Práctica 4

Kd-Tree

J. P. Abarca[1] C. T. Apari[1] C. A. Suca[1] A. Vargas[1]

[1]Universidad Nacional de San Agustín. Facultad de Producción y Servicios.
Escuela Profesional de Ciencias de la Computación
Maestría en Ciencias de la Computación
Docente: Mg. Vicente Machaca

24 de Julio del 2021

Contenido

Contenido

Definición

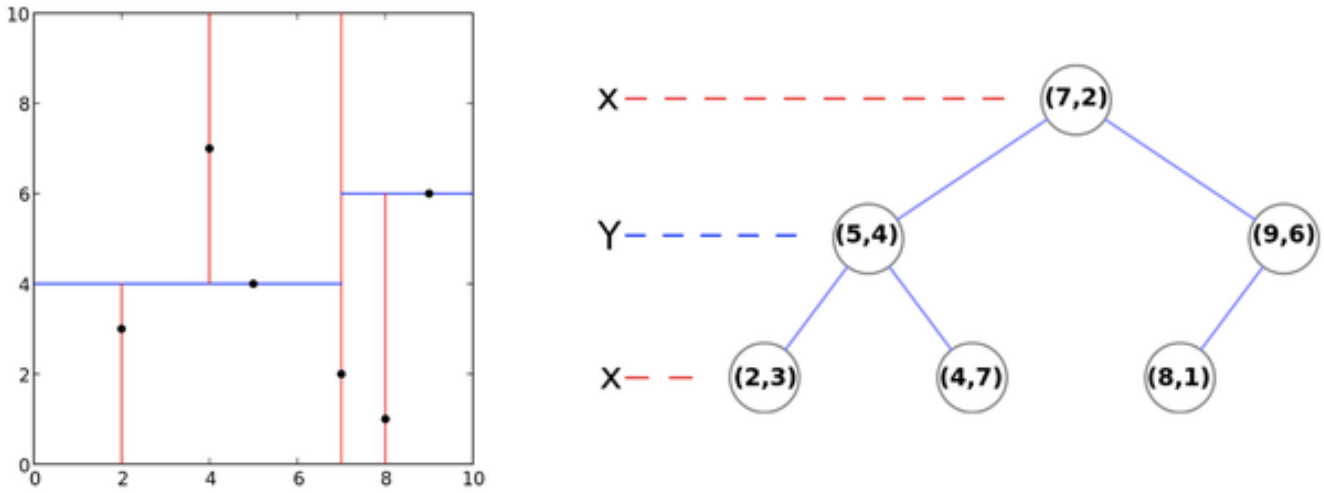

Figura: Construccion del Kd-Tree [1]

Build Kd-Tree

```
1  function build_kdtree(points, depth = 0) {
2      // verifica si esta vacio
3      var n = points.length;
4   var axis = depth % k;
5
6   if ( n<=0){
7    return null;
8   }
9   if(n==1){
10   return new Node(points[0],axis)
11  }
12  var median = Math.floor(points.length/2);
13
14  points.sort(function(a,b){
15   return a[axis] - b[axis];
16  });
17
18  var left = points.slice(0,median);
19  var right = points.slice(median+1);
20
21  var node = new Node(points[median].slice(0,k),axis);
22  node.left = build_kdtree(left,depth +1);
23  node.right = build_kdtree(right,depth +1);
24
25  return node;
26 }
```

Get Height y Generate Dot

- *get_Height*, Recursivamente retornamos la altura del nodo izquierdo y derecho, respecto al valor que estamos por recibir tanto de la altura izquierda y derecha
- *generate_Dot*, Requerimos solamente imprimir el punto padre que este apuntando a un hijo recursivamente para luego obtener el resultado mediante el graficador Graphviz de Python

Implementación

Directamente comparar la distancia del punto a consultar con todos los puntos existentes en la lista completa generada.

```
1  function closest_point_brute_force(points, point) {
2      var distance = null;
3      var best_distance = null;
4      var best_point = null;
5      for (let i = 0; i < points.length; i++) {
6          distance = distanceSquared(points[i], point);
7          if (best_distance === null || distance < best_distance) {
8              best_distance = distance;
9              best_point = points[i];
10         }
11     } return best_point;
12 }
```

Listing 2: closestPointBruteForce

Implementación

Tomando el nodo actual desde el padre hacia los hijos, claramente tiene el mismo comportamiento del que se espera del binary tree [2].

```
function naive_closest_point(node, point, depth = 0, best = null) {
    if(node == null)
        return best;

 var axis = depth % k;

 var best1 = null;
 var camino = null;

 if(best==null)
  best1=node.point;

 if(point[axis]>node.point[axis])
  camino=node.right
 else
  camino=node.left
 return naive_closest_point(camino,point,depth+1,best1)
}
```

Listing 3: naiveClosestPoint

Conjunto de datos (1)

```
var data = [
            [40 ,70] ,
            [70 ,130] ,
            [90 ,40] ,
            [110 , 100] ,
            [140 ,110] ,
            [160 , 100]
            ];
            var point = [140 ,90];
```

Conjunto de datos (1)

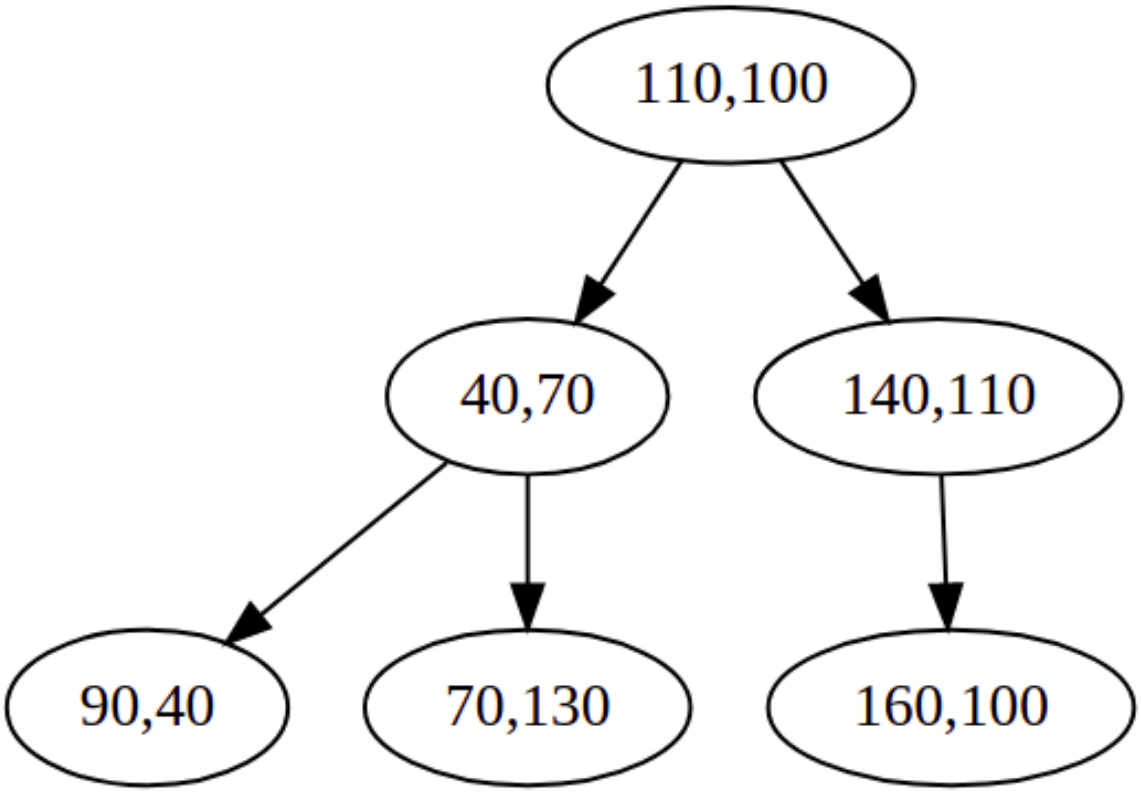

Figura: Resultado en GraphViz

Conjunto de datos (2)

```
var data = [
        [40 ,70] ,
        [70 ,130] ,
        [90 ,40] ,
        [110 , 100] ,
        [140 ,110] ,
        [160 , 100] ,
        [150 , 30]
        ];
        var point = [140 ,90];
```

