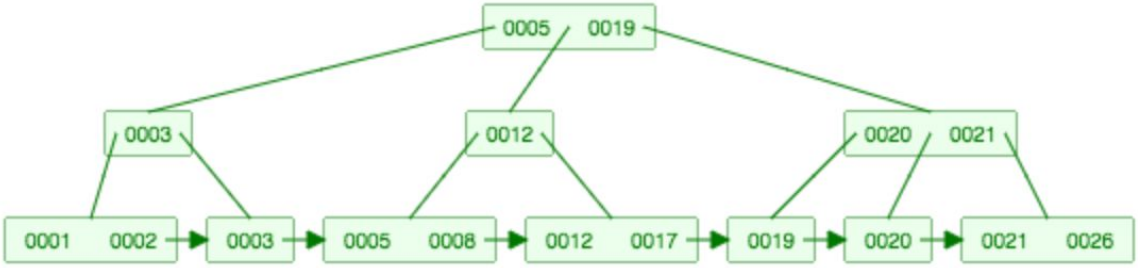
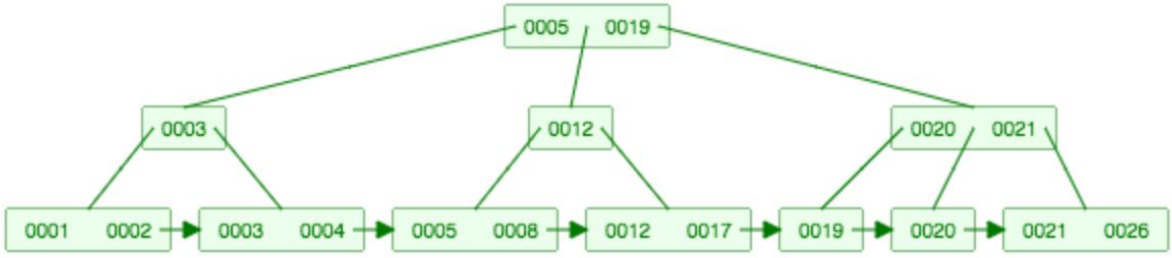
Q1: B+ Structure

1 - There should be **4** levels for representing 100000 entries using B+ structure with 25-entries per block.

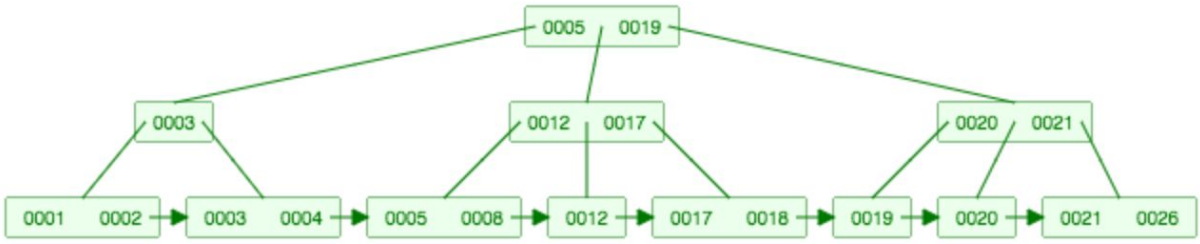
2 -
Insert 1:



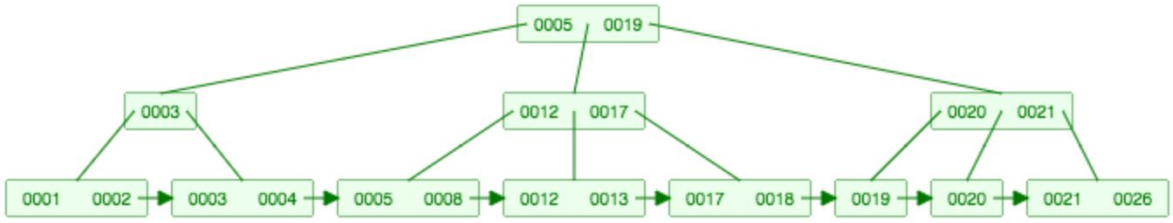
Insert 4:



