UTN-FRBA

GESTION DE DATOS

CREACION DE INDICES

DIRECTOR CATEDRA: ING. ENRIQUE REINOSA

INDICES - OBJETIVO

- Objetivo

El objetivo es crear una estructura adicional a la tabla que permita mantenerlos los datos ordenados en función de alguna clave.

Por ejemplo la creación de una PRIMARY KEY o un índice UNIQUE

INDICES - CONCEPTO

- Concepto

Dependiendo la arquitectura de hardware utilizada, el índice puede formar parte de la tabla o ser una estructura adicional a la tabla sobre la cual se crea, lo que genera un espacio adicional y la necesidad de incorporar y eliminar valores en ambos sitios.

Para el caso de la Arquitectura de PC (8086) el índice es una estructura adicional a la tabla.

INDICES - ALMACENAMIENTO

- Esquema de Almacenamiento

Cuando se ingresan filas en una tabla por ejemplo estas filas:

14, CARLOS, ….

2, PEDRO, …..

5, JOSE, ….

24, MARIA,…..

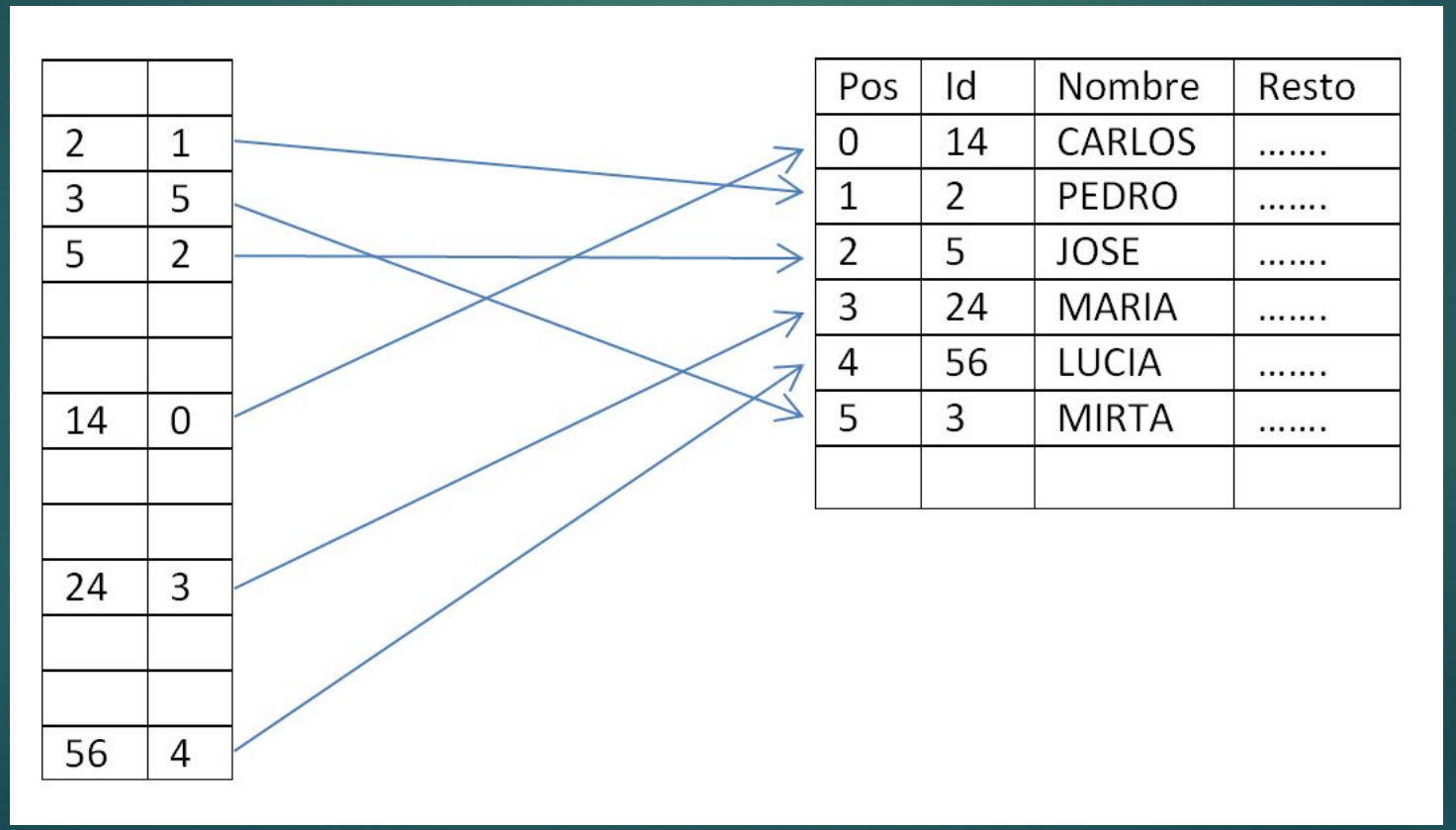
56, LUCIA, …..

3, MIRTA,…..

Pos	Id	Nombre	Resto
0	14	CARLOS	…….
1	2	PEDRO	…….
2	5	JOSE	…….
3	24	MARIA	…….
4	56	LUCIA	…….
5	3	MIRTA	…….

INDICES - ALMACENAMIENTO

- Esquema



2	1
3	5
5	2
14	0
24	3
56	4

Pos	Id	Nombre	Resto
0	14	CARLOS	.....
1	2	PEDRO	.....
2	5	JOSE	.....
3	24	MARIA	.....
4	56	LUCIA	.....
5	3	MIRTA	.....

INDICES - ACCESOS

- **Tipos de Acceso**: existen diferentes formas de acceder a los datos.
  - **Secuencial**: el acceso se realiza en función al modo en que ingresaron los datos.
  - **Secuencial Indexado**: el acceso se realiza en función de alguna clave que fue definida.
  - **Directo o Random**: el acceso es en forma directa a una clave sin realizar ningún recorrido

INDICES - METODOS

- En la práctica existen dos métodos para crear índices en un DBMS (Data Base Managment System).
  - Hashing
  - Árbol B (Btree)
- En la arquitectura 8086 el método utilizado es el de Arbol B y en la arquitectura de Mainframe y Minicomputadoras el método utilizado es Hashing

INDICES – HASHING - CONCEPTO

El método de Hashing trabaja sobre el concepto de una tabla y una función hash, también llamada función de dispersión, dicha función se utiliza para convertir algún tipo de dato en un pequeño número que puede servir como huella digital de ese dato. El término hash proviene, aparentemente, de la analogía con el significado en inglés de dicha palabra en el mundo real: picar y mezclar. El algoritmo de hash "pica y mezcla" los datos para crear las huellas digitales. Estas son llamadas valores o códigos hash, los cuales pueden ser utilizados como índices en tablas hash o bien como controles de integridad de datos o archivos.

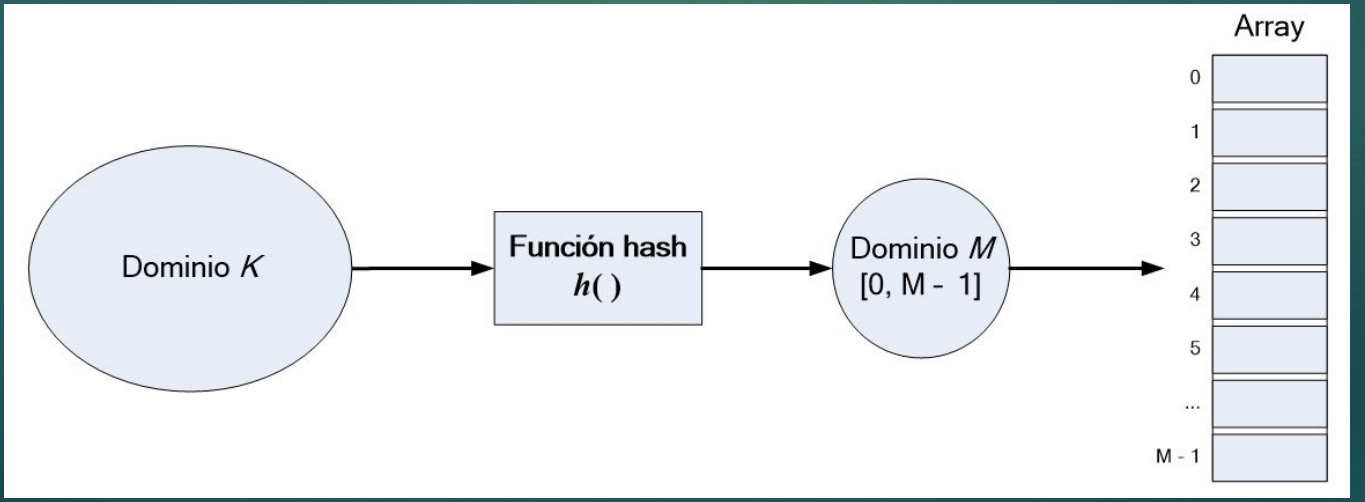
INDICES – HASHING – FUNCION HASH

Se considera que una buena función de hash es aquella que tiene las siguientes cualidades:

- **Evitar colisiones**: se cumple cuando dado un conjunto de valores de entrada, los resultados obtenidos en el conjunto de salida son distintos
- **Tiende a distribuir las claves uniformemente**: indica que tiende a distribuir uniformemente los valores de salolida respecto a los valores de entrada
- **Es fácil de calcular**: No necesariamente significa que sea fácil de escribir el algoritmo para calcular la función, sino que significa que el tiempo de ejecución de la función de hash debe ser ***O(1).***

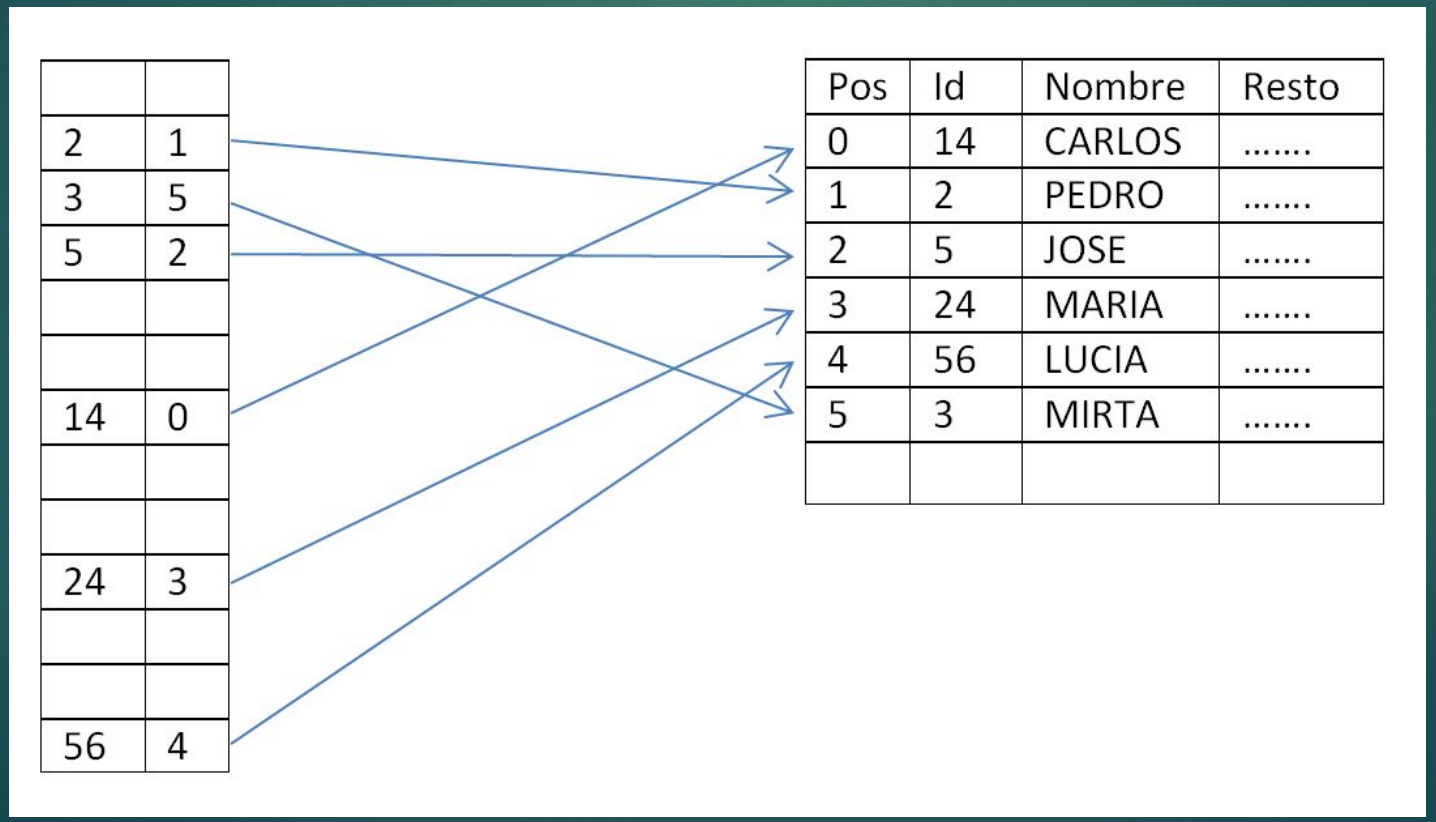
INDICES – HASHING – METODO

- El método crea una tabla (vector de dos dimensiones) donde en la primer dimensión colocará las claves y en la segunda dimensión las posiciones relativas de las mismas en la tabla donde se encuentran los datos correspondientes a esa clave.
- La función hash() recibe como entrada la clave a almacenar y devuelve un valor numérico entero que corresponde a la posición en la cual debería ir dicha clave en la tabla mencionada.



INDICES – HASHING – METODO

- El método crea una estructura adicional donde mantiene la clave ordenada y la posición relativa a los datos:



2	1
3	5
5	2
14	0
24	3
56	4

Pos	Id	Nombre	Resto
0	14	CARLOS	.....
1	2	PEDRO	.....
2	5	JOSE	.....
3	24	MARIA	.....
4	56	LUCIA	.....
5	3	MIRTA	.....

INDICES – HASHING – COLISIONES

Si ocurriese que una clave generara un valor de hash que apuntase a una posición de la tabla ya ocupada, estaríamos en presencia de una colisión.

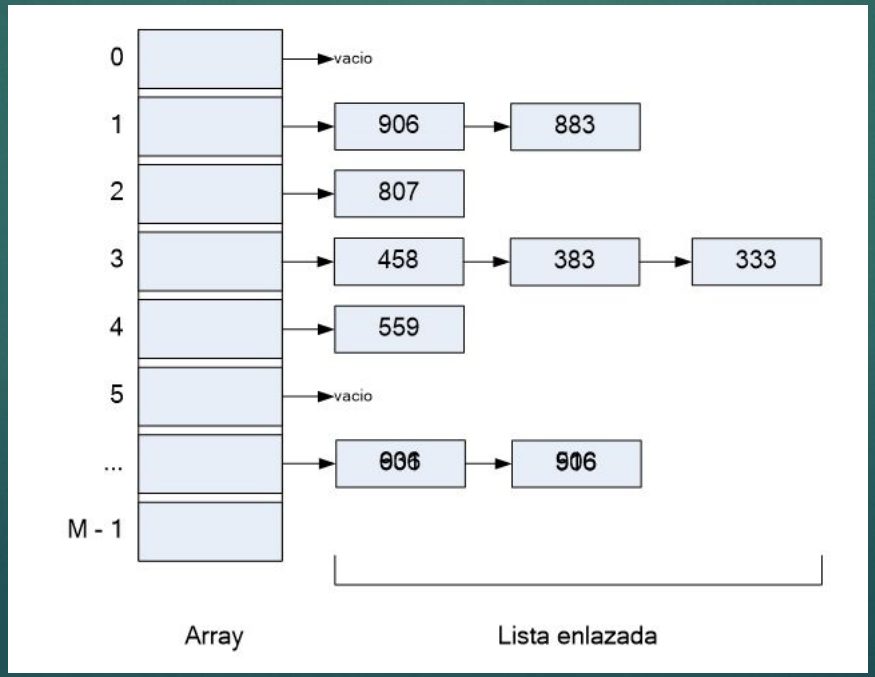
En este caso no podríamos almacenar dos registros en una misma posición, por lo que debemos encontrar otra ubicación en donde almacenar ese nuevo registro.

Hay varias técnicas de resolución de colisiones, entre ellas se encuentran:

- Encadenamiento
- Direccionamiento abierto

INDICES – HASHING – ENCADENAMIENTO

- En la técnica más simple de encadenamiento, cada casilla en el arreglo referencia una lista de los registros insertados que colisionan en la misma casilla. La inserción consiste en encontrar la casilla correcta y agregar al final de la lista correspondiente. El borrado consiste en buscar y quitar de la lista.

