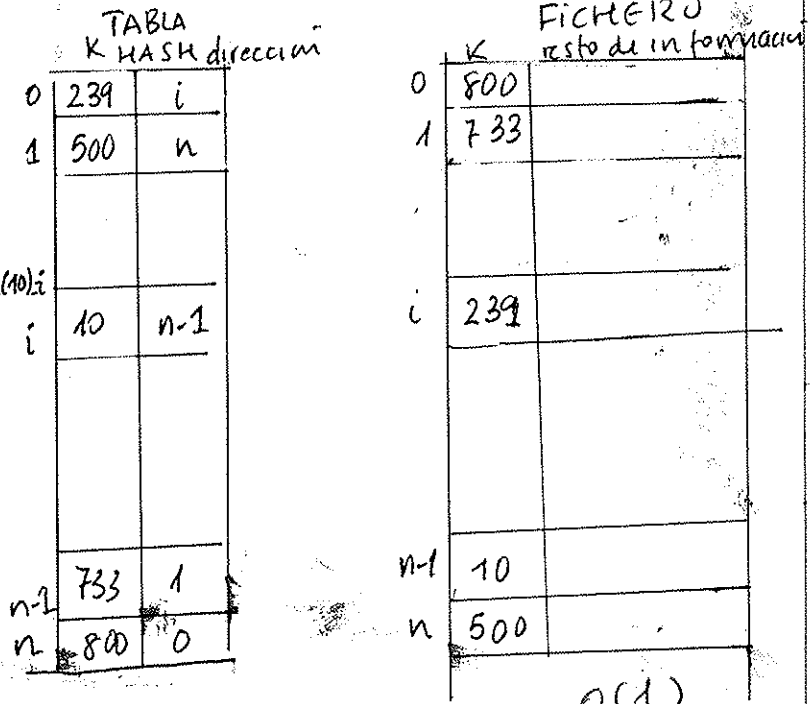
LECCIÓN 25: TABLAS HASH

OBJETIVO:

Dados un conjunto de datos identificados por una clave (KK) queremos obtener una función h(K) (función hash) que nos de la posición dentro de una tabla (tabla hash) en la que almacenamos un par clave (K) y la dirección de un fichero donde se encuentra la información asociada a esa clave.



OBjetivo: Realizar búsquedas en $O(1)$

PROBLEMA con las funciones Hash

Pueden producir COLISIONES

COLISIÓN: ocurre cuando dos claves K1 y K tienen asociadas el mismo valor $h(K_1)=h(K)$

. HAY QUE PENSAR QUE FUNCIÓN Produce MENOS COLISIONES.

CARACTERISTICAS de las FUNCIONES HASH

- Que sea rápida de calcular
- Que no produzca colisiones.

. Ver ejemplo transparencia.

- El espacio ocupado por la tabla hash debe ser asumible.

- DEBEMOS BUSCAR UNA ESTRATEGIA HASH QUE:
  1. El tiempo para resolver las colisiones no sea alto
  2. El tamaño de la tabla Hash no sea muy grande.

Ejemplo: FICHERO de DATOS

K			Dir en el fichero
12	Abad	Ruiz	0
21	Bernabe	Perez	1
68	Carrasco	Ruiz	2
38	Domingo	Cocer	3
52	Edet	Sanchez	4
70	Juan	Ruiz	5
44	Martin	Pérez	6
18	Perez	Catiano	7

- **Funciones hash.**

  Una función hash debería cumplir dos propiedades:

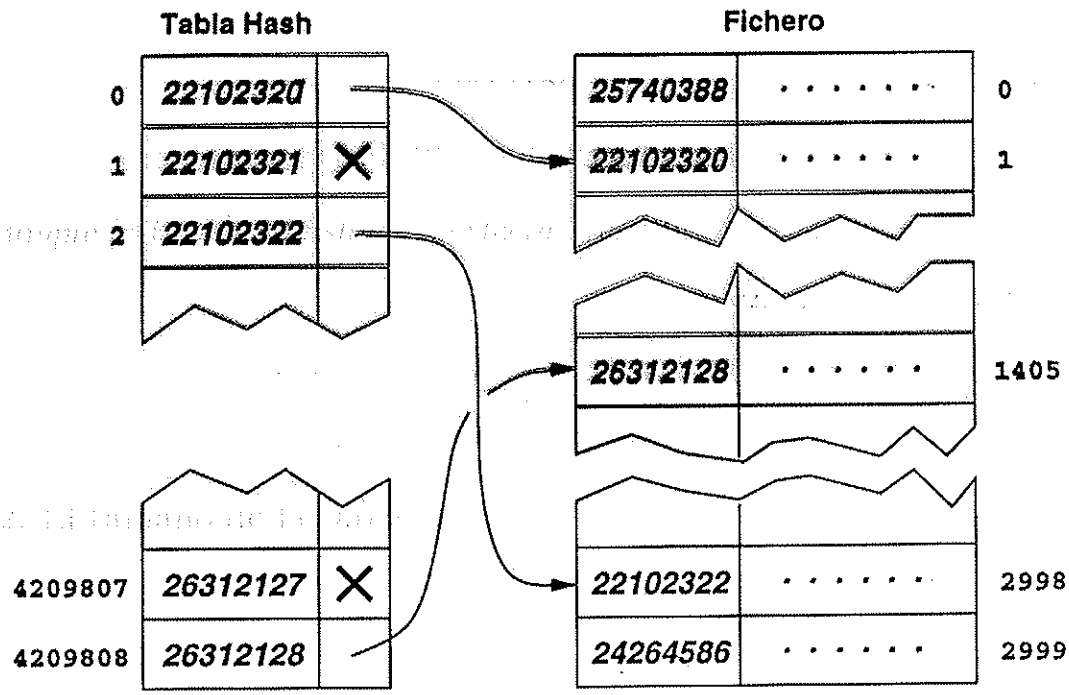
  1. Que sea fácil (rápida) de calcular.
  2. Que no produzca colisiones.

  Sin embargo ....

  – Fichero de alumnos de una Facultad (3000 alumnos).
  – El orden en que se encuentran es irrelevante.
  – Clave primaria: DNI (22102320 a 26312128).

$$h(k) = k - 22102320$$

La función h es rápida y no produce colisiones.



  – La tabla hash contiene 26312128 – 22102320 + 1 = 4209809 entradas.
  – Sólo 3000 (el 0.07 %) direccionan al fichero de datos.

Aunque la función hash es "perfecta" resulta imposible asumir esta solución por problemas de espacio.

Resulta preferible otra solución en la que el tamaño de la tabla sea menor (aunque genere colisiones) en la que se tenga en cuenta el equilibrio entre:

1. El tiempo de resolución de las colisiones
2. El tamaño de la tabla Hash

Fichero de Datos 2

12	Abad Ruiz	0
21	Bernabe Perez	1
68	Carrasco Ruiz	2
32	Duarte Lopez	3
56	Garcia Pi	4
77	Lopez Lopez	5
91	Perez Martín	6
18	Ruperez Galiano	7

LECCIÓN 25: TABLAS HASH

En este ejemplo tenemos 8 registros. Suponed que disponemos de una tabla hash con $11 = M$ posiciones y la función hash que vamos a usar es $h(K) = K \% 11$

TABLA HASH

h(K)	K	posición en el fichero di
0	44	6
1	12	0
2	68	2
3		
4	70	5
5	38	3
6		
7	18	7
8	52	4
9		
10	21	1

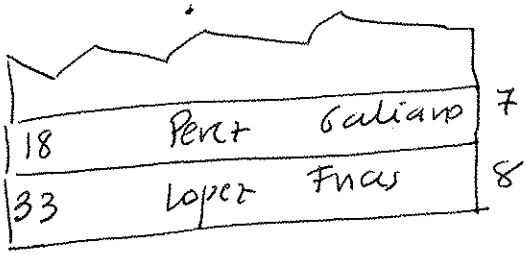
Como llevar a cabo las consultas

1) Suponed que queremos consultar la información con clave K=52 para obtener toda la información asociada a esta clave. Pasos a seguir:
   1) Obtener h(52) que es 8
   2) Accedemos a la posición 8 de la tabla hash y consultamos posición en el fichero di que es 4.
   3) Vamos a la posición 4 en el fichero de datos y obtenemos
      52 Fdez Sanchez.

2) Suponed que queremos consultar K = 14. Siguiendo los pasos anteriores calculamos h(14)=3. Vamos a la posición 3 de la tabla y vemos que está vacía luego el registro no existe.

3) Suponed que queremos añadir el registro
   33 Lopez Frias

1) Lo añadimos al fichero de datos en la posición 8



2) Debemos añadir una entrada en la tabla hash por este registro
   1) calculamos h(33) = 0

Pero la posición 0 ya está ocupada → COLISIÓN

¿Cómo resolver las colisiones?

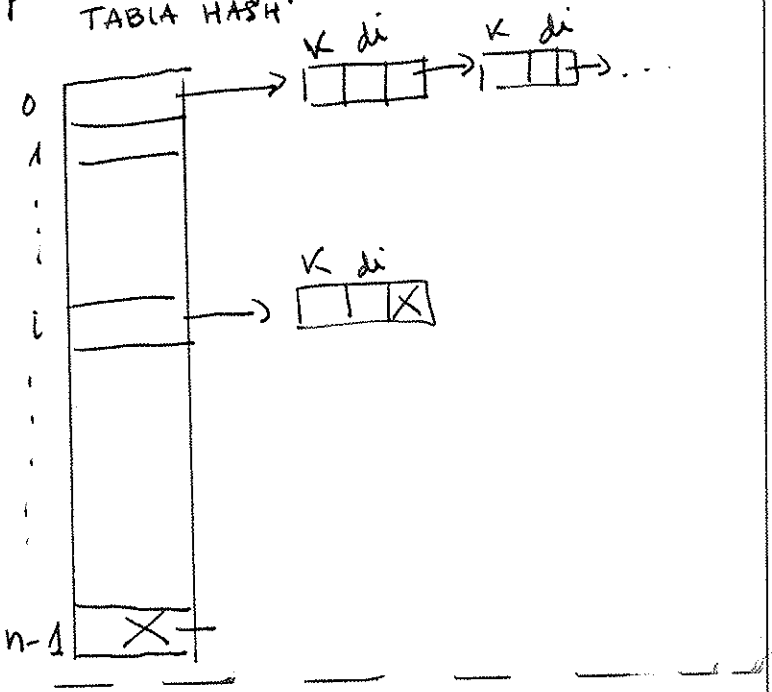
- HASHING ABIERTO
- HASHING CERRADO

LECCIÓN 25: TABLAS HASH

HASHING ABIERTO

El método de resolución de colisiones por hashing abierto consiste en asociar a cada entrada h(K) una lista donde se incluyen todas las claves con igual valor h(K). Así la lista sería un conjunto de celdas enlazadas compuesto por clave K y posición en el fichero

TABLA HASH

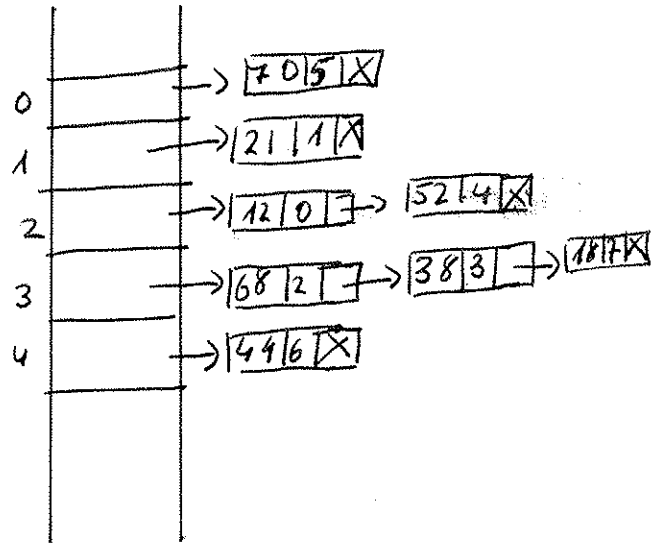


Fichero de DATOS.

Ejemplo

K		di
12	Abad Ruiz	0
21	Bernabe Perez	1
68	Carrasco Ruiz	2
38	Domingo Coca	3
52	Fernandez Sanchez	4
70	Juarez Ruiz	5
44	Martin Perez	6
18	Ruperez Galiano	7

Funcion hash $h(K) = K \% 5$ $H=5$

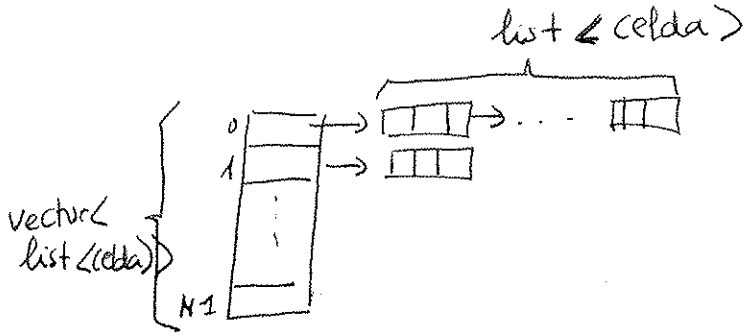


$h(12) = 12 \% 5 = 2$
$h(21) = 21 \% 5 = 1$
$h(68) = 68 \% 5 = 3$
$h(38) = 38 \% 5 = 3$ ← (colisión)
$h(52) = 52 \% 5 = 2$ ← (" ")
$h(70) = 70 \% 5 = 0$
$h(44) = 44 \% 5 = 4$
$h(18) = 18 \% 5 = 3$

VENTAJAS: El tamaño de la tabla hash puede ser mayor que el nº de claves.

INCONVENIENTES → Gestión de punteros.

TABLAS HASH
LECCION 25: HASH ABIERTO



Implementación

```
// hash-abierto.h
#include <list>
#include <vector>

class celda{
  private:
    int k; // valor de clave
    int di; // direccion en el fichero de datos con la informacion
            // de la clave.
  public:
    celda() : k(-1), di(-1) {}
    celda(int c, int p): k(c), di(p) {}
    int & clave() { return k; }
    int & Posicion() { return di; }
};
class TablaHash {
  private:
    vector<list<celda> > tabla;
    int fhash (int clave){
        return clave % tabla.size(); // h(K) = K % M
    }
};

pair<bool, list<celda>::iterator> Esta(int clave){
    list<celda>::iterator it;
    int pos = fhash(clave);
    for (it = tabla[pos].begin(); it != tabla[pos].end()
        && (*it).clave() != clave; ++it);
    bool find = false;
    if (it != tabla[pos].end()) find = true;
    return pair<bool, list<celda>>(find, it);
}
}
```

LECCION 25: TABLAS HASH. HASH ABIERTO

```
public:
    TablaHash(int tam){
            assert(tam>0);
            tabla.resize(tam);
    }
    bool Existe(int clave){
pair<bool,list<celda>::iterator> a;
            a=Esta(clave);
            return a.first;
    }
    bool Insertar(int clave, int di){
            pair<bool,list<celda>::iterator> a;
            a=Esta(clave);
            if(a.first==false){
                int pos=fhash(clave);
                tabla[pos].push_back(celda(clave,di));
                return true;
            }
            return false;
    }
bool CambiarDir(int clave, int nuevadi){
    pair<bool,list<celda>::iterator> a;
    a=Esta(clave);
    if(a.first==true){
        (*(a.second)).Posicion()=nuevadi;
        return true;
    }else return false;
}
```

LECCION 25: TABLAS HASH. HASH ABIERTO

```
int   ObtenerDir ( int clave) {
    pair <bool, list <celda>::iterator> a = Esta ( clave);
    if (a.first){
    return(*(a.second)). Posicion();

    }
    else return -1;
}
bool  Borrar(int  clave) {
   pair <bool, list <celda>:: iterator > a = Esta (clave);
    if (a.first ) {
        int pos = fhash(clave);
        tabla[pos].erase( a.second);
        return true;
    }
    return false;
}
friend ostream & operator<< (ostream &os, const TablaHash & T) {
      vector <list <celda> >:: iterator it1; int pos=0;
    for (it1 = T.tabla.begin(); it1 != T.tabla.end(); ++it1, ++pos
    {   os << "Datos en posicion " << pos << ':';
        list <celda> ::iterator itl;
        for (itl = (*it1).begin(); itl != (*it1).end(); ++itl){
            os<<(*itl).clave() << ' ' << (*itl).Posicion()
              << ' ';
        }
        os <<endl;
    }
    return os;
}

};
```