ESTRUCTURA DE DATOS 1
Código ST0245

Laboratorio Nro. 4
Tablas de Hash y Árboles

Diego Alexander Múnera Tobón
Universidad Eafit
Medellín, Colombia
damunerat@eafit.edu.co

María Antonia Velásquez
Universidad Eafit
Medellín, Colombia
mavelasqur@eafit.edu.co

1.1 Código en la sección de “código”

2.1) Código en la sección de “código en línea”

3.1) Estructura de datos para ejercicio 1.1

El objetivo de este algoritmo es identificar aquellas abejas que se encuentren a 100 metros o menos de distancia de otra abeja, para prevenir colisiones entre ellas.

Sea n el número de abejas, la complejidad del algoritmo para calcular las colisiones es O(n). En donde la totalidad de los métodos de las clases Octree son también O(n), excepto la función hashing, que es de 0(1). Se tiene en cuenta que colisionaran aquellas abejas que estén a 100 o menos metros de distancia. Para calcular las colisiones de un grupo de abejas robóticas en 3D de acuerdo a su longitud (grados), latitud (grados) y altura (metros) se implementa un árbol cuyos nodos tienen cada uno 8 nodos hijos.

Para esto se implementan arreglos de 8 posiciones en donde cada posición almacenará una lista enlazada de abejas y el objetivo es dividir estas listas enlazadas en subárboles de 8 nodos recursivamente hasta que cada lista enlazada cuente con una sola abeja o hasta que se identifique que este subgrupo de abejas colisionará. Para saber esto se tiene un método que calcula la diagonal de un cubo en 3D en donde una esquina será la posición con menor latitud, longitud y altura y la otra será la posición con mayor latitud, longitud y altura. Para este cálculo se deben pasar los valores de longitud y latitud de grados a metros

Luego, descartando que el rango de posiciones de este subgrupo de abejas es mayor a 100 metros, se crear un nuevo arreglo con 8 posiciones, en donde en cada posición se almacenará una lista enlazada de abejas. Para saber en qué posición va cada abeja se usa la función hashing la cual recibe como parámetro la abeja que se quiere posicionar y un array de doubles donde están los valores medios de la longitud, la latitud y altura del subgrupo al cual pertenece la abeja. Los valores medios se calculan restando el máximo menos el mínimo, dividirlo entre 2

y luego ese resultado sumándolo al mínimo, así estaríamos obteniendo el valor que está en el centro del mínimo y el máximo

La función hashing lo que hace es dividir ese espacio en 3 dimensiones que forma el subgrupo donde está la abeja y dividirlo en 8 sub-cubos de acuerdo al valor medio de cada coordenada. Así el cubo "0" está conformado por todas las abejas cuya longitud, latitud y altura este debajo del valor medio de cada coordenada. Así se garantiza que cada abeja pertenezca a uno y solamente a un cubo de los 8. Luego de identificar a cuál cubo pertenece la abeja, la función hashing retorna un entero de 0 a 7 el cual será la posición del arreglo al cual pertenece. Finalmente se tiene un método choque() que recibe la lista enlazada de aquellas abejas que colisionan e imprime las coordenadas de cada una de ellas

3.4) Calcular complejidad del algoritmo 2.1

En el caso de que no sea necesario digitar el INPUT, o sea, que el árbol venga completo, la complejidad sería O(1).
En el caso contrario de que sea necesario primero instanciar y llenar el árbol la complejidad sería O(n). Esto porque la lectura de los datos en la forma post-order tiene una complejidad constante y lo que afecta el tiempo de ejecución es la inserción de los datos.

3.5)

El algoritmo crea inicialmente los nodos y el árbol, dentro de las clases nodo y árbol hay 2 métodos en común que son los principales; insertarDato y recorridoPO.

insertarDato es el método más sencillo que es simplemente añadir datos al árbol con la convención de que los valores menores al valor de la raíz van a la izquierda y los mayores a la derecha. Al final de la clase árbol hay un método publico en el que se llama insertarDato que es un método privado.

recorridoPO en este método hacemos dos llamados recursivos que nos dicen que, si el nodo que pasamos por parámetro NO está vacío, imprimimos sus nodos "vecinos" primero el de la izquierda y luego el de la derecha

Al final de la clase árbol tenemos un método publico que se llama "inpt" este método lo que hace es agruparnos todos los demás métodos de la clase e instanciar la clase árbol para que la inserción sea más eficiente en términos de tiempo (al no tener que darle manualmente append a cada uno de los números del input) y finalmente llamamos este método mediante la clase Árbol y se ejecuta todo el código.

4) Simulacro de Parcial

4.1

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

- b) que inician con la misma letra colisionan
- d) 0(1)

4.2 c) 3

4.3

- false
- a.data()
- a.izq, suma-a.data()
- a.der, suma-a.data()

4.4

- c) T(n)=2.T(n/2)+C
- a) O(n)
- d) Wilkenson, Joaquina, Eustaquia, Florinda, Eustaquio, Jovin, Sufranio, Piolina, Wilberta, Piolin, Usnavy
- a) Cambiar el orden de las líneas 03, 04 y 05 por 05, 04, 03

4.5

- toInsert==null
- toInsert>p

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473