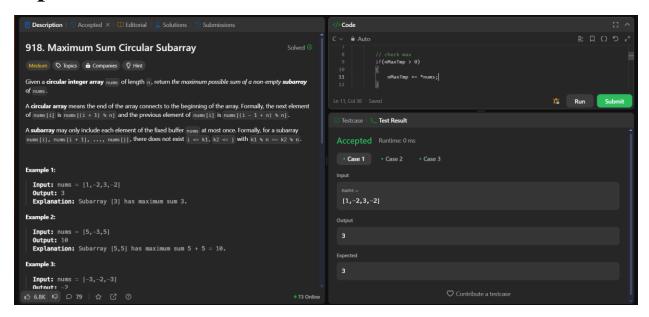# ADVANCED CODING

Assignment -2

OMSRI DABBEERU

VU21CSEN0300129

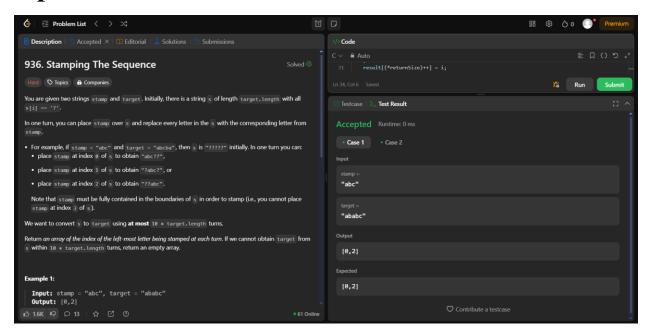Cse aiml

918. Maximum Sum Circular Subarray :

```c
int maxSubarraySumCircular(int* nums, int numsSize){
    short sNumsSize = numsSize;
    int nTotal = 0, nMax = INT_MIN, nMaxTmp = 0, nMin = INT_MAX, nMinTmp = 0;
    while(numsSize-- > 0)
    {
        nTotal += *nums;

        // check max
        if(nMaxTmp > 0)
        {
            nMaxTmp += *nums;
        }
        else
        {
            nMaxTmp = *nums;
        }
        if(nMaxTmp > nMax) nMax = nMaxTmp;

        // check min
        if(nMinTmp < 0)
        {
            nMinTmp += *nums;
        }
        else
        {
            nMinTmp = *nums;
        }
        if(nMinTmp < nMin) nMin = nMinTmp;



        nums++;
    }

    // get result from nTotal - nMin
    nTotal -= nMin;

    return nMax < 0? nMax: (nTotal > nMax? nTotal: nMax);
}
```

Op:

918. Maximum Sum Circular Subarray

Solved ✓

`Medium` 🏷 Topics 🔒 Companies 💡 Hint

Given a **circular integer array** `nums` of length `n`, return *the maximum possible sum of a non-empty* **subarray** *of* `nums`.

A **circular array** means the end of the array connects to the beginning of the array. Formally, the next element of `nums[i]` is `nums[(i + 1) % n]` and the previous element of `nums[i]` is `nums[(i - 1 + n) % n]`.

A **subarray** may only include each element of the fixed buffer `nums` at most once. Formally, for a subarray `nums[i]`, `nums[i + 1]`, ..., `nums[j]`, there does not exist `i <= k1`, `k2 <= j` with `k1 % n == k2 % n`.

Example 1:

```
Input: nums = [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3.
```

Example 2:

```
Input: nums = [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10.
```

Example 3:

```
Input: nums = [-3,-2,-3]
Output: -2
```

👍 6.8K 👎 💬 79 ☆ ⧉ ⦾   ● 73 Online

Code

C ⌄   🔒 Auto

```
 7
 8        // check max
 9        if(nMaxTmp > 0)
10        {
11            nMaxTmp += *nums;
12        }
```

Ln 11, Col 30   Saved   Run   Submit

Testcase | **Test Result**

Accepted Runtime: 0 ms

• **Case 1**   • Case 2   • Case 3

Input

```
nums =
[1,-2,3,-2]
```

Output

```
3
```

Expected

```
3
```

♡ Contribute a testcase
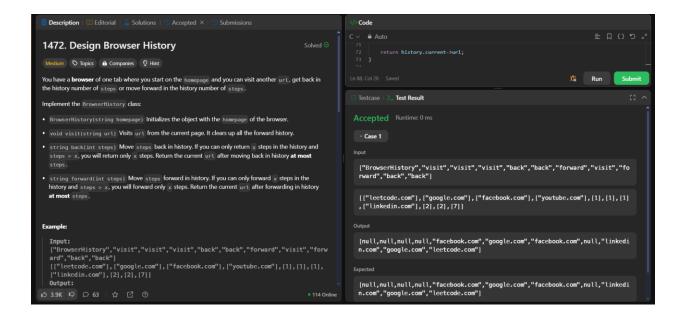
936. Stamping The Sequence :

```c
int isSubstr(char* s, char* a, int index){
  int lengthS = strlen(s), status = 0;
  for(int i = 0; i < lengthS; i++){
    if(s[i] != a[i+index] && a[i+index] != '*')
      return 0;
    if(s[i] == a[i+index])
      status = 1;
  }
  return status == 0 ? 0 : 1;
}

int notStar(char* target){
  int lengthTarget = strlen(target);
  for(int i = 0; i < lengthTarget; i++)
    if(target[i] != '*')
      return 1;
  return 0;
}

int* movesToStamp(char* stamp, char* target, int* returnSize){
  int* result = (int*)malloc(sizeof(int) * 10 * strlen(target));
  *returnSize = 0;
  int lengthT = strlen(target), lengthS = strlen(stamp);
  while(notStar(target)){
    int status = 0;
    for(int i = 0; i <= (lengthT - lengthS); i++){
      if(isSubstr(stamp, target, i)){
      for(int j = 0; j < lengthS; j++){
        target[j+i] = '*';
      }
      result[(*returnSize)++] = i;
      status = 1;
      }
    }
    if(status == 0){
      *returnSize = 0;
      break;
    }
  }
  for(int i = 0, j = *returnSize - 1; i < j; i++, j--){
    int temp = result[i];
    result[i] = result[j];
    result[j] = temp;
  }
  return result;}
```

Op:

📄 Description | ↻ Accepted ✕ | 📖 Editorial | 👤 Solutions | ↻ Submissions

936. Stamping The Sequence   Solved ⊘

`Hard` 🏷 Topics 🔒 Companies

You are given two strings `stamp` and `target`. Initially, there is a string `s` of length `target.length` with all `s[i] == '?'`.

In one turn, you can place `stamp` over `s` and replace every letter in the `s` with the corresponding letter from `stamp`.

- For example, if `stamp = "abc"` and `target = "abcba"`, then `s` is `"?????"` initially. In one turn you can:
  - place `stamp` at index `0` of `s` to obtain `"abc??"`,
  - place `stamp` at index `1` of `s` to obtain `"?abc?"`, or
  - place `stamp` at index `2` of `s` to obtain `"??abc"`.

    Note that `stamp` must be fully contained in the boundaries of `s` in order to stamp (i.e., you cannot place `stamp` at index `3` of `s`).

We want to convert `s` to `target` using **at most** `10 * target.length` turns.

Return *an array of the index of the left-most letter being stamped at each turn*. If we cannot obtain `target` from `s` within `10 * target.length` turns, return an empty array.

Example 1:

> **Input:** stamp = "abc", target = "ababc"
> **Output:** [0,2]

👍 1.6K 👎 | 💬 13 | ☆ | ⧉ | ⑦   ● 61 Online

</> Code

C ⌄   🔒 Auto   ≡ ⌖ {} ↻ ⤢

```
31        result[(*returnSize)++] = i;
```

Ln 34, Col 6 | Saved   🐛 | Run | Submit

☑ Testcase | >_ **Test Result**   ⛶ ⌃

Accepted Runtime: 0 ms

`• Case 1`   • Case 2

Input

stamp =
"abc"

target =
"ababc"

Output

[0,2]

Expected

[0,2]

♡ Contribute a testcase

1472. Design Browser History :

```c
typedef struct tab
{
    char* url;
    struct tab* next;
    struct tab* previous;
} tab;

typedef struct BrowserHistory
{
    tab* current;
} BrowserHistory;

tab* createTab(char* url)
{
    tab* newTab = malloc(sizeof(tab));

    newTab->url = url;
    newTab->next = NULL;
    newTab->previous = NULL;

    return newTab;
}
BrowserHistory* browserHistoryCreate(char* homepage)
{
    BrowserHistory* history = malloc(sizeof(BrowserHistory));
    history->current = createTab(homepage);

    return history;
}

void browserHistoryVisit(BrowserHistory* obj, char* url)
{
    BrowserHistory history = *obj;

    tab* newTab = createTab(url);
    history.current->next = newTab;
    newTab->previous = history.current;
    history.current = newTab;

    *obj = history;
}

char* browserHistoryBack(BrowserHistory* obj, int steps)
{
    BrowserHistory history = *obj;
```

```
    int back = 0;

    while(history.current->previous != NULL && back < steps)
    {
        back++;
        history.current = history.current->previous;
    }

    *obj = history;

    return history.current->url;
}

char * browserHistoryForward(BrowserHistory* obj, int steps)
{
    BrowserHistory history = *obj;
    int forward = 0;

    while(history.current->next != NULL && forward < steps)
    {
        forward++;
        history.current = history.current->next;
    }

    *obj = history;

    return history.current->url;
}

void browserHistoryFree(BrowserHistory* obj)
{
    BrowserHistory history = *obj;

    while(history.current->previous != NULL) history.current = history.current->previous;

    while(history.current != NULL)
    {
        tab* current = history.current;
        history.current = history.current->next;
        free(current);
    }

    *obj = history;
    free(obj);
}
```

Op:

146. LRU Cache :

```c
struct node {
    int key;
    int val;
    struct node *next;
    struct node *prev;
};

typedef struct {
    int cap;
    int count;
} LRUCache;

struct node *head;
struct node *tail;

struct node* rem[10001];
LRUCache* lRUCacheCreate(int capacity) {

    LRUCache *cache = malloc(sizeof(LRUCache));
    cache->cap = capacity;
    cache->count = 0;
    head = malloc(sizeof(struct node));
    tail = malloc(sizeof(struct node));

    head->next = tail;
    tail->prev = head;

    for (int i = 0; i < 10001; i++)
        rem[i] = NULL;

    return cache;
}

void del(struct node *curr) {

    curr->prev->next = curr->next;
    curr->next->prev = curr->prev;
}

void add(struct node *curr) {

    curr->next = head->next;
    curr->prev = head;
```

```c
        head->next->prev = curr;
        head->next = curr;

}

int lRUCacheGet(LRUCache* obj, int key) {
        if(obj->count == 0)
            return -1;

        if(rem[key] == NULL)
            return -1;

        int val;
        struct node *curr = rem[key];
        val = curr->val;
        del(curr);
        add(curr);
        return val;
}

void lRUCachePut(LRUCache* obj, int key, int value) {

        printf("key=%d\n",key);
        if(rem[key] != NULL) {

        struct node *curr = rem[key];
        curr->val = value;
        del(curr);
        add(curr);
        }

        else {

            if(obj->count == obj->cap) {
                obj->count--;
                rem[tail->prev->key] = NULL;
                del(tail->prev);
            }

            obj->count++;
            struct node *curr = malloc(sizeof(struct node));
            curr->val = value;
            curr->key = key;
            printf("curr=%x\n",curr);
            printf("add key=%d val =%d \n",key,value);
            rem[key] = curr;
            add(curr);
        }
```

```
}

void lRUCacheFree(LRUCache* obj) {


    for(int i = 0; i <= 10000; i++)
        if(rem[i])
            free(rem[i]);

    free(head);
    free(tail);
    free(obj);


}

/**
 * Your LRUCache struct will be instantiated and called as such:
 * LRUCache* obj = lRUCacheCreate(capacity);
 * int param_1 = lRUCacheGet(obj, key);

 * lRUCachePut(obj, key, value);

 * lRUCacheFree(obj);
 */
```

Op: