Juan Carlos Garfias Tovar, A016521238

Explicacion del Código

Este trabajo fungió como ejercicio para poner en practica el uso de listas enlazadas, en particular las listas enlazadas dobles, las cuales tienen un apuntador de izquierda y de derecha. En el caso particular del código entregado, estas también cuentan con una llave, con la cual se puede identificar la posición inicial en la que fueron insertadas y asi llevar un mejor control en caso de que asi se desee.

El código crea una lista en el main la cual tiene como data un struct de tipo RegistryEntry el cual contiene todos los datos de cada registro del txt para asi poder ser insertados de manera efectiva. Como primer paso, el programa toma el archivo, verifica su existencia y posteriormente comienza a leer línea por línea, insertando estos valores y asignándolos en un objeto temporal de tipo RegistryEntry el cual posteriormente es insertado en un nodo para ser colocado en la tail del linked list.

```
ifstream file("./bitacora.txt"); //abre el archivo
if(file.is_open()){
    string line;
    vector<string> words; //vector para almacenar palabras de entry
    vector<string> time;    //vector para el tiempo
    string errorString = "";    //string para el error

    int i =0;
    while(getline(file,line) /*&& i≤20*/){ //Se leen todos las entries por linea
        words = split(line," ");
        //print(words);
        RegistryEntry entry; //se crea entry y se hace split de elementos para llener el struct
        entry.month = words[0];
        entry.day = stoi(words[1]);
        time = split(words[2],":");
        entry.hour = stoi(time[0]);//
        entry.minute = stoi(time[1]);
        entry.second = stoi(time[2]);
        entry.ip = words[3];
        for (int i = 4; i < words.size(); i++){ //todos los elementos posteriores son parte del error de login
            if(i ≠4 && i ≠words.size()){
                errorString= errorString+ " " + words[i];
            }
            else{
                errorString+=words[i];
            };
        }

        entry.error = errorString;

        Node * n1 = new Node();
        n1 -> key = keyCount;
        n1 -> data = entry;
        entries.appendNode(n1);

        errorString = "";
        words.clear();
        time.clear();
        keyCount++;
        i++;
    }
    file.close();
}
```

En cuanto a los principales algoritmos se decidió ordenar los datos con un merge sort, esto ya que es mucho mas constante sin importar la cantidad de datos a insertar en el ordenamiento. Este se decidió realizar directamente en la double linked list con el fin de ahorrar memoria y evitar gastar recursos de manera innecesaria. Principalmente debido a que los pointers permiten hacer reasignación de los data originales, permitiendo asi ordenar los datos de manera rápida y sencilla sin recurrir a una tercera estructura de datos. Quedando asi con una complejidad O(nlogn).

En cuanto a la búsqueda se opto por usar un binary search, la cual resulta eficiente pero complicada de implementar en una linked list. Esto ya que a pesar de que hay apuntadores no existen índices per se, Esto hace que la búsqueda sea mas rudimentaria por lo que no resulta tan simple, sin embargo, la complejidad se mantiene como un O(log n). Esto ya que se puede implementar directamente el upper y lower limits para la query. La cual posteriormente es procesada como un string por medio de ofstream y finalmente insertada en un txt. Además de esto se decidió implementar unos elementos gráficos para la experiencia de usuario y para conocer datos como las entradas leídas, etc.