UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Estructura de Datos y Análisis de Algoritmos

Informe Laboratorio 1

Ariel Ignacio Tirado Maturana

Profesor:
Jacqueline Kohler Casasempere
Alejandro Cisterna Villalobos

Ayudante:
Gerardo Zuñiga Leal

Santiago - Chile
2-2016

TABLA DE CONTENIDOS

ÍNDICE DE IMÁGENES

CAPÍTULO 1. INTRODUCCIÓN

Una de las tareas más comunes en el procesamiento de imágenes es la identificación de sub-imágenes o proporciones de estas dentro de otras imágenes. Para este trabajo se realiza una descomposición de estas imágenes en pixeles, los cuales son determinados por la cantidad de color rojo, verde y azul que compone un pixel (RGB por sus siglas en inglés). La cantidad de cada color queda definida por un número entero, el cual está en el rango entre el número 0 y el 255, lo que finalmente da el color definitivo del pixel. Gracias a esto, es que se pueden lograr coincidencias entre distintas imágenes.

Para el presente laboratorio, se les ha solicitado a los alumnos del curso de Análisis de Algoritmos y Estructuras de Datos el diseño de un algoritmo que realice la comparación entre distintas imágenes, y se busquen coincidencias entre estas, generando un archivo de salida, en el cual se podrá obtener el resultado de dichas comparaciones.

A lo largo del presente informe, se presentará al lector la abstracción de este problema y la solución propuesta.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

2.1 ANÁLISIS DEL PROBLEMA

Para el presente laboratorio, se ha propuesto realizar un algoritmo de búsqueda de coincidencias entre diversas imágenes. Para desarrollar la solución para el problema planteado, se consideran a las imágenes como matrices de pixeles, donde cada pixel se encuentra en formato RGB. Estas imágenes se encuentran almacenadas en archivos de texto, los cuales deben ser leídos por el programa.

Con esta información se pueden considerar los siguientes sub-problemas:

- El algoritmo debe ser capaz de leer la información proporcionada por los archivos de entrada, en el formato pedido.
- Se deben almacenar estos datos en la memoria, para luego proceder con la comparación.
- Generar un archivo de salida, en el cual se pueda visualizar claramente el resultado obtenido.

2.2 DESCRIPCIÓN DE LA SOLUCIÓN

En base a los sub-problemas encontrados en la sección anterior, es que se planteó una solución para cada uno de estos.

2.2.1 Reservación de memoria para matrices

Como representación de las imágenes en el programa, se consideró el uso de matrices tridimensionales, más específicamente, matrices de tamaño *NxMx3,* donde *N* representa la cantidad de filas, *M* la cantidad de columnas y se crean tres capas en la tercera dimensión, donde cada capa representará el color rojo (primera capa), verde (segunda capa) y

finalmente azul (tercera capa). A continuación se muestra un pseudocódigo, donde se puede ver la reservación de memoria para estas matrices.

```
Matriz initMatrix(N, M)
    i, j = 0
    Matriz = reservarMemoria()
    Si Matriz es distinto de nulo, hacer:
        Para i = 0 hasta N, hacer:
            Matriz[i] = reservarMemoria()
            Para j = 0 hasta M, hacer:
                Matriz[i][j] = reservarMemoria()
        Devolver Matriz
    Caso contrario devolver nulo
```

Imagen 2-1: Pseudocódigo de la función initMatrix.

2.2.2 Lectura de archivos

Para la lectura de archivos se considera el uso de la función *initMatrix* para almacenar los datos obtenidos de estos archivos. El siguiente pseudocódigo muestra lo mencionado:

```
Matriz readImage(archivo, *filas, *columnas)
    i, j = 0
    r, g, b, N, M = 0
    Si el archivo existe, hacer:
        Leer(archivo, almacenar filas y columnas en N y M)
        Matriz = initMatrix(N, M)
        Si Matriz es distinto de nulo, hacer:
            Para i = 0 hasta N, hacer:
                Para j = 0 hasta M, hacer:
                    Leer(archivo, almacenar valores del pixel en r, g y b)
                    Matriz[i][j][0] = r
                    Matriz[i][j][1] = g
                    Matriz[i][j][2] = b
            *filas = N
            *columnas = M
            Devolver Matriz
    Caso contrario devolver nulo
```

Imagen 2-2: Pseudocódigo de la función readImage.

La función tiene como propósito leer el archivo de texto, e ir almacenando en la matriz de tres dimensiones definida anteriormente todos estos datos, para luego retornar la matriz creada. A continuación se presenta una figura que representa una imagen una matriz de *2x2x3*, donde se puede ver la representación dada sobre el pixel en RGB.

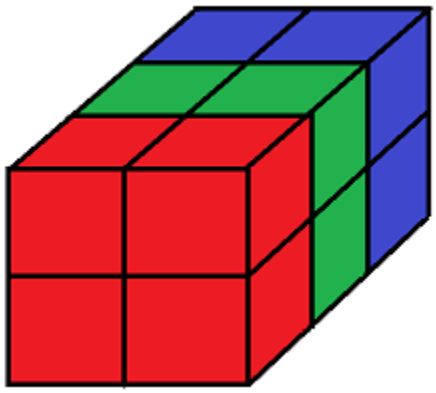

Imagen 2-3: Representación gráfica de una matriz de 2x2x3 en formato RGB.

2.2.3 Comparación de imágenes

Una vez almacenadas las imágenes en matrices tridimensionales, se procede a realizar la comparación entre estas. Para esto se propuso un algoritmo que simula como se realizaría la comparación entre dos imágenes en la vida real, esto es, ubicar de forma estática la imagen principal en alguna superficie, y recorrer por toda esta imagen la sub-imagen de la cual se busca alguna coincidencia. En caso de que no exista coincidencia en la primera iteración, se avanza sobre la imagen principal hacia el siguiente espacio y se repite el proceso de comparación. Este proceso termina si se encontró alguna coincidencia o en su defecto, se recorrió toda la imagen principal y no existió ninguna coincidencia. El siguiente pseudocódigo realiza este procedimiento.

```
Bool comparación(imagenPrincipal, filasMatrizPrincipal, columnasMatrizPrincipal,
subImagen, filasSubImagen, columnasSubImagen)

i, j, x, y = 0

Estado = 1

Si imagenPrincipal es distinto de nulo y subImagen es distinto de nulo, hacer:

    Para i = 0 hasta filasMatrizPrincipal − filasSubImagen, hacer:

        Para j = 0 hasta columnasMatrizPrincipal − columnasSubImagen, hacer:

            Para x = 0 hasta filasSubImagen, hacer:

                Para y = 0 hasta columnasSubImagen, hacer:

                    Si imagenPrincipal[i+x][j+y][0] es distinto de
                    subImagen[x][y][0] ó

                    imagenPrincipal[i+x][j+y][1] es distinto de
                    subImagen[x][y][1] ó

                    imagenPrincipal[i+x][j+y][2] es distinto de
                    subImagen[x][y][2], hacer:

                        Estado = 0

                        x = filasSubImagen

                        y = columnasSubImagen
```

Caso contrario hacer:

Estado = 1

Si Estado equivale a 1, Devolver 1

Caso contrario, Estado = 1

Caso contrario, Devolver 0;

Imagen 2-4: Pseudocódigo de la función comparar.

2.2.4 Resultado de la comparación

Luego de procesar todos los datos, se obtendrá el resultado de la comparación, donde se indicará si las imágenes coinciden o no dentro de la imagen principal. Para esto, el algoritmo, luego de verificar la coincidencia y, dependiendo del valor retornado por la función de carácter booleano *comparar()*, se procede a escribir en un archivo el resultado de estas comparaciones, tomando el formato de *Imagen n: Resultado*, donde *n* es un número natural, que va desde 1 hasta n, y *Resultado* puede tomar los valores de *Encontrada o No encontrada.* Cabe destacar que el algoritmo también verifica las rotaciones de la sub-imagen en 90, 180 y 270 grados.

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

Luego de realizar completamente el algoritmo, se procede medir de manera teórica y experimental los resultados de la ejecución de este. En primera instancia, se calculará el tiempo de ejecución teórico del algoritmo, el cual consiste en sumar la cantidad total de instrucciones en el programa. Para mejor comprensión, se realizará esta operación en cada función, y luego se sumarán todos estos resultados para así calcular el tiempo de ejecución total.

- *initMatrix:* $T_{initMatrix}(n) = 1 + 1 + 1 + 1 + n * 1 * m * 1 + 1$

  $$\Rightarrow T_{initMatrix}(n) = n * m + 5,$$

  $Asumiendo\ n = m\ y\ 5 \sim O(1), entonces:$

  $$T_{initMatrix}(n) = n^2 + O(1)$$

- *readImage:* $T_{readImage}(n) = O(1) + n * m * (1 + 1 + 1 + 1) + O(1)$

  $$\Rightarrow T_{readImage}(n) = 4n^2 + O(1)$$

- *comparacion:* $T_{comparacion}(n) = O(1) + (n - j) * (m - k) * j * k * 11$

  $$Asumiendo\ como\ peor\ caso\ que\ j = k = \frac{n}{2}, entonces:$$

  $$\Rightarrow T_{comparacion(n)} = \frac{11}{16}n^4 + O(1)$$

- *rotar:* $T_{rotar}(n) = O(1) + n * m * 3 + 1$

  $$\Rightarrow T_{rotar}(n) = 3n^2 + O(1)$$

Una vez calculadas todas estas funciones, se procede a calcular el tiempo de ejecución total del programa.

$$T_{algoritmo}(n) = T_{initMatrix}(n) + T_{readImage}(n) + T_{comparacion}(n) + T_{rotar}(n)$$

$$\Rightarrow T_{algoritmo}(n) = \frac{11}{16}n^4 + 8n^2 + O(1)$$

Además, gracias a las leyes de la notación O, se puede decir que el tiempo de ejecución se acota por la función de mayor orden. Para este caso, la función que acota al tiempo de ejecución es n^4. Esto se representa de la siguiente manera:

$$T_{algoritmo}(n) \sim O(n^4)$$

Esto se puede ver en los siguientes gráficos:

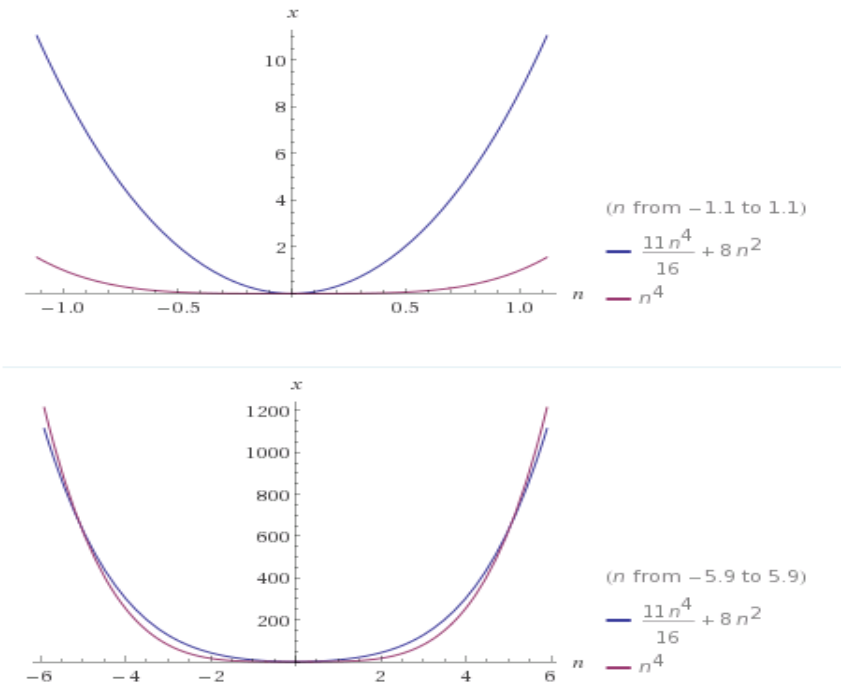

Imagen 3-1: Gráficos de comparación entre T(n) y O(n).

Hay que destacar que el dominio de la función es $n \in N$, por lo que el gráfico solo es válido en el primer cuadrante.

CAPÍTULO 4. CONCLUSIONES

Para concluir con el presente informe, se deben tomar en cuenta dos parámetros. Uno de estos es la eficacia del algoritmo, o dicho de otra forma, que el algoritmo cumpla con su objetivo. En base a este parámetro, el algoritmo se podría considerar exitoso, ya que efectivamente realiza la comparación entre imágenes y cumple con todos los requerimientos solicitados. El otro parámetro a considerar es la eficiencia del algoritmo a la hora de resolver el problema planteado, el cual no llega a ser satisfactorio, debido a que las herramientas que se poseían para realizar el algoritmo eran muy limitadas, lo que provoca que muchas partes de este debieron ser programadas de manera muy básica, y es debido a esto que el algoritmo se demora mucho tiempo en ser ejecutado en presencia de una cantidad de datos de entrada considerablemente grande, lo que en la vida real muestra un gran desperdicio de recursos tanto computacionales como humanos.

En conclusión, se obtuvieron los resultados esperados, tanto a nivel de eficacia como de eficiencia del algoritmo. Se espera para futuras entregas que la brecha entre eficacia y eficiencia del algoritmo disminuya de manera considerable.

CAPÍTULO 5. REFERENCIAS

- Enunciado Laboratorio 1 Análisis de Algoritmos y Estructuras de Datos (Septiembre de 2016). *UdeSantiagoVirtual.* Obtenido de UdeSantiago Virtual Moodle: *http://www.udesantiagovirtual.cl/*
- StackOverflow (Agosto de 2008), creado por Jeff Attwood. Utilizado para dudas varias. *http://stackoverflow.com/*