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.

Desarrollar un sistema computacional con la capacidad de crear grafos a partir del análisis de imágenes. Las imágenes que se probarán contienen círculos negros y otras figuras de cualquier otro color (excepto negro). Para la construcción del grafo, los círculos representarán vértices, estos tienen la información del círculo, como su coordenada y radio. Las aristas entre vértices existirán solamente si desde el centro del circulo que representa el vértice “origen”, se puede trazar una línea recta hasta el centro del círculo que representa otro vértice “destino” sin ser obstruida por ninguna otra figura en la imagen. El sistema debe tener la capacidad de crear y mostrar un árbol de recubrimiento mínimo (ARM) del grafo. Si el grafo no es conexo, debe mostrar el conjunto de ARMs para cada subgrafo conexo, además, debe identificar el número de subgrafos que contiene. Por motivos de evaluación, el ARM deberá generarse con los algoritmos Prim y el de Kruskal.

Objetivos.

Posibilidad de crear, con el algoritmo de Kruskal, un ARM o un “bosque de recubrimiento mínimo” en caso de que el grafo no sea conexo.

Posibilidad de crear, con el algoritmo de Prim (el usuario selecciona, en tiempo real, el vértice inicial), un ARM o un “bosque de recubrimiento mínimo” en caso de que el grafo no sea conexo.

Mostrar la cantidad de árboles creados (en el caso de que el grafo no sea conexo) con el algoritmo de Kruskal.

Mostrar la cantidad de árboles creados (en el caso de que el grafo no sea conexo) con el algoritmo de Prim. (10 puntos)

Mostrar gráficamente un ARM de un grafo

El sistema debe tener la capacidad de comparar las salidas de ambos algoritmos (vértices y aristas del árbol, y peso total del árbol) gráficamente.

Mostrar el orden de la selección de aristas para la generación de ARMs con Prim.

Mostrar el orden de la selección de aristas para la generación de ARMs con Kruskal.

Marco teórico.

Árbol de expansión mínima

El árbol de expansión de peso mínimo es aquel que comienza desde un vértice y encuentra todos sus nodos accesibles y las relaciones en conjunto que permiten que se conecten dichos nodos con el menor peso posible.

Algoritmos voraces

Los algoritmos voraces suelen ser bastante simples. Se emplean sobre todo para resolver problemas de optimización, como, por ejemplo, encontrar la secuencia óptima para procesar un conjunto de tareas por un computador, hallar el camino mínimo de un grafo, etc. Habitualmente, los elementos que intervienen son:

- Un conjunto o lista de candidatos (tareas a procesar, vértices del grafo, etc);
- Un conjunto de decisiones ya tomadas (candidatos ya escogidos);
- Una función que determina si un conjunto de candidatos es una solución al problema (aunque no tiene por qué ser la óptima);
- Una función que determina si un conjunto es completable, es decir, si añadiendo a este conjunto nuevos candidatos es posible alcanzar una solución al problema, suponiendo que esta exista;
- Una función de selección que escoge el candidato aún no seleccionado que es más prometedor;
- Una función objetivo que da el valor/coste de una solución (tiempo total del proceso, la longitud del camino, etc) y que es la que se pretende maximizar o minimizar;

Algoritmo de Kruskal

El algoritmo de Kruskal, dado un grafo conexo, no dirigido y ponderado, encuentra un árbol de expansión mínima. Es decir, es capaz de encontrar un subconjunto de las aristas que formen un árbol que incluya todos los vértices del grafo inicial, donde el peso total de las aristas del árbol es el mínimo posible.

Funcionamiento del algoritmo de Kruskal

1. Se selecciona, de entre todas las aristas restantes, la de menor peso siempre que no cree ningún ciclo.
2. Se repite el paso 1 hasta que se hayan seleccionado |V| - 1 aristas.

Siendo V el número de vértices.

```
función Kruskal(G)
  Para cada v en V[G] hacer
    Nuevo conjunto C(v) ← {v}.
  Nuevo heap Q que contiene todas las aristas de G, ordenando por su peso
  Defino un árbol T ← Ø
  // n es el número total de vértices
  Mientras T tenga menos de n-1 aristas y !Q.vacío() hacer
    (u,v) ← Q.sacarMin()
    // previene ciclos en T. agrega (u,v) si u y v están
    // diferentes componentes en el conjunto.
    // Nótese que C(u) devuelve la componente a la que pertenece u
    Si C(v) ≠ C(u) hacer
      Agregar arista (v,u) a  T
      Merge C(v) y C(u) en el conjunto
  Responder árbol T
```

Algoritmo de Prim

El algoritmo de Prim, dado un grafo conexo, no dirigido y ponderado, encuentra un árbol de expansión mínima. Es decir, es capaz de encontrar un subconjunto de las aristas que formen un árbol que incluya todos los vértices del grafo inicial, donde el peso total de las aristas del árbol es el mínimo posible.

Funcionamiento del algoritmo de Prim

1. Se marca un vértice cualquiera. Será el vértice de partida.
2. Se selecciona la arista de menor peso incidente en el vértice seleccionado anteriormente y se selecciona el otro vértice en el que incide dicha arista.
3. Repetir el paso 2 siempre que la arista elegida enlace un vértice seleccionado y otro que no lo esté. Es decir, siempre que la arista elegida no cree ningún ciclo.
4. El árbol de expansión mínima será encontrado cuando hayan sido seleccionados todos los vértices del grafo.

```
Prim (Grafo G)
    /* Inicializamos todos los nodos del grafo.
    La distancia la ponemos a infinito y el padre de cada nodo a NULL
     Encolamos, en una cola de prioridad
                donde la prioridad es la distancia,
            todas las parejas <nodo, distancia> del grafo*/
    por cada u en V[G] hacer
        distancia[u] = INFINITO
        padre[u] = NULL
        Añadir(cola,<u, distancia[u]>)
    distancia[u]=0
    Actualizar(cola,<u, distancia[u]>)
    mientras !esta_vacia(cola) hacer
        // OJO: Se entiende por mayor prioridad aquel nodo cuya distancia[u] es menor.
        u = extraer_minimo(cola) //devuelve el mínimo y lo elimina de la cola.
        por cada v adyacente a 'u' hacer
            si ((v ∈ cola) && (distancia[v] > peso(u, v)) entonces
                padre[v] = u
                distancia[v] = peso(u, v)
                Actualizar(cola,<v, distancia[v]>)
```

Bibliografía:

Algoritmo de Kruskal - Complejidad Algorítmica. (s. f.). Google Sites. Recuperado 30 de mayo de 2021, de https://sites.google.com/site/complejidadalgoritmicaes/kruskal

Algoritmo de Prim - Complejidad Algorítmica. (s. f.). Google Sites. Recuperado 31 de mayo de 2021, de https://sites.google.com/site/complejidadalgoritmicaes/prim

O. (2019, 29 octubre). *Árbol de expansión de peso mínimo*. GraphEverywhere. https://www.grapheverywhere.com/arbol-de-expansion-de-peso-minimo/#:%7E:text=El%20%C3%A1rbol%20de%20expansi%C3%B3n%20de,con%20el%20menor%20peso%20posible.

