1. Demonstrate singly and doubly linked list

```
<html>
<body>
<script>
class LLNode
{
constructor(data)
{
this.data=data
this.next=null
}
}
class LinkedList{
constructor()
{
this.head=null
}
create(data)
{
const newNode=new LLNode(data)
if(this.head==null)
{
this.head=newNode
}
else
{
let current=this.head
while(current.next!=null)
{
```

```
current=current.next
}
current.next=newNode

}
}
display(){
if(this.head==null)
console.log("List is empty")
else
{
let current=this.head
while(current!=null)
{
console.log(current.data)
current=current.next
}
}
}
del()
{
let p=this.head
while(p.next!=null)
{
var q=p
p=p.next
}
q.next=null
console.log("The elements deleted is",p.data)
```

```
}
}
const list = new LinkedList();
list.create(50);
list.create(30);
list.create(20);
list.create(10);
list.create(5);
console.log("The element in the SLL are:")
list.display();
list.del();
console.log("After delete new node:");
list.display();

</script>
</body>
```

ii) Demonstration of Doubly linked list

```
<script>
class LLNode{
constructor(data)
```

```
{
this.data=data
this.next=null
this.prev=null
}
}
class LinkedList{
constructor()
{
this.head=null
this.tail=null
}
create(data)
{
var newNode=new LLNode(data);
if(this.head==null)
{
this.head=this.tail=newNode;
}
else
{
newNode.prev=this.tail
this.tail.next=newNode
this.tail=newNode
}
```

```
}
display()
{
if(this.head==null)
console.log("List is empty");
else
{
let current=this.head;
while(current!=null)
{
console.log(current.data);
current=current.next;
}
}
}
del(){
let q=this.tail
var p=q.prev
console.log("The element deleted is",q.data)
p.next=null
}
}
var list=new LinkedList();
list.create(10);
list.create(20);
```

```
list.create(30);
list.create(40);
list.create(50);
console.log("The element in Dll are:")
list.display();
list.del();
console.log("After element deleted in list:");
list.display();

</script>
```

2. STACK implementation using Array with PUSH, POP operations

```
<script>
class Stack {
   constructor() {
      this.items = [];
   }
   add(element) {
      return this.items.push(element);
   }
   remove() {
      if(this.items.length > 0) {
         return this.items.pop();
      }
   }
   peek() {
      return this.items[this.items.length - 1];
   }
   isEmpty(){
     return this.items.length == 0;
   }
   size(){
      return this.items.length;
   }
   clear(){
      this.items = [];
```

```
    }
}

let stack = new Stack();
stack.add(1);
stack.add(2);
stack.add(4);
stack.add(8);
console.log(stack.items);

stack.remove();
console.log(stack.items);

console.log(stack.peek());

console.log(stack.isEmpty());

console.log(stack.size());

stack.clear();
console.log(stack.items);
```

</script>

3. Reverse a string using stack

```
<script>
let stack=[];
stack.push(1);
console.log(stack);
stack.push(2);
console.log(stack);
stack.push(3);
console.log(stack);
stack.push(4);
console.log(stack);
stack.push(5);
console.log(stack);

console.log(stack.pop()); //5
console.log(stack); //[1,2,3,4];

console.log(stack.pop()); //4
console.log(stack); // [1,2,3];

console.log(stack.pop()); //3
console.log(stack); // [1,2];

console.log(stack.pop()); //2
console.log(stack); // [1];
```

```
console.log(stack.pop()); //1
console.log(stack); // []; ->empty

console.log(stack.pop()); // undefined

//Implementing javascript reverse stack using an array
function reverse(str)
{
let stack=[];
//push letter into stack
for(let i=0;i<str.length;i++)
{
stack.push(str[i]);
}
//pop letter from the stack
let reverseStr=' ';
while(stack.length>0)
{
reverseStr +=stack.pop();
}
return reverseStr;
}
console.log("Reverse String " +reverse('Mumbai'));
```

