**ADVANCE CODING – 4**

**K.REVANTH**

**VU21CSEN0300033**

1. **Maximum Sum Circular Subarray**

**Code:**

class Solution:

    def maxSubarraySumCircular(self, nums: list[int]) -> int:

        def kadane(arr):

            max\_sum = current\_sum = arr[0]

            for num in arr[1:]:

                current\_sum = max(num, current\_sum + num)

                max\_sum = max(max\_sum, current\_sum)

            return max\_sum

        total\_sum = sum(nums)

        max\_sum = kadane(nums)

        # Invert nums for finding the minimum subarray sum

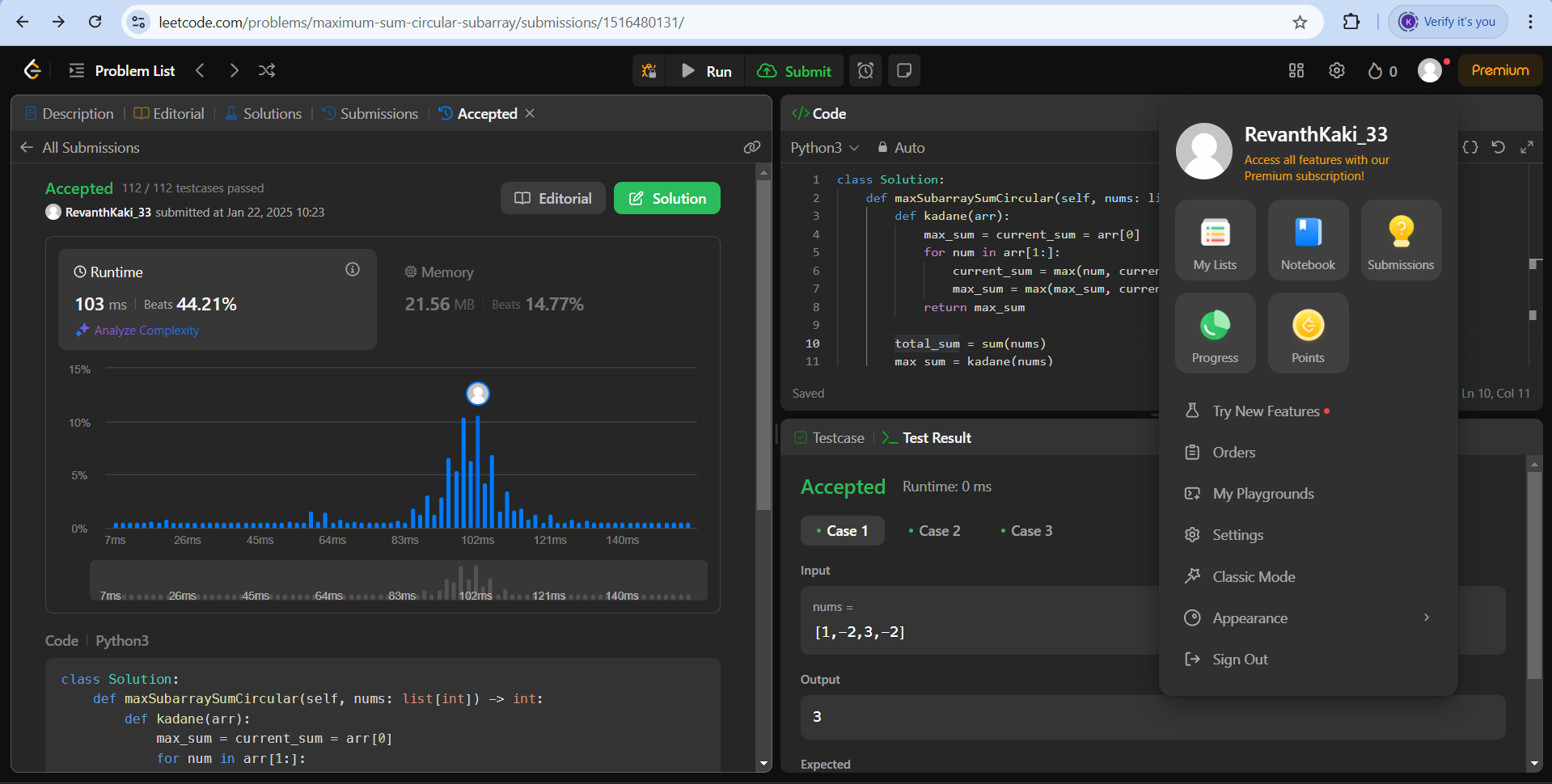
        min\_sum = kadane([-num for num in nums])

        if max\_sum < 0:

            return max\_sum

        return max(max\_sum, total\_sum + min\_sum)

