Data Structures Assignment 1

Data Structures Assignment 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?

ANSWER

The best way for P to store the frequencies of the scores is by using an array (or list) of size 101, where each index corresponds to a possible score (from 0 to 100). The value at each index represents the frequency (or count) of how many times that score appears.

Approach:

1. Initialize an array (or list) of size 101, with all elements set to 0. This will hold the frequencies of scores where the index represents the score.
2. Read and process the scores: For each score, increment the value at the corresponding index in the array.
3. Print the frequencies of scores above 50: After processing all the scores, iterate over the array starting from index 51 and print the scores that have a non-zero frequency.

5) 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 element be added?

ANSWER

In a circular queue, the front and rear pointers wrap around when they reach the end of the queue array, based on the size of the queue. Here we are given the following information:

- The circular queue has a size of 11, so the elements are indexed from q[0] to q[10].
- Both the front and rear pointers are initially at q[2].
- We are adding the ninth element to the queue.

6)Implementation of RB tree

ANSWER

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

struct Node {
   int data;
   struct Node *parent, *left, *right;
   int color; // 0 for red, 1 for black
};

struct Node *root = NULL;



struct Node *newNode(int data) {
   struct Node *node = (struct Node *)malloc(sizeof(struct Node));
   node->data = data;
   node->parent = node->left = node->right = NULL;
   node->color = 0; // Red by default
   return node;
}
```

```
void leftRotate(struct Node *node) {
    struct Node *rightChild = node->right;
    node->right = rightChild->left;
    if (rightChild->left != NULL) {
        rightChild->left->parent = node;
    }
    rightChild->parent = node->parent;
    if (node->parent == NULL) {
        root = rightChild;
    } else if (node == node->parent->left) {
        node->parent->left = rightChild;
    } else {
        node->parent->right = rightChild;
    }
    rightChild->left = node;
    node->parent = rightChild;
}


void rightRotate(struct Node *node) {
    struct Node *leftChild = node->left;
    node->left = leftChild->right;
    if (leftChild->right != NULL) {
```

```
        leftChild->right->parent = node;
    }
    leftChild->parent = node->parent;
    if (node->parent == NULL) {
        root = leftChild;
    } else if (node == node->parent->left) {
        node->parent->left = leftChild;
    } else {
        node->parent->right = rightChild;
    }
    leftChild->right = node;
    node->parent = leftChild;
}


void fixInsertion(struct Node *node) {
    while (node->parent != NULL && node->parent->color == 0) {
        if (node->parent == node->parent->parent->left) {
            struct Node *uncle = node->parent->parent->right;
            if (uncle->color == 0) {
                uncle->color = 1;
                node->parent->color = 1;
                node->parent->parent->color = 0;
                node = node->parent->parent;
```

```
        } else {
            if (node == node->parent->right) {
                node = node->parent;
                leftRotate(node);
            }
            node->parent->color = 1;
            node->parent->parent->color = 0;
            rightRotate(node->parent->parent);
        }
    } else {
        // Symmetric case, handle right uncle
    }
  }
  root->color = 1;
}

void insertNode(struct Node *node) {
  struct Node *y = NULL;
  struct Node *x = root;
  while (x != NULL) {
    y = x;
    if (node->data < x->data) {
      x = x->left;
    } else {
```

```
            x = x->right;
        }
    }
    node->parent = y;
    if (y == NULL) {
        root = node;
    } else if (node->data < y->data) {
        y->left = node;
    } else {
        y->right = node;
    }
    node->left = node->right = NULL;
    node->color = 0;
    fixInsertion(node);
}


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

```
int main() {
    int numNodes;
    printf("Enter the number of nodes: ");
    scanf("%d", &numNodes);

    for (int i = 0; i < numNodes; i++) {
        int data;
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        insertNode(newNode(data));
    }

    printf("Inorder traversal of the red-black tree:\n");
    printInorder(root);

    return 0;
}
```

Done by: KIRAN KRISHNA

S1 MCA

ROLL NO : 41

To: Akshara Miss