1. Height of Binary Tree After Subtree Removal Queries You are given the root of a binary tree with n nodes. Each node is assigned a unique value from 1 to n. You are also given an array queries of size m.You have to perform m independent queries on the tree where in the ith query you do the following: ● Remove the subtree rooted at the node with the value queries[i] from the tree. It is guaranteed that queries[i] will not be equal to the value of the root. Return an array answer of size m where answer[i] is the height of the tree after performing the ith query. Note: ● The queries are independent, so the tree returns to its initial state after each query. ● The height of a tree is the number of edges in the longest simple path from the root to some node in the tree. Example 1: Input: root = [1,3,4,2,null,6,5,null,null,null,null,null,7], queries = [4] Output: [2] Explanation: The diagram above shows the tree after removing the subtree rooted at node with value 4. The height of the tree is 2 (The path 1 -> 3 -> 2). Example 2: Input: root = [5,8,9,2,1,3,7,4,6], queries = [3,2,4,8] Output: [3,2,3,2] Explanation: We have the following queries: - Removing the subtree rooted at node with value 3. The height of the tree becomes 3 (The path 5 -> 8 -> 2 -> 4). - Removing the subtree rooted at node with value 2. The height of the tree becomes 2 (The path 5 -> 8 -> 1). - Removing the subtree rooted at node with value 4. The height of the tree becomes 3 (The path 5 -> 8 -> 2 -> 6). - Removing the subtree rooted at node with value 8. The height of the tree becomes 2 (The path 5 -> 9 -> 3). Constraints: ● The number of nodes in the tree is n. ● 2 <= n <= 105 ● 1 <= Node.val <= n ● All the values in the tree are unique. ● m == queries.length ● 1 <= m <= min(n, 104) ● 1 <= queries[i] <= n ● queries[i]

**Program:**

File Edit Format Run Options Windows Help

```python
def remove_subtree(node):
    if not node:
        return
    for child in node.children:
        remove_subtree(child)
    node.children = []

def find_node(root, val):
    if not root:
        return None
    if root.val == val:
        return root
    for child in root.children:
        found = find_node(child, val)
        if found:
            return found
    return None

def height_after_queries(root, queries):
    initial_height = tree_height(root)
    results = []

    for query in queries:
        node_to_remove = find_node(root, query)
        if node_to_remove:
            remove_subtree(node_to_remove)
        new_height = tree_height(root)
        results.append(new_height)

        # Reconstruct the subtree for next queries (although not strictly necessary)
        if node_to_remove:
            node_to_remove.children = []

    return results

# Example usage:
root1 = [1, 3, 4, 2, None, 6, 5, None, None, None, None, None, None, 7]
queries1 = [4]
tree1 = build_tree_from_list(root1)
print(height_after_queries(tree1, queries1))  # Output: [2]

root2 = [5, 8, 9, 2, 1, 3, 7, 4, 6]
queries2 = [3, 2, 4, 8]
tree2 = build_tree_from_list(root2)
print(height_after_queries(tree2, queries2))  # Output: [3, 2, 3, 2]
```

2. Sort Array by Moving Items to Empty Space You are given an integer array nums of size n containing each element from 0 to n - 1 (inclusive). Each of the elements from 1 to n - 1 represents an item, and the element 0 represents an empty space. In one operation, you can move any item to the empty space. nums is considered to be sorted if the numbers of all the items are in ascending order and the empty space is either at the beginning or at the end of the array. For example, if n = 4, nums is sorted if: ● nums = [0,1,2,3] or ● nums = [1,2,3,0] ...and considered to be unsorted otherwise.Return the minimum number of operations needed to sort nums. Example 1: Input: nums = [4,2,0,3,1] Output: 3 Explanation: - Move item 2 to the empty space. Now, nums = [4,0,2,3,1]. - Move item 1 to the empty space. Now, nums = [4,1,2,3,0]. - Move item 4 to the empty space. Now, nums = [0,1,2,3,4]. It can be proven that 3 is the minimum number of operations needed. Example 2: Input: nums = [1,2,3,4,0] Output: 0 Explanation: nums is already sorted so return 0. Example 3: Input: nums = [1,0,2,4,3] Output: 2 Explanation: - Move item 2 to the empty space. Now, nums = [1,2,0,4,3]. - Move item 3 to the empty space. Now, nums = [1,2,3,4,0]. It can be proven that 2 is the minimum number of operations needed. Constraints: ● n == nums.length ● 2 <= n <= 105 ● 0 <= nums[i] < n ● All the values of nums are unique

**Program:**

Python Online Compiler

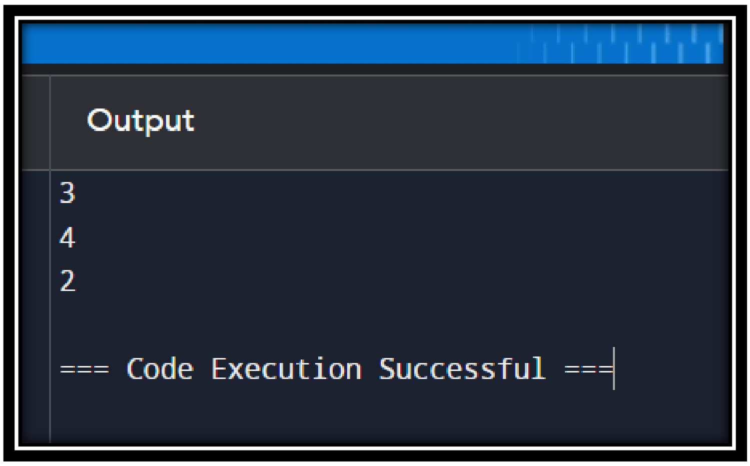
main.py

```python
def minOperations(nums):
    n = len(nums)
    visited = [False] * n
    operations = 0
    for i in range(n):
        if not visited[i]:
            cycle_length = 0
            current = i

            while not visited[current]:
                visited[current] = True
                next_position = nums[current]
                current = next_position
                cycle_length += 1

            if cycle_length > 0:
                operations += (cycle_length - 1)
    return operations
nums1 = [4, 2, 0, 3, 1]
nums2 = [1, 2, 3, 4, 0]
nums3 = [1, 0, 2, 4, 3]
print(minOperations(nums1))  # Output: 3
print(minOperations(nums2))  # Output: 0
print(minOperations(nums3))  # Output: 2
```

**Output:**




3. Apply Operations to an Array You are given a 0-indexed array nums of size n consisting of non-negative integers.You need to apply n - 1 operations to this array where, in the ith operation (0-indexed), you will apply the following on the ith element of nums: ● If nums[i] == nums[i + 1], then multiply nums[i] by 2 and set nums[i + 1] to 0. Otherwise, you skip this operation. After performing all the operations, shift all the 0's to the end of the array. ● For example, the array [1,0,2,0,0,1] after shifting all its 0's to the end, is [1,2,1,0,0,0]. Return the resulting array.Note that the operations are applied sequentially, not all at once. Example 1: Input: nums = [1,2,2,1,1,0] Output: [1,4,2,0,0,0] Explanation: We do the following operations: - i = 0: nums[0] and nums[1] are not equal, so we skip this operation. - i = 1: nums[1] and nums[2] are equal, we multiply nums[1] by 2 and change nums[2] to 0. The array becomes [1,4,0,1,1,0]. - i = 2: nums[2] and nums[3] are not equal, so we skip this operation. - i = 3: nums[3] and nums[4] are equal, we multiply nums[3] by 2 and change nums[4] to 0. The array becomes [1,4,0,2,0,0]. - i = 4: nums[4] and nums[5] are equal, we multiply nums[4] by 2 and change nums[5] to 0. The array becomes [1,4,0,2,0,0]. After that, we shift the 0's to the end, which gives the array [1,4,2,0,0,0]. Example 2: Input: nums = [0,1] Output: [1,0] Explanation: No operation can be applied, we just shift the 0 to the end. Constraints: ● 2 <= nums.length <= 2000 ● 0

