



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN
INGENIERÍA EN COMPUTACIÓN

ESTRUCTURA DE DATOS
MIGUEL ANGEL SANCHEZ HERNANDEZ

PROYECTO FINAL
OLVERA MARTÍNEZ LEONARDO
VILLA GARCÍA OSCAR JOAQUÍN
SOUBLETT MARTÍNEZ ERANDI

Índice

Objetivos.....	3
Introducción.....	3
Planteamiento del problema.....	3
Solución de la problemática.....	4
Desarrollo de la solución.....	4
Cola y Lista Simple.....	4
Estacionamiento.....	4
Verdugo.....	5
Ordenamiento: Burbuja.....	6
Ordenamiento: Selección.....	6
Ordenamiento: Inserción.....	7
Ordenamiento: Mezcla.....	7
Ordenamiento: Quicksort.....	8
Búsqueda: Secuencial.....	8
Búsqueda: Binario.....	9
Arboles Binarios.....	10
Conclusión.....	10
Bibliografía.....	11

Objetivos

- Utilizar los conocimientos previamente adquiridos sobre las Estructuras de datos como lo son las Listas Simples, Colas y Pilas para la resolución de distintas problemáticas.
- Demostrar la versatilidad de las Listas Simples en la solución de problemas.
- Aplicar los nuevos conocimientos adquiridos en usando Arboles Binarios y Grafos para resolver problemas.

Introducción

Uno de los más grandes desafíos de la programación y en general en el mundo computacional es el uso de los datos. Cualquier cosa que tenga un equipo electrónico usa y almacena datos, puede ser una computadora, tu celular, televisor, impresora o incluso tu cafetera. El uso de la información y los datos en la programación es tan importante que se dedican áreas de estudio y especialización completas en este ámbito pues pueden ser una cantidad de datos gigantes o mínimas, datos simples como números o tan complejos como el mismo programador lo permita, pero todo esto tiene que funcionar y de una manera eficiente y ese es el trabajo que nosotros tenemos que hacer como ingenieros en computación.

Los programadores principiantes se encuentran frente a un problema cuando quieren trabajar con datos, inicialmente trabajan estos datos con arreglos, pero estos tienen un problema ya que son estáticos y no pueden cambiar durante la ejecución del programa, si el usuario quiere agregar más o quitar no se puede, es entonces que usamos las estructuras de datos dinámicas. Las estructuras de datos dinámicas son mejores en comparación a las estáticas pues estas pueden aumentar y disminuir en tamaño durante la ejecución del programa además son más versátiles en cuestiones de memoria pues no necesitan un bloque unitario en la RAM.

Las estructuras que usaran en este proyecto son: Colas, las cuales son estructuras de datos dinámicas y funcionan como una “cola del super” en donde el primer elemento que nosotros metamos será el primero en salir; Listas Simples, son estructuras lineales y dinámicas de datos las cuales funcionan como un arreglo pero estas pueden agregar elementos en cualquier elemento de la misma lista; Arboles; Árboles, representan las estructuras de datos no-lineales y las dinámicas más relevantes en computación, gráficamente se ven como un árbol invertido; Grafos, Estructura de un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.

Planteamiento del problema

Utilizando las estructuras anteriormente mencionadas resolveremos distintos problemas tales como: la simulación de un estacionamiento donde entraran un número n de autos y saldrán algunos de esos de manera aleatoria pero mantendrán su lugar; una simulación de un verdugo donde tendremos n número de prisioneros el verdugo comenzará a matarlos con una secuencia determinada, el objetivo será decidir en qué prisionero iniciar para salvar a uno que será nuestro amigo; un gráfico en donde tomaremos datos de un archivo en forma de lista de personas con sus datos (nombre, edad, peso y altura) donde tendremos

que ordenar mediante una gráfica de barras por edad usando distintos métodos de ordenamiento como: Burbuja, Selección, Inserción, Mezcla, Quicksort; utilizar la lista anteriormente mencionada tomando en cuenta en los datos la edad y realizar una búsqueda de un dato utilizando la búsqueda secuencial y la binaria; mediante arboles resolver una expresión pre fija; mediante grafos resolver Dijkstra.

Solución de la problemática

Los problemas anteriores son formas de ordenar información, como planteamos en la introducción y en los objetivos usaremos las estructuras de datos dinámicas pues su versatilidad en el manejo de datos nos ayudará a tener una óptima solución en el manejo de los programas.

Para las simulaciones del estacionamiento y el verdugo usaremos la estructura de cola pues es la más adecuada para resolver este problema; para los métodos de ordenamiento usaremos las Listas Simples por su facilidad y versatilidad; para los métodos de búsqueda usaremos las Listas simples y los métodos de ordenamiento; para la expresión prefija usaremos árboles y finalmente para Dijkstra usaremos grafos.

Desarrollo de la solución

Cola y Lista Simple

Las estructuras de datos fueron creadas a partir de otras estructuras de datos. La lista simple, como su nombre lo dice, es sencilla de entender. Es creada a partir de otra estructura llamada Nodo que almacena datos. Tenemos un conjunto de nodos señalándose entre sí, así que programados distintos métodos para hacer más optima la lista simple como: “agregar cabeza” la cual agregará un nodo al principio de la estructura, “agregar cola” el cual agregará un nodo al final de la estructura, también los métodos “eliminar en cabeza” y “eliminar en cola, etc.

Posteriormente usando la clase de la Lista Simple haremos Cola. La cola es una estructura de datos en la cual primer dato en entrar sería el primero al que tenemos acceso (First Input First Output), como una cola para el super.

Estacionamiento

Creamos una clase llamada “Carro”, con el parámetro del lugar que le toca, la cual nos servirá en la siguiente clase a la que llamaremos “Estacionamiento”, crearemos un método para llamarlo cuando el usuario inicie la simulación. En el método tendremos una cola de tipo Carro y aun ArrayList de tipo Carro también, ahora lo que tenemos que hacer es meter un auto o sacarlo de manera aleatoria así que mediante una variable aleatoria vamos a tomar un valor, si nos da de 0 a 60 meteremos un carro al estacionamiento, si es de 61 a 90 sacaremos un auto. Si ingresamos el coche a la simulación crearemos un objeto de tipo coche con su respectivo ticket y lo meteremos a la cola y al ArrayList, esto debe mostrarse en pantalla en la aplicación. Si decide el programa sacar un auto primero revisaremos que la cola y el ArrayList no estén vacíos, si es así tomaremos un numero aleatorio de la cantidad de autos que están adentro del estacionamiento y sacaremos y meteremos a la cola hasta dar con ese número, una vez lo tengamos lo sacaremos y no lo volveremos a

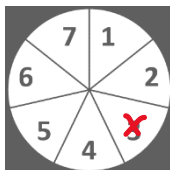
meter eliminándolo del estacionamiento y esto lo mostraremos en la aplicación. Todo este proceso lo haremos las veces que ha indicado el usuario al principio de la aplicación y al terminar este bucle mostraremos los autos que fueron quedando en la Cola sacándolos.

Verdugo

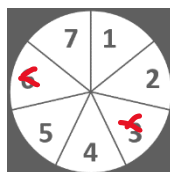
Creamos una Clase llamada: "Prisionero" que tendrá el campo de clase número el cual usaremos en la siguiente clase. Creamos la siguiente Clase en donde controlaremos lo que pase en la aplicación. Cuando el usuario le indique a la aplicación mediante un botón la aplicación creará una cola de tipo Prisionero y le asignaremos los datos que nos ha dado el usuario los cuales son en que posición se encuentra su amigo, cuantos reos hay en total y de cuantos en cuantos los irá matando. Creamos un objeto tipo prisionero con el número de reo que es y lo insertamos en la cola, esto hasta que lleguemos a los reos que nos solicitó el usuario. Posteriormente los comenzaremos a matar hasta que quede uno vivo, sabremos esto creando una variable que tenga el mismo número de reos que nos solicitó el usuario y la iremos disminuyendo en medida que vayamos matando a los reos.

Para matar a los reos haremos una simulación hasta que quede solo uno vivo y empezaremos a contar desde el 1. Usando una variable que ira aumentando veremos si la variable es igual a el patrón que nos ha dado el usuario, de no ser así significa que no es el reo que debemos matar así que lo sacamos y lo volvemos a meter a la cola, si se iguala significa que debemos matar a ese reo así que solo lo sacamos de la cola, igualamos la variable con la que nos estamos auxiliando del patrón a 0 pues si no el bucle se rompe y disminuimos los reos vivos.

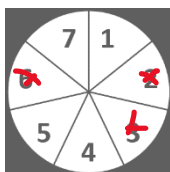
La simulación quedaría así tomando como ejemplo que sean 7 reos, nuestro amigo sea el 5 y los matemos de 3 en 3:



Tomando en cuenta que iniciamos en 1 y matamos de 3 en 3 el primero en morir sería el 3



Continuando, iniciamos a contar desde el 4 y el siguiente en morir es el 6



La simulación nos indica que el ultimo en morir será el reo 4, pero nuestro amigo es el 5, entonces debemos hacer una operación

sencilla. Tenemos tres posibles escenarios, que el sobreviviente sea exactamente nuestro amigo, en este caso no haremos nada ya que es el resultado que buscamos; que el sobreviviente sea un número menor que nuestro amigo, como en el caso del ejemplo, de ser así, recorreremos el numero donde iniciamos para que "mágicamente" salvemos a

nuestro amigo, esto lo haremos restando la posición de nuestro amigo menos la posición del último superviviente, dándonos así la diferencia entre el superviviente y nuestro amigo, esta diferencia la sumaremos a 1 ya que en esa posición iniciamos; y finalmente tenemos el último caso, que el número de lugar del superviviente sea mayor que nuestro amigo, si pasa esto lo que haremos va a ser restar el lugar del superviviente menos el de nuestro amigo, esto nos dará la diferencia para atrás que existe entre el superviviente y el amigo, después eso se lo restaremos a el máximo de supervivientes y le sumaremos uno.

Finalmente, en la aplicación mostraremos en texto el resultado obtenido: el número del amigo y desde donde debemos empezar a matar (resultado que obtenemos en las operaciones de los 3 casos anteriores y finalmente el patrón de muerte).

Ordenamiento: Burbuja

Para los métodos de ordenamiento creamos una clase llamada datos con los atributos: nombre, edad, peso y altura. Posteriormente creamos el archivo que tendrá los datos que usaremos en el programa, el archivo estará estructurado de la siguiente manera:

Nombre;Edad;Altura;Peso

Después en una lista simple de tipo Datos leeremos el archivo por líneas y cada vez que encontremos un “;” tomaremos lo siguiente en tokens y lo meteremos en un objeto asignándole cada token a lo que le corresponde (el nombre, edad, altura y peso), continuando meteremos este objeto a la lista.

Ahora que tenemos la lista con los datos lo siguiente es crear la gráfica de barras, mediante la herramienta de JavaFX creamos dicho gráfico y mediante un bucle for recorremos la lista, obtenemos la edad de cada Nodo en la lista y la transformamos en una barra de color aleatorio entre rojo, azul, amarillo y verde, dicha barra la agregamos en el gráfico.

Para sencillez en el programa lo que haremos será hacer una única aplicación y mediante 5 botones elegiremos con que método ordenar los gráficos.

Para el ordenamiento de tipo burbuja creamos un método el cual será llamado cuando el usuario presione el botón de “Burbuja”. El método hará ejecuciones por lapsos lo que nos permitirá ver gráficamente como se van ordenando las barras del gráfico. Sigue la parte de el ordenamiento, creamos dos bucles que recorran la lista, una iniciará desde 0 siendo i hasta uno menos de la longitud de la lista y la otra iniciara uno después de la primera siendo j hasta finalizar la lista, la lógica de este método de ordenamiento es que recorrerá la lista buscando en las edades, si encuentra un dato que su edad sea menor a la de i, intercambiará estos datos, las barras en el gráfico y continuará buscando uno menor hasta que termine en la lista, el problema de este método es que recorre la longitud del arreglo muchas veces y es demasiado lento.

Ordenamiento: Selección

Ya tenemos la lista hecha (en el anterior método de ordenamiento se explica cómo) ahora implementaremos mediante un método en el programa esta forma de ordenar datos.

Nuevamente llamamos una función que lo haga funcionar por lapsos de 1 segundo y empezamos a programar el método de ordenamiento. Es uno de los más sencillos ya que busca directamente los mas pequeños y los va acomodando en su lugar. Creamos una

variable temporal y le damos el valor de i en el primer bucle for, en el segundo buscaremos un dato en edad que sea menor que la variable temporal, una vez lo encontramos cambiamos el valor temporal por j , una vez cambiado el valor cambiamos los valores de el nodo en posición i y el nodo en la posición de la variable temporal, también cambiamos las barras en los gráficos.

Ordenamiento: Inserción

Este método consiste en tomar los dos primeros elementos del arreglo y ordenarlos entre ellos, después tomar el que sigue y ordenarlo en el lugar correspondiente entre este dato y los otros 2 que ya hemos tomado y así consecutivamente. Lo que haremos será crear dos variables de tipo String donde almacenaremos los datos del estilo de la barra de cada nodo que usemos, y empezamos el primer bucle donde tomaremos el primer elemento y comenzaremos otro bucle donde tomaremos otro elemento, después compararemos los datos de dichos elementos y los colocaremos en su respectivo lugar (como explicamos al inicio de este método) y les asignaremos los datos de las barras para cambiar también de manera visual; posteriormente continuaremos intercambiando hasta terminar la lista.

Ordenamiento: Mezcla

Este algoritmo es bastante sencillo de comprender, en palabras simples el método de ordenamiento consiste en partir lo que debemos ordenar (en este caso la lista) y ordenar una parte y ordenar la otra, después juntar ambas partes y ordenar, posee un principio de divide y vencerás, es más fácil que 10 personas ordenen un grupo de datos que solamente 1. Tomamos la misma función que hace que el algoritmo funcione por bucles de 1 segundo y creamos una variable que sea igual a la mitad de la longitud de la lista y una bandera, ahora usaremos un método de los anteriores que hemos programado, en este caso usaremos el método de burbuja. Creamos dos bucles que recorran desde 0 hasta la mitad de la lista y en ellos tomamos los datos i y j , los comparamos e intercambiamos, esto hasta encontrar al más pequeño. Creamos otros dos bucles que recorran desde la mitad de la lista hasta el final de la lista y hacemos lo mismo, ordenamos; ahora tenemos la lista ordenada de ambos extremos, solamente falta ordenarlos entre ellos, así que una vez haya terminado de ordenar el último número de la lista marcaremos la bandera indicándole al método que puede comenzar a ordenar entre ambas partes.



Ordenar ambos extremos y ordenar entre ellos

Ordenamiento: Quicksort

El método de Quicksort funciona de manera parecida al de mezcla. El método consiste en tomar un pivote de el arreglo (la lista en este caso), depende del pivote que elijamos será más o menos eficiente este método pero para efectos prácticos tomaremos siempre el dato que se encuentre hasta al final como pivote y tendremos otra variable que llamaremos marca inicializada en -1, ahora comenzaremos un bucle que recorrerá una sola vez la lista, en el bucle buscaremos un dato (la edad) que sea mayor a nuestro pivote y si lo encontraremos, avanzaremos nuestra variable “marca” un espacio y cambiaremos este dato que se encuentre en la posición marca con el dato que es más grande que nuestro pivote, al finalizar el bucle cambiaremos de posición el pivote con la marca mas 1, de esta forma el pivote quedará en el lugar que debe de estar y los datos que estén antes que este serán menores que el pivote y los que estén posteriores al pivote será mayores, siendo esta lógica solamente nos restaría ordenar estos datos con un método de ordenamiento ya programado. Tomamos el código del método de burbuja y lo operaremos hasta uno antes de el pivote, de esta manera ordenaremos e intercambiaremos los datos menores a el pivote y hacemos lo mismo para los elementos de la lista, desde el pivote hasta finalizar la lista. Terminado esto la gráfica de barras se ha ordenado de manera óptima.

Búsqueda: Secuencial

El siguiente par de códigos consisten en métodos de búsqueda, donde utilizaremos el mismo documento que usamos con los métodos de ordenamiento, el cual contiene datos de personas y esta estructurado de la siguiente manera:

Nombre;Edad;Altura;Peso

Para entender la magnitud y la posibilidad de estos métodos nuevamente usaremos la herramienta de JavaFX y aplicaremos un gráfico de barras para poder ver visualmente que está haciendo el código.






Creamos una clase llamada: “Datos” en donde pondremos de atributos: “Nombre, edad, peso y altura” e implementemos los métodos get y set para poder trabajar con los atributos al momento de crear los constructores. Posteriormente comenzaremos a trabajar con el documento, al igual que el de ordenamiento creamos un método que lea el archivo y lea la cadena por párrafos rompiéndola en tokens cada que encuentre un “;” usando un split, después creamos un constructor y agregamos cada token a su correspondido atributo y este objeto lo metemos en la lista, esto lo haremos hasta terminar el documento. Continuando crearemos el gráfico usando JavaFX y mediante un bucle haremos las gráficas dándoles el valor de los datos de la lista (usaremos nuevamente edad para fines prácticos) y un color aleatorio entre rojo, azul, verde y amarillo. Ordenamos los datos usando un método de ordenamiento programado anteriormente, en el proyecto usamos burbuja.

Finalmente toca crear el método de la búsqueda secuencial. Es el método más sencillo pues es lineal, recorre la lista y compara los datos hasta encontrar el que le hemos indicado, así que creamos un bucle for y recorremos la lista, una vez encuentre un dato que sea exactamente igual al que le ha indicado el usuario cambiamos de color la barra de este dato a negro para indicarle que lo hemos encontrado.

Búsqueda: Binario

La búsqueda binaria funciona de la siguiente manera:

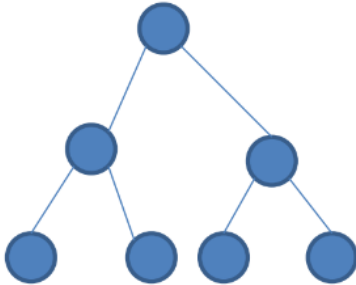
Primero ordenamos los datos, pero estos ya son previamente ordenados al inicio del programa.

Tenemos nuestro conjunto de datos en forma de lista,  partimos dicha cantidad en 2 para que nos queden dos extremos, así que tomamos el valor que queda en la mitad y comprobamos si el valor que se encuentra en la mitad no es el que nos piden, de no ser así revisamos si es más grande, si es más grande el valor de la mitad eso significa que el  valor que buscamos se encuentra en algún lugar de la primera mitad así que lo haremos será obtener el valor entre el inicio de la lista y la mitad (este valor será nuestra nueva mitad), esto lo conseguiremos restando el inicio menos el medio, dividirlo entre dos y sumarle el principio, y finalmente le asignamos al final el antiguo valor de tmp. De esta manera ahora tenemos dos  extremos posiblemente este nuestro valor así que hay que repetir este bucle de búsqueda hasta que lo encuentre, pero hasta ahora tenemos solo en caso que sea menor que la mitad, supongamos que ahora el valor que buscamos es mayor que nuestro nuevo tmp, aplicaremos algo parecido matemáticamente a lo que aplicamos anteriormente, guardamos temporalmente el valor de la  mitad y el nuevo valor de mitad será la resta entre el final y la mitad, lo dividimos entre dos y le sumamos la mitad, de esta manera el segmento que se ve ilustrado de color verde en el gráfico anterior esta partido en dos y podemos volver a hacer los métodos anteriores hasta encontrar el número  que nos ha indicado el usuario. Este método tiene ciertas desventajas y ventajas, pero podemos solucionar las desventajas aplicando las ventajas.

Una de las principales desventajas de este código es que el usuario puede intentar buscar un valor que no se encuentre en la lista y esto puede ser catastrófico pues el código se quedaría en bucle infinito buscando un dato que jamás encontrará, pero con la ventaja de este método solucionaremos este detalle. A diferencia del secuencial, la búsqueda binaria no tiene que recorrer todos los datos, en cuestión de posiblemente 5 entradas a el bucle puede encontrar el dato, tomemos a la ultima gráfica anteriormente mostrada, supongamos que representa 100 valores, en 3 entradas al bucle solamente tiene que evaluar a dígitos entre 25 – 50, la capacidad de este método radica que en solo 3 entradas al bucle descartó a 75 números, es así como quizá en 5 o 6 iteraciones podría encontrar al número indicado, pero ¿Cómo solucionamos el hecho que no lo encuentre y se vaya a un bucle infinito? Sencillo, solo creamos una variable que vaya contando las veces que entra al bucle, una vez que este contador pase de 20 significa que no ha encontrado el dato y es posible que se vaya a un bucle infinito así que detenemos el ciclo for y le indicamos al usuario que el número que busca no se encuentra en la lista.

Arboles Binarios

Las listas simples tienen un problema y es que están limitadas a una sola dimensión. Para superar estas limitaciones usaremos una nueva estructura llamada árbol que se compone de nodos y aristas.



Los árboles como en las listas simples utilizan nodos para guardar información y las aristas hacen referencia a estos nodos. La diferencia con las listas simples es que estas tienden a hacer referencia a sus nodos secuencialmente, es decir, solamente puede haber un nodo referenciado a otro, en cambio los árboles pueden referenciarse a 2 nodos. El nodo principal es llamado raíz, es el primer nodo, de este nodo parten 2, sus hijos, estos son llamados hojas y de estas hojas pueden partir más hijos.

Para empezar a programar crearemos una clase llamada: "NodoArbol", esta clase se encargará de crear los nodos que usaremos para almacenar los datos y hacer referencia a otros nodos, agregamos los atributos dato de tipo E, e izquierda y derecha de tipo NodoArbol, creamos los constructores y dos métodos que se encarguen de comparar los datos que obtendremos en el constructor para ver cual es mayor y así designarlo al nodo izquierdo y el nodo derecho. Posteriormente crearemos la clase "ArbolBinarioOrden", esta clase no servirá para ordenar adecuadamente nuestro árbol binario y para leerlo correctamente. Creamos los métodos para poder insertar un dato en nuestro árbol y vamos a usar la función para ver si es mayor que un dato o menor, el mayor se va a la derecha y el menor a la izquierda y si no existe raíz en el árbol se la agregamos, después creamos los métodos que nos servirán para recorrer el arreglo de forma post orden, pre orden y de orden normal, además del el método para eliminar una hoja.

Finalmente creamos la clase para ejecutar el programa en donde crearemos dos pilas de tipo NodoArbol, tomaremos lo que nos digitó el usuario, lo partimos en tokens, posteriormente vemos si no es un operador, de ser así lo vamos a insertar a una pila, si sí son operadores vamos a insertar los operadores en otra pila, una vez vacío lo que nos digitó el usuario acomodaremos los nodos de las respectivas hojas para formar así el árbol. Finalmente evaluamos la expresión recorriendo los nodos obteniendo el nodo izquierdo, el derecho y operándolos.

Conclusión

Hemos resuelto los problemas planteados usando las Estructuras de Datos aprendidas en el curso.

Como mencionamos al inicio del proyecto las Estructuras de Datos dinámicas nos ayudan a resolver problemáticas que normalmente no se podrían con las estáticas y es gracias a esta versatilidad sumado a los conocimientos y el dominio de estas, además de las nuevas herramientas aprendidas, que pudimos resolver problemas como el problema del verdugo, estacionamiento, incluso de los métodos de ordenamiento y los métodos de búsqueda, ya que su capacidad de aumentarse y disminuirse, de poder meter cualquier tipo de dato además de la fácil compresión y manejo de estas hace mas sencillo crear programas con un nivel de dificultad mayor.

Las estructuras dinámicas nos han permitido resolver los problemas anteriores con versatilidad, sencillez y sin destruir a la computadora en el proceso (pues las estructuras dinámicas trabajan bastante bien con la memoria RAM, ya que no exige un bloque de slot unitario y puede trabajar con varios pequeños libres por la memoria) permitiéndonos así trabajar con 100 datos de manera rápida, inmediata y eficiente.

En este proyecto se intentaron aplicar los conocimientos de Árboles Binarios y Grafos de manera óptima, sin embargo, no se llegó a este punto puesto a que se presentaron inconvenientes durante el desarrollo de los últimos dos programas, tales como problemas en la programación y comprensión del código que entorpecieron su funcionamiento y desarrollo,

Bibliografía

Cairo, O. y Guardati, S. (2006). Estructura de datos. McGraw Hill

Deitel, P. y Deitel, H. (2008). Como programar en Java. Pearson Educación

Joyanes, L. y Zahonero, I. (2008). Estructura de datos en Java. McGraw Hill