S.Y. Diploma : Sem. III
[CO/CM/IF/CW]

Data Structures Using 'C'

Time: 3 Hrs.] Prelim Question Paper Solution **[Marks : 70**

Q.1 Attempt any FIVE of the following : [10]

Q.1(a) Explain the concept of information, Next, Null pointer and empty list with respect to link list. [2]

Ans.:
- **Info Field:** It is used to store data inside the node.
- **NEXT:** It is used to store reference of next element in the list.
- **Null Pointer:** It is used to specify end of the list. The last element of list contains NULL pointer to specify end of list.
- **Empty List:** A linked list is said to be empty if head (start) node contains NULL pointer.

Q.1(b) (ii) Describe priority queue with example. [4]

Ans.: A priority queue is a queue in which the intrinsic ordering among the elements decides the result of its basic operations i.e. the ordering among the elements decides the manner in which Add and Delete operations will be performed. In a priority queue,

1) Each element is assigning a priority.
2) The elements are processed according to, higher priority element is processed before lower priority element and two elements of same priority are processed according to the order of insertion.

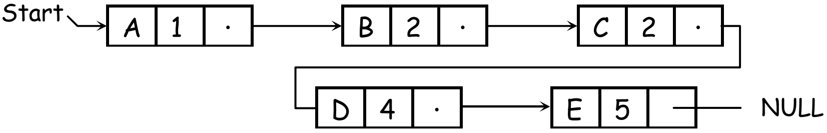
(Represent either with array or linked list)

Array representation: Array element of priority queue has a structure with data, priority and order. Priority queue with 5 elements:

C,1,4	B,3,2	B,3,5	A,4,1	D,5,3

OR

Linked Representation:

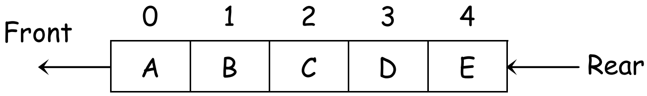


Above figure shows priority. Queue with 5 elements where B & C have same priority number.

Each node in above priority queue contains three items.
(i) Information field INFO
(ii) A priority number PR No
(iii) Link Next

Q.1(c) Define queue? Explain how pointer front and rear related to queue with diagram. **[2]**

Ans.: A Queue is an ordered collection of items. It has two ends, front and rear. Front end is used to delete element from queue. Rear end is used to insert an element in queue. Queue has two ends; the element entered first in the queue is removed first from the queue. So it is called as FIFO list.



Element is inserted in the list at a position indicated by rear end. For inserting an element rear end is incremented by one.

Element can be deleted from a list from the position indicated by front end. For deleting an element rear end is incremented by one.

Q.1(d) Convert the following infix expression to its postfix form using stack A + B – C*D/E + F. **[2]**

Ans.:

Symbol #	Scanned	Stack	Expression
1	(	(	nil
2	A	(	A
3	+	(+	A
4	B	(+	AB
5	-	(-	AB+
6	C	(-	AB+C
7	*	(-*	AB+C
8	D	(-*	AB+CD
9	/	(-/	AB+CD*
10	E	(-/	AB+CD*E
11	+	(+	AB+CD*E/-
12	F	(+	AB+CD*EF/-
13	)	nil	AB+CD*E/-F+

Q.1(e) Describe given two types of graphs: Directed and undirected graph. **[2]**

Ans.: Directed Graph

A directed graph G is also called digraph which is the same as multigraph except that each edge e in G is assigned a direction or in other words each edge in G is identified with an ordered pair (U, V) of nodes in G rather than an unordered pair. Figure 1 illustrates three directed graphs:

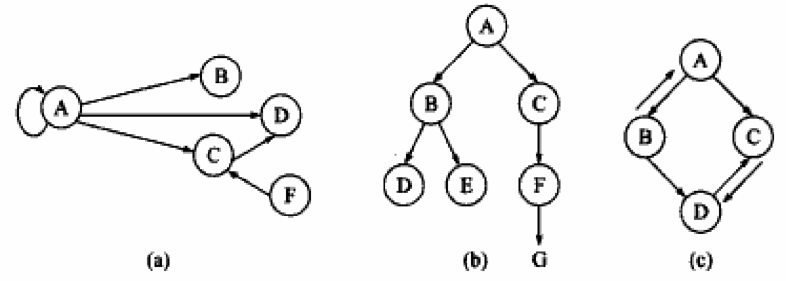


Fig.1: Directed Graphs

The set of edges for the graph in figure 1(b) is {<A, B>, <A, C>, <A, D>, <C, D>, <F, C> <E, G>, <A, A>). We use angle brackets to indicate an ordered pair.

