Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

Act 3.4 – Actividad Integral de BST (Evidencia Competencia)

Realiza en forma individual una investigación y reflexión de la importancia y eficiencia del uso de BST en una situación problema de esta naturaleza. ¿Cómo podrías determinar si una red está infectada o no?, generando un documento llamado **"ReflexAct3.4".**

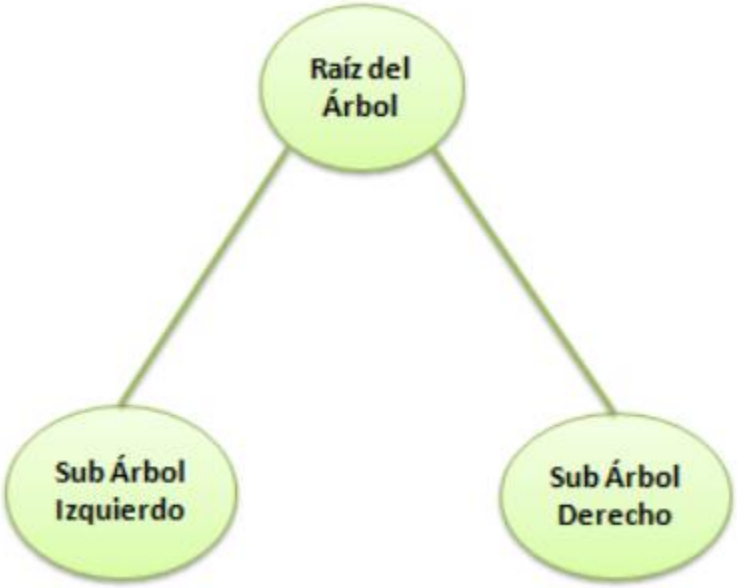
Los árboles a diferencia de las listas son unas estructuras de datos no lineal, atendiendo más a una estructura de tipo jerárquico. Los árboles son, sin duda, una de las estructuras de datos no lineales, empleadas en informática, tanto para resolver problemas de hardware como de software.

Definición de árboles:

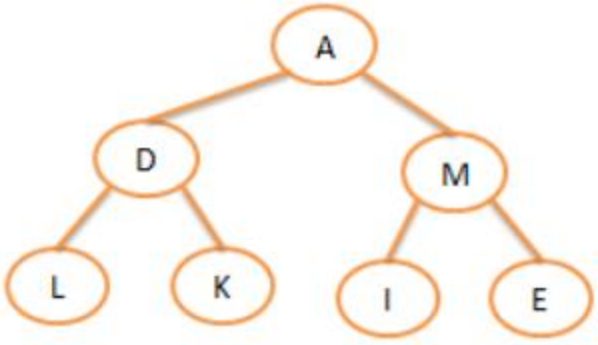
Los árboles binarios son estructuras de datos muy similares a las listas doblemente enlazadas, en el sentido que tienen dos punteros que apuntan a otros elementos, pero no tienen una estructura lógica de tipo lineal o secuencial como aquellas, sino ramificada. Tienen aspecto de árbol, ahí su nombre.

Un árbol binario es una estructura de datos no lineal en la que cada nodo puede apuntar a uno o máximo a dos nodos. También se suele dar una definición recursiva que indica que es una estructura compuesta por un dato y dos árboles. Este tipo de árbol se caracteriza porque tienen un vértice principal y de él se desprende dos ramas. La rama izquierda y la rama derecha a las que también se les conoce como subárboles.

Una representación gráfica de la estructura general de un árbol binario se puede visualizar de la siguiente manera.



La rama izquierda y la derecha también son dos árboles binarios. El vértice principal se denomina raíz y cada una de las ramas se puede denominar subárbol izquierdo y subárbol derecho.



Viendo la imagen podemos identificar algunas generalidades y partes del árbol.

Nodo: El árbol es un conjunto de elementos cada uno de los cuales se denomina nodo. Un árbol puede tener 0 nodos, tener sólo un nodo siendo este la raíz del árbol o puede tener un número finito de nodos. Cada nodo puede estar ramificado por la izquierda o por la derecha o no tener ninguna ramificación.

