Legajo:……. Nombre y Apellido: ……

Primer Parcial de Estructura de Datos y Algoritmos

Ejer 1	Ejer 2	Ejer 3	Nota
/4	/4	/2	/10

Duración: 2 horas

Condición Mínima de Aprobación. Deben cumplir estas 2 condiciones:

- Sumar **no menos de cuatro** puntos
- Tener por lo menos 2 ejercicios con 2 o más puntos cada uno.

Muy Importante

Al terminar el examen deberían subir los siguientes 2 grupos de archivos, según lo explicado en los ejercicios:

1) Clases Java: URLfy.java, ProximityIndex.java y ProximityIndexTests.java según lo pedido. Si hay código auxiliar, entregarlo también.

2) Para todos los ejercicios que no consistan en implementar código Java y pidan calcular complejidades, dibujar matrices, completar cuadros, hacer seguimientos, etc. pueden optar por alguna de estas estrategias:
   a. O completar este documento y subirlo también
   b. O directamente resolverlo en hojas de papel y sacarle fotos (formato jpg, png o pdf) y subir todas las imágenes.

Se considera una entrega aquella que corresponda a “un upload a Campus” para esta actividad, donde pondrán todos los archivos que deseen entregar.

Si bien pueden subir múltiples veces, sólo se corrige la última entrega realizada dentro del horario que dura la actividad.

Legajo:……. Nombre y Apellido: ……

Ejercicio 1

Queremos construir el método *void reemplazarEspacios(char[] str)* que recibe un arreglo de caracteres y modifica a dicho arreglo de la siguiente manera:
reemplazando cada caracter de espacio ' ' por los siguientes 3 caracteres consecutivos:
'%' '2' '0'

El arreglo recibido posee un contenido útil formado por caracteres (posiblemente algunos blancos, que son los que se quiere reemplazar) y luego, para indicar que finalizó su contenido útil, aparece una secuencia de caracteres '\0' hasta terminar la longitud del arreglo lo cual corresponde al espacio extra necesario para que la transformación pueda realizarse. Se garantiza que el caracter '\0' no aparece en el contenido útil ,sino sólo cuando este ha terminado.

El arreglo recibido tiene exactamente el tamaño necesario para que la transformación sea posible (ni más ni menos). O sea, luego de la transformación no hay más caracteres '\0' porque todo se vuelve contenido útil.

Ejemplo:
Si se invocara
urlfy.reemplazarEspacios(arregloParam);

donde arregloParam es el siguiente arreglo de caracteres (cada celda contiene un carácter)

'T'	'h'	'e'	' '	' '	'j'	'o'	'b'	' '	'2'	'0'	'2'	'2'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

Como se observa, la zona marcada en celeste marca el final del contenido útil y son los 6 caracteres necesarios para que luego de la transformación, el arreglo quede sólo con contenido útil y cada espacio sea reemplazado por los 3 caracteres consecutivos explicados.

Luego de la invocación del método, el arreglo recibido queda así:

'T'	'h'	'e'	'%'	'2'	'0'	'%'	'2'	'0'	'j'	'o'	'b'	'%'	'2'	'0'	'2'	'0'	'2'	'2'

Dado entonces, el siguiente código

Legajo:……. Nombre y Apellido: ……

```
public class URLfy {

 public static void main(String[] args) {

   URLfy urlfy = new URLfy();

   char [] arreglo = new char[] { 'e', 's', ' ', 'u', 'n', ' ', 'e', 'j', 'e', 'm', 'p', 'l', 'o', '\0', '\0', '\0', '\0'};

   urlfy.reemplazarEspacios(arreglo);

   System.out.println(arreglo);


     arreglo= new char [] {'a', ' ', 'b', ' ', 'c', ' ', 'd', ' ', 'e', ' ', 'f', ' ', 'g', ' ', 'h', 'o', 'l', 'a', '\0', '\0',
'\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0'};

  urlfy.reemplazarEspacios(arreglo);

  System.out.println(arreglo);


    arreglo= new char [] {' ', ' ', 'e', 's', 'p', 'a', 'c', 'i', 'o', 's', ' ', ' ', '\0', '\0', '\0', '\0', '\0', '\0',
'\0', '\0'};

  urlfy.reemplazarEspacios(arreglo);

  System.out.println(arreglo);
}

  public void reemplazarEspacios(char [] arregloParam) {
    // programar esto
  }
}
```

Se pide:

1) Implementar el método *public void reemplazarEspacios(char [] arregloParam)* de la clase URLfy con una **complejidad espacial O(1)** y una **complejidad temporal O(n)**, donde n es la longitud del argumento recibido.

