How to efficiently implement k stacks in a single array?

Last Updated : 24 May, 2023

We have discussed space-efficient implementation of 2 stacks in a single array. In this post, a general solution for k stacks is discussed. Following is the detailed problem statement. *Create a data structure kStacks that represents k stacks. Implementation of kStacks should use only one array, i.e., k stacks should use the same array for storing elements.*

The following functions must be supported by k Stacks. push(int x, int sn) –> pushes x to stack number 'sn' where sn is from 0 to k-1 pop(int sn) –> pops an element from stack number 'sn' where sn is from 0 to k-1

Method 1 (Divide the array in slots of size n/k) A simple way to implement k stacks is to divide the array in k slots of size n/k each, and fix the slots for different stacks, i.e., use arr[0] ... arr[n/k-1] for first

Easy problems on Stack

stack, and arr[n/k] to arr[2n/k-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be n. The problem with this method is inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the k is 2 and array size (n) is 6 and we push 3 elements to first and do not push anything to second stack. When we push 4th element to first, there will be overflow even if we have space for 3 more elements in array.

Method 2 (A space-efficient implementation) The idea is to use two extra arrays for efficient implementation of k stacks in an array. This may not make much sense for integer stacks, but stack items can be large for example stacks of employees, students, etc where every item is of hundreds of bytes. For such large stacks, the extra space used is comparatively very less as we use two *integer* arrays as extra space.

Following are the two extra arrays are used:

1) top[]: This is of size k and stores indexes of top elements in all stacks.

2) next[]: This is of size n and stores indexes of next item for the items in array arr[].

Here arr[] is actual array that stores k stacks. Together with k stacks, a stack of free slots in arr[] is also maintained. The top of this stack is stored in a variable ‘free’. All entries in top[] are initialized as -1

to indicate that all stacks are empty. All entries next[i] are initialized as i+1 because all slots are free initially and pointing to next slot. Top of free stack, 'free' is initialized as 0.

Algorithm:

1. *Initialize an array of size k to keep track of the top element of each stack.*
2. *Initialize an array next of size n, where n is the total size of the array that will hold the k stacks. Set the value of next[i] to i+1 for all 0 ? i < n-1, and next[n-1] to -1. This array will be used to keep track of the next element in the stack.*
3. *Initialize an array top of size k to store the index of the top element of each stack. Set the value of top[i] to -1 for all 0 ? i < k.*
4. *To push an element onto the i-th stack, do the following:*
   - *Check if the array is full by checking if next[0] is -1. If it is, return an error message indicating that the stack is full.*
   - *Set the value of next[0] to top[i].*
   - *Set the value of top[i] to 0.*

- *Set the value of next[top[i]] to the new element's index.*
- *Increment the value of top[i] by the block size.*

5. *To pop an element from the i-th stack, do the following:*
   - *Check if the stack is empty by checking if top[i] is -1. If it is, return an error message indicating that the stack is empty.*
   - *Decrement the value of top[i] by the block size.*
   - *Set the value of next[top[i]] to -1.*
   - *Return the element at the index top[i] + block size – 1.*

Following is the implementation of the above idea.

C++

```
// A C++ program to demonstrate implementation of
// array in time and space efficient way
#include<bits/stdc++.h>
using namespace std;

// A C++ class to represent k stacks in a single
```

```
class kStacks
{
    int *arr;   // Array of size n to store actua
    int *top;   // Array of size k to store index
    int *next;  // Array of size n to store next
                // and free list
    int n, k;
    int free; // To store beginning index of free
public:
    //constructor to create k stacks in an array
    kStacks(int k, int n);

    // A utility function to check if there is sp
    bool isFull()   {  return (free == -1);  }

    // To push an item in stack number 'sn' where
    void push(int item, int sn);

    // To pop an from stack number 'sn' where sn
    int pop(int sn);

    // To check whether stack number 'sn' is empt
    bool isEmpty(int sn)  {  return (top[sn] ==
};

//constructor to create k stacks in an array of s
kStacks::kStacks(int k1, int n1)
{
    // Initialize n and k, and allocate memory fo
    k = k1, n = n1;
    arr = new int[n];
    top = new int[k];
    next = new int[n];

    // Initialize all s
```

```
    for (int i = 0; i < k; i++)
        top[i] = -1;

    // Initialize all spaces as free
    free = 0;
    for (int i=0; i<n-1; i++)
        next[i] = i+1;
    next[n-1] = -1;  // -1 is used to indicate er
}

// To push an item in stack number 'sn' where sn
void kStacks::push(int item, int sn)
{
    // Overflow check
    if (isFull())
    {
        cout << "\nStack Overflow\n";
        return;
    }

    int i = free;       // Store index of first fr

    // Update index of free slot to index of next
    free = next[i];

    // Update next of top and then top for stack
    next[i] = top[sn];
    top[sn] = i;

    // Put the item in array
    arr[i] = item;
}

// To pop an element from stack number 'sn' where
int kStacks::pop(int sn
```

```
{
    // Underflow check
    if (isEmpty(sn))
    {
         cout << "\nStack Underflow\n";
         return INT_MAX;
    }


    // Find index of top item in stack number 'sr
    int i = top[sn];

    top[sn] = next[i];  // Change top to store ne

    // Attach the previous top to the beginning c
    next[i] = free;
    free = i;

    // Return the previous top item
    return arr[i];
}

/* Driver program to test twoStacks class */
int main()
{
    // Let us create 3 stacks in an array of size
    int k = 3, n = 10;
    kStacks ks(k, n);

    // Let us put some items in stack number 2
    ks.push(15, 2);
    ks.push(45, 2);

    // Let us put some items in stack number 1
    ks.push(17, 1);
```

```
    ks.push(49, 1);
    ks.push(39, 1);

    // Let us put some items in stack number 0
    ks.push(11, 0);
    ks.push(9, 0);
    ks.push(7, 0);

    cout << "Popped element from stack 2 is " <<
    cout << "Popped element from stack 1 is " <<
    cout << "Popped element from stack 0 is " <<

    return 0;
}
```

Java

```
// Java program to demonstrate implementation of
// array in time and space efficient way

public class GFG
{
    // A Java class to represent k stacks in a si
    static class KStack
    {
        int arr[];   // Array of size n to store
        int top[];   // Array of size k to store
        int next[];  // Array of size n to store
                     // and free list
        int n, k;
        int free; // To store beginning index of
```

```
//constructor to create k stacks in an ar
KStack(int k1, int n1)
{
    // Initialize n and k, and allocate m
    k = k1;
    n = n1;
    arr = new int[n];
    top = new int[k];
    next = new int[n];

    // Initialize all stacks as empty
    for (int i = 0; i < k; i++)
        top[i] = -1;

    // Initialize all spaces as free
    free = 0;
    for (int i = 0; i < n - 1; i++)
        next[i] = i + 1;
    next[n - 1] = -1; // -1 is used to ir
}

// A utility function to check if there i
boolean isFull()
{
    return (free == -1);
}

// To push an item in stack number 'sn' w
void push(int item, int sn)
{
    // Overflow check
    if (isFull())
    {
        System.out.println("Stack Overflc
        return;
```

```
    }

    int i = free; // Store index of first

    // Update index of free slot to index
    free = next[i];

    // Update next of top and then top fo
    next[i] = top[sn];
    top[sn] = i;

    // Put the item in array
    arr[i] = item;
}

// To pop an element from stack number 's
int pop(int sn)
{
    // Underflow check
    if (isEmpty(sn))
    {
        System.out.println("Stack Underfl
        return Integer.MAX_VALUE;
    }

    // Find index of top item in stack nu
    int i = top[sn];

    top[sn] = next[i]; // Change top to s

    // Attach the previous top to the beg
    next[i] = free;
    free = i;

    // Return t
```



```
item
```

```
            return arr[i];
        }

        // To check whether stack number 'sn' is
        boolean isEmpty(int sn)
        {
            return (top[sn] == -1);
        }

    }

    // Driver program
    public static void main(String[] args)
    {
        // Let us create 3 stacks in an array of
        int k = 3, n = 10;

        KStack ks = new KStack(k, n);

        ks.push(15, 2);
        ks.push(45, 2);

        // Let us put some items in stack number
        ks.push(17, 1);
        ks.push(49, 1);
        ks.push(39, 1);

        // Let us put some items in stack number
        ks.push(11, 0);
        ks.push(9, 0);
        ks.push(7, 0);

        System.out.println("Popped element from s
        System.out.println("Popped element from s
        System.out.prin
```



```
ment from s
```

```
    }
}
// This code is Contributed by Sumit Ghosh
```

Python3

```
# Python 3 program to demonstrate implementation
# of k stacks in a single array in time and space
# efficient way
class KStacks:

    def __init__(self, k, n):
        self.k = k # Number of stacks.
        self.n = n # Total size of array holding
                   # all the 'k' stacks.

        # Array which holds 'k' stacks.
        self.arr = [0] * self.n

        # All stacks are empty to begin with
        # (-1 denotes stack is empty).
        self.top = [-1] * self.k

        # Top of the free stack.
        self.free = 0

        # Points to the next element in either
        # 1. One of the 'k' stacks or,
        # 2. The 'free' stack.
        self.next = [i + 1 for i in range(self.n
        self.next[self.n - 1] = -1
```

```
    # Check whether given stack is empty.
    def isEmpty(self, sn):
        return self.top[sn] == -1

    # Check whether there is space left for
    # pushing new elements or not.
    def isFull(self):
        return self.free == -1

    # Push 'item' onto given stack number 'sn'.
    def push(self, item, sn):
        if self.isFull():
            print("Stack Overflow")
            return

        # Get the first free position
        # to insert at.
        insert_at = self.free

        # Adjust the free position.
        self.free = self.next[self.free]

        # Insert the item at the free
        # position we obtained above.
        self.arr[insert_at] = item

        # Adjust next to point to the old
        # top of stack element.
        self.next[insert_at] = self.top[sn]

        # Set the new top of the stack.
        self.top[sn] = insert_at

    # Pop item from given stack number 'sn'.
    def pop(self, sn):
```

```
        if self.isEmpty(sn):
            return None

        # Get the item at the top of the stack.
        top_of_stack = self.top[sn]

        # Set new top of stack.
        self.top[sn] = self.next[self.top[sn]]

        # Push the old top_of_stack to
        # the 'free' stack.
        self.next[top_of_stack] = self.free
        self.free = top_of_stack

        return self.arr[top_of_stack]

    def printstack(self, sn):
        top_index = self.top[sn]
        while (top_index != -1):
            print(self.arr[top_index])
            top_index = self.next[top_index]

# Driver Code
if __name__ == "__main__":

    # Create 3 stacks using an
    # array of size 10.
    kstacks = KStacks(3, 10)

    # Push some items onto stack number 2.
    kstacks.push(15, 2)
    kstacks.push(45, 2)

    # Push some items onto stack number 1.
    kstacks.push(17, 1)
```

```
    kstacks.push(49, 1)
    kstacks.push(39, 1)

    # Push some items onto stack number 0.
    kstacks.push(11, 0)
    kstacks.push(9, 0)
    kstacks.push(7, 0)

    print("Popped element from stack 2 is " +
                        str(kstacks.pop(2)))
    print("Popped element from stack 1 is " +
                        str(kstacks.pop(1)))
    print("Popped element from stack 0 is " +
                        str(kstacks.pop(0)))

    kstacks.printstack(0)

# This code is contributed by Varun Patil
```

C#

```
using System;

// C# program to demonstrate implementation of k
// array in time and space efficient way

public class GFG
{
    // A c# class to represent k stacks in a sing
    public class KStack
    {
        public int[] ar
```



```
size n to s
```

```
public int[] top; // Array of size k to s
public int[] next; // Array of size n to
                   // and free list
public int n, k;
public int free; // To store beginning i

//constructor to create k stacks in an ar
public KStack(int k1, int n1)
{
    // Initialize n and k, and allocate m
    k = k1;
    n = n1;
    arr = new int[n];
    top = new int[k];
    next = new int[n];

    // Initialize all stacks as empty
    for (int i = 0; i < k; i++)
    {
        top[i] = -1;
    }

    // Initialize all spaces as free
    free = 0;
    for (int i = 0; i < n - 1; i++)
    {
        next[i] = i + 1;
    }
    next[n - 1] = -1; // -1 is used to ir
}

// A utility function to check if there i
public virtual bool Full
{
    get
```

```
    {
        return (free == -1);
    }
}

// To push an item in stack number 'sn' w
public virtual void push(int item, int s
{
    // Overflow check
    if (Full)
    {
        Console.WriteLine("Stack Overflow
        return;
    }

    int i = free; // Store index of first

    // Update index of free slot to index
    free = next[i];

    // Update next of top and then top fo
    next[i] = top[sn];
    top[sn] = i;

    // Put the item in array
    arr[i] = item;
}

// To pop an element from stack number 's
public virtual int pop(int sn)
{
    // Underflow check
    if (isEmpty(sn))
    {
        Console.WriteLine("Stack Underflo
```

```
            return int.MaxValue;
        }

        // Find index of top item in stack nu
        int i = top[sn];

        top[sn] = next[i]; // Change top to s

        // Attach the previous top to the beg
        next[i] = free;
        free = i;

        // Return the previous top item
        return arr[i];
    }

    // To check whether stack number 'sn' is
    public virtual bool isEmpty(int sn)
    {
        return (top[sn] == -1);
    }

}

// Driver program
public static void Main(string[] args)
{
    // Let us create 3 stacks in an array of
    int k = 3, n = 10;

    KStack ks = new KStack(k, n);

    ks.push(15, 2);
    ks.push(45, 2);
```

```
        // Let us put some items in stack number
        ks.push(17, 1);
        ks.push(49, 1);
        ks.push(39, 1);

        // Let us put some items in stack number
        ks.push(11, 0);
        ks.push(9, 0);
        ks.push(7, 0);

        Console.WriteLine("Popped element from st
        Console.WriteLine("Popped element from st
        Console.WriteLine("Popped element from st
    }
}

// This code is contributed by Shrikant13
```

Javascript

```
// javascript program to demonstrate implementati
// array in time and space efficient way

    // A javascript class to represent k stacks i
     class KStack {

        // constructor to create k stacks in an a
        constructor(k1 , n1)
        {

            // Initialize n and k, and allocate m
            this.k = k1
```

```
    this.n = n1;
    this.arr = Array(n).fill(0);
    this.top = Array(k).fill(-1);
    this.next = Array(n).fill(0);


    // Initialize all spaces as free
    this.free = 0;
    for (var i = 0; i < n - 1; i++)
        this.next[i] = i + 1;
    this.next[n - 1] = -1; // -1 is used
}

// A utility function to check if there i
 isFull() {
    return (this.free == -1);
}

// To push an item in stack number 'sn' w
 push(item , sn)
 {

    // Overflow check
    if (this.isFull()) {
        document.write("Stack Overflow");
        return;
    }

    var i = this.free; // Store index of

    // Update index of free slot to index
    this.free = this.next[i];

    // Update next of top and then top fo
    this.next[i ];
```

```
        this.top[sn] = i;

        // Put the item in array
        this.arr[i] = item;
    }

    // To pop an element from stack number 's
     pop(sn)
     {

        // Underflow check
        if (this.isEmpty(sn)) {
            document.write("Stack Underflow")
            return Number.MAX_VALUE;
        }

        // Find index of top item in stack nu
        var i = this.top[sn];

        this.top[sn] = this.next[i]; // Chang

        // Attach the previous top to the beg
        this.next[i] = this.free;
        this.free = i;

        // Return the previous top item
        return this.arr[i];
    }

    // To check whether stack number 'sn' is
     isEmpty(sn) {
        return (this.top[sn] == -1);
    }

}
```

```
// Driver program

    // Let us create 3 stacks in an array of
    var k = 3;
    n = 10;

    var ks = new KStack(k, n);

    ks.push(15, 2);
    ks.push(45, 2);

    // Let us put some items in stack number
    ks.push(17, 1);
    ks.push(49, 1);
    ks.push(39, 1);

    // Let us put some items in stack number
    ks.push(11, 0);
    ks.push(9, 0);
    ks.push(7, 0);

    document.write("Popped element from stack
    document.write("<br/>Popped element from
    document.write("<br/>Popped element from

// This code is contributed by gauravrajput1
```

Output

```
Popped element from stack 2 is 45
Popped element from stack 1 is 39
Popped element from stack 0 is 7
```

Time complexities of operations push() and pop() is O(1). The best part of above implementation is, if there is a slot available in stack, then an item can be pushed in any of the stacks, i.e., no wastage of space.

Time Complexity of top() operation is also O(1)

Time Complexity: O(N), as we are using a loop to traverse N times.
Auxiliary Space: O(N), as we are using extra space for the stack.

which every advanced algorithm is built upon. Choose your preferred language and start learning today:

Recommended Problems

Frequently asked DSA Problems Solve Problems

266 Suggest improvement

Previous

Implement Stack using Queues

Next

Design a stack that supports getMin() in O(1) time and O(1) extra space

Skip to content

Share your thoughts in the comments

Add Your Comment

Similar Reads

- How to efficiently implement k Queues in a single array?
- Implement two Stacks in an Array
- Implement Dynamic Multi Stack (K stacks) using only one Data Structure
- Sorting array using Stacks
- Implement a stack using single queue
- C Program for efficiently print all prime factors of a given number
- Java Program for efficiently print all prime factors of a given number
- Finding the Parity of a number Efficiently
- Compute maximum of the function efficiently over all sub-arrays
- How to store a Sparse Vector efficiently?

S **Sachin**

Skip to content

Article Tags : DSA , Stack

Practice Tags : Stack

GeeksforGeeks
Sanchhaya Education Private Limited

A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

Company

About Us
Legal
Careers
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program

Explore

Job-A-Thon Hiring Challenge
Hack-A-Thon
GfG Weekly Contest
Offline Classes (Delhi/NCR)
DSA in JAVA/C++
Master System Design
Master CP
GeeksforGeeks Videos
Geeks Community

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial

DSA

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
DSA Interview Questions
Competitive Programming

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning Tutorial
ML Maths
Data Visualisation Tutorial
Pandas Tutorial
NumPy Tutorial
NLP Tutorial
Deep Learning Tutorial

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
NodeJs
Bootstrap
Tailwind CSS

Python Tutorial

Computer Science

DevOps

Git

System Design

School Subjects

Commerce

Accountancy

Skip to content