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

Estructura de Datos y Análisis de Algoritmos

Laboratorio N°3

Alumno: Israel Arias Panez.

Profesor: Mario Inostroza.
Ayudante: Esteban Silva.

Santiago - Chile
2-2020

TABLA DE CONTENIDOS

Índice de Figuras

CAPÍTULO 1. INTRODUCCIÓN

En este tercer laboratorio de la asignatura Análisis de algoritmos y estructura de datos se presenta el enunciado “Salvación Espacial II”, en el cual se presenta la continuación de la situación presentada en el enunciado del primer laboratorio de la asignatura: Luego de salir del planeta Tierra junto a un grupo de personas con la misión de salvar la humanidad, para ello la doctora Ximi le encarga a su piloto estrella, el capitán Yoda la conducción de la nave, entonces el capitán Yoda debe de averiguar la mejor ruta para llegar al planeta Pizza Planet, en un inicio el capitán pensaba ir en línea recta, pero el copiloto Bender le sugiere que reconsidere su idea, debido a que existe la posibilidad de quedar varados por falta de combustible, para ello Bender sugiere el buscar la mejor ruta, recorriendo planeta por planeta y tomando en cuenta que por ir a un planeta se gasta cierta cantidad de tiempo y de combustible y que hay planetas en los cuales es posible recargar el combustible de la nave en su totalidad. Bajo ese contexto el enunciado nos solicita la tarea de encontrar el camino que tome la menor cantidad de tiempo en llegar desde la Tierra hasta Pizza Planet, teniendo en cuenta que la nave no puede quedar varada, para ello se nos entrega un “mapa”, el cual contiene todos los distintos planetas, mostrando cuanto tiempo y combustible se gasta de ir de un planeta a otro, además de mostrar que planetas tienen una arista o camino con otro planeta y mostrando también en que planetas es posible efectuar una recarga de combustible. El mapa será entregado a través de un archivo de texto y en otro archivo de texto se entregará la cantidad de combustible con el cual sale la nave de la Tierra y el nombre de los planetas que tienen gasolineras.

Para lograr generar la solución pedida en este enunciado de laboratorio, se aplicarán técnicas, diseños y modelos con el conocimiento proveniente de la ciencia de la computación y de los algoritmos y estructura de datos, los cuales son conocimientos adquiridos en las clases de la asignatura, en la solución se ocuparán algoritmos de búsqueda de caminos mínimos en grafos, estructuras de datos, uso de punteros, direcciones de memoria, alocación de memoria, etc. También se hará el análisis del algoritmo de la solución implementada, en base al tiempo y su complejidad. En esta introducción del informe se ha contextualizado el problema a resolver en este laboratorio, en la próxima sección se hará el análisis del problema, haciendo énfasis en la metodología que se siguió para resolverlo, estructuras de datos usadas, entre otros. Luego en la sección de Algoritmos y estructura de datos se presentarán los algoritmos usados para el desarrollo de la solución, detallando su funcionamiento y tiempo T(n) y O( ). A continuación, en la sección de Análisis de resultados, se describirá los resultados obtenidos de la solución implementada, se analizarán también las falencias detectadas y propuestas de cómo mejorar estas mismas y otros ámbitos de la solución. Finalmente, en la sección de Conclusión se indicará el grado de logro de los objetivos iniciales y una conclusión respecto al trabajo en este laboratorio.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

2.1.1 Marco teórico

Para el mejor entendimiento de la solución que será planteada y del informe en general, conviene definir algunos conceptos que serán nombrados.

- Algoritmo: Conjunto de instrucciones o reglas definidas no ambiguas que permite solucionar un problema u otras tareas.
- Metodología: Conjunto de procedimientos racionales para alcanzar un objetivo.
- Archivo: Conjunto de bytes que son almacenados en un dispositivo, por ejemplo, un archivo de texto plano.
- Función: Pequeña parte de un programa que realiza una tarea en particular, recibiendo una entrada, procesándola y devolviendo una salida.
- Grafo: Es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.
- Dijkstra: s un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista.
- Struct: Declaración de un tipo de dato compuesto que permite almacenar conjuntos de datos en un bloque de memoria simulando una “estructura”.
- Tiempo de ejecución: Es el intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo.
- Complejidad: Es una descripción de la cantidad de tiempo que lleva ejecutar un algoritmo.
- Estructura de datos: Es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente.
- CPU: Es el hardware dentro de un ordenador u otros dispositivos programables, que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema.

2.1.2 Metodología de solución y herramientas utilizadas.

La metodología por seguir para la resolución de este laboratorio es el uso de una programación modular en el lenguaje de programación C o también llamada como "divide y vencerás", la cual consiste en dividir el problema en subproblemas a fin de ir solucionándolos poco a poco. "No existe proyecto que no pueda ser acometido o alcanzado. Ahora bien, es importante poder dimensionarlo de forma adecuada, dividiéndolo en subtareas de tamaño más pequeño que sean accesibles y practicables. Así, si un gran proyecto lo dividimos en varios hitos más pequeños, conseguiremos realizar pequeños-grandes progresos que, paso a paso nos permitirán alcanzar el fin último que pretendemos."[1]

Como datos de entrada se nos han entregado dos archivos:

- "mapa.in", el cual contiene la cantidad de planetas, además de que planeta esta "conectado" con cual y su coste en tiempo/combustible para ir de un planeta a otro.
- "combustible.in", el cual contiene la cantidad de combustible máxima y con la cual inicia su recorrido desde la tierra, además de tener el nombre de los planetas que permiten recargar combustible.

El programa seguirá la siguiente metodología de solución:

1) Leer el archivo de entrada "mapa.in".
2) Recolectar la información (aristas) del archivo, gracias a una struct construida para ello.
3) Crear Id's para cada nodo encontrado en la lectura del archivo mapa.in
4) Crear una matriz de adyacencia.
5) Añadir los datos recolectados de mapa.in a la matriz de adyacencia.
6) Leer el archivo de entrada "combustible.in".
7) Recolectar la información del archivo, consiguiendo la cantidad de combustible de la nave y el nombre de todos los planetas que tienen gasolinera.
8) Agregar la información de que planetas tienen gasolineras en la matriz de adyacencia.
9) Conseguir el camino más corto desde la tierra a Pizza Planet, tomando en consideración todos los datos recolectados, para ello se ejecutarán múltiples algoritmos.
10) Crear el archivo de salida "ruta.out", escribiendo el camino más corto junto con el tiempo empleado para ello.