Existen diferentes tipos de nodos que hacen parte a los árboles:

Nodo hijo: Cualquiera de los nodos apuntando por uno de los nodos del árbol. En la imagen ejemplo los nodos ‘D’ y ‘M’ son nodos hijos de ‘A’.

Nodo padre: Nodo que contiene un puntero al nodo actual. El nodo ‘A’ es padre de ‘D’ y ‘M’.

En cuanto a la posición dentro del árbol se tiene:

Nodo raíz: Nodo que no tiene padre. Es el nodo que usaremos para referirnos al árbol. En la imagen este vendría siendo el nodo ‘A’.

Nodo hoja: Nodo que no presenta hijos. En la imagen vendrían siendo ‘L’,’K’,’I’,’E’.

Existen otros conceptos que definen las características del árbol, en relación a su tamaño:

Orden: Es el número potencial de hijos que puede tener cada elemento de árbol. De ese modo, se dice que un árbol en el que cada nodo puede apuntar a otros dos es de orden dos, si puede apuntar a tres será de orden tres y así sucesivamente.

Grado: El número de hijos que tiene el elemento con más hijos dentro del árbol. En el árbol del ejemplo, el grado es dos, ya que tanto ‘A’ como ‘D’ y ‘M’ tiene dos hijos, y no existen elementos con más de dos hijos.

Nivel: Se define para cada elemento del árbol como la distancia a la raíz, medida en nodos. El nivel de la raíz siempre será cero y el de sus hijos uno y así sucesivamente. En la imagen ejemplo el

nodo ‘D’ tiene nivel 1 así como el nodo ‘M’, el nodo ‘A’ es cero y los nodos ‘L’,’K’,’I’,’E’ son nivel dos.

Altura: La altura de un árbol se define como el numero de niveles que hay, en el ejemplo se puede decir que el árbol tiene altura 3.

El uso y aplicación de árboles son de suma utilidad al tratar de manera un gran volumen de datos, los cuales tienen una gran aplicabilidad en la indexación de archivos, base de datos, búsqueda de información, etc.

Para este caso del trabajo los árboles nos sirvieron para saber cuántas veces se intentaron conectar las ip, si múltiples ips se intentaran conectar en un lapso corto de tiempo esto podría ser un indicio de un posible ataques de bots o un intento para que la red quede infectada de un virus o un tipo malware, pudiendo sobrecargar la página y tumbarla, el árbol de búsqueda binaria nos permite realizar el ordenamiento de las ips y así hacer búsquedas de manera efectiva y eficaz, gracias a su bajo nivel de complejidad de búsqueda siendo el promedio un O(log n) y el peor caso O(n), pudiendo detectar los factores mencionados anteriormente para identificar con antelación el ataque.

Explicación del código:

Para la realización de este trabajo se crearon funciones clave para la realización de este código, cuenta();, Elimina(), EliminarNodo(), las cuales explicaremos a continuación.

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

Cuenta() y Elimina()

```
int btc_tree::cuenta(vector<vector<string>> &token, int t, string n){
    int x = 0;
    string l;
    for(int i = 0; i < t; i++){
        if(n == token[i][3]){
            x++;
        }
    }
    return x;
}

void btc_tree::Elimina(vector<vector<string>> &token, int t, string n){
    for(int i = 0; i < t; i++){
        if(n == token[i][3]){
            token[i][3] = "";
        }
    }
}
```

Estas dos funciones fueron claves para así cuando pasemos el archivo este vaya contando las ips y se vaya acumulando en caso se repitan y una vez contando esta borrara las ips repetidas para que al momento de insertar las ips al árbol no tenga datos repetidos.

