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

Estructura de Datos y Análisis de Algoritmos

Informe Laboratorio 4

Ariel Ignacio Tirado Maturana

Profesor:
Jacqueline Kohler Casasempere
Alejandro Cisterna Villalobos

Ayudante:
Gerardo Zuñiga Leal

Santiago - Chile
2-2016

TABLA DE CONTENIDOS

Tabla de Contenidos I
Índice de Figuras I
CAPÍTULO 1. Introducción 1
CAPÍTULO 2. Descripción de la solución 2
2.1 Análisis del problema 2
2.1.1 Almacenar la matriz en la raíz del árbol cuaternario 2
2.1.2 Verificar el criterio de uniformidad 3
2.1.3 División de la imagen 3
2.1.4 Fusión 4
2.2 Descripción de la solución 4
2.2.1 Almacenamiento de la matriz en el primer nodo del árbol 4
2.2.2 Verificar el criterio de uniformidad 5
2.2.3 División de la imagen 6
2.2.4 Fusión 7
CAPÍTULO 3. Análisis de resultados 8
CAPÍTULO 4. Conclusiones 10
CAPÍTULO 5. Referencias 10

ÍNDICE DE FIGURAS

Imagen 2-1: Representación en el árbol cuaternario de la división de un nodo. 3
Imagen 2-2: Pseudocódigo de la función verificarCriterio. 5
Imagen 2-3: Pseudocódigo de la función preOrden. 6
Imagen 2-4: Representación de un árbol con todas sus regiones cumpliendo el criterio. 6
Imagen 2-5: Pseudocódigo de la función llenarPreOrden. 7

CAPÍTULO 1. INTRODUCCIÓN

Se puede decir que una imagen digital representada en escala de grises corresponde a una matriz de enteros no negativos con un rango entre *0* y *n*, donde cada uno de estos valores representará un pixel dentro de la imagen. La representación consiste en mostrar el nivel de gris de cada pixel, siendo 0 el negro y *n* el blanco.

Las imágenes digitales a su vez pueden ser representadas computacionalmente de diversas formas. Una de ellas es representarla en un *árbol cuaternario* o *quadtree.* Esta representación resulta muy útil para algunas tareas en específico, como poder verificar regiones uniformes dentro de una imagen.

Para el presente laboratorio, se les ha solicitado a los alumnos del curso de Análisis de Algoritmos y Estructuras de Datos la implementación de un árbol cuaternario, en la cual se almacene en una primera instancia la matriz que representa las escalas de grises de cada pixel de una imagen, para luego verificar su uniformidad y particionar en regiones dicha imagen hasta que todas estas cumplan con el criterio de uniformidad, para que posteriormente, se fusiones las regiones que cumplan con dicho criterio. Esta implementación se deberá realizar en el lenguaje de programación C, el cual es uno de los lenguajes de más renombre entre los lenguajes pertenecientes al paradigma imperativo.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

2.1 ANÁLISIS DEL PROBLEMA

Para el presente laboratorio, se ha propuesto realizar un algoritmo que ejecute cuatro tareas, dado una matriz de entrada. Estas tareas son:

- Almacenar la matriz en el nodo raíz de un árbol cuaternario.
- Verificar el criterio de uniformidad en la matriz.
- En caso de que no se cumpla dicho criterio, dividir la matriz en cuatro regiones y repetir este proceso hasta que se cumpla el criterio de uniformidad para todas las regiones de la imagen.
- Fusionar las regiones adyacentes que cumplan con el criterio de uniformidad.

Para obtener la matriz a estudiar, este será ingresado en un archivo de entrada, el cual tendrá por nombre *"Entrada.in"*. Además se debe tener en consideración que el archivo contendrá en la primera línea la cantidad de filas de la matriz, siendo la dimensión de esta *nxn*, y después viene la matriz de enteros, que oscilan en el rango de 256 niveles de gris. El criterio de uniformidad considerado para realizar las divisiones y fusiones es que la diferencia entre el mayor y mínimo valor de pixeles debe ser de a lo más 10 niveles de gris.

2.1.1 Almacenar la matriz en la raíz del árbol cuaternario

Debido a que la matriz a estudiar será ingresada en un archivo de entrada, se debe leer este y almacenar en una estructura adecuada para su posterior análisis. En este paso se deberá intentar recoger datos que serán utilizados posteriormente para estudiar las propiedades de la matriz. Entre los datos que serán recogidos, se encuentran la matriz propiamente tal y las dimensiones de esta. Estos datos serán almacenados en el primer nodo creado del árbol y representarán la imagen completa.

2.1.2 Verificar el criterio de uniformidad

Como se mencionó anteriormente, el criterio de uniformidad está dado por la relación $mayor\ valor - mínimo\ valor \leq 10$. Para verificar el criterio, el algoritmo debe ser capaz de encontrar el máximo y mínimo valor dentro de la matriz, para luego realizar la diferencia entre ellos y comprobar que se cumpla la condición.

2.1.3 División de la imagen

En caso de que el criterio de uniformidad no se cumpla, el algoritmo debe dividir la imagen en cuatro regiones de dimensiones $\frac{nxn}{2}$, donde cada sub matriz se almacenará en un nodo hijo del nodo que no cumplió con el criterio de uniformidad. A continuación se muestra una imagen que representa lo dicho anteriormente.

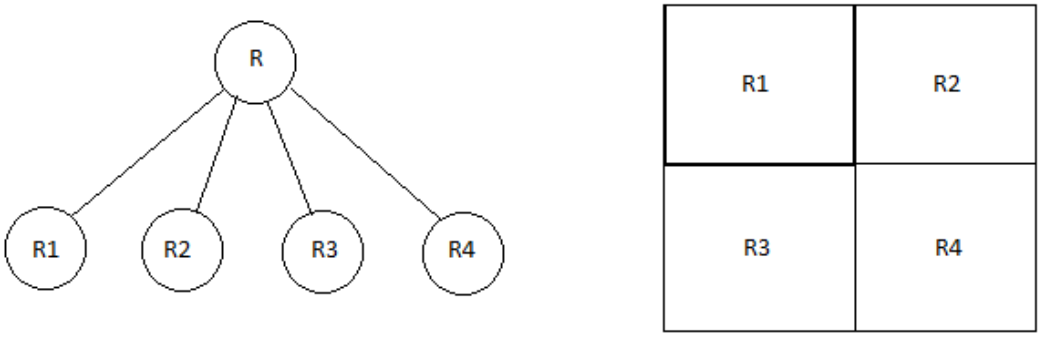

Imagen 2-1: Representación en el árbol cuaternario de la división de un nodo.

Realizando un desglose del problema, se generan los siguientes sub problemas.

- Verificar el criterio de uniformidad de un nodo.
- En caso de no cumplirse el criterio, particionar el nodo en cuatro nodos hijos.
- Realizar este proceso hasta que todas las hojas del árbol cumplan con el criterio.

2.1.4 Fusión

Luego de que el proceso de división termine, el algoritmo debe proceder a fusionar aquellas regiones que al compararlas con sus regiones adyacentes, cumplan con el criterio de uniformidad.

Realizando un desglose del problema, se obtienen los siguientes sub problemas.

- Visitar un nodo y todos sus nodos hermanos.
- Verificar que el nodo que se encuentra en análisis y el nodo hermano con el que se está comparando cumpla el criterio de uniformidad.

2.2 DESCRIPCIÓN DE LA SOLUCIÓN

En base a los problemas obtenidos en la sección anterior, es que se planteó una solución para cada uno de ellos.

2.2.1 Almacenamiento de la matriz en el primer nodo del árbol

Para esto, se consideran en la lectura del archivo dos factores; el primero es encontrar las dimensiones de la matriz, para esto se debe leer la primera línea del archivo, la cual corresponde a un número entero que indica lo que se está solicitando. El segundo paso consiste en leer esta matriz y almacenarla en una variable, la cual será insertada en el primer nodo del árbol, junto con las dimensiones de esta.

La estructura de dato considerada para representar un nodo del árbol contiene además otros datos que se generarán en el proceso de división. Los datos considerados se listan a continuación.

- Matriz que representa la imagen.
- Matriz con las posiciones de la imagen.

- Las dimensiones de la matriz.
- Variable booleana que indica si el nodo es padre o no.
- El valor mínimo y máximo de la matriz.
- Variable booleana que indica si el nodo se ha fusionado o no.
- Puntero hacia el nodo hermano derecho.
- Punteros a los cuatro hijos del nodo.

2.2.2 Verificar el criterio de uniformidad

Para verificar el criterio de uniformidad de un nodo, se implementó un algoritmo, el cual obtiene los valores extremos de la matriz, para luego generar la diferencia entre ellos y comprobar si se cumple el criterio. A continuación se muestra el pseudocódigo de este algoritmo.

```
Función verificarCriterio(Matriz, N, *Mayor, *Menor):
        menor <- Matriz[0][0]
        mayor <- Matriz[0][0]
        Para i = 0 hasta N - 1, hacer:
                Para j = 0 hasta N - 1, hacer:
                        Si menor > Matriz[i][j], entonces:
                                menor = Matriz[i][j]
                        Si mayor < Matriz[i][j], entonces:
                                mayor = Matriz[i][j]
        *Mayor = mayor
        *Menor = menor
        Si mayor - menor > 10, entonces:
                Devolver Falso
        Sino:
                Devolver Verdadero
```

Imagen 2-2: Pseudocódigo de la función verificarCriterio.

2.2.3 División de la imagen

Como se mencionó anteriormente, si un nodo no cumple con el criterio de uniformidad, se debe proceder a dividir este nodo en cuatro regiones, las cuales serán almacenadas en los nodos hijos de este. Para realizar este procedimiento, se hace uso de la función *verificarCriterio* y el recorrido en *preorden* del árbol, verificando de manera recursiva todos los nodos y en caso de que no se cumpla el criterio, realizar la división. A continuación se muestra el pseudocódigo que representa lo dicho anteriormente.

```
Función preOrden(raiz):
        Si raiz != Nulo y verificarCriterio() = 0, entonces:
                raiz.esPadre = 1
                raiz = particionar(raiz)
                preOrden(raiz.primerHijo)
                preOrden(raiz.segundoHijo)
                preOrden(raiz.tercerHijo)
                preOrden(raiz.cuartoHijo)
        Devolver raiz
```

Imagen 2-3: Pseudocódigo de la función preOrden.

Como resultado del proceso recursivo, se obtendrá el árbol resultante, donde además se tendrá una lista enlazada entre todos los nodos hermanos (nodos de un mismo nivel y con un mismo padre).

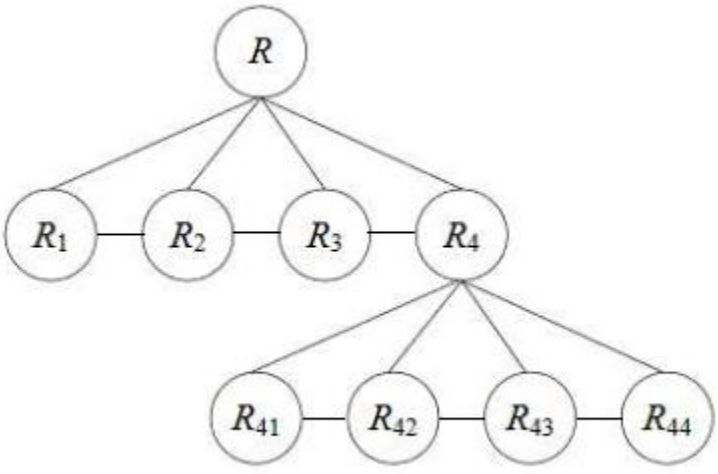

Imagen 2-4: Representación de un árbol con todas sus regiones cumpliendo el criterio.

