

## Day 2

### Array & Linked list

#### Very Easy

#### **Q 1 : Majority Elements**

Given an array `nums` of size `n`, return the majority element.

The majority element is the element that appears more than  $\lfloor n / 2 \rfloor$  times. You may assume that the majority element always exists in the array.

##### **Example 1:**

Input: `nums = [3,2,3]`

Output: 3

##### **Example 2:**

Input: `nums = [2,2,1,1,1,2,2]`

Output: 2

##### **Constraints:**

`n == nums.length`

`1 <= n <= 5 * 104`

`-109 <= nums[i] <= 109`

**Follow-up:** Could you solve the problem in linear time and in  $O(1)$  space?

##### **Code:**

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
int majorityElement(vector<int>& nums) {  
    int count = 0, candidate = 0;  
    for (int num : nums) {  
        if (count == 0) {  
            candidate = num;  
        }  
        count += (num == candidate) ? 1 : -1;  
    }  
    return candidate;  
}
```

```

int main() {
    vector<int> nums = {3, 2, 3};
    int majority = majorityElement(nums);
    cout << "Majority Element: " << majority << endl;
    return 0;
}

```

### Output:

The screenshot shows a C++ IDE with the following code in the editor:

```

1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int majorityElement(vector<int>& nums) {
7     int count = 0, candidate = 0;
8     for (int num : nums) {
9         if (count == 0) {
10             candidate = num;
11         }
12         count += (num == candidate) ? 1 : -1;
13     }
14     return candidate;
15 }
16
17 int main() {
18     vector<int> nums = {3, 2, 3};
19     int majority = majorityElement(nums);
20     cout << "Majority Element: " << majority << endl;
21     return 0;
22 }

```

On the right side, the 'Run' button is highlighted. Below it, the 'Output' section shows the status 'Status : Successfully executed'. The 'Time' is 0.0000 secs and 'Memory' is 3.472 Mb. The 'Your Output' section displays 'Majority Element: 3'.

### Question 2. Single Number

Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

#### Example 1:

Input: nums = [2,2,1]

Output: 1

#### Example 2:

Input: nums = [4,1,2,1,2]

Output: 4

#### Example 3:

Input: nums = [1]

Output: 1

#### Code:

```

#include <iostream>
#include <vector>

```

```

using namespace std;

```

```

int singleNumber(vector<int>& nums) {

```

```

    int result = 0;
    for (int num : nums) {
        result ^= num;
    }
    return result;
}

int main() {
    vector<int> nums1 = {2, 2, 1};
    cout << "Single Number 1: " << singleNumber(nums1) << endl;

    vector<int> nums2 = {4, 1, 2, 1, 2};
    cout << "Single Number 2: " << singleNumber(nums2) << endl;

    vector<int> nums3 = {1};
    cout << "Single Number 3: " << singleNumber(nums3) << endl;

    return 0;
}

```

**Output:**

main.cpp	Output
<pre> 1  #include &lt;iostream&gt; 2  #include &lt;vector&gt; 3 4  using namespace std; 5 6  int singleNumber(vector&lt;int&gt;&amp; nums) { 7      int result = 0; 8      for (int num : nums) { 9          result ^= num; 10     } 11     return result; 12 } 13 14 int main() { 15     vector&lt;int&gt; nums1 = {2, 2, 1}; 16     cout &lt;&lt; "Single Number 1: " &lt;&lt; singleNumber(nums1) &lt;&lt; endl; 17 18     vector&lt;int&gt; nums2 = {4, 1, 2, 1, 2}; 19     cout &lt;&lt; "Single Number 2: " &lt;&lt; singleNumber(nums2) &lt;&lt; endl; 20 21     vector&lt;int&gt; nums3 = {1}; 22     cout &lt;&lt; "Single Number 3: " &lt;&lt; singleNumber(nums3) &lt;&lt; endl; 23 24     return 0; 25 } </pre>	<pre> Single Number 1: 1 Single Number 2: 4 Single Number 3: 1  === Code Execution Successful === </pre>

### Question 3 Convert Sorted Array to Binary Search Tree

Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.

**Example 1:**

Input: nums = [-10,-3,0,5,9]

Output: [0,-3,9,-10,null,5]

Explanation: [0,-10,5,null,-3,null,9] is also accepted:

**Example 2:**

Input: nums = [1,3]

Output: [3,1]

Explanation: [1,null,3] and [3,1] are both height-balanced BSTs.

**Code:**

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

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

