CS 4440 Winter 2023
Assignment #1
Total Marks: 20; Due date: February 17, 2023 (11.59 pm)

What you need to submit:
A single zip file that contains the following:

1. Screenshot that shows you are logged in to the GISAID
2. Downloaded sequences as individual .fasta files organized as per the given folder structure
3. Code (two program files)
4. Screenshot of 3D plot produced by your code

Programming language: Use as per the course outline.
The objective of this assignment is to learn the following: downloading & cleaning data, setting up the dataset; computing distances, performing dimensionality reduction, and data visualization.

PROGRAM SCRIPT#1 (*downloading and pre-processing data; setting up dataset*)
Step 1 (2 marks): Go to the GISAID site and register yourself. You need to click on the "Registration" tab and follow the instructions. Please note that it may take 24-48 hours before your account gets approved. After you get the approval email, please log in, and take a screenshot for submission.
https://gisaid.org/register/

Step 2 (2 marks): Log in using your credentials, and click on the "EpiCoV" tab. You need to search the "EpiCoV" database and download the sequences in "fasta" format using the following filters:

1. All the sequences should be "complete".
2. Sequences with low coverage should be excluded.
3. All the sequences should be from "North America/Canada/Prince Edward Island".
4. The variant should be "Delta" or "Omicron".

Step 3 (3 marks): For each variant, your downloaded sequences are in a single .fasta file. For each variant, you need to process these sequences as follows:

1. Read the sequences from the downloaded .fasta file.
   Hints:
   **fasta file is like any other text file. The first line contains "> Header"; the rest of the lines contain the sequence.*
   **Any inbuilt or existing implementation for reading the fasta files can be used.*
   **Do not re-invent the wheel. All programming languages have built-in methods that can read the .fasta files; these methods will split the headers and sequences for you. If you could not find such methods after a simple google search, please ask*.
2. Some sequences may be in lowercase. So, change all sequences to upper-case.
3. Keep only the following letters: A, C, G, and T. You need to remove all other unrecognized letters, including N's.
4. Write these processed sequences in individual .fasta files (one sequence per file) using the following folder (directory) structure:
   Parent Directory: SARS_CoV_2_Variants
   2 Sub-directories: Omicron and Delta
   Content of each sub-directory: individual .fasta files named variantName_x.fasta, where 1 <= x, and variantName is the sub-directory name.

PROGRAM SCRIPT#2 (*reading the dataset; computing Chaos Game Representations; computing distances; performing dimensionality reduction; visualizing relationships between data points*)

Step 4 (2 marks): Read the individual fasta files containing SARS-CoV-2 sequences.
Step 5 (3 marks): Compute the two-dimensional numerical representations of the genomic sequences using the Chaos Game Representation (CGR). Generate the CGR plots with k = 7 for all sequences i.e., one CGR per sequence. CGR code is provided as part of this assignment. Please note that the CGR at k=7 represents the frequencies of all possible sub-words of length 7 constructed over the alphabet set {A, C, G, T}.
Step 6 (4 marks): Use any distance/dissimilarity measure of your choice to compute pairwise distances between every pair of CGRs to create a pairwise distance matrix D (*you can flatten the CGRs by concatenating the columns to construct 1D representations*). The pairwise distance matrix D should be symmetric with every D(a, a) = 0, D(a,b) = D(b, a).
Step 7 (4 marks): Use the distance matrix D as an input to the classical multi-dimensional scaling (MDS) to perform dimensionality reduction. Create a 3D plot visualizing the relationships between all data points using the three most significant dimensions.

Hints:

**You do not have to code for the distance measures and the MDS. Any inbuilt or existing implementation can be used.*

**A few distance measures which you may consider: Manhattan distance, Structural dissimilarity (DSSIM), (1-Pearson Correlation Coefficient)/2, Euclidean distance, etc.*

CGR code is provided on the next page.

MATLAB:

Example: sequence= ACGGGAT and k-value is 3

cgrPlot = cgr('ACGGGAT', 'ACGT', 3);

```
function [out] = cgr(chars, order, k)
    out = zeros(2^k);
    x = 2^(k-1);
    y = 2^(k-1);
    for i = 1:length(chars)
        char = chars(i);
        x = fix(x/2);
        if char == order(3) || char == order(4)
            x = x + 2^(k-1);
        end
        y = fix(y/2);
        if char == order(1) || char == order(4)
            y = y + 2^(k-1);
        end
        if i >= k
            out(y+1, x+1) = out(y+1, x+1) + 1;
        end
    end
end
```

Java:

Example: String seq="AGGCTAGCCCTT"; String order="ACGT"; int k=3;
int arr[][]=cgr(seq,order,k);

```
public static int[][] cgr(String seq, String order, int k)
  {
    int len = seq.length();
    int pw = (int)Math.pow(2,k);
    int[][] out = new int[pw][pw];
    int x = (int)Math.pow(2,k-1);
    int y = (int)Math.pow(2,k-1);

    for(int i=0;i<len;i++)
    {
      char ch = seq.charAt(i);
      x = x/2;
      y = y/2;
      if(ch == order.charAt(2) || ch == order.charAt(3))
        x = x + (int)Math.pow(2,k-1);
      if(ch == order.charAt(0) || ch == order.charAt(3))
        y = y + (int)Math.pow(2,k-1);
      if(i>=k-1)
        out[y][x] = out[y][x]+1;
    }
    return out;
  }
```

R:
Example:
seq="**AGGCTAGCCCTT** "
res = cgr(seq, "ACGT", 3)

```
library(stringr)
cgr <- function(seq, order, k){
ln = str_length(seq)
pw = 2^k
out = matrix(0, pw, pw)
x = 2^(k-1)
y = 2^(k-1)
for(i in 1:ln){
x=floor(x/2)
y=floor(y/2)
ch=substr(seq, i, i)
if(ch == substr(order, 3, 3) | ch == substr(order, 4, 4))
x=x+(2^(k-1))
if(ch == substr(order, 1, 1) | ch == substr(order, 4, 4))
y=y+(2^(k-1))
if(i>=k)
out[y+1,x+1]=out[y+1,x+1]+1
}
return(out)
}
```

Python:
Example:
res = cgr(seq="AGGCTAGCCCTT",order="ACGT", k=3)

```
def cgr(seq, order, k):
   ln = len(seq)
   pw = 2**k
   out = [[0 for i in range(pw)] for j in range(pw)]
   x = 2**(k-1)
   y = 2**(k-1)

   for i in range(0,ln):
      x=x//2
      y=y//2
      if(seq[i] == order[2] or seq[i] == order[3]):
         x = x + (2**(k-1))
      if(seq[i] == order[0] or seq[i] == order[3]):
         y = y + (2**(k-1))
      if(i>=k-1):
         out[y][x] = out[y][x]+1

   return out
```

C++:

Example:

```
string seq="AGGCTAGCCCTT";
char order[]="ACGT";
int k=3;
int** res = cgr(seq,order,k);
```

```
#include <iostream>
#include <math.h>
#include <string.h>
int** cgr(string seq, char order[], int k)
{
   int len = seq.length();
   int pw = pow(2,k);
   int** out = new int*[pw];
   for(int i=0;i<pw;i++)
   {
      out[i]=new int[pw];
      for(int j=0;j<pw;j++)
      {
         out[i][j]=0;
      }
   }

   int x = pow(2,k-1);
   int y = pow(2,k-1);

   for(int i=0;i<len;i++)
   {
      char ch = seq[i];
      x = x/2;
      y = y/2;

      if(ch == order[2] || ch == order[3])
         x = x + pow(2,k-1);

      if(ch == order[0] || ch == order[3])
         y = y + pow(2,k-1);

      if(i>=k-1)
         out[y][x] = out[y][x]+1;
   }
   return out;
}
```

C:

Example:

```
char* seq="AGGCTAGCCCTT";
char order[]="ACGT";
int k=3;
int** res = cgr(seq,order,k);
```

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int** cgr(char* seq, char order[], int k)
{
   int pw = pow(2,k);

   int** out = malloc(pw*sizeof(int *));

   for(int i=0;i<pw;i++)
   {
      out[i] = malloc(pw*sizeof(int));
      for(int j=0;j<pw;j++)
      {
         out[i][j]=0;
      }
   }
   int x = pow(2,k-1);
   int y = pow(2,k-1);
   int i=0;

   while (*seq != '\0')
   {
      char ch = *seq;
      x = x/2;
      y = y/2;

      if(ch == order[2] || ch == order[3])
         x = x + pow(2,k-1);

      if(ch == order[0] || ch == order[3])
         y = y + pow(2,k-1);

      if(i>=k-1)
         out[y][x] = out[y][x]+1;
      seq++;
      i++;
   }
   return out;
}
```