



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Experiment 7.1

**Student Name:** Vishal Bhatia

**Branch:** BE-CSE

**Semester:** 6<sup>th</sup>

**Subject Name:** Advanced Programming-2

**UID:** 22BCS12608

**Section/Group:** IoT\_641(A)

**Date of Performance:** 11/3/25

**Subject Code:** 22CSP-351

**1. Aim:** You are given an array representing the amount of money in each house on a street. You cannot rob two adjacent houses. Determine the maximum amount you can rob without alerting the police.

**2. Objective:** Implement dynamic programming.

**3. Implementation/Code:**

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        if (n == 1) {
            return nums[0];
        }
        vector<int> dp(n, 0);
        dp[0] = nums[0];
        dp[1] = max(nums[0], nums[1]);
        for (int i = 2; i < n; i++) {
            dp[i] = max(dp[i - 1], nums[i] + dp[i - 2]);
        }

        return dp[n - 1];
    }
};
```

**4. Output**

Case 1:

Input:

nums=[1,2,3,1]

Output:

4

Case 2:

Input:

nums=[2,7,9,3,1]

Output:

12

## Experiment 7.2

1. **Aim:** - Given an array where each element represents the maximum jump length from that position, determine if you can reach the last index starting from the first index.
2. **Objective:** Implement dynamic programming

### 3. Code:

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int jump=0;
        for(int i=0;i<nums.size()-1;i++)
        {
            if(jump<nums[i])
            {
                jump=nums[i];
            }
            if(jump--==0)
                return false;
        }
        return true;
    }
};
```

### 4. Output

Case 1:

Input:

nums= [2,3,1,1,4]

Output:

true

Case 2:

Input:

Root= [3,2,1,0,4]

Output:

false

## Experiment 7.3

1. **Aim:** Given an array of integers, find the contiguous subarray (of at least one element) that has the largest product and return its product.
2. **Objective:** Implement dynamic programming
3. **Code:**

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int maxi = INT_MIN;
        int prod=1;
        for(int i=0;i<nums.size();i++)
        {
            prod*=nums[i];
            maxi=max(prod,maxi);
            if(prod==0)
                prod=1;
        }
        prod=1;
        for(int i=nums.size()-1;i>=0;i--)
        {
            prod*=nums[i];
            maxi=max(prod,maxi);
            if(prod==0)
                prod=1;
        }
        return maxi;
    }
};
```

## 4. Output

Case 1:

Input:

nums= [2,3,-2,4]

Output:

6

Case 2:

Input:

nums= [-2,0,-1]

Output:

0

## 5. Learning Outcomes:

- Understand the steps involved in dynamic programming.
- Learn how to analyze and compare the time complexity.
- Implement appropriate algorithm based on the problem's constraints.