TRATAMIENTO DE COLISIONES DE ABEJAS ROBÓTICAS CON ÁRBOL QUADTREE

Andrés Felipe Tamayo Acevedo
Universidad EAFIT
Medellín, Colombia
aftamayoa@eafit.edu.co

Jamer José Rebolledo Quiroz
Universidad EAFIT
Medellín, Colombia
jjrebolleq@eafit.edu.co

Mauricio Toro
Universidad EAFIT
Medellín, Colombia
mtorobe@eafit.edu.co

RESUMEN

Cerca de tres cuartos de las especies que se utilizan en cultivos, desde manzanas hasta almendras, necesitan de la polinización de abejas y otros insectos. Infortunadamente, los pesticidas, la deforestación y el cambio climático han causado que disminuya la población de abejas, causando graves problemas a los agricultores. Un dron que pueda polinizar flores puede funcionar, en un futuro no muy lejano, para mejorar el rendimiento de los cultivos. Como un ejemplo, Eijiro Miyaco del instituto avanzado de ciencia y tecnología industrial de Japón, y sus colegas, han creado un dron que puede transportar polen entre flores." (Tomado de https://www.newscientist.com/article/2120832-robotic-bee-couldhelp- pollinate-crops-as-ral-bees-decline/).

Una solución alternativa a este problema es una estructura de datos que nos permita detectar colisiones entre las abejas robóticas, nosotros hemos optado por desarrollar un QuadTree, este nos permite tener una visión global del mapa y así mismo detectar las colisiones. De esta manera se puede tener un control sobre las abejas que están a punto de colisionar y hacer lo pertinente.

Teoría de la computación → Diseño y análisis de algoritmos → datos de diseño y análisis de estructuras → ordenación y búsqueda.

1. INTRODUCCIÓN

Hoy en día, la población de abejas se ha visto muy afectada y reducida por el uso de pesticidas en los cultivos de frutas y vegetales, frente a esta problemática se diseñó un prototipo de dron que hace las funciones de una abeja para así solucionar de una manera rápida y eficaz la falta de abejas.

2. PROBLEMA

El problema que vamos a solucionar es el de las colisiones entre estos drones – abejas, desarrollaremos una estructura de datos que detecte colisiones entre las abejas robóticas.

3. TRABAJOS RELACIONADOS

3.1 Hash espacial

Un hash espacial es una extensión de 2 o 3 dimensiones de la tabla hash, que debería serle familiar desde la biblioteca estándar o el libro / curso de algoritmos de su elección. Por supuesto, como ocurre con la mayoría de estas cosas, hay un giro.

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.

Normalmente, las claves para una tabla hash serían cadenas, pero en un hash espacial usamos 2 o 3 puntos dimensionales como claves. Y aquí es donde entra el giro: para una tabla hash normal, una buena función hash distribuye las teclas lo más uniformemente posible a través de las cubetas disponibles, en un esfuerzo por mantener corto el tiempo de búsqueda. El resultado de esto es que las claves que están muy cerca (lexicográficamente hablando) entre sí, es probable que terminen en cubos distantes. Pero en un hash espacial estamos tratando con ubicaciones en el espacio, y la localidad es muy importante para nosotros (especialmente para la detección de colisiones), por lo que nuestra función hash no cambiará la distribución de las entradas.

3.2 QuadTree

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 divido en cuatro cuadrantes. Al repetir el proceso, el cuadrante se divide de nuevo en cuatro cuadrantes, y así sucesivamente.

Una gran variedad de estructuras jerárquicas existen para representar los datos espaciales. Una técnica normalmente usada es Quadtree. El desarrollo de éstos fue motivado por la necesidad de guardar datos que se insertan con valores idénticos o similares. Este artículo trata de la

representación de datos en el espacio bidimensional. Quadtree también se usa para la representación de datos en los espacios tridimensionales o con hasta 'n' dimensiones.

El término Quadtree se usa para describir una clase de estructuras jerárquicas cuya propiedad en común es el principio de recursividad de descomposición del espacio.

Estas clases, basan su diferencia en los requisitos siguientes:

El tipo del dato en que ellas actúan.
El principio que las guías del proceso de descomposición.
La resolución (inconstante o ninguna).
La familia Quadtree se usa para representar puntos, áreas, curvas, superficies y volúmenes. La descomposición puede hacerse en las mismas partes en cada nivelado (la descomposición regular), o puede depender de los datos de la entrada. La resolución de la descomposición, en otros términos, el número de tiempos en que el proceso de descomposición es aplicado, puede tratarse de antemano, o puede depender de las propiedades de los datos de la entrada.

