DATA STRUCTURE ASSIGNMENT

Submitted to:

Ms. Akshara Sasidharan

Submitted By:

Sandra S Krishna

S1 MCA

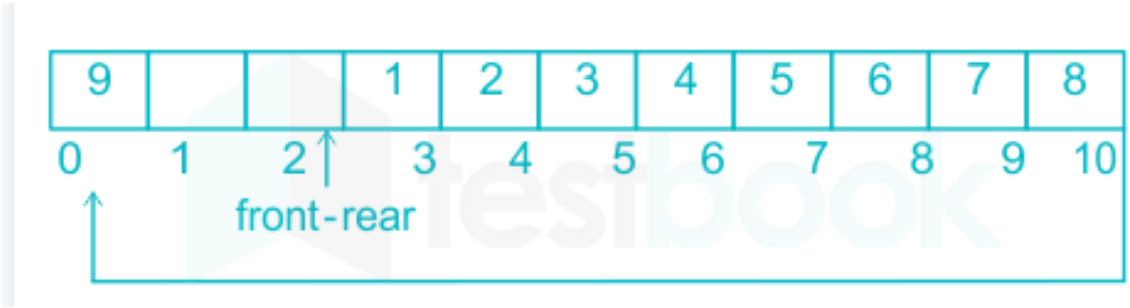
Roll No:54

1. 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?

   We can store the frequencies of score above 50 using an array. Because 0 to 100 means the score got by 500 students. So as we represent as an array we can access the score greater than 50.

2. Consider a standard circular queue 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 elements be added?

   Here the front and rear pointers are initialized to point at q[2]. So when we insert an new element to queue first increment REAR value with 1 that means in q[3] a new value 1 is added to it. Same way all the values upto 8 is added then the rear increment to q[0] because the space is vacant at the beginning. Then the value 9 is added to q[0].



3. Write a C program to implement red black tree.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int d;
```

```
int c;
struct node* p;
struct node* r;
struct node* l;
};
struct node* root = NULL;
struct node* bst(struct node* trav,struct node* temp)
{
if (trav == NULL)
return temp;
if (temp->d < trav->d)
{
trav->l = bst(trav->l, temp);
trav->l->p = trav;
}
else if (temp->d > trav->d)
{
trav->r = bst(trav->r, temp);
trav->r->p = trav;
}
return trav;
}
void rightrotate(struct node* temp)
{
struct node* left = temp->l;
temp->l = left->r;
if (temp->l)
temp->l->p = temp;
left->p = temp->p;
if (!temp->p)
root = left;
else if (temp == temp->p->l)
temp->p->l = left;
```

```
else
{
temp->p->r = left;
left->r = temp;
temp->p = left;
}
}
void leftrotate(struct node* temp)
{
struct node* right = temp->r;
temp->r = right->l;
if (temp->r)
temp->r->p = temp;
right->p = temp->p;
if (!temp->p)
root = right;
else if (temp == temp->p->l)
temp->p->l = right;
else
temp->p->r = right;
right->l = temp;
temp->p = right;
}
void fixup(struct node* root, struct node* pt)
{
struct node* parent_pt = NULL;
struct node* grand_parent_pt = NULL;
while ((pt != root) && (pt->c != 0)&& (pt->p->c == 1))
{
parent_pt = pt->p;
grand_parent_pt = pt->p->p;
if (parent_pt == grand_parent_pt->l)
{
```

```
struct node* uncle_pt = grand_parent_pt->r;
if (uncle_pt != NULL && uncle_pt->c == 1)
{
grand_parent_pt->c = 1;
parent_pt->c = 0;
uncle_pt->c = 0;
pt = grand_parent_pt;
}
else
{
if (pt == parent_pt->r)
{
leftrotate(parent_pt);
pt = parent_pt;
parent_pt = pt->p;
}
rightrotate(grand_parent_pt);
int t = parent_pt->c;
parent_pt->c = grand_parent_pt->c;
grand_parent_pt->c = t;
pt = parent_pt;
}
}
else
{
struct node* uncle_pt = grand_parent_pt->l;
if ((uncle_pt != NULL) && (uncle_pt->c == 1))
{
grand_parent_pt->c = 1;
parent_pt->c = 0;
uncle_pt->c = 0;
pt = grand_parent_pt;
}
```

```
else
{
if (pt == parent_pt->l)
{
rightrotate(parent_pt);
pt = parent_pt;
parent_pt = pt->p;
}
leftrotate(grand_parent_pt);
int t = parent_pt->c;
parent_pt->c = grand_parent_pt->c;
grand_parent_pt->c = t;
pt = parent_pt;
}
}
}
}
void inorder(struct node* trav)
{
if (trav == NULL)
return;
inorder(trav->l);
printf("%d ", trav->d);
inorder(trav->r);
}
int main()
{
int n = 7;
int a[7] = { 7, 6, 5, 4, 3, 2, 1 };
for (int i = 0; i < n; i++)
{
struct node  *temp= (struct node*)malloc(sizeof(struct node));
temp->r = NULL;
```

```
        temp->l = NULL;
        temp->p = NULL;
        temp->d = a[i];
        temp->c = 1;
        root = bst(root, temp);
        fixup(root, temp);
        root->c = 0;
        }
        printf("Inorder Traversal of Created Tree\n");
        inorder(root);
        return 0;
}
```