BINARY TREE TRAVERSALS

PREORDER

INORDER

POSTORDER

PREORDER

Visit root node, then left subtree and finally the right subtree.

```
def preorder(root):

  if root:
    # Traverse root
    print(str(root.val) + "->", end=‘’)
    # Traverse left
    preorder(root.left)
    # Traverse right
    preorder(root.right)
```

PREORDER

INORDER

Visit left subtree, then root node and finally the right subtree.

```
def inorder(root):

  if root:
     # Traverse left
     inorder(root.left)
     # Traverse root
     print(str(root.val) + "->", end='')
     # Traverse right
     inorder(root.right)
```

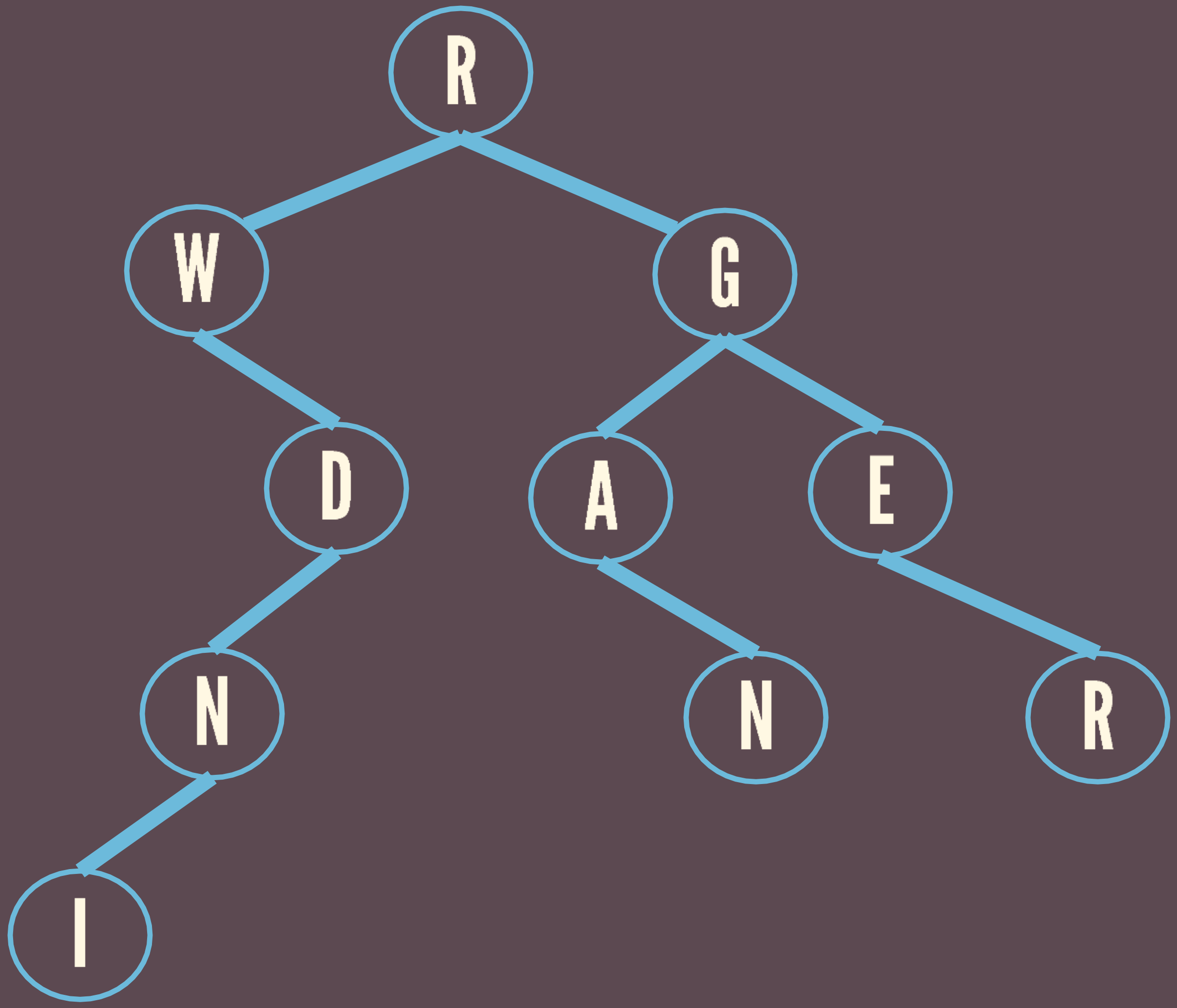
INORDER

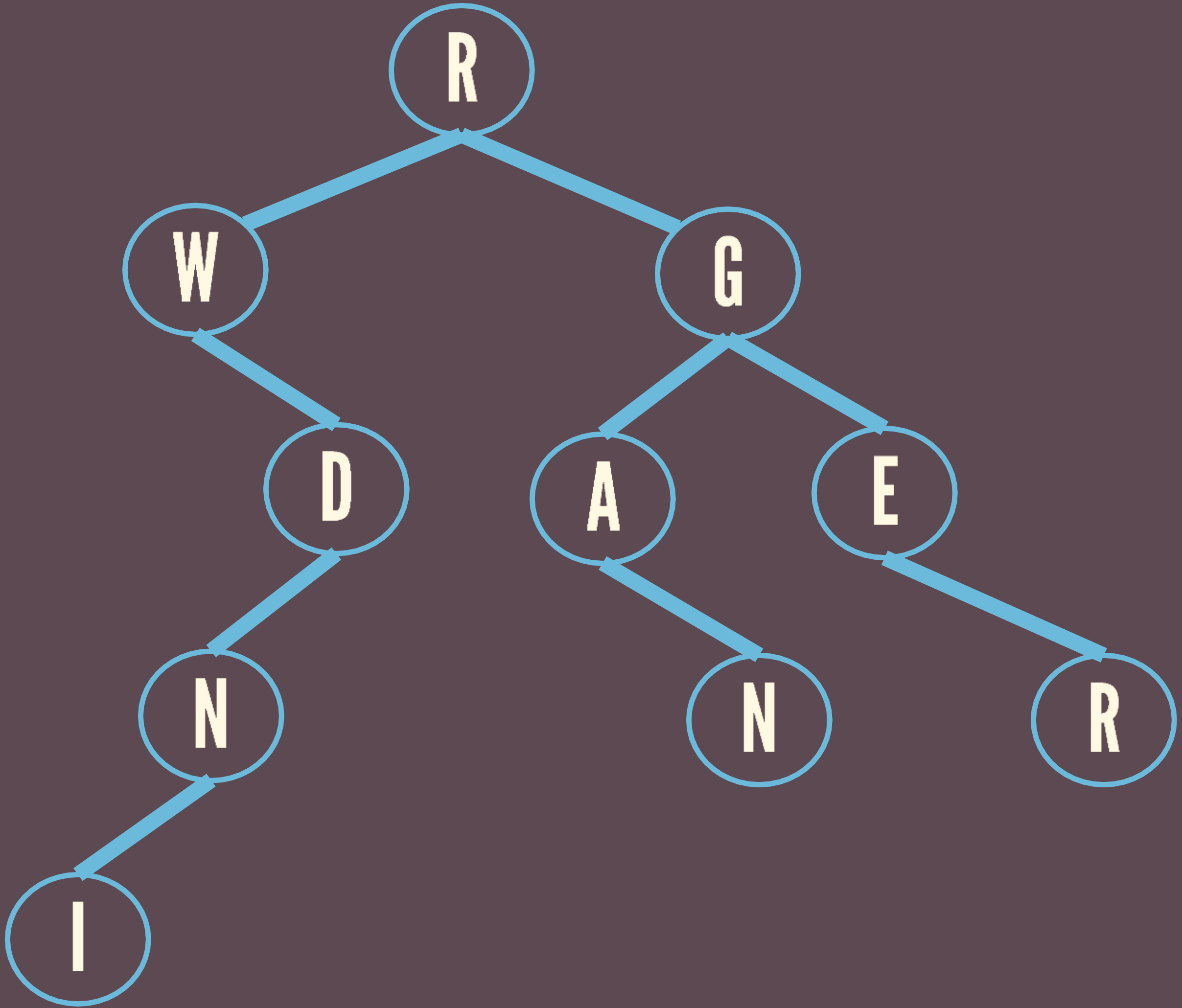
POSTORDER

Visit left subtree, then right subtree and finally the root node.

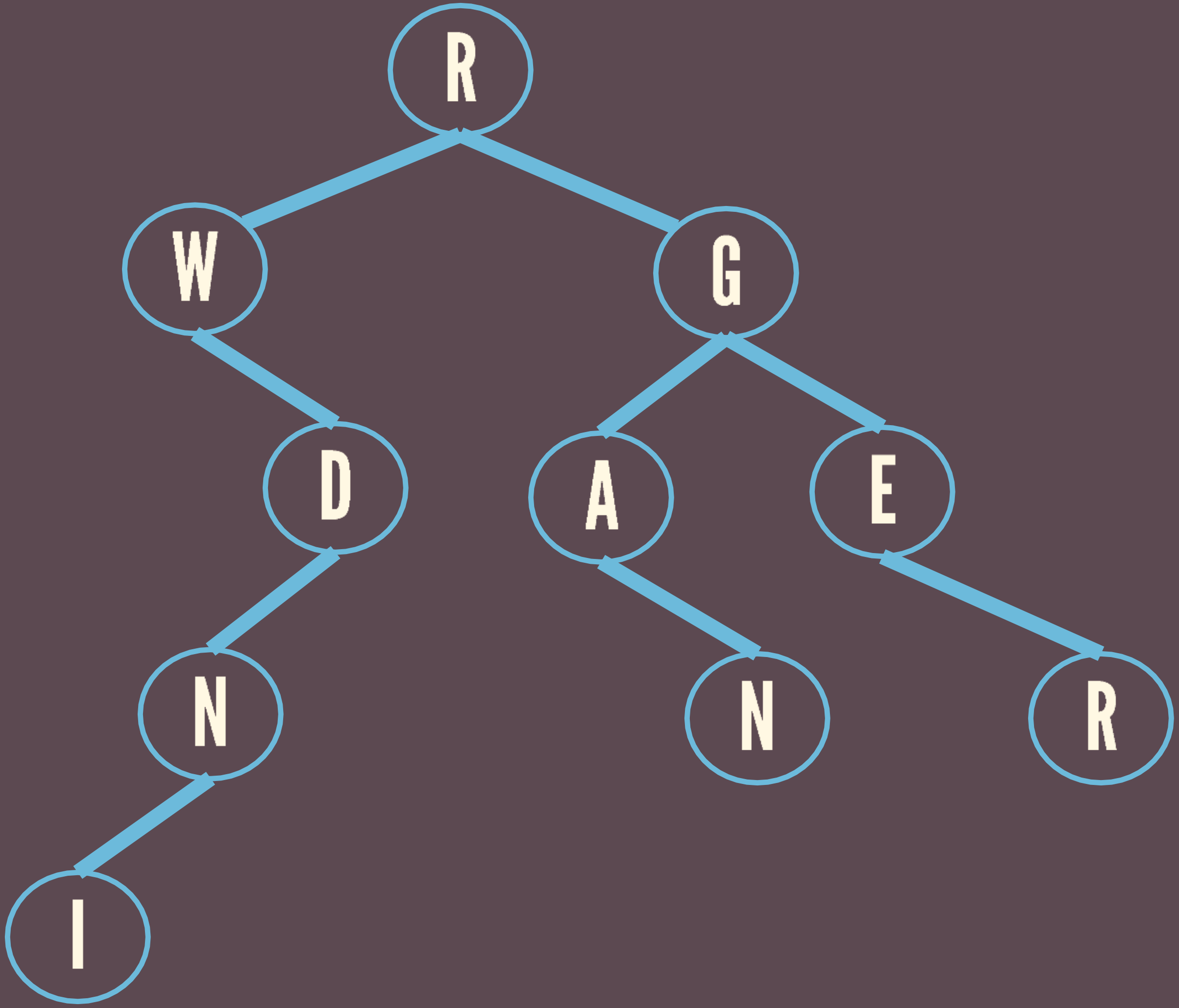
```python
def postorder(root):

  if root:
      # Traverse left
      postorder(root.left)
      # Traverse right
      postorder(root.right)
      # Traverse root
      print(str(root.val) + "->", end='')
```