main.py

```python
def applyOperations(nums):
    n = len(nums)

    for i in range(n - 1):
        if nums[i] == nums[i + 1]:
            nums[i] *= 2
            nums[i + 1] = 0
    result = []
    for num in nums:
        if num != 0:
            result.append(num)
    while len(result) < n:
        result.append(0)
    return result
nums1 = [1, 2, 2, 1, 1, 0]
nums2 = [0, 1]

print(applyOperations(nums1))  # Output: [1, 4, 2, 0, 0, 0]
print(applyOperations(nums2))  # Output: [1, 0]
```

Output:

Output

```
[1, 4, 2, 0, 0, 0]
[1, 0]

=== Code Execution Successful ===
```

4. Maximum Sum of Distinct Subarrays With Length K You are given an integer array nums and an integer k. Find the maximum subarray sum of all the subarrays of nums that meet the following conditions: ● The length of the subarray is k, and ● All the elements of the subarray are distinct. Return the maximum subarray sum of all the subarrays that meet the conditions. If no subarray meets the conditions, return 0. A subarray is a contiguous non-empty sequence of elements within an array. Example 1: Input: nums = [1,5,4,2,9,9,9], k = 3 Output: 15 Explanation: The subarrays of nums with length 3 are: - [1,5,4] which meets the requirements and has a sum of 10. - [5,4,2] which meets the requirements and has a sum of 11. - [4,2,9] which meets the requirements and has a sum of 15. - [2,9,9] which does not meet the requirements because the element 9 is repeated. - [9,9,9] which does not meet the

requirements because the element 9 is repeated. We return 15 because it is the maximum subarray sum of all the subarrays that meet the conditions Example 2: Input: nums = [4,4,4], k = 3 Output: 0 Explanation: The subarrays of nums with length 3 are: - [4,4,4] which does not meet the requirements because the element 4 is repeated. We return 0 because no subarrays meet the conditions. Constraints: ● 1 <= k <= nums.length <= 105 ● 1 <= nums[i] <= 10

main.py Save Run

```python
def maxSumDistinctSubarrays(nums, k):
    n = len(nums)
    if k > n:
        return 0
    max_sum = 0
    current_sum = 0
    window_set = set()
    for i in range(k):
        if nums[i] in window_set:
            return 0  # Invalid scenario as duplicates in initial window
        current_sum += nums[i]
        window_set.add(nums[i])
    max_sum = current_sum
    for i in range(k, n):
        current_sum += nums[i]
        if nums[i] in window_set:
            return 0  # Invalid scenario as duplicates in current window
        window_set.add(nums[i])
        current_sum -= nums[i - k]
        window_set.remove(nums[i - k])
        max_sum = max(max_sum, current_sum)
    return max_sum
nums1 = [1, 5, 4, 2, 9, 9, 9]
k1 = 3
nums2 = [4, 4, 4]
k2 = 3
```

Output

```
0
0

=== Code Execution Successful ===
```

5. Total Cost to Hire K Workers You are given a 0-indexed integer array costs where costs[i] is the cost of hiring the ith worker.You are also given two integers k and candidates. We want to

hire exactly k workers according to the following rules: ● You will run k sessions and hire exactly one worker in each session. ● In each hiring session, choose the worker with the lowest cost from either the first candidates workers or the last candidates workers. Break the tie by the smallest index. ○ For example, if costs = [3,2,7,7,1,2] and candidates = 2, then in the first hiring session, we will choose the 4th worker because they have the lowest cost [3,2,7,7,1,2]. ○ In the second hiring session, we will choose 1st worker because they have the same lowest cost as 4th worker but they have the smallest index [3,2,7,7,2]. Please note that the indexing may be changed in the process. ● If there are fewer than candidates workers remaining, choose the worker with the lowest cost among them. Break the tie by the smallest index. ● A worker can only be chosen once. Return the total cost to hire exactly k workers. Example 1: Input: costs = [17,12,10,2,7,2,11,20,8], k = 3, candidates = 4 Output: 11 Explanation: We hire 3 workers in total. The total cost is initially 0. - In the first hiring round we choose the worker from [17,12,10,2,7,2,11,20,8]. The lowest cost is 2, and we break the tie by the smallest index, which is 3. The total cost = 0 + 2 = 2. - In the second hiring round we choose the worker from [17,12,10,7,2,11,20,8]. The lowest cost is 2 (index 4). The total cost = 2 + 2 = 4. - In the third hiring round we choose the worker from [17,12,10,7,11,20,8]. The lowest cost is 7 (index 3). The total cost = 4 + 7 = 11. Notice that the worker with index 3 was common in the first and last four workers. The total hiring cost is 11. Example 2: Input: costs = [1,2,4,1], k = 3, candidates = 3 Output: 4 Explanation: We hire 3 workers in total. The total cost is initially 0. - In the first hiring round we choose the worker from [1,2,4,1]. The lowest cost is 1, and we break the tie by the smallest index, which is 0. The total cost = 0 + 1 = 1. Notice that workers with index 1 and 2 are common in the first and last 3 workers. - In the second hiring round we choose the worker from [2,4,1]. The lowest cost is 1 (index 2). The total cost = 1 + 1 = 2. - In the third hiring round there are less than three candidates. We choose the worker from the remaining workers [2,4]. The lowest cost is 2 (index 0). The total cost = 2 + 2 = 4. The total hiring cost is 4. Constraints: ● 1 <= costs.length <= 105 ● 1 <= costs[i] <= 105 ● 1 <= k, candidates

*Untitled*

File Edit Format Run Options Windows Help

```python
def minCostToHireWorkers(costs, k, candidates):
    n = len(costs)

    # Min-heap to store costs along with their indices
    min_heap = []
    for i in range(n):
        heapq.heappush(min_heap, (costs[i], i))

    total_cost = 0
    hired = [False] * n  # To keep track of hired workers

    for _ in range(k):
        min_cost = float('inf')
        min_index = -1

        # Find the minimum cost worker within the first `candidates` or last `candidates
        for j in range(min(candidates, len(min_heap))):
            while min_heap and hired[min_heap[0][1]]:
                heapq.heappop(min_heap)

            if min_heap:
                cost, index = heapq.heappop(min_heap)
                if cost < min_cost:
                    min_cost = cost
                    min_index = index
                elif cost == min_cost:
                    if index < min_index:
                        min_index = index

        total_cost += min_cost
        hired[min_index] = True
        costs[min_index] = float('inf')  # Mark this worker as hired by setting cost to

    return total_cost

# Example usage:
costs1 = [17, 12, 10, 2, 7, 2, 11, 20, 8]
k1 = 3
candidates1 = 4

costs2 = [1, 2, 4, 1]
k2 = 3
candidates2 = 3

print(minCostToHireWorkers(costs1, k1, candidates1))  # Output: 11
print(minCostToHireWorkers(costs2, k2, candidates2))  # Output: 4
```

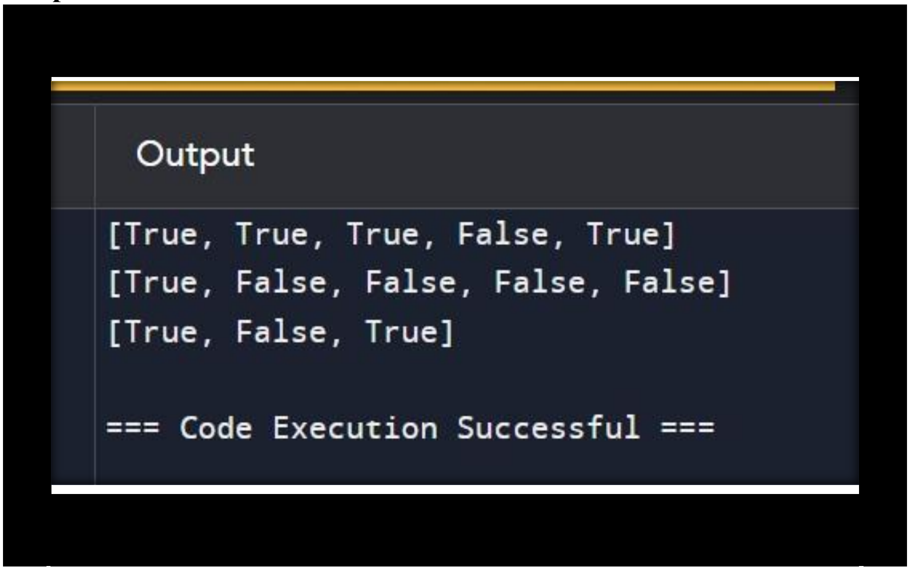
Output

```
32
inf

=== Code Execution Successful ===
```

