Dash

All

Articles

Videos

Problems

Quiz

« Prev

Next »

Article marked as read.

Implement K Stacks in an Array

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.*

Following functions must be supported by kStacks. 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] to arr[n/k-1] for first 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 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 o
The top of this stack is stored in a variable 'free'. All entries in top[] are initializ
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.

Following is implementation of the above idea.

C++ Java

```
// Java program to demonstrate implementation of k stacks in a single
// array in time and space efficient way

public class GFG
{
    // A Java class to represent k stacks in a single array of size n
    static class KStack
    {
        int arr[]; // Array of size n to store actual content to be stored in stacks
        int top[]; // Array of size k to store indexes of top elements of stacks
        int next[]; // Array of size n to store next entry in all stacks
                    // and free list
        int n, k;
        int free; // To store beginning index of free list

        //constructor to create k stacks in an array of size n
        KStack(int k1, int n1)
```

Dash
All
Articles
Videos
Problems
Quiz

Article marked as read.

```
{
    // Initialize n and k, and allocate memory for all arrays
    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 indicate end of free list
}

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

// To push an item in stack number 'sn' where sn is from 0 to k-1
void push(int item, int sn)
{
    // Overflow check
```

Dash
All
Articles
Videos
Problems
Quiz

Article marked as read.

```
        if (isFull())
        {
            System.out.println("Stack Overflow");
            return;
        }

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

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

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

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

    // To pop an from stack number 'sn' where sn is from 0 to k-1
    int pop(int sn)
    {
        // Underflow check
        if (isEmpty(sn))
        {
            System.out.println("Stack Underflow");
            return Integer.MAX_VALUE;
        }
```

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

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

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

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

    // To check whether stack number 'sn' is empty or not
    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 size 10
    int k = 3, n = 10;

    KStack ks = new KStack(k, n);
```

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



```
        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);

        System.out.println("Popped element from stack 2 is " + ks.pop(2));
        System.out.println("Popped element from stack 1 is " + ks.pop(1));
        System.out.println("Popped element from stack 0 is " + ks.pop(0));
    }
}
```

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.

Article marked as read.

Dash

All

Articles

Videos

Problems

Quiz

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 stack.

Marked as Read

Report An Issue

If you are facing any issue on this page. Please let us know.