Universidad Nacional Autónoma de México

Facultad de Ciencias

Profesor: M. en C. Sergio Hernández López

Genómica Computacional | Semestre 2026-1

Tarea 01

Martinez Oviedo Guillermo

Sánchez Cruz Norma Selene

Sosa Romo Juan Mario

1. Expresiones regulares

1. Considera la siguiente expresión regular, `r'TATAG[^AT](T*|AC)TATA'` y señala las cadenas que contengan instancias de la misma. También indica las posiciones donde cada instancia se encuentra.

   Ejemplo: En b) de la posición 10 a 20

   Para empezar tenemos que

   $$\underbrace{TATAG}_{1}\overbrace{[\hat{}AT]}^{2}\underbrace{(T*|AC)}_{3}\overbrace{TATA}^{4}$$

   donde,

   1. Esta parte nos dice que la cadena debe empezar tal cual con `TATAG`.
   2. Niega al conjunto de caracteres `A` y `T`, es decir, que no debe haber ni `A` ni `T` después de `TATAG`.
   3. Aquí nos indica que puede ser que aparezaca `T*` ó `AC`, pero tenemos que recordar que `*` es un cuantificador que nos dice que ese carácter esta cero o más veces, es decir, puede que no haya ninguna `T` en la cadena en este punto y saltarse a lo que pide 4 y estaría bien.
   4. Por último, como en 1, se espera que termine tal cual con `TATA`

   Teniendo esto podemos empezar a buscar las cadenas que contengan instancias.

   (a) TATACGCGTATAGAACTATAGCCCTATA

   $$\begin{array}{cccccccccccccccccccccccccccc} T & A & T & A & C & G & C & G & T & A & T & A & G & A & A & C & T & A & T & A & G & C & C & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \end{array}$$

   podemos ver que en las posiciones 9 a 13 se cumple 1. al empezar con `TATAG`, veamos si cumple con 2.

   $$\begin{array}{cccccccccccccccccccccccccccc} T & A & T & A & C & G & C & G & T & A & T & A & G & A & A & C & T & A & T & A & G & C & C & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \end{array}$$

   no cumple con 2. porque el carácter en la posición 14 es A justo un carácter del conjunto que no debía aparecer y solo por este caracter **NO será**, pues podemos que si cumple con 3. en las posiciones 15 y 16, y 4. en las posiciones 17 a 20.

   $$\begin{array}{cccccccccccccccccccccccccccc} T & A & T & A & C & G & C & G & T & A & T & A & G & A & A & C & T & A & T & A & G & C & C & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \end{array}$$

Podemos ver que hay otra cadena candidata que cumple 1. al empezar con TATAG en las posiciones 17 a 21

$$\begin{array}{cccccccccccccccccccccccccccc} T & A & T & A & C & G & C & G & T & A & T & A & G & A & A & C & T & A & T & A & G & C & C & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \end{array}$$

también cumple con 2. pues el carácter en la posición 22 es diferente de A y T,

$$\begin{array}{cccccccccccccccccccccccccccc} T & A & T & A & C & G & C & G & T & A & T & A & G & A & A & C & T & A & T & A & G & C & C & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \end{array}$$

pero vemos que las posiciones 23 y 24 no son T* ó AC, pues aunque se pueda pensar que son ceros T, para que esto fuera tendría que las posiciones 23 a 26 cumplir con 4., pero como no ocurre entonces no cumple con 3.

$$\begin{array}{cccccccccccccccccccccccccccc} T & A & T & A & C & G & C & G & T & A & T & A & G & A & A & C & T & A & T & A & G & C & C & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \end{array}$$

pero si en la posicion 23 hubiera estado A, la posiciones 25 a 28 cumplirían con 4.

$$\begin{array}{cccccccccccccccccccccccccccc} T & A & T & A & C & G & C & G & T & A & T & A & G & A & A & C & T & A & T & A & G & C & C & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \end{array}$$

En conclusión, no hay ninguna coincidencia válida, pues ninguna cumplio con la expresión regular.

(b) TATAGCGTATAGGACTATAGTATA

$$\begin{array}{cccccccccccccccccccccccc} T & A & T & A & G & C & G & T & A & T & A & G & G & A & C & T & A & T & A & G & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{array}$$

podemos ver que en las posiciones 1 a 5 se cumple 1. al empezar con TATAG, veamos si cumple con 2.

$$\begin{array}{cccccccccccccccccccccccc} T & A & T & A & G & C & G & T & A & T & A & G & G & A & C & T & A & T & A & G & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{array}$$

también cumple con 2. pues el carácter en la posición 6 es diferente de A y T,

$$\begin{array}{cccccccccccccccccccccccc} T & A & T & A & G & C & G & T & A & T & A & G & G & A & C & T & A & T & A & G & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{array}$$

pero vemos que la posicion 7 no es T, por lo que no cumple con 3., pero si hubiera sido T en lugar de G nos hubiera ayudado pues las posiciones 8 a 11 si cumplen con 4.

$$\begin{array}{cccccccccccccccccccccccc} T & A & T & A & G & C & G & T & A & T & A & G & G & A & C & T & A & T & A & G & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{array}$$

y tampoco podemos pensar que son ceros T, para que esto fuera tendría que las posiciones 7 a 10 cumplir con 4., pero no pasa y menos es AC.

