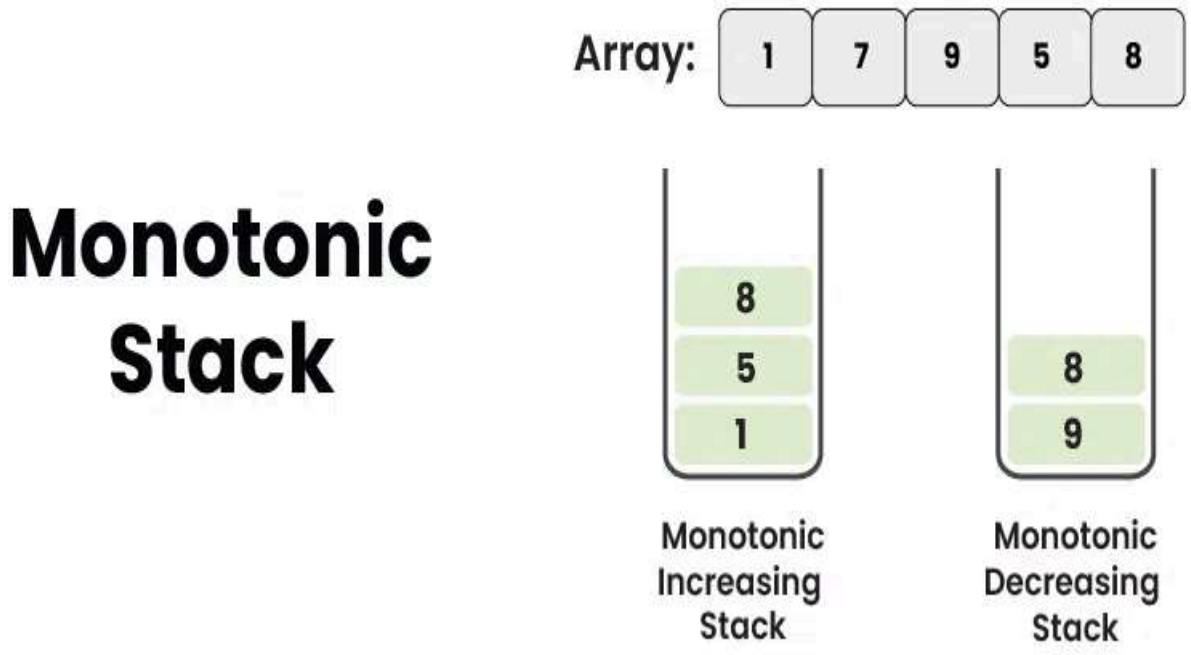
Introduction to Monotonic Stack – Data Structure and Algorithm Tutorials

Last Updated : 18 Apr, 2024

A **monotonic stack** is a special data structure used in algorithmic problem-solving. Monotonic Stack

DSA Data Structures Array String Linked List Stack Queue Tree Binary Tree Binary Search Tree Heap Hashing Graph Tr

efficiently solve problems such as finding the next greater or smaller element in an array etc.



Monotonic Stack

Table of Content

What is Monotonic Stack?

A **Monotonic Stack** is a common data structure in computer science that maintains its elements in a specific order. Unlike traditional stacks, Monotonic Stacks ensure that elements inside the stack are arranged in an **increasing** or **decreasing** order based on their arrival time. In order to achieve the monotonic stacks, we have to enforce the **push** and **pop** operation depending on whether we want a monotonic increasing stack or monotonic decreasing stack.

Let's understand the term **Monotonic Stacks** by breaking it down.

Monotonic: It is a word for mathematics functions. A function **y = f(x)** is monotonically increasing or decreasing when it follows the below conditions:

- As **x** increases, **y** also increases always, then it's a **monotonically increasing function**.
- As **x** increases, **y** decreases always, then it's a **monotonically decreasing function**.

See the below examples:

- **y = 2x +5**, it's a **monotonically increasing function**.
- **y = -(2^x)**, it's a **monotonically decreasing function**.

