S.Y. B.Sc. (IT) : Sem. III

Data Structures

Time : 2½ Hrs.] Prelim Question Paper Solution **[Marks : 75**

Q.1Attempt the following (any THREE) **[15]**

Q.1(a) What is an algorithm? Write down characteristics of algorithm. **[5]**

Ans.: An algorithm can be defined as finite collection of well-defined steps designed to solve a particular problem. An algorithm must have the following characteristics:

Input: An algorithm must take some inputs hat are required for the solution of a problem.

Process: An algorithm must perform certain operations on the input data which are necessary for the solution of a problem.

Output : An algorithm should produce certain output after processing the inputs.

Finiteness: An algorithm must terminate after executing certain finite number of steps.

Effectiveness: Every step of an algorithm should play a role in the solution to the problem. Each step must be unambiguous, feasible and definite. /

Q.1(b) What are the advantages and limitations of an array? **[5]**

Ans.: **Advantages of Array**

- Array is the simple kind of data structure which is very easy to implement.
- Address of any element of the array can be calculated easily as elements are stored in contiguous memory locations.
- Array can be used to implement other data structures such as stack, queue, trees and graph.
- If elements of an array are stored in some logical order then binary search can be applied to search an element in the array efficiently.

Limitations of An Array

- Array is the static kind of data structure. Memory used by the array cannot be increased or decreased whether it is allocated at runtime or compile time.
- Only homogenous elements can be stored in the array. Therefore in case we want to store the data of mixed type the array cannot be used.
- Insertion and deletion of elements are very time consuming in array. When new element is to be inserted in the middle of the array then elements are required to move to create the space for new element. If an element is to be deleted from the array, all its preceding elements need to be moved to fill the vacated position of the array.

Q.1(c) What is data structure? Explain different categories/classification of data structure. **[5]**

Ans.: A data structure is a way of storing the data in computer's memory so that it can be used efficiently. A data structure is a logical/ mathematical model of organization of data. The choice of data structure begins with the choice of an Abstract Data Type (ADT) such as array. The different categories of data structure are

Linear Data Structure: The elements in a linear data structure form a linear sequence. Example of linear data structures are: Array, Linked list, Queue, Stack etc.

Non-Linear Data Structure: The elements in a non-linear data structure do not form any linear sequence. Example Three and Graph.

Static Data Structure: Static data structures are those whose memory occupation is fixed. The memory taken by these data structures cannot be increased or decreased at run time. Example of the static data structure is an array. The size of an array is declared at the compile time and this size cannot be changes during the run time.

Dynamic Data Structure: Dynamic data structure are those whose memory occupation is not fixed. The memory taken by these data structures can be increased or decreased at run time. Example of the dynamic data structure is linked List. The size of the linked list can be changed during the run time.

Other data structures like stack, queue, tree, and graph can be static or dynamic depending on, whether these are implemented using an array or a linked list.

Homogeneous Data Structure: Homogenous data structures are those in which data of same type can be stored. Example is array.

Non-Homogeneous Data Structure: Non-Homogenous data structures are those in which data of different type can be stored. Example is linked list.

Q.1(d) Consider a two dimensional array D [5:7, –3:6]. If the base address of D is 1536 and each element takes 2 memory cells then find the address of D6, 0 element assuming that **[5]**

(i) Array D is stored in column major order

(ii) Array D is stored in row major order

Ans.: Lower index of 1st dimension (ℓ_{br}) = 3

Upper index of 1st dimension (u_{br}) = 7

Lower index of 2nd dimension (ℓ_{bc}) = –2

Upper index of 2nd dimension (u_{bc}) = 6

Length of the 1st dimension (No. of Rows)

$r = u_{br} - \ell_{br} + 1 \quad = 7 - 3 + 1 \quad = 5$

Length of the 2nd dimension (No. of columns)

$c = u_{bc} - \ell_{bc} + 1 \quad = 6 - (-2) + 1 \quad = 9$

Row Major Order :

Adjusted D[4] [0] $= \text{Base (D)} + W[c(1 - \ell_{br}] + [j - \ell_{bc}]$

$= 5639 + 2 * [9(4 - 3) + (0 - (2))]$

$= 5639 + 2 * [9(1) + (2)]$

$= 5639 + 2 * [11] \quad = 5661$

Column Major order :

Address of d[4] [0] $= \text{Base (D)} + W[(i(1 - \ell_{br}) + r(j - \ell_{bc})]$

$= 5639 + 2 * [(4 - 3) + 5 * (0 - (-2))]$

$= 5639 + 2 * [1 + 10]$

$= 5639 + 22 \quad = 5661$

Q.1(e) Differentiate between linear search and binary search. **[5]**

Ans.:

	Linear Search	**Binary Search**
1.	Elements of an array need not be in an sorted order	Elements of an array must be in any sorted order.
2.	Complexity of linear search is 0(n)	Complexity of binary search is 0($\log_2 n$)
3.	Linear search is accomplished by comparing desired element with each element of the array starting from 1st element of the array	In Binary search the desired element is compared with the middle element and selecting the half portion of the list in which element may be present. This procedure of having the list is repeated till the element is found or we conclude that element is not present.
4.	Can be applied an linear as well as non–linear data	Can be applied only on linear data

Q.1(f) Sort the following given elements using Bubble sort. 30, 20, 50, 40, 10, 70, 60, 80 **[5]**

Ans.:

0	1	2	3	4	5	6	7
30	20	50	40	10	70	60	80

Number of passes = n – 1
= 8 – 1
= 7

Bubble Sort

Pass 1 : Number of steps = n – 1 = 8 – 1 = 7

	0	1	2	3	4	5	6	7
S[0] > S[1] Yes	20	30	50	40	10	70	60	80

S[1] > S[2] No

	0	1	2	3	4	5	6	7
S[2] > S[3] Yes	20	30	40	50	10	70	60	80

	0	1	2	3	4	5	6	7
S[3] > S[4] Yes	20	30	40	10	50	70	60	80

S[4] > S[5] No

	0	1	2	3	4	5	6	7
S[5] > S[6] Yes	20	30	40	10	50	60	70	80

S[6] > S[7] No

Largest element 80 is at S[7]

Pass 2 : Number of step S = n – 2 = 8 – 2 = 6

S[0] > S[1] No

S[1] > S[2] No

	0	1	2	3	4	5	6	7
S[2] > S[3] Yes	20	30	10	40	50	60	70	80

S[3] > S[4] No

S[4] > S[5] No

S[5] > S[6] No

2nd largest element 70 is at S[6]

