Q.2 a. Explain with the help of an example how floating point numbers are stored.

Answer: Pg. No. 21 of C & Data Structures, P.S. Deshpande and O.G. Kakde, Dreamtech Press, 2005

b. What do you understand by forced conversions? Explain with example.

Answer: Pg. No. 26 of C & Data Structures, P.S. Deshpande and O.G. Kakde, Dreamtech Press, 2005

c. Differentiate between logical and arithmetic shift.

Answer: Pg. No. 43 of C & Data Structures, P.S. Deshpande and O.G. Kakde, Dreamtech Press, 2005

d. Do the following conversions:
(i) $(25)_8 = (?)_{16}$
(ii) $(A21)_{16} = (?)_{10}$

Answer: (i) 15
(ii) 2593

Q.3.a. Can any of the three initial expressions in the for statement be omitted? If so, what are the consequences of each omission?

Answer:

- From the syntactic standpoint all three expressions need not be included in the for statement, though the semicolon must be present.
- However the consequences of an omission should be clearly understood.
- The first and third expressions may be omitted if other means are provided for initializing the index and/or altering the index.
- If the second expression is omitted, however, it will be assumed to have a permanent value of 1 (true); thus, the loop will continue infinitely unless it is terminated by some other means, such as break or a return statement.
- As a practical matter, most for loops include all three expression.

b. Write a program that will read a positive integer and determine and print its binary equivalent.

Answer:

```
#include<stdio.h>
#include<conio.h>
void showbits(int h)
        {
                if(h==1)
                printf("%d",h);
                else
                {
                        showbits(h/2);
                        printf("%d",h%2);
                }
        }
void main()
{
        int nu;
        void showbits(int h);
        printf("Num?");scanf("%d",&nu);
        printf("\nBin eq of %d is ",nu);
        showbits(nu);

}
```

c. What is the output of the following program.

```
const int a=124;
void main()
{
        const int *sample();
        int *p;
        p=sample();
        printf("%d",*p);
}
const int *sample()
{
        return (&a);
}
```

Answer: Output = 124

d. Write a C program to reverse a given number.

Answer:

```
#include<stdio.h>
void main()
{
        int num, rno=0,rem=0;
        printf("Input the number to be reversed\");
        scanf("%d",&num);
        while(num !=0)
        {
                rem=num%10;
                rno = rno *10+rem;
                num = num/10;
        }
        Printf(" the reversed number is = %d ", rno);
}
```

Q.4.a. Distinguish between the following
i) int (*m)[5]; and int *m[5]
ii) int (*ptr)(); and int *ptr()

Answer:
i) int (*m)[5] = means m is an integer pointer to the 5th element of the array
int *m[5] = means m is an array of 5 integer pointer

ii) int (*ptr)() = ptr is a pointer to a function that returns return integer
int *ptr() = ptr is a function that return integer pointer

b. Write a program to show how elements of an array can be accessed using pointers.

Answer: **Pg. No. 88 of C & Data Structures, P.S. Deshpande and O.G. Kakde, Dreamtech Press, 2005**

c. With the help of an example show sequence of execution during function calls.

Answer: Pg. No. 104 of C & Data Structures, P.S. Deshpande and O.G. Kakde, Dreamtech Press, 2005

Q5 a. Write a program to copy the contents of one file into another file using command line arguments.

