Assignment- 1

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

**Practical No. 1. Program to implement Singly LinkedList**

**Solution:**

class Node{

constructor(data)

{

this.data=data; this.next=null;

} }

class LinkedList

{

constructor()

{

this.head=null; this.count=0;

} create(data)

{

if(this.head==null)

{

this.head=new Node(data);

} else{

var temp=this.head;

while(temp.next!=null)

{

temp=temp.next;

}

temp.next=new Node(data);

}

this.count=this.count+1;

}

insert(data,pos)

{ if(pos==1)

{

var n=new Node(data); n.next=this.head; this.head=n;

}

else if(pos>=this.count+1)

{

this.create(data);

} else

{

var n=new Node(data); var i=2; var temp=this.head;

while(i!=pos && temp.next!=null)

{

temp=temp.next; i=i+1;

}

n.next=temp.next;

temp.next=n;

}

this.count=this.count+1; console.log("After insertion of ",data," : ");

this.display();

} delete(data) {

if(this.head==null)

{

console.log("List is Empty....");

} else{ var temp=this.head; var prev=null; var flag=false;

while(temp.next!=null)

{

if(temp.data==data)

{ flag=true; break;

} prev=temp;

temp=temp.next;

}

if(flag)

{

if(temp.next==null)

{

prev.next=null;

}

else if(temp==this.head)

{

this.head=this.head.next;

temp.next=null;

} else

{

prev.next=temp.next;

temp.next=null;

}

}

this.count=this.count-1;

console.log("After Deletion of ",data," : ");

}

} display() {

if(this.head==null)

{

console.log("List is empty.....");

} else{ var temp=this.head; console.log("The list is as follow : "); while(temp!=null)

{

console.log(temp.data);

temp=temp.next;

}

}

} } var list=new LinkedList(); list.create(1); list.create(2); list.create(4); list.create(5); list.display(); list.insert(30,3); list.insert(40,4); list.delete(30); list.display();

**Output :**

The list is as follow :

1

2

4

5 After insertion of 30 :

The list is as follow :

1

2

30

4

5 After insertion of 40 :

The list is as follow :

1

2

30

40

4

5 After Deletion of 30 :

The list is as follow :

1

2

40

4

5

Assignment- 2

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

**Practical No. 2: Program to implement Doubly Linked List.**

**Solution :**

class Node {

constructor(data)

{

this.data=data; this.next=null; this.prev=null;

} }

class LinkedList

{

constructor()

{

this.head=null; this.count=0;

} create(data)

{

if(this.head==null)

{

this.head=new Node(data);

} else

{

var temp=this.head;

while(temp.next!=null)

{

temp=temp.next;

}

var n=new Node(data);

temp.next=n;

n.prev=temp;

}

this.count=this.count+1;

} insert(data,pos)

{ if(pos==1)

{

var n=new Node(data); n.next=this.head; this.head.prev=n; n.next=this.head; this.head=n;

}

else if(pos>=this.count+1)

{

this.create(data);

} else

{

var n=new Node(data); var i=2; var temp=this.head;

while(i!=pos && temp.next!=null)

{

temp=temp.next; i=i+1;

}

temp.next.prev=n; n.next=temp.next; temp.next=n;

n.prev=temp;

}

this.count=this.count+1; console.log("After insertion of ",data," : ");

this.display();

} delete(data) {

if(this.head==null)

{

console.log("List is Empty......");

} else

{

var temp=this.head; var prev=null;

var flag=false;

while(temp.next!=null)

{

if(temp.data==data)

{ flag=true; break;

}

temp=temp.next;

} if(flag)

{

if(temp.next==null)

{

prev.next=null;

}

else if(temp==this.head)

{

this.head=this.head.next; this.head.prev=null;

temp.next=null;

}

else{

temp.prev.next=temp.next; temp.next.prev=temp.prev; temp.next=null;

temp.prev=null;

}

}

this.count=this.count-1;

console.log("After delation of ",data," : ");

} } display()

{

console.log("List as Follow : ")

if(this.head==null)

{

console.log("List is Empty......")

