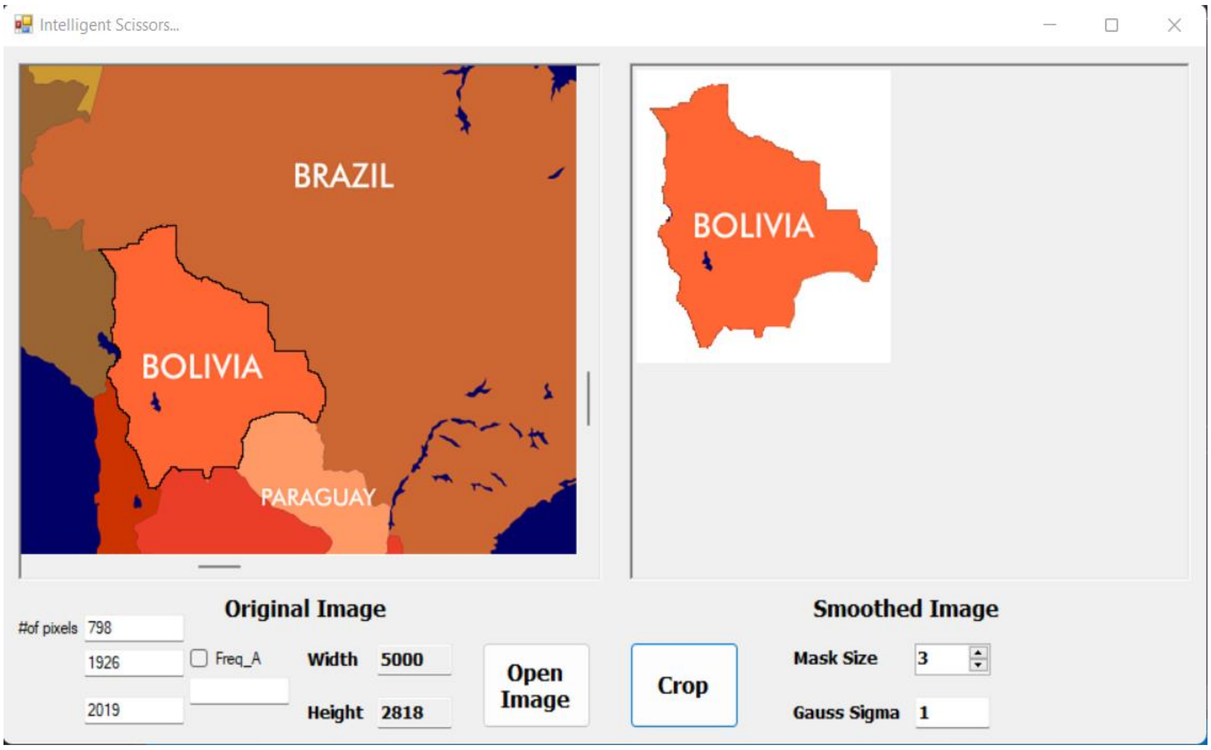
Dr: Mohamed Raafat Salem

Project Intelligent Scissors

Team Members:

1- (20191700070) احمد محمد وفيق خليل
2- (20191700039) احمد صلاح محمد مصطفى
3- (20191700917) نورا احمد حافظ احمد

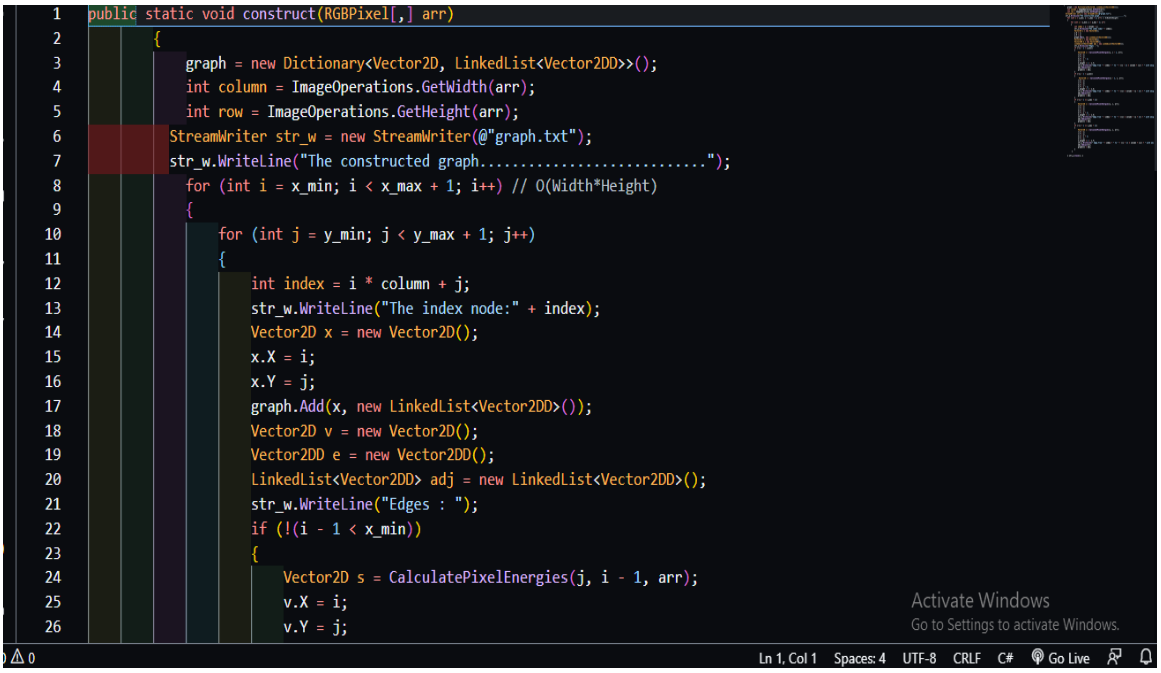
1) Crop Feature



#of pixels 798
1926
2019
Width 5000
Height 2818
Mask Size 3
Gauss Sigma 1

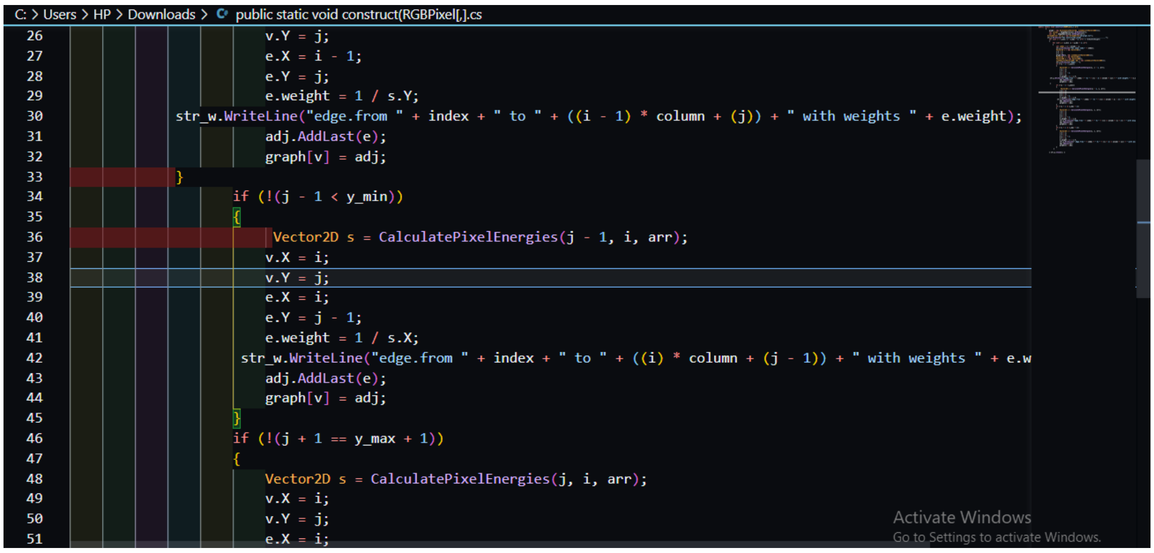
1) Function Construct

construct the graph using the 4-conectivity, and the total complexity is O (Height*Width)=O(V).each vertex represent the pixel.



```
public static void construct(RGBPixel[,] arr)
        {
            graph = new Dictionary<Vector2D, LinkedList<Vector2DD>>();
            int column = ImageOperations.GetWidth(arr);
            int row = ImageOperations.GetHeight(arr);
          StreamWriter str_w = new StreamWriter(@"graph.txt");
          str_w.WriteLine("The constructed graph...........................");
            for (int i = x_min; i < x_max + 1; i++) // O(Width*Height)
            {
                for (int j = y_min; j < y_max + 1; j++)
                {
                    int index = i * column + j;
                    str_w.WriteLine("The index node:" + index);
                    Vector2D x = new Vector2D();
                    x.X = i;
                    x.Y = j;
                    graph.Add(x, new LinkedList<Vector2DD>());
                    Vector2D v = new Vector2D();
                    Vector2DD e = new Vector2DD();
                    LinkedList<Vector2DD> adj = new LinkedList<Vector2DD>();
                    str_w.WriteLine("Edges : ");
                    if (!(i - 1 < x_min))
                    {
                        Vector2D s = CalculatePixelEnergies(j, i - 1, arr);
                        v.X = i;
                        v.Y = j;
```

