ESTRUCTURAS DE DATOS

DICCIONARIOS

Tablas hash redimensionables

Manuel Montenegro Montes

Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Recordatorio: factor de carga

- El **factor de carga** α de una tabla *hash* es el cociente entre el número de entradas en la tabla y el número de cajones.
- Sean:
 - n número de entradas en la tabla
 - *m* número de cajones

$$\alpha = \frac{n}{m}$$

Del vídeo anterior sacamos una conclusión importante: las operaciones en una tabla hash tienen coste constante bajo ciertas suposiciones.

Tablas redimensionables

- En el caso medio, las operaciones en una tabla hash abierta tienen coste $O(1 + \alpha)$.
- Por tanto, conseguimos coste constante en el caso medio si mantenemos el factor de carga acotado.
- Una **tabla** *hash* **redimensionable** es una <u>tabla que se amplía ca</u>da vez que el factor de carga supera un determinado valor umbral.



Implementación

FACTOR DE CARGA MÁXIMO QUE VAMOS A PERMITIR

```
const int INITIAL CAPACITY = 31;
                                            Factor de carga máximo permitido
const double MAX LOAD FACTOR = 0.8;
template <typename K, typename V, typename Hash = std::hash<K>>
class MapHash {
private:
  using List = std::forward list<MapEntry>;
  Hash hash;
  List *buckets;
  int num elems;
                                Tamaño del vector
 int capacity;
};
```

Implementación de insert (antes)

```
template <typename K, typename V, typename Hash = std::hash<K>>
class MapHash {
  void insert(const MapEntry &entry) {
    int h = hash(entry.key) % capacity;
    auto it = find in list(buckets[h], entry.key);
    if (it = buckets[h].end()) {
      buckets[h].push front(entry);
      num elems++;
private:
```

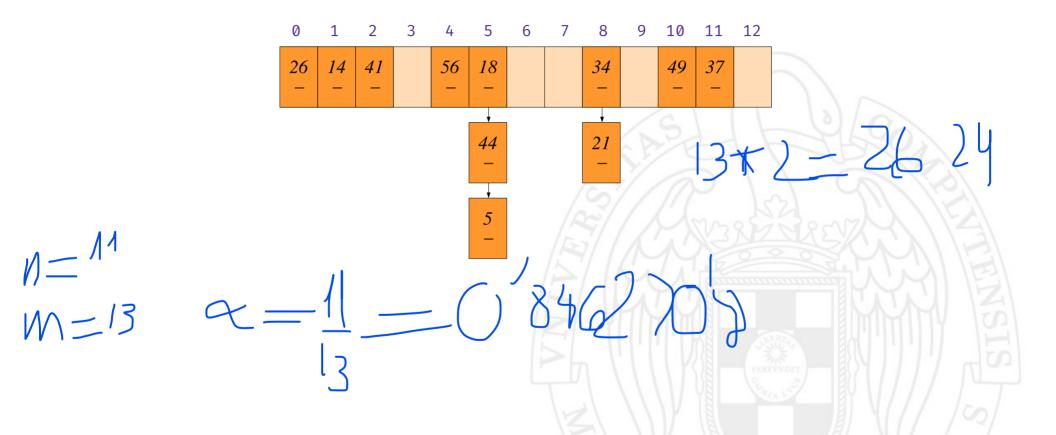
Implementación de insert (después)

```
template <typename K, typename V, typename Hash = std::hash<K>>
class MapHash {
  void insert(const MapEntry &entry) {
    int h = hash(entry.key) % capacity;
    auto it = find in list(buckets[h], entry.key);
    if (it = buckets[h].end()) {

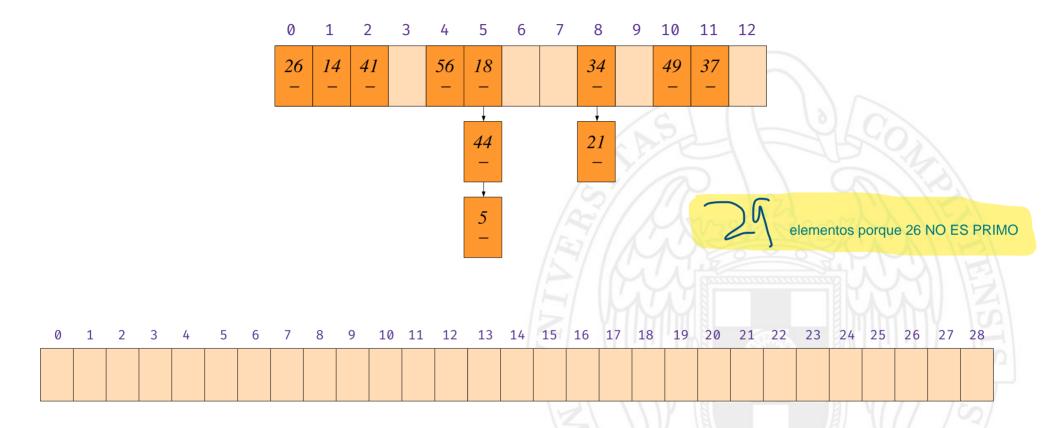
    En el caso de que no se encuentre primero aumentamos el número de elementos

       num elems++;-
       resize_if_necessary(); comprueba el factor de carga y si supera la constante anterior, entonces redimensiona la tabla
       h = hash(entry.key) % capacity;
puede cambiar la capacity
       buckets[h].push_front(entry);
private:
```

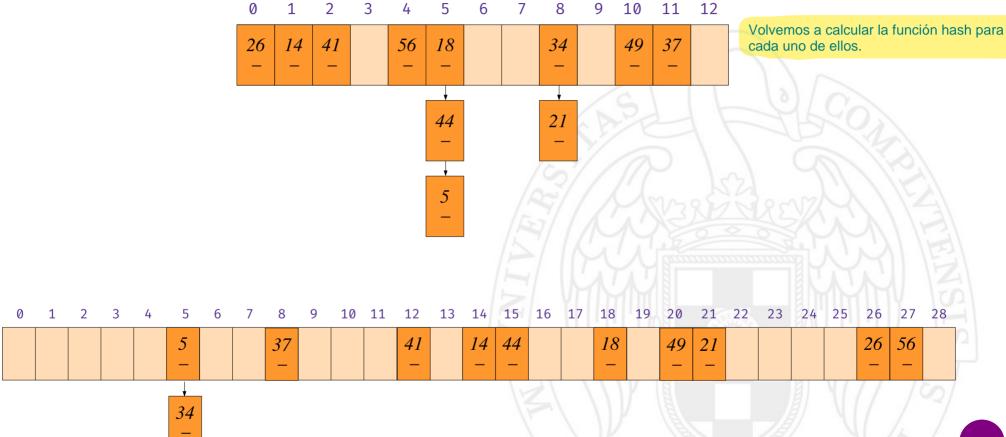
Ejemplo de redimensionamiento



Ejemplo de redimensionamiento



Ejemplo de redimensionamiento



Método auxiliar de redimensionamiento

```
template <typename K, typename V, typename Hash = std::hash<K>>>
class MapHash {
private:
                                              Casteo porque el número de elementos es int.
  void resize if necessary() {
    double load factor = ((double)num elems) / capacity;
                                                                   SI EL FACTOR DE CARGA NO SUPERA EL UMBRAL MÁXIMO
    if (load factor < MAX_LOAD_FACTOR) return;</pre>
    int new_capacity = next_prime_after(2 * capacity);
                                                                         Calculamos el siguiente primo más cercano con capacidad
    List *new array = new List[new capacity];
                                                                         mínima 2 por actual capacity
    for (int i = 0; i < capacity; i++) {
      for (const MapEntry &entry : buckets[i]) {
         int new pos = hash(entry.key) % new capacity;
         new array[new pos].push front(entry); `
                                                              Para cada una de las posiciones del vector
    capacity = new_capacity;
    delete[] buckets;
    buckets = new array;
                                                              Volvemos a calcular el valor hash de esa clave
                                                    Inserto al principio en el nuevo array,
```

Costes en tiempo

Suponiendo dispersión uniforme

Operación	Tabla <i>hash</i>	
constructor	O(1)	
empty	O(1)	360)
size	O(1)	
contains	O(1)	3.000
at	O(1)	
operator[]		que en el caso en el que no se encuentre la ve, la inserta
insert		si tenemos que redimensionar
erase	O(1)	AND THE STATE OF T

n = número de entradas en la tabla

Costes amortizados en tiempo

Suponiendo dispersión uniforme.

Operación	Tabla <i>hash</i>
constructor	O(1)
empty	O(1)
size	O(1)
contains	O(1)
at	O(1)
operator[]	O(1)
insert	O(1)
erase	O(1)

n = número de entradas en la tabla