Podemos ver que hay otra cadena candidata que cumple 1. al empezar con TATAG en las posiciones 8 a 12

$$\begin{array}{cccccccccccccccccccccccc} T & A & T & A & G & C & G & T & A & T & A & G & G & A & C & T & A & T & A & G & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{array}$$

también cumple con 2. pues el carácter en la posición 13 es diferente de A y T,

$$\begin{array}{cccccccccccccccccccccccc} T & A & T & A & G & C & G & T & A & T & A & G & G & A & C & T & A & T & A & G & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{array}$$

también cumple con 2. pues tenemos que tiene una de las dos opciones, es decir, en las posiciones 14 y 15 tenemos AC respectivamente,

$$\begin{array}{cccccccccccccccccccccccc} T & A & T & A & G & C & G & T & A & T & A & G & G & A & C & T & A & T & A & G & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{array}$$

y por último, también cumple con 4. pues termina tal cual con TATA en las posiciones 16 a 19

$$\begin{array}{cccccccccccccccccccccccc} T & A & T & A & G & C & G & T & A & T & A & G & G & A & C & T & A & T & A & G & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{array}$$

dejando así, que de la posición 8 a la 19 se cumple con la expresión regular.

Podemos ver que hay otra cadena candidata que cumple 1. al empezar con TATAG en las posiciones 16 a 20

$$\begin{array}{cccccccccccccccccccccccc} T & A & T & A & G & C & G & T & A & T & A & G & G & A & C & T & A & T & A & G & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{array}$$

pero ya no cumple con 2. pues el carácter en la posición 21 es T, si hubiera estado otro carácter diferente podríamos haber dicho que eran ceros T y de ser así ya tendríamos que las posiciones 21 a 24 cumplen con 4..

En conclusión, solo de la posición 8 a la 19 se cumple con la expresión regular.

(c) GTATGTATAGCCGACTTA

$$\begin{array}{cccccccccccccccccc} G & T & A & T & G & T & A & T & A & G & C & C & G & A & C & T & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \end{array}$$

podemos ver que en las posiciones 6 a 10 se cumple 1. al empezar con TATAG, veamos si cumple con 2.

$$\begin{array}{cccccccccccccccccc} G & T & A & T & G & T & A & T & A & G & C & C & G & A & C & T & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \end{array}$$

también cumple con 2. pues el carácter en la posición 11 es diferente de A y T,

$$\begin{array}{cccccccccccccccccc} G & T & A & T & G & T & A & T & A & G & C & C & G & A & C & T & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \end{array}$$

pero vemos que las siguientes posiciones no son T* ó AC, pues aunque se pueda pensar que son ceros T, para que esto fuera tendría que las posiciones 12 a 15 cumplir con 4., pero como no ocurre entonces no cumple con 3.. y a diferencia de los primeros incisos, apartir de este punto no hay modificaciones minimas posibles que podrían hacerlo funcionar.

En conclusión, no hay ninguna coincidencia válida, pues ninguna cumplio con la expresión regular.

(d) TATAGCCGACTATA

$$\begin{array}{ccccccccccccccc} T & A & T & A & G & C & C & G & A & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \end{array}$$

Podemos ver que las posiciones 1 a 5 se cumple con 1. al empezar con TATAG, veamos si cumple con 2.

$$\begin{array}{ccccccccccccccc} T & A & T & A & G & C & C & G & A & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \end{array}$$

y también cumple con 2. pues el carácter en la posición 6 es diferente de A y T,

$$\begin{array}{ccccccccccccccc} T & A & T & A & G & C & C & G & A & C & T & A & T & A \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \end{array}$$

pero no cumple ni con 3. pues vemos que las siguientes posiciones no son T* ó AC, pues aunque se pueda pensar que son ceros T para que esto fuera tendría que las posiciones 7 a 10 cumplir con 4., y a diferencia de los primeros incisos, apartir de este punto no hay modificaciones minimas posibles que podrían hacerlo funcionar.

En conclusión, no hay ninguna coincidencia válida, pues ninguna cumplio con la expresión regular.

2. A continuación se presentan 10 secuencias hipotéticas. Diseña una expresión regular que detecte regiones codificantes válidas y específica cuales de las secuencias la cumplen:

```
1.  ATATATACATACTGGTAATGGGCGCGCGTGTGTTAAGTTCTGTTGTAGGGGTGATTAGGGGCG
2.  GGCCCACACCCCACACCAATATATGTGGTGTGGGCTCCACTCTCTCGCGCTCGCGCTGGGGAT
3.  ATAAGGTGTGTGGGCGCGCCCCGCGCGCGCGTTTTTTCGCGCGCCCCCGCGCGCGCGCGCGCG
4.  GGCGCGGGACGCGGCGGCGGATCCCGATCCGTGCGTCAATACTATTATGGCCAGATAGAATAA
5.  GTGCTGCTGCGGCGCCCACACCTATTATCTCTCTCTCTCTGCCTCTCCACCTCGGGGCTTAAT
6.  GCGCTGCTGCTGGCTCGATGGGCGCGTGCGTCGTAGCTCGATGCTGGCTCGAGCTGTAATCTT
7.  GGCGCTCGCTCGGATGCGCGGCCGGGCTCTCTGCTCGCGCTCGCTTCGCGCTCGTGACCGCTG
8.  AATTGGTGCGCGCTCGCGCACACACAGAGAGAGGGTTTATATAGGATGATATATCCACATTGG
9.  ATGCTGCTGCTGGCTCTGCTTGCGCTCTGCTCGCTGGGGTGTGTGTGCCGCGCGCTGCTGCTC
10. GCTGGGCTCGCTCGATGCGCGCGGGCGCGCGACCGCGGACGGCGTCGCTGCTAAATGGGCTTC
```

