Homework 2015

Juana Paulina Águila Hernández

Características del equipo en donde se hicieron las pruebas:

- **Sistema operativo:** OS X Yosemite (10.10.4)
- **Procesador:** 2.5 GHz Intel Core i5
- **Memoria:** 4 GB 1600 MHz DDR3
- **Versión de Java:** 1.6.0_65-b14-466.1-11M4716
- **Memoria disponible:** 81 MB (Información proporcionada por la clase *Runtime*).
- **Memoria máxima disponible:** 123 MB (Información proporcionada por la clase *Runtime*).

Notas:

- El proyecto de eclipse se llama Homework2015
- Archivo donde se ejecutaron las pruebas: package/Principal/TestZone.java, en el cual, al ejecutarse el proyecto de eclipse el usuario tiene permitido escoger la prueba a ejecutar desde consola, pero no puede modificar los valores utilizados para ello. Por eso, se indican los valores a modificar desde su declaración en este archivo:
  - noEjecuciones: indica en número de ejecuciones que se harán en cada prueba
  - longMuestra: Indica la longitud de la cadena en el ejercicio 1.
- Solo se hace una implementación por ejecución.
- Repositorio en github: https://github.com/vegenisennawa/Homework2015.git
- El tiempo para las pruebas está en milisegundos.

Ejercicios:

1. Implementar la siguiente recursión:
   a. Dado un elemento de la clase *String* de Java, se necesita codificar una función recursiva que cuente todos los caracteres en un objeto *String*.
   Este ejercicio se puede ver dentro de la clase RecSring [package/MyString/RecString.java]

   Se tomó en cuenta un índice que se ingresa a la función *StrLen* y la cadena a medir.
   Si el iterador ingresado es menor a la longitud de la cadena, entonces va llamando a la función recursivamente hasta que no se cumpla esa condición, regresando el nuevo iterador.
   Si el iterador es mayor a la longitud de la cadena, la función regresa -1 para indicar un error.

b. Dé la recursión calculando T(n).

Cálculo para cuando la cadena tiene longitud 0.

Código	**Pasos**
StrLen(index, cadena)	
if(index < cadena.length())	1
return *StrLen*(++index, cadena);	0
else if(index > cadena.length())	0
index = -1;	0
return index;	1

Sumando: 1 + 1 = 2.

Cálculo para cuando la cadena tiene longitud 1.

Código	**Pasos**
StrLen(index, cadena)	
if(index < cadena.length())	1
return *StrLen*(++index, cadena);	1
else if(index > cadena.length())	0
index = -1;	0
return index;	1

Sumando: 1 + 1 + 1 = 3.

Cálculo para cuando la cadena tiene longitud mayor a la longitud de la cadena

Código	**Pasos**
StrLen(index, cadena)	
if(index < cadena.length())	1
return *StrLen*(++index, cadena);	0
else if(index > cadena.length())	1
index = -1;	1
return index;	1

Sumando: 1 + 1 + 1 +1 = 4

Cálculo para cuando la cadena tiene longitud = 0 y longitud = n.

Código	**Pasos**
StrLen(index, cadena)	
if(index < cadena.length())	1
return *StrLen*(++index, cadena);	n-1
else if(index > cadena.length())	0
index = -1;	0
return index;	1

Sumando: 1 + n - 1 +1 = 1 + n

c. Pruebe con una muestra suficientemente grande.

Se tomó como referencia a la cadena a, aumentando su longitud dependiendo de lo que se requería.
Para el tiempo, se tomó el promedio de 10 ejecuciones con cada longitud.

Longitud de la prueba	**Tiempo**
1×10^0	0.1
1×10^1	0.0
1×10^2	0.1
1×10^3	0.2
1×10^4	java.lang.StackOverflowError

2. Implemente la representación de la cadena y usando ideas de la clase, necesitas probar el tiempo promedio para cada método, add y remove, sobre la representación de la cadena usando un tiempo adecuado y un número random adecuado de integers.

   El tiempo escrito en la tabla representa el promedio de una ejecución de la definición de la lista con cada tamaño n de la muestra.

n	**Tiempo**
1×10^0	1.0
1×10^1	13.0
1×10^2	1.0
1×10^3	1.0
1×10^4	1.0
1×10^5	0.0
1×10^6	0.0

El tiempo escrito en la tabla representa el promedio de n inserciones en la lista.

n	Tiempo
1×10^0	0.0
1×10^1	1.7
1×10^2	0.01
1×10^3	0.029
1×10^4	0.0136
1×10^5	0.11056
1×10^6	Se detuvo la prueba

El tiempo escrito en la tabla representa el promedio de un rango de intentos de eliminación al hacer eliminaciones en la lista.
Cuando n va de 1 a 10, se toma ese número de intentos de eliminación.
Si n es mayor a 10, entonces se toma el numero resultante de dividir n entre 10.

n elementos en la lista	Tiempo
1×10^0	0.0
1×10^1	0.0
1×10^2	0.0
1×10^3	2.0
1×10^4	27.0
1×10^5	2447.0
1×10^6	Se detuvo la prueba

3. Imagina que tienes una representación de array de arrays de una matriz triangular inferior, por favor desarrolle una clase para este tipo de matrices con los siguientes elementos y métodos [ver los elementos y métodos en la clase].

   El tiempo escrito en la tabla representa el promedio de una ejecución de la definición de la matriz con cada tamaño n de la muestra.

n	Tiempo
1×10^0	0.0
1×10^1	2.0
1×10^2	1.0
1×10^3	3.0
1×10^4	java.lang.OutOfMemoryError: Java heap space

El tiempo escrito en la tabla representa el promedio de n ejecuciones al colocar datos de la matriz con cada tamaño n de la muestra.

n	Tiempo
1×10^0	0.0
1×10^1	0.0
1×10^2	0.0
1×10^3	0.0
1×10^4	java.lang.OutOfMemoryError: Java heap space

El tiempo escrito en la tabla representa el promedio de n ejecuciones al obtener datos de la matriz con cada tamaño n de la muestra.

n	Tiempo
1×10^0	0.0
1×10^1	0.0
1×10^2	0.0
1×10^3	0.0010
1×10^4	java.lang.OutOfMemoryError: Java heap space

El tiempo escrito en la tabla representa el promedio de n ejecuciones al eliminar datos de la matriz con cada tamaño n de la muestra.

n	Tiempo
1×10^0	0.0
1×10^1	0.0
1×10^2	0.0
1×10^3	0.0
1×10^4	java.lang.OutOfMemoryError: Java heap space

El tiempo escrito en la tabla representa el mostrar datos de la matriz con cada tamaño n de la muestra.

n	Tiempo
1×10^0	0.0
1×10^1	0.0
1×10^2	227.0
1×10^3	1759.0
1×10^4	java.lang.OutOfMemoryError: Java heap space

4. Dada la interface de la pila, por favor crea una implementación para esta.
   Los archivos se encuentran en el proyecto (package/MyStack/MyScratchStack.java y package/MyStack/Stack.java).