Answer:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

void main(int arg,char *arr[])
{
   FILE *fs,*ft;
   char ch;
   clrscr();
   if(arg!=3)
         {
                  printf("Argument Missing ! Press key to exit.");
                  getch();
                  exit(0);
         }

   fs = fopen(arr[1],"r");
   if(fs==NULL)
         {
                  printf("Cannot open source file ! Press key to exit.");
                  getch();
                  exit(0);
         }

   ft = fopen(arr[2],"w");
   if(ft==NULL)
         {
                  printf("Cannot copy file ! Press key to exit.");
                  fclose(fs);
                  getch();
                  exit(0);
         }

   while(1)
         {
                  ch = getc(fs);
         if(ch==EOF)
                  {
                           break;
                  }
         else
                  putc(ch,ft);
         }
```

```
  printf("File copied succesfully!");
  fclose(fs);
  fclose(ft);
}
```

b. How is a string stored in memory? Is there any difference between string and character array? Write a C program to copy one string to another using pointers and without using library functions.

Answer:
A C string is a character sequence terminated with a null character ('\0', called NUL in ASCII). It is usually stored as one-dimensional character array.
In C these are almost the same, though a string will have an additional null character at the end

```
#include<stdio.h>
#include<conio.h>
void stcpy(char *str1, char *str2);
void main()
{
        char *str1, *str2;
        clrscr();
        printf("nnt ENTER A STRING…: ");
        gets(str1);
        stcpy(str1,str2);
        printf("nt THE COPIED STRING IS…: ");
        puts(str2);
        getch();
}
void stcpy(char *str1, char *str2)
{
        int i, len = 0;
        while(*(str1+len)!='')
        len++;
        for(i=0;i<len;i++)
        *(str2+i) = *(str1+i);
        *(str2+i) = '';
}
```

c. What is a bit field? Why are bit fields used with structures?

Answer: In addition to declarators for members of a structure or union, a structure declarator can also be a specified number of bits, called a "bit field." Its length is set off from the declarator for the field name by a colon. A bit field is interpreted as an integral type.

struct-declarator:

declarator

type-specifier declarator $_{\text{opt}}$ **:** *constant-expression*

```
struct
{
   unsigned short icon : 8;
   unsigned short color : 4;
   unsigned short underline : 1;
   unsigned short blink : 1;
} screen[25][80];
```

Q.6.a. What is a heap? Write a C program to sort an array of integers using the heap sort method. Given: 6, 5, 3, 1, 8, 7, 2, 4 are elements of an array, show the different stages of sorting.

Answer:
A heap is a specialized tree-based data structure that satisfies the *heap property:* if *B* is a child node of *A*, then key(*A*) ≥ key(*B*). This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a *max-heap*. (Alternatively, if the comparison is reversed, the smallest element is always in the root node, which results in a *min-heap*.)

```
/* array of MAXARRAY length ... */
#define MAXARRAY 5

/* preform the heapsort */
void heapsort(int ar[], int len);
/* help heapsort() to bubble down starting at pos[ition] */
void heapbubble(int pos, int ar[], int len);

int main(void) {
int array[MAXARRAY];
int i = 0;

/* load some random values into the array */
for(i = 0; i < MAXARRAY; i++)
array[i] = rand() % 100;

/* print the original array */
```

```
printf("Before heapsort: ");
for(i = 0; i < MAXARRAY; i++)
{
printf(" %d ", array[i]);
}
printf("\n");

heapsort(array, MAXARRAY);

/* print the `heapsorted' array */
printf("After heapsort: ");
for(i = 0; i < MAXARRAY; i++)
{
printf(" %d ", array[i]);
}
printf("\n");

return 0;
}

void heapbubble(int pos, int array[], int len)
{
int z = 0;
int max = 0;
int tmp = 0;
int left = 0;
int right = 0;

z = pos;
for(;;) {
left = 2 * z + 1;
right = left + 1;

if(left >= len)
return;
else if(right >= len)
max = left;
else if(array[left] > array[right])
max = left;
else
max = right;

if(array[z] > array[max])
return;
```

```
tmp = array[z];
array[z] = array[max];
array[max] = tmp;
z = max;
}
}

void heapsort(int array[], int len)
{
int i = 0;
int tmp = 0;

for(i = len / 2; i >= 0; --i)
heapbubble(i, array, len);

for(i = len - 1; i > 0; i--)
{
tmp = array[0];
array[0] = array[i];
array[i] = tmp;
heapbubble(0, array, i);
}
}
```

Let { 6, 5, 3, 1, 8, 7, 2, 4 } be the list that we want to sort from the smallest to the largest

1. Build the heap

Heap	**newly added element**	**swap elements**
nil	6	
6	5	
6, 5	3	
6, 5, 3	1	
6, 5, 3 ,1	8	
6, **5**, 3, 1, **8**		5, 8
6, **8**, 3, 1, 5		6, 8
8, 6, 3, 1, 5	7	
8, 6, **3**, 1, 5, **7**		3, 7
8, 6, 7, 1, 5, 3	2	
8, 6, 7, 1, 5, 3, 2	4	
8, 6, 7, **1**, 5, 3, 2, **4**		1, 4
8, 6, 7, 4, 5, 3, 2, 1		

Sorting.

Heap	**swap elements**	**delete element**	**sorted array**	**details**
8, 6, 7, 4, 5, 3, 2, **1**	8, 1			swap 8 and 1 in order to delete 8 from heap
1, 6, 7, 4, 5, 3, 2, **8**		8		delete 8 from heap and add to sorted array
1, 6, **7**, 4, 5, 3, 2	1, 7		8	swap 1 and 7 as they are not in order in the heap
7, 6, **1**, 4, 5, **3**, 2	1, 3		8	swap 1 and 3 as they are not in order in the heap
7, 6, 3, 4, 5, 1, **2**	7, 2		8	swap 7 and 2 in order to delete 7 from heap
2, 6, 3, 4, 5, 1, **7**		7	8	delete 7 from heap and add to sorted array
2, **6**, 3, 4, 5, 1	2, 6		7, 8	swap 2 and 6 as thay are not in order in the heap
6, **2**, 3, 4, **5**, 1	2, 5		7, 8	swap 2 and 5 as they are not in order in the heap
6, 5, 3, 4, 2, **1**	6, 1		7, 8	swap 6 and 1 in order to delete 6 from heap
1, 5, 3, 4, 2, **6**		6	7, 8	delete 6 from heap and add to sorted array
1, **5**, 3, 4, 2	1, 5		6, 7, 8	swap 1 and 5 as they are not in order in the heap
5, **1**, 3, **4**, 2	1, 4		6, 7, 8	swap 1 and 4 as they are not in order in the heap
5, 4, 3, 1, **2**	5, 2		6, 7, 8	swap 5 and 2 in order to delete 5 from heap
2, 4, 3, 1, **5**		5	6, 7, 8	delete 5 from heap and add to sorted array
2, **4**, 3, 1	2, 4		5, 6, 7, 8	swap 2 and 4 as they are not in order in the heap
4, 2, 3, **1**	4, 1		5, 6, 7, 8	swap 4 and 1 in order to delete 4 from heap
1, 2, 3, **4**		4	5, 6, 7, 8	delete 4 from heap and add to sorted array
1, 2, **3**	1, 3		4, 5, 6, 7, 8	swap 1 and 3 as they are not in

				order in the heap
3, 2, **1**	3, 1		4, 5, 6, 7, 8	swap 3 and 1 in order to delete 3 from heap
1, 2, **3**		3	4, 5, 6, 7, 8	delete 3 from heap and add to sorted array
1, **2**	1, 2		3, 4, 5, 6, 7, 8	swap 1 and 2 as they are not in order in the heap
2, **1**	2, 1		3, 4, 5, 6, 7, 8	swap 2 and 1 in order to delete 2 from heap
1, **2**		2	3, 4, 5, 6, 7, 8	delete 2 from heap and add to sorted array
1		1	2, 3, 4, 5, 6, 7, 8	delete 1 from heap and add to sorted array
			1, 2, 3, 4, 5, 6, 7, 8	completed

b. Write a C program to search for an element using binary search.

Answer:

```
#include "stdio.h"
binarysearch(int a[],int n,int low,int high)
{
        int mid;
        if (low > high)
        return -1;
        mid = (low + high)/2;
        if(n == a[mid])
        {
                printf("The element is at position %d\n",mid+1);
                return 0;
        }
        if(n < a[mid])
        {
                high = mid - 1;
                binarysearch(a,n,low,high);
        }
        if(n > a[mid])
        {
                 low = mid + 1;
                binarysearch(a,n,low,high);
        }
 }
```

```
int main()
{
        int a[50];
        int n,no,x,result;
        printf("Enter the number of terms : ");
        scanf("%d",&no);
        printf("Enter the elements :\n");

for(x=0;x<no;x++)
  {
        scanf("%d",&a[x]);
        printf("Enter the number to be searched : ");
        scanf("%d",&n);
        result = binarysearch(a,n,0,no-1);
   }
 if(result == -1)
 {
        printf("Element not found");
        return 0;
  }
```

Q.7.a. Write a C program to convert the given infix expression into its equivalent postfix form.

Answer:

```
#include<stdio.h>
#include<conio.h>
#define MAX 20

int i=0,j=0,top=-1;
char infix[MAX],suffix[MAX],stack[MAX],push(),pop();

main()
{
        clrscr();
        printf("\nEnter a valid infix expression:");
        scanf("%s",infix);

while(infix[i]!='\0')
            {
                  switch(infix[i])
                        {
                              case '(': push(infix[i]);    /* push ( on to stack */
                              break;
                              case '+': push(infix[i]); /* push the operators on to stack */
                              break;
```

```
                        case '-': push(infix[i]);
                        break;
                        case '*': push(infix[i]);
                        break;
                        case '/': push(infix[i]);
                        break;
                        case ')': while(stack[top]!='(') /* pop all elements from stack
until a ( is encountered */
                        suffix[j++]=pop();
                        pop();      /* pop the ( from stack */
                        break;
                        default:  suffix[j]=infix[i]; /* if infix[i]=operand,put it
directly in suffix[]  */
                        j++;
                    }  /* end switch */
               i++;
               }  /* end while  */

while(top!=-1)  /* when stack is not empty */
          {
               if(stack[top]=='(')  /* if stack top is ( then remove it */
               pop();
               suffix[j++]=pop();   /* pop the remaining stack elements on to suffix */
          }
     printf("\nConverted suffix expression:");
     for(i=0;suffix[i]!='\0';i++)
     printf("%c",suffix[i]);
     getch();
}

     char push(char x) /* x= pushed element */
          {                /* a= stack top      */
                    char a=stack[top];
          while((a!='(') && ((x=='+'|| x=='-')&&(a=='*'||a=='/')) || (x=='-' && a=='+'))
          {
          suffix[j++]=pop(); /*{1:The element or operator x is pushed on to        */
          a=stack[top];      /*   stack only if the stack top has a lower          */
          }       /*   precedence than the operator to be pushed.                  */
          stack[++top]=x;     /* 2:If the stack top operator has higher precedence  */
          }       /*   than the operator to be pushed then the stack    */
           /*   top is poped to suffix[].                     */
          /* 3:Now the next operator in the stack becomes the   */
          /* stack top and step 1. is repeated.
```

```
}    */
char pop()
{
 return(stack[top--]);
}
```

b. Write a C program to implement the working of a queue of integers using an array. Provide the following operations.
i) insert ii) delete iii) display

Answer:

```
#include<stdio.h>
#include<conio.h>

int cirque[10],front,rear,n;
int del();
void insert(int);
void display(int);
int empty(int,int);
char full=0;

main()
{
        char c;
        int ch,x;

        clrscr();
        printf("\nInput the size of the queue==>");
        scanf("%d",&n);
        front=rear=0;
        do
        {
                printf("Press 1 for inserting\n");
                printf("Press 2 for deleting\n");
                printf("Press 3 for displaying the queue\n");
                printf("Press 4 to exit\n");
                printf("Enter your choice==>");
                scanf("%d",&ch);

        switch(ch)
                {
                        case 1: printf("\nEnter the element to be inserted==>");
                        scanf("%d",&x);
```

```
                    insert(x);
                    break;

                    case 2: printf("\nThe element deleted is %d",del());
                    break;

                    case 3: display(front);
                    break;
               }
          }while(ch!=4);
}

void insert(int x)
{
          if(!full)  /* if queue is not full */
          {
               cirque[rear++]=x;     /* insert at the rear end */
               if(rear==n)
               rear=0;
               if(rear==front)
               {
                    printf("Queue full!\n");
                    full=1;
               }
               return;
          }
          else
          {
               printf("Queue Overflow!\n");
               return;
          }
}

void display(int front)
{
          if(front!=rear||full)
          {
               int i;
               for(i=1;i<=n;i++)
               {
                    printf("%d\n",cirque[front++]);
                    if(front==n)
                    front=0;
                    if(front==rear)
                    break;
```

```
            }
        }
    }

    int del()
    {
        int y;
        if(empty(front,rear))
        {
            printf("Queue undeflow\n");     /* if the queue is already empty */
            return(0);
        }
        y=cirque[front++]; /* delete at the front end */
        if(front==n)
        front=0;
        if(front==rear)
        {
            printf("Queue is empty!\n");
            front=rear=0;
            full=0;
        }
        return(y);  /* to display the deleted element */
    }

    int empty(int front,int rear)
    {
        if(front==rear && !full)
        return(1);
        else
        return(0);
    }
```

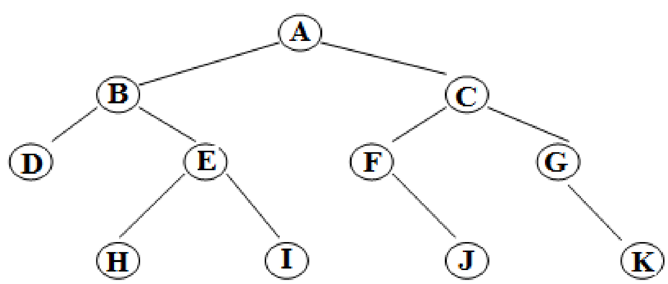
c. Write a C function to insert an element after a given node in a singly linked list.

Answer:

```
    void ins_aft(node *current)
    {
        int rno;                /* Roll number for inserting a node*/
        int flag=0;
        node *newnode;
        newnode=(node*)malloc(sizeof(node));
        printf("\nEnter the roll number after which you want to insert a node\n");
        scanf("%d",&rno);
        init(newnode);
```

```
		while(current->next!=NULL)
		{
			/***  Insertion checking for all nodes except last  ***/
			if(current->roll_no==rno)
			{
				newnode->next=current->next;
				current->next=newnode;
				flag=1;
			}
			current=current->next;
		}
		if(flag==0 && current->next==NULL && current->roll_no==rno)
		{

/***Insertion checking for last nodes  ***/
			newnode->next=current->next;
			current->next=newnode;
			flag=1;
		}
		if(flag==0 && current->next==NULL)
			printf("\nNo match found\n");
	}
```

Q.8.a. Give the order of visitation of the binary tree shown in the following figure.



i) Preorder traversal: A B D E H I C F J G K
ii) Inorder traversal : D B H E I A F J C G K
iii) Postorder traversal: D H I E B J F K G C A

b. Write an C function to insert an element into a binary search tree.

```
void insert(int val)
{
int f=0;
struct tree *n,*parent;
n=(struct tree*)malloc(sizeof(struct tree));
n->no=val;
```

```
n->l=n->r=NULL;
if (root==NULL)
{
root=n;
return;
}
parent=search(val ,&f);
if(f==1)
{
printf("\n\n\n DUPlicate number");
free(n);
return;
}
else if(val>parent->no)
parent->r=n;
else
parent->l=n;
}
```

c. Write a C function to search for an item in a binary search tree.

```
struct tree * search(int val,int *found)
{
struct tree *p=root,*par=NULL;
while(p!=NULL)
{
if(val==p->no)
{
*found=1;
break;
}
else if(val>p->no)
{
par=p;
p=p->r;
}
else
{
par=p;
p=p->l;
}
}
return par;
}
```

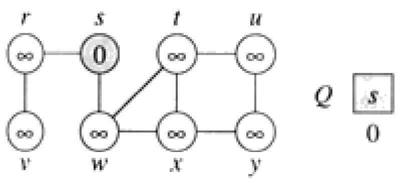
Q.9.a. Write a C program for BFS traversal. Explain the same with the help of an example.

Answer:

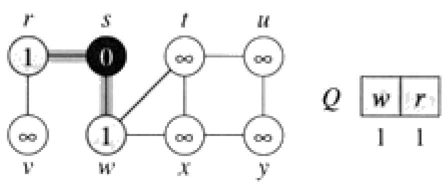
```
        #include <stdio.h>
        #define N 10
        void bfs(int adj[][N],int visited[],int start)
        {
                int q[N],rear=-1,front=-1,i;
                q[++rear]=start;
                visited[start]=1;
                while(rear != front)
                {
                                start = q[++front];
                                if(start==9)
                                        printf("10\t");
                                else
                                        printf("%c \t",start+49); //change to 65 in case of alphabets
                for(i=0;i<N;i++)
                {
                        if(adj[start][i] && !visited[i])
                        {
                                q[++rear]=i;
                                visited[i]=1;
                        }
                }
                }
        }
int main()
{
        int visited[N]={0};
        int adj[N][N]={{0,1,1,0,0,0,0,0,0,1},
        {0,0,0,0,1,0,0,0,0,1},
        {0,0,0,0,1,0,1,0,0,0},
        {1,0,1,0,0,1,1,0,0,1},
        {0,0,0,0,0,0,1,1,0,0},
        {0,0,0,1,0,0,0,1,0,0},
        {0,0,0,0,0,0,0,1,1,1},
        {0,0,1,0,0,0,0,0,0,0},
        {0,0,0,1,0,0,0,0,0,0},
        {0,0,1,0,0,0,0,1,1,0}};

        bfs(adj,visited,0);
        return 0;
}
```

Example**:** The following figure (from CLRS) illustrates the progress of breadth-first search on the undirected sample graph.

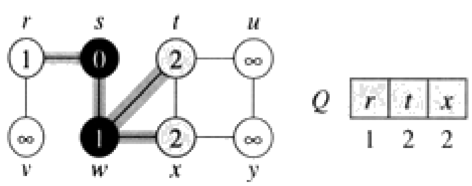
a. After initialization (paint every vertex white, set d[*u*] to infinity for each vertex *u*, and set the parent of every vertex to be NIL), the source vertex is discovered in line 5. Lines 8-9 initialize Q to contain just the source vertex *s*.



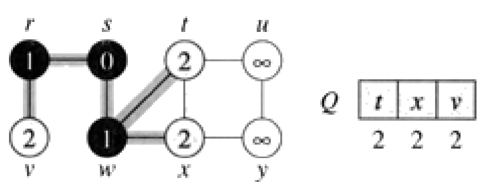
b**.** The algorithm discovers all vertices 1 edge from *s* i.e., discovered all vertices (*w* and *r*) at level 1.



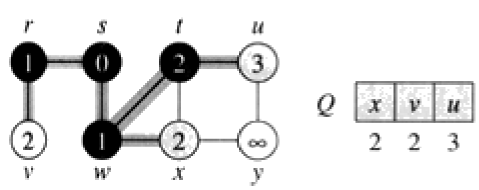
c.



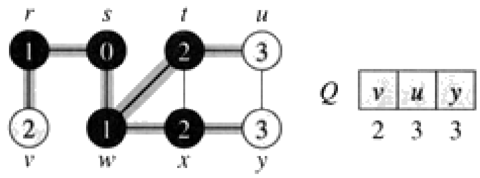
d. The algorithm discovers all vertices 2 edges from *s* i.e., discovered all vertices (*t*, *x*, and *v*) at level 2.



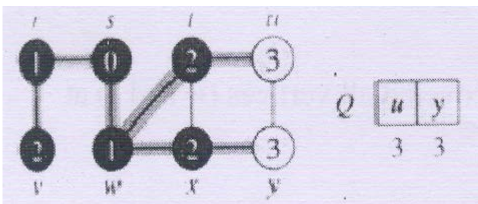
e.



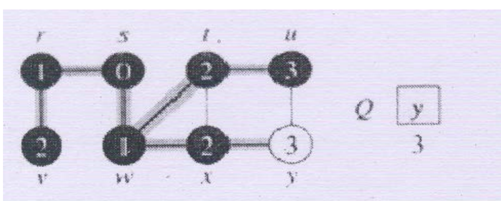
f.



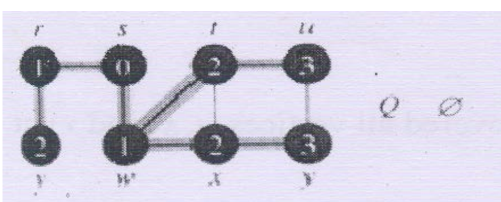
g. The algorithm discovers all vertices 3 edges from *s* i.e., discovered all vertices (*u* and *y*) at level 3.



h.



i. The algorithm terminates when every vertex has been fully explored.



b. Explain with the help of examples the following:

i. Adjacency Matrix

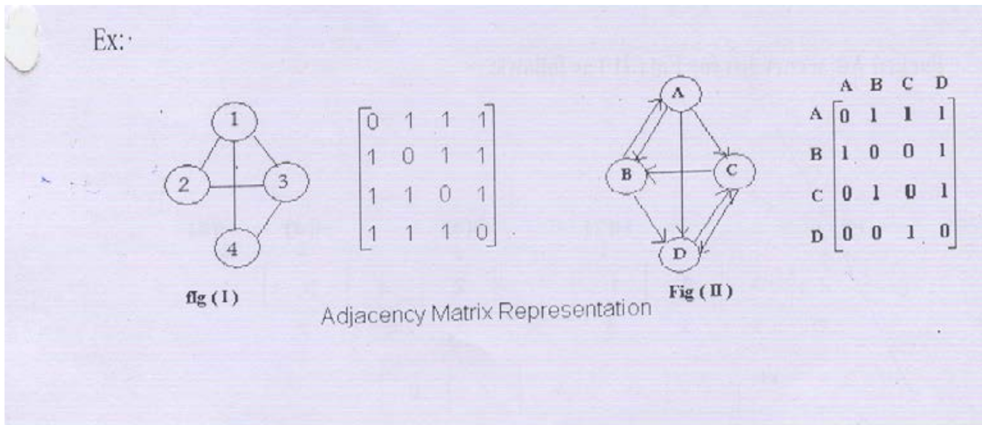
ii. Linked Adjacency Lists

Answer: The Adjacency matrix of an n-vertex graph G = (V, E) is an n*n matrix A. Each of A is either 0or 1. Let V= {1, 2…n}. If G is an undirected graph, then the elements of A are defined as follows:

A(i,j) = {1 if (i,j) belongs to E or (j,i) belongs to E
{0 otherwise

If G is an digraph, then the elements of A are defined as follows:

A(i,j) = {1 if (i,j) belongs to E
{0 otherwise

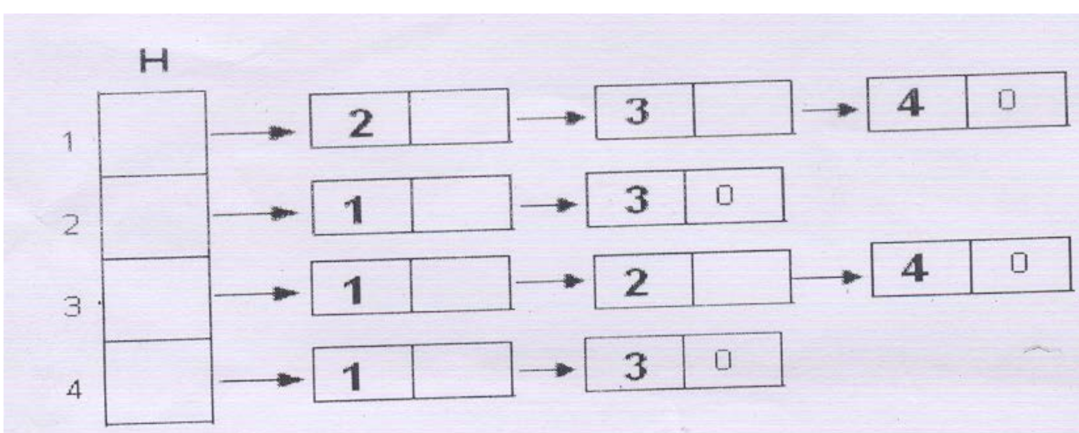


a) A(i, j) =0, 1<=i<=n for all n-vertex graph.
b) The adjacency matrix of an undirected graph is symmetric. I.e., A(i,j) = A(j,i), 1<=i<=n , 1<=j<=n.
c) For n-vertex undirected graph, A(i,j)= A(i,j) = d_i .
d) For n-vertex digraph, A(i,j)= d_i^{out} = A(i,j) = d_i^{in} , 1<=i<=n.

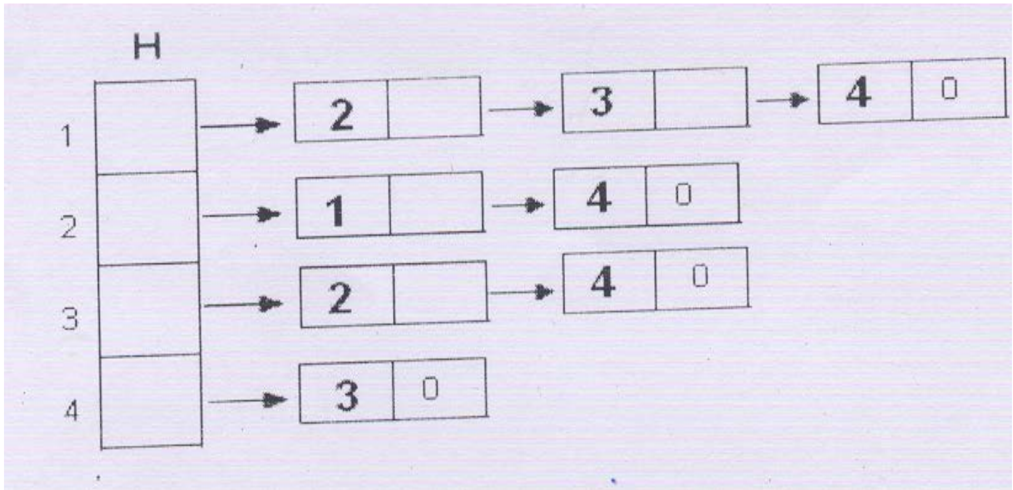
ii. Linked Adjacency Lists

Answer:
In this representation, each adjacency list is represented as a chain. An array H of head nodes of type chain keeps track of adjacency lists.

X: Linked Adjacency list for Fig (I) as follows:



Linked Adjacency list for Fig (II) as follows:



TEXTBOOK

C & Data Structures, P.S. Deshpande and O.G. Kakde, Dreamtech Press, 2005