Estructura de Datos y Algoritmos
Departamento de electrónica
Universidad Técnico Federico Santa María

DEPARTAMENTO DE ELECTRONICA

Ayudantía 11: Tablas Hash

Objetivo Principal: Conocer cómo analizar e implementar tablas Hash.
Autor: Kevin Pizarro Aguirre

---

1. Tabla Hash

Es una de las grandes estructuras de datos, utilizada ampliamente en muchas áreas y muy versátil. Se basa en la idea de un par llave-valor, donde a través de una llave (key) accedemos a un valor, modificamos o añadimos. En Python es implementado en los *diccionarios*, donde de igual manera se tiene un acceso a los valores según la llave correspondiente.

1.1. Tabla de direccionamiento directo

También conocidas como *Direct-Address Tables*. Sea un universo U que contiene m elementos de las llaves posibles a usar, K el conjunto de elementos con las llaves utilizadas y T la tabla de direccionamiento directo.

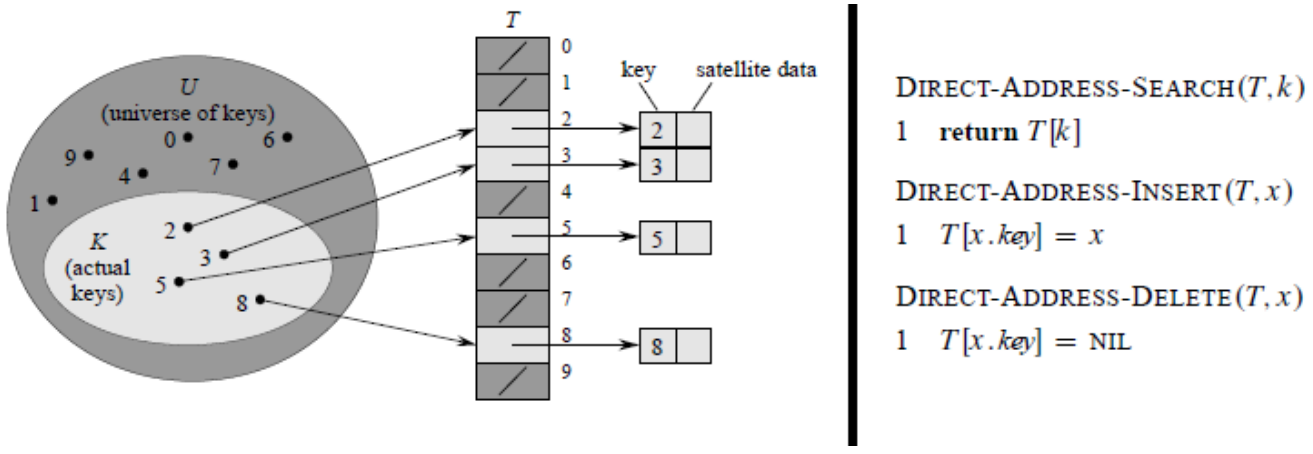


Figura 1: Tabla de direccionamiento directo y operaciones **Search**, **Insert** y **Delete**.

Notar que cada llave utilizada se corresponde **directamente** con la posición en la tabla de direccionamiento directo, la función de Hash[1] es la identidad (h(k) = k). También que las operaciones de interés son de tiempo constante. La mayor desventaja de esta implementación es el espacio de memoria y que en realidad en muchas ocasiones no se tiene certeza de las llaves que se utilizarán.

[1] Abordado posteriormente.

1.2. Colisión y encadenamiento

Entonces surge otra forma de abordar el problema y limitando el tamaño del universo. La pregunta en cuestión es, ¿y si utilizamos una función Hash que no sea directa?, es este el enfoque que se toma. Pero en este caso se puede dar el caso que para llaves diferentes se acceda al mismo índice en la tabla Hash, esta situación es llamada **colisión**.

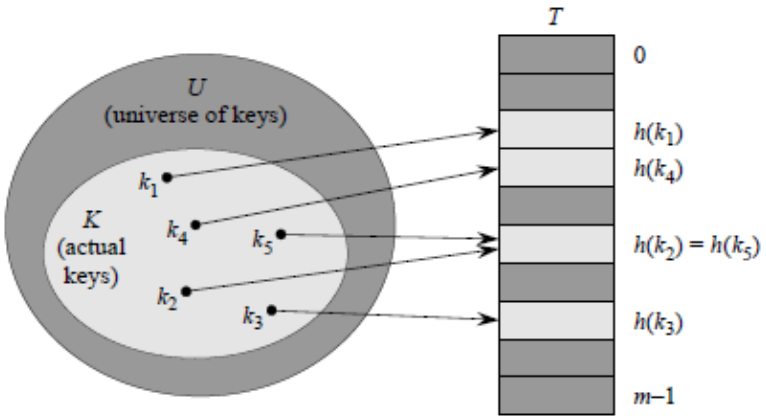
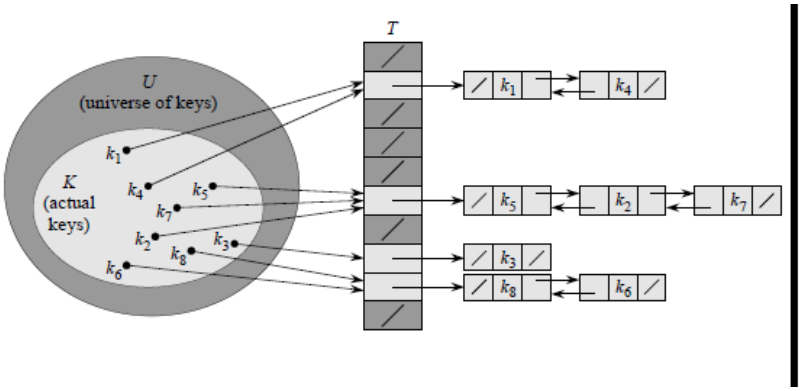


Figura 2: Tabla Hash con colisión.

Al haber una colisión hay múltiples formas de tratarlo, una de las cuales nos enfocaremos se llama **encadenamiento** (chaining). Cuando hay colisión simplemente se "encadenarán" los valores entrantes a los ya existentes, una de las formas de hacerlo es a través de listas (simplemente o doblemente enlazadas).




```
CHAINED-HASH-INSERT(T, x)
1  insert x at the head of list T[h(x.key)]
CHAINED-HASH-SEARCH(T, k)
1  search for an element with key k in list T[h(k)]
CHAINED-HASH-DELETE(T, x)
1  delete x from the list T[h(x.key)]
```

Figura 3: Resolución de colisiones mediante encadenamiento.

En el peor de los casos tendremos que todos los elementos "caen" dentro de una misma casilla, es decir, estaremos en presencia de una lista. Se demuestra que en dependencia de las funciones de hashing es que en realidad los elementos tienen la misma probabilidad de ser insertados en cualquier casilla de la tabla, existe independencia.

Además se puede definir un factor de carga $\alpha = n/m$, donde n es el número de elementos almacenados en la tabla y m el tamaño de la tabla Hash. Es deseable que sea cercano a 1, para valores cercanos a 0 implica que casi no se tienen elementos almacenados y para valores mayores que 1 se tienen muchas colisiones y sobrecarga. Finalmente $T(n) = \Theta(1 + \alpha)$.

2. Funciones Hash

Como fue mencionado anteriormente, es de vital importancia tener una buena función de hashing, caso contrario puede que tengamos muchas colisiones. Lo esencial es que la función pueda distribuir uniformemente las llaves a la tabla, lo que se conoce como Simple Uniform Hashing Asumption (SUHA). A continuación algunas de las funciones de hashing que son conocidas y logran SUHA bajo ciertas condiciones.

Caso 1: Sea k una clave, número real entre 0 y 1 con distribución uniforme. Bajo esta condición se define la siguiente función de hashing que cumple con SUHA.

$$h(k) = \lfloor km \rfloor \tag{1}$$

Caso 2 - Método de la división: Basado en la aritmética modular, es decir realizar la división entre dos números y obtener el residuo de la división. En este método lo importante es utilizar valores adecuados para m, si bien corresponde al tamaño de nuestra tabla esta no debe ser elegida al azar. No escoger potencias de 2, provocará colisiones[2]. Usualmente se escoge un número primo lejano a una potencia de 2. La clave k puede ser cualquier número natural, incluido el cero.

$$h(k) = k \ mod \ m \tag{2}$$

Caso 3 - Método de la multiplicación: En este caso las consideraciones que se deben tener son A un valor real entre 0 y 1, k pertenece a los naturales incluido el cero y que el valor de m ya no es crítico (generalmente $m = 2^p$). Knuth recomienda A = $(\sqrt{5} - 1)/2$[3].

$$h(k) = \lfloor m(kA \ mod \ 1) \rfloor \tag{3}$$

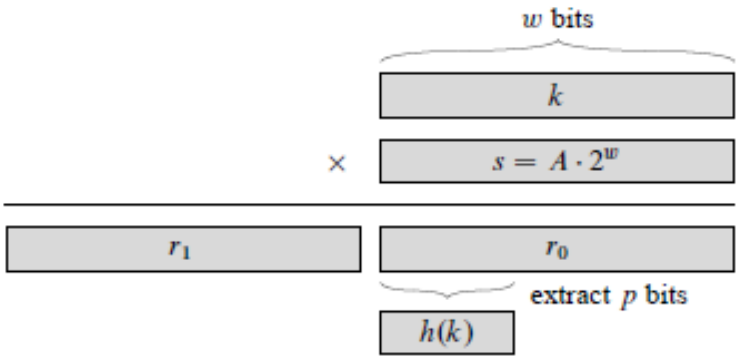


Figura 4: Esquema para el método de la multiplicación.

[2]Más detalles en el libro guía, Introduction to Algorithms.
[3]Se basa en *Fibonacci Hashing*.

3. Ejercicios

1.- Diseñe un algoritmo para verificar que una secuencia esta compuesta por enteros consecutivos. Por ejemplo, la secuencia {-1, 5, 4, 2, 0, 3, 1} está compuesta por enteros consecutivos del -1 al 5. Suponga que el rango de variación de los enteros está en [-100, 100]. Si su solución contempla el uso de una tabla hash ¿cuál es la función hash explícitamente?, ¿cuál es la complejidad de su algoritmo?

2.- Diseñee un algoritmo que ordene alfabéticamente un conjunto de n siglas de tres letras (solo mayúsculas). Cada letra se representa en según la tabla ASCII. Así la letra A corresponde al número 65 (0x41) y la Z corresponde al número 90 (0x5A). ¿Cuál es el tiempo de ejecución de su solución?