# Tarea 3 - Estructuras de Datos (INF-134)

Publicación: Viernes 06 de Junio
Entrega: Lunes 23 de Junio - 23:55h vía Aula

2025-1

## 1 Reglas

- La tarea debe realizarse en grupos de 2 personas. Cualquier situación excepcional debe ser aprobada por el ayudante coordinador (Tomás Barros).
- Debe usarse el lenguaje de programación C++. Al realizar la evaluación, la tarea será compilada usando el compilador g++ en Linux, usando el comando:

  `g++ tarea3.cpp -o tarea3 -Wall`.

  No se aceptarán entregas realizadas en Windows o compiladores online como Dev-C++, Replit, GDB, etc. En caso de usar Mac, se recomienda usar XCode, y de ser posible, revisar la tarea también en Linux.
- No se permite usar la biblioteca STL, así como ninguno de los contenedores y algoritmos definidos en ella (e.g. vector, list, etc.). En general no se permite importar ninguna estructura de datos, salvo por string.
- Una tarea que no compile tendrá nota 0. En dicha situación, el/la estudiante puede apelar con el ayudante coordinador.
- La tarea puede entregarse con máximo 1 día de atraso, en cuyo caso la nota máxima será 50.
- Solo una de las tres tareas del semestre puede tener nota inferior a 30.
- Si se detecta **COPIA** o abuso de herramientas de generación automática de código (ChatGPT), la situación será revisada por el ayudante coordinador y profesor coordinador. Si se considera que el estudiante no entiende el código que escribió, tendrá nota 0 en la tarea.

# 2 Problema a resolver

La última tarea será programar Uber.

Podemos representar una ciudad con un TDA Grafo, donde los nodos son cruces de calles y los arcos son calles. La imagen de la figura 1 representa una versión modificada del centro de Viña del Mar.

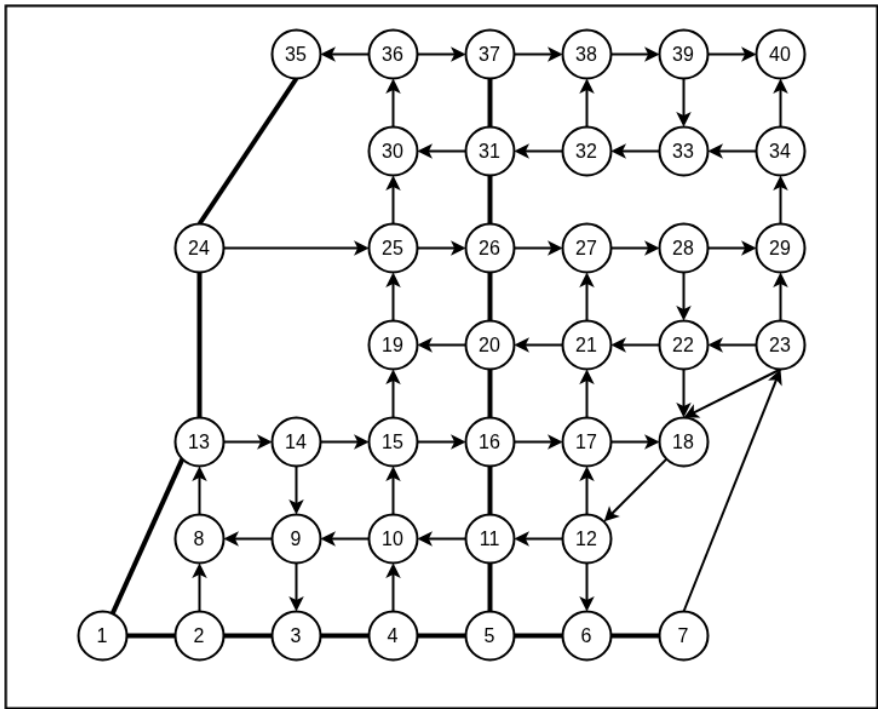


Figura 1: Centro de Viña del Mar representado como grafo (Con algunas modificaciones).

