Zapopan, Jalisco a 24 de octubre de 2016

Análisis y Diseño de Algoritmos

Juan Antonio Vega Fernández



Directorio

Santiago Ruiz Angulo 3118030

Oscar Dávila Ramírez 3118054

Sumario

[Tabla Hash 3](#_Toc465094592)

[Algoritmo Hash 3](#_Toc465094593)

[Arboles Balanceados 4](#_Toc465094594)

[Arboles AVL 5](#_Toc465094595)

[Operaciones 6](#_Toc465094596)

[Análisis de Complejidad 7](#_Toc465094597)

[Órdenes de Complejidad 7](#_Toc465094598)

[Función Hash 7](#_Toc465094599)

[Inserción 7](#_Toc465094600)

[Eliminación 7](#_Toc465094601)

[Búsqueda 7](#_Toc465094602)

[Árbol AVL 8](#_Toc465094603)

[Inserción 8](#_Toc465094604)

[Eliminación 8](#_Toc465094605)

[Búsqueda 8](#_Toc465094606)

[Complejidad en la aplicación 8](#_Toc465094607)

[Inserción 9](#_Toc465094608)

[Eliminación 9](#_Toc465094609)

[Búsqueda 9](#_Toc465094610)

[Anexo 10](#_Toc465094611)

[Uso de la aplicación: 10](#_Toc465094612)

[Crear un registro automáticamente 10](#_Toc465094613)

[Crear un registro nuevo manualmente: 11](#_Toc465094614)

[Buscar un registro 12](#_Toc465094615)

[Editar un registro 13](#_Toc465094616)

[Borrar un registro 14](#_Toc465094617)

[Referencias 15](#_Toc465094618)

# Tabla Hash

Las tablas hash son estructuras de datos que se utilizan para almacenar un número elevado de datos sobre los que se necesitan operaciones de búsqueda e inserción muy eficientes. Una tabla hash almacena un conjunto de pares “(clave, valor/datos)”. La clave es única para cada elemento de la tabla y es el dato que se utiliza para buscar un determinado valor/datos.

El tiempo medio de recuperación de información es constante, es decir, no depende del tamaño de la tabla ni del número de elementos almacenados en la misma.

Una tabla hash se construye con tres elementos básicos:

* Un vector capaz de almacenar “m” elementos.
* Función de dispersión que permita a partir de los datos (llamados clave) obtener el índice donde estará el dato en el arreglo.
* Una función de resolución de colisiones:
  + Encadenamiento separado: Las colisiones se resuelven insertándolas en una lista. De esa forma tendríamos como estructura un vector de listas. Al número medio de claves por lista se le llama factor de carga y habría que intentar que esté próximo a 1.
  + Direccionamiento abierto: Utilizamos un vector como representación y cuando se produzca una colisión la resolvemos reasignándole otro valor hash a la clave hasta que encontremos un hueco.

## Algoritmo Hash

**Función de Hashing**: Es una función h: X -> Z que transforma una cadena binaria o mensaje de longitud arbitraria (x) en otra cadena binaria (z) de longitud constante (n) llamada digesto.

Para la transformación del ID se utiliza el algoritmo de hash *FNV (Fowler/Noll/Vo)*, la cual surgió de una idea presentada al comité del IEEE POSIX P1003.2 por Glenn Fowler y Phong Vo en 1991, luego Landon Curt Noll la mejoró. Es rápida, produce pocas colisiones, realiza una buena dispersión de los hashes y funciona muy bien para hacer hashing en cadenas casi idénticas.

Cabe mencionar que el algoritmo FNV no es un algoritmo criptográfico. Dicho algoritmo cuenta con una variante llamada FNV-1a, el cual fue seleccionado para ser utilizado en el diseño del programa en cuestión.

Este algoritmo actualmente es capaz de generar resultados hash de 32, 64, 128, 256, 512 y 1024 bits.

El algoritmo está representado por el siguiente pseudocódigo:

*hash* = ***FNV\_offset\_basis***

for each *byte\_of\_data* to be hashed

*hash* = *hash* [XOR](https://en.wikipedia.org/wiki/XOR) *byte\_of\_data*

*hash* = *hash* × ***FNV\_prime***

return *hash*

Donde todas las variables son enteros sin signo y tienen el mismo número de bits que el hash generado por el algoritmo, a excepción de *byte\_of\_data* que es un entero sin signo de 8 bits.

Los parámetros utilizados en el algoritmo, ***FNV\_offset\_basis*** y ***FNV\_prime*** consisten en valores ya determinados para el algoritmo, más acerca de su historia y sus valores para el tamaño del hash requerido, puede ser encontrada en la página del algoritmo (Ver Página 9).

# Arboles Balanceados

Es un árbol binario de búsqueda en el cual se cumple: “Para todo nodo T del árbol, la altura de los subárboles izquierdo y derecho no debe diferir en más de una unidad”.

Esto es solamente posible cuando el número de hojas es 2*k*  para *k* ∈ Z+, en que caso el largo de todos los caminos desde la raíz hasta las hojas es exactamente *k*.

La idea central de éstos es la de realizar reacomodos o balanceos, después de inserciones o eliminaciones de elementos.

Los árboles balanceados se parecen mucho, en su mecanismo de formación, a los números de Fibonacci. El árbol de altura 0n es vacío, el árbol de altura 1 tiene un único nodo y en general el número de nodos del árbol con altura h>1 se calcula aplicando la siguiente fórmula recursiva:

K = Numero mínimo de nodos de árbol

h = Altura

## Arboles AVL

La denominación de árbol AVL viene dada por los creadores de tal estructura (Adelson-Velskii y Landis).

La propiedad de equilibrio que debe cumplir un árbol para ser AVL asegura que la profundidad del árbol sea por lo que las operaciones sobre estas estructuras no deberán recorrer mucho para hallar el elemento deseado. El tiempo de ejecución de las operaciones sobre estos árboles es, a lo sumo en el peor caso, donde n es la cantidad de elementos del árbol.

Sin embargo, esta misma propiedad de equilibrio de los árboles AVL implica una dificultad a la hora de insertar o eliminar elementos: estas operaciones pueden no conservar dicha propiedad.

### Operaciones

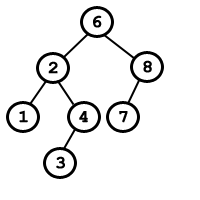


Figura 1. Árbol AVL de enteros

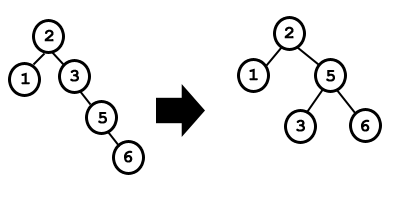


Figura 2. Rotación simple

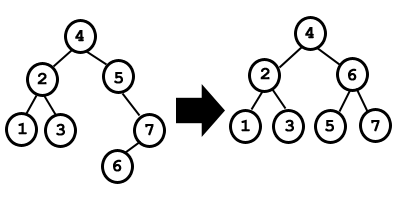


Figura 2. Rotación simple

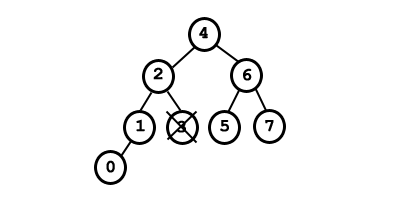
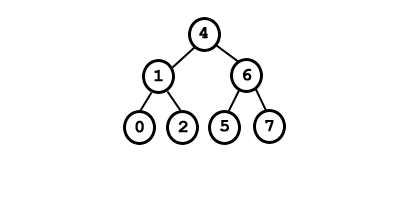
 

Figura 3. Borrado y Rotación

Las operaciones adicionales en un árbol AVL son análogas a las de árboles binarios de búsqueda.