</script>

4. Check for balanced parentheses by using Stacks

```
<script>

// Javascript program for checking
// balanced brackets

// Function to check if brackets are balanced
function areBracketsBalanced(expr)
{

   // Using ArrayDeque is faster
   // than using Stack class
   let stack = [];

   // Traversing the Expression
   for(let i = 0; i < expr.length; i++)
   {
      let x = expr[i];

      if (x == '(' || x == '[' || x == '{')
      {

         // Push the element in the stack
         stack.push(x);
         continue;
```

```
}

// If current character is not opening
// bracket, then it must be closing.
// So stack cannot be empty at this point.
if (stack.length == 0)
    return false;

let check;
switch (x){
case ')':
    check = stack.pop();
    if (check == '{' || check == '[')
        return false;
    break;

case '}':
    check = stack.pop();
    if (check == '(' || check == '[')
        return false;
    break;

case ']':
    check = stack.pop();
    if (check == '(' || check == '{')
```

```
            return false;
          break;
      }
    }

    // Check Empty Stack
    return (stack.length == 0);
}

// Driver code
let expr = "([{}])";

// Function call
if (areBracketsBalanced(expr))
    document.write("Balanced ");
else
    document.write("Not Balanced ");

// This code is contributed by rag2127

</script>
```

5. Implement Stack using Linked List

```
<script>
class Node{
constructor(data)
{
this.data=data
this.next=null
}
}
class StackLL{
constructor(){
this.top=null
}
push(data){
const newNode=new Node(data)
newNode.next=this.top
this.top=newNode
}
pop(){
if(this.top==null)
console.log("Stack is empty,no element to pop")
else
{
console.log("The value popped is",this.top.data)
this.top=this.top.next
```

```
}
}
display(){
if(this.top==null)
console.log("Stack is empty,no element to display")
else{
var p=this.top
while(p!=null){
console.log(p.data)
p=p.next
}
}
}
}
var s=new StackLL()
s.push(10)
s.push(20)
console.log("The element in the stack are:")
s.display()
s.pop()
console.log("The element in the stack are:")
s.display()
s.pop()
s.pop()
```

</script>

Console
Default levels
The element in the stack are:
The element in the stack are:
Stack is empty,no element to pop

6. Demonstration of Linear Queue, Circular Queue, Priority Queue

```
<script>
// program to implement queue data structure

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

   // add element to the queue
   enqueue(element) {
      return this.items.push(element);
   }

   // remove element from the queue
   dequeue() {
      if(this.items.length > 0) {
         return this.items.shift();
      }
   }

   // view the last element
   peek() {
      return this.items[this.items.length - 1];
   }
```

```
    // check if the queue is empty
    isEmpty(){
      return this.items.length == 0;
    }

    // the size of the queue
    size(){
       return this.items.length;
    }

    // empty the queue
    clear(){
       this.items = [];
    }
}

let queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
queue.enqueue(8);
console.log(queue.items);

queue.dequeue();
```

```
console.log(queue.items);

console.log(queue.peek());

console.log(queue.isEmpty());

console.log(queue.size());

queue.clear();
console.log(queue.items);
</script>
```

ii)Demonstration of circular queue

```
<script>
class Queue {
 constructor() {
   this.elements = {};
   this.head = 0;
   this.tail = 0;
 }
```

```
  enqueue(element) {
    this.elements[this.tail] = element;
    this.tail++;
  }
  dequeue() {
    const item = this.elements[this.head];
    delete this.elements[this.head];
    this.head++;
    return item;
  }
  peek() {
    return this.elements[this.head];
  }
  get length() {
    return this.tail - this.head;
  }
  get isEmpty() {
    return this.length === 0;
  }
}

let q = new Queue();
for (let i = 1; i <= 7; i++) {
  q.enqueue(i);
}
// get the current item at the front of the queue
console.log(q.peek()); // 1

// get the current length of queue
```

```
console.log(q.length); // 7

// dequeue all elements
while (!q.isEmpty) {
  console.log("The deleted item"+" "+  q.dequeue());
}
</script>
```

iii) Demonstration of priority queue

```
<script>
function PriorityQueue(){
  let items = [];

  //Container
  function QueueElement(element, priority){
    this.element = element;
    this.priority = priority;
  }

  //Add a new element in queue
  this.enqueue = function(element, priority){
    let queueElement = new QueueElement(element, priority);
```

```
  //To check if element is added
  let added = false;
  for(let i = 0; i < items.length; i++){
    //We are using giving priority to higher numbers
    //If new element has more priority then add it at that place
    if(queueElement.priority > items[i].priority){
      items.splice(i, 0, queueElement);

      //Mark the flag true
      added = true;
      break;
    }
  }

  //If element is not added
  //Then add it to the end of the queue
  if(!added){
    items.push(queueElement);
  }
}

//Remove element from the queue
this.dequeue = () => {
  return items.shift();
}

//Return the first element from the queue
this.front = () => {
  return items[0];
```

```
  }

  //Return the last element from the queue
  this.rear = () => {
    return items[items.length - 1];
  }

  //Check if queue is empty
  this.isEmpty = () => {
    return items.length == 0;
  }

  //Return the size of the queue
  this.size = () => {
   return items.length;
  }

  //Print the queue
  this.print = function(){
    for(let i = 0; i < items.length; i++){
      console.log(`${items[i].element} - ${items[i].priority}`);
    }
  }
}
let pQ = new PriorityQueue();
pQ.enqueue(1, 3);
pQ.enqueue(5, 2);
pQ.enqueue(6, 1);
pQ.enqueue(11, 1);
```

```
pQ.enqueue(13, 1);

pQ.enqueue(10, 3);

pQ.dequeue();

pQ.print();

</script>
```

7. Reverse stack using queue

```
<script>
   // Javascript program to reverse a Queue

   let queue = [];

   // Utility function to print the queue
   function Print()
   {
      while (queue.length > 0) {
         document.write( queue[0] + ", ");
         queue.shift();
      }
   }

   // Function to reverse the queue
   function reversequeue()
   {
      let stack = [];
      while (queue.length > 0) {
         stack.push(queue[0]);
         queue.shift();
      }
      while (stack.length > 0) {
         queue.push(stack[stack.length - 1]);
```

```
        stack.pop();
      }
    }

    queue = []
    queue.push(10);
    queue.push(20);
    queue.push(30);
    queue.push(40);
    queue.push(50);
    queue.push(60);
    queue.push(70);
    queue.push(80);
    queue.push(90);
    queue.push(100);

    reversequeue();
    Print();

</script>
```

8. Practical based on binary search tree implementation with its operations

```
<script>
class Node
{
constructor(val)
{
this.val=val;
this.right=null;
this.left=null;
};
};

class BST
{
constructor()
{
this.root=null;
};
create(val)
{
const newNode = new Node(val);
if(!this.root)
{
this.root= newNode;
return this;
```

```
};
let current = this.root;
//const addSide=side=>

function addSide(side)
{
if(!current[side])
{
current[side]=newNode;
return this;
};
current = current[side];
};
while(true)
{
if(val==current.val)
return this;
if(val<current.val)
addSide('left');
else
addSide('right');
};
};

BFS()
```

```
{
let visited=[], queue = [], current = this.root;

queue.push(current);
while(queue.length)
{
current = queue.shift();
visited.push(current.val);

if(current.left)
        queue.push(current.left);
if(current.right)
        queue.push(current.right);
};
return visited;
}

preOrder()
{
let visited = [], current = this.root;
let traverse = node =>
{
visited.push(node.val);

if(node.left)
```

```
        traverse(node.left);
if(node.right)
        traverse(node.right);
};
traverse(current);
return visited;
}

postOrder()
{
let visited = [], current = this.root;
// let traverse = node =>
function traverse(node)
{



if(node.left)
        traverse(node.left);
if(node.right)
        traverse(node.right);
visited.push(node.val);


};
traverse(current);
return visited;
```

```
}

inOrder()
{
let visited = [], current = this.root;
function traverse(node)
{
if(node.left)
        traverse(node.left);
visited.push(node.val);
if(node.right)
        traverse(node.right);
};
traverse(current);
return visited;
}
};      // Tree Class Ends

const tree = new BST();
tree.create(20);
tree.create(14);
tree.create(57);
tree.create(9);
tree.create(19);
tree.create(31);
```

```
tree.create(62);
tree.create(3);
tree.create(11);
tree.create(72);
console.log("Binary Tree is created successfully");
console.log("-------BFS---------");
console.log(tree.BFS());
console.log("PREORDER TRAVERSAL IS AS FOLLOW");
console.log(tree.preOrder());
console.log("POSTORDER TRAVERSAL IS AS FOLLOW");
console.log(tree.postOrder());
console.log("INORDER TRAVERSAL IS AS FOLLOW");
console.log(tree.inOrder());

</script>
```