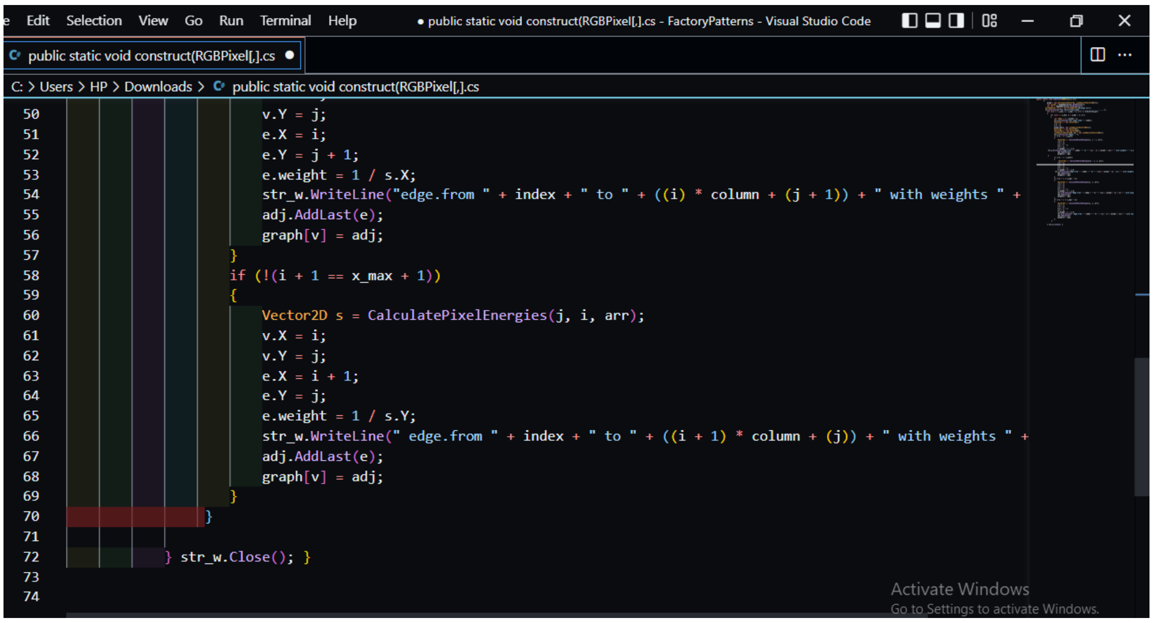
Activate Windows
Go to Settings to activate Windows.
⚠ 0
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
C#
Go Live



C: > Users > HP > Downloads > public static void construct(RGBPixel[,].cs

```
                        v.Y = j;
                        e.X = i - 1;
                        e.Y = j;
                        e.weight = 1 / s.Y;
             str_w.WriteLine("edge.from " + index + " to " + ((i - 1) * column + (j)) + " with weights " + e.weight);
                        adj.AddLast(e);
                        graph[v] = adj;
             }
                    if (!(j - 1 < y_min))
                    {
                         Vector2D s = CalculatePixelEnergies(j - 1, i, arr);
                        v.X = i;
                        v.Y = j;
                        e.X = i;
                        e.Y = j - 1;
                        e.weight = 1 / s.X;
                     str_w.WriteLine("edge.from " + index + " to " + ((i) * column + (j - 1)) + " with weights " + e.w
                        adj.AddLast(e);
                        graph[v] = adj;
                    }
                    if (!(j + 1 == y_max + 1))
                    {
                        Vector2D s = CalculatePixelEnergies(j, i, arr);
                        v.X = i;
                        v.Y = j;
                        e.X = i;
```

Activate Windows
Go to Settings to activate Windows.


e Edit Selection View Go Run Terminal Help • public static void construct(RGBPixel[,].cs - FactoryPatterns - Visual Studio Code

public static void construct(RGBPixel[,].cs

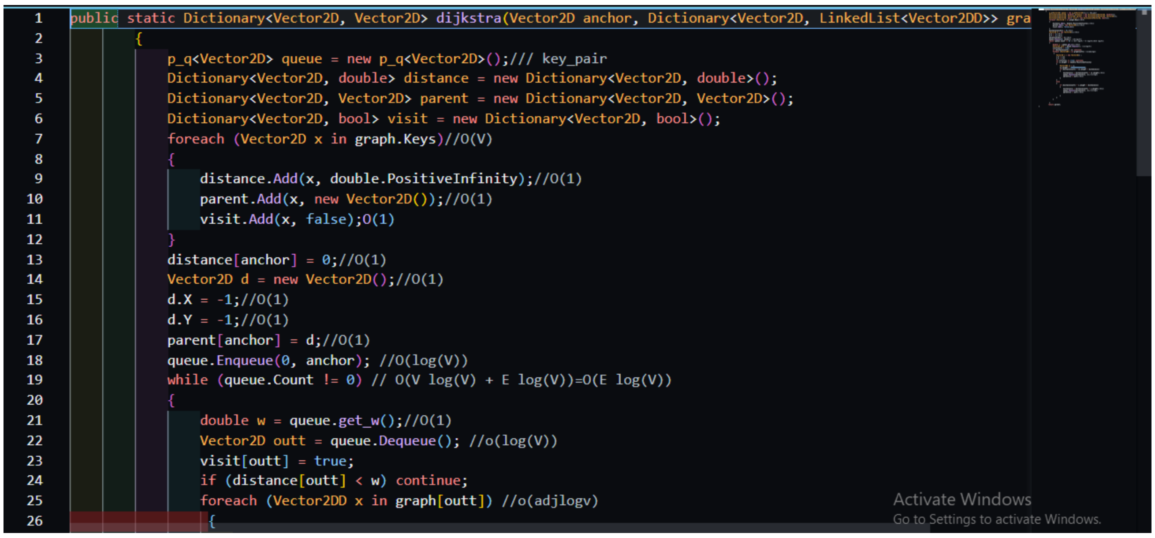
C: > Users > HP > Downloads > public static void construct(RGBPixel[,].cs

```
                v.Y = j;
                e.X = i;
                e.Y = j + 1;
                e.weight = 1 / s.X;
                str_w.WriteLine("edge.from " + index + " to " + ((i) * column + (j + 1)) + " with weights " +
                adj.AddLast(e);
                graph[v] = adj;
            }
            if (!(i + 1 == x_max + 1))
            {
                Vector2D s = CalculatePixelEnergies(j, i, arr);
                v.X = i;
                v.Y = j;
                e.X = i + 1;
                e.Y = j;
                e.weight = 1 / s.Y;
                str_w.WriteLine(" edge.from " + index + " to " + ((i + 1) * column + (j)) + " with weights " +
                adj.AddLast(e);
                graph[v] = adj;
            }
        }

    } str_w.Close(); }
```

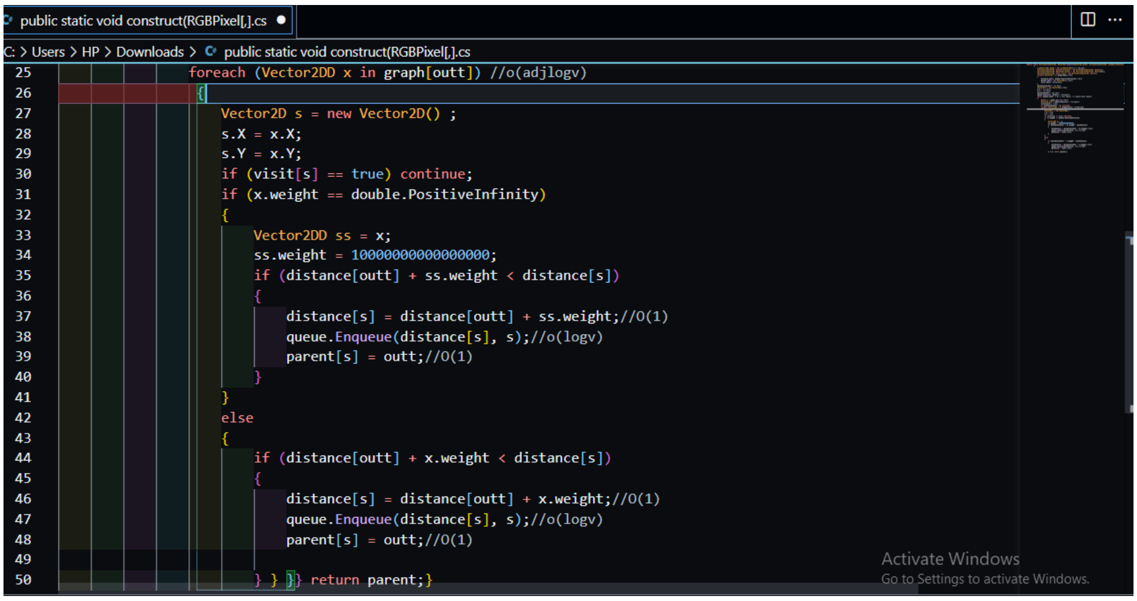
2) Function Dijikstra_Algorithm

find the shortest path between any two pixels in the image. Save the parent of each pixel(vertex) and the parent of the source(anchor) is null.