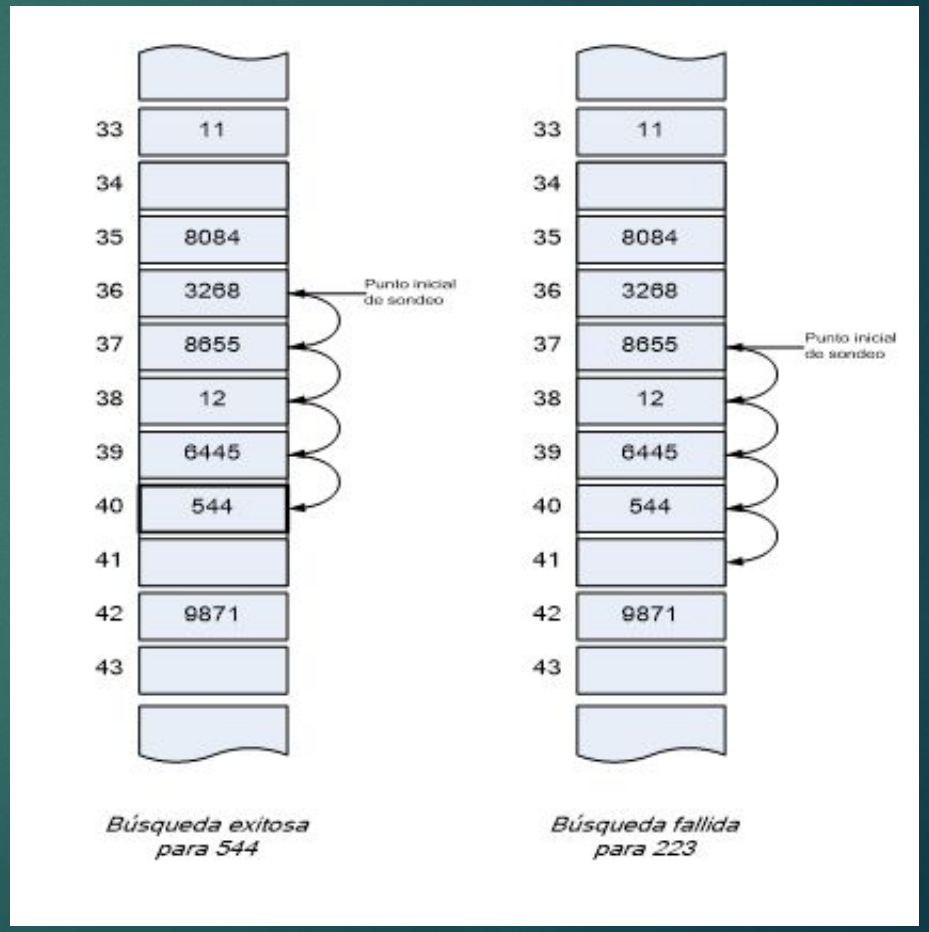


INDICES – HASHING – DIRECCIONAMIENTO ABIERTO

- En el direccionamiento abierto, cuando un registro no puede ser ubicado en el índice calculado por la función de hash, se busca otra posición dentro de la tabla. Hay varios métodos de elección de esa posición, los cuales varían en el método para buscar la próxima posición. Los más usados son:
  - Sondeo lineal
  - Sondeo cuadrático
  - Hashing doble

INDICES – HASHING – SONDEO LINEAL

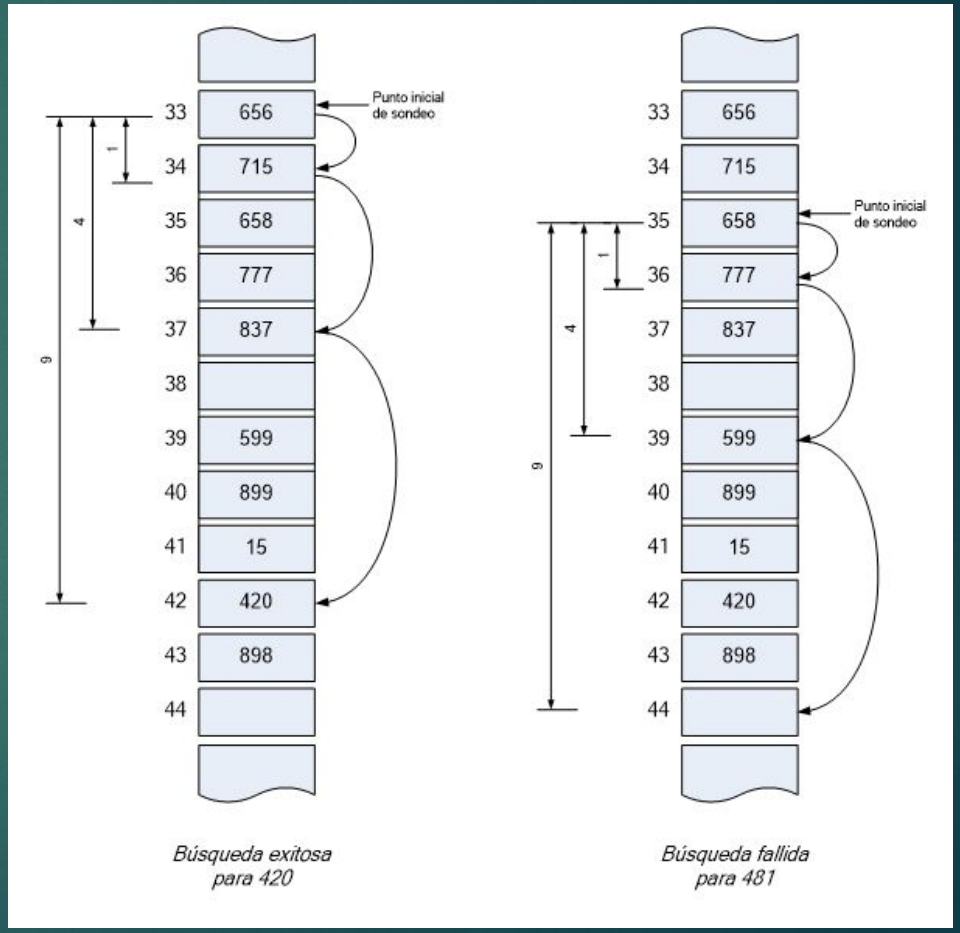
- **Sondeo lineal**: La forma más simple de resolver el problema es el sondeo lineal, el cual busca secuencialmente en la tabla hasta encontrar una posición vacía. En caso de alcanzar la última posición de la tabla, esta continúa buscando en la primera posición.



INDICES – HASHING – SONDEO CUADRATICO

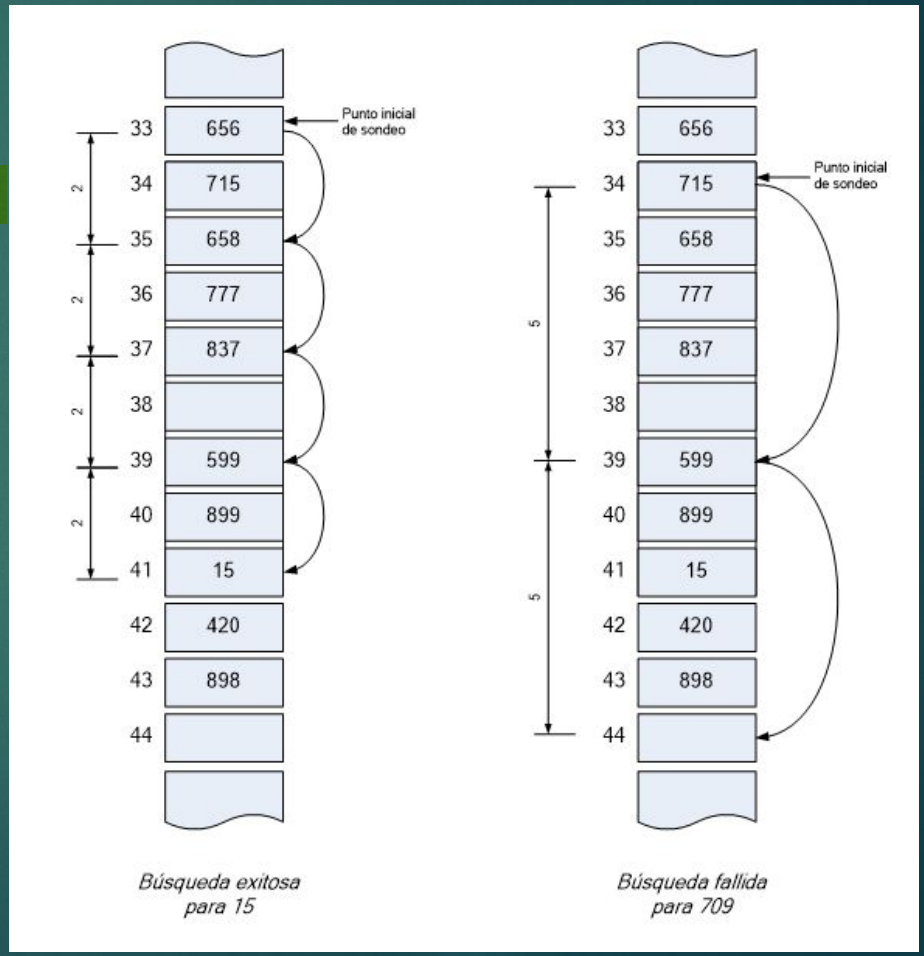
- **Sondeo cuadrático**: ubica la colisión examinando cierta posición a una distancia específica del punto inicial de sondeo. Esta variación permite una mejor distribución de las claves colisionadas.