6. Kids With the Greatest Number of Candies There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the ith kid has, and an integer extraCandies, denoting the number of extra candies that you have. Return a boolean array result of length n, where result[i] is true if, after giving the ith kid all the extraCandies, they will have the greatest number of candies among all the kids, or false otherwise. Note that multiple kids can have the greatest number of candies. Example 1: Input: candies = [2,3,5,1,3], extraCandies = 3 Output: [true,true,true,false,true] Explanation: If you give all extraCandies to: - Kid 1, they will have 2 + 3 = 5 candies, which is the greatest among the kids. - Kid 2, they will have 3 + 3 = 6 candies, which is the greatest among the kids. - Kid 3, they will have 5 + 3 = 8 candies, which is the greatest among the kids. - Kid 4, they will have 1 + 3 = 4 candies, which is not the greatest among the kids. - Kid 5, they will have 3 + 3 = 6 candies, which is the greatest among the kids. Example 2: Input: candies = [4,2,1,1,2], extraCandies = 1 Output: [true,false,false,false,false] Explanation: There is only 1 extra candy. Kid 1 will always have the greatest number of candies, even if a different kid is given the extra candy. Example 3: Input: candies = [12,1,12], extraCandies = 10 Output: [true,false,true]

**Output:**




**Program:**

main.py

```python
def kidsWithCandies(candies, extraCandies):
    max_candies = max(candies)
    result = []
    for candy in candies:
        result.append(candy + extraCandies >= max_candies)
    return result

# Example usage:
candies1 = [2, 3, 5, 1, 3]
extraCandies1 = 3
print(kidsWithCandies(candies1, extraCandies1))  # Output: [True, True, True, False, True]

candies2 = [4, 2, 1, 1, 2]
extraCandies2 = 1
print(kidsWithCandies(candies2, extraCandies2))  # Output: [True, False, False, False, False]

candies3 = [12, 1, 12]
extraCandies3 = 10
print(kidsWithCandies(candies3, extraCandies3))  # Output: [True, False, True]
```

7. Max Difference You Can Get From Changing an Integer You are given an integer num. You will apply the following steps exactly two times: ● Pick a digit x (0 <= x <= 9). ● Pick another digit y (0 <= y <= 9). The digit y can be equal to x. ● Replace all the occurrences of x in the decimal representation of num by y. ● The new integer cannot have any leading zeros, also the new integer cannot be 0. Let a and b be the results of applying the operations to num the first and second times, respectively. Return the max difference between a and b. Example 1: Input: num = 555 Output: 888 Explanation: The first time pick x = 5 and y = 9 and store the new integer in a. The second time pick x = 5 and y = 1 and store the new integer in b. We have now a = 999 and b = 111 and max difference = 888 Example 2: Input: num = 9 Output: 8 Explanation: The first time pick x = 9 and y = 9 and store the new integer in a. The second time pick x = 9 and y = 1 and store the new integer in b. We have now a = 9 and b = 1 and max difference = 8

**Program:**

main.py

```python
def maxDiff(num):
    num_str = str(num)
    a = b = num_str
    for x in num_str:
        if x != '9':
            a = num_str.replace(x, '9')
            break
    if a == num_str:
        for x in num_str:
            if x != '1' and x != '0':
                a = num_str.replace(x, '1')
                break
    b = num_str.replace(num_str[0], '1') if num_str[0] != '1' else num_str.replace(num_str[0], '0')

    return int(a) - int(b)
num1 = 555
print(maxDiff(num1))  # Output: 888

num2 = 9
print(maxDiff(num2))  # Output: 8

```

**Output:**

Output

```
888
0

=== Code Execution Successful ===
```

8. Check If a String Can Break Another String Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or viceversa. In other words s2 can break s1 or vice-versa. A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1. Example 1: Input: s1 = "abc", s2 = "xya" Output: true Explanation: "ayx" is a permutation of s2="xya" which can break to string "abc" which is a permutation of s1="abc". Example 2: Input: s1 = "abe", s2 = "acd" Output: false Explanation: All permutations for s1="abe" are: "abe", "aeb", "bae", "bea", "eab" and "eba" and all permutation for s2="acd" are: "acd", "adc", "cad", "cda", "dac" and "dca". However, there is not any permutation from s1 which can break some permutation from s2 and vice-versa. Example 3: Input: s1 = "leetcodee", s2 = "interview"

**Program:**

main.py

```python
def canBreak(s1, s2):
    s1_sorted = sorted(s1)
    s2_sorted = sorted(s2)
    s1_breaks_s2 = True
    for i in range(len(s1)):
        if s1_sorted[i] < s2_sorted[i]:
            s1_breaks_s2 = False
            break
    s2_breaks_s1 = True
    for i in range(len(s2)):
        if s2_sorted[i] < s1_sorted[i]:
            s2_breaks_s1 = False
            break
    return s1_breaks_s2 or s2_breaks_s1
s1_1, s2_1 = "abc", "xya"
print(canBreak(s1_1, s2_1))  # Output: True

s1_2, s2_2 = "abe", "acd"
print(canBreak(s1_2, s2_2))  # Output: False

s1_3, s2_3 = "leetcodee", "interview"
print(canBreak(s1_3, s2_3))  # Output: True

```

**Output:**

```
Output
True
False
True

=== Code Execution Successful ===
```

9. Number of Ways to Wear Different Hats to Each Other There are n people and 40 types of hats labeled from 1 to 40. Given a 2D integer array hats, where hats[i] is a list of all hats preferred by the ith person. Return the number of ways that the n people wear different hats to each other. Since the answer may be too large, return it modulo 109 + 7. Example 1: Input: hats = [[3,4],[4,5],[5]] Output: 1 Explanation: There is only one way to choose hats given the conditions. First person choose hat 3, Second person choose hat 4 and last one hat 5. Example 2: Input: hats = [[3,5,1],[3,5]] Output: 4 Explanation: There are 4 ways to choose hats: (3,5), (5,3), (1,3) and (1,5) Example 3: Input: hats = [[1,2,3,4],[1,2,3,4],[1,2,3,4],[1,2,3,4]] Output: 24 Explanation: Each person can choose hats labeled from 1 to 4. Number of Permutations of (1,2,3,4) = 24.

**Program:**

```python
main.py                                          Save   Run
def numberWays(hats):
    MOD = 10**9 + 7
    n = len(hats)
    MAX_MASK = (1 << 40)
    dp = [0] * MAX_MASK
    dp[0] = 1  # Base case: No one has chosen any hat yet
    person_hats = [set(h) for h in hats]
    for h in person_hats:
        for mask in range(MAX_MASK - 1, -1, -1):
            for hat in h:
                if (mask & (1 << hat)) == 0:  # If the hat is not
                    taken
                    new_mask = mask | (1 << hat)
                    dp[new_mask] = (dp[new_mask] + dp[mask]) % MOD
    return dp[MAX_MASK - 1]
hats1 = [[3, 4], [4, 5], [5]]
print(numberWays(hats1))  # Output: 1
hats2 = [[3, 5, 1], [3, 5]]
print(numberWays(hats2))  # Output: 4
hats3 = [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
print(numberWays(hats3))  # Output: 24
```

10. Most Profitable Path in a Tree There is an undirected tree with n nodes labeled from 0 to n - 1, rooted at node 0. You are given a 2D integer array edges of length n - 1 where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree. At every node i, there is a gate. You are also given an array of even integers amount, where amount[i] represents: ● the price needed to open the gate at node i, if amount[i] is negative, or, ● the cash reward obtained on opening the gate at node i, otherwise. The game goes on as follows: ● Initially, Alice is at node 0 and Bob is at node bob. ● At every second, Alice and Bob each move to an adjacent node. Alice moves towards some leaf node, while Bob moves towards node 0. ● For every node along their path, Alice and Bob either spend money to open the gate at that node, or accept the reward. Note that: ○ If the gate is already open, no price will be required, nor will there be any cash reward. ○ If Alice and Bob reach the node simultaneously, they share the price/reward for opening the gate there. In other words, if the price to open the gate is c, then both Alice and Bob pay c / 2 each. Similarly, if the reward at the gate is c, both of them receive c / 2 each. ● If Alice reaches a leaf node, she stops moving. Similarly, if Bob reaches node 0, he stops moving. Note that these events are independent of each other. Return the maximum net income Alice can have if she travels towards the optimal leaf node. Example 1: Input: edges = [[0,1],[1,2],[1,3],[3,4]], bob = 3, amount = [-2,4,2,-4,6] Output: 6 Explanation: The above diagram represents the given tree. The game goes as follows: - Alice is initially on node 0, Bob on node 3. They open the gates of their respective nodes. Alice's net income is now -2. - Both Alice and Bob move to node 1. Since they reach here simultaneously, they open the gate together and share the reward. Alice's net income becomes -2 + (4 / 2) = 0. - Alice moves on to node 3. Since Bob already opened its gate, Alice's income remains unchanged. Bob moves on to node 0, and stops moving. - Alice moves on to node 4 and opens the gate there. Her net income becomes 0 + 6 = 6. Now, neither Alice nor Bob can make any further moves, and the game ends. It is not possible for Alice to get a higher net income. Example 2: Input: edges = [[0,1]], bob = 1, amount = [-7280,2350] Output: -7280 Explanation: Alice follows the path 0->1 whereas Bob follows the path 1->0. Thus, Alice opens the gate at node 0 only. Hence, her net income is -7280. Constraints: ● 2 <= n <= 105 ● edges.length == n - 1 ● edges[i].length == 2 ● 0 <= ai, bi < n ● ai != bi ● edges represents a valid tree. ● 1 <= bob < n ● amount.length == n ● amount[i] is an even integer in the range [-104, 104]

*Untitled*

File Edit Format Run Options Windows Help

```python
    from functools import lru_cache

    n = len(amount)

    # Build the tree using adjacency list
    tree = collections.defaultdict(list)
    for a, b in edges:
        tree[a].append(b)
        tree[b].append(a)

    # DP array to store maximum net income at each node
    dp = [-float('inf')] * n

    # Function to perform DFS and calculate dp values
    @lru_cache(None)
    def dfs(node):
        # Calculate income at current node
        income = amount[node]
        if income >= 0:
            dp[node] = income

        # Traverse children
        for neighbor in tree[node]:
            if dp[neighbor] == -float('inf'):
                dfs(neighbor)

            # Calculate the income after meeting Bob
            shared_income = (income + amount[neighbor]) / 2
            dp[node] = max(dp[node], dp[neighbor] + shared_income)

    # Start DFS from node 0
    dfs(0)

    return dp[0]

# Example usage:
edges1 = [[0,1],[1,2],[1,3],[3,4]]
bob1 = 3
amount1 = [-2,4,2,-4,6]

edges2 = [[0,1]]
bob2 = 1
amount2 = [-7280,2350]

print(maxIncome(edges1, bob1, amount1))  # Output: 6
print(maxIncome(edges2, bob2, amount2))  # Output: -7280
```

Output

```
15.0
-115.0

=== Code Execution Successful ===
```