INF-134 Estructuras de Datos, 2023-1

Tarea 2

Profesores: Gabriel Carmona, Juan Pablo Castillo, Roberto Díaz, Hubert Hoffmann
gabriel.carmonat@sansano.usm.cl,
juan.castillog@sansano.usm.cl,
roberto.diazu@usm.cl
hoffmann@inf.utfsm.cl

Ayudantes San Joaquín:
Diego Debarca
diego.debarca@usm.cl
Gabriel Escalona
gabriel.escalona@usm.cl
Joaquín Gatica
joaquin.gatica@sansano.usm.cl
Juan Cucurella
juan.cucurella@usm.cl
Rodrigo Flores
rodrigo.floresf@sansano.usm.cl

Ayudantes Casa Central:
Catalina Cortez
catalina.cortez@usm.cl
Benjamin Cayo
benjamin.cayo@usm.cl
Sebastián Torrealba
sebastian.torrealba@usm.cl
Tomás Barros
tomas.barros@sansano.usm.cl

Fecha de entrega: 31 de mayo, 2023.
Plazo máximo de entrega: 5 días.

1. Reglas del Juego

La presente tarea debe hacerse en grupos de 3 personas. Toda excepción a esta regla debe ser conversada con los ayudantes (usando los correos indicados en el encabezado de esta tarea). No se permiten de ninguna manera grupos de más de 3 personas. Debe usarse el lenguaje de programación C++. Al evaluarlas, las tareas serán compiladas usando el compilador `g++`, usando la línea de comando `g++ archivo.cpp -o output -Wall`. **No se aceptarán variantes o implementaciones particulares de g++, como el usado por MinGW**. Se deben seguir los tutoriales que están en Aula USM, cualquier alternativa explicada allí es válida. Recordar que una única tarea en el semestre puede tener nota menor a 30. El no cumplimiento de esta regla implica reprobar el curso. No se permite usar la biblioteca *stl*, así como ninguno de los *containers* y algoritmos definidos en ella (e.g. `vector`, `list`, etc.). Está permitido usar otras funciones de utilidad definidas en bibliotecas estándar, como por ejemplo `math.h`, `string`, `fstream`, `iostream`, etc.

2. Objetivos

Entender y familiarizarse con la implementación y utilización de estructuras de datos tipo listas y árboles.

Además se fomentará el uso de las buenas prácticas y el orden en la programación de los problemas correspondientes.

3. Problemas a Resolver

En esta sección se describen los problemas a resolver implementando programas, funciones o clases en C++. Se debe entregar el código para cada uno de los problemas en archivos `.cpp` separados (y correspondientes `.hpp` de ser necesario).

3.1. Problema 1: Secuencia de Nucleótidos

Una secuencia de nucleótidos es una cadena de letras que representan la estructura primaria de una molécula de ADN. Las letras presentes en estas secuencias pueden ser: A, C, G y T, que simbolizan los cuatro tipos de nucleótidos adenina, citosina, guanina y timina respectivamente. Estas secuencias se representan concatenando los nucleótidos sin espacios, por ejemplo, AAAGTTCTGAC.

Generalmente las secuencias de nucleótidos tienen patrones repetitivos y similares. Por lo tanto, basta con almacenar una secuencia base y las modificaciones que se necesitan efectuar a ésta para obtener la secuencia original, realizando:

- **Inserciones**: dada una posición i y un nucleótido n, inserta en la posición i (de izquierda a derecha) el nucleótido n.
- **Borrados**: dada una posición i, elimina el nucleótido en la posición i de la secuencia base.
- **Intercambios**: dada una posición i y un nucleótido n, intercambia el nucleótido de la posición i de la secuencia base por n.

Un grupo de biólogos le han pedido a usted reconstruir secuencias de nucleótidos, utilizando una secuencia base y una serie de cambios, empleando **listas enlazadas**. Su programa también debe hacer uso eficiente de la memoria.

3.1.1. Entrada

La entrada se hará a través del archivo ASCII llamado `secuencias-nucleotidos.txt`. El archivo contiene un entero l que indica el largo de la secuencia base, luego seguido de un espacio le sigue la secuencia base de nucleótidos (de largo l). Luego, sigue un entero k y un salto de línea, que indica la cantidad de secuencias a reconstruir. Finalmente, por cada una de las k secuencias hay un entero m que indica la cantidad de modificaciones (y un salto de línea), luego le siguen las m modificaciones (separadas por un salto de línea cada una) a realizar a la secuencia base. Para identificar cada tipo de modificación, se tienen 3 parámetros (separados cada uno por un espacio en blanco):

1. Tipo de operación, las cuales pueden ser: `INSERTAR`, `BORRAR` o `INTERCAMBIAR`; que corresponden a inserciones, borrados e intercambios respectivamente tal como se especificó anteriormente.
2. Posición i, que identifica la posición dentro de la secuencia donde se quiere realizar la operación del ítem anterior.
3. Nucleótido n que está involucrado en la operación del ítem 1. Si la operación es `BORRAR`, este nucleótido no se encuentra especificado (es decir, solo se especifica hasta el ítem 2, tipo y posición).

Finalmente, las modificaciones se encontrarán ordenadas de menor a mayor con respecto a la posición i de las operaciones. Un ejemplo de la entrada es el siguiente:

```
11 AAAGTTCTGAC
2
3
INSERTAR 0 C
INTERCAMBIAR 2 G
```

```
BORRAR 3
2
BORRAR 3
INTERCAMBIAR 9 A
```

3.1.2. Salida

La salida de datos se hará a través del archivo `secuencias-reconstruidas.txt`. En el archivo se deben escribir las k secuencias reconstruidas, cada una seguida de un salto de línea.

