House thief

We'll cover the following

There are ‘n’ houses built in a line. A thief wants to steal maximum possible money from these houses. The only restriction the thief has is that he can’t steal from two consecutive houses, as that would alert the security system. How should the thief maximize his stealing?

Problem Statement

Given a number array representing the wealth of ‘n’ houses, determine the maximum amount of money the thief can steal without alerting the security system.

Example 1:

```
Input: {2, 5, 1, 3, 6, 2, 4}
Output: 15
Explanation: The thief should steal from houses 5 + 6 + 4
```

Example 2:

```
Input: {2, 10, 14, 8, 1}
Output: 18
Explanation: The thief should steal from houses 10 + 8
```

Let’s first start with a recursive brute-force solution.

Basic Solution

For every house ‘i’, we have two options:

1. Steal from the current house (‘i’), skip one and steal from (‘i+2’).
2. Skip the current house (‘i’), and steal from the adjacent house (‘i+1’).

The thief should choose the one with the maximum amount from the above two options. So our algorithm will look like:

Java JS Python3 C++

```
def find_max_steal(wealth):
  return find_max_steal_recursive(wealth, 0)


def find_max_steal_recursive(wealth, currentIndex):

  if currentIndex >= len(wealth):
    return 0

  # steal from current house and skip one to steal next
  stealCurrent = wealth[currentIndex] + find_max_steal_recursive(wealth, currentIndex + 2)
  # skip current house to steel from the adjacent house
  skipCurrent = find_max_steal_recursive(wealth, currentIndex + 1)

  return max(stealCurrent, skipCurrent)


def main():

  print(find_max_steal([2, 5, 1, 3, 6, 2, 4]))
  print(find_max_steal([2, 10, 14, 8, 1]))


main()
```

The time complexity of the above algorithm is exponential $O(2^n)$. The space complexity is $O(n)$ which is used to store the recursion stack.

Top-down Dynamic Programming with Memoization

To resolve overlapping subproblems, we can use an array to store the already solved subproblems.

Here is the code:

Java JS Python3 C++

```
def find_max_steal(wealth):
  dp = [0 for x in range(len(wealth))]
  return find_max_steal_recursive(dp, wealth, 0)


def find_max_steal_recursive(dp, wealth, currentIndex):
  if currentIndex >= len(wealth):
    return 0

  if dp[currentIndex] == 0:
    # steal from current house and skip one to steal next
    stealCurrent = wealth[currentIndex] + find_max_steal_recursive(dp, wealth, currentInde
    # skip current house to steel from the adjacent house
    skipCurrent = find_max_steal_recursive(dp, wealth, currentIndex + 1)

    dp[currentIndex] = max(stealCurrent, skipCurrent)

  return dp[currentIndex]


def main():

  print(find_max_steal([2, 5, 1, 3, 6, 2, 4]))
  print(find_max_steal([2, 10, 14, 8, 1]))


main()
```

Bottom-up Dynamic Programming

Let’s try to populate our `dp[]` array from the above solution, working in a bottom-up fashion. As we saw in the above code, every `findMaxStealRecursive()` is the maximum of the two recursive calls; we can use this fact to populate our array.

Code

Here is the code for our bottom-up dynamic programming approach:

Java JS Python3 C++

```
def find_max_steal(wealth):
  n = len(wealth)
  if n == 0:
    return 0
  dp = [0 for x in range(n+1)]  # '+1' to handle the zero house
  dp[0] = 0  # if there are no houses, the thief can't steal anything
  dp[1] = wealth[0]  # only one house, so the thief have to steal from it

  # please note that dp[] has one extra element to handle zero house
  for i in range(1, n):
    dp[i + 1] = max(wealth[i] + dp[i - 1], dp[i])

  return dp[n]


def main():

  print(find_max_steal([2, 5, 1, 3, 6, 2, 4]))
  print(find_max_steal([2, 10, 14, 8, 1]))


main()
```

The above solution has time and space complexity of $O(n)$.

Memory optimization

We can optimize the space used in our previous solution. We don't need to store all the previous numbers up to 'n', as we only need two previous numbers to calculate the next number in the sequence. Let's use this fact to further improve our solution:

Java JS Python3 C++

```python
def find_max_steal(wealth):
  n = len(wealth)
  if n == 0:
    return 0

  n1, n2 = 0, wealth[0]
  for i in range(1, n):
    n1, n2 = n2, max(n1 + wealth[i], n2)

  return n2


def main():

  print(find_max_steal([2, 5, 1, 3, 6, 2, 4]))
  print(find_max_steal([2, 10, 14, 8, 1]))


main()
```

The above solution has a time complexity of $O(n)$ and a constant space complexity $O(1)$.

Fibonacci number pattern

We can clearly see that this problem follows the Fibonacci number pattern. The only difference is that every Fibonacci number is a sum of the two preceding numbers, whereas in this problem every number (total wealth) is the maximum of previous two numbers.

← Back
Minimum jumps with fee

Next →
Longest Palindromic Subsequence

Mark as Completed

Report an Issue

Ask a Question
(https://discuss.educative.io/tag/house-thief__pattern-3-fibonacci-numbers__grokking-dynamic-programming-patterns-for-coding-interviews)