```
TreeNode* sortedArrayToBSTHelper(vector<int>& nums, int left, int right) {
    if (left > right) return nullptr;
    int mid = left + (right - left) / 2;
    TreeNode* root = new TreeNode(nums[mid]);
    root->left = sortedArrayToBSTHelper(nums, left, mid - 1);
    root->right = sortedArrayToBSTHelper(nums, mid + 1, right);
    return root;
}
```

```
TreeNode* sortedArrayToBST(vector<int>& nums) {
    return sortedArrayToBSTHelper(nums, 0, nums.size() - 1);
}
```

```
void printTree(TreeNode* root) {
    if (!root) return;
    cout << root->val << " ";
    printTree(root->left);
    printTree(root->right);
}
```

```

int main() {
    vector<int> nums = {-10, -3, 0, 5, 9};
    TreeNode* root = sortedArrayToBST(nums);
    printTree(root);
    return 0;
}

```

**Output:**

main.cpp	Output
<pre> 1  #include &lt;iostream&gt; 2  #include &lt;vector&gt; 3  using namespace std; 4 5  struct TreeNode { 6      int val; 7      TreeNode* left; 8      TreeNode* right; 9      TreeNode(int x) : val(x), left(nullptr), right(nullptr) {} 10 }; 11 12 TreeNode* sortedArrayToBSTHelper(vector&lt;int&gt;&amp; nums, int left, int     right) { 13     if (left &gt; right) return nullptr; 14     int mid = left + (right - left) / 2; 15     TreeNode* root = new TreeNode(nums[mid]); 16     root-&gt;left = sortedArrayToBSTHelper(nums, left, mid - 1); 17     root-&gt;right = sortedArrayToBSTHelper(nums, mid + 1, right); 18     return root; 19 } 20 21 TreeNode* sortedArrayToBST(vector&lt;int&gt;&amp; nums) { 22     return sortedArrayToBSTHelper(nums, 0, nums.size() - 1); 23 } 24 25 void printTree(TreeNode* root) { 26     if (!root) return; </pre>	<pre> 0 -10 -3 5 9 === Code Execution Successful === </pre>

[Easy](#)

## Question 1. Pascal's Triangle

Given an integer numRows, return the first numRows of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:

**Example 1:**

Input: numRows = 5

Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

**Example 2:**

Input: numRows = 1

Output: [[1]]

**Code:**

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
vector<vector<int>>> generate(int numRows) {  
    vector<vector<int>>> triangle(numRows);  
    for (int i = 0; i < numRows; i++) {  
        triangle[i].resize(i + 1);  
        triangle[i][0] = triangle[i][i] = 1;  
        for (int j = 1; j < i; j++) {  
            triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];  
        }  
    }  
    return triangle;  
}
```

```
int main() {  
    int numRows = 5;  
    vector<vector<int>>> result = generate(numRows);  
    for (const auto& row : result) {  
        for (int num : row) {  
            cout << num << " ";  
        }  
        cout << endl;  
    }  
    return 0;  
}
```

**Output:**

main.cpp	Output
<pre>1 #include &lt;iostream&gt; 2 #include &lt;vector&gt; 3 using namespace std; 4 5 vector&lt;vector&lt;int&gt;&gt; generate(int numRows) { 6     vector&lt;vector&lt;int&gt;&gt; triangle(numRows); 7     for (int i = 0; i &lt; numRows; i++) { 8         triangle[i].resize(i + 1); 9         triangle[i][0] = triangle[i][i] = 1; 10        for (int j = 1; j &lt; i; j++) { 11            triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j]; 12        } 13    } 14    return triangle; 15 } 16 17 int main() { 18     int numRows = 5; 19     vector&lt;vector&lt;int&gt;&gt; result = generate(numRows); 20     for (const auto&amp; row : result) { 21         for (int num : row) { 22             cout &lt;&lt; num &lt;&lt; " "; 23         } 24         cout &lt;&lt; endl; 25     } 26     return 0;</pre>	<pre>1 1 1 1 2 1 1 3 3 1 1 4 6 4 1  === Code Execution Successful ===</pre>

