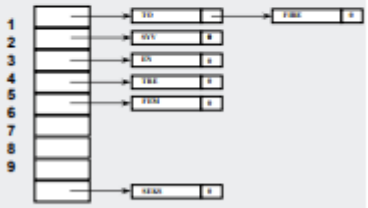
Resolución de colisiones por encadenamiento.

- Mantener N listas enlazadas,
- N cabeceras de listas.
- Inserción en listas múltiples.

Ejemplo: N = 9, K = EN, TO, TRE, FIRE, FEM, SEKS, SYV

h(K) +1 =3, 1, 4, 1, 5, 9, 2



9

Resolución de colisiones por doble desmenuzamiento

- Aunque un valor fijo de **c** reduce el fenómeno de amontonamiento, podemos mejorar la situación haciendo depender **C** de **K**.
- La idea es utilizar una segunda función de desmenuzamiento $h_2(K)$, para calcular el c correspondiente.
- Funciones sugeridas:
  - Si N es primo y $h_1(K) = K \bmod N$:
    - $h_2(K) = 1+(K \bmod (N-1))$
    - $h_2(K) = 1+(K \bmod (N-2))$

    (N y N-2 parejas de primos 1019 1021)
  - $h_2(K) = 1+(int(K/N) \bmod (N-2))$

    (int(K/M) puede estar disponible como subproducto del cálculo de h_1

19

Hashing, desventajas

- Tamaño fijo de la tabla: es necesaria una buena estimación "a priori" del número de elementos a clasificar.
- En caso de que se conozca el tamaño del conjunto, para lograr un buen rendimiento normalmente se dimensiona la tabla un 10% más grande de lo necesario.
- Si además de insertar y buscar, también es necesario eliminar, estas estructuras son muy ineficientes. La eliminación es un proceso muy difícil, a menos que se utilice encadenamiento directo en un área de desbordamiento independiente. Algunos sistemas operativos utilizan variantes de este método para clasificación externa.

TDA Mapa

- Almacena una colección de objetos en la forma ***clave-valor***
  - *tamaño()*
  - *estaVacio()*
  - *recuperar(k)*
  - *poner(k,v)*
  - *eliminar(k)*
  - *claves()*
  - *valores()*
  - *elementos()*

TDA Diccionario

- *tamaño()*
- *estaVacio()*
- *buscar(k)*
- *buscarTodos(k)*
- *insertar(k,v)*
- *eliminar(e)*
- *elementos()*
  - *Cada elemento tiene **getKey** y **getValue***

6

Collection	Ordering	Random Access	Key-Value	Duplicate Elements	Null Element	Thread Safety
ArrayList	✓	✓	✗	✓	✓	✗
LinkedList	✓	✗	✗	✓	✓	✗
HashSet	✗	✗	✗	✗	✓	✗
TreeSet	✓	✗	✗	✗	✗	✗
HashMap	✗	✓	✓	✗	✓	✗
TreeMap	✓	✓	✓	✗	✗	✗
Vector	✓	✓	✗	✓	✓	✓
Hashtable	✗	✓	✓	✗	✗	✓
Properties	✗	✓	✓	✗	✗	✓
Stack	✓	✗	✗	✓	✓	✓
CopyOnWriteArrayList	✓	✓	✗	✓	✓	✓
ConcurrentHashMap	✗	✓	✓	✗	✗	✓
CopyOnWriteArraySet	✗	✗	✗	✗	✓	✓

29

Property	HashMap	LinkedHashMap	TreeMap
Time Complexity (Big O notation) Get, Put, ContainsKey and Remove method	O(1)	O(1)	O(log n)
Iteration Order	Random	Sorted according to either Insertion Order or Access Order (as specified during construction)	Sorted according to either natural order of keys or comparator (as specified during construction)
Null Keys	allowed	allowed	Not allowed if Key uses Natural Ordering or Comparator does not support comparison on null Keys
Interface	Map	Map	Map, SortedMap and NavigableMap
Synchronization	none, use Collections.synchronizedMap()	None, use Collections.synchronizedMap()	none, use Collections.synchronizedMap()
Data Structure	List of Buckets, if more than 8 entries in bucket then Java 8 will switch to balanced tree from linked list	Doubly Linked List of Buckets	Red-Black Tree (a kind of self-balancing binary search tree) implementation of Binary Tree. This data structure offers O (log n) for Insert, Delete and Search operations and O (n) Space Complexity
Applications	General Purpose, fast retrieval, non-synchronized. ConcurrentHashMap can be used where concurrency is involved.	Can be used for LRU cache, other places where insertion or access order matters	Algorithms where Sorted or Navigable features are required. For example, find among the list of employees whose salary is next to given employee, Range Search, etc.
Requirements for Keys	Equals() and hashCode() needs to be overwritten	Equals() and hashCode() needs to be overwritten	Comparator needs to be supplied for Key implementation, otherwise natural order will be used to sort the keys

Universidad Católica del Uruguay