**Hash-Based Index**

Profesor Heider Sanchez

Integrantes:

* Wanly Obregón
* Milagros Yupanqui

**P1. Static Hash – Numero Primo:**

Una estrategia para disminuir la probabilidad del overflow page es usando un numero primo de buckets () que cumpla lo siguiente:

Función hash:

En donde: **,** es el numero de registros, es el factor de bloque y es el factor de corrección que significa el espacio libre en el archivo.

Por ejemplo: si ; entonces . Por lo que el siguiente número primo seria .

Muestre el estado final del archivo tanto para M=6 y M=7. Verifique si se reduce la cantidad de colisiones.

// -------------------------------------------------- SOLUCION -------------------------------------------------- \\

|  |  |  |
| --- | --- | --- |
| **0** | **18,36, 60** |  |
| **1** | **7, 19, 31, 37,** | **1** |
| **2** | **2,** |  |
| **3** | **3,** |  |
| **4** | **28, 36, 46 ,** |  |
| **5** | **5, 11, 17, 23** | **29,41, 53** |

|  |  |  |
| --- | --- | --- |
| **0** | **7, 28,** |  |
| **1** | **29, 36,** |  |
| **2** | **2, 23, 37,** |  |
| **3** | **3, 17,31,** |  |
| **4** | **11, 18,46,53** | **60** |
| **5** | **5, 19, 40, 61** |  |
| **6** | **41,** |  |

M=6 M=7

Keys

2,3,5

7,11,17

18,19,23

28,29,31

36,37,40

41,46,53

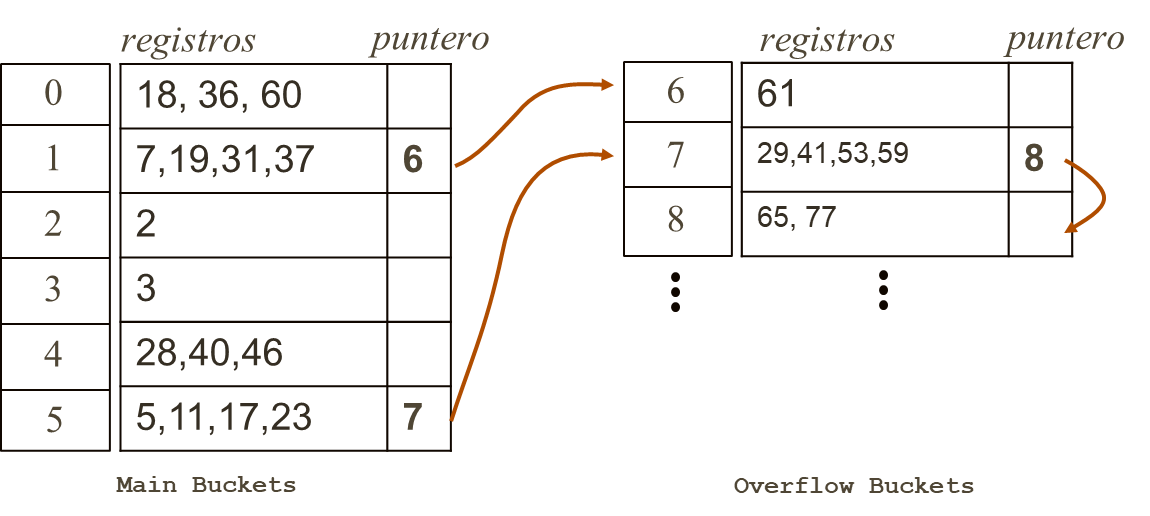
60,61

* **“Efectivamente usar el número primo si reduce la cantidad de colisiones”**

// -------------------------------------------------- FIN -------------------------------------------------- \\

**P2. Desbordamiento encadenado.**

Una alternativa para el control de colisiones de páginas en el archivo es el desbordamiento encadenado, el cual consiste en mantener un puntero al siguiente bucket colisionado, similar a como se muestra en la siguiente imagen (M = 6):



Diseñe la estructura del bucket en C++ con soporte al desbordamiento encadenado. Luego diseñe los siguientes algoritmos:

#include <iostream>

struct Bucket{

    Record records[4];

    long next;

    int count = 0;

}

//Algoritmo de Inserción

void insert (Record record){

    file open (“data.dat”);

    key\_hash = hash\_code(record.data);

    seek(key\_hash \* sizeof(Bucket));

    Bucket bucket;

    read ((char\*)&bucket, sizeof(Bucket));

    if (bucket.count < 4){

        bucket.records [bucket.count] = record;

        bucket.count++;

        seek(key\_hash \* sizeof(Bucket));

        write ((char\*)&bucket, sizeof(Bucket));

    }

    else{  //Recorrido por overflow

        while (bucket.next!=-1){

            pos\_last\_bucket = bucket.next;

            seek (bucket.next);

            read ((char\*)&bucket, sizeof(Bucket));