Para la lectura de archivos se usara la función fopen() con el modo de lectura.

Para trabajar de una manera más simple se crearon cuatro structs:

- Struct conexion: Creada para almacenar los parámetros de una arista o conexión: cantidad de gasolina, tiempo, id del nodo de inicio e id del nodo destino, nombre del nodo de inicio y nombre del nodo final.

- Struct nodo: Creada para representar la matriz de adyacencia, contiene cuatro parámetros: gasolina a usar, tiempo a usar, entero que dice si es una gasolinera o no, además del nombre del nodo o planeta.
- Struct recorrido: Usada para almacenar los resultados de ejecutar el algoritmo de Dijkstra, contiene cuatro arreglos que almacenan la distancia/tiempo mas corto a cualquier nodo desde el origen, arreglo padre que contiene desde donde se salto a un nodo para a futuro formar el camino más corto, un arreglo de booleanos que lleva cuenta de que nodos fueron procesados y por ultimo un arreglo que lleva la cuenta de la cantidad de gasolina que se ha ocupado en el recorrido hasta cierto nodo
- Struct Camino: Usada para almacenar los caminos más cortos encontrados con Dijkstra, contiene la cantidad de nodos por la cual se efectúa el recorrido, un arreglo que contiene el camino más corto nodo por nodo y por ultimo la distancia o tiempo usado en recorrer ese camino.

Para la recolección de datos para los puntos 2) y 6) se hará uso de la posibilidad de definir structs en C, para agrupar distintos tipos de datos en una sola variable, junto al uso de la estructura de dato “arreglo” se almacenaran las variables de struct creadas en un arreglo luego usando un algoritmo iterativo se recorrerá el archivo leído de principio a fin, almacenando todos los datos dentro de la struct, la cual, luego la instancia de esa struct se almacenara dentro del arreglo dispuesto, así hasta llegar al fin del documento, los valores se conseguirán con el uso de la función fscanf().

Para la recolección del grafo se implementó una matriz de adyacencia como se puede apreciar en el punto 3), la cual es un arreglo de dos dimensiones.

Para el punto 9) se hará uso de algoritmos iterativos para la escritura del archivo, además se debe velar respetar las estructuras del formato de escritura solicitados, el cual es son visibles en la figura 1.

```
39
tierra->luna->axiom->rigel_7->axiom->pizza_planet
```

Figura 1: Ejemplo formato de archivo de salida “ruta.out”.

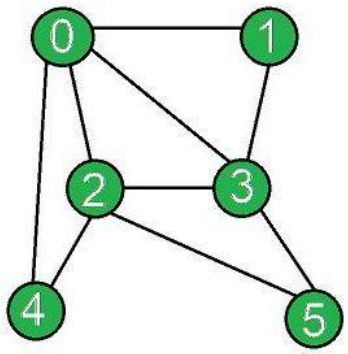
2.1.3 Algoritmos y estructuras de datos

Antes de describir los algoritmos usados para la metodología presentada en el punto 2.1.2, se presentarán las estructuras de datos que se utilizaran en la creación de esta solución, sin embargo, cabe preguntarse en un inicio, ¿por qué las estructuras de datos ayudan a facilitar la implementación de soluciones?

"Una "estructura de datos" es una **colección de valores**, la relación que existe entre estos valores y las operaciones que podemos hacer sobre ellos; en pocas palabras se refiere a cómo los datos están organizados y cómo se pueden administrar. Una estructura de datos describe el **formato** en que los valores van a ser almacenados, cómo van a ser accedidos y modificados, pudiendo así existir una gran cantidad de estructuras de datos. Las estructuras de datos son útiles porque siempre manipularemos datos, y si los datos están organizados, esta tarea será mucho más fácil." [1]

Entonces al tener los datos organizados en estructura de datos, la implementación de la solución se facilitará, entonces a continuación se presentarán las estructuras de datos utilizadas a fin de facilitar la creación de la solución:

La principal estructura de datos con la que se trabajará en esta experiencia de laboratorio corresponde a un "grafo", el cual puede ser interpretado a través de listas de adyacencia o matriz de adyacencia, para esta ocasión se ha elegido el trabajar con matriz de adyacencia, la cual se representa en código como un arreglo de dos dimensiones:



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

Figura 2: Representación de grafo como matriz de adyacencia

Como es posible observar en la figura 2, una matriz de adyacencia tiene la característica que permite saber que nodo se encuentra conectado con que otro nodo, lo cual se ve

representado con un 1 cuando existe una conexión y un 0 cuando no existe, por ejemplo, en el grafo de la figura 2, el nodo 0 tiene una conexión con los nodos 1, 2, 3 y 4, debido a que tiene un 1 en la posición [0][1], [0][2], [0][3] y [0][4].

Para la implementación realizada en esta experiencia de laboratorio, se modificó la implementación clásica de un grafo por matriz de adyacencia presentada en la figura 2, la matriz adyacencia usada en esta implementación posee las siguientes características:

La implementación de la enlazada, junto a sus modificaciones para adecuarse a lo solicitado:

	0	1	2	3	4	5
0	nodo	nodo	nodo	nodo	nodo	nodo
1	nodo	nodo	nodo	nodo	nodo	nodo
2	nodo	nodo	nodo	nodo	nodo	nodo
3	nodo	nodo	nodo	nodo	nodo	nodo
4	nodo	nodo	nodo	nodo	nodo	nodo
5	nodo	nodo	nodo	nodo	nodo	nodo

Figura 3: Representación modificada de matriz de adyacencia

Como es posible observar en la figura 3, la matriz de adyacencia implementada no se encuentra con enteros en su interior, si no con “nodos”, por lo que es relevante saber que incluye cada nodo:

```
typedef struct nodo nodo;
struct nodo{
    int esGasolinera, gasolina, tiempo;
    char nombre[100];
};
```

