INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN: 2278 – 0211 (Online)

Algorithm For Stack With Random Operations (Stack Using Random Array Operations)

Suchait Gaurav
Department Of Computer Science & Engineering
ShriRam College Of Engineering & Management, Gwalior, India

Abstract :
A stack is an ordered collection of items for which we can only add or remove items from one end (the top of the stack). The stack is another container class, much like a list, but with a much more limited set of operations: push, pop and size .
The proposed work presents a fundamental algorithm to perform add,delete and size operations on random locations in stack .Array based implementation of this algorithm can perform operation at any random location in stack (not limited with top only).The total algorithm works on two basic set of arrays in which one works as stack (for storing elements) and other works as confirmation table for stack (for storing and matching locations).

Key words*: Stack Algorithm , One dimensional array , data structure*

1.Introduction

As we all know a stack has only one end (top) for performing all operations,but using this technique /algorithm one can perform operations on stack with random locations .We can make the top random by implementation of arrays using this algorithm .
For getting this approach of stack with operations at random locations we need two things :

- A first set of one dimensional array – that works as stack
- A second set of one dimensional array – that keep the tracks of all locations of the stack , it works as a confirmation table for first set of arrays (stack).

What ever the operation you perform on the element of the stack , the location of that element will be kept on the confirmation table (i.e. a second set of array) for further confirmation and matching used in operations .

2.Proposed Algorithm

In this algorithm we are going to perform three basic operations on a stack .
Operations :

- Add
- Delete
- Show

```
1.  Algorithm add(item,location)
2.  {
//push a element/item into the stack at provided location
//save that location in confirmation table to keep the track of stack locations
3.  if(stack[location] not equal to  null ) then
4.  {
5.   stack[location]=item;
6.   table_location=table_location+1;
7.   }
8.   else
9.   {
10.  write("false");
11.  return false;
12.  }
```

```
13.  stack_table[table_location]=location;
14.      return true;
15.      }
```

```
1.   Algorithm  remove(item,location)
2.  {
//setting the item at given location to null
//and remove the location from confirmation table too
3.   for i:=1 to MAX do
4.   {
5.   if (stack[i] equals  to item)  then
6.   {
7.   stack[i]=null;
8.    }
9.    }
10.   for j:=1 to MAX do
11.   {
12.   if (stack_table[j] equals location ) then
13.   {
14.   stack_table[j]=null;
15.   }
16.   }
17.    }
```

```
1.   Algorithm show()
2.  {
3.  for k:=table_location to >= 1 do
4.  {
5.  if(stack_table[k] not equals to null) then
6.  {
7.  value=stack_table[k];
8.  write(stack[value]);
//it prints the value of stack at a given location and show the whole stack
9.  }
10.  }
11.  }
```

3.Explanation Of Algorithm And Operations

Our basic aim is to make the top of stack random , so that we can put the element into the stack at anywhere without having any condition of top.

So, for this we have two array memories now , as described above

Stack and confirmation table

So everytime when you add the element in the stack at a specific location it checks in the confirmation table , for confirming that is any element/item existing at that location or not .

For example if we have stack[5] and stack_table[5] (both must be of same index) then

stack[5]
stack[4]
stack[3]
stack[2]
stack[1]

Stack

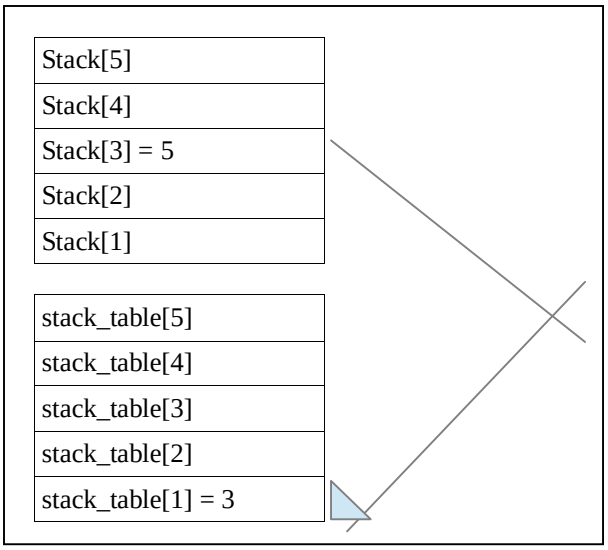
stack_table[5]
stack_table[4]
stack_table[3]
stack_table[2]
stack_table[1]

Stack table

4.Add Operation

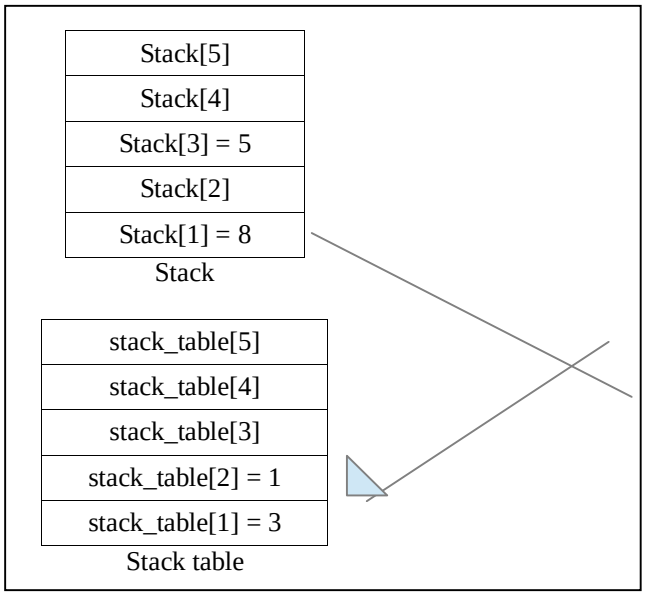
When you add a element into the stack at any random value then the location of that element will be automatically inserted into the stack_table using FILO phenomenon it means locations int stack_table will be inserted in incrementing order.

For example if you insert the value 5 at stack[3] ,then in stack table location of element i..e 3 will be automatically inserted into in stack table at stack_table[1]=3 .



Now when you insert the 8 in the stack suppose at location stack[1] ,then in stack table location of element i..e 1 will be automatically inserted into in stack table at stack_table[2]=1 .

What ever the operation you performed at any stack location ,the second confirmation table will automatically store that location into it .

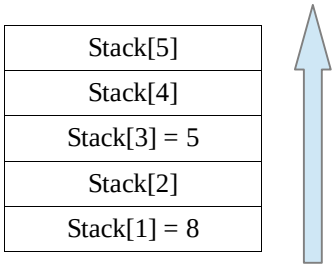


5.Delete Operation

For implementing a delete operation into the stack ,we set the memory to null state .Suppose if i want to delete to stack[1] , then For this we use a for loop to move all over the stack array .

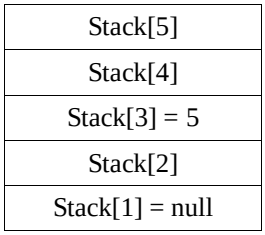
1. for(int i=1;i<=MAX;i++)

Now is start moving from stack[1] to stack[5]

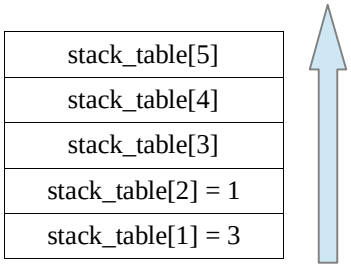


Loop is moving upward for finding the given element in the stack array.

2. if(stack[i] equals to item) then
   set stack[1]=null;

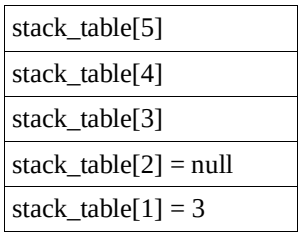


3. Now we have to set null to the location in confirmation table also.for implementing this we again use for loop for moving in the stack_table array .
   for(int q=1;q<=MAX;q++)



Loop is moving upward for finding the given element in the stack_table array.

4. if(stack_table[q] equals to location) then
   set stack_table[q]=null;



Now the delete operation at location stack[1] is done completely .

6.Show Operation

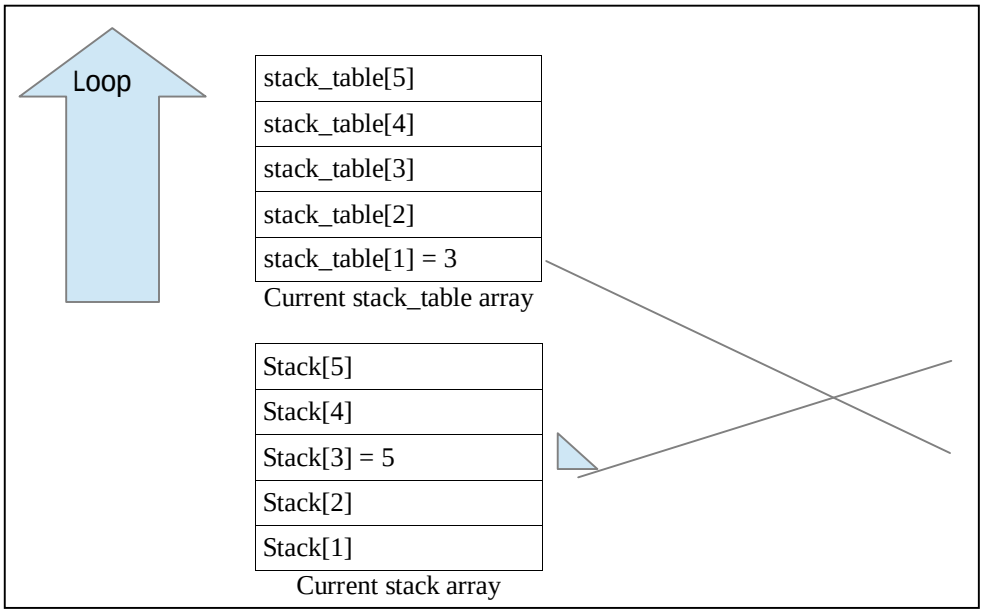
Now we have performed both operations on a stack (add and delete) now we wants to display the elements in stack array . There are two basic steps for displaying these elements from the stack .

- Take the locations from confirmation table that are starting from stack_table[1] .
- Match these locations in to stack array to get the elements at that location and display them .

Steps :

1. for (int y=table_location; y>=1; --y)
2. now check if(stack_table[y] not equals not null) then
3. suppose int h= stack_table[y] ,we get the location here from confirmation table
4. now use this location for getting value from stack .
5. Display stack[h]
6. end .

Now in our stack[5] and stack_tables arrays we have only one value left after deleting the 8 from location stack[1] .



And it will display the elements from stack array .

7.Experimental Implementation

The above algorithm can be easily implemented using c++.A small piece of code is provided for implementing this algorithm .

```
#include<iostream>
using namespace std;
#define MAX 5
class stack{
public :
int st[MAX],st_table[MAX];
int table_location;
stack(); //constructor to set the default value of table_location variable
void add(int item,int location); //function for pushing values into the stack
int remove(int item,int location); //function for removing the values from the stack
void show();   //function for displaying the values in the stack
};
stack::stack()
{
table_location=0;
}
void stack::add(int item,int location)
{
if(st[location]!='\0')
{
st[location]=item;
table_location++;  //incrementing table_location by 1
}
else
{
cout<<endl<<"Problem"<<endl;
```

```
}
//checking process begins from here
st_table[table_location]=location; //putting value in another array that working as a table for confirmation
} //ending push function here
int stack::remove(int item,int location)
{
for(int d=1;d<=MAX;d++)
{
if(st[d]==item)
{
st[d]='\0';
}
}
for(int v=1;v<=MAX;v++)
{
if(st_table[v]==location)
{
st_table[v]='\0';
}
}
}//ending remove function here
void stack::show()
{
cout<<endl;
cout<<"Elements in stack"<<endl;
for (int y=table_location; y>=1; --y)
{
if(st_table[y]!=0)
{
int h=st_table[y];
cout<<endl;
cout<<st[h];
cout<<endl;
}
}
}
int main()
{
stack s;
s.add(12,2);
s.add(21,5);
s.add(675,3);
s.add(7,4);
s.remove(21,5);
s.remove(12,2);
s.show();
return 0;
}
```

8.Experimental Result And Conclusion

By performing this experiment a proper accurate output given as the size of stack that displays the elements left in the stack .

Now 7 and 675 are the final elements left in stack after performing all operations ,that shows the size of the stack , it shows that we can store elements in stack at random locations using a implementation of confirmation table .

9.Acknowledgment
The principle author's acknowledgment is due to Sh.R.S.Sharma,chairman,ShriRam Group of Colleges(SRGOC) for the inspiration and dedication to carry the research .

10.References

1) Fundamentals of Computer Algorithms, Second Edition, by E.Horowitz,S.Sahni and Sanguthevar Rajasekaran
2) Fundamentals of Data Structures in C++, Second Edition, by E.Horowitz,S.Sahni and D.Mehta,Silicon Press,2007
3) Data Structures,Algorithms,and Applications in C++ , by S.Sahni,Second Edition ,Silicon Press,2005
4) Algorithms ,by Robert Sedgewick and Kevin Wayne ,Fourth Edition ,2011
5) M. Herlihy. Wait-free synchronization. ACM Transactions On Programming Languages and Systems, 13(1):123–149, Jan. 1991.
6) T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms, Second Edition.
7) C. C. McGeoch. Toward an experimental method for algorithm simulation. IN-
8) FORMS Journal on Computing, 1(1):1—15, Winter 1996.