|  |
| --- |
| **Estructura de Datos y Análisis de Algoritmos**  **Experiencia 4: Árboles.** |
|  |
| **Andrés Felipe Muñoz Bravo**  **19.646.487-5** |

|  |  |
| --- | --- |
|  | Profesor: |
|  | Mario Inostroza Ponta  Ayudante:  Javiera Torres Muñoz |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | Santiago - Chile |  |
|  | 2016 |  |

Tabla de Contenidos

[Tabla de Contenidos I](#_Toc467789021)

[Índice de Figuras II](#_Toc467789022)

[Índice de Tablas II](#_Toc467789023)

[CAPÍTULO 1. Introducción 3](#_Toc467789024)

[CAPÍTULO 2. Descripción de la solución 4](#_Toc467789025)

[1.1 Estructuras 4](#_Toc467789026)

[1.1.1 Imagen: 4](#_Toc467789027)

[1.1.2 Árbol: 5](#_Toc467789028)

[1.2 Funciones 6](#_Toc467789029)

[1.2.1 Partición masiva (Arbol\* arbol, int condicion,int\* id,Imagen\* imagen): 6](#_Toc467789030)

[1.2.2 marcar(Arbol\* arbol,Imagen\* imagen,int\* id): 7](#_Toc467789031)

[1.2.3 marcarGrupos(Arbol\* arbol, int condicion,int\* id,Imagen\* imagen): 7](#_Toc467789032)

[1.2.4 particionarArbol(Arbol\* arbol): 8](#_Toc467789033)

[CAPÍTULO 3. Análisis de los resultados 9](#_Toc467789034)

[CAPÍTULO 4. Conclusión 11](#_Toc467789035)

[CAPÍTULO 5. Referencias 11](#_Toc467789036)

Índice de Figuras

[Ilustración 1: Ejemplo grafo para algoritmo de búsqueda. 7](#_Toc465349148)

[Ilustración 2: Resultado del algoritmo de búsqueda 7](#_Toc465349149)

[Ilustración 3: Resultado algoritmo ordenamiento por centralidad de grado. 8](#_Toc465349150)

Índice de Tablas

[Tabla 1: Tiempo y orden de funciones para lista. 9](#_Toc465349151)

[Tabla 2: Tiempo y orden de funciones para cola. 9](#_Toc465349152)

[Tabla 3: Tiempo y orden de funciones para grafos. 10](#_Toc465349153)

# Introducción

Se ha pedido realizar un programa para analizar características de los grafos, estos grafos son entregados a través de un archivo de texto. El programa lee este archivo para posteriormente trabajar según la necesidad del usuario y los objetivos implementados mencionados a continuación. El análisis según las características del grafo es muy importante dentro de la informática, porque con estos se pueden representar una infinidad de cosas, como por ejemplo, verificar si es que existe una ruta que conecte dos ciudades.

Objetivos:

1. Implementar un TDA grafo, con funciones para cargar desde un archivo de texto.
2. Verificar si el grafo es conexo o no, esto quiere decir que hay que verificar si hay camino entre todo par de vértices.
3. Ordenar vértices del grafo según la centralidad de grado, esto quiere decir ordenar los vértices según la cantidad de vértices adyacentes que tiene cada uno de ellos.
4. Ordenar vértices del grafo según la centralidad de betweenness, esto quiere decir ordenar, según el cociente entre la cantidad de veces que un nodo está presente entre los caminos mínimos entre dos nodos y el total de esos caminos mínimos.

Para llevar a cabo este programa se ha trabajado con listas enlazadas circulares, utilizando el lenguaje de programación C, con el paradigma de programación imperativo procedural.

# Descripción de la solución

Para el desarrollo de este programa es utilizada la herramienta sublime text 3, el cual es un editor de texto. La principal ventaja de este editor es que utiliza colores para las diferentes palabras reservadas del lenguaje de programación C, de esta forma ayuda al programador revisar la sintaxis programada. A continuación se explicara detalladamente las estructuras herramientas, técnicas y las principales funciones del programa realizado.



