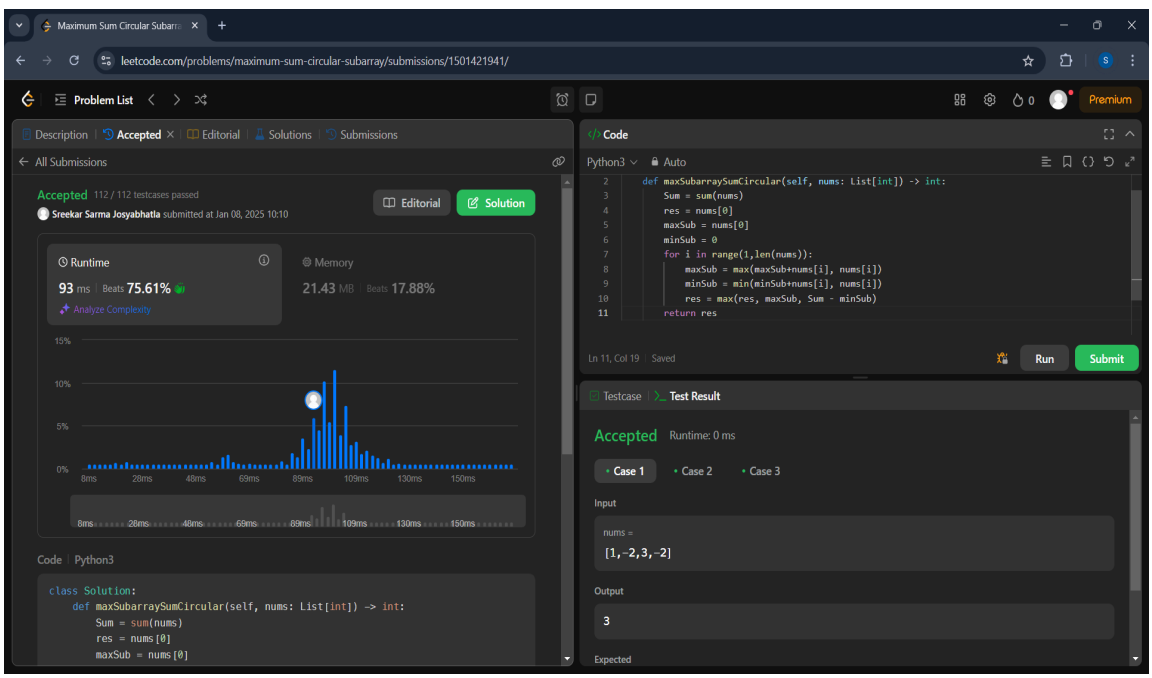
Assignment-2

J.Sreekar Sarma
VU21CSEN0300040

1. Maximum Sum Circular Subarray

```
class Solution:
      def maxSubarraySumCircular(self, nums: List[int]) ->
int:
        Sum = sum(nums)
        res = nums[0]
        maxSub = nums[0]
        minSub = 0
        for i in range(1,len(nums)):
            maxSub = max(maxSub+nums[i], nums[i])
            minSub = min(minSub+nums[i], nums[i])
            res = max(res, maxSub, Sum - minSub)
        return res
```

OUTPUT



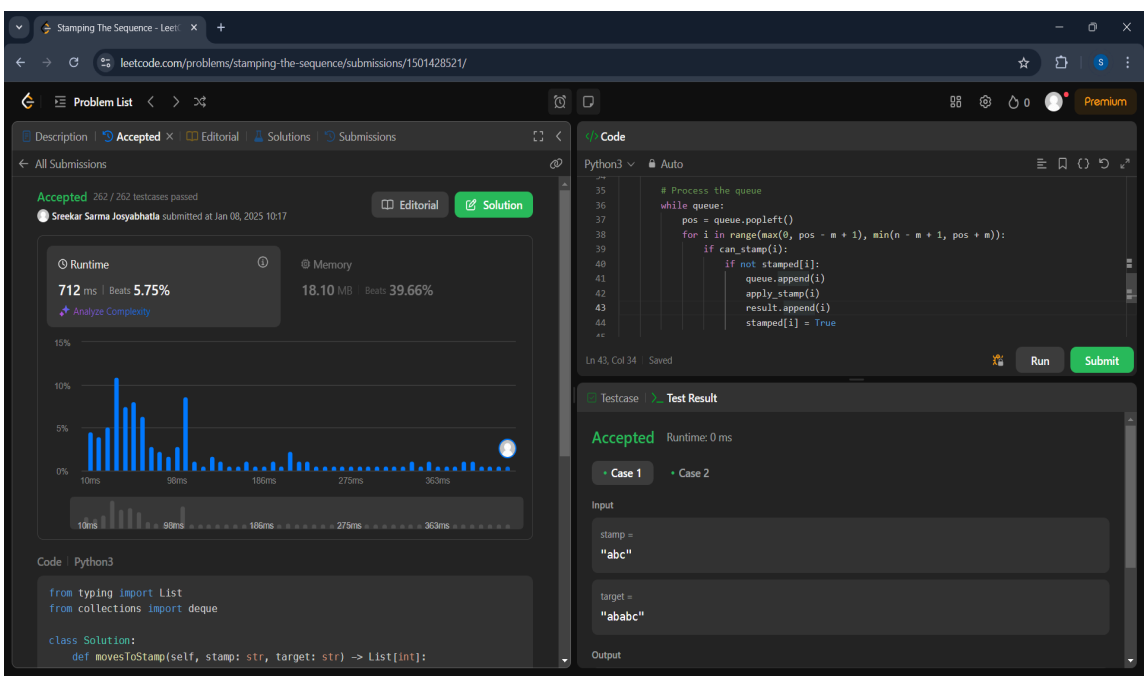
2. Stamping The Sequence

```
from typing import List
from collections import deque
```

```
class Solution:
      def movesToStamp(self, stamp: str, target: str) ->
List[int]:
        m, n = len(stamp), len(target)
        stamp = list(stamp)
        target = list(target)

           # Helper function to check if we can stamp at
position i
        def can_stamp(i):
            for j in range(m):
                  if target[i + j] != '?' and target[i + j]
!= stamp[j]:
                    return False
            return True

        # Helper function to apply stamp at position i
        def apply_stamp(i):
            for j in range(m):
                target[i + j] = '?'

        # Track visited positions and the stamping process
        stamped = [False] * n
        result = []
        queue = deque()

        # Initial queue fill
        for i in range(n - m + 1):
            if can_stamp(i):
                queue.append(i)
                apply_stamp(i)
                result.append(i)
                stamped[i] = True

        # Process the queue
```

```python
        while queue:
            pos = queue.popleft()
             for i in range(max(0, pos - m + 1), min(n - m +
1, pos + m)):
                if can_stamp(i):
                    if not stamped[i]:
                        queue.append(i)
                        apply_stamp(i)
                        result.append(i)
                        stamped[i] = True

        # Verify that all characters in target are stamped
        if all(c == '?' for c in target):
            return result[::-1]
        else:
            return []
```

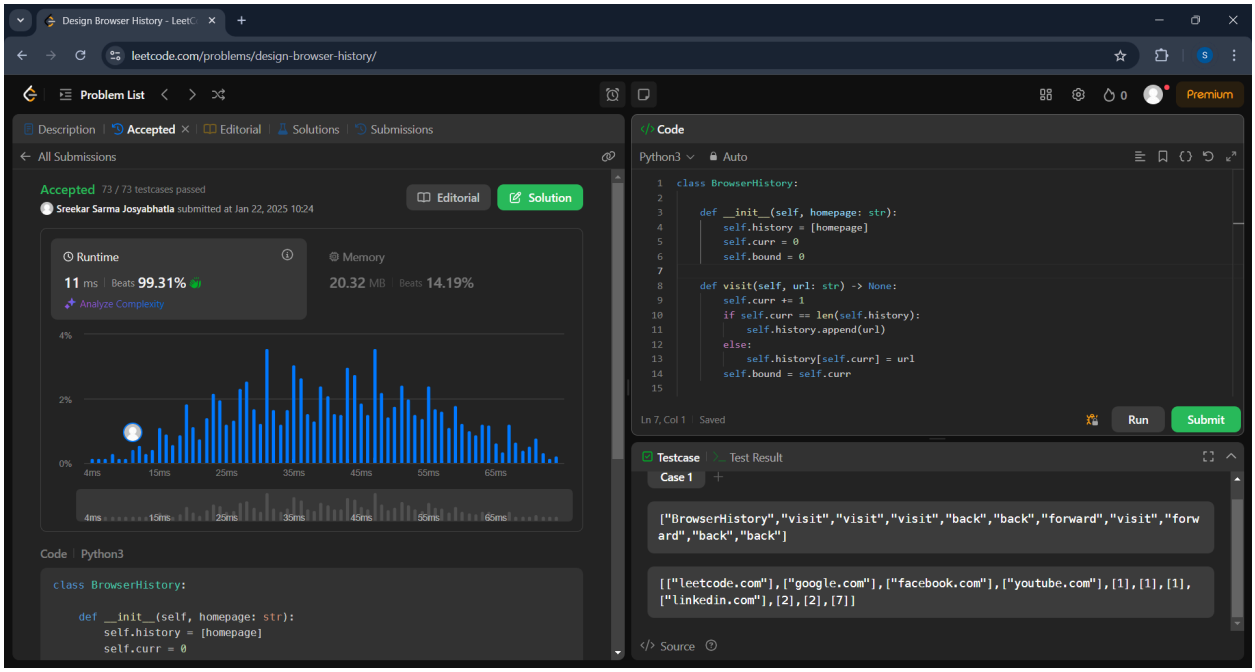
OUTPUT



3. Design Browser History

```python
class BrowserHistory:

    def __init__(self, homepage: str):
        self.history = [homepage]
        self.curr = 0
        self.bound = 0

    def visit(self, url: str) -> None:
        self.curr += 1
        if self.curr == len(self.history):
            self.history.append(url)
        else:
            self.history[self.curr] = url
        self.bound = self.curr

    def back(self, steps: int) -> str:
        self.curr = max(self.curr - steps, 0)
        return self.history[self.curr]

    def forward(self, steps: int) -> str:
        self.curr = min(self.curr + steps, self.bound)
        return self.history[self.curr]
```

<u>OUTPUT:</u>



4.LRU Cache

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

    def __init__(self, capacity: int):
        self.cap = capacity
        self.head = self.Node(-1, -1)
        self.tail = self.Node(-1, -1)
        self.head.next = self.tail
        self.tail.prev = self.head
        self.m = {}

    def addNode(self, newnode):
        temp = self.head.next
        newnode.next = temp
        newnode.prev = self.head
```

```
        self.head.next = newnode
        temp.prev = newnode

    def deleteNode(self, delnode):
        prevv = delnode.prev
        nextt = delnode.next
        prevv.next = nextt
        nextt.prev = prevv

    def get(self, key: int) -> int:
        if key in self.m:
            resNode = self.m[key]
            ans = resNode.val
            del self.m[key]
            self.deleteNode(resNode)
            self.addNode(resNode)
            self.m[key] = self.head.next
            return ans
        return -1

    def put(self, key: int, value: int) -> None:
        if key in self.m:
            curr = self.m[key]
            del self.m[key]
            self.deleteNode(curr)

        if len(self.m) == self.cap:
            del self.m[self.tail.prev.key]
            self.deleteNode(self.tail.prev)

        self.addNode(self.Node(key, value))
        self.m[key] = self.head.next
```

OUTPUT:

LRU Cache - LeetCode

leetcode.com/problems/lru-cache/

Problem List

Description | Accepted | Editorial | Solutions | Submissions

All Submissions

Accepted 23 / 23 testcases passed

Sreekar Sarma Josyabhatla submitted at Jan 22, 2025 10:36

Editorial | Solution

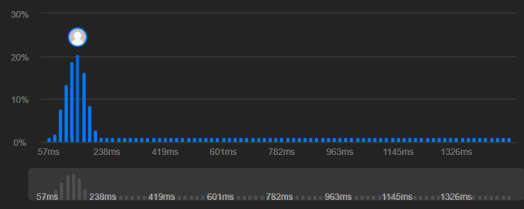
Runtime

149 ms | Beats 47.30%

Analyze Complexity

Memory

78.57 MB | Beats 32.70%



Code | Python3

```python
class LRUCache:
    class Node:
        def __init__(self, key, val):
            self.key = key
            self.val = val
```

Code

Python3 | Auto

```python
class LRUCache:
    class Node:
        def __init__(self, key, val):
            self.key = key
            self.val = val
            self.prev = None
            self.next = None

    def __init__(self, capacity: int):
        self.cap = capacity
        self.head = self.Node(-1, -1)
        self.tail = self.Node(-1, -1)
        self.head.next = self.tail
        self.tail.prev = self.head
        self.m = {}
```

Ln 14, Col 35 | Saved

Run | Submit

Testcase | Test Result

Input

```
["LRUCache","put","put","get","put","get","put","get","get","get"]
```

```
[[2],[1,1],[2,2],[1],[3,3],[2],[4,4],[1],[3],[4]]
```

Output

```
[null,null,null,1,null,-1,null,-1,3,4]
```