Análisis de tablas hash

Salvia Sisa Cortés y Sergio Souto Mourelle

Algoritmos y Estructuras de Datos - Curso 2024-2025
Grado en Ingeniería Informática
Universidade de Santiago de Compostela

Índice de contenidos

1 Introducción

1.1. Sobre las colisiones

En este informe vamos a comparar la eficiencia de las distintas **técnicas de resolución de colisiones en tablas hash: recolocación y encadenamiento**. Una **colisión** se produce cuando la función hash otorga a dos claves distintas la misma dirección; y se resuelve de forma distinta según se use una estrategia u otra. En **recolocación**, cuando se produce una colisión, se prueba con otra posición distinta dada por el tipo de recolocación utilizado (simple, lineal o cuadrática). Por otra parte, en el caso de resolver una colisión por **encadenamiento**, al momento de producirse esta, el dato se coloca en una lista enlazada en la posición otorgada por la función hash. Dado que la resolución de colisiones es computacionalmente **costosa**, resulta de gran importancia determinar con qué alternativa y con qué función hash se disminuye al mínimo el impacto en el rendimiento del sistema.

Para esto, nos situamos en el marco de una **base de datos** que almacena información sobre los **usuarios de un juego en línea basado en la serie Juego de Tronos**. Así, disponemos de un archivo que contiene el nombre y apellidos, nickname y correo electrónico asociado de cada uno de los jugadores. Para almacenar los jugadores en la tabla, emplearemos como **clave** el campo del **nickname de usuario/a**.

1.2. Modificaciones en el código

Al inicio **partimos de dos TADs**: uno para la implementación de tablas hash con estrategia de recolocación, y otro con estrategia de encadenamiento. A estos TADs les añadimos **modificaciones** para la contabilización tanto de las colisiones como de los pasos extras que debe realizar el código.
Variables globales añadidas a efecto de registrar datos relacionados con el proceso de inserción:

- `nColisionesI`: Es un contador que registra el **número de colisiones** producidas al obtener el hash de un elemento, y se calcula tanto en la estrategia de **encadenamiento** como en la de **recolocación**.
- `nPasosExtraI`: Mide los **pasos adicionales** necesarios para insertar un dato en la tabla hash en caso de colisión.
    - **Recolocación:** En esta estrategia, cuando ocurre una colisión, se busca una posición libre, lo que puede requerir varios intentos. Cada intento adicional se contabiliza para determinar cuántos **pasos extra** son necesarios para ubicar el dato (`nPasosExtraI++`).
    - **Encadenamiento:** Dado que en esta estrategia el dato se inserta directamente al principio de la lista enlazada correspondiente a su hash, **no existen pasos adicionales**.

Además, también añadimos una **variable global** para registrar los **pasos adicionales** necesarios en caso de que se produzca una colisión en el proceso de **búsqueda**: `nPasosExtraB`.

- **Recolocación**: Si el dato no está en la posición indicada por el hash, entonces significa que puede haber sido recolocado a otra posición libre, por lo que se cuentan los intentos adicionales necesarios para localizar el dato.
- **Encadenamiento**: Si el dato no está primero en la lista enlazada de esa posición, se recorren los elementos de la lista hasta encontrar el dato.

Así, estas nuevas variables son las que les **pasamos por referencia** a las **funciones** de los TADs que lo requieran: por ejemplo, en las funciones de inserción y búsqueda. Se les pasa como **punteros** para que el valor sea actualizado de forma consistente por todas las funciones del TAD cuando sea preciso.

Una vez llevadas a cabo las modificaciones anteriormente descritas, hicimos uso del código para ir probando las funciones de ambas estrategias con la finalidad de dilucidar cuál es la combinación más eficiente para nuestro caso.

2 Impacto del tamaño de la tabla hash (N) en la inserción

2.1. Elección de tamaños y elaboración de la tabla

En este apartado vamos a analizar el comportamiento de las implementaciones con encadenamiento y con recolocación simple (lineal con $a = 1$) en lo que a **número de colisiones** (`nColisionesI`) y **operaciones extra** requeridas en la **inserción** (`nOperacionesExtraI`) respecta.

