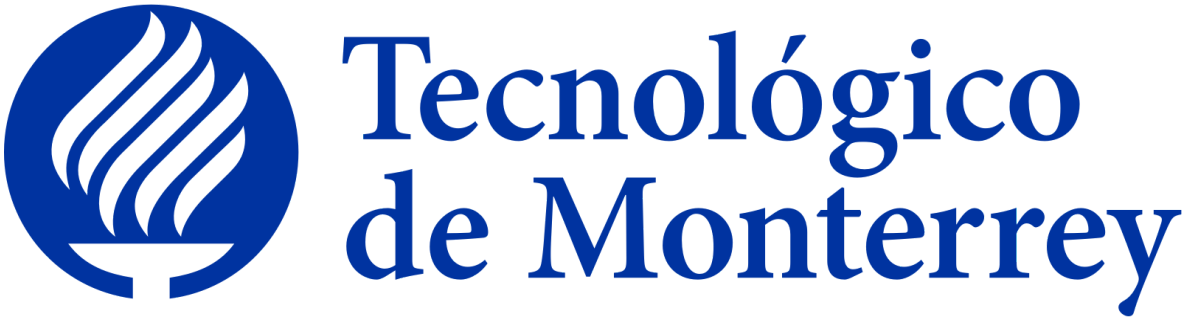
Instituto Tecnológico y de Estudios Superiores de Monterrey



Programación de Estructuras de Datos y Algoritmos Fundamentales

Ivan Reyez Amezcua

Actividad 1.3 Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales

Santos Alejandro Arellano Olarte // A01643742
Carlos Iván Armenta Naranjo // A01643070
Arturo Ramos Martínez //A01643269
Adair Virgilio Figueroa Medina //A00572826
Leonardo Mario Alberto Guillén Soria // A00574110

23 de Agosto del 2023

Para esta tarea se solicitó la creación un programa en c++ con las siguientes características:

1. Abra el archivo de entrada llamado “bitacora.txt” léalo y almacene los datos en un vector (Formato del archivo bitacora.pdf).
2. Ordene la información por fecha para la realización de las búsquedas.
3. Solicite al usuario las fechas de inicio y fin de búsqueda de información.
4. Despliegue los registros correspondientes a esas fechas.
5. Almacenar en un archivo el resultado del ordenamiento

Para lo cual desarrollamos el siguiente programa con X funciones en c++:

```
void ordenaMerge(vector<int>&a,int low,int high){
    if(low<high){
        int mid=low+(high-low)/2;
        ordenaMerge(a,low,mid);
        ordenaMerge(a,mid+1,high);
        vector<int> res(high+1);
        int i=low;
        int j=mid+1;
        int k=0;
        while(i<=mid&&j<=high){
            if(a[i]<=a[j]){
                res[k]=a[i];
                i+=1;
            }else{
                res[k]=a[j];
                j+=1;
            }
            k+=1;
        }
        while(i<=mid){
            res[k]=a[i];
            i+=1;
            k+=1;
        }
        while(j<=high){
            res[k]=a[j];
            j+=1;
            k+=1;
        }
        for(int h=0;h<k;h++){
            a[low+h]=res[h];
        }
    }
}
```

```
void leerAlmacenarOrdenar(string l, string r){
    ifstream bitacora;
    bitacora.open("fechas.txt", ios::in);
    if(bitacora.fail()){
        cout<< "Error al abrir el archivo" <<endl;
    }else{
        unordered_map<int, string> infoMap;
        string line;
        vector<int> Fechas_desordenadas;
        string mes, dia, hora, ip, fallo;
        while(getline(bitacora, line)){
            vector<string> Datos_sep;
            istringstream iss(line);
            iss >> mes >> dia >> hora >> ip;
            getline(iss, fallo);
            if(mes == "Jun"){
                mes = "6";
            }
            else if(mes == "Jul"){
                mes = "7";
            }
            else if(mes == "Aug"){
                mes = "8";
            }
            else if(mes == "Sep"){
                mes = "9";
            }
            else if(mes == "Oct"){
                mes = "10";
            }
            string fecha = mes+" "+dia+" "+hora;
            int key = formateaFecha(fecha);
            infoMap[key] = (line);
            Fechas_desordenadas.push_back(key);
        }
        ordenaMerge(Fechas_desordenadas, 0,
Fechas_desordenadas.size()-1);
        int fecha_inicio_int = formateaFecha(l);
        int fecha_final_int = formateaFecha(r);
        vector<string> rango_fechas;
        for(int i : Fechas_desordenadas){
            if(i > fecha_inicio_int && i< fecha_final_int){
            int key = i;
```

```
            auto it = infoMap.find(key);
            rango_fechas.push_back(it->second);
            cout<<i<<": "<<it->second<<endl;
        }
    }
        ofstream archivo("rangofechas.txt");
            if (archivo.is_open()) {
                for(string i : rango_fechas){
                    archivo <<i<<endl;
                }
                archivo.close();
            }else{
                cout<<"Error"<<endl;
            }
    bitacora.close();
}
}

int formateaFecha(string fecha){

    istringstream c(fecha);

    string mesS, diaS, horaS, minS, secS;

    c >> mesS >> diaS;

    getline(c, horaS, ':');

    getline(c, minS, ':');

    getline(c, secS);

    int mes_int = stoi(mesS);

    int dia_int = stoi(diaS);

    int hora_int = stoi(horaS);

    int min_int = stoi(minS);

    int sec_int = stoi(secS);

    return (mes_int * 100000000) + (dia_int * 1000000) + (hora_int *
10000) + (min_int * 100) + sec_int;

}
```

ordenaMerge()	Descripción	Este método de forma recursiva, usa una función que a grandes rasgos va dividiendo el arreglo hasta llegar al caso base, el cual es cuando el arreglo consta de un solo elemento, en ese momento esos valores, van ascendiendo hacia las funciones recursivas previas para ser comparados entre sí e ir ordenando cada pedazo de arreglo. En resumen, cada parte que fue separada, va siendo ordenada después de haber llegado al caso base, esto hasta llegar a la primera llamada de la función, la cual ordena las dos primeras mitades del arreglo. Es una operación comúnmente utilizada en algoritmos de ordenamiento y en la manipulación de contenedores como vectores, listas y colas de prioridad (heap).
	Entrada	Un vector con los datos enteros, un índice inicial (low) y uno final(high) correspondientes al primer y último índice del vector.
	Salida	La salida es el mismo vector, pero ya ordenado

	Precondición	El vector debe contener los datos a ordenar y estos deben ser de tipo entero
	Postcondición	El vector al final tendrá los datos ya ordenados
	Complejidad	La complejidad de esta función se representa como una función logarítmica de base 2 al ir dividiendo el arreglo en mitades e ir comparando las magnitudes: $T(n)=\log_{2(}n)$
leerAlmacenarOrdenar()	Descripción	Esta es una función que dentro de ella lee las líneas de texto de un archivo llamado fechas.txt, para después almacenarlas en un vector de manera desordenada y posteriormente ordenarlas convirtiendo su fecha y hora a números con ayuda de la función formateaFecha(), para finalmente guardarlos en otro vector con los datos ordenados por fecha y hora para posteriormente estos datos ya ordenados colocarlos en un archivo de texto llamado rangofechas.txt en donde se puedan almacenar según el rango que indico el usuario, el usuario

		puede hacer búsquedas en el archivo de registro en base al día, mes y hora que quiere que estén los registros que desea buscar.
	Entrada	La fecha de inicio y la fecha final del rango en el que desea buscar.
	Salida	Esta función de salida imprime en pantalla la lista de datos ordenados por fecha y hora
	Precondición	El archivo que va a leer y ordenar se debe de llamar fechas.txt y debe de estar en la misma carpeta además debe de contar con el formato establecido en las instrucciones.
	Postcondición	El vector final debe tener los datos ordenados por fecha y deben ser almacenados correctamente en el archivo txt resultante conforme al rango que brindó el usuario
	Complejidad	Al ordenar los datos por fecha y hora con un while e imprimir las fechas con un for ya ordenadas por separado su complejidad se comporta de manera lineal: T(n)= an+b

formateaFecha()	Descripción	Esta función lo que hace es convertir un string que contiene una fecha a una clave única para esta fecha contando con el mes, el día, la hora, los minutos y segundos, juntando como clave única estos datos.
	Entrada	Recibe un string con los datos de la fecha y la hora
	Salida	Devuelve una clave única para la fecha como un número entero
	Precondición	El dato que le mandas como string debe tener este formato para la fecha y la hora “Oct 12 00:01:57”
	Postcondición	La clave generada será única para un dia, mes, hora, minuto y segundo específicos
	Complejidad	Al no utilizar ningún ciclo ni algún proceso que requiere mucha complejidad, esta se representa así: T(n)=0n+b por lo tanto la complejidad es constante T(n)=b

Después de haber analizado el programa, sus funciones y sus complejidades se nos solicitó realizar reflexiones individuales del tema:

Carlos Iván Armenta Naranjo:

La importancia del uso y estudio de algoritmos de búsqueda y ordenamiento en problemas o situaciones como esta es crucial ya que al trabajar con bases de datos muy grandes la diferencia entre el tiempo que tarda en leer, buscar u ordenar estos datos es crucial y se nota la diferencia entre un algoritmo más o menos eficiente y es aquí donde el estudio de la complejidad de los algoritmos entra en juego para ayudar a nuestro programa a ser más eficiente.

Arturo Ramos Martínez :

De acuerdo con la experiencia que tuve tratando de hacer esta actividad, puedo decir que la situación cambia mucho a la hora de tratar con distintas cantidades de datos, de ahí la importancia de tener conocimiento acerca de la complejidad de cada algoritmo. En resumen, comprendí mejor que, escoger de qué manera manejar los datos para hacer el proceso más eficiente, es de vital importancia a la hora de manejar grandes cantidades de datos para aprovechar al máximo el poder computacional.

Por otro lado, se que muchos de estos algoritmos son la base de procesos que día a día realizamos, por ejemplo al hacer búsquedas en internet, los algoritmos de búsqueda o lo que gestionan las bases de datos, son usados prácticamente a diario. Además, las nuevas tecnologías como al entrenar una Inteligencia Artificial, es muy importante manejar y brindarle inmensas cantidades de datos para su óptimo desarrollo y funcionamiento.

Adair Virgilio Figueroa Medina:

La eficiencia de los algoritmos de ordenamiento y búsqueda es fundamental en el manejo de bitácoras con registros cronológicos. Ordenar los datos mediante algoritmos como Mergesort() y Quicksort() permite una visualización y análisis más claro, siendo el primero adecuado para grandes volúmenes de datos. Para buscar información específica en bitácoras ordenadas, la Búsqueda Binaria destaca por su rapidez (\(O(\log n)\)), mientras que la Búsqueda Lineal es viable para conjuntos más pequeños o después de ordenar. Estos algoritmos aceleran la identificación de patrones y tendencias, mejorando la eficiencia del sistema. La elección acertada de algoritmos según el tamaño y la naturaleza de los datos es esencial para optimizar el manejo de bitácoras y facilitar el análisis de información relevante.

Leonardo Mario Alberto Guillén Soria:
Los algoritmos sin duda alguna son de mucha ayuda para la optimización de muchísimos procesos, aún con más razón cuanto estamos hablando de bases de datos gigantescas y necesitamos de un procesamiento rápido y efectivo. Teniendo desde el algoritmo más básico Sort, que logra organizar cualquier lista o base de datos iterativamente recorriendo elemento por elemento, el cual deja de ser efectivo

cuantos más datos existan debido a que va de uno por uno, a diferencia de otros más efectivos como Merge(), que va partiendo en mitades y organizando a la vez que checara las mitades, para verificar, buscar u organizar una base de datos de forma efectiva.

Sin duda alguna los algoritmos son de gran ayuda y una gran herramienta que agilizan procesos complicados.

Santos Alejandro Arellano Olarte:

En tareas como esta actividad, los algoritmos de búsqueda y ordenamiento son como atajos inteligentes. Ayudan a organizar y encontrar información rápidamente. Son como tener una asistente personal para gestionar datos de manera eficiente en programación. La función "ordena Merge()" organiza datos caóticos y "leerAlmacenar()" extrae y ordena información. Comprender su complejidad nos guía en elegir la mejor opción.