Stickler Thief

Friday, December 24, 2021 1:58 PM

Stickler Thief

Easy Accuracy: 50.32% Submissions: 44989 Points: 2

Stickler the thief wants to loot money from a society having **n** houses in a single line. He is a weird person and follows a certain **rule** when looting the houses. According to the rule, he will **never loot two consecutive houses**. At the same time, he wants to **maximize** the amount he **loots**. The thief knows which house has what amount of money but is unable to come up with an optimal looting strategy. He asks for your help to **find the maximum money he can get** if he strictly **follows** the **rule**. Each house has **a[i]amount of money** present in it.

Example 1:

Input:n = 6a[] = {5,5,10,100,10,5}**Output:** 110**Explanation:** 5+100+5=110

Example 2:

Input:n = 3a[] = {1,2,3}**Output:** 4**Explanation:** 1+3=4

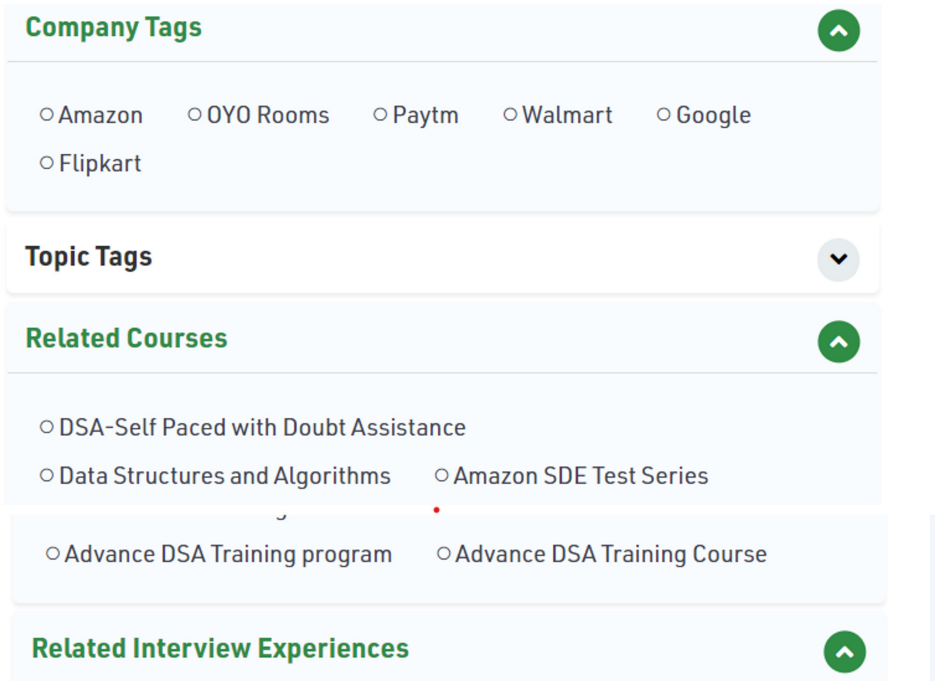
Your Task:

Complete the function**FindMaxSum()**which takes an array **arr[]** and **n** as input which returns the maximum money he can get following the rules

Expected Time Complexity:O(N).

Expected Space Complexity:O(N).

From <https://practice.geeksforgeeks.org/problems/stickler-theif-1587115621/1>



Related Interview Experiences

- Paytm interview experience set 9
- Walmart labs interview experience
- Amazon interview experience set 389 campus full time

Hint 1

1. Create 4 variables, excl_prev, incl_prev, excl_curr, incl_curr.
    - excl_prev = it includes sum upto previous element excluding it.
    - incl_prev = it includes sum upto previous element including it.
    - excl_curr = it includes sum upto curr element excluding it. (excl_curr = 0)
    - incl_curr = it includes sum upto curr element including it. (incl_curr = arr[0])
2. Now, iterate from 1 to n-1.For each iteration:
    - update sum up to current element excluding it as maximum of sum up to previous element excluding and including it. (excl_curr = max(incl_prev, excl_prev))
    - update sum up to current element including it as sum up to previous element excluding it + current element. (incl_curr = excl_prev + arr[i])
    - update sum up to previous element including and excluding it for next iteration. (excl_prev = excl_curr) , (incl_prev = incl_curr)
3. Return the maximum of sum up to current element including and excluding it.

