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.

```
<!DOCTYPE html>
<html>
<head>
    <title>QuadTree</title>
    <script src = "p5.min.js"></script>
    <script src = "quadtree.js"></script>
    <script src = "sketch.js"></script>
</head>
<body>
</body>
</html>
```

Respuesta:

```
<!DOCTYPE html>
<html>
<head>
    <title>QuadTree</title>
    <script src = "p5/p5.min.js"></script>
    <script src = "quadtree.js"></script>
    <script src = "sketch.js"></script>
</head>
<body>
</body>
</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).

```
class Point {
    constructor (x, y, userData ){
        this.x = x;
        this.y = y;
        this.userData = userData ;
    }
}
class Rectangle {
    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
           Rectangle
19      intersects ( range ){
20      }
21  }
```

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
           Rectangle
26      intersects(range) {
27          return !(range.x - range.w > this.x + this.w ||
28          range.x + range.w < this.x - this.w ||
29          range.y - range.h > this.y + this.h ||
30          range.y + range.h < this.y - this.h);
31      }
32  }
```

Código en GitHub : *quadtree.js*

Pregunta #03

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

```
class QuadTree{
    constructor(boundary, n){
        this.boundary = boundary ; // Rectangle
        this.capacity = n; // capacidad maxima de cada cuadrante
        this.points = []; // vector , almacena los puntos a
            almacenar
        this.divided = false ;
    }

    // divide el quadtree en 4 quadtrees
    subdivide(){
        // Algoritmo
        // 1: Crear 4 hijos : qt_northeast , qt_northwest ,
            qt_southeast , qt_southwest
        // 2: Asignar los QuadTree creados a cada hijo
        // this . northeast = qt_northeast ;
        // this . northwest = qt_northwest ;
        // this . southeast = qt_southeast ;
        // this . southwest = qt_southwest ;
        // 3. - Hacer : this . divided <- true
    }
    insert(point){
        // Algoritmo
        // 1: Si el punto no esta en los limites ( boundary )
            del quadtree Return
        // 2: Si ( this . points . length ) < ( this . capacity
            ),
        // 2.1 Insertamos en el vector this . points
        // Sino
        // 2.2 Dividimos si aun no ha sido dividido
        // 2.3 Insertamos recursivamente en los 4 hijos .
        // this . northeast . insert ( point );
        // this . northwest . insert ( point );
        // this . southeast . insert ( point );
        // this . southwest . insert ( point );
    }
    show () {
        stroke(255) ;
        strokeWeight(1) ;
        noFill() ;
        rectMode(CENTER);
        rect(this.boundary.x, this.boundary.y, this.boundary.w*2
            , this.boundary.h*2) ;
        if ( this.divided ){
            this.northeast.show() ;
            this.northwest.show() ;
            this.southeast.show() ;
            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
                   a ${typeof n}`);
13         }
14
15         if (n < 1) {
16             throw RangeError('capacity must be greater than 0');
17         }
18
19         this.boundary = boundary; // Rectangule
20         this.capacity = n; // Capacidad maxima de cada cuadrante
21         this.points = []; // Vector, almacena los punto a
               almacenar
22         this.divided = false;
23     }
24
25     // Divide el quadtree en 4 quadtrees
26     subdivide() {
27         let x = this.boundary.x;
28         let y = this.boundary.y;
29         let w = this.boundary.w / 2;
30         let h = this.boundary.h / 2;
31
32         // 1:  Creamos 4 hijos : qt_northeast  , qt_northwest,
               qt_southeast  , qt_southwest
33         let qt_northeast = new Rectangle(x + w, y - h, w, h);
34         let qt_northwest = new Rectangle(x - w, y - h, w, h);
35         let qt_southeast = new Rectangle(x + w, y + h, w, h);
36         let qt_southwest = new Rectangle(x - w, y + h, w, h);
37
38         // 2:  Asignamos  los  QuadTree  creados a cada  hijo
```

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

Editemos 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()
               * 400) ;
16         qt.insert(p);
17     }
18     background(0) ;
19     qt.show() ;
20 }
```

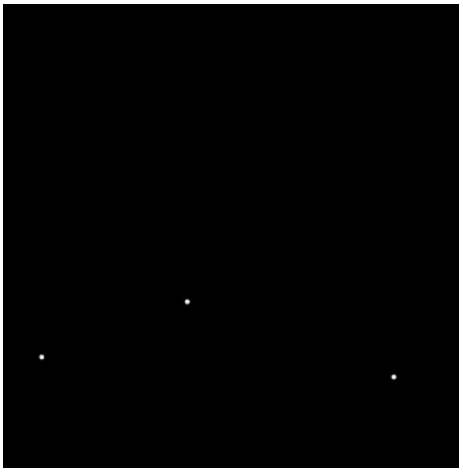

Figura 1: Visualización del QuadTree con 3 datos.

Respuesta:

```
let qt;
let count = 0;

function setup() {
    createCanvas(400 ,400) ;

    // centre point and half of width and height
    let boundary = new Rectangle(200 ,200 ,200 ,200);

    // each leave just could have 4 elements
    qt = new QuadTree( boundary , 4);

    console.log(qt);

    for (let i =0; i < 3; i ++) {
        let p = new Point( Math.random() * 400 , Math.random() *
            400) ;
        qt.insert(p);
    }
    background(0) ;
    qt.show() ;
}
```

Código en GitHub : *sketch.js*

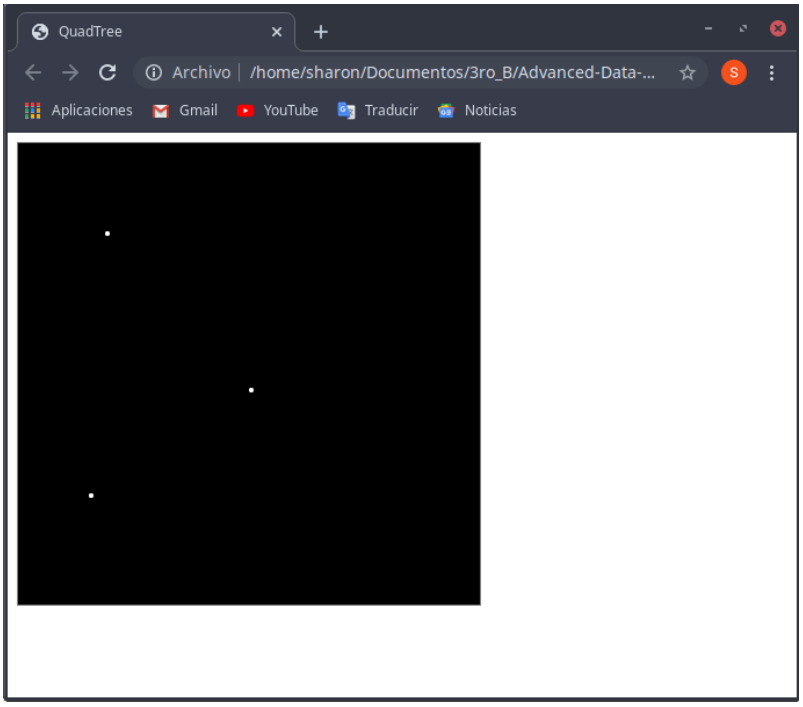

Figura 2: Programa en ejecución del QuadTree con 3 datos.

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*.

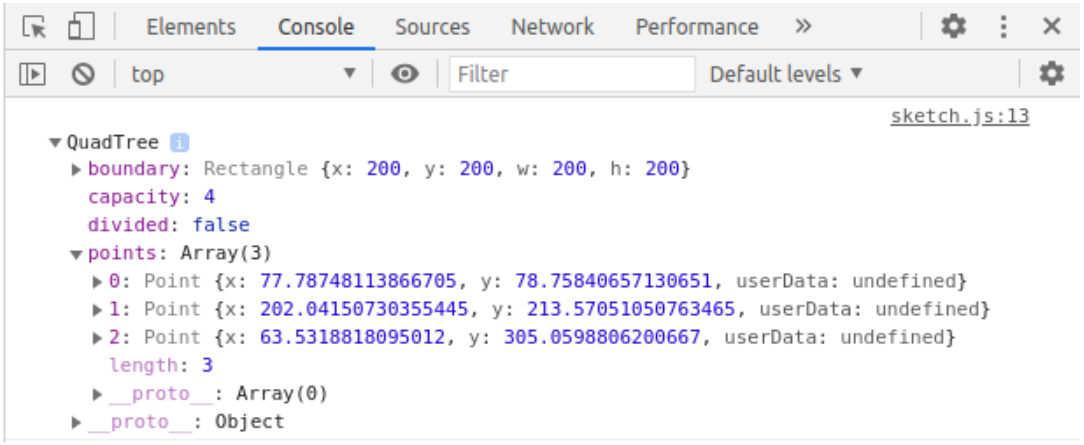

Figura 3: Vista a la Console de las opciones de desarrollador del navegador Web.

Pregunta #06

Inserte más puntos y muestre cómo varían sus resultados.

Respuesta:

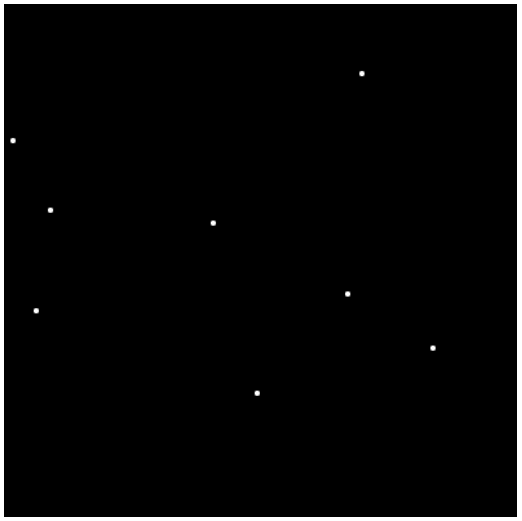

Figura 4: Visualización del QuadTree con 8 datos.

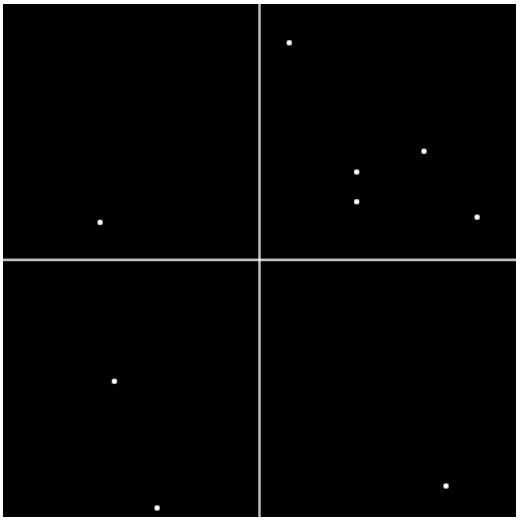

Figura 5: Visualización del QuadTree con 9 datos.

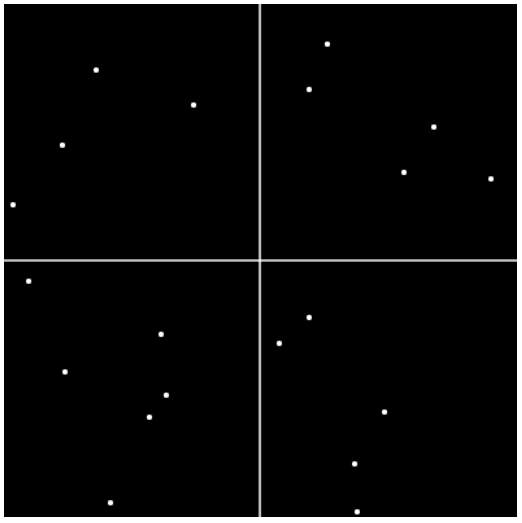

Figura 6: Visualización del QuadTree con 20 datos.

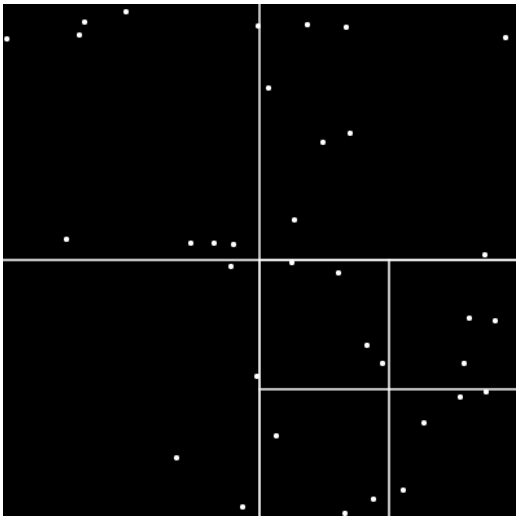

Figura 7: Visualización del QuadTree con 36 datos.

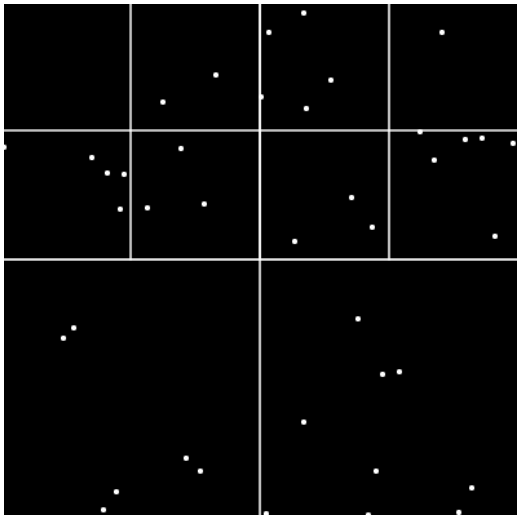

Figura 8: Visualización del QuadTree con 40 datos.

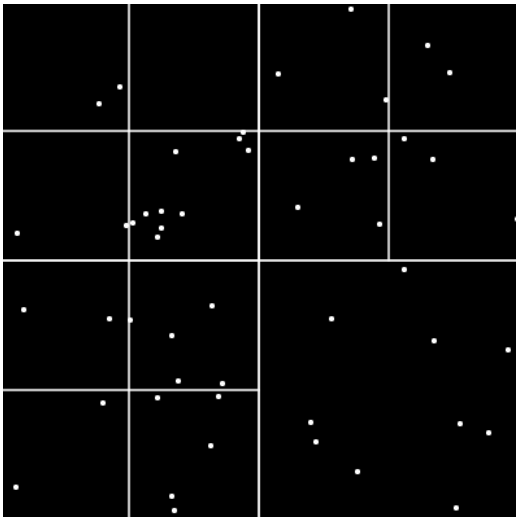

Figura 9: Visualización del QuadTree con 50 datos.

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.

```
let qt;
let count = 0;

function setup(){
    createCanvas(400,400);
    let boundary = new Rectangle(200,200,200,200);
    qt = new QuadTree(boundary,4);
}

function draw(){
    background(0);
    if (mouseIsPressed){
        for (let i = 0; i < 1; i ++){
            let m = new Point (mouseX + random( -5 ,5) , mouseY
                + random( -5 ,5) );
            qt.insert(m)
        }
    }
    background(0);
    qt.show();
}
```

Respuesta:

```
let qt;
let count = 0;

function setup() {
    createCanvas(400 ,400) ;

    // centre point and half of width and height
    let boundary = new Rectangle(200 ,200 ,200 ,200);

    // each leave just could have 8 elements
    qt = new QuadTree( boundary , 8);

    console.log(qt);

    for (let i =0; i < 50; i ++) {
        let p = new Point( Math.random() * 400 , Math.random() *
            400) ;
        qt.insert(p);
    }
    background(0) ;
    qt.show() ;
}

function draw() {
    background(0);
    if ( mouseIsPressed ) {
        for (let i = 0; i < 1; i ++) {
```

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

Código en GitHub : *sketch.js*

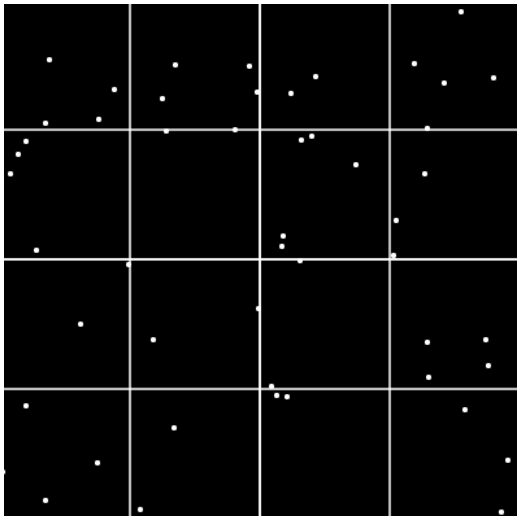

Figura 10: Visualización antes de insertar puntos con el mouse.

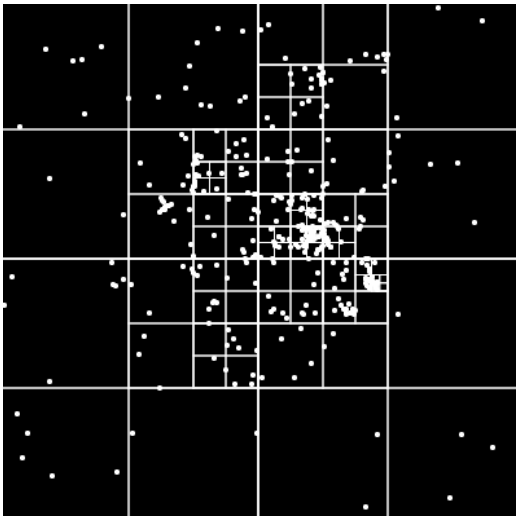

Figura 11: Visualización después de insertar puntos con el mouse.

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*.

Referencias

[1] Repositorio de la Práctica Nro. 02 en GitHub: https://github.com/sharon1160/Advanced-Data-Structure/tree/master/Practices/Practice%2002