UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE INGENIERÍA Y CIENCIAS
CC3001 - 1 Algoritmos y Estructuras de Datos

TAREA 2
Pila de Arena Optimizada

Daniela Campos
danielacamposfischer@gmail.com

Profesor:
Patricio Poblete

Fecha:
8 de Mayo del 2017

Índice

1 Introducción **2**

2 Análisis del Problema **3**
2.1 Problema . . . 3
2.2 Suposiciones sobre el Problema . . . 4
2.3 Casos Bordes . . . 4

3 Solución del Problema **5**
3.1 Clases y Métodos Utilizados . . . 5
3.1.1 Clase Ventana . . . 5
3.1.2 Método mostrarMatriz . . . 6
3.1.3 Clase PilaArenaOptimizada . . . 6
3.1.4 Método estabilizar . . . 6
3.1.5 Método Main . . . 6
3.1.6 Clase Pila . . . 6
3.1.7 Constructor Pila . . . 7
3.1.8 Método estaVacia . . . 7
3.1.9 Método apilar . . . 7
3.1.10 Método desapilar . . . 7
3.2 Ejemplos Entradas y Salidas . . . 8
3.3 Tiempos de Ejecución . . . 13

4 Modo de Uso **14**

5 Discusión **15**

6 Anexos **16**

1 Introducción

En el presente informe, al igual que en la Tarea 1 de Algoritmos y Estructuras de Datos, se estudiará el comportamiento de granos de arena en el espacio, al serle aplicada una regla de estabilidad debido a la gravedad. Se busca visualizar el comportamiento que tendrá una dada cantidad de granos de arena al ser depositados en el centro del "espacio", esto se logrará gracias a la modelación del fenómeno.

Al igual que en la Tarea 1, el problema fue modelado a través de una matriz, la cual supone ser un mallado del espacio. Los granos de arena son depositados en el centro de la matriz, pero a diferencia de la Tarea 1, el proceso de estabilización no se hace utilizando iteración, sino que implementando un dato abstracto(TDA) llamado Pila, en el cual se pueden insertar o eliminar en tiempo constante las casillas que tengan desborde y las que han sido corregidas respectivamente.

Las Pilas se rigen por el principio de **Last in First Out**, es decir que el último elemento que fue agregado es el primero en ser eliminado, lo que permite trabajar con la pila sin tener la necesidad de recorrerla, optimizando así el proceso de estabilización de la matriz. Para la implementación de la pila se utilizó una Lista Enlazada.

2 Análisis del Problema

2.1 Problema

El problema propuesto, consiste en que se tiene una superficie potencialmente infinita y en el centro de esta se depositarán granos de arena. La idea es modelar como se comportarían estos granos de arena en la vida real, para esto se debe tener en cuenta el efecto de la gravedad sobre una pila de arena, ya que si hay muchos granos en un mismo punto del espacio, se estará en un estado inestable, para pasar a uno estable, los granos de arena se desbordarán.

El proceso de estabilización se lleva a cabo de la siguiente manera, si existen cuatro o mas granos de arena en una celda, estos se desbordarán hacia las celdas adyacentes, sin ningún orden en particular debido a que el dominio generado por esta regla es un grupo abeliano, es decir el orden el que se aplica la regla no es relevante.

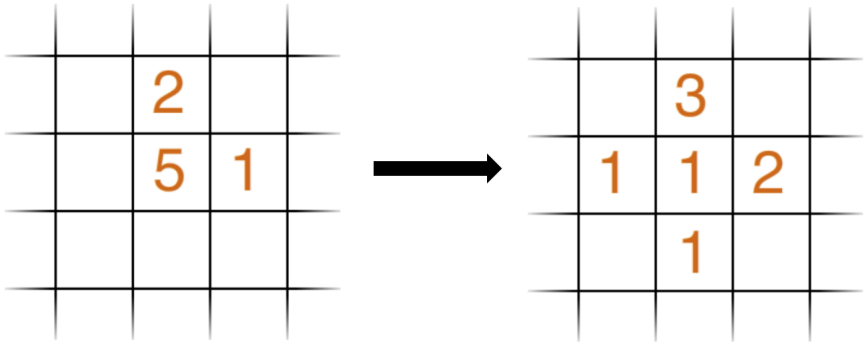

Figura 1: Estabilización de una celda

Como se puede observar en la Figura 2, tras estabilizar la central, no siempre es necesario estabilizar todas las celdas adyacentes y por lo tanto, la solución iterativa no es óptima. Debido a esto, se solicita utilizar un Pila para resolver este problema. La Pila comenzará insertando la celda central y a medida que se vaya realizando el proceso de estabilización, insertará las casillas que deban ser desbordadas.

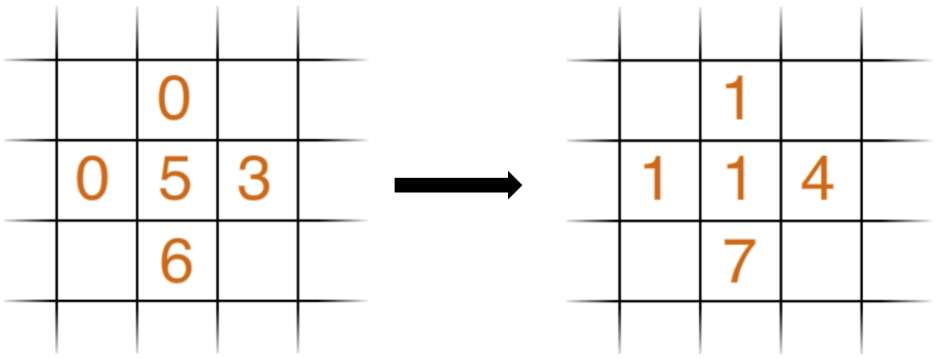

Figura 2: Efectos de la Estabilización de una celda en sus vecinas

2.2 Suposiciones sobre el Problema

Dado a que no es posible hacer una modelación de una superficie infinita, se considerará un espacio lo suficientemente grande para que poder observar la evolución de la Pila de Arena al variar la cantidad de granos de arena.

2.3 Casos Bordes

Como la matriz con la que se trabaja en la modelación es finita, no todas las celdas tienen una celda vecina sobre la cual se puedan desbordar, esto ocurre con bordes de la matriz. Para resolver esto, dado una cantidad de granos de arena N, crearemos un matriz de tamaño $\sqrt{N} \times \sqrt{N}$, de esta manera, el desborde de los granos de arena no llega nunca al borde de la matriz, evitando así que se trate de acceder a índices inexistentes.

3 Solución del Problema

Para realizar la modelación, se toman la cantidad de granos de arena del usuario y posteriormente se crea la Matriz y la Pila, y se apila la celda central, donde se encuentran los granos de arena. Luego mientras la Pila no esté vacía se sigue el proceso descrito en la Figura 3.

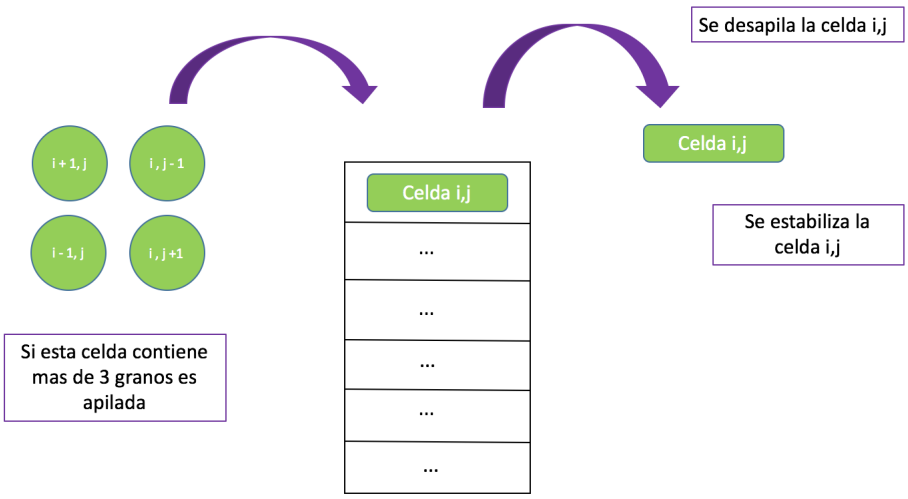

Figura 3: Diagrama Explicativo del funcionamiento del programa

Posteriormente, se crea la ventana y dibuja la matriz en esta.