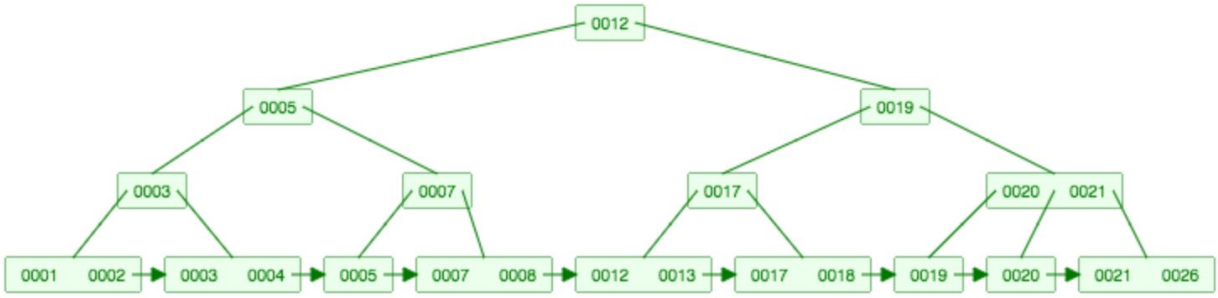
Insert 18:



Insert 13:

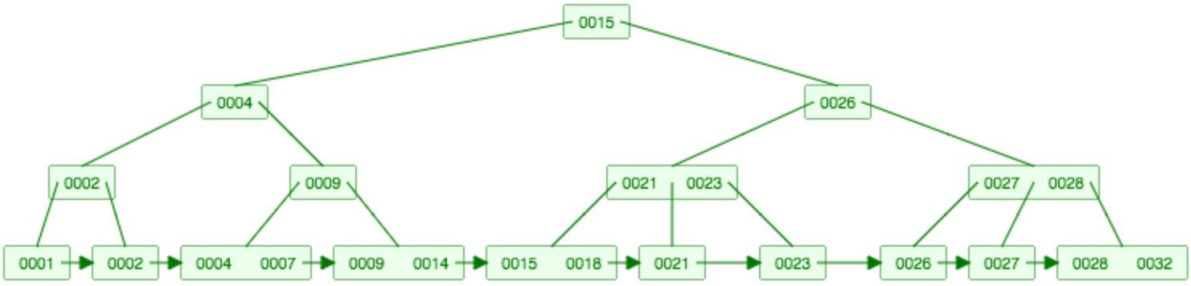


Insert 7:

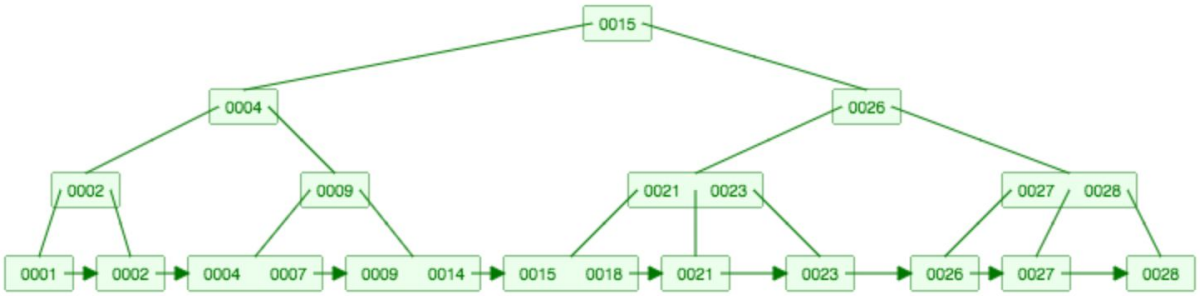


3 - keys which can make the tree height change using one insert operation are 7, 8, 9, 10, 23, 25, 26, 27, 28, 29, 30, 31, 32.

