Short Note's Answer

Design & Analysis of Algorithm

Greedy - The characteristic of greedy method is basically optimization problem generally of type inequality in nature. The component of greedy method are.

i) The set of element like node edge in a graph.

ii) A set of element which have already been used.

iii) To test whether a given set of element provide a soln. or not. How we the soln may be obtained.

iv) A selection func. that pick up some element which have not yet been used.

v) An obj. func. which associate a value to a solution.

* Feasible Region - The set of all possible soln. of an optimization problem is known as feasible region.

* Obj. func. - It is a func. associated with an optimization problem which determines how good a soln. is in general term.

* Feasible soln - Any element of a feasible region of an optimization problem is a feasible soln.

* Knapsack:

Solve this problem with the following condition

a) maximize $P = \sum_{i=1}^{n} a_i x_i$, $0 \le x_i \le 1$

b) weight of the $\sum_{i=1}^{n} g_i x_i$

Ex / Sol: Now consider the given condition.

Total weight (M) = 15

Total no. of elements = 7

weight $(g_1 \ldots g_7) = (2, 3, 5, 7, 1, 4, 1)$

Profit $(a_1 \ldots a_7) = (10, 5, 15, 7, 6, 18, 3)$

So, at first calculate,

$\frac{\text{Profit}}{\text{weight}}$

$\frac{a_1}{g_1} = \frac{10}{2} = 5$

$\frac{a_2}{g_2} = \frac{5}{3} = 1.67$

$= \frac{15}{5} = 3$

$= \frac{7}{7} = 1$

$= \frac{6}{1} = 6$

$= \frac{18}{4} = 4.5$

$= \frac{3}{3} = 3$

Max = 6. at 5

So $x_5 = 1$.

$\begin{cases} x_5 g_5 = 1 \\ a_5 x_5 = 6 \end{cases}$

So, $M_1 = M - f_5 x_5$

$= 15 - 1 * 1$

$= 14$

Next highest : 5 at 1

if $x_1 = 1$

Then, $f_1 x_1 = 2 * 1 = 2$

& $a_1 x_1 = 10$

Remaining wt. $M_2 = 14 - 2$

$M_2 = 8.12$

Next highest = 4.5

if $x_6 = 1$

Now, $f_6 x_6 = 4$

$a_6 x_6 = 18.$

Rem. wt. = 12 - 4

$M_3 = 8$

Next : 3 a

at x_3 & x_7.

Pick $x_7 = 1.$

$f_7 x_7 = 1$

$a_7 x_7 = 3$

Rem. wt, $M_4 = 7$

Next = 3

$x_3 = 1.$

$f_3 x_3 = 5$

$a_3 x_3 = 15$

Rem. wt., $M_5 = 7 - 5$

$= 2$

Next. = 1.67

$x_2 = 1$

$f_2 x_2 = 3$

$a_2 x_3 = 5$

It is exceeding 2, so we can take $x_4 = 1$.

(0 & 2)

So, $x_2 = \frac{2}{3}$

$z_2 x_2 = 3 \times \frac{2}{3}$ (To make two wings)

$= 2.$

$a_2 x_2 = 5 \times \frac{2}{3}$

$= \frac{10}{3} = 3.33.$

$M6 = 2 - 2$

$= \underline{0}.$

Max. profit = $a_5 x_5 + a_1 x_1 + a_6 x_6 + a_3 x_3 + a_4 x_4 + a_2 x_2$

$= 6 + 10 + 18 + 3 + 15 + 3.33$

$= \underline{55.33.}$

Algo-

S1. Take a function knapsack (int M, int z[], int a[], int n)
// M ← max. wt.
// z ← array of weight of each element
// a ← array of profit
// n ← no. of elements.

S1.1 We take an array
ratio[] of size n ← to store the ratio of profit/weight
& a bool visited[] to store which are in result.

S1.2 Take a loop to calculate -
for (i ← 0; i < n; i++) // to iterate from 0 to n.

S1.2.1 ratio[i] ← profit[i] / weight;
// storing ratio as the profit & weight

S1.3 Take 2 array ax ← to store axis & zx ← to store zixi

S1.4 Now we loop from 0 to n
for (i = 0; i < n; i++)

S1.4.1 calculate the max. of element in array ratio[].

S1.4.2 int y = max(ratio[], n) // func. to calculate the max.

S1.4.3. ~~and bool~~ = check if the weight not exceeds the left over M

S1.4.4 if (z[y] <= M)

then

S1.4.4.1 ax[i] = a[y] * r;
zx[i] = z[y]
visited[y] = true;

else

S1.4.4.1 to calculate the max. possible at.

Sl. 4.4.2 M. : 11/3/24

// Job sequencing problem - (Algo also)

For the prob. problem let n = 3 (3 jobs) & the profit & deadline are..

P = (10, 50, 25) & D = (2, 2, 1)

max.
When deadline = 2, we can execute 2 jobs as per execution for each job is 1 unit.

Need to execute one by one maximizing profit.

