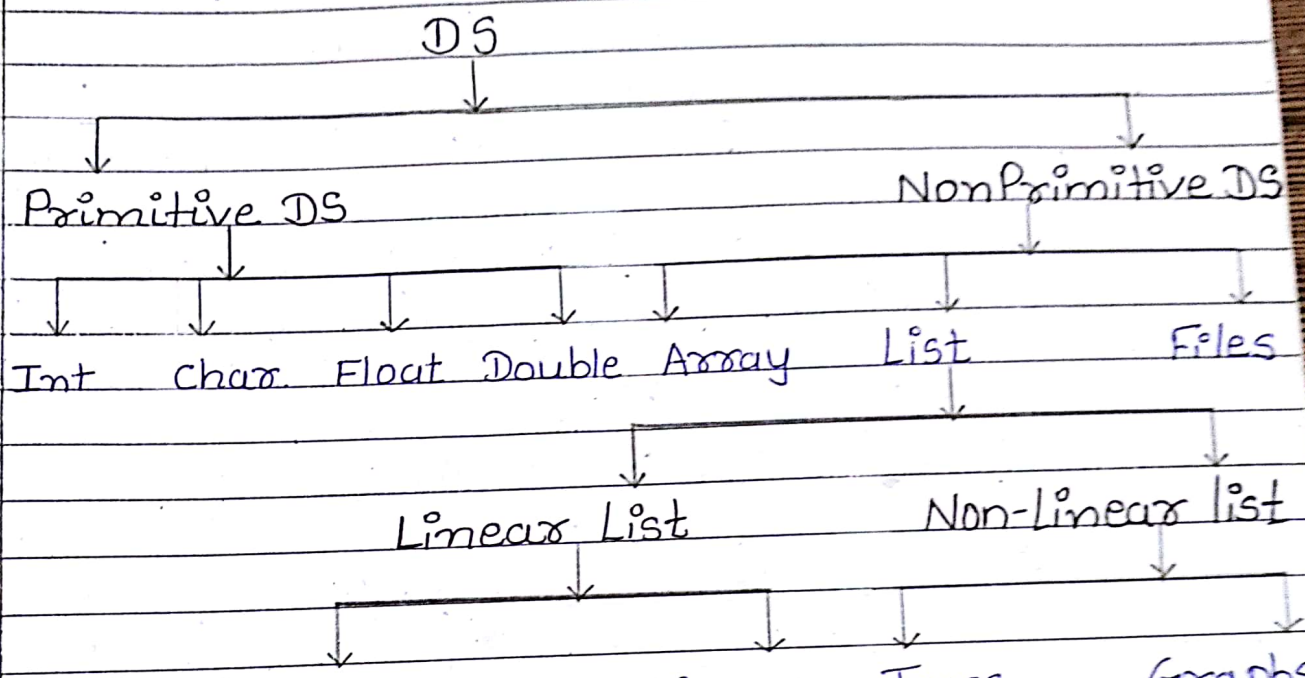
Date: 19-6-18
MON TUE WED THU FRI SAT

Introduction

Data Structure is a particular way of organizing a data in a computer. so it can be used efficiently.



Primitive Data Structure

Primitive Data Structure are provided by a programing language as a basic building block. This are also known as building types or basic type or primary data structure.

Ex: int, ch, float, double, etc

Date:
MON TUE WED THU FRI SAT

Non-Primitive Data Structure (Abstract DS)

The data structure which is derived from primitive data structure are known as Non-Primitive Data Structure.

Ex: Array, Stack, Queue, Union, Structure, Link list, Tree, Graph, etc

NOTE > This all data structure allows to perform different operation on data. We select this DS based on which type of operation is required.

Types of Data Structure

- Linear DS
  - a) Sequencial organization (Array, Stack, Queue)
  - b) Linked Organization (Linked List)
- Non-Linear DS
  - c) Hierachical Relationship (Tree)
  - d) Network Relationship (Graph)

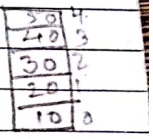
In linear Data Structure the elements are stored in sequencial order it means if we know the address of first data item then we can retay the second one & so on.

Date:
MON TUE WED THU FRI SAT

a) In sequencial organization elements occupy continues or consecutive memory location.

b) In this data structure components are accessed in certain sequence but they are not necessary stored in consecutive memory location. It uses a pointer which shows the relationship between the elements with the help of links.

