



Tecnológico de Monterrey

Documentación de código

Programming of data structures

Inés Alejandro Garcia Mosqueda A00834571
Jesus Fong Ruiz A01254062

Profesores: Luis Ricardo Peña Llamas
Jorge Gonzalez

26/10/2022

Para la entrega se necesita organizar una bitácora de negaciones de acceso de distintas Ip durante cierto periodo de tiempo. Además de organizar las ip de acuerdo con las negaciones de acceso hacia un servidor, que ha identificado en una bitácora

Para esto se especifica que la estructura de datos en la que se tiene que organizar la información de la bitácora tiene que ser un tipo de árbol binario y tener el registro de negaciones de accesos por cada Ip

Comenzando con la unidad básica para nuestra implementación del árbol binario, implementamos una clase llamada Node, la cual contiene toda la información de estas negaciones de acceso

```
8 //clase basica para almacenamiento de informacion de peticion
9 class Node{
10 public:
11     string month, day, hour, ip, reason;
12     int numAccess;
13     Node *left;
14     Node *right;
15     Node(){ //constructor vacio
16         numAccess = 0;
17         month = "";
18         day = "";
19         hour = "";
20         ip = "";
21         reason = "";
22         left = NULL;
23         right = NULL;
24     }
}
```

Esta contiene un codificador de ip y de Fecha con el fin de poder comparar entre los distintos elementos que se ingresen al árbol y decidir de una manera eficiente cual de ellas es mayor a la fecha o Ip de otro objeto de tipo Node

```
int dateIntCode(){ //codificacion de fecha
try
{
    std::map<std::string, std::string> monthNum = {
        {"Jun", "6"},
        {"Jul", "7"},
        {"Aug", "8"},
        {"Sep", "9"},
        {"Oct", "10"},
        {"Nov", "11"},
        {"Dic", "12"}
    };
    std::string result;
    if (day.length() == 1)
    {
        day = "0" + day;
    }

    result = monthNum[month]+day
    +hour[0]+hour[1]
    +hour[3]+hour[4]
    +hour[6]+hour[7];

    return stoi(result);
}
catch(const std::exception& e)
{
    std::cerr << e.what() << '\n';
    return -1;
}
}
```

```
unsigned long long ipIntCode(){ //codificacion ip
try
{
    std::string result = "";
    int dig = 0;
    int sep = 0;
    std::string cuarteto = "";
    for (int i = 0; i < ip.length(); i++)
    {
        if(ip[i] != '.' && ip[i] != ':')
        {
            cuarteto += ip[i];
            dig++;
        }
        else{
            while (dig<3)
            {
                cuarteto = "0" + cuarteto;
                dig++;
            }
            dig = 0;
            result += cuarteto;
            cuarteto = "";
        }
    }
    result+=cuarteto;
    return std::stoull(result);
}
catch(const std::exception& e)
{
    std::cerr << e.what() << ": Algun error por aqui" << '\n';
    return -1;
}
}
```

