UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE)

WINTER DOMAIN CAMP

Date : 29 /12 /2024

Submitted to:

Faculty name: Er.Rajni Devi

Submitted by:

Name: Akshi Datta

UID: 22BCS15369

Section: IOT-620

DAY 8(29/12/24)

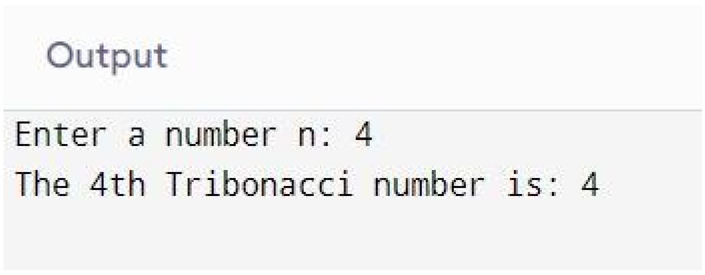
PROBLEM 1

Objective: N-th Tribonacci Number

Code:

```
#include <iostream>
using namespace std;
int tribonacci(int n) {
    if (n == 0) return 0;
    if (n == 1 || n == 2) return 1;
    int t0 = 0, t1 = 1, t2 = 1, t3;
    for (int i = 3; i <= n; ++i) {
        t3 = t0 + t1 + t2;  // Tn+3 = Tn + Tn+1 + Tn+2
        t0 = t1;  // update t0 to the next value
        t1 = t2;  // update t1 to the next value
        t2 = t3;  // update t2 to the next value
    }
    return t2;
}
int main() {
    int n;
    cout << "Enter a number n: ";
    cin >> n;
    cout << "The " << n << "th Tribonacci number is: " << tribonacci(n) << endl;
    return 0;
}
```

Output:



```
Enter a number n: 4
The 4th Tribonacci number is: 4
```

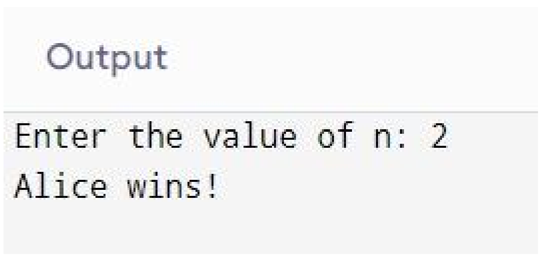
PROBLEM 2

Objective: Divisor Game

Code:

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

bool canWin(int n) {
   vector<bool> dp(n + 1, false);  // dp[i] will be true if the player can win starting with i

   // Iterate over all values from 2 to n
   for (int i = 2; i <= n; ++i) {
      // Check all possible moves
      for (int x = 1; x < i; ++x) {
         if (i % x == 0 && !dp[i - x]) {
            dp[i] = true;  // The current player can win by making the move i - x
            break;
         }
      }
   }

   return dp[n];
}
int main() {
   int n;
   cout << "Enter the value of n: ";
   cin >> n;

   if (canWin(n)) {
      cout << "Alice wins!" << endl;
   } else {
      cout << "Bob wins!" << endl;
   }

   return 0;
}
```

Output:



```
Enter the value of n: 2
Alice wins!
```

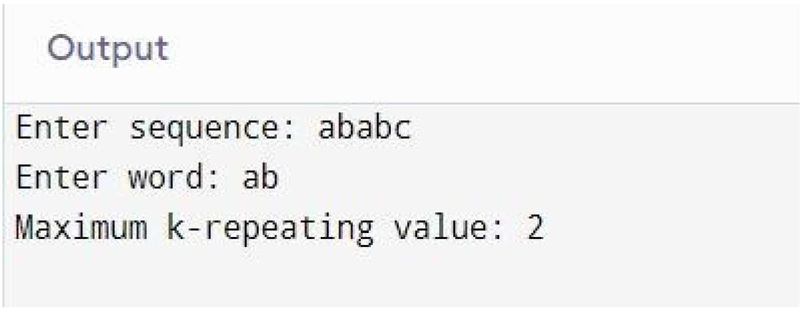
PROBLEM 3

Objective: Maximum Repeating Substring

Code:

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

int maxRepeating(string sequence, string word) {
    int k = 0;
    string temp = word;

    // Keep concatenating word to itself and check if it's a substring of sequence
    while (sequence.find(temp) != string::npos) {
        k++;
        temp += word;  // Concatenate word to temp
    }

    return k;
}

int main() {
    string sequence, word;
    cout << "Enter sequence: ";
    cin >> sequence;
    cout << "Enter word: ";
    cin >> word;

    int result = maxRepeating(sequence, word);
    cout << "Maximum k-repeating value: " << result << endl;

    return 0;
}
```

Output:



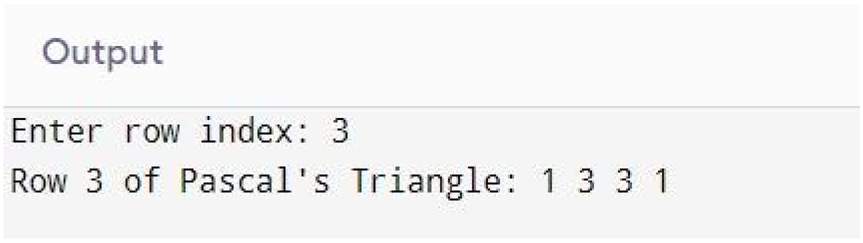
```
Enter sequence: ababc
Enter word: ab
Maximum k-repeating value: 2
```

PROBLEM 4

Objective: Pascal's Triangle II
Code:

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

vector<int> getRow(int rowIndex) {
   vector<int> row(rowIndex + 1, 1);  // Initialize the row with all 1s

   // Iterate through each index starting from 1 up to rowIndex
   for (int i = 1; i <= rowIndex / 2; ++i) {
      // Use the property of Pascal's triangle: row[i] = row[i-1] * (rowIndex - i + 1) / i
      row[i] = row[i - 1] * (rowIndex - i + 1) / i;
      row[rowIndex - i] = row[i];  // The row is symmetric, so copy the value to the other half
   }

   return row;
}

int main() {
   int rowIndex;
   cout << "Enter row index: ";
   cin >> rowIndex;

   vector<int> result = getRow(rowIndex);
   cout << "Row " << rowIndex << " of Pascal's Triangle: ";
   for (int val : result) {
      cout << val << " ";
   }
   cout << endl;

   return 0;
}
```

Output:



```
Enter row index: 3
Row 3 of Pascal's Triangle: 1 3 3 1
```

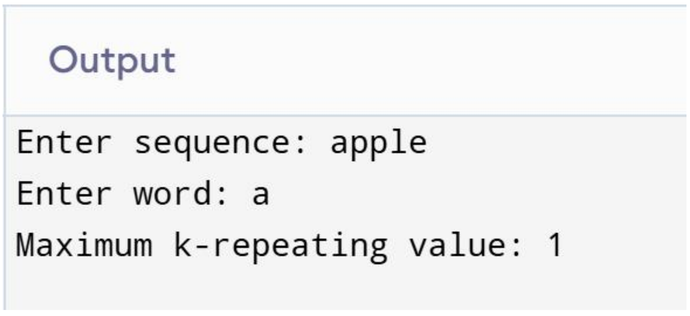
PROBLEM 5

Objective: Maximum Repeating Substring

Code:

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

int maxRepeating(string sequence, string word) {
   int k = 0;
   string temp = word;

   // Keep concatenating the word and check if it is a substring of sequence
   while (sequence.find(temp) != string::npos) {
      k++;
      temp += word;  // Concatenate word to temp
   }

   return k;
}

int main() {
   string sequence, word;
   cout << "Enter sequence: ";
   cin >> sequence;
   cout << "Enter word: ";
   cin >> word;

   int result = maxRepeating(sequence, word);
   cout << "Maximum k-repeating value: " << result << endl;

   return 0;
}
```

Output:



Output

```
Enter sequence: apple
Enter word: a
Maximum k-repeating value: 1
```

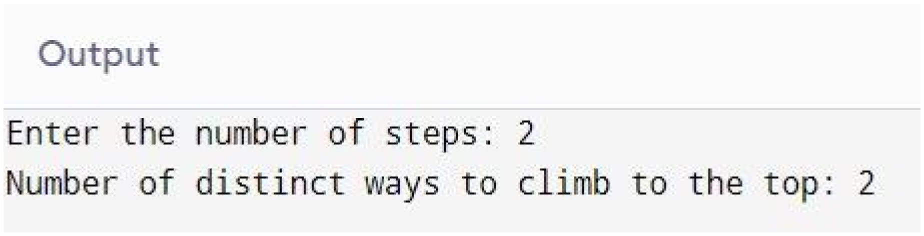
PROBLEM 6

Objective: Climbing Stairs

Code:

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

int climbStairs(int n) {
   if (n == 1) return 1;
   if (n == 2) return 2;

   int first = 1, second = 2;
   for (int i = 3; i <= n; i++) {
      int next = first + second;  // Ways to reach current step
      first = second;  // Update first to second
      second = next;   // Update second to next
   }

   return second;  // second holds the number of ways to reach the nth step
}

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: 2
Number of distinct ways to climb to the top: 2
```

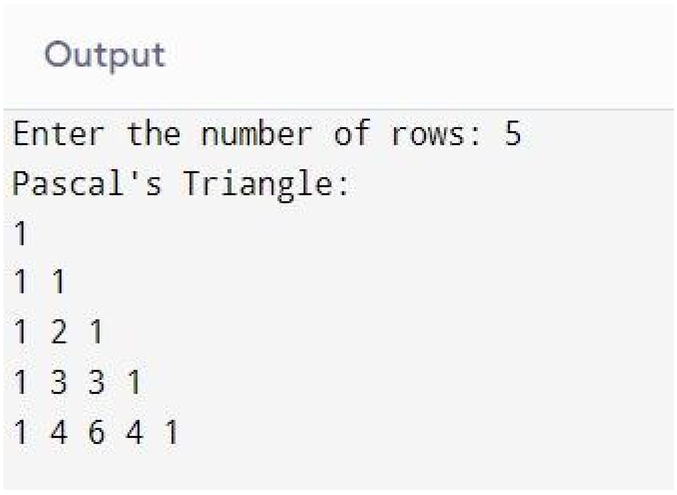
PROBLEM 7

Objective: Pascal's Triangle

Code:

```
#include <iostream>
#include <vector>
using namespace std;
vector<vector<int>> generate(int numRows) {
   vector<vector<int>> triangle;
   for (int i = 0; i < numRows; ++i) {
      vector<int> row(i + 1, 1);
      for (int j = 1; j < i; ++j) {
         row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
      }
      triangle.push_back(row);  // Add the row to the triangle
   }
   return triangle;
}
int main() {
   int numRows;
   cout << "Enter the number of rows: ";
   cin >> numRows;
   vector<vector<int>> result = generate(numRows);
   cout << "Pascal's Triangle:" << endl;
   for (const auto& row : result) {
      for (int val : row) {
         cout << val << " ";
      }
      cout << endl;
   }
   return 0;
}
```

Output:



```
Enter the number of rows: 5
Pascal's Triangle:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

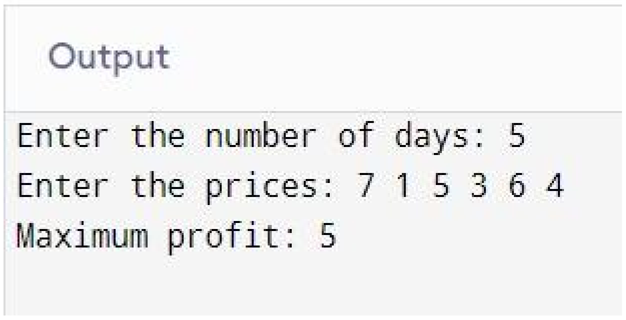
PROBLEM 8

Objective: Best Time to Buy and Sell Stock

Code:

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

int maxProfit(vector<int>& prices) {
   int min_price = prices[0];
   int max_profit = 0;
      for (int i = 1; i < prices.size(); i++) {
           int profit = prices[i] - min_price;
      max_profit = max(max_profit, profit);
      min_price = min(min_price, prices[i]);
   }
   return max_profit;
}
int main() {
   vector<int> prices;
   int n, price;
   cout << "Enter the number of days: ";
   cin >> n;
   cout << "Enter the prices: ";
   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: 5
Enter the prices: 7 1 5 3 6 4
Maximum profit: 5
```

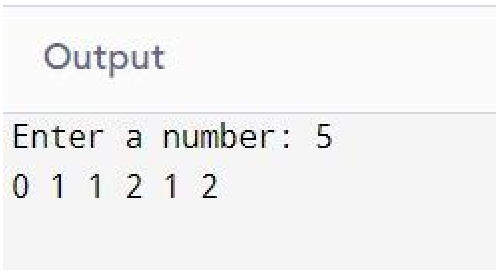
PROBLEM 9

Objective: Counting Bits

Code:

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

vector<int> countBits(int n) {
    vector<int> ans(n + 1, 0);  // Initialize the array with zeros

    for (int i = 1; i <= n; i++) {
        ans[i] = ans[i & (i - 1)] + 1;  // Use the recurrence relation
    }

    return ans;
}

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    vector<int> result = countBits(n);

    // Output the result
    for (int i = 0; i <= n; i++) {
        cout << result[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Output:



```
Enter a number: 5
0 1 1 2 1 2
```

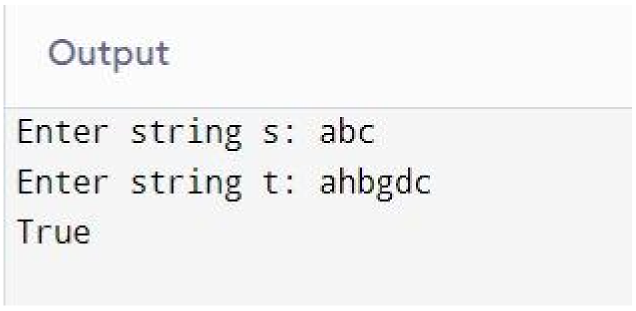
PROBLEM 10

Objective: Is Subsequence

Code:

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

bool isSubsequence(string s, string t) {
    int i = 0, j = 0;
    while (i < s.size() && j < t.size()) {
        if (s[i] == t[j]) {
            i++;
        }
        j++;
    }
    return i == s.size();
}
int main() {
    string s, t;
    cout << "Enter string s: ";
    cin >> s;
    cout << "Enter string t: ";
    cin >> t;

    if (isSubsequence(s, t)) {
        cout << "True" << endl;
    } else {
        cout << "False" << endl;
    }

    return 0;
}
```

Output:


Output

```
Enter string s: abc
Enter string t: ahbgdc
True
```

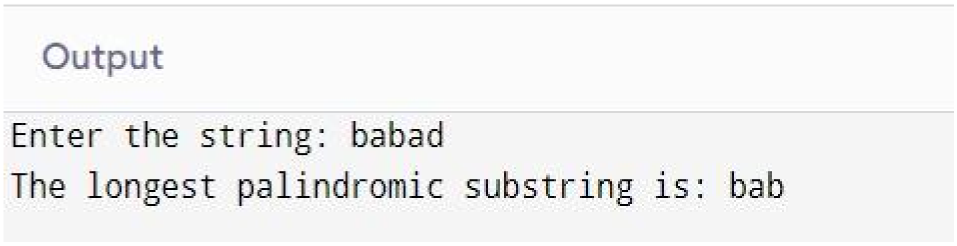
PROBLEM 11

Objective: Longest Palindromic Substring

Code:

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

string expandAroundCenter(const string &s, int left, int right) {
   while (left >= 0 && right < s.size() && s[left] == s[right]) {
      left--;
      right++;
   }
   return s.substr(left + 1, right - left - 1);
}
string longestPalindrome(string s) {
   if (s.empty()) return "";
   string longest = "";
   for (int i = 0; i < s.size(); i++) {
      string oddPalindrome = expandAroundCenter(s, i, i);
      if (oddPalindrome.size() > longest.size()) {
         longest = oddPalindrome;
      }
      string evenPalindrome = expandAroundCenter(s, i, i + 1);
      if (evenPalindrome.size() > longest.size()) {
         longest = evenPalindrome;
      }}
   return longest;
}
int main() {
   string s;
   cout << "Enter the string: ";
   cin >> s;
   string result = longestPalindrome(s);
   cout << "The longest palindromic substring is: " << result << endl;
   return 0;
}
```

Output:



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

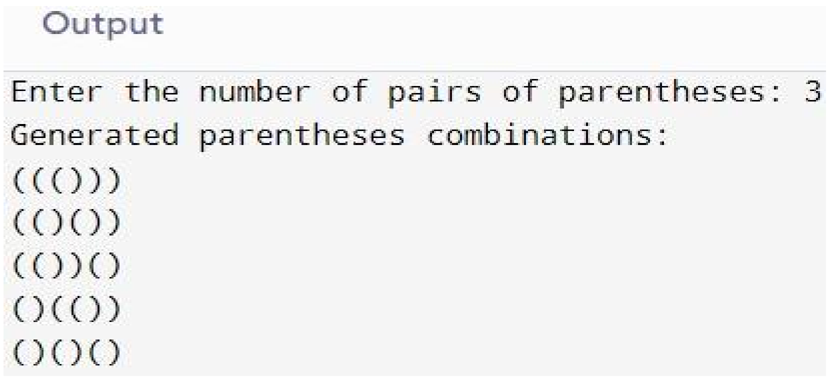
PROBLEM 12

Objective: Generate Parentheses

Code:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
void generateParenthesisHelper(int open, int close, int n, string current, vector<string>& result) {
   if (current.length() == 2 * n) {
      result.push_back(current);
      return;
   }
   if (open < n) {
      generateParenthesisHelper(open + 1, close, n, current + "(", result);
   }
   if (close < open) {
      generateParenthesisHelper(open, close + 1, n, current + ")", result);
   }}
vector<string> generateParenthesis(int n) {
   vector<string> result;
   generateParenthesisHelper(0, 0, n, "", result);
   return result;
}
int main() {
   int n;
   cout << "Enter the number of pairs of parentheses: ";
   cin >> n;
   vector<string> result = generateParenthesis(n);
   cout << "Generated parentheses combinations: \n";
   for (const string& s : result) {
      cout << s << endl;
   }
   return 0;
}
```

Output:



```
Enter the number of pairs of parentheses: 3
Generated parentheses combinations:
((()))
(()())
(())()
()(())
()()()
```

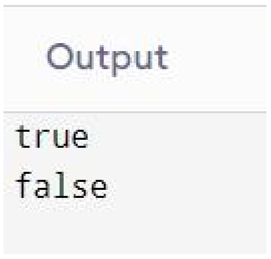
PROBLEM 13

Objective: Jump Game

Code:

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

bool canJump(vector<int>& nums) {
    int n = nums.size();
    int maxReach = 0; // The furthest index we can reach

    for (int i = 0; i < n; i++) {
        // If the current index is greater than the maximum reachable index, return false
        if (i > maxReach) {
            return false;
        }
        // Update the maximum reachable index
        maxReach = max(maxReach, i + nums[i]);
        // If we can reach the last index, return true
        if (maxReach >= n - 1) {
            return true;
        }
    }
    return false;
}

int main() {
    vector<int> nums1 = {2, 3, 1, 1, 4};
    vector<int> nums2 = {3, 2, 1, 0, 4};

    cout << (canJump(nums1) ? "true" : "false") << endl;  // Output: true
    cout << (canJump(nums2) ? "true" : "false") << endl;  // Output: false

    return 0;
}
```

Output:



```
true
false
```

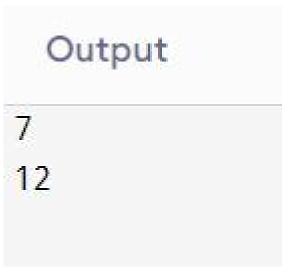
PROBLEM 14

Objective: Minimum Path Sum

Code:

```
#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 i = 1; i < m; i++) {
      dp[i][0] = dp[i - 1][0] + grid[i][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++) {
      for (int j = 1; j < n; j++) {
         dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j];
      }}
   return dp[m - 1][n - 1];  // The bottom-right corner holds the minimum path sum
}
int main() {
   vector<vector<int>> grid1 = {{1, 3, 1}, {1, 5, 1}, {4, 2, 1}};
   vector<vector<int>> grid2 = {{1, 2, 3}, {4, 5, 6}};

   cout << minPathSum(grid1) << endl;  // Output: 7
   cout << minPathSum(grid2) << endl;  // Output: 12

   return 0;
}
```

Output:



Output

```
7
12
```

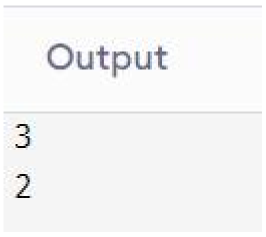
PROBLEM 15

Objective: Given an integer n, return the least number of perfect square numbers that sum to n.

Code:

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

int numSquares(int n) {
    vector<int> dp(n + 1, INT_MAX);
    dp[0] = 0;

    // Fill the dp array
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j * j <= i; j++) {
            dp[i] = min(dp[i], dp[i - j * j] + 1);
        }
    }

    return dp[n];  // The result will be stored at dp[n]
}

int main() {
    int n1 = 12;
    int n2 = 13;

    cout << numSquares(n1) << endl;  // Output: 3
    cout << numSquares(n2) << endl;  // Output: 2

    return 0;
}
```

Output:



```
3
2
```

PROBLEM 16

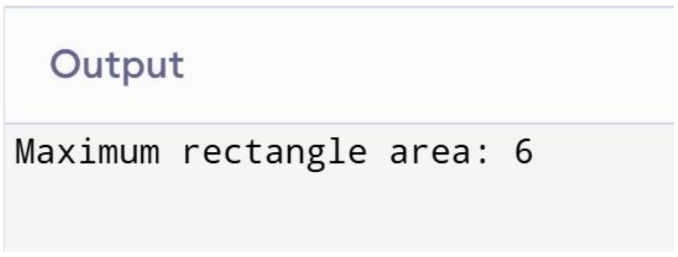
Objective: Maximal Rectangle

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) {
if (matrix[i][j] == '1') {
heights[j] += 1;
} else {
heights[j] = 0;
}}
maxArea = max(maxArea, largestRectangleArea(heights));
}
return maxArea;  }
private:
int largestRectangleArea(vector<int>& heights) {
stack<int> s;
heights.push_back(0); // Sentinel to ensure stack is emptied
int maxArea = 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() {
Solution solution;
vector<vector<char>> matrix = {
{'1', '0', '1', '0', '0'},{'1', '0', '1', '1', '1'},{'1', '1', '1', '1', '1'},{'1', '0', '0', '1', '0'}
```

```
    };
    cout << "Maximum rectangle area: " << solution.maximalRectangle(matrix) << endl;
    return 0;
}
```

Output:



```
Maximum rectangle area: 6
```

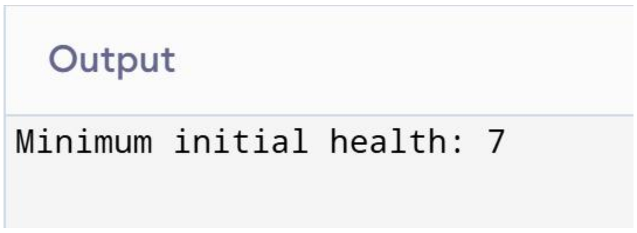
PROBLEM 17

Objective: Dungeon Game

Code:

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

class Solution {
public:
    int minimalInitialHealth(vector<vector<int>>& dungeon) {
        int m = dungeon.size();
        int n = dungeon[0].size();

        vector<vector<int>> dp(m, vector<int>(n, 0));
        dp[m - 1][n - 1] = max(1, 1 - dungeon[m - 1][n - 1]);

        for (int i = m - 2; i >= 0; --i) {
            dp[i][n - 1] = max(1, dp[i + 1][n - 1] - dungeon[i][n - 1]);
        }

        for (int j = n - 2; j >= 0; --j) {
            dp[m - 1][j] = max(1, dp[m - 1][j + 1] - dungeon[m - 1][j]);
        }

        for (int i = m - 2; i >= 0; --i) {
            for (int j = n - 2; j >= 0; --j) {
                dp[i][j] = max(1, min(dp[i + 1][j], dp[i][j + 1]) - dungeon[i][j]);
            }
        }
        return dp[0][0];
    }
};

int main() {
    Solution solution;
    vector<vector<int>> dungeon = {
        {-2, -3, 3},
        {-5, -10, 1},
        {10, 30, -5}
    };
    cout << "Minimum initial health: " << solution.minimalInitialHealth(dungeon) << endl;
    return 0;
}
```

Output:



Output

```
Minimum initial health: 7
```

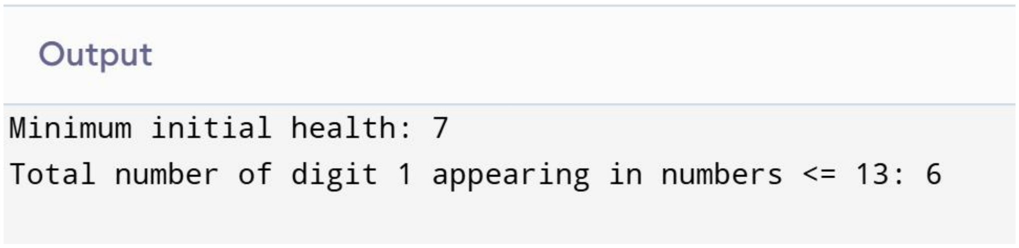
PROBLEM 18

Objective: Number of Digit One

Code:

```
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>
using namespace std;
class Solution {
public:
    int minimalInitialHealth(vector<vector<int>>& dungeon) {
        int m = dungeon.size();
        int n = dungeon[0].size();
        vector<vector<int>> dp(m, vector<int>(n, 0));
        dp[m - 1][n - 1] = max(1, 1 - dungeon[m - 1][n - 1]);
        for (int i = m - 2; i >= 0; --i) {
            dp[i][n - 1] = max(1, dp[i + 1][n - 1] - dungeon[i][n - 1]);
        }
        for (int j = n - 2; j >= 0; --j) {
            dp[m - 1][j] = max(1, dp[m - 1][j + 1] - dungeon[m - 1][j]);
        }

        for (int i = m - 2; i >= 0; --i) {
            for (int j = n - 2; j >= 0; --j) {
                dp[i][j] = max(1, min(dp[i + 1][j], dp[i][j + 1]) - dungeon[i][j]);
            }   }
        return dp[0][0];
    }
    int countDigitOne(int n) {
        int count = 0;
        for (long long place = 1; place <= n; place *= 10) {
            long long divisor = place * 10;
            count += (n / divisor) * place + min(max(n % divisor - place + 1, 0LL), place);
        }
        return count;
    }};
int main() {
    Solution solution;
    vector<vector<int>> dungeon = {
        {-2, -3, 3},
        {-5, -10, 1},
        {10, 30, -5}
    };
    cout << "Minimum initial health: " << solution.minimalInitialHealth(dungeon) << endl;
    int n = 13;
    cout << "Total number of digit 1 appearing in numbers <= " << n << ": " <<
solution.countDigitOne(n) << endl;
    return 0;
}
```

Output:



```
Minimum initial health: 7
Total number of digit 1 appearing in numbers <= 13: 6
```

PROBLEM 19

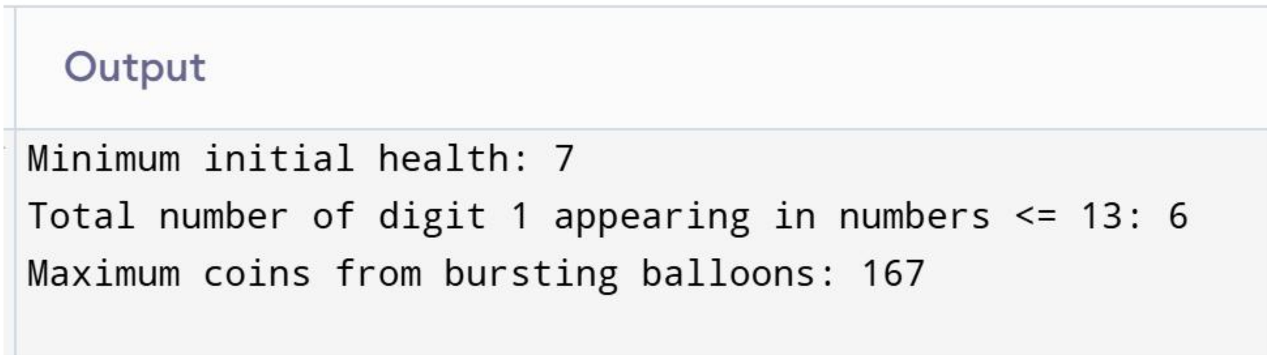
Objective: Burst Balloons

Code:

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

class Solution {
public:
   int minimalInitialHealth(vector<vector<int>>& dungeon) {
      int m = dungeon.size();
      int n = dungeon[0].size();
      vector<vector<int>> dp(m, vector<int>(n, 0));
      dp[m - 1][n - 1] = max(1, 1 - dungeon[m - 1][n - 1]);

      for (int i = m - 2; i >= 0; --i) {
         dp[i][n - 1] = max(1, dp[i + 1][n - 1] - dungeon[i][n - 1]);
      }
      for (int j = n - 2; j >= 0; --j) {
         dp[m - 1][j] = max(1, dp[m - 1][j + 1] - dungeon[m - 1][j]);
      }
      for (int i = m - 2; i >= 0; --i) {
         for (int j = n - 2; j >= 0; --j) {
            dp[i][j] = max(1, min(dp[i + 1][j], dp[i][j + 1]) - dungeon[i][j]);
         }
      }

      return dp[0][0];
   }
   int countDigitOne(int n) {
      int count = 0;
      for (long long place = 1; place <= n; place *= 10) {
         long long divisor = place * 10;
         count += (n / divisor) * place + min(max(n % divisor - place + 1, 0LL), place);
      }
      return count;
   }
   int maxCoins(vector<int>& nums) {
      int n = nums.size();
      vector<int> balloons(n + 2, 1);
      for (int i = 0; i < n; ++i) {
         balloons[i + 1] = nums[i];
      }
```

```
        vector<vector<int>> dp(n + 2, vector<int>(n + 2, 0));

        for (int length = 1; length <= n; ++length) {
            for (int left = 1; left <= n - length + 1; ++left) {
                int right = left + length - 1;
                for (int k = left; k <= right; ++k) {
                    dp[left][right] = max(dp[left][right],
                                          dp[left][k - 1] + balloons[left - 1] * balloons[k] * balloons[right + 1] +
dp[k + 1][right]);
                }
            }
        }

        return dp[1][n];
    }
};
int main() {
    Solution solution;

    // Dungeon Game Example
    vector<vector<int>> dungeon = {
        {-2, -3, 3},
        {-5, -10, 1},
        {10, 30, -5}
    };
    cout << "Minimum initial health: " << solution.minimalInitialHealth(dungeon) << endl;

    // Count Digit One Example
    int n = 13;
    cout << "Total number of digit 1 appearing in numbers <= " << n << ": " <<
solution.countDigitOne(n) << endl;

    // Burst Balloons Example
    vector<int> nums = {3, 1, 5, 8};
    cout << "Maximum coins from bursting balloons: " << solution.maxCoins(nums) << endl;

    return 0;
}
```

Output:


Output

```
Minimum initial health: 7
Total number of digit 1 appearing in numbers <= 13: 6
Maximum coins from bursting balloons: 167
```

PROBLEM 20

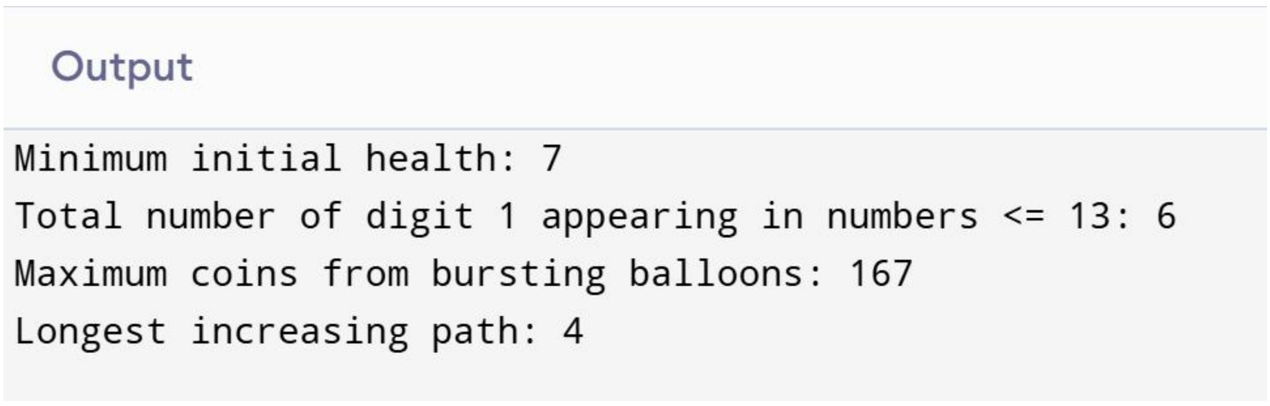
Objective: Longest Increasing Path in a Matrix

Code:

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

class Solution {
public:
    int minimalInitialHealth(vector<vector<int>>& dungeon) {
        int m = dungeon.size();
        int n = dungeon[0].size();

        vector<vector<int>> dp(m, vector<int>(n, 0));
        dp[m - 1][n - 1] = max(1, 1 - dungeon[m - 1][n - 1]);

        for (int i = m - 2; i >= 0; --i) {
            dp[i][n - 1] = max(1, dp[i + 1][n - 1] - dungeon[i][n - 1]);
        }

        for (int j = n - 2; j >= 0; --j) {
            dp[m - 1][j] = max(1, dp[m - 1][j + 1] - dungeon[m - 1][j]);
        }

        for (int i = m - 2; i >= 0; --i) {
            for (int j = n - 2; j >= 0; --j) {
                dp[i][j] = max(1, min(dp[i + 1][j], dp[i][j + 1]) - dungeon[i][j]);
            }
        }

        return dp[0][0];
    }

    int countDigitOne(int n) {
        int count = 0;
        for (long long place = 1; place <= n; place *= 10) {
            long long divisor = place * 10;
            count += (n / divisor) * place + min(max(n % divisor - place + 1, 0LL), place);
        }
        return count;
    }

    int maxCoins(vector<int>& nums) {
        int n = nums.size();
        vector<int> balloons(n + 2, 1);
        for (int i = 0; i < n; ++i) {
            balloons[i + 1] = nums[i];
        }

        vector<vector<int>> dp(n + 2, vector<int>(n + 2, 0));

        for (int length = 1; length <= n; ++length) {
            for (int left = 1; left <= n - length + 1; ++left) {
                int right = left + length - 1;
                for (int k = left; k <= right; ++k) {
                    dp[left][right] = max(dp[left][right],
                                          dp[left][k - 1] + balloons[left - 1] * balloons[k] * balloons[right + 1] +
dp[k + 1][right]);
                }
            }
        }

        return dp[1][n];
    }

    int longestIncreasingPath(vector<vector<int>>& matrix) {
        int m = matrix.size(), n = matrix[0].size();
        vector<vector<int>> dp(m, vector<int>(n, -1));
        int directions[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

        function<int(int, int)> dfs = [&](int i, int j) {
            if (dp[i][j] != -1) return dp[i][j];

            int maxPath = 1;
            for (auto& dir : directions) {
                int x = i + dir[0], y = j + dir[1];
                if (x >= 0 && x < m && y >= 0 && y < n && matrix[x][y] > matrix[i][j]) {
                    maxPath = max(maxPath, 1 + dfs(x, y));
                }
            }

            dp[i][j] = maxPath;
            return maxPath;
        };

        int result = 0;
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                result = max(result, dfs(i, j));
            }
        }

        return result;
    }
```

```
};

int main() {
    Solution solution;

    // Dungeon Game Example
    vector<vector<int>> dungeon = {
        {-2, -3, 3},
        {-5, -10, 1},
        {10, 30, -5}
    };
    cout << "Minimum initial health: " << solution.minimalInitialHealth(dungeon) << endl;

    // Count Digit One Example
    int n = 13;
    cout << "Total number of digit 1 appearing in numbers <= " << n << ": " <<
solution.countDigitOne(n) << endl;

    // Burst Balloons Example
    vector<int> nums = {3, 1, 5, 8};
    cout << "Maximum coins from bursting balloons: " << solution.maxCoins(nums) << endl;

    // Longest Increasing Path in Matrix Example
    vector<vector<int>> matrix = {
        {9, 9, 4},
        {6, 6, 8},
        {2, 1, 1}
    };
    cout << "Longest increasing path: " << solution.longestIncreasingPath(matrix) << endl;

    return 0;
}
```

<u>Output:</u>



Output

```
Minimum initial health: 7
Total number of digit 1 appearing in numbers <= 13: 6
Maximum coins from bursting balloons: 167
Longest increasing path: 4
```

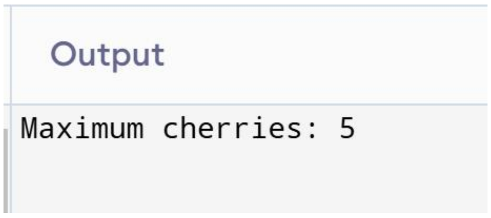
PROBLEM 21

Objective: Cherry Pickup
Code:

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

class Solution {
public:
   int cherryPickup(vector<vector<int>>& grid) {
      int n = grid.size();
      vector<vector<vector<int>>> dp(n, vector<vector<int>>(n, vector<int>(n, INT_MIN)));
      dp[0][0][0] = grid[0][0];

      for (int t = 1; t < 2 * n - 1; ++t) {
         vector<vector<vector<int>>>       newDp(n,       vector<vector<int>>(n,       vector<int>(n,
INT_MIN)));
         for (int x1 = max(0, t - (n - 1)); x1 <= min(n - 1, t); ++x1) {
            for (int x2 = max(0, t - (n - 1)); x2 <= min(n - 1, t); ++x2) {
               int y1 = t - x1;
               int y2 = t - x2;

               if (y1 >= n || y2 >= n || grid[x1][y1] == -1 || grid[x2][y2] == -1) {
                  continue;
               }

               int cherries = grid[x1][y1];
               if (x1 != x2) {
                  cherries += grid[x2][y2];
               }

               for (int px1 = x1 - 1; px1 <= x1; ++px1) {
                  for (int px2 = x2 - 1; px2 <= x2; ++px2) {
                     if (px1 >= 0 && px2 >= 0) {
                        newDp[x1][x2][t % n] = max(newDp[x1][x2][t % n], dp[px1][px2][(t - 1) % n]
+ cherries);
                     }
                  }
               }
            }
         }
         dp = move(newDp);
      }
```

```
        return max(0, dp[n - 1][n - 1][(2 * n - 2) % n]);
    }
};

int main() {
    Solution solution;
    vector<vector<int>> grid = {{0, 1, -1}, {1, 0, -1}, {1, 1, 1}};
    cout << "Maximum cherries: " << solution.cherryPickup(grid) << endl;
    return 0;
}
```

Output:



```
Maximum cherries: 5
```

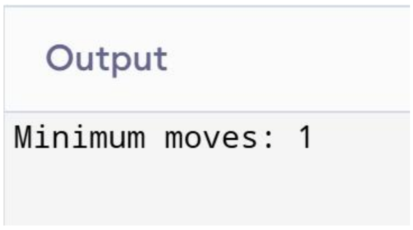
PROBLEM 22

Objective: Sliding Puzzle

Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_set>
#include <string>
using namespace std;

class Solution {
public:
    int slidingPuzzle(vector<vector<int>>& board) {
        string target = "123450";
        string start = "";
        for (auto& row : board) {
            for (auto& num : row) {
                start += to_string(num);
            }
        }

        vector<vector<int>> neighbors = {
            {1, 3}, {0, 2, 4}, {1, 5}, {0, 4}, {1, 3, 5}, {2, 4}
        };

        queue<pair<string, int>> q;
        unordered_set<string> visited;
        q.push({start, 0});
        visited.insert(start);

        while (!q.empty()) {
            auto [cur, steps] = q.front();
            q.pop();

            if (cur == target) {
                return steps;
            }

            int zeroPos = cur.find('0');
            for (int neighbor : neighbors[zeroPos]) {
                string next = cur;
                swap(next[zeroPos], next[neighbor]);
                if (!visited.count(next)) {
                    q.push({next, steps + 1});
                    visited.insert(next);
                }
            }
        }

        return -1;
    }
};

int main() {
    Solution solution;
    vector<vector<int>> board = {{1, 2, 3}, {4, 0, 5}};
    cout << "Minimum moves: " << solution.slidingPuzzle(board) << endl;
    return 0;
}
```

Output:



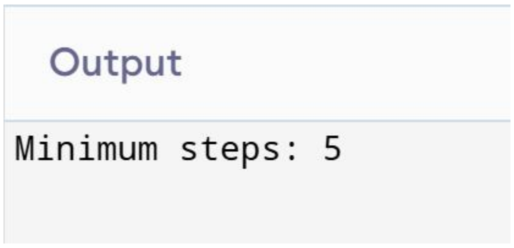
```
Minimum moves: 1
```

PROBLEM 23

Objective: Race Car
Code:

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

class Solution {
public:
    int racecar(int target) {
        queue<pair<int, int>> q;
        unordered_set<string> visited;

        q.push({0, 1}); // {position, speed}
        visited.insert("0,1");

        int steps = 0;
        while (!q.empty()) {
            int size = q.size();
            for (int i = 0; i < size; ++i) {
                auto [pos, speed] = q.front();
                q.pop();

                if (pos == target) return steps;

                // Accelerate
                int newPos = pos + speed;
                int newSpeed = speed * 2;
                string state = to_string(newPos) + "," + to_string(newSpeed);

                if (abs(newPos) <= 2 * target && !visited.count(state)) {
                    q.push({newPos, newSpeed});
                    visited.insert(state);
                }

                // Reverse
                newSpeed = speed > 0 ? -1 : 1;
                state = to_string(pos) + "," + to_string(newSpeed);

                if (!visited.count(state)) {
                    q.push({pos, newSpeed});
                    visited.insert(state);
                }
            }
            ++steps;
        }

        return -1;
    }
};

int main() {
    Solution solution;
    int target = 6;
    cout << "Minimum steps: " << solution.racecar(target) << endl;
    return 0;
}
```

Output:


Output

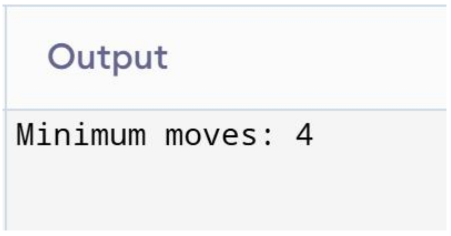
```
Minimum steps: 5
```

PROBLEM 24

Objective: Super Egg Drop
Code:

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

class Solution {
public:
    int superEggDrop(int k, int n) {
        vector<vector<int>> dp(k + 1, vector<int>(n + 1, 0));
        int moves = 0;

        while (dp[k][moves] < n) {
            ++moves;
            for (int eggs = 1; eggs <= k; ++eggs) {
                dp[eggs][moves] = dp[eggs - 1][moves - 1] + dp[eggs][moves - 1] + 1;
            }
        }

        return moves;
    }
};
int main() {
    Solution solution;
    int k = 3, n = 14;
    cout << "Minimum moves: " << solution.superEggDrop(k, n) << endl;
    return 0;
}
```

Output:



Output

```
Minimum moves: 4
```

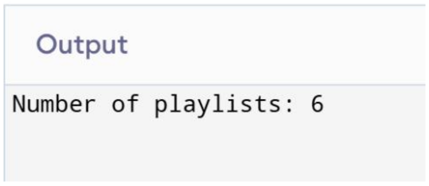
PROBLEM 25

Objective: Number of Music Playlists

Code:

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    int numMusicPlaylists(int n, int goal, int k) {
        const int MOD = 1e9 + 7;
        vector<vector<long long>> dp(goal + 1, vector<long long>(n + 1, 0));
        dp[0][0] = 1;

        for (int i = 1; i <= goal; ++i) {
            for (int j = 1; j <= n; ++j) {
                dp[i][j] = dp[i - 1][j - 1] * (n - (j - 1)) % MOD;
                if (j > k) {
                    dp[i][j] = (dp[i][j] + dp[i - 1][j] * (j - k) % MOD) % MOD;
                }
            }
        }

        return dp[goal][n];
    }
};
int main() {
    Solution solution;
    int n = 3, goal = 3, k = 1;
    cout << "Number of playlists: " << solution.numMusicPlaylists(n, goal, k) << endl;
    return 0;
}
```

Output:



```
Number of playlists: 6
```