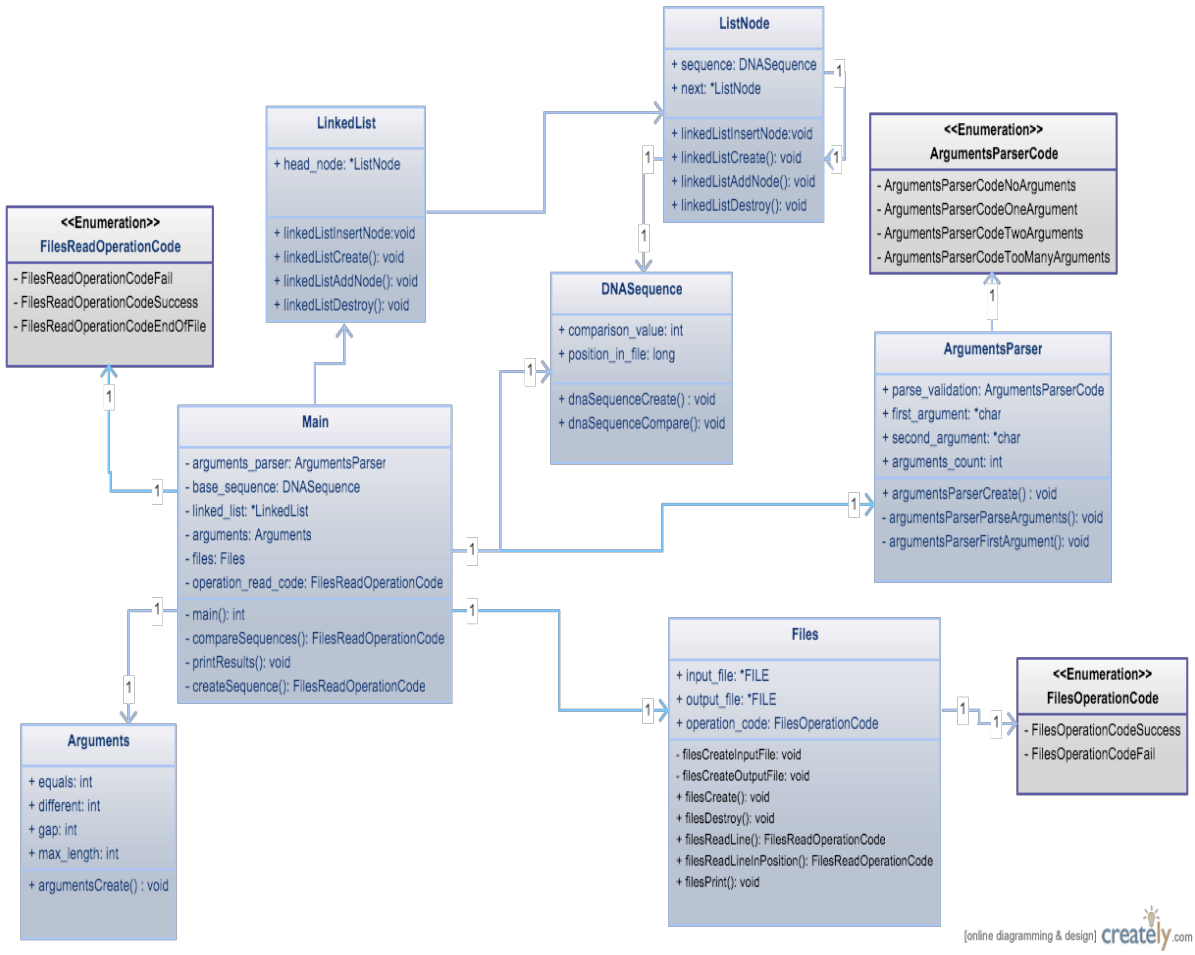
Informe

En el presente trabajo práctico tiene como objetivo comparar secuencias de ADN con el algoritmo de Smith-Waterman y listarse en orden descendente según su comparación con una secuencia de ADN base. Las cadenas estaban almacenadas en un archivo cuyo nombre era pasado como primer parámetro en la ejecución del programa.
Para la lectura de las secuencias se plantearon 2 tipos de datos: ArgumentsParser y Files. ArgumentsParser se encarga de reconocer los parámetros pasados en la ejecución del programa y Files se encarga de tomar dichos parámetros y abrir los archivos correspondientes para la lectura y, en algunos casos, para la escritura.
Cómo las cadenas de caracteres de las secuencias no podían guardarse en memoria, Files presenta dos métodos para lectura: uno para leer la siguiente línea; y otro para leer una línea en una posición específica indicada por parámetro.
Con Files se lee la primer línea, esta línea contiene los parámetros iguales, diferentes, gap y longitud máxima de una cadena de ADN y se crea un tipo de dato llamado Arguments, que contiene los valores necesarios para el algoritmo Smith-Waterman.
Se lee nuevamente una línea utilizando Files, esta línea se reconoce como la secuencia base. Con la línea leída, que es la primer secuencia de ADN tomada como base, creamos un tipo de dato llamado DNASequence, el cual contiene el valor de la comparación con la cadena base, y la posición en el archivo de entrada en donde se encuentra su cadena de caracteres. DNASequence también tiene el algoritmo de comparación entre dos cadenas.
Antes de leer las siguientes líneas de caracteres, se crea una lista (LinkedList
