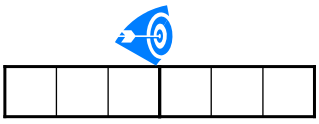
Representación de datos

Tablas Hash



Índice

- ***¿Qué son?***
  - Por qué son importantes
  - Características
- ***Métodos principales***
- ***Ejemplos de implementación***
  - Tabla hash cerrada
  - Tabla hash abierta

Tablas hash
Motivación

Se quieren guardar y consultar los tiempos realizados por 1000 atletas, cuyos dorsales están enumerados en el rango de [0…999]

3

Tablas hash
Motivación

Se quieren guardar y consultar las nominas de 1000 Empleados, cuyos DNIs están enumerados en el rango de [00000000…99999999]

4

Tablas hash

Por qué son importantes? Objetivos

Dado un conjunto de elementos ejecutar las operaciones básicas de la manera más eficiente:

- **Búsqueda**
- **Inserción**
- **Eliminación**

5

Tablas Hash

Qué son? Para qué se utilizan?

• Es una estructura de datos que permite indexar/localizar un conjunto de elementos a través de una función (función hash).

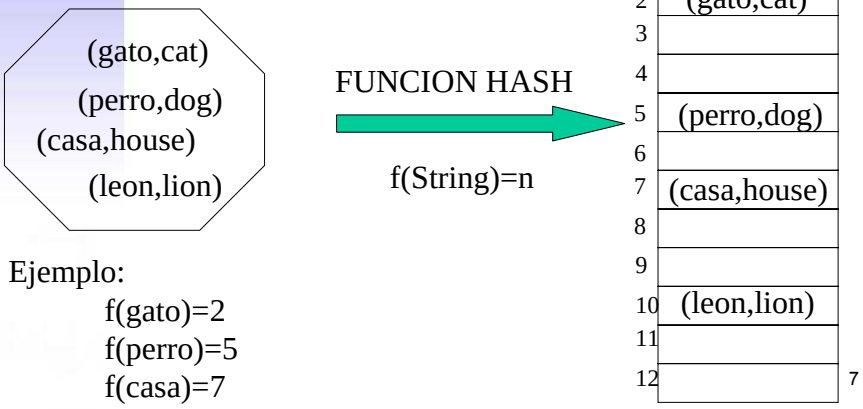
• La estructura que se utiliza es un array

• Ejemplos:
  • Almacenar palabras de un diccionario
  • Almacenar un conjunto de elementos dispersos
    • Empleados de una empresa
    • Conjunto de números

6

Tablas Hash

Qué son? Para qué se utilizan?



Ejemplo:

f(gato)=2
f(perro)=5
f(casa)=7

7

Tablas Hash

La función hash

- Es una función que a partir de una clave devuelve la posición en la tabla que le corresponde
- Características (deseables)
  - Cada elemento una posición distinta
  - Orden complejidad constante

8

Tablas Hash

Métodos principales (interfaz)

public interface Map<K,V>

Métodos	*Significado*
V put (K key, V value);	Introduce el elemento *value* con clave *key*. Si la clave key existe, entonces sustituye el nuevo valor (*value*) por el anterior. En este caso devuelve el valor anterior. Si la clave no existe devuelve **null**.
V get (K key);	Devuelve el objeto con clave *key*, o **null** si no existe.
V remove (K key);	Elimina la clave *key* y su correspondiente valor, a la vez que devuelve dicho valor. Si no existe, devuelve **null**.
void clear();	Elimina todos las claves y valores.
boolean isEmpty();	Devuelve True si está vacio o false, en caso contrario
boolean containsKey(K key);	Devuelve True si la clave existe o false, en caso contrario
int size();	Devuelve el número de valores en la tabla

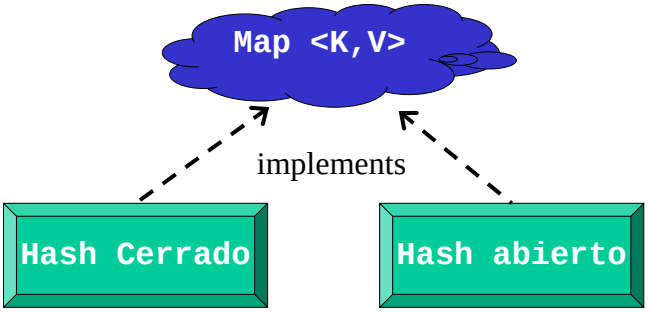
Tablas Hash

Métodos principales (interfaz)

```
public interface Map <K,V> {
    public V put (K key, V value);
    public V get (K key);
    public V remove (K key);
    public void clear();
    public boolean isEmpty();
    public boolean containsKey(K key);
    public int size();
}
```

Informazio osagarria:
http://java.sun.com/j2se/1.5.0/docs/api/java/util/Map.html#get(java.lang.Object)

