1. **Letters Collection :**

The Postmaster wants to write a program to answer the queries regarding letter collection in a city. A city is represented as a matrix **mat** of size**n\*m.** Each cell represents a house and contains letters which are denoted by a number in the cell. The program should answer **q**queries which are of following types:  
**1 i j**: To sum all the letters which are at a 1-hop distance from the cell (i,j) in any direction  
**2 i j :**To sum all the letters which are at a 2-hop distance from the cell (i,j) in any direction   
The queries are given in a 2D matrix **queries[][]**.  
In one hop distance postmaster can go to any of [(i-1,j-1), (i-1,j), (i-1,j+1), (i,j-1), (i,j+1), (i+1,j-1), (i+1,j), (i+1,j+1)] from (i,j).

**Example 1:**

**Input:**

n = 4, m = 5

mat = {{1, 2, 3, 4, 10},

  {5, 6, 7, 8, 10},

  {9, 1, 3, 4, 10},

  {1, 2, 3, 4, 10}}

q = 2

queries = {{1 0 1},

  {2 0 1}}

**Output:**   
22 29

**Explaination:**

For the first query sum is 1+5+6+7+3 = 22.

For the second query sum is 9+1+3+4+8+4 = 29.

**Example 2:**

**Input:**

n = 6, m = 6

mat = {{ 1, 2, 3, 4, 5, 6},

  { 7, 8, 9, 10, 11, 12},

  {13, 14, 15, 16, 17, 18},

  {19, 20, 21, 22, 23, 24},  
 {25, 26, 27, 28, 29, 30},  
 {31, 32, 33, 34, 35, 36}}

q = 1

queries = {{2 3 2}}

**Output:**   
336

**Explaination:**

The first query sum is 7+8+9+10+11+17+23+29+35+34+33+32+31+25+19+13 = 336.

**Your Task:**  
You do not need to read input or print anything. Your task is to complete the function **matrixSum()** which takes **n**, **m**, **mat**, **q**and **queries**as input parameters and returns a list of integers where the ith value is the answers for ith query.

**Expected Time Complexity:** O(q)  
**Expected Auxiliary Space:** O(q)

**Constraints:**  
1 ≤ n, m ≤ 103  
0 ≤ mat[i][j] ≤ 107  
1 ≤ q ≤ 105  
1 ≤ q[i][0] ≤ 2  
0 ≤ q[i][1] < n  
0 ≤ q[i][2] < m

Approach 1:

1. The brute force approach is to calculate the left right up and down index though given hop.
2. check the condition of where left, right, up and down are not outside the range and also check the index with in range that is from 0 to n (row) or 0 to m (column).
3. And take a variable sum and calculate sum all the value and push into the answer vector.
4. The above three statement will iterate number of given queries times.
5. Return the answer vector.

Code:

vector<int> matrixSum(int n, int m, vector<vector<int>> mat, int q, vector<int> queries[])

{

// code here

vector<int> ans1;

int left,right,up,down;

for(int i=0;i<q;i++)

{

left=queries[i][2]-queries[i][0];

right=queries[i][2]+queries[i][0];

up=queries[i][1]-queries[i][0];

down=queries[i][1]+queries[i][0];

int ans=0;

for(int j=left;j<=right&&down<n;j++)

{

if(j>=0&&j<m){

ans=ans+mat[down][j];

}

}

for(int j=left;j<=right&&up>=0;j++)

{

if(j>=0&&j<m){

ans=ans+mat[up][j];

}

}

for(int j=up+1;j<down&&(right<m);j++)

// important to skip the already taken element.

{

if(j>=0&&j<n){

ans=ans+mat[j][right];

}

}

for(int j=up+1;j<down&&(left>=0);j++)

{

if(j>=0&&j<n){

ans=ans+mat[j][left];

}

}

ans1.push\_back(ans);

}

return ans1;

}

==============================================================

### Number following a pattern ID problem.

Given a pattern containing only **I**'s and **D**'s. I for **increasing** and D for **decreasing**. Devise an algorithm to print the **minimum number** following that pattern. Digits from **1-9** and digits **can't repeat**.

