Assignment No. 3

1.) Applications of Linked list :-

Linked lists are used in wide variety of applications because of their exceptional characteristic of expansion and compression according to program requirement. They save wastage of memory and in some cases their response is exceptional.

Some common applications of linked lists are -

- Implementing stack
- Implementing queue
- Implementing non linear structures
- Maintaining directories
- Performing arithmetic operations on long integers
- Manipulating polynomials
- Representing sparse matrices etc.

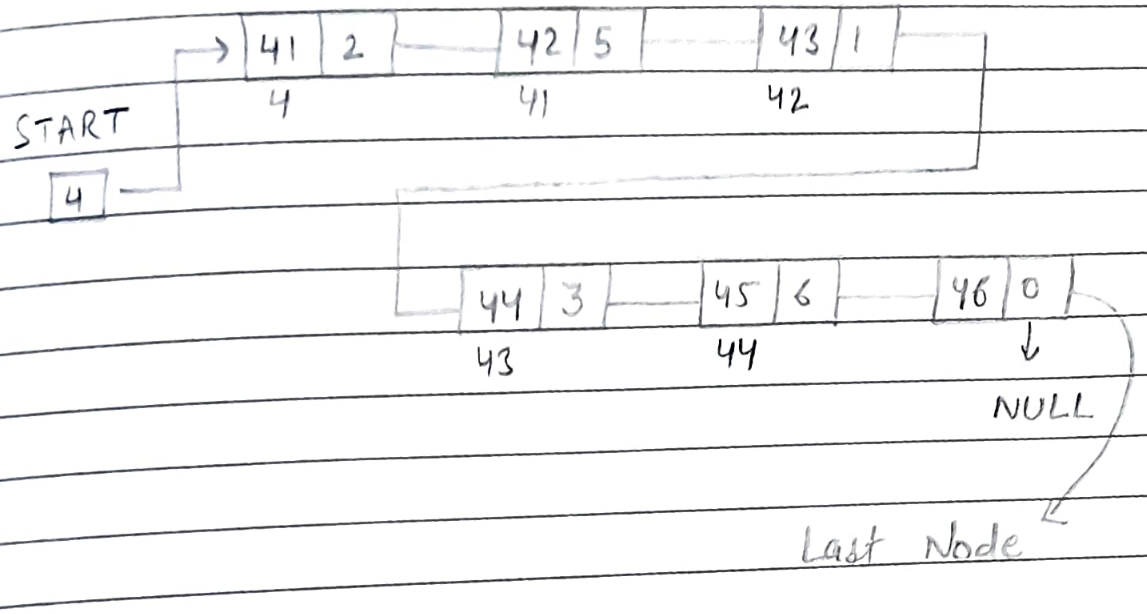
2.) Representation of Linked List in memory -

Consider a linked list called list.

(1) List requires two arrays. One for information part called INFO [K] and other for address

of next element called LINK [K].

(2) List requires three variables Start, Avail and NULL. Start contains address of first element. Avail contains address of first unused space and NULL contains unknown address to end the list. We will choose NULL=0.

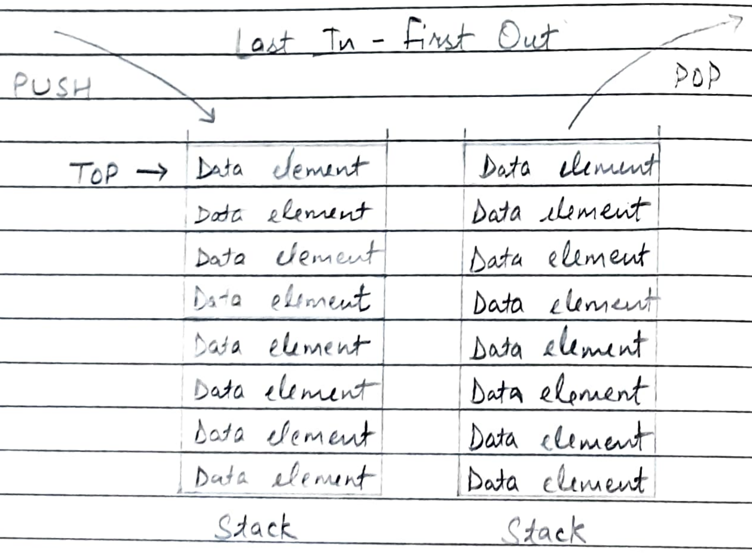


Assignment No. 4

1) Discuss the representation, implementation and application of stacks using suitable algorithms.

Stack - Stack is a list of in which all the insertion and deletion are made at one end only and hence its working is like last in first out. (LIFO)

* Representation of Stack -



Algorithm for PUSH operation -

```
begin procedure push : stack, data

    if stack is full
        return null
    endif

    top ← top + 1

    stack [top] ← data

end procedure
```

* Implementation of Stack -

The stack can be implemented in one of the two ways called

- Array implementation
- Linked list implementation

Algorithm Status - Array

Input : A stack with elements.
Output : States whether it is empty or full, available free space and item at TOP.
Data structure : An array A with TOP as the pointer.

Steps :

```
1.  If TOP < 1 then
2.      Print "Stack is empty"
3.  Else
4.      If (TOP ≥ SIZE) then
5.          Print "Stack is full"
6.      Else
7.          Print "The element at TOP is", A[TOP]
8.          free = (SIZE - TOP) / SIZE * 100
9.          Print "Percentage of free stack is", free
10.     EndIf
11. EndIf
12. Stop
```

* Application of stack -

i) Evaluation of Arithmetic Expressions
ii) Code Generation for stack Machines
iii) Implementation of Recursion
iv) factorial Calculation
v) Quick Sort
vi) Tower of Hanoi Problem
vii) Activation Record Management

Algorithm Move -

Input: Number of discs in the tower of Hanoi N, specification of ORG as from the pillar and DES as to the pillar, and INT as the intermediate pillar.
Output: Steps to moves of N discs from pillar ORG to DES pillar.

Steps :

```
1. If N>0 then          // N=0 is the termination
2.                         condition
3.   Move (N-1, ORG, DES, INT)
4.   ORG → DES (Move from ORG to DES)
5.   Move (N-1, INT, ORG, DES)
6. EndIf
7. Stop
```

2.) Difference between Circular queues and Dequeue :

Circular Queues	Doubly Ended Queues
i) In an queue when elements are arranged in sequential manner but logically we	i) A queue in which inserting and deleting of elements can be done from both the

assume it in circular format, then such queue is called as "Circular Queue".	ends, such queue is called as "Double Ended Queue" (DeQueue).
ii) It follows "First In first Out" method for insertion and deletion of elements	ii) Insertion and deletion of elements can be done from both ends in this
iii) Circular queue is used when we want the iterator to comeback to start when we reach the last element of the queue.	iii) Deque is a queue which stores address of first and last elements. So accessing elements or insertion at front or end will have O(1) complexity.
iv) Circular queue consume less memory.	iv) Double Ended Queue consumes more memory.
ii) It has no specific order of execution.	ii) It follows the FIFO principle in order to perform the tasks.
v) In circular queue, the insertion and deletion can take place from any end.	v) In dequeue, insertion and deletion can be done from both the ends (two).
vi) The memory can be more efficiently used.	vi) The usage of memory in inefficient.