Step 1: To get max. profit by combining 2 jobs.

j_1j_2 = 10 + 50 = 60

j_1j_3 = 35

j_2j_3 = 75

j_1 = 10

j_2 = 50

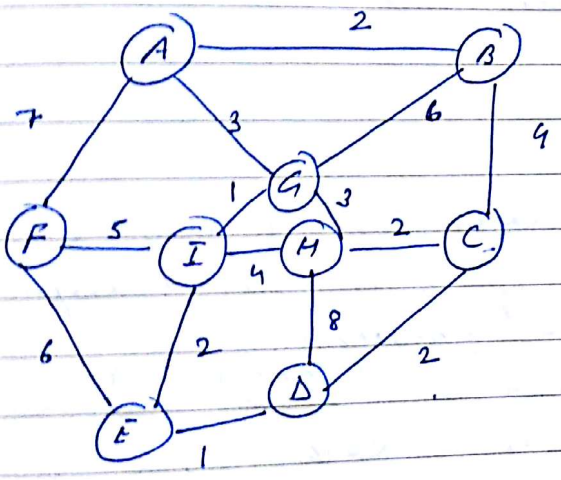
j_3 = 25

Eg. max. profit - (Descending order)

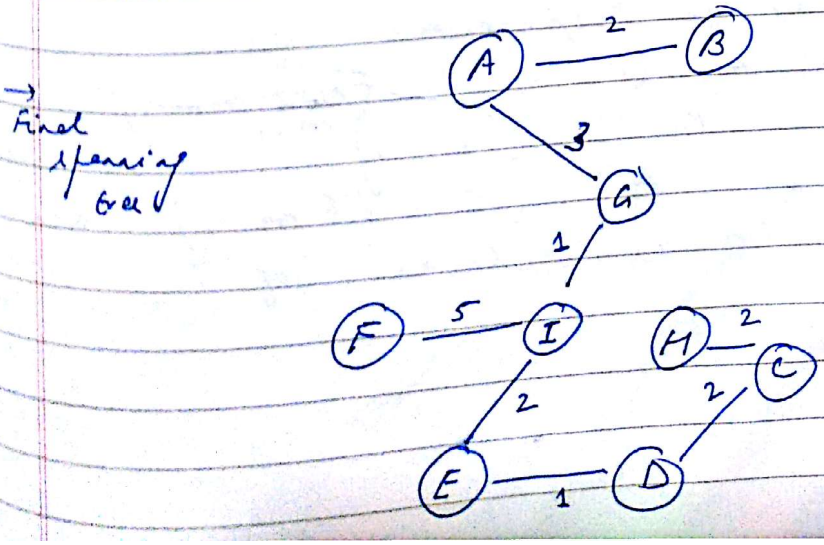
$j_1 j_3 = 75$
$j_1 j_2 = 60$
$j_2 = 50$
$j_1 j_3 = 35$
$j_3 = 25$
$j_1 = 10$

Tree
Spanning Tree → Prim
→ Kruskal

Prim's algorithm - (steps also)



start from A,
move to min. i.e. A to B , Now A to G.



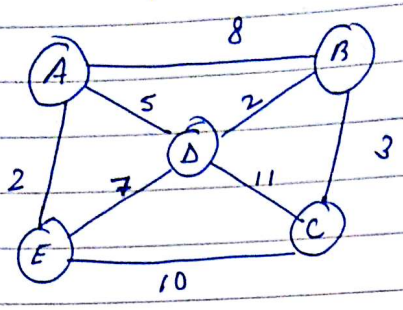
Final spanning tree

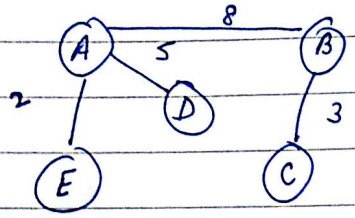
A B
A B G.
Add G-I = 1.
Add I to E = 2.
Add E to D = 2

(CT2)

Kruskal, BFS, DFS. ✓, Heap sort.

* Kruskal's algo -





(Pick the smaller edge & discard if cycle).

✓ Binary heap & heap sort.

M-way search Tree

⇒ A m-way search tree T may be an empty tree, if T is not empty it satisfy the following property -

i) For some integer m, known as the order of the tree, each node is of degree of which can reach a degree of m. (Each node at most m child node).

ii) A node may be represented as -

$A_0, (K_1, A_1)\ (K_2, A_2) \ldots (K_{m-1}, A_{m-1})$

where $K_i \ (1 \le i \le m-1)$, K_i are the keys & $A_i \ (0 \le i \le m-1)$ are pointer to the subtree of T.

Algo -

Imp. (M-way & B-Tree → Theory also - Insertion, Deletion, Search)

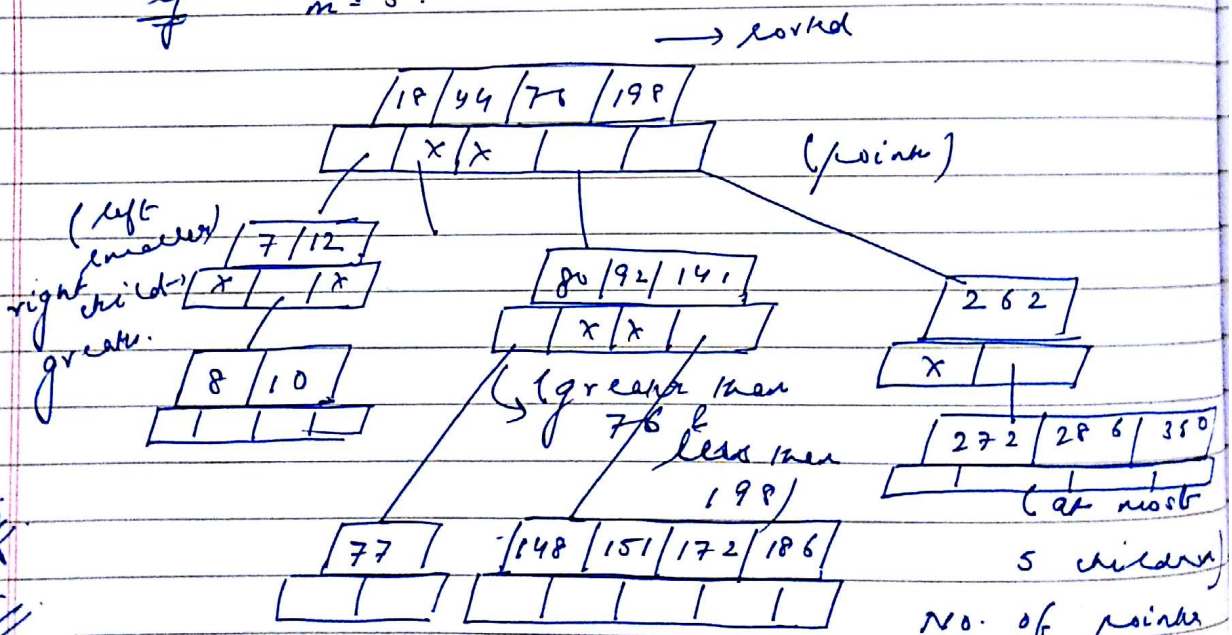
* M-way -

→ If a node has k-children where, $k \leq m$, then the node may have all keys (k-1 keys, $k_1, k_2 \ldots k_{k-1}$) contain in such that $k_1 < k_2 \ldots < k_{i+1}$.

→ For a node Ai, all the key values in the subtree pointed by Ai are less than the keys k_{i+1}. & all the keys value in subtree pointed by Am is greater by k_{m-1}.

→ Each subtree Ai, $0 \leq i \leq m-1$ are also m-way search tree.

eg $m = 5$.



No. of pointers = 5
& keys = 4

Algo.

Ex am

* Search - same as binary ST.

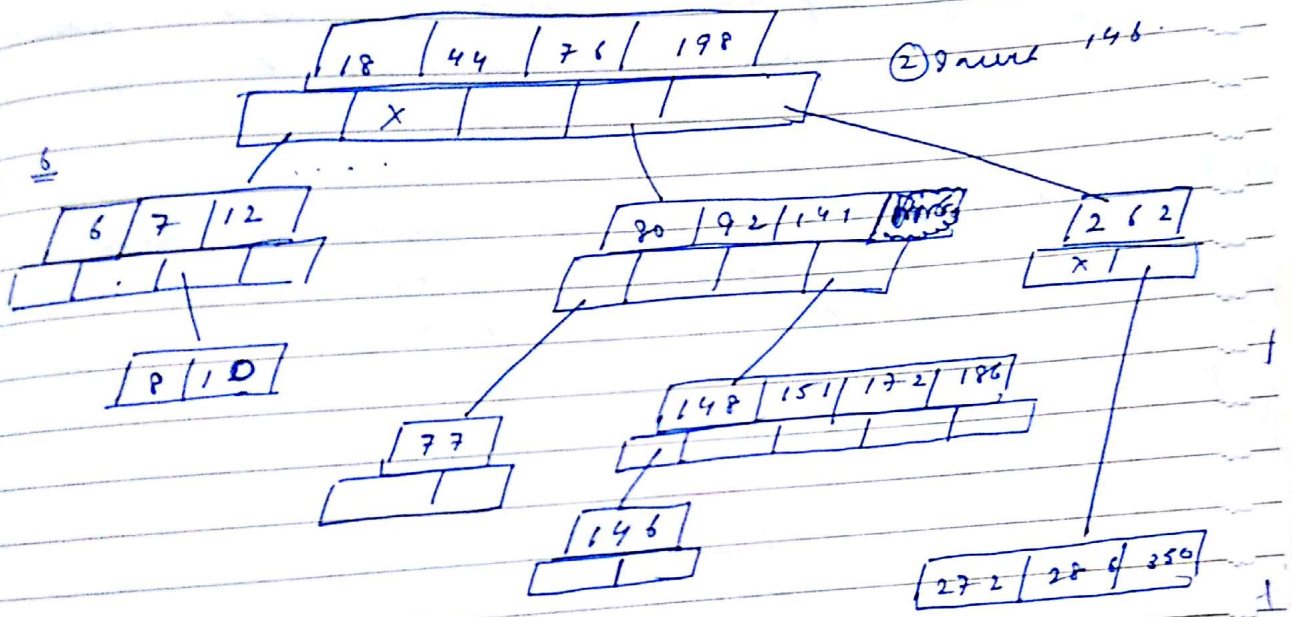
* Insertion -

→ Search for the given key value in tree until we reach the node which should contain the value. (search right pos.)

→ If the node is not yet full, then given key is put in the appropriate position. & do related work.

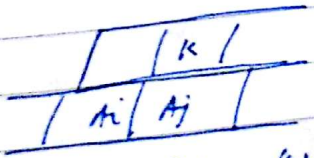
→ If the node is full, create a new child of it at the apt. position & keep the key where.

* Until



To delete 262, delete 272. Replace 262 by 272

Deletion -



Let K be the key to be deleted from m-way ST.

→ If Ai = Aj = NULL, delete K.

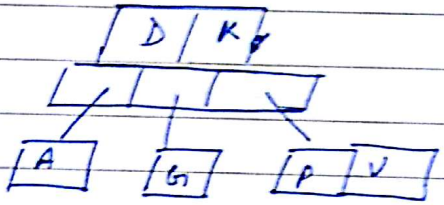
→ If Ai ≠ NULL & Aj = NULL, then, choose the largest of any element K' in the child node pointed to by Ai, delete the delete K' & replace K by K'. Obviously deletion of K' may call for subsequent replacement & deletion.

in the same manner.

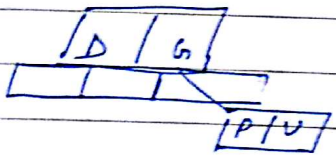
→ If A_i = NULL & $A_j \neq$ NULL, then there is no element of the key element k'' from the subtree pointed by A_j. Delete k'' & replace k by k''. Subsequent deletion may be done.

→ If $A_i \neq$ NULL & $A_j \neq$ NULL, then either replace by k' or k''.

Q. A 3-way ST constructed from an empty ST. key value are
D K P V A G.

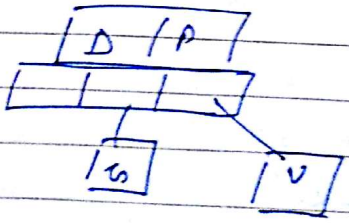


Delete A & K.

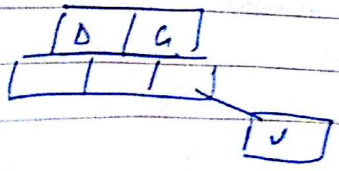


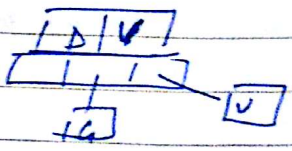
&

→



P deleted -





B-Tree → A B-Tree of order (m). if not empty, is an m-way search tree in which,

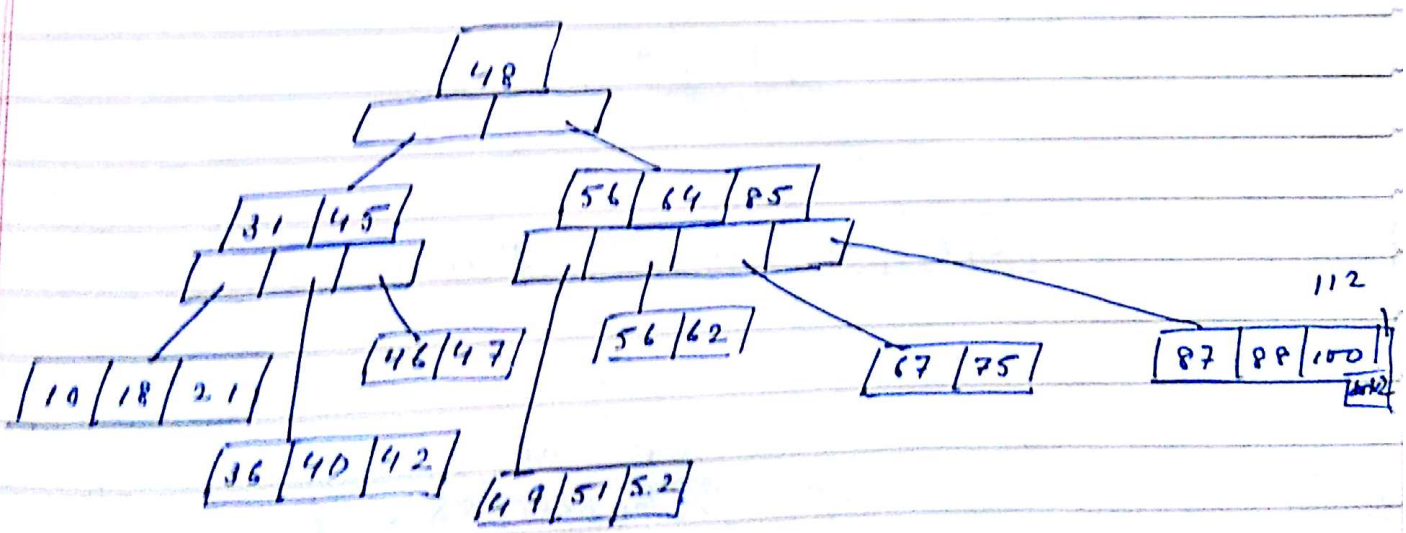
i) The root has at least 2 child nodes & at most m child node.

ii) The internal node, except the root have at least $m/2$ child node, at most m child node.

iii) The no. of keys in each internal node one less than the no. of child node & this keys partitioned the keys in the subtree of a node in a similar manner to that m-way search tree.

iv) All leaf node are on the same level.

Eg: B-Tree of order 5 →



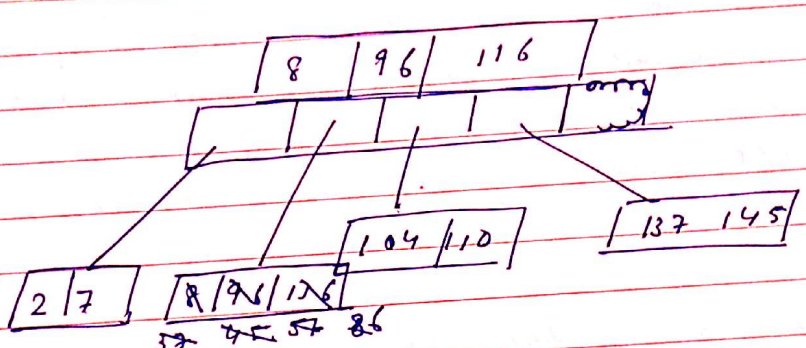
Search similar to m way as BST.

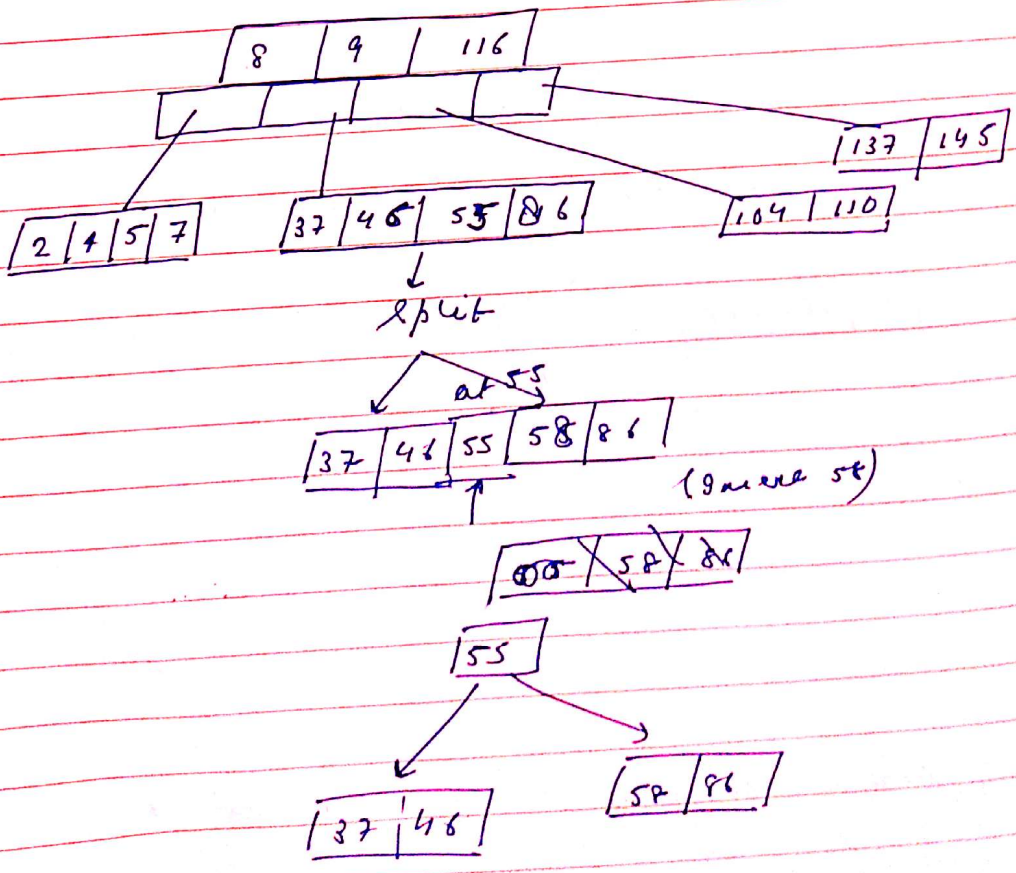
Deletion

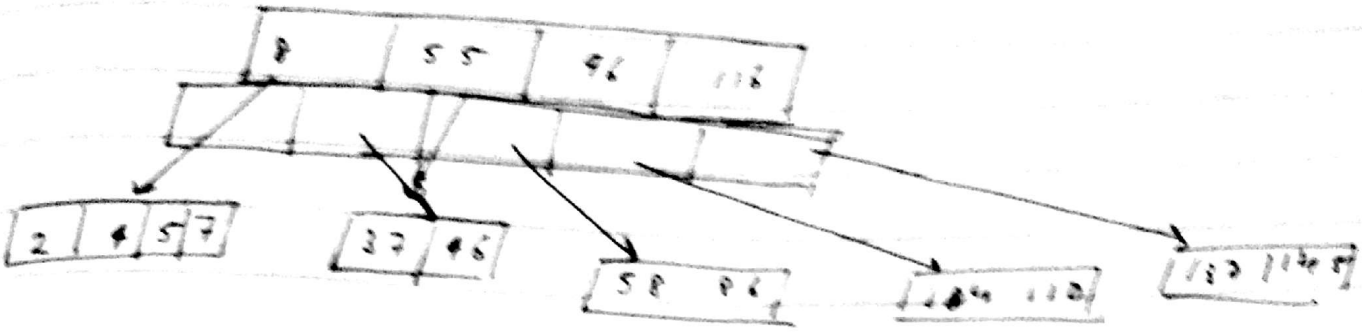
* Insertion - same way as m-way search tree except when the node is full it is to be split at the middle position. Repeat until a node is encountered which is underflowed & therefore can accomodate a new node

eg: Consider a B-Tree of order 5. The root contains 3 keys, 8, 96, 116. The left child contains 2, 7.

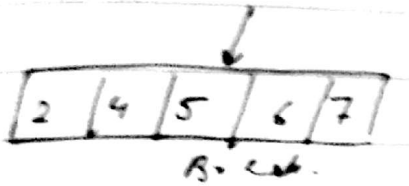
→ Insert 45, 58, 6



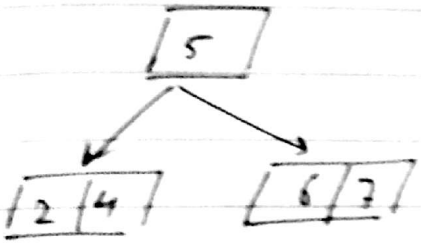




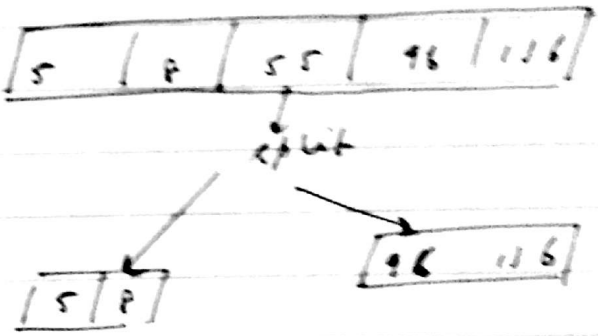
Insert 6 →

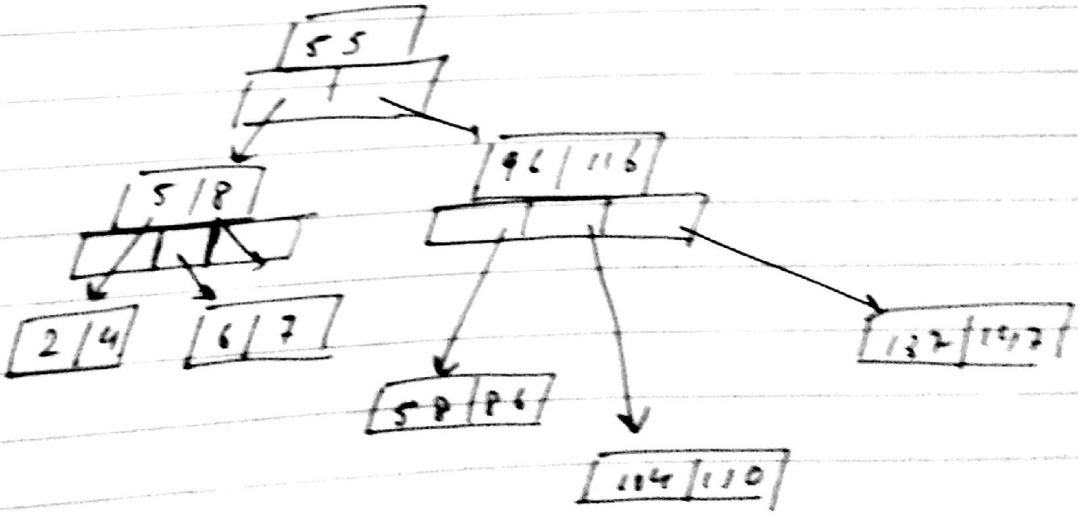


B. ext.



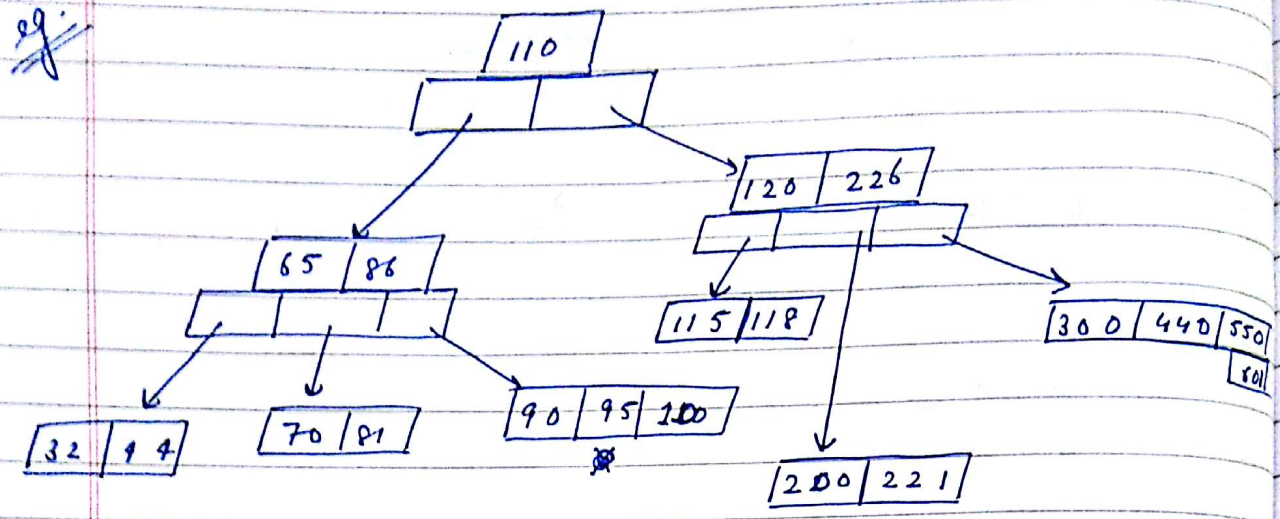
place 5 at root.



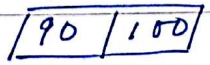


Deletion - reverse of Insertion.

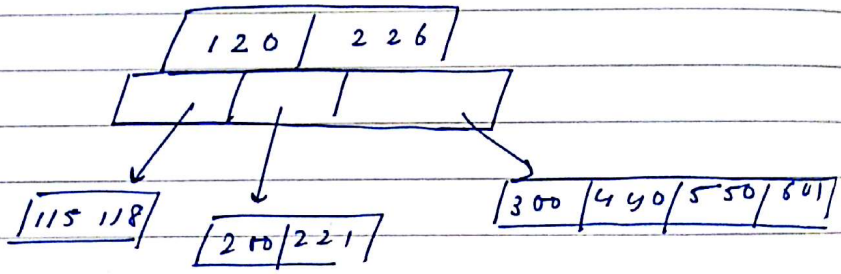
eg:

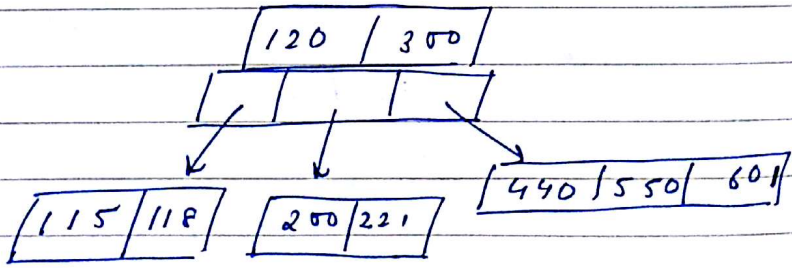


* Delete 95.

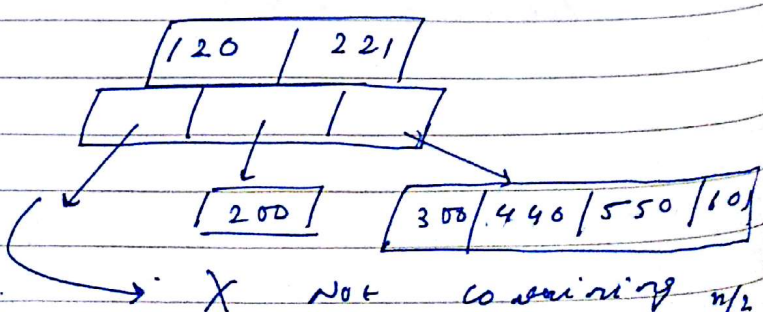


* Delete 226.



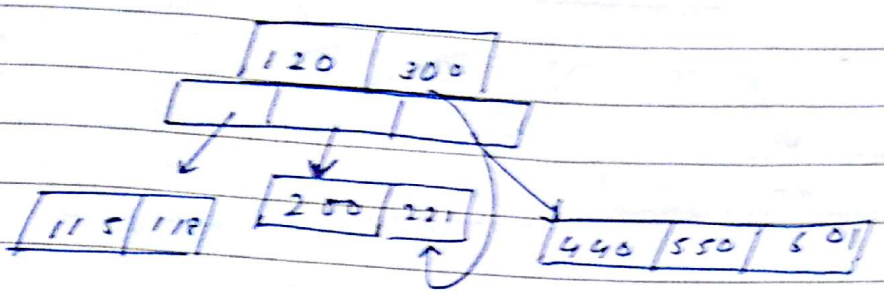


OR

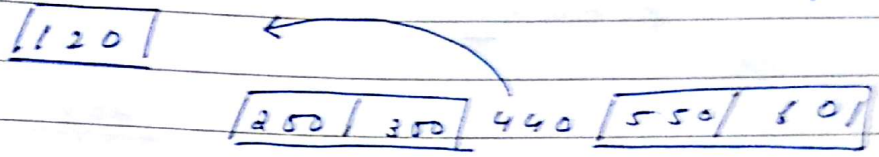


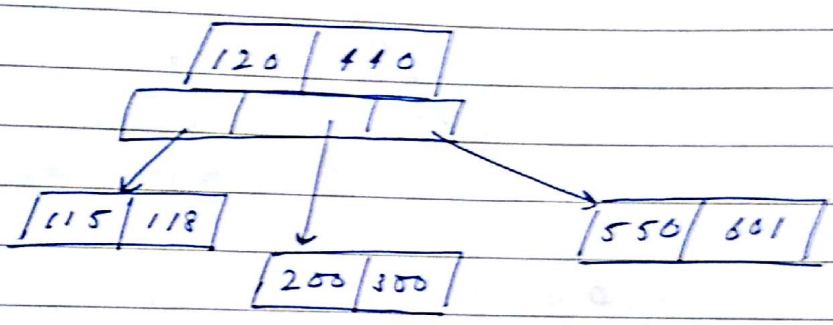
X Not covering m/2 child.

* Delete 221 →

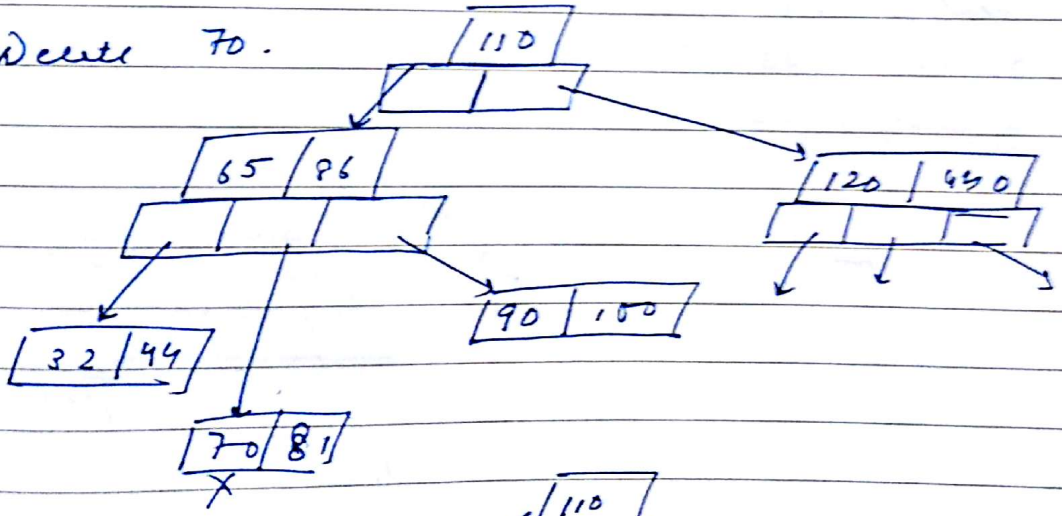


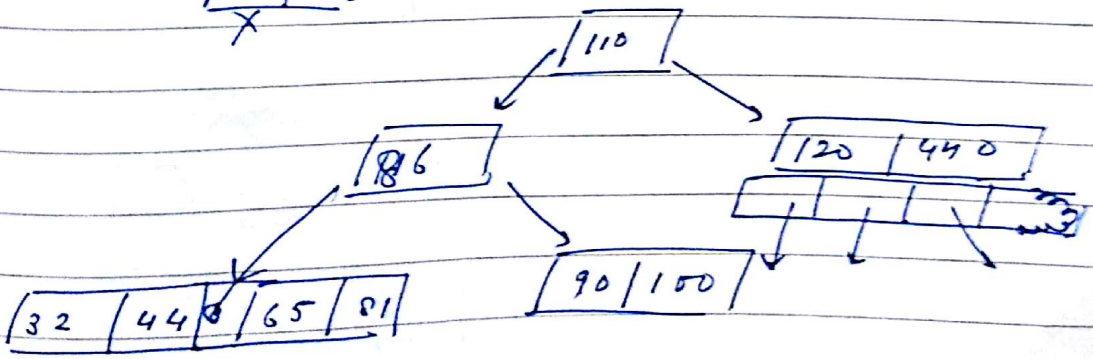
Node not m/l keys.

