PRACTICAL JOURNAL

ON

IT11L: Data Structure and Algorithms

SUBMITTED BY

Jadhav Gaurav Tukaram

Seat No:-26748

SAVITRIBAI PHULE PUNE UNIVERSITY (SPPU)

IN PARTIAL FULFILLMENT OF DEGREE FOR

MASTER OF COMPUTER APPLICATION

DR. D. Y. PATIL UNITECH SOCIETY'S

DR. D. Y. PATIL INSTITUTE OF MANAGEMENT AND RESEARCH,

PIMPRI, PUNE-18

Academic Year 2020-21

Dr. D. Y. Patil Unitech Society's
Dr. D.Y. PATIL INSTITUTE OF MANAGEMENT & RESEARCH,
Sant Tukaram Nagar, Pimpri, Pune-411018, Maharashtra, India.
(Approved by All India Council for Technical Education & Recognized by the Savitribai Phule Pune University)

Date: 15.05.21

CERTIFICATE

This is to certify that Mr. **Jadhav Gaurav Tukaram** has successfully completed the Practicals on IT 11 L : Data Structure and Algorithms as a partial fulfilment of their ***Master of Computer Application*** **(Sem-I)** under the curriculum of **Savitribai Phule Pune University(SPPU), Pune** for the academic year 2020-21.

Mr.Amit Shrivastava	Ms.Swati Narkhede	Dr. Shikha Dubey
Subject Incharge	**Course Coordinator**	**MCA,HOD**

TABLE OF CONTENTS

1. STACK implementation using Array with PUSH, POP, etc. operations.

```
class Stack {
    constructor(){
        this.data = [];
        this.top = 0;
    }
    push(element) {
      this.data[this.top] = element;
      this.top = this.top + 1;
    }
   length() {
      return this.top;
   }
   peek() {
      return this.data[this.top-1];
   }
   isEmpty() {
     return this.top === 0;
   }
   pop() {
    if( this.isEmpty() === false ) {
       this.top = this.top -1;
       return this.data.pop(); // removes the last element
     }
   }
   print() {
      var top = this.top - 1; // because top points to index where new    element to be inserted
      while(top >= 0) { // print upto 0th index
          console.log(this.data[top]);
           top--;
```

```
    }
  }

}
let s=new Stack();
console.log("Before inserting data top value",s.top);
console.log("After Pushing elements into stack");
s.push(11);// 0th
s.push(21);//1st
s.push(31);//2 nd
s.print();
console.log("After inserting data top value",s.top);
s.pop()
console.log("After Poping elements from stack");
s.print();
console.log("The length of stack",s.length());
```

OUTPUT:-

```
Before inserting data top value 0
After Pushing elements into stack
31
21
11
After inserting data top value 3
After Poping elements from stack
21
11
The length of stack 2.
```

2. Reverse a string using stack (e.g., input string MCA output: ACM).

```
class Stack {
  constructor(){
    this.data = [];
    this.top = 0;
  }
  push(element) {
   this.data[this.top] = element;
```

```
    this.top = this.top + 1;
   }
  length() {
    return this.top;
  }
  peek() {
    return this.data[this.top-1];
  }
  isEmpty() {
   return this.top === 0;
  }
  pop() {
   if( this.isEmpty() === false ) {
     this.top = this.top -1;
     return this.data.pop(); // removes the last element
    }
  }


}
let s=new Stack();
function Reverse(abc)
 {
   for(i=0;i<=abc.length;i++)
   {
     s.push(abc[i]);
   }
   let reverseString=" ";
   while(s.length()>0)
   {
     reverseString+=s.pop();
```

```
  }
  console.log("Reverse of ",abc,"is:", reverseString);
 }
 Reverse("MCA ");
```

OUTPUT:-

```
Reverse of  MCA  is:  undefined Y ACM
```

3. Check for balanced parentheses by using Stacks e.g. Input: exp = "[()] {}{[() ()]()}" Output: Balanced Input: exp = "[(])" Output: Not Balanced.

```
class Stack {
   constructor(){
      this.data = [];
      this.top = 0;
   }
   push(element) {
     this.data[this.top] = element;
     this.top = this.top + 1;
   }
  length() {
    return this.top;
  }
  isEmpty() {
   return this.top === 0;
  }
  pop() {
   if( this.isEmpty() === false ) {
     this.top = this.top -1;
     return this.data.pop();
   }
  }
    isBalanced(str){
```

```
    let map={
      '(':')',
      '{':'}',
      '[':']'
    }

let flag =0;
    for(let i=0;i<str.length;i++){
      var item=str[i];
      if(map[item]){
          this.push(str[i]);
      }
      else {
        let bracket=this.pop();
        if(str[i]!= map[bracket])
        flag=1;
        else
          flag=0;
      }
    }
    if(flag==1 && this.length!==0){
      console.log("Not Balanced");
    }
    if(flag==0)
      console.log("Balanced");
  }
}
let s=new Stack();
console.log("Input: [()]{}{[()()]()}");
s.isBalanced("[()]{}{[()()]()}");
```

```
console.log("Input: [(])")

s.isBalanced("[(])");
```

OUTPUT:-

```
Input: [()]{}{[()()]()}
Balanced
Input: [(])
Not Balanced
```

4. Linear Queue implementation using Array with Enqueue, Dequeue, etc operations.

```
class Queue
{

	constructor()
	{
		this.items = [];
	}



enqueue(element)
{

	this.items.push(element);
}
dequeue()
{

	if(this.isEmpty())
		return "Underflow";
	return this.items.shift();
}
// front function
front()
{
	// returns the Front element of
	// the queue without removing it.
	if(this.isEmpty())
		return "No elements in Queue";
	return this.items[0];
}
// isEmpty function
isEmpty()
{
	// return true if the queue is empty.
	return this.items.length == 0;
}
```

```
// printQueue function
printQueue()
{
        var str = "";
        for(var i = 0; i < this.items.length; i++)
                str += this.items[i] +" ";
        return str;
}

}
// creating object for queue class
var queue = new Queue();


// Testing dequeue and pop on an empty queue
// returns Underflow
console.log("When queue is empty",queue.dequeue());

// returns true
console.log(queue.isEmpty());

// Adding elements to the queue
// queue contains
queue.enqueue(111);
queue.enqueue(222);
queue.enqueue(333);
queue.enqueue(444);
queue.enqueue(555);
queue.enqueue(666);
queue.enqueue(777);
// returns 10
console.log("The front element in a queue",queue.front());
console.log("Afer adding elements in the queue",queue.printQueue());
// removes 10 from the queue
// queue contains [20, 30, 40, 50, 60]
console.log("Removing 111 from queue",queue.dequeue());
console.log("After removing 111  in the queue",queue.printQueue());
// returns 20
console.log("The new front element in the queue",queue.front());
```

OUTPUT:-

```
When queue is empty Underflow
true
The front element in a queue 111
After adding elements in the queue 111 222 333 444 555 666 777
Removing 111 from queue 111
After removing 111  in the queue 222 333 444 555 666 777
The new front element in the queue 222
```

5. Circular Queue implementation using Array with Enqueue, Dequeue, etc operations.

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

  enqueue(element)

  {
    if(this.length>=this.size)
          console.log("full");
      //rear=(rear+1)%size;


      this.data[(this.rear+1)%this.size] =element;// data[0]=10
      this.length++;
      console.log("Elements inserted in the Circular Queue",this.data)

  }
  getfront()
  {
    if(this.isEmpty())
    {
      console.log("no element in circular queue");
    }
    return this.data[this.front%this.size]//data[0]
  }
  dequeue()
  {
    if(this.isEmpty())
  console.log("no element");
  const value= this.getfront();
  this.data[this.front%this.size]=null;
  this.front++;
  this.length--;
```

```
  console.log(value);


  }

}
cq=new CircularQueue(5);
cq.enqueue(70);
cq.enqueue(71);
cq.enqueue(72);
cq.enqueue(73);
cq.enqueue(75);
console.log(" elements deleted  in circular queue");
cq.dequeue();
console.log(cq.getfront());
```

OUTPUT:-

```
Elements inserted in the Circular Queue [ 70 ]
Elements inserted in the Circular Queue [ 71 ]
Elements inserted in the Circular Queue [ 72 ]
Elements inserted in the Circular Queue [ 73 ]
Elements inserted in the Circular Queue [ 75 ]
 elements deleted  in circular queue
75
Undefined
```

6. Priority Queue implementation using Array with Enqueue, Dequeue, etc operations.

```
class QElement {
    constructor(element, priority)
    {
        this.element = element;
        this.priority = priority;
    }
}
class PriorityQueue {

    // An array is used to implement priority
    constructor()
    {
        this.items = [];
    }
```

```
    // functions to be implemented
    enqueue(element, priority)
{
    // creating object from queue element
    var qElement = new QElement(element, priority);
    var contain = false;

    // iterating through the entire
    // item array to add element at the
    // correct location of the Queue
    for (var i = 0; i < this.items.length; i++) {
        if (this.items[i].priority > qElement.priority) {
            // Once the correct location is found it is
            // enqueued
            this.items.splice(i, 0, qElement);
            contain = true;
            break;
        }
    }

    // if the element have the highest priority
    // it is added at the end of the queue
    if (!contain) {
        this.items.push(qElement);
    }
}
    dequeue()
{
    // return the dequeued element
    // and remove it.
    // if the queue is empty
```

```
    // returns Underflow
    if (this.isEmpty())
        return "Underflow";
    return this.items.shift();
}
    front()
{
    // returns the highest priority element
    // in the Priority queue without removing it.
    if (this.isEmpty())
        return "No elements in Queue";
    return this.items[0];
}
   rear()
{
    // returns the lowest priorty
    // element of the queue
    if (this.isEmpty())
        return "No elements in Queue";
    return this.items[this.items.length - 1];
}
isEmpty()
{
    // return true if the queue is empty.
    return this.items.length == 0;
}
printPQueue()
{
    var str = "";
    for (var i = 0; i < this.items.length; i++)
        str += this.items[i].element + " ";
    return str;
```

```
}

}
var priorityQueue = new PriorityQueue();

// testing isEmpty and front on an empty queue
// return true
console.log(priorityQueue.isEmpty());

// returns "No elements in Queue"
console.log(priorityQueue.front());

// adding elements to the queue
priorityQueue.enqueue("Nilesh", 2);
priorityQueue.enqueue("Ganesh", 1);
priorityQueue.enqueue("Suresh", 1);
priorityQueue.enqueue("Ramesh", 2);
priorityQueue.enqueue("Jignesh", 3);

// prints [Ganesh Suresh Nilesh Ramesh Jignesh ]
console.log(priorityQueue.printPQueue());

// prints Ganesh
console.log(priorityQueue.front().element);

// pritns Jignesh
console.log(priorityQueue.rear().element);

// removes Ganesh
// priorityQueue contains
// [Suresh Nilesh Ramesh Jignesh]
console.log(priorityQueue.dequeue().element);
```

```
// Adding another element to the queue
priorityQueue.enqueue("Nilesh", 2);


// prints [Suresh Nilesh Ramesh Nilesh Jignesh]
console.log(priorityQueue.printPQueue());
```

OUTPUT:-

```
true
No elements in Queue
Ganesh Suresh Nilesh Ramesh Jignesh
Ganesh
Jignesh
Ganesh
Suresh Nilesh Ramesh Nilesh Jignesh
```

7. Reverse stack using queue.

