**Coding practice Problems 13-11-2024**

**1. Anagram**

class Solution {

public:

bool areAnagrams(string& s1, string& s2) {

if (s1.size() != s2.size()) return false;

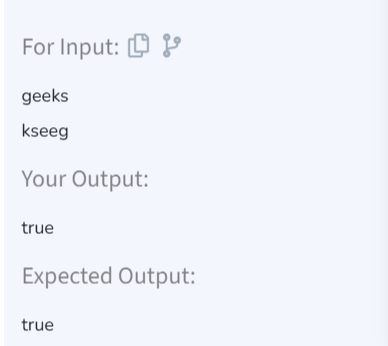
sort(s1.begin(), s1.end());

sort(s2.begin(), s2.end());

return s1 == s2;

}

};



TIME COMPLEXITY: O(n log n)  
SPACE COMPLEXITY: O(1)

**2. Row with Max 1s**

class Solution {

public:

int rowWithMax1s(vector<vector<int>>& arr) {

int n = arr.size();

int m = arr[0].size();

int maxRow = -1, j = m - 1;

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

while (j >= 0 && arr[i][j] == 1) {

--j;

maxRow = i;

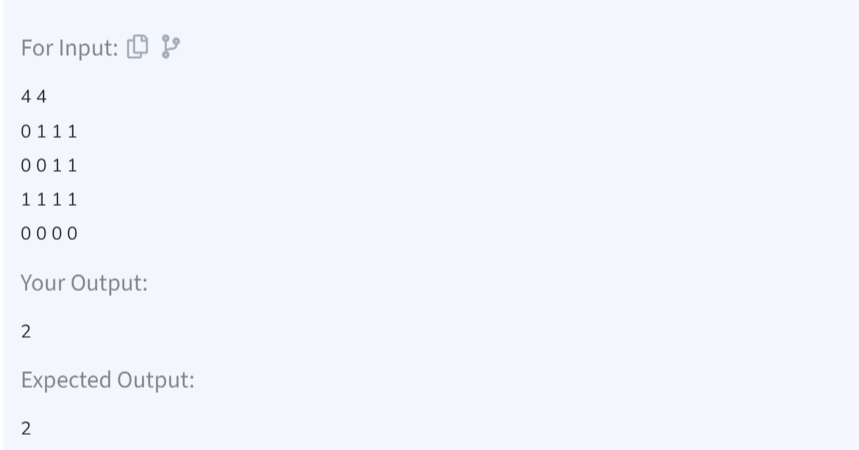
}

}

return maxRow;

}

};



TIME COMPLEXITY: O(n+m)  
SPACE COMPLEXITY: O(1)

**3. Longest Consecutive Subsequence**

class Solution {

public:

int findLongestConseqSubseq(vector<int>& arr) {

unordered\_set<int> s(arr.begin(), arr.end());

int maxLength = 0;

for (int num : s) {

if (s.find(num - 1) == s.end()) {

int currentNum = num, currentStreak = 1;

while (s.find(currentNum + 1) != s.end()) {

currentNum++;

currentStreak++;

}

maxLength = max(maxLength, currentStreak);

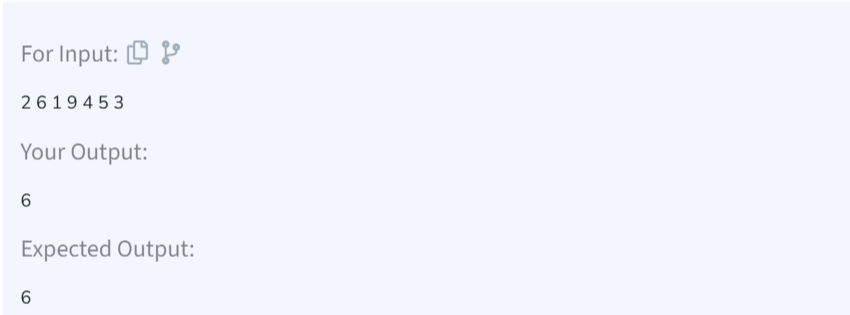
}

}

return maxLength;

}

};

****

TIME COMPLEXITY: O(n)  
SPACE COMPLEXITY: O(n)

**4. Longest Palindromic Subsequence**

class Solution{

public:

string longestPalindrome(string S) {

int n = S.size(), start = 0, maxLength = 1;

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

int len1 = expandAroundCenter(S, i, i);

int len2 = expandAroundCenter(S, i, i + 1);

int len = max(len1, len2);

if (len > maxLength) {

maxLength = len;

start = i - (len - 1) / 2;

}

}

return S.substr(start, maxLength);

}

int expandAroundCenter(string& s, int left, int right) {

while (left >= 0 && right < s.size() && s[left] == s[right]) {

left--;

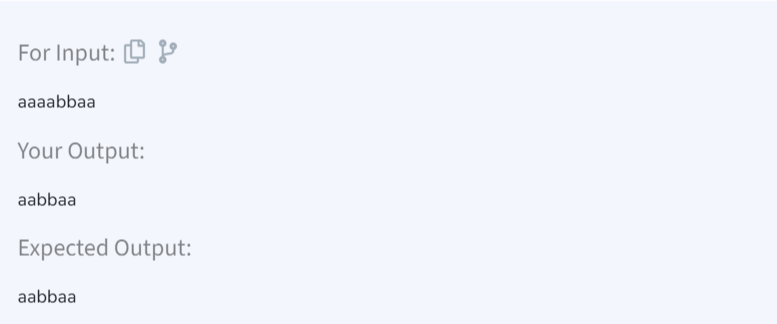
right++;

}

return right - left - 1;

}

};

****

TIME COMPLEXITY: O(n^2)  
SPACE COMPLEXITY: O(1)

**5. Rat in a Maze**

class Solution {

public:

void findPathsUtil(vector<vector<int>>& mat, vector<string>& result, string& path, int i, int j) {

int n = mat.size(), m = mat[0].size();

if (i < 0 || i >= n || j < 0 || j >= m || mat[i][j] == 0) return;

if (i == n - 1 && j == m - 1) {

result.push\_back(path);

return;

}

mat[i][j] = 0;

path.push\_back('D');

findPathsUtil(mat, result, path, i + 1, j);

path.pop\_back();

path.push\_back('L');

findPathsUtil(mat, result, path, i, j - 1);

path.pop\_back();

path.push\_back('R');

findPathsUtil(mat, result, path, i, j + 1);

path.pop\_back();

path.push\_back('U');

findPathsUtil(mat, result, path, i - 1, j);

path.pop\_back();

mat[i][j] = 1;

}

vector<string> findPath(vector<vector<int>>& mat) {

vector<string> result;

if (mat[0][0] == 1) {

string path = "";

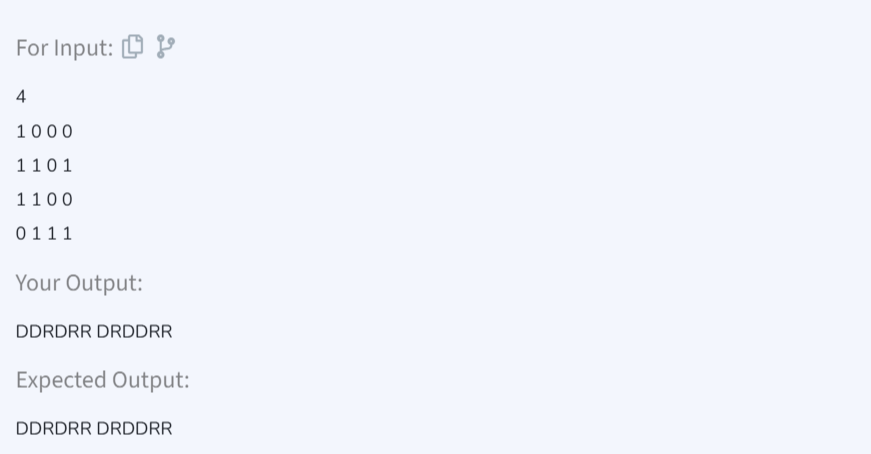
findPathsUtil(mat, result, path, 0, 0);

}

return result;

}

};



TIME COMPLEXITY: O(4^(n \* m))  
SPACE COMPLEXITY: O(n\*m)