3.1 Clases y Métodos Utilizados

A continuación se expondrán las Clases y Métodos utilizados para modelar de manera óptima la evolución de una Pila de Arena dado el efecto de al gravedad.

3.1.1 Clase Ventana

Esta clase fue provista por el cuerpo docente. Cumple la función de constructor, crea y muestra una ventana de tamaño y título especificados como parámetros.

3.1.2 Método mostrarMatriz

Este método se encuentra en la Clase Ventana. Toma como único parámetro una matriz y la dibuja como cuadros de colores en la ventana. La matriz entregada al método debe contener solo números entre 0 y 3.

3.1.3 Clase PilaArenaOptimizada

Esta clase contiene el algoritmo que provee la modelación. Posee a los métodos "estabilizar" y "main".

3.1.4 Método estabilizar

Este método toma como argumento, la matriz, los índices de la celda a estabilizar y la Pila de celdas por estabilizar. El método, obtiene la cantidad de granos de arena que deben ser repartidos en las celdas adyacentes (variable *cant*), y le otorga a la celda por estabilizar el resto de la división de su valor inicial en 4. Luego, a cada celda adyacente le suma *cant* y verifica si debe ser agregada a la Pila de las celdas por desbordar.[1]

3.1.5 Método Main

Este método permite la ejecución de la modelación, ya que obtiene del usuario la cantidad de granos de arena que deben ser depositados en el centro de la matriz.

Luego, crea la matriz de tamaño $\sqrt{N} \times \sqrt{N}$ junto con la Pila de celdas por desbordar, la cual inicialmente sólo contiene los índices de la celda central. Posteriormente, estabiliza las celdas que se encuentran en la Pila hasta que esta esté vacía.

Posterior a esto, crea la ventana con la clase provista y muestra la matriz.[2]

3.1.6 Clase Pila

Esta clase contiene al constructor de la Pila, el método "apilar", "desapilar" y "estaVacia".

[1] El código en detalle del método estabilizar puede ser encontrado en el Anexo 1
[2] El código en detalle del método main puede ser encontrado en el Anexo 2

3.1.7 Constructor Pila

La Pila se crea implementando una Lista Enlazada. El constructor crea la **Clase Nodo** junto con el Objeto Pila, el cual tiene dos elementos, el último nodo y el tamaño de la pila.[3]

3.1.8 Método estaVacia

Este método chequea si la Pila está vacía al revisar si el tamaño de esta es 0. Retorna un valor del tipo **boolean**, *True* si está vacía y *False* sino.[4]

3.1.9 Método apilar

Este método toma como argumento los índices de la celda a apilar y los agrega a un nodo, el cual es ingresado a la Pila, además hace crecer en uno el tamaño de la Pila. No retorna nada dado que es un **void**.[5]

3.1.10 Método desapilar

Este método desapila el último elemento de la Pila y retorna un arreglo con los índices de la celda por apilar. Si la Pila está vacía retorna un arreglo con -1. Además hace decrecer la variable tamaño en 1.[6]

[3]El código en detalle del Constructor de la Pila puede ser encontrado en el Anexo 3
[4]El código en detalle del método estaVacia puede ser encontrado en el Anexo 4
[5]El código en detalle del método apilar puede ser encontrado en el Anexo 5
[6]El código en detalle del método desapilar puede ser encontrado en el Anexo 6

3.2 Ejemplos Entradas y Salidas

A continuación se expondrán los resultados obtenidos dadas variadas cantidades de granos de arena.

1. **Entrada: N = 99999**

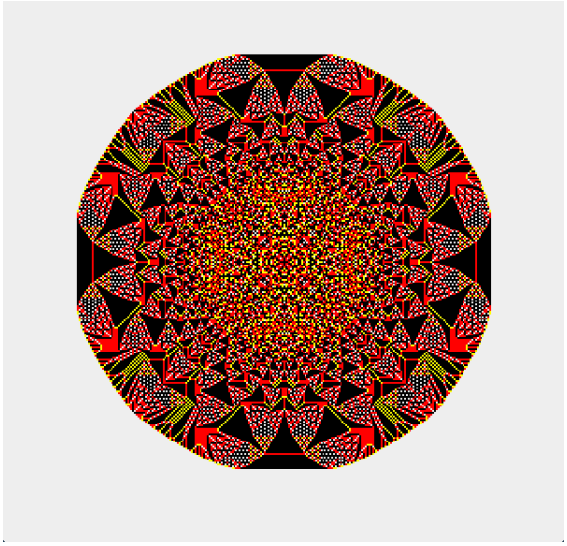

Figura 4: Caso Prueba N = 99999

2. **Entrada: N = 90099**

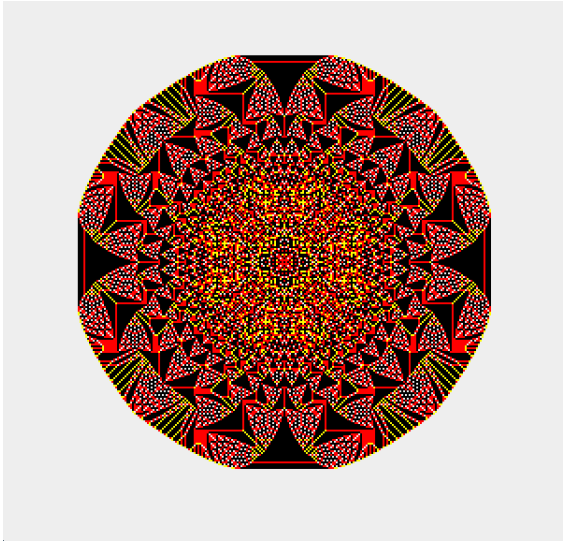

Figura 5: Caso Prueba N = 90099

3. **Entrada: N = 80199**

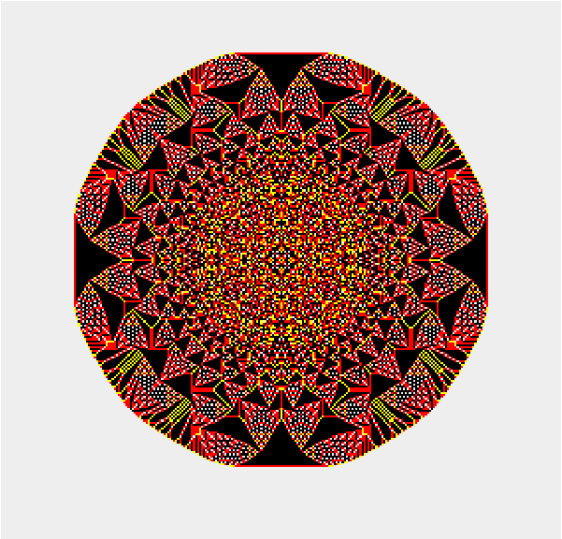

Figura 6: Caso Prueba N = 80199

4. **Entrada: N = 70299**

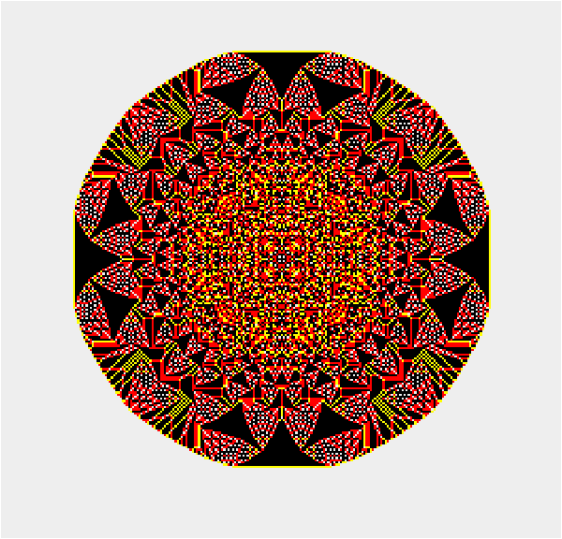

Figura 7: Caso Prueba N = 70299

5. **Entrada: N = 60399**

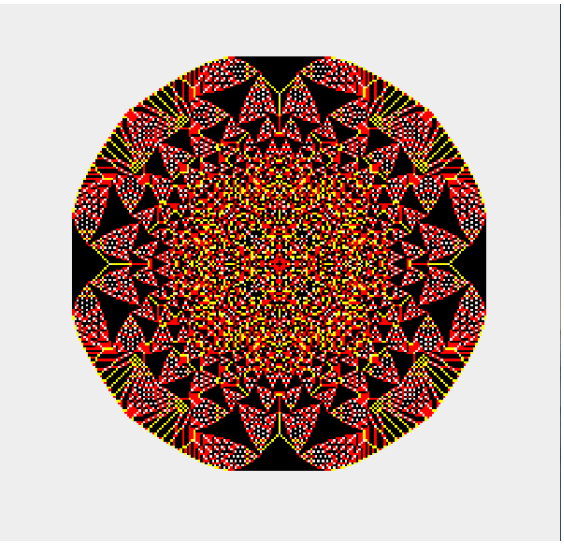

Figura 8: Caso Prueba N = 60399

6. **Entrada: N = 50499**

