

1 . Anagram Problem

Given two strings s1 and s2 consisting of lowercase characters. The task is to check whether two given strings are an anagram of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different. For example, act and tac are an anagram of each other. Strings s1 and s2 can only contain lowercase alphabets.

Note: You can assume both the strings s1 & s2 are non-empty.

Examples :

Input: s1 = "geeks", s2 = "kseeeg"

Output: true

Explanation: Both the string have same characters with same frequency. So, they are anagrams.

Input: s1 = "allergy", s2 = "allergic"

Output: false

Explanation: Characters in both the strings are not same, so they are not anagrams.



Input: s1 = "g", s2 = "g"

Output: true

Explanation: Character in both the strings are same, so they are anagrams.

```
class Solution {
public:
    // Function is to check whether two strings are anagram of
    bool areAnagrams(string& s1, string& s2) {
        int n=s1.size();
        vector<int> freq(26,0);
        if(s1.size()!=s2.size()) return false;
        for(int i=0;i<n;i++){
            freq[s1[i]-'a']++;
            freq[s2[i]-'a']--;
        }
        for(auto i:freq){
            if(i!=0) return false;
        }
        return true;
    }
};
```

Compilation Completed

For Input:  

geeks

kseeg

Your Output:

true

Expected Output:

true

Time Complexity : $O(n)$

Space Complexity: $O(1)$

2. Row with Maximum 1's

You are given a 2D array consisting of only 1's and 0's, where each row is sorted in non-decreasing order. You need to find and return the index of the first row that has the most number of 1s. If no such row exists, return -1.

Note: 0-based indexing is followed.

Examples:

Input: arr[][] = [[0, 1, 1, 1],
[0, 0, 1, 1],
[1, 1, 1, 1],
[0, 0, 0, 0]]

Output: 2

Explanation: Row 2 contains 4 1's.

Input: arr[][] = [[0, 0],
[1, 1]]

Output: 1

Explanation: Row 1 contains 2 1's.



Expected Time Complexity: $O(n+m)$

Expected Auxiliary Space: $O(1)$

Note : Here n,m refers to the number of rows and columns respectively.

```
class Solution {
public:
    int rowWithMax1s(vector<vector<int> > &arr) {
        int maxval=-1;
        int r=0,c=arr[0].size()-1;
        while(r<arr.size() && c>=0){
            if(arr[r][c]==0){
                r++;
            }
            else{
                maxval=r;
                c--;
            }
        }
        return maxval;
    }
};
```

Compilation Completed

For Input:  

4 4

0 1 1 1

0 0 1 1

1 1 1 1

0 0 0 0

Your Output:

2

Expected Output:

2

Time Complexity: $O(n+m)$

Space Complexity: $O(1)$

3. Longest consecutive subsequence

Given an array `arr` of non-negative integers. Find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Examples:

Input: `arr[] = [2, 6, 1, 9, 4, 5, 3]`

Output: 6

Explanation: The consecutive numbers here are 1, 2, 3, 4, 5, 6. These 6 numbers form the longest consecutive subsequence.

Input: `arr[] = [1, 9, 3, 10, 4, 20, 2]`

Output: 4

Explanation: 1, 2, 3, 4 is the longest consecutive subsequence.



Input: `arr[] = [15, 13, 12, 14, 11, 10, 9]`

Output: 7

Explanation: The longest consecutive subsequence is 9, 10, 11, 12, 13, 14, 15, which has a length of 7.

```
class Solution {
public:
    // Function to return length of longest subsequence of
    int findLongestConseqSubseq(vector<int>& arr) {
        map<int,int> map;
        int maxlen=0;
        for(auto i:arr) map[i]++;
        for(int i:arr){
            int len=0;
            while(map[i]){
                len++;
                i++;
            }
            maxlen=max(maxlen,len);
        }
        return maxlen;
    }
};
```

Compilation Completed

For Input:  

2 6 1 9 4 5 3

Your Output:

6

Expected Output:

6

Time Complexity: $O(n)$

Space Complexity: $O(n)$

4. longest palindrome in a string

Given a string s , your task is to find the longest palindromic substring within s . A substring is a contiguous sequence of characters within a string, defined as $s[i..j]$ where $0 \leq i \leq j < \text{len}(s)$.

A palindrome is a string that reads the same forward and backward. More formally, s is a palindrome if $\text{reverse}(s) == s$.

Note: If there are multiple palindromes with the same length, return the first occurrence of the longest palindromic substring from left to right.

Examples :

Input: $s = \text{"aaaabbaa"}$

Output: "aabbaa"

Explanation: The longest palindromic substring is "aabbaa" .

Input: $s = \text{"abc"}$

Output: "a"

Explanation: "a" , "b" , and "c" are all palindromes of the same length, but "a" appears first.



Input: $s = \text{"abacdfgdcaba"}$

Output: "aba"

Explanation: The longest palindromic substring is "aba" , which occurs twice. The first occurrence is returned.

```
class Solution {
public:
    string longestPalindrome(string s) {
        if (s.size()==0) {
            return "";
        }
        int len=0;
        int left,right;
        string res;
        for (int i = 0; i < s.size(); i++) {
            left=i;
            right=i;
            while (left >= 0 && right < s.size() && s[left] == s[right]) {
                if(right-left+1>len){
                    res=s.substr(left, right - left + 1);
                    len=right-left+1;
                }
                left--;
                right++;
            }
            left=i;
            right=i+1;
            while (left >= 0 && right < s.size() && s[left] == s[right]) {
                if(right-left+1>len){
                    res=s.substr(left, right - left + 1);
                    len=right-left+1;
                }
                left--;
                right++;
            }
        }
        return res;
    }
};
```

Compilation Completed

For Input:  

aaaabbaa

Your Output:

aabbaa

Expected Output:

aabbaa

Time Complexity: $O(n^2)$

Space Complexity : $O(1)$

5. rat in a maze problem

Consider a rat placed at (0, 0) in a square matrix mat of order $n \times n$. It has to reach the destination at (n - 1, n - 1). Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are 'U'(up), 'D'(down), 'L' (left), 'R' (right). Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it.

Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell. In case of no path, return an empty list. The driver will output "-1" automatically.

Examples:

Input: mat[][] = [[1, 0, 0, 0],
[1, 1, 0, 1],
[1, 1, 0, 0],
[0, 1, 1, 1]]

Output: DDRDRR DRDDRR

Explanation: The rat can reach the destination at (3, 3) from (0, 0) by two paths - DRDDRR and DDRDRR, when printed in sorted order we get DDRDRR DRDDRR.

Input: mat[][] = [[1, 0],
[1, 0]]

Output: -1

Explanation: No path exists and destination cell is blocked.

Expected Time Complexity: $O(3^{n^2})$

Expected Auxiliary Space: $O(l \times x)$

Here l = length of the path, x = number of paths.

```
class Solution {
public:
    void helper(int r, int c, int m, int n, string str, vector<vector<int>>& mat, vector<vector<bool>>& visited, vector<string>& res) {
        if (r == m - 1 && c == n - 1) {
            res.push_back(str);
            return;
        }

        if (r < 0 || c < 0 || r >= m || c >= n || mat[r][c] == 0 || visited[r][c]) return;

        visited[r][c] = true;

        if (r < m - 1) helper(r + 1, c, m, n, str + "D", mat, visited, res);
        if (c < n - 1) helper(r, c + 1, m, n, str + "R", mat, visited, res);
        if (r > 0) helper(r - 1, c, m, n, str + "U", mat, visited, res);
        if (c > 0) helper(r, c - 1, m, n, str + "L", mat, visited, res);



        visited[r][c] = false;
    }

    vector<string> findPath(vector<vector<int>>& mat) {
        vector<string> res;
        int m = mat.size();
        int n = mat[0].size();
        if (mat[0][0] == 0 || mat[m - 1][n - 1] == 0) return res;

        vector<vector<bool>> visited(m, vector<bool>(n, false));
        helper(0, 0, m, n, "", mat, visited, res);

        sort(res.begin(), res.end());
        return res;
    }
};
```

Compilation Completed

For Input:  

4

1 0 0 0

1 1 0 1

1 1 0 0

0 1 1 1

Your Output:

DDRDRR DRDDRR

Expected Output:

DDRDRR DRDDRR

Time Complexity: $O(2^{m*n})$

Space Complexity : $O(m*n + l*x)$