Binary Expression Tree

Expression tree from Infix

$$a * b / c + e / f * g + k - x * y$$

Leaves → operands

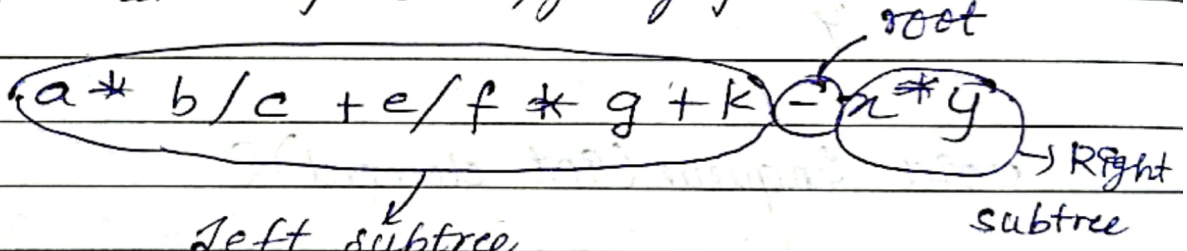
Internal → operators nodes

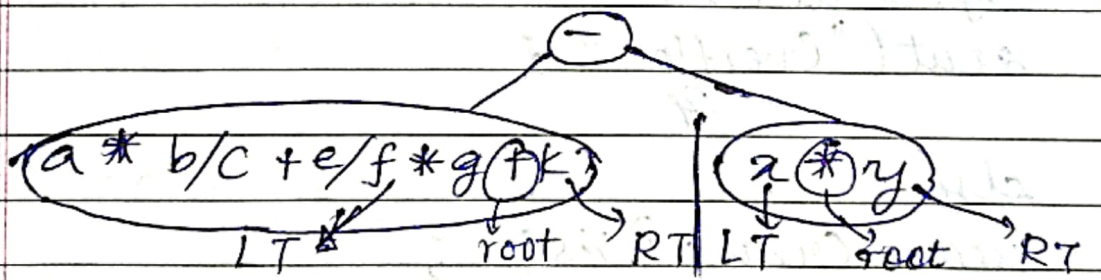
$\wedge$ → R-L

$*, /, \%$ → L-R

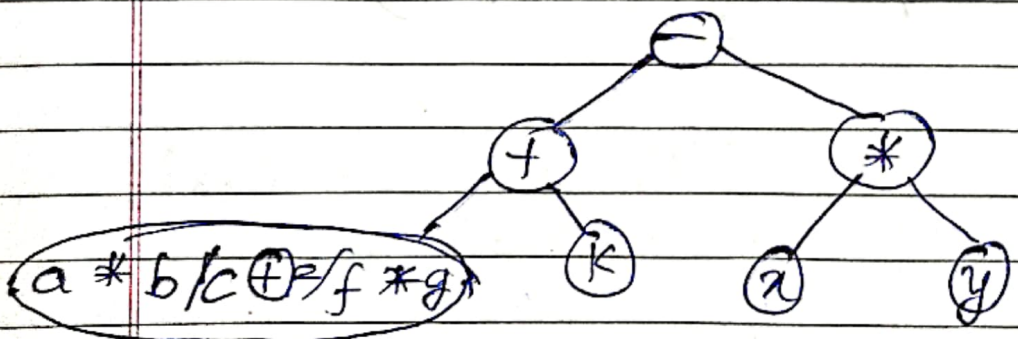
$-, +$ → L-R

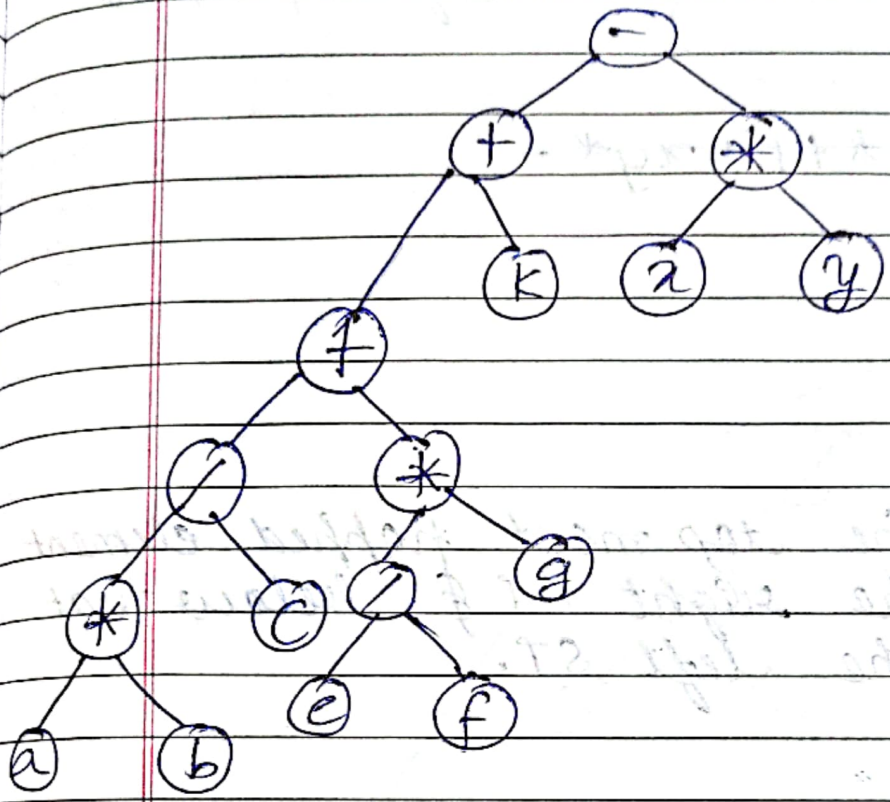
S-1 → Find operator with least precedence at the end of ~~tree~~ exp^n, going from L-R





S-2 → Repeat S-1 for left & right subtrees