Despues se tiene la estructura de datos principal, la cual es un arbol binario y donde se almacenaran los datos de la bitacora señalada, para esto se hace la implementacion de metodos como fillBST() e insertNode(), donde apartir de un archivo se llenara el arbol binario siendo esta fincion de complejidad $O(n \log n)$ para el llenado completo del arbol ya que la insercion individual por nodo es complejidad $O(\log n)$

```
//Estructura de datos para almacenamiento de todas los datos dentro de la bitacora de peticiones
class BST{
    Node* head;
    int orderType = 0;
    int len;
private:
    void insertNode(Node*&, Node*&); //Insercion de nodo dentro de la estructura
    int sortType(Node*&, Node*&); //Dependiendo el tipo de ordenamiento se requiera (1 por Ip, 2 por Fecha)
    void deleteNode(Node*&, string&, string&, string&, string&);
    void PreOrder(Node*&); //Imprime en preOrden
    void InOrder(Node*&); //Imprime en orden
    void saveInOrder(Node*&, ofstream&); //Guarda los datos del arbol de forma ascendente dentro un archivo
    void PostOrder(Node*&); // Imprime en postOrder
    void SubstituteToMin(Node*&, Node*&); // Encuentra el minimo despues de un nodo
    void searchNode(Node*&, string&, string&); //Busqueda de un nodo en especifico
    //void save(string);
    void fillHeap(Node*&, Heap&); //Con los datos del arbol llena una estructura Heap
    //void deleteTree(Node*&);

public:
    BST() { //Constructor default
        Node* head = NULL;
        len = 0;
    };
    //~BST(){ deleteTree(head); };
    void fillBST(std::string, int); //Llenado de arbol binario
    void insert(string&, string&, string&, string&, string&); //inserta un nodo
    void InOrder() { InOrder(head); }; //Imprime en orden
    void PreOrder() { PreOrder(head); }; //Imprime preOrder
    void PostOrder() { PostOrder(head); }; //Imprime posOrder
    void deleteNode(string&, string&, string&, string&, string&); //Borra un nodo en especifico
    void searchRange(string&, string&); //Busca por rango
    int size() { return len; }; //Retorna tamano de datos dentro del arbol
    void topAccess(); //Obtiene un archivo con las Ip ordenadas de acuerdo a la cantidad de accesos denegados
    void save(string);
    //void save(string fileName){ save(fileName); }; //Guarda ordenamiento de datos de acuerdo a la condicion de ordenamiento
};
```

Internamente la funcion de llenado del arbol contiene una funcion de ordenamiento llamada sortType(), la cual decide en base a que dato sera ordenado el arbol. Esta funcion evalua el atributo orderType de la estructura BST, la cual solo puede ser 1 o 2, donde orderType = 1 ordena en base a la Ip y orderType = 2 ordena de acuerdo a la fecha

```
int BST::sortType(Node*&node, Node*&newNode){
    if (orderType == 1)
    {
        unsigned long long nodeCode = node->ipIntCode();
        unsigned long long newNodeCode = newNode->ipIntCode();
        if (nodeCode > newNodeCode)
            return 1;
        else if (nodeCode < newNodeCode)
            return -1;
        node->addAttempt();
        return 0;
    }
    else if (orderType == 2){
        int nodeCode = node->dateIntCode();
        int newNodeCode = newNode->dateIntCode();
        if (nodeCode > newNodeCode)
            return 1;
        else if (nodeCode < newNodeCode)
            return -1;
        node->addAttempt();
        return 0;
    }
    throw invalid_argument("Opcion de ordenamiento inexistente");
}
```

Una vez ordenado el arbol podemos guardar la estructura con mediante la secuencia InOrder para guardar los datos de acuerdo al orden requerido, siendo complejidad $O(n)$ el recorrer todos los nodos del arbol para imprimirlos en el archivo por guardar

```
void BST::saveInOrder(Node* &node, ofstream& file){
    if(node != NULL)
    {
        saveInOrder(node->left, file);
        file << node->month << " "
        << node->day << " "
        << node->hour << " "
        << node->ip
        << node->reason << "\n";
        saveInOrder(node->right, file);
    }
}
```

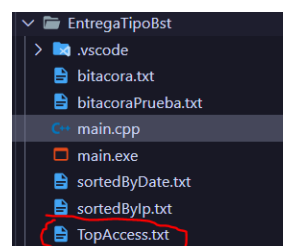
En cuanto a la obtencion del ordenamiento de los nodos de acuerdo a la cantidad de negaciones de acceso para cierta Ip se implementa una estructura de datos que nos ayude a obtener este ordenamiento de forma optima. Esta clase es la implementacion del concepto una estructura Heap de acuerdo con nuestras necesidades

```
// Estructura de datos para almacenar el valor maximo en el top de la lista
class Heap{
    int len, maxlen;
    Node** accessList;
private:
    void swap(int&, int&);
    void bubbleDown(int&);
public:
    Heap(int num){
        len = -1;
        maxlen = num;
        accessList = new Node*[num+1];
    };
    void push(Node*&);
    Node* pop();
    void save(string);
    void print();
};
```

Una vez se ejecuta el metodo topAccess() en el objeto de tipo BST se creara un objeto Heap que estructurara los datos de forma que los accesos con mayor cantidad de repeticiones negadas este siempre en la cabecera de la estructura Heap y apartir de las propiedades de este tipo de estructura poder extraer la cabeza de la estructura hasta vaciar la estructura e irla guardando en un documento llamado TopAccess.txt, que contendra el ordenamientdo de las Ip por cantidad de intentos de acceso.

El llenado de Heap es complejidad $O(n \log n)$ y el vaciado de la lista para guardarlo es $O(n \log n)$, ya que ejecuta un metodo para mantener la integridad de la estructura cada que se inserta o borra un dato

```
void BST::topAccess(){
    Heap heap(len);
    fillHeap(head, heap);
    heap.save("TopAccess.txt");
}
```

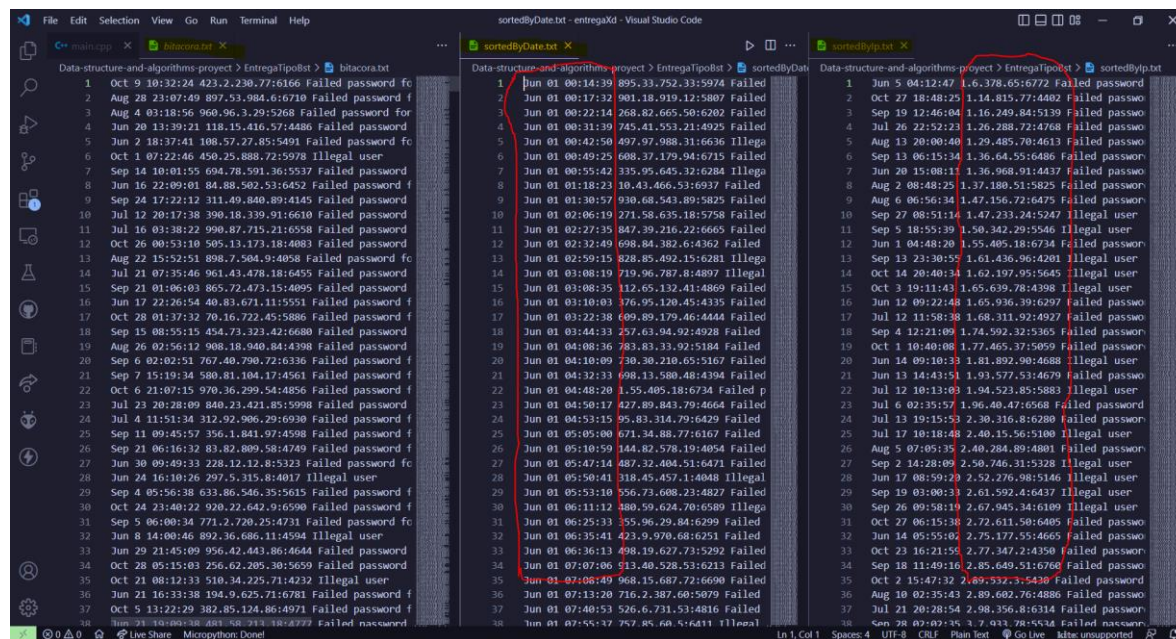


Caso prueba

Apartir del documento de bitacora.txt se llena la estructura BST de forma que apartir de esto se pueda hacer el ordenamiento por Fechas y por Ip en los documentos sortedByDate.txt y sortedByIp.txt respectivamente

```
int main()
{
    //Instancia de objeto BST
    BST listadoInformacion;
    //Llenado de estructura BST por Fecha de intento de acceso O(nlogn)
    listadoInformacion.fillBST("bitacora.txt", 2);
    //Guardado de la estructura O(n)
    listadoInformacion.save("sortedByDate.txt");
    //Llenado de estructura BST por Ip O(nlogn)
    listadoInformacion.fillBST("bitacora.txt", 1);
    //Guardado de la estructura O(n)
    listadoInformacion.save("sortedByIp.txt");
    cout<<endl;
    cout<< "Elementos evaluados: " <<listadoInformacion.size() <<endl;
    cout<<endl<<endl<<endl;
    //Llenado de Heap O(nlogn) y vaciado para guardar TopAcces.txt (nlogn)
    listadoInformacion.topAccess();
    //Impresion de Las 5 Ip que tuvieron mayor cantidad de intentos negados
    FiveTopAccess();
    return 0;
}
```

Obteniendo los siguientes documentos como resultado para el ordenamiento por Fecha e Ip



File	Content
bitacora.txt	1 Oct 9 10:32:24 423.2.230.77:6166 Failed password fo 2 Aug 28 23:07:49 897.53.984.6:6710 Failed password f 3 Aug 4 03:18:56 960.96.3.29:5268 Failed password fo 4 Jun 20 13:39:21 118.15.416.57:4486 Failed password 5 Jun 2 18:37:41 108.57.27.85:5491 Failed password fo 6 Oct 1 07:22:46 450.25.888.72:5978 Illegal user 7 Sep 14 10:01:55 694.78.591.36:5537 Failed password 8 Jun 16 22:09:01 84.88.582.53:6452 Failed password f 9 Sep 24 17:22:12 311.49.840.89:4145 Failed password 10 Jul 12 20:17:38 390.18.339.91:6610 Failed password 11 Jul 16 03:38:22 990.87.715.21:6558 Failed password 12 Oct 26 00:53:10 505.13.173.18:4083 Failed password 13 Aug 22 15:52:51 898.7.504.9:4058 Failed password fo 14 Jul 21 07:35:46 961.43.478.18:6455 Failed password 15 Sep 21 01:06:03 865.72.473.15:4095 Failed password 16 Jun 17 22:26:54 48.81.671.11:5551 Failed password f 17 Oct 20 01:37:32 70.16.722.45:5886 Failed password f 18 Sep 15 08:55:15 424.73.323.42:6680 Failed password 19 Aug 26 02:56:12 908.18.940.84:4398 Failed password 20 Sep 6 02:02:51 767.40.790.72:6336 Failed password f 21 Sep 7 15:19:34 580.81.104.17:4561 Failed password f 22 Oct 6 21:07:15 970.36.299.54:4856 Failed password f 23 Jul 23 20:28:09 840.23.421.85:5998 Failed password 24 Jul 4 11:51:34 312.92.906.29:6930 Failed password f 25 Sep 11 09:45:57 356.1.841.97:4598 Failed password f 26 Sep 21 06:16:32 83.82.809.58:4749 Failed password f 27 Jun 30 09:49:33 228.12.12.8:5323 Failed password fo 28 Jun 24 16:10:26 297.5.315.8:4017 Illegal user 29 Sep 4 05:56:38 633.86.546.35:5615 Failed password f 30 Oct 24 23:40:22 920.22.642.9:6590 Failed password f 31 Sep 5 06:00:34 771.2.720.25:4731 Failed password fo 32 Jun 8 14:00:46 892.36.686.11:4594 Illegal user 33 Jun 29 21:45:09 956.42.443.86:4644 Failed password 34 Oct 28 05:15:03 256.62.205.30:5659 Failed password 35 Oct 21 08:12:33 510.34.225.71:4232 Illegal user 36 Jun 21 16:33:38 194.9.625.71:6781 Failed password f 37 Oct 5 13:22:29 382.85.124.86:4971 Failed password f
sortedByDate.txt	1 Jun 01 00:14:39 895.33.752.33:5974 Failed 2 Jun 01 00:17:32 901.18.919.12:5807 Failed 3 Jun 01 00:22:14 268.82.665.50:6202 Failed 4 Jun 01 00:31:39 745.41.553.21:4925 Failed 5 Jun 01 00:42:50 497.97.988.31:6636 Illegal 6 Jun 01 00:49:25 688.37.179.94:6715 Failed 7 Jun 01 00:55:42 335.95.645.32:6284 Illegal 8 Jun 01 01:18:23 10.43.466.53:6937 Failed 9 Jun 01 01:30:57 930.68.543.89:5825 Failed 10 Jun 01 02:06:19 271.58.635.18:5758 Failed 11 Jun 01 02:27:35 847.39.216.22:6665 Failed 12 Jun 01 02:32:49 698.84.382.6:4362 Failed 13 Jun 01 02:59:15 828.85.492.15:6281 Illegal 14 Jun 01 03:08:19 719.96.787.8:4897 Illegal 15 Jun 01 03:08:35 112.65.132.41:4869 Failed 16 Jun 01 03:10:03 376.95.120.45:4135 Failed 17 Jun 01 03:22:38 680.89.179.46:4444 Failed 18 Jun 01 