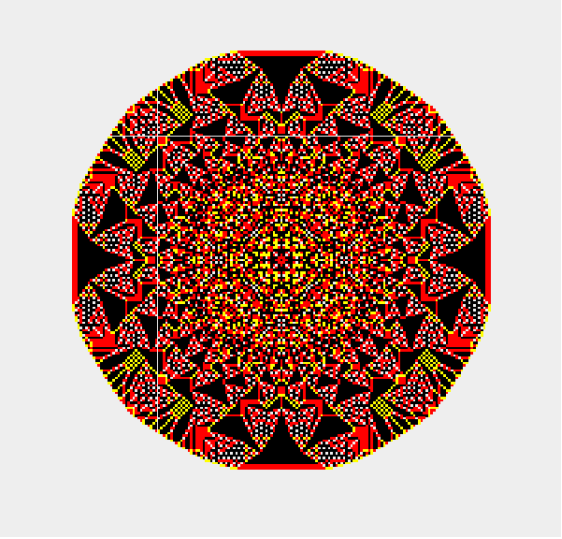

Figura 9: Caso Prueba N = 50499

7. **Entrada: N = 40599**

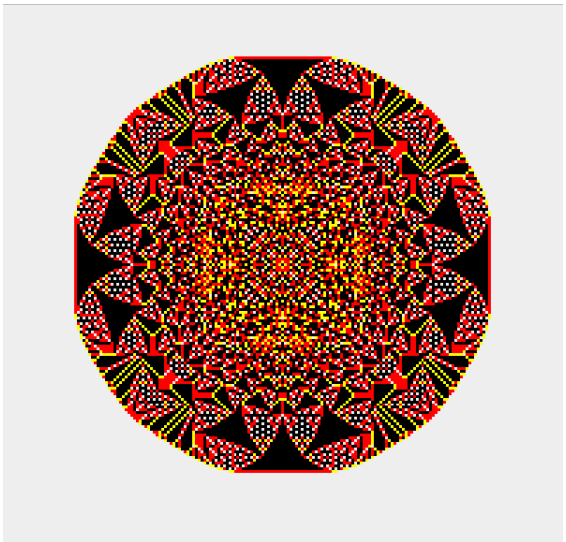

Figura 10: Caso Prueba N = 40599

8. **Entrada: N = 30699**

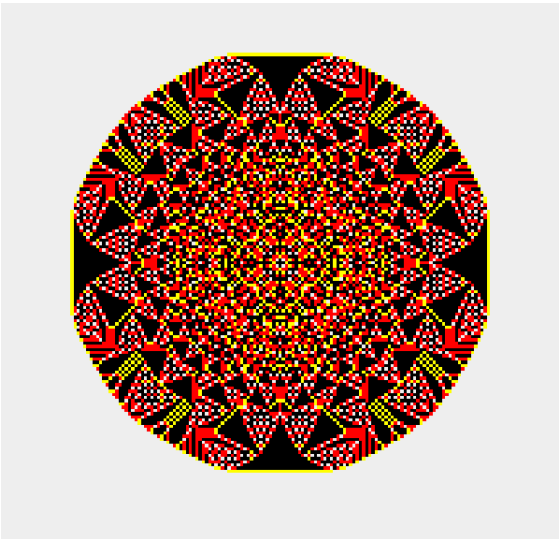

Figura 11: Caso Prueba N = 30699

9. **Entrada: N = 20799**

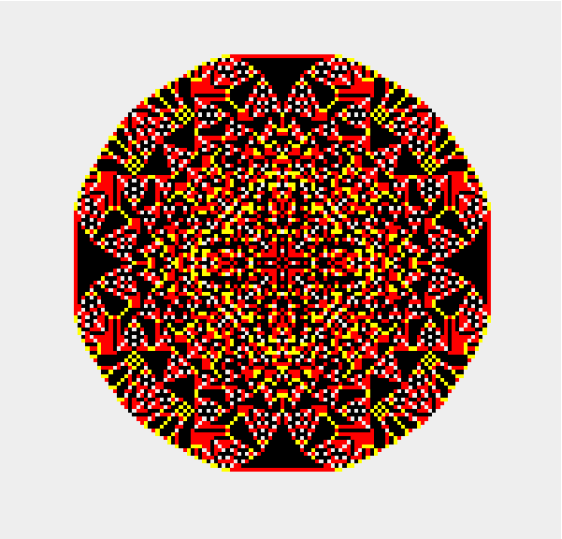

Figura 12: Caso Prueba N = 20799

10. **Entrada: N = 10899**

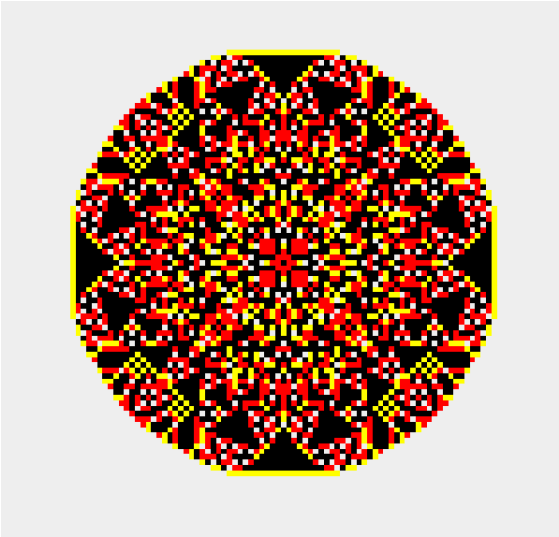

Figura 13: Caso Prueba N = 10899

3.3 Tiempos de Ejecución

En la Tabla 1, se encuentran los tiempos de ejecución de la modelación para la solución de la Tarea 1 y la tarea actual. Se dieron como entrada cantidades de granos de arenas $\in [10^3, 10^5]$, las cuales tenían entre ellas un paso de 9900. Con los datos de la Tabla 1, se hizo un gráfico con las proyecciones de los tiempos de ejecución de cada tarea, el cual se encuentra en la Figura 14.

N	Tiempo Tarea 1[s]	Tiempo Tarea 3 [s]
99999	6.57	15.19
90099	5.39	13.17
80199	4.32	11.55
70299	3.88	9.87
60399	2.90	8.25
50499	2.45	6.83
40599	1.86	5.46
30699	1.50	4.34
20799	1.28	3.06
10899	0.99	1.83

Tabla 1: Tabla comparativa de Tiempos de Ejecución

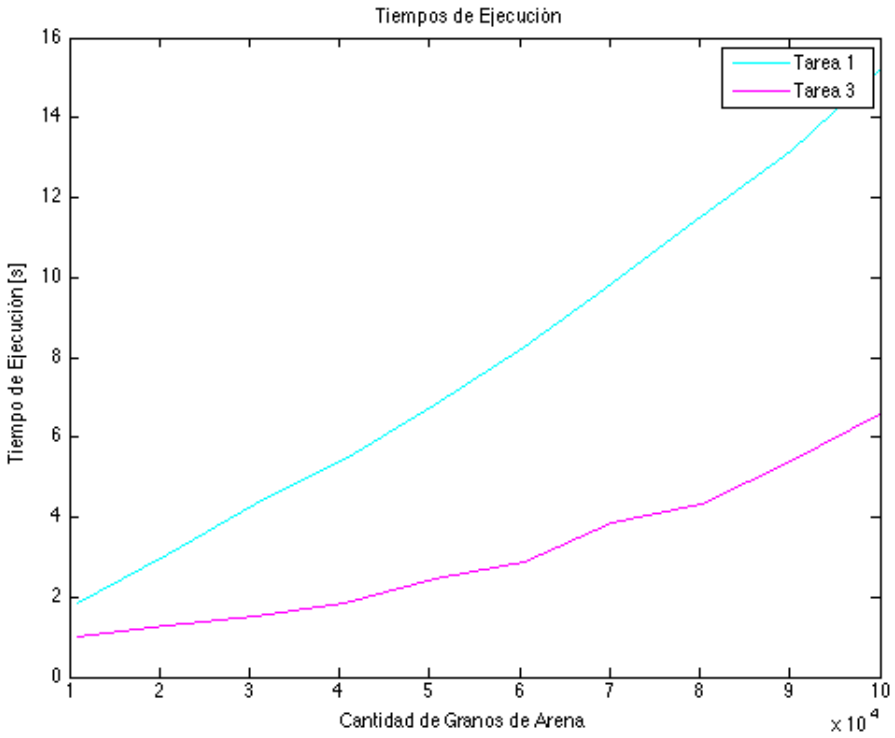

Figura 14: Gráfico Comparativo tiempos de Ejecución

4 Modo de Uso

Para que el programa funcione correctamente, se deben guardar los archivos PilaArenaOptimizada.java, Ventana.java y Pila.java en una misma carpeta. Se deben compilar Ventana.java y Pila.java antes de compilar PilaArenaOptimizada.java, tras compilar este y ejecutarlo en el terminal, se le preguntará al usuario cuantos granos de arena quiere depositar en el centro de la matriz, en esta instancia debe ser ingresado un *Interger*.

5 Discusión

Se puede apreciar que el programa funciona bien y que se obtuvieron los resultados esperados.

Todos los métodos de la **Clase Pila**, toman tiempo constante en ser realizados ya que sólo se realizan asignaciones de valores y estas toman tiempo constante, por lo tanto tienen Orden 1.

La solución obtenida es más rápida que la de la Tarea 1, pero no por un margen muy grande, como se puede apreciar en la Tabla 1. La diferencia en los tiempos de ejecución radica en que en la Tarea 1, se recorría la matriz hasta que no hubiera ninguna celda por desbordar[7]. En cambio, en la solución de la Tarea 3, cada celda se desborda una sola vez y no se recorre la matriz, logrando así que el tiempo de ejecución sea menor. Se puede observar en el grafico de la Figura 14, que a pesar de que para los valores evaluados la diferencia no es considerable, dado el comportamiento que tiene la curva asociada a la solución de la Tarea 1, se puede apreciar que para valor mas grandes de N esta tomara mucho mas tiempo que la solución de la Tarea 3.

Así,podemos concluir que la utilización de Datos Abstractos como las Pilas en este tipo de modelaciones optimiza de buena manera nuestro programa y se vuelve fundamental a a la hora de trabajar con valores muy grandes.

