Tablas de Dispersión (*hash tables*)

Estructuras de Datos

Andrea Rueda

Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas

Tablas de Dispersión

¿Búsquedas eficientes?

- Árboles binarios ordenados
- Árboles AVL
- Arboles R-N
- …

¿Existen otras posibilidades?

Tablas de Dispersión

- Problema:

  Organizar los periódicos que llegan diariamente para ubicarlos rápidamente.

Tablas de Dispersión

- Problema:

  Organizar los periódicos que llegan diariamente para ubicarlos rápidamente.

  - En una pila o lista, implica revisar uno por uno O(n)

Tablas de Dispersión

- Problema:

  Organizar los periódicos que llegan diariamente para ubicarlos rápidamente.

  - En una pila o lista, implica revisar uno por uno O(n)
  - Contenedor de 31 posiciones
    cada periódico se ubica de acuerdo al día
    varios periódicos en una misma posición
    O(log n)

Tablas de Dispersión

- Tablas de dispersión (*hash tables*)

  Método eficiente de almacenamiento, basado en arreglos, que asocia claves a valores.

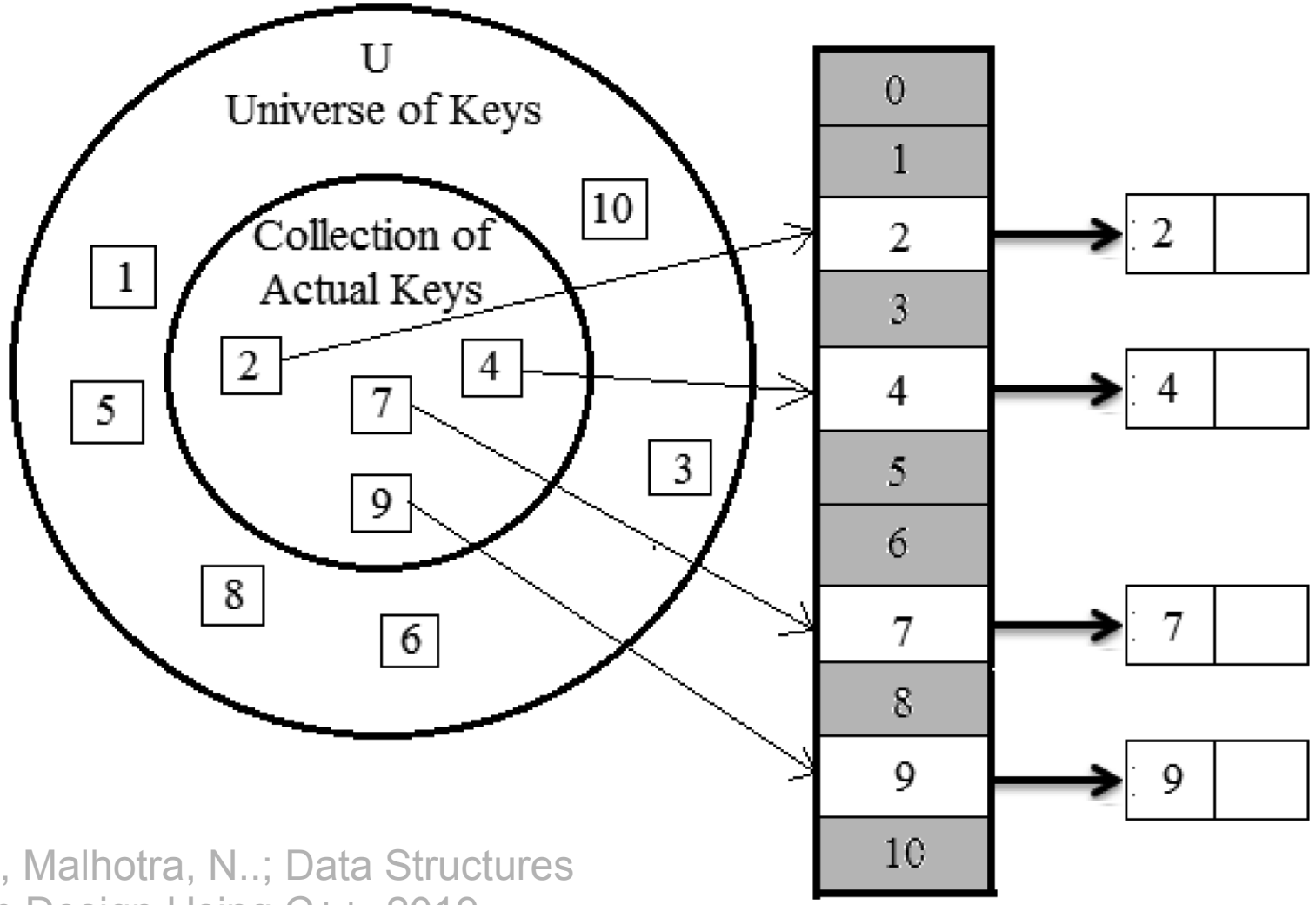
  - La clave se transforma usando una función *hash*, en un código *hash.*

  - El código identifica la posición en el arreglo del valor buscado.

  - Búsqueda se hace en O(1) con ayuda de la función *hash*.

Tablas de Dispersión

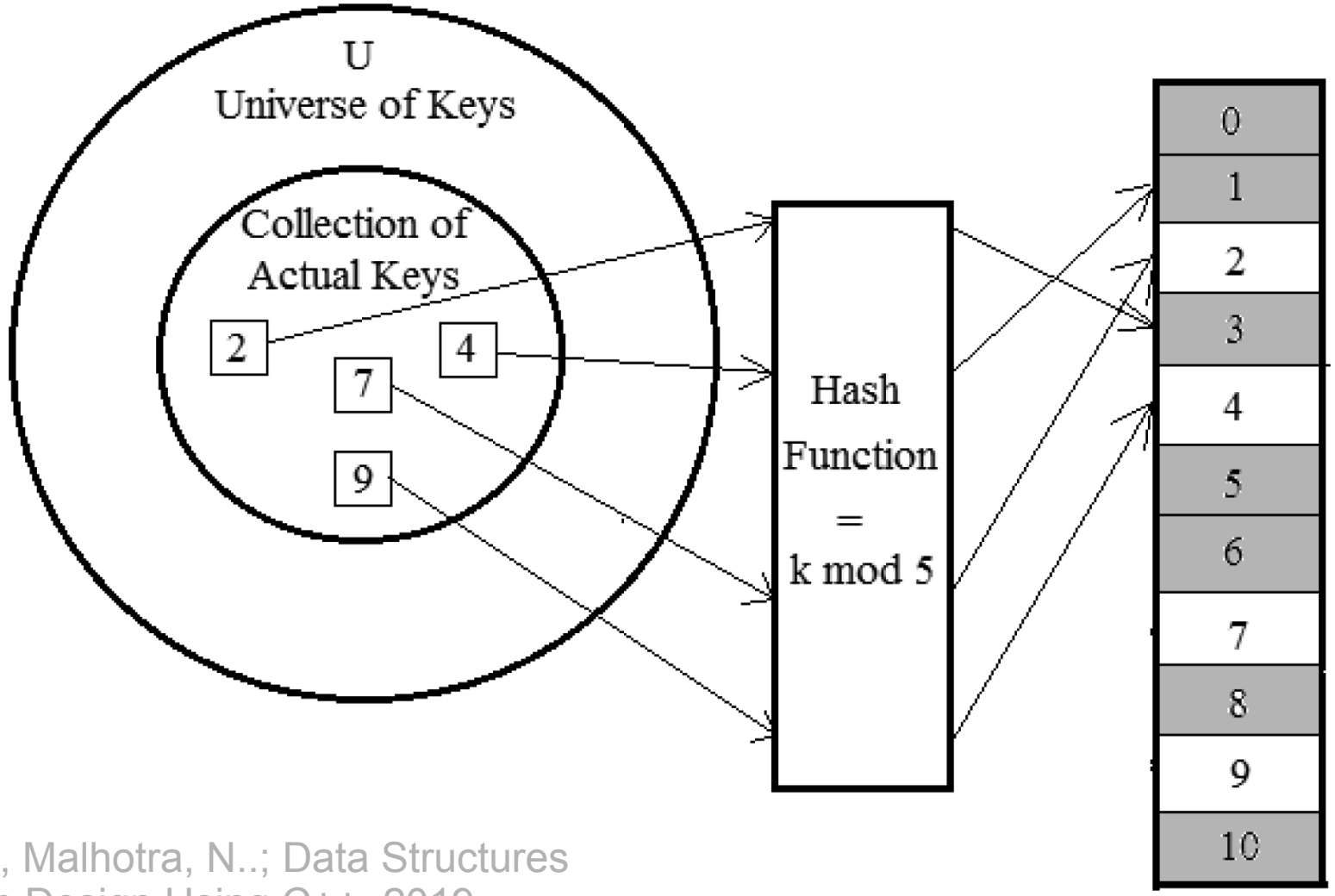
- Direccionamiento directo



Malhotra, D., Malhotra, N..; Data Structures and Program Design Using C++, 2019.

Tablas de Dispersión

- Tabla de dispersión



Malhotra, D., Malhotra, N..; Data Structures and Program Design Using C++, 2019.

Tablas de Dispersión

- Función *hash*

  Una función *hash* es una fórmula matemática que, aplicada a una clave, produce un entero que se usa como índice en la tabla de partición.

  Características:

  - Usa todos los datos de entrada
  - Debe generar diferentes códigos *hash*
  - Cada código *hash* se determina por completo a partir de los datos
  - Debe distribuir las claves de manera uniforme

Tablas de Dispersión

- Función *hash*
  Método de división

$$hash(x) = x \,\%\, N$$

siendo *N* el tamaño de la tabla.

- Método simple

- En algunos casos no distribuye bien las claves (muchas claves impares y *N* impar)

- Con *N* potencia de 2, extrae los bits más bajos de x

Tablas de Dispersión

- Función *hash*
  Hashing multiplicativo

  Basado en aritmética modular (%) y división entera (/).

  - Tamaño de la tabla hash: 2^d (*d*: dimensión)

  - Para un elemento x $\in$ {0, …, 2^w-1}

$$hash(x) = ((z \cdot x) \% 2^w) / 2^{(w-d)}$$

  z es un entero impar escogido aleatoriamente entre {1, …, 2^w-1}

Tablas de Dispersión

- Función *hash*
  Método de los medios cuadrados

  Se calcula el cuadrado de la clave dada, luego se extraen algunos dígitos del medio.

  x = 5025 -> x^2 = 25250625
  hash(x=5025) = 50

  Siempre deben tomarse los mismos dígitos (en este caso, siempre los dos de en medio), dependiendo del tamaño *N* de la tabla.

Tablas de Dispersión

- Función *hash*
  Método de doblado

  Se parte la clave en varias piezas del mismo número de dígitos (excepto la última) y se suman entre sí. Se ignora el acarreo.

  x = 12345678 se parte en 4 piezas de 2 dígitos
  12 + 34 + 56 + 78 = 180
  hash(x=12345678) = 80

  La cantidad de piezas (y su cantidad de dígitos) depende del tamaño *N* de la tabla.

Tablas de Dispersión

- Colisiones

  Ocurren cuando la función *hash* mapea dos claves diferentes a una misma ubicación en la tabla.

  ¿Cómo resolverlas?

  - Partición con encadenamiento:
  Listas en cada posición de la tabla.

  - Partición con direccionamiento abierto:
  Al ocurrir la colisión se busca una siguiente posición libre.

Tablas de Dispersión

- Partición con encadenamiento

  Almacena los datos en un arreglo de listas de tamaño *N*, con *n* la cantidad total de elementos en las listas.

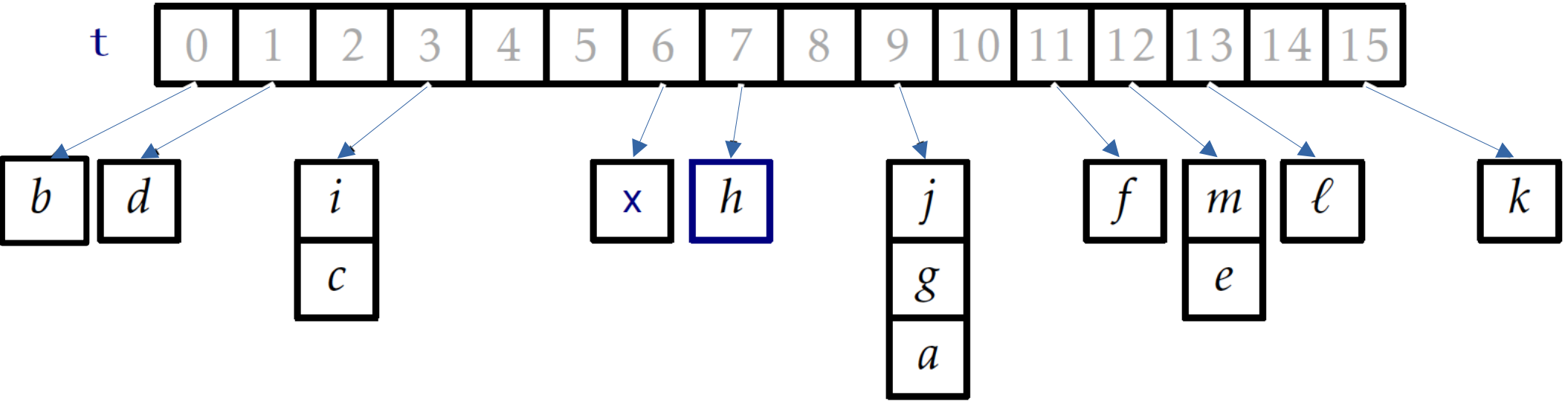
  $$0 \leq hash(x) \leq N-1$$

  Todos los elementos con *hash*(x)=*i* se almacenan en la lista en la posición *i* del arreglo.

  Para asegurar listas cortas, se mantiene el invariante $n \leq N$

Tablas de Dispersión

- Partición con encadenamiento



n = 14
N = 16

hash(x) = 6
hash(a) = 9
hash(j) = 9

Tablas de Dispersión

- Partición con encadenamiento

  Búsqueda:

  - Buscar la lista a la cual pertenece el elemento *hash*(x) = ?

  - Búsqueda linear dentro de la lista

```
T find(Object x) {
  for (T y : t[hash(x)])
    if (y.equals(x))
      return y;
  return null;
}
```

Morin, Pat.; Open Data Structures : An Introduction, 2013.

Tablas de Dispersión

- Partición con encadenamiento

  Inserción:

  - Dato no puede estar duplicado

  - El tamaño de la lista necesita ser incrementado?
    Si, duplicar tamaño y reinsertar datos
    No, seguir

  - Usar la función *hash* para obtener el índice de la lista correspondiente

  - Agregar el valor a la lista

Tablas de Dispersión

- Partición con encadenamiento

  Inserción:

```
boolean add(T x) {
  if (find(x) != null)
    return false;
  if (n+1 > N)
    resize();
  t[hash(x)].add(x);
  n++;
  return true;
}
```

Tablas de Dispersión

- Partición con encadenamiento

  Eliminación:

  - Usar la función *hash* para obtener el índice de la lista que contiene el elemento

  - Recorrer la lista hasta encontrar el elemento para posteriormente eliminarlo

Tablas de Dispersión

- Partición con encadenamiento

  Eliminación:

```
T remove(T x) {
  Iterator<T> it = t[hash(x)].iterator();
  while (it.hasNext()) {
    T y = it.next();
    if (y.equals(x)) {
      it.remove();
      n--;
      return y;
    }
  }
  return null;
}
```

Tablas de Dispersión

- Partición con encadenamiento

  Eficiencia de este esquema depende fundamentalmente de… la función *hash*!

  - Una buena función repartirá los elementos de manera uniforme en las listas...

  - Una mala función enviará todos los elementos a la misma lista en el arreglo...

Tablas de Dispersión

- Partición con direccionamiento abierto

  Utiliza un arreglo de elementos, donde en cada posición sólo puede almacenarse un dato.

  Si al usar la función *hash* ya hay un elemento en la posición que arroja (i), se busca una posición cercana para hacer el almacenamiento (i+1 % N, i+2 % N, …).

  También se conoce como direccionamiento abierto con sondeo o prueba lineal.

Tablas de Dispersión

- Partición con direccionamiento abierto

  De esta forma, hay tres tipos de elementos almacenados en la tabla:

  - Datos: valores almacenados.

  - Valores nulos: en ubicaciones donde aún no se han almacenado datos.

  - Valores eliminados: en ubicaciones donde antes se almacenó un dato pero luego fue eliminado.

Tablas de Dispersión

- Partición con direccionamiento abierto

  N = tamaño de la lista o arreglo

  n = cantidad actual de datos

  q = cantidad de valores no nulos (tipos 1 y 3)

  Invariante de tamaño: $2\,q \leq N$

Tablas de Dispersión

- Partición con direccionamiento abierto

  Búsqueda:

  - Iniciar en la posición dada por la función *hash*

  - Verificar posición i, (i+1)%N, (i+2)%N, etc, hasta encontrar el dato o llegar a un nulo

Tablas de Dispersión

- Partición con direccionamiento abierto

  Búsqueda:

```
T find(T x) {
  int i = hash(x);
  while (t[i] != null) {
    if (t[i] != del && x.equals(t[i]))
      return t[i];
    i = (i == N) ? 0 : i+1;
  }
  return null;
}
```

Tablas de Dispersión

- Partición con direccionamiento abierto

  Inserción:

  - Dato no puede estar duplicado

  - Usar la función *hash* para obtener el índice

  - Verificar posición i, (i+1)%N, (i+2)%N, etc, hasta encontrar un nulo o un eliminado, y almacenar allí el dato

  - Incrementar n y q (si es necesario)

Tablas de Dispersión

- Partición con direccionamiento abierto

  Inserción:

```
boolean add(T x) {
  if (find(x) != null)  return false;
  if (2*(q+1) > N)  resize();
  int i = hash(x);
  while (t[i] != null && t[i] != del)
    i = (i == N-1) ? 0 : i+1;
  if (t[i] == null) q++;
  n++;
  t[i] = x;
  return true;
}
```

Tablas de Dispersión

- Partición con direccionamiento abierto

  Eliminación:

  - Usar la función *hash* para obtener el índice

  - Verificar posición i, (i+1)%N, (i+2)%N, etc, hasta encontrar el dato o un nulo

    - Si es el dato, poner eliminado en esa posición

    - Si es nulo, el dato no está en la tabla

Tablas de Dispersión

- Partición con direccionamiento abierto

  Eliminación:

```
T remove(T x) {
  int i = hash(x);
  while (t[i] != null) {
    T y = t[i];
    if (y != del && x.equals(y)) {
      t[i] = del;
      n--;
      if (8*n < N)  resize();
      return y;
    }
    i = (i == N-1) ? 0 : i+1;
  }
  return null;
}
```

Tablas de Dispersión

- Partición con direccionamiento abierto

  La prueba lineal puede generar agrupamientos, haciendo cada vez más difícil encontrar un espacio libre.

  Puede reducirse usando otras técnicas:

  - Prueba cuadrática

  - *Hashing* doble

Tablas de Dispersión

- Partición con direccionamiento abierto
  Prueba cuadrática

  Al encontrar la posición *hash*(x) ocupada, se calcula una nueva posición siguiendo:

  $(hash(x) + c_1 j + c_2 j^2) \% N$

  c1, c2: constantes, diferentes de 0
  j: número de prueba

Tablas de Dispersión

- Partición con direccionamiento abierto
  Prueba cuadrática

  - Elimina los agrupamientos.

  - Una secuencia de pruebas puede cubrir sólo una pequeña parte de la tabla, generando dificultades para encontrar una posición libre.

Tablas de Dispersión

- Partición con direccionamiento abierto
  Hashing doble

  Utilizar dos funciones *hash*, en vez de una sola.

  ($hash_1$(x) + j$hash_2$(x)) % N

  j: número de prueba

  Si la posición dada por $hash_1$ está ocupada, se busca una siguiente usando el resultado de $hash_2$.

Tablas de Dispersión

- Partición con direccionamiento abierto
  Hashing doble

  - No genera agrupamientos simples (prueba lineal).

  - No genera agrupamientos en pequeñas regiones de la tabla (prueba cuadrática).

  - Minimiza colisiones repetidas.

Tablas de Dispersión

- Implementación en C++

  `unordered_map` → `#include <unordered_map>`

  Los elementos no están ordenados por la clave, pero sí se organizan en compartimientos (*buckets*) de acuerdo al valor de su función *hash*

  Así el programa puede hacer el acceso individual más rápidamente.

Tablas de Dispersión

```
struct ltstr {
  bool operator() (const char* s1, const char* s2) const {
    return strcmp(s1, s2) < 0;
  }
};

int main () {
  map<const char*, int, ltstr> months;

  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;
```

Tablas de Dispersión

```
  cout << "Accesing one element:" << endl;
  cout << "june -> " << months["june"] << endl << endl;

  cout << "Map size: " << months.size() << endl;

  cout << "Ordered elements in map:" << endl;
  map<const char*, int, ltstr>::iterator cur;
  for (cur = months.begin(); cur != months.end(); cur++;)
    cout << (*cur).first << endl;
  cout << endl;
}
```

Tablas de Dispersión

```
int main () {
  typedef unordered_map<const char*, int> unordmap;
  unordmap months;

  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;
```

Tablas de Dispersión

```
  cout << "Accesing one element:" << endl;
  cout << "june -> " << months["june"] << endl << endl;

  unordmap::hasher hashfunc = months.hash_function();

  cout << "Map size: " << months.size() << endl;

  cout << "Map buckets: " << months.bucket_count() << endl;

  cout << "Hashed elements in map:" << endl;
  unordmap::iterator cur = months.begin();
  while (cur != months.end()) {
    cout << hashfunc((*cur).first) << " | ";
    cout << months.bucket((*cur).first) << " -> ";
    cout << (*cur).first << endl;
    cur++;
  }
  cout << endl;
}
```

https://replit.com/@Andrea_del_Pila/EDD-UsandoUnorderedMap

Tablas de Dispersión

- `unordered_map`

  Información básica de uso en:
  https://linuxhint.com/use_c_unordered_map/

Referencias

- https://es.wikipedia.org/wiki/Tabla_hash
- https://en.wikipedia.org/wiki/Hash_table
- Algorithmic Thinking: A Problem-Based Introduction, Daniel Zingaro, No Starch Press, 2020
- Data Structures and Program Design Using C++, D. Malhotra, PhD, N. Malhotra, PhD, Stylus Publishing, LLC, 2019
- Open Data Structures: An Introduction, Pat Morin, Athabasca University Press, 2013