We are using the Priority_Queue to enhance the total complexity to O (E Log (V)) ,The total complexity is O(v+log(v)+ E Log (V))= O (E Log (V)).



```
public static Dictionary<Vector2D, Vector2D> dijkstra(Vector2D anchor, Dictionary<Vector2D, LinkedList<Vector2DD>> gra
    {
        p_q<Vector2D> queue = new p_q<Vector2D>();/// key_pair
        Dictionary<Vector2D, double> distance = new Dictionary<Vector2D, double>();
        Dictionary<Vector2D, Vector2D> parent = new Dictionary<Vector2D, Vector2D>();
        Dictionary<Vector2D, bool> visit = new Dictionary<Vector2D, bool>();
        foreach (Vector2D x in graph.Keys)//O(V)
        {
            distance.Add(x, double.PositiveInfinity);//O(1)
            parent.Add(x, new Vector2D());//O(1)
            visit.Add(x, false);O(1)
        }
        distance[anchor] = 0;//O(1)
        Vector2D d = new Vector2D();//O(1)
        d.X = -1;//O(1)
        d.Y = -1;//O(1)
        parent[anchor] = d;//O(1)
        queue.Enqueue(0, anchor); //O(log(V))
        while (queue.Count != 0) // O(V log(V) + E log(V))=O(E log(V))
        {
            double w = queue.get_w();//O(1)
            Vector2D outt = queue.Dequeue(); //o(log(V))
            visit[outt] = true;
            if (distance[outt] < w) continue;
            foreach (Vector2DD x in graph[outt]) //o(adjlogv)
            {
```

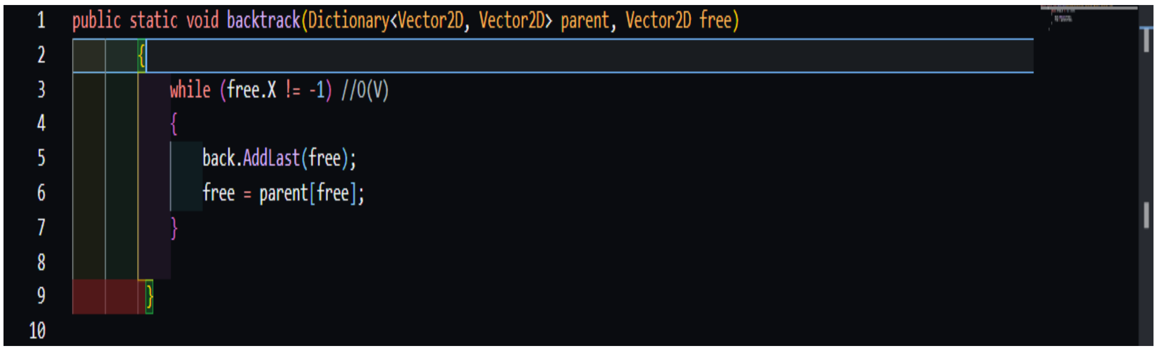

public static void construct(RGBPixel[,].cs

C: > Users > HP > Downloads > public static void construct(RGBPixel[,].cs

```
foreach (Vector2DD x in graph[outt]) //o(adjlogv)
{
    Vector2D s = new Vector2D() ;
    s.X = x.X;
    s.Y = x.Y;
    if (visit[s] == true) continue;
    if (x.weight == double.PositiveInfinity)
    {
        Vector2DD ss = x;
        ss.weight = 100000000000000000;
        if (distance[outt] + ss.weight < distance[s])
        {
            distance[s] = distance[outt] + ss.weight;//O(1)
            queue.Enqueue(distance[s], s);//o(logv)
            parent[s] = outt;//O(1)
        }
    }
    else
    {
        if (distance[outt] + x.weight < distance[s])
        {
            distance[s] = distance[outt] + x.weight;//O(1)
            queue.Enqueue(distance[s], s);//o(logv)
            parent[s] = outt;//O(1)

        } } }} return parent;}
```

3) Function backtrack

 Is used to get the path between the two points using the parent_Dictionary .The total complexity is O(V) where V is the number of vertex.

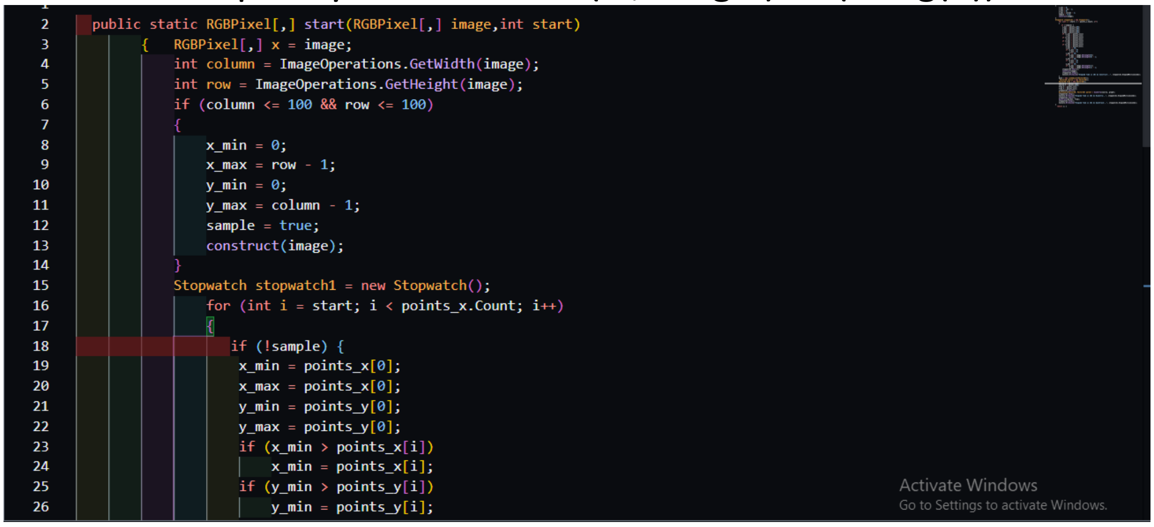


```
public static void backtrack(Dictionary<Vector2D, Vector2D> parent, Vector2D free)
{
    while (free.X != -1) //O(V)
    {
        back.AddLast(free);
        free = parent[free];
    }

}
```

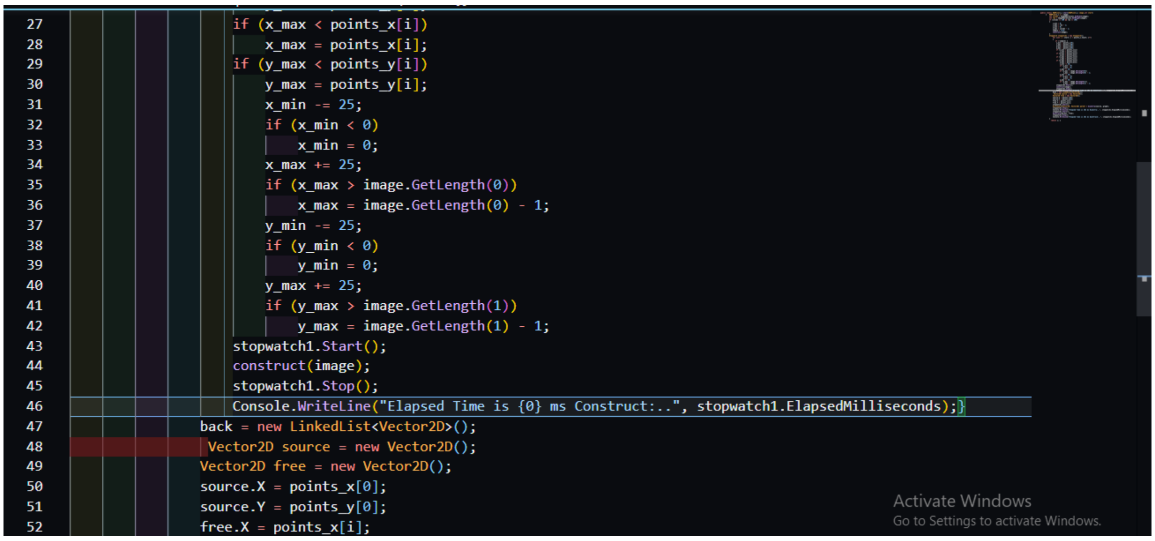
4) Function Start

manage the process of calling the other functions Starting from calling the construct function(if the sample test case we construct the full image one time, but in the complete test case we construct a needed part of the image), then calling the Dijikstra-Function to find the shortest path, then calling the backtrack-function to find the path, Noted that the for loop is iterate only one time when we call the start function to know which free point in the list we want to calculate the shortest path between and anchor and that point so, we take the time complexity of the body.

