Implementación de bases de datos para posición de objetos

Federico Banoy
Universidad Eafit
Colombia
fbanoyr@hotmail.com

Luis Felipe Cardona
Universidad Eafit
Colombia
lcardo45@eafit.edu.co

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

RESUMEN

En Colombia, Bello unas abejas robóticas para polinizar las flores, pero estas abejas no pueden colisionar unas con otras, por lo que se necesita crear un algoritmo para que estas no lo hagan, una solución posible seria comparar la posición de las abejas (solo se toman en un plano 2D) y compararlas con el resto de las abejas .

La solución para nosotros fue crear un Quadtree, ya que este nos permite comparar las distancias de las abejas cercanas a una abeja en concreto, es decir, no toca analizar todas las abejas que se insertaron, lo que reduca drásticamente el tiempo de análisis (aunque gaste mas memoria)

Palabras clave

Information systems → database management system engines

Software and its engineering → software notations and tools→ general programming languages → languages features → inheritance

Software and its engineering →software notations and tools → general programming → languages features → classes and objects

1. INTRODUCCIÓN

Cuando se está intentando manejar varios datos se tiene que tener cuenta la memoria que ocupa y la velocidad del programa, como por ejemplo drones de seguridad, para que no estén en la misma área estoy pueden obtener la posición del resto de los drones para estar en un lugar donde no haya ningún otro, por lo que se necesita un algoritmo donde obtenga la posición de los drones, y para esto podemos usar estructuras de datos.

2. PROBLEMA

En Bello hay unas abejas robóticas que necesitan polinizar flores y evitar estar en la misma zona, ya que así se pueden chocar, por lo que se necesita avisarle al sistema cuales abejas están a menos de 50 metros entre ellas.

3. TRABAJOS RELACIONADOS

3.1 Quadtree

"Quadtree" es un algoritmo nombrado por Raphael Finkel y J.L. Bentley en 1974, es derivado del Qtree, es un algoritmo para organizar datos, usado normalmente e colisiones, este tiene una capacidad, cuando esta se llena el Quadtree se parte en 4 y le pasa los datos a estos nuevos Quadtree.

3.2 Rtree

" R-tree" fue propuesto por Antonin Guttman en 198, es un algoritmo que permitre trabajar en varias dimensiones, y sirve para ubicar rectángulos y polígonos en un espacio, para ubicarlos y en el resto de lugares sepan donde esta esté rectángulo o polígono.

3.3 Btree

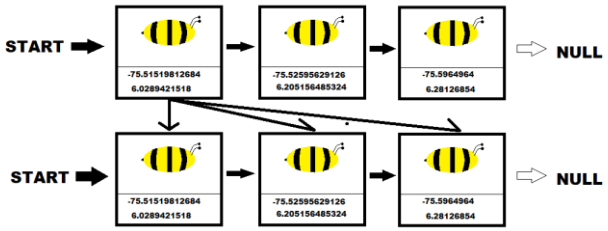
"B-tree" creado por Rudolf Bayer, Edward M. McCreight en 1971 (tiene una estructura parecida a el árbol binario, con la diferencia que este árbol puede tener más de 2 hijos por cada nodo) fue creada para insertar, buscar y borrar información en un tiempo más rápido, por lo que puede trabajar con más datos

3.4 B+ tree

"B+ tree" similar a el Btree, con la diferencia de que este lo normal es que tenga muchos sub nodos por cada nodo, y tiene un B^x (otra estructura da todos) que le ayuda a trabajar con datos no estacionarios (como partículas moviéndose en un mapa)

4. Double ArrayList

Comparar cada abeja con el resto de abejas



Gráfica 1: muestra las abejas guardadas en un Arraylist y por cada abeja recorre todo el ArrayList

4.1 Operaciones de la estructura de datos

La estructura es crear un ArrayList de abejas, guardando su posición den X y Y(abeja es un objeto) y después por medio de 2 for anidados, comparar cada abeja con el rest

4.2 Criterios de diseño de la estructura de datos

El diseño de esta solución es por su simpleza, aunque en memoria es de las que gasta menos, ya que solo tiene que crear un ArrayList de abejas, en velocidad es bastante malo, pero su modo simple sirve mucho para grupos de abejas pequeños, ya que la complejidad al ser n^2 entre más números, el algoritmo se demorara exponencialmente más.

4.3 Análisis de Complejidad

En el peor de los casos es $O(n^2)$ las abejas en peligro tienen que recorrer todo el Arraylist por abeja.

Añadir	O(1)
Buscar	O(n)
Imprimir abejas en peligro	$O(n^2)$

Tabla 1: complejidad de los métodos

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

Abejas	10	100	1000	10000	100000	1000000
operación(1) crea el ArrayList	0,1s	0,1s	0,1s	0,1s	0,2s	0,6s
operación(2)Ver si se choca	0,1s	0,1s	0,2s	0,5	312,7s	más de 10 min

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

4.5 Memoria

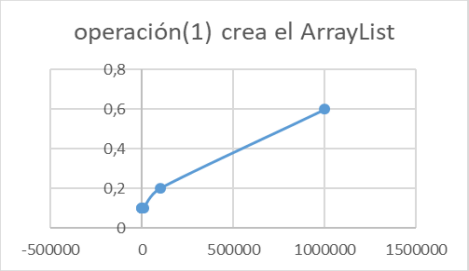
Mencionar la memoria que consume el programa para los conjuntos de datos

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

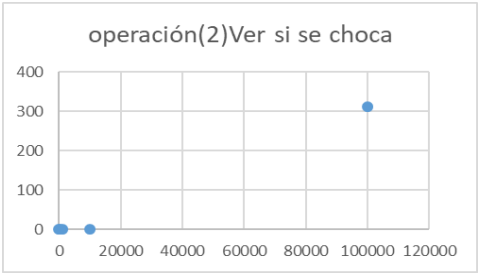
Tabla 3: Consumo de memoria de la estructura de datos con diferentes conjuntos de datos

4.6 Análisis de los resultados

Como era de esperarse por la complejidad de ambos métodos usados, uno es lineal (insertar)

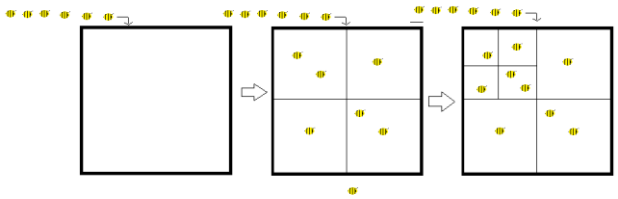


Grafica 1: tala que compara número de abejas con el tiempo que se tarda en crear el ArrayList de abejas



Grafica 2: compara el número de abejas con lo que se demora en ver si la abeja está en peligro de chocar

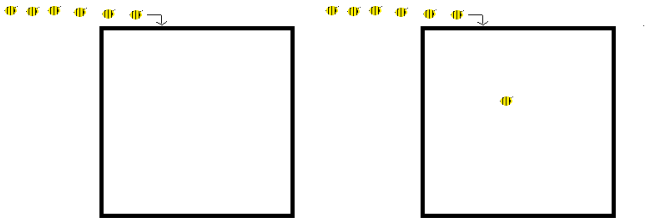
5. QuadTree de abejas



Grafica 1: en la lista se muestra un Quadtree que guarda abejas en sus instancias

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 2: imagen añadir añade una abeja a el quadtree

5.2 Criterios de diseño de la estructura de datos lo creamos de esta nueva manera para mejorar el tiempo que se tomaba este en analizar las distancias, ya que el anterior era muy ineficiente en este aspecto, por lo que temamos el Quadtree como referencia, aunque el Quadtree consuma más memoria guardando más variables que el ArrayList, por la velocidad que da para analizar, es bueno implementarlo

5.3 Análisis de la Complejidad

añadir	O(n)
nombrar	O(h+log$_4$n)
partir	O(n)
insertar	O(n)

Tabla 5: Tabla para reportar la complejidad

5.4 Tiempos de Ejecución

Abejas	10	100	1000	10000	100000	1000000
añadir	0,1s	0,1s	0,1s	0,1s	0,2s	0,6s
nombrar	0,1s	0,1s	0,1s	0,1s	0,5s	5s
partir	0,1s	0,1s	0,1s	0,1s	0,1s	0,1s
insertar	0,1s	0,1s	0,1s	0,1s	0,2s	0,6s

Tabla 6: Tiempos de ejecución de las operaciones de

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

Lo más importante para nosotros después de hacer el primer la segunda entrega (con los ArrayList) fue la velocidad, ya que cuando estábamos analizando el de un millón da abejas el computador, aunque la memoria permitiera hacer esto, nunca pudimos ver cuál era el resultado, ya que esperamos más de 15 minutos y no estaba cerca de terminar, incluso quitándole algunas instrucciones para que hiciera tolo los procesos internos, 15 minutos no era tiempo suficiente. Por lo que nuestra visión para esta segunda entrega era mejorar el tiempo para ver si la abeja iba a colisionar con otra.

se puede ver que consume mucho menos tiempo al realizar la acción de evaluar si 2 abejas se chocaron, pero al crear tantas instancias de cuadretes se llega a ocupar más memoria

Lo que nos gustaría hacer en un futuro, con este proyecto sería analizar partículas en movimiento, para poder crear algún Videojuego o para analizar verdaderamente objetos que se mueven y no están estacionarios. Un problema grande que tuvimos fue el rebalanceo del árbol, ya que cuando un Quadtree se llena este le debería pasar las abejas que tenía a sus nuevos sub Quadtree, pero no lo pudimos implementar

6.1 Trabajos futuros

Nos gustaría mejorar en un futuro entender más a profundidad la implementación de estos todo esto a situaciones de la vida real

AGRADECIMIENTOS

Nosotros agradecemos por su ayuda con la investigación a David Gomez Moreno, a Carla por la ayuda de la implementación de la clase Nodo, a Luis Bernado Zuluaga y a Manuela Zapata Girarlo por su ayuda con la implementación de la lectura de los archivos

REFERENCIAS

B-Trees with Inserts and Deletes: Why Free-at-Empty Is Better Than Merge-at-Half* THEODORE JOHNSON AND DENNIS SHASHA Courant Institute of Mathematical Sciences, New York University, New York, New York 10012 Received July 3, 1990; revised August 6, 1990

Parametric R-Tree: An Index Structure for Moving Objects∗ Mengchu Cai IBM Silicon Valley Laboratory 555 Bailey Ave. San Jose, CA 95141, USA mcai@us.ibm.com Peter Revesz Dept. of Computer Science and Engineering University of Nebraska–Lincoln Lincoln, NE 68588, USA revesz@cse.unl.edu

Implementation of Quad Tree Index. Insertion algorithm implemented based on design of quad tree index in H. Samet, The Design and Analysis of Spatial Data Structures. Massachusetts: Addison Wesley Publishing Company, 1989.