**A**

**Practical**

**On**

“Data Structure

&

Algorithm ”

**Submitted By**

**Mr. Mokal Dhiraj Sudhakar**

**PIRENS Institute of Business Management and Administration, Loni Bk**

Department of MCA

**2022-23**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Programs Title** | **Page No** | **Date** | **Sign** |
| 1 | Demonstrate singly and doubly linked list |  |  |  |
| 2 | STACK implementation using Array with PUSH, POP operations |  |  |  |
| 3 | Reverse a string using stack |  |  |  |
| 4 | Check for balanced parentheses by using Stacks |  |  |  |
| 5 | Implement Stack using Linked List |  |  |  |
| 6 | Demonstration of Linear Queue, Circular Queue, Priority Queue |  |  |  |
| 7 | Reverse stack using queue |  |  |  |
| 8 | Practical based on binary search tree implementation with its operations |  |  |  |
| 9 | Graph implementation and graph traversals |  |  |  |
| 10 | Implementation of Hashing |  |  |  |
| 11 | Practical based on Brute Force technique |  |  |  |
| 12 | Practical based on Greedy Algorithm-Prim's |  |  |  |
| 13 | Practical based on Divide and Conquer Technique- Binary Search. Tower of Hanoi |  |  |  |
| 14 | Implementation of Dynamic Programming- LCS. Regular Expression Matching |  |  |  |
| 15 | Practical based on backtracking-N Queen's problems |  |  |  |

**INDEX**

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Demonstrate singly and doubly linked list |

1. **Singly Linked List (Code)**

class Node {

constructor(data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

constructor() {

this.head = null;

this.size = 0;

}

insert\_first(data) {

let node = new Node(data);

if (this.head == null) {

this.head = node;

} else {

node.next = this.head; this.head = node;

}

this.size++;

}

insert\_last(data) {

let node = new Node(data);

if (this.head == null) {

this.head = node;

return;

} let current = this.head;

while (current.next) {

current = current.next;

}

current.next = node;

this.size++;

}

insertAt(data, index) {

if (index < 0 || index > this.size) {

console.log("Please enter a valid index");

return;

} else {

let node = new Node(data);

let curr, prev; curr = this.head;

if (index == 0) {

node.next = this.head; this.head = node;

} else {

curr = this.head; let it = 0;

while (it < index) {

it++; prev = curr; curr = curr.next;

}

node.next = curr; prev.next = node;

}

this.size++;

console.log(`inserted successfully number ${data} at index ${index}`);

}

}

displayList() {

let current = this.head;

while (current) {

console.log(current.data + " ");

current = current.next;

}

}

removeFrom(index) {

if (index < 0 || index >= this.size) {

console.log("Please Enter a valid index");

return;

} else {

let curr, prev, it = 0;

curr = this.head; prev = curr;

if (index === 0) {

this.head = curr.next;

} else {

while (it < index) {

it++; prev = curr; curr = curr.next;

}

prev.next = curr.next;

}

this.size--;

return curr.data;

}

}

removeElement(element) {

let current = this.head;

let prev = null;

while (current != null) {

if (current.data === element) {

if (prev == null) {

this.head = current.next;

} else {

prev.next = current.next;

}

this.size--;

return current.data;

}

prev = current;

current = current.next;

}

return -1;

}

search(data) {

let current = this.head;

let flag = 0;

let position = 1;

while (current) {

if (current.data == data) {

flag = 1;

break;

}

position++;

current = current.next;

}

if (flag == 1)

console.log(data + " data is found at " + position + " position");

else

console.log(data + " is not found in the list")

}

reverse() {

let p1, p2, p3;

if (this.head.next == null)

return;

p1 = this.head; p2 = p1.next;

p3 = p2.next; p1.next = null; p2.next = p1;

while (p3 != null) {

p1 = p2; p2 = p3; p3 = p3.next; p2.next = p1;

}

this.head = p2;

}

count() {

console.log(this.size);

}

}

const linkedlist = new LinkedList();

linkedlist.insert\_first(9);

linkedlist.insert\_first(8);

linkedlist.insert\_last(2);

linkedlist.insert\_last(4);

linkedlist.insertAt(23,2);

linkedlist.removeFrom(3)

linkedlist.removeElement(2)

linkedlist.search(23)

linkedlist.reverse()

linkedlist.count();

linkedlist.displayList();

1. **Doubly Linked List (Code)**

class Node {

constructor(element) {

this.element = element;

this.next = null;

this.prev = null;

}

}

class LinkedList {

constructor() {

this.head = null;

this.size = 0;

}

create\_list(element) {

var node = new Node(element);

var current;

if (this.head == null) {

node.prev = null;

this.head = node;

} else {

current = this.head;

while (current.next) {

current = current.next;

}

current.next = node;

node.prev = current;

}

this.size++;

}

insertToBegin(element) {

var node = new Node(element);

var current;

current = this.head;

node.next = this.head;

this.head.prev = node;

this.head = node;

this.size++;

