19AIE111 Data Structures & Algorithms 1 (1 2 0 3)

S2 B.Tech CSE(AI)

Project Report

Submitted by
Group 09,
N.SAI BHARATH, RAJA PAVAN KARTHIK
AM.EN.U4AIE20052,AM.EN.U4AIE20060

June 2021
Department of Computer science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri Campus
Kollam 690525
Kerala

Amrita Vishwa Vidyapeetham
Amritapuri Campus
Kollam 690525
Kerala

Department of Computer Science and Engineering

Certified that this is the bonafide project report for the course 19AIE111 Data Structures & Algorithms 1 submitted on its completion by **Group 09,N.Sai Bharath ,RAJA PAVAN KARTHIK AM.EN.U4AIE20052,AM.EN.U4AIE20060 ,**second semester students of B.Tech CSE(AI).

Abstract

In computer science, a **stack** is an **abstract data type** that serves as a collection of elements, with two main principal operations: Push, which adds an element to the collection, and. Pop, which removes the most recently added element that was not yet removed.For the search If the tree is empty, we have a search miss; if the search key is equal to the key at the root, we have a search hit. Otherwise, we search (recursively) in the appropriate subtree. The recursive get method implements this algorithm directly. It takes a node (root of a subtree) as first argument and a key as second argument, starting with the root of the tree and the search key. Insert is not much more difficult to implement than search. a search for a key not in the tree ends at a null link, and all that we need to do is replace that link with a new node containing the key. The recursive put method accomplishes this task using logic similar to that we used for the recursive search: If the tree is empty, we return a new node containing the key and value; if the search key is less than the key at the root, we set the left link to the result of inserting the key into the left subtree; otherwise, we set the right link to the result of inserting the key into the right subtree.

Contents Page No

1.Introduction

An **Array** is a collection of similar data types. Array is a container object that holds values of homogeneous type. It is also known as static data structure because size of an array must be specified at the time of its declaration. Array starts from zero index and goes to n-1 where n is the length of the array.

1. Arrays are used to implement stack and queues.
2. Trees, in turn, are used to implement various other types of data structures.
3. Data structures like a heap, map, and set use binary search trees and balanced binary trees which can be implemented using arrays.
4. Arrays are used to maintain multiple variables with the same name.
5. CPU scheduling algorithms are implemented using arrays.
6. All sorting algorithms use arrays at its core.

In Java, an array is treated as an object and stored into heap memory. It allows storing primitive values or reference values. Arrays can be single dimensional or multidimensional in Java.

Stack is an abstract data type with a bounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack and the only element that can be removed is the element that is at the top of the stack, just like a pile of objects.

1. Stack is an ordered list of similar data types.
2. Stack is a LIFO(Last in First out) structure.
3. **push()** function is used to insert new elements into the Stack
4. **pop()** function is used to remove an element from the stack.
5. Both insertion and removal are allowed at only one end of Stack called Top.
6. Stack is said to be in Overflow state when it is completely full and is said to be in underflow state if it is completely empty.

The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack. There are other uses also like Parsing, Expression Conversion(Infix to Postfix, Postfix to Prefix etc)

Queue is also an abstract data type or a linear data structure, just like stack data structure, in which the first element is inserted from one end called the REAR(also called tail), and the removal of the existing element takes place from the other end called as FRONT(also called head). This makes the queue a **FIFO(First in First Out)** data structure, which means that element inserted first will be removed first.

Data inserted first, will leave the queue first.The process to add an element into the queue is called Enqueue and the process of removal of an element from the queue is called Dequeue.

1. Like stack, queue is also an ordered list of elements of similar data types.
2. Queue is a **FIFO(First in First Out)** structure.
3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element.
4. **peek( )** function is often used to return the value of the first element without dequeuing it.

Serving requests on a single shared resource, like a printer, CPU task scheduling etc. In real life scenarios, Call Center phone systems use Queues to hold people calling them in an order, until a service representative is free. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.

A **binary tree** is a hierarchical data structure in which each node has at most two children generally referred to as left child and right child. The topmost node in the tree is called the root. An empty tree is represented by a NULL pointer. Each node contains three components:

1. Pointer to left subtree
2. Pointer to right subtree
3. Data element

Binary Tree: Common Terminologies

- Root: Topmost node in a tree.
- Parent: Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent.
- Child: A node directly connected to another node when moving away from the root.
- Leaf/External node: Node with no child.
- Internal node: Node with at least one child.
- Depth of a node: Number of edges from root to the node.
- Height of a node: Number of edges from the node to the deepest leaf. Height of the tree is the height of the root.

Trees are so useful and frequently used, because they have some very serious advantages, Trees reflect structural relationships in the data. Trees are used to represent hierarchies.Trees provide an efficient insertion and searching.Trees are very flexible data, allowing to move subtrees around with minimum effort.

Linked List is a very commonly used linear data structure which consists of a group of nodes in a sequence. Each node holds its own data and the address of the next node hence forming a chain-like structure. Linked Lists are used to create trees and graphs. They are dynamic in nature which allocates the memory when required. Insertion and deletion operations can be easily implemented. Stacks and queues can be easily executed. Linked List reduces the access time.

- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don't need to know the size in advance.

There are 3 different implementations of Linked List available, they are:

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

2. Software and Hardware Requirements:

Hardware Requirement for Java

- Minimum Windows 95 software
- IBM-compatible 486 system
- Hard Drive and Minimum of 8 MB memory
- A CD-ROM drive
- Mouse, keyboard and sound card, if required

Software requirement for Java

- Operating System
- Java SDK or JRE 1.6 or higher
- Java Servlet Container (Free Servlet Container available)
- Supported Database and library that supports the database connection with Java.

3. Problem Statement

Project Phase 1 – LISTS, STACKS.

Main Question 1

STACK

a. Implement a stack using an array.

b. Implement a stack using a linked list.

Advanced Question 1

Flattening a Linked List

You are given a pointer p to node, having a key field, and two pointers to similar nodes. The node p is the first node in the “main” list. The two pointers associated with each node are,

a. Next pointer to the next node in the linked list.

b. Down pointer which points to another linked list.

The node p starts up a “main” list, in which each node is connected to the next one through the next pointers. Each node in this main list also starts up through its “down” pointers, a “down” list of nodes, each connected to the next through down pointers. For the nodes in the down list (other than the first node) the next pointers are necessarily nil. Each node belongs to only one down list.

Advanced Question 2

Flatten a multilevel linked list

Given a linked list where in addition to the next pointer, each node has a child pointer, which may or may not point to a separate list. These child lists may have one or more children of their own, and so on, to produce a multilevel data structure, as shown in below figure. You are given the head of the first level of the list. Flatten the list so that all the nodes appear in a single-level linked list. You need to flatten the list in a way that all nodes at first level should come first, then nodes of second level, and so on.

A stack can be implemented using array as follows

Before implementing actual operations, first follow the below steps to create an empty stack.

Include all the header files which are used in the program and define a constant 'SIZE' with specific value.Step 2 - Declare all the functions used in stack implementation.Step 3 - Create a one dimensional array with fixed size (int stack[SIZE]) Step 4 - Define a integer variable 'top' and initialize with '-1'. (int top = -1)Step 5 - In the main method, display a menu with a list of operations and make suitable function calls to perform operations selected by the user on the stack.

To implement a stack using a linked list, we need to set the following things before implementing actual operations. Step 1 - Include all the **header files** which are used in the program. And declare all the **user defined functions**. Step 2 - Define a '**Node**' structure with two members **data** and **next**. Step 3 - Define a **Node** pointer '**top**' and set it to **NULL**. Step 4 - Implement the **main** method by displaying a Menu with a list of operations and make suitable function calls in the **main** method.

Flatten a multi-level linked list

If the node is NULL, return NULL.Store the next node of the current node (used in step 4).

Recursively flatten down the list. While flattening, keep track of the last visited node, so that the next list can be linked after it.Recursively flatten the next list (we get the next list from the pointer stored in step 2) and attach it after the last visited node.

Project Phase 2 – Queue, Phone Directory using BST ADT

Main Question 1

QUEUE

1. Implement a queue using an array.
2. Implement a queue using a linked list.

Advanced Question 2

Implement a Phone Directory

Develop a phone directory which loads and stores contact information which are name, phone number and email from an input text.file. So an **Insert function** should be there to insert the above details into a BST.

Your java program should function like a phone book, loading information from an input file into a Binary search tree(call insert() method of BST) and performing the following operations on said information.

ALGORITHM FOR QUEUE USING ARRAY

Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value. Declare all the **user defined functions** which are used in queue implementation. Create a one dimensional array with the above defined SIZE (**int queue[SIZE]**) Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**) Then implement the main method by displaying a menu of operations list and make suitable function calls to perform operations selected by the user on queue.

ALGORITHM FOR QUEUE USING LINKED LIST

In Queue insertion and deletion should happen at different ends. Include all the **header files** which are used in the program. And declare all the **user defined functions**.Step 2 - Define a '**Node**' structure with two members **data** and **next**.Step 3 - Define two **Node** pointers '**front**' and '**rear**' and set both to **NULL**.Step 4 - Implement the **main** method by displaying Menu of list of operations and make suitable function calls in the **main** method to perform user selected operation.

4. Modules (minimum 5 pages)

Explain how you implemented the algorithm. Explanations of main functionalities are enough. Explain about the input format used in each question Use Algorithms, Flowchart of your logic if any.

ALGORITHMS

PUSH

```
If Top=Max-1
Print “Overflow : Stack is full” and Exit
End If
Top=Top+1
Stack[TOP]=Element
End
```

POP

```
If TOP=-1
Print “Underflow: Stack is empty” and Exit
End if
Set Del_element=Stack[Top]
Top=Top-1   --(Del_Element  )
End
```

PEEK

```
If TOP=-1
Print “Underflow: Stack is empty” and Exit
End if Start Else
Return to top
```

end

PRINT

```
If TOP=-1

Print "Underflow: Stack is empty" and Exit

End else

for(the values in array)

Print the values

End
```

ENQUEUE

```
Algorithm enqueue(item){
    If(rear==n-1){
        print"Queue is full "
    }
    else if(IsEmpty()){
        rear=0;front=0;
        Queue[rear]=item;
    }
    else{
        rear=rear+1;
        Queue[rear]=item;
    }
}
```

DEQUEUE

```
Algorithm dequeue(){
    If(IsEmpty()){
        print"Deletion is not possible"
        else if(front==rear){
            front=-1;rear=-1;
        }
    }
    else{
        front=front+1;
    }
}
```

ISEMPTY

```
Algorithm IsEmpty(){
Boolean IsEmpty(){
        If(front==-1&&rear==-1)
        return true;
        else
            return false;
 }
}
```

PEEK/GETFRONT

```
Algorithm for getFront(){
        Algorithm getFront(){
            if(IsEmpty())
                return -1;
            else
                return Queue[front];
 }}
```

QUEUE USING ARRAY

ENQUEUE

Check whether the queue is FULL. (rear == SIZE-1)

If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.

If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value.

DEQUEUE

Check whether the queue is EMPTY. (front == rear)

If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.

If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as a deleted element. Then check whether both front and rear are equal (front == rear), if it is TRUE, then set both front and rear to '-1' (front = rear = -1).

SHOW/DISPLAY

Check whether the queue is EMPTY. (front == rear)

If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.

If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'.

Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value reaches to rear (i <= rear)

QUEUE USING LINKED LIST

enqueue()

Create a newNode with given value and set 'newNode → next' to NULL.Step 2 - Check whether queue is Empty (rear == NULL) Step 3 - If it is Empty then, set front = newNode and rear = newNode. Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode.

dequeue()

Check whether the queue is Empty (front == NULL). Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function. Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'. Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

peek()

The peek() method of Queue Interface returns the element at the front of the container. It does not delete the element in the container. This method returns the head of the queue. The method does not throw an exception when the Queue is empty, it returns null instead. Returns: This method returns the head of the Queue, it returns false when the Queue is empty.

DISPLAY/SHOW

Check whether the queue is Empty (front == NULL). Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function. Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front. Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL). Step 5 - Finally! Display 'temp → data ---> NULL'.

INSERTION

```
1. Create a new BST node and assign values to it.
2. insert(node, key)
        i) If root == NULL,
        return the new node to the calling function.
        ii) if root=>data < key
            call the insert function with root=>right and assign the return value in root=>right.
            root->right = insert(root=>right,key)
        iii) if root=>data > key
        call the insert function with root->left and assign the return value in root=>left.
        root=>left = insert(root=>left,key)
3. Finally, return the original root pointer to the calling function.
```

DELETION

If the node is leaf (both left and right will be NULL), remove the node directly and free its memory.

