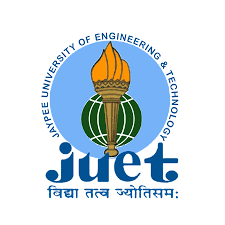
**JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY, GUNA**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



**A PRACTICAL WORK BOOK**

**of**

**Advanced Programming lab**

**(COURSE CODE: 18b17CI673)**

**SUBMITTED**

**TO**

**Dr. Ravindra kumar singh**

**Name of student\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ENRL NO.\_\_\_\_\_\_\_\_\_\_**

**Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ BATCH\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**BRANCH\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SESSION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Lab Exercise 1**

**Recursion**

**Q:1** Tower of Hanoi

#include <bits/stdc++.h>

using namespace std;

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

if (n == 0) {

return;

}

towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod); cout << "Move disk " << n << " from rod " << from\_rod << " to rod " << to\_rod << endl;

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

int main()

{

int N = 3;

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

return 0;

}

**Output :**

Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 3 from rod A to rod C

Move disk 1 from rod B to rod A

Move disk 2 from rod B to rod C

Move disk 1 from rod A to rod C

**Q:2** All permutations of a string taken from user

#include <bits/stdc++.h>

using namespace std;

void permute(string& a, int l, int r)

{

if (l == r)

cout << a << endl;

else {

for (int i = l; i <= r; i++) {

swap(a[l], a[i]);

permute(a, l + 1, r);

swap(a[l], a[i]); }}}

int main()

{

string str = "ABC";

int n = str.size();

permute(str, 0, n - 1);

return 0;

}

**Output :**

ABC

ACB

BAC

BCA

CBA

CAB

**Q:3** Printing all the binary string of length n

#include <bits/stdc++.h>

using namespace std;

void printTheArray(int arr[], int n)

{

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

cout << arr[i] << " ";

}

cout << endl;

}

void generateAllBinaryStrings(int n, int arr[], int i) {

if (i == n) {

printTheArray(arr, n);

return;

}

arr[i] = 0;

generateAllBinaryStrings(n, arr, i + 1);

arr[i] = 1;

generateAllBinaryStrings(n, arr, i + 1);

}

int main()

{

int n = 2;

int arr[n];

generateAllBinaryStrings(n, arr, 0);

return 0;

}

Output :

00

01

10

11

**Q:4** Calculate GCD using Euclid's algorithm

#include <stdio.h>

int gcd(int a, int b)

{

if (a == 0)

return b;

return gcd(b % a, a);

}

int main()

{

int a = 10, b = 15;

printf("GCD(%d, %d) = %d\n", a, b, gcd(a, b)); a = 35, b = 10;

printf("GCD(%d, %d) = %d\n", a, b, gcd(a, b));

a = 31, b = 2;

printf("GCD(%d, %d) = %d\n", a, b, gcd(a, b));

return 0;

}

**Output :**

GCD(5,15) = 5

GCD(21,35) = 7

GCD(21,2) = 1

**Q:5** Implement merge sort algorithm using recursion

#include <iostream>

using namespace std;

void merge(int array[], int const left, int const mid, int const right)

{

auto const subArrayOne = mid - left + 1;

auto const subArrayTwo = right - mid;

auto \*leftArray = new int[subArrayOne],

\*rightArray = new int[subArrayTwo];

for (auto i = 0; i < subArrayOne; i++)

leftArray[i] = array[left + i];

for (auto j = 0; j < subArrayTwo; j++)

rightArray[j] = array[mid + 1 + j];

auto indexOfSubArrayOne

= 0, indexOfSubArrayTwo

= 0; int indexOfMergedArray

= left; // Initial index of merged array

while (indexOfSubArrayOne < subArrayOne

&& indexOfSubArrayTwo < subArrayTwo) {

if (leftArray[indexOfSubArrayOne]

<= rightArray[indexOfSubArrayTwo]) {

array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

}

else {

array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

}

indexOfMergedArray++;

}

while (indexOfSubArrayOne < subArrayOne) { array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

indexOfMergedArray++;

}

while (indexOfSubArrayTwo < subArrayTwo) { array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

indexOfMergedArray++;