## Estructuras

Para el programa se utilizan dos estructuras; Imagen y Arbol. Con estas dos estructuras es posible llevar a cabo el desarrollo del programa. A continuación se detallaran las componentes de estas.

### Imagen:

Dentro de la estructura de Imagen se encuentran definido 5 datos:

1. Un doble puntero a entero, con eso se logra una matriz de enteros para guardar los valores de cada pixel de la imagen.
2. Un entero para guardar la dimensión de la imagen, cabe recalcar que solo basta con un valor dado que la imagen debe ser cuadrada.
3. Un entero para guardar la uniformidad de la imagen.
4. Un entero para guardar el valor máximo entre los pixeles guardados en la matriz.
5. Un entero para guardar el valor mínimo entre los pixeles guardados en la matriz.

### Árbol:

Dentro de la estructura árbol se encuentran definido 4 datos:

1. Puntero a una estructura imagen.
2. Entero para guardar la fila desde donde comienza la imagen al momento de ser particionada.
3. Entero para guardar la columna desde donde comienza la imagen al momento de ser particionada.
4. Arreglo de 4 punteros a la misma estructura árbol, para guardar las 4 particiones de la imagen.

## Funciones

### Partición masiva (Arbol\* arbol, int condicion,int\* id,Imagen\* imagen):

Esta es la función principal del programa, la cual recibe un árbol con la imagen inicial carga en él, un entero con la condición de uniformidad que debe cumplir cada partición de la imagen, un id pasado por referencia para marcar los diferentes grupos, y una imagen pasada por referencia también en la cual se guardaran las marcas realizadas.

Cabe recalcar que dentro de esta función subyacen más funciones para llevar a cabo el proceso de partición total y fusión posterior. A continuación se presenta el algoritmo empleado.

Algoritmo:

* Si la imagen del árbol tiene uniformidad mayor a la condición entregada y la dimensión de la imagen del árbol es mayor a uno:
  + Se realiza una partición al árbol.
  + Se hace una llamada recursiva entregando como árbol el primer hijo de la partición realizada.
  + Se hace una llamada recursiva entregando como árbol el segundo hijo de la partición realizada.
  + Se hace una llamada recursiva entregando como árbol el tercer hijo de la partición realizada.
  + Se hace una llamada recursiva entregando como árbol el cuarto hijo de la partición realizada.
  + Se marcar los grupos respectivos si cumplen la condición necesaria.
* En caso contrario :
  + Se pregunta si el árbol entregado es hoja, en caso afirmativo hacer:
    - Marcar el grupo entero de la imagen entregada, esto quiere decir que se pinta la imagen marca la imagen completa debido a que cumple con la condición de uniformidad.

### marcar(Arbol\* arbol,Imagen\* imagen,int\* id):

Esta función marca en una imagen con el id entregado todas las posiciones de la imagen del árbol, pero es necesario saber en qué parte de la imagen principal pertenece esta subimagen del subárbol del árbol principal entregado, esta parte es muy difícil de explicar, por lo que se presenta el algoritmo a continuación:

Algoritmo:

* fil = arbol.fila (asignacion)
* col = arbol.col (asignacion)
* para i=0 hasta arbol.imagen.dimension:
  + para j = 0 hasta arbol.imagen.dimension:
  + imagen.matriz[i][j] = \*id; (asignación y desreferenciacion)

### marcarGrupos(Arbol\* arbol, int condicion,int\* id,Imagen\* imagen):

Función que verifica la uniformidad de las imágenes adyacentes y marca aquellas imágenes que cumplen con la condición de uniformidad al momento de unirlas, para eso se entrega el mismo id a la función marcar (anteriormente detallada).

