# Documentation

## problem Overview:

- The "House Robber II" problem is an extension of the classic "House Robber" problem where houses are arranged in a circle, rather than a straight line. The goal is to determine the maximum amount of money that can be robbed from the houses without triggering the security system, which is activated if two adjacent houses are robbed on the same night. Since the first and last houses in the array are adjacent due to the circular arrangement, this adds complexity to the problem and prevents simply applying the solution for the linear case.

## Approach:

*To solve this problem, the main insight is to break the circular arrangement of houses into two separate linear cases:*

1. Rob houses from the first house to the second-to-last house (i.e., excluding the last house).
2. Rob houses from the second house to the last house (i.e., excluding the first house).

By solving these two linear subproblems independently, the circular adjacency problem is avoided, and the result is simply the maximum amount of money that can be robbed in either case.

## Subproblem (Linear House Robber):

*In both of the above cases (linear subproblems), the problem can be reduced to the classical "House Robber" problem, which is solved using dynamic programming. The idea is that at each house, the robber faces two choices:*

1. **Skip the current house:** Take the maximum amount of money robbed up to the previous house.

2. **Rob the current house:** Add the money from the current house to the maximum amount robbed up to the house before the previous house (because adjacent houses cannot both be robbed).

***The following recurrence relation represents this choice:***

- [ text{dp}[i] = max(text{dp}[i-1], text{dp}[i-2] + text{nums}[i]) ]
- Where dp[i] represents the maximum amount of money that can be robbed up to house i.

***The base cases are simple:***

- If there is only one house, the robber can only rob that house.
- If there are two houses, the robber can rob the house with a higher amount of money.

## Final Solution:

***Once the two linear subproblems are solved:***

- One subproblem calculates the maximum amount of money that can be robbed from house 0 to house n-2 (excluding the last house).
- The other subproblem calculates the maximum amount of money that can be robbed from house 1 to house n-1 (excluding the first house).

The final result is the maximum of these two values.

## Edge Cases:

- **Single House:** If there is only one house in the array, the result is its value.
- **Two Houses:** If there are exactly two houses, the solution will be the house with the maximum money, since robbing both is not allowed.

## Time and Space Complexity:

- **Time Complexity:** The time complexity of this solution is O(n), where n is the number of houses. This is because we solve two linear dynamic programming subproblems, each of which takes linear time.
- **Space Complexity:** The space complexity is O(n) because we store the results of subproblems in a dynamic programming array (dp). However, this can be optimized to O(1) space by only keeping track of the last two results, since the current result depends only on the last two values.

## Steps in Detail:

### 1. Problem Breakdown:

- The circular arrangement introduces a constraint where the first and last houses are adjacent. Therefore, robbing both is not possible.
- To handle this, the problem is divided into two simpler, linear subproblems by excluding either the first house or the last house.

### 2. Dynamic Programming Approach:

- In each subproblem, dynamic programming is used to solve the linear "House Robber" problem.
- *The dp array is initialized to store the maximum money robbed up to each house, with the base cases set as:*
  - ➢ dp[0] = nums[0]
  - ➢ dp[1] = max(nums[0], nums[1])
- *For each subsequent house i, the recurrence relation is applied:*
  - ➢ dp[i] = max(dp[i-1], dp[i-2] + nums[i])
- This ensures that for each house, the maximum amount of money is calculated by either robbing the house or skipping it.

3. **Final Calculation:**

- *After solving both linear subproblems, the final result is the maximum value between the two:*
    - Robbing houses from 0 to n-2.
    - Robbing houses from 1 to n-1.

4. **Edge Cases Handling:**

- If there is only one house, the solution is simply the value of that house, as no adjacent house exists.
- If there are two houses, the solution is the maximum value between the two houses, since robbing both will trigger the alarm.

5. **Efficiency:**

- The solution efficiently computes the maximum money that can be robbed with minimal overhead in both time and space, ensuring optimal performance even for the upper limit of house arrays (100 houses).

## Conclusion:

- This approach breaks down the complexity of the circular arrangement into manageable linear problems, making it easier to apply dynamic programming. By computing the solution in two separate cases (excluding either the first or last house), we can solve the problem without violating the adjacent house constraint. The result is an optimal solution that ensures the maximum amount of money is robbed without triggering the alarm.