[7]El código de la Tarea 1 se encontrará en el Anexo

6 Anexos

1. **Anexo 1:** Método main Clase PilaArenaOptimizada

```
 public static void estabilizar(int[][] mat, int i, int j, Pila p){
        int a = 0, b = 0, c = 0, d = 0, cant = 0;
        cant = mat[i][j] / 4;
        mat[i][j] = mat[i][j]%4;
        a = mat[i + 1][j];
        mat[i + 1][j] += cant;
        if(a < 4 && mat[i + 1][j] >= 4){
            p.apilar(i + 1, j);
        }
        b = mat[i][j + 1];
        mat[i][j + 1] += cant;
        if(b < 4 && mat[i][j + 1] >= 4){
            p.apilar(i, j + 1);
        }
        c = mat[i - 1][j];
        mat[i - 1][j] += cant;
        if(c < 4 && mat[i - 1][j] >= 4){
            p.apilar(i - 1, j);
        }
        d = mat[i][j - 1];
        mat[i][j - 1] += cant;
        if(d < 4 && mat[i][j - 1] >= 4){
            p.apilar(i, j - 1);
        }
}
```

2. **Anexo 2:** Método estabilizar Clase PilaArenaOptimizada

```
static public void main(String[] args){
        Scanner s = new Scanner(System.in);
        System.out.print("Cuantos granos de arena ... : ");
        int N = s.nextInt();
        double d = Math.floor(Math.sqrt(N));
        int dim = (int) d;
        int[][] mat = new int[dim][dim];
        mat[dim/2][dim/2] = N;
        Pila porDesbordar = new Pila();
        porDesbordar.apilar(dim/2,dim/2);
        int[] indices;

        while(!porDesbordar.estaVacia()){
            indices = porDesbordar.desapilar();
            estabilizar(mat, indices[0], indices[1], porDesbordar);
        }
        Ventana v = new Ventana(600);
        v.mostrarMatriz(mat);

}
```

3. **Anexo 3:** Constructor Clase Pila

```
    private Node ultimo;
    private int tamano;

    private class Node{
        int indicex;
        int indicey;
        Node sgte;
    }

    public Pila(){
        ultimo = null;
        tamano = 0;
    }
```

4. **Anexo 4:** Método estaVacia Clase Pila

```
boolean estaVacia(){
        if(tamano == 0){
            return true;
        }
        return false;
 }
```

5. **Anexo 5:** Método apilar Clase Pila

```
void apilar(int i, int j){
        if(estaVacia()){
            ultimo = new Node();
            ultimo.indicex = i;
            ultimo.indicey = j;
            ultimo.sgte = null;
        }
        else{
            Node anterior = ultimo;
            ultimo = new Node();
            ultimo.indicex = i;
            ultimo.indicey = j;
            ultimo.sgte = anterior;
        }
        ++tamano;
}
```

6. **Anexo 6:** Método desapilar Clase Pila

```
int[] desapilar() {
        int[] pos;
        pos = new int[2];

        if (estaVacia()) {
            System.out.println("Pila Vacia");
            pos[0] = -1;
            pos[1] = -1;
        }
        else {

            pos[0] = ultimo.indicex;
            pos[1] = ultimo.indicey;

            if (tamano == 1) {
                ultimo = null;
            } else {
                ultimo = ultimo.sgte;
            }
            --tamano;
        }

        return pos;
 }
```

7. **Anexo 7:** Clase PilaArena Tarea 1

```
import java.util.*;
public class PilaArena{
    public static void estabilizar(int[][] mat,int i,int j,int a){
        mat[i][j] = mat[i][j] - 4;
        if((i+1)<a-1)mat[i+1][j]++;
        if((j+1)<a-1)mat[i][j+1]++;
        if((i-1)>0)mat[i-1][j]++;
        if((j-1)>0)mat[i][j-1]++;
    }
    static public void main(String[] args){
        Scanner s = new Scanner(System.in);
        System.out.print("Cuantos granos de arena pondra en el centro? : ");
        int N = s.nextInt();
        int a = 301;
        int[][] mat = new int[a][a];
        for (int i=0; i<a; ++i) {
            for (int j=0; j<a; ++j) {
                mat[i][j]=0;
            }
        }
        mat[a/2][a/2] = N;
        int count = 0;
        while(count < a*a){
            count = 0;
            for (int i=0; i<a; ++i) {
                for (int j=0; j<a ; ++j) {
                    if(mat[i][j] >= 4){
                        estabilizar(mat,i,j,a);
                    }
                    else count++;
                }
            }
        }
        Ventana v = new Ventana(600);
        v.mostrarMatriz(mat);
    }
}
```