Desarrollo.

Para poder realizar los ejercicios me puse a analizar el video que nos proporcionó el profesor en el que explicaba los algoritmos de prim y kruskal. Para poder desarrollar kruskal necesite de la creación de una clase llamada AristasKruskal la cual me guardaba un vértice origen, destino y el peso de la arista. Gracias a la creación de esta clase me fue bastante sencillo poder hacer el algoritmo de kruskal, pero además también me funciono para aplicar el de prim. La representación de los algoritmos fue sencilla, pues solo seguí los pasos que el profesor nos mostró, lo más complicado de esto para mí fue recorrer el grafo de tal manera que se pudieran encontrar más árboles en caso de que el grafo fuera no conexo.

En el caso de kruskal, para encontrar los demás arboles lo que hice fue simplemente guardar las aristas de todo el grafo mediante un recorrido de adyacencias, al tener todas las aristas ya solamente aplique una función que me eliminaba las repetidas pero tomando en cuanta también los vértices de origen y destino ya que podría dar la coincidencia de que existiera una arista del mismo peso pero con diferentes

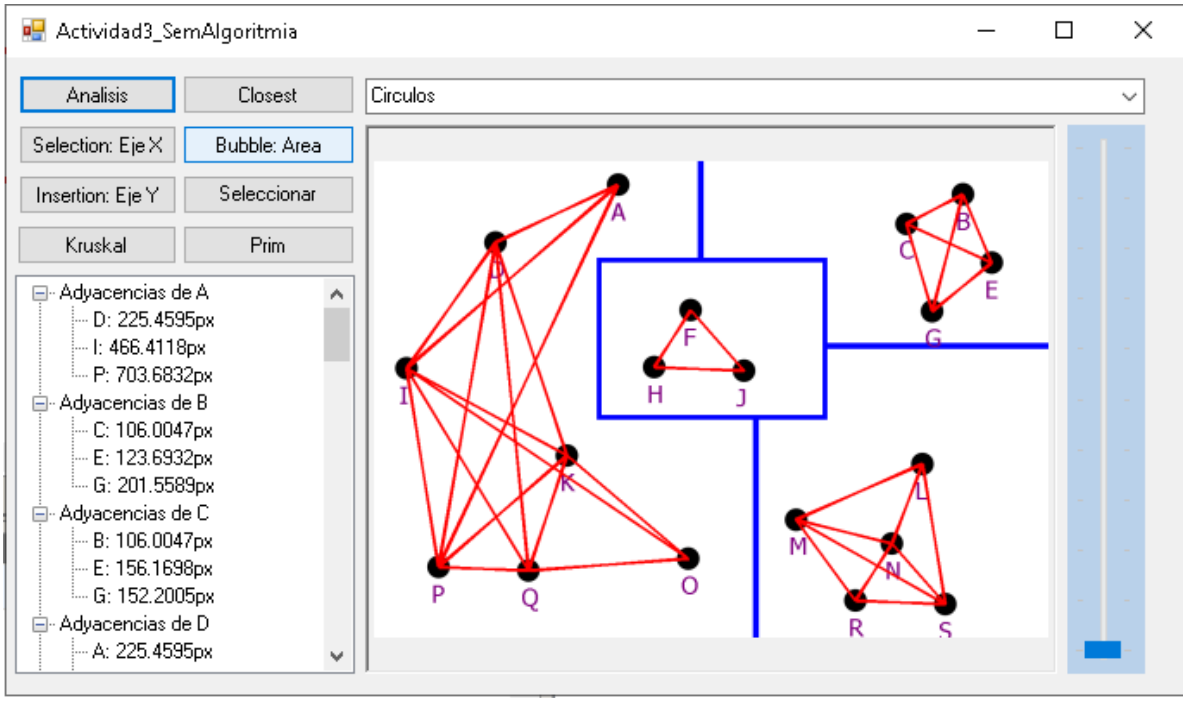
vértices, respecto a esta situación no tuve problemas ya que realice bien las comprobaciones. Teniendo las aristas correctas ya solo las ordené de menor a mayor y procedí a aplicar el algoritmo.

Para prim realice algo diferente para encontrar los arboles ya que en un principio no se tenían a todos los candidatos (aristas y vértices), se iniciaba con las aristas respecto a un vértice seleccionado, por lo tanto en esta situación decidí hacer una función recursiva que al final, después de aplicar el algoritmo, me comprueba si hay algún elemento que fuera de los candidatos que no exista, para esto reutilice la función que use en kruskal para obtener a todas las aristas y vértices existentes, con estos ya solo comparo si hay alguno que no existe en mi conjunto de vértices actuales, si no se encuentra pues entonces significa que el vértice pertenece a otro árbol y por consecuencia se vuelve a llamar a la misma función para que obtenga las aristas candidatas del vértice nuevo encontrado.

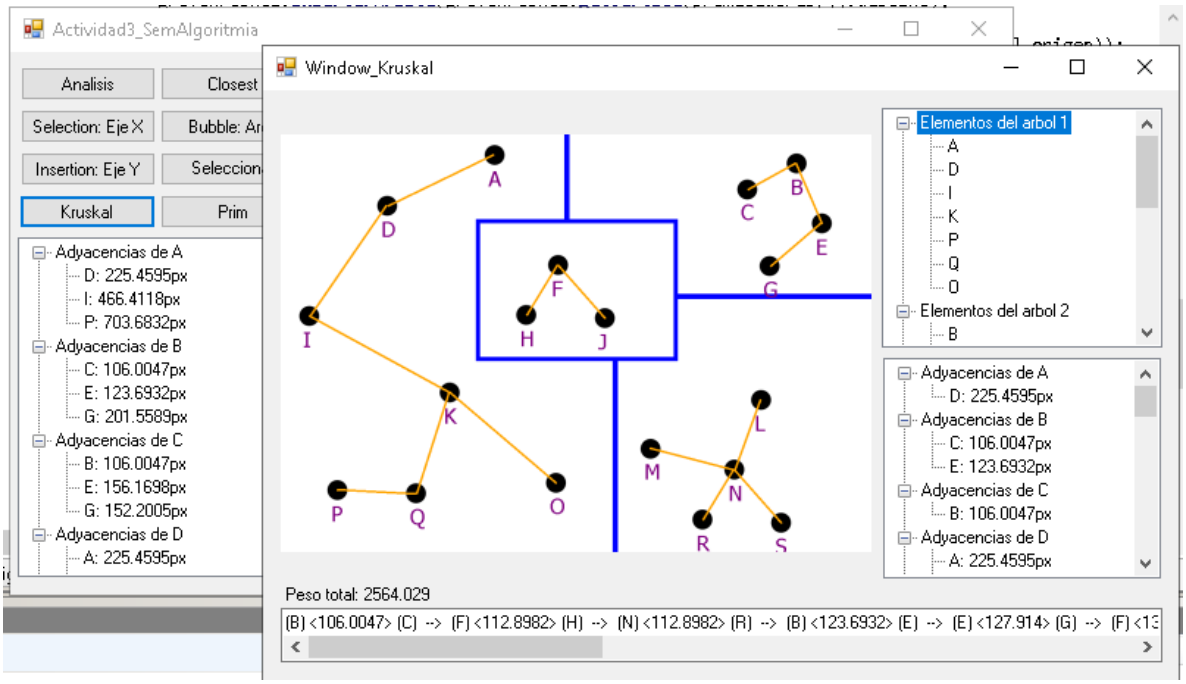
Al momento de comparar ambas salidas de los algoritmos lo realizo mediante otras dos nuevas ventanas que surgen al presionar los respectivos botones en las que se muestra el árbol resultante y en dos TreeView muestro los arboles generados y la lista de adyacencias del nuevo árbol.