- El nombre cuadrático deriva de la formula utilizada $F(i) = i^2$ para resolver las colisiones. Si resulta que la función de hash evalúa a la posición y la posición en es inconclusa, se intentan las posiciones $H+1^2, H+2^2, H+3^2, \ldots, H+i^2$



INDICES – HASHING – HASHING DOBLE

- **Hashing doble**: aplica la función hash a la clave una segunda vez, usando una función de hash distinta y usa ese resultado como tamaño de salto. Para que este método arroje buenos resultados la función de hash secundaria debe ser distinta que la primaria y debe arrojar valores mayores a cero (sino no habría saltos, y se produciría un bucle infinito).



INDICES – ARBOLES M-ARIOS

- Si se tiene un conjunto de datos muy grande, tan grande que no podemos colocarlo en memoria principal, nos veríamos obligados implementar el árbol de búsqueda en un almacenamiento secundario, como el disco. Las características de un disco a diferencia de la memoria principal hacen que sea necesario utilizar valores de ***M*** más grandes para poder implementar estos árboles de búsqueda de forma eficiente.
- El tiempo de acceso de un disco típico es de 1 a 10 ms, mientras que el tiempo de acceso típico de una memoria principal es de 10 a 100 ns. Por lo tanto, los accesos a memoria son entre 10.000 y 1.000.000 de veces más veloces que los accesos a disco. Para maximizar la performance es necesario minimizar a toda costa los accesos a disco.
- Además, los discos son dispositivos orientados a bloques. Los datos son transferidos en bloques de gran tamaño entre la memoria principal y el disco. Los tamaños de bloques típicos varían entre 512 y 4096 bytes. En consecuencia, tiene sentido tomar ventaja de esa habilidad para transferir grandes bloques de datos eficientemente.

INDICES – ARBOLES B

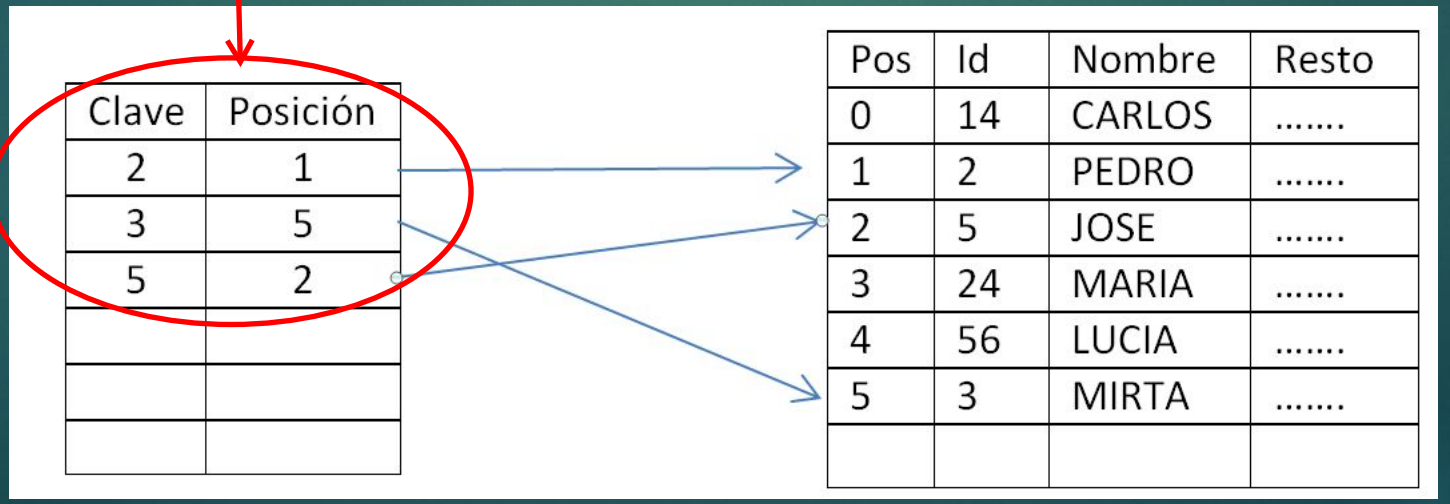
- El Árbol B es un tipo de árbol M-ario destinado a la creación de índices físicos para el acceso a la información.

- El objetivo principal es minimizar las operaciones de entrada y salida hacia el disco. Al imponer la condición de balance, el árbol es restringido de manera tal que se garantice que la búsqueda, la inserción y la eliminación sean todos de tiempo ***O(logn)***

- El grado M del Árbol B varía entre 50 y 2000 y se determina en base al tamaño de las claves y del tamaño de la página del disco.

INDICES – ARBOLES B – NODO

- El Árbol B es uno de los pocos árboles con dos tipos de nodos diferentes, el nodo raíz o rama y el nodo hoja.
- ***Nodo Hoja:*** tiene una componente de dato donde van los valores de las claves ordenados de menor a mayor y un componente puntero que contiene la posición relativa de los datos secuenciales correspondientes a esa clave

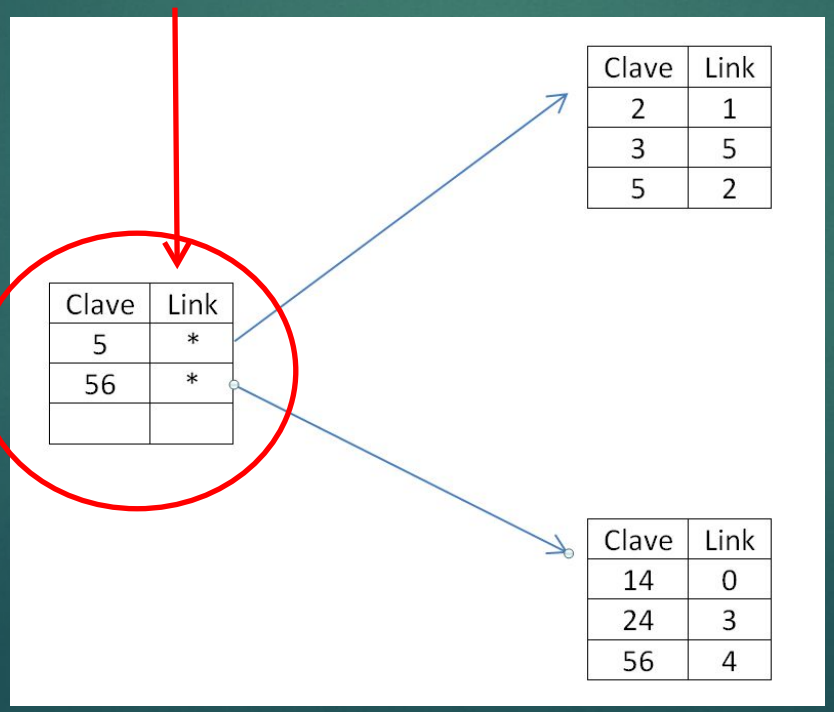


Clave	Posición
2	1
3	5
5	2

Pos	Id	Nombre	Resto
0	14	CARLOS	.....
1	2	PEDRO	.....
2	5	JOSE	.....
3	24	MARIA	.....
4	56	LUCIA	.....
5	3	MIRTA	.....

