Stacks Lab Session 1

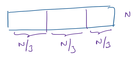
Sunday, 30 May 2021 10:06 AM

→ Implement k stacks in an array?

K=1 | K=2 ?

top1=0 | top2=n-1

K=3



Way-1 Divide array into 3 equal parts. (Memory wastage)

Way2:

→ We can define an array of vectors: [k]

→ Each vector acts as a stack.

→ array of vectors

vector<int> st[k], int stsz=N; At most we can have N elements.

```
void push(int no, int data){
    if(stsz==0) {overflow return}
    st[no].push_back(data)
    stsz--; // one empty space decreased

void pop(int no){
    if(stsz==N) {underflow return}
    st[no].pop_back()
    stsz++; // one empty space increased

int top(int no){
    int s = st[no].size();
    if(s==0){underflow, nothing}
    return st[no][s-1];
}
```

→ All operations: $O(1)$

→ SC: $O(1)$

→ Linear(N)/vector

N=10, K=3

0	1	2	3	4	5	6	7	8	9
10	11	13	25	29	39	18	25	X	

-1	0	1						2	

Top

0	-1 → 0 → 1 → 2 → 8
1	-1 → 3 → 4 → 5
2	-1 → 6 → 7

operations:

Insert: 10 in stack 1

: 11 in stack 1

: 13 in stack 1

: 25 in stack 2

: 29 in stack 2

: 39 in stack 2

: 18 in stack 3

: 25 in stack 3

: 4 in stack 1

Issue:

Pop element in stack 1:

pop element in stack 1:

To resolve above issue:

pre[N]:

For suppose we want to insert new element in st 3, → 7

pre[7] = top[2];

top[2] = 7

N=10, K=3

	0	1	2	3	4	5	6	7	8	9
ar[]:	10	11	X	25	29	X	18	25	X	
pre[]:	-1	0	X	-1	3	X	-1	6	X	

top[k]:

0	-1 → 0 → 1 → 2 → X → X → 1
1	-1 → 3 → 4 → 4 → 4
2	-1 → 6 → 7

Insert: 10 in stack 1 →

: 11 in stack 1 →

: 13 in stack 1 →

: 25 in stack 2 →

: 29 in stack 2 →

: 39 in stack 2 →

: 18 in stack 3 –

: 25 in stack 3 –

: 4 in stack 1 –

pop from stack 1:

8 is deleted

top[st] = pre[top[st]]

pop from stack 1:

pop from stack 2:

Insert elem to stack 1:

: Ideally we need to insert : 2

: Issue: we don't know where do we have to actually insert?

How to know at what index we have insert our element?

	0	1	2	3	4	5	6	7	8	9
ar:	10	11	13	X	11	8	18	25	4	40
pre:	-1	0	1	-1	2	7	-1	6	2	8
que:	8	X	X	X	X	X	X	X	X	9	8	3

top

1	-1 → 0 → 1 → 2 → 8 → X → X → 2 → 4
2	-1 → 2 → 1 → 8 → 4 → 3 → -1
3	-1 → X → X → 7 → 5

int ar[N], int pre[N], int top[k] = -1;

queue<int> q;

// Insert all 0, n-1 index value to q

```
void push(int st, int data){
    if(q.size()==0) {No empty space available return;}
    int cur = q.front();
    q.dequeue(); // Delete front's element
    ar[cur] = data;
    pre[cur] = top[st];
    top[st] = curr;
}
```

→ $O(1)$

```
void pop(int st){
    int find = top[st]
    if(find==-1) {current stack empty no delete}
    // find we are allow to delete
    top[st] = pre[find];
    q.enque(find)
}
```

$O(1)$

Insert: 10 in stack 1 →

: 11 in stack 1 →

: 13 in stack 1 →

: 25 in stack 2 →

: 29 in stack 2 →

: 39 in stack 2 →

: 18 in stack 3 –

: 25 in stack 3 –

: 4 in stack 1 –

: 40 in stack 1 →

pop from stack 2

pop from stack 2

pop from stack 1

pop from stack 1

Insert 8 in stack 3

Insert 11 in stack 1

pop from stack 2

→ If at all no predefined boundary

→ I will take ar[N], I will implement circular queue.

→ you cannot implement non queue or un predefined queue.

N=10:

	0	1	2	3	4	5	6	7	8	9
int ar[]:	30	40	X	10	X	19	29	X		
int pre[]:	-1	0	1	-1	X	-1	5	6		
int next[]:	1	2	4	4	8	6	7	2	9	-1

int curr = 0

Insert 30 in st 1 : curr = 1

Insert 40 in st 1 : curr = 2

Insert 50 in st 1 : curr = 3

Insert 10 in st 2 : curr = 4

Insert 15 in st 2 : curr = 5

Insert 19 in st 2 : curr = 6

Insert 39 in st 2 : curr = 7

Insert 29 in st 2 : curr = 8

pop from stack 2:

→ find we pop : 4

curr = 8

next[4] = curr_available

curr = 4

pop from stack 1:

→ find we pop: 2

→ curr = 4

→ next[2] = 4

→ curr = 2

pop from stack 2:

→ find : 7 // Recently deleted index

→ curr : 2 // current available

next[curr] = find

curr = find

int ar[N], pre[N], top[k],

int next[N] = [1, 2, 3, -- N, -1]

int curr = 0

```
void push(int st, int data){
    if(curr==-1) {overflow}
    ar[curr] = data; // at what index we insert
    pre[curr] = top[st] // updating pre of stack
    top[st] = curr // updating top of stack
    curr = next[curr] // updating curr
}
```

$O(1)$

```
void pop(int st){
    int find = top[st];
    if(find==-1) {stack is empty // skip}
    // If we are deleting this find
    top[st] = pre[find]; // updating top of stack.
    next[find] = curr
    curr = find;
}
```

$O(1)$

curr

curr

curr

NULL

curr

curr

Note: else: At any index position find

If data is filled at find: we only need to store its previous index

If data is available at find: we only need to store its next

→ At any given point at any given index we only need to store 2 value.

That means this entire thing can be done using 2 extra array.