Tecnológico de Monterrey

Actividad 3.4
Programacion de estructuras de datos y algoritmos fundamentales
TC1031 Grupo 4
Prof. Jesús Guillermo Falcón Cardona

Integrantes:
Patricio Mendoza Pasapera A00830337
Christian Duran Garcia A01654229
Jorge Leonardo Garcia Reynoso A01734836

22/10/2021

Investigación

Cuando utilizamos listas, descubrimos la principal desventaja que estas tienen, esta siendo que se encuentran obligados a “pasar” por cada uno de los datos hasta llegar al indicado, lo cual no es un gran problema si la lista no tiene muchos datos, pero si se cuenta con muchos puede llegar a ser muy tardado encontrar y dato, y ni decir si se quieren ordenar, lo cual también es un desafío extra, por lo que alguna forma que no requiera pasar por cada uno de los datos para introducir uno nuevo o para ordenarlos seria bastante util, es ahí donde el BST (Binary Search Tree) es bastante útil.

Un BST consiste de nodos unidos entre sí, pero a diferencia de una lista estas no se encuentran ligadas de forma lineal, sino que los nodos pueden tener hasta dos nodos hijos, esto mediante dos apuntadores, uno a la izquierda y otro a la derecha, y esto en cada nodo hasta llegar al final. Lo que separa a este tipo de arbol de otros es que en este árbol, al añadir nuevos datos estos son ordenados de forma automática, teniendo un nodo inicial conocido como la raíz, el cual tendrá asignado un valor numérico, el cual es importante, ya que cuando se añada un nuevo valor este será ubicado a la derecha o a la izquierda dependiendo de su valor, donde a la izquierda irán valores menores y a la derecha los mayores, esto se aplicará para todos los nuevos nodos que se creen, teniendo así que los que se encuentren hasta la izquierda serán los menores y hasta la derecha los mayores.

La principal ventaja es que podemos encontrar un valor numérico de forma mas rapida, ya que no requerimos pasar por todos los datos, sino solo los que sean necesarios para llegar a ese número, comparando el valor del nodo en el que nos encontramos con el valor buscado hasta encontrar el valor que buscamos. La principal desventaja es que, debido a que se ordena de forma automática dependiendo de la raíz, podemos tener un árbol que no está balanceado, teniendo muchos datos en un solo lado del árbol, haciéndolo muy parecido a las listas que vimos anteriormente.

Análisis de complejidad

Para la solución de nuestra actividad decidimos utilizar un BST, por lo que utilizamos funciones que son típicas de este tipo de lista, como el search, getleft, getright, entre otros, los cuales tienen una complejidad de O(n) u O(1), pero las dos funciones que son únicas para nuestro BST son para calcular y encontrar las ips que más se repitan, esto utilizando de base el recorrido inorden, pasando por cada uno de las hojas del árbol:

```
void calcMaxRep(Node<T>* current){ // Recorrido inorden para calcular max rep
    if(current == nullptr){
        return;
    }
    else{
        calcMaxRep(current->getLeft());
        maxAccess = max(maxAccess, current->getAccess());
        calcMaxRep(current->getRight());
    }
}
```

Θ(1)

T(n-1)

O(1)

T(n-k-1)

```
void printMaxAccess(Node<T>* current, int maxAccess){ // Recorrido inorden para desplegar max rep
    if(current == nullptr){
        return;
    }
    else{
        printMaxAccess(current->getLeft(), maxAccess);
        if(current->getAccess() >= maxAccess - 5){
            current->getData().imprimir();
            cout << "Num accesos: " << current->getAccess() << endl;
        }
        printMaxAccess(current->getRight(), maxAccess);
    }
}
```

O(1)

T(n-1)

O(1)

O(1)

T(n-k-1)

Como mencione anteriormente, ambas funciones utilizan el recorrido inorden, el cual aunque es un método recursivo requiere pasar por cada uno de los elementos del árbol, por lo que ambas funciones tienen una complejidad de O(n), que es igual para todos los tipos de recorrido de un árbol.

Reflexión

El uso de los árboles BST es bastante útil, ya que permite encontrar la información de manera más rápida que otros tipos de listas, además que se ordena en automático, por lo que no será necesario gastar recursos y tiempo en los distintos algoritmos de ordenamiento que estudiamos anteriormente, teniendo la capacidad de ser mucho más eficientes que el resto de listas que vimos en esta clase, sin embargo, al igual que tiene sus ventajas, también cuenta con sus desventajas, la principal, como mencione anteriormente, es la probabilidad de que no se encuentre balanceado, teniendo más hojas de un lado que de otro haciéndolo muy parecido a las otras listas que estudiamos, con la diferencia que siempre se encuentra ordenado.

Se sabe que los algoritmos más eficientes son aquellos que tienen una complejidad de O(1), y aunque esto sea casi imposible de lograr con este tipo de listas, siento que nos acercamos cada vez más a un algoritmo que sea el más eficiente, teniendo el BST la capacidad de ser uno de los más eficientes.

Referencias

Sedgewick R., Wayne K. (2011) *3.2 Binary Search Trees.* Algorithms 4ta Edición. Pearson. Consultado el 22/10/21, en: https://algs4.cs.princeton.edu/32bst/

JavaPoint (s.f.) *Binary Search Trees.* Consultado el 22/10/21, en: https://www.javatpoint.com/binary-search-tree