Pruebas y resultados.

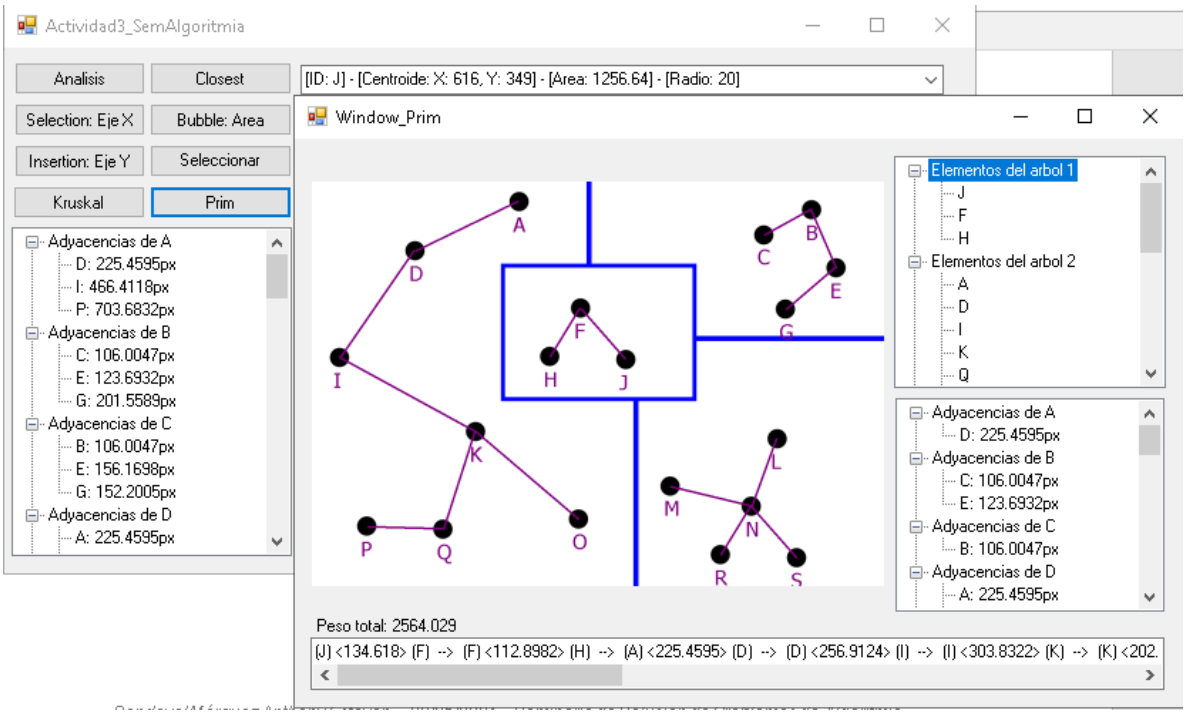
Análisis de la imagen.



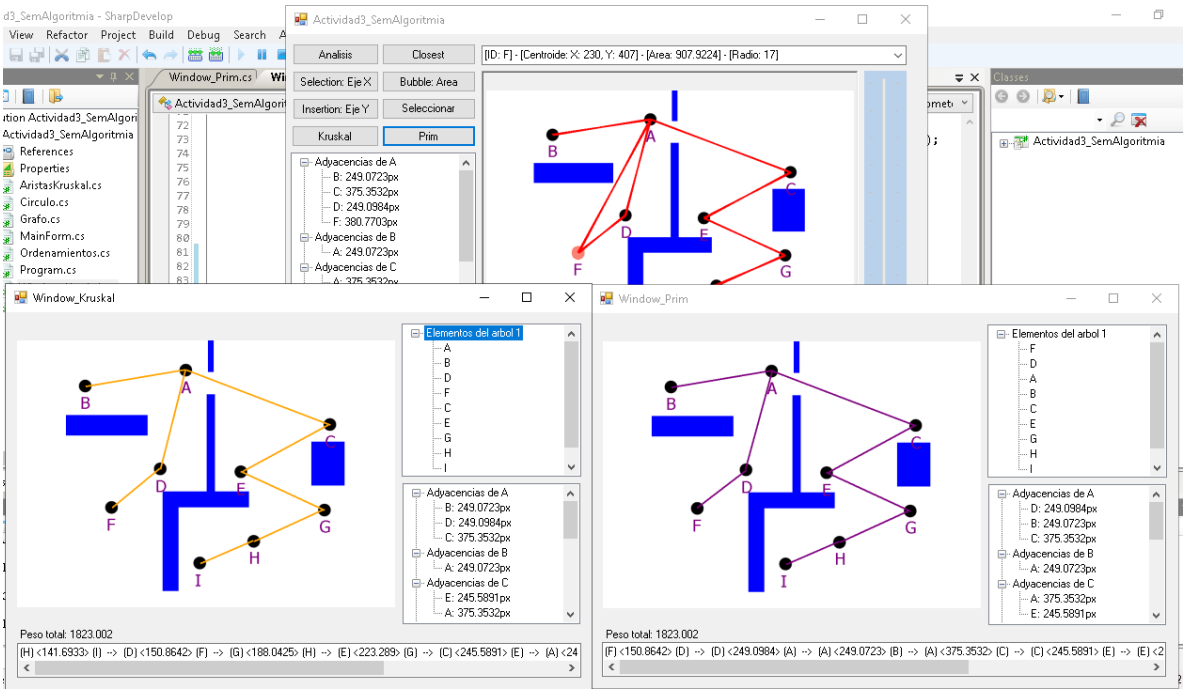
Ejecución del algoritmo de Kruskal

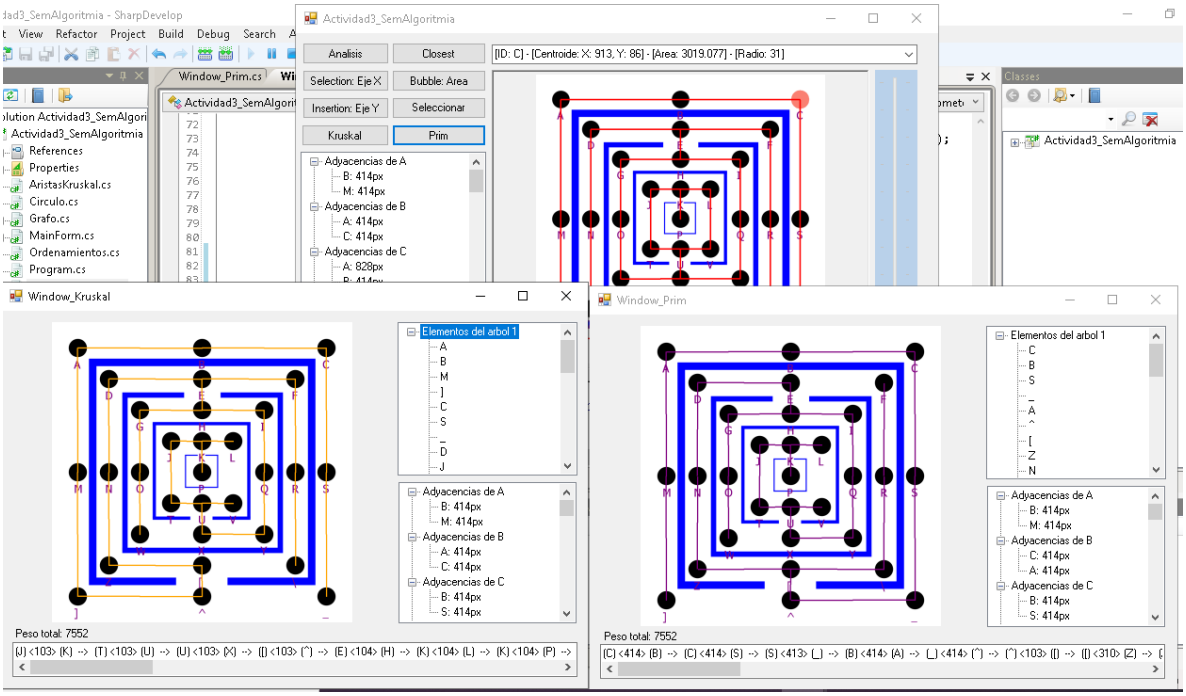


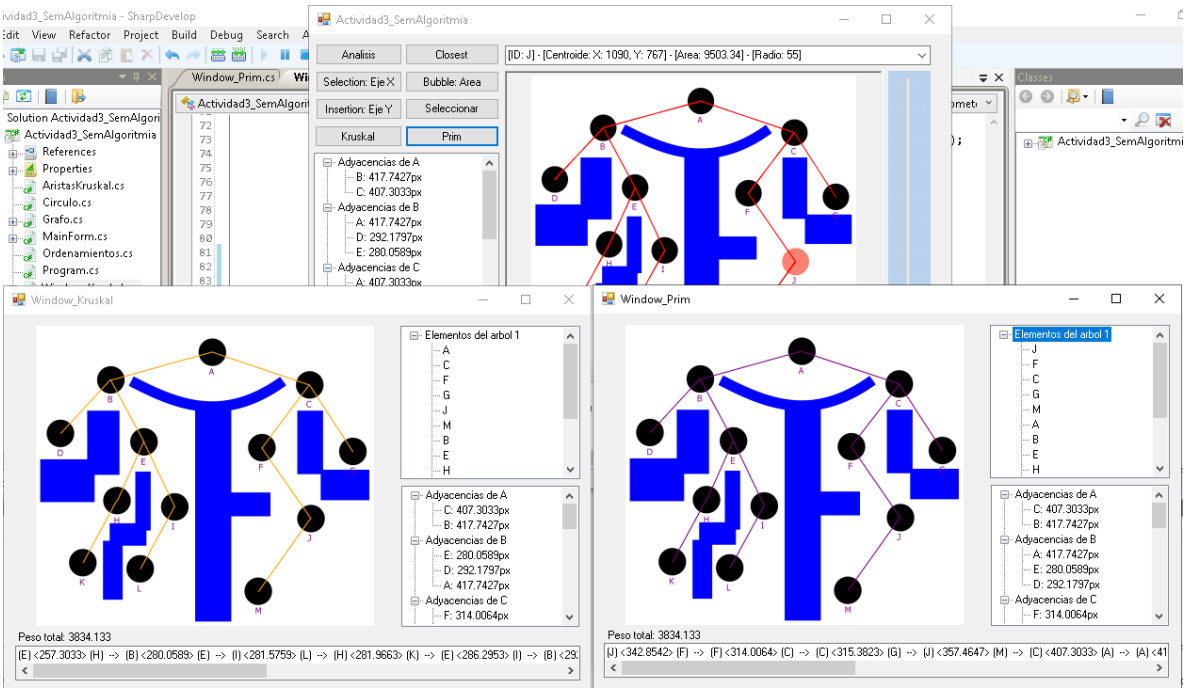
Selección del vértice J y ejecución del algoritmo de Prim

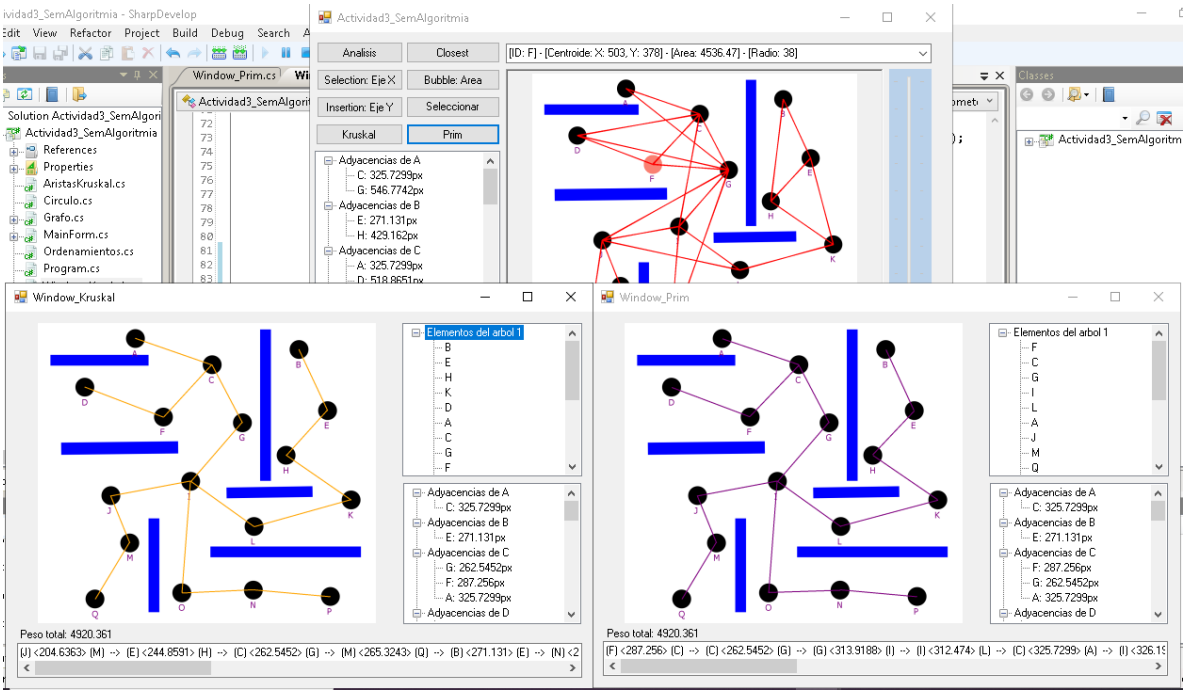


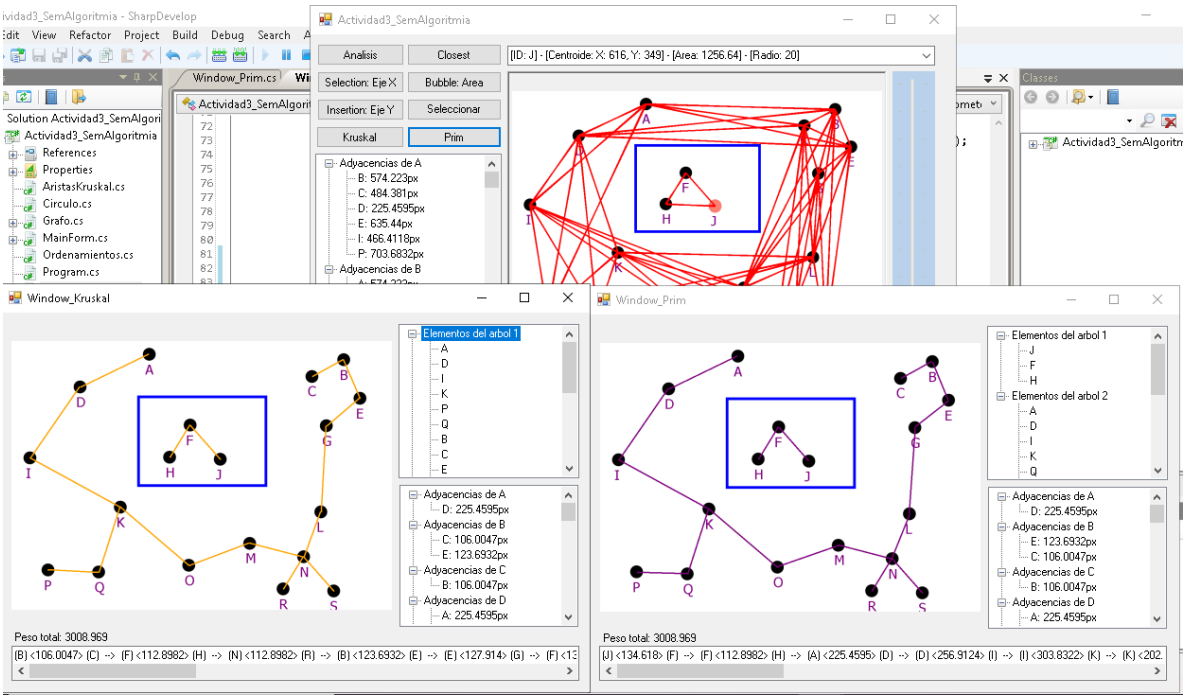
Pruebas con las demás imágenes

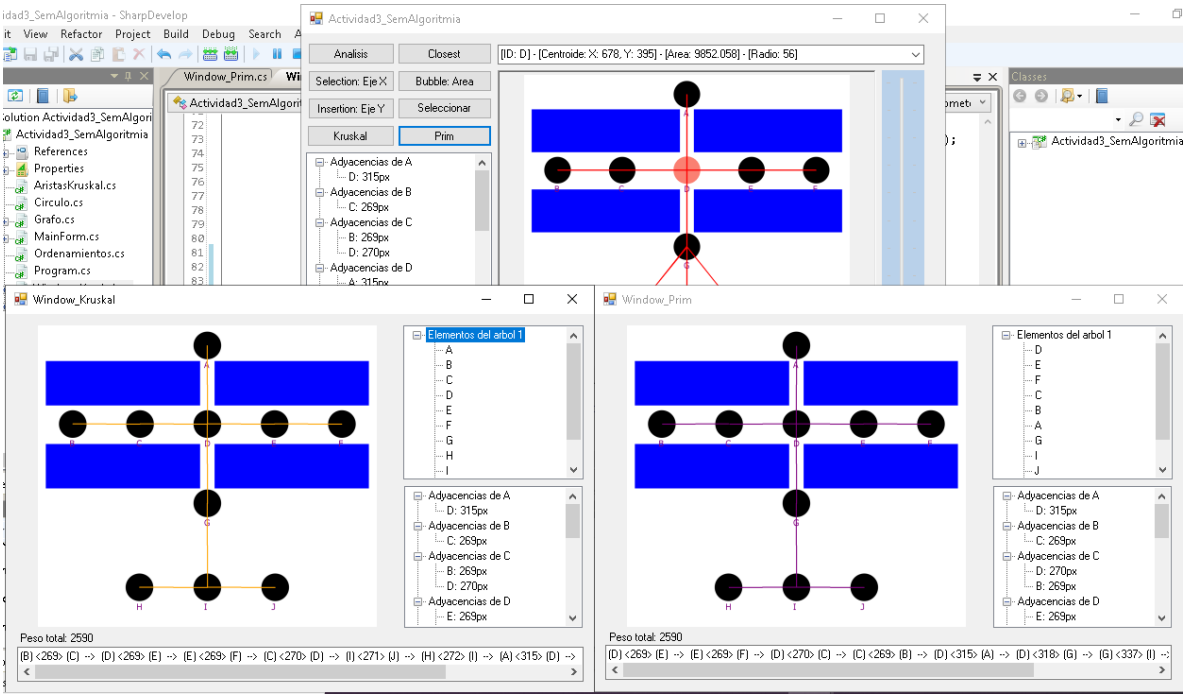





Actividad3_SemAlgoritmia
Analisis
Closest
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Window_Kruskal
Window_Prim
Peso total: 3834.133
Peso total: 3834.133


Actividad3_SemAlgoritmia
Analisis
Closest
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
Window_Kruskal
Window_Prim
Peso total: 4920.361
Peso total: 4920.361


Actividad3_SemAlgoritmia
Analisis
Closest
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
[ID: J] - [Centroide: X: 616, Y: 349] - [Area: 1256.64] - [Radio: 20]
Window_Kruskal
Window_Prim
Elementos del arbol 1
Elementos del arbol 2
Peso total: 3008.969


Actividad3_SemAlgoritmia
Analisis
Closest
Selection: Eje X
Bubble: Area
Insertion: Eje Y
Seleccionar
Kruskal
Prim
[ID: D] - [Centroide: X: 678, Y: 395] - [Area: 9852.058] - [Radio: 56]
Window_Kruskal
Window_Prim
Elementos del arbol 1
Peso total: 2590

Conclusiones.

En esta actividad creía que lo más difícil de realizar serían los algoritmos, pero al momento de empezar a hacerla me di cuenta de que no era así, lo más complicado para mi sin duda fue recorrer el grafo no conexo, pues me era difícil imaginarme como recorrer todos vértices si no tenían un elemento adyacente. Tarde mucho tiempo en solucionar esa cuestión, pero al momento de terminar kruskal ya la implementación de prim la realice más rápido pero realmente fue más difícil por el hecho de que no se tenían a todos los candidatos desde el principio. Respecto a la comparación de los dos algoritmos de forma gráfica decidí mostrarla en dos ventanas diferentes para así evitar saturar el Mainform de código que ya tiene bastante. El entendimiento de los algoritmos lo pude lograr perfectamente, entiendo cómo funciona cada uno.

Apéndice(s).

Código Agregado al MainForm

```
double pesoTotal = 0f;

    void Btn_KruskalClick(object sender, EventArgs e)
    {
      pesoTotal = 0f;
      subarbolesKuskal.Clear();
      wk = new Window_Kruskal();
      wk.Show();

      Ordenamientos ordenar =  new Ordenamientos();//para ordenar las aristas
      List<AristasKruskal> candidatos = grafo.getAristas();//todas las aristas del grafo
      List<AristasKruskal> prometedores = new List<AristasKruskal>();//aqui se ingresaran las aristas finales
seleccionadas
      List<List<Circulo>> cc = inicializaCC();//Conjunto de cojuntos de los circulos,
      //necesario para operar con Kruskal. Se inicializa con todos los vertices del grafo

      candidatos = elimininarRepetidos(candidatos);//Eliminino aristas repetidas, aristas no dirigidas. Solo
quedan dirigidas
      candidatos = ordenar.ordenarAristasK(candidatos);//Ordeno las aristas resultantes de menor a mayor

      //Inicia algoritmo de kruskal
      for(int i = 0; i < candidatos.Count; i++){//Recorro a los candidatos
        int indexOrigen = Buscar(candidatos[i].origen.getId(), cc);//Obtengo el indice del origen en el cc
        int indexDestino = Buscar(candidatos[i].destino.getId(), cc);//Obtengo el indice del destino en el cc

        if(indexOrigen != indexDestino){//Si los identificadores de los circulos estan en diferentes conjuntos
          prometedores.Add(candidatos[i]);//Se añade a prometedores la arista
          pesoTotal += candidatos[i].arista;
          //Se une el circulo destino al origen en el cc
          cc = fusionaCC(candidatos[i].origen, candidatos[i].destino, cc);
        }
```

```
        }

        //SUBARBOLES
        for(int o = 0; o < cc.Count; o++){
            if(cc[o].Count != 0){
                subarbolesKuskal.Add(cc[o]);
            }
        }


        //Creacion del grafo kruskal para visualizacion grafico
        wk.showGraph(c1, circulos, prometedores);
        wk.showInfo(subarbolesKuskal, pesoTotal);
    }

    //Busqueda de indices en el cc
    int Buscar(char comp, List<List<Circulo>> conjuntos){
        int i, j;
        for(i = 0; i < conjuntos.Count; i++){
            for(j = 0; j < conjuntos[i].Count; j++){
                if(conjuntos[i][j].getId() == comp){
                    return i;
                }
            }
        }
        return -1;
    }

    //Union de los circulos en el cc
    List<List<Circulo>> fusionaCC(Circulo c1, Circulo c2, List<List<Circulo>> cc){
        int o = Buscar(c1.getId(), cc);
        int d = Buscar(c2.getId(), cc);

        //Aqui se hace la union, se agregan todos los vertices del conjunto en destino en el conjunto en el
indice destino
        for(int i = 0; i < cc[d].Count; i++){
            cc[o].Add(cc[d][i]);
        }
        cc[d].Clear();
        return cc;
    }

    //Inicializacion del cc
    List<List<Circulo>> inicializaCC(){
        List<List<Circulo>> cc = new List<List<Circulo>>();
        for(int i = 0; i < circulos.Count; i++){
            List<Circulo> aux = new List<Circulo>();
            aux.Add(circulos[i]);
            cc.Add(aux);
        }
        return cc;
    }
```

```
    //Se eliminan aristas repetidas solo de los vertices que estan conectados de forma no dirigida
    //Por ejemplo: A->B hay 3.4, B->A hay 3.4, Se elimina B->A. Si fuera B->C y hay 3.4 entonces no se eliminaria
    List<AristasKruskal> elimininarRepetidos(List<AristasKruskal> prometedores){
      //Elimnar aristas repetidas
      for(int i = 0; i < prometedores.Count; i++){
        for(int j = 0; j < prometedores.Count; j++){
          if(i != j){
            if(prometedores[i].arista == prometedores[j].arista){
              if(prometedores[i].origen.getId() == prometedores[j].destino.getId()){
                if(prometedores[i].destino.getId() == prometedores[j].origen.getId()){
                  //Debug.WriteLine("*Comparado: " + prometedores[i].origen.getId()
                  //        + " - " + prometedores[i].destino.getId());
                  prometedores.RemoveAt(j);

                }
              }
            }
          }
        }
      }
      return prometedores;
    }


    void Btn_PrimClick(object sender, EventArgs e)
    {
      subarbolesPrim.Clear();
      wp = new Window_Prim();
      wp.Show();
      pesoTotal = 0f;

      List<AristasKruskal> prometedores = new List<AristasKruskal>();
      //Llamo a la funcion Prim, esto lo hago ya que estas es recursiva para que encuentre subarboles
      prometedores = Prim(prometedores, grafo.getVertice(circulos[actualCicleIndex]));
      //Rreutilizo la funcion para mostrar el grafo graficamente
      //crearGrafoKruskal_Prim(prometedores);

      wp.showGraph(c1, circulos, prometedores);
      wp.showInfo(subarbolesPrim, pesoTotal);
    }


    //Funcion para el algoritmo de prim, es recursiva para encontrar los subarboles
    List<AristasKruskal> Prim(List<AristasKruskal> prometedores, Vertice sel){

      Vertice seleccionado = sel;//Vertice inicial seleccionado
      List<AristasKruskal> candidatos = new List<AristasKruskal>();//Se guardan las aristas adyacentes de cada vertice agregado al conjunto
      List<Circulo> conjunto = new List<Circulo>();//Sive para comprobar los vertices que se han seleccionado
      conjunto.Add(seleccionado.circulo);//Agrego el primer vertice elegido
```

```
        //Obtengo a las aristas adyacentes del primer vertice seleccionado
        candidatos = getCandidatos(candidatos, seleccionado);

        //Los candidatos se van eliminado mientras se va ejecutando el algoritmo, cuando ya no halla terminara la ejecucucion
        while(candidatos.Count != 0){
          //Se obtiene la arista de menor peso adyacente a los candidatos disponibles
          AristasKruskal min = getMin(candidatos);

          //Si el vertice no esta añadido al conjunto entonces se procede a insertarlo en donde corresponde
          if(factible(conjunto, min)){
            pesoTotal += min.arista;
            prometedores.Add(min);//Se añade a prometedores el nuevo vertice que no esta en el conjunto
            //Si el origen de la arista esta en el conjunto, se añade el destino. Si no se hace lo contrario
            if(pertenece(min.origen, conjunto)){
              conjunto.Add(min.destino);//Se añade el nuevo vertice al conjunto
              //Se obtienen las aristas adyacentes del nuevo vertice, circulo y se añaden a candidatos
              candidatos = getCandidatos(candidatos, grafo.getVertice(min.destino));
            }else{
              conjunto.Add(min.origen);
              candidatos = getCandidatos(candidatos, grafo.getVertice(min.origen));
            }
          }
          //Se van eliminado las aristas que se van comprobado para que no se ejecute infinito
          eliminarCandidato(candidatos, min);
        }

        subarbolesPrim.Add(conjunto);

        //Esta funcion devuelve un circulo en caso de pertenecer a otro arbol no conexo
        Circulo c = revisarSubarboles(prometedores);

        //Si se obtuvo un circulo entonces significa que existe otro arbol
        if(c != null){
          //Debug.WriteLine("En subarbol: " + c.getId());
          //Se retorna lo que devuelve la funcion Prim pero ahora con los candidatos del otro subarbol
          return Prim(prometedores, grafo.getVertice(c));
        }else{
          //Si no hay otro arbol solo se retornan los candidatos del arbol unico
          return prometedores;
        }

      }

      //Esta funcion me ayuda a comprobar si existe algun otro subarbol
      Circulo revisarSubarboles(List<AristasKruskal> prometedores){
        //Obtengo todas las aristas existentes en el grafo y sus adyacencias
        List<AristasKruskal> aristasSubarboles = grafo.getAristas();
        bool band = false;

        //Recorro todas las aristas del grafo y comparo cada una de ellas con los elementos dentro de prometedores
```

```
    for(int i = 0; i < aristasSubarboles.Count; i++){
        for(int j = 0; j < prometedores.Count; j++){
            //Si se encuentran coincidencias significa que el vertice comprobado no esta en otro arbol
            if(aristasSubarboles[i].origen.getId() == prometedores[j].origen.getId()){
                band = true;
                break;
            }
            if(aristasSubarboles[i].origen.getId() == prometedores[j].destino.getId()){
                band = true;
                break;
            }
        }
        //Si nunca se encontraron coincidencias, es decir que la bandera nunca cambio de estado,
        //... se retorna el circulo que no deberia pertenecer a prometedores.
        if(band == false){
            //Debug.WriteLine("Retornado: " + aristasSubarboles[i].origen.getId());
            return aristasSubarboles[i].origen;
        }
        //Reset de la bandera
        band = false;
    }
    //Si siempre de encontraron coincidencias entonces no hay subarboles
    return null;
}

//Se comprueba si el vertice/circulo seleccionado esta ya agregado a conjuntos
bool factible(List<Circulo> conjunto, AristasKruskal min){
    for(int i = 0; i < conjunto.Count; i++){
        if(min.destino.getId() == conjunto[i].getId()){
            return false;
        }
    }
    //Si no se encontraron coincidencias en el for devuelve true para ingresar al seleccionado
    return true;
}

//Se comprueba si un circulo pertenece al conjunto de circulos
bool pertenece(Circulo c, List<Circulo> conjunto){
    for(int i = 0; i < conjunto.Count; i++){
        if(c.getId() == conjunto[i].getId()){
            return true;
        }
    }
    return false;
}

//Eliminacion de candidatos
List<AristasKruskal> eliminarCandidato(List<AristasKruskal> candidatos, AristasKruskal eliminado){
    for(int i = 0; i < candidatos.Count; i++){
        if(candidatos[i] == eliminado){
            candidatos.RemoveAt(i);
        }
    }
```

```
        return candidatos;
    }

    //Con esta funcion obtengo los cadidatos del vertice que se recibe, candidatos son las aristas
adyacentes
    List<AristasKruskal> getCandidatos(List<AristasKruskal> candidatos, Vertice seleccionado){
        Arista auxArista = seleccionado.ady;
        while(auxArista != null){
            AristasKruskal a
= new AristasKruskal(seleccionado.circulo, auxArista.ady.circulo, auxArista.distance);
            candidatos.Add(a);
            auxArista = auxArista.sig;
        }
        return candidatos;
    }



    AristasKruskal getMin(List<AristasKruskal> candidatos){
        //Obtengo el valor menor de las aristas
        AristasKruskal min = candidatos[0];
        for(int i = 1; i < candidatos.Count; i++){
            if(min.arista > candidatos[i].arista){
                min = candidatos[i];
            }
        }
        return min;
    }
```

