Assignment 03

Instructions

1. Write your functions according to the signature provided below in a python(.py) file and submit them to the autograder for evaluation.
2. The autograder has a limited number of attempts. Hence, test it thoroughly before submitting it for evaluation.
3. Save your python file as text(.txt) file and submit it to the canvas.
4. If the assignment has theoretical questions, upload those answers as a pdf file on the canvas.
5. Submission on the canvas is mandatory and the canvas submitted text file should be the same as that of the final submission of the autograder. If not, marks will be deducted.
6. No submission on canvas or no submission on autograder then marks will be deducted.
7. Access the auto grader at https://c200.luddy.indiana.edu.

Question 1

Once upon a time in a small, close-knit village, there were n friends who loved to play a traditional game called "Circle of Friendship." The game was a cherished tradition in their village, played every year during the annual Friendship Festival. The friends would gather in a large circle in the village square, each one assigned a unique number(consecutive) from 1 to n in clockwise order. They held hands, forming a circle of unity, and the game would begin with laughter and excitement. The rules of the game were simple.

- They would start with the 1st friend and count the next k friends in the clockwise direction, including the friend they started at. The counting would wrap around the circle, potentially counting some friends more than once. The friend they landed on after counting k steps would leave the circle and sadly, is eliminated from the game.
- If there were still more than one friend left in the circle, they would go back to above step, starting from the friend immediately clockwise of the eliminated friend, and repeat.
- This process would continue until only one friend remained in the circle, and that friend would be declared the winner of the Friendship Festival.

Your task is to write a Python function, who_wins (n, k), that takes two integers as input: - n: The number of friends in the circle. - k: The number of friends to count before eliminating one.

The function should return the number of the friend who wins the game according to the rules above.

Example

If n = 5 and k = 2, the game would proceed as follows:

- Start at friend 1.
- Count 2 friends clockwise: Friend 1 → Friend 2 → Eliminate Friend 2.
- Start at the next friend clockwise, which is Friend 3.
- Count 2 friends clockwise: Friend 3 → Friend 4 → Eliminate Friend 4.
- Start at the next friend clockwise, which is Friend 5.
- Count 2 friends clockwise: Friend 5 → Friend 1 → Eliminate Friend 1.
- Start at the next friend clockwise, which is Friend 3.

- Count 2 friends clockwise: Friend 3 → Friend 5 → Eliminate Friend 5.

Only one friend, Friend 3, remains, so Friend 3 wins the game.

Constraints

1 <= k <= n

Function

```
def who_wins(n, k):

  #Your code here

  return winner
```

Question 2

In the charming town of Willowbrook, a young student named Lily lived. This wasn't just any ordinary town; it was magical in the truest sense. Every day, the town's buildings moved mysteriously, changing their location coordinates (x, y) as if they had a life of their own. Lily, like any diligent student, had to attend school on weekdays. However, the ever-shifting nature of Willowbrook posed a unique challenge. In this mystical town, anyone currently located at coordinates (x, y) could take only two types of moves in the next step: either (x+y, y) or (x, x+y). These moves had to be carefully chosen to navigate the ever-changing town. The young student's home was situated at coordinates (x1, y1), while the school was located at coordinates (x2, y2). The question that intrigued the residents of Willowbrook was this: would Lily be able to reach school from home, given the daily shifting of coordinates?

For N days, you are tasked with solving this intriguing problem. You will be provided with the home coordinates (x1, y1) and the school coordinates (x2, y2) of Lily for each of these days. Your mission is to determine for how many days Lily will be able to successfully reach the school, defying the whims of Willowbrook's magical coordinates.

Examples

Example 1:

```
Input: home_coords = [[1,1],[2,4]], school_coords = [[4,5],[2,5]], N=2
Output: 1
```

Explanation: For the 1st case, Lily starts at home_coord = [1,1] and school_coord = [4,5]. Lily chooses the path: [1,1] -> [2,1] -> [3,1] -> [4,1] -> [4,5], successfully reaching the school. However, in the 2nd case, starting from home_coord = [2,4] and school_coord = [2,5], no combination of moves can align the y-coordinates, making it impossible for Lily to reach school. Thus, Lily will be able to make it only for the 1st case, resulting in an output of 1.

Example 2:

```
Input: home_coords = [[3,2],[1,7],[4,4]], school_coords = [[8,5],[4,2],[7,7]], N=3
Output: 1
```

Explanation: Lily can make to school only in the 1st case.

Constraints

- x1,y1,x2,y2 > 0

Function

```
def count_successful_school_commutes(home_coords, school_coords, N):
  # your code goes here

  return result
```

Question 3

In the mystical realm of Zenthar Island, the ancient civilization used numbers in a very unique manner. In their number system, the absolute difference between any two adjacent digits in a number had to be a fixed value, "K". As a new explorer on the island, you stumble upon an inscription on a stone tablet that describes this ancient numbering system. Fascinated, you take it upon yourself to decipher the numerical puzzles left behind by the ancient inhabitants.

Legend has it that the sequence of numbers you generate will unlock the secret chamber that holds the ultimate treasure of Zenthar Island. But first, you need to create this sequence.

The Quest: You're tasked with generating all possible numbers of length "N" in the ancient Zenthar number system in an ascending order and return the list of numbers. Remember, in this unique system, the absolute difference between adjacent digits must be exactly "K".

Write a Python function zenthar_puzzle(N,K) to generate and return list of all possible numbers that follows above criteria.

Examples

Example 1:

```
Input:N=3, k=8
Output: [191, 808, 919]
```

Explanation: In this example, The difference between adjacent digits in the output numbers return is 'k'. That is in number 191, 9-1 =8 and in 808, 8-0=8

Example 2:

```
Input:N=4,K=9
Output: [9090]
```

Note:

- A single integer "N" (1 N 9) which represents the length of the numbers you need to generate.
- Another integer "K" (1 K 9) which represents the absolute difference between adjacent digits in the numbers.
- For N=1, the result will be list of all the numbers from '1-9'

Function

```
def zenthar_puzzle( N,K):
    pass
    # it should return list of numbers
    return result
```

Question 4

Santa attended the Gen Con event last month and was absolutely captivated by the vast array of board games on display. Eager to share his experience with his friend Banta, Santa decided to recount every minute detail of his trip to Indianapolis in a lengthy email. However, upon completing the email, he realized that it had become excessively large to send. To address this issue, Santa decided to compress the text by using a specific format: p(compressed_str), where p represents a positive integer denoting the number of repetitions of the compressed string, enclosed within square brackets.

Now, Banta is faced with the task of deciphering the compressed text and restoring it to its original form. Can you assist Banta in extracting the original text?

Examples

Example 1:

```
Input: 2(ab7(c))
Output: abccccccabcccccccc
```

Example 2:

```
Input: 3(ab)2(r)pl
Output: abababrrpl
```

Constraints:

- p > 0
- characters in compressed_str [a-z]
- No whitespaces in compressed_str, no characters other than [a-z] in compressed_str.

Function:

```
def decompress(s):
    # The input s, compressed text sent by Santa, is string

  # Write your code here
  return decompressed_string
```

Question 5

Ron is enrolled in a Biological Manipulation and Organic Magic course taught by Professor Anderson. Professor Anderson has assigned Ron a task that combines both theory and practical skills.

For this task, Ron is given two things:

1. A list of potent values representing different magical potions. The order of these values corresponds to the level-order traversal of a complete binary tree that Ron needs to create.
2. A positive integer, k, which represents the minimum number of "Oracle's Extracts" that should exist in the binary tree Ron creates.

An Oracle's Extract in this context is defined as follows:

A node in the binary tree is considered an Oracle's Extract if, starting from the root node and following the path down to the current node, there are no nodes with a greater value than the current node along that path.

Ron's task is to create a complete binary tree using the given potent values in a level-order manner. After creating the tree, he needs to determine whether there are at least k Oracle's Extracts in the tree. If there are at least k such nodes, Ron should answer "True." Otherwise, he should answer "False."

ASSUMPTION: Always assume that the root node is an Oracle's Extract.

