	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	TISTA GARCÍA EDGAR
Asignatura:	ESTRUCTURA DE DATOS Y ALGORITMOS II
Grupo:	5
No de Práctica(s):	PRACTICA # 6 – 7. ALGORITMOS DE GRAFOS
Integrante(s):	CARRILLO CERVANTES IVETTE ALEJANDRA
No. de Equipo de cómputo empleado:	TRABAJO EN CASA
No. de Lista o Brigada:	09
Semestre:	2022 - 1
Fecha de entrega:	03 NOVIEMBRE 2021
Observaciones:	

CALIFICACIÓN: _____

PRACTICA # 6-7. ALGORITMOS DE GRAFOS

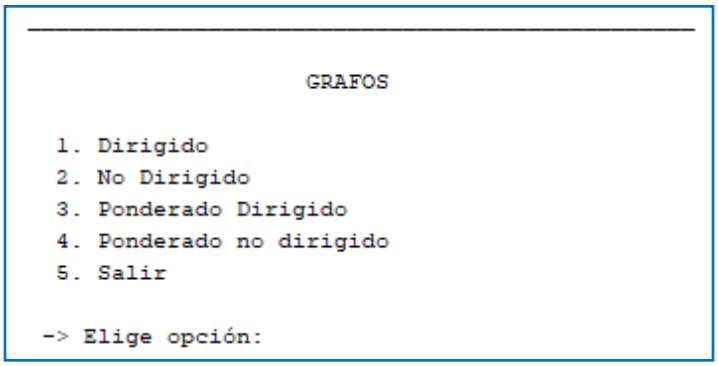
Objetivo General: El estudiante conocerá las formas de representar un grafo e identificará las características necesarias para comprender el algoritmo de búsqueda por expansión y profundidad.

Objetivo de Clase: El alumno será capaz de implementar y comprender los grafos, así como los recorridos.

ACTIVIDADES PARA EL DESARROLLO DE LA PRÁCTICA

Implementación de grafos en Java

Se creó un programa el cual tiene como función principal, implementar diferentes tipos de grafos en el lenguaje de programación JAVA, para ello y fuera más eficiente su ejecución, se creó el siguiente menú de opciones:



Menú de opciones

Para implementar cada uno de los grafos presentados en el menú de opciones, se creó una clase llamada ***Graph***, la cual tiene los siguientes atributos:

→ Rango que se quiere que tengan los nodos del grafo (variable tipo int).
→ LinkedList tipo Integer, será un arreglo en donde se irán almacenando los nodos creados según el rango. Ej. Si su rango es de 0-4, el arreglo tendrá 5 índices.
→ Tabla HashMap que tiene como clave un tipo de dato Integer (el valor de la arista, en caso de ser un grafo ponderado) y como valor una tabla HashMap, la tabla que tiene como valor tiene como clave y valor un tipo de dato Integer. (Nodo inicial y nodo final de la arista)

<Valor de la arista <Nodo Inicial, Nodo Final>>

Esta misma clase tiene los siguientes métodos:

→ **Graph.** Método constructor sobrecargado.
La primera forma de este método recibe como parámetro el rango que se quiere que tengan los nodos; mientras que, la segunda forma de este método, además de recibir como parámetro el rango que se quiere que tengan los nodos, recibe una tabla HashMap vacía (en donde se irá almacenando el valor de la arista, el valor del nodo inicial y el valor del nodo final). Ambas formas de este método inicializan el arreglo que se tiene como atributo (De tamaño según el rango ingresado) con listas de tipo Integer; cabe recalcar que solo en la segunda forma de usar este método se inicializa también la tabla HashMap.

→ **getTabla**. Es un método de acceso, el cual retorna la tabla HashMap de cada grafo ponderado.

→ **addEdge.** Tiene como parámetro el valor del nodo inicial, el valor del nodo final y la opción del menú principal que eligió el usuario. Agrega al indice correspondiente del arreglo, en donde se almacena cada nodo, el valor de los nodos que va conectado con este.
En caso de que la opción sea 1, es decir, que sea un grafo dirigido, entonces solo se agregará a la lista del indice correspondiente al nodo inicial, el valor del nodo final; mientras que, en caso de que la opción sea 2, es decir un grafo no dirigido, se agregará a la lista del indice correspondiente al nodo inicial, el valor del nodo final y viceversa.

