Experiment 2(B)

Student Name: Tanmaya Kumar Pani **UID: 22BCS12986**
Branch: CSE **Section/Group: IOT-613-B**
Semester: 5 **Date of Performance: 23/07/2024**
Subject Name: Advanced Programming Lab-1 **Subject Code: 22CSP-314**

1. **Title:** Game of two stack

2. Objective:

Alexa has two stacks of non-negative integers, stack a[n] and b[m[]stack where index denotes the top of the stack. Alexa challenges Nick to play the following game:

In each move, Nick can remove one integer from the top of either stack or stack .
Nick keeps a running sum of the integers he removes from the two stacks.
Nick is disqualified from the game if, at any point, his running sum becomes greater than some integer given at the beginning of the game.
Nick's final score is the total number of integers he has removed from the two stacks.
Given a,b , and maxsum for games, find the maximum possible score Nick can achieve.

3. Algorithm

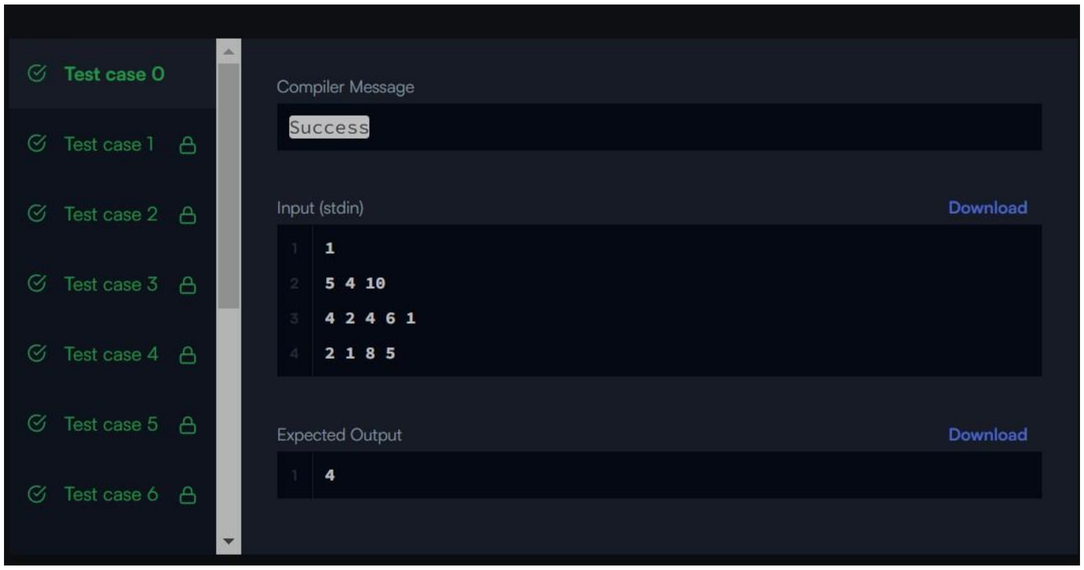
- Start with Stack A: Add elements from stack A until maxSum is reached.
- Track Count: Record the number of elements taken from stack A.
- Switch to Stack B: Add elements from stack B, adjusting by removing from stack A if maxSum is exceeded.
- Update Maximum: Track the maximum number of elements taken from both stacks within maxSum.

Output Result: Return the highest count achieved.

4. Implementation/Code:

```
#include <iostream>
#include <algorithm>
using namespace std;
const int MAX = 100000;
int twoStacks(int maxSum, int a[], int n, int b[], int m) {
    int ans = 0, sum = 0;
    int indexA = 0, indexB = 0;
    while (indexA < n && sum + a[indexA] <= maxSum) {
        sum += a[indexA];
        indexA++;
    }
    ans = indexA
    while (indexB < m && indexA >= 0) {
        sum += b[indexB];
        indexB++;
        while (sum > maxSum && indexA > 0) {
            indexA--;
            sum -= a[indexA];
        }
        if (sum <= maxSum) {
            ans = max(ans, indexA + indexB);
        }}
    return ans;
}
int main() {
    int  t,i;
    cin >> t;
    while (t--) {
        int n, m, maxSum;
        cin >> n >> m >> maxSum;
        int a[MAX], b[MAX];
        for (i=0;i<n;++i) cin >> a[i];
        for (i=0;i<m;++i) cin >> b[i];
        cout << twoStacks(maxSum, a, n, b, m) << endl;
    }
    return 0;
     }
```

5. Output:



```
Success
```

```
1
5 4 10
4 2 4 6 1
2 1 8 5
```

```
4
```

6. Learning Outcomes

- Efficiently navigate and compare two lists or stacks.
- To add and remove elements based on conditions to optimize results within a specified limit

Experiment 2(C)

Student Name: Tanmaya Kumar Pani **UID: 22BCS12986**
Branch: CSE **Section/Group: IOT-613-B**
Semester: 5 **Date of Performance: 23/07/2024**
Subject Name: Advanced Programming Lab-1 **Subject Code: 22CSP-314**

1. **Title:** Balanced Brackets

2. **Objective:**

A bracket is considered to be any one of the following characters: (,), {, }, [, or].

Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e.,),], or }) of the exact same type. There are three types of matched pairs of brackets: [], {}, and ().

A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. For example, {[(])} is not balanced because the contents in between { and } are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket,].

By this logic, we say a sequence of brackets is balanced if the following conditions are met:

It contains no unmatched brackets.

The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.

3. Algorithm:

- Initialize: Set up reference lists for brackets and an empty stack.
- Check Base Cases: Return "YES" for empty string, "NO" for odd length or incorrect first/last characters.
- Process Characters: Push indices for opening brackets to stack; for closing brackets, check and pop stack.
- Final Check: Return "YES" if stack is empty, else "NO".

4. Implementation/Code

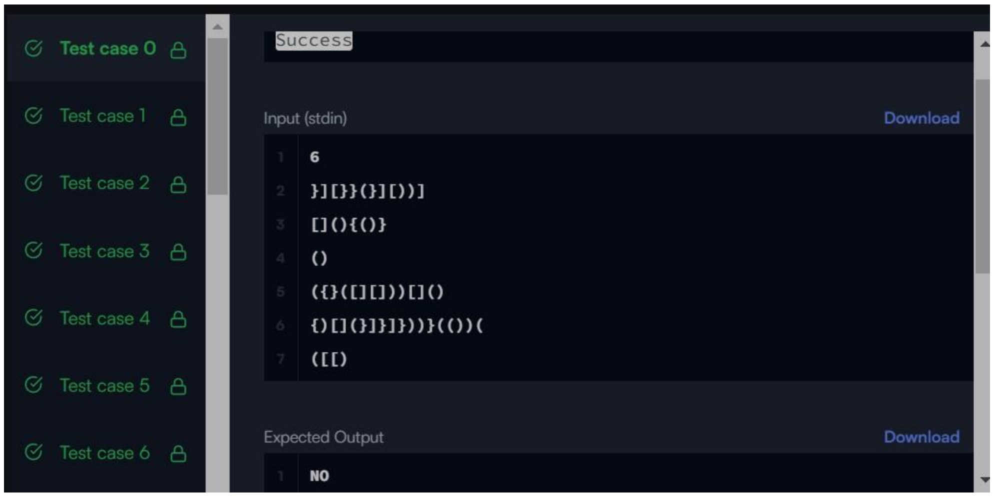
```
def isBalanced(s):
    stack = []

    for char in s:
        if char in '{[(':
            stack.append(char)
        else:
            if not stack:
                return "NO"
            top = stack.pop()
            if (char == '}' and top != '{') or (char == ']' and top != '[') or (char == ')' and top != '('):
                return "NO"

    return "YES" if not stack else "NO"

t = int(input())
for _ in range(t):
    s = input()
    print(isBalanced(s))
```

5. Output:



```
Success
```

Input (stdin)

```
6
}][}}(}][))]
[](){()}
()
({}([][]))[]()
{)[](}]}]}))}(())(
([[)
```

Expected Output

```
NO
```

6. Learning Outcomes:

- Manage and check matching pairs, like brackets in code.
- Checking for empty strings or mismatched brackets.

7. Time Complexity: O(n)

8. Space Complexity: O(n)