}

var temp=this.head;

while(temp!=null)

{

console.log(temp.data);

temp=temp.next;

}

} } var list=new LinkedList(); list.create(1); list.create(2); list.create(4); list.create(5); list.display(); list.insert(30,3); list.insert(40,50); list.delete(2); list.display();

**Output :**

List as Follow :

1

2

4

5 After insertion of 30 :

List as Follow :

1

2

30

4

5 After insertion of 40 :

List as Follow :

1

2

30

4

5

40 After delation of 2 :

List as Follow : 1

30 4

5

40

Assignment- 3

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

**Practical No. 3: Program to implement Stack using LinkedList – Push, Pop, Display operation.**

**Solution :**

//Stack using link list class Node {

constructor(data)

{

this.data=data; this.next=null

} } class Stack

{

constructor()

{

this.top=null;

}

isEmpty()

{ if(this.top==null)

{ return true;

} else{

return false;

}

}

push(data)

{

if(this.isEmpty())

{

this.top=new Node(data);

}

else{ var n=new Node(data); n.next=this.top; this.top=n;

} } pop() {

if(this.isEmpty())

{

console.log("Stack UnderFlow.............");

} else{ var n=this.top; this.top=this.top.next; this.n=null;

console.log("Popped : ",n.data);

} } display()

{ var temp=this.top; while(temp!=null)

{

console.log(temp.data);

temp=temp.next;

}

} } var s=new Stack(); console.log("Before Pop :: "); s.push(10);

s.push(30);

s.push(50);

s.push(70);

s.push(90);

s.push(100);

s.display();

s.pop();

s.pop(); console.log("After Pop :: "); s.display();

**Output:**

Before Pop ::

100

90

70

50

30

10

Popped : 100 Popped : 90 After Pop ::

70

50

30

10

Assignment- 4

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

**Practical No. 4: Program to Demonstate Linear Queue using LinkedList. Solution :**

class Node{

constructor(data)

{ this.data=data; this.next=null;

} } class Queue

{

constructor()

{

this.front=null; this.rear=null;

}

isEmpty()

{

if(this.front==null && this.rear==null)

{

return true;

} else{

return false;

}

}

push(data)

{

if(this.isEmpty())

{

this.front=this.rear=new Node(data);

} else{ var n=new Node(data); this.rear.next=n; this.rear=n;

} } pop() {

if(this.isEmpty())

{

console.log("Queue is Empty....");

} else

if(this.front==this.rear)

{

this.front=this.rear=null;

} else{ var temp=this.front this.front=this.front.next; temp.next=null;

console.log("Popped : ",temp.data);

} } display()

{ var temp=this.front;

while(temp!=null)

{

console.log(temp.data);

temp=temp.next;

}

} } var s=new Queue(); console.log("Before Pop :: "); s.push(10);

s.push(30);

s.push(50);

s.push(70);

s.push(90);

s.push(100);

s.display();

s.pop();

s.pop(); console.log("After Pop :: "); s.display();

**Output :**

Before Pop ::

10

30

50

70

90

100

Popped : 10

Popped : 30 After Pop ::

50

70

90

100

Assignment- 5

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

**Practical No. 5: Program to implement Stack with Array.**

**Solution :**

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()==true)

console.log(".....STACK UNDERFLOW ERROR....") else

{

var temp =this.data[this.top-1]; this.data[this.top-1]=null; this.top=this.top-1;

console.log("popped element is : "+temp);

//return this.data.pop();

}

} display(){ if(this.isEmpty()==true)

console.log("....Stack Underflow error.....");

else{ var temp=this.top-1;

while(temp>=0)

{

console.log(this.data[temp]);

temp--;

}

}

} } var stk=new Stack(); console.log("Before Pop :: "); stk.push(110); stk.push(220); stk.push(330); stk.push(440); stk.display(); stk.pop(); stk.pop(); console.log("After Pop :: "); stk.display(); stk.push(770); stk.push(990); console.log("After Push element :: ");

stk.display();

**Output :**

Before Pop ::

440

330

220 110 popped element is : 440 popped element is : 330 After Pop ::

220

110 After Push element ::

990

770

220

110

Assignment- 6

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

**Practical No. 6: Program to Reverse the String using Stack.**

**Solution :**

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;

}

display()

{

if(this.isEmpty == true)

{

console.log("Stack underflow error");

} else

{ var temp = this.top-1;

while(temp>=0)

{

console.log(this.data[temp]);

temp--;

