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

Estructura de Datos y Análisis de Algoritmos

Experiencia 3: Grafos.

Andrés Felipe Muñoz Bravo
19.646.487-5

Profesor:
Mario Inostroza Ponta
Ayudante:
Javiera Torres Muñoz

Santiago - Chile
2016

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1. 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. Estas representaciones y problemas tan complejos 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.

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.

CAPÍTULO 2. 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.

1.1 ESTRUCTURAS

Para el programa se utilizan tres estructuras; Nodo, List y Grafo. Estas dos primeras estructuras nos ayudan a implementar el TDA lista circular doblemente enlazada. Cabe mencionar que se implementó una pseudo cola con funciones del TDA List.

1.1.1 Nodo:

Dentro de la estructura de nodo se encuentran definido 3 datos:

1. Un número (int) representa el dato a guardar en la lista.
2. Un puntero a un nodo siguiente.
3. Un puntero a un nodo anterior.

1.1.2 List:

Dentro de la estructura List se encuentran definido 3 datos:

1. Un número (int) representa el largo de la lista.
2. Un puntero al primer nodo de la lista.
3. Un puntero al último nodo de la lista.

Algunas funciones implementadas para el manejo de estas son:

- Add: Para agregar un elemento al final de la lista
- Find: Para buscar una letra dentro de la lista
- getNumber: Para obtener una letra de la lista, según una posición.
- showListFL: Para mostrar el contenido de la lista.
- moveLeft: Para mover todos los datos de la lista una posición a la izquierda.
- listcpy: Para copiar una lista.

1.1.3 Grafo:

Dentro de la estructura Grafo se encuentran definido 5 datos:

1. Un número (int), representa el número de vértices del grafo.
2. Una lista de adyacencia, la cual es un arreglo del largo de la cantidad de vértices. Este arreglo contiene listas (TDA List).
3. Una matriz de adyacencia, conformada por numero enteros (int).
4. Un arreglo de enteros del largo de la cantidad de vértices, utilizado para hacer marcas o guardar información respectiva a cada nodo.

1.2 FUNCIONES

1.2.1 Búsqueda:

El algoritmo de búsqueda implementado en este programa es la búsqueda por anchura, para implementarlo es necesario un TDA cola, el cual fue implementado previamente. Este algoritmo utiliza los datos guardados en el TDA grafo. El pseudo código es el siguiente:

- Sea “grafo” un grafo entregado a través del paso por referencia.
- Sea “vertice” el vértice desde el cual se quiere comenzar la búsqueda entregado a través de paso por valor.
- Se ingresa 0 para cada posicion del arreglo de marcas en el grafo.
- Se define “cola” como una cola.
- Se agrega “vertice” a la cola (encolar “vertice”).
- Mientras la cola contenga algún elemento.
    - Se define “v” como el primer elemento de la cola.
    - Se ingresa un 2 en el arreglo de marcas en la posición “v”.
    - Para cada elemento adyacente al vértice “v” (ubicados en la lista de adyacencia del grafo).
        - Se define “w” como cada elemento de la lista de adyacencia del vértice “v”.
        - Si el arreglo de marcas en la posición “w”, contiene un 0.
        - Se agrega un 1 al arreglo de marcas en la posición “w”.
        - Se agrega “w” a la cola (encolar “w”).
    - Se elimina el primer elemento de la cola (desencolar “w”).

Al término del algoritmo de búsqueda queda en el arreglo de marcar dentro del grafo un 2 en la posición de cada vértice visitado.

Ejemplo: Grafo para explicación:

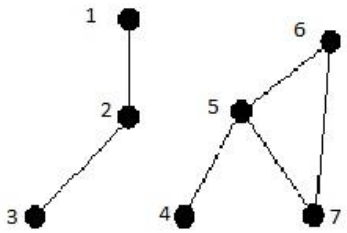

Ilustración 1: Ejemplo grafo para algoritmo de búsqueda.

El grafo anterior contiene 7 vértices, si se entrega este grafo y el vertice de partida es el 4, entonces el arreglo de marcas quedaría de la siguiente forma.

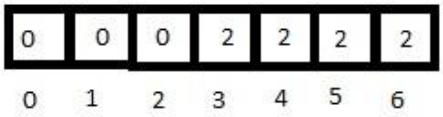

Ilustración 2: Resultado del algoritmo de búsqueda

Este resultado es debido a que no existe conexión entre los subgrafos.

