UNIVERSIDAD AUTÓNOMA GABRIEL RENE MORENO
FACULTAD DE INGENIERÍA Y CIENCIAS DE LA COMPUTACIÓN Y TELECOMUNICACIONES

ESTRUCTURAS DE DATOS 2

CONTENIDO

RECORRIDOS SOBRE GRAFOS: BACKTRACKING.

PORCENTAJE TERMINADO : 100%.

GRUPO:

Grupo	Nombre	Registro
12	Jorge Choque Calle	219074240
14	Cristian Gabriel Gedalge Cayhuara	219022062

Fecha de Presentación: Martes ,02 de julio de 2024

COMENTARIO:
Con esta tarea aprendimos sobre como se manejan los grafos, en cuanto sus relaciones con otros nodos y la relación de sus arcos que tienen cada nodo.(la manera de entenderla bien es abstraerla como una lista con varios punteros).

CLASE NODO

```
public class Nodo {
    public String name;
    public Nodo prox;
    public Arco prim;
    public Arco ult;
    public int cantArcos;
    public Nodo (String name){
        this.name=name;
        prox=null;
        prim=ult=null;
        cantArcos=0;
    }
    public boolean vacia(){
        return cantArcos==0;
    }
    public void insertarUlt(Nodo p,int valor){
        if(vacia()){
            prim=ult=new Arco(p,valor);
        }else{
            ult=ult.prox=new Arco(p,valor);
        }
        cantArcos++;
    }

    public boolean seEncuentraArco(String namee)
    {
        Arco arc=prim;
        while(arc!=null)
        {
            if(arc.pDest.name==namee)
            {
                return true;
            }
            arc=arc.prox;
        }return false;
    }
}
```

CLASE ARCO

```
public class Arco {
    public Nodo pDest;
    public int valor;
    public Arco prox;
    public Arco(Nodo pDest,int valor){
        this.pDest=pDest;
        this.valor=valor;
    }


}
```

CLASE GRAFO

```
public class Grafo {

    public Nodo prim;
    public Nodo ult;
    public int cantNodos;

    public Grafo() {
        prim = ult = null;
        cantNodos = 0;
    }

    public Nodo buscar(String name) {
        Nodo p = prim;
        while (p != null) {
            if (p.name.equals(name)) {
                return p;
            }
            p = p.prox;
        }
        return null;
    }

    public void insertarNodo(String name) {
        if (!seEncuentra(name)) {
            insertarUlt(name);
        }
    }

    public void insertarUlt(String name) {
        if (vacia()) {
```

```
            prim = ult = new Nodo(name);

        } else {
            ult = ult.prox = new Nodo(name);
        }
        cantNodos++;
    }

    public boolean vacia() {
        return prim == null && ult == null;
    }

    public boolean seEncuentra(String name) {
        Nodo p = prim;
        while (p != null) {
            if (p.name.equals(name)) {
                return true;
            }
            p = p.prox;
        }
        return false;
    }


    public void insertarArco(String name1, String name2,
int valor) {
        Nodo pOrigen = buscar(name1);
        Nodo pDest = buscar(name2);
        if (pOrigen == null) {
            insertarNodo(name1);
            insertarArco(name1, name2, valor);
        }
        if (pDest == null) {
            insertarNodo(name2);
            insertarArco(name1, name2, valor);
        }
        if (pDest != null) {
            pOrigen.insertarUlt(pDest, valor);
        }
    }
```

```
//1. G1.mostrarCaminos(name1, name2) : Método que muestra
todos los caminos posibles desde el nodo name1 hasta
name2. Sugerencia, utilizar una lista actual de camino
recorrido.
    public void mostrarCaminos(String name1,String name2)
    {
        Nodo pOrigen=buscar(name1);
        Nodo pDest=buscar(name2);
        if(pOrigen==null || pDest==null)
            return ;
        LinkedList<Nodo> L1=new LinkedList<Nodo>();
        L1.add(pOrigen);
        caminos(L1,pOrigen,pDest);
    }

    public void caminos(LinkedList<Nodo> L1,Nodo
pOrigen,Nodo pDest){
        if(seEncuentra(L1,pOrigen))return;
        if(pOrigen==pDest){
            System.out.println(L1);
            return;
        }
        Arco p=pOrigen.prim;
        while(p!=null){
            L1.add(p.pDest);
            caminos(L1, p.pDest, pDest);
            L1.removeLast();
            p=p.prox;
        }
    }

    public boolean seEncuentra(LinkedList<Nodo>L1,Nodo p){
        int i=0;
        while(i<L1.size()-1){
            if(L1.get(i)==p)return true;
            i++;
        }
        return false;
    }
//2. G1.cantidadCaminos(name1, name2) : Método que
devuelve la cantidad de caminos que existen desde el nodo
name1 hasta name2.
    public int cantidadCaminos(String name1,String name2){
        Nodo pOrigen = buscar(name1);
    Nodo pDestino = buscar(name2);
```

```
    if (pOrigen == null || pDestino == null) {
        return 0;
    }

    int[] contador = {0};

    contarCaminos(pOrigen, pDestino, contador, new
LinkedList<Nodo>());
    return contador[0];
    }


    private void contarCaminos(Nodo nodoActual, Nodo
pDestino, int[] contador, LinkedList<Nodo> caminoActual) {

    caminoActual.add(nodoActual);
    if (nodoActual == pDestino) {
        contador[0]++;
    } else {

        Arco arco = nodoActual.prim;
        while (arco != null) {
            contarCaminos(arco.pDest, pDestino, contador,
caminoActual);
            arco = arco.prox;
        }
    }

    caminoActual.removeLast();
}
//3. G1.mostrarTotalCamino(name1, name2) : Método que
muestra todos los caminos desde name1 a name2 con su
costos totales de recorrido desde el origen al destino.
    public void mostrarTotalCamino(String name1, String
name2) {
    Nodo pOrigen = buscar(name1);
    Nodo pDestino = buscar(name2);

    if (pOrigen == null || pDestino == null) {
        System.out.println("Alguno de los nodos no existe
en el grafo.");
        return;
    }

    mostrarCaminosRecursivo(pOrigen, pDestino, new
LinkedList<Nodo>(), 0);
```

```
}
    private void mostrarCaminosRecursivo(Nodo nodoActual,
Nodo pDestino, LinkedList<Nodo> caminoActual, int
costoTotal) {

    caminoActual.add(nodoActual);

    if (nodoActual == pDestino) {
        System.out.print("Camino encontrado: ");
        for (int i = 0; i < caminoActual.size(); i++) {
            System.out.print(caminoActual.get(i).name);
            if (i < caminoActual.size() - 1) {
                System.out.print(" -> ");
            }
        }
        System.out.println(" Costo total: " + costoTotal);
    } else {
        Arco arco = nodoActual.prim;
        while (arco != null) {
            mostrarCaminosRecursivo(arco.pDest, pDestino,
caminoActual, costoTotal + arco.valor);
            arco = arco.prox;
        }
    }

    caminoActual.removeLast();
}
//4. G1.otrosMetodos() : Implementar otros métodos
interesantes adicionales, sobre recorridos en el grafo G1.

    public String mostrarelem(String name1,String name2){
        Arco pOrigen=buscar(name1).prim;
        Nodo pDest=buscar(name2);
        String s="";

        while(pOrigen!=null)
        {
            if(pOrigen.pDest.name==name2)
            {
                s=pOrigen.valor+" - ";
            }
            pOrigen=pOrigen.prox;
        }
        return s;

    }
```

```
    public boolean contieneCiclo() {
    Set<Nodo> visitados = new HashSet<>();
    Set<Nodo> enPila = new HashSet<>();
    for (Nodo nodo : obtenerNodos()) {
        if (!visitados.contains(nodo)) {
            if (contieneCicloAux(nodo, visitados, enPila))
{
                return true;
            }
        }
    }
    return false;
}

private boolean contieneCicloAux(Nodo nodo, Set<Nodo>
visitados, Set<Nodo> enPila) {
    visitados.add(nodo);
    enPila.add(nodo);
    Arco arco = nodo.prim;
    while (arco != null) {
        if (!visitados.contains(arco.pDest)) {
            if (contieneCicloAux(arco.pDest, visitados,
enPila)) {
                return true;
            }
        } else if (enPila.contains(arco.pDest)) {
            return true;
        }
        arco = arco.prox;
    }
    enPila.remove(nodo);
    return false;
}

}
```