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

Estructura de Datos y Análisis de Algoritmos

Laboratorio 3 : Redes Sociales

Bastián Gonzalo Vera Palacios

Profesor: Jacqueline Köhler

Ayudantes: Nicole Henríquez

Sebastián Vallejos

Javiera Torres

Santiago - Chile
2-2017

Tabla de Contenidos

Índice de Figuras

CAPÍTULO 1. INTRODUCCIÓN

La tecnología se ha ido desarrollando con el paso de los años y junto con ella los medios de comunicación, de tal modo que las redes sociales ganan cada año más terreno en el ámbito de lo virtual y social, ya que estas permiten que gente de todo el mundo pueda interactuar en relación a sus intereses o diversos motivos, tanto sociales, de trabajo, familiares, entre otros.

Estas redes se pueden representar a través de "grafos", que corresponden a un tipo de dato abstracto que permite identificar relaciones recíprocas, en los cuales cada vértice o nodo representa a una persona y las líneas que los unen, llamadas enlaces o aristas significan que aquellos individuos son "amigos", es decir, ambos se tienen mutuamente como amigos y la red completa que se forman gracias a las diversas relaciones entre distintas personas corresponde a un grafo conexo.

En este laboratorio se pide desarrollar mediante una serie de algoritmos en lenguaje C un programa que reciba una red y posibilite determinar en primer lugar los agentes de vínculo, los cuales corresponden a individuos que al ser eliminados forman dos o más componentes conexas que se encuentran unidos a través de él, en este caso la labor del programa será identificar cuantos agentes de vínculo existen en el grafo conexo y a la vez cuantas componentes se crean a partir de su eliminación. En segundo lugar los algoritmos del programa deben hallar si en la red ingresada existe al menos un "grupo de mejores amigos" que corresponden a cuatro individuos que se tienen agregados entre sí.

Los métodos que se serán utilizados para desarrollar los algoritmos del programa en lenguaje C, serán principalmente listas y matrices de adyacencia, las cuales serán comparadas durante el desarrollo de este informe con el objetivo de identificar cual de ellas es más rápida y eficaz a la hora de realizar los procedimientos de identificación de agentes de vínculos y grupos de mejores amigos. El modo de emplear las matrices de adyacencia será a través de arreglos bidimensionales y en la caso de las listas de adyacencia será a través de el uso de TDA de listas.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

2.1 MARCO TEÓRICO

Una vez revisado el problema, se analiza cada uno de los contextos planteados para cada una de las soluciones. Una de ellas es la implementación de una matriz de adyacencia y por otro lado, la implementación de una lista de adyacencia. Ambas se deben generar según un archivo de texto que debe ser leído por el programa llamado Entrada.in el cual consiste en un primer valor que representa la cantidad de vértices presentes en el grafo, seguido por una serie de aristas (conexiones de vertice a vertice) que indican que vértices se encuentran unidos entre sí.

Después de que se tenga el grafo del archivo ingresado en la lista/matriz de adyacencia, se procede a realizar una búsqueda respecto a lo solicitado en el enunciado. Lo primero que se plantea es la búsqueda de un "grupo de mejores amigos", el cual consiste en buscar 4 "personas" que tengan amistad entre sí, en otras palabras más específicas, se desea buscar un clique de 4 nodos en nuestro grafo. Para poder desarrollar esta parte, se plantea como solución generar todo el espacio muestral de combinaciones entre 4 nodos dentro de nuestro grafo y a través de él ir filtrando las combinaciones válidas con el fin de generar soluciones. Esta solución es válida tanto para la matriz de adyacencia como para la lista de adyacencia.

Por otra parte, se solicita que se entregue los agentes de vínculo del grafo de amigos, tambien llamado punto de articulación, el cual consiste en aquellas personas que al eliminarlos transforman la red en dos redes "independientes". Para el desarrollo de este otro problema, se plantea el uso de una búsqueda en profundidad en el grafo entregado con el fin de recorrer todos los nodos e ir generando a la vez un nuevo grafo con forma de árbol, donde nuestro nodo a revisar seria la raiz de este árbol, esto con el fin de poder revisar la existencia de hijos de este nodo. La cantidad de hijos que posea la raiz, sera la cantidad de nodos convexos que se creen al remover ese nodo, por ende, si se da el caso de que el nodo raíz tenga 2 o más hijos, implica que este sería un agente de vínculo. Al igual que en el problema anterior, esta solución se plantea tanto como para la matriz de adyacencia com para la lista de adyacencia.

Una vez resuelto el problema para ambos casos, se medirán los tiempos de ejecución en ambas soluciones con el fin de evaluar la efectividad de ambos métodos empleados y realizar una comparación entre ellos.

2.2 HERRAMIENTAS Y TÉCNICAS

2.2.1 Librerías

Entre de las herramientas utilizadas para el desarrollo de este laboratorio, se encuentran las librerías de:

- stdio.h: Para la entrada, salida de datos y manejo de archivos.
- stdlib.h: Utilizada para el manejo de la memoria dinámica.
- time.h: Utilizada para calcular el tiempo de ejecución de los algoritmos.

2.2.2 Punteros

Los punteros son elementos bastante utilizados durante el desarrollo del código debido a que permiten trabajar directamente a las direcciones de memoria de las estructuras de datos que tenemos. Se utilizan tanto como para lectura del archivo como para poder acceder a las direcciones de estructuras dinámicas creadas con anterioridad según las dimensiones esperadas.

2.2.3 Listas

Señaladas inicialmente ya que se nos solicita la creación de una lista de adyacencia esta estructura similar a un arreglo nos permite generar un índice sobre el cual coinciden los vértices conexos unos con otros. Para esto se emplean funciones que son parte del TDA como: *agregar(), mostrar() o imprimir()*.

2.2.4 Memoria Dinámica

Para un óptimo avance en este codigo se utilizo memoria dinámica tanto en la matriz de adyacencia, para generar un arreglo dinamico en dos dimensiones, como en la lista de adyacencia para poder manejar los valores con un arreglo dinámico en cada nodo logrando así una mejor ejecución al momento de plantear una solución. Para esto se utilizó la función *calloc()* para poder asignar a la vez con el valor 0 cada vez que se pedía memoria y la función *free()* para ir liberando la memoria una vez que esta ya ha sido utilizada.

2.2.5 Estructuras

Las estructuras son un tipo de dato utilizado para poder agrupar distintos tipos de información y manejar datos que sería muy complicado de realizar de manera más primitiva. En el desarrollo de este codigo se utilizan para la implementación de listas ya que así se puede emplear de mejor manera conceptos como el nodo con la información.

2.3 ALGORITMOS Y ESTRUCTURAS DE DATOS

2.3.1 Estructuras Utilizadas

2.3.1.1 Fila

Estructura utilizada como nodo el cual contiene la el valor numérico del vértice, un arreglo dinamico, el largo del arreglo y un puntero hacia el nodo siguiente.

2.3.1.2 Matriz

Estructura que que posee el inicio y el fin de la lista y a la vez la cantidad de vértices del grafo.

2.3.1.3 grafoMatriz

Estructura que posee la matriz bidimensional y a la vez la cantidad de vértices del grafo.

2.3.2 Algoritmos Más Importantes

2.3.2.1 buscarCliques( )

Entradas: - grafoMatriz * ptr(puntero a la matriz)

- int vértices (cantidad de vértices del grafo)

Salidas: - Sin salidas.

Función encargada de realizar la búsqueda de 4 vértices que se encuentren conectados entre sí, formando un grupo de amigos. La función consiste en hacer una busqueda a todo el espacio solución entre cuatro posibles nodos logrando hacer todas las combinaciones posibles entre ellos para poder encontrar los resultados. Los valores a evaluar son los que se encuentran en la parte triangular superior de la matriz, ya que si revisamos la matriz completa, se repetirían resultados ya obtenidos pero en otro orden, lo cual no es necesario y a la vez es poco eficiente.

$T(n) = 5^4$

2.3.2.2 busquedaEnProfundidad( )

Entradas: - grafoMatriz * ptr (puntero a la matriz)

- int vértices (cantidad de vértices del grafo)

- int * visited (puntero a arreglo con los nodos visitados)

- int ** grafoRecorrido (puntero a matriz con nuevo recorrido del grafo)

- int cont (variable contador)

- int i (valor a evaluar dentro de la función)

Salidas: - Sin salidas.

Función que realiza un recorrido al grafo ingresado y genera otro de salida que corresponderia al mismo grafo pero recorrido por profundidad con el fin de poder generar un grafo con forma de árbol, donde nuestro vértice a evaluar será la raíz de este árbol y a su vez, el resto de los valores quedarán como ramificaciones o hijos.

