

UNIVERSIDAD NACIONAL DE SAN AGUSTIN DE
AREQUIPA



ESTRUCTURA DE DATOS AVANZADOS

QuadTree

Alumna :

Chullunquía Rosas, Sharon Rossely

Profesor :

Machaca Arceda, Vicente Enrique

29 de septiembre de 2020

Pregunta #01

Cree un archivo main.html, este llamara a los archivos javascript que vamos a crear. El archivo p5.min.js es una librería para gráficos, la puede descargar de internet o se la puede pedir al profesor. En el archivo quadtree.js estará todo el código de nuestra estructura y en el archivo sketch.js estará el código donde haremos pruebas con nuestro Quadtree.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>QuadTree</title>
5      <script src = "p5.min.js"></script>
6      <script src = "quadtree.js"></script>
7      <script src = "sketch.js"></script>
8  </head>
9  <body>
10 </body>
11 </html>
```

Respuesta:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>QuadTree</title>
5      <script src = "p5/p5.min.js"></script>
6      <script src = "quadtree.js"></script>
7      <script src = "sketch.js"></script>
8  </head>
9  <body>
10 </body>
11 </html>
```

Código en GitHub : [main.html](#)

Pregunta #02

En el archivo quadtree.js digitemos el siguiente código, además debe completar las funciones contains y intersects (ambas funciones devuelven true o false).

```
1  class Point {
2      constructor (x, y, userData ){
3          this.x = x;
4          this.y = y;
5          this.userData = userData ;
6      }
7  }
8  class Rectangle {
9      constructor (x, y, w, h){
```

```

10         this.x = x; // center
11         this.y = y;
12         this.w = w; // half width
13         this.h = h; // half height
14     }
15     // verifica si este objeto contiene un objeto Punto
16     contains ( point ){
17     }
18     // verifica si este objeto se intersecta con otro objeto
19     Rectangle
20     intersects ( range ){
21     }
22 }

```

Respuesta:

```

1  class Point {
2      constructor (x, y, userData ){
3          this.x = x;
4          this.y = y;
5          this.userData = userData ;
6      }
7  }
8
9  class Rectangle {
10     constructor (x, y, w, h){
11         this.x = x; // center
12         this.y = y;
13         this.w = w; // half width
14         this.h = h; // half height
15     }
16
17     // verifica si este objeto contiene un objeto Punto
18     contains(point) {
19         return (point.x >= this.x - this.w &&
20             point.x <= this.x + this.w &&
21             point.y >= this.y - this.h &&
22             point.y <= this.y + this.h);
23     }
24
25     // verifica si este objeto se intersecta con otro objeto
26     Rectangle
27     intersects(range) {
28         return !(range.x - range.w > this.x + this.w ||
29             range.x + range.w < this.x - this.w ||
30             range.y - range.h > this.y + this.h ||
31             range.y + range.h < this.y - this.h);
32     }
33 }

```

Código en GitHub : [quadtrees.js](#)

Pregunta #03

En el archivo quadtree.js digitemos el siguiente código y complete las funciones subdivide y insert.

```

1  class QuadTree{
2      constructor(boundary, n){
3          this.boundary = boundary ; // Rectangle
4          this.capacity = n; // capacidad maxima de cada cuadrante
5          this.points = []; // vector , almacena los puntos a
              almacenar
6          this.divided = false ;
7      }
8
9      // divide el quadtree en 4 quadtrees
10     subdivide(){
11         // Algoritmo
12         // 1: Crear 4 hijos : qt_northeast , qt_northwest ,
              qt_southeast , qt_southwest
13         // 2: Asignar los QuadTree creados a cada hijo
14         // this . northeast = qt_northeast ;
15         // this . northwest = qt_northwest ;
16         // this . southeast = qt_southeast ;
17         // this . southwest = qt_southwest ;
18         // 3. - Hacer : this . divided <- true
19     }
20     insert(point){
21         // Algoritmo
22         // 1: Si el punto no esta en los limites ( boundary )
              del quadtree Return
23         // 2: Si ( this . points . length ) < ( this . capacity
              ),
24         // 2.1 Insertamos en el vector this . points
25         // Sino
26         // 2.2 Dividimos si aun no ha sido dividido
27         // 2.3 Insertamos recursivamente en los 4 hijos .
28         // this . northeast . insert ( point );
29         // this . northwest . insert ( point );
30         // this . southeast . insert ( point );
31         // this . southwest . insert ( point );
32     }
33     show () {
34         stroke(255) ;
35         strokeWeight(1) ;
36         noFill() ;
37         rectMode(CENTER);
38         rect(this.boundary.x, this.boundary.y, this.boundary.w*2
              , this.boundary.h*2) ;
39         if ( this.divided ){
40             this.northeast.show() ;
41             this.northwest.show() ;
42             this.southeast.show() ;
43             this.southwest.show() ;

