STACKS AND QUEUES
NOTES
by
aminotes

STACKS

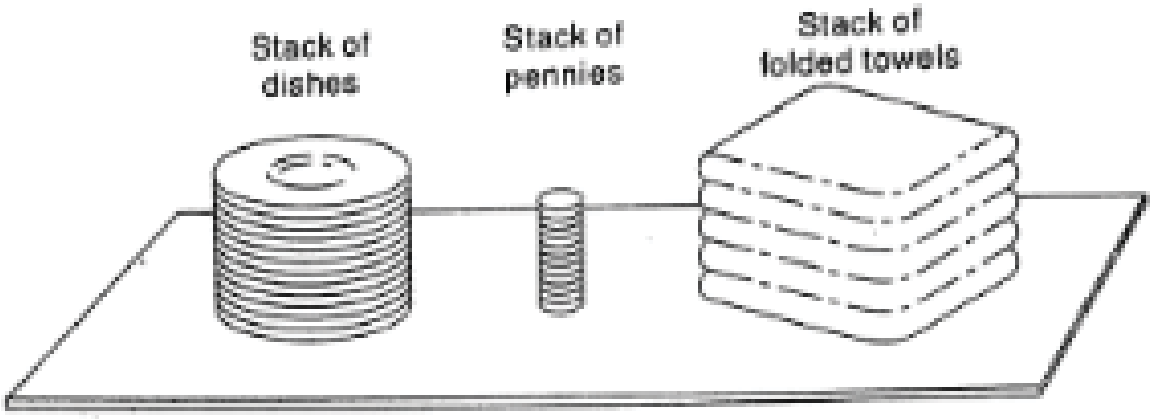
A Stack is a list of elements in which an element may be inserted or deleted only at one end, called the top of the stack. This means, in particular, that elements are removed from a stack in the reverse order of that in whicch they were inserted into the stack.

The two basic operations associated with stacks are:

(i) “PUSH” - It is the term used to insert an element into a stack.

(ii) “POP” - It is the term used to delete an element from a stack.

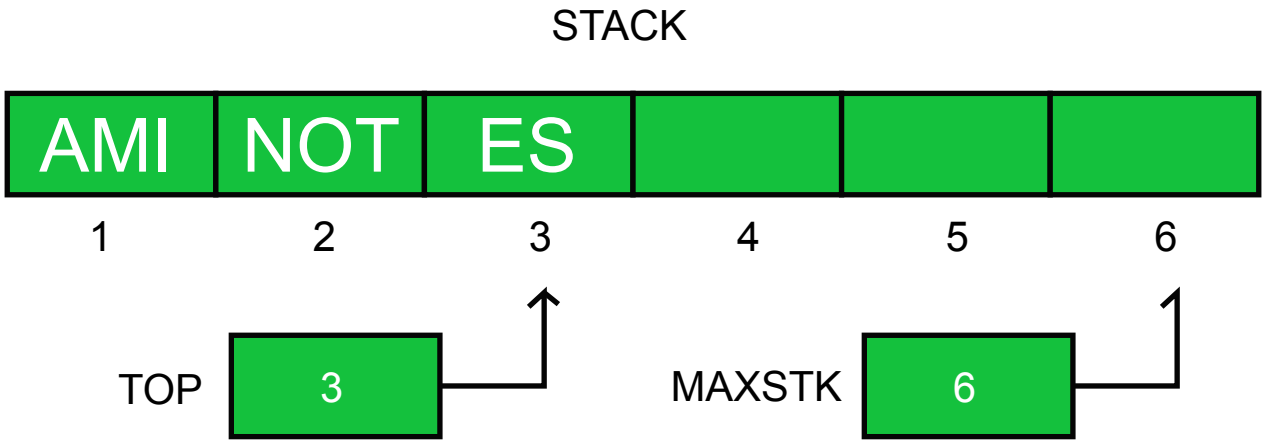
NOTE- These terms are used only with stacks and not with other data structures.



ARRAY REPRESENTATION OF A STACK

Stacks can be represented in a linear way. Each of our stacks will be maintained by a linear array STACK; a pointer variable TOP, which contains the location of the top element of the stackl and a variable MAXSTK which gives the number of elements that can be held by the stack. The condition TOP=0 or TOP=NULL will indicate that the stack is empty.

The figure represent an array representation of a stack. Since TOP=3, the stack has three elements, AMI, NOT and ES; and since MAXSTK =6, there is room for 3 more items in the stack.



To add an element (PUSH), one must first test whether there is a room in the stack for the new item, if NOT then we have the condition known as Overflow. Similarly, in executing the procedure delete (POP), one must first test whethere there is an element in the stack to be deleted, if NOT then we have the condition known as underflow.

ALGORITHM - STACK PUSH

PUSH(STACK, TOP, MAXSTK, ITEM)
This procedure pushes an ITEM onto a stack.

1. IF TOP = MAXSTK, then print OVERFLOW and return (it means the stack is full and we cannot add element into it.
2. SET TOP = TOP +1 (Increases the size of the stack, TOP by 1)
3. SET STACK[TOP] = ITEM (Inserts ITEM in new TOP position)
4. RETURN

ALGORITHM - STACK POP

POP(STACK, TOP, ITEM)
This procedure deletes the top element of the STACK and assigns it to the variable ITEM.

1. IF TOP = 0, then print UNDERERFLOW and return (If there is no element in the stack then we cannot delete)
2. SET ITEM= STACK[TOP] (Assigns the TOP element to ITEM)
3. SET TOP = TOP-1 (Decreases TOP by 1)
4. RETURN

CONVERSION INFIX TO POSTFIX

ALGORITHM -
1. PUSH “(” onto STACK, and add “)” to the end of Q.
2. SCAN Q from left to right and repeat Steps 3 to 6 for each element of Q until the stack is empty.
3. If an operand is encountered, add it to P.
4. If a left paranthesis is encountered, push it onto STACK.
5. If an operator is encountered then:
    (a) Repeatedly POP from STACK and add to each operator (on the top of STACK) which has the same precedenceas or higher precendence than ⊗
    (b) Add ⊗ to STACK.
6. If a right paranthesis is encountered, then:
    (a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a left paranthesis is encountered.
    (b) Remove the left parenthesis. [Do not add the left parenthesis on P.}
7. EXIT

CONVERSION INFIX TO PREFIX

ALGORITHM -
1. Reverse the Expression and parse the inputs in the expression one by one.
2. If the input is an operand, then place it in the output buffer.
3. If the input is an operator, push it into the stack.
4. If the operator in stack has equal or higher precedence than input operator, then pop the operator present in stack and add it to output buffer.
5. If the input is an close brace, push it into the stack.
6. If the input is a open brace, pop elements in stack one by one until we encounter open brace.
7. Discard braces while writing to output buffer.
8. EXIT

EXAMPLES ON INFIX TO POSTFIX

Consider the following arithmetic infix expression Q:

Q: A + (B * C – (D / E ↑ F) * G) * H

We simulate Algorithm 6.6 to transform Q into its equivalent postfix expression P. First we push "(" onto STACK, and then we add ")" to the end of Q to obtain:

Q:	A	+	(	B	*	C	–	(	D	/	E	↑	F	)	*
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
											G	)	*	H	)
											(16)	(17)	(18)	(19)	(20)

The elements of Q have now been labeled from left to right for easy reference. Figure 6.12 shows the status of STACK and of the string P as each element of Q is scanned. Observe that

(1) Each operand is simply added to P and does not change STACK.
(2) The subtraction operator (–) in row 7 sends * from STACK to P before it (–) is pushed onto STACK.
(3) The right parenthesis in row 14 sends ↑ and then / from STACK to P, and then removes the left parenthesis from the top of STACK.
(4) The right parenthesis in row 20 sends * and then + from STACK to P, and then removes the left parenthesis from the top of STACK.