STACK (LIFO)



Overflow Condition = TOP ≥ Max
Underflow Condition = Top = 0

PUSH : To insert or add the element into the Stack

POP : This Operation delete the elemer from the Stack

PEEP : Peep Operation returns the valu of top most element of the Stack.

Application

→ Reverse the string
→ Expression Conversion
  - Infix → Prefix
  - Infix → Postfix
  - Postfix → Prefix
  - Prefix → Postfix.
→ Parsing.

Disadvantage
→ Slow access to other element

Queue (FIFO)

	1	2	3	4	5
F=0		20	30	40	50
R=0		F			

The element in a queue are added at one end called rear. and remove from the other end called front.

Enqueue → means insefection or addition of element in a queue.
dequeue → deletion or removal of an element in a queue

Simple Queue
Circular Queue
DQueue (Double ended Queue)

Underflow F = 0 } For Simple Queue
Overflow R ≥ max }

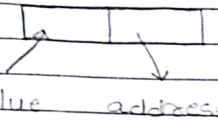
Underflow F = R = 0 } For Circular Queue
Overflow If R ≥ max && F = 1 }

Application
→ CPU task Scheduling
→ Ground rdering
→ Keyboard buffer
→ Queue of request at web Server.

→ Interupts are handle in the same order as they come.

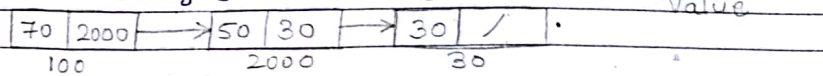
Linked List

1. Singly linked list
2. Doubly linked list
3. Circular linked list

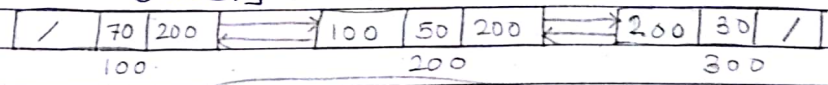


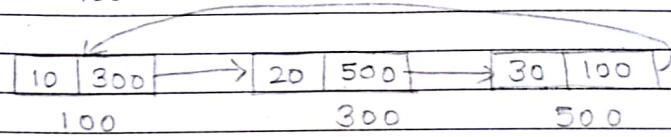
(Dynamic memory allocation)

Singly linked list



Doubly linked list





circular linked list

Linked list is a collection of item in which each item is linked with other and order is given by means of links from one item to another. Each item is denoted by node which consist of two field.

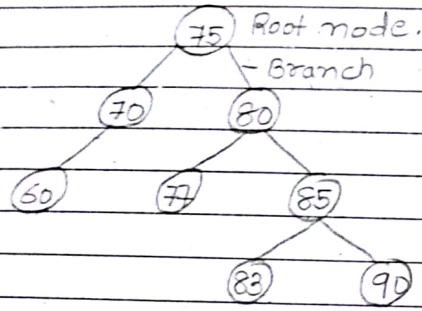
Advantages
- It is a dynamic data structure in which the element can be added or deleted from any where in the list.
- In linked list need not to worry about how many element will be stored in the linked list.
- In linked list each element is allocated memory as it is added to the list

Disadvantage
- It require more memory and slow search operations.
- Reverse traversing is difficult

Application
- It is used to impliment stack, queue, Graph, Trees.

Tree

It is a Non-linear DS in which data items called nodes are arrangeed in some sequence.



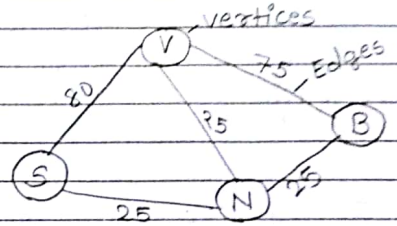
Advantage.
Element can be located very quickly in tree.
Insertion is fast.

Disadvantage
Deletion algorithm is Complex.

Application
Telephone directory assistance. Information stored in a tree so name & no. can be quikly find out.

TREE CONTAIN PURELY PARENT-CHILD RELATIONSHIP.

Graph



It is a basically a collection of vertices and edges that connect this Vertices.
It is a generalization of tree structure where instead of having purely Parent & child relationship betn tree node any kind of complex relationship betn the vertices can be representative.
Tree has a constraint that no one node can have many children but only one parent but in graph it doesn't have such type of restriction.

Application

- Node can represent city & edges can represent road since it can be used in GPS.
- A Graph can also used to represent computer network where nodes are called station & edges are network connection. So it can be used in work station.

Disadvantage

- Some algorithm are slow & very complex.

Performance analysis & Asymptotic notation.

Performance Analysis of an algorithm is a process of calculating space required by that algorithm and time required by that algorithm. It is of an algorithm is performed by using the following measures,

i) Space Complexity
ii) Time Complexity

⇒ Total amount of computer memory required by an algorithm to complete its execution is called as Space Complexity of that algori.

For any algorithm memories required by following purpose

i) Memory required to store program instruction or structure.
ii) Memory required to store constant values.
iii) Memory required to store variable.

Ex: void add (int n)
{
return n+n;
}

for n → 2 bytes
for return → 2 bytes
Static.
Total → 4 bytes.

If algorithm requires a fixed amount of space for all input values then that space complexity said to be Constant Space Complexity.

```
int Sum (int A[], int n)
{
    int Sum=0, i;
    for (i=0; i<n; i++)
        Sum = Sum + A[i];
    return Sum;
}
```

n Space = 2 byte
A[] Space = 2*n byte
sum Space = 2 byte
i Space = 2 byte
return Space = 2 byte
Total = 2n+8 byte

If the amount of space required by an algorithm is increase with increase of input value then the space complexity is said to be linear space complexity.

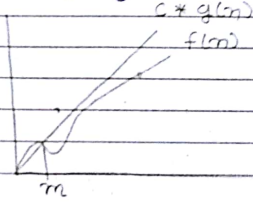
ii) ⇒ Total amount of time required by an algorithm to complete its execution is called as time complexity.
To calculate the time complexity we check only how our program is behaving for other diff. input values to perform all the operation like arithmetic, logical, assignment, any return value operation, etc.

If any program require fixed amount of time for all input values then its time complexity is said to be constant time complexity. and if amount of time required by an algorithm is increase with the increase in value then that type of complexity is said to be linear time complex.

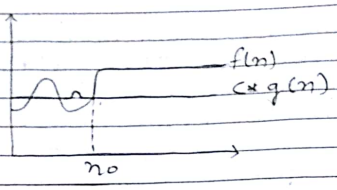
Three types of asymptotic notation

1. Big oh (O) → max. amount of time req. by alg. defines upperbound of alg.
2. Big omega (Ω) → min. amount
3. Big theta (Θ) → Avg.

1. It describes the worst case of an algo.
$f(n) = O(g(n))$
$f(n) <= C * g(n)$ for every $(n \geq n_0)$
C = some real constant
f(n) = algo runtime
g(n) = arbitary time complexity you are trying to relate to your algo.



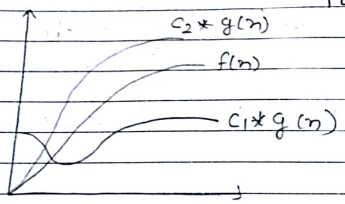
2. It describes the best case of an algori
$f(n) = \Omega * g(n)$
if $f(n) \geq C * g(n)$ for every
i/p size n $(n \geq n_0)$



3.

$0 \le c_1 * g(n) \le f(n) \le c_2 * g(n)$ for all $n \ge n_0$

$c_1, c_2 > 0$



Calculate the time Complexity

```
fun(n)
{
    for(i=1; i<=n; i++)      1   n+1   n
    {
        printf("Hello");     n
    }
}
```

$3n + 2$

$O(n)$

```
fun(n)
{
    for(i=1; i<=n; i=i+2)    1   n/2+1   n/2
    {
        Pf("Hi");            n/2
    }
}
```

$\frac{3n}{2} + 2$ $n = 1$ $n = 5$

```
for(i=1; i<n; i++)           1   n   n-1
{
    for(j=1; j<n; j++)       (n-1)  (n-1)(n)  (n-1)^2
    {
        pf(i*j);             (n-1)*n
    }
}
```

$1 + n + n-1 + n^2 - n + n^2 - n$

$n^2 - 2n + 1 + n^2 - n$

$3n^2 - n$

$O(n^2)$

$n = 5$

$i = 1$

$j = 1$
$j = 2$
$j = 3$
$j = 4$
5

$i = 2$

$i = 3$

$i = 4$
$= n - 1$

$n - 1$

Write an Algorithm to insert an element in simple Queue.

Procedure QINSERT (Q, F, R, N, Y)

F & R = It is a pointer which points to front of rear elements of the Queue.
Q = Queue.
N = Number of element in queue.
Y = element which is to be inserted
initially F & R set to zero

1. [Overflow?]
   if R ≥ N
   then write ('overflow')
   Return

2. [Increment rear Pointer)
   R ← R+1

3. [Insert Element]
   Q[R] ← Y

4. [Is front Pointer Property set?]
   1) F=0 then F ← 1
   Return

1	2	3	4	5
				.

FR.

(i) R=0, Y=5, N=5, F=0

1. [Overflow?]
   0 ≠ 5
   False.
2. R=1
3. Q[1] = 5
4. F = = 0
   F=1.

1	2	3	4	5
5				

FR.

(ii) R=1, Y=10, N=5, F=1.

1. 1 ≥ 5
   False
2. R=2
3. Q[2] = 10
4. F=≠0
   False.

1	2	3	4	5
5	10			

F R

(iii) R=2, Y=15, N=5, F=1

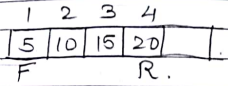
1. 2 ≥ 5
   False
2. R=3
3. Q[3] = 15
4. F = = 0
   False

1	2	3	4	5
5	10	15		

F R

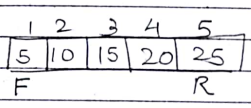
(iv) R=3, Y=20, N=5, F=1

1. 3 >= 5
   False
2. R = 4
3. Q[4] = 20
4. F == 0
   False



(v) R=4, Y=25, N=5, F=1

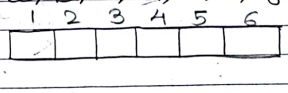
1. 4 >= 5
   False
2. R = 5
3. Q[5] = 25
4. F == 0
   False



(vi) R=5, Y=30, N=5, F=1

1. 5 >= 5
   True
   overflow

Draw a simple queue with 6 memory shell which has Front & Rear at zero position. Draw the simple queue structure & represent the position of front & Rear pointer for each of the foll. operation.
Insert a, b, c, d, e, f, g



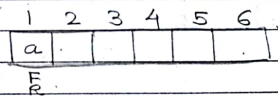
1. R=0, N=6, Y=a, F=0

(i) if R >= N
   → 0 >= 6.
   False.

(ii) R = 1.

(iii) Q[1] = a.

(iv) F == 0.
   → F = 1



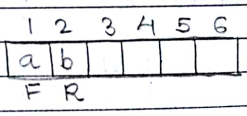
2. R=1, N=6, Y=b, F=1

(i) if R >= N
   → 1 >= 6
   False

(ii) R = 2

(iii) Q[2] = b.

(iv) F == 0
   False.



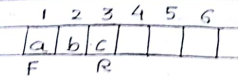
3. R=2, N=6, Y=C, F=1

(i) if R≥N
2≥6.
False

(ii) R=3

(iii) Q[3]=C

(iv) F==0
False



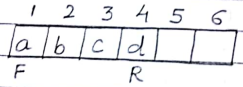
4. R=3, N=6, Y=d, F=1

(i) if R≥N
3≥6
False

(ii) R=4

(iii) Q[4]=d

(iv) F==0
False



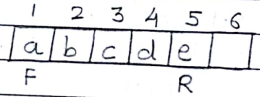
5. R=4, N=6, Y=e, F=1

(i) if R≥N
4≥6
False

(ii) R=5

(iii) Q[5]=e

(iv) F==0
False



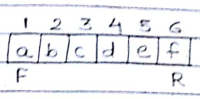
6. R=5, N=6, Y=f, F=1

(i) if R≥N
5≥6
False

(ii) R=6

(iii) Q[6]=f

(iv) F==0
False



7. R=6, N=6, Y=g, F=1

(i) if R≥N

(ii) 6≥6
True

(iii) Overflow

(iv)

(v)

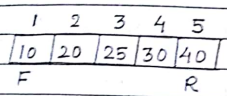
Deletion Algo (DEQUEUE)
Procedure QDELETE (Q, F, R)
Q = Queue
F & R = Pointer which points infront of rear
y = Temporary Variable.

1. [Underflow?]
if F=0
then write ('UNDERFLOW')
Return(0)

2. [Delete element]
   $y \leftarrow Q[F]$

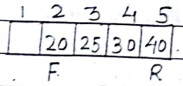
3. [Queue empty?]
   if F == R
   then $F \leftarrow R \leftarrow 0$
   else
   $F \leftarrow F+1$

4. [Return element]
   Return y

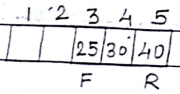


Deletion

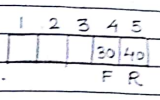
1. F=1, R=5
   $1 \neq 0$
   False
   $y = Q[1]$
   $y = 10$
   $1 \neq 5$ False
   $F = 1+1 = 2$
   $y = 10$



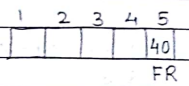
2. F=2, R=5
   $2 \neq 0$
   False
   $y = Q[2]$
   $y = 20$
   $2 \neq 5$ False
   $F = 2+1 = 3$
   $y = 20$



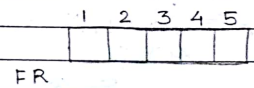
3. F=3, R=5
   $3 \neq 0$
   False
   $y = Q[3]$
   $y = 25$
   $3 \neq 5$ False
   $F = 3+1 = 4$
   $y = 25$



4. F=4, R=5
   $4 \neq 0$
   False
   $y = Q[4]$
   $y = 30$
   $4 \neq 5$ False
   $F = 4+1 = 5$
   $y = 30$

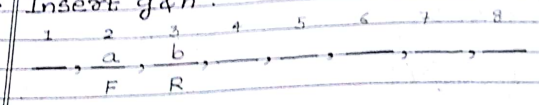


5. F=5, R=5
   $5 \neq 0$
   False
   $y = Q[5]$
   $y = 40$
   $5 == 5$
   True
   F=0, R=0
   $y = 40$



Consider the Simple Queue given below with 8 memory shell. which has F=2, R=3. Draw the Simple Queue structure & represent the position of Front & Rear pointers for each of the foll. operation

1) Insert c & d
2) Delete one alphabet
3) Insert e & f
4) Delete one alphabet
5) Insert g & h.



1.

a) F=2, R=3, N=8, Y=c
3 ≠ 8
False
R = 3+1 = 4
Q[4] = y
y = c
2 ≠ ∅
False

b) F=2, R=4, N=8, Y=d
4 ≠ 8
False
R = 4+1 = 5
y = Q[5]
y = d
2 ≠ ∅
False

2. F=2, R=5
2 ≠ ∅
False
y = Q[2]
y = a
2 ≠ 5
False
F = 2+1 = 3
y = a

3. a) F=3, R=5, N=8, Y=e
5 ≠ 8
False
R = 5+1 = 6
y = Q[6]
y = e
3 ≠ ∅
False

b) F=3, R=6, N=8, Y=f
6 ≠ 8
False
R = 6+1 = 7
y = Q[7]
y = f
3 ≠ ∅
False

4. F=3, R=7
   3 ≠ 0
   False
   y = Q[3]
   y = b
   3 ≠ 7
   F = 3+1 = 4
   y = b

5. a) F=4, R=7, N=8, Y=g
   7 ≥ 8
   False
   R = 7+1 = 8
   y = Q[8]
   y = g
   4 ≠ 0
   False

b) F=4, R=8, N=8, Y=h
   8 ≥ 8
   True
   Overflow

Procedure QINSERT (F, R, Q, N, Y)
F & R → Front & Rear pointer
Q → Queue
N → No. of element in queue
Y → element to be inserted

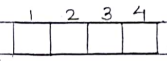
1. [Reset Rear Pointer?]
   if R = N
   then R ← 1
   else
   R ← R+1

2. [Overflow]
   if F = R
   then write ('Overflow')
   Return

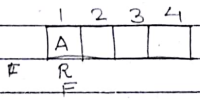
3. [Insert element]
   Q[R] ← y

4. [Is Front Pointer]
   F = 0
   then F ← 1
   Return

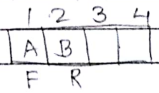
N=4, Y = A, B, C, D



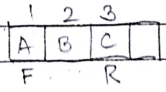
0 ≠ 4, R=1, Q[1]=A.



1 ≠ 4, R=2, Q[2]=B



2 ≠ 4, R=3, Q[3]=C



3 ≠ 4, R=4, Q[4]=D

Date.:
MON TUE WED THU FRI SAT

1	2	3	4
A	B	C	D

F R

4 = 4, R = 1 1 = 1
'overflow'

1	2	3	4
A	B	C	D

F R

Procedure [QDELETE (F, R, Q, N)

1. [underflow]
   if F = 0
   then write ('UNDERFLOW').
   Return(0)

2. [Delete element]
   y ← Q[F]

3. [Queue empty?]
   if F = R
   F ← R ← D
   Return(y)

4. [Increment front Pointer]
   if F = N
   F ← 1
   else
   F ← F + 1
   return(y)

Date.:
MON TUE WED THU FRI SAT

Operation

1. Delete A
2. Insert E
3. Delete B
4. Insert F
5. Delete C, D, E, F

1	2	3	4
A	B	C	D

F R

1) 1 ≠ 0
   y = Q[1]
   A = y
   1 ≠ 4.
   F = F + 1
   F = 2

1	2	3	4
	B	C	D

F R

2) 4 = 4 True
   R = 1
   2 ≠ 1 False
   Q[1] = E
   2 ≠ 0.

1	2	3	4
E	B	C	D

R F

3) 2 ≠ 0
   y = Q[2]
   y = B
   2 ≠ 1
   F = F + 1
   F = 3

1	2	3	4
E	B	C	D

R F

Date:
MON TUE WED THU FRI SAT

4) $1 \neq 4$
$R = R+1$
$R = 2$
$Q[2] = F$
$3 \neq 0$

1	2	3	4
E		C	D
R	F		

5) a) $3 \neq 0$
$y = Q[3]$
$y = C$
$3 \neq 2$
$F = F+1$
$F = 4$.

b) $4 \neq 0$
$y = Q[4]$
$y = D$
$4 \neq 2$
$F = N \quad 4 = 4$ True
$F = 1$

c) $1 \neq 0$
$y = Q[1]$
$y = E$
$1 \neq 2$
$F \neq N \quad 1 \neq 4$
$F = F+1$
$F = 2$

Date:
MON TUE WED THU FRI SAT

d) $2 \neq 0$
$y = Q[2]$
$y = F$
$2 = 2$
$F \leftarrow R \leftarrow 0$
$F = R = 0$, $F = 1$

$N = 6, F = 0, R = 0$
1. Insert 5, 10, 15, 20, 25
2. Delete 5 & 10
3. Insert 30, 35, 40
4. Delete 15, 20, 25
5. Insert 45, 50, 55
6. Delete 30, 35, 40, 45, 50, 55

1. a) $F = 0, R = 0, N = 6, y = 5$
$0 \neq 6$
$R = R+1$
$R = 1$
$Q[1] = 5$
$F = 0, F = 1$

1	2	3	4	5	6
5					
R					
F					

b) $F = 1, R = 1, N = 6, y = 10$
$1 \neq 6$
$R = R+1$
$R = 2$
$Q[2] = 10$
$F = 1$

1	2	3	4	5	6
5	10				
F	R				

c) F=1, R=2, N=6, Y=15
2 ≠ 6
R = R+1
= 3.
Q[3] = 15,
F=1

d) F=1, R=3, N=6, Y=20
3 ≠ 6.
R = R+1
= 4
Q[4] = ~~15~~ 20
F=1

e) F=1, R=4, N=6, Y=25.
4 ≠ 6
R = R+1
= 5
Q[5] = 25
F=1

2. 1 ≠ 0
y = Q[1]
y = 5.
1 ≠ 5
1 ≠ 6
F = 1+1
= 2

2 ≠ 0
y = Q[2]
y = 10
2 ≠ 5
2 ≠ 6
F = 2+1
= 3.

3. F=3, R=5, N=6, Y=30.
5 ≠ 6.
R = 6
3 ≠ 6
Q[6] = 30
3 ≠ 0

F=3, R=6, N=6, Y=35.
6 = 6
R ← 1
3 ≠ 1
Q[1] = 35.
3 ≠ 0

F=3, R=1, N=6, Y=40
1 ≠ 6
R = 2
3 ≠ 2
Q[2] = 40.
3 ≠ 0

4. F=3, R=2, N=6
3≠0
y=Q[3]
y=15
3≠2
F=F+1
F=4

F=4, R=2, N=6
4≠0
y=Q[4]
y=20
4≠2
F=4+1
F=5

F=5, R=2, N=6
5≠0
y=Q[5]
y=25
5≠2
F=5+1
F=6

5. F=6, R=2, N=6
2≠6
R=3
6≠0
Q[3]=

(task scheduling)

Priority Queue

A priority queue is an abstract data type in which each element is assigned a priority. The priority of the element will be used to determine the order in which this element will proceed. The general rule of processing the element of priority queue is

1. An element with higher priority is processed before an element with a lower priority.
2. Two element with the same priority are processed based on FirstCome First Served bases.

A Queue in which elements are inserted or removed according to some priority is reffered to as a priority queue.

Priority queue are widely used in operating system to execute the highest priority process first.

Implementation of Priority Queue

There are two ways
Sworted list
Unsorted list

Sorted list

O(n) ⇒ Insertion
O(1) ⇒ Deletion

3	10	15	25	45
			↑	
			f	

Unsorted list

O(1) ⇒ Insertion
O(n) ⇒ Deletion

Representation of Priority Queue

1. Link Representation of Priority queue
2. Array Representation of Priority queue.

1. Link Representation

R_1	R_2	...	R_{i-1}	O_1	O_2	---	O_{j-1}	B_1	B_2	---	B_{k-1}
1	1	---	1	2	2	---	2	3	3	---	3

Priority 1 | R_1 | R_2 | ---- | R_{i-1} |

Priority 2 | O_1 | O_2 | ---- | O_{j-1} |

Priority 3 | B_1 | B_2 | ---- | B_{k-1} |

If a Sequential Storage Structure is used for the priority queue then insertion of new element must be placed in the middle of the structure

This can require the moment of several items. It is better to split the priority queue into different different queue. Each having its own storage structure.

2. Array Representation

Priority Queue

Priority	Front	rear
1	3	3
2	1	3
3	4	5
4	4	1

Front & Rear Position

Prio.	1	2	3	4	5
1			A		
2	B	C	D		
3				E	F
4	I			G	H

P-1

1	2	3	4	5
		A		

P-2

1	2	3	4	5
B	C	D		

When an array are used to implement a priority queue then a separate queue for each priority number is maintained. Each of this queue will be implemented using circular queue.

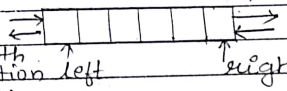
Here two dimensional array is used where each queue will be allocated the same amount of space.
To insert a new element with priority 'K' in the priority queue add the new element at the rear end of row 'K'.
K = row number as well as priority no. of that element.

To delete an element we find the first non-empty queue & then process the front element of the non-empty queue.

Double Ended Queue (DQueue)

Types:
1. Input restricted dqueue
2. Output restricted dqueue

A dqueue is a linear list of elements in which element may be inserted or deleted from the both end which is called as double ended queue or head-tail link list.



both insertion + deletion are possible

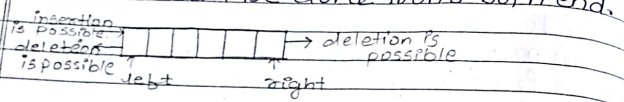
left

right

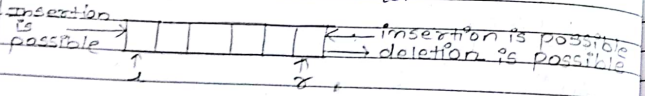
both insertion + deletion are possible

No element can added or deleted from the middle. In Computer memory dqueue is implemented as circular array or circular doubly link list.

1. ⇒ In input restricted dqueue insertion can be done only at one end. but deletion can be done from both end.



2. ⇒ In this dqueue deletion can be possible at only one end. & insertion can be possible from both end.

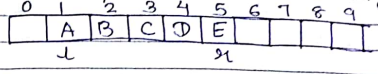


Consider a dqueue given below with 10 memory shells which has $l = 1$ & $r = 5$.

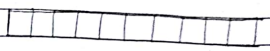
Draw the dqueue structure & represent position of left & right pointer for each of the foll. operation

1. Insert F on the left
2. Insert G & H on the right
3. Delete two alphabet from the left.
4. Insert I on the right.
5. Insert J on the left.

6. Delete two alphabet from right



1. Insert F on left
   $l--$
   $l = 0$ $dq[l] = F$



2. Insert G & H on right
   $r++$
   $r = 6$
   $dq[r] = G$
   $r++$
   $r = 7$
   $dq[r] = H$

3. Delete 2 alphabet from left.
   $l = 0$
   $F = y = dq[l]$
   $l = 1$
   $A = y = dq[l]$

4. Insert I on right
   $r++$
   $r = 8$
   $dq[r] = I$

5. Insert J on left
   $l--$
   $l = 1$
   $dq[l] = J$

Date: MON TUE WED THU FRI SAT

6. Delete two alphabet from right

r--

I = y = dq[r]

r--

H = y = dq[r]

0	1	2	3	4	5	6	7	8	9
	J	B	C	D	E	G	H		

If there is only one element into dqueue & if deletion operation is perform then after deletion set l & r to '0'.

Ap:

Date: 31-7-18
MON TUE WED THU FRI SAT

Linked list

A link list is a simple term & it is a linear collection of data element.

A list is a collection of items in which each item is linked with other & order is given by means of links from one item to another. Each item is denoted by node which consist of two field

1) Value of the node
2) Address of next node
3) and the last node contain the null in its address path.

Advantages of Array

1. Array is very simple to use & easy to create.
2. No memory management is needed.
3. In Array we can access any element with the help of index directly.
4. Array are quick to loop.

Disadvantages of Array.

1. Array is static in nature.
Memory is allocated at compile time.
2. Insertion & deletion of element is more complecated in comparison to the link list.

Advantages of Linked List.

1. It is a dynamic data structure therefore the size of the link list can grow & shrink in size during the execution of program.
2. In link list programmer didn't worry about how many element will be stored in a link list.
3. In a link list each element is allocated memory as it is added to the list.

Disadvantages of Linked list

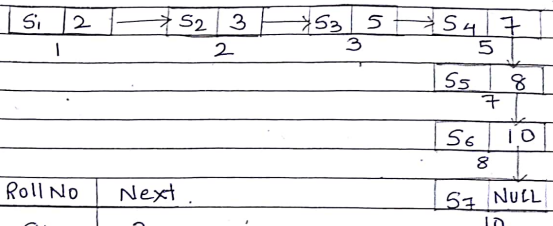
1. Lots of over head. (Costs of malloc call & pointer assignment)
2. We must traverse the entire list to go to the nth node.
3. It consume extra space to contain the address of next element of string in the list.

Application of Linked List.

1. Stack, Queue, Graph, Tree are implimented using link list.
2. Undo functionality, photoshop or word.
3. Consider the history section of web browser where it creates the link list of web pages visited. So when you press the back button the previous node data is fatched.

"Every node contain a pointer to another node which is of the same type". It is called as self refrencial data type."

Representation of Link list in memory.



	Roll No	Next
1 First →	S1	2
2	S2	3
3	S3	5
4 Avail →		Avail → 6
5	S4	7
6		9
7	S5	8
8	S6	10
9		-1
10	S7	-1

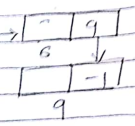
First = 1

4

6

9

	RollNo	Next
First→1	S_1	2
2	S_2	3
3	S_3	5
4	S_8	-1
5	S_4	7
Avail→6		
7	S_5	8
8	S_6	10
9		
10	S_7	4



When we delete a row from the linked list then operating system changes the status of memory occupied by it from memory available.
The Computer maintain a list of all the free memory shell. The list of available shell is called as free pool.
Linked list is a por

Link List as Pointer Variable

1) Which Store the address of the first node of the list.
Likewise for the free pool which is the linked list of all free memory shell we have a pointer variable 'Avail' which store the address of the first free space.
2) If we delete node then space occupied by that node must be given back to the free pool. So memory can be reused by some other program.
3) The operating System does this task of adding & removing memory from the free pool.
This task is perform when O.S finds the CPU ideal or the program are falling sort of memory.
The O.S scan through all the memory shell & marks those cell that are being used by some other program. Then it collect all the cell which are not being used & at their address to free pool. This process is called garbage collection.

Singly Linked List

1) Algorithm for Insertion at beginning

FIRST = INSERT(X, FIRST)
X = A new element which is to be inserted.
FIRST = A pointer to the first element to linked list whose typicall node contain info & Link.
New is a temporary variable.
Avail is a pointer to the top element of the availability stack