Pass 3 : Number of steps = n – 3 = 8 – 3 = 5

S[0] > S[1] No

	0	1	2	3	4	5	6	7
S[1] > S[2] Yes	20	10	30	40	50	60	70	80

S[2] > S[3] No

S[3] > S[4] No

S[4] > S[5] No

3rd largest element 60 is at S[5]

Pass 4 : Number steps = n – 4 = 8 – 4 = 4

	0	1	2	3	4	5	6	7
S[0] > S[1] Yes	10	20	30	40	50	60	70	80

S[1] > S[2] No

S[2] > S[3] No

S[3] > S[4] No

Pass 5 : Number of steps = n – 5 = 8 – 5 = 3
S[0] > S[1] No
S[1] > S[2] No
S[2] > S[3] No

Pass 6: No. of steps = n – 6 = 8 - 6 = 2
S[0] > S[1] No
S[1] > S[2] No

Pass 7 : Number of steps = n – 7 = 8 – 7 = 1
S[0] > S[1] No

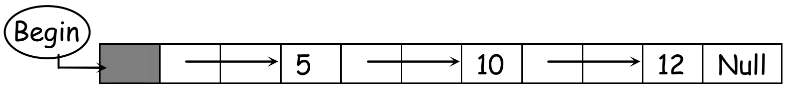
	0	1	2	3	4	5	6	7
S[1] > S[2] Yes	10	20	30	40	50	60	70	80

Q.2Attempt the following (any THREE) **[15]**

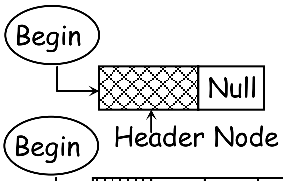
Q.2(a) What is header linked list? Explain different categories of header linked list. **[5]**

Ans.: A header linked list is a special kind of linked list which contains a special node at the beginning of the list. This special node is known as head node.

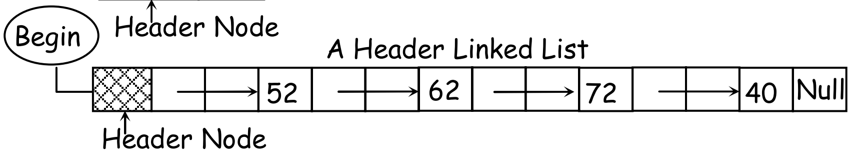
- The head node contains information regarding the linked list.
- This information may be total number of nodes in the linked list, some description for the user like creation date, modification date, whether the data in the list is sorted or unsorted.



- ***Grounded Header Linked List*** **:** It is list in which last mode of the list contains the Null in its Next pointer field. If grounded header linked list is empty then the Null value will be stored in the Next pointer field of the head node.
- ***Circular Header Linked List*** **:** It is a list in which the last node of the list points back to the header node that is Next Pointer field of the last bode contains the address of the header node. If circular header linked list is empty then address of the head node is stored in the Next pointer field of the head node itself.
- **Two way Header Linked List** **:** It is a list in which the Pre Part of the header node contains the address of the first node and the Next part contains the address of the last node.
- **Two way Circular Header Linked List** **:** It is the list in which the Pre part of the header node contains the address of the first node and the Pre part of the first node contains the address of the header node. The Next part of the header node contains the address of the last node and the Next part of the last node contains the address of the header node.

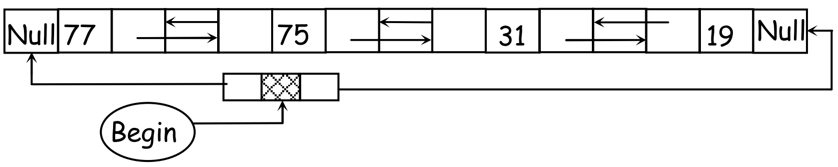


An Empty Grounded Header List Linked



A Header Linked List

Two way Header Linked List



Q.2(b) What is linked list? Write an algorithm to insert an element in the end of linked list. [5]

Ans.: Linked list is the most commonly used data structure used to store similar type of data in memory. The elements of a linked list are not stored in adjacent memory locations as in arrays.

It is a linear collection of data elements, called nodes, where the linear order is implemented by means of pointers. A linked list allocates memory for storing list elements and connects elements together using pointers.

Algorithm to insert an element in end of linked list
/* Function to insert a node at the end of a linked list */

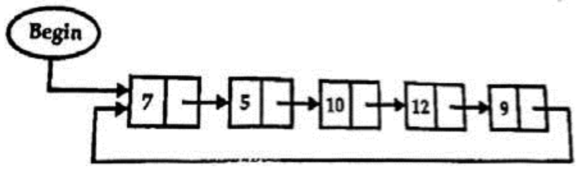
```
void creat (struct node **q. int no)
{
        struct node *temp. *r :

        if(*q == NULL)/* If the list is empty. create first node */
        {
            temp = malloc(sizeof(struct node)):
            temp –> data = no:
            temp –> next = NULL :
            *q = temp :
        }
        else
        {
            temp = *q :
            /* go to last node */
            while(temp –> next : = NULL)
                temp = temp –> next :

            /* Add node at the end */
            r = malloc(sizeof(struct node)) :
            r –> data = no :
            r –> next = NULL :
            temp –> next = r :
        }
}
```

Q.2(c) What is circular linked list> How to traverse circular linked list? [5]

Ans.: A circular linked list is a list in which last node points back to the first node instead of containing the **Null** pointer in the next part of the last node.



A Circular Linked List

- All the operations which can be performed on ordinary singular linked list can easily be performed on circular linked list with the following changes
- In case of 1-way singular linked list the next part of the last node will contain Null address but in case of circular linked list the next part of the last node consist of address of the first node i.e. Begin. Thus for reaching at the end of the circular linked list we will compare the address of the first node i.e. Begin with the address stored in Next part of each node. If both the addresses come out to be same then we have reached at the end of the circular list

- When a new node is to be inserted at the end of the circular linked list its Next part will contain the address of the first node instead of Null as in the case of singular linked list

Algorithm To Traverse A Circular Linked List

```
Step 1: If Begin = Null then
            Print: "Circular Linked List is empty"
            Exit
```

Q.2(d) What is the need of two way linked list? Write an algorithm to traverse a two linked from end to beginning. [5]

Ans.: In a 1-way linked list we traverse the list only in one direction i.e. from beginning to end
But in certain applications it is required to traverse the list in both directions i.e. in forward direction (from beginning to end) and in backward direction (from end to beginning). This can be accomplished with the help of two-way linked list or doubly linked list. In two-way linked list each node is divided into three parts **Pre, Info, Next**

- **Prev** contains the address of the preceding node
- **Next** contains the address of the next node

Tre	Info	Next

Structure of a Node used in a Two-way Linked List

In 2-way linked list two list pointers are used Begin and End which contains the address of the first and the last node respectively. The Pre part of the first node of a 2-way linked list will contain Null as there is no node preceding the first node and the Next part of the last node of a 2-way linked list will contain Null as there is no node following the last node.

