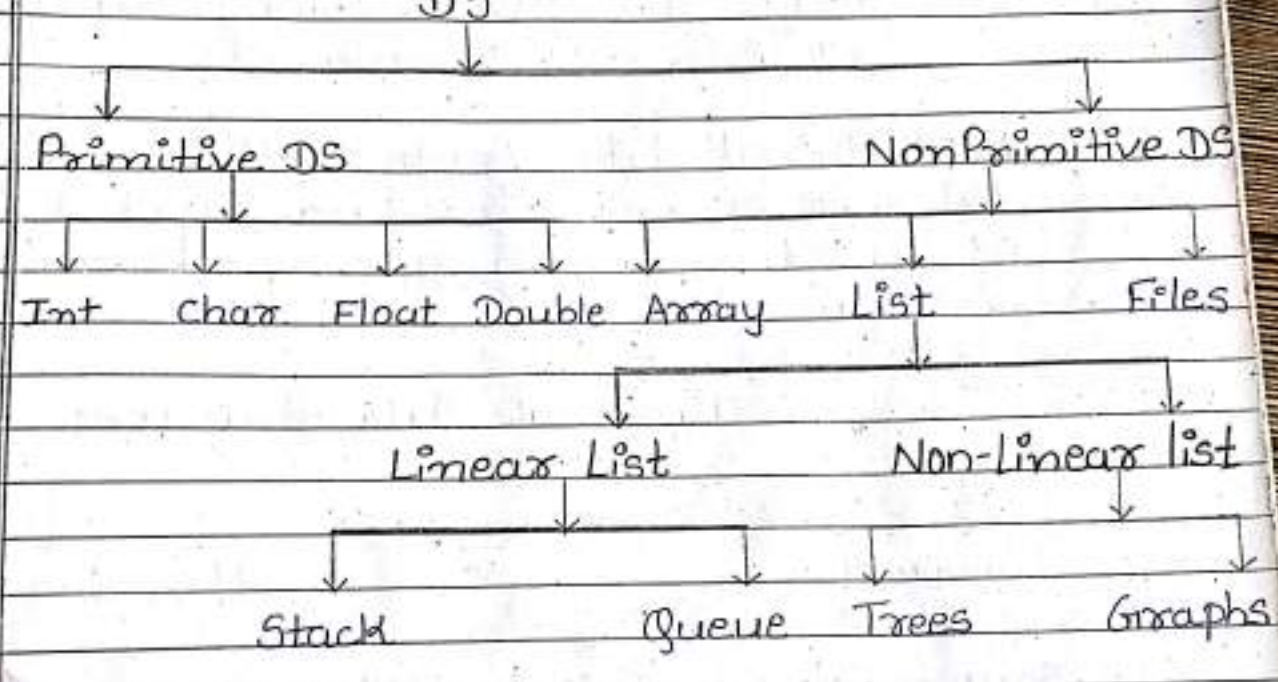
Date: 19-6-18
MON TUE WED THU FRI SAT

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 programming 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

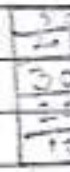
- Linear DS
  - a) Sequencial organization (Array, Stack, Queue)
  - b) Linked Organization (Linked List)
- Non-Linear DS
  - c) Hierachical Relationship (Tree)
  - d) Network Relationship (Graph)

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 retry the second one & so on.

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

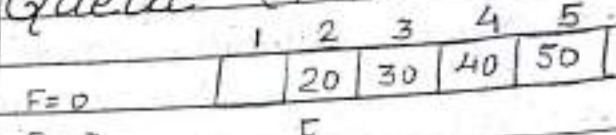
POP : This Operation delete the element from the Stack

PEEP : Peep Operation returns the value 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)



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
Overflow R ≥ max
} For Simple Queue

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

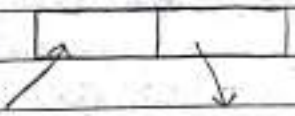
Application
→ CPU task Scheduling
→ Ground robering
→ Keyboard buffer
→ Queue of request at Web Server.

Date:
MON TUE WED THU FRI SAT

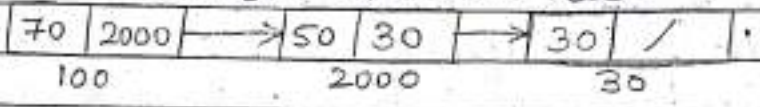
→ Interupts are handle in the same order as they come.
Linked List ..

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

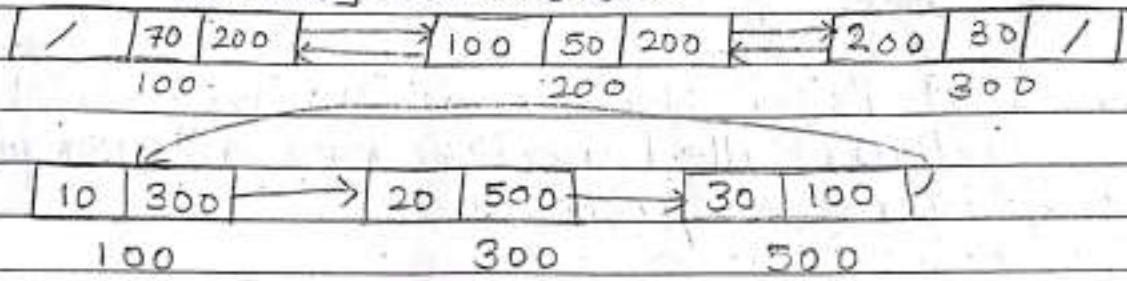
(Dynamic memory allocation)



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 Programmer 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

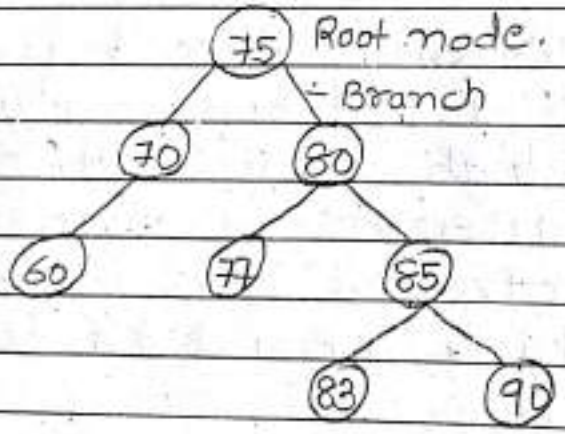
Date:
MON TUE WED THU FRI SAT

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 arranged in some sequence.



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

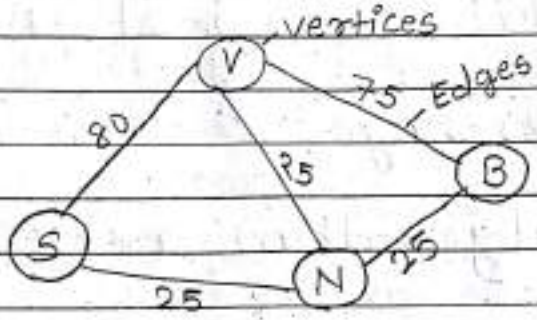
Disadvantage
Deletion algorithm is Complex.

Date.:
MON TUE WED THU FRI SAT

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.

Date.:
MON TUE WED THU FRI SAT

Application
- Node can represent city & edges can represent road hence 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 & Asympotic notation.

Performance Analysis of an algorithm is a process of calculating space required by that algorithm and time required by that algorith P.A of an algorithm is performed by using the foll. measures
i) Space Complexity
ii) Time Complexity

i) ⇒ 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
State.
Total ⇒ 4 bytes.

If algorithm require 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.

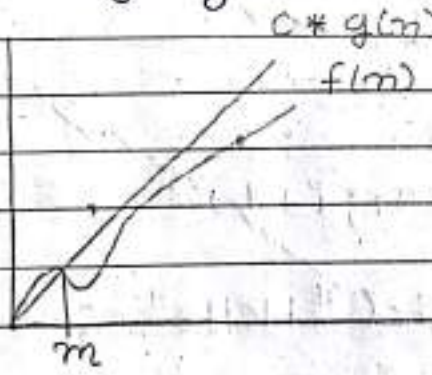
If any program require fixed amount of time for all input values then its time complexity is said to be Constant time complexity. and ~~th~~ if amount

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.

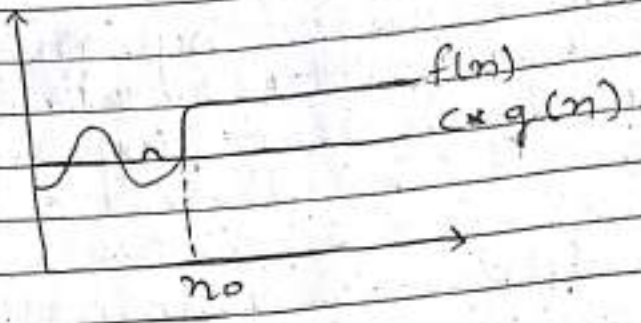
1 Three types of asympatic notation

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.

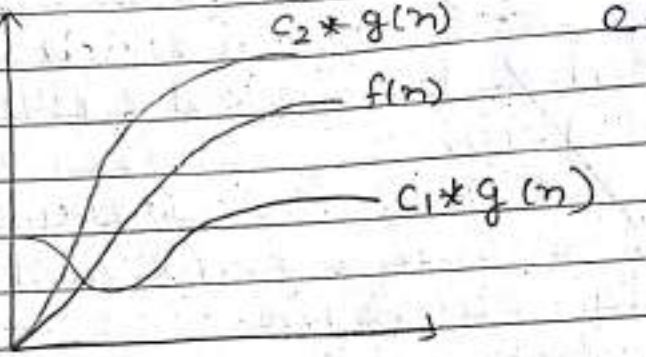
1. It describes the worst case of an algo.
   $f(n) = O(g(n))$
   $f(n) <= C * g(n)$ for every $(n \geq n_0)$
   C = Some real constant
   $f(n)$ = algo runtime
   $g(n)$ = arbitary time complexity you are trying to relate to your algo.



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 $(n > n_0)$



3. $0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ for all $n \geq n_0$, $c_1, c_2 > 0$



Calculate the time Complexity

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

$3n + 2$

$O(n)$

Date.:
MON TUE WED THU FRI SAT

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

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

$n = 5$

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

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

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

$3n^2 - n$

$O(n^2)$

$n = 5$

$i = 1$

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

$i = 2$

$i = 3$

$i = 4$

$= n - 1$

$n - 1$

Date:
MON TUE WED THU FRI SAT

Write an Algorithm to insert an element in Simple Queue.

