LABORATORIO 3: BÚSQUEDAS EN GENOMA HUMANO

Objetivos

Entender el comportamiento de los algoritmos búsquedas

Implementación y de algoritmos de búsqueda lineal

Implementación y adaptación de algoritmos de búsqueda binaria

Entender los problemas de escalabilidad

Herramientas proporcionadas

En el moodle de la asignatura dispone de herramientas para la corrección automática de entregas de código. Estas herramientas son sensibles al uso de mayúsculas/minúsculas, por lo que deberá respetar escrupulosamente las instrucciones dadas en este guión respecto al nombre de clases y métodos. Si tiene dudas sobre cómo usar esta herramienta puede consultar alguno de los vídeo-tutoriales (*screencast*) disponibles:
http://www.lab.dit.upm.es/~prog/screencast/SubirProyectosEclipseAMoodle.flv

Introducción

Un problema muy común en biomedicina y bioinformática es encontrar ocurrencia de una secuencia en otra secuencia. Algunos ejemplos: ensamblado de genomas, búsqueda de genéticas, comparaciones genómicas, análisis de proteínas, análisis genómicos/DNA de personas en particular.

Diversas organizaciones aplican todas estas técnicas para construir un mapa del genoma humano (http://genome.ucsc.edu) que se emplean en múltiples métodos de análisis médico que analizan enfermedades de origen genético[1].

Un ejemplo sencillo: donde podemos encontrar la secuencia GATNACA en el genoma humano? Lo primero que necesitamos es soporte que nos describa el genoma humano. Las organizaciones que hemos mencionado desarrollan mapas generales del genoma humano (distintas personas y razas de personas tienen variaciones de ese genoma de referencia). Un mapa de referencia podemos encontrarlo y descargarlo de: http://hgdownload.cse.ucsc.edu/downloads.html#human. Ellos

[1] Genoma humano: genómica, genética y aplicaciones en medicina. Rafael Oliva. Medicina Clínica. Vol. 116. Núm. 17. 12 Mayo 2001.
http://apps.elsevier.es/watermark/ctl_servlet?_f=10&pident_articulo=13013757&pident_usuario=0&pcontactid=&pident_revista=2&ty=144&accion=L&origen=zonadelectura&web=http://zl.elsevier.es&lan=es&fichero=2v116n17a13013757pdf001.pdf

proporcionan modelos de referencia de disposición pública, de los que destacamos un modelo en particular: Feb. 2009 (hg19, GRCh37). Este modelo es especialmente grande y por ello las aplicaciones no suelen buscar en el genoma entero (del orden de varios GB), sino que se centran en cromosomas concretos asociados al tipo de enfermedades que se estudian. Por ello es posible descargar o estudiar cromosomas en particular.

El modelo de genoma humano descompuesto en función de cromosomas se pueden encontrar en: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/. En concreto encontramos los ficheros de nombre chr*.fa.gz (por ejemplo chr22.fa.gz) que son la secuencia para cada cromosoma, en formato FASTA, y comprimidos. Cada fichero comprimido ocupa del orden de varios MB. El formato FASTA es un formato de texto que representan las secuencias de nucleótidos en texto (http://en.wikipedia.org/wiki/FASTA_format). Los ficheros de los cromosomas descomprimidos son texto con una forma como:

```
>chr19
GATCACAGAGGCTGGGCTGCTCCCCACCCTCTGCACACCTCCTGCTTCTA
ACAGCAGAGCTGCCAGGCCAGGCCCTCAGGCAAGGGCTCTGAAGTCAGGG
TCACCTACTTGCCAGGGCCGATCTTGGTGCCATCCAGGGGGCCTCTACAA
GGATAATCTGACCTGCAGGGTCGAGGAGTTGACGGTGCTGAGTTCCCTGC
ACTCTCAGTAGGGACAGGCCCTATGCTGCCACCTGTACATGCTATCTGAA
GGACAGCCTCCAGGGCACACAGAGGATGGTATTTACACATGCACACATGG
CTACTGATGGGGCAAGCACTTCACAACCCCTCATGATCACGTGCAGCAGA
```

En esto ficheros aparecen importantes secuencias de N (que representan largas secuencias de cualquier tipo de acido nucleótido). Son ficheros de texto con una primera línea que no tiene interés para lo que veremos en esta práctica. Si los descargases, borra esa primera línea.

En esta práctica aplicaremos algoritmos de búsqueda para poder localizar secuencias específicas, dentro de los modelos de genoma de referencia. De una forma parecida lo podríamos hacer para realizar el análisis de ADN para personas en particular, y no para un modelo de genoma de referencia que es como lo vamos a hacer.

Actividades a desarrollar

Descarga del proyecto desde GitHub (Trabajo previo)

La estructura del proyecto a desarrollar, así como algunas de las clases necesarias se encuentran disponibles en el repositorio de la asignatura en GitHub: https://github.com/ALED-UPM/Tema2

Antes del inicio de la sesión de laboratorio, deberá:

1. **Configurar EGit** para acceder al repositorio Tema-2 de la asignatura.
2. **Clonar el repositorio Tema2**.
3. **Importar el proyecto ALED-lab3** a su Eclipse. Este proyecto incluye algunos ficheros muy grandes (ficheros FASTA de cromosomas del genoma humano).

Si tienes problemas, haz la descarga de un fichero zip de moodle que incluye el proyecto comprimido, utiliza esta alternativa si te diese problemas GitHub.

4. **Explorar la clase *LectorFASTA* y los ficheros del directorio cromosomas.**
5. **Contestar a las preguntas 1-3.**

Implementar el método *buscar* con búsqueda lineal (Trabajo en laboratorio)

La clase *LectorFASTA* incluye un constructor que permite inicializar el contenido de la clase a partir de un fichero FASTA. El método:

```
public LectorFASTA(String ficheroCromosoma)
```

Crea un objeto instancia de *LectorFASTA* inicializándolo a partir de un fichero cuyo nombre fija el parámetro. La clase *LectorFASTA* incluye dos campos:

```
protected byte[] contenido;
protected int bytesValidos;
```

contenido es un array de bytes en donde cada byte representa el valor de un nucleótido ('A','T','G','N', ...). *bytesValidos* nos dice cuantos bytes del array *contenido* son válidos. Siempre se cumple *bytesValidos <= contenido.lenght*. Pero puede suceder que algunas posiciones del final de *contenido* no tengan información válida. Los ficheros FASTA son extensos, incluyen algunos caracteres que no nos interesan (por ejemplo saltos de línea). Antes de leer el fichero no podemos saber cuantos de esos caracteres nos encontraremos, y por tanto no se puede saber exactamente cual debería ser el tamaño concreto del array. Solo se puede saber al final, y para conseguir un tamaño de array exacto deberíamos leer el fichero dos veces, o hacer una copia de arrays muy extensos. Para conseguir un algoritmo mas eficiente en tiempo de ejecución, creamos un array que puede ser un poco mas grande lo necesario.

En esta clase está pendiente de implementar el método *buscar*. Este método tiene como cabecera:

```
public List<Integer> buscar(byte[] patron)
```

Su argumento es una secuencia de bytes que representan el patrón de nucleótidos que vamos buscando en el cromosoma. Ese patrón puede no encontrarse en ninguna parte del cromosoma, puede encontrarse en varios lugares, o puede encontrarse en un único sitio. El método devuelve una lista de enteros que estará vacía para el primer caso, incluirá varios índices que nos indican las posiciones concretas de la secuencia en la que se encuentra el patrón (esas posiciones deben ir de menor a mayor), y para el último caso la lista incluye un único índice de posición. Los índices son posiciones del array *contenido*. A partir de índices de posición de la secuencia, y de un tamaño, el método:

```
public String getSecuencia(int posicionInicial, int tamano)
```

nos permite extraer una secuencia concreta a partir de una posición (que es una posición de *contenido*).

Si el patrón del parámetro incluye 'N' en alguna posición, la secuencia puede tener cualquier valor ('A', 'T', ...). Para simplificar, no tenemos porqué cuidar otros valores como 'K','S','Y' ... que representan también múltiples valores.

En esta primera parte emplearemos un algoritmo lineal (fuerza bruta). Buscaremos el patrón empezando en cada una de las posiciones de *contenido*. Partiendo de una posición, compararemos el patrón con una secuencia de contenido que empieza en la posición y tiene como tamaño la del patrón. Lo vemos con algún ejemplo.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
G	A	T	N	A	C	A									

No ajusta el patrón en la posición 1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
	G	A	T	N	A	C	A								

Ajusta el patrón en la posición 2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
		G	A	T	N	A	C	A							

No ajusta el patrón en la posición 3

Miramos si ajusta en las posiciones 4,5,6,7,8 y

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
								G	A	T	N	A	C	A	

No ajusta el patrón en la posición 9

El patrón cuadra en la posición 2 de la secuencia. El método *buscar* nos devolverá una lista con un único elemento de valor 2.

Para implementar *buscar*, podemos emplear el método *compara*:

```
private boolean compara(byte[] patron, int posicionAComparar)
  throws FASTAException {
```

```
    if (posicionAComparar+patron.length > bytesValidos)
       throw new FASTAException("patron se sale del fichero FASTA");
    boolean fallo=false;
    for (int j=0; j < patron.length; j++)
       if (patron[j] != contenido[posicionAComparar+j] && patron[j] != 'N')
           fallo=true;
    return !fallo;
}
```

Este método devuelve *true* si el patrón cuadra con la posición *posicionAComparar* del atributo *contenido* de *LectorFASTA.* Sino devuelve *false*.

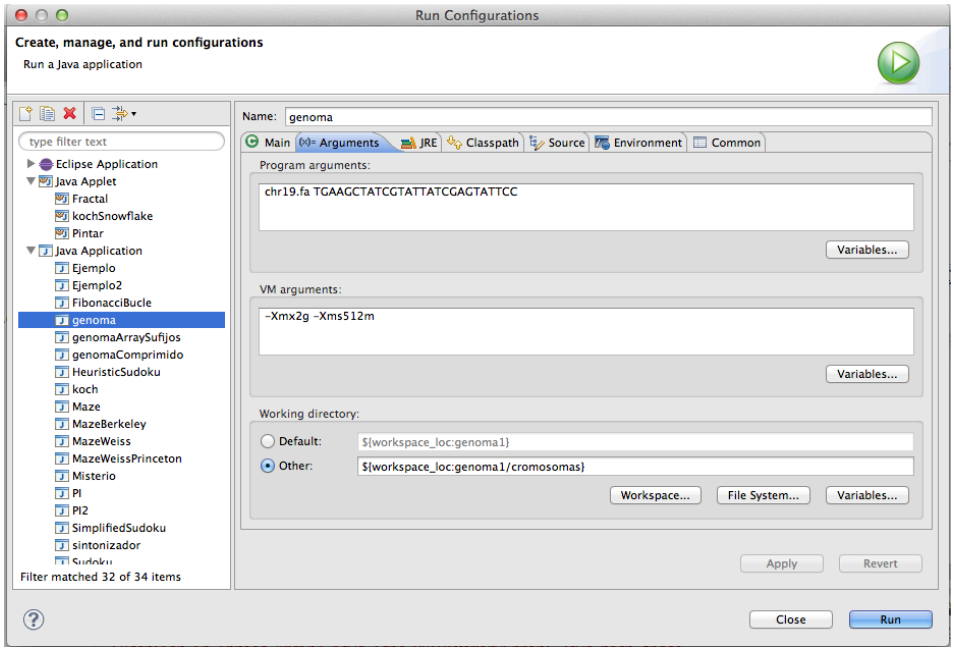
Para hacer las pruebas desde eclipse debemos configurar las ejecuciones teniendo en cuenta lo siguiente:

1. Los métodos *main* que incluyen las clases con las que trabajamos suponen que reciben los nombres de los ficheros que incluyen la secuencia sobre la que buscar. Y también reciben como argumenta la secuencia a buscar. Por tanto, debemos incluir esos parámetros en la configuración de *parámetros de programa* de ejecución.

2. Las pruebas que se van a hacer ocupan grandes espacios de memoria. Según tengamos configurado eclipse, las aplicaciones Java se inicializan con mayor o menor memoria. Si haciendo las pruebas se detecta a excepción *OutOfMemoryError*, podemos aumentar los espacios de memoria con las siguientes opciones de máquina virtual java: -Xmx2g -Xms512m. La primera configura el tamaño máximo de memoria de ubicación de nuevos objetos, en este caso 2GB; la segunda opción fija el tamaño inicial de ese espacio (en este ejemplo, inicialmente son 512MB).

3. Dado que los programas leerán ficheros, y esos ficheros se encuentran en el directorio *cromosomas*. La forma mas sencilla de trabajar es que la aplicación java se ejecute en ese directorio, y de esa forma los nombres de fichero pueden ser simplemente los nombres y no son necesario ningún tipo de camino de carpetas.

La siguiente figura es un ejemplo de estas configuraciones en la carpeta de argumentos.



Realizar las siguientes pruebas:

1. Ejecutar la búsqueda del patrón 'TGAAGCTATCGTATTATCGAGTATTCC' sobre el fichero ref100.fa. Cuantas veces aparece ese patrón?

2. Cuanto tiempo tarda en ejecutarse la búsqueda anterior. Puedes emplear la función *System.currentTimeMillis()*, que devuelve la fecha actual en milisegundos; guardamos la fecha antes de ejecutar *buscar*, y después, la diferencia entre el valor guardado y la fecha actual nos dice cuanto ha tardado.

3. Que tamaño tiene el fichero ref100.fa? No es un cromosoma completo, es solo un segmento de un fichero de cromosoma.

4. Cuanto tarda en encontrar el mismo patrón en el fichero chr19.fa? Debes tener en cuenta que cargar ese fichero puede suponer al menos 30 segundos y menos de 2 o tres minutos . Cuanto tarda la búsqueda con este segundo fichero (ese valor debe ser solo la búsqueda y no debe incluir la construcción o la carga del fichero).

5. **Contestar a las preguntas 4-7.**

Mejora de los tiempos medios de la búsqueda lineal (Trabajo en laboratorio)

Si repasas la implementación que os damos del método *compara,* este es mejorable para reducir los tiempos de ejecución.

La implementación de *compara* se recorre sistemáticamente todo el contenido del patrón. Pero si el patrón no cuadra en las primeras posiciones, ¿para que seguir comparando?

Modifica el método *compara* para mejorar los tiempos medios.

Realizar las siguientes pruebas:

1. Realizar la mismas prueba del punto 4 del apartado anterior, y compara los tiempos de ejecución.
2. **Contestar a las preguntas 8-10.**

Implementación basada en búsquedas binarias (Trabajo después de laboratorio)

La clase *LectorFASTAConArraySufijos* extiende la clase *LectorFASTA* para soportar una búsqueda binaria. Esta clase incluye el array *sufijos* que tiene tantas entradas como *bytesValidos* tiene *contenido*. Cada entrada representa un sufijo de la secuencia de caracteres que representa *contenido*. *sufijos* incluye tantas entradas como tamaño tenga *contenido,* porque una entrada representa toda la secuencia de contenidos desde el principio hasta el final, otra entrada representa todos menos el primer carácter, otra representa todos menos los dos primeros, … y una última representa solo el último carácter. Esa tabla nos permite buscar una secuencia en una posición cualquiera de *contenido*, de forma binaria, porque *LectorFASTAConArraySufijos* ordena alfabéticamente los *sufijos*. *sufijos* no contiene las subsecuenticas como tal, porque sino su tamaño sería: *1 + 2 + 3 + 5 + … + bytesValidos-1 + bytesValidos=bytesValidos*(bytesValidos+1)/2.* Si pensamos en ficheros del orden de 60MB (esto es lo que ocupan buena parte de los cromosomas) estaríamos pensando en un array *sufijos* que ocuparían varios miles de TB (y ningún ordenador de los que utilizamos actualmente dispone de esa memoria, ni tan siquiera de disco). *sufijos* es un array de enteros en donde cada entero es un índice de *contenido*. La posición de *sufijos* con valor 0, representa el sufijo con todos los caracteres, con valor 1 es la sub-cadena con todos los caracteres menos el primero, y la posición con valor *bytesValidos-1*, representa la sub-cadena con el último carácter únicamente.

Pongamos un ejemplo. Supongamos un fichero de solo 7 caracteres:

i	0	1	2	3	4	5	6
contenido	b	a	n	a	n	a	$

El valor de sufijos antes de ordenar es (0,1,2,3,4,5,6):

sufijos	**i**
banana$	0
anana$	1
nana$	2
ana$	3
na$	4
a$	5
$	6

Después de ordenar será (6,5,3,1,0,4,2):

sufijos	i
$	6
a$	5
ana$	3
anana$	1
banana$	0
na$	4
nana$	2

Podemos buscar alfabéticamente mediante algoritmos binarios, teniendo en cuenta que *sufijos* es un array ordenado, que incluye punteros a *contenidos*, donde se encuentra esa sub-cadena.

Realizar las siguientes pruebas:

1. Explorar contenido del fichero ref2.fa. En el método *main* de la clase *LectorFASTAConArraySufijos* toma los argumentos de entrada y se construye un lector de FASTA; si solo hay un argumento construye un *LectorFASTAConArraySufijos* a partir del fichero que se pasa como argumento. El constructor construye los sufijos a partir del contenido. Después de construir el lector con la sentencia:
```
LectorFASTA cr = new LectorFASTAConArraySufijos(args[0],sufijos);
```
podemos incluir la sentencia:
```
((LectorFASTAConArraySufijos) cr).imprime();
```
que imprimirá el contenido de la tabla de sufijos. Prueba *LectorFASTAConArraySufijos* imprimiendo la tabla de sufijos. Borra la sentencia que imprime, porque para ficheros grandes su ejecución es excesivamente larga. Después de ejecutar esta prueba refresca el contenido de la carpeta *cromosomas* (pulsando con el botón derecho del ratón sobre la carpeta *cromosomas*). Aparecerá el fichero *suf_ref2.fasuf* que ha creado la ejecución del programa, explora ese fichero y verás que es el contenido de la tabla *sufijos*. Después de la primera ejecución, podemos inicializar *sufijos* a partir de este fichero sin que tener que ordenar *sufijos*.

Hay que hacer una nueva implementación del método *buscar*, que redefina el método *buscar* de la superclase *LectorFASTA*. Ese algoritmo se debe hacer con una búsqueda binaria, teniendo en cuenta que *sufijos* está ordenado.

Realizar las siguientes pruebas:

1. Ejecutar la implementación de *LectorFASTAConArraySufijos* empleando como parámetros de programa “ref100.fa TGAAGCTATCGTATTATCGAGTATTCC”.
2. Medir los tiempos de ejecución que lleva la búsqueda, y el tiempo que lleva para el fichero ref100.fa la ordenación de sufijos. Debemos medir solo la ordenación y no incluir el tiempo que se lleva la carga de fichero.
3. Intenta hacer la prueba del punto 1, pero con el fichero chr19.fa. Pero **no esperes mucho a que termine**, porque es probable que te duermas antes de que termine.

El algoritmo de ordenación que emplea *LectorFASTAConArraySufijos* es un algoritmo bastante óptimo que normalmente sería de complejidad *N log N*. Pero el problema está en que para ordenar hay que comparar sufijos que pueden ser muy largos (para *chr19.fa* algunos sufijos pueden tener longitudes próximas a 60MB). Eso hace que el algoritmo de ordenación sea del orden N^2 log N. Teniendo en cuenta el tiempo que tarda en ordenarse ref100.fa y que ese fichero son 100 KB, cuanto podemos esperar que tarde en terminar *chr19.fa* que son 60 MB y que la ordenación tiene una complejidad N^2 log N.

Entregas

Instrucciones para la entrega **durante la sesión de laboratorio**:

- Las **preguntas** planteadas deben **entregarse en papel al final de la sesión**. Las preguntas 1-3 estarán resueltas antes del laboratorio, y el resto podrán resolverse en el laboratorio.
- El **código fuente y la documentación** de las clase ***LectorFASTA***. Deberá entregarse en el moodle de la asignatura antes del final de la sesión de laboratorio.
- El nombre del fichero que incluya los ficheros requeridos no debe contener tildes ni espacios.
- En el contenido del fichero comprimido de la entrega, el código fuente debe estar en el directorio relativo: "ALED-lab3/src/es/upm/dit/aled/lab3/".

Instrucciones para la entrega a realizar **después del laboratorio**:

- El **código fuente y la documentación** de las clases ***LectorFASTAConArraySufijos*** **y *LectorFASTA*** deberán entregarse en el Moodle de la asignatura **antes de las 23:59 del 2 de noviembre**, siguiendo las mismas instrucciones que la entrega de laboratorio respecto al nombre y ruta de los ficheros. **No incluir el directorio *cromosomas*.**
- Para la evaluación de la entrega **sólo se considerará el último envío realizado**.
- **No se aceptarán entregas fuera de plazo** bajo ningún concepto.

AVISO MUY IMPORTANTE

Se recuerda a los alumnos que el trabajo es individual, y que la copia de entregas supondrá el **suspenso en la asignatura de forma automática**, **tanto para quien copia como para quien se deja copiar**.

No está permitido:

- Realizar este trabajo en grupo.
- Copiar el trabajo de otro alumno, ni permitir la copia del propio trabajo, ni siquiera parcialmente.
- Usar código publicado sin citar el origen.

Responda a las siguientes preguntas según se le indica en el enunciado del laboratorio. Entregue al profesor esta hoja con sus respuestas antes de la sesión.

Nombre:_____ **DNI/NIE**:_____

1. Localizar el método *buscar* en la clase *LectorFASTA.* ¿Que debe hacer este método?

2. ¿Qué argumentos presupone que tendrá el método *main* de *LectorFASTA*?

3. Explora el contenido de los ficheros de cromosomas. ¿Que contenido y tamaño tienen?

4. ¿Cuantas veces aparece ese patrón 'TGAAGCTATCGTATTATCGAGTATTCC' sobre el fichero ref100.fa?

5. ¿Cuánto tiempo tarda la búsqueda sobre *ref100.fa*?

6. ¿ Cuánto tiempo tarda la búsqueda sobre *chr19.fa*?

7. ¿Qué complejidad tiene el algoritmo de búsqueda empleado? Contrasta los tiempos de las dos pruebas, y los tamaños de los ficheros, y di si los resultados se ajustan mas o menos.

8. ¿Cómo optimizas la ejecución de *compara*?

9. ¿Cuál es la diferencia entre las dos ejecuciones para *ref100.fa* y para *chr19.fa*?

10. ¿Cuantas iteraciones ejecutarían de media el bucle de *compara* en la versión original y en la nueva?