Informe sobre Tablas Hash

Adrián Quiroga Linares, Alberto Rodrigo Torres Montes

Introducción

Las **tablas hash** son estructuras de datos que permiten almacenar y recuperar información de manera eficiente. Su principal ventaja es que proporcionan un acceso en tiempo constante **O(1)**, lo que las hace ideales para aplicaciones como bases de datos, cachés y sistemas de búsqueda.

Un problema a gestionar en ellas son las **colisiones** a la hora de buscar o insertar un elemento en la tabla. Estas suceden cuando dos claves distintas generan el mismo índice en la tabla, lo cual implica tener que realizar pasos extra en ese proceso de búsqueda o inserción. Para ello existen dos estrategias:

- **Encadenamiento:** Cada posición de la tabla apunta a una lista enlazada donde se almacenan todas las claves que colisionan en esa posición.
- **Recolocación:** Se busca otra posición libre dentro de la tabla siguiendo una estrategia predeterminada. Nótese que cada intento fallido para encontrar una posición libre se considera una operación extra. En este informe, utilizamos recolocación lineal con $a = 1$ en la mayoría de casos.

Otros dos conceptos fundamentales en las tablas hash son la clave y el **factor de carga** (L). La primera es el valor que se pasa a la función hash para calcular su posición en la tabla, mientras que el segundo se define como la relación entre el número de elementos almacenados y el tamaño de la tabla:

$$\mathrm{L} = \frac{\text{número de elementos}}{\text{tamaño de la tabla}}$$

Un factor de carga óptimo es esencial para minimizar las colisiones y maximizar el rendimiento. Los valores recomendados varían en función de la estrategia, siendo $L \leq$ **0.5** para **recolocación** y $L \leq$ **0.75** para **encadenamiento**

En este informe probaremos distintos tamaños de tabla para ver la diferencia de edificiencia: valores cercanos a los recomendados (teniendo en cuenta que tenemos 10,000 elementos a almacenar serían 20,011 para recolocación y 13,337 para encadenamiento), valores mayores a estos para disminuir L (y mejorar el rendimiento) y valores menores a estos para superar los valores de L recomendados y comprobar cómo baja la eficiencia. Además, también probaremos con números primos cercanos a esos valores, puesto que distribuyen mejor las claves, reduciendo las colisiones.

Primer Experimento: Influencia del Tamaño de la Tabla Hash en el Proceso de Inserción

Implementación de los Contadores

1. **Encadenamiento:**
   - Se añadió un contador $n_{\text{ColisionesI}}$ que se incrementa cada vez que un elemento se inserta en una lista enlazada no vacía (y por tanto produce colisión).
2. **Recolocación:**
   - Se añadió un contador $n_{\text{ColisionesI}}$ para registrar cada colisión.
   - Se añadió un contador $n_{\text{OperacionesExtraI}}$ para registrar cada intento adicional al buscar una posición libre en la tabla durante la recolocación.

Ambos contadores se inicializan a cero y se actualizan dentro del proceso de inserción para cada elemento.

Resultados del Experimento

Table 1: Influencia de N en el proceso de inserción con Encadenamiento

Función hash2	**N=13337**	**N=20011**	**N=30011**	**N=10000**	**N=15000**	**N=25000**
$n_{\textbf{ColisionesI}}$	2977	2200	1460	9375	8133	7009

Table 2: Influencia de N en el proceso de inserción con Recolocación Simple (a=1)

Función hash2	**N=20011**	**N=30011**	**N=40009**	**N=10000**	**N=15000**	**N=25000**
$n_{\textbf{ColisionesI}}$	2587	1635	1234	9375	8136	7010
$n_{\textbf{OperacionesExtraI}}$	5540	2411	1701	672632	29321	16004

Análisis de Resultados

- **Con/sin números primos:**
  - Las tablas con tamaños **primos** producen menos colisiones en ambos métodos, ya que reducen patrones de agrupamiento al distribuir mejor las claves.
  - Los tamaños **no primos** generan más colisiones debido a distribuciones más uniformes.
- **Influencia del factor de carga (λ):**
  - En **encadenamiento**, cuando $L \leq 0.75$, las colisiones se reducen significativamente a medida que L decrece.
  - En **recolocación**, cuando $L \leq 0.5$, se observan menos colisiones y operaciones extra a medida que L decrece.

