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

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

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 tabla que se amplía cada vez que el factor de carga supera un determinado valor umbral.

Implementación

```
const int INITIAL_CAPACITY = 31;
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;
  int capacity;
   ...
};
```

Factor de carga máximo permitido

Tamaño del vector

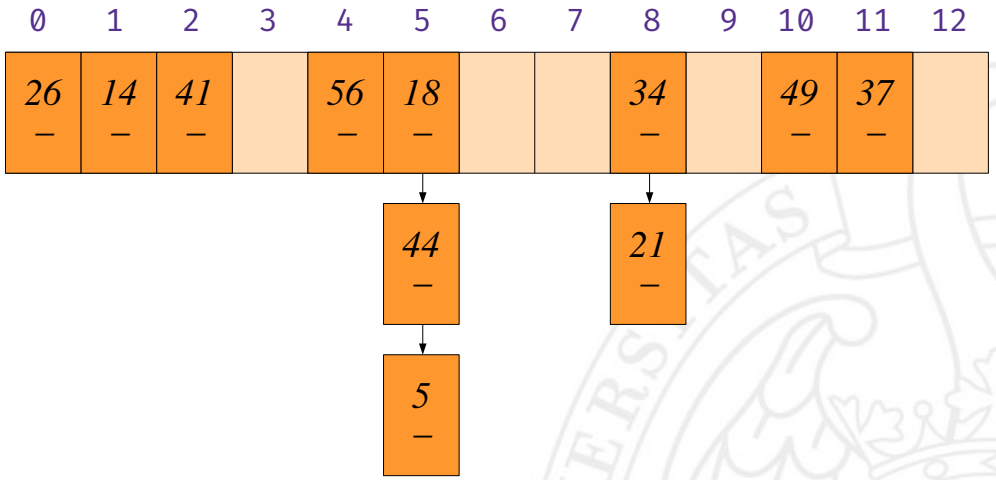
Implementación de `insert` (antes)

```
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:
   ...
};
```

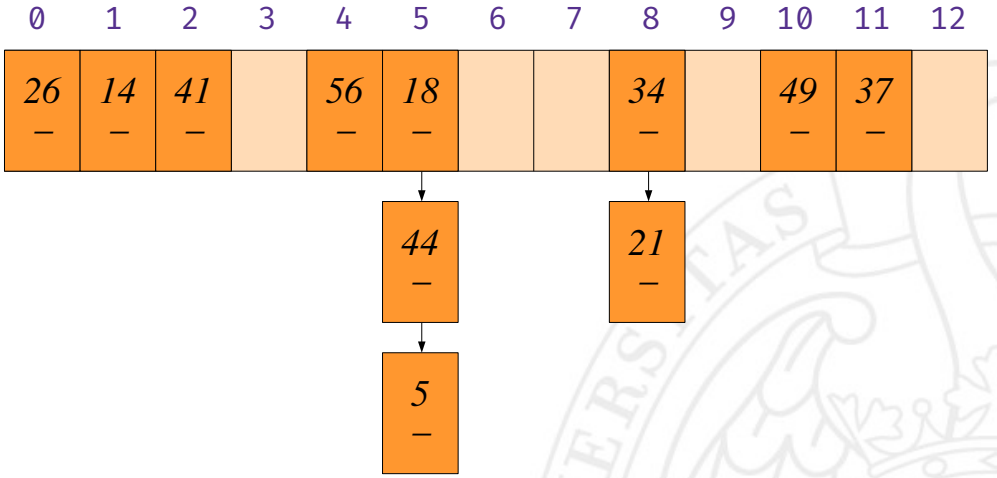
Implementación de `insert` (después)

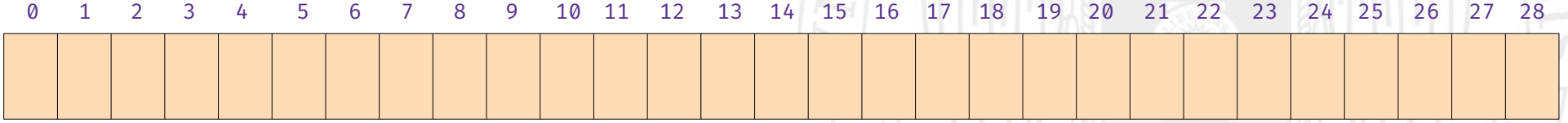
```
template <typename K, typename V, typename Hash = std::hash<K>>
class MapHash {
   ...

  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()) {
      num_elems++;
      resize_if_necessary();
      h = hash(entry.key) % capacity;
      buckets[h].push_front(entry);
    }
  }


private:
   ...
};
```

Ejemplo de redimensionamiento

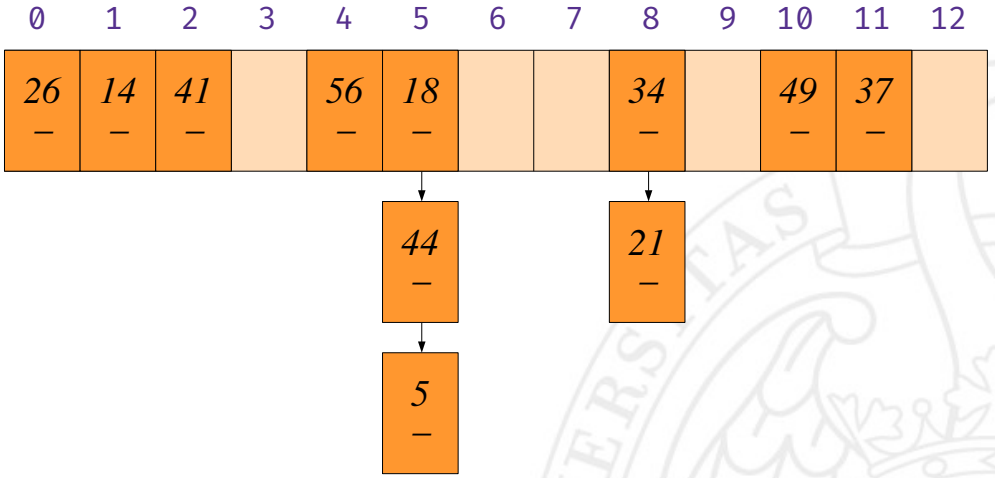


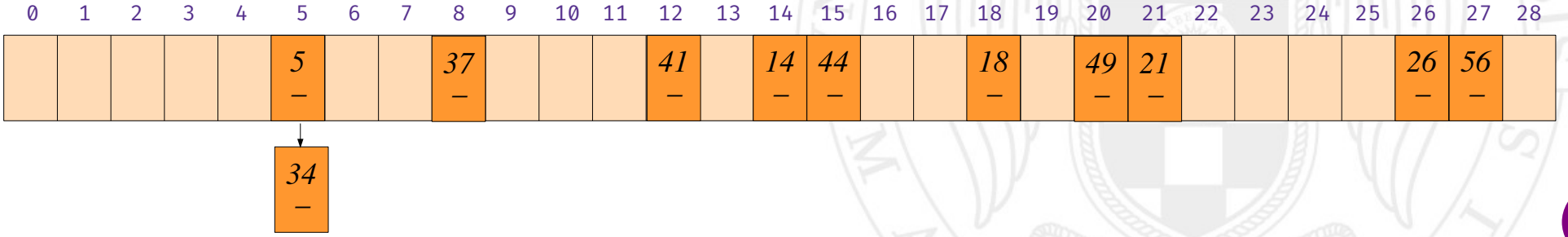
Ejemplo de redimensionamiento





Ejemplo de redimensionamiento





Método auxiliar de redimensionamiento

```
template <typename K, typename V, typename Hash = std::hash<K>>
class MapHash {
private:

  void resize_if_necessary() {
    double load_factor = ((double)num_elems) / capacity;
    if (load_factor < MAX_LOAD_FACTOR) return;

    int new_capacity = next_prime_after(2 * capacity);
    List *new_array = new List[new_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);
      }
    }

    capacity = new_capacity;
    delete[] buckets;
    buckets = new_array;
  }
  ...
};
```

Costes en tiempo

- Suponiendo **dispersión uniforme**

Operación	Tabla *hash*
constructor	$O(1)$
`empty`	$O(1)$
`size`	$O(1)$
`contains`	$O(1)$
`at`	$O(1)$
`operator[]`	$O(1)$ / $O(n)$
`insert`	$O(1)$ / $O(n)$
`erase`	$O(1)$

n = número de entradas en la tabla

Costes amortizados en tiempo

- Suponiendo **dispersión uniforme**.

Operación	Tabla *hash*
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