→ addEdgePonderado Tiene como parámetro el grafo creado, el valor del nodo inicial, el valor del nodo final y la opción del menú principal que eligió el usuario. Igual que el método anterior, agrega al indice correspondiente del arreglo el valor de los nodos que va conectado con este, siguiendo la misma lógica; sin embargo, esta vez al ser un grafo ponderado, se le pide al usuario el valor de la arista que conectan los dos nodos y se agrega a la tabla hash del correspondiente grafo, al igual que se agrega el valor del nodo inicial y el valor del nodo final a dicha tabla.

→ **crearGrafo.** Tiene como parámetro la opción del menú principal que eligió el usuario. Tiene como función principal, crear un grafo según la opción que se le pasa como parámetro; primero, se le asigna a cada grafo un rango de [0-999] para que el usuario puede ingresar cualquier valor entre ese rango como nodo y se crea el grafo correspondiente según la opción dada; después, se le pregunta al usuario cuantas aristas quiere que tenga su grafo y con respecto a ese número, se crea un ciclo de repetición *for* el cual tiene como objetivo pedir el valor del nodo inicial y el nodo final de cada arista y con el método addEdge, agregar dichos nodos al grafo.

→ **printGraph**. Este método tiene como parámetro el grafo que se quiere imprimir y la opción del menú principal que eligió el usuario. Para poder imprimir el grafo, lo que se hizo fue recorrer el arreglo que tiene como atributo cada grafo y para cada índice de dicho arreglo se verificaba si la lista que se encuentra en este está vacía o no, es decir, se verifica que el nodo en el índice "n" este conectado con algún otro, en caso de ser así se imprime la lista que contiene dicho índice del arreglo. Se repetirá las veces necesarias para formar la lista de adyacencia del grafo correspondido. Sin embargo, al imprimir un nodo ponderado, además de imprimir la lista de adyacencia, se va a llamar al método "verAristas", el cual imprimirá el nodo inicial, el nodo final y el valor de cada una de las aristas que tiene el nodo.

→ **verAristas.** Tiene como parámetro el grafo y la opción del menú principal que eligió el usuario. Tiene como función ver el valor de las aristas creadas en el grafo correspondientes, para ello primero se obtiene la tabla donde se almacena el valor del nodo inicial, final y el valor de cada arista, y así con base a esto y mediante un ciclo for each se va imprimiendo el valor de la tabla HashMap.

→ **BFS.** Tiene como parámetros el grafo donde se hará el recorrido BFS, el nodo donde se empezará y la opción del menú principal que eligió el usuario. Para realizar este recorrido lo primero que se hizo fue verificar que el nodo que ingresará el usuario exista y en dado caso de que si se crea una lista, la cual será la "queue" en donde se irán almacenando los valores de los nodos visitados, para agregar algún valor a esta lista se recorrió el arreglo de listas del grafo explorando los nodos que se encuentran conectados al nodo inicial y marcando con un valor booleano si es que ya se visito o no, en dado caso de que ya todos se encuentren visitados se imprimirá la lista con los nodos con forme se fueron visitando.

→ **DFS.** Tiene como parámetros el grafo donde se hará el recorrido DFS, el nodo donde se empezará y la opción del menú principal que eligió el usuario. Para realizar este recorrido lo primero que se hizo fue verificar que el nodo que ingresará el usuario exista y en dado caso de que si, se creará un arreglo de tipo booleano del tamaño del rango de los nodos y posterior a ello se llamará a la función **DFSUtil** la cual irá visitando nodo por nodo hasta que todos se encuentren visitados. Finalmente, imprimirá la lista correspondiente con todos los nodos visitados.

