 Name - Chayan Kumar Sengupta, Roll - 561, Code - CMSM4121 13.1.2024

GR - A

1 (i) "Quick sort is not always quick for all input instances"

→ Let $C(n)$ be the no of comparisons of keys by quick sort when applied to a list of length n and $S(n)$ be the no of swaps.

we have $C(1) = C(0) = 0$.

function partition( ) accounts for exactly $n-1$ key comparisons. If one sublist has length r, other one will have length $n-r-1$. Then no of comparisons done will be $C(r)$ and $C(n-r-1)$ respectively.

$$\therefore C(n) = n-1 + C(r) + C(n-r-1)$$

Now depending on the ordering of the entries diff cases arise.

(a) If we consider worst case for comparisons, we have seen that it occurs when pivot fails to split the list at all.

$\therefore$ we obtain, $C(n) = n-1 + C(n-1)$ $[\because C(0) = 0]$

after solving this recurrence relation, we get,

$$C(n) = n-1 + C(n-1) = (n-1) + (n-2) + \cdots 2 + 1$$

$$= \tfrac{1}{2}(n-1)n = \tfrac{1}{2}n^2 - \tfrac{1}{2}n$$

$\therefore$ In its worst case, quick sort is $= O(n^2)$.

as bad as the worst case of selection sort.

[selection sort also makes $\tfrac{1}{2}n^2 - \tfrac{1}{2}n$ comparisons]

(b) If we consider worst case for swaps, partition( ) makes $(n+1)$ swaps.

$$\therefore S(n) = n+1 + S(n-1) \quad ; \quad n \geq 2$$

$$= 3 \quad ; \quad n = 2$$

$\therefore$ If we solve this recurrence relation, we get.

$$= \tfrac{1}{2}(n+1)(n+2) - 3 = 0.5n^2 + 1.5n - 1 \text{ swaps}$$

$\therefore$ In it's worst case, contiguous insertion sort must make about twice as many comparisons and assignments of entries as it does in it's any case giving a total of $0.5n^2 + O(n)$ for each operation. Each swap in quick sort requires three assignments, so quick sort in it's worst-case does $1.5n^2 + O(n)$ assignments. Hence in it's worst case quick sort is worse than the poor aspect of insertion

sort and in regard to key comparisons, it's also bad as the poor aspect of selection sort. Indeed, in worst cases, quick sort is a disaster and it's name is nothing less than false advertising.

∴ quicksort is not always quick ← This statement is true for some instances.

(ii) "Dynamic programming algo avoid all duplicate sub instances of a problem that are otherwise generated by divide and conquer"

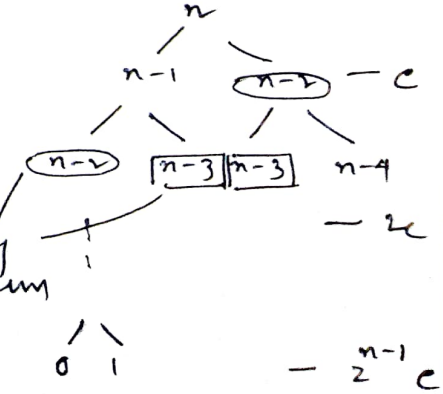
→ It sometimes happens that natural way of dividing an instance suggested by divide and conquer algo leads us to consider several overlapping subinstances. If we solve each of these independently, they will in turn create a host of identical subinstances. If we pay no attention to this duplication, we are likely to end up with an inefficient algo on the other hand, we take advantage of the duplication and arrange to solve each subinstance only once, saving the sol^n for later use then more efficient algo will result. The underlying idea of dynamic programming quite simple. It avoids calculating the same thing twice, usually by keeping a table of known results that fills up as subinstances are solved.

eg: $fib(n) = \begin{cases} n & ; \text{if } n = 0 \text{ or } 1 \\ fib(n-1) + fib(n-2) & ; n \geq 2 \end{cases}$

$$T(n) = \begin{cases} T(n-1) + T(n-2) + c & ; n \geq 2 \\ 1 & ; n \leq 1 \end{cases}$$

∴ $T(n) = O(2^n)$

which is exponential time.



$$\text{sum} = c\left(2^0 + 2^1 + \cdots 2^{n-1}\right)$$

* All dynamic programming problems have two characteristics in common.

 Name - Chayan Kumar Sengupta, Roll - 561, Code - CHSH4121 13.1.2026

① optimal substructure

A larger problem is being solved by taking the help of smaller problems of same kind.

② overlapping subproblems

There are subproblems that are being repeatedly solved in the recursion tree.
so there are actually 2^n ways.

① Top-down (memorization)

f[] ← array in which we'll store fib[i],
f[0] = 0 ; f[1] = 1 ; f[i] = NIL if i > 1.

```
int fib(n)
{   if f[n] == NIL do
        f[n] = fib(n-1) + fib(n-2);
    return f[n];
}
```

② Bottom-up (tabulation)

f[] ← array to store values
f[0] = 0 ; f[1] = 1 ; f[i] = NIL if i > 1

```
int fib(n)
{   for (i=2; i<=n; i++)
        f[i] = f[i-1] + f[i-2];
    return f[n];
}
```

So from $O(2^n)$, time complexity reduces to $O(n)$ which is huge improvement.

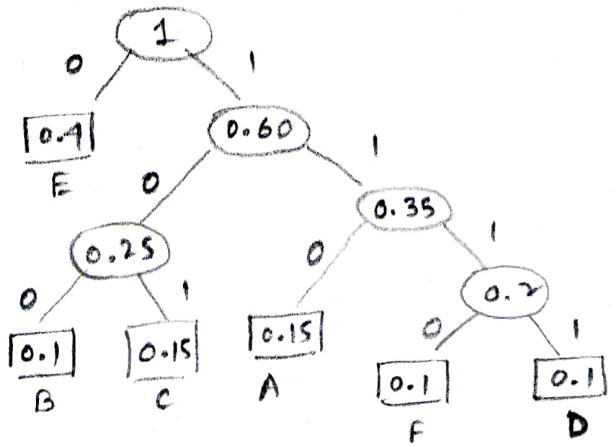
∴ The aforementioned statement is true.

Q

A - 0.15
B - 0.1
C - 0.15
D - 0.1
E - 0.4
F - 0.1

weights

0.1 (F) 0.1 (D) 0.1 (B) 0.15 (C) 0.15 (A) 0.4 (E)
0.2
0.25
0.35
0.60



Huffman tree

A - 110
B - 100
C - 101
D - 1111
E - 0
F - 1110

Huffman codes

The algo that has been used to solve is mentioned below,

```
Huffman (C)
{
    n = |C|
O(n) —— make a min heap Q
        with C
        for i = 1 to n-1
        do
            allocate a new
            node Z
O(log n) —— Z.left = extract_min
            (Q)
O(log n) —— Z.right = extract_min
            (Q)
O(log n) —— Insert (Q, Z);
done        return extract_min
            (Q)
}
```

(iii)

The immediate decodability of huffman code is clear. each character is associated with a leaf node in the huffman tree and there's a unique path from the root of the tree to each character. So no sequence of bits comprising the code for some character can be a prefix of a longer seq of bits for other character.

for ex:

In the prev ex we got

A - 110 , B - 100 , C - 101 , D - 1111 , E - 0 ,

F - 1110.

Let's say a message looks like : E F C D B A

E F C D B A

(0) (1110) (101) (1111) (100) (110)

∴ encoded message looks like : 0 1110 101 1111 100 110

Now while decoding,

when encounters 0 it returns E as no other starts with 0

for next bit. if it's 1 so possibilities are (F, C, D, B, A). checks

possibilities reduce to (B, C) and if 1 then (F, D, A)

and this goes till the length of the message is exhausted.

weighted external path length for prev ex is

$0.15 \times 3 + 0.1 \times 3 + 0.15 \times 3 + 0.1 \times 4 + 0.4 \times 1$

$+ 0.1 \times 4$

$= 0.45 + 0.3 + 0.45 + 0.4 + 0.4 + 0.4$

$= 2.40$ CP which is minimum.

 Name - Chayan kumar Sengupta, Roll-561, Code - CMSM4124 13.1.202

3.

$n = 3$

$W = 20$

$(v_1, v_2, v_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$

	obj1	obj2	obj3
v	25	24	15
w	18	15	10
v/w	1.39	1.6	1.5

This is an optimization problem. We want max profit.

Therefore we can try for greedy approach.

There could be three cases.

Case - 1 (greedy about profits)

So we will try to fill up the knapsack with higher profitable values.

knapsack :	w	v
obj1	18	25
obj2	2	$24 \times 2/15$
	20	28.2 (profit total)

Case - 2 (greedy about weights)

So we will try to fill up the knapsack with min weights such that we can put more items.

knapsack :	w	v
obj3	10	15
obj 2	10	$24 \times 10/15$
	20	31 (total profit)

Case - 3 (greedy about profit by weight)

We will choose the item which has highest profit/weight value.

	w	v	w/v
obj2	15	24	1.6
obj3	5	$15 \times 5/10$	1.5
	20	31.5 (total profit)	

∴ we can say that case - 3 (greedy about profit/weight) is optimal strategy which gives 31.5 profit (max).

5 ▨ Objective of Travelling salesman problem (TSP)

— This is a problem in graph theory.
— Given n cities which are represented by n vertices, we need to find the minimal path such that all the cities / vertices are visited exactly once starting from the source vertex and returning back to the source vertex.
— overall path length / distance should be minimal.
— we represent cities using vertices and distance between two cities using weight of the connecting edge. Therefore weights of every edges put together should be minimum.
— ex: Mail delivery system, goods delivery.

▨ Greedy approach to solve TSP

This problem can be solved using greedy technique.

steps:

step 1: create two primary data holders:
ⓐ A list that holds the indices of the cities in terms of the i/p matrix of distances between cities.
ⓑ result array which will have all cities that can be displayed.

step-2: Perform traversal on given adjacency matrix dsp[][] for all the city and if the cost of the reaching any city from current city is less than current cost, then update the cost.

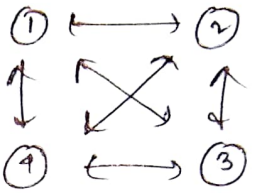
step-3: Generate the min path cycle using the above step and return min cost.

⊙ Time complexity: $O(N^2 * \log_2 N)$
Ⓐ Space " : $O(N)$.
(auxiliary space)

Greedy approach fails to give optimal result:

eg. (cost of distance matrix)

Adjacency matrix for this graph is:



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

So if we apply greedy to solve considering 1 as the starting node.

we get path:

① — ② — ③ — ④ — ①

So total distance gives : 39 unit.
(on cost)

since there exists another path whose cost is minimum then we can say that greedy fails to evaluate TSP.

① — ② — ④ — ③ — ①

which gives distance : 35 unit. (better than greedy)
(on cost)

Suitable algorithm to solve TSP:

~~Let the prob~~

Let the vertex set be $V = \{1, 2, 3, \dots n\}$

If source vertex is 1,

dij represents distance between vertex i and j

$S \subseteq V$; $S = \{2, 3, 4, \dots n\} = V - \{1\}$ (excluding the source vertex)

Recursive soln.

$$TSP(i, S) = \begin{cases} d_{i1} \text{ if } S \neq \phi \text{ (Base case)} \\ \min_{k \in S} \{ d_{ik} + TSP(k, S - \{k\}) \}, \text{ otherwise} \end{cases}$$

(min cost path from ith vertex to every vertex in S and back to source)

Bottom up (tabulation) approach for TSP :

```
function TSP (G, n)
{
    for (k = 2 upto n-1)
        C({k}, k) = d1,k   // cost from source to vertex k is updated.

    for (s = 2 upto n-1)
    do
        for all S ⊆ {2, 3, ... n}, |S| = s)
        do
            for all k ∈ S
            do
                C(S, k) = min [C(S - {k}, m) + dm,k]
                          m ≠ k and m ∈ S

            done
        done
    done
    optimal = min [C({2, 3, ... n}, k) + dk-1]
              k ≠ 1
    return optimal
}
```

 Name - Chayan kumar Sengupta, Roll-561, Code - CHSM4124 13.1.2021

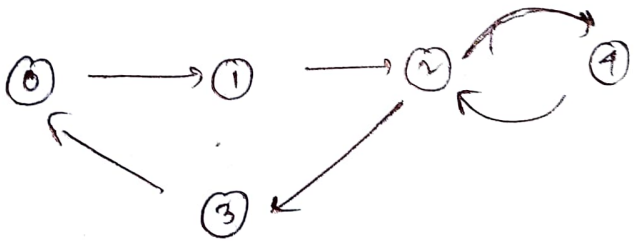
CA - B

6 :

(a) ☑ strongly connected

A graph is said to be strongly connected if every pair of vertices (u, v) in the graph contains a path b/w each other. In an undirected directed graph G, every pair of vertices u and v should have path in each direction b/w them i.e bidirectional path. The elements of the path matrix of such a matrix will contain all 1's

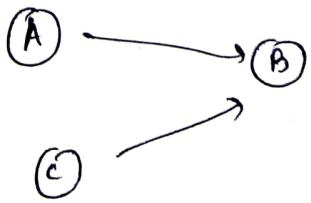
eg :



weakly connected

A graph is said to be weakly connected if there doesn't exist any path b/w two pairs of vertices. Hence, if a graph G doesn't contain a directed path from u to v or from v to u for every pair of vertices (u, v) Then it's weakly connected. The elements of such a path matrix of this graph would be random.

eg :



☑ Bipartite graph

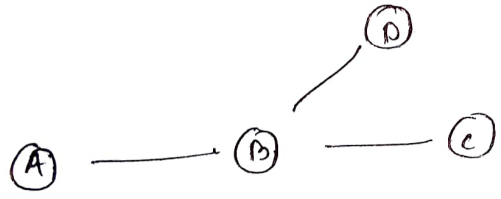
A graph whose vertices can be divided into two independent sets U and V such that every edge (u, v) either connects a vertex from u to v or from v to u. No edge is there that connects vertices of same set.

(c) ☑ Articulation point

(1) A vertex v in graph G with c connected components is an articulation point if it's removal increases the no. of connected components of G.

(2) If c' → no of connected components after removing vertex v, if c' > c then v is articulation point.

(3) ex:



Here B is articulation point.

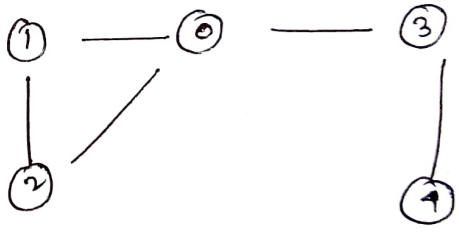
(4) A graph has articulation point doesn't imply it has bridge

Bridge

(1) An edge (u,v) in a graph G with c connected components is a bridge if it's removal increases the no of connected components of G.

(2) If c' → no of connected components after removing edge (u,v), if c' > c, then (u,v) is bridge.

(3) ex:



Here (0,3) and (3,4) are bridges

(4) A graph has bridge implies, it has articulation point.

☑ Pendant vertex

A vertex of a graph is said to be pendant if it's neighbourhood contains exactly one vertex. In the context of trees, pendant vertex usually known as terminal node / leaf node.

Name - Chayan kumar Sengupta, Roll - 561, Code - CHSH4124 13.1.2021

7

(a) AVL Tree with examples of Rotations

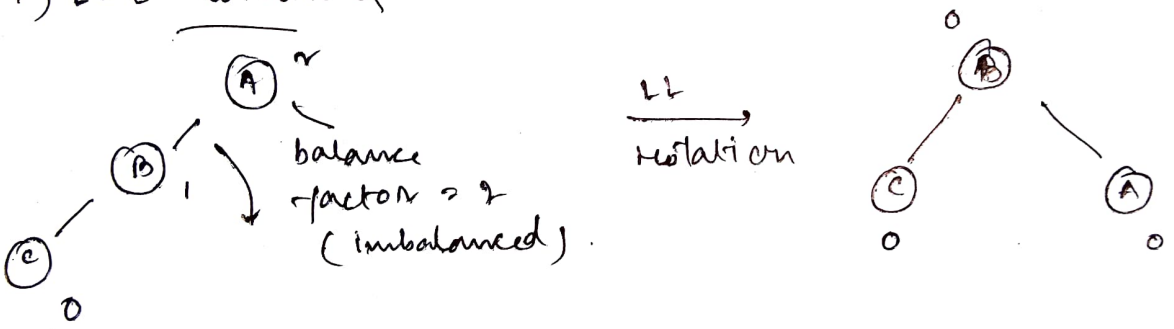
In computer science, an avl tree is self balancing binary search tree. In an avl tree, the heights of the two child subtrees of any node differ by at most one. If at any point they differ by more than one, rebalancing is done to restore this property.

There are 4 types of rotations present to make a height imbalanced tree, a height balanced tree.

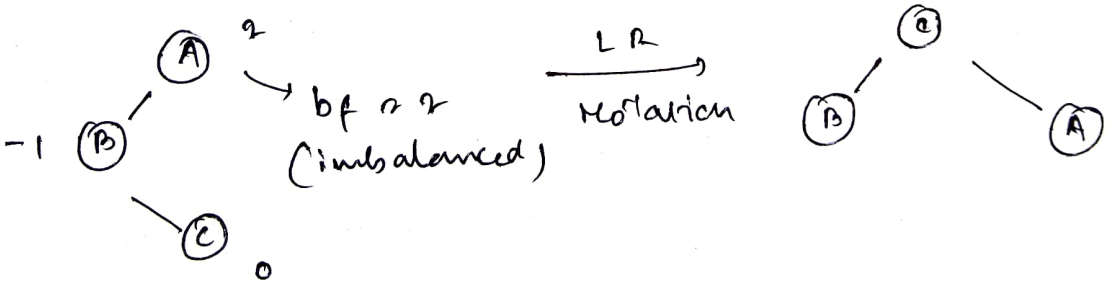
1) L-L Rotation 2) L-R Rotation.

3) R-L " 4) R-R "

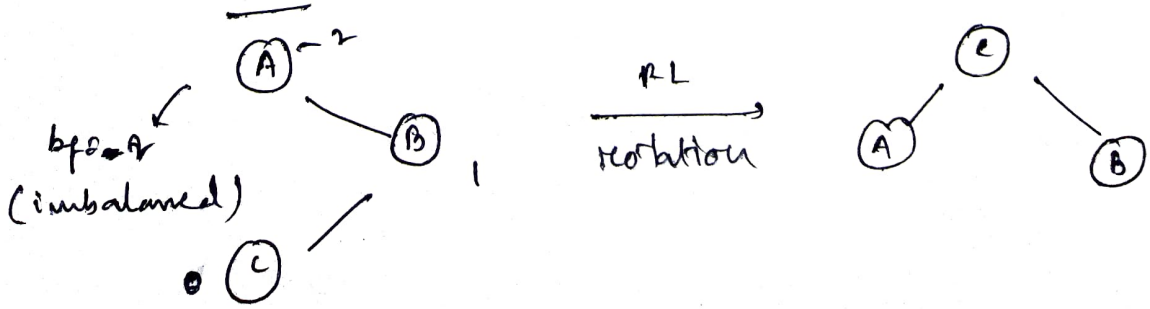
1) L-L Rotation



2) L-R Rotation

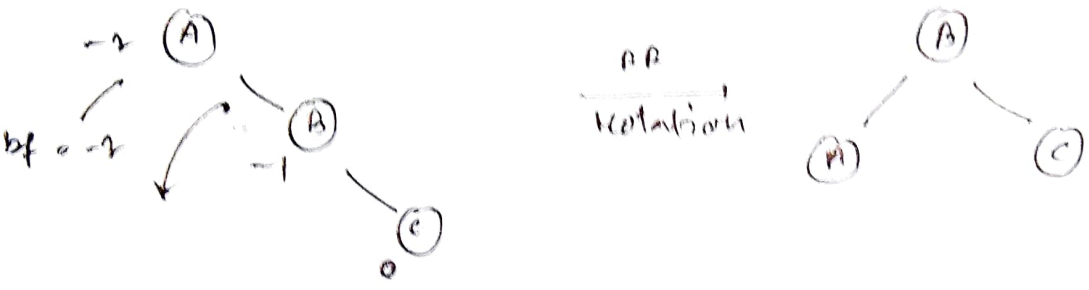


3) R-L Rotation



Pg - 17

1) R - R rotation

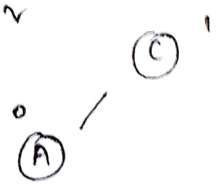


Given :

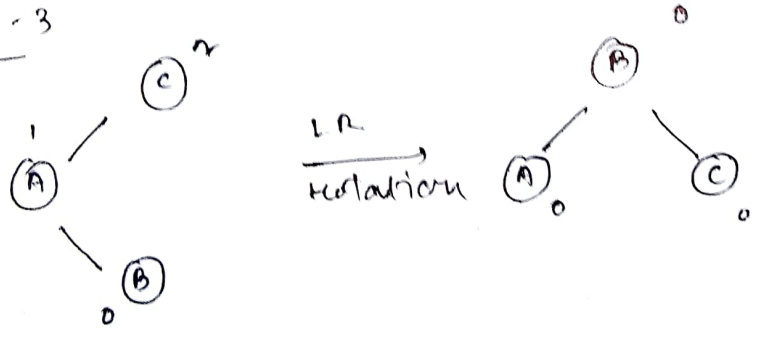
C, A, B, D, G, E, P, R, V, T, Z.

step - 1

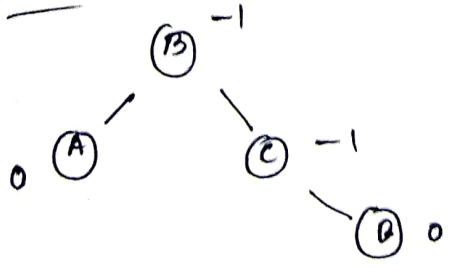
step - 2



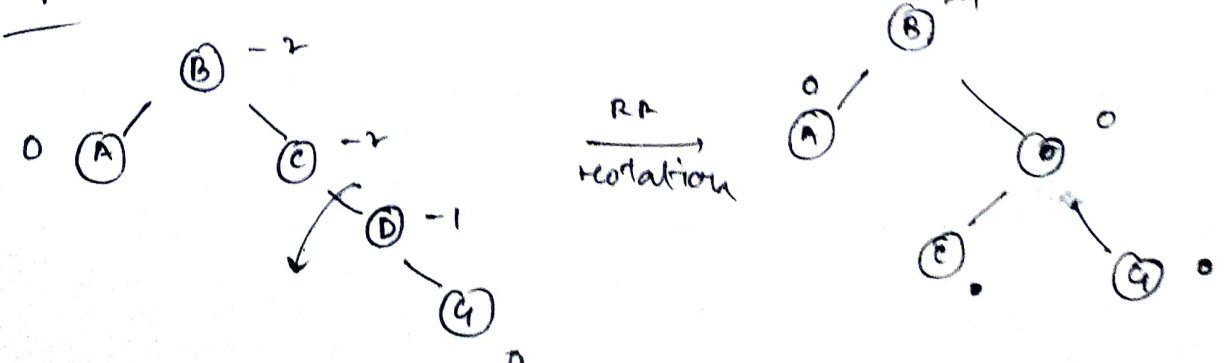
step - 3



step - 4

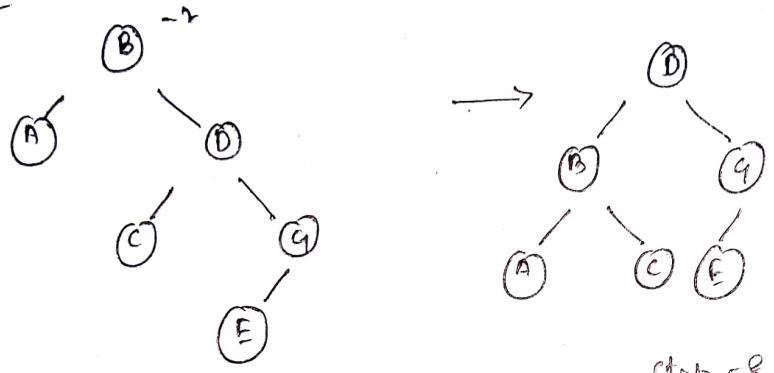


step - 5

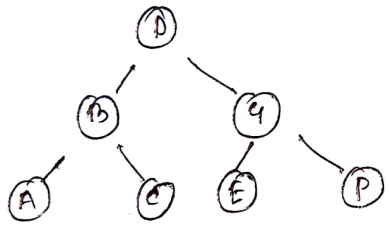


 Name- Chayan Kumar Sengupta, Roll-561, Code-CMSM9/21 13.1.21

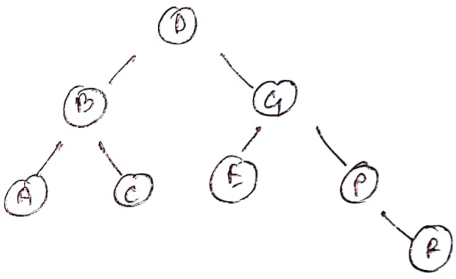
Step-6



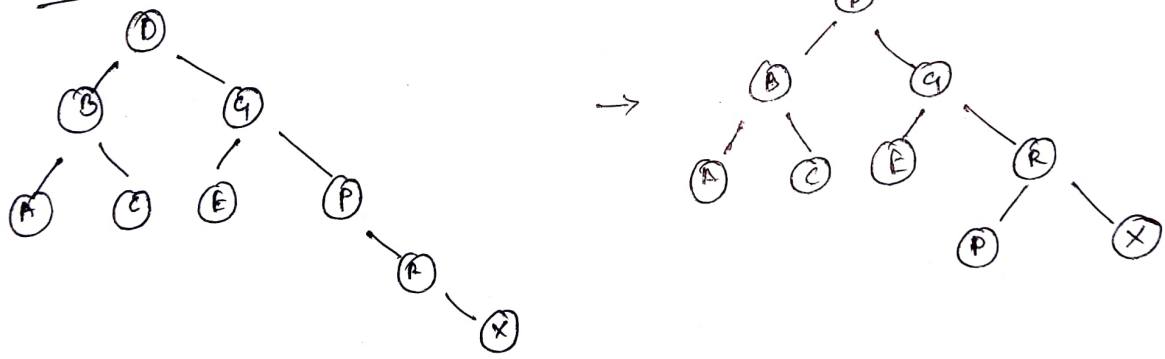
Step-7



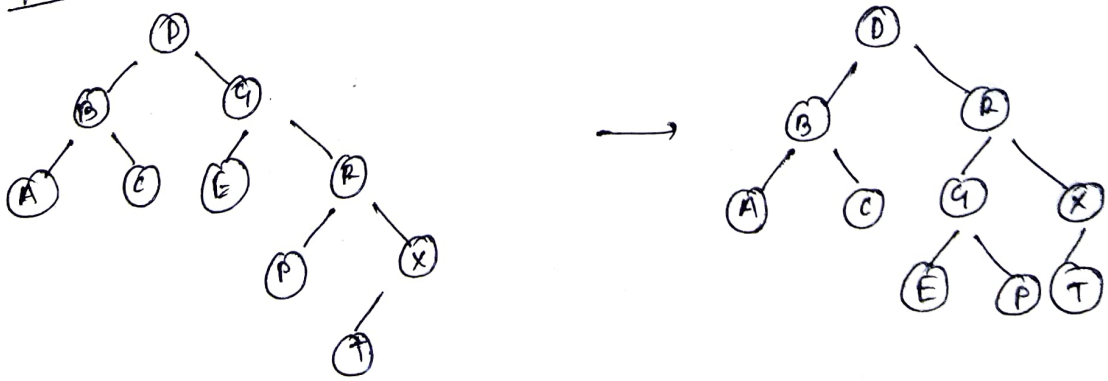
Step-8



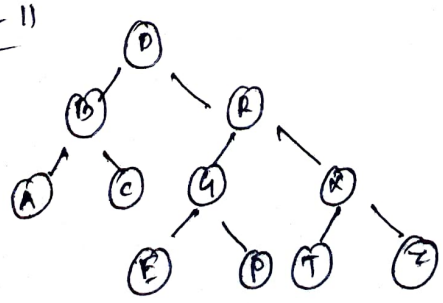
Step-9



Step-10



Step-11

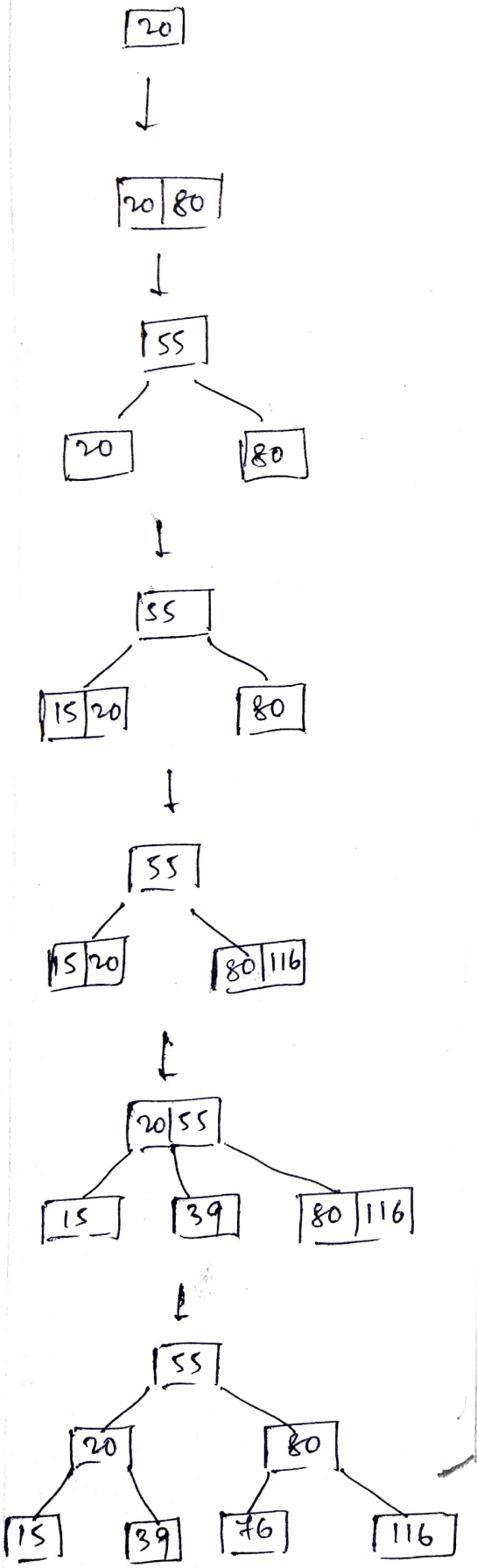


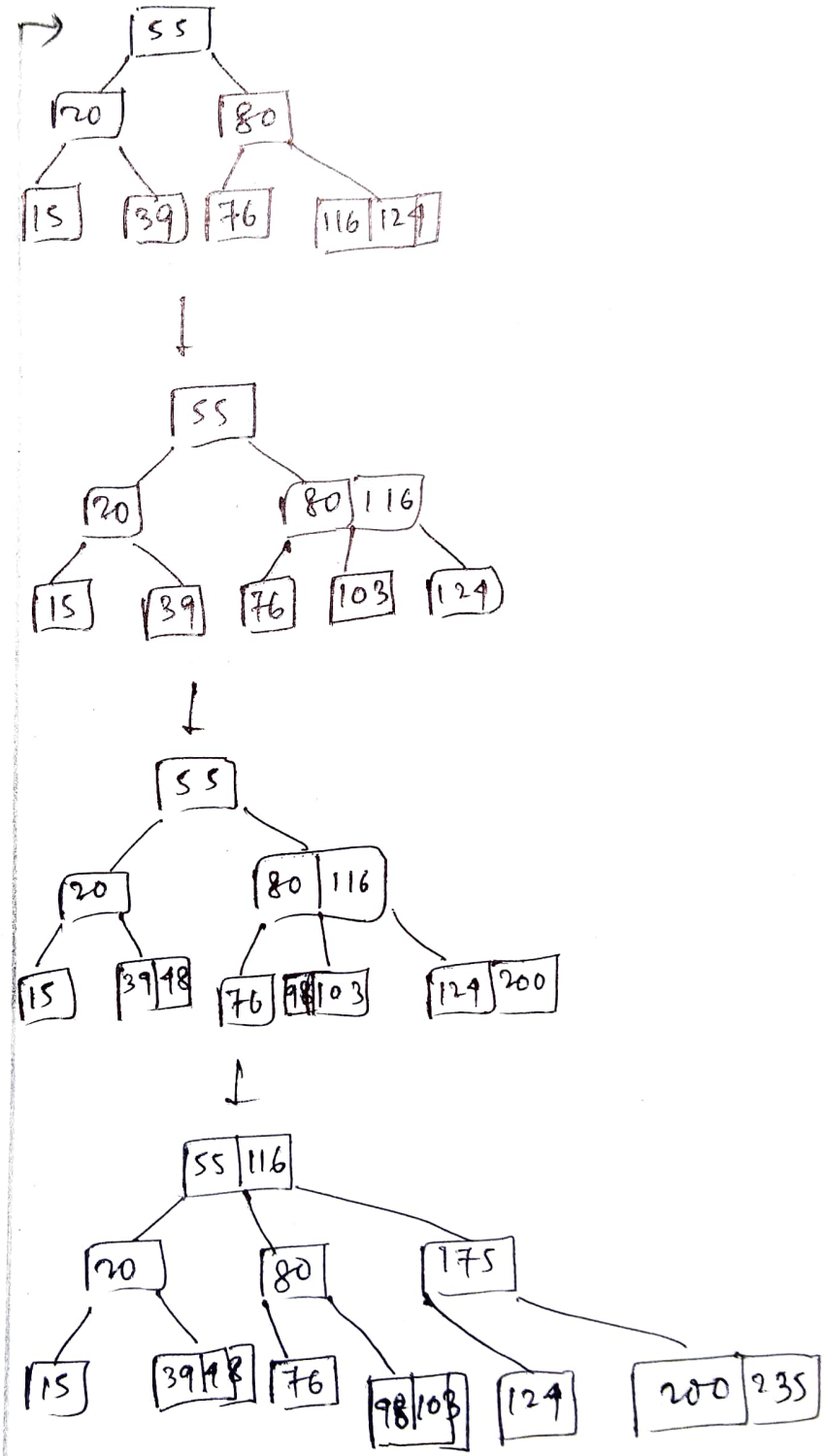
(Ans).

(b) order = 3 , key = 2