Debes entregar una lista con el número de línea en las que hay una región codificante válida (como la hayas revisado en la clase de biología), es decir, si en las líneas 0, 1 y 5 hubiera entonces el resultado debería ser:

$$[0, 1, 5]$$

Para saber si una región codificante es válida debemos de tomar en cuenta tres cosas, el codon de inicio que es ATG, tripletes de bases intermedias que son ATCG, y un codon de paro que es TAA. Tomando en cuenta lo anterior, hacemos el sigueinte código en python.

```python
import re

reg = re.compile(r"ATG([ATGC]{3})+TAA")

secuencias = [
    "ATATATACATACTGGTAATGGGCGCGCGTGTGTTAAGTTCTGTTGTAGGGGTGATTAGGGGCG",
    "GGCCCACACCCCACACCAATATATGTGGTGTGGGCTCCACTCTCTCGCGCTCGCGCTGGGGAT",
    "ATAAGGTGTGTGGGCGCGCCCCGCGCGCGCGTTTTTTCGCGCGCCCCCGCGCGCGCGCGCGCG",
    "GGCGCGGGACGCGGCGGCGGATCCCGATCCGTGCGTCAATACTATTATGGCCAGATAGAATAA",
    "GTGCTGCTGCGGCGCCCACACCTATTATCTCTCTCTCTCTGCCTCTCCACCTCGGGGCTTAAT",
    "GCGCTGCTGCTGGCTCGATGGGCGCGTGCGTCGTAGCTCGATGCTGGCTCGAGCTGTAATCTT",
    "GGCGCTCGCTCGGATGCGCGGCCGGGCTCTCTGCTCGCGCTCGCTTCGCGCTCGTGACCGCTG",
    "AATTGGTGCGCGCTCGCGCACACACAGAGAGAGGGTTTATATAGGATGATATATCCACATTGG",
    "ATGCTGCTGCTGGCTCTGCTTGCGCTCTGCTCGCTGGGGTGTGTGTGCCGCGCGCTGCTGCTC",
    "GCTGGGCTCGCTCGATGCGCGCGGGCGCGCGACCGCGGACGGCGTCGCTGCTAAATGGGCTTC"
]

validas = [i for i, sec in enumerate(secuencias) if reg.search(sec)]
print(validas)
```

Esto nos da como resultado 5, ya que solo este tiene lo requerido inicio con ATG, tripletes intermedios y final con TAA.

3. **En la siguiente secuencia $TATAATCGTATTGTACTAATATTGTATATATA$ reporta la posición de cada uno de los siguientes patrones**

$$\{ATT, ATC, CTA, GTA, ATAT\}$$

usando Aho-Corasick. También reporta el autómata correspondiente.

Usando Aho-Corasick en Colab

Para el desarrollo de este ejercicio utilice una pequeña implementación del algoritmo de Aho-Corasick escrito en Python 3 usando Colab. La idea es construir un trie (arbol de prefijos) con los patrones, después usar BFS para encontrar los "fails" que son cuando dado una cadena, no encontramos la siguiente letra del texto en nuestros patrones y regresamos al sufijo que también es prefijo mas grande, trivialmente todos tienen como fail "" o el nodo raíz. Finalmente, recorremos el texto y buscamos si segun nuestro trie, hay coincidencia o no, usando los fails para regresar y los "next" para seguir. Aqui la clase:

```python
from collections import deque, defaultdict
from typing import List, Dict, Tuple
```

```python
class AhoCorasick:
    def __init__(self, patterns: List[str]):
        self.nodes = [{'next': {}, 'fail': 0, 'out': []}]
        self._build_trie(patterns)
        self._build_failure_links()

    def _add_node(self):
        self.nodes.append({'next': {}, 'fail': 0, 'out': []})
        return len(self.nodes) - 1

    def _build_trie(self, patterns: List[str]):
        for pat in patterns:
            node = 0
            for ch in pat:
                if ch not in self.nodes[node]['next']:
                    nxt = self._add_node()
                    self.nodes[node]['next'][ch] = nxt
                node = self.nodes[node]['next'][ch]
            self.nodes[node]['out'].append(pat)

    def _build_failure_links(self):
        q = deque()
        for ch, nxt in list(self.nodes[0]['next'].items()):
            self.nodes[nxt]['fail'] = 0
            q.append(nxt)

        # BFS
        while q:
            r = q.popleft()
            for ch, s in list(self.nodes[r]['next'].items()):
                q.append(s)
                state = self.nodes[r]['fail']
                while state != 0 and ch not in self.nodes[state]['next']:
                    state = self.nodes[state]['fail']
                self.nodes[s]['fail'] = self.nodes[state]['next'].get(ch, 0) if \
                                        ch in self.nodes[state]['next'] else 0
                self.nodes[s]['out'] += self.nodes[self.nodes[s]['fail']]['out']

    def search(self, text: str) -> List[Tuple[int,int,str]]:
        results = []
        node = 0
        for i, ch in enumerate(text):
            while node != 0 and ch not in self.nodes[node]['next']:
                node = self.nodes[node]['fail']
            node = self.nodes[node]['next'].get(ch, 0) if ch in self.nodes[node]['next'] else 0
            if self.nodes[node]['out']:
                for pat in self.nodes[node]['out']:
                    start = i - len(pat) + 1
                    end = i
                    results.append((start, end, pat))
        return results
```

Utlizando el codigo anterior podemos obtener los siguientes resultados:

```
Texto: TATAATCGTATTGTACTAATATTGTATATATA

Patrones: ['ATT', 'ATC', 'CTA', 'GTA', 'ATAT']

Patrones encontrados (start, end, pattern):
(4, 6, 'ATC')
(7, 9, 'GTA')
(9, 11, 'ATT')
(12, 14, 'GTA')
(15, 17, 'CTA')
(18, 21, 'ATAT')
(20, 22, 'ATT')
(23, 25, 'GTA')
(25, 28, 'ATAT')
(27, 30, 'ATAT')

Posiciones de inicio agrupada por partron:
  ATT: [9, 20]
  ATC: [4]
  CTA: [15]
  GTA: [7, 12, 23]
  ATAT: [18, 25, 27]
```

Lo cual coincide con una búsqueda a mano. Si pedimos a GPT que nos cree un codigo para visualizar el automata que creamos usara este codigo:

