UNIVERSIDAD DE GRANADA

Python: Estructuras de Control. Aplicaciones en problemas biológicos Práctica 4 – Guión 9

Informática Aplicada a la Biología

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

DECSAI

Índice de contenido

1. Introducción

El objetivo de esta práctica es que el alumno se familiarice con las estructuras de control.

Es necesario insistir que las *transparencias de teoría* contienen todas las indicaciones necesarias para poder realizar la sesión de prácticas actual. Chequea el contenido de las mismas para poder trabajar cómodamente de forma autónoma.

Por último, se recuerda al estudiante a hacer uso de la ayuda integrada en el entorno como guía para investigar los elementos básicos de la interface y resolver la relación de ejercicios propuesta.

Recordamos que todos los ejercicios deben realizarse con el editor de Spyder, en el correspondiente fichero-py con nombre `P4_<num_ejercicio>.py` para no perder lo hecho (Ejemplo: `P4_1.py P4_2.py P4_3.py` para los ejercicios 1, 2 y 3 respectivamente y así en adelante).

Para el correcto desarrollo de esta sesión de prácticas, se deben realizar los siguientes pasos:

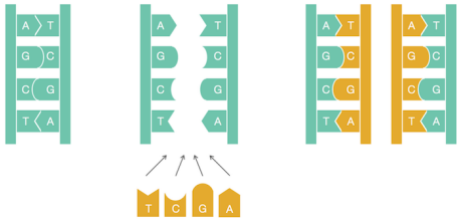
- **Debemos abrir Spyder.**
- **Elegimos como Carpeta de Trabajo el escritorio o una carpeta que hayamos creado para trabajar (este paso es importante).**
- **Recuerde usar `print` para que los cálculos (o resultados) sean visibles al ejecutar los scripts.**
- **Subiremos todos los archivos generados (incluso los no acabados) en la tarea creada en PRADO para cada sesión antes de irnos.**

Muy importante:

- La resolución de los problemas y actividades puede hacerse en grupo, pero la entrega durante las horas de prácticas es individual.
- La nota final de cada sesión dependerá del número de ejercicios entregados. Sin embargo, es preferible entregar solamente aquéllos que se sepan resolver.
- En caso de detectar algún tipo de copia, o una implementación que no se sepa explicar con claridad, se evaluará con un 0.
- Es muy importante que la asignatura se lleve al día para poder realizar los ejercicios propuestos en estos guiones.
- Debemos tener instalado *Anaconda* en casa tanto para realizar trabajo autónomo, como para la resolución del trabajo autónomo de Python.

2. Problemas biológicos: Búsqueda del "Dna-A Box"

Vamos a resolver un problema muy importante en el campo de la biología. Como sabemos, una de las tareas más importantes que se lleva a cabo en una célula Dos hilos de ADN se "desatan" o "desenrollan" durante la replicación y cada padre actúa como "plantilla" para la síntesis de una nueva hebra:



La replicación comienza en una zona denominada "Replication origin" o simplemente "ori" u "oriC". Asumiendo que un genoma tiene un único "ori" representado por una cadena de ADN o secuencia de nucleótidos {A, C, G, T}, nuestro objetivo será encontrar "ori".

La pregunta principal es ¿cómo sabe la célula dónde empezar a replicar? En concreto, se conoce que existe una proteína (DnaA) que se adhiere a un pequeño segmento conocido como "DnaA box". En nuestro caso, vamos a intentar realizar esta búsqueda para un genoma "pequeño" como el de una bacteria, concretamente el *Vibrio Cholerae* que provoca el cólera en los humanos.

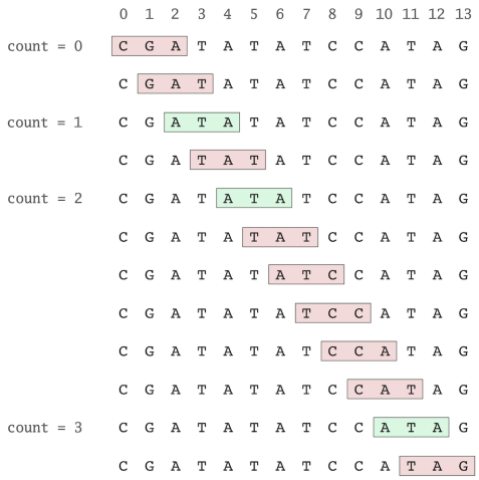
Puesto que no tenemos ni idea de en qué sección de "ori" se encuentran estos "DnaA-box", vamos a buscar algún "mensaje oculto" en dicha parte del genoma. Existen "palabras" frecuentes en el genoma "ori" de cualquier especie. Es lógico pensar que cuanto más aparezca una "subcadena", más probabilidad existe de la adhesión.

De acuerdo a lo anterior, podríamos decir por ejemplo que "ACTAT" es una cadena sorprendentemente frecuente de "ACA**ACTAT**GCAT**ACTAT**CGGGA**ACTAT**CCT".

Vamos a utilizar el término "*k-mer*" para una cadena de longitud "*k*", y buscaremos por tanto cuántas veces se repite un patrón dado (nuestro *k-mer*) en un texto (nuestro genoma). Ejemplo:

```
ContarPatron("ACTAT", "ACAACTATGCATACTATCGGGAACTATCCT") = 3.
ContarPatron ("ATA", "CGATATATCCATAG") = 3
```

Fijaos que para el segundo de los ejemplos no podemos utilizar la función "`count()`" que me proporciona el tipo de dato "*string*". Para poder chequear las apariciones solapadas tendré que usar una "ventana deslizante":



Antes de pensar en cómo utilizar esta ventana deslizante en un código de Python, debes determinar cómo realizar la comparación del "*k-mer*" con el texto, es decir, qué valores utilizarás en los índices. ¿Recuerdas el "*slicing*"? Te será de mucha ayuda aquí

1. Escribe una sentencia "`if`" para comprobar si una subcadena está en un texto mayor, y en caso afirmativo que sume 1 a un contador. En concreto, dados *kmer* = "ATA" y genoma = "CGATATATCCATAG" crea una orden que pueda comparar a partir de cualquier posición i-ésima del genoma, es decir, no uses valores absolutos si no relativos (variables).
2. Escribe en un comentario de código en qué posición deberemos parar la búsqueda de una ventana de 10 nucleótidos en un genoma de longitud 1000. Esto te indicará dónde empieza el último "*kmer*" que busques en tu cadena de ADN original. Pista: fíjate en el gráfico anterior.
3. Combina el ejercicio 1 dentro de un bucle que funcione como una ventana deslizante. Prueba con genoma GCGCG y el *kmer* GCG; la salida debería ser igual a 2.
4. Comprueba el funcionamiento del ejercicio con los siguientes valores reales (Vibrio Cholerae). Cuidado con los posibles "saltos de línea" al copiar la cadena que aparece abajo.

```
Genoma =
"""ATCAATGATCAACGTAAGCTTCTAAGCATGATCAAGGTGCTCACACAGTTTATCCACAACCTGAGTGGATGACATCAA
GATAGGTCGTTGTATCTCCTTCCTCTCGTACTCTCATGACCACGGAAAGATGATCAAGAGAGGATGATTTCTTGGCCATAT
CGCAATGAATACTTGTGACTTGTGCTTCCAATTGACATCTTCAGCGCCATATTGCGCTGGCCAAGGTGACGGAGCGGGATT
ACGAAAGCATGATCATGGCTGTTGTTCTGTTTATCTTGTTTTGACTGAGACTTGTTAGGATAGACGGTTTTTCATCACTGA
CTAGCCAAAGCCTTACTCTGCCTGACATCGACCGTAAATTGATAATGAATTTACATGCTTCCGCGACGATTTACCTCTTGA
TCATCGATCCGATTGAAGATCTTCAATTGTTAATTCTCTTGCCTCGACTCATAGCCATGATGAGCTCTTGATCATGTTTCC
TTAACCCTCTATTTTTTACGGAAGAATGATCAAGCTGCTGCTCTTGATCATCGTTTC"""

kmer = "TGATCA"
```

5. Escribe en un comentario del código cuál es el 2-mer más frecuente de la cadena GATCCAGATCCCCATAC

Lo que queremos ahora ampliar el ejercicio anterior, y buscar los kmers más frecuentes para un valor determinado de "k". Por ejemplo, la siguiente cadena CGATATATCCATAG para k = 3 contiene:

ATA --> 3 ATC --> 1 CAT --> 1 CCA --> 1 CGA --> 1 GAT --> 1
TAT --> 2 TCC --> 1 TAG --> 1

6. Implementa un código que resuelva el problema de generar el mapa de frecuencia para un k determinado. No desesperes, te guiaremos en la tarea.
    a. En primer lugar, crea un diccionario vacío: frecuencias = {}
    b. Recorre todo el texto haciendo "*slicing*" para el valor *k* y guarda como clave cada *kmer* y como valor un 0.
    c. Ahora actualiza la asignación directa para el valor 0 que hiciste anteriormente. Como pista, piensa que si el *kmer* está ya en tu diccionario de frecuencias debes sumarle uno, en lugar de asignarle un 0. Nota: para saber si un elemento está en un diccionario utiliza la orden "`if`" como "`clave in diccionario`"
    d. Prueba tu programa con los valores que indicamos antes: CGATATATCCATAG, k = 3
7. Última vuelta de tuerca: queremos dar como salida solamente los *kmers* más frecuentes calculados en el paso anterior. Para ello, debemos seguir los siguientes pasos:
    a. Saber cuál es el valor que más se repite. Utiliza la función "`max`" integrada en Python sobre los valores del diccionario (nota: `frecuencias.values()`)
    b. Crea una lista vacía llamada "*palabras*". Ahora, recorre todas las claves (*kmers*) de tu diccionario y comprueba si su valor es igual al máximo (apartado anterior 7.a). En caso afirmativo, añade el *kmer* a la lista "*palabras*".
8. Recupera el genoma de Vibrio Cholerae y calcula los kmers desde k=2 hasta k=9. ¿Te sorprenden los resultados que has obtenido? ¿Hay alguna relación especial entre los k-mers de "orden" 9? ¡¡Parece que finalmente has encontrado un "DNA-box"!!