Conjunto de datos (2)

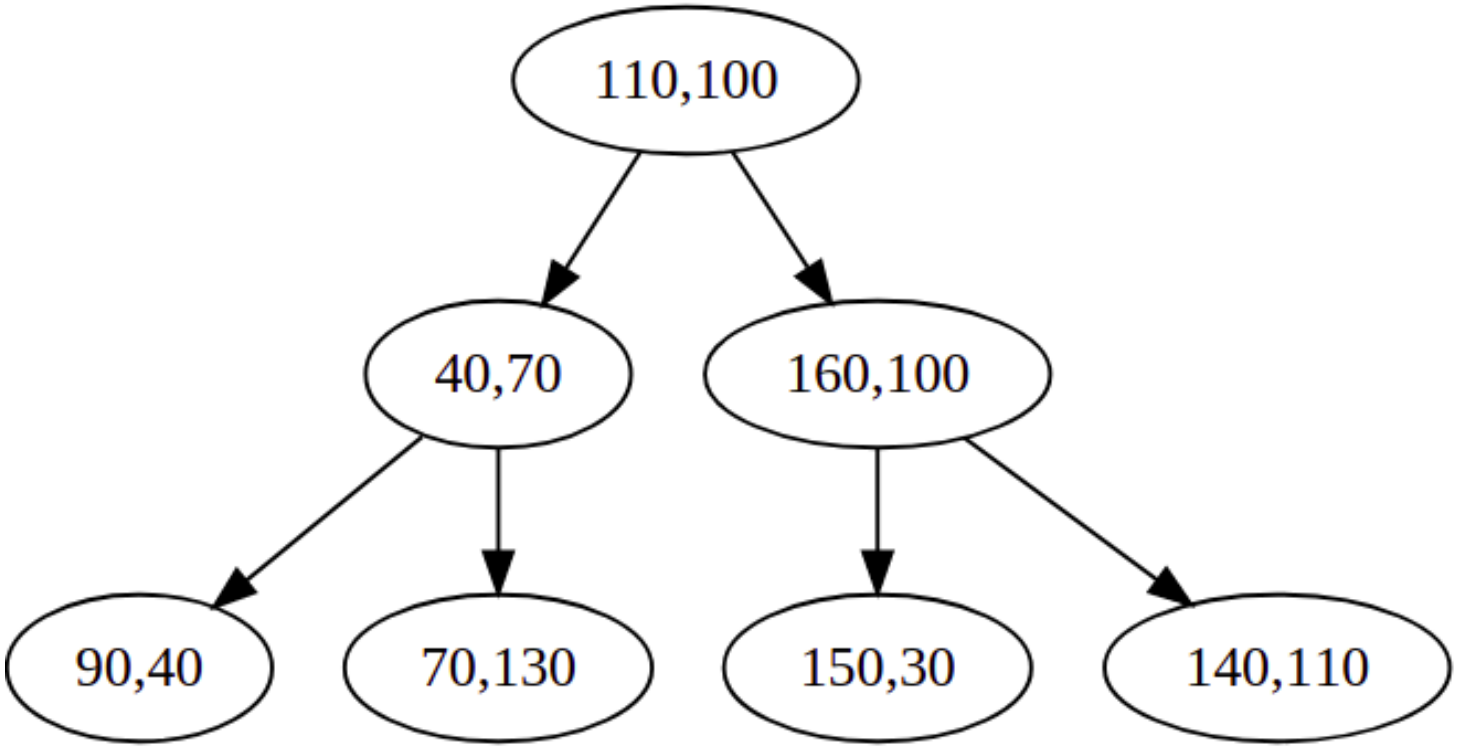

Figura: Resultado en GraphViz

Contenido

Implementación

```
1  function closest_point(node , point , depth = 0, best=null)
2  {
3   if(node==null)return best;
4
5   var axis=depth % k;
6   var camino = null;
7
8   if(best==null)
9   {
10   best1=node.point;
11  }
12  else if((distanceSquared(best, point)> distanceSquared(node.point,point)))
13   best1=node.point;
14  else
15   best1=best;
16
17  if(point[axis]>node.point[axis])
18   camino=node.right;
19
20  else
21   camino=node.left;
22
23  return closest_point(camino,point,depth+1,best1)
24 }
```

Listing 4: closestPoint

Implementación

```
1  function knn(node, puntoConsulta, kpoints, depth = 0) {
2      if (!node) {
3        return null;
4      }
5
6      var temp;
7      var subTree1 = node.left;
8      var subTree2 = node.right;
9
10     if (puntoConsulta[depth % k] >= node.point[depth % k]) {
11       subTree1 = node.right;
12       subTree2 = node.left;
13     }
14
15     // Mejor distancia entre el padre y el subTree1.
16     masCercano(puntoConsulta, knn(subTree1, puntoConsulta, kpoints, depth +1), node.point
           );
17
18     // Variable para ordenar por distancias
19     const sortByDistance = (a, b) => a[2] - b[2];
20
21     if (kpoints.length < vecinos) {
```

Listing 5: KNN1

Implementación

```
1       temp = node.point;
2       temp.push(Math.round(distanceSquared(puntoConsulta, temp)*100)/100);
3       kpoints.push(temp);
4       kpoints.sort(sortByDistance);
5     } else {
6       temp = node.point;
7       temp.push(Math.round(distanceSquared(puntoConsulta, temp)*100)/100);
8       if (temp[2] < kpoints[kpoints.length - 1][2]) {
9         kpoints.pop();
10        kpoints.push(temp);
11        kpoints.sort(sortByDistance);
12      }
13    }
14
15    if(kpoints.length < vecinos || kpoints[0][2] >= Math.abs(puntoConsulta[depth % k] -
          node.point[depth%k])) {
16      masCercano(puntoConsulta, knn(subTree2, puntoConsulta, kpoints, depth +1), node.
            point);
17    }
18  }
```

Listing 6: KNN2

Revisión (1)