La salida correspondiente al ejemplo planteado anteriormente es el siguiente:

```
CAGGTTCTGAC
AAATTCTGAA
```

3.2. Problema 2: Luces por arreglar

En una avenida principal, se tienen varios puntos en donde hay situados postes de luz. Estos postes de luz están situados en puntos $\texttt{i} \in \mathbb{N}_0$, para $0 \leq \texttt{i} < n$, donde n es la cantidad de postes de la avenida. Debido a la desorganización de la municipalidad donde se encuentra ubicada la avenida, un accidente ha provocado que todos los postes de luz estén apagados. Por temas administrativos y de presupuesto estos se encenderán de manera arbitraria. La municipalidad los ha contactado para resolver el problema identificando eficientemente las secciones apagadas. Para esto necesitan un programa que implemente las siguientes funciones:

- `ENCENDER i`: se debe encender el poste de la posición `i`, si el poste ya está encendido no ocurre nada.
- `CUANTOS_ENCENDER i`: se debe entregar cuántos postes están apagados entre la posición `i` y la posición del poste encendido más cercano en una posición menor o igual a `i`. Si no existe tal poste, debe responder la cantidad de postes apagados entre las posiciones `0` e `i`. Muestra por pantalla el resultado seguido de un salto de línea.
- `PARAR_PROGRAMA`: muestra por pantalla el total de postes encendidos seguido de un salto de línea y termina el programa

Para resolver esto debe implementar un Árbol Binario de Búsqueda (ABB) que soporte al menos las siguientes operaciones:

- `void insert(tElem x)`: inserta el elemento `x` en el árbol
- `bool find(tElem x)`: busca el elemento `x` en el árbol y retorna un valor verdadero de encontrarse en el árbol.
- `int lower_bound(tElem x)`: busca el primer valor que sea menor o igual a `x`.

3.2.1. Entrada

La entrada se hará a través de la entrada estándar. El principio de cada línea contendrá un string que podrá ser `ENCENDER`, `CUANTOS_ENCENDER` o `PARAR_PROGRAMA`. Si el string corresponde a `ENCENDER` o `CUANTOS_ENCENDER` le acompañará un entero i separado por un espacio.

Un ejemplo de la entrada es el siguiente:

```
CUANTOS_ENCENDER 9
ENCENDER 1
CUANTOS_ENCENDER 9
ENCENDER 8
CUANTOS_ENCENDER 9
```

```
ENCENDER 4
ENCENDER 10
CUANTOS_ENCENDER 9
ENCENDER 2
ENCENDER 4
CUANTOS_ENCENDER 2
CUANTOS_ENCENDER 4
PARAR_PROGRAMA
```

3.2.2. Salida

La salida debe realizarse también por la salida estándar. La salida correspondiente al ejemplo planteado anteriormente es el siguiente:

```
10
8
1
1
0
0
5
```

4. Entrega de la Tarea

Entregue la tarea enviando un archivo comprimido `tarea1-apellido1-apellido2-apellido3.zip` o `tarea1-apellido1-apellido2-apellido3.tar.gz` (reemplazando sus apellidos según corresponda) a la página aula.usm del curso, a más tardar el día 31 de mayo, 2023, a las 23:59:00 hs (Chile Continental), el cual contenga como mínimo:

- Los archivos con los códigos fuentes necesarios para el funcionamiento de la tarea. ¡Los archivos deben compilar!
- **nombres.txt**, Nombre, ROL, Paralelo y qué programó cada integrante del grupo.
- **README.txt**, Instrucciones de compilación en caso de ser necesarias, y la forma de compilación que usó para cada problema de la tarea (debe ser alguna de las indicadas en los tutoriales entregados en Aula USM).
- Los nombres de los archivos con el main para ejecutar cada problema deben ser: `problema1.cpp` y `problema2.cpp`

5. Restricciones y Consideraciones

- Por cada día de atraso en la entrega de la tarea se descontarán 10 puntos en la nota.
- El plazo máximo de entrega es 5 días después de la fecha original de entrega.
- **Las tareas que no compilen no serán revisadas y serán calificadas con nota 0**.
- Debe usar **obligatoriamente** alguna de las formas de compilación indicadas en los tutoriales entregados en Aula USM.
- Por cada *Warning* en la compilación se descontarán 5 puntos.

- Si se detecta **COPIA** la nota automáticamente sera 0 (CERO), para todos los grupos involucrados. El incidente será reportado al jefe de carrera.
- La prolijidad, orden y legibilidad del código fuente es obligatoria. Habrá descuentos si alguno de estos items no se cumple.

6. Consejos de Programación

El código fuente del programa debe estar estructurado adecuadamente en archivos (separados de ser necesario). Si el código fuente está desordenado, se pueden descontar hasta 20 puntos de la nota.

Cada función programada debe tener comentarios de la siguiente forma:

```
/*****
*   TipoFunción NombreFunción
******
*   Resumen Función
******
*   Input:
*       tipoParámetro NombreParámetro : Descripción Parámetro
*        .......
******
*   Returns:
*       TipoRetorno, Descripción retorno
*****/
```

Por cada comentario faltante, se restarán 5 puntos.

Por último, la indentación (1 TAB o 4 espacios), es muy importante. Por **cada bloque mal indentado, se quitarán 10 puntos**.