A directed graph G is said to be connected or strongly connected if for each pair (U, V) of nodes in G there is a path from U to V and there is also a path from V to U.

A directed graph G is said to simple if G has no parallel edges. A simple graph G may have loops but it cannot have more than one loop at a given node.

Undirected Graph

An undirected graph G is a graph in which each edge e is not assigned a direction. Examples of undirected graphs can be seen in figure 2 given below:

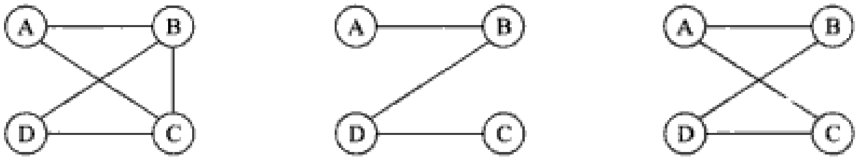


Fig. 2: Undirected Graph

Q.1(f) Explain time complexity and space complexity. **[2]**

Ans.: Time complexity

Time complexity of a program/algorithm is the amount of computer time that it needs to run to completion. While calculating time complexity, we develop frequency count for all key statements which are important and basic instructions of an algorithm.

Frequency count for algorithm A is 1 as a=a+1 statement will execute only once. Frequency count for algorithm B is n as a=a+1 is key statement executes n time as the loop runs n times. Frequency count for algorithm C is n as a = a + 1 is key statement executes n2 time as the inner loop runs n times, each time the outer loop runs and the outer loop also runs for n times.

Space Complexity :

Space complexity of a program/algorithm is the amount of memory that it needs to run to completion. The space needed by the program is the sum of the following components :

- Fixed space requirements : It includes space for instructions, for simple variables, fixed size structured variables and constants.
- Variable time requirements : It consists of space needed by structured variables whose size depends on particular instance of variables.

Q.1(g) Write any four applications of data structure. [2]

Ans.: Following are some real world applications of Data Structures:

(i) Mostly, Dictionaries are built using a Hash Table Data Structure. Hash Tables are also used for caches, database indexing, fast data lookup - symbol table for compilers.

(ii) Trees helps to build File System, Parsers.

(iii) B-Trees helps to build Database Design.

(iv) BSP tree is used in 3D computer graphics.

(v) Radix tree is used in IP routing table.

(vi) Stack : Real word applications like Java virtual machine, expression evaluation, UNDO\REDO operation in word processors etc.

(vii) Queues : Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.

(ix) Priority queues : process scheduling in the kernel.

Q.2 Attempt any THREE of the following : [12]

Q.2(a) Explain stack overflow and underflow conditions with example. [4]

Ans.: Stack Overflow

Sometimes when a new data is to be inserted into a stack but there is no available space, then this situation is known as Stack Overflow.

Example: If a stack has maximum capacity of 3 elements, and these three are already occupied and the programmer tries to push in a fourth element, then it will lead to overflow.

Stack Underflow

Sometimes when one wants to delete data from a stack but the stack is already empty, then this situation is known as Stack Underflow.

Example: If a stack is empty and a programmer tries to execute the POP operation on it, then it will cause the Underflow error.

Q.2(b) Write an algorithm to insert and delete an element from queue. [4]

Ans.: Insert procedure :

(i) Check queue full i.e. check rear position. If it is full then display message and return to calling function. Otherwise go to step 2.

(ii) Increment Rear by 1.

(iii) Insert element with rear pointer.

(iv) Check position of front. If inserted element is first element then set front to 0.

(v) Return to calling function

Delete procedure:

(i) Checks for queue empty i.e. check front position. If it is empty then display message and return to calling function. Otherwise go to step 2.

(ii) Check front and rear positions. If front and rear both are equal then show queue empty otherwise increment front by one.

(iii) Return to calling function

Q.2(c) Differentiate between tree and graph w.r.t. any 4 parameters. [4]

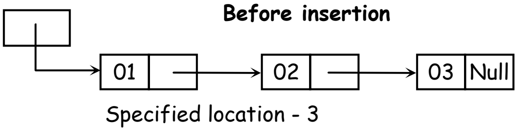
Ans.: Differentiate between tree and graph

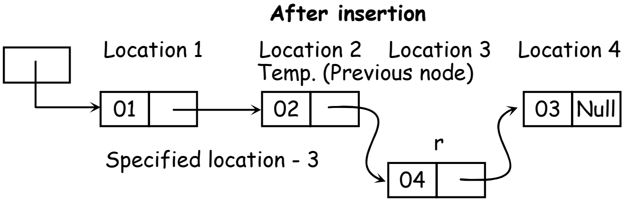
	Tree	**Graph**
(i)	Tree has only one path between two vertices.	Graph can have multiple paths between two vertices.
(ii)	Tree has no loops.	Graph can have loops.
(iii)	Tree has a root node.	Graph has no root node.
(iv)	In tree, there is concept of parent and child	In graph, there is no concept of parent and child.

Q.2(d) Write an algorithm to insert a node in between in a link list. [4]

Ans.: Algorithm to insert an element insert a node in between in a link list.

```
void add_at_specified(struct node *q, intloc, int no)
{
Step 1: Begin
Step 2: Allocate memory for temp node and new node (r); set r->info = no
Step 3: Initialize temp to the beginning of the linked list.
Step 4: Traverse temp till the previous node of the specified location given
by loc in the linked list (Previous node = temp node)
Step 5: Set r->next = temp-next;
        Set temp->next = r;
Step 6: Stop
}
```


Before insertion


After insertion

Q.3 Attempt any THREE of the following : [12]

Q.3(a) Write an algorithm for inorder traversal of binary tree. [4]

Ans.: The inorder traversal starts at the root of the binary tree. Then moves to leftchild node and this is repeated till there is no leftchild node. When there is no leftchild node process the data of the node and after processing it tries to move to right child. If there is no right child node it moves up by one node, process the data and tries to move right. To traverse a non-empty binary tree in inorder:

Inorder Traversal (Left- Data- Right)

a. Traverse the left sub tree in Inorder.

b. Visit the root.

c. Traverse the right subtree in Inorder.

Algorithm:

```
Step 1: temp =root
Step 2: If temp is not equal to NULL
Step 3: In order (temp - > left)
Step 4: process temp-> data          // (node visited)
Step 5: In order (temp ->right)
Step 6: stop
```

Q.3(b) For the following directed graph : **[4]**

(i) *Give adjacency matrix representation.*

(ii) *Give adjacency list representation.*

Ans.: **Adjacency List:**

A: B
B: D
C: A,D
D: A

Adjacency Matrix:

	A	B	C	D
A	0	1	0	0
B	0	0	0	1
C	1	0	0	1
D	1	0	0	0

Q.3(c) Write an algorithm to implement binary search. **[4]**

Ans.: (i) Binary search algorithm locates the position of an element in a sorted array.

(ii) Binary search works by comparing an input value to the middle element of the array.

(iii) The comparison determines whether the element equals the input, less than the input or greater.

(iv) When the element being compared to equals the input the search stops and typically returns the position of the element.

(v) If the element is not equal to the input then a comparison is made to determine whether the input is less than or greater than the element.

(vi) Depending on which it is the algorithm then starts over but only searching the top or bottom of the array's elements.

Q.3(d) Implement C Program for performing following operations on Array : Insertion, Display. **[4]**

Ans.:

```
#include<stdio.h>
#include<conio.h>
#define MAX 5

void insert(int *, int pos, int num);
void display(int *);

void main()
{
```

```
int arr[5];
clrscr();
insert(arr,1,11);
insert(arr,2,12);
insert(arr,3,13);
insert(arr,4,14);
insert(arr,5,15);
printf("\nElements of array are as follows: ");
display(arr);
getch();
}

void insert(int *arr, int pos, int num)
{
for(int i=MAX-1; i>=pos;i--)
{
arr[i]=arr[i-1];
}
arr[i]=num;
}

void display(int *arr)
{
for(int i=0; i<MAX; i++)
{
printf("%d\t", arr[i]);
}
}
```

Q.4 Attempt any THREE of the following : [12]

Q.4(a) Explain the concept of double ended queue. [4]

Ans.: Double Ended Queue :

(i) A double-ended queue or dequeue is an abstract data structure that implements a queue for which elements can only be added to or removed from the front (head) or back (tail).

