Universidad Metropolitana

Departamento de Gestión de Proyectos y Sistemas.
Estructuras de datos. BPTSP06.
Trimestre 2425-3.

Proyecto 2: Proyecto de Bioinformática

Contexto preliminar

Para afrontar este proyecto, su equipo de trabajo deberá aprender sobre los **Hash Table**s o **Tablas de Dispersión**.

Tenga en cuenta que la solución al problema debe ser implementada con un **TDA Hash Table** con **manejo de colisiones**.

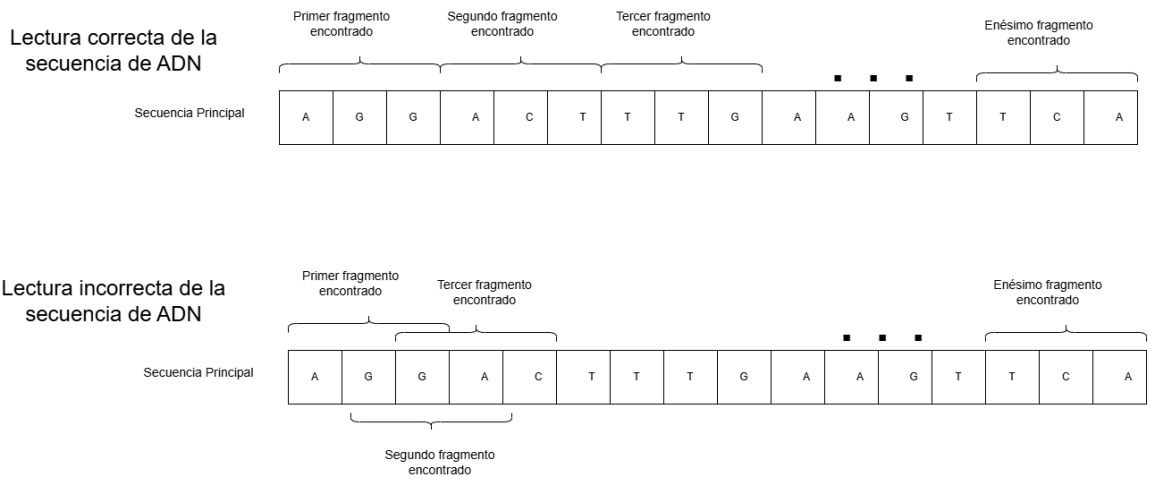
Problema

En bioinformática, la identificación de secuencias cortas y repetidas (patrones, o fragmentos) dentro de grandes cadenas de ADN es crucial para comprender la función génica, la regulación y la evolución. Este proyecto simulará una tarea básica de búsqueda de patrones, donde se deberán encontrar todas las ocurrencias de secuencias de ADN específicas dentro de una secuencia genética más larga, y luego analizar la frecuencia y ubicación de estas ocurrencias.

El objetivo de este proyecto es desarrollar una aplicación en **Java** capaz de identificar y analizar patrones dentro de secuencias genéticas, en otras palabras, orden exacto de los nucleótidos (adenina, timina, citosina y guanina) en una molécula de ADN. Usted deberá implementar estructuras de datos fundamentales como una **tabla hash (Hashtable)** y un **árbol binario de búsqueda (Binary Search Tree)** para gestionar eficientemente los datos y realizar las operaciones de búsqueda requeridas. Lo anterior no es limitativo, es decir, su equipo de trabajo podrá adoptar estructuras de datos auxiliares.

Requerimientos funcionales

A. **Cargar archivo:** El usuario puede seleccionar a través de **JFileChooser** un archivo **txt** (archivo de texto plano) para ser cargado en el sistema, el cual contará con la información **relativa** la secuencia de ADN principal (cadena larga de caracteres: adenina (A), guanina (G), citosina (C) y timina (T). Tras la lectura del archivo, el sistema deberá construir una tabla de dispersión con la información de los patrones presentes en la secuencia principal. Tome en cuenta que los fragmentos (patrones de ADN) a buscar deben tener una longitud de 3. La clave de la `Hashtable` deberá ser diseñada de forma que en la medida de lo posible se eviten colisiones en la tabla al momento de almacenar. Esto permitirá una recuperación rápida de todas las ocurrencias de un patrón dado. Es muy importante que considere que al comenzar a revisar la secuencia principal, el primer fragmento o patrón encontrado corresponderán a los caracteres 0, 1 y 2 de dicha secuencia; los caracteres 3, 4 y 5, corresponderán con otro patrón o fragmento, claro está si estos dos fragmentos son distintos. La siguiente imagen es ilustrativa.



B. **Lista de patrones almacenados**: el usuario dará la orden al programa para que muestre los patrones o fragmentos de ADN encontrados. El sistema deberá indicar por cada patrón: Frecuencia, Ubicaciones dentro de la secuencia principal. La lista deberá estar ordenada por la frecuencia de aparición de los fragmentos. Esta funcionalidad deberá responder a una complejidad de O(log n) o muy cercana.

C. **Buscar un patrón:** El sistema ofrecerá al usuario la posibilidad de mostrar la

información de un patrón en específico. Para lo cual, el sistema mostrará una lista desplegable ordenada por el primer carácter del fragmento. Una vez que el usuario seleccione una opción, el sistema mostrará la información correspondiente. Esta operación debe tener una complejidad de O(1).

D. **Identificar el patrón más frecuente o menos frecuente:** Para lo cual, se deberá implementar una solución que permita al sistema responder en un tiempo de O (log n) o al menos lo más cercano posible.

E. **Reporte de colisiones:** El sistema deberá generar un reporte en el que indique los patrones que, siendo tripletas distintas, generaron colisiones.

F. **Listar por cada aminoácido, las tripletas que la generan y su frecuencia de aparición en la secuencia principal.** Tome en cuenta que varias tripletas pueden generar un mismo aminoácido. Los aminoácidos son necesarios para la síntesis de proteínas. Además, la síntesis de proteínas es realizada por el ARN, por lo que debe considerar que las tripletas (fragmentos) están conformadas por los siguientes nucleótidos: adenina (A), guanina (G), citosina (C) y uracilo (U), que reemplaza a la timina. Así tenemos que, por ejemplo, las tripletas UUU y UUC (del ARN) sintetizan el aminoácido Fenilalanina y se corresponden con las tripletas TTT y TTC del ADN (secuencia principal). La siguiente tabla muestra la información correspondiente a los aminoácidos y las tripletas que los sintetizan. También deberán registrarse las tripletas correspondientes a inicio de una síntesis, a la parada de una síntesis y aquellas tripletas que no se correspondan con alguna tripleta válida para la síntesis de aminoácidos.

Primera Base	Segunda Base	Tercera Base	Aminoácido	Abreviatura (3 letras)	Abreviatura (1 letra)
U	U	U	Fenilalanina	Phe	F
U	U	C	Fenilalanina	Phe	F
U	U	A	Leucina	Leu	L
U	U	G	Leucina	Leu	L
U	C	U	Serina	Ser	S
U	C	C	Serina	Ser	S
U	C	A	Serina	Ser	S
U	C	G	Serina	Ser	S
U	A	U	Tirosina	Tyr	Y
U	A	C	Tirosina	Tyr	Y

U	A	A	STOP (Ocre)	-	-
U	A	G	STOP (Ámbar)	-	-
U	G	U	Cisteína	Cys	C
U	G	C	Cisteína	Cys	C
U	G	A	STOP (Ópalo)	-	-
U	G	G	Triptófano	Trp	W
C	U	U	Leucina	Leu	L
C	U	C	Leucina	Leu	L
C	U	A	Leucina	Leu	L
C	U	G	Leucina	Leu	L
C	C	U	Prolina	Pro	P
C	C	C	Prolina	Pro	P
C	C	A	Prolina	Pro	P
C	C	G	Prolina	Pro	P
C	A	U	Histidina	His	H
C	A	C	Histidina	His	H
C	A	A	Glutamina	Gln	Q
C	A	G	Glutamina	Gln	Q
C	G	U	Arginina	Arg	R
C	G	C	Arginina	Arg	R
C	G	A	Arginina	Arg	R
C	G	G	Arginina	Arg	R
A	U	U	Isoleucina	Ile	I
A	U	C	Isoleucina	Ile	I
A	U	A	Isoleucina	Ile	I
A	U	G	Metionina (Inicio)	Met	M
A	C	U	Treonina	Thr	T
A	C	C	Treonina	Thr	T
A	C	A	Treonina	Thr	T
A	C	G	Treonina	Thr	T
A	A	U	Asparagina	Asn	N

A	A	C	Asparagina	Asn	N
A	A	A	Lisina	Lys	K
A	A	G	Lisina	Lys	K
A	G	U	Serina	Ser	S
A	G	C	Serina	Ser	S
A	G	A	Arginina	Arg	R
A	G	G	Arginina	Arg	R
G	U	U	Valina	Val	V
G	U	C	Valina	Val	V
G	U	A	Valina	Val	V
G	U	G	Valina	Val	V
G	C	U	Alanina	Ala	A
G	C	C	Alanina	Ala	A
G	C	A	Alanina	Ala	A
G	C	G	Alanina	Ala	A
G	A	U	Ácido Aspártico	Asp	D
G	A	C	Ácido Aspártico	Asp	D
G	A	A	Ácido Glutámico	Glu	E
G	A	G	Ácido Glutámico	Glu	E
G	G	U	Glicina	Gly	G
G	G	C	Glicina	Gly	G
G	G	A	Glicina	Gly	G
G	G	G	Glicina	Gly	G

Requerimientos técnicos

1. La solución debe ser implementada, fundamentalmente, usando tablas de dispersión y árboles binarios de búsqueda.

2. Puede utilizar cualquier otra estructura auxiliar de ser necesario para mejorar los tiempos de respuesta del programa. Sin embargo, **NO podrá utilizar librerías para la implementación del tipo de dato abstracto**.

3. La aplicación debe ofrecer una interfaz gráfica al usuario. No se permite la entrada ni la salida de información por consola.

4. El programa debe poder cargar un archivo de texto para la lectura de datos. Para ello, es requerido el uso del componente **JFileChooser**.

5. Debe documentar el proyecto con **Javadoc**.

6. Junto al programa, cada equipo deberá presentar un **Diagrama de clases** *(arquitectura detallada)* que explique la solución obtenida.

7. Un archivo ejemplo será proporcionado por separado: archivo.txt.

Consideraciones

- Los equipos pueden tener como **máximo 3 personas.**
- Deberá entregar su carpeta de Proyecto NetBeans en un archivo ZIP en el aula del curso. Además, deberá cargar la dirección del repositorio.
- La carpeta deberá incluir un archivo llamado “readme”, el cual debe incluir una breve descripción del proyecto (una o dos oraciones que permitan entender el contenido sin consultar el enunciado), los nombres de los integrantes del grupo y la dirección web del repositorio de GitHub donde se desarrolló el proyecto.
- Los proyectos que no tengan interfaz gráfica, serán calificados con **0 (cero)**.
- Los proyectos que no tengan documentación interna alguna, serán calificados con **0 (cero)**.
- Los proyectos que implementen el tipo de dato abstracto usando librerías, serán calificados con **0 (cero)**.
- Los proyectos que sean iguales o parecidos, incluyendo aquellos que sean iguales o parecidos a los que se puedan encontrar en internet o generar con herramientas derivadas de ChatGPT, serán calificados con **0 (cero)**.
- Los programas que “no compilen” o “no corran”, serán calificados con **0 (cero)**
    - Si una entrega no incluye los archivos de proyecto de NetBeans, será considerado como que ***no compila***. El equipo debe estar seguro de que el archivo consignado contiene la totalidad del proyecto.
    - Si una entrega usa librerías en JAR y no incluye los archivos .jar en el zip de la entrega, será considerado como que ***no compila***.
    - Si un programa muestra una interfaz gráfica, pero la cierra antes de poder realizar ninguna acción útil, será considerado como que ***no corre.***
    - Si un programa muestra una interfaz gráfica de usuario sin funcionalidad alguna, será considerado como que ***no corre***.
- El enlace a GitHub en el *readme* ***debe*** conducir a un repositorio público válido. Los registros de Github que tengan todos los commits de un solo integrante o en un solo

día, serán considerados como ***no válidos***, pues no reflejan la participación activa y significativa de todos los integrantes del equipo.
- Los proyectos **podrán** ser sometidos a defensa, es decir, el facilitador y/o preparadores convocarán al equipo para una revisión **solo** si lo considera necesario.
- Se podrá considerar funcionalidad adicional para puntos adicionales siempre y cuando se identifique claramente, mantenga los estándares de calidad del resto del programa y sea razonable para el problema planteado.

Criterios de Evaluación (Rúbrica)

- **Resolución de problemas de ingeniería**. Capacidad para comprender, definir y resolver problemas de análisis de ingeniería en el campo de estudio pertinente, con el uso de conocimientos básicos y avanzados de métodos analíticos modernos.

    - Comprensión y definición de problemas de ingeniería. 30%
    - Aplicación de Métodos Analíticos Modernos. 15%
    - Resolución de Problemas con Conocimientos Básicos y Avanzados. 25%
- *Fiabilidad*: Capacidad para mantener un nivel especificado de prestaciones cuando se usa bajo ciertas condiciones:
    - *Madurez*: El programa no presenta fallas originadas por errores de programación, análisis o diseño. **(10%)**
    - *Tolerancia a fallos*: El programa responde adecuadamente al manejo inadecuado del usuario; es decir, mantiene su correcto funcionamiento aun cuando el usuario introduzca datos erróneos o manipule inadecuadamente las interfaces de usuario. **(10%)**
- *Usabilidad*: Capacidad del proyecto para ser entendido, aprendido, usado y al mismo tiempo, ser atractivo para el usuario, cuando se usa bajo condiciones específicas:
    - *Comprensibilidad*: El programa ofrece una interfaz de fácil comprensión, facilitando su aprendizaje y correcta utilización. El programa emite mensajes de alerta cuando se introducen valores erróneos. Existen elementos informativos que indican al usuario como operar el programa. **(5%)**
    - *Capacidad de ser atractivo*: El diseño de la interfaz de usuario, esto es: disposición de controles, esquema de colores, utilización de cajas de diálogo y demás elementos gráficos; es atractivo para el usuario. **(5%)**

Rúbrica Detallada

Resolución de problemas de ingeniería. Capacidad para comprender, definir y resolver problemas de análisis de ingeniería en el campo de estudio pertinente, con el uso de conocimientos básicos y avanzados de métodos analíticos modernos.

Criterio	Excelente (4 pts)	Satisfactorio (2-3 pts)	Insatisfactorio (0-1 pt)
1. Comprensión y definición de problemas de ingeniería	Identifica y define con clara precisión todos los problemas de bioinformática planteados (identificación de patrones, análisis de frecuencia, mapeo a aminoácidos), demostrando un entendimiento profundo del contexto.	Identifica la mayoría de los problemas, pero la definición carece de cierta claridad o exhaustividad en algunos aspectos.	No identifica o define los problemas básicos de ingeniería relacionados con la búsqueda y análisis de patrones de ADN de manera efectiva.
2. Aplicación de métodos analíticos modernos	Aplica los métodos analíticos modernos (Hash Tables con manejo de colisiones, Binary Search Tree) de manera óptima y eficiente para resolver los problemas, demostrando un conocimiento avanzado de sus principios y limitaciones.	Aplica los métodos analíticos, pero presenta algunas limitaciones en la eficiencia o en la comprensión de sus principios, lo que puede llevar a soluciones subóptimas.	No aplica los métodos analíticos modernos (Tablas Hash, Árbol Binario de Búsqueda) de manera efectiva o los utiliza incorrectamente.

3. Resolución de problemas con conocimientos básicos y avanzados	Resuelve todos los problemas planteados utilizando una combinación sólida y experta de conocimientos fundamentales de estructuras de datos (Tablas Hash, Árboles Binarios de Búsqueda) y conceptos avanzados de bioinformática (mapeo codones-aminoácidos).	Resuelve la mayoría de los problemas, pero con algunas dificultades en la aplicación de los conocimientos, lo que puede resultar en soluciones incompletas o con errores menores.	No resuelve los problemas básicos de manera efectiva, o la solución es incorrecta/incompleta debido a la falta de aplicación de los conocimientos requeridos.

Criterio de Evaluación	Subcriterio	Niveles de Desempeño	Descripción	Puntuación
Fiabilidad	Madurez	Excelente	El programa no presenta fallas y mantiene su funcionamiento incluso con entradas incorrectas.	10%
		Satisfactorio	El programa presenta fallas menores o errores en casos límite, pero se mantiene estable en general.	7%
		Necesita mejorar	El programa presenta fallas frecuentes y genera errores por consola.	3%
		Inaceptable	Las fallas del programa son tan graves que no puede ser usado de ninguna manera útil.	0%
	Tolerancia a fallos	Excelente	El programa maneja entradas incorrectas y errores de usuario de manera robusta y sin interrupciones.	10%
		Satisfactorio	El programa maneja la mayoría de las entradas incorrectas, pero puede bloquearse en algunos casos.	7%
		Necesita mejorar	El programa se bloquea o falla con entradas incorrectas o errores de usuario.	3%
		Inaceptable	No hay acciones que no generen errores.	0%

Criterio de Evaluación	Subcriterio	Niveles de Desempeño	Descripción	Puntuación
Usabilidad	Comprensibilidad	Excelente	La interfaz es intuitiva y fácil de usar, pudiendo entenderse cómo realizar cada una de las funcionalidades sin requerir explicación externa.	5%
		Bueno	La interfaz es generalmente intuitiva, pero algunos controles pueden ser confusos en cuanto a cómo deberían ser usados.	4%
		Satisfactorio	Aunque la mayoría de los controles son confusos, la interfaz logra comunicar efectivamente cómo llegar a cada una de las funcionalidades pedidas.	3%
		Necesita mejorar	Algunas de las funcionalidades se pueden realizar intuitivamente, pero es difícil encontrar cómo deben realizarse las demás.	2%
		Deficiente	La disposición de la interfaz es confusa, interfiriendo con la usabilidad general del programa	1%
		Inaceptable	El programa no tiene una interfaz entendible.	0%
	Capacidad de ser atractivo	Excelente	La interfaz tiene un diseño atractivo y moderno, con una buena disposición de controles y esquemas de colores.	5%
		Bueno	La interfaz contiene elementos que pueden dificultar la lectura, pero está generalmente bien diseñada.	4%
		Satisfactorio	La interfaz es funcional, pero la disposición de los controles y el diseño general es mejorable.	3%
		Necesita mejorar	El diseño de la interfaz interfiere ligeramente con la usabilidad y legibilidad del programa.	2%
		Deficiente	El diseño de la interfaz es descuidado e interfiere seriamente con la usabilidad y legibilidad del programa.	1%
		Inaceptable	El diseño de la interfaz es tan descuidado que el programa es inutilizable.	0%