Opciones del menú

Para cada opción del menú se llamará al método **crearGrafo** descrito anteriormente, pasándole como parámetro la opción del menú principal y según corresponda realizar las instrucciones necesarias para crear cualquier tipo de Grafo como se describió anteriormente.

```
. __ . __ . __ . __ . __ . __ . __ . __ .

                GRAFO DIRIGIDO

Ingresa el número de aristas que quieres que tenga tu grafo: 4
NOTA: Solo puedes ingresar nodos en un rango de [0 - 999]

Arista 1
Nodo Inicial: 1
Nodo Final: 2

Arista 2
Nodo Inicial: 4
Nodo Final: 6

Arista 3
Nodo Inicial: 7
Nodo Final: 8

Arista 4
Nodo Inicial: 5
Nodo Final: 4
```

Grafo Dirigido

```
. __ . __ . __ . __ . __ . __ . __ . __ .

                GRAFO NO DIRIGIDO

Ingresa el número de aristas que quieres que tenga tu grafo: 8
NOTA: Solo puedes ingresar nodos en un rango de [0 - 999]

Arista 1
Nodo Inicial: 1
Nodo Final: 5

Arista 2
Nodo Inicial: 5
Nodo Final: 4

Arista 3
Nodo Inicial: 4
Nodo Final: 0

Arista 4
Nodo Inicial: 0
Nodo Final: 2

Arista 5
Nodo Inicial: 0
Nodo Final: 6

Arista 6
Nodo Inicial: 5
Nodo Final: 7

Arista 7
Nodo Inicial: 7
Nodo Final: 3

Arista 8
Nodo Inicial: 4
Nodo Final: 7
```

Grafo no Dirigido

```
. __ . __ . __ . __ . __ . __ . __ . __ .

                GRAFO PONDERADO DIRIGIDO

Ingresa el número de aristas que quieres que tenga tu grafo: 4
NOTA: Solo puedes ingresar nodos en un rango de [0 - 999]

Arista 1
Nodo Inicial: 1
Nodo Final: 2
Ingresa el valor de la arista: 16

Arista 2
Nodo Inicial: 2
Nodo Final: 4
Ingresa el valor de la arista: 18

Arista 3
Nodo Inicial: 4
Nodo Final: 7
Ingresa el valor de la arista: 9

Arista 4
Nodo Inicial: 7
Nodo Final: 1
Ingresa el valor de la arista: 15
```

Grafo Ponderado Dirigido

```
 __  __  __  __  __  __  __  __  __

              GRAFO PONDERADO NO DIRIGIDO

Ingresa el número de aristas que quieres que tenga tu grafo: 5
NOTA: Solo puedes ingresar nodos en un rango de [0 - 999]

Arista 1
Nodo Inicial: 4
Nodo Final: 1
Ingresa el valor de la arista: 16

Arista 2
Nodo Inicial: 1
Nodo Final: 2
Ingresa el valor de la arista: 45

Arista 3
Nodo Inicial: 2
Nodo Final: 3
Ingresa el valor de la arista: 45

Arista 4
Nodo Inicial: 3
Nodo Final: 4
Ingresa el valor de la arista: 78

Arista 5
Nodo Inicial: 4
Nodo Final: 7
Ingresa el valor de la arista: 9
```