keys: 20, 80, 55, 15, 116, 39, 76, 124, 103, 48, 200, 98, 175, 235.

→



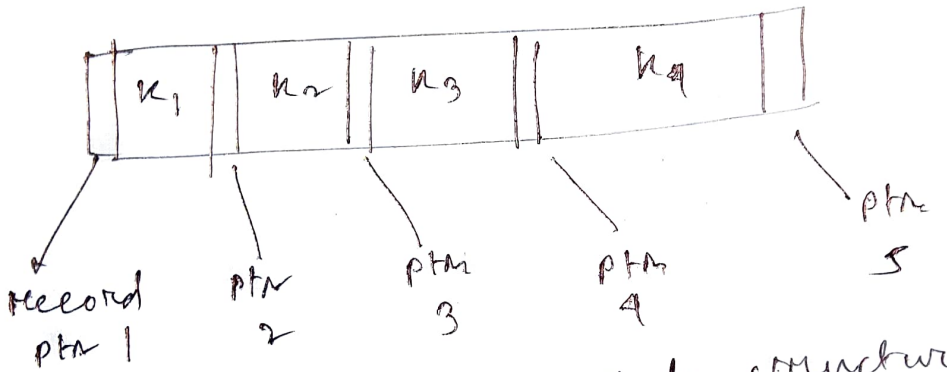


(Ans).

 Name - Chayan kumar Sengupta, Roll - 561, Code - CMSM9124 13.1.24

5-Way B Tree.

It means 5 children or 5 record pointers. that is max 4 keys are possible.



k_i s are keys.

Node structure of 5-way B Tree.