**DSA DAY-3**

1. **anagram program**

**#include <algorithm>**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**const int MAX\_CHAR = 26;**

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

**vector<int> freq(MAX\_CHAR, 0);**

**for(char ch: s1)**

**freq[ch - 'a']++;**

**for(char ch: s2)**

**freq[ch - 'a']--;**

**for (int count : freq) {**

**if (count != 0)**

**return false;**

**}**

**return true;**

**}**

**int main() {**

**string s1 = "geeks";**

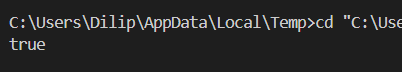
**string s2 = "kseeg";**

**cout << (areAnagrams(s1, s2) ? "true" : "false") << endl;**

**return 0;**

**}**

**OUTPUT:**

****

**TC-O(N+M)**

**2)** **row with max 1s'  
  
#include <bits/stdc++.h>**

**using namespace std;**

**int rowWithMax1s(vector<vector<bool>>& mat) {**

**int maxRow = -1, row = 0;**

**int R = mat.size();**

**int C = mat[0].size();**

**int col = C - 1;**

**while (row < R && col >= 0) {**

**if (mat[row][col] == 0) {**

**row += 1;**

**}**

**else {**

**maxRow = row;**

**col -= 1;**

**}**

**}**

**return maxRow;**

**}**

**int main() {**

**vector<vector<bool>> mat = { { 0, 0, 0, 1 },**

**{ 0, 1, 1, 1 },**

**{ 1, 1, 1, 1 },**

**{ 0, 0, 0, 0 } };**

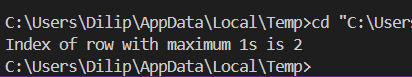
**cout << "Index of row with maximum 1s is "**

**<< rowWithMax1s(mat);**

**return 0;**

**}**

**OUTPUT:**

**  
  
TC-O(N+M)**

**3)** **Longest consequtive subsequence**

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

**using namespace std;**

**int findLongestConseqSubseq(int arr[], int n)**

**{**

**unordered\_set<int> S;**

**int ans = 0;**

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

**S.insert(arr[i]);**

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

**if (S.find(arr[i] - 1) == S.end()) {**

**int j = arr[i];**

**while (S.find(j) != S.end())**

**j++;**

**ans = max(ans, j - arr[i]);**

**}**

**}**

**return ans;**

**}**

**int main()**

**{**

**int arr[] = { 1, 9, 3, 10, 4, 20, 2 };**

**int n = sizeof arr / sizeof arr[0];**

**cout << "Length of the Longest contiguous subsequence "**

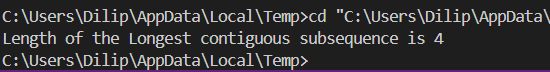
**"is "**

**<< findLongestConseqSubseq(arr, n);**

**return 0;**

**}**

**OUTPUT:**

**  
  
TC-O(N)**

**4)** **longest palindrome in a string**

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

**using namespace std;**

**string longestPalSubstr(string &s) {**

**int n = s.length();**

**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() {**

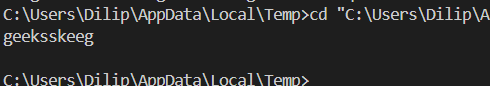
**string s = "forgeeksskeegfor";**

**cout << longestPalSubstr(s) << endl;**

**return 0;**

**}**

**OUTPUT:**

**  
  
TC-O(N^2)**

**5)** **rat in a maze problem**

**#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**

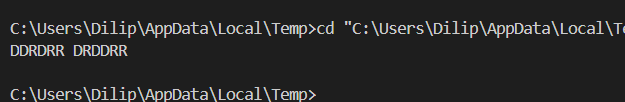
**for (int i = 0; i < result.size(); i++)**

**cout << result[i] << " ";**

**cout << endl;**

**return 0;**

**}**

**OUTPUT:  
  
  
TC-O(3\*(M\*N))**