Similarly, A stack is called a **monotonic stack** if all the elements starting from the bottom of the stack is either in **increasing** or in **decreasing order**.

Types of Monotonic Stack:

Monotonic Stacks can be broadly classified into two types:

1. Monotonic Increasing Stack
2. Monotonic Decreasing Stack

Monotonic Increasing Stack:

A **Monotonically Increasing Stack** is a stack where elements are placed in **increasing order** from the **bottom** to the **top**. Each new element added to the stack is greater than or equal to the one below it. If a new element is smaller, we remove elements from the top of the stack until we find one that is smaller or equal to the new element, or until the stack is empty. This ensures that the stack always stays in increasing order.

Example: 1, 3, 10, 15, 17

How to achieve Monotonic Increasing Stack?

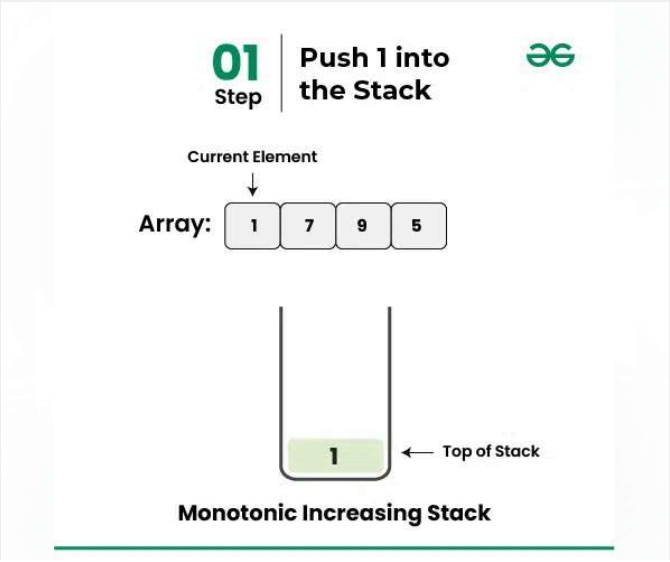
To achieve a monotonic increasing stack, you would typically push elements onto the stack while ensuring that the stack maintains a increasing order from bottom to top. When pushing a new

element, you would pop elements from the stack that are smaller than the new element until the stack maintains the desired monotonic increasing property.

To achieve a monotonic increasing stack, you can follow these step-by-step approaches:

- *Initialize an empty stack.*
- *Iterate through the elements and for each element:*
  - *while stack is not empty AND top of stack is more than the current element*
    - *pop element from the stack*
  - *Push the current element onto the stack.*
- *At the end of the iteration, the stack will contain the monotonic increasing order of elements.*

Let's illustrate the example for a monotonic increasing stack using the array **Arr[] = {1, 7, 9, 5}**:



Below is the implementation of the above approach:

C++ JavaScript

```cpp
#include <iostream>
#include <stack>
#include <vector>

using namespace std;

// Function to implement monotonic increasing stack
vector<int> monotonicIncreasing(vector<int>& nums)
{
    int n = nums.size();
    stack<int> st;
    vector<int> result;

    // Traverse the array
    for (int i = 0; i < n; ++i) {

        // While stack is not empty AND top of stack is more
        // than the current element
        while (!st.empty() && st.top() > nums[i]) {

            // Pop the top element from the
            // stack
            st.pop();
        }

        // Push the current element into the stack
        st.push(nums[i]);
    }

    // Construct the result array from the stack
    while (!st.empty()) {
        result.insert(result.begin(), st.top());
```

```
            st.pop();
        }

        return result;
    }

    int main() {
        // Example usage:
        vector<int> nums = {3, 1, 4, 1, 5, 9, 2, 6};
        vector<int> result = monotonicIncreasing(nums);
        cout << "Monotonic increasing stack: ";
        for (int num : result) {
            cout << num << " ";
        }
        cout << endl;

        return 0;
    }
```

Output

```
Monotonic increasing stack: 1 1 2 6
```

Complexity Analysis:

- **Time Complexity:** O(N), each element from the input array is pushed and popped from the stack exactly once. Therefore, even though there is a loop inside a loop, no element is processed more than twice.
- **Auxiliary Space:** O(N)

Monotonic Decreasing Stack:

A **Monotonically Decreasing Stack** is a stack where elements are placed in **decreasing order** from the **bottom** to the **top**. Each new element added to the stack must be smaller than or equal to the one below it. If a new element is greater than top of stack then we remove elements from the top of the stack until we find one that is greater or equal to the new element, or until the stack is empty. This ensures that the stack always stays in decreasing order.

Example: 17, 14, 10, 5, 1

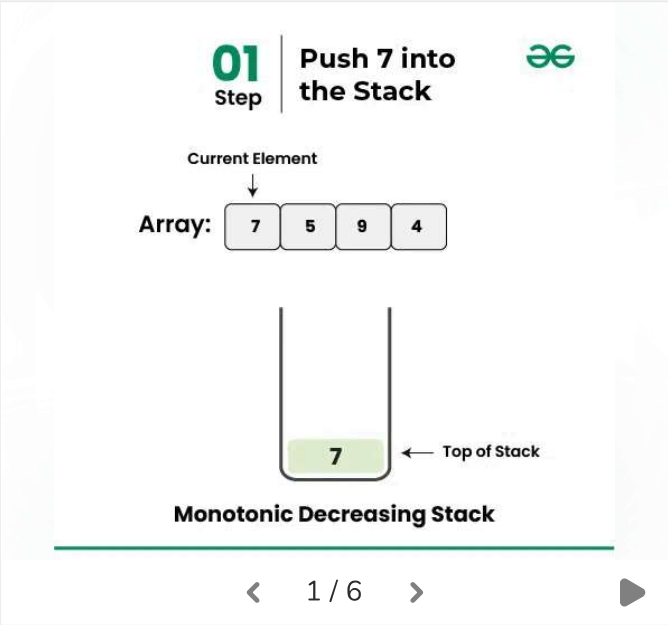
How to achieve Monotonic Decreasing Stack?

To achieve a **monotonic decreasing stack**, you would typically push elements onto the stack while ensuring that the stack maintains a decreasing order from bottom to top. When pushing a new element, you would pop elements from the stack that are greater than the new element until the stack maintains the desired monotonic decreasing property.

To achieve a monotonic decreasing stack, you can follow these step-by-step approaches:

- *Start with an empty stack.*
- *Iterate through the elements of the input array.*
  - *While stack is not empty AND top of stack is less than the current element:*
    - *pop element from the stack*
  - *Push the current element onto the stack.*
- *After iterating through all the elements, the stack will contain the elements in monotonic decreasing order.*

Let's illustrate the example for a monotonic decreasing stack using the array **Arr[] = {7, 5, 9, 4}**:



Below is the implementation of the above approach:

C++

```cpp
#include <iostream>
#include <stack>
#include <vector>

using namespace std;

// Function to implement monotonic decreasing stack
vector<int> monotonicDecreasing(vector<int>& nums)
{
    int n = nums.size();
    stack<int> st;

    // Traverse the array
    for (int i = 0; i < n; ++i) {
```

```
            // While stack is not empty AND top of stack is less
            // than the current element
            while (!st.empty() && st.top() < nums[i]) {

                // Pop top element from the
                // stack
                st.pop();
            }

            // Push the current element into the stack
            st.push(nums[i]);
        }
    }
```

Complexity Analysis:

- **Time Complexity:** O(N), each element is processed only twice, once for the push operation and once for the pop operation.
- **Auxiliary Space:** O(N)

Practice Problem on Monotonic Stack:

Problem
Next Greater Element (NGE) for every element in given Array
Next Smaller Element

Problem
Find next Smaller of next Greater in an array
Next Greater Frequency Element
Largest Rectangular Area in a Histogram using Stack
Check for Balanced Brackets in an expression (well-formedness)
Maximum size rectangle binary sub-matrix with all 1s
Expression Evaluation
The Stock Span Problem
Expression contains redundant bracket or not
Find the nearest smaller numbers on left side in an array
Find maximum of minimum for every window size in a given array

Problem
Minimum number of bracket reversals needed to make an expression balanced
Tracking current Maximum Element in a Stack
Lexicographically largest subsequence containing all distinct characters only once
Sum of maximum elements of all possible sub-arrays of an array

Applications of Monotonic Stack :

Here are some applications of monotonic stacks:

- **Finding Next Greater Element:** Monotonic stacks are often used to find the next greater element for each element in an array. This is a common problem in competitive programming and has applications in various algorithms.
- **Finding Previous Greater Element:** Similarly, monotonic stacks can be used to find the previous greater element for each element in an array.
- **Finding Maximum Area Histogram:** Monotonic stacks can be applied to find the maximum area of a histogram. This problem involves finding the largest rectangular area possible in a given histogram.
- **Finding Maximum Area in Binary Matrix:** Monotonic stacks can also be used to find the maximum area of a rectangle in a binary matrix. This is a variation of the maximum area

histogram problem.

- **Finding Sliding Window Maximum/Minimum:** Monotonic stacks can be used to efficiently find the maximum or minimum elements within a sliding window of a given array.
- **Expression Evaluation:** Monotonic stacks can be used to evaluate expressions involving parentheses, such as checking for balanced parentheses or evaluating the value of an arithmetic expression.

Advantages of Monotonic Stack:

- Efficient for finding the next greater or smaller element in an array.
- Useful for solving a variety of problems, such as finding the nearest smaller element or calculating the maximum area of histograms.
- In many cases, the time complexity of algorithms using monotonic stacks is linear, making them efficient for processing large datasets.

Disadvantages of Monotonic Stack:

- Requires extra space to store the stack.
- May not be intuitive for beginners to understand and implement.

Frequently Asked Questions (FAQs) on Monotonic Stack:

1. What is a monotonic stack?

A monotonic stack is a data structure that maintains either non-increasing or non-decreasing

order of elements.

2. How is a monotonic stack different from a regular stack?

Unlike a regular stack, a monotonic stack ensures that the elements are in a specific order, either non-increasing or non-decreasing.

3. What are the typical use cases for a monotonic stack?

Monotonic stacks are commonly used in problems involving finding the next greater element, next smaller element, or solving various algorithmic problems efficiently.

4. Can you provide an example of solving a problem using a monotonic stack?

One common example is finding the next greater element to the right for each element in an array using a monotonic stack.

5. Are there any common pitfalls or challenges when using a monotonic stack?

A common challenge is understanding when to use a monotonic stack and ensuring that the stack's ordering is maintained correctly throughout the algorithm.

Related Article:

- How to Identify and Solve the Monotonic Stack Problems

80

Suggest improvement

Previous

Implement a stack using singly linked list

Next

Difference between Stack and Queue Data Structures

Share your thoughts in the comments

Add Your Comment

Similar Reads

How to Identify and Solve Monotonic Stack Problems ?

Introduction to Monotonic Queues

Find element position in given monotonic sequence

Check if given Array is Monotonic

Check if level-wise Decimal equivalent of Binary Tree forms a Monotonic sequence or not

Monotonic shortest path from source to destination in Directed Weighted Graph

Python Program to check if given array is Monotonic

Static Data Structure vs Dynamic Data Structure

Design and Implement Special Stack Data Structure | Added Space Optimized Version

Stack Vs Heap Data Structure

rbkraj000

Article Tags : Technical Scripter 2022 , Tutorials , DSA , Stack , Technical Scripter

Practice Tags : Stack

GeeksforGeeks
Sanchhaya Education Private Limited
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305