Examples

Example 1:

```
Input: values = [3, 1, 4, 3, 1, 5], k = 2
Output: True
Explanation: In the complete binary tree
Root Node (3) is always a good node.
Node 4 -> (3,4) is the maximum value in the path starting from the root.
Node 5 -> (3,4,5) is the maximum value in the path
Node 3 -> (3,1,3) is the maximum value in the path.
Total number of Oracle's Extracts = 4 which is greater that 2, Result = True
```

Example 2:

```
Input: values = [11, 10, 9, 13, 5, 14, 1], k = 4
Output: False
Explanation: Total number of Oracle's Extracts = 3 which is less that 4, Result = False
```

Constraints:

- Do not use any sorting algorithms for this problem, Do it by using Tree data structure
- The list of potent values is never empty/Null
- Do add the TreeNode class to your py file while submitting

Function

```python
class TreeNode:
    def __init__(self, value: int):
        self.value = value
        self.left = None
        self.right = None
```

```
def create_bt_count_oracles_extract(values: : List[int], k: int):
  # it should return either True or false based on Ron's Answer
  return result
```

Question 6

In a distant land, a legendary Binary Tree of Prosperity was said to hold the key to great riches. The tree had multiple levels, each with a hidden treasure chest. However, not all chests were equal in value. Enter Finn, a fearless treasure hunter, determined to uncover the most valuable chest. But there was a twist – the chest with the highest treasure was guarded by a magical puzzle. Finn had to identify the level of the tree where the multiplication of the chest values was the greatest. Help Finn to crack this puzzle and unearth the chest filled with untold riches. Note - the level of the root is 1, the level of its children is 2, and so on…

Examples

Example 1:

`Input:`

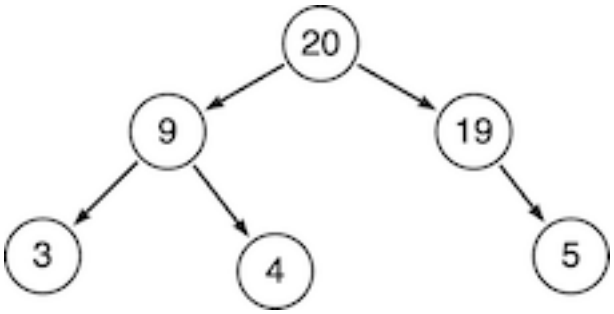

Figure 1: tree 2_1_1

```
Output: 2
Explanation:
Level 1: 20
Level 2: 9 * 19 = 171
Level 3: 3 * 4 * 5 = 60
```

Example 2:

`Input:`

```
Output: 4
Explanation:
Level 1: 20
Level 2: 9 * -100 = -900
Level 3: 2 * 14 * 4 = 112
Level 4: 51 * 21 = 1071
```

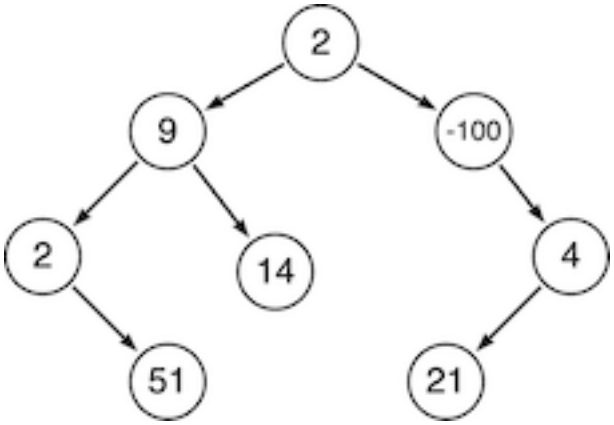

Figure 2: tree 2_1_2

Constraints

- No of nodes > 0

Function

```
def solve_puzzle(root):
  #your code goes here

  return result
```

Question 7

In a distant forest, a legend spoke of the "Tree of Numbers." This ancient tree possessed a mystical power: Each node contains a number between 0 to 9 digits. Every root-to-leaf path represented a distinct number. For example a root to leaf path contains 4->7->1->5. The distinct number will be 4715.

One fateful day, a brave adventurer named Alex ventured deep into the forest and encountered this legendary tree. A friendly forest spirit emerged, revealing the tree's secret: if Alex could calculate the sum of all the root-to-leaf numbers, a single wish would be granted.

Alex eagerly accepted the challenge, armed with the knowledge that the sum would fit within the bounds of a 32-bit integer. With determination in their heart, Alex embarked on a journey to unravel the magical properties of the Tree of Numbers.

Will you join Alex on this quest and help them decipher the mystical sum hidden within the tree's branches and leaves?

Examples

Example 1:

```
Input: Root node of the tree
```

```
Output: 1026
Explanation: In the complete binary tree
```

Example 2:

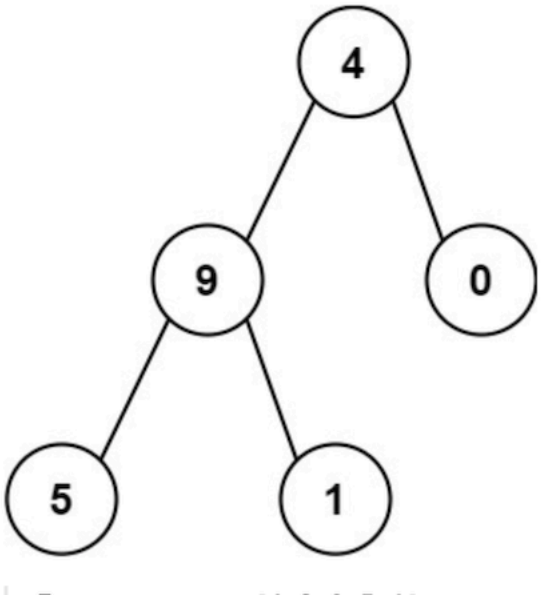

Figure 3: tree 3_2_2

```
The root-to-leaf path 4->9->5 represents the number 495.
The root-to-leaf path 4->9->1 represents the number 491.
The root-to-leaf path 4->0 represents the number 40.
Therefore, sum = 495 + 491 + 40 = 1026.
```

Constraints:

- 0 <= Node.val <= 9
- The depth of the tree will not exceed 10.
- Do not add the TreeNode class to your py file while submitting

Function

```python
class TreeNode:
    def __init__(self, value: int):
        self.data = data
        self.left = None
        self.right = None

def TreeOfNumbers(root) -> int:
  # it should return an integer which is sum of all the numbers generated from tree(root
  return sum
```

Question 8

Using Python Dictionary, implement Amortized Dictionary as a Python class, where the dictionary keys represent the levels and values are the elements of the corresponding level. At each level i, there can be either 2^i elements or no elements at all. Any level that contains elements, has them

in a sorted order. Implement the below methods for this class: - `Insert` method inserts an element into the amortoized dictionary. - `Search` method searches for the element, and returns the level if the element exists in the amortized dictionary, else returns -1. - `Print` method returns a list consisting of list of elements at each level. - All the above methods should be implemented as per the time complexity discussed in class.

Examples

Example 1:

```
ad = amor_dict([23, 12 ,24, 42])
print(ad.print())
# [[], [], [12, 23, 24, 42]]
ad.insert(11)
print(ad.print())
# [[11], [], [12, 23, 24, 42]]
ad.insert(74)
print(ad.print())
# [[], [11, 74], [12, 23, 24, 42]]
print(ad.search(74))
# 1
print(ad.search(77))
# -1
```

Example 2:

```
ad = amor_dict([1, 5, 2, 7, 8, 4, 3])
print(ad.print())
# [[3], [4, 8], [1, 2, 5, 7]]
print(ad.search(1))
# 2
ad.insert(11)
print(ad.print())
# [[], [], [], [1, 2, 3, 4, 5, 7, 8, 11]]
print(ad.search(1))
# 3
```

Function

```
class amor_dict():
    def __init__(self, num_list = []):
        # your code here
        pass

    def insert(self, num):
        # your code here
        pass

    def search(self, num):
        # your code here
```

```
    pass

def print(self):
    # your code here
    pass
```