Expression tree from Postfix

~~Posfi~~ Postfix:

~~a * b / c~~

ab * c / ef / g * + k + xy * -

Rules: Go from L-R

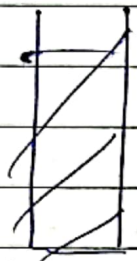
1) If we get an operand in given expression, then push it in the stack.

2) If character is an operator pop two values from the stack make them its child & push the current node again

In the end, the only element of the

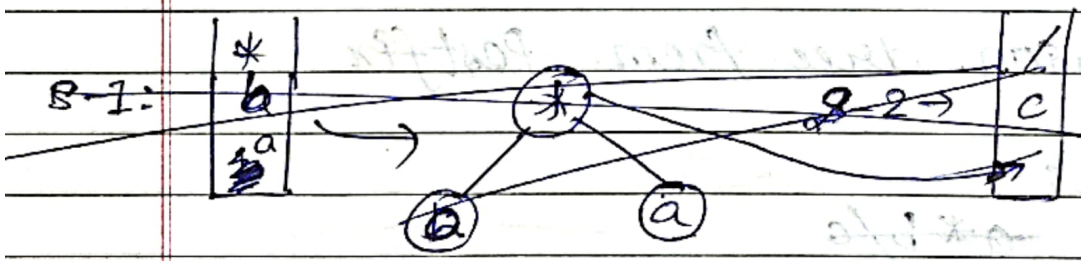
stack will be the root of expression tree.

Pf: ab*c/ef/g*+k+xy*-

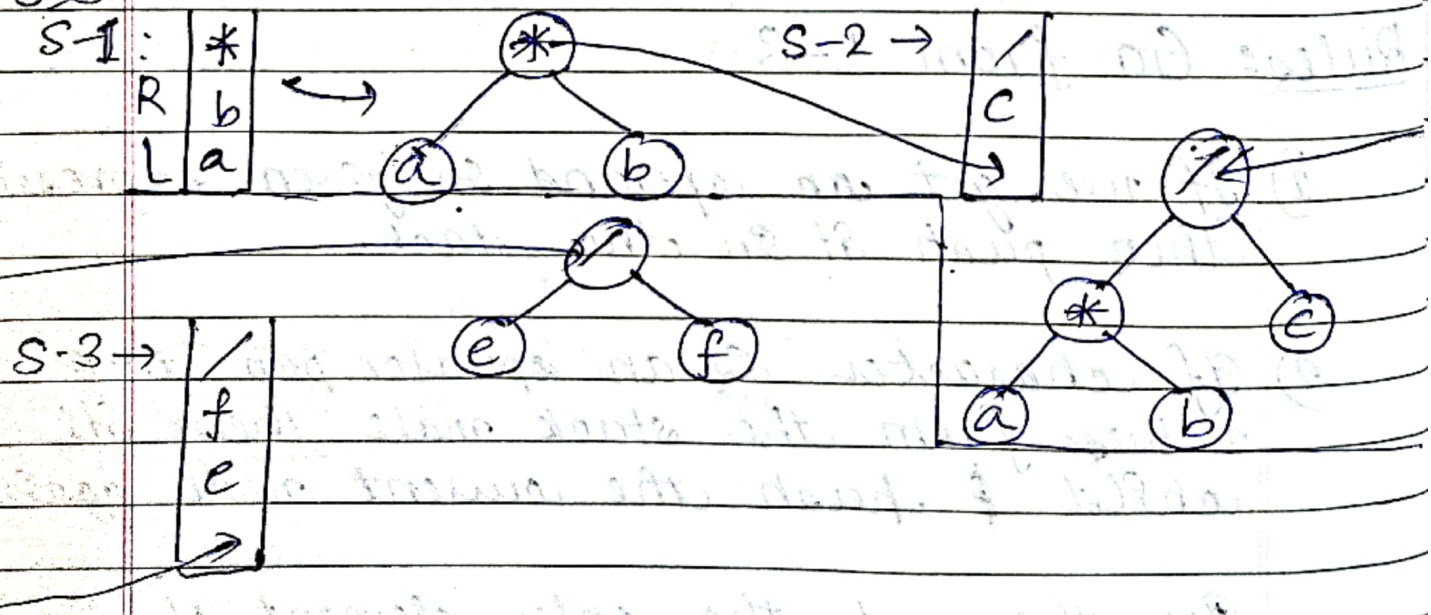


→ Remember the top most popped element would be the right ST & below that would be the left ST.