2) Justificar claramente (con fórmulas, etc.) que el cálculo de la complejidad espacial y temporal del algoritmo implementado es el solicitado.

Nota: No hay que validar que el arreglo tenga el tamaño necesario para la transformación, eso está asegurado.

Legajo:……. Nombre y Apellido: ……

Ejercicio 2

Tenemos un **índice ordenado que no acepta repetidos** y almacena datos de tipo *String*.

El índice se utilizará para encontrar un elemento y devolver el que se encuentra a cierta distancia desde él.

Por ejemplo:

- Buscar el elemento siguiente (distancia 1) al elemento "Juan"
- Buscar el elemento que está a dos lugares antes del elemento "Ana" (distancia -2)

Para realizar esto, queremos implementar el método
String search(String element, int distance).

Donde:

- El parámetro ***element*** es el elemento a partir del cual se quiere encontrar otro a cierta distancia.
- El parámetro ***distance*** puede ser negativo, positivo o cero, e indica la distancia del elemento requerido medido desde ***element***

Si ***element*** está presente en el índice, el valor devuelto debe ser el String que se encuentra a la distancia solicitada, realizando un tratamiento circular de los elementos del índice.

Si ***element*** no está presente en el índice el método debe devolver null.

Importante:

- Si el parámetro ***distance*** es tal que el elemento quedaría "out of bounds" del arreglo que usamos para el índice, debemos buscar circularmente desde el otro extremo del array hasta encontrarlo.
- La complejidad temporal debe ser O(log N), donde N es la cantidad de elementos del índice. No debe depender la complejidad temporal del parámetro *distance*. Más aún, distance podría ser un número muchíííííísimo mayor que N y eso **no afecta la performance**.

Ejemplos:
Teniendo el índice ordenado formado por 4 elementos
"Ana" - "Carlos" - "Juan" - "Yolanda"

- search("Carlos", 2) debe devolver "Yolanda"

- search("Carlos", 0) debe devolver "Carlos"
- search("Carlos", 3) debe devolver "Ana"
- search("Ana", 14) debe devolver "Juan"
- search("Ana", -2) → "Juan"
- search("Ana", -17) → "Yolanda"
- search("Juan", -4) → "Juan"
- search("XXX", -4) → null

Se pide:

- Descargar de campus el archivo **ProximityIndex.java**.
- Implementar dentro de la clase *ProximityIndex* el método *String search(String element, int distance)* que respete todo lo explicado previamente.
- Realizar los testeos de unidad correspondientes en una nueva clase **ProximityIndexTest**

Ejercicio 3

Se tiene la siguiente expresión aritmética infija y se la quiere **escribir en notación postfija utilizando el algoritmo visto en clase**.

Se deberá mostrar todo el procedimiento intermedio (no solo el resultado final), es decir, en cada paso visualizar:

- el estado del **stack**
- **cuál es el token del input que se está analizando**
- **cómo queda el string de output.**

Luego, el resultado final de la expresión.

La expresión en notación infija es:

5		^		(		1		-		1		)		^		(		3		/		1		)		*		5

Para evitar problemas de interpretación, se indica en cada casillero los tokens y los blancos. Como se observa, está garantizado el separador blanco entre tokens de operadores/paréntesis/operandos.

Mostrar gráficamente el Seguimiento (mostrar pasos intermedios como se hizo en clase: pila , input y salida) según lo pedido.

Legajo:…….. Nombre y Apellido: ……..

Completar seguimiento