DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name: Ansh Jain **UID: 22BCS15216**
Branch: BE-CSE::CS201 **Section/Group: 22BCS_FL_IOT-603/B**
Semester: 5th

➢ DAY-8 [27-12-2024]

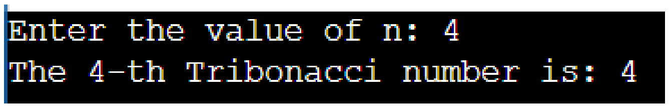
1. N-th Tribonacci Number *(Very Easy)*

The Tribonacci sequence Tn is defined as follows:
T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.
Given n, return the value of Tn.

Implementation/Code:

```
#include <iostream>
using namespace std;
class Solution {
public:
    int tribonacci(int n) {
        if (n == 0) return 0;
        if (n == 1 || n == 2) return 1;
        int t0 = 0, t1 = 1, t2 = 1;
        int t3;
        for (int i = 3; i <= n; ++i) {
            t3 = t0 + t1 + t2;
            t0 = t1;
            t1 = t2;
            t2 = t3;
        }
        return t3;
    }
};
int main() {
    int n;
    cout << "Enter the value of n: ";
    cin >> n;
    Solution solution;
    int result = solution.tribonacci(n);
    cout << "The " << n << "-th Tribonacci number is: " << result << endl;
    return 0;
}
```

Output:



```
Enter the value of n: 4
The 4-th Tribonacci number is: 4
```

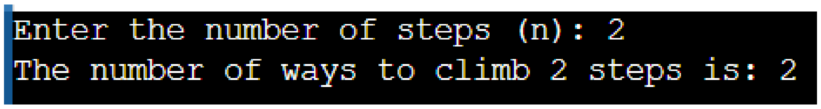
2. Climbing Stairs *(Easy)*

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Implementation/Code:

```
#include <iostream>
using namespace std;
class Solution {
public:
    int climbStairs(int n) {
        if (n == 1) return 1;
        int prev1 = 1, prev2 = 2;
        for (int i = 3; i <= n; ++i) {
            int curr = prev1 + prev2;
            prev1 = prev2;
            prev2 = curr;
        }
        return prev2;
    }
};
int main() {
    int n;
    cout << "Enter the number of steps (n): ";
    cin >> n;
    Solution solution;
    int result = solution.climbStairs(n);
    cout << "The number of ways to climb " << n << " steps is: " << result << endl;
    return 0;
}
```

Output:



```
Enter the number of steps (n): 2
The number of ways to climb 2 steps is: 2
```

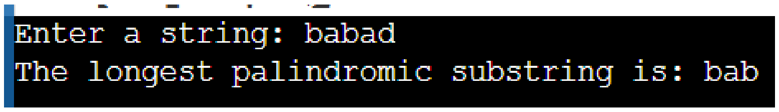
3. Longest Palindromic Substring ***(Medium)***

Given a string s, return the longest palindromic substring in s.

Implementation/Code:

```
#include <iostream>
#include <string>
using namespace std;
class Solution {
public:
    string longestPalindrome(string s) {
        int start = 0, maxLength = 0;
        for (int i = 0; i < s.size(); ++i) {
            expandAroundCenter(s, i, i, start, maxLength);
            expandAroundCenter(s, i, i + 1, start, maxLength);
        }
        return s.substr(start, maxLength);
    }
private:
    void expandAroundCenter(const string& s, int left, int right, int& start, int&
maxLength) {
        while (left >= 0 && right < s.size() && s[left] == s[right]) {
            int length = right - left + 1;
            if (length > maxLength) {
                maxLength = length;
                start = left;
            }
            --left;
            ++right;
        }
    }
};
int main() {
    string s;
    cout << "Enter a string: ";
    cin >> s;
    Solution solution;
    string result = solution.longestPalindrome(s);
    cout << "The longest palindromic substring is: " << result << endl;
    return 0;
}
```

Output:



```
Enter a string: babad
The longest palindromic substring is: bab
```

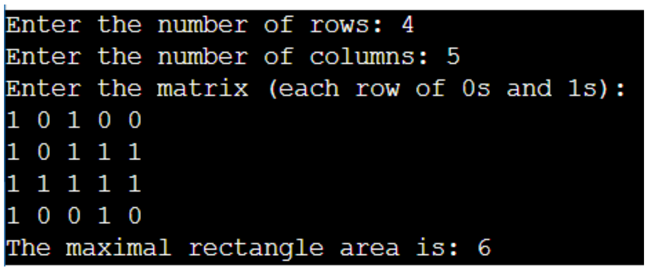
4. Maximal Rectangle *(Hard)*

Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

Implementation/Code:

```
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>
using namespace std;
class Solution {
public:
    int maximalRectangle(vector<vector<char>>& 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] == '1' ? heights[j] + 1 : 0;
            }
            maxArea = max(maxArea, largestRectangleArea(heights));
        }
        return maxArea;
    }
    int largestRectangleArea(vector<int>& heights) {
        stack<int> s;
        int maxArea = 0;
        heights.push_back(0);
        for (int i = 0; i < heights.size(); i++) {
            while (!s.empty() && heights[i] < heights[s.top()]) {
                int h = heights[s.top()];
                s.pop();
                int width = s.empty() ? i : i - s.top() - 1;
                maxArea = max(maxArea, h * width);
            }
            s.push(i);
        }
        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<char>> matrix(rows, vector<char>(cols));
    cout << "Enter the matrix (each row of 0s and 1s):" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix[i][j];
        }
    }
    Solution solution;
    int result = solution.maximalRectangle(matrix);
    cout << "The maximal rectangle area is: " << result << endl;
    return 0;
}
```