**Output:**



1. **Stamping The Sequence**

**Code:**

class Solution:

    def movesToStamp(self, stamp: str, target: str) -> list[int]:

        m, n = len(stamp), len(target)

        tl = list(target)

        s = ['?'] \* n

        res = []

        made\_change = True

        def can\_stamp(start):

            can\_place = False

            for i in range(m):

                if tl[start + i] != '?' and tl[start + i] != stamp[i]:

                    return False

                if tl[start + i] != '?':

                    can\_place = True

            return can\_place

        def do\_stamp(start):

            for i in range(m):

                tl[start + i] = '?'

        while made\_change:

            made\_change = False

            for i in range(n - m + 1):

                if can\_stamp(i):

                    do\_stamp(i)

                    res.append(i)

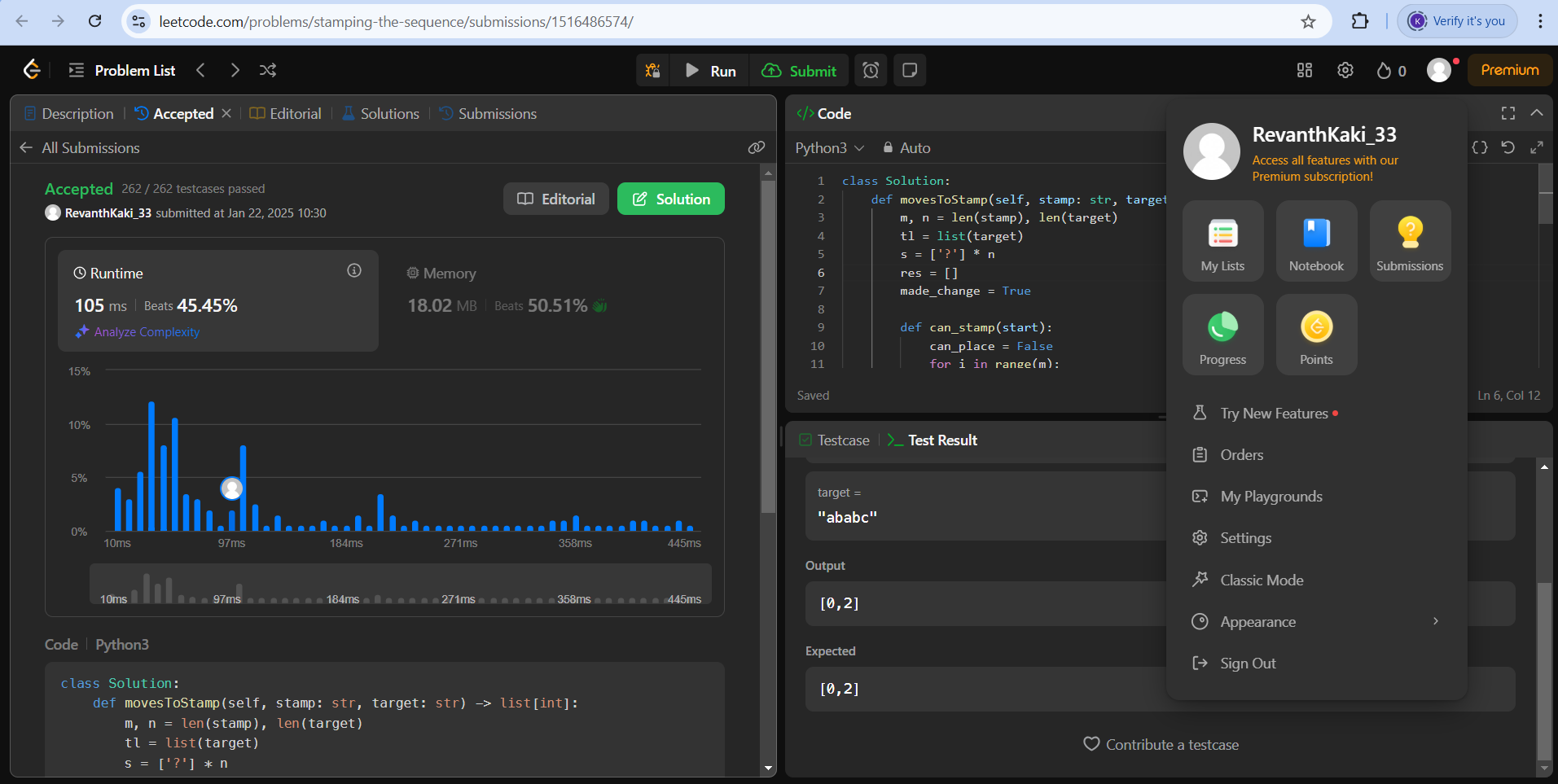
                    made\_change = True

        if any(c != '?' for c in tl):

            return []

        return res[::-1]

