Universidad de Magallanes
Facultad de Ingeniería
Departamento de Ingeniería
Estructuras de Datos – Tarea 3

Christian Vásquez Rebolledo
27 de octubre al 17 de noviembre 2025

0. TRASFONDO DEL PROYECTO

Uno de los problema estudiados en Bioinformática es buscar secuencias de genes cortas dentro de una secuencia genética grande. El análisis de secuencias de ADN y proteínas implica la búsqueda de patrones específicos. Los algoritmos de búsqueda de patrones son cruciales en Bioinformática para tareas como la identificación de genes y el alineamiento de secuencias de proteínas. El análisis de secuencias suele comprender la identificación de una serie determinada de bases nucleotídicas, búsqueda de patrones o secuencias repetitivas e identificación de características genéticas y genómicas.

1. OBJETIVOS

- Implementar y manipular estructuras de datos abstractas como árboles y listas enlazadas para almacenar y gestionar los datos manipulados.
- Desarrollar habilidades en programación en lenguaje C, centrándose en el manejo de memoria, punteros y eficiencia algorítmica.
- Implementar un sistema de búsqueda de patrones utilizando árboles *tries*.

2. DESCRIPCIÓN DEL PROYECTO

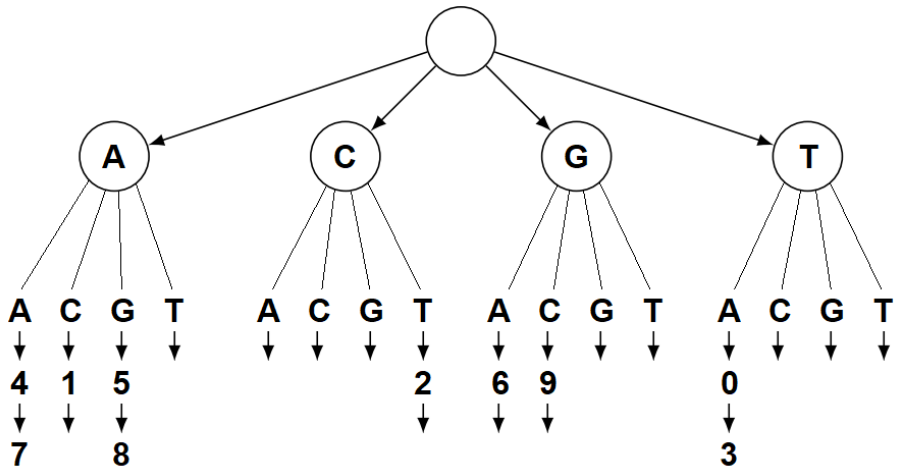
El problema a abordar consiste en buscar cuál es la frecuencia de repetición de un gen dentro de una secuencia genética e indicar las posiciones de dicho gen dentro de la secuencia. Considerando que una secuencia genética consiste en un string gigante (secuencia *S*, de tamaño *n*), compuesta por las letras 'A', 'C', 'G' y 'T', lo que interesa realizar es poder buscar strings cortos (gen *G*, de tamaño *m*) dentro de esta secuencia gigante.

La metodología a utilizar para resolver el problema es la siguiente: se considera un árbol 4-ario de *m* niveles de profundidad. Los nodos internos pueden tener 4 hijos, puesto que las posibles letras son 4, y las hojas contienen una lista de posiciones del gen *G* dentro de la secuencia *S*. Considere la siguiente estructura del nodo interno y de las hojas:

```
Nodo {
   Nodo hijoA;
   Nodo hijoC;
   Nodo hijoG;
   Nodo hijoT;
}

Hoja {
   ListaInt pos;
}
```

Por ejemplo, si se tiene la secuencia *S* = “TACTAAGAAGC”, al considerar genes de tamaño 2, el gen CC no está presente en *S*, en cambio, el gen AA se presenta 2 veces en *S*. Para este ejemplo en particular, se tiene el siguiente árbol:



Para usar esta estructura y buscar los genes presentes, es necesario hacer dos cosas:

1. Generar el árbol (con las listas vacías) para el tamaño del gen buscado
2. Leer un archivo que contiene la secuencia *S* y cargar las listas con las posiciones de los genes

Para cargar los genes en las lista se debe hacer lo siguiente:

1. Leer el gen i-ésimo
2. Descender en el árbol según las letras del gen i-ésimo
3. Una vez en la hoja, guardar la posición del gen(i) en la lista

Este ciclo se debe hacer para los *n* - *m* genes de la secuencia (el primer gen está en la posición 0).

3. NORMATIVAS DE CODIFICACIÓN EN C

Consistencia en el estilo del código:

- Usar nombres de variables y funciones descriptivos y seguir un esquema de nomenclatura coherente (snake_case o camelCase).
- Mantener la consistencia en la indentación y espaciado (uso de 4 espacios por nivel de indentación recomendado).

Modularidad:

- Dividir el código en funciones pequeñas y manejables que realicen una única tarea.
- Evitar funciones que superen las 50 líneas de código, a menos que sea estrictamente necesario.

Documentación:

- Incluir comentarios claros y precisos para explicar la funcionalidad de bloques de código complejos.
- Documentar cada función con comentarios que expliquen su propósito, parámetros de entrada, valores de retorno y posibles errores.

Gestión de Errores:

- Implementar mecanismos robustos para la gestión de errores, asegurándose de que todos los errores posibles sean capturados y manejados adecuadamente.

Uso eficiente de memoria:

- Evitar el uso innecesario de variables globales.
- Liberar memoria dinámicamente asignada cuando ya no sea necesaria para evitar fugas de memoria.

Optimización

- Escribir código optimizado en términos de tiempo y espacio, priorizando la eficiencia sin sacrificar la claridad.

4. EJEMPLO DE USO

Suponga que se necesita encontrar genes de tamaño 2 (sólo para el ejemplo).

<u>Creación del árbol:</u>
Debe inicializar el programa e indicarle que cree la estructura y asigne el tamaño indicado con *>bio start 2*
Tenga en cuenta que el tamaño de los genes será el equivalente a la altura de su árbol.

<u>Agregar secuencia *S*:</u>
Para leer *S* desde el archivo de texto “adn.txt”, el usuario debe ejecutar el comando *>bio read adn.txt*
El archivo debe contener *S* en una sola línea larga, terminada con un cambio de línea.

<u>Buscar un gen en particular:</u>
Para ingresar manualmente un gen *G* a buscar dentro de la secuencia *S*, el usuario puede usar el comando *>bio search* **
El programa debe revisar si el gen ingresado es parte de *S* y en caso de serlo debe indicar en qué posiciones se encuentra. Si el gen ingresado no es parte de *S*, el programa debe devolver el valor -1.
(El programa debe verificar que la cadena ingresada tiene el largo adecuado y está compuesta sólo por las letras ACGT)

<u>Buscar el gen *G* repetido más veces:</u>
Para el gen que aparezca más veces dentro de la secuencia S junto con sus posiciones, el usuario debe ejecutar *>bio max*
En caso de haber más de un gen que tenga el máximo de apariciones, se deben listar todos con sus respectivas posiciones.

<u>Buscar el gen G que aparece menos veces:</u>
Para el gen que aparece menos veces dentro de la secuencia S junto con sus posiciones, el usuario debe ejecutar *>bio min*
En caso de haber más de un gen que tenga el mínimo de repeticiones, se deben listar todos con sus respectivas posiciones.

Christian Vásquez Rebolledo

Ver todos los genes presentes:
Para ver todos los genes que aparecen en la secuencia junto a sus posiciones, el usuario puede ejecutar >*bio all*

Eliminación de estructuras:
Para eliminar las estructuras y cerrar el programa, el usuario debe ejecutar >*bio exit*

Flujo de ejecución:

```
>bio start 2
Tree created with height 2

>bio read adn.txt
Sequence S read from file

>bio search CC
-1

>bio search AA
4 7

>bio max
AA 4 7
AG 5 8
TA 0 3

>bio min
AC 1
CT 2
GA 6
GC 9

>bio all
AA 4 7
AC 1
AG 5 8
CT 2
GA 6
GC 9
TA 0 3

>bio exit
Clearing cache and exiting…
```

Para ejecutar pruebas, puede utilizar *S* de $n = 16$ y genes de $m = 2$.

Tenga en cuenta que al momento de la revisión se usará *S* de mayor longitud y genes de $m = 4$.

Christian Vásquez Rebolledo

5. ENTREGA Y EVALUACIÓN

- **Código fuente**: el código fuente correspondiente para la ejecución del programa solicitado. Debe estar bien documentado y seguir las normativas de codificación establecidas. *Debe incluir archivo Makefile correspondiente.* Debe poder ser compilado y ejecutado tanto en ambiente windows como linux.
- **Informe escrito**: se debe presentar un informe escrito, destacando las técnicas utilizadas y las decisiones detrás de su elección, así como los desafíos enfrentados y cómo se abordaron.

Para un mayor detalle de la forma en que se graduará esta tarea, revisar el documento que contiene la pauta de evaluación.

La tarea deberán realizarla en grupos de 4 personas (excepto dos grupos al azar que tendrán 5 integrantes).
Deben enviar por correo el repositorio que usarán para ir subiendo la tarea el día 27 de octubre, indicando los integrantes del grupo correspondiente.

La versión del código que se evaluará será la que esté presente en el repositorio designado para la entrega a las 23:59 del día lunes 17 de noviembre.
El informe escrito deben enviarlo por correo, teniendo como plazo la misma hora y fecha antes mencionadas.