Tablas de Dispersión
Funciones Hash

Estructura de Datos

Introducción

- Las tablas de dispersión son estructuras de datos que se usan en aplicaciones que manejan una secuencia de elementos.
- Cada elemento se asocia a una clave, número entero positivo perteneciente a un rango de valores relativamente pequeño.
- Cada uno de estos elementos debería tener una clave que lo identifique unívocamente.
- Por ejemplo el número de matrícula del alumno.

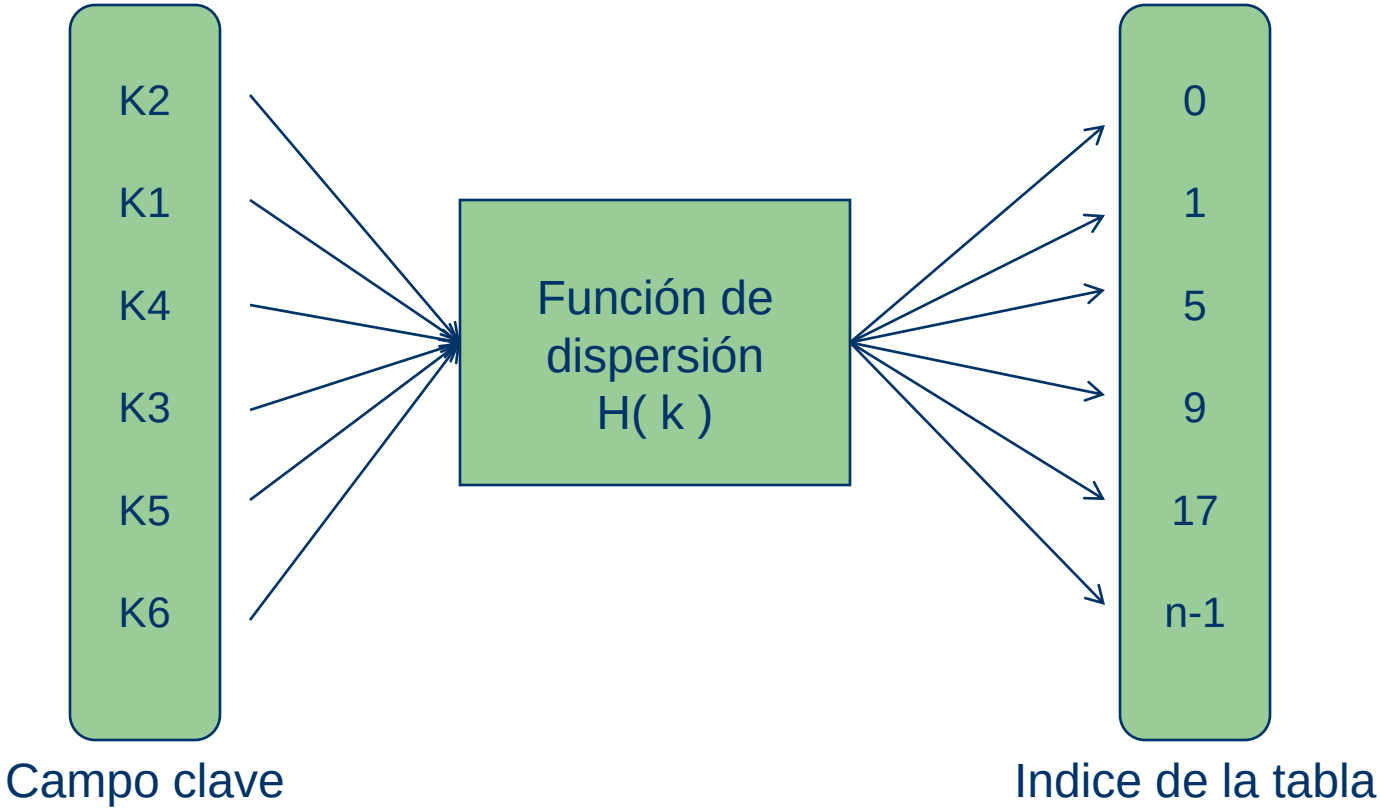
Introducción

- El hashing es una técnica usada para insertar, borrar y encontrar datos en tiempo promedio constante O(1). La estructura central es la tabla de hash.
- Una tabla de hash ideal pudiera pensarse como un arreglo de tamaño fijo conteniendo llaves (datos).
- Típicamente la llave es una cadena con algún valor asociado.
- Cada llave es mapeada a un número entero en el rango 0 a n-1, donde n es el tamaño de la tabla.

Introducción

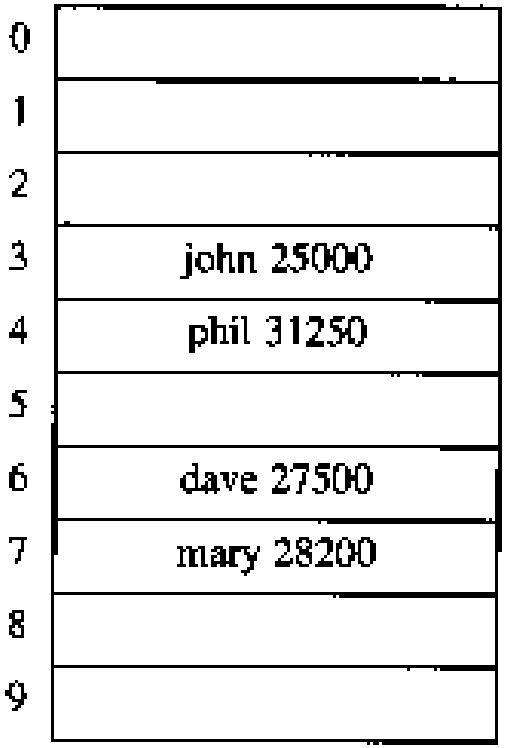
- Ese número indicará la celda donde la información debe residir.
- El mapeo es llamado función de hash, que idealmente debe ser simple de calcular y debe asegurar que dos llaves distintas correspondan también a dos celdas distintas.
- Esto último no es posible. Por principio de cuentas hay un número finito de celdas y podría haber un número infinito de llaves.

Función de Dispersión



Ejemplo

John es mapeado a 3
Phil a 4
Dave a 6 y
Mary a 7



Función de hash

- Si las llaves son enteros, una estrategia razonable es regresar *Llave mod n*.
- Sin embargo, no siempre es posible emplear tal estrategia (imagine que el tamaño de la tabla es 10 y todas las llaves terminan en 0).
- Por otro lado, en ocasiones las llaves serán cadenas alfanuméricas, así que la función de hash debe ser más elaborada.

Ejemplos de funciones de hash

- Una opción sería sumar los valores ASCII de los caracteres en la cadena.
- Por ejemplo, para una tabla de 120 elementos, el apellido JONES nos llevaría a: 74+ 79+ 78+ 69 + 83 = 383. Entonces su posición en la tabla sería 383 mod 120 = 23.
- Desafortunadamente, STEEN (83+84+69+69 +78 = 383) también nos llevará a la misma posición.
- Y ahora, ¿Qué haremos?

Ejemplos de funciones de hash

- Una opción sería buscar otra función de hash.
- ¿Qué tal si usamos las fechas de nacimiento?
- No es mala idea, pero también existe la probabilidad de que en un grupo de *n* personas, dos hayan nacido el mismo día.
- Así pues, en grupos grandes (en uno de 40 personas ya tenemos 10 a 1 a favor de tener duplicados), no es una buena solución.

- Entonces debemos resignarnos a vivir con funciones que producen ese comportamiento. De hecho, es tan común que se dice que cuando dos llaves son asignadas a la misma celda, ha ocurrido una *colisión.*

Técnica de Plegamiento

El registro de pasajeros de un tren se identifica con un campo de 6 dígitos, y se va a crear una tabla de dispersión de 1000 registros.

Entonces para las claves:

245643 245981 257135

Se genera:

h(245643) = 245 + 643 = 888

h(245981) = 245 + 981 = 1226 se toma 226

h(257135) = 257 + 135 = 392

Una variante para generar más dispersión puede ser:

h(245643) = 245 + 346 = 591

h(245981) = 245 + 189 = 434

h(257135) = 257 + 531 = 788

Técnica de mitad del cuadrado

Otra técnica es elevar el número al cuadrado, y seleccionar por ejemplo 3 dígitos, siempre los mismos para todos los casos.

Entonces para las claves:

245643 245981 257135

Se genera:

$h(245643) = 245643^2 = 60340483449 = 483$

$h(245981) = 245981^2 = 60506652361 = 652$

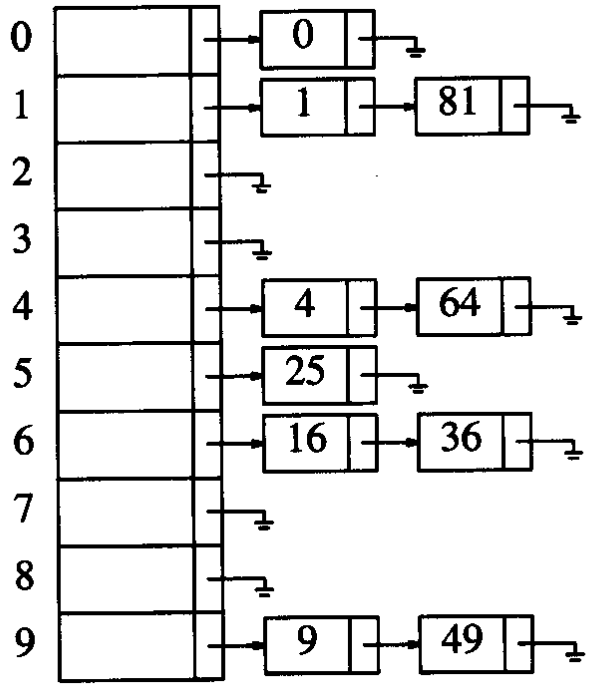
$h(257135) = 257135^2 = 66118408225 = 408$

Manejo de colisiones

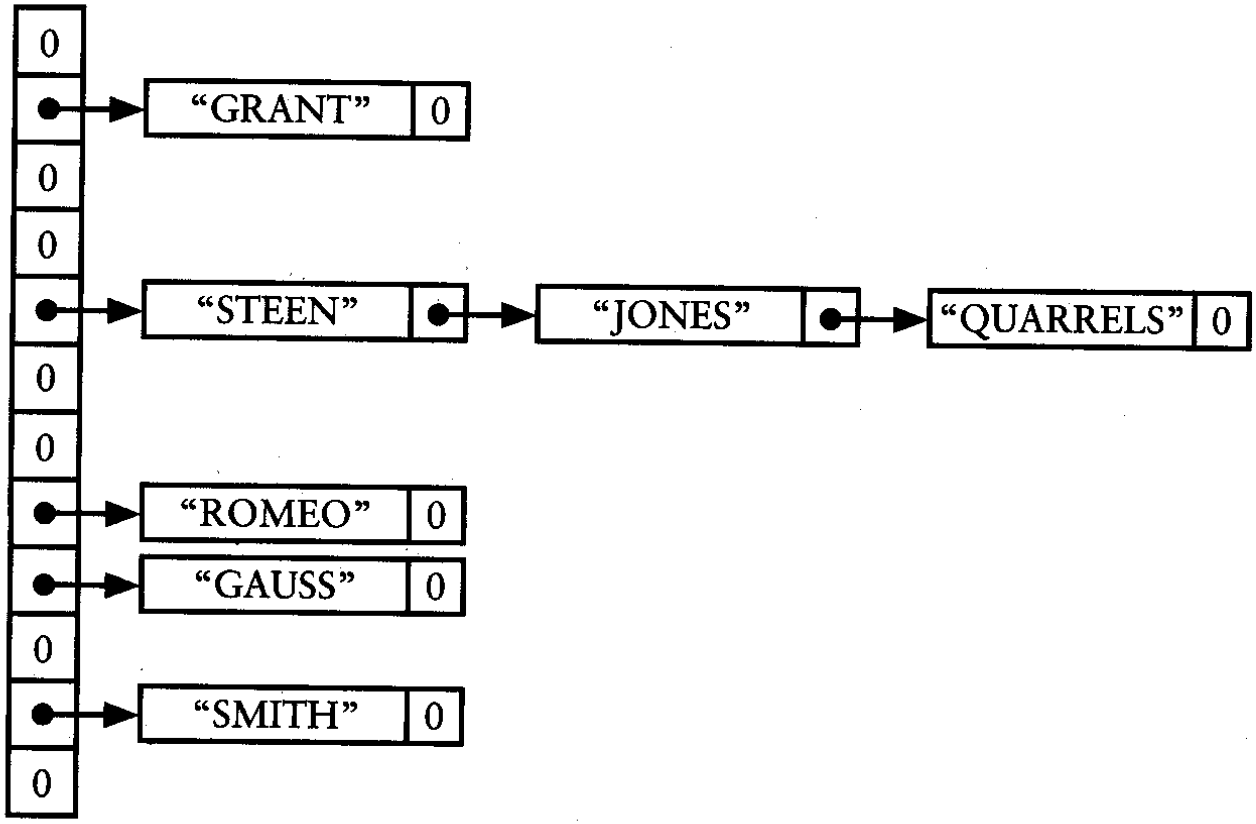
- La función de dispersión, puede generar siempre dos valores iguales para claves diferentes, esto produce una colisión, que hay que resolver.
- Existen varias estrategias para el manejo de colisiones, que podemos agrupar en dos clases:
- Open Hashing (direccionamiento abierto)
- Closed Hashing (direccionamiento cerrado)
  - Probado lineal.
  - Probado cuadrático.

Open Hashing

- Se trata de mantener una lista con todos los elementos que sean direccionados a la misma celda.



Otro ejemplo



Closed Hashing

- En este caso, si existe una colisión, se intentan celdas alternas, hasta que se encuentre una vacía.
- Más formalmente,

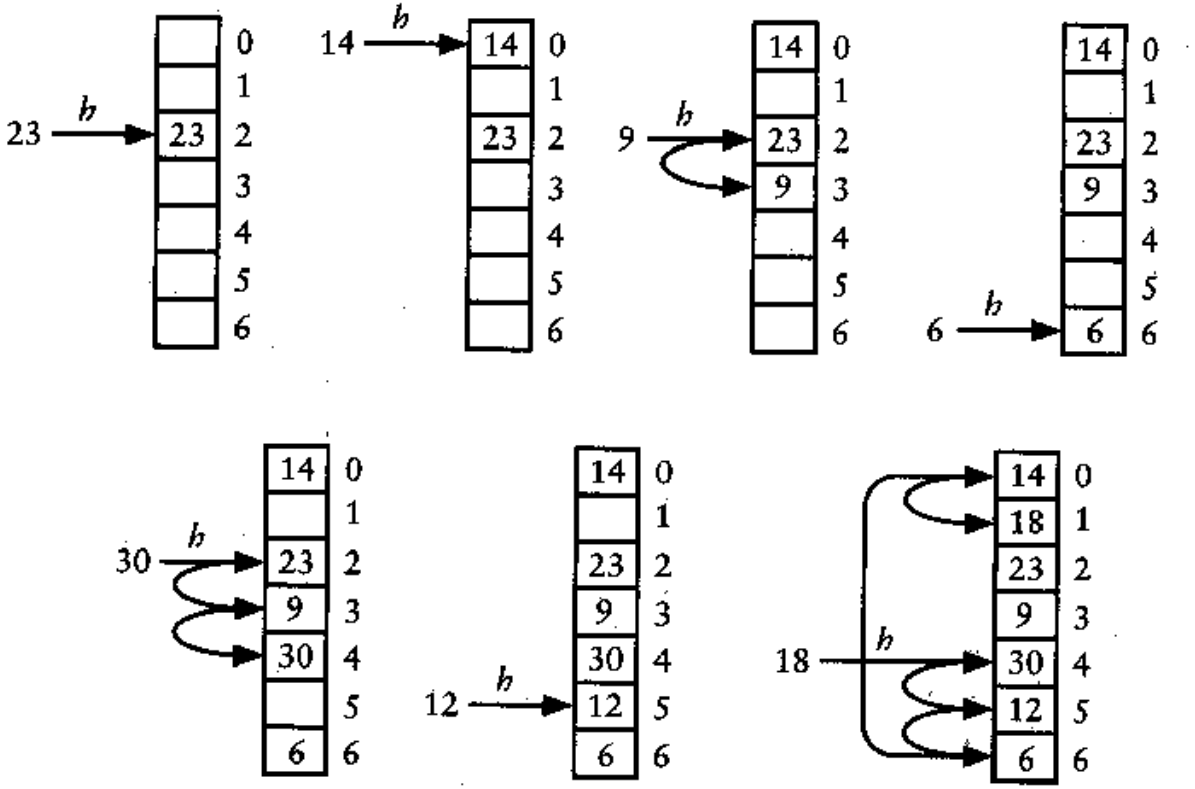
$$h_i(x) = \left(Hash(x) + f_i(i)\right) \bmod HSize$$

con *f(0)=0.* La función *f* es la estrategia de manejo de colisiones.

Ejemplo

{23, 14, 9, 6, 30, 12, 18}

$h(x) = x \bmod 7$



Closed hashing

- Existen algunos problemas con esta estrategia:
  - El tiempo para encontrar una celda vacía puede ser largo.
  - Pueden aparecer bloques de celdas ocupadas (clusters).
  - El número esperado de intentos es cercano a $\frac{1}{2}(1+1/(1-\lambda)^2)$ para búsquedas infructuosas, donde el factor de carga λ es la relación del número de elementos en la tabla al tamaño de la tabla.

Closed hashing

- Si para el caso de la estrategia lineal es mala idea dejar que la tabla llegue a estar casi llena, en el caso cuadrático es aún peor. No hay garantía de encontrar una celda vacía una vez que la tabla está más allá de la mitad de llena.

Observaciones

- Es muy importante que el tamaño de la tabla sea un número primo.
- Recuerde, un número primo sólo es divisible por sí mismo y por la unidad.

- Hay dos cuestiones que se deben considerar a la hora de diseñar una tabla Hash.
  - Primero seleccionar una buena función de dispersión.
  - Segundo, seleccionar un buen método para resolver colisiones.

Observaciones

- El principal inconveniente del direccionamiento abierto es el espacio adicional de cada elemento, necesario para enlazar un nodo de la lista con el siguiente nodo.

- La exploración lineal es fácil de implementar en cualquier lenguaje de programación. El principal inconveniente, es cuando el factor de carga de la tabla es superior al 50%. En estos casos, situar los elementos en posiciones contiguas aumenta el tiempo medio de la operación de búsqueda.