If the node has only the right child (left will be NULL), make the node point to the right node and free the node.

If the node has only the left child (right will be NULL), make the node point to the left node and free the node.

If the node has both left and right child,

1.find the smallest node in the right subtree. say min

2.make node->data = min

3.Again delete the min node.

SEARCH

check, whether the value in the current node and a new value are equal. If so, a duplicate is found. Otherwise, if a new value is less than the node's value: if a current node has no left child, a place for insertion has been found;otherwise, handle the left child with the same algorithm. if a new value is greater than the node's value: if a current node has no right child, place for insertion has been found; Otherwise, handle the right child with the same algorithm.

5. Implementation

Phase 1

PUSH

```
void push(int item)
                              {
                                        if(top>=arr.length-1)
                                        {
                                                  System.out.println("Stack  array  OVERFLOW");
                                        }
                                        else
                                        {
                                                  top=top+1;
                                                  arr[top]=item;
                                        }
                              }
```

Pop

```
void pop()
                              {
                                        if(IsEmpty())
                                                  System.out.println("Stack  array is EMPTY");
                                        else
                                        {
                                                  System.out.println(arr[top]);
                                                  top=top-1;
                                        }
                              }
```

PEEK

```
int peek()
                              {
                                        if(IsEmpty())
                                        {
                                                  System.out.println("Stack  array is EMPTY");
                                                  return -1;
                                        }
                                        else {
                                                  System.out.println(arr[top]);
                                                  return arr[top];
                                        }

                              }
```

PRINT

```
void print()
```

```
                        {
                                if (top==-1)
                                {
                                        System.out.print("Stack  array is EMPTY");
                                }
                                else {
                                for(int i=0;i<=top;i++)
                                {
                                        System.out.print(arr[i]+" ");
                                }
                                }
                        }
```

ENQUEUE

```
public void enqueue(int data){
        Node toAdd = new Node(data);
        if (head == null) {
                head = toAdd;
                   return;
                }
                Node temp = head;
                while (temp.next != null){
                        temp = temp.next;
                }
                temp.next = toAdd;
  }
```

DEQUEUE

```
public int dequeue(){
       if (head == null){
          return -1;
        }
        Node temp = head;
        head = temp.next;
        return temp.element;
}
```

PEEK

```
public int peek(){
        if (head == null){
```

```
                    return -1;
          }
   return head.element;
}
```

SHOW/DISPLAY

```
public void show(){
          if (head == null){
                    return;
   }
          Node temp = head;
          while (temp != null){
                    System.out.print(temp.element + " ");
      temp = temp.next;
   }
   System.out.println("");
}
```

SEARCH

```
public void Search(String name,BSTNode p){
          if (p != null){
                    Search(name,p.left);
                    if(p.name==name)
                    {
                              return ;
                    }
                    Search(name,p.right);
          }
}
```

INSERT

```
public void insert(String name, String email, int phone){
                    BSTNode p = root, prev = null;
                    while (p != null){
                              prev = p;
                              if (p.name.compareTo(name) < 0)
                                        p = p.right;
```

```
                        else p = p.left;
                }
                if (root == null)
                        root = new BSTNode(name, 0, name);
                else if (prev.name.compareTo(name) < 0)
                        prev.right = new BSTNode(name, 0, name);
                else prev.left = new BSTNode(name, 0, name);
        }
```

DISPLAY_FIRST

```
public void Display_First(BSTNode p)
        {
                if (p != null)
                {
                        Display(p.left);
                }
                System.out.print(p.name + " ");
        }
```

DISPLAY_LAST

```
        public void Display_Last(BSTNode p)
        {
                if (p != null)
                {
                        Display(p.right);
                }
                System.out.print(p.name + " ");
        }
```

Advanced question 1 phase 1
Taking the input in main class

```
 public static void main(String args[])
    {
        Q1 L = new Q1();
        L.head = L.push(L.head, 30);
        L.head = L.push(L.head, 8);
        L.head = L.push(L.head, 7);
        L.head = L.push(L.head, 5);
        L.head.right = L.push(L.head.right, 20);
        L.head.right = L.push(L.head.right, 10);

        L.head.right.right = L.push(L.head.right.right, 50);
        L.head.right.right = L.push(L.head.right.right, 22);
        L.head.right.right = L.push(L.head.right.right, 19);

        L.head.right.right.right = L.push(L.head.right.right.right, 45);
        L.head.right.right.right = L.push(L.head.right.right.right, 40);
        L.head.right.right.right = L.push(L.head.right.right.right, 35);
        L.head.right.right.right = L.push(L.head.right.right.right, 20);

        L.head = L.flatten(L.head);
        L.printList();
    }
}
```