**Output:**

****

1. **Maximum Sum Circular Subarray**

**Code:**

class BrowserHistory:

def \_\_init\_\_(self, homepage: str):

self.hist = [homepage]

self.cur = 0

def visit(self, url: str):

self.hist = self.hist[:self.cur + 1]

self.hist.append(url)

self.cur += 1

def back(self, steps: int) -> str:

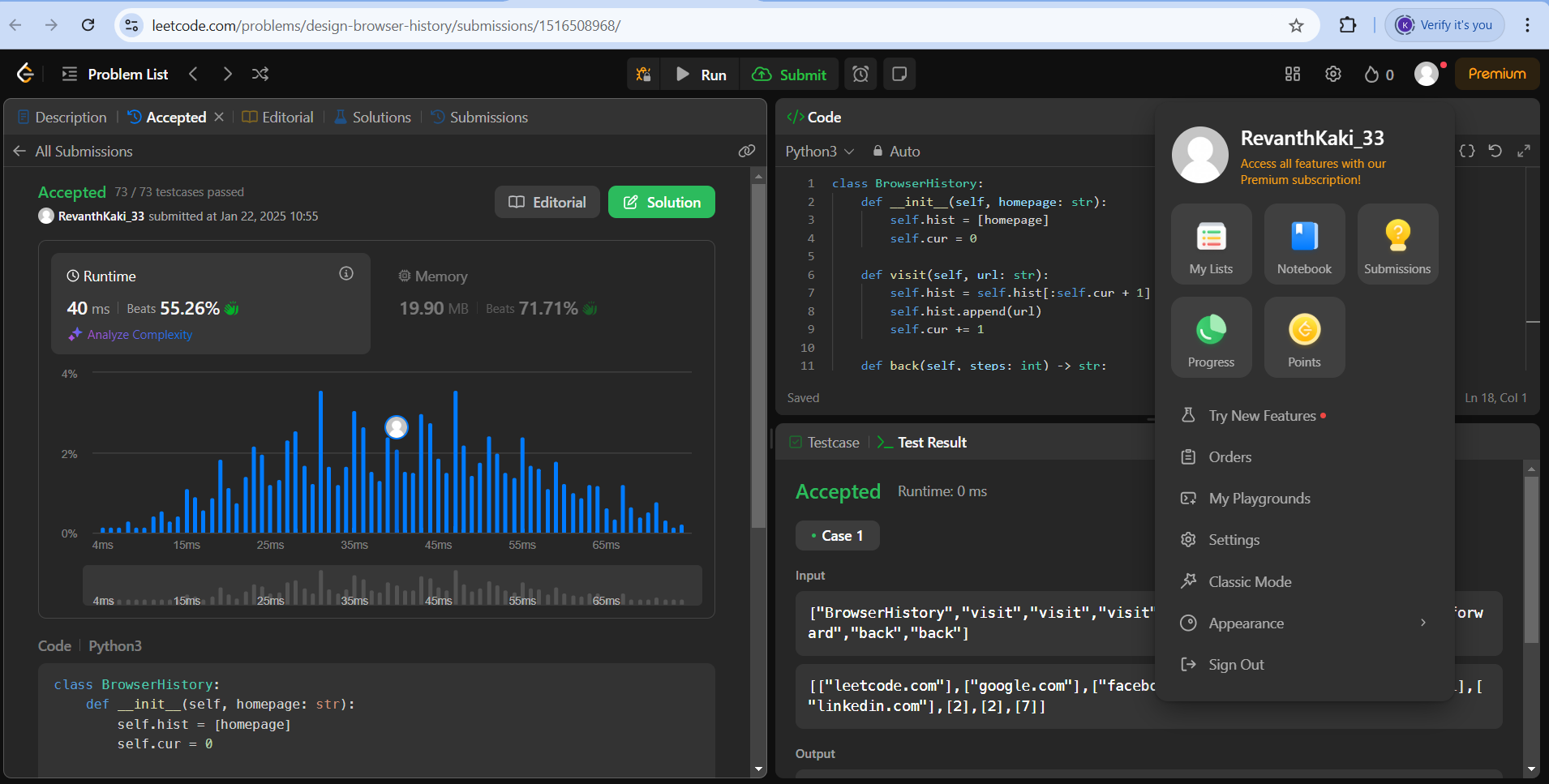
self.cur = max(0, self.cur - steps)

return self.hist[self.cur]

def forward(self, steps: int) -> str:

self.cur = min(len(self.hist) - 1, self.cur + steps)

return self.hist[self.cur]  
  
  
**Output:**



1. **LRU Cache**

**Code:**

from collections import OrderedDict

class LRUCache:

def \_\_init\_\_(self, capacity: int):

self.cache = OrderedDict()

self.cap = capacity

def get(self, key: int) -> int:

if key in self.cache:

self.cache.move\_to\_end(key)

return self.cache[key]

return -1

def put(self, key: int, value: int) -> None:

if key in self.cache:

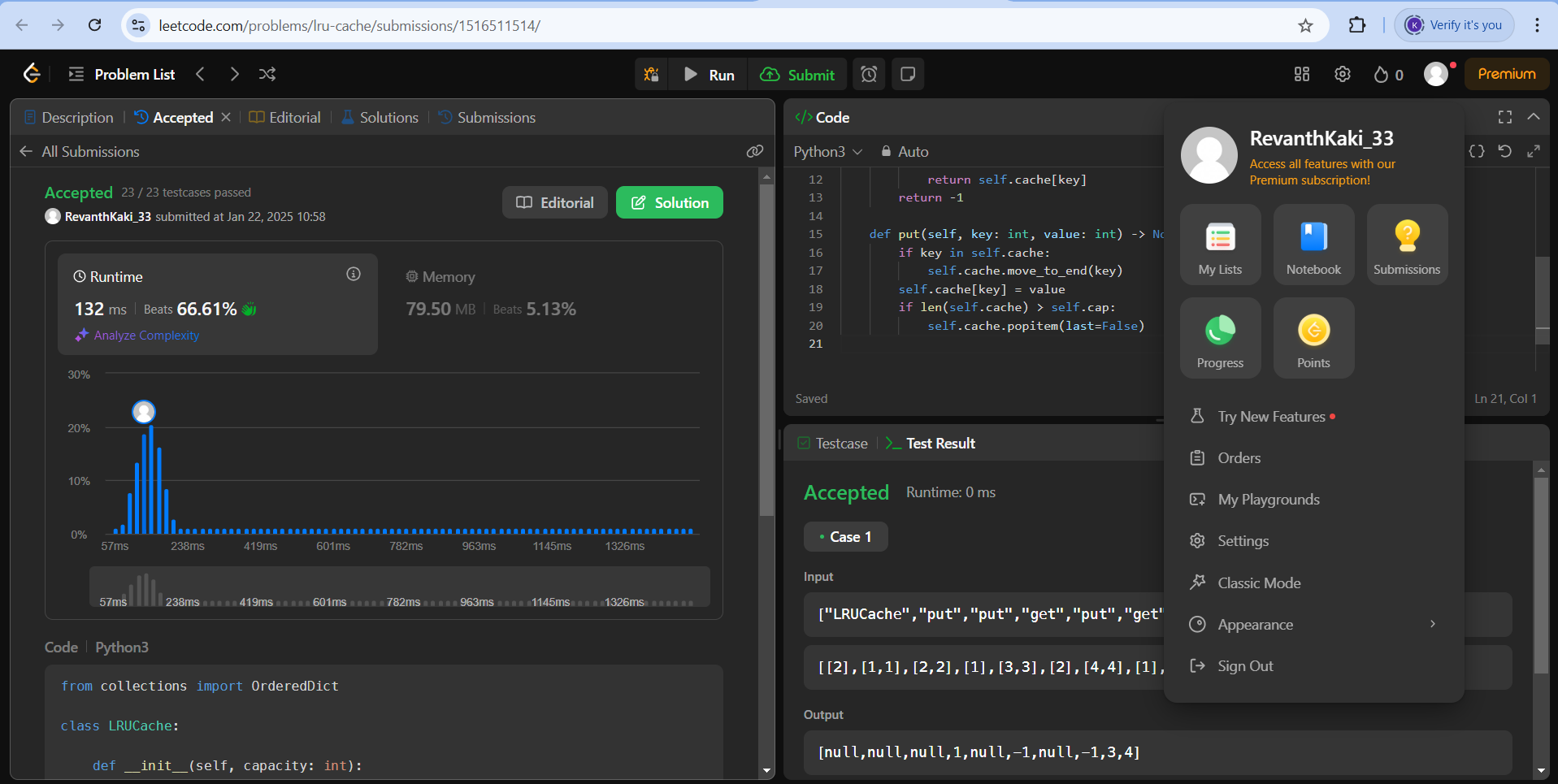
self.cache.move\_to\_end(key)

self.cache[key] = value

if len(self.cache) > self.cap:

self.cache.popitem(last=False)

**Output:**

****