## Question 2. Remove Element

Given an integer array `nums` sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in `nums`.

Consider the number of unique elements of `nums` to be `k`, to get accepted, you need to do the following things:

Change the array `nums` such that the first `k` elements of `nums` contain the unique elements in the order they were present in `nums` initially. The remaining elements of `nums` are not important as well as the size of `nums`.

Return `k`.

### Example 1:

Input: `nums = [1,1,2]`

Output: 2, `nums = [1,2,_]`

Explanation: Your function should return `k = 2`, with the first two elements of `nums` being 1 and 2 respectively.

It does not matter what you leave beyond the returned `k` (hence they are underscores).

### Example 2:

Input: nums = [0,0,1,1,1,2,2,3,3,4]

Output: 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

### Code:

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

int removeDuplicates(vector<int>& nums) {
    if (nums.empty()) return 0;
    int k = 1;
    for (int i = 1; i < nums.size(); i++) {
        if (nums[i] != nums[i - 1]) {
            nums[k] = nums[i];
            k++;
        }
    }
    return k;
}

int main() {
    vector<int> nums = {0, 0, 1, 1, 1, 2, 2, 3, 3, 4};
    int k = removeDuplicates(nums);
    cout << "k = " << k << ", nums = [";
    for (int i = 0; i < k; i++) {
        cout << nums[i];
        if (i < k - 1) cout << ", ";
    }
    cout << "]" << endl;
    return 0;
}
```

### Output





Input: ops = ["5","2","C","D","+"]

Output: 30

Explanation:

"5" - Add 5 to the record, record is now [5].

"2" - Add 2 to the record, record is now [5, 2].

"C" - Invalidate and remove the previous score, record is now [5].

"D" - Add  $2 * 5 = 10$  to the record, record is now [5, 10].

"+" - Add  $5 + 10 = 15$  to the record, record is now [5, 10, 15].

The total sum is  $5 + 10 + 15 = 30$ .

### Example 2:

Input: ops = ["5","-2","4","C","D","9","+","+"]

Output: 27

Explanation:

"5" - Add 5 to the record, record is now [5].

"-2" - Add -2 to the record, record is now [5, -2].

"4" - Add 4 to the record, record is now [5, -2, 4].

"C" - Invalidate and remove the previous score, record is now [5, -2].

"D" - Add  $2 * -2 = -4$  to the record, record is now [5, -2, -4].

"9" - Add 9 to the record, record is now [5, -2, -4, 9].

"+" - Add  $-4 + 9 = 5$  to the record, record is now [5, -2, -4, 9, 5].

"+" - Add  $9 + 5 = 14$  to the record, record is now [5, -2, -4, 9, 5, 14].

The total sum is  $5 + -2 + -4 + 9 + 5 + 14 = 27$ .

### Code:

```
#include <iostream>
```

```
#include <vector>
```

```
#include <stack>
```

```
using namespace std;
```

```
int calPoints(vector<string>& ops) {
```

```
    stack<int> scores;
```

```
    int sum = 0;
```

```
    for (const string& op : ops) {
```

```
        if (op == "+") {
```

```
            int top = scores.top();
```

```
            scores.pop();
```

```
            int secondTop = scores.top();
```

```
            scores.push(top);
```

```
            scores.push(top + secondTop);
```

```
        } else if (op == "D") {
```

```
            scores.push(scores.top() * 2);
```

```

        } else if (op == "C") {
            scores.pop();
        } else {
            scores.push(stoi(op));
        }
    }

