ATSS's
Institute of Industrial and Computer Management and Research, Nigdi Pune
MCA Department
Academic Year :2022-23

Practical Journal

on

IT11L- Data Structure and Algorithms (SEM-I)

Submitted By:

Achal anilsingh Pardeshi

20291

03

Date : Friday 14-03-2023

Course Outcomes:

Student will be able to :

CO1: Demonstrate Collection framework (Apply)

CO2: Develop GUI using awt and swing (Apply)

CO3: Develop Web application using JSP and Servlet, JDBC (Apply)

CO4: Apply Data Structure to solve problems using JavaScript (Apply)

CO5: Demonstrate the concepts of Core Java (Apply)

ATSS's

Institute of Industrial and Computer Management and Research, Nigdi Pune

MCA Department

INDEX

Students Name : Achal Anilsingh Pardeshi **Roll No.** 03

Sr. No	Program Title	Course Outcome	Page No.	Teacher's Sign with Date	Remarks
1.	Write a program to implement Singly linked list with required member function(Create, insert, delete, Display)				
2.	Write a program to implement Doubly linked list with required member function(Create, insert, delete, Display)				
3.	Write a program to implement STACK using Array with PUSH, POP operations				
4.	Write a program to implement Stack using Linked List				
5.	Write a application of stack to Check for balanced parentheses.				
6.	Write a program to Reverse a string using stack				
7.	Write a program to implement Linear Queue,				
8.	Write a program to Reverse stack using queue				
9.	Write a program to implement binary search tree with its operations				
10.	Write a program to implement Circular Queue,				
11.	Write a Program to print Adjacancy Matrix and AdjacancyList by reading Edges of Graph				
12.	Write a Program to find the element in an array using Binary Search				
13.	Write a Program to find the element in an array using Linear Search				
14.	Write a Program to implement the following 1.Print Pascal's triangle for n=5				

15.	Write a Program to implement the following GCD of two numbers using Euclidean Algorithm				
16.	Write a program to implement 1. tower of Hanoi where number of disks=4				
17.	Write a program to implement 2. Fibonacci series till N				

Q.1 Write a program to implement

Singly linked list with required member function(Create, insert, delete, Display)

Solution: Program

```
class Node{
   constructor(data)
   {
      this.data = data;
      this.next = null
   }
}

class LinkedList {
   constructor()
   {
      this.head = null;
      this.tail = null;
      this.size = 0;
      this.pos;
   }

   InsertAtBegin(data){
      var temp = new Node();
      if(this.head==null){
         this.head = temp;
         this.size++;
      }
      else{
         temp.next=this.head;
         this.head=temp;
         this.size++;
      }
   }


   InsertAtEnd(data){
      var temp = new Node();
      temp.data = data;
      temp.next=null;
```

```
    if(this.head==null){
        this.head = temp;
        this.size++;
    }
    else{
        var current = this.head;
        while(current.next!=null){
            current=current.next;
        }
        current.next = temp;
        this.size++;
    }
}

InsertAtPosition(data,pos){
    var temp = new Node();
    temp.data = data;
    temp.next=null;

    if(pos<0 || pos>=this.size+2){
        console.log('Invalid Position :(');
    }
    else{
        if(pos==1){
            InsertAtBegin(data);
        }
        else if(pos==this.size+1){
            InsertAtEnd(data);
        }
        else{
            var current = this.head;
            var index = 1;
            while(index<pos-1){
                current = current.next;
                index++;
            }
            temp.next = current.next;
            current.next = temp;
            this.size++;
        }
    }
}

DeleteAtFirst(){
    if(this.head==null){
```

```
        console.log('List Is Empty');
    }else{
        if(this.head.next==null){
            var temp = this.head;
            this.head=null;
           // this.tail=null;
            this.size--;
        }
        else{
            this.temp = this.head;
            this.head = this.head.next;
            this.size--;
        }
    }

}

DeleteAtLast(){
    if(this.head==null){
        console.log('List Is Empty');
    }
    else{
        if(this.head.next==null){
            var temp = head;
            this.head=null;
           // this.tail=null;
            this.size--;
        }else{
           var second_last = this.head;
           while (second_last.next.next != null)
              second_last = second_last.next;
           second_last.next = null;
           this.size--;

        }
    }
}

DeleteAtPos(pos){
   if(pos<=0 || pos>this.size){
      console.log("Invalid pos:(");
   }
   else if(pos==1){
      this.DeleteAtFirst();
   }
```

```
      else if(pos==this.size){
         this.DeleteAtLast();
      }
      else{
         var temp = new Node();
         var trv = this.head;
         var index = 1;
         while(index<pos-1){
            trv = trv.next;
            index++;
         }
         temp = trv.next;
         trv.next=temp.next;
         temp.next = null;
         this.size--;
      }
   }


    Display(){
       var current=this.head;
       if(this.head==null){
          console.log("List is Empty :(")
       }
       else{
          while(current!=null){
             console.log(current.data+" ");
             current=current.next;
          }
       }
    }

}

L1 = new LinkedList();
L1.InsertAtBegin(20);
L1.InsertAtBegin(30);
L1.InsertAtBegin(40);
L1.InsertAtEnd(100);
L1.InsertAtPosition(3,3);
console.log("After Inserting Element: ");
L1.Display();
console.log("Size of LL :"+L1.size);
console.log("After Deleting Element: ");
L1.DeleteAtFirst();
```

```
L1.DeleteAtLast();
L1.DeleteAtPos(2);
L1.Display();
console.log("Size of LL :"+L1.size);
```

Output:

```
PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\SinglyLL.js"
After Inserting Element:
40
30
3
20
100
Size of LL :5
After Deleting Element:
30
20
Size of LL :2
PS F:\Uniquesp42>
```

Q2. Write a program to implement

Doubly linked list with required member function(Create, insert, delete, Display)

