Dash

All

Articles

Videos

Problems

Quiz

« Prev

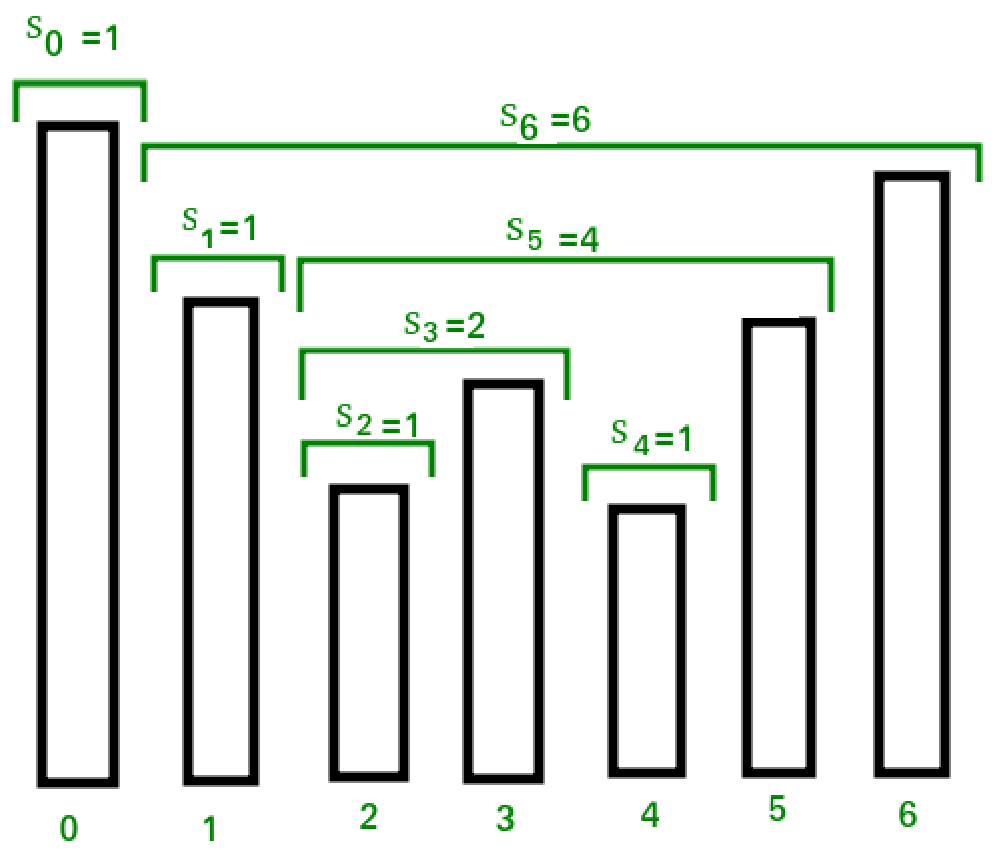
Next »

Stock span problem

The stock span problem is a financial problem where we have a series of **N** daily price quotes for a stock and we need to calculate the span of the stock's price for all N days. The span Si of the stock's price on a given day i is defined as the maximum number of consecutive days just before the given day, for which the price of the stock on the current day is less than its price on the given day.

Examples:

> **Input:** N = 7, price[] = [100 80 60 70 60 75 85]
> **Output:** 1 1 1 2 1 4 6
> **Explanation:** Traversing the given input span for 100 will be 1, 80 is smaller than 100 so the span is 1, 60 is smaller than 80 so the span is 1, 70 is greater than 60 so the span is 2 and so on. Hence the output will be 1 1 1 2 1 4 6.
>
> **Input:** N = 6, price[] = [10 4 5 90 120 80]
> **Output:**1 1 2 4 5 1
> **Explanation:** Traversing the given input span for 10 will be 1, 4 is smaller than 10 so the span will be 1, 5 is greater than 4 so the span will be 2 and so on. Hence, the output will be 1 1 2 4 5 1.



Naive Approach: To solve the problem follow the below idea:

> Traverse the input price array. For every element being visited, traverse elements on the left of it and increment the span value of it while elements on the left side are smaller

Below is the implementation of the above approach:

C++ Java

Dash
All
Articles
Videos
Problems
Quiz

```
 // Java implementation for brute force method to calculate
// stock span values

import java.util.Arrays;

class GFG {
    // method to calculate stock span values
    static void calculateSpan(int price[], int n, int S[])
    {
        // Span value of first day is always 1
        S[0] = 1;

        // Calculate span value of remaining days by
        // linearly checking previous days
        for (int i = 1; i < n; i++) {
            S[i] = 1; // Initialize span value

            // Traverse left while the next element on left
            // is smaller than price[i]
            for (int j = i - 1;
                 (j >= 0) && (price[i] >= price[j]); j--)
                S[i]++;
        }
    }

    // A utility function to print elements of array
    static void printArray(int arr[])
```

```
        {
            System.out.print(Arrays.toString(arr));
        }

        // Driver code
        public static void main(String[] args)
        {
            int price[] = { 10, 4, 5, 90, 120, 80 };
            int n = price.length;
            int S[] = new int[n];

            // Fill the span values in array S[]
            calculateSpan(price, n, S);

            // print the calculated span values
            printArray(S);
        }
    }
```