03:44:33 257.63.94.92:4928 Failed 19 Jun 01 04:08:36 783.83.33.92:5184 Failed 20 Jun 01 04:10:09 730.30.210.65:5167 Failed 21 Jun 01 04:32:33 698.13.580.48:4394 Failed 22 Jun 01 04:50:17 427.89.843.79:4664 Failed 23 Jun 01 04:53:15 95.83.314.79:6429 Failed 24 Jun 01 05:05:00 671.34.88.77:6167 Failed 25 Jun 01 05:10:59 144.82.578.19:4054 Failed 26 Jun 01 05:47:14 487.32.404.51:6471 Failed 27 Jun 01 05:50:41 118.45.457.1:4048 Illegal 28 Jun 01 05:53:10 556.73.688.23:4827 Failed 29 Jun 01 06:11:12 480.59.624.70:6589 Illegal 30 Jun 01 06:25:33 55.96.29.84:6299 Failed 31 Jun 01 06:35:41 423.9.970.68:6251 Failed 32 Jun 01 06:36:13 498.19.627.73:5292 Failed 33 Jun 01 07:07:06 513.40.528.53:6213 Failed 34 Jun 01 07:08:49 968.15.687.72:6690 Failed 35 Jun 01 07:13:20 716.2.387.60:5079 Failed 36 Jun 01 07:40:53 526.6.731.53:4816 Failed 37 Jun 01 07:55:37 757.85.60.5:6251 Illegal
sortedByIp.txt	1 Jun 5 04:12:47 1.6.378.65:6772 Failed password 2 Oct 27 18:48:25 1.14.815.77:4402 Failed password 3 Sep 19 12:46:04 1.16.749.84:5139 Failed password 4 Jul 26 22:52:23 1.26.288.72:4768 Failed password 5 Aug 13 20:00:40 1.29.485.70:4613 Failed password 6 Sep 13 06:15:34 1.36.64.55:6486 Failed password 7 Jun 20 15:00:11 1.36.968.91:4437 Failed password 8 Aug 2 08:48:24 1.37.180.51:5825 Failed password 9 Aug 6 06:56:34 1.47.156.72:6475 Failed password 10 Sep 27 08:51:14 1.47.233.24:5247 Illegal user 11 Sep 5 18:55:39 1.50.342.29:5546 Illegal user 12 Jun 1 04:48:20 1.55.405.18:6734 Failed password 13 Sep 13 23:30:55 1.61.436.96:4201 Illegal user 14 Oct 14 20:40:34 1.62.197.95:5645 Illegal user 15 Oct 3 19:11:43 1.65.639.78:4398 Illegal user 16 Jun 12 09:22:48 1.65.936.39:6297 Failed password 17 Jul 12 11:50:30 1.68.311.92:4097 Failed password 18 Sep 4 12:21:00 1.74.592.32:5365 Failed password 19 Oct 1 10:40:08 1.77.465.37:5059 Failed password 20 Jun 14 09:10:33 1.81.892.90:4688 Illegal user 21 Jun 13 14:43:51 1.93.577.53:4679 Failed password 22 Jul 12 10:13:03 1.94.523.85:5883 Illegal user 23 Jul 6 02:35:57 1.96.40.47:6568 Failed password 24 Jul 13 19:15:53 2.30.316.8:6280 Failed password 25 Jul 17 10:18:48 2.40.15.56:5100 Illegal user 26 Aug 5 07:05:35 2.40.284.89:4801 Failed password 27 Sep 2 14:28:09 2.50.746.31:5328 Illegal user 28 Jun 17 08:59:29 2.52.276.98:5146 Illegal user 29 Sep 19 03:00:31 2.61.592.43:6437 Illegal user 30 Sep 26 09:58:19 2.67.945.34:6109 Illegal user 31 Oct 27 06:15:30 2.72.611.50:6045 Failed password 32 Jun 14 05:55:02 2.75.177.55:4665 Failed password 33 Oct 23 16:21:55 2.77.347.2:4350 Failed password 34 Sep 18 11:49:16 2.85.649.51:6760 Failed password 35 Oct 2 15:47:32 2.89.532.3:5430 Failed password 36 Aug 10 02:35:43 2.89.682.76:4886 Failed password 37 Jul 21 20:28:54 2.98.356.8:6314 Failed password 38 Sep 28 02:02:35 3.7.933.78:5534 Failed password

