Universidad de los Andes > FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

Estructuras de Datos y Algoritmos (EDA)
Tarea 4 [opcional]: Grafos

Prof: José M. Saavedra Rondo
Ayudantes: Braulio Torres & Cristóbal Loyola & Francisco Jiménez

Noviembre 2022

1. Objetivo

Comprender la importancia e implementar estructuras de datos basadas en grafos para la solución de problemas computacionales.

2. Descripción

En esta tarea deberán crear una herramienta que permita leer un grafo ingresado a través de un archivo de texto. El grafo ingresado deberá ser representado a través de un **arreglo de aristas adyacentes**, una variante del AAL que almacena las aristas adyacentes de un vértice como una lista enlazada (ver capítulo 8 del libro guía). Una vez cargados los datos, deberán calcular el camino más corto entre cada par de vértices, siguiendo el algoritmo de **Dijkstra** (también descrito en el libro guía). Luego, la aplicación deberá permitir ingresar iterativamente un vértice inicial y un vértice final para mostrar el camino más corto entre ambos vértices. Si no existe camino entre los vértices ingresados, el programa deberá mostrar el mensaje "No existe camino". El programa terminará cuando un usuario ingresa -1 como vértice inicial.

3. Especificación Detallada

3.1. Archivo Grafo

El programa a crear deberá leer archivos de tipo grafo que tiene la siguiente sintaxis:

```
V:<número de vértices>
<arista_1>
<arista_2>
<arista_3>
...
```

Una arista en el archivo tiene la forma $a-b-w$, donde a es el vértice inicial, b es el vértice final y w es un valor asociado a la arista. Pueden suponer que siempre se ingresa vértices válidos. A continuación se muestra un ejemplo de archivo grafo, al que llamaremos *migrafo.grafo*.

```
V: 4
0-1-2
0-2-5
1-3-3
3-2-3
3-0-4
```

El archivo *migrafo.grafo* representa el grafo dirigido de la Figura 1.

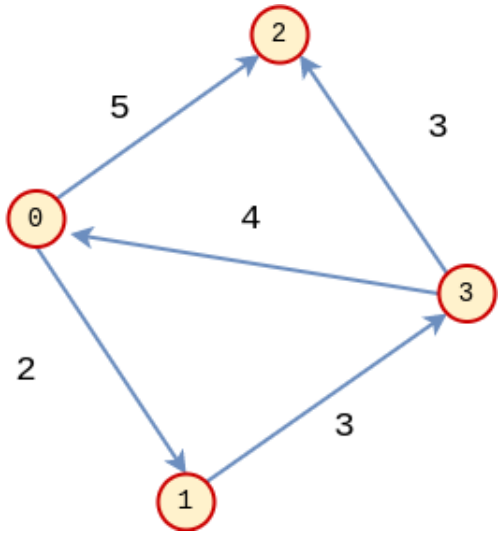

Figura 1: Grafo con 4 vértices y 5 aristas, según indica el archivo *migrafo.grafo*.

3.2. Representación como AAL, arreglo de aristas de adyacencia.

Deberán crear el ADT Graph, que permita representar el grafo de entrada como una **arreglo de aristas de adyacencia** (revisar capítulo 8 del libro guía). Para hacer la implementación más fácil, pueden considerar que un nodo que representa a un vértice no necesita conocer a sus nodos antecesores. Así, solo necesitan el *id* y la lista de aristas adyacentes para cada vértice.

3.3. Algoritmo Dijkstra

La solución del camino más corto deberá ser implementada a través del algoritmo Dijkstra visto en clase. Así, necesitan representar el arreglo de antecesores A, arreglo de distancias D y el *MIN_Heap* H que representa a los nodos alcanzables. **¡En esta tarea es indispensable utilizar el heap!**

3.4. Ejecución

El programa a ejecutar deberá ser nombrado como $grafo$, que recibe inicialmente un archivo de grafo y muestra el mensaje "proceso OK" si el grafo se ha creado correctamente, "Error" en caso contrario. Un ejemplo de la ejecución inicial es el siguiente:

```
$grafo  migrafo.grafo
Proceso OK
```

En seguida, el programa les pedirá un vértice inicial y uno final. El programa imprimirá el camino más corto, si existe, entre los vértices. Por ejemplo:

```
Vértice Inicial:  0
Vértice Final:  3
El camino más corto es:
0-1-3 (distancia = 5)
```

Note que junto al camino se mostrará la distancia de tal camino. El programa deberá seguir pidiendo un vértice inicial y uno final repetidamente, hasta que el usuario ingrese -1 como vértice inicial.

4. Informe

1. **Abstract o Resumen**: es el resumen del trabajo.
2. **Introducción**: se describe en el tópico del trabajo y los objetivos. Es recomendable comentar algunas aplicaciones del problema a resolver. (10 %)
3. **Desarrollo**: aquí se describe el diseño e implementación de los ADTs y funciones relacionados con su trabajo. (40 %).
4. **Resultados Experimentales y Discusión**: aquí se presentan ejemplos de ejecución. Deberá presentar 2 ejemplos de archivos de grafos mostrando la ejecución de su programa para encontrar los caminos más cortos entre vértices. Es recomendable mostrar casos en los que no hay camino. (40 %).
5. **Conclusiones**: ideas o hallazgos principales sobre el trabajo. (10 %)

5. Restricciones

1. Pueden trabajar en grupos de 2 estudiantes.
2. Todos los programas deben ser propios, permitiendo solamente utilizar el código disponible en el repositorio del curso `https://github.com/jmsaavedrar/eda_cpp/`.
3. El hallazgo de plagio será penalizado con nota 1.0, para todos los grupos involucrados.
4. Todos las implementaciones deben ser realizadas en C++.
5. **La entrega del informe es obligatorio**. Un trabajo sin informe no será calificado, asignando la nota mínima igual a 1.0.

6. Entrega

La entrega se debe realizar por Canvas hasta el sábado 19 de noviembre, 2022, 23:50 hrs. La entrega debe incluir:

1. Código fuente (en C++), junto a un README con los pasos de compilación.
2. Informe