# Análisis de Complejidad

La complejidad algorítmica representa la cantidad de recursos (temporales) que necesita un algoritmo para resolver un problema y por tanto permite determinar la eficiencia de dicho algoritmo.

## Órdenes de Complejidad

Se dice que define un **"orden de complejidad"**. Escogeremos como representante de este orden a la función f(n) más sencilla del mismo. Así tendremos

* O(1) orden constante
* O(n) orden lineal
* O(log n) orden logarítmico
* O(n log n) orden logarítmico
* O(n2) orden cuadrático
* O(na) orden polinomial (a > 2)
* O(an) orden exponencial (a > 2)
* O(n!) orden factorial

## Función Hash

La complejidad en notación Big O para cualquier algoritmo hash simple consiste en los siguientes valores:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Inserción | Eliminación | Búsqueda |
| Caso Promedio |  |  |  |
| Peor Caso |  |  |  |

## Árbol AVL

La complejidad en notación Big O para cualquier Árbol AVL consiste en los siguientes valores:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Inserción | Eliminación | Búsqueda |
| Caso Promedio |  |  |  |
| Peor Caso |  |  |  |

Esto demostrado a continuación:

representa el mínimo número de nodos que se requieren para formar un árbol de altura . Si conocemos y entonces es posible determinar . representa el mínimo número de nodos en el subárbol de la izquierda del nodo raíz y es el mínimo número de nodos en la derecha del subárbol del nodo raíz, entonces:

Donde 1 representa el nodo raíz en sí. De aquí podemos construir iterativamente , entonces, usando esta fórmula, podemos reducir tal que:

Demostrando entonces que la altura h de un árbol AVL es en efecto.

## Complejidad en la aplicación

En la mostrada implementación durante el desarrollo de la aplicación se utiliza una tabla hash en la cual se utiliza un árbol AVL para resolver las colisiones. Basándonos en el comportamiento asintótico de la ecuación generada por los algoritmos descritos anteriormente, podemos determinar la complejidad para cada una de las operaciones anteriormente descritas: Inserción, eliminación y búsqueda de un nodo; Estás operaciones entonces resultando en las complejidades mostradas a continuación:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Inserción | Eliminación | Búsqueda |
| Caso Promedio |  |  |  |
| Peor Caso |  |  |  |

Las cuales, como se puede observar corresponden a las mismas del árbol AVL.

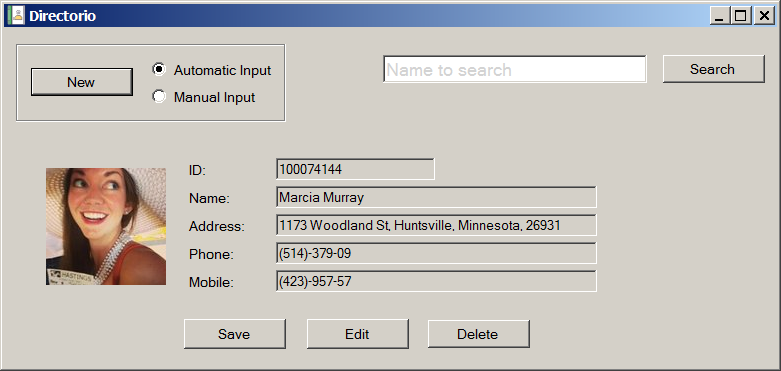
# Anexo

## Uso de la aplicación:

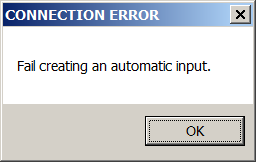
### Crear un registro automáticamente

1. Seleccionar **Automatic Input**
2. Seleccionar **New**

La información del nuevo registro será mostrada como se muestra en la siguiente imagen:

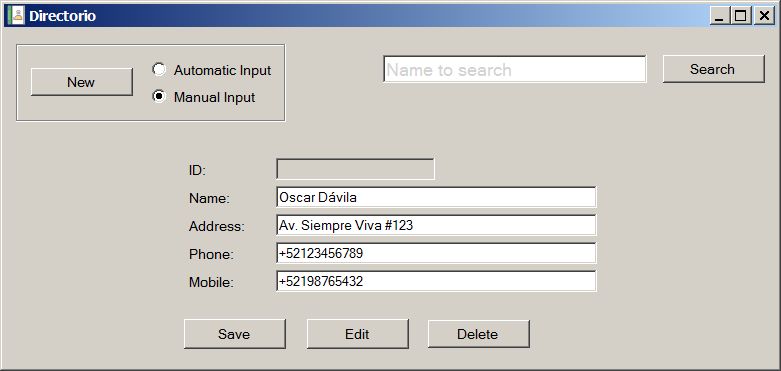


Para poder crear un registro automáticamente es necesario contar con internet y no estar detrás de un proxy, en dado caso de no cumplir alguna de estas condiciones, el siguiente mensaje de error será desplegado:

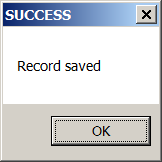


### Crear un registro nuevo manualmente:

1. Seleccionar **Manual Input**
2. Llenar los **Name, Address, Phone, Mobile**
3. Seleccionar **Save**



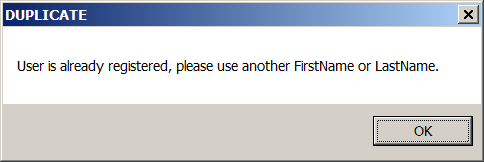
En caso de que el registro se haya guardado exitosamente, el siguiente mensaje será desplegado:



En caso de que se intente guardar un registro sin llenar todos los campos necesarios, el siguiente mensaje será desplegado:

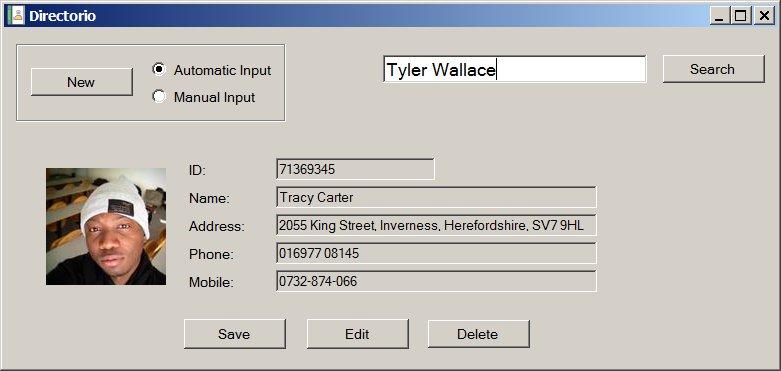


En caso de que se intente guardar un registro donde el campo Name contenga el mismo valor que un registro ya existente, el siguiente mensaje será desplegado:

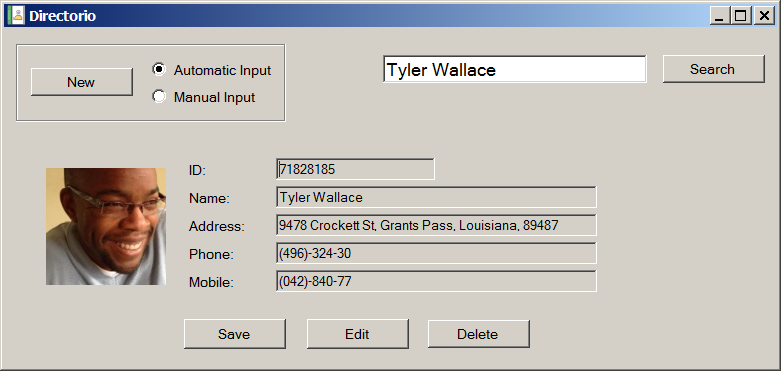


### Buscar un registro

1. Llenar el campo donde se muestra **Name to search**

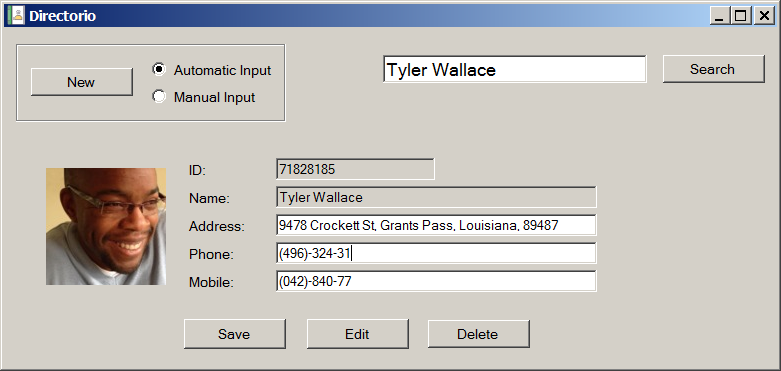


1. Seleccionar **Search**



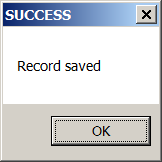
### Editar un registro

1. Buscar el registro
2. Seleccionar **Edit**
3. Editar los campos deseados



1. Seleccionar **Save**

En caso de que la edición del registro haya sido exitosa se mostrará el siguiente mensaje:

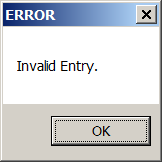


### Borrar un registro

1. Buscar el Registro
2. Seleccionar **Delete**

Una vez borrado el registro los campos se limpiarán.

En caso de intentar borrar un registro sin tener uno seleccionado actualmente en pantalla se mostrará el mensaje:



# Referencias

1. Pardo, A. (2011). *Tablas Hash*. Universidad Carlos III de Madrid. Recuperado el 16 de octubre de 2016, de <http://www.it.uc3m.es/abel/as/MMC/M2/HashTable_es.html>.
2. Benemérita Universidad Autónoma de Puebla. *Algoritmos de Búsqueda* [en línea]. [fecha de consulta: 16 de octubre de 2016]. Disponible en: <<http://www.cs.buap.mx/~iolmos/ada/TablasHashArbolesBinarios.pdf> >
3. Sama Villanueva, S. (s.f.). *Qué son las tablas hash.* Human Communication and Interaction. Recuperado el 16 de octubre de 2016, de <http://www.hci.uniovi.es/Products/DSTool/hash/hash-queSon.html>
4. CertiSur S.A. (s.f.). *Impacto De Recientes Ataques De Colisiones Contra Funciones De Hashing De Uso Corriente* [en línea]. [fecha de consulta: 16 de octubre de 2016]. Disponible en: <<https://www.certisur.com/sites/default/files/docs/ataques_funciones_hashing.pdf> >
5. Curt Noll, L. (s.f.). *FNV Hash*. Recuperado el 16 de octubre de 2016, de <http://www.isthe.com/chongo/tech/comp/fnv/index.html>
6. Á*rboles balanceados (AVL)* [en línea]. [fecha de consulta: 17 de octubre de 2016]. Disponible en: < <http://mapaches.itz.edu.mx/~mbarajas/edinf/avl.pdf> >
7. Universidad Autónoma de Nuevo León (s.f.). Á*rboles Balanceados* [en línea]. [fecha de consulta: 17 de octubre de 2016]. Disponible en: < <http://elisa.dyndns-web.com/teaching/aa/pdf/clase0210.pdf> >
8. Gurin, S. (2004). Á*rboles AVL* [en línea]. [fecha de consulta: 17 de octubre de 2016]. Disponible en: < <http://es.tldp.org/Tutoriales/doc-programacion-arboles-avl/avl-trees.pdf> >
9. Geeks for Geeks. [fecha de consulta: 17 de octubre de 2016] Disponible en: < <http://www.geeksforgeeks.org/avl-tree-set-1-insertion/> >
10. Geeks for Geeks. [fecha de consulta: 17 de octubre de 2016] Disponible en: < <http://www.geeksforgeeks.org/avl-tree-set-2-deletion/> >