Por otro lado en consola se imprimiran en el proceso los registros que el programa va detectando repetidos y los imprime en consola

```
Aug 26 05:06:46 859.78.637.61:4854 Failed password for illegal user test Repetido
Oct 14 11:59:21 915.81.466.75:4437 Failed password for illegal user guest Repetido
Oct 11 13:30:52 947.98.12.64:5209 Failed password for admin Repetido
Jul 27 11:14:10 177.47.222.33:5305 Failed password for illegal user guest Repetido
Jul 28 18:14:16 917.85.576.77:5725 Failed password for illegal user guest Repetido
Oct 14 18:39:17 892.26.494.94:6634 Failed password for illegal user guest Repetido
Oct 15 05:24:51 839.63.403.73:6402 Failed password for admin Repetido
Aug 14 04:56:20 109.61.399.70:4650 Failed password for root Repetido
Oct 12 19:50:17 403.73.326.72:4495 Failed password for admin Repetido
Aug 11 18:47:41 820.72.618.22:4161 Failed password for illegal user test Repetido
Jun 20 06:23:54 25.79.393.38:6876 Failed password for admin Repetido
Aug 28 14:14:52 598.4.902.10:4531 Failed password for root Repetido
Jun 7 04:55:38 807.68.577.60:4579 Illegal user Repetido
Jun 28 03:21:06 231.17.478.87:5420 Failed password for root Repetido
Aug 23 02:08:13 49.9.53.34:5982 Failed password for illegal user guest Repetido
Oct 3 08:04:27 619.45.172.56:6563 Failed password for illegal user guest Repetido

Elementos evaluados: 16807

Top de accesos denegados dentro del sistema:
619.45.172.56:6563 ---- 9
104.84.704.96:5896 ---- 7
577.7.456.52:4584 ---- 7
168.84.548.36:6782 ---- 7
254.42.860.96:6797 ---- 2
PS C:\Users\nessy\Desktop\entregaXd\Data-structure-and-algorithms-proyect\EntregaTipoBst>
```

Finalmente el programa imprime en consola la cantidad de datos que se han evaluado durante el proceso e imprime las Ip con mayor cantidad de intentos de acceso

El archivo obtenido para el ordenamiento de acuerdo a la cantidad de accesos fuer guardada en el archivo TopAcces.txt

