Stacks

Stacks : Data Structure

Purpose: LIFO : last in first out

A container with only 1 side open

Real World Examples:

1) Stack of plates
2) Pile of books
3) Undo/Redo functionality
4) Adding Plates to counter

Basic operations
: push()
: pop()
: top()/peek()
: size()
: isEmpty()

Ex: Data: 1 4 5 3 2 3 2 5 6 7 8 9 10

push(1) 1 | push(4) 1 4 | push(5) 1 4 5 | push(3) 1 4 5 3 | push(2) 1 4 5 3 2 | push(3) 1 4 5 3 2 3 | push(2) 1 4 5 3 2 3 2 | push(5) 1 4 5 3 2 3 2 5 | push(6) 1 4 5 3 2 3 2 5 6 | push(7) 1 4 5 3 2 3 2 5 6 7 | push(8) 1 4 5 3 2 3 2 5 6 7 8 | push(9) 1 4 5 3 2 3 2 5 6 7 8 9 | push(10) 1 4 5 3 2 3 2 5 6 7 8 9 10

Deleted

Pop the top because Take it as data

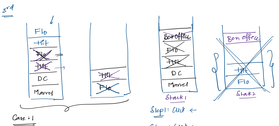
size() → 2

1
8
9
7
6
10

→ Usage of stack in computer science

1) Recursion/Call Stack
2) Undo/Redo
3) Browser ? & ! are implemented by stack
4) Expression Evaluator
5) Balanced Brackets in Not (( ))

2nd



Step3: [Searching for Bon offers] → Not found

[Empty stack, Return]

Marvel → DC → HTTP → File

Searching HTTP → HTTP : Found, goes Bonfliix

Bon offers →

Bonfliix → Bon offers →

Q1:

Marvel → DC → HTTP → File

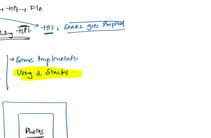
Searching HTTP → HTTP : Found, goes Bonfliix

→ Same implementation

Undo & Redo

Using 2 Stacks

3



→ Actually Circular linked list

→ Use 2 Stacks:

Not a stack

Implementation of Stacks

1) Arrays:
2) Dynamic Array → Vector/Array list
3) Linked list

int ar[100] = 0;

int top = -1;

void push(int x) {
if(top == 99) {
ar[++top] = x
}
}

void pop() {
if(top == -1) {
top--;
}
}

int peek() {
if(top == -1) {
return -1;
}
return ar[top]
}

int size() {
return top+1
}

bool Empty() {
if(top==-1) return True;
else return False;
}

0	1	2	3
5	10	20	
		T	

top = 2

Delete → top - 1

Insert 20 : 2

Inserting Elements

Push()
pop()
top()/peek()
size()
is Empty()

→ O(1)

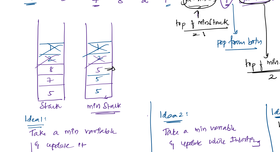
Inserted top Elements

top = 99

Can we Insert 1 more Element → No, Stack Overflow

→ Implement 1 more function in stack (getMin())

Ex: 5 7 8 2 1



Idea1:
Take a min variable & update it

→ While popping: we cannot update min value

Idea2:
Take a min variable & update while pushing

→ While popping: Update min before stack & update min value

⇒ We can store Stack, & basic operations

Idea3:
We have to use Extra Stack.

Stack1 Stack2

Insert(x): Insert x in Stack1, Insert min(x, Stack2.top()) in Stack2 : O(1)

pop() : pop()

pop() : Pop from both stacks

getMin() : return Stack2.top() : O(1)

SC: O(N) → For entire Stack.

1st We can do it for getMin()

2nd We can do it for getAvg(): We can keep an avg value & update it

avg for x = (sum*avg - deleted) / (size-1)

→ Print Kth Min/Max Element from 100

top = 99

int ar[100]; Implement 2 Stacks in Same array

top1 = -1

top2 = 99

Stack1 → [0 - 49]

Stack2 → [50 - 99]

Case 1

Stack1:
We prefer >0 Element
top = 49

→ We have empty space
→ We are not able to use it
→ Space wastage.

Stack2:
Can we even push 60 elements?

Case 2:

top1 = -1

Insert(x): ar[++top1] = x

pop(): top1--

peek(): return ar[top1]

size(): return top1+1

top2 = 100 :

Insert(x): ar[--top2] = x

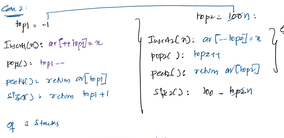
pop(): top2++

peek(): return ar[top2]

size(): 100 - top2

If we Expand Array
top2 = 98 ;
top2--;

of 2 Stacks



→ Memory wastage

Steps to implement 3 stacks in a array of size N, without wastage

Wastage of space: → 3 << N

HW Given a stack, sort it using another stack.

Observations

→ In Stack1 data should be sorted

→ From Stack2 → Stack1 there should be transfer for data.

→ Stack2 data has to be in decreasing order.

We have to top stack in decreasing order

Idea:

Until Stack1 is Empty:

temp = Stack1.top()

Stack1.pop()

// Keep temp with Stack2.top()

while(Stack2.size()>0 && temp < Stack2.top()) {

// Delete an element from Stack2

// Insert in Stack1

}

Stack2.Insert(temp) // Insert temp in Stack2

}

TC: O(N^2) SC: O(N)

→ Deques:

1) Non-fixed sized arrays:
2) Vertical & Horizontal lines:

→ Non-fixed prices > 0 → (1)

Non-fixed prices ≥ 0 → Non-fixed positive

5 :

100 5

→ 5 (100)(99)(98)(97)(96)

→ 5 → 10 → (10) → (100)