9. Graph implementation and graph traversals

```
<script>
class Graph{
constructor(noOfVertices)
{
this.noOfVertices=noOfVertices
this.adjList=new Map()
}
addVertex(v)
{
this.adjList.set(v,[ ])
}
addEdge(v,w)
{
this.adjList.get(v).push(w)
this.adjList.get(w).push(v)
}
printGraph(){
let key=this.adjList.keys()
for(let v of key)
{
let eList=this.adjList.get(v)
let data=' '
for(let e in eList)
{
```

```
data=data+eList[e]+'  '
}
console.log(v+'=>'+data+'null')
}
}
dfs(v)
{
let s=[ ]
let visited=[ ]
let keys=this.adjList.keys()
for(let i of keys)
{
visited[i]=false
}
s.push(v)
while(s.length>0)
{
let element=s.pop()
if(!visited[element])
{
console.log(element)
visited[element]=true
}
else
continue
```

```
let eList=this.adjList.get(element)
for(var i=eList.length-1;i>=0;i--)
{
let e=eList[i]
if(!visited[e])
{
s.push(e)
}
}
}
}
}
var g=new Graph(4)
g.addVertex('A')
g.addVertex('B')
g.addVertex('C')
g.addVertex('D')
g.addEdge('A','B')
g.addEdge('B','C')
g.addEdge('C','D')
g.addEdge('A','D')
g.printGraph()
console.log("DFS TRAVERSAL")
g.dfs('A')
</script>
```

ii) BFS

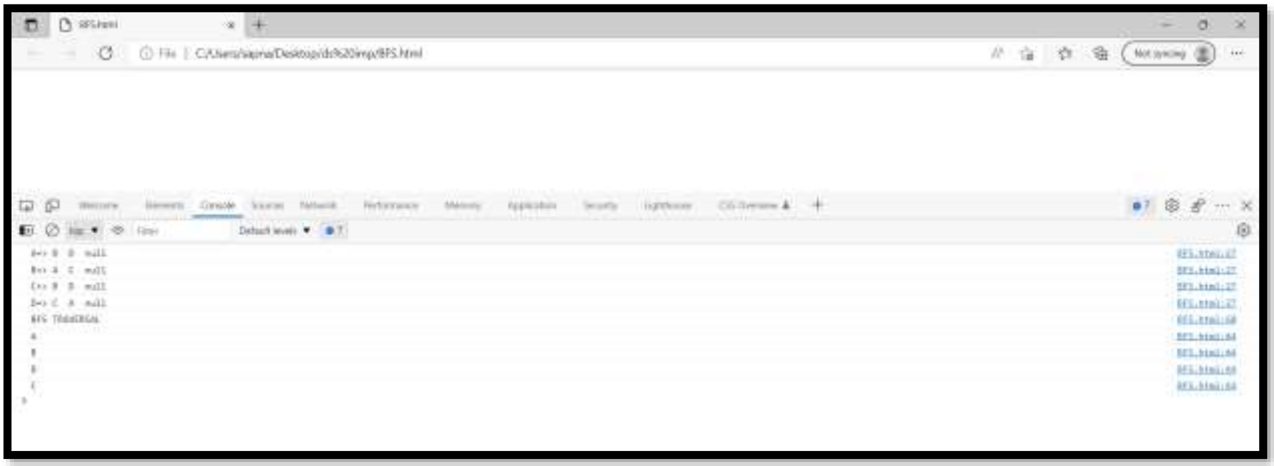
```
<script>
class Graph{
constructor(noOfVertices)
{
this.noOfVertices=noOfVertices
this.adjList=new Map()
}
addVertex(v)
{
this.adjList.set(v,[ ])
}
addEdge(v,w)
{
this.adjList.get(v).push(w)
this.adjList.get(w).push(v)
}
printGraph(){
let key=this.adjList.keys()
```

```
for(let v of key)
{
let eList=this.adjList.get(v)
let data=' '
for(let e in eList)
{
data=data+eList[e]+'  '
}
console.log(v+'=>'+data+'null')
}
}
bfs(v)
{
let q= []
let visited=[ ]
let keys=this.adjList.keys()
for(let i in keys)
{
visited[i]=false
}
q.push(v)
while(q.length>0)
{
let element=q.shift()
visited[element]=true
```

```
console.log(element)
let eList=this.adjList.get(element)
for(let i in eList)
{
let e=eList[i]
if(!visited[e])
{
q.push(e)
visited[e]=true
}
}
}
}
}
var g=new Graph(4)
g.addVertex('A')
g.addVertex('B')
g.addVertex('C')
g.addVertex('D')
g.addEdge('A','B')
g.addEdge('B','C')
g.addEdge('C','D')
g.addEdge('A','D')
g.printGraph()
console.log("BFS TRAVERSAL")
```