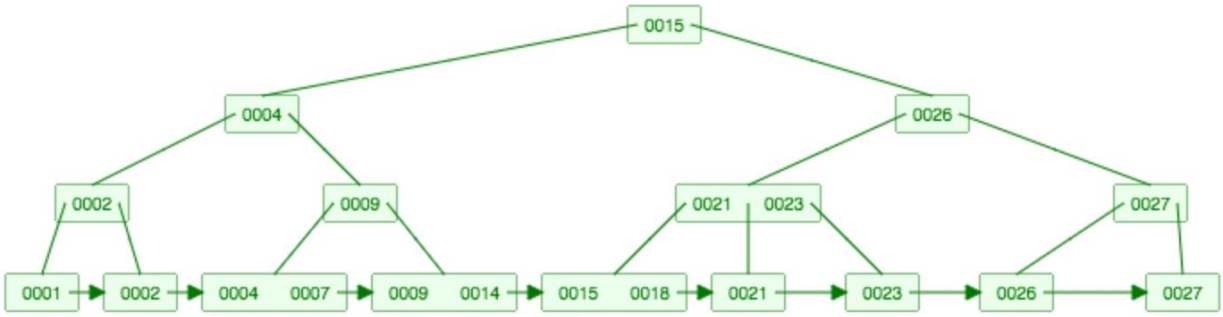
4 -
delete 24:



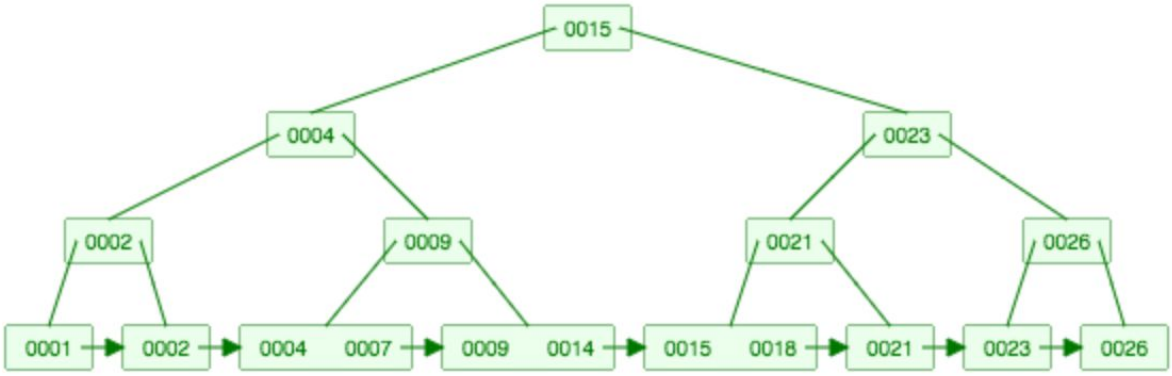
delete 32:



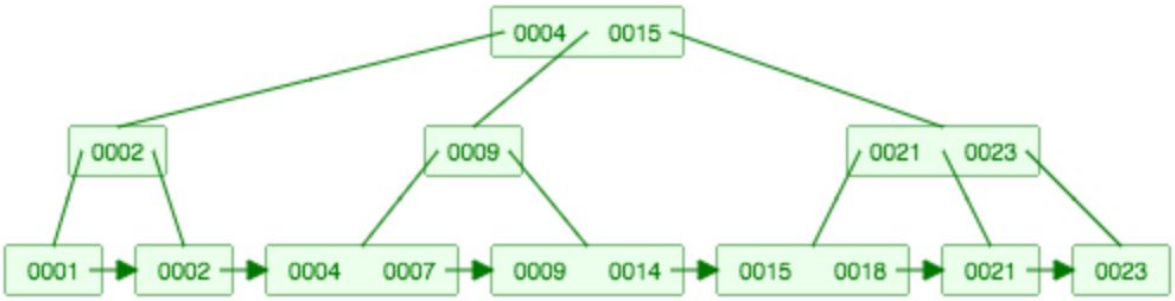
delete 28:



delete 27:

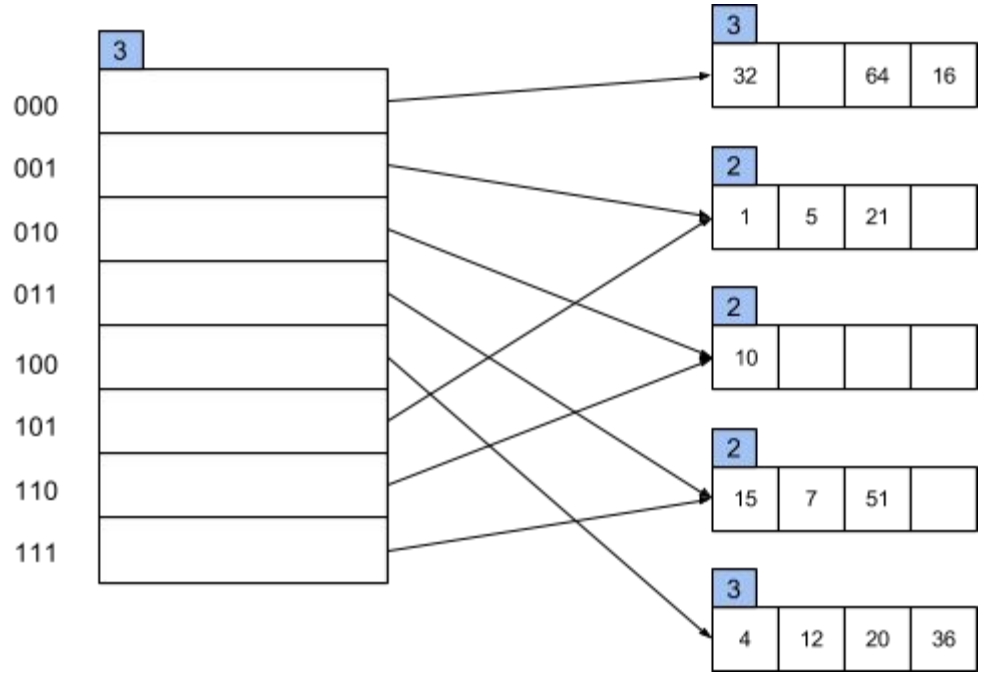


delete 26:

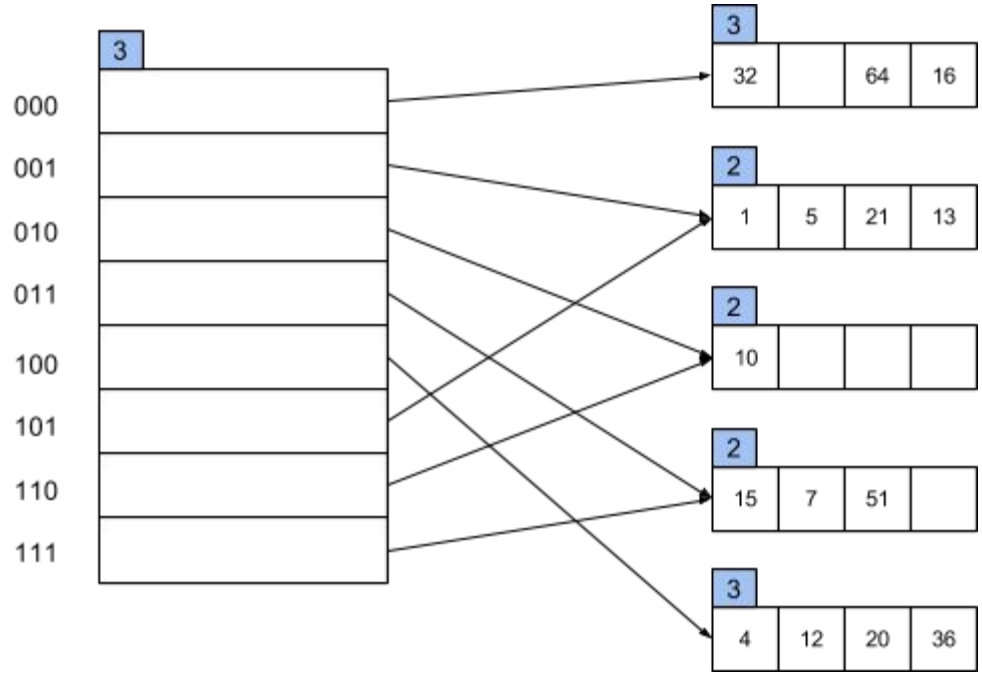

0004
0015
0002
0009
0021
0023
0001
0002
0004
0007
0009
0014
0015
0018
0021
0023

Q2: Extendable Hashing

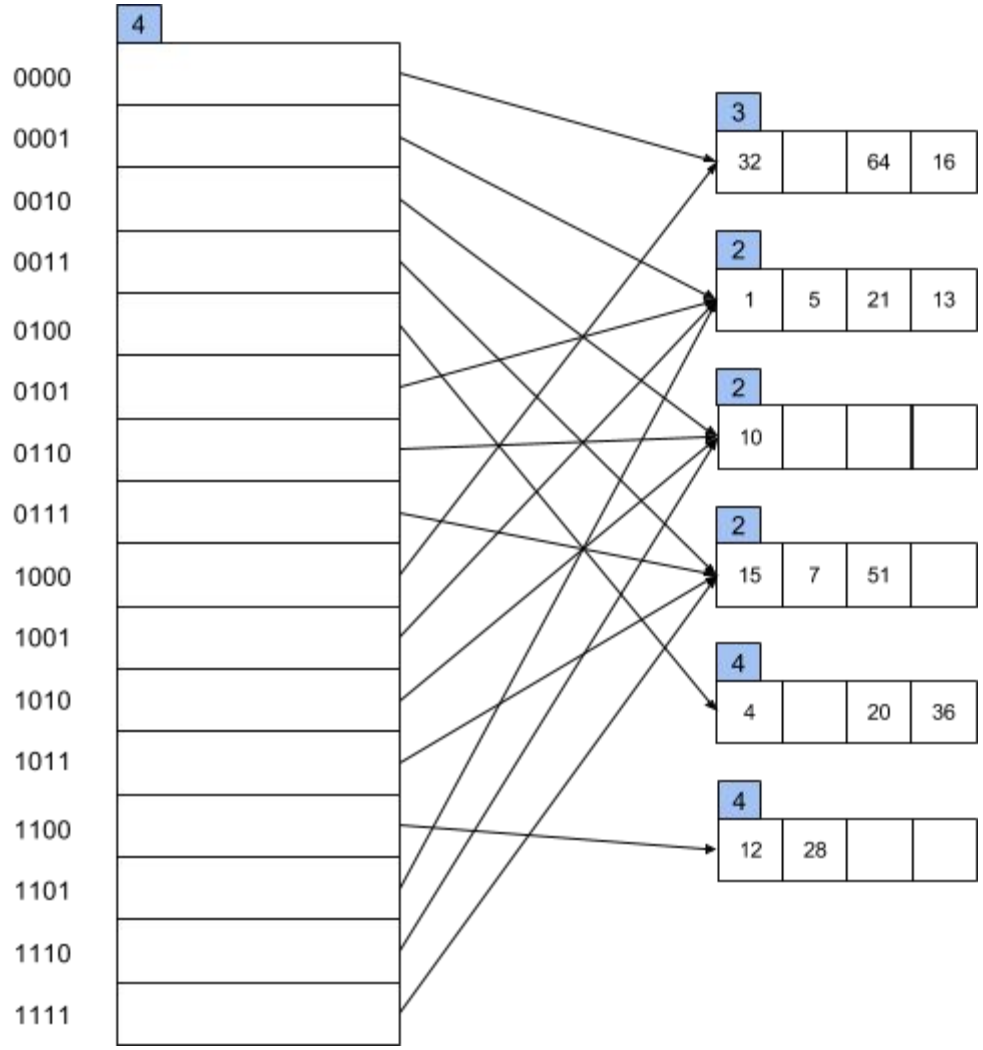
1 -
Insert 32:



Insert 13:

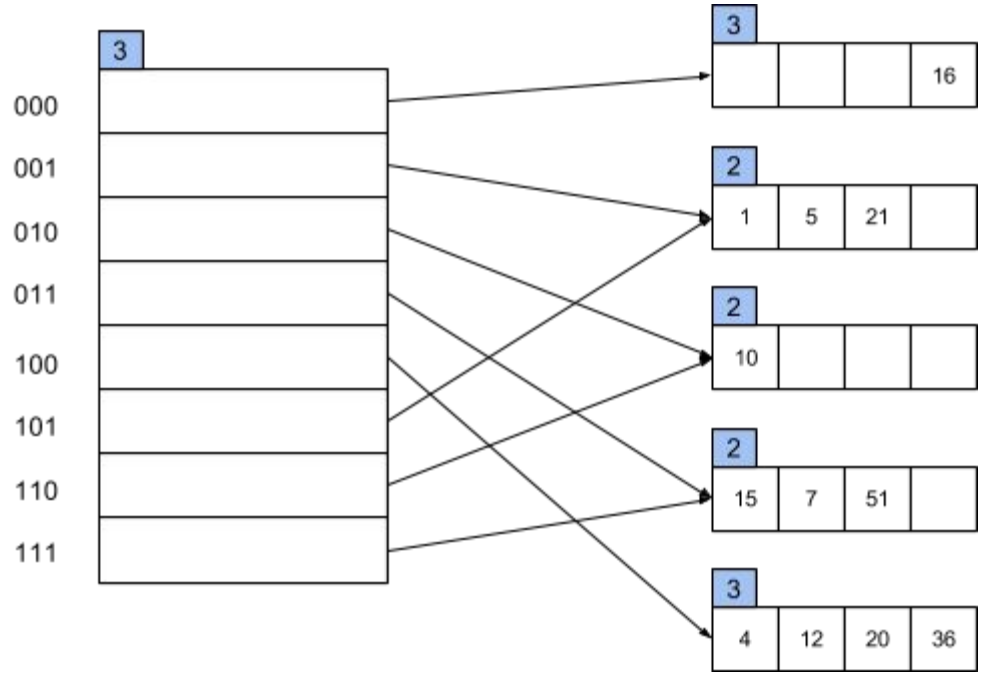


Insert 28:

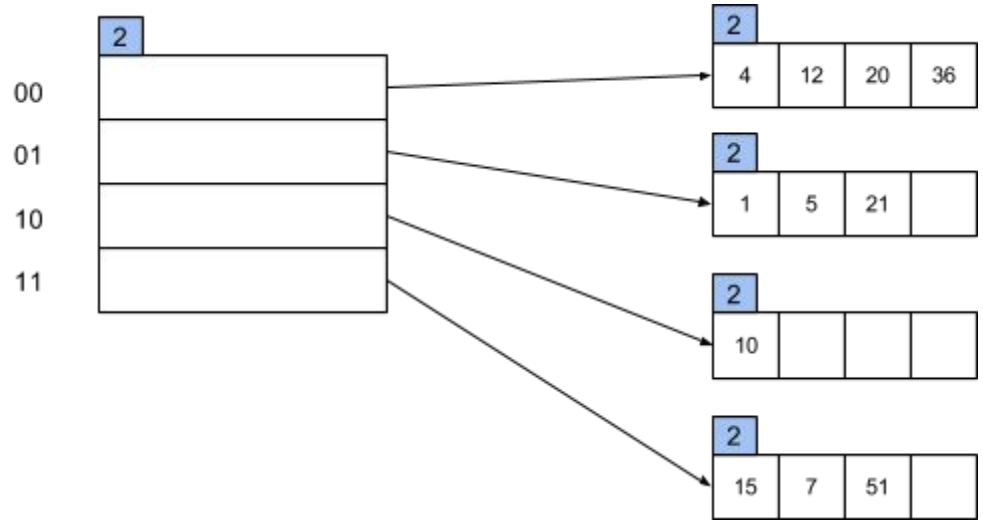

4
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
3
32
64
16
2
1
5
21
13
2
10
2
15
7
51
4
4
20
36
4
12
28

2 -

delete 64:



delete 16:



Q3: Optimizing queries by using indexes

Note: My queries can run in sqlite. Pay attention to that in different database APIs, there may be different functions to take the ‘year’ out of time data.

1 -
Query:

```
SELECT STRFTIME('%Y' ,dob) AS BirthYear,  COUNT(*) AS NumbersofUsers
FROM Users
GROUP BY STRFTIME('%Y' ,dob)
HAVING STRFTIME('%Y' ,dob) >=  '1970'
```

Suitable indexing technique: unclustered B+ index on Users.dob

2 -
Query:

```
SELECT COUNT(*) AS NumberofAds
FROM Ads
WHERE Ads.username = 'lhartj'
```

Suitable indexing technique: hashing index on Ads.username

3 -
Query:

```
SELECT COUNT(*) AS NumberofAds
FROM Ads
WHERE Ads.username = 'lhartj' AND Ads.price < 500
```

Suitable indexing technique: unclustered B+ index on Ads.username and Ads.price