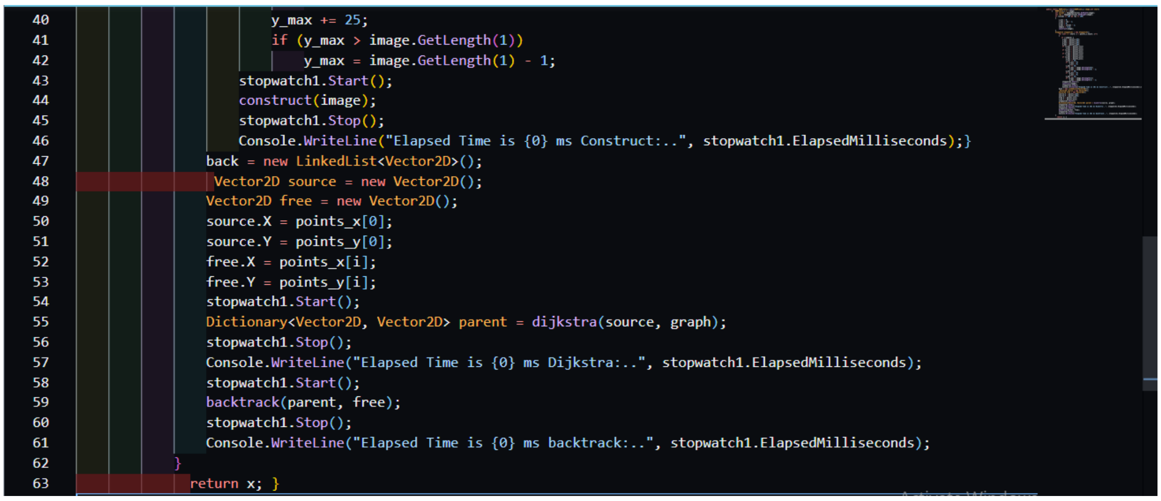
The total complexity Start = O Max(V ,E Log V)= O(E Log(V)).



```
public static RGBPixel[,] start(RGBPixel[,] image,int start)
    {   RGBPixel[,] x = image;
        int column = ImageOperations.GetWidth(image);
        int row = ImageOperations.GetHeight(image);
        if (column <= 100 && row <= 100)
        {
            x_min = 0;
            x_max = row - 1;
            y_min = 0;
            y_max = column - 1;
            sample = true;
            construct(image);
        }
        Stopwatch stopwatch1 = new Stopwatch();
            for (int i = start; i < points_x.Count; i++)
            {
               if (!sample) {
                x_min = points_x[0];
                x_max = points_x[0];
                y_min = points_y[0];
                y_max = points_y[0];
                if (x_min > points_x[i])
                    x_min = points_x[i];
                if (y_min > points_y[i])
                    y_min = points_y[i];
```



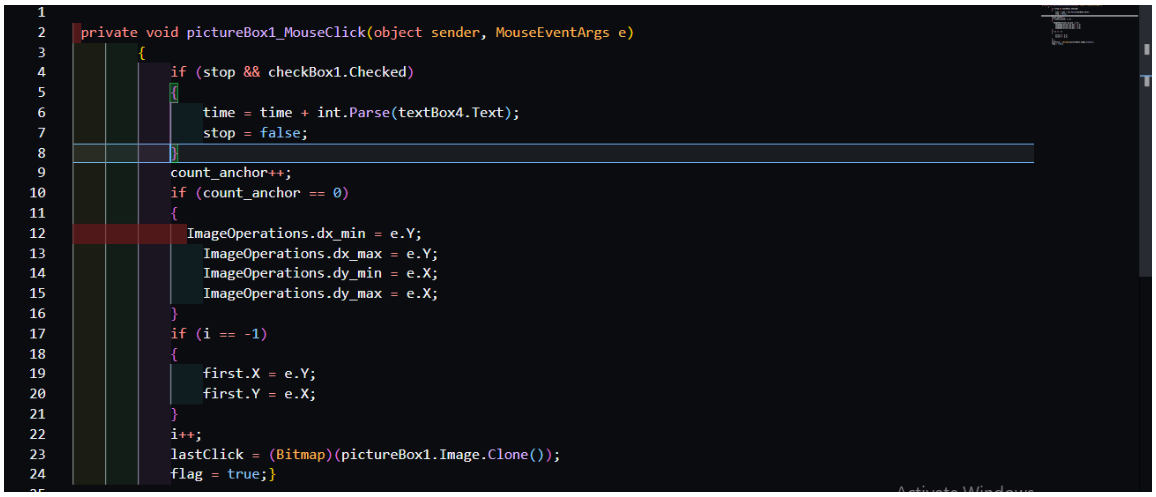
```
                if (x_max < points_x[i])
                    x_max = points_x[i];
                if (y_max < points_y[i])
                    y_max = points_y[i];
                    x_min -= 25;
                    if (x_min < 0)
                        x_min = 0;
                    x_max += 25;
                    if (x_max > image.GetLength(0))
                        x_max = image.GetLength(0) - 1;
                    y_min -= 25;
                    if (y_min < 0)
                        y_min = 0;
                    y_max += 25;
                    if (y_max > image.GetLength(1))
                        y_max = image.GetLength(1) - 1;
                stopwatch1.Start();
                construct(image);
                stopwatch1.Stop();
                Console.WriteLine("Elapsed Time is {0} ms Construct:..", stopwatch1.ElapsedMilliseconds);}
            back = new LinkedList<Vector2D>();
             Vector2D source = new Vector2D();
            Vector2D free = new Vector2D();
            source.X = points_x[0];
            source.Y = points_y[0];
            free.X = points_x[i];
```



```
                    y_max += 25;
                    if (y_max > image.GetLength(1))
                        y_max = image.GetLength(1) - 1;
                stopwatch1.Start();
                construct(image);
                stopwatch1.Stop();
                Console.WriteLine("Elapsed Time is {0} ms Construct:..", stopwatch1.ElapsedMilliseconds);}
            back = new LinkedList<Vector2D>();
             Vector2D source = new Vector2D();
            Vector2D free = new Vector2D();
            source.X = points_x[0];
            source.Y = points_y[0];
            free.X = points_x[i];
            free.Y = points_y[i];
            stopwatch1.Start();
            Dictionary<Vector2D, Vector2D> parent = dijkstra(source, graph);
            stopwatch1.Stop();
            Console.WriteLine("Elapsed Time is {0} ms Dijkstra:..", stopwatch1.ElapsedMilliseconds);
            stopwatch1.Start();
            backtrack(parent, free);
            stopwatch1.Stop();
            Console.WriteLine("Elapsed Time is {0} ms backtrack:..", stopwatch1.ElapsedMilliseconds);
        }
          return x; }
```

5) Function Mouse_Click

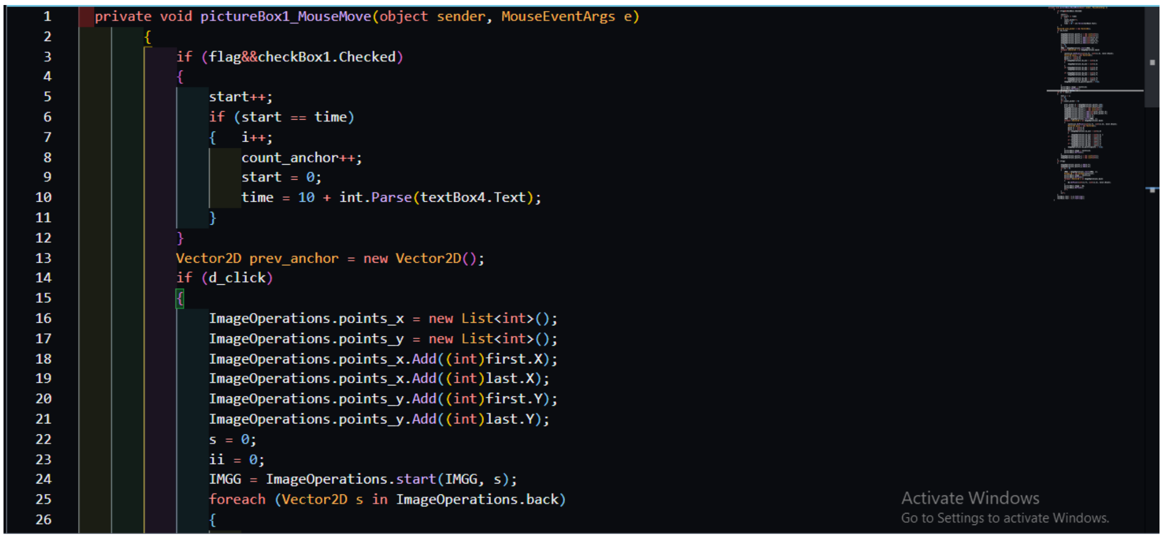
with every mouse click we apply a boolean variable to true to start the application logic and with every click we point an anchor point and count it and save the image in the (last_click) bitmap variable to use it when we draw the path to display it in the form.
With every click : all that statement take O(1)



