TRANSFORMACIÓN DE CLAVES - HASHING – DISPERSIÓN

Introducción

Hasta el momento hemos organizado como lista o árbol a un conjunto de elementos caracterizados por un conjunto de claves, sobre el que existe una relación de orden. En esas oportunidades pusimos especial atención al `costo` de almacenamiento y recuperación de un elemento, y en todos los casos concluimos que la cantidad de comparaciones involucradas depende de la cantidad de elementos -N.

Ahora nos ocuparemos de organizar el conjunto, para que el almacenamiento o la recuperación de un elemento, representado por medio de una clave, se realice con el menor esfuerzo posible. Dado que los elementos se almacenan en direcciones de la memoria, el problema radica en determinar esas direcciones por medio de una transformación –H- de claves –K- en direcciones –D-.

$$H : K \rightarrow D$$

Consideramos en este documento, por razones de simplicidad, que el espacio direccionable es una estructura de arreglo, por lo que en esta técnica, conocida como ***transformación de claves, hashing* o *dispersión***, H transforma las claves en índices del arreglo.

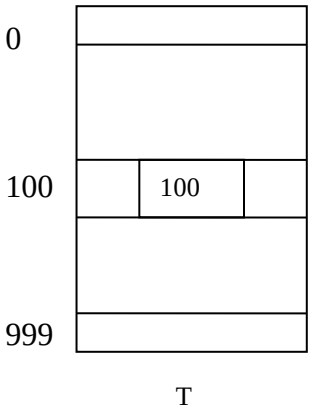
Si los elementos tienen un conjunto de claves correspondientes a un subconjunto de los naturales, con cotas inferior y superior y cardinalidad adecuada respecto a las dimensión del espacio direccionable disponible, y a la vez advertimos que no hay dos elementos distintos con el mismo valor de clave, entonces a estos elementos, que pueden almacenarse en una tabla T[0..|K|-1], puede asignárseles una `dirección` de T, que coincida con el valor de la clave. A estas tablas se las conoce como *tablas de acceso directo.*

Consideremos por ejemplo un conjunto K de claves correspondiente al Nº de legajo de 1000 empleados de una fábrica, que pertenecen al rango [000,999]. Dado que la tabla debe almacenar los datos de los 1000 empleados, debería contener al menos 1000 entradas.

Esto es $K = \{ k_i / 000 <= k_i <= 999\}$ y $D = \{ d_i / 000 <= d_i <= 999 \}$

¿Cuál es la función
H: K={ 000,...,999}→D={ 000,...,999} ?
Identidad : $K \rightarrow D$, esto es:
$h(k_i) = id(k_i) = d$

Si k = 100 entonces id(100) = 100 = d



Esta sería una situación ideal ya que:

- todas las posiciones de la tabla T están ocupadas
- para dos claves distintas k_1 y k_2, se cumple que $h(k_1)$ es distinta de $h(k_2)$.

Actividad

1. Explicite los siguientes conceptos:
    - Función
    - Inyectividad
    - Sobreyectividad
    - Biyectividad
2. Clasifique la función Identidad, teniendo en cuenta los conceptos del inciso 1.

Si la función de transformación H es biyectiva, entonces vamos a considerar que estamos trabajando con un ***Hashing Perfecto***, muy similar en eficiencia a las tablas de acceso directo.

El ejemplo presentado no refleja la mayoría de las situaciones reales ya que si bien la cardinalidad del conjunto de claves- |K|, puede ser N, el *conjunto de valores posibles* que estas pueden tener suele ser mucho mayor.

Si en la situación anterior nuestro campo clave fuese DNI, con valores posibles dentro del rango [5.000.000 , 20.000.000], sería absurdo dimensionar una tabla con 15.000.000 de componentes para almacenar los datos de 1000 empleados!

Actividad

Suponga que el campo clave es el Nombre del empleado, que puede tener una longitud máxima de 20 caracteres. ¿Cuál es la cardinalidad del conjunto de valores posibles de claves?

Técnica de Transformación de claves

Una alternativa efectiva a las tablas de acceso directo son las tablas dispersas. Una ***tabla dispersa***, o ***tabla hash,*** es una estructura de datos con disposición secuencial similar a una tabla de acceso directo, pues ambas aprovechan el acceso directo aportado por la disposición secuencial. Sin embargo, en lugar de utilizar un valor de clave para indexar directamente sobre la tabla dispersa, el índice es calculado a partir del valor de clave utilizando una ***función de dispersión***, que denotaremos ***h***[1].

El tamaño *M* de la tabla dispersa puede ser proporcional a la cantidad de elementos a almacenar –N, si elegimos una adecuada función de dispersión.

Es importante usar una función de dispersión H que distribuya uniformemente las claves en el espacio, pero aun así puede ocurrir que para dos claves distintas, $k_1 \neq k_2$, ocurre que $h(k_1) = h(k_2) = d$. Esta situación, en la que dos claves diferentes se dispersan en la misma `dirección` se conoce como ***colisión*** , asimismo k_1 y k_2 son claves ***sinónimas*** que han colisionado en la dirección *d* bajo la transformación *h*.

Por lo expuesto, distinguimos dos aspectos importantes a tener en cuenta cuando se utiliza esta técnica:

[1] Heileman, Gregory L. **Estructuras de Datos, Algoritmos y Programación Orientada a Objetos.** Madrid. McGraw-Hill. 306 páginas. Pág 142.

a) La elección de la función de transformación H.
b) La política de manejo de colisiones que se implementará.

a) Elección de la Función de Transformación

Si los valores posibles del conjunto de claves tienen la misma probabilidad de ocurrencia, debemos encontrar una función de transformación que, además de distribuir uniformemente las claves, sea calculable de modo eficiente, es decir, estar compuesta de un número reducido de operaciones aritméticas básicas.

Si por el contrario sabemos, por ejemplo, que las claves son progresiones aritméticas como las secuencias de identificadores X1, X2,....etc. podríamos desarrollar funciones de hash dependientes de esas distribuciones.

A continuación presentamos algunas funciones de transformación sencillas, en las que ***k*** representa una clave arbitraria, ***M*** representa el tamaño de la tabla dispersa y ***N*** representa el número de elementos a almacenar/recuperar :

- *Método de la División*

$$h(k) = k \bmod M$$

Este método produce resultados satisfactorios en muchos casos, especialmente con un divisor relativamente primo al tamaño M de la Tabla.

- *Extracción*

Esta función se basa en extraer de la clave, los dígitos que varían mas aleatoriamente. Estos pueden ser los últimos dígitos, los dígitos del medio o los del comienzo de la clave. La decisión respecto a que dígitos extraer se apoyará en un análisis previo de las características del conjunto de claves; y la cantidad de dígitos estará relacionada al rango de direcciones de la tabla.

- *Plegado*

El plegado consiste en subdividir la clave y posteriormente adicionar las partes resultantes:

Si $k_i = k_{i1}\, k_{i2} \mid \ldots \mid k_{in-1}\, k_{in}$ k_{ij} representa al dígito j-ésimo de k_i,

$$h(ki) = k_{i1}\, k_{i2} + \ldots + k_{in-1}\, k_{in}$$

La cantidad de dígitos que contendrá cada parte – dos en este caso-dependerá del rango de direcciones de la tabla.

- *Cuadrado Medio*

Este método consiste en elevar la clave al cuadrado y posteriormente extraer de ella los dígitos centrales:

$k_i = k_{i1}\ k_{i2}\ \ldots\ k_{in-1}\ k_{in}$

$h(k_i) = k_i^{\,2}$ y luego se extraen los dígitos centrales

$d_i = k_{ij}\ k_{ij+1} \ldots k_{ij+l}$ $\quad 1 < j < j+l < n$ $\quad$ y $\quad 0 <= k_{ij}\ k_{ij+1} \ldots k_{ij+l} <= M-1$

La cantidad de dígitos que se extraen depende del rango de direcciones de la tabla.

- *Funciones aplicables a claves alfanuméricas*

Si bien las funciones de transformación propuestas hasta el momento consideran que las claves son básicamente de naturaleza numérica, dichas claves pueden ser alfanuméricas.

La manera mas sencilla de transformar las claves alfanuméricas a números, es convertir cada caracter a su valor dentro de la tabla ASCII, y posteriormente concatenar dichos valores.

Sin embargo, al trabajar con cadenas compuestas por una cantidad importante de caracteres, puede resultar mas conveniente adicionar los valores que los representan, en lugar de considerar la dirección generada por su concatenación. Esto es:

$k_i = c_{i1}\ c_{i2}\ c_{i3} \ldots .\ c_{in}$

$h(k_i) = ASCII(c_{i1}) + ASCII(c_{i2}) + \ldots\ldots + ASCII(c_{in})$

Ahora bien, ¿Qué ocurre si existen cadenas con los mismos caracteres, ocupando posiciones diferentes? ROMA-AMOR-OMAR-RAMO; SAPO-POSA-PASO-OPAS-SOPA; ARBOL-LABOR; CARLA-CLARA-CALAR; etc.

Con la función de transformación propuesta, cadenas con las características mencionadas computan en la misma dirección. Una solución a este problema es considerar para cada carácter además de su valor numérico la posición que este ocupa dentro de la cadena , esto al multiplicarlo por la base del sistema de codificación elevada a esa posición.

$k_i = c_{i1}\ c_{i2}\ c_{i3} \ldots .\ c_{in}$

$h(k_i) = ASCII(c_{i1}) * b^1 + ASCII(c_{i2}) * b^2 + \ldots\ldots + ASCII(c_{in}) * b^n$

En todas las funciones propuestas debemos tener en cuenta si el resultado de la transformación, esto es la dirección computada, pertenece o no al rango de direcciones de la tabla.

Actividad

1- Elija un conjunto pequeño de claves numéricas y calcule las direcciones resultantes de aplicar las distintas funciones de transformación.
2- Para algunos de los valores de claves alfanuméricas presentados, calcule las direcciones resultantes de aplicar las dos funciones de transformación propuestas.
3- ¿Siempre la dirección resultante de aplicar una función de transformación cae dentro del rango de direcciones de la tabla? Si su respuesta es negativa, proponga una solución.

b) Políticas de Manejo de Colisiones

Si bien debemos elegir una función de transformación que minimice las colisiones, es importante tener en cuenta que las colisiones pueden ocurrir. En este sentido, Standish[2] hace referencia a la paradoja de cumpleaños de von Mises que expresa lo siguiente: si en una habitación se encuentran 23 o mas personas, entonces hay chance de que dos o más de las personas cumplan años el mismo día, si en cambio en la habitación hay 88 personas o más, entonces puede que tres o mas de las personas presentes tengan la misma fecha de cumpleaños.

Actividad

¿Cuál es el rango de direcciones de la tabla de dispersión, si la función de transformación retorna el día del año en que cumple años una persona?

A partir de lo expresado por Standish descubrimos que, salvo en el caso de estar trabajando con un hashing perfecto, siempre debemos considerar que hacer cuando dos claves computan en la misma dirección, ya que la eficiencia de la técnica de hashing también depende de la estrategia que se elija para tratar las claves sinónimas.

Presentamos a continuación tres de las políticas de resolución de colisiones propuestas por Standish[3]:

- Encadenamiento
- Uso de Buckets o cubos
- Direccionamiento abierto.

Asimismo, esta políticas pueden reconocerse bajo las denominaciones de Dispersión Abierta y Dispersión Cerrada[4]. La primera comprende a Encadenamiento, y la segunda a Uso de Cubos y a Direccionamiento Abierto.

- *Encadenamiento o Dispersión Abierta*

Esta estrategia se basa en mantener en listas los elementos que colisionan en la misma dirección.

Operativamente:

La tabla T consiste de M cabezas de listas, cada una de las cuales contiene claves sinónimas, es decir, claves que han colisionado en la misma dirección al aplicárseles la función de transformación H.

A una clave k se le computa una dirección d, siendo d = h (k). En la ´dirección´ d de la tabla T se encuentra una cabeza de lista que contiene los registros cuyas claves han computado su dirección en d. Si la operación es una búsqueda o supresión de la información asociada con la clave k, la lista será recorrida hasta encontrar el registro correspondiente a k. Para la operación inserción, la lista será recorrida hasta establecer que k no está, es decir, esta operación lleva implícita una búsqueda.

---

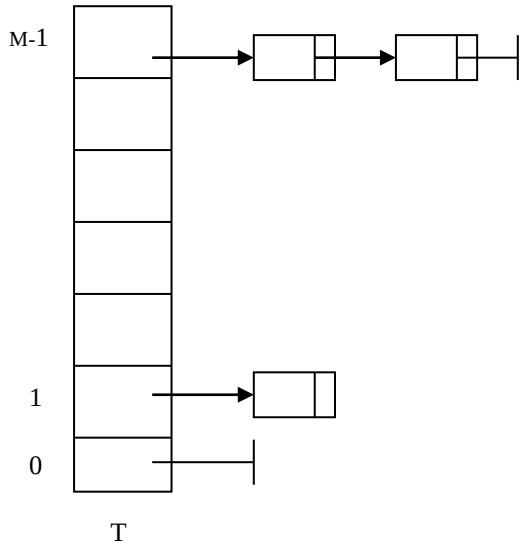
[2] Standish, Thomas A. **Data Structure Techniques.** California. Addison-Wesley Publishing Company. 1980. 448 páginas. Pág 142.

[3] Standish, Thomas A. Op. Cit. Pág 143.

[4] Heileman, Gregory L. Op. Cit. Pág 146.

Si se almacenan N claves en una tabla T de M entradas, entonces puede esperarse que el número promedio de claves en cada lista sea N / M. Esto implica que el trabajo involucrado en una búsqueda dentro de una lista dada, se decrementa en un factor de M con respecto a la cantidad total de elementos - N.

Representamos a la tabla T de la siguiente manera :



Al momento de trabajar con esta política, debemos tener en cuenta los distintos TAD vistos, con el objeto de seleccionar el mas adecuado para la técnica que estamos tratando. Una alternativa es mantener cada componente de la tabla como una lista ordenada por contenido, o un árbol, disminuyendo así la cantidad de comparaciones requeridas para encontrar la información asociada con ksi esta se encuentra en la lista, o para reportar que k no ha sido ingresada.

Si centramos nuestro interés en mejorar la performance en la etapa de ingreso de la información, una posibilidad es insertar los nuevos elementos en un extremo, por ejemplo al frente de la lista, manteniendo constante el tiempo requerido para esta operación.

- *Uso de Buckets*

En esta estrategia, cada una de las M entradas de la tabla, llamada bucket o cubo, contiene un grupo de *b* claves o registros. La función H aplicada a una clave k computa una dirección d = h(k) que representa un número de bucket. En inserción, si hay lugar en el bucket computado, la nueva clave es ingresada. Si el bucket está lleno, debe invocarse una política de manejo de desborde u overflow.

Es interesante tener en cuenta que si las claves dentro de un bucket están ordenadas, puede aplicarse una búsqueda binaria.

Como esta estrategia considera un espacio limitado - cada bucket puede contener a lo sumo b registros- la incluimos en la clasificación Dispersión Cerrada en contraste a la Dispersión Abierta que considera un espacio potencialmente ilimitado.

Al soportar cada bucket como máximo b registros, y aun habiendo elegido una función de transformación H adecuada, es imprescindible considerar un espacio adicional, o Área de Overflow, suficiente para soportar las claves que puedan desbordar de los buckets de Área Primaria.

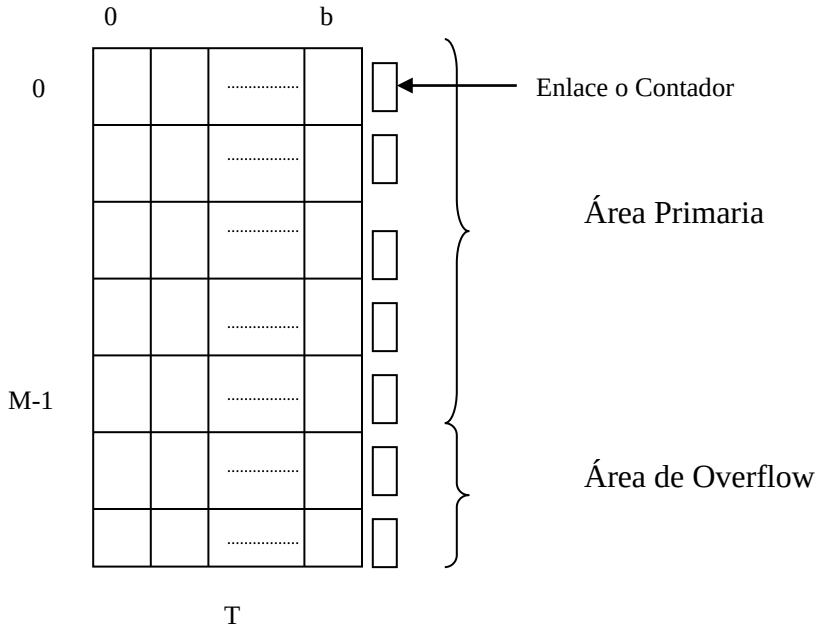
El espacio total constará entonces de un área primaria - de M buckets con b registros cada uno- y de un área de overflow. La dimensión de esta última puede ser, según pruebas empíricas, el 20% del espacio calculado para el área primaria.

El área de overflow puede estructurarse como buckets, pudiendo cada uno de estos contener desbordes producidos en distintos buckets del área primaria.

Al insertar una nueva clave, si ocurre que el bucket de área primaria computado por medio de h desborda, la clave debe entonces almacenarse en un bucket del área de overflow. Para registrar esta situación, al bucket de área primaria puede asociarse un enlace con el bucket del área de overflow. Otra alternativa es asignarle a cada bucket de área primaria un contador de claves sinónimas.

Aunque este método de manejo de overflow parece ineficiente, la cantidad de desbordes es estadísticamente pequeño – estamos operando bajo la hipótesis de que la función de transformación seleccionada minimiza las posibles colisiones.

El siguiente diagrama representa al espacio de almacenamiento para esta política de manejo de colisiones



- *Direccionamiento Abierto*

En esta estrategia, cada componente de la tabla T soporta una sola clave. Cuando a una clave k se le computa una dirección, h(k)= d pueden presentarse dos situaciones en el proceso de ingreso de claves:

. Que la entrada d esté vacía, en cuyo caso la inserción de la información es inmediata.

. Que la entrada d esté ocupada, en cuyo caso es necesario localizar una entrada vacía para almacenar la clave k.

En esta situación, debe establecerse algún criterio para localizar esa entrada vacía a la que hacemos referencia, dado que a partir del conjunto D de direcciones existen distintas alternativas a seguir en su inspección, que formalmente corresponden a permutaciones de las direcciones del conjunto D. B_0 , B_1,...., B_{m-1} es una de esas permutaciones, que es llamada *secuencia de prueba*, pues establece un criterio para localizar una posición vacía en la cual ingresar un elemento.

En el proceso de búsqueda de una clave, si ésta no es localizada en la posición d, deben inspeccionarse las direcciones según la secuencia de prueba seleccionada hasta encontrar una direccion que la contiene -búsqueda exitosa-, o descubrir una dirección vacía -búsqueda infructuosa. Las secuencias de prueba presentadas a continuación son propuestas por Standish[5].

- Secuencia de Prueba Lineal:

En este caso se genera una secuencia de prueba a partir de

$$(h(k) - i * p(k)) \bmod M \qquad 0 <= i <= M-1$$

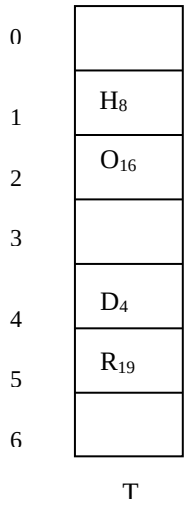
es decir, las direcciones obtenidas son una permutación de las direcciones 0,1....,M-1.

Si p(k)=1 , la secuencia de prueba para cualquier clave k es:

h(k), h(k)-1,.....,1, 0, M-1, M-2,....., h(k)+1

Operativamente primero se inspecciona la dirección h(k), y a partir de ella se inspeccionan circularmente todas las direcciones de la tabla atravesando primero las mas bajas. Esta secuencia recibe el nombre de *secuencia de prueba lineal*.

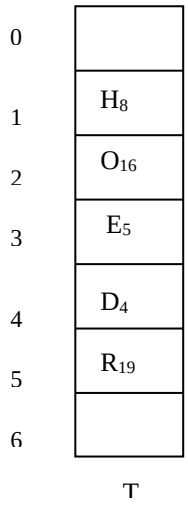
La secuencia de prueba lineal puede producir un fenómeno llamado *agrupamiento primario*. Retomamos el ejemplo propuesto por Standish[6] en el que las claves son de la forma L_n, donde *n* es un número que indica la posición de la letra *L* en el alfabeto (A_1, B_2, C_3, D_4, etc.). Si el tamaño M de la tabla es 7 y la función de transformación es el método de la división – H(k) = n mod 7-, luego de insertar las claves: O_{16}, H_8,R_{19},D_4, el contenido de la tabla T es el siguiente



[5] Standish, Thomas A. Op. Cit. Pág 145-148.
[6] Standish, Thomas A. Op. Cit. Pág.146.

Dos grupos de claves se han formado, uno que contiene a las claves H_8 y O_6, y el otro que contiene a las claves D_4 y R_{19}.

Estos agrupamientos llevan a que todas las claves que colisionan en una dirección B, extiendan los grupos (clusters) que contienen a B. Incluso clusters adyacentes tienden a unirse para formar clusters compuestos. La inserción de la clave E_5 en esta situación, provoca que se extienda el grupo al que pertenece la componente 5. Si bien $h(E_5)=5$, en esta componente ya ha sido almacenada previamente R_{19} , por lo que siguiendo la secuencia de prueba lineal E_5 es finalmente almacenada en la componente 3.



El principal inconveniente de estos agrupamientos, es que a medida que el tamaño de los grupos aumenta, las direcciones que se computan en ellos tienden a no tener las mismas probabilidades de ser ocupados. Según ello, la posición 0 de la tabla tiene 6 de 7 posibilidades de ser ocupada (por claves que computen en las direcciones 0 a 5), en cambio la posición 6 tiene 1 de 7.

De esta forma, los grandes grupos tienden a crecer mas rápido que los pequeños, lo que provoca una situación inestable y a la vez incrementa drásticamente los tiempos de inserción y recuperación de claves en la medida que la tabla se acerca a la saturación total.

- *Secuencia de prueba pseudo random*

Para evitar los inconvenientes generados por los agrupamientos primarios, se propone una secuencia de la forma

$h(k)+0, h(k)+r_1, ..., h(k)+r_{m-1}$

reducida a módulo tamaño de la tabla.

Los números $r_1, ...,r_{m-1}$ son una permutación de las direcciones de la tabla que puede ser generada por un generador de números aleatorios. Un principal problema que esto presenta es que los números deberían ser almacenados, para ser utilizados tanto en la inserción como en la posterior búsqueda de una clave.

Una alternativa es elegir $p(k)=c$ para todo k, siendo c un numero relativamente primo al tamaño de la tabla.

- *Búsqueda de residuo cuadrático*

Esta secuencia de prueba es de la forma:

$h(k), h(k)+i^2, h(k) - i^2 \quad 1 <= i <= M / 2$

todas reducidas a módulo tamaño de la tabla.

Las secuencias de prueba Pseudo Random y Residuo Cuadrático presentan como principal inconveniente la generación de *agrupamientos secundarios*. Esto significa que cuando dos claves k_1 y k_2 colisionan, seguirán la misma secuencia de prueba.

- *Doble Hashing*

El doble hashing surge en un intento de superar los agrupamientos primarios y secundarios producidos por las secuencias de prueba anteriores.

Esta secuencia de prueba es de la forma

$h(k) - i * h_2(k) \quad$ para $0 <= i <= M-1$

reducidas a módulo tamaño de la tabla. H_2 es una segunda función de transformación.

Con este método, claves diferentes que colisionan según la función H, no seguirán la misma secuencia de prueba. A esta conclusión han arribado distintos autores realizando pruebas empíricas.

Factor de Carga

Standish define al *factor de carga* α de una tabla como la razón entre el número de entradas N ocupadas en la tabla y el número total de entradas M en la tabla, esto es α**=N/M**. Este autor expresa que si $\alpha <= 0.7$, la cantidad de comparaciones involucradas en la recuperación de un elemento es adecuada, mientras que la performance se deteriora, especialmente al aplicar la secuencia de prueba lineal, a medida que la tabla se aproxima a la saturación total – cuando N se aproxima a M.

Actividad

Para un conjunto de 1000 claves, método de la división como función de transformación, y direccionamiento abierto como política de manejo de colisiones, defina el tamaño M de la tabla T teniendo en cuenta la noción de factor de carga.

Construcción del *TAD Tabla Hash*

Actividad

1. Enumere las operaciones que debe tener el TAD Tabla Hash.
2. Tomando como base el listado propuesto en el inciso 1, realice un análisis comparativo entre los TAD Tabla Hash y TAD ABB.
3. ¿Qué aspecto, de los analizados en el documento, debe ser tenido en cuenta al trabajar en la etapa de representación del objeto de datos correspondiente al TAD Tabla Hash?