COLISIONES ENTRE ABEJAS ROBÓTICAS

ESTE PROYECTO CONSISTE EN LOGRAR IDENTIFICAR LAS ABEJAS ROBOTICAS MAS CERCANAS PARA EVITAR COLISIONES ENTRE ELLAS.

Mariajose Franco Orozco
Universidad Eafit
Colombia
mfrancoo@eafit.edu.co

Susana Álvarez
Universidad Eafit
Colombia
salvarezz1@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

RESUMEN

Las abejas son insectos con un papel importante en el proceso de reproducción de las plantas y los cultivos. Los pesticidas y la deforestación han causado graves problemas para las abejas ya que ha reducido la población de estas, afectando a su vez a los cultivos y a los agricultores

Palabras Clave

Estructura de datos, Complejidad, Prevención de colisiones, Tiempo de ejecución, Algoritmos.

Palabras Clave de la clasificación de la ACM

Theory of computation → Design and analysis of algorithms → Data structures design and analysis → Sorting and searching

1. INTRODUCCIÓN

El objetivo de todo organismo vivo es crear descendencia. Las plantas generan polen con el objetivo de ser transmitido a otras plantas para así generar esa descendencia. Existen algunos insectos que ayudan a transmitir el polen entre las plantas como lo son las abejas. Es decir que en todos los cultivos es necesario que exista el proceso de la polinización.

Lastimosamente, el mundo ha sufrido cambios climáticos muy drásticos, lo que causa la deforestación de algunos terrenos, algo que también es consecuencia de ciertas acciones humanas. Muchos cultivos han sufrido a causa de esta deforestación y principalmente por los pesticidas que se les echa para prevenir alguna plaga; el problema con estos pesticidas es que afectan a estos insectos encargados de hacer la polinización entre las plantas, lo cual causa la extinción de una gran cantidad de abejas y a la vez de muchos cultivos.

2. PROBLEMA

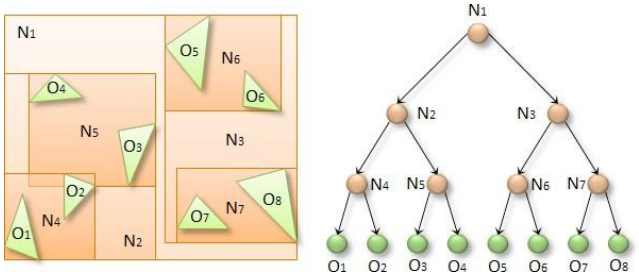
Es importante poder encontrar una solución a esta problemática ya que se espera conservar los cultivos, pero tampoco se quiere obtener ninguna plaga en estos; por esta razón se inventaron las abejas robóticas, las cuales consisten en unos drones encargados de hacer el trabajo de la polinización, pero hay un problema con estos drones ya que puede pasar que entre ellos haya colisiones. Con este proyecto queremos intentar lograr que no ocurra esto entre estas abejas robóticas, es decir, lograr identificar las más cercanas para no dirigir dos drones al mismo lugar.

3. TRABAJOS RELACIONADOS

Estos son algunos problemas de estructuras de datos similares:

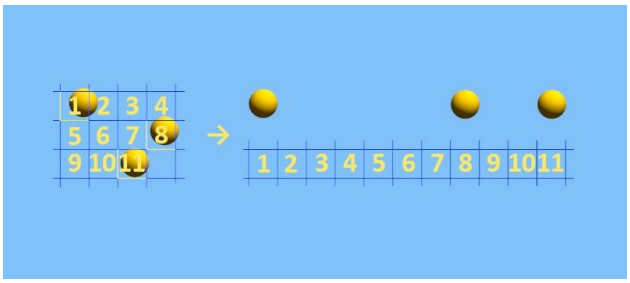
3.1 Dynamic AABB tree

Este consiste en una búsqueda binaria para particiones espaciales. Esta estructura de datos consiste en arreglo de nodos, los cuales se representan en dos formas: las ramas y las hojas. La idea de estos árboles es permitir que los datos se almacenen en los nodos de las hojas; los nodos de las ramas contienen una sola hoja y de ella salen otras dos, etc. Los arboles AABB se basan en 3 principales funciones que son insertar, remover y actualizar. La primera función que es la de insertar consiste en crear un AABB grueso para unir datos con un nuevo nodo de hoja, luego se realiza un recorrido por el árbol para encontrar con que hermano emparejarse y crear otra ramificación. La función de remover solo se puede realizar con los nodos hojas ya que cada rama debe tener 2 hijos válidos y solo las hojas contienen los datos del usuario. Luego, la función de actualizar se encarga de verificar que el AABB actual y ajustada, todavía este contenida en el árbol.



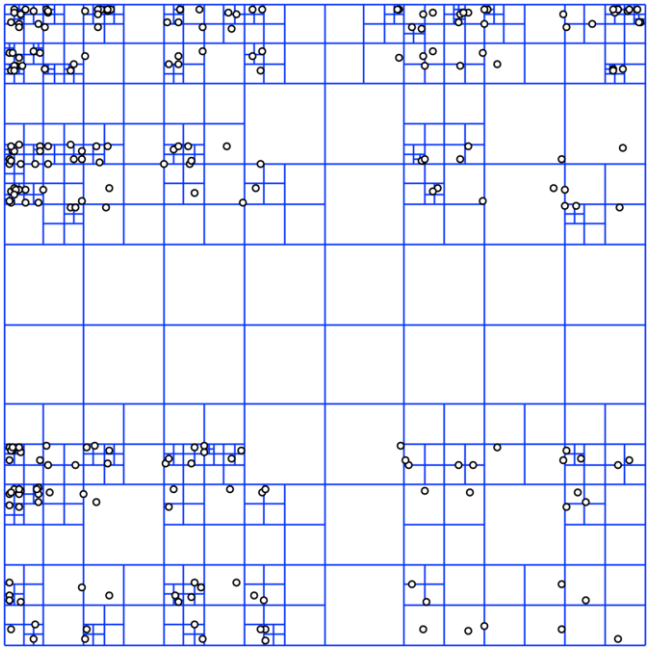
3.2 Spatial Hashing

Esta estructura de datos consiste en una extensión de las tablas hash en la que se puede trabajar de forma bidimensional o tridimensional. Las tablas de hash son estructuras de datos que asocian claves con valores. La primera operación que se realiza es la búsqueda del dato con el que se desea trabajar o se desea encontrar, este dato es guardado en forma de clave y luego por medio las funciones hash se realiza una transformación de esta clave a una posición, ubicando así el valor deseado. Generalmente, las tablas de hash trabajan con claves de una dimensión (strings), pero en un hash espacial se puede trabajar con claves de 2 y 3 dimensiones. Las tablas hash también son conocidas como celdas donde en cada una de estas se ha ubicado un objeto. Para evidenciar si puede haber o no alguna coalición se puede observar el desplazamiento de los objetos en las celdas.



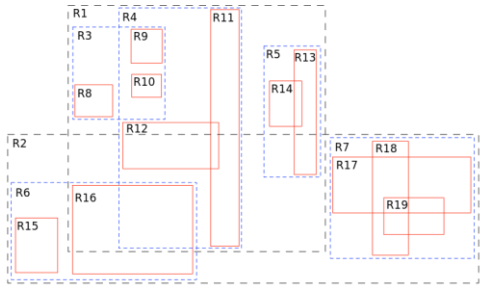
3.3 Quadtree

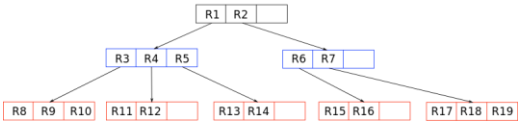
Un Quadtree, también llamado árbol cuaternario, es una estructura de datos que se basa en la descomposición recursiva del espacio. En esta estructura, cada nodo interno tiene cuatro hijos que son usados normalmente para dividir un espacio bidimensional. Los Quadtrees también permiten representar datos no solo en el espacio bidimensional si no también en los espacios tridimensionales o hasta con n dimensiones. En el espacio bidimensional, cada nodo representa un cuadrante, ya cuando se trabaja en un espacio de n dimensiones, cada cuadrante se van dividiendo en más y más cuadrantes. El desarrollo de este tipo de estructura de datos fue creado ya que se necesitaba una estructura que permitiera guardar datos, representar puntos, líneas, curvas, superficies, y todo tipo de objetos en diferentes dimensiones. Estos árboles también son utilizados para la compresión de imágenes en donde cada nodo representa el color promedio de cada uno de sus hijos. También se utiliza para detección de colisiones en dos dimensiones, solución de campos multidimensionales, encontrar la distancia más corta entre 2 o más puntos, y muchas otras cosas más.



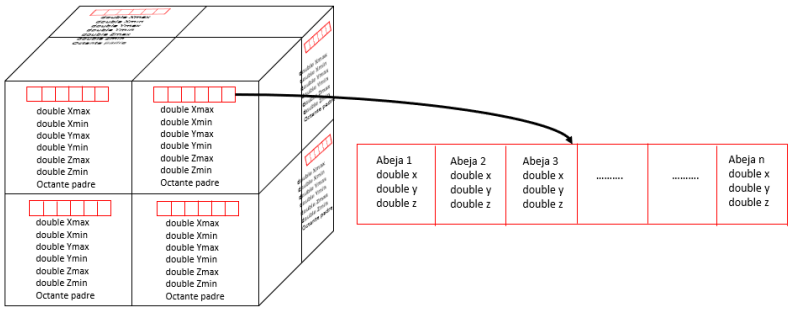
3.4 R-tree

Es una estructura de datos de árbol utilizada para tratar con la información espacial. Esta estructura fue creada por Antonin Guttman en 1984. El R- tree se utiliza normalmente para buscar datos utilizando una ubicación. La idea clave de la estructura de datos es agrupar objetos cercanos de la ubicación y representarlos con su rectángulo de límite mínimo en el nivel siguiente del árbol. Como todos los objetos se encuentran dentro de este rectángulo con limite, esto permite que la consulta no se intercepte con los objetos que no están contenidos en este rectángulo. Cada hoja representa un solo objeto. Un uso común de esta estructura de datos en la vida cotidiana es buscar restaurantes abiertos en un radio de 1 Km de la ubicación actual de una persona, buscar el mercado más cercano a la ubicación actual, entre otros.





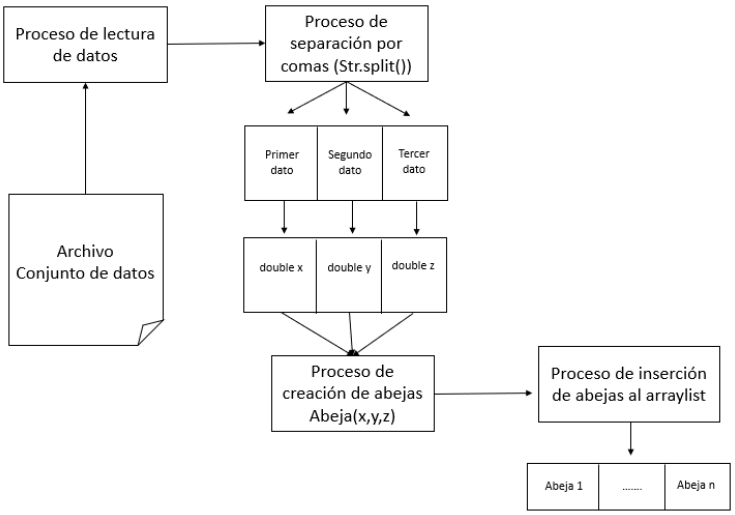
4. Diseño de la estructura de datos: Octree



Grafica 1: Octree: Cada octante contiene un arraylist de abejas, un octante padre y las coordenadas máximas y mínimas de las abejas. El arraylist de abejas en cada casilla contiene una abeja con coordenadas x, y, z.

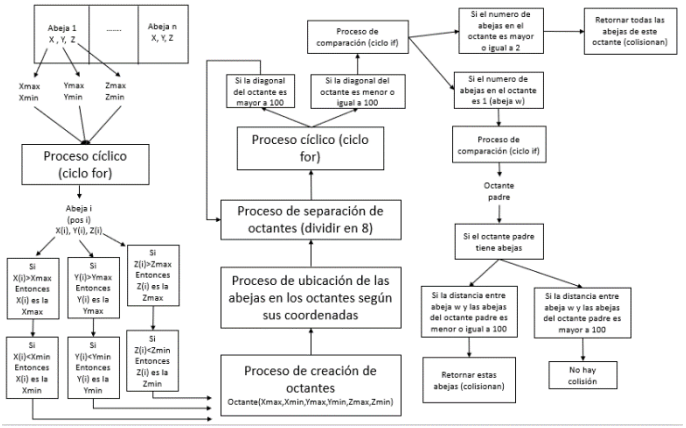
4.1 Operaciones de la estructura de datos

1. **Leer y guardar datos**



Grafica 2: Imagen del proceso de leer y guardar datos

2. **Detectar colisiones**



Grafica 3: Imagen del proceso de detección de colisiones

4.2 Criterios de diseño de la estructura de datos

Decidimos implementar esta estructura de datos ya que nos parece que no es difícil de interpretar y entender; nos parece que es una estructura de datos muy útil para lo que queremos lograr, funciona bien y la complejidad de sus métodos son O(n) y O(n + log(n)), lo que permite que sea muy rápido en su ejecución, esto también lo podemos evidenciar en las tablas de tiempos en este documento; allí podemos observar que los tiempos promedio de ejecución de cada método con los distintos conjuntos de datos no sobrepasan los 4 segundos, lo que significa que la estructura de datos esta funcionando bien y a gran velocidad, que es lo que buscábamos con esta.

4.3 Análisis de Complejidad

Método	Complejidad
leerYGuardarDatos()	O(n)
detectarColisiones()	O(n + log(n))

Tabla 1: Complejidad de los métodos implementados

4.4 Tiempos de Ejecución

- **Operación 1 =** leerYGuardarDatos()
- **Operación 2 =** detectarColisiones()

Operaciones	Conj de datos 1 (4 abejas)	Conj de datos 2 (15 abejas)	Conj de datos 3 (150 abejas)	Conj de datos 4 (1.500 abejas)	Conj de datos 5 (15.000 abejas)	Conj de datos 6 (150.000 abejas)	Conj de datos 7 (1'500.000 abejas)
1	0 ms	0 ms	0 ms	1 ms	13 ms	146 ms	292 ms
2	0 ms	0 ms	1 ms	15 ms	209 ms	3047 ms	656 ms

Tabla 2: Tiempos de operación con distintos conjuntos de datos

Operaciones	Peor tiempo	Mejor tiempo	Tiempo promedio
1	2 ms	0 ms	0 ms
2	3 ms	0 ms	0 ms

Tabla 3: Tiempos con un conjunto de datos de 4 abejas

Operaciones	Peor tiempo	Mejor tiempo	Tiempo promedio
1	3 ms	0 ms	0 ms
2	2 ms	0 ms	0 ms

Tabla 4: Tiempos con un conjunto de datos de 15 abejas

Operaciones	Peor tiempo	Mejor tiempo	Tiempo promedio
1	4 ms	0 ms	0 ms
2	4 ms	0 ms	1 ms

Tabla 5: Tiempos con un conjunto de datos de 150 abejas

Operaciones	Peor tiempo	Mejor tiempo	Tiempo promedio
1	12 ms	1 ms	1 ms
2	54 ms	2 ms	15 ms

Tabla 6: Tiempos con un conjunto de datos de 1.500 abejas

Operaciones	Peor tiempo	Mejor tiempo	Tiempo promedio
1	49 ms	10 ms	13 ms
2	548 ms	18 ms	209 ms

Tabla 7: Tiempos con un conjunto de datos de 15.000 abejas

Operaciones	Peor tiempo	Mejor tiempo	Tiempo promedio
1	1764 ms	99 ms	146 ms
2	14432 ms	116 ms	3057 ms

Tabla 8: Tiempos con un conjunto de datos de 150.000 abejas

Operaciones	Peor tiempo	Mejor tiempo	Tiempo promedio
1	1241 ms	166 ms	292 ms
2	1223 ms	121 ms	656 ms

Tabla 9: Tiempos con un conjunto de datos de 1’500.000 abejas

4.5 Memoria

- **Operación 1 =** leerYGuardarDatos()
- **Operación 2 =** detectarColisiones()

Operaciones	Conj de datos 1 (4 abejas)	Conj de datos 2 (15 abejas)	Conj de datos 3 (150 abejas)	Conj de datos 4 (1.500 abejas)	Conj de datos 5 (15.000 abejas)	Conj de datos 6 (150.000 abejas)	Conj de datos 7 (1’500.000 abejas)
1	1499170 bytes	2166097 bytes	2679581 bytes	8142645 bytes	61921156 bytes	516502937 bytes	3578425 77 bytes
2	850148 bytes	2280726 bytes	3635731 bytes	1713734 4 bytes	69564628 bytes	1226903533 bytes	5105599 61 bytes

Tabla 10: Consumo de memoria con distintos conjuntos de datos

Operaciones	Peor memoria	Mejor memoria	Memoria promedio
1	2202504 bytes	1448872 bytes	1499170 bytes
2	2186616 bytes	813432 bytes	850148 bytes

Tabla 11: Consumo de memoria con un conjunto de datos de 4 abejas

Operaciones	Peor memoria	Mejor memoria	Memoria promedio
1	2226648 bytes	1510368 bytes	2166097 bytes
2	2320648 bytes	1593744 bytes	2280726 bytes

Tabla 12: Consumo de memoria con un conjunto de datos de 15 abejas

Operaciones	Peor memoria	Mejor memoria	Memoria promedio
1	3031040 bytes	1681240 bytes	2679581 bytes
2	4429616 bytes	3077672 bytes	3635731 bytes

Tabla 13: Consumo de memoria con un conjunto de datos de 150 abejas

Operaciones	Peor memoria	Mejor memoria	Memoria promedio
1	11375736 bytes	4129352 bytes	8142645 bytes
2	23445808 bytes	10518184 bytes	17137344 bytes

Tabla 14: Consumo de memoria con un conjunto de datos de 1.500 abejas

Operaciones	Peor memoria	Mejor memoria	Memoria promedio
1	94955520 bytes	28023232 bytes	61921156 bytes
2	139695136 bytes	2563712 bytes	69564628 bytes

Tabla 15: Consumo de memoria con un conjunto de datos de 15.000 abejas

Operaciones	Peor memoria	Mejor memoria	Memoria promedio
1	862873888 bytes	129081136 bytes	516502937 bytes
2	1662167736 bytes	765838880 bytes	1226903533 bytes

Tabla 16: Consumo de memoria con un conjunto de datos de 150.000 abejas

Operaciones	Peor memoria	Mejor memoria	Memoria promedio
1	436768432 bytes	198681304 bytes	357842577 bytes
2	666177208 bytes	348513384 bytes	510559961 bytes

Tabla 17: Consumo de memoria con un conjunto de datos de 1’500.000 abejas

4.6 Análisis de los resultados obtenidos

Según lo observado en la ejecución de la estructura de datos implementada pudimos notar que el tiempo promedio que tardan los métodos en realizar sus operaciones no son mayores a 1 segundo excepto el de el método detectarColisiones() con 150.000 abejas, el cual tuvo un tiempo promedio de 3 segundos, que también es un tiempo muy rápido de ejecución. Observando los peores tiempos de cada método, solo hubo un método que superó los 2 segundos y fue el mencionado anteriormente, este obtuvo un tiempo de 14 segundos al realizarse con 150.000 abejas; aunque no es un tiempo muy largo, si es mucho mas alto que el tiempo obtenido con los otros conjuntos de datos. En cuanto a la memoria, pudimos observar que el consumo de esta no es tan bajo como sería lo ideal; el mayor consumo de memoria fue directamente proporcional a la cantidad de abejas, ya que a mayor cantidad de datos, mas memoria es requerida.

REFERENCIAS

Myopic Rhino. 2009. Spatial Hashing. (October 2009). RetrievedFebruary 16, 2019 from https://www.gamedev.net/articles/programming/general-and-gameplay-programming/spatial-hashing-r2697

Anon. 2019. Tabla hash. (January 2019). Retrieved February 15, 2019 from https://es.wikipedia.org/wiki/Tabla_hash

Randy Gaul. 2013. Dynamic AABB Tree. (October 2013). Retrieved February 17, 2019 from https://www.randygaul.net/2013/08/06/dynamic-aabb-tree/

Anon. 2018. Quadtree. (October 2018). Retrieved February 16, 2019 from https://es.wikipedia.org/wiki/Quadtree

Anon. 2019. R-tree. (February 2019). Retrieved February 15, 2019 from https://en.wikipedia.org/wiki/R-tree