Graphical solution to Dijktra's algorithm using Swing in Java

Solución gráfica al algoritmo de Dijkstra mediante el uso de Swing en Java

ABRIL-GARCÍA, José Humberto†*, CERÓN-FRANCO, Aureliano, GRIJALVA-ACUÑA, Juan Carlos and PITALÚA- DÍAZ, Nun

Universidad de Sonora, Departamento de Ingeniería Industrial. México.

ID 1st Author: *José Humberto, Abril-García* / **ORC ID**: 0000-0003-3494-6817, **Researcher ID Thomson:** F-4252-2018, **arXiv ID**: jhabril, **CVU CONACYT ID:** 204935

ID 1st Co-author: *Aureliano, Cerón-Franco* / **ORC ID:** 0000-0001-5953-1189, **Researcher ID Thomson**: HGU-6869-2022, **arXiv ID:** AURELIANO_CERON_1975, **CVU CONACYT ID:**166318

ID 2nd Co-author: *Juan Carlos, Grijalva-Acuña* / **ORC ID:** 0000-0001-9721-7879, **Researcher ID Thomson**: HGU-0872-2022, **arXiv ID:** JuanGrijalva1976, **CVU CONACYT ID:** 585450

ID 3er Co-author: *Nun, Pitalúa-Díaz* / **ORC ID:** 0000-0002-8671-1422, **CVU CONACYT ID:** 100050

DOI: 10.35429/JMQM.2022.11.6.24.38 Received July 30, 2022; Accepted December 30, 2022

Abstract

The search for new possibilities for graphical user interfaces has turned out to be a very important part of the software, not only for functional reasons or for using the application to fulfill the slogan of an intuitive task, but also because it has been demonstrated at times. Dijkstra's algorithm uses a data structure to store and query partial solutions ordered by distance from the beginning. Development giants spend a lot of resources creating more elegant and intuitive user interfaces for their products, because they have experienced firsthand how useful, well-designed software can sometimes fail when its GUI fails. Dijkstra's algorithm is an algorithm used to determine the shortest trajectory from a certain point to any other point on a graph. In the course of the work, it was possible to generate an application that graphically solves Dijkstra's algorithm. Compatible with Windows, Ubuntu, macOS operating systems using Java programming language. This application serves as a guide to understand concepts related to graph theory and understand how Dijkstra's algorithm works, as well as topics related to object-oriented programming.

Graph, Swing, GIU

Resumen

La búsqueda de nuevas posibilidades para las interfaces gráficas de usuario ha resultado ser una parte muy importante del software, no solo por cuestiones funcionales o por utilizar la aplicación para cumplir con la consigna de una tarea intuitiva, sino también porque se ha demostrado en ocasiones. El algoritmo de Dijkstra utiliza una estructura de datos para almacenar y consultar soluciones parciales ordenadas por distancia desde el principio. Los gigantes del desarrollo gastan muchos recursos para crear interfaces de usuario más elegantes e intuitivas para sus productos, porque han experimentado de primera mano cómo el software útil y bien diseñado a veces puede fallar cuando falla su GUI. El algoritmo de Dijkstra es un algoritmo utilizado para determinar la trayectoria más corta desde un cierto punto a cualquier otro punto en un gráfico. En el transcurso del trabajo se logró generar una aplicación que resuelve gráficamente el algoritmo de Dijkstra. Compatible con los sistemas operativos Windows, Ubuntu, macOS utilizando el lenguaje de programación Java. Esta aplicación sirve como guía para comprender conceptos relacionados con la teoría de grafos y entender cómo funciona el algoritmo de Dijkstra, así como temas relacionados con la programación orientada a objetos.

Grafo, Swing GIU

Citation: ABRIL-GARCÍA, José Humberto, CERÓN-FRANCO, Aureliano, GRIJALVA-ACUÑA, Juan Carlos and PITALÚA- DÍAZ, Nun. Graphical solution to Dijktra's algorithm using Swing in JavaJournal-Mathematical and Quantitative Methods. 2022. 6-11:24-38.

* Author Correspondence (E-mail: jose.abril@unison.mx)
† Researcher contributing as first author.

Introduction

The search for new possibilities for graphical user interfaces has turned out to be a very important part of software, not only for functional reasons or for using the application to fulfil an intuitive task, but also because it has sometimes been shown that it is necessary to improve the appearance of the software so that it does not lose its attractiveness.

Dijkstra's algorithm uses a data structure to store and query distance-ordered partial solutions from the beginning. While the original algorithm uses a minimum priority queue and runs in time $\Theta((|E|+|V|)\log|V|)$ (where |V| is the number of nodes and |E| is the number of vertices), it can also be implemented in $\Theta(|V|^2)$ using a matrix.

A graph is a structure consisting of nodes and connections between them. In graph terminology, the nodes are called vertices and the connections between them are called edges. Vertices are usually identified by a name or a label. For example, we might label points A, B, C and D. Edges are referenced by matching the vertices they connect. For example, we might have an edge (A, B), which means there is an edge from vertex A to vertex B.

An undirected graph is a graph in which the pairs representing the edges are not ordered. Listing an edge as (A, B) means that there is a link between A and B that can be traversed in any direction. In an unpublished graph, the listing edge (A, B) means exactly the same as the listing edge (B, A).

We know that the popularity of today's computer products is largely due to their appearance. Development giants spend a lot of resources to create sleeker and more intuitive user interfaces for their products, because they have experienced first-hand how useful and well-designed software can sometimes fail when its GUI fails. It is unattractive, yet other poorly designed but effective interfaces drive away the competition, because users consider them more effective, even though this rating is only supported by their graphical side.

Frame of reference

The beginnings of graph theory date back to 1735, when the Swiss mathematician Leonhard Euler sent a paper to St. Petersburg's St. Petersburg Academy and published in 1741 about the seven bridges of Königsberg. The challenge is to find a way back from one point to the same point by crossing each of the seven Königsberg bridges once and only once. A path that crosses each edge exactly once is called an Euler path or Euler cycle if it ends at the origin. By extension, a graph containing an Euler circuit is called an Euler graph and thus constitutes the first instance of the properties of the graph. Euler proved that a graph is Euler if each vertex has an even number of edges. The concept of a tree (a connected graph without cycles) was realised in 1845 by Gustav Kirchhoff, who used graph-theoretic ideas when calculating the current in a network or circuit. In 1852, Thomas Guthery discovered the famous four-colour problem. Then in 1856 Thomas. P. Kirkman and William R. Hamilton studied cycles in polyhydrates and invented a concept called the Hamiltonian graph, studying trips that visited certain places only once. In 1913, H. Dudeney mentioned an enigma. Although the four-colour problem was invented, it was not solved until a century later by Kenneth Apel and Wolfgang Hacken. This time is considered the birth of graph theory. [6]

1. Understanding graph theory

2. A graph G = (V,E) consists of V a set of non-empty vertices and E a set of edges. Each line has between one or two points connected to it, called its endpoints.

3. Types of graphs

4. Depending on the presence or absence of bracelets or double sides in a graph, graphs can generally be classified into two types. [2]

5. Simple graph. Graphs that contain no wristbands or double-sided graphs are called simple graphs. G_1 in Figure 1(a) is an example of a simple graph. In a simple graph, the sides are unordered pairs. Therefore, writing the side (u,v) is equivalent to (v,u). We can also define a simple graph G = (V,E) which consists of a non-empty set of vertices and E is a set of different unordered pairs called sides.

6. Graphs of non-simple graphs Graphs containing double faces or bracelets are called non-simple graphs. There are two types of non-simple graphs, namely dual graphs (multigraphs) and pseudographs (pseudographs). A double graph is a graph containing multiple edges (multiple edges or parallel edges). The double side connecting a pair of vertices can be more than two pieces. in Figure 1(b) is an example of a double graph because it has sides e_3 and e_4 that are double-sided. The double sides can be associated as unordered equal pairs. We can also define a dual graph G_2 G = (V,E) which consists of a non-empty set of vertices and E is a dual set (multiset) containing a dual side.

7. A pseudo-graph is a graph containing a bracelet or kalang (loop). G_3 in Figure 1(c) is an example of a pseudo-graph because it has a side e3 that is an armlet. A pseudograph is more common than a double graph because the pseudograph side is connected to itself.

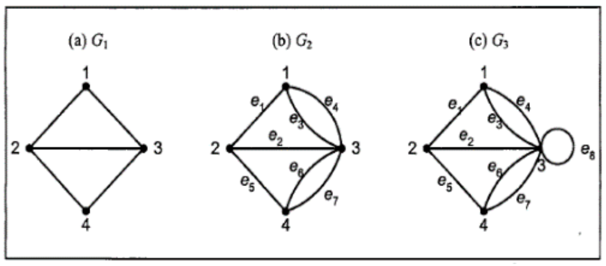


Figure 1 Three graphs (a) single graphs, (b) double graphs and (c) pseudographs

Dijkstra's algorithm

Dijkstra's algorithm is an algorithm used to determine the shortest path from one given point to another in a diagram. This algorithm was developed by Edsger Wybe Dijkstra in 1959. Dijkstra's algorithm uses an algorithmic strategy to select the best of all options. The algorithm occupies a side with a lower weight than the unselected nodes connected to an already selected node. The path from the source node to the new node is the shortest path.

The inputs to this algorithm are the weighted directed graph G and the source node s in G, where V is the set of all nodes in the graph G. Dijkstra's algorithm weights the graph G=(V,E) if all edges have non-negative weights. Therefore, in this section we assume $w(u,v) \geq 0$ for each edge $(u,v) \in E$ [2].

The algorithm uses the table S=[si]. where si=1 if the gradient i belongs. Shortest path and vice versa si=0 (if gradient i does not belong to the shortest path and table D=[di]), di=length of the path from the first node A to node I. [2].

Procedure to find the shortest path Using Djikstra's algorithm, for a graph G with n-vertices fruit with initial node a:

Initialise all distances in D with a relative infinite value (∞,-) since they are unknown at the beginning, except for x which should be set to 0 because the distance from x to x would be 0 (0,-).

Let a=x (we take a as the current node).

We traverse all adjacent nodes of a, except for the marked nodes, we will call these vi.

If the distance from x to vi stored in D is greater than the distance from x to a added to the distance from a to vi; this is replaced with the second named one, that is: if (Di>Da+d(a,vi)) then Di=Da+d(a,vi).

We mark node a as complete.

We take as the next current node the one with the smallest value in D (this can be done by storing the values in a priority queue) and go back to step 3 as long as there are unmarked nodes.

Once the algorithm is finished, D will be completely full. [7]

Method description

Given a weighted graph and an initial (source) vertex in the graph, Dijkstra's algorithm is used to find the shortest distance between the source node and all other nodes in the graph. As a result of running Dijkstra's algorithm on the graph, we obtain the shortest path tree with the source vertex as the root.

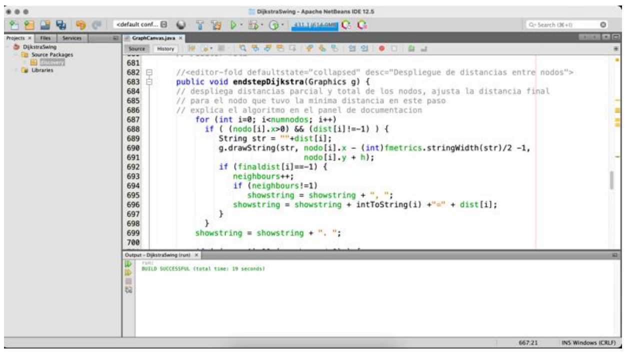
In Dijkstra's algorithm we maintain two sets or lists. One contains vertices that are part of the shortest path tree and the other contains vertices evaluated for inclusion in the shortest path. Therefore, for each iteration, we find a vertex from the second list with the shortest path. Implementing Dijkstra's shortest path algorithm in Java can be done in two ways.

We can use priority queues and adjacency lists or we can use matrices and adjacency matrices.

Using priority queues; in this implementation we use the priority queue to store the vertices with the shortest distance. The graph is defined by the adjacency list.

Use adjacency matrix; in this method, we use the adjacency matrix to represent the graph. For the set of shortest paths, we use a matrix.

For the development of the application, the Agile methodology was used. In the first sprint the back end is programmed which executes Dijkstra's algorithm on a directed graph. In figure 2 we can see a section of the code.



```
    //<editor-fold defaultstate="collapsed" desc="Despliegue de distancias entre nodos">
    public void endstepDijkstra(Graphics g) {
    // despliega distancias parcial y total de los nodos, ajusta la distancia final
    // para el nodo que tuvo la minima distancia en este paso
    // explica el algoritmo en el panel de documentacion
        for (int i=0; i<numnodos; i++)
          if ( (nodo[i].x>0) && (dist[i]!=-1) ) {
             String str = ""+dist[i];
             g.drawString(str, nodo[i].x - (int)fmetrics.stringWidth(str)/2 -1,
                               nodo[i].y + h);
             if (finaldist[i]==-1) {
                neighbours++;
                if (neighbours!=1)
                   showstring = showstring + ", ";
                showstring = showstring + intToString(i) +"=" + dist[i];
             }
          }
        showstring = showstring + ". ";
```

```
run:
BUILD SUCCESSFUL (total time: 19 seconds)
```

Figure 2 Sprint 1

In Sprint 2 (Figure 3) an application was programmed that allows drawing graphs on a screen, the user can add, delete and edit nodes. Edges can be added, fed and edited as well as their weight.

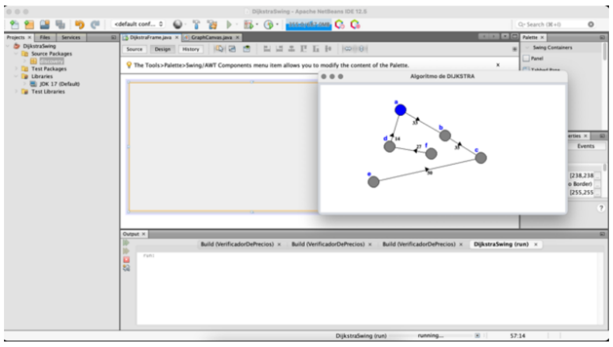
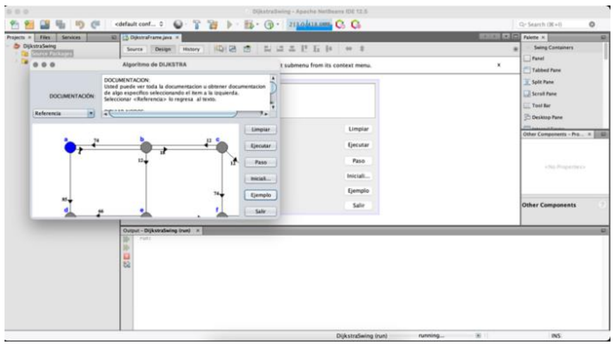


Figure 3 Sprint 2

In the third Sprint 3 (Figure 4), the development of the application is completed, integrating the Document, Clear, Step, Initialise, Example and Exit options, resulting in an application that allows the user to create a directed graph and solve the shortest path using Dijkstra's algotimo.

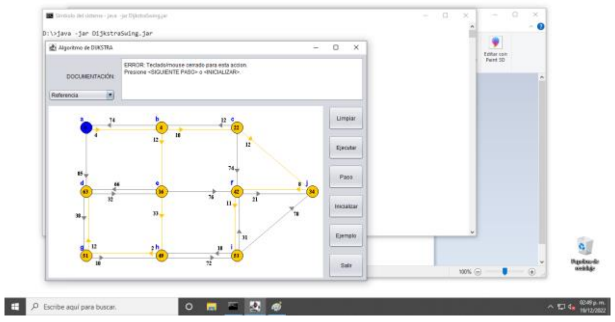


DOCUMENTACION:
Usted puede ver toda la documentacion u obtener documentacion de algo especifico seleccionando el item a la izquierda.
Seleccionar <Referencia> lo regresa al texto.

Figure 4 Sprint 3

Results

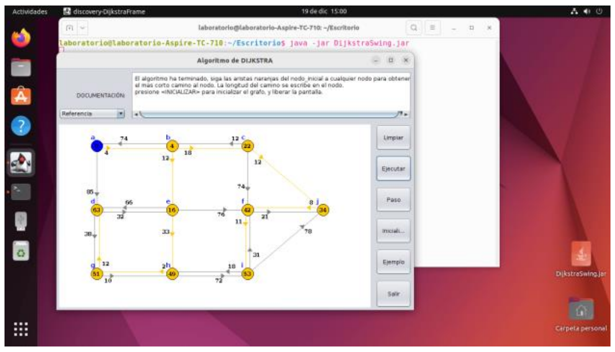
Figure 5 shows the application running on the Windows 10 Pro operating system, Figure 6 on the Ubuntu 22.04.1 LTS operating system, Figure 7 on the macOS Big Sur version 11 operating system, and Figure 8 on a Raspberry Pi B device with the Raspbian GNU/Linux 11 operating system (bullseye).



```
D:\>java -jar DijkstraSwing.jar
```

ERROR: Teclado/mouse cerrado para esta accion.
Presione <SIGUIENTE PASO> o <INICIALIZAR>.

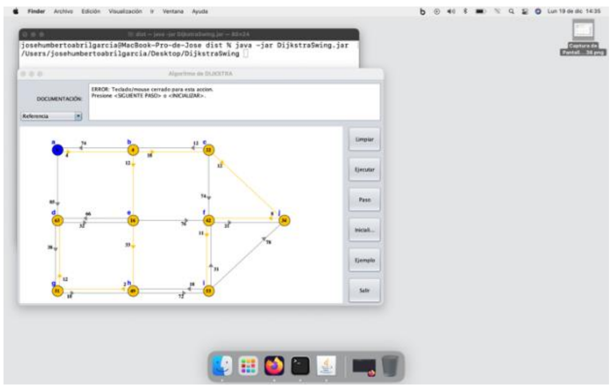
Figure 5 Windows 10 Pro



```
laboratorio@laboratorio-Aspire-TC-710:~/Escritorio$ java -jar DijkstraSwing.jar
```

El algoritmo ha terminado, siga las aristas naranjas del nodo inicial a cualquier nodo para obtener el mas corto camino al nodo. La longitud del camino se escribe en el nodo.
presione <INICIALIZAR> para inicializar el grafo, y liberar la pantalla.

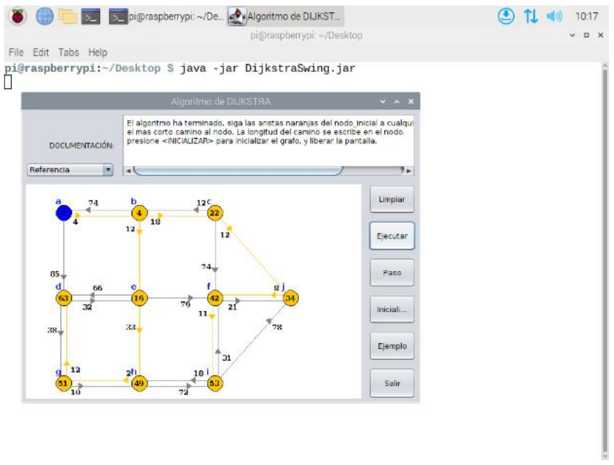
Figuea 6 Ubuntu 22.04.1 LTS



```
josehumbertoabrilgarcia@MacBook-Pro-de-Jose dist % java -jar DijkstraSwing.jar
/Users/josehumbertoabrilgarcia/Desktop/DijkstraSwing
```

ERROR: Teclado/mouse cerrado para esta accion.
Presione <SIGUIENTE PASO> o <INICIALIZAR>.

Figure 7 macOS Big Sur versión 11



```
pi@raspberrypi:~/Desktop $ java -jar DijkstraSwing.jar
```

El algoritmo ha terminado, siga las aristas naranjas del nodo_inicial a cualqu
el mas corto camino al nodo. La longitud del camino se escribe en el nodo.
presione <INICIALIZAR> para inicializar el grafo, y liberar la pantalla.

Figure 8 Raspbian GNU/Linux 11

With the development of this work it was possible to generate an application that solves Dijkstra's algorithm graphically. Compatible with Windows, Ubuntu, macOS and Raspbian GNU/Linux 11 operating systems, using the Java programming language. This application serves as a guide to understand concepts related to graph theory and to understand how Dijkstra's algorithm works, as well as topics related to object-oriented programming. Additionally, this program can be used as a basis for the development of more robust and complex problems. The source code of the application is shown in Annex 1.

Annex 1

```
/*
 * To change this license header, choose License
Headers in Project Properties.
 * To change this template file, choose Tools |
Templates
 * and open the template in the editor.
 */
package discovery;

import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Event;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Point;

public final class GraphCanvas extends Canvas
implements Runnable {

   DijkstraFrame parent;

   //   <editor-fold   defaultstate="collapsed"
desc="Constructores">
   GraphCanvas(DijkstraFrame myparent) {
   parent = myparent;
   init();
   algoritmo=DIJKSTRA;
   }
   // </editor-fold>

   //   <editor-fold   defaultstate="collapsed"
desc="Area de dibujo del grafo">
   final int MAXNODOS = 20;
   final int MAX = MAXNODOS+1;
   final int NODOSIZE = 26;
   final int NODORADIX = 13;
   final int DIJKSTRA = 1;
   // </editor-fold>

   //   <editor-fold   defaultstate="collapsed"
desc="Informacion basica del grafo">
   Point nodo[] = new Point[MAX];      // Nodo
   int peso[][] = new int[MAX][MAX];       //
Peso de arista
   Point arista[][] = new Point[MAX][MAX]; //
Posicion actual de la flecha
   Point startp[][] = new Point[MAX][MAX]; //
Punto inicial de arista
   Point endp[][] = new Point[MAX][MAX];   //
Punto final de arista
   float dir_x[][] = new float[MAX][MAX];   //
Direccion de arista
   float dir_y[][] = new float[MAX][MAX];   //
Direccion de arista
   // </editor-fold>

   //   <editor-fold   defaultstate="collapsed"
desc="Informacion  del  grafo  al  ejecutar  el
algoritmo">
   boolean        algedge[][]        =        new
boolean[MAX][MAX];
   int dist[] = new int[MAX];
```

```
  int finaldist[] = new int[MAX];
  Color colornodo[] = new Color[MAX];
  boolean changed[] = new boolean[MAX];   //
Indica cambio de distancia durante el algoritmo
  int numchanged =0;
  int neighbours=0;
  int paso=0;
  // </editor-fold>

  //   <editor-fold   defaultstate="collapsed"
desc="Informacion usada por el algoritmo para
encontrar el siguiente nodo con minima
distancia">
  int mindist, minnodo, minstart, minend;
  int numnodos=0;      // Numero ode nodos
  int emptyspots=0;     // Lugares vacios en el
arreglo nodo[] (por borrado de nodos)
  int startgrafo=0;    // Comienzo de grafo
  int hitnodo;        // Click del mouse en o cerca
del nodo
  int nodo1, nodo2;       // Numero de nodos
envueltos in la accion
  Point thispoint=new Point(0,0); // Posicion
actual del mouse
  Point oldpoint=new Point(0, 0); // Posicion
previa del nodo
  // </editor-fold>

  //   <editor-fold   defaultstate="collapsed"
desc="Accion actual">
  boolean newarista = false;
  boolean movearista = false;
  boolean movestart = false;
  boolean deletenodo = false;
  boolean movenodo = false;
  boolean performalg = false;
  boolean clicked = false;
  // </editor-fold>

  //   <editor-fold   defaultstate="collapsed"
desc="Fuentes">
  Font   roman=   new   Font("TimesRoman",
Font.BOLD, 12);
  Font    helvetica=   new    Font("Helvetica",
Font.BOLD, 15);
  FontMetrics             fmetrics             =
getFontMetrics(roman);
  int h = (int)fmetrics.getHeight()/3;
  // </editor-fold>

  //   <editor-fold   defaultstate="collapsed"
desc="Buffer doble">
  private Image offScreenImage;
  private Graphics offScreenGraphics;
  private Dimension offScreenSize;
  // </editor-fold>

  //   <editor-fold   defaultstate="collapsed"
desc="Hilo de ejecucion">
  Thread algrthm;

  public void start() {
  if (algrthm != null) algrthm.resume();
  }

  public void stop() {
  if (algrthm != null) algrthm.suspend();
  }

  @Override
  public void run() {
  for(int i=0; i<(numnodos-emptyspots); i++){
    nextstep();
    try { algrthm.sleep(2000); }
    catch (InterruptedException e) {}
  }
  algrthm = null;
  }
  // </editor-fold>

  //   <editor-fold   defaultstate="collapsed"
desc="Creacion de los nodos del grafo">
  public void init() {
  for (int i=0;i<MAXNODOS;i++) {
    colornodo[i]=Color.gray;
    for (int j=0; j<MAXNODOS;j++)
      algedge[i][j]=false;
  }
  colornodo[startgrafo]=Color.blue;
  performalg = false;
  }
  // </editor-fold>

  //   <editor-fold   defaultstate="collapsed"
desc="Algoritmo actual">
  int algoritmo;
  // </editor-fold>

  //   <editor-fold   defaultstate="collapsed"
desc="Informacion del algoritmo para ser
desplegado en la documentacion">
  String showstring = "";
  boolean stepthrough=false;
  // </editor-fold>

  //   <editor-fold   defaultstate="collapsed"
desc="Bloquear o desbloquear la pantalla
mientras se ejecuta el algoritmo">
  boolean Locked = false;
  public void lock() {
    Locked=true;
  }

  public void unlock() {
  Locked=false;
  }
  // </editor-fold>

  //    <editor-fold    defaultstate="collapsed"
desc="Remover grafo de la pantalla">
  public void limpiar() {
  startgrafo=0;
  numnodos=0;
  emptyspots=0;
  init();
  for(int i=0; i<MAXNODOS; i++) {
    nodo[i]=new Point(0, 0);
    for (int j=0; j<MAXNODOS;j++)
        peso[i][j]=0;
  }
  if (algrthm != null)
         algrthm.stop();
  unlock(); //sustituido
  repaint();
  }
  // </editor-fold>

  //    <editor-fold    defaultstate="collapsed"
desc="Inicializar un grafo despues de ejecutar
un algoritmo">
  public void inicializar() {
  init();
  if (algrthm != null) algrthm.stop();
  unlock(); //sustituido
  repaint();
  }
  // </editor-fold>

  //    <editor-fold    defaultstate="collapsed"
desc="Animacion del algoritmo">
  public void runalg() {
  lock(); //sustituido
  initalg();
  performalg = true;
  algrthm   =   new   Thread((Runnable)   this);
//Thread casteado
  algrthm.start();
  }
  // </editor-fold>

  //    <editor-fold    defaultstate="collapsed"
desc="Iniciar   ejecucion   paso   a   paso   del
algoritmo">
 public void stepalg() {
  lock(); //sustituido
  initalg();
  performalg = true;
  nextstep();
  }
  // </editor-fold>

  //    <editor-fold    defaultstate="collapsed"
desc="Ejecutar algoritmo">
  public void initalg() {
  init();
  for(int i=0; i<MAXNODOS; i++) {
   dist[i]=-1;
   finaldist[i]=-1;
   for (int j=0; j<MAXNODOS;j++)
        algedge[i][j]=false;
  }
    dist[startgrafo]=0;
  finaldist[startgrafo]=0;
  paso=0;
  }
  // </editor-fold>

  //    <editor-fold    defaultstate="collapsed"
desc="Calcular camino mas corto al siguiente
nodo">
  public void nextstep() {
    finaldist[minend]=mindist;
  algedge[minstart][minend]=true;
  colornodo[minend]=Color.orange;
  paso++;
  repaint();
  }
  // </editor-fold>

  //    <editor-fold    defaultstate="collapsed"
desc="Dibujar grafo de ejemplo">
  public void showejemplo() {
  int w, h;
  limpiar();
  init();
  numnodos=10;
  emptyspots=0;
  for(int i=0; i<MAXNODOS; i++) {
      nodo[i]=new Point(0, 0);
      for (int j=0; j<MAXNODOS;j++)
        peso[i][j]=0;
  }
  w=this.getWidth()/16;
  h=this.getHeight()/16;
  nodo[0]=new Point(w, h);
    nodo[1]=new Point(3*w, h);
  nodo[2]=new Point(5*w, h);
    nodo[3]=new Point(w, 4*h);
  nodo[4]=new Point(3*w, 4*h);
```

```
    nodo[5]=new Point(5*w, 4*h);
  nodo[6]=new Point(w, 7*h);
    nodo[7]=new Point(3*w, 7*h);
  nodo[8]=new Point(5*w, 7*h);
    nodo[9]=new Point(7*w, 4*h);
  peso[0][1]=4;
    peso[0][3]=85;
  peso[1][0]=74;
    peso[1][2]=18;
    peso[1][4]=12;
  peso[2][5]=74;
    peso[2][1]=12;
    peso[2][9]=12;
  peso[3][4]=32;
    peso[3][6]=38;
  peso[4][3]=66;
    peso[4][5]=76;
    peso[4][7]=33;
  peso[5][8]=11;
    peso[5][9]=21;
  peso[6][7]=10;
    peso[6][3]=12;
  peso[7][6]=2;
    peso[7][8]=72;
  peso[8][5]=31;
    peso[8][9]=78;
    peso[8][7]=18;
  peso[9][5]=8;
  for (int i=0;i<numnodos;i++)
        for (int j=0;j<numnodos;j++)
            if (peso[i][j]>0)
                aristaupdate(i, j, peso[i][j]);
  repaint();
  }
  // </editor-fold>

  //<editor-fold defaultstate="collapsed"
desc="Eventos del grafo relacionados con el
mouse">
  @Override
  public boolean mouseDown(Event evt, int x,
int y) {

  if (Locked)
    parent.showline("cerrado");
  else {
   clicked = true;
   if (evt.shiftDown()) {
   // Mover un nodo
     if (nodohit(x, y, NODOSIZE)) {
       oldpoint = nodo[hitnodo];
       nodo1 = hitnodo;
       movenodo=true;
     }
   }
   else if (evt.controlDown()) {
   // Borrar un nodo
     if (nodohit(x, y, NODOSIZE)) {
       nodo1 = hitnodo;
       if (startgrafo == nodo1) {
         movestart=true;
         thispoint = new Point(x,y);
            colornodo[startgrafo] = Color.gray;
       }
       else
         deletenodo= true;
     }
   }
   else if (aristahit(x, y, 5)) {
   // Cambiar peso de una arista
     movearista = true;
     repaint();
   }
   else if (nodohit(x, y, NODOSIZE)) {
   // Dibuja una nueva arista
     if (!newarista) {
       newarista = true;
       thispoint = new Point(x, y);
       nodo1 = hitnodo;
     }
    }
    else if ( !nodohit(x, y, 50) && !aristahit(x,
y, 50) )  {
    // Dibuja nuevo nodo
      if (emptyspots==0) {
      // toma el siguiente punto disponible en el
arreglo
        if (numnodos < MAXNODOS)
          nodo[numnodos++]=new Point(x, y);
        else parent.showline("maxnodos");
      }
      else {
      // Tomar un punto vacio en el array (de
algun nodo borrado previamente)
        int i;
        for (i=0;i<numnodos;i++)
          if (nodo[i].x==-100) break;
        nodo[i]=new Point(x, y);
        emptyspots--;
      }
    }
    else
    // Mouseclick para cerrar a un point r flecha,
probablemente un error
      parent.showline("toclose");
    repaint();
  }
  return true;
  }

  public boolean mouseDrag(Event evt, int x,
int y) {
  if ( (!Locked) && clicked ) {
    if (movenodo) {
    // mover nodo y ajustar aristas
entrando/saliendo del nodo
      nodo[nodo1]=new Point(x, y);
      for (int i=0;i<numnodos;i++) {
        if (peso[i][nodo1]>0) {
          aristaupdate(i, nodo1,
peso[i][nodo1]);
        }
        if (peso[nodo1][i]>0) {
          aristaupdate(nodo1, i,
peso[nodo1][i]);
        }
      }
      repaint();
    }
    else if (movestart || newarista) {
      thispoint = new Point(x, y);
      repaint();
    }
    else if (movearista) {
      changepeso(x, y);
      repaint();
    }
  }
  return true;
  }

  public boolean mouseUp(Event evt, int x, int
y) {
  if ( (!Locked) && clicked ) {
    if (movenodo) {
    // mover el nodo si la nueva posicion no esta
muy cerca a
    // otro nodo o fuera del panel
      nodo[nodo1]=new Point(0, 0);
      if ( nodohit(x, y, 50) || (x<0) ||
(x>this.size().width) ||
                (y<0) || (y>this.size().height) ) {
        nodo[nodo1]=oldpoint;
        parent.showline("toclose");
      }
      else nodo[nodo1]=new Point(x, y);
      for (int i=0;i<numnodos;i++) {
        if (peso[i][nodo1]>0)
          aristaupdate(i, nodo1,
peso[i][nodo1]);
        if (peso[nodo1][i]>0)
          aristaupdate(nodo1, i,
peso[nodo1][i]);
      }
      movenodo=false;
    }
    else if (deletenodo) {
      nododelete();
      deletenodo=false;
    }
    else if (newarista) {
      newarista = false;
      if (nodohit(x, y, NODOSIZE)) {
        nodo2=hitnodo;
        if (nodo1!=nodo2) {
          aristaupdate(nodo1, nodo2, 50);
          if (peso[nodo2][nodo1]>0) {
            aristaupdate(nodo2, nodo1,
peso[nodo2][nodo1]);
          }
          parent.showline("Cambiar pesos");
        }
        //else System.out.println("zelfde");
      }
    }
    else if (movearista) {
      movearista = false;
      if (peso[nodo1][nodo2]>0)
        changepeso(x, y);
    }
    else if (movestart) {
    // si la nueva posicion es un nodo, este nodo
es el nodo_inicial
      if (nodohit(x, y, NODOSIZE))
        startgrafo=hitnodo;
      colornodo[startgrafo]=Color.blue;
      movestart=false;
    }
    repaint();
  }
  return true;
  }
  //</editor-fold>

  //<editor-fold defaultstate="collapsed"
desc="Eventos del grafo relacionados con el
canvas">
  public boolean nodohit(int x, int y, int dist) {
  // checa si hay click sobre un nodo
  for (int i=0; i<numnodos; i++)
    if ( (x-nodo[i].x)*(x-nodo[i].x) +
          (y-nodo[i].y)*(y-nodo[i].y) < dist*dist
) {
      hitnodo = i;
      return true;
    }
  return false;
  }

  public boolean aristahit(int x, int y, int dist) {
  // checa si hay click sobre una arista
  for (int i=0; i<numnodos; i++)
    for (int j=0; j<numnodos; j++) {
      if ( ( peso[i][j]>0 ) &&
        (Math.pow(x-arista[i][j].x, 2) +
         Math.pow(y-arista[i][j].y, 2) <
Math.pow(dist, 2) ) ) {
        nodo1 = i;
        nodo2 = j;
        return true;
      }
    }
  return false;
  }

  public void nododelete() {
  // borra un nodo y las aristas que entran/salen
del nodo
  nodo[nodo1]=new Point(-100, -100);
  for (int j=0;j<numnodos;j++) {
    peso[nodo1][j]=0;
    peso[j][nodo1]=0;
  }
  emptyspots++;
  }

  public void changepeso(int x, int y) {
  // cambia el peso de una arista, cuando el
usuario arrastra
  // la flecha sobre la arista que conecta el nodo1
al nodo2.
  // direccion de la arista
  int diff_x = (int)(20*dir_x[nodo1][nodo2]);
  int diff_y = (int)(20*dir_y[nodo1][nodo2]);
  // dependiendo de la direccion de la arista ,
sigue el x, o el y
  // del mouse mientras la flecha se arrastra
  boolean siga_x=false;
    if (Math.abs(endp[nodo1][nodo2].x-
startp[nodo1][nodo2].x) >
         Math.abs(endp[nodo1][nodo2].y-
startp[nodo1][nodo2].y) ) {
    siga_x = true;
  }
  // encuentra la nueva posicion de la flecha, y
calcula
  // el peso correspondiente
  if (siga_x) {
    int hbound =
Math.max(startp[nodo1][nodo2].x,
           endp[nodo1][nodo2].x)-
Math.abs(diff_x);
    int lbound =
Math.min(startp[nodo1][nodo2].x,

endp[nodo1][nodo2].x)+Math.abs(diff_x);
    // la arista debe quedarse entre los nodos
    if (x<lbound) {
arista[nodo1][nodo2].x=lbound; }
    else if (x>hbound) {
arista[nodo1][nodo2].x=hbound; }
    else arista[nodo1][nodo2].x=x;
    arista[nodo1][nodo2].y=
    (arista[nodo1][nodo2].x-
startp[nodo1][nodo2].x) *
      (endp[nodo1][nodo2].y-
startp[nodo1][nodo2].y)/
        (endp[nodo1][nodo2].x-
startp[nodo1][nodo2].x) +
    startp[nodo1][nodo2].y;
    // nuevo peso
    peso[nodo1][nodo2]=
    Math.abs(arista[nodo1][nodo2].x-
startp[nodo1][nodo2].x-diff_x)*
      100/(hbound-lbound);
  }
  // hacer lo mismo si sigue y
  else {
    int hbound =
Math.max(startp[nodo1][nodo2].y,
         endp[nodo1][nodo2].y)-
Math.abs(diff_y);
    int lbound =
Math.min(startp[nodo1][nodo2].y,

endp[nodo1][nodo2].y)+Math.abs(diff_y);
    if (y<lbound) {
arista[nodo1][nodo2].y=lbound; }
    else if (y>hbound) {
arista[nodo1][nodo2].y=hbound; }
    else arista[nodo1][nodo2].y=y;
    arista[nodo1][nodo2].x=
    (arista[nodo1][nodo2].y-
startp[nodo1][nodo2].y) *
      (endp[nodo1][nodo2].x-
startp[nodo1][nodo2].x)/
      (endp[nodo1][nodo2].y-
startp[nodo1][nodo2].y) +
    startp[nodo1][nodo2].x;
    peso[nodo1][nodo2]=
    Math.abs(arista[nodo1][nodo2].y-
startp[nodo1][nodo2].y-diff_y)*
      100/(hbound-lbound);
  }
  }

  public void aristaupdate(int p1, int p2, int w)
{
  // hacer una arista nueva del nodo p1 al p2 con
peso w, o cambiar
  // el peso de la arista a w, calcular la
  // posicion resultante de la flecha
  int dx, dy;
  float l;
  peso[p1][p2]=w;
```

```
   // linea de direccion entre p1 y p2
   dx = nodo[p2].x-nodo[p1].x;
   dy = nodo[p2].y-nodo[p1].y;
   // distancia entre p1 y p2
   l = (float)( Math.sqrt((float)(dx*dx +
dy*dy)));
   dir_x[p1][p2]=dx/l;
   dir_y[p1][p2]=dy/l;
   // calcular el start y endpoints de la arista,
   // ajustar startpoints si tambien hay una arista
de p2 a p1
   if (peso[p2][p1]>0) {
     startp[p1][p2] = new
Point((int)(nodo[p1].x-5*dir_y[p1][p2]),

(int)(nodo[p1].y+5*dir_x[p1][p2]));
     endp[p1][p2] = new Point((int)(nodo[p2].x-
5*dir_y[p1][p2]),

(int)(nodo[p2].y+5*dir_x[p1][p2]));
   }
   else {
     startp[p1][p2] = new Point(nodo[p1].x,
nodo[p1].y);
     endp[p1][p2] = new Point(nodo[p2].x,
nodo[p2].y);
   }
   // la distancia de la flecha no es todo el camino
a los puntos de inicio/final
   int diff_x =
(int)(Math.abs(20*dir_x[p1][p2]));
   int diff_y =
(int)(Math.abs(20*dir_y[p1][p2]));
   // calcular nueva posicion en x de la flecha
   if (startp[p1][p2].x>endp[p1][p2].x) {
     arista[p1][p2] = new Point(endp[p1][p2].x +
diff_x +
      (Math.abs(endp[p1][p2].x-
startp[p1][p2].x) - 2*diff_x )*
        (100-w)/100 , 0);
   }
   else {
     arista[p1][p2] = new Point(startp[p1][p2].x
+ diff_x +
      (Math.abs(endp[p1][p2].x-
startp[p1][p2].x) - 2*diff_x )*
        w/100, 0);
   }
   // calcular nueva posicion en y de la flecha
   if (startp[p1][p2].y>endp[p1][p2].y) {
     arista[p1][p2].y=endp[p1][p2].y + diff_y +
      (Math.abs(endp[p1][p2].y-
startp[p1][p2].y) - 2*diff_y )*
        (100-w)/100;
   }
   else {
     arista[p1][p2].y=startp[p1][p2].y + diff_y +
      (Math.abs(endp[p1][p2].y-
startp[p1][p2].y) - 2*diff_y )*
        w/100;
   }
   }

   public String intToString(int i) {
   char c=(char)((int)'a'+i);
   return ""+c;
   }
   //</editor-fold>

   // <editor-fold defaultstate="collapsed"
desc="Preparar una nueva imagen fuera de la
pantalla">
   public final synchronized void
update(Graphics g) {
   // preparar nueva imagen fuera de la pantalla
   Dimension d=size();
   if ((offScreenImage == null) || (d.width !=
offScreenSize.width) ||
        (d.height != offScreenSize.height)) {
      offScreenImage = createImage(d.width,
d.height);
      offScreenSize = d;
      offScreenGraphics =
offScreenImage.getGraphics();
   }
   offScreenGraphics.setColor(Color.white);
   offScreenGraphics.fillRect(0, 0, d.width,
d.height);
   paint(offScreenGraphics);
   g.drawImage(offScreenImage, 0, 0, null);
   }
   // </editor-fold>

   //<editor-fold defaultstate="collapsed"
desc="Dibujar arista entre nodo i y nodo j">
   public void drawarista(Graphics g, int i, int j)
{
   // dibuja arista entre nodo i y nodo j
   int x1, x2, x3, y1, y2, y3;
   // calcular flecha
   x1= (int)(arista[i][j].x - 3*dir_x[i][j] +
6*dir_y[i][j]);
   x2= (int)(arista[i][j].x - 3*dir_x[i][j] -
6*dir_y[i][j]);
   x3= (int)(arista[i][j].x + 6*dir_x[i][j]);
   y1= (int)(arista[i][j].y - 3*dir_y[i][j] -
6*dir_x[i][j]);
   y2= (int)(arista[i][j].y - 3*dir_y[i][j] +
6*dir_x[i][j]);
   y3= (int)(arista[i][j].y + 6*dir_y[i][j]);
   int flecha_x[] = { x1, x2, x3, x1 };
   int flecha_y[] = { y1, y2, y3, y1 };
   // si la arista ya se escogio por el algoritmo
cambiar color
   if (algedge[i][j]) g.setColor(Color.orange);
   // dibuja arista
   g.drawLine(startp[i][j].x,          startp[i][j].y,
endp[i][j].x, endp[i][j].y);
   g.fillPolygon(flecha_x, flecha_y, 4);
   // escribe el peso de la arista en una posicion
apropiada
   int dx = (int)(Math.abs(7*dir_y[i][j]));
   int dy = (int)(Math.abs(7*dir_x[i][j]));
   String str = new String("" + peso[i][j]);
   g.setColor(Color.black);
   if        ((startp[i][j].x>endp[i][j].x)        &&
(startp[i][j].y>=endp[i][j].y))
     g.drawString(   str,    arista[i][j].x    +    dx,
arista[i][j].y - dy);
   if       ((startp[i][j].x>=endp[i][j].x)        &&
(startp[i][j].y<endp[i][j].y))
     g.drawString(      str,       arista[i][j].x       -
fmetrics.stringWidth(str) - dx ,
                arista[i][j].y - dy);
   if        ((startp[i][j].x<endp[i][j].x)        &&
(startp[i][j].y<=endp[i][j].y))
     g.drawString(      str,       arista[i][j].x       -
fmetrics.stringWidth(str) ,
                arista[i][j].y + fmetrics.getHeight());
   if       ((startp[i][j].x<=endp[i][j].x)        &&
(startp[i][j].y>endp[i][j].y))
       g.drawString( str, arista[i][j].x + dx,
                 arista[i][j].y  +  fmetrics.getHeight()
);
   }
   // </editor-fold>

   //<editor-fold          defaultstate="collapsed"
desc="Detalles">
   public void detailsDijkstra(Graphics g, int i,
int j) {
   // Checar que arista entre nodo i y nodo j esta
cerca de la arista s para
   // Escoger durante este paso del algoritmo
   // Checar si el nodo j tiene la siguiente minima
distancia al nodo_inicial
   if ( (finaldist[i]!=-1) && (finaldist[j]==-1) ) {
     g.setColor(Color.red);
     if            (           (dist[j]==-1)            ||
(dist[j]>=(dist[i]+peso[i][j])) ) {
          if ( (dist[i]+peso[i][j])<dist[j] ) {
              changed[j]=true;
              numchanged++;
          }
        dist[j] = dist[i]+peso[i][j];
        colornodo[j]=Color.red;
        if ( (mindist==0) || (dist[j]<mindist) ) {
          mindist=dist[j];
          minstart=i;
          minend=j;
        }
      }
    }
   else g.setColor(Color.gray);
   }

   public void detailsalg(Graphics g, int i, int j) {
   // mas algoritmos pueden ser agregados
   if (algoritmo==DIJKSTRA)
      detailsDijkstra(g, i, j);
   }

   public void endstepalg(Graphics g) {
   // mas algoritmos pueden ser agregados
   if (algoritmo==DIJKSTRA)
      endstepDijkstra(g);
   if ( ( performalg ) && (mindist==0) ) {
     if (algrthm != null) algrthm.stop();
     int nalcanzable = 0;
     for (int i=0; i<numnodos; i++)
       if (finaldist[i] > 0)
       nalcanzable++;
     if (nalcanzable == 0)
       parent.showline("ninguno");
       else     if     (nalcanzable<     (numnodos-
emptyspots-1))
       parent.showline("alguno");
     else
       parent.showline("hecho");
   }
   }
   //</editor-fold>

   //<editor-fold          defaultstate="collapsed"
desc="Despliegue de distancias entre nodos">
   public void endstepDijkstra(Graphics g) {
   // despliega distancias parcial y total de los
nodos, ajusta la distancia final
   // para el nodo que tuvo la minima distancia
en este paso
   //  explica  el  algoritmo  en  el  panel  de
documentacion
      for (int i=0; i<numnodos; i++)
     if ( (nodo[i].x>0) && (dist[i]!=-1) ) {
       String str = new String(""+dist[i]);
       g.drawString(str,          nodo[i].x          -
(int)fmetrics.stringWidth(str)/2 -1,
                 nodo[i].y + h);
       if (finaldist[i]==-1) {
         neighbours++;
         if (neighbours!=1)
           showstring = showstring + ", ";
         showstring = showstring + intToString(i)
+"=" + dist[i];
```

```
        }
      }
      showstring = showstring + ". ";

      if ( (paso>1) && (numchanged>0) ) {
        if (numchanged>1)
          showstring = showstring + "Note que
las distancias a ";
        else showstring = showstring + "Note que
la distancia a ";
        for (int i=0; i<numnodos; i++)
          if ( changed[i] )
            showstring = showstring +
intToString(i) +", ";
        if (numchanged>1)
          showstring = showstring + "han
cambiado!\n";
        else showstring = showstring + "ha
cambiado!\n";
      }
      else showstring = showstring + " ";
    if (neighbours>1) {
    // si hay otros candidatos explicar porque se
tomo este
      showstring = showstring + "El nodo " +
intToString(minend) +
                  " tiene la distancia minima.\n";
      //checar sy hay otros caminos a minend.
      int newcaminos=0;
      for (int i=0; i<numnodos; i++)
        if ( (nodo[i].x>0) && (peso[i][minend]>0)
&& ( finaldist[i] == -1 ) )
          newcaminos++;
      if (newcaminos>0)
        showstring = showstring + "Cualquier otro
camino a " + intToString(minend) +
                  " visita otro nodo de la red, y sera
mas largo que " + mindist + ".\n";
        else showstring = showstring +
              "No hay otras aristas entrando a "+
              intToString(minend) + ".\n";
    }
    else {
        boolean morenodos=false;
        for (int i=0; i<numnodos; i++)
        if ( ( nodo[i].x>0 ) && ( finaldist[i] == -1
) && ( peso[i][minend]>0 ) )
          morenodos=true;
        boolean bridge=false;
        for (int i=0; i<numnodos; i++)
        if ( ( nodo[i].x>0 ) && ( finaldist[i] == -1
) && ( peso[minend][i]>0 ) )
          bridge=true;
        if ( morenodos && bridge )
          showstring = showstring + "Dado que
este nodo forma un 'puente' a "+
            "los nodos restantes,\ncualquier otro
camino a este nodo sera mas largo.\n";
        else if ( morenodos && (!bridge) )
          showstring = showstring + "Los nodos
grises restantes no son alcanzables.\n";
        else showstring = showstring + "No hay
otras aristas entrando a "+
          intToString(minend) + ".\n";
      }
    showstring = showstring + "Node " +
intToString(minend) +
      " sera coloreado naranja para indicar que " +
mindist +
      " es la longitud del camino mas corto a " +
intToString(minend) +".";
    parent.showline(showstring);
    }
    //</editor-fold>

    //<editor-fold defaultstate="collapsed"
desc="Dibujar canvas">
    @Override
    public void paint(Graphics g) {
      mindist=0;
    minnodo=MAXNODOS;
    minstart=MAXNODOS;
    minend=MAXNODOS;
      for(int i=0; i<MAXNODOS; i++)
        changed[i]=false;
      numchanged=0;
      neighbours=0;
    g.setFont(roman);
    g.setColor(Color.black);
      if (paso==1)
        showstring="Algoritmo ejecutando: las
aristas rojas apuntan a nodos alcanzables desde "
+
              " el nodo_inicial.\nLa distancia a: ";
      else
        showstring="Paso " + paso + ": Las
aristas rojsa apuntan a nodos alcanzables desde "
+
              "nodos que ya tienen una distancia
final ." +
              "\nLa distancia a: ";
    // dibuja una nueva arista en la posicion del
mouse
    if (newarista)
      g.drawLine(nodo[nodo1].x, nodo[nodo1].y,
thispoint.x, thispoint.y);
    // dibuja todas las aristas
    for (int i=0; i<numnodos; i++)
     for (int j=0; j<numnodos; j++)
       if (peso [i][j]>0) {
         // si el algoritmo se esta ejecutando
entonces hacer el siguiente paso para esta arista
```

```
            if (performalg)
              detailsalg(g, i, j);
            drawarista(g, i, j);
          }
    // si la flecha ha sido arrastyrado a  0, dibujala,
para que el usuario
    //tenga la opcion de hacerla positiva de nuevo
    if (movearista && peso[nodo1][nodo2]==0) {
      drawarista(g, nodo1, nodo2);
      g.drawLine(startp[nodo1][nodo2].x,
startp[nodo1][nodo2].y,
              endp[nodo1][nodo2].x,
endp[nodo1][nodo2].y);
    }
    // dibuja los nodos
    for (int i=0; i<numnodos; i++)
      if (nodo[i].x>0) {
        g.setColor(colornodo[i]);
        g.fillOval(nodo[i].x-NODORADIX,
nodo[i].y-NODORADIX,
            NODOSIZE, NODOSIZE);
      }
    // refleja el nodo_inicial que se mueve
    g.setColor(Color.blue);
    if (movestart)
      g.fillOval(thispoint.x-NODORADIX,
thispoint.y-NODORADIX,
            NODOSIZE, NODOSIZE);

    g.setColor(Color.black);
    // termina este paso del algoritmo
    if (performalg) endstepalg(g);
    // dibuja circulos negros alrededor de los
nodos, escribe sus nombres a la pantalla
    g.setFont(helvetica);
    for (int i=0; i<numnodos; i++)
      if (nodo[i].x>0) {
        g.setColor(Color.black);
        g.drawOval(nodo[i].x-NODORADIX,
nodo[i].y-NODORADIX,
            NODOSIZE, NODOSIZE);
        g.setColor(Color.blue);
        g.drawString(intToString(i),  nodo[i].x-14,
nodo[i].y-14);
      }
    }
    //</editor-fold>

}
```

Conclusions

In the development of this project we came to the conclusion that for a better understanding of the concepts related to graph theory and the operation of Dijkstra's algorithm, especially in students of upper secondary and higher education with a profile focused on object-oriented programming, this is facilitated by using application tools in different environments, so using the Java tool is much simpler to transfer these concepts to the different devices that users use.

References

1. Deo, N. (2016). Graph Theory with Applications to Engineering & Computer Science. Mineola, New York: Dover Publications, Inc.

2. Hutapea, Y. P., Montolalu, C. E., & Komalig, H. A. (20 de 07 de 2022). Algoritma Dijkstra untuk Penentuan Lintasan Terpendek Pada Kasus Tujuh Hotel di Kota Manado Menuju Bandara Sam Ratulangi Manado. d'CartesiaN Jurnal Matematika dan Aplikasi , 11(1), 158-163. doi:10.35799/dc.9.2.2020.29146

3. Méndez Martínez, L., Rodrígue Colina, E., & Medina Ramírez, C. (28 de 5 de 2014). Toma de Decisiones Basadas en el Algoritmo Dijkstra. Redes de Ingeniería, 4(2), 35-42. doi:10.14483/2248762X.6357

4. Miranda, P. J., & La Guardia, G. G. (2018). Trascendiendo a Teoría de Grafos. XX Semana da Física da UEPG. Ponta Grossa. doi:10.29327/XX_Semana_da_Fisica.78486

5. myservername.com. (s.f.). Cómo implementar el algoritmo de Dijkstra en Java. Obtenido de myservername.com: https://spa.myservername.com/top-8-best-free-online-schedule-maker-tools

6. Rajpal, R. (s.f.). A Review of Graph Theory in Everyday Life and Computer Science. Obtenido de Academia.edu: https://www.academia.edu/36655063/A_Review_of_Graph_Theory_in_Everyday_Life_and_Computer_Science

7. Universidad de Guanajuato. (08 de 2018). Algoritmo de Dijkstra. Obtenido de Nodo Universitario: https://nodo.ugto.mx/wp-content/uploads/2018/08/Dijkstra.pdf