**AP EXP – 06**

**Stack-Based Implementations:-**

**Q1. Implement Queue using Stack**

**Code:**

#include <bits/stdc++.h>

using namespace std;

class MyQueue {

stack<int> s1, s2;

public:

MyQueue() { }

void push(int x) {

s1.push(x);

}

int pop() {

if (s2.empty()) {

while (!s1.empty()) {

s2.push(s1.top());

s1.pop();

} }

if (s2.empty()) return -1;

int x = s2.top();

s2.pop();

return x;

}

int peek() {

if (s2.empty()) {

while (!s1.empty()) {

s2.push(s1.top());

s1.pop();

} }

if (s2.empty()) return -1;

return s2.top();

}

bool empty() {

return s1.empty() && s2.empty();

} };

int main() {

MyQueue q;

q.push(1);

q.push(2);

cout << q.peek() << endl; // Output: 1

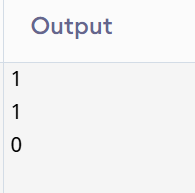
cout << q.pop() << endl; // Output: 1

cout << q.empty() << endl; // Output: 0 (false)

return 0;

}

**Output:**

****

**Q5. Implement Priority Queue using Stack**

**Code:**

#include <iostream>

#include <stack>

using namespace std;

class PriorityQueueUsingStack {

stack<int> s;

public:

void push(int x) {

stack<int> temp;

while (!s.empty() && s.top() > x) {

temp.push(s.top()); s.pop();

}

s.push(x);

while (!temp.empty()) {

s.push(temp.top()); temp.pop();

} }

int pop() {

if (s.empty()) return -1;

int x = s.top();

s.pop();

return x;

}

bool empty() {

return s.empty();

}

int top() {

if (s.empty()) return -1;

return s.top();

} };

int main() {

PriorityQueueUsingStack pq;

pq.push(5);

pq.push(1);

pq.push(3);

cout << pq.pop() << " "; // 1

cout << pq.pop() << " "; // 3

cout << pq.pop() << " "; // 5

return 0; }

Output:

****

**Queue-Based Implementations:-**

**Q8. Implement Stack using Queue**

**Code:**

class MyStack {

public:

queue<int> q1,q2 ;

MyStack() {

}

void push(int x) {

if(empty()){

q1.push(x);

}else if(q1.empty()){

q2.push(x);

}else{

q1.push(x);

} }

int pop() {

if(empty()) return 0;

else if(q1.empty()){

while(q2.size() != 1){

q1.push(q2.front());

q2.pop(); }

int temp = q2.front();

q2.pop();

return temp;

}else{

while(q1.size() != 1){

q2.push(q1.front());

q1.pop();

}

int temp = q1.front();

q1.pop();

return temp;

} }

int top() {

if(empty()) return 0;

else if(q1.empty()){

return q2.back();

}else{

return q1.back();//element from back of Queue

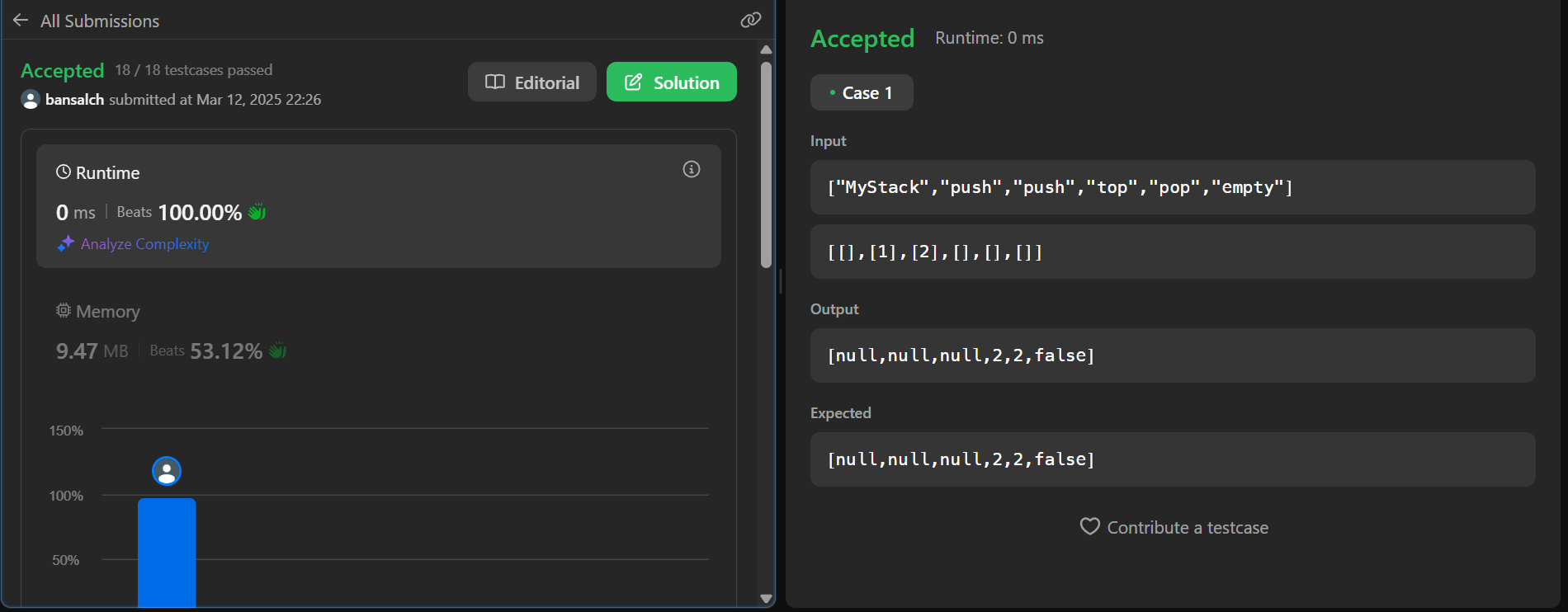
}}

bool empty() {

return q1.empty() && q2.empty();

} };

Output:



**Q10. Implement Circular Queue using Queue**

**Code:**

class MyCircularQueue {

private:

vector<int> queue;

int front;

int rear;

int n;

public:

MyCircularQueue(int k) {

n = k;

queue.resize(n);

front = rear = -1;

}

bool enQueue(int value) {

if ((rear + 1) % n == front) {

return false;

}

if (rear == -1) {

front = 0;

rear = 0;

} else {

rear = (rear + 1) % n;

}

queue[rear] = value;

return true;

}

bool deQueue() {

if (front == -1) {

return false;

}

if (front == rear) {

front = rear = -1;

} else {

front = (front + 1) % n;

}

return true;

}

int Front() {

if (front == -1) {

return -1;

}

return queue[front];