Figura 4: struct nodo

Como es posible apreciar en la figura 4, cada nodo incluye la información si ese nodo es una gasolinera, cuanta gasolina se gasta para llegar a ese nodo, cuanto tiempo se gasta para llegar a ese nodo y el nombre del planeta que se encuentra en ese nodo. Tomando en consideración el mapa otorgado para esta experiencia de laboratorio:

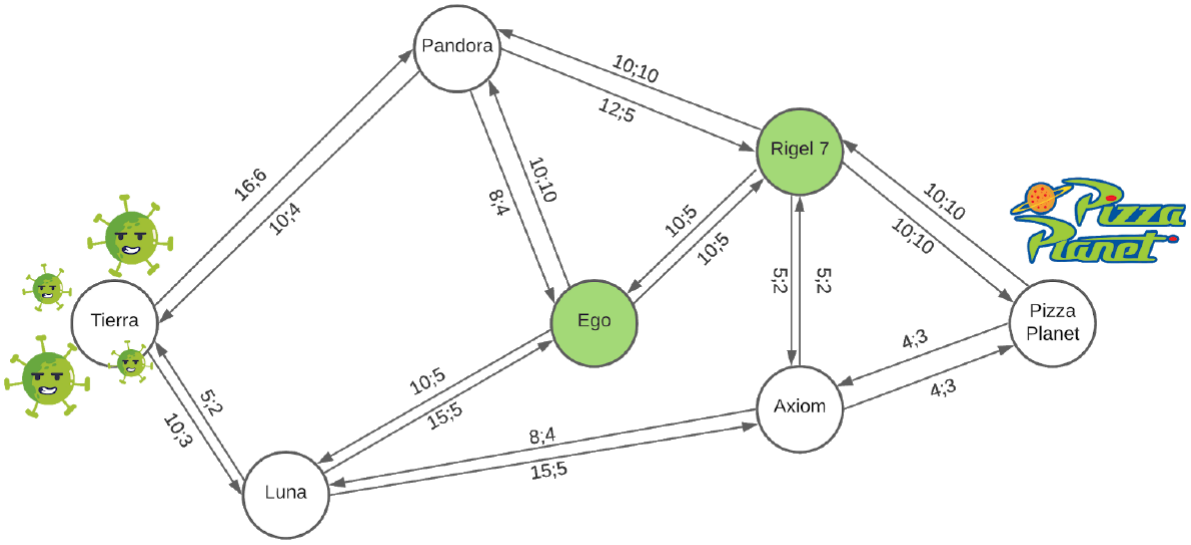


Figura 5: grafo del problema.

La representación en matriz de adyacencia es la siguiente:

```
Matriz:
() -1/-1 gasolinera?:0 | (pandora) 16/6 gasolinera?:0 | (luna) 10/3 gasolinera?:0 | () -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 |
(tierra) 10/4 gasolinera?:0 | () -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 | (ego) 8/4 gasolinera?:1 | (rigel_7) 12/5 gasolinera?:1 | () -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 |
(tierra) 5/2 gasolinera?:0 | () -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 | (ego) 15/5 gasolinera?:1 | () -1/-1 gasolinera?:0 | (axiom) 15/5 gasolinera?:0 | () -1/-1 gasolinera?:0 |
() -1/-1 gasolinera?:0 | (pandora) 10/10 gasolinera?:0 | (luna) 10/5 gasolinera?:0 | () -1/-1 gasolinera?:0 | (rigel_7) 10/5 gasolinera?:1 | () -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 |
() -1/-1 gasolinera?:0 | (pandora) 10/10 gasolinera?:0 | () -1/-1 gasolinera?:0 | (ego) 10/5 gasolinera?:1 | () -1/-1 gasolinera?:0 | (axiom) 5/2 gasolinera?:0 | (pizza_planet) 10/10 gasolinera?:0 |
() -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 | (luna) 8/4 gasolinera?:0 | () -1/-1 gasolinera?:0 | (rigel_7) 5/2 gasolinera?:1 | () -1/-1 gasolinera?:0 | (pizza_planet) 4/3 gasolinera?:0 |
() -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 | () -1/-1 gasolinera?:0 | (rigel_7) 10/10 gasolinera?:1 | (axiom) 4/3 gasolinera?:0 | () -1/-1 gasolinera?:0 |
```

Figura 6: Representación de la matriz de adyacencia del grafo entregado

	0	1	2	3	4	5	6
0	0	(pandora) 16/6 gasolinera?:0	(luna) 10/3 gasolinera?:0	0	0	0	0
1	(tierra) 10/4 gasolinera?:0	0	0	(ego) 8/4 gasolinera?:1	0	(axiom) 15/5 gasolinera?:0	0
2	(tierra) 5/2 gasolinera?:0	0	0	(ego) 15/5 gasolinera?:1	0	(axiom) 15/5 gasolinera?:0	0
3	0	(pandora) 10/10 gasolinera?:0	(luna) 10/5 gasolinera?:0	0	(rigel_7) 10/5 gasolinera?:1	0	0
4	0	(pandora) 10/10 gasolinera?:0	0	(ego) 10/5 gasolinera?:1	0	(axiom) 5/2 gasolinera?:0	(pizza_planet) 10/10 gasolinera?:0
5	0	nodo	(luna) 8/4 gasolinera?:0	0	(rigel_7) 5/2 gasolinera?:1	0	(pizza_planet) 4/3 gasolinera?:0
6	0	0	0	0	(rigel_7) 10/10 gasolinera?:1	(axiom) 4/3 gasolinera?:0	0

Figura 7: Representación dibujada de matriz de adyacencia grafo entregado.

La figura 7 ilustra exactamente lo mismo que la figura 6, pero de una manera más comprensible a entender, debido a que la figura 6 es lo que se muestra por pantalla cuando se pide mostrar la matriz en el programa implementado para esta experiencia de laboratorio. Es posible apreciar la representación del grafo de la figura 5 como la matriz de adyacencia modificada presentada en la figura 7, la cual contiene toda la información necesaria para poder trabajar en ella.

