Universidad de Guadalajara.
Centro Universitario de Ciencias Exactas e Ingenierías.

División de Electrónica y Computación.
Sem. de solución de problemas de algoritmia.

Sandoval Márquez Anthony Esteven
215660767

Planteamiento del problema.

Diseñar un sistema computacional que genere un grafo a partir de la imagen analizada, cada círculo de la imagen representa a un vértice y cada vértice contiene adyacencias (aristas) que lo conecta con todos los demás siempre y cuando se pueda trazar una línea recta desde un vértice (origen) a otro (destino). Cualquier figura en la imagen puede obstruir la conexión de un vértice a otro, incluso los mismos vértices.
El sistema debe tener la capacidad de añadir una partícula al grafo en un vértice específico. La partícula debe conocer los caminos más cortos, desde el vértice en el que se encuentra, a todos los demás vértices. Si la partícula ya está en el grafo (en reposo, en algún vértice), se puede seleccionar cualquier vértice, y se animará un recorrido del camino mínimo desde el vértice en el que se encuentra, hasta el vértice seleccionado.
Los recorridos podrán verse de forma animada, la partícula se desplaza desde un vértice a otro a través de sus aristas, es decir, una partícula x puede ir del vértice a al vértice b si existe una arista que conecta a los dos vértices. El desplazamiento es visualmente progresivo (se desplaza sobre la arista).

Objetivos.

- Paradigma de programación: P.O.O.
- El sistema debe contener una interfaz gráfica intuitiva.
- Mostrar la imagen a analizar.
- Mostrar una imagen que represente al grafo.
- Posibilidad de colocar una partícula en un vértice específico.
- Implementar el algoritmo de Dijkstra. (35 puntos)

- Posibilidad de mostrar gráficamente los caminos más cortos, desde el vértice donde se encuentra la partícula, a todos los demás vértices. (35 puntos)
- Mostrar la animación del recorrido de la partícula. (30 puntos)

Marco teórico.

Dijkstra

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

Algoritmo y Complejidad

Cálculo del coste del camino mínimo desde un vértice dado al resto en un grafo etiquetado con pesos no negativos – algoritmo de Dijkstra

```
{Pre: g es un grafo dirigido etiquetado no neg.}
función Dijkstra(g:grafo; v:vért)
            devuelve vector[vért] de etiq
variables S:cjtVért;
          D:vector[vért] de etiq
principio
  ∀w∈vért:D[w]:=etiqueta(g,v,w);
  D[v]:=0; S:={v};
  mq S no contenga todos los vértices hacer
  {D contiene caminos mínimos formados íntegramente
   por nodos de S (excepto el último), y los nodos de
   S corresponden a los caminos mínimos más cortos
   calculados hasta el momento}
    elegir w∉S t.q. D[w] es mínimo;
    S:=S∪{w};
    ∀u∉S:actualizar dist.mín. comprobando
         si por w hay un atajo
  fmq;
  devuelve D
fin
{Post: D=caminosMínimos(g,v)}
```

Implementación directa → $\Theta(n^2)$

Mejora usando un montículo (con op. de reducción de clave) → $\Theta((a+n)\log n)$

Bibliografía:

J. (s. f.). *Uso de colas con prioridad en el algoritmo de Dijkstra at Estructuras de Datos y Algoritmos (EDA)*. WEBDIIZ. Recuperado 19 de junio de 2021, de http://webdiis.unizar.es/asignaturas/EDA/?p=1789

Algoritmo de Dijkstra. (s. f.). I3CAMPUS. Recuperado 19 de junio de 2021, de http://i3campus.co/CONTENIDOS/wikipedia/content/a/algoritmo_de_dijkstra.html

Desarrollo.

Algoritmo y complejidad algorítmica de

- **Dijkstra:** Para la realización del código (**Apéndice 1**) me base en el algoritmo proporcionado por el profesor en el video correspondiente a la actividad el cual es el siguiente:

```
Dijkstra(G, verticeInicial )
 {
   VD = inicializaVectorPesos(G, verticeInicial)

   mientras(!solucion(VD)){
     v_d = seleccionaDefinitivo();
     VD = actualizaVD(VD,v_d);
   }
 }
```

  En cuanto a la complejidad de este corresponde a **O(|V|²),** Investigue algoritmos que usaban una cola de prioridad y que tenían una complejidad de **O(A log|V|)** pero yo utilice el que no la usa.

Animación de partícula (**Apéndice 2**)

Para animar la partícula realice casi lo mismo que el profesor explico en el video, con la diferencia de que en vez de hacer clic directamente en la imagen yo decidí reutilizar el combobox que ya tenía en desde practicas anteriores. Para la selección de origen y destino coloque unos check box lo cuales servirían para establecer si se va a elegir un vértice origen o uno destino en el combobox, cuando selecciono un origen se coloca la partícula. Teniendo ya definida esta funcionalidad solamente agregue dos botones, uno para activar la animación y otro encontrar los caminos más cercanos con Dijkstra, primero se debe aplicar el algoritmo para activar la animación.

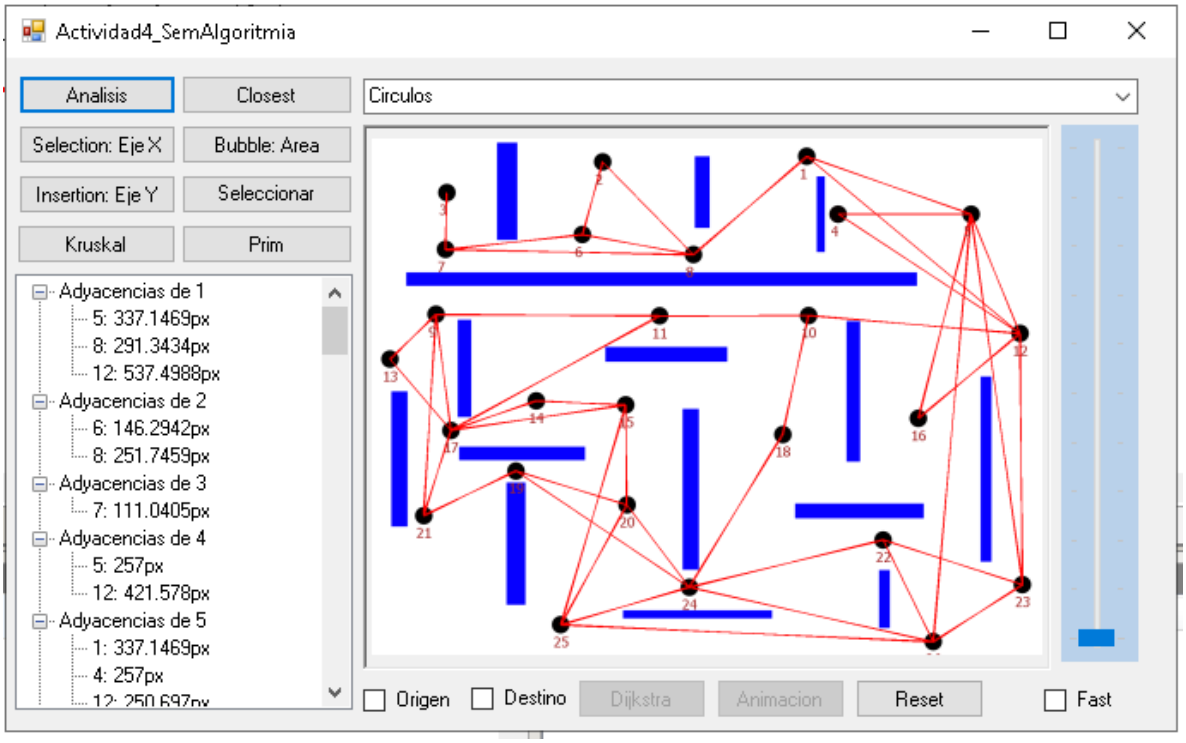
La animación de la partícula se hace a partir de cálculos y constantes actualizaciones en la imagen del picturebox

Camino más cercano (**Apéndice 3**)

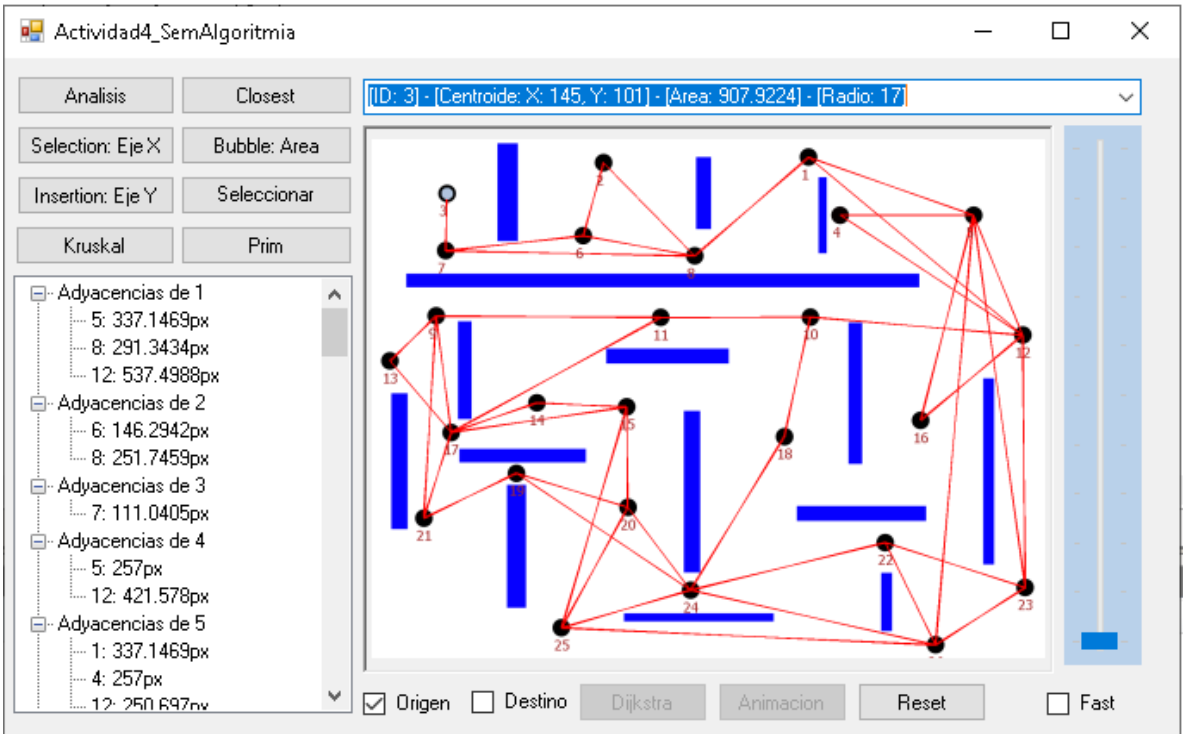
Para mostrar el camino, como después de aplicar el algoritmo ya tengo seleccionados el circulo origen, circulo destino y los datos proporcionados por Dijkstra, entonces solamente realizo una función que me pinta líneas entre los centros de los círculos que se van recorriendo al avanzar en el camino encontrado.

Pruebas y resultados.

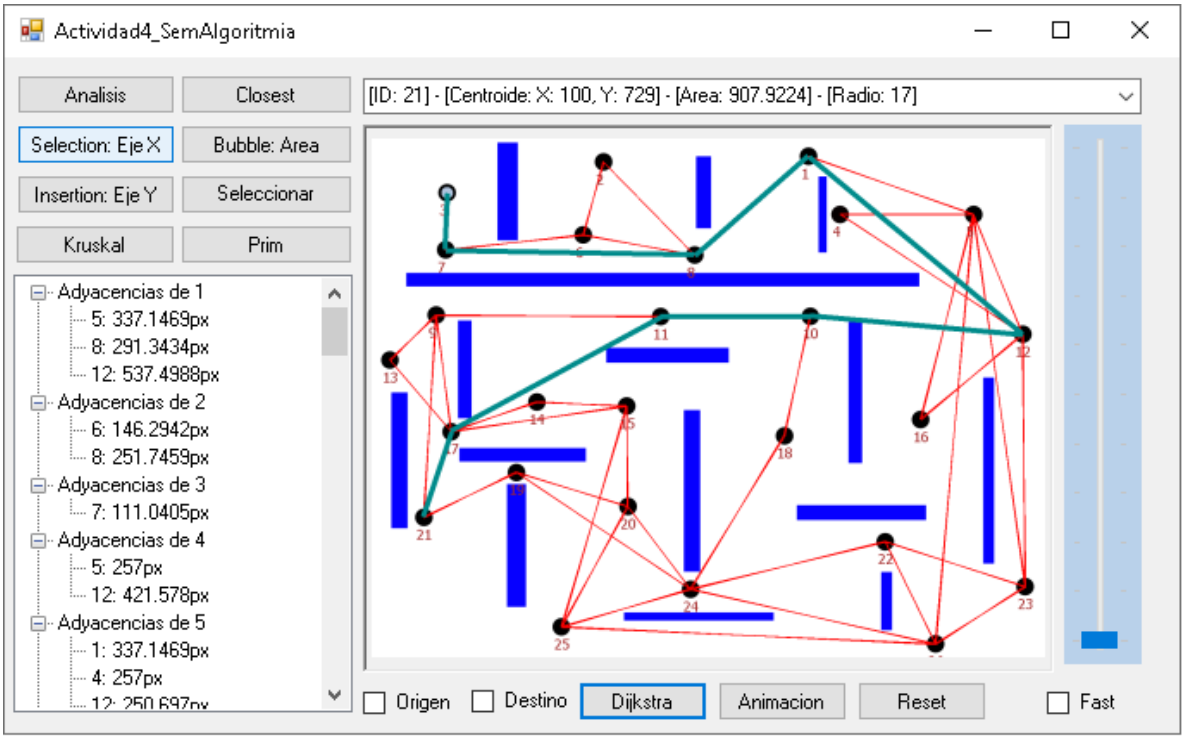
Análisis de la imagen



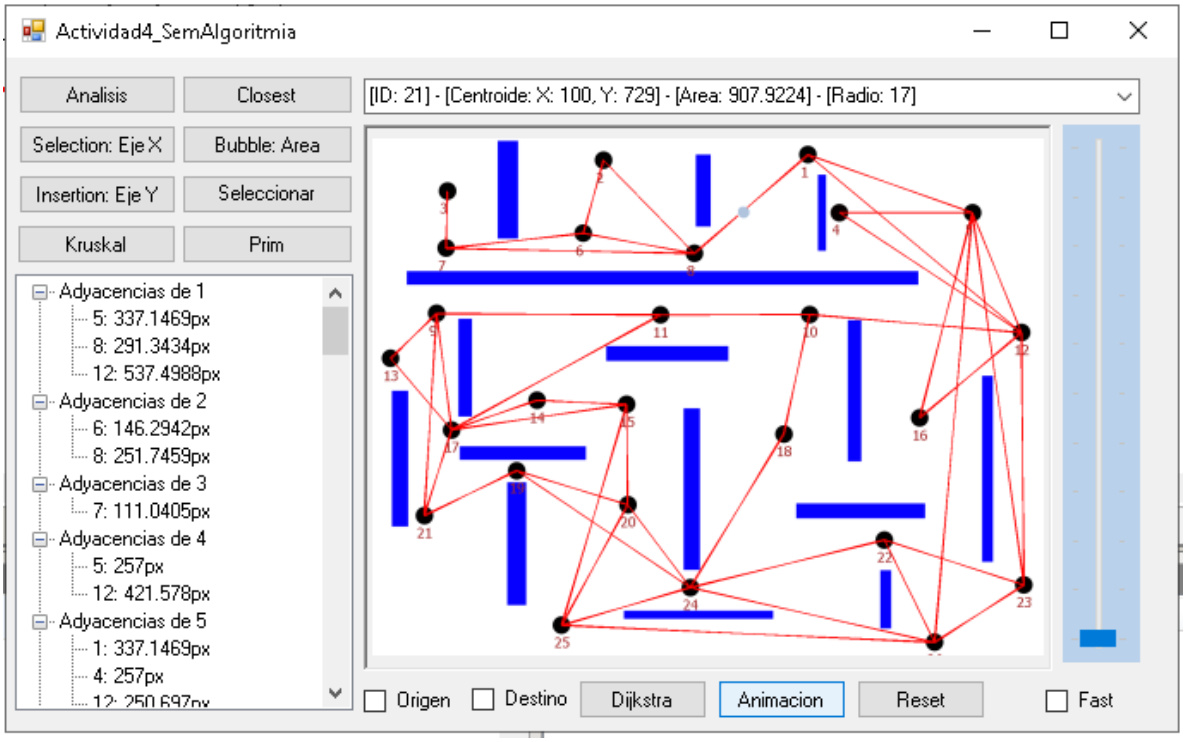
Se selecciona un vértice origen (numero 3)



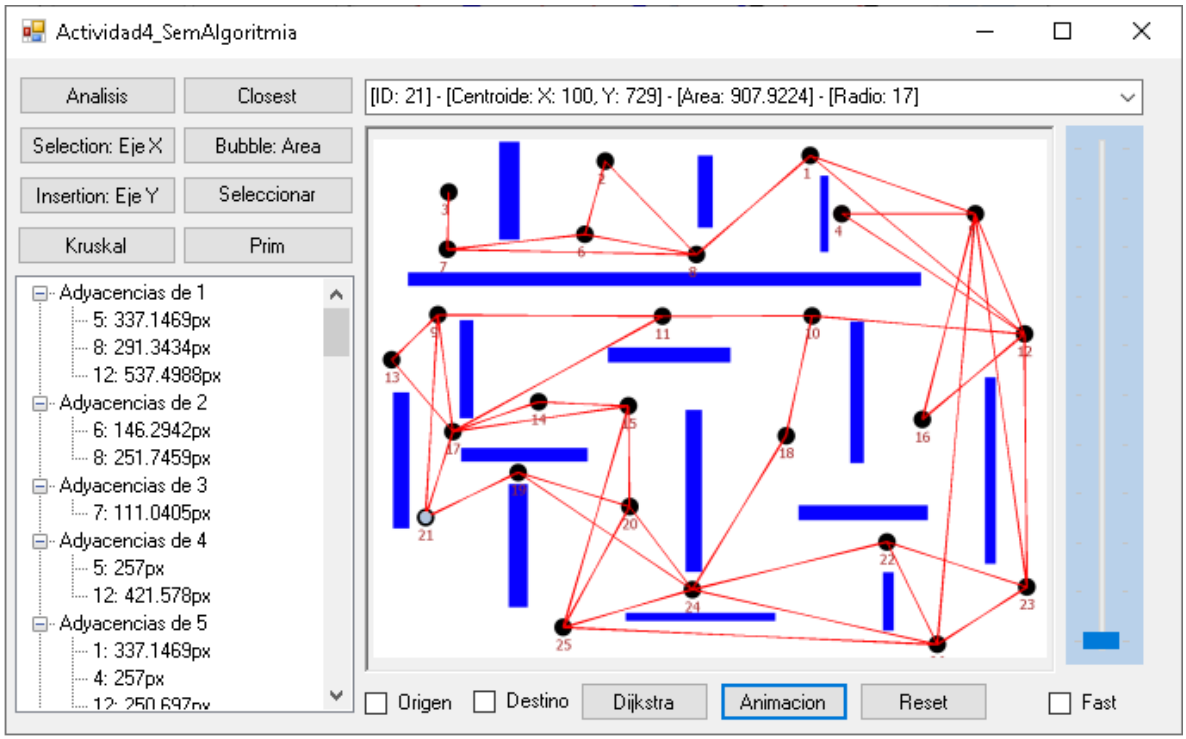
Se selecciona un vértice destino (numero 21) y clic al botón “Dijkstra”



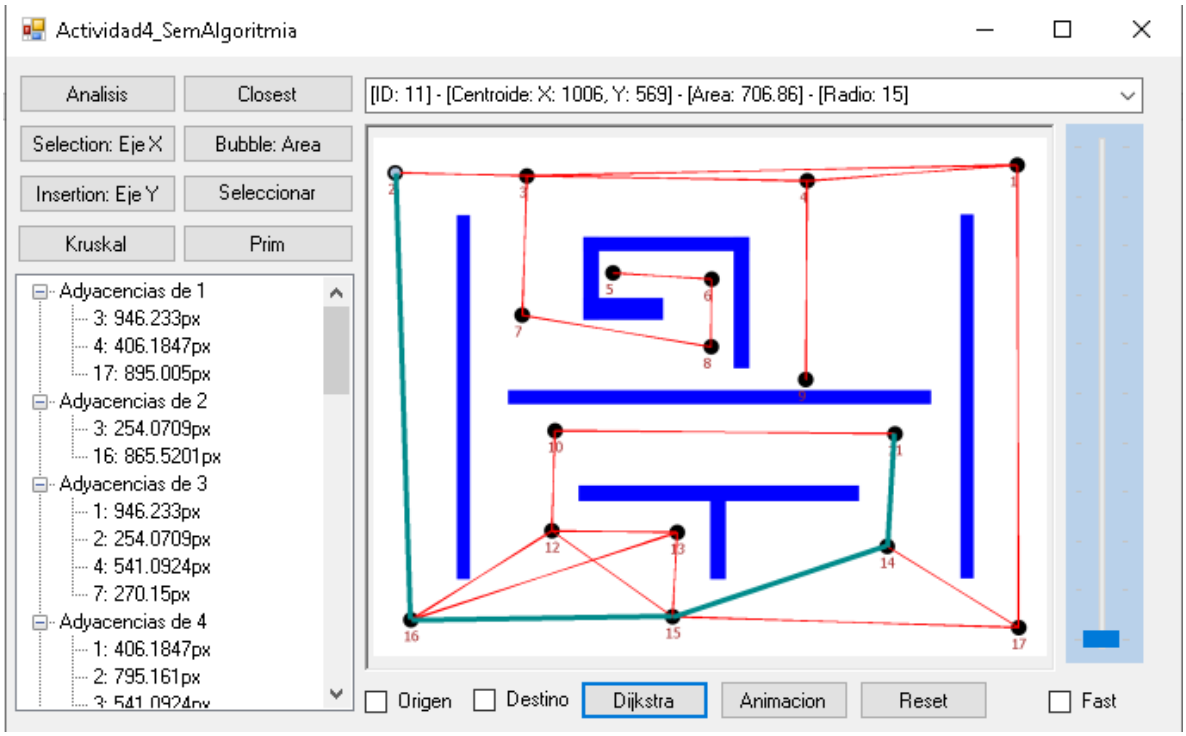
Clic en el botón de “Animación”. Partícula Avanzando:

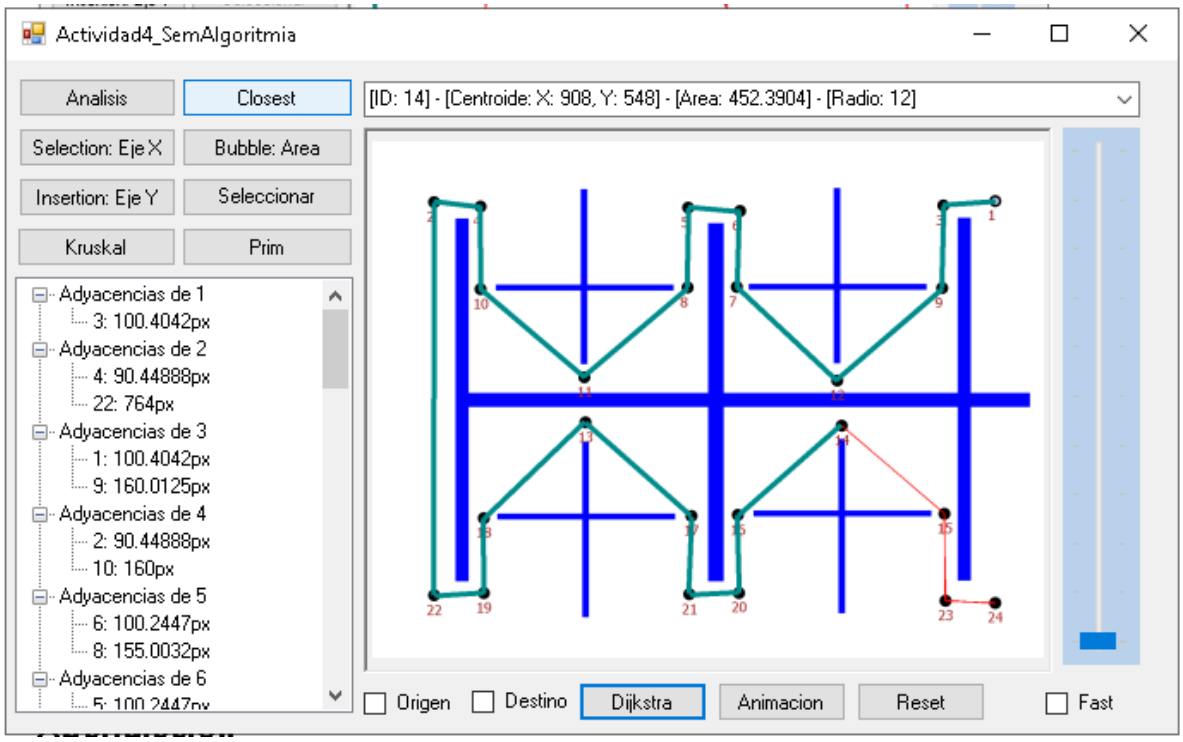


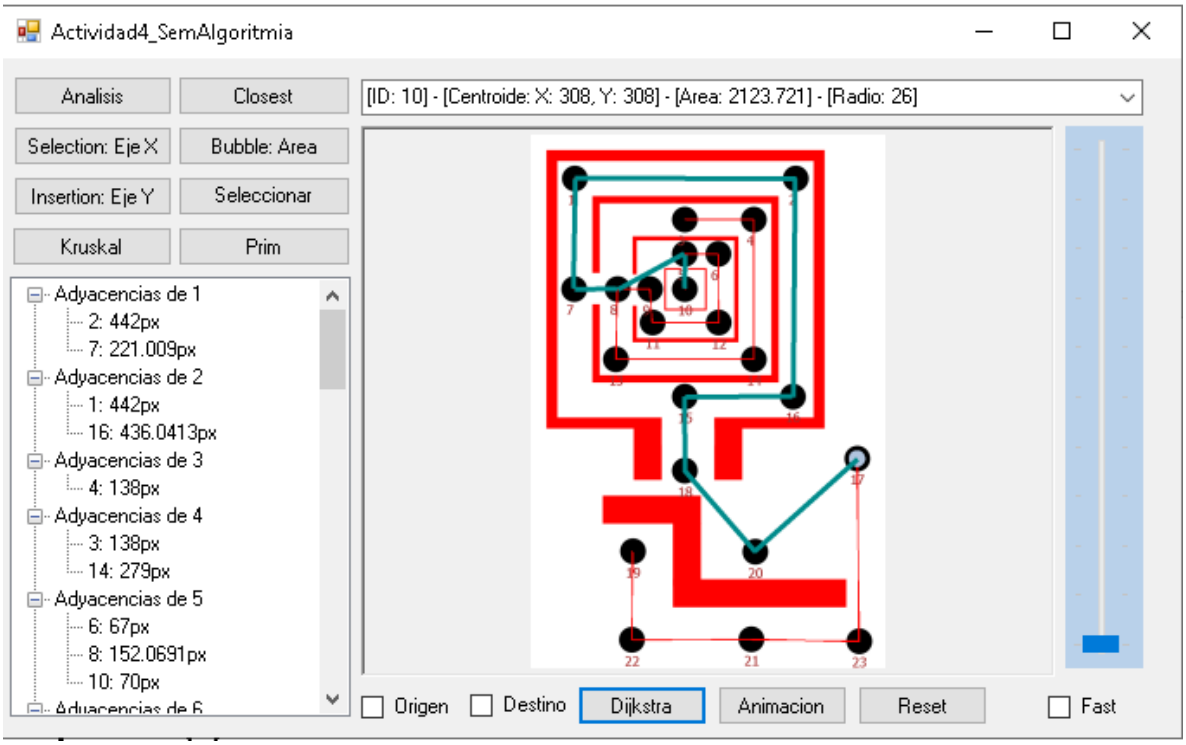
Partícula llega hasta a su destino

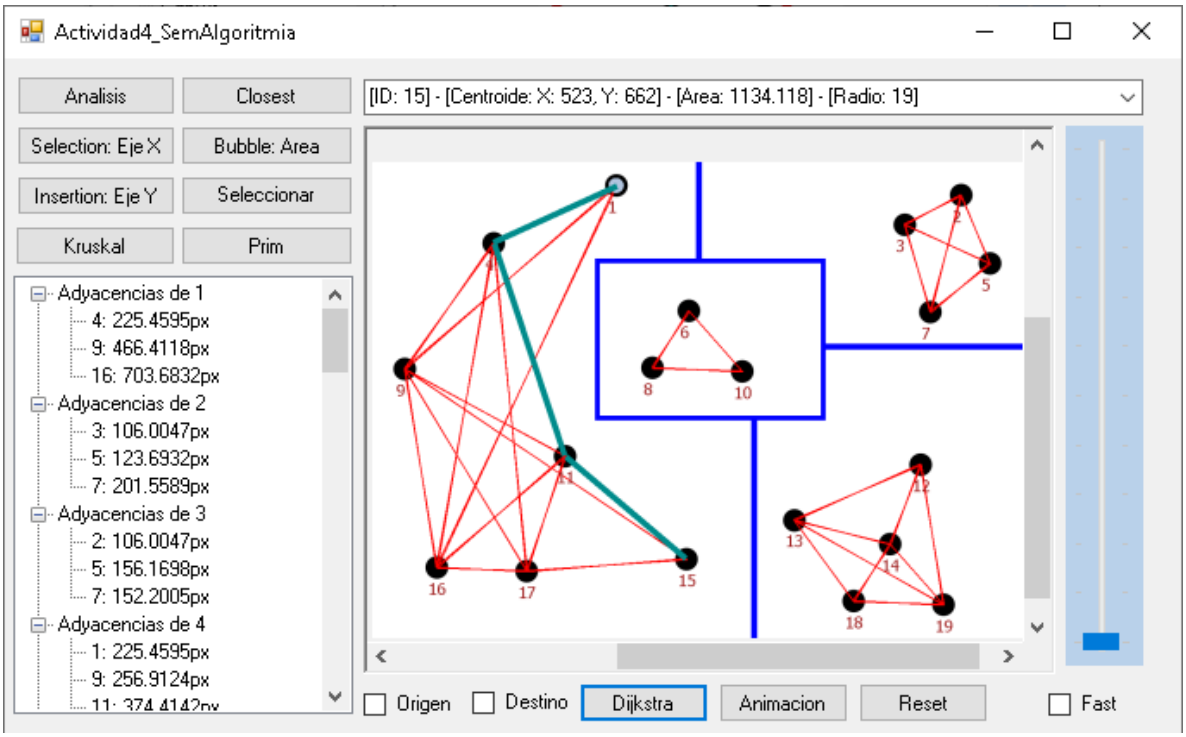


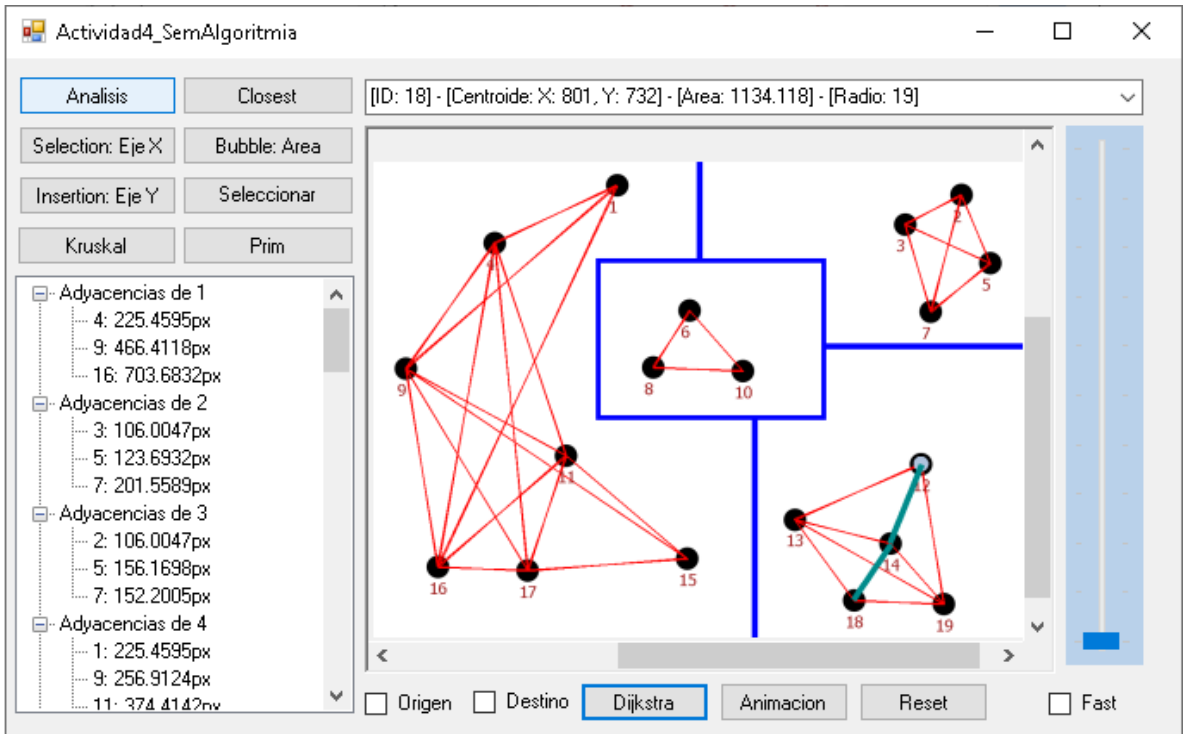
Pruebas con las otras imágenes:

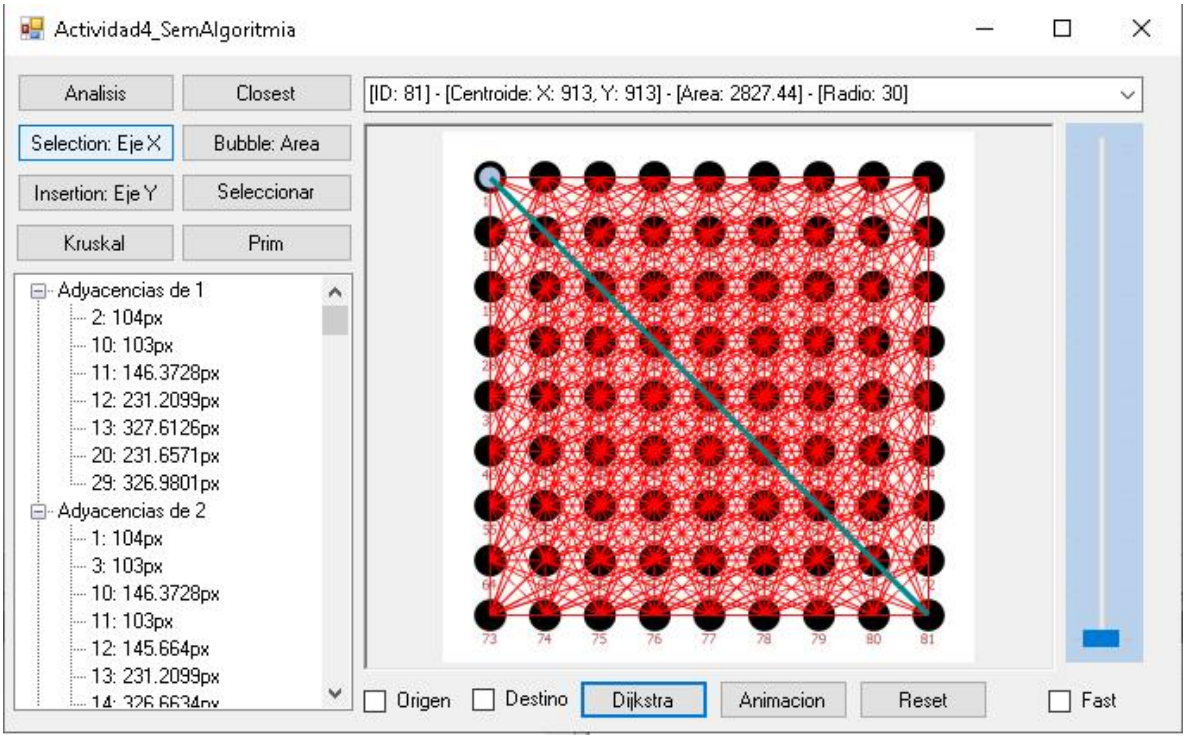


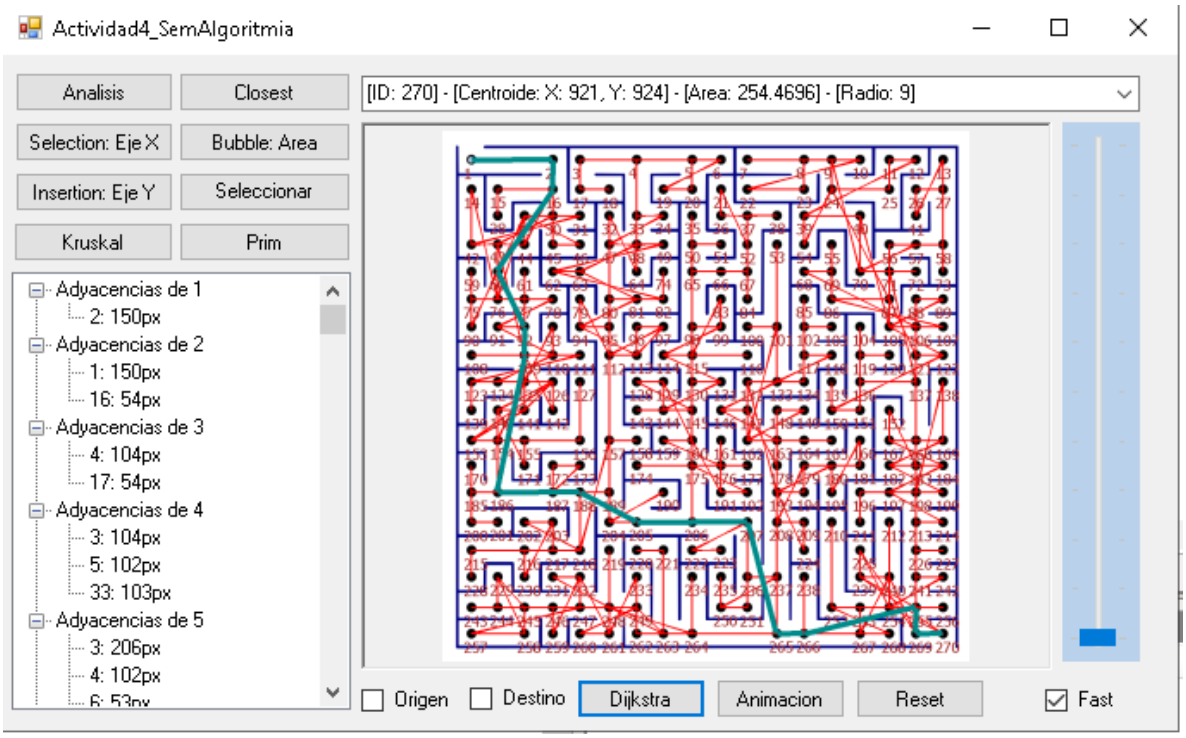


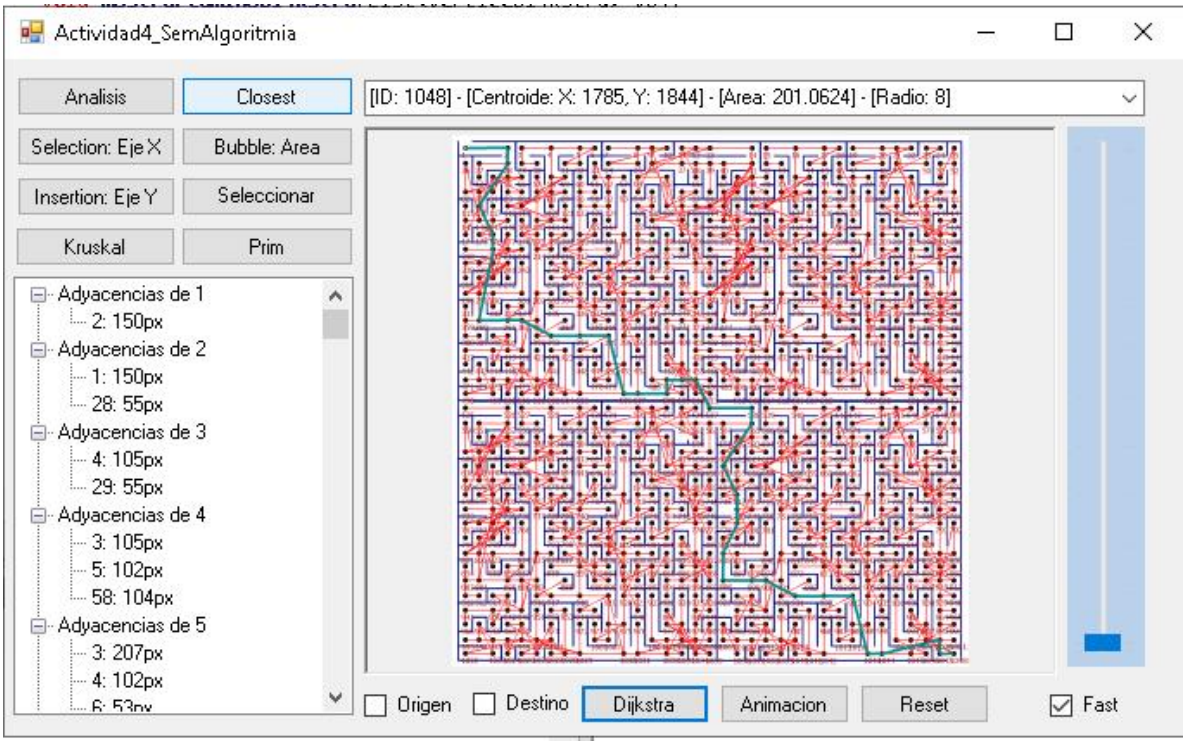

Actividad4_SemAlgoritmia
Analisis
Closest
[ID: 10] - [Centroide: X: 308, Y: 308] - [Area: 2123.721] - [Radio: 26]
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Adyacencias de 1
2: 442px
7: 221.009px
Adyacencias de 2
1: 442px
16: 436.0413px
Adyacencias de 3
4: 138px
Adyacencias de 4
3: 138px
14: 279px
Adyacencias de 5
6: 67px
8: 152.0691px
10: 70px
Origen
Destino
Dijkstra
Animacion
Reset
Fast

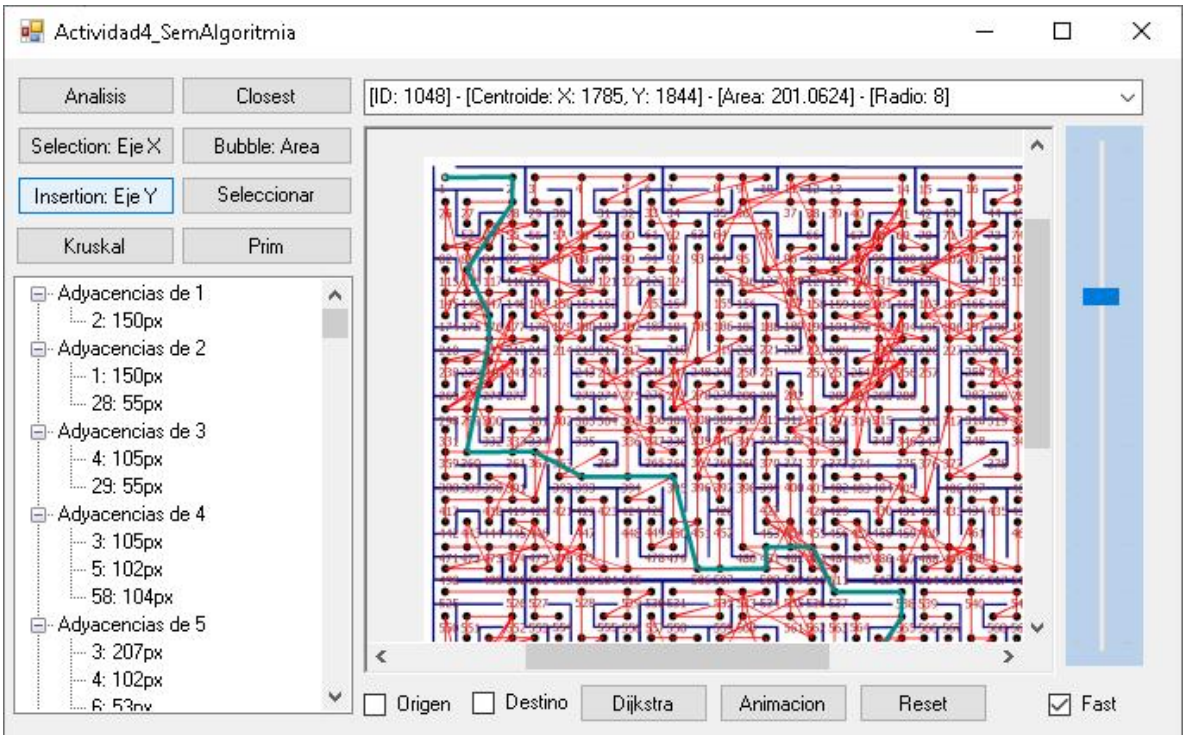

Actividad4_SemAlgoritmia
Analisis
Closest
[ID: 15] - [Centroide: X: 523, Y: 662] - [Area: 1134.118] - [Radio: 19]
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Adyacencias de 1
4: 225.4595px
9: 466.4118px
16: 703.6832px
Adyacencias de 2
3: 106.0047px
5: 123.6932px
7: 201.5589px
Adyacencias de 3
2: 106.0047px
5: 156.1698px
7: 152.2005px
Adyacencias de 4
1: 225.4595px
9: 256.9124px
Origen
Destino
Dijkstra
Animacion
Reset
Fast

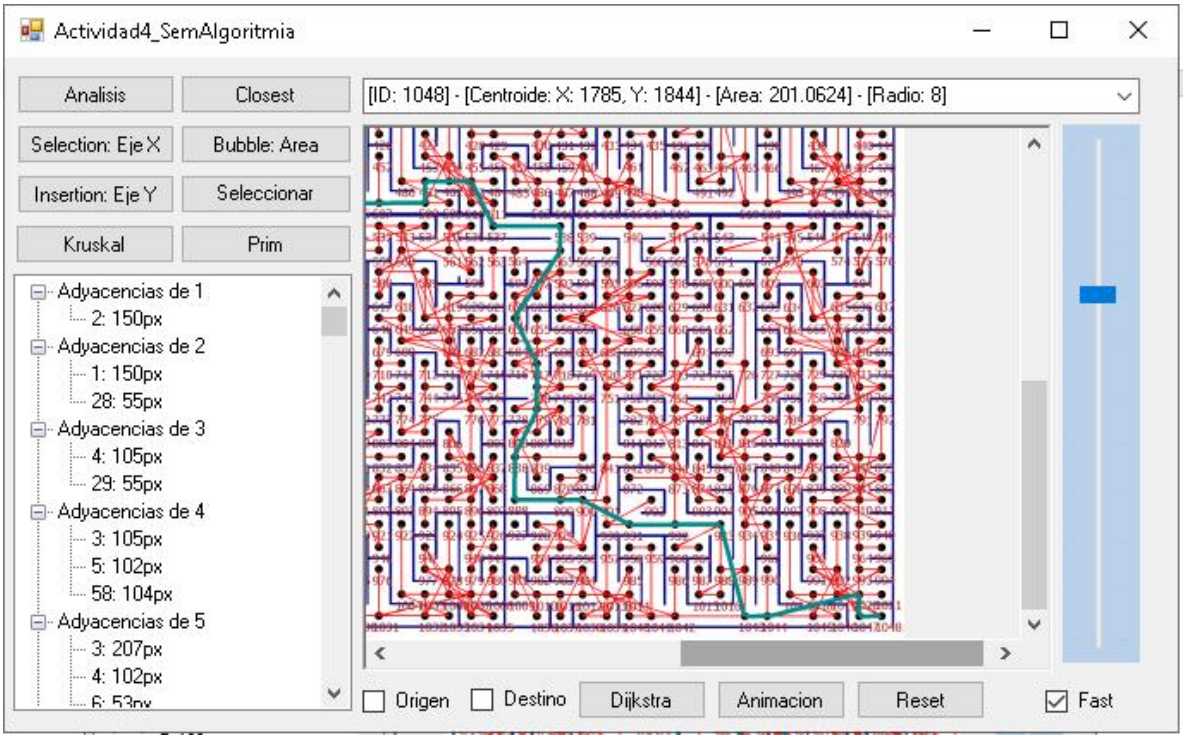

Actividad4_SemAlgoritmia
Analisis
Closest
[ID: 18] - [Centroide: X: 801, Y: 732] - [Area: 1134.118] - [Radio: 19]
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Adyacencias de 1
4: 225.4595px
9: 466.4118px
16: 703.6832px
Adyacencias de 2
3: 106.0047px
5: 123.6932px
7: 201.5589px
Adyacencias de 3
2: 106.0047px
5: 156.1698px
7: 152.2005px
Adyacencias de 4
1: 225.4595px
9: 256.9124px
Origen
Destino
Dijkstra
Animacion
Reset
Fast


Actividad4_SemAlgoritmia
Analisis
Closest
[ID: 81] - [Centroide: X: 913, Y: 913] - [Area: 2827.44] - [Radio: 30]
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Adyacencias de 1
2: 104px
10: 103px
11: 146.3728px
12: 231.2099px
13: 327.6126px
20: 231.6571px
29: 326.9801px
Adyacencias de 2
1: 104px
3: 103px
10: 146.3728px
11: 103px
12: 145.664px
13: 231.2099px
Origen
Destino
Dijkstra
Animacion
Reset
Fast


Actividad4_SemAlgoritmia
Analisis
Closest
[ID: 270] - [Centroide: X: 921, Y: 924] - [Area: 254.4696] - [Radio: 9]
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Adyacencias de 1
2: 150px
Adyacencias de 2
1: 150px
16: 54px
Adyacencias de 3
4: 104px
17: 54px
Adyacencias de 4
3: 104px
5: 102px
33: 103px
Adyacencias de 5
3: 206px
4: 102px
Origen
Destino
Dijkstra
Animacion
Reset
Fast


Actividad4_SemAlgoritmia
Analisis
Closest
[ID: 1048] - [Centroide: X: 1785, Y: 1844] - [Area: 201.0624] - [Radio: 8]
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Adyacencias de 1
2: 150px
Adyacencias de 2
1: 150px
28: 55px
Adyacencias de 3
4: 105px
29: 55px
Adyacencias de 4
3: 105px
5: 102px
58: 104px
Adyacencias de 5
3: 207px
4: 102px
Origen
Destino
Dijkstra
Animacion
Reset
Fast


Actividad4_SemAlgoritmia
Analisis
Closest
[ID: 1048] - [Centroide: X: 1785, Y: 1844] - [Area: 201.0624] - [Radio: 8]
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Adyacencias de 1
2: 150px
Adyacencias de 2
1: 150px
28: 55px
Adyacencias de 3
4: 105px
29: 55px
Adyacencias de 4
3: 105px
5: 102px
58: 104px
Adyacencias de 5
3: 207px
4: 102px
Origen
Destino
Dijkstra
Animacion
Reset
Fast


Actividad4_SemAlgoritmia
Analisis
Closest
[ID: 1048] - [Centroide: X: 1785, Y: 1844] - [Area: 201.0624] - [Radio: 8]
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Adyacencias de 1
2: 150px
Adyacencias de 2
1: 150px
28: 55px
Adyacencias de 3
4: 105px
29: 55px
Adyacencias de 4
3: 105px
5: 102px
58: 104px
Adyacencias de 5
3: 207px
4: 102px
Origen
Destino
Dijkstra
Animacion
Reset
Fast

Conclusiones.

La realización de este algoritmo me fue más fácil de implementar ya que ya tenía practica con la actividad de prim y kruskal, no era totalmente similar, pero tenía aspectos del mismo fin. En otra materia realice el algoritmo A* aplicado a un grafo y creo que Dijkstra es más fácil de implementar, aunque no estoy seguro, pero lo que si se es que Dijkstra utiliza más recursos para poder encontrar el camino más cercano ya que no solo busca uno, si no que busca todos los posibles caminos. En la realización de la animación lo único que se me dificulto fue eliminar las partículas que había puesto anteriormente, ya que aparecían cada vez que intentaba buscar otro camino, lo solucione reiniciando el bitmap pero no me convenció del todo. Me hubiera gustado que el movimiento de la partícula fuera más rápido, pero no encontré alguna forma.

Apéndice(s).

Apéndice 1. Algoritmo de Dijkstra y las funciones que utiliza

```
void BtnDijkstraClick(object sender, EventArgs e)
    {
      Dijkstra();
      btnAnimacion.Enabled = true;
    }

    void Dijkstra(){

      List<VerticeDijkstra> VD = new List<VerticeDijkstra>();
      VD = inicializarDijkstra(VD);
      List<Aristas> aristas = new List<Aristas>();
      aristas = obtenerPesos(aristas);
      VerticeDijkstra actual = new VerticeDijkstra();

      while(!solucionDijkstra(VD)){
        int indicMenor = seleccionaDefinitivo(VD);
        if(indicMenor == -1){
          break;
        }
        actual = VD[indicMenor];
        VD = actualizarVD(VD, indicMenor, aristas);
      }

      mostrarCaminoDijkstra(VD);
    }
```

```
bool solucionDijkstra(List<VerticeDijkstra> VD){
   for(int i = 0; i < VD.Count; i++){
      if(VD[i].definitivo == false){
         return false;
      }
   }
   return true;
}


List<VerticeDijkstra> inicializarDijkstra(List<VerticeDijkstra> VD){
   for(int i = 0; i < circulos.Count; i++){
      VerticeDijkstra v = new VerticeDijkstra();
      if(i == indiceOrigen){
         v.vertice = circulos[i];
         v.peso = 0;
      }else{
         v.vertice = circulos[i];
      }
      VD.Add(v);
   }
   return VD;
}


List<Aristas> obtenerPesos(List<Aristas> aristas){
   Ordenamientos ordenar =  new Ordenamientos();//para ordenar las aristas
   aristas = grafo.getAristas();//todas las aristas del grafo
   aristas = elimininarRepetidos(aristas);//Eliminino aristas repetidas, aristas no dirigidas. Solo
quedan dirigidas
   return aristas;
}

int seleccionaDefinitivo(List<VerticeDijkstra> VD){
   int indice_menor = menorNoDefinitivo(VD);
   if(indice_menor != -1){
      VD[indice_menor].definitivo = true;
   }

   return indice_menor;
}

int menorNoDefinitivo(List<VerticeDijkstra> VD){
```

```
        double menor = Single.PositiveInfinity;
        int indice = -1;
        for(int i  = 0; i < VD.Count; i++){
          if(VD[i].definitivo == false){
            //Debug.WriteLine("Minimo: " + VD[i].vertice.getId() + ", " + VD[i].peso);
            if(menor > VD[i].peso){
              //Debug.WriteLine("Menor: " + VD[i].vertice.getId());
              menor = VD[i].peso;
              indice = i;
            }
          }
        }
        return indice;
      }

      List<VerticeDijkstra> actualizarVD(List<VerticeDijkstra> VD, int indiceMenor, List<Aristas> aris
tas){
        List<Aristas> aristasActual = new List<Aristas>();
        double pesoActual = VD[indiceMenor].peso;
        VerticeDijkstra v = VD[indiceMenor];
        aristasActual = getAristas(aristas, aristasActual, v);

        for(int i = 0; i < aristasActual.Count; i++){
          double pesoTotal = pesoActual + aristasActual[i].arista;
          //Debug.WriteLine("Origen: " + v.vertice.getId() + ", Destino: " +
aristasActual[i].destino.getId());
          int indiceD = buscareEnVD(VD, aristasActual[i].destino);
          if(indiceD != -1){
            //Debug.WriteLine("PesoO: " + VD[indiceD].peso + ", PesoT: " + pesoTotal);
            if(VD[indiceD].peso > pesoTotal){
              VD[indiceD].peso = pesoTotal;
              VD[indiceD].padre = v.vertice;
            }

          }

        }
        return VD;
      }

      //Con esta funcion obtengo los cadidatos del vertice que se recibe, candidatos son las aristas
adyacentes
      List<Aristas> getAristas(List<Aristas> aristas, List<Aristas> aristasActual, VerticeDijkstra
actual){
```

```
        for(int i = 0; i < aristas.Count; i++){
            if(aristas[i].origen.getId() == actual.vertice.getId()){
                aristasActual.Add(aristas[i]);
            }
            if(aristas[i].destino.getId() == actual.vertice.getId()){
                Aristas a = new Aristas(aristas[i].destino, aristas[i].origen, aristas[i].arista);
                aristasActual.Add(a);
            }
        }
        return aristasActual;
    }

    int buscareEnVD(List<VerticeDijkstra> VD, Circulo c){
        for(int i = 0; i < VD.Count; i++){
            if(c != null){
                if(VD[i].vertice.getId() == c.getId()){
                    return i;
                }
            }
        }
        return -1;
    }
```

Apéndice 2. Animación de partícula

```
void dibujarOrigen(){
        if(checkBoxFast.Checked == false){
            bmpGlobal = new Bitmap(c1);
            drawLines(bmpGlobal);
            etiquetasCirculos(70, 50, bmpGlobal, 20);
        }
        imagen_PBox.Image = bmpGlobal;
        float r = circulos[actualCicleIndex].getRadio()/1.5f;
        int x = circulos[actualCicleIndex].getCentroX();
        int y = circulos[actualCicleIndex].getCentroY();
        Brush brush = new SolidBrush(Color.LightSteelBlue);
        Graphics g = Graphics.FromImage(bmpGlobal);
        g.FillEllipse(brush, (int)(Math.Round(x-r)), (int)(Math.Round(y-r)), 2*r, 2*r);
    }

    void animarParticula(Circulo origen, Circulo destino){
        float r = origen.getRadio()/1.5f;
        bmpAnimacion = new Bitmap(bmpGlobal.Width, bmpGlobal.Height);
        imagen_PBox.Image = bmpAnimacion;
        Brush brush = new SolidBrush(Color.LightSteelBlue);
```

```
        Brush brushBlanco = new SolidBrush(Color.White);
        Graphics g = Graphics.FromImage(bmpAnimacion);
        Graphics gGlobal = Graphics.FromImage(bmpGlobal);
        float x = origen.getCentroX();
        float y = destino.getCentroY();
        float xAnt = 0;
        float yAnt = 0;

        float m = 0, b = 0;
        int inc = 0;
        float xI = origen.getCentroX();
        float yI = origen.getCentroY();
        float xF = destino.getCentroX();
        float yF = destino.getCentroY();

        if(xF == xI){
           m = (yF - yI);
        }else{
           m = (yF - yI)/(xF - xI);
        }
        b = yF - xF * m;

        inc = 1;
        yAnt = yI;
        xAnt = xI;

        if(m > -1 && m < 1){
           if(xF < xI){
              inc = -1;
           }
           for(x = xI; x != xF; x += inc){
              y = m*x+b;
              g.FillEllipse(brushBlanco, (int)(Math.Round(x-inc-r)), (int)(Math.Round(yAnt-
r)), 2*r, 2*r);
              g.FillEllipse(brush, (int)(Math.Round(x-r)), (int)(Math.Round(y-r)), 2*r, 2*r);
              //imagen_PBox.Refresh();
              imagen_PBox.Image = bmpAnimacion;
              imagen_PBox.Update();
              yAnt = y;
              g.Clear(Color.Transparent);
           }
        }else{
           if(yF < yI){
              inc = -1;
           }
           for(y = yI; y != yF; y += inc){
              x = (y - b) / m;
              g.FillEllipse(brushBlanco, (int)(Math.Round(xAnt-r)), (int)(Math.Round(y-inc-
```

```
r)), 2*r, 2*r);
                g.FillEllipse(brush, (int)(Math.Round(x-r)), (int)(Math.Round(y-r)), 2*r, 2*r);
                //imagen_PBox.Refresh();
                imagen_PBox.Image = bmpAnimacion;
                imagen_PBox.Update();
                xAnt = x;
                g.Clear(Color.Transparent);
              }
            }

        }


        void BtnResetClick(object sender, EventArgs e)
        {
            resetBitmap(bmpGlobal);
            resetBitmap(bmpAnimacion);
        }

        void resetBitmap(Bitmap bmp){
            bmp = new Bitmap(c1);
            drawLines(bmp);
            etiquetasCirculos(70, 50, bmp, 20);
            imagen_PBox.Image = bmp;
            indiceOrigen = -1;
            indiceDestino = -1;
            btnAnimacion.Enabled = false;
            btnDijkstra.Enabled = false;
        }

        void CheckOrigenCheckedChanged(object sender, EventArgs e)
        {
            if(checkOrigen.Checked == true){
                checkDestino.Checked = false;
            }
        }

        void CheckDestinoCheckedChanged(object sender, EventArgs e)
        {
            if(checkDestino.Checked == true){
                checkOrigen.Checked = false;
            }
        }

    void BtnAnimacionClick(object sender, EventArgs e)
        {
            for(int i = caminosAnimacion.Count-1; i >= 0; i--){
                animarParticula(caminosAnimacion[i].padre, caminosAnimacion[i].vertice);
```

```
        }
    }
```

Apéndice 3. Camino más cercano

```
void mostrarCaminoDijkstra(List<VerticeDijkstra> VD){
        caminosAnimacion.Clear();
        int indiceSeleccionado = buscareEnVD(VD, circulos[indiceDestino]);
        int indiceO = buscareEnVD(VD, circulos[indiceOrigen]);
        VerticeDijkstra seleccionado = VD[indiceSeleccionado];
        VerticeDijkstra origen = VD[indiceO];

        while(seleccionado.vertice.getId() != origen.vertice.getId()){
          Debug.Write(seleccionado.vertice.getId() + " <- ");
          if(seleccionado.padre != null){
            drawLineDijkstra(seleccionado.vertice, seleccionado.padre, Color.DarkCyan, 10);
            caminosAnimacion.Add(seleccionado);
          }
          indiceSeleccionado = buscareEnVD(VD, seleccionado.padre);
          if(indiceSeleccionado != -1){
            seleccionado = VD[indiceSeleccionado];
          }else{
            Debug.WriteLine("Camino no encontrado");
            break;
          }

        }
        Debug.Write(seleccionado.vertice.getId());
        imagen_PBox.Image = bmpGlobal;
        imagen_PBox.Update();
    }

void drawLineDijkstra(Circulo origen, Circulo destino, Color c, int tam){
        Graphics g = Graphics.FromImage(bmpGlobal);
        Pen p = new Pen(c, tam);
        g.DrawLine(p, origen.getCentroX()-1, origen.getCentroY()-
1, destino.getCentroX()+1, destino.getCentroY()+1);
    }
```