1.2.2 Verificar si el grafo es conexo:

El algoritmo para verificar si un grafo es conexo, (que exista camino entre todo par de vértices), es muy simple pero es necesario utilizar el algoritmo de búsqueda de anchura implementado previamente. El psudo código es el siguiente:

- Se aplica el algoritmo de búsqueda entregando como paso por referencia el grafo a analizar, además de entregar el primer nodo de este como paso por valor. Luego de realizar este paso, el arreglo de marcas contenido en el grafo queda modificado con marcas en todos los nodos visitados.

- Para cada elemento del arreglo de marcas:

    - Si el elemento es distinto de 2:

        - Retornar falso, debido a que hubo al menos un vértice que no se visitó.

- Retornar verdadero, debido a no existió ningún elemento distinto a 2. En otras palabra existe se pudo llegar a todos los vértices existente en el grafo

1.2.3 Ordenar vértices según la centralidad de grado:

- Se recibe el grafo a través de un paso por referencia.
- Se cambia cada valor del arreglo de marcas por un 0.
- Se cambia cada valor del arreglo según el grado de cada vértice, el grado se conoce a partir de la longitud de la lista de adyacencia.
- Se busca el máximo valor de este arreglo
- Mientras máximo sea mayor que 0:
    - Se recorre el arreglo entregando por consola los elementos que contengan el valor igual a máximo
    - Se resta 1 a máximo.

Como se puede apreciar no se implementa un algoritmo de ordenamiento, sino que se entregan los valores a medida que se van encontrando.

Ejemplo del resultado entregado en la consola:

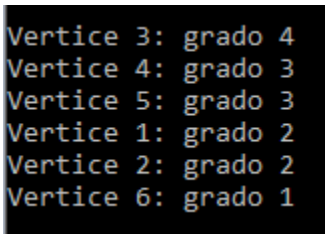

Ilustración 3: Resultado algoritmo ordenamiento por centralidad de grado.

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

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

Función	T(n)	Orden
createNodo	6	O(1)
createList	5	O(1)
add	12	O(1)
invert	13n+7	O(n)
deleteNodo	n+9	O(n)
deleteList	n^2+10n+8	O(n^2)
find	2+2n	O(n)
getNumber	2+2n	O(n)
showListFL	2n+5	O(n)
showListLF	2n+5	O(n)
listcpy	13n+7	O(n)

Tabla 1: Tiempo y orden de funciones para lista.

Función	T(n)	Orden
incio	2+2n	O(n)
encolar	12	O(1)
desencolar	n+9	O(n)
vacia	2	O(1)

Tabla 2: Tiempo y orden de funciones para cola.

Función	T(n)	Orden
existsFile	5	O(n)
rtrim	n	O(n)
inicializarMatrizAdy	n^2+n+4	O(n^2)
showMatriz	n^2+n+2	O(n^2)
inicializarGrafo	n^2+7n+9	O(n^2)
destruirGrafo	2n+3	O(n)
mostrarGrafo	2n^2+3n+9	O(n^2)
verticesAdy	20n+20	O(n)
cargarGrafo	2n^3+22n^2+29n+17	O(n^3)
busqueda	4n^2+21n+31	O(n^2)
verificarConexo	4n^2+22n+31	O(n^2)
centralidadGrado	2n^2+4n+3	O(n^2)

Tabla 3: Tiempo y orden de funciones para grafos.

Como se puede apreciar la función 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.

CAPÍTULO 4. CONCLUSIÓN

En esta tercera entrega no 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.

El único objetivo no cumplido fue el de ordenar los vértices según la centralidad de betweenness, el motivo por el cual no se cumplió fue solamente por tema de tiempo, debido a la cantidad de trabajos pedidos en otras asignaturas.

La importancia de este laboratorio, es que sirvió para entender de una forma más práctica el cómo funciona la representación de grafos dentro de una computadora, además de cómo operan los algoritmos de búsqueda para estos. Si bien no se cumplieron todos los objetivos se aprendió mucho sobre grafos, lo cual sirve para tener esta herramienta en trabajos futuros.

CAPÍTULO 5. 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.