ASSIGNMENT II

ADVANCED DATA STRUCTURES

24th SEPTEMBER 2024

SUBMITTED TO

MS. AKSHARA SHASIDHARAN

DEPARTMENT OF COMPUTER

APPLICATION

SUBMITTED BY

ANGEL SHIBU

S1 MCA

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:

The best way to store frequencies of scores above 50 would be to use an array of size 51, indexing scores from 51 to 100. We need to store frequencies of scores above 50. We can ignore scores below 50 and to index the scores above 50, we can subtract 51 from the score value. 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 because we only need to store frequencies for scores 51-100.
2. **Read Scores:** Read 500 integers representing student scores. For each score.
3. **Frequency Update:** if score is greater than 50, increment the corresponding index in the frequency array i.e frequencies[score - 51].
4. **Print frequencies:** After processing all scores**,** iterate through the array and print score and frequency.

Benefits:

- Low Memory Usage
- Easy implementation and minimal code
- Efficient- Array indexing allows for constant-time access to frequency values (O(1)).

Alternatives:

- Hash table (e.g., std::unordered_map in C++): Flexible, but slower lookups (O(1) average, O(n) worst-case).
- Vector or list of pairs (score, frequency): Dynamic, but more memory-intensive.

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 areinitialized to point at q[2] . In which position will the ninth element be added?

Solution:

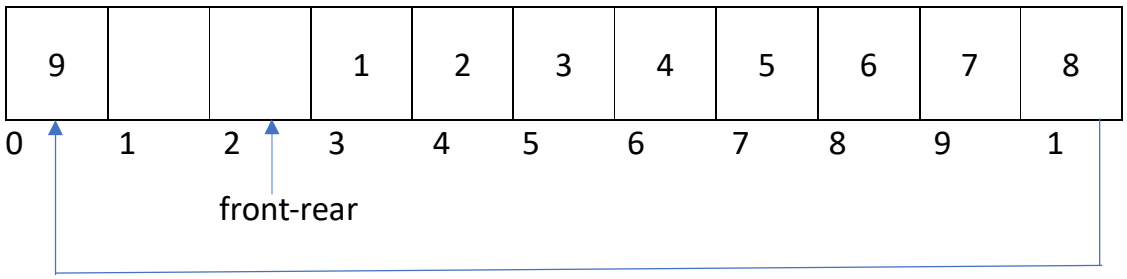
Initial State:

- 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 circular queue is not full, rear is incremented by one and the new element is inserted at the rear index. To remove an element from a non-empty circular queue front is incremented by one and the removes the element at front index.

Here, the front and rear pointers are both initialized at q[2]. The first element that will be inserted will be at index 2 (q[2]). As elements are inserted the front pointers will increment by one. 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)