A continuación, se presentará la manera que se obtuvo el camino más corto, refiriéndose al punto 9. El camino más corto fue conseguido principalmente gracias al algoritmo de Dijkstra “El Algortimo de Dijkstra, también denominado Algoritmo de caminos mínimos, es un modelo que se clasifica dentro de los algoritmos de búsqueda. Su objetivo, es determinar la ruta más corta, desde el nodo origen, hasta cualquier nodo de la red.” [3]. Como lo dice la cita, el algoritmo de Dijkstra permite conseguir la ruta más corta desde un nodo de origen que puede ser cualquier nodo de un grafo, a elección, hasta cualquier otro nodo de la red del grafo. En esta ocasión se implementó el algoritmo de Dijkstra con una modificación para aparte de llevar la cuenta de la menor distancia y camino hasta cualquier nodo, también llevar la cuenta de la cantidad de combustible usado para llegar a cualquier otro nodo.

Una vez implementado Dijkstra, se implemento la función principal, la cual funciona de la siguiente manera: Tomando como ejemplo el grafo entregado mostrado en la Figura 5:

En primer lugar, se ejecutará Dijkstra desde la tierra, lo que dará la cantidad de caminos más corta hacia cualquier otro nodo, por ejemplo, el camino más corto de la tierra hacia Pizza Planet:

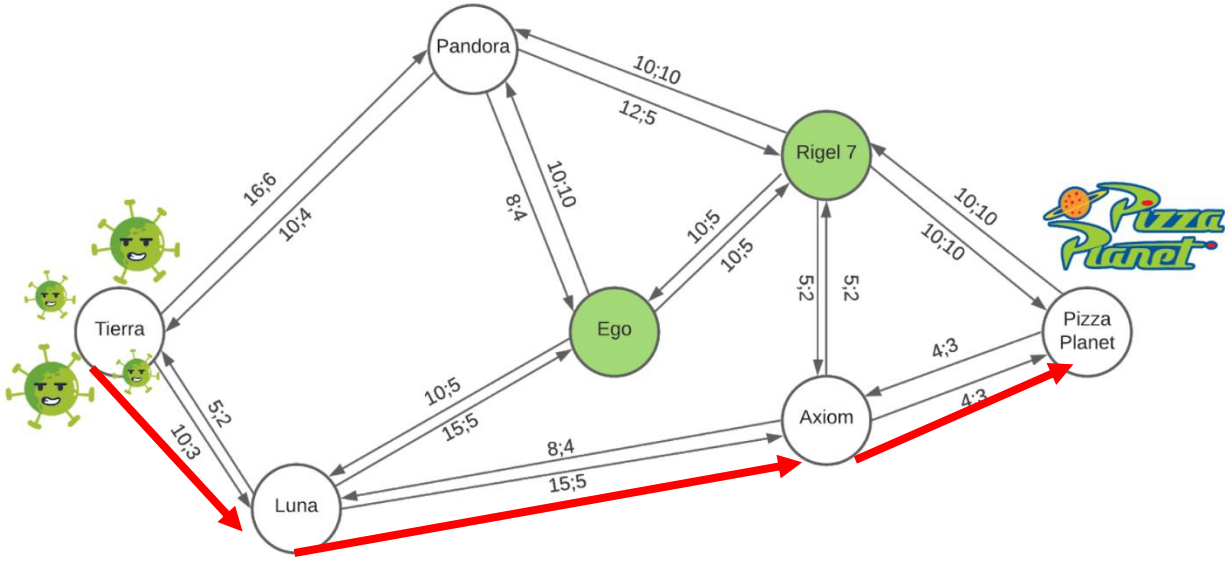


Figura 8: Camino más corto desde la tierra a Pizza Planet.

Como es posible observar en la figura 8, al ejecutar Dijkstra una vez el camino más corto obtenido desde la tierra hasta Pizza Planet es: Tierra -> Luna -> Axiom -> Pizza Planet.

Luego se verificará si es posible efectuar ese recorrido tomando en cuenta la cantidad de combustible que se gasta hasta llegar a Pizza Planet, en este caso el recorrido es imposible de recorrer, debido a que se necesitan 11 de combustible para efectuar el recorrido y se cuentan con 10 de combustible inicial, en el caso que el recorrido si fuese posible de efectuarse, el algoritmo retornaría ese recorrido encontrado y terminaría su ejecución.

En caso de que no haya sido posible recorrer la distancia más corta, se volverá a efectuar Dijkstra, pero en todas las gasolineras existentes en el grafo entregado, de tal manera de conseguir los caminos más cortos desde una bencinera a cualquier otro punto del grafo. Para ello se ejecutará un algoritmo iterativo que almacenara los distintos resultados de ejecutar Dijkstra en los distintos nodos en un arreglo que almacena structs “camino”, struct que fue descrita en el punto 2.1.2 del presente informe, luego de la ejecución de este algoritmo se tiene una cantidad n de nodos de los cuales se tiene la información que entrega el algoritmo de Dijkstra:

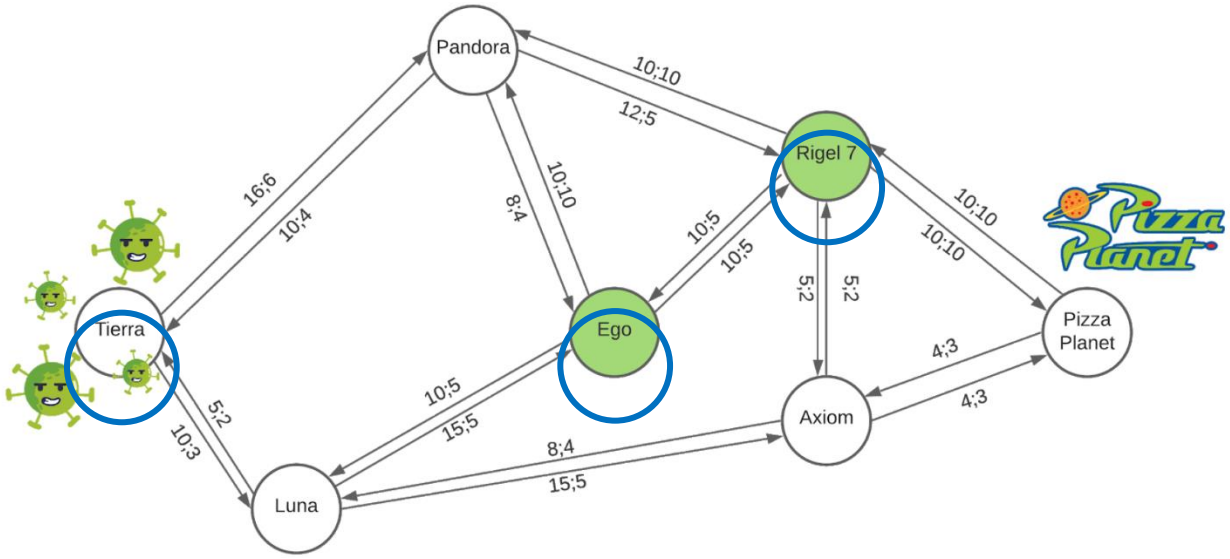


Figura 9: Nodos a los cuales se les ha ejecutado Dijkstra.

