Advanced Data Structures

ASSIGNMENT-1

Submitted To:

Ms. Akshara Sasidaran

Dept of Computer Applications

Submitted by:

Liya Sara Joseph

44

S1-MCA

ADVANCED DATA STRUCTURES

ASSIGNEMENT -1

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?

Ans) The best way for program P to store the frequencies of scores above 50 would be to use an array. Given that the scores are in the range [0..100], create an array of size 101 (indexing from 0 to 100). Steps are as follows:

Initialize an array: Create an integer array frequency[101] initialized to zero.

Read the scores: As you read each of the 500 scores, check if the score is greater than 50.

Update the frequency: If the score is greater than 50, increment the corresponding index in the frequency array.

Advantages

- Efficient memory usage: Since the scores range from 0 to 100, an array of size 51 (100 - 50 + 1) is sufficient to store the frequencies.
- Simple implementation: The code will be easy to understand and maintain.

Example

```
int frequencies[51] = {0}; // Initialize frequencies to 0
for (int i = 0; i < 500; i++)
{
   int score;
   scanf("%d", &score);
   if (score > 50)
 {
      frequencies[score - 51]++;
 }
}
for (int i = 0; i < 51; i++)
 {
   if (frequencies[i] > 0)
```

```
    {
        printf("Score %d: %d\n", i + 51, frequencies[i]);
    }
}
```

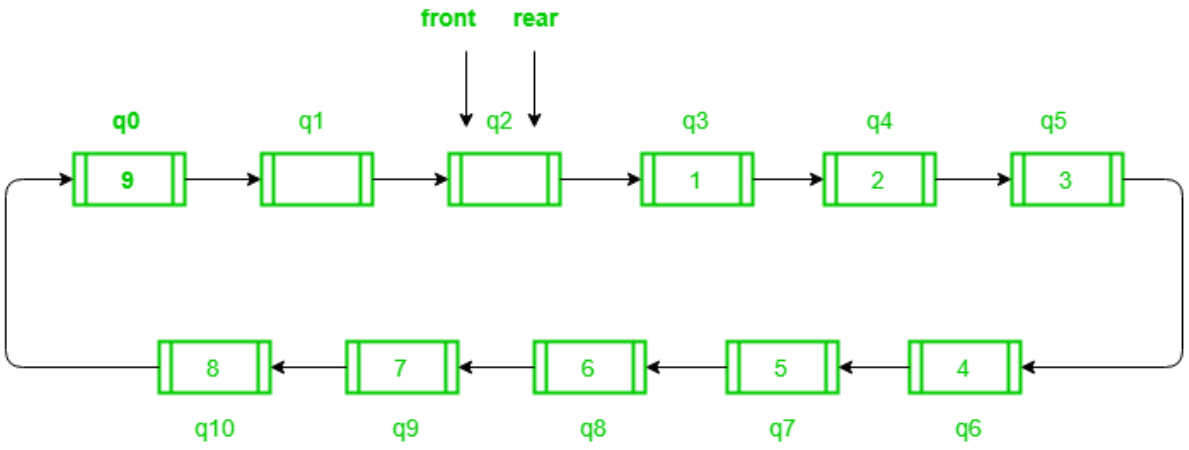
3. Consider a standard Circular Queue \'q'\ implementation (which has the same condition for Queue Full and Queue Empty) whose size is 11 and the elements of the queue are q[0], q[1], q[2].....,q[10]. The front and rear pointers are initialized to point at q[2] . In which position will the ninth element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

Ans) In a circular queue, elements are added at the rear pointer, and both the front and rear pointers move in a circular manner. Given the conditions:

- The queue size is 11.
- The front and rear pointers are both initialized to point at q[2]. The ninth element will be added at position q[0].

When we add elements to the queue, the rear pointer moves to the next position each time an element is added.

a. **Initial Position**: Both front and rear pointers start at q[2].
b. **First Element**: Added at q[2], rear moves to q[3].
c. **Second Element**: Added at q[3], rear moves to q[4].
d. **Third Element**: Added at q[4], rear moves to q[5].
e. **Fourth Element**: Added at q[5], rear moves to q[6].
f. **Fifth Element**: Added at q[6], rear moves to q[7].
g. **Sixth Element**: Added at q[7], rear moves to q[8].
h. **Seventh Element**: Added at q[8], rear moves to q[9].
i. **Eighth Element**: Added at q[9], rear moves to q[10].
j. **Ninth Element**: Added at q[10], rear moves to q[0] due to circular nature of queue



6. Write a C Program to implement Red Black Tree

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    int color;
    struct Node *left, *right, *parent;
};

struct Node* createNode(int data)
 {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->color = RED; // New nodes are always red initially
    newNode->left = newNode->right = newNode->parent = NULL;
    return newNode;
}

void leftRotate(struct Node **root, struct Node *x)
{
    struct 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(struct Node **root, struct Node *y)
{
    struct 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(struct Node **root, struct Node *z)
{
    while (z != *root && z->parent->color == RED)
     {
       if (z->parent == z->parent->parent->left)
        {
          struct Node *y = z->parent->parent->right;
          if (y != NULL && y->color == RED)
           {
              z->parent->color = BLACK;
              y->color = BLACK;
              z->parent->parent->color = RED;
              z = z->parent->parent;
          }
        else
           {
```

```
        if (z == z->parent->right)
        {
            z = z->parent;
            leftRotate(root, z);
        }
        z->parent->color = BLACK;
        z->parent->parent->color = RED;
        rightRotate(root, z->parent->parent);
    }
 }
else
 {
    struct Node *y = z->parent->parent->left;
     if (y != NULL && y->color == RED)
     {
       z->parent->color = BLACK;
       y->color = BLACK;
       z->parent->parent->color = RED;
       z = z->parent->parent;
     }
    else
     {
        if (z == z->parent->left)
        {
          z = z->parent;
          rightRotate(root, z);
        }
       z->parent->color = BLACK;
       z->parent->parent->color = RED;
       leftRotate(root, z->parent->parent);
    }
```

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

void insert(struct Node **root, int data)
 {
    struct Node *z = createNode(data);
    struct Node *y = NULL;
    struct Node *x = *root;
    while (x != NULL)
 {
        y = x;
        if (z->data < x->data) x = x->left;
        else x = x->right;
    }
    z->parent = y;
    if (y == NULL) *root = z;
    else if (z->data < y->data) y->left = z;
    else y->right = z;
    fixViolation(root, z);
}
void inorderTraversal(struct Node *root)
 {
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

int main() {
```

```
    struct 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:\n");
    inorderTraversal(root);
    return 0;
}
```