TECHNICAL TRAINING DSA- CODING PRACTICE PROBLEMS

Name: Sachin A

Dept: CSBS

Date: 12-11-2024

Anagram Program

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

bool checkAnagrams(string &str1, string &str2) {
    unordered_map<char, int> freqMap;

    for(char ch: str1)
        freqMap[ch] += 1;

    for(char ch: str2)
        freqMap[ch] -= 1;

    for (auto& pair : freqMap) {
        if (pair.second != 0) {
            return false;
        }
    }
```

```
    return true;
}

int main() {
    string str1 = "geeks";
    string str2 = "kseeg";
    cout << (checkAnagrams(str1, str2) ? "true" : "false") << endl;

    return 0;
}
```

OUTPUT:

```
PS D:\c++\output> & .\'ex4.exe'
true
PS D:\c++\output>
```

row with max 1s':

```
#include <bits/stdc++.h>
using namespace std;

int findFirstOne(vector<bool>& arr, int start, int end) {
    int position = -1;
    while (start <= end) {
        int middle = start + (end - start) / 2;
        if (arr[middle] == 1) {
            position = middle;
            end = middle - 1;
        } else {
```

```
            start = middle + 1;
        }
    }
    return position;
}

int rowWithMostOnes(vector<vector<bool>>& matrix) {
    int maxRow = -1, maxOnes = -1;
    int rowCount = matrix.size();
    int colCount = matrix[0].size();

    for (int i = 0; i < rowCount; i++) {
        int firstOneIndex = findFirstOne(matrix[i], 0, colCount - 1);
        if (firstOneIndex != -1 && colCount - firstOneIndex > maxOnes) {
            maxOnes = colCount - firstOneIndex;
            maxRow = i;
        }
    }

    return maxRow;
}

int main() {
    vector<vector<bool>> matrix = { { 0, 0, 0, 1 },
                                    { 0, 1, 1, 1 },
                                    { 1, 1, 1, 1 },
                                    { 0, 0, 0, 0 } };

    cout << rowWithMostOnes(matrix);
```

```
    return 0;
}
```

OUTPUT:

```
PS D:\c++\output> cd 'd:\c++\output'
PS D:\c++\output> & .\'ex4.exe'
2
```

Longest consequtive subsequence:

```
#include <bits/stdc++.h>
using namespace std;

int longestContiguousSubsequence(int arr[], int n) {
    int maxLength = 0, currentLength = 0;

    sort(arr, arr + n);

    vector<int> uniqueElements;
    uniqueElements.push_back(arr[0]);

    for (int i = 1; i < n; i++) {
        if (arr[i] != arr[i - 1])
            uniqueElements.push_back(arr[i]);
    }

    for (int i = 0; i < uniqueElements.size(); i++) {
        if (i > 0 && uniqueElements[i] == uniqueElements[i - 1] + 1)
            currentLength++;
```

```
        else
            currentLength = 1;

        maxLength = max(maxLength, currentLength);
    }
    return maxLength;
}

int main() {
    int arr[] = { 1, 2, 2, 3 };
    int n = sizeof arr / sizeof arr[0];
    cout << "Length of the Longest contiguous subsequence is "
         << longestContiguousSubsequence(arr, n);
    return 0;
}
```

OUTPUT:

```
PS D:\c++\output> & .\'ex4.exe'
Length of the Longest contiguous subsequence is 3
```

longest palindrome in a string:

```
#include <bits/stdc++.h>
using namespace std;

string findLongestPalindrome(string &s) {
    int length = s.length();
    if (length == 0) return "";

    int startIndex = 0, maxLength = 1;
```

```
    for (int i = 0; i < length; i++) {
        for (int offset = 0; offset <= 1; offset++) {
            int left = i;
            int right = i + offset;

            while (left >= 0 && right < length && s[left] == s[right]) {
                int currentLength = right - left + 1;
                if (currentLength > maxLength) {
                    startIndex = left;
                    maxLength = currentLength;
                }
                left--;
                right++;
            }
        }
    }

    return s.substr(startIndex, maxLength);
}

int main() {
    string input = "forgeeksskeegfor";
    cout << findLongestPalindrome(input) << endl;
    return 0;
}
```

OUTPUT:

```
PS D:\c++\output> & .\'ex4.exe'
geeksskeeg
PS D:\c++\output>
```