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

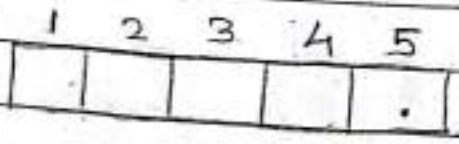
F & R = It is a pointer which points to front of rear elements of the Queue.
Q = Queue.
N = Number of element in queue.
Y = element which is to be inserted
initially F & R Set to zero

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

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

3. [Insert Element]
   Q[R] ← Y

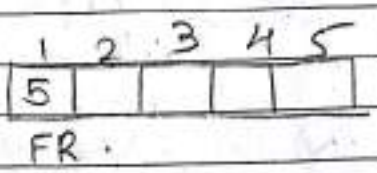
4. [Is front Pointer Property Set?]
   1) F=0 then F ← 1
   Return



Date:
MON TUE WED THU FRI SAT

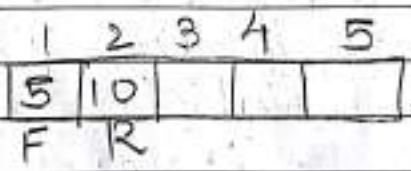
(i) R=0, Y=5, N=5, F=0

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



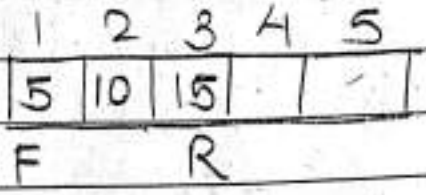
(ii) R=1, Y=10, N=5, F=1

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



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

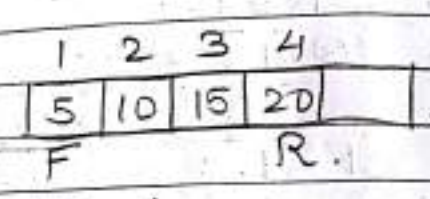
1. 2 ≥ 5
   False
2. R=3
3. Q[3]=15
4. F==0
   False



Date:
MON TUE WED THU FRI SAT

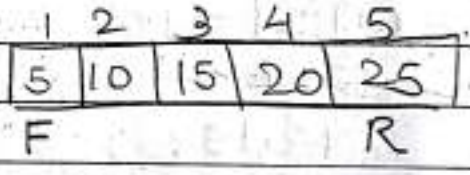
(iv) R=3, Y=20, N=5, F=1

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

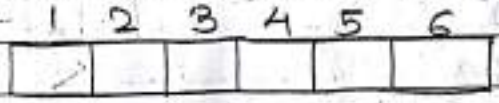


(Vi) R=5, Y=30, N=5, F=1

1. 5 >= 5
   True
   Overflow

Date.:
MON TUE WED THU FRI SAT

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



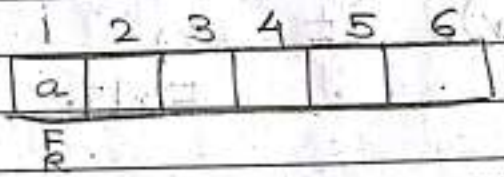
1. R=0, N=6, Y=a, F=0

(i) if R ≥ N
→ 0 ≥ 6.
False.

(ii) R=1.

(iii) Q[1] = a.

(iv) F==0.
→ F=1



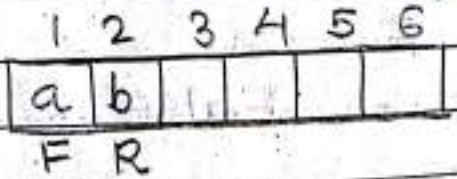
2. R=1, N=6, Y=b, F=1

(i) if R ≥ N
→ 1 ≥ 6
False

(ii) R=2

(iii) Q[2]=b

(iv) F==0
False.



Date:
MON TUE WED THU FRI SAT

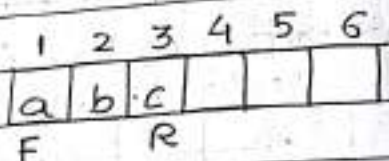
3. R=2, N=6, Y=C, F=1

(i) if R>N
2>6
False

(ii) R=3

(iii) Q[3]=C

(iv) F==0
False



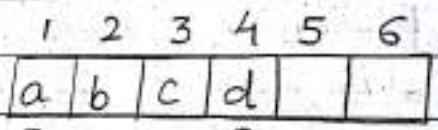
4. R=3, N=6, Y=d, F=1

(i) if R>N
3≯6
False

(ii) R=4

(iii) Q[4]=d

(iv) F==0
False



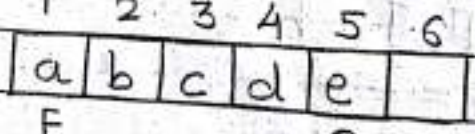
5 R=4, N=6, Y=e, F=1

(i) if R>N
4≯6
False

(ii) R=5

(iii) Q[5]=e

(iv) F==0
False



Date.:
MON TUE WED THU FRI SAT

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

1	2	3	4	5	6
a	b	c	d	e	f
F					R

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')

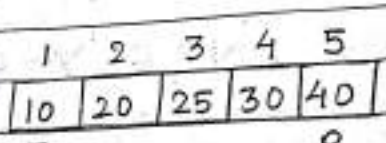
Return (0)

Date:
MON TUE WED THU FRI SAT

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

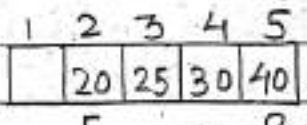
3. [Queue empty?]
   if F == R
      then $F \leftarrow R \leftarrow 0$
   else
      $F \leftarrow F+1$

4. [Return element]
   Return y

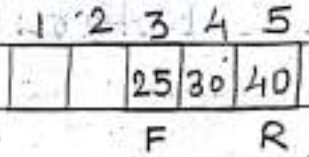


Deletion

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,

4. F=

5.

Date:
MON TUE WED THU FRI SAT

3. F=3, R=5

$3 \neq\neq 0$

False

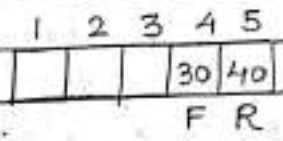
y = Q[3]

y = 25

$3 \neq\neq 5$ False

F = 3+1 = 4

y = 25



4. F=4, R=5

$4 \neq\neq 0$

False

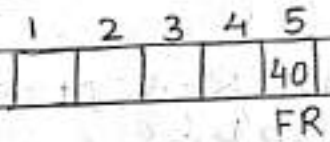
y = Q[4]

y = 30

$4 \neq\neq 5$ False

F = 4+1 = 5

y = 30



5. F=5, R=5

$5 \neq\neq 0$

False

y = Q[5]

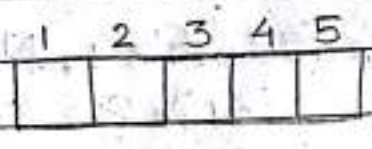
y = 40

5 == 5

True

F=0, R=0

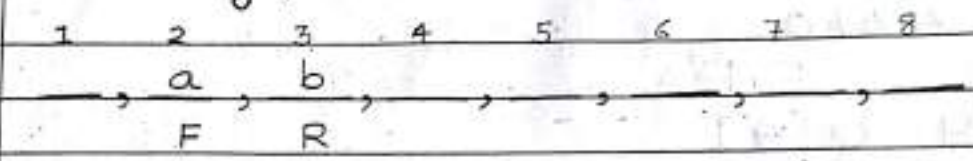
y = 40



Date:
MON TUE WED THU FRI SAT

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 ≠ ≠ 0

False

b) F=2, R=4, N=8, Y=d

4 ≯ 8

False

R = 4+1 = 5

y = Q[5]

y = d

2 ≠ ≠ 0

False

Date:
MON TUE WED THU FRI SAT

2. F=2, R=5

2 ≠ ≠ 0

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 ≠ ≠ 0

False

b) F=3, R=6, N=8, Y=f

6 ≠ 8

False

R = 6+1 = 7

y = Q[7]

y = f

3 ≠ ≠ 0

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

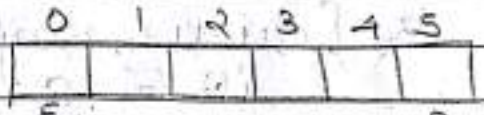
Date:
MON TUE WED THU FRI SAT

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

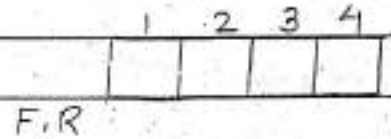
2. [overflow]
   if F = R
   then write ('overflow')
   Return

3. [Insert element]
   Q[R] ← y

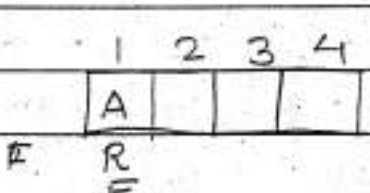
4. [Is Front Pointer]
   F = 0
   then F ← 1
   Return



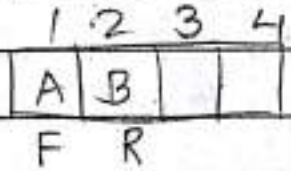
N = 4, Y = A, B, C, D



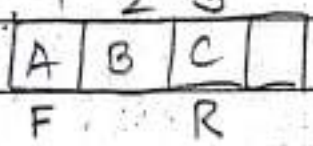
0 ≠ 4, R = 1, Q[1] = A



1 ≠ 4, R = 2, Q[2] = B

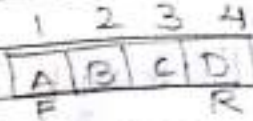


2 ≠ 4, R = 3, Q[3] = C

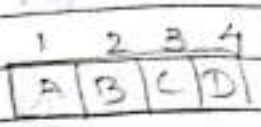


3 ≠ 4, R = 4, Q[4] = D

Date:
MON TUE WED THU FRI SU



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



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

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

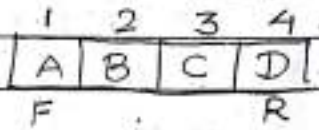
2. [Delete element]
   $y \leftarrow Q[F]$

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

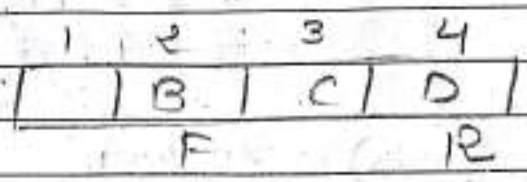
4. [Increment front Pointer]
   if F = N
   $F \leftarrow 1$
   else
   $F \leftarrow F + 1$
   return(y)

Operation

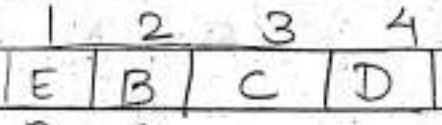
1. Delete A
2. Insert E
3. Delete B
4. Insert F
5. Delete C, D, E, F



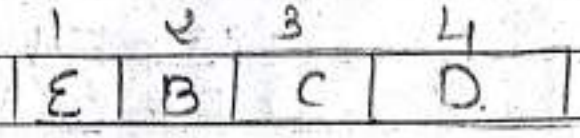
1) $1 \neq 0$
$y = Q[1]$
$A = y$
$1 \neq 4$.
$F = F + 1$
$F = 2$



2) $4 = 4$ True
$R = 1$.
$2 \neq 1$ False
$Q[1] = E$
$2 \neq 0$.