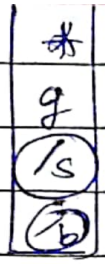
Pf: ab*c/ef/g*+k+xy*-



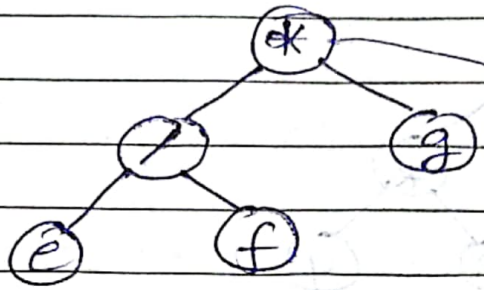
S-3



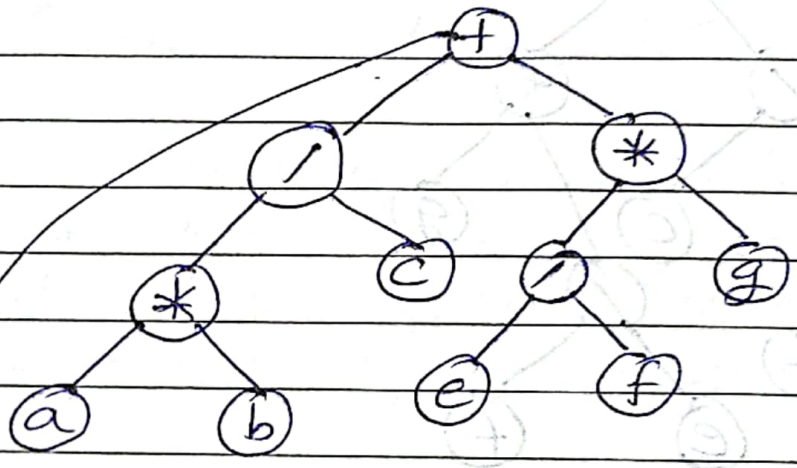
S-4



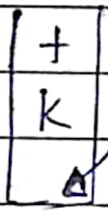
s to small /b to beg



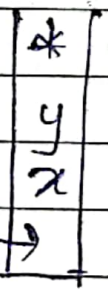
S-5



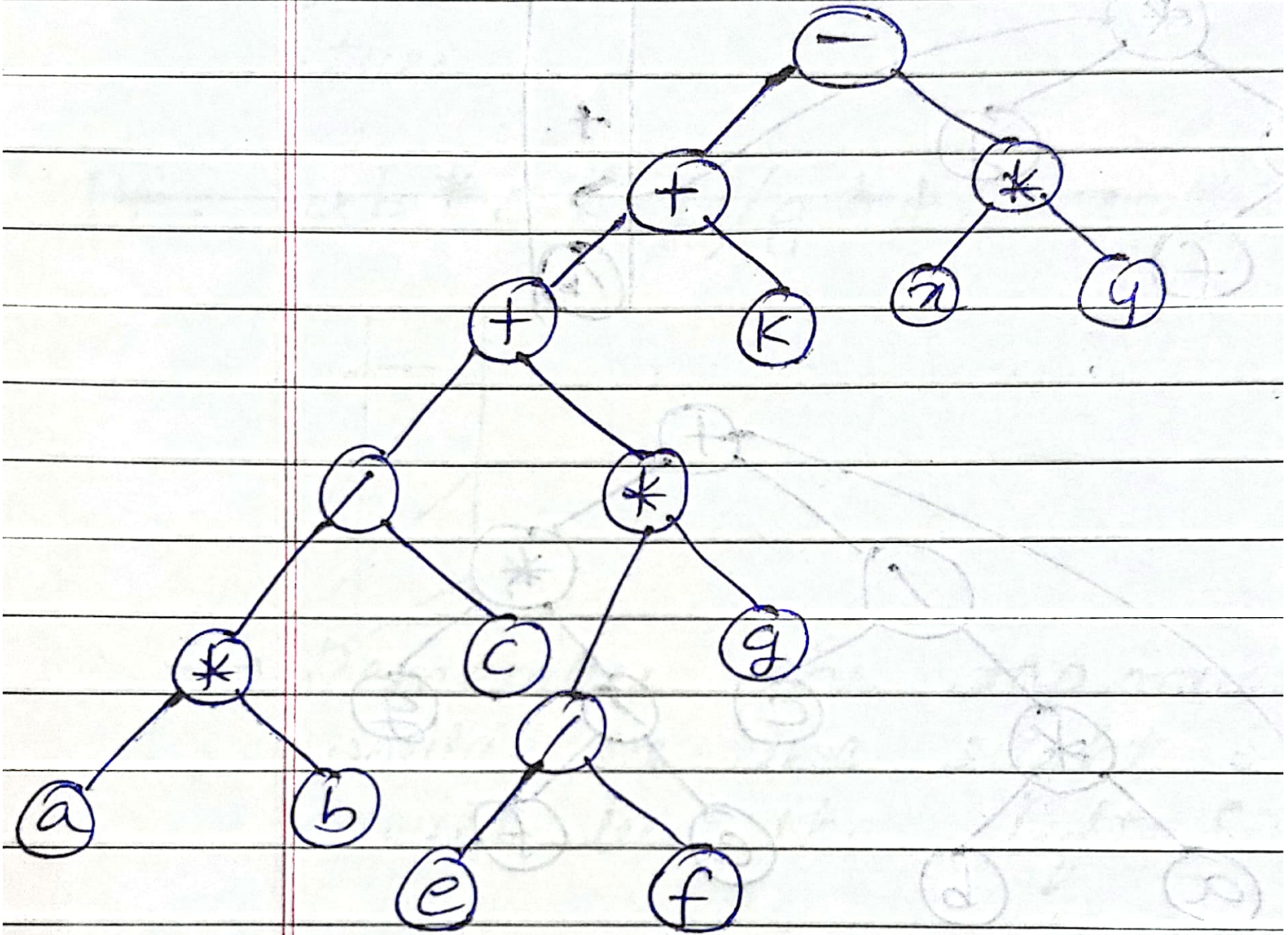
S-6



S-7



S-8


-
+
*
+
k
a
q
/
*
*
c
g
/
a
b
e
f

Const. of expression tree:

~~Postfix expression:~~
Prefix:

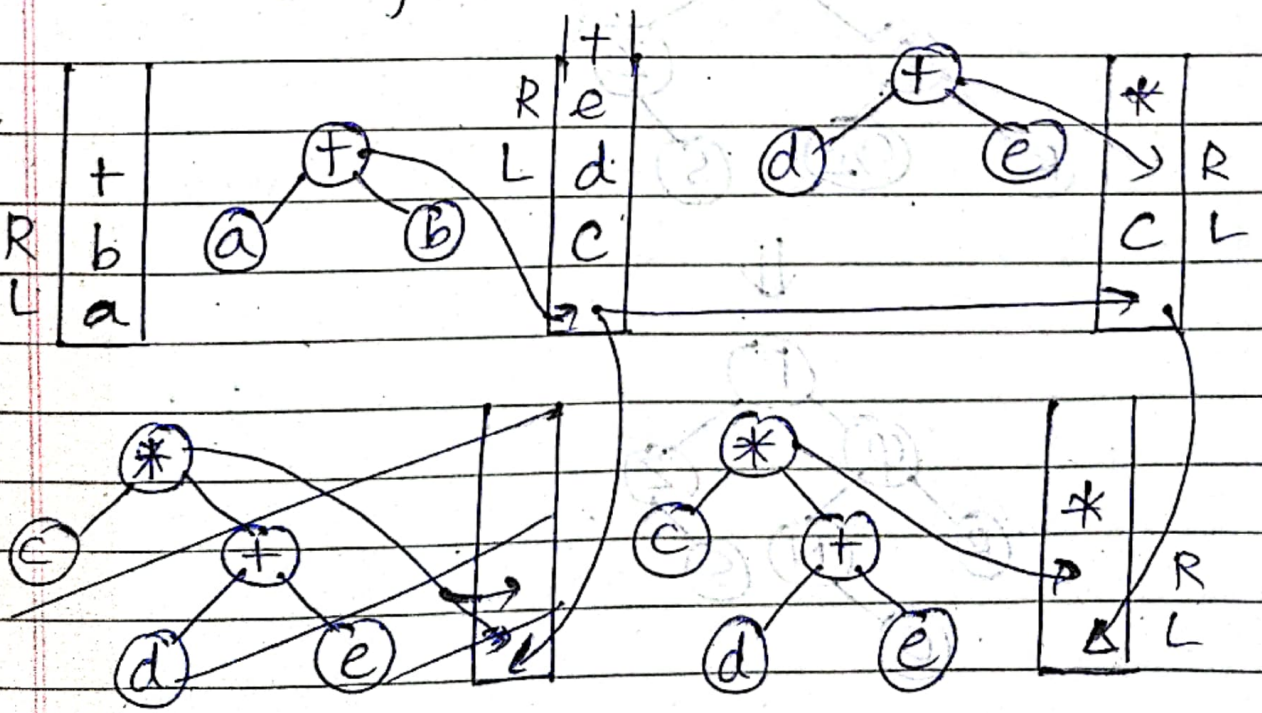
→ Go from R–L

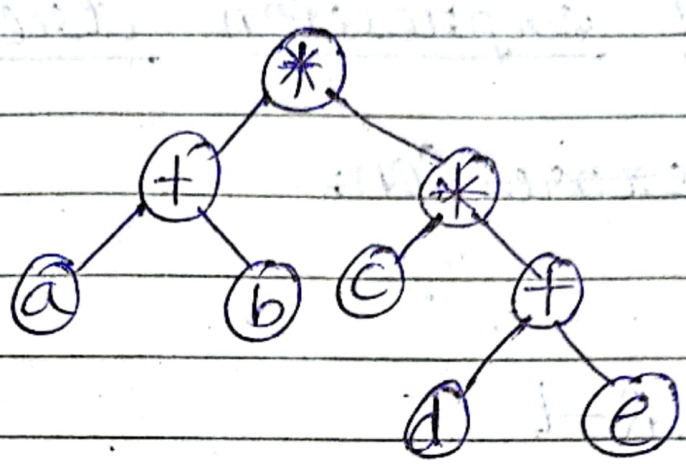
1) If operand push

2) If operator pop 2 values from stack:
~~top~~ 1st → left
2nd → right

3) At the end only element remaining is the root:

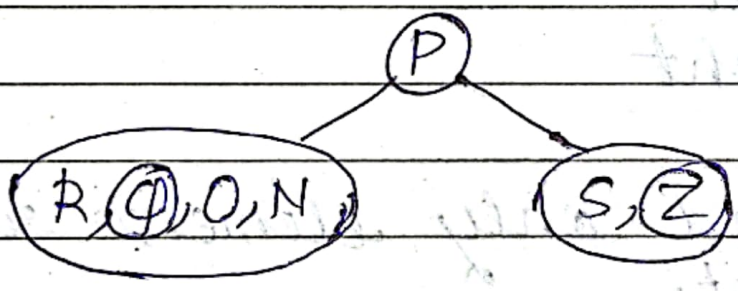
Postfix: ab+cde+**

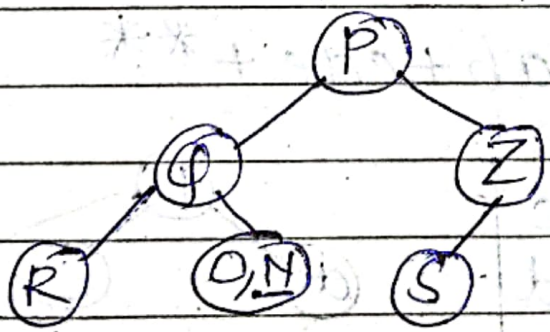


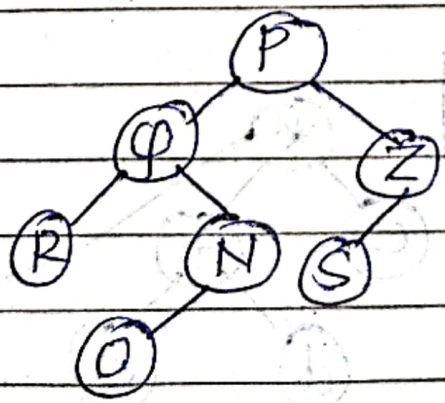


→ Construct a B-tree

Inorder : RQONPSZ
Postorder : RONQSZP







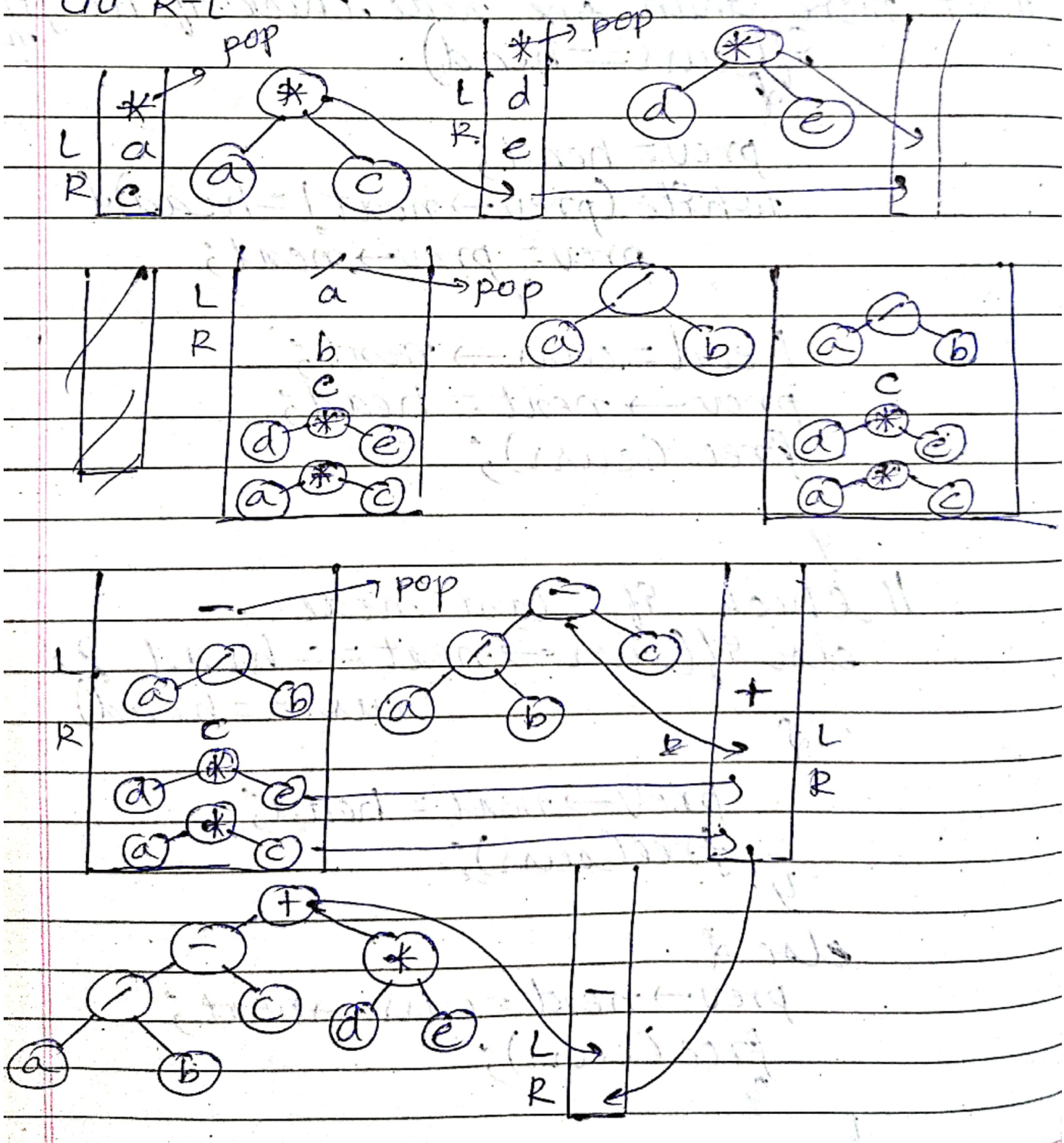
Expression tree from infix:

Prefix:

Prefix:

- + - / a b c * d e * a c

Go R-L



Final expression tree