g.bfs('A')

</script>



10. Implementation of Hashing

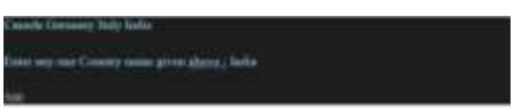
```
class classHashTable
{
constructor()
   {
this.values={};
this.length=0;
this.size=0;
   }
calculateHash(key)
   {
      return key.toString().length%this.size;
   }

   add(key,value)
   {
      const hash=this.calculateHash(key);
      if(!this.values.hasOwnProperty(hash))
this.values[hash]={};
      if(!this.values[hash].hasOwnProperty(key))
this.length++;
this.values[hash][key] = value;
   }

   search(key)
```

```
    {
        const hash=this.calculateHash(key);

if(this.values.hasOwnProperty(hash)&&this.values[hash].hasOwnProperty(key))
            return this.values[hash][key];
        else
            return null;
    }
}
//createobjectoftypehashtable
const ht=new HashTable();
//adddatatothehashtableht
ht.add("Canada","300");
ht.add("Germany","100");
ht.add("Italy","50");
ht.add("India","500");
//search
const ps=require("prompt-sync");
const prompt=ps();
console.log("Canada Germany Italy India");
let str = prompt('Enter any one Country name given above : ');
console.log(ht.search(str));
```



11. Practical based on Brute Force technique

```
//Rain Terrace
<script>
function maxWater(arr,n)
{
var res=0
for(i=1;i<n-1;i++)
{
var left=arr[i]
for(var j=0;j<i;j++)
{
left=Math.max(left,arr[j])
}
var right=arr[i]
for(j=i+1;j<n;j++)
{
right=Math.max(right,arr[j])
}
res+=Math.min(left,right)-arr[i]
}
return res
}
var arr=[0,1,0,2,1,0,1,3,2,1,2,1]
var n=arr.length
console.log(maxWater(arr,n))
```

```
</script>
```

```
//Linear Search
<script>
function linearSearch(arr,key)
{
for(let i=0;i<arr.lengt;i++)
{
if(arr[i]==key)
{
return i
}
}
return -1
}
var a=[80,45,70,40,35,50]
i=linearSearch(a,35)
if(i==-1)
console.log("Element is not in the list")
else
console.log("Element is present at position",i)
</script>
```


```
//Recursive Staircase
<script>
function fibo(n)
{
if(n<=1)
return n
else
return fibo(n-2)+fibo(n-1)
}
function count_ways(s)
{
return fibo(s+1)
}
console.log("The number of ways= ",count_ways(4))
</script>
```

12. Practical based on Greedy Algorithm-Prim's/Kruskal's algorithm

```
<script>
//Implementation of prims algorithm
let V = 5;
function minKey(key, mstSet)
{
   let min = Number.MAX_VALUE, min_index;

   for (let v = 0; v < V; v++)
      if (mstSet[v] == false && key[v] < min)
         min = key[v], min_index = v;

   return min_index;
}
function printMST(parent, graph)
{
   document.write("Edge      Weight" + "<br>");
   for (let i = 1; i < V; i++)
      document.write(parent[i] + "   -  " + i + "     " + graph[i][parent[i]] + "<br>");
}

function primMST(graph)
{
   let parent = [];
   let key = [];
```

```
    let mstSet = [];
    for (let i = 0; i < V; i++)
        key[i] = Number.MAX_VALUE, mstSet[i] = false;
    key[0] = 0;
    parent[0] = -1; // First node is always root of MST

    for (let count = 0; count < V - 1; count++)
    {
        let u = minKey(key, mstSet);
        mstSet[u] = true;
        for (let v = 0; v < V; v++)
   if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
 printMST(parent, graph);
}

let graph = [ [ 0, 2, 0, 6, 0 ],
[ 2, 0, 3, 8, 5 ],
[ 0, 3, 0, 0, 7 ],
[ 6, 8, 0, 0, 9 ],
[ 0, 5, 7, 9, 0 ] ];
 primMST(graph);
 </script>
```

ii) Kruskal’s algorithm

```
<script>
// Simple Javascript implementation for Kruskal's
// algorithm
var V = 5;
var parent = Array(V).fill(0);
var INF = 1000000000;

// Find set of vertex i
function find(i)
{
   while (parent[i] != i)
      i = parent[i];
   return i;
}

// Does union of i and j. It returns
// false if i and j are already in same
// set.
function union1(i, j)
```

```
{
    var a = find(i);
    var b = find(j);
    parent[a] = b;
}

// Finds MST using Kruskal's algorithm
function kruskalMST(cost)
{
    var mincost = 0; // Cost of min MST.

    // Initialize sets of disjoint sets.
    for (var i = 0; i < V; i++)
        parent[i] = i;

    // Include minimum weight edges one by one
    var edge_count = 0;
    while (edge_count < V - 1)
    {
        var min = INF, a = -1, b = -1;
        for (var i = 0; i < V; i++)
        {
            for (var j = 0; j < V; j++)
            {
                if (find(i) != find(j) && cost[i][j] < min)
```

```
            {
                min = cost[i][j];
                a = i;
                b = j;
            }
        }
    }

    union1(a, b);
    document.write(`Edge ${edge_count++}:(${a},
    ${b}) cost:${min} <br>`);
    mincost += min;
  }
  document.write(`<br> Minimum cost= ${mincost} <br>`);
}

// Driver code

/* Let us create the following graph
      2 3
  (0)--(1)--(2)
  | / \ |
  6| 8/ \5 |7
  | /    \ |
  (3)-------(4)
```

```
        9       */
var cost = [
    [INF, 2, INF, 6, INF],
    [2, INF, 3, 8, 5],
    [INF, 3, INF, INF, 7],
    [6, 8, INF, INF, 9],
    [INF, 5, 7, 9, INF]];
// Print the solution
kruskalMST(cost);
 </script>
```

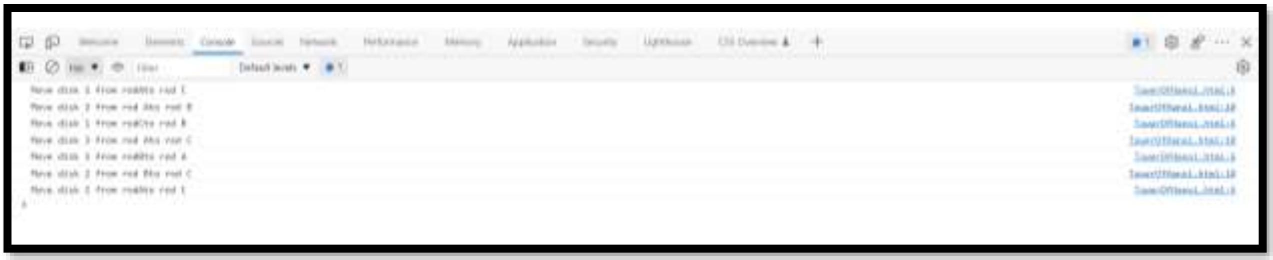
13. Practical based on Divide and Conquer Technique-Binary Search, Tower of Hanoi

i) Binary Search

```
<script>
//Implement of binary search
function binarySearch(arr,l,r,x)
{
if(r>=l)
{
var mid=Math.floor((l+r)/2)
if(arr[mid]==x)
return mid
if(arr[mid]>x)
return binarySearch(arr,l,mid-1,x)
else
return binarySearch(arr,mid+1,r,x)
}
return -1
}
let arr=[1,3,5,7,8,9]
let x=8
console.log(binarySearch(arr,0,arr.length-1,x))
</script>
```

ii) Tower of Hanoi

```
<script>
function towerOfHanoi(n,A,B,C)
{
if(n==1)
{
console.log("Move disk 1 from rod"+ A + "to rod "+ B)
return
}
towerOfHanoi(n-1,A,C,B)
console.log("Move disk "+ n +" from rod "+ A +"to rod "+ B)
towerOfHanoi(n-1,C,B,A)
}
var n=3
towerOfHanoi(n,'A','C','B')
</script>
```


Console
Default levels

14. Implementation of Dynamic Programming- LCS, Regular Expression Matching

i) LCS

```
<script>
function lcs(  X,  Y , m , n )
{
   if (m == 0 || n == 0)
   return 0;
   if (X[m-1] == Y[n-1])
   return 1 + lcs(X, Y, m-1, n-1);
   else
   return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}


function max(a , b)
{
   return (a > b)? a : b;
}




   var s1 = "AGGTAB";
   var s2 = "GXTXAYB";

   var X=s1;
   var Y=s2;
   var m = X.length;
   var n = Y.length;
```

```
  document.write("Length of LCS is" + " " +
                        lcs( X, Y, m, n ) );
</script>
```

Longest Common Substring

```
<script>
  function LCSubStr( X,  Y , m , n) {


      var LCStuff =
      Array(m + 1).fill().map(()=>Array(n + 1).fill(0));
      var result = 0;
      for (i = 0; i <= m; i++) {
         for (j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
               LCStuff[i][j] = 0;
            else if (X[i - 1] == Y[j - 1]) {
               LCStuff[i][j] = LCStuff[i - 1][j - 1] + 1;
               result = Math.max(result, LCStuff[i][j]);
            } else
               LCStuff[i][j] = 0;
         }
      }
      return result;
  }


      var X = "OldSite:GeeksforGeeks.org";
```

```
    var Y = "NewSite:GeeksQuiz.com";

    var m = X.length;
    var n = Y.length;

    document.write("Length of Longest Common Substring is " +
    LCSubStr(X, Y, m, n));
</script>
```

ii) Regular Expression Matching

```
<script>
//Implementation of regular expression matching using dynamic approach
class RE{
constructor(){
this.f=[20][20]
}
isMatch(s,p){
var m=s.length
var n=p.length
this.f=[20]
for(var i=0;i<=20;i++)
{
this.f[i]=[20]
```

```
for(var j=0;j<=20;j++)
{
this.f[i][j]=0
}
}
for(i=1;i<=m;i++)
this.f[i][0]=0;
for(j=1;j<=n;j++)
this.f[0][j]=j>1 && '*'==p[j-1] && this.f[0][j-2]
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
if(p[j-1]!='*')
this.f[i][j]=this.f[i-1][j-1] && (s[i-1]==p[j-1]||'.'==p[j-1])
else
this.f[i][j]=this.f[i][j-2] || (s[i-1] == p[j-2] || '.' == p[j-2]) && this.f[i-1][j]
return this.f[m][n]
}
}
var r=new RE
var s="aadb"
var p="c*a*b"
console.log(r.isMatch(s,p))
</script>
```


Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
CSS Overview
Default levels
false

15. Practical based on backtracking- N Queens problem

```
<script>
var N=5
function printSolution(board)
{
	for(var i=0;i<N;i++){
	var s=" "
		for (var j=0;j<N;j++)
		{
			s+=board[i][j]+" "
		}
		console.log(s)
	}
}
function isSafe(board,row,col)
{
	var i,j
	for(i=0;i<col;i++)
		if(board[row][i])
			return false
for(i=row,j=col;i>=0 && j>=0;i--,j--)
	if(board[i][j])
	return false
for(i=row,j=col;j>=0 && i<N;i++,j--)
	if(board[i][j])
	return false
return true
}
function solveNQUtil(board,col){
```

```
            if(col>=N)
            return true
for(var i=0;i<N;i++)
{
            if(isSafe(board,i,col))
            {
                        board[i][col]=1
                        if(solveNQUtil(board,col+1))
                                    return true
                        board[i][col]=0
            }
}
return false
}
var board=[[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]]
if (solveNQUtil(board,0) == false)
{
            console.log("Solution does not exist")
            //return False
}
printSolution(board)
</script>
```