Assignment

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

Alice and Bob are discussing the implementation of an efficient lookup dictionary. Alice came up with an idea of usage of skip lists. Help Alice and Bob implement a lookup dictionary using skip list.

Following are the instructions for SkipList:

Constraints

- Class should have the methods insert and search.
- `insert` method inserts the number into the skiplist.
- `search` method checks the presence of a target in the skiplist and returns `True` if it is present and returns `False` otherwise.
- All the methods should be implemented in `O(logn)` time complexity.

Note: Class Node is already written for you and will be useful in creating your skip list.

Example:

```
sl = SkipList()
sl.insert(1) # None
sl.insert(2) # None
sl.insert(3) # None
print(sl.search(4)) # False
sl.insert(4) # None
print(sl.search(4)) # True
print(sl.search(1)) # True
```

Function

```
class Node:
  def __init__(self, val):
      self.val = val
      self.next = None
```

```
        self.down = None

class SkipList:
    def __init__(self):
        # Any variables for intialization

    def search(self, target: int) -> bool:
        # Complete this function
        # Returns True if the element is present in skip list else False
        return False

    def insert(self, num: int) -> None:
        # Complete this function
        # Inserts the element into the skip list
        return None
```

Question 2

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**: This method allows you to insert an element into the amortized dictionary. The insertion is optimized to maintain the required structure, ensuring that elements are appropriately placed within the levels.
- **Search Method**: You can use this method to search for a specific element in the amortized dictionary. If the element is found, the method returns the corresponding level; otherwise, it returns -1, indicating that the element is not present in the dictionary.
- **Print Method**: This method returns a list consisting of lists of elements at each level. The elements are organized based on the amortized dictionary structure, and the resulting list provides a clear representation of the elements within each level.

All implemented methods are designed to operate within a time complexity that aligns with the principles discussed in class, ensuring efficiency and optimal performance. The class serves as a practical implementation of an amortized dictionary in Python, adhering to specified structural constraints and time complexity considerations.

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
```

Constraints

- Implement the functions according to the time complexity discussed in the class.

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):
    # Sample print function
    result = list()
    for level in self.levels:  # iterate over all the levels
        result.append(level[:])  # make a copy of each level to result
    return result
```

Question 3

Implement the Deque (Double Ended Queue) class as per the given structure.
A Deque, short for double-ended queue, allows for the insertion and removal of elements from either the front or the rear of the queue.

Examples

```
deque = Deque(3)
deque.pushRear(1) # True
deque.pushRear(2) # True
deque.pushRear(3) # True
deque.pushRear(4) # False (deque is full)
deque.pushFront(4) # False (deque is full)
deque.popFront() # 1
deque.popRear() # 3
deque.pushFront(4) # True
deque.pushFront(5) # True
deque.popRear() # 2
deque.popRear() # 4
deque.popRear() # 5
deque.popRear() # -1 (queue is empty)
deque.popFront() # -1 (queue is empty)
```

Constraints

- Time complexity of `pushFront()`, `pushRear()`, `popFront()`, and `popRear()` should be `O(1)`.
- Space complexity should be `O(max_size)`.

Function

```python
class Deque:

    # Initializes the object with the size of the deque to be 'max_size'.
    def __init__(self, max_size: int):
        pass

    # Inserts an element at the front of deque;
    # Return True if the operation is successful, else False.
    def pushFront(self, value: int) -> bool:
        pass

    # Inserts an element at the rear of deque;
    # Return True if the operation is successful, else False.
    def pushRear(self, value: int) -> bool:
        pass

    # Return the front value from the deque;
    # if the deque is empty, return -1.
    def popFront(self) -> int:
        pass

    # Return the rear value from the deque;
    # if the deque is empty, return -1.
    def popRear(self) -> int:
```

```
    pass
```

Question 4

In a big fight between Iron Man and Thanos, they used special symbols – 'A' for Iron Man's hits and 'T' for Thanos's strikes. Someone curious asked, "How long is the best part of their fight where Iron Man's 'A' hits matched perfectly with Thanos's 'T'?"

It was like finding the longest continuous time when Iron Man's moves fit exactly with Thanos's strikes. The challenge was clear: "Find out how long the coolest part is, where every 'A' from Iron Man lines up perfectly with a 'T' from Thanos."

And so, the quest started to figure out the sweet spot in their battle – the part where 'A' and 'T' matched perfectly. Audience eagerly waited to see this special balance in their cosmic clash.

Examples

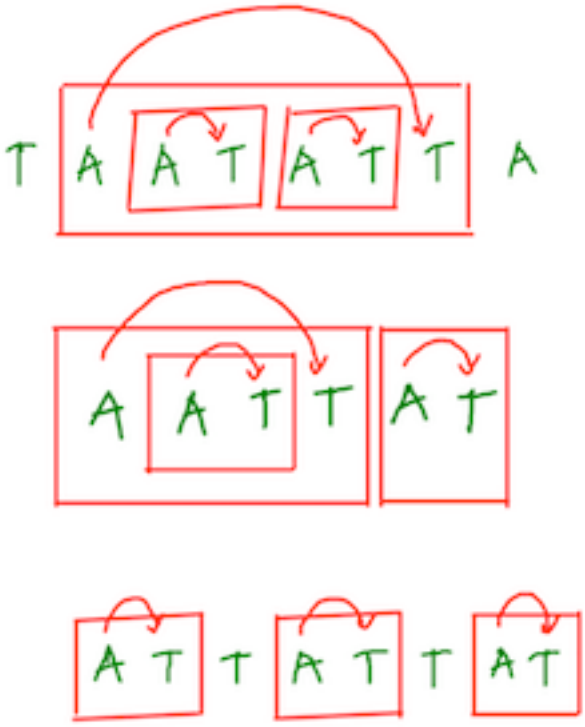

Figure 1: Iron man VS Thanos

Example 1:

```
Input: TAATATTA
Output: 6
```

Example 2:

```
Input: AATTAT
Output: 6
```

Example 3:

```
Input: ATTATTAT
Output: 2
```

Explanation: Longest continuous time where the battle was balanced is 2.

Constraints

- Solve using stack data structure
- Solve it in $O(N)$ Time.
- String contains only 'A' and 'T'.

Function

```
def ironman_vs_thanos_again(s):
  return 0
```

Question 5

In Bitland, a critical predicament has arisen due to an unintentional redirection of the left-side road onto the right during construction, leaving cars with no option to turn back. The primary objective is to rescue people impacted by this unforeseen road realignment. Your task is to devise the saveMaximumPeople(cars) function, which returns the number of people that can be saved considering each car's direction (positive for east, negative for west) and the number of passengers on board. Notably, a crucial rule dictates that only the car with the maximum number of passengers survives in the event of a collision. Furthermore, if both colliding cars carry an equal number of passengers, all occupants perish. Cars reaching either end of the road, whether east or west, are eligible for rescue. The government of Bitland urgently seeks an efficient solution to ascertain the number of citizens saved post-collisions, underscoring the importance of your algorithmic approach in this critical situation.

Examples

Example 1:

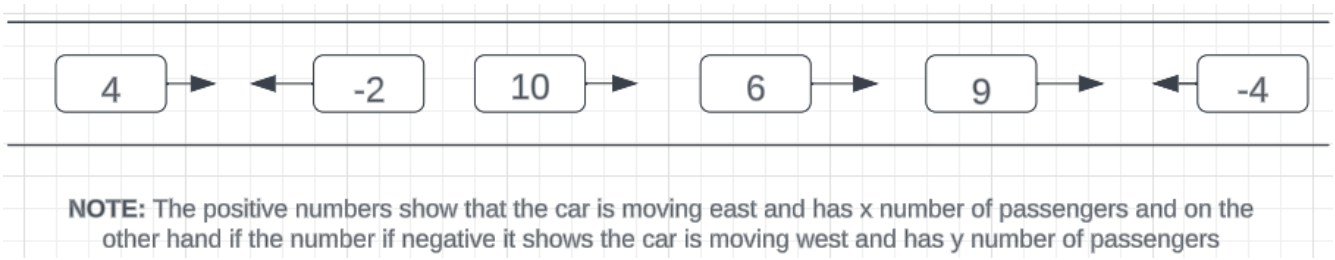

Figure 2: Bitland

```
Input:[4,-2,10,6,9,-4]
Output: 29
```

Explanation: For reference refer the image.As part of an optimal strategy, the car at index 1 will collide with the car at index 0. However, to maximize the number of people surviving, the decision

is made to sacrifice the occupants of the car at index 1. Additionally, the cars at index 2 and 3 will not collide, and the car at index 4 will collide with the car at index 5. Again, the choice is made to prioritize preserving lives, resulting in the decision to sacrifice the occupants of the car at index 5. This approach aims to ensure the highest possible number of survivors by strategically making choices during collisions.

Example 2:

```
Input:[1,1,1,1,1,1,1,1,1]
Output: 9
```

Explanation: Here the collisions will not happen and hence we can save everybody who's travelling.

Constraints

- Solve the question using stack
- Expected time complexity is $O(n)$

Function

```
def saveMaximumPeople(cars:List[int])->int:
  return maximumsavedpeople
```

Question 6

Step into the dynamic world of Bob, a stockbroker navigating the realms of high-frequency trading with an intriguing superpower—he possesses foresight into the future closing prices of HalfApple stocks. In the fast-paced landscape of stock markets, Bob grapples with a unique set of constraints: he's limited to purchasing just one stock per day, forbidden from selling at a loss, and compelled to sell as soon as the price surpasses the buying price for the first time.

Your mission is to craft a function named impatientBob(Prices) to aid Bob in optimizing his stock transactions. The "Prices" list encapsulates the daily closing prices that shape Bob's strategic decisions. The function should offer insights into the waiting period for each stock Bob procures on a given day. This waiting period is crucial, representing the duration until the stock's price surpasses the buying price for the first time, triggering a sell.

Can your function empower Bob to make judicious decisions, ensuring he maximizes profits while adhering to the intricate constraints of high-frequency trading? Unleash the power of code to assist Bob in conquering the stock market, where foresight meets strategic precision.

Examples

Example 1:

```
Input: Prices = [1,2,3,4]
Output: [1,1,1,0]
```

Explanation: The price of the stock on day 1 was 1 and it got above 1 on the very next day hence, bob only has to wait 1 day to sell the stock he bought on the first day. Similarly for all of the other days till 3rd. For the 4th one he cannot sell it.

Example 2:

```
Input: Prices = [10,8,7,6,11]
Output: [4,3,2,1,0]
```

Explanation: For the first day the stock price crosses 10 only on day 5 hence we waited 4 days for that to sell it. Similarly we sell all the stocks only on day 5.

Constraints

- Solve the question using stack
- Expected time complexity is $O(n)$

Function

```
def impatientBob(Prices:List[int])->List[int]:
  return patience
```

Question 7

Once upon a time in Parenthesis Kingdom, there was a magical string called 's' that loved to balance itself, meaning every opening parenthesis had a matching closing one. It had a unique scoring system:

- "()" had a score of 1.
- If you had two balanced strings, say A and B, their score when combined was A + B.
- And if a string was enclosed in parentheses, like (A), its score was 2 times A.

The townsfolk were intrigued by this magical scoring, and they asked: Can you find the score of a given balanced parentheses string 's'? Delve into the magic and uncover the score!

Examples

Example 1:

```
Input: s = "(())"
Output: 2
```

Example 2:

```
Input: s = "()()"
Output: 2
```

Example 3:

```
Input: s = "(()())"
Output: 4
```

Constraints

- 's' consists of only '(' and ')'
- Expected time complexity is $O(n)$

Function

```
def scoreOfMagicalString(s: str) -> int:
    pass
```