3) $2 \neq 0$
$y = Q[2]$
$y = B$
$2 \neq 1$
$F = F + 1$
$F = 3$



Date:
MON TUE WED THU FRI

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

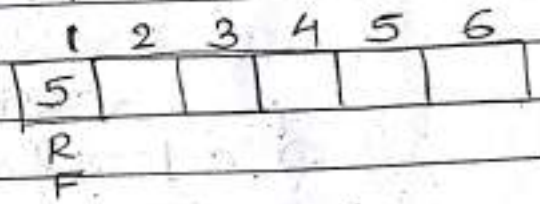
1
2.
3.
4
5.
6.

1.

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

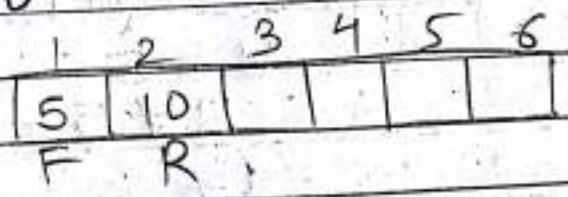
$N = 6, F = 0, R = 0$
1. Insert 5, 10, 15, 20, 25
2. Delete 5 & 10
3. Insert 30, 35, 40
4. Delete 15, 20, 25
5. Insert 45, 50, 55
6. Delete 30, 35, 40, 45, 50, 55

1. a) $F = 0, R = 0, N = 6, Y = 5$
$0 \neq 6$
$R = R + 1$
$R = 1$



$Q[1] = 5$
$F = 0$, $F = 1$

b) $F = 1, R = 1, N = 6, Y = 10$
$1 \neq 6$
$R = R + 1$
$R = 2$



$Q[2] = 10$
$F = 1$

c) F=1, R=2, N=6, Y=15

$2 \neq 6$

$R = R+1$

$= 3$

$Q[3] = 15$

$F = 1$

d) F=1, R=3, N=6, Y=20

$3 \neq 6$

$R = R+1$

$= 4$

$Q[4] = 20$

$F = 1$

e) F=1, R=4, N=6, Y=25

$4 \neq 6$

$R = R+1$

$= 5$

$Q[5] = 25$

$F = 1$

2.

$1 \neq 0$

$y = Q[1]$

$y = 5$

$1 \neq 5$

$1 \neq 6$

$F = 1+1$

$= 2$

3.

Date:
MON TUE WED THU FRI SAT

$2 \neq 0$

$y = Q[2]$

$y = 10$

$2 \neq 5$

$2 \neq 6$

$F = 2 + 1$

$= 3$.

3. $F = 3, R = 5, N = 6, Y = 30$.

$5 \neq 6$.

$R = 6$

$3 \neq 6$

$Q[6] = 30$

$3 \neq 0$

$F = 3, R = 6, N = 6, Y = 35$.

$6 = 6$

$R \leftarrow 1$

$3 \neq 1$

$Q[1] = 35$.

$3 \neq 0$

$F = 3, R = 1, N = 6, Y = 40$

$1 \neq 6$

$R = 2$

$3 \neq 2$

$Q[2] = 40$.

$3 \neq 0$

Date:
MON TUE WED THU FRI SAT

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 before 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

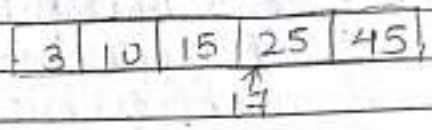
There are two ways

i) Shorted list
ii) Unsorted list

Date:
MON TUE WED THU FRI SAT

Sorted list

O(n) ⇒ Insertion
O(1) ⇒ Deletion



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_i	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

	1	2	3	4	5
P-1			A		

	1	2	3	4	5
P-2	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

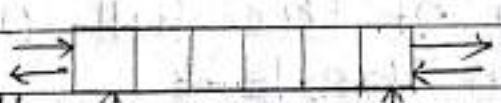
Here two dimensional array is used where each queue will be allocated the same amount of space.
To insert a new element with priority 'k' in the priority queue add the new element at the rear end of row 'k'.
K = row number as well as priority no. of that element.

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

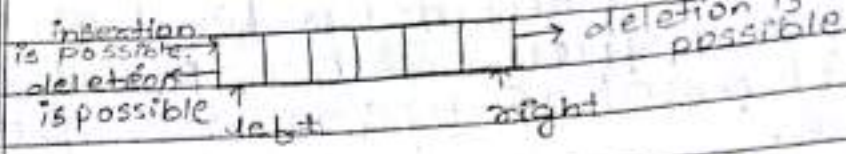
left

right

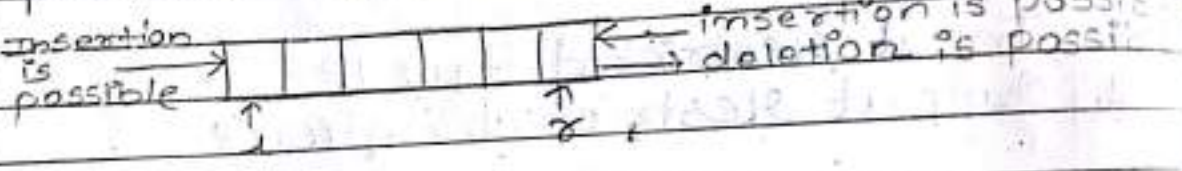
both insertion + deletion are possible

No element can added or deleted fr... the middle. In Computer memory dqueue is implemented as circula... 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 possib... at only one end. & insertion can be possible from both end.

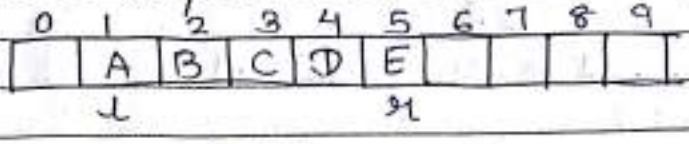


Consider a dqueue given below with 10 memory cells which has $l = 1$ & $r = 5$.
Draw the dqueue structure & represent position of left & right pointer for each of the foll. operation

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.

Date:
MON TUE WED THU FRI SAT

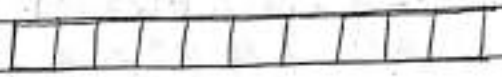
6. Delete two alphabet from right



1. Insert F on left

l--

l=0 dq[l] = F

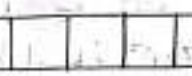


2. Insert G & H on right

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--

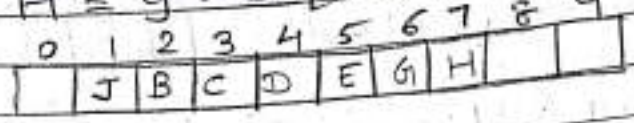
l=1

dq[l] = J

6. Delete two alphabet from right

```
r--
I = y = dq[r]

r--
H = y = dq[r]
```



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.

Date:
MON TUE WED THU FRI SAT

Advantages of Linked List.

1. It is a dynamic data structure there fore 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. (Lots of malloc call & pointer assignment)
2. We must traverse (scan) 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.

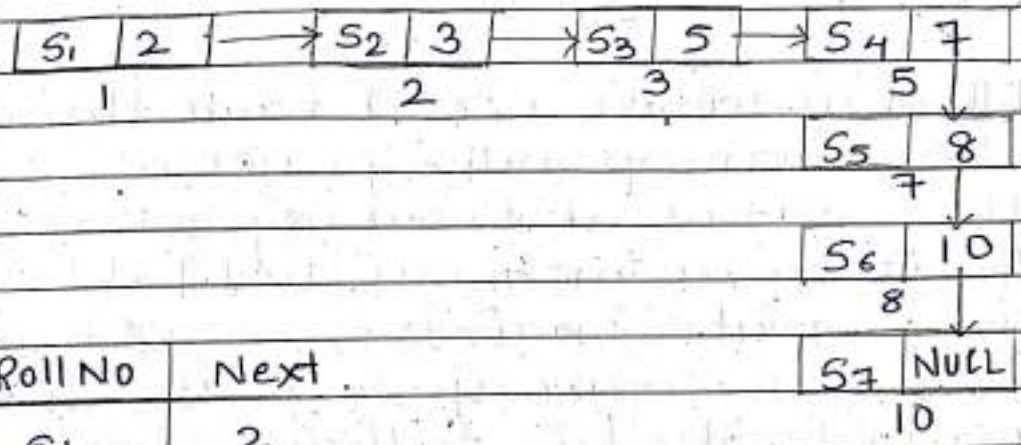
Application of Linked List.

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

botton the previous node data is fatched.

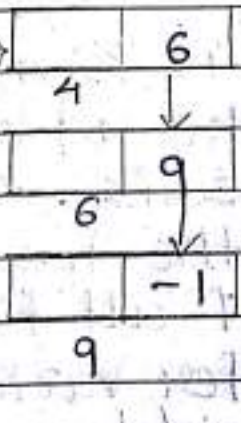
"Every node contain a pointer to another node which is of the same type". It is called as self refrencial data type."

Representation of link list in memory

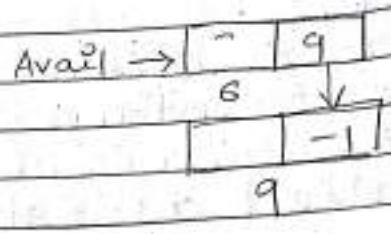


	Roll No	Next
1 First →	S1	2
2	S2	3
3	S3	5
4 Avail →		
5	S4	7
6		
7	S5	8
8	S6	10
9		
10	S7	-1

First = 1



	Roll No	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 po~~

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

(10, NULL)

1] [underflow?]
If Avail = Null
then write ('AVAILABILITY STACK EMPTY)
Return (FIRST)

2] [obtain address of next free node]
New ← Avail

3] [Remove free node from availability Sta
Avail ← Link (Avail)

4] [Initialize fields & new node & its link in the list]
INFO (NEW) ← x
LINK (NEW) ← FIRST

5] [Return address of new node]
Return (NEW)

Insert 10, 20, 30, 40, 50, 60
x First

1. First = (10, NULL)

200 ≠ NULL
New ← 200
Avail ← 250
INFO (New) ← 10
Link (New) ← Null New = 200
First ← 200

2. E

3.

4.

2. First = (20, 200)
250 ≠ NULL
New ← 250
Avail ← 300
Info (New) ← 20
link (New) ← 200 NEW = 250
First ← 250

3. First = (30, 250)
300 ≠ NULL
New ← 300
Avail ← 350
Info (New) ← 30
link (New) ← 250 NEW = 300
First ← 300

4. First = (40, 300)
350 ≠ NULL
New ← 350
Avail ← 400
Info (New) ← 40
link (New) ← 300 NEW = 350
First ← 350

5. First = (50, 350)
400 ≠ NULL
New ← 400
Avail ← NULL
Info (New) ← 50
link (New) ← 350 NEW = 400
First ← 400

2) Algorithm for insertion at end.

FIRST = INSERT (x, FIRST)

1] [underflow?]
AVAIL = NULL
then write ('Availability stack empty')
return (FIRST)

2] [obtain address of next free node]
New ← Avail

3] [Remove free node from availability stack]
Avail ← LINK (Avail)

4] [initialize fields of new node]
Info (New) ← x
LINK (New) ← NULL

5] [Is the list empty?]
If first = NULL
then Return (NEW)

6] [Initialize search for the last node]
SAVE ← FIRST

7] [search for end of LIST]
Repeat while LINK (SAVE) ≠ NULL
SAVE ← Link (SAVE)

8] [Set LINK Field of last node to NEW]
LINK (SAVE) ← NEW

9] [Return FIRST node pointer]
Return (FIRST)

```
temp = first
while (temp != NULL)
{
    pf ("%d", *temp);
    temp = link (temp);
}
```

Insert a node in sorted order

FIRST = INSORD (X, FIRST)

1) [underflow ?]
if AVAIL = NULL
then write ('Availability stack underflow')
Return (FIRST)

2) [obtain address of next free node]
NEW ← AVAIL

3) [Dom. Remove node from availability stack]
AVAIL ← LINK (AVAIL)

4) [Copy information contents into New node]
INFO(NEW) ← x

5) [Is the list empty?]
if FIRST = NULL
then LINK(NEW) ← NULL
Return (NEW)

6) [Does the new node precede all others in the list?]
if INFO(NEW) ≤ INFO(FIRST)
then LINK(NEW) ← FIRST
Return(NEW)

7) [Initialize temporary pointer]
SAVE ← FIRST

8) [Search for Predecessor of new node]
Repeat while LINK(SAVE) ≠ NULL and
INFO(LINK(SAVE)) < INFO(NEW)
SAVE ← LINK(SAVE)

9) [Set Link fields of new node & its predecessor]
LINK(NEW) ← LINK(SAVE)
LINK(SAVE) ← NEW

10) [Return FIRST node pointer]
Return(FIRST)

SSG 18

Date:

Delete an element from Singly link-list.

Procedure : DELETE (X, FIRST)

1) [Empty list ?]
   If FIRST = NULL
   then write ('Underflow')
   Return.

2) [Initialize Search for X]
   Temp ← FIRST

3) [Find X]
   Repeat through Step-5 while.
   INFO (TEMP) ≠ x and LINK (TEMP) ≠ NULL
   ↳ (If we take x as address then Temp ≠ x)

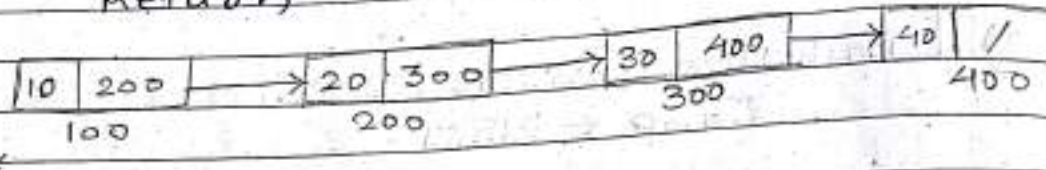
4) [Update Predecessor Markers]
   PRED ← TEMP.

5) [Move to next node]
   TEMP ← LINK (TEMP)

   add below step if x is value
   [T1 ← TEMP]

6) [End of the list?]
   If INFO (TEMP) ≠ x (If x as address Temp ≠ x)
   then write ('NODE NOT FOUND').
   Return.

7) [DELETE X] [If x is address X=FIRST]
If X = INFO (FIRST)
then FIRST ← LINK (FIRST)
else
LINK (PRED) ← LINK (TEMP)
8) [node to availability area]
Return LINK(X) ← AVAIL
AVAIL ← T
Return



Copy of Link List

The new list is to contain node whose information and pointer fields are denoted by field and Ptr respectively. address of first node in newly created list is to be placed in begin.

1) [Empty list?]
if FIRST = NULL
then Return (NULL)

2) [Copy FIRST node]
if AVAIL = NULL
then write (Availability Stack underflow)
Return(0).

3) else
NEW ← AVAIL
AVAIL ← LINK(AVAIL)
FIELD(New) ← INFO(FIRST)
BEGIN ← New

3) [Initialize traversal]
Save ← FIRST

4) [Move to next node if not at end of list.]
Repeat through Step.6 while
LINK(Save) ≠ NULL

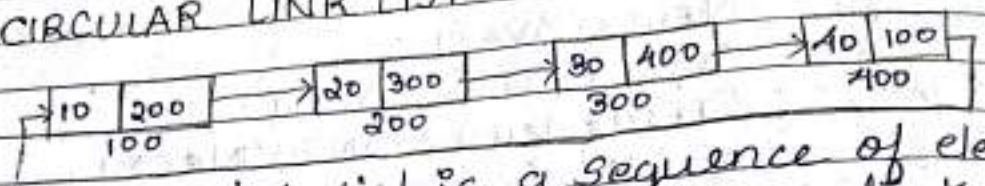
5) [Update predecessor and Save pointers]
PRED ← New
Save ← LINK(Save)

6) [Copy node].
if AVAIL = NULL
then write ('Availability Stack Underflow')
Return(0)
else NEW ← AVAIL
AVAIL ← LINK(AVAIL)
FIELD(NEW) ← INFO(SAVE)
PTR(PRED) ← New

7) [Set line of list node & return]
PTR(New) ← NULL
RETURN (BEGIN)

Date:
MON TUE WED THU FRI SAT

CIRCULAR LINK LIST



Circular link list is a sequence of element in which every element has linked to its next element in the sequence and the last element has a link to its first element.

Advantages

In a circular link list every node is accessible from a given node that is from this given node all nodes can be reached by chaining through the list.

Disadvantages

The only downside of the circular link list is the complexity of iteration.

Algorithm

→ Insert a node at the front/head of a circular link list

FIRST = CINSERT (x, FIRST)

Date:
MON TUE WED THU FRI SAT

① [underflow]
if Avail = Null
then write ('Availability stack underflow')
Return (FIRST)

② [obtain address of next free node]
NEW ← AVAIL

③ [Remove free node from availability stack]
AVAIL ← LINK(AVAIL)

④ [Initialize fields of new node]
INFO (NEW) ← X

⑤ [check for empty list]
if FIRST = NULL
then FIRST ← NEW.
LINK(NEW) ← FIRST
Return NEW.

⑥ [Set PTR]
PTR ← FIRST

⑦ [check for last node]
while LINK (PTR) != FIRST
PTR ← LINK(PTR)

⑧ [Add new node at front]
LINK (NEW) ← FIRST
LINK (PTR) ← NEW. FIRST ← NEW
Return FIRST

→ Insert a node at the end of circular link list

FIRST = CINSERT (X, FIRST)

① [underflow]
if AVAIL = NULL
then write ('Availability stack underflow')
Return (FIRST)

② [obtain address of next free node]
NEW ← Avail

③ [Remove free node from availability stack]
Avail ← LINK (Avail)

④ [Initialize fields of new node]
INFO(NEW) ← X

⑤ [check for empty list]
if FIRST = NULL
then FIRST ← NEW
LINK[NEW) ← FIRST
Return NEW

⑥ [Set LINK pointer of new node]
LINK (NEW) ← FIRST

⑦ [Set PTR]
PTR ← FIRST

Date:
MON TUE WED THU FRI SAT

⑧ [check for last node]
while LINK(PTR) != FIRST
PTR ← LINK(PTR)

⑨ [Add new node at end]
LINK(PTR) ← NEW
RETURN (FIRST)

→ Delete a node from front.