Para ello insertamos todos los datos del archivo en la tabla correspondiente y probamos la **función hash2**, que consiste en la **suma ponderada** de los valores ascii de los caracteres (interpretando la cadena como un entero en base $K = 256$), con distintos tamaños (N) de tabla. Para ello, tuvimos en cuenta el **factor de carga** L, que se define como $L = n/N$ tal que N es el tamaño de la tabla y n el número de datos (que en nuestro caso será siempre $n = 10{,}000$). Dado que para encadenamiento se recomienda un factor de carga $L \leq 0,75$ y que para recolocación la recomendación es de $L \leq 0,5$, la selección de los valores de N estará significativamente condicionada por estas restricciones. Además, también tuvimos en cuenta el empleo de números primos para N, ya que, cuando este parámetro toma valores no primos, da lugar a más colisiones debido a que tiene un mayor número de divisores, por lo que habrá más números que sean congruentes módulo N.

Teniendo en cuenta lo anterior, para encadenamiento, primero seleccionamos $N = 10{,}001$, lo que nos proporciona un factor de carga L muy cercano a 1, por lo que debería resultar muy poco óptimo. Luego seleccionamos $N = 13{,}381$, por ser un número primo que nos proporciona un factor de carga $L \simeq 0,75$, que es considerado como el máximo factor de carga aceptable para el método de encadenamiento. Además, con el objetivo de poner a prueba lo que ocurre cuando N no es un número primo, probamos con $N = 13{,}500$. Por último probamos con varios valores de N primos sucesivamente más grandes hasta llegar a $N = 19{,}997$.

Luego, para recolocación simple empezamos escogiendo N = 20,047, ya que es el primo siguiente a 20.000, por lo que el factor de carga es ligeramente inferior a 0,5. Al igual que en el caso de encadenamiento, también probamos con un valor no primo de N (N = 23,000) y para valores de N primos sucesivamente mayores, siendo el más elevado N = 40,009.

Encadenamiento	**N=10.001**	**N=13.381**	**N=13.500**	**N=15.017**	**N=16.979**	**N=19.997**
nColisionesI	3.654	2.931	6.780	2.679	2.492	2.083
Recolocación simple (a=1)	**N=20.047**	**N=23.000**	**N=23.017**	**N=30.029**	**N=35.027**	**N=40.009**
nColisionesI	2.469	7.213	2.134	1.636	1.417	1.234
nOperacionesExtraI	5.110	17.125	3.978	2.399	1.961	1.701

Cuadro 1: Influencia del tamaño de N en el proceso de inserción usando la función hash2

2.2. Conclusiones

Para empezar, lo más evidente es que cuando N no es un número primo, tanto para encadenamiento como para recolocación el número de colisiones y de operaciones extra, en el caso de recolocación simple, aumenta significativamente con respecto a cuando N toma valores primos. Además, también observamos que tanto el número de **colisiones** como el número de **operaciones** adicionales **decrece** significativamente al **aumentar el tamaño** de la tabla N en ambas estrategias. Así, concluímos que **el tamaño de N que tiene mejor comportamiento** para ambas estrategias es aquel que haga que $L \leq 0,5$, por lo que se trata de un **valor bastante alto** ($N \geq 2n$).

3 Influencia de la función hash y de la clave en la inserción

En esta sección vamos a discutir cómo influye la selección de las distintas **funciones hash** en cuanto al número de colisiones producidas en el proceso de inserción (`nColisionesI`) tanto para encadenamiento como para recolocación lineal. Para esto comprobaremos para los valores de N seleccionados anteriormente el número de colisiones con las siguientes funciones:

- **Hash1 (método de la división)**: se realiza la suma de los valores ascii de los caracteres de la cadena módulo N
- **Hash2 (suma ponderada con $K = 256$)**: se interpreta la cadena como un entero en base $K = 256$, es decir, se itera caracter a caracter (de fin a principio) tal que $suma = suma * K + cad[i](modN)$.
- **Hash3 (suma ponderada con otro valor de K)**: igual que la función hash2 pero con otro valor de K

Además, también vamos a probar a emplear distintos campos como clave, para determinar qué influencia tiene la elección de esta.

3.1. Con la utilización del nickname como clave

Primero probamos a utilizar el nombre de usuario como clave y obtuvimos las siguientes tablas:

	N=10.001	**N=13.381**	**N=13.500**	**N=15.017**	**N=16.979**	**N=19.997**
nColisionesI Hash1	9.271	9.271	9.271	9.271	9.271	9.271
nColisionesI Hash2	3.654	2.931	6.780	2.679	2.492	2.083
nColisionesI Hash3 K=500	3.574	3.005	9.973	2.657	2.448	2.055
nColisionesI Hash3 K=128	3.718	2.997	6.814	2.644	2.433	2.144

Cuadro 2: Influencia de la función hash en el proceso de inserción en Encadenamiento

	N=20.047	**N=23.000**	**N=23.017**	**N=30.029**	**N=35.027**	**N=40.009**
nColisionesI Hash1	9.784	9.784	9.784	9.784	9.784	9.784
nPasosExtraI Hash1	46.720.557	46.720.557	46.720.557	46.720.557	46.720.557	46.720.557
nColisionesI Hash2	2.469	7.213	2.134	1.636	1.417	1.234
nPasosExtraI Hash2	5.110	17.125	3.978	2.399	1.961	1.701
nColisionesI Hash3 K=500	2.389	9.954	2.145	1.646	1.409	1.229
nPasosExtraI Hash3 K=500	4.586	1.086.698	3.732	2.363	1.930	1.623
nColisionesI Hash3 K=128	2.481	7.226	2.155	1.625	1.467	1.296
nPasosExtraI Hash3 K=128	4.823	17.720	3.625	2.349	2.034	1.734

Cuadro 3: Influencia de la función hash en el proceso de inserción en Recolocación Lineal

Como podemos observar claramente, al emplear el **método de la división**, tanto el numero de **colisiones** como el número de **operaciones extra** permanece **constante** en ambas estrategias (siendo este último desmesuradamente grande), por lo que sería acertado suponer que esta función no es la más óptima en este caso. Esto se debe a que en este caso tenemos un valor de **N demasiado grande** como para que sea útil esta función hash.

Asimismo, podemos contrastar con el método de la **suma ponderada**, que logra que el número de colisiones y de pasos extra disminuya considerablemente a medida que aumenta N (siempre que N sea primo), siendo el valor de $K = 500$ mejor en ambas estrategias.

3.2. Con la utilización del correo como clave

A efectos de contraste, también realizamos la prueba empleando el correo como clave, para discernir su influencia. Al igual que en el caso anterior, elaboramos las siguientes tablas análogas:

	N=10.001	**N=13.381**	**N=13.500**	**N=15.017**	**N=16.979**	**N=19.997**
nColisionesI Hash1	8.787	8.787	8.787	8.787	8.787	8.787
nColisionesI Hash2	3.701	2.988	3.167	2.693	2.433	2.121
nColisionesI Hash3 K=500	3.648	2.926	9.435	2.713	2.389	2.121
nColisionesI Hash3 K=128	3.690	2.909	3.176	2.662	2.410	2.175

Cuadro 4: Influencia de la función hash en el proceso de inserción en Encadenamiento

	N=20.047	N=23.000	N=23.017	N=30.029	N=35.027	N=40.009
nColisionesI Hash1	9.485	9.485	9.485	9.485	9.485	9.485
nPasosExtraI Hash1	43.756.551	43.756.551	43.756.551	43.756.551	43.756.551	43.756.551
nColisionesI Hash2	2.525	2.543	2.179	1.730	1.401	1.230
nPasosExtraI Hash2	4.821	4.657	3.968	2.572	1.931	1.668
nColisionesI Hash3 K=500	2.546	9.665	2.178	1.656	1.423	1.235
nPasosExtraI Hash3 K=500	5.129	1.145.410	3.784	2.464	1.930	1.609
nColisionesI Hash3 K=128	2.502	2.483	2.119	1.634	1.378	1.258
nPasosExtraI Hash3 K=128	4.892	4.432	3.730	2.455	1.937	1.677

Cuadro 5: Influencia de la función hash en el proceso de inserción en Recolocación Lineal

En ambas estrategias se observa una **mejoría** en el número de operaciones y de pasos extra al usar el correo electrónico como clave en comparación al uso del nickname. Esto se debe principalmente a que el uso del correo electrónico conlleva una **distribución más uniforme** de las claves en la tabla hash, lo que reduce las colisiones. Cabe destacar que esta mejoría es algo más notoria en el caso del encadenamiento (Cuadro 4).

4 Influencia de estrategia de recolocación en el proceso de inserción

En esta sección escogeremos dos combinaciones de tamaño de tabla N y de función hash que hayan dado buen resultado en el ejercicio anterior (que hayan producido el menor número de colisiones) y las probaremos con **distintos tipos de estrategia de recolocación** (simple, lineal con varios valores de a y cuadrática). Para la elección de valores escogimos las combinaciones que menos colisiones ocasionaron en el apartado anterior: $N = 35{,}027$ con la función Hash3 ($K = 500$) y $N = 40{,}009$ con la función Hash2. Así, obtuvimos la siguiente tabla:

Casos	Variables	Simple (lineal con a=1)	Lineal (a=5)	Lineal (a=20)	Cuadrática
N=35.027 con Hash3 (K=500)	nColisionesI	1.409	1.408	1.403	1.406
	nPasosExtraI	1.930	1.983	1.886	1.864
N=40.009 con Hash2	nColisionesI	1.234	1.225	1.215	1.220
	nPasosExtraI	1.701	1.609	1.658	1.615

Cuadro 6: Influencia de la recolocación en el proceso de inserción

A la vista de los datos mostrados en la tabla, podemos denotar como **apenas hay diferencia** entre los tres tipos de recolocación, siendo muy cercanos en eficiencia la lineal con $a = 20$ y la cuadrática. Lo más notorio es que en ambos casos, con la cuadrática se logra reducir el número de pasos adicionales en casi 100 unidades.

5 Estimación de la eficiencia en el acceso a los datos

En esta sección compararemos el número de pasos requeridos en la **búsqueda** de los elementos de la tabla (`nOperacionesExtraB`). A efectos de este estudio pues, repetimos la experimentación de los cuadros 1,2, 3 y 6 para analizar el parámetro que contabiliza las operaciones extra en el caso de búsqueda, obteniendo esta tabla:

	N=10.001	N=13.381	N=13.500	N=15.017	N=16.979	N=18.517
Encadenamiento	10.001	13.381	13.500	15.017	16.979	18.517
Recolocación	20.047	23.017	23.000	30.029	35.027	40.009
Enc. Hash1	129.176	129.176	129.176	129.176	129.176	129.176
Enc. Hash2	4.946	3.707	3.269	3.059	2.760	14.511
Enc. Hash3 (K=500)	4.755	3.823	3.251	2.949	2.600	1.849.940
Enc. Hash3 (K=128)	5.036	3.884	3.197	2.911	2.597	14.763
Rec. Simple Hash1	46.720.557	46.720.557	46.720.557	46.720.557	46.720.557	46.720.557
Rec. Simple Hash2	5.110	3.978	17.125	2.399	1.961	1.701
Rec. Simple Hash3 (K=500)	4.586	3.732	1.086.698	2.363	1.930	1.623
Rec. Simple Hash3 (K=128)	4.823	3.625	17.720	2.349	2.034	1.734
Rec. Cuadratica Hash1	635.029	635.029	635.029	635.029	635.029	635.029
Rec. Cuadratica Hash2	4.310	3.411	84.651	2.298	1.867	1.615
Rec. Cuadratica Hash3 (K=500)	4.047	3.418	318.376	2.294	1.864	1.572
Rec. Cuadratica Hash3 (K=128)	4.157	3.331	90.373	2.268	1.963	1.675

Cuadro 7: nfluencia de N en el proceso de búsqueda (nOperacionesExtraB)

Primero nos centraremos en el caso de **Encadenamiento**. En el caso de Hash1, se mantiene un número constante de operaciones extra, 129.176, independientemente del valor de N, lo que indica un comportamiento ineficiente. Además, en Hash2 disminuye el número de operaciones extra a medida que aumenta N, lo cual muestra un comportamiento predecible y más eficiente; aunque hay un valor anómalo en N=18.517 (14.511). Luego en Hash3 con K=500 y K=128, ambas se comportan de forma muy similar a la función Hash2, excepto con N=18.517, donde K=500 tiene un incremento drástico a 1.849.940.

Luego en cuanto a la **Recolocación Simple**, con Hash 1 no cambia con el valor de N, manteniéndose en 46.720.557 operaciones extra. Y luego con Hash2 y Hash3 para ambos valores de K, ambas muestran una disminución del número de operaciones extra a medida que aumenta N.

Por último, si nos fijamos en la **Recolocación Cuadrática**, Hash1 vuelave a mostrar ser ineficiente para todos los valores de N; y, al igual que en el caso anterior, el número de operaciones adicionales disminuye conforme aumenta N.

Por lo tanto, podemos concluir que para estos tamaños de tabla, la función **Hash1** siempre va a ser la más **ineficiente**, y que, por otra parte, Hash2 tiende a ser más predecible y eficiente, al igual que Hash3.

6 Conclusiones finales

Después de realizar experimentos, análisis, y comparaciones de todos los datos obtenidos, se debe determinar la **mejor opción** para el manejo de los datos proporcionados. Es esencial considerar, en primer lugar, el **tamaño de la tabla hash** y cómo el valor de N influye en las funciones de búsqueda e inserción de elementos en la tabla. Hemos observado que los valores de N con un **factor de carga cercano o inferior a 0,5** optimizan el procesado de datos. En una situación práctica, y asumiendo que se extendería el análisis para otros valores de N, sería prudente observar el comportamiento de N que garantice un factor de carga inferior a 0,5.

Además, hemos observado que N garantiza mejores resultados cuando es un **número primo**, por lo que en este caso lo más apropiado sería definir N como el primer primo que nos garantice un factor de carga menor o igual a 0.5 (para esta situación particular, sería $N = 19{,}997$ para Encadenamiento y $N = 40{,}009$ para Recolocación). Sin embargo al elegir estos valores de N tan altos, la función **Hash1 siempre será extremadamente ineficiente**. En adición a esto, sabemos gracias a los experimentos anteriores, que lo más óptimo es tomar el **correo electrónico como clave**, ya que aumenta la **dispersión** en la tabla (lo que disminuye el número de colisiones).

Por otro lado, también hemos notado que la mejor opción a la hora de insertar un dato es el **Encadenamiento**, ya que usa una lista enlazada para tal acción. No obstante, el Encadenamiento puede llegar a ser ineficiente en el caso de que haya que recorrer una lista enlazada muy larga para llegar al dato. Así, vamos a centrarnos primero en encontrar la **operación de búsqueda más eficiente**.

En el caso de realizar una búsqueda mediante Encadenamiento, tanto Hash2 como Hash3 son funciones válidas ya que proporcionan valores constantes y relativamente pequeños de pasos extra y colisiones. Por lo tento, podemos concluír que **la mejor estrategia de implementación para los datos proporcionados es la del Encadenamiento con Hash2 o Hash3**, sin importar mucho el valor de K.