}

insertToLast(element) {

var node = new Node(element);

var curr;

curr = this.head;

var i = 0;

while (curr.next != null)

curr = curr.next;

curr.next = node;

node.prev = curr;

this.size++;

}

insertAt(element, index) {

if (index > 0 && index > this.size)

return false;

else {

var node = new Node(element);

var curr, curr1;

curr = this.head;

if (index == 0) {

node.next = this.head;

this.head.prev = node;

this.head = node;

} else {

curr = this.head;

curr1 = curr.next;

var it = 1;

while (it < index) {

it++;

curr1 = curr1.next;

curr = curr.next;

}

curr1.prev = node;

node.next = curr1;

node.prev = curr;

curr.next = node;

}

this.size++;

}

}

displayListForward() {

var current = this.head;

while (current) {

console.log(current.element + " ");

current = current.next;

}

}

displayListBackward() {

var current = this.head;

while (current.next != null)

current = current.next;

while (current) {

console.log(current.element + " ");

current = current.prev;

}

}

deleteFrom(index) {

if (index < 0 && index > this.size)

return -1;

else {

var curr, previous, it = 0;

curr = this.head;

previous = curr;

if (index == 1) {

this.head = curr.next;

this.head.prev = null;

} else {

while (it < index - 1) {

it++;

previous = curr;

curr = curr.next;

}

previous.next = curr.next;

}

this.size--;

return curr.element;

}

}

deleteElement(data) {

if (this.head.element == data) {

var value = this.head.element;

this.head = this.head.next;

this.head.prev

= null;

return value;

}

var previous = this.head;

var current

= previous.next;

while (current.next != null) {

if (current.element == data) {

previous.next = current.next;

current.next.prev = previous;

this.size--;

return current.element;

}

previous = current; current = current.next;

}

if (current.element == data) {

previous.next = null;

this.size--;

return current.element;

}

return -1;

}

search(data) {

var current = this.head;

var flag = 0; var position = 1;

while (current) {

if (current.element == data) {

flag = 1;

break;

}

position++;

current = current.next;

}

if (flag == 1)

console.log(data + "element is found at" + position + " position");

else

console.log(data + " is not found in the list")

}

count() {

return this.size;

}

}

var ll = new LinkedList();

ll.create\_list(10);

ll.displayListForward();

console.log("Insertion at Beginning :");

ll.insertToBegin(65);

ll.displayListForward();

console.log("Insertion at last :");

ll.insertToLast(50);

ll.displayListForward();

console.log("Insertion at Specified Position :");

ll.insertAt(44, 2);

ll.displayListForward();

ans = ll.deleteElement(10);

if (ans == -1)

console.log("Element" + data + " is not found in the list");

else {

console.log("Deleted Element is =" + ans);

ll.displayListForward();

}

console.log("Deletion from Position :");

ans = ll.deleteFrom(1);

if (ans == -1)

console.log("position is not within the range ");

else {

console.log("Deleted Element is =" + ans);

ll.displayListForward();

}

ll.search(10);

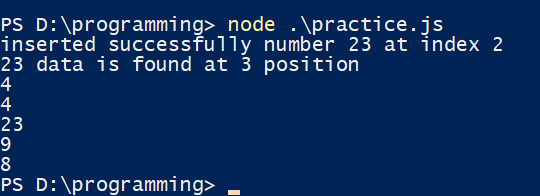
console.log("Total number of nodes in the linked list = " + ll.count());

ll.displayListForward();

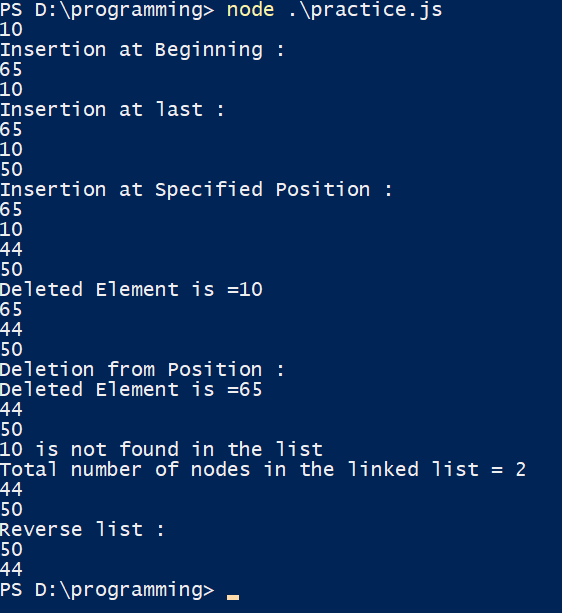
console.log("Reverse list :");

ll.displayListBackward();

**Singly Linked List (Output)**

****

**Doubly Linked List (Output)**

****

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** STACK implementation using Array with PUSH, POP operations |

1. **Stack (Code)**

class Stack {

constructor(size) {

this.items = [];

this.top = -1;

this.size = size;

}

push(element) {

if (this.top == this.size - 1) {

console.log("stack is Full")

return;

