

Reflexión Individual

Actividad 3.4

Santiago Álvarez Valdivia
A01640172

**Programación de Estructuras de Datos y
Algoritmos Fundamentales**

Tecnológico de Monterrey

Campus Guadalajara

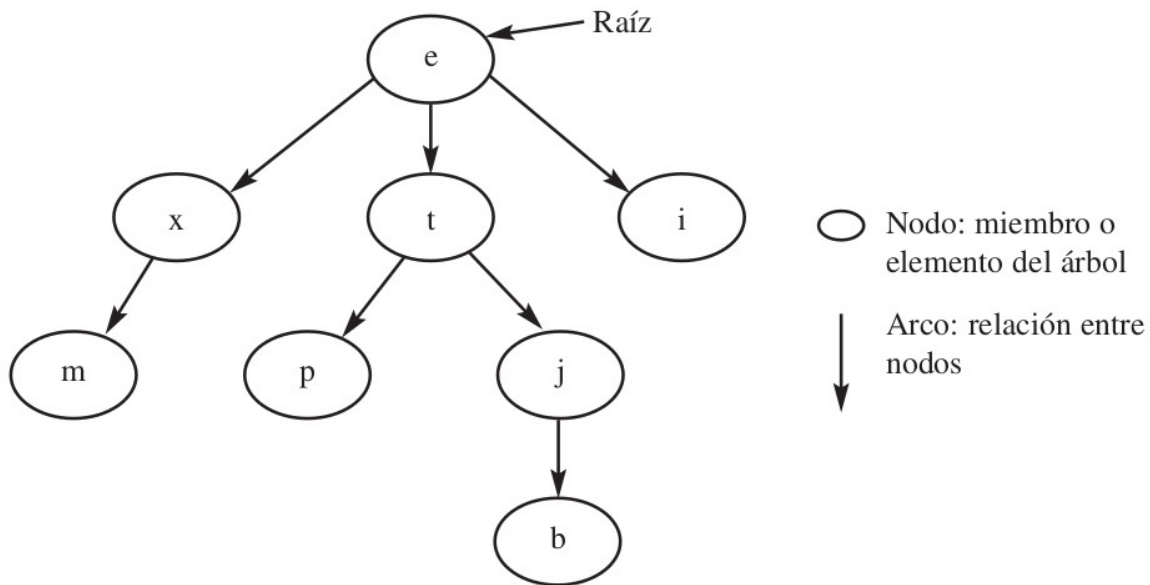
Escuela de Ingeniería y Ciencias

Profesor: Dr. Eduardo Arturo Rodríguez Tello

Sábado, 6 de noviembre de 2021

1 Árboles (estructura de datos)

En programación, un árbol es una estructura de datos no lineal. En estos cada elemento (conocido como nodo) puede tener varios sucesores (mejor conocidos como hijos), pero cada nodo sólo puede tener un antecesor (o padre). El primer nodo de un árbol es conocido como raíz.



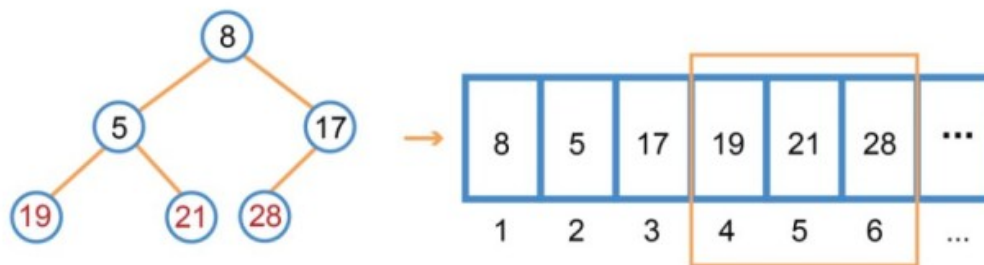
Por su parte, se le llama árbol binario a aquel en el que cada nodo sólo puede tener máximo 2 hijos. Existen distintos tipos de árboles binarios, de acuerdo a sus características. Algunos de estos tipos son:

- **Árboles binarios de búsqueda (BST):** su principal característica es que los hijos izquierdos deben ser menores que el nodo padre, y los derechos deben ser mayores.
- **Árboles balanceados:** Cumplen las mismas reglas que los árboles BST, sólo que estos también cuidan que el árbol esté balanceado, es decir, que todos los nodos hoja se encuentren más o menos al mismo nivel, para agilizar las operaciones de búsqueda.
- **Árboles Heap:** este tipo de árboles, se caracteriza porque cada elemento debe tener una prioridad mayor o igual que sus hijos. Así, el elemento con mayor prioridad de un árbol está en la raíz. Este tipo de árboles se describirán a mayor detalle a continuación.

2 Árboles Heap

Este tipo de árbol se caracteriza porque los valores hijos deben tener una menor prioridad que el valor padre, siendo la prioridad distinta dependiendo del contexto. Por ejemplo, algunos árbol priorizan a los mayores elementos (MaxHeap), mientras que otros priorizan a los menores.

Otra característica es que todos los niveles del árbol deben estar completos, a excepción del último nivel, que se debe llenar siempre de izquierda a derecha. Cabe mencionar que este tipo de árboles son frecuentemente representados como un arreglo unidimensional (un vector, por ejemplo), y que en caso del presente programa, así será implementado.



Debido a la característica de “completitud” de un árbol heap, los hijos de un nodo se pueden encontrar fácilmente en un arreglo. Si tenemos un elemento en la posición k , su hijo izquierdo estará en la posición $2k$, mientras que su hijo derecho estará en $2k+1$. Para que esta propiedad se cumpla, el primer elemento debe estar numerado con 1.

3 El Método de Ordenamiento HeapSort

HeapSort es un método de ordenamiento, el cual se caracteriza por ordenar los elementos de una lista del mismo modo en que se ordenarían para crear un árbol MaxHeap, es decir, que consiste en intercambiar elementos de una lista de tal modo que cada elemento en una posición k sea mayor o igual que los elementos en las posiciones $2k$ y $2k+1$ (de haberlos). Posteriormente, se va colocando el primer elemento (raíz) al final del arreglo, hasta terminar con los elementos que forman parte de nuestro Heap. Así, terminamos con una lista ordenada de menor a mayor.

Complejidad Temporal

$O(n \log n)$ [7]

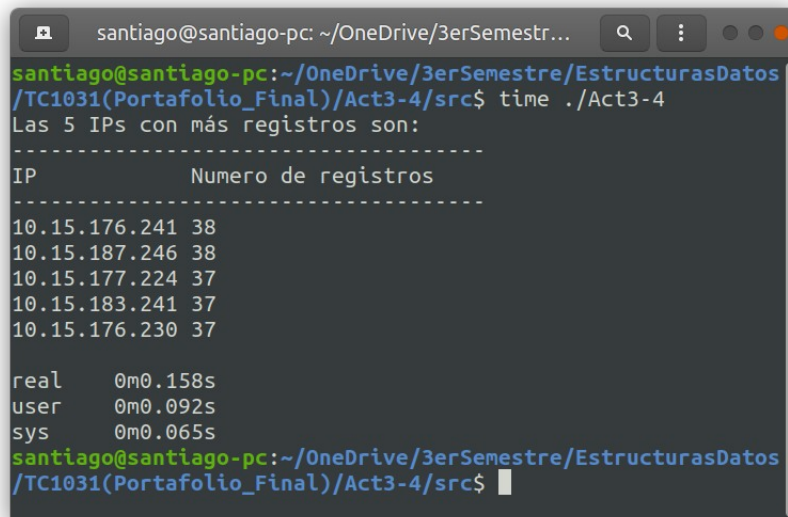
4 Árboles Heap y HeapSort en este programa

La funcionalidad del presente programa es tomar una lista de incidencias de inicio de sesión (que contienen IP de origen, fecha, hora y descripción corta del incidente), y obtener una lista de las IP junto con su número de incidencias, ordenada de mayor a menor.

Para realizar esta funcionalidad, el uso del algoritmo HeapSort fue fundamental. Primero, el programa ordenará todas las incidencias por su IP, de modo que las incidencias con la misma IP queden juntas, así, es más fácil contar las incidencias de cada IP, sin necesidad de recorrer la lista entera cada vez.

Por su parte, un árbol heap es utilizado también. Una vez que se tienen las incidencias de una IP, se añade esta (junto con sus incidencias) al árbol, que tiene como prioridad un mayor número de incidencias. Una vez que están todas las IP en el árbol, se procede a ir tomando el nodo raíz hasta terminar con el árbol. Así, al ir tomando el nodo raíz, vamos tomando las IP en orden descendente de incidencias.

En este caso, el uso de estas herramientas resultó muy útil y eficiente. Como podemos ver en la siguiente captura (con cronometraje), el tiempo de ejecución del programa se acercó a una décima de segundo¹.

A terminal window screenshot showing the execution of a program. The prompt is 'santiago@santiago-pc: ~/OneDrive/3erSemestre/EstructurasDatos/TC1031(Portafolio_Final)/Act3-4/src\$'. The command executed is 'time ./Act3-4'. The output shows the top 5 IPs with the most records, followed by timing information.

```
santiago@santiago-pc: ~/OneDrive/3erSemestre/EstructurasDatos/TC1031(Portafolio_Final)/Act3-4/src$ time ./Act3-4
Las 5 IPs con más registros son:
-----
IP                Numero de registros
-----
10.15.176.241 38
10.15.187.246 38
10.15.177.224 37
10.15.183.241 37
10.15.176.230 37

real    0m0.158s
user    0m0.092s
sys      0m0.065s
santiago@santiago-pc: ~/OneDrive/3erSemestre/EstructurasDatos/TC1031(Portafolio_Final)/Act3-4/src$
```

Por supuesto, el uso de estas herramientas también está relacionado a la naturaleza del programa. Por ejemplo, el árbol MaxHeap era el indicado porque queríamos obtener las mayores incidencias primero, y porque queríamos acceder a los resultados en orden una sola vez. Si quisiéramos, por ejemplo, acceder a algún resultado específico, y poder hacerlo muchas veces, habiéramos empleado un árbol binario de búsqueda (BST).

¹ Empleando compilación optimizada (-O3) en el compilador GNU g++, en una computadora personal con procesador Intel Core i5 4300U y 8GB de RAM

¿Cómo podrías determinar si una red está infectada o no?

Para determinar si la red está infectada, un buen indicador sería ver si existe algún usuario dentro, conectado desde alguna de las direcciones IP que tenemos registradas en incidencias, especialmente alguna con muchas incidencias registradas.

Por supuesto, esto no es 100% preciso, ya que un atacante podría haber surgido desde alguna IP nueva, o tal vez alguna de las IP registradas en incidencias es de un usuario legítimo, que cometió algún error humano como teclear mal su contraseña. Sin embargo, debería ser suficientemente bueno como medida de prevención.

Referencias

- [1] Buemo, S. G., Botello, F. O., & Cerpas, J. L. G. (2007). *Estructura de datos orientada a objetos*. Pearson Educación.
- [2] CPlusPlus. (2021a). `strftime`. En *C++ Reference*.
<https://www.cplusplus.com/reference/ctime/strftime/>
- [3] CPlusPlus. (2021b). `struct tm`. En *C++ Reference*.
<https://www.cplusplus.com/reference/ctime/tm/?kw=struct%20tm>
- [4] de la Cueva, V. M., González, L. H., & Salinas, E. G. (2020). *Estructuras de datos y algoritmos fundamentales*. Editorial Digital del Tecnológico de Monterrey.
- [5] GeeksforGeeks. (2021a, junio 28). *Sort the given IP addresses in ascending order*.
<https://www.geeksforgeeks.org/sort-the-given-ip-addresses-in-ascending-order/>
- [6] GeeksforGeeks. (2021b, septiembre 15). *HeapSort*. Recuperado 25 de octubre de 2021, de <https://www.geeksforgeeks.org/heap-sort/>
- [7] Universidad Autónoma de México. (s. f.). *HeapSort*. Recuperado 6 de noviembre de 2021, de http://aniei.org.mx/paginas/uam/CursoAA/curso_aa_20.html