Anagram

Difficulty: **Easy** Accuracy: **44.93%** Submissions: **338K+** Points: **2**

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 = "kseeg"
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.

Constraints:
$1 \le s1.size(), s2.size() \le 10^5$

Code:

```
bool areAnagrams(string& s1, string& s2) {
        if(s1.size()!=s2.size())return false;
        vector<int> arr1(26,0),arr2(26,0);
        for(int i=0;i<s1.size();i++){
            arr1[s1[i]-'a']++;
            arr2[s2[i]-'a']++;
        }
        for(int i=0;i<26;i++){
            if(arr1[i]!=arr2[i])return false;
        }
        return true;
}
```

TimeComplexity:O(n)
SpaceComplexity:O(26)

Longest consecutive subsequence

Difficulty: **Medium** Accuracy: **33.0%** Submissions: **309K+** Points: **4**

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 subsquence.

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.

Constraints:
$1 <= arr.size() <= 10^5$
$0 <= arr[i] <= 10^5$

Code:

```cpp
int findLongestConseqSubseq(vector<int>& arr) {
        unordered_set<int> st;
        for(auto a:arr){
            st.insert(a);
        }
        int maxi=1;
        for(int i=0;i<arr.size();i++){
            if(st.find(arr[i]-1)!=st.end()){
                continue;
            }
            int ans=1;
            int num=arr[i];
            while(st.find(num+1)!=st.end()){
                num++;
                ans++;
            }
            maxi=max(maxi,ans);
        }
        return maxi;
}
```

TimeComplexity:O(n)
SpaceComplexity:O(n)

Row with max 1s

Difficulty: **Medium** Accuracy: **33.09%** Submissions: **318K+** Points: **4**

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.

Code:

```
int rowWithMax1s(vector<vector<int> > &arr) {
        for(int i=0;i<arr[0].size();i++){
            for(int j=0;j<arr.size();j++){
                if(arr[j][i]==1){
                    return j;
                }
            }
        }
        return -1;
}
```

TimeComplexity:O(n*m)
SpaceComplexity:O(1)

Longest Palindrome in a String

Difficulty: **Medium** Accuracy: **23.2%** Submissions: **306K+** Points: **4**

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 $0 \le i \le j < \text{len(s)}$.

A **palindrome** is a string that reads the same forward and backward. More formally, s is a palindrome if 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 = "aaaabbaa"
Output: "aabbaa"
Explanation: The longest palindromic substring is "aabbaa".

Code:

```
string longestPalindrome(string s) {
        vector<vector<int>> arr(s.size(),vector<int>(s.size(),-1));
        int start=0,count=1;
        for(int i=0;i<s.size();i++){
            arr[i][i]=1;
        }
        for(int i=0;i<s.size()-1;i++){
            if(s[i]==s[i+1]){
                arr[i][i+1]=2;
                start=i;
                count=2;
            }
        }
        for(int i=3;i<=s.size();i++){
            for(int j=0;j<s.size()-i+1;j++){
                int k=j+i-1;
                if(arr[j+1][k-1]!=-1 and s[k]==s[j]){
                    arr[j][k]=arr[j+1][k-1]+2;
                    if(arr[j][k]>count){
                        count=arr[j][k];
                        start=j;
                    }
                }
            }
        }
        string ans="";
        for(int i=start;i<start+count;i++){
            ans+=s[i];
        }
        return ans;
}
```

TimeComplexity:O(n2)**
SpaceComplexity:O(n2)**

Rat in a Maze Problem - I

Difficulty: **Medium** Accuracy: **35.75%** Submissions: **302K+** Points: **4**

Consider a rat placed at **(0, 0)** in a square matrix **mat** of order **n* 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.

```
void rec(int r,int c,int n, vector<string> &ans,vector<vector<int>> &visited,string move,vector<vector<int>> &mat){
        if(r==n-1 and c==n-1){
            ans.push_back(move);
            return ;
        }

        if(r+1<n and visited[r+1][c]==0 and mat[r+1][c]==1){
            visited[r][c]=1;
            rec(r+1,c,n,ans,visited,move+'D',mat);
            visited[r][c]=0;
        }

        if(c-1>=0 and visited[r][c-1]==0 and mat[r][c-1]==1){
            visited[r][c]=1;
            rec(r,c-1,n,ans,visited,move+'L',mat);
            visited[r][c]=0;
        }

        if(c+1<n and visited[r][c+1]==0 and mat[r][c+1]==1){
            visited[r][c]=1;
            rec(r,c+1,n,ans,visited,move+'R',mat);
            visited[r][c]=0;
        }

        if(r-1>=0 and visited[r-1][c]==0 and mat[r-1][c]==1){
            visited[r][c]=1;
            rec(r-1,c,n,ans,visited,move+'U',mat);
            visited[r][c]=0;
        }
    }
```

TimeComplexity:O(4(n*m))**
SpaceComplexity:O(n2)**