POSTORDER

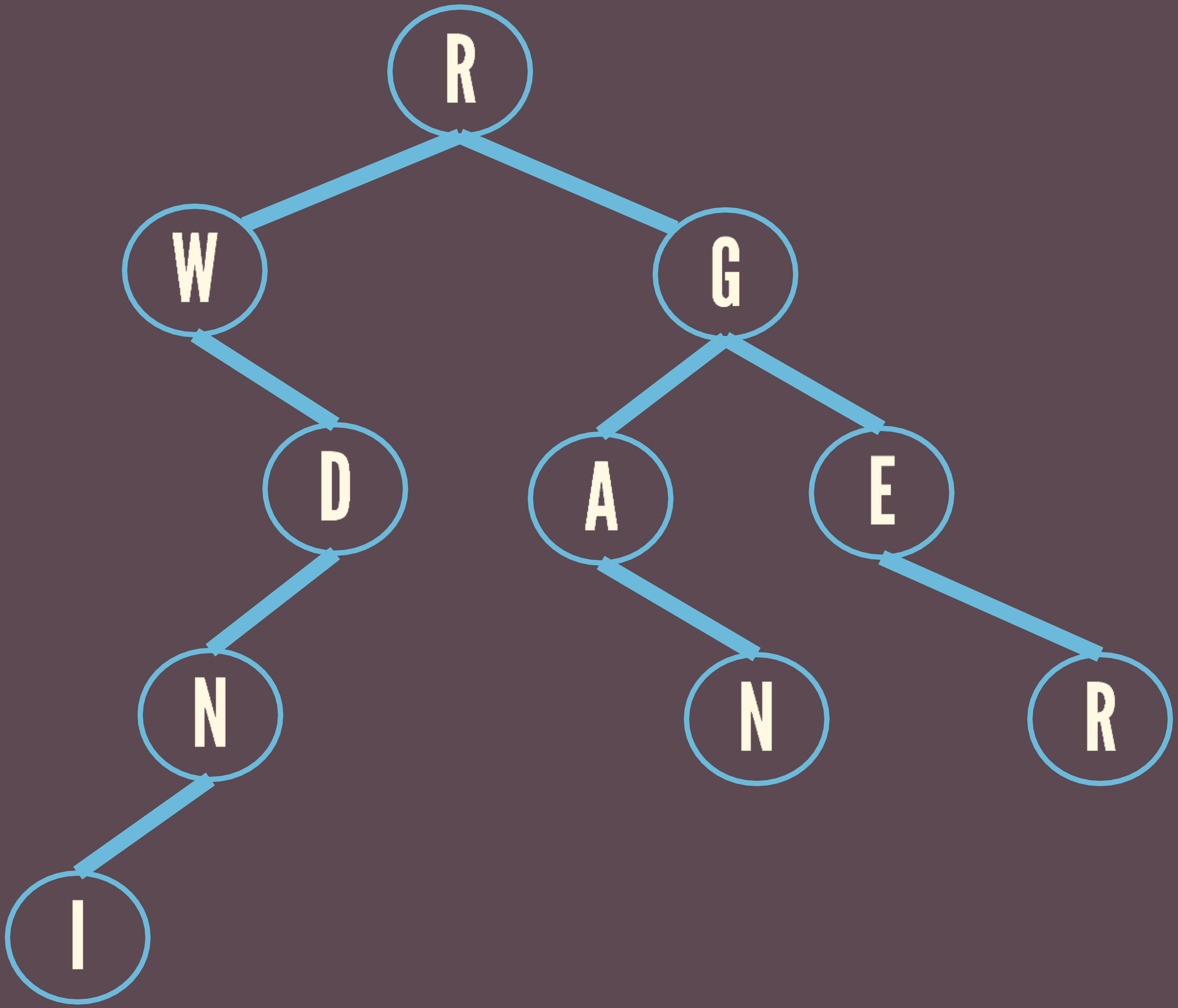

R
W
G
D
A
E
N
N
R
I



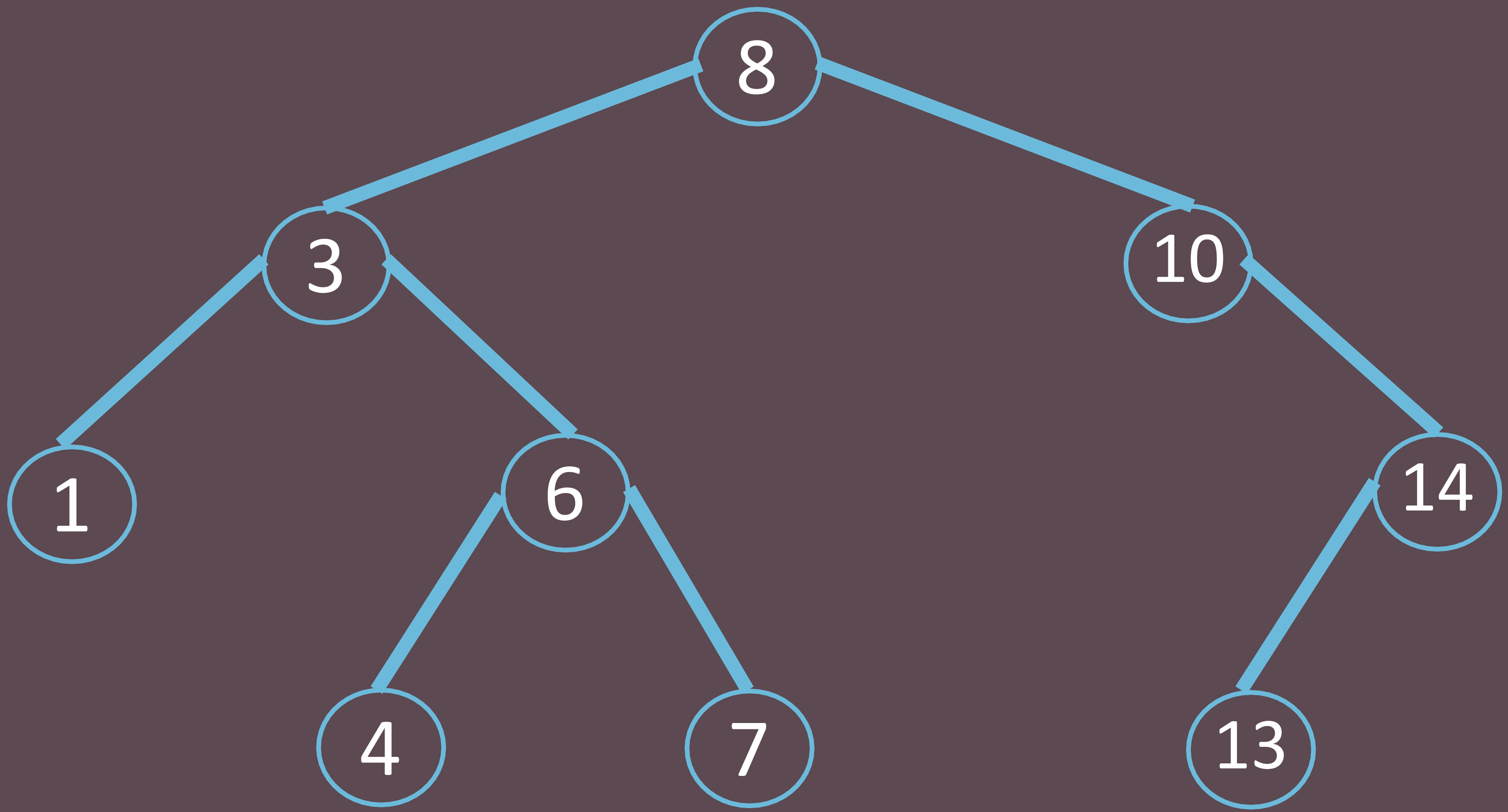
PREORDER: R W D N I G A N E R

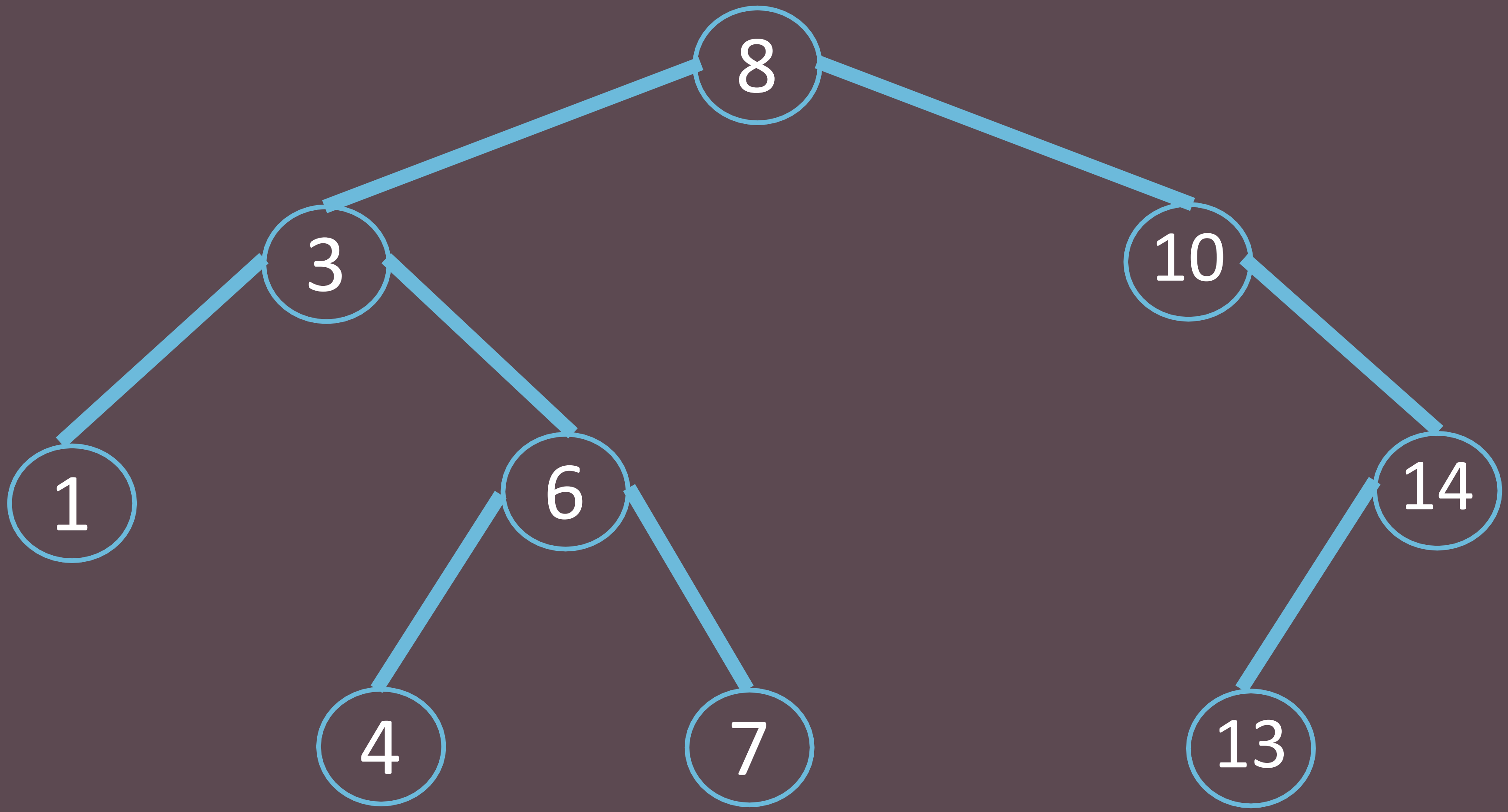


INORDER: W I N D R A N G E R

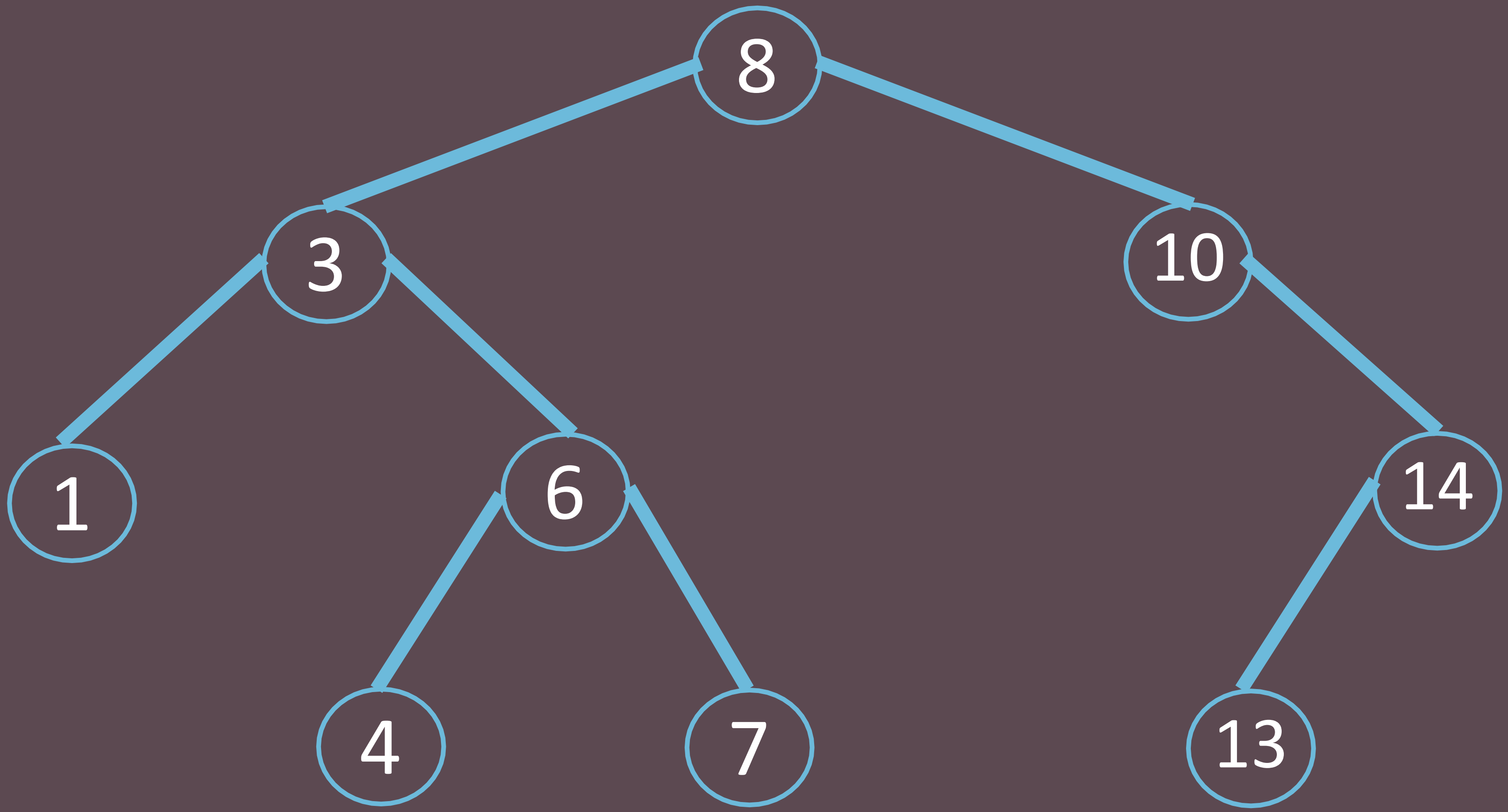


POSTORDER: I N D W N A R E G R

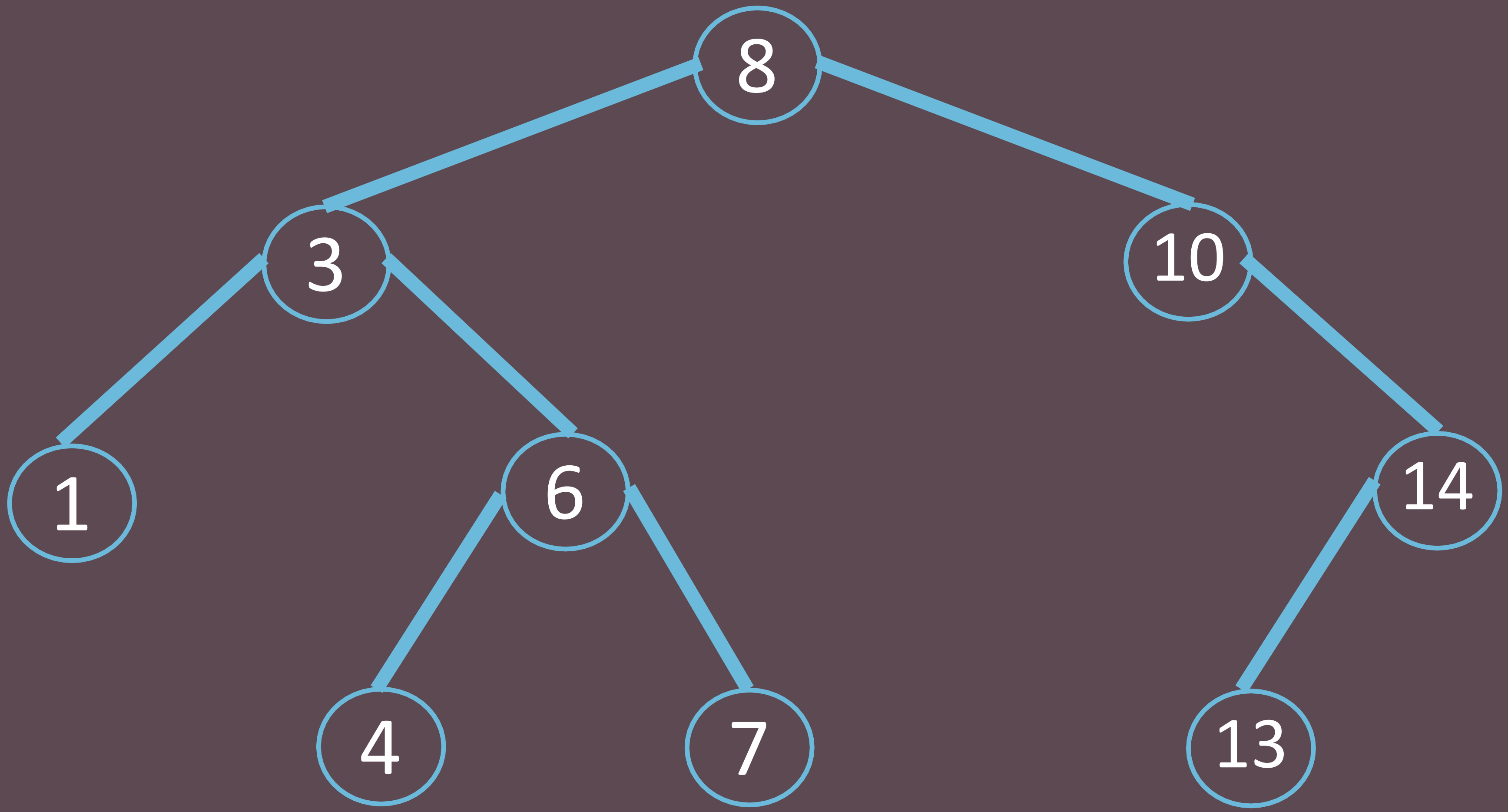

8
3
10
1
6
14
4
7
13



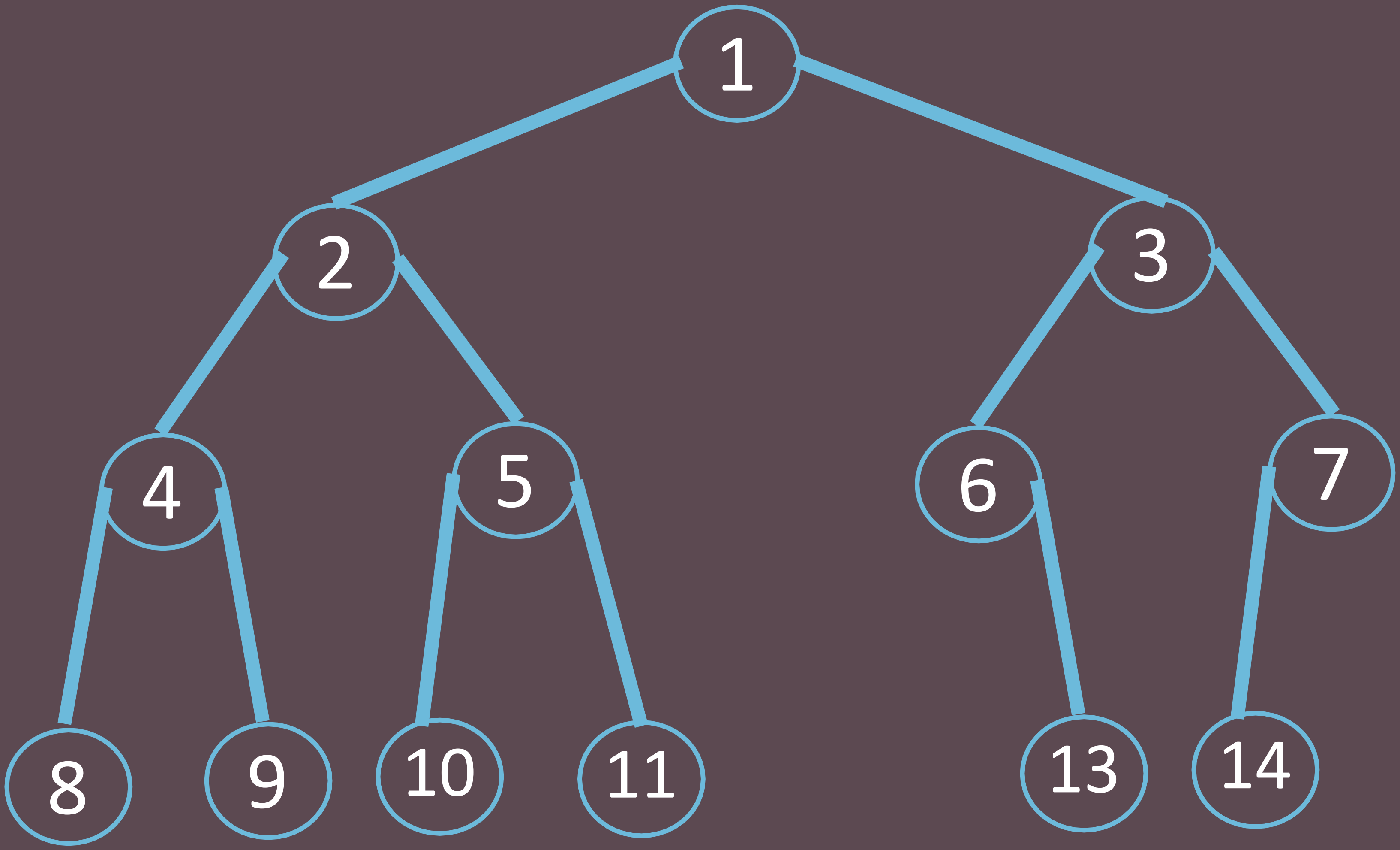
PREORDER: 8 3 1 6 4 7 10 14 13

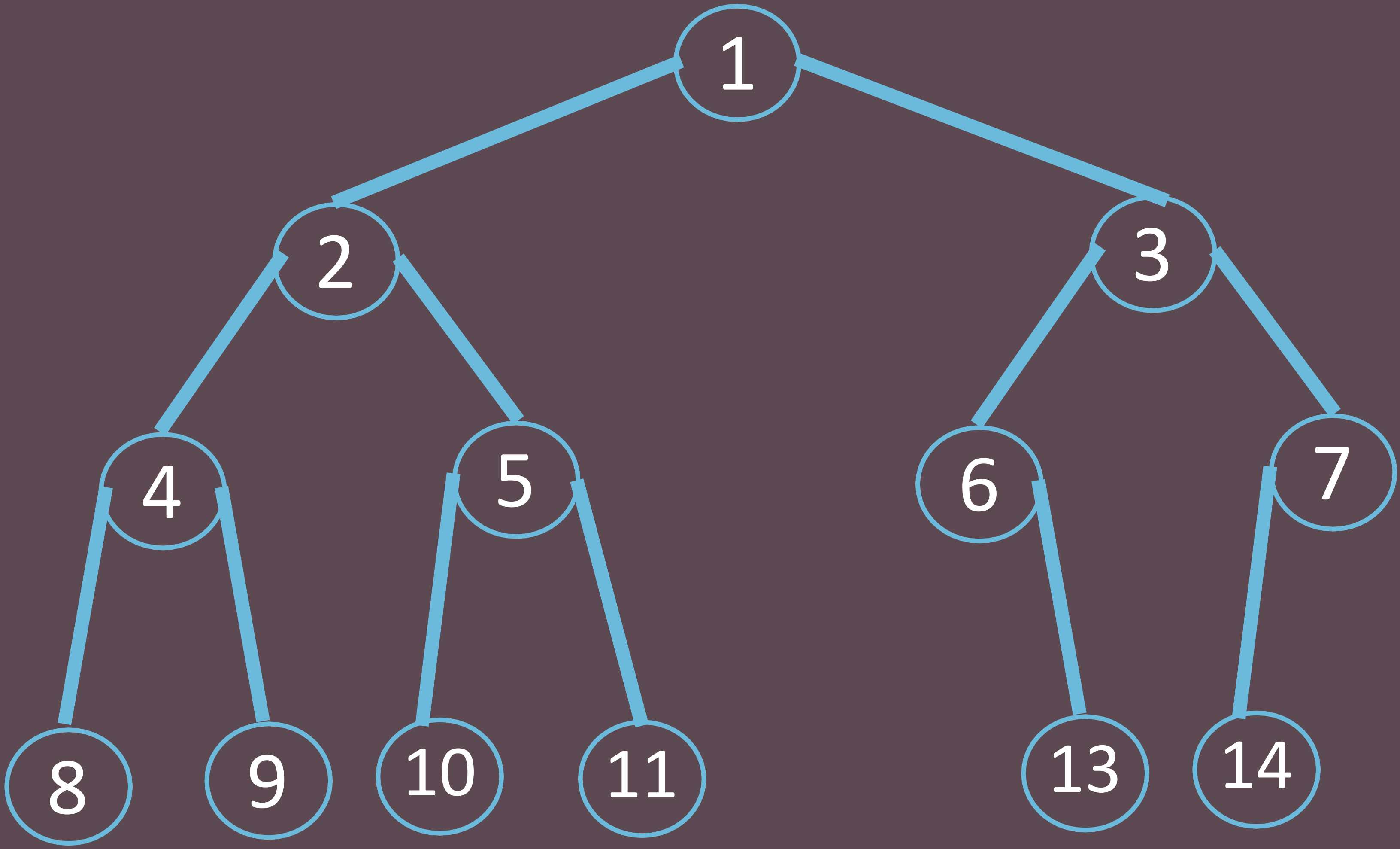


INORDER: 1 3 4 6 7 8 10 13 14

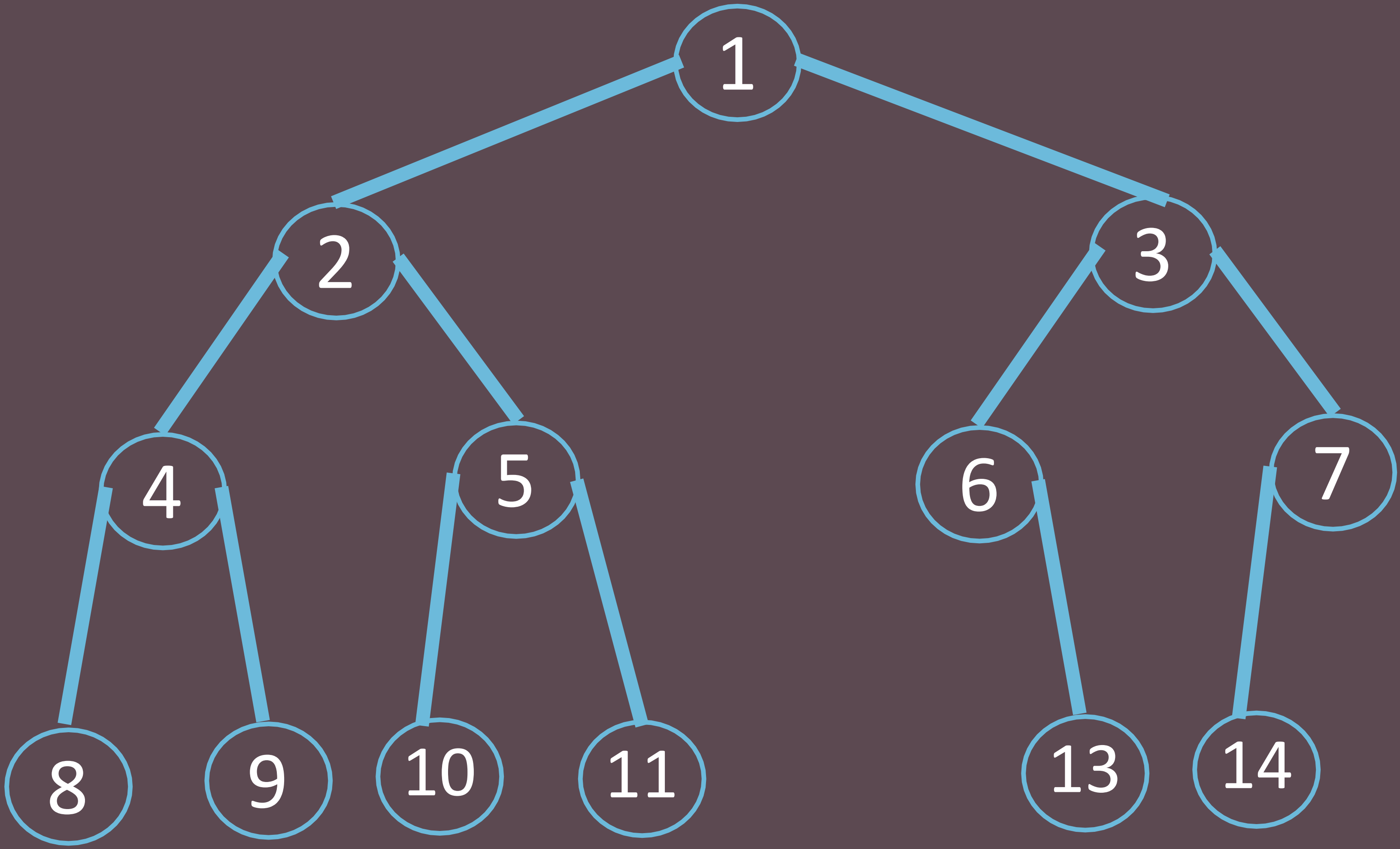


POSTORDER: 1 4 7 6 3 13 14 10 8

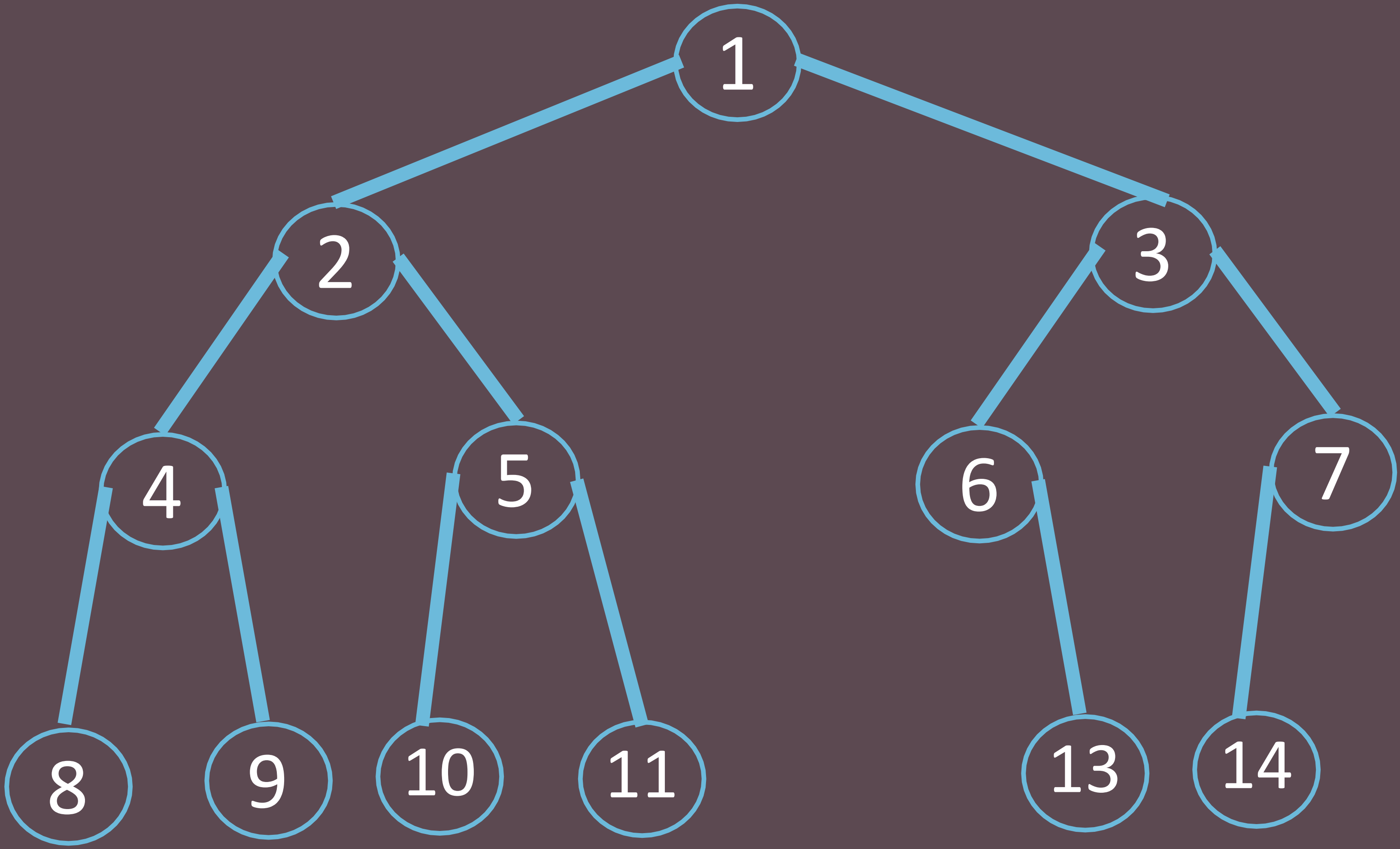

1
2
3
4
5
6
7
8
9
10
11
13
14



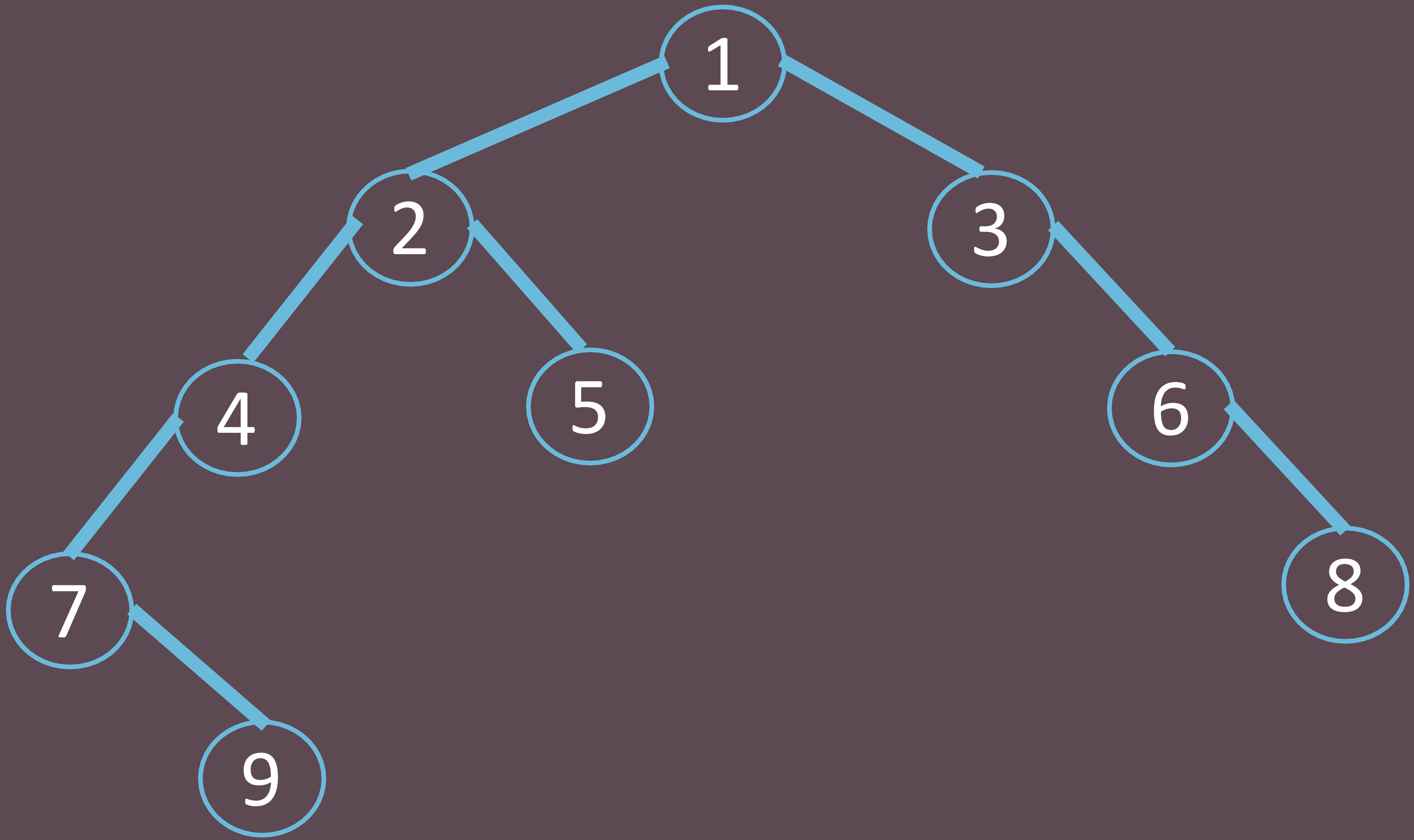
PREORDER: 1 2 4 8 9 5 10 11 3 6 13 7 14

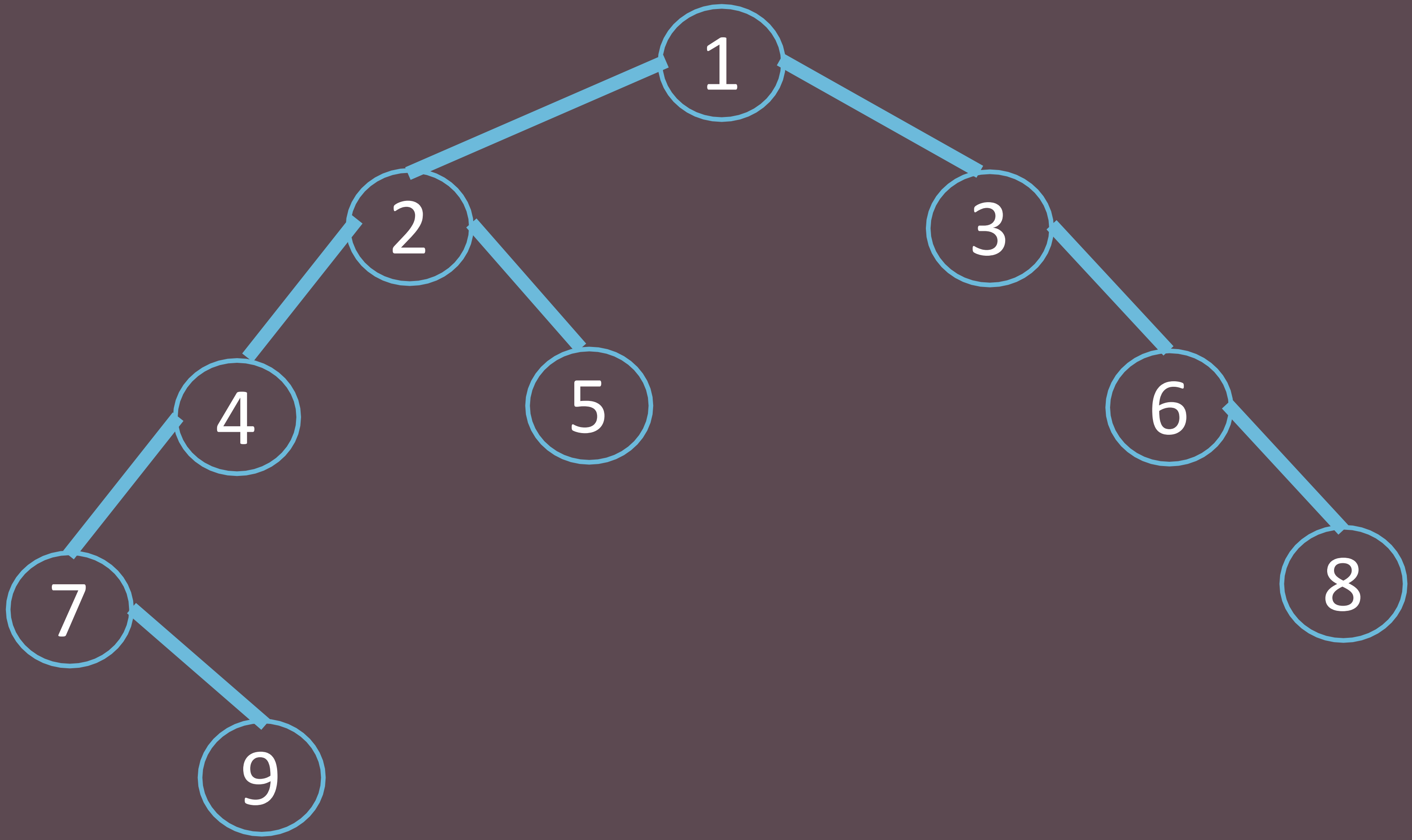


INORDER: 8 4 9 2 10 5 11 1 6 13 3 14 7

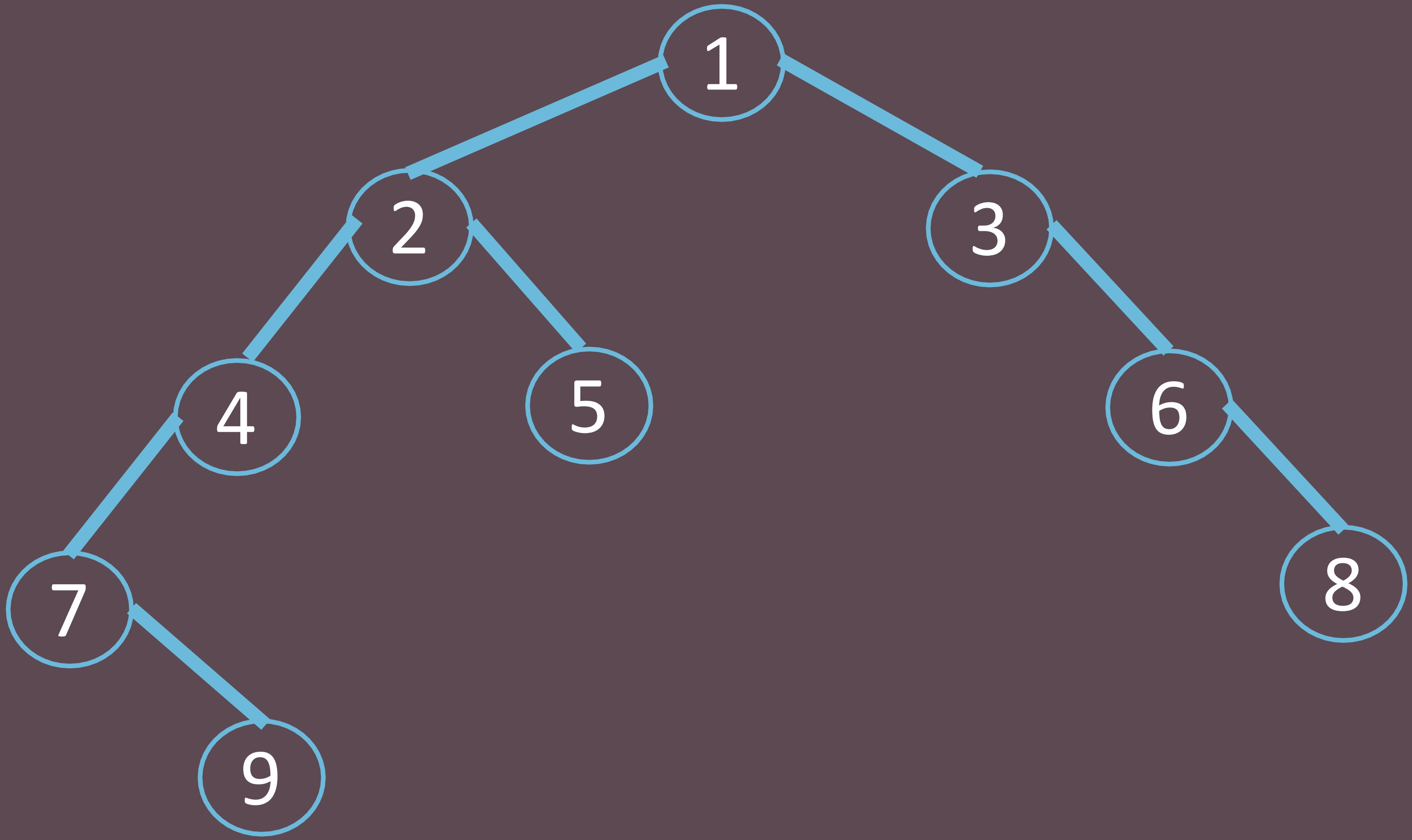


POSTORDER: 8 9 4 10 11 5 2 13 6 14 7 3 1

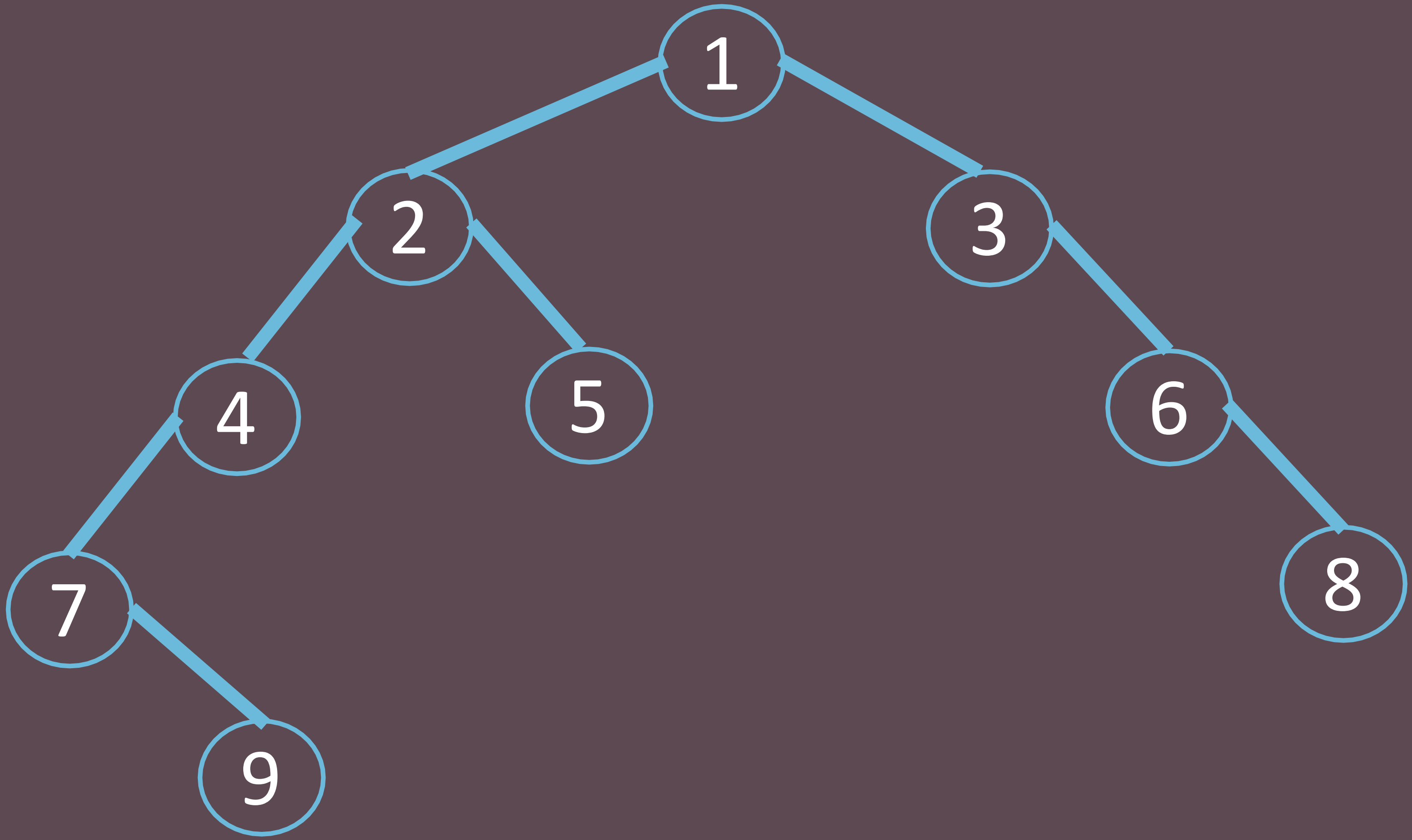

1
2
3
4
5
6
7
8
9



PREORDER: 1 2 4 7 9 5 3 6 8



INORDER: 7 9 4 2 5 1 3 6 8



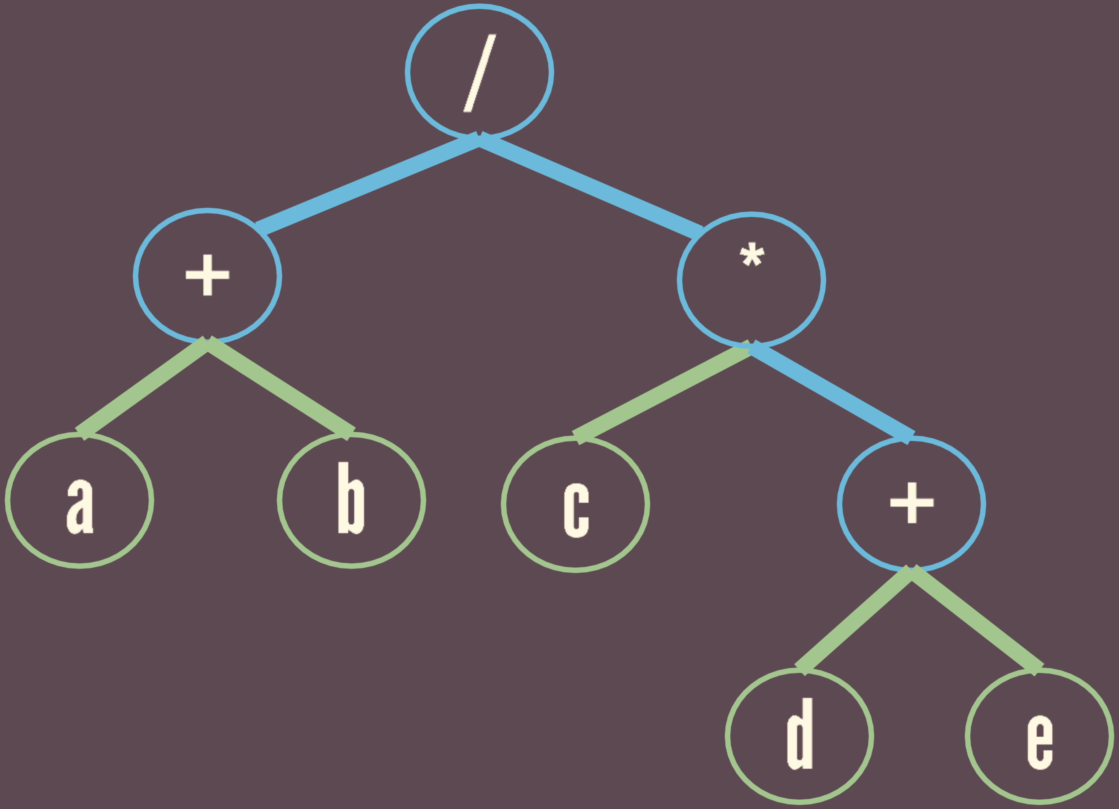
POSTORDER: 9 7 4 5 2 8 6 3 1

EXPRESSION TREES

LEAVES
OPERANDS

INTERNAL NODES
OPERATORS

(a + b) / (c * (d + e))

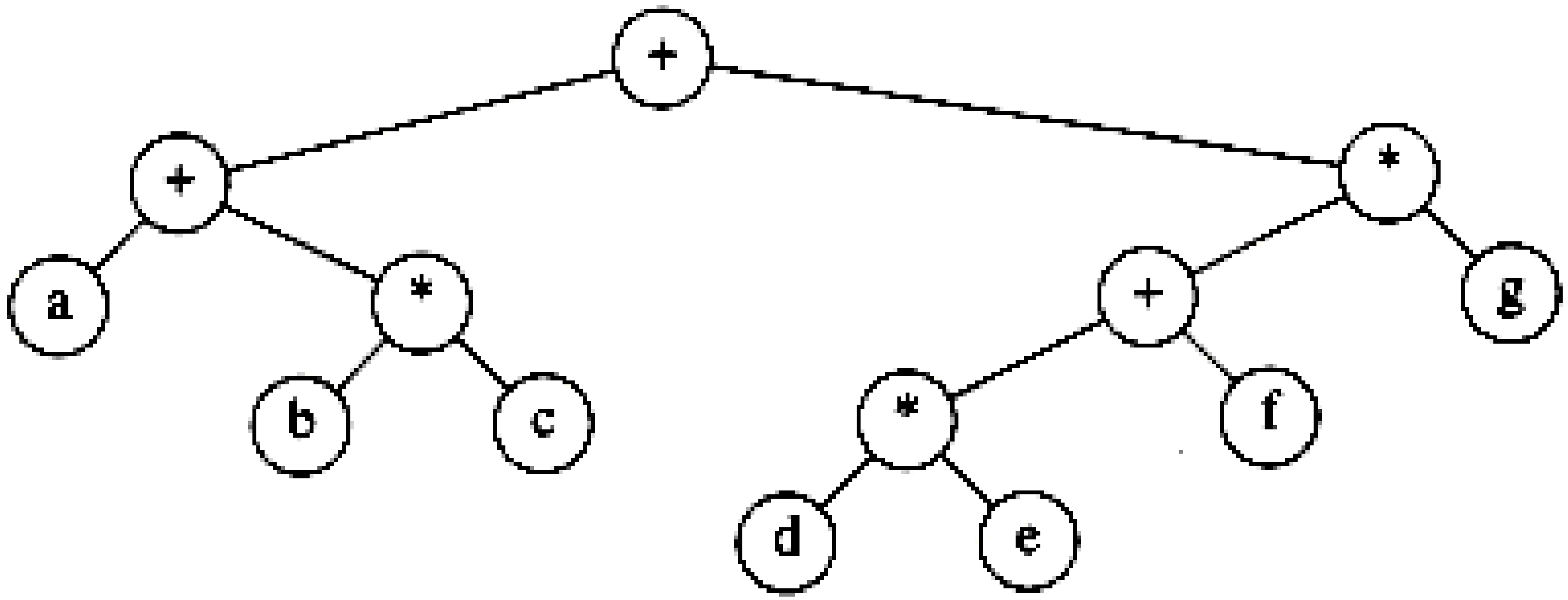


EXPRESSION TREE TRAVERSALS

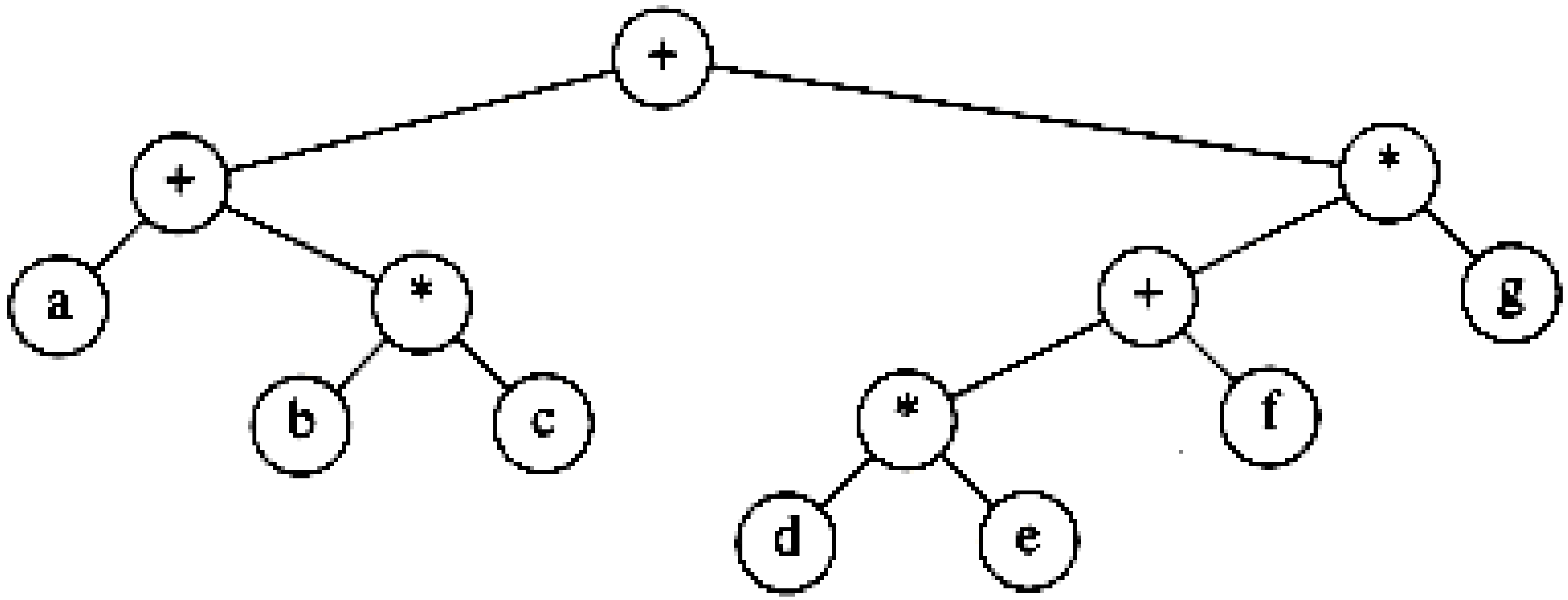
PREORDER
PREFIX

INORDER
INFIX

POSTORDER
POSFIX

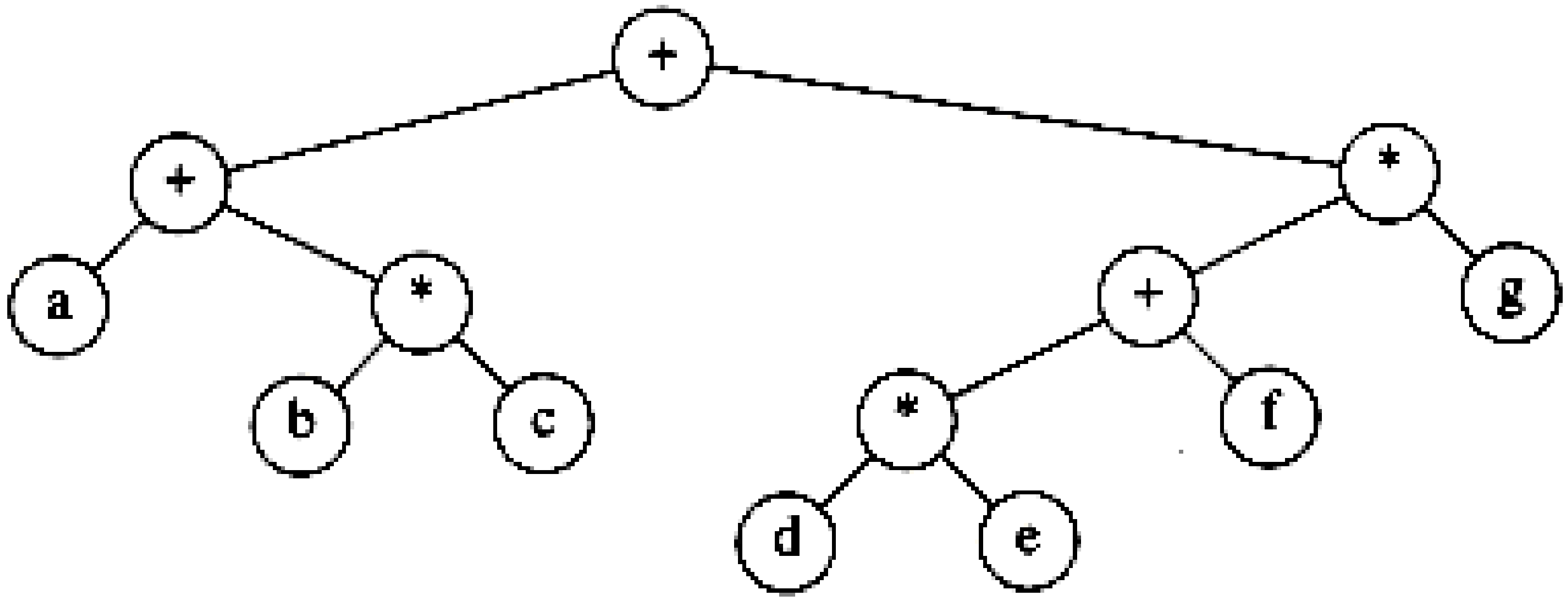


(a + b * c) + ((d * e + f) * g)



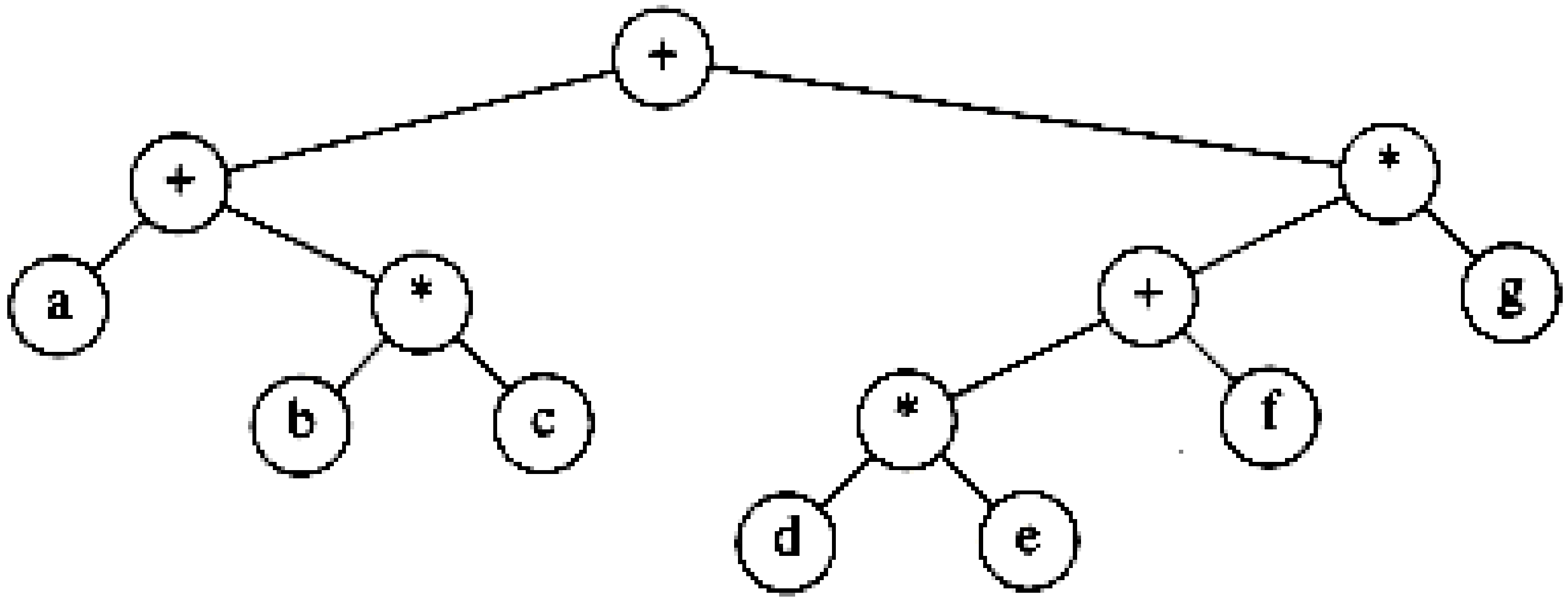
PREFIX

+ + a * b c * + * d e f g



POSTFIX

a b c * + d e * f + g * +



INFIX

a + b * c + d * e + f * g

ALGORITHM TO CONSTRUCT

EXPRESSION TREES

Convert the expression to postfix.

Use a stack.

Read the expression (postfix) one symbol at a time:

if the symbol is an operand,

- create a one-node tree
- push a pointer to it onto a stack

Read the expression (postfix) one symbol at a time:

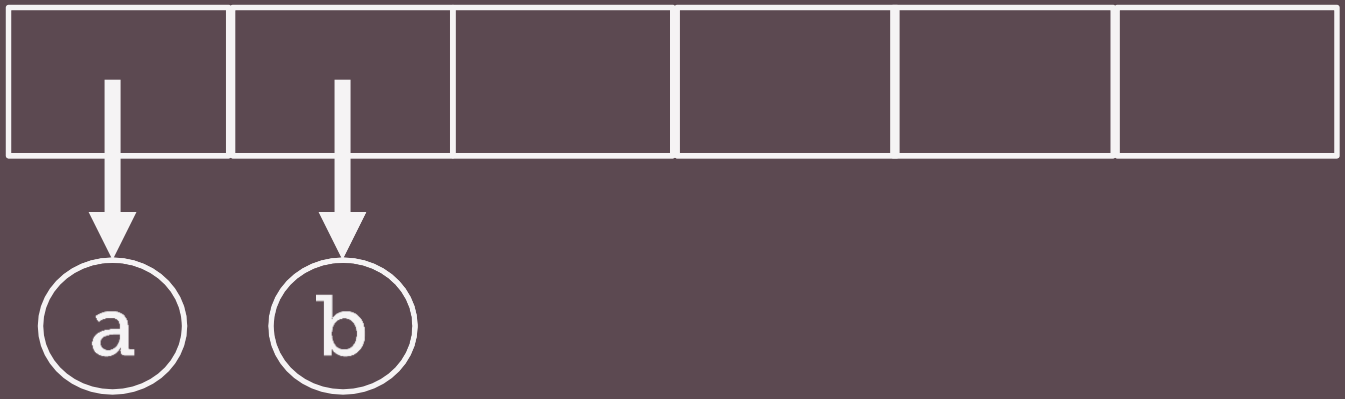
if the symbol is an operator,

- pop two pointers to two trees (T_1 and T_2).
- Form a new tree whose root is the operator with left and right child pointing to T_1 and T_2 respectively.
- push onto the stack a pointer to this new tree.

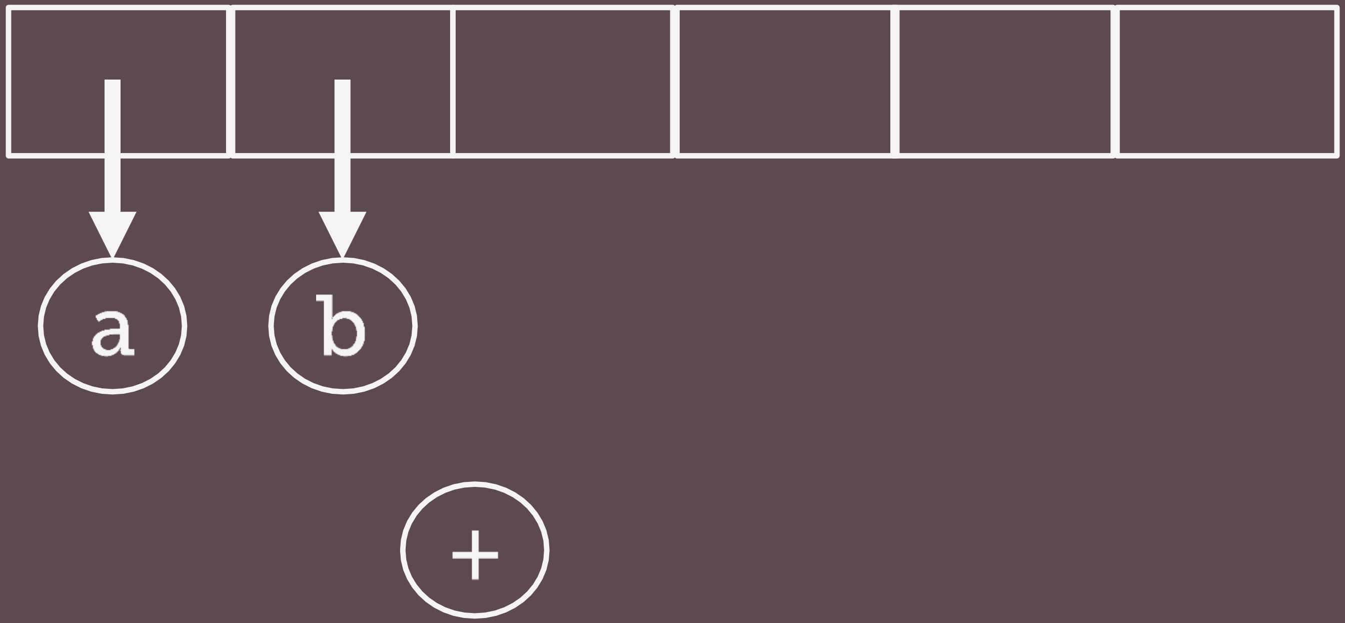
EXAMPLE

a b + c d e + * *

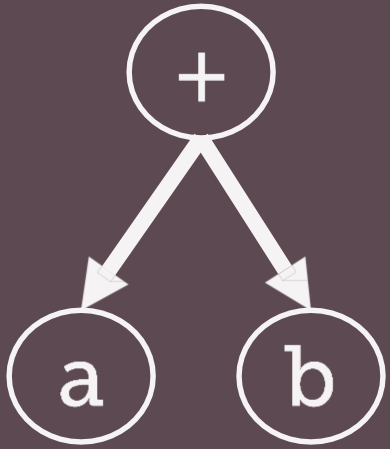
a b + c d e + * *



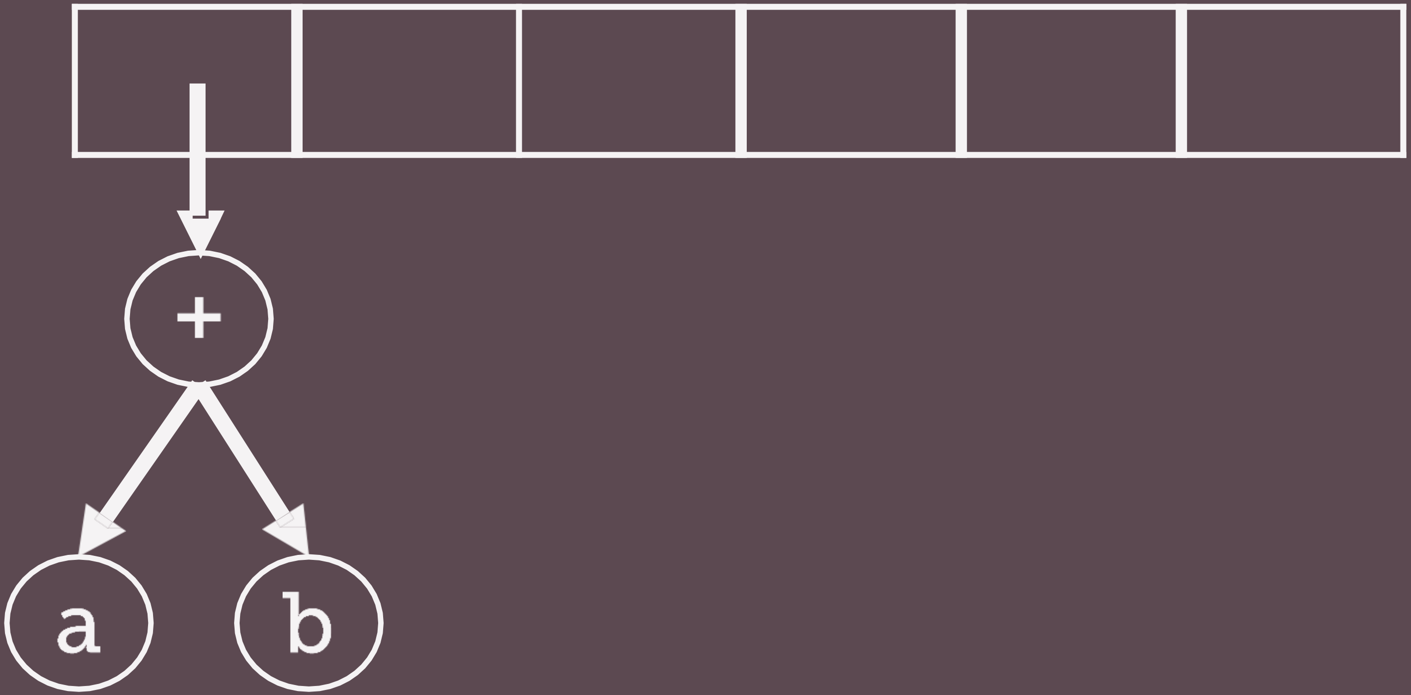
a b + c d e + * *



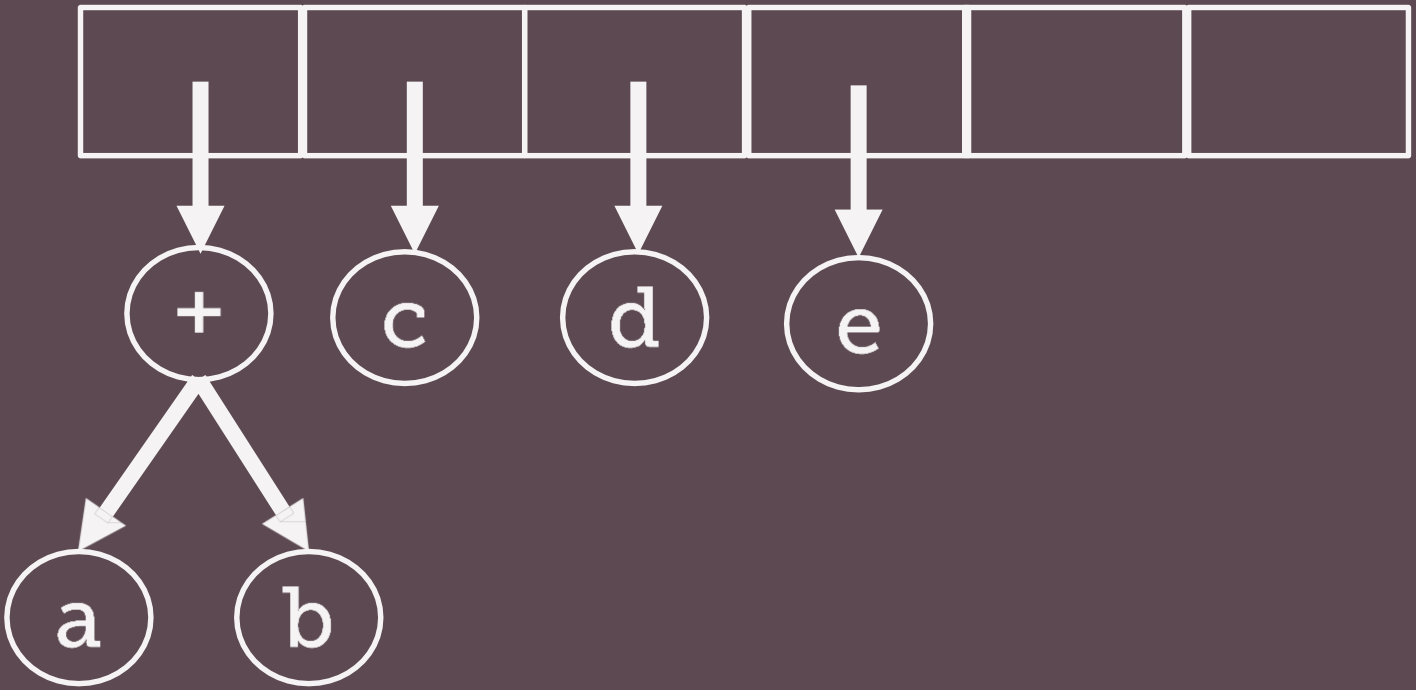
a b + c d e + * *



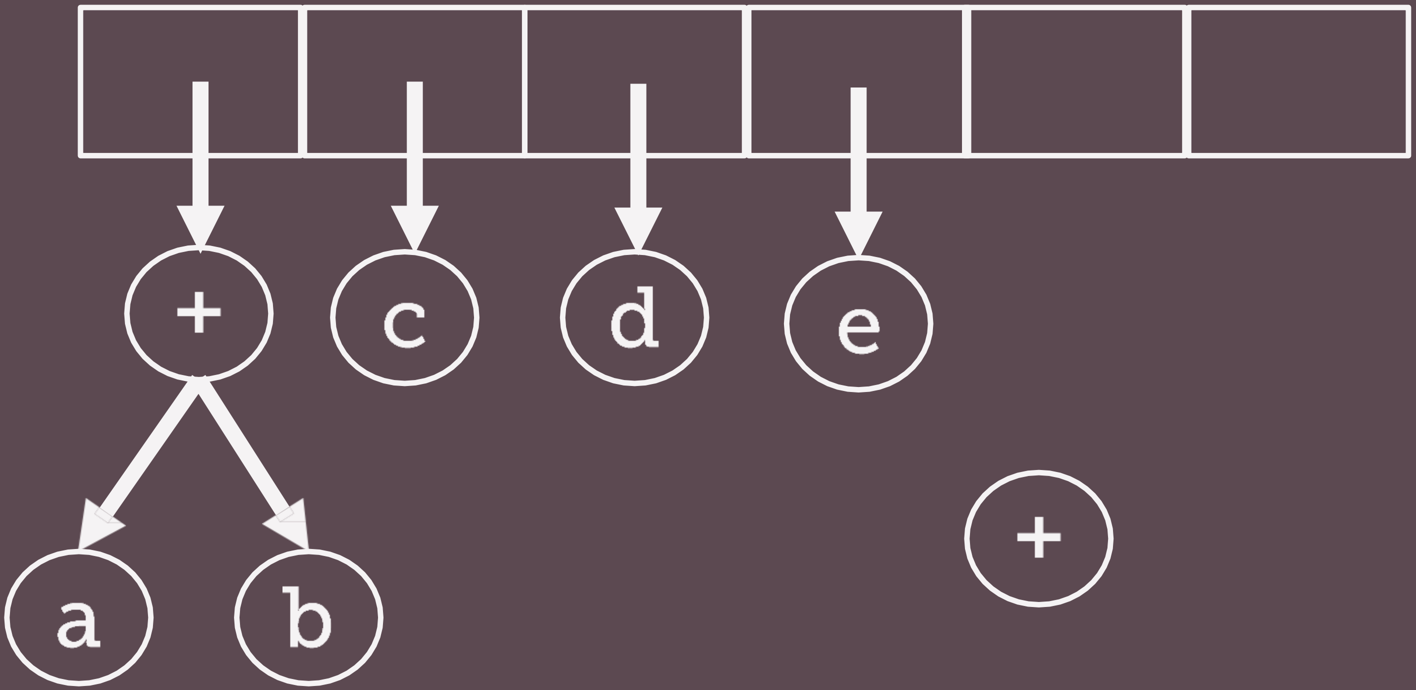
a b + c d e + * *



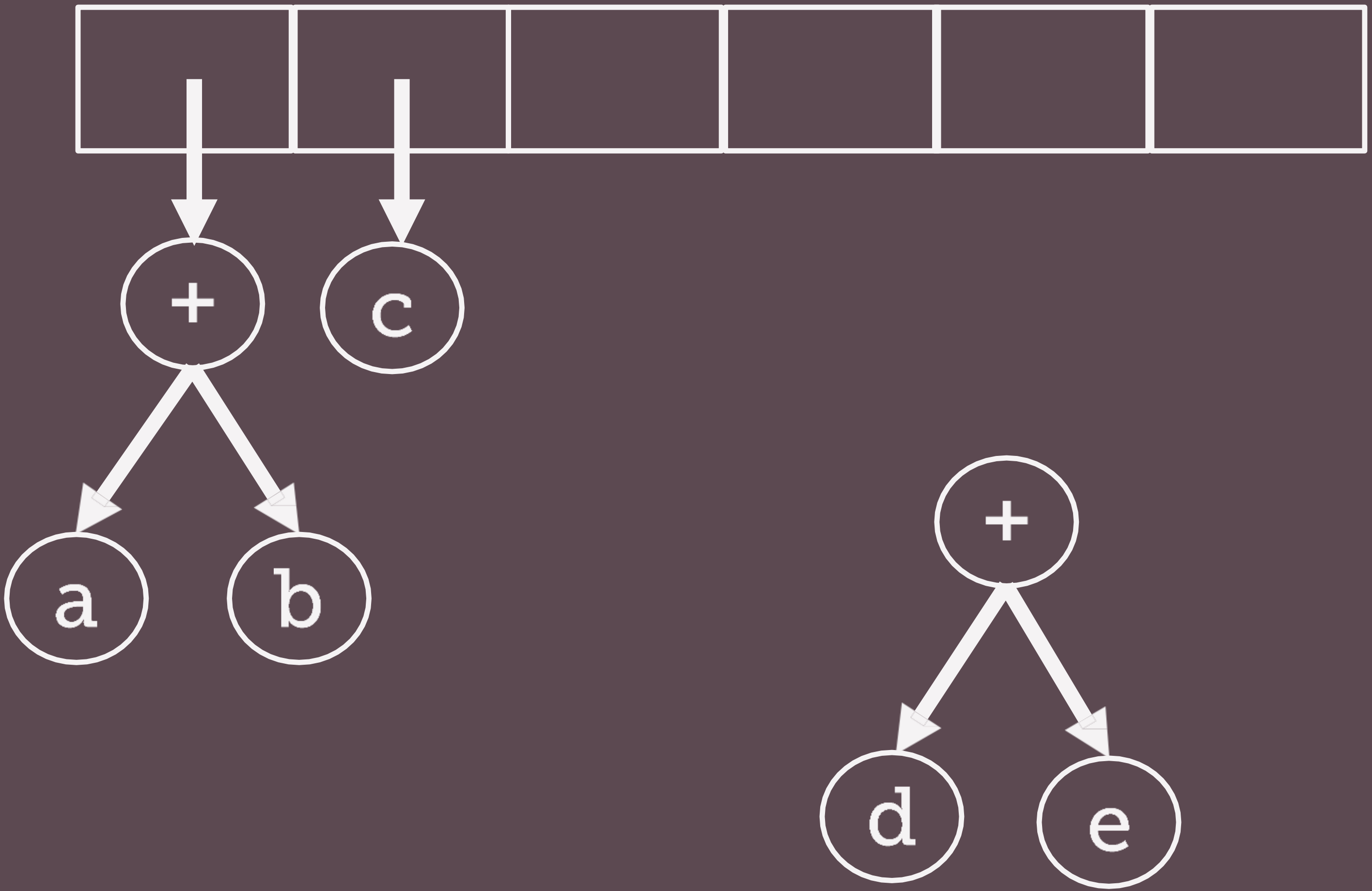
a b + c d e + * *



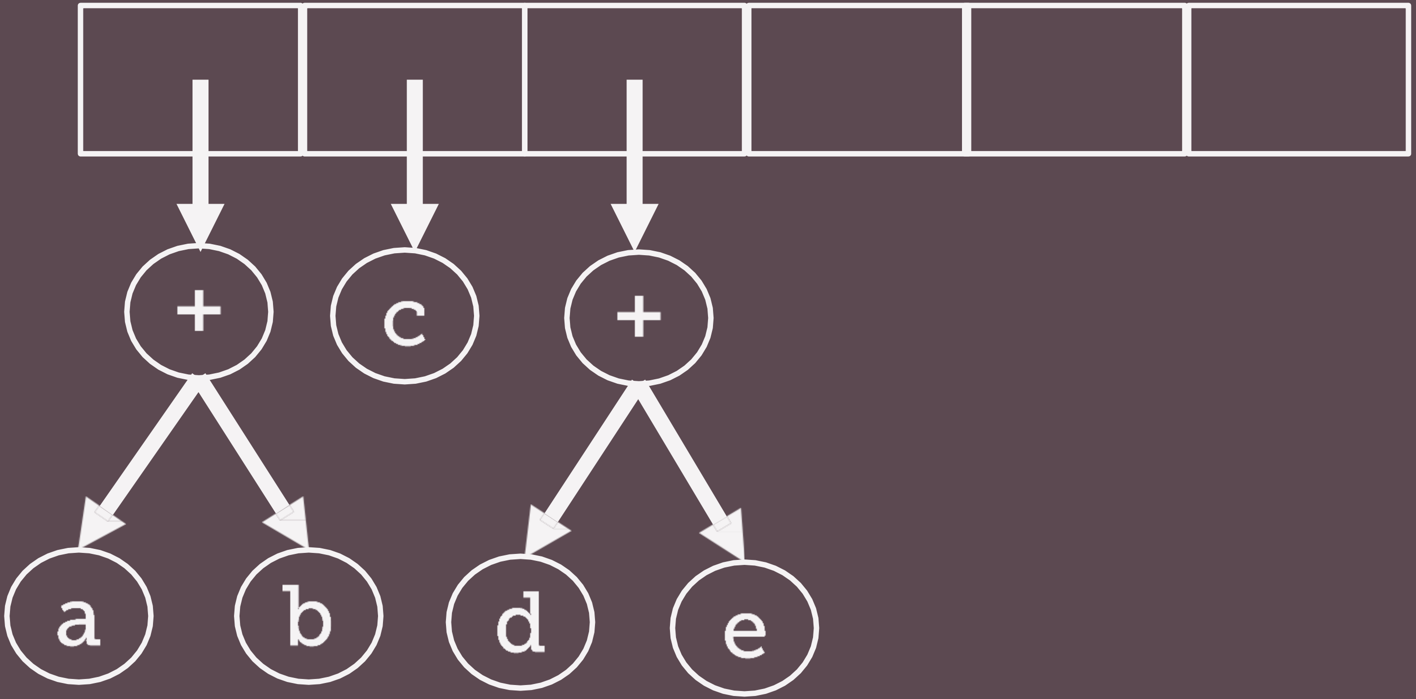
a b + c d e + * *



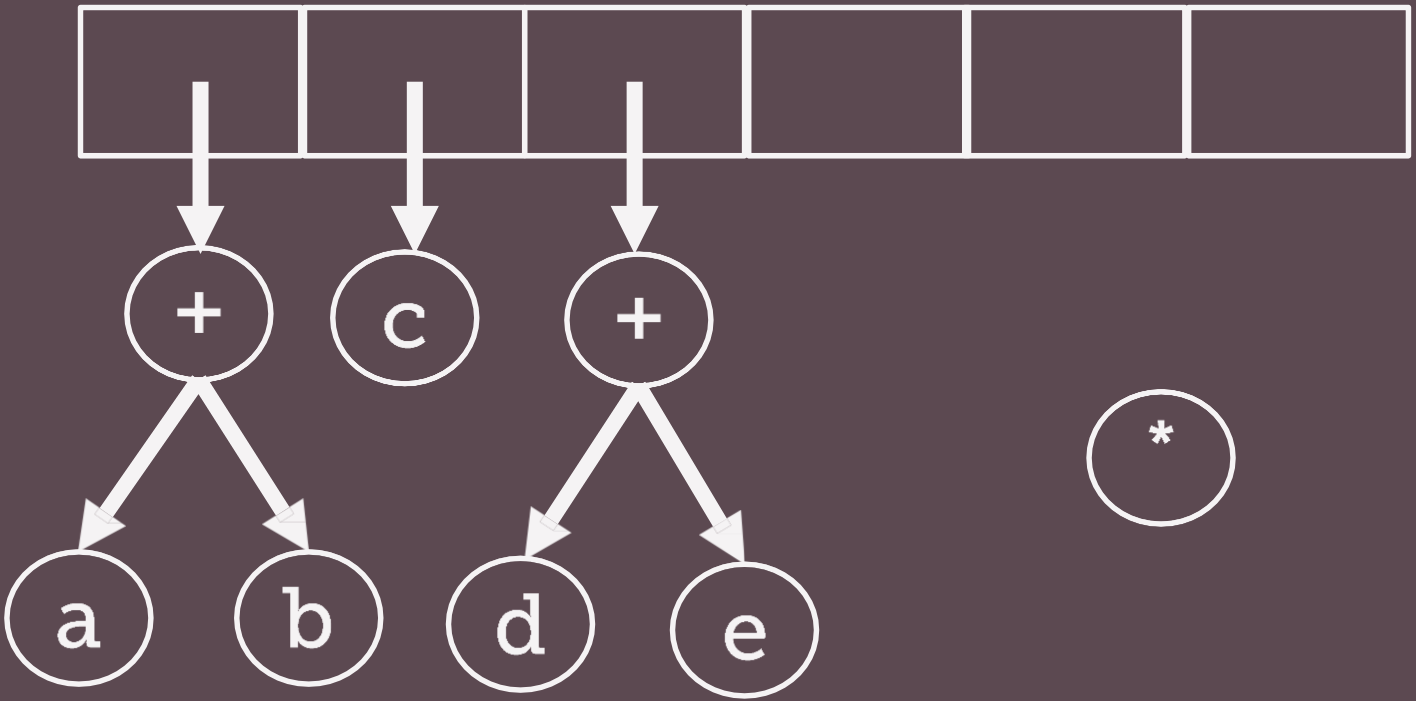
a b + c d e + * *



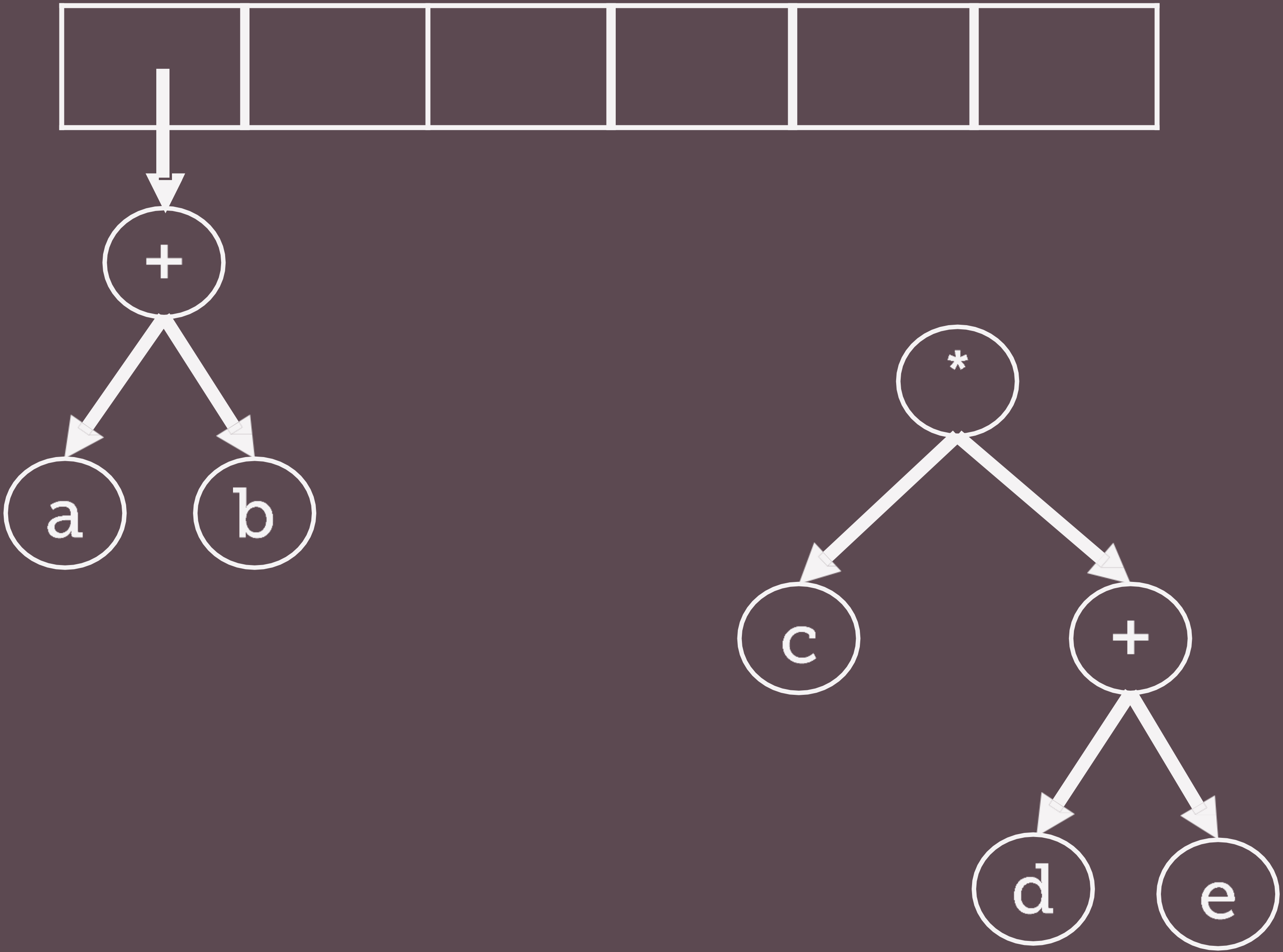
a b + c d e + * *



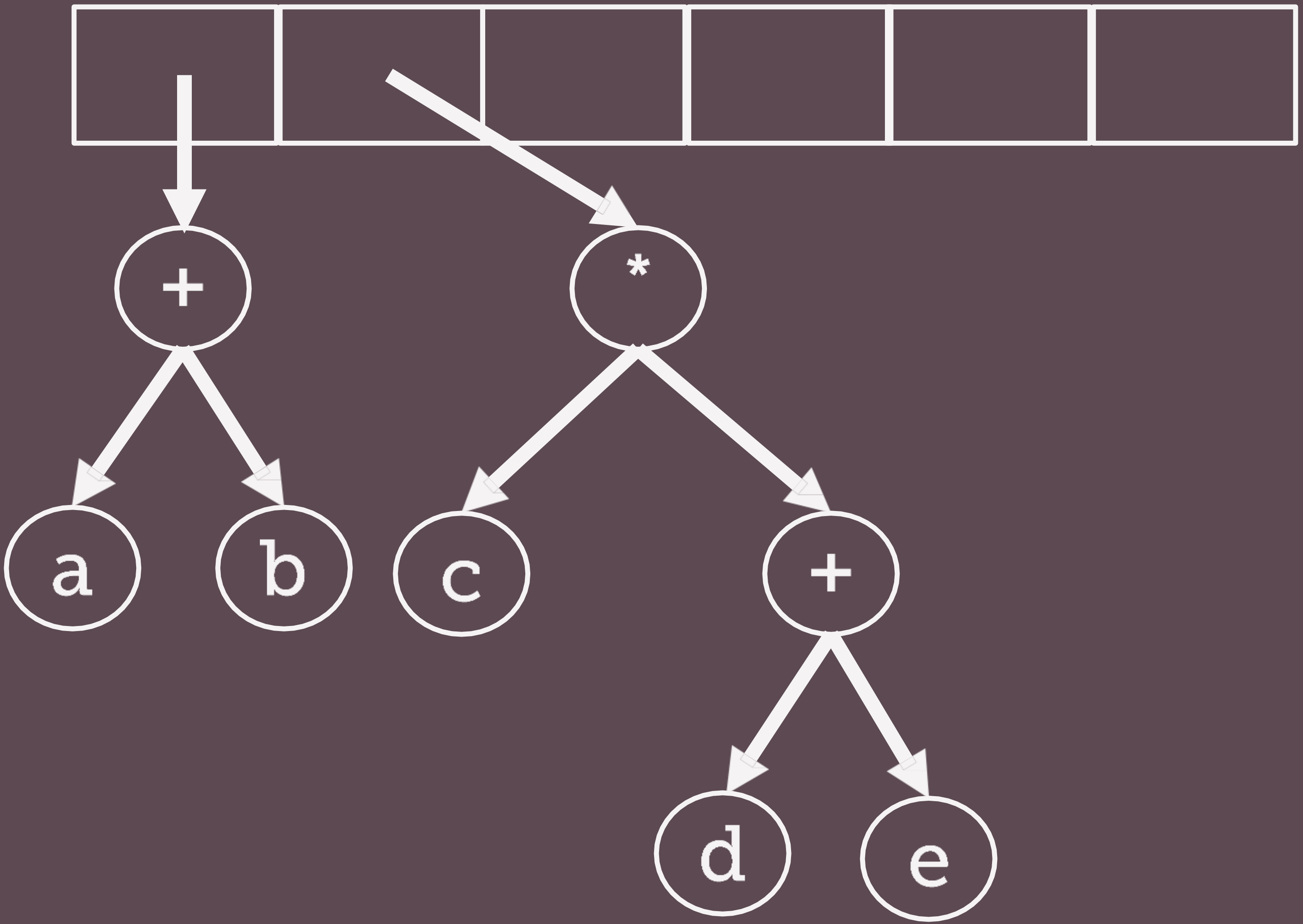
a b + c d e + * *



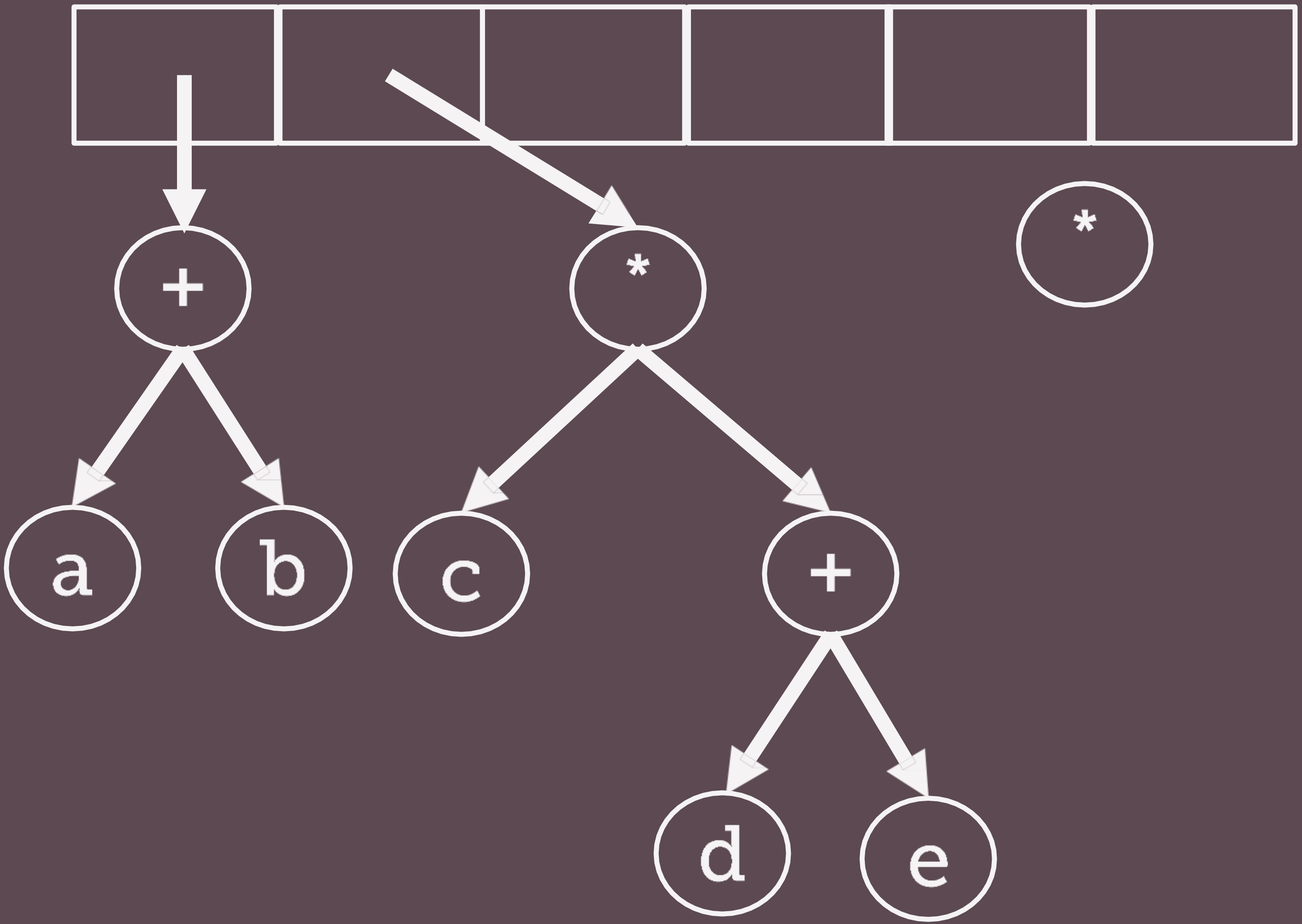
a b + c d e + * *



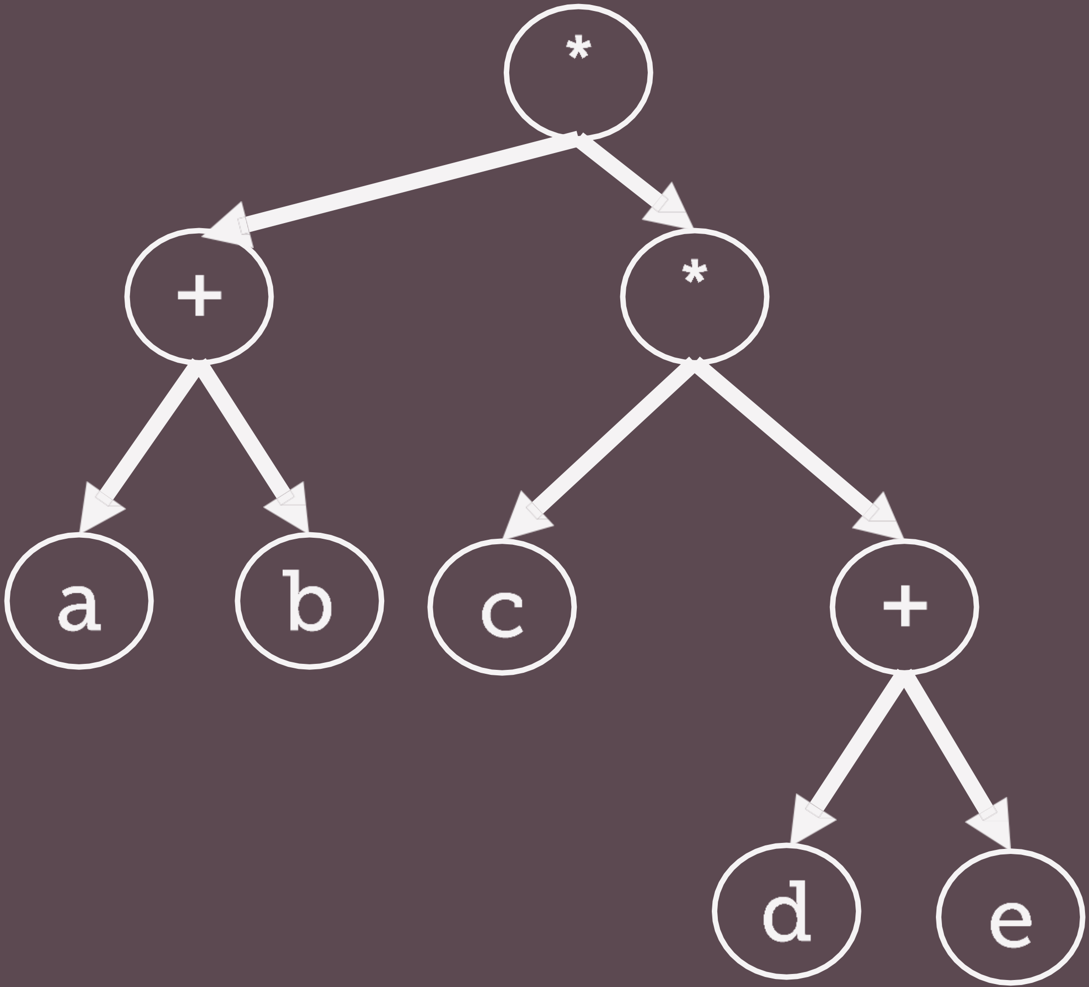
a b + c d e + * *



a b + c d e + * *



a b + c d e + * *



a b + c d e + * *