Tablas Hash:
Una interfaz y varias implementaciones



11

Tablas Hash:
Hash cerrado

- Una tabla hash es cerrada si el número de elementos que puede almacenar es fijo (MAXPOS)
- La función hash debe obtener a partir de la clave un número positivo:
  - hash(“JAVA”)=10+1+23+1=35
  - hash(“15564465”)=36
- Por aspectos de eficiencia se recomienda definir la tabla con el doble de elementos de los que se quiere almacenar

12

Tablas Hash:
Hash cerrado

- A partir del resultado de la función hash, se obtiene la posición de la clave en la tabla hash
  - indice=1+(hash(clave)%MAXPOS)
  - Si MAXPOS=20
  - La posición de la clave 15564465 seria ?
  - La posición de la clave “JAVA” sería ?

13

Tablas Hash:
Hash cerrado

- ¿Que ocurre si esta posición ya está ocupada por otra clave sinónima* (colisión) ? ➔
- ***Buscar en la tabla una posición libre lo antes posible !!***
- ***“CC” es sinónimo de “ADA” con la función hash utilizada***

14

Tablas Hash:
Colisión: Cómo localizar la posición?

- Prueba lineal: buscar la primera posición libre
  - ***while posiciónIndice ocupada do***
    Indice := (Indice + 1) % MAXPOS
- Prueba del número primo: Se selecciona un número primo P que no divida a MAXPOS (si MAXPOS=15 → P = 13)
  Indice:= (Indice + P) % MAXPOS
- Doble hashing: *La nueva posición se calcula mediante una nueva función hash*

15

Tablas Hash:
Implementación Prueba lineal: los datos y la tabla

```
class DataItem<K,V> {
      public K key;              // data item (key)
      public V data;
      public DataItem(K k, V d) { // constructor
        key = k;
        data = d; }
   }  // end class DataItem
```

```
class HashTableCerrado<K,V> implements Map<K,V>{
   DataItem<K,V>[] hashArray;  // array holds hash table
   int arraySize;
   DataItem<K,V> nonItem;
```

16

Tablas Hash:
El constructor y la función hash

```
public HashTableCerrado (int size) { // constructor
      arraySize = size;
      hashArray = new DataItem<K,V>[arraySize];
      nonItem = new DataItem<K,V>(new Integer(-1),null);
           // deleted item key is -1
      }
```

```
public int hashFunc(K key){
      return (key.hashCode() % arraySize; // hash function
                        // cada objeto tiene un código distinto,
                        // asignado por el sistema
}
```

Tablas Hash:
El método find. Prueba lineal

```
public boolean containsKey(K pkey) { // find item with key
     int hashVal = hashFunc(pkey);  // hash the key

     while( (hashArray[hashVal] != null)  &&
            (!hashArray[hashVal].key.equals(pkey)) ) {// not found
        hashVal++;                // go to next cell
        hashVal = hashVal % arraySize; // wraparound if necessary
      }
      if(hashArray[hashVal]==null)
           return false;   // can't find item
       else
           return true;
     }
```

Tablas Hash:
El método delete. Prueba lineal

```
public V remove(K pKey) { // delete a DataItem
     int hashVal = hashFunc(pkey);  // hash the key

     while ( (hashArray[hashVal] != null)  &&
           (!hashArray[hashVal].key.equals(pkey)) ) { // not found
        hashVal++;                                  // go to next cell
        hashVal = hashVal % arraySize;              // wraparound if necessary
      }

     if (hashArray[hashVal]==null)
           return null;                                        // can't find item
     else {
          DataItem<K,V> temp = hashArray[hashVal];  // save item
          hashArray[hashVal] = nonItem;                       // delete item
          return temp.data;                                   // return item
     }
}  // end remove()
```

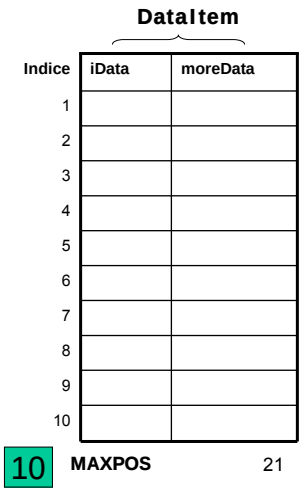
19

Tablas Hash:
El método insert. Prueba lineal

```
public V put(K pKey, V pData) { // insert a DataItem
   int hashVal = hashFunc(pkey);
   if (!this.containsKey(pKey)){
      while ( (hashArray[hashVal] != null) && (hashArray[hashVal]!=nonItem)){
          hashVal++;                        // go to next cell
          hashVal = hashVal % arraySize // wraparound if necessary
       }
       hashArray[hashVal] = new DataItem(pKey, pData);    // insert item
       return null;}
    else {
       while(!hashArray[hashVal].key.equals(pKey)) {
          hashVal= hashVal + 1;
          hashVal = hashVal % arraySize;
       }
       // if key exits, then change the value
       V temp = hashArray[hashVal].data;
       hashArray[hashVal].data=pData;
       return temp; }}
}  // end put()
```

Ejemplo Tabla hash cerrada

th=new HashTable(10)



Ejemplo Tabla hash cerrada

Suponemos que las claves son enteros de cuatro digitos (dddd):
HASH ('dddd') → d+d+d+d y
que las colisiones se resuelven mediante prueba lineal.

Dibujar el estado de la tabla hash tras las siguientes operaciones:

th.insert (new Integer(1237), "ADA");
th.insert (new Integer(2237), "C")
th.insert (new Integer(0111), "JAVA"));
th.modify(new Integer(2237),"C++");
th.remove (new Integer(0111));
th.insert(new Integer(1111), "C#")
th.remove(new Integer(1237));

22

Ejemplo Tabla hash cerrada

Indice	clave	datos
1		
2		
3		
4	1237	ADA
5	2237	C++
6	1111	C#
7		
8		
9		
10		

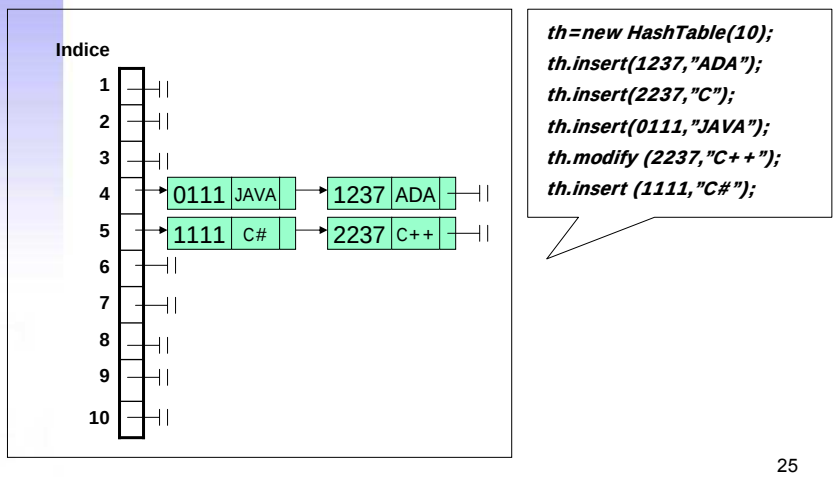
23

Tablas Hash:
Hash abierto

- **No existe** límite lógico de elementos
- Se utiliza una función hash para determinar la posición relativa del array que le corresponde a una clave
- Todos los sinónimos se almacenan en una lista ligada de su posición natural
- Se elimina el tratamiento de colisiones

24

Ejemplo Tabla hash abierta



Tablas hash abiertas

Analisis

- Si factor de carga es λ ➔ longitud media de cada lista es λ
  - Una búsqueda sin éxito visita λ nodos
  - Una búsqueda con éxito visita 1 + ½ λ nodos
- Si el factor de carga aumenta ➔ la eficiencia de búsqueda disminuye
- Si el factor de carga disminuye ➔ la memoria no utilizada aumenta

Tablas Hash:
Implementación: los datos y la tabla

```
class DataItem<K,V> {
      public K key;              // data item (key)
      public V data;
      public DataItem(K k, V d) { // constructor
        key = k; }
        data=d;
   }  // end class DataItem
```

```
class HashTableAbierto<K,V> implements Map<K,V> {
    LinkedList<DataItem<K,V>>[] hashArray;  // array holds hash table
     int arraySize;
}
```

27

Tablas Hash:
El constructor y la función hash

```
public HashTableAbierto(int size) { // constructor
      arraySize = size;
      hashArray = new LinkedList<DataItem<K,V>> [arraySize];
      for (int j=0;j<arraySize;j++)
           hashArray[j] = new LinkedList<DataItem<K,V>> ();
      }
```

```
public int hashFunc(K key) {
  return (key.hashCode() % arraySize; // hash function
      }
```

28

Tablas Hash:

El método insert. Prueba lineal

```
public V put (K pKey, V pData) { // insert a DataItem
   if (! containsKey(pKey)) {
      int hashVal = hashFunc(pKey); // hash the key
      hashArray[hashVal].insertFirst(new DataItem<K,V>(pKey, pData));
      return null;}
   else {
      int hashVal = hashFunc(pKey);
      hashArray[hashVal].goFirst();
      while (!hashArray[hashVal].getCurrent().key.equals(pKey)){
         hashArray[hashVal].goNext();
      }
      V elem = hashArray[hashVal].getCurrent().data;
       hashArray[hashVal].removeCurrent();
       hashArray[hashVal].insertFirst(new DataItem<K,V>(pKey, pData));
       return elem;
      }
} // end put()
```

Tablas Hash:

No sirven cuando.....

- No se conoce a priori el número de elementos
- Se deben recorrer los elementos en orden