**Complex Problem Assignment**

**Student Name: Akshit Boparai UID: 22BCS14939**

**Branch: CSE Section: FL\_IOT-601/A**

**Semester: 6th Subject Code: 22CSP-351**

**Subject: Advance Programming Lab**

**Problem 1: Set Matrix Zeroes**

Given an m x n matrix, if an element is 0, set its entire row and column to 0. The modification must be done in place without using additional storage for another matrix.

Example 1:

Input: matrix = [ [1, 1, 1], [1, 0, 1], [1, 1, 1] ] Output: [ [1, 0, 1], [0, 0, 0], [1, 0, 1] ] Explanation: The element at position (1,1) is 0. Therefore, the entire row 1 and column 1 are set to 0.

Example 2:

Input: matrix = [ [0, 1, 2, 0], [3, 4, 5, 2], [1, 3, 1, 5] ] Output: [ [0, 0, 0, 0], [0, 4, 5, 0], [0, 3, 1, 0] ] Explanation: The zeros in the first row (positions (0,0) and (0,3)) cause the entire first row and their corresponding columns to be set to 0.

**Code:**

#include <iostream>

#include <vector>

using namespace std;

void setZeroes(vector<vector<int>>& matrix) {

int m = matrix.size();

int n = matrix[0].size();

bool firstRowZero = false, firstColZero = false;

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

if (matrix[0][j] == 0) {

firstRowZero = true;

break;

}

}

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

if (matrix[i][0] == 0) {

firstColZero = true;

break;

}

}

for (int i = 1; i < m; i++) {

for (int j = 1; j < n; j++) {

if (matrix[i][j] == 0) {

matrix[i][0] = 0;

matrix[0][j] = 0;

}

}

}

for (int i = 1; i < m; i++) {

for (int j = 1; j < n; j++) {

if (matrix[i][0] == 0 || matrix[0][j] == 0) {

matrix[i][j] = 0;

}

}

}

if (firstRowZero) {

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

matrix[0][j] = 0;

}

}

if (firstColZero) {

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

matrix[i][0] = 0;

}

}

}

int main() {

int m, n;

cout << "Enter number of rows and columns: ";

cin >> m >> n;

vector<vector<int>> matrix(m, vector<int>(n));

cout << "Enter matrix elements:\n";

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

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

cin >> matrix[i][j];

setZeroes(matrix);

cout << "Modified matrix:\n";

for (const auto& row : matrix) {

for (int val : row)

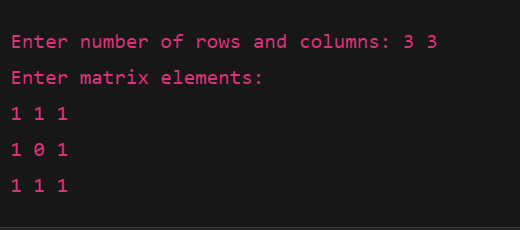
cout << val << " ";

cout << "\n";

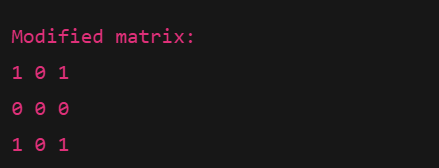
}

return 0;

}

INPUT :  


OUTPUT:



**Problem 2: Longest Substring Without Repeating Characters**

Given a string s, find the length of the longest substring that does not contain any repeating characters.

Example 1: Input: s = "abcabcbb" Output: 3 Explanation: The longest substring without repeating characters is "abc", which has a length of 3.

Example 2: Input: s = "bbbbb" Output: 1 Explanation: All characters in the string are the same, so the longest substring with unique characters is "b", with a length of 1.

Code:

#include <iostream>

#include <string>

#include <unordered\_set>

using namespace std;

int lengthOfLongestSubstring(string s) {

unordered\_set<char> chars;

int left = 0, right = 0, maxLength = 0;

while (right < s.size()) {

if (chars.find(s[right]) == chars.end()) {

chars.insert(s[right]);

maxLength = max(maxLength, right - left + 1);

right++;

} else {

chars.erase(s[left]);

left++;

}

}

return maxLength;

}

int main() {

string s;

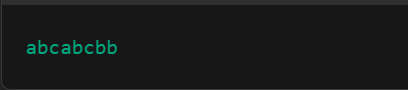
cin >> s;

cout << lengthOfLongestSubstring(s) << endl;

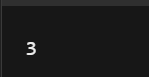
return 0;

}

**Input:**

****

**Output:**

****

**Problem 3: Reverse Linked List II**

Given the head of a singly linked list and two integers left and right, reverse the nodes of the list from position left to right, and return the modified list.

Example 1: Input: Linked list: [1, 2, 3, 4, 5]; left = 2; right = 4 Output: [1, 4, 3, 2, 5] Explanation: The sublist from position 2 to 4 ([2, 3, 4]) is reversed to become [4, 3, 2], while the rest of the list remains unchanged.

Example 2: Input: Linked list: [1, 2, 3, 4, 5]; left = 1; right = 5 Output: [5, 4, 3, 2, 1] Explanation: The entire list is reversed because the reversal starts at the first node and ends at the last node.

