

**2)A program P reads in 500 integers in the range [0..100] representing the scores of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies.**

1. Step 1: Create an array frequency of size 101 (to handle scores from 0 to 100). Each index represents a score, and the value at that index is the frequency of the score.

2. Step 2: Loop 500 times to read the scores. For each score, check if it's above 50. If it is, increment the corresponding element in the frequency array.

3. Step 3: After processing all scores, loop through the array starting from index 51 to 100. For each score with a non-zero frequency, print the score and its frequency.

**5) Consider a standard Circular Queue \’q\’ implementation (which has the same condition for Queue Full and Queue Empty) whose size is 11 and the elements of the queue are q[0], q[1], q[2].....,q[10]. The front and rear pointers are initialized to point at q[2] . In which position will the ninth element be added?**

Step 1. Initial position of rear: Rear pointer starts at q[2].

Step 2. First element: Added at q[2], rear moves to q[3].

Step 3. Second element: Added at q[3], rear moves to q[4].

Step 4. Third element: Added at q[4], rear moves to q[5].

Step 5. Fourth element: Added at q[5], rear moves to q[6].

Step 6. Fifth element: Added at q[6], rear moves to q[7].

Step 7. Sixth element: Added at q[7], rear moves to q[8].

Step 8. Seventh element: Added at q[8], rear moves to q[9].

Step 9. Eighth element: Added at q[9], rear moves to q[10].

Step 10. Ninth element: Added at q[10], rear moves to q[0] (wrap-around).

**6) Write a C Program to implement Red Black Tree**

#include <stdio.h>

#include <stdlib.h>

#define RED 1

#define BLACK 0

Typedef struct Node {

Int data;

Int color;

Struct Node \*left, \*right, \*parent;

} Node;

Node\* createNode(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->left = newNode->right = newNode->parent = NULL;

newNode->color = RED;

return newNode;

}

Void leftRotate(Node \*\*root, Node \*x) {

Node \*y = x->right;

x->right = y->left;

if (y->left != NULL)

y->left->parent = x;

y->parent = x->parent;

if (x->parent == NULL)

\*root = y;

Else if (x == x->parent->left)

x->parent->left = y;

else

x->parent->right = y;

y->left = x;

x->parent = y;

}

Void rightRotate(Node \*\*root, Node \*y)

{

Node \*x = y->left;

y->left = x->right;

if (x->right != NULL)

x->right->parent = y;

x->parent = y->parent;

if (y->parent == NULL)

\*root = x;

Else if (y == y->parent->left)

y->parent->left = x;

else

y->parent->right = x;

x->right = y;

y->parent = x;

}

Void fixViolation(Node \*\*root, Node \*newNode)

{

Node \*parent = NULL;

Node \*grandParent = NULL;

While ((newNode != \*root) && (newNode->color == RED) && (newNode->parent->color == RED)) {

Parent = newNode->parent;

grandParent = parent->parent;

if (parent == grandParent->left) {

Node \*uncle = grandParent->right;

If (uncle != NULL && uncle->color == RED) {

grandParent->color = RED;

parent->color = BLACK;

uncle->color = BLACK;

newNode = grandParent; // Move up the tree

} else {

If (newNode == parent->right) {

leftRotate(root, parent);

newNode = parent;

parent = newNode->parent;

}

rightRotate(root, grandParent);

int temp = parent->color;

parent->color = grandParent->color;

grandParent->color = temp;

newNode = parent; // Move up the tree

}

} else {

Node \*uncle = grandParent->left;

If (uncle != NULL && uncle->color == RED)

{

grandParent->color = RED;

parent->color = BLACK;

uncle->color = BLACK;

newNode = grandParent;

} else {

If (newNode == parent->left) {

rightRotate(root, parent);

newNode = parent;

parent = newNode->parent;

}

leftRotate(root, grandParent);

int temp = parent->color;

parent->color = grandParent->color;

grandParent->color = temp;

newNode = parent; // Move up the tree

}

}

}

(\*root)->color = BLACK;

}

Void insert(Node \*\*root, int data) {

Node \*newNode = createNode(data);

Node \*y = NULL;

Node \*x = \*root;

While (x != NULL) {

Y = x;

If (newNode->data < x->data)

X = x->left;

Else

X = x->right;

}

newNode->parent = y;

if (y == NULL) {

\*root = newNode;

} else if (newNode->data < y->data) {

y->left = newNode;

} else {

y->right = newNode;

}

fixViolation(root, newNode);

}

Void inOrder(Node \*root) {

If (root != NULL) {

inOrder(root->left);

printf(“%d “, root->data);

inOrder(root->right);

}

}

Int main() {

Node \*root = NULL;

Insert(&root, 10);

Insert(&root, 20);

Insert(&root, 30);

Insert(&root, 15);

Insert(&root, 25);

Printf(“In-order traversal of the Red-Black Tree: “);

inOrder(root);

printf(“\n”);

return 0;

}