```
function queue()
{
 this.data=[];
 this.rear=0;
 this.front=0;
 this.enqueue=enqueue;
 this.MaxSize=10;
 this.print=print;
 this.length=length;
 this.isEmpty=isEmpty;
 this.dequeue=dequeue;
 this.reverseS=reverseS;
}

function enqueue(element)
{
 if(this.rear<this.MaxSize)
 {
```

```
    this.data[this.rear++]=element; //data[0]=10
  }
}
function dequeue()
{
  if(this.isEmpty()==false)
  {
    this.rear=this.rear-1;
    return this.data.pop();
  }
}



function length()
{
  return this.rear;
}
function isEmpty()
{
  return this.rear=0;
}


function print()
{
  this.rear--;
  for(let i=this.rear;i>=0;i--)
  {
    console.log(this.data[i]);
  }
```

```
}


function reverseS()
{
   for(let i=0;i<=this.rear;i++)
  {
    console.log(this.data[i]);
  }
}
var s = new queue();
s.enqueue(10);
s.enqueue(20);
s.enqueue(30);
console.log("Stack elements");
s.print();
console.log("Stack reverse :")
s.reverseS();
```

Output:-

```
Stack elements
30
20
10
Stack reverse :
10
20
30
```

8. Demonstrate singly linked list operations.

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

  }
}
class LinkedList
{
```

```
constructor()
{
  this.head=null;
  this.size=0;

}
add(element)
{
  var temp=new Node(element);
  var temp1;
  if(this.head==null)
  this.head=temp;
  else
  {
    temp1= this.head;
    while(temp1.next)
    {
      temp1=temp1.next
    }
    //add Node
    temp1.next=temp;

  }
  this.size++;

}
// insertat location
  insertAt(element, index)
{
 if(index>0 && index>this.size)
 return false;
 else
 {
 var temp=new Node(element);

 var temp1, temp2;
 temp1=this.head;
 if (index==0)
 {
   this.next=this.head;
   this.head=temp;

 }
 else
 {
   temp1=this.head;
   var it=0;
   while(it<index)
   {
     it++;
```

```
      temp2=temp1;
      temp1=temp1.next;


    }
    temp.next = temp1;
       temp2.next = temp;
  }



 this.size++;
 }

}

 //PrintList
 PrintList()
 {
   var temp1=this.head;
   while(temp1)
   {
     console.log(temp1.element);
     temp1=temp1.next;

   }

 }

}
 ll =new LinkedList();
 ll.add(11);
 ll.add(21);
 ll.add(31);
 console.log("");
 ll.insertAt(51,1);
 console.log(ll.insertAt(101,6));
 //ll.insertAt(200,0);

 ll.PrintList();
```

OUTPUT:-

```
false
11
51
21
31
```

9. Demonstrate doubly linked list operations.

```
class node
{
  constructor(element)
  {
    this.element=element;
    this.next=null;
    this.prev=null;
  }
}
class linkedlist
{
  constructor()
  {
    this.size=null;
    this.head=null;
  }
  insertatend(element)
  {
    var temp=new node(element);
    var temp1;
    if(this.head==null)
    {
      this.head=temp;
      temp.prev=null;
      temp.next=null;
    }
    else
    {
      temp1=this.head;
      while(temp1.next)
```

```
    {
      temp1=temp1.next;
    }

    temp1.next=temp;
    temp.prev=temp1;
    temp.next=null;
  }
  this.size++;

}
insertatbeg(element)
{
  var temp=new node(element);
  if(this.head==null)
  {
    this.head=temp;
    temp.prev=null;
    temp.next=null;
  }
  else
  {
  var temp1=this.head;
  this.head=temp;
  temp.prev=null;
  temp.next=temp1;
  }
  this.size++;
}
insertatpos(element,pos)
{
```

```
  var temp=new node(element);
  var temp1=this.head;
  var i=1;
  var temp2=temp1;
  if(this.head==null)
  {
   console.log("No elements");
  }
  else
  {
   while(i!=pos)
   {
    temp2=temp1;
    temp1=temp1.next;
    i++;
   }
   temp2.next=temp;
   temp.prev=temp2;
   temp.next=temp1
   }
   this.size++;
}
deleteatbeg()
{
  var temp1;
  var temp2;
  temp1=this.head;
  temp2=temp1.next;
  this.head=temp2;
  temp1.next==null;
  this.size--;
```

```
}
deleteatend()
{
 var temp1=this.head;
 var temp2=temp1;
 if(this.head==null)
 {
  console.log("Linked list is empty1");
 }
 else
 {
  while(temp1.next!=null)
  {
   temp2=temp1;
   temp1=temp1.next;

  }
  temp2.next=null;
  temp1=null;
 }
 this.size--;
}
deleteatpos(pos)
{
 var temp1=this.head;
 var i=1;
 var temp2=temp1;
 if(this.head==null)
 {
  console.log("No elements");
 }
```

```
  else
  {
    while(i!=pos)
    {
      temp2=temp1;
      temp1=temp1.next;
      i++;
    }
    temp2.next=temp1.next;
     temp1=null;

    }
    this.size--;
 }
 display()
 {
  var temp1=this.head;
  while(temp1)
  {
    console.log(temp1.element);
    temp1=temp1.next;
  }
}
searchs(pos)
{
  var temp1=this.head;
  while(temp1) {
        if(temp1.element == pos) {
           var i=1;
        }
```

```
            temp1 = temp1.next;
        }
        if(i==1)
        {
       console.log("Element found",pos);
        }
        else
        {
          console.log("Element not found",pos);
        }
 }
}
var s=new linkedlist();
s.insertatbeg(10);
s.insertatbeg(20);
s.insertatbeg(60);
s.insertatend(70)
s.insertatend(40);
s.insertatend(50);
s.insertatpos(30,2);
s.insertatpos(100,3)
s.deleteatbeg();
s.deleteatpos(2);
s.deleteatend();
s.searchs(40);
s.searchs(80);
console.log(s.display());
```

Output:-

```
Element found 40
Element not found 80
30
20
10
70
```

```
40
undefined
```

10. Implement Stack using Linked List.

```
class Node {
  constructor(elm){
    this.element = elm;
    this.next = null;
  }
}
class stackUsingLL {
  constructor(){
    this.length = 0;
    this.head = null;
  }

   push(elm){
    let node = new Node(elm),
    current;
    current = this.head;
    node.next = current;
    this.head = node;
    this.length++;
  }
   pop(){
    let current = this.head;
    if(current){
      let elm = current.element;
      current = current.next;
      this.head = current;
```

```
    this.length--;
    return elm;
  }
  return null;
}
print(){
    let current = this.head;
    while(current){
    console.log(current.element);
    current = current.next;
  }
}
peek(){
  if(this.head){
    return this.head.element;
  }
  return null;
}
isEmpty(){
  return this.length === 0;
}


size(){
  return this.length;
}

clear(){
  this.head = null;
  this.length = 0;
```

```
  }
}
let stack = new stackUsingLL();   //creating new instance of Stack
 stack.push(111);
 stack.push(222);
 stack.push(333);
 console.log("stack elements : ");
 stack.print();
 console.log("Peek :",stack.peek());
 console.log("stack is Empty :",stack.isEmpty());
 console.log("stack Size :",stack.size());
 console.log("Pop element :",stack.pop());
 console.log("stack Size after pop :",stack.size());
 stack.clear(); //Clear the stack
 console.log("stack is Empty after clear:",stack.isEmpty());
```

OUTPUT:-

```
stack elements :
333
222
111
Peek : 333
stack is Empty : false
stack Size : 3
Pop element : 333
stack Size after pop : 2
stack is Empty after clear: true
```

11. Implement Binary Search Tree with its operations.

```
class Node {
   constructor(value) {
      this.val = value;
      this.leftChild = null;
      this.rightChild = null;
   }
```

```
}

class BinarySearchTree {
   constructor(root) {
      this.root = new Node(root);
   }
//insert into bst
   insert(temp, newValue) {
      if (temp === null) {
         temp = new Node(newValue);
      } else if (newValue < temp.val) {
        temp.leftChild = this.insert(temp.leftChild, newValue);
      } else {
         temp.rightChild = this.insert(temp.rightChild, newValue);
      }
     // console.log(temp)
      return temp;

   }

   insertBST(newValue) {
      if(this.root==null){
         this.root=new Node(newValue);
         //return;
      }
     else{

      this.insert(this.root, newValue);
     }
   }
```

```
  preOrderPrint(temp) {
     if (temp!==null) {
        console.log(temp.val);
        this.preOrderPrint(temp.leftChild);
        this.preOrderPrint(temp.rightChild);
     }

  }
inOrderPrint(temp) {
     if (temp!==null) {
        this.inOrderPrint(temp.leftChild);
        console.log(temp.val);
        this.inOrderPrint(temp.rightChild);
     }

  }
  postOrderPrint(temp) {
 //if the currentNode IS NOT EQUAL to null
  if (temp!==null) {
     //make recursive call to the left subtree
     this.postOrderPrint(temp.leftChild);
      //make recursive call to the right subtree
     this.postOrderPrint(temp.rightChild);
     //print its value
     console.log(temp.val);
  }
}

 //delete
 remove(val)
```

```
{
   // root is re-initialized with
   // root of a modified tree.
   this.root = this.removeNode(this.root, val);
}

// Method to remove node with a
// given data
// it recur over the tree to find the
// data and removes it
removeNode(node, key)
{
        //to find key element address
   // if the root is null then tree is
   // empty
   if(node === null)
      return null;

   // if data to be delete is less than
   // roots data then move to left subtree
   else if(key < node.val)
   {
      node.leftChild = this.removeNode(node.leftChild, key);
      return node;
   }

   // if data to be delete is greater than
   // roots data then move to right subtree
   else if(key > node.val)
   {
      node.rightChild = this.removeNode(node.rightChild, key);
```

```
    return node;
}

// if data is similar to the root's data
// then delete this node
else
{
    // deleting node with no children (adjustment)
    if(node.leftChild === null && node.rightChild === null)
    {
        node = null;
        return node;
    }

    // deleting node with one children
    if(node.leftChild === null)
    {
        node = node.rightChild;
        return node;
    }

    else if(node.rightChild === null)
    {
        node = node.leftChild;
        return node;
    }

    // Deleting node with two children
    // minumum node of the rigt subtree
    // is stored in aux
    var aux = this.findMinNode(node.rightChild);
```

```
        console.log("aux="+aux.val)
        node.val = aux.val;

        node.rightChild = this.removeNode(node.rightChild, aux.val);

        return node;//return to root
    }

}
findMinNode(node)
{
    // if left of a node is null
    // then it must be minimum node
    if(node.leftChild === null)
        return node;
    else
        return this.findMinNode(node.leftChild);
}

}

var BST = new BinarySearchTree(6);
//console.log("The root val for BST : ", BST.root.val)
BST.insertBST(4);
BST.insertBST(9);
BST.insertBST(5);
BST.insertBST(2);
BST.insertBST(8);
BST.insertBST(12);
BST.insertBST(10);
BST.insertBST(14);
```

```
BST.remove(10)

BST.remove(5)

BST.remove(6)

console.log("========================preOrderPrint================")

BST.preOrderPrint(BST.root);

console.log("========================inOrderPrint================")

BST.inOrderPrint(BST.root)

console.log("========================postOrderPrint================")

BST.postOrderPrint(BST.root)
```

Output:-

```
aux=8
========================preOrderPrint================
8
4
2
9
12
14
========================inOrderPrint================
2
4
8
9
12
14
========================postOrderPrint================
2
4
14
12
9
8
```

12. Implementation of DFS Graph traversals.

```
class Graph {
   // defining vertex array and adjacent list
   constructor(noOfVertices)
   {
      this.noOfVertices = noOfVertices;
      this.AdjList = new Map();
   }
   addVertex(v)
```

```
{
   // initialize the adjacent list with a null array
   this.AdjList.set(v, []);
}
addEdge(v, w)
{
   // get the list for vertex v and put the vertex w denoting edge between v and w
   this.AdjList.get(v).push(w);

   // Since graph is undirected, add an edge from w to v also
   this.AdjList.get(w).push(v);
}
printGraph()
{
   // get all the vertices
   var get_keys = this.AdjList.keys();

   // iterate over the vertices
   for (var i of get_keys)
{
      // get the corresponding adjacency listfor the vertex
      var get_values = this.AdjList.get(i);
      var data = "";

      // iterate over the adjacency list
      // concatenate the values into a string
      for (var j of get_values)
         data=data + j +""

      // print the vertex and its adjacency list
```

```
        console.log(i + " -> " + data);
    }
}



  dfs(v) {
        let visited = [];
        let keys=this.AdjList.keys();
        for(let v of keys)
        {
          visited[v]=false;
        }
        this.dfsRecursion(v, visited);
    }
    dfsRecursion(v, visited)
    {
        visited[v] = true;
        console.log("visited vertex="+v);

        let edgelist = this.AdjList.get(v);

        for (var node of edgelist) {
            if (!visited[node]) this.dfsRecursion(node, visited);
        }
    }
}

g = new Graph(5);
var vertices = [ 'A', 'B', 'C', 'D', 'E' ];
```

```
// adding vertices
for (var i = 0; i < vertices.length; i++) {
   g.addVertex(vertices[i]);
}



// adding edges


g.addEdge('A','B');
g.addEdge('A','C');
g.addEdge('B','D');
g.addEdge('C','E');


console.log("============dfs==================")


g.dfs('A')
//g.printGraph()
```

Output:-

```
============dfs==================
visited vertex=A
visited vertex=B
visited vertex=D
visited vertex=C
visited vertex=E
```

13. Implementation of BFS Graph traversals.

```
class Graph {
   // defining vertex array and adjacent list
   constructor(noOfVertices)
   {
      this.noOfVertices = noOfVertices;
      this.AdjList = new Map();
   }
   addVertex(v)
```

```
{
   // initialize the adjacent list with a null array
   this.AdjList.set(v, []);
}
addEdge(v, w)
{
   // get the list for vertex v and put the vertex w denoting edge between v and w
   this.AdjList.get(v).push(w);

   // Since graph is undirected, add an edge from w to v also
   this.AdjList.get(w).push(v);
}
printGraph()
{
   // get all the vertices
   var get_keys = this.AdjList.keys();

   // iterate over the vertices
   for (var i of get_keys)
{
      // get the corresponding adjacency listfor the vertex
      var get_values = this.AdjList.get(i);
      var data = "";

      // iterate over the adjacency list
      // concatenate the values into a string
      for (var j of get_values)
         data=data + j +""

      // print the vertex and its adjacency list
```

```
        console.log(i + " -> " + data);
    }
}

 bfs(v)
{
 var q = [];
 let visited=[];
 q.push(v); // add to back of queue
 let keys=this.AdjList.keys();
 for(let v of keys)
 {
   visited[v]=false;
 }
 while (q.length > 0)
 {
 var element = q.shift();
 visited[element]=true// remove from front of queue
 //console.log(v)
// /if (v==undefined)
 console.log("Visited vertex: " + element);
 let edgelist=this.AdjList.get(element)
for (let node of edgelist) {
            if (!visited[node]) {
                visited[node] = true;
                q.push(node);
            }
   }
}


}
```

```
}

g = new Graph(5);
var vertices = [ 'A', 'B', 'C', 'D', 'E' ];

// adding vertices
for (var i = 0; i < vertices.length; i++) {
   g.addVertex(vertices[i]);
}



// adding edges

g.addEdge('A','B');
g.addEdge('A','C');
g.addEdge('B','D');
g.addEdge('C','E');
console.log("============bfs==================")

g.bfs('A')
```

Output:-

```
============bfs==================
Visited vertex: A
Visited vertex: B
Visited vertex: C
Visited vertex: D
Visited vertex: E
```

14. Implementation of Hashing.

```
//hash table
function hash(key,size)
{
```

```
 //let hashkey=0;
/* for(let i=0;i<key.length;i++)
 {
   key=key.charCodeAt(i)


 }*/
 let h=key%size
 //console.log(h)
 return h
}
class  HashTable
{
  constructor()
  {
    this.size=100;
    this.buckets=Array(this.size)
   // console.log(this.buckets.length)
    for(let i=0;i<this.buckets.length;i++)
    {
      this.buckets[i]=new Map();

    }
  }

  insert(key,value)
  {
    let idx=hash(key,this.size)
    //console.log(idx)
 this.buckets[idx].set(key,value)
 console.log(this.buckets[idx])
```

```
 }
 remove(key)
 {
   let index=hash(key,this.size)
   let deleted=this.buckets[index].get(key)
   this.buckets[index].delete(key)

 }
 search(key)
 {
   let index=hash(key,this.size)
   console.log(this.buckets[index].get(key))
   return this.buckets[index].get(key)
 }
}

 let hashtable=new HashTable()
hashtable.insert(1,10)
hashtable.insert(45,10)
//hashtable.insert(25,10)
//hashtable.insert(66,10)
//hashtable.insert(56,10)
//hashtable.insert('yas','kid')
//console.log(hashtable)
hashtable.search(45)
//hash(23,10)
```

Output:-

```
Map { 1 => 10 }
Map { 45 => 10 }
10
```

15. Implementation of Linear Search using Brute Force technique.

```
function linearSearch(arr, key){
   for(let i = 0; i < arr.length; i++){
      if(arr[i] === key){
         return i
      }
   }
   return -1
}
arr=[1,3,5,8,9]
key=5
console.log(linearSearch(arr,key))
```

Output:-

2

16. Implementation of Rain Terraces using Brute Force technique.

```
function maxWater(arr, n)
   {

      // To store the maximum water
      // that can be stored
      let res = 0;

      // For every element of the array
      // except first and last element
      for(let i = 1; i < n - 1; i++)
      {

         // Find maximum element on its left
         let left = arr[i];
         for(let j = 0; j < i; j++)
         {
            left = Math.max(left, arr[j]);
```

```
        }

        // Find maximum element on its right
        let right = arr[i];
        for(let j = i + 1; j < n; j++)
        {
            right = Math.max(right, arr[j]);
        }

        // Update maximum water value
        res += Math.min(left, right) - arr[i];
    }
    return res;
}

let arr = [ 0, 1, 0, 2, 1, 0,
          1, 3, 2, 1, 2, 1 ];
let n = arr.length;

console.log(maxWater(arr,n));
```

Output:

6

17. Implementation of Recursive Staircase using Brute Force technique.

```
function climbStairs(n)  {
  const A =[1,1];
  if(n>1){
      for(let i = 2; i <= n ; i++){
        A[i] = A[i-1] + A[i-2];
      }
```

```
  };
  return A.pop()
}
console.log(climbStairs(7))
```

Output:

```
21
```

18. Implementation of Maximum Sub array using Brute Force technique.

/* Maximum subarray problem is the method to find the contiguous subarray within a one-dimensional array of numbers which has the largest sum.

The problem was originally proposed by Ulf Grenander of Brown University in 1977, as a simplified model for maximum likelihood estimation of patterns in digitized images.

Time complexity for Brute-Force method is O(n^2)*/

```
function maxisubarry()
{

//var arr= [13,-3,-25,20,-3,-16,-23,18,20,-7,12,-5,-22,15,-4,7];
var arr=[0,1,2,-2,3,2]
   len=arr.length;
  var sum = 0;
   var max = 0;

   // i is starting index for the sub array
   // subarray of every size starting from i
   // will be used to calculate the sum
   for(i = 0; i < len; i++) {
     sum = 0;

      // j represents size of subarray
```

```
      // starting from i - subarray of size 1
      // to len - 1 - biggest subarray possible
      // starting from i
      for(j = i; j < len; j++) {
         sum += arr[j];

         // compare sum till now againt
         // the max till now
         if(sum > max)
            max = sum;
      }
   }

   console.log( "max value is :"+ max )


}
maxisubarry()
```

Output:-

```
max value is :6
```

19. Implementation of Prim's using Greedy Algorithm.

```
/*
  create adjacency matrix for use in prims algorithm
  note: we could improve the running time of prims algorithm by
  implementing a priority queue data structure instead of a matrix
*/
function createAdjMatrix(V, G) {
```

```
  var adjMatrix = [];

  // create N x N matrix filled with 0 edge weights between all vertices
  for (var i = 0; i < V; i++) {
    adjMatrix.push([]);
    for (var j = 0; j < V; j++) { adjMatrix[i].push(0); }
  }
  console.log(G)

  // populate adjacency matrix with correct edge weights
  console.log("graph length="+G.length)
  for (var i = 0; i < G.length; i++) {


    adjMatrix[G[i][0]][G[i][1]] = G[i][2];  //[0] means start vertex and [1] end vertex and [2] means
weights
    //console.log(adjMatrix[G[i][0]][G[i][1]])
    adjMatrix[G[i][1]][G[i][0]] = G[i][2];
  }

  return adjMatrix;

}

function prims(V, G) {

  // create adj matrix from graph
  var adjMatrix = createAdjMatrix(V, G);

  // arbitrarily choose initial vertex from graph
  var vertex = 0;
```

```
// initialize empty edges array and empty MST
var MST = [];
var edges = [];
var visited = [];
var minEdge = [null,null,Infinity];

// run prims algorithm until we create an MST
// that contains every vertex from the graph
while (MST.length !== V-1) {

 // mark this vertex as visited
 visited.push(vertex);

 // add each edge to list of potential edges
 for (var r = 0; r < V; r++) {
  if (adjMatrix[vertex][r] !== 0) {
   edges.push([vertex,r,adjMatrix[vertex][r]]);
  }
 }

 // find edge with the smallest weight to a vertex
 // that has not yet been visited
 for (var e = 0; e < edges.length; e++) {
  if (edges[e][2] < minEdge[2] && visited.indexOf(edges[e][1]) === -1) {
   minEdge = edges[e];
  }
 }

 // remove min weight edge from list of edges
 edges.splice(edges.indexOf(minEdge), 1);
```

```
    // push min edge to MST
    MST.push(minEdge);

    // start at new vertex and reset min edge
    vertex = minEdge[1];
    minEdge = [null,null,Infinity];

  }

  return MST;

}

// graph vertices are actually represented as numbers
// like so: 0, 1, 2, ... V-1
var a = 0, b = 1, c = 2, d = 3, e = 4, f = 5;

// graph edges with weights
// diagram of graph is shown above
var graph = [
  [a,b,2],
  [a,c,3],
  [b,d,3],
  [b,c,5],
  [b,e,4],
  [c,e,4],
  [d,e,2],
  [d,f,3],
  [e,f,5]
];
```

```
// pass the # of vertices and the graph to run prims algorithm

console.log(prims(6, graph));
```

Output:-

```
[
  [ 0, 1, 2 ], [ 0, 2, 3 ],
  [ 1, 3, 3 ], [ 1, 2, 5 ],
  [ 1, 4, 4 ], [ 2, 4, 4 ],
  [ 3, 4, 2 ], [ 3, 5, 3 ],
  [ 4, 5, 5 ]
]
graph length=9
[ [ 0, 1, 2 ], [ 0, 2, 3 ], [ 1, 3, 3 ], [ 3, 4, 2 ], [ 3, 5, 3 ] ]
```

20. Implementation of Kruskal's algorithm using Greedy Algorithm.

```
/*

Kruskal's Minimum Spanning Tree Algorithm using DSU


*/


class DSU {
	constructor() {
		this.parents = [];
	}
	find(x) {
		if(typeof this.parents[x] != "undefined") {
			if(this.parents[x]<0) {
				return x; //x is a parent
			} else {
				//recurse until you find x's parent
				return this.find(this.parents[x]);
			}
		} else {
			// initialize this node to it's on parent (-1)
			this.parents[x]=-1;
```

```
                return x; //return the index of the parent
            }
        }
        union(x,y) {
            var xpar = this.find(x);
            var ypar = this.find(y);
            if(xpar != ypar) {
                // x's parent is now the parent of y also.
                // if y was a parent to more than one node, then
                // all of those nodes are now also connected to x's parent.
                this.parents[xpar]+=this.parents[ypar];
                this.parents[ypar]=xpar;
                return false;
            } else {
                return true; //this link creates a cycle
            }
        }
        console_print() {
            console.log(this.parents);
        }
}

function find_mst(nodes,costs) {
    var mst = []; //the order of the path we need to take
    // 1. Sort all the edges in non-decreasing order of their weight.
    nodes.sort(function(a,b){
        return costs[nodes.indexOf(a)]-costs[nodes.indexOf(b)];
    });
    costs.sort(function(a,b){ return a-b });
    // 2. Pick the smallest edge.
```

```
	var total_min_cost = 0;


	var dsu = new DSU();
	for(var c=0; c<costs.length; c++) {
		if(dsu.find(nodes[c][0]) != dsu.find(nodes[c][1])) {
			// If cycle is not formed, include this edge.
			dsu.union(nodes[c][0],nodes[c][1]);
			total_min_cost+=costs[c];
			mst.push(nodes[c][0]+"->"+nodes[c][1]);
		}// Else, discard it.
	}
	console.log("total_min_cost = ",total_min_cost);
	console.log("path = ",mst);
}

var nodes = [[0,1], [0, 4], [1, 4], [1, 3], [3, 2], [4, 2]];
var costs = [5, 100, 20, 70, 9, 17];
console.log("Given path=",nodes)

console.log("costs=",costs)
find_mst(nodes,costs);
```

Output:-

```
Given path= [ [ 0, 1 ], [ 0, 4 ], [ 1, 4 ], [ 1, 3 ], [ 3, 2 ], [ 4, 2 ] ]
costs= [ 5, 100, 20, 70, 9, 17 ]
total_min_cost =  51
path =  [ '0->1', '3->2', '4->2', '1->4' ]
```

21. Implementation of Binary Search using Divide and Conquer.

```
function binarySearch(sortedList, searchvalue){
   let Left = 0;
   let Right = sortedList.length - 1;
```

```
    while (Left <= Right) {
        let middle = Math.floor((Left + Right) / 2);

        if (sortedList[middle] === searchvalue) {
            // found the searchvalue
            return middle;
        } else if (sortedList[middle] < searchvalue) {
            // continue searching to the right
            Left = middle + 1;
        } else {
            // search searching to the left
            Right= middle - 1;
        }
    }
            // serachvalue wasn't found
    return -1;
}

sortedArray=[ 5,10,12,15,20,60,100];
console.log(binarySearch(sortedArray,100));
```

Output:-

6

22. Implementation of Tower of Hanoi.

```
 function toh(n,src,des,aux)
{
  if(n>=1)
  {
    console.log("src="+src);
    console.log("aux="+aux);
```

```
   console.log("des="+des);
   console.log("  ");
   toh(n-1,src,aux,des);
    console.log("src="+src);
   console.log("aux="+aux);
   console.log("des="+des);
   console.log("  ");
   console.log("Move disk from tower",src,"to tower",des);
   toh(n-1,aux,des,src);
    console.log("src="+src);
   console.log("aux="+aux);
   console.log("des="+des);
   console.log("  ");


 }
 return;
}
toh(2,"A","B","C");
```

Output:

```
src=A
aux=C
des=B

src=A
aux=B
des=C

src=A
aux=B
des=C

Move disk from tower A to tower C
src=A
aux=B
des=C

src=A
aux=C
des=B
```

```
Move disk from tower A to tower B
src=C
aux=A
des=B

src=C
aux=A
des=B

Move disk from tower C to tower B
src=C
aux=A
des=B

src=A
aux=C
des=B
```

23. Implementation of LCS using Dynamic Programming.

```
function LCS(str1, str2){
   var m = str1.length
   var n = str2.length
   var dp = [new Array(n+1).fill(0)] //The first line is all 0
   for(var i = 1; i <= m; i++){ //A total of m+1 lines
      dp[i] = [0] //The first column is all 0
      for(var j = 1; j <= n; j++){//A total of n+1 columns
         if(str1[i-1] === str2[j-1]){
            //Note here, the first character of str1 is in the second column, so you have to subtract 1,
and str2 is the same
            dp[i][j] = dp[i-1][j-1] + 1 //Diagonal +1
         } else {
            dp[i][j] = Math.max( dp[i-1][j], dp[i][j-1])
         }
      }
   }

  console.log("length="+dp[m][n])
   //return dp[m][n];
```

```
  return dp;
}
// print an LCS
function printLCS(dp, str1, str2, i, j){
   if (i == 0 || j == 0){
      return "";
   }
   if( str1[i-1] == str2[j-1] ){
      return printLCS(dp, str1, str2, i-1, j-1) + str1[i-1];
   }else{
      if (dp[i][j-1] > dp[i-1][j]){
         return printLCS(dp, str1, str2, i, j-1);
      }else{
         return printLCS(dp, str1, str2, i-1, j);
      }
   }
}
// convert the target string into regular, verify whether it is the LCS of the previous two strings
/*function validateLCS(el, str1, str2){
  var re =  new RegExp( el.split("").join(".*") )
  console.log(el, re.test(str1),re.test(str2))
  return re.test(str1) && re.test(str2)
}*/
//s1="ABCBDAB"
//s2="BDCABA"
s1="ACADB"
s2="CBDA"
var m= s1.length
   var n = s2.length
var c1 = LCS(s1,s2)
//console.log(c1) //4 BCBA、BCAB、BDAB
```

```
    console.log(printLCS(c1,s1,s2,m,n))



//var c2 = LCS("13456778" , "357486782" );

//console.log(c2) //5 34678

//var c3 = LCS("ACCGGTCGAGTGCGCGGAAGCCGGCCGAA" ,"GTCGTTCGGAATGCCGTTGCTCTGTAAA"
);

//console.log(c3) //20 GTCGTCGGAAGCCGGCCGAA
```

Output:-

```
length=2
CA
```

24. Implementation of Regular Expression Matching using Dynamic Programming.

```
const isMatch = (s, p) => {
    const dp = Array.from({ length: s.length+1 }, () => [false]);
    dp[0][0] = true;

    // first row
    for(let i = 1; i <= p.length; i++) {
        if(p[i-1] === '*') {
            dp[0][i] = dp[0][i-2]
        }
        else {
            dp[0][i] = false
        };
    }

    for(let r = 1; r <= s.length; r++) {
        for(let c = 1; c <= p.length; c++) {
            if(p[c-1] === '*') {
```

```
            if(p[c-2] === s[r-1] || p[c-2] === '.') {
                dp[r][c] = dp[r][c-2] || dp[r-1][c];
            } else dp[r][c] = dp[r][c-2];
        } else if(p[c-1] === s[r-1] || p[c-1] === '.') {
            dp[r][c] = dp[r-1][c-1];
        } else dp[r][c] = false;
      }
   }

   console.log(dp[s.length][p.length]);
};

// test case, you can modify it and check if it works
const a = "mississippi"
const b = "mis*is*ip*."
isMatch(a, b)

const c="Maharashtra"
const d="Maha*rash*t"
isMatch(c,d)
```

Output:-

```
true
false
```

25. Implementation of N Queen's problems.

```
var n =4 ;

solveNQ();

function printSolution(board){
  for(var i=0; i<n; i++){
    for(var j=0; j<n; j++){
```

```
    console.log(" "+board[i][j]+" ");
   }
   process.stdout.write("");
  }
process.stdout.write("");
}

function isSafe(board, row, col){

 // Checks the ← direction
 for(var i=0; i<col; i++){
  if (board[row][i] === 1) {
   return false;
  }
 }

 // Checks the ↖ direction
 for(var i=row, j=col; i>=0 && j>=0; i--, j--){
  if (board[i][j] === 1) {
   return false;
  }
 }

 // Checks the ↙ direction
 for(var i=row, j=col; j>=0 && i<n; i++, j--){
  if (board[i][j] === 1){
   return false;
  }
 }

 return true;
```

```
}



function recurseNQ(board, col){
  if(col===n){
      printSolution(board); // <-- print another solution when n==8
    return;
  }

  for(var i=0; i<n; i++)
{
    if(isSafe(board, i, col)){
      board[i][col]=1;

      recurseNQ(board, col+1);
      //if(recurseNQ(board, col+1)===true) //<-- you don't need this
          // return true;

      board[i][col]=0;
    }
  }
  return false;
}



function solveNQ(){
  var board = generateBoard(n);
  recurseNQ(board, 0);
  //if(recurseNQ(board, 0)===false){
    //console.log("No solution found");
   // return false;
```

```
 // }
 // printSolution(board);
}

function generateBoard(n){
  var board=[];
  for(var i=0; i<n; i++){
    board[i]=[];
    for(var j=0; j<n; j++){
      board[i][j]=0;
    }
  }
  return board;
}
console.log(solveNQ())
```

Output:

```
0
 0
 1
 0
 1
 0
 0
 0
 0
 0
 1
 0
 1
 0
 0
 0
 1
 0
 0
 0
 0
 0
 1
 1
 0
 0
 0
```

```
0
0
1
0
0
0
1
0
1
0
0
0
0
0
0
1
0
1
0
0
0
1
0
0
0
0
0
1
1
0
0
0
0
0
1
0
undefined
```