Solution:

```
class DNode{
   constructor(data)
   {
      this.data = data;
      this.next = null
      this.pre = null;
   }
}

class DoublyLL{
      constructor(){
         this.head = null;
         this.tail = null;
         this.pre = null;
         this.size = 0;
         this.pos;
      }

   InsertAtBegin(data){
      var temp = new DNode(data);
      temp.pre = null;
      temp.next = null;
      temp.data = data;

      if(this.head==null){
         this.head = temp;
         this.tail = temp;
         this.size++;
      }else{
         temp.next = this.head;
         this.head.pre = temp;
         this.head = temp;
         this.size++;
      }
   }

   InsertAtLast(data){
      var temp = new DNode(data);
```

```
    if(this.head==null){
        this.head=temp;
        this.tail=temp;
        this.size++;
    }
    else{
        var trv = this.head;
        while(trv.next!=null){
            trv = trv.next;
        }
        trv.next=temp;
        temp.pre = trv;
        this.tail = temp;
        this.size++;
    }
}

InsertAtPos(data,pos){
    var temp = new DNode(data);
    if(pos<0 || pos>=this.size+2){
        console.log("Invalid Pos:(");
    }
    else{
        if(pos==1){
            InsertAtBegin(data);
        }
        else if(pos==this.size+1){
            InsertAtEnd(data);
        }
        else{
            var current = this.head;
            var index = 1;
            while(index<pos-1){
                current = current.next;
                index++;
            }
             temp.next = current.next;
             current.next = temp;
             temp.next.pre = temp;
             this.size++;
        }
    }
}

DeleteAtFirst(){
```

```
    if(this.head==null){
        console.log('List Is Empty');
    }else{
        if(this.head.next==null){
            var temp = this.head;
            this.head=null;
           // this.tail=null;
            this.size--;
        }
        else{
            temp = this.head;
            this.head = this.head.next;
            this.head.pre = null;
            this.size--;
        }
    }

}

DeleteAtLast(){
    if(this.head==null){
        console.log('List Is Empty');
    }
    else{
        if(this.head.next==null){
            var temp = head;
            this.head=null;
           // this.tail=null;
            this.size--;
        }else{
          var trv = this.head;
          while(trv.next!=null){
           trv=trv.next;
          }
          trv = trv.pre;
          temp = trv.next;
          trv.next = null;
          this.size--;
        }
    }
}

DeleteAtPos(pos){
    if(pos<=0 || pos>this.size){
        console.log("Invalid pos:(");
```

```
        }
        else if(pos==1){
            this.DeleteAtFirst();
        }
        else if(pos==this.size){
            this.DeleteAtLast();
        }
        else{
            var temp = new DNode();
            var trv = this.head;
            var index = 1;
            while(index<pos-1){
                trv = trv.next;
                index++;
            }
            temp = trv.next;
            trv.next.pre = trv;
            this.size--;
        }
    }

    Display(){
        var current=this.head;
        if(this.head==null){
            console.log("List is Empty :(")
        }
        else{
            while(current!=null){
                console.log(current.data+" ");
                current=current.next;
            }
        }
    }

}

var DLL = new DoublyLL();
DLL.InsertAtLast(99);
DLL.InsertAtBegin(30);
DLL.InsertAtBegin(40);
DLL.InsertAtLast(78);
DLL.InsertAtPos(44,3)
console.log("After Inserting Element: ");
DLL.Display();
console.log("Size of LL :"+DLL.size);
```

```
console.log("After Deleting Element: ");
DLL.DeleteAtFirst();
DLL.DeleteAtLast();
DLL.DeleteAtPos(2);
DLL.Display();
console.log("Size of LL :"+DLL.size);
```

OUTPUT:

```
PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\DoublyLL.js"
After Inserting Element:
40
30
44
99
78
Size of LL :5
After Deleting Element:
30
44
99
Size of LL :2
PS F:\Uniquesp42>
```

Q3. Write a program to implement STACK using Array with PUSH, POP operations

Solution:

```
class Stack{
   constructor(sz){
      this.Arr = new Array(sz);
      this.top=-1;
      this.size=sz;
   }

   isFull(){
      if(this.size==this.size-1)
         return true;
      else
         return false;
   }

   isEmpty(){
      if(this.top==-1)
         return true;
      else
         return false;
   }

   push(data){
      if(this.isFull()){
         console.log("Stack Is Full :");
      }
      else{
         this.top++;
         this.Arr[this.top] = data;
      }
   }

   pop(){
      if(this.isEmpty()){
         console.log("Stack Is Empty :");
      }
      else{
         var t = this.Arr[this.top];
         this.top--;
      }
      return t;
```

```
    }

    display(){
        var str = " ";
        for(let i=0;i<=this.top;i++){
            str+=this.Arr[i]+" ";
        }
        console.log(str);
    }
}

let s1 = new Stack(5);
s1.push(3);
s1.push(4);
s1.push(6);
s1.push(8);
s1.push(56);
s1.push(47);
s1.display();
console.log("Poped element is : "+s1.pop());
s1.display();
```

OUTPUT:

```
PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\StackArr.js"
 3 4 6 8 56 47
Poped element is : 47
 3 4 6 8 56
PS F:\Uniquesp42>
```

Q4. Write a program to implement Stack using Linked List

Solution:

```
class StackNode{
   constructor(data){
      this.data=data;
      this.next=null;
   }
}

class StackLL{
      constructor(s){
         this.top=null;
         this.cnt=0;
         this.size=s;
      }

      GetSize(){
         return this.cnt;
      }

      IsEmpty(){
         if(this.top==null)
            return 1;
         else
            return 0;
      }

      IsFull(){
         if(this.cnt==this.size)
            return 1;
         else
            return 0;
      }

      push(data){
         if(this.IsFull()){
            console.log("Stack Is Full...");
            return;
         }
            var temp = new StackNode(data);
            if(this.top==null){
               this.top++;
               this.top=temp;
```

```
            this.cnt++;
        }
        else{
            temp.next=this.top;
            this.top=temp;
            this.cnt++;
        }
    }

    pop(){
        if(this.IsEmpty()){
            console.log("Stack is Empty....");
            return;
        }
        else{
            var t = this.top;
            this.top=t.next;
            this.cnt--;
            return t.data;
        }
    }

    Display(){
        if(this.top==null){
            console.log("List is Empty...")
            return;
        }
        else{
            var current=this.top;
         while(current!=null){
             console.log(current.data+" ");
             current=current.next;
            }
        }
    }

}

let s1 = new StackLL(5);
s1.push(3);
s1.push(4);
s1.push(7);
s1.push(6);
s1.push(1);
s1.push(2);
```

```
s1.Display();
console.log("Popped Elment is: "+s1.pop());
s1.Display();
```

OUTPUT:

```
PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\StackLL.js"
Stack Is Full...
1
6
7
4
3
Popped Elment is: 1
6
7
4
3
PS F:\Uniquesp42>
```

Q5. Write a application of stack to Check for balanced parentheses.

Solution:

```
// Stack
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Stack {
  constructor(size) {
    this.data = [];
    this.size = size;
    this.top = -1;
    this.length = 0;
  }

  isEmpty() {
    if (this.length === 0) {
      return true;
    }
    return false;
  }

  isFull() {
    if (this.length === this.size) {
      return true;
    }
    return false;
  }

  // unshift add element at first
  push(value) {
    if (this.isFull()) {
      return 'Stack is full';
    }
    this.top++;
    this.data[this.top] = value;
    this.length++;
    return true;
  }
```

```
  // shift //remove element from first
  pop() {
    if (this.isEmpty()) return 'Stack is empty';
    else {
      let removeElm = this.data[this.top];
      this.data.pop();
      this.top--;
      this.length--;

      return removeElm;
    }
  }

  display() {
    for (let i = 0; i < this.length; i++) {
      console.log(this.data[i]);
    }
  }
}

const parenthesisChecker = (str) => {
  const s = new Stack();

  for (let i of str) {
    if (i == '(' || i == '[' || i == '{') {
      s.push(i);
    }

    if (i == ')' || i == ']' || i == '}') {
      s.pop();
    }
  }
  if (!s.length) {
    console.log(` Valid parenthesis` );
  } else {
    console.log(` In-valid parenthesis` );
  }
};

parenthesisChecker('(a+b)+(a-b)');
parenthesisChecker('(a+b)+(a-b)[+[a/b]{');
```

OUTPUT:

```
PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\BalncedParanthesis.js"
Valid parenthesis
In-valid parenthesis
PS F:\Uniquesp42>
```

Q6. Write a program to Reverse a string using stack

Solution:

```
class Stack{
   constructor(n){
      this.top = -1;
      this.size = n;
      this.a = new Array(this.size);
   }

   isEmpty(){
      return(this.top < 0);
   }

   push(x){
      if (this.top >= this.size){
         console.log("Stack Overflow<br>");
         return false;
      }
      else{
         this.a[++this.top] = x;
         return true;
      }
   }

   pop(){
      if (this.top < 0){
         console.log("Stack Underflow<br>");
         return 0;
      }
      else{
         let x = this.a[this.top--];
         return x;
      }
   }
}

function reverse(str){

   let n = str.length;
   let obj = new Stack(n);

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

```
        obj.push(str[i]);

    for(i = 0; i < n; i++){
        let ch = obj.pop();
        str[i] = ch;
    }
}


let s = "GeeksQuiz".split("");
reverse(s);
console.log("Reversed string is " +s.join(""));
```

OUTPUT:

```
PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\ReverseStack.js"
Reversed string is ziuQskeeG
PS F:\Uniquesp42>
```

Q7. Write a program to implement Linear Queue

Solution:

```
class Queue {
  constructor(size) {
    this.data = [];
    this.front = -1;
    this.rear = -1;
    this.size = size;
    this.length = 0;
  }
  isEmpty() {
    return this.length === 0;
  }
  isFull() {
    return this.length === this.size;
  }

  enqueue(value) {
    if (this.isFull()) return 'Queue is full';
    if (this.isEmpty()) {
      this.rear++;
      this.front++;
      this.data[this.front] = value;
    } else {
      this.rear++;
      this.data[this.rear] = value;
    }
    this.length++;
    return true;
  }

  dequeue() {
    if (this.isEmpty()) return 'Queue is empty';
    if (this.length === 1) {
      var removeNode = this.data[this.front];
      this.data[this.front] = null;
      this.front = -1;
      this.rear = -1;
    } else {
      var removeNode = this.data[this.front];
      this.data[this.front] = null;
      this.front++;
    }
```

```
    this.length--;
    return `removeNode is ${removeNode}`;
  }

  display() {
    if (this.front === -1) return null;

    for (let i = this.front; i <= this.rear; i++) {
      console.log(this.data[i]);
    }
  }
}

let q = new Queue(4);
q.enqueue(4);
q.enqueue(5);
q.enqueue(41);
q.enqueue(1);
console.log("After Insertion in Queu: ");
q.display();
console.log(q.dequeue());
console.log(q.dequeue());
console.log("After Deletion in Queue: ");
q.display();
```

OUTPUT:

```
PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\LinearQ.js"
After Insertion in Queu:
4
5
41
1
removeNode is 4
removeNode is 5
After Deletion in Queue:
41
1
PS F:\Uniquesp42>
```

Q8 Write a program to Reverse stack using queue

Solution:

```
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Stack {
  constructor(size) {
    this.data = [];
    this.size = size;
    this.top = -1;
    this.length = 0;
  }

  isEmpty() {
    if (this.length === 0) {
      return true;
    }
    return false;
  }

  isFull() {
    if (this.length === this.size) {
      return true;
    }
    return false;
  }

  // unshift add element at first
  unShift(value) {
    if (this.isFull()) {
      return 'Stack is full';
    }
    this.top++;
    this.data[this.top] = value;
    this.length++;
    return true;
  }

  // shift //remove element from first
```

```
  shift() {
    if (this.isEmpty()) return 'Stack is empty';
    else {
      let removeElm = this.data[this.top];
      this.data.pop();
      this.top--;
      this.length--;

      return removeElm;
    }
  }

  display() {
    for (let i = 0; i < this.length; i++) {
      console.log(this.data[i]);
    }
  }
}

// Queue
class Queue {
  constructor(size) {
    this.data = [];
    this.front = -1;
    this.rear = -1;
    this.size = size;
    this.length = 0;
  }
  isEmpty() {
    return this.length === 0;
  }
  isFull() {
    return this.length === this.size;
  }
  // rear - back
  enqueue(value) {
    if (this.isFull()) return 'Queue is full';
    if (this.isEmpty()) {
      this.rear++;
      this.front++;
      this.data[this.front] = value;
    } else {
      this.rear++;
      this.data[this.rear] = value;
    }
```

```
    this.length++;
    return true;
  }

  dequeue() {
    if (this.isEmpty()) return 'Queue is empty';
    if (this.length === 1) {
      var removeNode = this.data[this.front];
      this.data[this.front] = null;
      this.front = -1;
      this.rear = -1;
    } else {
      var removeNode = this.data[this.front];
      this.data[this.front] = null;
      this.front++;
    }
    this.length--;
    return removeNode;
  }

  display() {
    if (this.front === -1) return null;
for (let i = this.front; i <= this.rear; i++) {
      console.log(this.data[i]);
    }
  }
}

const reverse = () => {
  const s = new Stack();
  const q = new Queue();

  s.unShift(33);
  s.unShift(8);
  s.unShift(17);
  s.unShift(13);
  s.unShift(71);

  console.log(` Stack is` );
  s.display();

  while (s.length) {
    q.enqueue(s.shift());
  }
```

```
  while (q.length) {
    s.unShift(q.dequeue());
  }

  console.log(` After reversing stack using queue` );
  s.display();
};

reverse();
```

OUTPUT:

```
PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\ReverseStkQue.js"
Stack is
33
8
17
13
71
After reversing stack using queue
71
13
17
8
33
PS F:\Uniquesp42>
```

Q9 Write a program to implement binary search tree with its operations

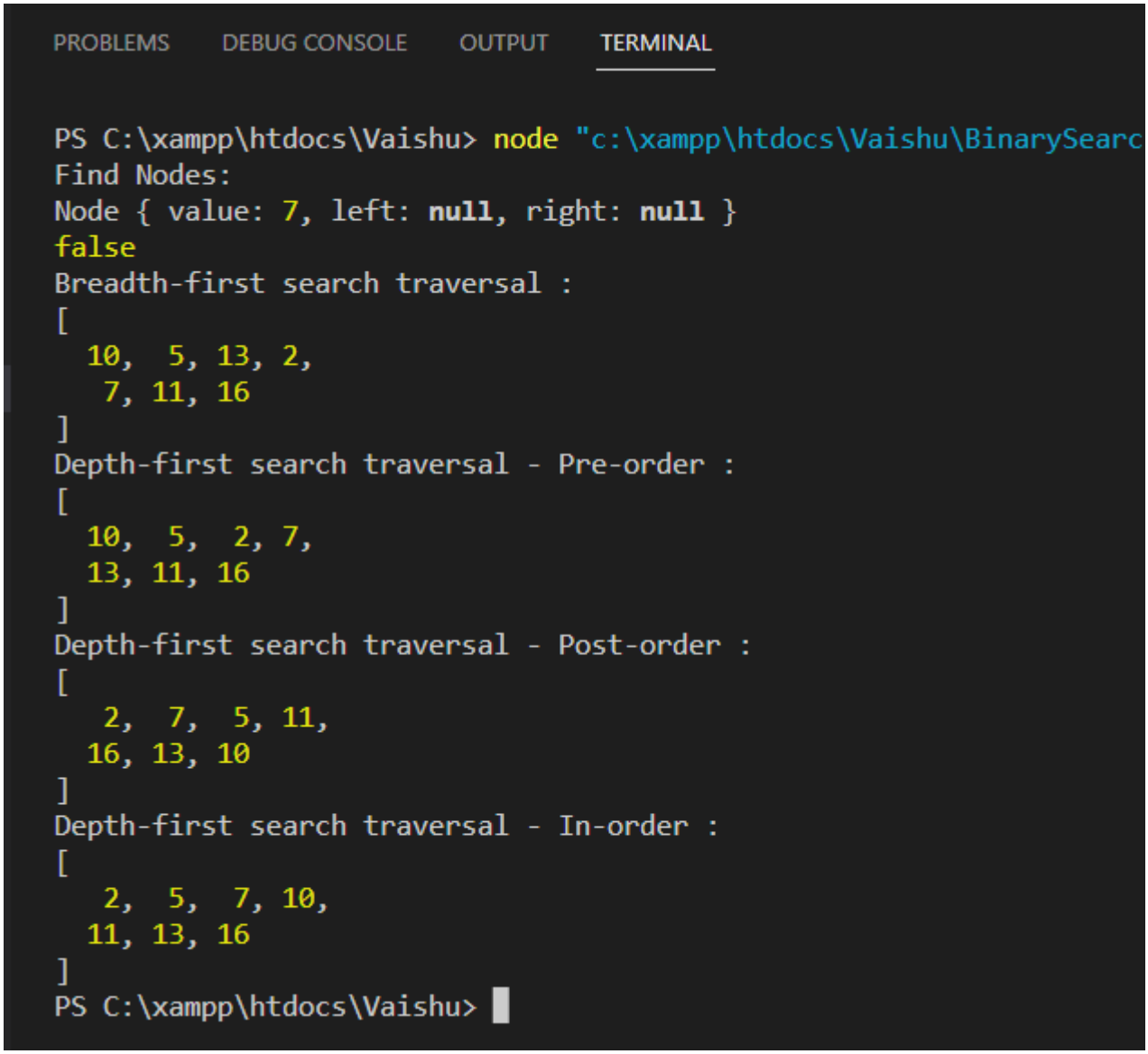
Solution:

```
class Node {
  constructor(value) {
    this.value = value;
    this.left = null;
    this.right = null;
  }
}

class BinarySearchTree {
  constructor() {
    this.root = null;
  }

  insert(value) {
    const newNode = new Node(value);

    if (this.root === null) {
      this.root = newNode;
    } else {
      let current = this.root;
      while (true) {
        if (value === current.value) return undefined;
        if (value < current.value) {
          if (current.left === null) {
            current.left = newNode;
            return this;
          }
          current = current.left;
        } else {
          if (current.right === null) {
            current.right = newNode;
            return this;
          }
          current = current.right;
        }
      }
    }
  }

  find(value) {
    if (this.root === null) return false;
    let current = this.root;
    let found = false;
```

```
  while (current && !found) {
    if (value < current.value) {
      current = current.left;
    } else if (value > current.value) {
      current = current.right;
    } else {
      found = true;
    }
  }
  if (!found) return false;
  return current;
}

bfs() {
  let node = this.root;
  let queue = [];
  let data = [];
  queue.push(node);

  while (queue.length) {
    node = queue.shift();
    data.push(node.value);
    if (node.left) queue.push(node.left);
    if (node.right) queue.push(node.right);
  }
  return data;
}

dfsPreOrder() {
  let data = [];
  function traverse(node) {
    data.push(node.value);
    if (node.left) traverse(node.left);
    if (node.right) traverse(node.right);
  }
  traverse(this.root);
  return data;
}

dfsPostOrder() {
  let data = [];
  function traverse(node) {
    if (node.left) traverse(node.left);
    if (node.right) traverse(node.right);
    data.push(node.value);
  }
  traverse(this.root);
```

```
    return data;
  }

  dfsInOrder() {
    let data = [];
    function traverse(node) {
      if (node.left) traverse(node.left);
      data.push(node.value);
      if (node.right) traverse(node.right);
    }
    traverse(this.root);
    return data;
  }
}

const bst = new BinarySearchTree();

// Insert nodes
bst.insert(10);
bst.insert(5);
bst.insert(13);
bst.insert(11);
bst.insert(2);
bst.insert(16);
bst.insert(7);

console.log("Find Nodes: ");
console.log(bst.find(7)); // Node { value: 7, left: null, right: null }
console.log(bst.find(12)); // false

console.log("Breadth-first search traversal : ");
console.log(bst.bfs()); // [ 10, 5, 13, 2, 7, 11, 16 ]

console.log("Depth-first search traversal - Pre-order : ")
console.log(bst.dfsPreOrder()); // [ 10, 5, 2, 7, 13, 11, 16 ]

console.log("Depth-first search traversal - Post-order : ");
console.log(bst.dfsPostOrder()); // [ 2, 7, 5, 11, 16, 13, 10 ]

console.log("Depth-first search traversal - In-order : ");
console.log(bst.dfsInOrder()); // [ 2, 5, 7, 10, 11, 13, 16 ]
```

OUTPUT:


PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
PS C:\xampp\htdocs\Vaishu> node "c:\xampp\htdocs\Vaishu\BinarySearc
Find Nodes:
Node { value: 7, left: null, right: null }
false
Breadth-first search traversal :
[
10, 5, 13, 2,
7, 11, 16
]
Depth-first search traversal - Pre-order :
[
10, 5, 2, 7,
13, 11, 16
]
Depth-first search traversal - Post-order :
[
2, 7, 5, 11,
16, 13, 10
]
Depth-first search traversal - In-order :
[
2, 5, 7, 10,
11, 13, 16
]
PS C:\xampp\htdocs\Vaishu>

Q10 Write a program to implement Circular Queue

Solution:

```
class Queue {
      constructor(size) {
        this.Arr = new Array();
        this.front = -1;
        this.rare = -1;
        this.size = size;
      }
      isEmpty() {
        if (this.front == -1) {
          return 1;
        } else {
          return 0;
        }
      }
      isFull() {
        if (
          (this.front == 0 && this.rare == this.size - 1) ||
          this.rare + 1 == this.front
        ) {
          return 1;
        } else {
          return 0;
        }
      }
      EnQueue(val) {
        if (this.isFull()) {
         console.log("Queue is full..");
          return;
        } else if (this.isEmpty()) {
          this.front++;
          this.rare++;
          this.Arr[this.rare] = val;
        } else {
          this.rare = this.rare + (1 % this.size);
          this.Arr[this.rare] = val;
        }
      }
      DeQueue() {
        if (this.isEmpty()) {
         console.log("Queue is empty");
        } else if (this.front == this.rare) {
```

```
        var val = Arr[this.front];
        this.front = -1;
        this.rare = -1;
        //return val;
      } else {
        var val = this.Arr[this.front];
        this.front = (this.front + 1) % this.size;
        //return val;
      }
    }
    Display() {
      var str = "";
      for (var i = this.front; i != this.rare; i = i + (1 % this.size)) {
        str = str + this.Arr[i] + " ";
      }
      str = str + this.Arr[i] + " ";
     console.log(str);
    }
  }
  const qobj = new Queue(7);
  qobj.EnQueue(10);
  qobj.EnQueue(20);
  qobj.EnQueue(30);
  qobj.EnQueue(40);
  qobj.EnQueue(50);
  qobj.Display();
  qobj.DeQueue();
  qobj.Display();
```

OUTPUT:

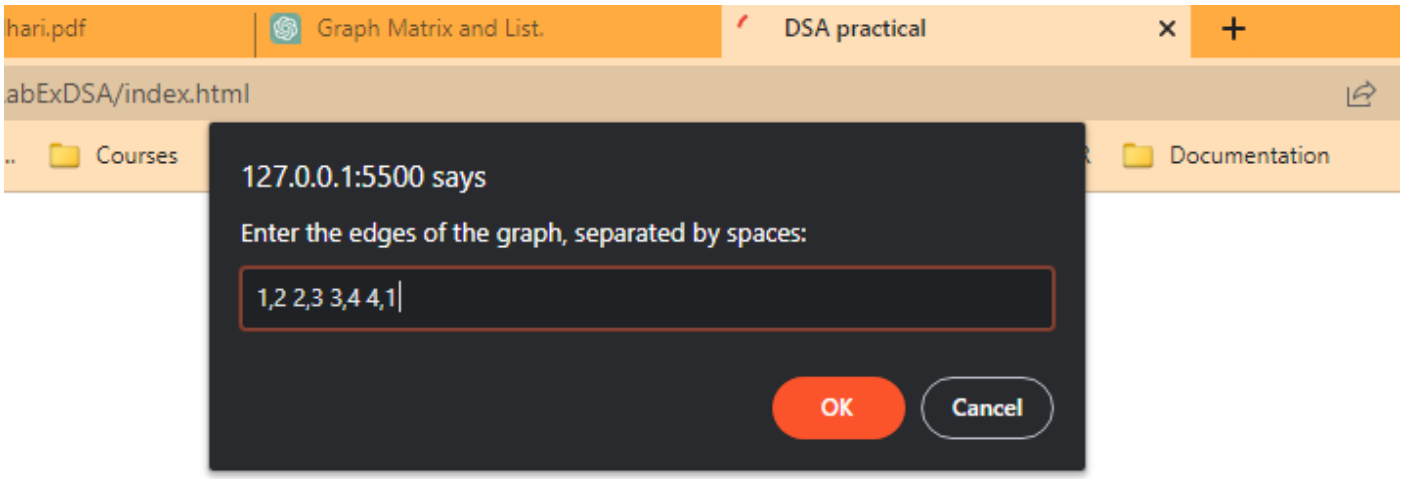
```
PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\CircularQ.js"
10 20 30 40 50
20 30 40 50
PS F:\Uniquesp42>
```

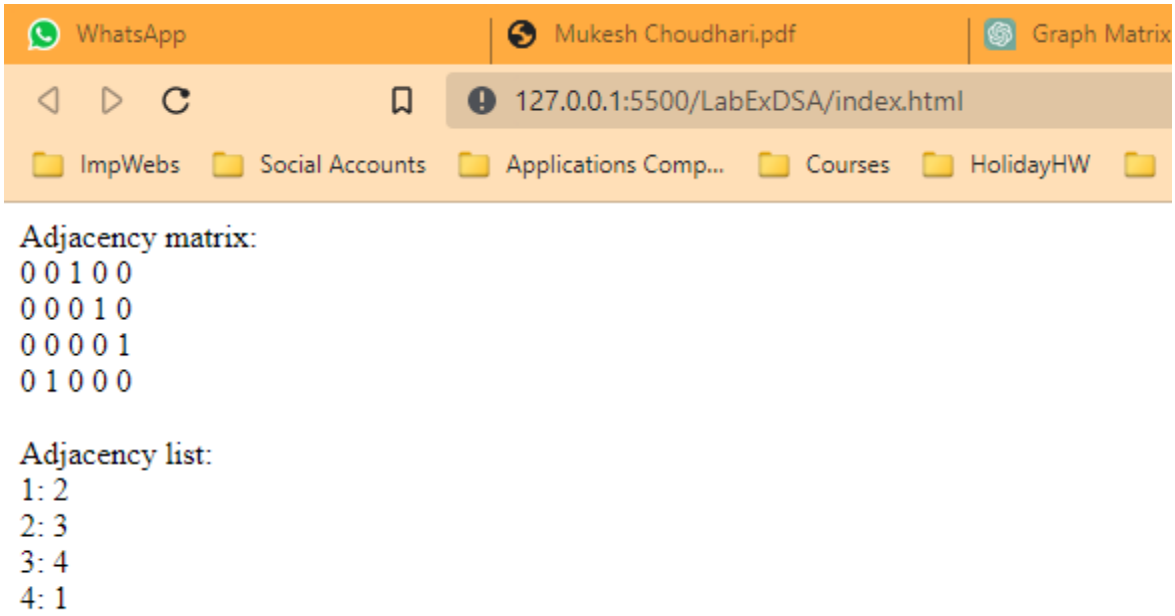
Q11. Write a Program to print Adjacancy Matrix and AdjacancyList by reading Edges of Graph

Solution:

```
function createMatrix(n) {
  const matrix = [];
  for (let i = 0; i < n; i++) {
    matrix.push(new Array(n).fill(0));
  }
  return matrix;
}

const input = prompt("Enter the edges of the graph, separated by spaces:");
const edges = input.split(" ");
let n = 0;
for (let i = 0; i < edges.length; i++) {
  const [u, v] = edges[i].split(",");
  n = Math.max(n, parseInt(u), parseInt(v));
}

const matrix = createMatrix(n + 1);
const list = new Array(n + 1).fill(null).map(() => []);
for (let i = 0; i < edges.length; i++) {
  const [u, v] = edges[i].split(",");
  matrix[u][v] = 1;
  list[u].push(v);
}

document.write("Adjacency matrix:");
document.write("</br>");
for (let i = 1; i <= n; i++) {
  document.write(matrix[i].join(" "));
document.write("</br>");
}

document.write("</br>");
document.write("Adjacency list:");
document.write("</br>");
for (let i = 1; i <= n; i++) {
  document.write(`${i}: ${list[i].join(", ")}`);
document.write("</br>");
}
```

OUTPUT:


hari.pdf
Graph Matrix and List.
DSA practical
abExDSA/index.html
Courses
Documentation
127.0.0.1:5500 says
Enter the edges of the graph, separated by spaces:
1,2 2,3 3,4 4,1
OK
Cancel


WhatsApp
Mukesh Choudhari.pdf
Graph Matrix
127.0.0.1:5500/LabExDSA/index.html
ImpWebs
Social Accounts
Applications Comp...
Courses
HolidayHW
Adjacency matrix:
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
0 1 0 0 0
Adjacency list:
1: 2
2: 3
3: 4
4: 1

Q12. Write a Program to find the element in an array using Binary Search

Solution:

```
function binarySearch(arr, val) {
  let start = 0;
  let end = arr.length - 1;

  while (start <= end) {
    let mid = Math.floor((start + end) / 2);

    if (arr[mid] === val) {
      return mid;
    } else if (arr[mid] < val) {
      start = mid + 1;
    } else {
      end = mid - 1;
    }
  }

  return -1;
}

const arr = [1, 3, 5, 7, 9];
const val = 5;

console.log(binarySearch(arr, val)); // outputs 2
```

OUTPUT:

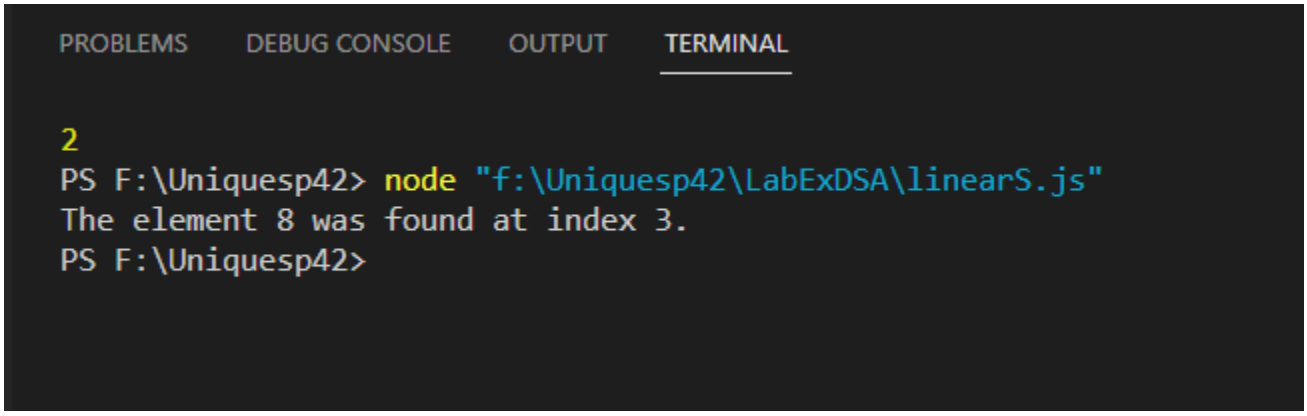
```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL

PS F:\Uniquesp42> node "f:\Uniquesp42\LabExDSA\Binary.js"
2
PS F:\Uniquesp42>
```

Q13. Write a Program to find the element in an array using Linear Search

Solution:

```
function linearSearch(arr, target) {
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] === target) {
      return i;
    }
  }
  return -1;
}

const array = [3, 6, 1, 8, 2, 10];
const targetElement = 8;
const index = linearSearch(array, targetElement);

if (index !== -1) {
  console.log(`The element ${targetElement} was found at index ${index}.`);
} else {
  console.log(`The element ${targetElement} was not found in the array.`);
}
```

OUTPUT:

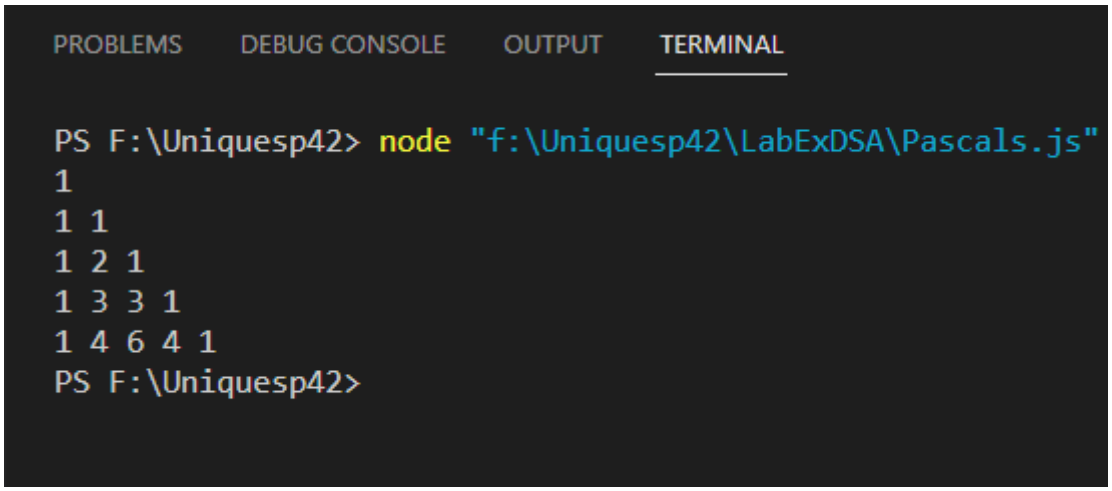


Q14. Write a Program to implement the following .Print Pascal's triangle for n=5

Solution:

