

# Tarea 3 - Análisis de Algoritmos y Estructura de Datos

Catalina Riquelme Zamora  
*Departamento de Ingeniería Informática*  
*Universidad de Santiago de Chile, Santiago, Chile*  
 1-2021

## I. INTRODUCCIÓN

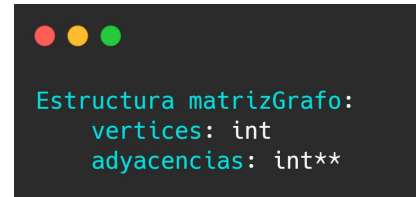
Tras el inicio de la pandemia de COVID-19, en donde se vio afectada casi en la totalidad los países, se han logrado desarrollar distintas vacunas con el fin de aminorar los efectos y contener el contagio de la pandemia, con el fin de que dichas vacunas se logren aplicar a la mayor población posible, manteniendo ciertas restricciones de almacenamiento de estas el centro de distribución de la comuna de Maipú busca contar con una buena estrategia de transporte de insumos médicos para poder mantener el ritmo de vacunación y presentar el menor costo de transporte, para lograr trazar las rutas debe tener en consideración la fórmula de  $\text{costoC} = (D \cdot P) / S$ , donde C es el costo en unidades monetarias, D se asocia con la distancia evaluada en kilómetros, por otra parte P hace referencia al peso evaluado en toneladas y finalmente S corresponde al subsidio.

## II. SOLUCIÓN PROPUESTA

Para resolver este problema es necesario abordar esta situación desde la implementación de grafos y listas de adyacencias, donde los grafos serán trabajados en base a una matriz de adyacencias, la cual contendrá los diversos vértices de este grafo junto con el peso de las aristas, logrando representar de manera correcta las diversas rutas de este. Llevándolo al plano de la problemática planteada los diversos centros a los cuales se les debe repartir las vacunas serán representados por los vértices de este grafo, mientras que los caminos que conectan serán las aristas de estas, por otra parte es importante destacar que el costo del camión a recorrer las diversas rutas hacia los centros serán los pesos de las aristas.

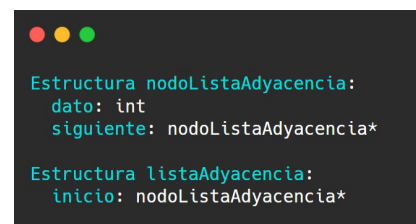
Para llevar a cabo esto es necesario definir dos estructuras principales, en donde la primera hace referencia al grafo en donde se almacenará el número total de vértices que forman el grafo con el que se desee trabajar, además de una matriz de adyacencia la cual contendrá las adyacencia de los diversos vértices acompañados de sus respectivos costos, esta estructura se puede ver en la Figura 1.

La segunda estructura principal hace referencia a una lista de adyacencias, la cual corresponde a una lista enlazada que contiene un dato y un puntero que apunta al siguiente nodo



```
Estructura matrizGrafo:
  vertices: int
  adyacencias: int**
```

Figura 1: Estructura grafo



```
Estructura nodoListaAdyacencia:
  dato: int
  siguiente: nodoListaAdyacencia*

Estructura listaAdyacencia:
  inicio: nodoListaAdyacencia*
```

Figura 2: Estructura lista adyacencia

de esta lista, esta estructura se puede ver en la Figura 2.

Con la finalidad de encontrar un camino mínimo dado un grafo G, junto a un vértice inicial v se implementa el algoritmo Dijkstra con la finalidad de encontrar la ruta la cual debe seguir el camión en consecuencia a los diversos caminos establecidos por el grafo, en donde cada vértice de este corresponde a un consultorio, siendo el vértice 0 el centro de distribución, es decir, el punto de partida. Con la implementación de este algoritmo se logrará encontrar un camino entre vértices de tal manera que la suma de sus costos de las aristas que lo constituyan sea mínima.

El algoritmo Dijkstra consiste en primera instancia en asignar cero a la distancia mínima para el vértice inicial, mientras que para el resto de los vértices estos empezaran con valor infinito, posteriormente se marca este vértice (inicial) como visitado, una vez realizada esta acción se calcula la distancia para llegar a cada uno de los vértices vecinos desde el inicial, una vez calculadas las diversas distancias se escoge el vértice con una distancia menor, además que considerando que no se encuentre visitado con la finalidad de que este se convierta en el nuevo vértice actual, una vez obtenido, se calcula la distancia para llegar a cada uno de sus vecinos, repitiendo el proceso hasta que no

```

dijkstra(grafo, inicio): arr
...Se dan los valores iniciales
FOR i<-0 TO grafo->vertices
...Se marca como no visitado
visitado[i] <- FALSE
padre[i] <- NULL
IF (A[inicio][i] > 0) THEN
  distancia[i] <- A[inicio][i]
ELSE
  distancia[i] <- INFINITE
...La distancia mínima para el vértice inicial es 0
distancia[inicio] <- 0
...Se marca el vértice inicial como visitado
visitado[inicio] <- TRUE
posicionRuta <- 0
...Mientras queden vértices por visitar
WHILE (esVacioVisitados(visitado, grafo) = 0) DO
...Se escoge el vértice no visitado con distancia más baja
minimo <- extraerMinimo(distancia, visitado, grafo->vertices)
...Se marca como visitado el vértice escogido
visitado[minimo] <- TRUE
listaAdy <- crearListaVacia()
...Se obtienen los adyacentes de este
listaAdy <- obtenerAdyacentes(grafo, minimo)
aux <- listaAdy->inicio
...Para el vértice actual, se calcula la distancia para llegar a cada uno de sus vecinos
WHILE (aux <= NULL) DO
  peso <- A[minimo][aux->dato]
  IF (distancia[aux->dato] > (distancia[minimo] + peso)) THEN
    distancia[aux->dato] <- distancia[minimo] + peso
    padre[aux->dato] <- minimo
  aux <- aux->siguiente
ruta[posicionRuta] <- minimo
posicionRuta <- posicionRuta + 1
RETURN ruta

```

Figura 3: Algoritmo Dijkstra

sea posible alcanzar un nuevo vértice. El pseudocódigo de este algoritmo se ve reflejado en la Figura 3.

