ASSIGNMENT-1

ADVANCED DATA STRUCTURES

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?

Answer

From the given question we can understand that the program needs to store the score above 50 and the only possible scores that can be stored are from 51 to 100.Therefore the best way for the program 'P' to store frquencies would be using an array having size of 50.Because each index in the array can represent the score from 51 to 100 also the value at each index would be the frequency of that score.

Eg: Suppose we input some data ,

3 scores of 51 , 2 scores of 52 and like that no other scores higher than or equal to 53. Then the intialization of the array will be like -:

- Frequency[51]=3
- Frequency[52]=2
- Frequency[i]=0 for all ' i' from 53 to 100 here 'i' represent the index of the array

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?

Answer

Here the size of circular queue is 11 and the index range is from 0 to 10. So according to the question, both the rear and front are initialized at the index 2 ie q[2].

The steps involved in the enqueue operation to add and find the 9th element into the queue are -:

- Starting from q[2]
- Add 1st element then rear moves to q[3]
- Add 2nd element to q[3] then rear moves to q[4]
- Add 3rd element to q[4] then rear moves to q[5]
- Add 4th element to q[5] then rear moves to q[6]
- Add 5th element to q[6] then rear moves to q[7]
- Add 6th element to q[7] then rear moves to q[8]
- Add 7th element to q[8] then rear moves to q[9]
- Add 8th element to q[9] then rear moves to q[10]
- Add 9th element to q[10] then rear moves to q[0]

So here the 9th element is added at q[10]

6) Write a C Program to implement Red Black Tree

Answer

```
#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;
}
```