        }

        if (bucket.count < 4){

            bucket.records [bucket.count] = record;

            bucket.count++;

            seek(pos\_last\_bucket);

            write ((char\*)&bucket, sizeof(Bucket));

        }

        else {

            Bucket NewBucket

            bucket.next = file.tellg();

            NewBucket.records [NewBucket.count] = record;

            NewBucket.count++;

            NewBucket.next = -1;

            write ((char\*)&NewBucket, sizeof(Bucket));

        }

    }

}

//Algortimo de Búsqueda

Record search(Record record){

    open ("data.dat");

    key\_hash = hash\_code(record.data);

    seek(key\_hash \* sizeof(Bucket))

    Bucket bucket;

    read ((char\*)&bucket, sizeof(Bucket));

    bool flag = 0;

    for (auto &v : bucket.records){

        if (v == record.data) {

            flag = 1;

            return v;

        }

    }

    //Recorrido por el overflow

    while (bucket.next!=-1){

        seek (bucket.next);

        read ((char\*)&bucket, sizeof(Bucket));

        for (auto &v : bucket.records){

            if (v = record.data) {

                flag = 1;

                return v;

            }

        }

    }

    if (flag == 0){

        throw ("El registro no existe");

    }

}

**P3. Hash Dinámico (Idea de construcción 1):**

Con el fin de optimizar el espacio que ocupa el Hash Index sin crear entradas innecesarias, es haciendo lo siguiente:

* La profundidad global siempre se mantiene fija como un parámetro de capacidad del sistema (e.g. D=8, D=16, D=32).
* Empezar el hash index con dos entradas 0 y 1.
* Para insertar un nuevo registro se debe hacer una búsqueda “secuencial” en el hash index para localizar la entrada que coincide con el sufijo de menor tamaño del key.
* Cuando se desborde un bucket hacer el Split, y agregar al índice la nueva entrada referenciando al siguiente bucket creado.

Aplicar esta idea a la siguiente secuencia de registros a insertar:

Keys

23, 2, 4, 5, 9, 35, 8,13, 16, 1, 20, 65, 40, 73, 74

Considere como profundidad global de tres bits y un factor de bloque de 3 registros . Ilustre paso a paso el proceso de spliting, es decir, cada vez que se genera un nuevo bucket.

|  |  |  |
| --- | --- | --- |
| key | HASH | BIN |
| 23 | 7 | 111 |
| 2 | 2 | 010 |
| 4 | 4 | 100 |
| 5 | 5 | 101 |
| 8 | 0 | 000 |
| 9 | 1 | 001 |
| 35 | 3 | 011 |
| 13 | 5 | 101 |
| 16 | 0 | 000 |
| 1 | 1 | 001 |
| 20 | 4 | 100 |
| 65 | 1 | 001 |
| 40 | 0 | 000 |
| 73 | 1 | 001 |
| 74 | 2 | 010 |

// -------------------------------------------------- SOLUCION -------------------------------------------------- \\

Paso 1:

|  |  |
| --- | --- |
| Index.dat | Data.dat |
| |  |  | | --- | --- | | 0 | 0 | | 1 | 1 | | |  |  |  | | --- | --- | --- | | 0 | 2, 4, | -1 |      |  |  |  | | --- | --- | --- | | 1 | 23, 5, 9, 35 | -1 | |

Hay colisión al insertar el 35 por lo tanto hacemos Split

Paso 2:

|  |  |
| --- | --- |
| Index.dat | Data.dat |
| |  |  | | --- | --- | | 0 | 0 | | 01 | 1 | | 11 | 2 | | |  |  |  | | --- | --- | --- | | 0 | 2, 4, 8, 16 | -1 |      |  |  |  | | --- | --- | --- | | 1 | 5, 9, 13, | -1 |  |  |  |  | | --- | --- | --- | | 2 | 23, 35, | -1 | |

Hay colisión al insertar el 16 por lo tanto hacemos Split

Paso 3:

|  |  |
| --- | --- |
| Index.dat | Data.dat |
| |  |  | | --- | --- | | 00 | 0 | | 01 | 1 | | 11 | 2 | | 10 | 3 | | |  |  |  | | --- | --- | --- | | 0 | 4, 8, 16, | -1 |      |  |  |  | | --- | --- | --- | | 1 | 5, 9, 13, 1 | -1 |  |  |  |  | | --- | --- | --- | | 2 | 23, 35, | -1 |  |  |  |  | | --- | --- | --- | | 3 | 2, | -1 | |

Hay colisión al insertar el 1 por lo tanto hacemos Split

Paso 4:

|  |  |
| --- | --- |
| Index.dat | Data.dat |
| |  |  | | --- | --- | | 00 | 0 | | 001 | 1 | | 11 | 2 | | 10 | 3 | | 101 | 4 | | |  |  |  | | --- | --- | --- | | 0 | 4, 8, 16, 20 | -1 |      |  |  |  | | --- | --- | --- | | 1 | 9, 1 | -1 |  |  |  |  | | --- | --- | --- | | 2 | 23, 35, | -1 |  |  |  |  | | --- | --- | --- | | 3 | 2, | -1 |  |  |  |  | | --- | --- | --- | | 4 | 5, 13, | -1 | |