### III. RESULTADOS Y ANÁLISIS

Al evaluar el tiempo en ejecutar el algoritmo Dijkstra evaluando diversos tamaños de entrada (aumentando el número de vértices con el finde producir un nuevo grafo, con un mayor número de aristas) se pueden apreciar los resultados en las siguientes tablas y gráficos:

n	Tiempo (s)
5	0.004
7	0.018
9	0.058

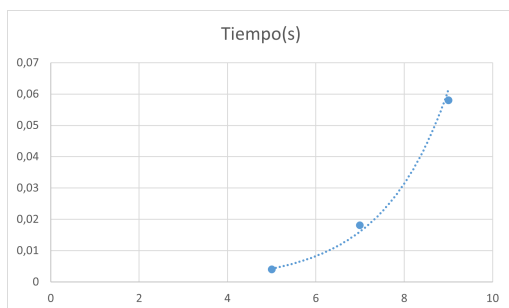


Figura 4: Tiempos de ejecución Dijkstra

Tras apreciar los resultados mostrados en la tabla y en gráfico en la Figura 4 se concluye que existe un aumento exponencial entre el aumento del tiempo y el aumento del número de vértices de la entrada, esto debido a que como se puede apreciar en la tabla al aumentar la entrada esto genera un aumento en los ciclos a realizar. Con esto se puede

confirmar que la complejidad teórica (complejidad cuadrática) calza con el tiempo al evaluar el algoritmo Dijkstra

### IV. CONCLUSIONES

A través del programa realizado en c, se logró escribir un archivo salida.out de manera exitosa utilizando grafos y listas de adyacencias, las cuales se encuentran relacionadas con listas enlazadas de manera simple. Al realizar la toma de tiempos de la función principal esta coincide con la complejidad de manera teórica lo que indica que fueron calculadas de manera correcta. Una de las desventajas del algoritmo propuesto para realizar esta tarea es la no implementación de verificadores en diversas partes del programa, de manera que si el usuario ingresa un archivo incorrecto, el programa caerá y no dirá la razón de esta caída, esto se puede mejorar verificando si los archivos leídos poseen la estructura requerida para el correcto funcionamiento del programa, evitando diversos problemas que se pueden generar al ingresar tanto los grafos como los insumos. Una de las ventajas del algoritmo es el uso de diversas listas mejorando la comprensión del código y permitiendo en futuras alteraciones quitar y agregar elementos con facilidad.

Para mejorar el trabajo el proximas entregas es necesario empezar el algoritmo con anterioridad para evitar que el tiempo afecte la calidad de la entrega final, además de obtener un mayor conocimiento respecto al uso de punteros y listas de adyacencias antes de empezar a trabajar.

### V. MANUAL DE USUARIO

Para ejecutar de manera correcta el programa se requieren dos archivos los cuales deben tener el nombre conexiones.in y insumos.in, el incorrecto nombramiento de los archivos puede causar una caída en el programa. Dentro del archivo conexiones se encuentra almacena la información necesaria trazar un grafo y lograr trabajar con el de manera correcta, logrando formar los vértices y aristas que lo conforman, dentro debe existir en la primera línea el número de aristas y el número de vértices, separadas por un espacio, ordenadas en el mismo nombradas anteriormente. Posteriormente cada línea debe contener como primer valor un vértice inicial, asimismo como segundo valor un vértice de llegada y como tercer valor el peso de la arista, estos valores deben encontrarse separados por un espacio, con esta información es posible generar una matriz representante del grafo con sus diversas aristas, vértices y costos, todos estos valores deben ser positivos, el ingreso erróneo de este archivo puede provocar una mala representación de la matriz provocando un mal cálculo de camino mínimo.

Por otra parte, dentro del archivo insumos.in se encuentra toda la información relacionada con los insumos a repartir, donde la primera línea contiene el número de centros (vértices) sin incluir el centro de distribución (vértice 0),

seguido por el subsidio entregado, cabe destacar que estos valores deben estar separados por un espacio, además del valor asignado a subsidio debe ser positivo. Posteriormente cada línea debe contener como primer valor el número de identificación de cada centro, estos deben ser a lo largo del archivo en orden ascendente, partiendo desde la identificación 1 y sin omitir ningún valor, como segundo valor se encuentra la capacidad de insumos, medida en toneladas, el ingreso erróneo de este archivo puede provocar cálculos erróneos.

El programa al ser ejecutado crea un archivo salida.out, el cual contendrá, la información relacionado con la capacidad, indicando el número de toneladas de insumos que tiene el camión desde su salida desde el centro de distribución, además el archivo contendrá el subsidio entregado por el gobierno, por otra parte, contendra del costo de la ruta, finalmente contendrá la ruta la cual seguirá el camion.