Entonces, ya que no fue posible el recorrer directamente desde la Tierra hasta Pizza Planet, tomando el camino que menos tiempo tomaba, significa que, si o si hay que pasar por una gasolinera para abastecerse de combustible, por ello se corrió Dijkstra en los nodos que son gasolineras, para tener la información de los caminos más cortos desde ellas a cualquier otro lado.

El siguiente paso es encontrar el camino más corto válido desde la tierra hacia cada bencinera y verificar que ese camino sea efectivamente valido tomando en cuenta el combustible, en caso de no ser válido, se “elimina” ese camino y se vuelve a ejecutar Dijkstra hasta encontrar un camino valido:

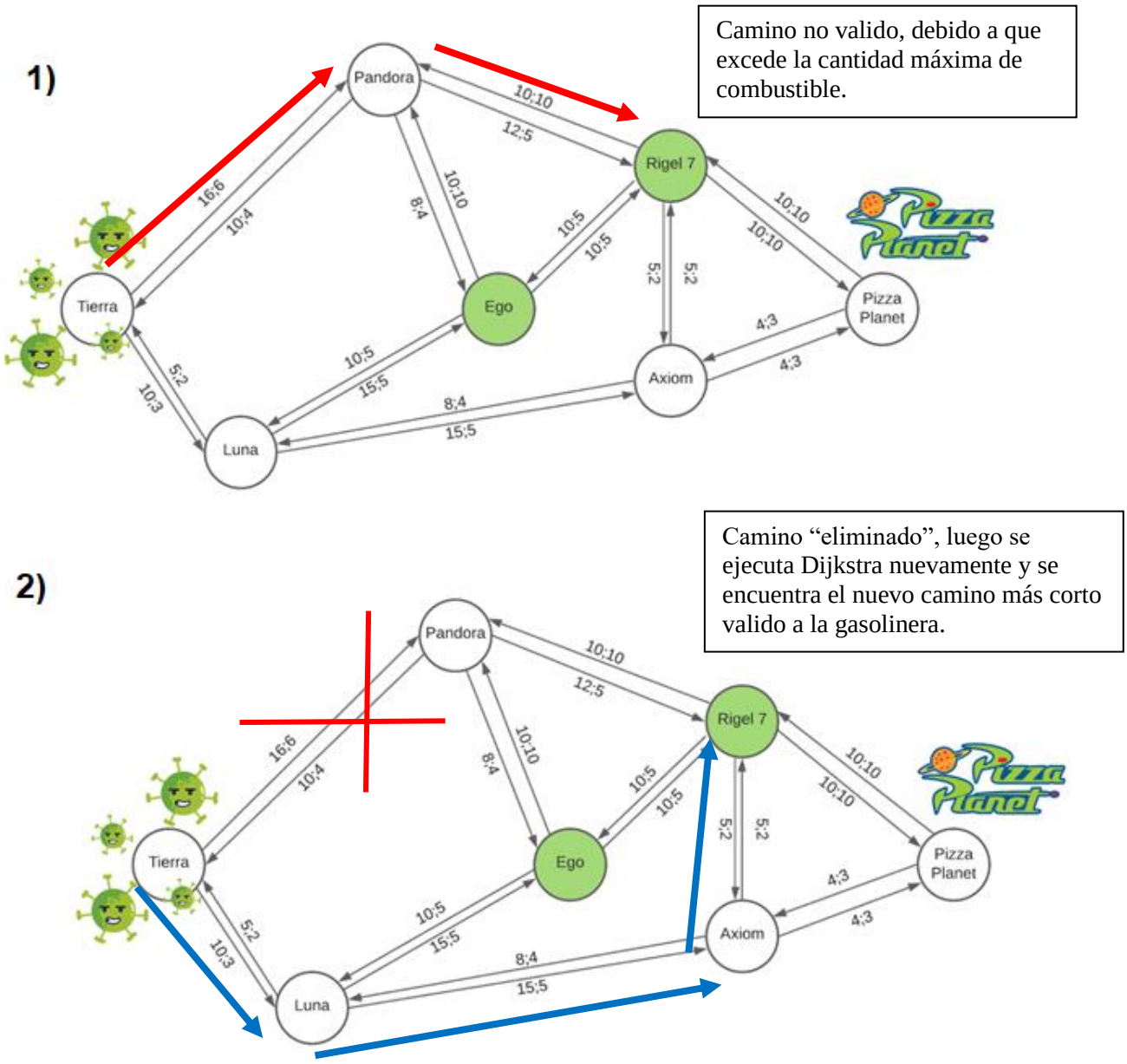


Figura 10: Algoritmo para encontrar el camino más corto valido a cada gasolinera

Los dos pasos antes ilustrados en la Figura 10 se repetirán múltiples veces hasta que se encuentre el camino más corto y que sea válido en términos de combustible, eso para todas las gasolineras disponibles. Una vez todos los caminos disponibles, se hará mediante una iteración la búsqueda de la combinación más corta entre el recorrido Tierra -> gasolinera -> Pizza Planet, en el caso del grafo del problema se deberá verificar que es más corto en términos de tiempo, si ir desde la tierra hasta ego y luego de ego hasta Pizza Planet o ir desde la tierra hasta Rigel 7 y luego de Rigel 7 hasta Pizza Planet, para ello se ejecuta un algoritmo iterativo que buscara el menor y almacenara que combinación es la menor. Cabe

destacar que en este punto ya no existen los caminos no validos o que sean imposibles de recorrer. Entonces el algoritmo selecciona el recorrido que tiene menos tiempo y luego finalmente se “arma” el recorrido final combinando el recorrido más corto desde la tierra hasta la gasolinera seleccionada y luego el recorrido desde la gasolinera hasta Pizza Planet.