(ii) It is also often called a head-tail linked list.

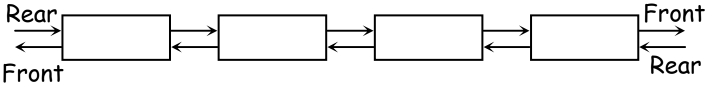
(iii) Dequeue is a special type of data structure in which insertions and deletions will be done either at the front end or at the rear end of the queue.

(iv) The operations that can be performed on dequeues are

(a) Insert an item from front end

(b) Insert an item from rear end

(c) Delete an item from front end
(d) Delete an item from rear end
(e) Display the contents of queue



Q.4(b) Describe circular queue with an example. [4]

Ans.: Circular Queue

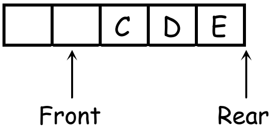
Circular queue are the queues implemented in circular form rather than in a straight line. Circular queues overcome the problem of unutilized space in linear queue implemented as an array. The main disadvantage of linear queue using array is that when elements are deleted from the queue, new elements cannot be added in their place in the queue, i.e. the position cannot be reused.

After rear reaches the last position, i.e. MAX-1 in order to reuse the vacant positions, we can bring rear back to the 0th position, if it is empty, and continue incrementing rear in same manner as earlier. Thus rear will have to be incremented circularly. For deletion, front will also have to be incremented circularly.

Rear can be incremented circularly by the following code.

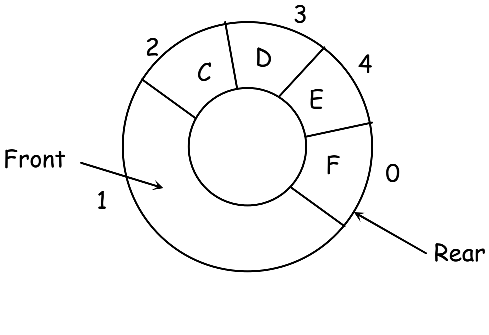
```
If ((rear == MAX-1) and (front !=0)
        Rear =0;
Else
        Rear= rear +1;
```

Example: Assuming that the queue contains three elements.



Now we insert an element F at the beginning by bringing rear to the first position in the queue. This can be represented circularly as shown.

In the above example, if another element, G is added to the queue, i.e. rear and front coincide. But rear and front coincide even when the queue is full is empty. Thus rear and front cannot be used for both i.e. to check for empty queue as well as the condition for a full queue.



The rear ==front condition is used to check for the empty queue since initially both are initialized to the same value. Thus to check for queue full condition there are three methods.

1) Use a counter to keep track of the number of elements in the queue. If this counter reaches to MAX the queue is full.
2) After remove operation rear = front, then set to -1. After add operation if rear = front then will say that queue is full.
3) By checking (rear + 1) % MAX== front.

Q.4(c) Show the effect of INSERT and DELETE operations on to the Linear queue of size 10. The Linear queue sequentially contains 10, 20, 30, 40, and 50 where 10 is at front of the queue. Show diagrammatically the effect of : **[4]**

(i) INSERT (12) **(ii) INSERT (34)**

(iii) DELETE **(iv) INSERT (56)**

Ans.: **Initial Queue:**

10	20	30	40	50

Step 1: INSERT (12)

10	20	30	40	50	12

Step 2: INSERT (34)

10	20	30	40	50	12	34

Step 3: DELETE

20	30	40	50	12	34

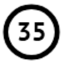
Step 4: INSERT (56)

20	30	40	50	12	34	56

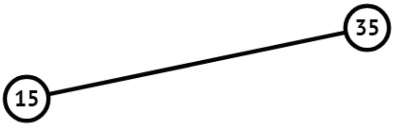
Q.4(d) Construct the binary search tree using following elements: 35,15,40,7,10,100,28,82,53,25,3. Show diagrammatically each step of construction of BST. **[4]**

Ans.: **Construction of Binary Search Tree (BST):**

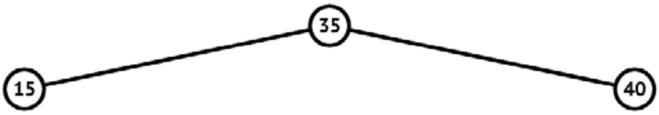
Step 1:



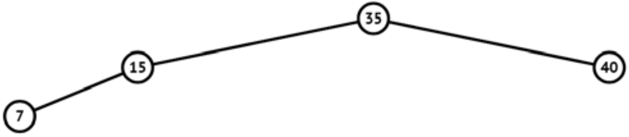
Step 2:



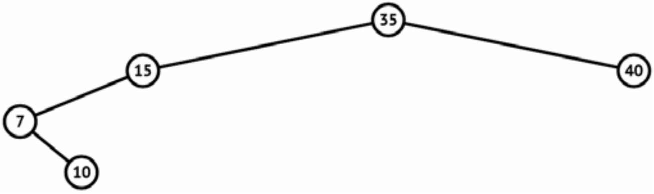
Step 3:



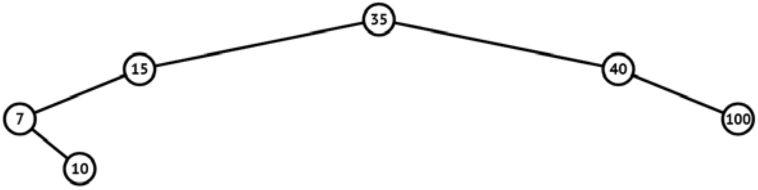
Step 4:



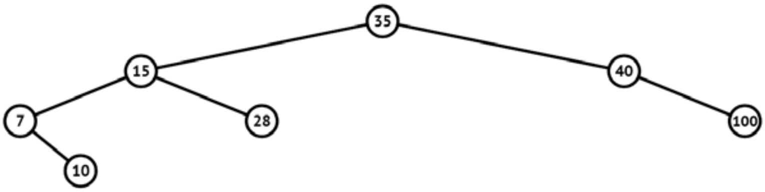
Step 5:



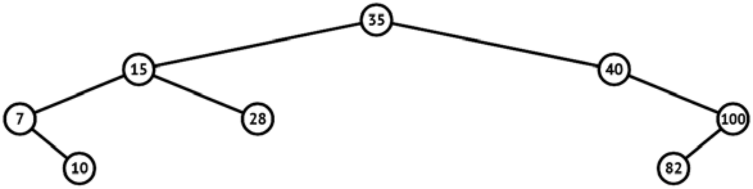
Step 6:



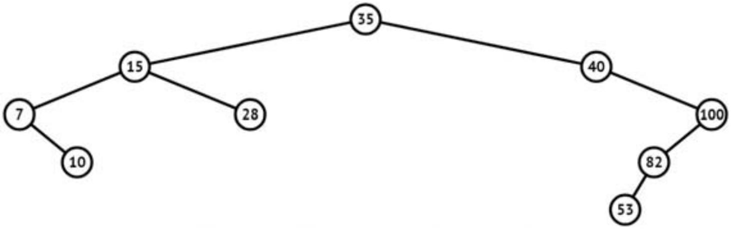
Step 7:



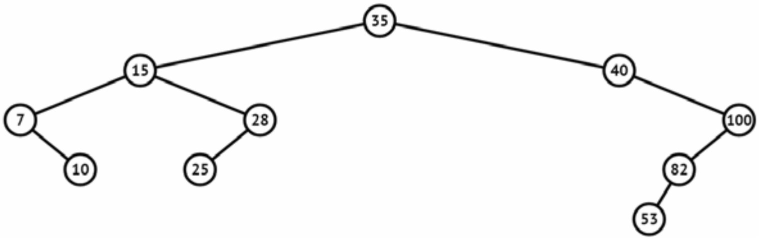
Step 8:



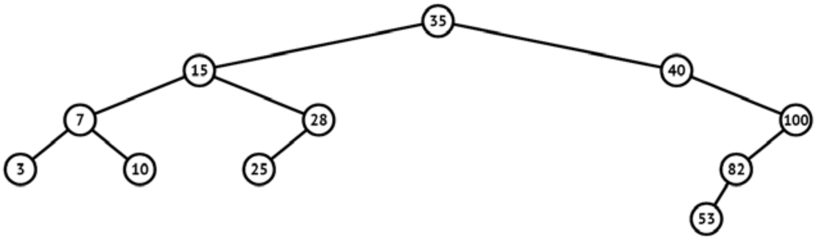
Step 9:



Step 10:



Step 11:



Q.5 Attempt any TWO of the following : [12]

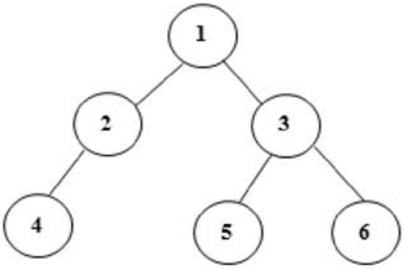
Q.5(a) Write an algorithm to count number of nodes in singly linked list. [6]

Ans.:

```
Step 1: Count = 0. SAVE = FIRST.
Step 2: Repeat step 3 while SAVE != NULL.
Step 3: Count= Count + 1. SAVE=SAVE->LINK.
Step 4: Return Count.
```

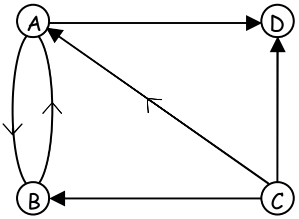
Q.5(b) From the given tree complete six answers : [6]

(i) Degree of tree
(ii) Degree of node 3
(iii) Level of node 5
(iv) Indegree of node 3
(v) Outdegree of node 3
(vi) Height of tree

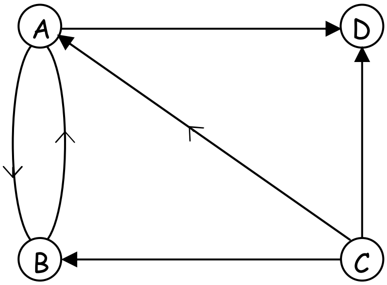


Ans.: (i) Degree of Tree = 3
(ii) Degree of node 3 = 3
(iii) Level of Node 5 = 2
(iv) Indegree of Node 3 = 1
(v) Outdegree of Node 3 = 2
(vi) Height of tree = 3

Q.5(c) Consider the graph given in Figure. Find its adjacency matrix and adjacency link representation. **[6]**



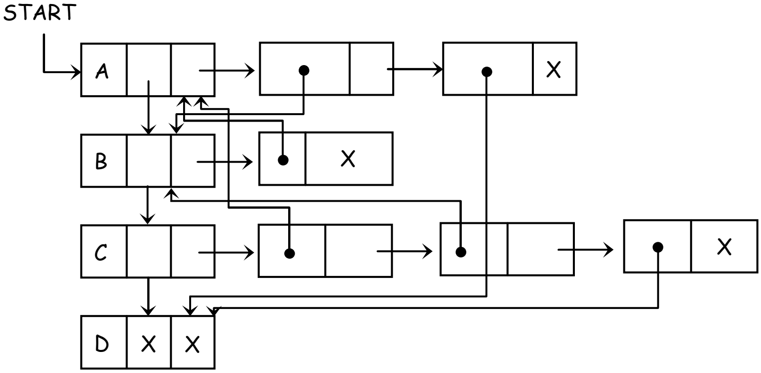
Ans.:



Adjacency Matrix

$$A = \begin{array}{cc} & \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{array}{c} A\\ B\\ C\\ D \end{array} & \begin{bmatrix} 0&1&0&1\\ 1&0&0&0\\ 1&1&0&1\\ 0&0&0&0 \end{bmatrix} \end{array} \quad \textbf{OR} \quad \begin{array}{cc} & \begin{array}{cccc} A & D & C & B \end{array} \\ \begin{array}{c} A\\ D\\ C\\ B \end{array} & \begin{bmatrix} 0&1&0&1\\ 0&0&0&0\\ 1&1&0&1\\ 1&0&0&0 \end{bmatrix} \end{array}$$

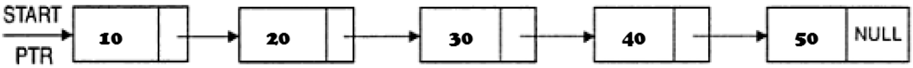
Adjacent nodes can contain two or three fields.



Q.6 Attempt any TWO of the following : **[12]**

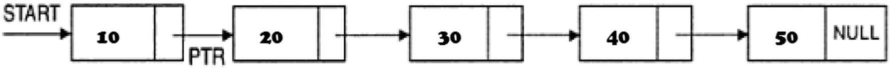
Q.6(a) Create a Singly Linked List using data fields 10, 20, 30, 40, 50. Search a node 40 from the SLL and show procedure step-by-step with the help of diagram from start to end. **[6]**

Ans.: Step 1:



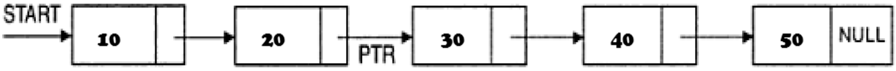
VALUE = 40. PTR != VALUE

Step 2:



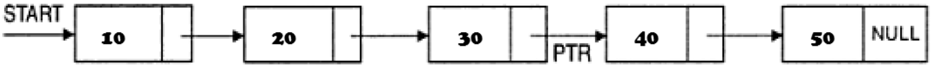
VALUE = 40. PTR != VALUE

Step 3:



VALUE = 40. PTR != VALUE

Step 4:



VALUE = 40. PTR = VALUE

Result: Value was found in the linked list. The pointer points to the node having the value.

Q.6(b) Describe breadth first search traversal in a graph with example. [6]

Ans.: Algorithm for BFS:

(i) Initialize all nodes to ready state.

(ii) Insert starting node in a queue and change its state to waiting state.

(iii) Repeat steps 4 to 6 till the queue becomes empty.

(iv) Remove front node N from queue 2 change its status to visit.

(v) Insert all adjacent nodes of N at the rear end of the queue and change their status to 'waiting state'.

(vi) From the origin find path from source node to destination node or from the queue element list find all nodes that are reachable.

(vii) Stop.

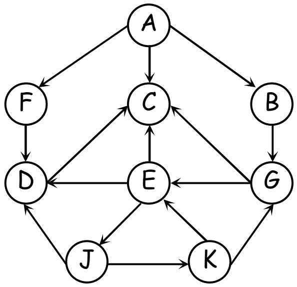
Example :

Front of the queue is set to '0'

Rear of the queue is set to '0'

Queue is used to indicate elements of the graph which are visited.

Origin is used to keep track of origin of each node.



Find all nodes reachable from 'A'

1) Insert A into a queue.

A									

Front=1 Rear=1

Queue=A Origin=0

2) Remove front element A and insert adjacent nodes of a in queue.

	F	C	B						

Queue=A front=1

Origin=O, A, A, A rear=3

3) Remove element F and insert its adjacent nodes in the queue.

		C	B	D					

Queue=A, F, C, B, D front=2

Origin=O, A, A, A, F rear=4

4) Remove front element C and insert its adjacent nodes in the queue (F is already visited)

			B	D					

Queue=A, F, C, B, D front=3

Origin=O, A, A, A, F rear=4

5) Remove front element B and insert adjacent nodes

				D	G				

Queue=A, F, C, B, D, G front=4

Origin=O, A, A, A, F, B rear=5

6) Remove front element D and insert adjacent nodes (C is already visited)

					G				

Queue=A, F, C, B, D, G front=5

Origin=O, A, A, A, F, B, G rear=5

7) Remove front element G and insert adjacent nodes

						E			

Queue=A, F, C, B, D, G, E front=6

Origin=O, A, A, A, F, B, G rear=6

8) Remove front element E and insert its adjacent nodes (D is already a visited node)

							J		

Queue=A, F, C, B, D, G, E, J

Origin=O, A, A, A, F, B, G, E

9) Remove J

Queue = A, F, C, B, D, G, E, J
All nodes readable from A-A, F, C, B, D, G, E, J
Origin= O, A, A, A, F, B, G, E, J

Q.6(c) Convert the given infix string to prefix expression and shows the details of stack at each step. [6]

(A – B/C) ∗ (D ∗ E – F)

Ans.:

⟵

Q	Stack	Prefix
)	Initialize ⟶))	–
F	))	F
–	)) –	F
E	)) –	FE
∗	)) – ∗	FE
D	)) – ∗	FED
C	)	FED ∗ –
∗	) ∗	FED ∗ –
)	) ∗)	FED ∗ –
C	) ∗)	FED ∗ – C
/	) ∗) /	FED ∗ – C
B	) ∗) /	FED ∗ – CB
–	) ∗) –	FED ∗ – CB /
A	) ∗) –	FED ∗ – CB / A
(	) ∗	FED ∗ – CB / A –
Initialize ⟶ (	empty	FED ∗ – CB / A – *

Prefix exp = ∗ – A / BC – ∗ DEF

❑ ❑ ❑ ❑ ❑