}

int Rear() {

if (rear == -1) {

return -1; }

return queue[rear]; }

bool isEmpty() {

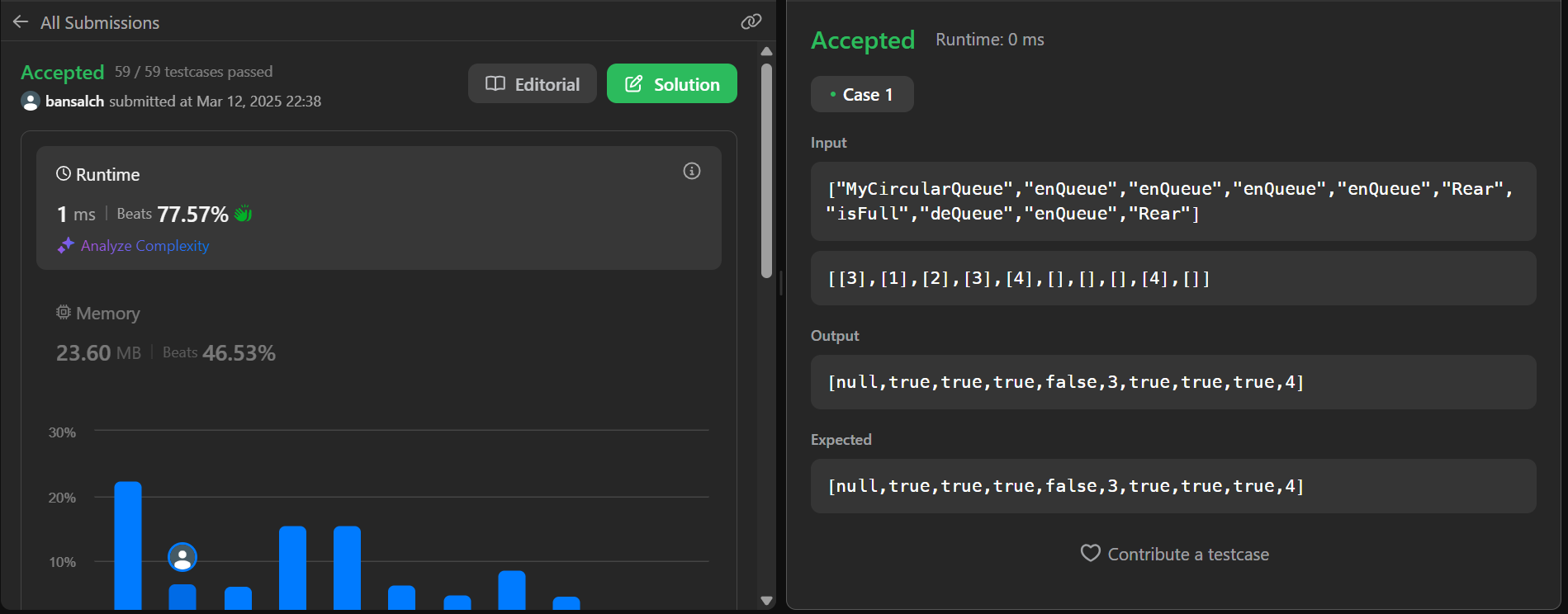
return front == -1; }

bool isFull() {

return (rear + 1) % n == front;

} };

**Output:**

****

**Array-Based Implementations:-**

**Q17. Implement Two Stacks in One Array**

**Code:**

#include <iostream>

using namespace std;

class TwoStacks {

int \*arr, top1, top2, size;

public:

TwoStacks(int n) {

size = n;

arr = new int[n];

top1 = -1;

top2 = n;

}

void push1(int x) {

if (top1 < top2 - 1) {

arr[++top1] = x;

} else {

cout << "Stack Overflow\n";

} }

void push2(int x) {

if (top1 < top2 - 1) {

arr[--top2] = x;

} else {

cout << "Stack Overflow\n";

} }

int pop1() {

if (top1 >= 0) return arr[top1--];

cout << "Stack Underflow\n";

return -1; }

int pop2() {

if (top2 < size) return arr[top2++];

cout << "Stack Underflow\n";

return -1;

}

~TwoStacks() {

delete[] arr;

} };

int main() {

TwoStacks ts(10);

ts.push1(5);

ts.push2(15);

cout << ts.pop1() << " " << ts.pop2() << "\n";

return 0;

}

**Output:**

****

**Q20. Implement Min Heap using an Array**

**Code:** #include <iostream>

#include <vector>

using namespace std;

class MinHeap {

vector<int> heap;

void heapifyUp(int index) {

while (index > 0) {

int parent = (index - 1) / 2;

if (heap[parent] > heap[index]) {

swap(heap[parent], heap[index]);

index = parent;

} else {

break;

} } }

void heapifyDown(int index) {

int size = heap.size();

while (index < size) {

int left = 2 \* index + 1;

int right = 2 \* index + 2;

int smallest = index;

if (left < size && heap[left] < heap[smallest])

smallest = left;

if (right < size && heap[right] < heap[smallest])

smallest = right;

if (smallest != index) {

swap(heap[index], heap[smallest]);

index = smallest;

} else {

break;

} } }

public:

void insert(int val) {

heap.push\_back(val);

heapifyUp(heap.size() - 1);

}

int extractMin() {

if (heap.empty()) return -1;

int minVal = heap[0];

heap[0] = heap.back();

heap.pop\_back();

heapifyDown(0);

return minVal;

}

int getMin() {

return heap.empty() ? -1 : heap[0];

} };

int main() {

MinHeap mh;

mh.insert(10);

mh.insert(20);

mh.insert(5);

mh.insert(30);

mh.insert(3);

cout << "Min: " << mh.getMin() << endl;

