Tablas hash

Carlos E. Alvarez[1].

[1]Dep. de Matemáticas aplicadas y Ciencias de la Computación, Universidad del Rosario

2019-II

Tablas hash

¿Por qué usar hashing?

- Los mapas son tipos de datos muy utilizados en donde quiera que sea útil usar una clave para referirse a un objeto ¿Cuál es la mejor manera de implementarlos?
- Lista enlazada de pares: Acceso de $\Theta(N)$ (lento)
- Arreglos de pares: inserción/remoción lentas en el peor caso $\Theta(N)$ o peor. Pero accesso muy rápido $O(1)$
- Árbol binario de pares: Operaciones (excepto recorrer) del orden $O(\lg N)$ si es AVL
- ¿Podríamos tener acceso de $O(1)$, pero con mejores tiempos de inserción/borrado que un arreglo?

Mapa

ADT de Map:
Estructura (S) que guarda parejas ordenadas x=(key, value), en donde a cada valor (x.value) lo identifica una clave (x.key).

ADT de Map (operaciones):

- Find(S, x.key): Retorna la ubicación i del elemento con clave x.key, si este se encuentra en S. De lo contrario retorna -1
- Insert(S, x): Si x.key no está en S, inserta el elemento x en S. De lo contrario, asigna el nuevo valor a x.value
- Remove(S, x.key): Remueve el elemento con clave x.key, si se encuentra en S. De lo contrario produce un error

Mapa

Implementación sencilla

```
#include <stdexcept>

template <typename KT, typename VT>
class MyMap {
private:
  struct KeyValuePair {
    KT key;
    VT value;
  };

  KeyValuePair *array;
  int capacity;
  int count;

  void expandCapacity();
  int findKey(KT key);
.
.
```

Mapa

```
.
.
public:
   MyMap();
   ~MyMap();

   int size();
   bool empty();
   void clear();
   void insert(KT key, VT value);
   VT get(KT key);
   bool containsKey(KT key);
   void remove(KT key);
};
```

Mapa

```
template <typename KT, typename VT>
int MyMap<KT,VT>::findKey(KT key) {
  //linear search algorithm
  for(int i = 0; i < count; i++){
    if(array[i].key == key)
    return i;
  }
  return -1;
}
```

Mapa

```
template <typename KT, typename VT>
void MyMap<KT,VT>::insert(KT key, VT value){
  int index = findKey(key);
  if(index == -1){
    if(count == capacity) expandCapacity();
    index = count++;
    array[index].key = key;
  }
  array[index].value = value;
}
```

Mapa

```
template <typename KT, typename VT>
void MyMap<KT,VT>::remove(KT key) {
  int index = findKey(key);
  if(index == -1)
    throw runtime_error("remove: Attempting to remove
                         value of non-existent key\n");
  for(int i = index; i < count-1; i++){
    array[i].key = array[i+1].key;
    array[i].value = array[i+1].value;
  }
  count--;
}
```

Mapa

Ejercicios:

- Implemente el mapa como se mostró en las diapositivas anteriores

Tablas hash

¿Por qué usar hashing?
Nuestro mapa usa un algoritmo de búsqueda del orden $O(N)$.
Presentaremos ahora la idea de función hash.

Función hash: Mapa que lleva de un espacio de claves (K) a los enteros en el intervalo $[0, \text{TabSize})$ (índice sobre un arreglo)

$$\text{hash} : K \to \{x \in \mathbb{Z} | 0 \leq x < \text{TabSize}\}$$

NOTA: La función hash en realidad NO es una función, es un mapa. Dado que más de un elemento en K puede tener la misma imagen. A este evento se le llama *colisión*.

Tablas hash

Tabla con una función hash simple para claves `int`

Función hash simple

```
int hash(int key, int tableSize) {
  return key  % tableSize;
}
```

Tablas hash

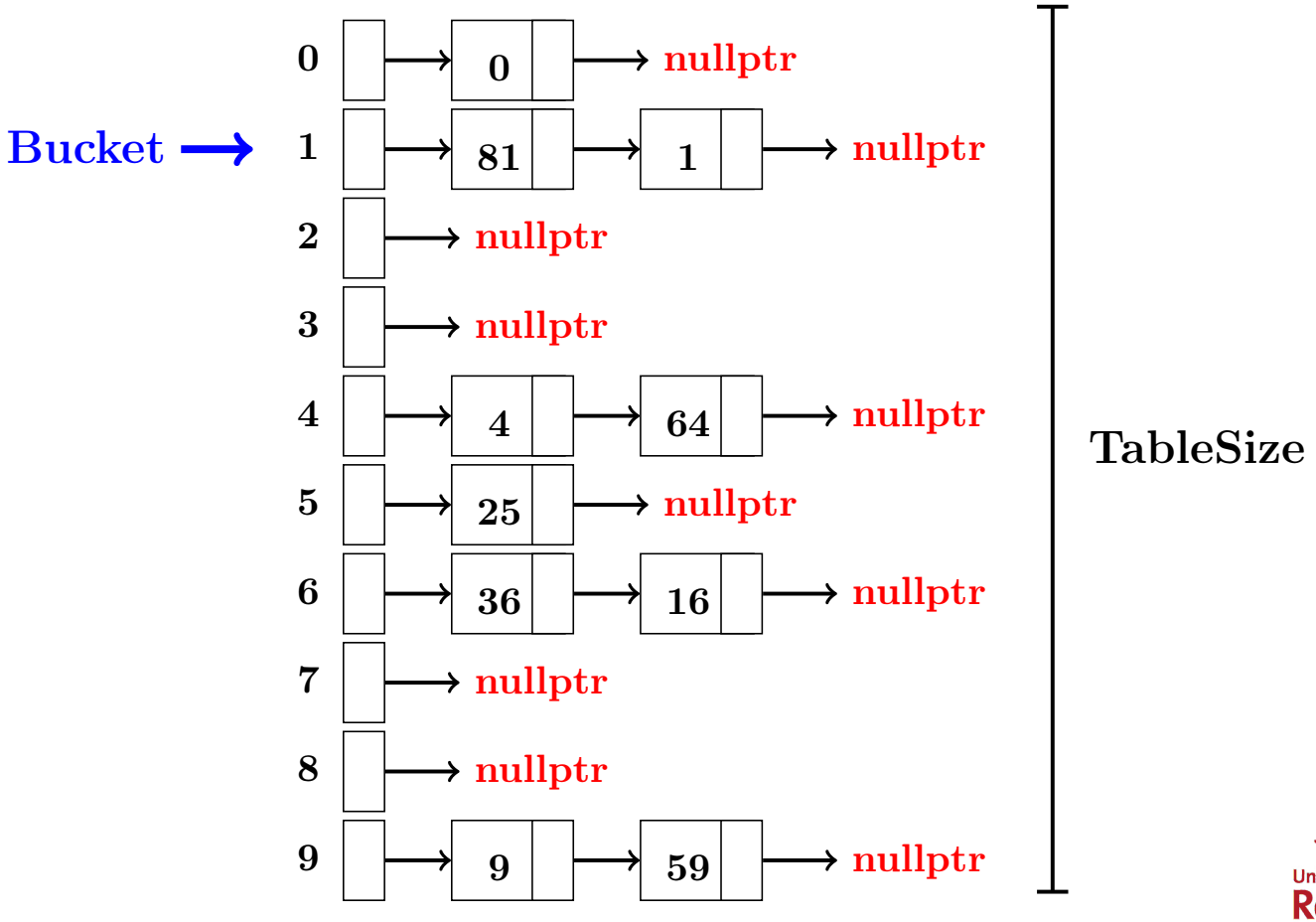
¿Por qué usar hashing?

Mientras no existan colisiones, la búsuqeda, inserción y accesso son $O(1)$, ya que después de evaluar la función para encintrar el índice el problema se reduce a ubicar un índice en un arreglo.

Tablas hash

¿Que hacer cuando se presenta una colisión?

Una posibilidad: Encadenamiento



Tablas hash

La búsqueda en una cadena puede tomar tiempos, por ejemplo del orden $O(k)$ (lineal) o $O(\lg k)$ (binaria), en donde k es la longitud de la cadena.

Clustering: La mayoría de los elementos mapea a un grupo pequeño de índices en la tabla. Esto hace que el k típico al buscar un elemento sea grande.

Una buena función hash minimisa el clustering.

Tablas hash

Dos ejemplos de función hash con claves `string`

Ejemplo 1

```
unsigned int hash1(string key, int tableSize) {
  unsigned int hashVal = 0;
  for(char c : key)
    hashVal += c;
  return hashVal  % tableSize;
}
```

Ejemplo 2

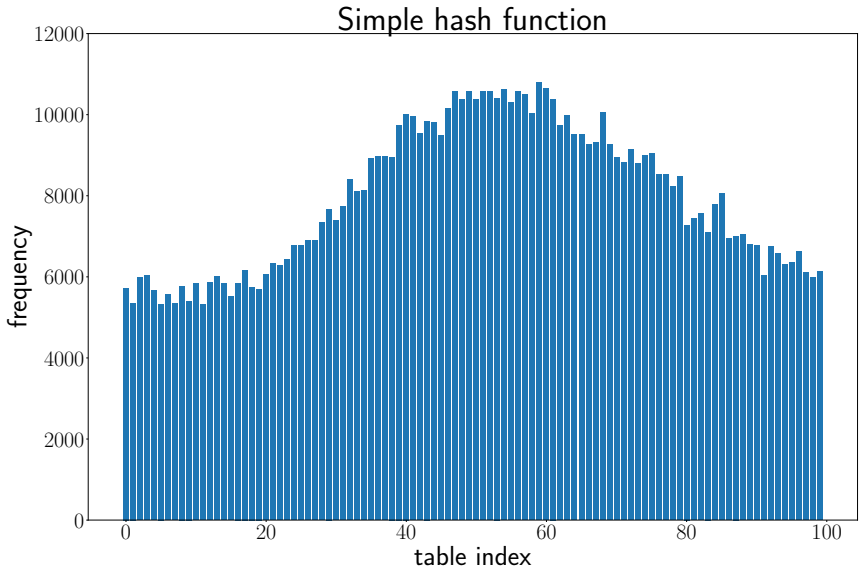
```
unsigned int hash2(string key, int tableSize) {
  unsigned int hashVal = 0;
  for(char c : key)
    hashVal += 64 * hashVal + c;
  return hashVal  % tableSize;
}
```

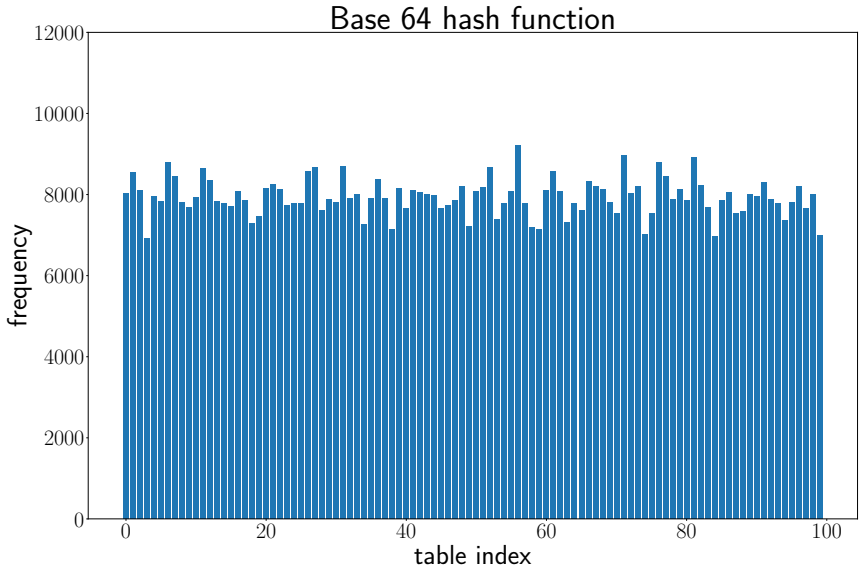
Tablas hash

Ejercicios:

- Implemente las dos funciones hash anteriores en un programa
- Haga que el programa lea las palabras del diccionario español (64 caracteres) en el archivo `palabras.txt` y las guarde en un vector de string
- Tomando TableSize = 100 y las palabras de la lista como claves, calcule el índice hash para cada palabra y cuente cuantas palabras caen en cada índice (colisiones)
- Presente los resultados como histogramas de frecuencias para cada función hash

Tablas hash


Simple hash function
frequency
table index
0
2000
4000
6000
8000
10000
12000
0
20
40
60
80
100


Base 64 hash function
frequency
table index
0
2000
4000
6000
8000
10000
12000
0
20
40
60
80
100

Tablas hash

TableSize es el número de buckets en la tabla.

El factor de carga -load factor (λ)- de la tabla es

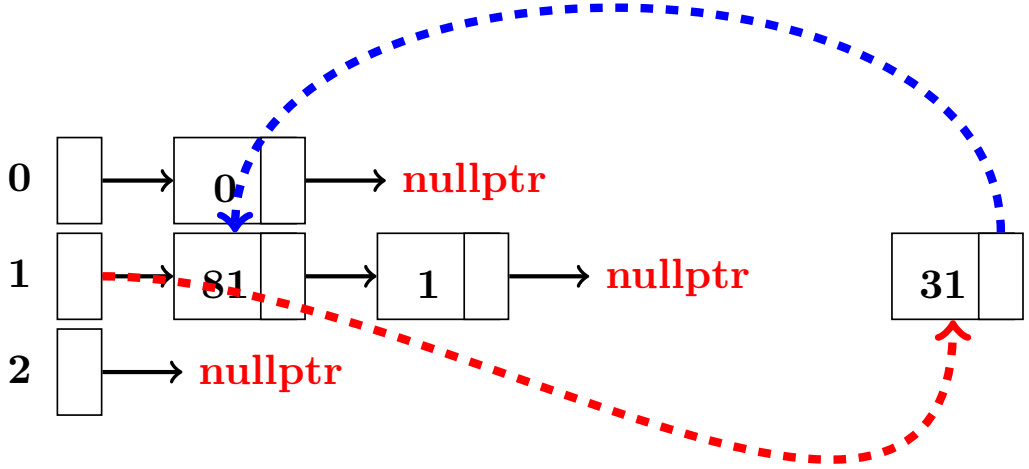
$$\lambda = \frac{N_{keys}}{\text{TableSize}},$$

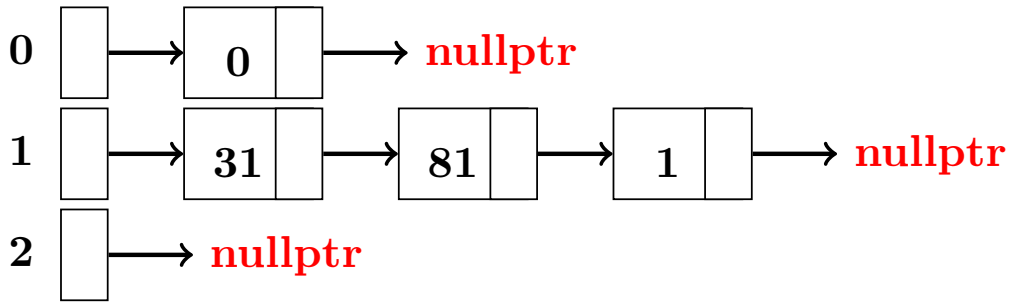
en donde N_{keys} es el número de elementos en la tabla.

λ representa el compromiso entre muchas colisiones (N_{keys} >Tablesize) y desperdicio de memoria (TableSize> N_{keys}).

Tablas hash

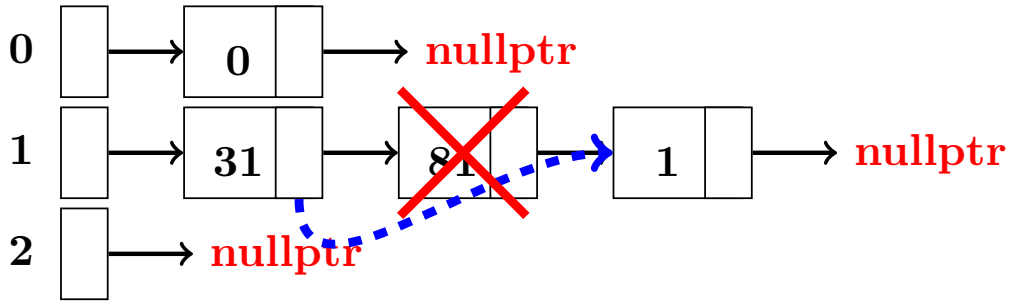
Inserción

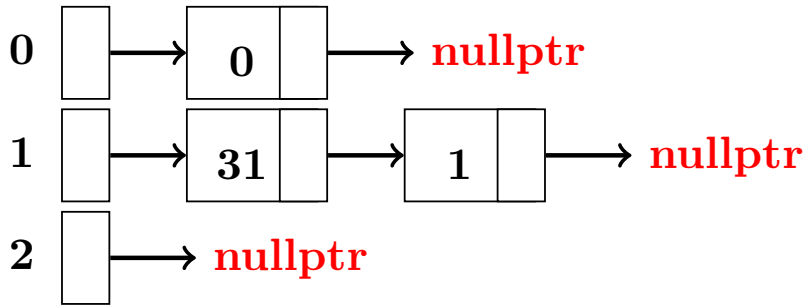




Tablas hash

Borrado





Tablas hash

Ejercicios:

- Con los conocimientos adquiridos en la implementación sencilla del mapa, las funciones hash, y su conocimiento sobre listas encadenadas, implemente un mapa que reciba claves `string` y valores de cualquier tipo, usando una tabla hash con encadenamiento
- Implemente una prueba que, dado un N_{keys} y TableSize, calcula el tiempo promedio de búsqueda en la tabla (buscando elementos aleatorios)...