}

}

}

}

var st = new Stack(); var s = "DSA Lab";

var i ; for(i=0;i<s.length;i++) { st.push(s.charAt(i)); } console.log("Reverse Array : ") st.display();

**Output :**

Reverse Array :

b a L

A

S

D

Assignment- 7

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 7: To Demonstrate Circular Queue Using Linked List:

# class Node {

constructor(data) {

this.data = data;

this.next = null;

}

}

class CircularQueue {

constructor() {

this.front = null;

this.rear = null;

}

enqueue(data) {

const newNode = new Node(data);

if (!this.front) {

this.front = newNode;

this.rear = newNode;

newNode.next = this.front; // Make it circular

} else {

this.rear.next = newNode;

this.rear = newNode;

this.rear.next = this.front; // Make it circular

}

}

dequeue() {

if (!this.front) {

console.log("Queue is empty");

return null;

}

const removedData = this.front.data;

if (this.front === this.rear) {

this.front = null;

this.rear = null;

} else {

this.front = this.front.next;

this.rear.next = this.front; // Update rear's next to maintain circular structure

}

return removedData;

}

display() {

if (!this.front) {

console.log("Queue is empty");

return;

}

let current = this.front;

do {

console.log(current.data);

current = current.next;

} while (current !== this.front);

}

}

// Example usage

const circularQueue = new CircularQueue();

circularQueue.enqueue(1);

circularQueue.enqueue(2);

circularQueue.enqueue(3);

console.log("Queue after enqueue:");

circularQueue.display();

circularQueue.dequeue();

console.log("Queue after dequeue:");

circularQueue.display();

**Output :**

Queue after enque

1

2

3

Queue after deque

2

3

Assignment- 8

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 8: To Demonstrate Priority Queue using linked list

# class Node {

# constructor(data, priority) {

# this.data = data;

# this.priority = priority;

# }

# }

# class PriorityQueue {

# constructor() {

# this.queue = [];

# }

# enqueue(data, priority) {

# const newNode = new Node(data, priority);

# let added = false;

# for (let i = 0; i < this.queue.length; i++) {

# if (newNode.priority < this.queue[i].priority) {

# this.queue.splice(i, 0, newNode);

# added = true;

# break;

# }

# }

# if (!added) {

# this.queue.push(newNode);

# }

# }

# dequeue() {

# if (this.isEmpty()) {

# return "Queue is empty";

# }

# return this.queue.shift().data;

# }

# isEmpty() {

# return this.queue.length === 0;

# }

# display() {

# if (this.isEmpty()) {

# console.log("Queue is empty");

# return;

# }

# console.log("Priority Queue:");

# this.queue.forEach((node) => {

# console.log(Data: ${node.data}, Priority: ${node.priority});

# });

# }

# }

# // Example usage

# const priorityQueue = new PriorityQueue();

# priorityQueue.enqueue("Task 1", 3);

# priorityQueue.enqueue("Task 2", 1);

# priorityQueue.enqueue("Task 3", 2);

# console.log("Priority Queue after enqueue:");

# priorityQueue.display();

# const dequeuedItem = priorityQueue.dequeue();

# console.log(Dequeued item: ${dequeuedItem});

# console.log("Priority Queue after dequeue:");

# priorityQueue.display();

# Output:

Priority Queue after enqueue:

Priority Queue:

Data: Task 2, Priority: 1

Data: Task 3, Priority: 2

Data: Task 1, Priority: 3

Dequeued item: Task 2

Priority Queue after dequeue:

Priority Queue:

Data: Task 3, Priority: 2

Data: Task 1, Priority:

Assignment- 9

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 9: Balance Parenthesis using Stack.

function balanceParentheses(expression) {

const stack = [];

const openingBrackets = ['(', '[', '{'];

const closingBrackets = [')', ']', '}'];

for (let char of expression) {

if (openingBrackets.includes(char)) {

stack.push(char);

} else if (closingBrackets.includes(char)) {

const lastOpeningBracket = stack.pop();

const correspondingOpening = openingBrackets[closingBrackets.indexOf(char)];

if (lastOpeningBracket !== correspondingOpening) {

return "Unbalanced parentheses";

}

}

}

return stack.length === 0 ? "Balanced parentheses" : "Unbalanced parentheses";

}

// Example usage:

const expression1 = "((a + b) \* (c - d))";

const expression2 = "((a + b) \* (c - d)";

console.log(balanceParentheses(expression1));

console.log(balanceParentheses(expression2));

# Output:

Balanced parentheses

Unbalanced parentheses

Assignment- 10

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 10: Program to Implement Reverse Stack Using Queue

# Solution:

class Stack {

constructor() {

this.items = [];

}

push(element) {

this.items.push(element);

}

pop() {

if (this.items.length === 0) {

return "Underflow";

}

return this.items.pop();

}

isEmpty() {

return this.items.length === 0;

}

size() {

return this.items.length;

}

print() {

console.log(this.items.join(' '));

}

}

function reverseStack(stack) {

const reversedStack = new Stack();

while (!stack.isEmpty()) {

reversedStack.push(stack.pop());

}

return reversedStack;

}

// Example usage

const originalStack = new Stack();

originalStack.push(1);

originalStack.push(2);

originalStack.push(3);

console.log("Original Stack:");

originalStack.print();

const reversedStack = reverseStack(originalStack);

console.log("Reversed Stack:");

reversedStack.print();

**Output :**

Original Stack:

1 2 3

Reversed Stack:

3 2 1

Assignment- 11

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 11: Program to implement Binary Search Tree with its operations

# Solution:

class Node {

constructor(key) {

this.key = key;

this.left = null;

this.right = null;

}

}

class BinarySearchTree {

constructor() {

this.root = null;

}

insert(key) {

this.root = this.\_insertRec(this.root, key);

}

\_insertRec(root, key) {

if (root === null) {

return new Node(key);

}

if (key < root.key) {

root.left = this.\_insertRec(root.left, key);

} else if (key > root.key) {

root.right = this.\_insertRec(root.right, key);

}

return root;

}

search(key) {

return this.\_searchRec(this.root, key);

}

\_searchRec(root, key) {

if (root === null || root.key === key) {

return root;

}

if (key < root.key) {

return this.\_searchRec(root.left, key);

} else {

return this.\_searchRec(root.right, key);

}

}

inOrderTraversal(callback) {

this.\_inOrderTraversalRec(this.root, callback);

}

\_inOrderTraversalRec(root, callback) {

if (root !== null) {

this.\_inOrderTraversalRec(root.left, callback);

callback(root.key);

this.\_inOrderTraversalRec(root.right, callback);

}

}

}

// Example usage

const bst = new BinarySearchTree();

bst.insert(10);

bst.insert(5);

bst.insert(15);

bst.insert(3);

bst.insert(7);

bst.insert(12);

bst.insert(18);

console.log("In-order Traversal:");

bst.inOrderTraversal(key => console.log(key));

const searchResult = bst.search(7);

console.log("Search result for key 7:", searchResult ? "Found" : "Not Found");

# Output :

In-order Traversal:

3

5

7

10

12

15

18

Search result for key 7: Found

Assignment- 12

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 12: Program to implement Graph traversals

# Solution:

class Graph {

constructor() {

this.vertices = [];

this.edges = new Map();

}

addVertex(vertex) {

this.vertices.push(vertex);

this.edges.set(vertex, []);

}

addEdge(vertex1, vertex2) {

this.edges.get(vertex1).push(vertex2);

this.edges.get(vertex2).push(vertex1);

}

depthFirstTraversal(startingVertex) {

const visited = new Set();

const result = [];

const dfs = (vertex) => {

visited.add(vertex);

result.push(vertex);

for (const neighbor of this.edges.get(vertex)) {

if (!visited.has(neighbor)) {

dfs(neighbor);

}

}

};

dfs(startingVertex);

return result;

}

}

// Example usage:

const graph = new Graph();

graph.addVertex("A");

graph.addVertex("B");

graph.addVertex("C");

graph.addVertex("D");

graph.addVertex("E");

graph.addEdge("A", "B");

graph.addEdge("B", "C");

graph.addEdge("B", "D");

graph.addEdge("C", "E");

const traversalResult = graph.depthFirstTraversal("A");

console.log("Depth-First Traversal:", traversalResult);

# Output :

Depth-First Traversal: [ 'A', 'B', 'C', 'E', 'D' ]

Assignment- 13

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 13: Program to Implement Linear Search(Brute Force)

# Solution :

function linearSearch(arr, target) {

for (let i = 0; i < arr.length; i++) {

if (arr[i] === target) {

return i; // Return the index if the target is found

}

}

return -1; // Return -1 if the target is not found

}

// Example usage

const array = [4, 2, 8, 1, 5, 7];

const targetValue = 5;

console.log("Array:", array);

console.log("Target Value:", targetValue);

const resultIndex = linearSearch(array, targetValue);

if (resultIndex !== -1) {

console.log(Target value ${targetValue} found at index ${resultIndex}.);

} else {

console.log(Target value ${targetValue} not found in the array.);

}

**Output :** Array: [4, 2, 8, 1, 5, 7]

Target Value: 5

Target value 5 found at index 4.D

Assignment- 14

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 14: Program to Implement binary search (Divide and Conquer)

# Solution :

function binarySearch (arr, target) {

let low = 0;

let high = arr.length - 1;

while (low <= high) {

const mid = Math.floor((low + high) / 2);

if (arr[mid] === target) {

return mid; // Return the index if the target is found

} else if (arr[mid] < target) {

low = mid + 1; // If target is greater, ignore the left half

} else {

high = mid - 1; // If target is smaller, ignore the right half

}

}

return -1; // Return -1 if the target is not found

}

// Example usage

const sortedArray = [1, 3, 5, 7, 9, 11, 13, 15];

const targetValue = 7;

console.log("Sorted Array:", sortedArray);

console.log("Target Value:", targetValue);

const resultIndex = binarySearch(sortedArray, targetValue);

if (resultIndex !== -1) {

console.log(Target value ${targetValue} found at index ${resultIndex}.);

} else {

console.log(Target value ${targetValue} not found in the array.);

}

**Output :**

Sorted Array: [1, 3, 5, 7, 9, 11, 13, 15]

Target Value: 7

Target value 7 found at index 3.

Assignment- 15

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 15: Program to Implement Hashing

# Solution :

class HashTable {

constructor(size = 10) {

this.size = size;

this.table = new Array(size);

}

hash(key) {

let hashValue = 0;

for (let i = 0; i < key.length; i++) {

hashValue += key.charCodeAt(i);

}

return hashValue % this.size;

}

insert(key, value) {

const index = this.hash(key);

if (!this.table[index]) {

this.table[index] = [];

}

this.table[index].push({ key, value });

}

search(key) {

const index = this.hash(key);

if (!this.table[index]) {

return null;

}

for (const pair of this.table[index]) {

if (pair.key === key) {

return pair.value;

}

}

return null;

}

display() {

console.log("Hash Table:");

this.table.forEach((bucket, index) => {

if (bucket) {

console.log(Index ${index}:, bucket.map(pair => {${pair.key}: ${pair.value}}));

} else {

console.log(Index ${index}: Empty);

}

});

}

}

// Example usage

const hashTable = new HashTable();

hashTable.insert("name", "John");

hashTable.insert("age", 25);

hashTable.insert("city", "Example City");

console.log("After Insertion:");

hashTable.display();

const ageValue = hashTable.search("age");

console.log("Search Result for 'age':", ageValue);

**Output :**

After Insertion:

Hash Table:

Index 0: Empty

Index 1: Empty

Index 2: Empty

Index 3: [{name: John}]

Index 4: Empty

Index 5: [{age: 25}, {city: Example City}]

Index 6: Empty

Index 7: Empty

Index 8: Empty

Index 9: Empty

Search Result for 'age': 25

Assignment- 16

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 16: Implement Prim’s algorithm using greedy technique.

