

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 BinaryTree {
7 public:
8     int value;
9     BinaryTree *left = NULL;
10    BinaryTree *right = NULL;
11
12    BinaryTree(int value);
13 };
14
15 vector<BinaryTree *> flattenTree(BinaryTree *node);
16 void connectNodes(BinaryTree *one, BinaryTree *two);
17 BinaryTree *getLeftMost(BinaryTree *node);
18
19 // O(n) time | O(d) space - where n is the number of nodes in the Bina
20 // and d is the depth (height) of the Binary Tree
21 BinaryTree *flattenBinaryTree(BinaryTree *root) {
22     flattenTree(root);
23     return getLeftMost(root);
24 }
25
26 vector<BinaryTree *> flattenTree(BinaryTree *node) {
27     BinaryTree *leftMost;
28     BinaryTree *rightMost;
29
30     if (node->left == NULL) {
31         leftMost = node;
32     } else {
33         vector<BinaryTree *> leftAndRightMostNodes = flattenTree(node->lef
```

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 BinaryTree {
6 public:
7     int value;
8     BinaryTree *left = NULL;
9     BinaryTree *right = NULL;
10
11    BinaryTree(int value);
12 };
13
14 BinaryTree *flattenBinaryTree(BinaryTree *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.tolerance = 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_links = self.graph.in_edges(node)
20                 total_incoming = sum(self.pagerank[src] for src, _ in in_links)
21                 new_pagerank[node] = (1 - self.damping) / num_nodes + self.damping * total_incoming
22
23             # Check for convergence
24             max_diff = 0
25             for node in self.graph.nodes:
26                 diff = abs(self.pagerank[node] - new_pagerank[node])
27                 max_diff = max(max_diff, diff)
28
29             self.pagerank = new_pagerank
30             if max_diff < self.tolerance:
31                 break
32
33     def get_pagerank(self):
34         return self.pagerank
35
36     def __str__(self):
37         return str(self.pagerank)
38
39     def __repr__(self):
40         return f'PageRanker({self.graph}, {self.pagerank})'

```

Run or submit code when you're ready.