Output:



```
Enter the number of rows: 4
Enter the number of columns: 5
Enter the matrix (each row of 0s and 1s):
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
The maximal rectangle area is: 6
```

5. Cherry Pickup *(Very Hard)*

You are given an n x n grid representing a field of cherries, each cell is one of three possible integers.

0 means the cell is empty, so you can pass through,

1 means the cell contains a cherry that you can pick up and pass through, or

-1 means the cell contains a thorn that blocks your way.

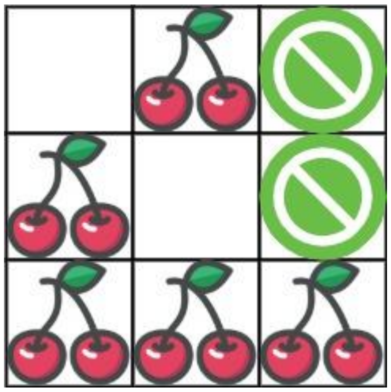
Return the maximum number of cherries you can collect by following the rules below:

Starting at the position (0, 0) and reaching (n - 1, n - 1) by moving right or down through valid path cells (cells with value 0 or 1).

After reaching (n - 1, n - 1), returning to (0, 0) by moving left or up through valid path cells.

When passing through a path cell containing a cherry, you pick it up, and the cell becomes an empty cell 0.

If there is no valid path between (0, 0) and (n - 1, n - 1), then no cherries can be collected.

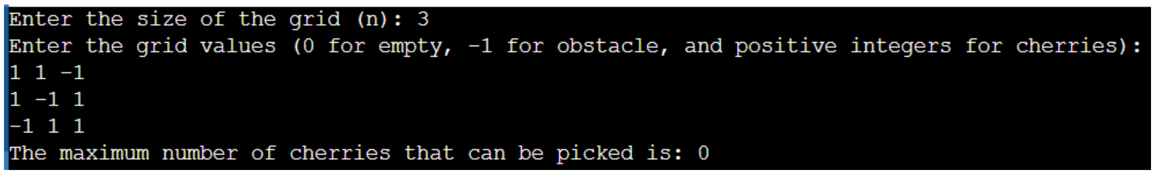


Implementation/Code:

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

```
class Solution {
public:
    int cherryPickup(vector<vector<int>>& grid) {
        int n = grid.size();
        vector<vector<vector<int>>> dp(2 * n - 1, vector<vector<int>>(n, vector<int>(n, -
1)));
        dp[0][0][0] = grid[0][0] + grid[0][0];
        for (int step = 1; step < 2 * n - 1; ++step) {
            for (int x1 = max(0, step - n + 1); x1 < n && x1 <= step; ++x1) {
                for (int x2 = max(0, step - n + 1); x2 < n && x2 <= step; ++x2) {
                    int y1 = step - x1;
                    int y2 = step - x2;
                    if (grid[x1][y1] == -1 || grid[x2][y2] == -1)
                        continue;
                    int cherries = grid[x1][y1] + grid[x2][y2];
                    if (x1 != x2)
                        cherries -= grid[x1][y1];
                    int maxPrev = -1;
                    for (int dx1 = -1; dx1 <= 0; dx1++) {
                        for (int dx2 = -1; dx2 <= 0; dx2++) {
                            int prevX1 = x1 + dx1;
                            int prevX2 = x2 + dx2;
                            if (prevX1 >= 0 && prevX2 >= 0 && prevX1 < n && prevX2 < n) {
                                maxPrev = max(maxPrev, dp[step - 1][prevX1][prevX2]);
                            }
                        }
                    }
                    if (maxPrev != -1) {
                        dp[step][x1][x2] = max(dp[step][x1][x2], maxPrev + cherries);
                    }
                }
            }
        }
        return dp[2 * n - 2][n - 1][n - 1] == -1 ? 0 : dp[2 * n - 2][n - 1][n - 1];
    }
};
int main() {
    int n;
    cout << "Enter the size of the grid (n): ";
    cin >> n;
    vector<vector<int>> grid(n, vector<int>(n));
    cout << "Enter the grid values (0 for empty, -1 for obstacle, and positive integers for cherries):" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> grid[i][j];
        }
    }
    Solution solution;
    int result = solution.cherryPickup(grid);
    cout << "The maximum number of cherries that can be picked is: " << result << endl;
    return 0;
}
```

Output:



```
Enter the size of the grid (n): 3
Enter the grid values (0 for empty, -1 for obstacle, and positive integers for cherries):
1 1 -1
1 -1 1
-1 1 1
The maximum number of cherries that can be picked is: 0
```