Code:

#include <iostream>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

ListNode(int x) : val(x), next(nullptr) {}

};

ListNode\* reverseBetween(ListNode\* head, int left, int right) {

if (!head || left == right) return head;

ListNode dummy(0);

dummy.next = head;

ListNode\* prev = &dummy;

for (int i = 1; i < left; i++) prev = prev->next;

ListNode\* curr = prev->next;

for (int i = 0; i < right - left; i++) {

ListNode\* temp = curr->next;

curr->next = temp->next;

temp->next = prev->next;

prev->next = temp;

}

return dummy.next;

}

void printList(ListNode\* head) {

while (head) {

cout << head->val;

if (head->next) cout << " ";

head = head->next;

}

cout << endl;

}

int main() {

int n, val, left, right;

cin >> n;

ListNode\* head = nullptr;

ListNode\* tail = nullptr;

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

cin >> val;

ListNode\* node = new ListNode(val);

if (!head) {

head = node;

tail = node;

} else {

tail->next = node;

tail = node;

}

}

cin >> left >> right;

head = reverseBetween(head, left, right);

printList(head);

return 0;

}

**Input:**

****

**Output:**

****

**Problem 4: Detect a Cycle in a Linked List**

Given the head of a linked list, determine whether the linked list contains a cycle. A cycle occurs if a node's next pointer points to a previous node in the list.

Example 1: Input: Linked list: [3, 2, 0, -4] with the tail node (-4) pointing to the node with value 2. Output: true Explanation: The tail node connects back to an earlier node, forming a cycle.

Example 2: Input: Linked list: [1, 2] with no cycle (each node points to null at the end). Output: false Explanation: There is no cycle since no node points back to a previous node.

Code:

#include <iostream>

#include <unordered\_map>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

ListNode(int x) : val(x), next(nullptr) {}

};

bool hasCycle(ListNode\* head) {

ListNode\* slow = head;

ListNode\* fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) return true;

}

return false;

}

int main() {

int n, pos;

cin >> n;

ListNode\* head = nullptr;

ListNode\* tail = nullptr;

ListNode\* cycleNode = nullptr;

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

int val;

cin >> val;

ListNode\* node = new ListNode(val);

if (!head) {

head = node;

tail = node;

} else {

tail->next = node;

tail = node;

}

if (i == pos) cycleNode = node;

}

cin >> pos;

if (pos != -1) {

ListNode\* curr = head;

for (int i = 0; i < pos; i++) curr = curr->next;

tail->next = curr;

}

cout << (hasCycle(head) ? "true" : "false") << endl;

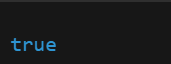
return 0;

}

**Input:**

****

**Output:**

****

**Problem 5: The Skyline Problem**

Given a list of buildings represented as [left, right, height], where each building is a rectangle, return the key points of the skyline. A key point is represented as [x, y], where x is the x-coordinate where the height changes to y.

Example 1: Input: buildings = [ [2, 9, 10], [3, 7, 15], [5, 12, 12], [15, 20, 10], [19, 24, 8] ] Output: [ [2, 10], [3, 15], [7, 12], [12, 0], [15, 10], [20, 8], [24, 0] ] Explanation: The skyline starts at x = 2 with height 10, rises to height 15 at x = 3, drops to 12 at x = 7, then to 0 at x = 12, rises again at x = 15, changes at x = 20, and finally drops to 0 at x = 24.

Example 2: Input: buildings = [ [0, 2, 3], [2, 5, 3] ] Output: [ [0, 3], [5, 0] ] Explanation: Both buildings have the same height of 3. The skyline begins at x = 0 with height 3 and ends at x = 5 when the building ends, resulting in a drop to 0.

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <set>

using namespace std;

vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

vector<pair<int, int>> events;

for (auto& b : buildings) {

events.push\_back({b[0], -b[2]});

events.push\_back({b[1], b[2]});

}

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

multiset<int> heights = {0};

vector<vector<int>> result;

int prev = 0;

for (auto& e : events) {

int x = e.first, h = e.second;

if (h < 0) heights.insert(-h);

else heights.erase(heights.find(h));

int curr = \*heights.rbegin();

if (curr != prev) {

result.push\_back({x, curr});

prev = curr;

}

}

return result;

}

int main() {

int n;

cin >> n;

vector<vector<int>> buildings(n, vector<int>(3));

for (int i = 0; i < n; i++) cin >> buildings[i][0] >> buildings[i][1] >> buildings[i][2];

vector<vector<int>> skyline = getSkyline(buildings);

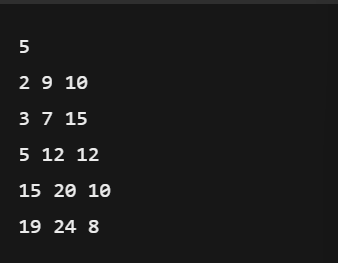
for (auto& point : skyline) cout << "[" << point[0] << ", " << point[1] << "] ";

cout << endl;

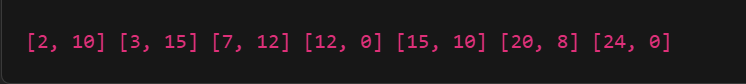
return 0;

}

**Input:**

****

**Output:**

****