) vacía cuyos nodos (ListNode) contienen un tipo de dato DNASequence y un puntero al siguiente nodo.
Se continua leyendo cadenas de caracteres del archivo de entrada con Files, se crean tipos de datos DNASequence con las líneas leídas y se los comparan con la secuencia base mediante el método de DNASequence llamado dnaSequenceCompare.
La mayor complejidad del problema era resolver la comparación, ya que había que generar una matriz que no podía estar alocada en el stack, sino tenía que estar en el heap:
1 - Primero ello se crea un arreglo de 2 dimensiones.
2 - Se aloca memoria para las filas del arreglo, que son tantas como el largo de la cadena a comparar + 1 (El algoritmo pide una fila completa de 0), y su tipo de dato es punteros a enteros (Que en definitiva van a ser las columnas).
3 - Por cada fila, se aloca memoria para las columnas de la matriz, son tantas como el largo de la cadena base + 1, y el tipo de dato de las columnas es entero.
4 - Se inicializa toda la matriz en 0.
5 - Por último, en base a los argumentos del algoritmo de Smith-Waterman, se consigue el valor de las diferentes celdas de la matriz, empezando por la celda (1,1), siguiendo por la (1,2)...(1,x) y luego por la (2,1)(2,2)...(2,y), siendo x= el número de columnas e y el número de filas.
6 - Se toma el valor de la matriz (x,y), que es el valor buscado y se lo asocia al tipo de dato DNASequence creado en base a la secuencia que se esta comparando contra la secuencia base.
Una vez que ya se obtuvo el valor de comparación de una cadena con la cadena base, se agrega la cadena a la lista enlazada utitlizando el método linkedListAddNode. Este método crea un nuevo nodo (ListNode) y lo agrega ordenadamente a la lista. El orden es

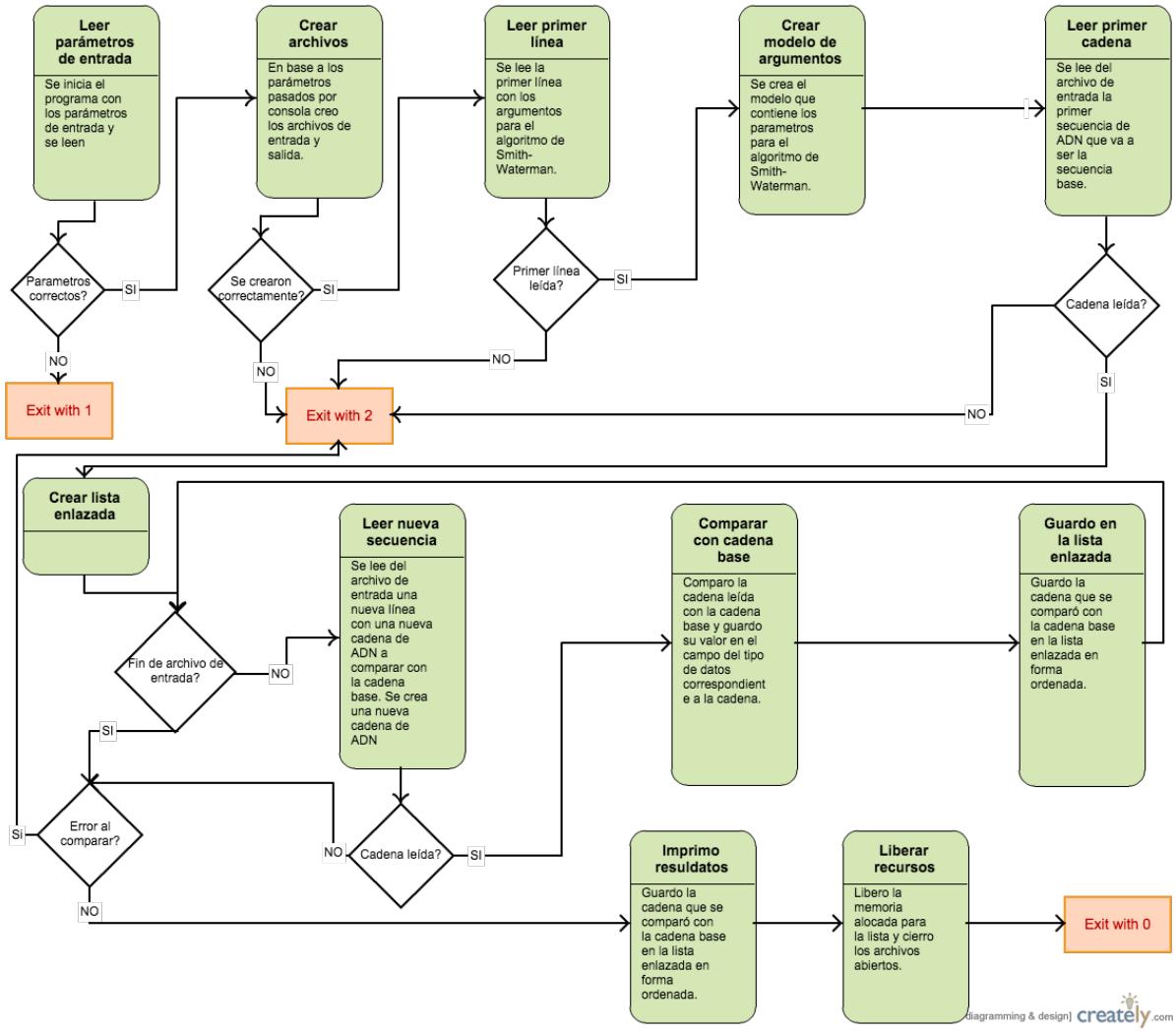
descendente para el valor de la comparación con la cadena base y, en caso de ser iguales estos valores, ascendente para el caso de la posición de la cadena en el archivo de lectura.
Por último, se imprime donde corresponda la cadena. Lo primero que se hace, es posicionarse en el archivo de entrada en el lugar donde se encuentra la cadena. Se lee la cadena y luego se llama al método: filesPrint.
Como en el caso de la escritura se podía dar por consola, escritura en archivo o en salida estándar, el método filesPrint encapsula dicho comportamiento, dependiendo de si un archivo de salida fue pasado como segundo parámetro o no, e imprime donde corresponda la cadena de caracteres.
Para finalizar se eliminan los recursos alocados en la lista y se cierran los archivos abiertos en la creación de un tipo de datos Files.

Esquema del diseño

El siguiente esquema intenta dar a entender los tipos de datos abstractos distinguidos en la resolución del problema. Se aclara que no intenta ser un diagrama de clases (Aunque se utilize UML y pareciera ser tal), simplemente se intenta describir los tipos abstractos de datos y las funciones asociadas a grandes rasgos.



A continuación se deja un diagrama de flujo para entender cómo fue pensada la resolución por pasos:


Leer parámetros de entrada
Se inicia el programa con los parámetros de entrada y se leen
Crear archivos
En base a los parámetros pasados por consola creo los archivos de entrada y salida.
Leer primer línea
Se lee la primer línea con los argumentos para el algoritmo de Smith-Waterman.
Crear modelo de argumentos
Se crea el modelo que contiene los parametros para el algoritmo de Smith-Waterman.
Leer primer cadena
Se lee del archivo de entrada la primer secuencia de ADN que va a ser la secuencia base.
Parametros correctos?
SI
NO
Exit with 1
Se crearon correctamente?
SI
NO
Exit with 2
Primer línea leída?
SI
NO
Cadena leída?
SI
NO
Crear lista enlazada
Fin de archivo de entrada?
NO
SI
Leer nueva secuencia
Se lee del archivo de entrada una nueva línea con una nueva cadena de ADN a comparar con la cadena base. Se crea una nueva cadena de ADN
Error al comparar?
SI
NO
Cadena leída?
NO
SI
Comparar con cadena base
Comparo la cadena leída con la cadena base y guardo su valor en el campo del tipo de datos correspondiente a la cadena.
Guardo en la lista enlazada
Guardo la cadena que se comparó con la cadena base en la lista enlazada en forma ordenada.
Imprimo resuldatos
Guardo la cadena que se comparó con la cadena base en la lista enlazada en forma ordenada.
Liberar recursos
Libero la memoria alocada para la lista y cierro los archivos abiertos.
Exit with 0
[diagramming & design] creately.com