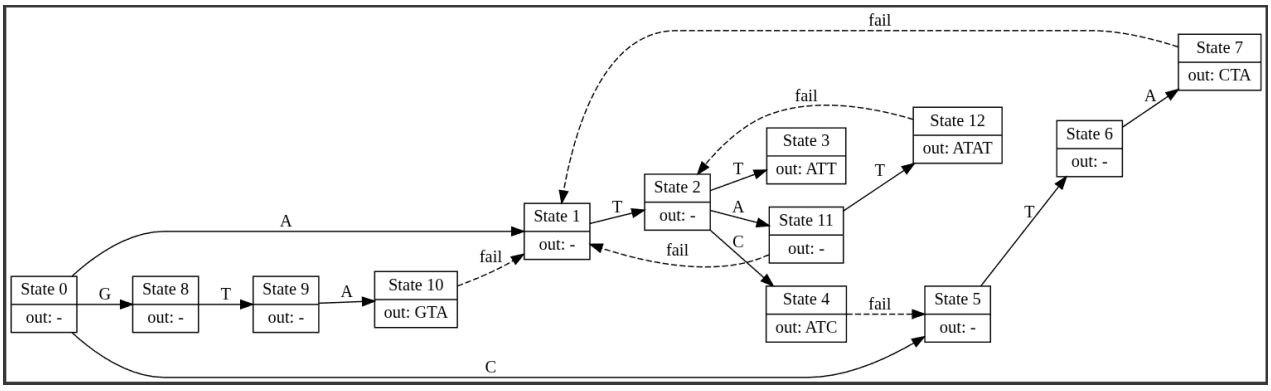
```
from graphviz import Source

dot_lines = ["digraph aho {", "  rankdir=LR;", "  node [shape=record];"]
for idx, node in enumerate(ac.nodes):
    out_join = ','.join(node['out']) if node['out'] else '-'
    label = f"<s>State {idx}|out: {out_join}"
    dot_lines.append(f"  n{idx} [label=\"{label}\"];")

for idx, node in enumerate(ac.nodes):
    for ch, nxt in node['next'].items():
        dot_lines.append(f"  n{idx} -> n{nxt} [label=\"{ch}\"];")
    if node['fail'] != 0:
        dot_lines.append(f"  n{idx} -> n{node['fail']} [style=dashed,label=\"fail\"];")
dot_lines.append("}")
dot_graph = "\n".join(dot_lines)

Source(dot_graph)
```

Y nos dara el siguiente automata:



2. Computación y complejidad

Contesta las siguientes preguntas sobre computación y complejidad computacional

1. *Algoritmo misterio* Haz un *script* en python que reciba como entrada un entero mismo que será usado como la cantidad de iteraciones de las siguientes instrucciones

Algorithm 1: Misterio

Input: M iteraciones
Output: Número flotante x
$i \leftarrow 1$;
$D \leftarrow 0$;
while $i < M$ **do**
  $i \leftarrow i + 1$;
  $x \leftarrow -1 \leq uniform() \leq 1$;
  $y \leftarrow -1 \leq uniform() \leq 1$;
  $d \leftarrow \sqrt{x^2 + y^2}$;
  if $d \leq 1$ **then**
    $D \leftarrow D + 1$;
$x \leftarrow 4 \cdot D/i$;
return x;

Tip: Revisa la documentación del paquete `random`, particularmente de la biblioteca de `numpy`.

El siguiente código se puede ejecutar en Ejercicio2.1:

```
import numpy as np

def Misterio(M):
  i = 1
  D = 0

  while i < M:
    i = i + 1
    x = np.random.uniform(-1, 1)
    y = np.random.uniform(-1, 1)
    d = np.sqrt(x**2 + y**2)

    if d <= 1:
      D = D + 1

  x = 4 * (D / i)

  return x
```

¿Qué está calculando el algoritmo 1? Esta estimando el valor de Pi utilizando el método Monte Carlo [1].

2. *Raro:* Determina la complejidad temporal en el peor caso usando la notación O grande del siguiente algoritmo

Algorithm 2: Raro

Input: $n \in \mathbb{Z}$
Output: x e y actualizados
$i, j, x, y \leftarrow$ int;
for $i \leftarrow 1$ ***to*** n **do**
  if i *is odd* **then**
    for $j \leftarrow 1$ ***to*** n **do**
      $x \leftarrow x + 1$;
    for $j \leftarrow 1$ ***to*** i **do**
      $y \leftarrow y + 1$;

La notación en O grande sería $O(n^2)$. Ya en el peor de los casos se ejecutan dos for anidados que tienen como complejidad n x n.

3. *Expansión - Modificación* Haz un función en python que reciba una cadena semilla y que con probabilidad `p` mute alguna posición elegida con probabilidad uniforme de la cadena original y con `1-p` la concatene con ella misma.

$$f(\sigma) \leftarrow \begin{cases} \text{muta}(\sigma) & \texttt{p} \\ \sigma\sigma & \texttt{1 - p} \end{cases}$$

Otra función debe recibir una cantidad de iteraciones para aplicar esta función. El alfabeto a usar es $\Sigma = \{0, 1\}$

Ejemplo:

```
cadena=0
cad_final = itera_n_veces (f(cadena(0.25)), 5)
cad_final
00100010
```

Función única en Colab

Para este ejercicio igualmente hicimos uso de Google Colab, aquí esta el código:

```python
import random

def modificaCadena(cadena : str, p : float, iteraciones = 1) -> str:
  def muta(cadena : str) -> str:
    posicion = random.randint(0,len(cadena)-1)
    ch = "0" if cadena[posicion]=="1" else "1"
    return cadena[:posicion]+ch+cadena[posicion+1:]

  while iteraciones > 0:
    iteraciones-=1
    if random.random() <= p:
      cadena = muta(cadena)
    else:
      cadena += cadena

  return cadena

print(modificaCadena("0", .25, 5))
```

La idea es sencilla, tomamos los argumentos que se nos pide, mientras haya iteraciones por hacer vemos si caemos dentro del caso de mutación o caso de duplicación, en el caso de mutación, buscamos una posición aleatoria dentro de la cadena, y cambiamos el carácter dependiendo de su valor inicial. Como nota, usamos una sola función para ambas cosas con un parámetro opcional iteraciones.

