UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Análisis de algoritmos y estructura de datos

Laboratorio 2 – Veterinaria

Alberto Rodríguez Z.

Profesor:	Pablo Schwarzenberg Riveros
Ayudantes:	Javiera Torres
	Javiera Sáez
	Diego Opazo

Santiago - Chile

1-2018

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

Uno de los contenidos dentro del curso Algoritmos y Estructura de datos son los grafos, una estructura de datos que consiste en un conjunto de elementos llamados nodos o vértices que son unidos a través de aristas o arcos. Los grafos son una forma de representación de las interacciones de elementos entre sí.

Este laboratorio cuenta con una veterinaria llamada el "Bulto Feliz", que guarda la información contenida en los archivos de texto "Consultorio.in" y "DondeLlevarAlBulto.in", estos contienen la información de los consultorios y las conexiones entre ellos. Se tiene que implementar un programa que permita entregar el camino mínimo de un consultorio a otro cuando se ingresa un paciente que está en un consultorio inicial que necesita la especialidad solicitada, este camino debe ilustrarse en el archivo de salida "WiiuuWiiu.out". Además, el programa debe dar de alta pacientes de algún consultorio y actualizar la información general.

Para lograrlo se utilizarán la estructura anteriormente mencionada, se trabajará con el lenguaje de programación C, específicamente con ANSI C (estándar de lenguaje publicado por el Instituto Nacional Estadounidense de Estándares). El algoritmo trabaja esta información a través de una matriz de adyacencia.

A continuación, se mencionarán los objetivos de este informe junto a la descripción del problema y como se abordó para solucionarlo.

Objetivos

1.1 Generales:

El propósito central de este proyecto es la implementación de grafos y trabajar con ellos en la solución de la veterinaria.

1.2 Específicos:

1. Aplicación de conceptos clave de grafos aprendidos en clase, para poder trabajarlos con los archivos de la veterinaria
2. Estudiar los algoritmos desde variados aspectos, desde la representación a través de los tipos de datos abstractos hasta los tiempos de ejecución de aquellos algoritmos.
3. Elaborar un programa que respete los estándares ANSI C con el fin de crear un programa eficiente, optimo y estable.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

A continuación, se explicarán las herramientas, técnicas y algoritmos empleados para la elaboración de una solución además del material investigado el cual sirvió como base para resolver el problema.

2.1 Marco teórico:

Dentro de los conceptos claves de este informe se encuentran el tipo de dato abstracto llamados grafos. Para entenderlos se desarrollará una breve definición.

Un Tipo de dato abstracto (abreviado como TDA) es un conjunto de datos u elementos al cual se le asocian operaciones. El TDA provee de una interfaz con el cual es posible realizar dichas operaciones, de esta manera es posible acceder a los datos a través de las operaciones provistas por aquella interfaz.

Un grafo como ya antes mencionado consiste en un conjunto de nodos o vértices y un conjunto de arcos o aristas que establecen relaciones entre los nodos.

Los grafos pueden ser dirigidos (que solo tienen un sentido las aristas) o no dirigidos (que las aristas tienen ambos sentidos), estos grafos por lo general se pueden representar como una matriz de adyacencia o como una lista de adyacencia. Para poder representar estos grafos en el lenguaje C se crean estructuras (explicadas anteriormente en el laboratorio 1) que representaran la información de un nodo y las conexiones de estos.

A través de estos grafos nos permitirá desarrollan las conexiones entre los consultorios debido a que son óptimos para realizar búsquedas otorgando tiempo de ejecución mínimos (O(logn) y O(N) en un peor caso).

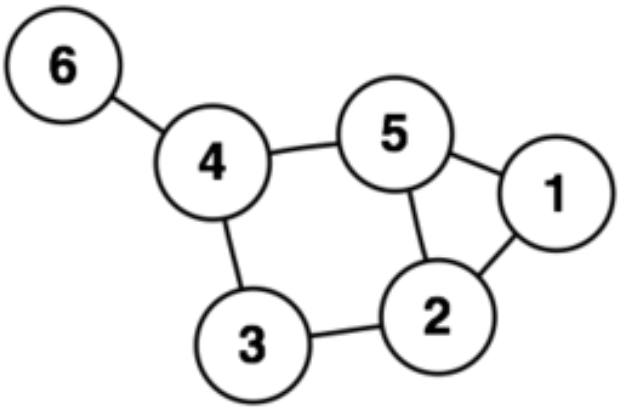


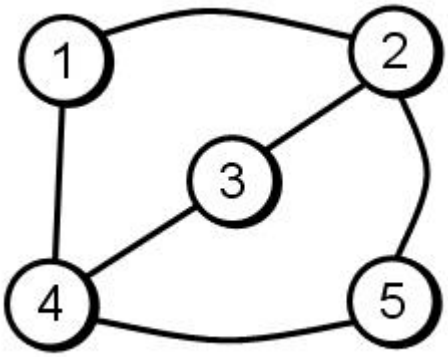
Fig 2.1 Ejemplo de representación de un grafo.

