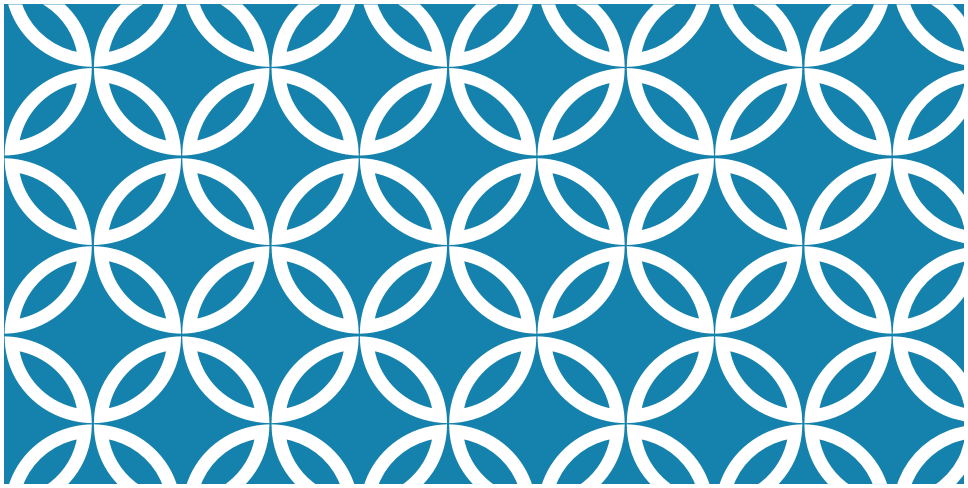


TABLAS HASH

+Mapeo
+Manejo de colisiones

INTRODUCCIÓN

- Muchas aplicaciones requieren un conjunto dinámico que soporte las operaciones de un diccionario:
  - Insertar
  - Buscar
  - Cambiar
  - Eliminar
- Por ejemplo el compilador cuando guarda los identificadores de un programa.

INTRODUCCIÓN

- Es posible hacer uso de una lista enlazada con un tiempo lineal O(n) ; sin embargo, este tiempo se puede reducir notablemente a orden O(1) en la mayoría de los casos usando una tabla hash.
  - La idea surge del acceso a la memoria de una computadora, el cual es un caso especial de una técnica denominada direccionamiento directo o polinomio de direccionamiento, en la que una clave conduce directamente al elemento de datos.
  - Otro ejemplo es el almacenamiento de datos en arreglos, en este caso el índice del arreglo juega el papel de la clave.

INTRODUCCIÓN

- El problema de los esquemas de direccionamiento directo reside en que requieren un almacenamiento equivalente al rango de todas las posibles claves y no es proporcional al número de elementos que realmente se almacenan.
- Consideremos el ejemplo de los números del IFE. Un esquema de direccionamiento directo para almacenar la información de todos los ciudadanos Mexicanos requeriría de una tabla de mas de 100,000,000 de entradas, puesto que el número del IFE es de 9 dígitos.
- No importa si esperamos almacenar datos de 100 o de 100,000 ciudadanos, el esquema de direccionamiento directo necesitará una tabla que pueda incluir millones de entradas potenciales!.

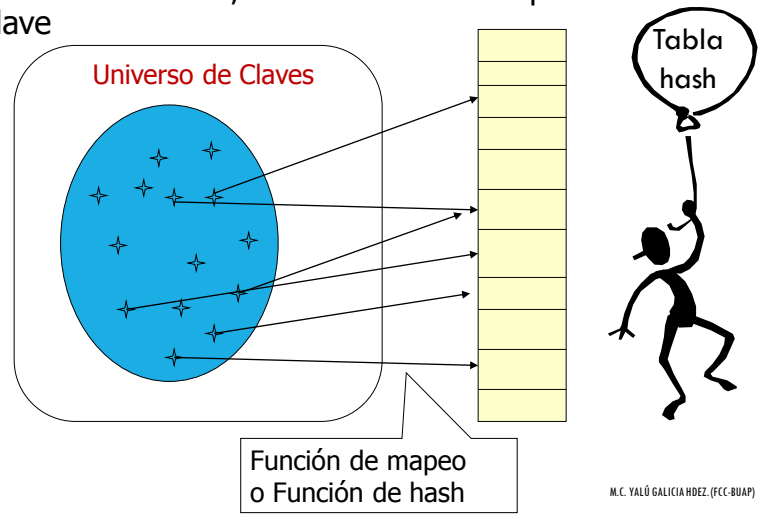
TABLA HASH

- El hashing es una técnica que ofrece una velocidad comparable a la del direccionamiento directo (O(1)) con requisitos de memoria (O(n)) mucho más gestionables, donde n es el número de entradas almacenadas en la tabla, conocida como tabla hash.
- La técnica hashing utiliza una función de mapeo, llamada función hash, para generar un código hash pseudoaleatorio a partir de la clave del objeto y luego utiliza este código (~dirección directa) para indexar en la tabla hash.

M.C. YALÚ GALICIA HDEZ. (FCC-BUAP) 5

HASHING

- Una función hash, obtiene un índice a partir de una clave



M.C. YALÚ GALICIA HDEZ. (FCC-BUAP) 6

EJEMPLO

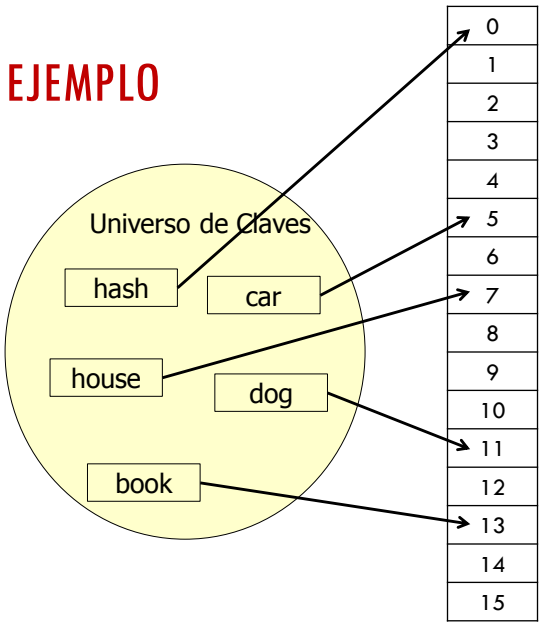
- Supongamos que queremos una pequeña tabla hash con capacidad de 16 entradas para almacenar palabras en inglés.
  - Necesitaremos una función hash que asigne las palabras a los enteros 0, 1, ..., 15.
- Normalmente, la tarea se divide en crear una función hash en dos partes:
  - **Paso 1**: Asignar la clave a un entero.
  - **Paso 2**: Asignar el entero "aleatoriamente" o en un rango bien distribuido de enteros ({ 0, ..., m-1 }, donde m es la capacidad o el número de entradas que se utilizarán para indexar en la tabla hash.

EJEMPLO DE FUNCIÓN DE HASH

- Como ejemplo, consideremos una función de hash o código hash que toma el valor numérico del primer carácter de la palabra y lo suma al valor numérico del último carácter (paso 1)
- Después, toma este resultado y hace

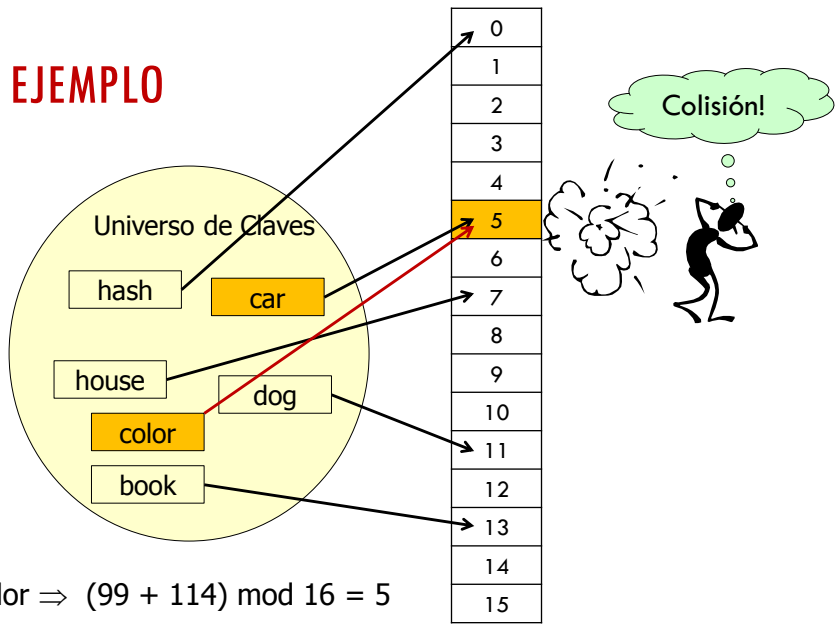
  res mod16 (paso 2).
- Por ejemplo, el valor numérico de una "c" es 99 y el de una "r" es 114. Por lo que, "car" se dispersaría en (99 + 114) mod 16 = 5.

EJEMPLO



M.C. YALÚ GALICIA HDEZ. (FCC-BUAP) 9

EJEMPLO



Color ⇒ (99 + 114) mod 16 = 5

M.C. YALÚ GALICIA HDEZ. (FCC-BUAP) 10

COLISIONES

- Así "car" y "color" se dispersan al mismo valor utilizando esta función hash, ya que tienen la misma letra inicial y final, lo que indica que nuestra función hash podría no ser tan "aleatoria" como debiera.
- Pero si n > m, los códigos hash duplicados, también conocidos como colisiones, son inevitables.
  - Donde n es número de claves y m tamaño de la tabla
- De hecho, incluso con n < m, las colisiones pueden seguir siendo consecuencia del argumento von Mises (también conocido como la paradoja del cumpleaños: si hay 23 personas en una habitación, la probabilidad de que al menos dos de ellas cumplan los años el mismo día supera el 50%).

TAREA DE HASHING

- Diseñar una función hash adecuada para asignar códigos hash a las claves, de tal forma que un conjunto no aleatorio de claves genere un conjunto equilibrado "aleatorio" de código hash
- Si los códigos hash no son aleatorios, el número excesivo de colisiones "agolpará" las claves en la misma dirección directa.
- Tratar las posibles colisiones que aparezcan después de aplicar hashing.

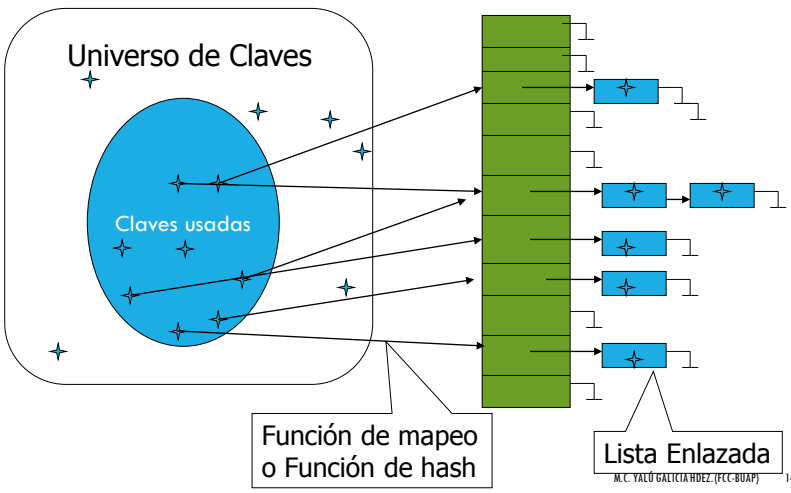
MANEJO DE COLISIONES

- El encadenamiento o hashing abierto es un enfoque sencillo y eficaz para gestionar las colisiones.
- En una tabla hash con encadenamiento, las entradas de la tabla (normalmente llamadas slots o buckets, no contienen los propios objetos almacenados, sino listas enlazadas de los objetos.
  - Los objetos con claves colisionadas se insertan el la misma lista.
- La inserción, la búsqueda y la eliminación se convierten en procesos de 2 pasos:
  - Paso 1: Utilizar la función hash para seleccionar el slot adecuado.
  - Paso 2. Realizar la operación necesaria en la lista enlazada a la que se hace referencia en dicho slot.

M.C. YALÚ GALICIA HDEZ. (FCC-BUAP) 13

HASHING ABIERTO O ENCADENAMIENTO

- Desde un "gran" Universo sólo un número reducido de claves serán consideradas.



M.C. YALÚ GALICIA HDEZ. (FCC-BUAP) 14

FACTOR DE CARGA Y RENDIMIENTO

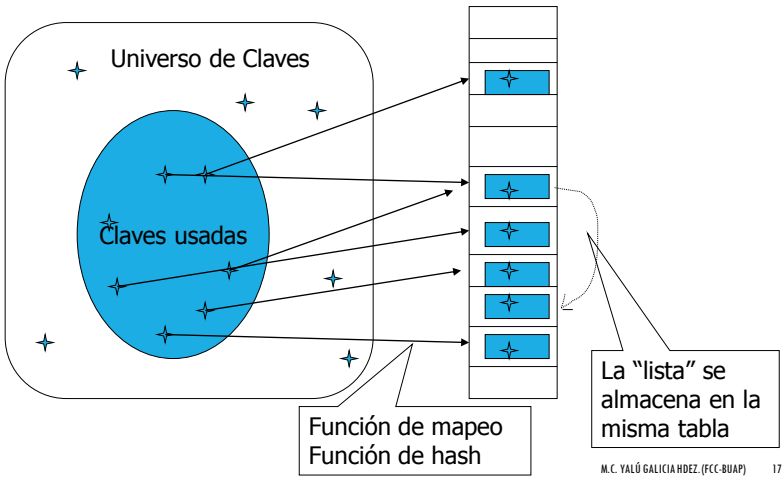
- La relación entre el número de elementos almacenados, n, y el número de slots de la tabla, m, n/m, recibe el nombre de factor de carga.
  - Como las listas enlazadas a las que se hace referencia en los slots hash pueden contener un número arbitrario de elementos, no existe límite en la capacidad de la tabla hash que utiliza el encadenamiento.
  - Si la función hash empleada no distribuye bien las claves, el rendimiento de la tabla disminuirá.
  - El peor caso para una tabla hash y para un árbol binario es el de la lista enlazada. Esto ocurre cuando todas las claves se dispersan al mismo slot.
  - Sin embargo, dada una buena función hash, se puede demostrar que una tabla hash con encadenamiento y un factor de carga L puede realizar las operaciones básicas de inserción, búsqueda y eliminación en un tiempo O(1 + L).
  - Para una mayor eficacia, el factor de carga debe ser ≤ 0.75

MANEJO DE COLISIONES

- Otro método para gestionar colisiones es el hashing cerrado.
  - Con este método todos los elementos o claves son almacenadas en la tabla hashing misma. Es decir, cada entrada de la tabla contiene un elemento del conjunto o NULL.
  - Cuando se busca, examinamos varias entradas hasta encontrar lo buscado o es claro que no está.
  - No hay una lista ni elementos almacenados fuera de la "tabla".
  - La tabla se podría llenar. El factor de carga no puede exceder 1.
  - La gran ventaja de hashing cerrado es que elimina totalmente las referencias usadas en la lista enlazada. Se libera así espacio de memoria, el que puede ser usado en más entradas de la tabla y menor número de colisiones.

VISIÓN GRÁFICA (HASHING CERRADO)

- Desde un "gran" Universo sólo un número reducido de claves serán consideradas.



M.C. YALÚ GALICIA HDEZ. (FCC-BUAP) 17

HASHING CERRADO

- La inserción se efectúa probando la tabla hasta encontrar un espacio vacío. La función de hash usa un segundo argumento que es el número de la prueba.
h: U x { 0, 1, 2, .. , m-1} ----> {0, 1, 2, ... m-1}
Para una clave k se prueban sucesivamente: h(k,0) , h(k,1), .. h(k,m-1)

```
Hash_Insert(T, k) {
 /* pseudo código */
   int i,j;
   for (i = 0; i<m; i++) {
            j=h(k,i);
            if (T[j] == NULL){
                 T[j]=k;
                 return;
            }
   }
   printf(" hash overflow");
}
```

```
Int Hash_Search(T, k) {
/* Pseudo código*/
  int i,j;
  for (i = 0; i<m; i++) {
            j=h(k,i);
            if (T[j] == NULL)
return -1;
            else if  (T[j] == k)
                        return j;
   }
}
```

18

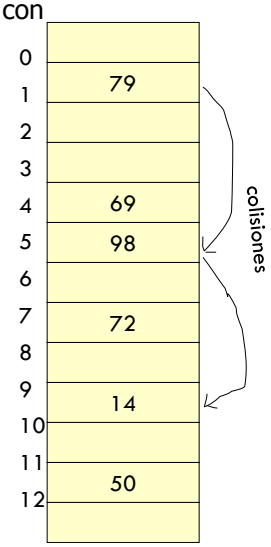
FUNCIONES DE HASH H(K,I)

- Existen al menos dos formas para definir esta función: prueba lineal y doble hashing.
- Prueba lineal
  - La función es:
    h(k,i) = (h'(k) +i) mod m
  - Una desventaja de este método es la tendencia a crear largas secuencias de entradas ocupadas, incrementando el tiempo de inserción y búsqueda.
- Doble hashing
  - La función es:
    h(k,i) = (h1(k) + i*h2(k)) mod m
  - Por ejemplo:
    h1 = k mod m
    h2 = 1 + (k mod (m-1))

EJEMPLO DE HASHING CERRADO

- Sea h(k,i) = $(h_1(k) + i*h_2(k)) \bmod 13$; con
  h_1 = k mod 13
  h_2 = 1 + (k mod 11)

- h(79,0) = 1
  h(72,0) = 7
  h(98,0) = 7
  h(98,1) = (7+11) mod 13 =5
  h(14,0) = 1
  h(14,1) = (1+4) mod 13 = 5
  h(14,2) = (1+2*4) mod 13 = 9



HASHCODE

- En un lenguaje orientado a objetos como es Java, la primera fase de hashing, la función hash1, es responsabilidad de la **clase de la clave**, no de la clase de la tabla hash.
- La tabla hash almacenará las entradas como Object, por lo que no tiene información suficiente para generar un código hash desde Object, ya que podría ser un String, un Integer o un objeto personalizado.

HASHCODE

- Java obtiene esto mediante el método **hashCode()** de Object.
  - Como todas las clases de Java extienden implícita o explícitamente de Object, entonces Object posee un método hashCode() que devuelve un int y que debe ser **sobreescrito** en la clase de la clave
- Cuidado: el método hashCode() puede devolver un **entero negativo** en Java; si queremos un entero no negativo (y solemos quererlo) deberemos tomar el **valor absoluto** de hashCode().

¿ QUE PROPIEDADES DEBE TENER UN CÓDIGO HASH?

- Como utilizamos códigos hash para indexar en la tabla hash y encontrar objetos, el código hash del objeto debe permanecer fijo a lo largo del programa.
  - Un número realmente aleatorio no sería válido para un código hash aceptable, ya que generaría un valor distinto cada vez que dispersásemos.
- Si dos objetos son equivalentes (pero no necesariamente idénticos), como dos copias de la misma cadena, los códigos hash de los dos objetos también serían iguales, ya que deben recuperar el mismo objeto de la tabla hash.
  - Formalmente, si o1 y o2 son Object y o1.equals(o2), entonces o1.hashCode() == o2.hashCode().

M.C. YALÚ GALICIA HDEZ. (FCC-BUAP) 23

DISEÑO DE CÓDIGO HASH

- El hashing tiene mucho más de arte que de ciencia, especialmente en el diseño de funciones hash1.
  - La última comprobación de un buen código hash es ver si distribuye sus claves de un modo "aleatorio" correcto.
- Se debe seguir algunos principios:
  - Un código hash debe depender tanto como sea posible de la clave.
  - Un código hash debería asumir que sufrirá manipulaciones posteriores para adaptarse al tamaño concreto de la tabla: la fase hash2 .
  - Ver ejemplo TablaHash (hashing.zip)

M.C. YALÚ GALICIA HDEZ. (FCC-BUAP) 24

ESO ES TODO!