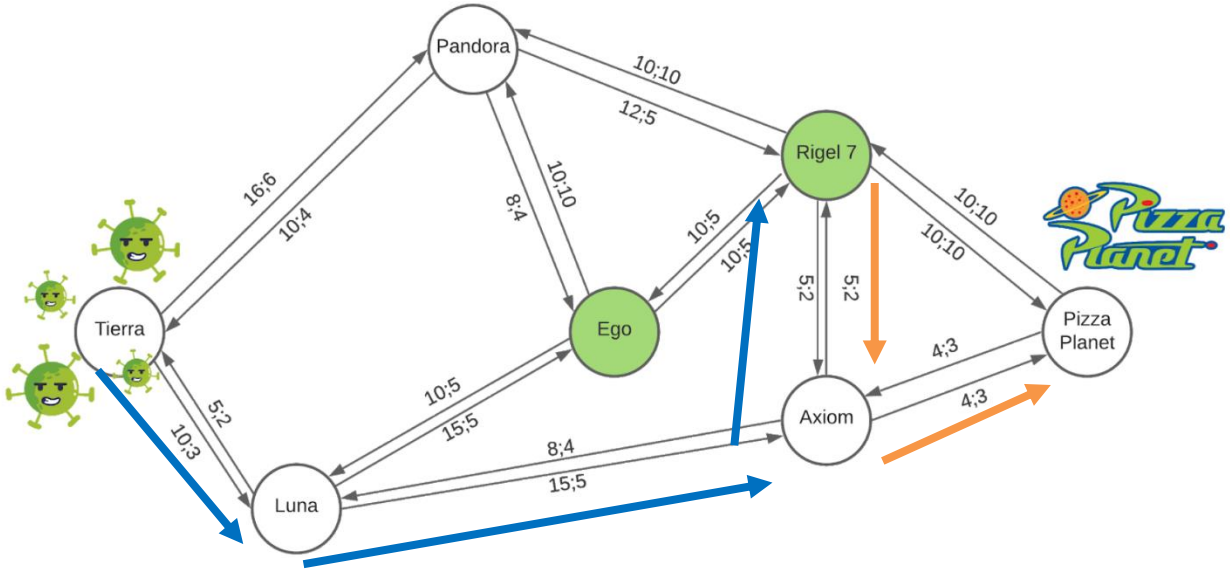


Figura 11: Recorrido final.

En la figura 11 las flechas en azul representan el recorrido más corto desde la tierra hasta el planeta seleccionado por el algoritmo, las flechas en naranjo representan el camino más corto desde la gasolinera hasta Pizza Planet, el juntar ambos caminos otorga el recorrido final que se busca, el cual es retornado por el algoritmo.