FIRST = CDELETE (FIRST)

① [check for empty list]
if FIRST = NULL
then write ('LIST IS EMPTY')
Return (FIRST)

② [check for only one in LIST]
if FIRST = LINK(FIRST)
then Return (NULL)

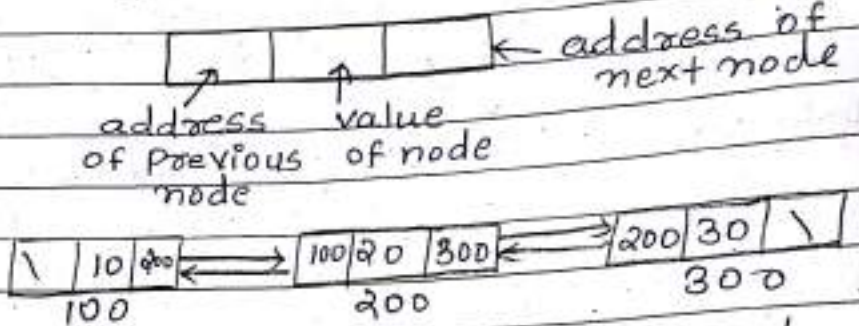
③ [Set temp variable]
temp ← FIRST

④ [check for last node]
while LINK(temp) != FIRST
then temp ← LINK(temp)

⑤ [Set Pointer]
FIRST ← LINK(FIRST)
LINK(temp) ← FIRST

⑥ [Return first node Pointer]
Return (FIRST)

Doubly linked list:



A Doubly link list Contain two link field. The links are used to denote predecessor and successor of node. The link denoting the predessor of a node is called as left link & the node denoting the sucessor is calle as right link.

Definition: A Doubly link list is a Sequence of element in which every element has a link to its previous & next element in the sequence

Advantages

1) We can traverse in both direction from starting to end as well as end to starting.
2) It is easy to reverse the link list

3) If we are at one node then we can go to any node which is not possible in Singly link list.

4)

Disadvantage.

1) It require more space per node because one extra field is required for pointer for previous node.
2) Element are access sequencial.

Application.

1) To implement tree.
2) Browser catch which allows you to with the back botton
3) A Music player which has next & previous botton.
4) Undo & Redo functionality in word.

Insert a node into doubly link list.

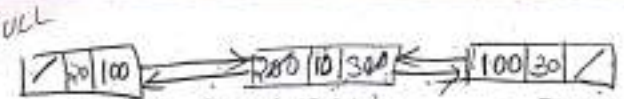
Procedure : DOUBINS (L, R, M, X)
L & R = pointer variable denoting the left most & Right most in the list respectively.
M = The insertion is to be performed to the left of a specified node whose address is given by the pointer variable M.
X = Element to be Inserted

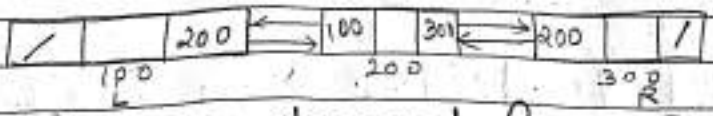
Info → Information field of a node
LPTR → Left & Right link of a node
RPTR
Initially L & R = Null

1) [obtain a new node from the availability stack]
   New ← Node
2) [copy information field]
   Info (New) ← x
3) [Insertion into an empty list ?]
   if R = Null
then LPTR(New) ← RPTR(New) ← NULL
   L ← R ← New
   Return
4) [Left-most insertion]
   if M = L
   then LPTR(New) ← Null
   RPTR(New) ← M
   LPTR(M) ← New
   L ← New
   Return

if M = R
RPTR(New) ←
LPTR(New) ←
RPTR(M) ← N
Return

5) [Insertion in middle]
   LPTR(New) ← LPTR(M)
   RPTR(New) ← M
   LPTR(M) ← New
   RPTR(LPTR(New)) ← New
   Return





Delete an element from Doubly Link List

Procedure : DOUBDEL (L, R, OLD)

OLD = Address of a node which is to be deleted

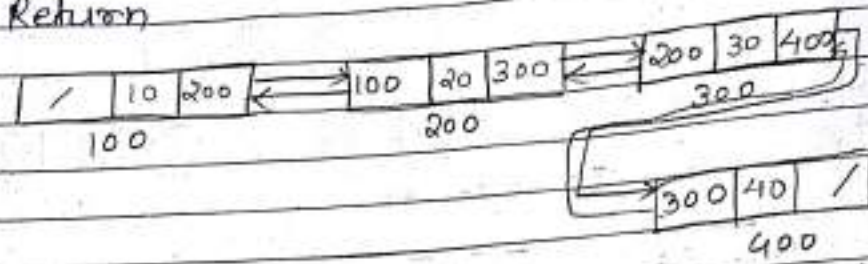
① [underflow?]
if R = Null
then write ('Underflow')
Return

② [Delete node]
if L = R (Single node in List)
then L ← R ← NULL
else if OLD = L
L ← RPTR(L) (Left-most node of Linked List)
LPTR(L) ← NULL
else if OLD = R (Right most node)
then R ← LPTR(R)
RPTR(R) ← NULL
else
RPTR(LPTR(OLD)) ← RPTR(OLD)
LPTR(RPTR(OLD)) ← LPTR(OLD)

③ [Return deleted node]
Restore (OLD)

Date:
MON TUE WED THU FRI SAT

Return



Advantages of Doubly linked List for deletion of a node over singly link list

1) Deletion of a node in a singly link list we have to find the predecessor of the discarded group & the search was performed by chasing through Successive node

2) The search was necessary in order to change the link of the predecessor node to a value that would point to the sucessor of the node which is to be deleted.

3) Such search would be very time consuming depending on the number of deletion & no. of nodes in the linked list.

4) In doubly linked list no such search is required simply given the address of a node which is to be deleted & the predecessor & sucessor are the immediately node so doubly linked list are much more efficient w.r.t.

Sing

Appli

1) Polyn

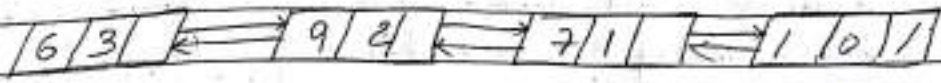
Date:-
MON TUE WED THU FRI SAT

Singly linked list.

Application of Linked List

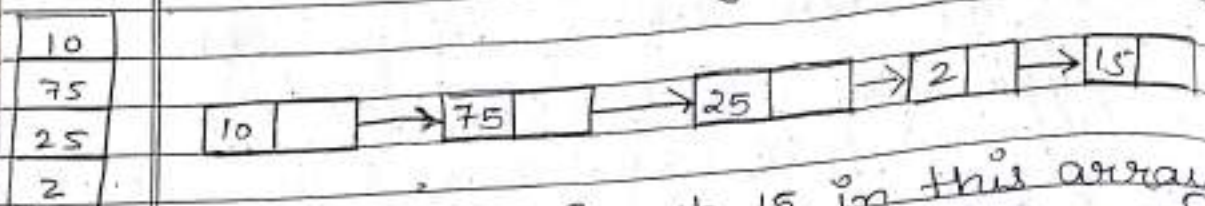
1) Polynomial Representation.

$6x^3 + 9x^2 + 7x + 1$



Date:
MON TUE WED THU FRI SAT

Hashing



If I want to search 15 in this array I have to traverse whole array then my searching Complexity is O(n)

→ The Complexity depends on the total no. of elements in that data structure so it is not desirable for large data set.

→ To deal with this issue Hashing is introduce. Hashing is one approach in which time require to search an element doesn't depend on the no. of element using hashing data structure an element is search with constant time Complexity.

Hashtable

Hash Table is a data structure in which keys or elements are mapped to an array position by a hash function.

Date:
MON TUE WED THU FRI SAT

Hashing works like
H(key) = location
H = hash fun^n
Key = Value to be stored
location = Index value which is derived from hash fun^n

196, 182, 144, 147, 293, 1395, 1495
Hash table size = 10
Hash fun^n = Key mod n

Hash Value	Key
0	
1	
2	182
3	293
4	144
5	1395
6	196
7	147
8	
9	

← actual element

196 mod 10
= 6
182 mod 10
= 2
O(1).

Key → Hash fun^n → hash value
↑ elements which is to be stored
↑ index or location or hash value

Here at position 5 1395 is already stored tha so if we insert 1495 then old value is overwritten. So this is the problem of collision.

Hashing : The process of mapping the keys to appropriate location in a hash table is called hashing

Suppose we have a set of strings {"abc", "def", "ghi"} & I would like to store in hash table.
Suppose hash table has a size = 5.

0	1	2	3	4
def	abc			ghi

abc = 6 6 mod 5 = 1
def = 15 15 mod 5 = 0
ghi = 24 24 mod 5 = 4

a	1
b	2
c	3
d	4
e	5
f	6
g	7
h	8
i	9

Hash Function

A Hash fun^n is simply a mathematical formula which applied to a key produce an integer which can be used as a index or location for the key in the Hash Table. The main aim of a Hash fun^n is that element should be relatively, randomly & uniformly distributed. To achieve a good hashing mechanism it is important to have a good hash fun^n with foll. characteristics

1. Easy
2. Dete
3. Unifo

1.⇒ It m mus Exi an log Ha log

2.⇒ The Th m

3.⇒ I

⇒ 1⇒

2⇒

Date:
MON TUE WED THU FRI SAT

1. Easy to Compute
2. Determinism
3. Uniformity

1.) ⇒ It means the cost of computing a Hash fun^n must be small.
Ex: If binary search algorithm can search an element from a sorted table of n items log(n) key comparision then the cost of Hash fun^n must be less then performing a log(n) key comparision.

2.) ⇒ The Hash procedure must be deterministic. This means where the same Hash value must be generated for given input value.

3.) ⇒ It should provide a uniform distribution across the Hash table. A good Hash fun^n must map the keys as evenly as possible.

⇒ Hashing are implimented by two steps.
1) An element is converted into integer by using a Hash fun^n. This integer is used as index or location to store the original element.
2) Then the element is stored into the Hash Table which can be quickly retrived using index.

There are different Hash function

1. Division Method
2. Multiplication Method
3. Midsquare method
4. Folding method

1.> In division method hash fun^n

$h(z) = z \bmod m$

↑ key ↑ size of Hash Table

if m & z is even then h(z) is even
if m is even & z is odd then h(z) is odd

if even keys are more than odd keys then the division method will not spread the Hash value uniformly. So it is best to choose m to be a prime number. because making m a prime no. increases the likely hood that the keys are mapped with a uniformity in the output range of values.

Calculate the Hash Values of Key 1, 2, 3, 4, 5, 462 & m = 97.

h(1234) = 97 | 1234 (12

97

264

194

70

70 is the index

2.> 1.
2
3
4

Ex:

Date:
MON TUE WED THU FRI SAT

$h(5462) = 97 \overline{)5462}$ 56

485

612

582

30

30 is the index

2.) 1. Choose a Constant A such that $0 < A < 1$
2. Multiply the key K with A.
3. Extract the Fractional Part of KA.
4. Multiply the result of Step-3 by m and take the floor.

Hash fun^n given by $h(x) = \lfloor m(KA \bmod 1) \rfloor$

$(KA \bmod 1)$ gives the fractional part of KA

m = total number of indices in the hash table

A = 0.618033987

Ex: Given a Hash table of Size 1000 map the key 12345 to a appropriate location in a hash table.

m = 1000

key = 12345

$h(12345) = \lfloor m(KA \bmod 1) \rfloor$

$= \lfloor (1000)((12345)(0.618033987)) \bmod 1) \rfloor$

$= \lfloor (1000)(7629.629591 \bmod 1) \rfloor$

$= \lfloor (1000)(0.629591) \rfloor$

$= \lfloor 629.591 \rfloor$

$= 629$

3.) 1. Square the value of key i.e k^2
2. Extract the middle r bits of the result obtain in step-1.
3. In the mid square method same bit must be choosen from all the key so the hash fun^n is
$h(K) = S \Rightarrow S$ is obtained by selecting r bits from k^2

If we most significant digit then most of the time MSD are same for most of the no. So if we select mid bits then chances of collision is less. Calculate th

Calculate the Hash value of key 1234, 5642 using the midsquare middle method. The hash table has 100 memory location.
0-99
$r = 2$
$h(1234) = k^2 = 1522756$ $h(K) = 27$
$h(5642) = k^2 = 31832164$ $h(K) = 21$

4.) 1. Divide the key value into the no. of Parts. i.e divide K into $K_1, K_2, \ldots K_n$ where each part has the same no. of digits except the last part which may have lesser digit then the

Ex: G

other parts.

2. Add the individual part that is obtain the sum of $K_1, K_2, \ldots K_n$. The hash value is produce by ignoring the last carry if any.

3. If the Hash Table has size of 1000 then to address this 1000 location we need atleast 3 digits. therefore each part of the key must have 3 digit except last part which may have lesser digit

Ex: Given a Hash table of 1000 location calculate the hash value using folding method for keys 5678, 321 & 34567

key	5678	321	34567
parts	56, 78	32, 1	34, 56, 7
sum	(1)34	33	97
Hash value	34	33	97

ignore the last carry

Collision

Collision Resolution techniques

1) open addressing
   - → Linear probing
   - → quadratic probing
   - → Double Hashing
   - → Rehashing
2) chaining
3) closed hashing

Once a collision takes place open addressing or closed hashing computes new position using a probe sequence and next record is stored in that position. Hash table contains two types of value.

1) Data value & 2) Sentinel values

Sentinel value indicate that the location contain no values.

The process of examing memory location in the hash table is called as probing.

L.P→ If a value is already stored at a location generated by h(K) then the foll. hash fun^n is used to resolve the collision

Linear Probing

$h(K,i) = [h'(K) + i] \bmod m$

- $h'(K)$ → $K \bmod m$
- i → it is the probe no. varies from 0 to m-1
- m → size of hash table

Example: Consider a Hash Table of size 10 using linear probing insert the key 72, 27, 36, 24, 63, 81, 92 and 101

(1) $i = 0$

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

(2) Insert 72

$h(K,i) = [h'(K) + i] \bmod m$

$= [K \bmod m + i] \bmod m$

Date:
MON TUE WED THU FRI SAT

$= [72 \bmod 10 + 0] \bmod 10$

$= [2 + 0] \bmod 10$

$= [2] \bmod 10$

$= 2$

0	1	2	3	4	5	6	7	8	9
-1	-1	2	-1	-1	-1	-1	-1	-1	-1

③ Insert 27

$h(K, i) = [h'(K) + i] \bmod m$

$= [K \bmod m + i] \bmod m$

$= [27 \bmod 10 + 0] \bmod 10$

$= [7 + 0] \bmod 10$

$= [7] \bmod 10$

$= 7$

④ Insert 36

$h(K, i) = [h'(K) + i] \bmod m$

$= [K \bmod m + i] \bmod m$

$= [36 \bmod 10 + 0] \bmod 10$

$= [6] \bmod 10$

$= 6$

⑤ Insert 24

$h(K, i) = [h'(K) + i] \bmod m$

$= [K \bmod m + i] \bmod m$

$= [24 \bmod 10 + 0] \bmod 10$

$= [4] \bmod 10$

$= 4$

⑥ Insert 63

$h(K, i) = [h'(K) + i] \bmod m$

$= [K \bmod m + i] \bmod m$

$= [63 \bmod 10 + 0] \bmod m$

= [3] mod 10
= 3

⑦ 81 Insert 81
h(K, i) = [h'(K) + i] mod m
= [K mod m + i] mod m
= [81 mod 10 + 0] mod 10
= [81] mod 10
= 81

⑧ Insert 92
h(K, i) = [h'(K) + i] mod m
= [K mod m + i] mod m
= [92 mod 10 + 0] mod 10
= [2] mod 10
= 2
i++
h(K, i) = [h'(K) + i] mod m
= [K mod m + i] mod m
= [92 mod 10 + 1] mod 10
= [2 + 1] mod 10
= 3
i++
h(K, i) = [92 mod 10 + 2] mod 10
= [2 + 2] mod 10
= [4] mod 10
= 4 i++
h(K, i) = [92 mod 10 + 3] mod 10
= [2 + 3] mod 10
= [5] mod 10
= 5

⑨ Inser
h(K,
h
h(K
h(
h
r

Date:
MON TUE WED THU FRI SAT

⑨ Insert 101

$h(K,i) = [h'(K)+i] \bmod m$

$= [K \bmod m + i] \bmod m$

$= [101 \bmod 10 + 0] \bmod 10$

$= [1] \bmod 10$

$= 1$

i++

$h(K,i) = [101 \bmod 10 + 1] \bmod 10$

$= [1+1] \bmod 10$

$= [2] \bmod 10$

$= 2$ i++

$h(K,i) = [101 \bmod 10 + 2] \bmod 10$

$= [1+2] \bmod 10$

$= 3$

$h(K,i) = [101 \bmod 10 + 3] \bmod 10$

$= [1+3] \bmod 10$

$= 4$

$h(K,i) = [101 \bmod 10 + 4] \bmod 10$

$= [1+4] \bmod 10$

$= 5$

$h(K,i) = [101 \bmod 10 + 5] \bmod 10$

$= [1+5] \bmod 10$

$= 6$

$h(K,i) = [101 \bmod 10 + 6] \bmod 10$

$= [1+6] \bmod 10$

$= 7$

$h(K,i) = [101 \bmod 10 + 7] \bmod 10$

$= [1+7] \bmod 10$

$= 8$

Problem of linear Probing

Insert the key (7, 77, 177, 187) &
m = 10

1. Insert 7
$h(K,i) = [h'(K)+i] \bmod m$
$= [K \bmod m + i] \bmod m$
$[7 \bmod 10 + 0] \bmod m$
$= [7] \bmod 10$
$= 7$

2. Insert 77
$h(K,i) = [h'(K)+i] \bmod m$
$= [K \bmod m + i] \bmod m$
$= [77 \bmod 10 + 0] \bmod 10$
$= [7] \bmod 10$
$= 7$
i++
$h(K,i) = [77 \bmod 10 + 1] \bmod 10$
$= [7+1] \bmod 10$
$= 8$

3. Insert 177
$h(K,i) = [h'(K)+i] \bmod m$
$= [177 \bmod 10 + 0] \bmod 10$
$= [7+0] \bmod 10$
$= 7$
i++
$h(K,i) = [177 \bmod 10 + 1] \bmod 10$
$= [7+1] \bmod 10$

= 8

i++

$h(K,i) = [187 \bmod 10 + 2] \bmod 10$

$= [7 + 2] \bmod 10$

$= 9$

4. Insert 187

$h(K,i) = [h'(K) + i] \bmod m$

$= [K \bmod m + i] \bmod m$

$= [187 \overset{\text{mod } 10}{\wedge} + 0] \bmod 10$

$= [7] \bmod 10$

$= 7$

i++

$h(K,i) = [187 \bmod 10 + 1] \bmod 10$

$= [7 + 1] \bmod 10$

$= 8$

i++

$h(K,i) = [187 \bmod 10 + 2] \bmod 10$

$= [7 + 2] \bmod 10$

$= 9$

i++

$h(K,i) = [187 \bmod 10 + 3] \bmod 10$

$= [7 + 3] \bmod 10$

$= [10] \bmod 10$

$= 0$

Here one of the portion of hash table is filled with keys where other portion remain unused so this is known as primary cluster problem To avoid primary clustering other techniques such as quadretic probing & double hashing are used

Searching a value using linear probing

searching in a linear probing is same as storing. while searching for a value in a Hash table the array index is recomputed & the key of the element stored at that location is compared with the value that has to be searched. If the match is found the search operation is successive so the time complexity is O(1). If the key doesn't match then the search fun begins a sequencial search until

i) the value is found
ii) the search fun encounter a vacant location in the table indicating the value is not present.
iii) Search fun terminates because it reaches to the end of the table & the value is not present.

In a worst case the search operation has a(n-1) comparition so the run time of comparition algo is O(n).

Date:
MON TUE WED THU FRI SAT

Worst case encounters when after scanning all the n-1 elements the value is either present at the last location or not present in the table.
Here with the increase in the no. of collision the distance between the array index computed by the Hash fun^n & the actual location of the element increase wher there by increase in the search time.

Quadretic Probing

$h(K,i) = [h'(K) + c_1 i + c_2 i^2] \bmod m$
$= [K \bmod m + c_1 i + c_2 i^2] \bmod m$

c_1, c_2 = It is a constant such that
$c_1, c_2 \neq 0$.

In quadratic probing eliminates the primary clustering problem of linear probing because it does quadr. search. Using quadratic probing insert the key

Using quadretic probing insert the keys 72, 27, 36, 24, 63, 81, 101, 92
$c_1 = 1$ & $c_2 = 3$ & $m = 10$.

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Date:
MON TUE WED THU FR SAT

Insert 72

$h(K,i) = [h'(K) + C_1 i + C_2 i^2] \bmod m$

$= [K \bmod m + C_1 i + C_2 i^2] \bmod m$

$= [72 \bmod 10 + 0 + 0] \bmod 10$

$= [2] \bmod 10$

$= 2$

Insert 27

$h(K,i) = [h'(K) + C_1 i + C_2 i^2] \bmod m$

$= [K \bmod m + C_1 i + C_2 i^2] \bmod m$

$= [27 \bmod 10 + 0 + 0] \bmod 10$

$= [7] \bmod 10$

$= 7$

Insert 36

$h(K,i) = [h'(K) + C_1 i + C_2 i^2] \bmod m$

$= [36 \bmod 10 + 0 + 0] \bmod 10$

$= [6] \bmod 10$

$= 6$

Insert 24

$h(K,i) = [h'(K) + C_1 i + C_2 i^2] \bmod m$

$= [24 \bmod 10 + 0 + 0] \bmod 10$

$= [4] \bmod 10$

$= 4$

Insert 63

$= [63 \bmod 10 + 0 + 0] \bmod 10$

$= [3] \bmod 10$

$= 3$

Date:
MON TUE WED THU FRI SAT

Insert 81
= [81 mod 10 + 0 + 0] mod 10
= [1] mod 10
= 1

Insert 101
= [101 mod 10 + 0 + 0] mod 10
= [1] mod 10
= 1
i++
= [101 mod 10 + 1 + 3] mod 10
= [5] mod 10
= 5

Insert 92
i=0 = [92 mod 10 + 0 + 0] mod 10
= [2] mod 10
= 2
i++
i=1 = [92 mod 10 + 1 + 3] mod 10
= [6] mod 10
= 6
i++
i=2 = [92 mod 10 + 2 + 12] mod 10
= [16 mod 10] mod 10
= [6] mod 10
= 6

Even though hash table is empty we can't store some of the value in Hash table.

Secondary Clustering

All though quadretic probing is free from primary clustering still encounter a secondary clustering it means there is a collision bet two keys the same prob sequence will follow for both. If the quadretic probing the problem of multiple collision increases as the table becomes full.

Double Hashing

Double Hashing uses one hash value then repeatedly step forward at interval until an empty location reach.

The interval is decided using a second independent hash fun^n hence the name Double Hashing

$h(K,i) = (h_1(K) + i h_2(K)) \bmod m$

m = Size of the hash table.

$h_1(K)$ & $h_2(K)$ = two hash fun^n

$h_1(K) = K \bmod m$

$h_2(K) = K \bmod m'$

i = Prob sequence varies from 0 to $m-1$

m' = choosen to be less than m

$m' = m-1$ or $m-2$.

Date.:
MON TUE WED THU FRI SAT

Advantages of Double Hashing

- Double Hashing minimize repeated collision & the effect of clustering i.e double hashing is free from Problem associated with primary clustering as well as secondary clustering.

Ex: Consider a Hash Table of size 10 using Double Hashing. Insert the keys 72, 27, 36, 24, 81, 92 & 101 into the hash table. Take H. $H_1 = (K) \bmod 10$
$h_2 = (K) \bmod 8$

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

i) $h(72,0) = [K \bmod m + i * K \bmod m'] \bmod m$

$= [72 \bmod 10 + (0) * (72) \bmod 8] \bmod 10$

$= [2] \bmod 10$

$= 2$

ii) Insert 27

$h(27,0)$

$= [27 \bmod 10 + (0) * (27) \bmod 8] \bmod 10$

$= [7] \bmod 10$

$= 7$

(iii) h(36, 0)
= [36 mod 10 + (0) * (36 mod 8)] mod 10
= [6] mod 10
= 6

(iv) h(24, 0)
= [24 mod 10 + (0) * 24 mod 8] mod 10
= [4] mod 10
= 4

(v) h(81, 0)
= [81 mod 10 + (0) * 81 mod 8] mod 10
= [1] mod 10
= 1

(vi) h(92, 0)
= [92 mod 10 + (0) * 92 mod 8] mod 10
= [2] mod 10
= 2 i++

(vii) h(63, 0)
= [63 mod 10 + (0) * 63 mod 8] mod 10
= [3] mod 10
= 3

(viii) h(101, 0)
=

Date:
MON TUE WED THU FRI SAT

h(92, 1)
= [92 mod 10 + 1 * 92 mod 8] mod 10
= [2 + 4] mod 10
= [6] mod 10
= 6 i++

h(92, 2)
= [92 mod 10 + 2 * 92 mod 8] mod 10
= [2 + 8] mod 10
= [10] mod 10
= 0

(viii) h(101, 0)
= [101 mod 10 + 0 * 101 mod 8] mod 10
= [1] mod 10
= 1
i++

h(101, 1)
= [101 mod 10 + 0 * 101 mod 8] mod 10
= [1 + 5] mod 10
= [6] mod 10
= 6 i++

h(101, 2)
= [101 mod 10 + 2 * 101 mod 8] mod 10
= [1 + 10] mod 10
= [11] mod 10
= 1
i++

h(101, 3)
= [101 mod 10 + 3 * 101 mod 8] mod 10
= [1 + 15] mod 10
= [16] mod 10
= 6

0	1	2	3	4	5	6	7	8	9
92	81	72	63	24	-1	36	27	-1	-1

Although Double hashing is a very efficient algorithm it always requires m to be a prime number.
Now take m = 11 & m' = 7

(i)
h(72, 0)
= [72 mod 11 + (0) * 72 mod 7] mod 10
= [6] mod 11
= 6

(ii)
h(27, 0)
= [27 mod 11 + (0) * 27 mod 7] mod 11
= [5] mod 11
= 5

(iii) h(36, 0)
= [36 mod 11 + (0) * 36 mod 7] mod 11
= [3] mod 11
= 3

(iv) h(24
= [24
= [2]
= 2

(v) h(6
= [63
= [8
= 8

(vi) h(
= [
= [
=

(vii)
=
=
=
=

(iv) $h(24, 0)$
$= [24 \bmod 11 + (0) * 24 \bmod 7] \bmod 11$
$= [2] \bmod 11$
$= 2$

(v) $h(63, 0)$
$= [63 \bmod 11 + (0) * 63 \bmod 7] \bmod 11$
$= [8] \bmod 11$
$= 8$

(vi) $h(81, 0)$
$= [81 \bmod 11 + (0) * 81 \bmod 7] \bmod 11$
$= [4] \bmod 11$
$= 4$

(vii) $h(92, 0)$
$= [92 \bmod 11 + (0) * 92 \bmod 7] \bmod 11$
$= [4] \bmod 11$
$= 4$

i++

$= [92 \bmod 11 + (1) * 92 \bmod 7] \bmod 11$
$= [4 + 1] \bmod 11$
$= [5] \bmod 11$
$= 5$

i++

$= [92 \bmod 11 + (2) * 92 \bmod 7] \bmod 11$
$= [4 + 2] \bmod 11$
$= [6] \bmod 11$
$= 6$

$= [92 \bmod 11 + 3 * 92 \bmod 7] \bmod 11$

$= [4 + 37] \bmod 11$

$= [7] \bmod 11$

$= 7$

(viii) $h(101, 0)$

$= [101 \bmod 11 + 0 * 101 \bmod 7] \bmod 11$

$= [2] \bmod 11$

$= 2$

i++

$= [101 \bmod 11 + 1 * 101 \bmod 7] \bmod 11$

$= [2 + 3] \bmod 11$

$= [5] \bmod 11$

$= 5$

i++

$h(101, 2)$

$= [101 \bmod 11 + 2 * 101 \bmod 7] \bmod 11$

$= [2 + 6] \bmod 11$

$= [8] \bmod 11$

$= 8$

i++

$(h(101, 3)$

$= [2 + 9] \bmod 11 + 3 * 101 \bmod 7] \bmod 11$

$= [11] \bmod 11$

$= 0$

0	1	2	3	4	5	6	7	8	9
101	-1	24	36	81	27	72	92	63	-1

Date:
MON TUE WED THU FRI SAT

Rehashing

When the Hash table becomes nearly full the no. of collision increases where by degrading the performance of insertion & search operation. In such cases a better option is to create a new hash table with size double of the original hash table.
All the entries in the original hash table will then have to be move to the new hash table. This is done by taking each entry & computing its new hash value and then inserting into the new hash table. Consider the k

Consider the hash table of size 5,

$h(x) = x \% 5$

0	1	2	3	4
	26	31	43	17

Size - 10

$h(26) = 26 \% 10$

$= 26 \bmod 10$

$= 6$

0	1	2	3	4	5	6	7	8	9
						26			

$h(31) = 31 \% 10$

$= 31 \bmod 10$

$= 1$

0	1	2	3	4	5	6	7	8	9
	31					26			

$h(43) = 43 \% 10$

$= 43 \bmod 10$

$= 3$

0	1	2	3	4	5	6	7	8	9
	31		43			26			

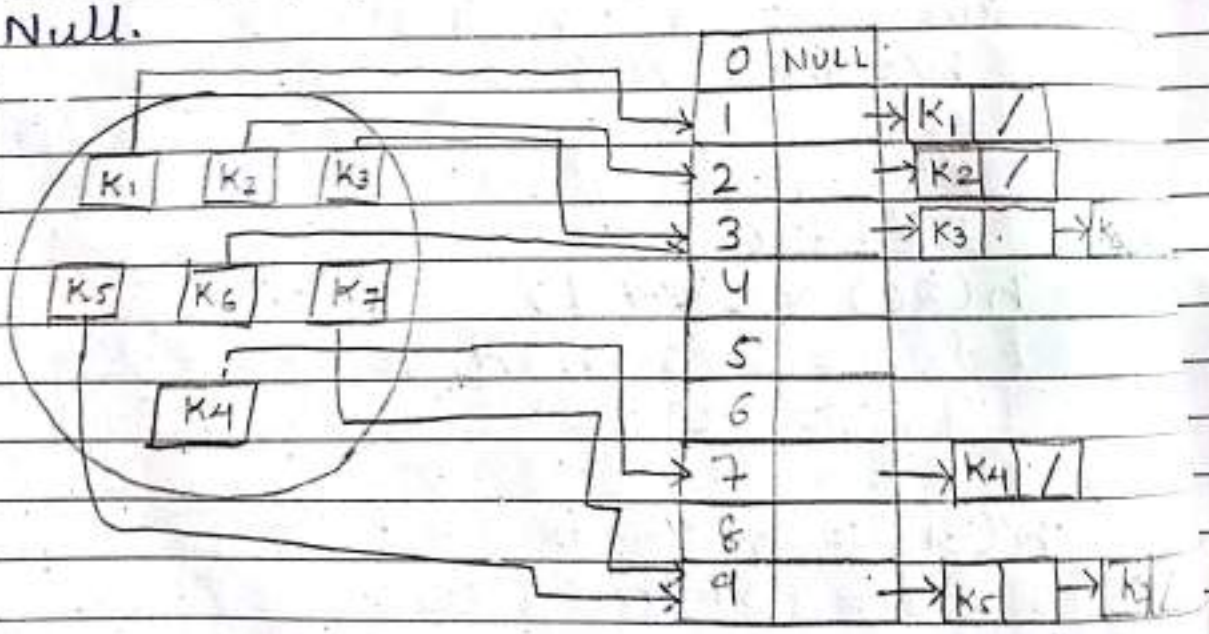
$h(17) = 17 \bmod 10$

$= 7$

0	1	2	3	4	5	6	7	8	9
	31		43			26	17		

Collision Resolution By chaining

In chaining each location in the hash table stores a pointer in the link list. that contain all the key values that are hash to that location. i.e (location 1 in the hash table points to the head of the link list of all the key values that has to 1). If no key values has to 1 then location in the hash table contains Null.



Date:
MON TUE WED THU FRI SAT

Operation on a Chain Hash Table

Searching for a value in a chain hash table is as simple as scanning a link list for an entry in a given tree.

Insertion operation append a key and/or front of the link list pointed by the hash location

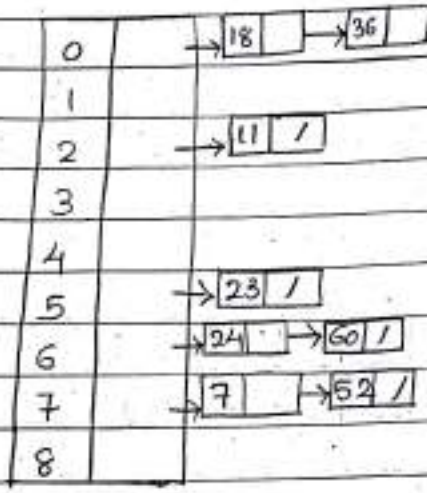
Deletina a key require searching a list & removing the element

Chain hash table with ~~list~~ link list are widely used. The code for this algorithm is exactly the same as that for inserting, deleting & searching a value in a singly link list. The cost of insertion a key in hash table is of O(1). & cost of deletion is O(m)

m = no. of element in the list of that location. In a worst case searching complexity is O(n).

n = no. of keys stored in the chain hash table. This case arise when all the key values are inserted into the link list of the same location.

Insert a key 7, 24, 18, 52, 36, 54, 11, 23, 60.
Chain Hash table of 9 memory locations use $h(k) = k \bmod m$



$h(7) = 7 \bmod 9$
$= 7$

$h(24) = 24 \bmod 9$
$= 6$

$h(18) = 18 \bmod 9$
$= 0$

$h(52) = 52 \bmod 9$
$= 7$

$h(36) = 36 \bmod 9$
$= 0$

$h(54) = 54 \bmod 9$
$= 0$

$h(11) = 11 \bmod 9$
$= 2$

$h(23) = 23 \bmod 9$
$= 5$

$h(60) = 60 \bmod 9$
$= 6$

Date:
MON TUE WED THU FRI SAT

Advantages An

1. It remains effective when the no. of keys to be stored is much higher than the no. of location in the hash table. However the no. of keys increases the performance of chain hash table decrease linearly.
2. For ex: A chain hash table with 1000 memory location & 10000 stored key will give 5-10 time less performance. as compared to the performance of chain hash table. But chain hash table is still 1000 time faster than simple hash table.
3. The other advantages of chaining of collision resolution is that it's performance unlike quadretic probing doesn't degrade when the table is more than half full.
4. This technique is absolute free from clustering.

Disadvantages

1. It inherits the disadvantages of link list. To store even a key value the space overhead of the next pointer in each entry can be significant.
2. Traversing a link list has poor cache performance making the processor catche ineffective.

Date:
MON TUE WED THU FRI SAT

Advantages & Disadvantages of Hashing

1. The main advantages of hash table over other data structure is speed. It is more advantages when the no. of keys is large.
2. Being able to look up & insert the data in order of O(1). which is independent of the size of the data structure.

1. Hash table can be difficult to implement.
2. Choosing a effective hash fun^n for specific application is more an art than a science.
3. Hashing is not effective when the no. of entries is very small.
4. It becomes ineffective when there are many collision.

Application of Hashing.

1. Hash table are widely used in situation where large amount of data have to be access to quickly search & retrive information.
2. Hashing is used as a Symbol table.
3. Hash table can be use to store massive amount of information. For ex: To store Drivers license record. given the Driver license no. hash table helps to quickly

Date:
MON TUE WED THU FRI SAT

get the information about the doivers.

4. Hashing is used in telephone database to quickly find the person telephone no.

5. Hashing is used in Database Indexing
- Some database stores a saparate file known as indexes when data has to be retrive from a file. The key information is first foriind in the appropriate index file which reference the exact memory location of data in database file. This key infor. in the index file is often stored as a hash value.

Symbole table

Symbole table is an important data structure created & maintained by Compiler in order to store the inform. about the occurance of various entities such as variable name, obj., classes and interface, etc.

A Symbole table may surve the foll. purpose depending upon the language

1) To store the name or symbols of all entites in structured form.
2) To verify if variable is declared or not.
3) To implment type checking by verifying assignments & expres. in the source code are symentically

correct or not.

4) To determine the scope of name. ex:
A Symbole table is simply a hash tab... which can be either linear or hash table.
It maintains the entry for each ... in the foll. format.

<Symbol name, type, attribute>

↓

Ex: Static int Interest

<interest, int, static>

Implimentation

If Compiler is to handle a small amount of data then symbole table can be implimented as an unorder list whi... is easy to code but it is only suitable ... small tables only.

A Symbole table can be implement... in one of the foll. ways:

1) Linear list (sorted or unsorted)
2) BST
3) Hash table

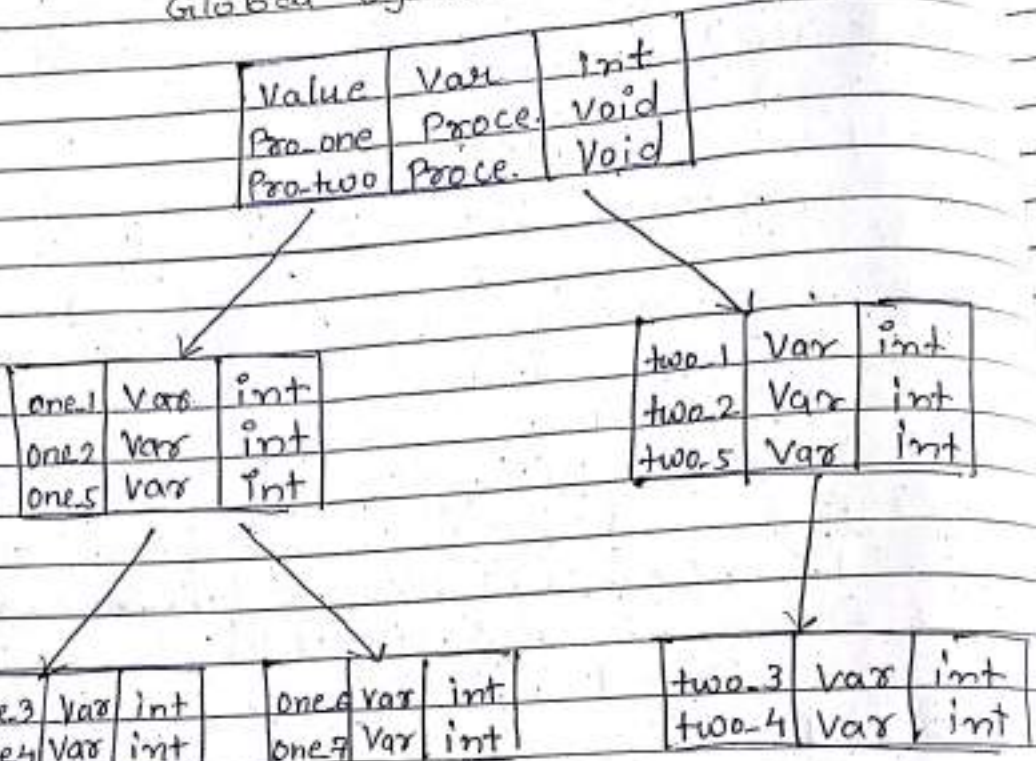
Symbole table are mostly implime... nted as hash table, where the source code symbole itself is treated as a key for the hash function and return the value is the information about the symbole.

Date:
MON TUE WED THU FRI SAT

ex:

```
int value=10;
Void proo_one()
 {
    int one_1;
    int one_2;
     {
        int one_3; } inner-scope 1
        int one_4; }
     }
    int one_5;
     {
        int one_6; } inner-scope 2
        int one_7; }
     }
 }
Void proo_two()
 {
    int two_1;
    int two_2;
     {
        int two_3; } inner-scope 3
        int two_4; }
     }
     int two_5;
 }
```

Global Symbol Table



Runtime of Linear Search & Binary Search.

It's complexity is order O(log n). For binary search algorithm it is mandatory for the target array to be sorted.

mid = low + (high - low)/2

= 0 + (9 - 0)/2

= 4

0	1	2	3	4	5	6	7	8	9
10	14	19	26	27	31	33	35	42	44

5	6	7	8	9
31	33	35	42	44

= 5 + (9 - 5)/2

= 5 + 2 = 7

Date:
MON TUE WED THU FRI SAT

5	6
31	33

$= 5 + (6-5)/2$

$= 5 + 1/2$

$= 5$

→ Suppose we start an array of length 8 then Incorrect guesses reduce the size of the portion to 4, 2, 1

→ Once, the reasonable portion contain just one element then no further guess occurs & the last one extra guess for the one element portion is either correct or inco.

→ with an array of length 8, Binary search needs at most 4 comparison.

→ n+1 guesses is needed for any.

→ To represent this thing one mathem. fun^n is there which is log n.

→ ∴ The complexity is binary search is O(log n)