Homework 2015

Juana Paulina Águila Hernández

*Características del equipo en que 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*).

*Ejercicios:*

1. Implementar la siguiente recursión:
   1. Dado un elemento de la clase *String* de Java, se necesita codificar una función recursiva que cuente todos los caracteres en un objeto *String*.

Para este ejercicio 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.

* 1. Dé la recursión calculando T(n).

Cálculo para cuando la cadena tiene longitud 0.

Código Cálculo

StrLen(int index, String 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 Cálculo

StrLen(int index, String 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 Cálculo

StrLen(int index, String 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 Cálculo

StrLen(int index, String 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

* 1. Pruebe con una muestra suficientemente grande.

El tiempo escrito en la tabla representa el promedio de 10 ejecuciones con cada longitud de la muestra.

La muestra es la cadena 1.

|  |  |
| --- | --- |
| **Longitud de la prueba** | **Tiempo** |
| 1 x 100 | 4.4 |
| 1 x 101 | 0.6 |
| 1 x 102 | 4.9 |
| 1 x 103 | 3.5 |
| 5 x 103 | 1.6 |
| 9 x 103 | 4.8 |
| 9922 | 13.7 |
| 1 x 104 | java.lang.StackOverflowError |

1. 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 x 100 | 1 |
| 1 x 101 | 1 |
| 1 x 102 | 1 |
| 1 x 103 | 1 |
| 1 x 104 | 19 |

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

|  |  |
| --- | --- |
| **n** | **Tiempo** |
| 1 x100 | 26 |
| 1 x 101 | 27 |
| 1 x 102 | 41 |
| 1 x 103 | 19 1 1 1 1 |
| 1 x 104 |  |

El tiempo escrito en la tabla representa el promedio de n ejecuciones al hacer eliminaciones en la lista.

|  |  |
| --- | --- |
| n | Tiempo |
| 1 x100 | 0 |
| 1 x 101 | 0 |
| 1 x 102 | 0 |
| 1 x 103 | 1 1 |
| 1 x 104 |  |

1. 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 hoja de la tarea].

Según dado en las sesiones, un array de arrays es una estructura de datos que representa columnas y filas de tamaño uniforme en la que sus datos se pueden obtener por medio de una coordenada (en java puede ser matriz[*x*][*y*], donde *x* y *y* son valores de tipo int y están dentro del rango de tamaño de la matriz).

La matriz triangular inferior es una matriz de nxn en donde los valores por encima de su diagonal principal son cero.

La definición de los elementos y métodos se pueden ver en el siguiente pseudocódigo:

public class Matrix

int[][] Matriz

public Matrix(int n)

Matriz = new int[n][n]

for(i=0; i<n; i++)

for(j=0; j<n; j++)

Matriz[i][j] = 0

public void add(int i, int j, int valor)

if((i>=0 && i<Matriz.length) && (j>=0 && j<Matriz.length))

if(i>=j)

if(valor != -1 && valor != 0)

Matriz[i][j] = valor

public int get(int i, int j)

if((i>=0 && i<Matriz.length) && (j>=0 && j<Matriz.length))

return Matriz[i][j]

else

return -1

public int remove(int i, int j)

if((i>=0 && i<Matriz.length) && (j>=0 && j<Matriz.length))

int retorno = get(i, j)

add(i, j, 0)

return retorno

else

return -1

public void output()

for(int i=0; i<Matriz.length; i++)

for(int j=0; j<Matriz.length; j++)

System.out.print(Matriz[i][j] + " ")

System.out.println()

Los cálculos de complejidad se pueden visualizar en este punto:

Inicialización de la matriz para longitud n cuando n >= 0.

Código Cálculo

public Matrix(int n)

Matriz = new int[n][n] 1

for(i=0; i<n; i++) n

for(j=0; j<n; j++) i+1

Matriz[i][j] = 0 i

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 x 100 | 0 |
| 1 x 101 | 1 |
| 1 x 102 | 1 |
| 1 x 103 | 14 |
| 1 x 104 | java.lang.OutOfMemoryError: Java heap space |

El tiempo escrito en la tabla representa el promedio de 10 ejecuciones al colocar datos de la matriz con cada tamaño n de la muestra.

|  |  |
| --- | --- |
| **n** | **Tiempo** |
| 1 x100 | 0 |
| 1 x 101 | 0 |
| 1 x 102 | 1 |
| 1 x 103 | 0 |
| 1 x 104 | java.lang.OutOfMemoryError: Java heap space |

El tiempo escrito en la tabla representa el promedio de 10 ejecuciones al obtener datos de la matriz con cada tamaño n de la muestra.

|  |  |
| --- | --- |
| n | Tiempo |
| 1 x100 | 0 |
| 1 x 101 | 0 |
| 1 x 102 | 1= |
| 1 x 103 | 1= |
| 1 x 104 | java.lang.OutOfMemoryError: Java heap space |

El tiempo escrito en la tabla representa el promedio de 10 ejecuciones al eliminar datos de la matriz con cada tamaño n de la muestra.

|  |  |
| --- | --- |
| n | Tiempo |
| 1 x100 | 0 |
| 1 x 101 | 1 = |
| 1 x 102 | 1, = |
| 1 x 103 | 128, 1,1= |
| 1 x 104 | java.lang.OutOfMemoryError: Java heap space |

El tiempo escrito en la tabla representa el promedio de 10 ejecuciones al mostrar datos de la matriz con cada tamaño n de la muestra.

|  |  |
| --- | --- |
| n | Tiempo |
| 1 x100 | 0 |
| 1 x 101 | 2 |
| 1 x 102 | 133 |
| 1 x 103 | 1829 |
| 1 x 104 | java.lang.OutOfMemoryError: Java heap space |

1. Dada la interface de la pila, por favor crea una implementación para esta.
2. Implemente el diccionario ADT usando hashing y separando cadenas. Use una cadena con nodos vinculados como cada cubo.