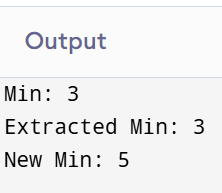
cout << "Extracted Min: " << mh.extractMin() << endl;

cout << "New Min: " << mh.getMin() << endl;

return 0;

}

**Output:**

****

**Linked List-Based Implementations:-**

**Q25. Implement Stack using Linked List**

**Code:**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node(int val) : data(val), next(nullptr) {}

};

class Stack {

private:

Node\* top;

public:

Stack() { top = nullptr; }

void push(int val) {

Node\* newNode = new Node(val);

newNode->next = top;

top = newNode;

}

int pop() {

if (!top) {

cout << "Stack Underflow\n";

return -1;

}

int val = top->data;

Node\* temp = top;

top = top->next;

delete temp;

return val;

}

int peek() {

if (!top) {

cout << "Stack is empty\n";

return -1; }

return top->data; }

bool isEmpty() {

return top == nullptr;

}

void display() {

Node\* temp = top;

while (temp) {

cout << temp->data << " ";

temp = temp->next; }

cout << "\n";

} };

int main() {

Stack s;

s.push(10);

s.push(20);

s.push(30);

s.display();

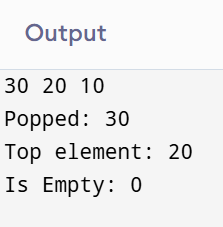
cout << "Popped: " << s.pop() << "\n";

cout << "Top element: " << s.peek() << "\n";

cout << "Is Empty: " << s.isEmpty() << "\n";

return 0; }

**Output:**

****

**Q28. Implement Circular Queue using Linked List**

**Code:** #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node(int val) : data(val), next(nullptr) {} };

class CircularQueue {

private:

Node\* front;

Node\* rear;

public:

CircularQueue() {

front = rear = nullptr;

}

void enqueue(int val) {

Node\* newNode = new Node(val);

if (!front) {

front = rear = newNode;

rear->next = front;

} else {

rear->next = newNode;

rear = newNode;

rear->next = front;

} }

int dequeue() {

if (!front) {

cout << "Queue Underflow\n";

return -1; }

int val = front->data;

if (front == rear) {

delete front;

front = rear = nullptr;

} else {

Node\* temp = front;

front = front->next;

rear->next = front;

delete temp;

}

return val;

}

void display() {

if (!front) {

cout << "Queue is empty\n";

return; }

Node\* temp = front;

do {

cout << temp->data << " ";

temp = temp->next;

} while (temp != front);

cout << "\n"; }

bool isEmpty() {

return front == nullptr;

} };

int main() {

CircularQueue q;

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.display();

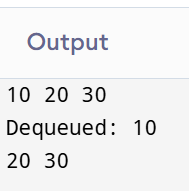
cout << "Dequeued: " << q.dequeue() << "\n";

q.display();

return 0;

}

**Output:**

****

**Heap-Based Implementations:-**

**Q33. Implement Priority Queue using Heap**

**Code:** #include <iostream>

#include <vector>

using namespace std;

class PriorityQueue {

private:

vector<int> heap;

int parent(int i) { return (i - 1) / 2; }

int leftChild(int i) { return 2 \* i + 1; }

int rightChild(int i) { return 2 \* i + 2; }

void heapifyUp(int index) {

while (index > 0 && heap[parent(index)] > heap[index]) {

swap(heap[parent(index)], heap[index]);

index = parent(index);

} }

void heapifyDown(int index) {

int smallest = index;

int left = leftChild(index);

int right = rightChild(index);

if (left < heap.size() && heap[left] < heap[smallest])

smallest = left;

if (right < heap.size() && heap[right] < heap[smallest])

smallest = right;

if (smallest != index) {

swap(heap[index], heap[smallest]);

heapifyDown(smallest);

} }

public:

void push(int val) {

heap.push\_back(val);

heapifyUp(heap.size() - 1);

}

int pop() {

if (heap.empty()) {

cout << "Priority Queue is empty\n";

return -1;

}

int top = heap[0];

heap[0] = heap.back();

heap.pop\_back();

heapifyDown(0);

return top;

}

int top() {

if (heap.empty()) return -1;

return heap[0];

}

bool empty() {

return heap.empty();

}

void display() {

for (int val : heap)

cout << val << " ";

cout << "\n";

} };

int main() {

PriorityQueue pq;

pq.push(10);

pq.push(5);

pq.push(15);

pq.push(3);

pq.display();

cout << "Top element: " << pq.top() << "\n";

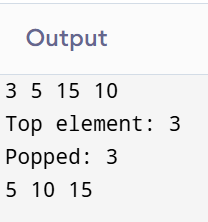
cout << "Popped: " << pq.pop() << "\n";

pq.display();

return 0;

}

**Output:**

****

**Q35. Implement Max Heap using Min Heap**

**Code:** #include <iostream>

#include <queue>

using namespace std;

class MaxHeap {

private:

priority\_queue<int, vector<int>, greater<int>> minHeap;

public:

void push(int val) {

minHeap.push(-val); }

int pop() {

if (minHeap.empty()) {

cout << "Max Heap is empty\n";

return -1; }

int maxVal = -minHeap.top();

minHeap.pop();

return maxVal; }

int top() {

if (minHeap.empty()) return -1;

return -minHeap.top(); }

bool empty() {

return minHeap.empty(); }

void display() {

priority\_queue<int, vector<int>, greater<int>> temp = minHeap;

while (!temp.empty()) {

cout << -temp.top() << " ";

temp.pop(); }

cout << "\n";

} };

int main() {

MaxHeap maxHeap;

maxHeap.push(10);

maxHeap.push(20);

maxHeap.push(5);

maxHeap.push(30);

maxHeap.display();

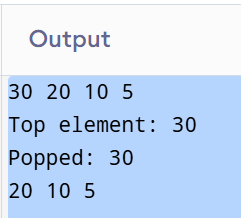
cout << "Top element: " << maxHeap.top() << "\n";

cout << "Popped: " << maxHeap.pop() << "\n";

maxHeap.display();

return 0; }

**Output:**

****

**Tree-Based Implementations:-**

**Q38. Implement BST using Linked List**

**Code:**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = right = nullptr;

} };

class BST {

private:

Node\* root;

Node\* insert(Node\* root, int val) {

if (!root) return new Node(val);

if (val < root->data) root->left = insert(root->left, val);

else root->right = insert(root->right, val);

return root; }

void inorder(Node\* root) {

if (!root) return;

inorder(root->left);

cout << root->data << " ";

inorder(root->right); }

bool search(Node\* root, int val) {

if (!root) return false;

if (root->data == val) return true;

return (val < root->data) ? search(root->left, val) : search(root->right, val); }

Node\* deleteNode(Node\* root, int val) {

if (!root) return root;

if (val < root->data) root->left = deleteNode(root->left, val);

else if (val > root->data) root->right = deleteNode(root->right, val);

else {

if (!root->left) {

Node\* temp = root->right;

delete root;

return temp; }

else if (!root->right) {

Node\* temp = root->left;

delete root;

return temp; }

Node\* temp = root->right;

while (temp->left) temp = temp->left;

root->data = temp->data;

root->right = deleteNode(root->right, temp->data); }

return root; }

public:

BST() { root = nullptr; }

void insert(int val) { root = insert(root, val); }

void inorder() {

inorder(root);

cout << endl; }

bool search(int val) { return search(root, val); }

void deleteNode(int val) { root = deleteNode(root, val); } };

int main() {

BST tree;

tree.insert(50);

tree.insert(30);

tree.insert(70);

tree.insert(20);

tree.insert(40);

tree.insert(60);

tree.insert(80);

cout << "Inorder Traversal: ";

tree.inorder();

cout << "Search 40: " << (tree.search(40) ? "Found\n" : "Not Found\n");

cout << "Search 90: " << (tree.search(90) ? "Found\n" : "Not Found\n");

tree.deleteNode(50);

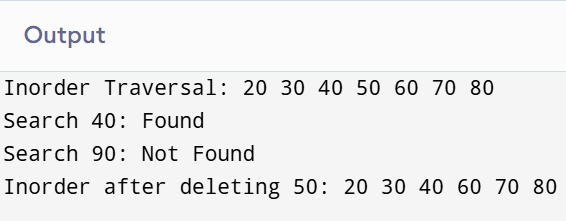
cout << "Inorder after deleting 50: ";

tree.inorder();

return 0;

}

**Output:**

****

**Q41. Implement Heap using BST**

**Code:**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = right = nullptr;

} };

class BST {

private:

Node\* root;

Node\* insert(Node\* root, int val) {

if (!root) return new Node(val);

if (val < root->data) root->left = insert(root->left, val);

else root->right = insert(root->right, val);

return root;

}

void inorder(Node\* root) {

if (!root) return;

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

bool search(Node\* root, int val) {

if (!root) return false;

if (root->data == val) return true;

return (val < root->data) ? search(root->left, val) : search(root->right, val);

}

Node\* deleteNode(Node\* root, int val) {

if (!root) return root;

if (val < root->data) root->left = deleteNode(root->left, val);

else if (val > root->data) root->right = deleteNode(root->right, val);

else {

if (!root->left) {

Node\* temp = root->right;

delete root;

return temp;

}

else if (!root->right) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = root->right;

while (temp->left) temp = temp->left;

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root; }

public:

BST() { root = nullptr; }

void insert(int val) { root = insert(root, val); }

void inorder() {

inorder(root);

cout << endl;

}

bool search(int val) { return search(root, val); }

void deleteNode(int val) { root = deleteNode(root, val); } };

int main() {

BST tree;

tree.insert(50);

tree.insert(30);

tree.insert(70);

tree.insert(20);

tree.insert(40);

tree.insert(60);

tree.insert(80);

cout << "Inorder Traversal: ";

tree.inorder();

cout << "Search 40: " << (tree.search(40) ? "Found\n" : "Not Found\n");

cout << "Search 90: " << (tree.search(90) ? "Found\n" : "Not Found\n");

tree.deleteNode(50);

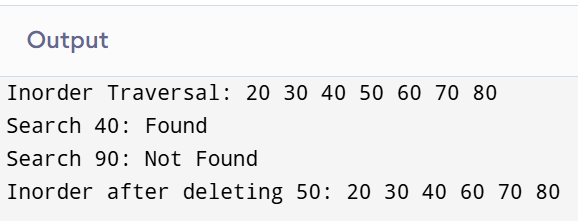
cout << "Inorder after deleting 50: ";

tree.inorder();

return 0;

}

**Output:**

****

**Hash Table-Based Implementations:-**

**Q47. Implement Trie using Hash Table**

**Code**: #include <iostream>

#include <unordered\_map>

using namespace std;

// Trie Node structure

class TrieNode {

public:

unordered\_map<char, TrieNode\*> children;

bool isEndOfWord;

TrieNode() {

isEndOfWord = false;

} };

class Trie {

private:

TrieNode\* root;

public:

Trie() {

root = new TrieNode();

}

void insert(string word) {

TrieNode\* node = root;

for (char c : word) {

if (!node->children.count(c))

node->children[c] = new TrieNode();

node = node->children[c];

}

node->isEndOfWord = true;’ 1; }

bool search(string word) {

TrieNode\* node = root;

for (char c : word) {

if (!node->children.count(c))

return false;

node = node->children[c];

}

return node->isEndOfWord;

}

bool startsWith(string prefix) {

TrieNode\* node = root;

for (char c : prefix) {

if (!node->children.count(c))

return false;

node = node->children[c];

}

return true;

} };

int main() {

Trie trie;

trie.insert("apple");

trie.insert("app");

trie.insert("bat");

cout << "Search 'apple': " << (trie.search("apple") ? "Found\n" : "Not Found\n");

cout << "Search 'bat': " << (trie.search("bat") ? "Found\n" : "Not Found\n");

cout << "Search 'bad': " << (trie.search("bad") ? "Found\n" : "Not Found\n");

cout << "Starts with 'app': " << (trie.startsWith("app") ? "Yes\n" : "No\n");

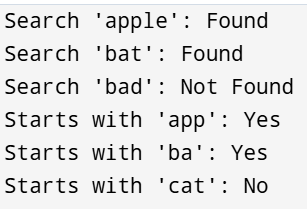
cout << "Starts with 'ba': " << (trie.startsWith("ba") ? "Yes\n" : "No\n");

cout << "Starts with 'cat': " << (trie.startsWith("cat") ? "Yes\n" : "No\n"); // No

return 0;

}

**Output:**

****

**Q48. Implement Graph using Hash Table (Adjacency List Representation)**

**Code:**

#include <iostream>

#include <unordered\_map>

#include <list>

using namespace std;

class Graph {

private:

unordered\_map<int, list<int>> adjList;

public:

void addEdge(int u, int v, bool isDirected = false) {

adjList[u].push\_back(v);

if (!isDirected) {

adjList[v].push\_back(u);

} }

void printGraph() {

for (auto &pair : adjList) {

cout << pair.first << " -> ";

for (int neighbor : pair.second) {

cout << neighbor << " ";

}

cout << endl;

} } };

int main() {

Graph g;

g.addEdge(1, 2);

g.addEdge(1, 3);

g.addEdge(2, 4);

g.addEdge(3, 5);

g.addEdge(4, 5);

g.addEdge(5, 6, true);

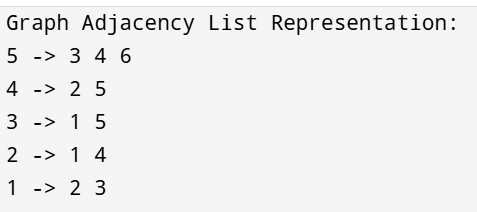
cout << "Graph Adjacency List Representation:\n";

g.printGraph();

return 0;

}

**Output:**

****

**Graph-Based Implementations:-**

**Q50. Implement Graph using Adjacency Matrix (2D Array)**

**Code:**

#include <iostream>

using namespace std;

class Graph {

private:

int \*\*adjMatrix;

int vertices;

public:

Graph(int v) {

vertices = v;

adjMatrix = new int\*[v];

for (int i = 0; i < v; i++) {

adjMatrix[i] = new int[v];

for (int j = 0; j < v; j++) {

adjMatrix[i][j] = 0; } } }

void addEdge(int u, int v, bool isDirected = false) {

adjMatrix[u][v] = 1;

if (!isDirected) {

adjMatrix[v][u] = 1;

} }

void printGraph() {

cout << "Adjacency Matrix Representation:\n";

for (int i = 0; i < vertices; i++) {

for (int j = 0; j < vertices; j++) {

cout << adjMatrix[i][j] << " ";

}

cout << endl;

} }

~Graph() {

for (int i = 0; i < vertices; i++) {

delete[] adjMatrix[i];

}

delete[] adjMatrix;

} };

int main() {

int vertices = 5;

Graph g(vertices);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(2, 3);

g.addEdge(3, 4);

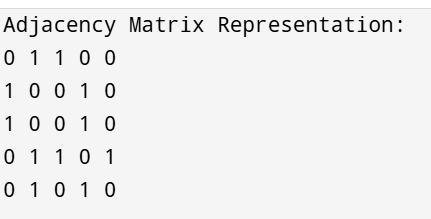
g.addEdge(4, 1, true);

g.printGraph();

return 0;

}

**Output:**



**Q53. Implement Topological Sorting using Graph + Stack (DFS)**

**Code:**

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

class Graph {

int V;

vector<vector<int>> adj;

public:

Graph(int vertices) {

V = vertices;

adj.resize(V);

}

void addEdge(int u, int v) {

adj[u].push\_back(v);

}

void dfs(int node, vector<bool> &visited, stack<int> &st) {

visited[node] = true;

for (int neighbor : adj[node]) {

if (!visited[neighbor]) {

dfs(neighbor, visited, st);

} }

st.push(node);

}

void topologicalSort() {

vector<bool> visited(V, false);

stack<int> st;

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

if (!visited[i]) {

dfs(i, visited, st);

} }

cout << "Topological Order: ";

while (!st.empty()) {

cout << st.top() << " ";

st.pop();

}

cout << endl;

} };

int main() {

Graph g(6);

g.addEdge(5, 2);

g.addEdge(5, 0);

g.addEdge(4, 0);

g.addEdge(4, 1);

g.addEdge(2, 3);

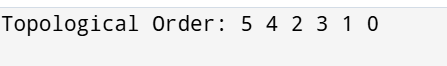
g.addEdge(3, 1);

g.topologicalSort();

return 0;

}

**Output:**

****