Tecnológico de Monterrey

Actividad 1.3
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

10/09/2021

Investigación

Los algoritmos de ordenamiento se encargan de, como su nombre indica, ordenar información basándose en un sistema numérico, es decir, se ordena dependiendo del tamaño de los datos. Normalmente se utiliza para ordenar números de menor a mayor, lo cual puede sernos útil al momento de organizar cualquier tipo de información de forma rápida y eficiente. Estas herramientas son convenientes para el humano pues este tipo de tareas son indispensables a la hora de almacenar, analizar y representar datos.

Existen distintos tipos de algoritmos de ordenamiento, cada uno con diferentes métodos y complejidad computacional (tiempo que tarda un algoritmo en completar una tarea), por lo que existirán algoritmos no óptimos por el tiempo que tardan en ordenar un gran número de datos, pero la elección de cuál usar dependerá totalmente de las necesidades del usuario.

También existen los algoritmos de búsqueda, que se encargan de buscar un dato de entre un conjunto de estos. Un ejemplo de este tipo de algoritmos es el de un buscador web, como lo puede ser Google, Yahoo, entre otros, el cual busca en su base de datos páginas que cuenten con las palabras claves que el usuario entregue al buscador.

De igual manera que los algoritmos de ordenamiento, diferentes algoritmos de búsqueda pueden tener diferentes complejidades computacionales y su uso dependerá completamente de las necesidades del usuario. Se sabe que los algoritmos de búsqueda implementados en diversos lugares están siendo optimizados constantemente, ya que pueden entregar millones de resultados en cuestión de segundos.

Documentación del código

Para la resolución de esta actividad utilizamos el algoritmo de ordenamiento conocido como Mergesort, el cual es un método recursivo que toma inspiración del refrán “divide y vencerás”. Este algoritmo se basa en dividir continuamente la lista a ordenar por la mitad, hasta que solo tengamos sublistas de tamaño 1. Después, unimos las sublistas de 2 en 2 de forma ordenada. Esto nos devuelve un arreglo ordenado en un tiempo O(nlogn), el cual es el menor tiempo posible para ordenar un arreglo.

También utilizamos un algoritmo de búsqueda secuencial que, no es el más rápido, pero se acomoda a nuestras necesidades. Este algoritmo se basa en cómo una persona común buscaría un dato en una lista, revisando cada elemento y comparándolo con el dato que queremos encontrar. Dicho algoritmo nos devuelve la posición del dato en una lista con una complejidad espacial O(n), lo cual es bastante aceptable para nuestros términos. La razón por la cual no usamos un algoritmo más rápido, por ejemplo el binario, es porque en nuestra lista hay datos repetidos. Si lo hiciéramos de forma binaria, el algoritmo nos pudiera devolver posiciones en las cuales, si bien estarían el dato que estamos buscando, pudiera ser el caso en donde antes de la posición devuelta, siga habiendo datos que nos interesen.

También creamos una estructura la cual nombramos como Entrada, la cual permite almacenar todos los datos necesarios, IP, razon, fecha, etc, además de una función para imprimir los datos y otra la cual permite crear la fecha como un número entero, donde el mes serán los primeros dos números y el dia los otros dos. En el main se crea un vector con todos los datos de la bitácora, después se ordenan en otro vector y después de preguntar al usuario de que fecha a que fecha se quieren los datos, se busca la información que esté entre esas fechas y se imprimen, a la vez de que se almacenan en un archivo txt.

Nosotros decidimos ir un poco más allá de los requerimientos del problema de ordenamiento, no sólo nos enfocamos en el mes y el día, sino que también ordenamos los datos tomando en cuenta la hora de cada una de las entradas del archivo de caso prueba. Para ello hicimos uso de objetos de la librería stringstream para obtener solamente los caracteres numéricos, concatenarlos, y transformarlos a un entero; así obtenemos un valor adicional para realizar el ordenamiento. Usamos el mes mult aplicándolo por 100 y le sumamos el día; para complementar nuestros datos de tiempo le concatenamos éste nuevo dato numérico dado por la hora. Con ésto obtenemos una medida para poder realizar el ordenamiento considerando todos los datos de tiempo disponibles en el caso prueba.

Análisis de complejidad Merge Sort:

```
void unir(vector<Entrada> &arr,int inicio, int mitad, int fin){
    int i,j,k;
    i = inicio;
    j = mitad + 1;
    k = 0;
    vector<Entrada> aux(fin - inicio + 1);

    while(i <= mitad && j <= fin){
        if(arr[i].fecha < arr[j].fecha){
            aux[k] = arr[i];
            i++;
        } else {
            aux[k] = arr[j];
            j++;
        }
        k++;
    }

    while(i <= mitad){
        aux[k] = arr[i];
        i++;
        k++;
    }

    while(j <= fin){
        aux[k] = arr[j];
        j++;
        k++;
    }

    for(int i = inicio; i <= fin; i++){
        arr[i] = aux[i - inicio];
    }
}
```

Θ(1)

Θ(n)

Θ(n)

Θ(n)

Θ(n)

```
void mergeSort(vector<Entrada> &arr, int inicio, int fin){
   int mitad;
   if(inicio < fin){
      mitad = (inicio + fin) / 2;
      mergeSort(arr,inicio,mitad);
      mergeSort(arr,mitad + 1,fin);
      unir(arr,inicio,mitad,fin);
   }
}
```

Θ(1)
Θ(1)
Θ(1)
T(n/2)
T(n/2)
Θ(n)

$$T(n)\begin{cases}1, & n=1 \\ 2T(\frac{n}{2})+n, & n=2^k\end{cases}$$

$$T(n)=2T(\tfrac{n}{2})+n$$
$$T(n)=2(2T(\tfrac{n}{2^2})+\tfrac{n}{2})+n$$
$$T(n)=2^2T(\tfrac{n}{2^2})+2n$$
$$T(n)=2^2(2T(\tfrac{n}{2^3})+2\tfrac{n}{2^2})+2n$$
$$T(n)=2^3T(\tfrac{n}{2^3})+3n$$
$$T(n)=2^kT(\tfrac{n}{2^k})+kn$$
$$T(n)=2^{\log_2 n}T(\tfrac{n}{2^{\log_2 n}})+n\log_2 n$$
$$T(n)=nT(1)+n\log_2 n=n+n\log_2 n$$
$$T(n)=O(n)+O(n\log_2 n)$$
$$T(n)=O(n\log_2 n)$$

$\frac{n}{2^k}=1$

$n=2^k$

$k=\log_2 n$

Ecuación de recurrencia: $T(n) = O(n\log_2 n)$

Análisis de Complejidad Búsqueda Secuencial:

```
//Busqueda Secuencial
int busqSecuencial(vector<Entrada> &arr, int dato){
    for(int i = 0; i < arr.size(); i++){
        if(arr[i].fecha >= dato){
            return i;
        }
    }
    return -1;
}
```

Ésta es la función de búsqueda secuencial que implementamos en nuestro programa, donde le pasamos como parámetro un vector de Entradas y el dato que buscamos, en éste caso es una fecha en el formato propuesto en la sección de documentación del código. Tomando en cuenta que el vector está ordenado, si logramos encontrar el índice de la primera Entrada con la fecha que buscamos, sabremos que desde ése índice tenemos los datos que el usuario requiere, más adelante pondremos la implementación en el main con el dato de la fecha final.
En tanto a la complejidad computacional, ésta se puede deducir gracias a la inclusión de un sólo for que está en función del tamaño del vector, por lo que se concluye que la complejidad es de O(n).

Implementación de Ordenamiento y Búsqueda en Main

Para poder realizar el ordenamiento de manera adecuada debimos utilizar distintos recursos ya previamente mencionados, ésta sección va dedicada a puntualizar éstos y mostrarlos cómo se implementaron en el código del main.

Para poder realizar ésta tarea requerimos de 3 factores principales, un método específico dentro de la struct Entrada para crear la fecha con el formato adecuado, el uso del mergesort en base a éstos datos procesados, y la búsqueda a partir de los datos ingresados por el usuario y el uso de la búsqueda secuencial.

El primer elemento es la conversión de los datos de una forma en que puedan ser ordenados. Éste proceso se encuentra como un método de la struct Entrada.

```
void Entrada::setFecha(string m, int d){
    string meses[5] = {"Jun","Jul","Aug","Sep","Oct"};
    for(int i = 0; i < 5; i++){
        if(meses[i] == m){
            mes = i + 6;
        }
    }
    int pfecha = mes * 100 + d;
    stringstream sM,sD,sF;
    sM << m; sD << d;
    sMes = sM.str();
    sDia = sD.str();
    sF << pfecha << hora[0] << hora[1] << hora[3] << hora[4] << hora[6] << hora[7];
    fecha = stoi(sF.str());
}
```

Es posible notar cómo es que le asignamos un número al mes a partir del dato que se encuentra en el archivo de datos. Se multiplica éste por 100 y se le suma el día. Después se crea el objeto stringstream para realizarle una serie de push con los datos adecuados de la fecha, que están en tipo string. Por último convertimos ese objeto en string, para después convertirlo en entero y así poder ordenar las instancias.

El segundo factor es la generación de las instancias y el ordenamiento.

```
// Ciclo while para leer todos los datos dentro del archivo txt
while(!bitacora.eof()){
    bitacora >> tMes; // Lectura y asigancion de valores a variables
    bitacora >> tDia;
    bitacora >> tHora;
    bitacora >> tIP;
    getline(bitacora,tRazon); // Lectura de toda la razon

    entrada.hora = tHora; // Asignaciones de valores a la struct
    entrada.setFecha(tMes,tDia);
    entrada.IP = tIP;
    entrada.razon = tRazon;
    entradas.push_back(entrada); // Agregar al final cada instancia nueva
}

mergeSort(entradas,0,entradas.size()-1); // Ordenamiento por Merge Sort
```

Primero se leen los datos y se llama el método se setFecha para realizar la operación antes descrita. Para finalizar se aplica el mergesort, que cómo visto anteriormente, hace el ordenamiento en base a las fechas.

El último factor es la búsqueda secuencial en base a las fechas ingresadas por el usuario.

```
// Obtencion de informacion del usuario
cout << endl << "INICIO DE BUSQUEDA" << endl;
cout << "Mes: ";
cin >> mesInicio;
cout << "Dia: ";
cin >> diaInicio;
fechaInicial = (mesInicio * 100 + diaInicio) * 1000000;

cout << endl << "FINAL DE BUSQUEDA" << endl;
cout << "Mes: ";
cin >> mesFinal;
cout << "Dia: ";
cin >> diaFinal;
fechaFinal = (mesFinal * 100 + diaFinal) * 1000000 + 240000;
```

Primero se obtienen los datos del usuario y se manipulan de tal manera que concuerden con el formato manejado por fecha dentro de las instancias de Entrada generadas.

```
//Buscamos el primer dato
int i = busqSecuencial(entradas,fechaInicial);
// Ciclo de busqueda secuencial para obtener todos los datos con las especificaciones
for(i; i < entradas.size(); i++){
    if(entradas[i].fecha >= fechaInicial && entradas[i].fecha <= fechaFinal){
        // Almacenar los datos en un archivo
        bitacoraOrdenada << entradas[i].sMes << " ";
        bitacoraOrdenada << entradas[i].sDia << " ";
        bitacoraOrdenada << entradas[i].hora << " ";
        bitacoraOrdenada << entradas[i].IP;
        bitacoraOrdenada << entradas[i].razon << endl;
        //Imprimimos a consola para verificar los resultados
        entradas[i].imprimir();
    }
    else if(entradas[i].fecha > fechaFinal){
        break;
    }
}
```

Ahora se llama a la función de búsqueda secuencial para obtener el índice con el cuál empezar a almacenar y mostrar datos al usuario. Dentro del ciclo for corroboramos que la fecha se encuentre entre los rangos, cuando supere a la fecha final, el ciclo termina con un break. El programa termina cerrando ambos archivos, los casos prueba y el ordenamiento requerido por el usuario.

Reflexión

Los algoritmos de ordenamiento pueden facilitar tareas que para un ser humano sería muy tardado o simplemente imposible por la complejidad de la tarea, por lo que es esencial tener una forma de hacer esa clase de tareas en solo unos segundos, aunque también hay que tomar en cuenta el tipo de ordenamiento ya que algunos podrían tardar incluso años en terminar la organización de una tarea simple, por lo que hay que tomar en cuenta la cantidad de datos a analizar y la complejidad y recursividad del método a utilizar.

Referencias

LWH (s.f.) *Algoritmos de ordenamiento.* Consultado el 08 de septiembre del 2021, en: http://lwh.free.fr/pages/algo/tri/tri_es.htm

Alfaro T (s. f.) *Algoritmos de búsqueda y ordenamiento.* Universidad Técnica Federico Santa Maria. Consultado el 08 de septiembre del 2021, en: https://www.inf.utfsm.cl/~noell/IWI-131-p1/Tema8b.pdf

Aveiro C. (2017) *¿Qué son los algoritmos de búsqueda?* Consultado el 08 de septiembre del 2021, en: https://aveiroperoni.com/que-son-los-algoritmos-de-busqueda/

Runestone Academy (s.f) Ordenamiento por mezcla. Consultado el 08 de septiembre del 2021, en: https://runestone.academy/runestone/static/pythoned/SortSearch/ElOrdenamientoPorMezcla.html