Para motivos de la tarea, consideraremos que todas las calles son del mismo largo, y por ende, el grafo no tendrá pesos. El grafo es dirigido, con algunos arcos bidireccionales indicando avenidas. Los arcos dirigidos se muestran con flechas en la figura 1. Los arcos bidireccionales se representan como líneas gruesas sin flechas en la figura.

## 2.1 Cargado de mapa (10 pts)

Se debe leer un archivo `data.txt` que contenga la información necesaria para crear un grafo. Además de la información de nodos y arcos, se debe leer la ubicación inicial de un conjunto de posibles conductores para los viajes. La figura 2 muestra una versión del mapa del centro de la ciudad de Viña con cuatro conductores situados en las intersecciones 2, 7 , 31 y 40.

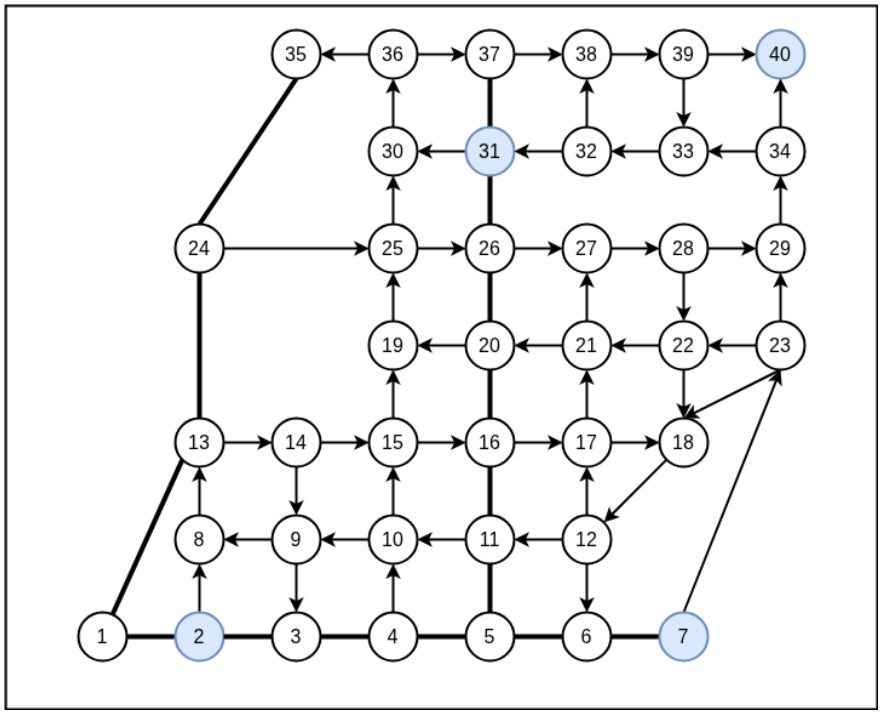


Figura 2: Centro de Viña del Mar representado como grafo (Con algunas modificaciones). Inicialmente, hay un conductor en cada nodo azul.

El formato del archivo se presenta en la sección 8.

## 2.2 Solicitar Uber (80 pts)

Se debe implementar un método `solicitar_uber(...)` que reciba dos nodos (inicial y final), y entregue el recorrido y el costo del camino más corto para el viaje.

El costo se define de la siguiente manera:

$$\text{Costo} = 300 \times D_{Conductor} + 500 \times D_{Viaje}$$

Donde:

- $D_{Conductor}$ es la distancia desde el conductor más cercano[1] hasta el pasajero.
- $D_{Viaje}$ es la distancia desde la ubicación inicial del pasajero hasta el destino.

Cabe destacar que tras realizar el viaje, **la nueva posición del conductor será el nodo de destino del viaje.**

[1] En caso de empate, se toma el conductor ubicado en el nodo de índice menor

#### Ejemplo

Si hay un conductor en el nodo 31 y otro en el nodo 37, la ruta óptima desde el nodo 35 al nodo 8 es:

1. {31, 30, 36, 35}, para que el conductor recoja al pasajero.
2. {35, 24, 13, 1, 2, 8}, para el viaje[2].