}

delete[] leftArray;

delete[] rightArray;

}

void mergeSort(int array[], int const begin, int const end) {

if (begin >= end)

return; // Returns recursively

auto mid = begin + (end - begin) / 2;

mergeSort(array, begin, mid);

mergeSort(array, mid + 1, end);

merge(array, begin, mid, end);

}

void printArray(int A[], int size)

{

for (auto i = 0; i < size; i++)

cout << A[i] << " ";

}

int main()

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

auto arr\_size = sizeof(arr) / sizeof(arr[0]);

cout << "Given array is \n";

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

cout << "\nSorted array is \n";

printArray(arr, arr\_size);

return 0;

}

**Output :**

Given Array is

2 34 5 23 12 46

Sorted Array is

2 5 12 23 34 46

**Q:6** Solve n queen problem in chess using recursion.

#include <bits/stdc++.h>

#define N 4

using namespace std;

void printSolution(int board[N][N])

{

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

for (int j = 0; j < N; j++)

if(board[i][j])

cout << "Q ";

else cout<<". ";

printf("\n");

}

}

bool isSafe(int board[N][N], int row, int col) {

int 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;

}

bool solveNQUtil(int board[N][N], int col)

{

if (col >= N)

return true;

for (int 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; // BACKTRACK

}

}

return false;

}

bool solveNQ()

{

int board[N][N] = { { 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 } };

if (solveNQUtil(board, 0) == false) {

cout << "Solution does not exist";

return false;

}

printSolution(board);

return true;

}

int main()

{

solveNQ();

return 0;

}

**Output :**

. . Q .

Q . . .

. . . Q

. Q . .

**Q.7** Remove a given element form a linked list using recursion.

#include <bits/stdc++.h>

using namespace std;

struct Node {

int data;

struct Node\* next;

};

Node\* deleteNode(Node\* start, int k)

{

if (k < 1)

return start;

if (start == NULL)

return NULL;

if (k == 1)

{

Node \*res = start->next;

delete(start);

return res;

}

start->next = deleteNode(start->next, k-1); return start;

}

void push(struct Node \*\*head\_ref, int new\_data) {

struct Node \*new\_node = new Node;

new\_node->data = new\_data;

new\_node->next = \*head\_ref;

\*head\_ref = new\_node;

}

void printList(struct Node \*head)

{

while (head!=NULL)

{

cout << head->data << " ";

head = head->next;

}

printf("\n");

}

int main()

{

struct Node \*head = NULL;

push(&head,3);

push(&head,2);

push(&head,6);

push(&head,5);

push(&head,11);

push(&head,10);

push(&head,15);

push(&head,12);

int k = 3;

head = deleteNode(head, k);

printf("\nModified Linked List: ");

printList(head);

return 0;

}

**Output :**

Modified linked list : 12 15 11 5 6 2 3

**Q:8** Solve egg drop problem using recursion.

#include <bits/stdc++.h>

using namespace std ;

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

int eggDrop(int n, int k)

{ if (k == 1 || k == 0)

return k;

if (n == 1)

return k;

int min = INT\_MAX, x, res;

for (x = 1; x <= k; x++) {

res = max(eggDrop(n - 1, x - 1), eggDrop(n, k - x)); if (res < min)

min = res;

}

return min + 1;

}

int main()

{

int n = 2, k = 10;

cout << "Minimum number of trials "

"in worst case with "

<< n << " eggs and " << k << " floors is "

<< eggDrop(n, k) << endl;

return 0;

}

**Output :**

Minimum number of trails in worst case with 2 eggs and 10 floors is 4

**Lab Exercise 2**

Binary Search Trees and AVL Trees

**Q:1** Write a program to construct a Binary Search Tree and check by inserting the elements. 25,15, 5, 20, 40, 3, 6, 16, 21.

**Q:2** Write a program for deleting the elements from the above Binary Search Tree. Check it by deleting the elements 25, 40.

#include<iostream>

using namespace std;

class BST {

struct node {

int data;

node\* left;

node\* right;

};

node\* root;

node\* makeEmpty(node\* t) {

if(t == NULL)

return NULL;{

makeEmpty(t->left);

makeEmpty(t->right);

delete t; }

return NULL; }

node\* insert(int x, node\* t){

if(t == NULL){

t = new node;

t->data = x;

t->left = t->right = NULL;}

else if(x < t->data)

t->left = insert(x, t->left);

else if(x > t->data)

t->right = insert(x, t->right);

return t;}

node\* findMin(node\* t) {

if(t == NULL)

return NULL;

else if(t->left == NULL)

return t;

else

return findMin(t->left) }

node\* findMax(node\* t) {

if(t == NULL)

return NULL;

else if(t->right == NULL)

return t;

else

return findMax(t->right); }

node\* remove(int x, node\* t) {

node\* temp;

if(t == NULL)

return NULL;

else if(x < t->data)

t->left = remove(x, t->left);

else if(x > t->data)

t->right = remove(x, t->right);

else if(t->left && t->right){

temp = findMin(t->right);

t->data = temp->data;

t->right = remove(t->data, t->right); } else {

temp = t;

if(t->left == NULL)

t = t->right;

else if(t->right == NULL)

t = t->left;

delete temp;}

return t;}

void inorder(node\* t) {

if(t == NULL)

return;

inorder(t->left);

cout << t->data << " ";

inorder(t->right);}

node\* find(node\* t, int x) {

if(t == NULL)

return NULL;

else if(x < t->data)

return find(t->left, x);

else if(x > t->data)

return find(t->right, x);

else

return t; }

public:

BST() {

root = NULL;}

~BST() {

root = makeEmpty(root);}

void insert(int x) {

root = insert(x, root); }

void remove(int x) {

root = remove(x, root); }

void display() {

inorder(root);

cout << endl;}

void search(int x) {

root = find(root, x); }};

int main() {

BST t;

t.insert(25);

t.insert(15);

t.insert(5);

t.insert(20);

t.insert(40);

t.insert(3);

t.insert(6);

t.display();

t.remove(25);

t.display();

t.remove(40);

t.display();

return 0;

}

**Q:3** Write a program to construct the AVL tree using proper rotations by inserting the elements. Check by inserting the elements 5, 10, 15, 20, 15, 40, 45, 80, 90.

#include<bits/stdc++.h>

using namespace std;

class Node

{

public:

int key;

Node \*left;

Node \*right;

int height;

};

int height(Node \*N)

{

if (N == NULL)

return 0;

return N->height;

}

int max(int a, int b)

{

return (a > b)? a : b;

}

Node\* newNode(int key)

{

Node\* node = new Node();

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1; // new node is initially

// added at leaf

return(node);

}

Node \*rightRotate(Node \*y)

{

Node \*x = y->left;

Node \*T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left),

height(y->right)) + 1;

x->height = max(height(x->left),

height(x->right)) + 1;

return x;

}

Node \*leftRotate(Node \*x)

{

Node \*y = x->right;

Node \*T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left),

height(x->right)) + 1;

y->height = max(height(y->left),

height(y->right)) + 1;

return y;

}

int getBalance(Node \*N)

{

if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

Node\* insert(Node\* node, int key)

{

/\* 1. Perform the normal BST insertion \*/

if (node == NULL)

return(newNode(key));

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

else // Equal keys are not allowed in BST

return node;

node->height = 1 + max(height(node->left),

height(node->right));

node to check whether this node became

unbalanced \*/

int balance = getBalance(node);

if (balance > 1 && key < node->left->key) return rightRotate(node);

if (balance < -1 && key > node->right->key) return leftRotate(node);

if (balance > 1 && key > node->left->key) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && key < node->right->key) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

void preOrder(Node \*root)

{

if(root != NULL)

{

cout << root->key << " ";

preOrder(root->left);

preOrder(root->right);

}

}

int main()

{

Node \*root = NULL;

root = insert(root, 5);

root = insert(root, 10);

root = insert(root, 15);

root = insert(root, 20);

root = insert(root, 15);

root = insert(root, 40);

root = insert(root, 45);

root = insert(root, 80);

root = insert(root, 90);

return 0;

}

**Lab Exercise 3**

Red Black Trees

**Q:1** Write a program to construct a Red Black Tree using sequence of insertions. Elements should be taken from the user. You should also print the resultant tree after every insertion.

**Q:2** Write a program to delete the element from a given red black tree. You also also print the leftover Red Black Tree

**Sol. 1,2**

class Node {

int data; // holds the key

Node parent; // pointer to the parent

Node left; // pointer to left child

Node right; // pointer to right child

int color; // 1 . Red, 0 . Black

}

public class RedBlackTree {

private Node root;

private Node TNULL;

private void preOrderHelper(Node node) {

if (node != TNULL) {

System.out.print(node.data + " ");

preOrderHelper(node.left);

preOrderHelper(node.right);

}

}

private void inOrderHelper(Node node) {

if (node != TNULL) {

inOrderHelper(node.left);

System.out.print(node.data + " ");

inOrderHelper(node.right);

}

}

private void postOrderHelper(Node node) {

if (node != TNULL) {

postOrderHelper(node.left);

postOrderHelper(node.right);

System.out.print(node.data + " ");

}

}

private Node searchTreeHelper(Node node, int key) { if (node == TNULL || key == node.data) {

return node;

}

if (key < node.data) {

return searchTreeHelper(node.left, key);

}

return searchTreeHelper(node.right, key);

}

private void fixDelete(Node x) {

Node s;

while (x != root && x.color == 0) {

if (x == x.parent.left) {

s = x.parent.right;

if (s.color == 1) {

s.color = 0;

x.parent.color = 1;

leftRotate(x.parent);

s = x.parent.right;

}

if (s.left.color == 0 && s.right.color == 0) {

s.color = 1;

x = x.parent;

} else {

if (s.right.color == 0) {

s.left.color = 0;

s.color = 1;

rightRotate(s);

s = x.parent.right;

}

s.color = x.parent.color;

x.parent.color = 0;

s.right.color = 0;

leftRotate(x.parent);

x = root;

}

} else {

s = x.parent.left;

if (s.color == 1) {

s.color = 0;

x.parent.color = 1;

rightRotate(x.parent);

s = x.parent.left;

}

if (s.right.color == 0 && s.right.color == 0) {

s.color = 1;

x = x.parent;

} else {

if (s.left.color == 0) {

s.right.color = 0;

s.color = 1;

leftRotate(s);

s = x.parent.left;

}

s.color = x.parent.color;

x.parent.color = 0;

s.left.color = 0;

rightRotate(x.parent);

x = root;

}

}

}

x.color = 0;

}

private void rbTransplant(Node u, Node v){

if (u.parent == null) {

root = v;

} else if (u == u.parent.left){

u.parent.left = v;

} else {

u.parent.right = v;

}

v.parent = u.parent;

}

private void deleteNodeHelper(Node node, int key) {

Node z = TNULL;

Node x, y;

while (node != TNULL){

if (node.data == key) {

z = node;

}

if (node.data <= key) {

node = node.right;

} else {

node = node.left;

}

}

if (z == TNULL) {

System.out.println("Couldn't find key in the tree"); return;

}

y = z;

int yOriginalColor = y.color;

if (z.left == TNULL) {

x = z.right;

rbTransplant(z, z.right);

} else if (z.right == TNULL) {

x = z.left;

rbTransplant(z, z.left);

} else {

y = minimum(z.right);

yOriginalColor = y.color;

x = y.right;

if (y.parent == z) {

x.parent = y;

} else {

rbTransplant(y, y.right);

y.right = z.right;

y.right.parent = y;

}

rbTransplant(z, y);

y.left = z.left;

y.left.parent = y;

y.color = z.color;

}

if (yOriginalColor == 0){

fixDelete(x);

}

}

private void fixInsert(Node k){

Node u;

while (k.parent.color == 1) {

if (k.parent == k.parent.parent.right) {

u = k.parent.parent.left; // uncle

if (u.color == 1) {

u.color = 0;

k.parent.color = 0;

k.parent.parent.color = 1;

k = k.parent.parent;

} else {

if (k == k.parent.left) {

k = k.parent;

rightRotate(k);

}

k.parent.color = 0;

k.parent.parent.color = 1;

leftRotate(k.parent.parent);

}

} else {

u = k.parent.parent.right; // uncle

if (u.color == 1) {

u.color = 0;

k.parent.color = 0;

k.parent.parent.color = 1;

k = k.parent.parent;

} else {

if (k == k.parent.right) {

k = k.parent;

leftRotate(k);

}

// mirror case 3.2.1

k.parent.color = 0;

k.parent.parent.color = 1;

rightRotate(k.parent.parent);

}

}

if (k == root) {

break;

}

}

root.color = 0;

}

private void printHelper(Node root, String indent, boolean last) {

if (root != TNULL) {

System.out.print(indent);

if (last) {

System.out.print("R----");

indent += " ";

} else {

System.out.print("L----");

indent += "| ";

}

String sColor = root.color == 1?"RED":"BLACK"; System.out.println(root.data + "(" + sColor + ")"); printHelper(root.left, indent, false);

printHelper(root.right, indent, true);

}

}

public RedBlackTree() {

TNULL = new Node();

TNULL.color = 0;

TNULL.left = null;

TNULL.right = null;

root = TNULL;

}

public void preorder() {

preOrderHelper(this.root);

}

public void inorder() {

inOrderHelper(this.root);

}

public void postorder() {

postOrderHelper(this.root);

}

public Node searchTree(int k) {

return searchTreeHelper(this.root, k);

}

public Node minimum(Node node) {

while (node.left != TNULL) {

node = node.left;

}

return node;

}

public Node maximum(Node node) {

while (node.right != TNULL) {

node = node.right;

}

return node;

}

public Node successor(Node x) {

if (x.right != TNULL) {

return minimum(x.right);

}

Node y = x.parent;

while (y != TNULL && x == y.right) {

x = y;

y = y.parent;

}

return y;

}

public Node predecessor(Node x) { if (x.left != TNULL) {

return maximum(x.left);

}

Node y = x.parent;

while (y != TNULL && x == y.left) { x = y;

y = y.parent;

}

return y;

}

public void leftRotate(Node x) {

Node y = x.right;

x.right = y.left;

if (y.left != TNULL) {

y.left.parent = x;

}

y.parent = x.parent;

if (x.parent == null) {

this.root = y;

} else if (x == x.parent.left) {

x.parent.left = y;

} else {

x.parent.right = y;

}

y.left = x;

x.parent = y;

}

public void rightRotate(Node x) { Node y = x.left;

x.left = y.right;

if (y.right != TNULL) {

y.right.parent = x;

}

y.parent = x.parent;

if (x.parent == null) {

this.root = y;

} else if (x == x.parent.right) {

x.parent.right = y;

} else {

x.parent.left = y;

}

y.right = x;

x.parent = y;

}

public void insert(int key) {

Node node = new Node();

node.parent = null;

node.data = key;

node.left = TNULL;

node.right = TNULL;

node.color = 1; // new node must be red

Node y = null;

Node x = this.root;

while (x != TNULL) {

y = x;

if (node.data < x.data) {

x = x.left;

} else {

x = x.right;

}

}

node.parent = y;

if (y == null) {

root = node;

} else if (node.data < y.data) {

y.left = node;

} else {

y.right = node;

}

if (node.parent == null){

node.color = 0;

return;

}

if (node.parent.parent == null) {

return;

}

fixInsert(node);

}

public Node getRoot(){

return this.root;

}

public void deleteNode(int data) {

deleteNodeHelper(this.root, data);

}

public void prettyPrint() {

printHelper(this.root, "", true);

}

public static void main(String [] args){ RedBlackTree bst = new RedBlackTree(); bst.insert(8);

bst.insert(18);

bst.insert(5);

bst.insert(15);

bst.insert(17);

bst.insert(25);

bst.insert(40);

bst.insert(80);

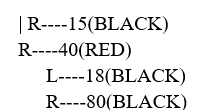
bst.deleteNode(25);

bst.prettyPrint();

}

}

**Output**

R----17(BLACK)

L----8(RED)

| L----5(BLACK)

| R----15(BLACK)

R----25(RED)

L----18(BLACK)

R----40(BLACK)

R----80(RED)

R----17(BLACK)

L----8(RED)

| L----5(BLACK)