$T(n) = n + 1 + t(n - 1)$

2.3.2.3 buscarArticulacion( )

Entradas: - grafoMatriz * ptr (puntero a la matriz)

- int vértices (cantidad de vértices del grafo)

Salidas: - Sin salidas.

Función encargada de realizar la revisión de cada vértice dentro del nuevo grafo generado por la función *busquedaEnProfundidad()* con el fin de revisar si el vértice, siendo la raíz del árbol, posee dos o más hijos, lo que indica que nos encontramos con un vértice que es punto de articulación o agente de vínculo.

$O(n) = aprox.\ ^{3}$

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

Estos son los T(n) calculados del código:

Tabla 1.1: Tabla de tiempos de ejecución

Funciones	T(n)
crearLista ( )	4
agregarEnLista ( )	$4n + 1$
verticesEnLista ( )	$4n + 3$
MostrarLista( )	$4n + 1$
liberarMemoria( )	$2n + 2$
imprimirMatriz( )	n^2+n
archivoMatriz( )	$4n + 4$
iniciarMatriz( )	$n + 1$
cantVertices( )	2
verticeInicial( )	2
busquedaEnProfundidad( )	$n + 1 + n(n - 1)$
buscarArticulacion ( )	$aprox.\ n^3$
buscarCliques( )	$5n^4$
matriz( )	$5n^4+3n^3 + 10n^2 + 13n + 3$
lista( )	$n^5 + 5n^4 + 6n^3 + 14n^2 + 21n + 9$
main( )	$n^5 + 10n^4 + 9n^3 + 24n^2 + 34n + 13$

El orden de complejidad de este codigo es O(n) = n^5

Durante el desarrollo de la matriz de adyacencia se logra un rendimiento óptimo durante la búsqueda de grupos de mejores amigos logrando un menor tiempo que el esperado por la lista de adyacencia. La función con la que se ha pensado implementar la búsqueda del agente de vínculo en la lista y la matriz de adyacencia desafortunadamente no se ha logrado desarrollar en su totalidad por lo que no se pudo obtener resultados con los cuales comparar los obtenidos anteriormente.

Pese a los lamentables inconvenientes en el desarrollo del problema, se logra implementar el algoritmo para revisar la existencia del grupo de mejores amigos (tambien llamado clique) el cual se encuentra en perfecto funcionamiento entregando los valores de los vértices, la lectura del archivo y la inserción de estos datos a la matriz de adyacencia, el TDA respectivo en el caso de las listas de adyacencia y las respectivas liberaciones de memoria para cada caso.

Respecto a las falencias detectadas durante el desarrollo del laboratorio, la más significativa respecto los tiempos de ejecución del programa, podemos ver como la función *buscarClique()* posee un alto gasto debido a generar todas las combinaciones posibles entre los vértices a evaluar, por ende implica operaciones o valores que puede que exista una manera óptima de desarrollarla sin necesidad de generar todo el espacio Solución.

CAPÍTULO 4. CONCLUSIONES

Los objetivos planteados en este laboratorio no se cumplen de manera efectiva debido a que algunas funciones no se han podido realizar en su totalidad impidiendo el desarrollo de este. De todos modos esto no impide el funcionamiento parcial del código, tanto en Windows como en Linux, pudiendo así realizar ciertas solicitudes del problema como el caso de el grupo de mejores amigos.

Apesar de que no cumple ciertas funciones, el manual de usuario representa el código como si estuviera en completo funcionamiento.

En este laboratorio se ha fomentado de manera fuertemente a pensar bien en la manera más efectiva para resolver el problema, ya que, en los laboratorios anteriores se trabajó respecto al aprendizaje de nuevas herramientas. Durante este laboratorio el trabajo fue más analitico ya que habia que evaluar la mejor manera de poder desarrollar los problemas entregados, como es el caso del agente de vínculo, el cual se pudo haber realizado tanto como por búsqueda por profundidad como búsqueda por anchura.

Se espera que en una próxima oportunidad se pueda desarrollar el laboratorio en su totalidad y poder obtener resultados más concretos para un correcto análisis del trabajo realizado.

CAPÍTULO 5. REFERENCIAS

Vértice de Corte.(2017,16) de abril. Wikipedia, La enciclopedia libre. Fecha consulta: 22:41, octubre 28, 2017 from https://es.wikipedia.org/wiki/Vértice_de_corte