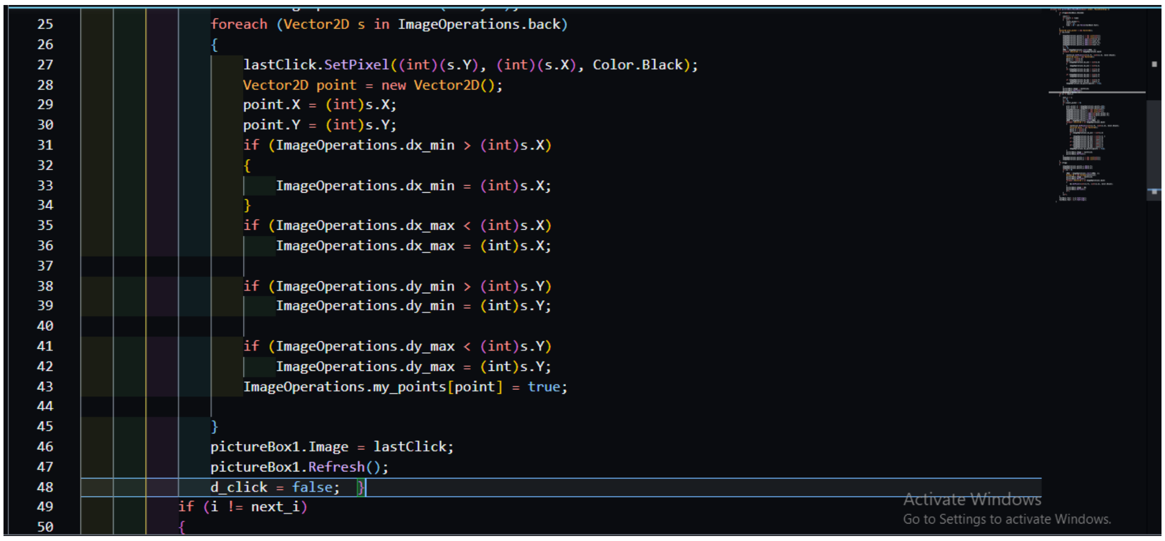
```
private void pictureBox1_MouseClick(object sender, MouseEventArgs e)
    {
        if (stop && checkBox1.Checked)
        {
            time = time + int.Parse(textBox4.Text);
            stop = false;
        }
        count_anchor++;
        if (count_anchor == 0)
        {
          ImageOperations.dx_min = e.Y;
            ImageOperations.dx_max = e.Y;
            ImageOperations.dy_min = e.X;
            ImageOperations.dy_max = e.X;
        }
        if (i == -1)
        {
            first.X = e.Y;
            first.Y = e.X;
        }
        i++;
        lastClick = (Bitmap)(pictureBox1.Image.Clone());
        flag = true;}
```

6) Function Mouse_Move

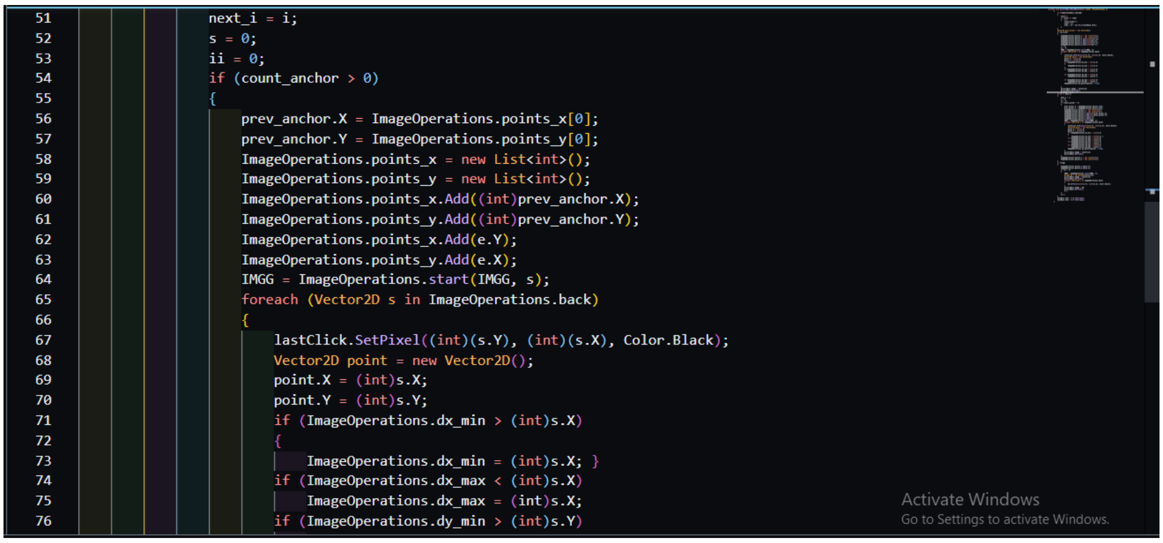
with every mouse move we calculate the shortest path between the anchor point that selected and the free points until click a new anchor point in this case we calculate the shortest path between the selected new anchor point and the previous anchor point and use the (lastclick) bitmap variable to modify the image to show the path in the form noted that when we don't use it when we calculate shortest path between anchor and free we use a copy of it to show the path and refresh the image with the original bitmap variable until we select a new anchor .
Note: when we double click we calculate the shortest path between first anchor and last anchor to close the shape.
Complexity: we draw the path in any condition in the mouse move which take O(V) as we loop through the path that calculated in the backtrack function and color it



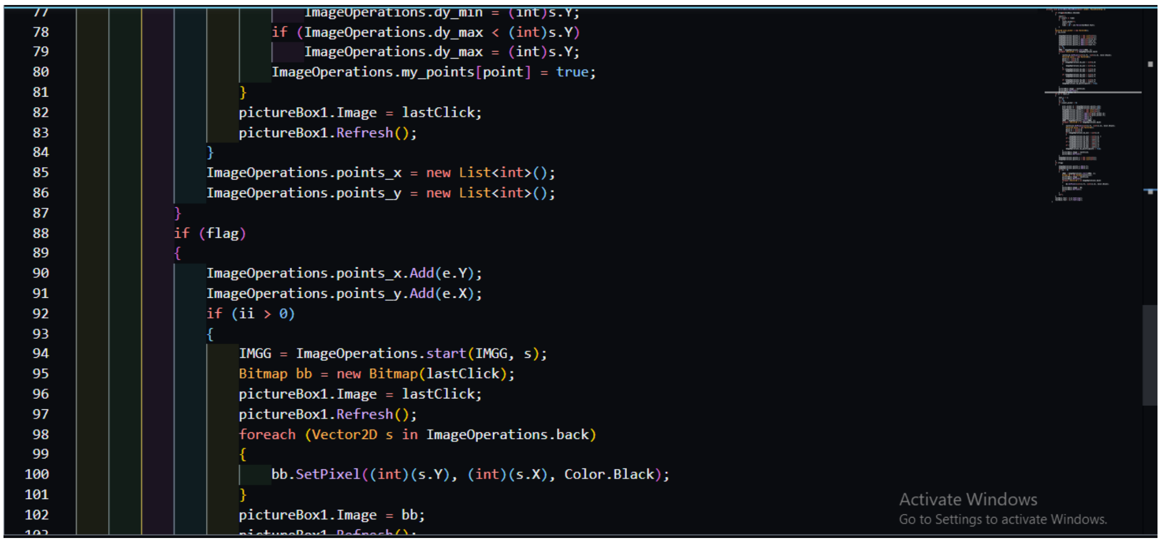
```
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
      {
          if (flag&&checkBox1.Checked)
          {
              start++;
              if (start == time)
              {   i++;
                  count_anchor++;
                  start = 0;
                  time = 10 + int.Parse(textBox4.Text);
              }
          }
          Vector2D prev_anchor = new Vector2D();
          if (d_click)
          {
              ImageOperations.points_x = new List<int>();
              ImageOperations.points_y = new List<int>();
              ImageOperations.points_x.Add((int)first.X);
              ImageOperations.points_x.Add((int)last.X);
              ImageOperations.points_y.Add((int)first.Y);
              ImageOperations.points_y.Add((int)last.Y);
              s = 0;
              ii = 0;
              IMGG = ImageOperations.start(IMGG, s);
              foreach (Vector2D s in ImageOperations.back)
              {
```



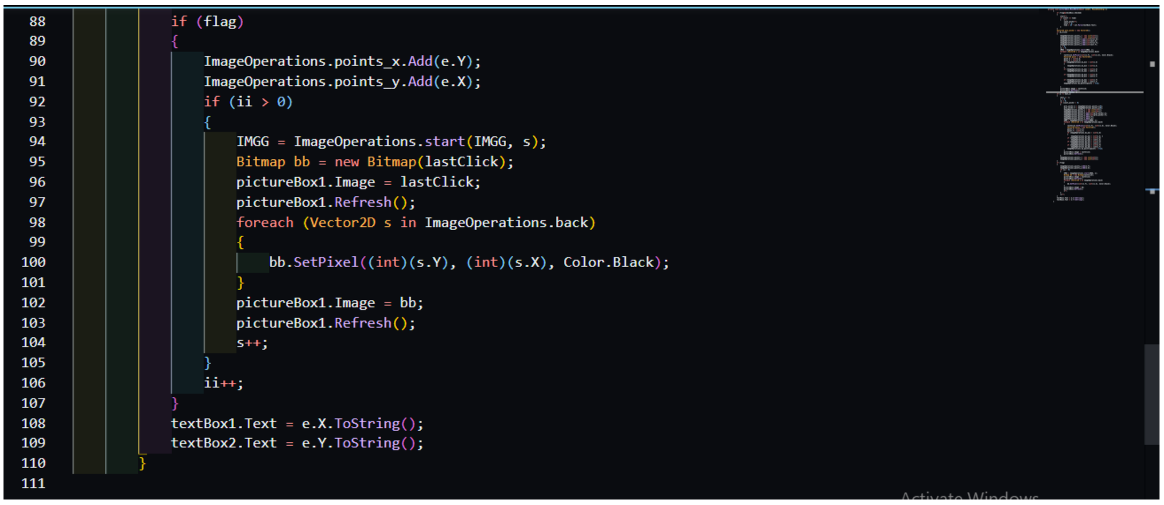
```
              foreach (Vector2D s in ImageOperations.back)
              {
                  lastClick.SetPixel((int)(s.Y), (int)(s.X), Color.Black);
                  Vector2D point = new Vector2D();
                  point.X = (int)s.X;
                  point.Y = (int)s.Y;
                  if (ImageOperations.dx_min > (int)s.X)
                  {
                      ImageOperations.dx_min = (int)s.X;
                  }
                  if (ImageOperations.dx_max < (int)s.X)
                      ImageOperations.dx_max = (int)s.X;

                  if (ImageOperations.dy_min > (int)s.Y)
                      ImageOperations.dy_min = (int)s.Y;

                  if (ImageOperations.dy_max < (int)s.Y)
                      ImageOperations.dy_max = (int)s.Y;
                  ImageOperations.my_points[point] = true;

              }
              pictureBox1.Image = lastClick;
              pictureBox1.Refresh();
              d_click = false;  }
          if (i != next_i)
          {
```



```
              next_i = i;
              s = 0;
              ii = 0;
              if (count_anchor > 0)
              {
                  prev_anchor.X = ImageOperations.points_x[0];
                  prev_anchor.Y = ImageOperations.points_y[0];
                  ImageOperations.points_x = new List<int>();
                  ImageOperations.points_y = new List<int>();
                  ImageOperations.points_x.Add((int)prev_anchor.X);
                  ImageOperations.points_y.Add((int)prev_anchor.Y);
                  ImageOperations.points_x.Add(e.Y);
                  ImageOperations.points_y.Add(e.X);
                  IMGG = ImageOperations.start(IMGG, s);
                  foreach (Vector2D s in ImageOperations.back)
                  {
                      lastClick.SetPixel((int)(s.Y), (int)(s.X), Color.Black);
                      Vector2D point = new Vector2D();
                      point.X = (int)s.X;
                      point.Y = (int)s.Y;
                      if (ImageOperations.dx_min > (int)s.X)
                      {
                          ImageOperations.dx_min = (int)s.X; }
                      if (ImageOperations.dx_max < (int)s.X)
                          ImageOperations.dx_max = (int)s.X;
                      if (ImageOperations.dy_min > (int)s.Y)
```



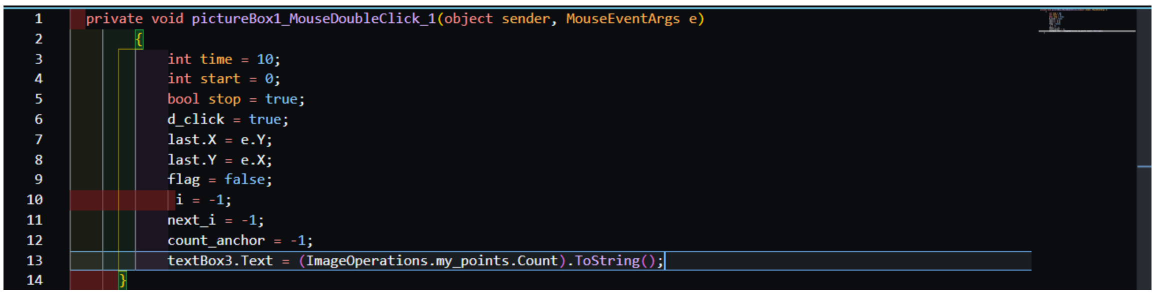
```
                        ImageOperations.dy_min = (int)s.Y;
                    if (ImageOperations.dy_max < (int)s.Y)
                        ImageOperations.dy_max = (int)s.Y;
                    ImageOperations.my_points[point] = true;
                }
                pictureBox1.Image = lastClick;
                pictureBox1.Refresh();
            }
            ImageOperations.points_x = new List<int>();
            ImageOperations.points_y = new List<int>();
        }
        if (flag)
        {
            ImageOperations.points_x.Add(e.Y);
            ImageOperations.points_y.Add(e.X);
            if (ii > 0)
            {
                IMGG = ImageOperations.start(IMGG, s);
                Bitmap bb = new Bitmap(lastClick);
                pictureBox1.Image = lastClick;
                pictureBox1.Refresh();
                foreach (Vector2D s in ImageOperations.back)
                {
                    bb.SetPixel((int)(s.Y), (int)(s.X), Color.Black);
                }
                pictureBox1.Image = bb;
```



```
        if (flag)
        {
            ImageOperations.points_x.Add(e.Y);
            ImageOperations.points_y.Add(e.X);
            if (ii > 0)
            {
                IMGG = ImageOperations.start(IMGG, s);
                Bitmap bb = new Bitmap(lastClick);
                pictureBox1.Image = lastClick;
                pictureBox1.Refresh();
                foreach (Vector2D s in ImageOperations.back)
                {
                    bb.SetPixel((int)(s.Y), (int)(s.X), Color.Black);
                }
                pictureBox1.Image = bb;
                pictureBox1.Refresh();
                s++;
            }
            ii++;
        }
        textBox1.Text = e.X.ToString();
        textBox2.Text = e.Y.ToString();
    }
```

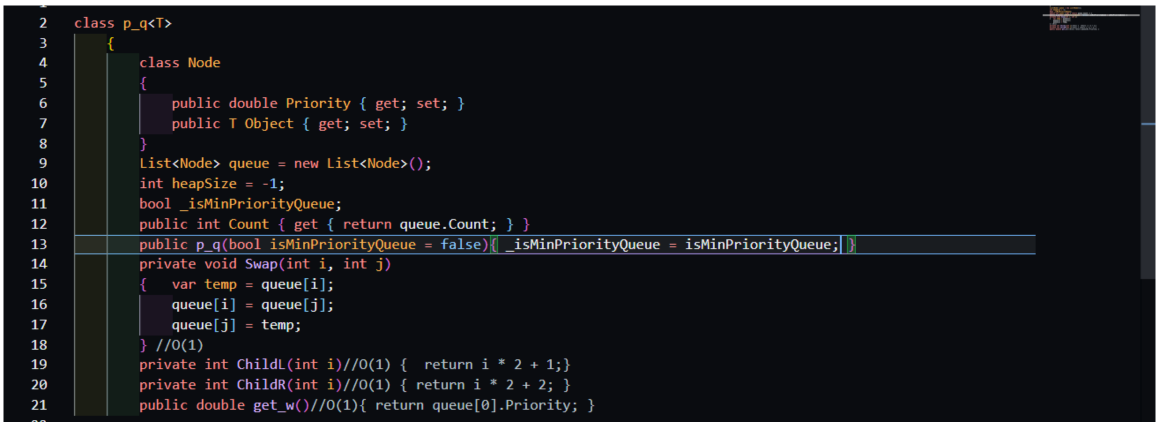
7) Function Mouse_Double_Click

when we double click the mouse we calculate the shortest path between first anchor point and last anchor point and draw it by using a Boolean variable to do this operation in the (mouse move) function then we reset every variable to stop the application logic and to start again as we like.
With every double_click: all this statement is O(1)



```
private void pictureBox1_MouseDoubleClick_1(object sender, MouseEventArgs e)
    {
        int time = 10;
        int start = 0;
        bool stop = true;
        d_click = true;
        last.X = e.Y;
        last.Y = e.X;
        flag = false;
        i = -1;
        next_i = -1;
        count_anchor = -1;
        textBox3.Text = (ImageOperations.my_points.Count).ToString();
    }
```

8) Prirority Queue

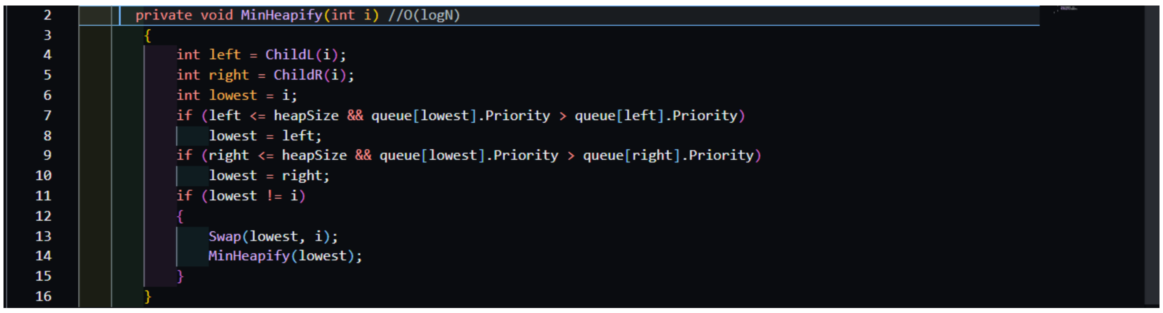


```
class p_q<T>
    {
        class Node
        {
            public double Priority { get; set; }
            public T Object { get; set; }
        }
        List<Node> queue = new List<Node>();
        int heapSize = -1;
        bool _isMinPriorityQueue;
        public int Count { get { return queue.Count; } }
        public p_q(bool isMinPriorityQueue = false){ _isMinPriorityQueue = isMinPriorityQueue; }
        private void Swap(int i, int j)
        {   var temp = queue[i];
            queue[i] = queue[j];
            queue[j] = temp;
        } //O(1)
        private int ChildL(int i)//O(1) {  return i * 2 + 1;}
        private int ChildR(int i)//O(1) { return i * 2 + 2; }
        public double get_w()//O(1){ return queue[0].Priority; }
```

9) Function Min_Heapify

this function put the minimum weight in the root noted that we check for check for the left and right child is smaller than than the heap_size to ensure that the node has left and right child, as it is binary heap so the max number of nodes = (2 power (h+1)) -1 where (h) is the height of the tree ,so the relation between height and number of nodes is that height = log(N)

Time Complexity : it depends on the height as we check for the left and right child of the node which is O(2*log(N)) which is equal O(log(N).

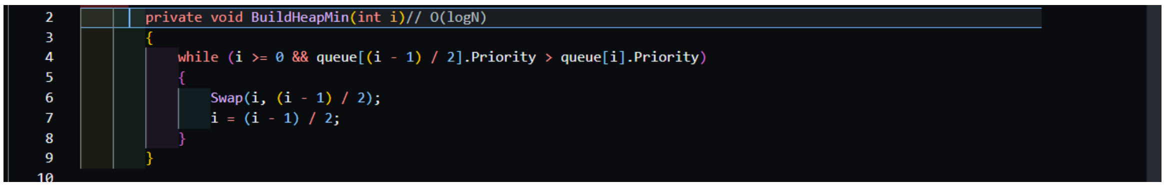


```
        private void MinHeapify(int i) //O(logN)
        {
            int left = ChildL(i);
            int right = ChildR(i);
            int lowest = i;
            if (left <= heapSize && queue[lowest].Priority > queue[left].Priority)
                lowest = left;
            if (right <= heapSize && queue[lowest].Priority > queue[right].Priority)
                lowest = right;
            if (lowest != i)
            {
                Swap(lowest, i);
                MinHeapify(lowest);
            }
        }
```

10) Function Build_Heap

this function when we add a new element we want to put it in the best place in the tree which achieve the goal of this tree which make each sub tree its node is the minimum weight but her childs bigger than it in the weight so this function depends on the height of the tree as it is when we add a new element we check for its parent is it is bigger than it or not if it is swap it to make the weight of the parent smaller else stop checking.

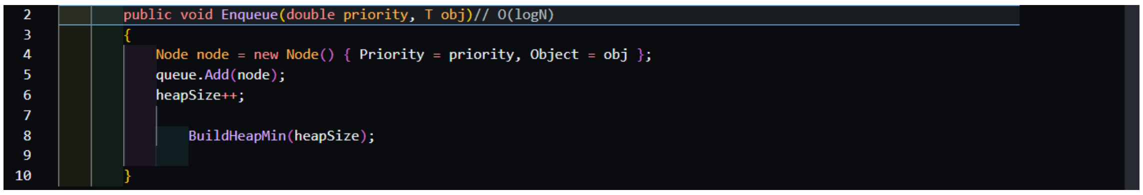
Time Complexity: O(log(N)) as it is depends on the height.



```
private void BuildHeapMin(int i)// O(logN)
{
    while (i >= 0 && queue[(i - 1) / 2].Priority > queue[i].Priority)
    {
        Swap(i, (i - 1) / 2);
        i = (i - 1) / 2;
    }
}
```

111) Function Eneque

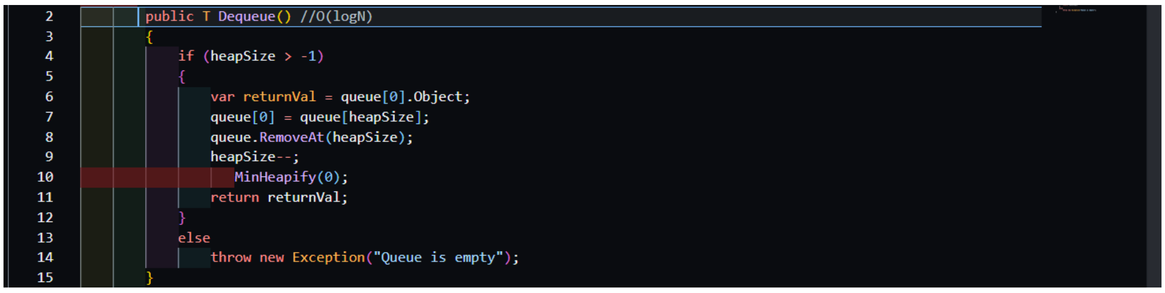
this function call BuildHeapMin when we add a new element to put his in the appropriate place in the tree.
Time Complexity: O(log(N))



```
public void Enqueue(double priority, T obj)// O(logN)
{
    Node node = new Node() { Priority = priority, Object = obj };
    queue.Add(node);
    heapSize++;

        BuildHeapMin(heapSize);

}
```

12) Function Dequeue

this function return the root of the tree which has the minimum weight which take O(1) but after that we want to make the root has the minimum weight after removing the returned root so, we use the (MinHeapify) to achieve that goal which take O(log(N)) so , the final time complexity is O(log(N).



```
public T Dequeue() //O(logN)
{
    if (heapSize > -1)
    {
        var returnVal = queue[0].Object;
        queue[0] = queue[heapSize];
        queue.RemoveAt(heapSize);
        heapSize--;
          MinHeapify(0);
        return returnVal;
    }
    else
        throw new Exception("Queue is empty");
}
```

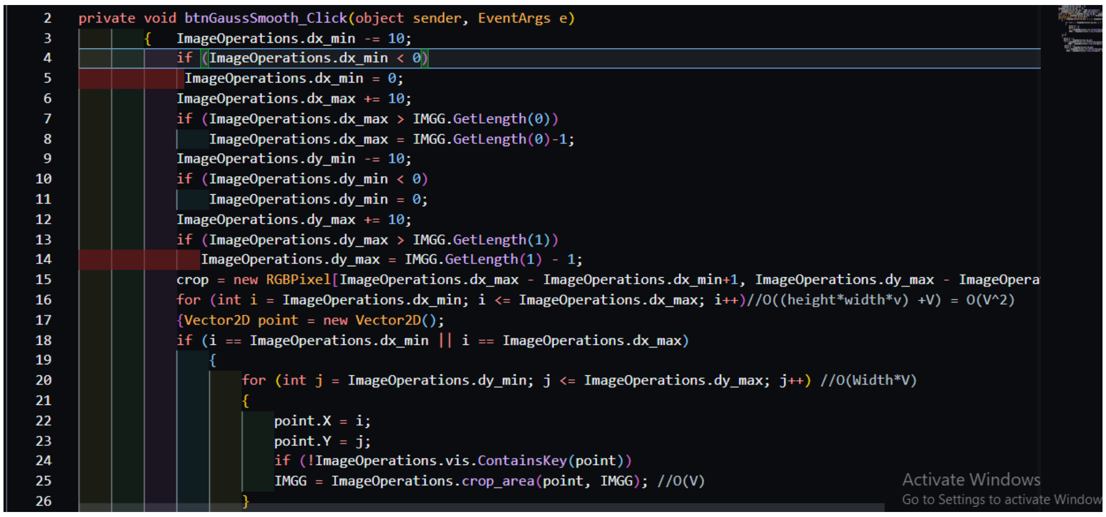
Bonuses:

1) Automatic anchor : by check the checked_box in the form we apply a a automatic anchor by initializing a counter = 10 and another iterative =0 and when the cursor move the iterative is incremented until it equal the counter after that the automatic anchor is applied in the form after that the iterative become zero and do the samething to apply another automatic anchor.

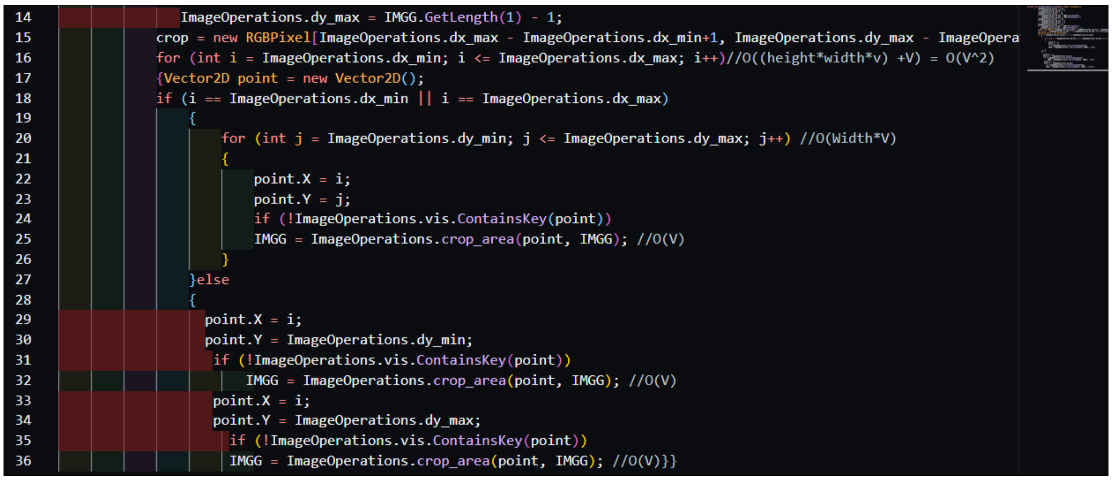
2) Automatic anchor with frequency : frequency is a text_box where the user put it to modify the speed of putting automatic anchor when the frequency increase the time to put anchor point increase when the user enter the frequency the counter is incremented by that frequency.

3) Crop the image: by applying the fill area algorithm we can use it to crop the selected path in the image as we keep track of the minimum point of x and y and the max of them and make shape around the path and visit only the boundaries of the shape that is not visited and when we visit it is visit its adjacent and draw a white color and stop drawing until it visit one point of path that we want to crop.

In this function: we keep track of the boundaries to visit its adjacent which it take O(V)*O(V)= O(V^2) , then map the selected crop of the image in another variable and display it in the form which it take O(V)



```
private void btnGaussSmooth_Click(object sender, EventArgs e)
    {   ImageOperations.dx_min -= 10;
        if (ImageOperations.dx_min < 0)
         ImageOperations.dx_min = 0;
        ImageOperations.dx_max += 10;
        if (ImageOperations.dx_max > IMGG.GetLength(0))
            ImageOperations.dx_max = IMGG.GetLength(0)-1;
        ImageOperations.dy_min -= 10;
        if (ImageOperations.dy_min < 0)
            ImageOperations.dy_min = 0;
        ImageOperations.dy_max += 10;
        if (ImageOperations.dy_max > IMGG.GetLength(1))
           ImageOperations.dy_max = IMGG.GetLength(1) - 1;
        crop = new RGBPixel[ImageOperations.dx_max - ImageOperations.dx_min+1, ImageOperations.dy_max - ImageOpera
        for (int i = ImageOperations.dx_min; i <= ImageOperations.dx_max; i++)//O((height*width*v) +V) = O(V^2)
        {Vector2D point = new Vector2D();
        if (i == ImageOperations.dx_min || i == ImageOperations.dx_max)
            {
                for (int j = ImageOperations.dy_min; j <= ImageOperations.dy_max; j++) //O(Width*V)
                {
                    point.X = i;
                    point.Y = j;
                    if (!ImageOperations.vis.ContainsKey(point))
                    IMGG = ImageOperations.crop_area(point, IMGG); //O(V)
                }
```

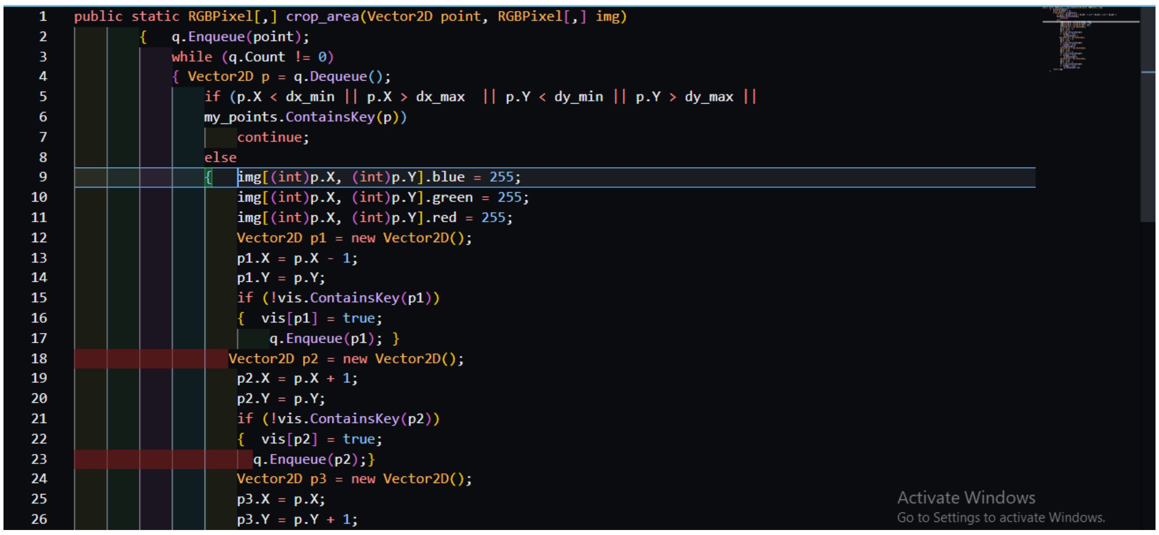


```
           ImageOperations.dy_max = IMGG.GetLength(1) - 1;
        crop = new RGBPixel[ImageOperations.dx_max - ImageOperations.dx_min+1, ImageOperations.dy_max - ImageOpera
        for (int i = ImageOperations.dx_min; i <= ImageOperations.dx_max; i++)//O((height*width*v) +V) = O(V^2)
        {Vector2D point = new Vector2D();
        if (i == ImageOperations.dx_min || i == ImageOperations.dx_max)
            {
                for (int j = ImageOperations.dy_min; j <= ImageOperations.dy_max; j++) //O(Width*V)
                {
                    point.X = i;
                    point.Y = j;
                    if (!ImageOperations.vis.ContainsKey(point))
                    IMGG = ImageOperations.crop_area(point, IMGG); //O(V)
                }
            }else
            {
              point.X = i;
              point.Y = ImageOperations.dy_min;
               if (!ImageOperations.vis.ContainsKey(point))
                   IMGG = ImageOperations.crop_area(point, IMGG); //O(V)
               point.X = i;
               point.Y = ImageOperations.dy_max;
                 if (!ImageOperations.vis.ContainsKey(point))
                 IMGG = ImageOperations.crop_area(point, IMGG); //O(V)}}
```

13) Function Crop_area

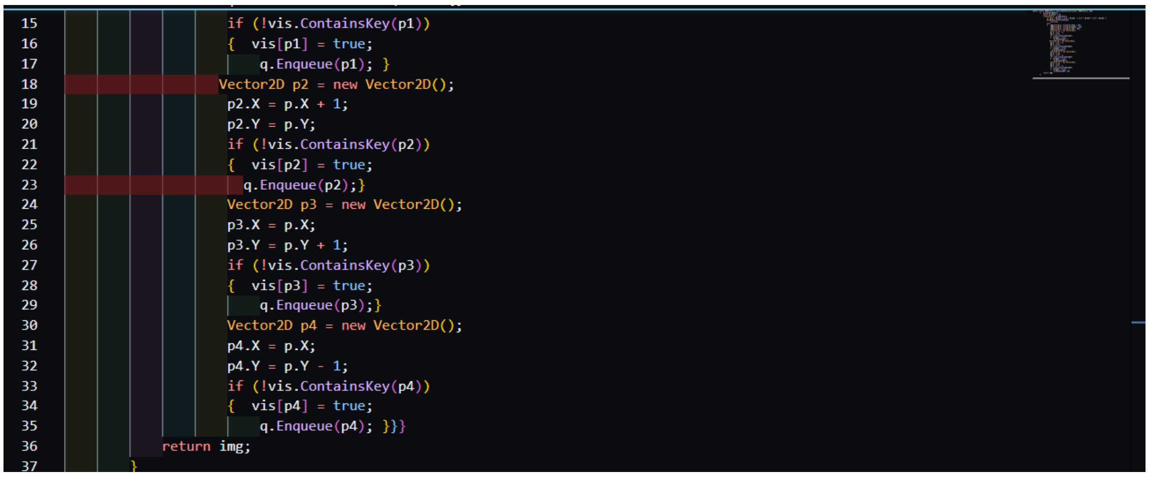
this function is like to (BFS Search) as it when we select a point in the boundary of the shape it is visiting its adjacent.

Time Complexity: O(width*height)=O(V)



```
public static RGBPixel[,] crop_area(Vector2D point, RGBPixel[,] img)
        {   q.Enqueue(point);
            while (q.Count != 0)
            { Vector2D p = q.Dequeue();
                if (p.X < dx_min || p.X > dx_max  || p.Y < dy_min || p.Y > dy_max ||
                my_points.ContainsKey(p))
                    continue;
                else
                {   img[(int)p.X, (int)p.Y].blue = 255;
                    img[(int)p.X, (int)p.Y].green = 255;
                    img[(int)p.X, (int)p.Y].red = 255;
                    Vector2D p1 = new Vector2D();
                    p1.X = p.X - 1;
                    p1.Y = p.Y;
                    if (!vis.ContainsKey(p1))
                    {  vis[p1] = true;
                        q.Enqueue(p1); }
                   Vector2D p2 = new Vector2D();
                    p2.X = p.X + 1;
                    p2.Y = p.Y;
                    if (!vis.ContainsKey(p2))
                    {  vis[p2] = true;
                      q.Enqueue(p2);}
                    Vector2D p3 = new Vector2D();
                    p3.X = p.X;
                    p3.Y = p.Y + 1;
```

Activate Windows
Go to Settings to activate Windows.



```
                    if (!vis.ContainsKey(p1))
                    {  vis[p1] = true;
                        q.Enqueue(p1); }
                   Vector2D p2 = new Vector2D();
                    p2.X = p.X + 1;
                    p2.Y = p.Y;
                    if (!vis.ContainsKey(p2))
                    {  vis[p2] = true;
                      q.Enqueue(p2);}
                    Vector2D p3 = new Vector2D();
                    p3.X = p.X;
                    p3.Y = p.Y + 1;
                    if (!vis.ContainsKey(p3))
                    {  vis[p3] = true;
                        q.Enqueue(p3);}
                    Vector2D p4 = new Vector2D();
                    p4.X = p.X;
                    p4.Y = p.Y - 1;
                    if (!vis.ContainsKey(p4))
                    {  vis[p4] = true;
                        q.Enqueue(p4); }}}
            return img;
        }
```