Análisis de Complejidad del grafo:

**Cola.java**  
public class Cola {   
 private NodoDJ cabeza; 1  
 public Cola(){  
 cabeza = null; 1  
 }  
  
 public NodoDJ getCabeza(){  
 return cabeza; 1  
 }

public void push(int x, int p){ 2  
 NodoDJ nuevo = new NodoDJ(x, p); 1  
 if(cabeza == null){  
 cabeza = nuevo; 1  
 return; 1  
 }  
 NodoDJ temp = cabeza; 2  
 NodoDJ ante = null; 2  
  
 while(temp != null && temp.getDistancia() < p){ n  
 ante = temp; 1  
 temp = temp.getNext(); 1  
 }  
  
 if(temp == null){  
 ante.setNext(nuevo); 1  
 } else {   
 if(ante == null){  
 nuevo.setNext(cabeza); 1  
 cabeza = nuevo; 1  
 } else {   
 nuevo.setNext(temp); 1  
 ante.setNext(nuevo); 1  
 }  
 }  
 }  
  
 public NodoDJ pop(){  
 if(isEmpty()){  
 return null; 1  
 }   
 NodoDJ temp = cabeza; 2  
 cabeza = cabeza.getNext(); 1  
 if(isEmpty()){  
 cabeza=null; 1  
 }  
 return temp; 1  
 }  
  
 public boolean isEmpty(){  
 return (cabeza) == null; 1  
 }  
  
 public void print(){  
 NodoDJ temp = cabeza; 2  
 while(temp != null){ n  
 System.out.println(temp.getVertice()+" , "+temp.getDistancia()); 1  
 temp = temp.getNext(); 1  
 }  
 }  
  
}

T(n) = 2n + 30

**NodoDJ.kava**  
  
public class NodoDJ implements Serializable {  
 private int vertice, distancia; 2  
 private NodoDJ next; 1  
 public NodoDJ(int v, int d){ 2  
 vertice = v; 1  
 distancia = d; 1  
 next = null; 1  
 }  
 public int getVertice(){  
 return vertice; 1  
 }  
 public int getDistancia(){  
 return distancia; 1  
 }  
 public NodoDJ getNext(){  
 return next; 1  
 }  
 public void setNext(NodoDJ nx){  
 next = nx; 1  
 }  
}

T(n) = 12

**Dijkstra.java**  
  
public class Dijkstra {  
  
 public static int[] dijkstra(int tam, ArrayList<Lista> grafo, int org) throws CloneNotSupportedException { 4  
 int[] dist = new int[tam]; 2  
 for(int i=0; i<tam; i++){ n + 5  
 dist[i] = Integer.MAX\_VALUE; 1  
 }  
 dist[org] = 0; 1  
  
 Cola prioridad = new Cola(); 1  
 prioridad.push(org, 0); n

while(!prioridad.isEmpty()){ n  
 NodoDJ temp = prioridad.pop(); 2  
 for(int i=0; i<grafo.get(temp.getVertice()).getCont(); i++){ n + 5  
 NodoDJ n = grafo.get(temp.getVertice()).clone().getPos(i); 2  
 if(dist[temp.getVertice()] + n.getDistancia() < dist[n.getVertice()]){  
 dist[n.getVertice()] = n.getDistancia() + dist[temp.getVertice()]; 1  
 prioridad.push(n.getVertice(), dist[n.getVertice()]); n  
 }  
 }  
 }   
 return dist; 1  
  
 }  
}

T(n) = 3n + 25

**Grafo.java**  
  
public class Grafo implements Serializable{  
 int cant; 1  
  
 ArrayList<Lista> grafo; 1  
 int origen; 1  
  
 String[] nombres; 1  
  
 public Grafo(int cantidad, int org){ 2  
 cant = cantidad; 1  
 grafo = new ArrayList<>(); 1  
 for(int i=0; i<cant; i++) { n + 5  
 grafo.add(new Lista()); 1  
 }  
 origen = org; 1  
 nombres = new String[cantidad]; 2  
 Arrays.fill(nombres, ""); 1  
 }

T(n) = n + 18  
  
 public Grafo(){}  
  
 public void insert(int org, int v, int d) throws WrongInputException { 3  
 if(org > cant-1 || org < 0 || v > cant-1 || v < 0){  
 throw new WrongInputException("Formato incorrecto!"); 1  
 }  
 grafo.get(org).addFinal(v, d); n  
 }  
T(n) = n + 4

public void caminoMasCorto(JTextPane console) throws CloneNotSupportedException {  
 int[] dist = Dijkstra.dijkstra(cant, grafo, origen); 4  
 System.out.println("Vertice\tDistancia desde origen"); 1  
 console.setText("Vertice\tDistancia desde origen"); 1  
 for(int i=0; i<cant; i++){ n + 5  
 System.out.println(i + "\t" + dist[i]); 2  
 console.setText(console.getText() + "\n" + i + "\t\t" + dist[i]); 2  
 }  
 }  
T(n) = n + 15

public void caminoMasCortoNombres(JTextPane console) throws CloneNotSupportedException {  
 int[] dist = Dijkstra.dijkstra(cant, grafo, origen); 4  
 System.out.println("Vertice\tDistancia desde origen"); 1  
 console.setText("Vertice\tDistancia desde origen"); 1  
 for(int i=0; i<cant; i++){ n + 5  
 System.out.println(traducirOUT(i) + "\t" + dist[i]);  
 console.setText(console.getText() + "\n" + traducirOUT(i) + "\t\t" + dist[i]); 3  
 }  
 }  
  
 public int getCant() {  
 return cant; 1  
 }  
  
 public int getOrigen() {  
 return origen; 1  
 }  
  
 public ArrayList<Lista> getGrafo() {  
 return grafo; 1  
 }  
  
 private static final long SerialVersionUID = 10L; 2

T(n) = n + 19

public static void guardar(Grafo g) throws IOException {  
 ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("grafo.txt"));  
 out.writeObject(g); 2  
 out.close(); 1  
 java.awt.Toolkit.getDefaultToolkit().beep(); 1  
 JOptionPane.showConfirmDialog(null, "El grafo fue exportado",  
 "Contactos", JOptionPane.DEFAULT\_OPTION); 1  
 }

