Programación de Sistemas y Concurrencia

Control Bloque 1, Temas 1 - 2
Curso 2023-2024

Computadores C + Informática C + Software C

- Para que un apartado puntúe, la función correspondiente tiene que compilar correctamente.
- En la puntuación se valorará, además de la corrección de la solución, su la claridad y rendimiento.
- Solo se pueden utilizar las plantillas y el enunciado. No se pueden usar diapositivas, ejercicios resueltos, vídeos o cualquier otro material disponible en internet.

Descripción del sistema

En este ejercicio vamos a desarrollar un módulo simplificado para la manipulación de cadenas de ADN. Una cadena de ADN (Chain) es una lista de fragmentos de ADN. Para simplificar el problema, un fragmento es una cadena de caracteres de longitud FRAGLENGTH+1, donde los primeros FRAGLENGTH caracteres representan nucleótidos (adenina (A), citosina (C), guanina (G) y timina (T)) y el último carácter es el de fin de cadena ('\0'). Para FRAGLENGTH igual a 4, "AAAA" o "GTCC" son fragmentos válidos.

Vamos a utilizar las siguientes definiciones de tipos, que se encuentran en el fichero ADN.h.

```
#define FRAGLENGTH 4
typedef struct SNode *Chain;

typedef struct SNode{
   int id;
   char fragment[FRAGLENGTH+1];
   Chain next, prev;
} Node;
```

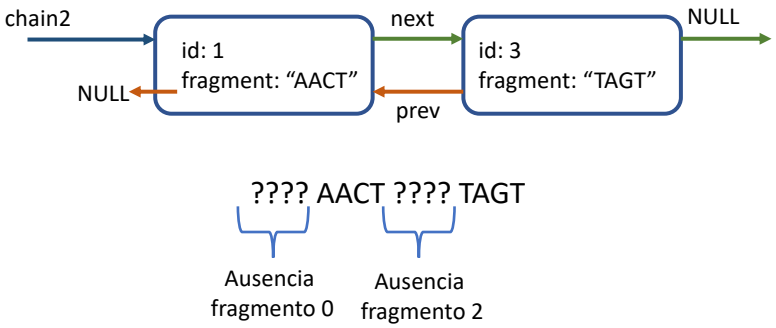
El tipo registro struct SNode almacena la información de un fragmento que consiste en su id, que nos permite identificar su posición dentro la cadena, y fragment que es la cadena de caracteres. Además, los campos next y prev son referencias al fragmento siguiente y anterior en la cadena, respectivamente. Una cadena de ADN (Chain) es una **lista doblemente enlaza que está ordenada por el id** de cada fragmento **de forma creciente**. El tipo Chain se define como un puntero al tipo struct SNode.

En la siguiente imagen se muestra un ejemplo de cadena de ADN compuesta de 4 fragmentos. Observa que el campo next de cada fragmento nodo apunta al siguiente fragmento con id mayor, excepto en el caso del último fragmento cuyo campo next apunta a NULL. De forma similar, el campo prev de cada fragmento apunta al fragmento anterior con id menor, excepto en el caso del primer fragmento cuyo campo prev apunta a NULL.

Entre las operaciones que ofrece este módulo está la inserción de un fragmento en la cadena con un id dado (addFragment). La inserción debe generar una cadena sigue estando ordenada de forma creciente por el campo id. Si se quiere insertar un fragmento con un id que ya está en la cadena, lo que se hace es actualizar el fragmento de ese id con el nuevo fragmento. Por ejemplo, si en la cadena anterior se pide insertar el

fragmento “GGGG” con id 8, en la cadena resultante el fragmento con id 8 contendrá “GGGG” en vez de “CCCC”, tal y como se muestra en la siguiente figura.

El módulo también incluye una operación que muestra por pantalla la secuencia de ADN completa (show), es decir, muestran los nucleótidos desde el fragmento con id 0 al fragmento con mayor id existente en la cadena. Para los fragmentos en ese rango de id que no estén explícitamente en la cadena se mostrará 4 interrogaciones (“????”). Para entenderlo mejor, usaremos la siguiente cadena como ejemplo:



La cadena incluye los fragmentos con id 1 y 3 y, por tanto, show debe mostrar los nucleótidos de los fragmentos del intervalo 0 a 3. Como resultado, muestra la cadena de caracteres ????AACT????TAGT. Observa que como faltan los fragmentos 0 y 2, la secuencia de ADN incluye ???? para representar la ausencia del fragmento 0 y del fragmento 2.

Para la realización del ejercicio se proporciona el fichero de cabecera (ADN.h) y un fichero para probar la implementación (Driver.c). Este último fichero no hace pruebas exhaustivas, si quieres probar otros casos puedes modificar la función main. Al final de este documento está la salida esperada del programa principal y un anexo con las funciones para lectura/escritura en ficheros y manipulación de cadenas de texto.
Se pide implementar en el fichero ADN.c las siguientes funciones de forma eficiente:

```
/* (0,25 puntos) Crea una cadena vacia. */
void emptyChain(Chain *chain);

/* (0,75 puntos) Muestra por pantalla la cadena de ADN suponiendo que empieza con el fragmento 0. En concreto,
muestra la secuencia de nucleótidos resultante de unir TODOS los fragmentos del 0 al último id. Si falta algún
fragmento entre el id 0 al último id, muestra ???? (4 interrogaciones) en la posición de ese fragmento para indicar que
no está. */
void showChain(Chain chain);

/* (1 punto) Elimina todos los fragmentos de la cadena, liberando toda la memoria
   dinamia asociada. */
void destroyChain(Chain *chain);

/* (1,5 puntos) Añade el fragmento con el id dado a la cadena. Los id tienen que ser  >=0. Si el id es <0 entonces
devuelve 0, en otro caso devuelve 1. Si el id ya está en la cadena, entonces reemplaza el fragmento existente por el
nuevo. Recuerda que la cadena tiene sus fragmentos en orden creciente de id. */
```

```
int addFragment(Chain *chain, char *frag, int id);

/* (1,5 puntos) Crea una cadena con la información contenida en el fichero binario. Si la cadena que recibe no está
vacía, primero destruye su contenido. El formato del fichero almacena para cada fragmento primero su id (un int) y
luego los FRAGLENGTH caracteres que representan los nucleótidos. Observa que el carácter '\0' de los fragmentos
no está en el fichero. */
void loadFromFile(char *filename, Chain *chain);

/* (1,5 puntos) Almacena en un fichero binario la información de la cadena. El formato es el mismo que en el apartado
anterior: primero está el id y luego los 4 caracteres
que representan los nucleótidos. Observa que el carácter '\0' de cada fragmento no se almacena en el fichero.
Observa también que solo se guardan en el fichero la información que están explícitamente en la cadena. */
void saveToFile(char *filename, Chain chain);

/* (1,5 puntos) Elimina de la cadena todas las ocurrencias del fragmento (secuencia de nucleótidos) y devuelve el
número de fragmentos eliminados. */
int cut(Chain *chain, char *frag);

/* (2 puntos) Elimina de la cadena el fragmento con el id dado y los anteriores a éste que sean consecutivos. Si el
fragmento no existe devuelve 0 y no modifica la cadena. Si el fragmento existe devuelve el número de fragmentos que
se han eliminado.
Por ejemplo, si en la cadena tienen los fragmentos con id 1->4->5->8 y se pide extraer el 5, la cadena resultante es 1
8 (se elimina el fragmento 4 porque es consecutivo al 5) y la función devuelve 2.
 Si en la cadena están los fragmentos con id 1->4->5->8 y se pide extraer el 9, la cadena no se modifica y la función
devuelve 0. */
int extract(Chain *chain, int id);
```

A continuación, se muestra la salida al ejecutar el programa principal incluido en Driver.c

```
Fragment added

Fragment added

Fragment added

????????????????AAAA????????????AAAA????????????AAAA

Unable to add fragment

Fragment added

Fragment added

????ACGT????????AAAAACGT????????AAAA????????????AAAA

Chain destroyed

Chain loaded from file:
```

????ACGT????????AAAAACGT????????AAAA????????????AAAA

Cut AAAA - removed fragments: 3

????ACGT????????????ACGT

Chain loaded from file:

????ACGT????????AAAAACGT????????AAAA????????????AAAA

Extract fragment 5 - extracted fragments: 2

????ACGT??????????????????????????AAAA????????????AAAA

Extract fragment 1 - extracted fragments: 1

??????????????????????????????????AAAA????????????AAAA

ANEXO

Los prototipos de las funciones para manipular strings (incluidas en <string.h>) son:

char* strcpy(char *s1, char *s2): Copia los caracteres de la cadena s2 (hasta el carácter '\0', incluido) en la cadena s1. El valor devuelto es la cadena s1.

int strcmp(char *s1, char *s2): Devuelve 0 si las dos cadenas son iguales, <0 si s1 es menor que s2, y >0 si s1 es mayor que s2.

size_t strlen(const char *s): Calcula el número de caracteres de la cadena apuntada por s. *Esta función no cuenta el carácter '\0' que finaliza la cadena*.

Los prototipos de las funciones para **manipulación de ficheros binarios** (incluidos en <stdio.h>) son los siguientes (se dan por conocidos los prototipos de las funciones de <stdlib.h> que necesites, como free o malloc):

FILE *fopen(const char *path, const char *mode): Abre el fichero especificado en el modo indicado ("rb"/"wb" para lectura/escritura binaria y "rt"/"wt" para lectura/escritura de texto). Devuelve un puntero al manejador del fichero en caso de éxito y NULL en caso de error.

int fclose(FILE *fp): Guarda el contenido del buffer y cierra el fichero especificado. Devuelve 0 en caso de éxito y -1 en caso de error.

LECTURA/ ESCRITURA TEXTO

int fscanf(FILE *stream, const char *format, ...): Lee del fichero *stream* los datos con el formato especificado en el parámetro *format*, el resto de parámetros son las variables en las que se almacenan los datos leídos en el formato correspondiente. La función devuelve el número de variables que se han leído con éxito.

int fprintf(FILE *stream, const char *format, …): Escribe en el fichero *stream* los datos con el formato especificado en el parámetro *format*. El resto de parámetros son las variables en las que se almacenan los datos que hay que escribir. La función devuelve el número de variables que se han escrito con éxito.

LECTURA/ ESCRITURA BINARIA

unsigned fread(void *ptr, unsigned size, unsigned nmemb, FILE *stream): Lee *nmemb* datos, cada uno de tamaño *size* bytes, del fichero *stream* y los almacena en la dirección apuntada por *ptr*. Devuelve el número de elementos leídos.

unsigned fwrite(const void *ptr, unsigned size, unsigned nmemb, FILE *stream): Escribe *nmemb* datos, cada uno de tamaño *size* bytes, en el fichero *stream*. Los datos se obtienen desde la dirección apuntada por *ptr*. Devuelve el número de elementos escritos.