Grafo ponderado no Dirigido

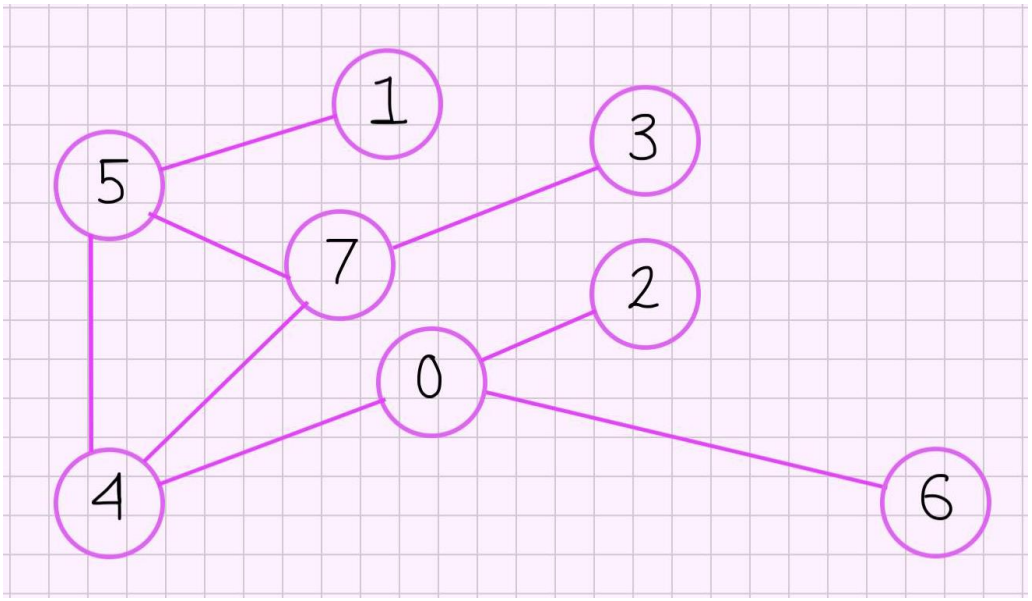
Submenú de opciones

Al terminar de llenar los datos correspondientes a el grafo que se eligió en el menú principal, se mostrará en pantalla el siguiente submenú para que el usuario decida que quiere hacer con su grafo

```
- - - - - - - - - - - - - - - - - - - - - - - -

                    SUBMENU

  1) Imprimir Grafo
  2) Recorrer Grafo (BFS)
  3) Recorrer Grafo (DFS)
  4) Algoritmo de Primm
  5) Salir

 -> Elige opción:
```

Submenú de opciones

Para probar cada una de las opciones que tiene este submenú, se baso en el siguiente grafo:



Grafo

Imprimir Grafo

```
. . . . . .

        Imprimir Grafo
Lista de adyacencia del vertice 0
 -> 4 -> 2 -> 6

Lista de adyacencia del vertice 1
 -> 5

Lista de adyacencia del vertice 2
 -> 0

Lista de adyacencia del vertice 3
 -> 7

Lista de adyacencia del vertice 4
 -> 5 -> 0 -> 7

Lista de adyacencia del vertice 5
 -> 1 -> 4 -> 7

Lista de adyacencia del vertice 6
 -> 0

Lista de adyacencia del vertice 7
 -> 5 -> 3 -> 4
```

Impresión para el grafo Dirigido y no Dirigido

```
. . . . . .

        Imprimir Grafo
Lista de adyacencia del vertice 0
 -> 4 -> 2 -> 6

Lista de adyacencia del vertice 1
 -> 5

Lista de adyacencia del vertice 2
 -> 0

Lista de adyacencia del vertice 3
 -> 7

Lista de adyacencia del vertice 4
 -> 5 -> 7 -> 0

Lista de adyacencia del vertice 5
 -> 1 -> 7 -> 4

Lista de adyacencia del vertice 6
 -> 0

Lista de adyacencia del vertice 7
 -> 5 -> 3 -> 4

.       .       .       .       .       .

        Nodos y el valor de sus aristas

   (1) --- 16 --- (5)
   (5) --- 17 --- (4)
   (4) --- 2 --- (0)
   (4) --- 21 --- (7)
   (0) --- 6 --- (6)
   (0) --- 8 --- (2)
   (7) --- 12 --- (3)
   (5) --- 15 --- (7)
```

Impresión para el grafo Ponderado Dirigido y No Dirigido

Recorrer Grafo BFS (Breadth First Search)

Para esta opción se mandó a llamar al método correspondiente **BFS** de la clase **Graph**, para probar dicho método, se inició el recorrido con tres nodos diferentes:

```
. . . . . .

        Recorrer Grafo (BFS)

¿Desde que vertice quieres iniciar el recorrido? 1

        Visitados: { 1  5  4  7  0  3  2  6 }
```

Empezando por el nodo 1

```
. . . . . .

        Recorrer Grafo (BFS)

¿Desde que vertice quieres iniciar el recorrido? 3

        Visitados: { 3  7  5  4  1  0  2  6 }
```

Empezando por el nodo 3

```
. . . . . .

        Recorrer Grafo (BFS)

¿Desde que vertice quieres iniciar el recorrido? 6

        Visitados: { 6  0  4  2  5  7  1  3 }
```

Empezando por el nodo 6

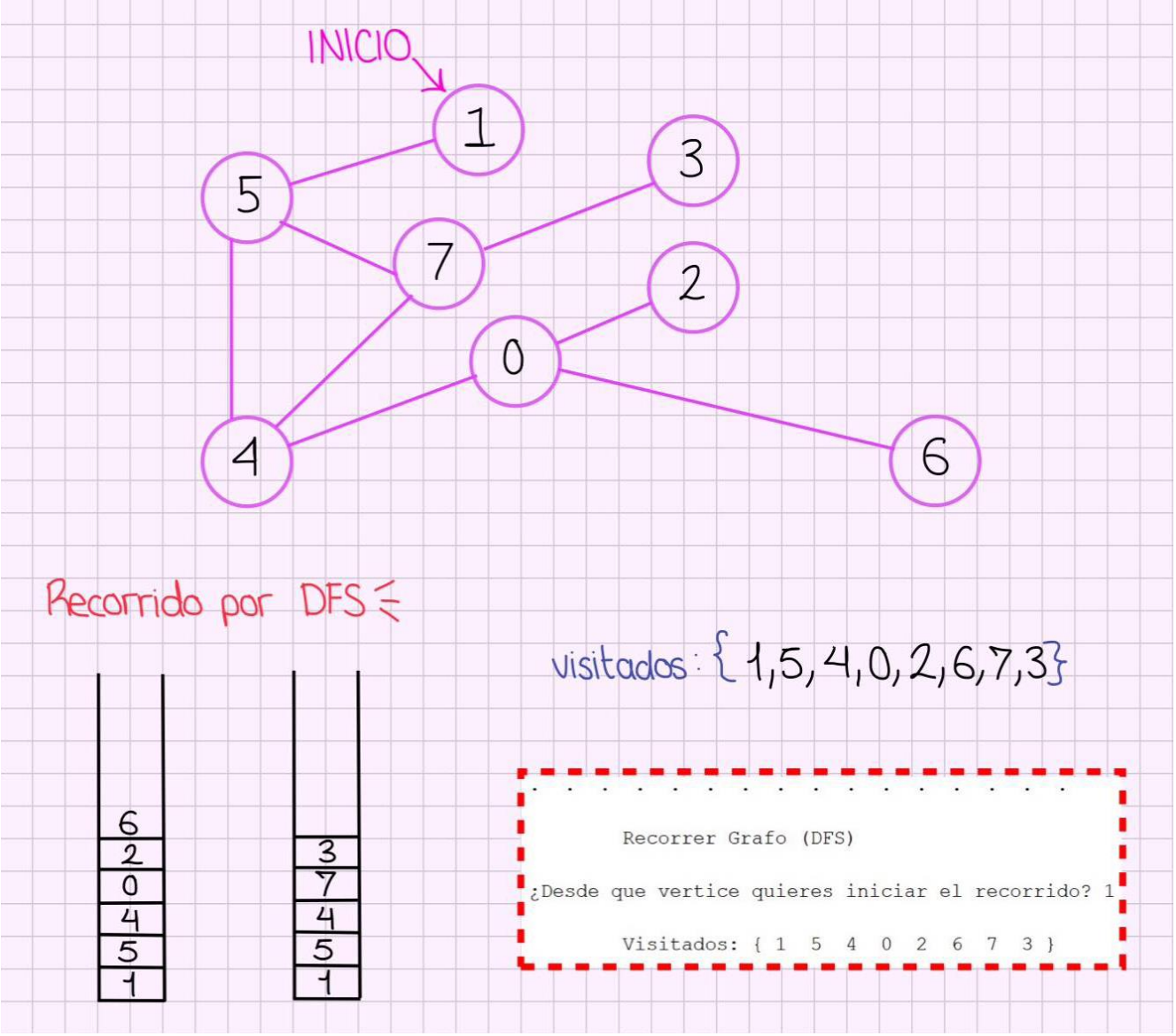
```
. . . . . .

        Recorrer Grafo (BFS)

¿Desde que vertice quieres iniciar el recorrido? 16

        No hay, no existe ese nodo ):
```

```
. . . . . .

        Recorrer Grafo (BFS)

¿Desde que vertice quieres iniciar el recorrido? 16

        No existe ese nodo o no va dirigido a otro ):
```

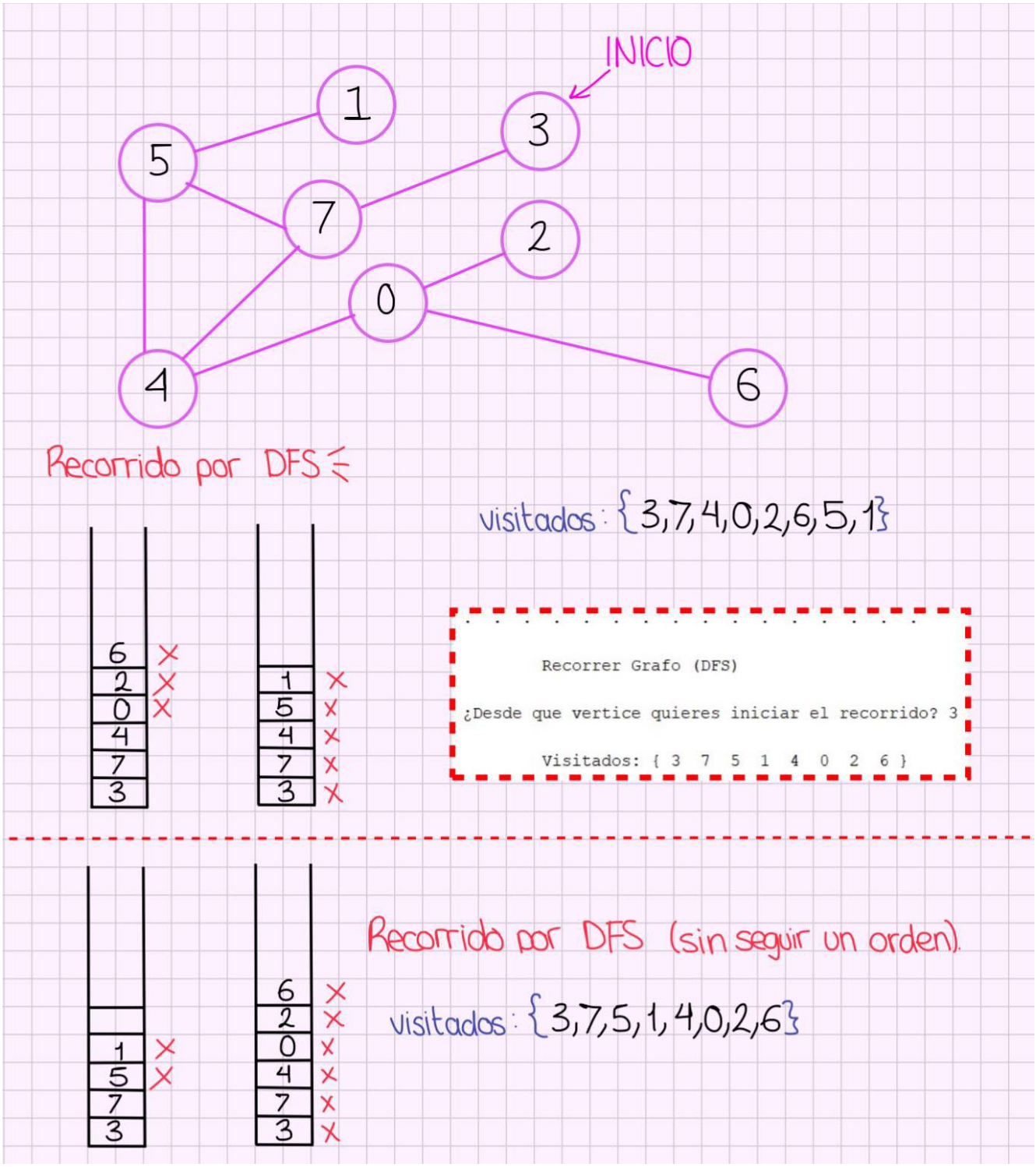
En caso de que el valor del nodo no exista o que no vaya dirigido a algún otro.

Recorrer Grafo DFS (Depth First Search)

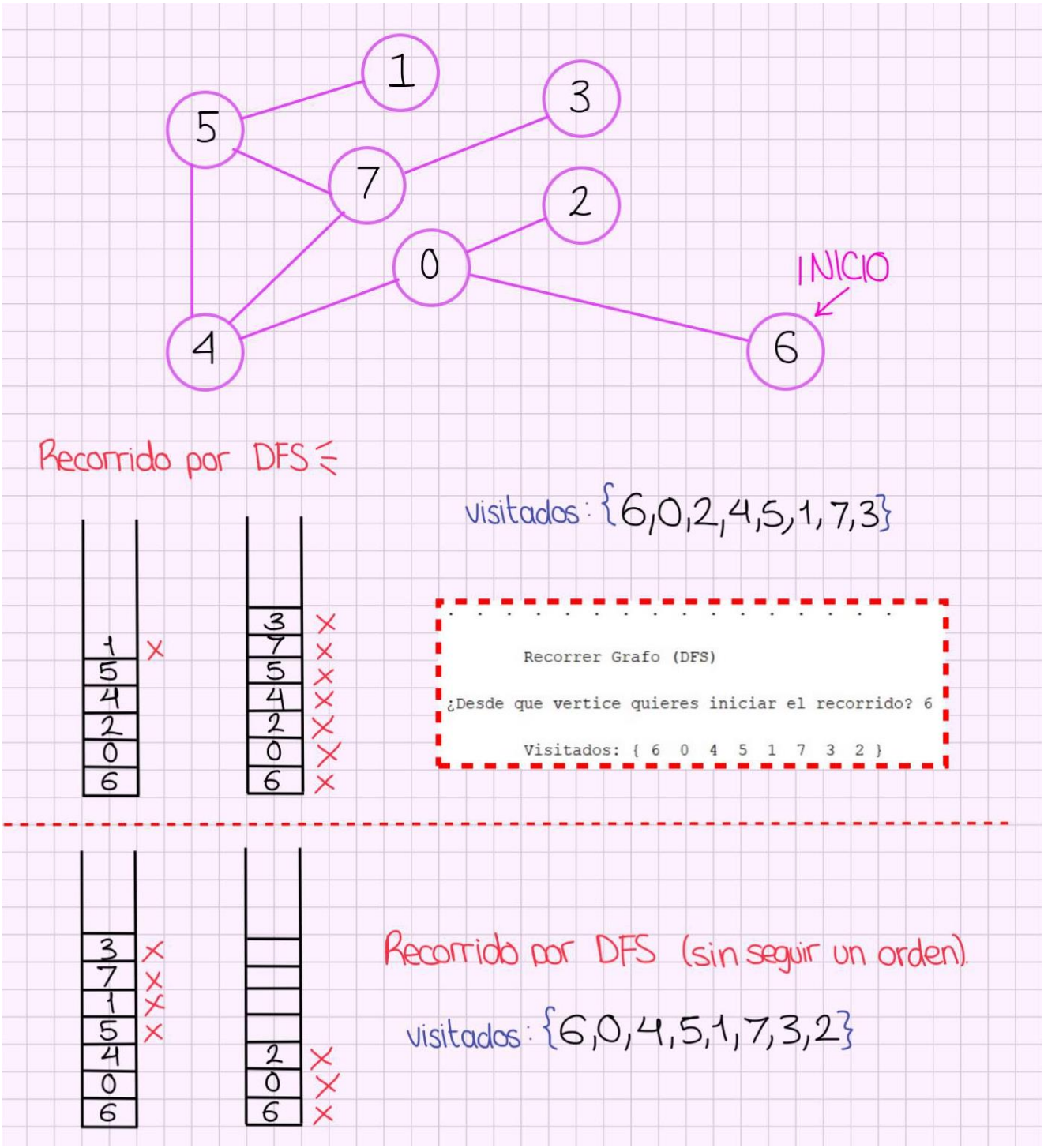
Para esta opción se mandó a llamar al método correspondiente **DFS** de la clase **Graph**, para probar dicho método, al igual que la opción anterior, se inició el recorrido con tres nodos diferentes; sin embargo, también se realizó la prueba de escritorio de cada uno de ellos:



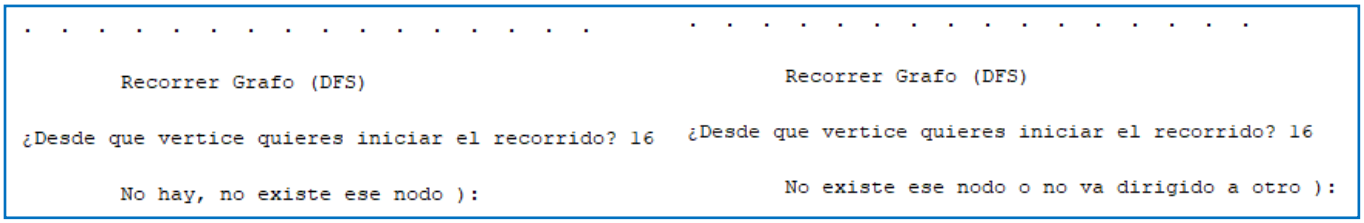
Empezando por el nodo 1



Empezando por el nodo 3



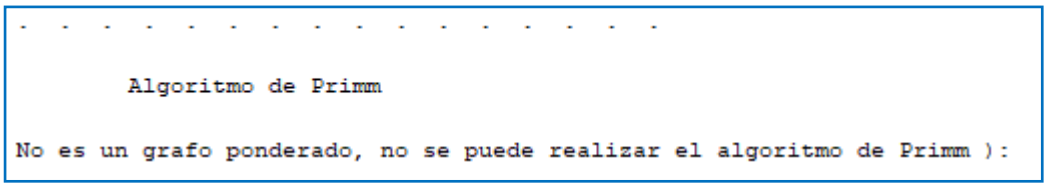
Empezando por el nodo 6



En caso de que el valor del nodo no exista o que no vaya dirigido a algún otro.

Algoritmo de Primm

Esta opción tiene como función principal, implementar el algoritmo de Primm en los nodos ponderados que se creen, en caso de ser un grafo dirigido o no dirigido, se imprimirá en pantalla lo siguiente:



Dirigido y no Dirigido

De lo contrario, en caso de ser un grafo Dirigido Ponderado o un grafo no Dirigido Ponderado, se realizará el algoritmo previamente descrito.

* Esta implementación no me quedo, ya que al utilizar tablas HashMap para almacenar el valor del nodo inicial, final y el valor de la arista, se me complicó mucho. Considero que si hubiera elegido otra forma de almacenar el valor de las aristas que tienen cada par de nodos, como por ejemplo alguna otra colección, me hubiese sido posible completar este ejercicio.*

Conclusiones

Considero que se cumplieron con los objetivos de esta práctica, ya que conocimos las diferentes formas de representar un grafo, así como los algoritmos de búsqueda por expansión y profundidad de cada uno de estos, tanto en la prueba de escritorio (en clase) como mediante la implementación de código (en práctica). Personalmente, se me hizo un poco complicado comprender los códigos dados para esta práctica, debido al uso de un arreglo con listas, pero al momento de ir escribiendo el código y al hacer pequeñas pruebas de escritorio para ver su funcionamiento me quedo claro como se iba a implementar cada grafo. También me costó un poco de trabajo encontrar la forma de como realizar los grafos ponderados, ya que al principio pensé hacerlos con una lista auxiliar pero quedaba todo revuelto, así que mejor decidí hacer una Tabla HashM donde se almacenaría el valor de cada uno de los nodos (inicial y final), así como el valor de cada una de las aristas que contiene el grafo para así posteriormente poder imprimir dichos valores.

Con respecto a los ejercicios que se presentaron para esta práctica, observe que hubo una diferencia muy notoria con estos y los ejercicios vistos en clase en el recorrido por BFS y DFS, ya que en clase hacíamos pruebas de escritorio siguiendo un determinado orden para que el grupo tuviera la misma respuesta; sin embargo, en estos ejercicios me di cuenta que no se sigue un orden como tal, pero da un resultado correcto.

Estos ejercicios se me hicieron adecuados para reforzar el tema visto en clase, al igual que para ver la implementación de los grafos en código. (: