

main.cpp



Share

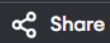
Run

Output

```
1 #include <vector>
2 #include <iostream>
3 using namespace std;
4
5 int majorityElement(vector<int>& nums) {
6     int candidate = 0, count = 0;
7     for (int num : nums) {
8         if (count == 0) {
9             candidate = num;
10        }
11        count += (num == candidate) ? 1 : -1;
12    }
13    return candidate;
14 }
15
16 int main() {
17     vector<int> nums1 = {3, 2, 3};
18     vector<int> nums2 = {2, 2, 1, 1, 1, 2, 2};
19     cout << majorityElement(nums1) << endl; // Output: 3
20     cout << majorityElement(nums2) << endl; // Output: 2
21     return 0;
22 }
23
```

```
3
2
=== Code Execution Successful ===
```

main.cpp



Run

Output

```
1 #include <vector>
2 #include <iostream>
3 using namespace std;
4
5 vector<vector<int>> generate(int numRows) {
6     vector<vector<int>> triangle;
7     for (int i = 0; i < numRows; ++i) {
8         vector<int> row(i + 1, 1);
9         for (int j = 1; j < i; ++j) {
10             row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
11         }
12         triangle.push_back(row);
13     }
14     return triangle;
15 }
16
17 void printTriangle(const vector<vector<int>>& triangle) {
18     for (const auto& row : triangle) {
19         for (int num : row) {
20             cout << num << " ";
21         }
22         cout << endl;
23     }
24 }
25
26 int main() {
27     int numRows = 5;
28     vector<vector<int>> triangle = generate(numRows);
29     printTriangle(triangle);
```

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

=== Code Execution Successful ===

main.cpp



Share

Run

Output

```
1  #include <vector>
2  #include <iostream>
3  #include <algorithm>
4  using namespace std;
5
6  int maxArea(vector<int>& height) {
7      int left = 0, right = height.size() - 1;
8      int max_area = 0;
9      while (left < right) {
10         int width = right - left;
11         max_area = max(max_area, min(height[left], height[right]) * width);
12         if (height[left] < height[right]) {
13             ++left;
14         } else {
15             --right;
16         }
17     }
18     return max_area;
19 }
20
21 int main() {
22     vector<int> height1 = {1, 8, 6, 2, 5, 4, 8, 3, 7};
23     vector<int> height2 = {1, 1};
24     cout << maxArea(height1) << endl; // Output: 49
25     cout << maxArea(height2) << endl; // Output: 1
26     return 0;
27 }
28
```

49

1

=== Code Execution Successful ===

main.cpp



Share

Run

Output

```
14     int hash = carry;
15     for (int r : state) {
16         hash = hash * 31 + r;
17     }
18     static unordered_map<int, int> memo;
19     if (memo.count(hash)) return memo[hash];
20
21     int max_happy = 0;
22     for (int i = 0; i < batchSize; ++i) {
23         if (state[i] > 0) {
24             --state[i];
25             max_happy = max(max_happy, dfs(state, (carry + i) % batchSize)
26                             + (carry == 0 ? 1 : 0));
27             ++state[i];
28         }
29         return memo[hash] = max_happy;
30     };
31
32     return dfs(remainders, 0) + remainders[0];
33 }
34
35 int main() {
36     vector<int> groups1 = {1, 2, 3, 4, 5, 6};
37     vector<int> groups2 = {1, 3, 2, 5, 2, 2, 1, 6};
38     cout << maxHappyGroups(3, groups1) << endl; // Output: 4
39     cout << maxHappyGroups(4, groups2) << endl; // Output: 4
40     return 0;
41 }
```

6
5

=== Code Execution Successful ===

main.cpp



Share

Run

Output

```
1 #include <vector>
2 #include <iostream>
3 #include <algorithm>
4 #include <numeric> // For accumulate
5 #include <functional> // For function
6
7 using namespace std;
8
9 int minimumTimeRequired(vector<int>& jobs, int k) {
10     vector<int> workers(k, 0);
11     int n = jobs.size();
12     sort(jobs.rbegin(), jobs.rend()); // Sort jobs in descending order
13
14     // Helper function using lambda
15     function<bool(int, int)> canDistribute = [&](int idx, int limit) {
16         if (idx == n) return true;
17         for (int i = 0; i < k; ++i) {
18             if (workers[i] + jobs[idx] <= limit) {
19                 workers[i] += jobs[idx];
20                 if (canDistribute(idx + 1, limit)) return true;
21                 workers[i] -= jobs[idx];
22             }
23             if (workers[i] == 0) break; // Prune redundant cases
24         }
25         return false;
26     };
27
28     int left = jobs[0], right = accumulate(jobs.begin(), jobs.end(), 0);
29     while (left < right) {
30         int mid = left + (right - left) / 2;
```

5
15

=== Code Execution Successful ===