```
function pascalsTriangle(n) {
  const triangle = [];
  for (let i = 0; i < n; i++) {
    const row = [];
    for (let j = 0; j <= i; j++) {
      if (j === 0 || j === i) {
        row.push(1);
      } else {
        const prevRow = triangle[i - 1];
        row.push(prevRow[j - 1] + prevRow[j]);
      }
    }
    triangle.push(row);
  }
  for (let i = 0; i < n; i++) {
    console.log(triangle[i].join(' '));
  }
}
pascalsTriangle(5);
```

OUTPUT:

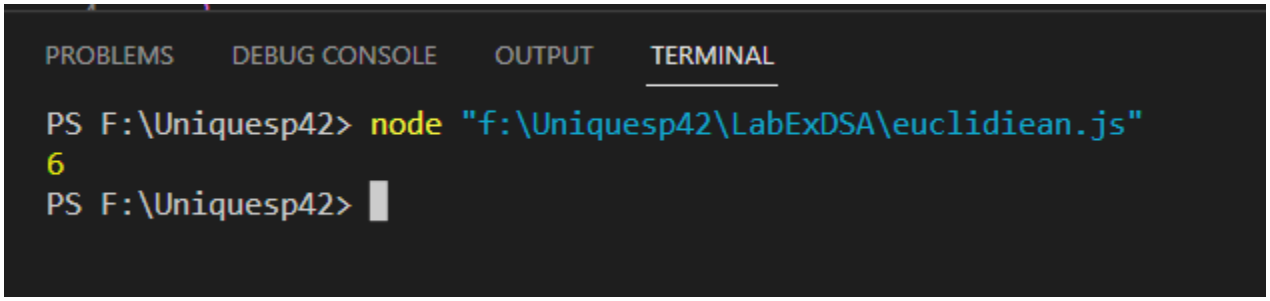


Q15. Write a Program to implement the following GCD of two numbers using Euclidean Algorithm.

Solution:

```
function gcd(a, b) {
  if (a < b) {
    [a, b] = [b, a];
  }

  while (b !== 0) {
    const temp = b;
    b = a % b;
    a = temp;
  }

  return a;
}

const a = 84;
const b = 18;
const result = gcd(a, b);
console.log(result); // Output: 6
```

OUTPUT:

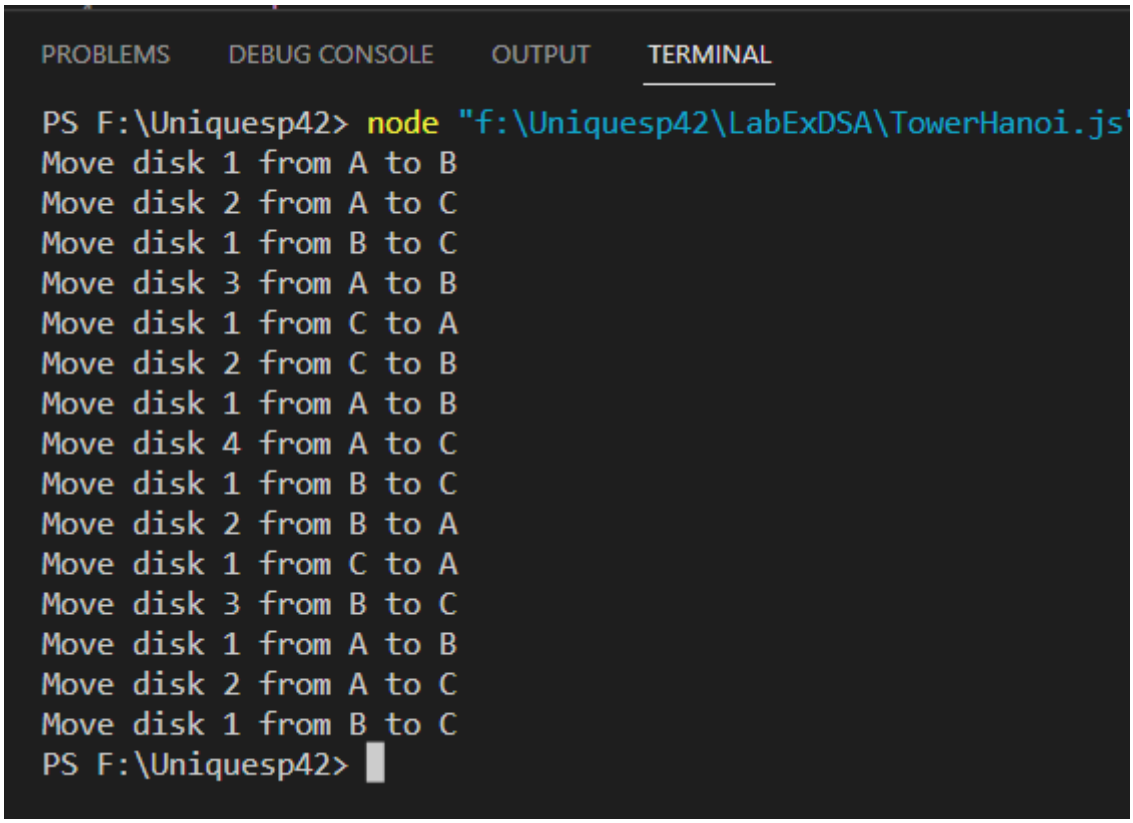


Q16. Write a program to implement tower of Hanoi where number of disks=4

Solution:

```
function towerOfHanoi(n, source, destination, auxiliary) {
  if (n === 1) {
    console.log(`Move disk 1 from ${source} to ${destination}`);
    return;
  }

  towerOfHanoi(n - 1, source, auxiliary, destination);
  console.log(`Move disk ${n} from ${source} to ${destination}`);
  towerOfHanoi(n - 1, auxiliary, destination, source);
}

towerOfHanoi(4, 'A', 'C', 'B');
```

OUTPUT:



Q17. Write a program to implement Fibonacci series till N

Solution:

```
function fibonacciSeries(n) {
  if (n < 1) {
    return [];
  }
  if (n === 1) {
    return [0];
  }
  let series = [0, 1];
  while (series.length < n) {
    let len = series.length;
    series.push(series[len - 1] + series[len - 2]);
  }
  return series;
}

console.log(fibonacciSeries(10));
```

OUTPUT: