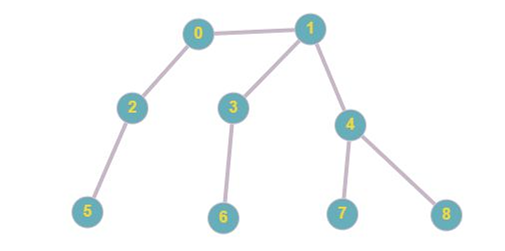
***Actividades de Clase 12***

**Alumnas:** Pierrot María Paula, Korniansky Eden, Prevosti Chiara y Chen Hui Jun.

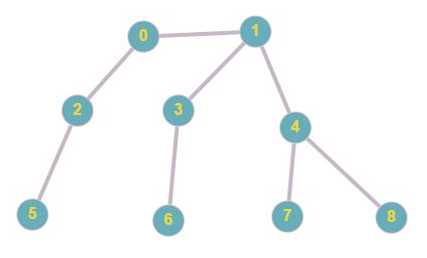
**Actividad 1**:*Dado el siguiente grafo, generar el recorrido DFS, a partir del nodo 0.*



Recorrido:

2,5,1,3,6,4,7,8.

**Actividad 2:** *Dado el siguiente grafo, generar el recorrido BFS, a partir del nodo 0.*



0,2,1,5,3,4,6,7,8.

**Actividad 3:**  *Desarrolla una aplicación que permita modelar una red de almacenes interconectados. El sistema debe permitir agregar almacenes, conectar almacenes entre sí mediante rutas directas, y realizar recorridos en profundidad (DFS) y en anchura (BFS) para explorar la red de distribución. Implementa una clase Almacén que represente un almacén. Cada almacén debe tener un identificador único y un nombre. Implementa una clase Grafo que gestione la red de almacenes. Esta clase debe: Permitir agregar almacenes al grafo. Permitir conectar almacenes entre sí (crear rutas directas entre almacenes). Implementar un método DFS para realizar un recorrido en profundidad desde un almacén de inicio. Implementar un método BFS para realizar un recorrido en anchura desde un almacén de inicio.El grafo debe estar representado utilizando una lista de adyacencia.*

import java.util.ArrayList;

import java.util.LinkedList;

public class Grafo {

ArrayList<Almacen> almacenes = new ArrayList<>();

// Constructor

public Grafo(ArrayList<Almacen> almacenes) {

this.almacenes = almacenes;

}

public void agregarAlmacen(Almacen almacen) {

almacenes.add(almacen);

}

private Almacen obtenerAlmacenPorId(int id) {

for (Almacen almacen : almacenes) {

if (almacen.getId() == id) {

return almacen;

}

}

return null;

}

// Implementación de BFS

public void BFS(int inicio) {

boolean[] visitado = new boolean[this.almacenes.size()]; // Array de visitados

LinkedList<Integer> cola = new LinkedList<>(); // Cola para BFS

visitado[inicio] = true; // Marcar el nodo de inicio como visitado

cola.add(inicio); // Añadirlo a la cola

while (!cola.isEmpty()) {

inicio = cola.poll(); // Extraer el primer nodo de la cola

System.*out*.print(inicio + " "); // Procesar el nodo

// Obtener todos los nodos adyacentes de "inicio"

Almacen almacenActual = obtenerAlmacenPorId(inicio);

if (almacenActual != null) {

for (Integer destino : almacenActual.getDestinos()) {

if (!visitado[destino]) {

visitado[destino] = true;

cola.add(destino);

}

}

}

}

}

// Método auxiliar de DFS que se llama recursivamente

private void DFSUtil(int v, boolean[] visitado) {

// Marcar el vértice actual como visitado y mostrarlo

visitado[v] = true;

System.*out*.print(v + " ");

Almacen almacenActual = this.obtenerAlmacenPorId(v);

// Recorrer todos los vértices adyacentes al vértice actual

if (almacenActual != null) {

for (Integer destino : almacenActual.getDestinos()) {

if (!visitado[destino]) {

DFSUtil(destino, visitado);

}

}

}

}

// Método principal DFS que llama a DFSUtil

public void DFS(int inicio) {

// Array de booleanos para marcar los vértices visitados

boolean[] visitado = new boolean[this.almacenes.size()];

// Llamar al método recursivo DFSUtil comenzando desde el vértice "inicio"

DFSUtil(inicio, visitado);

}

// Clase interna Almacen

public class Almacen {

private int id;

private String nombre;

private ArrayList<Integer> destinos = new ArrayList<>();

public Almacen(int id, String nombre) {

this.id = id;

this.nombre = nombre;

}

public void agregarDestino(Integer destino) {

destinos.add(destino);

}

public ArrayList<Integer> getDestinos() {

return destinos;

}

public int getId() {

return id;

}

public String getNombre() {

return nombre;

}

@Override

public String toString() {

return nombre;

}

}

// Método main para ejecutar el ejemplo

public static void main(String[] args) {

// Crear almacenes

Grafo grafo = new Grafo(new ArrayList<>());

Almacen almacen1 = grafo.new Almacen(0, "Almacen A");

Almacen almacen2 = grafo.new Almacen(1, "Almacen B");

Almacen almacen3 = grafo.new Almacen(2, "Almacen C");

Almacen almacen4 = grafo.new Almacen(3, "Almacen D");

// Definir los destinos de cada almacen

almacen1.agregarDestino(1); // Almacen A -> Almacen B

almacen1.agregarDestino(2); // Almacen A -> Almacen C

almacen2.agregarDestino(3); // Almacen B -> Almacen D

almacen3.agregarDestino(3); // Almacen C -> Almacen D

// Crear la lista de almacenes y el grafo

grafo.agregarAlmacen(almacen1);

grafo.agregarAlmacen(almacen2);

grafo.agregarAlmacen(almacen3);

grafo.agregarAlmacen(almacen4);

// Ejecutar BFS y DFS desde el nodo de inicio 0 (Almacen A)

System.*out*.println("Recorrido BFS comenzando desde Almacen A:");

grafo.BFS(0);

System.*out*.println("\n\nRecorrido DFS comenzando desde Almacen A:");

grafo.DFS(0);

}

}

**Actividad 4:** *Diseña e implementa un sistema que modele una red social utilizando grafos. Cada usuario será un nodo, y las amistades entre usuarios serán las aristas. El sistema debe permitir: Agregar usuarios a la red, cada uno con un identificador único y nombre. Conectar usuarios indicando que son amigos (relación bidireccional). Implementar un método para realizar un recorrido DFS (Depth First Search) desde un usuario dado, mostrando los usuarios alcanzables a través de sus amistades. Implementar un recorrido BFS (Breadth First Search) para explorar la red desde un usuario dado, siguiendo las conexiones de amistad.*

package Clase12;

import java.util.\*;

public class Clase12Actividad4 {

/\*

\* Diseña e implementa un sistema que modele una red social utilizando grafos.

\* Cada usuario será un nodo, y las amistades entre usuarios serán las aristas.

\* El sistema debe permitir: Agregar usuarios a la red,

\* cada uno con un identificador único y nombre.

\* Conectar usuarios indicando que son amigos (relación bidireccional).

\* Implementar un metodo para realizar un recorrido DFS (Depth First Search)

\* desde un usuario dado, mostrando los usuarios alcanzables a través de sus amistades.

\* Implementar un recorrido BFS (Breadth First Search) para

\* explorar la red desde un usuario dado,

\* siguiendo las conexiones de amistad.

\* \*/

public static class Grafo {

private Map<Usuario, List<Usuario>> adjList;

public Grafo() {

adjList = new HashMap<>();

}

// Agregar un vértice

public void agregarUsuario(Usuario usuario) {

adjList.putIfAbsent(usuario, new ArrayList<>());

//el putIfAbsent se fija si está o no. Si no está, lo agrega.

}

// Agregar una arista

public void agregarConexion(Usuario origen, Usuario destino) {

adjList.putIfAbsent(origen, new ArrayList<>());

adjList.putIfAbsent(destino, new ArrayList<>());

adjList.get(origen).add(destino);

adjList.get(destino).add(origen); // Por ser un grafo no dirigido

}

public void bfs(Usuario inicio) {

Set<Usuario> visitados = new HashSet<>();

Queue<Usuario> cola = new LinkedList<>();

cola.add(inicio);

visitados.add(inicio);

while (!cola.isEmpty()) {

Usuario usuario = cola.poll();

System.*out*.print(usuario.getNombre() + " ");

for (Usuario vecino : adjList.get(usuario)) {

if (!visitados.contains(vecino)) {

cola.add(vecino);

visitados.add(vecino);

}

}

}

System.*out*.println();

}

// DFS (Depth-First Search)

public void dfs(Usuario inicio) {

Set<Usuario> visitados = new HashSet<>();

Stack<Usuario> pila = new Stack<>();

pila.push(inicio);

visitados.add(inicio);

while (!pila.isEmpty()) {

Usuario usuario = pila.pop();

System.*out*.print(usuario.getNombre()+ " ");

for (Usuario vecino : adjList.get(usuario)) {

if (!visitados.contains(vecino)) {

pila.push(vecino);

visitados.add(vecino);

}

}

}

System.*out*.println();

}

public static class Usuario { //Este tipo de dato compone los nodos del grafo

private String nombre;

private int id;

private static int *cantUsuarios* = 0; //se utiliza para hacer los ids unicos

public Usuario(String nombre) {

this.nombre = nombre;

this.id = ++*cantUsuarios*;

//Crea un nuevo usuario.

}

public String getNombre() {

return nombre;

}

public int getId() {

return id;

}

}

public static void main(String[] args) {

Grafo grafo = new Grafo();

Usuario usuario1 = new Usuario("Paula");

Usuario usuario2 = new Usuario("Chiara");

Usuario usuario3 = new Usuario("Chen");

Usuario usuario4 = new Usuario("Eden");

grafo.agregarUsuario(usuario1);

grafo.agregarUsuario(usuario2);

grafo.agregarUsuario(usuario3);

grafo.agregarUsuario(usuario4);

grafo.agregarConexion(usuario1, usuario2);

grafo.agregarConexion(usuario1, usuario3);

grafo.agregarConexion(usuario2, usuario4);

//para testear, usamos el usuario 1 en ambas busquedas

System.*out*.println("BFS desde el usuario 1:");

grafo.bfs(usuario1);

System.*out*.println("DFS desde el usuario 1:");

grafo.dfs(usuario1);

}

}

}