Los resultados confirman que el uso de números primos y factores de carga óptimos son fundamentales para maximizar el rendimiento de las tablas hash. También podemos apreciar que cuando el factor de carga supera los márgenes preestablecidos el número de colisiones y pasos extras se incrementa en gran medida en ambas tablas.

Segundo Experimento: Influencia de la función hash y de la clave en el proceso de Inserción

Resultados Obtenidos

A continuación, se presentan las tablas con los resultados obtenidos en el segundo experimento. Estas tablas muestran la influencia de las funciones hash y los tamaños de las tablas sobre el número de colisiones y pasos adicionales durante el proceso de inserción, considerando diferentes configuraciones de clave y métodos de resolución de colisiones.

Table 3: Influencia de la función hash en el proceso de inserción en Encadenamiento Clave=nickname

Clave = nickname	**N=13337**	**N=20011**	**N=30011**	**N=10000**	**N=15000**	**N=25000**
nColisionesI Hash1	9271	9271	9271	9271	9271	9271
nColisionesI Hash2	2977	2200	1460	9375	8133	7009
nColisionesI Hash3 K=500	2984	2125	1574	9982	9970	9979
nColisionesI Hash3 K=97	3027	2131	1446	3665	2675	1758

Table 4: Influencia de la función hash en el proceso de inserción en Recolocación Lineal (a=1) Clave=nickname

Clave = nickname	**N=20011**	**N=30011**	**N=40009**	**N=10000**	**N=15000**	**N=25000**
nColisionesI Hash1	9784	9784	9784	9784	9784	9784
nPasosExtraI Hash1	46720557	46720557	46720557	46720557	46720557	46720557
nColisionesI Hash2	2587	1635	1234	9375	8136	7010
nPasosExtraI Hash2	5540	2411	1701	672632	29321	16004
nColisionesI Hash3 K=500	2506	1771	1229	9982	9970	9979
nPasosExtraI Hash3 K=500	4991	2771	1623	16904000	2362037	12605708
nColisionesI Hash3 K=97	2503	1600	1245	4921	3293	1995
nPasosExtraI Hash3 K=97	4779	2364	1605	453005	8946	3394

Table 5: Influencia de la función hash en el proceso de inserción en Encadenamiento Clave=correo

Clave = correo	**N=13337**	**N=20011**	**N=30011**	**N=10000**	**N=15000**	**N=25000**
nColisionesI Hash1	8787	8787	8787	8787	8787	8787
nColisionesI Hash2	2974	2153	1465	4550	3045	2068
nColisionesI Hash3 K=500	2927	2138	1519	9951	9871	9946
nColisionesI Hash3 K=97	2926	2114	1484	3654	9999	1729

Table 6: Influencia de la función hash en el proceso de inserción en Recolocación Lineal (a=1) Clave=correo

Clave = correo1	**N=20011**	**N=30011**	**N=40009**	**N=10000**	**N=15000**	**N=25000**
nColisionesI Hash1	9485	9485	9485	9485	9485	9485
nPasosExtraI Hash1	43756551	43756551	43756551	43756551	43756551	43756551
nColisionesI Hash2	2533	1616	1230	5927	3678	2348
nPasosExtraI Hash2	5174	2382	1668	935470	10535	3947
nColisionesI Hash3 K=500	2538	1681	1235	9956	9910	9952
nPasosExtraI Hash3 K=500	4953	2483	1609	20765418	5411192	12460501
nColisionesI Hash3 K=97	2481	1647	1324	5027	3345	1978
nPasosExtraI Hash3 K=97	4858	2442	1707	695625	9694	3384

Influencia de la Función Hash en el Número de Operaciones para la Inserción

La función hash juega un papel determinante en cómo se distribuyen las claves en las posiciones de la tabla, lo que afecta directamente al número de colisiones durante la inserción.

Las tablas utilizadas en los experimentos incluyen diferentes tamaños N y diferentes funciones hash, lo que permite analizar cómo la elección de la función hash afecta el comportamiento del número de colisiones y, por ende, la eficiencia del proceso de inserción.

Hash1

La función **Hash1** genera un número de colisiones constante a medida que se varía el tamaño de la tabla N independientemente de si se usa como clave el *nickname* o el *correo*. Este comportamiento indica que la función hash no está dispersando bien las claves y que, independientemente del tamaño de la tabla, siempre se generan muchas colisiones. Esto sugiere que Hash1 produce una distribución de claves muy concentrada, lo que lleva a un mayor número de operaciones durante la inserción.

Hash2

En contraste, la función **Hash2** muestra un comportamiento más favorable. A medida que aumenta el tamaño de la tabla, el número de colisiones disminuye significativamente. Esto indica que Hash2 dispersa las claves de manera más uniforme, lo que mejora la eficiencia del proceso de inserción. Sin embargo, aunque mejora respecto a Hash1.

Hash3 (K=500 y K=97)

La función **Hash3**, cuando se utiliza con parámetros $K = 500$ y $K = 97$, muestra una mejora significativa en la dispersión de las claves en comparación con Hash1. Aunque las colisiones no desaparecen por completo, los números son mucho más bajos que con Hash1. Además, se observa que el valor de K puede influir en la eficiencia del hash, ya que variaciones en este valor resultan en diferentes comportamientos. Por ejemplo, al usar $K = 500$, las colisiones tienden a ser más bajas, pero aún hay un cierto nivel de colisiones. En general, Hash3 proporciona una mejor distribución de claves, reduciendo el número de colisiones, aunque estas siguen siendo una parte importante del proceso de inserción. Sin embargo en nuestro caso las diferencias entre **Hash2**, **Hash3** con $K = 500$ y **Hash3** con $K = 97$ no son muy notables (por norma general 500 es ligeramente mejor).

Análisis en Función de los Datos Usados como Clave

El tipo de clave utilizada también influye en el número de colisiones. En las tablas comparadas, las claves utilizadas son *nickname* y *correo*, y su efecto en el número de colisiones es notablemente distinto.

- **Clave = nickname:** En las tablas 3 y 4, las colisiones con *nickname* tienden a ser más altas cuando se utiliza la función Hash1, lo que refleja que las claves generadas por *nickname* no están distribuidas uniformemente en la tabla hash. A medida que se usan funciones hash más eficientes como Hash2 y Hash3, las colisiones disminuyen considerablemente, pero el tipo de clave sigue teniendo un

impacto, ya que el *nickname* es más corto y menos variado, lo que resulta en una menor dispersión de las claves en comparacion a usar *correo*.

- **Clave = correo:** En las tablas 5 y 6, las colisiones con *correo* siguen un patrón similar al de *nickname*, aunque las claves de *correo* son más largas y tienen más variabilidad. Esto generar una mayor dispersión. Sin embargo, las diferencias en las colisiones entre *nickname* y *correo* no son muy significativas, lo que sugiere que la longitud de la clave no es un factor muy determinante en el número de colisiones (aunque sea mejor en este caso el *correo*). Podemos concluir que la función hash utilizada juega un papel más importante en la reducción de las colisiones, como se observa con Hash2 y Hash3, que la clave escogida en este caso.

Tercer Experimento: Influencia de la estrategia de recolocación en el proceso de inserción

Para esta experimentación, se seleccionan dos combinaciones de **tamaño de tabla** y **función hash** que presentaron un buen rendimiento en cuanto a la cantidad de colisiones y pasos extra en las pruebas anteriores. Estas combinaciones se compararán para analizar cómo influye la estrategia de recolocación en el proceso de inserción, específicamente en los valores de **nColisionesI** y **nPasosExtraI**.

Selección de combinaciones de tamaño y función hash

- **Caso 1:**
  - **Tamaño:** 40009
  - **Función Hash:** Hash3 con $K = 97$
  - Justificación: Esta combinación mostró un buen comportamiento en las pruebas anteriores, con una reducción consistente en las colisiones al aumentar el tamaño de la tabla.
- **Caso 2:**
  - **Tamaño:** 30011
  - **Función Hash:** Hash2
  - Justificación: Esta combinación presentó buenos resultados con respecto a la reducción de colisiones y un rendimiento eficiente en pasos extra.

Estrategias de recolocación

Se investigará el comportamiento de las siguientes estrategias de recolocación:

- **Recolocación Simple (lineal con a=1):** estrategia básica que busca la siguiente posición disponible al sumar una constante fija (a=1) al índice hash original.
- **Recolocación Lineal (con diferentes valores de a):** se evalúan dos valores distintos de la constante a para observar su impacto en las colisiones y los pasos extra.

- **Recolocación Cuadrática:** estrategia que incrementa el valor de a de forma cuadrática, ayudando a evitar agrupaciones fuertes y a distribuir las claves más uniformemente.

Resultados experimentales

Table 7: Influencia de la estrategia de recolocación en el proceso de inserción

Casos / Variables	**Simple (a=1)**	**Lineal (a=73)**	**Lineal (a=13)**	**Cuadrática**
Caso 1: nColisionesI	1245	1251	1231	1238
Caso 1: nPasosExtraI	1605	1689	1645	1531
Caso 2: nColisionesI	1635	1658	1631	1627
Caso 2: nPasosExtraI	2411	2693	2378	2288

Justificación de los resultados

- En **Caso 1**, los resultados muestran que la estrategia de recolocación cuadrática no reduce significativamente el número de colisiones en comparación con las estrategias lineales. Sin embargo, logra disminuir notablemente el número de pasos extra, indicando una mejor distribución de claves una vez que ocurre una colisión. Esto evidencia que, aunque no minimiza la frecuencia de colisiones, la recolocación cuadrática es más eficiente en términos de manejo posterior.

- En **Caso 2**, los resultados indican que las diferencias en el número de colisiones entre las estrategias son mínimas. La recolocación cuadrática presenta un comportamiento competitivo, reduciendo tanto el número de colisiones como los pasos extra en comparación con las estrategias lineales. Esto sugiere que, con un tamaño de tabla menor y una función hash menos eficiente, la recolocación cuadrática aprovecha mejor el espacio disponible, distribuyendo las claves de forma más uniforme.

Cuarto Experimento: Estimación de la eficiencia en el acceso a los datos

En este último apartado se repetirán experimentaciones pasadas centrándose ahora en el número de pasos requeridos en la búsqueda de los elementos en la tabla. Se buscarán todos los elementos una vez hayan sido insertados y se comprobará el número de operaciones extra para encontrar cada uno.

Para ello se añadió contador $n_{\text{OperacionesExtraB}}$ que lleva el registro de esos pasos a mayores que hay que realizar cuando se produce una colisión durante la búsqueda de un dato. Nótese que estas colisiones se producen por distintos motivos según la estrategia: en encadenamiento se dan cuando el dato no es el primero de la lista en esa posición, mientras que en recolocación se dan cuando el dato fue recolocado y por tanto no está en la posición indicada.

Resultados experimentales

Table 8: Influencia de N en el proceso de búsqueda en Encadenamiento ($n_{OperacionesExtraB}$)

Encadenamiento	**N=13337**	**N=20011**	**N=30011**	**N=10000**	**N=15000**	**N=25000**
Hash1	129176	129176	129176	129176	129176	129176
Hash2	3820	2605	1610	79333	26752	15952
Hash3 K=500	3763	2465	1773	4397778	2014065	4109407
Hash3 K=97	3860	2516	1592	4957	3246	2014

Table 9: Influencia de N en el proceso de búsqueda en Recolocación Simple (*nOperacionesExtraB*)

(*a*=1)	**N=20011**	**N=30011**	**N=40009**	**N=10000**	**N=15000**	**N=25000**
Hash1	46720557	46720557	46720557	46720557	46720557	46720557
Hash2	5540	2411	1701	672632	29321	16004
Hash3 K=500	4991	2771	1623	16904000	2362037	12605708
Hash3 K=97	4779	2364	1605	453005	8946	3394

Table 10: Influencia de N en el proceso de búsqueda en Recolocación Cuadrática (*nOperacionesExtraB*)

R. Cuadrática	**N=20011**	**N=30011**	**N=40009**	**N=10000**	**N=15000**	**N=25000**
Hash1	635029	635029	635029	681312	635029	635029
Hash2	4582	2288	1615	19166	50925	100655
Hash3 K=500	4341	2545	1572	342852	312382	2313314
Hash3 K=97	4250	2213	1531	64788	6879	3041

Análisis de los resultados

- Para la estrategia de **Encadenamiento** la función Hash1 obtiene un número constante de operaciones extra sin importar el tamaño de la tabla (como va a hacer con el resto de estrategias), pero es un valor muy alto en comparación con el resto de funciones. Para el resto de funciones se nota claramente la importancia de usar un número primo como tamaño de tabla (con una gran diferencia para Hash3 y K=500), y como también se van reduciendo las operaciones conforme aumentamos el tamaño de la misma. La que obtiene mejores resultados es Hash3 con K=97 pero tanto Hash2 como Hash3 (K=500) se acercan razonablemente.

- Para la estrategia de **Recolocación Simple** la función Hash1 tiene unos resultados pésimos. Si comparamos el resto de parámetros tenemos un razonamiento parecido a Encadenamiento: fundamental que el tamaño de la tabla sea un número primo que permita llegar o superar al factor de carga recomendado, y todas las funciones menos Hash1 presentan un número de operaciones extra de búsqueda parecidos para cada valor de N primo. No obstante, en la mayoría de los casos la estrategia de Encadenamiento obtenía ligeramente mejores resultados.

- La estrategia de **Recolocación Cuadrática** es la que mejor trabaja con la función Hash1 en comparación con las demás, pero sigue tienendo valores excesivos respecto

del resto de funciones. También es la menos sensible a la variación entre valores primos de N o valores no primos, pero esto no quita lo fundamental de escoger un número primo. El resto de parámetros influyen de manera análoga a las otras estrategias, obteniendo su mejor resultado al usar N=40009 y Hash3 (K=97), pero siendo Hash2 y Hash3 (K=500) opciones a tener en cuenta.

Conclusiones

Después de este análisis sobre cómo afectan distintos parámetros al rendimiento de los HashMaps, exponemos lo siguientes puntos:

1. Tamaño de tabla
    - Fundamental usar números primos para obtener una mejor distribución de las claves, como valor mínimo uno que satisfaga el valor de carga recomendado en la estrategia concreta, pero siempre que se pueda aumentar mejorará la eficiencia.
2. Estrategia y función hash
    - Tanto usar Encadenamiento con la función Hash2 como usar Recolocación Cuadrática con la función Hash3 (y K=97) son las que alcanzan mejores resultados, mejorando la segunda ligeramente respecto de la primera.
3. Clave
    - Desde luego es el factor que menos varía el rendimiento pero ofrece una ligera mejora en el proceso de inserción usar el correo respecto del nickname como clave.

Estos factores afectan de manera parecida tanto al proceso de inserción como al de búsqueda. Sin embargo en este último hay que tener un factor más en cuenta: la relación entre el tiempo de búsqueda con el tamaño de la memoria que se usa. Como ambas estrategias (recolocación y encadenamiento) tienen un tiempo medio de **O(1)** el valor decisivo va a ser el tamaño de la memoria.

Siendo N el tamaño de la tabla Hash y n el número de elementos a insertar, podemos calcular el tamaño de la memoria de la siguiente manera:

$$\text{tamañoMemoria (Encadenamiento)} = n \cdot (\text{tamClave} + \text{tamPunt}) + N \cdot (\text{tamPunt})$$

$$\text{tamañoMemoria (Recolocación)} = 2n \cdot \text{tamClave}$$

Ahora se puede apreciar la importancia de escoger bien la clave y ajustar el tamaño de la tabla. Para optimizar la memoria, y teniendo en cuenta que en el resto de operaciones ofrecen resultados similares, se debe escoger el nickname como clave. El tamaño de tabla, sin embargo, sólo influye a la memoria en Encadenamiento, teniendo en este caso que si el tamaño de tabla ofreciese un valor de carga correcto aumentarlo implicaría aumentar la memoria, y por tanto no se recomienda.

En nuestro caso tenemos que n=10000, estimamos que los nicknames tienen una longitud media de 8 caracteres (que equivalen a 8 bytes) y el tamaño de los punteros puede ser de 4 u 8 bytes (consideremos 4). Aún para los valores más ajustados de N que garantizan un factor de carga correcto para Encadenamiento no se llega al tamaño de memoria constante que ofrece la Recolocación. Para N=13337 recolocación tiene un coste de memoria de $173,348$ bytes, mientras que Encadenamiento siempre presenta un coste de memoria de $160,000$ bytes.

Teniendo todos los factores expuestos en cuenta se concluye que la mejor implementación para este caso concreto sería:

- Recolocación Cuadrática
- Función Hash3 (K=97)
- N=40009
- Usar el nickname como clave