

QNO.1 a) Explain different types of data structures.

05 mks

Ans:

Different types of data structures:

1. Stack – (01)

Linear data structure, Operating principle LIFO.

Example

2. Queue- (01)

Linear data structure, Operating principle FIFO.

Example

3. Linked list (01)

Linear data structure.

Linked list is a very common data structure used to store similar data in memory. The Elements of the linked list are not stored in continuous memory location. They are scattered all over the memory but These elements are still bounded to each other.

Example.

4. Tree— (01)

Non Linear data structure.

Tree is finite set of element that is either empty or partitioned into disjoint subsets. First subset contains a single element called root of the tree. The other subsets are themselves trees. They are called the Subtree of the root. These subtrees can be empty. Each element of a tree is called node of the tree.

Example.

5. Graph – (01)

Non Linear data structure.

DATA STRUCTURES AND FILES

It is collection of vertices and edges. $G=\{V, E\}$. No root in graph.

Example.

QNO.1 b. Write recursive java method that finds min and max values in an array of int values

Without using loops.

05 mks

Ans:

```
import java.io.*;
class fminmax
{
    static int max,min;
    static int a[]=new int[10];
    public static void main(String[] args) throws IOException
    {
        DataInputStream in=new DataInputStream(System.in);
        System.out.println("input n");
        Int n=Integer.parseInt(in.readLine());
        System.out.println("input element");
        for(int i=0;i<n;i++)
            a[i]=Integer.parseInt(in.readLine());
        min=a[0];
        minmax(n-1);
        System.out.println("max= "+max);
    }
}
```

DATA STRUCTURES AND FILES

```
System.out.println("min= "+min);  
  
}
```

```
static void minmax(int n)
```

```
{
```

```
if (n==1)
```

```
{
```

```
return;
```

```
}
```

```
if(min>=a[n])
```

```
{
```

```
min=a[n];
```

```
minmax(n-1);
```

```
}
```

```
else
```

```
minmax(n-1);
```

```
if(max<a[n])
```

```
{
```

```
max=a[n];
```

```
minmax(n-1);
```

```
}
```

```
else
```

```
minmax(n-1);
```

}

}

QNO.1 c. Define

Ans:

i. Abstract data type

ADT = type + function names + Behavior of each function

ii. Binary tree

An important structure which can be built dynamically is the binary tree. Conceptually a binary tree is a structure where each node may contain some information, and may have two child nodes, called left and right for convenience. These child nodes themselves may also each have two child nodes, and so on. A node may also have only a left child or only a right child, or no child nodes at all. A node with no child node is called a "leaf" node. All other nodes are called "interior" nodes.

Example

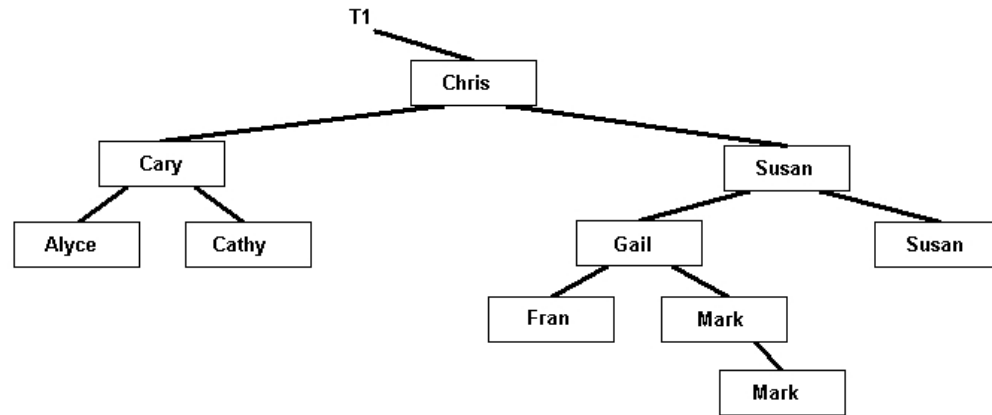
ii. Binary tree

Ans:

An important structure which can be built dynamically is the binary tree. Conceptually a binary tree is a structure where each node may contain some information, and may have two child nodes, called left and right for convenience. These child nodes themselves may also each have two child nodes, and so on. A node may also have only a left child or only a right child, or no child nodes at all. A node with no child node is called a "leaf" node. All other nodes are called "interior" nodes.

Here are some examples:

DATA STRUCTURES AND FILES



iii. Graph

05 mks

Non Linear data structure.

It is collection of vertices and edges. $G = \{V, E\}$. No root in graph.

Example.

QNO.1 d. Priority queue—

05 mks

Ans:

Operating principle: we insert element in random order but removed element which has highest priority.

Two types of priority queues:

1. Ascending priority queue: smallest element in queue removed first.
2. Descending priority queue: largest element in queue removed first.

Example

(2)

DATA STRUCTURES AND FILES

QNO 2. a. Write a program in java to delete a node from given binary search tree. Consider all

Cases.

10 mks

Ans:

Function to delete a node from given binary search tree.

(4+3+3)

Void Del (node temp, int key) throws NullPointerException

```
{
node temp_succ;
sop(" the node to be delete:"+temp.data);
parent=get_parent(root, temp);
sop("the parent node"+parent.data);

// CASE 1: deleting node with two children

If(temp.left!=null&& temp.right!=null)
{
Parent=temp;
Temp_succ=temp_succ.left;
While (temp_succ.left!=null)
{
Parent=temp_succ;
Temp_succ=temp_succ.left;
}
Temp.data=temp_succ.data;
```

DATA STRUCTURES AND FILES

```
Temp_succ=null;

Parent.left=null;

Sop("now deleted");

Return;

}

// CASE 2: deleting a node having only one child

// the node to be deleted has left child

If(temp.left!=NULL&&temp.right!=null)

{

If(parent.left==temp)

Parent.left==temp.left;

Else

Parent.right=temp.left;

Temp=null;

Sop("now deleted");

Return;

}

//deleting a node having only one child

// the node to be deleted has right child

If(temp.left!=NULL&&temp.right!=null)

{

If(parent.left==temp)

Parent.left==temp.right;
```

DATA STRUCTURES AND FILES

```
Else

Parent.right=temp.right;

Temp=null;

Sop("now deleted");

Return;

}

// CASE 3: deleting a node having no child

If(temp.left!=NULL&&temp.right!=null)

{

If(parent.left==temp)

Parent.left==null;

Else

Parent.right=null;

Sop("now deleted");

Return;

}

}
```

DATA STRUCTURES AND FILES

QNO.2 b. Write a program for insertion sort. Sort the following using insertion sort

10,3,8,4,2

10 mks

Ans:

Program for insertion sort

(05+05)

```
import java.io.*;
```

```
class InsertionSort
```

```
{
```

```
    static int a[]=new int[50];
```

```
    public static void main(String args[])throws IOException
```

```
    {
```

```
        int i,n;
```

```
        DataInputStream in=new DataInputStream(System.in);
```

```
        System.out.println("ENTER NUMBER OF OF ELEMENT WANT TO SORT");
```

```
        n=Integer.parseInt(in.readLine());
```

```
        System.out.println("ENTER ELEMENT");
```

```
        for(i=0;i<n;i++)
```

```
            a[i]=Integer.parseInt(in.readLine());
```

```
        insertion(n);
```

```
        System.out.println("ARRAY ELEMENT IN ASCENDING ORDER");
```

```
        for(i=0;i<n;i++)
```

```
            System.out.println(a[i]);
```

```
    }
```

DATA STRUCTURES AND FILES

```
static void insertion(int n)
{
    int k,i,y;
    for (k=1;k<n ;k++ )
    {
        y=a[k];
        for (i=k-1;i>=0&& y<a[i] ;i-- )
        {
            a[i+1]=a[i];
        }
        a[i+1]=y;
        System.out.println("Elements in Array after pass "+k);
        for(int j=1;j<=n;j++)
            System.out.println(a[j]);
    }
}
```

Sorting following elements using insertion sort:

10, 3, 8,4, 2

Output after each pass:

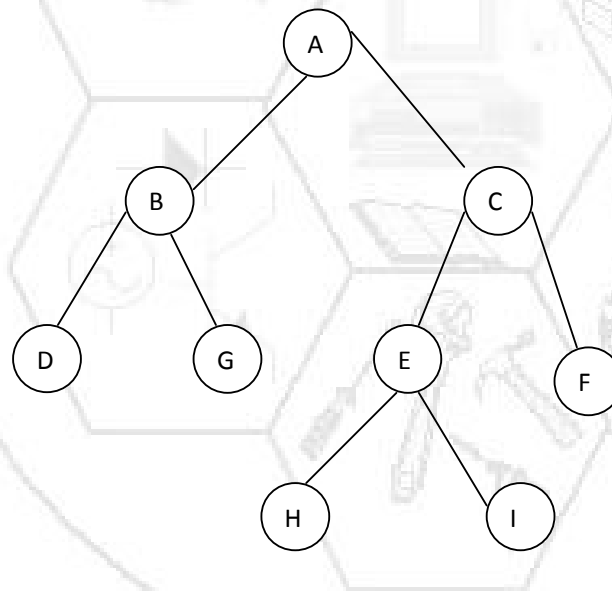
DATA STRUCTURES AND FILES

Pass i \	0	1	2	3	4
1	3	10	8	4	2
2	3	8	10	4	2
3	3	4	8	10	2
4	2	3	4	8	10

QNO 3 a. Construct of binary tree for preorder and inorder sequence given below.

Preorder: ABDGCEHIF

Inorder : DGBAHEICF



DATA STRUCTURES AND FILES

QNO. 3 b. Write program to read text and count all occurrences of particular word

10 mks

Ans:

```
import java.io.*;
public class cnt2
{
public static void main(String args[])throws IOException
{
int times=0,count=0,x=0,no=0;
InputStreamReader ir=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(ir);
String s,w;
System.out.println("Enter the sentence:");
s=br.readLine();
System.out.println("Enter the word:");
w=br.readLine();
try{
for(int i=0;i<s.length();i++)
{
if(w.charAt(0)==s.charAt(i))
{
for(int j=0;j<w.length();j++,i++)
{
if(s.charAt(i)==w.charAt(j))
{
count=count+1;
}
}
}
}
if(count==w.length())
{no=no+1;count=0;};
}
}
```

```
}  
catch(Exception e){ }  
if(no==0)  
{  
System.out.println("word is not present");  
}  
else  
{  
System.out.println("word is present "+no+" times");  
}  
}  
}
```

QNO 4 a) Write a program for circular linked list.

10 mks

Ans:

```
import java.io.*;  
public class CircularLinkedList<E> {  
    private Entry<E> head;  
  
    // Last element of the list. tail.next = head  
    private Entry<E> tail;  
  
    private int size = 0;  
  
    /**  
     * Class to hold the individual elements.  
     * @param <E>  
     */  
    private static class Entry<E> {  
        E element;  
  
        Entry<E> next;
```

DATA STRUCTURES AND FILES

```
Entry(E element, Entry<E> next) {
    this.element = element;
    this.next = next;
}

Entry(E element) {
    this.element = element;
}

public CircularLinkedList() {
    head = null;
}

/**
 * Remove obj from the circular linked list and return the removed object
 * @param obj
 * @return
 */
public E remove(E obj) {
    if (head == null || tail == null)
        throw new NoSuchElementException();
    Entry<E> current = head, temp = head, found = null;
    if (obj.equals(head.element)) {
        if (head.next == head) {
            found = head;
            head = null;
            tail = null;
            size--;
            return found.element;
        } else {
            found = head;
            temp = tail;
        }
    }
}
```

DATA STRUCTURES AND FILES

```
} else {
    current = head.next;
    while (current != head) {
        if (current.element.equals(obj)) {
            found = current;
            break;
        }
        temp = current;
        current = current.next;
    }
}
if (found == null) {
    throw new NoSuchElementException(obj.toString());
}
E result = found.element;
temp.next = found.next;
found.next = null;
found.element = null;
size--;
return result;
}

/**
 * Add obj to the circular linked list.
 * @param obj
 */
public void add(E obj) {
    Entry e = new Entry(obj);
    if (head == null) {
        size++;
        head = e;
        head.next = head;
        tail = head;
    }
    return;
```

DATA STRUCTURES AND FILES

```
}  
size++;  
e.next = head;  
head = e;  
tail.next = head;  
}  
  
/**  
 * Size of the list.  
 * @return  
 */  
public int size() {  
    return size;  
}  
  
public static void main(String[] args) {  
    CircularLinkedList<String> list = new CircularLinkedList<String>();  
    list.add("One");  
    list.add("Two");  
    list.add("Three");  
    list.add("Four");  
  
    System.out.println(list.remove("Three"));  
    System.out.println(list.remove("Two"));  
    System.out.println(list.remove("One"));  
    System.out.println(list.remove("Four"));  
    System.out.println(list.remove("Four"));  
  
}  
}
```

DATA STRUCTURES AND FILES

QNO 4 b. Hash the following in a table of size 11. use any two collision resolution techniques.

23,0,52,61,78,33,100,8,10,90,14

10 mks

Ans:

Using division method for hashing

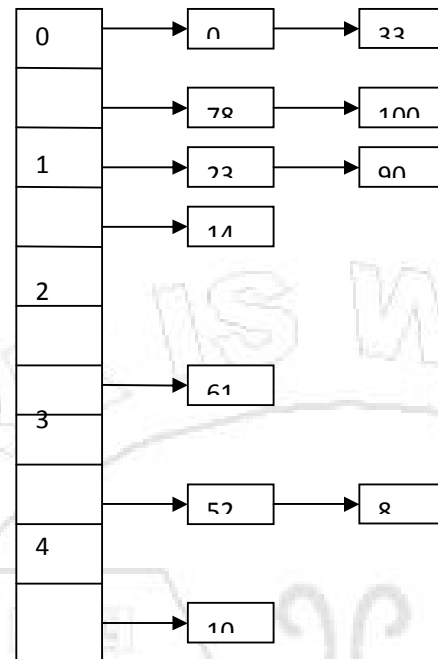
(02+04+04)

Linear probing method of collision handling

Index	Data	
0	0	Clash for 33
1	23	Clash for 78 and 100
2	78	Clash for 90
3	33	Clash for 14
4	100	
5	90	
6	61	
7	14	
8	52	Clash for 8
9	8	
10	10	

Chaining

DATA STRUCTURES AND FILES



QNO 5 a. Write a program in java to create a doubly linked list and perform following

Operations. i) insert into list ii) search for data iii) delete from list iv) display

Ans:

(03+03+03+03)

// function for insertion node at last

```
public void insert_last(node head, int val) throws IOException
```

```
{
```

```
node temp;
```

```
node new=new node();
```

```
new.data=val;
```

```
new.next=null;
```

```
new.prev=null;
```

DATA STRUCTURES AND FILES

```
if(head==null)

head=new;

else

{

temp=head;

while(temp.next!=null)

temp=temp.next;

temp.next=new;

new.prev=temp;

new.next=null;

sop("element is inserted");

}

// function for searching

node search(node head,int key)

{

node temp;

int found=0;

temp=head;

if(temp==null)

{

sop("empty");
```

```
return null;

}

while(temp!=null && found==0)

{

if(temp.data!=key)

temp=temp.next;

else

found=1;

}

if(found==1)

{

sop("element present in list");

return temp;

}

else

return null;

}

}
```

```
// function for deletion
```

```
public node delete(node head,int key) throws NullPointerException
```

```
{
```

```
node temp, temp_prev;
```

DATA STRUCTURES AND FILES

```
temp=head;

if(temp==null)

{

sop("empty");

return null;

}

temp=search(head,key);

if(temp!=null)

{

temp_prev=temp_prev;

if(temp.next==null)

{

temp_prev.next=null;

temp=null;

}

else

{

temp_prev.next=temp.next;

(temp.next).prev=temp_prev;

temp.next=null;

temp=null;

}

}
```

DATA STRUCTURES AND FILES

```
else

{

if(temp.next==null&& temp.prev==null)

return null;

head=temp.next;

head.prev=null;

temp=null;

}

sop("deleted");

}

return head;

}

// function for display

Public void display(node head)

{

Node temp;

Temp=head;

If(temp==null)

{

Sop("empty");

}

Sop("linked list as:");

While(temp!=null)
```

```

{
    Sop(" "+temp.data);
    Temp=temp.next;
}
}

```

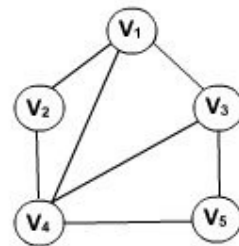
QNO. 5 b) what are different methods to represent in graph in memory? What are applications of graph.

Ans:

- i. Graph representation.
- ii. Adjacency matrix representation of graph

An adjacency matrix of a graph $G=(V,E)$ (let $V = \{ v_1, v_2, \dots, v_n \}$) is a $n \times n$ matrix A , such that $A[i, j] = 1$ 0 otherwise

if there is edge between v_i and v_j .

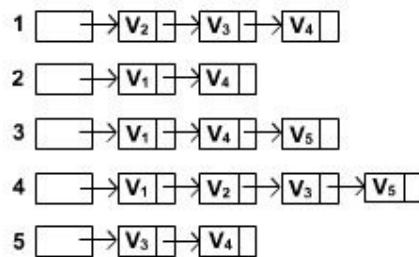
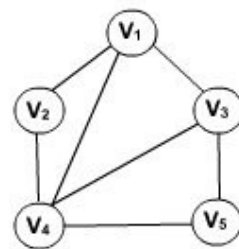


DATA STRUCTURES AND FILES

	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	0
3	1	0	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0

ii. Adjacency list representation of graph

It consists of a list of vertices, which can be represented either by linked list or array. For each vertex, adjacent vertices are represented in the form of a linked list.



Applications of graph:

- i. in computer networking
- ii. telephone cabling graph
- iii. job scheduling algorithm

iv. routing

QNO. 6 a. Write program to implement conversion of given number to its equivalent binary form

Using stack.

10 mks

Ans:

(4+2+2+2)

```
While( num>=1)
{
Item=num%2;
Push(item);
Num=num/2;
}
Sop("equivalent binary no is: "+obj.Pop());
```

Need to be define

Push, pop and display function

QNO. 6 b. Write a programs for reading data from file.

10 mks

// correct program carries 10 mks

Ans:

```
Class readfile
```

```
{
```

```
    Psvm(string arg[])
```

```
{
```

DATA STRUCTURES AND FILES

```
FileInputStream fp;  
  
DataInputStream in;  
  
If(args.length!=0)  
{  
    Try  
    {  
        fp=new FileInputStream(args[0]);  
        in = new DataInputStream(fp);  
  
        while(in.available()!=0)  
        {  
            Sop(in.readLine());  
        }  
  
        In.close();  
    }  
  
    Catch(Exception e)  
    {  
        Sop("error");  
    }  
}  
  
Else  
  
    Sop("input file not mentioned");  
}
```

Input

C:\Java readfile .read.txt

QNO. 7. Write notes on--

a. AVL tree

(01+02+02)

An AVL tree is another balanced binary search tree. Named after their inventors, Adelson-Velskii and Landis, they were the first dynamically balanced trees to be proposed. Like red-black trees, they are not perfectly balanced, but pairs of sub-trees differ in height by at most 1, maintaining an $O(\log n)$ search time. Addition and deletion operations also take $O(\log n)$ time.

Definition of an AVL tree

An AVL tree is a binary search tree which has the following properties:

The sub-trees of every node differ in height by at most one.

Every sub-tree is an AVL tree.



Following rotation applied to balance AVL tree:

LL rotation

RR rotation

RL rotation

DATA STRUCTURES AND FILES

LR rotation

Example:

QNO.7 b. **Array Representation of graph**

(02+02+01)

List can be represented by using arrays. Basically list is collection of elements. To show the list using arrays we will have data and next fields in the array. The array can be created as:

class node

```
{public int data;
```

```
public int next;
```

```
}
```

Now array for this class can be created as:

```
node []a=new node[10];
```

We will insert some value in this node as:

```
a[0].data=10;
```

```
a[0]=-1;
```

Thus implementation of list using array as

index	Data	Next
A[0]	20	4
	40	3
A[1]	10	0

DATA STRUCTURES AND FILES

A[2]	50	-1
A[3]	30	1
A[4]		

Various operations that can be performed on the list using array.

- i. creation list
- ii. Display list
- iii. Insertion of any element in the list
- iv. Deletion of any element from the list.
- v. Searching

QNO.7 c. Binary Search

(01+02+02)

Definition: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Algorithm:

A static int BinarySearch(int n,int key)

DATA STRUCTURES AND FILES

```

{
    int low=0;

    int high=n-1;

    int mid;

    while(low<high)
    {
        mid=(low+high)/2;
        if(key==x[mid])
            return mid;
        else if(key<x[mid])
            high=mid-1;
        else
            low=mid+1;
    }
    return -1;
}

```

Example:

QNO 7 d. **Graph traversal algorithm**

(01+02+02)

- i. Breadth first search
- ii. Depth first search

i. Breadth First Search

DATA STRUCTURES AND FILES

BFS is graph traversal method in this method we can start searching from any vertex. Vertex v_1 in the graph will be visited first then all the vertices adjacent to v_1 will be travelled suppose adjacent to v_1 are so $v_2, v_3 \dots v_n$ will be printed first then again from v_2 the adjacent vertices will be printed this processes will be continue for all the vertices to get encountered to keep track of all the vertices and their adjacent vertices.

Algorithm-

Step 1: Start with any node and mark it as visited.

Step 2: find the adjacent nodes to the node mark in step1 and them in Queue

Step 3: Visit the node which is at the front of Queue delete this node from Queue.

Step 4: Repeat step3 still the Queue is not empty.

Step 5: Stop.

Example:

ii. Depth first search

DFS, the next vertex to be visited will be the adjacent vertex (to the starting point), which has the highest depth value and then the next adjacent vertex with the next higher depth value and so on till all the adjacent nodes for that vertex are not visited. We repeat this same procedure for every visited vertex of the graph.

Algorithm:-

Step-1: Set the starting point of traversal and push it inside the stack

Step-2: Pop the stack and add the popped vertex to the list of visited vertices

Step-3: Find the adjacent vertices for the most recently visited vertex(from the Adjacency Matrix)

Step-4: Push these adjacent vertices inside the stack (in the increasing order of their depth) if they are not visited and not there in the stack already

Step-5: Go to step-2 if the stack is not empty

Example:

