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:

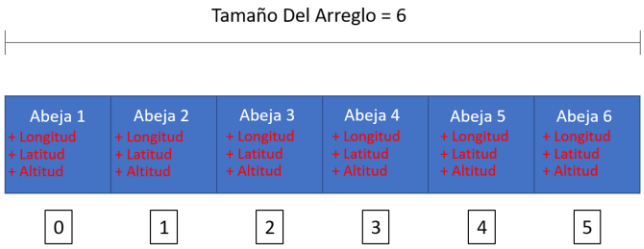
- Insertar
- Retirar
- Actualizar

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 datos1(10) Segundos	Conjunto de datos 2 (100) Segundos	Conjunto de datos 3 (1000) Segundos	Conjunto de datos 4 (10000) Segundos	Conjunto de datos 5 (100000) segundos
Creacion	0	0,001	0,013	0,39	20,717
operación 1	0	0,003	0,011	0,325	20,786
operación 2	0	0,002	0,014	0,32	19,394
operación 3	0	0,001	0,012	0,328	18,46
operación 4	0	0,002	0,014	0,319	18,863
operación 5	0	0,002	0,011	0,338	18,863
operación 6	0	0,001	0,012	0,326	19,394
operación 7	0	0,001	0,015	0,338	18,863
operación 8	0	0,002	0,011	0,319	20,717
operación 9	0	0,002	0,016	0,325	18,46
operación 10	0	0,001	0,015	0,319	20,786
operación 11	0	0,003	0,014	0,325	19,394
operación 12	0	0,001	0,011	0,326	20,786
operación 13	0	0,002	0,015	0,319	20,717
operación 14	0	0,003	0,012	0,338	18,863
operación 15	0	0,002	0,011	0,326	18,46
operación 16	0	0,001	0,014	0,319	19,394
operación 17	0	0,003	0,011	0,338	20,717
operación 18	0	0,001	0,015	0,328	18,863
operación 19	0	0,002	0,012	0,326	19,394
operación 20	0	0,003	0,014	0,338	18,863
operación 21	0	0,002	0,012	0,328	18,46
operación 22	0	0,003	0,015	0,338	20,717
operación 23	0	0,001	0,014	0,326	19,394
operación 24	0	0,002	0,011	0,319	18,46
operación 25	0	0,001	0,015	0,325	18,863
operación 26	0	0,002	0,014	0,328	18,46
operación 27	0	0,003	0,011	0,326	20,786
operación 28	0	0,002	0,015	0,325	18,46
operación 29	0	0,001	0,011	0,326	18,46
operación 30	0	0,002	0,014	0,325	18,46
operación 31	0	0,001	0,015	0,319	20,717
operación 32	0	0,002	0,012	0,326	18,863
operación 33	0	0,003	0,015	0,338	18,46
operación 34	0	0,001	0,015	0,338	19,394
operación 35	0	0,002	0,011	0,328	18,863
operación 36	0	0,003	0,012	0,325	20,717
operación 37	0	0,001	0,011	0,338	18,46
operación 38	0	0,003	0,015	0,325	20,717
operación 39	0	0,003	0,012	0,338	19,394
operación 40	0	0,001	0,011	0,326	18,46
operación 41	0	0,002	0,015	0,319	18,46
operación 42	0	0,003	0,012	0,328	20,717
operación 43	0	0,002	0,014	0,325	18,863
operación 44	0	0,003	0,015	0,328	19,394
operación 45	0	0,002	0,015	0,319	20,786
operación 46	0	0,001	0,012	0,326	18,46
operación 47	0	0,001	0,012	0,328	20,786
operación 48	0	0,001	0,011	0,326	20,717
operación 49	0	0,002	0,015	0,319	18,46
operación 50	0	0,001	0,015	0,325	19,394
operación 51	0	0,001	0,012	0,338	18,46
operación 52	0	0,003	0,015	0,328	19,394
operación 53	0	0,002	0,015	0,338	18,46
operación 54	0	0,003	0,014	0,319	20,786
operación 55	0	0,001	0,015	0,328	20,717
operación 56	0	0,002	0,012	0,325	20,786
operación 57	0	0,001	0,015	0,319	19,394
operación 58	0	0,001	0,014	0,326	20,786
operación 59	0	0,002	0,015	0,319	20,717
operación 60	0	0,001	0,015	0,319	19,394
operación 61	0	0,001	0,011	0,325	20,786
operación 62	0	0,002	0,012	0,319	20,717
operación 63	0	0,001	0,012	0,338	19,394
operación 64	0	0,001	0,011	0,328	20,717
operación 65	0	0,001	0,015	0,326	18,46
operación 66	0	0,001	0,015	0,319	20,717
operación 67	0	0,002	0,015	0,328	18,863
operación 68	0	0,001	0,012	0,338	18,863
operación 69	0	0,001	0,011	0,328	19,394
operación 70	0	0,001	0,015	0,326	19,394
operación 71	0	0,001	0,012	0,319	18,46
operación 72	0	0,002	0,014	0,338	20,717
operación 73	0	0,001	0,011	0,328	19,394
operación 74	0	0,001	0,014	0,338	20,717
operación 75	0	0,002	0,012	0,328	20,717
operación 76	0	0,001	0,011	0,319	19,394
operación 77	0	0,001	0,015	0,325	18,46
operación 78	0	0,002	0,012	0,338	19,394
operación 79	0	0,001	0,014	0,326	20,717
operación 80	0	0,001	0,011	0,338	19,394
operación 81	0	0,002	0,014	0,319	18,46
operación 82	0	0,001	0,012	0,328	18,863
operación 83	0	0,001	0,015	0,328	18,46
operación 84	0	0,002	0,012	0,319	20,717
operación 85	0	0,001	0,015	0,328	18,46
operación 86	0	0,001	0,012	0,338	19,394
operación 87	0	0,002	0,012	0,319	19,394
operación 88	0	0,001	0,011	0,326	18,46
operación 89	0	0,001	0,012	0,325	20,717
operación 90	0	0,001	0,011	0,328	19,394
operación 91	0	0,002	0,011	0,319	18,46
operación 92	0	0,003	0,015	0,325	18,863
operación 93	0	0,001	0,014	0,328	20,717
operación 94	0	0,002	0,012	0,328	19,394
operación 95	0	0,001	0,014	0,328	18,46
operación 96	0	0,002	0,012	0,325	20,717
operación 97	0	0,003	0,015	0,319	19,394
operación 98	0	0,001	0,012	0,338	18,863
operación 99	0	0,003	0,014	0,326	20,717
operación 100	0	0,001	0,015	0,328	20,717
PROMEDIO	0	0,001681168	0,013138614	0,327653465	19,54

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) MB	Conjunto de datos 2 (100) MB	Conjunto de datos 3 (1000) MB	Conjunto de datos 4 (10000) MB	Conjunto de datos 5 (100000) MB
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 major 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.

	Mejor Tiempo (segundos)	Peor Tiempo (segundos)	Tiempo Promedio (segundos)	Mejor Memoria (MB)	Peor Memoria (MB)	Memoria Promedio (MB)
Conjunto de datos1(10)	0	0	0	7	11	9
Conjunto de datos 2 (100)	0,001	0,003	0,001683168	20	40	30
Conjunto de datos 3 (1000)	0,011	0,015	0,013138614	60	100	80
Conjunto de datos 4 (10000)	0,319	0,39	0,327653465	100	200	150
Conjunto de datos 5 (100000)	18,46	20,786	19,54	400	680	540

Table 4: Análisis de los resultados obtenidos con la implementación de la estructura de datos

REFERENCIAS

[1] Macdonald, T. Spatial Hashing. GameDev.net, 2018. https://www.gamedev.net/articles/programming/general-andgameplay-programming/spatial-hashing-r2697/. [2] Introductory Guide to AABB Tree Collision Detection – Azure From The Trenches. Azurefromthetrenches.com, 2018. https://www.azurefromthetrenches.com/introductory-guideto-aabb-tree-collision-detection

[3]Quadtree.En.wikipedia.org,2018.https://en. wikipedia.org/wiki/Quadtree. [4] GAUL, R. Dynamic AABB Tree Gaul, R. Dynamic AABB Tree. Randy Gaul's Game Programming Blog, 2018.