Organizaciones clásicas de ficheros

Archivos Relativos (hashing)

Introducción

- Fundamento:
  - Los registros son de tamaño constante.
  - Podremos posicionarnos de forma aleatoria en un registro.
  - Aplicar a la clave un algoritmo de dispersión (función hash).
  - La función hash proporciona como salida una dirección de registro o de bloque en el que se almacenará el registro.
  - Idealmente:
    - O bien no existirán dos registros que generen la misma dirección.
    - O bien siempre habrá espacio suficiente para buscar otra posición.

Introducción

- Tipos de hashing:
  - Estático: se conoce a priori el número máximo de registros a almacenar:
    - Cerrado:
      - Solo existe un único bloque (tabla).
      - La función hash obtiene el desplazamiento interior.
      - Para almacenamiento interno.
    - Abierto:
      - El número de bloques (cubos) disponibles es mayor que 1.
      - La función de hash proporciona un dirección de bloque.
      - Para almacenamiento externo.
  - Dinámico: No se sabe a priori el número de registros a almacenar.
    - Se utiliza en cada momento el tamaño de espacio menor posible para el número de registros almacenados.

Introducción

- Función de hashing:
  - Convertir la clave a un dominio numérico.
  - Mapear el valor numérico de clave al rango posible de posiciones/bloques reservados.
  - Colisión: la función hash proporciona la misma dirección de bloque (cubo) para dos claves distintas.


```
long hash(const char *key, int len)
{
   long v = 1;
   for (int c=0; c<len; ++c)
      v = (v * key[c]) % K;
   return v % SPACE_SIZE;
}
```

Introducción

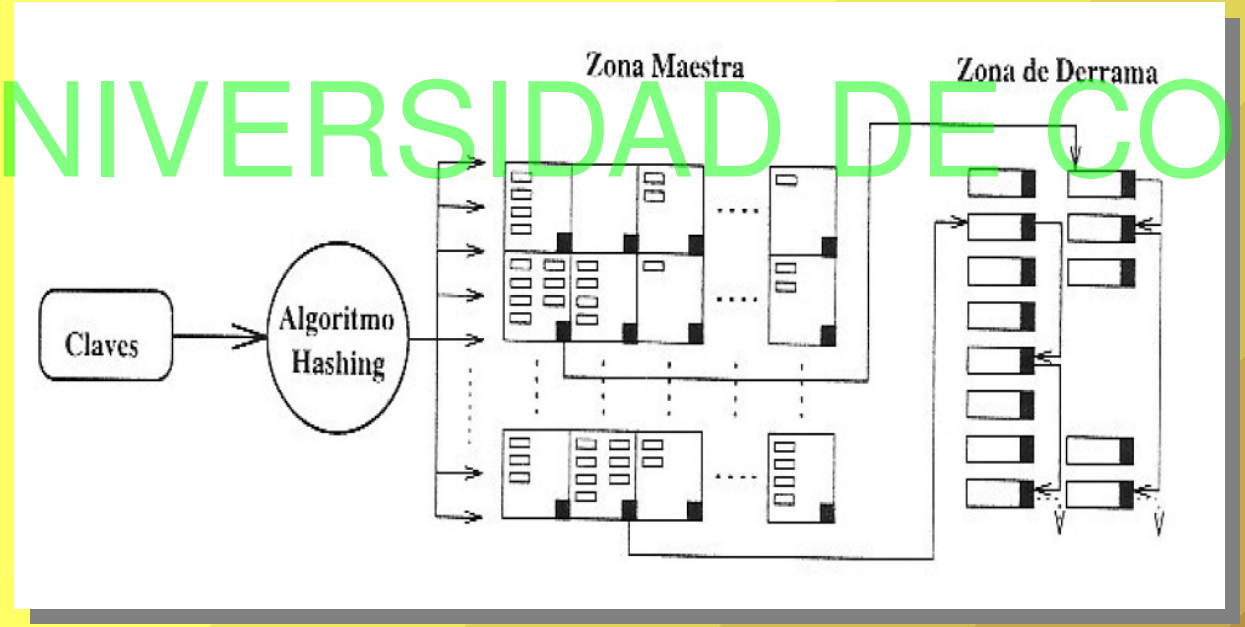
- Cubo:
  - Unidades de espacio de direccionamiento a disposición de la función hash.
  - Num. reg. por cubo $F_c = C/R$ (C=tam. cubo, R=Tam. registro).
  - Núm. de cubos necesarios $c = r/F_c$ (r=núm registros).
  - Densidad de empaquetamiento: $d = r/(F_c * c)$.
  - Observaciones:
    - Los registros dentro de un cubo se almacenan sin orden.
    - Cuando se direcciona un cubo y este está lleno → Desborde.
    - Tam. cubo pequeño -> muchas colisiones.
    - Tam. cubo grande -> búsqueda/proc. del reg. dentro de cubo costosa.
    - A mayor densidad, mayor probabilidad de colisión.

Introducción

- Tratamiento de colisiones.
  - Desborde: la colisiones de claves provocan que el cubo se llene de registros y se provoque un “desborde”.
  - Tratamiento de los desbordes:
    - Direccionamiento abierto: En zona maestra, buscar en secuencia hasta encontrar la primera entrada (siguiente cubo maestro) libre.
    - Utilizar cubos de desborde: distribuir uniformemente cubos para almacenar los desbordes de un conjunto de cubos maestros.
    - Encadenado: crear una cadena para esa dirección en una zona de derrama separada.
    - Hashing múltiple: usar una segunda, tercera, … función de hash. Si continua existiendo colisión aplicar Direccionamiento Abierto.

Ejemplo

- Hashing estático abierto con z. de derrama encadenada.



Operaciones

- Inserción O(1).
  - Si hay colisión se aplica la resolución de colisiones.
  - Podría llegar hasta O(N) en caso extremo.
- Localización.
  - Usando clave= O(1) y posible resolución de colisión.
  - No se usa clave = O(N)
- Lectura secuencial O(N) (no hay orden)
- Lectura exhaustiva O(N).
- Lectura ordenada $O(N^2)$.
- Actualización:
  - Sin modificar clave O(1).
  - Modificando clave: marcar como borrado O(1) + nueva inserción O(1).
- Borrado O(1): marcar como borrado.
- Reorganización.
  - Para eliminar la z. de derrama.
  - Para dimensionar el número de cubos para disminuir la densidad de empaquetado.

Resumen

- Ventajas:
  - Inserciones/localizaciones rápidas.
  - Estructura simple.
- Inconvenientes:
  - Gran coste para operaciones en orden.
  - Desperdicio de espacio en variantes estáticas.

Referencias

- Luque Ruiz I. y otros, “Ficheros. Organizaciones clásicas para el almacenamiento del a información”, U. de Córdoba, 1998.