Se prefirió el ejemplificar y explicar el algoritmo creado para la solución mediante el mismo grafo de ejemplo entregado, debido a la complejidad del mismo algoritmo. El algoritmo recién explicado implementado en código es el siguiente:

```
//Entrada: Entero con el numero de nodos, entero con el combustible inicial, entero con la cantidad de gasolineras, la matriz de adyacencia,
//Salida: Arreglo que contiene el camino nodo por nodo mas corto desde la tierra hasta pizza planet
//Objetivo: Conseguir el camino mas corto desde la tierra hasta pizza planet, tomando en consideracion el gasto de combustible y las gasoline
int * conseguirCaminoMasCorto(int numeroNodos, int combustibleInicial, int * cantidadGasolineras, nodo matriz[numeroNodos][numeroNodos], int
    //Se debe conseguir el camino mas corto desde la tierra hasta pizza planeta, entonces
    //En primer lugar se conseguira el id de la tierra y el id de pizza planeta leyendo la matriz de adyacencia
    //Tambien se leeran los id de las gasolineras y se almacenaran en un arreglo
    int idTierra, idPizzaPlanet;
    int numeroGasolinerasEncontradas = 0;
    int * idBencineras = (int*)malloc(sizeof(int)*numeroNodos);
    for(int i = 0; i < numeroNodos; i++){
        for(int j = 0; j < numeroNodos; j++){
            if(strcmp(matriz[i][j].nombre,"tierra") == 0){
                idTierra = j;
            }
            if(strcmp(matriz[i][j].nombre,"pizza_planet") == 0){
                idPizzaPlanet = j;
            }
            if(matriz[i][j].esGasolinera == 1){
                int seEncuentra = 0;
                for(int k = 0; k < numeroGasolinerasEncontradas; k++){
                    if(idBencineras[k] == j){
                        seEncuentra = 1;
                    }
                }
                if(seEncuentra == 0){
                    idBencineras[numeroGasolinerasEncontradas] = j;
                    numeroGasolinerasEncontradas = numeroGasolinerasEncontradas + 1;
                }
            }
        }
    }
```

```
//Se correra el algoritmo de dijkstra para conseguir el camino mas corto midiendo el tiempo desde la tierra a todas las otras paradas
//t inicial
matriz[1][idBencineras[1]].tiempo = 50;
recorrido camino = dijkstra(numeroNodos,matriz,idTierra);
int numeroNodosCamino = 0;
//De todos los caminos conseguidos de correr dijkstra sobre la tierra se conseguira el recorrido a seguir desde la tierra a pizza planet
int * recorridoMasCorto = guardarCamino(idTierra,idPizzaPlanet,camino.padre,numeroNodos,&numeroNodosCamino);

//En primer lugar se verificara si es posible llegar de la tierra a pizza planet directamente
//Se consulta si el camino mas corto desde la tierra hasta pizza planet gasta menos combustible que el disponible inicialmente
if(camino.gasolina[idPizzaPlanet] <= combustibleInicial){
    //Si efectivamente gasta menos combustible que el combustible inicial
    //Se retorna el recorrido mas corto encontrado inicialmente con dijkstra, debido a que alcanza el combustible
    return recorridoMasCorto;
}
Camino * arregloCaminos = (Camino*)malloc(sizeof(Camino)*numeroGasolinerasEncontradas);
int k = 0;
//En caso de que no haya alcanzado el combustible, sera necesario el conseguir la distancia de la tierra hacia todas las bencineras disponibles
//y de las bencineras hasta pizza planet, para luego seleccionar la suma de recorridos que tenga una menor cantidad de tiempo
//Para ello se ejecutara dijkstra para todas las gasolineras que se busco inicialmente
for(int i = 0; i < numeroGasolinerasEncontradas; i++){
    Camino nuevoTrayecto;
    recorrido nuevoCamino = dijkstra(numeroNodos,matriz,idBencineras[i]);
    int numNodosNuevoCamino = 0;
    nuevoTrayecto.camino = guardarCamino(idBencineras[i],idPizzaPlanet,nuevoCamino.padre,numeroNodos,&numNodosNuevoCamino);
    nuevoTrayecto.numeroNodos = numNodosNuevoCamino;
    nuevoTrayecto.distancia = nuevoCamino.distancia[idPizzaPlanet];
    arregloCaminos[k] = nuevoTrayecto;
    k = k + 1;
}
```

```
//Ahora se tienen almacenados todos los posibles caminos desde las gasolineras hasta pizza planet, entonces se calculara cual de todos los caminos de:
// tierra -> gasolinera -> pizza_planet es el mas corto
//se fija el menor tiempo en infinito inicialmente
int menorTiempo = INT_MAX;
int idBencineraConMenorTiempo;
int idArregloBencineraConMenorTiempo;
for(int i = 0; i < numeroGasolinerasEncontradas; i++){
    //Primero se verifica que el camino encontrado es valido, para ello se mide la gasolina, en caso de no hacerlo se vbuscara un nuevo camino haciendo dijskstra
    if(camino.distancia[idBencineras[i]] > camino.distancia[idBencineras[i+1]]){
        for(int i = 0; i < numeroGasolinerasEncontradas; i++){
            Camino nuevoTrayecto;
            recorrido nuevoCamino = dijkstra(numeroNodos,matriz,idBencineras[i]);
            int numNodosNuevoCamino = 0;
            nuevoTrayecto.camino = guardarCamino(idBencineras[i],idPizzaPlanet,nuevoCamino.padre,numeroNodos,&numNodosNuevoCamino);
            nuevoTrayecto.numeroNodos = numNodosNuevoCamino;
            nuevoTrayecto.distancia = nuevoCamino.distancia[idPizzaPlanet];
            arregloCaminos[k] = nuevoTrayecto;
            k = k + 1;
        }
    }

    //Se consigue la cantidad de tiempo invertida desde la gasolinera hasta pizza planet
    //Se suma el tiempo que tarda la tierra a la gasolinera con el de la gasolinera a pizza planet
    int tiempo1 = arregloCaminos[i].distancia;
    int tiempo2 = camino.distancia[idBencineras[i]];
    int tiempoCalculado = tiempo1+tiempo2;
    if(tiempoCalculado < menorTiempo){
        menorTiempo = tiempoCalculado;
        idBencineraConMenorTiempo = idBencineras[i];
        idArregloBencineraConMenorTiempo = i;
    }
}
    }
    //Ahora se tiene la minima distancia posible, entonces se procedera a generar el camino final:
    //Se generara el camino mas corto desde la tierra hasta la bencinera seleccionada anteriormente
    numeroNodosCamino = 0;
    int * recorridoMasCortoGasolinera = guardarCamino(idTierra,idBencineraConMenorTiempo,camino.padre,numeroNodos,&numeroNodosCamino);
    //Ahora se conseguiora el camino desde la gasolinera hasta pizza planet
    arregloCaminos[idBencineraConMenorTiempo].camino;

    *numeroTotalNodos = numeroNodosCamino + arregloCaminos[idArregloBencineraConMenorTiempo].numeroNodos;
    //Se generara un arreglo con el camino Final
    int  *arregloCaminoFinal = (int*)malloc(sizeof(int)* *numeroTotalNodos);
    k = 0;
    //Se agrega el recorrido desde la tierra hasta la gasolinera
    for(int i = numeroNodosCamino-1; i >= 0 ; i--){
        arregloCaminoFinal[k] = recorridoMasCortoGasolinera[i];
        k = k + 1;
    }
    //Se agrega el recorrido desde la gasolinera hasta pizza planet
    for(int i = arregloCaminos[idArregloBencineraConMenorTiempo].numeroNodos-2; i >= 0; i--){
        arregloCaminoFinal[k] = arregloCaminos[idArregloBencineraConMenorTiempo].camino[i];
        k = k + 1;
    }
    //Finalmente se retorna el recorrido del camino mas corto
    *numeroTotalNodos = *numeroTotalNodos -1;
    *tiempoMasCorto = menorTiempo;
    return arregloCaminoFinal;

}
```

Figura 12: Implementación del algoritmo descrito para la solución.

El algoritmo “conseguirCaminoMasCorto” hace uso de todos los datos recopilados presentados en la metodología del punto 2.1.2 y debido a que el algoritmo de Dijkstra en matriz de adyacencia tiene una complejidad de $O(V^2)$ donde V es la cantidad de nodos o planetas en este caso, hay que sumarle el hecho de que en caso de no conseguir inmediatamente el recorrer de principio a fin se ejecutara Dijkstra múltiples veces para la

cantidad de gasolineras existentes entonces la complejidad de la solución implementada es de $O(N*V^2)$ donde N es la cantidad de gasolineras y V es la cantidad de planetas.