Implementación:

```
for(int i = 0; i < btoken.size(); i++){

    string n;
    int na;
    n = btoken[i][3];
    if(n != ""){
        na = btc_tree::cuenta(btoken, btoken.size(), n);
        r = bst.insert(r,na,n,NULL);
        btc_tree::Elimina(btoken, btoken.size(), n);
    }

}
```

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

En la implementación lo que hacemos es un for para que pase por toda la lista dentro del for incializamos una variable tipo string para que por cada iteración este vaya guardando la ip correspondiente, inicializamos una variable int que esta guardara el numero retornado por la función cuenta que retorna el numero de veces que se repite la ips, hacemos una condicional if que en caso que el string que guarde n sea una cadena vacia no entre, en caso contrario este entra obtenemos el número y usamos la función insert(), para insertar un nuevo nodo en el árbol en donde sus parámetros son insert(el árbol, el número, la ip, su nodo padre) una vez insertado el nodo luego usamos la función Elimina que lo que hace es eliminar las ips repetidas para que no se vuelvan a insertar al árbol y así para toda la lista.

eliminar() y eliminarNodo().

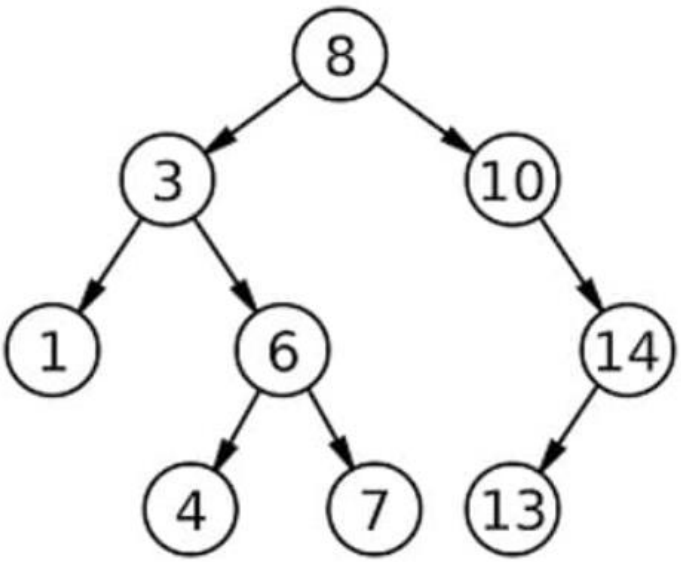
Para estas funciones lo que hace eliminar() es buscar el nodo que se busca borrar del árbol y este entra a eliminarNodo().

```
void btc_tree::eliminar(btc* t, int key){
  if(t == NULL){
    return;
  }
  else if(key < t->key){
    eliminar(t->left,key);
  }
  else if(key > t->key){
    eliminar(t->right,key);
  }
  else{
    eliminarNodo(t);
  }
}
```

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

```
void btc_tree::eliminarNodo(btc* nodoEliminar){
  if(nodoEliminar->left && nodoEliminar->right){
    btc* menor = minimo(nodoEliminar->right);
    nodoEliminar->key = menor->key;
    nodoEliminar->ip = menor->ip;
    eliminarNodo(menor);
  }
  else if(nodoEliminar->left){
    reemplazar(nodoEliminar,nodoEliminar->left);
    destruir(nodoEliminar);
  }
  else if(nodoEliminar->right){
    reemplazar(nodoEliminar,nodoEliminar->right);
    destruir(nodoEliminar);
  }
  else{
    reemplazar(nodoEliminar,NULL);
    destruir(nodoEliminar);
  }
}
```

Dentro de la función eliminarNodo(), se hicieron condicionales en los cuales lo que hace es preguntar por cualquiera de los 3 casos que se puedan presentar en el árbol ya que la eliminación de un nodo cambia dependiendo de su situación:



- Cuando el nodo tiene tanto parte derecha como izquierda:

  En la imagen ejemplo este vendría representado, el nodo raíz 8, 3 y el 6.

```
void btc_tree::eliminarNodo(btc* nodoEliminar){
  if(nodoEliminar->left && nodoEliminar->right){
    btc* menor = minimo(nodoEliminar->right);
    nodoEliminar->key = menor->key;
    nodoEliminar->ip = menor->ip;
    eliminarNodo(menor);
  }
```

- Cuando el nodo tiene solamente parte izquierda o parte derecha:

  En la imagen ejemplo estos serían los nodos, 10 y 14.

```
  else if(nodoEliminar->left){
    reemplazar(nodoEliminar,nodoEliminar->left);
    destruir(nodoEliminar);
  }
  else if(nodoEliminar->right){
    reemplazar(nodoEliminar,nodoEliminar->right);
    destruir(nodoEliminar);
  }
```

- Cuando el nodo es una hoja:

  En la imagen ejemplo estos serían los nodos 1, 4, 7 y 13.

```
  else{
    reemplazar(nodoEliminar,NULL);
    destruir(nodoEliminar);
  }
}
```

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

Funciones complemento para la función eliminarNodo():

```
btc* btc_tree::minimo(btc* t){
  if(t == NULL){
    return NULL;
  }
  if(t->left){
    return minimo(t->left);
  }
  else{
    return t;
  }
}
```

```
void btc_tree::reemplazar(btc* t, btc* nuevonodo){
  if(t->padre){
    if(t->key == t->padre->left->key){
      t->padre->left = nuevonodo;
    }
    if(t->key == t->padre->right->key){
      t->padre->right = nuevonodo;
    }
  }
  if(nuevonodo){
    nuevonodo->padre = t->padre;
  }
}
```

```
void btc_tree::destruir(btc* t){
  t->left = NULL;
  t->right = NULL;
  delete t;
}
```

Y la función para poder obtener la ip con mayor numero de accesos fue con la función searchipmayor().

```
void btc_tree::searchipmayor(btc* t){
  if(t->right == NULL){
    cout << "ip: " << t->ip << " numero de accesos: " << t->key << endl;
    eliminar(t,t->key);
    return;
  }
  searchipmayor(t->right);
}
```

Como sabemos que las ips con mayor números de accesos estarán al final del subárbol derecho, entonces buscamos en el árbol siempre su lado derecho hasta que este encuentre NULL, queriendo decir que se encontró la ip con mayor número de accesos, imprimiendo en pantalla la ip y el numero de accesos, luego usamos la función eliminar para borrar la ip impresa para que al momento de volver a usar la función searchipmayor, esta nos arroje la nueva ip con mayor cantidad de accesos y así sucesivamente hasta imprimir las 5 ips con mayor números de accesos.

```
cout << "Top 5 ips con mayor numero de accesos: " << endl;
bst.searchipmayor(r);
bst.searchipmayor(r);
bst.searchipmayor(r);
bst.searchipmayor(r);
bst.searchipmayor(r);
return 0;
```

```
Top 5 ips con mayor numero de accesos:
ip: 297.5.315.8:4017 numero de accesos: 6
ip: 505.13.173.18:4083 numero de accesos: 5
ip: 865.72.473.15:4095 numero de accesos: 4
ip: 898.7.504.9:4058 numero de accesos: 3
ip: 390.18.339.91:6610 numero de accesos: 2
```

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

	CASOS DE PRUEBA ACT 3.4				
No.	Precondiciones	Resumen	Pasos	Resultado Esperado	Resultado obtenido
1	Tener el código y tener un bst vació	El usuario introduce una la función searchipmayor, en un árbol vació.	1. Iniciar el programa. 2. Llamar a la función e insertar un árbol vació.	Una vez insertado el árbol en la función, este no debería regresar nada.	El programa no despliega nada en pantalla.
2	Tener el código y tener un bst válido	El usuario introduce el bst a la función searchipmayor	1. Correr el programa. 2. Introducir un bst a la función.	El programa debería imprimir los 5 nodos con mayor número de accesos.	El programa imprime las ips con mayor número de accesos.

Caso prueba 1:

```
Top 5 ips con mayor numero de accesos:
```

Caso prueba 2:

```
Top 5 ips con mayor numero de accesos:
ip: 297.5.315.8:4017 numero de accesos: 6
ip: 505.13.173.18:4083 numero de accesos: 5
ip: 865.72.473.15:4095 numero de accesos: 4
ip: 898.7.504.9:4058 numero de accesos: 3
ip: 390.18.339.91:6610 numero de accesos: 2
```