PRACTICA 3 DE LABORATORIO

DOCENTE	CARRERA	CURSO
Vicente Machaca Arceda	Escuela Profesional de Ciencias de la Computación	Estructuras de Datos Avanzadas

PRÁCTICA N°	TEMA	DURACIÓN (HORAS)
03	Quadtree	2

1. OBJETIVOS

- Entender el funcionamiento de Quadtree
- Implementar una estructura Quadtree
- Implementar la función Query del QuadTree

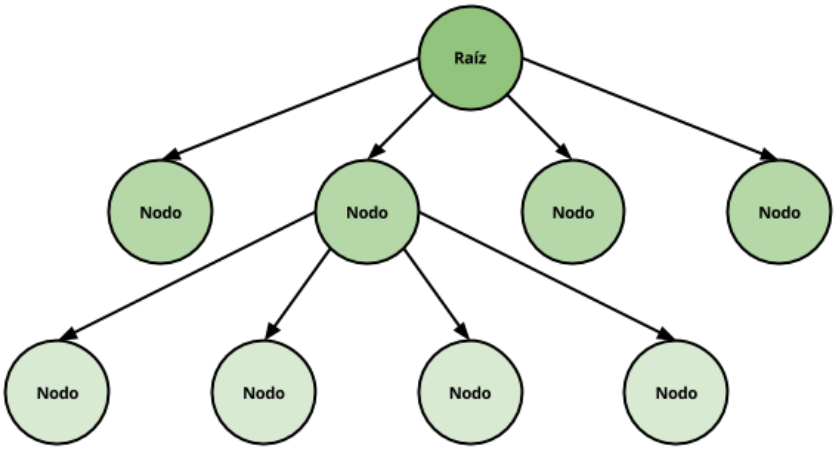
2. TEMAS A TRATAR

- Quadtree

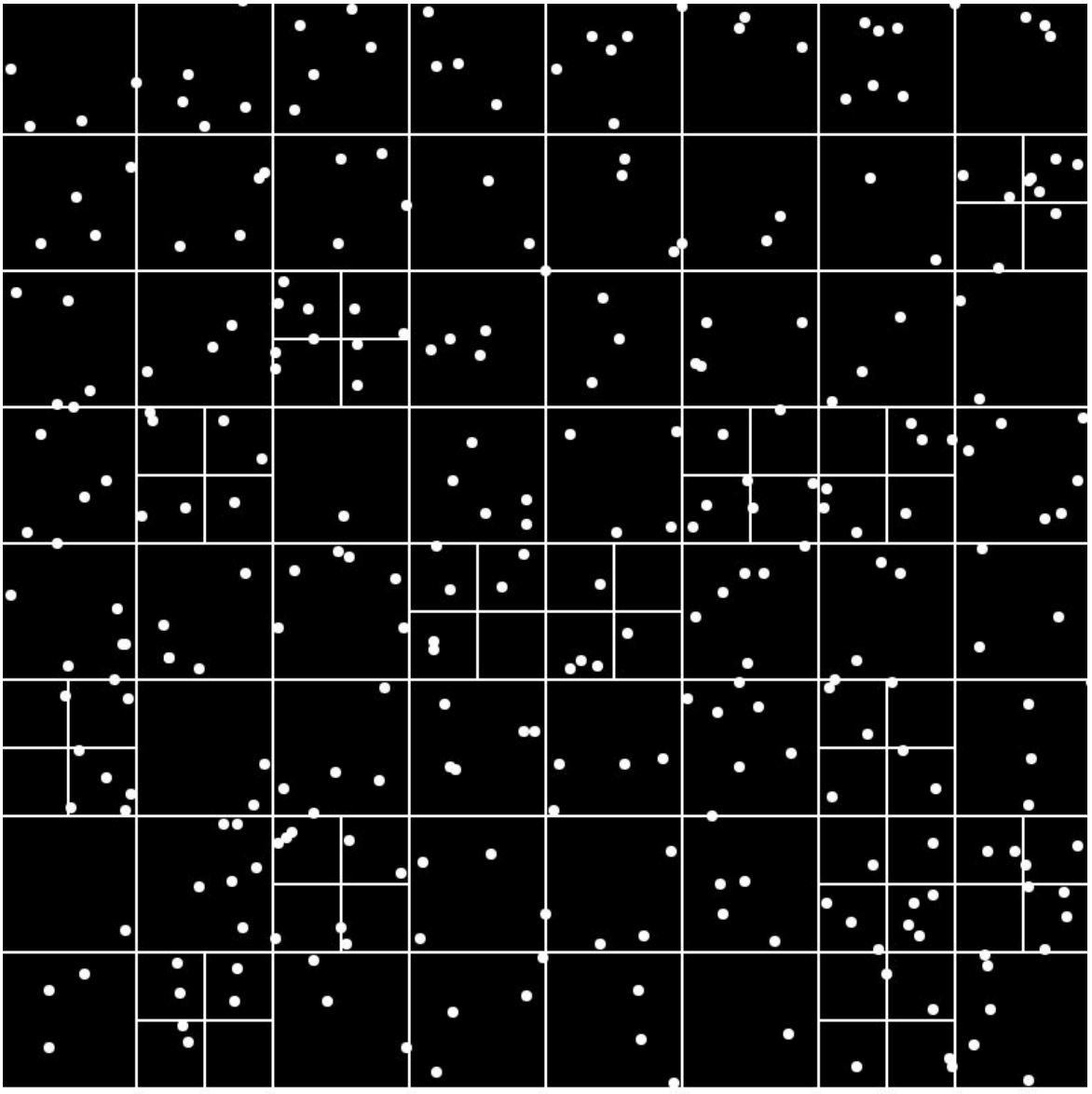
3. MARCO TEÓRICO

QUADTREE
(Extraido de: https://www.genbeta.com/desarrollo/teoria-de-colisiones-2d-quadtree)

Los Quadtrees son un tipo de estructura de datos en el que cada nodo tiene a su vez cuatro nodos hijos, sería algo similar a un árbol binario, pero en vez de dos ramas con cuatro ramas. El término **Quadtree**, o árbol cuaternario, se utiliza para describir clases de estructuras de datos jerárquicas cuya propiedad común es que están basados en el principio de descomposición recursiva del espacio. En un QuadTree de puntos, el centro de una subdivisión está siempre en un punto. Al insertar un nuevo elemento, el espacio queda dividido en cuatro cuadrantes. Al repetir el proceso, el cuadrante se divide de nuevo en cuatro cuadrantes, y así sucesivamente.



El Quadtree divide un espacio en cuadrantes y estos a su vez se dividen en más cuadrantes, de manera tal que se indexen cada elemento del espacio. Por ejemplo en la imagen de abajo, se visualiza el espacio, cada círculo representa un dato. La idea es que al tener todo el espacio dividido en cuadrantes resulta mucho más fácil hacer consultas, como la cantidad de puntos que están en un rango determinado, ver los puntos con colisionan, etc.



PSEUDOCODIGO DE QUADTREE
(Extraído de:https://en.wikipedia.org/wiki/Quadtree)

```
// Simple coordinate object to represent points and vectors
struct XY
{
 float x;
 float y;
```

```
 function __construct(float _x, float _y) {...}
}

// Axis-aligned bounding box with half dimension and center
struct AABB
{
 XY center;
 float halfDimension;

 function __construct(XY center, float halfDimension) {...}
 function containsPoint(XY point) {...}
 function intersectsAABB(AABB other) {...}
}

class QuadTree
{
 // Arbitrary constant to indicate how many elements can be stored in this quadtree node
 constant int QT_NODE_CAPACITY = 4;

 // Axis-aligned bounding box stored as a center with half-dimensions
 // to represent the boundaries of this quad tree
 AABB boundary;

 // Points in this quadtree node
 Array of XY [size = QT_NODE_CAPACITY] points;

 // Children
 QuadTree* northWest;
 QuadTree* northEast;
 QuadTree* southWest;
 QuadTree* southEast;

 // Methods
 function __construct(AABB _boundary) {...}
 function insert(XY p) {...}
 function subdivide() {...} // create four children that fully divide this quad into four quads
of equal area
 function queryRange(AABB range) {...}
}


// Insert a point into the QuadTree
 function insert(XY p)
 {
   // Ignore objects that do not belong in this quad tree
   if (!boundary.containsPoint(p))
     return false; // object cannot be added

   // If there is space in this quadtree and if doesn't have subdivisions, add the object here
   if (points.size < QT_NODE_CAPACITY && northWest == null)
   {
     points.append(p);
```

```
    return true;
  }

  // Otherwise, subdivide and then add the point to whichever node will accept it
  if (northWest == null)
    subdivide();
  //We have to add the points/data contained into this quad array to the new quads if we want
that only
  //the last node holds the data

  if (northWest->insert(p)) return true;
  if (northEast->insert(p)) return true;
  if (southWest->insert(p)) return true;
  if (southEast->insert(p)) return true;

  // Otherwise, the point cannot be inserted for some unknown reason (this should never happen)
  return false;
}


// Find all points that appear within a range
function queryRange(AABB range)
{
  // Prepare an array of results
  Array of XY pointsInRange;

  // Automatically abort if the range does not intersect this quad
  if (!boundary.intersectsAABB(range))
    return pointsInRange; // empty list

  // Check objects at this quad level
  for (int p = 0; p < points.size; p++)
  {
    if (range.containsPoint(points[p]))
      pointsInRange.append(points[p]);
  }

  // Terminate here, if there are no children
  if (northWest == null)
    return pointsInRange;

  // Otherwise, add the points from the children
  pointsInRange.appendArray(northWest->queryRange(range));
  pointsInRange.appendArray(northEast->queryRange(range));
  pointsInRange.appendArray(southWest->queryRange(range));
  pointsInRange.appendArray(southEast->queryRange(range));

  return pointsInRange;
}
```

4. EJERCICIOS

En este ejercicio se continuará con la implementación del Quadtree. **Deberá desarrollar cada actividad y documentarla en un informe.**

A. Editemos el archivo quadtree.js y completemos la función **query**

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

   insert(point){
       if(!this.boundary.contains(point)){
           return;
       }
       if (this.points.length < this.capacity){
           this.points.push(point);
       }
       else{
           if(!this.divided){
               this.subdivide();
           }
           this.northeast.insert(point);
           this.northwest.insert(point);
           this.southeast.insert(point);
           this.southwest.insert(point);
       }
   }

    //divide nuestro quadtree en 4 quadtrees
   subdivide(){
       let x = this.boundary.x;
       let y = this.boundary.y;
       let w = this.boundary.w;
       let h = this.boundary.h;

       let ne = new Rectangle(x + w/2, y - h/2, w/2, h/2);
       this.northeast = new QuadTree(ne, this.capacity);
       let nw = new Rectangle(x - w/2, y - h/2, w/2, h/2);
       this.northwest = new QuadTree(nw, this.capacity);
       let se = new Rectangle(x + w/2, y + h/2, w/2, h/2);
       this.southeast = new QuadTree(se, this.capacity);
       let sw = new Rectangle(x - w/2, y + h/2, w/2, h/2);
       this.southwest = new QuadTree(sw, this.capacity);

       this.divided = true;
   }
```

```
    //Retorna los puntos que están en un rango.
    query(range, found){
       //algoritmo
       //1.- Si el rango (range) no se intersecta con los límites del quadtree retornamos.
       //2.- Hacemos un ciclo por cada punto de este queadtree (this.points) y verificamos si
están dentro del rango, si es así los insertamos en el vector found.
       //3.  Si el quadtree ha sido dividido, llamamos recursivamente a query en cada hijo.
   }
}
```

B. Editemos el archivo sketch.js con el siguiente código. Muestre sus resultados y comente.

```
let qt;
let count = 0;

function setup(){
   createCanvas(400,400);
   let boundary = new Rectangle(200,200,200,200); //centr point and half of width and height
   qt = new QuadTree(boundary, 4); //each section just could have 4 elements
   for (let i=0; i < 300; i++){
       let p = new Point(Math.random() * 400, Math.random() * 400);
       qt.insert(p);
   }


   background(0);
   qt.show();

   stroke(0,255,0);
   rectMode(CENTER);
   let range = new Rectangle(random(200),random(200),random(50),random(50))
   rect(range.x, range.y, range.w*2,range.h*2);
   let points = [];
   qt.query(range, points);
   //console.log(points);
   for (let p of points){
       strokeWeight(4);
       point(p.x, p.y);
   }
   console.log(count); //CANTIDAD DE VECES QUE CONSULTO

}
```

C. En este caso vamos a verificar cuantas veces se consulta un punto en la función query, para esto usaremos la variable global **count** creada en el paso anterior. Esta variable la incrementaremos en la función **query**. Muestre sus resultados y comente.

```
query(range, found){
       //algoritmo
       //1.- Si el rango (range) no se intersecta con los límites del quadtree retornamos.
       //2.- Hacemos un ciclo por cada punto de este quadtree (this.points) y verificamos si
están dentro del rango, si es así los insertamos en el vector found. (En este ciclo
incrementamos count)
       //3.  Si el quadtree ha sido dividido, llamamos recursivamente a query en cada hijo.
   }
```

D. Editemos el archivo sketch.js. En este caso haremos consultas con el mouse. Muestre sus resultados y comente.

```
let qt;
let count = 0;

function setup(){
   createCanvas(400,400);
   let boundary = new Rectangle(200,200,200,200); //centr point and half of width and height
   qt = new QuadTree(boundary, 4); //each section just could have 4 elements
   for (let i=0; i < 300; i++){
       let p = new Point(Math.random() * 400, Math.random() * 400);
       qt.insert(p);
   }
}
function draw(){
   background(0);
   qt.show();

   stroke(0,255,0);
   rectMode(CENTER);
   let range = new Rectangle(mouseX,mouseY,50,50)
   rect(range.x, range.y, range.w*2,range.h*2);
   let points = [];
   qt.query(range, points);
   //console.log(points);
   for (let p of points){
       strokeWeight(4);
       point(p.x, p.y);
   }
}
```

REFERENCIAS

Weiss, M. A. (2012). *Data structures & algorithm analysis in C++*. Pearson Education.