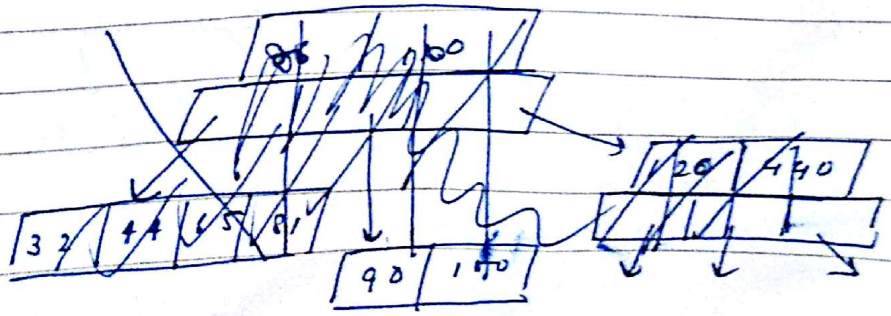




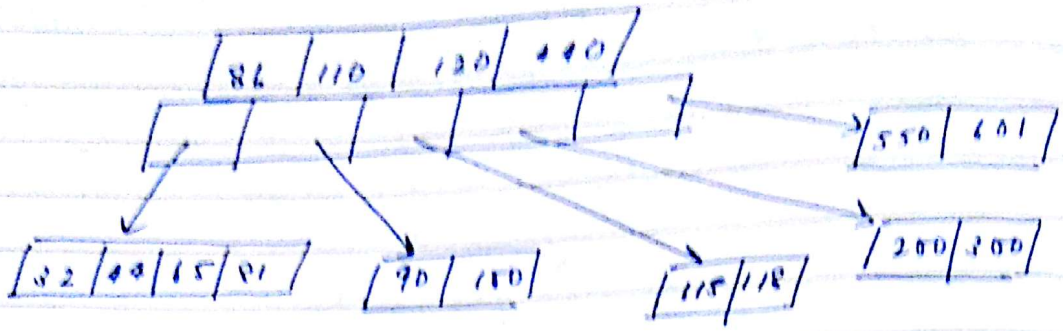
* Delete 70.



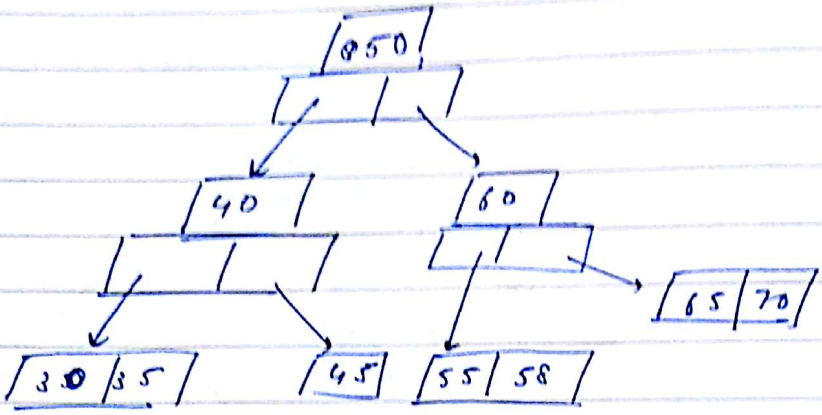




(merge to root)



Eg. Order - 3

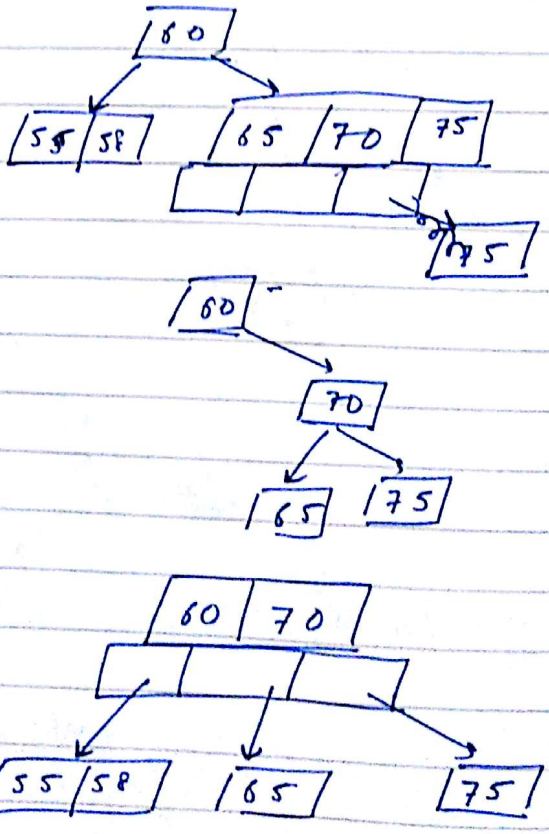


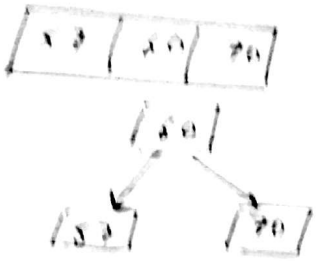
Perform the op.

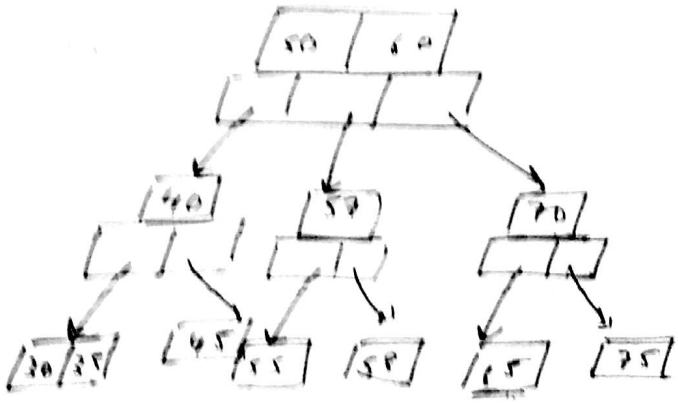
→ Insert 75, 57.

→ Delete 35, 65

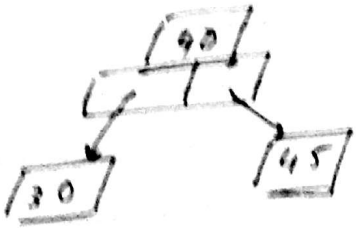
75







2. Delete 35.



4. Delete 65 →