    while (!scores.empty()) {
        sum += scores.top();
        scores.pop();
    }

    return sum;
}

int main() {
    vector<string> ops1 = {"5", "2", "C", "D", "+"};
    cout << "Total Score 1: " << calPoints(ops1) << endl;

    return 0;
}

```

**Output:**

main.cpp	Run	Output
<pre> 1  #include &lt;iostream&gt; 2  #include &lt;vector&gt; 3  #include &lt;stack&gt; 4 5  using namespace std; 6 7  int calPoints(vector&lt;string&gt;&amp; ops) { 8      stack&lt;int&gt; scores; 9      int sum = 0; 10 11     for (const string&amp; op : ops) { 12         if (op == "+") { 13             int top = scores.top(); 14             scores.pop(); 15             int secondTop = scores.top(); 16             scores.push(top); 17             scores.push(top + secondTop); 18         } else if (op == "D") { 19             scores.push(scores.top() * 2); 20         } else if (op == "C") { 21             scores.pop(); 22         } else { 23             scores.push(stoi(op)); 24         } 25     } 26 </pre>	Run	<pre> Total Score 1: 30  === Code Execution Successful === </pre>

**Medium:**

Question 1. Container With Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.

### Example 1:

Input: height = [1,8,6,2,5,4,8,3,7]

Output: 49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

### Code:

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

int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int maxArea = 0;
    while (left < right) {
        int width = right - left;
        int currentArea = min(height[left], height[right]) * width;
        maxArea = max(maxArea, currentArea);
        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }
    return maxArea;
}

int main() {
    vector<int> height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    cout << "Output: " << maxArea(height) << endl;
    return 0;
}
```

}

## Output:

main.cpp	Output
<pre>1 #include &lt;iostream&gt; 2 #include &lt;vector&gt; 3 using namespace std; 4 5 int maxArea(vector&lt;int&gt;&amp; height) { 6     int left = 0, right = height.size() - 1; 7     int maxArea = 0; 8     while (left &lt; right) { 9         int width = right - left; 10        int currentArea = min(height[left], height[right]) * width; 11        maxArea = max(maxArea, currentArea); 12        if (height[left] &lt; height[right]) { 13            left++; 14        } else { 15            right--; 16        } 17    } 18    return maxArea; 19 } 20 21 int main() { 22     vector&lt;int&gt; height = {1, 8, 6, 2, 5, 4, 8, 3, 7}; 23     cout &lt;&lt; "Output: " &lt;&lt; maxArea(height) &lt;&lt; endl; 24     return 0; 25 }</pre>	<p>Output: 49</p> <p>=== Code Execution Successful ===</p>

## Question 2. Valid Sudoku

Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

Each row must contain the digits 1-9 without repetition.

Each column must contain the digits 1-9 without repetition.

Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.

Note:

A Sudoku board (partially filled) could be valid but is not necessarily solvable.

Only the filled cells need to be validated according to the mentioned rules.

### Example 1:

Input: board =

```
[["5","3",".",".","7",".",".",".","."],
["6",".",".","1","9","5",".",".","."],
[[".","9","8",".",".",".","6","."],
["8",".",".","6",".",".","3"],
["4",".","8","3",".","1"],
["7",".","2",".",".","6"]]
```

```

,[".", "6", ".", ".", ".", ".", "2", "8", "."]
,[".", ".", ".", "4", "1", "9", ".", ".", "5"]
,[".", ".", ".", ".", "8", ".", ".", "7", "9"]]

```

Output: true

### Code:

```

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

bool isValidSudoku(vector<vector<char>>& board) {
    for (int i = 0; i < 9; i++) {
        unordered_set<char> rows, cols, box;
        for (int j = 0; j < 9; j++) {
            if (board[i][j] != '.' && !rows.insert(board[i][j]).second) return false;
            if (board[j][i] != '.' && !cols.insert(board[j][i]).second) return false;
            int rowIndex = 3 * (i / 3), colIndex = 3 * (i % 3);
            char cell = board[rowIndex + j / 3][colIndex + j % 3];
            if (cell != '.' && !box.insert(cell).second) return false;
        }
    }
    return true;
}

int main() {
    vector<vector<char>> board = {
        {'5', '3', '.', '.', '7', '.', '.', '.', '.'},
        {'6', '.', '.', '1', '9', '5', '.', '.', '.'},
        {'.', '9', '8', '.', '.', '.', '.', '6', '.'},
        {'8', '.', '.', '.', '6', '.', '.', '.', '3'},
        {'4', '.', '.', '8', '.', '3', '.', '.', '1'},
        {'7', '.', '.', '.', '2', '.', '.', '.', '6'},
        {'.', '6', '.', '.', '.', '.', '2', '8', '.'},
        {'.', '.', '.', '4', '1', '9', '.', '.', '5'},
        {'.', '.', '.', '.', '8', '.', '.', '7', '9'}
    };
    cout << "Output: " << (isValidSudoku(board) ? "true" : "false") << endl;
    return 0;
}

```

**Output:**



```

int jump(vector<int>& nums) {
    int jumps = 0, currentEnd = 0, farthest = 0;
    for (int i = 0; i < nums.size() - 1; i++) {
        farthest = max(farthest, i + nums[i]);
        if (i == currentEnd) {
            jumps++;
            currentEnd = farthest;
        }
    }
    return jumps;
}

int main() {
    vector<int> nums = {2, 3, 1, 1, 4};
    cout << "Output: " << jump(nums) << endl;
    return 0;
}

```

**Output:**

main.cpp	Run	Output
<pre> 1  #include &lt;iostream&gt; 2  #include &lt;vector&gt; 3  using namespace std; 4 5  int jump(vector&lt;int&gt;&amp; nums) { 6      int jumps = 0, currentEnd = 0, farthest = 0; 7      for (int i = 0; i &lt; nums.size() - 1; i++) { 8          farthest = max(farthest, i + nums[i]); 9          if (i == currentEnd) { 10             jumps++; 11             currentEnd = farthest; 12         } 13     } 14     return jumps; 15 } 16 17 int main() { 18     vector&lt;int&gt; nums = {2, 3, 1, 1, 4}; 19     cout &lt;&lt; "Output: " &lt;&lt; jump(nums) &lt;&lt; endl; 20     return 0; 21 } 22 </pre>	Run	<p>Output: 2</p> <p>=== Code Execution Successful ===</p>

## Hard

### Question 1. Maximum Number of Groups Getting Fresh Donuts



There is a donuts shop that bakes donuts in batches of batchSize. They have a rule where they must serve all of the donuts of a batch before serving any donuts of the next batch. You are given an integer batchSize and an integer array groups, where groups[i] denotes that there is a group of groups[i] customers that will visit the shop. Each customer will get exactly one donut.

When a group visits the shop, all customers of the group must be served before serving any of the following groups. A group will be happy if they all get fresh donuts. That is, the first customer of the group does not receive a donut that was left over from the previous group.

You can freely rearrange the ordering of the groups. Return the maximum possible number of happy groups after rearranging the groups.

### Example 1:

Input: batchSize = 3, groups = [1,2,3,4,5,6]

Output: 4

Explanation: You can arrange the groups as [6,2,4,5,1,3]. Then the 1st, 2nd, 4th, and 6th groups will be happy.

### Code:

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

int maxHappyGroups(int batchSize, vector<int>& groups) {
    int n = groups.size();
    vector<int> dp(batchSize, -1);
    dp[0] = 0;

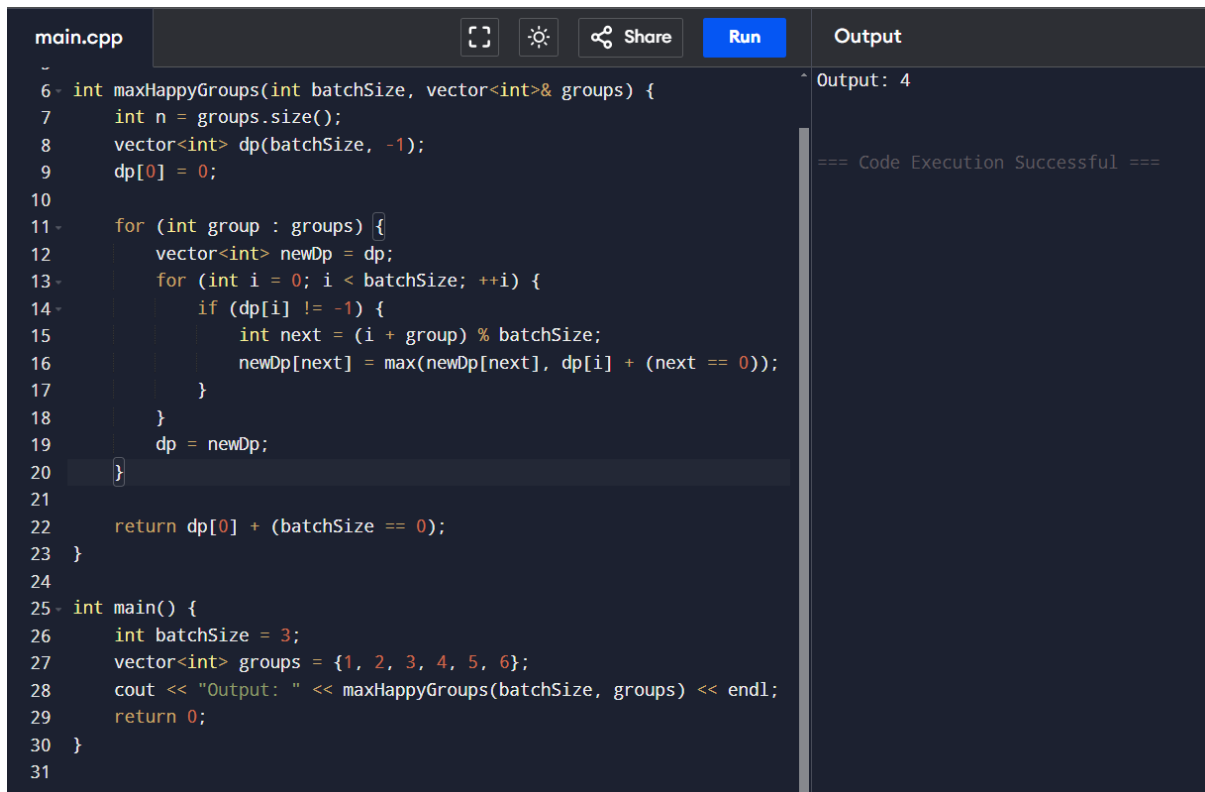
    for (int group : groups) {
        vector<int> newDp = dp;
        for (int i = 0; i < batchSize; ++i) {
            if (dp[i] != -1) {
                int next = (i + group) % batchSize;
                newDp[next] = max(newDp[next], dp[i] + (next == 0));
            }
        }
        dp = newDp;
    }

    return dp[0] + (batchSize == 0);
}
```

```
}
```

```
int main() {  
    int batchSize = 3;  
    vector<int> groups = {1, 2, 3, 4, 5, 6};  
    cout << "Output: " << maxHappyGroups(batchSize, groups) << endl;  
    return 0;  
}
```

**Output:**



The screenshot shows a C++ code editor with a dark theme. The code is for a function `maxHappyGroups` and a `main` function. The `maxHappyGroups` function takes a `batchSize` and a `vector<int> groups` as input. It initializes a `dp` array of size `batchSize` with `-1`, except for `dp[0]` which is `0`. It then iterates over each group in the `groups` vector. For each group, it creates a new `dp` array (`newDp`) and updates it based on the current group's value. The update logic is: for each `i` from `0` to `batchSize - 1`, if `dp[i] != -1`, then `newDp[(i + group) % batchSize] = max(newDp[(i + group) % batchSize], dp[i] + (next == 0))`, where `next` is `(i + group) % batchSize`. After updating `newDp`, it assigns `dp = newDp`. Finally, it returns `dp[0] + (batchSize == 0)`. The `main` function sets `batchSize = 3` and `groups = {1, 2, 3, 4, 5, 6}`, and prints the result of `maxHappyGroups(batchSize, groups)`. The output is `4`. The code execution is successful.

```
main.cpp  
-  
6 int maxHappyGroups(int batchSize, vector<int>& groups) {  
7     int n = groups.size();  
8     vector<int> dp(batchSize, -1);  
9     dp[0] = 0;  
10  
11     for (int group : groups) {  
12         vector<int> newDp = dp;  
13         for (int i = 0; i < batchSize; ++i) {  
14             if (dp[i] != -1) {  
15                 int next = (i + group) % batchSize;  
16                 newDp[next] = max(newDp[next], dp[i] + (next == 0));  
17             }  
18         }  
19         dp = newDp;  
20     }  
21  
22     return dp[0] + (batchSize == 0);  
23 }  
24  
25 int main() {  
26     int batchSize = 3;  
27     vector<int> groups = {1, 2, 3, 4, 5, 6};  
28     cout << "Output: " << maxHappyGroups(batchSize, groups) << endl;  
29     return 0;  
30 }  
31
```

Output  
Output: 4  
=== Code Execution Successful ===

## Question 2 Cherry Pickup II

You are given a rows x cols matrix grid representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the (i, j) cell.

You have two robots that can collect cherries for you:

Robot #1 is located at the top-left corner (0, 0), and

Robot #2 is located at the top-right corner (0, cols - 1).

Return the maximum number of cherries collection using both robots by following the rules below:

From a cell (i, j), robots can move to cell (i + 1, j - 1), (i + 1, j), or (i + 1, j + 1).

When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell.

When both robots stay in the same cell, only one takes the cherries.  
Both robots cannot move outside of the grid at any moment.  
Both robots should reach the bottom row in grid.

### Example 1:

Input: grid = [[3,1,1],[2,5,1],[1,5,5],[2,1,1]]

Output: 24

Explanation: Path of robot #1 and #2 are described in color green and blue respectively.

Cherries taken by Robot #1,  $(3 + 2 + 5 + 2) = 12$ .

Cherries taken by Robot #2,  $(1 + 5 + 5 + 1) = 12$ .

Total of cherries:  $12 + 12 = 24$ .

Code:

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

int cherryPickup(vector<vector<int>>& grid) {
    int rows = grid.size(), cols = grid[0].size();
    vector<vector<int>> dp(rows, vector<int>(cols, 0));

    // Initialize the last row with the cherries available at each cell
    for (int j = 0; j < cols; j++) {
        dp[rows - 1][j] = grid[rows - 1][j];
    }

    // DP to fill up the grid from the bottom row to the top row
    for (int i = rows - 2; i >= 0; i--) {
        for (int j1 = 0; j1 < cols; j1++) {
            for (int j2 = 0; j2 < cols; j2++) {
                int maxCherries = 0;
                // Try all possible moves for robot 1 and robot 2
                for (int dj1 = -1; dj1 <= 1; dj1++) {
                    for (int dj2 = -1; dj2 <= 1; dj2++) {
                        int nj1 = j1 + dj1, nj2 = j2 + dj2;
                        if (nj1 >= 0 && nj1 < cols && nj2 >= 0 && nj2 < cols) {
                            maxCherries = max(maxCherries, dp[i + 1][nj1] + dp[i + 1][nj2]);
                        }
                    }
                }
                dp[i][j1] = maxCherries + grid[i][j1];
            }
        }
    }
}
```

```

    return dp[0][0]; // Return the maximum cherries at the starting position
}

int main() {
    vector<vector<int>> grid = {{3,1,1},{2,5,1},{1,5,5},{2,1,1}};
    cout << "Output: " << cherryPickup(grid) << endl;
    return 0;
}

```

**Output:**

```

main.cpp
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 using namespace std;
5
6 int cherryPickup(vector<vector<int>>& grid) {
7     int rows = grid.size(), cols = grid[0].size();
8     vector<vector<int>> dp(rows, vector<int>(cols, 0));
9
10    // Initialize the last row with the cherries available at each
11    // cell
12    for (int j = 0; j < cols; j++) {
13        dp[rows - 1][j] = grid[rows - 1][j];
14    }
15
16    // DP to fill up the grid from the bottom row to the top row
17    for (int i = rows - 2; i >= 0; i--) {
18        for (int j1 = 0; j1 < cols; j1++) {
19            for (int j2 = 0; j2 < cols; j2++) {
20                int maxCherries = 0;
21                // Try all possible moves for robot 1 and robot 2
22                for (int dj1 = -1; dj1 <= 1; dj1++) {
23                    for (int dj2 = -1; dj2 <= 1; dj2++) {
24                        int nj1 = j1 + dj1, nj2 = j2 + dj2;
25                        if (nj1 >= 0 && nj1 < cols && nj2 >= 0 &&
26                            nj2 < cols) {
27                            maxCherries = max(maxCherries, dp[i +

```

Output: 45

=== Code Execution Successful ===

### Question 3: Maximum Number of Darts Inside of a Circular Dartboard

Alice is throwing  $n$  darts on a very large wall. You are given an array darts where  $\text{darts}[i] = [x_i, y_i]$  is the position of the  $i$ th dart that Alice threw on the wall.

Bob knows the positions of the  $n$  darts on the wall. He wants to place a dartboard of radius  $r$  on the wall so that the maximum number of darts that Alice throws lie on the dartboard.

Given the integer  $r$ , return the maximum number of darts that can lie on the dartboard.

#### Example 1:

Input: darts =  $[[-2,0],[2,0],[0,2],[0,-2]]$ ,  $r = 2$

Output: 4

Explanation: Circle dartboard with center in (0,0) and radius = 2 contain all points.

**Code:**

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

int maxDartsInBoard(vector<vector<int>>& darts, int r) {
    int n = darts.size();
    int maxDarts = 0;

    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            // Find the center of the dartboard as the midpoint of the two darts
            double centerX = (darts[i][0] + darts[j][0]) / 2.0;
            double centerY = (darts[i][1] + darts[j][1]) / 2.0;

            int count = 0;
            // Count how many darts are within the dartboard with center (centerX, centerY) and
            // radius r
            for (int k = 0; k < n; k++) {
                double dist = sqrt(pow(darts[k][0] - centerX, 2) + pow(darts[k][1] - centerY, 2));
                if (dist <= r) {
                    count++;
                }
            }
            maxDarts = max(maxDarts, count);
        }
    }

    return maxDarts;
}

int main() {
    vector<vector<int>> darts = {{-2, 0}, {2, 0}, {0, 2}, {0, -2}};
    int r = 2;
    cout << "Output: " << maxDartsInBoard(darts, r) << endl;
    return 0;
}
```

**Output**

main.cpp



Run

Output

```
1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <algorithm>
5 using namespace std;
6
7 int maxDartsInBoard(vector<vector<int>>& darts, int r) {
8     int n = darts.size();
9     int maxDarts = 0;
10
11     for (int i = 0; i < n; i++) {
12         for (int j = i; j < n; j++) {
13             // Find the center of the dartboard as the midpoint of the
14             double centerX = (darts[i][0] + darts[j][0]) / 2.0;
15             double centerY = (darts[i][1] + darts[j][1]) / 2.0;
16
17             int count = 0;
18             // Count how many darts are within the dartboard with center
19             (centerX, centerY) and radius r
20             for (int k = 0; k < n; k++) {
21                 double dist = sqrt(pow(darts[k][0] - centerX, 2) + pow
22                 (darts[k][1] - centerY, 2));
23                 if (dist <= r) {
24                     count++;
25                 }
26             }
27         }
28     }
29     return maxDarts;
30 }
```

Output: 4

=== Code Execution Successful ===