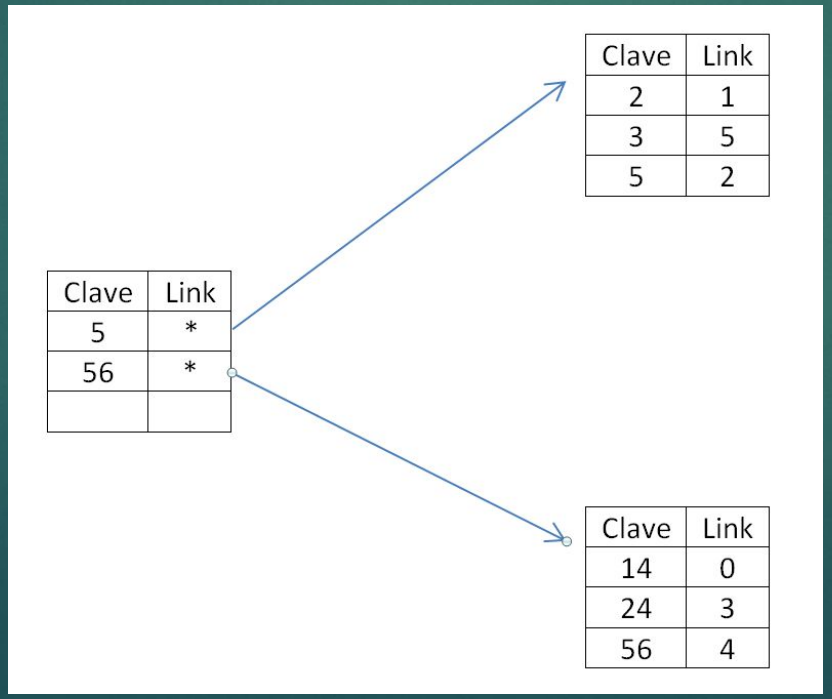
INDICES – ARBOLES B – NODO

- ***Nodo Raíz o Rama***: tiene una componente de dato donde van los valores de las claves ordenados de menor a mayor y un componente puntero que apunta al nodo que contiene claves menores o iguales que ella.



INDICES – ARBOLES B – BUSQUEDA

- ***Búsqueda:*** Buscar en un árbol-B es muy parecido a buscar en un árbol binario de búsqueda, excepto que en vez de hacer una decisión binaria, o de dos caminos en cada nodo, hacemos una decisión multicamino en base al número de hijos del nodo.



INDICES – ARBOLES B – INSERCION

- ***Inserción:*** Para insertar un elemento x, comenzamos en la raíz y realizamos una búsqueda para él. Asumiendo que el elemento no está previamente en el árbol, la búsqueda sin éxito terminará en un nodo hoja. Este es el punto en el árbol donde el x va a ser insertado.

- ***Split***: Si ocurre que cuando se llega a la hoja no hay espacio para insertar el nodo se produce lo que se denomina ***split*** que es un proceso que divide el nodo en dos dejando la mitad de elementos en cada uno respetando el orden de menor a mayor, quedando la mitad de los elementos más chicos en un nodo y la mitad de los elementos mas grandes en el otro.

INDICES – ARBOLES B – ELIMINACION

- ***Eliminación***: Para eliminar un elemento x, comenzamos en la raíz y realizamos una búsqueda para él. Asumiendo que el elemento existe, si existe se llegarà a la hoja donde esta y se borra, sino se dirà que no existe.

- ***Fusión***: Si ocurre que cuando se elimina el elemento x el nodo queda vacío, debe eliminarse el nodo, lo que puede generar una baja potencial en todos los antecesores de dicho nodo.

INDICES – ARBOLES B – CASO PRACTICO

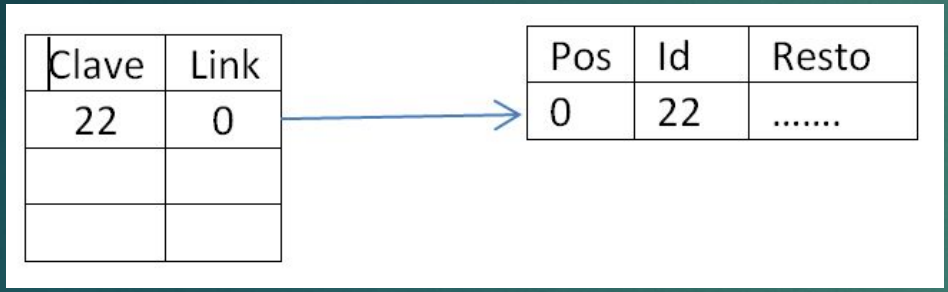
Dadas los siguientes números de claves posibles realizaremos el proceso de inserción, búsqueda y eliminación de elementos del árbol.

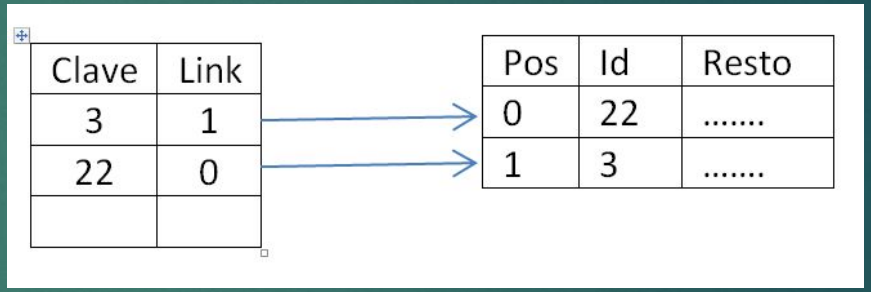
22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9

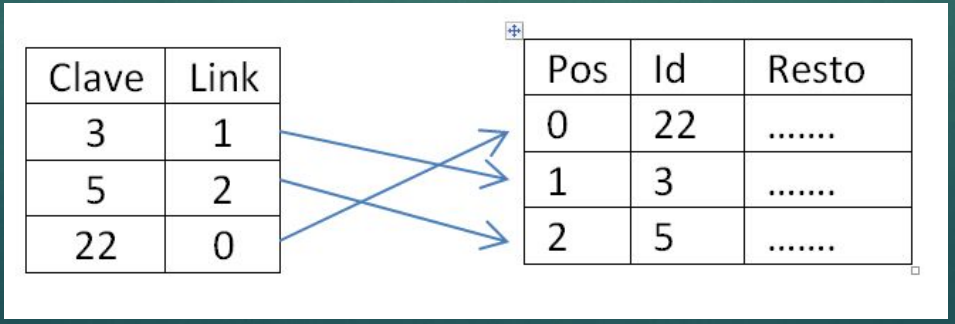
A modo práctico lo realizaremos con un árbol de grado 3 para ver las diferentes situaciones que pueden ocurrir.

INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9

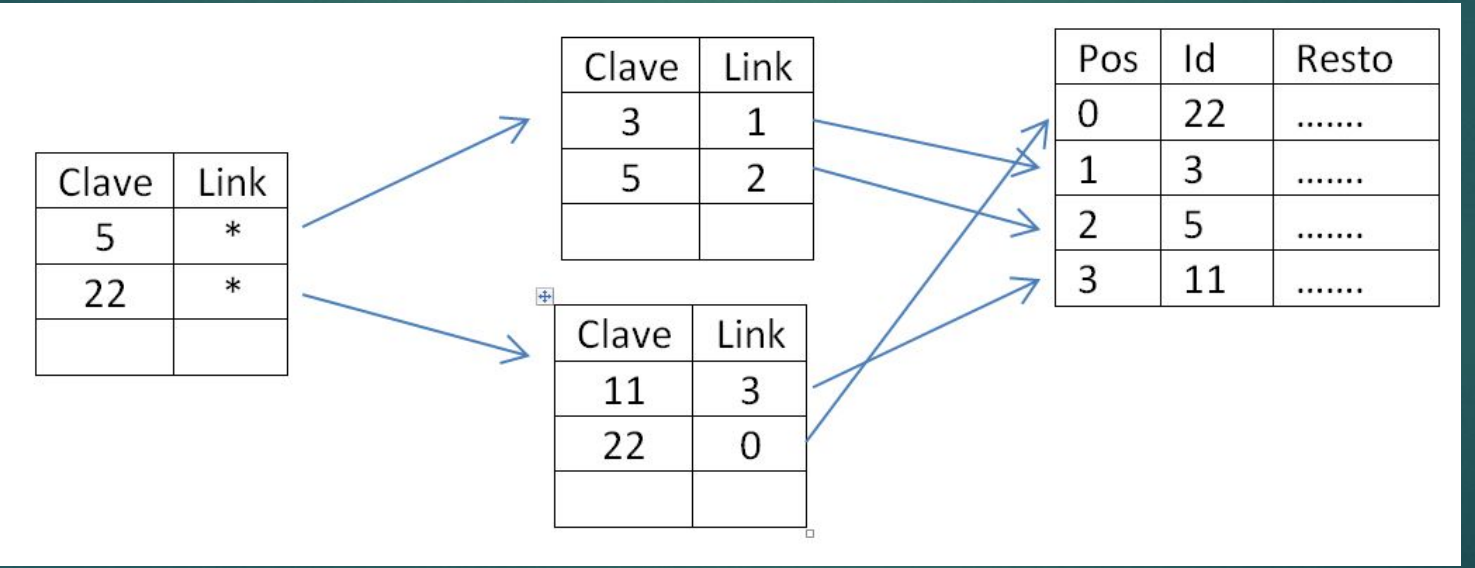






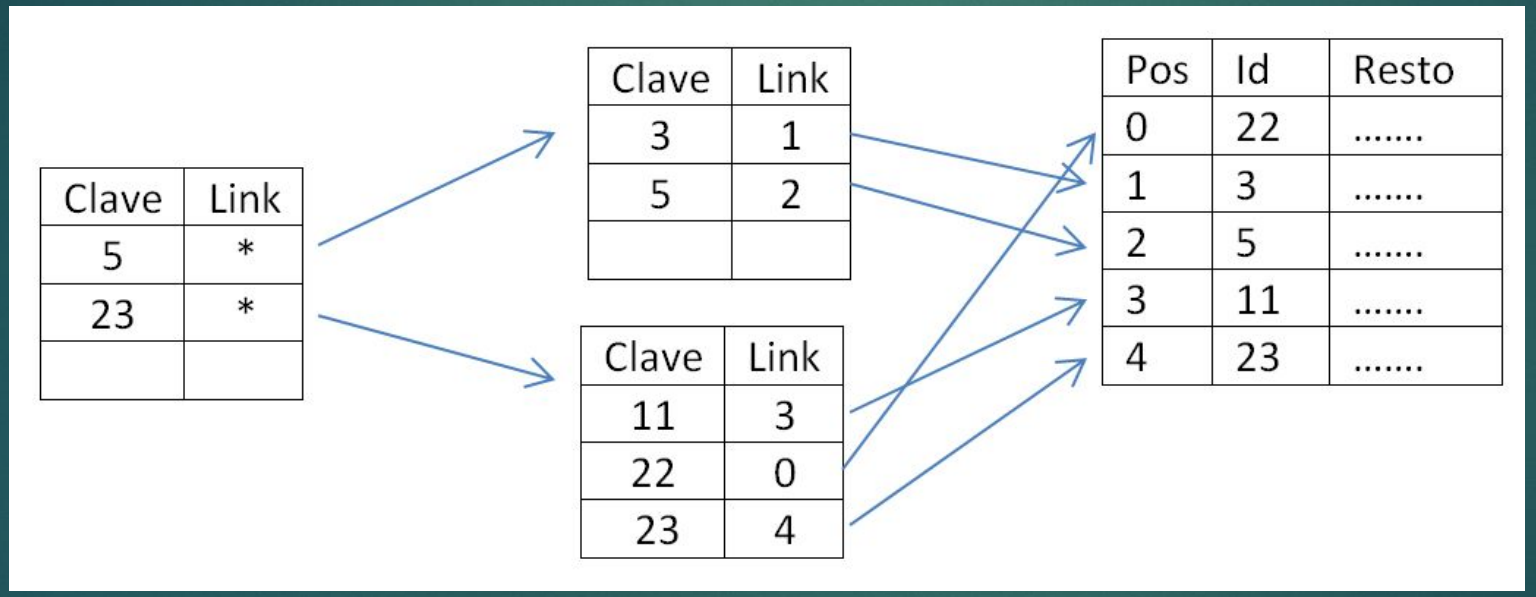
INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9



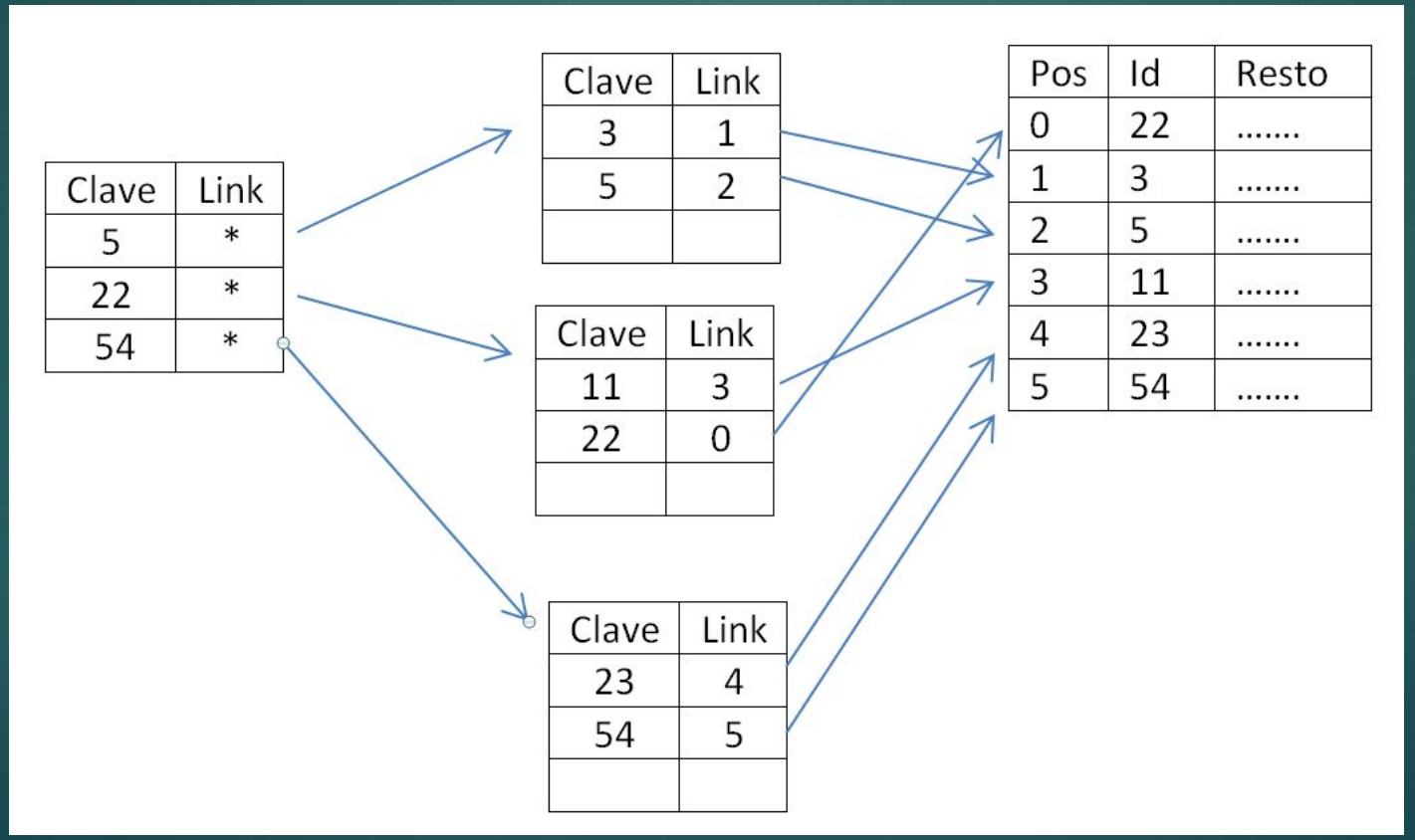
INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9



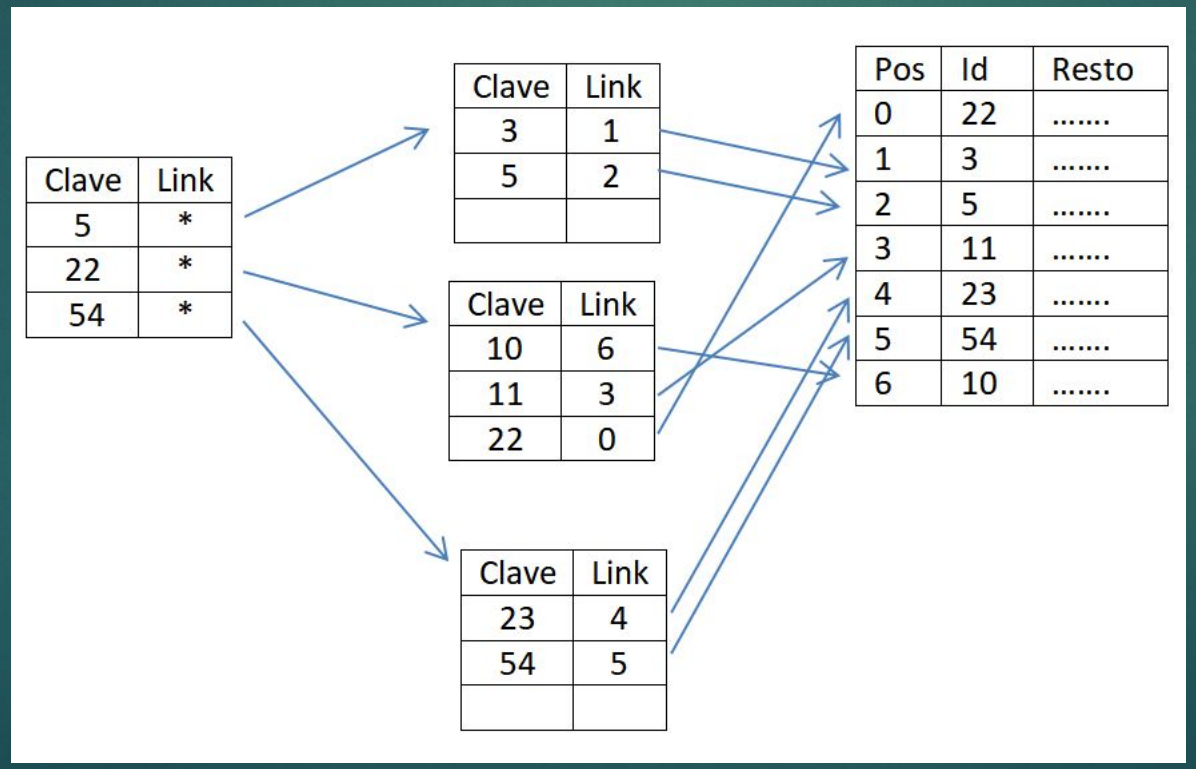
INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9



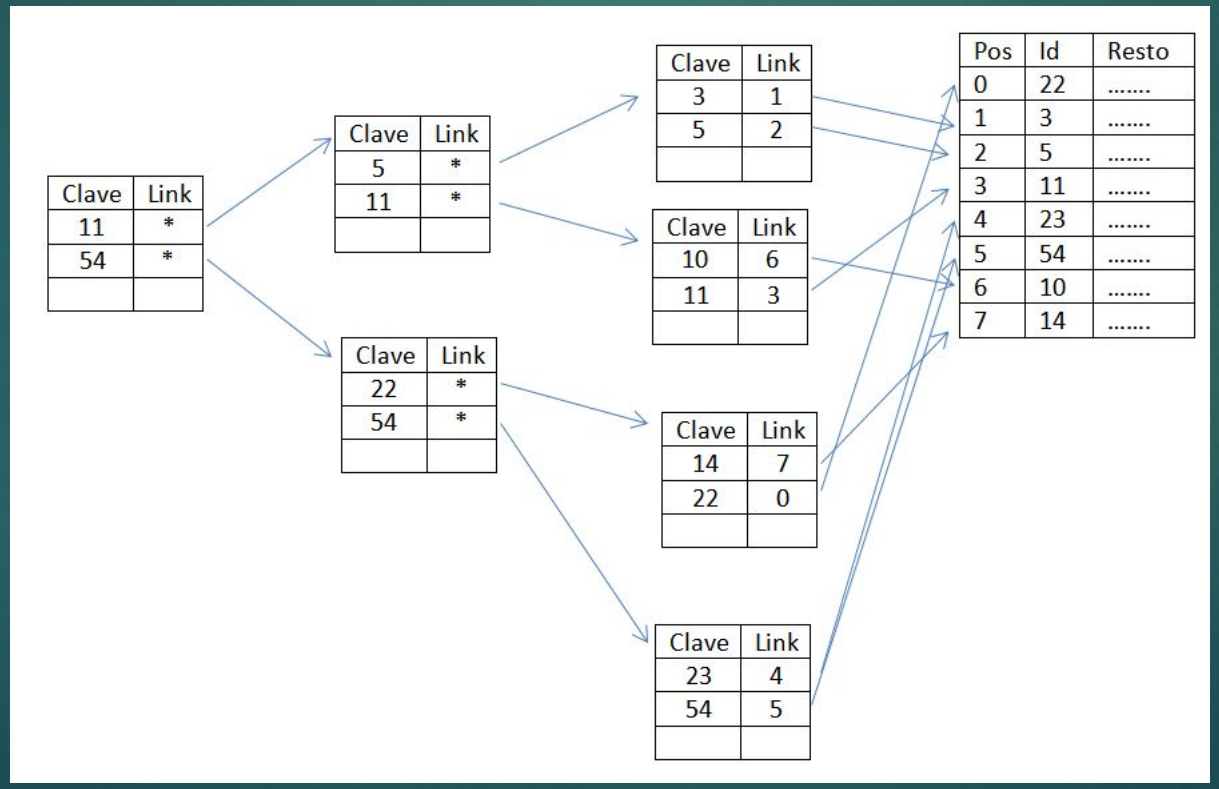
INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9



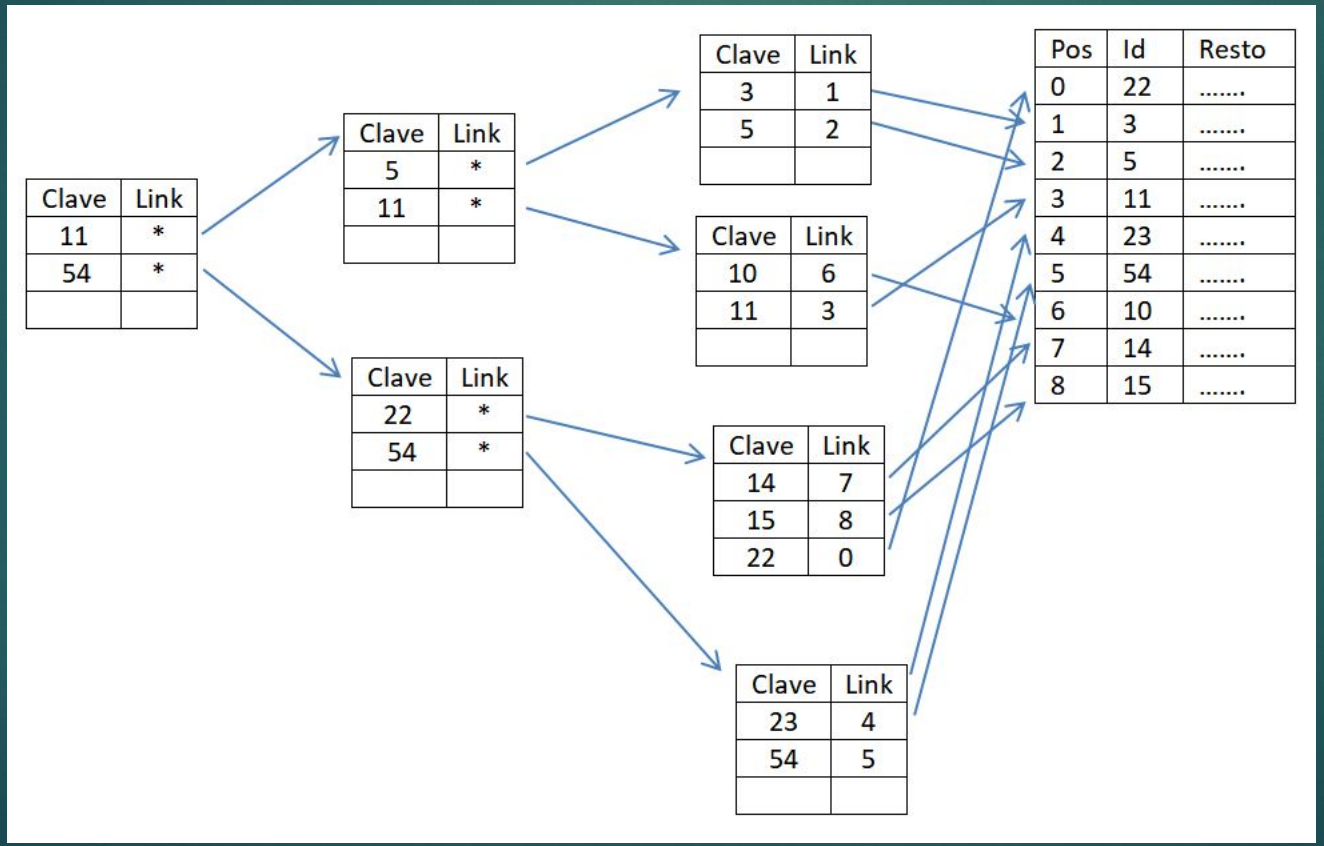
INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9