El costo del recorrido sería:

$$\text{Costo} = 300 \times 3 + 500 \times 5 = 3400$$

Y el conductor quedará situado en el nodo 8 al final del viaje.

### 2.3 Análisis de Complejidad (10 pts)

La funcionalidad del programa debe lograrse utilizando `BFS` o `Dijkstra` (a elección propia). Como último requisito, se solicita realizar un análisis de complejidad temporal sobre su función `solicitar_uber(...)`, el cual debe incluir:

#### Definición de la complejidad

Se debe presentar el desglose y resultado final de su análisis usando notación Big O.
*Ej:* $O(5n^3 + 2n^2) = O(5n^3) = O(n^3)$

#### Justificación

Se debe justificar el análisis elemento por elemento.
*Ej: La complejidad de Dijkstra es X, y se ejecuta N veces, además hay un ciclo for...*

**Nota:** Recuerden que la implementación del TDA Grafo afecta la complejidad de algunas operaciones. Este impacto debe ser comentado en su análisis.

## 3 Ejecución

Al ejecutar el programa con el mapa de Viña (`data1.txt`), éste deberá colocar conductores en los nodos especificados en el archivo.

Luego, a través de líneas de comando se indicarán los viajes a calcular. Para cada viaje el programa deberá presentar por pantalla las rutas y costos correspondientes. A continuación se presenta un ejemplo de ejecución para el mapa de Viña.

```
usuario@pc:~$ g++ tarea3.cpp -o tarea3 -Wall
usuario@pc:~$ ./tarea3
Ingrese viaje: 30 8
Ruta : {30, 36, 35, 24, 13, 1, 2, 8} - Costo: 3800
Ingrese viaje: 40 19
```

[2]Existe una ruta del mismo largo: {35, 24, 13, 14, 9, 8}. Ambos son válidos

```
Ruta : {} - Costo: -1
Ingrese viaje: 15 23
Ruta : {15, 16, 11, 5, 6, 7, 23} - Costo : 3900
Ingrese viaje: 7 12
Ruta : {7, 23, 18, 12} - Costo: 1500
Ingrese viaje: 36 39
Ruta : {36, 37, 38, 39} - Costo: 3300
Ingrese viaje: -1 -1
```

Actualización: El costo de la ultima ruta se corrigió, de 3000 a 3300.

Notar que el programa termina al recibir el input `-1 -1`.

# 4 Detalles de implementación

1. Se debe utilizar un TDA Grafo para almacenar la información de la ciudad.
2. La implementación particular del TDA Grafo queda a elección propia.
3. Se debe implementar el algoritmo que permita encontrar el camino más corto especificando las correspondientes rutas y costos.
4. Si al solicitar un Uber se pide un viaje imposible, este debe retornar una ruta vacía y un costo de -1.

# 5 Notas adicionales

- Toda la memoria que dependa del input del programa debe ser solicitada de manera dinámica, y debe ser liberada correctamente al finalizar el programa. Generalmente, la solicitud de memoria será en función de los contenidos del archivo correspondiente.
- Deben entregar dos archivos data.txt distintos: `data1.txt` y `data2.txt`. `data1.txt` debe contener la información del mapa de Viña, mientras que `data2.txt` puede contener un mapa a su elección.

# 6 Formato de Entrega

Se debe entregar un archivo llamado `tarea3-'apellido1'-'apellido2'.zip`[3] que contenga:

1. `tarea3.cpp` : El archivo que contiene su código
2. Los archivos `data1.txt` y `data2.txt` necesarios para su funcionamiento.
3. `README.txt` con los nombres y roles de los integrantes. Para esta tarea, este archivo también deberá contener su análisis de complejidad.

[3] 1 apellido por cada integrante. Se debe entregar el *primer* apellido.