2.2 Análisis de los resultados

Para esta experiencia de laboratorio y debido a la falta de distintos archivos de prueba, solo fue posible el probar el programa creado para el grafo de ejemplo creado, para el cual el programa funciona de buena manera y cumple con generar el camino más corto desde la Tierra hasta Pizza Planeta cumpliendo todas las restricciones de combustible impuestas, generando como salida el siguiente archivo:

```
39
tierra->luna->axiom->rigel_7->axiom->pizza_planet
```

Figura 13: Archivo de salida generado “ruta.out”

Respecto a los tiempos de ejecución y como se mencionó en el último párrafo de la sección 2.1.3 del informe la complejidad de la solución implementada es de $O(N*V^2)$ donde N es la cantidad de gasolineras y V es la cantidad de planetas. tomando en cuenta todos los algoritmos y funciones presentes en la implementación de la solución en, no hay otro algoritmo con una complejidad mayor a la del algoritmo principal, por ende, posee la complejidad más alta de todas las funciones implementadas. Considerando la complejidad del programa, es posible notar que la solución implementada podría presentar tiempos de ejecución muy altos dependiendo si el grafo que se entrega inicialmente es muy extenso, como podría ser un grafo que corresponda a una ciudad entera, por ejemplo, o contextualizado al problema, un grafo correspondiente a toda una galaxia.

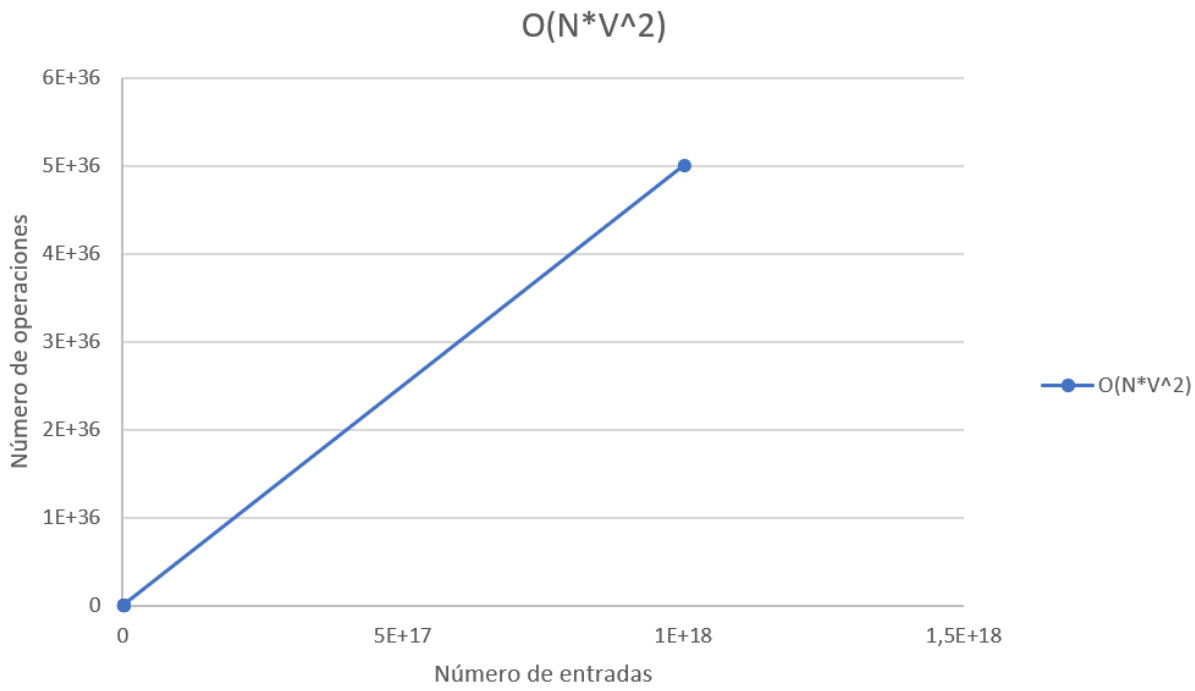


Figura 14: Gráfico de medición de tiempo por complejidad.

La figura 13 muestra el gráfico para un algoritmo de complejidad $O(N*V^2)$, según sus números de entradas sin embargo para facilitar la generación del grafico se fijó a 5 las gasolineras, para cada medición.

Haciendo un análisis del gráfico de la figura 13, el cual presenta medición de tiempo para distintos grafos de entrada, es posible determinar que efectivamente, el algoritmo es poco efectivo mientras más cantidad de datos deba analizar debido a su complejidad.

2.3 Conclusión

Para concluir, se logró a cabalidad el objetivo principal de este laboratorio, que era el crear una solución que lograra encontrar el camino más corto desde el planeta Tierra hasta el planeta Pizza Planet, tomando en consideración todos los elementos entregados en el grafo, como combustible y tiempo de viajar de un planeta a otro.

Sin embargo, la solución creada presenta el problema que se aprecia en el gráfico de la figura 13 del punto 2.2 del informe: es ineficiente para cantidades de datos grandes, por ende, si se intentara generar un recorrido más corto de un punto a otro, pero el grafo tiene 10 mil nodos tomaría una enorme cantidad de tiempo en lograr determinar cuál es el camino más corto. La principal mejora que se le puede hacer al programa creado es el de optimizar mejor los algoritmos usados para poder trabajar con grandes cantidades de datos que permitan mejorar la complejidad del algoritmo y por tanto tenga altos tiempos de ejecución, por ejemplo, quizá usar otro algoritmo de búsqueda que no sea Dijkstra, o usar algún método que aún no se haya visto en la asignatura.

Para finalizar, esta experiencia de trabajo de laboratorio sirvió para darse cuenta de la importancia de los distintos algoritmos y su utilidad, por ejemplo el algoritmo recién implementado es una variación de Dijkstra, al saber cómo funciona en su totalidad, permite el apreciar los distintos campos en los cuales el mismo algoritmo podría ser de gran ayuda para poder solucionar distintas problemáticas como la presentada en esta experiencia de laboratorio.

2.4 Referencias

[1] Vicedo J. (2013). *Divide y vencerás. La ventaja de los pequeños-grandes objetivos.* (Recuperado 16/01/2020). https://maximopotencial.com/divide-y-venceras-la-ventaja-de-los-pequenos-grandes-objetivos/

[2] Sena M. (2019). *Estructuras de datos.* (Recuperado 17/01/2020). https://medium.com/techwomenc/estructuras-de-datos-a29062de5483

[3] Salazar B. (2019). *Algoritmo de Dijkstra.* (Recuperado 17/01/2020). https://www.ingenieriaindustrialonline.com/investigacion-de-operaciones/algoritmo-de-dijkstra/