**Example 1:**

**Input:**

D

**Output:**

21

**Explanation:**

D is meant for decreasing, so we choose the minimum number among all possible numbers like 21,31,54,87,etc.

**Example 2:**

**Input:**

IIDDD

**Output:**

126543

**Explanation:**

Above example is self- explanatory,

1 < 2 < 6 > 5 > 4 > 3

I - I - D - D - D

**Your Task:**

You don't need to read input or print anything. Your task is to complete the function **printMinNumberForPattern()** which takes the string **S** and returns a string containing the minimum number following the valid number.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 ≤ Length of String ≤ 8

Approach 1:

1. Using stack.
2. One observation is to size of answer is size of string +1.
3. Take one variable the increment the value each time push into the stack.
4. It I is in counter then pop all the elements from the stake and store into answer.
5. Finally, increment the last time of value and store into stack again pop all the element and store answer.

Code:

string printMinNumberForPattern(string S){

// code here

int n=S.size();

string ans="";

stack<int> st;

int num=1;

for(int i=0;i<n;i++)

{

st.push(num);

num++;

if(S[i]=='I')

{

while(!st.empty())

{

ans+='0'+st.top();

st.pop();

}

}

}

st.push(num);

while(!st.empty())

{

ans+='0'+st.top();

st.pop();

}

return ans;

}

**Time Complexity:** O(N)  
**Auxiliary Space:** O(N)

Approach 2:

1. Initial two string type and one integer type variable.
2. Iterate the for loop from 0 to n
3. And increment the count variable each time convert into string and store into temp.
4. If D is encounter into string swap the value until the I not encounter in reverse order of string.
5. If I is encounter and the answer and increment the count value.
6. End.

Code:

string printMinNumberForPattern(string S){

// code here

int n=S.size();

string ans="";

int count=1;

string temp="1";

for(int i=0;i<n;i++)

{

if(S[i]=='D')

{

count++;

temp=to\_string(count)+temp;

//its perform to reverse the string until I not encounter in reverse order.

}

else

{

ans+=temp;

count++;

temp=to\_string(count);

}

}

ans+=temp;

return ans;

}

**Time Complexity:** O (N)  
**Auxiliary Space:** O (1)

============================================================

### Check if string is rotated by two places

Given two strings **a**and **b**. The task is to find if the string 'b' can be obtained by rotating (**in any direction**) string 'a' by **exactly 2** places.

**Example 1:**

**Input:**

a = amazon

b = azonam

**Output:**1

**Explanation:**amazon can be rotated anti-clockwise by two places, which will make it as azonam.

**Example 2:**

**Input:**

a = geeksforgeeks

b = geeksgeeksfor

**Output:**0

**Explanation:**If we rotate geeksforgeeks by two place in any direction, we won't get geeksgeeksfor.

**Your Task:**  
The task is to complete the function **isRotated()** which takes two strings as input parameters and checks if given strings can be formed by rotations. The function returns true if string 1 can be obtained by rotating string 2 by two places, else it returns false.

**Expected Time Complexity:** O(N).  
**Expected Auxilary Complexity:** O(N).  
**Challenge:**Try doing it in O(1) space complexity.

**Constraints:**  
1 ≤ length of a, b ≤ 105

Approach 1:

1. Using two string that will update by two clockwise and anticlockwise.
2. Check weather is equal to second string.

Code:

bool isRotated(string str1, string str2)

{

// Your code here

int n=str1.length();

int m=str2.length();

if(n!=m||n==1||str1.compare(str2)==0)return 0;

string clockwise="", anticlockwise="";

clockwise=clockwise+str1.substr(2,n)+str1.substr(0,2);

anticlockwise=anticlockwise+str1.substr(n-2,n)+str1.substr(0,n-2);

return str2.compare(clockwise)==0||str2.compare(anticlockwise)==0;

}

Approach 2:

Optimise the space:

bool isRotated(string str1, string str2)

{

// Your code here

int n=str1.length();

int m=str2.length();

if(n!=m||n==1||str1.compare(str2)==0)return 0;

int clockwise=1, anticlockwise=1;

for(int i=0;i<n;i++)