Output

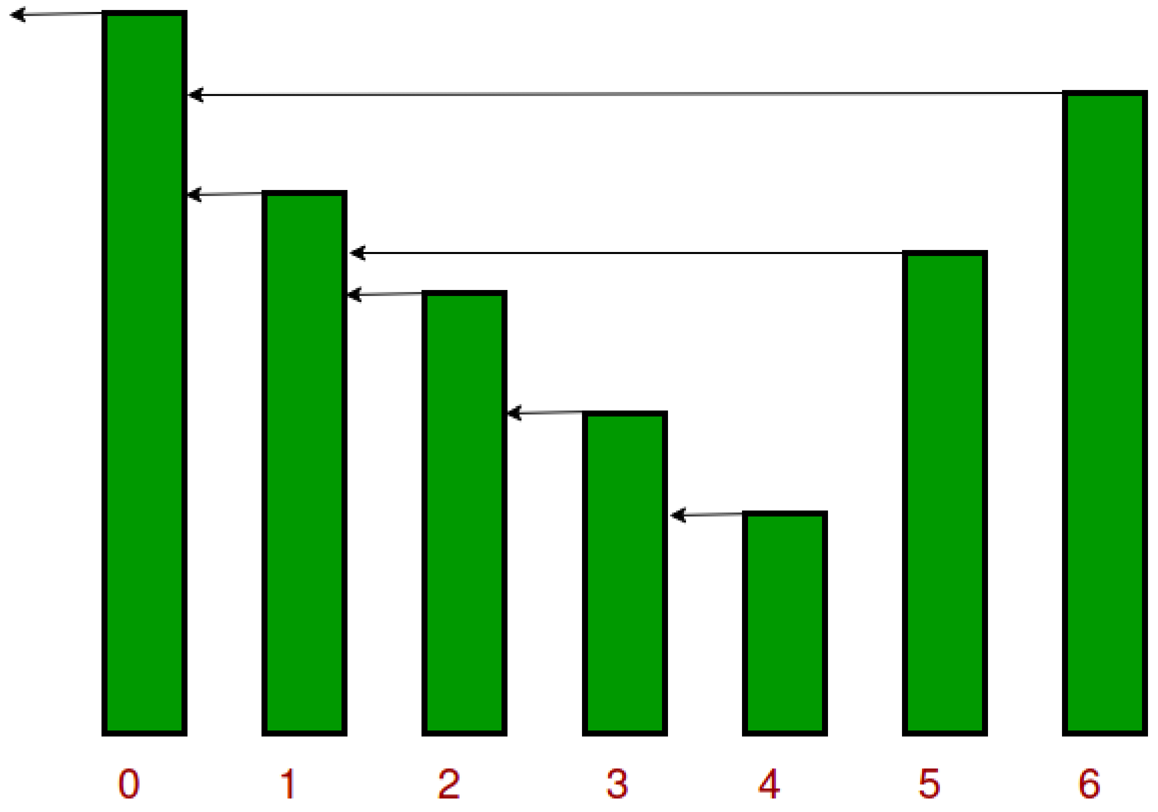
```
1 1 2 4 5 1
```

Time Complexity: O(N^2)
Auxiliary Space: O(N)

The stock span problem using stack:

To solve the problem follow the below idea:

We see that S[i] on the day **i** can be easily computed if we know the closest day preceding **i**, such that the price is greater than on that day than the price on the day i. If such a day exists, let's call it **h(i)**, otherwise, we define **h(i) = -1**
The span is now computed as **S[i] = i - h(i)**. See the following diagram



To implement this logic, we use a stack as an abstract data type to store the days i, h(i), h(h(i)), and so on. When we go from day i-1 to i, we pop the days when the price of the stock was less than or equal to price[i] and then push the value of day i back into the stack.

Dash

All

Articles

Videos

Problems

Quiz

Follow the below steps to solve the problem:

- Create a stack of type int and push 0 in it
- Set the answer of day 1 as 1 and run a for loop to traverse the days
- While the stack is not empty and the price of st.top is less than or equal to the price of the current day, pop out the top value
- Set the answer of the current day as i+1 if the stack is empty else equal to i - st.top
- Push the current day into the stack
- Print the answer using the answer array

Below is the implementation of the above approach:

Note: We have to check also for a case when all the stock prices should be the same so therefore we have to just check whether the current stock price is bigger than the previous one or not. We will not pop from the stack when the current and previous stock prices are the same.

C++ Java

```
 // Java linear time solution for stock span problem

import java.util.ArrayDeque;
import java.util.Arrays;
import java.util.Deque;

public class GFG {
    // A stack based efficient method to calculate
    // stock span values
    static void calculateSpan(int price[], int n, int S[])
    {
        // Create a stack and push index of first element
```

Dash
All
Articles
Videos
Problems
Quiz

```
// to it
Deque<Integer> st = new ArrayDeque<Integer>();
// Stack<Integer> st = new Stack<>();
st.push(0);

// Span value of first element is always 1
S[0] = 1;

// Calculate span values for rest of the elements
for (int i = 1; i < n; i++) {

    // Pop elements from stack while stack is not
    // empty and top of stack is smaller than
    // price[i]
    while (!st.isEmpty()
           && price[st.peek()] <= price[i])
        st.pop();

    // If stack becomes empty, then price[i] is
    // greater than all elements on left of it,
    // i.e., price[0], price[1], ..price[i-1]. Else
    // price[i] is greater than elements after top
    // of stack
    S[i] = (st.isEmpty()) ? (i + 1)
                          : (i - st.peek());

    // Push this element to stack
    st.push(i);
}
```

```
    }

    // A utility function to print elements of array
    static void printArray(int arr[])
    {
        System.out.print(Arrays.toString(arr));
    }

    // Driver code
    public static void main(String[] args)
    {
        int price[] = { 10, 4, 5, 90, 120, 80 };
```

```
        // Fill the span values in array S[]
        calculateSpan(price, n, S);

        // print the calculated span values
        printArray(S);
    }
}
```

Output

```
1 1 2 4 5 1
```

Time Complexity: O(N). It seems more than O(N) at first look. If we take a closer look, we can observe that every element of the array is added and removed from the stack at most once.

Auxiliary Space: O(N) in the worst case when all elements are sorted in decreasing order.

Dash
All
Articles
Videos
Problems
Quiz

The stock span problem using dynamic programming:

To solve the problem follow the below idea:

> Since the same subproblems are called again, this problem has the Overlapping Subproblems property. S0, the recomputations of the same subproblems can be avoided by constructing a temporary array to store already calculated answers

- Store the answer for every index in an array and calculate the value of the next index using previous values
- i.e check the answer for previous element and if the value of the current element is greater than the previous element
- Add the answer to the previous index into current answer and then check the value of (previous index - answer of previous index)
- Check this condition repeatedly

Below is the implementation of the above approach:

C++ Java

```
// Java program for a linear time
// solution for stock span problem
// without using stack

import java.io.*;

class GFG {
```

Dash
All
Articles
Videos
Problems
Quiz

```
// An efficient method to calculate
// stock span values implementing the
// same idea without using stack
static void calculateSpan(int A[], int n, int ans[])
{
    // Span value of first element is always 1
    ans[0] = 1;

    // Calculate span values for rest of the elements
    for (int i = 1; i < n; i++) {
        int counter = 1;
        while ((i - counter) >= 0
               && A[i] >= A[i - counter]) {
            counter += ans[i - counter];
        }
        ans[i] = counter;
    }
}

// A utility function to print elements of array
static void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}

// Driver code
public static void main(String[] args)
```

```
    {
        int price[] = { 10, 4, 5, 90, 120, 80 };
        int n = price.length;
        int S[] = new int[n];

        // Fill the span values in array S[]
        calculateSpan(price, n, S);

        // print the calculated span values
        printArray(S, n);
    }
}
```

Output

```
1 1 2 4 5 1
```

Time Complexity: O(N)
Auxiliary Space: O(N)

The stock span problem using two stacks:

Follow the below steps to solve the problem:

- In this approach, I have used the data structure stack to implement this task.
- Here, two stacks are used. One stack stores the actual stock prices whereas, the other stack is a temporary stack.
- The stock span problem is solved using only the Push and Pop functions of the Stack.
- Just to take input values, I have taken array 'price' and to store output, used array 'span'.

Below is the implementation of the above approach:

C++ Java

```
 // C++ program for brute force method
// to calculate stock span values
#include <bits/stdc++.h>
using namespace std;

vector<int> calculateSpan(int arr[], int n)
{
    // Your code here
    stack<int> s;
    vector<int> ans;
    for (int i = 0; i < n; i++) {
        while (!s.empty() and arr[s.top()] <= arr[i])
            s.pop();

        if (s.empty())
            ans.push_back(i + 1);
        else {
            int top = s.top();
            ans.push_back(i - top);
        }
        s.push(i);
    }

    return ans;
}
```

Dash
All
Articles
Videos
Problems
Quiz

```
// A utility function to print elements of array
void printArray(vector<int> arr)
{
    for (int i = 0; i < arr.size(); i++)
        cout << arr[i] << " ";
}

// Driver code
int main()
{
    int price[] = { 10, 4, 5, 90, 120, 80 };
    int n = sizeof(price) / sizeof(price[0]);
    int S[n];

      // Function call
    vector<int> arr = calculateSpan(price, n);
    printArray(arr);

    return 0;
}
```

Output

```
1 1 2 4 5 1
```

Time Complexity: O(N), where N is the size of the array.
Auxiliary Space: O(N), where N is the size of the array.

Mark as Read

Dash

All

Articles

Videos

Problems

Quiz