

Our Solution(s)

Run Code

Your Solutions

Run Code

Solution 1

Solution 2

```
1 // Copyright © 2020 AlgoExpert, LLC. All rights reserved.
2
3 #include <vector>
4 using namespace std;
5
6 class BinaryTreeNode {
7 public:
8     int value;
9     BinaryTreeNode *left = NULL;
10    BinaryTreeNode *right = NULL;
11
12    BinaryTreeNode(int value);
13 };
14
15 vector<BinaryTreeNode*> getNodesInOrder(BinaryTreeNode *tree,
16                                       vector<BinaryTreeNode*> *array);
17
18 // O(n) time | O(n) space - where n is the number of nodes in the Bina
19 BinaryTreeNode *flattenBinaryTree(BinaryTreeNode *root) {
20     vector<BinaryTreeNode*> inOrderNodes =
21         getNodesInOrder(root, new vector<BinaryTreeNode*>{});
22     for (int i = 0; i < inOrderNodes.size() - 1; i++) {
23         BinaryTreeNode *leftNode = inOrderNodes[i];
24         BinaryTreeNode *rightNode = inOrderNodes[i + 1];
25         leftNode->right = rightNode;
26         rightNode->left = leftNode;
27     }
28     return inOrderNodes[0];
29 }
30
31 vector<BinaryTreeNode*> getNodesInOrder(BinaryTreeNode *tree,
32                                       vector<BinaryTreeNode*> *array) {
33     if (tree != NULL) {
```

Solution 1

Solution 2

Solution 3

```
1 #include <vector>
2 using namespace std;
3
4 // This is the class of the input root. Do not edit it.
5 class BinaryTreeNode {
6 public:
7     int value;
8     BinaryTreeNode *left = NULL;
9     BinaryTreeNode *right = NULL;
10
11    BinaryTreeNode(int value);
12 };
13
14 BinaryTreeNode *flattenBinaryTree(BinaryTreeNode *root) {
15     // Write your code here.
16     return root;
17 }
18
```

Our Tests

Custom Output

Submit Code

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

```

1  class PageRanker:
2      def __init__(self, graph):
3          self.graph = graph
4          self.pagerank = {}
5          self.damping = 0.85
6          self.max_iter = 100
7          self.epsilon = 1e-6
8
9      def calculate_pagerank(self):
10         # Initialize pagerank values
11         num_nodes = len(self.graph.nodes)
12         for node in self.graph.nodes:
13             self.pagerank[node] = 1.0 / num_nodes
14
15         # Iterate until convergence
16         for i in range(self.max_iter):
17             new_pagerank = {}
18             for node in self.graph.nodes:
19                 in_degree = self.graph.in_degree(node)
20                 if in_degree == 0:
21                     new_pagerank[node] = 0.0
22                 else:
23                     total_pagerank = 0.0
24                     for neighbor in self.graph.neighbors(node):
25                         total_pagerank += self.pagerank[neighbor]
26                     new_pagerank[node] = (self.damping * total_pagerank) / in_degree
27
28             # Check for convergence
29             diff = 0.0
30             for node in self.graph.nodes:
31                 diff += abs(self.pagerank[node] - new_pagerank[node])
32             self.pagerank = new_pagerank
33             if diff < self.epsilon:
34                 break
35
36     def get_pagerank(self, node):
37         return self.pagerank.get(node, 0.0)

```

Run or submit code when you're ready.