Experiment 9

Student Name: Ravideep Singh
UID: 22BCS10650
Branch: BE-CSE
Section/Group: NTPP_IOT_603_A
Semester: 6th
Date of Performance: 31-03-25
Subject Name: AP Lab-2
Subject Code: 22CSP-351

1. Aim: Set Matrix Zeroes:

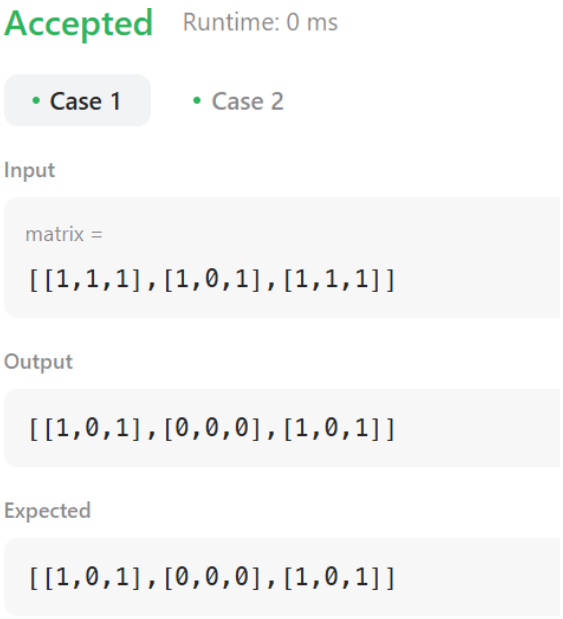
2. Objective:

Given an m x n matrix, if an element is 0, set its entire row and column to 0.

3. Implementation/Code:

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {

        int n = matrix.size();
        int m = matrix[0].size();
        vector<vector<int>> visited = matrix;
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                if(matrix[i][j] == 0){
                    for(int k=0; k<m; k++){
                        visited[i][k] = 0;
                    }
                }
            }
        }
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                if(matrix[i][j] == 0){
                    for(int k=0; k<n; k++){
                        visited[k][j] = 0;
                    }
                }
            }
        }
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                matrix[i][j] = visited[i][j];
            }
        }
    }
};
```

4. Output



5. Learning Outcome:

i. We Learn About the use of 2D-Array.
ii. We Learn About the use of Nested Loop.
iii. We Learn About the use of Vector Cases.

Question 2

1. Aim:- Detect a Cycle in a Linked List

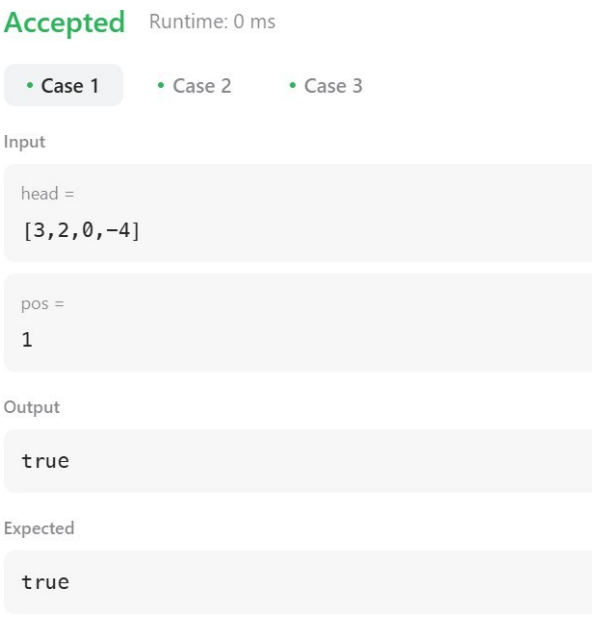
2. Objective:-

Given the head of a linked list, determine whether the linked list contains a cycle. A cycle occurs if a node's next pointer points to a previous node in the list.

3. Implementation/Code:-

```
class Solution {
    public:
       bool hasCycle(ListNode *head) {

      if (head == nullptr) {
          return false;
       }

       ListNode* slow = head;
       ListNode* fast = head;

       while (fast != nullptr && fast->next != nullptr) {
          slow = slow->next;
          fast = fast->next->next;

          if (slow == fast) {
             return true;
          }
       }

       return false;
       }
};
```

4. Output:-



5. **Learning Outcome:**
   1. We Learn about the Node Creation.
   2. We Learn about the Slow Fast Pointer.
   3. We learned about recursion.

Question 3

6. **Aim**:- **Search a 2D Matrix II**

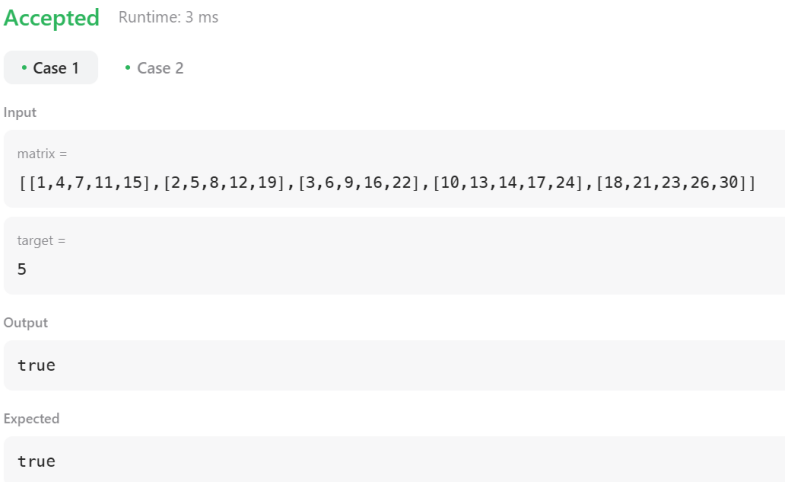
7. **Objective**:-

Given an m x n matrix where each row is sorted in ascending order from left to right and each column is sorted in ascending order from top to bottom, and an integer target, determine if the target exists in the matrix.

8. **Implementation/Code:-**

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {

        int row = matrix.size();
        int col = matrix[0].size();

        int s = 0;
        int e = col - 1;

        while(s<row && e >=0){

            int ele = matrix[s][e];

            if(ele == target){
                return 1;
            }
            if(ele < target){
                s++;
            }else{
                e--;
            }

        }
        return 0;
    }
};
```

9. Output:-



10. Learning Outcome:

- We learn about to traverse in 0(n).
- We learn about binary search function .

Question 4

11. Aim:- **Trapping Rain Water**

12. Objective:-

Given an m x n matrix where each row is sorted in ascending order from left to right and each column is sorted in ascending order from top to bottom, and an integer target, determine if the target exists in the matrix.

13. Implementation/Code:-

```
class Solution {
public:

    int trap(vector<int>& height) {

        int l=0;
        int r=height.size()-1;
        int lmax=INT_MIN;
        int rmax=INT_MIN;
        int ans=0;

        while(l<r){
            lmax=max(lmax,height[l]);
            rmax=max(rmax,height[r]);
            ans+=(lmax<rmax)?lmax-height[l++]:rmax-height[r--];
        }
        return ans;
    }
};
```

14. Output:-

15. Learning Outcome:

- We learn about to traverse in 0(n).
- We learn about while loop function .

Question 5

16. Aim:- **Word Break**

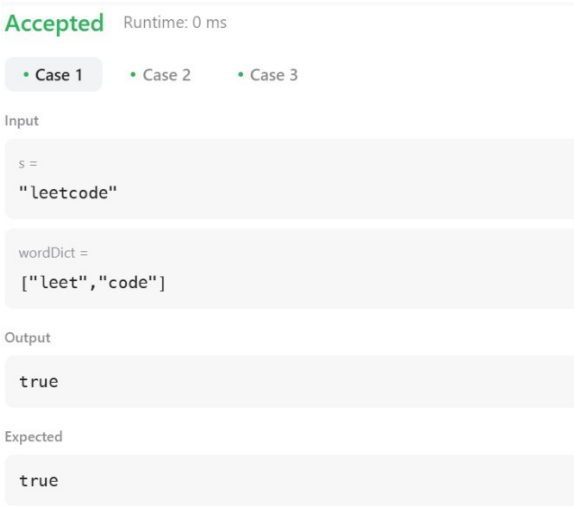
17. Objective:-

Given a string s and a dictionary wordDict containing a list of words, determine if s can be segmented into a space-separated sequence of one or more dictionary words. The same word can be reused multiple times.

18. Implementation/Code:-

```
class Solution {
public:
    bool wordBreak(string s, vector<string>& wordDict) {

        unordered_set<string> wordSet(wordDict.begin(), wordDict.end());
        vector<bool> dp(s.length() + 1, false);
        dp[0] = true;

        for (int i = 1; i <= s.length(); i++) {
            for (int j = 0; j < i; j++) {
                if (dp[j] && wordSet.find(s.substr(j, i - j)) !=
wordSet.end()) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[s.length()];
    }
};
```

19. Output:-



20. Learning Outcome:

- We learn about to Unordered_Set.
- We learn about DP .