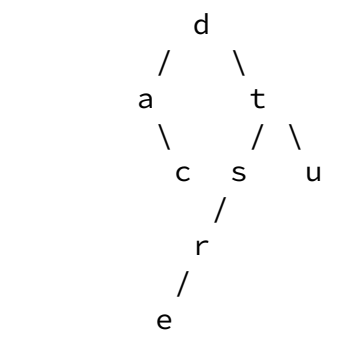
List: "datastructure" => d = 4, a = 1, t = 20, s = 19, r = 18, u = 21, c = 3, e = 5
Basic assumption:
    a is minimum[duplication, ignore.] & z is maximum.



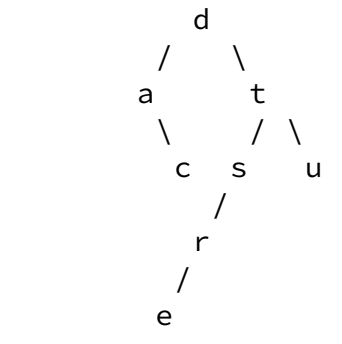
Why we should ignore duplication in BST ?
Problem:
    We cannot generage "datastructure" from the tree.
    It's a lossy represetnation.

    intution: we can search any alphabets(item) from the string "datastructure"
    but we cannot search if whole "datastructure" is present in the
    tree or not.
    Lossy representation is alright for BST representation.
    We donot search position of alphabet. (it cannot be representated by BST)

In Order Traversal: sequence you have given to build the tree.
    go left, visit and go right.

Pre Order: visit node, go left, go right.

Post Order: go left, go right, node



1. a, c, d, e, r, s t, u

    a. visit d after visiting left sub-tree. left child 'a'.

```
45      b. visit a after visiting left sub-tree.
46      c. there no left sub-tree for 'a', we completed visiting
    the left sub-tree for a.
47      d. visit 'a'. completed visiting 'a'.
48      e. visit 'c' after visiting left sub-tree of c.
49      f. there is no left sub-tree for 'c', completed visiting
    left sub-tree of c.
50      g. visit 'c'.
51      h. visit right sub-tree of 'c'.
52      i. there is no right sub-tree of 'c', completed
    visisting right sub-tree of c.
53      j. complete visisting 'a'.
54      k. visit 'd'.
55      l. visist right sub-tree of 'd'.
56      m. visit 't' after visiting left sub-tree of 't'.
57
58
59     2. d, a, c, t, s, r, e, u
60
61      a. visit 'd'
62      b. visit all the left sub-tree of 'd'. i.e. from child
    'a'.
63      c. visit 'a'.
64      d. visit all the left sub-tree of 'a' from no child.
    Complete it.
65      e. visit right sub-tree of 'a' from child 'c'.
66      ...
67
68  Why do we measure balanced factor while building AVL tree?
69  ------------------------------------------------------------
70
71  Why do we need AVL tree ?? then ???
72
73      >> We want to minimize search space than the binary
    search tree.
74      >> balanced binary tree => AVL tree.
75
76
77      [97, 98, 99, 100] => [98, 97, 99, 100]
78
79      we want to construct a BST.
80
81      97                          98
82        \                       /   \
83        98                    97   99
84          \                           \
85           99                          100
86             \
87              100
88
```

```
89      In that case, if you want to search 100, we have to go,
    4 look-ups.
90
91      That's why we AVL which is balanced BST.
92
93      How ??
94
95      [97, 98, 99, 100]
96
97      97
98        \
99        98
100         \
101         99  Case of Right-Right. RR
102         Make 98 as root. Make its previous root to its left
    child, keep right same.
103
104     98
105    /  \
106   97  99
107         \
108         100
109
110  LL =>
111
112             99
113            /
114          98
115          /
116         97
117         Make 98 as root. Make its previous root to its right
    child, keep left same.
118
119 LR =>
120
121     100
122    /
123   98
124
    \
125      99
126      Make two rotations.
127      a. Replace 98 by its right child 99, and put 98 to its
    left child.
128      b. Make 99 as root. Put 100 to its right child and keep
    left same.
129
130      99
131     /  \
132    98  100  Search in balanced tree required less amount
```

```
    lookup than unbalaced tree.
133
134   RL => ??
135
136
137 Infix to Postfix: 3 + 2 * 4
138
139     =>  11,  20(wrong)
140
141     input 3, => out 3.: 3 : Stack => []
142     input +, => push + to stack. Stack => [+]
143     input 2, => out 2.: 3 2 : Stack =>[+]
144     input *, => push * to stack, because it is higher
    precedence than +. [+ *]
145     input 4, => out 4. : 3 2 4: Stack => [+ *]
146     input empty, => Pop from stack. => 3 2 4 * +
147
148 Infix to Postfix: 3 * 2 + 4 => 10.
149
150     input 3, => out 3, Stack => []
151     input *, => push *, Stack => [*]
152     input 2, => out 3 2, Stack => [*]
153     input +, => out 3 2 *, Stack => [+]
154     input 4, => out 3 2 * 4, Stack => [+]
155     input empty, => Post from the stack. => 3 2 * 4 +
156
157     $: exponent operator has highest precedence.
158
159      4+-5 wrong.
160      4$*(5 wrong.
161
162
```