Tarea 2 – Bioinfo

- Contesten 5 de las 6 preguntas; ustedes elijan cuál ignorar. 20 puntos por pregunta.
- Hay tres archivos que se usan en algunas preguntas; están en https://www.dropbox.com/sh/6m4qlic6cud3ygb/AADnGtltEp9RBOWZyPtCfumja?dl=0

P1. Retome las 5 proteínas encontradas en la tarea 1 (los primeros 5 matches que les dio BLAST).

a) Alinee sus secuencias (completas). Puede usar cualquiera de las herramientas onlines listadas en https://www.ebi.ac.uk/Tools/msa/ (especifique cuál!). Muestre el alineamiento. ¿Quedan alineados entre sí los segmentos que se alineaban con su nombre?

b) Corte, de cada secuencia, el segmento que se alinea con su nombre. Con las seis secuencias (agregando su nombre), haga un nuevo alineamiento.

c) Usando ese alineamiento, construya y dibuje ("a mano", es decir, sin usar software de HMM) un HMM, similar al ejemplo que vimos en clases.

d) Determine la secuencia de estados internos más probable en ese HMM para emitir su nombre.

P2. Considere las siguientes cuatro secuencias:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S1	A	A	G	A	C	G	G	C	A	A	A	C	A	T	C
S2	A	G	C	C	T	G	G	C	G	A	G	T	C	T	T
S3	C	G	A	T	T	A	C	C	A	C	G	T	C	G	G
S4	A	G	A	G	G	A	C	T	G	C	C	T	A	G	G

a) Aplicando a mano el método de máxima parsimonia, determine el mejor árbol filogenético.

Después de hacerlo, asuma que la secuencia 1 es un "outgroup" y explique

b) cómo quedaría en ese caso el parentezco entre las otras 3, y

c) qué se puede inferir sobre su secuencia ancestral.

Para las siguientes dos preguntas, trabaje en Python e instale http://biopython.org/wiki/Main_Page. No necesita interiorizarse demasiado en las opciones de Biopython; lo que sí nos interesa (como mínimo) es manipular objetos de tipo Seq[1]; además le interesará informarse más sobre las cosas específicas que las preguntas usan. Hay un extenso y detallado tutorial en http://biopython.org/DIST/docs/tutorial/Tutorial.html y también una API y documentación wiki listados en http://biopython.org/wiki/Documentation; en ellos pueden buscar la información relevante.

[1] Ver http://biopython.org/wiki/Seq, y/o http://people.duke.edu/~ccc14/pcfb/biopython/BiopythonSequences.html

P3. Lea el siguiente ejemplo de código provisto en el "Cookbook" de Biopython: http://biopython.org/wiki/Degenerated_Codons. Incluye una función, *degenerated_subseq*, que toma una secuencia codificadora (es decir, la parte de un gen que parte con AUG y termina con un codón de Stop), e imprime una subsecuencia que se salta todos los codones que pertenezcan a aminoácidos con menos de 'x' codones. Por ejemplo, si se pidió x=3, entonces un codón UUC no aparecería en el output, pues su aminoácido es codificado sólo por 2 codones (UUC y UUU); en cambio, un codón UCA sí aparecería, pues su aminoácido es codificado por 6 codones distintos. Nos interesa hacer algo diferente, pero este código y sus funciones auxiliares debieran resultar iluminadores para poder hacer lo que se pide, sin que tengan que revisar el resto de la documentación de Biopython.

a) Escriba una función que tome una secuencia codificadora y la aleatorice, reemplazando cada codón por un codón al azar escogido de entre los que codifican al mismo aminoácido. Por ejemplo, donde hay un UUC, pondremos ya sea UUC o bien UUU, con 1/2 de probabilidad para cada uno; donde haya un UCA, pondremos cualquiera de los 6 codones que codifican serina, con probabilidad 1/6 para cada uno.

b) Escriba una función que tome una secuencia codificadora y la reemplace por otra que codifica exactamente la misma proteína, pero que tiene el mayor porcentaje de G y C posible.

c) Aplique sus funciones al archivo cds.fna, y guarde los resultados en archivo cds_random.fna y cds_maxGC.fna (también en formato FASTA).

d) Sus funciones en a) y en b) están realizando *traducción inversa de proteínas*, pues están creando un gen a partir (en el fondo) de una secuencia de proteína dada. Sugiera una forma de realizar esta traducción inversa de modo que el gen resultante tenga (aproximadamente) un nivel de %GC ingresado como argumento. [No la programe, sólo describa su estrategia.]

Nota: en el código aparece un parámetro *table*, que especifica la versión de código genético; asuma *table=1* (el código genético estándar).

P4. Lea con detención la sección 20.1.13 del tutorial de Biopython, http://biopython.org/DIST/docs/tutorial/Tutorial.html#sec384. Incluye dos versiones de código para listar todos los ORFs[2] de una secuencia leída desde un archivo en formato FASTA.

a) Modifique el final del código (cualquiera de las dos versiones) para que sólo informe los tres ORFs con porcentaje de G+C más alto, y los tres con porcentaje de G+C más bajo[3],[4].

b) Aplique su función al archivo NC_005816.fna.

P5. El archivo plasmido.fna contiene el genoma de un cierto plásmido de *Escherichia coli*.

---

[2] Un ORF ("open reading frame") es un tramo de ADN que, si se lee de a tres letras, no contiene un codón de término (o tiene uno al final, después de muchos que no lo son). Es un "candidato a gen", cuando se está anotando un genoma. Ver http://en.wikipedia.org/wiki/Open_reading_frame.

[3] Puede simplemente contar las G y C del string, o bien usar la función GC (descrita en https://biopython.org/docs/1.75/api/Bio.SeqUtils.html#Bio.SeqUtils.GC) que toma un objeto Seq como argumento.

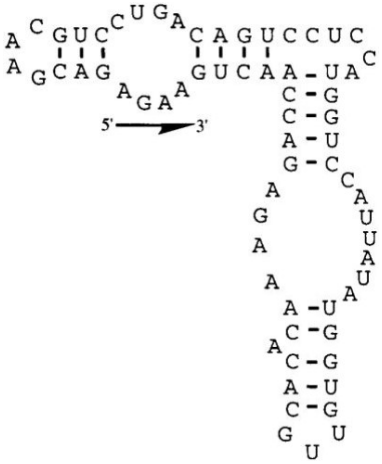
[4] Ojo que esto es el porcentaje de *nucleótidos* que son C ó G. Lo menciono porque un error frecuente de alumnos pasados fue calcular la frecuencia de C y G en la secuencia de amino ácidos.

a) Escriba un pequeño código en Python que lea un archivo de este tipo y genere un archivo similar, pero en que los nucleótidos hayan sido reordenados al azar (manteniendo la cantidad de cada tipo). Se sugiere usar la función *shuffle* del módulo *random*.

b) Use su función para generar una versión aleatorizada del genoma del plásmido.

c) Use Genmark[5], disponible en http://opal.biology.gatech.edu/GeneMark/, para buscar posibles genes. Aplíquelo tanto al genoma original del plásmido como a la versión aleatorizada que generó. Comente sobre las semejanzas o diferencias entre los resultados. (¿Hay más, hay menos? ¿Más cortos, más largos? ¿Por qué será?)

P6. El objetivo de esta pregunta es plegar (determinar estructura secundaria) para secuencias *circulares* de ARN; es decir, secuencias como la del dibujo, que no son una simple hebra, sino que el final se enlaza con el comienzo, de modo que, por ejemplo, "AUUCUAG" es la misma secuencia que "UCUAGAU" (simplemente la escribimos a partir de un punto distinto del círculo).



a) Escriba[6] una función (de preferencia en Python) que implemente el algoritmo de Nussinov (común y silvestre), y entregue una secuencia de paréntesis balanceados reflejando la estructura óptima. Al implementar el algoritmo de Nussinov haga un pequeño cambio para impedir que la primera y la última letra de la secuencia se puedan parear entre sí (es decir, en el "Max" de la esquina final de la tabla, no incluya el tercer caso).

b) Escriba una función que aplique Nussinov a secuencias circulares. Para esto lo que hacemos es considerar cada punto de la secuencia como posible punto de corte, y para cada una de las secuencias lineales resultantes aplicar Nussinov normal. Por ejemplo, si el input era AUUCUAG, calcularemos Nussinov para AUUCUAG, UUCUAGA, UCUAGAU, CUAGAUU, UAGAUUC, AGAUUCU, y GAUUCUA; después escojemos el que dio el máximo de bases pareadas. Gracias a la modificación que hicimos a Nussinov en la parte (a), tenemos garantizado que la primera y la última letra no se parearán entre sí (que sería erróneo, porque en realidad son vecinas en la forma circular).

c) Aplique su programa de (b) a la secuencia AUGGUACUGCCAUUCAAGAUGA[7] y a otra secuencia del mismo largo creada al azar[8]. Muestre la codificación de paréntesis y dibuje las estructuras resultantes.

d) Vaya a http://rna.tbi.univie.ac.at/cgi-bin/RNAWebSuite/RNAfold.cgi y úselo para generar estructuras para las mismas secuencias de la parte (c); asegúrese de chequear la opción "assume RNA molecule to be circular" en las "advanced options" (inicialmente ocultas). Muestre la

[5] http://en.wikipedia.org/wiki/GeneMark
[6] Recomiendo modificar algún código preexistente (especifique cuál); hay varios, y el que linkeé en el ppt, pensándolo mejor, no es muy útil; preferible sería http://biohackers.net/wiki/Nussinov.py que incluye traceback, o tal vez http://www.biocomp.unibo.it/piero/corso/note/node90.html que está en italiano pero además incluye la construcción de los paréntesis balanceados.
[7] Por si a alguien le interesa, esa secuencia es la protagonista del paper http://dx.doi.org/10.1016/j.jtbi.2007.07.010.
[8] No hace falta que escriba una función para generar secuencia al azar; con que tipee algo al tuntún basta.

estructura que la página entrega para cada una (incluyendo el dibujo). Comente sobre la similitud o diferencia entre esas estructuras y las que obtuvo usted con su algoritmo.

P6. ¿Alguno se parece? ¿Se detecta *algo* en el aleatorio? Etc.)