Q.2(e) Explain Reversing a Linked List. [5]

Ans.: The reversing of the list means that last node becomes the first node and first becomes the last.

```
    void reverse (struct node **x)
    {
        struct node *q. *r. *s :

        q = *x :
        r = NULL :
        /* Traverse the entire linked list */
        while(q != NULL)
        {
            s = r :
            r = q :
            q = q –> next :
            r –> next = s :
        }
        *x = r :
}
```

The function reverse () receives the parameter struct node **x, which is the address of the pointer of the first node in the linked list. To traverse the linked list, a variable of type struct node * is required, q has been initialized the value of x, so q starts pointing to the first node.

The NULL value is stored in the link part of the first node, i.e.

```
    s = r;
    r = q;
    r → next = s;
```

r which is of type struct node * is initialized to a NULL value. Since r contains NULL, s would also contain NULL. Now r is assigned to q so that r also starts pointing to the first node. Finally, r → next is assigned to s so that r → next becomes NULL which is nothing but the next part of the first node. Before storing a NULL value in the next part of the first node, q is made to point to the second node through

q = q → next:

When there is second iteration of while loop, r points to the first node and q points to the second node. Now the next part of the second node should point to first node. q is made to point to the second node.

q = q → next:

During the second interation of while loop, r points to the first node and q points to the second node. Now the next part of the second node should point to first node which is done by :

s = r;
r = q;
r → next = s:

Since r points to the first node, s would also point to the first node. Now r is assigned the value of q so that r now points to the second node. Finally, r → next is assigned to s so that r → next starts pointing to the first node. But if we store the value of s in the second node then the address of third node would be lost. Hence, before storing the value of s in r → next, q is made to point to the third node by :

q = q → next:

While traversing the nodes through the while loop, q starts pointing to the next node in the list and r starts pointing to the previous node. By the end of while loop the first node becomes the last node and the last node becomes the first node.

When *x=r is executed, it ensures that pointer p now starts pointing to the node which is the last node in the original list.

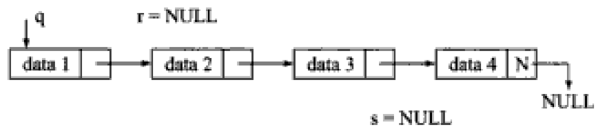


Fig.: Initial

Now, r=q, i.e. r also starts pointing to the first node (Fig.) q points to the second node as q = q → next.

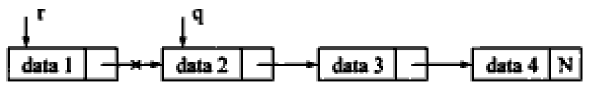


Fig.

r points to the first node and s also points to the first node as s = r (Fig.). Now, r is assigned the value of q, therefore, r now points to the second node (Fig.). From the above assignment as s was pointing to the first node so, r → next = s.

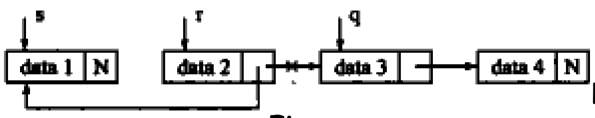
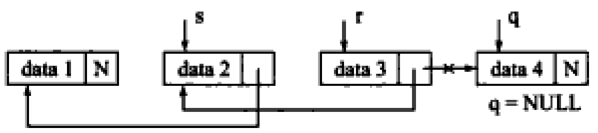
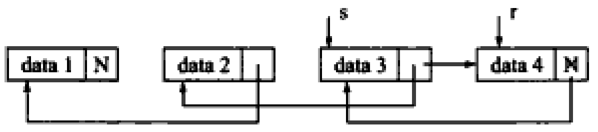


Fig.)

Fig.





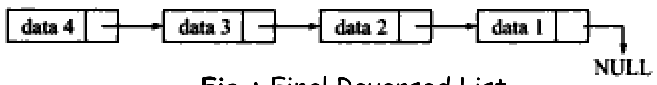
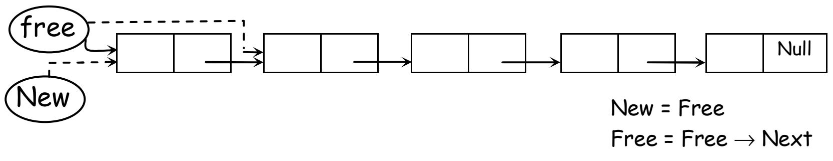


Fig.: Final Reversed List

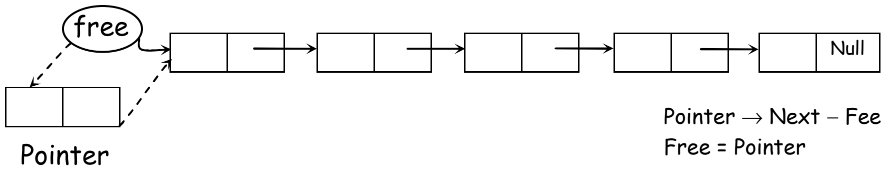
Q.2(f) Explain how memory is allocated and deallocated for linked list. [5]

Ans.: To insert an element in a linked list the first requirement is to get a free Node.

- To delete a node from a linked list it is desirable to return the memory taken by deleted Node for its reusability.
- The task of obtaining an empty Node for insertion and returning free Node after deletion is accomplished by maintaining a separate list of free Nodes that begin with pointer free which points to first available free Node.
- Whenever a new Node is to be inserted into the linked list then free memory is checked for the availability. If the Node is available then the Node is added to the linked list and pointer Free will point to next available Node.
- If there is no free node then it is indicated by Free = Null and this condition is known as overflow.
- Whenever a node is deleted from the linked list it is inserted as the first node of the free storage. The pointer Free will point to the recently added node.
- The operation of getting a free node from and returning the node to the free storage list is shown below :



Removed of a Node from the free storage List (Allocation)

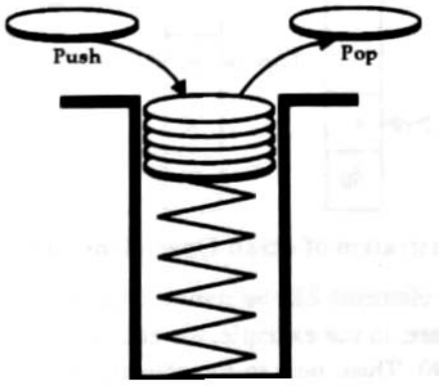


Insertion of a Node into the Free Storage list (Deallocation)

Q.3Attempt the following (any THREE) [15]

Q.3(a) Define stack. Write algorithms to perform push and pop operations on stack with array representation. [5]

Ans.: Stack is one of the most commonly used linear data structures of variable size. In the linear data structure i.e. array, the insertion and deletion of an element can take place at any position of the array but in the case of stack, the insertion and deletion of an element can occur at only one end known as Top. In stack, insertion operation is known as push and deletion operation is known as pop. Stack is also called Last In First Out (LIFO) list. It means that the last item added to the stack will be the first item to be removed from the stack. Consider an example of stack of dish plates in which clean plates are added to the top of the stack. Also, plates are removed from the top of the stack. The first plate put on the stack is the last plate to be removed from the stack, as shown in the figure below:



Following algorithms explain the push and pop operation on the stack when it is represented using an array:

Algorithm : Push Operation – Insert a new element 'Data' at the top of the stack represented by an array 'S' of size 'Max' with a stack index variable 'Top' pointing to the topmost element of the stack.

```
Step 1 :  If Top = Max Then
              Print : "Stack is already full, Overflow Condition"
              Exit
          [End If]
Step 2 :  SetTop = Top + 1
Step 3 :  Set S[Top] = Data
Step 4 :  Exit
```

Algorithm : Pop Operation – Delete an element from the stack represented by an array 'S' and returns the element 'Data' which is at the top of the stack.

```
Step 1 :  If Top = NULL Then
              Print: "Stack is already empty, Underflow Condition"
              Exit
          [End If]
Step 2 :  Set Data = S[Top]
Step 3 :  Set Top = Top – 1
Step 4 :  Exit
```

Q.3(b) Explain with example priority queue. [5]

Ans.: **Priority Queues:** A queue in which we are able to insert items or remove items from any position based on some property (such as priority of the task to be processed) is often referred to as a priority queue. Figure (a) represents a priority queue of jobs waiting to use a computer. Priorities of 1, 2 and 3 have been attached to jobs of type real-time, on-line, and batch, respectively. Therefore, if a job is initiated with priority i, it is inserted immediately at the end of the list of other jobs with priority i, for i = 1, 2 or 3. In this example, jobs are always removed from the front of the queue. (In general, this is not a necessary restriction on a priority queue.)

A priority queue can be conceptualized as a series of queue representing situations in which it is known a priori what priorities are associated with queue items. Figure (b) shows how the single-priority queue can be visualized as three separate queue, each exhibiting a strictly FIFO behavior. Elements in the second queue are removed only when the first queue is empty, and elements from the third queue are removed only when the first and second queues are empty. This separation of a single-priority queue into a series of queues also suggests an efficient storage representation of a priority queue. When elements are inserted, they are always added at the end of one of the queues as determined by the priority. Alternatively, if a single sequential storage structure is used for the priority queue, then insertion may mean that the new element must be placed in the middle of the structure.

Task identification

R_1	R_2	…	R_{i-1}	O_1	O_2	…	O_{i-1}	B_1	B_2	…	B_{k-1}	…
1	1	…	1	2	2	…	2	3	4	…	3	…

Priority ↑ R_i ↑ R_j ↑ R_k

(a) : A priority queue viewed as a single queue with insertion allowed at any position.

Priority 1

R_1	R_2	…	R_{i-1}	…	$\leftarrow R_i$

Priority 2

O_1	O_2	…	O_{i-1}	…	$\leftarrow O_j$

Priority 3

B_1	B_2	…	B_{k-1}	…	$\leftarrow B_k$

(b) : A priority queue viewed as a setoff queue.

Q.3(c) Write an algorithm to insert and delete node from a circular queue. [5]

Ans.: Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called as "Ring Buffer"

Algorithm: Insert a new element 'Data' into a queue.

Algorithm: Delete an element from the queue

```
Step 1: If Front = Null Then
            Print: "Queue is Empty, Underflow
        Condition "Exit (End if]
Step 2: Set Data = Q[Front]
Step 3: If Front = Rear Then
            Set Front = Null and
        Rear = Null Else If Front =
        n Then Set Front = 1
        Else
            Set Front =
        Front + 1 [End IF)
Step 4: Exit
```

Q.3(d) Write an algorithm to convert infix expression to postfix expression. [5]

Ans.:

```
Step 1 : Push a left parenthesis (onto the stack.
Step 2 : Append a right parenthesis) at the end of the given expression I.
Step 3 : Repeat Steps from 4 to 8 by scanning I character by character from left to right
         until the stack is empty.
Step 4 : If the current character in I is a white space, simply ignore it.
Step 5 : If the current character in I is an operand, write it as the next element of the
         postfix expression P.
Step 6 : If the current character in I is a left parenthesis (, push it onto the stack.
Step 7 : If the current character in I is an operator Then
```

(a) Pop operators (if there is any) at the top of stack while they have equal or higher precedence than the current operator and put the popped operators in the postfix expression P.
(b) Push the currently scanned operator on the stack.

Step 8 : If the current character in I is a right parenthesis Then
(a) Pop operator from the top of the stack and insert them in the postfix expression P until a left parenthesis is encountered at the top of the stack.
(b) Pop and discard left parenthesis (from the stack.
[End Loop]

Step 9 : Exit

Explanation

In the 1st step, a left brace is pushed into the stack so that stack is initially not empty and a corresponding matching right brace is placed at the end of the expression in step 2. In step 3, the loop is repeated until the end of the expression which is indicated by Null value in the stack.

Q.3(e) Convert following infix expression into prefix and postfix expressions. [5]

(i) a × b × (c − d) − (e ^ 3 × f) + g / h

(ii) (a × b x c ^ 2) + d − (c / d + e)

Ans.: (i) a × b (c − d) − (e ^ 3 × f) + 9 / h

- Prefix :
  a × b × (cd-) − (^ e3 × f) + /gh
  a × b × (cd-) − (× ^ e 3 f) + /gh
  a × × b −cd − × ^ e 3 f + / gh
  xab − cd − x ^ e 3 f + /gb
  − x × ab − cdx ^ e3f + /gh
  + x × ab − cdx ^ e3f + /gh

- Postfix :
  a × b × (cd−) − (e3 ^ x f) + gh/
  a × b × (cd−) − (e3 ^ f x) + gh/
  abx × cd − − e3 ^ fx + gh/
  ab × cd - x − e3 ^ fx + gh/
  ab × cd − x e 3 ^ f x − gh /t

(ii) (a × b × c ^ 2) + d − (c / d + e)

- Prefix :
  (a × b × ^ c2) + d − (/cd + e)
  (xab × ^ c2) + d − (+ / cde)
  (xxab ^ c2) + d − (+ / cde)
  + x x ab ^ c2d − + / cde
  −+ xx ab ^ c2d + /cde

- Postfix :
  (axbxc2^) + d −(cd/ +e)
  (abxxc2^) + d−(cd/ +e +)
  (abxc2 ^ x) + d −(cd/+e +)
  ab x c2 ^ x d + − cd /e+
  ab x c2 ^ x d + cd /e+-

Q.3(f) What is recursion? Write and explain an algorithm to find factorial of number (Use recursion) **[5]**

Ans.: **Recursion**

Recursion is very important and powerful tool for developing algorithms for various problems. Recursion is the ability of a procedure either to call itself or calling to some other procedure which may result in call to the original procedure. In computer science, solution of many problems can be best defined recursively. Two very important conditions/requirements that must be satisfied by any procedure to be defined recursively are :

- There must be a decision criterion that stops the further call to the procedure called base criteria.
- Each time a procedure calls itself either directly or indirectly, it must be nearer to the solution i.e. nearer to base criteria.

A procedure having these two properties is called a well defined procedure and can be defined recursively. Recursive procedure can be implemented in various programming languages but compilers of some programming languages are not able to handle recursive procedures because they do not have stack mechanism required by the recursive procedures. Programming language such as PASCAL, ALGOL, C, and C++ can be used to implement recursive procedure calls.

Algorithm : Calculate the value of n! recursively.

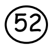
```
Factorial(n)
If n = 0 Then
    Set Fact = 1
    Return
Else
    Set Fact = n × Factorial(n – 1)
    Return
[End If]
```

Q.4Attempt the following (any THREE) **[15]**

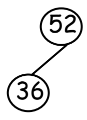
Q.4(a) Make a binary search tree by inserting the following numbers in sequence **[5]**

52 36 98 29 123 39 15 56 31 365 278 45 72

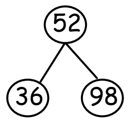
Ans.: 1)



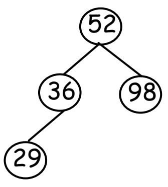
2)



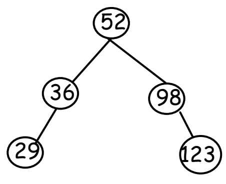
3)



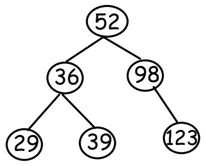
4)



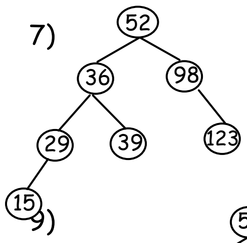
5)



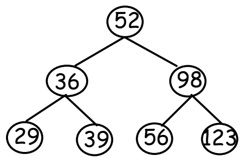
6)



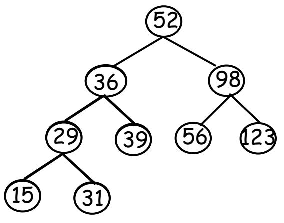
7)

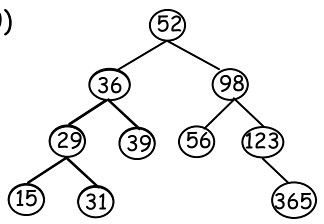


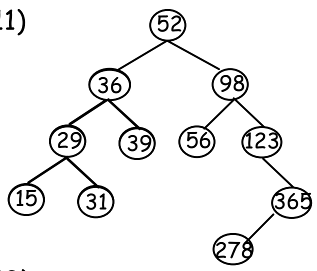
8)

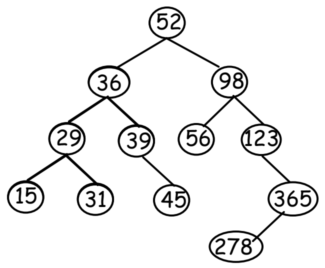


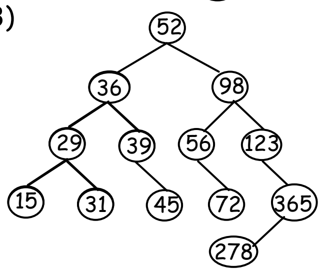
9)



10)


11)


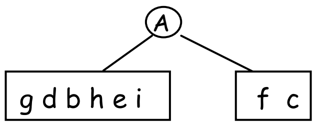
12)


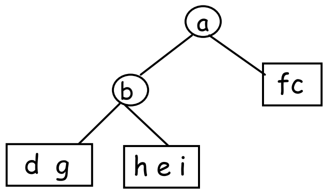
13)


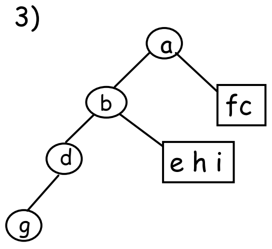
Q.4(b) Draw the binary tree whose inorder and preorder traversals are : [5]

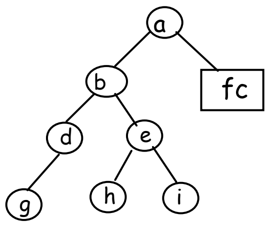
In–order : g d b h e i a f c

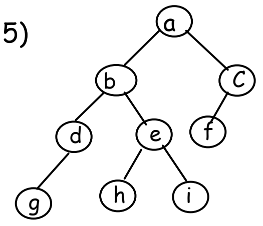
Pre–order : a b d g e h i c f

Ans.: 1)


2)


3)


4)


5)


Q.4(c) What is AVL tree? How balancing is done in AVL tree? Explain with example. [5]

Ans.: AVL tree is also called as height balanced tree. AVL tree is named after its discovers Adelson, Velskii and Landes. A binary tree is said to be height balanced tree if the nodes of the tree are organized in such a way that the difference in the heights of the left subtree and the right subtree of any node in the tree is less than or equal to one, if the difference in the heights becomes more than one then the tree is unbalanced. In height balanced trees additional field is associated with each node of the tree. This additional field stores the balance factor (bf) of the node. The balance factor (bf) of the node is the difference of the heights of its left subtree and right subtree. The structure of the node will be Left Info Bf Right.