Hay colisión al insertar el 20 por lo tanto hacemos Split

Paso 5:

|  |  |
| --- | --- |
| Index.dat | Data.dat |
| |  |  | | --- | --- | | 000 | 0 | | 001 | 1 | | 11 | 2 | | 10 | 3 | | 101 | 4 | | 100 | 5 | | |  |  |  | | --- | --- | --- | | 0 | 8, 16, 40, | -1 |      |  |  |  | | --- | --- | --- | | 1 | 9, 1, 65, 73 | -1 |  |  |  |  | | --- | --- | --- | | 2 | 23, 35, | -1 |  |  |  |  | | --- | --- | --- | | 3 | 2, | -1 |  |  |  |  | | --- | --- | --- | | 4 | 5, 13, | -1 |  |  |  |  | | --- | --- | --- | | 5 | 4, 20, | -1 | |

Hay colisión al insertar el 73, ya no se puede puede hacer split por que se supera el máximo de profundidad, en este caso se realiza el enlazamiento. Para el enlace “para este caso estamos pasando todos los antiguos valores al nuevo bucket, y el actual valor insertado se queda en el bucket anterior”

Paso 6:

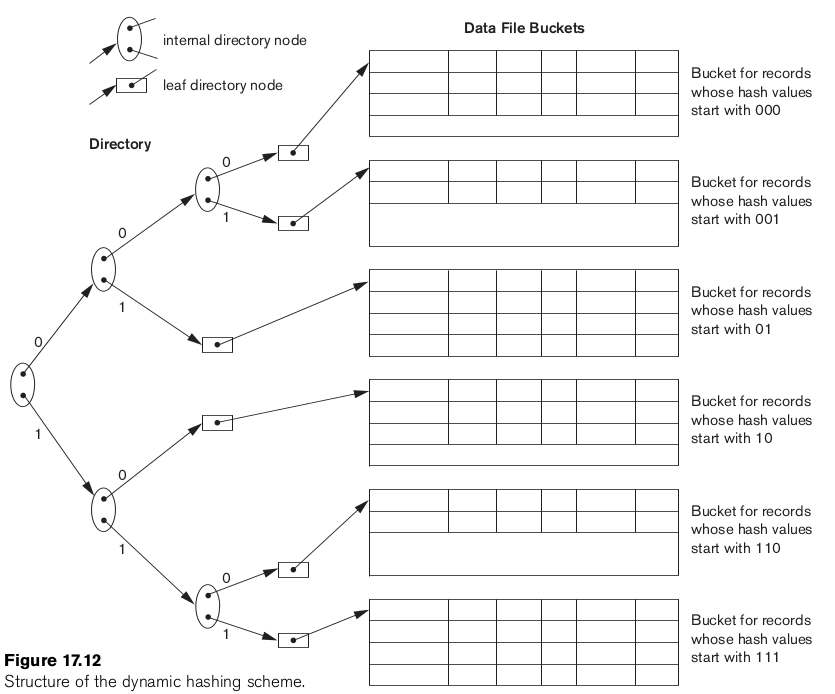
|  |  |
| --- | --- |
| Index.dat | Data.dat |
| |  |  | | --- | --- | | 000 | 0 | | 001 | 1 | | 11 | 2 | | 10 | 3 | | 101 | 4 | | 100 | 5 | | |  |  |  | | --- | --- | --- | | 0 | 8, 16, 40, | -1 |      |  |  |  | | --- | --- | --- | | 1 | 73 | 6 |  |  |  |  | | --- | --- | --- | | 2 | 23, 35, | -1 |  |  |  |  | | --- | --- | --- | | 3 | 2, 74 | -1 |  |  |  |  | | --- | --- | --- | | 4 | 5, 13, | -1 |  |  |  |  | | --- | --- | --- | | 5 | 4, 20, | -1 |  |  |  |  | | --- | --- | --- | | 6 | 9, 1, 65, | -1 | |

// -------------------------------------------------- FIN -------------------------------------------------- \\

**P3. Hash Dinámico (Idea de construcción 2):**

Resolver el caso anterior con un árbol digital: mantener el directorio en forma de árbol binario en donde los nodos hojas guardan los punteros a los buckets en el data file.

* El algoritmo inicia con dos buckets (0 y 1),
* Para ubicar un registro en su respectivo bucket se toma los d-primeros dígitos del hash (d hace referencia al nivel del árbol).
* Cuando un bucket se desborda, se procede a crear dos nodos hijos incrementando el nivel de bits (01 010,011).



// -------------------------------------------------- SOLUCION -------------------------------------------------- \\