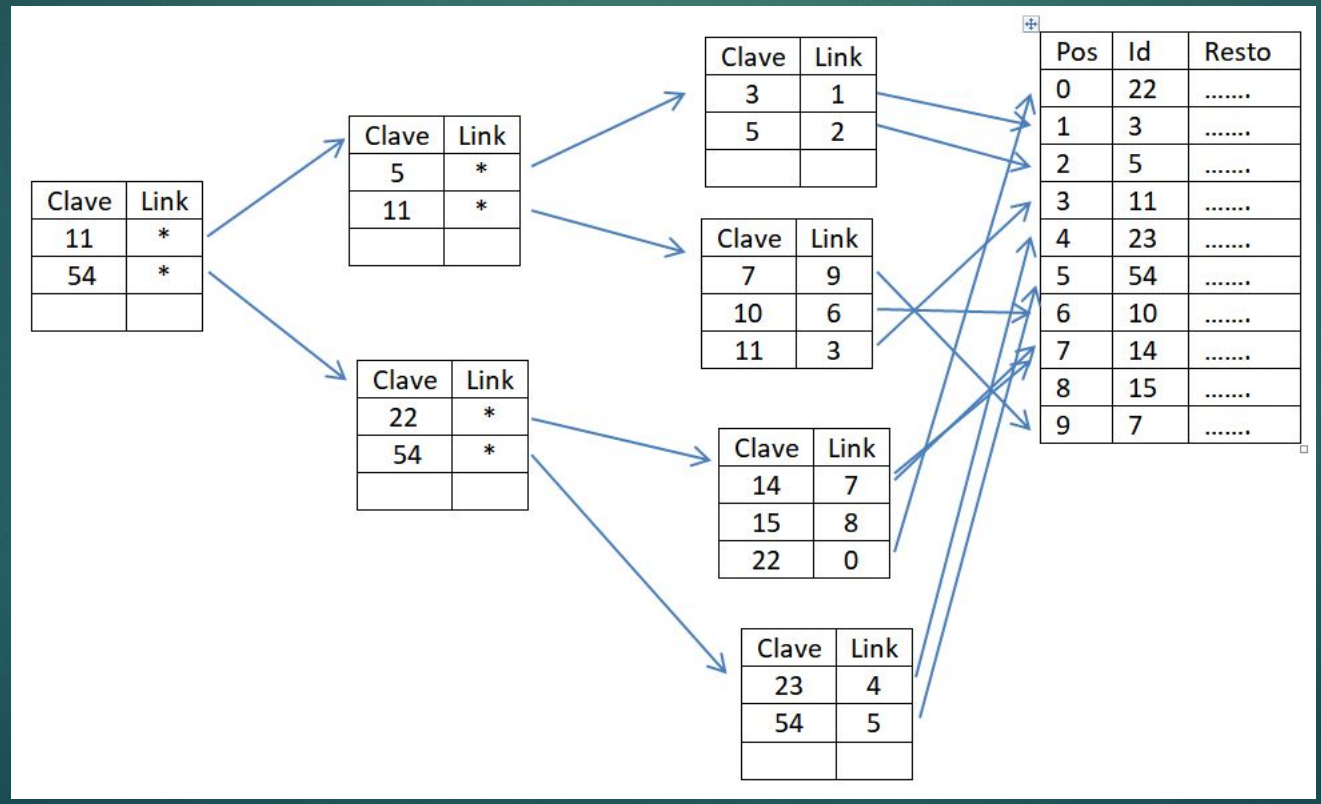
INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9



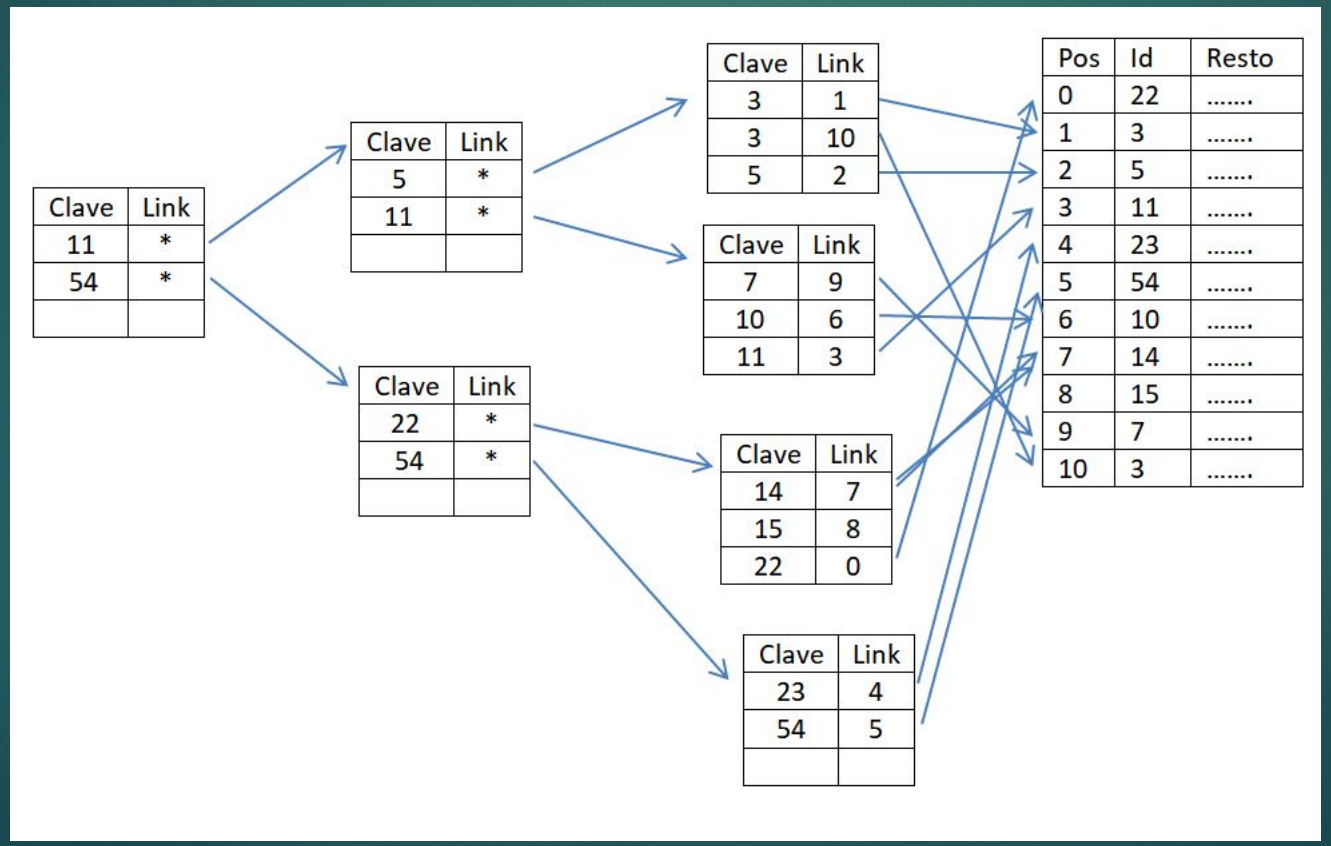
INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9



INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9



INDICES – ARBOLES B – CASO PRACTICO

22 – 3 – 5 – 11 – 23 – 54 – 10 – 14 – 15 – 7 – 3 – 9