{

if(str1[i]!=str2[(i+2)%n])

{

clockwise=0;

break;

}

}

for(int i=0;i<n;i++)

{

if(str1[(i+2)%n]!=str2[i])

{

anticlockwise=0;

break;

}

}

return clockwise==1||anticlockwise==1;

}

============================================================

### Shortest Common Supersequence

Given two strings **X** and **Y** of lengths **m** and **n** respectively, find the length of the **smallest string**which has both, **X and Y**as its **sub-sequences**.  
**Note:** **X** and **Y**can have both uppercase and lowercase letters.

**Example 1**

**Input:**

X = abcd, Y = xycd

**Output:** 6

**Explanation:** Shortest Common Supersequence

would be **abxycd** which is of **length 6** and

has both the strings as its subsequences.

**Example 2**

**Input:**

X = efgh, Y = jghi

**Output:** 6

**Explanation:** Shortest Common Supersequence

would be **ejfghi** which is of **length 6** and

has both the strings as its subsequences.

**Your Task:**  
You don't have to take any input or print anything. Your task is to complete **shortestCommonSupersequence()** function that takes **X, Y, m,**and **n**as arguments and returns the **length**of the required string.

**Expected Time Complexity:**O(Length(X) \* Length(Y)).  
**Expected Auxiliary Space:**O(Length(X) \* Length(Y)).

**Constraints:**  
1<= |X|, |Y| <= 100

Approach 1:

Bottom up:

1. Find the longest common sub sequence of two string.
2. Return the answer after compute the sum of length of both the string minus longest sub sequence.

Code of longest sub sequence:

int fun(string X ,string Y,int i,int j)

{

if(i==X.size())

{

return 0;

}

if(j==Y.size())

{

return 0;

}

int ans=0;

if(X[i]==Y[j])

{

ans=1+fun(X,Y,i+1,j+1,dp);

}

else

{

ans = max(fun(X,Y,i+1,j,dp),fun(X,Y,i,j+1,dp));

}

return ans;

}

int shortestCommonSupersequence(string X, string Y, int m, int n)

{

return m+n-fun(X,Y,0,0,dp);

// return m+n-len;

}

Memorize the solution using dp:

int fun(string X ,string Y,int i,int j,vector<vector<int>> &dp)

{

if(i==X.size())

{

return 0;

}

if(j==Y.size())

{

return 0;

}

if(dp[i][j]!=-1)

{

return dp[i][j];

}

int ans=0;

if(X[i]==Y[j])

{

ans=1+fun(X,Y,i+1,j+1,dp);

}

else

{

ans = max(fun(X,Y,i+1,j,dp),fun(X,Y,i,j+1,dp));

}

return dp[i][j]=ans;

}

int shortestCommonSupersequence(string X, string Y, int m, int n)

{

//code here

vector<vector<int>> dp(m,vector<int>(n,-1));

return m+n-fun(X,Y,0,0,dp);

// return m+n-len;

}

Top Down Approach:

======================================================================

### Better String

Given a pair of strings of equal lengths, Geek wants to find the **better string**. The **better string**is the string having more number of **distinct** **subsequences**.  
If both the strings have equal count of distinct subsequence then return **str1**.

**Example 1:**

**Input:**

str1 = "gfg", str2 = "ggg"

**Output:** "gfg"

**Explanation: "**gfg" have 7 distinct subsequences whereas "ggg" have 4 distinct subsequences.

**Example 2:**

**Input:** str1 = "a", str2 = "b"

**Output:** "a"

**Explanation:** Both the strings have only 1 distinct subsequence.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **betterString()** which takes **str1** and **str2** as input parameters and returns the better string.

**Expected Time Complexity:** O( max( str1.length, str2.length ) )

**Expected Auxiliary Space:** O( max( str1.length, str2.length ) )

**Constraints:**  
1 <= str1.length , str2.length <= 30

Approach 1:

Using print subsequence method:

1. Either we will consider the character in the string or not.

Code:

// User function Template for C++

class Solution {

public:

void find1(string input,string output,set<string> &s)

{

if(input.size()==0)

{

s.insert(output);

return;

}

find1(input.substr(1),output+input[0],s);//consider the character

find1(input.substr(1),output,s); // not consider the character

}

string betterString(string str1, string str2) {

// code here

set<string> s1,s2;

find1(str1,"",s1);

find1(str2,"",s2);

return s1.size()>=s2.size()?str1:str2;

}

};

Approach 2:

Using dp :

1. We check each character of total number of possible sub sequence
2. If the any char of sub sequence is found then subtract the number of sub sequence to present sub sequence.

Code:

int distinct(string s)

{

int n=s.length();

int dp[n+1]={0};

dp[0]=1;

unordered\_map<char,int>m;

for(int i=1;i<n+1;i++)

{

dp[i]=2\*dp[i-1];

if(m.find(s[i-1])!=m.end())

{

dp[i]=dp[i]-dp[m[s[i-1]]-1];

}

m[s[i-1]]=i;

}

return dp[n];

}

string betterString(string str1, string str2) {

// code here

return distinct(str1)>=distinct(str2)?str1:str2;

}

Approach 3:

Optimise the some space like dp:

Code:

int distinct(string s)

{

int total\_dis=0;

int n=s.length();

vector<int>v(26,0);

for(int i=0;i<n;i++)

{

int present=total\_dis+1;

if(v[s[i]-'a'])

{

total\_dis+=(present-v[s[i]-'a']);

}

else

{

total\_dis+=present;

}

v[s[i]-'a']=present;

}

return total\_dis;

}

============================================================

### Find the String

Date of problem : 16 November 2023

==========================================================

### Shuffle integers:

**G**iven an array **arr** of **n**elements in the following format {a1, a2, a3, a4, ... , an/2, b1, b2, b3, b4, ... , bn/2}, the task is shuffle the array to {a1, b1, a2, b2, a3, b3, ... , an/2, bn/2} without using extra space.  
Note that n is **even**.

**Example 1:**

**Input:**n = 4, arr = {1, 2, 9, 15}

**Output:**   
1 9 2 15

**Explanation**:   
a1=1, a2=2, b1=9, b2=15. So the final array will be: a1, b1, a2, b2 = {1,9,2,15}.

**Example 2:**

**Input:**n = 6 arr = {1, 2, 3, 4, 5, 6}   
**Output:**   
1 4 2 5 3 6

**Your Task:**  
This is a function problem. You don't need to take any input, as it is already accomplished by the driver code. You just need to complete the function **shuffleArray()**that takes array**arr,**and an integer**n** as parameters and modifies the given array according to the above-given pattern.

**Expected Time Complexity:** O(n).  
**Expected Auxiliary Space:** O(1).

**Constraints:**  
1 ≤ n ≤ 105  
1 ≤ arri≤ 103

code:Approach:

void shuffleArray(int arr[],int n){int m=n/2;

for(int i=0;i<n/2;i++) {

arr[m+i]+=arr[i]\*1001;}

for(int i=0;i<n/2;i++){

arr[2\*i]=arr[m+i]/1001;

arr[2\*i+1]=arr[m+i]%1001; }

}

Time Complexity : O(N)

space Complexity: O (1)

======================================================================================

### Euler circuit and Path

**Eulerian Cycle:**An undirected graph has Eulerian cycle if following two conditions are true.

1. All vertices with non-zero degree are connected. We don’t care about vertices with zero degree because they don’t belong to Eulerian Cycle or Path (we only consider all edges).
2. All vertices have even degree.

**Eulerian Path:**An undirected graph has Eulerian Path if following two conditions are true.

1. Same as condition (a) for Eulerian Cycle.
2. If zero or two vertices have odd degree and all other vertices have even degree. Note that only one vertex with odd degree is not possible in an undirected graph (sum of all degrees is always even in an undirected graph)

Code:

int ec=0,ep=0;

for(int i=0;i<V;i++){

if(adj[i].size()%2==0){ec++;}

else{ep++;}}

if(ec==V)return 2;

if(ep==2)return 1;

return 0;