Proyecto 2 · Manual técnico

Karla Ernestina González Polanco
202006688

Moviecats

Tabla de contenidos

DESCRIPCIÓN DEL PROBLEMA

Moviecats es una conocida franquicia guatemalteca especializada en alquiler de cine y videojuegos a través de tiendas físicas, servicios por correo y video bajo demanda. La empresa fue fundada en el año 2019, contaba con más de 50 establecimientos a nivel nacional.

Moviecats se vio lastrada por la pandemia que vino a restringir el acceso a todas franquicias, haciendo que las ventas empezarán a disminuir haciendo que la empresa se viniera a pique, los directivos de la empresa propusieron transformar su negocio y competir con otros servicios de Streaming como, por ejemplo: Disney+, Netflix, HBO, DAZN. Por lo que se le solicita a usted que ha desarrollado la aplicación de CatsBooks exitosamente, para que pueda implementar esta nueva aplicación Web, con el fin de agilizar, y brindar un nuevo servicio a los clientes. Para ello dicha aplicación web estará desarrollada en el lenguaje de javascript.

REQUERIMIENTOS DEL SISTEMA

- Espacio en memoria: 10 MB como mínimo.
- Memoria RAM: 1 GB como mínimo.
- Tarjeta gráfica: no requerida.
- Navegador a internet: el disponible, de preferencia Google Chrome.
- Acceso a internet.

ESTRUCTURAS DE DATOS

Lista simplemente enlazada

Inserción

Recibe un valor que será objeto de un nuevo nodo que llamaremos temporal. La inserción se realiza insertando cada nuevo valor en la cabeza de la lista, por último, se aumenta el tamaño de la lista.

Vacía

Retorna el valor de verificar si existe o no una cabeza. En caso de que no exista se asume que la lista está vacía.

Existe

Recibe los valores de user y password y recorre la lista buscando al elemento que tenga los parámetros buscados. En caso de no encontrarlo se retorna null.

Árbol Binario de Búsqueda

Inserción

- **AgregarR:** Método que inicialmente recibe el valor que se insertará y el valor de su raíz el cual irá cambiando (aumentando el número de nodos) después de insertar este nuevo valor.
- **Agregar:**
    - Si no existe la raíz, retorna el valor en forma de nodo y ahora la raíz será este nodo.
    - Si ya hay una raíz se compara su valor (dni) con el del nuevo nodo que se quiere agregar y se hace una función recursiva.
        - Si es menor, se verifica si el nodo izquierdo es nulo (utilizando la función agregar pero como parámetro el nodo izquierdo al nodo actual), si lo es se asigna el nuevo nodo a la izquierda del nodo padre, si no es nulo se sigue haciendo la comparación hasta que el nodo izquierdo al nodo actual sea nulo (significaría que ya llegó al último nivel).
        - Si es mayor, se verifica si el nodo derecho es nulo (utilizando la función agregar pero como parámetro el nodo izquierdo al nodo actual), si lo es se asigna el nuevo nodo a la derecha del nodo padre, si no es nulo se sigue haciendo la comparación hasta que el nodo derecho al nodo actual sea nulo (significaría que ya llegó al último nivel).

  Al final se retorna el nodo inicial (la cabeza) pero con el nuevo nodo ya insertado en la posición correspondiente y todos los demás nodos que tenía hasta antes de la inserción.

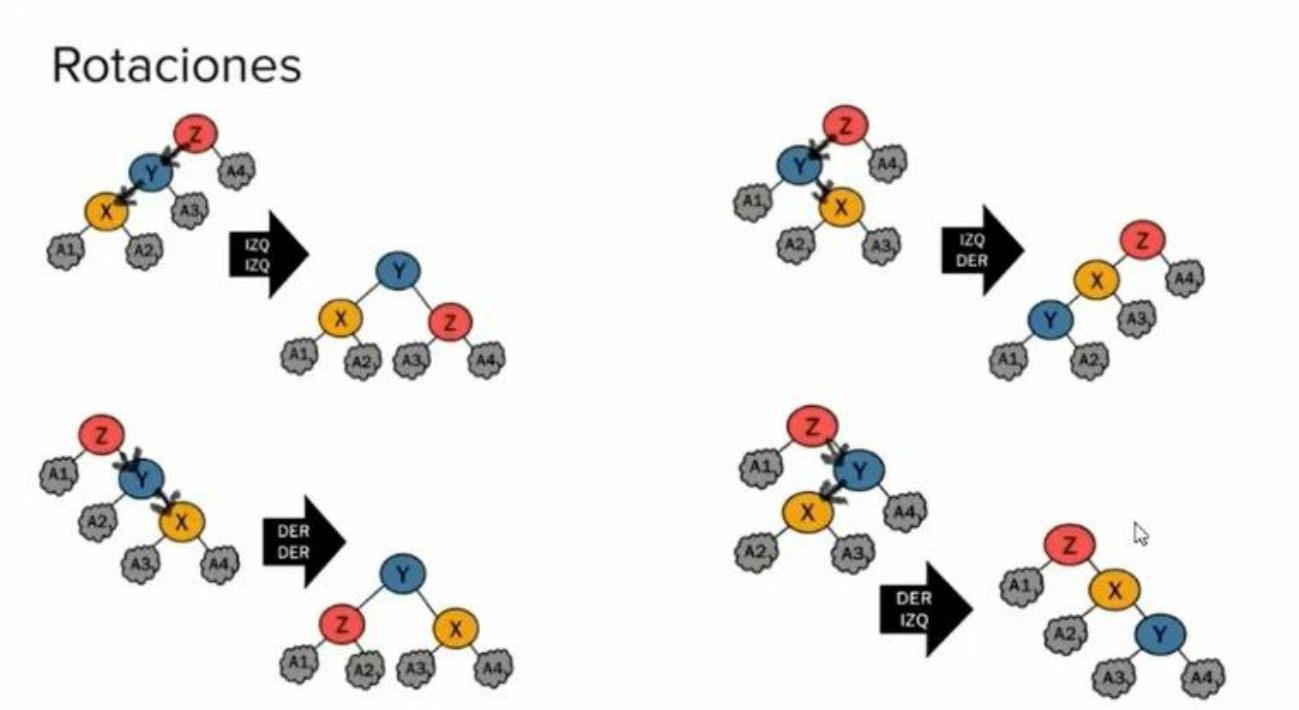
Graficación

Utilizando graphviz y d3 para renderizar el archivo dot – primero se verifica si el nodo que se pasa como parámetro (primeramente, la raíz) es nulo, en caso de ser así no se agrega nada a dot. Si existe se va recorriendo el árbol viendo si el nodo actual tiene un valor a su derecha e izquierda y dependiendo de si tiene uno de los dos hijos o los dos se le agregan las etiquetas <C#> al nodo actual para así apuntarlos gráficamente. El mismo proceso se realiza recursivamente para cada nodo del árbol.

Ordenamientos

Únicamente se hace un recorrido del árbol dependiendo del tipo de ordenamiento. Al pasar por el nodo actual se hace uso de la función newDiv que retorna el código HTML que tendrá el div del actor en el HTML. Su visualización inicial en el HTML es en el archivo Pages, función showActors donde se muestran en postorden.

Árbol AVL



Inserción

- **agregarNodo:** Método que inicialmente recibe el valor que se insertará y el valor de su raíz el cual irá cambiando (aumentando el número de nodos) después de insertar este nuevo valor.
- **Agregar:**
    - Si no existe la raíz, retorna el valor en forma de nodo y ahora la raíz será este nodo.
    - Si ya hay una raíz se compara su valor (id) con el del nuevo nodo que se quiere agregar y se hace una función recursiva.
        - Si es menor, se verifica si el nodo izquierdo es nulo (utilizando la función agregar pero como parámetro el nodo izquierdo al nodo actual), si lo es se verifica la altura que tendrá el nodo restando la altura del nodo derecho – la altura del nodo izquierdo. En caso de que el resultado sea -2, significará que el árbol no está balanceado y tendrá que hacerse otra verificación.

          Si el valor del id actual es menor al id del nodo izquierdo al actual, significará que el desbalanceo viene de la izquierda por lo que tendrá que hacerse una rotación simple a la izquierda (figura 1. derecha arriba). Si el valor es mayor, el desbalanceo viene de la derecha por lo que será necesaria una rotación doble a la izquierda (figura 1. Izquierda arriba).
        - Si es mayor, se verifica si el nodo derecho es nulo (utilizando la función agregar pero como parámetro el nodo derecho al nodo actual), si lo es se verifica la altura que tendrá el nodo restando la altura del nodo derecho – la altura del nodo izquierdo. En caso de que el resultado sea 2, significará que el árbol no está balanceado y tendrá que hacerse otra verificación.

          Si el valor del id actual es mayor al id del nodo derecho al actual, significará que el desbalanceo viene de la derecha por lo que tendrá que hacerse una rotación simple a la derecha (figura 1. derecha abajo). Si el valor es

menor, el desbalanceo viene de la izquierda por lo que será necesaria una rotación doble a la derecha (figura 1. Izquierda abajo).

- Si no es mayor ni menor, significará que está repetido por lo que solo se asigna nuevamente el valor al nodo.

Se cambia el atributo de altura del nodo con el método de maxheight sumándole 1 ya que se agregó un nuevo nodo. Se retorna el nodo con todos los nuevos nodos agregados al árbol.

Rotaciones

- **Rotación simple a la izquierda:** Figura 1 – izquierda arriba. Recibe como parámetro un nodo, que al ver la figura será el nodo Z y hace uso de un auxiliar que será el nodo Y.

  Primero se hace el cambio de hijos, el hijo derecho del nodo auxiliar Y pasa a ser el hijo derecho del nodo Z y el nodo Z pasa a ser el hijo derecho del nodo auxiliar. Se asignan las nuevas alturas de ambos nodos que ahora son parte de un árbol balanceado sumando 1. Por último, ahora el nodo en el nivel superior de la rotación será el nodo auxiliar por lo que se retorna este.

- **Rotación simple a la derecha:** Figura 1 – derecha abajo. Recibe como parámetro un nodo, que al ver la figura será el nodo Z y hace uso de un auxiliar que será el nodo Y.

  Primero se hace el cambio de hijos, el hijo izquierdo del nodo auxiliar Y pasa a ser el hijo izquierdo del nodo Z y el nodo Z pasa a ser el hijo izquierdo del nodo auxiliar. Se asignan las nuevas alturas de ambos nodos que ahora son parte de un árbol balanceado sumando 1. Por último, ahora el nodo en el nivel superior de la rotación será el nodo auxiliar por lo que se retorna este.

- **Rotación doble por la izquierda:** Recibe como parámetro un nodo al cual se asignará a su derecha una rotación simple a la izquierda de su nodo derecho. Por último, se retorna una rotación

Tabla HASH

*si el dato se inserta a la lista no se suma

El tamaño inicial de la tabla es de 20 posiciones, al sobrepasar el 75% se hace un rehashing – aumentar 5 posiciones por cada posición ocupada.

$$20 * 75\% = 15$$

Tendrá que hacerse un rehashing cuando la tabla tenga 16 elementos.

$$16 * 5 = 80 \rightarrow \text{nuevo tamaño de la tabla}$$

Constructor

- Ocupados. Representa la cantidad de datos que se han insertado en la tabla (cabecera).
- Tabla. Arreglo simple que almacena las “cabeceras” desde 0 hasta el tamaño.

- Size. Tamaño del arreglo.
- Para cada índice de la tabla se hace push de una lista simple.

Inserción

- Ecuación para inserción:

índice = id_categoría % tamaño

Se crea una variable i que almacena el valor de retorno de la función hash_pos. Se hace la validación: si la tabla en el índice obtenido no tiene valor insertados a su lista simple, se suma uno a la cantidad de ocupados. Luego se agrega el nuevo valor a la tabla simplemente enlazada del índice y se hace la función de rehashing.

Índice del valor (Hash_pos)

Función que retorna el índice que tendrá el valor ingresado (en este caso el id de la categoría) al hacer la ecuación para inserción ya indicada.

Rehashing

Se obtiene el porcentaje de ocupación que tiene la tabla en ese momento. Si este es menor o igual al 75% se termina la función. Si el porcentaje es mayor al 75% del tamaño del array, se crean dos variables, una donde se almacena la tabla original y la segunda que guarda el tamaño original de la tabla. Se setea el nuevo tamaño que tendrá la tabla

size = ocupados * 5

Se "vacía" la tabla y con ayuda de los temporales se recorre la tabla original y para cada índice se valida que no esté vacío este índice de cabecera. Se recorre la lista que tenía cada índice y se utiliza la función de inserción para volver a calcular el índice que tendrán.

LIBRERÍAS UTILIZADAS

Sha256

https://cdnjs.cloudflare.com/ajax/libs/js-sha256/0.9.0/sha256.min.js

Librería desarrollada para Javascript que utiliza funciones hash criptográficas para encriptar información. Utilizada para encriptar las contraseñas de los usuarios y la información de los bloques en el árbol Merkle para el Blockchain.

d3 - Graphviz

https://unpkg.com/d3-graphviz@3.0.5/build/d3-graphviz.js

Renderiza el texto con formato .dot que se genera de las estructuras creadas.

SaveSVGasPNG

https://cdnjs.cloudflare.com/ajax/libs/save-svg-as-png/1.4.17/saveSvgAsPng.js

Utilizadas para las opciones de descarga. Toman el objeto svg que se le pase como parámetro y se especifica el nombre con el que deberá guardarse.