic function with 1 method)

In [10]: reconstructed_string = reconstruction(ord_kmer, PrefDict, k)

Out[10]: "TACTGTT"
```

(Only the last part of code is shown here as I would like to show only the output-Code is present in the previous page.)

The only problem that might occur is when there is presence of duplicates of either prefix or suffix would cause a different string as output.