```

```

44     }
45     for (let p of this.points ){
46         strokeWeight (4) ;
47         point (p.x, p.y);
48     }
49 }
50 }

```

Respuesta:

```

1  class QuadTree {
2      constructor(boundary, n) {
3          if (!boundary) {
4              throw TypeError('boundary is null or undefined');
5          }
6
7          if (!(boundary instanceof Rectangle)) {
8              throw TypeError('boundary should be a Rectangle');
9          }
10
11         if (typeof n !== 'number') {
12             throw TypeError('capacity should be a number but is
13                 a ${typeof n}');
14         }
15
16         if (n < 1) {
17             throw RangeError('capacity must be greater than 0');
18         }
19
20         this.boundary = boundary; // Rectangle
21         this.capacity = n; // Capacidad maxima de cada cuadrante
22         this.points = []; // Vector, almacena los punto a
23             almacenar
24         this.divided = false;
25     }
26
27     // Divide el quadtree en 4 quadtrees
28     subdivide() {
29         let x = this.boundary.x;
30         let y = this.boundary.y;
31         let w = this.boundary.w / 2;
32         let h = this.boundary.h / 2;
33
34         // 1: Creamos 4 hijos : qt_northeast , qt_northwest,
35             qt_southeast , qt_southwest
36         let qt_northeast = new Rectangle(x + w, y - h, w, h);
37         let qt_northwest = new Rectangle(x - w, y - h, w, h);
38         let qt_southeast = new Rectangle(x + w, y + h, w, h);
39         let qt_southwest = new Rectangle(x - w, y + h, w, h);
40
41         // 2: Asignamos los QuadTree creados a cada hijo

```

```

39     this.northeast = new QuadTree(qt_northeast,
40                                   this.capacity);
41     this.northwest = new QuadTree(qt_northwest,
42                                   this.capacity);
43     this.southeast = new QuadTree(qt_southeast,
44                                   this.capacity);
45     this.southwest = new QuadTree(qt_southwest,
46                                   this.capacity);
47
48     // 3. - Hacemos : this . divided <- true
49     this.divided = true;
50 }
51
52 insert(point) {
53     // 1: Si el punto no esta en los limites ( boundary )
54     // del quadtree Return
55     if (!this.boundary.contains(point)) {
56         return false;
57     }
58
59     // 2: Si ( this . points . length ) < ( this . capacity )
60     if (this.points.length < this.capacity) {
61         this.points.push(point); // 2.1 Insertamos en el
62         // vector this . points
63         return true;
64     }
65     else{
66         // 2.2 Dividimos si aun no ha sido dividido
67         if (!this.divided) {
68             this.subdivide();
69         }
70         // 2.3 Insertamos en los 4 hijos
71         if (this.northeast.insert(point) ||
72             this.northwest.insert(point) ||
73             this.southeast.insert(point) ||
74             this.southwest.insert(point)) {
75             return true;
76         }
77     }
78 }
79
80 show () {
81     stroke(255) ;
82     strokeWeight(1) ;
83     noFill() ;
84     rectMode( CENTER );
85     rect( this.boundary.x, this.boundary.y, this.boundary.w
86           * 2 , this.boundary.h * 2) ;
87     if( this.divided ){
88         this.northeast.show();
89         this.northwest.show();

```

```
81         this.southeast.show();
82         this.southwest.show();
83     }
84     for (let p of this.points ){
85         strokeWeight(4) ;
86         point(p.x, p.y);
87     }
88 }
89 }
```

Código en GitHub : [quadtree.js](#)

Pregunta #04

Editamos el archivo sketch.js. En este archivo estamos creando un QuadTree de tamaño 400x400 con 3 puntos. Ejecute (obtendra un resultado similar a la Figura 1) y comente los resultados (muestre capturas de pantalla).

```
1  let qt;
2  let count = 0;
3  function setup(){
4      createCanvas(400,400) ;
5
6      // centre point and half of width and height
7      let boundary = new Rectangle(200 ,200 ,200 ,200);
8
9      // each leave just could have 4 elements
10     qt = new QuadTree( boundary , 4);
11
12     console.log(qt);
13
14     for (let i =0; i < 3; i ++) {
15         let p = new Point ( Math.random() * 400 , Math.random()
16             * 400) ;
17         qt.insert(p);
18     }
19     background(0) ;
20     qt.show() ;
21 }
```

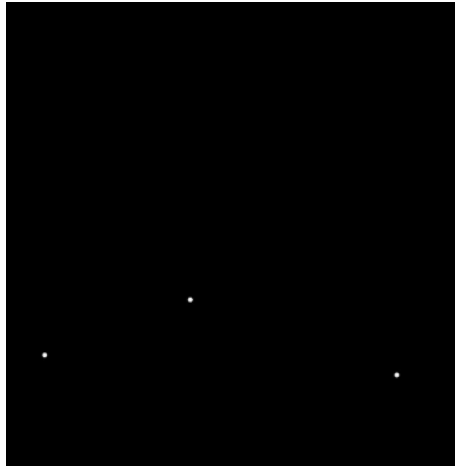


Figura 1: Visualización del QuadTree con 3 datos.

Respuesta:

```
1 let qt;  
2 let count = 0;  
3  
4 function setup() {  
5   createCanvas(400, 400);  
6  
7   // centre point and half of width and height  
8   let boundary = new Rectangle(200, 200, 200, 200);  
9  
10  // each leaf just could have 4 elements  
11  qt = new QuadTree(boundary, 4);  
12  
13  console.log(qt);  
14  
15  for (let i = 0; i < 3; i++) {  
16    let p = new Point(Math.random() * 400, Math.random() *  
17      400);  
18    qt.insert(p);  
19  }  
20  background(0);  
21  qt.show();  
22 }
```

Código en GitHub : [sketch.js](#)

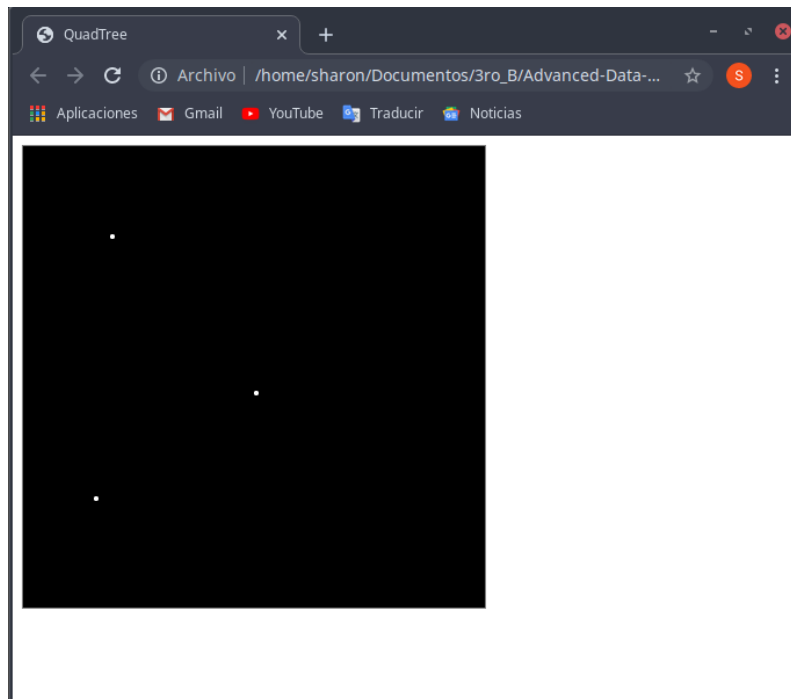


Figura 2: Programa en ejecución del QuadTree con 3 datos.

Pregunta #05

Abra las opciones de desarrollador (opciones/más herramientas/ opciones de desarrollador) de su navegador para visualizar la console. Comente que datos encuentra y muestre una captura de pantalla.

Respuesta:

Los datos que se muestran en la Figura 3 son los siguientes :

- **Boundary** : Representa el tamaño del *QuadTree*, el cual es de 400x400.
- **Capacity** : La capacidad máxima de puntos en cada cuadrante del *QuadTree*, en este caso 4.
- **Divided** : Un booleano que nos indica si el *Quadtree* está dividido, en este caso *false*, porque aún no se han realizado divisiones, así como se puede ver en la Figura 2.
- **Points** : Representa un arreglo de tamaño 3 que almacena puntos, y es de tamaño 3 porque por el momento tenemos sólo 3 puntos creados; también podemos observar que de cada punto se muestra sus coordenadas en el plano bidimensional del *QuadTree*.

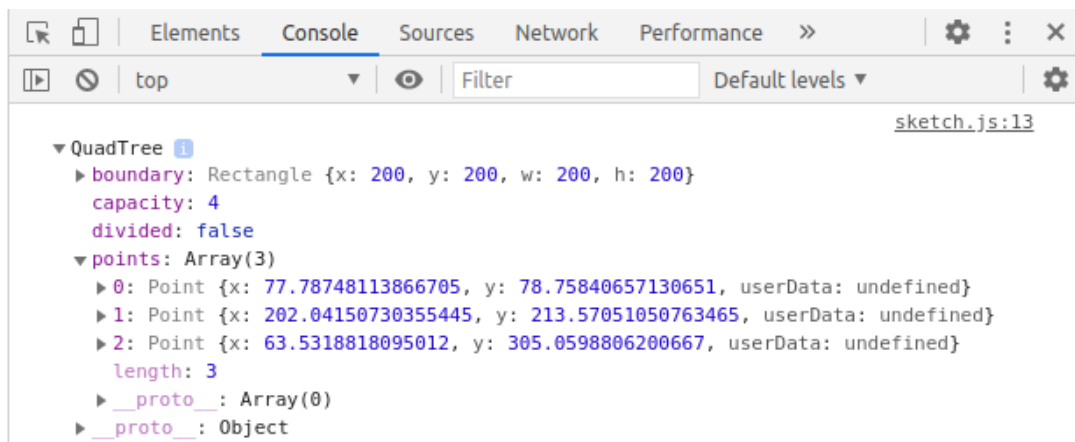


Figura 3: Vista a la Console de las opciones de desarrollador del navegador Web.

Pregunta #06

Inserte más puntos y muestre cómo varían sus resultados.

Respuesta:



Figura 4: Visualización del QuadTree con 8 datos.

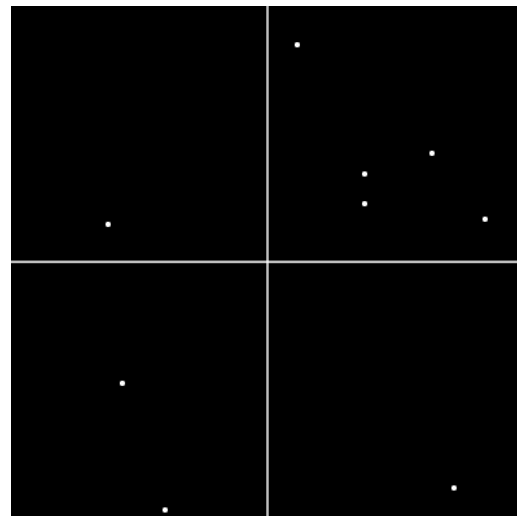


Figura 5: Visualización del QuadTree con 9 datos.

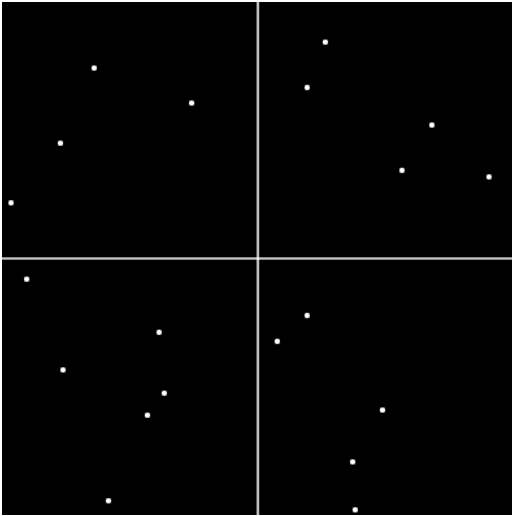


Figura 6: Visualización del QuadTree con 20 datos.

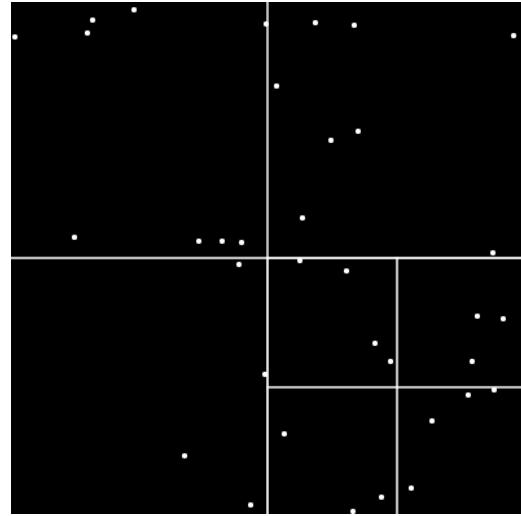


Figura 7: Visualización del QuadTree con 36 datos.

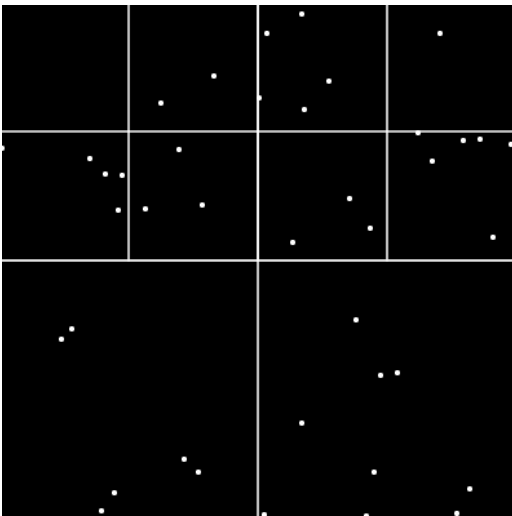


Figura 8: Visualización del QuadTree con 40 datos.

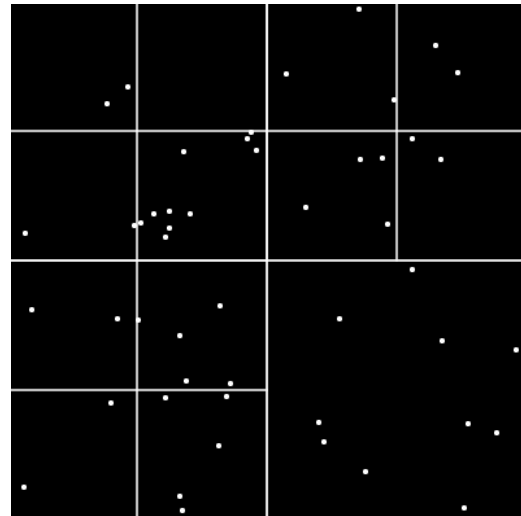


Figura 9: Visualización del QuadTree con 50 datos.

Pregunta #07

Edite el archivo `sketch.js` con el siguiente código. En este caso, nos da la posibilidad de insertar los puntos con el mouse. Muestre sus resultados y comente cómo funciona el código.

```

1  let qt;
2  let count = 0;
3
4  function setup(){
5      createCanvas(400,400);
6      let boundary = new Rectangle(200,200,200,200);
7      qt = new QuadTree(boundary,4);
8  }
9
10 function draw(){
11     background(0);
12     if (mouseIsPressed){
13         for (let i = 0; i < 1; i++){
14             let m = new Point (mouseX + random( -5 ,5) , mouseY
15                 + random( -5 ,5) );
16             qt.insert(m)
17         }
18     }
19     background(0);
20     qt.show();
21 }

```

Respuesta:

```

1  let qt;
2  let count = 0;
3
4  function setup() {
5      createCanvas(400 ,400) ;
6
7      // centre point and half of width and height
8      let boundary = new Rectangle(200 ,200 ,200 ,200);
9
10     // each leave just could have 8 elements
11     qt = new QuadTree( boundary , 8);
12
13     console.log(qt);
14
15     for (let i =0; i < 50; i++) {
16         let p = new Point( Math.random() * 400 , Math.random() *
17             400) ;
18         qt.insert(p);
19     }
20     background(0) ;
21     qt.show() ;
22 }
23
24 function draw() {
25     background(0);
26     if ( mouseIsPressed ) {
27         for (let i = 0; i < 1; i++) {

```

```
27         let m = new Point( mouseX + random( -5 ,5) , mouseY
28             + random( -5 ,5) );
29         qt.insert(m)
30     }
31     background(0);
32     qt.show();
33 }
```

Código en GitHub : [sketch.js](#)

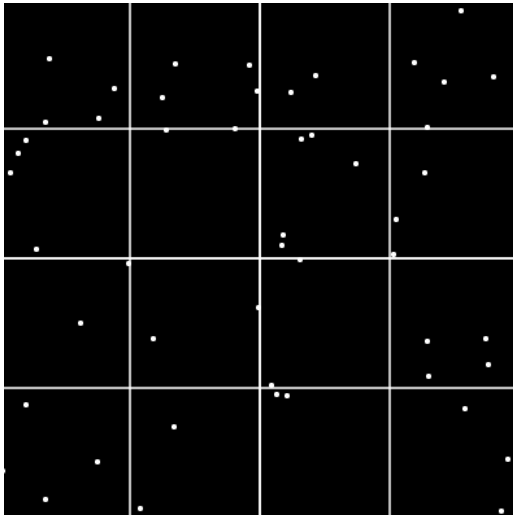


Figura 10: Visualización antes de insertar puntos con el mouse.

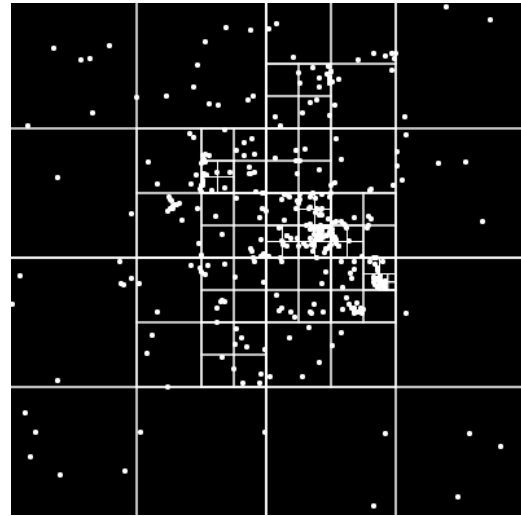


Figura 11: Visualización después de insertar puntos con el mouse.

Nuestro *QuadTree* inicia con 50 datos que están ubicados dentro del límite definido con posiciones aleatorias (Figura 10), en este caso nuestra capacidad máxima es de 8 puntos en cada cuadrante; todo esto está implementado en nuestra función *setup()*. Sobre este estado inicial del *QuadTree*, podemos insertar puntos con el mouse, donde se puede notar que al hacerlo se produce más y más divisiones, tal y como se ve en la Figura 11; claramente se observa que existen más divisiones en las zonas donde existe una gran concentración de puntos, y esto es porque en nuestra clase *QuadTree* hemos definido una función *insert()* en la que tenemos como condicion lo siguiente; si la cantidad de puntos dentro del cuadrante es menor a la capacidad máxima, se ubica el punto en el cuadrante y se almacena en el arreglo de puntos; caso contrario, si en el cuadrante, la cantidad de puntos llega a pasar de la capacidad máxima, el cuadrante se divide en 4 nodos hijos, y nuestro valor booleano *divided* se convierte en un *true*.