# Solution:

# class Graph {

# constructor(vertices) {

# this.vertices = vertices;

# this.adjacencyMatrix = [];

# for (let i = 0; i < vertices; i++) {

# this.adjacencyMatrix.push(new Array(vertices).fill(0));

# }

# }

# addEdge(src, dest, weight) {

# this.adjacencyMatrix[src][dest] = weight;

# this.adjacencyMatrix[dest][src] = weight;

# }

# primMST() {

# const parent = new Array(this.vertices).fill(-1);

# const key = new Array(this.vertices).fill(Number.MAX\_VALUE);

# const mstSet = new Array(this.vertices).fill(false);

# key[0] = 0; // Start from the first vertex

# for (let count = 0; count < this.vertices - 1; count++) {

# const u = this.minKey(key, mstSet);

# mstSet[u] = true;

# for (let v = 0; v < this.vertices; v++) {

# if (

# this.adjacencyMatrix[u][v] !== 0 &&

# mstSet[v] === false &&

# this.adjacencyMatrix[u][v] < key[v]

# ) {

# parent[v] = u;

# key[v] = this.adjacencyMatrix[u][v];

# }

# }

# }

# this.printMST(parent);

# }

# minKey(key, mstSet) {

# let min = Number.MAX\_VALUE;

# let minIndex = -1;

# for (let v = 0; v < this.vertices; v++) {

# if (mstSet[v] === false && key[v] < min) {

# min = key[v];

# minIndex = v;

# }

# }

# return minIndex;

# }

# printMST(parent) {

# console.log("Edge \tWeight");

# for (let i = 1; i < this.vertices; i++) {

# console.log(`${parent[i]} - ${i}\t${this.adjacencyMatrix[i][parent[i]]}`);

# }

# }

# }

# const g = new Graph(5);

# g.addEdge(0, 1, 2);

# g.addEdge(0, 3, 6);

# g.addEdge(1, 2, 3);

# g.addEdge(1, 3, 8);

# g.addEdge(1, 4, 5);

# g.addEdge(2, 4, 7);

# g.addEdge(3, 4, 9);

# g.primMST();

# Output

# Edge Weight

# 0 - 1 2

# 1 - 2 3

# 0 - 3 6

# 1 - 4 5

Assignment- 17

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 17: Implement regular expression matching using dynamic programming.

# Solution:

# function isMatch(s, p) {

# const memo = {};

# function dp(i, j) {

# if (memo[i + '-' + j] !== undefined) {

# return memo[i + '-' + j];

# }

# if (j === p.length) {

# return i === s.length;

# }

# const firstMatch = i < s.length && (p[j] === s[i] || p[j] === '.');

# if (j + 1 < p.length && p[j + 1] === '\*') {

# memo[i + '-' + j] = (dp(i, j + 2) || (firstMatch && dp(i + 1, j)));

# } else {

# memo[i + '-' + j] = firstMatch && dp(i + 1, j + 1);

# }

# return memo[i + '-' + j];

# }

# return dp(0, 0);

# }

# // Example usage:

# const inputString = "aa";

# const pattern = "a\*";

# const result = isMatch(inputString, pattern);

# console.log(result);

# Output

# true

Assignment- 18

Name: Ganesh Vitthal Nagpure

Class: MCA-I

Roll No.: MC232540

# Practical 18: Implement N queen’s problem using backtracking.

# Solution:

# function solveNQueens(n) {

# const result = [];

# const board = Array.from(Array(n), () => Array(n).fill('.'));

# const isValid = (row, col) => {

# // Check if there is a queen in the same column

# for (let i = 0; i < row; i++) {

# if (board[i][col] === 'Q') {

# return false;

# }

# }

# // Check upper diagonal on the left side

# for (let i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--) {

# if (board[i][j] === 'Q') {

# return false;

# }

# }

# // Check upper diagonal on the right side

# for (let i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++) {

# if (board[i][j] === 'Q') {

# return false;

# }

# }

# return true;

# };

# const solve = (row) => {

# if (row === n) {

# const solution = board.map((row) => row.join(''));

# result.push(solution);

# return;

# }

# for (let col = 0; col < n; col++) {

# if (isValid(row, col)) {

# board[row][col] = 'Q';

# solve(row + 1);

# board[row][col] = '.';

# }

# }

# };

# solve(0);

# return result;

# }

# const n = 4;

# const solutions = solveNQueens(n);

# console.log(`Solutions for ${n}-Queens:`);

# solutions.forEach((solution, index) => {

# console.log(`Solution ${index + 1}:`);

# solution.forEach((row) => console.log(row));

# console.log('\n');

# });

# Output

# Solutions for 4-Queens:

# Solution 1:

# .Q..

# ...Q

# Q...

# ..Q.

# Solution 2:

# ..Q.

# Q...

# ...Q

# .Q.