2.2 Herramientas y Técnicas:

La matriz de adyacencia es una forma de representación del grafo en ella las filas y las columnas representaran los nodos de este y las posiciones el tiempo que hay desde un nodo a otro. De esta forma esta información se nos realiza más fácil poder obtener el valor mínimo de un nodo ya que solo se analizan las posiciones de la matriz.

Una técnica fue trabajar con una lista a parte del grafo donde se almacenan todos los nodos que este contiene, con esta se modifican más fáciles los nodos de manera aparte sin tener que modificar la matriz. La matriz solo es usada para obtener los valores de tiempo de las aristas.

Como se utiliza el TDA de grafos, se crea funciones que trabajan como sus operaciones correspondientes, están son encolar (agregar nodo), mínimo (se obtiene el nodo de distancia mínima) y extraerNodo (se excluye nodo de la lista). Estas funciones son utilizadas para obtener los valores mínimos de un nodo a los demás (Dijkstra).



M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Fig 2.2.1 Ejemplo de grafo implementado como matriz de adyacencia.

Otra técnica importante fue como trabajar con el tipo de dato string dinámico. En el lenguaje C trabajar con string dinámicos tiene sus ciertas funciones que operan sobre los strings, algunas de las funciones utilizadas en el programa fueron:

-char *strcpy(char *cad1,char *cad2): copia el contenido de cad2 en cad, devolviendo cad1

-char *strcat(char *cad1,char *cad2): añade el string cad2 en cad1 y devuelve cad1 con cad2 unido a él.

-char * strcmp(char *cad1,char *cad2): compara dos strings, de ser iguales devolverá un 0

2.3 Algoritmos y estructura de datos

Estructuras:

Se crean 3 estructuras, en la primera se contiene toda la información de un nodo en particular, en la segunda estructura es una lista que contiene el orden de los nodos.

La tercera estructura será la representación del grafo, gracias a los punteros cada de nodo de esta podría trabajarse como un arreglo donde se guardará la información de cada nodo en cada posición. Además se trabaja con un tipo de dato "doublé" que es un doble puntero, este representa la matriz de adyacencia que contiene los tiempos entre las aristas de nodos del grafo.

```
typedef struct nodo{
    char nombreConsultorio[30];
    char especialidad[30];
    int capacidad;
    int pacientesActuales;
    int id;
    int distancia;
    int nodoDestino;
    struct nodo* siguiente;
}nodo;

typedef struct lista{
    nodo* inicio;
    nodo* final;
}lista;

typedef struct grafo{
    nodo *vertice;
    int nodos;
    double** adj;
}grafo;
```

Fig 2.3.1 Representación de las tres estructuras.

Algoritmos:

- *void leerTextos( ):*

Función que lee los dos archivos "Consultorios.in" y "DondeLlevarAlBulto.in" ,la información de los consultorios los guarda en el "nodo *vértice" de la estructura grafo, mientras los tiempo entre los nodos los guarda en la matriz de adyacencia según cada posición.

Tiempo de ejecución: T(n) = n ^2 + 4n + c
Orden de complejidad: O(n^2).

- *void Dijkstra(grafo *G):*

Función que solicita ingresar el nodo inicial y la especialidad que se busca, con esta información y con la información del grafo, calcula el tiempo mínimo que hay desde ese nodo inicial a todos los nodos del grafo. Después busca cual nodo cumple con la especialidad y con los cupos disponibles, si existen dos o mas nodos que cumplen las condiciones seleccionara el que tenga el menos tiempo en llegar a el. Para finalizar creando el archivo "WiiuuWiiu.out" donde se registra el recorrido del nodo inicial al final.

Tiempo de ejecución: T(n) = n ^2 + 8(2log(n)) + nc + c
Orden de complejidad: O(n^2).

- *void darAlta(grafo *G):*

Función que al ingresar un consultorio da de alta un paciente de este, es decir disminuye en 1 el tipo de dato que contiene la cantidad de pacientes actuales.

Tiempo de ejecución: T(n) = n ^2 + 2(n+c) + c
Orden de complejidad: O(n^2).

- *void guardar(grafo *G):*

Función que actualiza los datos y imprime por pantalla los consultorios con sus datos actualizados.

Tiempo de ejecución: T(n) = nc + c
Orden de complejidad: O(n).

- *nodo* minimo(lista *L):*

Selecciona un nodo, los selecciona según el valor que tenga de distancia, elige el nodo de menor distancia.

Tiempo de ejecución: T(n) = nc + c
Orden de complejidad: O(n).

- *void extraerNodo(lista *L,nodo *N):*

Elimina el nodo seleccionado de la lista, con esto la lista se irá reduciendo y se hace más fácil analizar las distancias de los nodos en la función Dijkstra.

Tiempo de ejecución: T(n) = nc + c
Orden de complejidad: O(n).

2.4 ANÁLISIS DE LOS RESULTADOS

Al trabajar en el lenguaje de programación C, una de sus características es el manejo de memoria a través de punteros, la cual para objetivos de este laboratorio nos permite realizar un problema que aparte de resolver el problema, sea mas optimo en cuanto a tiempos de ejecución, para eso se analizó a través de formulas que permiten calcular cuanto se demora cada ciclo. Con estas formulas podemos determinar la eficiencia de nuestro programa.

Una de las funciones mas importantes del programa es la que leer ambos textos “leerTextos”, ya que debe leer guardar los datos de ambos archivos en las estructuras para después trabajar con ellas. Probando con archivos se consigue tiempo de prueba efectivo, realiza la lectura de forma rápida dando un orden de complejidad óptimo de O(n^2).

Otra de las funciones importantes es la que busca el camino mínimo en un grafo (Dijkstra), que tiene que calcular el camino mínimo desde un nodo a todos los nodos del grafo, para eso llama a otras funciones las veces que sea necesaria logrando ser eficiente ya que el tiempo que demora es mínimo con un orden de complejidad O(n^2).

Estas al ser las funciones mas importantes y conociendo su orden de complejidad, al realizar las pruebas con distintos archivos, se logro obtener que el tiempo de ejecución es efectivo como se tenia previsto, estas pruebas nos dan la información de que el programa es eficiente y el tiempo de demora es mínimo.

Retomando los objetivos, podemos ver un logro satisfactorio, implementando correctamente un programa que aborda el enunciado que se pide, a excepción de la función de ingresar paciente, ya que el programa si calcula el camino mínimo desde un nodo a otro, pero no logra imprimir el trayecto de línea por línea en el archivo, solo imprime el camino del nodo inicial al final con el tiempo de ese trayecto.

Dentro de los objetivos específicos podemos ver como logramos estudiar los conceptos claves de grafos y desarrollando un programa que cumpla este tipo de dato correctamente, logrando cumplir la gran mayoría del programa inicial. También este programa cumple nuestro objetivo de respetar los estándares de ANSI C compilando en entornos de Windows y Linux. Este programa cumple ser optimo y estable para recibir al usuario y permitirle el uso de las herramientas creadas.

CAPÍTULO 3. CONCLUSIÓN

Como se mencionó anteriormente en el análisis, el laboratorio tuvo un logro bastante alto ya que se logró estudiar en profundidad y con el tiempo TDA de un grafo (dirigido o no), logrando plantear algoritmos necesarios para su operación y de manera exitosa, sin errores o fallas, logrando resolver la mayoría de los problemas que eran planteados en el enunciado.

Gracias a este presente laboratorio fue posible verificar en la praxis como la implementación de una estructura como grafos en conjunto con los algoritmos de búsqueda sirve para realizar programas eficientes y óptimos, recalcando que un orden de complejidad O(n^2) es uno de los resultados talvez no más óptimos al ser n-exponencial, pero si eficiente sabiendo que el grafo se implementó con matrices y listas.

A futuro se espera poder realizar un manual de usuario mas completo ya que si bien se aborda todo lo especificado, siempre se podrían detallarse mejor las instrucciones de compilado y uso del programa juntos a sus limitaciones.

REFERENCIAS

PassItEDU (2014). *Matemáticas Discretas - Algoritmo de Dijkstra*. (España) Recuperado de: https://www.youtube.com/watch?v=Ffslqr-hs_I.

Jacqueline Köhler Casasempere (2017). *Apuntes de la asignatura Análisis de Algoritmos y Estructuras de Datos*. Chile, Departamento de ingeniería informática (DINF), Usach.