ASSIGNMENT –II

ADVANCED DATA STRUCTURE

Submitted To

Ms.Akshara Sasidharan

Dept.of Computer Application

Submitted By

Dona Mary Shaju

S1 MCA 24-26

Roll NO.:34

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?

Solution:

To store the frequencies of scores above 50, a simple and efficient approach is to use an array. Since the scores are in the range [0..100], you can create an array of size 51 (to cover scores from 51 to 100) where each index represents the score, and the value at that index represents the frequency of that score.

1. **Initialize an Array**: Create an array called frequency of size 51 (indices 0 to 50 will be unused, corresponding to scores 0 to 50).
2. **Read Scores**: As you read each of the 500 scores, check if the score is greater than 50. If it is, increment the corresponding index in the frequency array.
3. **Print Frequencies**: After processing all scores, print the frequencies for scores from 51 to 100.

Summary

- **Efficiency**: Using an array allows for O(1)O(1)O(1) access time for counting frequencies.
- **Simplicity**: The logic is straightforward and easy to implement.
- **Low Memory Usage**: You only need to store counts for 50 possible scores, making it memory-efficient.

Using an array of size 51 to track frequencies is efficient in both space and time, making it easy to access and manipulate the data. This method allows you to quickly count and display the frequencies of scores above 50.

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?

Solution:

- **Initial state**:

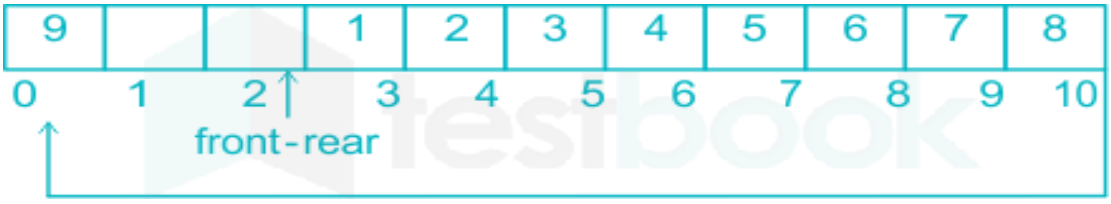
  front = q[2]

  rear = q[2]

- **Adding elements**:

In a circular queue, elements are added at the rear end and removed from the front. If the front and rear pointers are both initialized at q[2], the first element gets added at q[2]. After this, the rear **pointer** will increment by one for every added element. So, the second element gets added at q[3], third at q[4], and so on until the rear pointer reaches q[10] at the 9th **element**. After reaching the end (q[10]), the rear pointer starts from the beginning (q[0]).

So, if you add the 9th element into the queue, it will be added in the position q[10].



6) Write a C Program to implement Red Black Tree

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#define RED 0
#define BLACK 1
typedef struct Node {
    int data;
    struct Node *left, *right, *parent;
    int color;
} Node;

Node* createNode(int data);
Node* rotateLeft(Node **root, Node *x);
Node* rotateRight(Node **root, Node *x);
void fixViolation(Node **root, Node *node);
void insertNode(Node **root, Node *dataNode);
void inorderTraversal(Node *root);
void printTree(Node *root, int space);
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;
}
Node* rotateLeft(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;
    return *root;
}

Node* rotateRight(Node **root, Node *x) {
    Node *y = x->left;
    x->left = y->right;
    if (y->right != NULL)
        y->right->parent = x;
    y->parent = x->parent;
    if (x->parent == NULL)
        *root = y;
    else if (x == x->parent->right)
        x->parent->right = y;
    else
        x->parent->left = y;
    y->right = x;
    x->parent = y;
    return *root;
}

void fixViolation(Node **root, Node *node) {
    Node *parent, *grandParent;

    while ((node != *root) && (node->parent->color == RED)) {
        parent = node->parent;
        grandParent = parent->parent;
```

```
if (parent == grandParent->left) {
    Node *uncle = grandParent->right;
    if (uncle && uncle->color == RED) {
        parent->color = BLACK;
        uncle->color = BLACK;
        grandParent->color = RED;
        node = grandParent;
    } else {
        if (node == parent->right) {
            node = parent;
            *root = rotateLeft(root, node);
        }
        parent->color = BLACK;
        grandParent->color = RED;
        *root = rotateRight(root, grandParent);
    }
} else {
    Node *uncle = grandParent->left;

    if (uncle && uncle->color == RED) {
        parent->color = BLACK;
        uncle->color = BLACK;
        grandParent->color = RED;
        node = grandParent;
    } else {
        if (node == parent->left) {
            node = parent;
            *root = rotateRight(root, node);
        }
        parent->color = BLACK;
        grandParent->color = RED;
        *root = rotateLeft(root, grandParent);
    }
}
```

```
    }
    (*root)->color = BLACK;
}

void insertNode(Node **root, Node *dataNode) {
    Node *parent = NULL;
    Node *current = *root;

    while (current != NULL) {
        parent = current;
        if (dataNode->data < current->data)
            current = current->left;
        else
            current = current->right;
    }

    dataNode->parent = parent;

    if (parent == NULL) {
        *root = dataNode;
    } else if (dataNode->data < parent->data) {
        parent->left = dataNode;
    } else {
        parent->right = dataNode;
    }

    fixViolation(root, dataNode);
}

void inorderTraversal(Node *root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
```

```
}
void printTree(Node *root, int space) {
    if (root == NULL)
        return;
    space += 10;
    printTree(root->right, space);

    printf("\n");
    for (int i = 10; i < space; i++)
        printf(" ");
    printf("%d(%s)\n", root->data, root->color == RED ? "RED" : "BLACK");

    printTree(root->left, space);
}
int main() {
    Node *root = NULL;

    insertNode(&root, createNode(10));
    insertNode(&root, createNode(20));
    insertNode(&root, createNode(30));
    insertNode(&root, createNode(15));
    insertNode(&root, createNode(25));

    printf("Inorder Traversal:\n");
    inorderTraversal(root);
    printf("\n");

    printf("Red-Black Tree Visualization:\n");
    printTree(root, 0);

    return 0;
}
```

Output:

Inorder Traversal:

10 15 20 25 30

Red-Black Tree Visualization:

30(BLACK)

25(RED)

20(BLACK)

15(RED)

10(BLACK)