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

ESTRUCTURAS DE DATOS 2

CONTENIDO

TAREA-1. OPERACIONES BÁSICAS SOBRE GRAFOS

PORCENTAJE TERMINADO : 100%.

GRUPO:

Grupo	Nombre	Registro
14	Cristian Gabriel Gedalge Cayhuara	219022062

Fecha de Presentación: Lunes ,01 de julio de 2024

COMENTARIO:

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

CLASS 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++;
    }
}
```

CLASS 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;
    }


}
```

CLASS 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;
    }



//1. G1.insertarNodo(name) : Método que insertar un nodo
en el grafo G1, con rótulo name. Sugerencia, insertar al
último de la Lista de Nodos.
    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;
    }

//2. G1.insertarArco(name1, name2, valor): Método que
inserta un arco en el grafo G1, desde el nodo name1 hasta
name2 con un peso del arco igual a valor. Sugerencia,
insertar al último de la Lista de Arcos que sales de
name1.
    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);
        }
    }
//3. G1.mostrar(): Método que muestra el grafo G. Muestra
la lista de nodos. Para cada nodo,
//    muestra la lista de arcos que salen de él, con sus
nodos destinos y sus respectivos valores.

    public void mostrar() {
        Nodo p = prim;
        while (p != null) {
            System.out.println(p.name);
            Arco arco = p.prim;
            while (arco != null) {
                System.out.print(arco.pDest.name + "," +
arco.valor);
                arco = arco.prox;
            }
        }
        System.out.println();
    }
//4. G1.cantidadArcos() : Método que devuelve la cantidad
de arcos que contiene el grafo G1.

    public int cantidadArcos() {
```

```
        Nodo p = prim;
        int suma = 0;
        while (p != null) {
            suma = suma + p.cantArcos;
            p = p.prox;
        }
        return suma;
    }
//5. G1.cantidadLlegadas(name1) : Método que devuelve la
cantidad de arcos que llegan al nodo name1.

    public int cantidadLlegadas(String name1) {
        Nodo p = prim;
        int suma = 0;
        while (p != null) {
            Arco arco=p.prim;
            while(arco!=null){
                if(arco.pDest.name.equals(name1))
                    suma++;
                arco=arco.prox;
            }
            p = p.prox;
        }
        return suma;
    }
//6. G1.cantidadSalidas(name1) : Método que devuelve la
cantidad de arcos que salen al nodo name1

    public int cantidadSalidas(String name1) {
        Nodo p = prim;
        while (p != null) {
            if (p.name.equals(name1)) {
                return p.cantArcos;
            }
            p = p.prox;
        }
        return 0;
    }
//7. G1.mostrarNodosBucle(): Método que muestra los nodos
que tienen arcos así mismos.

    public void mostrarNodoBucle() {
        Nodo p = prim;
        while (p != null) {
            Arco arco = p.prim;
            while (arco != null) {
```

```
                if (arco.pDest == p) {
                    System.out.println("Nodo con bucle" +
p.name);
                }
                arco=arco.prox;
            }
            p = p.prox;
        }
    }
//8. G1.mostrarNodosIslas() : Método que muestra los nodos
islas. Nodos islas, son aquellos nodos que no tienen arcos
que salen de él, ni arcos que llegan a él.

    public void mostrarNodosIslas() {
        Nodo p = prim;
        while (p != null) {
            if (cantidadLlegadas(p.name) == 0 &&
cantidadLlegadas(p.name) == 0) {
                System.out.println("Nodo Isla: " +
p.name);
            }
            p = p.prox;
        }
    }
//9. G1.mismosNodos(G2) : Método lógico que devuelve True,
si los grafos G1 y G2 tienen los mismos nodos.

    public boolean mismoNodo(Grafo G2) {
        Nodo p1 = prim;
        Nodo p2 = G2.prim;

        while (p1 != null && p2 != null) {
            if (p1.name.equals(p2.name)) {
                return false;
            }
            p1 = p1.prox;
            p2 = p2.prox;
        }
        return true;
    }

//10. G1.mayorValor() : Método que devuelve el mayor valor
de los arcos del grafo G1.
    public int mayorValor() {
        Nodo p = this.prim;
        int mayor = 0;
```

```
        while (p != null) {
            Arco arco = p.prim;
            while (arco != null) {
                if (arco.valor > mayor) {
                    mayor = arco.valor;
                }
                arco = arco.prox;
            }
            p=p.prox;
        }
        return mayor;
    }
//11. G1.cantidadIdaVuelta() : Método que devuelve la
cantidad de parejas de
//nodos del grafo G1,  unidos por arcos de ida y vuelta
inmediatas. Es decir; caminos directos.
/////verificar

    public int cantidadIdaVuelta() {
        Nodo p = this.prim;
        int cant = 0;
        while (p != null) {
            Arco arco = p.prim;
            while (arco != null) {
                if ( arco.pDest.prim!=null
&&arco.pDest.prim.pDest.name==p.name) {
                    cant++;
                }
                arco = arco.prox;
            }
            p = p.prox;
        }
        return cant;
    }

//12. G1.mostrarParalelos(): Método que muestra los nodos
con sus arcos paralelos. Arcos paralelos son aquellos que
tiene el mismo origen y destino inmediato.
    public void mostrarParalelos() {
        Nodo p = prim;
        int may = 0;
        while (p != null) {
            Nodo q = p;
            while (q != null) {
                int m = mismoDestino(p.prim, q.name);
                if (m > 1) {
```

```
                    System.out.println(p.name + "->" +
q.name);
                }
                q = q.prox;

            }
            p = p.prox;
        }
    }

    public int mismoDestino(Arco p, String destino) {
        int cant = 0;
        while (p != null) {
            if (p.pDest.name.equals(destino)) {
                cant++;
            }
            p = p.prox;
        }
        return cant;
    }

//13. G1.nodosVecinos(node1, node2) : Método lógico que
devuelve True, si los nodos node1 y node2 son vecino.
Tienen un arco directo que les une.
    public boolean nodosVecinos(String name1, String
name2) {
        Nodo p = buscar(name1);
        Nodo p2 = buscar(name2);
        while (p != null) {
            Arco arco = p.prim;
            while (arco != null) {
                if (arco.pDest.name == name2) {
                    return true;
                }
            }
        }

        while (p2 != null) {
            Arco arco2 = p2.prim;
            while (arco2 != null) {
                if (arco2.pDest.name == name1) {
                    return true;
                }
            }
        }
        return false;
```

```
    }

//14. G1.cantidadVecinos(node1) : Método que devuelve la
cantidad de vecino del nodo node1.
    public int cantidadVecinos(String node1) {
        Nodo p = prim;
        int cant = 0;
        while (p != null) {
            if (nodosVecinos(node1, p.name)) {
                cant++;
            }
            p = p.prox;
        }
        return cant;
    }

//15. G1.regular() : Método lógico que devuelve True, si
G1 es un grafo regular. Es regular si todos los nodos
tienen la misma cantidad de vecinos.
    public boolean regular() {
        Nodo p = prim;
        int numVecinos = cantidadVecinos(p.name);
        while (p != null) {
            if (cantidadVecinos(p.name) != numVecinos) {
                return false;
            }
            p = p.prox;
        }
        return false;
    }

//16. G1.subGrafo(G2) : Método lógico que devuelve True,
si G1 es subgrafo de G2.
    public boolean subGrafo(Grafo G2) {
        Nodo p = prim;
        Nodo q = G2.buscar(p.name);
        return mismoNodo(p, q);
    }

    public boolean mismoNodo(Nodo p, Nodo q) {
        while (p != null && q != null) {
            Arco arcp = p.prim;
            Arco arcq = q.prim;
            while (arcp != null && arcq != null) {
                if (!(arcp.valor == arcq.valor &&
arcp.pDest.name == arcq.pDest.name)) {
```

```
                    return false;
                }
                arcp = arcp.prox;
                arcq = arcq.prox;
            }
            if (arcp != null) {
                return false;
            }
            p = p.prox;
            q = q.prox;
        }
        return p == null;
    }

//17. G1.eliminarNodo(name) : Método que elimina el nodo
que  contiene la etiqueta name.
    public void eliminarNodo(String name) {
        Nodo p = prim;
        Nodo q = prim;
        if (name == prim.name) {
            prim = p.prox;
        }
        if (prim.name == name && ult.name == name) {
            prim = ult = null;
        }
        while (p != null && name != p.name) {
            q = p;
            p = p.prox;
        }
        if (p.name == ult.name) {
            ult = q;
        }

        q.prox = p.prox;
        limpiar(name);
    }

    public void limpiar(String name) {
        Nodo p = prim;
        while (p != null) {
            Arco arc = p.prim;
            Arco arcCopy = p.prim;
            if (arc != null) {
                if (arc.pDest.name == name) {
                    p.prim = arc.prox;
                }
```

```
//             if (name == p.ult.pDest.name) {
//                 p.ult = p.ult.prox;
//             }
                while (arcCopy != null &&
arcCopy.pDest.name == name) {
                    arc = arcCopy;
                    arcCopy = arcCopy.prox;
                }
                arc.prox = arcCopy.prox;
            }
            p = p.prox;
        }
    }

//18. Implementar al menos 3 métodos adicionales
interesantes cualesquiera, sobre Grafos dirigidos.
    //1 se encuentra el node name
     public boolean seEncuentra(String name) {
        Nodo p = prim;
        while (p != null) {
            if (p.name.equals(name)) {
                return true;
            }
            p = p.prox;
        }
        return false;
    }


    //2 retorna que hay entre name1(Salida) name
2(LLegada)
     public String valorEntre(String name1,String name2)
     {
         Arco arco=buscar(name1).prim;
         while(arco!=null)
         {
             if(arco.pDest.name==name2)
                 return arco.valor+"";
            arco=arco.prox;
         }
         return "null";
     }

    //3. sumalaLista de elem que sale de name1 no importa
a donde lleguen
```

```
    public  int suma(String name1)
    {
        int suma=0;
         Arco arco=buscar(name1).prim;
         while(arco!=null)
         {
             suma=suma+arco.valor;
             arco=arco.prox;
         }
         return suma;
    }
    public static void main(String[] args) {
        Grafo G1 = new Grafo();
        G1.insertarNodo("A");
        G1.insertarNodo("B");
        G1.insertarNodo("C");
        G1.insertarNodo("D");
        G1.insertarNodo("E");
        G1.insertarNodo("F");

        G1.insertarArco("A", "F", 3);
        G1.insertarArco("E", "A", 5);
        G1.insertarArco("B", "D", 10);
        G1.insertarArco("C", "D", 6);
        G1.insertarArco("C", "E", 20);
        G1.eliminarNodo("A");

//
System.out.println(G1.prim.prox.prox.prox.prox.prox.name);
//        System.out.println(G1.buscar("E").prim);

    }

}
```