### particionarArbol(Arbol\* arbol):

Esta función realiza la partición de un árbol, creando los cuatro hijos, los cuales son nuevos árboles con sus respectivas imágenes particionadas desde el árbol anterior. A continuación se presenta el algoritmo:

Algoritmo:

* Se inicializan los cuatros arboles dentro del arreglo hijos contenido en el árbol
* Se inicializan 4 imágenes
* Para i = 0 hasta la dimensión de la imagen del árbol principal
  + Para j = 0 hasta la dimensión de la imagen del árbol principal
    - Se ingresan los datos la partición respectiva a cada imagen inicializada anteriormente.
* Se actualiza la información de las imágenes creadas, esto quiere decir que se calcula la uniformidad, el mínimo y máximo para cada una de ellas.
* Se carba a los arboles inicializados (hijos) las imágenes creadas (particiones)
* Se actualiza la fila y columna de cada hijo, esto se refiere a la ubicación en la cual inicia cada imagen particionada dentro de la imagen principal.

# Análisis de los resultados

Como se puede apreciar las funciones que obtiene mayor orden es la de cargar un grafo. Al analizar detalladamente esta función, se encontró una solución para disminuir el tiempo y orden de esta. La modificación es bien simple, eliminar la matriz de adyacencia de las características del grafo. Al realizar esta eliminación no es necesario cargar la matriz de adyacencia con los respectivos valores y justamente este último paso es el que hace que la función obtenga un orden O(n^3). De esta forma el orden resultante seria O(n^2) lo cual hace que el programa sea mucho más eficiente.



## Tiempos y orden de las funciones

A continuación se muestran todas las funciones del programa con su respectivo orden:

|  |  |  |
| --- | --- | --- |
| Función | T(n) | Orden |
| existsFile | 5 | O(n) |
| rtrim | n | O(n) |
| inicializarMatriz | n^2+n+4 | O(n^2) |
| showMatriz | n^2+n+2 | O(n^2) |
| inicializarImagen | n^2+7n+9 | O(n^2) |
| showImagen | 2n+3 | O(n) |
| actualizarInfo | 2n^2+3n+9 | O(n^2) |
| destruirImagen | 20n+20 | O(n) |
| createImagen | 2n^3+22n^2+29n+17 | O(n^3) |
| cargarImagenArbol | 4n^2+21n+31 | O(n^2) |
| particionarArbol | 4n^2+22n+31 | O(n^2) |
| marcar | 2n^2+4n+3 | O(n^2) |
| marcarGrupos |  |  |
| uniformidadDosImagenes |  |  |
| uniformidadTresImagenes |  |  |
| particionMasiva |  |  |
| esHoja |  |  |
| recorrido |  |  |
| esNumero |  |  |
| titulo |  |  |

Tabla 3: Tiempo y orden de funciones para grafos.

# Conclusión

En esta cuarta entrega se cumplió el objetivo completamente. Dentro lo los objetivos que se cumplieron están implementar el grafo con las funciones para cargar desde un archivo de texto, otro objetivo cumplido es el de verificar si un grafo es conexo, y el último objetivo cumplido fue el de ordenar los vértices según la centralidad de grado.

La importancia de este laboratorio, es que sirvió para entender de una forma más práctica el cómo funciona la representación de árboles dentro de una computadora, además de cómo operan los algoritmos de búsqueda para estos.

Cabe recalcar que estas representaciones y problemas tan complejos y complicados de resolver para una persona o incluso cientos de personas, pueden ser resueltos de una forma mucho más rápida por un computador, solo basta representar de una buena forma.

# Referencias

Departamento de Ingeniería Informática USACH. (2016). Enunciado Laboratorio 3. 2016: USACH.

DCC Universidad de Chile (2007). *Tipos de datos abstractos*. (2016) <https://users.dcc.uchile.cl/~bebustos/apuntes/cc30a/TDA/>.

Departamento de Ingeniería Informática USACH. (2016). Apuntes de la asignatura, Análisis de algoritmos y estructura de datos. Jacqueline Köhler C. 2016: USACH.