DAY 8

NAME: -Shivam Yadav

UID: -22BCS15259

DATE: -28/12/2024

Section: - 620-A

QUESTION 1: -

Divisor Game

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

bool canWin(int n) {
    vector<bool> dp(n + 1, false);
    dp[1] = false;
    for (int i = 2; i <= n; ++i) {
        for (int x = 1; x < i; ++x) {
            if (i % x == 0 && !dp[i - x]) {
                dp[i] = true;
                break;
            }
        }
```

```
    }
    return dp[n];
}
int main() {
    int n;
    cout << "Enter the initial number n: ";
    cin >> n;
    if (canWin(n)) {
        cout << "Alice wins!" << endl;
    } else {
        cout << "Bob wins!" << endl;
    }
    return 0;
}
```

OUTPUT: -

```
Enter the initial number n: 5
Bob wins!
```

QUESTION 2: -

Maximum Repeating Substring

```
#include <iostream>
#include <string>
```

```
#include <vector>
using namespace std;

int maxKRepetition(string sequence, string word) {
    int sequenceLen = sequence.size();
    int wordLen = word.size();
    if (wordLen > sequenceLen) {
        return 0;
    }
    int maxK = 0;
    for (int k = 1; k * wordLen <= sequenceLen; ++k) {
        string repeatedWord = word;
        for (int i = 1; i < k; ++i) {
            repeatedWord += word;
        }
        if (sequence.find(repeatedWord) != string::npos) {
            maxK = k;
        } else {
            break;
        }
    }
    return maxK;
```

```
}
int main() {
    string sequence, word;
    cout << "Enter sequence: ";
    cin >> sequence;
    cout << "Enter word: ";
    cin >> word;
    int result = maxKRepetition(sequence, word);
    cout << "The maximum k-repeating value is: " << result << endl;
    return 0;
}
```

OUTPUT: -

```
Enter sequence: abcabcabcabc
Enter word: abc
The maximum k-repeating value is: 4
```

QUESTION 3: -

Pascal's Triangle II

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> generateRow(int rowIndex) {
```

```
    vector<int> row(rowIndex + 1, 1);
    for (int i = 1; i < rowIndex; ++i) {
        for (int j = i; j > 0; --j) {
            row[j] = row[j] + row[j - 1];
        }
    }

    return row;
}
int main() {
    int rowIndex;
    cout << "Enter the row index: ";
    cin >> rowIndex;
    vector<int> row = generateRow(rowIndex);
    cout << "The " << rowIndex << "-th row of Pascal's triangle
is: ";
    for (int num : row) {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}
```

OUTPUT: -

```
Enter the row index: 5
The 5-th row of Pascal's triangle is: 1 5 10 10 5 1
```

QUESTION 4: -

Climbing Stairs

```
#include <iostream>
#include <vector>
using namespace std;
int climbStairs(int n) {
    if (n == 1) return 1;
    vector<int> dp(n + 1, 0);
    dp[0] = 1;
    dp[1] = 1;
    for (int i = 2; i <= n; ++i) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    return dp[n];
}
int main() {
    int n;
    cout << "Enter the number of steps: ";
    cin >> n;
```

```
    int result = climbStairs(n);
    cout << "Number of distinct ways to climb to the top: " << result << endl;
    return 0;
}
```

OUTPUT: -

```
Enter the number of steps: 4
Number of distinct ways to climb to the top: 5
```

QUESTION 5: -

Best Time to Buy and Sell Stock

```
#include <iostream>
#include <vector>
#include <climits> // For INT_MAX
using namespace std;

int maxProfit(vector<int>& prices) {
    int n = prices.size();
    if (n <= 1) return 0;
    int minPrice = INT_MAX;
    int maxProfit = 0;
    for (int i = 0; i < n; ++i) {
        minPrice = min(minPrice, prices[i]);
```

```
        int profit = prices[i] - minPrice;
        maxProfit = max(maxProfit, profit);
    }
    return maxProfit;
}
int main() {
    vector<int> prices;
    int n, price;
    cout << "Enter the number of days: ";
    cin >> n;
    cout << "Enter the prices for each day: ";
    for (int i = 0; i < n; ++i) {
        cin >> price;
        prices.push_back(price);
    }
    int result = maxProfit(prices);
    cout << "Maximum profit: " << result << endl;
    return 0;
}
```