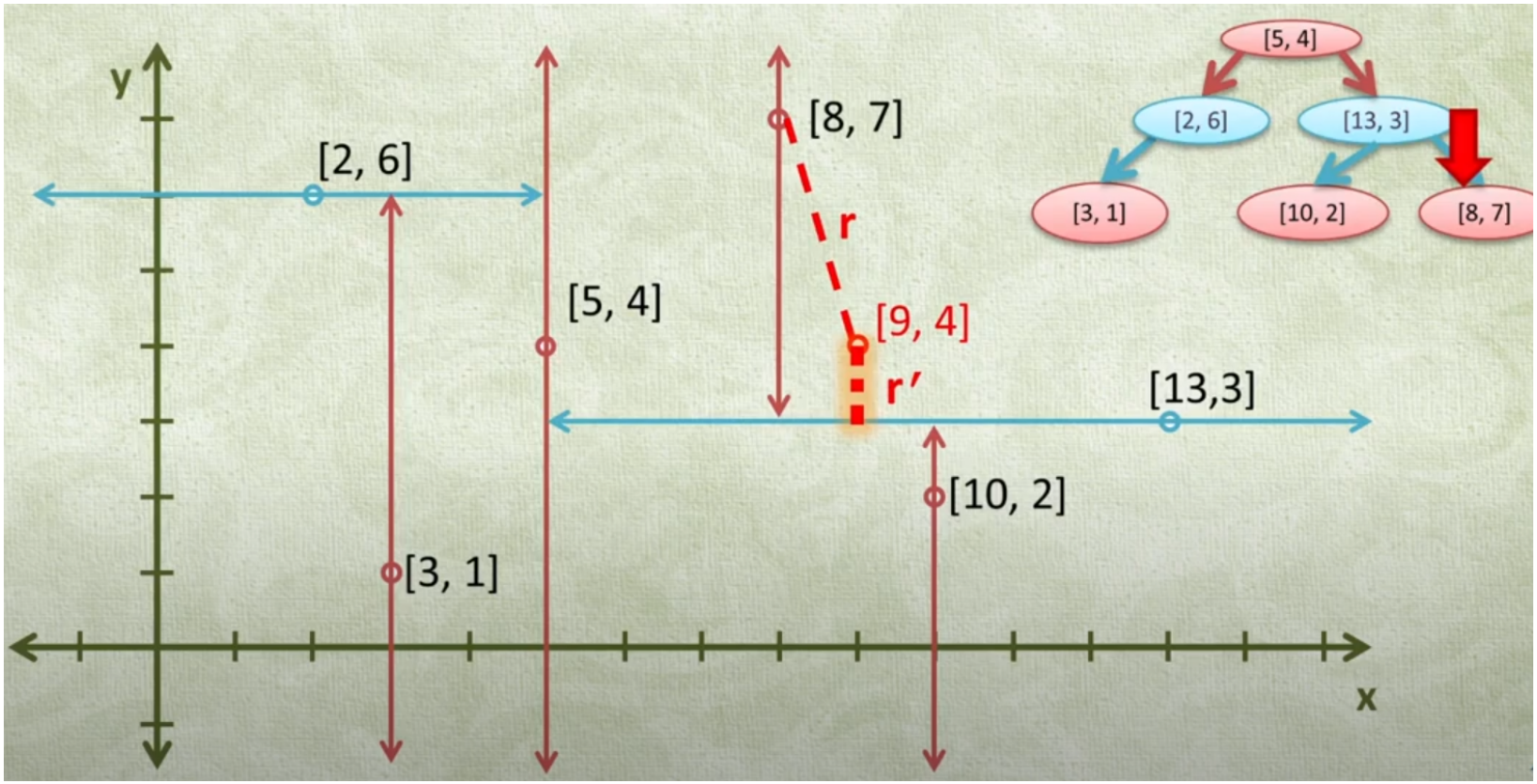

Figura: Punto de revisión

Revisión (2)

