**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**

* We have 500 integers representing student scores in the range 0 to 100
* Scores range from 51 to 100, there are 50 possible scores in this range.
* Says only score frequency above 50 need to be printed.
* Best way to store is to create an array of size 50 to store the frequency of the scores.
* For example, index 0 of the array can represent score 51, index 1 can represent score 52, and so on, up to index 49 representing score 100.
* By using an array of 50 numbers, the program can directly access the frequency for each score by its corresponding index values.

**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 standard cir -queue, if elements added from q[2], then 9th element added at position 10
* In a standard cir -queue, if elements added after q[2], then 9th element added at position 0

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: JUDE JOBY JOSEPH

S1 MCA

To: Akshara Miss