Experiment 7

Student Name: Anurag Nayak **UID: 22BET10069**
Branch: BE-IT **Section/Group: 22BET-703/B**
Semester: 6 **Date of Performance: 18.3.2025**
Subject Name: Advance Programming Lab **Subject Code: 22ITP-367**

1. **Aim:**

   Write C++ programs for the problems based on Trees:

   - Climbing Stairs
   - Best Time to Buy and Sell a Stock
   - Maximum Subarray
   - House Robber
   - Jump Game
   - Unique Paths
   - Coin Change

2. **Implementation/Code:**

   1. Climbing Stairs:

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

int climbStairs(int n) {
    if (n <= 2) return n;
    int first = 1, second = 2, ways;
    for (int i = 3; i <= n; i++) {
        ways = first + second;
        first = second;
        second = ways;
    }
    return ways;
}
```

```
int main() {
    int n;
    cout << "Enter number of steps: ";
    cin >> n;
    cout << "Ways to climb: " << climbStairs(n) << endl;
    return 0;
}
```

2. Best Time to Buy and Sell a Stock:

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

int maxProfit(vector<int>& prices) {
    int minPrice = INT_MAX, maxProfit = 0;

    for (int price : prices) {
        minPrice = min(minPrice, price);
        maxProfit = max(maxProfit, price - minPrice);
    }

    return maxProfit;
}

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

    vector<int> prices(n);
    cout << "Enter stock prices: ";
    for (int i = 0; i < n; i++) {
        cin >> prices[i];
```

```
    }

    cout << "Maximum Profit: " << maxProfit(prices) << endl;
    return 0;
}
```

3. Maximum Subarray:

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

int maxSubArray(vector<int>& nums) {
    int maxSum = nums[0], currentSum = nums[0];

    for (int i = 1; i < nums.size(); i++) {
        currentSum = max(nums[i], currentSum + nums[i]);
        maxSum = max(maxSum, currentSum);
    }

    return maxSum;
}

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

    vector<int> nums(n);
    cout << "Enter elements: ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }
```

```
    cout << "Maximum Subarray Sum: " << maxSubArray(nums) << endl;
    return 0;
}
```

4. House Robber:

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

int rob(vector<int>& nums) {
    int prev1 = 0, prev2 = 0;

    for (int num : nums) {
        int temp = max(prev1, prev2 + num);
        prev2 = prev1;
        prev1 = temp;
    }

    return prev1;
}

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

    vector<int> nums(n);
    cout << "Enter money in each house: ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }

    cout << "Maximum money that can be robbed: " << rob(nums) << endl;
```

```
    return 0;
}
```

5. Jump Game:

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

bool canJump(vector<int>& nums) {
    int reachable = 0;

    for (int i = 0; i < nums.size(); i++) {
        if (i > reachable) return false;
        reachable = max(reachable, i + nums[i]);
    }

    return true;
}

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

    vector<int> nums(n);
    cout << "Enter jump lengths: ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }

    cout << (canJump(nums) ? "true" : "false") << endl;
    return 0;
}
```

6. Unique Paths:

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

long long factorial(int num, int stop) {
    long long result = 1;
    for (int i = num; i > stop; i--) {
        result *= i;
    }
    return result;
}

int uniquePaths(int m, int n) {
    int N = m + n - 2;
    int r = min(m - 1, n - 1);
    long long num = factorial(N, N - r);
    long long den = factorial(r, 0);
    return num / den;
}

int main() {
    int m, n;
    cout << "Enter grid size (m, n): ";
    cin >> m >> n;

    cout << "Unique Paths: " << uniquePaths(m, n) << endl;
    return 0;
}
```

7. Coin Change:

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

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

int coinChange(vector<int>& coins, int amount) {
    vector<int> dp(amount + 1, INT_MAX);
    dp[0] = 0; // Base case: 0 coins needed for amount 0

    for (int i = 1; i <= amount; i++) {
        for (int coin : coins) {
            if (i >= coin && dp[i - coin] != INT_MAX) {
                dp[i] = min(dp[i], dp[i - coin] + 1);
            }
        }
    }

    return (dp[amount] == INT_MAX) ? -1 : dp[amount];
}

int main() {
    vector<int> coins;
    int n, amount;

    cout << "Enter number of coin types: ";
    cin >> n;
    coins.resize(n);

    cout << "Enter coin denominations: ";
    for (int i = 0; i < n; i++) {
        cin >> coins[i];
    }

    cout << "Enter the total amount: ";
    cin >> amount;
```

```
    int result = coinChange(coins, amount);
    cout << "Minimum coins required: " << result << endl;

    return 0;
}
```