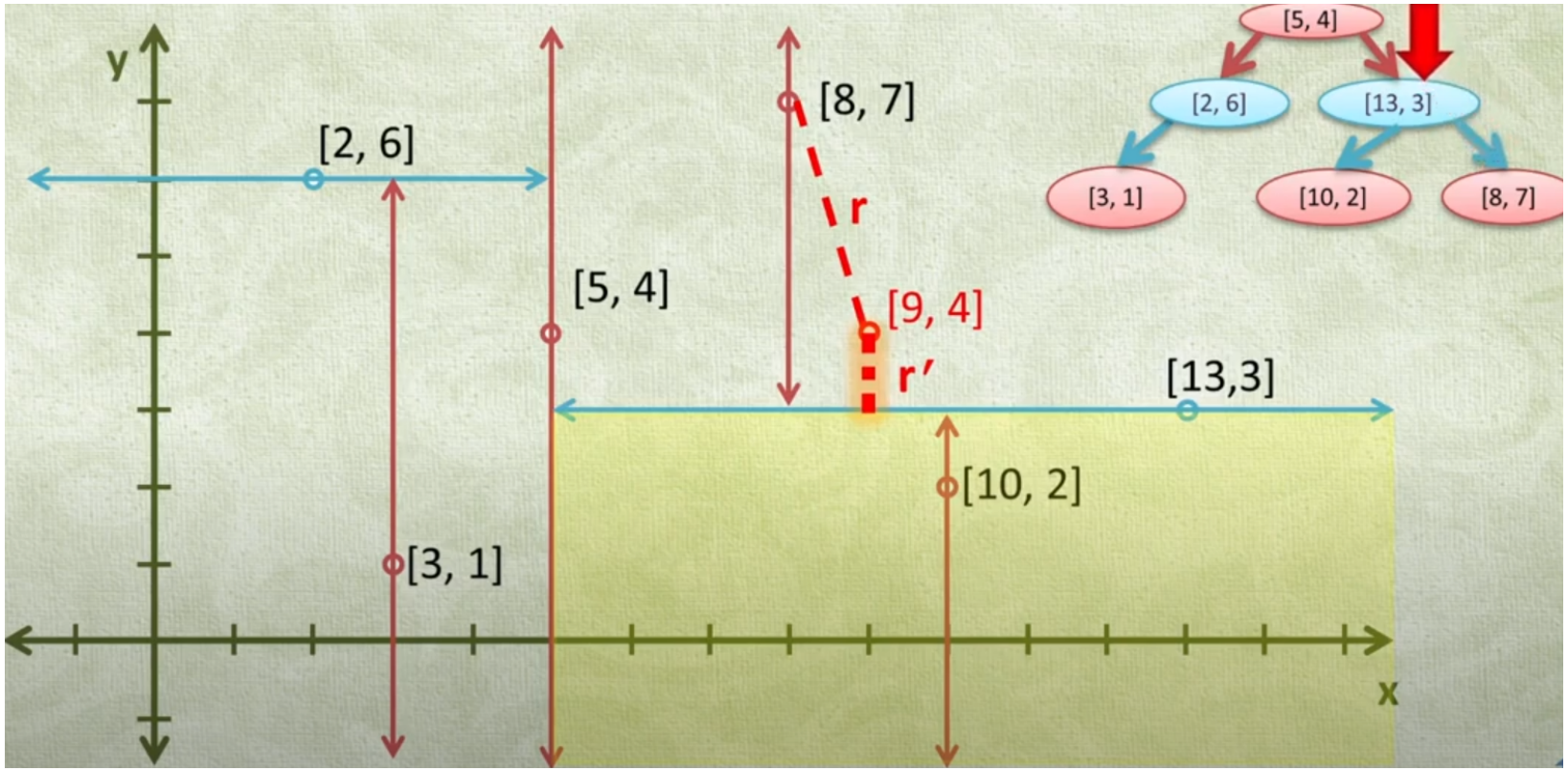

Figura: Punto de revisión

Función *range_query_circle* del KD-Tree

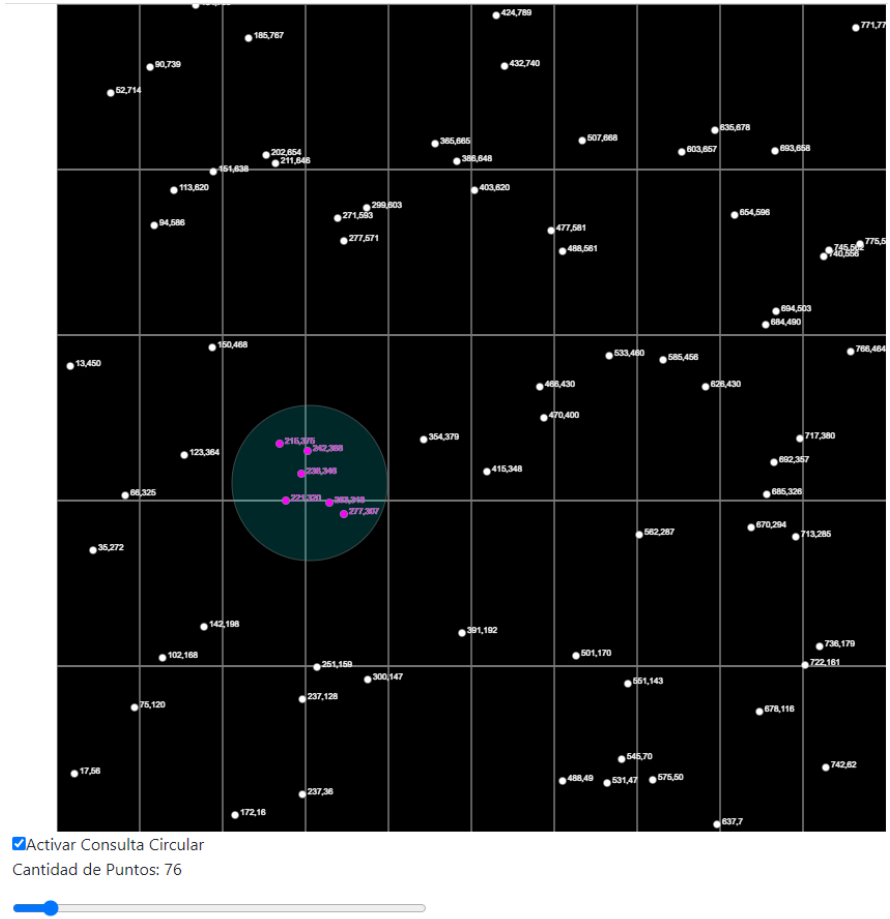

Figura: Consulta Circular

Función *range_query_rect* del KD-Tree

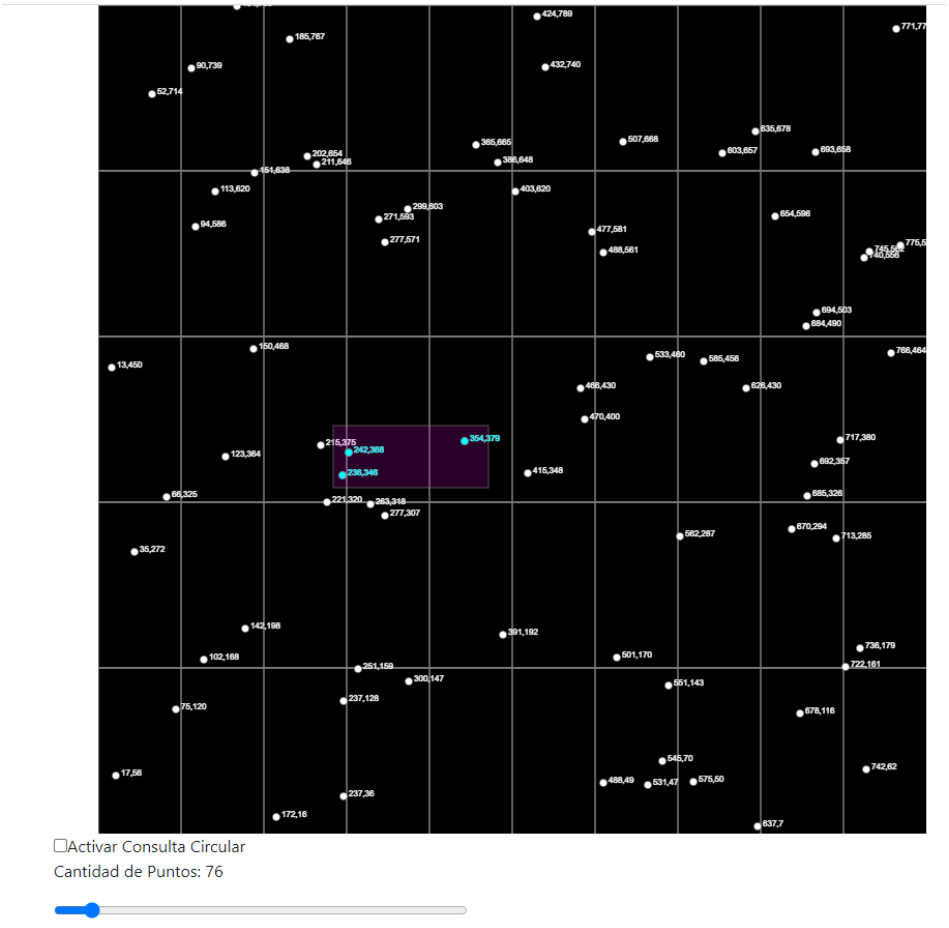

Figura: Consulta Rectangular

GRACIAS

Repositorio: `https://github.com/jabarcamu/EDA_Practica4`

Contenido

References I

[1] H. Samet, *Foundations of multidimensional and metric data structures.*

Morgan Kaufmann series in data management systems, Academic Press, 2006.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*.
Springer-Verlag, second ed., 2000.