Clase AristasKruskal

```
using System;

namespace Actividad3_SemAlgoritmia
{

  public class AristasKruskal
  {

    public Circulo origen;
    public Circulo destino;
    public double arista;

    public AristasKruskal(Circulo or, Circulo des, double ar)
    {
        origen = or;
        destino = des;
        arista = ar;
    }
  }
}
```

Ventanas para mostrar Prim.

```
/*
 * Created by SharpDevelop.
 * User: AnthonySandoval
 * Date: 30/05/2021
 * Time: 11:45 a. m.
 *
 * To change this template use Tools | Options | Coding | Edit Standard Headers.
 */
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Drawing.Drawing2D;

namespace Actividad3_SemAlgoritmia
{

   public partial class Window_Prim : Form
   {
      Bitmap bmp_Kruskal_Prim;
      List<Circulo> circulos;
      Grafo grafoKruskal = new Grafo();

      public Window_Prim()
      {
         InitializeComponent();
         grafoKruskal.inicializar();
         textBoxK.Multiline = true;
         textBoxK.WordWrap = false;
         textBoxK.ScrollBars = System.Windows.Forms.ScrollBars.Horizontal;
      }

      public void showInfo(List<List<Circulo>> subarboles, double pesoT){
         grafoKruskal.listaAdyacenciaTreeView(treeViewAdy);

         treeViewGrafo.BeginUpdate();
         for(int i = 0; i < subarboles.Count; i++){
            treeViewGrafo.Nodes.Add("Elementos del arbol " + (i+1));
            for(int j = 0; j < subarboles[i].Count; j++){
               treeViewGrafo.Nodes[i].Nodes.Add(Char.ToString(subarboles[i][j].getId()));
            }
            treeViewGrafo.Nodes[i].Expand();
         }
         treeViewGrafo.EndUpdate();
         peso.Text = "Peso total: " + (float)pesoT;
      }

      public void showGraph(string c1, List<Circulo> circulos, List<AristasKruskal> prometedores){

         this.circulos = circulos;
```

```
        //Para mostrar el grafo
        bmp_Kruskal_Prim = new Bitmap(c1);
        pBox_Prim.Image = bmp_Kruskal_Prim;
        etiquetasCirculos(35, 50, bmp_Kruskal_Prim, 30);

        //Cargo todos los circulos que hay
        for(int i = 0; i < circulos.Count; i++){
           grafoKruskal.insertarVertice(circulos[i]);
        }
        //Cargo las conexiones, aristas respecto a krusgal
        for(int j = 0; j < prometedores.Count; j++){

           //Ingreso dos veces la aristas en sentidos diferentes para que siga siendo un grafo no dirigido
           grafoKruskal.insertarArista(grafoKruskal.getVertice(prometedores[j].origen),
                              grafoKruskal.getVertice(prometedores[j].destino),
                              calculeDistance(prometedores[j].origen, prometedores[j].destino));

           grafoKruskal.insertarArista(grafoKruskal.getVertice(prometedores[j].destino),
                              grafoKruskal.getVertice(prometedores[j].origen),
                              calculeDistance(prometedores[j].destino, prometedores[j].origen));
           //Coloreado de las lineas
           drawLine(bmp_Kruskal_Prim, prometedores[j].origen.getCentroX(),
                         prometedores[j].origen.getCentroY(),
                         prometedores[j].destino.getCentroX(),
                         prometedores[j].destino.getCentroY()
                        );
        }

        for(int i = 0; i < prometedores.Count; i++){
           textBoxK.Text += "(" + prometedores[i].origen.getId() + ") <" +
                      (float)prometedores[i].arista + "> ("
                      + prometedores[i].destino.getId() + ")  -->  ";
        }
     }

     void etiquetasCirculos(int width, int heigth, Bitmap bmp, int tamLetra){
        for(int i = 0; i < circulos.Count; i++){
           ponerEtiqueta(circulos[i].getCentroX()-
18, (circulos[i].getCentroY() +  Convert.ToInt32(circulos[i].getRadio())), width, heigth, bmp, circulos[i].getId()
, tamLetra);
        }
     }


     void ponerEtiqueta(int x, int y, int width, int height, Bitmap bmp, char id, int tamLetra){
        RectangleF rectf = new RectangleF(x, y, width, height);

        Graphics g = Graphics.FromImage(bmp);

        g.SmoothingMode = SmoothingMode.AntiAlias;
        g.InterpolationMode = InterpolationMode.HighQualityBicubic;
        g.PixelOffsetMode = PixelOffsetMode.HighQuality;
        g.DrawString(Char.ToString(id), new Font("Tahoma", tamLetra), Brushes.Purple, rectf);
```

```
            g.Flush();
        }

        //Se dibuja una linea entre dos puntos
        void drawLine(Bitmap bmp, int x1, int y1, int x2, int y2){
            Graphics g = Graphics.FromImage(bmp);
            Pen p = new Pen(Color.Purple, 4);

            g.DrawLine(p, x1-1, y1-1, x2+1, y2+1);
        }


        double calculeDistance(Circulo origen, Circulo destino){
            return Math.Sqrt(Math.Pow(origen.getCentroX() - destino.getCentroX(), 2)
                             + Math.Pow(origen.getCentroY() - destino.getCentroY(), 2));
        }

    }
}
```