T(n) = 5  
  
 public static Grafo cargar() throws IOException, ClassNotFoundException {  
 ObjectInputStream in = new ObjectInputStream(new FileInputStream("grafo.txt"));  
 Grafo g = (Grafo)in.readObject(); 3  
 in.close(); 1  
 java.awt.Toolkit.getDefaultToolkit().beep();  
 JOptionPane.showConfirmDialog(null, "El grafo fue importado",  
 "Contactos", JOptionPane.DEFAULT\_OPTION); 1  
 return g; 1  
 }

T(n) = 6

public void nombrar(String nm){ 1  
 if(!nombres[nombres.length-1].equals("")){  
 System.out.println("Arreglo lleno");  
 return; 1  
 }  
 for(int i=0; i<nombres.length; i++){ n + 5  
 if(nombres[i].equals("")){  
 nombres[i] = nm; 1  
 return; 1  
 }  
 }  
 }  
T(n) = n + 9

public int traducirIN(String nm){ 1  
 int i=0; 2  
 for(String n : nombres){ n + 2  
 if(nm.equals(n)){  
 return i; 1  
 }  
 i++; 2  
 }  
 return -1; 1  
 }  
T(n) = n + 9

public String traducirOUT(int nm){ 1  
 return nombres[nm]; 1  
 }  
  
 public void mostrar(JTextPane console){ 1  
 if(cant==0 || grafo==null){return;}  
 console.setText(""); 1  
 console.setText(console.getText() + 0 + " -> | " + grafo.get(0).print(true, this)); 2  
 for(int i=1; i<cant; i++){ n + 5  
 console.setText(console.getText() + "\n" + i + " -> | " + grafo.get(i).print(true, this)); 3  
 }  
 }

T(n) = n + 14  
  
 public void mostrarN(JTextPane console){ 4  
 if(cant==0 || grafo==null){return;}  
 console.setText(""); 1  
 console.setText(console.getText() + traducirOUT(0) + " -> | " + grafo.get(0).print(false, this)); 3  
 for(int i=1; i<cant; i++){ n + 5  
 console.setText(console.getText() + "\n" + traducirOUT(i) + " -> | " + grafo.get(i).print(false, this)); 3  
 }  
 }  
  
}

T(n) = n + 16

**Lista.java**

public class Lista implements Cloneable, Serializable {  
 private NodoDJ base; 1  
 private int cont; 1  
  
 public Lista(){  
 base = null; 1  
 cont = 0; 1  
 }  
  
 public int getCont(){  
 return cont; 1  
 }  
  
 public void addFinal(int v, int d){ 2  
 NodoDJ nuevo = new NodoDJ(v, d); 1  
 nuevo.setNext(null); 1  
  
 if(base == null){  
 base = nuevo; 1  
 }  
 else{  
 NodoDJ ulti = base; 2  
 while(ulti.getNext() != null){ n  
 ulti = ulti.getNext(); 1  
 }  
 ulti.setNext(nuevo); 1  
 }  
 cont++; 2  
 }

T(n) = n + 16  
  
 public void addInicio(int v, int d){ 2  
 NodoDJ nuevo = new NodoDJ(v, d); 1  
 if(base == null){  
 base = nuevo; 1  
 } else {  
 nuevo.setNext(base); 1  
 base = nuevo; 1  
 }  
 }  
  
 public void eliminarInicio(){  
 if(base != null){  
 base = base.getNext(); 1  
 }  
 }

T(n) = 7  
  
 public String print(boolean opc, Grafo grafo){ 2  
 StringBuilder ret = new StringBuilder(); 1  
 if(base == null){  
 //System.out.println("lista vacia");  
 return "--"; 2  
 } else {  
 //System.out.println("Lista:");  
 NodoDJ temp = base; 2  
 while (temp != null){ n  
 //System.out.println(temp.getVertice() + " , " + temp.getDistancia());  
 if (opc) {  
 ret.append(temp.getVertice()).append(" , ").append(temp.getDistancia()).append(" | "); 1  
 } else {  
 ret.append(grafo.traducirOUT(temp.getVertice())).append(" , ").append(temp.getDistancia()).append(" | "); 1  
 }  
 temp = temp.getNext(); 1  
 }  
 return ret.toString(); 1  
 }  
 }  
  
 public NodoDJ getPos(int pos){   
 if(base == null){  
 return null; 1  
 } else {  
 NodoDJ temp = base; 2  
 int i = 0; boolean flag = false; 4  
 while (i != pos && temp!=null){ n  
 temp = temp.getNext(); 1  
 i++; 2  
 }   
 if(temp!=null){  
 flag = true; 1  
 }   
 if(flag || i==pos){  
 return temp; 1  
 }  
 }  
 return null; 1  
 }  
  
 public Lista clone() throws CloneNotSupportedException {  
 return (Lista) super.clone(); 1  
 }  
  
}

T(n) = 2n + 25