Recordar que no seguir el formato de entrega conlleva hasta 10 puntos de descuento. *Favor adjuntar roles*. La compilación de su tarea es imperativa para la evaluación. **Una tarea que no compile tendrá nota 0.**

Recordar que es su responsabilidad asegurarse de que se adjunten los archivos correctos a la entrega.

# 7 Directrices de programación

Las directrices corresponden a buenas prácticas de programación, las cuales serán tomadas en consideración para la evaluación de la tarea. Si el código fuente está desordenado, se pueden descontar hasta 20 puntos de la nota. Un código entregado completamente sin comentarios, tendrá **nota 0**.

Cada función de código cuya función no sea inmediatamente aparente, debe tener un comentario de la siguiente forma:

```
/*****
*   TipoFunción NombreFunción
******
*   Resumen Función
******
*   Input:
*       tipoParámetro NombreParámetro : Descripción Parámetro
*        .......
******
*   Returns:
*       TipoRetorno, Descripción retorno
*****/
```

Además,

- Se deben utilizar nombres de variables descriptivos.
- Se deben seguir buenas prácticas respecto al uso de funciones. Es decir, se deben utilizar funciones para evitar redundancia y para modularizar las componentes de su código.
- Se debe considerar la indentación correcta del código.

# 8 Formato de Entrada

El formato de archivos de entrada se presenta a continuación. La primera línea indica la cantidad de nodos, la cantidad de arcos y la cantidad de conductores. Luego, por cada uno de los arcos en cada línea se presenta el nodo de inicio y nodo de término (numerados de 1 en adelante). Observe que los arcos bidireccionales se listan dos veces, una por cada sentido. Por último se listan los nodos en que se ubican los conductores (numerados de 1 en adelante). Actualización: Se agregaron dos arcos faltantes: 24 13 y 13 24

```
40 79 4
1 2
2 1
1 13
13 1
2 3
3 2
2 8
3 4
4 3
4 5
5 4
4 10
5 6
6 5
5 11
11 5
6 7
7 6
7 23
8 13
9 3
9 8
10 9
10 15
11 10
11 16
16 11
12 6
12 11
12 17
13 14
14 9
14 15
15 16
15 19
16 20
20 16
16 17
17 21
17 18
18 12
19 25
24 13
13 24
20 19
20 26
26 20
21 20
21 27
22 21
22 18
23 22
23 18
```

```
23 29
24 35
35 24
24 25
25 30
25 26
26 31
31 26
26 27
27 28
28 22
28 29
29 34
30 36
31 30
31 37
37 31
32 31
32 38
33 32
34 33
34 40
36 35
36 37
37 38
38 39
39 33
33 40
2 7 31 40
```

# 9 Contacto

## Profesores

- Elizabeth Montero (**Coordinadora**) (elizabeth.montero@usm.cl)
- Alondra Rojas (alondra.rojas@usm.cl)
- José Miguel Cazorla (jcazorla@usm.com)
- Ricardo Salas (ricardo.salas@usm)
- Juan Pablo Castillo (juan.castillog@sansano.usm.cl)
- Roberto Díaz (roberto.diazu@usm.cl)

## Ayudantes (CC)

- Tomás Barros (**Coordinador**) (tomas.barros@sansano.usm.cl)
- Jaime Inostroza (jinostroza@usm.cl)
- Facundo Riquelme Lucero (friquelmel@usm.cl)
- Franco Cerca (franco.cerda@usm.cl)
- Bastián Jimenez (bjimenez@usm.cl)

## Ayudantes (SJ)

- Juan Alegría (juan.alegria@usm.cl)
- Damián Rojas (damian.rojas@usm.cl)
- Ignacia Nettle Oliveri (inettle@usm.cl)
- Alvaro Aceituno Alarcon (alvaro.aceitunoa@usm.cl)
- Lucas Morrison (lucas.morrison@sansano.usm.cl)
- Javier Matamala Gonzalez (javier.matamalag@usm.cl)