

Rehashing

When the hash table becomes too full in open addressing, the successive insertion operation will take more time to complete. To overcome this situation rehashing technique is used.

* Build a hash table twice as big and scan down the entire original hash table place the hash table value to new table.

Eg: Insert {13, 15, 24, 6, 23} into a hash table size 7.

hash function is $h(x) = x \text{ mod } 7$

0	6
1	15
2	23
3	24
4	
5	
6	13

$$h(13) = 13 \% 7 = 6$$

$$h(15) = 15 \% 7 = 1$$

$$h(24) = 24 \% 7 = 3$$

$$h(6) = 6 \% 7 = 6 \quad h_0()$$

$$h(23) = 23 \% 7 = 2 \quad h_1()$$

→ double the table size & consider next prime no.

$$7 \times 2 = 14$$

17 ← Prime no.

0	
1	
2	
3	
4	
5	23
6	24
7	6
8	
9	
10	
11	
12	
13	13
14	
15	15
16	
17	

$$13 = 13 \% 17 = 13$$

$$15 = 15 \% 17 = 15$$

$$24 = 24 \% 17 = 6$$

$$h_0(6) = 6 \% 17 = 6 \quad \text{full}$$

$$h_1(6) = 7 \% 17 = 7$$

~~23~~

$$23 = 23 \% 17 = 5$$

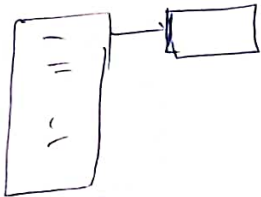
→ This entire operation is called Rehashing.

→ Implemented in several ways.

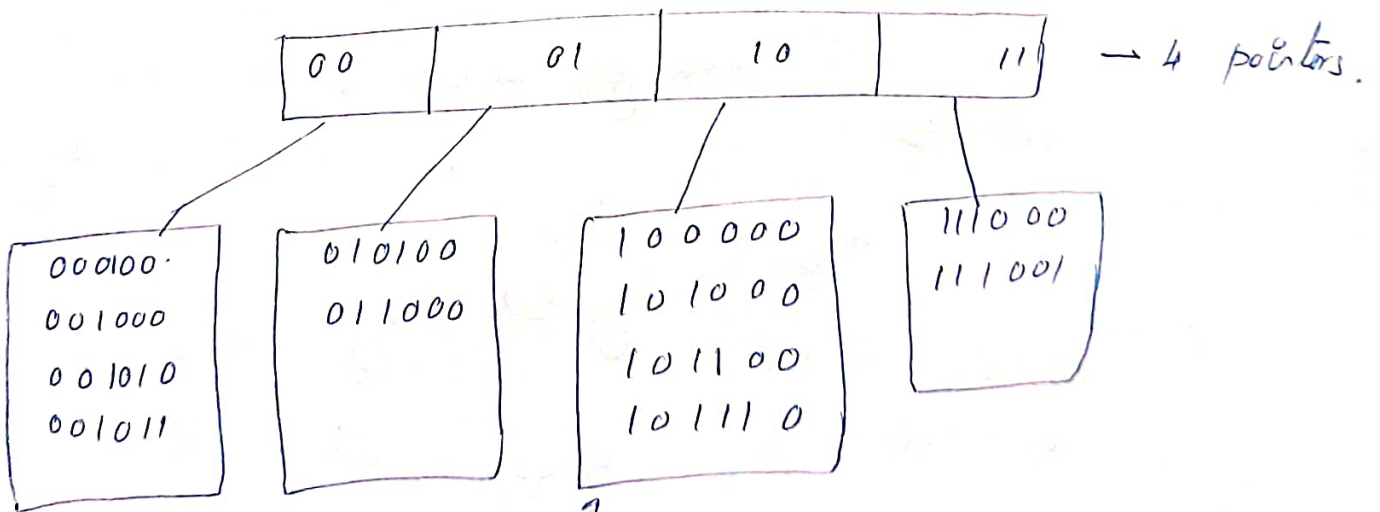
- ① Rehash as soon as table is half full.
- ② Rehash only when an insertion fails.
- ③ " when the table reaches a certain load factor.

Extendible hashing:

- Insertion, deletion difficult in open addressing.
- hash table stored in main memory & buckets (small table) on disk
- pointer used to point secondary table (bucket)



- pointer created by using hash function.
- stores 6-bit number.
- 2 bit pointer.



insert 100100 ↗ already filled, split and increase ptr.