Close

From <https://practice.geeksforgeeks.org/problems/stickler-theif-1587115621/1#>

SOLUTION

Given an array of positive numbers, find the maximum sum of a subsequence with the constraint that no 2 numbers in the sequence should be adjacent in the array. So 3 2 7 10 should return 13 (sum of 3 and 10) or 3 2 5 10 7 should return 15 (sum of 3, 5 and 7).Answer the question in most efficient way.

Examples :

```
Input : arr[] = {5, 5, 10, 100, 10, 5}
Output : 110
```

```
Input : arr[] = {1, 2, 3}
Output : 4
```

```
Input : arr[] = {1, 20, 3}
Output : 20
```

Algorithm:

Loop for all elements in arr[] and maintain two sums incl and excl where incl = Max sum including the previous element and excl = Max sum excluding the previous element.

Max sum excluding the current element will be max(incl, excl) and max sum including the current element will be excl + current element (Note that only excl is considered because elements cannot be adjacent).

At the end of the loop return max of incl and excl.

Example:

```
  arr[] = {5,  5, 10, 40, 50, 35}

incl = 5
excl = 0

For i = 1 (current element is 5)
incl =  (excl + arr[i])  = 5
excl =  max(5, 0) = 5

For i = 2 (current element is 10)
incl =  (excl + arr[i]) = 15
excl =  max(5, 5) = 5

For i = 3 (current element is 40)
incl = (excl + arr[i]) = 45
excl = max(5, 15) = 15

For i = 4 (current element is 50)
incl = (excl + arr[i]) = 65
excl =  max(45, 15) = 45

For i = 5 (current element is 35)
incl =  (excl + arr[i]) = 80
excl =  max(65, 45) = 65

And 35 is the last element. So, answer is max(incl, excl) =  80
```

Thanks to Debanjan for providing code.

Implementation:

```
classMaximumSum
{
/*Function to return max sum such that no two elements
    are adjacent */
intFindMaxSum(intarr[], intn)
{
  intincl=arr[0];
  intexcl=0;
  intexcl_new;
  inti;
  for(i=1; i<n; i++)
  {
    /* current max excluding i */
    excl_new=(incl>excl) ? incl: excl;
    /* current max including i */
    incl=excl+arr[i];
    excl=excl_new;
  }
  /* return max of incl and excl */
  return((incl>excl) ? incl: excl);
}
// Driver program to test above functions
publicstaticvoidmain(String[] args)
{
  MaximumSumsum=newMaximumSum();
  intarr[] =newint[]{5, 5, 10, 100, 10, 5};
```

Run

PythonC#PHPJavascript

Output:

```
110
```

Time Complexity: O(n)

https://www.youtube.com/watch?v=6w60Zi1NtL8

This Problem can also be solved using Dynamic Programming

Time Complexity : O(N)

Space Complexity:O(N)

Refer Find maximum possible stolen value from houses for more explanation.

Now try the same problem for an array with negative numbers also.

Please write comments if you find any bug in the above program/algorithm or other ways to solve

the same problem.

Useful links to help you understand the concepts of this problem:

- https://www.geeksforgeeks.org/maximum-sum-such-that-no-two-elements-are-adjacent/

From <https://practice.geeksforgeeks.org/problems/stickler-theif-1587115621/1#>

Maximum sum such that no two elements are adjacent

- Difficulty Level : Medium
- Last Updated : 30 Nov, 2021

Given an array of positive numbers, find the maximum sum of a subsequence with the constraint that no 2 numbers in the sequence should be adjacent in the array. So 3 2 7 10 should return 13 (sum of 3 and 10) or 3 2 5 10 7 should return 15 (sum of 3, 5 and 7).Answer the question in most efficient way.

Examples :


```
Input : arr[] = {5, 5, 10, 100, 10, 5}
Output : 110
Input : arr[] = {1, 2, 3}
Output : 4

Input : arr[] = {1, 20, 3}
Output : 20
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Algorithm:

Loop for all elements in arr[] and maintain two sums incl and excl where incl = Max sum including the previous element and excl = Max sum excluding the previous element.

Max sum excluding the current element will be max(incl, excl) and max sum including the current element will be excl + current element (Note that only excl is considered because elements cannot be adjacent).

At the end of the loop return max of incl and excl.

Example:

```
  arr[] = {5,  5, 10, 40, 50, 35}
incl = 5
  excl = 0

For i = 1 (current element is 5)
  incl =  (excl + arr[i])  = 5
  excl =  max(5, 0) = 5
```

```
For i = 2 (current element is 10)
  incl =  (excl + arr[i]) = 15
  excl =  max(5, 5) = 5

For i = 3 (current element is 40)
  incl = (excl + arr[i]) = 45
  excl = max(5, 15) = 15

For i = 4 (current element is 50)
  incl = (excl + arr[i]) = 65
  excl =  max(45, 15) = 45

For i = 5 (current element is 35)
  incl =  (excl + arr[i]) = 80
  excl =  max(65, 45) = 65

And 35 is the last element. So, answer is max(incl, excl) =  80
```

From <https://www.geeksforgeeks.org/maximum-sum-such-that-no-two-elements-are-adjacent/>

EDITORIAL 1

Find maximum possible stolen value from houses

- Difficulty Level : Medium
- Last Updated : 03 May, 2021

There are n houses build in a line, each of which contains some value in it. A thief is going to steal the maximal value of these houses, but he can't steal in two adjacent houses because the owner of the stolen houses will tell his two neighbors left and right side. What is the maximum stolen value?

Examples:

```
Input: hval[] = {6, 7, 1, 3, 8, 2, 4}
Output: 19
Explanation: The thief will steal 6, 1, 8 and 4 from the house.

Input: hval[] = {5, 3, 4, 11, 2}
Output: 16

Explanation: Thief will steal 5 and 11
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Naive Approach: Given an array, the solution is to find the maximum sum subsequence where no two selected elements are adjacent. So the approach to the problem is a recursive solution. So there are two cases.

1. If an element is selected then the next element cannot be selected.
2. if an element is not selected then the next element can be selected.

So the recursive solution can easily be devised. The sub-problems can be stored thus reducing the complexity and converting the recursive solution to a Dynamic programming problem.

Algorithm:

1. Create an extra space *dp*, DP array to store the sub-problems.
2. Tackle some basic cases, if the length of the array is 0, print 0, if the length of the array is 1, print the first element, if the length of the array is 2, print maximum of two elements.
3. Update *dp[0]* as *array[0]* and *dp[1]* as maximum of *array[0]* and *array[1]*
4. Traverse the array from the second element (2nd index) to the end of array.
5. For every index, update *dp[i]* as maximum of *dp[i-2] + array[i]* and *dp[i-1]*, this step defines the two cases, if an element is selected then the previous element cannot be selected and if an element is not selected then the previous element can be selected.
6. Print the value *dp[n-1]*

Implementation:

- C++
- Java
- Python
- C#
- PHP
- Javascript

```java
// Java program to find the maximum stolen value
import java.io.*;

class GFG
{
    // Function to calculate the maximum stolen value
    static int maxLoot(int hval[], int n)
    {
        if (n == 0)
        return 0;
        if (n == 1)
            return hval[0];
        if (n == 2)
            return Math.max(hval[0], hval[1]);

        // dp[i] represent the maximum value stolen
        // so far after reaching house i.
        int[] dp = new int[n];

        // Initialize the dp[0] and dp[1]
        dp[0] = hval[0];
        dp[1] = Math.max(hval[0], hval[1]);

        // Fill remaining positions
        for (int i = 2; i<n; i++)
            dp[i] = Math.max(hval[i]+dp[i-2], dp[i-1]);

        return dp[n-1];
    }

    // Driver program
    public static void main (String[] args)
    {
        int hval[] = {6, 7, 1, 3, 8, 2, 4};
        int n = hval.length;
        System.out.println("Maximum loot value : " + maxLoot(hval, n));
    }
}

// Contributed by Pramod Kumar
```

Output:

```
Maximum loot value : 19
```

Complexity Analysis:

- **Time Complexity:**

.
Only one traversal of original array is needed. So the time complexity is O(n)

- **Space Complexity:**
  .
  An array is required of size n, so space complexity is O(n).
  Efficient Approach: By carefully observing the DP array, it can be seen that the values of previous two indices matter while calculating the value for an index. To replace the total DP array by two variables.

  Algorithm:

1. Tackle some basic cases, if the length of the array is 0, print 0, if the length of the array is 1, print the first element, if the length of the array is 2, print maximum of two elements.
2. Create two variables *value1* and *value2 value1* as *array[0]* and *value2* as maximum of *array[0]* and *array[1]* and a variable *max_val* to store the answer
3. Traverse the array from the second element (2nd index) to the end of array.
4. For every index, update *max_val* as maximum of *value1 + array[i]* and *value2*, this step defines the two cases, if an element is selected then the previous element cannot be selected and if an element is not selected then the previous element can be selected.
5. For every index, Update *value1 = value2* and *value2 = max_val*
6. Print the value of *max_val*
   Implementation:

- C++
- Java
- Python
- C#
- PHP
- Javascript

```
// Java program to find the maximum stolen value
import java.io.*;

class GFG
{
    // Function to calculate the maximum stolen value
    static int maxLoot(int hval[], int n)
    {
        if (n == 0)
        return 0;

        int value1 = hval[0];
        if (n == 1)
            return value1;

        int value2 = Math.max(hval[0], hval[1]);
        if (n == 2)
            return value2;

        // contains maximum stolen value at the end
        int max_val = 0;

        // Fill remaining positions
        for (int i=2; i<n; i++)
        {
            max_val = Math.max(hval[i]+value1, value2);
            value1 = value2;
            value2 = max_val;
        }

        return max_val;
    }

    // driver program
    public static void main (String[] args)
```

```
    {
        int hval[] = {6, 7, 1, 3, 8, 2, 4};
        int n = hval.length;
        System.out.println("Maximum loot value : " + maxLoot(hval, n));
    }
}

// Contributed by Pramod kumar
```

Output:

```
Maximum loot value : 19
```

Complexity Analysis:

- **Time Complexity:**
  , Only one traversal of original array is needed. So the time complexity is O(n).
- **Auxiliary Space:**
  , No extra space is required so the space complexity is constant.
  This article is contributed by **Atul Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

  Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

From <https://www.cdn.geeksforgeeks.org/find-maximum-possible-stolen-value-houses/>

Maximum sum such that no two elements are adjacent

- Difficulty Level : Medium
- Last Updated : 30 Nov, 2021

Given an array of positive numbers, find the maximum sum of a subsequence with the constraint that no 2 numbers in the sequence should be adjacent in the array. So 3 2 7 10 should return 13 (sum of 3 and 10) or 3 2 5 10 7 should return 15 (sum of 3, 5 and 7).Answer the question in most efficient way.

Examples :


```
Input : arr[] = {5, 5, 10, 100, 10, 5}
Output : 110
Input : arr[] = {1, 2, 3}
Output : 4

Input : arr[] = {1, 20, 3}
Output : 20
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Algorithm:

Loop for all elements in arr[] and maintain two sums incl and excl where incl = Max sum including the previous element and excl = Max sum excluding the previous element.

Max sum excluding the current element will be max(incl, excl) and max sum including the current element will be excl + current element (Note that only excl is considered because elements cannot be adjacent).

At the end of the loop return max of incl and excl.

Example:

```
  arr[] = {5,  5, 10, 40, 50, 35}
incl = 5
  excl = 0

For i = 1 (current element is 5)
  incl =  (excl + arr[i])  = 5
  excl =  max(5, 0) = 5

For i = 2 (current element is 10)
  incl =  (excl + arr[i]) = 15
  excl =  max(5, 5) = 5

For i = 3 (current element is 40)
  incl = (excl + arr[i]) = 45
  excl = max(5, 15) = 15

For i = 4 (current element is 50)
  incl = (excl + arr[i]) = 65
  excl =  max(45, 15) = 45

For i = 5 (current element is 35)
  incl =  (excl + arr[i]) = 80
  excl =  max(65, 45) = 65

And 35 is the last element. So, answer is max(incl, excl) =  80
```

From <https://www.geeksforgeeks.org/maximum-sum-such-that-no-two-elements-are-adjacent/>

```
class MaximumSum
{
    /*Function to return max sum such that no two elements
      are adjacent */
    int FindMaxSum(int arr[], int n)
    {
        int incl = arr[0];
        int excl = 0;
        int excl_new;
        int i;

        for (i = 1; i < n; i++)
        {
            /* current max excluding i */
            excl_new = (incl > excl) ? incl : excl;

            /* current max including i */
            incl = excl + arr[i];
            excl = excl_new;
        }

        /* return max of incl and excl */
        return ((incl > excl) ? incl : excl);
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        MaximumSum sum = new MaximumSum();
        int arr[] = new int[]{5, 5, 10, 100, 10, 5};
        System.out.println(sum.FindMaxSum(arr, arr.length));
    }
}

// This code has been contributed by Mayank Jaiswal
```

From <https://www.geeksforgeeks.org/maximum-sum-such-that-no-two-elements-are-adjacent/>