

Práctica 04

DOCENTE	CARRERA	CURSO
Vicente Machaca Arceda	Maestría en Ciencia de la Computación	Algoritmos y Estructura de Datos

PRÁCTICA	TEMA	DURACIÓN
04	Kd-tree	3 horas

1. Datos de los estudiantes

- Grupo: 07
- Integrantes:
 - Acuña Melo, Edgar
 - López Torres Herrera, Luis Rodrigo
 - Luna Flores, Julio Paolo
 - Vilela Arias, Roger

2. Introducción

En el siguiente trabajo se realizó el análisis e implementación de algoritmos eficientes que resuelven problemas computacionales, mediante la estructura de datos multidimensional Kd-Tree.

3. Desarrollo del Ejercicio

1. Cree un archivo main.html.

```
1  <html>
2    <head>
3      <title>Kd tree</title>
4      <script src="p5.min.js"></script>
5      <script src="kdtree.js"></script>
6      <script src="sketch.js"></script>
7    </head>
8    <body>
9    </body>
10 </html >
```

2. Cree un archivo kdtree.js.

Complete las funciones:

- build kdtree: Construye el KD-Tree y retorna el nodo raíz.
- getHeight: Retorna la altura del árbol.

- generate dot: Genera al árbol en formato dot.

```
1  k = 2;
2
3  class Node {
4      constructor (point, axis) {
5          this.point = point;
6          this.left = null;
7          this.right = null;
8          this.axis = axis;
9      }
10 }
11
12 function getHeight (node) {
13     // caso base: el árbol vacío tiene una altura de -1
14     if (node == null) {
15         return -1;
16     }
17
18     // recorre al subárbol izquierdo y derecho y considera la altura máxima
19     return 1 + Math.max(getHeight(node.left), getHeight(node.right));
20 }
21
22 function dibujarIzquierda(node) {
23
24     if (node.left == null) {
25         return "";
26     } else {
27         let texto = "";
28         if (node.left != null && node.right) {
29             texto = node.point[0] + ", " + node.point[1] + "-> " + node.left.point[0] + ", " + node.left.point[1] + ";\n"
30             texto += dibujarIzquierda(node.left);
31             texto += dibujarDerecha(node.left);
32         }
33         return texto;
34     }
35 };
36
37
38 function dibujarDerecha(node) {
39
40     if (node.right == null) {
41         return "";
42     } else {
43         let texto = "";
44         if (node.right != null && node.left) {
45             texto = node.point[0] + ", " + node.point[1] + "-> " + node.right.point[0] + ", " + node.right.point[1] + ";\n"
46             texto += dibujarDerecha(node.right);
47             texto += dibujarIzquierda(node.right);
48         }
49         return texto;
50     }
51 };
52
```

```
53 function generate_dot (node) {
54     var texto = "digraph G\n"
55         + "{\n"
56         + if (node.point != null) {
57             texto += dibujarIzquierda(node);
58             texto += dibujarDerecha(node);
59         }
60         + "\n}";
61
62     const doc = document.createElement("a");
63     const archivo = new Blob([texto], { type: 'application/msword' });
64     const url = URL.createObjectURL(archivo);
65     doc.href = url;
66     doc.download = "kd-tree.dot";
67     doc.click();
68     URL.revokeObjectURL(url);
69     return texto;
70 }

72 function build_kdtree (points, depth = 0) {
73     var n = points.length;
74     var axis = depth % k;
75
76     if (n <= 0){
77         return null;
78     }
79     if (n == 1){
80         return new Node(points[0], axis)
81     }
82
83     var median = Math.floor(points.length / 2);
84
85     // sort by the axis
86     points.sort(function(a, b) {
87         return a[axis] - b[axis];
88     });
89     //console.log(points);
90
91     var left = points.slice(0, median);
92     var right = points.slice(median + 1);
93
94     //console.log(right);
95
96     var node = new Node(points[median].slice(0, k), axis);
97     node.left = build_kdtree(left, depth + 1);
98     node.right = build_kdtree(right, depth + 1);
99
100     return node;
101 }
102
103 function distanceSquared (point1, point2) {
104     var distance = 0;
105     for (var i = 0; i < k; i++)
106         distance += Math.pow (( point1 [i] - point2 [i] ) , 2) ;
107     return Math.sqrt ( distance );
108 }
```

```
110 function closest_point_brute_force(points, point) {
111     var n = points.length;
112     var d;
113     var menor = -1;
114     var pto = null;
115     for(let i = 0; i < n; i++){
116         d = distanceSquared(point, points[i]);
117         if (menor >= 0){
118             if (d < menor){
119                 pto = points[i];
120                 menor = d;
121             }
122         }else{
123             menor = d;
124             pto = points[i];
125         }
126     }
127     return pto;
128 }
129
130 function naive_closest_point (node, point, depth = 0, best = null ) {
131     if (node == null) {
132         return best;
133     }
134     var axis = depth % k;
135     let dis1 = distanceSquared(node.point, point);
136     let dis2;
137     console.log(depth + ": ");
138     if (best != null) {
139         dis2 = distanceSquared(best, point);
140         best = (dis1 < dis2)? node.point : best;
141         console.log(node.point + ": " + dis1);
142         console.log(best + ": " + dis2);
143         console.log(best + ": " + ((dis1 < dis2)? dis1 : dis2));
144     } else {
145         best = node.point;
146         console.log(best + ": " + dis1);
147     }
148     if (node.left == null && node.right == null) {
149         return best;
150     }
151 }
```

```
160 function closest_point(node , point , depth = 0) {
161     if (node == null) {
162         return null;
163     }
164     var axis = depth % k;
165     let dis1 = distanceSquared(node.point, point);
166     let disLeft;
167     let disRight;
168     let bestLeft;
169     let bestRight;
170
171     console.log(depth + ": ");
172     console.log(depth + ": point => " + node.point);
173     console.log(depth + ": dis1 => " + dis1);
174     if (node.left == null && node.right == null) {
175         console.log(depth + ": return => " + node.point);
176         return node.point;
177     } else {
178         disLeft = (node.left != null)? distanceSquared(node.left.point, point) : null;
179         console.log(depth + ": disLeft => " + disLeft);
180         disRight = (node.right != null)? distanceSquared(node.right.point, point) : null;
181         console.log(depth + ": disRight => " + disRight);
182         if (point[axis] < node.point[axis]) {
183             bestLeft = closest_point(node.left, point, depth + 1);
184             bestRight = (disLeft != null && disLeft > dis1)? closest_point(node.right, point, depth + 1) : null;
185         } else {
186             bestRight = closest_point(node.right, point, depth + 1);
187             bestLeft = (disRight != null && disRight > dis1)? closest_point(node.left, point, depth + 1) : null;
188         }
189     }
190
191     console.log(depth + ": bestLeft => " + bestLeft);
192     console.log(depth + ": bestRight => " + bestRight);
193     if (bestLeft == null && bestRight == null) {
194         console.log(depth + ": return => " + node.point);
195         return node.point;
196     } else if (bestLeft == null) {
197         disRight = distanceSquared(bestRight, point);
198         console.log(depth + ": disRight => " + disRight);
199         return (dis1 < disRight)? node.point : bestRight;
200     } else if (bestRight == null) {
201         disLeft = distanceSquared(bestLeft, point);
202         console.log(depth + ": disLeft => " + disLeft);
203         return (dis1 < disLeft)? node.point : bestLeft;
204     } else {
205         disLeft = distanceSquared(bestLeft, point);
206         console.log(depth + ": disLeft => " + disLeft);
207         disRight = distanceSquared(bestRight, point);
208         console.log(depth + ": disRight => " + disRight);
209         if (disLeft < disRight) {
210             return (dis1 < disLeft)? node.point : bestLeft;
211         } else {
212             return (dis1 < disRight)? node.point : bestRight;
213         }
214     }
215 }
```

```
216 function range_query_circle(node, center, radio, queue, depth=0) {
217   if(node != null)
218   {
219     var dist = distanceSquared(node.point, center);
220     if (dist <= radio){
221       queue.push(node.point);
222     }
223     range_query_circle(node.left, center, radio, queue, depth+1);
224     range_query_circle(node.right, center, radio, queue, depth+1);
225   }
226 }
227
228 function range_query_rec(node, lefttop, rightbottom, queue, depth=0) {
229   if(node != null)
230   {
231     var dentro = node.point[0]>=lefttop[0] && node.point[0]<=rightbottom[0] && node.point[1]<=lefttop[1] && node.point[1]>=rightbottom[1];
232     if (dentro) {
233       queue.push(node.point);
234     }
235     range_query_rec(node.left, center, radio, queue, depth+1);
236     range_query_rec(node.right, center, radio, queue, depth+1);
237   }
238 }
```

3. Cree un archivo sketch.js y evalúe sus resultados.
4. Implemente la función closest point brute force y naive closest point :
5. Evalúe el resultado de las dos funciones implementadas anteriormente con este conjunto de datos:
6. Evalúe el resultado de las dos funciones implementadas anteriormente con este conjunto de datos:
7. Ahora implemente la función closest point, siguiendo las recomendaciones dadas por el docente:

```
1 function setup() {
2   var width = 250;
3   var height = 200;
4   createCanvas(width, height);
5
6   background(0);
7   for (var x = 0; x < width; x += width / 10) {
8     for (var y = 0; y < height; y += height / 5) {
9       stroke(125, 125, 125);
10      strokeWeight(1);
11      line(x, 0, x, height);
12      line(0, y, width, y);
13    }
14  }
15
16  //PUNTO 03
17  /*
18  var data = [];
19  for (let i = 0; i < 12; i++) {
20    var x = Math.floor(Math.random() * height);
21    var y = Math.floor(Math.random() * height);
22    data.push([x, y]);
23
24    fill(255, 255, 255);
25    circle(x, height - y, 7); // 200 -y para q se dibuje apropiadamente
26    textSize(8);
27    text(x + ', ' + y, x + 5, height - y); // 200 -y para q se dibuje apropiadamente
28  }
29
30  var root = build_kdtree(data);
31  console.log(root);
32  */
33 }
```

```
34 //PUNTO 05
35 /*
36 var data = [
37     [40 ,70] ,
38     [70 ,130] ,
39     [90 ,40] ,
40     [110 , 100] ,
41     [140 ,110] ,
42     [160 , 100]
43 ];
44 var point = [140 ,90]; // query
45 for ( let i = 0; i < 6; i ++ ) {
46     fill (255 , 255 , 255) ;
47     circle (data[i][0], height - data[i][1], 7) ; // 200 -y para q se dibuje apropiadamente
48     textSize (8) ;
49     text (data[i][0] + ',' + data[i][1], data[i][0] + 5, height - data[i][1]); // 200 -y para q se dibuje apropiadamente
50 }
51 var root = build_kdtree ( data ) ;
52 console.log ( root );
53
54 var ptol = closest_point_brute_force(data, point);
55 console.log("FzaBruta : " + ptol);
56
57 let best = null;
58 best = naive_closest_point(root, point, 0, best);
59 console.log(best);
60 */
61
62 //PUNTO 06
63 var data = [
64     [40 ,70] ,
65     [70 ,130] ,
66     [90 ,40] ,
67     [110 , 100] ,
68     [140 ,110] ,
69     [160 , 100] ,
70     [150 , 30]
71 ];
```








```
72   var point = [140,90]; // query
73   for ( let i = 0; i < data.length; i++) {
74       fill (255 , 255 , 255) ;
75       circle (data[i][0], height - data[i][1], 7) ; // 200 -y para q se dibuje apropiadamente
76       textSize (8) ;
77       text (data[i][0] + ',' + data[i][1], data[i][0] + 5, height - data[i][1]); // 200 -y para q se dibuje apropiadamente
78   }
79   var root = build_kdtree ( data ) ;
80   console.log ( root );
81   var ptol = closest_point_brute_force(data, point);
82   console.log("FzaBruta : " + ptol);
83
84   let best = null;
85   best = naive_closest_point(root, point, 0, best);
86   console.log("Best naive closest: " + best);
87   best = null;
88   best = closest_point(root, point, 0);
89   console.log("Best closest: " + best);
90
91   //PUNTO 02 subsección 2
92   //Comentar para que no descargue el archivo dot
93   console.log("DOT consola\n" + generate_dot(root));
94 }
```

8. implemente una función KNN, que retorna los k puntos mas cercanos a un punto.
9. Implemente la función range query circle del KD-Tree:
10. Implemente la función range query rec del KD-Tree, esta vez el range representa un rectángulo.

4. Implementación

El desarrollo de la practica se utilizo un repositorio compartido para la practica, los archivos de codigo fuente del desarrollo de la presente practica, se encuentran alojados en el siguiente enlace: <https://github.com/JulioLunaUNSA/INFORME04.git>

 README.md	Initial commit
 kdtree.js	Func. range query rec
 main.html	Commit inicial
 p5.min.js	Commit inicial
 sketch.js	Añadiendo método de generate_dot