Ventanas para mostrar Kruskal.

```
/*
 * Created by SharpDevelop.
 * User: AnthonySandoval
 * Date: 30/05/2021
 * Time: 11:44 a. m.
 *
 * To change this template use Tools | Options | Coding | Edit Standard Headers.
 */
using System;
using System.Diagnostics;
using System.Drawing;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Drawing.Drawing2D;

namespace Actividad3_SemAlgoritmia
{
    public partial class Window_Kruskal : Form
    {
        Bitmap bmp_Kruskal_Prim;
        List<Circulo> circulos;
        Grafo grafoKruskal = new Grafo();

        public Window_Kruskal()
        {

            InitializeComponent();
            grafoKruskal.inicializar();
            textBoxK.Multiline = true;
            textBoxK.WordWrap = false;
            textBoxK.ScrollBars = System.Windows.Forms.ScrollBars.Horizontal;
```

```
}

public void showInfo(List<List<Circulo>> subarboles, double pesoT){
   grafoKruskal.listaAdyacenciaTreeView(treeViewAdy);

   treeViewGrafo.BeginUpdate();
   for(int i = 0; i < subarboles.Count; i++){
      treeViewGrafo.Nodes.Add("Elementos del arbol " + (i+1));
      for(int j = 0; j < subarboles[i].Count; j++){
         treeViewGrafo.Nodes[i].Nodes.Add(Char.ToString(subarboles[i][j].getId()));
      }
      treeViewGrafo.Nodes[i].Expand();
   }
   treeViewGrafo.EndUpdate();
   peso.Text = "Peso total: " + (float)pesoT;
}

public void showGraph(string c1, List<Circulo> circulos, List<AristasKruskal> prometedores){

   this.circulos = circulos;

   //Para mostrar el grafo
   bmp_Kruskal_Prim = new Bitmap(c1);
   pBox_Kruskal.Image = bmp_Kruskal_Prim;
   etiquetasCirculos(35, 50, bmp_Kruskal_Prim, 30);

   //Cargo todos los circulos que hay
   for(int i = 0; i < circulos.Count; i++){
      grafoKruskal.insertarVertice(circulos[i]);
   }
   //Cargo las conexiones, aristas respecto a krusgal
   for(int j = 0; j < prometedores.Count; j++){

      //Ingreso dos veces la aristas en sentidos diferentes para que siga siendo un grafo no dirigido
      grafoKruskal.insertarArista(grafoKruskal.getVertice(prometedores[j].origen),
                                  grafoKruskal.getVertice(prometedores[j].destino),
                                  calculeDistance(prometedores[j].origen, prometedores[j].destino));

      grafoKruskal.insertarArista(grafoKruskal.getVertice(prometedores[j].destino),
                                  grafoKruskal.getVertice(prometedores[j].origen),
                                  calculeDistance(prometedores[j].destino, prometedores[j].origen));
      //Coloreado de las lineas
      drawLine(bmp_Kruskal_Prim, prometedores[j].origen.getCentroX(),
                        prometedores[j].origen.getCentroY(),
                        prometedores[j].destino.getCentroX(),
                        prometedores[j].destino.getCentroY()
                       );
   }

   for(int i = 0; i < prometedores.Count; i++){
      textBoxK.Text += "(" + prometedores[i].origen.getId() + ") <" +
                  (float)prometedores[i].arista + "> ("
```

```
                    + prometedores[i].destino.getId() + ")  -->  ";
        }

    }

    void etiquetasCirculos(int width, int heigth, Bitmap bmp, int tamLetra){
        for(int i = 0; i < circulos.Count; i++){
            ponerEtiqueta(circulos[i].getCentroX()-
18, (circulos[i].getCentroY() +  Convert.ToInt32(circulos[i].getRadio())), width, heigth, bmp, circulos[i].getId()
, tamLetra);
        }
    }


    void ponerEtiqueta(int x, int y, int width, int height, Bitmap bmp, char id, int tamLetra){
        RectangleF rectf = new RectangleF(x, y, width, height);

        Graphics g = Graphics.FromImage(bmp);

        g.SmoothingMode = SmoothingMode.AntiAlias;
        g.InterpolationMode = InterpolationMode.HighQualityBicubic;
        g.PixelOffsetMode = PixelOffsetMode.HighQuality;
        g.DrawString(Char.ToString(id), new Font("Tahoma", tamLetra), Brushes.Purple, rectf);

        g.Flush();
    }

    //Se dibuja una linea entre dos puntos
    void drawLine(Bitmap bmp, int x1, int y1, int x2, int y2){
        Graphics g = Graphics.FromImage(bmp);
        Pen p = new Pen(Color.Orange, 4);

        g.DrawLine(p, x1-1, y1-1, x2+1, y2+1);
    }


    double calculeDistance(Circulo origen, Circulo destino){
        return Math.Sqrt(Math.Pow(origen.getCentroX() - destino.getCentroX(), 2)
                + Math.Pow(origen.getCentroY() - destino.getCentroY(), 2));
    }
  }
}
```