OUTPUT: -

```
Enter the number of days: 6
Enter the prices for each day: 7 1 5 3 6 4
Maximum profit: 5
```

QUESTION 6: -

Longest Palindromic Substring

```
#include <iostream>

#include <vector>

#include <string>

using namespace std;

string longestPalindrome(string s) {

    int n = s.length();

    if (n == 0) return "";

    vector<vector<bool>> dp(n, vector<bool>(n, false));

    int start = 0; g

    int maxLength = 1;

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

        dp[i][i] = true;

    }

    for (int i = 0; i < n - 1; i++) {

        if (s[i] == s[i + 1]) {

            dp[i][i + 1] = true;

            start = i;
```

```
            maxLength = 2;
        }
    }
    for (int length = 3; length <= n; length++) {
        for (int i = 0; i < n - length + 1; i++) {
            int j = i + length - 1;
            if (s[i] == s[j] && dp[i + 1][j - 1]) {
                dp[i][j] = true;
                if (length > maxLength) {
                    start = i;
                    maxLength = length;
                }
            }
        }
    }
    return s.substr(start, maxLength);
}
int main() {
    string s;
    cout << "Enter a string: ";
    cin >> s;
    string result = longestPalindrome(s);
```

```
    cout << "Longest palindromic substring: " << result << endl;
    return 0;
}
```

OUTPUT: -

```
Enter a string: babd
Longest palindromic substring: bab
```

QUESTION 7: -

Minimum Path Sum

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int minPathSum(vector<vector<int>>& grid) {
    int m = grid.size();
    int n = grid[0].size();
    vector<vector<int>> dp(m, vector<int>(n, 0));
    dp[0][0] = grid[0][0];
    for (int j = 1; j < n; j++) {
        dp[0][j] = dp[0][j-1] + grid[0][j];
    }
    for (int i = 1; i < m; i++) {
        dp[i][0] = dp[i-1][0] + grid[i][0];
```

```
    }
    for (int i = 1; i < m; i++) {
        for (int j = 1; j < n; j++) {
            dp[i][j] = grid[i][j] + min(dp[i-1][j], dp[i][j-1]);
        }
    }
    return dp[m-1][n-1];
}
int main() {
    int m, n;
    cout << "Enter the number of rows (m): ";
    cin >> m;
    cout << "Enter the number of columns (n): ";
    cin >> n;
    vector<vector<int>> grid(m, vector<int>(n));
    cout << "Enter the grid values:\n";
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> grid[i][j];
        }
    }
    int result = minPathSum(grid);
```

```
    cout << "The minimum path sum is: " << result << endl;
    return 0;
}
```

OUTPUT: -

```
Enter the grid values:
1 3 1
1 5 1
4 2 1
The minimum path sum is: 7
```

QUESTION 8: -

Maximal Rectangle

```
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>
using namespace std;
int largestRectangleInHistogram(const vector<int>& heights) {
    stack<int> stk;
    int maxArea = 0;
    int n = heights.size();
    for (int i = 0; i <= n; i++) {
        while (!stk.empty() && (i == n || heights[stk.top()] >
heights[i])) {
            int height = heights[stk.top()];
```

```
            stk.pop();
            int width = stk.empty() ? i : i - stk.top() - 1;
            maxArea = max(maxArea, height * width);
        }
        stk.push(i);
    }
    return maxArea;
}
int maximalRectangle(vector<vector<int>>& matrix) {
    if (matrix.empty() || matrix[0].empty()) return 0;
    int rows = matrix.size();
    int cols = matrix[0].size();
    vector<int> heights(cols, 0);
    int maxArea = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            heights[j] = (matrix[i][j] == 0) ? 0 : heights[j] + 1;
        }
        maxArea = max(maxArea,
largestRectangleInHistogram(heights));
    }
    return maxArea;
```

```
}
int main() {
    int rows, cols;
    cout << "Enter the number of rows: ";
    cin >> rows;
    cout << "Enter the number of columns: ";
    cin >> cols;
    vector<vector<int>> matrix(rows, vector<int>(cols));
    cout << "Enter the matrix values (0's and 1's):\n";
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix[i][j];
        }
    }
    int result = maximalRectangle(matrix);
    cout << "The area of the largest rectangle containing only
1's is: " << result << endl;
    return 0;
}
```

OUTPUT: -

```
Enter the number of rows: 4
Enter the number of columns: 5
Enter the matrix values (0's and 1's):
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
The area of the largest rectangle containing only 1's is: 6
```

QUESTION 9: -

Number of Digit One

```
#include <iostream>

#include <cmath>

using namespace std;

int countDigitOne(int n) {

    int count = 0;

    long long factor = 1;


    while (factor <= n) {

        int lowerNumbers = n - (n / factor) * factor;

        int currentDigit = (n / factor) % 10;

        int higherNumbers = n / (factor * 10);

        if (currentDigit == 0) {

            count += higherNumbers * factor;

        } else if (currentDigit == 1) {

            count += higherNumbers * factor + lowerNumbers + 1;

        } else {
```

```
            count += (higherNumbers + 1) * factor;
        }
        factor *= 10;
    }
    return count;
}
int main() {
    int n;
    cout << "Enter an integer n: ";
    cin >> n;
    int result = countDigitOne(n);
    cout << "Total number of digit '1' appearing in all numbers from 0 to " << n << " is: " << result << endl;
    return 0;
}
```

OUTPUT: -

```
Enter an integer n: 13
Total number of digit '1' appearing in all numbers from 0 to 13 is: 6
```

QUESTION 10: -

Cherry Pickup

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```
using namespace std;
int cherryPickup(vector<vector<int>>& grid) {
    int n = grid.size();
    if (grid[0][0] == -1 || grid[n-1][n-1] == -1) return 0;
    vector<vector<vector<int>>> dp(n, vector<vector<int>>(n,
vector<int>(n, -1)));
    dp[0][0][0] = grid[0][0];
    for (int t = 1; t < 2 * n - 1; ++t) {
        for (int r1 = max(0, t - (n - 1)); r1 <= min(t, n - 1); ++r1) {
            for (int r2 = max(0, t - (n - 1)); r2 <= min(t, n - 1); ++r2) {
                int c1 = t - r1, c2 = t - r2;
                if (grid[r1][c1] == -1 || grid[r2][c2] == -1) continue;
                int current_cherries = grid[r1][c1] + (r1 != r2 || c1 !=
c2 ? grid[r2][c2] : 0);
                int best_prev = -1;
                for (int dr1 = -1; dr1 <= 0; ++dr1) {
                    for (int dr2 = -1; dr2 <= 0; ++dr2) {
                        int prev_r1 = r1 + dr1, prev_r2 = r2 + dr2;
                        if (prev_r1 >= 0 && prev_r1 < n && prev_r2 >= 0
&& prev_r2 < n) {
                            best_prev = max(best_prev, dp[prev_r1][c1 -
1][prev_r2]);
                        }
```

```
                }
            }
            dp[r1][c1][r2] = best_prev + current_cherries;
        }
    }
}
return dp[n - 1][n - 1][n-1];
}
int main() {
    int n;
    cout << "Enter the grid size: ";
    cin >> n;
    vector<vector<int>> grid(n, vector<int>(n));
    cout << "Enter the grid values (0 for empty, 1 for cherry, -1
for thorn):\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> grid[i][j];
        }
    }
    int result = cherryPickup(grid);
    cout << "Maximum cherries collected: " << result << endl;
```

```
    return 0;
}
```

OUTPUT: -

```
Enter the grid size: 3
Enter the grid values (0 for empty, 1 for cherry, -1 for thorn):
0 1 0
0 -1 0
0 1 0
```