}

this.items[++(this.top)] = element;

}

pop() {

if (this.items.length == 0)

return "Underflow";

let ch = this.items[this.top]; this.top = this.top - 1;

console.log("poped element = " + ch);

}

display() {

let i;

for (i = 0; i <= this.top; i++)

console.log(this.items[i]);

}

}

let stack = new Stack(3);

stack.push(17);

stack.push(69);

stack.push(13);

stack.push(4);

stack.display();

stack.pop();

stack.display();

**Stack Array (Push, Pop) Output**

****

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Reverse a string using stack |

1. **Stack (Reverse String) Code:**

function reverse(str) {

let stack = [];

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

stack.push(str[i]);

}

let reverseStr = '';

while (stack.length > 0) {

reverseStr += stack.pop();

}

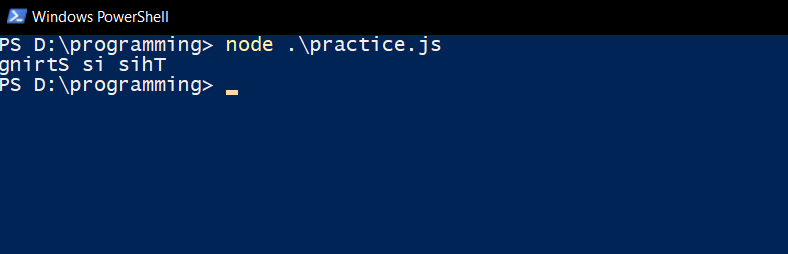
return reverseStr;

}

let str = "This is String";

console.log(reverse(str));

**Stack (Reverse String) Output**



|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Check for balanced parentheses by using Stacks |

1. **Check balanced Parenthesis (stack) Code:**

class Stack {

constructor() {

this.items = [];

this.top = -1;

}

push(element) {

this.items[++(this.top)] = element;

}

pop() {

if (this.items.length == 0)

return "Underflow";

let ch = this.items[this.top--];

return ch;

}

}

let stack = new Stack();

let valid = true;

let temp;

"use strict"

const ps = require("prompt-sync")

const prompt = ps();

let exp = prompt('Enter the Expression which is to be checked :');

for (var i = 0; i < exp.length; i++) {

if (exp[i] == '(' || exp[i] == '{' || exp[i] == '[')

stack.push(exp[i]);

if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']')

if (stack.top == -1)

valid = false;

else {

temp = stack.pop();

if (exp[i] == ')' && (temp == '{' || temp == '['))

valid = false;

if (exp[i] == '}' && (temp == '(' || temp == '['))

valid = false;

if (exp[i] == ']' && (temp == '(' || temp == '{'))

valid = false;

}

}

if (stack.top >= 0)

valid = false;

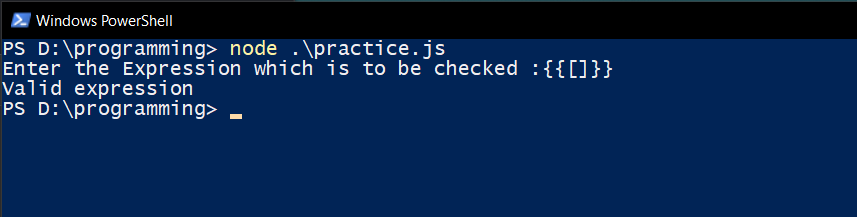
if (valid == true)

console.log("Valid expression ");

else

console.log("Invalid expression ");

**Check balanced Parenthesis (stack) Output:**

****

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title**: Implement Stack using Linked List |

1. **Implement Stack (Linked List) Code:**

class Node {

constructor(element) {

this.element = element;

this.next = null

}

}

class stack {

constructor() {

this.head = null;

this.size = 0;

}

push(element) {

let value, data;

let top = new Node(element);

if (this.head == null)

this.head = top;

else {

top.next = this.head;

this.head = top;

}

}

pop() {

let item;

if (this.head == null) {

console.log("Stack Undeflow");

} else {

item = this.head.element;

this.head = this.head.next;

console.log("Item popped is" + item);

}

}

display() {

if (this.head == null) {

console.log("Stack is Empty");

return;

}

let top = this.head;

console.log("Elements in the Stack are");

while (top) {

console.log(top.element + " ");

top = top.next;

}

}

}

let ll = new stack();

ll.push(10);

ll.push(20);

ll.push(30);

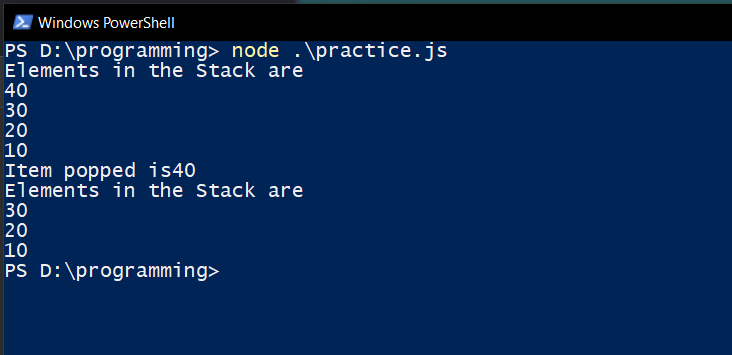
ll.push(40);

ll.display();

ll.pop();

ll.display();

**Implement Stack (Linked List) Output:**

****

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Demonstration of Linear Queue, Circular Queue, Priority Queue |

1. **Linear Queue, Circular Queue, Priority Queue Code:**
2. **Linear Queue**

class Queue {

constructor(size) {

this.items = [];

this.rear = -1;

this.front = -1;

this.size = size;

}

insert(element) {

if (this.rear == this.size - 1) {

console.log("Queue Overflow");

return;

}

if (this.rear == -1)

this.front = 0;

this.rear = this.rear + 1;; this.items[this.rear] = element;

}

dequeue() {

if (this.front == -1 || this.front == this.rear + 1) {

console.log("Queue Underflow");

return;

}

let ch = this.items[this.front]; this.front = this.front + 1; console.log("deleted data = " +

ch);

}

display() {

let i;

for (i = this.front; i <= this.rear; i++) {

console.log(this.items[i]);

}

}

}

let queue = new Queue(5);

queue.insert(10);

queue.insert(20);

queue.insert(30);

queue.insert(40);

queue.display();

queue.dequeue();

queue.display();

1. **Circular Queue**

const Queue = function (maxSize) {

this.queue = [];

this.reset = function () {

this.tail = -1;

this.head = -1;

};

this.reset();

this.maxSize = maxSize || Queue.MAX\_SIZE;

this.increment = function (number) {

return (number + 1) % this.maxSize;

};

};

Queue.MAX\_SIZE = Math.pow(2, 53) - 1;

Queue.prototype.enQueue = function (record) {

if (this.isFull()) {

throw new Error("Queue is full can't add new records");

}

if (this.isEmpty()) {

this.head = this.increment(this.head);

}

this.tail = this.increment(this.tail);

this.queue[this.tail] = record;

};

Queue.prototype.setMaxSize = function (maxSize) {

this.maxSize = maxSize;

};

Queue.prototype.push = Queue.prototype.enQueue;

Queue.prototype.insert = Queue.prototype.enQueue;

Queue.prototype.isFull = function () {

return this.increment(this.tail) === this.head;

};

Queue.prototype.deQueue = function () {

if (this.isEmpty()) {

throw new Error("Can't remove element from an empty Queue");

}

var removedRecord = this.queue[this.head];

this.queue[this.head] = null;

if (this.tail === this.head) {

this.reset();

} else {

this.head = this.increment(this.head);

}

return removedRecord;

};

Queue.prototype.pop = Queue.prototype.deQueue;

Queue.prototype.front = function () {

return this.queue[this.head] || null;

};

Queue.prototype.peak = Queue.prototype.front;

Queue.prototype.isEmpty = function () {

return this.tail === -1 && this.head === -1;

};

Queue.prototype.print = function () {

for (var i = this.head; i <= this.tail; i++) {

console.log(this.queue[i]);

}

};

var q = new Queue(5);

q.enQueue(1);

q.enQueue(2);

q.enQueue(3);

q.enQueue(4);

q.deQueue();

q.deQueue();

q.deQueue();

q.enQueue(5);

q.enQueue(6);

q.enQueue(7);

q.enQueue(8);

q.deQueue();

q.deQueue();

q.deQueue();

q.deQueue();

q.deQueue();

console.clear();

q.print();

console.log("head", q.head);

console.log("tail", q.tail);

console.log(q.queue);

1. **Priority Queue**

class Node {

constructor(priority, element) {

this.priority = priority;

this.element = element;

this.next = null

}

}

class priority\_queue {

constructor() {

this.front = null;

this.size = 0;

}

insert(priority, element) {

let temp = new Node(priority, element);

let q;

if (this.front == null || priority < this.front.priority) {

temp.next = this.front;

this.front = temp;

} else {

q = this.front;

while (q.next != null && q.next.priority <= priority) q = q.next;

temp.next = q.next;

q.next = temp;

}

}

delete() {

if (this.front == null)

console.log("Queue underflow");

else {

console.log("Deleted item is" + this.front.element);

this.front = this.front.next;

}

}

displayqueue() {

if (this.front == null)

console.log("Queue is empty ");

else {

var current = this.front;

while (current) {

console.log(current.element + " "); current = current.next;

}

}

}

}

const pq = new priority\_queue();

pq.insert(3, 1);

pq.insert(2, 3);

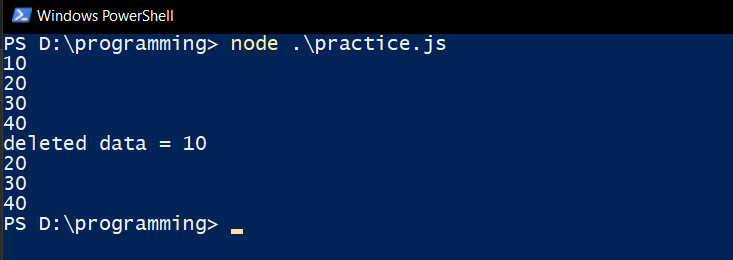
pq.insert(1, 2);

pq.displayqueue();

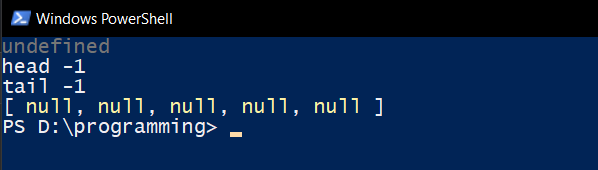
pq.delete();

pq.displayqueue();

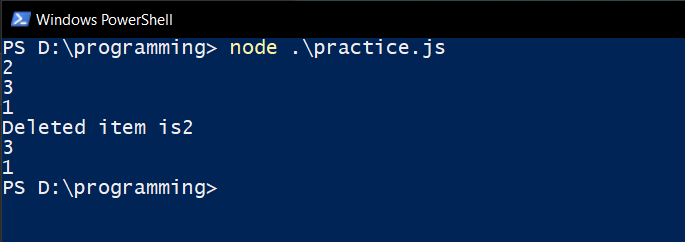
1. **Linear Queue (Output):**



1. **Circular Queue (Output):**



1. **Priority Queue (Output):**



|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Reverse stack using queue |

1. **Reverse Stack (Queue) Code:**

class Stack {

constructor() {

this.elements = [];

}

push(element) {

this.elements.push(element);

}

pop() {

if (this.isEmpty()) return "Underflow situation";

else return this.elements.pop();

}

isEmpty() {

return this.elements.length == 0;

}

print() {

return this.elements;

}

}

class Queue {

constructor() {

this.elements = [];

}

enqueue(element) {

this.elements.push(element)

}

dequeue() {

if (!this.isEmpty()) {

return this.elements.shift();

} else {

return 'Underflow situation';

}

}

isEmpty() {

return this.elements.length == 0;

}

}

function reverse(stack) {

const queue = new Queue();

while (!stack.isEmpty()) {

queue.enqueue(stack.pop());

}

while (!queue.isEmpty()) {

stack.push(queue.dequeue());

}

}

const stack = new Stack();

stack.push('Welcome');

stack.push('There');

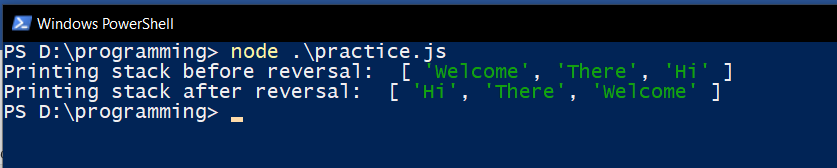
stack.push('Hi');

console.log('Printing stack before reversal: ', stack.print());

reverse(stack);

console.log('Printing stack after reversal: ', stack.print());

**Reverse Stack (Queue) Output:**



|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Practical based on binary search tree implementation with its operations |

1. **Binary Search tree implementation and its operations**

class Node {

constructor(data) {

this.data = data;

this.left = null;

this.right = null;

}

}

class BinarySearchTree {

constructor() {

this.root = null;

}

insert(data) {

const newNode = new Node(data);

if (this.root === null) {

this.root = newNode;

} else {

this.insertNode(this.root, newNode);

}

}

insertNode(node, newNode) {

if (newNode.data < node.data) {

if (node.left === null) {

node.left = newNode;

} else {

this.insertNode(node.left, newNode);

}

} else {

if (node.right === null) {

node.right = newNode;

} else {

this.insertNode(node.right, newNode);

}

}

}

remove(data) {

this.root = this.removeNode(this.root, data);

}

removeNode(node, key) {

if (node === null) {

return null;

} else if (key < node.data) {

node.left = this.removeNode(node.left, key);

return node;

} else if (key > node.data) {

node.right = this.removeNode(node.right, key);

return node;

} else {

if (node.left === null && node.right === null) {

node = null;

return node;

} else if (node.left === null) {

node = node.right;

return node;

} else if (node.right === null) {

node = node.left;

return node;

}

const aux = this.findMinNode(node.right);

node.data = aux.data;

node.right = this.removeNode(node.right, aux.data);

return node;

}

}

findMinNode(node) {

if (node.left === null) {

return node;

} else {

return this.findMinNode(node.left);

}

}

getRootNode() {

return this.root;

}

inorder(node) {

if (node !== null) {

this.inorder(node.left);

console.log(node.data);

this.inorder(node.right);

}

}

preorder(node) {

if (node !== null) {

console.log(node.data);

this.preorder(node.left);

this.preorder(node.right);

}

}

postorder(node) {

if (node !== null) {

this.postorder(node.left);

this.postorder(node.right);

console.log(node.data);

}

}

search(node, data) {

if (node === null) {

return null;

} else if (data < node.data) {

return this.search(node.left, data);

} else if (data > node.data) {

return this.search(node.right, data);

} else {

return node;

}

}

}

const BST = new BinarySearchTree();

BST.insert(5);

BST.insert(3);

BST.insert(1);

BST.insert(4);

BST.insert(7);

BST.insert(6);

BST.insert(9);

const root = BST.getRootNode();

console.log("inorder traversal:");

BST.inorder(root);

console.log("remove data:");

BST.remove(7);

console.log("inorder traversal:");

BST.inorder(root);

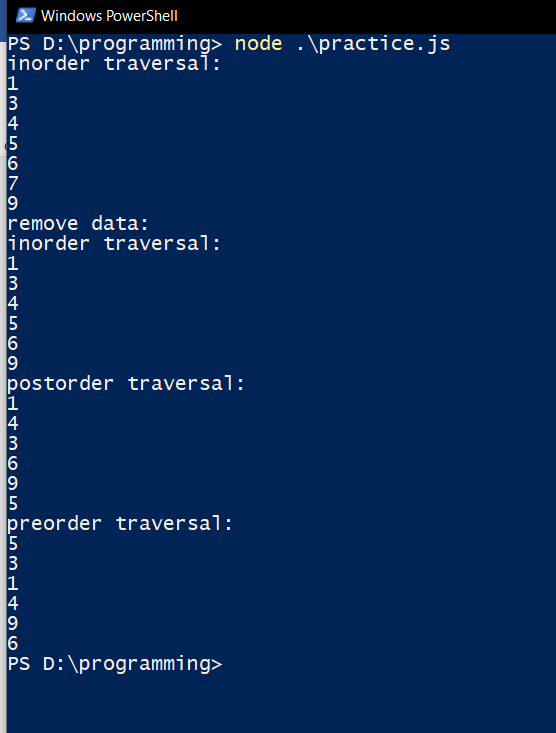
console.log("postorder traversal:");

BST.postorder(root);

console.log("preorder traversal:");

BST.preorder(root);

**Binary Search tree implementation and its operations**

****

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Graph implementation and graph traversals |

1. **Graph implementation and Graph traversal (Code):**

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 get\_keys = this.AdjList.keys();

for (let i of get\_keys) {

let get\_values = this.AdjList.get(i);

let conc = "";

for (let j of get\_values)

conc += j + " ";

console.log(i + "> " + conc);

}

}

dfs(startingNode) {

let visited = {};

this.DFSUtil(startingNode, visited);

}

DFSUtil(vert, visited) {

visited[vert] = true;

console.log(vert);

let get\_neighbours = this.AdjList.get(vert);

for (let i in get\_neighbours) {

let get\_elem = get\_neighbours[i];

if (!visited[get\_elem])

this.DFSUtil(get\_elem, visited);

}

}

}

let g = new Graph(6);

let vertices = ['A', 'B', 'C', 'D', 'E', 'F'];

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

g.addVertex(vertices[i]);

}

g.addEdge('A', 'B');

g.addEdge('A', 'D');

g.addEdge('A', 'E');

g.addEdge('B', 'C');

g.addEdge('D', 'E');

g.addEdge('E', 'F');

g.addEdge('E', 'C');

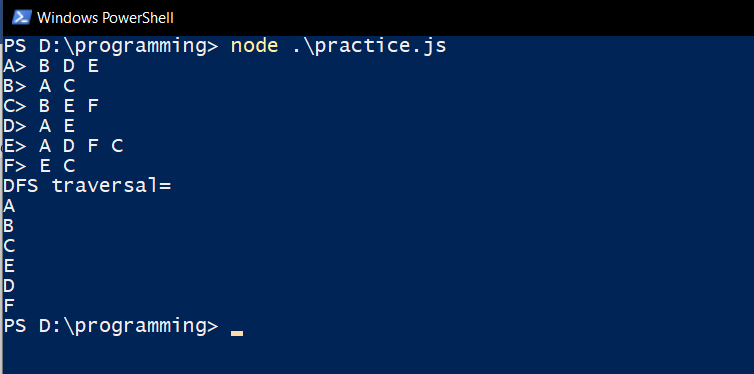
g.addEdge('C', 'F');

g.printGraph();

console.log("DFS traversal=");

g.dfs('A')

**Graph implementation and Graph traversal (Code):**



|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Implementation of Hashing |

1. **Implementation of Hashing (Code):**

class HashTable {

constructor(size) {

this.values = {};

this.size = size;

}

add(key, value) {

const hash = this.genHash(key)

if (!this.values.hasOwnProperty(hash))

this.values[hash] = {};

this.values[hash][key] = value;

}

remove(key) {

const hash = this.genHash(key);

if (this.values.hasOwnProperty(hash) &&

this.values[hash].hasOwnProperty(key)) {

delete this.values[hash][key];

}

}

genHash(key) {

let keyStr = key.toString();

let sum = 0;

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

sum += keyStr.charCodeAt(i);

return sum % this.size

}

getValue(key) {

const hash = this.genHash(key);

if (this.values.hasOwnProperty(hash)

&& this.values[hash].hasOwnProperty(key))

return this.values[hash][key];

else

return undefined;

}

printAll() {

for (const val in this.values)

for (const key in this.values[val])

console.log("{", key, ", ", this.values[val][key], "}")

}

}

let hashTable = new HashTable(5);

hashTable.add("key1", "value1");

hashTable.add("key2", "value2");

hashTable.add("key3", "value3");

hashTable.printAll();

console.log(`value of key3: `, hashTable.getValue("key3"))

console.log(`delete key3`)

hashTable.remove("key3")

hashTable.printAll();

console.log(`value of key3: `, hashTable.getValue("key3"))

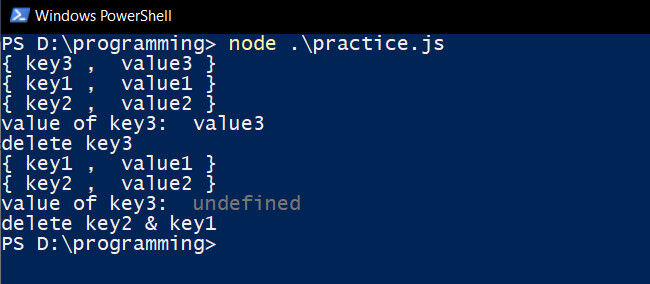
console.log(`delete key2 & key1`)

hashTable.remove("key2")

hashTable.remove("key1")

hashTable.printAll();

**Implementation of Hashing (Output):**



|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Practical based on Brute Force technique |

1. **Brute Force technique**

function search(arr, search\_Element) {

let left = 0; let length = arr.length;

let right = length - 1; let position = -1;

for (left = 0; left <= right;) {

if (arr[left] == search\_Element) {

position = left;

console.log("Element found in Array at " + (position + 1) + " Position with " + (left + 1) + " Attempt");

break;

}

if (arr[right] == search\_Element) {

position = right;

console.log("Element found in Array at " + (position + 1) + " Position with " + (length - right) + " Attempt");

break;

}

left++; right--;

}

if (position == -1)

console.log("Not found in Array with " + left + " Attempt");

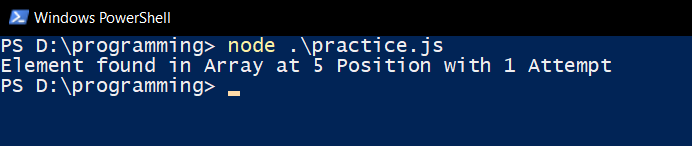
}

let arr = [1, 2, 3, 4, 5];

let search\_element = 5;

search(arr, search\_element);

**Brute Force (Output):**



|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Practical based on Greedy Algorithm-Prim's |

1. **Prim’s Algorithm**

function createAdjMatrix(V, G) {

let adjMatrix = [];

for (let i = 0; i < V; i++) {

adjMatrix.push([]);

for (let j = 0; j < V; j++) { adjMatrix[i].push(0); }

}

console.log(G)

console.log("graph length=" + G.length)

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

adjMatrix[G[i][0]][G[i][1]] = G[i][2];

adjMatrix[G[i][1]][G[i][0]] = G[i][2];

}

return adjMatrix;

}

function prims(V, G) {

let adjMatrix = createAdjMatrix(V, G);

let vertex = 0; let MST = []; let edges = []; let visited = [];

let minEdge = [null, null, Infinity];

while (MST.length !== V - 1) {

visited.push(vertex);

for (let r = 0; r < V; r++) {

if (adjMatrix[vertex][r] !== 0) {

edges.push([vertex, r, adjMatrix[vertex][r]]);

}

}

for (let e = 0; e < edges.length; e++) {

if (edges[e][2] < minEdge[2] && visited.indexOf(edges[e][1]) === -1) {

minEdge = edges[e];

}

}

edges.splice(edges.indexOf(minEdge), 1);

MST.push(minEdge);

vertex = minEdge[1];

minEdge = [null, null, Infinity];

}

return MST;

console.log(MST)

}

let a = 0, b = 1, c = 2, d = 3;

let graph = [

[a, b, 2],

[a, c, 3],

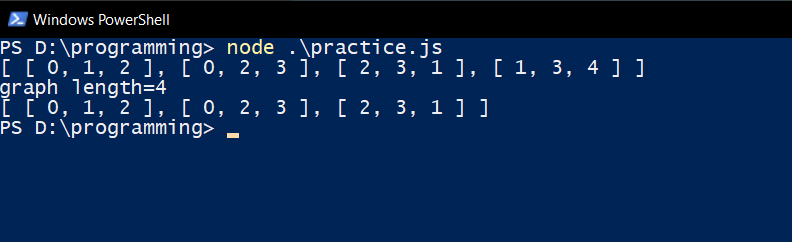
[c, d, 1],

[b, d, 4]

];

console.log(prims(4, graph));

**Prim’s Algorithm (Output):**

****

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Practical based on Divide and Conquer Technique- Binary Search. Tower of Hanoi |

1. **Divide and conquer (Binary search, Tower of honoi)**
2. **Binary Search (Code):**

function binarySearch(arr, l, r, x) {

if (r >= l) {

let mid = l + Math.floor((r - l) / 2);

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

return binarySearch(arr, mid + 1, r, x);

}

return -1;

}

let arr = [2, 3, 4, 10, 40], x = 10, n = arr.length, result = binarySearch(arr, 0, n - 1, x);

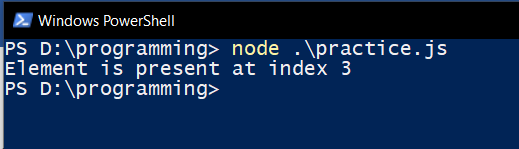
if (result == -1)

console.log("Element is not present in array")

else

console.log("Element is present at index " + result);

**Binary Search (Output):**

****

1. **Tower of Honoi (Code):**

function towerOfHanoi(n, from\_rod, to\_rod, aux\_rod) {

if (n == 1) {

console.log("Move disk 1 from rod " + from\_rod + " to rod " + to\_rod);

return;

}

towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

console.log("Move disk " + n + " from rod " + from\_rod + " to rod " + to\_rod);

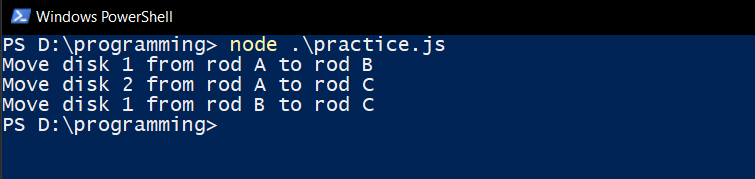
towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

let n = 2;

towerOfHanoi(n, 'A', 'C', 'B');

**Tower of Honoi (Output):**

****

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Implementation of Dynamic Programming- LCS. Regular Expression Matching |

1. **Dynamic Programming (LCS) Coding:**

function longest\_common\_starting\_substring(arr1) {

let arr = arr1.concat().sort(),

a1 = arr[0],

a2 = arr[arr.length - 1],

L = a1.length,

i = 0;

while (i < L && a1.charAt(i) === a2.charAt(i)) i++;

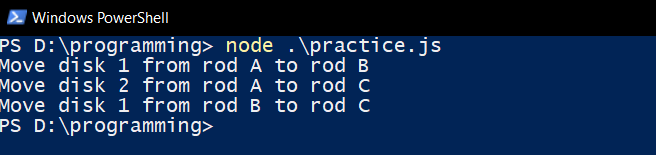
return a1.substring(0, i);

}

console.log(longest\_common\_starting\_substring(['go', 'google']));

console.log(longest\_common\_starting\_substring(['SQLInjection', 'SQLTutorial']));

**Dynamic Programming (LCS) Output:**

****

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Seat Number:** MC022057 **Date:** / / **Sign:** |
| **Student Name:** Mokal Dhiraj Sudhakar |
| **Subject Name:** IT12 – Datastructure And Algorithm |
| **Program Title:** Practical based on backtracking-N Queen's problems |

1. **N Queen’s Problem (Code):**

class N\_queen\_problem {

constructor(size) {

this.size = size;

this.board = [];

for (let i = 0; i < size; i++) {

this.board.push([]);

for (let j = 0; j < size; j++) {

this.board[i][j] = 0;

}

}

}

is\_attack(i, j) {

let k, l;

for (k = 0; k < this.size; k++) {

if ((this.board[i][k] == 1) || (this.board[k][j] == 1))

return 1;

}

for (k = 0; k < this.size; k++) {

for (l = 0; l < this.size; l++) {

if (((k + l) == (i + j)) || ((k- l) == (i - j))) {

if (this.board[k][l] == 1)

return 1;

}

}

}

return 0;

}

N\_queen(n) {

let i, j;

if (n == 0)

return 1;

for (i = 0; i < this.size; i++) {

for (j = 0; j < this.size; j++) {

{

this.board[i][j] = 1;

if (this.N\_queen(n - 1) == 1) {

return 1;

}

this.board[i][j] = 0;

}

}

}

return 0;

}

printBoard() {

for (let i = 0; i < this.size; i++) {

let row = '';

for (let j = 0; j < this.size; j++) {

row += ` ${this.board[i][j]}`;

}

console.log(row);

}

}

}

"use strict"

const ps = require("prompt-sync");

const prompt = ps();

let n = prompt("Enter the value of N for NxN chessboard");

let nQueen = new N\_queen\_problem(n);

nQueen.N\_queen(n);

nQueen.printBoard();

**N Queen’s Problem (Output):**