El primer ejemplo de un Quadtree se relaciona a la representación de un área bidimensional. La región Quadtree que representa las áreas es el tipo más estudiado. Este ejemplo es basado en la subdivisión sucesiva del espacio en cuatro cuadrantes del mismo tamaño. El subcuadrante que contiene datos simplemente se denomina área Negra, y los que no contienen datos se denominan área Blanca. Un subcuadrante que contiene partes de ambos se denomina área Ceniza. Los subcuadrantes Ceniza, que contienen aéreas Blancas y Negras (Vacío y Datos), deben subdividirse sucesivamente hasta que solo queden cuadrantes Negros Y Blancos... (Datos y Vacíos).

Cada cuadrante representa un nodo del Quadtree, los espacios negros y blancos siempre están en las hojas, mientras todos los nodos interiores representan los espacios grises.

3.3 Arbol dinámico AABB

Un árbol dinámico de AABB es un árbol de búsqueda binario para la partición espacial.

Una función de amplificación en una simulación física tiene el trabajo de informar cuándo los cuerpos están colisionando potencialmente. Por lo general, esto se hace a través de pruebas de intersección baratas entre volúmenes delimitadores simples (AABB en este caso).

Hay algunas propiedades interesantes de un árbol dinámico AABB que hacen que la estructura de datos sea bastante simple en términos de implementación.

Hay un par de funciones principales para implementar descritas aquí:

Insertar

Retirar

Actualizar

La estructura de datos subyacente debe ser una gran variedad de nodos. Esto es mucho más óptimo en términos de rendimiento de caché que muchos nodos distribuidos en el montón. Esto es muy importante ya que toda la matriz de nodos se va a recuperar de la memoria cada vez que se actualiza la broadphase.

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. Esto lleva a una breve descripción de invariantes para la estructura de datos:

Todas las ramas deben tener dos hijos válidos

Solo las hojas contienen datos de usuario

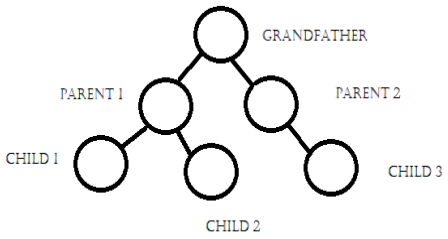
La primera regla permite cierta simplificación de operaciones como insertar / eliminar. No se requiere una bifurcación de código adicional para comprobar si hay elementos NULL ni aumentar el rendimiento del código.

3.4 Estructura de datos cinéticos

Una estructura de datos cinéticos es una estructura de datos utilizada para rastrear un atributo de un sistema geométrico que se mueve continuamente. Por ejemplo, una estructura de datos de casco cinético convexo mantiene el casco convexo de un grupo de n puntos de movimiento. El desarrollo de estructuras de datos cinéticos estuvo motivado por problemas de geometría computacional que involucran

objetos físicos en movimiento continuo, como detección de colisión o visibilidad en robótica, animación o gráficos por computadora.

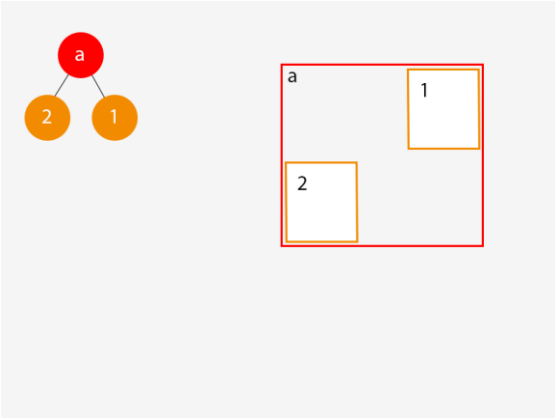
4. Árbol AABB



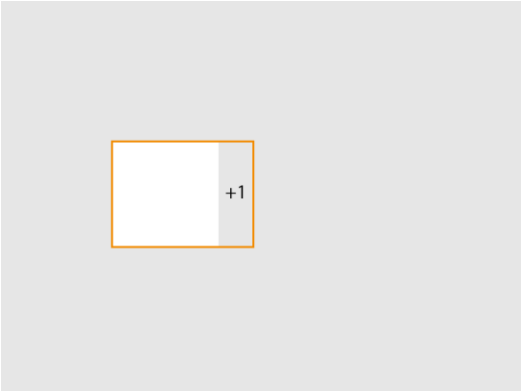
Gráfica 1: 3 generaciones de un árbol AABB.

4.1 Operaciones de la estructura de datos
En nuestro árbol sólo hay un nodo.

Ahora añadimos el segundo nodo.



Gráfica 2: Imagen de una operación de insertar a un árbol AABB.



Gráfica 3: Imagen de una operación de actualizar a un árbol AABB.

4.2 Criterios de diseño de la estructura de datos

Usamos árboles AABB porque son eficientes en cuanto a recorrerlos, lo cual nos sirve para detectar las colisiones entre las abejas.

4.3 Análisis de Complejidad

Calculen la complejidad de las operaciones de la estructura de datos para el peor de los casos. Vean un ejemplo para reportarla:

Método	Complejidad
Insertar	O(log n)
Agregar	O(n log n)
Recorrer	O(n)

Tabla 1: Tabla para reportar la complejidad

4.4 Tiempos de Ejecución
Calculen, (I) el tiempo de ejecución y (II) la memoria usada para las operaciones de la estructura de datos, para el Conjunto de Datos que está en el ZIP

Tomen 100 veces el tiempo de ejecución y memoria de ejecución, para cada conjunto de datos y para cada operación de la estructura de datos

	Conjunto de Datos 1	Conjunto de Datos 2	...Conjunto de Datos n
Creación	10 sg	20 sg	5 sg
Operación 1	12 sg	10 sg	35 sg
Operación 2	15 sg	21 sg	35 sg
Operación n	12 sg	24 sg	35 sg

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

4.5 Memoria

Mencionar la memoria que consume el programa para los conjuntos de datos

	Conjunto de Datos 1	Conjunto de Datos 2	...Conjunto de Datos n
Consumo de memoria	10 MB	20 MB	5 MB

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

4.6 Análisis de los resultados

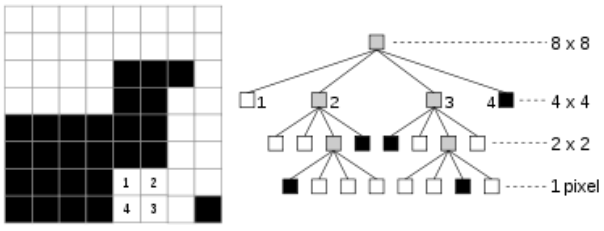
Expliquen los resultados obtenidos. Hagan una gráfica con los datos obtenidos, como por ejemplo:

Tabla de valores durante la ejecución

Estructuras de autocompletado	LinkedList	Arrays	HashMap
Espacio en el Heap	60MB	175MB	384MB
Tiempo creación	1.16 - 1.34 s	0.82 - 1.1 s	2.23 - 2.6 s
Tiempo búsqueda ("a")	0.31 - 0.39 s	0.37 - 0.7 s	0.22 - 0.28 s
Tiempo búsqueda ("zyzzyvas")	0.088 ms	0.038 ms	0.06 ms
Búsqueda ("aerobacteriologically")	0.077 ms	0.041 ms	0.058 ms
Tiempo búsqueda todas las palabras	6.1 - 8.02 s	4.07 - 5.19 s	4.79 - 5.8 s

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

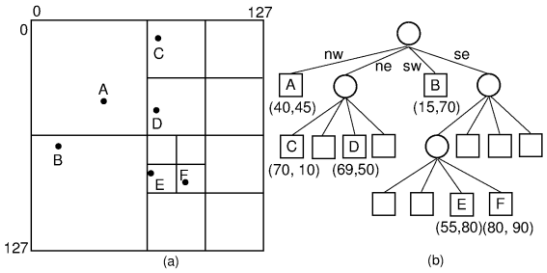
5. QUADTREE



Gráfica 3: QuadTree, cada cuadro es una partición del mapa de abejas, si está en negro, hay abejas y debemos dividirlo, si está en blanco no hay abejas y no es necesario seguirlo dividiendo.

5.1 Operaciones de la estructura de datos

Diseñen las operaciones de la estructura de datos para solucionar finalmente el problema. Incluyan una imagen explicando cada operación



Gráfica 4: Imagen de una operación de insertado en un QuadTree.

5.2 Criterios de diseño de la estructura de datos

Elegimos un QuadTree porque es una estructura de fácil implementación, hay mucha fuente de información sobre este en internet, y se puede usar tanto como para mapas en n dimensiones. Expliquen con criterios objetivos, por qué diseñaron así la estructura de datos. Criterios objetivos son, por ejemplo, la eficiencia en tiempo y memoria. Criterios no objetivos y que rebajan la nota son: “me enfermé”, “fue la primera que encontré”, “la hice el último día”, etc. Recuerden: este es el numeral que más vale en la evaluación con 40%

5.3 Análisis de la Complejidad

Operación	Complejidad
Split	O(1)
Insertar	O(1)
Dfs	O(V + E)

Tabla 5: Tabla para reportar la complejidad

5.4 Tiempos de Ejecución

Número de abejas	10	100	1000	10000	100000	1000000
Creación	4 msg	4 msg	4 msg	4 msg	4 msg	4 msg
Insert	3 msg	10 msg	52 msg	48 msg	3799 msg	7293 msg
Dfs	3 msg	78 msg	628 msg	2748 msg	155129 msg	

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

5.5 Memoria

Mencionar la memoria que consume el programa para los conjuntos de datos

	Conjunto de Datos 1	*Conjunto de Datos 2*	*...Conjunto de Datos n*
Consumo de memoria	10 MB	20 MB	5 MB

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

5.6 Análisis de los resultados

Expliquen los resultados obtenidos. Hagan una gráfica con los datos obtenidos, como por ejemplo:

Tabla de valores durante la ejecución

Estructuras de autocompletado	LinkedList	Arrays	HashMap
Espacio en el Heap	60MB	175MB	384MB
Tiempo creación	1.16 - 1.34 s	0.82 - 1.1 s	2.23 - 2.6 s
Tiempo búsqueda ("a")	0.31 - 0.39 s	0.37 - 0.7 s	0.22 - 0.28 s
Tiempo búsqueda ("zyzzyvas")	0.088 ms	0.038 ms	0.06 ms
Búsqueda ("aerobacteriologically")	0.077 ms	0.041 ms	0.058 ms
Tiempo búsqueda todas las palabras	6.1 - 8.02 s	4.07 - 5.19 s	4.79 - 5.8 s

Tabla 8: Tabla de valores durante la ejecución

6. CONCLUSIONES

Para escribirlas, procedan de la siguiente forma: 1. En un párrafo escriban un resumen de lo más importante que hablaron en el reporte. 2. En otro expliquen los resultados más importantes, por ejemplo, los que se obtuvieron con la solución final. 3. Luego, comparen la primera solución que hicieron con los trabajos relacionados y la solución final. 4. Por último, expliquen los trabajos futuros para una posible continuación de este Proyecto. Aquí también pueden mencionar los problemas que tuvieron durante el desarrollo del proyecto

Lo más importante del proyecto fue que pudimos aprender mucho, implementar una estructura de datos como lo es un QuadTree te acerca mucho a comprender las estructuras de datos y los algoritmos que estas requieren.

Pudimos observar que en cuanto al tema de colisiones es muy bueno usar un QuadTree por eficiencia y facilidad de implementación.

En las primeras soluciones los algoritmos no eran tan eficaces, y no tenían el mejor gasto de memoria, poco a poco fuimos cambiando esas fallas y en la edición final se ven que están en gran parte mejoradas.

A lo largo del desarrollo del proyecto tuvimos problemas como la falta de documentación de ciertas estructuras, la complejidad de algunas, el manejo de datos a gran escala, de hecho cambiamos nuestra estructura original que era un árbol AABB a un QuadTree.

6.1 Trabajos futuros

Como trabajos futuros sería bueno verlo con cuestiones como el vector velocidad de las abejas, ya que no es sólo ver si están muy cerca, poder implementar un algoritmo correctivo en caso de una colisión inmediata que verifique la posición de las demás abejas e implementarlo en 3D.

AGRADECIMIENTOS

Yo, Jamer, quisiera agradecer personalmente a Tamayo, Andrés Felipe, que gracias a sus aportes, tiempo y dedicación logramos sacar este 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.

3. James. Introductory Guide to AABB Tree Collision Detection. AZURE FROM THE TRENCHES, No City, 2017.

4. Basch, Julien. Kinetic Data Structures (Thesis). Stanford University. California, 1999.