1) // given 2 surfaces for platter so we have 8 surfaces.

// 1 MB = 1024 KB

Last position of disk head = <3, 4095, 127>

a) size of disk = no of cylinders × number of surfaces × $\frac{(\text{num of sectors})}{\text{track}}$ × sector size

= 8192 × 8 × 512 × 512 B

= 16 GB

b) // assume: the last head position is the first empty sector. we write from root sector.

Ideally the file would be stored contiguously. This is so that when file is accessed (sequentially storing this way) the block access time is minimized. we assume no block is skipped while reading previous blocks.

1000 MB = 1000 × $(1024)^2$ B is file size

Number of tracks reqd = $\frac{1000 \times (1024)^2 \text{ B}}{\text{Total size of a track}}$

Total size of track = 512 × 512 B = $\frac{(1024)^2}{4}$ B

(512: num of sectors; 512 B: size of sector)

Number of tracks reqd = 1000 × 4 = 4000.

final sector (last) = <3, 8095, 127>.

c) First we seek the 1st track (4095) of file and read 128 sectors.
next we seek adjacent track and read 512 sectors. } so on.

we seek last track (8095) and read 128 sectors

So, a total of (4000+1) seeks and 3999 full rounds and two half rounds (1st track of file & last track of file) are needed to read file. (i.e) (4001)(5 ms) + 4000 (time taken for 1 round)

time taken for 1 round $= \frac{1}{7200} \times 60 \times 1000$ ms. (given, 7200 RPM)

$= 8.333$ ms.

Total time taken $= (4000)(5\text{ ms}) + 4000 \times (8.333\text{ ms})$

$= 20\text{s} + 33.333\text{s} + 0.005\text{s}$

$= \underline{53.338\text{s}}$

d) A. FCFS.

8095 → 200 → 5000 → 4200 → 5 → 7200 → 4000 → 2200
→ 800 → 6500.

$\Rightarrow 7895 + 4800 + 800 + 4195 + 7195 + 3200 + 1800 + 1400 + 5700$

$= \underline{36985}$

B. Elevator

Previously it was going outward (i.e) direction is towards right)

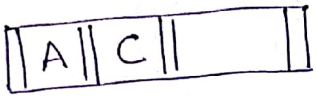
order in which it visits is:

8095.
⟶
then direction changes to left as no more requests >8095.
and serves all till it reaches 5 and stops.
(i.e) 7200, 6500, 5000, 4200, 4000, 2200, 800, 200, 5.
Total distance travelled $= |8095 - 5| = \underline{8090}$

2)

a) Insertion of A -

Insertion of C



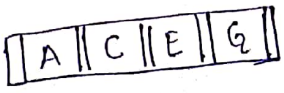
Insertion of E

Insertion of G -

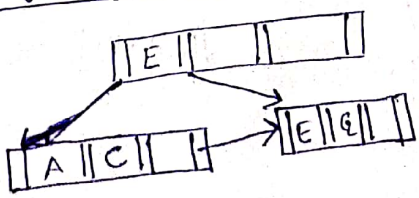
Above node is split into 2 ~~nds~~ and height of tree is increased by 1 where half of the values are in one node and other half in other. 'E' is passed to parent & in this case becomes root.

~~Before~~

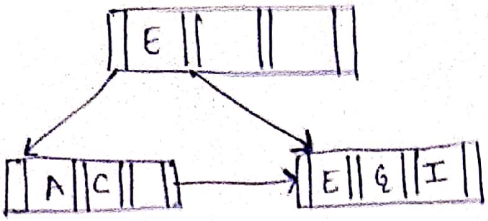
i) Temporary node before split



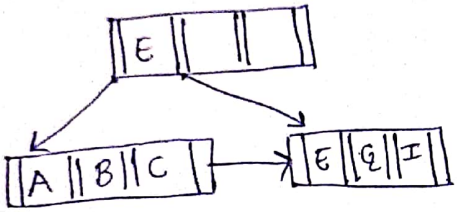
ii) After split



After insertion of I



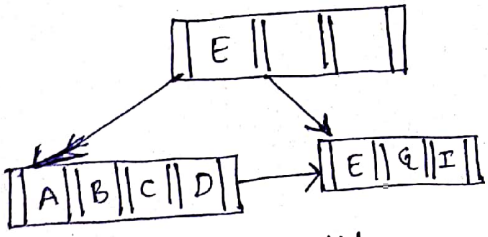
After insertion of B



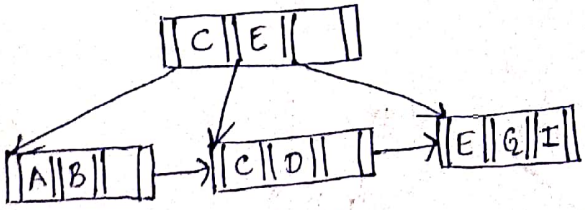
After insertion of D

The leftmost leaf node is split into 2 nodes similar to ~~insertion~~ one of the previous cases. 'C' is passed to parent node to be inserted into it.

i) Temporary node before split



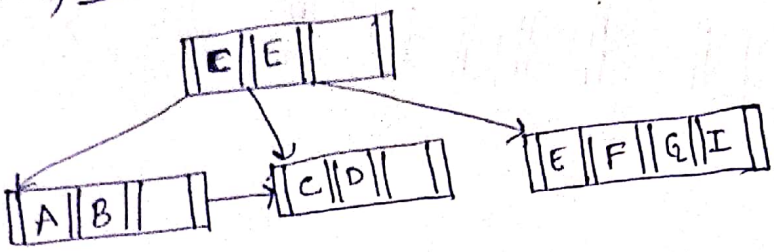
ii) After split



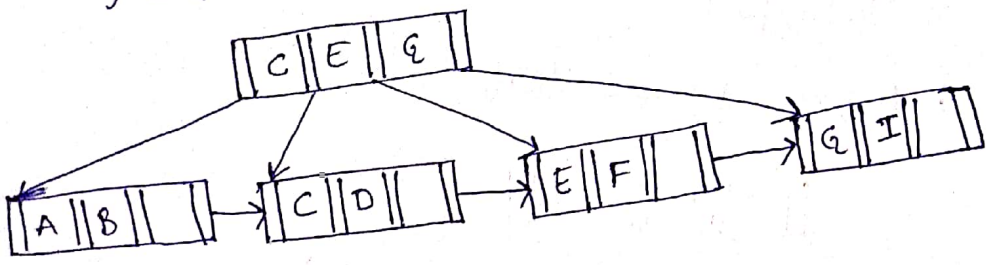
After insertion of F

The rightmost leaf node is split into 2 nodes similar to previous case. 'G' is passed to parent to be inserted into it.

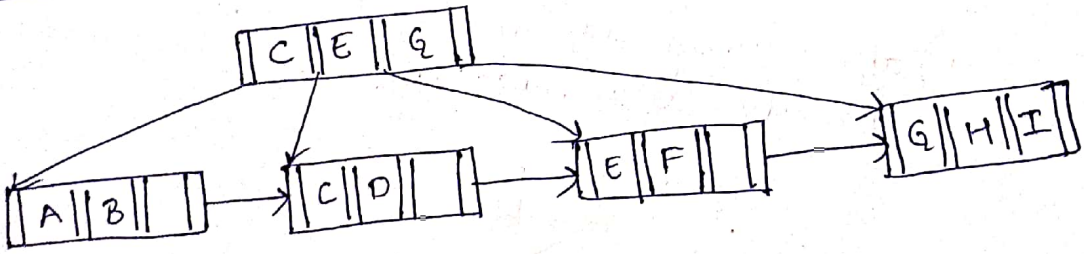
i) Temporary node before split



ii) After split



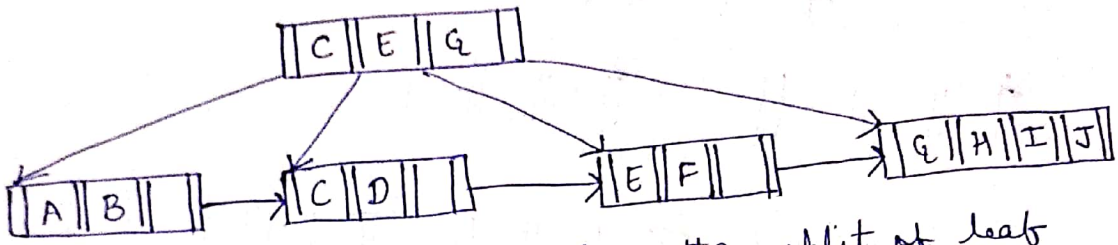
After insertion of H



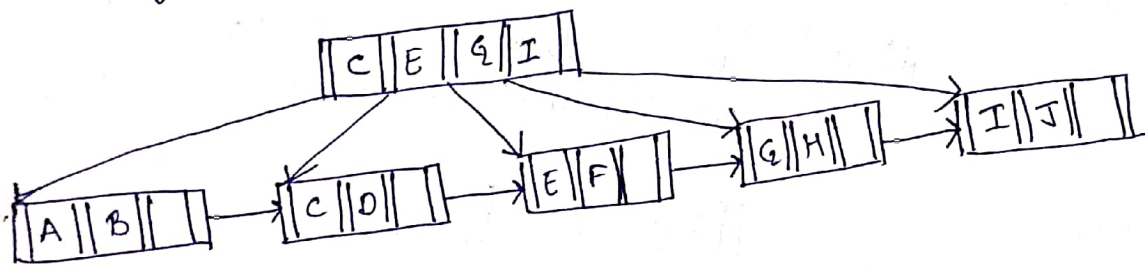
After insertion of J

The rightmost leaf node is split into 2 nodes similar to previous case. 'I' is passed to parent to be inserted into it.

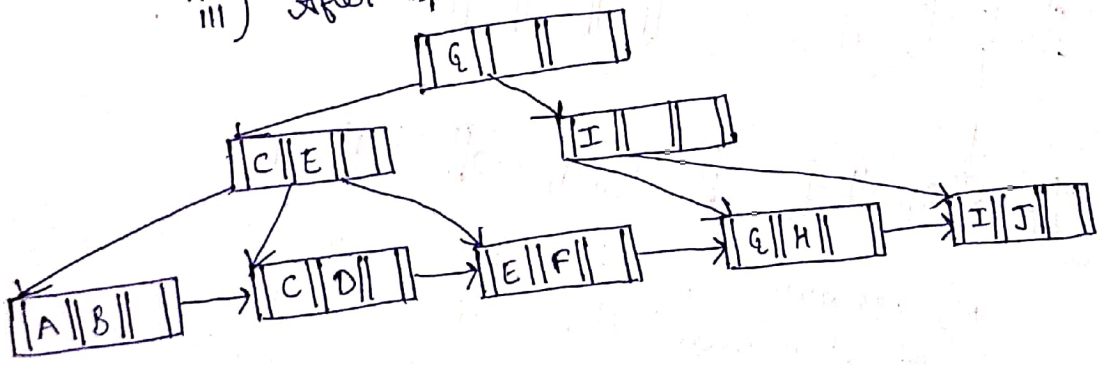
i) Temporary node before split



ii) Parent temporary node, after split of leaf



iii) After split

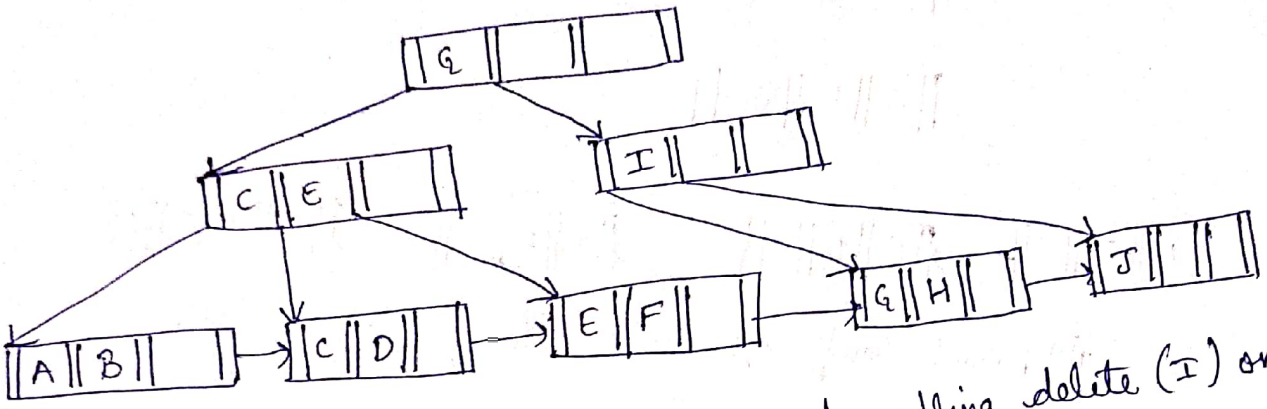


The parent is overfull so it is split into two nodes just like previous cases and height is increased by 1. 'G' is passed to parent and in this case becomes root.

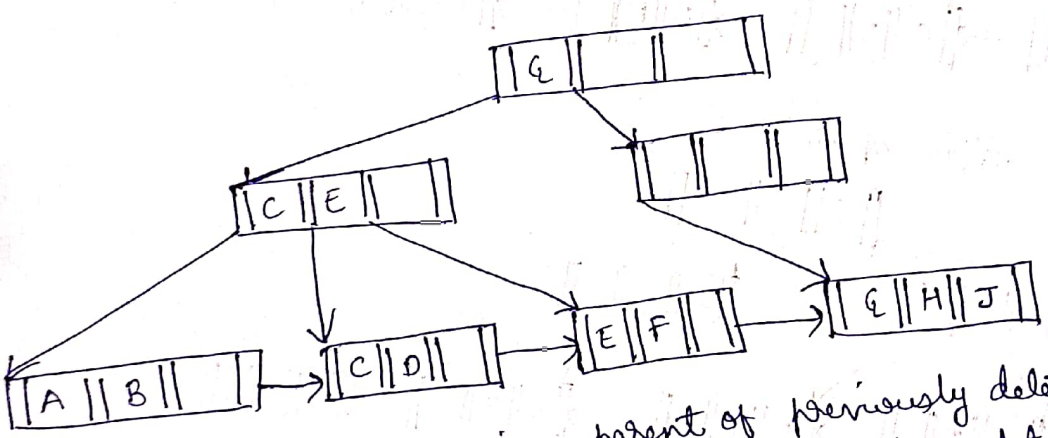
b) After deleting I

the rightmost node is underfull so it combines with the sibling as it has space.

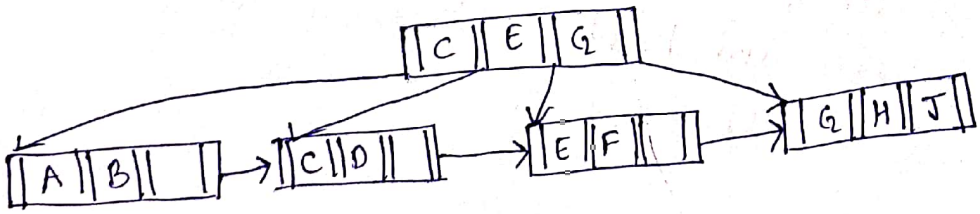
i) Before joining leaf



ii) After joining leaf and calling delete (I) on its parent, just after delete.

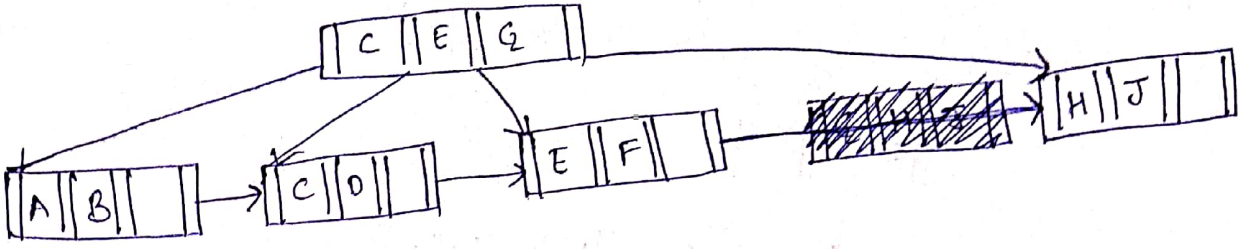


iii) removing parent of previously deleted leaf because it is underfull. results in deletion of its parent (root) as root has only value & Merges with sibling to become root, so height is decreased by 1.



After deleting G

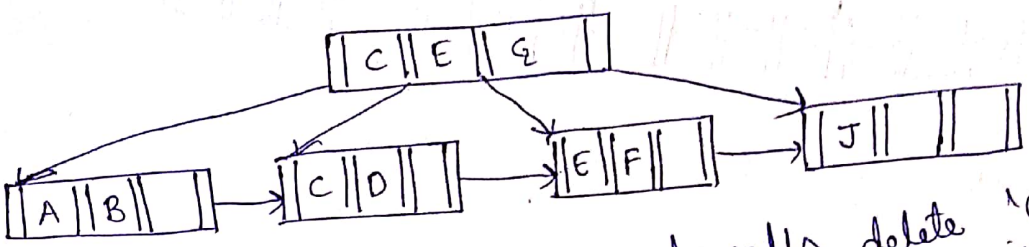
After 'G's deletion, the node is not underfull so it is the end of the procedure.



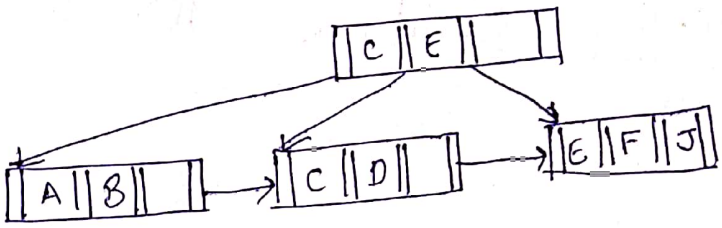
After deleting H

rightmost leaf node is underfull so it combines with sibling as it has space.

i) Before joining leaf.

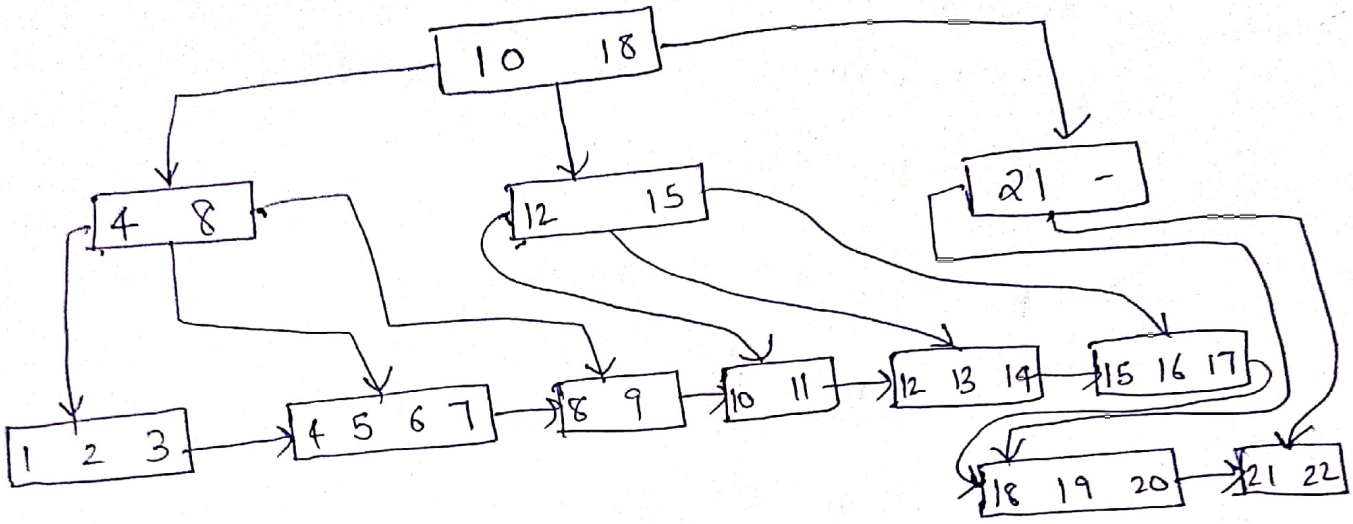


ii) after joining leaf and calls delete 'G' on parent, just after delete node satisfies its conditions so end.

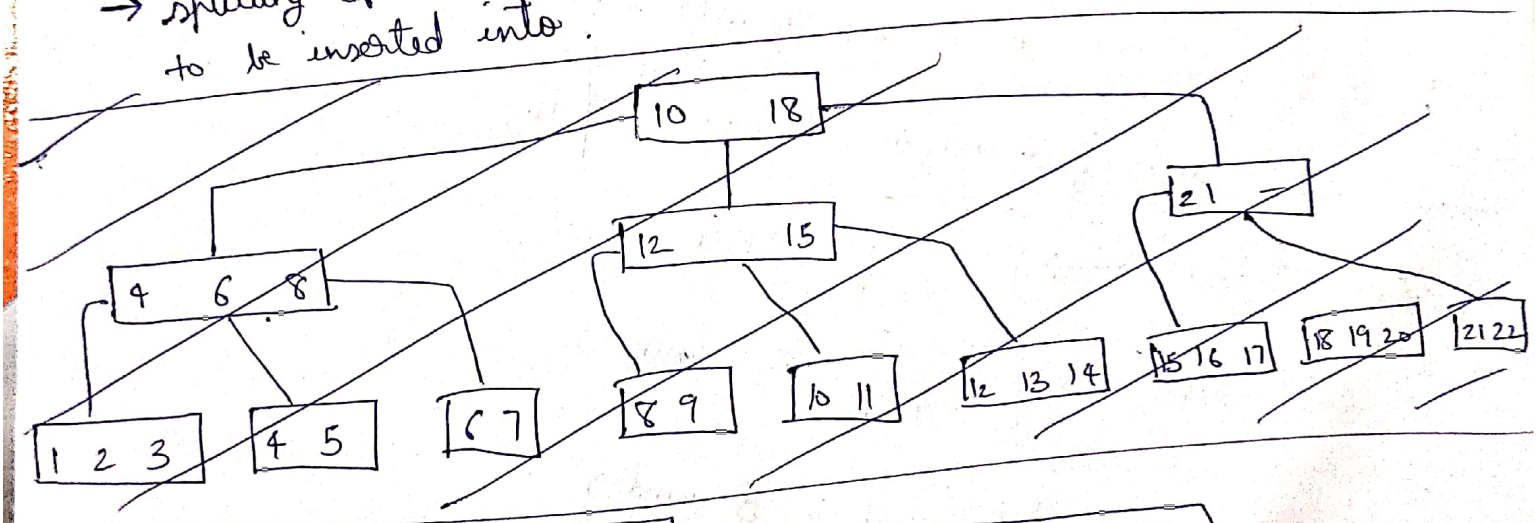


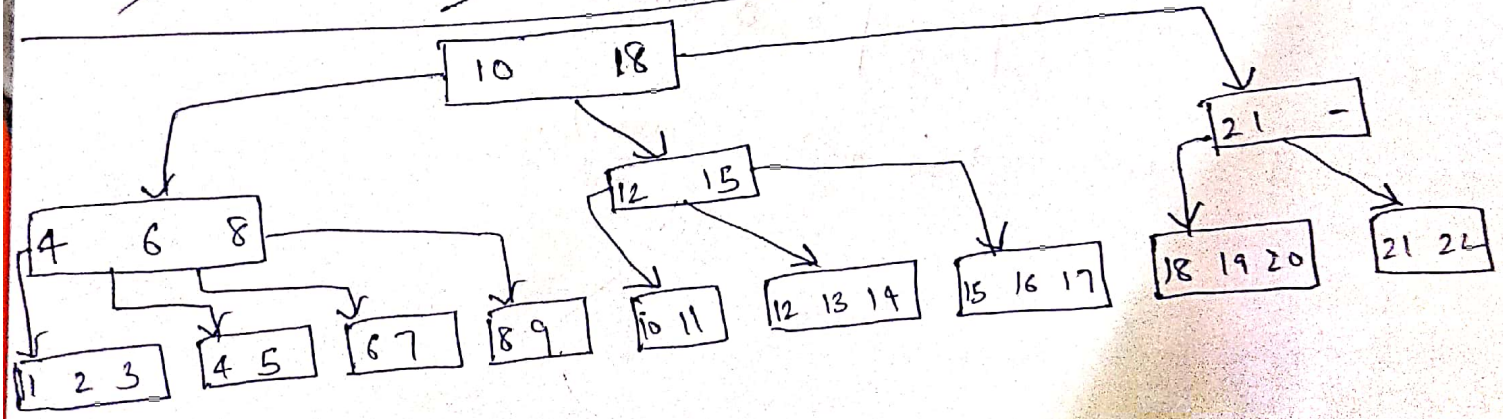
3)

→ After insertion of 5 and before splitting node, the temporary node



→ splitting of temporary node and passing of '6' to the parent to be inserted into.





4) A leaf node can have n pointers and n-1 key values.

(a)

size of node = $n(15) + (n-1)(45)$

since node can be atmost of size of one disk page.

we need to maximize n, s.t.,

$n(15) + (n-1)(45) \leq 1500$.

$\Rightarrow n + (n-1)3 \leq 100$

$\Rightarrow 4n \leq 103$

$\Rightarrow n = 25$

~~Levels in tree = $\lceil \log_{\lceil 25/2 \rceil} 30000 \rceil$. ; there are 30000 records in total.~~

Each leaf can store $(n-1) = 24$ records maximum.

Hence, total levels (~~fully~~ using the bulk algo) = $\lceil \log_{24} 30{,}000 \rceil = 4$

b) 4th level $\longrightarrow \lceil \frac{30000}{24} \rceil = 1250$ nodes

3rd level $\longrightarrow \lceil \frac{1250}{24} \rceil = 53$ nodes

2nd level $\longrightarrow \lceil \frac{53}{24} \rceil = 3$ nodes

1st level $\longrightarrow$ 1 node (root).

c) A leaf node can have n pointers and n-1 key values
size of leaf node = $n(15) + (n-1)(10)$.
as node can be almost of size one disk page

$\Rightarrow$ maximize n s.t.

$$n(15) + (n-1)(10) \leq 1500$$

$$n(3) + (n-1)(2) \leq 300$$

$$5n \leq 302$$

$$\Rightarrow \underline{\underline{n = 60}}$$

each leaf can store maximum of $(n-1) = 59$ records.

Total levels $= \left\lceil \log_{59}^{30000} \right\rceil = \underline{\underline{3}}$

d) All nodes are 70% full.

$\Rightarrow$ similar to previous case where we maximize n s.t.

$$n(15) + (n-1)(45) \leq \frac{70}{100} \times 1500.$$

$$\Rightarrow n + (n-1)(3) \leq \frac{70}{100} \times 100$$

$$\Rightarrow 4n \leq 73$$

$$\Rightarrow \underline{\underline{n = 18}}$$

each leaf can store maximum of $(n-1) = 17$ records.

Total levels $= \left\lceil \log_{17}^{30000} \right\rceil = \underline{\underline{4}}$

5)

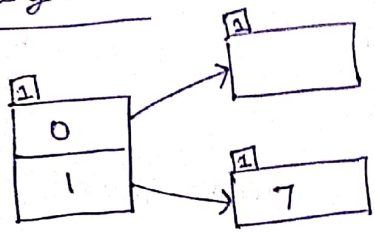
Hash values to binary

$H(x) = x \bmod 6$.

$H(7) = 1 = 001$
$H(8) = 2 = 010$
$H(10) = 4 = 100$
$H(11) = 5 = 101$
$H(15) = 3 = 011$
$H(16) = 4 = 100$

$H(17) = 5 = 101$
$H(19) = 1 = 001$
$H(20) = 2 = 010$
$H(33) = 3 = 011$
$H(53) = 5 = 101$
$H(58) = 4 = 100$

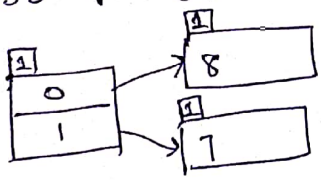
Inserting 7



∴ 1 LSB of $H(7)$ is 1.

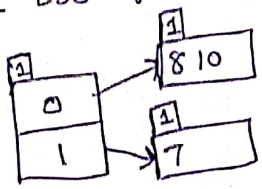
Inserting 8

1 LSB of $H(8) = 0$



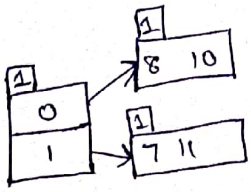
Inserting 10

1 LSB of $H(10) = 0$



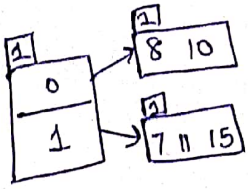
Inserting 11

1 LSB of $H(11) = 1$



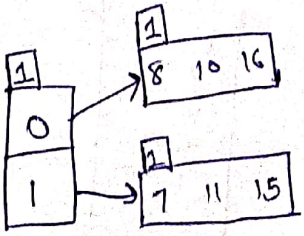
Inserting 15

1 LSB of $H(15) = 1$



Inserting 16

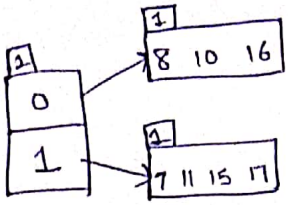
1 LSB of $H(16) = 0$



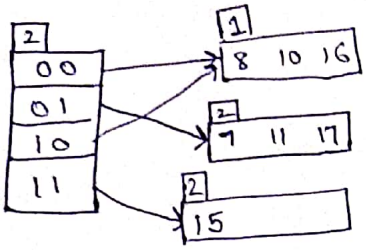
Inserting 17

1 LSB of H(17) = 1.

overflow condition ⇒



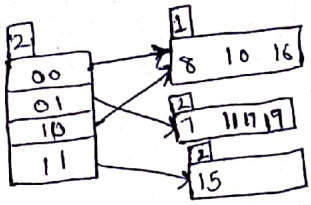
since local depth(1) = global depth(1), we expand bucket & directory.
we rehash the bucket values in directory 1.
which are [01, 01, 11, 01] respectively.



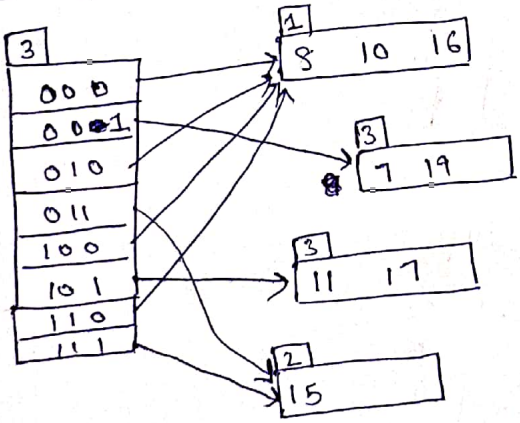
Inserting 19

2 LSB of H(19) = 01.

overflow condition ⇒



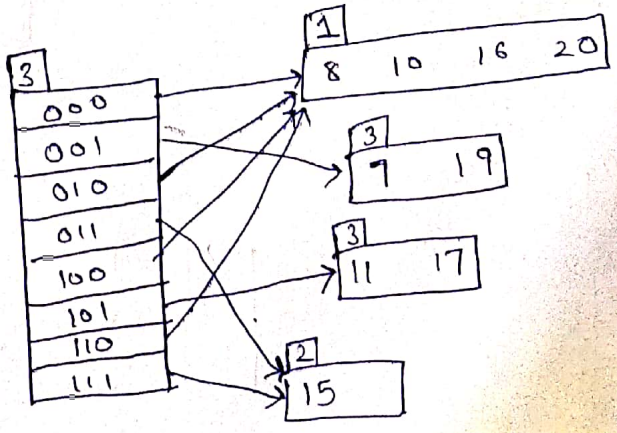
since local depth(2) = global depth(2). we expand bucket & directory.
we rehash the bucket values in directory 01 which are
[001, 101, 101, 001]



Inserting 20

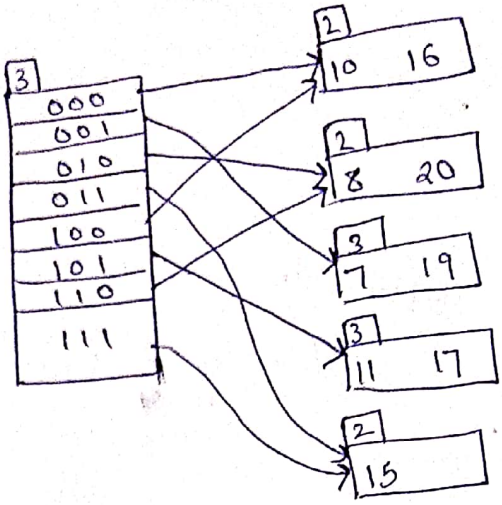
1 LSB of H(20) = 0.

overflow condition ⇒



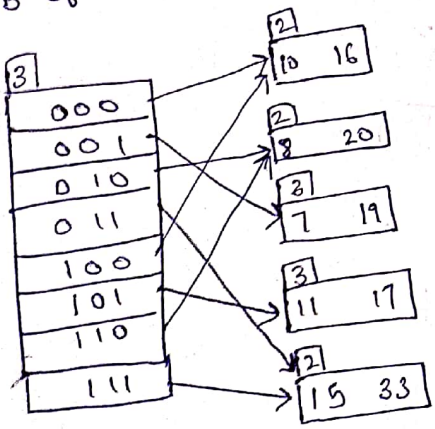
since local depth (2) < global depth (3). we expand only bucket. we rehash bucket values in directory (0) which are

[10, 00, 00, 10]



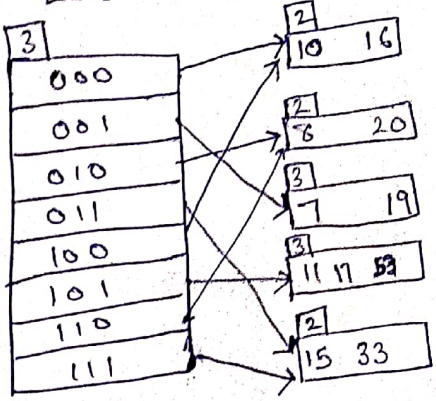
Inserting 33

3 LSB of H(33) = 011



Inserting 53

3 LSB of H(53) = 101



Inserting 58

3 LSB of H(58) = 100.