3. Probabilidad y estadística

1. En el archivo `promotores.txt` se encuentra la lista de secuencias tomadas del genoma de *Vitis vinifera* y cada una de las secuencias puede que tenga alguno de las diferentes formas en las que se ha encontrado el promotor GATA:

$$\{AGATAG, TGATAG, AGATAA, TGATAA\}$$

Deseamos estudiar estas regiones en función del promotor GATA y por lo tanto lo primero que deseamos es saber cuántas veces aparecen los promotores en cada región.

- **Haz un boxplot con la distribución de cada uno de los promotores**

El procedimiento para hacer un boxplot con la distribución de cada uno de los promotores se encuentra en Ejercicio3.1, pero el código "principal" para esta partes es

```
import re
import matplotlib.pyplot as plt
import numpy as np

# Crear boxplot
datos = list(aparecenPromotores.values())
labels = list(aparecenPromotores.keys())
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111)

box = ax.boxplot(datos, patch_artist= True, notch= True, vert=0)

colores = ["#8B008B", "#CAFF70", "#B0E2FF", "#FF6EB4"]
for patch, color in zip(box['boxes'], colores):
  patch.set_facecolor(color)

for whisker in box['whiskers']:
  whisker.set(color='#8B008B', linewidth=1.5, linestyle=":")

for cap in box['caps']:
  cap.set(color='#8B008B', linewidth=2)

for median in box['medians']:
  median.set(color='#CAFF70', linewidth=3)

for flier in box['fliers']:
  flier.set(marker='D', color='#B03060', alpha=1.0)

ax.set_xticklabels(labels)
ax.set_ylabel('Frecuencia en que aparece')
ax.set_title("Distribución de frecuencias de promotores")
ax.get_xaxis().tick_bottom()
ax.get_yaxis().tick_left()
ax.grid(True, linestyle = '--', alpha = 0.6)

# Mostrar
plt.show()
```

y con esto obteniendo lo siguiente:

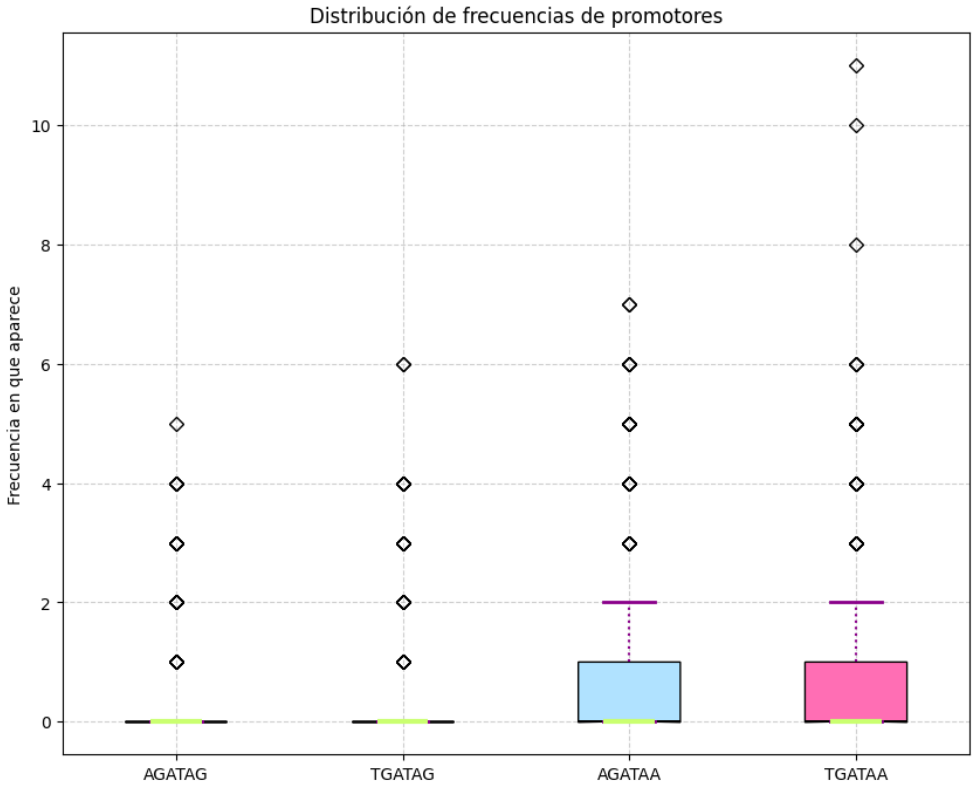


Figure 1: Boxplot con la distribución de cada uno de los promotores

- **¿Cuál es la media y desviación estándar de cada promotor?**

  De igual forma el código se encuentra en Ejercicio3.1, pero el código "principal" para esta partes es

```
import re
import matplotlib.pyplot as plt
import numpy as np

promotores = ['AGATAG', 'TGATAG', 'AGATAA', 'TGATAA']
aparecenPromotores = {prom:[] for prom in promotores}

# Contar las veces que aparece cada promotor en secuencia
for sec in secuencias:
  for prom in promotores:
    contador = sec.count(prom)
    aparecenPromotores[prom].append(contador)

# Calcular media y desviacion estandar para cada promotor
estadisticas = {}
print("-------------------------------------------------")
print("Promotor |       Media        | Desviación estándar ")
print("-------------------------------------------------")

for prom, con in aparecenPromotores.items():
  media = np.mean(con)
  desviacion = np.std(con)
  estadisticas[prom] = {'media': media, 'desviación estándar': desviacion}
  print(f"{prom}   | {media} | {desviacion}")

print("-------------------------------------------------")
```

  y se obtuvo que

Promotor	Media	Desviación estándar
AGATAG	0.18453538513570644	0.4388497692545206
TGATAG	0.22846030269721823	0.4931851689779564
AGATAA	0.4821171894816944	0.7340057927064843
TGATAA	0.6139342183140273	0.8284502958464521

2. Utilizando los siguientes datos de sensibilidad (93 por ciento) y especificidad (99 por ciento) reportados para cierta prueba rápida de antígeno para detectar la infección por virus SARS-COV2 y considerando una prevalencia actual de COVID en México estimada a partir del promedio de casos nuevos observados a lo largo de 2 semanas de 16000 casos activos respecto a una población total de 120,000,000 de habitantes encuentra:

   a) ¿Cuál es la probabilidad de que si uno de ustedes se realiza una prueba rápida de este tipo y esta resulta positiva ustedes en realidad sean portadores del virus SARS-COV2?

   b) ¿Cuál es la probabilidad de que si la prueba resulta negativa ustedes en realidad no sean portadores del virus SARS-COV2?

   c) Entre marzo y junio de 2021 se tuvo un promedio de nuevos contagios semanales de alrededor de 3000 casos, por lo que a lo largo de dos semanas se tendría una prevalencia aproximada de 6000 casos activos respecto a 120000000 de habitantes. Calcula las probabilidades referidas en los dos incisos anteriores pero considerando este nuevo dato de prevalencia. ¿Qué puedes concluir respecto a las probabilidades obtenidas en ambos escenarios?, ¿consideras que en el caso de las pruebas de detección de COVID es necesaria una mayor sensibilidad o una mayor especificidad? Justifica tu respuesta.

Respuesta

Tenemos lo siguiente:

- Sensibilidad: $S = 0.93$
- Especificidad: $E = 0.99$
- Prevalencia inicial: $P(D) = \frac{16{,}000}{120{,}000{,}000} \approx 0.0001333$

Fórmulas

Sea D el evento de estar infectado y $\neg D$ no estar infectado:

$$P(D|\text{positivo}) = \frac{S \cdot P(D)}{S \cdot P(D) + (1 - E) \cdot P(\neg D)}$$

$$P(\neg D|\text{negativo}) = \frac{E \cdot P(\neg D)}{E \cdot P(\neg D) + (1 - S) \cdot P(D)}$$

—

Escenario 1: Prevalencia alta

$$P(D) = 0.0001333, \quad P(\neg D) = 0.999867$$

a) Probabilidad de estar infectado si test positivo:

$$P(D|\text{positivo}) = \frac{0.93 \cdot 0.0001333}{0.93 \cdot 0.0001333 + 0.01 \cdot 0.999867} \approx \frac{0.000124}{0.010123} \approx 0.01225$$

$$\boxed{P(D|\text{positivo}) \approx 1.2\%}$$

b) Probabilidad de no estar infectado si test negativo:

$$P(\neg D|\text{negativo}) = \frac{0.99 \cdot 0.999867}{0.99 \cdot 0.999867 + 0.07 \cdot 0.0001333} \approx \frac{0.989868}{0.989877} \approx 0.9999906$$

$$\boxed{P(\neg D|\text{negativo}) \approx 99.999\%}$$

—

Escenario 2: Prevalencia baja

$$P(D) = \frac{6{,}000}{120{,}000{,}000} = 0.00005, \quad P(\neg D) = 0.99995$$

a) Probabilidad de estar infectado si test positivo:

$$P(D|\text{positivo}) = \frac{0.93 \cdot 0.00005}{0.93 \cdot 0.00005 + 0.01 \cdot 0.99995} \approx \frac{0.0000465}{0.010046} \approx 0.00463$$

$$\boxed{P(D|\text{positivo}) \approx 0.46\%}$$

b) Probabilidad de no estar infectado si test negativo:

$$P(\neg D|\text{negativo}) = \frac{0.99 \cdot 0.99995}{0.99 \cdot 0.99995 + 0.07 \cdot 0.00005} \approx \frac{0.9899505}{0.989954} \approx 0.9999965$$

$$\boxed{P(\neg D|\text{negativo}) \approx 99.9997\%}$$

—

Conclusiones

Cuando tenemos una prevalencia baja, un resultado positivo tiene probabilidad muy baja de indicar una infección real, es decir, tenemos muchos falsos positivos, también podemos observar que los resultados negativos son más confiables incluso con prevalencia más alta. Para determinar si damos más o menos importancia a la especificidad o a la sensibilidad depende de que queramos, si tenemos baja prevalencia sería mejor tener una especificidad alta para que los positivos sean confiables, mayor prevalencia, es mejor una sesibilidad alta.

3. La estacionariedad es una característica propia de las series de tiempo. Una serie de tiempo es una sucesión de estados (valores) cuyo orden refleja un proceso cronológico. La formación de una secuencia de ADN o ARN puede ser modelada dentro de este paradigma. Decimos que una serie o secuencia es estacionaria si cada uno de los estados presentes en la misma provienen de una misma distribución de probabilidad. A continuación se te presentarán 5 secuencias de ADN de hebra simple en las que tu tarea consistirá en listar cuáles son estacionarias y cuáles no. Para lograr lo anterior se te recomienda:

   a) Calcular la Esperanza y la Varianza de cada serie para distintas ventanas "temporales".

   b) Recordar, qué tipo de variable aleatoria nos permite modelar la frecuencia de los nucleótidos en una secuencia de ADN.

   c) Tu respuesta debe incluir, el código utilizado para responder la pregunta.

   Secuencias:

```
0. CGGAGACTTTTCCACTGTCGTCGGAGTAGTAAAATAACGGTACGTCTTAGTGTGCACCATCGACTCTTT
GTATTGCTCGTTAGGGGTCGCAGCCTCTTGTTAAGCCGTAATGGGTGATCCCCGCTCGTGAAACGGTGCGAT
CCTGTGATCTGTCAGTATCGAAGGAGTGAAAAAGCGATTGCTAGCCGAGGCGTACCGTG

1. CGGAGTCCCCCCCCACCGCCGCCGGTGCTGCATATCTACGGCACGCCCCAGTGTGCACCATCGACTCTTT
GTATTGCTCGTTAGGGGTCGCAGCCTCTTGTTAAGCCGTAATGGGTGATCCCCGCTCGTGAAACGGTGCGAT
CCTGTGATCTATTGATGTTAGCAACATAGCCGCATAGTTATTGATTACAATATCTTATA
```

2. CGGAGTCCCCCCCACCGCCGCCGGTGCTGCATATCTACGGCACGCCCCAGCGCGCTCCACCGACCCCCC
GCACCGCCCGCCAGGGGCCGCAGCCCCCCGCCATGCCGCTACGGGCGTCCCCCGCCCGCGATTCGGCGCGAC
CCCGCGACCCGCCTGCACCGTAGGAGCGTAATAGCGTCCGCCTGCCGAGGCGCACCGCG

3. CCCTACCCGGCAGACCCCTCCACGCCCCCCCGTAGCACCGGAACCAGATCCGCGGAGCGGGGAGGGAGG
CGGGGGGGGGAGGTCGGTCGGGGGTGGGGCGACAAGAGAGGAAACGGCAAGGGGAAGGGAGGAAAAGGAGTG
GAACGGAGGATTCATAGTAACTTGCGAAACGCACACGAAATGTGAACCATCACTACCAG

Metodología

La idea que vamos a utilizar es, usar una distribución multinomial para describir cuantas veces sale cada nucleótido en nuestro bloque y ventana. De manera intuitiva, tomamos una secuencia que va a ser una cadena, formamos una ventana deslizante y vamos calculando tanto esperanza (solo vamos a sacar las frecuencias relativas de cada letra) como varianza (que tanto se aleja la la frecuencia en una ventana especifica con respecto al promedio de todas las ventanas). Aquí el código de Python en Colab:

```python
from collections import Counter
import numpy as np
import pandas as pd

def clean_seq(seq):
    return ''.join([c for c in seq.upper() if c in {'A','C','G','T'}])

def sliding_windows(seq, w, step):
    L = len(seq)
    if w > L:
        return
    for i in range(0, L - w + 1, step):
        yield seq[i:i+w], i

def counts_and_freqs(window):
    c = Counter(window)
    n = len(window)
    freqs = {b: c.get(b, 0) / n for b in ['A','C','G','T']}
    return freqs

def analyze_sequence(seq, window_sizes=[25,50,75,100], step=None):
    seq = clean_seq(seq)
    L = len(seq)

    last_var_sum = -1

    for w in window_sizes:
        st = w // 4 if step is None else step
        freqs_list = [counts_and_freqs(win) for win, _ in sliding_windows(seq, w, st)]

        df_freqs = pd.DataFrame(freqs_list)
        mean_freq = df_freqs.mean()
        var_between = df_freqs.var(ddof=1)

        print('' + '='*40)
        print(f'Ventana w = {w}   (n_ventanas = {len(df_freqs)})')
        print('-'*40)
        print('Base | mean_freq | var_between')
        for b in ['A','C','G','T']:
            mf = mean_freq[b]
            vb = var_between[b]
            print(f"{b:>4}  | {mf:8.4f}   | {vb:10.6f}")
```

Como notas de este código, la parte de regresar un yield en vez de una lista directamente es para guardar memoria pero en si no cambia mucho, ademas, las ventanas en este caso se sobrelapan, por lo que no son independientes, hicimos pruebas con ventanas de tamaños 25,50,75,100 para obtener varias ventanas en cada caso, usamos también pasos de tamaño variable de un cuarto del tamaño de la ventana. La manera de calcular la varianza esta medio rara pero esta aplicando esta formula:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

Corriendo el código anterior en las diferentes secuencias de ADN obtenemos lo siguiente:

(a) Para la primera secuencia tenemos:

```
===== ANÁLISIS DE LA SECUENCIA 0 =====
======================================
Ventana w = 25   (n_ventanas = 30)
--------------------------------------
Base | mean_freq | var_between
   A  |   0.2227   |   0.009124
   C  |   0.2200   |   0.005269
   G  |   0.2840   |   0.002687
   T  |   0.2733   |   0.005195
======================================
Ventana w = 50   (n_ventanas = 13)
--------------------------------------
Base | mean_freq | var_between
   A  |   0.2092   |   0.002841
   C  |   0.2231   |   0.001256
   G  |   0.2846   |   0.001344
   T  |   0.2831   |   0.001990
======================================
Ventana w = 75   (n_ventanas = 7)
--------------------------------------
Base | mean_freq | var_between
   A  |   0.1943   |   0.001651
   C  |   0.2343   |   0.000762
   G  |   0.2800   |   0.000830
   T  |   0.2914   |   0.001033
======================================
Ventana w = 100   (n_ventanas = 5)
--------------------------------------
Base | mean_freq | var_between
   A  |   0.2020   |   0.000920
   C  |   0.2320   |   0.000270
   G  |   0.2860   |   0.000780
   T  |   0.2800   |   0.001850
```

Analizando, vemos que las frecuencias de los nucleótidos se mantienen relativamente estables al aumentar el tamaño de las ventanas, ademas, la varianza entre ventanas es relativamente chica, si sumamos la varianza de todos los casos con tamaño de ventana de 50, obtenemos $0.0028+0.0012+0.0013+0.0019 = 0.0072$ lo cual es relativamente bajo y podemos decir con relativa confianza que es estacionaria, ademas, a medida que hacemos mas grande la ventana esta varianza decrece bastante.

(b) Para la segunda secuencia tenemos:

```
===== ANÁLISIS DE LA SECUENCIA 1 =====
=======================================
Ventana w = 25   (n_ventanas = 30)
---------------------------------------
Base | mean_freq | var_between
   A  |   0.1973   |   0.006282
   C  |   0.2800   |   0.015448
   G  |   0.2480   |   0.003796
   T  |   0.2747   |   0.009681
=======================================
Ventana w = 50   (n_ventanas = 13)
---------------------------------------
Base | mean_freq | var_between
   A  |   0.1877   |   0.002703
   C  |   0.2754   |   0.008210
   G  |   0.2615   |   0.001431
   T  |   0.2754   |   0.004810
=======================================
Ventana w = 75   (n_ventanas = 7)
---------------------------------------
Base | mean_freq | var_between
   A  |   0.1733   |   0.001481
   C  |   0.2838   |   0.004013
   G  |   0.2686   |   0.000677
   T  |   0.2743   |   0.000999
=======================================
Ventana w = 100   (n_ventanas = 5)
---------------------------------------
Base | mean_freq | var_between
   A  |   0.1840   |   0.002930
   C  |   0.2760   |   0.004030
   G  |   0.2600   |   0.000600
   T  |   0.2800   |   0.000750
```

Igualmente tenemos que las frecuencias relativas no cambian mucho dependiendo del tamaño de la ventana, esto es buena señal, sin embargo, si sumamos la varianza de todos en la ventana de tamaño 50 obtenemos $0.0027 + 0.0082 + 0.0014 + 0.0048 = 0.0171$ que es mas del doble de varianza de la anterior, lo cual no es tan buena señal, podemos decir con baja confianza que esta secuencia puede no ser estacionaria pero como digo no es 100%.

(c) Para la tercera secuencia tenemos:

```
===== ANÁLISIS DE LA SECUENCIA 2 =====
=======================================
Ventana w = 25   (n_ventanas = 30)
---------------------------------------
Base | mean_freq | var_between
   A  |   0.1280   |   0.002913
   C  |   0.4933   |   0.011292
   G  |   0.2840   |   0.002687
   T  |   0.0947   |   0.003584
=======================================
Ventana w = 50   (n_ventanas = 13)
---------------------------------------
Base | mean_freq | var_between
   A  |   0.1231   |   0.000923
   C  |   0.5062   |   0.003959
   G  |   0.2846   |   0.001344
   T  |   0.0862   |   0.001426
=======================================
Ventana w = 75   (n_ventanas = 7)
---------------------------------------
Base | mean_freq | var_between
   A  |   0.1162   |   0.000516
   C  |   0.5257   |   0.002362
   G  |   0.2800   |   0.000830
   T  |   0.0781   |   0.000737
=======================================
Ventana w = 100   (n_ventanas = 5)
---------------------------------------
Base | mean_freq | var_between
   A  |   0.1200   |   0.000150
   C  |   0.5120   |   0.002870
   G  |   0.2860   |   0.000780
   T  |   0.0820   |   0.000370
```

En este caso la frecuencia relativa cambia un poco mas, especialmente si vemos el caso de la “C", sin embargo, el resto mantiene una frecuencia relativamente estable por lo que no es concluyente. Si nos pasamos al análisis de la varianza obtenemos que $0.0009 + 0.0039 + 0.0013 + 0.0014 = 0.0075$ que es casi la misma varianza que la primera por lo que decimos con algo de confianza que es estacionaria, aunque el nucleótido “C” parece variar bastante mas especialmente si vemos el caso de ventanas de tamaño 25 donde tiene una varianza de 0.011 indicando ventanas ricas en este nucleótido, 80/20 en este caso.

(d) Para la cuarta secuencia tenemos:

```
===== ANÁLISIS DE LA SECUENCIA 3 =====
========================================
Ventana w = 25   (n_ventanas = 30)
----------------------------------------
Base | mean_freq | var_between
   A  |   0.2747   |   0.016412
   C  |   0.2280   |   0.033341
   G  |   0.4120   |   0.044596
   T  |   0.0853   |   0.005046
========================================
Ventana w = 50   (n_ventanas = 13)
----------------------------------------
Base | mean_freq | var_between
   A  |   0.2646   |   0.012344
   C  |   0.2046   |   0.022877
   G  |   0.4508   |   0.028641
   T  |   0.0800   |   0.002467
========================================
Ventana w = 75   (n_ventanas = 7)
----------------------------------------
Base | mean_freq | var_between
   A  |   0.2514   |   0.008085
   C  |   0.1867   |   0.014933
   G  |   0.4933   |   0.010726
   T  |   0.0686   |   0.000737
========================================
Ventana w = 100   (n_ventanas = 5)
----------------------------------------
Base | mean_freq | var_between
   A  |   0.2640   |   0.009530
   C  |   0.1940   |   0.008430
   G  |   0.4680   |   0.010370
   T  |   0.0740   |   0.000930
```

En este caso vemos que las frecuencias se mueven bastante si cambiamos el tamaño de las ventanas, por ejemplo , en el nucleótido “G” pasa de una frecuencia de .41 en la primera a .49 en las ventanas de tamaño 75 esto ya es una mala señal. Si analizamos la varianza en el caso de las ventanas de tamaño 50, tenemos que la varianza es de $0.0123 + 0.0228 + 0.02864 + 0.0024 = 0.066$, casi 4 veces mas que el mas alto y casi 9 veces mas alto que los mas chicos; con esto, podemos concluir con alta confianza que esta secuencia no es estacionaria.

References

[1] Pérez, R. (s.f.). *El método Monte Carlo. Estimación del valor de PI.* GeoGebra. https://www.geogebra.org/m/cF7RwK3H