Estructuras de Datos

Hash

Colisiones - Direccionamiento Abierto

La idea de Direccionamiento Abierto consiste en una sucesión de funciones hash, es decir, $\{h_0, h_1, \ldots, h_n\}$.

Supongamos que tenemos al elemento X. Lo primero que se intenta es almacenar el dato aplicando la función de hash h_0, si tabla[$h_0(X)$] está ocupado se prueba con h_1, es decir me fijo si tabla[$h_1(X)$] está ocupado, y así sucesivamente.

Es decir, se resuelven las colisiones aplicando una función hash. Hay diferentes formas de generar esta sucesión de funciones, comenzamos con Linear probing.

Colisiones - Direccionamiento Abierto - Linear probing

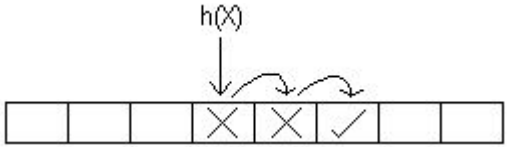
Es el método más simple de direccionamiento abierto, en donde las funciones de hash se definen como:

$$h_0(X)=h(X)$$
$$h_{k+1}(X)=(h_k(X)+1) \ mod \ m$$

siendo m el tamaño de la tabla.

Colisiones - Direccionamiento Abierto - Linear probing

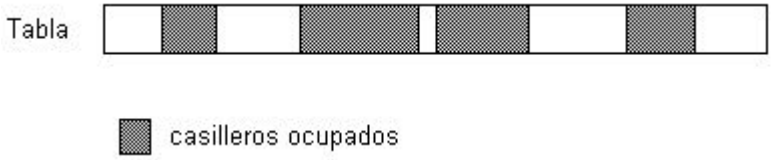
Si analizamos esta definición, podemos ver que simplemente va incrementando en 1 el resultado de la función anterior (módulo m para no irnos del tamaño de la tabla), es decir, si el casillero está ocupado, prueba con el siguiente hasta encontrar el primer casillero libre.



El problema que podemos marcar rápidamente es que cuando el factor de carga es alto este método resulta ser muy lento.

Colisiones - Direccionamiento Abierto - Linear probing

Un punto que es importante analizar es que a medida que la tabla se va llenando, se observa que empiezan a aparecer *clusters* de casilleros ocupados consecutivos:



Colisiones - Direccionamiento Abierto - Linear probing

Si la función de hash distribuye los elementos uniformemente dentro de la tabla, la probabilidad que un cluster crezca es proporcional a su tamaño.
Esto significa que una mala situación se vuelve cada vez peor con mayor probabilidad. Esto se conoce como *clustering primario*.

Sin embargo este no es todo el problema, puesto que lo mismo sucede en hashing con encadenamiento y no es tan malo. El verdadero problema ocurre cuando 2 clusters están separados solo por un casillero libre y ese casillero es ocupado por algún elemento: ambos clusters se unen en uno mucho más grande. Esto genera clusters que dificultan las operaciones.

Colisiones - Direccionamiento Abierto - Linear probing

Otra característica, menos grave que la anterior, que se genera con Linear probing es conocida como *clustering secundario*.
El mismo consiste en lo siguiente: supongamos que al realizar la búsqueda de dos elementos en la tabla se encuentran con el mismo casillero ocupado, entonces toda la búsqueda subsiguiente es la misma para ambos elementos.

Colisiones - Direccionamiento Abierto - Linear probing

Eliminar un elemento es complejo. ¿Por qué? Vamos a analizarlo.

Análogamente a lo que sucedía en Hashing con Listas Mezcladas donde debíamos reenlazar las listas al eliminar un elemento, acá no se puede eliminar un elemento y simplemente dejar su casillero vacío porque las búsquedas terminarían en dicho casillero.

¿Qué podemos hacer para resolver esto? Vamos a ver dos posibles soluciones.

Colisiones - Direccionamiento Abierto - Linear probing

Existen dos maneras para eliminar elementos de la tabla en este caso:

- Marcar el casillero como "eliminado", pero sin liberar el espacio. Esto produce que las búsquedas puedan ser lentas incluso si el factor de carga de la tabla es pequeño.
- Eliminar el elemento, liberar el casillero y mover elementos dentro de la tabla hasta que un casillero "verdaderamente" libre sea encontrado. Implementar esta operación es complejo y costoso.

Colisiones - Direccionamiento Abierto - Hashing Doble

Esta variante de Direccionamiento Abierto usa dos funciones hash:

1. una función conocida como *dirección inicial h,* tal que $h(X) \in [0, ..., m-1]$
2. y una función conocida como *paso s,* tal que $s(X) \in [1, ..., m-1]$

Con ellas construimos la secuencia de funciones de la siguiente forma:

$$h_0(X)=h(X)$$
$$h_{k+1}(X)=(h_k(X)+s(X)) \ mod \ m$$

Colisiones - Direccionamiento Abierto - Hashing Doble

Podemos ver que cuando $s(X)=1$ para todo X tenemos Linear probing.

Elegir m primo asegura que se va a visitar toda la tabla antes que se empiecen a repetir los casilleros. Obervación: sólo basta que m y $s(X)$ sean primos relativos.

Colisiones - Direccionamiento Abierto - Hashing Doble

Existen heurísticas para resolver el problema de las colisiones en hashing con direccionamiento abierto, como por ejemplo *last-come-first-served* hashing (el elemento que se mueve de casillero no es el que se inserta sino el que ya lo ocupaba) y *Robin Hood* hashing (el elemento que se queda en el casillero es aquel que se encuentre más lejos de su posición original), que si bien mantienen el promedio de búsqueda con respecto al método original (*first-come-first-served*) disminuyen notablemente su varianza.