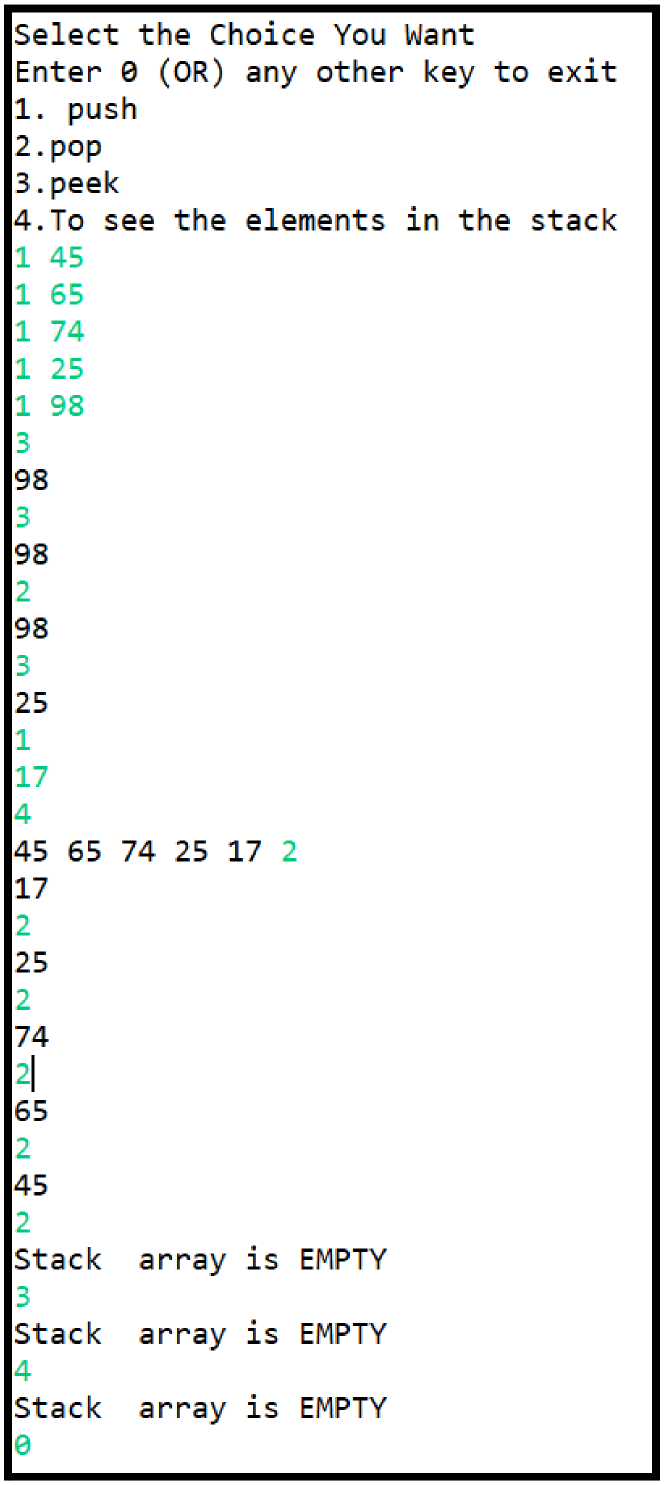
Advanced question 2 phase 1

flattenList which take the array input in and help to generate output

```
 void flattenList(Node node) {
       if (node == null) {
           return;
       }
       Node tmp = null;
       Node tail = node;
       while (tail.next != null) {
           tail = tail.next;
       }
       Node cur = node;
       while (cur != tail) {
           if (cur.child != null) {
               tail.next = cur.child;
               tmp = cur.child;
               while (tmp.next != null) {
                   tmp = tmp.next;
               }
               tail = tmp;
           }
           cur = cur.next;
       }
   }
```

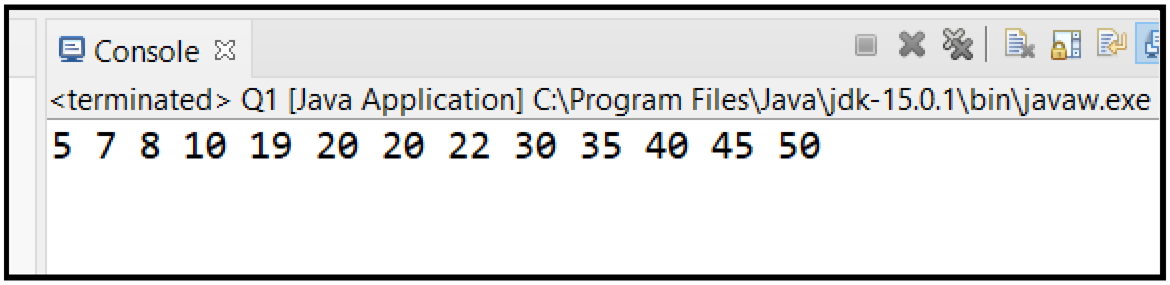
6. Experimentation Result

stack using an ARRAY & stack using an LINKEDLIST



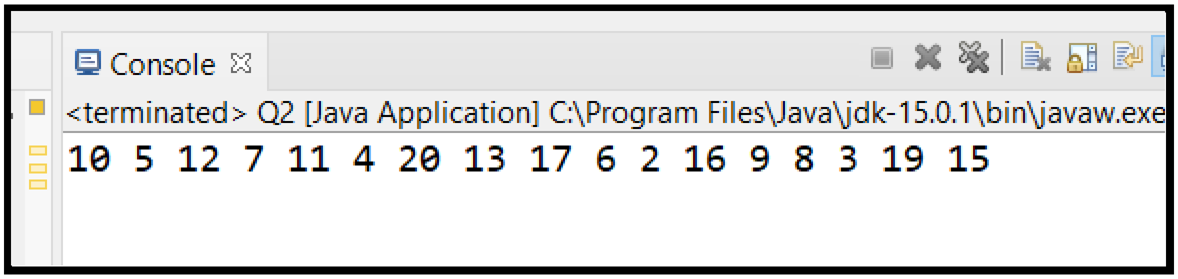
```
Select the Choice You Want
Enter 0 (OR) any other key to exit
1. push
2.pop
3.peek
4.To see the elements in the stack
1 45
1 65
1 74
1 25
1 98
3
98
3
98
2
98
3
25
1
17
4
45 65 74 25 17 2
17
2
25
2
74
2
65
2
45
2
Stack  array is EMPTY
3
Stack  array is EMPTY
4
Stack  array is EMPTY
0
```

Advanced Question 1



```
<terminated> Q1 [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe
5 7 8 10 19 20 20 22 30 35 40 45 50
```

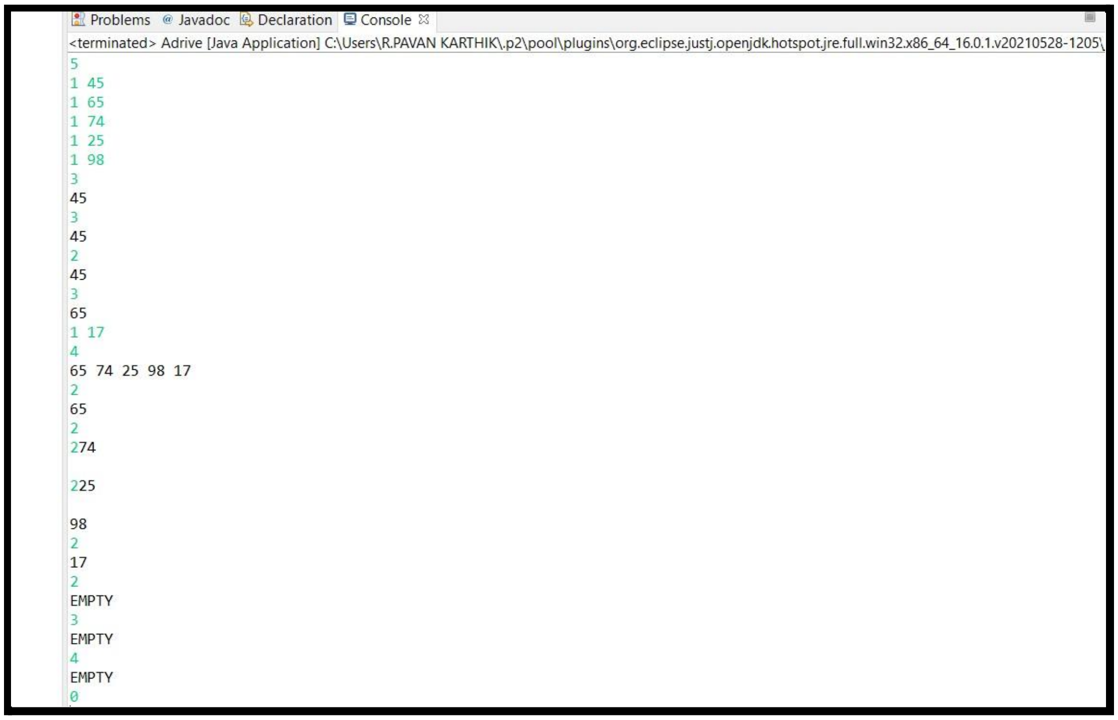
Advanced Question 2



```
<terminated> Q2 [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe
10 5 12 7 11 4 20 13 17 6 2 16 9 8 3 19 15
```

PHASE 2

QUEUE using an ARRAY

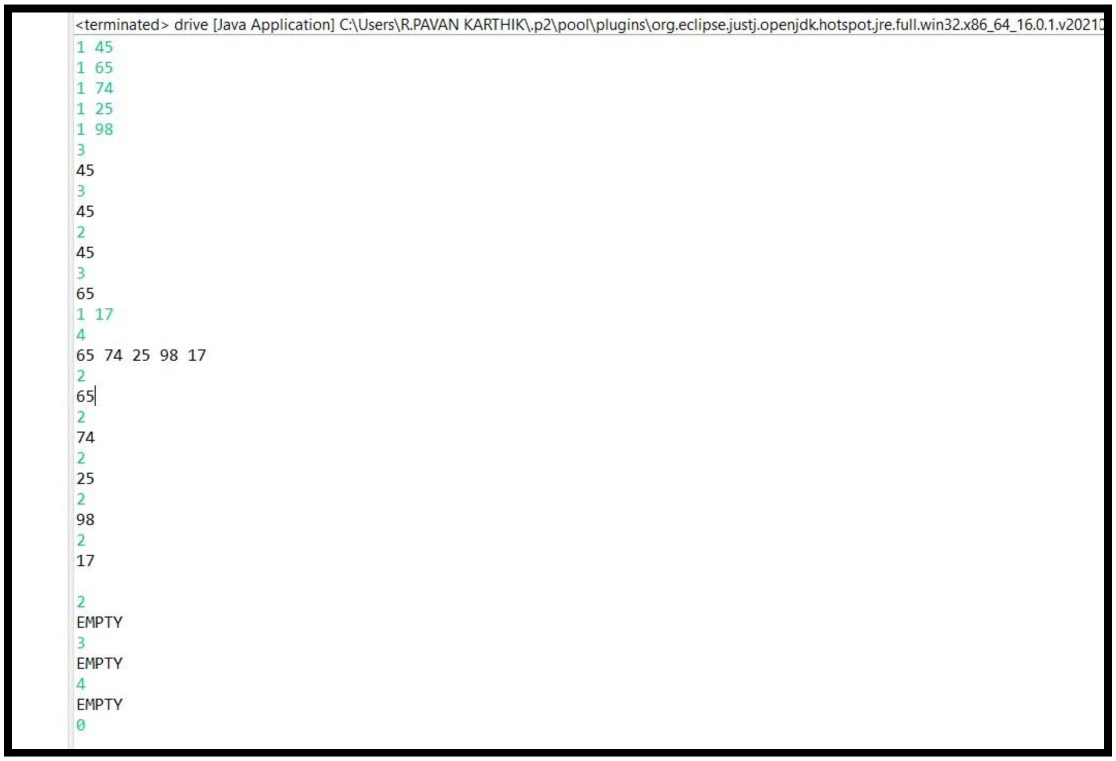


```
<terminated> Adrive [Java Application] C:\Users\R.PAVAN KARTHIK\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\
5
1 45
1 65
1 74
1 25
1 98
3
45
3
45
2
45
3
65
1 17
4
65 74 25 98 17
2
65
2
274

225

98
2
17
2
EMPTY
3
EMPTY
4
EMPTY
0
```

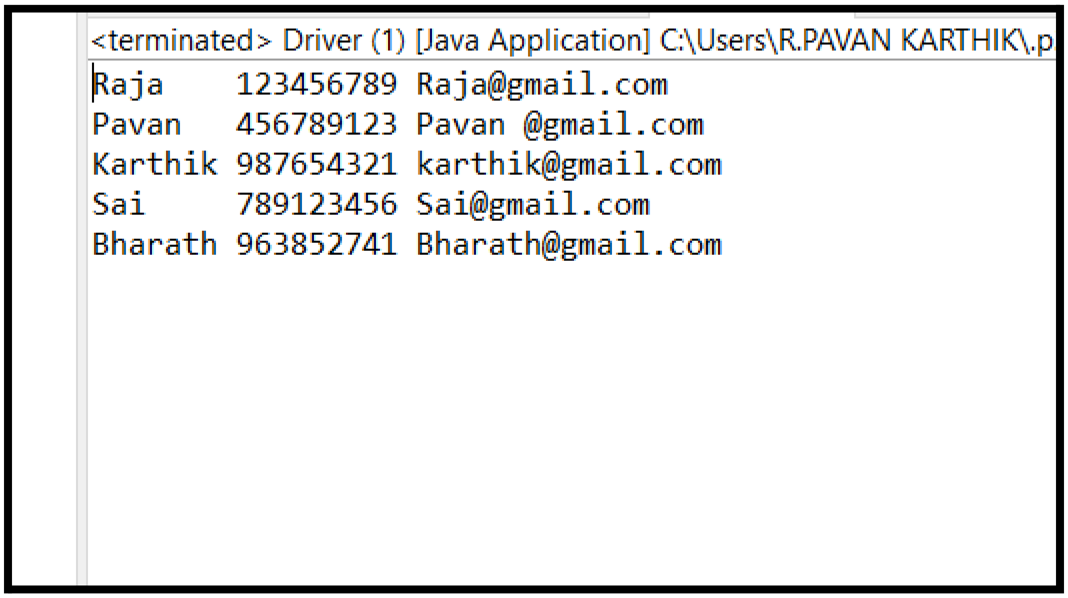
QUEUE using an LINKEDLIST



```
<terminated> drive [Java Application] C:\Users\R.PAVAN KARTHIK\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v2021
1 45
1 65
1 74
1 25
1 98
3
45
3
45
2
45
3
65
1 17
4
65 74 25 98 17
2
65
2
74
2
25
2
98
2
17

2
EMPTY
3
EMPTY
4
EMPTY
0
```

PHONE DIRECTORY USING BST



```
<terminated> Driver (1) [Java Application] C:\Users\R.PAVAN KARTHIK\.p
Raja    123456789 Raja@gmail.com
Pavan   456789123 Pavan @gmail.com
Karthik 987654321 karthik@gmail.com
Sai     789123456 Sai@gmail.com
Bharath 963852741 Bharath@gmail.com
```

7. Conclusion

In this project we worked on the different data structures like stack, linked list, queue...etc. Stacks can be easily implemented using a linked list. Stack is a data structure to which data can be added using the push() method and data can be removed from it using the pop() method. With Linked list, the push operation can be replaced by the addAtFront() method of linked list and pop operation can be replaced by a function which deletes the front node of the linked list.
Worked on the Implementation of queue using an array and Implementation of queue using a linked list. With the help of the enqueue, dequeue,peek and show we successfully ran the programs.for the phone directory java program, like a phone book, loading information from an input file into a Binary search tree, we used insert for Display to show the contents of the entire Directory. Display_first to Display the Contact detail that comes first in the ordering of your phone Book. Display_last to Display the Contact detail that comes last in the ordering of your phone Book. Handle duplicates to This method should handle duplicates in a BST.

8. References:

Rajinikanth. “Study Materials.” *BTech Smart Class*, www.btechsmartclass.com/index.html.

“Simplified Programming Education.” *Studytonight.com*, www.studytonight.com/.

“Where Developers Learn, Share, &amp; Build Careers.” *Stack Overflow*, stackoverflow.com/.

“A Computer Science Portal for Geeks.” *GeeksforGeeks*, www.geeksforgeeks.org/.

9. Appendix

Stack Using Array

```
public class SArray
{
              int arr[];
                  int top=-1;
                  SArray(int sz)
                  {
                        top=-1;
                        arr=new int[sz];
                  }
                  void push(int item)
                  {
                        if(top>=arr.length-1)
                        {
                              System.out.println("Stack  array  OVERFLOW");
                        }
                        else
                        {
                              top=top+1;
                              arr[top]=item;
                        }
                  }
                  void pop()
                  {
                        if(IsEmpty())
                              System.out.println("Stack  array is EMPTY");
                        else
                        {
                              System.out.println(arr[top]);
                              top=top-1;
                        }
                  }
                  boolean IsEmpty()
                  {
                        return (top==-1)?true:false;

                }
                int peek()
                {
                        if(IsEmpty())
                        {
                                System.out.println("Stack  array is EMPTY");
                                return -1;
                        }
                        else {
                                System.out.println(arr[top]);
                                return arr[top];
                        }

                }
                void print()
                {
                        if (top==-1)
                        {
                                System.out.print("Stack  array is EMPTY");
                        }
                        else {
                        for(int i=0;i<=top;i++)
                        {
                                System.out.print(arr[i]+" ");
                        }
                        }
                }
                void printStack() {
                        while (!IsEmpty()) {
                                System.out.print("  "+peek());
                                pop();
                        }

                }
}
```

Stack using Linked List

```
class StackLinked {
    Node top;
      class Node
      {
        int data;
        Node next;
        Node(int value)
        { data = value;
          next = null;
        }
       }
      public boolean isEmpty()
        {
            return top == null;
        }

      void push(int item)
          {
            Node new_node = new Node(item);
            new_node.data = item;
            new_node.next = null;
            new_node.next = top;
            top = new_node;
          }
      void pop()
      {
          if(top==null)
              System.out.println("Stack linked list is EMPTY");
          else
          {
              System.out.print(top.data);
              top = top.next;
          }
      }
      public int peek()
      {
          if (!isEmpty())
          {
            System.out.print(top.data);
              return top.data;
          }
          else
          {
                System.out.println("Stack linked list is EMPTY");
                return -1;
            }
        }
        public void display()
       {
            if (top == null)
            {
                System.out.printf("\nStack linked list is EMPTY");
            }
            else
            {
                Node temp = top;
                while (temp != null)
                {
                    System.out.printf("%d  ", temp.data);
                    temp = temp.next;
                }
            }
        }
}
```

Driver class for Main Question 1 Phase 1

```
public class MainQuestion {

      public static void main(String[] args)
      {
            Scanner in=new Scanner(System.in);
            System.out.println("Please select"+"\n"+
                                 "1. Implement a stack using an array"+
                                     "\n2.Implement a stack using a linked
list");
            int type=in.nextInt();
            if (type==1)
{
            System.out.println("Please enter the Stack limit");
            int size=in.nextInt();
      SArray o1=new SArray(size);
      boolean Work=true;
      System.out.println("Select the Choice You Want"+"\n"+
              "Enter 0 (OR) any other key to exit"+"\n"+
                  "1. push"+"\n"+
                  "2.pop"+"\n"+
                  "3.peek"+"\n"+
                  "4.To see the elements in the stack");
       while (Work)
       {
       int Choice=in.nextInt();
        // reading  the choice
        switch(Choice)
        {
      case 1:
        int E_=in.nextInt();
        o1.push(E_);
             break;
        case 2:
              o1.pop();
              break;
      case 3:
        o1.peek();
              break;
      case 4:
        o1.print();
          break;
        default:
              Work=false;
       break;
        }
     }
}
            if (type==2)
{
            StackLinked o1 =new StackLinked();

            boolean Work=true;
            System.out.println("Select the Choice You Want"+"\n"+
                    "Enter 0 (OR) any other key to exit"+"\n"+
                        "1. push"+"\n"+
                        "2.pop"+"\n"+
                        "3.peek"+"\n"+
                        "4.To see the elements in the stack");
            while (Work)
            {
```

```
            int Choice=in.nextInt();
             // reading  the choice
             switch(Choice)
             {
           case 1:
             int E_=in.nextInt();
             o1.push(E_);
                  break;
             case 2:
                   o1.pop();
                   break;
         case 3:
             o1.peek();
                   break;
         case 4:
             o1.display();
               break;
             default:
                   Work=false;
            break;
             }
         }
}
}
}
```

Advanced Q1

```
class Q1
{
    Node head;
    class Node
    {
        int data;
        Node right, down;
        Node(int data)
        {
            this.data = data;
            right = null;
            down = null;
        }
    }
    Node merge(Node a, Node b)
    {
        if (a == null)     return b;

        if (b == null)      return a;
        Node result;
        if (a.data < b.data)
        {
            result = a;
            result.down =  merge(a.down, b);
        }
        else
        {
            result = b;
            result.down = merge(a, b.down);
        }
        result.right = null;
        return result;
    }
    Node flatten(Node root)
    {
        if (root == null || root.right == null)
            return root;
        root.right = flatten(root.right);
        root = merge(root, root.right);
        return root;
    }

    Node push(Node head_ref, int data)
    {
        Node new_node = new Node(data);
        new_node.down = head_ref;
        head_ref = new_node;
        return head_ref;
    }
    void printList()
    {
        Node temp = head;
        while (temp != null)
        {
            System.out.print(temp.data + " ");
            temp = temp.down;
        }
        System.out.println();
    }
    public static void main(String args[])
    {
        Q1 L = new Q1();
        L.head = L.push(L.head, 30);
        L.head = L.push(L.head, 8);
        L.head = L.push(L.head, 7);
        L.head = L.push(L.head, 5);

        L.head.right = L.push(L.head.right, 20);
        L.head.right = L.push(L.head.right, 10);

        L.head.right.right = L.push(L.head.right.right, 50);
        L.head.right.right = L.push(L.head.right.right, 22);
        L.head.right.right = L.push(L.head.right.right, 19);

        L.head.right.right.right = L.push(L.head.right.right.right, 45);
        L.head.right.right.right = L.push(L.head.right.right.right, 40);
        L.head.right.right.right = L.push(L.head.right.right.right, 35);
        L.head.right.right.right = L.push(L.head.right.right.right, 20);

        L.head = L.flatten(L.head);
        L.printList();
    }
}
```

Advanced Q2

```
class Q2 {
   static Node head;

   class Node {

       int data;
       Node next, child;

       Node(int d) {
           data = d;
```

```
            next = child = null;
        }
    }
    Node createList(int arr[], int n) {
        Node node = null;
        Node p = null;
        int i;
        for (i = 0; i < n; ++i) {
            if (node == null) {
                node = p = new Node(arr[i]);
            } else {
                p.next = new Node(arr[i]);
                p = p.next;
            }
            p.next = p.child = null;
        }
        return node;
    }
    void printList(Node node) {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
        System.out.println("");
    }

    Node createList() {
        int arr1[] = new int[]{10, 5, 12, 7, 11};
        int arr2[] = new int[]{4, 20, 13};
        int arr3[] = new int[]{17, 6};
        int arr4[] = new int[]{9, 8};
        int arr5[] = new int[]{19, 15};
        int arr6[] = new int[]{2};
        int arr7[] = new int[]{16};
        int arr8[] = new int[]{3};

       Node head1 = createList(arr1, arr1.length);
        Node head2 = createList(arr2, arr2.length);
        Node head3 = createList(arr3, arr3.length);
        Node head4 = createList(arr4, arr4.length);
        Node head5 = createList(arr5, arr5.length);
        Node head6 = createList(arr6, arr6.length);
        Node head7 = createList(arr7, arr7.length);
        Node head8 = createList(arr8, arr8.length);

        head1.child = head2;
        head1.next.next.next.child = head3;
        head3.child = head4;
        head4.child = head5;
        head2.next.child = head6;
        head2.next.next.child = head7;
        head7.child = head8;

        return head1;
    }
    void flattenList(Node node) {
        if (node == null) {
            return;
        }
        Node tmp = null;
        Node tail = node;
        while (tail.next != null) {
            tail = tail.next;
        }
        Node cur = node;
        while (cur != tail) {
            if (cur.child != null) {
                tail.next = cur.child;
                tmp = cur.child;
                while (tmp.next != null) {
                    tmp = tmp.next;
                }
                tail = tmp;
            }
            cur = cur.next;
        }
    }
    public static void main(String[] args) {
        Q2 list = new Q2();
        head = list.createList();
        list.flattenList(head);
        list.printList(head);
    }
}
```

Q1

```
//Implement a queue using an array.
   package phase2;
public class AQUEUE
{
       int[] queue;
       int size = 0;
           // Initializing the array with a capacity
       public AQUEUE(int capacity)
       {
             queue = new int[capacity];
       }
           // This method is used to enqueue an item.
           // If the size is full, it will print overflow
       public void enqueue(int element)
       {
             if (size >= queue.length)
             {
                   System.out.println("OVERFLOW");
              return;
           }
           queue[size++] = element;
       }
           // Method to dequeue the item
       public int dequeue()
       {
             if (size == 0)
             {
                   return -1;
           }
           int toReturn = queue[0];
           for (int i = 1; i < queue.length; i++)
           {
             queue[i - 1] = queue[i];
           }
           size--;
           return toReturn;
       }
           // Method to peek the first item in queue
       public int peek()
       {
             if (size == 0)
             {
                   return -1;
           }
           return queue[0];
       }
           // Method to print the queue
       public void show()
       {
             if (size == 0)
             {
                   System.out.println("EMPTY");
           }
             else
             {
                   for (int i = 0; i < size; i++)
                   {
                         System.out.print(queue[i] + " ");
              }
              System.out.println("");
           }
       }
}

package phase2;
import java.util.Scanner;

public class Adrive
{
       public static void main(String[] args) {

         Scanner sc = new Scanner(System.in);
         int size = sc.nextInt();
         AQUEUE q = new AQUEUE(size);

         int ch = sc.nextInt();

         while (ch != 0) {
             switch (ch) {
                 case 1:
                     int value = sc.nextInt();
                     sc.nextLine();
                     q.enqueue(value);
                     break;

                 case 2:
                     value = q.dequeue();
                     if (value == -1) {
                         System.out.println("EMPTY");
                     } else {
                         System.out.println(value);
                     }
                     break;

                 case 3:
                     value = q.peek();
                     if (value == -1) {
                         System.out.println("EMPTY");
                     } else {
                         System.out.println(value);
                     }
                     break;

                 case 4:
                     value = q.peek();
                     if (value == -1) {
                         System.out.println("EMPTY");
                     } else {
                         q.show();
                     }
                     break;
             }

             ch = sc.nextInt();
         }
         sc.close();
     }
       }
```

Q2

```
//QUEUE USING LINKED LIST
package phase2;

public class LLQUEUE
{
      Node head;
    public void enqueue(int data)
    {
      Node toAdd = new Node(data);
      if (head == null)
             {
             head = toAdd;
                 return;
```

```
            }
            Node temp = head;
            while (temp.next != null)
            {
                  temp = temp.next;
            }
            temp.next = toAdd;
    }
                /*
                 * This method will remove the first value in the queue and return
that value.
                 * If the queue is empty, it will return -1.
                 */
            public int dequeue()
            {
                  if (head == null)
                  {
                        return -1;
                }
                  Node temp = head;
                head = temp.next;
                return temp.element;
            }
                /*
                 * This method will return the first value in the queue without
actually removing it.
                 * If the queue is empty, it will return -1.
                 */
            public int peek()
            {
                  if (head == null)
                  {
                        return -1;
                }
                  return head.element;
            }
                // Method to display the entire queue
            public void show()
            {
                  if (head == null)
                  {
                        return;
                }
                  Node temp = head;
                  while (temp != null)
                  {
                        System.out.print(temp.element + " ");
```

```
                    temp = temp.next;
                }
                System.out.println("");
            }
}
            class Node
            {
                public int element;
                public Node next;
                public Node(int element)
                {
                    this.element = element;
                    this.next = null;
                }
          }
package phase2;

import java.util.Scanner;

public class drive
{
      public static void main(String[] args)
      {
        Scanner sc = new Scanner(System.in);
        LLQUEUE q = new LLQUEUE();
        int ch = sc.nextInt();
        while (ch != 0)
        {
            switch (ch)
            {
                case 1:
                    int value = sc.nextInt();
                    sc.nextLine();
                    q.enqueue(value);
                    break;

                case 2:
                    value = q.dequeue();
                    if (value == -1)
                    {
                        System.out.println("EMPTY");
                    }
                    else
                    {
                        System.out.println(value);
                    }
                    break;
                case 3:
                    value = q.peek();
                    if (value == -1)
                    {
                        System.out.println("EMPTY");
                    }
                    else
                    {
                        System.out.println(value);
                    }
                    break;

                case 4:
                    value = q.peek();
                    if (value == -1)
                    {
                        System.out.println("EMPTY");
                    } else
                    {
                        q.show();
                    }
                    break;
            }
            ch = sc.nextInt();
        }
        sc.close();
    }
}
```

ADVANCED Q3

```
package phase2;

public class BST
{
        protected BSTNode root = null;
        public BST(){}

        public void clear()
        {
                root = null;
        }
        public boolean isEmpty()
        {
                return root == null;
        }
        public void insert(String name, String email, int phone)
        {
                BSTNode p = root, prev = null;
                while (p != null)
                {
                        prev = p;
                        if (p.name.compareTo(name) < 0)
                                p = p.right;
                        else p = p.left;
                }
                if (root == null)
                        root = new BSTNode(name, 0, name);
                else if (prev.name.compareTo(name) < 0)
                        prev.right = new BSTNode(name, 0, name);
                else prev.left = new BSTNode(name, 0, name);
        }
        public void Display()
        {
                Display(root);
        }
public void Display(BSTNode p)
        {
                if (p != null)
                {
                        Display(p.left);
                        System.out.print(p.name + " ");
                        Display(p.right);
                }
        }
        public void Display_First()
        {
                Display(root);
        }
        public void Display_First(BSTNode p)
        {
                if (p != null)
                {
                        Display(p.left);
                }
                System.out.print(p.name + " ");
        }
        public void Display_Last()
        {
                Display(root);
        }
        public void Display_Last(BSTNode p)
        {
                if (p != null)
```

```
            {
                    Display(p.right);
            }
            System.out.print(p.name + " ");
        }
        public void Search(String name)
        {
            Search(toString() ,root);
        }
        public void Search(String name,BSTNode p)
        {
            if (p != null)
            {
                    Search(name,p.left);
                    if(p.name==name)
                    {
                            return ;
                    }
                    Search(name,p.right);
            }
        }
}
package phase2;

public class BSTNode
{
        protected String name;
        protected int phone;
        protected String email;
        protected BSTNode left, right;

        public BSTNode()
        {
            left = null;
            right = null;
        }
        public BSTNode(String name,int phone,String email)
        {
            this(name,phone,email,null,null);
        }
        public BSTNode(String name,int phone,String email, BSTNode lt, BSTNode rt)
        {
            this.name = name;
            this.phone = phone;
            this.email = email;
            left = lt;
            right = rt;
```

```
       }
       public void insert(String read)
       {
       // TODO Auto-generated method stub
       }
}
package phase2;
import java.io.*;
import java.util.Scanner;
public class Driver
{
       public static void main(String[] args) throws IOException
       {
             Scanner in = new Scanner(System.in);
             BSTNode fg = new BSTNode();
       //    BST fgn = new BST();
             File fileIn = new File("D:\\EOC\\src\\phase2\\rtyt.txt");
        Scanner fr = new Scanner(fileIn);
        String data;
 //       File file = new File("C:\\Users\\pankaj\\Desktop\\test.txt");

        BufferedReader br = new BufferedReader(new FileReader(fileIn));
        String st;
        while ((st = br.readLine()) != null)
          System.out.println(st);
        }
}
```

THANK YOU!