UNIVERSIDAD AUTÓNOMA GABRIEL RENÉ MORENO

FACULTAD DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN Y TELECOMUNICACIONES

TAREA Nro. 1

CONTENIDO: Operaciones básica sobre grafos
TERMINADO: 100%
Trabajo Individual

Integrantes	Registro
Vino Apaza Vanesa	220053243

Comentario: Con esta tarea aprendí sobre como se manejan los grafos, en cuanto sus relaciones cono otros nodos y la relación de sus arcos que tienen cada nodo.

Fecha de presentación: 17 de diciembre
Fecha Presentada: 16 de diciembre

Santa Cruz – Bolivia
2023

CÓDIGO GRAFOS

Class Nodo

```
public class Nodo {
    public String nombre;
    public int cantArcos;
    public Nodo prox;
    public Arco prim;
    public Arco ult;

    public Nodo(String nombre, Nodo prox) {
        this.nombre = nombre;
        this.prox = prox;
        this.prim=this.ult=null;
        this.cantArcos=0;
    }
    public String toString(){
        String s="";
        return s;
    }
    public boolean vacia(){
        return cantArcos==0;
    }
    public void insertArco(int valor, Nodo desti,Arco proxi){
        if(vacia())
            prim = ult = new Arco(valor, desti, proxi);
        else
            ult.prox=ult=new Arco(valor,desti,proxi);
        cantArcos++;
    }
}
```

Class Arco

```
public class Arco {
    public int paso;
    public Arco prox;
    public Nodo dest;

    public Arco(int paso, Nodo dest,Arco prox) {
        this.paso = paso;
        this.prox = prox;
        this.dest = dest;
    }
    public Arco(Arco prox, Nodo dest) {
        this.prox = prox;
        this.dest = dest;
    }
    public String toString(){
        String s="";
        return s;
    }
}
```

Class Grafo

```
public class Grafo {
   public Nodo prim;
   public Nodo ult;
   public int cantNodos;

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

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

//metodos de la clase
}
```

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))
            return;
      if(vacia())
         prim=ult=new Nodo(name,null);
      else
         ult.prox=ult=new Nodo(name,null);
      cantNodos++;
   }
```

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 origin=buscarNodo(name1);
    Nodo desti=buscarNodo(name2);
    if(origin==null ||desti==null)
        return;
    origin.insertArco(valor, desti, null);
}
```

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.nombre+": ");
       Arco arco=p.prim;
        while(arco!=null){
           System.out.println(" ("+arco.dest.nombre+", "+arco.paso+") ");
           arco=arco.prox;
        }
        System.out.println();
        p=p.prox;
    }
}
```

4. **G1.cantidadArcos() :** Método que devuelve la cantidad de arcos que contiene el grafo G1.

```
public int cantidadArcos(){
    int cantArcos=0;
    Nodo p=prim;
    while(p!=null){
        cantArcos+=p.cantArcos;
        p=p.prox;
    }
    return cantArcos;
}
```

5. **G1.cantidadLlegadas(name1) :** Método que devuelve la cantidad de arcos que llegan al nodo name1.

```
public int cantidadLlegadas(String name1){
    int cantLlegadas=0;
    Nodo p=prim;
    while(p != null){
        Arco arco=p.prim;
        while(arco!=null){
            if(arco.dest.nombre.equals(name1))
                cantLlegadas++;
            arco=arco.prox;
        }
        p=p.prox;
    }
    return cantLlegadas;
}
```

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.nombre.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 mostrarNodosBucle(){
    Nodo p=prim;
    while(p!=null){
        Arco arco=p.prim;
        while(arco!=null){
            if(arco.dest==p){
                System.out.println("Nodo con buble: "+p.nombre+",");
                //break;
            }
            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.nombre)==0 && cantidadSalidas(p.nombre)==0)
            System.out.println("Nodos Isla :"+p.nombre);
        p=p.prox;
    }
}
```

9. **G1.mismosNodos(G2) :** Método lógico que devuelve True, si los grafos G1 y G2 tienen los mismos nodos.

```
public String stringGrafo(){
    Nodo p=prim;  String cad="";
    while(p!=null){
       cad+=p.nombre;
       p=p.prox;
    }
    return cad;
}
public boolean mismoNodo(Grafo G2){
    String graf1=stringGrafo();
    String graf2=G2.stringGrafo();
    return graf1.equals(graf2);
}
```

10. **G1.mayorValor() :** Método que devuelve el mayor valor de los arcos del grafo G1.

```
public int mayorValor(){
    int mayor=0;
    Nodo p=prim;
    while(p!=null){
        Arco arco=p.prim;
        while(arco !=null){
            if(arco.paso>mayor)
                mayor=arco.paso;
            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.

```
public int cantidadIdaVuelta(){
    int cantIda=0;
    Nodo p = prim;
    while(p !=null){
        Arco arco = p.prim;
        while(arco != null){
            if(arco.dest.prim != null && arco.dest.prim.dest == p)
                cantIda++;
            arco = arco.prox;
        }
        p = p.prox;
    }
    return cantIda;
}
```

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.nombre);
            if(m>1)
                System.out.println(p.nombre+" apunta paralelamente: "+q.nombre);
            q=q.prox;
        }
        p=p.prox;
    }
}
public int mismoDestino(Arco p, String destino){
    int cant=0;
    while(p!=null){
        if(p.dest.nombre.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 nom, String nam){
    Nodo p=buscarNodo(nom);
    Nodo q=buscarNodo(nam);

    Arco arco=p.prim;
    while(arco!=null){
        if(arco.dest.nombre.equals(nam))
            return true;
        arco=arco.prox;
    }

    Arco arc=q.prim;
    while(arc!=null){
        if(arc.dest.nombre.equals(nom))
            return true;
        arc=arc.prox;
    }
   return false;
 }
```

14. **G1.cantidadVecinos(node1) :** Método que devuelve la cantidad de vecino del nodo node1.

```
   public int cantVecinos(String nom){
       int cant=0;
       System.out.println("");
       Nodo q=prim;
       while(q!=null){
           if(nodosVecinos(nom, q.nombre))
               cant++;
           q=q.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 reg=cantidadVecinos(p.nombre);
       while(p!=null){
           if(cantidadVecinos(p.nombre)!=reg)
               return false;
```

```
            p=p.prox;
        }
        return true;
    }
```

16. **G1.subGrafo(G2) :** Método lógico que devuelve True, si G1 es subgrafo de G2

```
public boolean subGrafo(Grafo G2){
    Nodo p=prim;
    while(p!=null){
        Nodo q=G2.buscarNodo(p.nombre);
        if(!p.nombre.equals(q.nombre))
            return false;
        if(!mismosArcos(p,q))
            return false;
        p=p.prox;
    }
    return true;
}

   public boolean mismosArcos(Nodo p, Nodo q){
       if(p.cantArcos!=q.cantArcos)
           return false;

       Arco arco=p.prim;
       Arco arc=q.prim;
       while(arco!=null){
           if(!arco.dest.nombre.equals(arc.dest.nombre) && arco.paso!=arc.paso)
               return false;
           arco=arco.prox;
       }
       return true;
   }
```