2.2.4 Fusión

Una vez que todos los nodos han sido divididos y se ha formado completamente el árbol, se procede a realizar el proceso de fusión de este. Para esto se considera el recorrido en *preorden* del árbol, verificando que el nodo no sea padre y que no haya sido fusionado. En caso de cumplirse esta condición, el algoritmo recorre la lista enlazada de nodos hermanos, tomando como referencia el primer nodo, y verificando que se cumpla el criterio de uniformidad. En caso de cumplirse, se escribe la región del primer nodo en las posiciones de los nodos hermanos que cumplieron el criterio en una matriz con el resultado de la fusión. A continuación se muestra el pseudocódigo del algoritmo encargado de este procedimiento.

```
Función llenarPreOrden(raiz, Matriz, region):
        Si raiz != Nulo, entonces:
                matriz = llenarPreOrden(raiz.primerHijo, Matriz, region + 1)
                matriz = llenarPreOrden(raiz.segundoHijo, Matriz, region + 2)
                matriz = llenarPreOrden(raiz.tercerHijo, Matriz, region + 3)
                matriz = llenarPreOrden(raiz.cuartoHijo, Matriz, region + 4)

                Si raiz.fusionado = 0 y raiz.esPadre = 0, entonces:
                        aux = raiz
                        Para i = 0 hasta raiz.N - 1, hacer:
                                Para j = 0 hasta raiz.N - 1, hacer:
                                        Matriz[raiz.matrizPos[i][j][0]][raiz.matrizPos[i][j][1] = region

                Mientras aux.hermanoDerecho != Nulo, hacer:
                        Si raiz.mayor - aux.hermanoDerecho.menor <= 10 y raiz.mayor - aux.hermanoDerecho.menor >= 0, entonces:
                                Si aux.hermanoDerecho.menor < raiz.menor, entonces:
                                        raiz.menor = aux.hermanoDerecho.menor
                                aux.hermanoDerecho.fusionado = 1
                                Para i = 0 hasta raiz.N - 1, hacer:
                                        Para j = 0 hasta raiz.N - 1, hacer:
                                                Matriz[aux.hermanoDerecho.matrizPos[i][j][0]][aux.hermanoDerecho.matrizPos[i][j][1]] = region

                        Si aux.hermanoDerecho.mayor - raiz.menor <= 10 y aux.hermanoDerecho.mayor - raiz.menor >= 0, entonces:
                                Si aux.hermanoDerecho.mayor > raiz.mayor, entonces:
                                        raiz.menor = aux.hermanoDerecho.menor
                                aux.hermanoDerecho.fusionado = 1
                                Para i = 0 hasta raiz.N - 1, hacer:
                                        Para j = 0 hasta raiz.N - 1, hacer:
                                                Matriz[aux.hermanoDerecho.matrizPos[i][j][0]][aux.hermanoDerecho.matrizPos[i][j][1]] = region
                        aux = aux.siguiente
        Devolver Matriz
```

Imagen 2-5: Pseudocódigo de la función llenarPreOrden.

Una vez terminadas las llamadas recursivas de este algoritmo, la matriz resultante con las regiones se escribirá en un archivo de texto, llamado *Salida.out.*

CAPÍTULO 3. ANÁLISIS DE RESULTADOS

Luego de realizar completamente el algoritmo, se procede a medir la eficiencia de este, mediante los conceptos adquiridos en clases. A continuación se muestra el cálculo de la función que representa el tiempo de ejecución del algoritmo.

- *crearMatriz( ):*

$$T_{crearMatriz}(n) = 1 + 1 + n(1 + 1) + 1$$

$$T_{crearMatriz}(n) = 2n + 3$$

- *crearMatrizPos( ):*

$$\begin{aligned} T_{crearMatrizPos}(n) &= 1 + 1 + 1 \\ &+ n\big(1 + 1 + n(1 + 1 + 1) + n(1 + 1 + 1 + 1 + 1 + 1 + 1 + 1)\big) \\ &+ 1 \end{aligned}$$

$$T_{crearMatrizPos}(n) = 11n^2 + 2n + 4$$

- *verificarCriterio( ):*

$$\begin{aligned} T_{verificarCriterio}(n) &= 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + n\big(n(1 + 1 + 1 + 1 + 1 + 1)\big) \\ &+ 1 + 1 + 1 + 1 + 1 + 1 \end{aligned}$$

$$T_{verificarCriterio}(n) = 6n^2 + 16$$

- *particionar( ):*

$$T_{particionar}(n) \sim O(n^2)$$

- *preOrden( ):*

$$T_{preOrden}(n) = 1 + 1 + 6n^2 + 16 + 1 + O(n^2) + T_{preOrden}\left(\frac{n}{4}\right) + T_{preOrden}\left(\frac{n}{4}\right) + T_{preOrden}\left(\frac{n}{4}\right) + T_{preOrden}\left(\frac{n}{4}\right) + 1$$

$$T_{preOrden}(n) = 4T_{preOrden}\left(\frac{n}{4}\right) + O(n^2) + 4 \sim O(n^2)$$

- *llenarPreOrden( ):*

$$T_{llenarPreOrden}(n) = O(1) + T_{llenarPreOrden}\left(\frac{n}{4}\right) + T_{llenarPreOrden}\left(\frac{n}{4}\right) + T_{llenarPreOrden}\left(\frac{n}{4}\right) + T_{llenarPreOrden}\left(\frac{n}{4}\right) + 1 + 1 + 1 + n\big(n(13)\big) + 4\Big(22 + n\big(n(13)\big)\Big) + 2 + 1$$

$$T_{llenarPreOrden}(n) = 4T_{llenarPreOrden}\left(\frac{n}{4}\right) + O(n^2) \sim O(n^2)$$

- *Fusionar( ):*

$$T_{fusionar}(n) = 1 + 1 + 1 + 1 + 1 + n(n(3) + 1) + 1 + O(n)$$

$$T_{fusionar}(n) = 3n^2 + O(n) + 6$$

Finalmente, el tiempo de ejecución del programa es el siguiente.

$$T(n) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + n\big(n(3)\big) + 1 + 1 + 1 + T_{preOrden}(n) + T_{fusionar}(n)$$

$$T(n) = 3n^3 + O(n^2) + 3n^2 + O(n) + 6 + 12$$

$$T(n) = 6n^2 + O(n) + 18 \sim O(n^2)$$

CAPÍTULO 4. CONCLUSIONES

Para concluir con el presente informe, se deben tomar en cuenta dos parámetros. El primero de ellos es la eficacia del algoritmo, o dicho de otra forma, que el algoritmo cumpla con su objetivo. En base a este parámetro, se podría considerar en su mayoría exitoso, ya que si bien logra dividir y fusionar sin problemas una imagen, el resultado no muestra las regiones en un rango de *1* a *n*, sino que lo hace con el número correspondiente a la región del nodo que se está estudiando, pudiéndose prestar a confusiones. El otro parámetro a considerar es la eficiencia del algoritmo, la que se puede cuantificar mediante la función que expresa el tiempo de ejecución, la cual resultó ser de orden $O(n^2)$. Esto puede ser considerado eficiente en entradas con poca cantidad de datos, pero en presencia de una gran cantidad de estos, el tiempo de ejecución aumenta significativamente. Esto puede mostrar un problema en la vida real, ya que si se pretende utilizar el algoritmo con el fin de estudiar imágenes con gran cantidad de pixeles, el algoritmo demoraría mucho tiempo en entregar una respuesta.

Siendo esta la última entrega de laboratorio del curso, se puede concluir que la eficiencia de los algoritmos fue aumentando favorablemente a medida que se iban aprendiendo nuevas estructuras de datos, las cuales iban a su vez acompañadas de algoritmos diseñados para ser eficientes y eficaces. Se espera que los contenidos aprendidos en estas cuatro entregas de laboratorio sean de utilidad, tanto para la experiencia académica como laboral.

CAPÍTULO 5. REFERENCIAS

- Enunciado Laboratorio 4 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/*