ALGORITMO PARA PREVENIR COLICIONES ENTRE ABEJAS ROBOTICAS

Juan Camilo Guerrero Alarcón
Universidad Eafit
Colombia
jcguerrera@eafit.edu.co

Santiago Pulgarin Vasquez
Universidad Eafit
Colombia
spulgarinv@eafit.edu.co

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

RESUMEN

El principal problema radica en que no se tiene un algoritmo implementado para detectar las abejas robóticas y que estas no se colisionen, es demasiado importante para que no disminuya la cantidad de abejas, saber dónde eran y a que objetivo tienen que llegar sin ningún problema, por el momento existen problemas similares como lo son Quadtree, AABB Tree entre otros.

Palabras clave
Estructura de datos, Colisión, Polinizacion, Industria, Flores, Latitud, Longitud, Memoria, Coordenadas geodésicas, Tecnologias.

Palabras clave de la clasificación de la ACM

Design and analysis of algorithms → Graph algorithms analysis → Shortest paths

1. INTRODUCCIÓN

Gran parte de cultivos en todo el mundo necesitan de un proceso llamado polinización el cual como es bien sabido es realizado por abejas y otros insectos. Por algunos fenómenos climáticos las abejas han reducido en gran cantidad lo cual está siendo un serio problema para los agricultores, lo que nos lleva a pensar si se podrían implementar una especie de RoboBees los cuales podrían ayudar con el problema e implementarlos en un futuro no muy lejano y a su vez mejorar la calidad de los cultivos.

2. PROBLEMA

La implementación de estas abejas resulta ser una buena manera para este caso, pero con dicha implementación se necesitará una forma y algoritmo eficiente para poder prevenir choques o colisiones entre ellas para que se desarrolle el proceso en dichos cultivos con eficiencia.

3. TRABAJOS RELACIONADOS

3.1 Spatial Hashing

Consiste en la falta de forma simple de reducir la cantidad de objetos probados durante la detección de colisiones, ya que los métodos tradicionales que se venían implementando no cumplen lo requerido y requieren demasiado tiempo.

Una posible solución es pensar en un hash espacial es una extensión de 2 o 3 dimensiones de la tabla hash, La idea básica de una tabla hash es que tomes un dato (la 'clave'), lo ejecutes a través de alguna función (la 'función hash') para producir un nuevo valor (el 'hash'), y luego usar el hash como un índice en un conjunto de ranuras ('cubos').

Para almacenar un objeto en una tabla hash, ejecuta la clave a través de la función hash y almacena el objeto en el depósito al que hace referencia el hash. Para encontrar un objeto, ejecuta la tecla a través de la función hash y busca en el depósito al que hace referencia el hash. [1]

3.2 AABB Tree Collision Detection

Radica básicamente en que no se tiene la maneja más optima de encontrar o detectar colisiones en el mundo a todo lo que se mueva y los métodos actuales no han dado los mejores resultados para que esto empiece a tomar cartas en el asunto, y en este problema se considera como un algoritmo implementado en un videojuego.

La solución que se ha revisado trata de Los AABB, son más simples de lo que parecen: son esencialmente cajas cuyos ejes (así x, y para 2d y x, y, z para 3d) se alinean / corren en la misma dirección. La parte delimitadora del nombre se debe a que, cuando se usan para la detección de colisiones o como parte de un árbol, comúnmente contienen o unen otras casillas. [2]

3.3 Quadtree

Un quadtree es una estructura de datos de árbol en la que cada nodo interno tiene exactamente cuatro hijos. Los Quadtrees son el análogo bidimensional de octrees y se usan con mayor frecuencia para dividir un espacio bidimensional subdividiéndolo recursivamente en cuatro cuadrantes o regiones. Los datos asociados con una célula de la hoja varían según la aplicación, pero la célula de la

hoja representa una "unidad de información espacial interesante".

Las regiones subdivididas pueden ser cuadradas o rectangulares, o pueden tener formas arbitrarias. Esta estructura de datos fue nombrada quadtree por Raphael Finkel y JL Bentley en 1974. Una partición similar también se conoce como Q-tree. [3]

3.4 Dynamic AABB Tree

Un árbol dinámico de AABB es un árbol de búsqueda binario para la partición espacial. Este árbol cuenta con unas propiedades para implementar y que lo hace más optimo:

- Insertar
- Retirar
- Actualizar

El nodo del árbol AABB puede construirse cuidadosamente

para ocupar un espacio mínimo, ya que los nodos están siempre en uno de dos estados: ramas y hojas. Como los nodos se almacenan en una matriz, esto permite que los nodos sean referenciados por índice integral en lugar de puntero. Esto permite que la matriz interna crezca o se reduzca según sea necesario sin temor a dejar punteros colgantes en cualquier lugar.

La idea del árbol es permitir que los datos del usuario se almacenen solo dentro de los nodos hoja. Todos los nodos de sucursales contienen solo una AABB que envuelve a ambos de sus hijos. [4]

4. Array

Gráfica 1: ArrayList de abejas, una abeja es una clase que contiene longitud, latitud y altura.

4.1 Operaciones de la estructura de datos

Gráfica 2: Imagen de una operación Add de un ArrayList.

4.2 Criterios de diseño de la estructura de datos

La razón por la cual escogimos la estructura de datos Arraylist es debido a que es una de las más usadas, más conocidas y fácil de implementar a tipos de problemas como este, además de su excelente implementación que facilita el análisis de los datos, aunque este tipo de estructura no es muy eficiente y nos genera un déficit en memoria y en tiempo de procesamiento lo que genera poca efectividad para grandes cantidades de datos.

4.3 Análisis de Complejidad

Metodo	Complejidad
distancia()	O(n)
leerArchivo	O(n)
detectarColisiones()	O(n^2)
guardarArchivo()	O(n)
main()	O(1)

Tabla 1: Tabla para reportar la complejidad

4.4 Tiempos de Ejecución

	CONJUNTO DE DATOS 1 (10)	CONJUNTO DE DATOS 2 (100)	CONJUNTO DE DATOS 3 (1000)	CONJUNTO DE DATOS 4 (100000)	CONJUNTO DE DATOS 5 (1000000)
TIEMPO QUE TARDO	0 segundos	0,0016831683168 3168 segundos	0,013138613861 3861 segundos	0,32765346534 6535 segundos	19,54 segundos

Tabla 2: Tiempos de ejecución de las operaciones de la estructura de datos con diferentes conjuntos de datos

4.5 Memoria

	Conjunto de datos1 (10)	CONJUNTO DE DATOS 2 (100)	CONJUNTO DE DATOS 3 (1000)	CONJUNTO DE DATOS 4 (100000)	CONJUNTO DE DATOS 5 (1000000)
Consumo de memoria	9	30	80	150	540

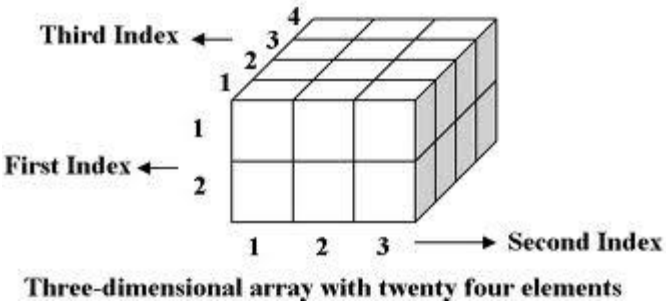
Tabla 3: Consumo de memoria de la estructura de datos con diferentes conjuntos de datos

4.6 Análisis de los resultados

El algoritmo que esta implementado en la solucion se basa en ArrayList que puede tomar en el peor de los casos 20,766 segundos para encontrar las abejas que se van a chocar en un arreglo de 100.000 abejas; pero en el mayor de los casos le toma 18,46 segundos y en el caso promedio 19,54 segundos, esto nos dice que si bien es un buen algoritmo que funciona para el promedio, hay casos en el cual puede tardar mucho tiempo. Lo anteriormente mencionado es debido principalmente a que su complejidad como algoritmo es de (n^2).

La estructura de datos ArrayList, es muy eficiente para acceder, por el contrario para buscar, insertar y eliminar es mas demorado, la salvedad aqui es que en el problema no es necesario eliminar elementos del arreglo lo que hace que su eficiencia mejore notablemente.

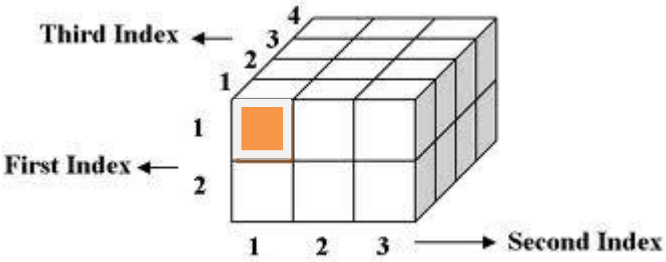
5. Matriz 3D



Three-dimensional array with twenty four elements

Gráfica 3: matriz 3D de abejas. Una abeja es una clase que contiene coordenada “x” , coordenada “y” y coordenada “z”

5.1 Operaciones de la estructura de datos



Three-dimensional array with twenty four elements

Gráfica 4: Imagen de una operación de insertado en una matriz 3D

5.2 Criterios de diseño de la estructura de datos

La razón por la cual escogimos la Matriz 3D es debido a que es una de las más usadas cuando se tratan problemas de ubicación espacial, más conocidas y muy acertada para tipos de problemas como este, además de su excelente implementación que facilita el análisis de los datos.

5.3 Análisis de la Complejidad

Clase	Complejidad
Detectar Colisiones	O(n log n)

Tabla 5: Tabla para reportar la complejidad

5.4 Tiempos de Ejecución

	CONJUNTO DE DATOS 1 (10)	CONJUNTO DE DATOS 2 (100)	CONJUNTO DE DATOS 3 (1000)	CONJUNTO DE DATOS 4 (100000)	CONJUNTO DE DATOS 5 (1000000)
TIEMPO QUE TARDO	0.0000 segundos	0,0020 segundos	0,0052 segundos	0,78 segundos	0.756 segundos

Tabla 6: Tiempos de ejecución de las operaciones de la estructura de datos con diferentes conjuntos de datos

5.5 Memoria
Establecido en MB.

	Conjunto de datos1 (10)	CONJUNTO DE DATOS 2 (100)	CONJUNTO DE DATOS 3 (1000)	CONJUNTO DE DATOS 4 (100000)	CONJUNTO DE DATOS 5 (1000000)
Consumo de memoria	113	123	123	155	500

Tabla 7: Consumo de memoria de la estructura de datos con diferentes conjuntos de datos

5.6 Análisis de los resultados

El algoritmo que esta implementado en la solucion se basa en una Matriz 3D que puede tomar en el peor de los casos 10 segundos para encontrar las abejas que se van a chocar en un arreglo de 1.000.000 abejas; pero en el mejor de los casos le toma 0.000001. esto nos dice que es un buen algoritmo que funciona para el promedio, debido a que su complejidad es O(n log n).

6. CONCLUSIONES

El escoger la adecuada estructura de datos para el problema que se quiere solucionar es de suma importancia, esto se ve evidenciado en los resultados obtenidos anteriormente.

Si bien ocurrieron varios problemas tratando de implementar una solución para abejas con tres coordenadas, pudimos implementar una solución para abejas con dos coordenadas, dicha solución logro una rebaja en los tiempos de ejecución con respecto a la anterior solución y rebajar la complejidad asintótica de O(n^2) a O(n).1

Uno de los mas grandes problemas que nos impidió generar una solución para abejas con coordenadas en tres dimensiones, fue el de hacer la función hash para saber en que cuadrante debía estar ubicada la abeja, ya que, cuando se trata con tres dimensiones, el algoritmo para encontrar la ubicación se volvía muy complejo.

6.1 Trabajos futuros

Para trabajos futuros nos gustaría poder implementar la estructura de datos, pero para abejas con tres coordenadas.

AGRADECIMIENTOS

Agradecemos generalmente a todas las personas que de manera indirecta y directa aportaron al desarrollo del proyecto.

REFERENCIAS

Referenciar las fuentes usando el formato para referencias de la ACM. Léase en http://bit.ly/2pZnE5g Vean un ejemplo:

1. Adobe Acrobat Reader 7, Asegúrense de justificar el texto. http://www.adobe.com/products/acrobat/.

2. Fischer, G. and Nakakoji, K. Amplifying designers' creativity with domainoriented design environments. in Dartnall, T. ed. Artificial Intelligence and Creativity: An Interdisciplinary Approach, Kluwer Academic Publishers, Dordrecht, 1994, 343-364.