TECHNICAL TRAINING DSA - CODING PRACTICE

PROBLEMS

Name: Rishi Kumar S
Dept: CSBS
Date: 12-11-2024

Question 1:
Anagram program
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.

Code:

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

class Solution {
  public:or not.
    bool areAnagrams(string& s1, string& s2) {
        // Your code here
        unordered_map<char,int> map;
        for(char s:s1) map[s]++;
        for(char s:s2) map[s]--;
        for(auto it=map.begin();it!=map.end();it++){
            if(it->second!=0) return false;
        }
        return true;
    }
};

int main() {

    int t;

    cin >> t;

    while (t--) {
```

```
        string c, d;

        cin >> c >> d;
        Solution obj;
        if (obj.areAnagrams(c, d))
            cout << "true" << endl;
        else
            cout << "false" << endl;
        cout << "~" << endl;
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

Question 2:
Row with max 1s'

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.

Code:

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

class Solution {
  public:
    int rowWithMax1s(vector<vector<int> > &arr) {
        // code here
        int n =arr.size();
        int m =arr[0].size();
        int j=m-1;
        int tracker=-1;
        for(int i=0;i<n;i++){
            while(j>=0 && arr[i][j]==1){
                j--;
                tracker=i;
            }
        }
        return tracker;
```

```
    }
};

//{ Driver Code Starts.
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n, m;
        cin >> n >> m;
        vector<vector<int> > arr(n, vector<int>(m));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                cin >> arr[i][j];
            }
        }
        Solution ob;
        auto ans = ob.rowWithMax1s(arr);
        cout << ans << "\n";

        cout << "~"
             << "\n";
    }
    return 0;
}

// } Driver Code Ends
```

Time Complexity: O(m+n)
Space Complexity: O(1)

Question 3:
Longest consequtive 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.

Code:

```
#include <bits/stdc++.h>

using namespace std;
```

```
class Solution {
  public:
integers.
    int findLongestConseqSubseq(vector<int>& arr) {
        unordered_set<int> set(arr.begin(),arr.end());
        int n=arr.size();
        int ans=0;
        for(int i=0;i<n;i++){
            if(set.find(arr[i]-1)==set.end()){
                int j=arr[i];
                while(set.find(j)!=set.end()){
                    j++;
                }
                ans=max(ans,j-arr[i]);
        }

    }
    return ans;
    }
};
\int main() {
    int t;
    cin >> t;
    cin.ignore();
    while (t--) {
        vector<int> arr;
        string input;

        // Read first array
        getline(cin, input);
        stringstream ss(input);
        int number;
        while (ss >> number) {
            arr.push_back(number);
        }

        Solution ob;
        int res = ob.findLongestConseqSubseq(arr);

        cout << res << endl << "~" << endl;
    }
    return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(n)

Question 4: Longest palindrome in a string
Given a string S, find the longest palindromic substring in S. Substring of string S: S[i . . . . j] where $0 \leq i \leq j < len(S)$. Palindrome string: A string which reads the same backwards. More formally, S is palindrome if reverse(S) = S. Incase of conflict, return the substring which occurs first (with the least starting index).

Code:

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

class Solution{
public:
   string longestPalindrome(string s){
   if (n == 0) return "";

   int start = 0, maxLen = 1;
   for (int i = 0; i < n; i++) {
      for (int j = 0; j <= 1; j++) {
         int low = i;
         int hi = i + j;
         while (low >= 0 && hi < n && s[low] == s[hi]){
            int currLen = hi - low + 1;
            if (currLen > maxLen) {
               start = low;
               maxLen = currLen;
            }
            low--;
            hi++;
         }
      }
   }
   return s.substr(start, maxLen);
   }
};

int main(){
   int t;
   cin>>t;
```

```
   while(t--){
      string S;
      cin>>S;
      Solution ob;
      cout<<ob.longestPalindrome(S)<<endl;

cout << "~" << "\n";
}
   return 0;
}
```

Time Complexity: O(N^2)
Space Complexity: O(1)

Question 5: Rat in a Maze

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.

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

string direction = "DLRU";
int dr[4] = { 1, 0, 0, -1 };
int dc[4] = { 0, -1, 1, 0 };

bool isValid(int row, int col, int n, vector<vector<int> >& maze)
{
   return row >= 0 && col >= 0 && row < n && col < n
         && maze[row][col];
}

void findPath(int row, int col, vector<vector<int> >& maze,
            int n, vector<string>& ans,
            string& currentPath)
{
```

```
    if (row == n - 1 && col == n - 1) {
        ans.push_back(currentPath);
        return;
    }
    maze[row][col] = 0;

    for (int i = 0; i < 4; i++) {
        int nextrow = row + dr[i];
        int nextcol = col + dc[i];
        if (isValid(nextrow, nextcol, n, maze)) {
            currentPath += direction[i];
            findPath(nextrow, nextcol, maze, n, ans,
                    currentPath);
            currentPath.pop_back();
        }
    }
    maze[row][col] = 1;
}

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

    int n = maze.size();
    vector<string> result;
    string currentPath = "";

    if (maze[0][0] != 0 && maze[n - 1][n - 1] != 0) {
        findPath(0, 0, maze, n, result, currentPath);
    }

    if (result.size() == 0)
        cout << -1;
    else
        for (int i = 0; i < result.size(); i++)
            cout << result[i] << " ";
    cout << endl;

    return 0;
}
```