The balance factor (bf) of the node in an AVL tree can be –1, 0 or 1

- The bf of the node will be negative if the height of its left subtree is less than the height of its right subtree
- The bf of the node will be zero if the height of its left subtree is equal to the height of its right subtree

- The bf of the node will be positive if the height of its left subtree is less than the height of its right subtree

Balancing in the AVL tree is done using rotations. To perform the rotation we mark a node which is the nearest ancestor to the newly inserted node whose balance factor has become other then –1, 0 or 1. This marked node is known as pivot node.

There are 4 types of rotations
Left–Left Rotation – When the new node is to be inserted in the left subtree of the pivot node then left–left rotation can be performed.

Q.4(d) Write an algorithm to sort elements in ascending order using insertion sort. Explain with example. [5]

Ans.: Consider an unsorted array A of size 8 shown below :

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
22	35	17	8	13	44	5	28

Initially, we consider that the whole list of elements is divided into two parts i.e. the sorted part consisting of the 1st element of the array and the unsorted part consisting of the remaining n – 1 elements. The division of the whole list in two parts is shown below.

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
22	35	17	8	13	44	5	28

Now, starting from the first element of the unsorted part, we will insert elements are by one into the sorted part. Here, the first element A[2] = (= 35) of the unsorted part is compared with the element A[1] (= 22) in the sorted part of the list. As A[2] > A[1] the element A[2] will remain at its position as shown below :

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
22	35	17	8	13	44	5	28

Now, the sorted part of the list has 2 elements and remaining 6 elements are present in the unsorted part of the list.

Now, comparing element A[3] (= 17), the first element of the unsorted part of the list, with the elements of the sorted part of the list i.e. A[2] (= 35) and A[1] (= 22), for element A[3] will be inserted at the 1st position of the list as shown below:

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
17	22	35	8	13	44	5	28

Now, the sorted part of the list has 3 elements and remaining 5 elements are present in the unsorted part of the list.

Now, it's the turn for the element A[4] (= 8), which will be compared with the element in sorted part of the list until an element smaller than A[4] is found. As the element A[3] (= 35), A[2] (= 22), and A[1] (= 17) are greater than A[4] (= 8), so elements A[3], A[2], and A[1] will be shifted one position towards right and the element A[4] will be inserted at position A[1] as shown below :

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
8	17	22	35	13	44	5	28

Now, the sorted part of the list has 4 sorted elements and remaining 4 elements are present in the unsorted part of the list.

In the next step, the element A[5] = (= 13) will be inserted at position A[2] after shifting elements A[2], A[3] and A[4] one position towards right as shown in figure below :

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
8	13	17	22	35	44	5	28

In the next step, element A[6] will remain at the same position because it is the largest element as compared to the elements in the sorted part of the list.

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
8	13	17	22	35	44	5	28

Next, element A[7](= 5) will be inserted at position A[1] as shown below :

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
5	8	13	17	22	35	44	28

In the final step, the element A[8] (= 28) will be inserted at position A[6] and the resulting sorted list will be :

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
5	8	13	17	22	28	35	44

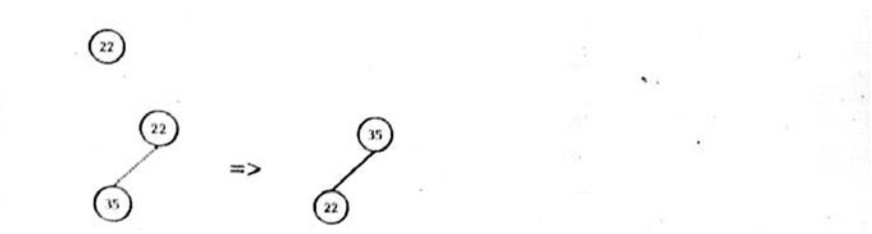
Thus, the given array has beed sorted.

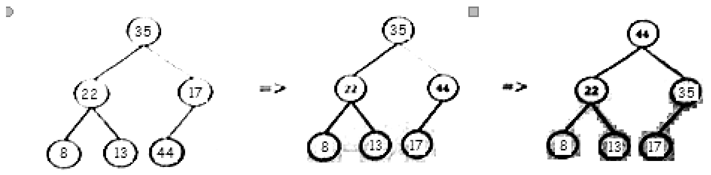
Q.4(e) Sort the following data elements using heap sort algorithm. [5]
22, 35, 17, 8, 13, 44, 5, 28

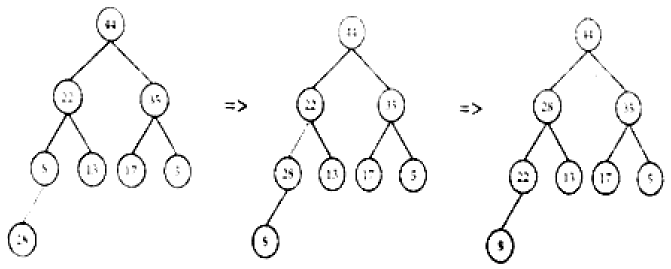
Ans.:

22	35	17	8	13	44	5	28

First of all array will be converted into mac heap as shown below







44	28	35	22	13	17	5	8

Step 1 : Delete the largest element 44 and place it at the last position in the array. Reheap the remaining elements in the first seven positions. After completion the array will be as shown below

35	28	17	22	13	8	5	44

Step 2: In the second step 35 will be removed and the remaining 5=6 elements in the array will be reheaped in the first six positions and the deleted element will be placed at the 2nd last position of the array as shown below

28	22	17	5	13	8	35	44

Step 3 : In the third step 28 will be removed and the remaining elements in the array will be reheaped in the first six positions and the deleted element will be placed at the 3rd last position of the array as shown below

22	13	17	5	8	28	35	44

Step 4 : In the fourth step 22 will be removed and the remaining elements in the array will be reheaped in the first six positions and the deleted element will be placed at the 4th last position of the array as shown below

17	13	8	5	22	28	35	44

Step 5: In the fourth step 17 will be removed and the remaining elements in the array will be reheaped in the first six positions and the deleted element will be placed at the 5th last position of the array as shown below

13	5	8	17	22	28	35	44

Step 6 : In the fourth step 13 will be removed and the remaining elements in the array will be reheaped in the first six positions and the deleted element will be placed at the 6th last position of the array as shown below

8	5	13	17	22	28	35	44

Step 7 : In the fourth step 8 will be removed and the remaining elements in the array will be reheaped in the first six positions and the deleted element will be placed at the 6th last position of the array as shown below

5	8	13	17	22	28	35	44

Q.4(f) Sort the following elements using Merge Sort **[5]**
23, 56, 13, 34, 78, 62, 98, 63, 49, 82

Ans.:

23	56	13	34	78	62	98	53	49	82
23	56	13	34	78	62	98	53	49	82
23	56	13	34	78	62	98	53	49	82
23	56	13	34	78	62	98	53	49	82
13	23	56	34	78	53	62	98	49	82
13	23	34	56	78	49	53	62	82	98
13	23	34	49	53	56	62	78	82	98

Q.5Attempt the following (any THREE) **[15]**

Q.5(a) List different hashing methods. Explain with example any two of them. **[5]**

Ans.: Hashing is mainly a searching technique. However the main aim of hashing function is to map a relatively large domain of key values to a small range of addresses. Commonly used hashing methods are:

- Mid Square Method
- Division – Remainder Method
- Folding Method
- Shifting Method

- **Mid square Method:** It is one of the most commonly used for address calculation. In this method the key value is squared and digits are taken from the middle of the square of the key value. The number of digits to be picked up from the middle of the square usually comes from the number of digits required for relative address range. The following table shows relative address for source key values.

Key Value	Squared key value	Relative	Address
5010	25100100	100	Collision
5016	25160256	160	
5142	26440164	440	
5176	26790976	790	
5301	28100601	100	
5400	29160000	160	Collision

(ii) Division Remainder Method: The division remainder method of calculating addresses is one of the methods which comes first into existence. In this method for mapping key to address, the key is divided by a number which is approximately equal to the number of available addresses and the remainder generated is taken as relative address for the record. In this method, selection of approximate divisor is of great importance in distributing the keys to the address space evenly. Generally a prime number which is nearly equal to number of available addresses is taken as divisor. The following table shows the relative address for some key values.

Key Value	Remainder After Dividing the key By 997 (Relative Address)
1098	101 ———————— Collision
1120	123
1185	188
1230	233
1410	413
1866	869
2095	101
2117	123 ———————— Collision

Q.5(b) Describe following collision resolution techniques. **[5]**
(i) Linear probing **(ii) Chaining**

Ans.: **Linear Probing:** In Linear Probing, if a record with key K is mapped to an address which is already occupied by some other record then linear search starting from address generated by hash function is done until a free location is found for the storage of the new record. An empty location is always met if it is available. Linear probing algorithms are implemented in such a way that, starting from the address generated by the hash function, the status of each location is checked whether it is empty or not. Once an empty location is encountered, record is stored over there. On the other hand if the end of the address space is reached without meeting any free location then rather quitting, we starts looking for the empty location from the first address of the address space until a free location is encountered or location having address which was generated by the hash function is encountered again. In the later case the file is completely full and there is no space to accommodate the new record.

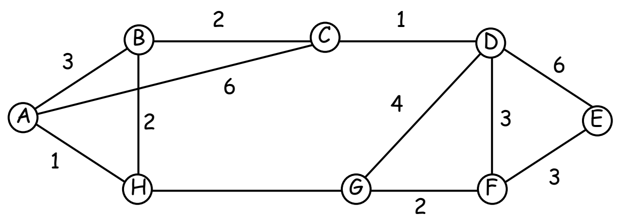
For using linear probing technique the hash function will be:
Here H(K) is the Hash function used to find the relative address before collision.
So, H(K) = K mod m p is the probe number which can be 0, 1, 2 m-1, and m is the size of the hash table. Consider the table of size 10. Suppose we want to insert some Chaining – One of the most efficient methods of handling colliding records is called synonym chaining. In this approach, colliding records. A separate one-way list is maintained for each set of records which are colliding to the same location. The records in these lists are not kept according to any specific order but whenever a new collision occurs, the colliding record is inserted at the front of the appropriate linked list. In this approach of handling collision, instead of storing records at locations in the address space, pointers are placed where each pointer points to the chain of records which shares the same hash location.

Q.5(c) Explain with example Prim's algorithm to find the minimum spanning tree (MTS) [5]

Ans.: Prims algorithm starts by choosing an arbitrary vertex node of the graph. That arbitrary node is then considered as the root of the tree. At each step a new node will be added to the tree. This algorithm stop when all the nodes from the graph are added to the tree. The nodes of the MST spans at the rate of one node at a time.

```
1.  Let N be the set of selected nodes in the tree. Initially N = {φ}
2.  Let M be the set of selected edges in the tree. Initially M = {φ}
3.  Let S be the set of graph edges
4.  Initialize N with initial vertex. (Starting Node).
5.  Repeat while | N | ≠ n
6.  Let (u, v) be the least cost edge in S such that U ∈ N and v not ∈ N.
7.  If there is no such edge. Then GoTo step 10
8.  Add the node v to N
9.  Add Edge (u, v) to M and set S = S–{u, v}
    [End loop]
10. If | N | = n Then
    Print "M contains the edges of MST"
    Exit
    Else
        Print "Graph is not connected "
    [End If]
11. Exit
```

Consider the following graph on which we will apply Prim's algorithm.



Set of selected nodes $N^2 = \{\phi\}$

Set of Selected edges $M = \{\phi\}$

Set of Graph edges $S = \left\{\underset{3}{ab}, \underset{2}{bc}, \underset{1}{cd}, \underset{6}{de}, \underset{3}{ef}, \underset{2}{fg}, \underset{10}{gh}, \underset{1}{ha}, \underset{6}{ac}, \underset{2}{bh}, \underset{4}{gd}, \underset{3}{df}\right\}$

Starting with node A we choose the least cost edge which start from node A and do not from a cycle. So

N = {P, H} M = {Ah} $S = \left\{\underset{3}{AB}, \underset{2}{BC}, \underset{1}{CD}, \underset{6}{DE}, \underset{3}{EF}, \underset{2}{FG}, \underset{10}{GH}, \underset{6}{AC}, \underset{2}{BH}, \underset{4}{GD}, \underset{3}{DF}\right\}$

Now the least cost edge BC is chosen

N = {P, h, B} M = {Ph, BH}

$S = \left\{\underset{3}{AB}, \underset{1}{CD}, \underset{6}{DE}, \underset{3}{EF}, \underset{2}{FG}, \underset{10}{GH}, \underset{6}{AC}, \underset{2}{BH}, \underset{4}{GD}, \underset{3}{DF}\right\}$

Now the least cost edge BC is chosen

N = {A, H, B, C} M = {AH, BH, BC}

$S = \left\{\underset{3}{AB}, \underset{1}{CD}, \underset{6}{DE}, \underset{3}{EF}, \underset{2}{FG}, \underset{10}{GH}, \underset{6}{AC}, \underset{2}{BH}, \underset{4}{GD}, \underset{3}{DF}\right\}$

Now the least cost edge CD is chosen.

M = {A, H, B, C, D} M = {AH, BH, BC, CD}

$S = \{\underset{3}{AB}, \underset{6}{DE}, \underset{3}{EF}, \underset{2}{FG}, \underset{10}{GH}, \underset{6}{AC}, \underset{4}{GD}, \underset{3}{DF}\}$

Now the least cost edge DF is chosen

N = {A, H, B, C, D, F}

M = {AH, BH, BC, CD, DF}

$S = \{\underset{3}{AB}, \underset{6}{DE}, \underset{3}{EF}, \underset{2}{FG}, \underset{6}{AC}, \underset{4}{GD}\} \underset{10}{GH}\}$

Now the least cost edge GF is CHOSEN

N = {A, H, B, C, D, F, G}

M = {AH, BH, BC, CD, DF, FG}

S = {$\underset{3}{AB}$, $\underset{6}{DE}$, $\underset{3}{EF}$, $\underset{10}{GH}$, $\underset{6}{AC}$, $\underset{4}{GD}$}

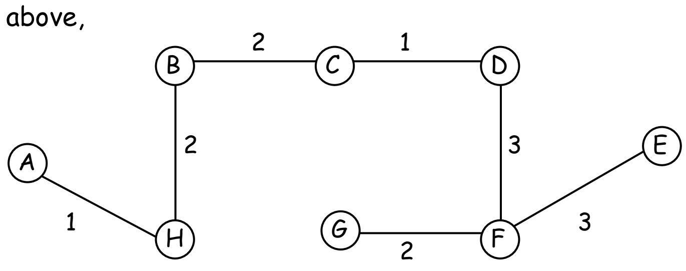
Now, the least cost edge EF is chosen

N = {A, H, B, C, D, F, G, E}

M = {AH, BH, BC, CD, DF, EG, EF}

S = {$\underset{3}{AB}$, $\underset{6}{DE}$, $\underset{3}{EF}$, $\underset{10}{GH}$, $\underset{6}{AC}$, $\underset{4}{GD}$}

The above,



The above is the resulting MST whose weight is 14.

Q.5(d) Explain Warshall's algorithm of finding path matrix of graph. [5]

Ans.: **Path matrix for a Graph:** Consider a directed graph G of order n having set of vertices **Vg** and set of edges **Eg**. The path matrix **P** of the graph **G** will be a **n x n** matrix. The significance of finding the path matrix is that it shows the existence or absence of path between the pair of vertices. In the path matrix **P** for any graph **G,** if the entry corresponding to any pair of vertices is non-zero then it represents that there exist at **Warshall's Algorithm:** It is used to compute the existence of paths within a directed graph using Boolean operators and matrices. Create an adjacency matrix for the graph and then based on the adjacency matrix define **n** boolean matrices $\mathbf{A^1}$, $\mathbf{A^2}$, $\mathbf{A^J}$, **A"** Let $a^k ij$ denotes (i, j) th entry of matrix A^k then $a^k i.j$ is 1 if there exist a simple path from vi to vj which does not use any other vertex as intermediate point and 0 if there is no

Q.5(e) List graph traversal techniques. Write and explain algorithm for any one. Give suitable example. [5]

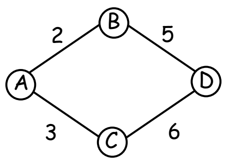
Ans.: **Path matrix for a Graph:** Consider a directed graph G of order n having set of vertices **Vg** and set of edges. **Eg**. The path matrix **P** of the graph **G** will be a **n x n** matrix. The significance of finding the path matrix is that it shows the existence or absence of path between the pair of vertices. In the path matrix **P** for any graph **G,** if the entry corresponding to any pair of vertices is non-zero then it represents that there exist at **Warshall's Algorithm:** It is used to compute the existence of paths within a directed graph using Boolean operators and matrices. Create an adjacency matrix for the graph and then based on the adjacency matrix define **n** boolean matrices $\mathbf{A^1}$, $\mathbf{A^2}$, $\mathbf{A^J}$, **A"**

Let $a^k ij$ denotes (i,j)th entry of matrix A^k then $a^k i.j$ is 1 if there exist a simple path from vi to vj which does not use any other vertex as intermediate point and 0 if there is no

```
                Set P[i][j] = A[i][j]
            [End Loop]
        [End Loop]
Step 3: Repeat Step 4 and 5 For k = 1 to n
Step 4:         Repeat Step 5 For I = 1 to n
Step 5:             Repeat For j = 1 to n
                        Set P[i][j] OR (P[i][k] And P[k][j])
                    [End Loop]
                [End Loop]
Step 6: Exit
```

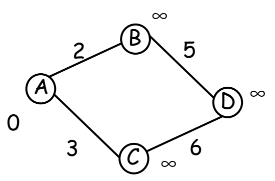
Q.5(f) Explain with example Dijkstra's shortest path algorithm. **[5]**

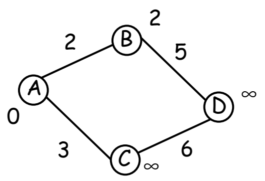
Ans.: Dijkstra's algorithm is an application of finding the shortest path from source vertex to all other vertices. This type of shortest path is known as single source shortest path. It is a greedy algorithm whose essential feature is the order in which the paths are determined. It starts by initializing a vertex in the graph.



We give permanent label to A with Cost O and all other vertices are given temporary, labels with costs ∞.

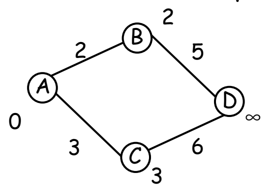
In the next step to select the least cost edge connecting a vertex with a permanent label to vertex with temporary label.



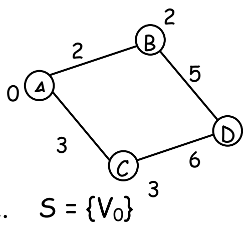


Value = 2 to = 2

Now find the least cost edge from any of the permanent labels to a temporary label and then make that label permanent.



In the final step we select the least cost edge from any permanent labels to a temporary vertex.



Value = 5 + 2 = 7

```
1. S = {V_0}
2. Repeat step for i =1 to n=1 to n
       D[1] = C [V_0, i]
   [End loop]
3. Repeat step 4 to 6 for i = 1 to n–1
4. Choose a node j in v–s such that D[j] is minimum.
5. Add j to s
6. Repeat step for each vertex k in v–s
       D [k] = Min (D [k], D[j] + C[j, k])
   [End Loop]
7. Exit
```

❑ ❑ ❑ ❑ ❑