After Step 20 is executed, the STACK is empty and

P: A B C * D E F ↑ / G * – H * +

which is the required postfix equivalent of Q.

EXAMPLES ON INFIX TO POSTFIX

Symbol Scanned		STACK	Expression P
(1)	A	(	A
(2)	+	(+	A
(3)	(	(+ (	A
(4)	B	(+ (	A B
(5)	•	(+ (•	A B
(6)	C	(+ (•	A B C
(7)	−	(+ (−	A B C •
(8)	(	(+ (− (	A B C •
(9)	D	(+ (− (	A B C • D
(10)	/	(+ (− (/	A B C • D
(11)	E	(+ (− (/	A B C • D E
(12)	↑	(+ (− (/ ↑	A B C • D E
(13)	F	(+ (− (/ ↑	A B C • D E F
(14)	)	(+ (−	A B C • D E F ↑ /
(15)	•	(+ (− •	A B C • D E F ↑ /
(16)	G	(+ (− •	A B C • D E F ↑ / G
(17)	)	(+	A B C • D E F ↑ / G • −
(18)	•	(+ •	A B C • D E F ↑ / G • −
(19)	H	(+ •	A B C • D E F ↑ / G • − H
(20)	)		A B C • D E F ↑ / G • − H • +

EXAMPLES ON INFIX TO PREFIX

We have infix expression in the form of:

((A-B)+C*(D+E))-(F+G)

Now reading expression from right to left and pushing operators into stack and variables to output stack

Input	Output_stack	Stack
)	EMPTY	)
G	G	)
+	G	)+
F	GF	)+
(	GF+	EMPTY
-	GF+	-
)	GF+	-)
)	GF+	-))
E	GF+E	-))
+	GF+E	-))+
D	GF+ED	-))+
(	GF+ED+	-)
*	GF+ED+	-)*
C	GF+ED+C	-)*
+	GF+ED+C*	-)+
)	GF+ED+C*	-)+)
B	GF+ED+C*B	-)+)
-	GF+ED+C*B	-)+)-
A	GF+ED+C*BA	-)+)-
(	GF+ED+C*BA-	-)+
(	GF+ED+C*BA-+	-
EMPTY	GF+ED+C*BA-+-	EMPTY

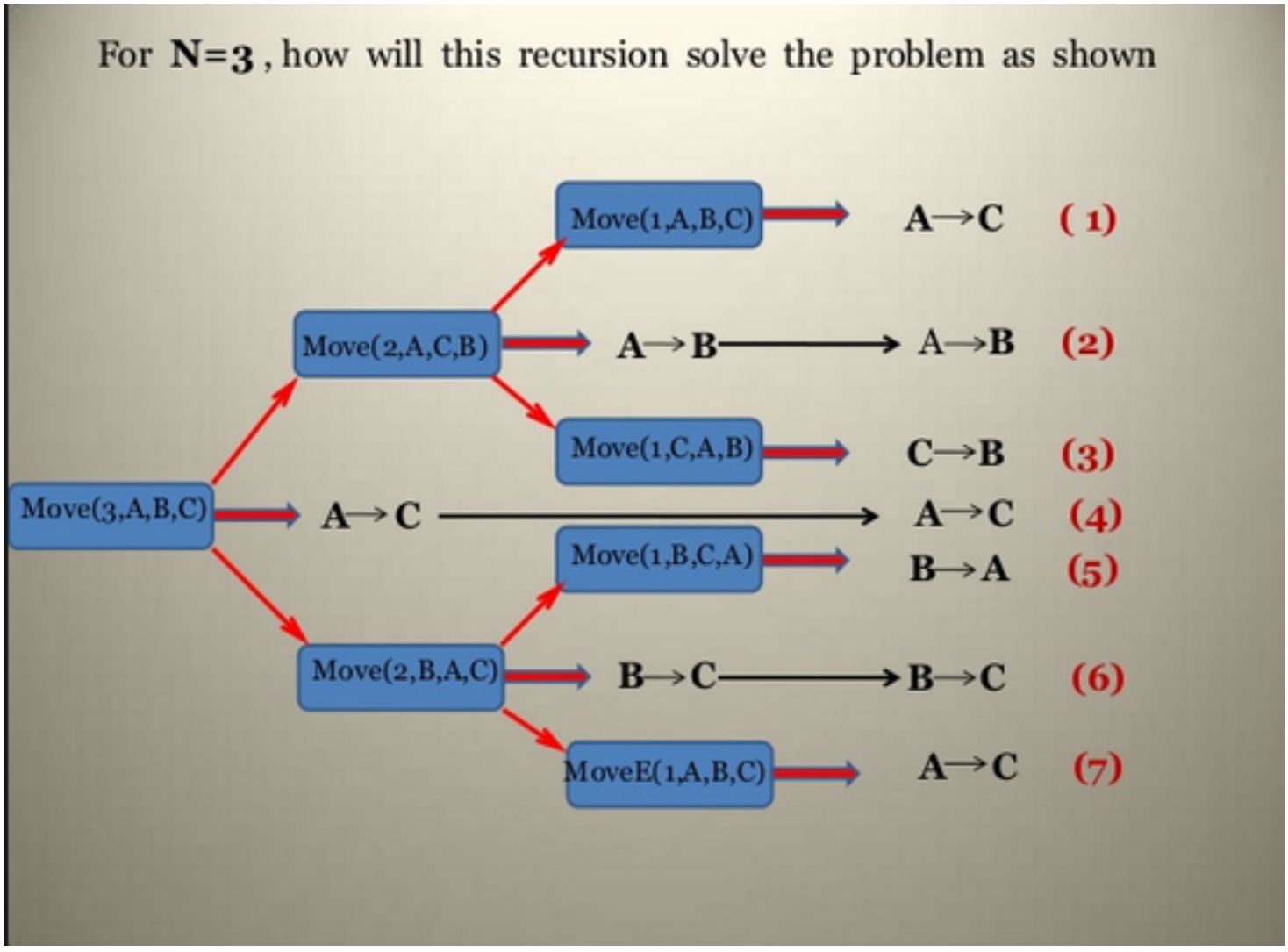
Out put_stack = **GF+ED+C*BA-+-**

Reversing the output_stack we get prefix expression: **-+-AB*C+DE+FG**

TOWER OF HANOI

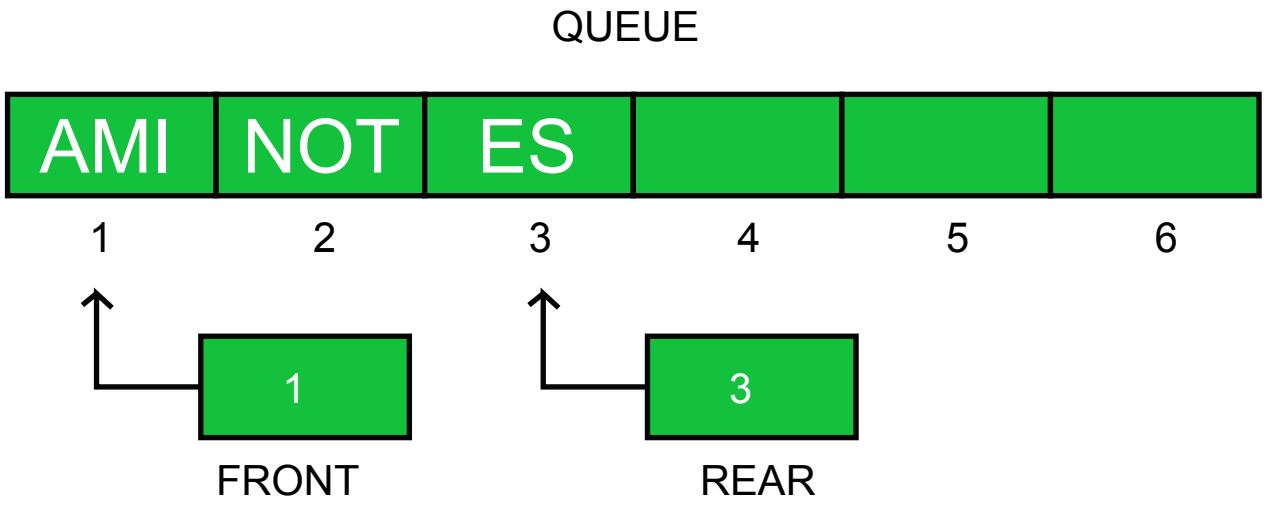
ALGORITHM -
TOWER(N, BEG, AUX, END) for general N
1. If N=1, then:
   (a) Write: BEG -> END.
   (b) Return.
2. Call TOWER(N-1, BEG, END, AUX) [Move N-1 disks from peg BEG to peg AUX].
3. Write: BEG -> END.
4. Call TOWER (N-1, AUX, BEG, END) [Move N-1 disks from peg AUX to peg END].
5. Return.

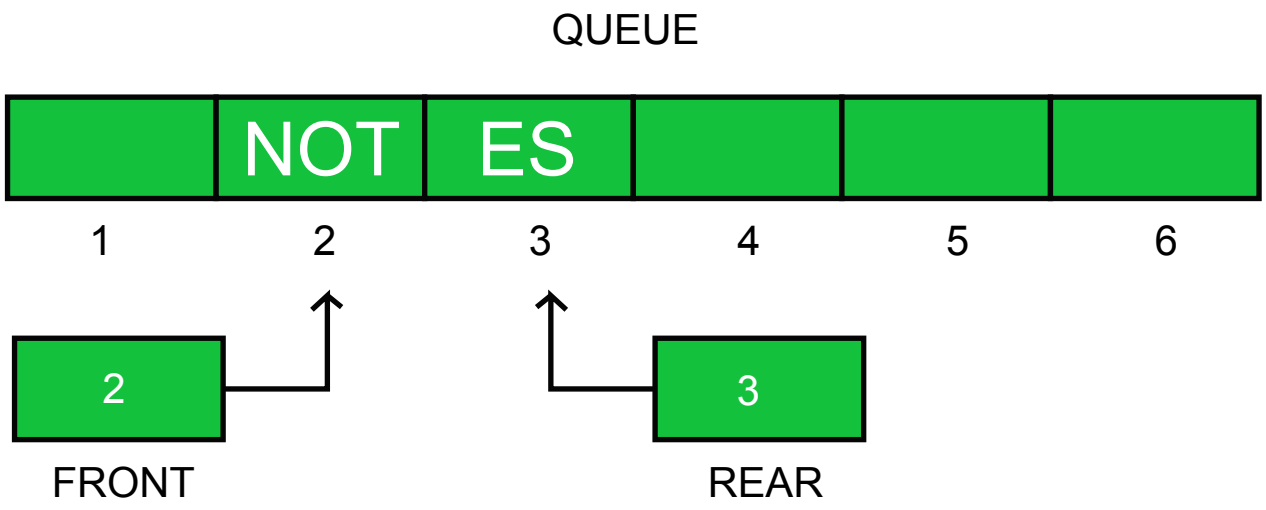
TOWER OF HANOI



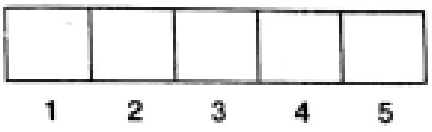
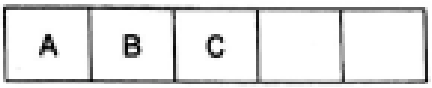
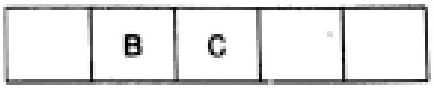
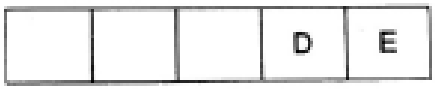
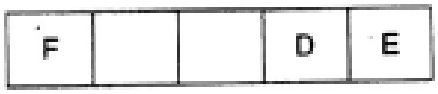
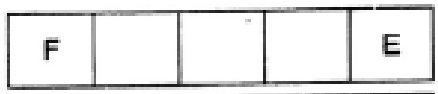
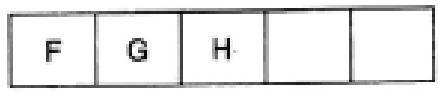
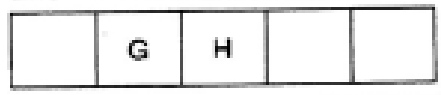
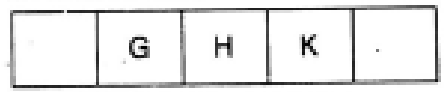
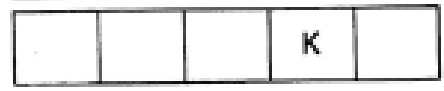
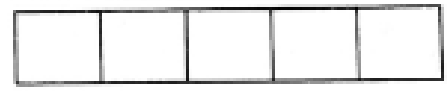
QUEUES

A queue is a linear list of elements in which deletions can take place only at one end, called the front, and insertions can take place only at the other end, called the rear. The terms “front” and “rear” are used in describing a linear list only when it tis implemented as a queue.
Queues are also called first-in-first-out (FIFO) lists.The order in which elements enter a queue is the order in which they leave. This contrasts with stack, which are last-in-first-out (LIFO) lists.





QUEUES

		QUEUE
(a) Initially empty:	FRONT: 0 REAR: 0	
		1 2 3 4 5
(b) A, B and then C inserted:	FRONT: 1 REAR: 3	
(c) A deleted:	FRONT: 2 REAR: 3	
(d) D and then E inserted:	FRONT: 2 REAR: 5	
(e) B and C deleted:	FRONT: 4 REAR: 5	
(f) F Inserted:	FRONT: 4 REAR: 1	
(g) D deleted:	FRONT: 5 REAR: 1	
(h) G and then H inserted:	FRONT: 5 REAR: 3	
(i) E deleted:	FRONT: 1 REAR: 3	
(j) F deleted:	FRONT: 2 REAR: 3	
(k) K inserted:	FRONT: 2 REAR: 4	
(l) G and H deleted:	FRONT: 4 REAR: 4	
(m) K deleted, QUEUE empty:	FRONT: 0 REAR: 0	

QUEUES

INSERTION

LINKQ_INSERT(INFO, LINK, FRONT, REAR, AVAIL, ITEM)
This procedure inserts an ITEM into a linked queue.
1. If AVAIL = NULL then write OVERFLOW and EXIT (Checking space)
2. Set NEW = AVAIL and AVAIL = LINK[AVAIL]
3. Set INFO[NEW] = ITEM and LINK[NEW] = NULL
4. If (FRONT=NULL) then FRONT= REAR = NEW
else set LINK[REAR] = NEW and REAR = NEW
5. Exit

INSERTION

LINKQ_DELETE(INFO, LINK, FRONT, REAR, AVAIL, ITEM)
This procedure deletes the front element of the linked queue and stored it in ITEM
1. If FRONT = NULL then write UNDERERFLOW and EXIT (Checking space)
2. Set TEMP = FRONT
3. ITEM = INFO[TEMP]
4. FRONT = LINK[TEMP]
5. LINK[TEMP] = AVAIL and AVAIL = TEMP
6. Exit